Bioinformatics Toolbox™ Reference


The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government’s needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.
<table>
<thead>
<tr>
<th>Date</th>
<th>Type</th>
<th>Revision Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 2005</td>
<td>Online only</td>
<td>New for Version 2.1 (Release 14SP2+)</td>
</tr>
<tr>
<td>September 2005</td>
<td>Online only</td>
<td>Revised for Version 2.1.1 (Release 14SP3)</td>
</tr>
<tr>
<td>November 2005</td>
<td>Online only</td>
<td>Revised for Version 2.2 (Release 14SP3+)</td>
</tr>
<tr>
<td>March 2006</td>
<td>Online only</td>
<td>Revised for Version 2.2.1 (Release 2006a)</td>
</tr>
<tr>
<td>May 2006</td>
<td>Online only</td>
<td>Revised for Version 2.3 (Release 2006a+)</td>
</tr>
<tr>
<td>September 2006</td>
<td>Online only</td>
<td>Revised for Version 2.4 (Release 2006b)</td>
</tr>
<tr>
<td>March 2007</td>
<td>Online only</td>
<td>Revised for Version 2.5 (Release 2007a)</td>
</tr>
<tr>
<td>April 2007</td>
<td>Online only</td>
<td>Revised for Version 2.6 (Release 2007a+)</td>
</tr>
<tr>
<td>September 2007</td>
<td>Online only</td>
<td>Revised for Version 3.0 (Release 2007b)</td>
</tr>
<tr>
<td>March 2008</td>
<td>Online only</td>
<td>Revised for Version 3.1 (Release 2008a)</td>
</tr>
<tr>
<td>October 2008</td>
<td>Online only</td>
<td>Revised for Version 3.2 (Release 2008b)</td>
</tr>
<tr>
<td>March 2009</td>
<td>Online only</td>
<td>Revised for Version 3.3 (Release 2009a)</td>
</tr>
<tr>
<td>September 2009</td>
<td>Online only</td>
<td>Revised for Version 3.4 (Release 2009b)</td>
</tr>
<tr>
<td>March 2010</td>
<td>Online only</td>
<td>Revised for Version 3.5 (Release 2010a)</td>
</tr>
<tr>
<td>September 2010</td>
<td>Online only</td>
<td>Revised for Version 3.6 (Release 2010b)</td>
</tr>
<tr>
<td>April 2011</td>
<td>Online only</td>
<td>Revised for Version 3.7 (Release 2011a)</td>
</tr>
<tr>
<td>September 2011</td>
<td>Online only</td>
<td>Revised for Version 4.0 (Release 2011b)</td>
</tr>
<tr>
<td>March 2012</td>
<td>Online only</td>
<td>Revised for Version 4.1 (Release 2012a)</td>
</tr>
<tr>
<td>September 2012</td>
<td>Online only</td>
<td>Revised for Version 4.2 (Release 2012b)</td>
</tr>
<tr>
<td>March 2013</td>
<td>Online only</td>
<td>Revised for Version 4.3 (Release 2013a)</td>
</tr>
<tr>
<td>September 2013</td>
<td>Online only</td>
<td>Revised for Version 4.3.1 (Release 2013b)</td>
</tr>
<tr>
<td>March 2014</td>
<td>Online only</td>
<td>Revised for Version 4.4 (Release 2014a)</td>
</tr>
<tr>
<td>October 2014</td>
<td>Online only</td>
<td>Revised for Version 4.5 (Release 2014b)</td>
</tr>
<tr>
<td>March 2015</td>
<td>Online only</td>
<td>Revised for Version 4.5.1 (Release 2015a)</td>
</tr>
<tr>
<td>September 2015</td>
<td>Online only</td>
<td>Revised for Version 4.5.2 (Release 2015b)</td>
</tr>
<tr>
<td>March 2016</td>
<td>Online only</td>
<td>Revised for Version 4.6 (Release 2016a)</td>
</tr>
<tr>
<td>September 2016</td>
<td>Online only</td>
<td>Revised for Version 4.7 (Release 2016b)</td>
</tr>
<tr>
<td>March 2017</td>
<td>Online only</td>
<td>Revised for Version 4.8 (Release 2017a)</td>
</tr>
<tr>
<td>September 2017</td>
<td>Online only</td>
<td>Updated for Version 4.9 (Release 2017b)</td>
</tr>
<tr>
<td>March 2018</td>
<td>Online only</td>
<td>Updated for Version 4.10 (Release 2018a)</td>
</tr>
<tr>
<td>September 2018</td>
<td>Online only</td>
<td>Updated for Version 4.11 (Release 2018b)</td>
</tr>
<tr>
<td>March 2019</td>
<td>Online only</td>
<td>Updated for Version 4.12 (Release 2019a)</td>
</tr>
<tr>
<td>September 2019</td>
<td>Online only</td>
<td>Updated for Version 4.13 (Release 2019b)</td>
</tr>
<tr>
<td>March 2020</td>
<td>Online only</td>
<td>Updated for Version 4.14 (Release 2020a)</td>
</tr>
</tbody>
</table>
Functions and Apps
Functions and Apps
**aa2int**

Convert amino acid sequence from letter to integer representation

**Syntax**

\[ SeqInt = \text{aa2int}(SeqChar) \]

**Input Arguments**

<table>
<thead>
<tr>
<th>SeqChar</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string containing single-letter codes specifying an amino acid sequence. For valid letter codes, see the table Mapping Amino Acid Letter Codes to Integers. Unknown characters are mapped to 0. Integers are arbitrarily assigned to IUB/IUPAC letters.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB® structure containing a Sequence field that contains an amino acid sequence, such as returned by <code>fastaread</code>, <code>getgenpept</code>, <code>genpeptread</code>, <code>getpdb</code>, or <code>pdbread</code>.</td>
</tr>
</tbody>
</table>

**Output Arguments**

| SeqInt | Amino acid sequence specified by a row vector of integers. |

**Description**

\[ SeqInt = \text{aa2int}(SeqChar) \] converts \( SeqChar \), a character vector or string containing single-letter codes specifying an amino acid sequence, to \( SeqInt \), a row vector of integers specifying the same amino acid sequence. For valid letter codes, see the table Mapping Amino Acid Letter Codes to Integers.
### Mapping Amino Acid Letter Codes to Integers

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>Code</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alanine</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>Arginine</td>
<td>R</td>
<td>2</td>
</tr>
<tr>
<td>Asparagine</td>
<td>N</td>
<td>3</td>
</tr>
<tr>
<td>Aspartic acid (Aspartate)</td>
<td>D</td>
<td>4</td>
</tr>
<tr>
<td>Cysteine</td>
<td>C</td>
<td>5</td>
</tr>
<tr>
<td>Glutamine</td>
<td>Q</td>
<td>6</td>
</tr>
<tr>
<td>Glutamic acid (Glutamate)</td>
<td>E</td>
<td>7</td>
</tr>
<tr>
<td>Glycine</td>
<td>G</td>
<td>8</td>
</tr>
<tr>
<td>Histidine</td>
<td>H</td>
<td>9</td>
</tr>
<tr>
<td>Isoleucine</td>
<td>I</td>
<td>10</td>
</tr>
<tr>
<td>Leucine</td>
<td>L</td>
<td>11</td>
</tr>
<tr>
<td>Lysine</td>
<td>K</td>
<td>12</td>
</tr>
<tr>
<td>Methionine</td>
<td>M</td>
<td>13</td>
</tr>
<tr>
<td>Phenylalanine</td>
<td>F</td>
<td>14</td>
</tr>
<tr>
<td>Proline</td>
<td>P</td>
<td>15</td>
</tr>
<tr>
<td>Serine</td>
<td>S</td>
<td>16</td>
</tr>
<tr>
<td>Threonine</td>
<td>T</td>
<td>17</td>
</tr>
<tr>
<td>Tryptophan</td>
<td>W</td>
<td>18</td>
</tr>
<tr>
<td>Tyrosine</td>
<td>Y</td>
<td>19</td>
</tr>
<tr>
<td>Valine</td>
<td>V</td>
<td>20</td>
</tr>
<tr>
<td>Asparagine or Aspartic acid (Aspartate)</td>
<td>B</td>
<td>21</td>
</tr>
<tr>
<td>Glutamine or Glutamic acid (Glutamate)</td>
<td>Z</td>
<td>22</td>
</tr>
<tr>
<td>Unknown amino acid (any amino acid)</td>
<td>X</td>
<td>23</td>
</tr>
<tr>
<td>Translation stop</td>
<td>*</td>
<td>24</td>
</tr>
<tr>
<td>Gap of indeterminate length</td>
<td>-</td>
<td>25</td>
</tr>
<tr>
<td>Unknown character (any character or symbol not in table)</td>
<td>?</td>
<td>0</td>
</tr>
</tbody>
</table>

### Examples

**Convert an amino acid sequence to integer representation**

Create a random amino acid sequence.

```matlab
seq = randseq(20, 'alphabet', 'amino')
seq =
'TYNYMRQLVVDVVITNHYSV'
```

Convert the sequence from letter to integer representation.
seqInt = aa2int(seq)
seqInt = 1x20 uint8 row vector
    17   19    3   19   13    2    6   11   20   20    4   20   20   10   17    3    9   19   16

See Also
aminolookup | int2aa | int2nt | nt2int

Introduced before R2006a
**getStop**

**Class:** BioMap

Compute stop positions of aligned read sequences from BioMap object

**Syntax**

```matlab
Stop = getStop(BioObj)
Stop = getStop(BioObj, Subset)
```

**Description**

`Stop = getStop(BioObj)` returns `Stop`, a vector of integers specifying the stop position of aligned read sequences with respect to the position numbers in the reference sequence from a BioMap object.

`Stop = getStop(BioObj, Subset)` returns a stop position for only read sequences specified by `Subset`.

**Input Arguments**

**BioObj**

Object of the BioMap class.

**Default:**

**Subset**

One of the following to specify a subset of the elements in `BioObj`:

- Vector of positive integers
- Logical vector
- Cell array of character vectors containing valid sequence headers

**Note** If you use a cell array of headers to specify `Subset`, be aware that a repeated header specifies all elements with that header.

**Default:**

**Output Arguments**

**Stop**

Vector of integers specifying the stop position of aligned read sequences with respect to the position numbers in the reference sequence. `Stop` includes the stop positions for only read sequences specified by `Subset`. 
Examples

Construct a BioMap object, and then compute the stop position for different sequences in the object:

% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Compute the stop position of the second sequence in the object
Stop_2 = getStop(BMObj1, 2)
Stop_2 =
  37
% Compute the stop positions of the first and third sequences in
% the object
Stop_1_3 = getStop(BMObj1, [1 3])
Stop_1_3 =
  36
  39
% Compute the stop positions of all sequences in the object
Stop_All = getStop(BMObj1);

See Also
BioMap | getStart

Topics
“Manage Sequence Read Data in Objects”

External Websites
Sequence Read Archive
SAM format specification
filterByFlag

Class: BioMap

Filter sequence reads by SAM flag

Syntax

Indices = filterByFlag(BioObj, FlagName, FlagValue)
Indices = filterByFlag(BioObj, Subset, FlagName, FlagValue)
Indices = filterByFlag(..., FlagName1, FlagValue1, FlagName2, FlagValue2, ...)

Description

Indices = filterByFlag(BioObj, FlagName, FlagValue) returns Indices, a vector of logical indices, indicating the read sequences in BioObj, a BioMap object, with FlagName set to FlagValue.

Indices = filterByFlag(BioObj, Subset, FlagName, FlagValue) returns Indices, a vector of logical indices, indicating the read sequences that meet the specified criteria from a subset of entries in a BioMap object.

Indices = filterByFlag(..., FlagName1, FlagValue1, FlagName2, FlagValue2, ...) applies multiple flag filters in a single statement.

Input Arguments

BioObj

Object of the BioMap class.

Default:

Subset

Either of the following to specify a subset of the elements in BioObj:

- Vector of positive integers
- Logical vector

Default:

FlagName

Character vector or string specifying one of the following flags to filter by:

- 'pairedInSeq' — The read is paired in sequencing, regardless if it is mapped as a pair.
- 'pairedInMap' — The read is mapped in a proper pair.
- 'unmappedQuery' — The read is unmapped.
• 'unmappedMate' — The mate is unmapped.
• 'strandQuery' — Strand direction of the read (0 = forward, 1 = reverse).
• 'strandMate' — Strand direction of the mate (0 = forward, 1 = reverse).
• 'readIsFirst' — The read is first in a pair.
• 'readIsSecond' — The read is second in a pair.
• 'alnNotPrimary' — The read's alignment is not primary.
• 'failedQualCheck' — The read fails platform or vendor quality checks.
• 'duplicate' — The read is a PCR or optical duplicate.

Default:

FlagValue

Logical value indicating the status of a flag. A 0 indicates false or forward, and a 1 indicates true or reverse.

Default:

Output Arguments

Indices

Vector of logical indices, indicating the read sequences in BioObj with FlagName set to FlagValue.

Examples

Construct a BioMap object, and then determine the read sequences that are both mapped in a proper pair and first in a pair:

% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Filter the elements using 'pairedInMap' and 'readIsFirst' flags
Indices = filterByFlag(BMObj1, 'pairedInMap', true,...
'readIsFirst', true);
% Return the headers of the filtered elements
filtered_Headers = BMObj1.Header(Indices);

See Also

BioMap

Topics

"Manage Sequence Read Data in Objects"

External Websites

Sequence Read Archive
SAM format specification
**getAlignment**

**Class:** BioMap

Construct alignment represented in BioMap object

**Syntax**

\[
\text{Alignment} = \text{getAlignment}(\text{BioObj}, \text{StartPos}, \text{EndPos})
\]

\[
\text{Alignment} = \text{getAlignment}(\text{BioObj}, \text{StartPos}, \text{EndPos}, R)
\]

\[
\text{Alignment} = \text{getAlignment}(..., '\text{ParameterName}', \text{ParameterValue})
\]

\[
[\text{Alignment}, \text{Indices}] = \text{getAlignment}(...)
\]

**Description**

\[
\text{Alignment} = \text{getAlignment}(\text{BioObj}, \text{StartPos}, \text{EndPos})\]

returns \text{Alignment}, a character array containing the aligned read sequences from \text{BioObj}, a BioMap object. The read sequences must align within a specific region of the reference sequence, which is defined by \text{StartPos} and \text{EndPos}, two positive integers such that \text{StartPos} is less than \text{EndPos}, and both are smaller than the length of the reference sequence.

\[
\text{Alignment} = \text{getAlignment}(\text{BioObj}, \text{StartPos}, \text{EndPos}, R)
\]

selects the reference where getAlignment reconstructs the alignment.

\[
\text{Alignment} = \text{getAlignment}(..., '\text{ParameterName}', \text{ParameterValue})\]

accepts one or more comma-separated parameter name/value pairs. Specify \text{ParameterName} inside single quotes.

\[
[\text{Alignment}, \text{Indices}] = \text{getAlignment}(...)
\]

returns \text{Indices}, a vector of indices specifying the read sequences that align within a specific region of the reference sequence.

**Input Arguments**

- **BioObj**
  
  Object of the BioMap class.

  **Default:**

  - **StartPos**
    
    Positive integer that defines the start of a region of the reference sequence. \text{StartPos} must be less than \text{EndPos}, and smaller than the total length of the reference sequence.

    **Default:**

  - **EndPos**
    
    Positive integer that defines the end of a region of the reference sequence. \text{EndPos} must be greater than \text{StartPos}, and smaller than the total length of the reference sequence.

    **Default:**
R

Positive integer indexing the SequenceDictionary property of `BioObj`, or a character vector or string specifying the actual name of the reference.

**Parameter Name/Value Pairs**

**OffsetPad**

Specifies if padding blanks are added at the beginning of each aligned sequence to represent the offset of the start position of each aligned sequence with respect to the reference. Choices are `true` or `false` (default).

**Default:**

**Output Arguments**

**Alignment**

Character array containing the aligned read sequences from `BioObj` that align within a specific region of the reference sequence. Each row of the character array contains one aligned sequence, that is, the sequence positions that fall within the specified region of the reference sequence. Each aligned sequence can include gaps.

**Indices**

Vector of indices specifying the read sequences from `BioObj` that align within a specific region of the reference sequence.

**Examples**

Construct a `BioMap` object, and then reconstruct the alignment between positions 10 and 25 of the reference sequence:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Construct the alignment between positions 10 and 25 of the reference sequence.
Alignment = getAlignment(BMObj1, 10, 25)
```

```
Alignment =

CTCATTTGAAATGTGT
CTCATTTGAAATGTGT
CTCATTTGAAATGTGT
CTCATTTGAAATTTTTT
CTCATTTGAAATGTGT
ATGTAAATGTGT
ATGTAAATGTGT
TGTAATGTGT
AAATGTGT
GTGT
GTGT
GT
```
**Algorithms**

`getAlignment` assumes the reference sequence has no gaps. Therefore, positions in reads corresponding to insertions (I) and padding (P) do not appear in the alignment.

Because soft clipped positions (S) are not associated with positions that align to the reference sequence, they do not appear in the alignment.

A skipped position (N) appears as a . (period) in the alignment.

Hard clipped positions (H) do not appear in the sequences or the alignment.

**See Also**
BioMap | align2cigar | cigar2align | getBaseCoverage | getCompactAlignment

**Topics**
“Manage Sequence Read Data in Objects“

**External Websites**
Sequence Read Archive
SAM format specification
**getBaseCoverage**

**Class:** BioMap

Return base-by-base alignment coverage of reference sequence in `BioMap` object

**Syntax**

```matlab
Cov = getBaseCoverage(BioObj, StartPos, EndPos)
Cov = getBaseCoverage(BioObj, StartPos, EndPos, R)
Cov = getBaseCoverage(..., Name,Value)
[Cov, BinStart] = getBaseCoverage(...)```

**Description**

`Cov = getBaseCoverage(BioObj, StartPos, EndPos)` returns `Cov`, a row vector of nonnegative integers. This vector indicates the base-by-base alignment coverage of a range or set of ranges in the reference sequence in `BioObj`, a `BioMap` object. The range or set of ranges are defined by `StartPos` and `EndPos`. `StartPos` and `EndPos` can be two nonnegative integers such that `StartPos` is less than `EndPos`, and both integers are smaller than the length of the reference sequence. `StartPos` and `EndPos` can also be two column vectors representing a set of ranges (overlapping or segmented). When `StartPos` and `EndPos` specify a segmented range, `Cov` contains NaN values for base positions between segments.

`Cov = getBaseCoverage(BioObj, StartPos, EndPos, R)` selects the reference where `getBaseCoverage` calculates the coverage.

`Cov = getBaseCoverage(..., Name,Value)` returns alignment coverage information with additional options specified by one or more `Name,Value` pair arguments.

`[Cov, BinStart] = getBaseCoverage(...)` returns `BinStart`, a row vector of positive integers specifying the start position of each bin (when binning occurs).

**Input Arguments**

**BioObj**

Object of the `BioMap` class.

Default:

**StartPos**

Either of the following:

- Nonnegative integer that defines the start of a range in the reference sequence. `StartPos` must be less than `EndPos` and smaller than the total length of the reference sequence.
- Column vector of nonnegative integers, each defining the start of a range in the reference sequence.

Default:
EndPos

Either of the following:

• Nonnegative integer that defines the end of a range in the reference sequence. EndPos must be greater than StartPos and smaller than the total length of the reference sequence.

• Column vector of nonnegative integers, each defining the end of a range in the reference sequence.

R

Positive integer indexing the SequenceDictionary property of BioObj, or a character vector or string specifying the actual name of the reference.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

binWidth

Positive integer specifying the bin width, in number of base pairs (bp). Bins are centered within min(StartPos) and max(EndPos). Thus, the first and last bins span approximately equally outside the range from min(StartPos) to max (EndPos).

Note You cannot specify both binWidth and numberOfBins.

Default:

numberOfBins

Positive integer specifying the number of equal-width bins to use to span the requested region. Bins are centered within min(StartPos) and max(EndPos). Thus, the first and last bins span approximately equally outside the range from min(StartPos) to max (EndPos).

Note You cannot specify both binWidth and numberOfBins.

Default:

binType

Character vector or string specifying the binning algorithm. Choices are:

• 'max' — From the bin, getBaseCoverage selects the base position with the most reads aligned to it, then uses its alignment coverage value for the bin.

• 'min' — From the bin, getBaseCoverage selects the base position with the least reads aligned to it, then uses its alignment coverage value for the bin.

• 'mean' — Uses the average alignment coverage, computed from all base positions within the bin.

Default: 'max'
complementRanges

Specifies whether to return the alignment coverage for the base positions between segments, instead of within segments. If true, the length of Cov is numel(min(StartPos):max(EndPos)), and Cov contains NaN values for base positions within segments.

**Default:** false

Spliced

Logical specifying whether short reads are spliced during mapping (as in mRNA-to-genome mapping). N symbols in the Signature property of the object are not counted.

**Default:** false

**Output Arguments**

Cov

Row vector of nonnegative integers. This vector specifies the number of read sequences that align with each base position or bin in the requested regions. A set of ranges can be overlapping or segmented. For a range, the length of Cov is numel(StartPos:EndPos). For a segmented range, the length of Cov is numel(min(StartPos):max(EndPos)). Cov contains NaN values for base positions between segments. When binning occurs, the number of elements in Cov equals the number of bins.

BinStart

Row vector of positive integers specifying the start position of each bin. BinStart is the same length as Cov. If no binning occurs, then BinStart equals min(StartPos):max(EndPos).

**Examples**

Construct a BioMap object, and then return the alignment coverage of each of the first 12 base positions of the reference sequence:

```matlab
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Return the number of reads that align to each of
% the first 12 base positions of the reference sequence
cov = getBaseCoverage(BMObj1, 1, 12)
cov =
   1     1     2     2     3     4     4     4     5     5     5     5
```

Construct a BioMap object, and then return the alignment coverage of the range between 1 and 1000, on a bin-by-bin basis, using bins with a width of 100 bp:

```matlab
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Return the number of reads that align to each 100-bp bin
% in the 1:1000 range of the reference sequence. Also return the
% start position of each bin
[cov, bin_starts] = getBaseCoverage(BMObj1, 1, 1000, 'binWidth', 100)
```
cov =
    17  20  41  44  45  48  48  45  46  42

bin_starts =
    1  101  201  301  401  501  601  701  801  901

See Also
BioMap | align2cigar | cigar2align | getAlignment | getCompactAlignment | getCounts | getIndex

Topics
“Manage Sequence Read Data in Objects”

External Websites
Sequence Read Archive
SAM format specification
getCompactAlignment

Class: BioMap

Construct compact alignment represented in BioMap object

Syntax

\[
\text{CompAlignment} = \text{getCompactAlignment}(\text{BioObj}, \text{StartPos}, \text{EndPos})
\]

\[
\text{CompAlignment} = \text{getCompactAlignment}(\text{BioObj}, \text{StartPos}, \text{EndPos}, \text{R})
\]

\[
\text{CompAlignment} = \text{getCompactAlignment}(\text{..., 'ParameterName'}, \text{ParameterValue})
\]

\[
[\text{CompAlignment}, \text{Indices}] = \text{getCompactAlignment}(\text{...})
\]

\[
[\text{CompAlignment}, \text{Indices}, \text{Rows}] = \text{getCompactAlignment}(\text{...})
\]

Description

\[
\text{CompAlignment} = \text{getCompactAlignment}(\text{BioObj}, \text{StartPos}, \text{EndPos})
\]

returns \(\text{CompAlignment}\), a character array containing the aligned read sequences from \(\text{BioObj}\), a BioMap object, in a compact format. The read sequences must align within a specific region of the reference sequence, which is defined by \(\text{StartPos}\) and \(\text{EndPos}\), two positive integers such that \(\text{StartPos}\) is less than \(\text{EndPos}\), and both are smaller than the length of the reference sequence.

\[
\text{CompAlignment} = \text{getCompactAlignment}(\text{BioObj}, \text{StartPos}, \text{EndPos}, \text{R})
\]

selects the reference where \text{getCompactAlignment} reconstructs the alignment.

\[
\text{CompAlignment} = \text{getCompactAlignment}(\text{..., 'ParameterName'}, \text{ParameterValue})
\]

accepts one or more comma-separated parameter name/value pairs. Specify \text{ParameterName} inside single quotes.

\[
[\text{CompAlignment}, \text{Indices}] = \text{getCompactAlignment}(\text{...})
\]

returns \(\text{Indices}\), a vector of indices specifying the read sequences that align within a specific region of the reference sequence.

\[
[\text{CompAlignment}, \text{Indices}, \text{Rows}] = \text{getCompactAlignment}(\text{...})
\]

returns \(\text{Rows}\), a vector of positive numbers specifying the row in \(\text{CompAlignment}\) where each read sequence is best displayed.

Input Arguments

\text{BioObj}

Object of the BioMap class.

Default:

\text{StartPos}

Positive integer that defines the start of a region of the reference sequence. \text{StartPos} must be less than \text{EndPos}, and smaller than the total length of the reference sequence.

Default:
**EndPos**

Positive integer that defines the end of a region of the reference sequence. *EndPos* must be greater than *StartPos*, and smaller than the total length of the reference sequence.

**Default:**

*R*

Positive integer indexing the `SequenceDictionary` property of `BioObj`, or a character vector or string specifying the actual name of the reference.

**Parameter Name/Value Pairs**

**Full**

Specifies whether or not to include only the read sequences that fully align with the defined region of the reference sequence, that is, they are completely contained within the region, and do not extend beyond the region. Choices are `true` or `false` (default).

**Default:** `false`

**TrimAlignment**

Specifies whether or not to trim empty leading and trailing columns from the alignment. Choices are `true` or `false`. Default is `false`, which does not trim the alignment, but includes any empty leading or trailing columns, and returns an alignment always of length $EndPos - StartPos + 1$.

**Default:** `false`

**Output Arguments**

**CompAlignment**

Character array containing the aligned read sequences from `BioObj` that align within the requested region. The character array represents a compact alignment, that is each row of the character array contains one or more aligned sequences, such that the number of rows in the character array is minimized. Each aligned sequence includes only the sequence positions that fall within the requested region, and each aligned sequence can include gaps.

**Indices**

Vector of indices specifying the read sequences from `BioObj` that align within the requested region.

**Rows**

Vector of positive numbers specifying the row in `CompAlignment` where each read sequence is best displayed.

**Examples**

Construct a `BioMap` object, and then construct the compact alignment between positions 30 and 59 of the reference sequence:
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Construct the compact alignment between positions 30 and 59 of
% the reference sequence, and return the indices of the reads in the
% compact alignment, as well as the row each read is in.
[CompAlignment, Ind, Row] = getCompactAlignment(BMObj1, 30, 59)

CompAlignment =
TAACTCG      GCCCAGCATTAGGGAGC
TAACTCGT     CATTAGGGAGC
TAACTCGTCC   ATTAGGGAGC
TAACTCGTCTCT TTAGGGAGC
TAACTCGTCCATGG TAGGGAGC
TAACTCGTCCCTGGCCA C
TAACTCGTCCATGGCCCAG
TAACTCGTCCATTGGCCAGC
TAACTCGTCCATGGCCCAGC
TAACTCGTCCATGGCCCAGCATT
TAACTCGTCCATGGCCCAGCATTAGGG
TAACTCGTCCATGGCCCAGCATTAGGGAGC
TAACTCGTCCATGGCCCAGCATTAGGGATC
TAACTCGTCCATGGCCCAGCATTAGGGAGC
AACTCGTCCATGGCCCAGCATTAGGGAGC
GTACATGGCCCAGCATTAGGGAGC
TCCATGGCCCAGCATTAGGGCG

Ind =
   1
   2
   3
   4
   5
   6
   7
   8
   9
  10
  11
  12
  13
  14
  15
  16
  17
  18
  19
  20
  21
  22
  23

Row =
   1
Algorithms

getCompactAlignment assumes the reference sequence has no gaps. Therefore, positions in reads corresponding to insertions (I) and padding (P) do not appear in the alignment.

Because soft clipped positions (S) are not associated with positions that align to the reference sequence, they do not appear in the alignment.

A skipped position (N) appears as a - (hyphen) in the alignment.

Hard clipped positions (H) do not appear in the sequences or the alignment.

See Also
BioMap | align2cigar | cigar2align | getAlignment | getBaseCoverage

Topics
“Manage Sequence Read Data in Objects”

External Websites
Sequence Read Archive
SAM format specification
getCounts

Class: BioMap

Return count of read sequences aligned to reference sequence in BioMap object

Syntax

Count = getCounts(BioObj, StartPos, EndPos)
GroupCount = getCounts(BioObj, StartPos, EndPos, Groups)
GroupCount = getCounts(BioObj, StartPos, EndPos, Groups, R)
___ = getCounts(___, Name,Value)

Description

Count = getCounts(BioObj, StartPos, EndPos) returns Count, a nonnegative integer specifying the number of read sequences in BioObj, a BioMap object, that align to a specific range or set of ranges in the reference sequence. The range or set of ranges are defined by StartPos and EndPos. StartPos and EndPos can be two nonnegative integers such that StartPos is less than EndPos, and both integers are smaller than the length of the reference sequence. StartPos and EndPos can also be two column vectors representing a set of ranges (overlapping or segmented).

By default, getCounts counts each read only once. Therefore, if a read spans multiple ranges, that read instance is counted only once. When StartPos and EndPos specify overlapping ranges, the overlapping ranges are considered as one range.

GroupCount = getCounts(BioObj, StartPos, EndPos, Groups) specifies Groups, a vector of integers or cell array of character vectors or string vector, indicating groups that segmented ranges belong to. The segmented ranges are treated independently.

GroupCount = getCounts(BioObj, StartPos, EndPos, Groups, R) specifies a reference for each of the segmented ranges defined by StartPos, EndPos, and Groups.

___ = getCounts(___, Name,Value) uses additional options specified by one or more Name,Value pair arguments.

Input Arguments

BioObj

Object of the BioMap class.

Default:

StartPos

Either of the following:

• Nonnegative integer that defines the start of a range in the reference sequence. StartPos must be less than EndPos, and smaller than the total length of the reference sequence.
• Column vector of nonnegative integers, each defining the start of a range in the reference sequence.

**Default:**

**EndPos**

Either of the following:

• Nonnegative integer that defines the end of a range in the reference sequence. *EndPos* must be greater than *StartPos*, and smaller than the total length of the reference sequence.

• Column vector of nonnegative integers, each defining the end of a range in the reference sequence.

**Default:**

**Groups**

Row vector of integers, cell array of character vectors, or string vector of the same size as *StartPos* and *EndPos*. This vector indicates the group to which each range belongs.

**R**

Vector of positive integers indexing the `SequenceDictionary` property of `BioObj`, or a cell array of character vectors or string vector of the reference names. *R* must be scalar or must have the same number of elements as *Groups*.

For a given value of *Groups*, all the corresponding elements in *R* must be the same.

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Independent**

Logical that specifies whether to treat the ranges defined by *StartPos* and *EndPos* independently. If `true`, *Count* is a column vector containing the same number of elements as *StartPos* and *EndPos*. In this case, a read that spans multiple ranges, is counted once in each range.

**Note** This name-value pair argument is ignored when using the *Groups* input argument, because `getCounts` assumes that each group of ranges is independent.

**Default:** `false`

**Overlap**

Specifies the minimum number of base positions that a read must overlap in a range or set of ranges, to be counted. This value can be any of the following:

• Positive integer

• ‘full’ — A read must be fully contained in a range or set of ranges to be counted.
• ‘start’ — A read’s start position must lie within a range or set of ranges to be counted.

**Default:** 1

**Spliced**

Logical specifying whether short reads are spliced during mapping (as in mRNA-to-genome mapping). N symbols in the **Signature** property of the object are not counted.

**Default:** false

**Method**

Character vector or string specifying the method to measure the abundance of reads. Choices are:

• ‘raw’ — Raw counts
• ‘rpkm’ — Counts of reads per kilobase pairs per million aligned reads
• ‘mean’ — Average coverage depth computed base-by-base
• ‘max’ — Maximum coverage depth computed base-by-base
• ‘min’ — Minimum coverage depth computed base-by-base
• ‘sum’ — Sum of all aligned bases in all the reads

**Default:** ‘raw’

**Output Arguments**

**Count**

Either of the following:

• When **Independent** is false, this value is a nonnegative integer. The integer specifies the number of reads that align to a range or set of ranges (overlapping or segmented) of the reference sequence in **BioObj**, a **BioMap** object. Each read is counted only once, even if the read spans multiple ranges.

• When **Independent** is true, this value is a vector of nonnegative integers. This vector indicates the number of reads that align to the independent ranges specified by **StartPos** and **EndPos**. This vector contains the same number of elements as **StartPos** and **EndPos**.

**GroupCount**

Either of the following:

• If no reference or a single reference is specified, this value is a vector containing the number of reads for each unique group in **Groups**. The order of elements in **GroupsCount** corresponds to the ascending order of unique elements in **Groups**.

• If multiple references are specified, **GroupCount** is a cell array, where the ith element contains the number of reads for each unique group in the ith reference. The order of elements in **GroupsCount** corresponds to the ascending order of unique elements in **R**.

**Examples**
Compute the number of reads mapped to regions of reference sequence

Create a BioMap object.

```
obj = BioMap('ex1.sam');
```

Return the number of reads that cover at least one base of the segmented range 1:50 and 71:100. By default, the ranges are not treated independently, that is, a read is counted once even if it maps to both segmented ranges.

```
counts_1 = getCounts(obj,[1;71],[50;100])
counts_1 = 37
```

Compute the number of reads, treating the segmented ranges [1:50] and [71:100] independently. Observe that \( \text{sum}(\text{counts}_2) \) is greater than \( \text{counts}_1 \) because there are four reads that span over the two segments and are counted twice in the second case.

```
counts_2 = getCounts(obj,[1;71],[50;100], 'Independent', true)
counts_2 = 2×1
    20
    21
```

Compute the number of reads that align to the segmented range 30:60 (associated with group 1) and the segmented range [1:10 50:60] (associated with group 2).

```
counts_3 = getCounts(obj,[1;30;50],[10;60;60],[2 1 2])
counts_3 = 2×1
    25
    22
```

Return the total number of reads aligned to the reference sequence.

```
getCounts(obj, min(getStart(obj)), max(getStop(obj)))
ans = 1482
```

See Also

BioMap | align2cigar | cigar2align | featurecount | getAlignment | getBaseCoverage | getCompactAlignment | getIndex

Topics

“Manage Sequence Read Data in Objects”

External Websites

Sequence Read Archive
SAM format specification
getCoverage

Class: BioMap

Compute read coverage in BioMap object

Note getCoverage has been removed. Use getBaseCoverage, getCounts, or getIndex instead.

Syntax

Cov = getCoverage(BioObj, StartPos, EndPos)
[Cov, Indices] = getCoverage(BioObj, StartPos, EndPos)
[Cov, Indices, Seqs] = getCoverage(BioObj, StartPos, EndPos)
... = getCoverage(BioObj, StartPos, EndPos, 'ParameterName', ParameterValue)

Description

Cov = getCoverage(BioObj, StartPos, EndPos) returns Cov, a nonnegative integer indicating the number of read sequences that cover (align within) a specific region of the reference sequence in BioObj, a BioMap object. The specific region of the reference sequence is defined by StartPos and EndPos. StartPos and EndPos can be two nonnegative integers such that StartPos is less than EndPos, and both are smaller than the length of the reference sequence. StartPos and EndPos can also be two column vectors representing a collection of regions of the reference sequence. In this case, Cov is a column vector of nonnegative integers indicating the number of read sequences that cover each region.

[Cov, Indices] = getCoverage(BioObj, StartPos, EndPos) also returns Indices, a vector of indices specifying the read sequences that align within a specific region of the reference sequence.

[Cov, Indices, Seqs] = getCoverage(BioObj, StartPos, EndPos) also returns Seqs, a cell array of strings containing the read sequences that align within a specific region of the reference sequence.

... = getCoverage(BioObj, StartPos, EndPos, 'ParameterName', ParameterValue) accepts one or more comma-separated parameter name/value pairs. Specify ParameterName inside single quotes.

Input Arguments

BioObj

Object of the BioMap class.

Default:
StartPos

Either of the following:
• Nonnegative integer that defines the start of a region of the reference sequence. \textit{StartPos} must be less than \textit{EndPos}, and smaller than the total length of the reference sequence.
• Column vector of nonnegative integers, each defining the start of a region of the reference sequence.

Default:

\textbf{EndPos}

Either of the following:
• Nonnegative integer that defines the end of a region of the reference sequence. \textit{EndPos} must be greater than \textit{StartPos}, and smaller than the total length of the reference sequence.
• Column vector of nonnegative integers, each defining the end of a region of the reference sequence.

Default:

\textbf{Parameter Name/Value Pairs}

\textbf{Base}

Specifies if the output \textit{Cov} is computed base-by-base, that is determining the number of nongap symbols that align with each position in the specified region of the reference sequence. If \texttt{true}, \textit{Cov} is a vector of positive integers corresponding to the base positions in the specified region of the reference sequence.

Default: \texttt{false}

\textbf{Full}

Specifies to include only the read sequences that fully align with the defined region of the reference sequence, that is, they are completely contained within the region, and do not extend beyond the region.

Default: \texttt{false}

\textbf{Output Arguments}

\textbf{Cov}

Either of the following:
• Nonnegative integer indicating the number of read sequences that cover (align within) a specific region of the reference sequence in \textit{BioObj}.
• Column vector of nonnegative integers indicating the number of read sequences that cover each region specified by \textit{StartPos} and \textit{EndPos}, when they are both column vectors. In this case, \textit{Cov} is the same length as \textit{StartPos} and \textit{EndPos}.

\textbf{Indices}

Vector of indices specifying the read sequences from \textit{BioObj} that align within a specific region of the reference sequence.
Seqs

Cell array of strings containing the read sequences from BioObj that align within a specific region of the reference sequence. Each string is a sequence read without alignment information.

Examples

Construct a BioMap object, and then retrieve the coverage of the first 50 positions of the reference sequence:

```matlab
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the number of sequences that cover the first 50
% positions of the reference sequence
cov = getCoverage(BMObj1, 1, 50)
cov = 20
```

Construct a BioMap object, and then retrieve the starting positions for the read sequences that cover the first 50 positions of the reference sequence:

```matlab
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the number of sequences that cover the first 50
% positions of the reference sequence
% Also retrieve the indices of these sequences
[cov, idx] = getCoverage(BMObj1, 1, 50);
% Use the indices for these sequences to determine their start
% positions
startPositions = getStart(BMObj1, idx);
```

Construct a BioMap object, and then retrieve the coverage of the first 50 positions of the reference sequence, considering only read sequences that align fully within the region:

```matlab
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the number of sequences that cover the first 50
% positions of the reference sequence
% Consider only read sequences that align fully within the region
fullCov = getCoverage(BMObj1, 1, 50, 'full', true)
fullCov = 8
```

Construct a BioMap object, and then retrieve the coverage for the first 10 positions of the reference sequence, on a base-by-base basis:

```matlab
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the number of sequences that cover each base position of
% the first 10 positions of the reference sequence
baseCov = getCoverage(BMObj1, 1, 10, 'base', true)
baseCov =
```
**Tips**

Use the *Indices* output from the `getCoverage` method as input to other `BioMap` methods. Doing so lets you determine other information about the read sequences in the coverage region, such as header, start position, mapping quality, etc.

**See Also**

`BioMap` | `align2cigar` | `cigar2align` | `getAlignment` | `getCompactAlignment`

**Topics**

“Manage Sequence Read Data in Objects”

**External Websites**

Sequence Read Archive
SAM format specification
getFlag

Class: BioMap

Retrieve read sequence flags from BioMap object

Syntax

Flag = getFlag(BioObj)
Flag = getFlag(BioObj, Subset)

Description

Flag = getFlag(BioObj) returns Flag, a vector of nonnegative integers indicating the bit-wise information that specifies the status of the 11 flags described by the SAM format specification. Each integer corresponds to one read sequence from a BioMap object.

Flag = getFlag(BioObj, Subset) returns flag integers for only object elements specified by Subset.

Input Arguments

BioObj

Object of the BioMap class.

Default:

Subset

One of the following to specify a subset of the elements in BioObj:

• Vector of positive integers
• Logical vector
• Cell array of character vectors or string vector containing valid sequence headers

Note If you use a cell array of headers to specify Subset, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

Flag

Vector of nonnegative integers. Each integer corresponds to one read sequence and indicates the bit-wise information that specifies the status of the 11 flags described by the SAM format specification. These flags describe different sequencing and alignment aspects of a read sequence. Flag includes flag integers for only read sequences specified by Subset.
Examples

Construct a BioMap object, and then retrieve the SAM flag values for different elements in the object:

```matlab
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve integer specifying bit-wise information for 11
% SAM flags of the second element
flagValue = getFlag(BMObj1, 2)

flagValue =
    73

% Retrieve integers specifying bit-wise information for 11
% SAM flags of the first and third elements
flagValues = getFlag(BMObj1, [1 3])

flagValues =
    73
    137

% Retrieve integers specifying bit-wise information for 11
% SAM flags of all elements
allFlagValues = getFlag(BMObj1);

% Determine the status of the fourth flag (mate is unmapped)
% for the second element, which has a flag value of 73
bitget(73, 4)

ans =
    1
```

Tips

After using the `getFlag` method to return the integer specifying the bit-wise information for the SAM flags, use the `bitget` function to determine the status of a specific SAM flag. For more information, see “Examples” on page 1-29.

Alternatives

An alternative to using the `getFlag` method is to use dot indexing with the `Flag` property:

```
BioObj.Flag(Indices)
```

In the previous syntax, `Indices` is a vector of positive integers or a logical vector. `Indices` cannot be a cell array of character vectors or string vector containing sequence headers.

See Also

BioMap | bitget | setFlag

Topics

“Manage Sequence Read Data in Objects”
External Websites
Sequence Read Archive
SAM format specification
**getIndex**

Class: BioMap

Return indices of read sequences aligned to reference sequence in BioMap object

**Syntax**

\[
\text{Indices} = \text{getIndex}(\text{BioObj}, \text{StartPos}, \text{EndPos}) \\
\text{Indices} = \text{getIndex}(\text{BioObj}, \text{StartPos}, \text{EndPos}, R) \\
\text{Indices} = \text{getIndex}(\ldots, \text{Name}, \text{Value})
\]

**Description**

\[
\text{Indices} = \text{getIndex}(\text{BioObj}, \text{StartPos}, \text{EndPos}) \text{ returns } \text{Indices}, \text{ a column vector of indices specifying the read sequences that align to a range or set of ranges in the reference sequence in } \text{BioObj}, \text{ a BioMap object. The range or set of ranges are defined by } \text{StartPos} \text{ and } \text{EndPos}. \text{ StartPos and EndPos can be two nonnegative integers such that StartPos is less than EndPos, and both integers are smaller than the length of the reference sequence. StartPos and EndPos can also be two column vectors representing a set of ranges (overlapping or segmented).}
\]

getIndex includes each read only once. Therefore, if a read spans multiple ranges, the index for that read appears only once.

\[
\text{Indices} = \text{getIndex}(\text{BioObj}, \text{StartPos}, \text{EndPos}, R) \text{ selects the reference associated with the range specified by StartPos and EndPos.}
\]

\[
\text{Indices} = \text{getIndex}(\ldots, \text{Name}, \text{Value}) \text{ returns indices with additional options specified by one or more Name,Value pair arguments.}
\]

**Input Arguments**

**BioObj**

Object of the BioMap class.

**Default:**

**StartPos**

Either of the following:

- Nonnegative integer that defines the start of a range in the reference sequence. StartPos must be less than EndPos, and smaller than the total length of the reference sequence.
- Column vector of nonnegative integers, each defining the start of a range in the reference sequence.

**Default:**

**EndPos**

Either of the following:
• Nonnegative integer that defines the end of a range in the reference sequence. \textit{EndPos} must be greater than \textit{StartPos}, and smaller than the total length of the reference sequence.
• Column vector of nonnegative integers, each defining the end of a range in the reference sequence.

\textbf{Default:}

\textbf{R}

Positive integer indexing the \texttt{SequenceDictionary} property of \textit{BioObj}, or a character vector or string specifying the actual name of the reference.

\textbf{Name-Value Pair Arguments}

Specify optional comma-separated pairs of \texttt{Name}, \texttt{Value} arguments. \texttt{Name} is the argument name and \texttt{Value} is the corresponding value. \texttt{Name} must appear inside quotes. You can specify several name and value pair arguments in any order as \texttt{Name1,Value1,...,NameN,ValueN}.

\textbf{Overlap}

Specifies the minimum number of base positions that a read must overlap in a range or set of ranges, to be included. This value can be any of the following:
• Positive integer
• 'full' — A read must be fully contained in a range or set of ranges to be counted.
• 'start' — A read's start position must lie within a range or set of ranges to be counted.

\textbf{Default: 1}

\textbf{Depth}

Specifies to decimate the output indices. The coverage depth at any base position is less than or equal to \texttt{Depth}, a positive integer.

\textbf{Default: Inf}

\textbf{Spliced}

Logical specifying whether short reads are spliced during mapping (as in mRNA-to-genome mapping). \texttt{N} symbols in the \texttt{Signature} property of the object are not counted.

\textbf{Default: false}

\textbf{Output Arguments}

\textbf{Indices}

Column vector of indices specifying the reads that align to a range or set of ranges in the specified reference sequence in \textit{BioObj}, a \texttt{BioMap} object.

\textbf{Examples}

Construct a \texttt{BioMap} object, and then use the indices of the reads to retrieve the start and stop positions for the reads that are fully contained in the first 50 positions of the reference sequence:
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Return the indices of reads that are fully contained in the
% first 50 positions of the reference sequence
indices = getIndex(BMObj1, 1, 50, 'overlap', 'full');
% Use these indices to return the start and stop positions of
% the reads
starts = getStart(BMObj1, indices)
stops = getStop(BMObj1, indices)

starts =
    1
    3
    5
    6
    9
   13
   13
   15

stops =
    36
    37
    39
    41
    43
    47
    48
    49

Construct a BioMap object, and then use the indices of the reads to retrieve the sequences for the
reads whose alignments overlap a segmented range by at least one base pair:

% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Return the indices of the reads that overlap the
% segmented range 98:100 and 198:200, by at least 1 base pair
indices = getIndex(BMObj1, [98;198], [100;200], 'overlap', 1);
% Use these indices to return the sequences of the reads
sequences = getSequence(BMObj1, indices);

**Tips**

Use the *Indices* output from the `getIndex` method as input to other BioMap methods. Doing so lets
you retrieve other information about the reads in the range, such as header, start position, mapping
quality, sequences, etc.

**See Also**

BioMap | align2cigar | cigar2align | getAlignment | getBaseCoverage |
getcCompactAlignment | getCounts | getSequence | getStart | getStop

**Topics**

“Manage Sequence Read Data in Objects”
External Websites
Sequence Read Archive
SAM format specification
getInfo

Class: BioMap

Retrieve information for single element of BioMap object

Syntax

\[ Info = \text{getInfo}(BioObj, \ Element) \]

Description

\[ Info = \text{getInfo}(BioObj, \ Element) \] returns \( Info \), a tab-delimited character vector containing information about a single element in \( BioObj \), a BioMap object.

Input Arguments

BioObj

Object of the BioMap class.

Default:

Element

One of the following to specify one element in \( BioObj \):

- Scalar specifying an element index
- Logical vector
- Character vector containing a valid sequence header

Default:

Output Arguments

Info

Tab-delimited character vector containing information about a single element in \( BioObj \), a BioMap object. The character vector contains the information from the following properties in order:

- Header
- Flag
- Start
- MappingQuality
- Signature
- Sequence
- Quality
Examples

Construct a BioMap object, and then retrieve information for the second element in the object:

```matlab
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve information for the second element in the object
element2Info = getInfo(BMObj1, 2)

element2Info =
EAS54_65:7:152:368:113 73 3 99 35M
CTAGTGCTCATTGTAAATGTGTGGTTTAACTCGT
<<<<<<<<0<<<<655<<<<7<<<<3<<<<6):```

See Also
BioMap

Topics
“Manage Sequence Read Data in Objects”

External Websites
Sequence Read Archive
SAM format specification
getMappingQuality

Class: BioMap

Retrieve sequence mapping quality scores from BioMap object

Syntax

MappingQuality = getMappingQuality(BioObj)
MappingQuality = getMappingQuality(BioObj, Subset)

Description

MappingQuality = getMappingQuality(BioObj) returns MappingQuality, a vector of integers specifying mapping quality scores for each read sequence in BioObj, a BioMap object.

MappingQuality = getMappingQuality(BioObj, Subset) returns mapping quality scores for only object elements specified by Subset.

Input Arguments

BioObj

Object of the BioMap class.

Default:

Subset

One of the following to specify a subset of the elements in BioObj:

• Vector of positive integers
• Logical vector
• Cell array of character vectors or string vector containing valid sequence headers

Note If you use a cell array of headers to specify Subset, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

MappingQuality

MappingQuality property of a subset of elements in BioObj. MappingQuality is a vector of integers specifying the mapping quality scores for read sequences specified by Subset.
Examples

Construct a BioMap object, and then retrieve the mapping quality scores for different elements in the object:

% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the mapping quality property of the second element in
% the object
MQ_2 = getMappingQuality(BMObj1, 2)

MQ_2 =
    99

% Retrieve the mapping quality properties of the first and third
% elements in the object
MQ_1_3 = getMappingQuality(BMObj1, [1 3])

MQ_1_3 =
    99
    99

% Retrieve the mapping quality properties of all elements in the
% object
MQ_All = getMappingQuality(BMObj1);

Alternatives

An alternative to using the getMappingQuality method is to use dot indexing with the MappingQuality property:

BioObj.MappingQuality(Indices)

In the previous syntax, Indices is a vector of positive integers or a logical vector. Indices cannot be a cell array of character vectors or string vector containing sequence headers.

See Also
BioMap | setMappingQuality

Topics
“Manage Sequence Read Data in Objects”

External Websites
Sequence Read Archive
SAM format specification
getMatePosition

Class: BioMap

Retrieve mate positions of read sequences from BioMap object

Syntax

MatePos = getMatePosition(BioObj)
MatePos = getMatePosition(BioObj,Subset)

Description

MatePos = getMatePosition(BioObj) returns MatePos, a vector of nonnegative integers specifying the mate positions of read sequences with respect to the position numbers in the reference sequence from a BioMap object.

MatePos = getMatePosition(BioObj,Subset) returns mate positions for only read sequences specified by Subset.

Input Arguments

BioObj
Object of the BioMap class.

Default:

Subset
One of the following to specify a subset of the elements in BioObj:

• Vector of positive integers
• Logical vector
• Cell array of character vectors or string vector containing valid sequence headers

Note If you use a cell array of headers to specify Subset, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

MatePos
MatePosition property of all or a subset of elements in BioObj. MatePos is a vector of nonnegative integers specifying the mate positions of read sequences with respect to the position numbers in the reference sequence. MatePos includes the mate positions for only read sequences specified by Subset.
Not all values in the `MatePosition` vector represent valid mate positions, for example, mates that map to a different reference sequence or mates that do not map. To determine if a mate position is valid, use the `filterByFlag` method with the 'pairedInMap' flag.

**Examples**

Construct a `BioMap` object, and then retrieve the mate position for different sequences in the object:

```matlab
% Construct a BioMap object from a SAM file and determine the header for the 17th element
BMObj1 = BioMap('ex1.sam');
BMObj1.Header(17)
ans =
    'EAS114_32:5:78:583:499'

% Retrieve the MatePosition property of the 17th element in the object using the header
MatePos_17 = getMatePosition(BMObj1,{'EAS114_32:5:78:583:499'})
MatePos_17 =
    229
    37

Notice the previous example returned two mate positions. This is because the header 'EAS114_32:5:78:583:499' is a repeated header in the `BMObj1` object. The `getMatePosition` method returns mate positions for all elements in the object with that header.

% Retrieve the MatePosition properties of the 37th and 47th elements in
% the object
MatePos_37_47 = getMatePosition(BMObj1, [37 47])
MatePos_37_47 =
    95
    283

% Retrieve the MatePosition properties of all elements in the object
MatePos_All = getMatePosition(BMObj1);
```

**Alternatives**

An alternative to using the `getMatePosition` method is to use dot indexing with the `MatePosition` property:

```matlab
BioObj.MatePosition(Indices)
```

In the previous syntax, `Indices` is a vector of positive integers or a logical vector. `Indices` cannot be a cell array of character vectors or string vector containing sequence headers.

**See Also**

`BioMap | filterByFlag | setMatePosition`

**Topics**

“Manage Sequence Read Data in Objects”
External Websites
Sequence Read Archive
SAM format specification
**getReference**

**Class:** BioMap

Retrieve reference sequence from BioMap object

**Syntax**

\[ \text{Ref} = \text{getReference}(\text{BioObj}) \]

**Description**

\[ \text{Ref} = \text{getReference}(\text{BioObj}) \] returns the name of the reference sequence from a BioMap object. This is the Reference property of the object.

**Input Arguments**

**BioObj**

Object of the BioRead or BioMap class.

**Default:**

**Output Arguments**

**Ref**

Reference property of BioObj, the BioMap object. It is a character vector or string vector specifying the name of the reference sequence.

**Examples**

Construct a BioMap object, and then retrieve the reference sequence from the object:

```matlab
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the reference sequence from the object
refSeq = getReference(BMObj1)
```

**Alternatives**

An alternative to using the getReference method is to use dot indexing with the Reference property:

\[ BioObj.\text{Reference} \]
See Also
BioMap | setReference

Topics
“Manage Sequence Read Data in Objects”

External Websites
Sequence Read Archive
SAM format specification
getSignature

Class: BioMap

Retrieve signature (alignment information) from BioMap object

Syntax

Signature = getSignature(BioObj)
Signature = getSignature(BioObj, Subset)

Description

Signature = getSignature(BioObj) returns Signature, a cell array of CIGAR-formatted strings, each representing how a read sequence in a BioMap object aligns to the reference sequence.

Signature = getSignature(BioObj, Subset) returns signature strings for only object elements specified by Subset.

Input Arguments

BioObj

Object of the BioMap class.

Default:

Subset

One of the following to specify a subset of the elements in BioObj:

- Vector of positive integers
- Logical vector
- Cell array of character vectors or string vector containing valid sequence headers

Note If you use a cell array of headers to specify Subset, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

Signature

Signature property of a subset of elements in BioObj. Signature is a cell array of CIGAR-formatted strings, each representing how read sequences, specified by Subset, align to the reference sequence.
Examples

Construct a BioMap object, and then retrieve the signatures for different elements in the object:

```matlab
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the signature property of the second element in
% the object
Sig_2 = getSignature(BMObj1, 2)

Sig_2 =
    '35M'

% Retrieve the signature properties of the first and third
% elements in the object
Sig_1_3 = getSignature(BMObj1, [1 3])

Sig_1_3 =
    '36M'
    '35M'

% Retrieve the signature properties of all elements in the object
Sig_All = getSignature(BMObj1);
```

Alternatives

An alternative to using the `getSignature` method is to use dot indexing with the `Signature` property:

```
BioObj.Sgnature(Indices)
```

In the previous syntax, `Indices` is a vector of positive integers or a logical vector. `Indices` cannot be a cell array of character vectors or string vector containing sequence headers.

See Also

`BioMap` | `getAlignment` | `setSignature`

Topics

"Manage Sequence Read Data in Objects"

External Websites

Sequence Read Archive
SAM format specification
**getStart**

*Class:* BioMap

Retrieve start positions of aligned read sequences from BioMap object

**Syntax**

\[
\text{Start } = \text{getStart}(\text{BioObj}) \\
\text{Start } = \text{getStart}(\text{BioObj}, \text{Subset})
\]

**Description**

\(\text{Start } = \text{getStart}(\text{BioObj})\) returns \(\text{Start}\), a vector of integers specifying the start position of aligned read sequences with respect to the position numbers in the reference sequence from a BioMap object.

\(\text{Start } = \text{getStart}(\text{BioObj}, \text{Subset})\) returns a start position for only read sequences specified by \(\text{Subset}\).

**Input Arguments**

**BioObj**

Object of the BioMap class.

**Default:**

**Subset**

One of the following to specify a subset of the elements in \(\text{BioObj}\):

- Vector of positive integers
- Logical vector
- Cell array of character vectors or string vector containing valid sequence headers

**Note** If you use a cell array of headers to specify \(\text{Subset}\), be aware that a repeated header specifies all elements with that header.

**Default:**

**Output Arguments**

**Start**

\(\text{Start}\) property of a subset of elements in \(\text{BioObj}\). It is a vector of integers specifying the start position of aligned read sequences with respect to the position numbers in the reference sequence. It includes the start positions for only read sequences specified by \(\text{Subset}\).
Examples

Construct a BioMap object, and then retrieve the start position for different sequences in the object:

% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the start property of the second element in the object
Start_2 = getStart(BMObj1, 2)
Start_2 =
    3
% Retrieve the start properties of the first and third elements in
% the object
Start_1_3 = getStart(BMObj1, [1 3])
Start_1_3 =
    1
    5
% Retrieve the start properties of all elements in the object
Start_All = getStart(BMObj1);

Alternatives

An alternative to using the getStart method is to use dot indexing with the Start property:

BioObj.Start(Indices)

In the previous syntax, Indices is a vector of positive integers or a logical vector. Indices cannot be a cell array of character vectors or string vector containing sequence headers.

See Also

BioMap | getStop | setStart

Topics

“Manage Sequence Read Data in Objects”

External Websites

Sequence Read Archive
SAM format specification
**getSummary**

*Class:* BioMap

Print summary of BioMap object

**Syntax**

```matlab
getSummary(BioObj)
ds = getSummary(BioObj)
```

**Description**

`getSummary(BioObj)` prints a summary of a BioMap object. The summary includes the names of references, the number of sequences mapped to each reference, and the genomic range that the sequences cover in each reference.

`ds = getSummary(BioObj)` returns the summary information in a dataset array.

**Input Arguments**

BioObj

Object of the BioMap class.

**Output Arguments**

ds

dataset array containing the summary of the BioMap object, BioObj. The dataset array has an observation (row) for each reference in BioObj, and two variables (columns): the number of sequences mapped to each reference and the genomic range that the sequences cover in each reference.

`getSummary` stores additional metadata for the BioMap object in the UserData property of ds, which you can access using `ds.Properties.UserData`.

**Examples**

Construct a BioMap object, and then display a summary of the object:

```matlab
% Construct a BioMap object from a SAM file
BMObj2 = BioMap('ex2.sam');
getSummary(BMObj2)

BioMap summary:
    Name: '
    Container_Type: 'Data is file indexed.'
    Total_Number_of_Sequences: 3307
    Number_of_References_in_Dictionary: 2
```
<table>
<thead>
<tr>
<th></th>
<th>Number_of_Sequences</th>
<th>Genomic_Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq1</td>
<td>1501</td>
<td>1  1569</td>
</tr>
<tr>
<td>seq2</td>
<td>1806</td>
<td>1  1567</td>
</tr>
</tbody>
</table>

See Also
BioMap

Topics
“Manage Sequence Read Data in Objects”

External Websites
Sequence Read Archive
SAM format specification
setFlag

Class: BioMap
Set read sequence flags for BioMap object

Syntax

NewObj = setFlag(BioObj, FlagValues)
NewObj = setFlag(BioObj, FlagValues, Subset)

Description

NewObj = setFlag(BioObj, FlagValues) returns NewObj, a new BioMap object, constructed from BioObj, an existing BioMap object, with the Flag property set to FlagValues, a vector of nonnegative integers indicating the bit-wise information that specifies the status of each of the 11 flags described by the SAM format specification.

NewObj = setFlag(BioObj, FlagValues, Subset) sets the Flag property of the elements specified by Subset to FlagValues.

Input Arguments

BioObj — Object of BioMap class
BioMap object

Object of the BioMap class, specified as a BioMap object

Note If BioObj was constructed from a BioIndexedFile object, you cannot set its Flag property.

FlagValues — SAM flag values
vector of nonnegative integers

SAM flag values, specified as vector of nonnegative integers. Each integer corresponds to one read sequence and indicates the bit-wise information that specifies the status of each of the 11 flags described by the SAM format specification. These flags describe different sequencing and alignment aspects of a read sequence.

Subset — Subset of elements in BioObj
vector of positive integers | logical vector | cell array of character vectors

Subset of elements in BioObj, specified as one of the following:

• vector of positive integers
• logical vector
• cell array of character vectors containing valid sequence headers
**Note** A one-to-one relationship must exist between the number and order of elements in `FlagValues` and `Subset`. If you use a cell array of headers to specify `Subset`, be aware that a repeated header specifies all elements with that header.

### Output Arguments

- **NewObj** — Object of BioMap class  
  BioMap object

Object of the BioMap class, returned as a `BioMap` object.

### Examples

**Update SAM flag values of BioMap object**

Create a BioMap object from a SAM file. Set `'InMemory'` to true to allow modifying the object properties.

```matlab
bm = BioMap('ex2.sam','InMemory',true);
```

Check the SAM flag value of the 5th read sequence.

```matlab
bm.Flag(5)
ans = uint16
    137
```

Update the flag value.

```matlab
bm2 = setFlag(bm,75,5);
bm2.Flag(5)
ans = uint16
    75
```

Update the flag values of the first and third elements.

```matlab
bm3 = setFlag(bm,[0 0],[1 3]);
bm3.Flag(1)
ans = uint16
    0
```

```matlab
bm3.Flag(3)
ans = uint16
    0
```

### Alternatives

An alternative to using the `setFlag` method to update an existing object is to use dot indexing with the `Flag` property:
In the previous syntax, `Indices` is a vector of positive integers or a logical vector. `Indices` cannot be a cell array of character vectors containing sequence headers. `NewFlag` is a vector of nonnegative integers indicating the bit-wise information that specifies the status of each of the 11 flags described by the SAM format specification. Each integer corresponds to one read sequence in a `BioMap` object. `Indices` and `NewFlag` must have the same number and order of elements.

**See Also**
`BioMap` | `getFlag`

**Topics**
"Manage Sequence Read Data in Objects"

**External Websites**
Sequence Read Archive
SAM format specification
setMappingQuality

Class: BioMap

Set sequence mapping quality scores for BioMap object

Syntax

NewObj = setMappingQuality(BioObj, MappingQuality)
NewObj = setMappingQuality(BioObj, MappingQuality, Subset)

Description

NewObj = setMappingQuality(BioObj, MappingQuality) returns NewObj, a new BioMap object, constructed from BioObj, an existing BioMap object, with the MappingQuality property set to MappingQuality, a vector of integers specifying the mapping quality scores for read sequences.

NewObj = setMappingQuality(BioObj, MappingQuality, Subset) returns NewObj, a new BioMap object, constructed from BioObj, an existing BioMap object, with the MappingQuality property of a subset of the elements set to MappingQuality, a vector of integers specifying the mapping quality scores for read sequences. It sets the mapping quality scores for only the object elements specified by Subset.

Input Arguments

BioObj
Object of the BioMap class.

Note If BioObj was constructed from a BioIndexedFile object, you cannot set its MappingQuality property.

Default:

MappingQuality
Vector of integers specifying the mapping quality scores for read sequences.

Default:

Subset
One of the following to specify a subset of the elements in BioObj:

- Vector of positive integers
- Logical vector
- Cell array of character vectors containing valid sequence headers
A one-to-one relationship must exist between the number and order of elements in `MappingQuality` and `Subset`. If you use a cell array of headers to specify `Subset`, be aware that a repeated header specifies all elements with that header.

**Default:**

**Output Arguments**

- **NewObj**: Object of the BioMap class.

**Examples**

Construct a BioMap object, and then set a subset of the mapping quality scores:

```matlab
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Set the Mapping Quality property of the second element to a new value
BMObj1 = setMappingQuality(BMObj1, 74, 2);
```

**Tips**

To update mapping quality scores in an existing BioMap object, use the same object as the input `BioObj` and the output `NewObj`.

**Alternatives**

An alternative to using the `setMappingQuality` method to update an existing object is to use dot indexing with the `MappingQuality` property:

```matlab
BioObj.MappingQuality(Indices) = NewMappingQuality
```

In the previous syntax, `Indices` is a vector of positive integers or a logical vector. `Indices` cannot be a cell array of character vectors containing sequence headers. `NewMappingQuality` is a vector of integers specifying the mapping quality scores for read sequences. `Indices` and `NewQuality` must have the same number and order of elements.

**See Also**

- BioMap | `getMappingQuality`

**Topics**

- “Manage Sequence Read Data in Objects”

**External Websites**

- Sequence Read Archive
- SAM format specification
setMatePosition

Class: BioMap

Set mate positions of read sequences in BioMap object

Syntax

NewObj = setMatePosition(BioObj,MatePos)
NewObj = setMatePosition(BioObj,MatePos,Subset)

Description

NewObj = setMatePosition(BioObj,MatePos) returns NewObj, a new BioMap object, constructed from BioObj, an existing BioMap object, with the MatePosition property set to MatePos, a vector of nonnegative integers specifying the mate positions of the read sequences with respect to the position numbers in the reference sequence.

NewObj = setMatePosition(BioObj,MatePos,Subset) returns NewObj, a new BioMap object, constructed from BioObj, an existing BioMap object, with the MatePosition property of a subset of the elements set to MatePos, a vector of nonnegative integers specifying the mate positions of the read sequences with respect to the position numbers in the reference sequence. The setMatePosition method sets the mate positions for only the object elements specified by Subset.

Input Arguments

BioObj

Object of the BioMap class.

Note  If BioObj was constructed from a BioIndexedFile object, you cannot set its MatePosition property.

Default:

MatePos

Vector of nonnegative integers specifying the mate positions of the read sequences with respect to the position numbers in the reference sequence.

Default:

Subset

One of the following to specify a subset of the elements in BioObj:

- Vector of positive integers
- Logical vector
• Cell array of character vectors containing valid sequence headers

**Note** A one-to-one relationship must exist between the number and order of elements in `MatePos` and `Subset`. If you use a cell array of headers to specify `Subset`, be aware that a repeated header specifies all elements with that header.

**Default:**

**Output Arguments**

`NewObj`

Object of the `BioMap` class.

**Examples**

Construct a `BioMap` object, and then set a subset of the sequence mate position values:

```matlab
% Construct a BioMap object from a SAM file and determine the header for the second element
BMObj1 = BioMap('ex1.sam');
BMObj1.Header(2)
ans =
    'EAS54_65:7:152:368:113'
% Set the MatePosition property of the second element to a new value of 5
BMObj1 = setMatePosition(BMObj1, 5, {'EAS54_65:7:152:368:113'});
% Set the MatePosition properties of the first and third elements in
% the object to 6 and 7 respectively
BMObj1 = setMatePosition(BMObj1, [6 7], [1 3]);
% Set the MatePosition property of all elements in the object to zero
y = zeros(1,BMObj1.NSeqs);
BMObj1 = setMatePosition(BMObj1,y);
```

**Tips**

• To update mate positions in an existing `BioMap` object, use the same object as the input `BioObj` and the output `NewObj`.

**Alternatives**

An alternative to using the `setMatePosition` method to update an existing object is to use dot indexing with the `MatePosition` property:

```matlab
BioObj.MatePosition(Indices) = NewMatePos
```

In the previous syntax, `Indices` is a vector of positive integers or a logical vector. `Indices` cannot be a cell array of character vectors containing sequence headers. `NewMatePos` is a vector of integers specifying the mate positions of the read sequences with respect to the position numbers in the reference sequence. `Indices` and `NewMatePos` must have the same number and order of elements.
See Also
BioMap | getMatePosition

Topics
“Manage Sequence Read Data in Objects”

External Websites
Sequence Read Archive
SAM format specification
setReference

Class: BioMap

Set name of reference sequence for BioMap object

Syntax

newObj = setReference(Obj,RefSeqName)
newObj = setReference(Obj,RefSeqName,Indices)

Description

newObj = setReference(Obj,RefSeqName) returns a new BioMap object newObj, constructed from an existing object Obj, with the Reference property set to RefSeqName, a character vector or string specifying the name of the reference sequence.

newObj = setReference(Obj,RefSeqName,Indices) sets the Reference property of the elements indexed by Indices to RefSeqName.

Input Arguments

Obj — Object of BioMap class
BioMap object

Object of the BioMap class, specified as a BioMap object.

RefSeqName — Name of reference sequence
character vector | string | string vector | cell array of character vectors

Name of reference sequence, specified as a character vector, string, string vector, or cell array of character vectors.

Indices — Indices to elements of the input object
positive integer | vector of positive integers | logical array | character vector | string | string vector | cell array of character vectors

Indices to the elements of the input object, specified as a positive integer, vector of positive integers, logical array, character vector, string, string vector, or cell array of character vectors. For character vectors or strings, they must correspond to the valid Header values of the input object Obj.

Output Arguments

newObj — Object of BioMap class
BioMap object

Object of the BioMap class, returned as a BioMap object.

Examples
**Change reference sequence of BioMap object**

Create a BioMap object from a SAM file. Set 'InMemory' to true to allow modifying the object properties.

```matlab
bm = BioMap('ex2.sam','InMemory',true);
```

Check the reference sequence of the 5th read sequence.

```matlab
bm.Reference(5)
ans = 1x1 cell array
     {'seq1'}
```

Change the reference sequence to another one in the sequence dictionary.

```matlab
bm2 = setReference(bm,{'seq2'},5);
bm2.Reference(5)
ans = 1x1 cell array
     {'seq2'}
```

**See Also**

BioMap | getReference

**Topics**

“Manage Sequence Read Data in Objects”

**External Websites**

Sequence Read Archive  
SAM format specification
setSignature

Class: BioMap

Set signature (alignment information) for BioMap object

Syntax

NewObj = setSignature(BioObj, Signature)
NewObj = setSignature(BioObj, Signature, Subset)

Description

NewObj = setSignature(BioObj, Signature) returns NewObj, a new BioMap object, constructed from BioObj, an existing BioMap object, with the Signature property set to Signature, a cell array of CIGAR-formatted character vectors, each representing how a read sequence aligns to the reference sequence.

NewObj = setSignature(BioObj, Signature, Subset) returns NewObj, a new BioMap object, constructed from BioObj, an existing BioMap object, with the Signature property of a subset of the elements set to Signature, a cell array of CIGAR-formatted character vectors, each representing how read sequences, specified by Subset, align to the reference sequence. It sets the signature for only the object elements specified by Subset.

Input Arguments

BioObj

Object of the BioMap class.

Note If BioObj was constructed from a BioIndexedFile object, you cannot set its Signature property.

Default:

Signature

Cell array of CIGAR-formatted character vectors, each representing how a read sequence aligns to the reference sequence. Signature can be empty.

Default:

Subset

One of the following to specify a subset of the elements in BioObj:

- Vector of positive integers
- Logical vector
setSignature

- Cell array of character vectors containing valid sequence headers

**Note** A one-to-one relationship must exist between the number and order of elements in `Signature` and `Subset`. If you use a cell array of headers to specify `Subset`, be aware that a repeated header specifies all elements with that header.

**Default:**

**Output Arguments**

**NewObj**

Object of the `BioMap` class.

**Examples**

Construct a `BioMap` object, and then set a subset of the signatures:

```matlab
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Set the Signature property of the second element to a new value
BMObj1 = setSignature(BMObj1, {'36M'}, 2);
```

**Tips**

- To update signatures in an existing `BioMap` object, use the same object as the input `BioObj` and the output `NewObj`.
- If you modify sequences or start positions in an object, you may need to use the `setSignature` method to modify the `Signature` property of modified sequences accordingly.

**Alternatives**

An alternative to using the `setSignature` method to update an existing object is to use dot indexing with the `Signature` property:

```matlab
BioObj.Sig
```narmal

In the previous syntax, `Indices` is a vector of positive integers or a logical vector. `Indices` cannot be a cell array of character vectors containing sequence headers. `NewSignature` is a character vector or a cell array of CIGAR-formatted character vectors, each representing how a read sequence aligns to the reference sequence. Signature can be empty. `Indices` and `NewSignature` must have the same number and order of elements.

**See Also**

`BioMap` | `getAlignment` | `getSignature` | `setSequence` | `setStart`

**Topics**

"Manage Sequence Read Data in Objects"
External Websites
Sequence Read Archive
SAM format specification
setStart

Class: BioMap

Set start positions of aligned read sequences in BioMap object

Syntax

NewObj = setStart(BioObj, Start)
NewObj = setStart(BioObj, Start, Subset)

Description

NewObj = setStart(BioObj, Start) returns NewObj, a new BioMap object, constructed from BioObj, an existing BioMap object, with the Start property set to Start, a vector of positive integers specifying the start positions of the aligned read sequences with respect to the position numbers in the reference sequence. Modifying the Start property shifts the aligned sequences.

NewObj = setStart(BioObj, Start, Subset) returns NewObj, a new BioMap object, constructed from BioObj, an existing BioMap object, with the Start property of a subset of the elements set to Start, a vector of positive integers specifying the start positions of the aligned read sequences with respect to the position numbers in the reference sequence. It sets the start positions for only the object elements specified by Subset.

Input Arguments

BioObj

Object of the BioMap class.

Note If BioObj was constructed from a BioIndexedFile object, you cannot set its Start property.

Default:

Start

Vector of positive integers specifying the start positions of the aligned read sequences with respect to the position numbers in the reference sequence.

Default:

Subset

One of the following to specify a subset of the elements in BioObj:

• Vector of positive integers
• Logical vector
• Cell array of character vectors containing valid sequence headers
A one-to-one relationship must exist between the number and order of elements in `Start` and `Subset`. If you use a cell array of headers to specify `Subset`, be aware that a repeated header specifies all elements with that header.

Default:

**Output Arguments**

`NewObj`

Object of the `BioMap` class.

**Examples**

Construct a `BioMap` object, and then set a subset of the sequence start values:

```matlab
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Set the Start property of the second element to a new value
BMObj1 = setStart(BMObj1, 5, 2);
```

**Tips**

- To update start positions in an existing `BioMap` object, use the same object as the input `BioObj` and the output `NewObj`.
- If you modify sequences or signatures in an object, you may need to use the `setStart` method to modify the `Start` property to shift the alignment of modified sequences accordingly.

**Alternatives**

An alternative to using the `setStart` method to update an existing object is to use dot indexing with the `Start` property:

```matlab
BioObj.Start(Indices) = NewStart
```

In the previous syntax, `Indices` is a vector of positive integers or a logical vector. `Indices` cannot be a cell array of character vectors containing sequence headers. `NewStart` is a vector of integers specifying the start positions of the aligned read sequences with respect to the position numbers in the reference sequence. `Indices` and `NewStart` must have the same number and order of elements.

**See Also**

`BioMap` | `getStart` | `setSequence` | `setSignature`

**Topics**

“Manage Sequence Read Data in Objects”

**External Websites**

Sequence Read Archive

SAM format specification
BioMap class

Superclasses: BioRead

Contain sequence, quality, alignment, and mapping data

Description

The BioMap class contains data from short-read sequences, including sequence headers, read sequences, quality scores for the sequences, and data about how each sequence aligns to a given reference. This data is typically obtained from a high-throughput sequencing instrument.

Construct a BioMap object from short-read sequence data. Each element in the object has a sequence, header, quality score, and alignment/mapping information associated with it. Use the object properties and methods to explore, access, filter, and manipulate all or a subset of the data, before analyzing or viewing the data.

Construction

BioMapobj = BioMap constructs BioMapobj, which is an empty BioMap object.

BioMapobj = BioMap(File) constructs BioMapobj, a BioMap object, from File, a SAM- or BAM-formatted file whose reads are ordered by start position in the reference sequence. The data remains in the source file, and the BioMap object accesses it using one or two auxiliary index files. For a SAM-formatted file, MATLAB uses or creates one index file that must have the same name as the source file, but with an .idx extension. For a BAM-formatted file, MATLAB uses or creates two index files that must have the same name as the source file, but with *.bai and *.linearindex extensions. If the index files are not found in the same folder as the source file, the BioMap constructor function creates the index files in that folder.

When you pass in an unordered BAM-formatted file, the constructor automatically orders the file and writes the data to an ordered file using the same base name and extension with an added character vector "ordered" before the extension. The new file is indexed and used to instantiate the new BioMap object.

Note  Because the data remains in the source file and is accessed using the index files:

- Do not delete the source file (SAM or BAM).
- Do not delete the index files (*.idx, *.bai, or *.linearindex).
- You cannot modify BioMapobj properties.

Tip  To determine the number of reference sequences included in your source file, use the saminfo or baminfo function. Use SAMtools to check if the reads in your source file are ordered by position in the reference sequence, and also to reorder them, if needed.

BioMapobj = BioMap(Struct) constructs BioMapobj, a BioMap object, from Struct, a MATLAB structure containing sequence and alignment information, such as returned by the samread or
bamread function. The data from Struct remains in memory, which lets you modify the BioMapobj properties.

\[
\text{BioMapobj} = \text{BioMap(} \_\_\_, \text{'}Name\', \text{Value}) \]

constructs the BioMap object using any of previous input arguments and additional options, specified as name-value pair arguments as follows.

\[
\text{BioMapobj} = \text{BioMap(} \_\_\_, \text{'}SelectReference\', \text{SelectRefValue}) \]

selects one or more references when the source data contains sequences mapped to more than one reference. By default, the constructor includes all of the references in the header dictionary of the source file. When the header dictionary is not available, the constructor defaults to including all reference names found in the source data. \text{SelectRefValue} is a character vector, string, string vector, or cell array of character vectors. By using this option, you can prevent the BioMap constructor from creating auxiliary index files for references that you will not use in your analysis. If any reads mapped to selected references are paired and BioMapobj is written to a file, the reference sequences of the mates are also included in the file header.

\[
\text{BioMapobj} = \text{BioMap(File, \text{'}InMemory\', \text{InMemoryValue})} \]

specifies whether to place the data in memory or leave the data in the source file. Leaving the data in the source file and accessing via an index file is more memory efficient, but does not let you modify properties of BioMapobj. Choices are true or false (default). If the first input argument is not a file name, then this name-value pair argument is ignored, and the data is automatically placed in memory.

**Tip** Set the \text{'}InMemory\' name-value pair argument to true if you want to modify the properties of BioMapobj.

\[
\text{BioMapobj} = \text{BioMap(} \_\_\_, \text{'}IndexDir\', \text{IndexDirValue}) \]

specifies the path to the folder where the index files (*.idx,*.bai, or *.linearindex) either exist or will be created.

**Tip** Use the \text{'}IndexDir\' name-value pair argument if you do not have write access to the folder where the source file is located.

\[
\text{BioMapobj} = \text{BioMap(} \_\_\_, \text{'}Sequence\', \text{SequenceValue}) \]

constructs BioMapobj, a BioMap object, from SequenceValue that contains the letter representations of nucleotide sequences. This name-value pair works only if the data is read into memory.

\[
\text{BioMapobj} = \text{BioMap(} \_\_\_, \text{'}Header\', \text{HeaderValue}) \]

constructs BioMapobj, a BioMap object, from HeaderValue that contains header text for nucleotide sequences. This name-value pair works only if the data is read into memory.

\[
\text{BioMapobj} = \text{BioMap(} \_\_\_, \text{'}Quality\', \text{QualityValue}) \]

constructs BioMapobj, a BioMap object, from QualityValue that contains the ASCII representation of per-base quality scores for nucleotide sequences. This name-value pair works only if the data is read into memory.

\[
\text{BioMapobj} = \text{BioMap(} \_\_\_, \text{'}Reference\', \text{ReferenceValue}) \]

constructs BioMapobj, a BioMap object, and sets the Reference property to ReferenceValue that contains the names of the reference sequences. This name-value pair works only if the data is read into memory.

\[
\text{BioMapobj} = \text{BioMap(} \_\_\_, \text{'}Signature\', \text{SignatureValue}) \]

constructs BioMapobj, a BioMap object, from SignatureValue that contains information describing the alignment of each read sequence with the reference sequence. This name-value pair works only if the data is read into memory.
BioMapobj = BioMap(___,'Start',StartValue) constructs BioMapobj, a BioMap object, from StartValue, a vector of positive integers specifying the position in the reference sequence where the alignment of each read sequence starts. This name-value pair works only if the data is read into memory.

BioMapobj = BioMap(___,'Flag',FlagValue) constructs BioMapobj, a BioMap object, from FlagValue, a vector of positive integers indicating the bit-wise information for the status of the 11 flags specified by the SAM format specification. These flags describe different sequencing and alignment aspects of the read sequences. This name-value pair works only if the data is read into memory.

BioMapobj = BioMap(___,'MappingQuality',MappingQualityValue) constructs BioMapobj, a BioMap object, from MappingQualityValue, a vector of positive integers specifying the mapping quality for each read sequence. This name-value pair works only if the data is read into memory.

BioMapobj = BioMap(___,'MatePosition',MatePositionValue) constructs BioMapobj, a BioMap object, from MatePositionValue, a vector of nonnegative integers specifying the mate position for each read sequence. This name-value pair works only if the data is read into memory.

**Input Arguments**

**File**

Character vector or string specifying a SAM- or BAM-formatted file that contains only one reference sequence and whose reads are ordered by start position in the reference sequence.

**Struct**

MATLAB structure containing sequence and alignment information, such as returned by the samread or bamread function. The structure must have a one-based start position.

**Default:**

**SelectRefValue**

Character vector, string, string vector, or cell array of character vectors specifying the name of the reference sequences in File or Struct. Use saminfo or baminfo to see a complete list of reference sequences in File.

**InMemoryValue**

Logical specifying whether to place the data in memory or leave the data in the source file. Leaving the data in the source file and accessing it via an index file is more memory efficient, but does not let you modify properties of the BioMap object. If the first input argument is not a file name, then this name-value pair argument is ignored, and the data is automatically placed in memory.

**Default:** false

**IndexDirValue**

Character vector or string specifying the path to the folder where the index file either exists or will be created.

**Default:** Folder where File is located
**SequenceValue**

String vector or cell array of character vectors containing the letter representations of nucleotide sequences. This information populates the BioMap object's Sequence property. The `samread` and `bamread` functions return this information in the Sequence field of the output structure.

**QualityValue**

String vector or cell array of character vectors containing the ASCII representation of per-base quality scores for nucleotide sequences. This information populates the BioMap object's Quality property. The `samread` and `bamread` functions return this information in the Quality field of the output structure.

**HeaderValue**

String vector or cell array of character vectors containing header text for nucleotide sequences. This information populates the BioMap object's Header property. The `samread` and `bamread` functions return this information in the QueryName field of the return structure.

**NameValue**

Character vector or string describing the BioMap object. This information populates the object's Name property.

*Default:* `' '`, an empty character vector

**ReferenceValue**

String vector or cell array of character vectors containing the names of the reference sequences. This information populates the object's Reference property. The `samread` function returns this information in the ReferenceName field of the SAMStruct output argument. The `bamread` function returns this information in the Reference field of the HeaderStruct output structure.

*Default:*

**SignatureValue**

String vector or cell array of character vectors containing information describing the alignment of each read sequence with the reference sequence. The `samread` and `bamread` functions return this information in the CigarString field of the return structure. This information populates the object's Signature property.

*Default:*

**StartValue**

Vector of positive integers specifying the position in the reference sequence where the alignment of each read sequence starts. This information populates the object's Start property. The `samread` and `bamread` functions return this information in the Position field of the output structure.

*Default:*

**FlagValue**

Vector of positive integers indicating the bit-wise information for the status of the 11 flags specified by the SAM format specification. These flags describe different sequencing and alignment aspects of
the read sequences. This information populates the object's Flag property. The samread and bamread functions return this information in the Flag field of the output structure.

**Default:**

**MappingQualityValue**

Vector of positive integers specifying the mapping quality for each read sequence. This information populates the object's MappingQuality property. The samread and bamread functions return this information in the MappingQuality field of the output structure.

**Default:**

**MatePositionValue**

Vector of nonnegative integers specifying the mate position for each read sequence. This information populates the object's MatePosition property. The samread and bamread functions return this information in the MatePosition field of the output structure.

**Default:**

**Properties**

**Flag**

Flags associated with all read sequences represented in the BioMap object.

Vector of positive integers such that there is an integer for each read sequence in the object. Each integer indicates the bit-wise information that specifies the status of the 11 flags described by the SAM format specification. These flags describe different sequencing and alignment aspects of a read sequence. A one-to-one relationship exists between the number and order of elements in Flag and Sequence, unless Flag is an empty vector.

**Header**

Headers associated with all read sequences represented in the BioMap object.

Cell array of character vectors, such that there is a header for each read sequence in the object. Headers can be empty. A one-to-one relationship exists between the number and order of elements in Header and Sequence, unless Header is an empty cell array.

**MatePosition**

Positions of the mates for all read sequences represented in the BioMap object.

Vector of nonnegative integers such that there is an integer for each read sequence in the object. Each integer indicates the position of the corresponding mate sequence, relative to the reference sequence. A one-to-one relationship exists between the number and order of elements in MatePosition and Sequence, unless MatePosition is an empty vector.

Not all values in the MatePosition vector represent valid mate positions, for example, mates that map to a different reference sequence or mates that do not map. To determine if a mate position is valid, use the filterByFlag method with the 'pairedInMap' flag.
**MappingQuality**

Mapping quality scores associated with all read sequences represented in the BioMap object.

Vector of integers, such that there is a mapping quality score for each read sequence in the object. A one-to-one relationship exists between the number and order of elements in MappingQuality and Sequence, unless MappingQuality is an empty vector.

**Name**

Description of the BioMap object.

Character vector describing the BioMap object.

**Default:** ' ', an empty character vector

**NSeqs**

Number of sequences in the BioMap object.

This information is read-only.

**Quality**

Per-base quality scores associated with all read sequences represented in the BioMap object.

Cell array of character vectors, such that there is a quality for each read sequence in the object. Each quality is an ASCII representation of per-base quality scores for a read sequence. Quality can be an empty character vector. A one-to-one relationship exists between the number and order of elements in Quality and Sequence, unless Quality is an empty cell array.

**Reference**

Reference sequences in the BioMap object.

BioMapobj.NSeqs-by-1 cell array of character vectors specifying the names of the reference sequences.

The reference sequences are the sequences against which the read sequences are aligned.

**Sequence**

Read sequences in the BioMap object.

Cell array of character vectors containing the letter representations of the read sequences.

**SequenceDictionary**

Cell array of character vectors that catalogs the names of the references available in the BioMap object.

This information is read-only.

**Signature**

Alignment information associated with all read sequences represented in the BioMap object.
Cell array of CIGAR-formatted character vectors, such that there is alignment information for each read sequence in the object. Each character vector represents how a read sequence aligns to the reference sequence. Signatures can be empty character vectors. A one-to-one relationship exists between the number and order of elements in Signature and Sequence, unless Signature is an empty cell array.

**Start**

Start positions of all aligned read sequences represented in the BioMap object.

Vector of integers, such that there is a start position for each read sequence in the object. Each integer specifies the start position of the aligned read sequence with respect to the position numbers in the reference sequence. A one-to-one relationship exists between the number and order of elements in Start and Sequence, unless Start is an empty vector.

**Methods**

- **getStop** Compute stop positions of aligned read sequences from BioMap object
- **filterByFlag** Filter sequence reads by SAM flag
- **getAlignment** Construct alignment represented in BioMap object
- **getBaseCoverage** Return base-by-base alignment coverage of reference sequence in BioMap object
- **getCompactAlignment** Construct compact alignment represented in BioMap object
- **getCounts** Return count of read sequences aligned to reference sequence in BioMap object
- **getFlag** Retrieve read sequence flags from BioMap object
- **getIndex** Return indices of read sequences aligned to reference sequence in BioMap object
- **getInfo** Retrieve information for single element of BioMap object
- **getMappingQuality** Retrieve sequence mapping quality scores from BioMap object
- **getMatePosition** Retrieve mate positions of read sequences from BioMap object
- **getReference** Retrieve reference sequence from BioMap object
- **getSignature** Retrieve signature (alignment information) from BioMap object
- **getStart** Retrieve start positions of aligned read sequences from BioMap object
- **getSummary** Print summary of BioMap object
- **setFlag** Set read sequence flags for BioMap object
- **setMappingQuality** Set sequence mapping quality scores for BioMap object
- **setMatePosition** Set mate positions of read sequences in BioMap object
- **setName** Set name of reference sequence for BioMap object
- **setSignature** Set signature (alignment information) for BioMap object
- **setStart** Set start positions of aligned read sequences in BioMap object
Inherited Methods

- **combine**: Combine two objects
- **get**: Retrieve property of object
- **getHeader**: Retrieve sequence headers from object
- **getQuality**: Retrieve sequence quality information from object
- **getSequence**: Retrieve sequences from object
- **getSubsequence**: Retrieve partial sequences from object
- **getSubset**: Retrieve subset of elements from object
- **set**: Set property of object
- **setHeader**: Update header information of reads
- **setQuality**: Update quality information
- **setSequence**: Update read sequences
- **setSubsequence**: Update partial sequences
- **setSubset**: Update elements of object
- **write**: Write contents of BioRead or BioMap object to file

Copy Semantics

Value. To learn how value classes affect copy operations, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

Indexing

BioMap objects support dot . indexing to extract, assign, and delete data.

Examples

Construct a BioMap object

This example shows how to construct a BioMap object from a SAM file and from a structure.

Construct a BioMap object from a SAM-formatted file that is provided with Bioinformatics Toolbox™ and set the Name property.

```matlab
BMObj1 = BioMap('ex1.sam', 'Name', 'MyObject')
BMObj1 = BioMap with properties:
    SequenceDictionary: 'seq1'
    Reference: [1501x1 File indexed property]
    Signature: [1501x1 File indexed property]
    Start: [1501x1 File indexed property]
    MappingQuality: [1501x1 File indexed property]
    Flag: [1501x1 File indexed property]
    MatePosition: [1501x1 File indexed property]
```
Construct a structure containing information from a SAM file.

SAMStruct = samread('ex1.sam');

Construct a BioMap object from this structure.

BMObj2 = BioMap(SAMStruct)

BMObj2 =
BioMap with properties:

SequenceDictionary: {'seq1'}
Reference: {1501x1 cell}
Signature: {1501x1 cell}
Start: [1501x1 uint32]
MappingQuality: [1501x1 uint8]
Flag: [1501x1 uint16]
MatePosition: [1501x1 uint32]
Quality: {1501x1 cell}
Sequence: {1501x1 cell}
Header: {1501x1 cell}
NSeqs: 1501
Name: ''

See Also
BioIndexedFile | BioRead | align2cigar | bamindexread | baminfo | bamread | cigar2align | saminfo | samread

Topics
"Manage Sequence Read Data in Objects"

External Websites
Sequence Read Archive
SAM format specification
aa2nt

Convert amino acid sequence to nucleotide sequence

Syntax

SeqNT = aa2nt(SeqAA)

SeqNT = aa2nt(SeqAA, ...'GeneticCode', GeneticCodeValue, ...)
SeqNT = aa2nt(SeqAA, ...'Alphabet' AlphabetValue, ...)

Input Arguments

<table>
<thead>
<tr>
<th>SeqAA</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string of single-letter codes specifying an amino acid sequence. For valid letter codes, see the table Mapping Amino Acid Letter Codes to Integers. Unknown characters are mapped to 0.</td>
</tr>
<tr>
<td></td>
<td>• Row vector of integers specifying an amino acid sequence. For valid integers, see the table Mapping Amino Acid Integers to Letter Codes.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that contains an amino acid sequence, such as returned by fastaread, getgenpept, genpeptread, getpdb, or pdbread.</td>
</tr>
<tr>
<td></td>
<td>Examples: ‘ARN’ or [1 2 3]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GeneticCodeValue</th>
<th>Integer, character vector, or string specifying a genetic code number or code name from the table Genetic Code. Default is 1 or 'Standard'.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tip</td>
<td>If you use a code name, you can truncate the name to the first two letters of the name.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AlphabetValue</th>
<th>Character vector or string specifying a nucleotide alphabet. Choices are:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• 'DNA' (default) — Uses the symbols A, C, G, and T.</td>
</tr>
<tr>
<td></td>
<td>• 'RNA' — Uses the symbols A, C, G, and U.</td>
</tr>
</tbody>
</table>

Output Arguments

| SeqNT          | Nucleotide sequence specified by a character vector of letter codes. |

Description

SeqNT = aa2nt(SeqAA) converts an amino acid sequence, specified by SeqAA, to a nucleotide sequence, returned in SeqNT, using the standard genetic code.
In general, the mapping from an amino acid to a nucleotide codon is not a one-to-one mapping. For amino acids with multiple possible nucleotide codons, this function randomly selects a codon corresponding to that particular amino acid. For the ambiguous characters B and Z, one of the amino acids corresponding to the letter is selected randomly, and then a codon sequence is selected randomly. For the ambiguous character X, a codon sequence is selected randomly from all possibilities.

\[
SeqNT = \text{aa2nt}(\text{SeqAA}, \ldots, 'PropertyName', PropertyValue, \ldots)
\]
calls \text{aa2nt} with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each \textit{PropertyName} must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

\[
SeqNT = \text{aa2nt}(\text{SeqAA}, \ldots, 'GeneticCode', GeneticCodeValue, \ldots)
\]
specifies a genetic code to use when converting an amino acid sequence to a nucleotide sequence. \textit{GeneticCodeValue} can be an integer, character vector, or string specifying a code number or code name from the table Genetic Code. Default is 1 or 'Standard'. The amino acid to nucleotide codon mapping for the Standard genetic code is shown in the table Standard Genetic Code.

\textbf{Tip} If you use a code name, you can truncate the name to the first two letters of the name.

\[
SeqNT = \text{aa2nt}(\text{SeqAA}, \ldots, 'Alphabet', AlphabetValue, \ldots)
\]
specifies a nucleotide alphabet. \textit{AlphabetValue} can be 'DNA', which uses the symbols A, C, G, and T, or 'RNA', which uses the symbols A, C, G, and U. Default is ‘DNA’.
## Genetic Code

<table>
<thead>
<tr>
<th>Code Number</th>
<th>Code Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Standard</td>
</tr>
<tr>
<td>2</td>
<td>Vertebrate Mitochondrial</td>
</tr>
<tr>
<td>3</td>
<td>Yeast Mitochondrial</td>
</tr>
<tr>
<td>4</td>
<td>Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma/Spiroplasma</td>
</tr>
<tr>
<td>5</td>
<td>Invertebrate Mitochondrial</td>
</tr>
<tr>
<td>6</td>
<td>Ciliate, Dasycladacean, and Hexamita Nuclear</td>
</tr>
<tr>
<td>9</td>
<td>Echinoderm Mitochondrial</td>
</tr>
<tr>
<td>10</td>
<td>Euplotid Nuclear</td>
</tr>
<tr>
<td>11</td>
<td>Bacterial and Plant Plastid</td>
</tr>
<tr>
<td>12</td>
<td>Alternative Yeast Nuclear</td>
</tr>
<tr>
<td>13</td>
<td>Ascidian Mitochondrial</td>
</tr>
<tr>
<td>14</td>
<td>Flatworm Mitochondrial</td>
</tr>
<tr>
<td>15</td>
<td>Blepharisma Nuclear</td>
</tr>
<tr>
<td>16</td>
<td>Chlorophycean Mitochondrial</td>
</tr>
<tr>
<td>21</td>
<td>Trematode Mitochondrial</td>
</tr>
<tr>
<td>22</td>
<td>Scenedesmus Obliquus Mitochondrial</td>
</tr>
<tr>
<td>23</td>
<td>Thraustochytrium Mitochondrial</td>
</tr>
</tbody>
</table>
## Standard Genetic Code

<table>
<thead>
<tr>
<th>Amino Acid Name</th>
<th>Amino Acid Code</th>
<th>Nucleotide Codon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alanine</td>
<td>A</td>
<td>GCT GCC GCA GCG</td>
</tr>
<tr>
<td>Arginine</td>
<td>R</td>
<td>CGT CGC CGA CGG AGA AGG</td>
</tr>
<tr>
<td>Asparagine</td>
<td>N</td>
<td>AAT AAC</td>
</tr>
<tr>
<td>Aspartic acid (Aspartate)</td>
<td>D</td>
<td>GAT GAC</td>
</tr>
<tr>
<td>Cysteine</td>
<td>C</td>
<td>TGT TGC</td>
</tr>
<tr>
<td>Glutamine</td>
<td>Q</td>
<td>CAA CAG</td>
</tr>
<tr>
<td>Glutamic acid (Glutamate)</td>
<td>E</td>
<td>GAA GAG</td>
</tr>
<tr>
<td>Glycine</td>
<td>G</td>
<td>GGT GGC GGA GGG</td>
</tr>
<tr>
<td>Histidine</td>
<td>H</td>
<td>CAT CAC</td>
</tr>
<tr>
<td>Isoleucine</td>
<td>I</td>
<td>ATT ATC ATA</td>
</tr>
<tr>
<td>Leucine</td>
<td>L</td>
<td>TTA TTG CTT CTC CTA CTG</td>
</tr>
<tr>
<td>Lysine</td>
<td>K</td>
<td>AAA AAG</td>
</tr>
<tr>
<td>Methionine</td>
<td>M</td>
<td>ATG</td>
</tr>
<tr>
<td>Phenylalanine</td>
<td>F</td>
<td>TTT TTC</td>
</tr>
<tr>
<td>Proline</td>
<td>P</td>
<td>CCT CCC CCA CCG</td>
</tr>
<tr>
<td>Serine</td>
<td>S</td>
<td>TCT TCC TCA TCG AGT AGC</td>
</tr>
<tr>
<td>Threonine</td>
<td>T</td>
<td>ACT ACC ACA ACG</td>
</tr>
<tr>
<td>Tryptophan</td>
<td>W</td>
<td>TGG</td>
</tr>
<tr>
<td>Tyrosine</td>
<td>Y</td>
<td>TAT, TAC</td>
</tr>
<tr>
<td>Valine</td>
<td>V</td>
<td>GTT GTC GTA GTG</td>
</tr>
<tr>
<td>Asparagine or Aspartic acid (Aspartate)</td>
<td>B</td>
<td>Random codon from D and N</td>
</tr>
<tr>
<td>Glutamine or Glutamic acid (Glutamate)</td>
<td>Z</td>
<td>Random codon from E and Q</td>
</tr>
<tr>
<td>Unknown amino acid (any amino acid)</td>
<td>X</td>
<td>Random codon</td>
</tr>
<tr>
<td>Translation stop</td>
<td>*</td>
<td>TAA TAG TGA</td>
</tr>
<tr>
<td>Gap of indeterminate length</td>
<td>-</td>
<td>- - -</td>
</tr>
<tr>
<td>Unknown character (any character or symbol not in table)</td>
<td>?</td>
<td>???</td>
</tr>
</tbody>
</table>

### Examples

**Convert an amino acid sequence to a nucleotide sequence**

Create an amino acid sequence.
seq = randseq(20,'alphabet','amino')
seq = 'TYNYMRQLVVDVVITNHYSV'

Convert it to a nucleotide sequence using the standard genetic code.

aa2nt(seq)
an = 'ACATATAACTACATGACCGTTGACGTTGTCATTACTAACTATAGCGTT'

Convert it using the Vertebrate Mitochondrial genetic code.

aa2nt(seq,'GeneticCode',2)
an = 'ACCTATAACTACATACGCCAACTCGTAGTGATATTACTATCCATCCGTT'

Convert using the Echinoderm Mitochondrial genetic code and the RNA alphabet.

aa2nt(seq,'GeneticCode','ec','Alphabet','RNA')
an = 'ACGUAUAACUACAAUGCGCGCAGUUAGUGUGUCGUGAUUACGCAACAUUAUUGUGUC'

See Also
aminolookup | baselookup | geneticcode | nt2aa | rand | revgeneticcode | seqviewer

Introduced before R2006a
aaccount

Count amino acids in sequence

Syntax

\[ \text{AAStruct} = \text{aaccount}(\text{SeqAA}) \]
\[ \text{AAStruct} = \text{aaccount}(\text{SeqAA}, \ldots '\text{Ambiguous}', \text{AmbiguousValue}, \ldots) \]
\[ \text{AAStruct} = \text{aaccount}(\text{SeqAA}, \ldots '\text{Gaps}', \text{GapsValue}, \ldots) \]
\[ \text{AAStruct} = \text{aaccount}(\text{SeqAA}, \ldots '\text{Chart}', \text{ChartValue}, \ldots) \]

Input Arguments

<table>
<thead>
<tr>
<th>SeqAA</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string containing single-letter codes specifying an amino acid sequence. For valid letter</td>
</tr>
<tr>
<td></td>
<td>codes, see the table Mapping Amino Acid Letter Codes to Integers. Unknown characters are mapped to 0.</td>
</tr>
<tr>
<td></td>
<td>• Row vector of integers specifying an amino acid sequence. For valid integers, see the table Mapping Amino Acid</td>
</tr>
<tr>
<td></td>
<td>Integers to Letter Codes.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that contains an amino acid sequence, such as returned by fastaread,</td>
</tr>
<tr>
<td></td>
<td>getgenpept, genpeptread, getpdb, or pdbread.</td>
</tr>
<tr>
<td></td>
<td>Examples: 'ARN' or [1 2 3]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AmbiguousValue</th>
<th>Character vector or string specifying how to treat ambiguous amino acid characters (B, Z, or X). Choices are:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• 'ignore' (default) — Skips ambiguous characters</td>
</tr>
<tr>
<td></td>
<td>• 'bundle' — Counts ambiguous characters and reports the total count in the Ambiguous field.</td>
</tr>
<tr>
<td></td>
<td>• 'prorate' — Counts ambiguous characters and distributes them proportionately in the appropriate fields. For</td>
</tr>
<tr>
<td></td>
<td>example, the counts for the character B are distributed evenly between the D and N fields.</td>
</tr>
<tr>
<td></td>
<td>• 'individual' — Counts ambiguous characters and reports them in individual fields.</td>
</tr>
<tr>
<td></td>
<td>• 'warn' — Skips ambiguous characters symbols and displays a warning.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GapsValue</th>
<th>Specifies whether gaps, indicated by a hyphen (-), are counted or ignored. Choices are true or false (default).</th>
</tr>
</thead>
</table>

| ChartValue | Character vector or string specifying a chart type. Choices are 'pie' or 'bar'.                                        |
Output Arguments


Description

**AAStruct** = aacount(SeqAA) counts the number of each type of amino acid in SeqAA, an amino acid sequence, and returns the counts in AAStruct, a 1-by-1 MATLAB structure containing fields for the standard 20 amino acids (A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, and V).

- Ambiguous amino acid characters (B, Z, or X), gaps, indicated by a hyphen (-), and end terminators (*) are ignored by default.
- Unrecognized characters are ignored and cause the following warning message.

  Warning: Unknown symbols appear in the sequence. These will be ignored.

**AAStruct** = aacount(SeqAA, ...'PropertyName', PropertyValue, ...) calls aacount with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

**AAStruct** = aacount(SeqAA, ...'Ambiguous', AmbiguousValue, ...) specifies how to treat ambiguous amino acid characters (B, Z, or X). Choices are:

- 'ignore' (default)
- 'bundle'
- 'prorate'
- 'individual'
- 'warn'

**AAStruct** = aacount(SeqAA, ...'Gaps', GapsValue, ...) specifies whether gaps, indicated by a hyphen (-), are counted or ignored. Choices are true or false (default).

**AAStruct** = aacount(SeqAA, ...'Chart', ChartValue, ...) creates a chart showing the relative proportions of the amino acids. ChartValue can be 'pie' or 'bar'.

Examples

Count amino acids in a sequence

Use the fastaread function to load the sequence of the human p53 tumor protein.

```matlab
p53 = fastaread('p53aa.txt')
p53 = struct with fields:
    Header: 'gi|8400738|ref|NP_000537.2| tumor protein p53 [Homo sapiens]'
    Sequence: 'MEEPQDSPSVEPPLSQETFSDLWLKLLPLLNNVSLPSQAMDDLMLSPDIEQWFTEDPGDEAPRMPEAAPRVAAPAPAPP'
```

Count the amino acids in the sequence, and display the results in a pie chart.
count = aacount(p53,'Chart','pie');

See Also
aminolookup | atomiccomp | basecount | codoncount | dimercount | isoelectric | molweight | proteinplot | proteinpropplot | seqviewer

Introduced before R2006a
abstract

**Class:** bioma.ExpressionSet  
**Package:** bioma

Retrieve or set abstract describing experiment in ExpressionSet object

**Syntax**

```
Abstract = abstract(ESObj)
NewESObj = abstract(ESObj, NewAbstract)
```

**Description**

`Abstract = abstract(ESObj)` returns a character vector containing the abstract information describing the experiment from a MIAME object in an ExpressionSet object.

`NewESObj = abstract(ESObj, NewAbstract)` replaces the abstract information in the MIAME object in `ESObj`, an ExpressionSet object, with `NewAbstract`, a character vector containing new abstract information, and returns `NewESObj`, a new ExpressionSet object.

**Input Arguments**

**ESObj**
Object of the bioma.ExpressionSet class.

**Default:**

**NewAbstract**
Character vector containing new abstract information.

**Default:**

**Output Arguments**

**Abstract**
Character vector containing the abstract information describing the experiment from a MIAME object in an ExpressionSet object.

**NewESObj**
Object of the bioma.ExpressionSet class, returned after replacing the abstract information.

**Examples**

Construct an ExpressionSet object, `ESObj`, as described in the “Examples” on page 1-0 section of the bioma.ExpressionSet class reference page. Retrieve the abstract information stored in the MIAME object stored in the ExpressionSet object:
% Retrieve abstract text from the MIAME object
Abstract = abstract(ESObj)

**See Also**
bioma.ExpressionSet | bioma.data.MIAME

**Topics**
“Managing Gene Expression Data in Objects”
addTitle

Add title to heatmap or clustergram

Syntax

addTitle(hm_cg_object,title)
addTitle(hm_cg_object,title,Name,Value)
textObj = addTitle(__)

Description

addTitle(hm_cg_object,title) adds a title to the heatmap or clustergram.

addTitle(hm_cg_object,title,Name,Value) specifies the title text object properties using name-value pair arguments. For example, addTitle(hm_cg_object,'Gene Expression Data','Color','red','FontSize',12) displays the title in red 12-point font. You can specify multiple name-value pairs. Enclose each property name in quotes.

textObj = addTitle(__) returns a text object textObj used as the title of the heatmap or clustergram using any of the input argument combinations in the previous syntaxes.

Examples

Add Custom Title and Labels to Heatmap

Load a sample of gene expression data.

load bc_train_filtered

Display a heatmap of the gene expression values for 4918 genes from 78 samples.

hmo = HeatMap(bcTrainData.Log10Ratio);

    Standardize: '[column | row | {none}]'
    Symmetric: '[true | false].'
    DisplayRange: 'Scalar.'
    Colormap: []
    ImputeFun: 'string -or- function handle -or- cell array'
    ColumnLabels: 'Cell array of strings, or an empty cell array'
    RowLabels: 'Cell array of strings, or an empty cell array'
    ColumnLabelsRotate: []
    RowLabelsRotate: []
    Annotate: '[on | {off}]'
    AnnotPrecision: []
    AnnotColor: []
    ColumnLabelsColor: 'A structure array.'
    RowLabelsColor: 'A structure array.'
    LabelsWithMarkers: '[true | false].'
    ColumnLabelsLocation: '[ top | {bottom} ]'
    RowLabelsLocation: '{ {left} | right }'
Add a title to the heatmap in red.

title = addTitle(hmo,'Gene Expression Data','Color','red');
Change the title font size.

title.FontSize = 12;
Add labels to the x-axis and y-axis.

```matlab
addXLabel(hmo,'Samples','FontSize',12);
addYLabel(hmo,'Genes','FontSize',12);
```
**Input Arguments**

hm_cg_object — Heatmap or clustergram object  
HeatMap object | clustergram object

Heatmap or clustergram object, specified as a HeatMap object or clustergram object.

title — Title of heatmap or clustergram  
character vector | string

Title of the heatmap or clustergram, specified as a character vector or string.

Example: 'Inverse of an Upper Hessenberg Matrix'

Data Types: char | string

**Output Arguments**

textObj — Heatmap or clustergram title  
Text object

Heatmap or clustergram title, returned as a Text object.

**See Also**  
HeatMap | Text Properties | clustergram
Introduced in R2009b
addXLabel

Label x-axis of heatmap or clustergram

Syntax

addXLabel(hm_cg_object,label)
addXLabel(hm_cg_object,label,Name,Value)
textObj = addXLabel(___)

Description

addXLabel(hm_cg_object,label) adds a label below the x-axis of the heatmap or clustergram.

addXLabel(hm_cg_object,label,Name,Value) specifies the label text object properties using name-value pair arguments. For example, addXLabel(hmObj,'Samples','Color','red','FontSize',12) displays the label in red 12-point font. You can specify multiple name-value pairs. Enclose each property name in quotes.

textObj = addXLabel(___ ) returns a text object textObj used as the label of the heatmap or clustergram using any of the input argument combinations from the previous syntaxes.

Examples

Add Custom Title and Labels to Heatmap

Load a sample of gene expression data.

close bc_train_filtered

Display a heatmap of the gene expression values for 4918 genes from 78 samples.

hmo = HeatMap(bcTrainData.Log10Ratio);

    Standardize: '[column | row | {none}]'
    Symmetric: '[true | false]'
    DisplayRange: 'Scalar.'
    Colormap: []
    ImputeFun: 'string -or- function handle -or- cell array'
    ColumnLabels: 'Cell array of strings, or an empty cell array'
    RowLabels: 'Cell array of strings, or an empty cell array'
    ColumnLabelsRotate: []
    RowLabelsRotate: []
    Annotate: '[on | {off}]'
    AnnotPrecision: []
    AnnotColor: []
    ColumnLabelsColor: 'A structure array.'
    RowLabelsColor: 'A structure array.'
    LabelsWithMarkers: '[true | false]'
    ColumnLabelsLocation: '[ top | {bottom} ]'
    RowLabelsLocation: '[ {left} | right ]'
Add a title to the heatmap in red.

title = addTitle(hmo,'Gene Expression Data','Color','red');
Change the title font size.

title.FontSize = 12;
Add labels to the x-axis and y-axis.

```matlab
addXLabel(hmo,'Samples','FontSize',12);
addYLabel(hmo,'Genes','FontSize',12);
```
Input Arguments

**hm_cg_object — Heatmap or clustergram object**

HeatMap object | clustergram object

Heatmap or clustergram object, specified as a HeatMap object or clustergram object.

**label — x-axis label**

character vector | string

x-axis label, specified as a character vector or string.

Example: 'Samples'

Data Types: char | string

Output Arguments

**textObj — x-axis label**

Text object

x-axis label, returned as a Text object.

See Also

HeatMap | Text Properties | clustergram
Introduced in R2009b
addYLabel

Label y-axis of heatmap or clustergram

Syntax

addYLabel(hm_cg_object,label)
addYLabel(hm_cg_object,label,Name,Value)
textObj = addYLabel(____)  

Description

addYLabel(hm_cg_object,label) adds a label below the y-axis of the heatmap or clustergram.

addYLabel(hm_cg_object,label,Name,Value) specifies the label text object properties using name-value pair arguments. For example,

addYLabel(hmObj,'Samples','Color','red','FontSize',12) displays the label in red 12-point font. You can specify multiple name-value pairs. Enclose each property name in quotes.

textObj = addYLabel(____) returns a text object textObj used as the label of the heatmap or clustergram using any of the input argument combinations from the previous syntaxes.

Examples

Add Custom Title and Labels to Heatmap

Load a sample of gene expression data.

load bc_train_filtered

Display a heatmap of the gene expression values for 4918 genes from 78 samples.

hmo = HeatMap(bcTrainData.Log10Ratio);

    Standardize: '[column | row | {none}]'
    Symmetric: '[true | false].'
    DisplayRange: 'Scalar.'
    Colormap: []
    ImputeFun: 'string -or- function handle -or- cell array'
    ColumnLabels: 'Cell array of strings, or an empty cell array'
    RowLabels: 'Cell array of strings, or an empty cell array'
    ColumnLabelsRotate: []
    RowLabelsRotate: []
    Annotate: '[on | {off}]'
    AnnotPrecision: []
    AnnotColor: []
    ColumnLabelsColor: 'A structure array.'
    RowLabelsColor: 'A structure array.'
    LabelsWithMarkers: '[true | false].'
    ColumnLabelsLocation: '[ top | {bottom} ]'
    RowLabelsLocation: '[ {left} | right ]'
Add a title to the heatmap in red.

title = addTitle(hmo,'Gene Expression Data','Color','red');
Change the title font size.

title.FontSize = 12;
Add labels to the x-axis and y-axis.

addXLabel(hmo,'Samples','FontSize',12);
addYLabel(hmo,'Genes','FontSize',12);
**Input Arguments**

- **hm_cg_object** — Heatmap or clustergram object
  
  HeatMap object | clustergram object

  Heatmap or clustergram object, specified as a HeatMap object or clustergram object.

- **label** — *y*-axis label
  
  character vector | string

  *y*-axis label, specified as a character vector or string.

  Example: ‘Genes’

  Data Types: char | string

**Output Arguments**

- **textObj** — *y*-axis label
  
  Text object

  *y*-axis label, returned as a Text object.

**See Also**

HeatMap | Text Properties | clustergram
Introduced in R2009b
**affygcrma**

Perform GC Robust Multi-array Average (GCRMA) procedure on Affymetrix microarray probe-level data

**Syntax**

Expression = affygcrma(CELFiles, CDFFile, SeqFile)
Expression = affygcrma(ProbeStructure, Seq)

Expression = affygcrma(CELFiles, CDFFile, SeqFile, ...'CELPath', CELPathValue, ...)
Expression = affygcrma(CELFiles, CDFFile, SeqFile, ...'CDFPath', CDFPathValue, ...)
Expression = affygcrma(CELFiles, CDFFile, SeqFile, ...'SeqPath', SeqPathValue, ...)
Expression = affygcrma(..., 'ChipIndex', ChipIndexValue, ...)
Expression = affygcrma(..., 'OpticalCorr', OpticalCorrValue, ...)
Expression = affygcrma(..., 'CorrConst', CorrConstValue, ...)
Expression = affygcrma(..., 'Method', MethodValue, ...)
Expression = affygcrma(..., 'TuningParam', TuningParamValue, ...)
Expression = affygcrma(..., 'GSBCorr', GSBCorrValue, ...)
Expression = affygcrma(..., 'Median', MedianValue, ...)
Expression = affygcrma(..., 'Output', OutputValue, ...)
Expression = affygcrma(..., 'Showplot', ShowplotValue, ...)
Expression = affygcrma(..., 'Verbose', VerboseValue, ...)

**Input Arguments**

| CELFiles | Any of the following:
| --- | --- |
| Any of the following: | • Character vector or string specifying a single CEL file name.
| • '*' , which reads all CEL files in the current folder. | • ' ', which opens the Select CEL Files dialog box from which you select the CEL files. From this dialog box, you can press and hold Ctrl or Shift while clicking to select multiple CEL files.
| • Cell array of character vectors or string vector containing CEL file names. | • Cell array of character vectors or string vector containing CEL file names. |

| CDFFile | Either of the following:
| --- | --- |
| Either of the following: | • Character vector or string specifying a CDF file name.
| • ' ', which opens the Select CDF File dialog box from which you select the CDF file. | • Character vector or string specifying a CDF file name.
| • ' ', which opens the Select CDF File dialog box from which you select the CDF file. |
**SeqFile**
Either of the following:

- Character vector or string specifying a file name of a sequence file (tab-separated or FASTA) that contains the following information for a specific type of Affymetrix® GeneChip® array:
  - Probe set IDs
  - Probe x-coordinates
  - Probe y-coordinates
  - Probe sequences in each probe set
  - Affymetrix GeneChip array type (FASTA file only)

The sequence file (tab-separated or FASTA) must be on the MATLAB search path or in the Current Folder (unless you use the SeqPath property). In a tab-separated file, each row represents a probe; in a FASTA file, each header represents a probe.

- An N-by-25 matrix of sequence information, such as returned by `affyprobeseqread`.

**Seq**
An N-by-25 matrix of sequence information, such as returned by `affyprobeseqread`.

**ProbeStructure**
MATLAB structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs, returned by the `celintensityread` function.

**CELPathValue**
Character vector or string specifying the path and folder where the files specified in `CELFiles` are stored.

**CDFPathValue**
Character vector or string specifying the path and folder where the file specified in `CDFFile` is stored.

**SeqPathValue**
Character vector or string specifying a folder or path and folder where `SeqFile` is stored.

**ChipIndexValue**
Positive integer specifying a chip. This chip's sequence information and mismatch probe intensity data is used to compute probe affinities. Default is 1.

**OpticalCorrValue**
Controls the use of optical background correction on the input probe intensity values. Choices are `true` (default) or `false`.

**CorrConstValue**
Value that specifies the correlation constant, rho, for log background intensity for each PM/MM probe pair. Choices are any value $\geq 0$ and $\leq 1$. Default is 0.7.

**MethodValue**
Character vector or string that specifies the method to estimate the signal. Choices are 'MLE', a faster, ad hoc Maximum Likelihood Estimate method, or 'EB', a slower, more formal, empirical Bayes method. Default is 'MLE'.
**TuningParamValue**
Value that specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is 5 (MLE) or 0.5 (EB).

**Tip** For information on determining a setting for this parameter, see Wu et al., 2004 on page 1-659.

**GSBCorrValue**
Specifies whether to perform gene-specific binding (GSB) correction using probe affinity data. Choices are true (default) or false. If there is no probe affinity information, this property is ignored.

**MedianValue**
Specifies the use of the median of the ranked values instead of the mean for normalization. Choices are true or false (default).

**OutputValue**
Specifies the scale of the returned gene expression values. Choices are:
- 'log'
- 'log2'
- 'log10'
- 'linear'
- @functionname

In the last instance, the data is transformed as defined by the function functionname. Default is 'log2'.

**ShowplotValue**
Controls the display of a plot showing the $\log_2$ of mismatch (MM) probe intensity values from a specified chip (CEL file), versus that chip's MM probe affinities. The plot also shows the LOWESS fit for computing NSB data of the specified chip. Choices are true, false, or I, an integer specifying a chip. If set to true, the first chip is plotted. Default is:
- false — When return values are specified.
- true — When return values are not specified.

**VerboseValue**
Controls the display of the status of the reading of files and GCRMA processing. Choices are true (default) or false.

### Output Arguments

**Expression**
DataMatrix object on page 1-532 containing the $\log_2$ gene expression values that have been background adjusted, normalized, and summarized using the GC Robust Multi-array Average (GCRMA) procedure.

Each row in **Expression** corresponds to a gene (probe set), and each column corresponds to an Affymetrix CEL file.
Description

Expression = affygcrma(CELFiles, CDFFile, SeqFile) reads the specified Affymetrix CEL files, the associated CDF library file (created from Affymetrix GeneChip arrays for expression or genotyping assays), and the associated sequence file or matrix. It then processes the probe intensity values using GCRMA background adjustment, quantile normalization, and median-polish summarization procedures, then returns Expression, a DataMatrix object on page 1-532 containing the log₂ based gene expression values in a matrix, the probe set IDs as row names, and the CEL file names as column names. Note that each row in Expression corresponds to a gene (probe set), and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)

CELFiles is a character vector, string, string vector, or cell array of character vectors containing CEL file names. CDFFile is a character vector or string specifying a CDF file name. If you set CELFiles to '*' , then it reads all CEL files in the current folder. If you set CELFiles or CDFFile to ' ' , then it opens the Select Files dialog box from which you select the CEL files or CDF file. From this dialog box, you can press and hold Ctrl or Shift while clicking to select multiple CEL files. SeqFile is a file or matrix containing sequence information for probes on a specific type of Affymetrix GeneChip array.

Note For details on the reading of files and GCRMA processing, see celintensityread, affyprobeseqread, affyprobeaffinities, gcrma, gcrmabackadj, quantilenorm, and rmasummary.

Expression = affygcrma(ProbeStructure, Seq) uses GCRMA background adjustment, quantile normalization, and median-polish summarization procedures to process the probe intensity values in ProbeStructure. ProbeStructure is a MATLAB structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs, returned by the celintensityread function. Seq is a matrix containing sequence information for probes on a specific type of Affymetrix GeneChip array.

Expression = affygcrma(..., 'PropertyName', PropertyValue, ...) calls affygcrma with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Expression = affygcrma(CELFiles, CDFFile, SeqFile, ...'CELPath', CELPathValue, ...) specifies a path and folder where the files specified by CELFiles are stored.

Expression = affygcrma(CELFiles, CDFFile, SeqFile, ...'CDFPath', CDFPathValue, ...) specifies a path and folder where the file specified by CDFFile is stored.

Expression = affygcrma(CELFiles, CDFFile, SeqFile, ...'SeqPath', SeqPathValue, ...) specifies a path and folder where the file specified by SeqFile is stored.

Expression = affygcrma(..., 'ChipIndex', ChipIndexValue, ...) computes probe affinities from MM probe intensity data using sequence information and mismatch probe intensity values from the chip specified by ChipIndexValue. Default ChipIndexValue is 1.
Expression = affygcrma(..., 'OpticalCorr', OpticalCorrValue, ...) controls the use of optical background correction on the input probe intensity values. Choices are true (default) or false.

Expression = affygcrma(..., 'CorrConst', CorrConstValue, ...) specifies the correlation constant, rho, for background intensity for each PM/MM probe pair. Choices are any value \( \geq 0 \) and \( \leq 1 \). Default is 0.7.

Expression = affygcrma(..., 'Method', MethodValue, ...) specifies the method to estimate the signal. Choices are 'MLE', a faster, ad hoc Maximum Likelihood Estimate method, or 'EB', a slower, more formal, empirical Bayes method. Default is 'MLE'.

Expression = affygcrma(..., 'TuningParam', TuningParamValue, ...) specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is 5 (MLE) or 0.5 (EB).

Tip For information on determining a setting for this parameter, see Wu et al., 2004 on page 1-659.

Expression = affygcrma(..., 'GSBCorr', GSBCorrValue, ...) specifies whether to perform gene-specific binding (GSB) correction using probe affinity data. Choices are true (default) or false. If there is no probe affinity information, this property is ignored.

Expression = affygcrma(..., 'Median', MedianValue, ...) specifies the use of the median of the ranked values instead of the mean for normalization. Choices are true or false (default).

Expression = affygcrma(..., 'Output', OutputValue, ...) specifies the scale of the returned gene expression values. OutputValue can be:

- 'log'
- 'log2'
- 'log10'
- 'linear'
- @functionname

In the last instance, the data is transformed as defined by the function functionname. Default is 'log2'.

Expression = affygcrma(..., 'Showplot', ShowplotValue, ...) controls the display of a plot showing the log2 of mismatch (MM) probe intensity values from a specified chip (CEL file), versus that chip's MM probe affinities. The plot also shows the LOWESS fit for computing NSB data of the specified chip. Choices are true, false, or I, an integer specifying a chip. If set to true, the first chip is plotted. Default is:

- false — When return values are specified.
- true — When return values are not specified.

Expression = affygcrma(..., 'Verbose', VerboseValue, ...) controls the display of the status of the reading of files and GCRMA processing. Choices are true (default) or false.
Examples

The following example assumes that you have the HG_U95Av2.CDF library file stored at D:\Affymetrix\LibFiles\HGGenome, and that your current folder points to a location containing CEL files and a sequence file associated with this CDF library file. In this example, the `affygcrma` function reads all the CEL files and the sequence file in the current folder and a CDF file in a specified folder. It also performs GCRMA background adjustment, quantile normalization, and summarization procedures on the PM probe intensity values, and returns a DataMatrix object, containing the metadata and processed data.

```r
Expression = affygcrma('*', 'HG_U95Av2.CDF','HG-U95Av2_probe_tab',
'CDFPath', 'D:\Affymetrix\LibFiles\HGGenome');
```

References


See Also

`affyprobeaffinities | affyprobeseqread | affyrma | celintensityread | gcrma | gcrmabackadj | mafdr | mattest | quantilenorm | rmasummary`
Introduced in R2008b
affyinvarsetnorm

Perform rank invariant set normalization on probe intensities from multiple Affymetrix CEL or DAT files

Syntax

NormData = affyinvarsetnorm(Data)
[NormData, MedStructure] = affyinvarsetnorm(Data)

... affyinvarsetnorm(..., 'Baseline', BaselineValue, ...)
... affyinvarsetnorm(..., 'Thresholds', ThresholdsValue, ...)
... affyinvarsetnorm(..., 'StopPercentile', StopPercentileValue, ...)
... affyinvarsetnorm(..., 'RayPercentile', RayPercentileValue, ...)
... affyinvarsetnorm(..., 'Method', MethodValue, ...)
... affyinvarsetnorm(..., 'Showplot', ShowplotValue, ...)

Arguments

<table>
<thead>
<tr>
<th>Data</th>
<th>Matrix of intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL or DAT file. (Each CEL or DAT file is generated from a separate chip. All chips should be of the same type.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MedStructure</td>
<td>Structure of each column's intensity median before and after normalization, and the index of the column chosen as the baseline.</td>
</tr>
<tr>
<td>BaselineValue</td>
<td>Property to control the selection of the column index $N$ from Data to be used as the baseline column. Default is the column index whose median intensity is the median of all the columns.</td>
</tr>
<tr>
<td>ThresholdsValue</td>
<td>Property to set the thresholds for the lowest average rank and the highest average rank, which are used to determine the invariant set. The rank invariant set is a set of data points whose proportional rank difference is smaller than a given threshold. The threshold for each data point is determined by interpolating between the threshold for the lowest average rank and the threshold for the highest average rank. Select these two thresholds empirically to limit the spread of the invariant set, but allow enough data points to determine the normalization relationship. ThresholdsValue is a 1-by-2 vector [$LT$, $HT$] where $LT$ is the threshold for the lowest average rank and $HT$ is threshold for the highest average rank. Values must be between 0 and 1. Default is [0.05, 0.005].</td>
</tr>
</tbody>
</table>
### Description

**NormData** = `affyinvarsetnorm(Data)` normalizes the values in each column (chip) of probe intensities in `Data` to a baseline reference, using the invariant set method. **NormData** is a matrix of normalized probe intensities from `Data`.

Specifically, `affyinvarsetnorm`:

- Selects a baseline index, typically the column whose median intensity is the median of all the columns.
- For each column, determines the proportional rank difference (`prd`) for each pair of ranks, `RankX` and `RankY`, from the sample column and the baseline reference.

\[
prd = \frac{\text{abs}(RankX - RankY)}{\text{abs}(MedianX - MedianY)} 
\]

- For each column, determines the invariant set of data points by selecting data points whose proportional rank differences (`prd`) are below `threshold`, which is a predetermined threshold for a given data point (defined by the `ThresholdsValue` property). It repeats the process until either no more data points are eliminated, or a predetermined percentage of data points is reached.

The invariant set is data points with a `prd < threshold`.

- For each column, uses the invariant set of data points to calculate the lowess or running median smoothing curve, which is used to normalize the data in that column.

\[
[\text{NormData}, MedStructure] = \text{affyinvarsetnorm}(\text{Data})
\]
also returns a structure of the index of the column chosen as the baseline and each column's intensity median before and after normalization.
Note If \( \textit{Data} \) contains NaN values, then \( \textit{NormData} \) will also contain NaN values at the corresponding positions.

... \texttt{affyinvarsetnorm(..., 'PropertyName', PropertyValue, ...)} calls \texttt{affyinvarsetnorm} with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each \textit{PropertyName} must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... \texttt{affyinvarsetnorm(..., 'Baseline', BaselineValue, ...)} lets you select the column index \( N \) from \( \textit{Data} \) to be the baseline column. Default is the index of the column whose median intensity is the median of all the columns.

... \texttt{affyinvarsetnorm(..., 'Thresholds', ThresholdsValue, ...)} sets the thresholds for the lowest average rank and the highest average rank, which are used to determine the invariant set. The rank invariant set is a set of data points whose proportional rank difference is smaller than a given threshold. The threshold for each data point is determined by interpolating between the threshold for the lowest average rank and the threshold for the highest average rank. Select these two thresholds empirically to limit the spread of the invariant set, but allow enough data points to determine the normalization relationship.

\( \textit{ThresholdsValue} \) is a 1-by-2 vector \([LT, HT]\), where \( LT \) is the threshold for the lowest average rank and \( HT \) is threshold for the highest average rank. Values must be between 0 and 1. Default is \([0.05, 0.005]\).

... \texttt{affyinvarsetnorm(..., 'StopPercentile', StopPercentileValue, ...)} stops the iteration process when the number of data points in the invariant set reaches \( N \) percent of the total number of data points. Default is 1.

Note If you do not use this property, the iteration process continues until no more data points are eliminated.

... \texttt{affyinvarsetnorm(..., 'RayPercentile', RayPercentileValue, ...)} selects the \( N \) percentage of the highest ranked invariant set of data points to fit a straight line through, while the remaining data points are fitted to a running median curve. The final running median curve is a piecewise linear curve. Default is 1.5.

... \texttt{affyinvarsetnorm(..., 'Method', MethodValue, ...)} selects the smoothing method for normalizing the data. When \textit{MethodValue} is 'lowess', \texttt{affyinvarsetnorm} uses the lowess method. When \textit{MethodValue} is 'runmedian', \texttt{affyinvarsetnorm} uses the running median method. Default is 'lowess'.

... \texttt{affyinvarsetnorm(..., 'Showplot', ShowplotValue, ...)} plots two pairs of scatter plots (before and after normalization). The first pair plots baseline data versus data from a specified column (chip) from the matrix \( \textit{Data} \). The second is a pair of M-A scatter plots, which plots \( M \) (ratio between baseline and sample) versus \( A \) (the average of the baseline and sample). When \textit{ShowplotValue} is 'all', \texttt{affyinvarsetnorm} plots a pair of scatter plots for each column or chip. When \textit{ShowplotValue} is a number(s) or range of numbers, \texttt{affyinvarsetnorm} plots a pair of scatter plots for the indicated column numbers (chips).
Examples

Normalize Affymetrix data

This example shows how to normalize affymetrix data. The prostatecancerrawdata.mat file used in the example contains data from Best et al., 2005.

Load a MAT-file, included with the Bioinformatics Toolbox™ software, which contains Affymetrix data variables, including pmMatrix, a matrix of PM probe intensity values from multiple CEL files.

load prostatecancerrawdata

Normalize the data in pmMatrix and plot data from columns (chips) 2 and 3. Column 1 is the baseline.

NormMatrix = affyinvarsetnorm(pmMatrix, 'Showplot',[2 3]);
References


[2] https://sites.google.com/site/dchipsoft/


See Also

affyread | celintensityread | mainvarsetnorm | malowess | manorm | quantilenorm | rma backadj | rmasummary
Introduced in R2006a
affyprobeaffinities

Compute Affymetrix probe affinities from their sequences and MM probe intensities

Syntax

\[
\text{[AffinPM, AffinMM]} = \text{affyprobeaffinities(SequenceMatrix, MMIntensity)}
\]
\[
\text{[AffinPM, AffinMM, BaseProf]} = \text{affyprobeaffinities(SequenceMatrix, MMIntensity)}
\]
\[
\text{[AffinPM, AffinMM, BaseProf, Stats]} = \text{affyprobeaffinities(SequenceMatrix, MMIntensity)}
\]
... = affyprobeaffinities(SequenceMatrix, MMIntensity, ...'ProbeIndices', ProbeIndicesValue, ...)
... = affyprobeaffinities(SequenceMatrix, MMIntensity, ...'Showplot', ShowplotValue, ...)

Input Arguments

| SequenceMatrix | An N-by-25 matrix of sequence information for the perfect match (PM) probes on an Affymetrix GeneChip array, where N is the number of probes on the array. Each row corresponds to a probe, and each column corresponds to one of the 25 sequence positions. Nucleotides in the sequences are represented by one of the following integers:
|               | • 0 — None
|               | • 1 — A
|               | • 2 — C
|               | • 3 — G
|               | • 4 — T
| **Tip**       | You can use the affyprobeseqread function to generate this matrix. If you have this sequence information in letter representation, you can convert it to integer representation using the nt2int function. |

| MMIntensity   | Column vector containing mismatch (MM) probe intensities from a CEL file, generated from a single Affymetrix GeneChip array. Each row corresponds to a probe. |
| **Tip**       | You can extract this column vector from the MMIntensities matrix returned by the celintensityread function. |
**ProbeIndicesValue**  
Column vector containing probe indexing information. Probes within a probe set are numbered 0 through \(N - 1\), where \(N\) is the number of probes in the probe set.

**Tip** You can use the `affyprobeseqread` function to generate this column vector.

**ShowplotValue**  
Controls the display of a plot showing the affinity values of each of the four bases (A, C, G, and T) for each of the 25 sequence positions, for all probes on the Affymetrix GeneChip array. Choices are true or false (default).

### Output Arguments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AffinPM</td>
<td>Column vector of PM probe affinities, computed from their probe sequences and MM probe intensities.</td>
</tr>
<tr>
<td>AffinMM</td>
<td>Column vector of MM probe affinities, computed from their probe sequences and MM probe intensities.</td>
</tr>
<tr>
<td>BaseProf</td>
<td>4-by-4 matrix containing the four parameters for a polynomial of degree 3, for each base, A, C, G, and T. Each row corresponds to a base, and each column corresponds to a parameter. These values are estimated from the probe sequences and intensities, and represent all probes on an Affymetrix GeneChip array.</td>
</tr>
</tbody>
</table>
| Stats     | Row vector containing four statistics in the following order:  
- R-square statistic  
- F statistic  
- p-value  
- Error variance |

### Description

\([\text{AffinPM}, \text{AffinMM}] = \text{affyprobeaffinities(SequenceMatrix, MMIntensity)}\) returns a column vector of PM probe affinities and a column vector of MM probe affinities, computed from their probe sequences and MM probe intensities. Each row in AffinPM and AffinMM corresponds to a probe. NaN is returned for probes with no sequence information. Each probe affinity is the sum of position-dependent base affinities. For a given base type, the positional effect is modeled as a polynomial of degree 3.

\([\text{AffinPM}, \text{AffinMM}, \text{BaseProf}] = \text{affyprobeaffinities(SequenceMatrix, MMIntensity)}\) also estimates affinity coefficients using multiple linear regression. It returns BaseProf, a 4-by-4 matrix containing the four parameters for a polynomial of degree 3, for each base, A, C, G, and T. Each row corresponds to a base, and each column corresponds to a parameter. These values are estimated from the probe sequences and intensities, and represent all probes on an Affymetrix GeneChip array.

\([\text{AffinPM}, \text{AffinMM}, \text{BaseProf}, \text{Stats}] = \text{affyprobeaffinities(SequenceMatrix, MMIntensity)}\) also returns Stats, a row vector containing four statistics in the following order:
• R-square statistic
• F statistic
• p-value
• Error variance

... = affyprobeaffinities(SequenceMatrix, MMIntensity, ...'PropertyName', PropertyValue, ...) calls affyprobeaffinities with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = affyprobeaffinities(SequenceMatrix, MMIntensity, ...'ProbeIndices', ProbeIndicesValue, ...) uses probe indices to normalize the probe intensities with the median of their probe set intensities.

**Tip** Use of the ProbeIndices property is recommended only if your MMIntensity data are not from a nonspecific binding experiment.

... = affyprobeaffinities(SequenceMatrix, MMIntensity, ...'Showplot', ShowplotValue, ...) controls the display of a plot of the probe affinity base profile. Choices are true or false (default).

**Examples**

**Calculate Affymetrix probe affinities**

This example shows how to calculate Affymetrix PM and MM probe affinities from their sequences and MM probe intensities.

Load the MAT-file, included with the Bioinformatics Toolbox™ software, that contains Affymetrix data from a prostate cancer study. The variables in the MAT-file include seqMatrix, a matrix containing sequence information for PM probes, mmMatrix, a matrix containing MM probe intensity values, and probeIndices, a column vector containing probe indexing information.

```matlab
load prostatecancerrawdata
```

Compute the Affymetrix PM and MM probe affinities from their sequences and MM probe intensities, and also plot the affinity values of each of the four bases (A, C, G, and T) for each of the 25 sequence positions, for all probes on the Affymetrix GeneChip array.

```matlab
[apm, amm] = affyprobeaffinities(seqMatrix, mmMatrix(:,1), ...
    'ProbeIndices', probeIndices, 'showplot', true);
```
The prostatecancerrawdata.mat file used in this example contains data from Best et al., 2005.

References


See Also

affygcrma | affyprobeseqread | affyread | celintensityread | probelibraryinfo

Introduced in R2007a
**affyprobeseqread**

Read data file containing probe sequence information for Affymetrix GeneChip array

**Syntax**

\[ \text{Struct} = \text{affyprobeseqread} \left( \text{SeqFile, CDFFile} \right) \]

\[ \text{Struct} = \text{affyprobeseqread} \left( \text{SeqFile, CDFFile, ...'SeqPath', SeqPathValue, ...} \right) \]

\[ \text{Struct} = \text{affyprobeseqread} \left( \text{SeqFile, CDFFile, ...'CDFPath', CDFPathValue, ...} \right) \]

\[ \text{Struct} = \text{affyprobeseqread} \left( \text{SeqFile, CDFFile, ...'SeqOnly', SeqOnlyValue, ...} \right) \]

**Input Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **SeqFile**| Character vector or string specifying a file name of a sequence file (tab-separated or FASTA) that contains the following information for a specific type of Affymetrix GeneChip array:  
  • Probe set IDs  
  • Probe x-coordinates  
  • Probe y-coordinates  
  • Probe sequences in each probe set  
  • Affymetrix GeneChip array type (FASTA file only)  

The sequence file (tab-separated or FASTA) must be on the MATLAB search path or in the Current Folder (unless you use the SeqPath property). In a tab-separated file, each row represents a probe; in a FASTA file, each header represents a probe. |
| **CDFFile**| Either of the following:  
  • Character vector or string specifying a file name of an Affymetrix CDF library file, which contains information that specifies which probe set each probe belongs to on a specific type of Affymetrix GeneChip array. The CDF library file must be on the MATLAB search path or in the MATLAB Current Folder (unless you use the CDFPath property).  
  • CDF structure, such as returned by the affyread function, which contains information that specifies which probe set each probe belongs to on a specific type of Affymetrix GeneChip array. |
| **SeqPathValue**| Character vector or string specifying a folder or path and folder where SeqFile is stored. |
| **CDFPathValue**| Character vector or string specifying a folder or path and folder where CDFFile is stored. |

**Caution** Make sure that SeqFile and CDFFile contain information for the same type of Affymetrix GeneChip array.
**SeqOnlyValue** Controls the return of a structure, *Struct*, with only one field, *SequenceMatrix*. Choices are true or false (default).

**Output Arguments**

<table>
<thead>
<tr>
<th><em>Struct</em></th>
<th>MATLAB structure containing the following fields:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• ProbeSetIDs</td>
</tr>
<tr>
<td></td>
<td>• ProbeIndices</td>
</tr>
<tr>
<td></td>
<td>• SequenceMatrix</td>
</tr>
</tbody>
</table>

**Description**

*Struct* = `affyprobeseqread(SeqFile, CDFFile)` reads the data from files *SeqFile* and *CDFFile*, and stores the data in the MATLAB structure *Struct*, which contains the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProbeSetIDs</td>
<td>Cell array containing the probe set IDs from the Affymetrix CDF library file.</td>
</tr>
<tr>
<td>ProbeIndices</td>
<td>Column vector containing probe indexing information. Probes within a probe set are numbered 0 through (N - 1), where (N) is the number of probes in the probe set.</td>
</tr>
<tr>
<td>SequenceMatrix</td>
<td>An (N)-by-25 matrix of sequence information for the perfect match (PM) probes on the Affymetrix GeneChip array, where (N) is the number of probes on the array. Each row corresponds to a probe, and each column corresponds to one of the 25 sequence positions. Nucleotides in the sequences are represented by one of the following integers:</td>
</tr>
<tr>
<td></td>
<td>• 0 — None</td>
</tr>
<tr>
<td></td>
<td>• 1 — A</td>
</tr>
<tr>
<td></td>
<td>• 2 — C</td>
</tr>
<tr>
<td></td>
<td>• 3 — G</td>
</tr>
<tr>
<td></td>
<td>• 4 — T</td>
</tr>
</tbody>
</table>

**Note** Probes without sequence information are represented in *SequenceMatrix* as a row containing all 0s.

**Tip** You can use the `int2nt` function to convert the nucleotide sequences in *SequenceMatrix* to letter representation.

*Struct* = `affyprobeseqread(SeqFile, CDFFile, ...'PropertyName', PropertyValue, ...)` calls `affyprobeseqread` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:
Struct = affyprobeseqread(SeqFile, CDFFile, ...'SeqPath', SeqPathValue, ...) lets you specify a path and folder where SeqFile is stored.

Struct = affyprobeseqread(SeqFile, CDFFile, ...'CDFPath', CDFPathValue, ...) lets you specify a path and folder where CDFFile is stored.

Struct = affyprobeseqread(SeqFile, CDFFile, ...'SeqOnly', SeqOnlyValue, ...) controls the return of a structure, Struct, with only one field, SequenceMatrix. Choices are true or false (default).

Examples

1. Read the data from a FASTA file and associated CDF library file, assuming both are located on the MATLAB search path or in the Current Folder.
   
   S1 = affyprobeseqread('HG-U95A_probe_fasta', 'HG_U95A.CDF');

2. Read the data from a tab-separated file and associated CDF structure, assuming the tab-separated file is located in the specified folder and the CDF structure is in your MATLAB Workspace.
   
   S2 = affyprobeseqread('HG-U95A_probe_tab', hgu95aCDFStruct,...
   'seqpath','C:\Affymetrix\SequenceFiles\HGGenome');

3. Access the nucleotide sequences of the first probe set (rows 1 through 20) in the SequenceMatrix field of the S2 structure.
   
   seq = int2nt(S2.SequenceMatrix(1:20,:))

See Also

affygcrcma | affyinvarsetnorm | affyread | celintensityread | int2nt | probelibraryinfo | probesetlink | probesetlookup | probesetplot | probesetvalues

Introduced in R2007a
affyread

Read microarray data from Affymetrix GeneChip file

Syntax

AffyStruct = affyread(File)
AffyStruct = affyread(File, LibraryPath)

Description

AffyStruct = affyread(File) reads an Affymetrix file and creates a MATLAB structure. The affyread function can read Affymetrix EXP, DAT, CEL, CLF, BGP, CDF, and GIN files associated with Affymetrix GeneChip arrays for expression, genotyping (SNP), or resequencing assays. It can read Affymetrix CHP files associated with Affymetrix GeneChip arrays for expression assays only.

AffyStruct = affyread(File, LibraryPath) specifies the path and folder of a CDF or GIN library file.

Input Arguments

File

Character vector or string specifying a file name or a path and file name of one of the following Affymetrix file types associated with Affymetrix GeneChip arrays for expression, genotyping (SNP), or resequencing assays. However, if the file name is for a CHP file, it must be associated with an Affymetrix GeneChip array for an expression assay.

- **EXP** — Data file containing information about experimental conditions and protocols.
- **DAT** — Data file containing raw image data (pixel intensity values).
- **CEL** — Data file containing information about the intensity values of the individual probes.
- **CHP** — Data file containing summary information of the probe sets, including intensity values.
- **CLF** — Cell layout file that maps probe IDs to a location (x- and y-coordinates) in the CEL file.
- **BGP** — Background probe file that lists the probes to use for background correction.
- **CDF** — Library file containing information about which probes belong to which probe set.
- **GIN** — Library file containing information about the probe sets, such as the gene name associated with the probe set.

If you specify only a file name, put that file on the MATLAB search path or in the current folder. If you specify only a file name of a CDF or GIN library file, you can specify the path and folder in the **LibraryPath** input argument.

LibraryPath

Character vector or string specifying the path and folder of a:

- CDF library file associated with **File** when **File** is a CHP file
• CDF library file when `File` is a CDF file
• GIN library file when `File` is a GIN file

**Note** If you do not specify `LibraryPath` when reading a CHP file, `affyread` looks in the current folder for the CDF file. If it does not find the CDF file, it still reads the CHP file. However, it omits the probe set names and types from the return value, `AffyStruct`.

### Output Arguments

**AffyStruct**

MATLAB structure containing information from an Affymetrix data or library file, for expression, genotyping (SNP), or resequencing assay types.

The following tables describe the fields in `AffyStruct` for the different Affymetrix file types.

**EXP, DAT, CEL, CHP, CLF, BGP, CDF, and GIN Files**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>File name.</td>
</tr>
<tr>
<td>DataPath</td>
<td>Path and folder of the file.</td>
</tr>
<tr>
<td>LibPath</td>
<td>Path and folder of the CDF and GIN library files associated with the file you are reading.</td>
</tr>
<tr>
<td>FullPathName</td>
<td>Path and folder of the file.</td>
</tr>
<tr>
<td>ChipType</td>
<td>Name of the Affymetrix GeneChip array (for example, DrosGenome1 or HG-Focus).</td>
</tr>
<tr>
<td>Date or CreateDate</td>
<td>File creation date.</td>
</tr>
</tbody>
</table>
### EXP File

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChipLot</td>
<td>Information about experimental conditions and protocols captured by the Affymetrix software.</td>
</tr>
<tr>
<td>Operator</td>
<td></td>
</tr>
<tr>
<td>SampleType</td>
<td></td>
</tr>
<tr>
<td>SampleDesc</td>
<td></td>
</tr>
<tr>
<td>Project</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td></td>
</tr>
<tr>
<td>Reagents</td>
<td></td>
</tr>
<tr>
<td>ReagentLot</td>
<td></td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
</tr>
<tr>
<td>Station</td>
<td></td>
</tr>
<tr>
<td>Module</td>
<td></td>
</tr>
<tr>
<td>HybridizeDate</td>
<td></td>
</tr>
<tr>
<td>ScanPixelSize</td>
<td></td>
</tr>
<tr>
<td>ScanFilter</td>
<td></td>
</tr>
<tr>
<td>ScanDate</td>
<td></td>
</tr>
<tr>
<td>ScannerID</td>
<td></td>
</tr>
<tr>
<td>NumberOfScans</td>
<td></td>
</tr>
<tr>
<td>ScannerType</td>
<td></td>
</tr>
<tr>
<td>NumProtocolSteps</td>
<td></td>
</tr>
<tr>
<td>ProtocolSteps</td>
<td></td>
</tr>
</tbody>
</table>
**DAT File**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumPixelsPerRow</td>
<td>Number of pixels per row in the image created from the GeneChip array</td>
</tr>
<tr>
<td>NumRows</td>
<td>Number of rows in the image created from the GeneChip array.</td>
</tr>
<tr>
<td>MinData</td>
<td>Minimum intensity value in the image created from the GeneChip array.</td>
</tr>
<tr>
<td>MaxData</td>
<td>Maximum intensity value in the image created from the GeneChip array.</td>
</tr>
<tr>
<td>PixelSize</td>
<td>Size of one pixel in the image created from the GeneChip array.</td>
</tr>
<tr>
<td>CellMargin</td>
<td>Size of gaps between cells in the image created from the GeneChip array.</td>
</tr>
<tr>
<td>ScanSpeed</td>
<td>Speed of the scanner used to create the image.</td>
</tr>
<tr>
<td>ScanDate</td>
<td>Date the scan was performed.</td>
</tr>
<tr>
<td>ScannerID</td>
<td>Name of the scanning device used.</td>
</tr>
<tr>
<td>UpperLeftX</td>
<td>Pixel coordinates of the scanned image.</td>
</tr>
<tr>
<td>UpperLeftY</td>
<td></td>
</tr>
<tr>
<td>UpperRightX</td>
<td></td>
</tr>
<tr>
<td>UpperRightY</td>
<td></td>
</tr>
<tr>
<td>LowerLeftX</td>
<td></td>
</tr>
<tr>
<td>LowerLeftY</td>
<td></td>
</tr>
<tr>
<td>LowerRightX</td>
<td></td>
</tr>
<tr>
<td>LowerRightY</td>
<td></td>
</tr>
<tr>
<td>ServerName</td>
<td>Not used.</td>
</tr>
<tr>
<td>Image</td>
<td>A NumRows-by-NumPixelsPerRow image of the scanned GeneChip array.</td>
</tr>
</tbody>
</table>
### CEL File

<table>
<thead>
<tr>
<th><strong>Field</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>FileVersion</td>
<td>Version of the CEL file format.</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Algorithm used in the image-processing step that converts from DAT format to CEL format.</td>
</tr>
<tr>
<td>AlgParams</td>
<td>Character vector containing parameters used by the algorithm in the image-processing step.</td>
</tr>
<tr>
<td>NumAlgParams</td>
<td>Number of parameters in AlgParams.</td>
</tr>
<tr>
<td>CellMargin</td>
<td>Size of gaps between cells in the image created from the GeneChip array, used for computing the intensity values of the cells.</td>
</tr>
<tr>
<td>Rows</td>
<td>Number of rows of probes.</td>
</tr>
<tr>
<td>Cols</td>
<td>Number of columns of probes.</td>
</tr>
<tr>
<td>NumMasked</td>
<td>Number of masked probes, which are not used in subsequent processing.</td>
</tr>
<tr>
<td>NumOutliers</td>
<td>Number of cells identified as outliers (extremely high or extremely low intensity) by the image-processing step.</td>
</tr>
<tr>
<td>NumProbes</td>
<td>Number of probes (Rows * Cols) on the GeneChip array.</td>
</tr>
<tr>
<td>UpperLeftX</td>
<td>Pixel coordinates of the scanned image.</td>
</tr>
<tr>
<td>UpperLeftY</td>
<td></td>
</tr>
<tr>
<td>UpperRightX</td>
<td></td>
</tr>
<tr>
<td>UpperRightY</td>
<td></td>
</tr>
<tr>
<td>LowerLeftX</td>
<td></td>
</tr>
<tr>
<td>LowerLeftY</td>
<td></td>
</tr>
<tr>
<td>LowerRightX</td>
<td></td>
</tr>
<tr>
<td>LowerRightY</td>
<td></td>
</tr>
<tr>
<td>ProbeColumnNames</td>
<td>Cell array containing the eight column names in the Probes field:</td>
</tr>
<tr>
<td></td>
<td>• PosX — x-coordinate of the cell</td>
</tr>
<tr>
<td></td>
<td>• PosY — y-coordinate of the cell</td>
</tr>
<tr>
<td></td>
<td>• Intensity — Intensity value of the cell</td>
</tr>
<tr>
<td></td>
<td>• StdDev — Standard deviation of intensity value</td>
</tr>
<tr>
<td></td>
<td>• Pixels — Number of pixels in the cell</td>
</tr>
<tr>
<td></td>
<td>• Outlier — True/false flag indicating if the cell was marked as an outlier</td>
</tr>
<tr>
<td></td>
<td>• Masked — True/false flag indicating if the cell was masked</td>
</tr>
<tr>
<td></td>
<td>• ProbeType — Integer indicating the probe type (for example, 1 = expression)</td>
</tr>
<tr>
<td>Probes</td>
<td>NumProbes-by-8 array of information about the individual probes, including intensity values. The ProbeColumnNames field contains the column names of this array.</td>
</tr>
</tbody>
</table>
### CHP File

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssayType</td>
<td>Type of assay associated with the GeneChip array (for example, Expression, Genotyping, or Resequencing).</td>
</tr>
<tr>
<td>CellFile</td>
<td>File name of the CEL file from which the CHP file was created.</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Algorithm used to convert from CEL format to CHP format.</td>
</tr>
<tr>
<td>AlgVersion</td>
<td>Version of the algorithm used to create the CHP file.</td>
</tr>
<tr>
<td>NumAlgParams</td>
<td>Number of parameters in AlgParams.</td>
</tr>
<tr>
<td>AlgParams</td>
<td>Character vector containing parameters used in steps required to create the CHP file (for example, background correction).</td>
</tr>
<tr>
<td>NumChipSummary</td>
<td>Number of entries in ChipSummary.</td>
</tr>
<tr>
<td>ChipSummary</td>
<td>Summary information for the GeneChip array, including background average, standard deviation, max, and min.</td>
</tr>
<tr>
<td>BackgroundZones</td>
<td>Structure containing information about the zones used in the background adjustment step.</td>
</tr>
<tr>
<td>Rows</td>
<td>Number of rows of probes.</td>
</tr>
<tr>
<td>Cols</td>
<td>Number of columns of probes.</td>
</tr>
<tr>
<td>NumProbeSets</td>
<td>Number of probe sets on the GeneChip array.</td>
</tr>
<tr>
<td>NumQCProbeSets</td>
<td>Number of QC probe sets on the GeneChip array.</td>
</tr>
<tr>
<td><strong>Field</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ProbeSets</td>
<td>NumProbeSets-by-1 structure array containing information for each expression probe set, including the following fields:</td>
</tr>
<tr>
<td>(Expression GeneChip array)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- <strong>Name</strong> — Name of the probe set.</td>
</tr>
<tr>
<td></td>
<td>- <strong>ProbeSetType</strong> — Type of the probe set.</td>
</tr>
<tr>
<td></td>
<td>- <strong>CompDataExists</strong> — True/false flag indicating if the probe set has additional computed information.</td>
</tr>
<tr>
<td></td>
<td>- <strong>NumPairs</strong> — Number of probe pairs in the probe set.</td>
</tr>
<tr>
<td></td>
<td>- <strong>NumPairsUsed</strong> — Number of probe pairs in the probe set used for calculating the probe set signal (not masked).</td>
</tr>
<tr>
<td></td>
<td>- <strong>Signal</strong> — Summary intensity value for the probe set.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Detection</strong> — Indicator of statistically significant difference between the intensity value of the PM probes and the intensity value of the MM probes in a single probe set (Present, Absent, or Marginal).</td>
</tr>
<tr>
<td></td>
<td>- <strong>DetectionPValue</strong> — P-value for the Detection indicator.</td>
</tr>
<tr>
<td></td>
<td>- <strong>CommonPairs</strong> — When CompDataExists is true, contains the number of common pairs between the experiment and the baseline after the removal of outliers and masked probes.</td>
</tr>
<tr>
<td></td>
<td>- <strong>SignalLogRatio</strong> — When CompDataExists is true, contains the change in signal between the experiment and baseline.</td>
</tr>
<tr>
<td></td>
<td>- <strong>SignalLogRatioLow</strong> — When CompDataExists is true, contains the lowest ratios of probes between the experiment and the baseline.</td>
</tr>
<tr>
<td></td>
<td>- <strong>SignalLogRatioHigh</strong> — When CompDataExists is true, contains the highest ratios of probes between the experiment and the baseline.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Change</strong> — When CompDataExists is true, describes how the probe changes versus a baseline experiment. Choices are Increase, Marginal Increase, No Change, Decrease, or Marginal Decrease.</td>
</tr>
<tr>
<td></td>
<td>- <strong>ChangePValue</strong> — When CompDataExists is true, contains the p-value associated with Change.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ProbeSets</td>
<td>NumProbeSets-by-1 structure array containing information for each genotyping probe set, including the following fields:</td>
</tr>
<tr>
<td>(Genotyping GeneChip array)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Name — Name of the probe set.</td>
</tr>
<tr>
<td></td>
<td>• AlleleCall — Allele that is present for the probe set. Possibilities are AA (homozygous for the major allele), AB (heterozygous for the major and minor allele), BB (homozygous for the minor allele), or NoCall (unable to determine allele).</td>
</tr>
<tr>
<td></td>
<td>• Confidence — Measure of the accuracy of the allele call.</td>
</tr>
<tr>
<td></td>
<td>• RAS1 — Relative Allele Signal 1 for the SNP site, which is calculated using sense probes.</td>
</tr>
<tr>
<td></td>
<td>• RAS2 — Relative Allele Signal 2 for the SNP site, which is calculated using antisense probes.</td>
</tr>
<tr>
<td></td>
<td>• PValueAA — p-value for an AA call.</td>
</tr>
<tr>
<td></td>
<td>• PValueAB — p-value for an AB call.</td>
</tr>
<tr>
<td></td>
<td>• PValueBB — p-value for a BB call.</td>
</tr>
<tr>
<td></td>
<td>• PValueNoCall — p-value for a NoCall call.</td>
</tr>
<tr>
<td>ProbeSets</td>
<td>NumProbeSets-by-1 structure array containing information for each resequencing probe set, including the following fields:</td>
</tr>
<tr>
<td>(Resequencing GeneChip array)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• CalledBases — 1-by-NumProbeSets character vector containing the bases called by the resequencing algorithm. Possible values are a, c, g, t, and n.</td>
</tr>
<tr>
<td></td>
<td>• Scores — 1-by-NumProbeSets array containing the score associated with each base call.</td>
</tr>
</tbody>
</table>
## CLF File

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LibSetName</td>
<td>Name of a collection of related library files for a given chip. There is only one LibSetName for a CLF file. For example, PGF and CLF files intended for use together must have the same LibSetName.</td>
</tr>
<tr>
<td>LibSetVersion</td>
<td>Version of a collection of related library files for a given chip. There is only one LibSetVersion for a CLF file. For example, PGF and CLF files intended for use together must have the same LibSetVersion.</td>
</tr>
<tr>
<td>GUID</td>
<td>Unique identifier for the CLF file.</td>
</tr>
<tr>
<td>CLFFormatVersion</td>
<td>Version of the CLF file format.</td>
</tr>
<tr>
<td>Rows</td>
<td>Number of rows in the CEL file.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>The CLF file is 1 base, which means the first row and column are designated 1,1, not 0,0.</td>
</tr>
<tr>
<td>Cols</td>
<td>Number of columns in the CEL file.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>The CLF file is 1 base, which means the first row and column are designated 1,1, not 0,0.</td>
</tr>
<tr>
<td>StartID</td>
<td>Starting number for the numbering of elements in the CLF file.</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>This information is useful when numbering does not start with 1.</td>
</tr>
<tr>
<td>EndID</td>
<td>Ending number for the numbering of elements in the CLF file.</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>This information is useful when numbering does not start with 1 and/or there are gaps in the numbering.</td>
</tr>
<tr>
<td>Order</td>
<td>Order in which the probe IDs are numbered in the CEL file, either 'row_major' or 'col_major'.</td>
</tr>
<tr>
<td>DataColNames</td>
<td>Names of the columns in the CEL file that contain data.</td>
</tr>
<tr>
<td>Data</td>
<td>If the numbering of elements in the CLF file is sequential, this field contains a function handle that calculates the x- and y- coordinates of each element in the file from the probe ID. If the numbering of elements in the CLF file is not sequential, this field contains a matrix indicating the number value of each element in the file.</td>
</tr>
</tbody>
</table>
## BGP File

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LibSetName</td>
<td>Name of a collection of related library files for a given chip. There is only one <code>LibSetName</code> for a BGP file.</td>
</tr>
<tr>
<td>LibSetVersion</td>
<td>Version of a collection of related library files for a given chip. There is only one <code>LibSetVersion</code> for a BGP file.</td>
</tr>
<tr>
<td>GUID</td>
<td>Unique identifier for a BGP file.</td>
</tr>
<tr>
<td>ExecGUID</td>
<td>Information about the algorithm used to generate the BGP file.</td>
</tr>
<tr>
<td>ExecVersion</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>Structure containing the following fields:</td>
</tr>
<tr>
<td></td>
<td>• <code>probe_id</code> — ID of the probe to use for background correction.</td>
</tr>
<tr>
<td></td>
<td>• <code>probeset_id</code> — ID of the probe set in the PGF file to which the probe belongs.</td>
</tr>
<tr>
<td></td>
<td>• <code>type</code> — Classification information for the probe.</td>
</tr>
<tr>
<td></td>
<td>• <code>gc_count</code> — Combined number of G and C bases in the probe.</td>
</tr>
<tr>
<td></td>
<td>• <code>probe_length</code> — Length of the probe in base pairs.</td>
</tr>
<tr>
<td></td>
<td>• <code>interrogation_position</code> — Interrogation position of the probe. It is typically 13 for 25-mer PM/MM probes.</td>
</tr>
<tr>
<td></td>
<td>• <code>probe_sequence</code> — Sequence of the probe on the array, going in the direction from array surface to solution. For most standard Affymetrix arrays, this direction is from 3' to 5'. For example, for a sense target (st) probe (see the <code>probe_type</code> field), complement the sequence in this field before looking for matches to transcript sequences. For an antisense target (at), reverse this sequence.</td>
</tr>
<tr>
<td></td>
<td>• <code>atom_id</code> — ID of the atom to which the probe belongs.</td>
</tr>
<tr>
<td></td>
<td>• <code>x</code> — Column coordinate of the probe in the CEL file.</td>
</tr>
<tr>
<td></td>
<td>• <code>y</code> — Row coordinate of the probe in the CEL file.</td>
</tr>
<tr>
<td></td>
<td>• <code>probeset_type</code> — Classification information for the probe set, such as control, affx, or spike. This type information can include multiple classifications and can also be nested.</td>
</tr>
<tr>
<td></td>
<td>• <code>probe_type</code> — Classification information for the probe, such as pm (perfect match), mm (mismatch), st (sense target), or at (antisense target). This type information can include multiple classifications and can also be nested.</td>
</tr>
</tbody>
</table>
CDF File

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows</td>
<td>Number of rows of probes.</td>
</tr>
<tr>
<td>Cols</td>
<td>Number of columns of probes.</td>
</tr>
<tr>
<td>NumProbeSets</td>
<td>Number of probe sets on the GeneChip array.</td>
</tr>
<tr>
<td>NumQCProbeSets</td>
<td>Number of QC probe sets on the GeneChip array.</td>
</tr>
<tr>
<td>ProbeSetColumnNames</td>
<td>Cell array containing the six column names in the ProbePairs field in the</td>
</tr>
<tr>
<td></td>
<td>ProbeSets array:</td>
</tr>
<tr>
<td></td>
<td>• GroupNumber — Number identifying the group to which the probe pair</td>
</tr>
<tr>
<td></td>
<td>belongs. For expression arrays, this value is always 1. For</td>
</tr>
<tr>
<td></td>
<td>genotyping arrays, this value is typically 1 (allele A, sense), 2</td>
</tr>
<tr>
<td></td>
<td>(allele B, sense), 3 (allele A, antisense), or 4 (allele B,</td>
</tr>
<tr>
<td></td>
<td>antisense).</td>
</tr>
<tr>
<td></td>
<td>• Direction — Number identifying the direction of the probe pair. 1</td>
</tr>
<tr>
<td></td>
<td>= sense and 2 = antisense.</td>
</tr>
<tr>
<td></td>
<td>• PMPosX — x-coordinate of the perfect match probe.</td>
</tr>
<tr>
<td></td>
<td>• PMPosY — y-coordinate of the perfect match probe.</td>
</tr>
<tr>
<td></td>
<td>• MMPosX — x-coordinate of the mismatch probe.</td>
</tr>
<tr>
<td></td>
<td>• MMPosY — y-coordinate of the mismatch probe.</td>
</tr>
<tr>
<td>ProbeSets</td>
<td>NumProbeSets-by-1 structure array containing information for each</td>
</tr>
<tr>
<td></td>
<td>probe set, including the following fields:</td>
</tr>
<tr>
<td></td>
<td>• Name — Name of the probe set.</td>
</tr>
<tr>
<td></td>
<td>• ProbeSetType — Type of the probe set.</td>
</tr>
<tr>
<td></td>
<td>• CompDataExists — True/false flag indicating if the probe set has</td>
</tr>
<tr>
<td></td>
<td>additional computed information.</td>
</tr>
<tr>
<td></td>
<td>• NumPairs — Number of probe pairs in the probe set.</td>
</tr>
<tr>
<td></td>
<td>• NumQCProbes — Number of QC probes in the probe set.</td>
</tr>
<tr>
<td></td>
<td>• QCType — Type of QC probes.</td>
</tr>
<tr>
<td></td>
<td>• GroupNames — Name of the group to which the probe set belongs.</td>
</tr>
<tr>
<td></td>
<td>For expression arrays, this field contains the name of the probe set.</td>
</tr>
<tr>
<td></td>
<td>For genotyping arrays, this field contains the name of the alleles,</td>
</tr>
<tr>
<td></td>
<td>for example {'A' 'C' 'A' 'C'}.</td>
</tr>
<tr>
<td></td>
<td>• ProbePairs — NumPairs-by-6 array of information about the probe pairs.</td>
</tr>
<tr>
<td></td>
<td>The column names of this array are contained in the</td>
</tr>
<tr>
<td></td>
<td>ProbeSetColumnNames field.</td>
</tr>
</tbody>
</table>
**GIN File**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>GIN file format version.</td>
</tr>
<tr>
<td>ProbeSetName</td>
<td>Probe set ID/name.</td>
</tr>
<tr>
<td>ID</td>
<td>Identifier for the probe set (gene ID).</td>
</tr>
<tr>
<td>Description</td>
<td>Description of the probe set.</td>
</tr>
<tr>
<td>SourceNames</td>
<td>Source or sources of the probe sets.</td>
</tr>
<tr>
<td>SourceURL</td>
<td>Source URL or URLs for the probe sets.</td>
</tr>
<tr>
<td>SourceID</td>
<td>Vector of numbers specifying which SourceNames or SourceURL each probe set is associated with.</td>
</tr>
</tbody>
</table>

**Examples**

**Visualize Microarray Data**

This example shows how to read and visualize microarray data from Affymetrix® GeneChip® file.

This example uses sample data from the *E. coli* Antisense Genome Array. Download the data from Demo_Data_E-coli-antisense.zip. Extract the data files from the DTT archive using the Data Transfer Tool.

You also need to download Ecoli_Asv2.CDF and Ecoli_Asv2.GIN, the library files for the *E. coli* Antisense Genome Array. You may already have these files if you have any Affymetrix GeneChip software installed on your machine. If not, get the library files by downloading and unzipping the *E. coli* Antisense Genome Array zip file.

Read the contents of a CEL file into a MATLAB structure.

```matlab
celStruct = affyread('Ecoli-antisense-121502.CEL');
```

Display a spatial plot of the probe intensities.

```matlab
maimage(celStruct, 'Intensity')
```
Zoom in on a specific region of the plot.

axis([200 340 0 70])
Read the contents of a DAT file into a MATLAB structure. Display the raw image data, and then use the `axis image` command to set the correct aspect ratio.

```matlab
datStruct = affyread('Ecoli-antisense-121502.dat');
imagesc(datStruct.Image)
axis image
```
Zoom in on a specific region of the plot.

axis([1900 2800 160 650])
Read the contents of a CHP file into a MATLAB structure, specifying the location of the associated CDF library file. Then extract information for probe set 3315278.

```matlab
chpStruct = affyread('Ecoli-antisense-121502.chp','C:\LibFiles\');
geneName = probesetlookup(chpStruct,'3315278')
```

geneName =

```matlab
struct with fields:
    Identifier: '3315278'
    ProbeSetName: 'argG_b3172_at'
    CDFIndex: 5213
    GINIndex: 3074
    Description: '/start=3316278 /end=3317621 /direction=+ /description=argininosuccinate synthetase'
    Source: 'NCBI EColi Genome'
    SourceURL: 'http://www.ncbi.nlm.nih.gov/cgi-bin/Entrez/altvik?gi=115&db=g&from=3315278'
```

See Also
affygcma | affyrm | affysnpannotread | affysnpintensitiesplit | agferead |
celintensityread | geoseriesread | gprread | ilmnbsread | probelibraryinfo |
probesetlink | probesetlookup | probesetplot | probesetvalues | sptread

Topics
“Working with Affymetrix® Data”
“Preprocessing Affymetrix® Microarray Data at the Probe Level”
“Analyzing Affymetrix® SNP Arrays for DNA Copy Number Variants”

Introduced before R2006a
affyrma

Perform Robust Multi-array Average (RMA) procedure on Affymetrix microarray probe-level data

Syntax

<Expression> = affyrma(CELFiles, CDFFile)
<Expression> = affyrma(ProbeStructure)

<Expression> = affyrma(CELFiles, CDFFile, ...'CELPath', CELPathValue, ...)
<Expression> = affyrma(CELFiles, CDFFile, ...'CDFPath', CDFPathValue, ...)
<Expression> = affyrma(..., 'Method', MethodValue, ...)
<Expression> = affyrma(..., 'Truncate', TruncateValue, ...)
<Expression> = affyrma(..., 'Median', MedianValue, ...)
<Expression> = affyrma(..., 'Output', OutputValue, ...)
<Expression> = affyrma(..., 'Showplot', ShowplotValue, ...)
<Expression> = affyrma(..., 'Verbose', VerboseValue, ...)

Input Arguments

| CELFiles          | Any of the following:
|-------------------|---------------------------------------------------------------|
|                   | • Character vector or string specifying a single CEL file name.
|                   | • ' * ', which reads all CEL files in the current folder.
|                   | • ' ', which opens the Select CEL Files dialog box from which you
|                   | select the CEL files. From this dialog box, you can press and hold
|                   | Ctrl or Shift while clicking to select multiple CEL files.
|                   | • Cell array of character vectors or string vector containing CEL file
|                   | names.                                                           |

| CDFFile           | Either of the following:
|-------------------|---------------------------------------------------------------|
|                   | • Character vector or string specifying a CDF file name.
|                   | • ' ', which opens the Select CDF File dialog box from which you
|                   | select the CDF file.                                          |

| ProbeStructure    | MATLAB structure containing information from the CEL files, including
|                   | probe intensities, probe indices, and probe set IDs, returned by the
|                   | celintensityread function.                                    |

| CELPathValue      | Character vector or string specifying the path and folder where the files
|                   | specified in CELFiles are stored.                             |

| CDFPathValue      | Character vector or string specifying the path and folder where the file
|                   | specified in CDFFile is stored.                               |

| MethodValue       | Specifies the estimation method for the background adjustment model
|                   | parameters. Choices are 'RMA' (to use estimation method described by
|                   | Bolstad, 2005) or 'MLE' (to estimate the parameters using maximum
<p>|                   | likelihood). Default is 'RMA'.                                |</p>
<table>
<thead>
<tr>
<th><strong>TruncateValue</strong></th>
<th>Specifies the background noise model. Choices are true (use a truncated Gaussian distribution) or false (use a nontruncated Gaussian distribution). Default is true.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MedianValue</strong></td>
<td>Specifies the use of the median of the ranked values instead of the mean for normalization. Choices are true or false (default).</td>
</tr>
<tr>
<td><strong>OutputValue</strong></td>
<td>Specifies the scale of the returned gene expression values. Choices are:</td>
</tr>
<tr>
<td></td>
<td>'log'</td>
</tr>
<tr>
<td></td>
<td>'log2'</td>
</tr>
<tr>
<td></td>
<td>'log10'</td>
</tr>
<tr>
<td></td>
<td>'linear'</td>
</tr>
<tr>
<td></td>
<td>@functionname</td>
</tr>
<tr>
<td></td>
<td>In the last instance, the data is transformed as defined by the function functionname. Default is 'log2'.</td>
</tr>
<tr>
<td><strong>ShowplotValue</strong></td>
<td>Controls the plotting of a histogram showing the distribution of PM probe intensity values (blue) and the convoluted probability distribution function (red), with estimated parameters mu, sigma and alpha. Enter either 'all' (plot a histogram for each column or chip) or specify a subset of columns (chips) by entering the column number, list of numbers, or range of numbers. For example:</td>
</tr>
<tr>
<td></td>
<td>• (...) , 'Showplot', 3, ...) plots the intensity values in column 3.</td>
</tr>
<tr>
<td></td>
<td>• (...) , 'Showplot', [3,5,7], ...) plots the intensity values in columns 3, 5, and 7.</td>
</tr>
<tr>
<td></td>
<td>• (...) , 'Showplot', 3:9, ...) plots the intensity values in columns 3 to 9.</td>
</tr>
<tr>
<td><strong>VerboseValue</strong></td>
<td>Controls the display of the status of the reading of files and RMA processing. Choices are true (default) or false.</td>
</tr>
</tbody>
</table>

### Output Arguments

| **Expression** | DataMatrix object on page 1-532 containing the log₂ based gene expression values that have been background adjusted, normalized, and summarized using the Robust Multi-array Average (RMA) procedure. Each row in Expression corresponds to a gene (probe set), and each column corresponds to an Affymetrix CEL file. |

### Description

*Expression = affyrma(CELFiles, CDFFile)* reads the specified Affymetrix CEL files and the associated CDF library file (created from Affymetrix GeneChip arrays for expression or genotyping assays), processes the probe intensity values using RMA background adjustment, quantile normalization, and summarization procedures, then returns *Expression*, a DataMatrix object on
page 1-532 containing the log, based gene expression values in a matrix, the probe set IDs as row names, and the CEL file names as column names. Note that each row in Expression corresponds to a gene (probe set), and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)

**CELFiles** is a character vector, string, string vector, or cell array of character vectors containing CEL file names. **CDFFile** is a character vector or string specifying a CDF file name. If you set CELFiles to ‘*’, then it reads all CEL files in the current folder. If you set CELFiles to ‘ ’, then it opens the Select CEL Files dialog box from which you select the CEL files.

**Note** For details on the reading of files and RMA processing, see celintensityread, rmabackadj, quantilenorm, and rmasummary.

```matlab
Expression = affyrma(ProbeStructure) uses RMA background adjustment, quantile normalization, and summarization procedures to process the probe intensity values in
ProbeStructure, a MATLAB structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs, returned by the celintensityread function, and returns Expression.

Expression = affyrma(..., 'PropertyName', PropertyValue, ...) calls affyrma with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Expression = affyrma(CELFiles, CDFFile, ...'CELPath', CELPathValue, ...) specifies a path and folder where the files specified by CELFiles are stored.

Expression = affyrma(CELFiles, CDFFile, ...'CDFPath', CDFPathValue, ...) specifies a path and folder where the file specified by CDFFile is stored.

Expression = affyrma(..., 'Method', MethodValue, ...) specifies the estimation method for the background adjustment model parameters. When MethodValue is 'RMA', affyrma implements the estimation method described by Bolstad, 2005. When MethodValue is 'MLE', affyrma estimates the parameters using maximum likelihood. Default is 'RMA'.

Expression = affyrma(..., 'Truncate', TruncateValue, ...) specifies the background noise model used. When TruncateValue is false, affyrma uses nontruncated Gaussian as the background noise model. Default is true.

Expression = affyrma(..., 'Median', MedianValue, ...) specifies the use of the median of the ranked values instead of the mean for normalization. Choices are true or false (default).

Expression = affyrma(..., 'Output', OutputValue, ...) specifies the scale of the returned gene expression values. OutputValue can be:

- 'log'
- 'log2'
- 'log10'
- 'linear'
- @functionname
```
In the last instance, the data is transformed as defined by the function functionname. Default is 'log2'.

Expression = affyrma(..., 'Showplot', ShowplotValue, ...) lets you plot a histogram showing the distribution of PM probe intensity values (blue) and the convoluted probability distribution function (red), with estimated parameters mu, sigma and alpha. When ShowplotValue is 'all', rmabackadj plots a histogram for each column or chip. When ShowplotValue is a number, list of numbers, or range of numbers, rmabackadj plots a histogram for the indicated column number (chip).

For example:
• (...,'Showplot', 3,...) plots the intensity values in column 3.
• (...,'Showplot', [3,5,7],...) plots the intensity values in columns 3, 5, and 7.
• (...,'Showplot', 3:9,...) plots the intensity values in columns 3 to 9.

Expression = affyrma(..., 'Verbose', VerboseValue, ...) controls the display of the status of the reading of files and RMA processing. Choices are true (default) or false.

Examples

The following example assumes that you have the HG_U95Av2.CDF library file stored at D:\Affymetrix\LibFiles\HGGenome, and that your current folder points to a location containing CEL files associated with this CDF library file. In this example, the affyrma function reads all the CEL files in the current folder and a CDF file in a specified folder. It also performs RMA background adjustment, quantile normalization, and summarization procedures on the PM probe intensity values, and returns a DataMatrix object, containing the metadata and processed data.

Expression = affyrma('*', 'HG_U95Av2.CDF', ...,'CDFPath', 'D:\Affymetrix\LibFiles\HGGenome');

References


See Also

affygcma | celintensityread | gcrma | mafdr | mattest | quantilenorm | rmabackadj | rmasummary

1-142
Introduced in R2008b
affysnpannotread

Read Affymetrix Mapping DNA array data from CSV-format annotation file

Syntax

AnnotStruct = affysnpannotread(File, PID)
AnnotStruct = affysnpannotread(File, PID, 'LookUpField', LookUpFieldValue)

Input Arguments

| File       | Character vector or string specifying a file name or a path and file name of an Affymetrix CSV annotation file for a Mapping 10K array set, Mapping 100K array set, or Mapping 500K array set. If you specify only a file name, that file must be on the MATLAB search path or in the current folder. |
| PID        | Character vector, string, string vector, or cell array of character vectors specifying one or more probe set IDs on an Affymetrix mapping array. |
| LookUpFieldValue | Character vector, string, string vector, or cell array of character vectors specifying one or more column headers in an Affymetrix CSV annotation file. Default are the fields shown in the following table. |

Output Arguments

| AnnotStruct | MATLAB structure containing information for one or more probe sets from File, an Affymetrix CSV annotation file. AnnotStruct contains a subset of the fields in File. The fields are described in the table below. |

Description

AnnotStruct = affysnpannotread(File, PID) reads File, an Affymetrix CSV annotation file for a Mapping 10K array set, Mapping 100K array set, or Mapping 500K array set, and returns AnnotStruct, a MATLAB structure containing annotation information for one or more probe sets specified by PID, a character vector, string, string vector, or cell array of character vectors specifying one or more probe set IDs. AnnotStruct contains a subset of the fields in File. The fields are described in the following table.
Structure Created from an Affymetrix CSV Annotation File

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProbeSetIDs</td>
<td>Cell array containing the unique probe set IDs specified by the PID input.</td>
</tr>
<tr>
<td>Chromosome</td>
<td>Cell array containing the chromosome number on which each probe set is located.</td>
</tr>
<tr>
<td>ChromPosition</td>
<td>Cell array containing the SNP genomic position on the chromosome for each probe set.</td>
</tr>
<tr>
<td>Cytoband</td>
<td>Cell array containing the cytogenetic banding region of the chromosome on which each probe set is located.</td>
</tr>
<tr>
<td>Sequence</td>
<td>Cell array containing the sequence of each probe set.</td>
</tr>
<tr>
<td>AlleleA</td>
<td>Cell array containing the base that is allele A for each probe set.</td>
</tr>
<tr>
<td>AlleleB</td>
<td>Cell array containing the base that is allele B for each probe set.</td>
</tr>
<tr>
<td>Accession</td>
<td>Cell array containing the GenBank® accession number for each probe set.</td>
</tr>
<tr>
<td>FragmentLength</td>
<td>Cell array containing the length of each probe set.</td>
</tr>
</tbody>
</table>

AnnotStruct = affysnpannotread(File, PID, 'LookUpField', LookUpFieldValue) returns annotation information from only the field (column) specified by LookUpFieldValue, a character vector, string, string vector, or cell array of character vectors specifying one or more column headers in an Affymetrix CSV annotation file. Default are the fields shown in the previous table.

Note You can download Affymetrix CSV annotation files such as Mapping50K_Xba240.na25.annot.csv from:
https://www.affymetrix.com/support/technical/annotationfilesmain.affx

Examples

The following example assumes that you have the Mapping50K_Xba240.CDF file stored at C:\AffyLibFiles\, and that your current folder points to a location containing the Mapping50K_Xba240.na25.annot.csv annotation file.

1 Use the affyread function to create a structure containing information from the Mapping50K_Xba240.CDF library file.
   cdf = affyread('C:\AffyLibFiles\Mapping50K_Xba240.CDF');

2 Create a variable containing a cell array of the names of the probe sets, which are stored in the Name field of the ProbeSets field of the cdf structure.
   probesetIDs = {cdf.ProbeSets.Name}';

3 Return a structure containing annotation information for all the probe sets in the Mapping50K_Xba240.na25.annot.csv annotation file.
   snpInfo = affysnpannotread('Mapping50K_Xba240.na25.annot.csv', probesetIDs)
   snpInfo =
ProbeSetIDs: {59024x1 cell}
Chromosome: [59024x1 int8]
ChromPosition: [59024x1 double]
Cytoband: {59024x1 cell}
Sequence: {59024x1 cell}
AlleleA: {59024x1 cell}
AlleleB: {59024x1 cell}
Accession: {59024x1 cell}
FragmentLength: [59024x1 double]

See Also
affyread | affysnpintensitysplit

Introduced in R2008b
**affysnpintensitysplit**

Split Affymetrix SNP probe intensity information for alleles A and B

**Syntax**

```plaintext
ProbeStructSplit = affysnpintensitysplit(ProbeStruct)
ProbeStructSplit = affysnpintensitysplit(ProbeStruct, 'Controls', ControlsValue)
```

**Input Arguments**

- **ProbeStruct**: MATLAB structure containing probe intensity information from an Affymetrix Mapping DNA array, such as returned by `celintensityread`.
- **ControlsValue**: Controls the inclusion of control probes in `ProbeStructSplit`. Choices are `true` or `false` (default).

**Output Arguments**

- **ProbeStructSplit**: MATLAB structure containing probe intensity information from an Affymetrix Mapping DNA array, split into information for alleles A and B.

**Description**

`ProbeStructSplit = affysnpintensitysplit(ProbeStruct)` splits `ProbeStruct`, a structure containing probe intensity information from an Affymetrix Mapping DNA array, into `ProbeStructSplit`, a structure containing probe intensity information from an Affymetrix Mapping DNA array, split into information for alleles A and B.

`ProbeStructSplit` contains the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDFName</td>
<td>File name of the Affymetrix CDF library file.</td>
</tr>
<tr>
<td>CELNames</td>
<td>Cell array of names of the Affymetrix CEL files.</td>
</tr>
<tr>
<td>NumChips</td>
<td>Number of CEL files read into the input structure.</td>
</tr>
<tr>
<td>NumProbeSets</td>
<td>Number of probe sets in each CEL file.</td>
</tr>
<tr>
<td>NumProbes</td>
<td>Maximum number of probes for just one allele in each CEL file.</td>
</tr>
<tr>
<td>ProbeSetIDs</td>
<td>Cell array of the probe set IDs from the Affymetrix CDF library file.</td>
</tr>
</tbody>
</table>

*Note* If the number of probes for allele A is not the same as for allele B, the larger number is used.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProbeIndices</td>
<td>Column vector containing probe indexing information for just one allele in each cell file. Probes within a probe set are numbered 0 through N - 1, where N is the number of probes for one allele in the probe set.</td>
</tr>
<tr>
<td>Note ProbeIndices</td>
<td>has the same number of elements as NumProbes.</td>
</tr>
<tr>
<td>PMAIntensities</td>
<td>Matrix containing perfect match (PM) probe intensity values for allele A. Each row corresponds to an allele A probe, and each column corresponds to a CEL file. The rows are ordered the same way as in ProbeIndices, and the columns are ordered the same way as in the CELFiles input argument to the celintensityread function.</td>
</tr>
<tr>
<td>PMBIIntensities</td>
<td>Matrix containing perfect match (PM) probe intensity values for allele B. Each row corresponds to an allele B probe, and each column corresponds to a CEL file. The rows are ordered the same way as in ProbeIndices, and the columns are ordered the same way as in the CELFiles input argument to the celintensityread function.</td>
</tr>
<tr>
<td>MMAIntensities (optional)</td>
<td>Matrix containing mismatch (MM) probe intensity values for allele A. Each row corresponds to an allele A probe, and each column corresponds to a CEL file. The rows are ordered the same way as in ProbeIndices, and the columns are ordered the same way as in the CELFiles input argument to the celintensityread function.</td>
</tr>
<tr>
<td>MMBIntensities (optional)</td>
<td>Matrix containing mismatch (MM) probe intensity values for allele B. Each row corresponds to an allele B probe, and each column corresponds to a CEL file. The rows are ordered the same way as in ProbeIndices, and the columns are ordered the same way as in the CELFiles input argument to the celintensityread function.</td>
</tr>
</tbody>
</table>

ProbeStructSplit = affysnpintensitiesplit(ProbeStruct, 'Controls', ControlsValue) controls the return of control probe intensities. Choices are true or false (default).

Note Control probes sometimes contain information for only one allele. In this case, the value for the corresponding allele (A or B) that is not present is set to NaN.

**Examples**

The following example assumes that your current folder points to a location containing the Mapping50K_Hind240.CDF library file and 18 CEL files associated with this CDF library file. These files are associated with an Affymetrix Mapping DNA array.

1 Use the celintensityread function to read the Mapping50K_Hind240.CDF library file and 18 CEL files associated with it into a MATLAB structure.

```matlab
ps = celintensityread('**','Mapping50K_Hind240.CDF')
```
ps =
    CDFName: 'Mapping50K_Hind240.CDF'
    CELNames: {18x1 cell}
    NumChips: 18
    NumProbeSets: 57299
    NumProbes: 1145780
    ProbeSetIDs: {57299x1 cell}
    ProbeIndices: [1145780x1 uint8]
    GroupNumbers: [1145780x1 uint8]
    PMIntensities: [1145780x18 single]

2  Extract the PM probe intensities for allele A and allele B into another MATLAB structure, without
including intensity information for the control probes.

ps_split = affysnpintensitysplit(ps)

ps_split =
    CDFName: 'Mapping50K_Hind240.CDF'
    CELNames: {18x1 cell}
    NumChips: 18
    NumProbeSets: 57275
    NumProbes: 572750
    ProbeSetIDs: {57275x1 cell}
    ProbeIndices: [572750x1 uint8]
    PMAIntensities: [572750x18 single]
    PMBIntensities: [572750x18 single]

See Also
affyread | affysnpannotread | celintensityread

Introduced in R2008b
**affysnpquartets**

Create table of SNP probe quartet results for Affymetrix probe set

**Syntax**

```
SNPQStruct = affysnpquartets(CELStruct, CDFStruct, PS)
```

**Input Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CELStruct</strong></td>
<td>Structure created by the <code>affyread</code> function from an Affymetrix CEL file, which contains information about the intensity values of the individual probes.</td>
</tr>
<tr>
<td><strong>CDFStruct</strong></td>
<td>Structure created by the <code>affyread</code> function from an Affymetrix CDF library file associated with the CEL file. The CDF library file contains information about which probes belong to which probe set.</td>
</tr>
<tr>
<td><strong>PS</strong></td>
<td>Probe set index or the probe set ID/name.</td>
</tr>
</tbody>
</table>

**Output Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SNPQStruct</strong></td>
<td>Structure containing probe quartet results for a specific SNP probe set from the probe-level data in a CEL file and associated CDF library file.</td>
</tr>
</tbody>
</table>

**Description**

`SNPQStruct = affysnpquartets(CELStruct, CDFStruct, PS)` creates **SNPQStruct**, a structure containing probe quartet results for a specific SNP probe set, specified by **PS**, from the probe-level data in a CEL file and associated CDF library file. **CELStruct** is a structure created by the `affyread` function from an Affymetrix CEL file. **PS** is a probe set index or probe set ID/name from **CDFStruct**, a structure created by the `affyread` function from an Affymetrix CDF library file associated with the CEL file. **SNPQStruct** is a structure containing the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ProbeSet</code></td>
<td>Identifier for the probe set.</td>
</tr>
<tr>
<td><code>AlleleA</code></td>
<td>Character vector specifying the base that is allele A for the probe set.</td>
</tr>
<tr>
<td><code>AlleleB</code></td>
<td>Character vector specifying the base that is allele B for the probe set.</td>
</tr>
<tr>
<td><code>Quartet</code></td>
<td>Structure array containing intensity values for PM (perfect match) and MM (mismatch) probe pairs, including the sense and antisense probes for alleles A and B. Each structure in the array corresponds to a probe pair in the probe set.</td>
</tr>
</tbody>
</table>

**Examples**

The following example uses the `NA06985_Hind_B5_3005533.CEL` file. You can download this and other sample CEL files from:
The NA06985_Hind_B5_3005533.CEL file is included in the 100K_trios.hind.1.zip file.

The following example uses the CDF library file for the Mapping 50K Hind 240 array, which you can download from:

https://www.affymetrix.com/support/technical/byproduct.affx?product=100k

The following example assumes that the NA06985_Hind_B5_3005533.CEL file is stored on the MATLAB search path or in the current folder. It also assumes that the associated CDF library file, Mapping50K_Hind240.cdf, is stored at D:\Affymetrix\LibFiles\.

1 Read the contents of a CEL file into a MATLAB structure.

```
celStruct = affyread('NA06985_Hind_B5_3005533.CEL');
```

2 Read the contents of a CDF file into a MATLAB structure.

```
cdfStruct = affyread('D:\Affymetrix\LibFiles\Mapping50K_Hind240.cdf');
```

3 Create a structure containing SNP probe quartet results for the SNP_A-1684395 probe set.

```
SNPQStruct = affysnpquartets(celStruct,cdfStruct,'SNP_A-1684395')
```

```
SNPQStruct =
    ProbeSet: 'SNP_A-1684395'
    AlleleA: 'A'
    AlleleB: 'G'
    Quartet: [1x5 struct]
```

4 View the intensity values of the first probe pair in the probe set.

```
SNPQStruct.Quartet(1)
```

```
ans =

    A_Sense_PM: 5013
    B_Sense_PM: 1290
    A_Sense_MM: 1485
    B_Sense_MM: 686
    A_Antisense_PM: 3746
    B_Antisense_PM: 1406
    A_Antisense_MM: 1527
    B_Antisense_MM: 958
```

See Also
- affyread
- probesetvalues

Introduced in R2008a
agferead

Read Agilent Feature Extraction Software file

Syntax

\[ AGFEData = agferead(File) \]

Arguments

| File   | Microarray data file generated with the Agilent® Feature Extraction Software. |

Description

\( AGFEData = agferead(File) \) reads files generated with the Feature Extraction Software from Agilent microarray scanners and creates a structure (AGFEData) containing the following fields:

- Header
- Stats
- Columns
- Rows
- Names
- IDs
- Data
- ColumnNames
- TextData
- TextColumnNames

The Feature Extraction Software takes an image from an Agilent microarray scanner and generates raw intensity data for each spot on the plate.

Examples

1. Read in a sample Agilent Feature Extraction Software file. Note that the file fe_sample.txt is not provided with the Bioinformatics Toolbox software.

\[ \text{agfeStruct} = \text{agferead('fe_sample.txt')} \]

2. Plot the median foreground.

\[ \text{maimage}(	ext{agfeStruct},'gMedianSignal'); \]
\[ \text{maboxplot}(	ext{agfeStruct},'gMedianSignal'); \]

See Also

affyread | celintensityread | galread | geoseriesread | geosoftread | gprread | ilmnbsread | imagenered | magetfield | sptread
Introduced before R2006a
align2cigar

Convert aligned sequences to corresponding signatures in CIGAR format

Syntax

[Cigars,Starts] = align2cigar(Alignment,Ref)

Description

[Cigars,Starts] = align2cigar(Alignment,Ref) converts aligned sequences represented in Alignment, a cell array of aligned character vectors, string vector, or character array, into Cigars, a cell array of corresponding CIGAR-formatted character vectors or string vector, using the reference sequence specified by Ref, a character vector or string. It also returns Starts, a vector of integers indicating the start position of each aligned sequence with respect to the ungapped reference sequence.

Input Arguments

Alignment

Cell array of character vector, string vector, or a character array representing aligned sequences. Soft clippings are assumed to be represented by lowercase letters in the aligned sequences. Skipped positions are assumed to be represented by . in the aligned sequences.

Ref

Character vector or string specifying an aligned reference sequence. The length of Ref must equal the number of columns in Alignment.

Output Arguments

Cigars

Cell array of CIGAR-formatted character vectors or string vector corresponding to each aligned sequence in Alignment.

Starts

Vector of integers indicating the start position of each aligned sequence with respect to the ungapped reference sequence.

Examples

Convert aligned sequences to CIGAR strings

This example shows how to convert aligned strings to CIGAR strings
Create a cell array of aligned strings, create a string specifying a reference sequence, and then convert the alignment to CIGAR strings:

```matlab
aln = ['ACG-ATGC'; 'ACGT-TGC'; '  GTAT-C']
aln = 3x8 char array
   'ACG-ATGC'
   'ACGT-TGC'
   '  GTAT-C'

ref =  'ACGTATGC';
[cigar, start] = align2cigar(aln, ref)
cigar = 1x3 cell
   {'3=1D4='}    {'4=1D3='}    {'4=1D1='}
start = 1x3
   1     1     3
```

**See Also**
BioMap | cigar2align | getAlignment | getBaseCoverage | getCompactAlignment | multialign

**Topics**
“Manage Sequence Read Data in Objects”

**External Websites**
Sequence Read Archive
SAM format specification

**Introduced in R2010b**
**allshortestpaths (biograph)**

Find all shortest paths in biograph object

**Syntax**

\[
\text{dist} = \text{allshortestpaths}(\text{BGObj})
\]

\[
\text{dist} = \text{allshortestpaths}(\text{BGObj}, \ldots \text{'Directed'}, \text{DirectedValue}, \ldots)
\]

\[
\text{dist} = \text{allshortestpaths}(\text{BGObj}, \ldots \text{'Weights'}, \text{WeightsValue}, \ldots)
\]

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BGObj</strong></td>
<td>Biograph object created by biograph (object constructor).</td>
</tr>
<tr>
<td><strong>Directed</strong></td>
<td>Property that indicates whether the graph is directed or undirected. Enter false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.</td>
</tr>
<tr>
<td><strong>Weights</strong></td>
<td>Column vector that specifies custom weights for the edges in the N-by-N adjacency matrix extracted from a biograph object, <strong>BGObj</strong>. It must have one entry for every nonzero value (edge) in the matrix. The order of the custom weights in the vector must match the order of the nonzero values in the matrix when it is traversed column-wise. This property lets you use zero-valued weights. By default, allshortestpaths gets weight information from the nonzero entries in the matrix.</td>
</tr>
</tbody>
</table>

**Description**

**Tip** For introductory information on graph theory functions, see “Graph Theory Functions”.

\[
\text{dist} = \text{allshortestpaths}(\text{BGObj})
\]

finds the shortest paths between every pair of nodes in a graph represented by an N-by-N adjacency matrix extracted from a biograph object, **BGObj**, using Johnson’s algorithm. Nonzero entries in the matrix represent the weights of the edges.

Output **dist** is an N-by-N matrix where **dist**(S, T) is the distance of the shortest path from source node **S** to target node **T**. Elements in the diagonal of this matrix are always 0, indicating the source node and target node are the same. A 0 not in the diagonal indicates that the distance between the source node and target node is 0. An Inf indicates there is no path between the source node and the target node.

Johnson’s algorithm has a time complexity of \(O(N^2\log(N)+N^2E)\), where N and E are the number of nodes and edges respectively.

\[\ldots\] = allshortestpaths (BGObj, 'PropertyName', PropertyValue, ...) calls allshortestpaths with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each **PropertyName** must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:
[dist] = allshortestpaths(BGObj, ...'Directed', DirectedValue, ...) indicates whether the graph is directed or undirected. Set DirectedValue to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.

[dist] = allshortestpaths(BGObj, ...'Weights', WeightsValue, ...) lets you specify custom weights for the edges. WeightsValue is a column vector having one entry for every nonzero value (edge) in the N-by-N adjacency matrix extracted from a biograph object, BGObj. The order of the custom weights in the vector must match the order of the nonzero values in the N-by-N adjacency matrix when it is traversed column-wise. This property lets you use zero-valued weights. By default, allshortestpaths gets weight information from the nonzero entries in the N-by-N adjacency matrix.

References


See Also

biograph | conncomp | graphallshortestpaths | isdag | isomorphism | isspantree | maxflow | minspantree | shortestpath | topoorder | traverse

Topics

biograph object on page 1-185

Introduced in R2006b
aminolookup

Find amino acid codes, integers, abbreviations, names, and codons

Syntax

aminolookup
aminolookup(SeqAA)
aminolookup('Code', CodeValue)
aminolookup('Integer', IntegerValue)
aminolookup('Abbreviation', AbbreviationValue)
aminolookup('Name', NameValue)

Arguments

<table>
<thead>
<tr>
<th>SeqAA</th>
<th>Character vector or string containing single-letter codes or three-letter abbreviations representing an amino acid sequence. For valid codes and abbreviations, see the table Amino Acid Lookup.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeValue</td>
<td>Character vector or string specifying a single-letter code representing an amino acid. For valid single-letter codes, see the table Amino Acid Lookup.</td>
</tr>
<tr>
<td>IntegerValue</td>
<td>Single integer representing an amino acid. For valid integers, see the table Amino Acid Lookup.</td>
</tr>
<tr>
<td>AbbreviationValue</td>
<td>Character vector or string specifying a three-letter abbreviation representing an amino acid. For valid three-letter abbreviations, see the table Amino Acid Lookup.</td>
</tr>
<tr>
<td>NameValue</td>
<td>Character vector or string specifying an amino acid name. For valid amino acid names, see the table Amino Acid Lookup.</td>
</tr>
</tbody>
</table>

Description

aminolookup displays a table of amino acid codes, integers, abbreviations, names, and codons.
Amino Acid Lookup

<table>
<thead>
<tr>
<th>Code</th>
<th>Integer</th>
<th>Abbreviation</th>
<th>Amino Acid Name</th>
<th>Codons</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>Ala</td>
<td>Alanine</td>
<td>GCU GCC GCA GCG</td>
</tr>
<tr>
<td>R</td>
<td>2</td>
<td>Arg</td>
<td>Arginine</td>
<td>CGU CGC CGA CGG AGA AGG</td>
</tr>
<tr>
<td>N</td>
<td>3</td>
<td>Asn</td>
<td>Asparagine</td>
<td>AAU AAC</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>Asp</td>
<td>Aspartic acid</td>
<td>GAU GAC</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>Cys</td>
<td>Cysteine</td>
<td>UGU UGC</td>
</tr>
<tr>
<td>Q</td>
<td>6</td>
<td>Gln</td>
<td>Glutamine</td>
<td>CAA CAG</td>
</tr>
<tr>
<td>E</td>
<td>7</td>
<td>Glu</td>
<td>Glutamic acid</td>
<td>GAA GAG</td>
</tr>
<tr>
<td>G</td>
<td>8</td>
<td>Gly</td>
<td>Glycine</td>
<td>GGU GGC GGA GGG</td>
</tr>
<tr>
<td>H</td>
<td>9</td>
<td>His</td>
<td>Histidine</td>
<td>CAU CAC</td>
</tr>
<tr>
<td>I</td>
<td>10</td>
<td>Ile</td>
<td>Isoleucine</td>
<td>AUU AUC AUA</td>
</tr>
<tr>
<td>L</td>
<td>11</td>
<td>Leu</td>
<td>Leucine</td>
<td>UUA UUG CUU CUC CUA CUG</td>
</tr>
<tr>
<td>K</td>
<td>12</td>
<td>Lys</td>
<td>Lysine</td>
<td>AAA AAG</td>
</tr>
<tr>
<td>M</td>
<td>13</td>
<td>Met</td>
<td>Methionine</td>
<td>AUG</td>
</tr>
<tr>
<td>F</td>
<td>14</td>
<td>Phe</td>
<td>Phenylalanine</td>
<td>UUU UUC</td>
</tr>
<tr>
<td>P</td>
<td>15</td>
<td>Pro</td>
<td>Proline</td>
<td>CCU CCC CCA CCG</td>
</tr>
<tr>
<td>S</td>
<td>16</td>
<td>Ser</td>
<td>Serine</td>
<td>UCU UCC UCA UCG AGU AGC</td>
</tr>
<tr>
<td>T</td>
<td>17</td>
<td>Thr</td>
<td>Threonine</td>
<td>ACU ACC ACA ACG</td>
</tr>
<tr>
<td>W</td>
<td>18</td>
<td>Trp</td>
<td>Tryptophan</td>
<td>UGG</td>
</tr>
<tr>
<td>Y</td>
<td>19</td>
<td>Tyr</td>
<td>Tyrosine</td>
<td>UAU UAC</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
<td>Val</td>
<td>Valine</td>
<td>GUU GUC GUA GUG</td>
</tr>
<tr>
<td>B</td>
<td>21</td>
<td>Asx</td>
<td>Asparagine or</td>
<td>AAU AAC GAU GAC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Aspartic acid</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Aspartate)</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>22</td>
<td>Glx</td>
<td>Glutamine or</td>
<td>CAA CAG GAA GAG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Glutamic acid</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Glutamate)</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>23</td>
<td>Xaa</td>
<td>Any amino acid</td>
<td>All codons</td>
</tr>
<tr>
<td>*</td>
<td>24</td>
<td>END</td>
<td>Termination codon</td>
<td>UAA UAG UGA</td>
</tr>
<tr>
<td>-</td>
<td>25</td>
<td>GAP</td>
<td>Gap of unknown</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>length</td>
<td></td>
</tr>
</tbody>
</table>

`aminolookup(SeqAA)` converts between single-letter codes and three-letter abbreviations for an amino acid sequence. If the input is a character vector or string of single-letter codes, then the output
is a character vector of three-letter abbreviations. If the input is a character vector or string of three-letter abbreviations, then the output is a character vector of the corresponding single-letter codes.

If you enter one of the ambiguous single-letter codes B, Z, or X, this function displays the corresponding abbreviation for the ambiguous amino acid character.

aminolookup('abc')
ans =
AlaAsxCys

aminolookup('Code', CodeValue) displays the corresponding amino acid three-letter abbreviation and name.

aminolookup('Integer', IntegerValue) displays the corresponding amino acid single-letter code, three-letter abbreviation, and name.

aminolookup('Abbreviation', AbbreviationValue) displays the corresponding amino acid single-letter code and name.

aminolookup('Name', NameValue) displays the corresponding amino acid single-letter code and three-letter abbreviation.

**Examples**

**Convert an amino acid sequence to single-letter or three-letter abbreviations**

Convert an amino acid sequence in single-letter codes to the corresponding three-letter abbreviations.

aminolookup('MWKQAEDIRDIYDF')
ans =
'MetTrpLysGlnAlaGluAspIleArgAspIleTyrAspPhe'

Convert an amino acid sequence in three-letter abbreviations to the corresponding single-letter codes.

aminolookup('MetTrpLysGlnAlaGluAspIleArgAspIleTyrAspPhe')
ans =
'MWKQAEDIRDIYDF'

Display the three-letter abbreviation and name for the amino acid corresponding to the single-letter code R.

aminolookup('Code', 'R')
ans =
'Arg  Arginine'

Display the single-letter code, three-letter abbreviation, and name for the amino acid corresponding to the integer 1.

aminolookup('Integer', 1)
ans =
'  A    Ala    Alanine

Display the single-letter code and name for the amino acid corresponding to the three-letter abbreviation asn.
aminolookup('Abbreviation', 'asn')
ans =
'    N    Asparagine

Display the single-letter code and three-letter abbreviation for the amino acid proline.
aminolookup('Name', 'proline')
ans =
'P    Pro

See Also
aa2int | aa2nt | aacount | geneticcode | int2aa | isotopicdist | nt2aa | revgeneticcode

Introduced before R2006a
atomiccomp

Calculate atomic composition of protein

Syntax

\[ NumberAtoms = \text{atomiccomp}(\text{SeqAA}) \]

Arguments

| SeqAA | Amino acid sequence. Enter a character vector or vector of integers from the table Mapping Amino Acid Letter Codes to Integers. You can also enter a structure with the field Sequence. |

Description

\[ NumberAtoms = \text{atomiccomp}(\text{SeqAA}) \] counts the type and number of atoms in an amino acid sequence (\text{SeqAA}) and returns the counts in a 1-by-1 structure (\text{NumberAtoms}) with fields C, H, N, O, and S.

Examples

Calculate the atomic composition of a protein

Retrieve an amino acid sequence from the NCBI GenPept database.

\[
\text{rhodopsin} = \text{getgenpept('NP_000530')}
\]

\[
\text{rhodopsin} =
\]

\[
\text{struct with fields:}
\]

| LocusName: 'NP_000530' |
| LocusSequenceLength: '348' |
| LocusNumberOfStrands: '' |
| LocusTopology: 'linear' |
| LocusMoleculeType: '' |
| LocusGenBankDivision: 'PRI' |
| LocusModificationDate: '07-AUG-2015' |
| Definition: 'rhodopsin [Homo sapiens].' |
| Accession: 'NP_000530' |
| Version: 'NP_000530.1' |
| GI: '4506527' |
| Project: [] |
| DBLink: 'DBSOURCE REFSEQ: accession NM_000539.3' |
| Keywords: 'RefSeq.' |
| Segment: [] |
| Source: 'Homo sapiens human' |
| SourceOrganism: [4×65 char] |
Count the atoms in the sequence.

\[
\text{rhodopsinAC} = \text{atomiccomp(rhodopsin)}
\]

\[
\text{rhodopsinAC} = \\
\text{struct with fields:}
\]

- C: 1814
- H: 2725
- N: 423
- O: 477
- S: 25

Retrieve the number of carbon atoms.

\[
\text{rhodopsinAC.C}
\]

\[
\text{ans} = \\
1814
\]

**See Also**

aadcount | molweight | proteinplot

**Introduced before R2006a**
**bamindexread**

Read BAM Index, BAI, file

**Syntax**

```plaintext
Index = bamindexread(File)
```

**Description**

`Index = bamindexread(File)` reads `File`, a BAI file, and returns `Index`, a MATLAB structure that specifies the offsets into the compressed Binary Sequence Alignment/Map (BAM) file and decompressed data block for each reference sequence and range of positions (bins) on each reference sequence.

**Input Arguments**

`File`

Character vector or string specifying a file name, or a path and a file name, of a BAM file or a BAI file. If `File` is a BAM file, `bamindexread` reads the corresponding BAI file, that is, the BAI file with the same root name and stored in the same folder as the BAM file. If you specify only a file name, that file must be on the MATLAB search path or in the Current Folder.

**Default:**

**Output Arguments**

`Index`

MATLAB array of structures that specifies the offsets into the compressed Binary Sequence Alignment/Map (BAM) file and decompressed data block for each reference sequence and range of positions (bins) on the reference sequence. `Index` contains the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>Name of the BAM file or BAI file used to create the <code>Index</code> array of structures.</td>
</tr>
</tbody>
</table>
Field | Description
--- | ---
Index | A 1-by-\(N\) array of structures, where \(N\) is the number of reference sequences in the corresponding BAM file. Each structure contains the following fields:
- BinID — Array of bin IDs for one reference sequence.
- BGZFOffsetStart — Offset in the BAM file to the start of the first BGZF block where alignment records associated with the corresponding BinID are stored.
- BGZFOffsetEnd — Offset in the BAM file to the start of the last BGZF block where alignment records associated with the corresponding BinID are stored.
- DataOffsetStart — Offset in the decompressed data block to the start of where alignment records associated with the corresponding BinID are stored.
- DataOffsetEnd — Offset in the decompressed data block to the end of where alignment records associated with the corresponding BinID are stored.
- LinearBGZFOffset — Offset in the BAM file to the first alignment in the corresponding 16384 bp interval.
- LinearDataOffset — Offset in the decompressed data file to the first alignment in the corresponding 16384 bp interval.

Examples

**Generate an index structure from a BAM file**

This example shows how to generate an index structure from a BAM index file.

```matlab
ind = bamindexread('ex1.bam')
```

```
ind = struct with fields:
    Filename: 'ex1.bam.bai'
    Index: [1x2 struct]
```

**See Also**

baminfo | bamread

**Topics**

“Manage Sequence Read Data in Objects”

**External Websites**

Sequence Read Archive
SAM format specification

**Introduced in R2010b**
**baminfo**

Return information about BAM file

**Syntax**

InfoStruct = baminfo(File)
InfoStruct = baminfo(File,Name,Value)

**Description**

InfoStruct = baminfo(File) returns a MATLAB structure containing summary information about a BAM-formatted file.

InfoStruct = baminfo(File,Name,Value) returns a MATLAB structure with additional options specified by one or more Name,Value pair arguments.

**Input Arguments**

**File**

Character vector or string specifying a file name or path and file name of a BAM-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the Current Folder.

Default:

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

**ScanDictionary**

Logical that controls the scanning of the BAM-formatted file to determine the reference names and the number of reads aligned to each reference. If true, the ScannedDictionary and ScannedDictionaryCount fields contain this information.

Default: false

**NumOfReads**

Logical that controls the scanning of a BAM-formatted file to determine the number of alignment records in the file. If true, the NumReads field contains this information.

Default: false
## Output Arguments

**InfoStruct**

MATLAB structure containing summary information about a BAM-formatted file. The structure contains these fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>Name of the BAM-formatted file.</td>
</tr>
<tr>
<td>FilePath</td>
<td>Path to the file.</td>
</tr>
<tr>
<td>FileSize</td>
<td>Size of the file in bytes.</td>
</tr>
<tr>
<td>FileModDate</td>
<td>Modification date of the file.</td>
</tr>
<tr>
<td>Header**</td>
<td>Structure containing the file format version, sort order, and group order.</td>
</tr>
<tr>
<td>ReadGroup**</td>
<td>Structure containing the:</td>
</tr>
<tr>
<td></td>
<td>• Read group identifier</td>
</tr>
<tr>
<td></td>
<td>• Sample</td>
</tr>
<tr>
<td></td>
<td>• Library</td>
</tr>
<tr>
<td></td>
<td>• Description</td>
</tr>
<tr>
<td></td>
<td>• Platform unit</td>
</tr>
<tr>
<td></td>
<td>• Predicted median insert size</td>
</tr>
<tr>
<td></td>
<td>• Sequencing center</td>
</tr>
<tr>
<td></td>
<td>• Date</td>
</tr>
<tr>
<td></td>
<td>• Platform</td>
</tr>
<tr>
<td>SequenceDictionary**</td>
<td>Structure containing the:</td>
</tr>
<tr>
<td></td>
<td>• Sequence name</td>
</tr>
<tr>
<td></td>
<td>• Sequence length</td>
</tr>
<tr>
<td></td>
<td>• Genome assembly identifier</td>
</tr>
<tr>
<td></td>
<td>• MD5 checksum of sequence</td>
</tr>
<tr>
<td></td>
<td>• URI of sequence</td>
</tr>
<tr>
<td></td>
<td>• Species</td>
</tr>
<tr>
<td>Program**</td>
<td>Structure containing the:</td>
</tr>
<tr>
<td></td>
<td>• Program name</td>
</tr>
<tr>
<td></td>
<td>• Version</td>
</tr>
<tr>
<td></td>
<td>• Command line</td>
</tr>
<tr>
<td>NumReads</td>
<td>Number of reference sequences in the BAM-formatted file.</td>
</tr>
<tr>
<td>ScannedDictionary*</td>
<td>Cell array of character vectors specifying the names</td>
</tr>
<tr>
<td></td>
<td>of the reference sequences in the BAM-formatted file.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScannedDictionaryCount*</td>
<td>Cell array specifying the number of reads aligned to each reference sequence.</td>
</tr>
</tbody>
</table>

* — The `ScannedDictionary` and `ScannedDictionaryCount` fields are empty if you do not set the `ScanDictionary` name-value pair argument to `true`.

** — These structures and their fields appear in the output structure only if they are in the BAM file. The information in these structures depends on the information in the BAM file.

### Examples

#### Retrieve information about a BAM file

This example shows how to retrieve information about the `ex1.bam` file included with the Bioinformatics Toolbox™.

```matlab
info = baminfo('ex1.bam','ScanDictionary',true,'numofreads',true)
```

```
info = struct with fields:
    Filename: 'ex1.bam'
    FilePath: 'B:\matlab\toolbox\bioinfo\bioinfodata'
    FileSize: 126692
    FileModDate: '07-May-2010 16:12:05'
    Header: [1x1 struct]
    ReadGroup: [1x2 struct]
    SequenceDictionary: [1x2 struct]
    NumReads: 3307
    ScannedDictionary: {2x1 cell}
    ScannedDictionaryCount: [2x1 uint64]
```

List the number of references found in the BAM file.

```matlab
numel(info.ScannedDictionary)
```

```
an = 2
```

Alternatively, you can use the available header information from a BAM file to find out the number of references, thus avoiding the whole traversal of the source file.

```matlab
info = baminfo('ex1.bam');
NRefs = numel(info.SequenceDictionary)
```

```
NRefs = 2
```

### Tips

Use `baminfo` to investigate the size and content of a BAM-formatted file, including reference sequence names, before using the `bamread` function to read the file contents into a MATLAB structure.

### See Also

`bamindexread` | `bamread`
Topics
"Manage Sequence Read Data in Objects"

External Websites
Sequence Read Archive
SAM format specification

Introduced in R2010b
bamread

Read data from BAM file

Syntax

BAMStruct = bamread(File,RefSeq,Range)
[BAMStruct,HeaderStruct] = bamread(File,RefSeq,Range)
... = bamread(File,RefSeq,Range,Name,Value)

Description

BAMStruct = bamread(File,RefSeq,Range) reads the alignment records in File, a BAM-formatted file, that align to RefSeq, a reference sequence, in the range specified by Range. It returns the alignment data in BAMStruct, a MATLAB array of structures.

[BAMStruct,HeaderStruct] = bamread(File,RefSeq,Range) also returns the header information in HeaderStruct, a MATLAB structure.

... = bamread(File,RefSeq,Range,Name,Value) reads the alignment records with additional options specified by one or more Name,Value pair arguments.

Input Arguments

File

Character vector or string specifying a file name or path and file name of a BAM-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the Current Folder.

Note  The function requires the BAM file to be ordered, except when returning reads that are not mapped to any reference.

Default:

RefSeq

Either of the following:

• Character vector or string specifying the name of a reference sequence in the BAM file.
• Positive integer specifying the index of a reference sequence in the BAM file. This number is also the index of the reference sequence in the Reference field of the InfoStruct structure returned by baminfo.

Default:

Range

Two-element vector specifying the begin and end range positions on the reference sequence, RefSeq. Both values must be positive, and are one-based. The second value must be ≥ to the first value.
Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Full

Controls the return of only alignment records that are fully contained within the range specified by Range. Choices are true or false (default).

Default: false

Tags

Controls the reading of the optional tags in addition to the first 11 fields for each alignment in the BAM-formatted file. Choices are true (default) or false.

Default: true

ToFile

Character vector or string specifying a nonexisting file name or a path and file name for saving the alignment records in the specified range of a specific reference sequence. The ToFile name-value pair argument creates a SAM-formatted file. If you specify only a file name, the file is saved to the MATLAB Current Folder.

The SAM-formatted file is always one-based, even if you set the ZeroBased name-value pair argument to true. You can use the SAM-formatted file as input when creating a BioMap object.

Default:

ZeroBased

Logical specifying whether bamread uses zero-based indexing when reading a file. The logical controls the return of zero-based or one-based positions in the Position and MatePosition fields in BAMStruct. Choices are true or false (default), which returns one-based positions.

This name-value pair argument affects the Position and MatePosition fields of BAMStruct. It does not affect the Range input argument or the SAM file created when using the ToFile name-value pair argument. SAM files are always one-based.

Caution If you plan to use the BAMStruct output argument to construct a BioMap object, make sure the ZeroBased name-value pair argument is false.

Default: false
## Output Arguments

**BAMStruct**

An $N$-by-1 array of structures containing sequence alignment and mapping information from a BAMformatted file, where $N$ is the number of alignment records stored in the specified range. Each structure contains the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryName</td>
<td>Name of the read sequence (if unpaired) or the name of sequence pair (if paired).</td>
</tr>
<tr>
<td>Flag</td>
<td>Integer indicating the bit-wise information that specifies the status of each of 11 flags described by the SAM format specification.</td>
</tr>
<tr>
<td></td>
<td><strong>Tip</strong> You can use the <code>bitget</code> function to determine the status of a specific SAM flag.</td>
</tr>
<tr>
<td>ReferenceIndex</td>
<td>Index of the reference sequence.</td>
</tr>
<tr>
<td></td>
<td><strong>Tip</strong> To convert this index to a reference name, see the Reference field in the <code>HeaderStruct</code> output argument</td>
</tr>
<tr>
<td>Position</td>
<td>Position of the forward reference sequence where the leftmost base of the alignment of the read sequence starts. This position is zero-based or one-based, depending on the <code>ZeroBased</code> name-value pair argument.</td>
</tr>
<tr>
<td>MappingQuality</td>
<td>Integer specifying the mapping quality score for the read sequence.</td>
</tr>
<tr>
<td>CigarString</td>
<td>CIGAR-formatted string representing how the read sequence aligns with the reference sequence.</td>
</tr>
<tr>
<td>MateReferenceIndex</td>
<td>Index of the reference sequence associated with the mate. If there is no mate, then this value is 0.</td>
</tr>
<tr>
<td>MatePosition</td>
<td>Position of the forward reference sequence where the leftmost base of the alignment of the mate of the read sequence starts. This position is zero-based or one-based, depending on the <code>ZeroBased</code> name-value pair argument.</td>
</tr>
<tr>
<td>InsertSize</td>
<td>The number of base positions between the read sequence and its mate, when both are mapped to the same reference sequence. Otherwise, this value is 0.</td>
</tr>
<tr>
<td>Sequence</td>
<td>Character vector containing the letter representations of the read sequence. It is the reverse complement if the read sequence aligns to the reverse strand of the reference sequence.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Quality</td>
<td>Character vector containing the ASCII representation of the per-base quality score for the read sequence. The quality score is reversed if the read sequence aligns to the reverse strand of the reference sequence.</td>
</tr>
<tr>
<td>Tags</td>
<td>List of applicable SAM tags and their values.</td>
</tr>
</tbody>
</table>

**HeaderStruct**

MATLAB structure containing header information for the BAM-formatted file in the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRefs</td>
<td>Number of reference sequences in the BAM-formatted file.</td>
</tr>
</tbody>
</table>
| Reference        | 1-by-NRefs array of structures containing these fields:  
  • Name — Name of the reference sequence.  
  • Length — Length of the reference sequence. |
| Header*          | Structure containing the file format version, sort order, and group order.  |
| SequenceDictionary* | Structure containing the:  
  • Sequence name  
  • Sequence length  
  • Genome assembly identifier  
  • MD5 checksum of sequence  
  • URI of sequence  
  • Species |
| ReadGroup*       | Structure containing the:  
  • Read group identifier  
  • Sample  
  • Library  
  • Description  
  • Platform unit  
  • Predicted median insert size  
  • Sequencing center  
  • Date  
  • Platform |
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program*</td>
<td>Structure containing the:</td>
</tr>
<tr>
<td></td>
<td>• Program name</td>
</tr>
<tr>
<td></td>
<td>• Version</td>
</tr>
<tr>
<td></td>
<td>• Command line</td>
</tr>
</tbody>
</table>

* These structures and their fields appear in the output structure only if they are present in the BAM file. The information in these structures depends on the information present in the BAM file.

## Examples

### Retrieve alignment records that align to reference sequences

Read multiple alignment records from the ex1.bam file that align to two different reference sequences.

```matlab
data1 = bamread('ex1.bam', 'seq1', [100 200])
```

**data1** = 59×1 struct array with fields:
- QueryName
- Flag
- Position
- MappingQuality
- CigarString
- MatePosition
- InsertSize
- Sequence
- Quality
- Tags
- ReferenceIndex
- MateReferenceIndex

```matlab
data2 = bamread('ex1.bam', 'seq2', [100 200])
```

**data2** = 79×1 struct array with fields:
- QueryName
- Flag
- Position
- MappingQuality
- CigarString
- MatePosition
- InsertSize
- Sequence
- Quality
- Tags
- ReferenceIndex
- MateReferenceIndex

Read alignments from the ex1.bam file that are fully contained in the 100 to 200 bp range of the seq1 reference sequence.

```matlab
data3 = bamread('ex1.bam', 'seq1', [100 200], 'full', true)
```
data3=30×1 struct array with fields:
   QueryName
   Flag
   Position
   MappingQuality
   CigarString
   MatePosition
   InsertSize
   Sequence
   Quality
   Tags
   ReferenceIndex
   MateReferenceIndex

Read alignments from the ex1.bam file that align to the 100 to 300 bp range of the seq1 reference sequence. Read the same alignments using zero-based indexing. Compare the position of the 27th record in the two outputs.

data_one = bamread('ex1.bam','seq1', [100 300]);
data_zero = bamread('ex1.bam','seq1', [100 300], 'zerobased', true);
data_one(27).Position
ans = uint32
    135

data_zero(27).Position
ans = uint32
    134

Tips

- The bamread function requires a BAM file.
- Use the baminfo function to investigate the size and content, including reference sequence names, of a BAM-formatted file before using the bamread function to read the file contents into a MATLAB array of structures.
- If your BAM-formatted file is too large to read using available memory, try either of the following:
  - Use a smaller range.
  - Use bamread without specifying outputs, but using the ToFile Name,Value pair arguments to create a SAM-formatted file. You can then use samread with the BlockRead Name,Value pair arguments to read the SAM-formatted file. Or you can pass the SAM-formatted file to the BioIndexedFile constructor function to construct a BioIndexedFile object, which you can use to create a BioMap object.
  - Use the BAMStruct output argument that bamread returns to construct a BioMap object, which lets you explore, access, filter, and manipulate all or a subset of the data, before doing subsequent analyses or viewing the data.
References


See Also

BioIndexedFile | BioMap | bamindexread | baminfo | fastainfo | fastaread | fastawrite | fastqinfo | fastqread | fastqwrite | saminfo | samread | sffinfo | sffread | soapread

Topics

“Manage Sequence Read Data in Objects”
“Work with Next-Generation Sequencing Data”

External Websites

Sequence Read Archive
SAM format specification

Introduced in R2010b
bamsort

Sort BAM files

Syntax

\[
\text{sortedFile} = \text{bamsort}(\text{inFile}) \\
\text{bamsort}(\text{inFile}, \text{outFile})
\]

Description

\[
\text{sortedFile} = \text{bamsort}(\text{inFile}) \text{ sorts a BAM file \text{inFile} and returns the name of the sorted } \\
\text{BAM file \text{sortedFile}. The function sorts the alignment records by the reference sequence name } \\
\text{first, and then by position within the reference.}
\]

\[
\text{bamsort}(\text{inFile}, \text{outFile}) \text{ sorts \text{inFile} and produces a sorted BAM file named \text{outFile}.}
\]

Examples

Sort BAM File

Sort a sample BAM file. The sorted file has the same base name as the input file, but with the extension ".sorted.bam". By default, the sorted file is saved in the current directory.

\[
\text{sortedFile} = \text{bamsort("ex1.bam")}
\]

\[
\text{sortedFile} = \\
\text{"ex1.sorted.bam"}
\]

You can change the name of output file by specifying it as the second input.

\[
\text{bamsort("ex1.bam","sortedEx1.bam")}
\]

\[
\text{ans} = \\
\text{"sortedEx1.bam"}
\]

You can also save the output file to an existing directory by providing the file path information.

\[
\text{mkdir("./OutputEx1BAM");} \\
\text{bamsort("ex1.bam","./OutputEx1BAM/sortedEx1.bam")}
\]

\[
\text{ans} = \\
\text{"./OutputEx1BAM/sortedEx1.bam"}
\]

Input Arguments

\[
\text{inFile — Name of input BAM file to sort} \\
\text{character vector | string}
\]

Name of the input BAM file to sort, specified as a string or character vector. You can specify a file name or a path and file name. The file name must have the extension .bam.
Example: "./InputData/ex1.bam"
Data Types: char | string

**outFile — Name of output BAM file**
character vector | string

Name of the output BAM file, specified as a string or character vector. You can specify a file name or a path and file name. The file name must have the extension .bam. The file is saved in the current directory by default unless you specify the path information. If you specify the file path, the listed directories must exist before you run the function.

Example: "./OutputData/ex1Sorted.bam"
Data Types: char | string

### Output Arguments

**sortedFile — Name of output BAM file**
string

Name of the output BAM file, returned as a string. `sortedFile` has the same base name as `inFile` but with the extension .sorted.bam. The file is saved in the current directory by default.

### See Also

BioMap | baminfo | bamread | samread

### Topics

“Data Import”
“Manage Sequence Read Data in Objects”

### External Websites

Sequence Read Archive
SAM format specification

### Introduced in R2019b
# basecount

Count nucleotides in sequence

## Syntax

```
NTStruct = basecount(SeqNT)
NTStruct = basecount(SeqNT, ...'Ambiguous', AmbiguousValue, ...)
NTStruct = basecount(SeqNT, ...'Gaps', GapsValue, ...)
NTStruct = basecount(SeqNT, ...'Chart', ChartValue, ...)
```

## Input Arguments

<table>
<thead>
<tr>
<th><strong>SeqNT</strong></th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Character vector or string containing codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers</td>
<td></td>
</tr>
<tr>
<td>• Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes</td>
<td></td>
</tr>
<tr>
<td>• MATLAB structure containing a Sequence field that contains a nucleotide sequence, such as returned by fastaread, fastqread, emblread, getembl, genbankread, or getgenbank.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>AmbiguousValue</strong></th>
<th>Character vector or string specifying how to treat ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N). Choices are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 'ignore' (default) — Skips ambiguous characters</td>
<td></td>
</tr>
<tr>
<td>• 'bundle' — Counts ambiguous characters and reports the total count in the Ambiguous field.</td>
<td></td>
</tr>
<tr>
<td>• 'prorate' — Counts ambiguous characters and distributes them proportionately in the appropriate fields. For example, the counts for the character R are distributed evenly between the A and G fields.</td>
<td></td>
</tr>
<tr>
<td>• 'individual' — Counts ambiguous characters and reports them in individual fields.</td>
<td></td>
</tr>
<tr>
<td>• 'warn' — Skips ambiguous characters and displays a warning.</td>
<td></td>
</tr>
</tbody>
</table>

| **GapsValue** | Specifies whether gaps, indicated by a hyphen (-), are counted or ignored. Choices are true or false (default). |

| **ChartValue** | Character vector or string specifying a chart type. Choices are 'pie' or 'bar'. |

## Output Arguments

```
NTStruct 1-by-1 MATLAB structure containing the fields A, C, G, and T.```
Description

\[ NTStruct = basecount(SeqNT) \] counts the number of each type of base in \( SeqNT \), a nucleotide sequence, and returns the counts in \( NTStruct \), a 1-by-1 MATLAB structure containing the fields A, C, G, and T.

- The character U is added to the T field.
- Ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N), and gaps, indicated by a hyphen (-), are ignored by default.
- Unrecognized characters are ignored and cause the following warning message.
  
  Warning: Unknown symbols appear in the sequence. These will be ignored.

\[ NTStruct = basecount(SeqNT, ...'PropertyName', PropertyValue, ...) \] calls \( basecount \) with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each \( PropertyName \) must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

\[ NTStruct = basecount(SeqNT, ...'Ambiguous', AmbiguousValue, ...) \] specifies how to treat ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N). Choices are:

- 'ignore' (default)
- 'bundle'
- 'prorate'
- 'individual'
- 'warn'

\[ NTStruct = basecount(SeqNT, ...'Gaps', GapsValue, ...) \] specifies whether gaps, indicated by a hyphen (-), are counted or ignored. Choices are true or false (default).

\[ NTStruct = basecount(SeqNT, ...'Chart', ChartValue, ...) \] creates a chart showing the relative proportions of the nucleotides. \( ChartValue \) can be 'pie' or 'bar'.

Examples

Count nucleotides in a sequence

Count the bases in a DNA sequence and return the results in a structure.

\[
bases = basecount(\text{'TAGCTGGCAAGCGAGCTTG'})
\]

\[
bases = \text{struct with fields:}
\begin{align*}
A: & \quad 4 \\
C: & \quad 5 \\
G: & \quad 7 \\
T: & \quad 4
\end{align*}
\]

Get the count for adenosine (A) bases.

\[
bases.A
\]

\[
an = 4
\]
Count the bases in a DNA sequence containing ambiguous characters (R, Y, K, M, S, W, B, D, H, V, or N), listing each of them in a separate field.

```plaintext
basecount('ABCDGGCAAGCGAGCTTG','Ambiguous','individual')
ans = struct with fields:
    A: 4
    C: 5
    G: 6
    T: 2
    R: 0
    Y: 0
    K: 0
    M: 0
    S: 0
    W: 0
    B: 1
    D: 1
    H: 0
    V: 0
    N: 0
```

**See Also**

`aaccount` | `baselookup` | `codoncount` | `cpgisland` | `dimercount` | `nmercount` | `ntdensity` | `seqviewer`

**Introduced before R2006a**
baselookup

Find nucleotide codes, integers, names, and complements

Syntax

baselookup
baselookup('Complement', SeqNT)
baselookup('Code', CodeValue)
baselookup('Integer', IntegerValue)
baselookup('Name', NameValue)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeqNT</td>
<td>Nucleotide sequence(s) represented by one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Character vector or string containing single-letter codes from the table</td>
</tr>
<tr>
<td></td>
<td>Nucleotide Lookup</td>
</tr>
<tr>
<td></td>
<td>• Cell array of sequences</td>
</tr>
<tr>
<td></td>
<td>• Two-dimensional character array of sequences</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> If the input is multiple sequences, the complement for each</td>
</tr>
<tr>
<td></td>
<td>sequence is determined independently.</td>
</tr>
<tr>
<td>CodeValue</td>
<td>Nucleotide letter code represented by one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Character vector or string specifying a single-letter code representing</td>
</tr>
<tr>
<td></td>
<td>a nucleotide. For valid single-letter codes, see the table Nucleotide</td>
</tr>
<tr>
<td></td>
<td>Lookup.</td>
</tr>
<tr>
<td></td>
<td>• Cell array of letter codes.</td>
</tr>
<tr>
<td></td>
<td>• Two-dimensional character array of letter codes.</td>
</tr>
<tr>
<td>IntegerValue</td>
<td>Single integer representing a nucleotide. For valid integers, see the</td>
</tr>
<tr>
<td></td>
<td>table Nucleotide Lookup.</td>
</tr>
<tr>
<td>NameValue</td>
<td>Nucleotide name represented by one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Character vector or string specifying a nucleotide name. For valid</td>
</tr>
<tr>
<td></td>
<td>nucleotide names, see the table Nucleotide Lookup.</td>
</tr>
<tr>
<td></td>
<td>• Cell array of names.</td>
</tr>
<tr>
<td></td>
<td>• Two-dimensional character array of names.</td>
</tr>
</tbody>
</table>

Description

baselookup displays a table of nucleotide codes, integers, names, and complements.
### Nucleotide Lookup

<table>
<thead>
<tr>
<th>Code</th>
<th>Integer</th>
<th>Nucleotide Name</th>
<th>Meaning</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>Adenine</td>
<td>A</td>
<td>T</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>Cytosine</td>
<td>C</td>
<td>G</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>Guanine</td>
<td>G</td>
<td>C</td>
</tr>
<tr>
<td>T</td>
<td>4</td>
<td>Thymine</td>
<td>T</td>
<td>A</td>
</tr>
<tr>
<td>U</td>
<td>4</td>
<td>Uracil</td>
<td>U</td>
<td>A</td>
</tr>
<tr>
<td>R</td>
<td>5</td>
<td>Purine</td>
<td>A or G</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>6</td>
<td>Pyrimidine</td>
<td>C or T</td>
<td>R</td>
</tr>
<tr>
<td>K</td>
<td>7</td>
<td>Keto</td>
<td>G or T</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>8</td>
<td>Amino</td>
<td>A or C</td>
<td>K</td>
</tr>
<tr>
<td>S</td>
<td>9</td>
<td>Strong interaction (3 H bonds)</td>
<td>C or G</td>
<td>S</td>
</tr>
<tr>
<td>W</td>
<td>10</td>
<td>Weak interaction (2 H bonds)</td>
<td>A or T</td>
<td>W</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>Not A</td>
<td>C or G or T</td>
<td>V</td>
</tr>
<tr>
<td>D</td>
<td>12</td>
<td>Not C</td>
<td>A or G or T</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>13</td>
<td>Not G</td>
<td>A or C or T</td>
<td>D</td>
</tr>
<tr>
<td>V</td>
<td>14</td>
<td>Not T or U</td>
<td>A or C or G</td>
<td>B</td>
</tr>
<tr>
<td>N, X</td>
<td>15</td>
<td>Any nucleotide</td>
<td>A or C or G or T or U</td>
<td>N</td>
</tr>
<tr>
<td>-</td>
<td>16</td>
<td>Gap of indeterminate length</td>
<td>Gap</td>
<td>-</td>
</tr>
</tbody>
</table>

baselookup('Complement', SeqNT) displays the complementary nucleotide sequence.

baselookup('Code', CodeValue) displays the corresponding meaning and nucleotide name. For ambiguous nucleotide codes (R, Y, K, M, S, W, B, D, H, V, N, and X), the nucleotide name is a descriptive name.

baselookup('Integer', IntegerValue) displays the corresponding letter code, meaning, and nucleotide name.

baselookup('Name', NameValue) displays the corresponding letter code, meaning, and nucleotide name or descriptive name.

### Examples

**Convert a nucleotide sequence to its complementary sequence**

baselookup('Complement', 'TAGCTGRCCAAGCCAAGCGAGCTTN')

```matlab
ans =
'ATCGACYGGTTCCGTTGCTGCTGAAN'
```
Display the meaning and nucleotide name or descriptive name for the nucleotide codes G and Y.

```matlab
baselookup('Code', 'G')
ans =
    'G'    Guanine

baselookup('Code', 'Y')
ans =
    'T|C'   pYrimidine
```

Display the nucleotide letter code, meaning, and nucleotide name or descriptive name for the integers 1 and 7.

```matlab
baselookup('Integer', 1)
ans =
    'A'    A - Adenine

baselookup('Integer', 7)
ans =
    'K'    G|T - Keto
```

Display the corresponding nucleotide letter code, meaning, and name for cytosine and purine.

```matlab
baselookup('Name','cytosine')
ans =
    'C'    C - Cytosine

baselookup('Name','purine')
ans =
    'R'    G|A - puRine
```

**See Also**

- `aa2nt` | `basecount` | `codoncount` | `dimercount` | `geneticcode` | `int2nt` | `nt2aa` | `nt2int` | `revgeneticcode` | `seqviewer`

*Introduced before R2006a*
biograph object

Data structure containing generic interconnected data used to implement directed graph

Description

A biograph object is a data structure containing generic interconnected data used to implement a directed graph. Nodes represent proteins, genes, or any other biological entity, and edges represent interactions, dependences, or any other relationship between the nodes. A biograph object also stores information, such as color properties and text label characteristics, used to create a 2-D visualization of the graph.

You create a biograph object using the object constructor function `biograph`. You can view a graphical representation of a biograph object using the `view` method.

Method Summary

Following are methods of a biograph object:

- `allshortestpaths` (biograph) Find all shortest paths in biograph object
- `conncomp` (biograph) Find strongly or weakly connected components in biograph object
- `dolayout` (biograph) Calculate node positions and edge trajectories
- `get` (biograph) Retrieve information about biograph object
- `getancestors` (biograph) Find ancestors of a node in biograph object
- `getdescendants` (biograph) Find descendants of a node in biograph object
- `getedgesbynameid` (biograph) Get handles to edges in biograph object
- `getmatrix` (biograph) Get connection matrix from biograph object
- `getweightmatrix` (biograph) Get connection matrix with weights from biograph object
- `getnodesbyid` (biograph) Get handles to nodes
- `getrelatives` (biograph) Find relatives of a node in biograph object
- `isdag` (biograph) Test for cycles in biograph object
- `isomorphism` (biograph) Find isomorphism between two biograph objects
- `isspantree` (biograph) Determine if tree created from biograph object is spanning tree
- `maxflow` (biograph) Calculate maximum flow in biograph object
- `minspantree` (biograph) Find minimal spanning tree in biograph object
- `set` (biograph) Set property of biograph object
- `shortestpath` (biograph) Solve shortest path problem in biograph object
- `topoorder` (biograph) Perform topological sort of directed acyclic graph extracted from biograph object
- `traverse` (biograph) Traverse biograph object by following adjacent nodes
- `view` (biograph) Draw figure from biograph object

Following are methods of a node object:
getancestors (biograph)  Find ancestors of a node in biograph object
getdescendants (biograph)  Find descendants of a node in biograph object
getrelatives (biograph)  Find relatives of a node in biograph object

**Property Summary**

A biograph object contains two kinds of objects, node objects and edge objects, that have their own properties. For a list of the properties of node objects and edge objects, see the following tables.
## Properties of a Biograph Object

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Character vector to identify the biograph object. Default is <code>'</code>.</td>
</tr>
<tr>
<td>Label</td>
<td>Character vector to label the biograph object. Default is <code>'</code>.</td>
</tr>
<tr>
<td>Description</td>
<td>Character vector that describes the biograph object. Default is <code>'</code>.</td>
</tr>
<tr>
<td>LayoutType</td>
<td>Character vector that specifies the algorithm for the layout engine. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'hierarchical' (default) — Uses a topological order of the graph to assign levels, and then arranges the nodes from top to bottom, while minimizing crossing edges.</td>
</tr>
<tr>
<td></td>
<td>• 'radial' — Uses a topological order of the graph to assign levels, and then arranges the nodes from inside to outside of the circle, while minimizing crossing edges.</td>
</tr>
<tr>
<td></td>
<td>• 'equilibrium' — Calculates layout by minimizing the energy in a dynamic spring system.</td>
</tr>
<tr>
<td>EdgeType</td>
<td>Character vector that specifies how edges display. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'straight'</td>
</tr>
<tr>
<td></td>
<td>• 'curved' (default)</td>
</tr>
<tr>
<td></td>
<td>• 'segmented'</td>
</tr>
<tr>
<td>Note</td>
<td>Curved or segmented edges occur only when necessary to avoid obstruction by nodes. Biograph objects with LayoutType equal to 'equilibrium' or 'radial' cannot produce curved or segmented edges.</td>
</tr>
<tr>
<td>Scale</td>
<td>Positive number that post-scales the node coordinates. Default is 1.</td>
</tr>
<tr>
<td>LayoutScale</td>
<td>Positive number that scales the size of the nodes before calling the layout engine. Default is 1.</td>
</tr>
<tr>
<td>EdgeTextColor</td>
<td>Three-element numeric vector of RGB values. Default is [0, 0, 0], which defines black.</td>
</tr>
<tr>
<td>EdgeFontSize</td>
<td>Positive number that sets the size of the edge font in points. Default is 8.</td>
</tr>
<tr>
<td>ShowArrows</td>
<td>Controls the display of arrows with the edges. Choices are 'on' (default) or 'off'.</td>
</tr>
<tr>
<td>ArrowSize</td>
<td>Positive number that sets the size of the arrows in points. Default is 8.</td>
</tr>
<tr>
<td>ShowWeights</td>
<td>Controls the display of text indicating the weight of the edges. Choices are 'on' or 'off' (default).</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ShowTextInNodes</td>
<td>Character vector that specifies the node property used to label nodes when you display a biograph object using the <code>view</code> method. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'Label' — Uses the Label property of the node object (default).</td>
</tr>
<tr>
<td></td>
<td>• 'ID' — Uses the ID property of the node object.</td>
</tr>
<tr>
<td></td>
<td>• 'None'</td>
</tr>
<tr>
<td>NodeAutoSize</td>
<td>Controls precalculating the node size before calling the layout engine. Choices are 'on' (default) or 'off'.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> Set it to off if you want to apply different node sizes by changing the Size property.</td>
</tr>
<tr>
<td>NodeCallback</td>
<td>User-defined callback for all nodes. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the view function to display the biograph object in the Biograph Viewer, you can double-click a node to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, @(node) inspect(node), which displays the Property Inspector dialog box.</td>
</tr>
<tr>
<td>EdgeCallback</td>
<td>User-defined callback for all edges. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the view function to display the biograph object in the Biograph Viewer, you can right-click and select a callback to activate. Default is the anonymous function, @(edge) inspect(edge), which displays the Property Inspector dialog box.</td>
</tr>
<tr>
<td>CustomNodeDrawFcn</td>
<td>Function handle to a customized function to draw nodes. Default is {}</td>
</tr>
<tr>
<td>Nodes</td>
<td>Read-only column vector with handles to node objects of a biograph object. The size of the vector is the number of nodes. For properties of node objects, see Properties of a Node Object.</td>
</tr>
<tr>
<td>Edges</td>
<td>Read-only column vector with handles to edge objects of a biograph object. The size of the vector is the number of edges. For properties of edge objects, see Properties of an Edge Object.</td>
</tr>
</tbody>
</table>
Properties of a Node Object

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ID</strong></td>
<td>Character vector defined when the biograph object is created, either by the NodeIDs input argument or internally by the biograph constructor function. You can modify this property using the set method, but each node object's ID must be unique.</td>
</tr>
<tr>
<td><strong>Label</strong></td>
<td>Character vector for labeling a node when you display a biograph object using the view method. Default is ''</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Character vector that describes the node. Default is ''</td>
</tr>
<tr>
<td><strong>Position</strong></td>
<td>Two-element numeric vector of x- and y-coordinates, for example, [150, 150]. If you do not specify this property, default is initially [], then when the layout algorithms are executed, it becomes a two-element numeric vector of x- and y-coordinates computed by the layout engine.</td>
</tr>
<tr>
<td><strong>Shape</strong></td>
<td>Character vector that specifies the shape of the nodes. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'box'(default)</td>
</tr>
<tr>
<td></td>
<td>• 'ellipse'</td>
</tr>
<tr>
<td></td>
<td>• 'circle'</td>
</tr>
<tr>
<td></td>
<td>• 'rect' or 'rectangle'</td>
</tr>
<tr>
<td></td>
<td>• 'diamond'</td>
</tr>
<tr>
<td></td>
<td>• 'trapezium'</td>
</tr>
<tr>
<td></td>
<td>• 'invtrapezium'</td>
</tr>
<tr>
<td></td>
<td>• 'house'</td>
</tr>
<tr>
<td></td>
<td>• 'invhouse'</td>
</tr>
<tr>
<td></td>
<td>• 'parallelogram'</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>Two-element numeric vector calculated before calling the layout engine using the actual font size and shape of the node. Default is [10, 10].</td>
</tr>
<tr>
<td><strong>Color</strong></td>
<td>Three-element numeric vector of RGB values that specifies the fill color of the node. Default is [1, 1, 0.7], which defines yellow.</td>
</tr>
<tr>
<td><strong>LineWidth</strong></td>
<td>Positive number. Default is 1.</td>
</tr>
<tr>
<td><strong>LineColor</strong></td>
<td>Three-element numeric vector of RGB values that specifies the outline color of the node. Default is [0.3, 0.3, 1], which defines blue.</td>
</tr>
<tr>
<td><strong>FontSize</strong></td>
<td>Positive number that sets the size of the node font in points. Default is 8.</td>
</tr>
<tr>
<td><strong>TextColor</strong></td>
<td>Three-element numeric vector of RGB values that specifies the color of the node labels. Default is [0, 0, 0], which defines black.</td>
</tr>
<tr>
<td><strong>UserData</strong></td>
<td>Miscellaneous, user-defined data that you want to associate with the node. The node does not use this property, but you can access and specify it using the get and set functions. Default is [].</td>
</tr>
</tbody>
</table>
Properties of an Edge Object

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Character vector automatically generated from the node IDs when the biograph object is created by the biograph constructor function. You can modify this property using the set method, but each edge object's ID must be unique.</td>
</tr>
<tr>
<td>Label</td>
<td>Character vector for labeling an edge. Default is ''.</td>
</tr>
<tr>
<td>Description</td>
<td>Character vector that describes the edge. Default is ''.</td>
</tr>
<tr>
<td>Weight</td>
<td>Value that represents the weight (cost, distance, length, or capacity) associated with the edge. Default is 1.</td>
</tr>
<tr>
<td>LineWidth</td>
<td>Positive number. Default is 0.5.</td>
</tr>
<tr>
<td>LineColor</td>
<td>Three-element numeric vector of RGB values that specifies the color of the edge. Default is [0.5, 0.5, 0.5], which defines gray.</td>
</tr>
<tr>
<td>UserData</td>
<td>Miscellaneous, user-defined data that you want to associate with the edge. The edge does not use this property, but you can access and specify it using the get and set functions. Default is [].</td>
</tr>
</tbody>
</table>

Examples

Create a Biograph object and specify its properties

This example shows how to create a biograph object, access, and update its properties.

Create a biograph object with custom node IDs.

```matlab
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
ids = {'M30931', 'L07625', 'K03454', 'M27323', 'M15390'};
bg1 = biograph(cm, ids)
```

Biograph object with 5 nodes and 9 edges.

Specify the ID property of the object.

```matlab
bg1.ID = 'mybg';
```

Use the get function to display the node IDs.

```matlab
get(bg1.nodes, 'ID')
```

ans = 5x1 cell

{‘M30931’}
{‘L07625’}
{‘K03454’}
{‘M27323’}
{‘M15390’}

Display all properties and their current values of the 5th node and 5th edge of the object.

```matlab
bg1.nodes(5)
```
Set the LineWidth property of the 5th node to 2.
```
bg1.nodes(5).LineWidth = 2.0;
```

Alternatively use `getnodesbyid` function to create a handle for the 5th node, and set its Shape property to 'circle'.
```
nh1 = getnodesbyid(bg1,'M15390')
```
nh1.Shape = 'circle';

Specify the LineColor property of the 5th edge.
bg1.edges(5).LineColor = [0.7 0.0 0.1];

Alternatively use getedgesbynodeid to retrieve the handle to the edge by providing a source node id and a sink node id.

eh1 = getedgesbynodeid(bg1, 'L07625', 'M15390')

ID: 'L07625 -> M15390'
Label: ''
Description: ''
Weight: 1
LineWidth: 0.5000
LineColor: [0.7000 0 0.1000]
UserData: []

Use the handle to specify the LineWidth property or any other properties of the edge.

eh1.LineWidth = 2.0;

View the biograph object.

view(bg1)
See Also
allshortestpaths | biograph | conncomp | dolayout | get | getancestors | getdescendants | getedgesbynodeid | getmatrix | getnodesbyid | getrelatives | isdag | isomorphism | isspantree | maxflow | minspantree | set | shortestpath | topoorder | traverse | view

Introduced in R2006b
biograph

Create biograph object

Syntax

BGobj = biograph(CMatrix)
BGobj = biograph(CMatrix, NodeIDs)

BGobj = biograph(CMatrix, NodeIDs, ...'ID', IDValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'Label', LabelValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'Description', DescriptionValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'LayoutType', LayoutTypeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'EdgeType', EdgeTypeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'Scale', ScaleValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'EdgeTextColor', EdgeTextColorValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'EdgeFontSize', EdgeFontSizeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'ShowArrows', ShowArrowsValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'ArrowSize', ArrowSizeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'ShowWeights', ShowWeightsValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'ShowTextInNodes', ShowTextInNodesValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'NodeAutoSize', NodeAutoSizeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'NodeCallback', NodeCallbackValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'EdgeCallback', EdgeCallbackValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'CustomNodeDrawFcn', CustomNodeDrawFcnValue, ...)

Arguments

| CMatrix | Full or sparse square matrix that acts as a connection matrix. That is, a value of 1 indicates a connection between nodes while a 0 indicates no connection. The number of rows/columns is equal to the number of nodes. |
**NodeIDs**

Node labels. Enter any of the following:

- Cell array of character vectors or string vector with the number of character vectors (or strings) equal to the number of rows or columns in the connection matrix `CMatrix`. Each character vector (or string) must be unique.
- Character array with the number of rows equal to the number of nodes. Each row in the array must be unique.
- Character vector or string with the number of characters equal to the number of nodes. Each character must be unique.

Default values are the row or column numbers.

**Note** You must specify `NodeIDs` if you want to specify property name/value pairs. Set `NodeIDs` to `[]` to use the default values of the row/column numbers.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IDValue</strong></td>
<td>Character vector or string to identify the biograph object. Default is <code>''</code>.</td>
</tr>
<tr>
<td><strong>LabelValue</strong></td>
<td>Character vector or string to label the biograph object. Default is <code>''</code>.</td>
</tr>
<tr>
<td><strong>DescriptionValue</strong></td>
<td>Character vector or string that describes the biograph object. Default is <code>''</code>.</td>
</tr>
<tr>
<td><strong>LayoutTypeValue</strong></td>
<td>Character vector or string that specifies the algorithm for the layout engine. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• <code>'hierarchical'</code> (default) — Uses a topological order of the graph to assign levels, and then arranges the nodes from top to bottom, while minimizing crossing edges.</td>
</tr>
<tr>
<td></td>
<td>• <code>'radial'</code> — Uses a topological order of the graph to assign levels, and then arranges the nodes from inside to outside of the circle, while minimizing crossing edges.</td>
</tr>
<tr>
<td></td>
<td>• <code>'equilibrium'</code> — Calculates layout by minimizing the energy in a dynamic spring system.</td>
</tr>
</tbody>
</table>

**EdgeTypeValue**

Character vector or string that specifies how edges display. Choices are:

- `'straight'`
- `'curved'` (default)
- `'segmented'`

**Note** Curved or segmented edges occur only when necessary to avoid obstruction by nodes. Biograph objects with `LayoutType` equal to `'equilibrium'` or `'radial'` cannot produce curved or segmented edges.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ScaleValue</strong></td>
<td>Positive number that post-scales the node coordinates. Default is 1.</td>
</tr>
<tr>
<td><strong>LayoutScaleValue</strong></td>
<td>Positive number that scales the size of the nodes before calling the layout engine. Default is 1.</td>
</tr>
<tr>
<td><strong>EdgeTextColorValue</strong></td>
<td>Three-element numeric vector of RGB values. Default is [0, 0, 0], which defines black.</td>
</tr>
<tr>
<td><strong>EdgeFontSizeValue</strong></td>
<td>Positive number that sets the size of the edge font in points. Default is 8.</td>
</tr>
<tr>
<td><strong>ShowArrowsValue</strong></td>
<td>Controls the display of arrows for the edges. Choices are 'on' (default) or 'off'.</td>
</tr>
<tr>
<td><strong>ArrowSizeValue</strong></td>
<td>Positive number that sets the size of the arrows in points. Default is 8.</td>
</tr>
<tr>
<td><strong>ShowWeightsValue</strong></td>
<td>Controls the display of text indicating the weight of the edges. Choices are 'on' or 'off' (default).</td>
</tr>
<tr>
<td><strong>ShowTextInNodesValue</strong></td>
<td>Character vector or string that specifies the node property used to label nodes when you display a biograph object using the view method. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'Label' — Uses the Label property of the node object (default).</td>
</tr>
<tr>
<td></td>
<td>• 'ID' — Uses the ID property of the node object.</td>
</tr>
<tr>
<td></td>
<td>• 'None'</td>
</tr>
<tr>
<td><strong>NodeAutoSizeValue</strong></td>
<td>Controls precalculating the node size before calling the layout engine. Choices are 'on' (default) or 'off'.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> Set it to off if you want to apply different node sizes by changing the Size property.</td>
</tr>
<tr>
<td><strong>NodeCallbackValue</strong></td>
<td>User callback for all nodes. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the view function to display the biograph in the Biograph Viewer, you can double-click a node to activate the first callback, or right-click and select a callback to activate. Default is @(node) inspect(node), which displays the Property Inspector dialog box.</td>
</tr>
<tr>
<td><strong>EdgeCallbackValue</strong></td>
<td>User callback for all edges. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the view function to display the biograph object in the Biograph Viewer, you can right-click and select a callback to activate. Default is the anonymous function, @(edge) inspect(edge), which displays the Property Inspector dialog box.</td>
</tr>
<tr>
<td><strong>CustomNodeDrawFcnValue</strong></td>
<td>Function handle to a customized function to draw nodes. Default is [].</td>
</tr>
</tbody>
</table>
Description

`BGobj = biograph(CMatrix)` creates a biograph object, `BGobj`, using a connection matrix, `CMatrix`. All nondiagonal and positive entries in the connection matrix, `CMatrix`, indicate connected nodes, rows represent the source nodes, and columns represent the sink nodes.

`BGobj = biograph(CMatrix, NodeIDs)` specifies the node labels. `NodeIDs` can be:

- Cell array of character vectors or string vector with the number of character vectors (or strings) equal to the number of rows or columns in the connection matrix `CMatrix`. Each character vector or string must be unique.
- Character array with the number of rows equal to the number of nodes. Each row in the array must be unique.
- Character vector or string with the number of characters equal to the number of nodes. Each character must be unique.

Default values are the row or column numbers.

**Note** If you want to specify property name/value pairs, you must specify `NodeIDs`. Set `NodeIDs` to `[]` to use the default values of the row/column numbers.

`BGobj = biograph(..., 'PropertyName', PropertyValue, ...)` calls `biograph` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`BGobj = biograph(CMatrix, NodeIDs, ...'ID', IDValue, ...)` specifies an ID for the biograph object. Default is ''.

`BGobj = biograph(CMatrix, NodeIDs, ...'Label', LabelValue, ...)` specifies a label for the biograph object. Default is ''.

`BGobj = biograph(CMatrix, NodeIDs, ...'Description', DescriptionValue, ...)` specifies a description of the biograph object. Default is ''.

`BGobj = biograph(CMatrix, NodeIDs, ...'LayoutType', LayoutTypeValue, ...)` specifies the algorithm for the layout engine.

`BGobj = biograph(CMatrix, NodeIDs, ...'EdgeType', EdgeTypeValue, ...)` specifies how edges display.

`BGobj = biograph(CMatrix, NodeIDs, ...'Scale', ScaleValue, ...)` post-scales the node coordinates. Default is 1.

`BGobj = biograph(CMatrix, NodeIDs, ...'LayoutScale', LayoutScaleValue, ...)` scales the size of the nodes before calling the layout engine. Default is 1.

`BGobj = biograph(CMatrix, NodeIDs, ...'EdgeTextColor', EdgeTextColorValue, ...)` specifies a three-element numeric vector of RGB values. Default is [0, 0, 0], which defines black.
$BGobj = biograph(CMatrix, NodeIDs, ...'EdgeFontSize', EdgeFontSizeValue, ...)$ sets the size of the edge font in points. Default is 8.

$BGobj = biograph(CMatrix, NodeIDs, ...'ShowArrows', ShowArrowsValue, ...)$ controls the display of arrows for the edges. Choices are 'on' (default) or 'off'.

$BGobj = biograph(CMatrix, NodeIDs, ...'ArrowSize', ArrowSizeValue, ...)$ sets the size of the arrows in points. Default is 8.

$BGobj = biograph(CMatrix, NodeIDs, ...'ShowWeights', ShowWeightsValue, ...)$ controls the display of text indicating the weight of the edges. Choices are 'on' (default) or 'off'.

$BGobj = biograph(CMatrix, NodeIDs, ...'ShowTextInNodes', ShowTextInNodesValue, ...)$ specifies the node property used to label nodes when you display a biograph object using the view method.

$BGobj = biograph(CMatrix, NodeIDs, ...'NodeAutoSize', NodeAutoSizeValue, ...)$ controls precalculating the node size before calling the layout engine. Choices are 'on' (default) or 'off'.

$BGobj = biograph(CMatrix, NodeIDs, ...'NodeCallback', NodeCallbackValue, ...)$ specifies user callback for all nodes.

$BGobj = biograph(CMatrix, NodeIDs, ...'EdgeCallback', EdgeCallbackValue, ...)$ specifies user callback for all edges.

$BGobj = biograph(CMatrix, NodeIDs, ...'CustomNodeDrawFcn', CustomNodeDrawFcnValue, ...)$ specifies function handle to customized function to draw nodes. Default is [].

**Examples**

**Create a biograph object**

This example shows how to create a biograph object.

Create a biograph object with default node IDs, and then use the get function to display the node IDs.

```matlab
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
bgl = biograph(cm)
Biograph object with 5 nodes and 9 edges.
get(bgl.nodes, 'ID')
ans = 5x1 cell
    {'Node 1'}
    {'Node 2'}
    {'Node 3'}
    {'Node 4'}
    {'Node 5'}
```

Create a biograph object, assign the node IDs, and then use the get function to display the node IDs.
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
ids = {'M30931','L07625','K03454','M27323','M15390'};
bg2 = biograph(cm,ids);
get(bg2.nodes,'ID')

ans = 5x1 cell
{'M30931'}
{'L07625'}
{'K03454'}
{'M27323'}
{'M15390'}

Display the biograph object.

view(bg2)
See Also
allshortestpaths | conncomp | dolayout | get | getancestors | getdescendants |
getedgesbynodeid | getmatrix | getnodesbyid | getrelatives | isdag | isomorphism |
isspantree | maxflow | minspantree | set | shortestpath | topoorder | traverse | view

Topics
biograph object on page 1-185

Introduced in R2006a
BioIndexedFile class

Allow quick and efficient access to large text file with nonuniform-size entries

Description

The BioIndexedFile class allows access to text files with nonuniform-size entries, such as sequences, annotations, and cross-references to data sets. It lets you quickly and efficiently access this data without loading the source file into memory.

This class lets you access individual entries or a subset of entries when the source file is too big to fit into memory. You can access entries using indices or keys. You can read and parse one or more entries using provided interpreters or a custom interpreter function.

Construction

BioIFobj = BioIndexedFile(Format,SourceFile) returns a BioIndexedFile object BioIFobj that indexes the contents of SourceFile following the parsing rules defined by Format, where SourceFile and Format specify the names of a text file and a file format, respectively. It also constructs an auxiliary index file to store information that allows efficient, direct access to SourceFile. The index file by default is stored in the same location as the source file and has the same name as the source file, but with an IDX extension. The BioIndexedFile constructor uses the index file to construct subsequent objects from SourceFile, which saves time.

BioIFobj = BioIndexedFile(Format,SourceFile,IndexDir) returns a BioIndexedFile object BioIFobj by specifying the relative or absolute path to a folder to use when searching for or saving the index file.

BioIFobj = BioIndexedFile(Format,SourceFile,IndexFile) returns a BioIndexedFile object BioIFobj by specifying a file name, optionally including a relative or absolute path, to use when searching for or saving the index file.

BioIFobj = BioIndexedFile(__,Name,Value) returns a BioIndexedFile object BioIFobj by using any input arguments from the previous syntaxes and additional options, specified as one or more Name,Value pair arguments.

Input Arguments

Format

Character vector or string specifying a file format. Choices are:

- 'SAM' — SAM-formatted file
- 'FASTQ' — FASTQ-formatted file
- 'FASTA' — FASTA-formatted file
- 'TABLE' — Tab-delimited table with multiple columns. Keys can be in any column. Rows with the same key are considered separate entries.
• 'MRTAB' — Tab-delimited table with multiple columns. Keys can be in any column. Contiguous rows with the same key are considered a single entry. Noncontiguous rows with the same key are considered separate entries.

• 'FLAT' — Flat file with concatenated entries separated by a character vector, typically '//'. Within an entry, the key is separated from the rest of the entry by a white space.

**Note** For all file formats, the file contents must only use ASCII text characters. Non-ASCII characters may not be properly indexed.

**Default:**

**SourceFile**

Character vector or string specifying the name of a text file. It can include a relative or absolute path.

**IndexDir**

Character vector or string specifying the relative or absolute path to a folder to use when searching for or saving the index file.

**IndexFile**

Character vector or string specifying a file name, optionally including a relative or absolute path, to use when searching for or saving the index file.

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

**IndexedByKeys**

Specifies if you can access the object BioIFobj using keys. Choices are true or false.

**Tip** Set the value to false if you do not need to access entries in the object using keys. Doing so saves time and space when creating the object.

**Default:** true

**MemoryMappedIndex**

Specifies whether the constructor stores the indices in the auxiliary index file and accesses them via memory maps (true) or loads the indices into memory at construction time (false).

**Tip** If memory is not an issue and you want to maximize performance when accessing entries in the object, set the value to false.

**Default:** true
**Interpreter**

Handle to a function that the `read` method uses when parsing entries from the source file. The interpreter function must accept a character vector of one or more concatenated entries and return a structure or an array of structures containing the interpreted data.

When `Format` is a general-purpose format such as 'TABLE', 'MRTAB', or 'FLAT', then the default is [], which means the function is an anonymous function in which the output is equivalent to the input.

When `Format` is an application-specific format such as 'SAM', 'FASTQ', or 'FASTA', then the default is a function handle appropriate for that file type and typically does not require you to change it.

**Default:**

**Verbose**

Controls the display of the status of the object construction. Choices are `true` or `false`.

**Default:** `true`

---

**Note** The following name-value pair arguments apply only when both of the following are true:

- There is no pre-existing index file associated with your source file.
- Your source file has a general-purpose format such as 'TABLE', 'MRTAB', or 'FLAT'.

For source files with application-specific formats, the following name-value pairs are pre-defined and you cannot change them.

**KeyColumn**

Positive integer specifying the column in the 'TABLE' or 'MRTAB' file that contains the keys.

**Default:** 1

**KeyToken**

Character vector or string that occurs in each entry before the key, for 'FLAT' files that contain keys. If the value is '', it indicates the key is the first character vector (or string) in each entry and is delimited by blank spaces.

**Default:** ''

**HeaderPrefix**

Character vector or string specifying a prefix that denotes header lines in the source file so the constructor ignores them when creating the object. If the value is [], it means the constructor does not check for header lines in the source file.

**Default:** []
CommentPrefix

Character vector or string specifying a prefix that denotes comment lines in the source file so the constructor ignores them when creating the object. If the value is [], it means the constructor does not check for comment lines in the source file.

Default: []

ContiguousEntries

Specifies whether entries are on contiguous lines, which means they are not separated by empty lines or comment lines, in the source file or not. Choices are true or false.

Tip Set the value to true when entries are not separated by empty lines or comment lines. Doing so saves time and space when creating the object.

Default: false

TableDelimiter

Character vector or string specifying a delimiter symbol to use as a column separator for SourceFile when Format is 'TABLE' or 'MRTAB'. Choices are \t (horizontal tab), ' ' (blank space), or ',', (comma).

Default: \t

EntryDelimiter

Character vector or string specifying a delimiter symbol to use as an entry separator for SourceFile when Format is 'FLAT'.

Default: '//'
**IndexedByKeys**

Whether or not the entries in the source file can be indexed by an alphanumeric key.

This information is read only.

**IndexFile**

Path and file name of the auxiliary index file.

This information is read only. Use this property to confirm the name and location of the index file associated with the object.

**InputFile**

Path and file name of the source file.

This information is read only. Use this property to confirm the name and location of the source file from which the object was constructed.

**Interpreter**

Handle to a function used by the `read` method to parse entries in the source file.

This interpreter function must accept a character vector of one or more concatenated entries and return a structure or an array of structures containing the interpreted data. Set this property when your source file has a `TABLE`, `MRTAB`, or `FLAT` format. When your source file is an application-specific format such as `SAM`, `FASTQ`, or `FASTA`, then the default is a function handle appropriate for that file type and typically does not require you to change it.

**MemoryMappedIndex**

Whether the indices to the source file are stored in a memory-mapped file or in memory.

**NumEntries**

Number of entries indexed by the object.

This information is read only.
Methods

getDictionary  Retrieve reference sequence names from SAM-formatted source file associated with BioIndexedFile object
getEntryByIndex Retrieve entries from source file associated with BioIndexedFile object using numeric index
getEntryByKey Retrieve entries from source file associated with BioIndexedFile object using alphanumeric key
getIndexByKey Retrieve indices from source file associated with BioIndexedFile object using alphanumeric key
getKeys Retrieve alphanumeric keys from source file associated with BioIndexedFile object
getSubset Create object containing subset of elements from BioIndexedFile object
read Read one or more entries from source file associated with BioIndexedFile object

Copy Semantics

Value. To learn how value classes affect copy operations, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

Examples

Construct a BioIndexedFile object and access its gene ontology (GO) terms

This example shows how to construct a BioIndexedFile object and access its gene ontology (GO) terms.

Create a variable containing full absolute path of source file.

sourcefile = which('yeastgenes.sgd');

Copy the file to the current working directory.

copyfile(sourcefile,'yeastgenes_copy.sgd');

Construct a BioIndexedFile object from the source file that is a tab-delimited file, considering contiguous rows with the same key as a single entry. Indicate that keys are located in column 3 and that header lines are prefaced with '!'.

gene2goObj = BioIndexedFile('mrtab','yeastgenes_copy.sgd','KeyColumn',3,'HeaderPrefix','!');

Source File: yeastgenes_copy.sgd
  Path: C:\TEMP\Bdoc20a_1326390_8984\ib9D0363\4\tpa6d49ae8\bioinfo-ex58973989
  Size: 21455392 bytes
  Date: 15-Mar-2018 17:45:16
Creating new index file ...
Indexer found 36266 entries after parsing 111912 text lines.
Index File: yeastgenes_copy.sgd.idx
  Path: C:\TEMP\Bdoc20a_1326390_8984\ib9D0363\4\tpa6d49ae8\bioinfo-ex58973989
  Size: 494723 bytes
  Date: 29-Feb-2020 00:50:26
Mapping object to yeastgenes_copy.sgd.idx ... 
Done.

Return the GO term from all entries that are associated with the gene YAT2. Access entries that have a key of YAT2.

\[
YAT2\_entries = \text{getEntryByKey}(\text{gene2goObj}, 'YAT2');
\]

Adjust object interpreter to return only the column containing the GO term.

\[
\text{gene2goObj.Interpreter} = @(x) \text{regexp}(x, 'GO:\d+', 'match');
\]

Parse the entries with a key of YAT2 and return all GO terms from those entries.

\[
\text{GO\_YAT2\_entries} = \text{read}(\text{gene2goObj}, 'YAT2')
\]

\[
\text{GO\_YAT2\_entries} = 1x14 ~\text{cell}
\]

Columns 1 through 4

\[
\{\text{GO:0004092} \} \quad \{\text{GO:0006066} \} \quad \{\text{GO:0006066} \} \quad \{\text{GO:0009437} \}
\]

Columns 5 through 8

\[
\{\text{GO:0005829} \} \quad \{\text{GO:0005737} \} \quad \{\text{GO:0004092} \} \quad \{\text{GO:0016740} \}
\]

Columns 9 through 12

\[
\{\text{GO:0016746} \} \quad \{\text{GO:0006629} \} \quad \{\text{GO:0016746} \} \quad \{\text{GO:0005737} \}
\]

Columns 13 through 14

\[
\{\text{GO:0006631} \} \quad \{\text{GO:0005737} \}
\]

See Also

fastaread | fastqread | genbankread | memmapfile | samread

Topics

“Work with Next-Generation Sequencing Data”
bioma.data.ExptData class

Package: bioma.data

Contain data values from microarray experiment

Description

The ExptData class is designed to contain data values, such as gene expression values, from a microarray experiment. It stores the data values in one or more DataMatrix objects on page 1-532, each having the same row names (feature names) and column names (sample names). It provides a convenient way to store related experiment data in a single data structure (object). It also lets you manage and subset the data.

The ExptData class includes properties and methods that let you access, retrieve, and change data values from a microarray experiment. These properties and methods are useful to view and analyze the data.

Construction

$EDobj = \text{bioma.data.ExptData}(Data1, Data2, \ldots)$ creates an ExptData object, from one or more matrices of data. Each matrix can be a logical matrix, a numeric matrix, or a DataMatrix object on page 1-532.

$EDobj = \text{bioma.data.ExptData}(\ldots, \{DMobj1, Name1\}, \{DMobj2, Name2\}, \ldots)$ specifies an element name for each DataMatrix object. $Name#$ is a character vector or string specifying a unique name. Default names are $Elmt1, Elmt2$, etc.

$EDobj = \text{bioma.data.ExptData}(\{Data1, Data2, \ldots\})$ creates an ExptData object, from a cell array of matrices of data. Each matrix can be a logical matrix, a numeric matrix, or a DataMatrix object on page 1-532.

$EDobj = \text{bioma.data.ExptData}(\ldots, 'PropertyName', PropertyValue)$ constructs the object using options, specified as property name/property value pairs.

$EDobj = \text{bioma.data.ExptData}(\ldots, 'ElementNames', ElementNamesValue)$ specifies element names for the matrix inputs. $ElementNamesValue$ is a cell array of character vectors or string vector. Default names are $Elmt1, Elmt2$, etc.

$EDobj = \text{bioma.data.ExptData}(\ldots, 'FeatureNames', FeatureNamesValue)$ specifies feature names (row names) for the ExptData object.

$EDobj = \text{bioma.data.ExptData}(\ldots, 'SampleNames', SampleNamesValue)$ specifies sample names (column names) for the ExptData object.

Input Arguments

Data#

Matrix of experimental data values specified by any of the following:
• Logical matrix
• Numeric matrix
• DataMatrix object on page 1-532

All inputs must have the same dimensions. All DataMatrix objects must also have the same row
names and columns names. If you provide logical or numeric matrices, bioma.data.ExptData
converts them to DataMatrix objects with either default row and column names, or the row and
column names of DataMatrix inputs, if provided.

The rows must correspond to features and the columns must correspond to samples.

Default:

DMobj#

Variable name of a DataMatrix object in the MATLAB Workspace.

Default:

Name#

Character vector or string specifying an element name for the corresponding DataMatrix object

ElementNamesValue

Cell array of character vectors or string vector that specifies unique element names for the matrix
inputs. The number of elements in ElementNamesValue must equal the number input matrices.

Default: {Elmt1, Elmt2, ...}

FeatureNamesValue

Feature names (row names) for the ExptData object, specified by one of the following:

• Cell array of character vectors
• Character array
• String vector
• Numeric or logical vector
• Character vector or string, which is used as a prefix for the feature names, with feature numbers
appended to the prefix
• Logical true or false (default). If true, bioma.data.ExptData assigns unique feature names
using the format Feature1, Feature2, etc.

If you use a cell array of character vectors, character array, string vector, numeric or logical vector,
then the number of elements must be equal in number to the number of rows in Data1.

SampleNamesValue

Sample names (column names) for the ExptData object, specified by one of the following:

• Cell array of character vectors
• Character array
• String vector
• Numeric or logical vector
• Character vector or string, which is used as a prefix for the sample names, with sample numbers appended to the prefix
• Logical true or false (default). If true, `bioma.data.ExptData` assigns unique sample names using the format Sample1, Sample2, etc.

If you use a cell array of character vectors, character array, string vector, numeric or logical vector, then the number of elements must be equal in number to the number of columns in `Data1`. If the ExptData object is part of an ExpressionSet object that contains a MetaData object, the sample names (column names) in the ExptData object must match the sample names (row names) in a MetaData object.

**Default:**

**Properties**

**ElementClass**

Class type of the DataMatrix objects in the experiment

Cell array of character vectors specifying the class type of each DataMatrix object in the ExptData object. Possible values are MATLAB classes, such as `single`, `double`, and `logical`. This information is read-only.

**Attributes:**

SetAccess: private

**Name**

Name of the ExptData object.

Character vector specifying the name of the ExptData object. Default is `[]`.

**NElements**

Number of elements in the experiment

Positive integer specifying the number of elements (DataMatrix objects) in the experiment data. This value is equivalent to the number of DataMatrix objects in the ExptData object. This information is read-only.

**Attributes:**

SetAccess: private

**NFeatures**

Number of features in the experiment

Positive integer specifying the number of features in the experiment. This value is equivalent to the number of rows in each DataMatrix object in the ExptData object. This information is read-only.
Attributes:
SetAccess private

NSamples
Number of samples in the experiment
Positive integer specifying the number of samples in the experiment. This value is equivalent to the number of columns in each DataMatrix object in the ExptData object. This information is read-only.

Attributes:
SetAccess private

Methods
combine Combine two ExptData objects
dmNames Retrieve or set Name properties of DataMatrix objects in ExptData object
elementData Retrieve or set data element (DataMatrix object) in ExptData object
elementNames Retrieve or set data element names of DataMatrix objects in ExptData object
featureNames Retrieve or set feature names in ExptData object
isempty Determine whether ExptData object is empty
sampleNames Retrieve or set sample names in ExptData object
size Return size of ExptData object

Instance Hierarchy
An ExpressionSet object contains an ExptData object. An ExptData object contains one or more DataMatrix objects.

Attributes
To learn about attributes of classes, see Class Attributes (MATLAB).

Copy Semantics
Value. To learn how this affects your use of the class, see Copying Objects (MATLAB).

Indexing
ExptData objects support 1-D parenthesis ( ) indexing to extract, assign, and delete data.

ExptData objects do not support:
- Dot . indexing
- Curly brace { } indexing
Examples

Construct an ExptData object

This example shows how to construct an ExptData object containing one DataMatrix object.

Import bioma.data package to make constructor functions available.

```matlab
import bioma.data.*
```

Create a DataMatrix object from .txt file containing expression values from microarray experiment.

```matlab
dmObj = DataMatrix('File', 'mouseExprsData.txt');
```

Construct an ExptData object from the DataMatrix object.

```matlab
EDObj = ExptData(dmObj)
```

EDObj =

Experiment Data:
500 features, 26 samples
1 elements
Element names: Elmt1

References


See Also

bioma.ExpressionSet | bioma.data.ExptData | bioma.data.MIAME | bioma.data.MetaData

getgeodata

Topics

“Working with Objects for Microarray Experiment Data”
“Analyzing Illumina® Bead Summary Gene Expression Data”
Class Attributes (MATLAB)
Property Attributes (MATLAB)
“Representing Experiment Information in a MIAME Object”
bioma.data.MetaData class

Package: bioma.data

Contain metadata from microarray experiment

Description

The MetaData class is designed to contain metadata (variable values and descriptions) from a microarray experiment. It provides a convenient way to store related metadata in a single data structure (object). It also lets you manage and subset the data.

The metadata is a collection of variable names, for example related to samples or microarray features, along with descriptions and values for the variables. A MetaData object stores the metadata in two dataset arrays.

• **Values dataset array** — A dataset array containing the measured value of each variable per sample or feature. In this dataset array, the columns correspond to variables and rows correspond to either samples or features. The number and names of the columns in this dataset array must match the number and names of the rows in the Descriptions dataset array. If this dataset array contains *sample* metadata, then the number and names of the rows (samples) must match the number and names of the columns in the DataMatrix objects in the same ExpressionSet object. If this dataset array contains *feature* metadata, then the number and names of the rows (features) must match the number and names of the columns in the DataMatrix objects in the same ExpressionSet object.

• **Descriptions dataset array** — A dataset array containing a list of the variable names and their descriptions. In this dataset array, each row corresponds to a variable. The row names are the variable names, and a column, named *VariableDescription*, contains a description of the variable. The number and names of the rows in the Descriptions dataset array must match the number and names of the columns in the Values dataset array.

The MetaData class includes properties and methods that let you access, retrieve, and change metadata variables, and their values and descriptions. These properties and methods are useful to view and analyze the metadata.

Construction

MDobj = bioma.data.MetaData(VarValues) creates a MetaData object from one dataset array whose rows correspond to sample (observation) names and whose columns correspond to variables. The dataset array contains the measured value of each variable per sample.

MDobj = bioma.data.MetaData(VarValues, VarDescriptions) creates a MetaData object from two dataset arrays. **VarDescriptions** is a dataset array whose rows correspond to variables. The row names are the variable names, and another column, named *VariableDescription*, contains a description of each variable.

MDobj = bioma.data.MetaData(VarValues, VarDesc) creates a MetaData object from a dataset array and **VarDesc** a cell array of character vectors containing descriptions of the variables.

MDobj = bioma.data.MetaData(..., 'PropertyName', PropertyValue) constructs the object using options, specified as property name/property value pairs.
MDobj = bioma.data.MetaData('File', FileValue) creates a MetaData object from a text file containing a table of metadata. The table row labels must be sample names, and its column headers must be variable names.

MDobj = bioma.data.MetaData('File', FileValue, 'Path', PathValue) specifies a folder or path and folder where FileValue is stored.

MDobj = bioma.data.MetaData('File', FileValue, 'Delimiter', DelimiterValue) specifies a delimiter symbol to use as a column separator for FileValue. Default is '\t'.

MDobj = bioma.data.MetaData('File', FileValue, 'RowNames', RowNamesValue) specifies the row names (sample names) for the MetaData object. Default is the information in the first column of the table.

MDobj = bioma.data.MetaData('File', FileValue, 'ColumnNames', ColumnNamesValue) specifies the columns of data to read from the table. ColumnNamesValue is a cell array of character vectors specifying the column header names. Default is to read all columns of data from the table, assuming the first row contains column headers.

MDobj = bioma.data.MetaData('File', FileValue, 'VarDescChar', VarDescCharValue) specifies that lines in the table prefixed by VarDescCharValue to be read as descriptions and used to create the VarDescriptions dataset array. By default, bioma.data.MetaData does not read variable description information, and does not create a Descriptions dataset array. These prefixed lines must appear at the top of the file, before the table of metadata values.

MDobj = bioma.data.MetaData(...'Name', NameValue) specifies a name for the MetaData object.

MDobj = bioma.data.MetaData('File', FileValue, 'Description', DescriptionValue) specifies a description for the MetaData object.

MDobj = bioma.data.MetaData('File', FileValue, 'SampleNames', SampleNamesValue) specifies sample names (row names) for the MetaData object.

MDobj = bioma.data.MetaData('File', FileValue, 'VariableNames', VariableNamesValue) specifies variable names (column names) for the MetaData object.

Input Arguments

VarValues

Dataset array whose rows correspond to sample (observation) names and whose columns correspond to variables. The dataset array contains the measured value of each variable per sample or feature.

The number and names of the columns in the VarValues dataset array must match the number and names of the rows in the VarDescriptions dataset array. If VarValues contains sample metadata, then the number and names of the rows (samples) must match the number and names of the columns in the DataMatrix objects in the same ExpressionSet object. If VarValues contains feature metadata, then the number and names of the rows (features) must match the number and names of the rows in the DataMatrix objects in the same ExpressionSet object.
**VarDescriptions**

Dataset array whose rows correspond to variables. The row names are the variable names, and a column, named `VariableDescription`, contains a description of the variable. The number and names of the rows in the `VarDescriptions` dataset array must match the number and names of the columns in the `VarValues` dataset array.

**VarDesc**

Cell array of character vectors containing descriptions of the variables. The number of elements in `VarDesc` must equal the number of columns (variable names) in `VarValues`.

**FileValue**

Character vector specifying a text file containing a table of metadata. The table row labels must be sample or feature names, and its column headers must be variable names. The text file must be on the MATLAB search path or in the Current Folder (unless you use the `Path` property).

**Default:**

**PathValue**

Character vector specifying a folder or path and folder where `FileValue` is stored.

**DelimiterValue**

Character vector specifying a delimiter symbol to use as a column separator for `FileValue`. Typical choices are:

- `' '`
- `'\t'` (default)
- `','`
- `';'`
- `'|'`

**RowNamesValue**

Row names (sample or feature names) for the MetaData object, specified by one of the following:

- Cell array of character vectors
- Single number indicating the column of the table containing the row names
- Character vector indicating the column header of the table containing the row names

If you specify `[]` for `RowNamesValue`, then `bioma.data.MetaData` provides numbered row names, starting with 1.

**Default:** 1, which specifies the information in the first column of the table

**ColumnNamesValue**

Cell array of character vectors specifying the column header names to indicate which columns of data to read from the table. Default is to read all columns of data from the table, assuming the first row contains column headers. If the table does not have column headers, specify `[]` for
**ColumnNamesValue** to read all columns of data and provide numbered column names, starting with 1.

**VarDescCharValue**
Character vector specifying a character to prefix lines in the table that are to be read as descriptions and used to create the *VarDescriptions* dataset array. By default, *bioma.data.MetaData* does not read variable description information, and does not create a *VarDescriptions* dataset array. These prefixed lines must appear at the top of the file, before the table of metadata values.

**NameValue**
Character vector specifying a name for the MetaData object.

**DescriptionValue**
Character vector specifying a description for the MetaData object.

**SampleNamesValue**
Cell array of character vectors specifying sample names for the MetaData object. The number of elements in the cell array must equal the number of samples in the MetaData object. This input overwrites sample names from the input file. Default are the sample names (row names) from the input file.

**VariableNamesValue**
Cell array of character vectors specifying variable names for the MetaData object. The number of elements in the cell array must equal the number of variables in the MetaData object. This input overwrites variable names from the input file. Default are the variable names (column names) from the input file.

**Properties**

**Description**
Description of the MetaData object.
Character vector specifying a description of the MetaData object. Default is `[]`.

**DimensionLabels**
Row and column labels for the MetaData object.
Two-element cell array containing character vectors specifying labels of the rows and columns respectively in the MetaData object. Default is `{'Samples', 'Variables'}`.

**Name**
Name of the MetaData object.
Character vector specifying the name of the MetaData object. Default is `[]`.

**NSamples**
Number of samples (observations) in the experiment
Positive integer specifying the number of samples in the experiment. This value is equivalent to the number of rows in the \textit{VarValues} dataset array. This information is read-only.

\textbf{Attributes:}

\begin{itemize}
  \item \textbf{SetAccess} \hspace{1cm} \textit{private}
\end{itemize}

\textbf{NVariables}

Number of variables in the experiment.

Positive integer specifying the number of variables in the experiment. This value is equivalent to the number of columns in the \textit{VarValues} dataset array. This information is read-only.

\textbf{Attributes:}

\begin{itemize}
  \item \textbf{SetAccess} \hspace{1cm} \textit{private}
\end{itemize}

\textbf{Methods}

\begin{itemize}
  \item \textbf{combine} \hspace{1cm} Combine two MetaData objects
  \item \textbf{isempty} \hspace{1cm} Determine whether MetaData object is empty
  \item \textbf{sampleNames} \hspace{1cm} Retrieve or set sample names in MetaData object
  \item \textbf{size} \hspace{1cm} Return size of MetaData object
  \item \textbf{variableDesc} \hspace{1cm} Retrieve or set variable descriptions for samples in MetaData object
  \item \textbf{variableNames} \hspace{1cm} Retrieve or set variable names for samples in MetaData object
  \item \textbf{variableValues} \hspace{1cm} Retrieve or set variable values for samples in MetaData object
  \item \textbf{varValuesTable} \hspace{1cm} Create 2-D graphic table GUI of variable values in MetaData object
\end{itemize}

\textbf{Instance Hierarchy}

An ExpressionSet object contains two MetaData objects, one for sample information and one for microarray feature information. A MetaData object contains two dataset arrays. One dataset array contains the measured value of each variable per sample or feature. The other dataset array contains a list of the variable names and their descriptions.

\textbf{Attributes}

To learn about attributes of classes, see Class Attributes (MATLAB).

\textbf{Copy Semantics}

Value. To learn how this affects your use of the class, see Copying Objects (MATLAB).

\textbf{Indexing}

MetaData objects support 2-D parenthesis ( ) indexing and dot . indexing to extract, assign, and delete data.
MetaData objects do not support:

- Curly brace `{ }` indexing
- Linear indexing

**Examples**

Construct a MetaData object containing sample variable information from a text file:

```matlab
% Import bioma.data package to make constructor function available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Display information about the MetaData object
MDObj2
% Supply a description for the MetaData object
MDObj2.Description = 'This MetaData Object contains sample variable info.'
```

**See Also**

bioma.ExpressionSet | bioma.data.ExptData | bioma.data.MIAME | bioma.data.MetaData | getgeodata

**Topics**

“Working with Objects for Microarray Experiment Data”
“Analyzing Illumina® Bead Summary Gene Expression Data”
Class Attributes (MATLAB)
Property Attributes (MATLAB)
“Representing Experiment Information in a MIAME Object”
bioma.data.MIAME class

Package: bioma.data

Contain experiment information from microarray gene expression experiment

Description

The MIAME class is designed to contain information about experimental methods and conditions from a microarray gene expression experiment. It loosely follows the Minimum Information About a Microarray Experiment (MIAME) specification. It can include information about:

- Experiment design
- Microarrays used in the experiment
- Samples used
- Sample preparation and labeling
- Hybridization procedures and parameters
- Normalization controls
- Preprocessing information
- Data processing specifications

It provides a convenient way to store related information about a microarray experiment in a single data structure (object).

The MIAME class includes properties and methods that let you access, retrieve, and change experiment information related to a microarray experiment. These properties and methods are useful to view and analyze the information.

Construction

\[ \text{MIAMEobj} = \text{bioma.data.MIAME}() \] creates an empty MIAME object for storing experiment information from a microarray gene expression experiment.

\[ \text{MIAMEobj} = \text{bioma.data.MIAME(\text{GeoSeriesStruct})} \] creates a MIAME object from a structure containing Gene Expression Omnibus (GEO) Series data.

\[ \text{MIAMEobj} = \text{bioma.data.MIAME(..., 'PropertyName', PropertyValue)} \] constructs the object using options, specified as property name/property value pairs.

\[ \text{MIAMEobj} = \text{bioma.data.MIAME(..., 'Investigator', InvestigatorValue)} \] specifies the name of the experiment investigator.

\[ \text{MIAMEobj} = \text{bioma.data.MIAME(..., 'Lab', LabValue)} \] specifies the laboratory that conducted the experiment.

\[ \text{MIAMEobj} = \text{bioma.data.MIAME(..., 'Contact', ContactValue)} \] specifies the contact information for the experiment investigator or laboratory.

\[ \text{MIAMEobj} = \text{bioma.data.MIAME(..., 'URL', URLValue)} \] specifies the experiment URL.
**Input Arguments**

**GeoSeriesStruct**

Gene Expression Omnibus (GEO) Series data specified by either:

- MATLAB structure returned by the `getgeodata` function
- `Structure.Header.Series` substructure returned by the `getgeodata` function

**Default:**

**InvestigatorValue**

Character vector specifying the name of the experiment investigator.

**Default:**

**LabValue**

Character vector specifying the laboratory that conducted the experiment.

**Default:**

**ContactValue**

Character vector specifying the contact information for the experiment investigator or laboratory.

**Default:**

**URLValue**

Character vector specifying the experiment URL.

**Default:**

**Properties**

**Abstract**

Abstract describing the experiment

Character vector containing an abstract describing the experiment.

**Arrays**

Information about the microarray chips used in the experiment

Cell array containing information about the microarray chips used in the experiment. Information can include array name, array platform, number of features on the array, and so on.

**Contact**

Contact information for the experiment investigator or laboratory

Character array containing contact information for the experiment investigator or laboratory.
**ExptDesign**
Brief description of the experiment design
Character array containing description of the experiment design.

**Hybridization**
Information about the experiment hybridization
Cell array containing information about the hybridization protocol used in the experiment. Information can include hybridization time, concentration, volume, temperature, and so on.

**Investigator**
Name of the experiment investigator
Character array containing the name of the experiment investigator.

**Laboratory**
Name of the laboratory where the experiment was conducted
Character array containing the name of laboratory.

**Other**
Other information about the experiment
Cell array containing other information about the experiment, not covered by the other properties.

**Preprocessing**
Information about the experiment preprocessing steps
Cell array containing information about the preprocessing steps used on the data from the experiment.

**PubMedID**
PubMed identifiers for relevant publications.
Character array containing PubMed identifiers for papers relevant to the data set used in the experiment.

**QualityControl**
Information about the experiment quality controls
Cell array containing information about the experiment quality control steps. Information can include replicates, dye swap, and so on.

**Samples**
Information about samples used in the experiment
Cell array containing information about the samples used in the experiment. Information can include sample source, sample organism, treatment type, compound, labeling protocol, external control, and so on.

**Title**

Experiment title

Character array containing a single sentence experiment title.

**URL**

URL for the experiment

Character array containing URL for the experiment.

**Methods**

- `combine`: Combine two MIAME objects
- `isempty`: Determine whether MIAME object is empty

**Instance Hierarchy**

An ExpressionSet object contains a MIAME object.

**Attributes**

To learn about attributes of classes, see Class Attributes (MATLAB).

**Copy Semantics**

Value. To learn how this affects your use of the class, see Copying Objects (MATLAB).

**Examples**

**Construct a MIAME object**

Create a MATLAB structure containing Gene Expression Omnibus (GEO) series data.

```
geoStruct = getgeodata('GSE4616');
```

Import bioma.data package to make the constructor function available.

```
import bioma.data.*
```

Construct MIAME object from the structure.

```
MIAMEObj1 = MIAME(geoStruct)
MIAMEObj1 =
```
Experiment Description:
  Author name: Mika, Silvennoinen
  Riikka, Kivelä
  Maarit, Lehti
  Anna-Maria, Touvras
  Jyrki, Komulainen
  Veikko, Vihko
  Heikki, Kainulainen
Laboratory: LIKES - Research Center
Contact information: Mika, Silvennoinen
URL: www.nonexistinglab.com
PubMedIDs: 17003243
Abstract: A 90 word abstract is available. Use the Abstract property.
Experiment Design: A 234 word summary is available. Use the ExptDesign property.
Other notes:
supply a URL for the MIAME object.

MIAMEObj1.URL = 'www.nonexistinglab.com'

MIAMEObj1 =

Experiment Description:
  Author name: Mika, Silvennoinen
  Riikka, Kivelä
  Maarit, Lehti
  Anna-Maria, Touvras
  Jyrki, Komulainen
  Veikko, Vihko
  Heikki, Kainulainen
Laboratory: LIKES - Research Center
Contact information: Mika, Silvennoinen
URL: www.nonexistinglab.com
PubMedIDs: 17003243
Abstract: A 90 word abstract is available. Use the Abstract property.
Experiment Design: A 234 word summary is available. Use the ExptDesign property.
Other notes:
Alternatively you can construct a MIAME object using customized properties.

MIAMEObj2 = MIAME('investigator', 'Jane Researcher', ...
  'lab', 'One Bioinformatics Laboratory', ...
  'contact', 'jresearcher@lab.not.exist', ...
  'url', 'www.lab.not.exist', ...
  'title', 'Normal vs. Diseased Experiment', ...
  'abstract', 'Example of using expression data', ...
  'other', '{Notes:Created from a text file.}')

MIAMEObj2 =

Experiment Description:
  Author name: Jane Researcher
  Laboratory: One Bioinformatics Laboratory
  Contact information: jresearcher@lab.not.exist
  URL: www.lab.not.exist
  PubMedIDs:
Abstract: A 4 word abstract is available. Use the Abstract property.
No experiment design summary available.
Other notes:
'Notes:Created from a text file.'

See Also
bioma.ExpressionSet | bioma.data.ExptData | bioma.data.MIAME | bioma.data_MetaData | getgeodata

Topics
“Working with Objects for Microarray Experiment Data”
“Analyzing Illumina® Bead Summary Gene Expression Data”
Class Attributes (MATLAB)
Property Attributes (MATLAB)
“Representing Experiment Information in a MIAME Object”
**bioma.ExpressionSet class**

**Package:** bioma

Contain data from microarray gene expression experiment

**Description**

The ExpressionSet class is designed to contain data from a microarray gene expression experiment, including expression values, sample and feature metadata, and information about experimental methods and conditions. It provides a convenient way to store related information about a microarray gene expression experiment in a single data structure (object). It also lets you manage and subset the data.

The ExpressionSet class includes properties and methods that let you access, retrieve, and change data, metadata, and other information about the microarray gene expression experiment. These properties and methods are useful for viewing and analyzing the data.

**Construction**

`ExprSetobj = bioma.ExpressionSet(Data)` creates an ExpressionSet object, from `Data`, a numeric matrix, a DataMatrix object on page 1-532, or an ExptData on page 1-208 object, which contains one or more DataMatrix objects with the same dimensions, row names and column names.

`ExprSetobj = bioma.ExpressionSet(Data, {DMobj1, Name1}, {DMobj2, Name2}, ...)` creates an ExpressionSet object, from `Data`, and additional DataMatrix objects with specified element names. All DataMatrix objects must have the same dimensions, row names, and column names.

`ExprSetobj = bioma.ExpressionSet(..., 'PropertyName', PropertyValue)` constructs the object using options, specified as property name/property value pairs.

`ExprSetobj = bioma.ExpressionSet(..., 'SData', SDataValue)` includes a MetaData on page 1-213 object containing sample metadata in the ExpressionSet object.

`ExprSetobj = bioma.ExpressionSet(..., 'FData', FDataValue)` includes a MetaData on page 1-213 object containing microarray feature metadata in the ExpressionSet object.

`ExprSetobj = bioma.ExpressionSet(..., 'EInfo', EInfoValue)` includes a MIAME on page 1-219 object, which contains experiment information, in the ExpressionSet object.

**Input Arguments**

**Data**

Any of the following:

- Numeric matrix
- DataMatrix object on page 1-532
- ExptData on page 1-208 object, which contains one or more DataMatrix objects having the same dimensions
If you provide a DataMatrix object, `bioma.ExpressionSet` creates an ExptData object from it and names the DataMatrix object `Expressions`. If you provide an ExptData object, `bioma.ExpressionSet` renames the first DataMatrix object in the ExptData object to `Expressions`, unless another DataMatrix object in the ExptData object is already named `Expressions`.

`DMobj#`

Variable name of a DataMatrix object. Each DataMatrix object must have the same dimensions as `Data`.

`Name#`

Character vector or string specifying an element name for the corresponding DataMatrix object. Each DataMatrix object in an ExpressionSet object has an element name. At least one DataMatrix object in an ExpressionSet object has an element name of `Expressions`. By default, it is the first DataMatrix object.

`SDataValue`

Variable name of a MetaData on page 1-213 object containing sample metadata for the experiment. The variable name must exist in the MATLAB Workspace.

`FDataValue`

Variable name of a MetaData on page 1-213 object containing microarray feature metadata for the experiment. The variable name must exist in the MATLAB Workspace.

`EInfoValue`

Variable name of a MIAME on page 1-219 object, which contains information about the experiment methods and conditions. The variable name must exist in the MATLAB Workspace.

**Properties**

**NElements**

Number of elements in the experiment

Positive integer specifying the number of elements (DataMatrix objects) in the experiment data. This value is equivalent to the number of DataMatrix objects in the ExperimentSet object. This information is read-only.

**Attributes:**

`SetAccess` private

**NFeatures**

Number of features in the experiment

Positive integer specifying the number of features in the experiment. This value is equivalent to the number of rows in each DataMatrix object in the ExperimentSet object. This information is read-only.

**Attributes:**
**SetAccess**

private

**NSamples**

Number of samples in the experiment

Positive integer specifying the number of samples in the experiment. This value is equivalent to the number of columns in each DataMatrix object in the ExperimentSet object. This information is read-only.

**Attributes:**

SetAccess private

**Methods**

- abstract: Retrieve or set abstract describing experiment in ExpressionSet object
- elementData: Retrieve or set data element (DataMatrix object) in ExpressionSet object
- elementNames: Retrieve or set element names of DataMatrix objects in ExpressionSet object
- expressions: Retrieve or set Expressions DataMatrix object from ExpressionSet object
- exprWrite: Write expression values in ExpressionSet object to text file
- exptData: Retrieve or set experiment data in ExpressionSet object
- exptInfo: Retrieve or set experiment information in ExpressionSet object
- featureData: Retrieve or set feature metadata in ExpressionSet object
- featureNames: Retrieve or set feature names in ExpressionSet object
- featureVarDesc: Retrieve or set feature variable descriptions in ExpressionSet object
- featureVarNames: Retrieve or set feature variable names in ExpressionSet object
- featureVarValues: Retrieve or set feature variable data values in ExpressionSet object
- pubMedID: Retrieve or set PubMed IDs in ExpressionSet object
- sampleData: Retrieve or set sample metadata in ExpressionSet object
- sampleNames: Retrieve or set sample names in ExpressionSet object
- sampleVarDesc: Retrieve or set sample variable descriptions in ExpressionSet object
- sampleVarNames: Retrieve or set sample variable names in ExpressionSet object
- sampleVarValues: Retrieve or set sample variable values in ExpressionSet object
- size: Return size of ExpressionSet object

**Instance Hierarchy**

An ExpressionSet object contains an ExptData object, two MetaData objects, and a MIAME object. These objects can be empty.

**Attributes**

To learn about attributes of classes, see Class Attributes (MATLAB).
Copy Semantics

Value. To learn how this affects your use of the class, see Copying Objects (MATLAB).

Indexing

ExpressionSet objects support 2-D parenthesis ( ) indexing to extract, assign, and delete data.

ExpressionSet objects do not support:

- Dot . indexing
- Curly brace { } indexing
- Linear indexing

Examples

Construct an ExpressionSet Object

This example shows how to construct an ExpressionSet object. The mouseExprsData.txt file used in this example contains data from Hovatta et al., 2005.

Import bioma.data package to make the constructor function available.

```matlab
import bioma.data.*
```

Create a DataMatrix object from .txt file containing expression values from microarray experiment.

```matlab
dmObj = DataMatrix('File', 'mouseExprsData.txt');
```

Construct an ExptData object.

```matlab
EDObj = ExptData(dmObj)
```

Construct a MetaData object from .txt file.

```matlab
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#')
```

Sample Names:

A, B, ..., Z (26 total)

Variable Names and Meta Information:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gender</td>
<td>Gender of the mouse in study</td>
</tr>
<tr>
<td>Age</td>
<td>The number of weeks since mouse birth</td>
</tr>
<tr>
<td>Type</td>
<td>Genetic characters</td>
</tr>
<tr>
<td>Strain</td>
<td>The mouse strain</td>
</tr>
<tr>
<td>Source</td>
<td>The tissue source for RNA collection</td>
</tr>
</tbody>
</table>
Create a MATLAB structure containing GEO Series data.

```matlab
geoStruct = getgeodata('GSE4616');
```

Construct a MIAME object.

```matlab
MIAMEObj = MIAME(geoStruct)
```

MIAMEObj =

```matlab
Experiment Description:
  Author name: Mika,,Silvennoinen
  Riikka,,KivelÄ"
  Maarit,,Lehti
  Anna-Maria,,Touvras
  Jyrki,,Komulainen
  Veikko,,Vihko
  Heikki,,Kainulainen
  Laboratory: LIKES - Research Center
  Contact information: Mika,,Silvennoinen
  URL:
  PubMedIDs: 17003243
  Abstract: A 90 word abstract is available. Use the Abstract property.
  Experiment Design: A 234 word summary is available. Use the ExptDesign property.
  Other notes:
  [1x84 char]
```

Import bioma package to make constructor function available.

```matlab
import bioma.*
```

Construct an ExpressionSet object.

```matlab
ESObj = ExpressionSet(EDObj, 'SData', MDObj2, 'EInfo', MIAMEObj)
```

```matlab
ESObj =

ExpressionSet
Experiment Data: 500 features, 26 samples
  Element names: Expressions
Sample Data:
  Sample names: A, B, ...,Z (26 total)
  Sample variable names and meta information:
    Gender: Gender of the mouse in study
    Age: The number of weeks since mouse birth
    Type: Genetic characters
    Strain: The mouse strain
    Source: The tissue source for RNA collection
Feature Data: none
Experiment Information: use 'exptInfo(obj)'
```

**References**

See Also
bioma.ExpressionSet | bioma.data.ExptData | bioma.data.MIAME | bioma.data.MetaData | getgeodata

Topics
“Working with Objects for Microarray Experiment Data”
“Analyzing Illumina® Bead Summary Gene Expression Data”
Class Attributes (MATLAB)
Property Attributes (MATLAB)
“Representing Experiment Information in a MIAME Object”
BioRead

Contain sequence reads and their quality data

Description

The BioRead object contains sequencing read data, including sequence headers, nucleotide sequences, and quality scores.

Create a BioRead object from NGS (next-generation sequencing) data stored in an FASTQ- or SAM-formatted file. Each element in the object has a sequence, header, and quality score associated with it. Use the object properties and functions to explore, access, filter, and manipulate all the data or a subset of the data. If you have data with reads that are already mapped to a reference sequence, and you need to access alignment records, use BioMap instead.

Creation

Syntax

bioreadObj = BioRead
bioreadObj = BioRead(File)
bioreadObj = BioRead(S)
bioreadObj = BioRead(Seqs)
bioreadObj = BioRead(Seqs,Quals)
bioreadObj = BioRead(Seqs,Quals,Headers)
bioreadObj = BioRead(___ ,Name,Value)

Description

bioreadObj = BioRead creates an empty BioRead object bioreadObj.

bioreadObj = BioRead(File) creates a BioRead object from File, an FASTQ- or SAM-formatted file. The data remains in the source file after the object is created, and you have access to data through the object properties but cannot modify the properties, except the Name property.

bioreadObj = BioRead(S) creates a BioRead object from S, a MATLAB structure, containing the fields Header, Sequence, and Quality. The data from S remains in memory, and you can modify the properties of the object.

bioreadObj = BioRead(Seqs) creates a BioRead object from Seqs, a cell array of character vectors or string vector containing nucleotide sequences.

bioreadObj = BioRead(Seqs,Quals) creates a BioRead object from Seqs and sets the Quality property of the object to Quals, a cell array of character vectors or string vector containing the ASCII representation of per-base quality scores for each read.

bioreadObj = BioRead(Seqs,Quals,Headers) also sets the Header property of the object to Headers, a cell array of character vectors or string vector containing the header text for each read.
bioreadObj = BioRead(___,'Name',Value) specifies options using one or more name-value pair arguments in addition to input arguments in previous syntaxes. For instance, br = BioRead('SRR005164_1_50.fastq','InMemory',true) specifies to load the data in memory instead of leaving it in the source file.

**Input Arguments**

**File** — Name of FASTQ- or SAM-formatted file  
character vector | string  

Name of FASTQ- or SAM-formatted file, specified as a character vector or string.

The BioRead object accesses data using an auxiliary index file. The index file must have the same name as the source file, but with an .idx extension. If the index file is not in the same folder as the source file, the BioRead function creates the index file in that folder.

**Note** Because the data remains in the source file, do not delete the source file and auxiliary index file.

Example: 'ex1.sam'

Data Types: char

**S** — Sequence information  
structure

Sequence information, specified as a structure. S must contain the fields Header, Sequence, and Quality. For instance, the fastqread and samread functions return such a structure.

Example: S

Data Types: struct

**Seqs** — Nucleotide sequences  
cell array of character vectors | string vector

Nucleotide sequences, specified as a cell array of character vectors or string vector.

Data Types: cell

**Quals** — Sequence quality information  
cell array of character vectors | string vector

Sequence quality information, specified as a cell array of character vectors.

Data Types: cell

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: br = BioRead('SRR005164_1_50.fastq','InMemory',true) specifies to load the data in memory instead of leaving it in the source file.
InMemory — Boolean indicator to keep data in memory
false (default) | true

Boolean indicator to keep data in memory, specified as the comma-separated pair consisting of 'InMemory' and true or false.

When you create a BioRead object from a file, the object does not load the data in memory, but leaves it in the source file and accesses it using an index file to make the process more memory efficient. You cannot modify the object properties if you do not load the data in memory.

If the first input is not a file, this name-value pair argument is ignored, and the data is automatically placed in memory.

Example: 'InMemory',true
Data Types: logical

IndexDir — Path to index file folder
character vector | string

Path to the index file folder where the index file exists or is created, specified as the comma-separated pair consisting of 'IndexDir' and a character vector or string.

Example: 'IndexDir','C:\data\'
Data Types: char

Properties

Header — Header information of reads
cell array of character vectors

Header information of reads, specified as a cell array of character vectors. Each character vector represents the header text for each read. There is a one-to-one relationship between the number and order of character vectors (elements) in the Header and Sequence properties, unless Header is an empty cell array.

Data Types: cell

Name — Object name
character vector | string

Object name, specified as a character vector or string.

Example: 'seqdata'
Data Types: char

NSeqs — Number of reads
positive integer

Number of reads in the object, specified as a positive integer.

Example: 20000
Data Types: double

Quality — Per-base quality scores for all reads

cell array of character vectors
Per-base quality scores for all reads, specified as a cell array of character vectors. Each element is an ASCII representation of per-base quality scores for each read. A one-to-one relationship exists between the number and order of elements in `Quality` and `Sequence`, unless `Quality` is an empty cell array.

Example: `{'<:<<<','<<<7<:'}`

Data Types: `cell`

**Sequence — Nucleotide sequences**

Cell array of character vectors

Nucleotide sequences (reads), specified as a cell array of character vectors.

Example: `{ 'TATCTG', 'ATCTAC' }`

Data Types: `cell`

**Object Functions**

- `combine`: Combine two objects
- `get`: Retrieve property of object
- `getHeader`: Retrieve sequence headers from object
- `getQuality`: Retrieve sequence quality information from object
- `getSequence`: Retrieve sequences from object
- `getSubsequence`: Retrieve partial sequences from object
- `getSubset`: Retrieve subset of elements from object
- `set`: Set property of object
- `setHeader`: Update header information of reads
- `setQuality`: Update quality information
- `setSequence`: Update read sequences
- `setSubsequence`: Update partial sequences
- `setSubset`: Update elements of object
- `write`: Write contents of BioRead or BioMap object to file

**Examples**

**Create BioRead Object from NGS Data**

Create a BioRead object from sequencing read data saved in a FASTQ-formatted file.

```matlab
br = BioRead('SRR005164_1_50.fastq')
```

```
BioRead with properties:

Quality: [50x1 File indexed property]
Sequence: [50x1 File indexed property]
Header: [50x1 File indexed property]
NSeqs: 50
Name: ''
```

By default, when creating a BioRead object from a file, the function also creates an index file if one does not already exist. This example uses an existing index file created and saved in:
The data remains in the source file, and the object accesses the data using the index file, making the process more memory efficient. But you cannot edit the object properties, except the Name property.

To edit the properties, set 'InMemory' to true.

```matlab
brEdit = BioRead('SRR005164_1_50.fastq','InMemory',true);
brEdit.Header(1) = {'SR1'};
```

```matlab
brEdit.Header(1)
ans = 1x1 cell array
    {'SR1'}
```

If you create the object from a MATLAB structure or cell array of nucleotide sequences, the sequence data is always saved in memory by default, and the InMemory option is ignored.

For instance, generate MATLAB variables containing synthetic sequences and quality scores.

```matlab
seqs = {randseq(10);randseq(15);randseq(20)};
quals = {repmat('!',1,10); repmat('%',1,15);repmat('&',1,20)};
headers = {'H1';'H2';'H3'};
```

Create a structure using these variables.

```matlab
structData = struct('Header',headers,'Sequence',seqs,'Quality',quals);
```

Create a BioRead object from the structure.

```matlab
brStruct = BioRead(structData);
```

You can edit the properties of the object because the data remains in memory.

```matlab
brStruct.Header(1) = {'H1.1'};
```

```matlab
brStruct.Header(1)
ans = 1x1 cell array
    {'H1.1'}
```

See Also
BioIndexedFile | BioMap | fastqinfo | fastqread | saminfo | samread | seqqcplot

Topics
“Manage Sequence Read Data in Objects”

External Websites
Sequence Read Archive
SAM format specification

Introduced in R2010a
**blastformat**

Create local BLAST database

**Syntax**

\[
\text{blastformat('Inputdb', InputdbName)} \\
\text{blastformat(..., 'FormatPath', FormatPathValue, ...)} \\
\text{blastformat(..., 'Title', TitleValue, ...)} \\
\text{blastformat(..., 'Log', LogValue, ...)} \\
\text{blastformat(..., 'Protein', ProteinValue, ...)} \\
\text{blastformat(..., 'FormatArgs', FormatArgsValue, ...)}
\]

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>InputdbName</strong></td>
<td>Character vector or string specifying a file name or path and file name of a FASTA file containing a set of sequences to be formatted as a blastable database. If you specify only a file name, that file must be on the MATLAB search path or in the current folder. (This corresponds to the formatdb option -i.)</td>
</tr>
<tr>
<td><strong>FormatPathValue</strong></td>
<td>Character vector or string specifying the full path to the formatdb executable file, including the name and extension of the executable file. Default is the system path.</td>
</tr>
<tr>
<td><strong>TitleValue</strong></td>
<td>Character vector or string specifying the title for the local database. Default is the input FASTA file name. (This corresponds to the formatdb option -t.)</td>
</tr>
<tr>
<td><strong>LogValue</strong></td>
<td>Character vector or string specifying the file name or path and file name for the log file associated with the local database. Default is formatdb.log. (This corresponds to the formatdb option -l.)</td>
</tr>
<tr>
<td><strong>ProteinValue</strong></td>
<td>Specifies whether the sequences formatted as a local BLAST database are protein or not. Choices are true (default) or false. (This corresponds to the formatdb option -p.)</td>
</tr>
<tr>
<td><strong>FormatArgsValue</strong></td>
<td>NCBI formatdb command, that is, a character vector or string containing one or more instances of -x and the option associated with it, used to specify input arguments.</td>
</tr>
</tbody>
</table>

**Description**

**Note** To use the blastformat function, you must have a local copy of the NCBI formatdb executable file available from your system. You can download the formatdb executable file by accessing BLAST executables. Run the downloaded executable and configure it for your system. For convenience, consider placing the NCBI formatdb executable file on your system path.

\[
\text{blastformat('Inputdb', InputdbName)} \text{ calls a local version of the NCBI formatdb executable file with } InputdbName, \text{ a file name or path and file name of a FASTA file containing a set}
\]
of sequences. If you specify only a file name, that file must be on the MATLAB search path or in the current folder. (This corresponds to the `formatdb` option -i.)

It then formats the sequences as a local, blastable database, by creating multiple files, each with the same name as the `InputdbValue` FASTA file, but with different extensions. The database files are placed in the same location as the FASTA file.

**Note** If you rename the database files, make sure they all have the same name.

```
blastformat(..., 'PropertyName', PropertyValue, ...) calls blastformat with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows.

- **FormatPath**: specifies the full path to the `formatdb` executable file, including the name and extension of the executable file. Default is the system path.

- **Title**: specifies the title for the local database. Default is the input FASTA file name. (This corresponds to the `formatdb` option -t.)

  **Note** The 'Title' property does not change the file name of the database files. This title is used internally only, and appears in the report structure returned by the `blastlocal` function.

- **Log**: specifies the file name or path and file name for the log file associated with the local database. Default is `formatdb.log`. The log file captures the progress of the database creation and formatting. (This corresponds to the `formatdb` option -l.)

- **Protein**: specifies whether the sequences formatted as a local BLAST database are protein or not. Choices are true (default) or false. (This corresponds to the `formatdb` option -p.)

- **FormatArgs**: specifies options using the input arguments for the NCBI `formatdb` function. `FormatArgsValue` is a character vector or string containing one or more instances of -x and the option associated with it. For example, to specify that the input is a database in ASN.1 format, instead of a FASTA file, you would use the following syntax:

  ```
  blastformat('Inputdb', 'ecoli.asn', 'FormatArgs', '-a T')
  ```

  **Tip** Use the 'FormatArgs' property to specify `formatdb` options for which there are no corresponding property name/property value pairs.

  **Note** For a complete list of valid input arguments for the NCBI `formatdb` function, make sure that the `formatdb` executable file is located on your system path or current folder, then type the following at your system's command prompt.

  ```
  formatdb -
  ```
Using formatdb Syntax

You can also use the syntax and input arguments accepted by the NCBI formatdb function, instead of the property name/property value pairs listed previously. To do so, supply a character vector or string containing multiple options using the `-x option` syntax. For example, you can specify the `ecoli.nt` FASTA file, a title of `myecoli`, and that the sequences are not protein by using

```r
blastformat('-i ecoli.nt -t myecoli -p F')
```

**Note** For a complete list of valid input arguments for the NCBI formatdb function, make sure that the formatdb executable file is located on your system path or current folder, then type the following at your system's command prompt.

```r
formatdb -
```

Examples

### Example 1.1. Using blastformat with Property Name/Value Pairs

The following example assumes you have a FASTA nucleotide file, such as the *E. coli* file `NC_004431.fna`. For FASTA files from NCBI, visit ftp://ftp.ncbi.nlm.nih.gov/genomes/genbank/bacteria/.

Create a local blastable database from the `NC_004431.fna` FASTA file and give it a title using the 'title' property.

```r
blastformat('inputdb', 'NC_004431.fna', 'protein', 'false',
            'title', 'myecoli_nt');
```

### Example 1.2. Using blastformat with formatdb Syntax and Input Arguments

Create a local blastable database from the `NC_004431.faa` FASTA file and rename the title and log file using formatdb syntax and input arguments.

```r
blastformat('inputdb', 'NC_004431.faa',
            'formatargs', '-t myecoli_aa -l ecoli_aa.log');
```

References


See Also

blastlocal | blastncbi | blastread | blastreadlocal | getblast

Introduced in R2007b
**blastlocal**

Perform search on local BLAST database to create BLAST report

**Syntax**

```
blastlocal('InputQuery', InputQueryValue)
Data = blastlocal('InputQuery', InputQueryValue)
```

... blastlocal(..., 'Program', ProgramValue, ...)
... blastlocal(..., 'Database', DatabaseValue, ...)
... blastlocal(..., 'BlastPath', BlastPathValue, ...)
... blastlocal(..., 'Expect', ExpectValue, ...)
... blastlocal(..., 'Format', FormatValue, ...)
... blastlocal(..., 'ToFile', ToFileValue, ...)
... blastlocal(..., 'Filter', FilterValue, ...)
... blastlocal(..., 'GapOpen', GapOpenValue, ...)
... blastlocal(..., 'GapExtend', GapExtendValue, ...)
... blastlocal(..., 'BLASTArgs', BLASTArgsValue, ...)

**Input Arguments**

<table>
<thead>
<tr>
<th>InputArgument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>InputQueryValue</strong></td>
<td>Character vector or string specifying the file name or path and file name of a FASTA file containing query nucleotide or amino acid sequence(s). (This corresponds to the blastall option -i.)</td>
</tr>
</tbody>
</table>
| **ProgramValue**    | Character vector or string specifying a BLAST program. Choices are:
|                     | • 'blastp' (default) — Search protein query versus protein database.                                                                                                                                       |
|                     | • 'blastn' — Search nucleotide query versus nucleotide database.                                                                                                                                              |
|                     | • 'blastx' — Search translated nucleotide query versus protein database.                                                                                                                                    |
|                     | • 'tblastn' — Search protein query versus translated nucleotide database.                                                                                                                                   |
|                     | • 'tblastx' — Search translated nucleotide query versus translated nucleotide database.                                                                                                                                 |
|                     | (The **ProgramValue** argument corresponds to the blastall option -p.)                                                                                                                                       |
| **DatabaseValue**   | Character vector or string specifying a file name or path and file name of a local BLAST database (formatted using the NCBI formatdb function) to search. Default is a local version of the nr database in the MATLAB current folder. (This corresponds to the blastall option -d.) |
| **BlastPathValue**   | Character vector or string specifying the full path to the blastall executable file, including the name and extension of the executable file. Default is the system path.                                             |
| **ExpectValue** | Value specifying the statistical significance threshold for matches against database sequences. Choices are any real number. Default is 10. (This corresponds to the blastall option -e.) |
| **FormatValue** | Integer specifying the alignment format of the BLAST search results. Choices are:  
- 0 (default) — Pairwise  
- 1 — Query-anchored, showing identities  
- 2 — Query-anchored, no identities  
- 3 — Flat query-anchored, showing identities  
- 4 — Flat query-anchored, no identities  
- 5 — Query-anchored, no identities and blunt ends  
- 6 — Flat query-anchored, no identities and blunt ends  
- 8 — Tabular  
- 9 — Tabular with comment lines  
(This corresponds to the blastall option -m.) |
| **ToFileValue** | Character vector or string specifying a file name or path and file name in which to save the contents of the BLAST report. (This corresponds to the blastall option -o.) |
| **FilterValue** | Controls the application of a filter (DUST filter for the blastn program or SEG filter for other programs) to the query sequence(s). Choices are true (default) or false. (This corresponds to the blastall option -F.) |
| **GapOpenValue** | Integer that specifies the penalty for opening a gap in the alignment of sequences. Default is -1. (This corresponds to the blastall option -G.) |
| **GapExtendValue** | Integer that specifies the penalty for extending a gap in the alignment of sequences. Default is -1. (This corresponds to the blastall option -E.) |
| **BLASTArgsValue** | NCBI blastall command, that is a character vector or string containing one or more instances of -x and the option associated with it, used to specify input arguments. |

### Output Arguments

| **Data** | MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from a local BLAST report. |

### Description

The Basic Local Alignment Search Tool (BLAST) offers a fast and powerful comparative analysis of protein and nucleotide sequences against known sequences in online or local databases.
**Note** To use the `blastlocal` function, you must have a local copy of the NCBI blastall executable file (version 2.2.17) available from your system. Run the downloaded executable and configure it for your system. For convenience, consider placing the NCBI blastall executable file on your system path.

`blastlocal('InputQuery', InputQueryValue)` submits query sequence(s) specified by `InputQueryValue`, a FASTA file containing nucleotide or amino acid sequence(s), for a BLAST search of a local BLAST database, by calling a local version of the NCBI blastall executable file. The BLAST search results are displayed in the MATLAB Command Window. (This corresponds to the `blastall` option `-i`.)

`Data = blastlocal('InputQuery', InputQueryValue)` returns the BLAST search results in `Data`, a MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from a local BLAST report.

`Data` contains a subset of the following fields, based on the specified alignment format.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>NCBI algorithm used to do a BLAST search.</td>
</tr>
<tr>
<td>Query</td>
<td>Identifier of the query sequence submitted to a BLAST search.</td>
</tr>
<tr>
<td>Length</td>
<td>Length of the query sequence.</td>
</tr>
<tr>
<td>Database</td>
<td>All databases searched.</td>
</tr>
<tr>
<td>Hits.Name</td>
<td>Name of a database sequence (subject sequence) that matched the query sequence.</td>
</tr>
<tr>
<td>Hits.Score</td>
<td>Alignment score between the query sequence and the subject sequence.</td>
</tr>
<tr>
<td>Hits.Expect</td>
<td>Expectation value for the alignment between the query sequence and the subject sequence.</td>
</tr>
<tr>
<td>Hits.Length</td>
<td>Length of a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.Score</td>
<td>Pairwise alignment score for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.Expect</td>
<td>Expectation value for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.Identities</td>
<td>Identities (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.Positives</td>
<td>Identical or similar residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject amino acid sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.Gaps</td>
<td>Nonaligned residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
</tbody>
</table>

**Note** This field applies only to translated nucleotide or amino acid query sequences and/or databases.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hits.HSPs.Mismatches</td>
<td>Residues that are not similar to each other (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.Frame</td>
<td>Reading frame of the translated nucleotide sequence for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> This field applies only when performing translated searches, that is, when using tblastx, blastn, and blastx.</td>
</tr>
<tr>
<td>Hits.HSPs.Strand</td>
<td>Sense (Plus = 5’ to 3’ and Minus = 3’ to 5’) of the DNA strands for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> This field applies only when using a nucleotide query sequence and database.</td>
</tr>
<tr>
<td>Hits.HSPs.Alignment</td>
<td>Three-row matrix showing the alignment for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.QueryIndices</td>
<td>Indices of the query sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.SubjectIndices</td>
<td>Indices of the subject sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.AlignmentLength</td>
<td>Length of the pairwise alignment for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Alignment</td>
<td>Entire alignment for the query sequence and the subject sequence(s).</td>
</tr>
<tr>
<td>Statistics</td>
<td>Summary of statistical details about the performed search, such as lambda values, gap penalties, number of sequences searched, and number of hits.</td>
</tr>
</tbody>
</table>

... blastlocal(..., 'PropertyName', PropertyValue, ...) calls blastlocal with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows.

... blastlocal(..., 'Program', ProgramValue, ...) specifies the BLAST program. Choices are 'blastp' (default), 'blastn', 'blastx', 'tblastn', and 'tblastx'. (This corresponds to the blastall option -p.) For help in selecting an appropriate BLAST program, visit: https://blast.ncbi.nlm.nih.gov/producttable.shtml
... blastlocal(..., 'Database', DatabaseValue, ...) specifies the local BLAST database (formatted using the NCBI formatdb function) to search. Default is a local version of the nr database in the MATLAB current folder. (This corresponds to the blastall option -d.)

... blastlocal(..., 'BlastPath', BlastPathValue, ...) specifies the full path to the blastall executable file, including the name and extension of the executable file. Default is the system path.

... blastlocal(..., 'Expect', ExpectValue, ...) specifies a statistical significance threshold for matches against database sequences. Choices are any real number. Default is 10. (This corresponds to the blastall option -e.) You can learn more about the statistics of local sequence comparison at:


... blastlocal(..., 'Format', FormatValue, ...) specifies the alignment format of the BLAST search results. Choices are:

• 0 (default) — Pairwise
• 1 — Query-anchored, showing identities
• 2 — Query-anchored, no identities
• 3 — Flat query-anchored, showing identities
• 4 — Flat query-anchored, no identities
• 5 — Query-anchored, no identities and blunt ends
• 6 — Flat query-anchored, no identities and blunt ends
• 7 — Not used
• 8 — Tabular
• 9 — Tabular with comment lines

(This corresponds to the blastall option -m.)

... blastlocal(..., 'ToFile', ToFileValue, ...) saves the contents of the BLAST report to the specified file. (This corresponds to the blastall option -o.)

... blastlocal(..., 'Filter', FilterValue, ...) specifies whether a filter (DUST filter for the blastn program or SEG filter for other programs) is applied to the query sequence(s). Choices are true (default) or false. (This corresponds to the blastall option -F.)

... blastlocal(..., 'GapOpen', GapOpenValue, ...) specifies the penalty for opening a gap in the alignment of sequences. Default is -1. (This corresponds to the blastall option -G.)

... blastlocal(..., 'GapExtend', GapExtendValue, ...) specifies the penalty for extending a gap in the alignment of sequences. Default is -1. (This corresponds to the blastall option -E.)

... blastlocal(..., 'BLASTArgs', BLASTArgsValue, ...) specifies options using the input arguments for the NCBI blastall function. BLASTArgsValue is a character vector or string containing one or more instances or -x and the option associated with it. For example, to specify the BLOSUM 45 matrix, you would use the following syntax:

blastlocal('InputQuery', ecoliquery.txt, 'BLASTArgs', '-M BLOSUM45')
Tip Use the ‘BlastArgs’ property to specify blastall options for which there are no corresponding property name/property value pairs.

Note For a complete list of valid input arguments for the NCBI blastall function, make sure that the blastall executable file is located on your system path or current folder, then type the following at your system's command prompt:

blastall -

Using blastall Syntax

You can also use the syntax and input arguments accepted by the NCBI blastall function, instead of the property name/property value pairs listed previously. To do so, supply a character vector or string containing multiple options using the -x option syntax. For example, you can specify the ecoliquery.txt FASTA file as your query sequences, the blastp program, and the ecoli local database, by using

blastlocal('-i ecoliquery.txt -p blastp -d ecoli')

Note For a complete list of valid input arguments for the NCBI blastall function, make sure that the blastall executable file is located on your system path or current folder, then type the following at your system's command prompt:

blastall -

Examples

The following examples assume you have a FASTA nucleotide file and a FASTA amino acid file for *E. coli*, such as the files NC_004431.fna and NC_004431.faa, saved to your MATLAB current folder.

**Example 1.3. Performing a Nucleotide Translated Search**

1. Create local blastable databases from the NC_004431.fna and NC_004431.faa FASTA files by using the blastformat function.

   ```matlab
   blastformat('inputdb', 'NC_004431.fna', 'protein', 'false');
   blastformat('inputdb', 'NC_004431.faa');
   ```

2. Use the getgenbank function to retrieve sequence information for the *E. coli* threonine operon from the GenBank database.

   ```matlab
   S = getgenbank('M28570');
   ```

3. Create a query file by using the fastawrite function to create a FASTA file named query_nt.fa from this sequence information, using only the accession number as the header.

   ```matlab
   S.Header = S.Accession;
   fastawrite('query_nt.fa', S);
   ```

4. Use MATLAB syntax to submit the query sequence in the query_nt.fa FASTA file for a BLAST search of the local amino acid database NC_004431.faa. Specify the BLAST program blastx. Return the BLAST search results in results, a MATLAB structure.
results = blastlocal('inputquery', 'query_nt.fa', ...
    'database', 'NC_004431.faa', ...
    'program', 'blastx');

**Example 1.4. Performing a Nucleotide Search Using blastall Syntax**

1. If you have not already done so, create local blastable databases and a query file as described previously.

2. Use blastall syntax to submit the query sequence in the `query_nt.fa` FASTA file for a BLAST search of the local nucleotide database `NC_004431.fna`. Specify the BLAST program `blastn` and an expectation value of 0.0001. Return the BLAST search results in `results`, a MATLAB structure.

```matlab
results = blastlocal('-i query_nt.fa -d NC_004431.fna ... -p blastn -e 0.0001');
```

**Example 1.5. Performing a Nucleotide Search and Creating a Formatted Report**

1. If you have not already done so, create local blastable databases and a query file as described previously.

2. Submit the query sequence in the `query_nt.fa` FASTA file for a BLAST search of the local nucleotide database `NC_004431.fna`. Specify the BLAST program `blastn` and a tabular alignment format. Save the contents of the BLAST report to a file named `myecoli_nt.txt`.

```matlab
blastlocal('inputquery', 'query_nt.fa', ...
    'database', 'NC_004431.fna', 'tofile', ...
    'myecoli_nt.txt', 'blastargs', '-p blastn -m 8');
```

**References**


**See Also**

`blastformat | blastncbi | blastread | blastreadlocal | getblast`

**Introduced in R2007b**
**blastncbi**

Create remote NCBI BLAST report request ID or link to NCBI BLAST report

**Syntax**

blastncbi(Seq,Program)
RID = blastncbi(Seq,Program)
[RID,RTOE] = blastncbi(Seq,Program)
___ = blastncbi(___,Name,Value)

**Description**

blastncbi(Seq,Program) sends a BLAST request to NCBI against Seq, a nucleotide or amino acid sequence, using Program, a specified BLAST program. Then it returns a link to the NCBI BLAST report. For help in selecting an appropriate BLAST program, visit https://blast.ncbi.nlm.nih.gov/producttable.shtml.

RID = blastncbi(Seq,Program) returns RID, the Request ID for the report.

[RID,RTOE] = blastncbi(Seq,Program) returns both RID, the Request ID for the NCBI BLAST report, and RTOE, the Request Time Of Execution, which is an estimated time needed for the search to finish.

___ = blastncbi(___,Name,Value) uses additional options specified by one or more name-value pair arguments, and any of the arguments in the previous syntaxes.

**Examples**

**Perform BLAST search**

Perform a BLAST search on a protein sequence and save the results to an XML file.

Get a sequence from the Protein Data Bank and create a MATLAB structure.

S = getpdb('1CIV');

Use the structure as input for the BLAST search with a significance threshold of 1e-10. The first output is the request ID, and the second output is the estimated time (in minutes) until the search is completed.

[RID1,RTOE] = blastncbi(S,'blastp','expect',1e-10);

Get the search results from the report. You can save the XML-formatted report to a file for an offline access. Use RTOE as the wait time to retrieve the results.

report1 = getblast(RID1,'WaitTime',RTOE,'ToFile','1CIV_report.xml')

Blast results are not available yet. Please wait ...
Use **blastread** to read BLAST data from the XML-formatted BLAST report file.

```matlab
blastdata = blastread('1CIV_report.xml')
```

```matlab
blastdata =

struct with fields:
    RID: 
    Algorithm: 'BLASTP 2.6.1+
    Database: 'nr'
    QueryID: 'Query_224139'
    QueryDefinition: 'unnamed protein product'
    Hits: [1x100 struct]
    Parameters: [1x1 struct]
    Statistics: [1x1 struct]
```

Alternatively, run the BLAST search with an NCBI accession number.

```matlab
RID2 = blastncbi('AAA59174','blastp','expect',1e-10)
```

```matlab
RID2 =
    'R49WAPMH014'
```

Get the search results from the report.

```matlab
report2 = getblast(RID2)
```

```matlab
Blast results are not available yet. Please wait ...
```

```matlab
report2 =

struct with fields:
    RID: 'R49WAPMH014'
    Algorithm: 'BLASTP 2.6.1+
    Database: 'nr'
    QueryID: 'AAA59174.1'
    QueryDefinition: 'insulin receptor precursor [Homo sapiens]
    Hits: [1x100 struct]
    Parameters: [1x1 struct]
Input Arguments

**Seq — Nucleotide or amino acid sequence**
character vector | string | MATLAB structure

Nucleotide or amino acid sequence, specified as a character vector, string, or MATLAB structure containing a `Sequence` field.

If `Seq` is a character vector or string, the available options are:

- GenBank, GenPept, or RefSeq accession number
- Name of a FASTA file
- URL pointing to a sequence file

**Program — BLAST program**
character vector | string

BLAST program, specified as one of the following:

- `'blastn'` — Search nucleotide query versus nucleotide database.
- `'blastp'` — Search protein query versus protein database.
- `'blastx'` — Search (translated) nucleotide query versus protein database.
- `'megablast'` — Search for highly similar nucleotide sequences.
- `'tblastn'` — Search protein query versus translated nucleotide database.
- `'tblastx'` — Search (translated) nucleotide query versus (translated) nucleotide database.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Matrix','PAM70','Expect',1e-10` uses the PAM70 substitution matrix with the significance threshold for matches set to 1e-10.

**Database — Database to search**

- `'nr'` (default) | character vector | string

Database to search, specified as the comma-separated pair consisting of `'Database'` and a character vector or string.

For nucleotide databases, valid choices are:

- `'nr'` (default)
- `'refseq_rna'`
- `'refseq_genomic'`
- `'est'`
• 'est_human'
• 'est_mouse'
• 'est_others'
• 'gss'
• 'htgs'
• 'pat'
• 'pdb'
• 'alu'
• 'dbsts'
• 'chromosome'

For protein databases, valid choices are:

• 'nr' (default)
• 'refseq_protein'
• 'swissprot'
• 'pat'
• 'pdb'
• 'env_nr'

**Note** Available databases may change. Check the NCBI website for more information.

For help in selecting an appropriate database, visit


**MaxNumberSequences** — Maximum number of hits to return

100 (default) | positive integer

Maximum number of hits to return, specified as the comma-separated pair consisting of 'MaxNumberSequences' and a positive integer. The actual search results may have fewer hits than what you specify, depending on the query, database, expectation value, and other parameters. The default value is 100.

**Filter** — Filter applied to query sequence

character vector | string

Filter applied to the query sequence, specified as the comma-separated pair consisting of 'Filter' and one of the following:

• 'L' — Mask regions of low compositional complexity.
• 'R' — Mask human repeat elements (valid for blastn and megablast only).
• 'm' — Mask the query while producing blast seeds, but not during extension.
• 'none' — No mask is applied.
• ‘l’ — Mask any letter that is lowercase in the query.

You can specify multiple valid letters in a single character vector or string to apply multiple filters at once. For example, ‘Lm’ applies both the low compositional complexity filter and the mask.

Choices vary depending on the selected Program. For more information, see the table Choices for Optional Properties by BLAST Program.

**Expect — Statistical significance threshold for matches**

10 (default) | positive real number

Statistical significance threshold for matches against database sequences, specified as the comma-separated pair consisting of ‘Expect’ and a positive real number. The default is 10.

You can learn more about the statistics of local sequence comparison at https://blast.ncbi.nlm.nih.gov/tutorial/Altschul-1.html#head2.

**Word — Word length for query sequence**

positive integer

Word length for the query sequence, specified as the comma-separated pair consisting of ‘Word’ and a positive integer.

Choices for a protein query search are:

• 2
• 3 (default)

Choices for a nucleotide query search are:

• 7
• 11 (default)
• 15

Choices when Program is set to ‘megablast’ are:

• 16
• 20
• 24
• 28 (default)
• 32
• 48
• 64
• 128

**Matrix — Substitution matrix for amino acid sequences**

‘BLOSUM62’ (default) | character vector | string

Substitution matrix for amino acid sequences, specified as the comma-separated pair consisting of ‘Matrix’ and a character vector or string. The matrix assigns the score for a possible alignment of any two amino acid residues. Choices are:
• 'PAM30'
• 'PAM70'
• 'BLOSUM45'
• 'BLOSUM62' (default)
• 'BLOSUM80'

**MatchScores — Matching and mismatching scores in nucleotide alignment**

two-element numeric vector

Matching and mismatching scores in a nucleotide alignment, specified as the comma-separated pair consisting of 'MatchScores' and a two-element numeric vector [R Q]. The first element R is the match score, and the second element Q is the mismatch score. This option is for blastn and megablast only.

To ensure accurate evaluation of the alignment significance, only a limited set of combinations are supported. See the table "BLAST Optional Properties" on page 1-252 for all the supported values. The default value for megablast is [1 -2], and the default value for blastn is [1 -3].

**GapCosts — Penalties for opening and extending gap**

two-element numeric vector

Penalties for opening and extending a gap, specified as the comma-separated pair consisting of 'GapCosts' and a two-element numeric vector. The vector contains two integers: the first is the penalty for opening a gap, and the second is the penalty for extending a gap.

Valid gap costs for blastp, blastx, tblastn, and tblastx vary according to the protein substitution matrix. For details, see GapCosts for blastp, blastx, tblastn, and tblastx.

Valid gap costs for blastn and megablast vary according to MatchScores ([R Q]). For details, see GapCosts for blastn and megablast.

**CompositionAdjustment — Compositional adjustment type to compensate for amino acid compositions**

'none' (default) | 'cbs' | 'ccsm' | 'ucsm'

Compositional adjustment type to compensate for the amino acid compositions of the sequences being compared, specified as the comma-separated pair consisting of 'CompositionAdjustment' and one of the following values:

• 'none' — No adjustment is applied (default).
• 'cbs' — Composition-based statistics approach is used for score adjustments.
• 'ccsm' — Conditional compositional score matrix is used for score adjustments.
• 'ucsm' — Universal compositional score matrix is used for score adjustments.

This option is for blastp, blastx, and tblastn only. The resulting scaled scores yield more accurate E-values than the standard, unscaled scores. For details, see Compositional adjustments.

**Entrez — Entrez query syntax to search a subset of selected database**

character vector | string

Entrez query syntax to search a subset of the selected database, specified as the comma-separated pair consisting of 'Entrez' and a character vector or string. Use this option to limit searches based
on molecule types, sequence lengths, organisms, and so on. For more information on limiting searches, see https://blast.ncbi.nlm.nih.gov/blastcgihelp.shtml#entrez_query.

**Adv — Advanced options**

character vector | string

Advanced options, specified as the comma-separated pair consisting of 'Adv' and a character vector or string. For instance, to specify the reward and penalty values for nucleotide matches and mismatches, use '-r 1 -q -3'. For more information, see https://www.ncbi.nlm.nih.gov/blast/Doc/urlapi.html.

**Output Arguments**

**RID — Request ID for NCBI BLAST report**

character vector

Request ID for the NCBI BLAST report, returned as a character vector.

**RTOE — Request time of execution**

integer

Request time of execution, returned as an integer. This is an estimated time in minutes until the search is completed.

**Tip** If you use the getblast function to retrieve the BLAST report, use this time estimate as the 'WaitTime' option.

**More About**

**BLAST Optional Properties**

<table>
<thead>
<tr>
<th>Choices for Optional Properties by BLAST Program</th>
<th>Choices for Optional Properties by BLAST Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>When BLAST is...</td>
<td>Then choices for the following options</td>
</tr>
<tr>
<td>T program</td>
<td>Gap Costs</td>
</tr>
<tr>
<td>database filter</td>
<td>Match scores</td>
</tr>
<tr>
<td>strategy</td>
<td>Request ID for NCBI BLAST report</td>
</tr>
<tr>
<td>databases</td>
<td>Request time of execution</td>
</tr>
<tr>
<td>Options</td>
<td>RID</td>
</tr>
<tr>
<td></td>
<td>RTOE</td>
</tr>
<tr>
<td></td>
<td>Tips</td>
</tr>
<tr>
<td></td>
<td>More About</td>
</tr>
<tr>
<td></td>
<td>BLAST Optional Properties</td>
</tr>
</tbody>
</table>

**Choices for Optional Properties by BLAST Program**

<table>
<thead>
<tr>
<th>Choices for Optional Properties by BLAST Program</th>
<th>Choices for Optional Properties by BLAST Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>When BLAST is...</td>
<td>Then choices for the following options</td>
</tr>
<tr>
<td>T program</td>
<td>Gap Costs</td>
</tr>
<tr>
<td>database filter</td>
<td>Match scores</td>
</tr>
<tr>
<td>strategy</td>
<td>Request ID for NCBI BLAST report</td>
</tr>
<tr>
<td>databases</td>
<td>Request time of execution</td>
</tr>
<tr>
<td>Options</td>
<td>RID</td>
</tr>
<tr>
<td></td>
<td>RTOE</td>
</tr>
<tr>
<td></td>
<td>Tips</td>
</tr>
<tr>
<td></td>
<td>More About</td>
</tr>
<tr>
<td></td>
<td>BLAST Optional Properties</td>
</tr>
</tbody>
</table>

1-252
<table>
<thead>
<tr>
<th>When BLAST program is...</th>
<th>Then choices for the following options are...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Database</td>
</tr>
<tr>
<td>'tblastx'</td>
<td></td>
</tr>
<tr>
<td>'blastp'</td>
<td></td>
</tr>
<tr>
<td>'blastx'</td>
<td></td>
</tr>
</tbody>
</table>

**Table:**
- **Database:** Options include 'nr', 'refseq_prot', 'swiss_prot', 'pat', 'pdb', 'env_nr', 'L' (default) 'm', 'l', 'none'
- **Filter:** Options include 'm', 'l', 'none'
- **Word:** Options include 3 (default)
- **Matrix:** Options include 'BLOSUM 45', 'BLOSUM 62' (default), 'BLOSUM 80'
- **Gap Costs:** Options include 'BLASTp', 'BLASTx', 'tblastn', and 'tblastx'
### GapCosts for blastp, blastx, tblastn, and tblastx

<table>
<thead>
<tr>
<th>Substitution Matrix</th>
<th>Valid 'GapCosts' Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'PAM30'</td>
<td>[7 2]</td>
</tr>
<tr>
<td></td>
<td>[6 2]</td>
</tr>
<tr>
<td></td>
<td>[5 2]</td>
</tr>
<tr>
<td></td>
<td>[10 1]</td>
</tr>
<tr>
<td></td>
<td>[9 1] (default)</td>
</tr>
<tr>
<td></td>
<td>[8 1]</td>
</tr>
<tr>
<td>'PAM70'</td>
<td>[8 2]</td>
</tr>
<tr>
<td></td>
<td>[7 2]</td>
</tr>
<tr>
<td>'BLOSUM80'</td>
<td>[6 2]</td>
</tr>
<tr>
<td></td>
<td>[11 1]</td>
</tr>
<tr>
<td></td>
<td>[10 1] (default)</td>
</tr>
<tr>
<td></td>
<td>[9 1]</td>
</tr>
<tr>
<td>'BLOSUM45'</td>
<td>[13 3]</td>
</tr>
<tr>
<td></td>
<td>[12 3]</td>
</tr>
<tr>
<td></td>
<td>[11 3]</td>
</tr>
<tr>
<td></td>
<td>[10 3]</td>
</tr>
<tr>
<td></td>
<td>[15 2] (default)</td>
</tr>
<tr>
<td></td>
<td>[14 2]</td>
</tr>
<tr>
<td></td>
<td>[13 2]</td>
</tr>
<tr>
<td></td>
<td>[12 2]</td>
</tr>
<tr>
<td></td>
<td>[19 1]</td>
</tr>
<tr>
<td></td>
<td>[18 1]</td>
</tr>
<tr>
<td></td>
<td>[17 1]</td>
</tr>
<tr>
<td></td>
<td>[16 1]</td>
</tr>
<tr>
<td>'BLOSUM62'</td>
<td>[9 2]</td>
</tr>
<tr>
<td></td>
<td>[8 2]</td>
</tr>
<tr>
<td></td>
<td>[7 2]</td>
</tr>
<tr>
<td></td>
<td>[12 1]</td>
</tr>
<tr>
<td></td>
<td>[11 1] (default)</td>
</tr>
<tr>
<td></td>
<td>[10 1]</td>
</tr>
</tbody>
</table>
## GapCosts for blastn and megablast

<table>
<thead>
<tr>
<th>MatchScores [R Q]</th>
<th>Valid 'GapCosts' Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1 -4]</td>
<td>[5 2] (default)</td>
</tr>
<tr>
<td></td>
<td>[1 2]</td>
</tr>
<tr>
<td></td>
<td>[0 2]</td>
</tr>
<tr>
<td></td>
<td>[2 1]</td>
</tr>
<tr>
<td></td>
<td>[1 1]</td>
</tr>
<tr>
<td>[1 -3]</td>
<td>[5 2] (default)</td>
</tr>
<tr>
<td></td>
<td>[2 2]</td>
</tr>
<tr>
<td></td>
<td>[1 2]</td>
</tr>
<tr>
<td></td>
<td>[0 2]</td>
</tr>
<tr>
<td></td>
<td>[2 1]</td>
</tr>
<tr>
<td></td>
<td>[1 1]</td>
</tr>
<tr>
<td>[1 -2]</td>
<td>[5 2] (default)</td>
</tr>
<tr>
<td></td>
<td>[2 2]</td>
</tr>
<tr>
<td></td>
<td>[1 2]</td>
</tr>
<tr>
<td></td>
<td>[0 2]</td>
</tr>
<tr>
<td></td>
<td>[3 1]</td>
</tr>
<tr>
<td></td>
<td>[2 1]</td>
</tr>
<tr>
<td></td>
<td>[1 1]</td>
</tr>
<tr>
<td>[1 -1]</td>
<td>[5 2] (default)</td>
</tr>
<tr>
<td></td>
<td>[3 2]</td>
</tr>
<tr>
<td></td>
<td>[2 2]</td>
</tr>
<tr>
<td></td>
<td>[1 2]</td>
</tr>
<tr>
<td></td>
<td>[0 2]</td>
</tr>
<tr>
<td></td>
<td>[4 1]</td>
</tr>
<tr>
<td></td>
<td>[3 1]</td>
</tr>
<tr>
<td></td>
<td>[2 1]</td>
</tr>
<tr>
<td>[2 -3]</td>
<td>[5 2] (default)</td>
</tr>
<tr>
<td></td>
<td>[4 4]</td>
</tr>
<tr>
<td></td>
<td>[2 4]</td>
</tr>
<tr>
<td></td>
<td>[0 4]</td>
</tr>
<tr>
<td></td>
<td>[3 3]</td>
</tr>
<tr>
<td></td>
<td>[6 2]</td>
</tr>
<tr>
<td></td>
<td>[4 2]</td>
</tr>
<tr>
<td></td>
<td>[2 2]</td>
</tr>
<tr>
<td>[4 -5]</td>
<td>[5 2] (default)</td>
</tr>
<tr>
<td></td>
<td>[6 5]</td>
</tr>
<tr>
<td></td>
<td>[5 5]</td>
</tr>
<tr>
<td></td>
<td>[4 5]</td>
</tr>
<tr>
<td></td>
<td>[3 5]</td>
</tr>
</tbody>
</table>

## Compatibility Considerations

'psiblast' BLAST program has been removed  
*Errors starting in R2017b*

The BLAST program 'psiblast' has been removed from one of supported programs.
'Inclusion' option has been removed
Errors starting in R2017b

The 'Inclusion' name-value pair has been removed since it only applies to the psiblast program which has been also removed.

'Descriptions' option has been removed
Errors starting in R2017b

The 'Descriptions' name-value pair has been removed. Use 'MaxNumberSequences' instead to specify the maximum number of hits to return.

'Alignments' option has been removed
Errors starting in R2017b

The 'Alignments' name-value pair has been removed. Use 'MaxNumberSequences' instead to specify the maximum number of hits to return.

'GapOpen' option has been removed
Errors starting in R2017b

The 'GapOpen' name-value pair has been removed. Use 'GapCosts' instead.

'ExtendGap' option has been removed
Errors starting in R2017b

The 'ExtendGap' name-value pair has been removed. Use 'GapCosts' instead.

'Pct' option has been removed
Errors starting in R2017b

The 'Pct' name-value pair has been removed.

References


See Also
blastformat | blastlocal | blastread | blastreadlocal | getblast

External Websites
BLAST Help
Entrez Help

Introduced before R2006a
blastread

Read data from NCBI BLAST report file

Syntax

blastdata = blastread(blastreport)

Description

blastdata = blastread(blastreport) reads the NCBI BLAST report data from an XML-formatted file, blastreport, and returns blastdata, a structure containing the corresponding BLAST data.

Examples

Perform BLAST search

Perform a BLAST search on a protein sequence and save the results to an XML file.

Get a sequence from the Protein Data Bank and create a MATLAB structure.

S = getpdb('1CIV);

Use the structure as input for the BLAST search with a significance threshold of 1e-10. The first output is the request ID, and the second output is the estimated time (in minutes) until the search is completed.

[RID1,ROTE] = blastncbi(S,'blastp','expect',1e-10);

Get the search results from the report. You can save the XML-formatted report to a file for an offline access. Use ROTE as the wait time to retrieve the results.

report1 = getblast(RID1,'WaitTime',ROTE,'ToFile','1CIV_report.xml')

Blast results are not available yet. Please wait ...

report1 =

struct with fields:

    RID: 'R49TJMCFO14'
    Algorithm: 'BLASTP 2.6.1+'
    Database: 'nr'
    QueryID: 'Query_224139'
    QueryDefinition: 'unnamed protein product'
    Hits: [1x100 struct]
    Parameters: [1x1 struct]
    Statistics: [1x1 struct]

Use blastread to read BLAST data from the XML-formatted BLAST report file.
blastdata = blastread('ICIV_report.xml')

blastdata =
    struct with fields:
    
    RID: '
    Algorithm: 'BLASTP 2.6.1+'
    Database: 'nr'
    QueryID: 'Query_224139'
    QueryDefinition: 'unnamed protein product'
    Hits: [1x100 struct]
    Parameters: [1x1 struct]
    Statistics: [1x1 struct]

Alternatively, run the BLAST search with an NCBI accession number.

RID2 = blastncbi('AAA59174','blastp','expect',1e-10)

RID2 =
    'R49WAPMH014'

Get the search results from the report.

report2 = getblast(RID2)

Blast results are not available yet. Please wait ...

report2 =
    struct with fields:
    
    RID: 'R49WAPMH014'
    Algorithm: 'BLASTP 2.6.1+'
    Database: 'nr'
    QueryID: 'AAA59174.1'
    QueryDefinition: 'insulin receptor precursor [Homo sapiens]'
    Hits: [1x100 struct]
    Parameters: [1x1 struct]
    Statistics: [1x1 struct]

**Input Arguments**

**blastreport** — Name of BLAST report file
character vector | string

Name of an XML-formatted BLAST report file, specified as a character vector or string.
Example: 'blastreport.xml'
**Output Arguments**

**blastdata — BLAST report data**

structure

BLAST report data, returned as a structure that contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID</td>
<td>Request ID for retrieving results from a specific NCBI BLAST search</td>
</tr>
<tr>
<td>Algorithm</td>
<td>NCBI algorithm used to perform the BLAST search</td>
</tr>
<tr>
<td>Database</td>
<td>All databases searched</td>
</tr>
<tr>
<td>QueryID</td>
<td>Identifier of the query sequence</td>
</tr>
<tr>
<td>QueryDefinition</td>
<td>Definition of the query sequence</td>
</tr>
<tr>
<td>Hits</td>
<td>Structure containing information on the hit sequences, such as IDs, accession numbers, lengths, and HSPs (high-scoring segment pairs)</td>
</tr>
<tr>
<td>Parameters</td>
<td>Structure containing information on the input parameters used to perform the search</td>
</tr>
<tr>
<td>Statistics</td>
<td>Summary of statistical details about the performed search, such as lambda, kappa, and entropy values</td>
</tr>
</tbody>
</table>

**More About**

**Hits**

This table lists each field of blastdata.Hits.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>ID of the subject sequence that matched the query sequence</td>
</tr>
<tr>
<td>Definition</td>
<td>Description of the subject sequence</td>
</tr>
<tr>
<td>Accession</td>
<td>Accession of the subject sequence</td>
</tr>
<tr>
<td>Length</td>
<td>Length of the subject sequence</td>
</tr>
<tr>
<td>Hsp</td>
<td>Structure containing Information on the high-scoring segment pairs (HSPs)</td>
</tr>
</tbody>
</table>

**Hits.Hsp**

This table summarizes the fields of Hits.Hsp.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>Pairwise alignment score for a high-scoring segment pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>BitScore</td>
<td>Bit score for a high-scoring segment pair.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Expect</td>
<td>Expectation value for a high-scoring segment pair.</td>
</tr>
<tr>
<td>Identities</td>
<td>Number of identical or similar residues for a high-scoring segment pair.</td>
</tr>
<tr>
<td>Positives</td>
<td>Number of identical or similar residues for a high-scoring sequence pair.</td>
</tr>
<tr>
<td></td>
<td>This field applies only to translated nucleotide or amino acid query sequences and databases.</td>
</tr>
<tr>
<td>Gaps</td>
<td>Nonaligned residues for a high-scoring segment pair.</td>
</tr>
<tr>
<td>AlignmentLength</td>
<td>Length of the alignment for a high-scoring segment pair.</td>
</tr>
<tr>
<td>QueryIndices</td>
<td>Indices of the query sequence residue positions for a high-scoring segment pair.</td>
</tr>
<tr>
<td>SubjectIndices</td>
<td>Indices of the subject sequence residue positions for a high-scoring segment pair.</td>
</tr>
<tr>
<td>Frame</td>
<td>Reading frame of the translated nucleotide sequence for a high-scoring segment pair.</td>
</tr>
<tr>
<td>Alignment</td>
<td>3-by-N character array showing the alignment for a high-scoring sequence pair.</td>
</tr>
</tbody>
</table>

**See Also**

blastformat | blastlocal | blastncbi | blastreadlocal | getblast

**Introduced before R2006a**
**blastreadlocal**

Read data from local BLAST report

**Syntax**

\[ \text{Data} = \text{blastreadlocal}(\text{BLASTReport}, \text{Format}) \]

**Input Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BLASTReport</strong></td>
<td>BLAST report specified by any of the following:</td>
</tr>
<tr>
<td></td>
<td>• File name or path and file name of a locally created BLAST report file, such as returned by the <code>blastlocal</code> function with the <code>'ToFile'</code> property.</td>
</tr>
<tr>
<td></td>
<td>• One-dimensional character array or string that contains the text for a local BLAST report.</td>
</tr>
<tr>
<td></td>
<td>If you specify only a file name, that file must be on the MATLAB search path or in the current folder.</td>
</tr>
<tr>
<td><strong>Format</strong></td>
<td>Integer specifying the alignment format used to create <code>BLASTReport</code>. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 0 — Pairwise</td>
</tr>
<tr>
<td></td>
<td>• 1 — Query-anchored, showing identities</td>
</tr>
<tr>
<td></td>
<td>• 2 — Query-anchored, no identities</td>
</tr>
<tr>
<td></td>
<td>• 3 — Flat query-anchored, showing identities</td>
</tr>
<tr>
<td></td>
<td>• 4 — Flat query-anchored, no identities</td>
</tr>
<tr>
<td></td>
<td>• 5 — Query-anchored, no identities and blunt ends</td>
</tr>
<tr>
<td></td>
<td>• 6 — Flat query-anchored, no identities and blunt ends</td>
</tr>
<tr>
<td></td>
<td>• 7 — Not used</td>
</tr>
<tr>
<td></td>
<td>• 8 — Tabular</td>
</tr>
<tr>
<td></td>
<td>• 9 — Tabular with comment lines</td>
</tr>
</tbody>
</table>

**Output Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong></td>
<td>MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from a local BLAST report.</td>
</tr>
</tbody>
</table>

**Description**

The Basic Local Alignment Search Tool (BLAST) offers a fast and powerful comparative analysis of protein and nucleotide sequences against known sequences in online and local databases. BLAST reports can be lengthy, and parsing the data from the various formats can be cumbersome.
**Data** = blastreadlocal(`BLASTReport`, `Format`) reads `BLASTReport`, a locally created BLAST report file, and returns `Data`, a MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from a local BLAST report. `Format` is an integer specifying the alignment format used to create `BLASTReport`.

**Note** The function assumes the BLAST report was produced using version 2.2.17 of the `blastall` executable.

`Data` contains a subset of the following fields, based on the specified alignment format.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>NCBI algorithm used to do a BLAST search.</td>
</tr>
<tr>
<td>Query Identifier</td>
<td>Identifier of the query sequence submitted to a BLAST search.</td>
</tr>
<tr>
<td>Length</td>
<td>Length of the query sequence.</td>
</tr>
<tr>
<td>Database</td>
<td>All databases searched.</td>
</tr>
<tr>
<td>Hits.Name</td>
<td>Name of a database sequence (subject sequence) that matched the query sequence.</td>
</tr>
<tr>
<td>Hits.Score</td>
<td>Alignment score between the query sequence and the subject sequence.</td>
</tr>
<tr>
<td>Hits.Expect</td>
<td>Expectation value for the alignment between the query sequence and the subject sequence.</td>
</tr>
<tr>
<td>Hits.Length</td>
<td>Length of a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.Score</td>
<td>Pairwise alignment score for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.Expect</td>
<td>Expectation value for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.Identities</td>
<td>Identities (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.Positives</td>
<td>Identical or similar residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject amino acid sequence.</td>
</tr>
<tr>
<td>Note</td>
<td>This field applies only to translated nucleotide or amino acid query sequences and/or databases.</td>
</tr>
<tr>
<td>Hits.HSPs.Gaps</td>
<td>Nonaligned residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Hits.HSPs.Mismatches</td>
<td>Residues that are not similar to each other (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.Frame</td>
<td>Reading frame of the translated nucleotide sequence for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> This field applies only when performing translated searches, that is, when using tblastx, tblastn, and blastx.</td>
</tr>
<tr>
<td>Hits.HSPs.Strand</td>
<td>Sense (Plus = 5' to 3' and Minus = 3' to 5') of the DNA strands for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> This field applies only when using a nucleotide query sequence and database.</td>
</tr>
<tr>
<td>Hits.HSPs.Alignment</td>
<td>Three-row matrix showing the alignment for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.QueryIndices</td>
<td>Indices of the query sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.SubjectIndices</td>
<td>Indices of the subject sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Hits.HSPs.AlignmentLength</td>
<td>Length of the pairwise alignment for a high-scoring sequence pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Alignment</td>
<td>Entire alignment for the query sequence and the subject sequence(s).</td>
</tr>
<tr>
<td>Statistics</td>
<td>Summary of statistical details about the performed search, such as lambda values, gap penalties, number of sequences searched, and number of hits.</td>
</tr>
</tbody>
</table>

**Examples**

The following examples assume you have a FASTA nucleotide file for *E. coli*, such as the file NC_004431.fna.

**Example 1.6. Reading Data Using a Tabular Alignment Format**

1. Create a local blastable database from the NC_004431.fna FASTA file.
blastformat('inputdb', 'NC_004431.fna', 'protein', 'false');

2 Use the getgenbank function to retrieve two sequences from the GenBank database.

S1 = getgenbank('M28570.1');
S2 = getgenbank('M12565');

3 Create a query file by using the fastawrite function to create a FASTA file named query_multi_nt.fa from these two sequences, using the only accession number as the header.

Seqs(1).Header = S1.Accession;
Seqs(1).Sequence = S1.Sequence;
Seqs(2).Header = S2.Accession;
Seqs(2).Sequence = S2.Sequence;
fastawrite('query_multi_nt.fa', Seqs);

4 Submit the query sequences in the query_multi_nt.fa FASTA file for a BLAST search of the local nucleotide database NC_004431.fna. Specify the BLAST program blastn and a tabular alignment format. Save the contents of the BLAST report to a file named myecoli_nt8.txt, and then read the local BLAST report.

blastlocal('inputquery', 'query_multi_nt.fa',... 
'database', 'NC_004431.fna',... 
'tofile', 'myecoli_nt8.txt', 'program', 'blastn',... 
'format', 8);
blastreadlocal('myecoli_nt8.txt', 8);

Example 1.7. Reading Data Using a Query Anchored Format

1 If you have not already done so, create a local blastable database and a query file as described previously.

2 Submit the query sequences in the query_multi_nt.fa FASTA file for a BLAST search of the local nucleotide database NC_004431.fna. Specify the BLAST program blastn and a query-anchored format. Save the contents of the BLAST report to a file named myecoli_nt1.txt, and then read the local BLAST report, saving the results in results, an array of structures.

blastlocal('inputquery', 'query_multi_nt.fa',... 
'database', 'NC_004431.fna',... 
'tofile', 'myecoli_nt1.txt', 'program', 'blastn',... 
'format', 1);
results = blastreadlocal('myecoli_nt1.txt', 1);

References


For more information about reading and interpreting BLAST reports, see:

See Also
blastformat | blastlocal | blastncbi | blastread | getblast
Introduced in R2007b
blosum

Return BLOSUM scoring matrix

Syntax

Matrix = blosum(Identity)
[Matrix, MatrixInfo] = blosum(Identity)

... = blosum(Identity, ...'Extended', ExtendedValue, ...)
... = blosum(Identity, ...'Order', OrderValue, ...)

Input Arguments

<table>
<thead>
<tr>
<th>Identity</th>
<th>Scalar specifying a percent identity level. Choices are:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Values from 30 to 90 in increments of 5</td>
</tr>
<tr>
<td></td>
<td>• 62</td>
</tr>
<tr>
<td></td>
<td>• 100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ExtendedValue</th>
<th>Controls the listing of extended amino acid codes. Choices are true (default) or false.</th>
</tr>
</thead>
</table>

| OrderValue    | Character vector or string containing legal amino acid characters that specifies the order amino acids are listed in the matrix. The length of the character vector or string must be 20 or 24. |

Output Arguments

<table>
<thead>
<tr>
<th>Matrix</th>
<th>BLOSUM (Blocks Substitution Matrix) scoring matrix with a specified percent identity.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatrixInfo</td>
<td>Structure of information about Matrix containing the following fields:</td>
</tr>
<tr>
<td></td>
<td>• Name</td>
</tr>
<tr>
<td></td>
<td>• Scale</td>
</tr>
<tr>
<td></td>
<td>• Entropy</td>
</tr>
<tr>
<td></td>
<td>• ExpectedScore</td>
</tr>
<tr>
<td></td>
<td>• HighestScore</td>
</tr>
<tr>
<td></td>
<td>• LowestScore</td>
</tr>
<tr>
<td></td>
<td>• Order</td>
</tr>
</tbody>
</table>

Description

Matrix = blosum(Identity) returns a BLOSUM (Blocks Substitution Matrix) scoring matrix with a specified percent identity. The default ordering of the output includes the extended characters B, Z, X, and *.

A R N D C Q E G H I L K M F P S T W Y V B Z X *
[Matrix, MatrixInfo] = blosum(Identity) returns MatrixInfo, a structure of information about Matrix, a BLOSUM matrix. MatrixInfo contains the following fields:

- Name
- Scale
- Entropy
- ExpectedScore
- HighestScore
- LowestScore
- Order

... = blosum(Identity, ...'PropertyName', PropertyValue, ...) calls blosum with optional properties that use property name/property value pairs. You can specify one or more properties in any order. EachPropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = blosum(Identity, ...'Extended', ExtendedValue, ...) controls the listing of extended amino acid codes. Choices are true (default) or false. If ExtendedValue is false, returns the scoring matrix for the standard 20 amino acids. Ordering of the output when ExtendedValue is false is

A R N D C Q E G H I L K M F P S T W Y V

... = blosum(Identity, ...'Order', OrderValue, ...) returns a BLOSUM matrix ordered by OrderValue, a character vector or string containing legal amino acid characters that specifies the order amino acids are listed in the matrix. The length of the character vector or string must be 20 or 24.

Examples

Retrieve BLOSUM scoring matrix

Return a BLOSUM matrix with a percent identity level of 50.

B50 = blosum(50)

B50 = 24×24

Return a BLOSUM matrix with the amino acids in a specific order.
B75 = blosum(75,'Order','CSTPAGNDEQHRKMILVFYW')

B75 = 20×20

<table>
<thead>
<tr>
<th></th>
<th>9</th>
<th>-1</th>
<th>-4</th>
<th>-1</th>
<th>-3</th>
<th>-3</th>
<th>-4</th>
<th>-5</th>
<th>-3</th>
<th>-4</th>
<th>-4</th>
<th>-4</th>
<th>-2</th>
<th>-1</th>
<th>-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>5</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-4</td>
<td>-1</td>
<td>-1</td>
<td>8</td>
<td>-1</td>
<td>-3</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>4</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>-2</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-4</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-3</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-2</td>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See Also
dayhoff | gonnet | localalign | nuc44 | nalign | pam | swalign

Introduced before R2006a
bowtie

Map short reads to reference sequence using Burrows-Wheeler transform

Syntax

bowtie(indexBaseName, reads, outputFileName)
bowtie(indexBaseName, reads, outputFileName, Name, Value)

Description

bowtie(indexBaseName, reads, outputFileName) aligns the reads specified in reads to the indexed reference specified by indexBaseName, and writes the results to the BAM-formatted file outputFileName.

Note  bowtie runs on Mac and UNIX® platforms only.

bowtie(indexBaseName, reads, outputFileName, Name, Value) aligns reads using additional options specified by one or more name-value pair arguments.

Examples

Align Short Reads

Download the *E. coli* genome from NCBI.

```
getgenbank('NC_008253', 'tofile', 'NC_008253.fna', 'SequenceOnly', true)
```

Built a Bowtie index with the base name ECOLI.

```
bowtiebuild('NC_008253.fna', 'ECOLI')
```

Find the path to the example FASTQ file *ecoli100.fq*, which has *E. Coli* short reads.

```
fastqfile = which('ecoli100.fq')
```

Align the short reads in *ecoli100.fq* to the built index with base name ECOLI.

```
bowtie('ECOLI', fastqfile, 'ecoli100.bam')
```

Access the mapped reads using BioMap.

```
bm = BioMap('ecoli100.bam')
```

BioMap with properties:

```
SequenceDictionary: {'gi|110640213|ref|NC_008253.1|'}
Reference: [73x1 File indexed property]
```
Input Arguments

indexBaseName — Name of indexed reference file
character vector | string

Name of indexed reference file for short read alignment, specified as a character vector or string containing as the path and base name of the Bowtie index file.

reads — Short reads to align
character vector | string | string vector | cell array of character vectors

Short reads to align to the indexed reference, specified as a character vector, string, string vector, or cell array of character vectors indicating one or more FASTQ formatted files with the input reads.

outputFileName — Name for output file
character vector | string

Name for output file containing the results of the short read alignment, specified as a character vector or string. By default, the output file is BAM-formatted, and bowtie automatically adds the .bam extension if it is missing from the file name.

To specify a SAM-formatted output file, use the name-value pair argument BamFileOutput,false. In this case, bowtie automatically adds the .sam extension if it is missing from the file name.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: 'BamFileOutput',false,'Paired',true specifies the output file is SAM-formatted, and bowtie performs pair-read alignment.

BamFileOutput — Indicator for output file format
true (default) | false

Indicator for the output file format, specified as the comma-separated pair consisting of 'BamFileOutput' and either true or false.

- If true (the default), then the output file is BAM-formatted, with a .bam extension.
- If false, then the output file is SAM-formatted, with a .sam extension.

bowtie automatically adds the corresponding file extension if it is missing from the input argument outputFileName.
Example: 'BamFileOutput',false
Data Types: logical

**Paired** — Indicator for paired-read alignment performance
false (default) | true

Indicator for paired-read alignment performance, specified as the comma-separated pair consisting of 'Paired' and either true or false (the default). If false, then bowtie performs paired-read alignment using the odd elements in reads as the upstream mates and the even elements in reads as the downstream mates.

Example: 'Paired',true
Data Types: logical

**BowtieOptions** — Additional bowtie options
character vector | string

Additional bowtie options, specified as a character vector or string for any valid bowtie options. Type bowtie('--help') for available options.

Example: 'BowtieOptions','-k 5 -m 4'

**Tips**

- More information on the Bowtie algorithm (Version 0.12.7) can be found at http://bowtie-bio.sourceforge.net/index.shtml.
- Some prebuilt index files for model organisms can be downloaded directly from the Bowtie repository.

**See Also**

BioMap | baminfo | bowtie2 | bowtiebuild | fastainfo | fastqinfo | saminfo | samread

**Introduced in R2012b**
bowtie2

Map sequence reads to reference sequence

Syntax

bowtie2(indexBaseName, reads1, reads2, outputFileName)
bowtie2(___, alignOptions)
flag = bowtie2(___)

Description

bowtie2(indexBaseName, reads1, reads2, outputFileName) maps the sequencing reads from reads1 and reads2 against the reference sequence and writes the results to the output file outputFileName. The input indexBaseName represents the base name (prefix) of the reference index files.

bowtie2 requires the Bioinformatics Toolbox Interface for Bowtie Aligner support package. If this support package is not installed, then the function provides a download link.

Note bowtie2 is supported on Mac and UNIX platforms only.

bowtie2(___, alignOptions) uses the additional options specified by alignOptions. Specify these options after all other input arguments.

flag = bowtie2(___) returns an exit flag of the function using any of the input arguments in the previous syntaxes.

Examples

Align Reads to Reference Sequence Using Bowtie 2

Build a set of index files for the Drosophila genome. An error message appears if you do not have the Bioinformatics Toolbox Interface for Bowtie Aligner support package installed when you run the function. Click the provided link to download the package from the Add-on menu.

For this example, the reference sequence Dmel_chr4.fa is already provided with the toolbox.

status = bowtie2build('Dmel_chr4.fa', 'Dmel_chr4_index');

If the index build is successful, the function returns 0 and creates the index files (*.bt2) in the current folder. The files have the prefix 'Dmel_chr4_index'.

Sometimes the index files exist, and you want to know the reference sequence used to build the index. In this case, use the bowtie2inspect function to get more information about the reference.

bowtie2inspect('Dmel_chr4', 'Dmel_chr4_retrieved.fa');
By default, the output file Dmel_chr4_retrieved.fa contains the sequence of the reference. You can also get a summary information about the reference name and lengths instead of the actual sequence. For details on the available options, see Bowtie2InspectOptions.

Once the index is ready, map the read sequences to the reference using the bowtie2 function. The paired-end read files (SRR6008575_10k_1.fq and SRR6008575_10k_2.fq) are already provided with the toolbox.

```matlab
bowtie2('Dmel_chr4','SRR6008575_10k_1.fq','SRR6008575_10k_2.fq','SRR6008575_10k_chr4.sam');
```

The output is a SAM-formatted file that contains the mapping results.

You can specify different alignment options by passing in a Bowtie 2 syntax string or using a Bowtie2AlignOptions object.

Suppose you want to trim some residues from the 3' end before aligning. First, create a Bowtie2AlignOptions object.

```matlab
alignOpt = Bowtie2AlignOptions;
```

Trim four residues from the 3' end before aligning.

```matlab
alignOpt.Trim3 = 4;
```

Map reads to the reference using the specified alignment option.

```matlab
flag = bowtie2('Dmel_chr4','SRR6008575_10k_1.fq','SRR6008575_10k_2.fq','SRR6008575_10k_chr4trimmed.sam',alignOpt);
```

### Input Arguments

- **indexBaseName** — Base name of reference index files
  
  Character vector | string
  
  Base name (prefix) of the reference index files, specified as a character vector or string. The file extension for index files is either *.bt2 or *.bt21.
  
  Example: 'Dmel_chr4'
  
  Data Types: char | string

- **reads1** — Names of files with first mate reads
  
  Character vector | string
  
  Names of files with the first mate reads, specified as a character vector or string.
  
  Example: 'SRR6008575_10k_1.fq'
  
  Data Types: char | string

- **reads2** — Names of files with second mate reads
  
  Character vector | string
  
  Names of files with the second mate reads, specified as a character vector or string.
  
  Example: 'SRR6008575_10k_2.fq'
  
  Data Types: char | string
**outputFileName — Output file name**
character vector | string

Output file name, specified as a character vector or string. This file contains the mapping results.
Example: 'SRR6008575_10k_chr4.sam'
Data Types: char | string

**alignOptions — Alignment options**
character vector | string | Bowtie2AlignOptions object

Alignment options, specified as a character vector, string, or Bowtie2AlignOptions object. The character vector or string must be in the Bowtie 2 option syntax (prefixed by one or two dashes) [1].
For a Bowtie2AlignOptions object, only the modified properties are used to run the function.
Example: '--trim5 10 -s 5'

**Output Arguments**

**flag — Exit status**
integer

Exit status of the function, returned as an integer. flag is 0 if the function runs without errors or warning. Otherwise, it is nonzero.

**References**


**See Also**
Bowtie2AlignOptions | Bowtie2BuildOptions | Bowtie2InspectOptions | bowtie | bowtie2build | bowtie2inspect | cufflinks | featurecount

**Topics**
“Count Features from NGS Reads”

**External Websites**
Bowtie 2 manual

**Introduced in R2018a**
**Bowtie2AlignOptions**

Options to map reads to reference sequence

**Description**

A `Bowtie2AlignOptions` object contains options to run the `bowtie2` function, which aligns reads to a reference sequence.

**Creation**

**Syntax**

```
alignOptions = Bowtie2AlignOptions
alignOptions = Bowtie2AlignOptions(Name,Value)
alignOptions = Bowtie2AlignOptions(S)
```

**Description**

`alignOptions = Bowtie2AlignOptions` creates a `Bowtie2AlignOptions` object with default property values.

`Bowtie2AlignOptions` requires the Bioinformatics Toolbox Interface for Bowtie Aligner support package. If this support package is not installed, then the function provides a download link.

**Note** `Bowtie2AlignOptions` is supported on Mac and UNIX platforms only.

```
alignOptions = Bowtie2AlignOptions(Name,Value) sets properties on page 1-275 using one or more name-value pair arguments. Enclose each property name in quotes. For example,

`alignOptions = Bowtie2AlignOptions('Trim5',10)` specifies to trim 10 residues from the 5' end.

`alignOptions = Bowtie2AlignOptions(S)` specifies optional parameters in a character vector `S`.
```

**Input Arguments**

`S — Alignment parameters`  
character vector  

Alignment parameters, specified as a character vector. `S` must be in the Bowtie 2 option syntax (prefixed by one or two dashes) [1].

**Properties**

`AllowDovetail — Flag to allow dovetail configurations`  
false (default) | true
Flag to allow dovetail configurations, specified as true or false. This property specifies whether the alignment of one mate can extend past the beginning of the alignment of the other mate and be considered concordant.

This property applies to paired-end reads only.

Example: 'AllowDovetail',true

Data Types: logical

**AmbiguousPenalty — Penalty for positions with ambiguous characters**

Penalty for positions with ambiguous characters on the read sequence, reference sequence, or both, specified as a nonnegative integer.

Example: 'AmbiguousPenalty',2

Data Types: double

**Encoding — Encoding format of base quality**

Encoding format of the base quality in the input files, specified as one of the following: 'Phred33', 'Phred64', or 'Solexa'.

Example: 'Encoding','Phred64'

Data Types: char | string

**ExcludeContain — Flag to allow one mate alignment to contain other mate**

Flag to allow one mate alignment to contain the alignment of the other mate and to be considered concordant, specified as true or false.

This property applies to paired-end reads only.

Example: 'ExcludeContain',true

Data Types: logical

**ExcludeDiscordant — Flag to include discordant alignments**

Flag to include discordant alignments, specified as true or false. A discordant alignment is an alignment where both mates align uniquely, but not in a way that satisfies the paired-end constraints.

Example: 'ExcludeDiscordant',true

Data Types: logical

**ExcludeMixed — Flag to exclude mixed alignments**

Flag to exclude mixed alignments, specified as true or false. A mixed alignment consists of mate reads that are not concordant or discordant, but align individually.

This property applies to paired-end reads only.
Example: 'ExcludeMixed',true
Data Types: logical

**ExcludeOverlap** — Flag to allow mate alignment overlap
false (default) | true

Flag to allow the alignment of one mate to overlap with the alignment of the other mate and to be considered concordant, specified as true or false.
Example: 'ExcludeOverlap',true
Data Types: logical

**ExcludeUnaligned** — Flag to exclude reads that failed to align
false (default) | true

Flag to exclude reads that failed to align, specified as true or false.
Example: 'ExcludeUnaligned',true
Data Types: logical

**ExtraBowtie2Command** — Additional options not included in object properties
'' (default) | character vector

Additional options not included in the object properties, specified as a character vector. The character vector must be in the Bowtie 2 option syntax (prefixed by one or two dashes). The default value is an empty character vector ''.
Example: 'ExtraBowtie2Command','--version'
Data Types: char | string

**IgnoreQuality** — Flag to ignore read position quality
false (default) | true

Flag to ignore the actual read position quality when a mismatch occurs, specified as true or false. Setting this property to true allows the quality value at that mismatched position to be the highest possible, regardless of the actual value.
Example: 'IgnoreQuality',true
Data Types: logical

**MatchBonus** — Reward added to alignment score
2 (default) | nonnegative integer

Reward added to the alignment score when a position in the read matches a position in the reference, specified as a nonnegative integer.
Example: 'MatchBonus',5
Data Types: double

**MaxAmbiguousFunction** — Function governing maximum number of ambiguous characters
'L,0,0.15' (default) | character vector | string

Function governing the maximum number of ambiguous characters allowed in a read, specified as a character vector or string.
The function has the format ‘f,B,A’, where f is a function type, B is a constant term, and A is a coefficient. Available function types are:

- ‘C’ – Constant
- ‘L’ – Linear
- ‘S’ – Square root
- ‘G’ – Natural log

The resulting function is \( H(x) = B + A \times f(x) \), where \( x \) is the read length.

The default function is ‘L,0,0.15’, that is, \( H(x) = 0 + 0.15 \times x \).

Example: ‘MaxAmbiguousFunction’,‘L,-0.4,-0.6’

Data Types: char | string

**MemoryMappedIndex** — Flag to use memory mapping when loading index
false (default) | true

Flag to use memory mapping (instead of file I/O) when loading the index, specified as true or false. Memory mapping allows many concurrent processes to share the memory image of the index, resulting in a more efficient parallelization of the task.

Example: ‘MemoryMappedIndex’,true

Data Types: logical

**MinScoreFunction** — Function governing minimum score threshold of alignment
character vector | string

Function governing the minimum score threshold of an alignment, specified as a character vector or string.

The function has the format ‘f,B,A’, where f is a function type, B is a constant term, and A is a coefficient. Available function types are:

- ‘C’ – Constant
- ‘L’ – Linear
- ‘S’ – Square root
- ‘G’ – Natural log

The resulting function is \( H(x) = B + A \times f(x) \), where \( x \) is the read length.

For the ‘EndToEnd’ alignment mode, the default function is ‘L,-0.6,-0.6’. For the ‘Local’ mode, the default function is ‘G,20,8’.

Example: ‘MinScoreFunction’,‘L,-0.4,-0.6’

Data Types: char | string

**MismatchPenalty** — Maximum and minimum values to compute mismatch penalty
[6 2] (default) | two-element vector

Maximum and minimum values to compute the mismatch penalty during alignment, specified as a two-element vector. The first element is the maximum value and the second element is the minimum value.
A number less than or equal to the maximum value, and greater than or equal to the minimum value is subtracted from the alignment score for each position where a read character aligns to a reference character, the characters do not match, and neither is an N character.

Example: 'MismatchPenalty',[5 3]

Data Types: double

**Mode — Alignment mode**

'EndToEnd' (default) | 'Local'

Alignment mode, specified as 'EndToEnd' or 'Local'.

In the 'Local' mode, only part of the read must align to the reference, and some residues can be omitted (soft-clipped) to achieve the best alignment score. In the 'EndToEnd' mode, the entire read must align without any soft-clipping.

Example: 'Mode','Local'

Data Types: char | string

**Nondeterministic — Flag to reinitialize pseudo-random generator**

false (default) | true

Flag to reinitialize the pseudo-random generator for each read using the current time, specified as true or false. If true, the alignments reported for two identical reads can be different. The default value is false, that is, the pseudo-random generator is reinitialized using a seed derived from read information and the seed number.

Example: 'Nondeterministic',true

Data Types: logical

**NoGapPositions — Number of positions where gaps are not allowed**

4 (default) | nonnegative integer

Number of positions at the beginning or end of each read where gaps are not allowed, specified as a nonnegative integer.

Example: 'NoGapPositions',5

Data Types: double

**NumAlignments — Maximum number of valid alignments to report**

'Best' (default) | 'All' | positive integer

Maximum number of valid alignments to report before terminating the search, specified as a positive integer, 'Best', or 'All'. If you specify a positive integer N, the function searches for up to N distinct, valid alignments for each read. 'Best' reports the best alignment for each read. 'All' reports all the valid alignments for each read sorted by alignment scores.

The alignment score for a paired-end alignment equals the sum of the alignment scores of individual mates.

Example: 'NumAlignments','All'

Data Types: double | char | string

**NumReseedings — Maximum number of reseeding attempts**

2 (default) | nonnegative integer
Maximum number of reseeding attempts with repetitive seeds, specified as a nonnegative integer. During reseeding, the function chooses a new set of reads at different offsets to find more alignments.

Example: 'NumReseedings', 5

Data Types: double

NumSeedExtensions — Maximum number of consecutive seed extension attempts

15 (default) | nonnegative integer

Maximum number of consecutive seed extension attempts before getting a new seed, specified as a nonnegative integer. A seed extension fails if it does not yield an alignment with the best (or second-best) score.

Example: 'NumSeedExtensions', 10

Data Types: double

NumSeedMismatches — Number of allowed mismatches in seed alignment

0 (default) | 1

Number of allowed mismatches in a seed alignment during the multiseed alignment, specified as 0 or 1.

Example: 'NumSeedMismatches', 1

Data Types: double

NumThreads — Number of parallel threads to perform alignment

1 (default) | positive integer

Number of parallel threads to perform the alignment, specified as a positive integer. Threads run on separate processors or cores. Increasing the number of threads provides a significant increase in speed (close to linear) but also increases the memory footprint.

Example: 'NumThreads', 4

Data Types: double

Offrate — Offrate to use when reading index

NaN (default) | positive integer

Offrate to use when reading the index to reduce the memory footprint, specified as a positive integer. The offrate must be greater than the offrate used to build the index.

Example: 'Offrate', 20

Data Types: double

PadPositions — Position in reference sequence where alignment begins

15 (default) | nonnegative integer

Position in the reference sequence where the alignment for each sequence begins, specified as a nonnegative integer.

Example: 'PadPositions', 10

Data Types: double

ReadGapCosts — Gap costs for opening and extending gap

[5 3] (default) | two-element vector of nonnegative integers
Gap costs for opening and extending a gap on the read, specified as a two-element vector of nonnegative integers. The first element is the cost of opening a gap, and the second element is the cost of extending a gap. Given the cost vector \([G_0 \ GE]\), a read gap of length \(N\) is assigned a penalty of \(G_0 + N \times GE\).

Example: ‘ReadGapCosts’,[4 2]
Data Types: double

**ReadGroupID — Read group ID to add on @RG header line**

' ' (default) | character vector | string

Read group ID to add on the @RG header line in the output SAM report, specified as a character vector or string. If you specify any read group ID, the function prints the @RG header line with the tag ID: followed by the specified group ID.

Example: ‘ReadGroupID’,‘ID1’
Data Types: char | string

**ReadGroup — Read group information to add as field on @RG header line**

‘ ‘ (default) | character vector | string

Read group information to add as a field on the @RG header line in the output SAM report, specified as a character vector or string. This property applies only if you specify ‘ReadGroupID’.

Example: ‘ReadGroup’, ‘Control’
Data Types: char | string

**RefGapCosts — Gap costs for opening and extending gap**

[5 3] (default) | two-element vector of nonnegative integers

Gap costs for opening and extending a gap on the reference, specified as a two-element vector of nonnegative integers. The first element is the cost of opening a gap, and the second element is the cost of extending a gap. Given the cost vector \([G_0 \ GE]\), a reference gap of length \(N\) is assigned a penalty of \(G_0 + N \times GE\).

Example: ‘RefGapCosts’,[4 2]
Data Types: double

**Reorder — Flag to reorder SAM records**

false (default) | true

Flag to reorder SAM records to maintain the same order as in the input files, specified as true or false. This property applies only when the number of parallel threads is greater than one. When you use one thread, the order of the records in the output is the same as the order of the input.

Example: ‘Reorder’,true
Data Types: logical

**Seed — Number to set seed in pseudo-random number generator**

0 (default) | nonnegative integer

Number to set the seed in the pseudo-random number generator, specified as a nonnegative integer.

Example: ‘Seed’,3
Data Types: double

**SeedIntervalFunction** — Function governing distance between seed substrings

character vector | string

Function governing the distance between seed substrings during the multiseed alignment, specified as a character vector or string.

The function has the format ’f,B,A’, where f is a function type, B is a constant term, and A is a coefficient. Available function types are:

- ’C’– Constant
- ’L’– Linear
- ’S’– Square root
- ’G’– Natural log

The resulting function is \( H(x) = B + A \cdot f(x) \), where \( x \) is the read length.

For the 'EndToEnd' alignment mode, the default function is ’S,1,1.15’. For the 'Local' mode, the default function is ’S,1,0.75’.

Example: ‘SeedIntervalFunction’,’S,2,2.15’

Data Types: char | string

**SeedLength** — Seed substring length to align during multiseed alignment

20 (default) | positive integer

Seed substring length to align during the multiseed alignment, specified as a positive integer.

Example: ‘SeedLength’,25

Data Types: double

**Skip** — Number of reads to ignore

0 (default) | nonnegative integer

Number of reads to ignore from the beginning of the input files, specified as a nonnegative integer.

Example: ‘Skip’,5

Data Types: double

**Trim3** — Number of residues to trim from 3’ end

0 (default) | nonnegative integer

Number of residues to trim from the 3’ end of each read before aligning, specified as a nonnegative integer.

Example: ‘Trim3’,5

Data Types: double

**Trim5** — Number of residues to trim from 5’ end

0 (default) | nonnegative integer

Number of residues to trim from the 5’ end of each read before aligning, specified as a nonnegative integer.
Example: ‘Trim5’, 5
Data Types: double

**UpTo — Number of reads to consider from beginning of input files**

Inf (default) | positive integer

Number of reads to consider from the beginning of input files, specified as a positive integer. The default value is Inf, that is, all reads are considered.

Example: ‘UpTo’, 1000
Data Types: double

**Object Functions**

- getBowtie2Command: Translate object properties to Bowtie 2 options
- getBowtie2Table: Retrieve table with object properties and equivalent Bowtie 2 options
- preset: Set combination of alignment options
- run: Map sequence reads to reference sequence using Bowtie 2

**Examples**

**Align Reads to Reference Sequence Using Bowtie 2**

Build a set of index files for the Drosophila genome. An error message appears if you do not have the Bioinformatics Toolbox Interface for Bowtie Aligner support package installed when you run the function. Click the provided link to download the package from the Add-on menu.

For this example, the reference sequence `Dmel_chr4.fa` is already provided with the toolbox.

```matlab
status = bowtie2build('Dmel_chr4.fa', 'Dmel_chr4_index');
```

If the index build is successful, the function returns 0 and creates the index files (*.bt2) in the current folder. The files have the prefix `Dmel_chr4_index`.

Sometimes the index files exist, and you want to know the reference sequence used to build the index. In this case, use the `bowtie2inspect` function to get more information about the reference.

```matlab
bowtie2inspect('Dmel_chr4', 'Dmel_chr4_retrieved.fa');
```

By default, the output file `Dmel_chr4_retrieved.fa` contains the sequence of the reference. You can also get a summary information about the reference name and lengths instead of the actual sequence. For details on the available options, see `Bowtie2InspectOptions`.

Once the index is ready, map the read sequences to the reference using the `bowtie2` function. The paired-end read files (`SRR6008575_10k_1.fq` and `SRR6008575_10k_2.fq`) are already provided with the toolbox.

```matlab
bowtie2('Dmel_chr4','SRR6008575_10k_1.fq','SRR6008575_10k_2.fq','SRR6008575_10k_chr4.sam');
```

The output is a SAM-formatted file that contains the mapping results.

You can specify different alignment options by passing in a Bowtie 2 syntax string or using a `Bowtie2AlignOptions` object.
Suppose you want to trim some residues from the 3' end before aligning. First, create a Bowtie2AlignOptions object.

```matlab
alignOpt = Bowtie2AlignOptions;
```

Trim four residues from the 3' end before aligning.

```matlab
alignOpt.Trim3 = 4;
```

Map reads to the reference using the specified alignment option.

```matlab
flag = bowtie2('Dmel_chr4','SRR6008575_10k_1.fq','SRR6008575_10k_2.fq','SRR6008575_10k_chr4_trimmed.sam',alignOpt);
```

**References**


**See Also**

Bowtie2AlignOptions | Bowtie2BuildOptions | Bowtie2InspectOptions | bowtie2 | bowtie2build | bowtie2inspect

**External Websites**

Bowtie 2 manual

**Introduced in R2018a**
bowtie2build

Create Bowtie 2 index files from reference sequences

Syntax

bowtie2build(referenceFileNames,indexBaseName)
bowtie2build( ___,buildOptions)
flag = bowtie2build( ___ )

Description

bowtie2build(referenceFileNames,indexBaseName) builds Bowtie2 index files from the reference sequence information saved in the FASTA files specified by referenceFileNames.

bowtie2build requires the Bioinformatics Toolbox Interface for Bowtie Aligner support package. If this support package is not installed, then the function provides a download link.

Note  bowtie2build is supported on Mac and UNIX platforms only.

bowtie2build( ___,buildOptions) uses the additional options specified by buildOptions. Specify these options after all other input arguments.

flag = bowtie2build( ___ ) returns an exit flag of the function using any of the input arguments in the previous syntaxes.

Examples

Build Bowtie2 Index Files for Reference Sequence

Build a set of index files for the Drosophila genome. An error message appears if you do not have the Bioinformatics Toolbox Interface for Bowtie Aligner support package installed when you run the function. Click the provided link to download the package from the Add-on menu.

For this example, the reference sequence Dmel_chr4.fa is already provided with the toolbox.

status = bowtie2build('Dmel_chr4.fa', 'Dmel_chr4_index');

If the index build is successful, the function returns 0 and creates the index files (*.bt2) in the current folder. The files have the prefix 'Dmel_chr4_index'.

You can specify different options by using a Bowtie2BuildOptions object or by passing in a Bowtie 2 syntax string. For instance, you can specify whether to force the creation of a large index even if the reference is less than four billion nucleotides long as follows.

buildOpt = Bowtie2BuildOptions;
Set the ForceLargeIndex option to true.
buildOpt.ForceLargeIndex = true;

Build the index files using the specified option.

bowtie2build('Dmel_chr4.fa', 'Dmel_chr4_index_large',buildOpt);

Alternatively, you can pass in a Bowtie 2 syntax string.

flag = bowtie2build('Dmel_chr4.fa', 'Dmel_chr4_index_large2','--large-index');

**Input Arguments**

`referenceFileNames` — Names of files with reference sequence information

string | character vector | string array | cell array of character vectors

Names of files with reference sequence information, specified as a string, character vector, string array, or cell array of character vectors.

Example: 'Dmel_chr4.fa'

Data Types: char | string | cell

`indexBaseName` — Base name of reference index files

character vector | string

Base name (prefix) of the reference index files, specified as a character vector or string. The file extension for index files is either *.bt2 or *.bt21.

Example: 'Dmel_chr4'

Data Types: char | string

`buildOptions` — Options to build index files

character vector | string | Bowtie2BuildOptions object

Options to build index files, specified as a character vector, string, or Bowtie2BuildOptions object. The character vector or string must be in the Bowtie 2 option syntax (prefixed by one or two dashes) [1].

For a Bowtie2BuildOptions object, only the modified properties are used to run the function.

Example: '--trim5 10 -s 5'

**Output Arguments**

`flag` — Exit status

integer

Exit status of the function, returned as an integer. flag is 0 if the function runs without errors or warning. Otherwise, it is nonzero.

**References**

See Also
Bowtie2AlignOptions | Bowtie2BuildOptions | Bowtie2InspectOptions | bowtie2 | bowtie2inspect | cufflinks | featurecount

Topics
“Count Features from NGS Reads”

External Websites
Bowtie 2 manual

Introduced in R2018a
Bowtie2BuildOptions

Contain options to create Bowtie 2 index files from reference sequences

Description

A Bowtie2BuildOptions object contains options to run the bowtie2build function that builds Bowtie 2 index files from reference sequences.

Creation

Syntax

buildOptions = Bowtie2BuildOptions
buildOptions = Bowtie2BuildOptions(Name,Value)
buildOptions = Bowtie2BuildOptions(S)

Description

buildOptions = Bowtie2BuildOptions creates a Bowtie2BuildOptions object with default property values.

Bowtie2BuildOptions requires the Bioinformatics Toolbox Interface for Bowtie Aligner support package. If this support package is not installed, then the function provides a download link.

Note Bowtie2BuildOptions is supported on Mac and UNIX platforms only.

buildOptions = Bowtie2BuildOptions(Name,Value) sets properties on page 1-289 using one or more name-value pairs. Enclose each property name in quotes. For example, buildOptions = Bowtie2BuildOptions('ForceLargeIndex',true) specifies to force the creation of a large index even if the reference is less than 4 billion nucleotides long.

buildOptions = Bowtie2BuildOptions(S) specifies optional parameters in a character vector or string S.

Input Arguments

S — Parameters to build index files
character vector

Parameters to build the index files, specified as a character vector. S must be in the Bowtie 2 option syntax (prefixed by one or two dashes) [1].

Data Types: char | string
Properties

**BuildOnlyReference** — Boolean indicator to build only bitpacked reference
false (default) | true

Boolean indicator to build only the 3. bt2 and 4. bt2 files that correspond to the bitpacked version of reference sequences, specified as true or false.

Example: 'BuildOnlyReference',true
Data Types: logical

**BuildNoReference** — Boolean indicator to omit building bitpacked reference
false (default) | true

Boolean indicator to omit building the 3. bt2 and 4. bt2 files that correspond to the bitpacked version of reference sequences, specified as true or false.

Example: 'BuildNoReference',true
Data Types: logical

**ExtraBowtie2Command** — Additional options not included in object properties
' ' (default) | character vector

Additional options not included in the object properties, specified as a character vector. The character vector must be in the Bowtie 2 option syntax (prefixed by one or two dashes). The default value is an empty character vector ' '.

Example: 'ExtraBowtie2Command','--version'
Data Types: char | string

**ForceLargeIndex** — Boolean indicator to force building large index
false (default) | true

Boolean indicator to force building a large index even if the reference is less than four billion nucleotides long, specified as true or false.

Example: 'ForceLargeIndex',true
Data Types: logical

**NumThreads** — Number of parallel threads to build index files
1 (default) | positive integer

Number of parallel threads to build the index files, specified as a positive integer. Threads are run on separate processors or cores. Increasing the number of threads provides significant speed-up (close to linear) but also increases the memory footprint.

Example: 'NumThreads',4
Data Types: double

**Offrate** — Number of Burrows-Wheeler rows to mark when building index
5 (default) | positive integer

Number of Burrows-Wheeler rows to mark when building the index files, specified as a positive integer. To map the alignment back to positions on the reference sequences, the function uses this
number to mark some of the rows in the Burrows-Wheeler algorithm with their corresponding location on the genome. The function marks every $2^n$, where $n$ is the offrate.

Increasing the number of marked rows makes the reference position lookups faster, but requires more memory.

Example: 'Offrate',6
Data Types: double

Seed — Number to set seed in pseudo-random number generator

0 (default) | nonnegative integer

Number to set the seed in the pseudo-random number generator, specified as a nonnegative integer.

Example: 'Seed',3
Data Types: double

Object Functions

getBowtie2Command
Translate object properties to Bowtie 2 options

getBowtie2Table
Retrieve table with object properties and equivalent Bowtie 2 options

run
Build Bowtie 2 index files

Examples

Build Bowtie 2 Index Files for Reference Sequence

Build a set of index files for the Drosophila genome. An error message appears if you do not have the Bioinformatics Toolbox Interface for Bowtie Aligner support package installed when you run the function. Click the provided link to download the package from the Add-on menu.

For this example, the reference sequence Dmel_chr4.fa is already provided with the toolbox.

status = bowtie2build('Dmel_chr4.fa', 'Dmel_chr4_index');

If the index build is successful, the function returns 0 and creates the index files (*.bt2) in the current folder. The files have the prefix 'Dmel_chr4_index'.

You can specify different options by using a Bowtie2BuildOptions object or by passing in a Bowtie 2 syntax string. For instance, you can specify whether to force the creation of a large index even if the reference is less than four billion nucleotides long as follows.

buildOpt = Bowtie2BuildOptions;

Set the ForceLargeIndex option to true.

buildOpt.ForceLargeIndex = true;

Build the index files using the specified option.

bowtie2build('Dmel_chr4.fa', 'Dmel_chr4_index_large',buildOpt);

Alternatively, you can pass in a Bowtie 2 syntax string.
flag = bowtie2build('Dmel_chr4.fa', 'Dmel_chr4_index_large2', '--large-index');

References


See Also
Bowtie2AlignOptions | Bowtie2InspectOptions | bowtie2 | bowtie2build | bowtie2inspect

External Websites
Bowtie 2 manual

Introduced in R2018a
**bowtie2inspect**

Inspect Bowtie 2 index files

**Syntax**

```matlab
bowtie2inspect(indexBaseName, outputFileName)
bowtie2inspect(___ , inspectOptions)
flag = bowtie2inspect(___)
```

**Description**

`bowtie2inspect(indexBaseName, outputFileName)` inspects Bowtie2 index files with the prefix `indexBaseName`, checks the original reference sequences used to build the index, and saves the reference sequences in an output file `outputFileName`.

`bowtie2inspect` requires the Bioinformatics Toolbox Interface for Bowtie Aligner support package. If this support package is not installed, then the function provides a download link.

**Note** `bowtie2inspect` is supported on Mac and UNIX platforms only.

`bowtie2inspect(___ , inspectOptions)` uses the additional options specified by `inspectOptions`. Specify the options after all other input arguments.

`flag = bowtie2inspect(___)` returns an exit flag of the function using any of the input arguments in the previous syntaxes.

**Examples**

**Inspect Bowtie2 Index and Retrieve Reference Sequence Information**

Get information about the reference sequence used to build the corresponding index files by using `bowtie2inspect`. An error message appears if you do not have the Bioinformatics Toolbox Interface for Bowtie Aligner support package installed when you run the function. Click the provided link to download the package from the Add-on menu.

```matlab
bowtie2inspect('Dmel_chr4', 'Dmel_chr4_retrieved.fa');
```

By default, the output file `Dmel_chr4_retrieved.fa` contains the actual sequence of the reference used to build the index.

You can also get summary information about the reference name and lengths instead of the actual sequence.

Create an options object.

```matlab
inspectOpt = Bowtie2InspectOptions;
```

Set the `Summary` property to `true`. 

```matlab
```
inspectOpt.Summary = true;

Run the function again using the specified option.

flag = bowtie2inspect('Dmel_chr4', 'Dmel_chr4_summary.fa',inspectOpt);

If the index inspection is successful, the function returns 0, and the output file now contains summary information of the reference sequence.

Alternatively, you can pass in a Bowtie 2 syntax string instead of using the option object.

flag = bowtie2inspect('Dmel_chr4', 'Dmel_chr4_summary2.fa','-s');

### Input Arguments

**indexBaseName** — Base name of reference index files  
character vector | string  

Base name (prefix) of the reference index files, specified as a character vector or string. The file extension for index files is either *.bt2 or *.bt21.

Example: 'Dmel_chr4'

Data Types: char | string

**outputFileName** — Name of output file  
string | character vector  

Name of an output file, specified as a string or character vector. By default, the output file contains the reference sequences that are used to build the index files.

Example: 'refSeq.fa'

Data Types: char | string

**inspectOptions** — Options to inspect index files  
character vector | string | Bowtie2InspectOptions object  

Options to inspect index files, specified as a character vector, string, or Bowtie2InspectOptions object. The character vector or string must be in the Bowtie 2 option syntax (prefixed by one or two dashes) [1].

For a Bowtie2InspectOptions object, only the modified properties are used to run the function.

Example: '--trim5 10 -s 5'

### Output Arguments

**flag** — Exit status  
integer  

Exit status of the function, returned as an integer. flag is 0 if the function runs without errors or warning. Otherwise, it is nonzero.
References


See Also

Bowtie2AlignOptions | Bowtie2BuildOptions | Bowtie2InspectOptions | bowtie2 | bowtie2build

External Websites

Bowtie 2 manual

Introduced in R2018a
**Bowtie2InspectOptions**

Contain options to inspect Bowtie 2 index files

**Description**

A `Bowtie2InspectOptions` object contains options to run the `bowtie2inspect` function that inspects Bowtie 2 index files.

**Creation**

**Syntax**

```matlab
inspectOptions = Bowtie2InspectOptions
inspectOptions = Bowtie2InspectOptions(Name,Value)
inspectOptions = Bowtie2InspectOptions(S)
```

**Description**

`inspectOptions = Bowtie2InspectOptions` creates a `Bowtie2InspectOptions` object with default property values.

`Bowtie2InspectOptions` requires the Bioinformatics Toolbox Interface for Bowtie Aligner support package. If this support package is not installed, then the function provides a download link.

**Note** `Bowtie2InspectOptions` is supported on Mac and UNIX platforms only.

```matlab
inspectOptions = Bowtie2InspectOptions(Name,Value) sets properties on page 1-296 using one or more name-value pairs. Enclose each property name in quotes. For instance, `inspectOptions = Bowtie2InspectOptions('Summary',true)` specifies to return a summary of the index content instead of reference sequences.

inspectOptions = Bowtie2InspectOptions(S) specifies optional parameters in a character vector or string `S`.

**Input Arguments**

**S — Parameters to inspect index files**

*character vector*

Parameters to inspect the index files, specified as a character vector. `S` must be in the Bowtie 2 option syntax (prefixed by one or two dashes) [1].

Data Types: `char` | `string`
Properties

ExtraBowtie2Command — Additional options not included in object properties
'default' | character vector

Additional options not included in the object properties, specified as a character vector. The character vector must be in the Bowtie 2 option syntax (prefixed by one or two dashes). The default value is an empty character vector ''. Example: 'ExtraBowtie2Command','--version'

Data Types: char | string

NamesOnly — Boolean indicator to print only reference sequence names in output
false (default) | true

Boolean indicator to print only the reference sequence names in the output file, specified as true or false. Example: 'NamesOnly',true

Data Types: logical

Summary — Boolean indicator to print summary instead of actual reference sequences
false (default) | true

Boolean indicator to print a summary information about reference sequence names and lengths instead of actual sequences in the output file, specified as true or false. Example: 'Summary',true

Data Types: logical

Object Functions

getBowtie2Command Translate object properties to Bowtie 2 options
getBowtie2Table Retrieve table with object properties and equivalent Bowtie 2 options
run Inspect Bowtie 2 index files

Examples

Inspect Bowtie 2 Index and Retrieve Reference Sequence Information

Get information about the reference sequence used to build the corresponding index files by using bowtie2inspect. An error message appears if you do not have the Bioinformatics Toolbox Interface for Bowtie Aligner support package installed when you run the function. Click the provided link to download the package from the Add-on menu.

bowtie2inspect('Dmel_chr4', 'Dmel_chr4_retrieved.fa');

By default, the output file Dmel_chr4_retrieved.fa contains the actual sequence of the reference used to build the index.

You can also get summary information about the reference name and lengths instead of the actual sequence.
Create an options object.
inspectOpt = Bowtie2InspectOptions;

Set the Summary property to true.
inspectOpt.Summary = true;

Run the function again using the specified option.
flag = bowtie2inspect('Dmel_chr4', 'Dmel_chr4_summary.fa',inspectOpt);

If the index inspection is successful, the function returns 0, and the output file now contains summary information of the reference sequence.

Alternatively, you can pass in a Bowtie 2 syntax string instead of using the option object.
flag = bowtie2inspect('Dmel_chr4', 'Dmel_chr4_summary2.fa', '-s');

**References**


**See Also**
Bowtie2AlignOptions | Bowtie2BuildOptions | bowtie2 | bowtie2build | bowtie2inspect

**External Websites**
Bowtie 2 manual

**Introduced in R2018a**
**bowtiebuild**

Generate index using Burrows-Wheeler transform

**Syntax**

bowtiebuild(input,indexBaseName)

bowtiebuild(input,indexBaseName,'BowtieBuildOptions',options)

**Description**

bowtiebuild(input,indexBaseName) builds an index using the reference sequence(s) in input, and saves it to the index file indexBaseName.

*Note* bowtiebuild runs on Mac and UNIX platforms only.

bowtiebuild(input,indexBaseName,'BowtieBuildOptions',options) specifies additional options.

**Examples**

**Build a Bowtie Index**

Download the *E. coli* genome from NCBI.

getgenbank('NC_008253','tofile','NC_008253.fna','SequenceOnly',true)

Built a Bowtie index with the base name ECOLI.

bowtiebuild('NC_008253.fna','ECOLI')

**Input Arguments**

**input — FASTA-formatted files**
character vector | string | string vector | cell array of character vectors

FASTA-formatted files with the reference sequences to be indexed, specified as a character vector, string, string vector, or cell array of character vectors. Use a cell array of character vectors or string vector to specify multiple files.

**indexBaseName — Name for indexed reference file**
character vector | string

Name for indexed reference file, specified as a character vector or string containing the path and base name for the resulting Bowtie index file.

**options — Additional bowtiebuild options**
character vector | string
Additional `bowtiebuild` options, specified as a character vector or string for any valid
`bowtiebuild` options. Type `bowtiebuild('--help')` for available options.
Example: `'-t 5 -C'

**Tips**

- More information on the Bowtie algorithm (Version 0.12.7) can be found at http://bowtie-
bio.sourceforge.net/index.shtml.
- Some prebuilt index files for model organisms can be downloaded directly from the Bowtie
repository.

**See Also**

BioMap | bamInfo | bowtie | fastainfo | fastqinfo | samInfo | samread

**Introduced in R2012b**
celintensityread

Read probe intensities from Affymetrix CEL files

Syntax

ProbeStructure = celintensityread(CELFiles, CDFFile)

ProbeStructure = celintensityread(..., 'CELPath', CELPathValue, ...)
ProbeStructure = celintensityread(..., 'CDFPath', CDFPathValue, ...)
ProbeStructure = celintensityread(..., 'PMOnly', PMOnlyValue, ...)
ProbeStructure = celintensityread(..., 'Verbose', VerboseValue, ...)

Input Arguments

<table>
<thead>
<tr>
<th>CELFiles</th>
<th>Any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Character vector or string specifying a single CEL file name.</td>
<td></td>
</tr>
<tr>
<td>• '*' , which reads all CEL files in the current folder.</td>
<td></td>
</tr>
<tr>
<td>• ' ' , which opens the Select CEL Files dialog box from which you select the CEL files. From this dialog box, you can press and hold Ctrl or Shift while clicking to select multiple CEL files.</td>
<td></td>
</tr>
<tr>
<td>• Cell array of character vectors or string vector containing CEL file names.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CDFFile</th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Character vector or string specifying a CDF file name.</td>
<td></td>
</tr>
<tr>
<td>• ' ' , which opens the Select CDF File dialog box from which you select the CDF file.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CELPathValue</th>
<th>Character vector or string specifying the path and folder where the files specified in CELFiles are stored.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDFPathValue</td>
<td>Character vector or string specifying the path and folder where the file specified in CDFFile is stored.</td>
</tr>
</tbody>
</table>

| PMOnlyValue      | Property to include or exclude the mismatch (MM) probe intensity values in the returned structure. Enter true to return only perfect match (PM) probe intensities. Enter false to return both PM and MM probe intensities. Default is true. |

| VerboseValue     | Controls the display of a progress report showing the name of each CEL file as it is read. When VerboseValue is false, no progress report is displayed. Default is true. |

Output Arguments

| ProbeStructure   | MATLAB structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs. |
**Description**

$\text{ProbeStructure} = \text{celintensityread}(\text{CELFiles}, \text{CDFFile})$ reads the specified Affymetrix CEL files and the associated CDF library file (created from Affymetrix GeneChip arrays for expression or genotyping assays), and then creates $\text{ProbeStructure}$, a structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs. $\text{CELFiles}$ is a character vector, string, string vector, or cell array of character vectors containing CEL file names. $\text{CDFFile}$ is a character vector or string specifying a CDF file name.

If you set $\text{CELFiles}$ to ‘*’, then it reads all CEL files in the current folder. If you set $\text{CELFiles}$ to ‘ ’, then it opens the Select CEL Files dialog box from which you select the CEL files. From this dialog box, you can press and hold Ctrl or Shift while clicking to select multiple CEL files.

If you set $\text{CDFFile}$ to ‘ ’, then it opens the Select CDF File dialog box from which you select the CDF file.

$\text{ProbeStructure} = \text{celintensityread}(\ldots, \text{'PropertyName'}, \text{PropertyValue}, \ldots)$ calls celintensityread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each $\text{PropertyName}$ must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

$\text{ProbeStructure} = \text{celintensityread}(\ldots, \text{'CELPath'}, \text{CELPathValue}, \ldots)$ specifies a path and folder where the files specified by $\text{CELFiles}$ are stored.

$\text{ProbeStructure} = \text{celintensityread}(\ldots, \text{'CDFPath'}, \text{CDFPathValue}, \ldots)$ specifies a path and folder where the file specified by $\text{CDFFile}$ is stored.

$\text{ProbeStructure} = \text{celintensityread}(\ldots, \text{'PMOnly'}, \text{PMOnlyValue}, \ldots)$ includes or excludes the mismatch (MM) probe intensity values. When $\text{PMOnlyValue}$ is true, celintensityread returns only perfect match (PM) probe intensities. When $\text{PMOnlyValue}$ is false, celintensityread returns both PM and MM probe intensities. Default is true.

**Tip** Reading a large number of CEL files and/or a large CEL file can require extended amounts of memory from the operating system.

- If you receive errors related to memory, try the following:
  - Increase the virtual memory (swap space) for your operating system as described in “Resolve ‘Out of Memory’ Errors” (MATLAB).
  - If you receive errors related to Java® heap space, increase your Java heap space:
    - If you have MATLAB version 7.10 (R2010a) or later, see “Java Heap Memory Preferences” (MATLAB).
    - If you have MATLAB version 7.9 (R2009b) or earlier, see https://www.mathworks.com/support/solutions/en/data/1-1812C/.

$\text{ProbeStructure}$ contains the following fields.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDFName</td>
<td>File name of the Affymetrix CDF library file.</td>
</tr>
<tr>
<td>CELNames</td>
<td>Cell array of names of the Affymetrix CEL files.</td>
</tr>
<tr>
<td>NumChips</td>
<td>Number of CEL files read into the structure.</td>
</tr>
<tr>
<td>NumProbeSets</td>
<td>Number of probe sets in each CEL file.</td>
</tr>
<tr>
<td>NumProbes</td>
<td>Number of probes in each CEL file.</td>
</tr>
<tr>
<td>ProbeSetIDs</td>
<td>Cell array of the probe set IDs from the Affymetrix CDF library file.</td>
</tr>
<tr>
<td>ProbeIndices</td>
<td>Column vector containing probe indexing information. Probes within a probe set are numbered 0 through N - 1, where N is the number of probes in the probe set.</td>
</tr>
</tbody>
</table>
| GroupNumbers  | Column vector containing group numbers for probes within the probe set. For gene expression data, the group number for all probes is 1. For SNP (genotyping) data, the group numbers for probes are:  
- 1 — Allele A - (sense)  
- 2 — Allele B - (sense)  
- 3 — Allele A + (antisense)  
- 4 — Allele B + (antisense) |
| PMIntensities | Matrix containing perfect match (PM) probe intensity values. Each row corresponds to a probe, and each column corresponds to a CEL file. The rows are ordered the same way as in ProbeIndices, and the columns are ordered the same way as in the CELFiles input argument. |
| MMIntensities (optional) | Matrix containing mismatch (MM) probe intensity values. Each row corresponds to a probe, and each column corresponds to a CEL file. The rows are ordered the same way as in ProbeIndices, and the columns are ordered the same way as in the CELFiles input argument. |

`ProbeStructure = celintensityread(..., 'Verbose',VerboseValue,...)` controls the display of a progress report showing the name of each CEL file as it is read. When `VerboseValue` is false, no progress report is displayed. Default is true.

**Examples**

The following example assumes that you have the HG_U95Av2.CDF library file stored at D:\Affymetrix\LibFiles\HGGenome, and that your current folder points to a location containing CEL files associated with this CDF library file. In this example, the `celintensityread` function reads all the CEL files in the current folder and a CDF file in a specified folder. The next command line uses the `rmabackadj` function to perform background adjustment on the PM probe intensities in the PMIntensities field of PMProbeStructure.

```matlab
PMProbeStructure = celintensityread('*', 'HG_U95Av2.CDF',... 'CDFPath', 'D:\Affymetrix\LibFiles\HGGenome');
BackAdjustedMatrix = rmabackadj(PMProbeStructure.PMIntensities);
```
See Also
affygcrgma | affyinvarsetnorm | affyprobeseqread | affyreadd | affyrma |
affysnpintensitysplit | agferead | gcrma | gcrmabackadj | gprread | ilmnbsread |
probelibraryinfo | probesetlink | probesetlookup | probesetplot | probesetvalues |
rmabackadj | rmasummary | sptread

Topics
“Preprocessing Affymetrix® Microarray Data at the Probe Level”
“Analyzing Affymetrix® SNP Arrays for DNA Copy Number Variants”

Introduced in R2006a
cghcbs

Perform circular binary segmentation (CBS) on array-based comparative genomic hybridization (aCGH) data

Syntax

SegmentStruct = cghcbs(CGHData)

SegmentStruct = cghcbs(CGHData, ...'Alpha', AlphaValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Permutations', PermutationsValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Method', MethodValue, ...)
SegmentStruct = cghcbs(CGHData, ...'StoppingRule', StoppingRuleValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Smooth', SmoothValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Prune', PruneValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Errsum', ErrsumValue, ...)
SegmentStruct = cghcbs(CGHData, ...'WindowSize', WindowSizeValue, ...)
SegmentStruct = cghcbs(CGHData, ...'SampleIndex', SampleIndexValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Chromosome', ChromosomeValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Showplot', ShowplotValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Verbose', VerboseValue, ...)

Input Arguments

<table>
<thead>
<tr>
<th>CGHData</th>
<th>Array-based comparative genomic hybridization (aCGH) data in either of the following forms:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Structure with the following fields:</td>
</tr>
<tr>
<td></td>
<td>• Sample — Cell array of character vectors or string vector containing the sample names (optional).</td>
</tr>
<tr>
<td></td>
<td>• Chromosome — Vector containing the chromosome numbers on which the clones are located.</td>
</tr>
<tr>
<td></td>
<td>• GenomicPosition — Vector containing the genomic positions (in any unit) to which the clones are mapped.</td>
</tr>
<tr>
<td></td>
<td>• Log2Ratio — Matrix containing log₂ ratio of test to reference signal intensity for each clone. Each row corresponds to a clone, and each column corresponds to a sample.</td>
</tr>
<tr>
<td></td>
<td>• Matrix in which each row corresponds to a clone. The first column contains the chromosome number, the second column contains the genomic position, and the remaining columns each contain the log₂ ratio of test to reference signal intensity for a sample.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AlphaValue</th>
<th>Scalar that specifies the significance level for the statistical tests to accept change points. Default is 0.01.</th>
</tr>
</thead>
</table>

<p>| PermutationsValue | Scalar that specifies the number of permutations used for p-value estimation. Default is 10,000. |</p>
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MethodValue</strong></td>
<td>Character vector or string that specifies the method to estimate the p-values. Choices are 'Perm' or 'Hybrid' (default). 'Perm' does a full permutation, while 'Hybrid' uses a faster, tail probability-based permutation. When using the 'Hybrid' method, the 'Perm' method is applied automatically when segment data length becomes less than 200.</td>
</tr>
<tr>
<td><strong>StoppingRuleValue</strong></td>
<td>Controls the use of a heuristic stopping rule, based on the method described by Venkatraman and Olshen (2007) on page 1-312, to declare a change without performing the full number of permutations for the p-value estimation, whenever it becomes very likely that a change has been detected. Choices are true or false (default).</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>Set this property to true to increase processing speed. Set this property to false to maximize accuracy.</td>
</tr>
<tr>
<td><strong>SmoothValue</strong></td>
<td>Controls the smoothing of outliers before segmenting using the procedure explained by Olshen et al. (2004) on page 1-312. Choices are true (default) or false.</td>
</tr>
<tr>
<td><strong>PruneValue</strong></td>
<td>Controls the elimination of change points identified due to local trends in the data that are not indicative of real copy number change, using the procedure explained by Olshen et al. (2004) on page 1-312. Choices are true or false (default).</td>
</tr>
<tr>
<td><strong>ErrsumValue</strong></td>
<td>Scalar that specifies the allowed proportional increase in the error sum of squares when eliminating change points using the 'Prune' property. Commonly used values are 0.05 and 0.1. Default is 0.05.</td>
</tr>
<tr>
<td><strong>WindowSizeValue</strong></td>
<td>Scalar that specifies the size of the window (in data points) used to divide the data when using the 'Perm' method on large data sets. Default is 200.</td>
</tr>
<tr>
<td><strong>SampleIndexValue</strong></td>
<td>A single sample index or a vector of sample indices that specify the sample(s) to analyze. Default is all sample indices.</td>
</tr>
<tr>
<td><strong>ChromosomeValue</strong></td>
<td>A single chromosome number or a vector of chromosome numbers that specify the data to analyze. Default is all chromosome numbers.</td>
</tr>
<tr>
<td><strong>ShowplotValue</strong></td>
<td>Controls the display of plots of the segment means over the original data. Choices are either:</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>true</strong></td>
<td>All chromosomes in all samples are plotted. If there are multiple samples in \textit{CGHData}, then each sample is plotted in a separate Figure window.</td>
</tr>
<tr>
<td><strong>false</strong></td>
<td>No plot.</td>
</tr>
<tr>
<td><strong>W</strong></td>
<td>The layout displays all chromosomes in the whole genome in one plot in the Figure window.</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td>The layout displays each chromosome in a subplot in the Figure window.</td>
</tr>
<tr>
<td><strong>I</strong></td>
<td>An integer specifying only one of the chromosomes in \textit{CGHData} to be plotted.</td>
</tr>
<tr>
<td>Default is:</td>
<td></td>
</tr>
<tr>
<td><strong>false</strong></td>
<td>When return values are specified.</td>
</tr>
<tr>
<td><strong>true</strong></td>
<td>When return values are not specified.</td>
</tr>
</tbody>
</table>

| **VerboseValue** | Controls the display of a progress report of the analysis. Choices are \textit{true} (default) or \textit{false}. |

## Output Arguments

<table>
<thead>
<tr>
<th><strong>SegmentStruct</strong></th>
<th>Structure containing segmentation information in the following fields:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sample</strong></td>
<td>Sample name from \textit{CGHData} input argument. If the input argument does not include sample names, then sample names are assigned as \textit{Sample1}, \textit{Sample2}, and so forth.</td>
</tr>
<tr>
<td><strong>SegmentData</strong></td>
<td>Structure array containing segment data for the sample in the following fields:</td>
</tr>
<tr>
<td><strong>Chromosome</strong></td>
<td>Chromosome number on which the segment is located.</td>
</tr>
<tr>
<td><strong>Start</strong></td>
<td>Genomic position at the start of the segment (in the same units as used for the \textit{CGHData} input).</td>
</tr>
<tr>
<td><strong>End</strong></td>
<td>Genomic position at the end of the segment (in the same units as used for the \textit{CGHData} input).</td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td>Mean value of the log$_2$ ratio of the test to reference signal intensity for the segment.</td>
</tr>
</tbody>
</table>

## Description

\textit{SegmentStruct} = \textit{cghcbs(CGHData)} performs circular binary segmentation (CBS) on array-based comparative genomic hybridization (aCGH) data to determine the copy number alteration segments (neighboring regions of DNA that exhibit a statistical difference in copy number) and change points.
Note The CBS algorithm recursively splits chromosomes into segments based on a maximum $t$ statistic estimated by permutation. This computation can be time consuming. If $n = \text{number of data points}$, then computation time $\sim O(n^2)$.

$SegmentStruct = \text{cghcbs}(\text{CGHData}, \ldots, 'PropertyName', \text{PropertyValue}, \ldots)$ calls cghcbs with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each $PropertyName$ must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

$SegmentStruct = \text{cghcbs}(\text{CGHData}, \ldots, 'Alpha', \text{AlphaValue}, \ldots)$ specifies the significance level for the statistical tests to accept change points. Default is $0.01$.

$SegmentStruct = \text{cghcbs}(\text{CGHData}, \ldots, 'Permutations', \text{PermutationsValue}, \ldots)$ specifies the number of permutations used for p-value estimation. Default is $10,000$.

$SegmentStruct = \text{cghcbs}(\text{CGHData}, \ldots, 'Method', \text{MethodValue}, \ldots)$ specifies the method to estimate the p-values. Choices are 'Perm' or 'Hybrid' (default). 'Perm' does a full permutation, while 'Hybrid' uses a faster, tail probability-based permutation. When using the 'Hybrid' method, the 'Perm' method is applied automatically when segment data length becomes less than 200.

$SegmentStruct = \text{cghcbs}(\text{CGHData}, \ldots, 'StoppingRule', \text{StoppingRuleValue}, \ldots)$ controls the use of a heuristic stopping rule, based on the method described by Venkatraman and Olshen (2007) on page 1-312, to declare a change without performing the full number of permutations for the p-value estimation, whenever it becomes very likely that a change has been detected. Choices are true or false (default).

$SegmentStruct = \text{cghcbs}(\text{CGHData}, \ldots, 'Smooth', \text{SmoothValue}, \ldots)$ controls the smoothing of outliers before segmenting, using the procedure explained by Olshen et al. (2004) on page 1-312. Choices are true (default) or false.

$SegmentStruct = \text{cghcbs}(\text{CGHData}, \ldots, 'Prune', \text{PruneValue}, \ldots)$ controls the elimination of change points identified due to local trends in the data that are not indicative of real copy number change, using the procedure explained by Olshen et al. (2004) on page 1-312. Choices are true or false (default).

$SegmentStruct = \text{cghcbs}(\text{CGHData}, \ldots, 'Errsum', \text{ErrsumValue}, \ldots)$ specifies the allowed proportional increase in the error sum of squares when eliminating change points using the 'Prune' property. Commonly used values are $0.05$ and $0.1$. Default is $0.05$.

$SegmentStruct = \text{cghcbs}(\text{CGHData}, \ldots, 'WindowSize', \text{WindowSizeValue}, \ldots)$ specifies the size of the window (in data points) used to divide the data when using the 'Perm' method on large data sets. Default is $200$.

$SegmentStruct = \text{cghcbs}(\text{CGHData}, \ldots, 'SampleIndex', \text{SampleIndexValue}, \ldots)$ analyzes only the sample(s) specified by SampleIndexValue, which can be a single sample index or a vector of sample indices. Default is all sample indices.

$SegmentStruct = \text{cghcbs}(\text{CGHData}, \ldots, 'Chromosome', \text{ChromosomeValue}, \ldots)$ analyzes only the data on the chromosomes specified by ChromosomeValue, which can be a single chromosome number or a vector of chromosome numbers. Default is all chromosome numbers.
SegmentStruct = cghcbs(CGHData, ...'Showplot', ShowplotValue, ...) controls the display of plots of the segment means over the original data. Choices are true, false, W, S, or I, an integer specifying one of the chromosomes in CGHData. When ShowplotValue is true, all chromosomes in all samples are plotted. If there are multiple samples in CGHData, then each sample is plotted in a separate Figure window. When ShowplotValue is W, the layout displays all chromosomes in one plot in the Figure window. When ShowplotValue is S, the layout displays each chromosome in a subplot in the Figure window. When ShowplotValue is I, only the specified chromosome is plotted. Default is either:

- false — When return values are specified.
- true and W — When return values are not specified.

SegmentStruct = cghcbs(CGHData, ...'Verbose', VerboseValue, ...) controls the display of a progress report of the analysis. Choices are true (default) or false.

Examples

Perform circular binary segmentation on comparative genomic hybridization data

Analyze data from the Coriell cell line study

Load the array-based CGH (aCGH) data from the Coriell cell line study (Snijders, A. et al., 2001).

load coriell_baccgh

Analyze all chromosomes of sample 3 (GM05296) of the aCGH data and return segmentation data in a structure, S. Plot the segment means over the original data for all chromosomes of this sample.

S = cghcbs(coriell_data,'sampleindex',3,'showplot',true);

Analyzing: GM05296. Current chromosome 1
Analyzing: GM05296. Current chromosome 2
Analyzing: GM05296. Current chromosome 3
Analyzing: GM05296. Current chromosome 4
Analyzing: GM05296. Current chromosome 5
Analyzing: GM05296. Current chromosome 6
Analyzing: GM05296. Current chromosome 7
Analyzing: GM05296. Current chromosome 8
Analyzing: GM05296. Current chromosome 9
Analyzing: GM05296. Current chromosome 10
Analyzing: GM05296. Current chromosome 11
Analyzing: GM05296. Current chromosome 12
Analyzing: GM05296. Current chromosome 14
Analyzing: GM05296. Current chromosome 15
Analyzing: GM05296. Current chromosome 16
Analyzing: GM05296. Current chromosome 17
Analyzing: GM05296. Current chromosome 18
Analyzing: GM05296. Current chromosome 19
Analyzing: GM05296. Current chromosome 20
Analyzing: GM05296. Current chromosome 21
Analyzing: GM05296. Current chromosome 22
Analyzing: GM05296. Current chromosome 23
Analyzing: GM05296. Current chromosome 26
Chromosome 10 shows a gain, while chromosome 11 shows a loss.

**Display copy number alteration regions aligned to a chromosome ideogram**

Create a structure containing segment gain and loss information for chromosomes 10 and 11 from sample 3, making sure the segment data is in bp units. (You can determine copy number variance (CNV) information by exploring S, the structure of segments returned by the cghcbs function. For the 'CNVType' field, use 1 to indicate a loss and 2 to indicate a gain.

```matlab
cnvStruct = struct('Chromosome', [10 11], ...
    'CNVType', [2 1],...
    'Start', [S.SegmentData(10).Start(2),...
              S.SegmentData(11).Start(2)]*1000, ...
    'End', [S.SegmentData(10).End(2),...
             S.SegmentData(11).End(2)]*1000)
```

```matlab
cnvStruct =
struct with fields:
    Chromosome: [10 11]
    CNVType: [2 1]
    Start: [66905000 35416000]
```
Pass the structure to the `chromosomeplot` function using the 'CNV' option to display the copy number gains (green) and losses (red) aligned to the human chromosome ideogram. Specify kb units for the display of segment information in the data tip.

\[
\text{chromosomeplot('hs\_cytoBand.txt', 'CNV', cnvStruct, 'unit', 2)}
\]

Analyze data from a pancreatic cancer study

Load the aCGH data from a pancreatic cancer study (Aguirre, A. et al., 2004).

\[
\text{load pancrea\_oligocgh}
\]

Analyze only chromosome 9 in sample 32 of the CGH data and return the segmentation data in a structure, PS. Plot the segment means over the original data for chromosome 9 in this sample.

\[
\text{PS = cghcbs(pancrea\_data,'sampleindex',32,'chromosome',9,...}
\text{'showplot',9);}
\]

Chromosome 9 contains two segments that indicate losses. For more detailed information on interpreting the data, see Aguirre, A. et al. (2004).

Use the `chromosomeplot` function with the 'addtoplot' option to add the ideogram of chromosome 9 for Homo sapiens to the plot of the segmentation data.

`chromosomeplot('hs_cytoBand.txt', 9, 'addtoplot', gca)`
References


See Also

chromosomeplot | cytobandread
Introduced in R2007b
cghfreqplot

Display frequency of DNA copy number alterations across multiple samples

Syntax

FreqStruct = cghfreqplot(CGHD ata)

FreqStruct = cghfreqplot(CGHD ata, ...'Threshold', ThresholdValue, ...)
FreqStruct = cghfreqplot(CGHD ata, ...'Group', GroupValue, ...)
FreqStruct = cghfreqplot(CGHD ata, ...'Subgrp', SubgrpValue, ...)
FreqStruct = cghfreqplot(CGHD ata, ...'Subplot', SubplotValue, ...)
FreqStruct = cghfreqplot(CGHD ata, ...'Cutoff', CutoffValue, ...)
FreqStruct = cghfreqplot(CGHD ata, ...'Chromosome', ChromosomeValue, ...)
FreqStruct = cghfreqplot(CGHD ata, ...'IncludeX', IncludeXValue, ...)
FreqStruct = cghfreqplot(CGHD ata, ...'IncludeY', IncludeYValue, ...)
FreqStruct = cghfreqplot(CGHD ata, ...'Chrominfo', ChrominfoValue, ...)
FreqStruct = cghfreqplot(CGHD ata, ...'ShowCentr', ShowCentrValue, ...)
FreqStruct = cghfreqplot(CGHD ata, ...'Color', ColorValue, ...)
FreqStruct = cghfreqplot(CGHD ata, ...'YLim', YLimValue, ...)
FreqStruct = cghfreqplot(CGHD ata, ...'Titles', TitlesValue, ...)

Input Arguments

<table>
<thead>
<tr>
<th>CGHD ata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array-based comparative genomic hybridization (aCGH) data in either of the following forms:</td>
</tr>
<tr>
<td>• Structure with the following fields:</td>
</tr>
<tr>
<td>• Sample — Cell array of character vectors or string vector containing the sample names (optional).</td>
</tr>
<tr>
<td>• Chromosome — Vector containing the chromosome numbers on which the clones are located.</td>
</tr>
<tr>
<td>• GenomicPosition — Vector containing the genomic positions (in bp, kb, or mb units) to which the clones are mapped.</td>
</tr>
<tr>
<td>• Log2Ratio — Matrix containing log₂ ratio of test to reference signal intensity for each clone. Each row corresponds to a clone, and each column corresponds to a sample.</td>
</tr>
<tr>
<td>• Matrix in which each row corresponds to a clone. The first column contains the chromosome number, the second column contains the genomic position, and the remaining columns each contain the log₂ ratio of test to reference signal intensity for a sample.</td>
</tr>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
</tbody>
</table>
| **ThresholdValue** | Positive scalar or vector that specifies the gain/loss threshold. A clone is considered to be a gain if its log₂ ratio is above `ThresholdValue`, and a loss if its log₂ ratio is below negative `ThresholdValue`. The `ThresholdValue` is applied as follows:  
• If a positive scalar, it is the gain and loss threshold for all the samples.  
• If a two-element vector, the first element is the gain threshold for all samples, and the second element is the loss threshold for all samples.  
• If a vector of the same length as the number of samples, each element in the vector is considered as a unique gain and loss threshold for each sample.  
Default is `0.25`. |
| **GroupValue**   | Specifies the sample groups to calculate the frequency from. Choices are:  
• A vector of sample column indices (for data with only one group). The samples specified in the vector are considered a group.  
• A cell array of vectors of sample column indices (for data divided into multiple groups). Each element in the cell array is considered a group.  
Default is a single group of all the samples in `CGHData`. |
| **SubgrpValue**  | Controls the analysis of samples by subgroups. Choices are `true` (default) or `false`. |
| **SubplotValue** | Controls the display of all plots in one Figure window when more than one subgroup is analyzed. Choices are `true` (default) or `false` (displays plots in separate windows). |
| **CutoffValue**  | Scalar or two-element numeric vector that specifies a cutoff, which controls the plotting of only the clones with frequency gains or losses greater than or equal to `CutoffValue`. If a two-element vector, the first element is the cutoff for gains, and the second element is for losses. Default is `0`. |
| **ChromosomeValue** | Single chromosome number or a vector of chromosome numbers that specify the chromosomes for which to display frequency plots. Default is all chromosomes in `CGHData`. |
| **IncludeXValue** | Controls the inclusion of the X chromosome in the analysis. Choices are `true` (default) or `false`. |
| **IncludeYValue** | Controls the inclusion of the Y chromosome in the analysis. Choices are `true` or `false` (default). |
| **ChrominfoValue** | Cytogenetic banding information specified by either of the following:  
| | - Structure returned by the `cytobandread` function  
| | - Character vector or string specifying the file name of an NCBI ideogram text file or a UCSC Genome Browser cytoband text file  
| | Default is *Homo sapiens* cytogenetic banding information from the UCSC Genome Browser, NCBI Build 36.1 ([https://genome.ucsc.edu](https://genome.ucsc.edu)). |

| **ShowCentrValue** | Controls the display of the centromere positions as vertical dashed lines in the frequency plot. Choices are `true` (default) or `false`. |

**Tip** The centromere positions are obtained from `ChrominfoValue`. |

| **ColorValue** | Color scheme for the vertical lines in the plot, indicating the frequency of the gains and losses, specified by either of the following:  
| | - Name of or handle to a function that returns a colormap  
| | - M-by-3 matrix containing RGB values. If M equals 1, then that single color is used for all gains and losses. If M equals 2 or more, then the first row is used for gains, the second row is used for losses, and remaining rows are ignored. For example, `[0 1 0; 1 0 0]` specifies green for gain and red for loss.  
| | The default color scheme is a range of colors from pure green (gain = 1) through yellow (0) to pure red (loss = -1). |

| **YLimValue** | Two-element vector specifying the minimum and maximum values on the vertical axis. Default is `[1, -1]`. |

| **TitlesValue** | Character vector, string, string vector, or a cell array of character vectors that specifies titles for the group(s), which are added to the tops of the plot(s). |
Output Arguments

<table>
<thead>
<tr>
<th><code>FreqStruct</code></th>
<th>Structure containing frequency data in the following fields:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• <strong>Group</strong> — Structure array, with each structure representing a group of samples. Each structure contains the following fields:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Sample</strong> — Cell array containing names of samples within the group.</td>
</tr>
<tr>
<td></td>
<td>• <strong>GainFrequency</strong> — Column vector containing the average gain for each clone for a group of samples.</td>
</tr>
<tr>
<td></td>
<td>• <strong>LossFrequency</strong> — Column vector containing the average loss for each clone for a group of samples.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Chromosome</strong> — Column vector containing the chromosome numbers on which the clones are located.</td>
</tr>
<tr>
<td></td>
<td>• <strong>GenomicPosition</strong> — Column vector containing the genomic positions of the clones.</td>
</tr>
</tbody>
</table>

**Tip** You can use this output structure as input to the `cghfreqplot` function.

Description

`FreqStruct = cghfreqplot(CGHDdata)` displays the frequency of copy number gain or loss across multiple samples for each clone on an array against their genomic position along the chromosomes.

`FreqStruct = cghfreqplot(CGHDdata, ...'PropertyName', PropertyValue, ...)` calls `cghfreqplot` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`FreqStruct = cghfreqplot(CGHDdata, ...'Threshold', ThresholdValue, ...)` specifies the gain/loss threshold. A clone is considered to be a gain if its log\(_2\) ratio is above `ThresholdValue`, and a loss if its log\(_2\) ratio is below negative `ThresholdValue`.

The `ThresholdValue` is applied as follows:

- If a positive scalar, it is the gain and loss threshold for all the samples.
- If a two-element vector, the first element is the gain threshold for all samples, and the second element is the loss threshold for all samples.
- If a vector of the same length as the number of samples, each element in the vector is considered as a unique gain and loss threshold for each sample.

Default is 0.25.

`FreqStruct = cghfreqplot(CGHDdata, ...'Group', GroupValue, ...)` specifies the sample groups to calculate the frequency from. Choices are:

- A vector of sample column indices (for data with only one group). The samples specified in the vector are considered a group.
A cell array of vectors of sample column indices (for data divided into multiple groups). Each element in the cell array is considered a group.

Default is a single group of all the samples in CGHData.

FreqStruct = cghfreqplot(CGHData, ...'Subgrp', SubgrpValue, ...) controls the analysis of samples by subgroups. Choices are true (default) or false.

FreqStruct = cghfreqplot(CGHData, ...'Subplot', SubplotValue, ...) controls the display of all plots in one Figure window when more than one subgroup is analyzed. Choices are true (default) or false (displays plots in separate windows).

FreqStruct = cghfreqplot(CGHData, ...'Cutoff', CutoffValue, ...) specifies a cutoff value, which controls the plotting of only the clones with frequency gains or losses greater than or equal to CutoffValue. CutoffValue is a scalar or two-element numeric vector. If a two-element numeric vector, the first element is the cutoff for gains, and the second element is for losses. Default is 0.

FreqStruct = cghfreqplot(CGHData, ...'Chromosome', ChromosomeValue, ...) displays the frequency plots only of chromosome(s) specified by ChromosomeValue, which can be a single chromosome number or a vector of chromosome numbers. Default is all chromosomes in CGHData.

FreqStruct = cghfreqplot(CGHData, ...'IncludeX', IncludeXValue, ...) controls the inclusion of the X chromosome in the analysis. Choices are true (default) or false.

FreqStruct = cghfreqplot(CGHData, ...'IncludeY', IncludeYValue, ...) controls the inclusion of the Y chromosome in the analysis. Choices are true or false (default).

FreqStruct = cghfreqplot(CGHData, ...'Chrominfo', ChrominfoValue, ...) specifies the cytogenetic banding information for the chromosomes. ChrominfoValue can be either of the following

- Structure returned by the cytobandread function
- Character vector or string specifying the file name of an NCBI ideogram text file or a UCSC Genome Browser cytoband text file

Default is Homo sapiens cytogenetic banding information from the UCSC Genome Browser, NCBI Build 36.1 (https://genome.ucsc.edu).

Tip You can download files containing cytogenetic G-banding data from the NCBI or UCSC Genome Browser ftp site. For example, you can download the cytogenetic banding data for Homo sapiens from:

ftp://hgdownload.cse.ucsc.edu/goldenPath/hg18/database/cytoBandIdeo.txt.gz

FreqStruct = cghfreqplot(CGHData, ...'ShowCentr', ShowCentrValue, ...) controls the display of the centromere positions as vertical dashed lines in the frequency plot. Choices are true (default) or false.

Tip The centromere positions are obtained from ChrominfoValue.
FreqStruct = cghfreqplot(CGHData, ...'Color', ColorValue, ...) specifies a color scheme for the vertical lines in the plot, indicating the frequency of the gains and losses. Choices are:

- Name of or handle to a function that returns a colormap.
- M-by-3 matrix containing RGB values. If M equals 1, then that single color is used for all gains and losses. If M equals 2 or more, then the first row is used for gains, the second row is used for losses, and remaining rows are ignored. For example, [0 1 0;1 0 0] specifies green for gain and red for loss.

The default color scheme is a range of colors from pure green (gain = 1) through yellow (0) to pure red (loss = -1).

FreqStruct = cghfreqplot(CGHData, ...'YLim', YLimValue, ...) specifies the y vertical limits for the frequency plot. YLimValue is a two-element vector specifying the minimum and maximum values on the vertical axis. Default is [1, -1].

FreqStruct = cghfreqplot(CGHData, ...'Titles', TitlesValue, ...) specifies titles for the group(s), which are added to the tops of the plot(s). TitlesValue can be a character vector, string, string vector, or a cell array of character vectors.

Examples

Display the frequency of copy number alterations from multiple samples

Plot data from the Coriell cell line study

Load the array-based CGH (aCGH) data from the Coriell cell line study (Snijders, A. et al., 2001).

load coriell_baccgh

Display a frequency plot of the copy number alterations across all samples.

Struct = cghfreqplot(coriell_data);
View data tips for the data, chromosomes, and centromeres. First click the **Data Cursor** button on the toolbar, then click the black chromosome boundary line, or a dotted centromere line in the plot. To delete this data tip, right-click it, then select **Delete Current Datatip**.

Display a color bar indicating the degree of gain or loss by clicking the **Insert Colorbar** button on the toolbar.

**Plot data from a pancreatic cancer study**

Load the aCGH data from a pancreatic cancer study (Aguirre, A. et al., 2004).

```matlab
load pancrea_oligocgh
```

Display a frequency plot of the copy number alterations across all samples using a green and red color scheme.

```matlab
cghfreqplot(pancrea_data, 'Color', [0 1 0; 1 0 0])
```
Plotting groups of aCGH Data

Define two groups of data.

```matlab
grp1 = strncmp('PA.C', pancrea_data.Sample,4);
grp1_ind = find(grp1);
grp2 = strncmp('PA.T', pancrea_data.Sample,4);
grp2_ind = find(grp2);
```

Display a frequency plot of the copy number alterations across all samples in the two groups and limit the plotting to only the clones with frequency gains or losses greater than or equal to 0.25.

```matlab
SP = cghfreqplot(pancrea_data, 'Group', {grp1_ind, grp2_ind},...
    'Title', {'CL', 'PT'}, 'Cutoff', 0.25);
```
Display a frequency plot of the copy number alterations across all samples in the first group and limit the plot to chromosome 4 only.

```matlab
SP = cghfreqplot(pancrea_data, 'Group', grp1_ind, ...
                 'Title', 'CL Group on Chr 4', 'Chromosome', 4);
```
Use the `chromosomeplot` function with the 'addtoplot' option to add the ideogram of chromosome 4 for Homo sapiens to this frequency plot. Because the plot of the frequency data from the pancreatic cancer study is in kb units, use the 'Unit' option to convert the ideogram data to kb units.

`chromosomeplot('hs_cytoBand.txt', 4, 'addtoplot', gca, 'Unit', 2);`
References


See Also
cghcbs|chromosomeplot|cytobandread

Introduced in R2008a
**chromosomeplot**

Plot chromosome ideogram with G-banding pattern

**Syntax**

chromosomeplot(CytoData)
chromosomeplot(CytoData, ChromNum)

chromosomeplot(CytoData, ChromNum, ..., 'Orientation', OrientationValue, ...)
chromosomeplot(CytoData, ChromNum, ..., 'ShowBandLabel', ShowBandLabelValue, ...)
chromosomeplot(CytoData, ChromNum, ..., 'AddToPlot', AddToPlotValue, ...)
chromosomeplot(..., 'Unit', UnitValue, ...)
chromosomeplot(..., 'CNV', CNVValue, ...)

**Arguments**

| **CytoData** | Either of the following:
| - Character vector or string specifying a file containing cytogenetic G-banding data (in bp units), such as an NCBI ideogram text file or a UCSC Genome Browser cytoband text file.
| - Structure containing cytogenetic G-banding data (in bp units) in the following fields:
| - ChromLabels
| - BandStartBPs
| - BandEndBPs
| - BandLabels
| - GieStains
| **Tip** Use the cytobandread function to create the structure to use for **CytoData**.

| **ChromNum** | Scalar or character vector or string specifying a single chromosome to plot. Valid entries are integers, 'X', and 'Y'.

**Note** Setting **ChromNum** to 0 will plot ideograms for all chromosomes.

| **OrientationValue** | Character vector or string or number that specifies the orientation of the ideogram of a single chromosome specified by **ChromNum**. Choices are 'Vertical' or 1 (default) and 'Horizontal' or 2.

| **ShowBandLabelValue** | Controls the display of band labels (such as q25.3) when plotting a single chromosome ideogram, specified by **ChromNum**. Choices are true (default) or false.
AddToPlotValue | Variable name of a figure axis to which to add the single chromosome ideogram, specified by ChromNum.

**Note** If you use this property to add the ideogram to a plot of genomic data that is in units other than bp, use the 'Unit' property to convert the ideogram data to the appropriate units.

**Tip** Before printing a figure containing an added chromosome ideogram, change the background to white by issuing the following command:

```
set(gcf,'color','w')
```

UnitValue | Integer that specifies the units (base pairs, kilo base pairs, or mega base pairs) for the starting and ending genomic positions. This unit is used in the data tip displayed when you hover the cursor over chromosomes in the ideogram. This unit can also be used when using the 'AddToPlot' property to add the ideogram to a plot that is in units other than bp. Choices are 1 (bp), 2 (kb), or 3 (mb). Default is 1 (bp).

CNVValue | Controls the display of copy number variance (CNV) data, provided by CNVValue, aligned to the chromosome ideogram. Gains are shown in green to the right or above the ideogram, while losses are shown in red to the left or below the ideogram. CNVValue is a structure array containing the four fields described in the table below.

**Description**

chromosomeplot(CytoData) plots the ideogram of all chromosomes, using information from CytoData, a structure containing cytogenetic G-banding data (in bp units), or a character vector or string specifying a file containing cytogenetic G-banding data (in bp units), such as an NCBI ideogram text file or a UCSC Genome Browser cytoband text file. The G bands distinguish different areas of the chromosome. For example, for the Homo sapiens ideogram, possible G bands are:

- gneg — white
- gpos25 — light gray
- gpos50 — medium gray
- gpos75 — dark gray
- gpos100 — black
- acen — red (centromere)
- stalk — indented region (region with repeats)
- gvar — light blue

Darker bands are AT-rich, while lighter bands are GC-rich.

chromosomeplot(CytoData, ChromNum) plots the ideogram of a single chromosome specified by ChromNum.
chromosomeplot(..., 'PropertyName', PropertyValue, ...) calls chromosomeplot with optional properties that use property name/property value pairs. You can specify one or more properties in any order. EachPropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

chromosomeplot(CytoData, ChromNum, ..., 'Orientation', OrientationValue, ...) specifies the orientation of the ideogram of a single chromosome specified by ChromNum. Choices are 'Vertical' or 1 (default) and 'Horizontal' or 2.

**Note** When plotting the ideogram of all chromosomes, the orientation is always vertical.

chromosomeplot(CytoData, ChromNum, ..., 'ShowBandLabel', ShowBandLabelValue, ...) displays band labels (such as q25.3) when plotting a single chromosome ideogram, specified by ChromNum. Choices are true (default) or false.

chromosomeplot(CytoData, ChromNum, ..., 'AddToPlot', AddToPlotValue, ...) adds the single chromosome ideogram, specified by ChromNum, to a figure axis specified by AddToPlotValue.

**Note** If you use this property to add the ideogram to a plot of genomic data that is in units other than bp, use the 'Unit' property to convert the ideogram data to the appropriate units.

**Tip** Before printing a figure containing an added chromosome ideogram, change the background to white by issuing the following command:

genesis(gcf,'color','w')

chromosomeplot(..., 'Unit', UnitValue, ...) specifies the units (base pairs, kilo base pairs, or mega base pairs) for the starting and ending genomic positions. This unit is used in the data tip displayed when you hover the cursor over chromosomes in the ideogram. This unit can also be used when using the 'AddToPlot' property to add the ideogram to a plot that is in units other than bp. Choices are 1 (bp), 2 (kb), or 3 (mb). Default is 1 (bp).

chromosomeplot(..., 'CNV', CNVValue, ...) displays copy number variance (CNV) data, provided by CNVValue, aligned to the chromosome ideogram. Gains are shown in green to the right or above the ideogram, while losses are shown in red to the left or below the ideogram. CNVValue is a structure array containing the following fields. Each field must contain the same number of elements.
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| Chromosome| Either of the following:  
  - Numeric vector containing the chromosome number on which each CNV is located.  
  - Character array containing the chromosome number on which each CNV is located. |
|           | Note For the sex chromosome, X, use N, where $N$ = number of autosomes + 1. For the sex chromosome, Y, use M, where $M$ = number of autosomes + 2. For example, for *Homo sapiens* use 23 for X and 24 for Y, and for *Mus musculus* (lab mouse), use 20 for X and 21 for Y. |
| CNVType   | Numeric vector containing the type of each CNV, either 1 (loss) or 2 (gain). |
| Start     | Numeric vector containing the starting genomic position of each CNV. Units must be in base pairs. |
| End       | Numeric vector containing the ending genomic position of each CNV. Units must be in base pairs. |

### Examples

**Plot Chromosome Ideograms**

Read the cytogenetic banding information for *Homo sapiens* into a structure.

```matlab
hs_cytobands = cytobandread('hs_cytoBand.txt')
hs_cytobands = struct with fields:
  ChromLabels: {862x1 cell}
  BandStartBPs: [862x1 int32]
  BandEndBPs: [862x1 int32]
  BandLabels: {862x1 cell}
  GieStains: {862x1 cell}
```

Plot the entire chromosome ideogram.

```matlab
chromosomeplot(hs_cytobands);
title('Human Karyogram')
```
You can display the ideogram of a specific chromosome by right-clicking it in the plot, then selecting **Display in New Figure > Vertical** or **Horizontal**.

You can also programmatically display the ideogram of a specific chromosome, set the orientation, and the units used in the data tip to kilo base pairs.

```
chromosomeplot(hs_cytobands, 15, 'Orientation', 2, 'Unit', 2);
```
Hover over the chromosome to view a data tip. To get more information about a specific band, select the Data Cursor button on the toolbar and click the band in the plot. Use the context menu (right-click) to see more options such as deleting or creating a data tip.

**Display copy number alteration data aligned to chromosome ideogram**

Load the array-based CGH (aCGH) data from the Coriell cell line study (Snijders, A. et al., 2001).

```matlab
load coriell_baccgh
```

Use the `cghcbs` function to analyze chromosome 10 of sample 3 (GM05296) of the aCGH data and return copy number variance (CNV) data in a structure, S. Plot the segment means over the original data for only chromosome 10 of sample 3.

```matlab
S = cghcbs(coriell_data,'sampleindex',3,'chromosome',10,...
    'showplot',10);
```

Analyzing: GM05296. Current chromosome 10
Use the chromosomeplot function with the 'addtoplot' option to add the ideogram of chromosome 10 for Homo sapiens to the plot. Because the plot of the CNV data from the Coriell cell line study is in kb units, use the 'Unit' property to convert the ideogram data to kb units.

```matlab
set(gcf,'color','w'); % Set the background of the figure to white.
chromosomeplot('hs_cytoBand.txt', 10, 'addtoplot', gca,
    'Unit', 2);
```

Use the chromosomeplot function with the 'addtoplot' option to add the ideogram of chromosome 10 for Homo sapiens to the plot. Because the plot of the CNV data from the Coriell cell line study is in kb units, use the 'Unit' property to convert the ideogram data to kb units.
References


See Also
cghcbs | cytobandread

Introduced in R2007b
cigar2align

Convert unaligned sequences to aligned sequences using signatures in CIGAR format

Syntax

Alignment = cigar2align(Seqs,Cigars)
[GapSeq, Indices] = cigar2align(Seqs,Cigars)
... = cigar2align(Seqs,Cigars,Name,Value)

Description

Alignment = cigar2align(Seqs,Cigars) converts unaligned sequences in Seqs, a cell array of character vectors or string vector, into Alignment, a matrix of aligned sequences, using the information stored in Cigars, a cell array of CIGAR-formatted character vectors or string vector.

[GapSeq, Indices] = cigar2align(Seqs,Cigars) converts unaligned sequences in Seqs, a cell array of character vectors or string vector, into GapSeq, a cell array of character vectors of aligned sequences, and also returns Indices, a vector of numeric indices, using the information stored in Cigars, a cell array of CIGAR-formatted character vectors or string vector. When an alignment has many columns, this syntax uses less memory and is faster.

... = cigar2align(Seqs,Cigars,Name,Value) converts unaligned sequences in Seqs, a cell array of character vectors or string vector, into Alignment, a matrix of aligned sequences, using the information stored in Cigars, a cell array of CIGAR-formatted character vectors or string vector, with additional options specified by one or more Name,Value pair arguments.

Input Arguments

Seqs

Cell array of character vectors or string vector containing unaligned sequences. Seqs must contain the same number of elements as Cigars.

Default:

Cigars

Cell array of valid CIGAR-formatted character vectors or string vector. Cigars must contain the same number of elements as Seqs.

Default:

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.
Start

Vector of positive integers specifying the reference sequence position at which each aligned sequence starts. By default, each aligned sequence starts at position 1 of the reference sequence.

Default:

GapsInRef

Logical specifying whether to display positions in the aligned sequences that correspond to gaps in the reference sequence. Choices are true (1) or false (0). If your reference sequence has gaps and you set GapsInRef to false (0), and then later use Alignment as input to align2cigar, the returned CIGAR-formatted character vectors will not match the original ones.

Default: false (0)

SoftClipping

Logical specifying whether to include characters in the aligned read sequences corresponding to soft clipping ends. Choices are true (1) or false (0).

Default: false (0)

OffsetPad

Logical specifying whether to add padding blanks to the left of each aligned read sequence to represent the offset of the start position from the first position of the reference sequence. Choices are true (1) or false (0). When false, the matrix of aligned sequences starts at the start position of the leftmost aligned read sequence.

Default: false (0)

Output Arguments

Alignment

Matrix of aligned sequences, in which the number of rows equals the number of character vectors in Seqs.

GapSeq

Cell array of character vectors of aligned sequences, in which the number character vectors equals the number of character vectors in Seqs.

Indices

Vector of numeric indices indicating the starting column for each aligned sequence in Alignment. These indices are not necessarily the same as the start positions in the reference sequence for each aligned sequence. This is because either of the following:

- The reference sequence can be extended to account for insertions.
- An aligned sequence can have leading soft clippings, padding, or insertion characters.
Examples

Create a cell array of character vectors containing unaligned sequences, create a cell array of corresponding CIGAR-formatted character vectors associated with a reference sequence of ACGTATGC, and then reconstruct the alignment:

```matlab
r = {'ACGACTGC', 'ACGTTGC', 'AGGTATC'}; % unaligned sequences
c = {'3M1D1M1I3M', '4M1D1P3M', '5M1P1M1D1M'}; % cigar-formatted
aln1 = cigar2align(r, c)
aln1 =
ACG-ATGC
ACGT-TGC
AGGTAT-C
```

Reconstruct the same alignment to display positions in the aligned sequences that correspond to gaps in the reference sequence:

```matlab
aln2 = cigar2align(r, c,'GapsInRef',true)
aln2 =
ACG-ACTGC
ACGT--TGC
AGGTA-T-C
```

Reconstruct the alignment adding an offset padding of 5:

```matlab
aln3 = cigar2align(r, c, 'start', [5 5 5], 'OffsetPad', true)
aln3 =
ACG-ATGC
ACGT-TGC
AGGTAT-C
```

Algorithms

When `cigar2align` reconstructs the alignment, it does not display hard clipped positions (H) or soft clipped positions (S). Also, it does not consider soft clipped positions as start positions for aligned sequences.

Alternatives

If your CIGAR information is captured in the `Signature` property of a `BioMap` object, you can use the `getAlignment` method to construct the alignment.

See Also

`BioMap` | `align2cigar` | `getAlignment` | `getBaseCoverage` | `getCompactAlignment` | `seqalignviewer`

Topics

“Manage Sequence Read Data in Objects”
External Websites
Sequence Read Archive
SAM format specification

Introduced in R2010b
**classperf**

Evaluate classifier performance

**Syntax**

```matlab
classperf
cp = classperf(groundTruth)
cp = classperf(groundTruth,classifierOutput)
```

```matlab
classperf(cp,classifierOutput)
classperf(cp,classifierOutput,testIdx)
classperf(____,Name,Value)
```

**Description**

Classperf without input arguments displays the properties of a `classperformance` object. For more information, see `classperformance` Properties.

```matlab
cp = classperf(groundTruth) creates an empty `classperformance` object `cp` using the true labels `groundTruth` for every observation in your data set.
```

```matlab
cp = classperf(groundTruth,classifierOutput) creates a `classperformance` object `cp` using the true labels `groundTruth`, and then updates the object properties based on the results of the classifier `classifierOutput`. Use this syntax when you want to know the classifier performance on a single validation run.
```

```matlab
classperf(cp,classifierOutput) updates the `classperformance` object `cp` with the results of a classifier `classifierOutput`. Use this syntax to update the performance of the classifier iteratively, such as inside a `for` loop for multiple cross-validation runs.
```

```matlab
classperf(cp,classifierOutput,testIdx) uses `testIdx` to compare the results of the classifier to the true labels and update the object `cp`. `testIdx` represents a subset of the true labels (ground truth) in the current validation.
```

```matlab
classperf(____,Name,Value) specifies additional options with one or more `Name,Value` pair arguments. Specify these options after all other input arguments.
```

**Examples**

**Perform 10-Fold Cross-Validation**

Create indices for the 10-fold cross-validation and classify measurement data for the Fisher iris data set. The Fisher iris data set contains width and length measurements of petals and sepals from three species of irises.

Load the data set.

```matlab
load fisheriris
```
Create indices for the 10-fold cross-validation.

\[
\text{indices} = \text{crossvalind('Kfold',species,10)};
\]

Initialize an object to measure the performance of the classifier.

\[
\text{cp} = \text{classperf(species)};
\]

Perform the classification using the measurement data and report the error rate, which is the ratio of the number of incorrectly classified samples divided by the total number of classified samples.

\[
\text{for } i = 1:10 \\
\quad \text{test} = (\text{indices} == i); \\
\quad \text{train} = \neg \text{test}; \\
\quad \text{class} = \text{classify(meas(test,:),meas(train,:),species(train,:))}; \\
\quad \text{classperf(cp,class,test)}; \\
\end{align*}
\]

\[
\text{cp.ErrorRate} \\
\text{ans} = 0.0200
\]

Suppose you want to use the observation data from the setosa and virginica species only and exclude the versicolor species from cross-validation.

\[
\text{labels} = \{'\text{setosa}','\text{virginica}'\}; \\
\text{indices} = \text{crossvalind('Kfold',species,10,'Classes',labels)};
\]

\[
\text{indices} \text{ now contains zeros for the rows that belong to the versicolor species.}
\]

Perform the classification again.

\[
\text{for } i = 1:10 \\
\quad \text{test} = (\text{indices} == i); \\
\quad \text{train} = \neg \text{test}; \\
\quad \text{class} = \text{classify(meas(test,:),meas(train,:),species(train,:))}; \\
\quad \text{classperf(cp,class,test)}; \\
\end{align*}
\]

\[
\text{cp.ErrorRate} \\
\text{ans} = 0.0160
\]

**Classify Fisher Iris Data Using K-Nearest Neighbor**

Load the data set.

\[
\text{load fisheriris} \\
\text{X} = \text{meas}; \\
\text{Y} = \text{species};
\]

\[
\text{X} \text{ is a numeric matrix that contains four petal measurements for 150 irises.} \text{ Y contains the true class names (labels) of the corresponding iris species.}
\]

Initialize the classperformance object using the true labels.

\[
\text{cp} = \text{classperf(Y)}
\]
Perform the classification using the k-nearest neighbor classifier. Cross-validate the model 10 times by using 145 samples as the training set and 5 samples as the test set. After each cross-validation run, update the classifier performance object with the results.

```matlab
for i = 1:10
    [train,test] = crossvalind('LeaveMOut',Y,5);
    mdl = fitcknn(X(train,:),Y(train),'NumNeighbors',3);
    predictions = predict(mdl,X(test,:));
    classperf(cp,predictions,test);
end
```

Report the classification error rate, which is a ratio of the number of incorrectly classified samples divided by the total number of classified samples.

```matlab
cp.ErrorRate
```

```
ans = 0.0467
```

### Input Arguments

- **groundTruth** — True labels  
  vector of integers | logical vector | string vector | cell array of character vectors

True labels for all observations in your data set, specified as a vector of integers, logical vector, string vector, or cell array of character vectors.

- **classifierOutput** — Classification results  
  vector of integers | logical vector | string vector | cell array of character vectors

Input Arguments
Classification results from a classifier, specified as a vector of integers, logical vector, string vector, or cell array of character vectors. When `classifierOutput` is a cell array of character vectors or string vector, an empty character vector or string represents an inconclusive result. For a vector of integers, `NaN` represents an inconclusive result.

- If you do not specify `testIdx`, `classifierOutput` must be the same size and data type as `groundTruth`.
- If you specify `testIdx` as a vector of integers, `classifierOutput` must have the same number of elements as `testIdx`. If `testIdx` is a logical vector, the number of elements in `classifierOutput` must equal `sum(testIdx)`.

**cp — Classifier performance information**

`classperformance` object

Classifier performance information, specified as a `classperformance` object. For details, see `classperformance` Properties.

**testIdx — Subset of true labels**

Vector of integers | Logical vector

Subset of true labels (`groundTruth`), specified as a vector of integers or logical vector. The `testIdx` argument indicates a subset of true labels (from a test set). The function uses `testIdx` as an index vector to get a subset of labels from `groundTruth`, such as `groundTruth(testIdx)`.

- If `testIdx` is a logical vector, its length must equal the total number of observations (`cp.NumberOfObservations`).
- If `testIdx` is a vector of integers, it cannot contain duplicate integers, and each integer must be greater than `0` but less than or equal to the total number of observations.

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `cp = classperf(groundTruth,classifierOutput,'Positive',[1 2 3])` specifies the labels for the target (diseased) classes.

**Positive — Labels for target classes**

Vector of integers | Logical vector | String vector | Cell array of character vectors

Labels for the target classes, specified as the comma-separated pair consisting of 'Positive' and a vector of integers, logical vector, string vector, or cell array of character vectors.

- If `groundTruth` is a vector of integers, the positive label and negative label (specified by the 'Negative' name-value pair argument) must be vectors of integers.
- If `groundTruth` is a string vector or cell array of character vectors, the positive label and negative label can be string vectors, cell arrays of character vectors, or vectors of positive integers. The entries must be a subset of `grp2idx(groundTruth)`.

By default, the positive label corresponds to the first class returned by `grp2idx(groundTruth)` and the negative label corresponds to all other classes.

The function uses the positive label to set the `TargetClasses` property of the `cp` object.
The positive and negative labels are disjoint subsets of \( \text{unique(groundTruth)} \). For example, suppose you have a data set that contains data from six patients. Five patients have ovarian, lung, prostate, skin, or brain cancer, and one patient does not have cancer. Then \text{ClassLabels} = \{'Ovarian', 'Lung', 'Prostate', 'Skin', 'Brain', 'Healthy'\}. You can test a classifier for lung cancer only by setting the positive label to [2] and the negative label to [1 3 4 5 6]. Alternatively, you can test for any type of cancer by setting the positive label to [1 2 3 4 5] and the negative label to [6].

In clinical tests, the function counts inconclusive values (empty character vector '' or NaN) as false negatives to calculate the specificity and as false positives to calculate the sensitivity. The function does not count any tested observation with its true class not within the union of positive label and negative label. However, if the true class of a tested observation is within the union but its predicted class is not covered by groundTruth, the function counts that observation as inconclusive.

Example: 'Positive',[1 2]

**Negative — Labels for control classes**

vector of integers | logical vector | string vector | cell array of character vectors

Labels for the control classes, specified as the comma-separated pair consisting of 'Negative' and a vector of integers, logical vector, string vector, or cell array of character vectors.

- If groundTruth is a vector of integers, the positive label and negative label (specified by the 'Negative' name-value pair argument) must be vectors of integers.
- If groundTruth is a string vector or cell array of character vectors, the positive label and negative label can be string vectors, cell arrays of character vectors, or vectors of positive integers. The entries must be a subset of grp2idx(groundTruth).

By default, the positive label corresponds to the first class returned by grp2idx(groundTruth) and the negative label corresponds to all other classes.

The function uses the negative label to set the ControlClasses property of the cp object. For details on how the function uses the positive and negative labels, see “Positive” on page 1-0.

Example: 'Negative',[3]

**See Also**

classify | classperformance Properties | crossvalind | grp2idx

**Introduced before R2006a**
classperformance Properties

Classifier performance information

Description

To view the performance-related information of a classifier, create a `classperformance` object by using the `classperf` function. Use dot notation to access the object properties, such as `CorrectRate`, `ErrorRate`, `Sensitivity`, and `Specificity`.

Properties

Name and Description

**Label** — Name of classifier object

```
' ' (default) | character vector
```

Name of the classifier object, specified as a character vector. Use dot notation to set this property.

Example: `cp_kfold`

Data Types: `char`

**Description** — Description of object

```
' ' (default) | character vector
```

Description of the object, specified as a character vector. Use dot notation to set this property.

Example: `performance_data_kfold`

Data Types: `char`

True Labels and Indices

**ClassLabels** — Unique set of true labels

```
vector of positive integers | cell array of character vectors
```

This property is read-only.

Unique set of true labels from `groundTruth`, specified as a vector of positive integers or cell array of character vectors. This property is equivalent to the output when you run `unique(groundTruth)`.

Example: `{'ovarian','liver','normal'}`

Data Types: `double` | `cell`

**GroundTruth** — True labels for all observations

```
vector of positive integers | cell array of character vectors
```

This property is read-only.

True labels for all observations in your data set, specified as a vector of positive integers or cell array of character vectors.

Example: `{'ovarian','liver','normal','ovarian','ovarian','liver'}`
Data Types: double | cell

**NumberOfObservations — Number of observations**

positive integer

This property is read-only.

Number of observations in your data set, specified as a positive integer.

Example: 200

Data Types: double

**ControlClasses — Indices to control classes from true labels**

vector of positive integers

Indices to the control classes from the true labels (`ClassLabels`), specified as a vector of positive integers. This property indicates the control (or negative) classes in the diagnostic test. By default, `ControlClasses` contains all classes other than the first class returned by `grp2idx(groundTruth)`.

You can set this property by using dot notation or the 'Negative' name-value pair argument with the `classperf` function.

Example: [3]

Data Types: double

**TargetClasses — Indices to target classes from true labels**

vector of positive integers

Indices to the target classes from the true labels (`ClassLabels`), specified as a vector of positive integers. This property indicates the target (or positive) classes in the diagnostic test. By default, `TargetClasses` contains the first class returned by `grp2idx(groundTruth)`.

You can set this property by using dot notation or the 'Positive' name-value pair argument with the `classperf` function.

Example: [1 2]

Data Types: double

**Sample and Error Distributions**

**SampleDistribution — Number of evaluations for each sample**

numeric vector

This property is read-only.

Number of evaluations for each sample during the validation, specified as a numeric vector. For example, if you use resubstitution, `SampleDistribution` is a vector of ones and `ValidationCounter = 1`. If you have a 10-fold cross-validation, `SampleDistribution` is also a vector of ones, but `ValidationCounter = 10`.

`SampleDistribution` is useful when performing Monte Carlo partitions of the test sets, and it can help determine if each sample is tested an equal number of times.

Example: [0 0 2 0]
Data Types: double

**ErrorDistribution** — Frequency of misclassification of each sample
numeric vector

This property is read-only.

Frequency of misclassification of each sample, specified as a numeric vector.
Example: [0 0 1 0]

Data Types: double

**SampleDistributionByClass** — Frequency of true classes during validation
numeric vector

This property is read-only.

Frequency of the true classes during the validation, specified as a numeric vector.
Example: [10 10 0]

Data Types: double

**ErrorDistributionByClass** — Frequency of errors for each class
numeric vector

This property is read-only.

Frequency of errors for each class during the validation, specified as a numeric vector.
Example: [0 0 0]

Data Types: double

**Performance Statistics**

**ValidationCounter** — Number of validations
positive integer

This property is read-only.

Number of validations, specified as a positive integer.
Example: 10

Data Types: double

**CountingMatrix** — Classification confusion matrix
numeric array

This property is read-only.

Classification confusion matrix, specified as a numeric array. The order of the rows and columns in the matrix is the same as in `grp2idx(groundTruth)`. Columns represent the true classes, and rows represent the classifier prediction. The last row in CountingMatrix is reserved for counting inconclusive results.
Example: [10 0 0; 10 0; 0 0 0; 0 0 0]
Data Types: double

**CorrectRate — Correct rate of classifier**

positive scalar

This property is read-only.

Correct rate of the classifier, specified as a positive scalar. CorrectRate is defined as the number of correctly classified samples divided by the number of classified samples. Inconclusive results are not counted.

Example: 1

Data Types: double

**ErrorRate — Error rate of classifier**

positive scalar

This property is read-only.

Error rate of the classifier, specified as a positive scalar. ErrorRate is defined as the number of incorrectly classified samples divided by the number of classified samples. Inconclusive results are not counted.

Example: 0

Data Types: double

**LastCorrectRate — Correct rate of classifier during last run**

positive scalar

This property is read-only.

Correct rate of the classifier during the last validation run, specified as a positive scalar. In contrast with CorrectRate, LastCorrectRate only applies to the evaluated samples from the most recent validation run of the classifier performance object.

Example: 1

Data Types: double

**LastErrorRate — Error rate of classifier during last validation**

positive scalar

This property is read-only.

Error rate of the classifier during the last validation run, specified as a positive scalar. In contrast with ErrorRate, LastErrorRate only applies to the evaluated samples from the most recent validation run of the classifier performance object.

Example: 0

Data Types: double

**InconclusiveRate — Inconclusive rate of classifier**

positive scalar

This property is read-only.
Inconclusive rate of the classifier, specified as a positive scalar. `InconclusiveRate` is defined as the number of nonclassified (inconclusive) samples divided by the total number of samples.

Example: 0
Data Types: double

**ClassifiedRate — Classified rate of classifier**

positive scalar

This property is read-only.

Classified rate of the classifier, specified as a positive scalar. `ClassifiedRate` is defined as the number of classified samples divided by the total number of samples.

Example: 1
Data Types: double

**Sensitivity — Sensitivity of classifier**

positive scalar

This property is read-only.

Sensitivity of the classifier, specified as a positive scalar. `Sensitivity` is defined as the number of correctly classified positive samples divided by the number of true positive samples.

Inconclusive results that are true positives are counted as errors for computing `Sensitivity`. In other words, inconclusive results can decrease the diagnostic value of the test.

Example: 1
Data Types: double

**Specificity — Specificity of classifier**

positive scalar

This property is read-only.

Specificity of the classifier, specified as a positive scalar. `Specificity` is defined as the number of correctly classified negative samples divided by the number of true negative samples.

Inconclusive results that are true negatives are counted as errors for computing `Specificity`. In other words, inconclusive results can decrease the diagnostic value of the test.

Example: 0.8
Data Types: double

**PositivePredictiveValue — Positive predictive value of classifier**

positive scalar

This property is read-only.

Positive predictive value of the classifier, specified as a positive scalar. `PositivePredictiveValue` is defined as the number of correctly classified positive samples divided by the number of positive classified samples.

Inconclusive results are classified as negative when computing `PositivePredictiveValue`. 
NegativePredictiveValue — Negative predictive value of classifier
positive scalar

This property is read-only.

Negative predictive value of the classifier, specified as a positive scalar. NegativePredictiveValue is defined as the number of correctly classified negative samples divided by the number of negative classified samples.

Inconclusive results are classified as positive when computing NegativePredictiveValue.

Example: 1
Data Types: double

PositiveLikelihood — Positive likelihood of classifier
positive scalar

This property is read-only.

Positive likelihood of the classifier, specified as a positive scalar. PositiveLikelihood is defined as Sensitivity / (1 - Specificity).

Example: 5
Data Types: double

NegativeLikelihood — Negative likelihood of classifier
positive scalar

This property is read-only.

Negative likelihood of the classifier, specified as a positive scalar. NegativeLikelihood is defined as (1 - Sensitivity)/Specificity.

Example: 0
Data Types: double

Prevalence — Prevalence of classifier
positive scalar

This property is read-only.

Prevalence of the classifier, specified as a positive scalar. Prevalence is defined as the number of true positive samples divided by the total number of samples.

Example: 1
Data Types: double

DiagnosticTable — Diagnostic table
2-by-2 numeric array

This property is read-only.
Diagnostic table, specified as a two-by-two numeric array. The first row indicates the number of samples classified as positive, with the number of true positives in the first column and the number of false positives in the second column. The second row indicates the number of samples classified as negative, with the number of false negatives in the first column and the number of true negatives in the second column.

Correct classifications appear in the diagonal elements and errors appear in the off-diagonal elements. Inconclusive results are considered errors and are counted in the off-diagonal elements. For an example, see “Diagnostic Table Example” on page 1-348.

Example: \([20 \ 0; 0 \ 0]\)

Data Types: double

More About

Diagnostic Table Example

Suppose that a cancer study of 10 patients yields these results.

<table>
<thead>
<tr>
<th>Patient</th>
<th>Classifier Output</th>
<th>Has Cancer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Positive</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Positive</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Positive</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Positive</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>Negative</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>Negative</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Negative</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>Negative</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>Negative</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>Inconclusive</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Using these results, the function computes the DiagnosticTable as follows:

![Diagnostic Table Example Diagram]

See Also

classify | classperf | crossvalind | grp2idx

Introduced before R2006a
cleave

Cleave amino acid sequence with enzyme

Syntax

\[ \text{Fragments} = \text{cleave} (\text{SeqAA}, \text{Enzyme}) \]

\[ \text{Fragments} = \text{cleave} (\text{SeqAA}, \text{PeptidePattern}, \text{Position}) \]

\[ [\text{Fragments}, \text{CuttingSites}] = \text{cleave}(...) \]

\[ [\text{Fragments}, \text{CuttingSites}, \text{Lengths}] = \text{cleave}(...) \]

\[ [\text{Fragments}, \text{CuttingSites}, \text{Lengths}, \text{Missed}] = \text{cleave}(...) \]

cleave(..., 'PartialDigest', \text{PartialDigestValue}, ...)

cleave(..., 'MissedSites', \text{MissedSitesValue}, ...)

cleave(..., 'Exception', \text{ExceptionValue}, ...)

Input Arguments

<table>
<thead>
<tr>
<th>SeqAA</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string containing single-letter codes specifying an amino acid sequence.</td>
</tr>
<tr>
<td></td>
<td>• Row vector of integers specifying an amino acid sequence.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that contains an amino acid sequence, such as returned by fastaread, getgenpept, genpeptread, getpdb, or pdbread.</td>
</tr>
<tr>
<td></td>
<td>Examples: 'ARN' or ([1 \ 2 \ 3]).</td>
</tr>
</tbody>
</table>

| Enzyme | Character vector or string specifying a name or abbreviation code for an enzyme or compound for which the literature specifies a cleavage rule. |

**Tip** Use the cleavelookup function to display the names of enzymes and compounds in the cleavage rule library.

<table>
<thead>
<tr>
<th>PeptidePattern</th>
<th>Short amino acid sequence to search for in SeqAA, a larger sequence. PeptidePattern can be any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string</td>
</tr>
<tr>
<td></td>
<td>• Vector of integers</td>
</tr>
<tr>
<td></td>
<td>• Regular expression (MATLAB)</td>
</tr>
</tbody>
</table>

| Position | Integer from 0 to the length of the PeptidePattern, that specifies a position in the PeptidePattern to cleave. |

**Note** Position 0 corresponds to the N terminal end of PeptidePattern.
### PartialDigestValue
- Value from 0 to 1 (default) specifying the probability that a cleavage site will be cleaved.

### MissedSitesValue
- Nonnegative integer specifying the maximum number of missed cleavage sites. The output includes all possible peptide fragments that can result from missing MissedSitesValue or less cleavage sites. Default is 0, which is equivalent to an ideal digestion.

### ExceptionValue
- Regular expression (MATLAB) specifying an exception rule to the cleavage rule associated with Enzyme. By default, exception rules are only applied in the case of trypsin, and all other enzymes have no exception rule, which is specified as an empty character vector. To prevent the use of the default exceptions for trypsin, use an empty character vector as the exception rule.

To see the regular expression for trypsin’s exception rules, check the Cleave Lookup table.

### Output Arguments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fragments</td>
<td>Cell array of character vectors representing the fragments from the cleavage.</td>
</tr>
<tr>
<td>CuttingSites</td>
<td>Numeric vector containing indices representing the cleavage sites.</td>
</tr>
<tr>
<td>Lengths</td>
<td>Numeric vector containing the length of each fragment.</td>
</tr>
<tr>
<td>Missed</td>
<td>Numeric vector containing the number of missed cleavage sites for every peptide fragment.</td>
</tr>
</tbody>
</table>

### Description

Fragments = cleave(SeqAA, Enzyme) cuts SeqAA, an amino acid sequence, into parts at the cleavage sites specific for Enzyme, a character vector or string specifying a name or abbreviation code for an enzyme or compound for which the literature specifies a cleavage rule. It returns Fragments, a cell array of character vectors representing the fragments from the cleavage.

Tip Use the cleavelookup function to display the names of enzymes and compounds in the cleavage rule library.

Fragments = cleave(SeqAA, PeptidePattern, Position) cuts SeqAA, an amino acid sequence, into parts at the cleavage sites specified by a peptide pattern and position.

[ Fragments, CuttingSites ] = cleave(...) returns a numeric vector containing indices representing the cleavage sites.
The `cleave` function adds a 0 to the list, so \( \text{numel}(\text{CuttingSites}) = \text{numel}(\text{Fragments}) \). Use `CuttingSites + 1` to point to the first amino acid of every fragment respective to the original sequence.

\[
[\text{Fragments}, \text{CuttingSites}, \text{Lengths}] = \text{cleave}(...) \text{ returns a numeric vector containing the length of each fragment.}
\]

\[
[\text{Fragments}, \text{CuttingSites}, \text{Lengths}, \text{Missed}] = \text{cleave}(...) \text{ returns a numeric vector containing the number of missed cleavage sites for every fragment.}
\]

cleave(..., 'PropertyName', PropertyValue, ...) calls cleave with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each PropertyName in single quotation marks. Each PropertyName is case insensitive. These property name/property value pairs are as follows:

cleave(..., 'PartialDigest', PartialDigestValue, ...) simulates a partial digestion where PartialDigestValue is the probability of a cleavage site being cut. PartialDigestValue is a value from 0 to 1 (default).

This table lists some common proteases and their cleavage sites.

<table>
<thead>
<tr>
<th>Protease</th>
<th>Peptide Pattern</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspartic acid N</td>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td>Chymotrypsin</td>
<td><a href="?!P">WYF</a></td>
<td>1</td>
</tr>
<tr>
<td>Glutamine C</td>
<td><a href="?!P">ED</a></td>
<td>1</td>
</tr>
<tr>
<td>Lysine C</td>
<td><a href="?!P">K</a></td>
<td>1</td>
</tr>
<tr>
<td>Trypsin</td>
<td><a href="?!P">KR</a></td>
<td>1</td>
</tr>
</tbody>
</table>

cleave(..., 'MissedSites', MissedSitesValue, ...) returns all possible peptide fragments that can result from missing MissedSitesValue or less cleavage sites. MissedSitesValue is a nonnegative integer. Default is 0, which is equivalent to an ideal digestion.

cleave(..., 'Exception', ExceptionValue, ...) specifies an exception rule to the cleavage rule associated with Enzyme. ExceptionValue is a regular expression (MATLAB). By default, exception rules are only applied in the case of trypsin, and all other enzymes have no exception rule, which is specified as an empty character vector. To prevent the use of the default exceptions for trypsin, specify an empty character vector as the exception rule.

**Examples**

**Cleave a sequence**

This example shows how to cleave a sequence using trypsin.

Retrieve a protein sequence from the GenPept database.

```matlab
S = getgenpept('AAA59174');
```

Cleave the sequence using trypsin's cleavage rules and all known exceptions.

```matlab
parts = cleave(S.Sequence,'trypsin');
```
Display the first ten fragments.

\[
\text{parts}(1:10)
\]

\[
\text{ans} = \\
'MGTGGR' \\
'R' \\
'GAAAAPLLVAVAALLLGAAGHLYPGEVCPGMDIR' \\
'NNLTR' \\
'LHELENCSVIEGHLQILLMFK' \\
'TRPEDFR' \\
'DLSFPK' \\
'LIMIDYLLLFR' \\
'VYGLESLK' \\
'DLFPNLTVIR'
\]

Cleave the sequence using trypsin's cleavage rules and a single specific exception rule.

\[
\text{parts} = \text{cleave}(	ext{S.Sequence},'\text{trypsin}','\text{exception}','\text{KD}');
\]

\[
\text{parts}(1:10)
\]

\[
\text{ans} = \\
'MGTGGR' \\
'R' \\
'GAAAAPLLVAVAALLLGAAGHLYPGEVCPGMDIR' \\
'NNLTR' \\
'LHELENCSVIEGHLQILLMFK' \\
'TRPEDFR' \\
'DLSFPK' \\
'LIMIDYLLLFR' \\
'VYGLESLKDLFPNLTVIR' \\
'GSR'
\]

Cleave the sequence using one of trypsin's cleavage rules, which is to cleave after K or R when the next residue is not P.

\[
[\text{parts, sites, lengths}] = \text{cleave}(	ext{S.Sequence},'[\text{KR}](!P)',1);
\]

\[
\text{for } i = 1:10 \\
\text{fprintf}('%5d %5d %s \n', \text{sites}(i), \text{lengths}(i), \text{parts}(i)) \\
\end
\]

\[
0 6  MGTGGR \\
6 1  R \\
7 34  GAAAAPLLVAVAALLLGAAGHLYPGEVCPGMDIR \\
41 5  NNLTR \\
46 21  LHELENCSVIEGHLQILLMFK \\
67 7  TRPEDFR \\
74 6  DLSFPK \\
80 12  LIMITDYLLLFR \\
92 8  VYGLESLK \\
100 10  DLFPNLTVIR
\]

Cut the sequence using trypsin, allowing for 1 missed cleavage site.
[parts2, sites2, lengths2, missed] = cleave(S.Sequence,'trypsin','missedsites',1);

Display the first 10 fragments that have 1 missed cleavage site.

```matlab
idx = find(missed);
for i = 1:10
    fprintf('%5d%5d   %s
',sites2(idx(i)),lengths2(idx(i)),parts2{idx(i)})
end
```

```
0    7   MGTGRR
6   35   RGAAAAAPLLVAVAALLLLGAAGHLYPGEVCPGMDIR
7   39   GAAAAAPLLVAVAALLLLGAAGHLYPGEVCPGMDIRNNLTR
41   26   NNLTRLHELENCSVIEGHLQILLMFK
46   28   LHELENCSVIEGHLQILLMFKTRPEDFR
67   13   TRPEDFRDLSFPK
74   18   DLSFPKLIMITYLLLFR
80   20   LIMITYLLLLFRVYGLESLK
92   18   VYGLESLKDLPNLTIVIR
100   13   DLFPNLTIVIRGSR
```

**See Also**
cleavelookup | rebasecuts | regexp | restrict | seqshowwords

**Introduced before R2006a**
cleavelookup

Find cleavage rule for enzyme or compound

**Syntax**

cleavelookup
cleavelookup('Code', CodeValue)
cleavelookup('Name', NameValue)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeValue</td>
<td>Character vector specifying a code representing an abbreviation code for an enzyme or compound. For valid codes, see the table Cleave Lookup.</td>
</tr>
<tr>
<td>NameValue</td>
<td>Character vector specifying an enzyme or compound name. For valid names, see the table Cleave Lookup.</td>
</tr>
</tbody>
</table>

**Description**

cleavelookup displays a table of abbreviation codes, cleavage positions, cleavage patterns, and full names of enzymes and compounds for which cleavage rules are specified by the cleavage rule library. Trysin’s exception rules are also listed in the table. For more information, see the ExPASy PeptideCutter tool.
## Cleave Lookup

<table>
<thead>
<tr>
<th>Code</th>
<th>Position</th>
<th>Pattern</th>
<th>Full Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARG-C</td>
<td>1</td>
<td>R</td>
<td>ARG-C proteinase</td>
</tr>
<tr>
<td>ASP-N</td>
<td>2</td>
<td>D</td>
<td>ASP-N endopeptidase</td>
</tr>
<tr>
<td>BNPS</td>
<td>1</td>
<td>W</td>
<td>BNPS-Skatole</td>
</tr>
<tr>
<td>CASP1</td>
<td>1</td>
<td>(?&lt;=FWYL)\w[HAT])D(?=[^PEDQKR])</td>
<td>Caspase 1</td>
</tr>
<tr>
<td>CASP2</td>
<td>1</td>
<td>(?&lt;=DVA)D(?=[^PEDQKR])</td>
<td>Caspase 2</td>
</tr>
<tr>
<td>CASP3</td>
<td>1</td>
<td>(?&lt;=DMQ)D(?=[^PEDQKR])</td>
<td>Caspase 3</td>
</tr>
<tr>
<td>CASP4</td>
<td>1</td>
<td>(?&lt;=LEV)D(?=[^PEDQKR])</td>
<td>Caspase 4</td>
</tr>
<tr>
<td>CASP5</td>
<td>1</td>
<td>(?&lt;=[LW]EH)D</td>
<td>Caspase 5</td>
</tr>
<tr>
<td>CASP6</td>
<td>1</td>
<td>(?&lt;=VE[HI])D(?=[^PEDQKR])</td>
<td>Caspase 6</td>
</tr>
<tr>
<td>CASP7</td>
<td>1</td>
<td>(?&lt;=DEV)D(?=[^PEDQKR])</td>
<td>Caspase 7</td>
</tr>
<tr>
<td>CASP8</td>
<td>1</td>
<td>(?&lt;=[IL]ET)D(?=[^PEDQKR])</td>
<td>Caspase 8</td>
</tr>
<tr>
<td>CASP9</td>
<td>1</td>
<td>(?&lt;=LEH)D</td>
<td>Caspase 9</td>
</tr>
<tr>
<td>CASP10</td>
<td>1</td>
<td>(?&lt;=IEA)D</td>
<td>Caspase 10</td>
</tr>
<tr>
<td>CH-HI</td>
<td>1</td>
<td>(<a href="?=%5E%5BP%5D">FY</a>)</td>
<td>(W(?=[^MP]))</td>
</tr>
<tr>
<td>CH-LO</td>
<td>1</td>
<td>(<a href="?=%5E%5BP%5D">FLY</a>)</td>
<td>(W(?=[^MP]))</td>
</tr>
<tr>
<td>CLOST</td>
<td>1</td>
<td>R</td>
<td>Clostripain</td>
</tr>
<tr>
<td>CNBR</td>
<td>1</td>
<td>M</td>
<td>CNBR</td>
</tr>
<tr>
<td>ELANE</td>
<td>1</td>
<td>[AV]</td>
<td>Neutrophil elastase</td>
</tr>
<tr>
<td>ENTKIN</td>
<td>1</td>
<td>(?&lt;=DE)K</td>
<td>Enterokinase</td>
</tr>
<tr>
<td>FACTXA</td>
<td>1</td>
<td>(?&lt;=[AFGILTVM]DE)G</td>
<td>Factor XA</td>
</tr>
<tr>
<td>FORMIC</td>
<td>1</td>
<td>D</td>
<td>Formic acid</td>
</tr>
<tr>
<td>GLUEND</td>
<td>1</td>
<td>E</td>
<td>Glutamyl endopeptidase</td>
</tr>
<tr>
<td>GRANB</td>
<td>1</td>
<td>(?&lt;=IEP)D</td>
<td>Granzyme B</td>
</tr>
<tr>
<td>HYDROX</td>
<td>1</td>
<td>N(?=G)</td>
<td>Hydroylamine</td>
</tr>
<tr>
<td>IODOB</td>
<td>1</td>
<td>W</td>
<td>Iodosobenzoic acid</td>
</tr>
<tr>
<td>LYSC</td>
<td>1</td>
<td>K</td>
<td>Lysc</td>
</tr>
<tr>
<td>NLATEV</td>
<td>1</td>
<td>(?&lt;=Y\w)Q(?=[GS])</td>
<td>NLA in tobacco etch virus</td>
</tr>
<tr>
<td>NTCB</td>
<td>1</td>
<td>C</td>
<td>NTCB</td>
</tr>
<tr>
<td>PEPS</td>
<td>1</td>
<td>((?&lt;=[^HKR][^P])<a href="?=FL">^R</a>[^P])</td>
<td>Pepsin PH = 1.3</td>
</tr>
<tr>
<td>PEPS2</td>
<td>1</td>
<td>((?&lt;=[^HKR][^P])<a href="?=FLW">^R</a>[^P])</td>
<td>Pepsin PH &gt; 2</td>
</tr>
</tbody>
</table>
cleavelookup('Code', CodeValue) displays the cleavage position, cleavage pattern, and full name of the enzyme or compound specified by CodeValue, a character vector specifying an abbreviation code.

cleavelookup('Name', NameValue) displays the cleavage position, cleavage pattern, and abbreviation code of the enzyme or compound specified by NameValue, a character vector specifying an enzyme or compound name.

Examples

Example 1.8. Using cleavelookup with an Enzyme Name

Display the cleavage position, cleavage pattern, and abbreviation code of the enzyme Caspase 1.

cleavelookup('name', 'CASPASE 1')

ans =
1 (?<=\w[HAT])D(?=\^PEDQKR) CASP1

Example 1.9. Using cleavelookup with an Abbreviation Code

Display the cleavage position, cleavage pattern, and full name of the enzyme with a abbreviation code of CASP1.

cleavelookup('code', 'CASP1')

ans =
1 (?<=\w[HAT])D(?=\^PEDQKR) CASPASE 1

See Also

cleave|rebasecuts|restrict

Introduced in R2008b
cluster (phytree)

Validate clusters in phylogenetic tree

Syntax

LeafClusters = cluster(Tree, Threshold)
[LeafClusters, NodeClusters] = cluster(Tree, Threshold)
[LeafClusters, NodeClusters, Branches] = cluster(Tree, Threshold)
cluster(..., 'Criterion', CriterionValue, ...)
cluster(..., 'MaxClust', MaxClustValue, ...)
cluster(..., 'Distances', DistancesValue, ...)

Input Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>Phylogenetic tree object created, such as created with the phytree constructor function.</td>
</tr>
<tr>
<td>Threshold</td>
<td>Scalar specifying a threshold value.</td>
</tr>
<tr>
<td>CriterionValue</td>
<td>Character vector or string specifying the criterion to determine the number of clusters as a function of the species pairwise distances. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'maximum' (default) — Maximum within cluster pairwise distance (W_{max}). Cluster splitting stops when W_{max} ≤ Threshold.</td>
</tr>
<tr>
<td></td>
<td>• 'median' — Median within cluster pairwise distance (W_{med}). Cluster splitting stops when W_{med} ≤ Threshold.</td>
</tr>
<tr>
<td></td>
<td>• 'average' — Average within cluster pairwise distance (W_{avg}). Cluster splitting stops when W_{avg} ≤ Threshold.</td>
</tr>
<tr>
<td></td>
<td>• 'ratio' — Between/within cluster pairwise distance ratio, defined as</td>
</tr>
<tr>
<td></td>
<td>BW_{rat} = (trace(B)/(k - 1)) / (trace(W)/(n - k))</td>
</tr>
<tr>
<td></td>
<td>where B and W are the between- and within-scatter matrices, respectively. k is the number of clusters, and n is the number of species in the tree. Cluster splitting stops when BW_{rat} ≥ Threshold.</td>
</tr>
<tr>
<td></td>
<td>• 'gain' — Within cluster pairwise distance gain, defined as</td>
</tr>
<tr>
<td></td>
<td>W_{gain} = (trace(W_{old})/ (trace(W) - 1) * (n - k - 1))</td>
</tr>
<tr>
<td></td>
<td>where W and W_{old} are the within-scatter matrices for k and k - 1, respectively. k is the number of clusters, and n is the number of species in the tree. Cluster splitting stops when W_{gain} ≤ Threshold.</td>
</tr>
<tr>
<td></td>
<td>• 'silhouette' — Average silhouette width (SW_{avg}). SW_{avg} ranges from -1 to +1. Cluster splitting stops when SW_{avg} ≥ Threshold. For more information, see silhouette.</td>
</tr>
</tbody>
</table>
### Output Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LeafClusters</strong></td>
<td>Column vector containing a cluster index for each species (leaf) in <code>Tree</code>, a phylogenetic tree object.</td>
</tr>
<tr>
<td><strong>NodeClusters</strong></td>
<td>Column vector containing the cluster index for each leaf node and branch node in <code>Tree</code>.</td>
</tr>
<tr>
<td><strong>Branches</strong></td>
<td>Two-column matrix containing, for each step in the algorithm, the index of the branch being considered and the value of the criterion. Each row corresponds to a step in the algorithm. The first column contains branch indices, and the second column contains criterion values.</td>
</tr>
</tbody>
</table>

**Tip** Use the `LeafClusters` or `NodeClusters` output vectors with the handle returned by the `plot` method to modify graphic elements of the phylogenetic tree object. For more information, see “Examples” on page 1-359.

### Description

`LeafClusters = cluster(Tree, Threshold)` returns a column vector containing a cluster index for each species (leaf) in a phylogenetic tree object. It determines the optimal number of clusters as follows:

- Starting with two clusters ($k = 2$), selects the partition that optimizes the criterion specified by the 'Criterion' property.
• Increments $k$ by 1 and again selects the optimal partition
• Continues incrementing $k$ and selecting the optimal partition until a criterion value $= Threshold$ or $k = \text{the maximum number of clusters (that is, number of leaves)}$
• From all possible $k$ values, selects the $k$ value whose partition optimizes the criterion

$[\text{LeafClusters}, \text{NodeClusters}] = \text{cluster}(\text{Tree}, \text{Threshold})$ returns a column vector containing the cluster index for each leaf node and branch node in $\text{Tree}$.

$[\text{LeafClusters}, \text{NodeClusters}, \text{Branches}] = \text{cluster}(\text{Tree}, \text{Threshold})$ returns a two-column matrix containing, for each step in the algorithm, the index of the branch being considered and the value of the criterion. Each row corresponds to a step in the algorithm. The first column contains branch indices, and the second column contains criterion values.

$\text{cluster}(\ldots, '\text{PropertyName}', \text{PropertyValue}, \ldots)$ calls $\text{cluster}$ with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each $\text{PropertyName}$ in single quotation marks. Each $\text{PropertyName}$ is case insensitive. These property name/property value pairs are as follows:

$\text{cluster}(\ldots, '\text{Criterion}', \text{CriterionValue}, \ldots)$ specifies the criterion to determine the number of clusters as a function of the species pairwise distances.

$\text{cluster}(\ldots, '\text{MaxClust}', \text{MaxClustValue}, \ldots)$ specifies the maximum number of possible clusters for the tested partitions. Default is the number of leaves in the tree.

$\text{cluster}(\ldots, '\text{Distances}', \text{DistancesValue}, \ldots)$ substitutes the patristic distances in $\text{Tree}$ with a user-provided pairwise distance matrix.

**Examples**

Validate the clusters in a phylogenetic tree:

```matlab
% Read sequences from a multiple alignment file into a MATLAB structure
gagaa = multialignread('aagag.aln');

% Build a phylogenetic tree from the sequences
gag_tree = seqneighjoin(seqpdist(gagaa),'equivar',gagaa);

% Validate the clusters in the tree and find the best partition
% using the 'gain' criterion
[i,j] = cluster(gag_tree,[],'criterion','gain','maxclust',10);

% Use the returned vector of indices to color the branches of each % cluster in a plot of the tree
h = plot(gag_tree);
set(h.BranchLines(j==2),'Color','b')
set(h.BranchLines(j==1),'Color','r')```

1-359
References


See Also
cluster | phytree | phytreeread | phytreeviewer | plot | seqlinkage | seqneighjoin | seqpdist | silhouette | view

Topics
phytree object on page 1-1274
**clustergram**

Object containing hierarchical clustering analysis data

**Description**

The clustergram function creates a clustergram object. The object contains hierarchical clustering analysis data that you can view in a heatmap and dendrogram.

**Creation**

**Syntax**

clustergram(data)
clustergram(data,Name,Value)

**Description**

cgObj = clustergram(data) performs hierarchical clustering analysis on the values in data. The returned clustergram object cgObj contains analysis data and displays a dendrogram and heatmap.

cgObj = clustergram(data,Name,Value) sets the object properties on page 1-361 using name-value pairs. For example, clustergram(data,'Standardize','column') standardizes the values along the columns of data. You can specify multiple name-value pairs. Enclose each property name in quotes.

**Input Arguments**

data — Source data
DataMatrix object | numeric matrix

Source data, specified as a DataMatrix object or numeric matrix. Typically, if the matrix contains gene expression data, each row corresponds to a gene and each column corresponds to a sample.

**Name-Value Pair Arguments**

Use comma-separated name-value pair arguments to set the object properties. Enclose each property name in single quotes.

Example: cg = clustergram(data,'Colormap',redbluecmap,'Annotate',true)

**Properties**

**Standardize** — Dimension for standardizing data values

none (default) | row | column | 3 | 2 | 1

Dimension for standardizing data values, specified as a character vector, string, or positive integer. Choices are:
• 'column' or 1 — Standardize along the columns of data.
• 'row' or 2 — Standardize along the rows of data.
• 'none' or 3 — Do not standardize.

If you specify 'column' or 'row', the function transforms the standardized values so that the mean is 0 and the standard deviation is 1 in the specified dimension.

Example: 'column'
Data Types: double | char | string

**Symmetric** — Flag to make the heatmap color scale symmetric around zero

true (default) | false

Flag to make the heatmap color scale symmetric around zero, specified as true or false.

Example: false
Data Types: logical

**ImputeFun** — Name of function or function handle to impute missing data

character vector | cell array

Name of a function or function handle to impute missing data, specified as a character vector or cell array. If you specify a cell array, the first element must be the name of a function or function handle, and the remaining elements must be name-value pairs used as inputs to the function. Missing data points are colored gray in the heatmap.

If data points are missing, use this property to impute the missing values. Otherwise, the clustergram function errors.

Example: 'func1'
Data Types: char

**Colormap** — Heatmap colors

redgreencmap (default) | matrix | name of function handle

Heatmap colors, specified as a three-column (M-by-3) matrix of red-green-blue (RGB) values or the name of a function handle that returns a colormap, such as redgreencmap or redbluecmap.

The default colormap is redgreencmap, in which red represents values above the mean, black represents the mean, and green represents values below the mean of a row (gene) across all columns (samples).

Example: redbluecmap
Data Types: double | char

**ColumnLabels** — Column labels

[1x0 double] (default) | string vector | cell array of character vectors | numeric vector

Column labels, specified as a string vector, cell array of character vectors, or numeric vector. The size of the vector must match the number of columns in the input data.

If the number of column labels is 200 or more, the labels do not appear in the clustergram plot.

Example: ["sample1","sample2","sample3"]
Data Types: double | string | cell

**RowLabels — Row labels**
\[
[] \text{(default)} | \text{string vector} | \text{cell array of character vectors} | \text{numeric vector}
\]
Row labels, specified as a string vector, cell array of character vectors, or numeric vector. The size of the vector must match the number of rows in the input data.

If the number of row labels is 200 or more, the labels do not appear in the clustergram plot.

Example: ['gene1','gene2','gene3']
Data Types: double | string | cell

**ColumnLabelsRotate — Orientation of column labels**
90 (default) | numeric scalar
Orientation of column labels, specified as a numeric scalar. Specify the value of rotation in degrees (positive angles cause counterclockwise rotation).

Example: 30
Data Types: double

**RowLabelsRotate — Orientation of row labels**
0 (default) | numeric scalar
Orientation of row labels, specified as a numeric scalar. Specify the value of rotation in degrees (positive angles cause counterclockwise rotation).

Example: 30
Data Types: double

**Annotate — Flag to display data values in heatmap**
false (default) | true
Flag to display data values in the heatmap, specified as true or false.

Example: true
Data Types: logical

**AnnotPrecision — Display precision of data values**
2 (default) | numeric scalar
Display precision of data values in the heatmap, specified as a numeric scalar. The default number of digits of precision is 2.

Example: 3
Data Types: double

**LabelsWithMarkers — Flag to display colored markers for row and column labels**
false (default) | true
Flag to display colored markers instead of colored text for the row and column labels, specified as true or false.

Example: true
Data Types: logical

**AnnotColor — Text color of displayed data values**

'w' (default) | character vector | string | three-element numeric vector

Text color of displayed data values in the heatmap, specified as a character vector, string, or three-element numeric vector. For example, to use cyan, you can enter [0 1 1], 'c', "c", "cyan", or 'cyan'. For details, see “Color Options” on page 1-377.

Example: 'red'

Data Types: char | string | double

**DisplayRange — Display range of standardize values**

3 (default) | positive scalar

Display range of standardize values, specified as a positive scalar.

The default value 3 means that there is a color variation for values between -3 and 3, but values greater than 3 are the same color as 3, and values less than -3 are the same color as -3.

For example, if you specify redgreencmap for the 'Colormap' property, pure red represents values greater than or equal to the specified display range value and pure green represents values less than or equal to the negative of the specified display range value.

Example: 3

Data Types: double

**ColumnLabelsColor — Color information for column labels**

[] (default) | structure | structure array

Warning: This property will be removed in a future release. Set LabelsWithMarkers to true for colored markers instead of colored texts.

Color information for column labels, specified as a structure or structure array.

For a single structure, you must specify the following fields.

- **Labels** — Cell array of character vectors specifying column labels listed in the ColumnLabels property.
- **Colors** — Character vector or string specifying a color for the column labels. If this field is empty, the default color (black) is used.

For a structure array, you must specify a single element in each field for each structure.

- **Labels** — Character vector or string specifying a column label listed in the ColumnLabels property.
- **Colors** — Character vector or string specifying a color for the column labels. If this field is empty, the default color (black) is used.

For more information on specifying colors, see “Color Options” on page 1-377.

Data Types: struct
RowLabelsColor — Color information for row labels

[] (default) | structure | structure array

**Warning** This property will be removed in a future release. Set LabelsWithMarkers to true for colored markers instead of colored texts.

Color information for row labels, specified as a structure or structure array.

For a single structure, you must specify the following fields.

- **Labels** — Cell array of character vectors specifying row labels listed in the RowLabels property.
- **Colors** — Character vector or string specifying a color for the row labels. If this field is empty, the default color (black) is used.

For a structure array, you must specify a single element in each field for each structure.

- **Labels** — Character vector or string specifying a row label listed in the RowLabels property.
- **Colors** — Character vector or string specifying a color for the row labels. If this field is empty, the default color (black) is used.

For more information on specifying colors, see “Color Options” on page 1-377.

**Cluster — Dimension for data clustering**

'all' (default) | 1 | 2 | 3 | 'column' | 'row'

Dimension for data clustering, specified as a positive integer, character vector, or string. Choices are:

- 'column' or 1 — Cluster along the columns of data only, which results in clustered rows.
- 'row' or 2 — Cluster along the rows of data only, which results in clustered columns.
- 'all' or 3 — Cluster along the columns of data, then cluster along the rows of row-clustered data.

Example: 2

Data Types: double | char | string

ColumnGroupMarker — Information for annotating groups of columns

structure | structure array

Information for annotating groups of columns, specified as a structure or structure array.

If you specify a single structure, each field must contain a cell array of elements. If you specify a structure array, each structure must have a single element in each field.

The fields are:

- **GroupNumber** — Scalar specifying the column group number to annotate.
- **Annotation** — Character vector specifying text to annotate the column group.
- **Color** — Character vector or three-element vector of RGB values specifying a color to label the column group. For more information on specifying colors, see “Color Options” on page 1-377. If this field is empty, the default value is 'blue'.

Data Types: struct
ColumnPDist — Distance metric to pass to pdist function
'euclidean' (default) | character vector | cell array

Distance metric to pass to the pdist function to calculate the pairwise distances between columns, specified as a character vector or cell array. Specify a cell array if the distance metric requires extra arguments. For example, to use the Minkowski distance with an exponent p, specify {'minkowski', p}.

Example: 'jaccard'
Data Types: char | cell

Dendrogram — Color threshold information to pass to dendrogram function
scalar | two-element numeric vector | character vector | cell array of character vectors

Color threshold information to pass to the dendrogram function to create a dendrogram plot, specified as a scalar, two-element numeric vector, character vector, or cell array of character vectors. This option sets the ‘ColorThreshold’ property of the dendrogram plot. If you specify a two-element numeric vector or cell array, the first element is for the rows, and the second element is for the columns.

Data Types: double | cell

DisplayRatio — Ratio of space that row and column dendrograms occupy
1/5 (default) | scalar between 0 and 1 | two-element vector

Ratio of space that the row and column dendrograms occupy relative to the heatmap, specified as a scalar between 0 and 1 or two-element vector. If you specify a scalar, the function uses it as the ratio for both row and column dendrograms. If you specify a two-element vector, the function uses the first element for the ratio of the row dendrogram width to the heatmap width, and the second element for the ratio of the column dendrogram height to the heatmap height. The second element is ignored for one-dimensional clustergrams.

Example: 0.5
Data Types: double

Linkage — Linkage method to create hierarchical cluster tree
'average' (default) | character vector | two-element cell array of character vectors

Linkage method passed to the linkage function to create the hierarchical cluster tree for rows and columns, specified as a character vector or two-element cell array of character vectors. If you specify a cell array, the function uses the first element for linkage between rows, and the second element for linkage between columns.

Example: 'centroid'
Data Types: char | cell

LogTrans — Flag to log_2 transform data
false (default) | true

Flag to log_2 transform the data from natural scale, specified as true or false.

Example: true
Data Types: logical
**OptimalLeafOrder — Flag to calculate optimal leaf order**

`true` | `false`

Flag to calculate the optimal leaf order that maximizes the similarity between neighboring leaves, specified as `true` or `false`. The default value depends on the size of the input data. If the number of rows or columns in `data` exceeds 1500, the default value is `false`. Otherwise, the default value is `true`.

Disabling the optimal leaf ordering calculation can be useful when working with large datasets because this calculation consumes a lot of memory and time.

Example: `true`

**Data Types:** logical

**RowGroupMarker — Information for annotating groups of rows**

`structure` | `structure array`

Information for annotating groups of rows, specified as a structure or structure array.

If you specify a single structure, each field must contain a cell array of elements. If you specify a structure array, each structure must have a single element in each field.

The fields are

- `GroupNumber` — Scalar specifying the column group number to annotate.
- `Annotation` — Character vector specifying text to annotate the column group.
- `Color` — Character vector or three-element vector of RGB values specifying a color to label the column group. For more information on specifying colors, see “Color Options” on page 1-377. If this field is empty, the default value is 'blue'.

**Data Types:** struct

**RowPDist — Distance metric to pass to pdist function**

`'euclidean'` (default) | `character vector` | `cell array`

Distance metric to pass to the `pdist` function to calculate the pairwise distances between rows, specified as a character vector or cell array. Specify a cell array if the distance metric requires extra arguments. For example, to use the Minkowski distance with an exponent `p`, specify `{'minkowski',p}`.

Example: `'jaccard'`

**Data Types:** char | cell

**ShowDendrogram — Flag to show dendrogram tree diagrams with clustergram**

`'on'` (default) | `'off'`

Flag to show the dendrogram tree diagrams with the clustergram, specified as `'on'` or `'off'`.

Example: `'off'`

**Data Types:** char

**Object Functions**

- `view` Display heatmap or clustergram
Examples

Perform hierarchical clustering on gene expression data

Load microarray data containing gene expression levels of *Saccharomyces cerevisiae* (yeast) during the metabolic shift from fermentation to respiration (Derisi, J. et al., 1997).

load filteredyeastdata

This MAT-file includes three variables, which are added to the MATLAB® workspace:

- **yeastvalues** - A matrix of gene expression data from *Saccharomyces cerevisiae* during the metabolic shift from fermentation to respiration
- **genes** - A cell array of GenBank® accession numbers for labeling the rows in yeastvalues
- **times** - A vector of time values for labeling the columns in yeastvalues

Create a clustergram object and display the heat map from the gene expression data in the first 30 rows of the yeastvalues matrix and standardize along the rows of data.

cgo = clustergram(yeastvalues(1:30,:), 'Standardize', 'Row')

Clustergram object with 30 rows of nodes and 7 columns of nodes.
Use the `set` method and the `genes` and `times` vectors to add meaningful row and column labels to the clustergram.

```matlab
set(cgo,'RowLabels',genes(1:30),'ColumnLabels',times)
```
Add a color bar to the clustergram by clicking the Insert Colorbar button on the toolbar.

View a data tip containing the intensity value, row label, and column label for a specific area of the heat map by clicking the Data Cursor button on the toolbar, then clicking an area in the heat map. To delete this data tip, right-click it, then select Delete Current Datatip.

Display intensity values for each area of the heat map by clicking the Annotate button on the toolbar. Click the Annotate button again to remove the intensity values.

Tip: If the amount of data is large enough, the cells within the clustergram are too small to display the intensity annotations. Zoom in to see the intensity annotations.

Remove the dendrogram tree diagrams from the figure by clicking the Show Dendrogram button on the toolbar. Click it again to display the dendrograms.

Use the get method to display the properties of the clustergram object, cgo.

get(cgo)

    Cluster: 'ALL'
    RowPDist: {'Euclidean'}
Change the clustering parameters by changing the linkage method and changing the color of the groups of nodes in the dendrogram whose linkage is less than a threshold of 3.

```matlab
set(cgo,'Linkage','complete','Dendrogram',3)
```
Place the cursor on a branch node in the dendrogram to highlight (in blue) the group associated with it. Press and hold the mouse button to display a data tip listing the group number and the nodes (genes or samples) in the group.
Right-click a branch node in the dendrogram to display a menu of options.
The following options are available:

- **Set Group Color** - Change the cluster group color.
- **Print Group to Figure** - Print the group to a Figure window.
- **Copy Group to New Clustergram** - Copy the group to a new Clustergram window.
- **Export Group to Workspace** - Create a clustergram object of the group in the MATLAB Workspace.
- **Export Group Info to Workspace** - Create a structure containing information about the group in the MATLAB Workspace. The structure contains these fields:

1. **GroupNames** - Cell array of character vectors containing the names of the row or column groups.
2. **RowNodeNames** - Cell array of character vectors containing the names of the row nodes.
3. **ColumnNodeNames** - Cell array of text character vectors containing the names of the column nodes.
4. **ExprValues** - An M-by-N matrix of intensity values, where M and N are the number of row nodes and of column nodes respectively. If the matrix contains gene expression data, typically each row corresponds to a gene and each column corresponds to sample.
Create a clustergram object in the MATLAB Workspace of Group 18 by right-clicking it, then selecting Export Group to Workspace. In the Export to Workspace dialog box, type Group18, then click OK.

Use the view method to view the clustergram object, Group18.

view(Group18)

View all the gene expression data using a diverging red and blue colormap and standardize along the rows of data.

cgo_all = clustergram(yeastvalues,'Colormap','redbluecmap','Standardize','Row')

Clustergram object with 614 rows of nodes and 7 columns of nodes.
Create structure arrays to specify marker colors and annotations for two groups of rows (510 and 593) and two groups of columns (4 and 5).

```matlab
rm = struct('GroupNumber',[510,593],'Annotation',{'A','B'},... 'Color',{'b','m'});
```

```matlab
cm = struct('GroupNumber',{4,5},'Annotation',{'Time1','Time2'},... 'Color',{{1 1 0},{0.6 0.6 1}});
```

Use the 'RowGroupMarker' and 'ColumnGroupMarker' properties to add the color markers and annotations to the clustergram.

```matlab
set(cgo_all,'RowGroupMarker',rm,'ColumnGroupMarker',cm)
```
More About

Color Options

The following lists the predefined colors and their RGB triplet equivalents. The short names and long names are character vectors that specify one of eight preset colors. The RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color; the intensities must be in the range [0 1].

<table>
<thead>
<tr>
<th>RGB Triplet</th>
<th>Short Name</th>
<th>Long Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1 1 0]</td>
<td>y</td>
<td>yellow</td>
</tr>
<tr>
<td>[1 0 1]</td>
<td>m</td>
<td>magenta</td>
</tr>
<tr>
<td>[0 1 1]</td>
<td>c</td>
<td>cyan</td>
</tr>
<tr>
<td>[1 0 0]</td>
<td>r</td>
<td>red</td>
</tr>
<tr>
<td>[0 1 0]</td>
<td>g</td>
<td>green</td>
</tr>
<tr>
<td>[0 0 1]</td>
<td>b</td>
<td>blue</td>
</tr>
<tr>
<td>RGB Triplet</td>
<td>Short Name</td>
<td>Long Name</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>[1 1 1]</td>
<td>w</td>
<td>white</td>
</tr>
<tr>
<td>[0 0 0]</td>
<td>k</td>
<td>black</td>
</tr>
</tbody>
</table>

**See Also**
HeatMap | redblue cmap | redgreen cmap

**Introduced before R2006a**
**clusterGroup**

Select cluster group

**Syntax**

```matlab
clusterGroup(cgObj1, groupIndex, dim)
cgObj2 = clusterGroup(cgObj1, groupIndex, dim)
cgObj2 = clusterGroup(cgObj1, groupIndex, dim, 'InfoOnly', tf_InfoOnly)
cgObj2 = clusterGroup(cgObj1, groupIndex, dim, 'Color', colorChoice)
```

**Description**

`clusterGroup(cgObj1, groupIndex, dim)` selects and highlights the cluster group specified by `groupIndex` along the row or column dimension `dim` in the Clustergram window.

`cgObj2 = clusterGroup(cgObj1, groupIndex, dim)` creates and returns the clustergram object `cgObj2` for the specified cluster group. This syntax is equivalent to selecting the Export Group to Workspace option from the context menu after right-clicking a group in the Clustergram window.

`cgObj2 = clusterGroup(cgObj1, groupIndex, dim, 'InfoOnly', tf_InfoOnly)` specifies whether to return the cluster group as a structure or a clustergram object.

`cgObj2 = clusterGroup(cgObj1, groupIndex, dim, 'Color', colorChoice)` specifies the color to use to highlight the dendrogram of the selected cluster group.

**Examples**

**Select Cluster Group**

Load microarray data containing some measurements of gene expression levels.

```matlab
load filteredyeastdata
```

Create a clustergram object and display a heatmap from the gene expression data.

```matlab
cgo = clustergram(yeastvalues(1:30,:), 'Standardize', 'Row')
```

Clustergram object with 30 rows of nodes and 7 columns of nodes.
From the command line, use the `clusterGroup` function to select and highlight the Group 4 column cluster in the Clustergram window.

cgroup4 = clusterGroup(cgo,4,'column')

Clustergram object with 30(30) rows of nodes and 4(7) columns of nodes.
Input Arguments

cgObj1 — Clustergram object
classic.Vector
Clustergram object, specified as a clustergram object.

groupId — Group index for cluster group
positive integer
Group index for a cluster group in cgObj1, specified as a positive integer.

Example: 4

Data Types: double

dim — Dimension of cluster group
'row' | 'column'
Dimension of the cluster group, specified as 'row' or 'column'.
Example: 'row'
Data Types: char

tf_InfoOnly — Flag to return structure
false (default) | true

Flag to return a structure instead of a clustergram object, specified as true or false.
Example: true
Data Types: logical

colorChoice — Color to highlight dendrogram
character vector | three-element numeric vector

Color to highlight the dendrogram of the selected cluster group, specified as a character vector or
three-element numeric vector of RGB values. For example, to use cyan, specify [0 1 1], 'c', or
'cyan'.

For more information on specifying colors, see “Color Options” on page 1-382.
Example: 'red'
Data Types: double | char

Output Arguments

cgObj2 — Selected cluster group
clustergram object | structure

Selected cluster group, returned as a clustergram object or structure. If you specify 'InfoOnly' as
true, the function returns cgObj2 as a structure.

More About

Color Options

The following lists the predefined colors and their RGB triplet equivalents. The short names and long
names are character vectors that specify one of eight preset colors. The RGB triplet is a three-
element row vector whose elements specify the intensities of the red, green, and blue components of
the color; the intensities must be in the range [0 1].

<table>
<thead>
<tr>
<th>RGB Triplet</th>
<th>Short Name</th>
<th>Long Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1 1 0]</td>
<td>y</td>
<td>yellow</td>
</tr>
<tr>
<td>[1 0 1]</td>
<td>m</td>
<td>magenta</td>
</tr>
<tr>
<td>[0 1 1]</td>
<td>c</td>
<td>cyan</td>
</tr>
<tr>
<td>[1 0 0]</td>
<td>r</td>
<td>red</td>
</tr>
<tr>
<td>[0 1 0]</td>
<td>g</td>
<td>green</td>
</tr>
<tr>
<td>[0 0 1]</td>
<td>b</td>
<td>blue</td>
</tr>
<tr>
<td>[1 1 1]</td>
<td>w</td>
<td>white</td>
</tr>
<tr>
<td>RGB Triplet</td>
<td>Short Name</td>
<td>Long Name</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>[0 0 0]</td>
<td>k</td>
<td>black</td>
</tr>
</tbody>
</table>

**See Also**

HeatMap, clustergram

**Introduced in R2009b**
codonbias

Calculate codon frequency for each amino acid coded for in nucleotide sequence

Syntax

\[
\text{CodonFreq} = \text{codonbias}(\text{SeqNT})
\]

\[
\text{CodonFreq} = \text{codonbias}(\text{SeqNT}, \ldots \text{'GeneticCode'}, \text{GeneticCodeValue}, \ldots)
\]

\[
\text{CodonFreq} = \text{codonbias}(\text{SeqNT}, \ldots \text{'Frame'}, \text{FrameValue}, \ldots)
\]

\[
\text{CodonFreq} = \text{codonbias}(\text{SeqNT}, \ldots \text{'Reverse'}, \text{ReverseValue}, \ldots)
\]

\[
\text{CodonFreq} = \text{codonbias}(\text{SeqNT}, \ldots \text{'Ambiguous'}, \text{AmbiguousValue}, \ldots)
\]

\[
\text{CodonFreq} = \text{codonbias}(\text{SeqNT}, \ldots \text{'Pie'}, \text{PieValue}, \ldots)
\]

Input Arguments

<table>
<thead>
<tr>
<th>SeqNT</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a nucleotide sequence</td>
</tr>
<tr>
<td></td>
<td>• Row vector of integers specifying a nucleotide sequence</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that contains a nucleotide sequence, such as returned by fastaread, fastqread, emblread, getembl, genbankread, or getgenbank</td>
</tr>
<tr>
<td></td>
<td>Valid characters include A, C, G, T, and U.</td>
</tr>
<tr>
<td></td>
<td>codonbias does not count ambiguous nucleotides or gaps.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GeneticCodeValue</th>
<th>Integer, character vector, or string specifying a genetic code number or code name from the table Genetic Code. Default is 1 or 'Standard'.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tip</td>
<td>If you use a code name, you can truncate the name to the first two letters of the name.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FrameValue</th>
<th>Integer specifying a reading frame in the nucleotide sequence. Choices are 1 (default), 2, or 3.</th>
</tr>
</thead>
</table>

| ReverseValue | Controls the return of the codon frequency for the reverse complement sequence of the nucleotide sequence specified by SeqNT. Choices are true or false (default). |
### AmbiguousValue

Character vector or string specifying how to treat codons containing ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N). Choices are:

- `'ignore'` (default) — Skips codons containing ambiguous characters
- `'prorate'` — Counts codons containing ambiguous characters and distributes them proportionately in the appropriate codon fields. For example, the counts for the codon ART are distributed evenly between the AAT and AGT fields.
- `'warn'` — Skips codons containing ambiguous characters and displays a warning.

### PieValue

Controls the creation of a figure of 20 pie charts, one for each amino acid. Choices are true or false (default).

### Output Arguments

| CodonFreq | MATLAB structure containing a field for each amino acid, each of which contains the associated codon frequencies as percentages. |

### Description

Many amino acids are coded by two or more nucleic acid codons. However, the probability that a specific codon (from all possible codons for an amino acid) is used to code an amino acid varies between sequences. Knowing the frequency of each codon in a protein coding sequence for each amino acid is a useful statistic.

\[
\text{CodonFreq} = \text{codonbias}(\text{SeqNT})
\]

calculates the codon frequency in percent for each amino acid coded for in `SeqNT`, a nucleotide sequence, and returns the results in `CodonFreq`, a MATLAB structure containing a field for each amino acid.

\[
\text{CodonFreq} = \text{codonbias}(\text{SeqNT}, \ldots \text{PropertyName}', \text{PropertyValue}, \ldots)
\]
calls `codonbias` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

\[
\text{CodonFreq} = \text{codonbias}(\text{SeqNT}, \ldots \text{'GeneticCode', GeneticCodeValue}, \ldots)
\]
specifies a genetic code. Choices for `GeneticCodeValue` are an integer, character vector, or string specifying a code number or code name from the table Genetic Code. If you use a code name, you can truncate the name to the first two characters of the name. Default is 1 or `'Standard'`.

**Tip** If you use a code name, you can truncate the name to the first two letters of the name.

\[
\text{CodonFreq} = \text{codonbias}(\text{SeqNT}, \ldots \text{'Frame', FrameValue}, \ldots)
\]
calculates the codon frequency in the reading frame specified by `FrameValue`, which can be 1 (default), 2, or 3.

\[
\text{CodonFreq} = \text{codonbias}(\text{SeqNT}, \ldots \text{'Reverse', ReverseValue}, \ldots)
\]
controls the return of the codon frequency for the reverse complement of the nucleotide sequence specified by `SeqNT`. Choices are true or false (default).
CodonFreq = codonbias(SeqNT, ...'Ambiguous', AmbiguousValue, ...) specifies how to treat codons containing ambiguous nucleotide characters. Choices are 'ignore' (default), 'prorate', and 'warn'.

CodonFreq = codonbias(SeqNT, ...'Pie', PieValue, ...) controls the creation of a figure of 20 pie charts, one for each amino acid. Choices are true or false (default).

Genetic Code

<table>
<thead>
<tr>
<th>Code Number</th>
<th>Code Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Standard</td>
</tr>
<tr>
<td>2</td>
<td>Vertebrate Mitochondrial</td>
</tr>
<tr>
<td>3</td>
<td>Yeast Mitochondrial</td>
</tr>
<tr>
<td>4</td>
<td>Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma/Spiroplasma</td>
</tr>
<tr>
<td>5</td>
<td>Invertebrate Mitochondrial</td>
</tr>
<tr>
<td>6</td>
<td>Ciliate, Dasycladacean, and Hexamita Nuclear</td>
</tr>
<tr>
<td>9</td>
<td>Echinoderm Mitochondrial</td>
</tr>
<tr>
<td>10</td>
<td>Euplotid Nuclear</td>
</tr>
<tr>
<td>11</td>
<td>Bacterial and Plant Plastid</td>
</tr>
<tr>
<td>12</td>
<td>Alternative Yeast Nuclear</td>
</tr>
<tr>
<td>13</td>
<td>Ascidian Mitochondrial</td>
</tr>
<tr>
<td>14</td>
<td>Flatworm Mitochondrial</td>
</tr>
<tr>
<td>15</td>
<td>Blepharisma Nuclear</td>
</tr>
<tr>
<td>16</td>
<td>Chlorophycean Mitochondrial</td>
</tr>
<tr>
<td>21</td>
<td>Trematode Mitochondrial</td>
</tr>
<tr>
<td>22</td>
<td>Scenedesmus Obliquus Mitochondrial</td>
</tr>
<tr>
<td>23</td>
<td>Thraustochytrium Mitochondrial</td>
</tr>
</tbody>
</table>

Examples

Calculate Codon Frequency for Each Amino Acid

Import a nucleotide sequence from the GenBank database into the MATLAB software. For example, retrieve the DNA sequence that codes for a human insulin receptor.

S = getgenbank('M10051');

Calculate the codon frequency for each amino acid coded for by the DNA sequence, and then plot the results.

cb = codonbias(S.Sequence,'PIE',true)
Get the codon frequency for the alanine (A) amino acid.

```
cb.Ala
ans =
```

```
Codon: {'GCA' 'GCC' 'GCG' 'GCT'}
Freq: [0.1600 0.3867 0.2533 0.2000]
```

**See Also**

aminolookup | codoncount | geneticcode | nt2aa

**Introduced before R2006a**
codoncount

Count codons in nucleotide sequence

Syntax

Codons = codoncount(SeqNT)
[Codons, CodonArray] = codoncount(SeqNT)

... = codoncount(SeqNT, ...'Frame', FrameValue, ...)
... = codoncount(SeqNT, ...'Reverse', ReverseValue, ...)
... = codoncount(SeqNT, ...'Ambiguous', AmbiguousValue, ...)
... = codoncount(SeqNT, ...'Figure', FigureValue, ...)
... = codoncount(SeqNT, ...'GeneticCode', GeneticCodeValue, ...)

Input Arguments

<table>
<thead>
<tr>
<th>SeqNT</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers</td>
</tr>
<tr>
<td></td>
<td>• Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that contains a nucleotide sequence, such as returned by fastaread, fastqread, emblread, getembl, genbankread, or getgenbank.</td>
</tr>
<tr>
<td>Examples:</td>
<td>'ACGT' or [1 2 3 4]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FrameValue</th>
<th>Integer specifying a reading frame in the nucleotide sequence. Choices are 1 (default), 2, or 3.</th>
</tr>
</thead>
</table>

| ReverseValue | Controls the return of the codon count for the reverse complement sequence of the nucleotide sequence specified by SeqNT. Choices are true or false (default). |
Character vector or string specifying how to treat codons containing ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N). Choices are:

- 'ignore' (default) — Skips codons containing ambiguous characters
- 'bundle' — Counts codons containing ambiguous characters and reports the total count in the Ambiguous field of the Codons output structure.
- 'prorate' — Counts codons containing ambiguous characters and distributes them proportionately in the appropriate codon fields containing standard nucleotide characters. For example, the counts for the codon ART are distributed evenly between the AAT and AGT fields.
- 'warn' — Skips codons containing ambiguous characters and displays a warning.

Controls the display of a heat map of the codon counts. Choices are true or false (default).

Integer, character vector, or string specifying a genetic code number or code name from the table Genetic Code. Default is 1 or 'Standard'. You can also specify 'None'.

If you use a code name, you can truncate the name to the first two letters of the name.

MATLAB structure containing fields for the 64 possible codons (AAA, AAC, AAG, ..., TTG, TTT), which contain the codon counts in SeqNT.

A 4-by-4-by-4 array containing the raw count data for each codon. The three dimensions correspond to the three positions in the codon, and the indices to each element are represented by 1 = A, 2 = C, 3 = G, and 4 = T. For example, the element (2, 3, 4) in the array contains the number of CGT codons.

Codons = codoncount(SeqNT) counts the codons in SeqNT, a nucleotide sequence, and returns the codon counts in Codons, a MATLAB structure containing fields for the 64 possible codons (AAA, AAC, AAG, ..., TTG, TTT).

- For sequences that have codons containing the character U, these codons are added to the corresponding codons containing a T.
- If the sequence contains gaps indicated by a hyphen (-), then codons containing gaps are ignored.
- If the sequence contains unrecognized characters, then codons containing these characters are ignored, and the following warning message appears:

  Warning: Unknown symbols appear in the sequence. These will be ignored.
[Codons, CodonArray] = codoncount(SeqNT) returns CodonArray, a 4-by-4-by-4 array containing the raw count data for each codon. The three dimensions correspond to the three positions in the codon, and the indices to each element are represented by 1 = A, 2 = C, 3 = G, and 4 = T. For example, the element (2,3,4) in the array contains the number of CGT codons.

... = codoncount(SeqNT, ...'PropertyName', PropertyValue, ...) calls codoncount with optional properties that use property name/property value pairs. You can specify one or more properties in any order. EachPropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = codoncount(SeqNT, ...'Frame', FrameValue, ...) counts the codons in the reading frame specified by FrameValue, which can be 1 (default), 2, or 3.

... = codoncount(SeqNT, ...'Reverse', ReverseValue, ...) controls the return of the codon count for the reverse complement sequence of SeqNT. Choices are true or false (default).

... = codoncount(SeqNT, ...'Ambiguous', AmbiguousValue, ...) specifies how to treat codons containing ambiguous nucleotide characters. Choices are:

• 'ignore' (default)
• 'bundle'
• 'prorate'
• 'warn'

... = codoncount(SeqNT, ...'Figure', FigureValue, ...) controls the display of a heat map of the codon counts. Choices are true or false (default).

... = codoncount(SeqNT, ...'GeneticCode', GeneticCodeValue, ...) controls the overlay of a grid on the heat map figure. The grid groups the synonymous codons according to GeneticCodeValue.

Examples

Count codons in a nucleotide sequence

seq = randseq(1000);
codons = codoncount(seq)
codons = struct with fields:
    AAA: 11
    AAC: 5
    AAG: 8
    AAT: 6
    ACA: 6
    ACC: 7
    ACG: 4
    ACT: 7
    AGA: 6
    AGC: 9
    AGG: 5
    AGT: 2
    ATA: 6
Count the codons in the second frame for the reverse complement of a sequence.

\[ r2\text{codons} = \text{codoncount}(\text{seq}, 'Frame', 2, 'Reverse', \text{true}) \]
r2codons = struct with fields:
    AAA: 5
    AAC: 2
    AAG: 5
    AAT: 6
    ACA: 8
    ACC: 4
    ACG: 5
    ACT: 2
    AGA: 5
    AGC: 5
    AGG: 5
    AGT: 7
    ATA: 4
    ATC: 4
    ATG: 10
    ATT: 6
    CAA: 8
    CAC: 5
    CAG: 4
    CAT: 4
    CCA: 5
    CCG: 10
    CCA: 5
    CCT: 5
    CGA: 5
    CGC: 8
    CGG: 8
    CGT: 4
    CTA: 1
    CTC: 5
    CTG: 7
    CTT: 8
    GAA: 1
    GAC: 6
    GAG: 7
    GAT: 4
    GCA: 1
    GCC: 7
    GCG: 6
    GCT: 9
    GGA: 2
    GGC: 2
    GGG: 4
    GGT: 7
    GTA: 4
    GTC: 6
    GTG: 5
    GTT: 5
    TAA: 6
    TAC: 2
    TAG: 4
    TAT: 6
    TCA: 4
    TCC: 6
    TCG: 7
    TCT: 6
    TGA: 6
TGC: 3  
TGG: 5  
TGT: 6  
TTA: 2  
TTC: 5  
TTG: 3  
TTT: 11

Create a heat map of the codons and overlay a grid that groups the synonymous codons according to the standard genetic code.

```
codoncount(seq,'Figure', true);
```

<table>
<thead>
<tr>
<th>Codon</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>11</td>
</tr>
<tr>
<td>AAC</td>
<td>5</td>
</tr>
<tr>
<td>AAG</td>
<td>8</td>
</tr>
<tr>
<td>AAT</td>
<td>6</td>
</tr>
<tr>
<td>ACA</td>
<td>6</td>
</tr>
<tr>
<td>ACC</td>
<td>7</td>
</tr>
<tr>
<td>ACG</td>
<td>4</td>
</tr>
<tr>
<td>ACT</td>
<td>7</td>
</tr>
<tr>
<td>AGA</td>
<td>6</td>
</tr>
<tr>
<td>AGC</td>
<td>9</td>
</tr>
<tr>
<td>AGG</td>
<td>5</td>
</tr>
<tr>
<td>AGT</td>
<td>2</td>
</tr>
<tr>
<td>ATA</td>
<td>6</td>
</tr>
<tr>
<td>ATC</td>
<td>4</td>
</tr>
<tr>
<td>ATG</td>
<td>4</td>
</tr>
<tr>
<td>ATT</td>
<td>6</td>
</tr>
<tr>
<td>CAA</td>
<td>3</td>
</tr>
<tr>
<td>CAC</td>
<td>5</td>
</tr>
<tr>
<td>CAG</td>
<td>7</td>
</tr>
<tr>
<td>CAT</td>
<td>10</td>
</tr>
<tr>
<td>CCA</td>
<td>5</td>
</tr>
<tr>
<td>CCC</td>
<td>4</td>
</tr>
<tr>
<td>CGG</td>
<td>8</td>
</tr>
<tr>
<td>CCT</td>
<td>5</td>
</tr>
<tr>
<td>CGA</td>
<td>7</td>
</tr>
<tr>
<td>CGC</td>
<td>6</td>
</tr>
<tr>
<td>CGG</td>
<td>5</td>
</tr>
<tr>
<td>CGT</td>
<td>5</td>
</tr>
<tr>
<td>CTA</td>
<td>4</td>
</tr>
<tr>
<td>CTC</td>
<td>7</td>
</tr>
<tr>
<td>CTG</td>
<td>4</td>
</tr>
<tr>
<td>CTT</td>
<td>5</td>
</tr>
<tr>
<td>GAA</td>
<td>5</td>
</tr>
<tr>
<td>GAC</td>
<td>6</td>
</tr>
<tr>
<td>GAG</td>
<td>5</td>
</tr>
<tr>
<td>GAT</td>
<td>4</td>
</tr>
<tr>
<td>GCA</td>
<td>3</td>
</tr>
<tr>
<td>GCC</td>
<td>2</td>
</tr>
<tr>
<td>GCG</td>
<td>8</td>
</tr>
<tr>
<td>GCT</td>
<td>5</td>
</tr>
<tr>
<td>GGA</td>
<td>6</td>
</tr>
<tr>
<td>GGC</td>
<td>7</td>
</tr>
<tr>
<td>GGG</td>
<td>10</td>
</tr>
<tr>
<td>GGT</td>
<td>4</td>
</tr>
<tr>
<td>GTA</td>
<td>2</td>
</tr>
<tr>
<td>GTC</td>
<td>6</td>
</tr>
<tr>
<td>GTG</td>
<td>5</td>
</tr>
<tr>
<td>GTT</td>
<td>2</td>
</tr>
<tr>
<td>TAA</td>
<td>2</td>
</tr>
<tr>
<td>TAC</td>
<td>4</td>
</tr>
<tr>
<td>TAG</td>
<td>1</td>
</tr>
<tr>
<td>TAT</td>
<td>4</td>
</tr>
<tr>
<td>TCA</td>
<td>6</td>
</tr>
<tr>
<td>TCC</td>
<td>2</td>
</tr>
<tr>
<td>TCG</td>
<td>5</td>
</tr>
<tr>
<td>TCT</td>
<td>5</td>
</tr>
<tr>
<td>TGA</td>
<td>4</td>
</tr>
<tr>
<td>TGC</td>
<td>1</td>
</tr>
<tr>
<td>TGG</td>
<td>5</td>
</tr>
<tr>
<td>TGT</td>
<td>8</td>
</tr>
<tr>
<td>TTA</td>
<td>6</td>
</tr>
<tr>
<td>TTC</td>
<td>1</td>
</tr>
<tr>
<td>TTG</td>
<td>8</td>
</tr>
<tr>
<td>TTT</td>
<td>5</td>
</tr>
</tbody>
</table>
See Also
aaccount | basecount | baselookup | codonbias | dimercount | nmercount | ntdensity | seqcomplement | seqrcomplement | seqreverse | seqwordcount

Introduced before R2006a
colnames (DataMatrix)

Retrieve or set column names of DataMatrix object

Syntax

ReturnColNames = colnames(DMObj)
ReturnColNames = colnames(DMObj, ColIndices)
DMObjNew = colnames(DMObj, ColIndices, ColNames)

Input Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMObj</td>
<td>DataMatrix object, such as created by DataMatrix (object constructor).</td>
</tr>
<tr>
<td>ColIndices</td>
<td>One or more columns in DMObj, specified by any of the following:</td>
</tr>
<tr>
<td></td>
<td>• Positive integer</td>
</tr>
<tr>
<td></td>
<td>• Vector of positive integers</td>
</tr>
<tr>
<td></td>
<td>• Character vector specifying a column name</td>
</tr>
<tr>
<td></td>
<td>• Cell array of character vectors</td>
</tr>
<tr>
<td></td>
<td>• Logical vector</td>
</tr>
<tr>
<td>ColNames</td>
<td>Column names specified by any of the following:</td>
</tr>
<tr>
<td></td>
<td>• Numeric vector</td>
</tr>
<tr>
<td></td>
<td>• Cell array of character vectors</td>
</tr>
<tr>
<td></td>
<td>• Character array</td>
</tr>
<tr>
<td></td>
<td>• Single character vector, which is used as a prefix for column names, with column numbers appended to the prefix</td>
</tr>
<tr>
<td></td>
<td>• Logical true or false (default). If true, unique column names are assigned using the format col1, col2, col3, etc. If false, no column names are assigned.</td>
</tr>
</tbody>
</table>

Note The number of elements in ColNames must equal the number of elements in ColIndices.

Output Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReturnColNames</td>
<td>Character vector or cell array of character vectors containing column names in DMObj.</td>
</tr>
<tr>
<td>DMObjNew</td>
<td>DataMatrix object created with names specified by ColIndices and ColNames.</td>
</tr>
</tbody>
</table>

Description

ReturnColNames = colnames(DMObj) returns ReturnColNames, a cell array of character vectors specifying the column names in DMObj, a DataMatrix object.
ReturnColNames = colnames(DMObj, ColIndices) returns the column names specified by ColIndices. ColIndices can be a positive integer, vector of positive integers, character vector specifying a column name, cell array of character vectors, or a logical vector.

DMObjNew = colnames(DMObj, ColIndices, ColNames) returns DMObjNew, a DataMatrix object with columns specified by ColIndices set to the names specified by ColNames. The number of elements in ColIndices must equal the number of elements in ColNames.

See Also
DataMatrix | rownames

Topics
DataMatrix object on page 1-532

Introduced in R2008b
**combine**

**Class:** bioma.data.ExptData  
**Package:** bioma.data

Combine two ExptData objects

**Syntax**

\[ \text{NewEDObj} = \text{combine}(\text{EDObj1}, \text{EDObj2}) \]

**Description**

\[ \text{NewEDObj} = \text{combine}(\text{EDObj1}, \text{EDObj2}) \] combines data from two ExptData objects and returns a new ExptData object. The number and names of features (rows) in both ExptData objects must match. The number and names of samples (columns) in both ExptData objects must match.

**Input Arguments**

**EDObj#**

Object of the bioma.data.ExptData class.

**Default:**

**See Also**

bioma.data.ExptData

**Topics**

“Representing Expression Data Values in ExptData Objects”
combine

**Class:** bioma.data.MetaData
**Package:** bioma.data

Combine two MetaData objects

**Syntax**

\[ \text{NewMDObj} = \text{combine}(\text{MDObj1}, \text{MDObj2}) \]

**Description**

\( \text{NewMDObj} = \text{combine}(\text{MDObj1}, \text{MDObj2}) \) combines data from two MetaData objects and returns a new MetaData object. The sample or feature names in the two MetaData objects being combined must be unique. The variable names in the two MetaData objects can be unique or the same. If a variable name is common to the two MetaData objects, then the variable occupies one column in the new MetaData object. Variable names unique to either of the two MetaData objects occupy their own column and contain values only for the samples or features where the variable is present.

**Input Arguments**

\( \text{MDObj#} \)

Object of the bioma.data.MetaData class.

**Default:**

**See Also**

bioma.data.MetaData

**Topics**

“Representing Sample and Feature Metadata in MetaData Objects”
**combine**

**Class:** bioma.data.MIAME  
**Package:** bioma.data

Combine two MIAME objects

**Syntax**

\[ \text{NewMIAMEObj} = \text{combine(MIAMEObj1, MIAMEObj2)} \]

**Description**

\[ \text{NewMIAMEObj} = \text{combine(MIAMEObj1, MIAMEObj2)} \] combines data from two MIAME objects and returns a new MIAME object. The combine method concatenates the properties of the two objects together.

**Input Arguments**

MIAMEObj#

Object of the bioma.data.MIAME class.

**Default:**

**Examples**

Construct two MIAME objects, and then combine them:

```matlab
% Create a MATLAB structure containing GEO Series data
geoStruct1 = getgeodata('GSE4616');
% Create a second MATLAB structure containing GEO Series data
geoStruct2 = getgeodata('GSE11287');
% Import bioma.data package to make constructor function available
import bioma.data.*
% Construct MIAME object from the first structure
MIAMEobj1 = MIAME(geoStruct1);
% Construct MIAME object from the second structure
MIAMEobj2 = MIAME(geoStruct2);
% Combine the two MIAME objects
newMIAMEobj = combine(MIAMEobj1, MIAMEobj2)
```

**See Also**

bioma.data.MIAME

**Topics**

“Representing Experiment Information in a MIAME Object”
### combine

Combine two objects

#### Syntax

```
combinedData = combine(data1, data2)
newObj = combine(data1, data2, 'Name', objName)
```

#### Description

`combinedData = combine(data1, data2)` combines sequence data from two objects of the same class (BioRead or BioMap) and returns the data `combinedData` in a new object. The `combine` function concatenates the properties of the two objects.

`newObj = combine(data1, data2, 'Name', objName)` specifies the name `objName` for `newObj`.

#### Examples

**Combine NGS Data**

Load the first data set.

```
br1 = BioRead('ex1.sam')
```

```
br1 = BioRead with properties:
    Quality: [1501x1 File indexed property]
    Sequence: [1501x1 File indexed property]
    Header: [1501x1 File indexed property]
    NSeqs: 1501
    Name: ''
```

Load the second data set.

```
br2 = BioRead('ex2.sam')
```

```
br2 = BioRead with properties:
    Quality: [3307x1 File indexed property]
    Sequence: [3307x1 File indexed property]
    Header: [3307x1 File indexed property]
    NSeqs: 3307
    Name: ''
```

Combine the two data sets. Set the name of the new object to 'combinedData'.

```
```
br3 = combine(br1,br2,'Name','combinedData')

br3 =
  BioRead with properties:
  
  Quality: [4808x1 cell]
  Sequence: [4808x1 cell]
  Header: [4808x1 cell]
  NSeqs: 4808
  Name: 'combinedData'

**Input Arguments**

data1 — First sequence read data
BioRead | BioMap

First sequence read data to be combined with the second sequence read data, specified as a BioRead or BioMap object.

data2 — Second sequence read data
BioRead | BioMap

Second sequence read data to be combined with the first sequence read data, specified as a BioRead or BioMap object.

objName — Name of combined object
'' (default) | character vector | string

Name of the combined object, specified as a character vector or string.

**See Also**

BioMap | BioRead

**Topics**

“Manage Sequence Read Data in Objects”

**Introduced in R2010a**
**conncomp (biograph)**

Find strongly or weakly connected components in biograph object

**Syntax**

\[
[S, C] = \text{conncomp}(BGObj)
\]

\[
[S, C] = \text{conncomp}(BGObj, \ldots, 'Directed', DirectedValue, \ldots)
\]

\[
[S, C] = \text{conncomp}(BGObj, \ldots, 'Weak', WeakValue, \ldots)
\]

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{BGObj}</td>
<td>Biograph object created by \textit{biograph} (object constructor).</td>
</tr>
<tr>
<td>\textit{DirectedValue}</td>
<td>Property that indicates whether the graph is directed or undirected. Enter false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.</td>
</tr>
<tr>
<td>\textit{WeakValue}</td>
<td>Property that indicates whether to find weakly connected components or strongly connected components. A weakly connected component is a maximal group of nodes that are mutually reachable by violating the edge directions. Set \textit{WeakValue} to true to find weakly connected components. Default is false, which finds strongly connected components. The state of this parameter has no effect on undirected graphs because weakly and strongly connected components are the same in undirected graphs. Time complexity is (O(N+E)), where (N) and (E) are number of nodes and edges respectively.</td>
</tr>
</tbody>
</table>

**Description**

**Tip** For introductory information on graph theory functions, see “Graph Theory Functions”.

\[
[S, C] = \text{conncomp}(BGObj)
\]

finds the strongly connected components of an \(N\)-by-\(N\) adjacency matrix extracted from a biograph object, \textit{BGObj} using Tarjan’s algorithm. A strongly connected component is a maximal group of nodes that are mutually reachable without violating the edge directions. The \(N\)-by-\(N\) sparse matrix represents a directed graph; all nonzero entries in the matrix indicate the presence of an edge.

The number of components found is returned in \(S\), and \(C\) is a vector indicating to which component each node belongs.

Tarjan’s algorithm has a time complexity of \(O(N+E)\), where \(N\) and \(E\) are the number of nodes and edges respectively.
[S, C] = conncomp(BGObj, ...'PropertyName', PropertyValue, ...) calls conncomp with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[S, C] = conncomp(BGObj, ...'Directed', DirectedValue, ...) indicates whether the graph is directed or undirected. Set DirectedValue to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true. A DFS-based algorithm computes the connected components. Time complexity is \(O(N+E)\), where \(N\) and \(E\) are number of nodes and edges respectively.

[S, C] = conncomp(BGObj, ...'Weak', WeakValue, ...) indicates whether to find weakly connected components or strongly connected components. A weakly connected component is a maximal group of nodes that are mutually reachable by violating the edge directions. Set WeakValue to true to find weakly connected components. Default is false, which finds strongly connected components. The state of this parameter has no effect on undirected graphs because weakly and strongly connected components are the same in undirected graphs. Time complexity is \(O(N+E)\), where \(N\) and \(E\) are number of nodes and edges respectively.

**Note** By definition, a single node can be a strongly connected component.

**Note** A directed acyclic graph (DAG) cannot have any strongly connected components larger than one.

**References**


**See Also**

allshortestpaths | biograph | graphconncomp | isdag | isomorphism | isspantree | maxflow | minspantree | shortestpath | topoorder | traverse

**Topics**

biograph object on page 1-185

**Introduced in R2006b**
cpgisland

Locate CpG islands in DNA sequence

Syntax

\[
\text{cpgStruct} = \text{cpgisland}(\text{SeqDNA})
\]

\[
\text{cpgStruct} = \text{cpgisland}(\text{SeqDNA}, ...'\text{Window}', \text{WindowValue}, ...)
\]

\[
\text{cpgStruct} = \text{cpgisland}(\text{SeqDNA}, ...'\text{MinIsland}', \text{MinIslandValue}, ...)
\]

\[
\text{cpgStruct} = \text{cpgisland}(\text{SeqDNA}, ...'\text{GCmin}', \text{GCminValue}, ...)
\]

\[
\text{cpgStruct} = \text{cpgisland}(\text{SeqDNA}, ...'\text{CpGoe}', \text{CpGoeValue}, ...)
\]

\[
\text{cpgStruct} = \text{cpgisland}(\text{SeqDNA}, ...'\text{Plot}', \text{PlotValue}, ...)
\]

Input Arguments

\begin{itemize}
\item \textit{SeqDNA} \quad One of the following:
\begin{itemize}
\item Character vector or string specifying a nucleotide sequence
\item Row vector of integers specifying a nucleotide sequence
\item MATLAB structure containing a \texttt{Sequence} field that contains a DNA nucleotide sequence, such as returned by \texttt{fastaread}, \texttt{fastqread}, \texttt{emblread}, \texttt{getembl}, \texttt{genbankread}, or \texttt{getgenbank}
\end{itemize}
\end{itemize}

Valid characters include A, C, G, and T.

\texttt{cpgisland} does not count ambiguous nucleotides or gaps.

\begin{itemize}
\item \textit{WindowValue} \quad Integer specifying the window size for calculating GC content and \texttt{CpGobs}/\texttt{CpGexp} ratios. Default is 100 bases. A smaller window size increases the noise in a plot.
\item \textit{MinIslandValue} \quad Integer specifying the minimum number of consecutive marked bases to report as a CpG island. Default is 200 bases.
\item \textit{GCminValue} \quad Value specifying the minimum GC percent in a window needed to mark a base. Choices are a value between 0 and 1. Default is 0.5.
\item \textit{CpGoeValue} \quad Value specifying the minimum \texttt{CpGobs}/\texttt{CpGexp} ratio in each window needed to mark a base. Choices are a value between 0 and 1. Default is 0.6. This ratio is defined as:

\[
\text{CPGobs/CpGexp} = (\text{NumCpGs*Length})/(\text{NumGs*NumCs})
\]
\item \textit{PlotValue} \quad Controls the plotting of GC content, \texttt{CpGoe} content, CpG islands greater than the minimum island size, and all potential CpG islands for the specified criteria. Choices are \texttt{true} or \texttt{false} (default).
\end{itemize}

Output Arguments

\begin{itemize}
\item \textit{cpgStruct} \quad MATLAB structure containing the starting and ending bases of the CpG islands greater than the minimum island size.
\end{itemize}
Description

cpgStruct = cpgisland(SeqDNA) searches SeqDNA, a DNA nucleotide sequence, for CpG islands with a GC content greater than 50% and a CpGobserved/CpGexpected ratio greater than 60%. It marks bases meeting this criteria within a moving window of 100 DNA bases and then returns the results in cpgStruct, a MATLAB structure containing the starting and ending bases of the CpG islands greater than the minimum island size of 200 bases.

cpgStruct = cpgisland(SeqDNA, ...'PropertyName', PropertyValue, ...) calls cpgisland with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

cpgStruct = cpgisland(SeqDNA, ...'Window', WindowValue, ...) specifies the window size for calculating GC content and CpGobserved/CpGexpected ratios. Default is 100 bases. A smaller window size increases the noise in a plot.

cpgStruct = cpgisland(SeqDNA, ...'MinIsland', MinIslandValue, ...) specifies the minimum number of consecutive marked bases to report as a CpG island. Default is 200 bases.

cpgStruct = cpgisland(SeqDNA, ...'GCmin', GCminValue, ...) specifies the minimum GC percent in a window needed to mark a base. Choices are a value between 0 and 1. Default is 0.5.

cpgStruct = cpgisland(SeqDNA, ...'CpGoe', CpGoeValue, ...) specifies the minimum CpGobserved/CpGexpected ratio in each window needed to mark a base. Choices are a value between 0 and 1. Default is 0.6. This ratio is defined as:

\[ \text{CPGobs/CpGexp} = \frac{\text{NumCpGs} \times \text{Length}}{\text{NumGs} \times \text{NumCs}} \]

cpgStruct = cpgisland(SeqDNA, ...'Plot', PlotValue, ...) controls the plotting of GC content, CpGoe content, CpG islands greater than the minimum island size, and all potential CpG islands for the specified criteria. Choices are true or false (default).

Examples

1. Import a nucleotide sequence from the GenBank database. For example, retrieve a sequence from *Homo sapiens* chromosome 12.

   S = getgenbank('AC156455');

2. Calculate the CpG islands in the sequence and plot the results.

   cpgisland(S.Sequence,'PL0T',true)

   ans =

   Starts: [4510 29359]
   Stops: [5468 29604]

   The CpG islands greater than 200 bases in length are listed and a plot displays.
See Also
basecount | ntdensity | seqshorworf

Introduced before R2006a
crossvalind
Generate indices for training and test sets

Syntax

cvIndices = crossvalind(cvMethod,N,M)
[train,test] = crossvalind(cvMethod,N,M)
___ = crossvalind(___,Name,Value)

Description

cvIndices = crossvalind(cvMethod,N,M) returns the indices cvIndices after applying
cvMethod on N observations using M as the selection parameter.

[train,test] = crossvalind(cvMethod,N,M) returns the logical vectors train and test,
representing observations that belong to the training set and the test (evaluation) set, respectively.
You can specify any supported method except 'Kfold', which accepts a scalar output only.

___ = crossvalind(___,Name,Value) specifies additional options using one or more name-
value pair arguments in addition to the arguments in previous syntaxes. For example, cvIndices =
crossvalind('HoldOut',Groups,0.2,'Class',{'Cancer','Control'}) specifies to use
observations from the 'Cancer' and 'Control' groups to generate indices that represent 20% of
observations as the holdout set and 80% as the training set.

Examples

Perform 10-Fold Cross-Validation

Create indices for the 10-fold cross-validation and classify measurement data for the Fisher iris data
set. The Fisher iris data set contains width and length measurements of petals and sepals from three
species of irises.

Load the data set.

load fisheriris

Create indices for the 10-fold cross-validation.

indices = crossvalind('Kfold',species,10);

Initialize an object to measure the performance of the classifier.

cp = classperf(species);

Perform the classification using the measurement data and report the error rate, which is the ratio of
the number of incorrectly classified samples divided by the total number of classified samples.

for i = 1:10
    test = (indices == i);
    train = ~test;
class = classify(meas(test,:),meas(train,:),species(train,:));
classperf(cp,class,test);
end
cp.ErrorRate
ans = 0.0200

Suppose you want to use the observation data from the setosa and virginica species only and exclude the versicolor species from cross-validation.

labels = {'setosa','virginica'};
indices = crossvalind('Kfold',species,10,'Classes',labels);

indices now contains zeros for the rows that belong to the versicolor species.

Perform the classification again.

for i = 1:10
    test = (indices == i);
    train = ~test;
    class = classify(meas(test,:),meas(train,:),species(train,:));
    classperf(cp,class,test);
end
cp.ErrorRate
ans = 0.0160

Perform Leave-One-Out Cross-Validation

Load the carbig data set.

load carbig;
x = Displacement;
y = Acceleration;
N = length(x);

Train a second degree polynomial model with the leave-one-out cross-validation, and evaluate the averaged cross-validation error. The function randomly selects one observation to hold out for the evaluation set, and using this method within a loop does not guarantee disjointed evaluation sets, and you may see a different CVerr for each run.

sse = 0; % Initialize the sum of squared error.
for i = 1:N
    [train,test] = crossvalind('LeaveMOut',N,1);
    yhat = polyval(polyfit(x(train),y(train),2),x(test));
    sse = sse + sum((yhat - y(test)).^2);
end
CVerr = sse / 100;

Input Arguments

cvMethod — Cross-validation method
character vector | string
Cross-validation method, specified as a character vector or string.

This table describes the valid cross-validation methods. Depending on the method, the third input argument \((M)\) has different meanings and requirements.

<table>
<thead>
<tr>
<th>cvMethod</th>
<th>M</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Kfold'</td>
<td>(M) is the fold parameter, most commonly known as (K) in the (K)-fold cross-validation. (M) must be a positive integer. The default value is 5.</td>
<td>The method uses (K)-fold cross-validation to generate indices. This method uses (M-1) folds for training and the last fold for evaluation. The method repeats this process (M) times, leaving one different fold for evaluation each time.</td>
</tr>
<tr>
<td>'HoldOut'</td>
<td>(M) is the proportion of observations to hold out for the test set. (M) must be a scalar between 0 and 1. The default value is 0.5, corresponding to a 50% holdout.</td>
<td>The method randomly selects approximately (N*M) observations to hold out for the test (evaluation) set. Using this method within a loop is similar to using (K)-fold cross-validation one time outside the loop, except that nondisjointed subsets are assigned to each evaluation.</td>
</tr>
<tr>
<td>'LeaveMOut'</td>
<td>(M) is the number of observations to leave out for the test set. (M) must be a positive integer. The default value is 1, corresponding to the leave-one-out cross-validation (LOOCV).</td>
<td>The method randomly selects (M) observations to hold out for the evaluation set. Using this cross-validation method within a loop does not guarantee disjointed evaluation sets. To guarantee disjointed evaluation sets, use 'Kfold' instead.</td>
</tr>
</tbody>
</table>
| 'Resubstitution' | \(M\) must be specified as a two-element vector \([P, Q]\). Each element must be a scalar between 0 and 1. The default value is \([1, 1]\), corresponding to the full resubstitution. | The method randomly selects \(N\*P\) observations for the evaluation set and \(N\*Q\) observations for the training set. The method selects the sets while minimizing the number of observations used in both sets. \(Q = 1-P\) corresponds to the holdout \((100\*P)\)%.

Example: 'Kfold'

Data Types: char | string

\(N\) — Total number of observations or grouping information
positive integer | vector of positive integers | logical vector | cell array of character vectors

Total number of observations or grouping information, specified as a positive integer, vector of positive integers, logical vector, or cell array of character vectors.
N can be a positive integer specifying the total number of samples in your data set, for instance.

N can also be a vector of positive integers or logical values, or a cell array of character vectors, containing grouping information or labels for your samples. The partition of the groups depends on the type of cross-validation. For 'Kfold', each group is divided into M subsets, approximately equal in size. For all other methods, approximately equal numbers of observations from each group are selected for the evaluation (test) set. The training set contains at least one observation from each group regardless of the cross-validation method you use.

Example: 100
Data Types: double | cell

M — Cross-validation parameter

positive scalar | positive integer | two-element vector

Cross-validation parameter, specified as a positive scalar between 0 and 1, positive integer, or two-element vector. Depending on the cross-validation method, the requirements for M differ. For details, see cvMethod.

Example: 5
Data Types: double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: [train,test] = crossvalind('LeaveMOut',groups,1,'Min',3) specifies to have at least three observations in each group in the training set when performing the leave-one-out cross-validation.

Classes — Class or group information

vector of positive integers | character vector | string | string vector | cell array of character vectors

Class or group information, specified as the comma-separated pair consisting of 'Classes' and a vector of positive integers, character vector, string, string vector, or cell array of character vectors. This option lets you restrict the observations to only the specified groups.

This name-value pair argument is applicable only when you specify N as a grouping variable. The data type of 'Classes' must match that of N. For example, if you specify N as a cell array of character vectors containing class labels, you must use a cell array of character vectors to specify 'Classes'. The output arguments you specify contain the value 0 for observations belonging to excluded classes.

Example: 'Classes',{''Cancer','Control''}
Data Types: double | cell

Min — Minimum number of observations

1 (default) | positive integer

Minimum number of observations for each group in the training set, specified as the comma-separated pair consisting of 'Min' and a positive integer. Setting a large value can help to balance the training groups, but causes partial resubstitution when there are not enough observations.

This name-value pair argument is not applicable for the 'Kfold' method.
Example: 'Min',3
Data Types: double

Output Arguments

cvIndices — Cross-validation indices
vector

Cross-validation indices, returned as a vector.

If you are using 'Kfold' as the cross-validation method, cvIndices contains equal (or approximately equal) proportions of the integers 1 through M, which define a partition of the N observations into M disjointed subsets.

For other cross-validation methods, cvIndices is a logical vector containing 1s for observations that belong to the training set and 0s for observations that belong to the test (evaluation) set.

train — Training set
logical vector

Training set, returned as a logical vector. This argument specifies which observations belong to the training set.

test — Test set
logical vector

Test set, returned as a logical vector. This argument specifies which observations belong to the test set.

See Also
classify | classperf | grp2idx

Introduced before R2006a
cuffcompare

Compare assembled transcripts across multiple experiments

Syntax

\[
\begin{align*}
\text{statsFile} &= \text{cuffcompare} (\text{gtfFiles}) \\
\text{statsFile} &= \text{cuffcompare} (\text{gtfFiles}, \text{compareOptions}) \\
\text{statsFile} &= \text{cuffcompare} (\text{gtfFiles}, \text{Name}, \text{Value}) \\
[\text{statsFile}, \text{combinedGTF}, \text{lociFile}, \text{trackingFile}] &= \text{cuffcompare}( \_\_ _ )
\end{align*}
\]

Description

\(\text{statsFile} = \text{cuffcompare} (\text{gtfFiles})\) compares the assembled transcripts in \text{gtfFiles} and returns summary statistics in the output file \text{statsFile}[1].

cuffcompare requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

**Note** cuffcompare is supported on the Mac and UNIX platforms only.

\(\text{statsFile} = \text{cuffcompare} (\text{gtfFiles}, \text{compareOptions})\) uses additional options specified by \text{compareOptions}.

\(\text{statsFile} = \text{cuffcompare} (\text{gtfFiles}, \text{Name}, \text{Value})\) uses additional options specified by one or more name-value pair arguments. For example, \(\text{statsFile} = \text{cuffcompare} (\text{gtfFile}, '\text{OutputPrefix}', '\text{cuffComp}')\) appends the prefix "cuffComp" to the output file names.

\([\text{statsFile}, \text{combinedGTF}, \text{lociFile}, \text{trackingFile}] = \text{cuffcompare} (\_\_ _ )\) returns the names of the output files using any of the input argument combinations in the previous syntaxes. By default, the function saves all files to the current directory.

Examples

**Assemble Transcriptome and Perform Differential Expression Testing**

Create a CufflinksOptions object to define cufflinks options, such as the number of parallel threads and the output directory to store the results.

\[
\begin{align*}
\text{cflOpt} &= \text{CufflinksOptions}; \\
\text{cflOpt.NumThreads} &= 8; \\
\text{cflOpt.OutputDirectory} &= "./\text{cufflinksOut}";
\end{align*}
\]

The SAM files provided for this example contain aligned reads for *Mycoplasma pneumoniae* from two samples with three replicates each. The reads are simulated 100bp-reads for two genes (gyrA and gyrB) located next to each other on the genome. All the reads are sorted by reference position, as required by cufflinks.
Assemble the transcriptome from the aligned reads.

\[ \text{gtfs, isofpkms, genes, skipped} = \text{cufflinks}(\text{sams, cflOpt}); \]

\( \text{gtfs} \) is a list of GTF files that contain assembled isoforms.

Compare the assembled isoforms using \text{cuffcompare}.

\[ \text{stats} = \text{cuffcompare}(\text{gtfs}); \]

Merge the assembled transcripts using \text{cuffmerge}.

\[ \text{mergedGTF} = \text{cuffmerge}(\text{gtfs, 'OutputDirectory', './cuffMergeOutput'}); \]

\( \text{mergedGTF} \) reports only one transcript. This is because the two genes of interest are located next to each other, and \text{cuffmerge} cannot distinguish two distinct genes. To guide \text{cuffmerge}, use a reference GTF (\text{gyrAB.gtf}) containing information about these two genes. If the file is not located in the same directory that you run \text{cuffmerge} from, you must also specify the file path.

\[ \text{gyrAB = which('gyrAB.gtf');} \]
\[ \text{mergedGTF2 = cuffmerge(gtfs, 'OutputDirectory', './cuffMergeOutput2', ...} \]
\[ \text{'ReferenceGTF', gyrAB}); \]

Calculate abundances (expression levels) from aligned reads for each sample.

\[ \text{abundances1 = cuffquant(mergedGTF2, ['Myco_1_1.sam', 'Myco_1_2.sam', 'Myco_1_3.sam'], ...} \]
\[ \text{'OutputDirectory', './cuffquantOutput1'}); \]
\[ \text{abundances2 = cuffquant(mergedGTF2, ['Myco_2_1.sam', 'Myco_2_2.sam', 'Myco_2_3.sam'], ...} \]
\[ \text{'OutputDirectory', './cuffquantOutput2'}); \]

Assess the significance of changes in expression for genes and transcripts between conditions by performing the differential testing using \text{cuffdiff}. The \text{cuffdiff} function operates in two distinct steps: the function first estimates abundances from aligned reads, and then performs the statistical analysis. In some cases (for example, distributing computing load across multiple workers), performing the two steps separately is desirable. After performing the first step with \text{cuffquant}, you can then use the binary CXB output file as an input to \text{cuffdiff} to perform statistical analysis. Because \text{cuffdiff} returns several files, specify the output directory is recommended.

\[ \text{isoformDiff = cuffdiff(mergedGTF2, [abundances1, abundances2], ...} \]
\[ \text{'OutputDirectory', './cuffdiffOutput'}); \]

Display a table containing the differential expression test results for the two genes \text{gyrB} and \text{gyrA}.

\[ \text{readtable(isoformDiff, 'FileType', 'text')} \]

\[ \text{ans =} \]

\[
\begin{array}{cccccccc}
\text{test_id} & \text{gene_id} & \text{gene} & \text{locus} & \text{sample}_1 & \text{sample}_2 & \text{status} & \text{log2_fold_change} & \text{p_value} & \text{q_value} & \text{significant} \\
\hline
\text{'TCONS_00000001'} & \text{'XLOC_0000001'} & \text{'gyrB'} & \text{'NC_000912.1:2868-7340'} & \text{'q1'} & \text{'q2'} & \text{'OK'} & \text{1.9522} & \text{5e-05} & \text{5e-05} & \text{'yes'} \\
\text{'TCONS_00000002'} & \text{'XLOC_0000001'} & \text{'gyrA'} & \text{'NC_000912.1:2868-7340'} & \text{'q1'} & \text{'q2'} & \text{'OK'} & \text{-1.6064} & \text{5e-05} & \text{5e-05} & \text{'yes'} \\
\end{array}
\]

1-413
You can use `cuffnorm` to generate normalized expression tables for further analyses. `cuffnorm` results are useful when you have many samples and you want to cluster them or plot expression levels for genes that are important in your study. Note that you cannot perform differential expression analysis using `cuffnorm`.

Specify a cell array, where each element is a string vector containing file names for a single sample with replicates.

```matlab
alignmentFiles = {
    {'Myco_1_1.sam','Myco_1_2.sam','Myco_1_3.sam'}, ...
    {'Myco_2_1.sam', 'Myco_2_2.sam', 'Myco_2_3.sam'}
};
```

```matlab
isoformNorm = cuffnorm(mergedGTF2, alignmentFiles, ...
    'OutputDirectory', './cuffnormOutput');
```

Display a table containing the normalized expression levels for each transcript.

```matlab
readtable(isoformNorm,'FileType','text')
```

```
ans =
2x7 table

<table>
<thead>
<tr>
<th>tracking_id</th>
<th>q1_0</th>
<th>q1_2</th>
<th>q1_1</th>
<th>q2_1</th>
<th>q2_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>'TCONS_00000001'</td>
<td>1.0913e+05</td>
<td>78628</td>
<td>1.2132e+05</td>
<td>4.3639e+05</td>
<td>4.2228e+05</td>
</tr>
<tr>
<td>'TCONS_00000002'</td>
<td>3.5158e+05</td>
<td>3.7458e+05</td>
<td>3.4238e+05</td>
<td>1.0483e+05</td>
<td>1.1546e+05</td>
</tr>
</tbody>
</table>
```

Column names starting with `q` have the format: `conditionX_N`, indicating that the column contains values for replicate `N` of `conditionX`.

### Input Arguments

- **gtfFiles** — Names of GTF files
  
  string array | cell array of character vectors
  
  Names of GTF files, specified as a string vector or cell array of character vectors. Each GTF file corresponds to a sample produced by `cufflinks`.
  
  Example: `['Myco_1_1.transcripts.gtf','Myco_2_1.transcripts.gtf']`
  
  Data Types: string | cell

- **compareOptions** — cuffcompare options
  
  CuffCompareOptions object | character vector | string
  
  `cuffcompare` options, specified as a `CuffCompareOptions` object, character vector, or string. The character vector or string must be in the original `cuffcompare` option syntax (prefixed by one or two dashes), such as `-d 100 -e 80` [1].

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `statsFile = cuffcompare(gtfFile,'OutputPrefix','cuffComp','MaxGroupingRange',90)`
**ConsensusPrefix — Prefix for consensus transcript names**
"TCONS" (default) | string | character vector

Prefix for consensus transcript names in the output combined.gtf file, specified as a string or character vector. This option must be a string or character vector with a non-zero length.

Example: 'ConsensusPrefix','consensusTs'

Data Types: char | string

**DiscardIntronRedundant — Flag to ignore intron-redundant transfrags**
false (default) | true

Flag to ignore intron-redundant transfrags if they have the same 5' end but different 3' ends, specified as true or false.

Example: 'DiscardIntronRedundant',true

Data Types: logical

**DiscardSingleExonAll — Flag to discard single-exon transfrags and reference transcripts**
false (default) | true

Flag to discard single-exon transfrags and reference transcripts, specified as true or false.

Example: 'DiscardSingleExonAll',true

Data Types: logical

**DiscardSingleExonReference — Flag to discard single-exon reference transcripts**
false (default) | true

Flag to discard single-exon reference transcripts, specified as true or false.

Example: 'DiscardSingleExonReference',true

Data Types: logical

**ExtraCommand — Additional commands**
"" (default) | character vector | string

Additional commands, specified as a string or character vector. The commands must be in the original syntax (prefixed by one or two dashes). Use this option to apply undocumented flags and flags without corresponding MATLAB properties. When the function converts the original flags to MATLAB properties, it stores any unrecognized flags in this option.

Example: 'ExtraCommand','--library-type fr-secondstrand'

Data Types: char | string

**GTFManifest — Name of text file containing list of GTF files to process**
string | character vector

Name of the text file containing a list of GTF files to process, specified as a string or character vector. The file must contain one GTF file path per line. You can use this option as an alternative to passing an array of file names to cuffcompare.

Example: 'GTFManifest','gtfManifestFile.txt'

Data Types: char | string
**GenericGFF — Flag to treat input GTF files as GFF**

false (default) | true

Flag to treat input GTF files as GFF files, specified as `true` or `false`. Use this option when the input GFF or GTF files are not produced by `cufflinks`.

Example: `'GenericGFF',true`

Data Types: logical

**IncludeAll — Flag to include all available options**

false (default) | true

Flag to include all available options with the corresponding default values when converting to the original options syntax, specified as `true` or `false`. The original syntax is prefixed by one or two dashes, such as `-d 100 -e 80`. By default, the function converts only the specified options. If the value is `true`, the function converts all available options, with default values for unspecified options, to the original syntax.

Example: `'IncludeAll',true`

Data Types: logical

**IncludeContained — Flag to include transfrags contained by other transfrags**

false (default) | true

Flag to include transfrags contained by other transfrags in the same locus in the output combined.gtf, specified as `true` or `false`. By default, `cuffcompare` does not include these contained transfrags. If the value is `true`, the contained transfrags include a `contained_in` attribute indicating the first container transfrag found.

Example: `'IncludeContained',true`

Data Types: logical

**MaxAccuracyRange — Number of bases from terminal exons to use when assessing exon accuracy**

100 (default) | positive integer

Number of bases from the free ends of terminal exons to use when assessing exon accuracy, specified as a positive integer.

Example: `'MaxAccuracyRange',80`

Data Types: double

**MaxGroupingRange — Number of bases to use for grouping transcript start sites**

100 (default) | positive integer

Number of bases to use for grouping transcript start sites, specified as a positive integer.

Example: `'MaxGroupingRange',90`

Data Types: double

**OutputPrefix — Prefix for cuffcompare output files**

"cuffcmp" (default) | string | character vector

Prefix for `cuffcompare` output files, specified as a string or character vector. This option must be a string or character vector with a non-zero length.
Example: 'OutputPrefix', "cuffcompareOut"
Data Types: char | string

**ReferenceGTF — Name of GTF or GFF file containing reference transcripts**

string | character vector

Name of the GTF or GFF file containing reference transcripts to compare to each sample, specified as a string or character vector. If you provide a file, the function compares each sample to the references in the file and marks isoforms as overlapping, matching, or novel. The function stores these tags in the output files .refmap and .tmap files.

Example: 'ReferenceGTF', "references.gtf"
Data Types: char | string

**SequenceDirectory — Name of directory containing FASTA sequences to classify input transcripts as repeats**

string | character vector

Name of directory containing FASTA sequences to classify input transcripts as repeats, specified as a string or character vector. The directory must contain FASTA-format files with the underlying genomic sequences and contain one FASTA file per reference. Name each FASTA file after the chromosome with the extension .fa or .fasta.

Example: 'SequenceDirectory', "/SequenceDirectory/"
Data Types: char | string

**SnCorrection — Flag to consider only reference transcripts that overlap with input transfrags**

false (default) | true

Flag to consider only reference transcripts that overlap with any of the input transfrags, specified as true or false. If the value is true:

- The function ignores any reference transcripts that do not overlap with any of the input transfrags.
- You must also specify the ReferenceGTF option.

Example: 'SnCorrection', true
Data Types: logical

**SpCorrection — Flag to consider only input transcripts that overlap with reference transcripts**

false (default) | true

Flag to consider only input transcripts that overlap with any of the reference transcripts, specified as true or false. If the value is true:

- The function ignores any input transcripts that do not overlap with any of the reference transcripts and reports no novel loci.
- You must also specify the ReferenceGTF option.

Example: 'SpCorrection', true
Data Types: logical
SuppressMapFiles — Flag to prevent creation of .tmap and .refmap files
false (default) | true

Flag to prevent the creation of .tmap and .refmap files, specified as true or false. Set the value to true to prevent the function from generating the files.

Example: 'SuppressMapFiles', true

Data Types: logical

Output Arguments

statsFile — Name of text file containing statistics
"cuffcmp.stats"

Name of the text file containing statistics related to the accuracy of the transcripts in each sample, returned as a string. The function performs the tests for sensitivity (Sn) and specificity (Sp) at various levels, including the nucleotide, exon, and intron levels, and reports the results in this file.

The default file name is "cuffcmp.stats". If you specify OutputPrefix, the function uses it instead of "cuffcmp".

combinedGTF — Name of file containing union of all transfrags in each sample
"cuffcmp.combined.gtf"

Name of the file containing the union of all transfrags in each sample, returned as a string.

The default file name is "cuffcmp.combined.gtf". If you specify OutputPrefix, the function uses it instead of "cuffcmp".

lociFile — Name of file with all processed loci
"cuffcmp.loci"

Name of file with all processed loci across all transcripts, returned as a string.

The default file name is "cuffcmp.loci". If you specify OutputPrefix, the function uses it instead of "cuffcmp".

trackingFile — Name of file containing transcripts with identical coordinates
"cuffcmp.tracking"

Name of the file containing transcripts with identical coordinates, introns, and strands, returned as a string.

The default file name is "cuffcmp.tracking". If you specify OutputPrefix, the function uses it instead of "cuffcmp".

References

See Also
CuffCompareOptions | cufflinks

External Websites
Cufflinks manual

Introduced in R2019a
CuffCompareOptions

Option set for cuffcompare

Description

A CuffCompareOptions object specifies options for the cuffcompare function, which compares assembled transcripts across several experiments [1].

Creation

Syntax

cuffcompareOpt = CuffCompareOptions
cuffcompareOpt = CuffCompareOptions(Name,Value)
cuffcompareOpt = CuffCompareOptions(S)

Description

cuffcompareOpt = CuffCompareOptions creates a CuffCompareOptions object with the default property values.

CuffCompareOptions requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

Note CuffCompareOptions is supported on the Mac and UNIX platforms only.

cuffcompareOpt = CuffCompareOptions(Name,Value) sets the object properties on page 1-421 using one or more name-value pair arguments. Enclose each property name in quotes. For example, cuffcompareOpt = CuffCompareOptions('SupressMapFiles',true) prevents the creation of .tmap and .refmap files.

cuffcompareOpt = CuffCompareOptions(S) specifies optional parameters using the string or character vector S.

Input Arguments

S — cuffcompare options
string | character vector

cuffcompare options, specified as a string or character vector. S must be in the original cuffcompare option syntax (prefixed by one or two dashes).

Example: ‘-d 100 -e 80’
### Properties

**ConsensusPrefix — Prefix for consensus transcript names**

"TCONS" (default) | string | character vector

Prefix for consensus transcript names in the output combined.gtf file, specified as a string or character vector. This option must be a string or character vector with a non-zero length.

Example: "consensusTs"

Data Types: char | string

**DiscardIntronRedundant — Flag to ignore intron-redundant transfrags**

false (default) | true

Flag to ignore intron-redundant transfrags if they have the same 5' end but different 3' ends, specified as true or false.

Example: true

Data Types: logical

**DiscardSingleExonAll — Flag to discard single-exon transfrags and reference transcripts**

false (default) | true

Flag to discard single-exon transfrags and reference transcripts, specified as true or false.

Example: true

Data Types: logical

**DiscardSingleExonReference — Flag to discard single-exon reference transcripts**

false (default) | true

Flag to discard single-exon reference transcripts, specified as true or false.

Example: true

Data Types: logical

**ExtraCommand — Additional commands**

"" (default) | string | character vector

Additional commands, specified as a string or character vector. The commands must be in the original syntax (prefixed by one or two dashes). Use this option to apply undocumented flags and flags without corresponding MATLAB properties. When the function converts the original flags to MATLAB properties, it stores any unrecognized flags in this option.

Example: "--library-type fr-secondstrand"

Data Types: char | string

**GTFManifest — Name of text file containing list of GTF files to process**

string | character vector

Name of the text file containing a list of GTF files to process, specified as a string or character vector. The file must contain one GTF file path per line. You can use this option as an alternative to passing an array of file names to cuffcompare.

Example: "gtfManifestFile.txt"
Data Types: char | string

**GenericGFF** — Flag to treat input GTF files as GFF
false (default) | true

Flag to treat input GTF files as GFF files, specified as true or false. Use this option when the input GFF or GTF files are not produced by cufflinks.

Example: true

Data Types: logical

**IncludeAll** — Flag to include all object properties when converting to original syntax
false (default) | true

Flag to include all the object properties with the corresponding default values when converting to the original options syntax, specified as true or false. You can convert the properties to the original syntax prefixed by one or two dashes (such as '-d 100 -e 80') by using getCommand. The default value false means that when you call getCommand(optionsObject), it converts only the specified properties. If the value is true, getCommand converts all available properties, with default values for unspecified properties, to the original syntax.

Example: true

Data Types: logical

**IncludeContained** — Flag to include transfrags contained by other transfrags
false (default) | true

Flag to include transfrags contained by other transfrags in the same locus in the output combined.gtf, specified as true or false. By default, cuffcompare does not include these contained transfrags. If the value is true, the contained transfrags include a contained_in attribute indicating the first container transfrag found.

Example: true

Data Types: logical

**MaxAccuracyRange** — Number of bases from terminal exons to use when assessing exon accuracy
100 (default) | positive integer

Number of bases from the free ends of terminal exons to use when assessing exon accuracy, specified as a positive integer.

Example: 80

Data Types: double

**MaxGroupingRange** — Number of bases to use for grouping transcript start sites
100 (default) | positive integer

Number of bases to use for grouping transcript start sites, specified as a positive integer.

Example: 90

Data Types: double

**OutputPrefix** — Prefix for cuffcompare output files
"cuffcmp" (default) | string | character vector
Prefix for `cuffcompare` output files, specified as a string or character vector. This option must be a string or character vector with a non-zero length.

Example: "cuffcompareOut"

Data Types: `char` | `string`

**ReferenceGTF — Name of GTF or GFF file containing reference transcripts**

Name of the GTF or GFF file containing reference transcripts to compare to each sample, specified as a string or character vector. If you provide a file, the function compares each sample to the references in the file and marks isoforms as overlapping, matching, or novel. The function stores these tags in the output files `.refmap` and `.tmap` files.

Example: "references.gtf"

Data Types: `char` | `string`

**SequenceDirectory — Name of directory containing FASTA sequences to classify input transcripts as repeats**

Name of directory containing FASTA sequences to classify input transcripts as repeats, specified as a string or character vector. The directory must contain FASTA-format files with the underlying genomic sequences and contain one FASTA file per reference. Name each FASTA file after the chromosome with the extension `.fa` or `.fasta`.

Example: "./SequenceDirectory/

Data Types: `char` | `string`

**SnCorrection — Flag to consider only reference transcripts that overlap with input transfrags**

Flag to consider only reference transcripts that overlap with any of the input transfrags, specified as `true` or `false`. If the value is `true`:

- The function ignores any reference transcripts that do not overlap with any of the input transfrags.
- You must also specify the ReferenceGTF option.

Example: `true`

Data Types: `logical`

**SpCorrection — Flag to consider only input transcripts that overlap with reference transcripts**

Flag to consider only input transcripts that overlap with any of the reference transcripts, specified as `true` or `false`. If the value is `true`:

- The function ignores any input transcripts that do not overlap with any of the reference transcripts and reports no novel loci.
- You must also specify the ReferenceGTF option.
Example: true
Data Types: logical

**SuppressMapFiles — Flag to prevent creation of .tmap and .refmap files**
false (default) | true

Flag to prevent the creation of .tmap and .refmap files, specified as true or false. Set the value to true to prevent the function from generating the files.
Example: true
Data Types: logical

**Version — Supported version**
string

This property is read-only.

Supported version of the original cufflinks software, returned as a string.
Example: "2.2.1"
Data Types: string

**Object Functions**
getCommand     Translate object properties to original options syntax
getOptionsTable Return table with all properties and equivalent options in original syntax

**Examples**

**Create CuffCompareOptions Object**

Create a CuffCompareOptions object with the default values.

```
opt = CuffCompareOptions;
```

Create an object using name-value pairs.

```
opt2 = CuffCompareOptions('GenericGFF',true,'MaxAccuracyRange',80)
```

Create an object using the original syntax.

```
opt3 = CuffCompareOptions('-d 100 -e 80')
```

**Assemble Transcriptome and Perform Differential Expression Testing**

Create a CufflinksOptions object to define cufflinks options, such as the number of parallel threads and the output directory to store the results.

```
cflOpt = CufflinksOptions;
cflOpt.NumThreads = 8;
cflOpt.OutputDirectory = "./cufflinksOut";
```
The SAM files provided for this example contain aligned reads for *Mycoplasma pneumoniae* from two samples with three replicates each. The reads are simulated 100bp-reads for two genes (gyrA and gyrB) located next to each other on the genome. All the reads are sorted by reference position, as required by cufflinks.

```matlab
sams = ['Myco_1_1.sam', 'Myco_1_2.sam', 'Myco_1_3.sam', ...
        'Myco_2_1.sam', 'Myco_2_2.sam', 'Myco_2_3.sam'];
```

Assemble the transcriptome from the aligned reads.

```matlab
[gtfs, isofpkm, genes, skipped] = cufflinks(sams, cflOpt);
gtfs is a list of GTF files that contain assembled isoforms.

Compare the assembled isoforms using cuffcompare.

```matlab
stats = cuffcompare(gtfs);
```

Merge the assembled transcripts using cuffmerge.

```matlab
mergedGTF = cuffmerge(gtfs, 'OutputDirectory', './cuffMergeOutput');
```

`mergedGTF` reports only one transcript. This is because the two genes of interest are located next to each other, and cuffmerge cannot distinguish two distinct genes. To guide cuffmerge, use a reference GTF (gyrAB.gtf) containing information about these two genes. If the file is not located in the same directory that you run cuffmerge from, you must also specify the file path.

```matlab
gyrAB = which('gyrAB.gtf');
mergedGTF2 = cuffmerge(gtfs, 'OutputDirectory', './cuffMergeOutput2', ...
                       'ReferenceGTF', gyrAB);
```

Calculate abundances (expression levels) from aligned reads for each sample.

```matlab
abundances1 = cuffquant(mergedGTF2, ['Myco_1_1.sam', 'Myco_1_2.sam', 'Myco_1_3.sam'], ...
                         'OutputDirectory', './cuffquantOutput1');
abundances2 = cuffquant(mergedGTF2, ['Myco_2_1.sam', 'Myco_2_2.sam', 'Myco_2_3.sam'], ...
                         'OutputDirectory', './cuffquantOutput2');
```

Assess the significance of changes in expression for genes and transcripts between conditions by performing the differential testing using cuffdiff. The cuffdiff function operates in two distinct steps: the function first estimates abundances from aligned reads, and then performs the statistical analysis. In some cases (for example, distributing computing load across multiple workers), performing the two steps separately is desirable. After performing the first step with cuffquant, you can then use the binary CXB output file as an input to cuffdiff to perform statistical analysis. Because cuffdiff returns several files, specify the output directory is recommended.

```matlab
isoformDiff = cuffdiff(mergedGTF2, [abundances1, abundances2], ...
                       'OutputDirectory', './cuffdiffOutput');
```

Display a table containing the differential expression test results for the two genes gyrB and gyrA.

```matlab
readtable(isoformDiff, 'FileType', 'text')
ans =
  2×14 table
    test_id            gene_id        gene     locus            sample_1    sample_2    status     value_1       value_2       log2_fold_change_    test_stat    p_value    q_value    significant
```
You can use `cuffnorm` to generate normalized expression tables for further analyses. `cuffnorm` results are useful when you have many samples and you want to cluster them or plot expression levels for genes that are important in your study. Note that you cannot perform differential expression analysis using `cuffnorm`.

Specify a cell array, where each element is a string vector containing file names for a single sample with replicates.

```
alignmentFiles = {{"Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam"},
                 {{"Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"}}};
```

Display a table containing the normalized expression levels for each transcript.

```
readtable(isoformNorm,'FileType','text')
```

<table>
<thead>
<tr>
<th>tracking_id</th>
<th>q1_0</th>
<th>q1_2</th>
<th>q1_1</th>
<th>q2_1</th>
<th>q2_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>'TCONS_00000001'</td>
<td>1.0913e+05</td>
<td>78628</td>
<td>1.2132e+05</td>
<td>4.3639e+05</td>
<td>4.2228e+05</td>
</tr>
<tr>
<td>'TCONS_00000002'</td>
<td>3.5158e+05</td>
<td>3.7458e+05</td>
<td>3.4238e+05</td>
<td>1.0483e+05</td>
<td>1.1546e+05</td>
</tr>
</tbody>
</table>

Column names starting with q have the format: `conditionX_N`, indicating that the column contains values for replicate N of conditionX.

**References**


**See Also**

cuffcompare | cufflinks

**External Websites**

Cufflinks manual

**Introduced in R2019a**
cuffdiff

Identify significant changes in transcript expression

Syntax

cuffdiff(transcriptsAnnot, alignmentFiles)
cuffdiff(transcriptsAnnot, alignmentFiles, opt)
cuffdiff(transcriptsAnnot, alignmentFiles, Name, Value)
[isoformsDiff, geneDiff, tssDiff, cdsExp, splicingDiff, cdsDiff, promotersDiff] =
cuffdiff( ___ )

Description

cuffdiff(transcriptsAnnot, alignmentFiles) identifies significant changes in transcript expression between the samples in alignmentFiles using the transcript annotation file transcriptsAnnot.

cuffdiff requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

Note cuffdiff is supported on the Mac and UNIX platforms only.

cuffdiff(transcriptsAnnot, alignmentFiles, opt) uses additional options specified by opt.

cuffdiff(transcriptsAnnot, alignmentFiles, Name, Value) uses additional options specified by one or more name-value pair arguments. For example, cuffdiff('gyrAB.gtf', ['Myco_1_1.sam', 'Myco_2_1.sam'], 'NumThreads', 5) specifies five parallel threads.

[isoformsDiff, geneDiff, tssDiff, cdsExp, splicingDiff, cdsDiff, promotersDiff] =
cuffdiff( ___ ) returns the names of files containing differential expression test results using any of the input argument combinations in the previous syntaxes. By default, the function saves all files to the current directory.

Examples

Assemble Transcriptome and Perform Differential Expression Testing

Create a CufflinksOptions object to define cufflinks options, such as the number of parallel threads and the output directory to store the results.

cflOpt = CufflinksOptions;
cflOpt.NumThreads = 8;
cflOpt.OutputDirectory = './cufflinksOut';

The SAM files provided for this example contain aligned reads for *Mycoplasma pneumoniae* from two samples with three replicates each. The reads are simulated 100bp-reads for two genes (gyrA and gyrB) located next to each other on the genome. All the reads are sorted by reference position, as required by cufflinks.
sams = ["Myco_1_1.sam", "Myco_1_2.sam", "Myco_1_3.sam", ...
        "Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"];

Assemble the transcriptome from the aligned reads.
[gtfs, isofpkm, genes, skipped] = cufflinks(sams, cflOpt);
gtfs is a list of GTF files that contain assembled isoforms.

Compare the assembled isoforms using cuffcompare.
stats = cuffcompare(gtfs);

Merge the assembled transcripts using cuffmerge.
mergedGTF = cuffmerge(gtfs, 'OutputDirectory', './cuffMergeOutput');

mergedGTF reports only one transcript. This is because the two genes of interest are located next to each other, and cuffmerge cannot distinguish two distinct genes. To guide cuffmerge, use a reference GTF (gyrAB.gtf) containing information about these two genes. If the file is not located in the same directory that you run cuffmerge from, you must also specify the file path.

gyrAB = which('gyrAB.gtf');
mergedGTF2 = cuffmerge(gtfs, 'OutputDirectory', './cuffMergeOutput2', ...
                       'ReferenceGTF', gyrAB);

Calculate abundances (expression levels) from aligned reads for each sample.

abundances1 = cuffquant(mergedGTF2, ["Myco_1_1.sam", "Myco_1_2.sam", "Myco_1_3.sam"], ...
                         'OutputDirectory', './cuffquantOutput1');
abundances2 = cuffquant(mergedGTF2, ["Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"], ...
                         'OutputDirectory', './cuffquantOutput2');

Assess the significance of changes in expression for genes and transcripts between conditions by performing the differential testing using cuffdiff. The cuffdiff function operates in two distinct steps: the function first estimates abundances from aligned reads, and then performs the statistical analysis. In some cases (for example, distributing computing load across multiple workers), performing the two steps separately is desirable. After performing the first step with cuffquant, you can then use the binary CXB output file as an input to cuffdiff to perform statistical analysis. Because cuffdiff returns several files, specify the output directory is recommended.

isoformDiff = cuffdiff(mergedGTF2, [abundances1, abundances2], ...
                       'OutputDirectory', './cuffdiffOutput');

Display a table containing the differential expression test results for the two genes gyrB and gyrA.
readtable(isoformDiff, 'FileType', 'text')

ans =

2×14 table

<table>
<thead>
<tr>
<th>test_id</th>
<th>gene_id</th>
<th>gene</th>
<th>locus</th>
<th>sample_1</th>
<th>sample_2</th>
<th>status</th>
<th>value_1</th>
<th>value_2</th>
<th>log2_fold_change_</th>
<th>test_stat</th>
<th>p_value</th>
<th>q_value</th>
<th>significant</th>
</tr>
</thead>
<tbody>
<tr>
<td>'TCONS_00000001'</td>
<td>'XLOC_000001'</td>
<td>'gyrB'</td>
<td>'NC_000912.1:2868-7340'</td>
<td>'q1'</td>
<td>'q2'</td>
<td>'OK'</td>
<td>1.0913e+05</td>
<td>4.2228e+05</td>
<td>1.9522</td>
<td>7.8886</td>
<td>5e-05</td>
<td>5e-05</td>
<td>'yes'</td>
</tr>
<tr>
<td>'TCONS_00000002'</td>
<td>'XLOC_000001'</td>
<td>'gyrA'</td>
<td>'NC_000912.1:2868-7340'</td>
<td>'q1'</td>
<td>'q2'</td>
<td>'OK'</td>
<td>3.5158e+05</td>
<td>1.1546e+05</td>
<td>-1.6064</td>
<td>-7.3811</td>
<td>5e-05</td>
<td>5e-05</td>
<td>'yes'</td>
</tr>
</tbody>
</table>
You can use `cuffnorm` to generate normalized expression tables for further analyses. `cuffnorm` results are useful when you have many samples and you want to cluster them or plot expression levels for genes that are important in your study. Note that you cannot perform differential expression analysis using `cuffnorm`.

Specify a cell array, where each element is a string vector containing file names for a single sample with replicates.

```plaintext
alignmentFiles = {"Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam"},...
                ["Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"]
isoformNorm = cuffnorm(mergedGTF2, alignmentFiles,...
                      'OutputDirectory', './cuffnormOutput');
```

Display a table containing the normalized expression levels for each transcript.

```plaintext
readtable(isoformNorm,'FileType','text')
an = 2×7 table
      tracking_id        q1_0        q1_2        q1_1        q2_1        q2_0
    ______________    __________    __________    __________    __________    __________
'TCONS_00000001'     1.0913e+05    78628    1.2132e+05    4.3639e+05    4.2228e+05    4.2814e+05
'TCONS_00000002'     3.5158e+05    3.7458e+05    3.4238e+05    1.0483e+05    1.1546e+05    1.1105e+05
```

Column names starting with `q` have the format: `conditionX_N`, indicating that the column contains values for replicate `N` of `conditionX`.

**Input Arguments**

- **transcriptsAnnot** — Name of transcript annotation file
  
  string | character vector

  Name of the transcript annotation file, specified as a string or character vector. The file can be a GTF or GFF file produced by `cufflinks`, `cuffcompare`, or another source of GTF annotations.

  Example: "gyrAB.gtf"

  Data Types: char | string

- **alignmentFiles** — Names of SAM, BAM, or CXB files
  
  string vector | cell array

  Names of SAM, BAM, or CXB files containing alignment records for each sample, specified as a string vector or cell array. If you use a cell array, each element must be a string vector or cell array of character vectors specifying alignment files for every replicate of the same sample.

  Example: ["Myco_1_1.sam", "Myco_2_1.sam"]

  Data Types: char | string | cell

- **opt** — cuffdiff options
  
  CuffDiffOptions object | string | character vector

  `cuffdiff` options, specified as a `CuffDiffOptions` object, string, or character vector. The string or character vector must be in the original `cuffdiff` option syntax (prefixed by one or two dashes) [1].
Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1, Value1,..., NameN, ValueN.

Example: `cuffdiff('gyrAB.gtf', ['Myco_1_1.sam', 'Myco_2_1.sam'], 'NumThreads', 5, 'DispersionMethod', 'per-condition')`

**ConditionLabels — Sample labels**

string | string vector | character vector | cell array of character vectors

Sample labels, specified as a string, string vector, character vector, or cell array of character vectors. The number of labels must equal the number of samples or the value must be empty `[]`.

Example: `'ConditionLabels', ['Control', 'Mutant1', 'Mutant2']`
Data Types: string | char | cell

**ContrastFile — Contrast file name**

string | character vector

Contrast file name, specified as a string or character vector. The file must be a two-column tab-delimited text file, where each line indicates two conditions to compare using `cuffdiff`. The condition labels in the file must match either the labels specified for `ConditionLabels` or the sample names. The file must have a single header line as the first line, followed by one line for each contrast. An example of the contrast file format follows.

<table>
<thead>
<tr>
<th>condition_A</th>
<th>condition_B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>Mutant1</td>
</tr>
<tr>
<td>Control</td>
<td>Mutant2</td>
</tr>
</tbody>
</table>

If you do not provide this file, `cuffdiff` compares every pair of input conditions, which can impact performance.

Example: `'ContrastFile', 'contrast.txt'`
Data Types: char | string

**DispersionMethod — Method to model variance in fragment counts**

"pooled" (default) | "per-condition" | "blind" | "poisson"

Method to model the variance in fragment counts across replicates, specified as one of the following options:

- "pooled" — The function uses each replicated condition to build a model and averages these models into a global model for all conditions in the experiment.
- "per-condition" — The function produces a model for each condition. You can use this option only if all conditions have replicates.
- "blind" — The function treats all samples as replicates of a single global distribution and produces one model.
- "poisson" — Variance in fragment counts is a poisson model, where the fragment count is predicted to be the mean across replicates. This method is not recommended.
Select a method depending on whether you expect the variability in each group of samples to be similar.

- When comparing two groups where the first group has low cross-replicate variability and the second group has high variability, choose the per-condition method.
- If the conditions have similar levels of variability, choose the pooled method.
- If you have only a single replicate in each condition, choose the blind method.

Example: `'DispersionMethod','blind'`

Data Types: char | string

DoIsoformSwitch — Flag to perform isoform switching tests

`true (default) | false`

Flag to perform isoform switching tests, specified as `true` or `false`. These tests estimate how much differential splicing exists in isoforms from a single primary transcript. By default, the value is `true` and the test results are saved in the output file `splicing.diff`.

Example: `'DoIsoformSwitch',false`

Data Types: logical

EffectiveLengthCorrection — Flag to normalize fragment counts

`true (default) | false`

Flag to normalize fragment counts to fragments per kilobase per million mapped reads (FPKM), specified as `true` or `false`.

Example: `'EffectiveLengthCorrection',false`

Data Types: logical

ExtraCommand — Additional commands

`````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````
experiments. Providing reference transcripts improves the accuracy of the transcript abundance estimates.

Example: 'FragmentBiasCorrection','bias.fasta'

Data Types: char | string

**FragmentLengthMean — Expected mean fragment length in base pairs**

200 (default) | positive integer

Expected mean fragment length, specified as a positive integer. The default value is 200 base pairs. The function can learn the fragment length mean for each SAM file. Using this option is not recommended for paired-end reads.

Example: 'FragmentLengthMean',100

Data Types: double

**FragmentLengthSD — Expected standard deviation for fragment length distribution**

80 (default) | positive scalar

Expected standard deviation for the fragment length distribution, specified as a positive scalar. The default value is 80 base pairs. The function can learn the fragment length standard deviation for each SAM file. Using this option is not recommended for paired-end reads.

Example: 'FragmentLengthSD',70

Data Types: double

**GenerateAnalysisDiff — Flag to create differential analysis files**

true (default) | false

Flag to create differential analysis files (*.diff), specified as true or false.

Example: 'GenerateAnalysisDiff',false

Data Types: logical

**IncludeAll — Flag to include all object properties**

false (default) | true

Flag to include all the object properties with the corresponding default values when converting to the original options syntax, specified as true or false. You can convert the properties to the original syntax prefixed by one or two dashes (such as ' -d 100 -e 80 ') by using getCommand. The default value false means that when you call getCommand(optionsObject), it converts only the specified properties. If the value is true, getCommand converts all available properties, with default values for unspecified properties, to the original syntax.

Example: 'IncludeAll',true

Data Types: logical

**IsoformShiftReplicates — Minimum number of replicates to test genes for differential regulation**

3 (default) | positive integer

Minimum number of replicates to test genes for differential regulation, specified as a positive integer. The function skips the tests when the number of replicates is smaller than the specified value.

Example: 'IsoformShiftReplicates',2
Data Types: double

**LengthCorrection — Flag to correct by transcript length**
true (default) | false

Flag to correct by the transcript length, specified as true or false. Set this value to false only when the fragment count is independent of the feature size, such as for small RNA libraries with no fragmentation and for 3' end sequencing, where all fragments have the same length.

Example: 'LengthCorrection',false

Data Types: logical

**LibraryNormalizationMethod — Method to normalize library size**
"geometric" (default) | "classic-fpkm" | "quartile"

Method to normalize the library size, specified as one of the following options:

- "geometric" — The function scales the FPKM values by the median geometric mean of fragment counts across all libraries as described in [2].
- "classic-fpkm" — The function applies no scaling to the FPKM values or fragment counts.
- "quartile" — The function scales the FPKM values by the ratio of upper quartiles between fragment counts and the average value across all libraries.

Example: 'LibraryNormalizationMethod','classic-fpkm'

Data Types: char | string

**MaskFile — Name of GTF or GFF file containing transcripts to ignore**
string | character vector

Name of the GTF or GFF file containing transcripts to ignore during analysis, specified as a string or character vector. Some examples of transcripts to ignore include annotated rRNA transcripts, mitochondrial transcripts, and other abundant transcripts. Ignoring these transcripts improves the robustness of the abundance estimates.

Example: 'MaskFile','excludes.gtf'

Data Types: char | string

**MaxBundleFrags — Maximum number of fragments to include for each locus before skipping**
500000 (default) | positive integer

Maximum number of fragments to include for each locus before skipping new fragments, specified as a positive integer. Skipped fragments are marked with the status HIDATA in the file skipped.gtf.

Example: 'MaxBundleFrags',400000

Data Types: double

**MaxFragAlignments — Maximum number of aligned reads to include for each fragment**
Inf (default) | positive integer

Maximum number of aligned reads to include for each fragment before skipping new reads, specified as a positive integer. Inf, the default value, sets no limit on the maximum number of aligned reads.

Example: 'MaxFragAlignments',1000
Data Types: double

MaxMLEIterations — Maximum number of iterations for maximum likelihood estimation
5000 (default) | positive integer

Maximum number of iterations for the maximum likelihood estimation of abundances, specified as a positive integer.
Example: 'MaxMLEIterations',4000
Data Types: double

MinAlignmentCount — Minimum number of alignments required in locus for significance testing
10 (default) | positive integer

Minimum number of alignments required in a locus to perform the significance testing for differences between samples, specified as a positive integer.
Example: 'MinAlignmentCount',8
Data Types: double

MinIsoformFraction — Minimum abundance of isoform to include in differential expression tests
1e-5 (default) | scalar between 0 and 1

Minimum abundance of an isoform to include in differential expression tests, specified as a scalar between 0 and 1. For alternative isoforms quantified at below the specified value, the function rounds down the abundance to zero. The specified value is a fraction of the major isoform. The function performs this filtering after MLE estimation but before MAP estimation to improve the robustness of confidence interval generation and differential expression analysis. Using a parameter value other than the default is not recommended.
Example: 'MinIsoformFraction',1e-5
Data Types: double

MultiReadCorrection — Flag to improve abundance estimation using rescue method
false (default) | true

Flag to improve abundance estimation for reads mapped to multiple genomic positions using the rescue method, specified as true or false. If the value is false, the function divides multimapped reads uniformly to all mapped positions. If the value is true, the function uses additional information, including gene abundance estimation, inferred fragment length, and fragment bias, to improve transcript abundance estimation.

The rescue method is described in [3].
Example: 'MultiReadCorrection',true
Data Types: logical

NormalizeCompatibleHits — Flag to use only fragments compatible with reference transcript to calculate FPKM values
true (default) | false

Flag to use only fragments compatible with a reference transcript to calculate FPKM values, specified as true or false.
Example: `NormalizeCompatibleHits',false
Data Types: logical

**NormalizeTotalHits — Flag to include all fragments to calculate FPKM values**
false (default) | true

Flag to include all fragments to calculate FPKM values, specified as true or false. If the value is true, the function includes all fragments, including fragments without a compatible reference.
Example: `NormalizeTotalHits',true
Data Types: logical

**NumFragAssignmentDraws — Number of fragment assignments to perform on each transcript**
50 (default) | positive integer

Number of fragment assignments to perform on each transcript, specified as a positive integer. For each fragment drawn from a transcript, the function performs the specified number of assignments probabilistically to determine the transcript assignment uncertainty and to estimate the variance-covariance matrix for the assigned fragment counts.
Example: 'NumFragAssignmentSamples',40
Data Types: double

**NumFragDraws — Number of draws from negative binomial random number generator**
100 (default) | positive integer

Number of draws from the negative binomial random number generator for each transcript, specified as a positive integer. Each draw is a number of fragments that the function probabilistically assigns to transcripts in the transcriptome to determine the assignment uncertainty and to estimate the variance-covariance matrix for assigned fragment counts.
Example: 'NumFragSamples',90
Data Types: double

**NumThreads — Number of parallel threads to use**
1 (default) | positive integer

Number of parallel threads to use, specified as a positive integer. Threads are run on separate processors or cores. Increasing the number of threads generally improves the runtime significantly, but increases the memory footprint.
Example: 'NumThreads',4
Data Types: double

**OutputDirectory — Directory to store analysis results**
current directory ("./") (default) | string | character vector

Directory to store analysis results, specified as a string or character vector.
Example: 'OutputDirectory',"./AnalysisResults/
Data Types: char | string
Seed — Seed for random number generator
0 (default) | nonnegative integer

Seed for the random number generator, specified as a nonnegative integer. Setting a seed value ensures the reproducibility of the analysis results.

Example: 'Seed', 10

Data Types: double

TimeSeries — Flag to treat input samples as time series
false (default) | true

Flag to treat input samples as a time series rather than as independent experimental conditions, specified as true or false. If you set the value to true, you must provide samples in order of increasing time: the first SAM file must be for the first time point, the second SAM file for the second time point, and so on.

Example: 'TimeSeries', true

Data Types: logical

Output Arguments

isoformsDiff — Name of file containing transcript-level differential expression results
"./isoform_exp.diff"

Name of a file containing transcript-level differential expression results, returned as a string.

The output string also includes the directory information defined by OutputDirectory. The default is the current directory. If you set OutputDirectory to "/local/tmp/", the output becomes "/local/tmp/isoform_exp.diff".

geneDiff — Name of file containing gene-level differential expression results
"./gene_exp.diff"

Name of a file containing gene-level differential expression results, returned as a string.

The output string also includes the directory information defined by OutputDirectory. The default is the current directory. If you set OutputDirectory to "/local/tmp/", the output becomes "/local/tmp/gene_exp.diff".

tssDiff — Name of file containing primary transcript differential expression results
"./tss_group_exp.diff"

Name of a file containing primary transcript differential expression results, returned as a string.

The output string also includes the directory information defined by OutputDirectory. The default is the current directory. If you set OutputDirectory to "/local/tmp/", the output becomes "/local/tmp/tss_group_exp.diff".

cdsExp — Name of file containing coding sequence differential expression results
"./cds_exp.diff"

Name of a file containing coding sequence differential expression results, returned as a string.
The output string also includes the directory information defined by `OutputDirectory`. The default is the current directory. If you set `OutputDirectory` to "'/local/tmp/'", the output becomes "'/local/tmp/cds_exp.diff'".

**splicingDiff** — Name of file containing differential splicing results for isoforms

"./splicing.diff"

Name of a file containing differential splicing results for isoforms, returned as a string.

The output string also includes the directory information defined by `OutputDirectory`. The default is the current directory. If you set `OutputDirectory` to "'/local/tmp/'", the output becomes "'/local/tmp/splicing.diff'".

**cdsDiff** — Name of file containing differential coding sequence output

"./cds.diff"

Name of a file containing differential coding sequence output, returned as a string.

The output string also includes the directory information defined by `OutputDirectory`. The default is the current directory. If you set `OutputDirectory` to "'/local/tmp/'", the output becomes "'/local/tmp/cds.diff'".

**promotersDiff** — Name of file containing information on differential promoter use

"./promoters.diff"

Name of a file containing information on differential promoter use that exists between samples, returned as a string.

The output string also includes the directory information defined by `OutputDirectory`. The default is the current directory. If you set `OutputDirectory` to "'/local/tmp/'", the output becomes "'/local/tmp/promoters.diff'".

**References**


**See Also**

CuffDiffOptions | cufflinks

**External Websites**

Cufflinks manual

**Introduced in R2019a**
CuffDiffOptions

Option set for cuffdiff

Description

A CuffDiffOptions object sets options for the cuffdiff function, which identifies significant changes in transcript expression [1].

Creation

Syntax

cuffdiffOpt = CuffDiffOptions

cuffdiffOpt = CuffDiffOptions(Name,Value)

cuffdiffOpt = CuffDiffOptions(S)

Description

cuffdiffOpt = CuffDiffOptions creates a CuffDiffOptions object with the default property values.

CuffDiffOptions requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

Note CuffDiffOptions is supported on the Mac and UNIX platforms only.

cuffdiffOpt = CuffDiffOptions(Name,Value) sets the object properties using one or more name-value pair arguments. Enclose each property name in quotes. For example, cuffdiffOpt = CuffDiffOptions('SupressMapFiles',true) prevents the creation of .tmap and .refmap files.

cuffdiffOpt = CuffDiffOptions(S) specifies optional parameters using a string or character vector S.

Input Arguments

S — cuffdiff options
string | character vector

cuffdiff options, specified as a string or character vector. S must be in the original cuffdiff option syntax (prefixed by one or two dashes).

Example: '--seed 5'
Properties

**ConditionLabels — Sample labels**
string | string vector | character vector | cell array of character vector

Sample labels, specified as a string, string vector, character vector, or cell array of character vectors. The number of labels must equal the number of samples or the value must be empty `[]`.

Example: `"Control","Mutant1","Mutant2"`

Data Types: string | char | cell

**ContrastFile — Contrast file name**
string | character vector

Contrast file name, specified as a string or character vector. The file must be a two-column tab-delimited text file, where each line indicates two conditions to compare using cuffdiff. The condition labels in the file must match either the labels specified for `ConditionLabels` or the sample names. The file must have a single header line as the first line, followed by one line for each contrast. An example of the contrast file format follows.

<table>
<thead>
<tr>
<th>condition_A</th>
<th>condition_B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>Mutant1</td>
</tr>
<tr>
<td>Control</td>
<td>Mutant2</td>
</tr>
</tbody>
</table>

If you do not provide this file, cuffdiff compares every pair of input conditions, which can impact performance.

Example: "contrast.txt"

Data Types: char | string

**DispersionMethod — Method to model variance in fragment counts**
"pooled" (default) | "per-condition" | "blind" | "poisson"

Method to model the variance in fragment counts across replicates, specified as one of the following options:

- "pooled" — The function uses each replicated condition to build a model and averages these models into a global model for all conditions in the experiment.
- "per-condition" — The function produces a model for each condition. You can use this option only if all conditions have replicates.
- "blind" — The function treats all samples as replicates of a single global distribution and produces one model.
- "poisson" — Variance in fragment counts is a poisson model, where the fragment count is predicted to be the mean across replicates. This method is not recommended.

Select a method depending on whether you expect the variability in each group of samples to be similar.

- When comparing two groups where the first group has low cross-replicate variability and the second group has high variability, choose the `per-condition` method.
- If the conditions have similar levels of variability, choose the `pooled` method.
If you have only a single replicate in each condition, choose the blind method.

Example: "blind"

Data Types: char | string

**DoIsoformSwitch** — Flag to perform isoform switching tests

true (default) | false

Flag to perform isoform switching tests, specified as true or false. These tests estimate how much differential splicing exists in isoforms from a single primary transcript. By default, the value is true and the test results are saved in the output file `splicing.diff`.

Example: false

Data Types: logical

**EffectiveLengthCorrection** — Flag to normalize fragment counts

true (default) | false

Flag to normalize fragment counts to fragments per kilobase per million mapped reads (FPKM), specified as true or false.

Example: false

Data Types: logical

**ExtraCommand** — Additional commands

"" (default) | string | character vector

Additional commands, specified as a string or character vector. The commands must be in the original syntax (prefixed by one or two dashes). Use this option to apply undocumented flags and flags without corresponding MATLAB properties. When the function converts the original flags to MATLAB properties, it stores any unrecognized flags in this option.

Example: '--library-type fr-secondstrand'

Data Types: char | string

**FalseDiscoveryRate** — False discovery rate

0.05 (default) | scalar between 0 and 1

False discovery rate used during statistical tests, specified as a scalar between 0 and 1.

Example: 0.01

Data Types: double

**FragmentBiasCorrection** — Name of FASTA file with reference transcripts to detect bias

string | character vector

Name of the FASTA file with reference transcripts to detect bias in fragment counts, specified as a string or character vector. Library preparation can introduce sequence-specific bias into RNA-Seq experiments. Providing reference transcripts improves the accuracy of the transcript abundance estimates.

Example: "bias.fasta"

Data Types: char | string
**FragmentLengthMean — Expected mean fragment length in base pairs**

200 (default) | positive integer

Expected mean fragment length, specified as a positive integer. The default value is 200 base pairs. The function can learn the fragment length mean for each SAM file. Using this option is not recommended for paired-end reads.

Example: 100

Data Types: double

**FragmentLengthSD — Expected standard deviation for fragment length distribution**

80 (default) | positive scalar

Expected standard deviation for the fragment length distribution, specified as a positive scalar. The default value is 80 base pairs. The function can learn the fragment length standard deviation for each SAM file. Using this option is not recommended for paired-end reads.

Example: 70

Data Types: double

**GenerateAnalysisDiff — Flag to create differential analysis files**

true (default) | false

Flag to create differential analysis files (*.diff), specified as true or false.

Example: false

Data Types: logical

**IncludeAll — Flag to use all object properties**

false (default) | true

Flag to include all the object properties with the corresponding default values when converting to the original options syntax, specified as true or false. You can convert the properties to the original syntax prefixed by one or two dashes (such as '-d 100 -e 80') by using getCommand. The default value false means that when you call getCommand(optionsObject), it converts only the specified properties. If the value is true, getCommand converts all available properties, with default values for unspecified properties, to the original syntax.

Example: true

Data Types: logical

**IsoformShiftReplicates — Minimum number of replicates to test genes for differential regulation**

3 (default) | positive integer

Minimum number of replicates to test genes for differential regulation, specified as a positive integer. The function skips the tests when the number of replicates is smaller than the specified value.

Example: 2

Data Types: double

**LengthCorrection — Flag to correct by transcript length**

true (default) | false
Flag to correct by the transcript length, specified as true or false. Set this value to false only when the fragment count is independent of the feature size, such as for small RNA libraries with no fragmentation and for 3' end sequencing, where all fragments have the same length.

Example: false
Data Types: logical

**LibraryNormalizationMethod — Method to normalize library size**
"geometric" (default) | "classic-fpkm" | "quartile"

Method to normalize the library size, specified as one of the following options:

- "geometric" — The function scales the FPKM values by the median geometric mean of fragment counts across all libraries as described in [2].
- "classic-fpkm" — The function applies no scaling to the FPKM values or fragment counts.
- "quartile" — The function scales the FPKM values by the ratio of upper quartiles between fragment counts and the average value across all libraries.

Example: "classic-fpkm"
Data Types: char | string

**MaskFile — Name of GTF or GFF file containing transcripts to ignore**
string | character vector

Name of the GTF or GFF file containing transcripts to ignore during analysis, specified as a string or character vector. Some examples of transcripts to ignore include annotated rRNA transcripts, mitochondrial transcripts, and other abundant transcripts. Ignoring these transcripts improves the robustness of the abundance estimates.

Example: "excludes.gtf"
Data Types: char | string

**MaxBundleFrgs — Maximum number of fragments to include for each locus before skipping**
500000 (default) | positive integer

Maximum number of fragments to include for each locus before skipping new fragments, specified as a positive integer. Skipped fragments are marked with the status HIDATA in the file skipped.gtf.

Example: 400000
Data Types: double

**MaxFragAlignments — Maximum number of aligned reads to include for each fragment**
Inf (default) | positive integer

Maximum number of aligned reads to include for each fragment before skipping new reads, specified as a positive integer. Inf, the default value, sets no limit on the maximum number of aligned reads.

Example: 1000
Data Types: double

**MaxMLEIterations — Maximum number of iterations for maximum likelihood estimation**
5000 (default) | positive integer
Maximum number of iterations for the maximum likelihood estimation of abundances, specified as a positive integer.
Example: 4000
Data Types: double

**MinAlignmentCount — Minimum number of alignments required in locus for significance testing**
10 (default) | positive integer

Minimum number of alignments required in a locus to perform the significance testing for differences between samples, specified as a positive integer.
Example: 8
Data Types: double

**MinIsoformFraction — Minimum abundance of isoform to include in differential expression tests**
1e-5 (default) | scalar between 0 and 1

Minimum abundance of an isoform to include in differential expression tests, specified as a scalar between 0 and 1. For alternative isoforms quantified at below the specified value, the function rounds down the abundance to zero. The specified value is a fraction of the major isoform. The function performs this filtering after MLE estimation but before MAP estimation to improve the robustness of confidence interval generation and differential expression analysis. Using a parameter value other than the default is not recommended.
Example: 1e-5
Data Types: double

**MultiReadCorrection — Flag to improve abundance estimation using rescue method**
false (default) | true

Flag to improve abundance estimation for reads mapped to multiple genomic positions using the rescue method, specified as true or false. If the value is false, the function divides multimapped reads uniformly to all mapped positions. If the value is true, the function uses additional information, including gene abundance estimation, inferred fragment length, and fragment bias, to improve transcript abundance estimation.

The rescue method is described in [3].
Example: true
Data Types: logical

**NormalizeCompatibleHits — Flag to use only fragments compatible with reference transcript to calculate FPKM values**
true (default) | false

Flag to use only fragments compatible with a reference transcript to calculate FPKM values, specified as true or false.
Example: false
Data Types: logical
NormalizeTotalHits — Flag to include all fragments to calculate FPKM values
false (default) | true

Flag to include all fragments to calculate FPKM values, specified as true or false. If the value is true, the function includes all fragments, including fragments without a compatible reference.

Example: true
Data Types: logical

NumFragAssignmentDraws — Number of fragment assignments to perform on each transcript
50 (default) | positive integer

Number of fragment assignments to perform on each transcript, specified as a positive integer. For each fragment drawn from a transcript, the function performs the specified number of assignments probabilistically to determine the transcript assignment uncertainty and to estimate the variance-covariance matrix for the assigned fragment counts.

Example: 40
Data Types: double

NumFragDraws — Number of draws from negative binomial random number generator
100 (default) | positive integer

Number of draws from the negative binomial random number generator for each transcript, specified as a positive integer. Each draw is a number of fragments that the function probabilistically assigns to transcripts in the transcriptome to determine the assignment uncertainty and to estimate the variance-covariance matrix for assigned fragment counts.

Example: 90
Data Types: double

NumThreads — Number of parallel threads to use
1 (default) | positive integer

Number of parallel threads to use, specified as a positive integer. Threads are run on separate processors or cores. Increasing the number of threads generally improves the runtime significantly, but increases the memory footprint.

Example: 4
Data Types: double

OutputDirectory — Directory to store analysis results
current directory ("./") (default) | string | character vector

Directory to store analysis results, specified as a string or character vector.

Example: "./AnalysisResults/
Data Types: char | string

Seed — Seed for random number generator
0 (default) | nonnegative integer

Seed for the random number generator, specified as a nonnegative integer. Setting a seed value ensures the reproducibility of the analysis results.
Example: 10
Data Types: double

**TimeSeries — Flag to treat input samples as time series**
false (default) | true

Flag to treat input samples as a time series rather than as independent experimental conditions, specified as true or false. If you set the value to true, you must provide samples in order of increasing time: the first SAM file must be for the first time point, the second SAM file for the second time point, and so on.

Example: true
Data Types: logical

**Version — Supported version**
string

This property is read-only.

Supported version of the original cufflinks software, returned as a string.

Example: "2.2.1"
Data Types: string

**Object Functions**
getCommand Translate object properties to original options syntax
getOptionsTable Return table with all properties and equivalent options in original syntax

**Examples**

**Create CuffDiffOptions Object**

Create a CuffDiffOptions object with the default values.

opt = CuffDiffOptions;

Create an object using name-value pairs.

opt2 = CuffDiffOptions('FalseDiscoveryRate',0.01,'NumThreads',4)

Create an object by using the original syntax.

opt3 = CuffDiffOptions('--FDR 0.01 --num-threads 4')

**Assemble Transcriptome and Perform Differential Expression Testing**

Create a CufflinksOptions object to define cufflinks options, such as the number of parallel threads and the output directory to store the results.

cflOpt = CufflinksOptions;
cflOpt.NumThreads = 8;
cflOpt.OutputDirectory = "./cufflinksOut";
The SAM files provided for this example contain aligned reads for *Mycoplasma pneumoniae* from two samples with three replicates each. The reads are simulated 100bp-reads for two genes (gyrA and gyrB) located next to each other on the genome. All the reads are sorted by reference position, as required by cufflinks.

```matlab
sams = ['Myco_1_1.sam','Myco_1_2.sam','Myco_1_3.sam', ...
    'Myco_2_1.sam', 'Myco_2_2.sam', 'Myco_2_3.sam'];
```

Assemble the transcriptome from the aligned reads.

```matlab
[gtfs, isofpkm, genes, skipped] = cufflinks(sams, clfOpt);
gtfs is a list of GTF files that contain assembled isoforms.
```

Compare the assembled isoforms using cuffcompare.

```matlab
stats = cuffcompare(gtfs);
```

Merge the assembled transcripts using cuffmerge.

```matlab
mergedGTF = cuffmerge(gtfs,'OutputDirectory','./cuffMergeOutput');
```

mergedGTF reports only one transcript. This is because the two genes of interest are located next to each other, and cuffmerge cannot distinguish two distinct genes. To guide cuffmerge, use a reference GTF (gyrAB.gtf) containing information about these two genes. If the file is not located in the same directory that you run cuffmerge from, you must also specify the file path.

```matlab
gyrAB = which('gyrAB.gtf');
mergedGTF2 = cuffmerge(gtfs, 'OutputDirectory','./cuffMergeOutput2',...
    'ReferenceGTF', gyrAB);
```

Calculate abundances (expression levels) from aligned reads for each sample.

```matlab
abundances1 = cuffquant(mergedGTF2,["Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam"],...
    'OutputDirectory','./cuffquantOutput1');
abundances2 = cuffquant(mergedGTF2,["Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"],...
    'OutputDirectory','./cuffquantOutput2');
```

Assess the significance of changes in expression for genes and transcripts between conditions by performing the differential testing using cuffdiff. The cuffdiff function operates in two distinct steps: the function first estimates abundances from aligned reads, and then performs the statistical analysis. In some cases (for example, distributing computing load across multiple workers), performing the two steps separately is desirable. After performing the first step with cuffquant, you can then use the binary CXB output file as an input to cuffdiff to perform statistical analysis. Because cuffdiff returns several files, specify the output directory is recommended.

```matlab
isoformDiff = cuffdiff(mergedGTF2,[abundances1, abundances2],...
    'OutputDirectory','./cuffdiffOutput');
```

Display a table containing the differential expression test results for the two genes gyrB and gyrA.

```matlab
readtable(isoformDiff,'FileType','text')
an =
2×14 table
    test_id    gene_id    gene    locus    sample_1    sample_2    status    value_1    value_2    log2_fold_change_    test_stat    p_value    q_value    significant
```
You can use `cuffnorm` to generate normalized expression tables for further analyses. `cuffnorm` results are useful when you have many samples and you want to cluster them or plot expression levels for genes that are important in your study. Note that you cannot perform differential expression analysis using `cuffnorm`.

Specify a cell array, where each element is a string vector containing file names for a single sample with replicates.

```matlab
alignmentFiles = {"Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam",...
                 "Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"};
isoformNorm = cuffnorm(mergedGTF2, alignmentFiles,...
                      'OutputDirectory', './cuffnormOutput');
```

Display a table containing the normalized expression levels for each transcript.

```matlab
readtable(isoformNorm,'FileType','text')
```

You can use `cuffnorm` to generate normalized expression tables for further analyses. `cuffnorm` results are useful when you have many samples and you want to cluster them or plot expression levels for genes that are important in your study. Note that you cannot perform differential expression analysis using `cuffnorm`.

You can use `cuffnorm` to generate normalized expression tables for further analyses. `cuffnorm` results are useful when you have many samples and you want to cluster them or plot expression levels for genes that are important in your study. Note that you cannot perform differential expression analysis using `cuffnorm`.

Specify a cell array, where each element is a string vector containing file names for a single sample with replicates.

```matlab
alignmentFiles = {"Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam",...
                 "Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"};
isoformNorm = cuffnorm(mergedGTF2, alignmentFiles,...
                      'OutputDirectory', './cuffnormOutput');
```

Display a table containing the normalized expression levels for each transcript.

```matlab
readtable(isoformNorm,'FileType','text')
```

Column names starting with `q` have the format: `conditionX_N`, indicating that the column contains values for replicate `N` of `conditionX`.

References


See Also

CufflinksOptions | cuffcompare | cuffdiff | cuffgffread | cuffgtf2sam | cufflinks | cuffmerge | cuffnorm | cuffquant

External Websites
Cufflinks manual
Introduced in R2019a
cuffgffread

Filter and convert GFF and GTF files

Syntax

cuffgffread(input,output)
cuffgffread(input,output,opt)
cuffgffread(input,output,Name,Value)

Description

cuffgffread(input,output) reads the input GFF or GTF file and writes the mandatory columns to the output GFF file [1]. The function can also return the GTF-format file using the 'GTFOutput' option.

cuffgffread requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

Note cuffgffread is supported on the Mac and UNIX platforms only.

cuffgffread(input,output,opt) uses the additional options specified by opt.

cuffgffread(input,output,Name,Value) uses additional options specified by one or more name-value pair arguments. For example,
cuffgffread('gyrAB.gtf','gyrAB.gff','PreserveAttributes',true) retains all attributes in the output file.

Examples

Convert GTF to GFF Format

Convert a GTF file to a GFF file while retaining all attributes.
cuffgffread('gyrAB.gtf','gyrABout.gff','PreserveAttributes',true)

You can also set the options using an object. For instance, specify the output to be in the GTF format.

opt = CuffGFFReadOptions;
opt.GTFOutput = true;
opt.PreserveAttributes = true;
cuffgffread('gyrAB.gtf','gyrABout.gtf',opt);

Once you have the options object, you can retrieve the equivalent original options for all object properties using getOptionsTable.

getOptionsTable(opt)

ans =
### Input Arguments

**input — Input file name**

string | character vector

Input file name, specified as a string or character vector. The file can be a GTF or GFF file.

Example: 'gyrAB.gtf'

Data Types: char | string

**output — Output file name**

string | character vector

Output file name, specified as a string or character vector. By default, the output is a GFF file. Set 'GTFOutput' to true to get a GTF output file.

Example: 'gyrAB.gff'
Data Types: char | string

**opt — cuffgffread options**
CuffGFFReadOptions object | string | character vector

cuffgffread options, specified as a CuffGFFReadOptions object, string, or character vector. The string or character vector must be in the original gffread option syntax (prefixed by one or two dashes) [1].

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example:
cuffgffread('gyrAB.gtf','gyrAB.gff','CoordinateRange','+NC_000912.1:4821..734 0')

**AppendDescription — Flag to add file descriptions to descr attribute**
false (default) | true

Flag to add file descriptions from sequence files to the descr attribute of the output GFF record, specified as true or false. Specify the sequence files using the SequenceInfo option.

Example: 'AppendDescription',true

Data Types: logical

**CheckOppositeStrand — Flag to check opposite strand when checking for in-frame stop codons**
false (default) | true

Flag to check opposite strand when checking for in-frame stop codons, specified as true or false.

Example: 'CheckOppositeStrand',true

Data Types: logical

**CheckPhase — Flag to adjust coding sequence phase**
false (default) | true

Flag to adjust coding sequence phase when checking for in-frame stop codons, specified as true or false.

Example: 'CheckPhase',true

Data Types: logical

**Cluster — Flag to cluster input transcripts into loci**
true (default) | false

Flag to cluster the input transcripts into loci, specified as true or false. This option is the same as the Merge property, except that it does not collapse fully contained transcripts with identical introns.

Example: 'Cluster',false

Data Types: logical
CodingOnly — Flag to discard transcripts with no coding sequence
false (default) | true

Flag to discard transcripts with no coding sequence feature (CDS), specified as true or false.
Example: 'CodingOnly',true
Data Types: logical

CollapseContainer — Flag to collapse fully contained transcripts
false (default) | true

Flag to collapse fully contained transcripts that are shorter with fewer introns than the container, specified as true or false. This property applies only when you set Merge to true.
Example: 'CollapseContainer',true
Data Types: logical

CollapseFull — Flag to collapse shorter transcripts overlapping at least 80% with another exon
false (default) | true

Flag to collapse shorter transcripts overlapping at least 80% with another single exon transcript, specified as true or false. This property applies only when you set Merge to true.
Example: 'CollapseFull',true
Data Types: logical

CoordinateRange — Genomic range to filter transcripts
string | character vector

Genomic range to filter transcripts, specified as a string or character vector. The format must be "[[<strand>]<chr>]:<start>..<end>", where start and end are genomic positions, chr is an optional chromosome or contig name, and an optional strand ('+' or '-').
Example: 'CoordinateRange','+NC_000912.1:4821..7340'
Data Types: char | string

DiscardInvalidCDS — Flag to ignore mRNA transcripts either lacking start or stop codon or having in-frame stop codon
false (default) | true

Flag to ignore mRNA transcripts either lacking a start or stop codon or having an in-frame stop codon, specified as true or false.
Example: 'DiscardInvalidCDS',true
Data Types: logical

DiscardNonCanonicalSplice — Flag to ignore multiexon mRNA transcripts that have intron with noncanonical splice sequence
false (default) | true

Flag to ignore multiexon mRNA transcripts that have an intron with a noncanonical splice sequence, specified as true or false. A noncanonical splice sequence is any splice sequence other than "GT-AG", "CG-AG", or "AT-AC".
Example: 'DiscardNonCanonicalSplice',true
Data Types: logical

**DiscardSingleExon — Flag to ignore transcripts spanning single exon**
false (default) | true

Flag to ignore transcripts spanning a single exon, specified as true or false.
Example: 'DiscardSingleExon',true
Data Types: logical

**DiscardTerminatedCDS — Flag to ignore transcripts with in-frame stop codon**
false (default) | true

Flag to ignore transcripts with an in-frame stop codon, specified as true or false.
Example: 'DiscardTerminatedCDS',true
Data Types: logical

**ExtraCommand — Additional commands**
"" (default) | character vector | string

Additional commands, specified as a string or character vector. The commands must be in the original syntax (prefixed by one or two dashes). Use this option to apply undocumented flags and flags without corresponding MATLAB properties. When the function converts the original flags to MATLAB properties, it stores any unrecognized flags in this option.
Example: 'ExtraCommand','-E'
Data Types: char | string

**FastaCDSFile — Name of file to save spliced coding sequences**
string | character vector

Name of a file to save the spliced coding sequences in the FASTA format, specified as a string or character vector.
Example: 'FastaCDSFile','splicedCoding.FASTA'
Data Types: char | string

**FastaExonsFile — Name of file to save spliced exons**
string | character vector

Name of a file to save the spliced exons in the FASTA format, specified as a string or character vector.
Example: 'FastaExonsFile','splicedExon.FASTA'
Data Types: char | string

**FastaProteinFile — Name of file to save protein translation of coding sequences**
string | character vector

Name of a file to save the protein translation of coding sequences in the FASTA format, specified as a string or character vector.
Example: 'FastaProteinFile','translated.FASTA'
Data Types: char | string

FirstExonOnly — Flag to parse additional attributes only from first exon
false (default) | true
Flag to parse additional attributes only from the first exon, specified as true or false.
Example: 'FirstExonOnly',true
Data Types: logical

ForceExons — Flag to list lowest-level GFF features as exon features
false (default) | true
Flag to list the lowest-level GFF features as exon features in the output file, specified as true or false.
Example: 'ForceExons',true
Data Types: logical

FullyContained — Flag to discard transcripts not contained fully
false (default) | true
Flag to discard transcripts not contained fully within the range, specified as true or false. Specify the range using the CoordinateRange option.
Example: 'FullyContained',true
Data Types: logical

GTFOutput — Flag to output GTF-format transcript files
false (default) | true
Flag to output GTF-format transcript files, specified as true or false.
Example: 'GTFOutput',true
Data Types: logical

IncludeAll — Flag to apply all available options
false (default) | true
Flag to include all available options with the corresponding default values when converting to the original options syntax, specified as true or false. The original syntax is prefixed by one or two dashes, such as '-d 100 -e 80'. By default, the function converts only the specified options. If the value is true, the function converts all available options, with default values for unspecified options, to the original syntax.
Example: 'IncludeAll',true
Data Types: logical

MaxIntronLength — Maximum intron length for transcript to include in output
Inf (default) | positive integer
Maximum intron length for a transcript to include in the output file, specified as a positive integer. Inf, the default value, sets no limit on the intron length.
Example: 'MaxIntronLength',500
Data Types: double

**Merge** — Flag to merge transcripts to loci

false (default) | true

Flag to merge transcripts into loci by collapsing transcripts with identical introns, specified as true or false.

Example: 'Merge',true

Data Types: logical

**MergeCloseExons** — Flag to merge exons into single exon

false (default) | true

Flag to merge exons into a single exon when separated by fewer than 4 base-pair introns, specified as true or false.

Example: 'MergeCloseExons',true

Data Types: logical

**MergeInfoFile** — Name of file to save information on duplicates when merging

string | character vector

Name of a file to save information on duplicates when merging, specified as a string or character vector. This property applies only when you set Merge to true.

Example: 'MergeInfoFile','duplicates.txt'

Data Types: char | string

**PreserveAttributes** — Flag to retain all attributes in output

false (default) | true

Flag to retain all attributes in the output file, specified as true or false.

Example: 'PreserveAttributes',true

Data Types: logical

**Pseudo** — Flag to filter out records containing "pseudo"

true (default) | false

Flag to filter out records containing the word "pseudo," specified as true or false.

Example: 'Pseudo',false

Data Types: logical

**ReplacementTable** — Name of file containing replacement table

string | character vector

Name of a file containing a replacement table, specified as a string or character vector. The table must have two columns, where the first column contains the original transcript IDs and the second column contains the new transcript IDs. An example table follows.

<table>
<thead>
<tr>
<th>origTranscript1</th>
<th>newTranscript1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-455</td>
<td></td>
</tr>
</tbody>
</table>
If you provide a replacement table, the function replaces the transcript IDs found in the first column with the new transcripts IDs from the second column and filters out those transcripts not found.

Example: 'ReplacementTable',"replaceTbl.txt"

Data Types: char | string

**SequenceFile** — Name of FASTA-format file containing genomic sequences

string | character vector

Name of a FASTA-format file containing genomic sequences for all input mappings, specified as a string or character vector.

Example: 'SequenceFile',"seqs.fasta"

Data Types: char | string

**SequenceInfo** — Name of tab-delimited file with additional information on input sequence

string | character vector

Name of a tab-delimited file with additional information on each input sequence, specified as a string or character vector. This file must have three columns: a sequence name column, a sequence length column, and a sequence description column. If AppendDescription is true, the sequence description is included as an attribute in the output GFF file.

Example: 'SequenceInfo',"seqinfo.txt"

Data Types: char | string

**UrlDecode** — Flag to decode URL-encoded characters in attribute names

false (default) | true

Flag to decode url-encoded characters in attribute names, specified as true or false. For instance, "transcript%20description" is decoded to "transcript description".

Example: 'UrlDecode',true

Data Types: logical

**UseEnsemblConversion** — Flag to use GTF-to-GFF3 conversion method from Ensembl

false (default) | true

Flag to use the GTF-to-GFF3 conversion method from Ensembl, specified as true or false.

Example: 'UseEnsemblConversion',true

Data Types: logical

**UseNonTranscript** — Flag to include nontranscript GFF records in output file

false (default) | true

Flag to include nontranscript GFF records in the output file, specified as true or false.

Example: 'UseNonTranscript',true

Data Types: logical
UseTrackName — Flag to use track name in second column of GFF output line
false (default) | true

Flag to use the track name in the second column of the GFF output line, specified as true or false.
Example: 'UseTrackName',true
Data Types: logical

WriteCoordinates — Flag to write exon coordinates projected onto spliced sequence
false (default) | true

Flag to write the exon coordinates projected onto the spliced sequence, specified as true or false. This property applies only when FastaExonsFile or FastaCDSFile is specified.
Example: 'WriteCoordinates',true
Data Types: logical

References

See Also
CuffGFFReadOptions | cufflinks

External Websites
Cufflinks manual

Introduced in R2019a
CuffGFFReadOptions

Option set for cuffgffread

Description

A CuffGFFReadOptions object contains options for the cuffgffread function, which filters and converts GFF and GTF files [1].

Creation

Syntax

cuffgffreadOpt = CuffGFFReadOptions
cuffgffreadOpt = CuffGFFReadOptions(Name,Value)
cuffgffreadOpt = CuffGFFReadOptions(S)

Description

cuffgffreadOpt = CuffGFFReadOptions creates a CuffGFFReadOptions object with the default property values.

CuffGFFReadOptions requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

Note CuffGFFReadOptions is supported on the Mac and UNIX platforms only.

cuffgffreadOpt = CuffGFFReadOptions(Name,Value) sets the object properties on page 1-459 using one or more name-value pair arguments. Enclose each property name in quotes. For example, cuffgffreadOpt = CuffGFFReadOptions('DiscardSingleExon',true) discards transcripts spanning a single exon.

cuffgffreadOpt = CuffGFFReadOptions(S) specifies optional parameters using the string or character vector S.

Input Arguments

S — cuffgffread options
string | character vector

cuffgffread options, specified as a string or character vector. S must be in the original gffread option syntax (prefixed by one or two dashes).

Example: ' -U'
Properties

AppendDescription — Flag to add file descriptions to descr attribute
false (default) | true

Flag to add file descriptions from sequence files to the descr attribute of the output GFF record, specified as true or false. Specify the sequence files using the SequenceInfo option.
Example: true
Data Types: logical

CheckOppositeStrand — Flag to check opposite strand when checking for in-frame stop codons
false (default) | true

Flag to check opposite strand when checking for in-frame stop codons, specified as true or false.
Example: true
Data Types: logical

CheckPhase — Flag to adjust coding sequence phase
false (default) | true

Flag to adjust coding sequence phase when checking for in-frame stop codons, specified as true or false.
Example: true
Data Types: logical

Cluster — Flag to cluster input transcripts into loci
true (default) | false

Flag to cluster the input transcripts into loci, specified as true or false. This option is the same as the Merge property, except that it does not collapse fully contained transcripts with identical introns.
Example: false
Data Types: logical

CodingOnly — Flag to discard transcripts with no coding sequence
false (default) | true

Flag to discard transcripts with no coding sequence feature (CDS), specified as true or false.
Example: true
Data Types: logical

CollapseContainer — Flag to collapse fully-contained transcripts
false (default) | true

Flag to collapse fully contained transcripts that are shorter with fewer introns than the container, specified as true or false. This property applies only when you set Merge to true.
Example: true
Data Types: logical
CollapseFull — Flag to collapse shorter transcripts overlapping at least 80% with another exon
false (default) | true
Flag to collapse shorter transcripts overlapping at least 80% with another single exon transcript, specified as true or false. This property applies only when you set Merge to true.
Example: true
Data Types: logical

CoordinateRange — Genomic range to filter transcripts
string | character vector
Genomic range to filter transcripts, specified as a string or character vector. The format must be “[[<strand>]<chr>:]<start>..<end>”, where start and end are genomic positions, chr is an optional chromosome or contig name, and an optional strand (’+’ or ’-’).
Example: “+NC_000912.1:4821..7340”
Data Types: char | string

DiscardInvalidCDS — Flag to ignore mRNA transcripts either lacking start or stop codon or having in-frame stop codon
false (default) | true
Flag to ignore mRNA transcripts either lacking a start or stop codon or having an in-frame stop codon, specified as true or false.
Example: true
Data Types: logical

DiscardNonCanonicalSplice — Flag to ignore multiexon mRNA transcripts that have intron with noncanonical splice sequence
false (default) | true
Flag to ignore multiexon mRNA transcripts that have an intron with a noncanonical splice sequence, specified as true or false. A noncanonical splice sequence is any splice sequence other than "GT-AG", "CG-AG", or "AT-AC".
Example: true
Data Types: logical

DiscardSingleExon — Flag to ignore transcripts spanning single exon
false (default) | true
Flag to ignore transcripts spanning a single exon, specified as true or false.
Example: true
Data Types: logical

DiscardTerminatedCDS — Flag to ignore transcripts with in-frame stop codon
false (default) | true
Flag to ignore transcripts with an in-frame stop codon, specified as true or false.
Example: true
Data Types: logical

**ExtraCommand — Additional commands**

```
" " (default) | character vector | string
```

Additional commands, specified as a string or character vector. The commands must be in the original syntax (prefixed by one or two dashes). Use this option to apply undocumented flags and flags without corresponding MATLAB properties. When the function converts the original flags to MATLAB properties, it stores any unrecognized flags in this option.

Example: "-E"

Data Types: char | string

**FastaCDSFile — Name of file to save spliced coding sequences**

```
string | character vector
```

Name of a file to save the spliced coding sequences in the FASTA format, specified as a string or character vector.

Example: "splicedCoding.FASTA"

Data Types: char | string

**FastaExonsFile — Name of file to save spliced exons**

```
string | character vector
```

Name of a file to save the spliced exons in the FASTA format, specified as a string or character vector.

Example: "splicedExon.FASTA"

Data Types: char | string

**FastaProteinFile — Name of file to save protein translation of coding sequences**

```
string | character vector
```

Name of a file to save the protein translation of coding sequences in the FASTA format, specified as a string or character vector.

Example: "translated.FASTA"

Data Types: char | string

**FirstExonOnly — Flag to parse additional attributes only from first exon**

```
false (default) | true
```

Flag to parse additional attributes only from the first exon, specified as true or false.

Example: true

Data Types: logical

**ForceExons — Flag to list lowest-level GFF features as exon features**

```
false (default) | true
```

Flag to list the lowest-level GFF features as exon features in the output file, specified as true or false.

Example: true
Data Types: logical

**FullyContained** — Flag to discard transcripts not contained fully
false (default) | true

Flag to discard transcripts not contained fully within the range, specified as true or false. Specify the range using the CoordinateRange option.

Example: true

Data Types: logical

**GTFOutput** — Flag to output GTF-format transcript files
false (default) | true

Flag to output GTF-format transcript files, specified as true or false.

Example: true

Data Types: logical

**IncludeAll** — Flag to use all object properties
false (default) | true

Flag to include all the object properties with the corresponding default values when converting to the original options syntax, specified as true or false. You can convert the properties to the original syntax prefixed by one or two dashes (such as ‘-d 100 -e 80’) by using getCommand. The default value false means that when you call getCommand(optionsObject), it converts only the specified properties. If the value is true, getCommand converts all available properties, with default values for unspecified properties, to the original syntax.

Example: true

Data Types: logical

**MaxIntronLength** — Maximum intron length for transcript to be included in output
Inf (default) | positive integer

Maximum intron length for a transcript to include in the output file, specified as a positive integer. Inf, the default value, sets no limit on the intron length.

Example: 500

Data Types: double

**Merge** — Flag to merge transcripts to loci
false (default) | true

Flag to merge transcripts into loci by collapsing transcripts with identical introns, specified as true or false.

Example: true

Data Types: logical

**MergeCloseExons** — Flag to merge exons into single exon
false (default) | true

Flag to merge exons into a single exon when separated by fewer than 4 base-pair introns, specified as true or false.
Example: true
Data Types: logical

MergeInfoFile — Name of file to save information on duplicates when merging
string | character vector

Name of a file to save information on duplicates when merging, specified as a string or character vector. This property applies only when you set Merge to true.
Example: "duplicates.txt"
Data Types: char | string

PreserveAttributes — Flag to retain all attributes in output
false (default) | true

Flag to retain all attributes in the output file, specified as true or false.
Example: true
Data Types: logical

Pseudo — Flag to filter out records containing "pseudo"
true (default) | false

Flag to filter out records containing the word "pseudo," specified as true or false.
Example: false
Data Types: logical

ReplacementTable — Name of file containing replacement table
string | character vector

Name of a file containing a replacement table, specified as a string or character vector. The table must have two columns, where the first column contains the original transcript IDs and the second column contains the new transcript IDs. An example table follows.

<table>
<thead>
<tr>
<th>origTranscript1</th>
<th>newTranscript1</th>
</tr>
</thead>
<tbody>
<tr>
<td>origTranscript2</td>
<td>newTranscript2</td>
</tr>
<tr>
<td>origTranscript3</td>
<td>newTranscript3</td>
</tr>
</tbody>
</table>

If you provide a replacement table, the function replaces the transcript IDs found in the first column with the new transcripts IDs from the second column and filters out those transcripts not found.
Example: "replaceTbl.txt"
Data Types: char | string

SequenceFile — Name of FASTA-format file containing genomic sequences
string | character vector

Name of a FASTA-format file containing genomic sequences for all input mappings, specified as a string or character vector.
Example: "seqs.fasta"
Data Types: char | string
**SequenceInfo — Name of tab-delimited file with additional information on input sequence**
string | character vector

Name of a tab-delimited file with additional information on each input sequence, specified as a string or character vector. This file must have three columns: a sequence name column, a sequence length column, and a sequence description column. If `AppendDescription` is `true`, the sequence description is included as an attribute in the output GFF file.

Example: "seqinfo.txt"
Data Types: char | string

**UrlDecode — Flag to decode URL-encoded characters in attribute names**
false (default) | true

Flag to decode url-encoded characters in attribute names, specified as `true` or `false`. For instance, "transcript%20description" is decoded to "transcript description".

Example: `true`
Data Types: logical

**UseEnsemblConversion — Flag to use GTF-to-GFF3 conversion method from Ensembl**
false (default) | true

Flag to use the GTF-to-GFF3 conversion method from Ensembl, specified as `true` or `false`.

Example: `true`
Data Types: logical

**UseNonTranscript — Flag to include nontranscript GFF records in output file**
false (default) | true

Flag to include nontranscript GFF records in the output file, specified as `true` or `false`.

Example: `true`
Data Types: logical

**UseTrackName — Flag to use track name in second column of GFF output line**
false (default) | true

Flag to use the track name in the second column of the GFF output line, specified as `true` or `false`.

Example: `true`
Data Types: logical

**Version — Supported version**
string

This property is read-only.

Supported version of the original cufflinks software, returned as a string.

Example: "$2.2.1$
Data Types: string
**WriteCoordinates — Flag to write exon coordinates projected onto spliced sequence**

Flag to write the exon coordinates projected onto the spliced sequence, specified as `true` or `false`. This property applies only when `FastaExonsFile` or `FastaCDSFile` is specified.

Example: `true`

Data Types: `logical`

**Object Functions**

- `getCommand`: Translate object properties to original options syntax
- `getOptionsTable`: Return table with all properties and equivalent options in original syntax

**Examples**

**Create CuffGFFReadOptions Object**

Create a `CuffGFFReadOptions` object with the default values.

```matlab
opt = CuffGFFReadOptions;
```

Create an object using name-value pairs.

```matlab
opt2 = CuffGFFReadOptions('DiscardSingleExon',true,'FastaExonsFile','exons.fa');
```

Create an object by using the original syntax.

```matlab
opt3 = CuffGFFReadOptions('-U -w exons.fa')
```

**Convert GTF to GFF Format**

Convert a GTF file to a GFF file while retaining all attributes.

```matlab
cuffgffread('gyrAB.gtf','gyrABOut.gff','PreserveAttributes',true)
```

You can also set the options using an object. For instance, specify the output to be in the GTF format.

```matlab
opt = CuffGFFReadOptions;
opt.GTFOutput = true;
opt.PreserveAttributes = true;
cuffgffread('gyrAB.gtf','gyrABOut.gff',opt);
```

Once you have the options object, you can retrieve the equivalent original options for all object properties using `getOptionsTable`.

```matlab
getOptionsTable(opt)
```

Output:

```
33×3 table

<table>
<thead>
<tr>
<th>PropertyName</th>
<th>FlagName</th>
<th>FlagShortName</th>
</tr>
</thead>
</table>
```
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Option</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppendDescription</td>
<td>AppendDescription</td>
<td><code>-A</code></td>
<td></td>
</tr>
<tr>
<td>CheckOppositeStrand</td>
<td>CheckOppositeStrand</td>
<td><code>-B</code></td>
<td></td>
</tr>
<tr>
<td>CheckPhase</td>
<td>CheckPhase</td>
<td><code>-H</code></td>
<td></td>
</tr>
<tr>
<td>Cluster</td>
<td>Cluster</td>
<td><code>-cluster-only</code></td>
<td></td>
</tr>
<tr>
<td>CodingOnly</td>
<td>CodingOnly</td>
<td><code>-C</code></td>
<td></td>
</tr>
<tr>
<td>CollapseContainer</td>
<td>CollapseContainer</td>
<td><code>-K</code></td>
<td></td>
</tr>
<tr>
<td>CollapseFull</td>
<td>CollapseFull</td>
<td><code>-Q</code></td>
<td></td>
</tr>
<tr>
<td>CoordinateRange</td>
<td>CoordinateRange</td>
<td><code>-r</code></td>
<td></td>
</tr>
<tr>
<td>DiscardInvalidCDS</td>
<td>DiscardInvalidCDS</td>
<td><code>-J</code></td>
<td></td>
</tr>
<tr>
<td>DiscardNonCanonicalSplice</td>
<td>DiscardNonCanonicalSplice</td>
<td><code>-N</code></td>
<td></td>
</tr>
<tr>
<td>DiscardSingleExon</td>
<td>DiscardSingleExon</td>
<td><code>-U</code></td>
<td></td>
</tr>
<tr>
<td>DiscardTerminatedCDS</td>
<td>DiscardTerminatedCDS</td>
<td><code>-V</code></td>
<td></td>
</tr>
<tr>
<td>FastaCDSFile</td>
<td>FastaCDSFile</td>
<td><code>-x</code></td>
<td></td>
</tr>
<tr>
<td>FastaExonsFile</td>
<td>FastaExonsFile</td>
<td><code>-w</code></td>
<td></td>
</tr>
<tr>
<td>FastaProteinFile</td>
<td>FastaProteinFile</td>
<td><code>-y</code></td>
<td></td>
</tr>
<tr>
<td>FirstExonOnly</td>
<td>FirstExonOnly</td>
<td><code>-G</code></td>
<td></td>
</tr>
<tr>
<td>ForceExons</td>
<td>ForceExons</td>
<td><code>-force-exons</code></td>
<td></td>
</tr>
<tr>
<td>FullyContained</td>
<td>FullyContained</td>
<td><code>-R</code></td>
<td></td>
</tr>
<tr>
<td>GTFFormatOutput</td>
<td>GTFFormatOutput</td>
<td><code>-T</code></td>
<td></td>
</tr>
<tr>
<td>MaxIntronLength</td>
<td>MaxIntronLength</td>
<td><code>-i</code></td>
<td></td>
</tr>
<tr>
<td>Merge</td>
<td>Merge</td>
<td><code>-merge</code></td>
<td><code>-M</code></td>
</tr>
<tr>
<td>MergeCloseExons</td>
<td>MergeCloseExons</td>
<td><code>-Z</code></td>
<td></td>
</tr>
<tr>
<td>MergeInfoFile</td>
<td>MergeInfoFile</td>
<td><code>-d</code></td>
<td></td>
</tr>
<tr>
<td>PreserveAttributes</td>
<td>PreserveAttributes</td>
<td><code>-F</code></td>
<td></td>
</tr>
<tr>
<td>Pseudo</td>
<td>Pseudo</td>
<td><code>-no-pseudo</code></td>
<td></td>
</tr>
<tr>
<td>ReplacementTable</td>
<td>ReplacementTable</td>
<td><code>-m</code></td>
<td></td>
</tr>
<tr>
<td>SequenceFile</td>
<td>SequenceFile</td>
<td><code>-g</code></td>
<td></td>
</tr>
<tr>
<td>SequenceInfo</td>
<td>SequenceInfo</td>
<td><code>-s</code></td>
<td></td>
</tr>
<tr>
<td>UrlDecode</td>
<td>UrlDecode</td>
<td><code>-D</code></td>
<td></td>
</tr>
<tr>
<td>UseEnsemblConversion</td>
<td>UseEnsemblConversion</td>
<td><code>-L</code></td>
<td></td>
</tr>
<tr>
<td>UseNonTranscript</td>
<td>UseNonTranscript</td>
<td><code>-O</code></td>
<td></td>
</tr>
<tr>
<td>UseTrackName</td>
<td>UseTrackName</td>
<td><code>-t</code></td>
<td></td>
</tr>
<tr>
<td>WriteCoordinates</td>
<td>WriteCoordinates</td>
<td><code>-W</code></td>
<td></td>
</tr>
</tbody>
</table>

**References**


**See Also**

cuffcompare | cufflinks

**External Websites**

Cufflinks manual

**Introduced in R2019a**
cuffgtf2sam

Convert GTF files to SAM files

Syntax

cuffgtf2sam(input,output)
cuffgtf2sam(input,output,Name,Value)

Description

cuffgtf2sam(input,output) converts the assembled transcripts in the GTF file input to the SAM-format file output [1].

cuffgtf2sam requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

Note cuffgtf2sam is supported on the Mac and UNIX platforms only.

cuffgtf2sam(input,output,Name,Value) uses additional options specified by one or more name-value pair arguments. For example, gtf2sam('hum37_2_1M.gtf','hum37_2_1M.sam','UseFPKM',true) inserts the FPKM value into the SAM records.

Examples

Convert GTF to SAM

Convert a GTF file to a SAM file.

cuffgtf2sam('hum37_2_1M.gtf','hum37_2_1M.sam')

Input Arguments

input — Names of input files
string | character vector | string vector | cell array of character vectors

Names of input files, specified as a string, character vector, string vector, or cell array of character vectors.

Example: 'gyrAB.gtf'

Data Types: cell | char | string

output — Output SAM file name
string | character vector

Output SAM file name, specified as a string or character vector.
Example: 'gyrAB.sam'
Data Types: char | string

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: gtf2sam('hum37_2_1M.gtf','hum37_2_1M.sam','UseFPKM',true)

ReferenceFASTA — Name of reference FASTA file
string | character vector

Name of a reference FASTA file, specified as a string or character vector. If you specify a FASTA file, the function recreates the sequences of transcripts by comparing to the reference sequences in the provided FASTA file. If you do not specify 'ReferenceFASTA', the function omits the sequence information from the output SAM file.

Example: 'ReferenceFASTA','ref.fasta'
Data Types: char | string

UseFPKM — Flag to insert FPKM value
false (default) | true

Flag to insert the FPKM value into the SAM records instead of the isoform fraction, specified as true or false.

Example: 'UseFPKM',true
Data Types: logical

References


See Also
GTFAnnotation | GFFAnnotation | cuffgffread | cufflinks | samread

External Websites
Cufflinks manual

Introduced in R2019a
cufflinks

Assemble transcriptome from aligned reads

Syntax

cufflinks(alignmentFiles)
cufflinks(alignmentFiles,cufflinksOptions)
cufflinks(alignmentFiles,Name,Value)
[transcripts,isoforms,genes,skippedTranscripts] = cufflinks(___)

Description

cufflinks(alignmentFiles) assembles a transcriptome from aligned reads in alignmentFile and quantifies the level of expression for each transcript [1]. By default, the function writes the results to a GTF file named transcripts.gtf in the current directory.

cufflinks requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

Note cufflinks is supported on the Mac and UNIX platforms only.

cufflinks(alignmentFiles,cufflinksOptions) uses additional options specified by cufflinksOptions.

cufflinks(alignmentFiles,Name,Value) uses additional options specified by one or more name-value pair arguments. For example, cufflinks(alignmentFile,'TrimCoverageThreshold',5) specifies the minimum average coverage for 3' end trimming.

[transcripts,isoforms,genes,skippedTranscripts] = cufflinks(__) returns the file names of the assembled transcriptome using any of the input argument combinations from the previous syntaxes. By default, the function saves all files to the current directory.

Examples

Assemble Transcriptome and Perform Differential Expression Testing

Create a CufflinksOptions object to define cufflinks options, such as the number of parallel threads and the output directory to store the results.

cflOpt = CufflinksOptions;
cflOpt.NumThreads = 8;
cflOpt.OutputDirectory = "./cufflinksOut";

The SAM files provided for this example contain aligned reads for Mycoplasma pneumoniae from two samples with three replicates each. The reads are simulated 100bp-reads for two genes (gyrA and gyrB) located next to each other on the genome. All the reads are sorted by reference position, as required by cufflinks.
Assemble the transcriptome from the aligned reads.

\[ \text{gtfs, isofpkm, genes, skipped} = \text{cufflinks}(\text{sams}, \text{cflOpt}); \]

\( \text{gtfs} \) is a list of GTF files that contain assembled isoforms.

Compare the assembled isoforms using cuffcompare.

\[ \text{stats} = \text{cuffcompare(gtfs)}; \]

Merge the assembled transcripts using cuffmerge.

\[ \text{mergedGTF} = \text{cuffmerge(gtfs, 'OutputDirectory', './cuffMergeOutput');} \]

\( \text{mergedGTF} \) reports only one transcript. This is because the two genes of interest are located next to each other, and cuffmerge cannot distinguish two distinct genes. To guide cuffmerge, use a reference GTF (\( \text{gyrAB.gtf} \)) containing information about these two genes. If the file is not located in the same directory that you run cuffmerge from, you must also specify the file path.

\[ \text{gyrAB} = \text{which('gyrAB.gtf');} \]

\[ \text{mergedGTF2} = \text{cuffmerge(gtfs, 'OutputDirectory', './cuffMergeOutput2', 'ReferenceGTF', gyrAB)}; \]

Calculate abundances (expression levels) from aligned reads for each sample.

\[ \text{abundances1} = \text{cuffquant(mergedGTF2, ["Myco_1_1.sam", "Myco_1_2.sam", "Myco_1_3.sam"], 'OutputDirectory', './cuffquantOutput1');} \]

\[ \text{abundances2} = \text{cuffquant(mergedGTF2, ["Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"], 'OutputDirectory', './cuffquantOutput2');} \]

Assess the significance of changes in expression for genes and transcripts between conditions by performing the differential testing using cuffdiff. The cuffdiff function operates in two distinct steps: the function first estimates abundances from aligned reads, and then performs the statistical analysis. In some cases (for example, distributing computing load across multiple workers), performing the two steps separately is desirable. After performing the first step with cuffquant, you can then use the binary CXB output file as an input to cuffdiff to perform statistical analysis. Because cuffdiff returns several files, specify the output directory is recommended.

\[ \text{isoformDiff} = \text{cuffdiff(mergedGTF2, [abundances1, abundances2], 'OutputDirectory', './cuffdiffOutput');} \]

Display a table containing the differential expression test results for the two genes \( \text{gyrB} \) and \( \text{gyrA} \).

\[ \text{readtable(isoformDiff, 'FileType', 'text')} \]

\[ \text{ans} = \]

\[
\begin{array}{cccccc}
\text{test_id} & \text{gene_id} & \text{gene} & \text{locus} & \text{sample_1} & \text{sample_2} \\
\hline
\text{'TCONS_00000001'} & \text{'XLOC_000001'} & \text{'gyrB'} & \text{'NC_000912.1:2868-7340'} & \text{'q1'} & \text{'q2'} \\
\text{'TCONS_00000002'} & \text{'XLOC_000001'} & \text{'gyrA'} & \text{'NC_000912.1:2868-7340'} & \text{'q1'} & \text{'q2'} \\
\end{array}
\]
You can use cuffnorm to generate normalized expression tables for further analyses. cuffnorm results are useful when you have many samples and you want to cluster them or plot expression levels for genes that are important in your study. Note that you cannot perform differential expression analysis using cuffnorm.

Specify a cell array, where each element is a string vector containing file names for a single sample with replicates.

```matlab
alignmentFiles = {{"Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam"},...
    ["Myco_2_1.sam","Myco_2_2.sam","Myco_2_3.sam"]};
isoformNorm = cuffnorm(mergedGTF2, alignmentFiles,...
    'OutputDirectory', './cuffnormOutput');
```

Display a table containing the normalized expression levels for each transcript.

```matlab
readtable(isoformNorm,'FileType','text')
```

<table>
<thead>
<tr>
<th>tracking_id</th>
<th>q1_0</th>
<th>q1_2</th>
<th>q1_1</th>
<th>q2_1</th>
<th>q2_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>'TCONS_00000001'</td>
<td>1.0913e+05</td>
<td>78628</td>
<td>1.2132e+05</td>
<td>4.3639e+05</td>
<td>4.2228e+05</td>
</tr>
<tr>
<td>'TCONS_000000002'</td>
<td>3.5158e+05</td>
<td>3.7458e+05</td>
<td>3.4238e+05</td>
<td>1.0483e+05</td>
<td>1.1546e+05</td>
</tr>
</tbody>
</table>

Column names starting with `q` have the format: `conditionX_N`, indicating that the column contains values for replicate `N` of `conditionX`.

**Input Arguments**

- **alignmentFiles** — Names of SAM or BAM files
  - string | string vector | character vector | cell array of character vectors
  - Names of SAM or BAM files, specified as a string, string vector, character vector, or cell array of character vectors. The input files must be sorted by reference position.
  - Example: `'Myco_1_1.sam'`
  - Data Types: char | string

- **cufflinksOptions** — Cufflinks options
  - CufflinksOptions object | character vector | string
  - Cufflinks options, specified as a CufflinksOptions object, character vector, or string. The character vector or string must be in the cufflinks option syntax (prefixed by one or two dashes) [1].

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

- Example:
  ```matlab
cufflinks(alignmentFile,'TrimCoverageThreshold',5,'FragmentLengthMean',180)
  ```
**EffectiveLengthCorrection** — Flag to normalize fragment counts
true (default) | false

Flag to normalize fragment counts to fragments per kilobase per million mapped reads (FPKM), specified as true or false.

Example: 'EffectiveLengthCorrection',false

Data Types: logical

**ExtraCommand** — Additional commands
"" (default) | string | character vector

Additional commands, specified as a string or character vector. The commands must be in the original syntax (prefixed by one or two dashes). Use this option to apply undocumented flags and flags without corresponding MATLAB properties. When the function converts the original flags to MATLAB properties, it stores any unrecognized flags in this option.

Example: 'ExtraCommand','--library-type fr-secondstrand'

Data Types: char | string

**FauxReadTiling** — Flag to include reference transcripts in assembled output
ture (default) | false

Flag to include reference transcripts in the assembled output as faux-reads during RABT (advanced reference annotation based transcript) assembly, specified as true or false.

**Note** The function only performs the RABT assembly if you specify GTFGuide. Otherwise, FauxReadTiling, regardless of being true or false, has no effect on the assembled transcript.

Example: 'FauxReadTiling',false

Data Types: logical

**FragmentBiasCorrection** — Name of FASTA file with reference transcripts to detect bias
string | character vector

Name of the FASTA file with reference transcripts to detect bias in fragment counts, specified as a string or character vector. Library preparation can introduce sequence-specific bias into RNA-Seq experiments. Providing reference transcripts improves the accuracy of the transcript abundance estimates.

Example: 'FragmentBiasCorrection','ref.fasta'

Data Types: char | string

**FragmentLengthMean** — Expected mean fragment length
200 (default) | positive integer

Expected mean fragment length, specified as a positive integer. The default value is 200 base pairs. The function can learn the fragment length mean for each SAM file. Using this option is not recommended for paired-end reads.

Example: 'FragmentLengthMean',100

Data Types: double
**FragmentLengthSD** — Expected standard deviation for fragment length distribution

80 (default) | positive scalar

Expected standard deviation for the fragment length distribution, specified as a positive scalar. The default value is 80 base pairs. The function can learn the fragment length standard deviation for each SAM file. Using this option is not recommended for paired-end reads.

Example: 'FragmentLengthSTD', 70

Data Types: double

**GTFGuide** — Name of GTF file to guide RABT assembly

string | character vector

Name of a GTF file to guide the RABT assembly, specified as a string or character vector.

Example: 'GTFGuide', 'tr.gtf'

Data Types: char | string

**IncludeAll** — Flag to apply all available options

false (default) | true

Flag to include all available options with the corresponding default values when converting to the original options syntax, specified as true or false. The original syntax is prefixed by one or two dashes, such as '-d 100 -e 80'. By default, the function converts only the specified options. If the value is true, the function converts all available options, with default values for unspecified options, to the original syntax.

Example: 'IncludeAll', true

Data Types: logical

**RABTOverhangTolerance** — Number of base pairs allowed to overlap with transcript intron

8 (default) | positive integer

Number of base pairs from a read allowed to overlap with a transcript intron when determining if a read is mappable to another transcript during the RABT assembly, specified as a positive integer. The default value is 8.

**Note** The function only performs the RABT assembly if you specify GTFGuide. Otherwise, **RABTOverhangTolerance** has no effect on the assembled transcript.

Example: 'IntronOverhangTolerance', 10

Data Types: double

**JunctionAlpha** — Alpha value in binomial test to filter false-positive alignments

0.001 (default) | scalar between 0 and 1

Alpha value in the binomial test to filter false-positive alignments, specified as a scalar between 0 and 1.

Example: 'JunctionAlpha', 0.005

Data Types: double
**LengthCorrection** — Flag to correct by transcript length

true (default) | false

Flag to correct by the transcript length, specified as true or false. Set this value to false only when the fragment count is independent of the feature size, such as for small RNA libraries with no fragmentation and for 3' end sequencing, where all fragments have the same length.

Example: ‘LengthCorrection’,false

Data Types: logical

**MaskFile** — Name of GTF or GFF file containing transcripts to ignore

string | character vector

Name of the GTF or GFF file containing transcripts to ignore during analysis, specified as a string or character vector. Some examples of transcripts to ignore include annotated rRNA transcripts, mitochondrial transcripts, and other abundant transcripts. Ignoring these transcripts improves the robustness of the abundance estimates.

Example: ‘MaskFile’, 'excludes.gtf'

Data Types: char | string

**MaxBundleFrags** — Maximum number of fragments to include for each locus before skipping

500000 (default) | positive integer

Maximum number of fragments to include for each locus before skipping new fragments, specified as a positive integer. Skipped fragments are marked with the status HIDATA in the file skipped.gtf.

Example: ‘MaxBundleFrags’, 400000

Data Types: double

**MaxBundleLength** — Maximum genomic length in base pairs for bundle

3500000 (default) | positive integer

Maximum genomic length in base pairs for a bundle, specified as a positive integer.

Example: ‘MaxBundleLength’, 3400000

Data Types: double

**MaxFragAlignments** — Maximum number of aligned reads to include for each fragment

Inf (default) | positive integer

Maximum number of aligned reads to include for each fragment before skipping new reads, specified as a positive integer. Inf, the default value, sets no limit on the maximum number of aligned reads.

Example: ‘MaxFragAlignments’, 1000

Data Types: double

**MaxIntronLength** — Maximum number of bases in intron

300000 (default) | positive integer

Maximum number of bases in an intron to report, specified as a positive integer. cufflinks also ignores SAM alignments with REF_SKIP CIGAR operations longer than this property value.

Example: ‘MaxIntronLength’, 350000
Data Types: double

**MaxMLEIterations** — Maximum number of iterations for maximum likelihood estimation

5000 (default) | positive integer

Maximum number of iterations for the maximum likelihood estimation of abundances, specified as a positive integer.
Example: 'MaxMLEIterations',4000

Data Types: double

**MinFragsPerTransfrag** — Minimum number of aligned RNA-Seq fragments to report

10 (default) | positive integer

Minimum number of aligned RNA-Seq fragments to report on an assembled transfrag, specified as a positive integer.
Example: 'MinFragsPerTransfrag',15

Data Types: double

**MinIntronLength** — Minimum number of base pairs for intron in genome

50 (default) | positive integer

Minimum number of base pairs for an intron in the genome, specified as a positive integer.
Example: 'MinIntronLength',50

Data Types: double

**MinIsoformFraction** — Cuffoff value to report abundance of isoform

0.1 (default) | scalar between 0 and 1

Cuffoff value to report the abundance of a particular isoform as a fraction of the most abundant isoform (major isoform), specified as a scalar between 0 and 1. The function filters out transcripts with abundances below the specified value because isoforms expressed at low levels often cannot be assembled reliably. The default value is 0.1, or 10%, of the major isoform of the gene.
Example: 'MinIsoformFraction',0.20

Data Types: double

**MultiReadCorrection** — Flag to improve abundance estimation using rescue method

false (default) | true

Flag to improve abundance estimation for reads mapped to multiple genomic positions using the rescue method, specified as true or false. If the value is false, the function divides multimapped reads uniformly to all mapped positions. If the value is true, the function uses additional information, including gene abundance estimation, inferred fragment length, and fragment bias, to improve transcript abundance estimation.

The rescue method is described in [2].
Example: true

Data Types: logical
NormalizeCompatibleHits — Flag to use only fragments compatible with reference transcript to calculate FPKM values
false (default) | true

Flag to use only fragments compatible with a reference transcript to calculate FPKM values, specified as true or false.
Example: 'NormalizeCompatibleHits',false
Data Types: logical

NormalizeTotalHits — Flag to include all fragments to calculate FPKM values
false (default) | true

Flag to include all fragments to calculate FPKM values, specified as true or false. If the value is true, the function includes all fragments, including fragments without a compatible reference.
Example: 'NormalizeTotalHits',true
Data Types: logical

NumFragAssignmentDraws — Number of fragment assignments to perform on each transcript
50 (default) | positive integer

Number of fragment assignments to perform on each transcript, specified as a positive integer. For each fragment drawn from a transcript, the function performs the specified number of assignments probabilistically to determine the transcript assignment uncertainty and to estimate the variance-covariance matrix for the assigned fragment counts.
Example: 'NumFragAssignmentSamples',40
Data Types: double

NumFragDraws — Number of draws from negative binomial random number generator
100 (default) | positive integer

Number of draws from the negative binomial random number generator for each transcript, specified as a positive integer. Each draw is a number of fragments that the function probabilistically assigns to transcripts in the transcriptome to determine the assignment uncertainty and to estimate the variance-covariance matrix for assigned fragment counts.
Example: 'NumFragSamples',90
Data Types: double

NumThreads — Number of parallel threads to use
1 (default) | positive integer

Number of parallel threads to use, specified as a positive integer. Threads are run on separate processors or cores. Increasing the number of threads generally improves the runtime significantly, but increases the memory footprint.
Example: 'NumThreads',4
Data Types: double

OutputDirectory — Directory to store analysis results
current directory ("./") (default) | string | character vector
Directory to store analysis results, specified as a string or character vector.
Example: 'OutputDirectory', './AnalysisResults/
Data Types: char | string

OverhangTolerance — Number of base pairs of overlap with intron
8 (default) | positive integer

Number of base pairs of overlap with an intron that the function allows when determining if the read is compatible with another transcript, specified as a positive integer.
Example: 'OverhangTolerance', 5
Data Types: double

RABTOverhangTolerance3 — Number of base pairs allowed to overhang 3' end of reference transcript
600 (default) | positive integer

Number of base pairs allowed to overhang the 3' end of each reference transcript during the RABT assembly, specified as a positive integer. The function uses this property when deciding if an assembled transcript is novel or should be merged with the reference.

Note: The function only performs the RABT assembly if you specify GTFGuide. Otherwise, RABTOverhangTolerance3 has no effect on the assembled transcript.
Example: 'OverhangTolerance3', 500
Data Types: double

OverlapRadius — Distance between transfags
50 (default) | positive integer

Distance between transfags, specified as a positive integer. If the distance is below the specified value, the function merges the transfags. The default value is 50 base pairs.
Example: 'OverlapRadius', 40
Data Types: double

PreMRNAFraction — Threshold to include alignments in intronic intervals
0.15 (default) | scalar between 0 and 1

Threshold to include alignments in intronic intervals in the assembly, specified as a scalar between 0 and 1. The function ignores the intronic alignments if the minimum depth of coverage divided by the number of spliced reads is below the specified value. Use this property to filter reads originating from incompletely spliced transcripts.
Example: 'PreMRNAFraction', 0.10
Data Types: double

ReferenceGTF — Name of GTF or GFF file containing reference annotation
string | character vector

Name of a GTF or GFF file containing reference annotation used to estimate isoform expression, specified as a string or character vector. If you provide a ReferenceGTF file, the function does not
assemble any novel transcripts and ignores any alignments incompatible with the reference transcripts.
Example: 'ReferenceGTF','isoest.gtf'
Data Types: char | string

**Seed — Seed for random number generator**

0 (default) | nonnegative integer

Seed for the random number generator, specified as a nonnegative integer. Setting a seed value ensures the reproducibility of the analysis results.
Example: 'Seed',10
Data Types: double

**SmallAnchorFraction — Minimum percentage of alignment on each side of splice junction**

0.09 (default) | scalar between 0 and 1

Minimum percentage of alignment on each side of a splice junction, specified as a scalar between 0 and 1. The function filters alignments with a percentage smaller than this property value prior to assembly.
Example: 'SmallAnchorFraction',0.1
Data Types: double

**TranscriptPrefix — Prefix for reported transfrags in output GTF file**

"CUFF" (default) | string | character vector

Prefix for the reported transfrags in the output GTF file, specified as a string or character vector. This option must be a string or character vector with a non-zero length.
Example: 'TranscriptPrefix','tfrags'
Data Types: char | string

**TrimCoverageThreshold — Minimum average coverage needed for 3' trimming**

10 (default) | positive integer

Minimum average coverage for 3' trimming, specified as a positive integer.
Example: 'TrimCoverageThreshold',8
Data Types: double

**TrimDropoffFraction — Minimum percentage of average coverage**

0.1 (default) | scalar between 0 and 1

Minimum percentage of average coverage for trimming the 3' end of assembled transcripts, specified as a scalar between 0 and 1.
Example: 'TrimDropoffFraction',0.15
Data Types: double
Output Arguments

transcripts — Transcript file name
"./transcripts.gtf" (default)

Transcript file name, returned as a string. The name of the file is "transcripts.gtf". The file contains the assembled isoforms, along with attributes describing the abundance of reads originating from each transcript.

The output string also includes the directory information defined by OutputDirectory. The default is the current directory. If you set OutputDirectory to "/local/tmp/", the output becomes "/local/tmp/transcripts.gtf".

isoforms — Estimated isoform-level expression file name
"./isoforms.fpkm_tracking.gtf" (default)

Estimated isoform-level expression file name, returned as a string. By default, the name of the file is "isoforms.fpkm_tracking.gtf". The file contains estimates for isoform-level expression in cufflinks FPKM tracking format.

The output string also includes the directory information defined by OutputDirectory. The default is the current directory. If you set OutputDirectory to "/local/tmp/", the output becomes "/local/tmp/isoforms.fpkm_tracking.gtf".

genes — Estimated gene-level expression file name
"./genes.fpkm_tracking.gtf" (default)

Estimated gene-level expression file name, returned as a string. By default, the name of the file is "genes.fpkm_tracking.gtf". The file contains estimates for gene-level expression in cufflinks FPKM tracking format.

The output string also includes the directory information defined by OutputDirectory. The default is the current directory. If you set OutputDirectory to "/local/tmp/", the output becomes "/local/tmp/genes.fpkm_tracking.gtf".

skippedTranscripts — Name of file containing skipped transcripts
"./skipped.gtf" (default)

Name of the file containing skipped transcripts when processing a locus, returned as a string. By default, the name of the file is "skipped.gtf". The 'MaxBundleFrags' option specifies the maximum number of transcripts (fragments) to include for each locus. After reaching the threshold, the function puts the skipped fragments in this file.

The output string also includes the directory information defined by OutputDirectory. The default is the current directory. If you set OutputDirectory to "/local/tmp/", the output becomes "/local/tmp/skipped.gtf".

References


**See Also**
CufflinksOptions | bowtie2 | cuffcompare | cuffdiff | cuffgffread | cuffgtf2sam | cuffmerge | cuffnorm | cuffquant

**External Websites**
Cufflinks manual

**Introduced in R2019a**
CufflinksOptions

Option set for cufflinks

Description

A CufflinksOptions object contains options for the cufflinks function, which assembles a transcriptome from aligned reads [1].

Creation

Syntax

cufflinksOpt = CufflinksOptions

cufflinksOpt = CufflinksOptions(Name,Value)
cufflinksOpt = CufflinksOptions(S)

Description

cufflinksOpt = CufflinksOptions creates a CufflinksOptions object with the default property values.

CufflinksOptions requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

Note  CufflinksOptions is supported on the Mac and UNIX platforms only.

cufflinksOpt = CufflinksOptions(Name,Value) sets the object properties on page 1-482 using one or more name-value pair arguments. Enclose each property name in quotes. For example, cufflinksOpt = CufflinksOptions('TrimCoverageThreshold',5) specifies the minimum average coverage for 3' end trimming.

cufflinksOpt = CufflinksOptions(S) specifies optional parameters using a string or character vector S.

Input Arguments

S — Cufflinks options
character vector | string

Cufflinks options, specified as a character vector or string. S must be in the Cufflinks option syntax (prefixed by one or two dashes).

Example: '--trim-3-avgcov-thresh 5'
Properties

EffectiveLengthCorrection — Flag to normalize fragment counts
true (default) | false

Flag to normalize fragment counts to fragments per kilobase per million mapped reads (FPKM), specified as true or false.
Example: false
Data Types: logical

ExtraCommand — Additional commands
"" (default) | character vector | string

Additional commands, specified as a string or character vector. The commands must be in the original syntax (prefixed by one or two dashes). Use this option to apply undocumented flags and flags without corresponding MATLAB properties. When the function converts the original flags to MATLAB properties, it stores any unrecognized flags in this option.
Example: '--library-type fr-secondstrand'
Data Types: char | string

FauxReadTiling — Flag to include reference transcripts in assembled output
ture (default) | false

Flag to include reference transcripts in the assembled output as faux-reads during RABT (advanced reference annotation based transcript) assembly, specified as true or false.

Note The function only performs the RABT assembly if you specify GTFGuide. Otherwise, FauxReadTiling, regardless of being true or false, has no effect on the assembled transcript.

Example: false
Data Types: logical

FragmentBiasCorrection — Name of FASTA file with reference transcripts to detect bias
string | character vector

Name of the FASTA file with reference transcripts to detect bias in fragment counts, specified as a string or character vector. Library preparation can introduce sequence-specific bias into RNA-Seq experiments. Providing reference transcripts improves the accuracy of the transcript abundance estimates.
Example: "bias.fasta"
Data Types: char | string

FragmentLengthMean — Expected mean fragment length
200 (default) | positive integer

Expected mean fragment length, specified as a positive integer. The default value is 200 base pairs. The function can learn the fragment length mean for each SAM file. Using this option is not recommended for paired-end reads.
Example: 100
Data Types: double

**FragmentLengthSD — Expected standard deviation for fragment length distribution**
80 (default) | positive scalar

Expected standard deviation for the fragment length distribution, specified as a positive scalar. The default value is 80 base pairs. The function can learn the fragment length standard deviation for each SAM file. Using this option is not recommended for paired-end reads.

Example: 70
Data Types: double

**GTFGuide — Name of GTF file to guide RABT assembly**
string | character vector

Name of a GTF file to guide the RABT assembly, specified as a string or character vector.

Example: 'tr.gtf'
Data Types: char | string

**IncludeAll — Flag to use all object properties**
false (default) | true

Flag to include all the object properties with the corresponding default values when converting to the original options syntax, specified as true or false. You can convert the properties to the original syntax prefixed by one or two dashes (such as ' -d 100 -e 80 ') by using getCommand. The default value false means that when you call getCommand(optionsObject), it converts only the specified properties. If the value is true, getCommand converts all available properties, with default values for unspecified properties, to the original syntax.

Example: true
Data Types: logical

**JunctionAlpha — Alpha value in binomial test to filter false-positive alignments**
0.001 (default) | scalar between 0 and 1

Alpha value in the binomial test to filter false-positive alignments, specified as a scalar between 0 and 1.

Example: 0.005
Data Types: double

**LengthCorrection — Flag to correct by transcript length**
true (default) | false

Flag to correct by the transcript length, specified as true or false. Set this value to false only when the fragment count is independent of the feature size, such as for small RNA libraries with no fragmentation and for 3' end sequencing, where all fragments have the same length.

Example: false
Data Types: logical
**MaskFile** — Name of GTF or GFF file containing transcripts to ignore
string | character vector

Name of the GTF or GFF file containing transcripts to ignore during analysis, specified as a string or character vector. Some examples of transcripts to ignore include annotated rRNA transcripts, mitochondrial transcripts, and other abundant transcripts. Ignoring these transcripts improves the robustness of the abundance estimates.

Example: ‘excludes.gtf’
Data Types: char | string

**MaxBundleFrags** — Maximum number of fragments to include for each locus before skipping
500000 (default) | positive integer

Maximum number of fragments to include for each locus before skipping new fragments, specified as a positive integer. Skipped fragments are marked with the status HIDATA in the file skipped.gtf.

Example: 400000
Data Types: double

**MaxBundleLength** — Maximum genomic length in base pairs for bundle
3500000 (default) | positive integer

Maximum genomic length in base pairs for a bundle, specified as a positive integer.

Example: 3400000
Data Types: double

**MaxFragAlignments** — Maximum number of aligned reads to include for each fragment
Inf (default) | positive integer

Maximum number of aligned reads to include for each fragment before skipping new reads, specified as a positive integer. Inf, the default value, sets no limit on the maximum number of aligned reads.

Example: 1000
Data Types: double

**MaxIntronLength** — Maximum number of bases in intron
300000 (default) | positive integer

Maximum number of bases in an intron to report, specified as a positive integer. cufflinks also ignores SAM alignments with REF_SKIP CIGAR operations longer than this property value.

Example: 350000
Data Types: double

**MaxMLEIterations** — Maximum number of iterations for maximum likelihood estimation
5000 (default) | positive integer

Maximum number of iterations for the maximum likelihood estimation of abundances, specified as a positive integer.

Example: 4000
Data Types: double

**MinFragsPerTransfrag** — Minimum number of aligned RNA-Seq fragments to report
10 (default) | positive integer

Minimum number of aligned RNA-Seq fragments to report on an assembled transfrag, specified as a positive integer.
Example: 15
Data Types: double

**MinIntronLength** — Minimum number of base pairs for intron in genome
50 (default) | positive integer

Minimum number of base pairs for an intron in the genome, specified as a positive integer.
Example: 50
Data Types: double

**MinIsoformFraction** — Cuffoff value to report abundance of isoform
0.1 (default) | scalar between 0 and 1

Cuffoff value to report the abundance of a particular isoform as a fraction of the most abundant isoform (major isoform), specified as a scalar between 0 and 1. The function filters out transcripts with abundances below the specified value because isoforms expressed at low levels often cannot be assembled reliably. The default value is 0.1, or 10%, of the major isoform of the gene.
Example: 0.20
Data Types: double

**MultiReadCorrection** — Flag to improve abundance estimation using rescue method
false (default) | true

Flag to improve abundance estimation for reads mapped to multiple genomic positions using the rescue method, specified as true or false. If the value is false, the function divides multimapped reads uniformly to all mapped positions. If the value is true, the function uses additional information, including gene abundance estimation, inferred fragment length, and fragment bias, to improve transcript abundance estimation.

The rescue method is described in [2].
Example: true
Data Types: logical

**NormalizeCompatibleHits** — Flag to use only fragments compatible with reference transcript to calculate FPKM values
false (default) | true

Flag to use only fragments compatible with a reference transcript to calculate FPKM values, specified as true or false.
Example: true
Data Types: logical
**NormalizeTotalHits — Flag to include all fragments to calculate FPKM values**

- **false** (default) | **true**

Flag to include all fragments to calculate FPKM values, specified as **true** or **false**. If the value is **true**, the function includes all fragments, including fragments without a compatible reference.

Example: **true**

Data Types: **logical**

**NumFragAssignmentDraws — Number of fragment assignments to perform on each transcript**

- **50** (default) | **positive integer**

Number of fragment assignments to perform on each transcript, specified as a positive integer. For each fragment drawn from a transcript, the function performs the specified number of assignments probabilistically to determine the transcript assignment uncertainty and to estimate the variance-covariance matrix for the assigned fragment counts.

Example: **40**

Data Types: **double**

**NumFragDraws — Number of draws from negative binomial random number generator**

- **100** (default) | **positive integer**

Number of draws from the negative binomial random number generator for each transcript, specified as a positive integer. Each draw is a number of fragments that the function probabilistically assigns to transcripts in the transcriptome to determine the assignment uncertainty and to estimate the variance-covariance matrix for assigned fragment counts.

Example: **90**

Data Types: **double**

**NumThreads — Number of parallel threads to use**

- **1** (default) | **positive integer**

Number of parallel threads to use, specified as a positive integer. Threads are run on separate processors or cores. Increasing the number of threads generally improves the runtime significantly, but increases the memory footprint.

Example: **4**

Data Types: **double**

**OutputDirectory — Directory to store analysis results**

- **current directory ("./") (default) | string | character vector**

Directory to store analysis results, specified as a string or character vector.

Example: **"./AnalysisResults/"**

Data Types: **char** | **string**

**OverhangTolerance — Number of base pairs of overlap with intron**

- **8** (default) | **positive integer**

Number of base pairs of overlap with an intron that the function allows when determining if the read is compatible with another transcript, specified as a positive integer.
Example: 5
Data Types: double

**OverlapRadius — Distance between transfrags**
50 (default) | positive integer

Distance between transfrags, specified as a positive integer. If the distance is below the specified value, the function merges the transfrags. The default value is 50 base pairs.

Example: 40
Data Types: double

**PreMRNAFraction — Threshold to include alignments in intronic intervals**
0.15 (default) | scalar between 0 and 1

Threshold to include alignments in intronic intervals in the assembly, specified as a scalar between 0 and 1. The function ignores the intronic alignments if the minimum depth of coverage divided by the number of spliced reads is below the specified value. Use this property to filter reads originating from incompletely spliced transcripts.

Example: 0.10
Data Types: double

**RABTOverhangTolerance — Number of base pairs allowed to overlap with transcript intron**
8 (default) | positive integer

Number of base pairs from a read allowed to overlap with a transcript intron when determining if a read is mappable to another transcript during the RABT assembly, specified as a positive integer. The default value is 8.

*Note* The function only performs the RABT assembly if you specify GTFGuide. Otherwise, RABTOverhangTolerance has no effect on the assembled transcript.

Example: 10
Data Types: double

**RABTOverhangTolerance3 — Number of base pairs allowed to overhang 3' end of reference transcript**
600 (default) | positive integer

Number of base pairs allowed to overhang the 3' end of each reference transcript during the RABT assembly, specified as a positive integer. The function uses this property when deciding if an assembled transcript is novel or should be merged with the reference.

*Note* The function only performs the RABT assembly if you specify GTFGuide. Otherwise, RABTOverhangTolerance3 has no effect on the assembled transcript.

Example: 500
Data Types: double
**ReferenceGTF — Name of GTF or GFF file used to estimate isoform expression**

*string | character vector*

Name of a GTF or GFF file containing reference annotation used to estimate isoform expression, specified as a string or character vector. If you provide a ReferenceGTF file, the function does not assemble any novel transcripts and ignores any alignments incompatible with the reference transcripts.

Example: 'isoest.gtf'

Data Types: char | string

**Seed — Seed for random number generator**

*0 (default) | nonnegative integer*

Seed for the random number generator, specified as a nonnegative integer. Setting a seed value ensures the reproducibility of the analysis results.

Example: 10

Data Types: double

**SmallAnchorFraction — Minimum percentage of alignment on each side of splice junction**

*0.09 (default) | scalar between 0 and 1*

Minimum percentage of alignment on each side of a splice junction, specified as a scalar between 0 and 1. The function filters alignments with a percentage smaller than this property value prior to assembly.

Example: 0.1

Data Types: double

**TranscriptPrefix — Prefix for reported transfrags in output GTF file**

"CUFF" (default) | string | character vector

Prefix for the reported transfrags in the output GTF file, specified as a string or character vector. This option must be a string or character vector with a non-zero length.

Example: "tfrags"

Data Types: char | string

**TrimCoverageThreshold — Minimum average coverage needed for 3' trimming**

*10 (default) | positive integer*

Minimum average coverage for 3' trimming, specified as a positive integer.

Example: 8

Data Types: double

**TrimDropoffFraction — Minimum percentage of average coverage**

*0.1 (default) | scalar between 0 and 1*

Minimum percentage of average coverage for trimming the 3' end of assembled transcripts, specified as a scalar between 0 and 1.

Example: 0.15
Data Types: double

**Version — Supported version**

string

This property is read-only.

Supported version of the original cufflinks software, returned as a string.

Example: "2.2.1"

Data Types: string

**Object Functions**

getCommand  Translate object properties to original options syntax
getOptionsTable  Return table with all properties and equivalent options in original syntax

**Examples**

**Create CufflinksOptions**

Create a CufflinksOptions object with default values.

```
opt = CufflinksOptions;
```

Create an object using name-value pairs.

```
opt2 = CufflinksOptions('TranscriptPrefix','MATLAB','NumThreads',4)
```

Create an object by using the original cufflinks syntax.

```
opt3 = CufflinksOptions('--label MATLAB --num-threads 4')
```

**Assemble Transcriptome and Perform Differential Expression Testing**

Create a CufflinksOptions object to define cufflinks options, such as the number of parallel threads and the output directory to store the results.

```
cflOpt = CufflinksOptions;
cflOpt.NumThreads = 8;
cflOpt.OutputDirectory = './cufflinksOut';
```

The SAM files provided for this example contain aligned reads for *Mycoplasma pneumoniae* from two samples with three replicates each. The reads are simulated 100bp-reads for two genes (gyrA and gyrB) located next to each other on the genome. All the reads are sorted by reference position, as required by cufflinks.

```
sams = ['Myco_1_1.sam','Myco_1_2.sam','Myco_1_3.sam',...
    'Myco_2_1.sam', 'Myco_2_2.sam', 'Myco_2_3.sam'];
```

Assemble the transcriptome from the aligned reads.

```
[gtfs,isofpkm,genes,skipped] = cufflinks(sams,cflOpt);
```
gtfs is a list of GTF files that contain assembled isoforms. Compare the assembled isoforms using cuffcompare.

\[
\text{stats} = \text{cuffcompare(gtfs)};
\]

Merge the assembled transcripts using cuffmerge.

\[
\text{mergedGTF} = \text{cuffmerge(gtfs,}'\text{OutputDirectory}','.\text{/cuffMergeOutput}'\text{)};
\]

mergedGTF reports only one transcript. This is because the two genes of interest are located next to each other, and cuffmerge cannot distinguish two distinct genes. To guide cuffmerge, use a reference GTF (gyrAB.gtf) containing information about these two genes. If the file is not located in the same directory that you run cuffmerge from, you must also specify the file path.

\[
\text{gyrAB} = \text{which('gyrAB.gtf')};
\]

\[
\text{mergedGTF2} = \text{cuffmerge(gtfs,}'\text{OutputDirectory}','.\text{/cuffMergeOutput2}'\text{,...
'\text{ReferenceGTF}',gyrAB)};
\]

Calculate abundances (expression levels) from aligned reads for each sample.

\[
\text{abundances1} = \text{cuffquant(mergedGTF2,['Myco_1_1.sam','Myco_1_2.sam','Myco_1_3.sam'],...
'\text{OutputDirectory}','.\text{/cuffquantOutput1}')};
\]

\[
\text{abundances2} = \text{cuffquant(mergedGTF2,['Myco_2_1.sam','Myco_2_2.sam','Myco_2_3.sam'],...
'\text{OutputDirectory}','.\text{/cuffquantOutput2}')};
\]

Assess the significance of changes in expression for genes and transcripts between conditions by performing the differential testing using cuffdiff. The cuffdiff function operates in two distinct steps: the function first estimates abundances from aligned reads, and then performs the statistical analysis. In some cases (for example, distributing computing load across multiple workers), performing the two steps separately is desirable. After performing the first step with cuffquant, you can then use the binary CXB output file as an input to cuffdiff to perform statistical analysis. Because cuffdiff returns several files, specify the output directory is recommended.

\[
\text{isoformDiff} = \text{cuffdiff(mergedGTF2,[abundances1,abundances2],...
'\text{OutputDirectory}','.\text{/cuffdiffOutput}')};
\]

Display a table containing the differential expression test results for the two genes gyrB and gyrA.

\[
\text{readtable(isoformDiff,'FileType','text')}
\]

\[
\text{ans} = \\
2\times14 \text{ table}
\]

\[
\begin{align*}
\text{test_id} & \quad \text{gene_id} & \quad \text{gene} & \quad \text{locus} & \quad \text{sample_1} & \quad \text{sample_2} & \quad \text{status} & \quad \text{value_1} & \quad \text{value_2} & \quad \text{log2\_fold\_change} & \quad \text{test\_stat} & \quad \text{p\_value} & \quad \text{q\_value} & \quad \text{significant} \\
\text{TCONS_00000001} & \quad \text{XLOC_000001} & \quad \text{gyrB} & \quad \text{NC_000912.1:2868-7340} & \quad \text{q1} & \quad \text{q2} & \quad \text{OK} & \quad 1.0913e+05 & \quad 4.2228e+05 & \quad 1.9522 & \quad 7.8886 & \quad 5e-05 & \quad 5e-05 & \quad \text{yes} \\
\text{TCONS_00000002} & \quad \text{XLOC_000001} & \quad \text{gyrA} & \quad \text{NC_000912.1:2868-7340} & \quad \text{q1} & \quad \text{q2} & \quad \text{OK} & \quad 3.5158e+05 & \quad 1.1546e+05 & \quad -1.6064 & \quad -7.3811 & \quad 5e-05 & \quad 5e-05 & \quad \text{yes}
\end{align*}
\]

You can use cuffnorm to generate normalized expression tables for further analyses. cuffnorm results are useful when you have many samples and you want to cluster them or plot expression levels for genes that are important in your study. Note that you cannot perform differential expression analysis using cuffnorm.

Specify a cell array, where each element is a string vector containing file names for a single sample with replicates.
alignmentFiles = {{"Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam"},...
["Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"]}
isoformNorm = cuffnorm(mergedGTF2, alignmentFiles,...
'OutputDirectory', './cuffnormOutput');

Display a table containing the normalized expression levels for each transcript.

readtable(isoformNorm,'FileType','text')

ans =
2×7 table

<table>
<thead>
<tr>
<th>tracking_id</th>
<th>q1_0</th>
<th>q1_2</th>
<th>q1_1</th>
<th>q2_1</th>
<th>q2_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>'TCONS_00000001'</td>
<td>1.0913e+05</td>
<td>78628</td>
<td>1.2132e+05</td>
<td>4.3639e+05</td>
<td>4.2228e+05</td>
</tr>
<tr>
<td>'TCONS_00000002'</td>
<td>3.5158e+05</td>
<td>3.7458e+05</td>
<td>3.4238e+05</td>
<td>1.0483e+05</td>
<td>1.1546e+05</td>
</tr>
</tbody>
</table>

Column names starting with q have the format: conditionX_N, indicating that the column contains values for replicate N of conditionX.

References


See Also
cuffcompare | cuffdiff | cuggffread | cuffgtf2sam | cufflinks | cuffmerge | cuffnorm | cuffquant

External Websites
Cufflinks manual

Introduced in R2019a
cuffmerge

Merge RNA-seq assemblies into a master transcriptome

**Syntax**

mergedGTF = cuffmerge(gtfFiles)
mergedGTF = cuffmerge(gtfFiles,opt)
mergedGTF = cuffmerge(gtfFiles,Name,Value)

**Description**

mergedGTF = cuffmerge(gtfFiles) merges assembled transcriptome from two or more GTF files [1]. Merging GTF files is a required step to perform the downstream differential analysis with cuffdiff.

cuffmerge requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

**Note** cuffmerge is supported on the Mac and UNIX platforms only.

mergedGTF = cuffmerge(gtfFiles,opt) uses additional options specified by opt.

mergedGTF = cuffmerge(gtfFiles,Name,Value) uses additional options specified by one or more name-value pair arguments. For example,
cuffmerge(["Myco_1_1.transcripts.gtf","Myco_1_2.transcripts.gtf"],'NumThreads',5) specifies to use five parallel threads.

**Examples**

**Assemble Transcriptome and Perform Differential Expression Testing**

Create a CufflinksOptions object to define cufflinks options, such as the number of parallel threads and the output directory to store the results.

cflOpt = CufflinksOptions;
cflOpt.NumThreads = 8;
cflOpt.OutputDirectory = "./cufflinksOut";

The SAM files provided for this example contain aligned reads for *Mycoplasma pneumoniae* from two samples with three replicates each. The reads are simulated 100bp-reads for two genes (gyrA and gyrB) located next to each other on the genome. All the reads are sorted by reference position, as required by cufflinks.

sams = ["Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam",...
       "Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"];

Assemble the transcriptome from the aligned reads.
gtfs, isofpkm, genes, skipped = cufflinks(sams, cflOpt);

gtfs is a list of GTF files that contain assembled isoforms.

Compare the assembled isoforms using cuffcompare.

stats = cuffcompare(gtfs);

Merge the assembled transcripts using cuffmerge.

mergedGTF = cuffmerge(gtfs,'OutputDirectory','./cuffMergeOutput');

mergedGTF reports only one transcript. This is because the two genes of interest are located next to each other, and cuffmerge cannot distinguish two distinct genes. To guide cuffmerge, use a reference GTF (gyrAB.gtf) containing information about these two genes. If the file is not located in the same directory that you run cuffmerge from, you must also specify the file path.

gyrAB = which('gyrAB.gtf');
mergedGTF2 = cuffmerge(gtfs,'OutputDirectory','./cuffMergeOutput2',...
'ReferenceGTF',gyrAB);

Calculate abundances (expression levels) from aligned reads for each sample.

abundances1 = cuffquant(mergedGTF2,["Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam"],...
'OutputDirectory','./cuffquantOutput1');
abundances2 = cuffquant(mergedGTF2,["Myco_2_1.sam","Myco_2_2.sam","Myco_2_3.sam"],...
'OutputDirectory','./cuffquantOutput2');

Assess the significance of changes in expression for genes and transcripts between conditions by performing the differential testing using cuffdiff. The cuffdiff function operates in two distinct steps: the function first estimates abundances from aligned reads, and then performs the statistical analysis. In some cases (for example, distributing computing load across multiple workers), performing the two steps separately is desirable. After performing the first step with cuffquant, you can then use the binary CXB output file as an input to cuffdiff to perform statistical analysis. Because cuffdiff returns several files, specify the output directory is recommended.

isoformDiff = cuffdiff(mergedGTF2,[abundances1,abundances2],...
'OutputDirectory','./cuffdiffOutput');

Display a table containing the differential expression test results for the two genes gyrB and gyrA.

readtable(isoformDiff,'FileType','text')

ans =

2x14 table

test_id gene_id gene locus sample_1 sample_2

'TCONS_00000001' 'XLOC_000001' 'gyrB' 'NC_000912.1:2868-7340' 'q1' 'q2'
'TCONS_00000002' 'XLOC_000001' 'gyrA' 'NC_000912.1:2868-7340' 'q1' 'q2'

You can use cuffnorm to generate normalized expression tables for further analyses. cuffnorm results are useful when you have many samples and you want to cluster them or plot expression levels for genes that are important in your study. Note that you cannot perform differential expression analysis using cuffnorm.
Specify a cell array, where each element is a string vector containing file names for a single sample with replicates.

alignmentFiles = {{"Myco_1_1.sam", "Myco_1_2.sam", "Myco_1_3.sam"},
    {"Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"}}

isoformNorm = cuffnorm(mergedGTF2, alignmentFiles,
    'OutputDirectory', './cuffnormOutput');

Display a table containing the normalized expression levels for each transcript.

readtable(isoformNorm,'FileType','text')

ans =

2×7 table

<table>
<thead>
<tr>
<th>tracking_id</th>
<th>q1_0</th>
<th>q1_2</th>
<th>q1_1</th>
<th>q2_1</th>
<th>q2_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>'TCONS_00000001'</td>
<td>1.0913e+05</td>
<td>78628</td>
<td>1.2132e+05</td>
<td>4.3639e+05</td>
<td>4.2228e+05</td>
</tr>
<tr>
<td>'TCONS_00000002'</td>
<td>3.5158e+05</td>
<td>3.7458e+05</td>
<td>3.4238e+05</td>
<td>1.0483e+05</td>
<td>1.1546e+05</td>
</tr>
</tbody>
</table>

Column names starting with *q* have the format: *conditionX_N*, indicating that the column contains values for replicate *N* of *conditionX*.

**Input Arguments**

*gtfFiles* — Names of GTF files

string vector | cell array of character vectors

Names of GTF files, specified as a string vector or cell array of character vectors.

Example: ["Myco_1_1.transcripts.gtf", "Myco_1_2.transcripts.gtf"]

Data Types: string | cell

*opt* — cuffgffread options

CuffMergeOptions object | string | character vector

cuffgffread options, specified as a CuffMergeOptions object, string, or character vector. The string or character vector must be in the original cuffmerge option syntax (prefixed by one or two dashes) [1].

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of *Name*, *Value* arguments. *Name* is the argument name and *Value* is the corresponding value. *Name* must appear inside quotes. You can specify several name and value pair arguments in any order as *Name1*, *Value1*,...,*NameN*, *ValueN*.

Example:
cuffmerge(["Myco_1_1.transcripts.gtf","Myco_1_2.transcripts.gtf"], 'NumThreads', 5)

*ExtraCommand* — Additional commands

"" (default) | string | character vector

Additional commands, specified as a string or character vector. The commands must be in the original syntax (prefixed by one or two dashes). Use this option to apply undocumented flags and flags
without corresponding MATLAB properties. When the function converts the original flags to MATLAB properties, it stores any unrecognized flags in this option.

Example: `'ExtraCommand','--library-type fr-secondstrand'`

Data Types: char | string

**IncludeAll — Flag to apply all available options**

false (default) | true

Flag to include all available options with the corresponding default values when converting to the original options syntax, specified as true or false. The original syntax is prefixed by one or two dashes, such as `-d 100 -e 80`. By default, the function converts only the specified options. If the value is true, the function converts all available options, with default values for unspecified options, to the original syntax.

Example: `'IncludeAll',true`

Data Types: logical

**MinIsoformFraction — Minimum abundance of isoform to be included in merged assembly**

0.5 (default) | scalar between 0 and 1

Minimum abundance of an isoform to be included in the merged assembly, specified as a scalar between 0 and 1. This value is expressed as a percentage of the most abundant (major) isoform.

Example: `'MinIsoformFraction',0.4`

Data Types: double

**NumThreads — Number of parallel threads to use**

1 (default) | positive integer

Number of parallel threads to use, specified as a positive integer. Threads are run on separate processors or cores. Increasing the number of threads generally improves the runtime significantly, but increases the memory footprint.

Example: `'NumThreads',4`

Data Types: double

**OutputDirectory — Directory to store analysis results**

current directory ("./") (default) | string | character vector

Directory to store analysis results, specified as a string or character vector.

Example: `'OutputDirectory',"./AnalysisResults/```

Data Types: char | string

**ReferenceGTF — Name of optional reference annotation GTF file**

string | character vector

Name of an optional reference annotation GTF file to be included in the combined assembly, specified as a string or character vector.

Example: `'ReferenceGTF','ref.gtf`

Data Types: char | string
**ReferenceSequence** — Name of directory or FASTA file containing genomic sequences  
string | character vector

Name of a directory or FASTA file containing genomic DNA sequences for the reference, specified as a string or character vector.

- If you specify a directory, it must contain one FASTA file per contig. In other words, the directory must contain one FASTA file per reference chromosome, and each file must be named after the chromosome and have a `.fa` or `.fasta` extension.
- If you specify a FASTA file, it must contain all the reference sequences.

The function uses the provided sequences to improve transfrag classification and exclude artifacts.

Example: 'ReferenceSequence', "allrefs.fasta"

Data Types: char | string

**Output Arguments**

mergedGTF — Name of output GTF file

"./merged_asm/merged.gtf"

Name of the output GTF file containing the merged transcriptome, returned as a string.

The output string also includes the directory information defined by OutputDirectory. By default, the function

- Creates the `merged_asm` subfolder in the current directory and saves the output file (merged.gtf) in that folder.
- Creates a subfolder named `logs` inside `merged_asm` folder and saves a log file.

If you set OutputDirectory to "/local/tmp/", mergedGTF becomes "/local/tmp/merged.gtf". The function also creates the `logs` folder inside the specified output directory.

**References**


**See Also**

CuffMergeOptions | cufflinks

**External Websites**

Cufflinks manual

**Introduced in R2019a**
CuffMergeOptions

Option set for cuffmerge

Description

A CuffMergeOptions object contains options for the cuffmerge function, which merges cufflinks transcript assemblies [1].

Creation

Syntax

cuffmergeOpt = CuffMergeOptions
cuffmergeOpt = CuffMergeOptions(Name,Value)
cuffmergeOpt = CuffMergeOptions(S)

Description

cuffmergeOpt = CuffMergeOptions creates a CuffMergeOptions object with the default property values.

CuffMergeOptions requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

Note CuffMergeOptions is supported on the Mac and UNIX platforms only.

cuffmergeOpt = CuffMergeOptions(Name,Value) sets the object properties on page 1-497 using one or more name-value pair arguments. Enclose each property name in quotes. For example, cuffmergeOpt = CuffMergeOptions('NumThreads',8) specifies eight parallel threads.

cuffmergeOpt = CuffMergeOptions(S) specifies optional parameters using a string or character vector S.

Input Arguments

S — cuffmerge options
string | character vector

cuffmerge options, specified as a string or character vector. S must be in the original cuffmerge option syntax (prefixed by one or two dashes).

Example: '--num-thread 5'

Properties

ExtraCommand — Additional commands

"" (default) | string | character vector

""
Additional commands, specified as a string or character vector. The commands must be in the original syntax (prefixed by one or two dashes). Use this option to apply undocumented flags and flags without corresponding MATLAB properties. When the function converts the original flags to MATLAB properties, it stores any unrecognized flags in this option.

Example: `'-library-type fr-secondstrand'`

Data Types: char | string

IncludeAll — Flag to use all object properties
false (default) | true

Flag to include all the object properties with the corresponding default values when converting to the original options syntax, specified as true or false. You can convert the properties to the original syntax prefixed by one or two dashes (such as `'-d 100 -e 80'`) by using `getCommand`. The default value false means that when you call `getCommand(optionsObject)`, it converts only the specified properties. If the value is true, `getCommand` converts all available properties, with default values for unspecified properties, to the original syntax.

Example: true

Data Types: logical

MinIsoformFraction — Minimum abundance of isoform to be included in merged assembly
0.5 (default) | scalar between 0 and 1

Minimum abundance of an isoform to be included in the merged assembly, specified as a scalar between 0 and 1. This value is expressed as a percentage of the most abundant (major) isoform.

Example: 0.4

Data Types: double

NumThreads — Number of parallel threads to use
1 (default) | positive integer

Number of parallel threads to use, specified as a positive integer. Threads are run on separate processors or cores. Increasing the number of threads generally improves the runtime significantly, but increases the memory footprint.

Example: 4

Data Types: double

OutputDirectory — Directory to store analysis results
current directory ("./") (default) | string | character vector

Directory to store analysis results, specified as a string or character vector.

Example: "./AnalysisResults/

Data Types: char | string

ReferenceGTF — Name of optional reference annotation GTF file
string | character vector

Name of an optional reference annotation GTF file to be included in the combined assembly, specified as a string or character vector.

Example: "ref.gtf"
Data Types: char | string

**ReferenceSequence — Name of directory or FASTA file containing genomic sequences**
string | character vector

Name of a directory or FASTA file containing genomic DNA sequences for the reference, specified as a string or character vector.

- If you specify a directory, it must contain one FASTA file per contig. In other words, the directory must contain one FASTA file per reference chromosome, and each file must be named after the chromosome and have a `.fa` or `.fasta` extension.
- If you specify a FASTA file, it must contain all the reference sequences.

The function uses the provided sequences to improve transfrag classification and exclude artifacts.

Example: "allrefs.fasta"

Data Types: char | string

**Version — Supported version**
string

This property is read-only.

Supported version of the original cufflinks software, returned as a string.

Example: "2.2.1"

Data Types: string

**Object Functions**

- getCommand — Translate object properties to original options syntax
- getOptionsTable — Return table with all properties and equivalent options in original syntax

**Examples**

**Create CuffMergeOptions Object**

Create a CuffMergeOptions object with the default values.

```matlab
opt = CuffMergeOptions;
```

Create an object using name-value pairs.

```matlab
opt2 = CuffMergeOptions('OutputDirectory', './merged', 'MinIsoformFraction', 0.1)
```

Create an object by using the original syntax.

```matlab
opt3 = CuffMergeOptions('-o ./merged --min-isoform-fraction 0.1')
```

**Assemble Transcriptome and Perform Differential Expression Testing**

Create a CufflinksOptions object to define cufflinks options, such as the number of parallel threads and the output directory to store the results.
cflOpt = CufflinksOptions;
cflOpt.NumThreads = 8;
cflOpt.OutputDirectory = "./cufflinksOut";

The SAM files provided for this example contain aligned reads for *Mycoplasma pneumoniae* from two samples with three replicates each. The reads are simulated 100bp-reads for two genes (gyrA and gyrB) located next to each other on the genome. All the reads are sorted by reference position, as required by cufflinks.

sams = ["Myco_1_1.sam", "Myco_1_2.sam", "Myco_1_3.sam",...
   "Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"];

Assemble the transcriptome from the aligned reads.
[gtfs,isofpkm,genes,skipped] = cufflinks(sams,cflOpt);

gtfs is a list of GTF files that contain assembled isoforms.

Compare the assembled isoforms using cuffcompare.
stats = cuffcompare(gtfs);

Merge the assembled transcripts using cuffmerge.
mergedGTF = cuffmerge(gtfs,'OutputDirectory','./cuffMergeOutput');

mergedGTF reports only one transcript. This is because the two genes of interest are located next to each other, and cuffmerge cannot distinguish two distinct genes. To guide cuffmerge, use a reference GTF (gyrAB.gtf) containing information about these two genes. If the file is not located in the same directory that you run cuffmerge from, you must also specify the file path.

gyrAB = which('gyrAB.gtf');
mergedGTF2 = cuffmerge(gtfs,'OutputDirectory','./cuffMergeOutput2',...
   'ReferenceGTF',gyrAB);

Calculate abundances (expression levels) from aligned reads for each sample.
abundances1 = cuffquant(mergedGTF2,"["Myco_1_1.sam", "Myco_1_2.sam", "Myco_1_3.sam"]",...
   'OutputDirectory','./cuffquantOutput1');
abundances2 = cuffquant(mergedGTF2,"["Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"]",...
   'OutputDirectory','./cuffquantOutput2');

Assess the significance of changes in expression for genes and transcripts between conditions by performing the differential testing using cuffdiff. The cuffdiff function operates in two distinct steps: the function first estimates abundances from aligned reads, and then performs the statistical analysis. In some cases (for example, distributing computing load across multiple workers), performing the two steps separately is desirable. After performing the first step with cuffquant, you can then use the binary CXB output file as an input to cuffdiff to perform statistical analysis. Because cuffdiff returns several files, specify the output directory is recommended.

isoformDiff = cuffdiff(mergedGTF2,abundances1,abundances2,...
   'OutputDirectory','./cuffdiffOutput');

Display a table containing the differential expression test results for the two genes gyrB and gyrA.
readtable(isoformDiff,'FileType','text')

ans =
You can use cuffnorm to generate normalized expression tables for further analyses. cuffnorm results are useful when you have many samples and you want to cluster them or plot expression levels for genes that are important in your study. Note that you cannot perform differential expression analysis using cuffnorm.

Specify a cell array, where each element is a string vector containing file names for a single sample with replicates.

alignmentFiles = {
    "Myco_1_1.sam", "Myco_1_2.sam", "Myco_1_3.sam",
    "Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"},

isoformNorm = cuffnorm(mergedGTF2, alignmentFiles,...
    'OutputDirectory', './cuffnormOutput');

Display a table containing the normalized expression levels for each transcript.

readtable(isoformNorm,'FileType','text')

ans =

2x7 table

<table>
<thead>
<tr>
<th>tracking_id</th>
<th>q1_0</th>
<th>q1_2</th>
<th>q1_1</th>
<th>q2_1</th>
<th>q2_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>'TCONS_00000001'</td>
<td>1.0913e+05</td>
<td>78628</td>
<td>1.2132e+05</td>
<td>4.3639e+05</td>
<td>4.2228e+05</td>
</tr>
<tr>
<td>'TCONS_00000002'</td>
<td>3.5158e+05</td>
<td>3.7458e+05</td>
<td>3.4238e+05</td>
<td>1.0483e+05</td>
<td>1.1546e+05</td>
</tr>
</tbody>
</table>

Column names starting with q have the format: conditionX_N, indicating that the column contains values for replicate N of conditionX.

References


See Also

CufflinksOptions | cuffcompare | cuffdiff | cuffffread | cuffgtf2sam | cuffmerge | cuffnorm | cuffquant

External Websites

Cufflinks manual

Introduced in R2019a
**cuffnorm**

Normalize transcript expression levels

**Syntax**

```matlab
    cuffnorm(transcriptsAnnot,alignmentFiles)
    cuffnorm(transcriptsAnnot,alignmentFiles,opt)
    cuffnorm(transcriptsAnnot,alignmentFiles,Name,Value)
    [isoform,gene,tss,cds] = cuffnorm(____)
```

**Description**

`cuffnorm(transcriptsAnnot,alignmentFiles)` normalizes transcript expression to FPKM for the samples in `alignmentFiles` and corrects for differences in library size [1].

`cuffnorm` requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

**Note**: `cuffnorm` is supported on the Mac and UNIX platforms only.

`cuffnorm(transcriptsAnnot,alignmentFiles,opt)` uses additional options specified by `opt`.

`cuffnorm(transcriptsAnnot,alignmentFiles,Name,Value)` uses additional options specified by one or more name-value pair arguments. For example, `cuffnorm('gyrAB.gtf', ["Myco_1_1.sam", "Myco_2_1.sam"],'NumThreads',5)` specifies to use five parallel threads.

```matlab
    [isoform,gene,tss,cds] = cuffnorm(____) returns the names of files containing normalized results using any of the input argument combinations in the previous syntaxes. By default, the function saves all files to the current directory.
```

**Examples**

**Assemble Transcriptome and Normalize Expression Levels**

Create a CufflinksOptions object to define cufflinks options, such as the number of parallel threads and the output directory to store the results.

```matlab
    cflOpt = CufflinksOptions;
    cflOpt.NumThreads = 8;
```

The SAM files provided for this example contain aligned reads for *Mycoplasma pneumoniae* from two samples with three replicates each. The reads are simulated 100bp-reads for two genes (gyrA and gyrB) located next to each other on the genome. All the reads are sorted by reference position, as required by cufflinks.

```matlab
    sams = ["Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam",...
            "Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"];
```
Assemble the transcriptome from the aligned reads.

\[
\text{[gtfs, isofpkm, genes, skipped] = cufflinks(sams, cflOpt);}
\]

gtfs is a list of GTF files that contain assembled isoforms.

Compare the assembled isoforms using cuffcompare.

\[
\text{stats = cuffcompare(gtfs);}
\]

Merge the assembled transcripts using cuffmerge.

\[
\text{mergedGTF = cuffmerge(gtfs,'OutputDirectory','./cuffMergeOutput');}
\]

mergedGTF reports only one transcript. This is because the two genes of interest are located next to each other, and cuffmerge cannot distinguish two distinct genes. To guide cuffmerge, use a reference GTF (gyrAB.gtf) containing information about these two genes. If the file is not located in the same directory that you run cuffmerge from, you must also specify the file path.

\[
\text{gyrAB = which('gyrAB.gtf');}
\]

\[
\text{mergedGTF2 = cuffmerge(gtfs,'OutputDirectory','./cuffMergeOutput2',...}
\]

\[
\text{ReferenceGTF', gyrAB);}
\]

Calculate abundances (expression levels) from aligned reads for each sample.

\[
\text{abundances1 = cuffquant(mergedGTF2, ['Myco_1_1.sam', 'Myco_1_2.sam', 'Myco_1_3.sam'],...}
\]

\[
\text{'OutputDirectory','./cuffquantOutput1');}
\]

\[
\text{abundances2 = cuffquant(mergedGTF2, ['Myco_2_1.sam', 'Myco_2_2.sam', 'Myco_2_3.sam'],...}
\]

\[
\text{'OutputDirectory','./cuffquantOutput2');}
\]

Assess the significance of changes in expression for genes and transcripts between conditions by performing the differential testing using cuffdiff. The cuffdiff function operates in two distinct steps: the function first estimates abundances from aligned reads, and then performs the statistical analysis. In some cases (for example, distributing computing load across multiple workers), performing the two steps separately is desirable. After performing the first step with cuffquant, you can then use the binary CXB output file as an input to cuffdiff to perform statistical analysis. Because cuffdiff returns several files, specify the output directory is recommended.

\[
\text{isoformDiff = cuffdiff(mergedGTF2, [abundances1, abundances2],...}
\]

\[
\text{'OutputDirectory','./cuffdiffOutput');}
\]

Display a table containing the differential expression test results for the two genes gyrB and gyrA.

\[
\text{readtable(isoformDiff,'FileType','text')}\]

\[
\text{ans =}
\]

\[
\text{2x14 table}
\]

\[
\begin{array}{cccccc}
\text{test_id} & \text{gene_id} & \text{gene} & \text{locus} & \text{sample_1} & \text{sample_2} \\
\hline
\text{'TCONS_00000001'} & \text{'XLOC_0000001'} & \text{'gyrB'} & \text{'NC_000912.1:2868-7340'} & \text{'q1'} & \text{'q2'} \\
\text{'TCONS_00000002'} & \text{'XLOC_0000001'} & \text{'gyrA'} & \text{'NC_000912.1:2868-7340'} & \text{'q1'} & \text{'q2'} \\
\end{array}
\]

You can use cuffnorm to generate normalized expression tables for further analyses. cuffnorm results are useful when you have many samples and you want to cluster them or plot expression
levels for genes that are important in your study. Note that you cannot perform differential expression analysis using cuffnorm.

Specify a cell array, where each element is a string vector containing file names for a single sample with replicates.

```matlab
alignmentFiles = {{"Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam"},...
                  {{"Myco_2_1.sam","Myco_2_2.sam","Myco_2_3.sam"}}
isoformNorm = cuffnorm(mergedGTF2, alignmentFiles,...
        'OutputDirectory', './cuffnormOutput');
```

Display a table containing the normalized expression levels for each transcript.

```matlab
readtable(isoformNorm,'FileType','text')
```

ans =

2×7 table

<table>
<thead>
<tr>
<th>tracking_id</th>
<th>q1_0</th>
<th>q1_2</th>
<th>q1_1</th>
<th>q2_1</th>
<th>q2_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>'TCONS_00000001'</td>
<td>1.0913e+05</td>
<td>78628</td>
<td>1.2132e+05</td>
<td>4.3639e+05</td>
<td>4.2228e+05</td>
</tr>
<tr>
<td>'TCONS_00000002'</td>
<td>3.5158e+05</td>
<td>3.7458e+05</td>
<td>3.4238e+05</td>
<td>1.0483e+05</td>
<td>1.1105e+05</td>
</tr>
</tbody>
</table>

Column names starting with `q` have the format: `conditionX_N`, indicating that the column contains values for replicate `N` of `conditionX`.

### Input Arguments

- **transcriptsAnnot** — Name of transcript annotation file  
  string | character vector  
  
  Name of the transcript annotation file, specified as a string or character vector. The file can be a GTF or GFF file produced by cufflinks, cuffcompare, or another source of GTF annotations.
  
  Example: "gyrAB.gtf"  
  Data Types: char | string

- **alignmentFiles** — Names of SAM, BAM, or CXB files  
  string vector | cell array  
  
  Names of SAM, BAM, or CXB files containing alignment records for each sample, specified as a string vector or cell array. If you use a cell array, each element must be a string vector or cell array of character vectors specifying alignment files for every replicate of the same sample.
  
  Example: ["Myco_1_1.sam", "Myco_2_1.sam"]  
  Data Types: char | string | cell

- **opt** — cuffnorm options  
  CuffNormOptions object | string | character vector  
  
  cuffnorm options, specified as a CuffNormOptions object, string, or character vector. The string or character vector must be in the original cuffnorm option syntax (prefixed by one or two dashes) [1].
Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: cuffnorm('gyrAB.gtf', ['Myco_1_1.sam', 'Myco_2_1.sam'], 'NumThreads', 5)

**ExtraCommand — Additional commands**

```
"" (default) | string | character vector
```

Additional commands, specified as a string or character vector. The commands must be in the original syntax (prefixed by one or two dashes). Use this option to apply undocumented flags and flags without corresponding MATLAB properties. When the function converts the original flags to MATLAB properties, it stores any unrecognized flags in this option.

Example: 'ExtraCommand', '--library-type fr-secondstrand'

Data Types: char

**IncludeAll — Flag to apply all available options**

false (default) | true

Flag to include all available options with the corresponding default values when converting to the original options syntax, specified as true or false. The original syntax is prefixed by one or two dashes, such as '-d 100 -e 80'. By default, the function converts only the specified options. If the value is true, the function converts all available options, with default values for unspecified options, to the original syntax.

Example: 'IncludeAll', true

Data Types: logical

**Labels — Labels for samples**

```
[] (default) | string | character vector | string vector | cell array of character vectors
```

Labels for samples, specified as a string, character vector, string vector, or cell array of character vectors. If you are providing labels, you must specify the same number of labels as input samples.

Example: 'Labels', ['mutant1', 'mutant2']

Data Types: char | string | cell

**LibraryNormalizationMethod — Method to normalize library size**

```
"geometric" (default) | "classic-fpkm" | "quartile"
```

Method to normalize the library size, specified as one of the following options:

- "geometric" — The function scales the FPKM values by the median geometric mean of fragment counts across all libraries as described in [2].
- "classic-fpkm" — The function applies no scaling to the FPKM values or fragment counts.
- "quartile" — The function scales the FPKM values by the ratio of upper quartiles between fragment counts and the average value across all libraries.

Example: 'LibraryNormalizationMethod', "classic-fpkm"

Data Types: char | string
NormalizeCompatibleHits — Flag to use only fragments compatible with reference transcript to calculate FPKM values
true (default) | false
Flag to use only fragments compatible with a reference transcript to calculate FPKM values, specified as true or false.
Example: ‘NormalizeCompatibleHits’,false
Data Types: logical

NormalizeTotalHits — Flag to include all fragments to calculate FPKM values
false (default) | true
Flag to include all fragments to calculate FPKM values, specified as true or false. If the value is true, the function includes all fragments, including fragments without a compatible reference.
Example: ‘NormalizeTotalHits’,true
Data Types: logical

NumThreads — Number of parallel threads to use
1 (default) | positive integer
Number of parallel threads to use, specified as a positive integer. Threads are run on separate processors or cores. Increasing the number of threads generally improves the runtime significantly, but increases the memory footprint.
Example: ‘NumThreads’,4
Data Types: double

OutputDirectory — Directory to store analysis results
current directory ("./") (default) | string | character vector
Directory to store analysis results, specified as a string or character vector.
Example: 'OutputDirectory','./AnalysisResults/
Data Types: char | string

OutputFormat — Format for result files
"simple-table" (default) | "cuffdiff"
Format for result files, specified as "simple-table" or "cuffdiff".
- "simple-table" — The output is in tab-delimited table format.
- "cuffdiff" — The output is in the same form used by cuffdiff.
Example: ‘OutputFormat’,"cuffdiff"
Data Types: char | string

Seed — Seed for random number generator
0 (default) | nonnegative integer
Seed for the random number generator, specified as a nonnegative integer. Setting a seed value ensures the reproducibility of the analysis results.
Example: 'Seed', 10
Data Types: double

**Output Arguments**

**isoform** — Name of file containing normalized expression level for isoform

```
"./isoforms.fpkm_table"
```

Name of a file containing the normalized expression level for each isoform, returned as a string.

The output string also includes the directory information defined by `OutputDirectory`. The default is the current directory. If you set `OutputDirectory` to "/local/tmp/", the output becomes "/local/tmp/isoforms.fpkm_table".

**gene** — Name of file containing normalized expression level for gene

```
"./genes.fpkm_table"
```

Name of a file containing the normalized expression level for each gene, returned as a string.

The output string also includes the directory information defined by `OutputDirectory`. The default is the current directory. If you set `OutputDirectory` to "/local/tmp/", the output becomes "/local/tmp/genes.fpkm_table".

**tss** — Name of file containing normalized expression level for transcript start site

```
"./tss_groups.fpkm_table"
```

Name of a file containing the normalized expression level for each transcript start site (TSS), returned as a string.

The output string also includes the directory information defined by `OutputDirectory`. The default is the current directory. If you set `OutputDirectory` to "/local/tmp/", the output becomes "/local/tmp/tss_groups.fpkm_table".

**cds** — Name of file containing normalized expression level for coding sequence

```
"./cds.fpkm_table"
```

Name of a file containing the normalized expression level for each coding sequence, returned as a string.

The output string also includes the directory information defined by `OutputDirectory`. The default is the current directory. If you set `OutputDirectory` to "/local/tmp/", the output becomes "/local/tmp/cds.fpkm_table".

**References**


**See Also**

CuffNormOptions | cufflinks
External Websites
Cufflinks manual

Introduced in R2019a
**CuffNormOptions**

Option set for `cuffnorm`

**Description**

A `CuffNormOptions` object contains options for the `cuffnorm` function, which generates expression tables normalized for library size [1].

**Creation**

**Syntax**

```matlab
CuffnormOpt = CuffNormOptions
CuffnormOpt = CuffNormOptions(Name,Value)
CuffnormOpt = CuffNormOptions(S)
```

**Description**

`CuffnormOpt = CuffNormOptions` creates a `CuffNormOptions` object with the default property values.

`CuffNormOptions` requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

**Note** `CuffNormOptions` is supported on the Mac and UNIX platforms only.

`CuffnormOpt = CuffNormOptions(Name,Value)` sets the object properties on page 1-509 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `CuffnormOpt = CuffNormOptions('NumThreads',8)` specifies to use eight parallel threads.

`CuffnormOpt = CuffNormOptions(S)` specifies optional parameters using the string or character vector `S`.

**Input Arguments**

`S — cuffmerge options`  
string | character vector

`cuffmerge` options, specified as a string or character vector. `S` must be in the original `cuffmerge` option syntax (prefixed by one or two dashes).

Example: `--seed 5`

**Properties**

**ExtraCommand — Additional commands**

```
"" (default) | string | character vector
```

"" (default) | string | character vector
Additional commands, specified as a string or character vector. The commands must be in the original syntax (prefixed by one or two dashes). Use this option to apply undocumented flags and flags without corresponding MATLAB properties. When the function converts the original flags to MATLAB properties, it stores any unrecognized flags in this option.

Example: '---library-type fr-secondstrand'

Data Types: char | string

IncludeAll — Flag to use all object properties
false (default) | true

Flag to include all the object properties with the corresponding default values when converting to the original options syntax, specified as true or false. You can convert the properties to the original syntax prefixed by one or two dashes (such as '-d 100 -e 80') by using getCommand. The default value false means that when you call getCommand(optionsObject), it converts only the specified properties. If the value is true, getCommand converts all available properties, with default values for unspecified properties, to the original syntax.

Example: true

Data Types: logical

Labels — Labels for samples
[] (default) | string | character vector | string vector | cell array of character vectors

Labels for samples, specified as a string, character vector, string vector, or cell array of character vectors. If you are providing labels, you must specify the same number of labels as input samples.

Example: ['mutant1', 'mutant2']

Data Types: double | char | string | cell

LibraryNormalizationMethod — Method to normalize library size
"geometric" (default) | "classic-fpkm" | "quartile"

Method to normalize the library size, specified as one of the following options:

- "geometric" — The function scales the FPKM values by the median geometric mean of fragment counts across all libraries as described in [2].
- "classic-fpkm" — The function applies no scaling to the FPKM values or fragment counts.
- "quartile" — The function scales the FPKM values by the ratio of upper quartiles between fragment counts and the average value across all libraries.

Example: "classic-fpkm"

Data Types: char | string

NormalizeCompatibleHits — Flag to use only fragments compatible with reference transcript to calculate FPKM values
true (default) | false

Flag to use only fragments compatible with a reference transcript to calculate FPKM values, specified as true or false.

Example: false

Data Types: logical
**NormalizeTotalHits — Flag to include all fragments to calculate FPKM values**

false (default) | true

Flag to include all fragments to calculate FPKM values, specified as true or false. If the value is true, the function includes all fragments, including fragments without a compatible reference.

Example: true

Data Types: logical

**NumThreads — Number of parallel threads to use**

1 (default) | positive integer

Number of parallel threads to use, specified as a positive integer. Threads are run on separate processors or cores. Increasing the number of threads generally improves the runtime significantly, but increases the memory footprint.

Example: 4

Data Types: double

**OutputDirectory — Directory to store analysis results**

current directory ("./") (default) | string | character vector

Directory to store analysis results, specified as a string or character vector.

Example: "./AnalysisResults/

Data Types: char | string

**OutputFormat — Format for result files**

"simple-table" (default) | "cuffdiff"

Format for result files, specified as "simple-table" or "cuffdiff".

- "simple-table" — The output is in tab-delimited table format.
- "cuffdiff" — The output is in the same form used by cuffdiff.

Example: "cuffdiff"

Data Types: char | string

**Seed — Seed for random number generator**

0 (default) | nonnegative integer

Seed for the random number generator, specified as a nonnegative integer. Setting a seed value ensures the reproducibility of the analysis results.

Example: 10

Data Types: double

**Version — Supported version**

string

This property is read-only.

Supported version of the original cufflinks software, returned as a string.

Example: "2.2.1"
Data Types: string

**Object Functions**
- `getCommand`  Translate object properties to original options syntax
- `getOptionsTable`  Return table with all properties and equivalent options in original syntax

**Examples**

**Create CuffNormOptions Object**

Create a `CuffNormOptions` object with the default values.

```
opt = CuffNormOptions;
```

Create an object using name-value pairs.

```
opt2 = CuffNormOptions('OutputDirectory','./norm','Seed',20)
```

Create an object by using the original syntax.

```
opt3 = CuffNormOptions('--output-dir ./norm --seed 20')
```

**Assemble Transcriptome and Normalize Expression Levels**

Create a `CufflinksOptions` object to define `cufflinks` options, such as the number of parallel threads and the output directory to store the results.

```
cflOpt = CufflinksOptions;
cflOpt.NumThreads = 8;
cflOpt.OutputDirectory = './cufflinksOut';
```

The SAM files provided for this example contain aligned reads for *Mycoplasma pneumoniae* from two samples with three replicates each. The reads are simulated 100bp-reads for two genes (gyrA and gyrB) located next to each other on the genome. All the reads are sorted by reference position, as required by `cufflinks`.

```
sams = ['Myco_1_1.sam','Myco_1_2.sam','Myco_1_3.sam',...
       'Myco_2_1.sam', 'Myco_2_2.sam', 'Myco_2_3.sam'];
```

Assemble the transcriptome from the aligned reads.

```
[gtfs,isofpkm,genes,skipped] = cufflinks(sams,cflOpt);
gtfs is a list of GTF files that contain assembled isoforms.
```

Compare the assembled isoforms using `cuffcompare`.

```
stats = cuffcompare(gtfs);
```

Merge the assembled transcripts using `cuffmerge`.

```
mergedGTF = cuffmerge(gtfs,'OutputDirectory','./cuffMergeOutput');
```
mergedGTF reports only one transcript. This is because the two genes of interest are located next to each other, and cuffmerge cannot distinguish two distinct genes. To guide cuffmerge, use a reference GTF (gyrAB.gtf) containing information about these two genes. If the file is not located in the same directory that you run cuffmerge from, you must also specify the file path.

gyrAB = which('gyrAB.gtf');
mergedGTF2 = cuffmerge(gtfs,'OutputDirectory','./cuffMergeOutput2',
    'ReferenceGTF',gyrAB);

Calculate abundances (expression levels) from aligned reads for each sample.

abundances1 = cuffquant(mergedGTF2,['Myco_1_1.sam','Myco_1_2.sam','Myco_1_3.sam'],
    'OutputDirectory','./cuffquantOutput1');
abundances2 = cuffquant(mergedGTF2,['Myco_2_1.sam', 'Myco_2_2.sam', 'Myco_2_3.sam'],
    'OutputDirectory','./cuffquantOutput2');

Assess the significance of changes in expression for genes and transcripts between conditions by performing the differential testing using cuffdiff. The cuffdiff function operates in two distinct steps: the function first estimates abundances from aligned reads, and then performs the statistical analysis. In some cases (for example, distributing computing load across multiple workers), performing the two steps separately is desirable. After performing the first step with cuffquant, you can then use the binary CXB output file as an input to cuffdiff to perform statistical analysis. Because cuffdiff returns several files, specify the output directory is recommended.

isoformDiff = cuffdiff(mergedGTF2,[abundances1,abundances2],
    'OutputDirectory','./cuffdiffOutput');

Display a table containing the differential expression test results for the two genes gyrB and gyrA.

readtable(isoformDiff,'FileType','text')

ans =
2x14 table

<table>
<thead>
<tr>
<th>test_id</th>
<th>gene_id</th>
<th>gene</th>
<th>locus</th>
<th>sample_1</th>
<th>sample_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>'TCONS_00000001'</td>
<td>'XLOC_000001'</td>
<td>'gyrB'</td>
<td>NC_000912.1:2868-7340</td>
<td>'q1'</td>
<td>'q2'</td>
</tr>
<tr>
<td>'TCONS_00000002'</td>
<td>'XLOC_000001'</td>
<td>'gyrA'</td>
<td>NC_000912.1:2868-7340</td>
<td>'q1'</td>
<td>'q2'</td>
</tr>
</tbody>
</table>

You can use cuffnorm to generate normalized expression tables for further analyses. cuffnorm results are useful when you have many samples and you want to cluster them or plot expression levels for genes that are important in your study. Note that you cannot perform differential expression analysis using cuffnorm.

Specify a cell array, where each element is a string vector containing file names for a single sample with replicates.

alignmentFiles = {
    ['Myco_1_1.sam','Myco_1_2.sam','Myco_1_3.sam'],
    ['Myco_2_1.sam', 'Myco_2_2.sam', 'Myco_2_3.sam']
}
isoformNorm = cuffnorm(mergedGTF2, alignmentFiles,
    'OutputDirectory','./cuffnormOutput');

Display a table containing the normalized expression levels for each transcript.

readtable(isoformNorm,'FileType','text')
ans =

2×7 table

<table>
<thead>
<tr>
<th>tracking_id</th>
<th>q1_0</th>
<th>q1_2</th>
<th>q1_1</th>
<th>q2_1</th>
<th>q2_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>'TCONS_00000001'</td>
<td>1.0913e+05</td>
<td>78628</td>
<td>1.2132e+05</td>
<td>4.3639e+05</td>
<td>4.2228e+05</td>
</tr>
<tr>
<td>'TCONS_00000002'</td>
<td>3.5158e+05</td>
<td>3.7458e+05</td>
<td>3.4238e+05</td>
<td>1.0483e+05</td>
<td>1.1546e+05</td>
</tr>
</tbody>
</table>

Column names starting with \( q \) have the format: \( \text{conditionX}_N \), indicating that the column contains values for replicate \( N \) of \( \text{conditionX} \).

**References**


**See Also**

CufflinksOptions | cuffcompare | cuffdiff | cuffgffread | cuffgtf2sam | cuffmerge | cuffnorm | cuffquant

**External Websites**

Cufflinks manual

**Introduced in R2019a**
cuffquant

Quantify gene and transcript expression profiles

Syntax

cxbFile = cuffquant(transcriptsAnnot,alignmentFiles)
cxbFile = cuffquant(transcriptsAnnot,alignmentFiles,opt)
cxbFile = cuffquant(transcriptsAnnot,alignmentFiles,Name,Value)

Description

cxbFile = cuffquant(transcriptsAnnot,alignmentFiles) generates abundance estimates for the samples in alignmentFiles using the reference annotation file transcriptsAnnot [1]. You can use the generated CXB-format abundance (*.CXB) as input for cuffdiff to perform downstream differential expression analysis.

cuffquant requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

Note cuffquant is supported on the Mac and UNIX platforms only.

cxbFile = cuffquant(transcriptsAnnot,alignmentFiles,opt) uses additional options specified by opt.

cxbFile = cuffquant(transcriptsAnnot,alignmentFiles,Name,Value) uses additional options specified by one or more name-value pair arguments. For example, cuffquant('gyrAB.gtf', ["Myco_1_1.sam", "Myco_2_1.sam"], 'NumThreads', 5) specifies to use five parallel threads.

Examples

Assemble Transcriptome and Perform Differential Expression Testing

Create a CufflinksOptions object to define cufflinks options, such as the number of parallel threads and the output directory to store the results.

cflOpt = CufflinksOptions;  
cflOpt.NumThreads = 8;  
cflOpt.OutputDirectory = "./cufflinksOut";

The SAM files provided for this example contain aligned reads for Mycoplasma pneumoniae from two samples with three replicates each. The reads are simulated 100bp-reads for two genes (gyrA and gyrB) located next to each other on the genome. All the reads are sorted by reference position, as required by cufflinks.

sams = ["Myco_1_1.sam", "Myco_1_2.sam", "Myco_1_3.sam", ...
                    "Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"];
Assemble the transcriptome from the aligned reads.

\[
\text{[gtfs, isofpkm, genes, skipped]} = \text{cufflinks(sams, cflOpt)};
\]

gtfs is a list of GTF files that contain assembled isoforms.

Compare the assembled isoforms using cuffcompare.

\[
\text{stats} = \text{cuffcompare(gtfs)};
\]

Merge the assembled transcripts using cuffmerge.

\[
\text{mergedGTF} = \text{cuffmerge(gtfs, 'OutputDirectory', './cuffMergeOutput')};
\]

mergedGTF reports only one transcript. This is because the two genes of interest are located next to each other, and cuffmerge cannot distinguish two distinct genes. To guide cuffmerge, use a reference GTF (gyrAB.gtf) containing information about these two genes. If the file is not located in the same directory that you run cuffmerge from, you must also specify the file path.

\[
\text{gyrAB} = \text{which('gyrAB.gtf')};
\]

\[
\text{mergedGTF2} = \text{cuffmerge(gtfs, 'OutputDirectory', './cuffMergeOutput2',... 'ReferenceGTF', gyrAB)};
\]

Calculate abundances (expression levels) from aligned reads for each sample.

\[
\text{abundances1} = \text{cuffquant(mergedGTF2, ['Myco_1_1.sam', 'Myco_1_2.sam', 'Myco_1_3.sam'],... 'OutputDirectory', './cuffquantOutput1')};
\]

\[
\text{abundances2} = \text{cuffquant(mergedGTF2, ['Myco_2_1.sam', 'Myco_2_2.sam', 'Myco_2_3.sam'],... 'OutputDirectory', './cuffquantOutput2')};
\]

Assess the significance of changes in expression for genes and transcripts between conditions by performing the differential testing using cuffdiff. The cuffdiff function operates in two distinct steps: the function first estimates abundances from aligned reads, and then performs the statistical analysis. In some cases (for example, distributing computing load across multiple workers), performing the two steps separately is desirable. After performing the first step with cuffquant, you can then use the binary CXB output file as an input to cuffdiff to perform statistical analysis. Because cuffdiff returns several files, specify the output directory is recommended.

\[
\text{isoformDiff} = \text{cuffdiff(mergedGTF2, [abundances1, abundances2],... 'OutputDirectory', './cuffdiffOutput')};
\]

Display a table containing the differential expression test results for the two genes gyrB and gyrA.

\[
\text{readtable(isoformDiff, 'FileType', 'text')}
\]

\[
\text{ans} =
\]

\[
\begin{array}{cccccc}
\text{test_id} & \text{gene_id} & \text{gene} & \text{locus} & \text{sample}_1 & \text{sample}_2 \\
\text{TCONS_00000001} & \text{XLOC_000001} & \text{gyrB} & \text{NC_000912.1:2868-7340} & \text{q1} & \text{q2} \\
\text{TCONS_00000002} & \text{XLOC_000001} & \text{gyrA} & \text{NC_000912.1:2868-7340} & \text{q1} & \text{q2} \\
\end{array}
\]

You can use cuffnorm to generate normalized expression tables for further analyses. cuffnorm results are useful when you have many samples and you want to cluster them or plot expression
levels for genes that are important in your study. Note that you cannot perform differential expression analysis using cuffnorm.

Specify a cell array, where each element is a string vector containing file names for a single sample with replicates.

```matlab
alignmentFiles = {{"Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam"},
                  {{"Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"}}}
isoformNorm = cuffnorm(mergedGTF2, alignmentFiles,...
                     'OutputDirectory', './cuffnormOutput');
```

Display a table containing the normalized expression levels for each transcript.

```matlab
readtable(isoformNorm,'FileType','text')
```

ans =

```
2×7 table

tracking_id          q1_0          q1_2          q1_1          q2_1          q2_0
________________    __________    __________    __________    __________    __________
'TCONS_00000001'    1.0913e+05         78628    1.2132e+05    4.3639e+05    4.2228e+05    4.2814e+05
'TCONS_00000002'    3.5158e+05    3.7458e+05    3.4238e+05    1.0483e+05    1.1546e+05    1.1105e+05
```

Column names starting with q have the format: conditionX_N, indicating that the column contains values for replicate N of conditionX.

**Input Arguments**

**transcriptsAnnot — Name of transcript annotation file**
string | character vector

Name of the transcript annotation file, specified as a string or character vector. The file can be a GTF or GFF file produced by cufflinks, cuffcompare, or another source of GTF annotations.

Example: "gyrAB.gtf"

Data Types: char | string

**alignmentFiles — Names of SAM, BAM, or CXB files**
string vector | cell array

Names of SAM, BAM, or CXB files containing alignment records for each sample, specified as a string vector or cell array. If you use a cell array, each element must be a string vector or cell array of character vectors specifying alignment files for every replicate of the same sample.

Example: ["Myco_1_1.sam", "Myco_2_1.sam"]

Data Types: char | string | cell

**opt — cuffquant options**
CuffQuantOptions object | string | character vector

cuffquant options, specified as a CuffQuantOptions object, string, or character vector. The string or character vector must be in the original cuffquant option syntax (prefixed by one or two dashes) [1].
Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: cuffquant(transcripts,alignmentFiles,'NumThreads',4,'Seed',1)

EffectiveLengthCorrection — Flag to normalize fragment counts

Flag to normalize fragment counts to fragments per kilobase per million mapped reads (FPKM), specified as true or false.

Example: 'EffectiveLengthCorrection',false

Data Types: logical

ExtraCommand — Additional commands

"" (default) | string | character vector

Additional commands, specified as a string or character vector. The commands must be in the original syntax (prefixed by one or two dashes). Use this option to apply undocumented flags and flags without corresponding MATLAB properties. When the function converts the original flags to MATLAB properties, it stores any unrecognized flags in this option.

Example: 'ExtraCommand','--library-type fr-secondstrand'

Data Types: char | string

FragmentBiasCorrection — Name of FASTA file with reference transcripts to detect bias

string | character vector

Name of the FASTA file with reference transcripts to detect bias in fragment counts, specified as a string or character vector. Library preparation can introduce sequence-specific bias into RNA-Seq experiments. Providing reference transcripts improves the accuracy of the transcript abundance estimates.

Example: 'FragmentBiasCorrection','bias.fasta'

Data Types: char | string

FragmentLengthMean — Expected mean fragment length in base pairs

200 (default) | positive integer

Expected mean fragment length, specified as a positive integer. The default value is 200 base pairs. The function can learn the fragment length mean for each SAM file. Using this option is not recommended for paired-end reads.

Example: 'FragmentLengthMean',100

Data Types: double

FragmentLengthSD — Expected standard deviation for fragment length distribution

80 (default) | positive scalar

Expected standard deviation for the fragment length distribution, specified as a positive scalar. The default value is 80 base pairs. The function can learn the fragment length standard deviation for each SAM file. Using this option is not recommended for paired-end reads.
Example: 'FragmentLengthSD',70
Data Types: double

**IncludeAll** — Flag to apply all available options
ture (default) | true

Flag to include all available options with the corresponding default values when converting to the original options syntax, specified as true or false. The original syntax is prefixed by one or two dashes, such as '-d 100 -e 80'. By default, the function converts only the specified options. If the value is true, the function converts all available options, with default values for unspecified options, to the original syntax.

Example: 'IncludeAll',true
Data Types: logical

**LengthCorrection** — Flag to correct by transcript length
ture (default) | false

Flag to correct by the transcript length, specified as true or false. Set this value to false only when the fragment count is independent of the feature size, such as for small RNA libraries with no fragmentation and for 3' end sequencing, where all fragments have the same length.

Example: 'LengthCorrection',false
Data Types: logical

**MaskFile** — Name of GTF or GFF file containing transcripts to ignore
string | character vector

Name of the GTF or GFF file containing transcripts to ignore during analysis, specified as a string or character vector. Some examples of transcripts to ignore include annotated rRNA transcripts, mitochondrial transcripts, and other abundant transcripts. Ignoring these transcripts improves the robustness of the abundance estimates.

Example: 'MaskFile','excludes.gtf'
Data Types: char | string

**MaxBundleFrags** — Maximum number of fragments to include for each locus before skipping
500000 (default) | positive integer

Maximum number of fragments to include for each locus before skipping new fragments, specified as a positive integer. Skipped fragments are marked with the status HIDATA in the file skipped.gtf.

Example: 'MaxBundleFrags',400000
Data Types: double

**MaxFragAlignments** — Maximum number of aligned reads to include for each fragment
Inf (default) | positive integer

Maximum number of aligned reads to include for each fragment before skipping new reads, specified as a positive integer. Inf, the default value, sets no limit on the maximum number of aligned reads.

Example: 'MaxFragAlignments',1000
Data Types: double
MaxMLEIterations — Maximum number of iterations for maximum likelihood estimation
5000 (default) | positive integer

Maximum number of iterations for the maximum likelihood estimation of abundances, specified as a positive integer.
Example: 'MaxMLEIterations',4000
Data Types: double

MinAlignmentCount — Minimum number of alignments required in locus for significance testing
10 (default) | positive integer

Minimum number of alignments required in a locus to perform the significance testing for differences between samples, specified as a positive integer.
Example: 'MinAlignmentCount',8
Data Types: double

MultiReadCorrection — Flag to improve abundance estimation using rescue method
false (default) | true

Flag to improve abundance estimation for reads mapped to multiple genomic positions using the rescue method, specified as true or false. If the value is false, the function divides multimapped reads uniformly to all mapped positions. If the value is true, the function uses additional information, including gene abundance estimation, inferred fragment length, and fragment bias, to improve transcript abundance estimation.

The rescue method is described in [2].
Example: 'MultiReadCorrection',true
Data Types: logical

NumThreads — Number of parallel threads to use
1 (default) | positive integer

Number of parallel threads to use, specified as a positive integer. Threads are run on separate processors or cores. Increasing the number of threads generally improves the runtime significantly, but increases the memory footprint.
Example: 'NumThreads',4
Data Types: double

OutputDirectory — Directory to store analysis results
current directory ("./") (default) | string | character vector

Directory to store analysis results, specified as a string or character vector.
Example: "./AnalysisResults/
Data Types: char | string

Seed — Seed for random number generator
0 (default) | nonnegative integer
Seed for the random number generator, specified as a nonnegative integer. Setting a seed value ensures the reproducibility of the analysis results.

Example: 10

Data Types: double

**Output Arguments**

- **cxbFile — Name of abundances file**
  
  "./abundances.cxb"

  Name of the abundances file, returned as a string.

  The output string also includes the directory information defined by `OutputDirectory`. The default is the current directory. If you set `OutputDirectory` to "/local/tmp/", the output becomes "/local/tmp/abundances.cxb".

**References**


**See Also**

- CuffQuantOptions
- CufflinksOptions
- cuffcompare
- cuffdiff
- cuffgffread
- cuffgtf2sam
- cufflinks
- cuffmerge
- cuffnorm
- cuffquant

**External Websites**

Cufflinks manual

**Introduced in R2019a**
**CuffQuantOptions**

Option set for cuffquant

**Description**

A CuffQuantOptions object contains options to run the cuffquant function, which quantifies gene and transcript expression data [1].

**Creation**

**Syntax**

```matlab
cuffquantOpt = CuffQuantOptions
cuffquantOpt = CuffQuantOptions(Name,Value)
cuffquantOpt = CuffQuantOptions(S)
```

**Description**

`cuffquantOpt = CuffQuantOptions` creates a CuffQuantOptions object with default property values.

CuffQuantOptions requires the Cufflinks Support Package for Bioinformatics Toolbox. If the support package is not installed, then the function provides a download link.

**Note** CuffQuantOptions is supported on the Mac and UNIX platforms only.

`cuffquantOpt = CuffQuantOptions(Name,Value)` sets the object properties on page 1-522 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `cuffquantOpt = CuffQuantOptions('NumThreads',8)` specifies to use eight parallel threads.

`cuffquantOpt = CuffQuantOptions(S)` specifies optional parameters using a string or character vector S.

**Properties**

**EffectiveLengthCorrection** — Flag to normalize fragment counts

- true (default)
- false

**EffectiveLengthCorrection**

- Flag to normalize fragment counts

**Input Arguments**

**S** — Cuffquant options

- string | character vector

Cuffquant options, specified as a string or character vector. S must be in the cuffquant option syntax (prefixed by one or two dashes).

Example: `--seed 5`
Flag to normalize fragment counts to fragments per kilobase per million mapped reads (FPKM), specified as true or false.
Example: false
Data Types: logical

ExtraCommand — Additional commands
"" (default) | string | character vector

Additional commands, specified as a string or character vector. The commands must be in the original syntax (prefixed by one or two dashes). Use this option to apply undocumented flags and flags without corresponding MATLAB properties. When the function converts the original flags to MATLAB properties, it stores any unrecognized flags in this option.
Example: '--library-type fr-secondstrand'
Data Types: char | string

FragmentBiasCorrection — Name of FASTA file with reference transcripts to detect bias
string | character vector

Name of the FASTA file with reference transcripts to detect bias in fragment counts, specified as a string or character vector. Library preparation can introduce sequence-specific bias into RNA-Seq experiments. Providing reference transcripts improves the accuracy of the transcript abundance estimates.
Example: "bias.fasta"
Data Types: char | string

FragmentLengthMean — Expected mean fragment length in base pairs
200 (default) | positive integer

Expected mean fragment length, specified as a positive integer. The default value is 200 base pairs. The function can learn the fragment length mean for each SAM file. Using this option is not recommended for paired-end reads.
Example: 100
Data Types: double

FragmentLengthSD — Expected standard deviation for fragment length distribution
80 (default) | positive scalar

Expected standard deviation for the fragment length distribution, specified as a positive scalar. The default value is 80 base pairs. The function can learn the fragment length standard deviation for each SAM file. Using this option is not recommended for paired-end reads.
Example: 70
Data Types: double

IncludeAll — Flag to use all object properties
false (default) | true

Flag to include all the object properties with the corresponding default values when converting to the original options syntax, specified as true or false. You can convert the properties to the original syntax prefixed by one or two dashes (such as '-d 100 -e 80') by using getCommand. The default
value `false` means that when you call `getCommand(optionsObject)`, it converts only the specified properties. If the value is `true`, `getCommand` converts all available properties, with default values for unspecified properties, to the original syntax.

Example: `true`
Data Types: `logical`

**LengthCorrection — Flag to correct by transcript length**

`true` (default) | `false`

Flag to correct by the transcript length, specified as `true` or `false`. Set this value to `false` only when the fragment count is independent of the feature size, such as for small RNA libraries with no fragmentation and for 3' end sequencing, where all fragments have the same length.

Example: `false`
Data Types: `logical`

**MaskFile — Name of GTF or GFF file containing transcripts to ignore**

String | character vector

Name of the GTF or GFF file containing transcripts to ignore during analysis, specified as a string or character vector. Some examples of transcripts to ignore include annotated rRNA transcripts, mitochondrial transcripts, and other abundant transcripts. Ignoring these transcripts improves the robustness of the abundance estimates.

Example: `'excludes.gtf'`
Data Types: `char | string`

**MaxBundleFrags — Maximum number of fragments to include for each locus before skipping**

500000 (default) | positive integer

Maximum number of fragments to include for each locus before skipping new fragments, specified as a positive integer. Skipped fragments are marked with the status `HIDATA` in the file `skipped.gtf`.

Example: `400000`
Data Types: `double`

**MaxFragAlignments — Maximum number of aligned reads to include for each fragment**

Inf (default) | positive integer

Maximum number of aligned reads to include for each fragment before skipping new reads, specified as a positive integer. Inf, the default value, sets no limit on the maximum number of aligned reads.

Example: `1000`
Data Types: `double`

**MaxMLEIterations — Maximum number of iterations for maximum likelihood estimation**

50000 (default) | positive integer

Maximum number of iterations for the maximum likelihood estimation of abundances, specified as a positive integer.

Example: `4000`
Data Types: double

**MinAlignmentCount — Minimum number of alignments in locus to perform significance testing**

10 (default) | positive integer

Minimum number of alignments required in a locus to perform the significance testing for differences between samples, specified as a positive integer.

Example: 8

Data Types: double

**MultiReadCorrection — Flag to improve abundance estimation using rescue method**

false (default) | true

Flag to improve abundance estimation for reads mapped to multiple genomic positions using the rescue method, specified as true or false. If the value is false, the function divides multimapped reads uniformly to all mapped positions. If the value is true, the function uses additional information, including gene abundance estimation, inferred fragment length, and fragment bias, to improve transcript abundance estimation.

The rescue method is described in [2].

Example: true

Data Types: logical

**NumThreads — Number of parallel threads to use**

1 (default) | positive integer

Number of parallel threads to use, specified as a positive integer. Threads are run on separate processors or cores. Increasing the number of threads generally improves the runtime significantly, but increases the memory footprint.

Example: 4

Data Types: double

**OutputDirectory — Directory to store analysis results**

current directory ("./") (default) | string | character vector

Directory to store analysis results, specified as a string or character vector.

Example: "./AnalysisResults/

Data Types: char | string

**Seed — Seed for random number generator**

0 (default) | nonnegative integer

Seed for the random number generator, specified as a nonnegative integer. Setting a seed value ensures the reproducibility of the analysis results.

Example: 10

Data Types: double

**Version — Supported version**

string
This property is read-only.

Supported version of the original cufflinks software, returned as a string.

Example: "2.2.1"

Data Types: string

**Object Functions**

- getCommand: Translate object properties to original options syntax
- getOptionsTable: Return table with all properties and equivalent options in original syntax

**Examples**

**Create CuffQuantOptions Object**

Create a CuffQuantOptions object with the default values.

```matlab
opt = CuffQuantOptions;
```

Create an object using name-value pairs.

```matlab
opt2 = CuffQuantOptions('NumThreads',4,'MinAlignmentCount',50)
```

Create an object by using the original syntax.

```matlab
opt3 = CuffQuantOptions('-p 4 --min-alignment-count 50')
```

**Assemble Transcriptome and Perform Differential Expression Testing**

Create a CufflinksOptions object to define cufflinks options, such as the number of parallel threads and the output directory to store the results.

```matlab
cflOpt = CufflinksOptions;
cflOpt.NumThreads = 8;
cflOpt.OutputDirectory = "./cufflinksOut";
```

The SAM files provided for this example contain aligned reads for *Mycoplasma pneumoniae* from two samples with three replicates each. The reads are simulated 100bp-reads for two genes (gyrA and gyrB) located next to each other on the genome. All the reads are sorted by reference position, as required by cufflinks.

```matlab
sams = ["Myco_1_1.sam", "Myco_1_2.sam", "Myco_1_3.sam",...
        "Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"];
```

Assemble the transcriptome from the aligned reads.

```matlab
[gtfs,isofpkm,genes,skipped] = cufflinks(sams,cflOpt);
```

**gtfs** is a list of GTF files that contain assembled isoforms.

Compare the assembled isoforms using cuffcompare.

```matlab
stats = cuffcompare(gtfs);
```
Merge the assembled transcripts using cuffmerge.

```matlab
mergedGTF = cuffmerge(gtfs,'OutputDirectory','./cuffMergeOutput');
```

mergedGTF reports only one transcript. This is because the two genes of interest are located next to each other, and cuffmerge cannot distinguish two distinct genes. To guide cuffmerge, use a reference GTF (gyrAB.gtf) containing information about these two genes. If the file is not located in the same directory that you run cuffmerge from, you must also specify the file path.

```matlab
gyrAB = which('gyrAB.gtf');
mergedGTF2 = cuffmerge(gtfs,'OutputDirectory','./cuffMergeOutput2',...
                     'ReferenceGTF',gyrAB);
```

Calculate abundances (expression levels) from aligned reads for each sample.

```matlab
abundances1 = cuffquant(mergedGTF2,
"Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam",...
'OutputDirectory','./cuffquantOutput1');
abundances2 = cuffquant(mergedGTF2,
"Myco_2_1.sam","Myco_2_2.sam","Myco_2_3.sam",...
'OutputDirectory','./cuffquantOutput2');
```

Assess the significance of changes in expression for genes and transcripts between conditions by performing the differential testing using cuffdiff. The cuffdiff function operates in two distinct steps: the function first estimates abundances from aligned reads, and then performs the statistical analysis. In some cases (for example, distributing computing load across multiple workers), performing the two steps separately is desirable. After performing the first step with cuffquant, you can then use the binary CXB output file as an input to cuffdiff to perform statistical analysis. Because cuffdiff returns several files, specify the output directory is recommended.

```matlab
isoformDiff = cuffdiff(mergedGTF2,[abundances1,abundances2],...
                      'OutputDirectory','./cuffdiffOutput');
```

Display a table containing the differential expression test results for the two genes gyrB and gyrA.

```matlab
readtable(isoformDiff,'FileType','text')
```

You can use cuffnorm to generate normalized expression tables for further analyses. cuffnorm results are useful when you have many samples and you want to cluster them or plot expression levels for genes that are important in your study. Note that you cannot perform differential expression analysis using cuffnorm.

Specify a cell array, where each element is a string vector containing file names for a single sample with replicates.

```matlab
alignmentFiles = {
"Myco_1_1.sam","Myco_1_2.sam","Myco_1_3.sam",... 
"Myco_2_1.sam", "Myco_2_2.sam", "Myco_2_3.sam"}
isoformNorm = cuffnorm(mergedGTF2, alignmentFiles,...
                      'OutputDirectory', './cuffnormOutput');
```
Display a table containing the normalized expression levels for each transcript.

```matlab
readtable(isoformNorm,'FileType','text')
```

```
ans =

2x7 table

tracking_id      q1_0      q1_2      q1_1      q2_1      q2_0
________________________    __________    __________    __________    __________    __________
'TCONS_000000001'        1.0913e+05    78628    1.2132e+05    4.3639e+05    4.2228e+05    4.2814e+05
'TCONS_000000002'        3.5158e+05    3.7458e+05    3.4238e+05    1.0483e+05    1.1546e+05    1.1105e+05

Column names starting with q have the format: conditionX_N, indicating that the column contains values for replicate N of conditionX.

References


See Also

Cufflinks|Options | cuffcompare | cuffdiff | cuffffread | cuffgtf2sam | cuffmerge | cuffnorm | cuffquant

External Websites

Cufflinks manual

Introduced in R2019a
cytobandread

Read cytogenetic banding information

Syntax

\[ CytoStruct = \text{cytobandread}(File) \]

Input Arguments

| File          | Character vector or string specifying a file containing cytogenetic G-banding data, such as an NCBI ideogram text file or a UCSC Genome Browser cytoband text file. |

Output Arguments

<table>
<thead>
<tr>
<th>CytoStruct</th>
<th>Structure containing cytogenetic G-banding data in the following fields:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• ChromLabels</td>
</tr>
<tr>
<td></td>
<td>• BandStartBPs</td>
</tr>
<tr>
<td></td>
<td>• BandEndBPs</td>
</tr>
<tr>
<td></td>
<td>• BandLabels</td>
</tr>
<tr>
<td></td>
<td>• GieStains</td>
</tr>
</tbody>
</table>

Description

\[ CytoStruct = \text{cytobandread}(File) \] reads File, which is a character vector or string specifying a file containing cytogenetic G-banding data, and returns CytoStruct, which is a structure containing the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChromLabels</td>
<td>Cell array containing the chromosome label (number or letter) on which each band is located.</td>
</tr>
<tr>
<td>BandStartBPs</td>
<td>Column vector containing the number of the base pair at the start of each band.</td>
</tr>
<tr>
<td>BandEndBPs</td>
<td>Column vector containing the number of the base pair at the end of each band.</td>
</tr>
<tr>
<td>BandLabels</td>
<td>Cell array containing the FISH label of each band, for example, p32.3.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GieStains</td>
<td>Cell array containing the Giemsa staining result for each band. Possible stain results depend on the species. For example, for <em>Homo sapiens</em>, the possibilities are:</td>
</tr>
<tr>
<td></td>
<td>• gneg</td>
</tr>
<tr>
<td></td>
<td>• gpos25</td>
</tr>
<tr>
<td></td>
<td>• gpos50</td>
</tr>
<tr>
<td></td>
<td>• gpos75</td>
</tr>
<tr>
<td></td>
<td>• gpos100</td>
</tr>
<tr>
<td></td>
<td>• acen</td>
</tr>
<tr>
<td></td>
<td>• stalk</td>
</tr>
<tr>
<td></td>
<td>• gvar</td>
</tr>
</tbody>
</table>

**Tip** You can download files containing cytogenetic G-banding data from the NCBI or UCSC Genome Browser ftp site. For example, you can download the cytogenetic banding data for *Homo sapiens* from:

ftp://hgdownload.cse.ucsc.edu/goldenPath/hg18/database/cytoBandIdeo.txt.gz

### Examples

**Read cytogenetic banding information from a file**

Read the cytogenetic banding information for *Homo sapiens* into a structure.

```matlab
bands = cytobandread('hs_cytoBand.txt')
```

`bands = struct with fields:

  ChromLabels: {862x1 cell}
  BandStartBPs: [862x1 int32]
  BandEndBPs: [862x1 int32]
  BandLabels: {862x1 cell}
  GieStains: {862x1 cell}

Plot the entire chromosome ideogram.

```matlab
chromosomeplot(bands)
title('Human Karyogram')
```
See Also

chromosomeplot

Introduced in R2007b
**DataMatrix object**

Data structure encapsulating data and metadata from microarray experiment so that it can be indexed by gene or probe identifiers and by sample identifiers.

**Description**

A DataMatrix object is a data structure encapsulating measurement data and feature metadata from a microarray experiment so that it can be indexed by gene or probe identifiers and by sample identifiers. A DataMatrix object stores experimental data in a matrix, with rows typically corresponding to gene names or probe identifiers, and columns typically corresponding to sample identifiers. A DataMatrix object also stores metadata, such as the gene names or probe identifiers and sample identifiers, in row names and column names.

You create a DataMatrix object using the object constructor function `DataMatrix`. 
## Property Summary

### Properties of a DataMatrix Object

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Character vector that describes the DataMatrix object. Default is ''.</td>
</tr>
<tr>
<td>RowNames</td>
<td>Empty array or cell array of character vectors that specifies the names for the rows, typically gene names or probe identifiers. The number of elements in the cell array must equal the number of rows in the matrix. Default is an empty array.</td>
</tr>
<tr>
<td>ColNames</td>
<td>Empty array or cell array of character vectors that specifies the names for the columns, typically sample identifiers. The number of elements in the cell array must equal the number of columns in the matrix.</td>
</tr>
<tr>
<td>NRows</td>
<td>Read-only. Positive number that specifies the number of rows in the matrix.</td>
</tr>
<tr>
<td>Note</td>
<td>You cannot modify this property directly. You can access it using the <code>get</code> method.</td>
</tr>
<tr>
<td>NCols</td>
<td>Read-only. Positive number that specifies the number of columns in the matrix.</td>
</tr>
<tr>
<td>Note</td>
<td>You cannot modify this property directly. You can access it using the <code>get</code> method.</td>
</tr>
<tr>
<td>NDims</td>
<td>Read-only. Positive number that specifies the number of dimensions in the matrix.</td>
</tr>
<tr>
<td>Note</td>
<td>You cannot modify this property directly. You can access it using the <code>get</code> method.</td>
</tr>
<tr>
<td>ElementClass</td>
<td>Read-only. Character vector that specifies the class type of the elements in the DataMatrix object, such as single or double.</td>
</tr>
<tr>
<td>Note</td>
<td>You cannot modify this property directly. You can access it using the <code>get</code> method.</td>
</tr>
</tbody>
</table>
## Method Summary

### General Methods of a DataMatrix Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>colnames</td>
<td>Retrieve or set column names of DataMatrix object.</td>
</tr>
<tr>
<td>disp</td>
<td>Display DataMatrix object.</td>
</tr>
<tr>
<td>display</td>
<td>Display DataMatrix object, printing DataMatrix object name. To invoke this method, enter the name of a DataMatrix object at the command prompt.</td>
</tr>
<tr>
<td>dmwrite</td>
<td>Write DataMatrix object to text file.</td>
</tr>
<tr>
<td>double</td>
<td>Convert DataMatrix object to double-precision array.</td>
</tr>
<tr>
<td>get</td>
<td>Retrieve information about DataMatrix object.</td>
</tr>
<tr>
<td>isempty</td>
<td>Determine if DataMatrix object is empty.</td>
</tr>
<tr>
<td>isfinite</td>
<td>Determine if DataMatrix object elements are finite.</td>
</tr>
<tr>
<td>isinf</td>
<td>Determine if DataMatrix object elements are infinite.</td>
</tr>
<tr>
<td>isnan</td>
<td>Determine if DataMatrix object elements are NaN.</td>
</tr>
<tr>
<td>isscalar</td>
<td>Determine if DataMatrix object is scalar.</td>
</tr>
<tr>
<td>isequal</td>
<td>Test DataMatrix objects for equality.</td>
</tr>
<tr>
<td>isequaln</td>
<td>Test DataMatrix objects for equality, treating NaNs as equal.</td>
</tr>
<tr>
<td>isvector</td>
<td>Determine if DataMatrix object is vector.</td>
</tr>
<tr>
<td>length</td>
<td>Return length of DataMatrix object.</td>
</tr>
<tr>
<td>ndims</td>
<td>Return number of dimensions in DataMatrix object.</td>
</tr>
<tr>
<td>numel</td>
<td>Return number of elements in DataMatrix object.</td>
</tr>
<tr>
<td>plot</td>
<td>Draw 2-D line plot of DataMatrix object.</td>
</tr>
<tr>
<td>rownames</td>
<td>Retrieve or set row names of DataMatrix object.</td>
</tr>
<tr>
<td>set</td>
<td>Set property of DataMatrix object.</td>
</tr>
<tr>
<td>single</td>
<td>Convert DataMatrix object to single-precision array.</td>
</tr>
<tr>
<td>size</td>
<td>Return size of DataMatrix object.</td>
</tr>
</tbody>
</table>
### Methods for Manipulating the Data in a DataMatrix Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>Concatenate DataMatrix objects. The horzcat and vertcat methods implement special cases.</td>
</tr>
<tr>
<td>horzcat</td>
<td>Concatenate DataMatrix objects horizontally.</td>
</tr>
<tr>
<td>sortcols</td>
<td>Sort columns of DataMatrix object in ascending or descending order.</td>
</tr>
<tr>
<td>sortrows</td>
<td>Sort rows of DataMatrix object in ascending or descending order.</td>
</tr>
<tr>
<td>subsasgn</td>
<td>Subscripted assignment for DataMatrix object. To invoke this method, use parentheses or dot indexing described in &quot;Accessing Data in DataMatrix Objects&quot;.</td>
</tr>
<tr>
<td>subsref</td>
<td>Subscripted reference for DataMatrix object. To invoke this method, use parentheses or dot indexing described in &quot;Accessing Data in DataMatrix Objects&quot;.</td>
</tr>
<tr>
<td>transpose</td>
<td>Transpose DataMatrix object.</td>
</tr>
<tr>
<td>vertcat</td>
<td>Concatenate DataMatrix objects vertically.</td>
</tr>
</tbody>
</table>

### Descriptive Statistics and Statistical Learning Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kmeans</td>
<td>K-means clustering.</td>
</tr>
<tr>
<td>max</td>
<td>Return maximum values in DataMatrix object.</td>
</tr>
<tr>
<td>mean</td>
<td>Return average or mean values in DataMatrix object.</td>
</tr>
<tr>
<td>median</td>
<td>Return median values in DataMatrix object.</td>
</tr>
<tr>
<td>min</td>
<td>Return minimum values in DataMatrix object.</td>
</tr>
<tr>
<td>nanmax</td>
<td>Return maximum values in DataMatrix object ignoring NaN values.</td>
</tr>
<tr>
<td>nanmean</td>
<td>Return average or mean values in DataMatrix object ignoring NaN values.</td>
</tr>
<tr>
<td>nanmedian</td>
<td>Return median values in DataMatrix object ignoring NaN values.</td>
</tr>
<tr>
<td>nanmin</td>
<td>Return minimum values in DataMatrix object ignoring NaN values.</td>
</tr>
<tr>
<td>nanstd</td>
<td>Return standard deviation values in DataMatrix object ignoring NaN values.</td>
</tr>
<tr>
<td>nansum</td>
<td>Return sum of elements in DataMatrix object ignoring NaN values.</td>
</tr>
<tr>
<td>nanvar</td>
<td>Return variance values in DataMatrix object ignoring NaN values.</td>
</tr>
<tr>
<td>pca</td>
<td>Principal component analysis on data.</td>
</tr>
<tr>
<td>pdist</td>
<td>Pairwise distance.</td>
</tr>
<tr>
<td>std</td>
<td>Return standard deviation values in DataMatrix object.</td>
</tr>
<tr>
<td>sum</td>
<td>Return sum of elements in DataMatrix object.</td>
</tr>
<tr>
<td>var</td>
<td>Return variance values in DataMatrix object.</td>
</tr>
</tbody>
</table>
Unary Methods — Exponential

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp</td>
<td>Exponential.</td>
</tr>
<tr>
<td>log</td>
<td>Natural logarithm.</td>
</tr>
<tr>
<td>log10</td>
<td>Common (base 10) logarithm.</td>
</tr>
<tr>
<td>log2</td>
<td>Base 2 logarithm and dissect floating-point numbers into exponent and mantissa.</td>
</tr>
<tr>
<td>pow2</td>
<td>Base 2 power and scale floating-point numbers.</td>
</tr>
<tr>
<td>sqrt</td>
<td>Square root.</td>
</tr>
</tbody>
</table>

Unary Methods — Integer

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ceil</td>
<td>Round DataMatrix object toward infinity.</td>
</tr>
<tr>
<td>fix</td>
<td>Round DataMatrix object toward zero.</td>
</tr>
<tr>
<td>floor</td>
<td>Round DataMatrix object toward minus infinity.</td>
</tr>
<tr>
<td>round</td>
<td>Round DataMatrix object to nearest integer.</td>
</tr>
</tbody>
</table>

Unary Methods — Custom

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmarrayfun</td>
<td>Apply function to each element in DataMatrix object.</td>
</tr>
</tbody>
</table>

Binary Methods — Arithmetic Operator

<table>
<thead>
<tr>
<th>Operator</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>plus</td>
<td>Add DataMatrix objects</td>
</tr>
<tr>
<td>-</td>
<td>minus</td>
<td>Subtract DataMatrix objects.</td>
</tr>
<tr>
<td>.*</td>
<td>times</td>
<td>Multiply DataMatrix objects.</td>
</tr>
<tr>
<td>./</td>
<td>rdivide</td>
<td>Right array divide DataMatrix objects.</td>
</tr>
<tr>
<td>.\</td>
<td>ldivide</td>
<td>Left array divide DataMatrix objects.</td>
</tr>
<tr>
<td>.^</td>
<td>power</td>
<td>Array power DataMatrix objects.</td>
</tr>
</tbody>
</table>

Binary Methods — Relational Operator

<table>
<thead>
<tr>
<th>Operator</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>lt</td>
<td>Test DataMatrix objects for less than.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>le</td>
<td>Test DataMatrix objects for less than or equal to.</td>
</tr>
<tr>
<td>&gt;</td>
<td>gt</td>
<td>Test DataMatrix objects for greater than.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>ge</td>
<td>Test DataMatrix objects for greater than or equal to.</td>
</tr>
<tr>
<td>==</td>
<td>eq</td>
<td>Test DataMatrix objects for equality.</td>
</tr>
<tr>
<td>~=</td>
<td>ne</td>
<td>Test DataMatrix objects for inequality.</td>
</tr>
</tbody>
</table>
### Binary Methods — Custom

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmbsxfun</td>
<td>Apply element-by-element binary operation to two DataMatrix objects with singleton expansion enabled.</td>
</tr>
</tbody>
</table>

### Examples

#### Example 1.10. Determining Properties and Property Values of a DataMatrix Object

You can display all properties and their current values of a DataMatrix object, `DMobj`, by using the following syntax:

```matlab
get(DMobj)
```

You can return all properties and their current values of `DMobj`, a DataMatrix object, to `DMstruct`, a scalar structure in which each field name is a property of a DataMatrix object, and each field contains the value of that property, by using the following syntax:

```matlab
DMstruct = get(DMobj)
```

You can return the value of a specific property of a DataMatrix object, `DMobj`, by using either of the following syntaxes:

```matlab
PropertyValue = get(DMobj, 'PropertyName')
```

```matlab
PropertyValue = DMobj.PropertyName
```

You can return the value of specific properties of a DataMatrix object, `DMobj`, by using the following syntax:

```matlab
[PropertyValue1, PropertyValue2, ...] = get(DMobj, ...
'PropertyName1', 'PropertyName2', ...)
```

#### Example 1.11. Determining Possible Values of DataMatrix Object Properties

You can display possible values for all properties that have a fixed set of property values in a DataMatrix object, `DMobj`, by using the following syntax:

```matlab
set(DMobj)
```

You can display possible values for a specific property that has a fixed set of property values in a DataMatrix object, `DMobj`, by using the following syntax:

```matlab
set(DMobj, 'PropertyName')
```

#### Example 1.12. Specifying Properties of a DataMatrix Object

You can set a specific property of a DataMatrix object, `DMobj`, by using either of the following syntaxes:

```matlab
DMobj = set(DMobj, 'PropertyName', PropertyValue)
```

```matlab
DMobj.PropertyName = PropertyValue
```

You can set multiple properties of a DataMatrix object, `DMobj`, by using the following syntax:
set(DMobj, 'PropertyName1', PropertyValue1, ...
   'PropertyName2', PropertyValue2, ...)

**Note**  For more examples of creating and using DataMatrix objects, see "Representing Expression Data Values in DataMatrix Objects".

**Compatibility Considerations**

**princomp method has been renamed**

*Errors starting in R2017b*

The princomp method of DataMatrix object has been renamed. Replace instances of princomp with pca.

**See Also**

DataMatrix | colnames | disp | dmirrorfun | dmbsxfun | dmirror | double | eq | ge | get | gt | horzcat | isequal | isequaln | ldivide | le | lt | max | mean | median | min | minus | ndims | ne | numel | plot | plus | power | rdivide | rownames | set | single | sortcols | sortrows | std | sum | times | var | vertcat

**Introduced in R2008b**
DataMatrix

Create DataMatrix object

Syntax

\[
\begin{align*}
    \text{DMobj} &= \text{DataMatrix}(\text{Matrix}) \\
    \text{DMobj} &= \text{DataMatrix}(\text{Matrix}, \text{RowNames}, \text{ColumnNames}) \\
    \text{DMobj} &= \text{DataMatrix}(\text{"File"}, \text{FileName}) \\
    \text{DMobj} &= \text{DataMatrix}(\ldots, \text{\'RowNames\'}, \text{RowNamesValue}, \ldots) \\
    \text{DMobj} &= \text{DataMatrix}(\ldots, \text{\'ColNames\'}, \text{ColNamesValue}, \ldots) \\
    \text{DMobj} &= \text{DataMatrix}(\ldots, \text{\'Name\'}, \text{NameValue}, \ldots) \\
    \text{DMobj} &= \text{DataMatrix}(\text{\'File\'}, \text{FileName}, \ldots \text{\'Delimiter\'}, \text{DelimiterValue}, \ldots) \\
    \text{DMobj} &= \text{DataMatrix}(\text{\'File\'}, \text{FileName}, \ldots \text{\'HLine\'}, \text{HLineValue}, \ldots) \\
    \text{DMobj} &= \text{DataMatrix}(\text{\'File\'}, \text{FileName}, \ldots \text{\'Rows\'}, \text{RowsValue}, \ldots) \\
    \text{DMobj} &= \text{DataMatrix}(\text{\'File\'}, \text{FileName}, \ldots \text{\'Columns\'}, \text{ColumnsValue}, \ldots)
\end{align*}
\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>Two-dimensional numeric or logical array.</td>
</tr>
<tr>
<td>RowNames</td>
<td>Row names for the DataMatrix object, specified by a numeric vector; character array, string vector, or cell array of character vectors, whose elements are equal in number to the number of rows in Matrix. RowNames are typically gene names or probe identifiers from a microarray experiment.</td>
</tr>
<tr>
<td>Note</td>
<td>The row names do not need to be unique.</td>
</tr>
<tr>
<td>ColumnNames</td>
<td>Column names for the DataMatrix object, specified by a numeric vector; character array, string vector, or cell array of character vectors, whose elements are equal in number to the number of columns in Matrix. ColumnNames are typically sample identifiers from a microarray experiment.</td>
</tr>
<tr>
<td>Note</td>
<td>The column names do not need to be unique.</td>
</tr>
<tr>
<td>FileName</td>
<td>Character vector or string specifying a file name or a path and file name of a tab-delimited TXT or XLS file that contains table-oriented data and metadata.</td>
</tr>
<tr>
<td>Note</td>
<td>Typically, the first row of the table contains column names, the first column contains row names, and the numeric data starts at the 2,2 position. The DataMatrix function will detect if the first column does not contain row names, and read data from the first column. However, if the first row does not contain header text (column names), set the HLine property to 0.</td>
</tr>
<tr>
<td><strong>RowNamesValue, ColNamesValue</strong></td>
<td>Row names or column names for the DataMatrix object. Choices are:</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>• Numeric vector, character array, string vector, or a cell array of character vectors, whose elements are equal in number to the number of rows or number of columns of numeric data in the input matrix.</td>
</tr>
<tr>
<td></td>
<td>• A character vector or string, which is used as a prefix for row or column names. Numbers will be appended to the prefix.</td>
</tr>
<tr>
<td></td>
<td>• true — Unique row or column names will be assigned using the formats row1, row2, row3, etc., or col1, col2, col3, etc.</td>
</tr>
<tr>
<td></td>
<td>• false — Default. No row or column names are assigned.</td>
</tr>
</tbody>
</table>

**Note** The row or column names do not need to be unique.

<table>
<thead>
<tr>
<th><strong>NameValue</strong></th>
<th>Character vector or string specifying a name for the DataMatrix object. Default is ''.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>DelimiterValue</strong></th>
<th>Character vector or string specifying a delimiter symbol to use for the input file. Typical choices are:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• ' '</td>
</tr>
<tr>
<td></td>
<td>• '	' (default)</td>
</tr>
<tr>
<td></td>
<td>• ','</td>
</tr>
<tr>
<td></td>
<td>• ';'</td>
</tr>
<tr>
<td></td>
<td>• '</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>HLineValue</strong></th>
<th>Positive integer that specifies which row of the input file contains the column header text (column names). Default is 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>When creating the DataMatrix object <strong>DMobj</strong>, the DataMatrix function loads data from (<em>HLineValue</em> + 1) to the end of the file.</td>
</tr>
</tbody>
</table>

**Tip** If the input file does not contain column header text (column names), set **HLineValue** to 0.

<table>
<thead>
<tr>
<th><strong>RowsValue, ColumnsValue</strong></th>
<th>A subset of rows or columns in <strong>File</strong>, for the DataMatrix function to use to create the DataMatrix object. Choices are:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Cell array of character vectors</td>
</tr>
<tr>
<td></td>
<td>• Character array</td>
</tr>
<tr>
<td></td>
<td>• String vector</td>
</tr>
<tr>
<td></td>
<td>• Numeric or logical vector</td>
</tr>
</tbody>
</table>

**Description**

A DataMatrix object encapsulates measurement data and feature metadata from a microarray experiment so that it can be indexed by gene names or probe identifiers and by sample identifiers. For examples of creating and using DataMatrix objects, see “Representing Expression Data Values in DataMatrix Objects”.

1-540
The DataMatrix constructor function is part of the microarray object package. To make it available, type the following in the MATLAB command line:

```matlab
import bioma.data.*
```

Otherwise, use `bioma.data.DataMatrix` instead of `DataMatrix`, in the following syntaxes.

```matlab
DMobj = DataMatrix(Matrix)
```
creates a DataMatrix object, `DMobj`, from `Matrix`, a two-dimensional numeric or logical array. `Matrix` can also be a DataMatrix object.

```matlab
DMobj = DataMatrix(Matrix, RowNames, ColumnNames)
```
creates a DataMatrix object, `DMobj`, from `Matrix`, a two-dimensional numeric or logical array, with row names and column names specified by `RowNames` and `ColumnNames`. `RowNames` and `ColumnNames` can be a numeric vector, character array, string vector, or cell array of character vectors, whose elements are equal in number to the number of rows and number of columns, respectively, in `Matrix`. `RowNames` are typically gene names or probe identifiers, while `ColumnNames` are typically sample identifiers.

The row or column names do not need to be unique.

```matlab
DMobj = DataMatrix('File', FileName)
```
creates a DataMatrix object, `DMobj`, from `FileName`, a character vector or string specifying a file name or a path and file name of a tab-delimited TXT or XLS file that contains table-oriented data and metadata.

**Note** Typically, the first row of the table contains column names, the first column contains row names, and the numeric data starts at the 2,2 position. The `DataMatrix` function will detect if the first column does not contain row names, and read data from the first column. However, if the first row does not contain header text (column names), set the `HLine` property to 0.

```matlab
DMobj = DataMatrix(..., 'PropertyName', PropertyValue, ...)
```
calls `DataMatrix` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

```matlab
DMobj = DataMatrix(..., 'RowNames', RowNamesValue, ...)
```
specifies row names for `DMobj`. `RowNamesValue` can be any of the following:

- Numeric vector, character array, string vector, or a cell array of character vectors, whose elements are equal in number to the number of rows of numeric data in the input matrix.
- A character vector or string, which is used as a prefix for row names. Row numbers will be appended to the prefix.
- `true` — Unique row names will be assigned using the format `row1`, `row2`, `row3`, etc.
- `false` — Default. No row names are assigned.

**Note** The row names do not need to be unique.

```matlab
DMobj = DataMatrix(..., 'ColNames', ColNamesValue, ...)
```
specifies column names for `DMobj`. `ColNamesValue` can be any of the following:
• Numeric vector, character array, string vector, or a cell array of character vectors, whose elements are equal in number to the number of columns of numeric data in the input matrix.
• A character vector or string, which is used as a prefix for column names. Column numbers will be appended to the prefix.
• true — Unique column names will be assigned using the format col1, col2, col3, etc.
• false — Default. No column names are assigned.

**Note** The column names do not need to be unique.

\[
\text{DMobj} = \text{DataMatrix}(\ldots, \text{'Name'}, \text{NameValue}, \ldots) \text{ specifies a name for DMobj. Default is } \text{''}. \\
\text{DMobj} = \text{DataMatrix}('File', \text{FileName}, \ldots \text{'Delimiter'}, \text{DelimiterValue}, \ldots) \\
\text{specifies a delimiter symbol to use for the input file. Typical choices are:} \\
\begin{itemize}
  \item '\t'
  \item '\t' (default)
  \item '
  \item ','
  \item ';'
  \item '|'
\end{itemize}
\text{DMobj} = \text{DataMatrix}('File', \text{FileName}, \ldots \text{'HLine'}, \text{HLineValue}, \ldots) \text{ specifies which row of the input file contains the column header text (column names). HLineValue is a positive integer. Default is 1. When creating the DataMatrix object DMobj, the DataMatrix function loads data from (HLineValue + 1) to the end of the file.} \\
\text{Tip} \text{ If the input file does not contain column header text (column names), set HLineValue to 0.}
\text{DMobj} = \text{DataMatrix}('File', \text{FileName}, \ldots \text{'Rows'}, \text{RowsValue}, \ldots) \text{ specifies a subset of row names in File for the DataMatrix function to use to create DMobj. RowsValue can be a cell array of character vectors, a character array, string vector, or a numeric or logical vector.}
\text{DMobj} = \text{DataMatrix}('File', \text{FileName}, \ldots \text{'Columns'}, \text{ColumnsValue}, \ldots) \text{ specifies a subset of column names in File for the DataMatrix function to use to create DMobj. ColumnsValue can be a cell array of character vectors, string vector, character array, or numeric or logical vector.}

**Examples**

For examples of creating and using DataMatrix objects, see “Representing Expression Data Values in DataMatrix Objects”.

**See Also**

colnames | disp | dmmatrix | dmmatrixfun | dmsxfun | dmwrite | double | eq | ge | get | gt | horzcat | isequal | isequaln | ldivide | le | lt | max | mean | median | min | minus | ndims | ne | numel | plot | plus | power | rdivide | rownames | set | single | sortcols | sortrows | std | sum | times | var | vertcat
Topics
DataMatrix object on page 1-532

Introduced in R2008b
dayhoff

Return Dayhoff scoring matrix

Syntax

ScoringMatrix = dayhoff

Description

ScoringMatrix = dayhoff returns a PAM250 type scoring matrix. The order of amino acids in the matrix is A R N D C Q E G H I L K M F P S T W Y V B Z X *.

See Also

blosum | gonnet | localalign | nuc44 | nalign | pam | swalign

Introduced before R2006a
dimercount

Count dimers in nucleotide sequence

Syntax

\[
\text{Dimers} = \text{dimercount}(\text{SeqNT}) \\
[\text{Dimers, Percent}] = \text{dimercount}(\text{SeqNT}) \\
\ldots = \text{dimercount}(\text{SeqNT}, \text{'Ambiguous'}, \text{AmbiguousValue}) \\
\ldots = \text{dimercount}(\text{SeqNT}, \text{'Chart'}, \text{ChartValue})
\]

Input Arguments

<table>
<thead>
<tr>
<th>\text{SeqNT}</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers.</td>
</tr>
<tr>
<td></td>
<td>• Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a \text{Sequence} field that contains a nucleotide sequence, such as returned by \text{fastaread}, \text{fastqread}, \text{emblread}, \text{getembl}, \text{genbankread}, or \text{getgenbank}.</td>
</tr>
</tbody>
</table>

Examples: 'ACGT' or [1 2 3 4]

<table>
<thead>
<tr>
<th>\text{AmbiguousValue}</th>
<th>Character vector or string specifying how to treat dimers containing ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N). Choices are:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• 'ignore' (default) — Skips dimers containing ambiguous characters</td>
</tr>
<tr>
<td></td>
<td>• 'bundle' — Counts dimers containing ambiguous characters and reports the total count in the \text{Ambiguous} field of the \text{Dimers} output structure.</td>
</tr>
<tr>
<td></td>
<td>• 'prorate' — Counts dimers containing ambiguous characters and distributes them proportionately in the appropriate dimer fields containing standard nucleotide characters. For example, the counts for the dimer AR are distributed evenly between the AA and AG fields.</td>
</tr>
<tr>
<td></td>
<td>• 'warn' — Skips dimers containing ambiguous characters and displays a warning.</td>
</tr>
</tbody>
</table>

| \text{ChartValue} | Character vector or string specifying a chart type. Choices are 'pie' or 'bar'. |
Output Arguments

<table>
<thead>
<tr>
<th>Dimers</th>
<th>MATLAB structure containing the fields AA, AC, AG, AT, CA, CC, CG, CT, GA, GC, GG, GT, TA, TC, TG, and TT, which contain the dimer counts in SeqNT.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percent</td>
<td>A 4-by-4 matrix with the relative proportions of the dimers in SeqNT. The rows correspond to A, C, G, and T in the first element of the dimer, and the columns correspond to A, C, G, and T in the second element of the dimer.</td>
</tr>
</tbody>
</table>

Description

Dimers = dimercount(SeqNT) counts the nucleotide dimers in SeqNT, a nucleotide sequence, and returns the dimer counts in Dimers, a MATLAB structure containing the fields AA, AC, AG, AT, CA, CC, CG, CT, GA, GC, GG, GT, TA, TC, TG, and TT.

- For sequences that have dimers with the character U, these dimers are added to the corresponding dimers containing a T.
- If the sequence contains gaps indicated by a hyphen (-), the gaps are ignored, and the two characters on either side of the gap are counted as a dimer.
- If the sequence contains unrecognized characters, then dimers containing these characters are ignored, and the following warning message appears:

  Warning: Unknown symbols appear in the sequence. These will be ignored.

[Dimers, Percent] = dimercount(SeqNT) returns Percent, a 4-by-4 matrix with the relative proportions of the dimers in SeqNT. The rows correspond to A, C, G, and T in the first element of the dimer, and the columns correspond to A, C, G, and T in the second element of the dimer.

... = dimercount(SeqNT, 'Ambiguous', AmbiguousValue) specifies how to treat dimers containing ambiguous nucleotide characters. Choices are:

- 'ignore' (default)
- 'bundle'
- 'prorate'
- 'warn'

... = dimercount(SeqNT, 'Chart', ChartValue) creates a chart showing the relative proportions of the dimers. ChartValue can be 'pie' or 'bar'.

Examples

Count dimers in a nucleotide sequence

seq = randseq(100)

seq =
'TTATGACGTTATCTCTCTTATGAGAATGCTACCTACGCCGGAACTCGATCGGTTCGAACTCTATCACGCCTGGTCTTCGAAGTTAGCAC'

[Dimers, Percent] = dimercount(seq)

Dimers = struct with fields:
  AA: 4

Percent = 4x4

0.0404  0.0909  0.0303  0.0606
0.0303  0.0303  0.0808  0.0909
0.0808  0.0404  0.0404  0.0606
0.0707  0.0808  0.0707  0.1010

See Also
aacount | basecount | baselookup | codoncount | nmercount | ntdensity

Introduced before R2006a
disp (DataMatrix)

Display DataMatrix object

Syntax

disp(DMObj)

Arguments

| DMObj          | DataMatrix object, such as created by DataMatrix (object constructor). |

Description

disp(DMObj) displays the DataMatrix object DMObj, including row names and column names, without printing the DataMatrix object name.

See Also

DataMatrix

Topics

DataMatrix object on page 1-532

Introduced in R2008b
# dmarrayfun (DataMatrix)

Apply function to each element in DataMatrix object

## Syntax

\[
DMObjNew1 = dmarrayfun(Func, DMObj1) \\
DMObjNew1 = dmarrayfun(Func, DMObj1, DMObj2, ...) \\
[DMObjNew1, DMObjNew2, ...] = dmarrayfun(Func, DMObj1, ...) \\
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'UniformOutput', UniformOutputValue, ...) \\
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'DataMatrixOutput', DataMatrixOutputValue, ...) \\
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'Rows', RowsValue, ...) \\
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'Columns', ColumnsValue, ...) \\
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'ErrorHandler', ErrorFuncHandle, ...) \\
\]

## Input Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Func</strong></td>
<td>Function handle for a function that returns one or more scalars, and returns values of the same class each time it is called.</td>
</tr>
<tr>
<td><strong>DMObj1</strong></td>
<td>DataMatrix object, such as created by DataMatrix (object constructor).</td>
</tr>
<tr>
<td><strong>DMObj2</strong></td>
<td>Either of the following:</td>
</tr>
<tr>
<td></td>
<td>• DataMatrix object, such as created by DataMatrix (object constructor)</td>
</tr>
<tr>
<td></td>
<td>• MATLAB numeric array</td>
</tr>
</tbody>
</table>

**Note** DMObj2 and subsequent input objects or arrays must be the same size (number of rows and columns) as DMObj1.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UniformOutputValue</strong></td>
<td>Specifies whether Func must return output values without encapsulation in a cell array. Choices are true (default) or false. If true, dmarrayfun must return scalar values that can be concatenated into an array. These values can also be a cell array. If false, dmarrayfun returns a cell array (or multiple cell arrays), where the I,Jth cell contains the value equal to Func(DMObj1(I,J),...).</td>
</tr>
<tr>
<td><strong>DataMatrixOutputValue</strong></td>
<td>Specifies whether return values must be DataMatrix objects. Choices are true (default) or false. If you set the 'UniformOutput' property to false, this property is ignored.</td>
</tr>
</tbody>
</table>
**RowsValue, ColumnsValue**

Specifies the rows or columns to which to apply the function. Choices are:
- Positive integer
- Vector of positive integers
- Character vector specifying a row or column name
- Cell array of character vectors
- Logical vector

**ErrorFuncHandle**

Specifies a function handle to a function that `dmarrayfun` calls if the call to `Func` fails.

## Output Arguments

| DMObjNew1, DMObjNew2 | DataMatrix objects created from applying the function to each element in one or more DataMatrix objects. The size (number of rows and columns), row names, and column names will be the same as `DMObj1`. |

## Description

DMObjNew1 = `dmarrayfun(Func, DMObj1)` applies the function specified by `Func` to each element in `DMObj1`, a DataMatrix object, and returns the results in `DMObjNew1`, a new DataMatrix object. `DMObjNew1` has the same size (number of rows and columns), row names, and column names as `DMObj1`. The I,Jth element of `DMObjNew1` is equal to `Func(DMObj1(I,J))`, where `Func` is a function handle for a function that takes one input argument, returns one scalar value, and returns values of the same class each time it is called.

DMObjNew1 = `dmarrayfun(Func, DMObj1, DMObj2, ...)` evaluates the function specified by `Func` using elements in `DMObj1, DMObj2, etc. as input arguments. The I,Jth element of `DMObjNew1` is equal to `Func(DMObj1(I,J), DMObj2(I,J), ...))`, where `Func` is a function handle for a function that takes multiple input arguments, returns one scalar, and returns values of the same class each time it is called.

[DMObjNew1, DMObjNew2, ...] = `dmarrayfun(Func, DMObj1, ...)` evaluates the function specified by `Func` using elements in `DMObj1`, and possibly other input arguments. `Func` is a function handle for a function that takes one or more input arguments, returns multiple scalars, and returns values of the same class each time it is called. It returns DataMatrix objects `DMObjNew1, DMObjNew2, etc. with each one corresponding to one of the outputs of `Func`. The outputs of `Func` may be of different classes, however; but each output must be the same each time it is called.

[DMObjNew1, ...] = `dmarrayfun(Func, DMObj1, ...'PropertyName', PropertyValue, ...)` calls `dmarrayfun` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

- `'UniformOutput'`, `UniformOutputValue`, ... specifies whether `Func` must return output values without encapsulation in a cell array. Choices are `true` (default) or `false`. If `true`, `dmarrayfun` must return scalar values that can be concatenated into an array. These values can also be a cell array. If `false`,
dmarrayfun returns a cell array (or multiple cell arrays), where the I,Jth cell contains the value equal to \( \text{Func}(\text{DMObj1}(I,J),\ldots) \).

\[
[\text{DMObjNew1}, \ldots] = \text{dmarrayfun}(\text{Func}, \text{DMObj1}, \ldots 'DataMatrixOutput', \text{DataMatrixOutputValue}, \ldots)
\]

specifies whether return values must be DataMatrix objects. Choices are true (default) or false. If you set the 'UniformOutput' property to false, this property is ignored.

\[
[\text{DMObjNew1}, \ldots] = \text{dmarrayfun}(\text{Func}, \text{DMObj1}, \ldots 'Rows', \text{RowsValue}, \ldots)
\]
applies the function only to the rows in the DataMatrix object specified by \text{RowsValue}, which can be a positive integer, vector of positive integers, character vector specifying a row name, cell array of character vectors, or a logical vector.

\[
[\text{DMObjNew1}, \ldots] = \text{dmarrayfun}(\text{Func}, \text{DMObj1}, \ldots 'Columns', \text{ColumnsValue}, \ldots)
\]
applies the function only to the columns in the DataMatrix object specified by \text{ColumnsValue}, which can be a positive integer, vector of positive integers, character vector specifying a column name, cell array of character vectors, or a logical vector.

\[
[\text{DMObjNew1}, \ldots] = \text{dmarrayfun}(\text{Func}, \text{DMObj1}, \ldots 'ErrorHandler', \text{ErrorFuncHandle}, \ldots)
\]
specifies a function handle to a function that \text{dmarrayfun} calls if the call to \text{Func} fails. The error handling function will be called with these input arguments:

- Structure with the following fields:
  - identifier — Identifier of the error
  - message — Error message text
  - index — Linear index into the input array(s) at which the error occurred
- Set of input arguments at which the call to the function failed

If you do not specify \text{ErrorFuncHandle}, \text{dmarrayfun} rethrows the error from the call to \text{Func}.

**See Also**
DataMatrix | arrayfun | dmbsxfun

**Topics**
DataMatrix object on page 1-532

**Introduced in R2008b**
dmbsxfun (DataMatrix)

Apply element-by-element binary operation to two DataMatrix objects with singleton expansion enabled

Syntax

\[ \text{DMObjNew} = \text{dmbsxfun}(\text{Func}, \text{DMObj1}, \text{DMObj2}) \]

Input Arguments

| \( \text{Func} \) | Function handle for a function or a built-in function. For more information on built-in functions, see \text{bsxfun}. |
| \( \text{DMObj1}, \text{DMObj2} \) | Either of the following:  
* DataMatrix object, such as created by DataMatrix (object constructor)  
* MATLAB numeric array  
At least one of these input arguments must be a DataMatrix object. |

Output Arguments

| \( \text{DMObjNew} \) | DataMatrix object or MATLAB numeric array created from element-by-element binary operation of two DataMatrix objects with singleton expansion enabled. |

Description

\( \text{DMObjNew} = \text{dmbsxfun}(\text{Func}, \text{DMObj1}, \text{DMObj2}) \) applies an element-by-element binary operation to the DataMatrix objects \( \text{DMObj1} \) and \( \text{DMObj2} \), with singleton expansion enabled. \( \text{Func} \) is a function handle, and can be for a function or a built-in function. For more information on built-in functions, see \text{bsxfun}.

\( \text{DMObj1} \) and \( \text{DMObj2} \) can be DataMatrix objects or MATLAB numeric arrays; however, at least one of these input arguments must be a DataMatrix object. \( \text{DMObj1} \) and \( \text{DMObj2} \) must have the same number of rows or the same number or columns. If they don't have the same number of rows, then one must be a row vector and its rows are expanded down to be equal to the larger matrix. If they don't have the same number of columns, then one must be a column vector and its columns are expanded across to be equal to the larger matrix.

\( \text{DMObjNew} \) is a DataMatrix object, unless the larger input argument is a MATLAB numeric array; then \( \text{DMObjNew} \) is also a numeric array. The size (number of rows and columns) of \( \text{DMObjNew} \) is equal to the larger of the two input arguments. The row names and column names of \( \text{DMObjNew} \) come from the larger input argument, or, if both inputs are the same size, from the first input argument.
Examples

1. Use the DataMatrix constructor function to create a DataMatrix object.
   
   ```
   A = bioma.data.DataMatrix(magic(3), 'RowNames', true, ...
   'ColNames',true)
   ```

2. Use the built-in function @minus to subtract the column means from this DataMatrix object.
   
   ```
   A = dmbsxfun(@minus, A, mean(A))
   ```

See Also
DataMatrix | bsxfun

Topics
DataMatrix object on page 1-532

Introduced in R2008b
dmNames

Class: bioma.data.ExptData
Package: bioma.data

Retrieve or set Name properties of DataMatrix objects in ExptData object

Syntax

\[
\text{DMNames} = \text{dmNames}(\text{EDObj})
\]
\[
\text{DMNames} = \text{dmNames}(\text{EDObj}, \text{Subset})
\]
\[
\text{NewEDObj} = \text{dmNames}(\text{EDObj}, \text{Subset}, \text{NewDMNames})
\]

Description

\text{DMNames} = \text{dmNames}(\text{EDObj}) \text{ returns a cell array of character vectors specifying the Name properties of all the DataMatrix objects in an ExptData object.}

\text{DMNames} = \text{dmNames}(\text{EDObj}, \text{Subset}) \text{ returns a cell array of character vectors specifying the Name properties of a subset of the DataMatrix objects in an ExptData object.}

\text{NewEDObj} = \text{dmNames}(\text{EDObj}, \text{Subset}, \text{NewDMNames}) \text{ replaces the Name properties of DataMatrix objects specified by Subset in EDObj, an ExptData object, with NewDMNames, and returns NewEDObj, a new ExptData object.}

Input Arguments

\text{EDObj}

Object of the bioma.data.ExptData class.

Default:

\text{Subset}

One of the following to specify the names of a subset of the DataMatrix objects in an ExptData object:

- Character vector specifying a name
- Cell array of character vectors specifying names
- Positive integer
- Vector of positive integers
- Logical vector

Default:

\text{NewDMNames}

New names for specific DataMatrix objects within an ExptData object, specified by one of the following:
**Numeric vector**
- Character vector or cell array of character vectors
- Character vector, which `dmNames` uses as a prefix for the DataMatrix object names, with numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `dmNames` assigns unique names using the format `DM1`, `DM2`, etc.

The number of elements in `NewDMNames` must equal the number of DataMatrix objects specified by `Subset`.

**Default:**

**Output Arguments**

**DMNames**
Cell array of character vectors specifying the names of all or some of the DataMatrix objects in an ExptData object.

**NewEDObj**
Object of the `bioma.data.ExptData` class, returned after replacing names of specific DataMatrix objects.

**Examples**

Construct an ExptData object, and then retrieve the names of DataMatrix objects from it:

```matlab
% Import bioma.data package to make constructor functions available
import bioma.data.*
% Create DataMatrix object from .txt file containing expression values from microarray experiment
dmObj = DataMatrix('File', 'mouseExprsData.txt');
% Construct ExptData object
EDObj = ExptData(dmObj);
% Retrieve DataMatrix object names
DMNames = dmNames(EDObj);
```

**See Also**
DataMatrix | `bioma.data.ExptData` | `elementNames` | `featureNames` | `sampleNames`

**Topics**
“Representing Expression Data Values in ExptData Objects”
dmwrite (DataMatrix)

Write DataMatrix object to text file

Syntax

dmwrite(DMObj, File)
dmwrite(..., 'Delimiter', DelimiterValue, ...)
dmwrite(..., 'Precision', PrecisionValue, ...)
dmwrite(..., 'Header', HeaderValue, ...)
dmwrite(..., 'Annotated', AnnotatedValue, ...)
dmwrite(..., 'Append', AppendValue, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMObj</td>
<td>DataMatrix object, such as created by DataMatrix (object constructor).</td>
</tr>
<tr>
<td>File</td>
<td>Character vector specifying either a file name or a path and file name for saving the text file.</td>
</tr>
<tr>
<td>DelimiterValue</td>
<td>Character vector specifying a delimiter symbol to use as a matrix column separator. Typical choices are:</td>
</tr>
<tr>
<td></td>
<td>• <code>' '</code></td>
</tr>
<tr>
<td></td>
<td>• <code>'\t'</code> (default)</td>
</tr>
<tr>
<td></td>
<td>• <code>','</code></td>
</tr>
<tr>
<td></td>
<td>• <code>';'</code></td>
</tr>
<tr>
<td></td>
<td>• `'</td>
</tr>
<tr>
<td>PrecisionValue</td>
<td>Precision for writing the data to the text file, specified by either:</td>
</tr>
<tr>
<td></td>
<td>• Positive integer specifying the number of significant digits</td>
</tr>
<tr>
<td></td>
<td>• C-style format character vector starting with %, such as <code>%6.5f</code></td>
</tr>
<tr>
<td></td>
<td>Default is 5.</td>
</tr>
<tr>
<td>HeaderValue</td>
<td>Character vector specifying the first line of the text file. Default is the Name property for the DataMatrix object.</td>
</tr>
<tr>
<td>AnnotatedValue</td>
<td>Controls the writing of row and column names to the text file. Choices are true (default) or false.</td>
</tr>
<tr>
<td>AppendValue</td>
<td>Controls the appending of DMObj to File when it is an existing file. Choices are true or false (default). If false, dmwrite overwrites File.</td>
</tr>
</tbody>
</table>

Description

dmwrite(DMObj, File) writes a DataMatrix object to a text file using the delimiter `\t` to separate DataMatrix columns. dmwrite writes the data starting at the first column of the first row in the destination file.
dmwrite(..., 'PropertyName', PropertyValue, ...) calls dmwrite with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each PropertyName in single quotation marks. Each PropertyName is case insensitive. These property name/property value pairs are as follows:

```
dmwrite(..., 'Delimiter', DelimiterValue, ...) specifies a delimiter symbol to use as a column separator for separating matrix columns. Default is '	'.

dmwrite(..., 'Precision', PrecisionValue, ...) specifies the precision for writing the data to the text file. Default is 5.

dmwrite(..., 'Header', HeaderValue, ...) specifies the first line of the text file. Default is the Name property for the DataMatrix object.

dmwrite(..., 'Annotated', AnnotatedValue, ...) controls the writing of row and column names to the text file. Choices are true (default) or false.

dmwrite(..., 'Append', AppendValue, ...) controls the appending of DMObj to File when it is an existing file. Choices are true or false (default). If false, dmwrite overwrites File.
```

**Examples**

Create a DataMatrix object and write the contents to a text file:

```
% Create a DataMatrix object
dmobj = bioma.data.DataMatrix(rand(2,3), {'Row1', 'Row2'}, {'Col1', 'Col2', 'Col3'})
% Write the DataMatrix object to a text file
dmwrite(dmobj,'testdm.txt')
```

**See Also**

DataMatrix

**Topics**

DataMatrix object on page 1-532

**Introduced in R2009b**
**dna2rna**

Convert DNA sequence to RNA sequence

**Syntax**

```matlab
SeqRNA = dna2rna(SeqDNA)
```

**Arguments**

<table>
<thead>
<tr>
<th><code>SeqDNA</code></th>
<th>DNA sequence specified by any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string with the characters A, C, G, T, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N,</td>
</tr>
<tr>
<td></td>
<td>• Row vector of integers from the table Mapping Nucleotide Integers to Letter Codes.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a <code>Sequence</code> field that contains a DNA sequence, such as returned by <code>fastaread</code>, <code>fastqread</code>, <code>emblread</code>, <code>getembl</code>, <code>genbankread</code>, or <code>getgenbank</code>.</td>
</tr>
</tbody>
</table>

**Description**

`SeqRNA = dna2rna(SeqDNA)` converts a DNA sequence to an RNA sequence by converting any thymine nucleotides (T) in the DNA sequence to uracil nucleotides (U). The RNA sequence is returned in the same format as the DNA sequence. For example, if `SeqDNA` is a vector of integers, then so is `SeqRNA`.

**Examples**

Convert a DNA sequence to an RNA sequence

```matlab
dna = randseq(100)
dna =
'TTATGACGTATTTGATTGTGCGAGACATTATGCTACCTACGTCGGGAACCTCGATCGGTTGAACTCTATCACGCCTGGTCTTCGAAGTTAGCAC'
rna = dna2rna(dna)
rna =
'UUUAUGACGUUAUCUACUUUGAUGUUGCGAGACAAUGCUACCUACCUGUGCCGAAACUCGAUCGUUGAAGAACUACACGCGCCUGGUCUUCGAAGUU'
```

**See Also**

`regexp` | `rna2dna` | `strrep`

```
Introduced before R2006a
```

1-558
dnds

Estimate synonymous and nonsynonymous substitution rates

Syntax

\[
[Dn, Ds, Vardn, Vards] = \text{dnds}(\text{SeqNT1, SeqNT2})
\]

\[
[Dn, Ds, Vardn, Vards] = \text{dnds}(\text{SeqNT1, SeqNT2}, ...'GeneticCode', \text{GeneticCodeValue}, ...)
\]

\[
[Dn, Ds, Vardn, Vards] = \text{dnds}(\text{SeqNT1, SeqNT2}, ...'Method', \text{MethodValue}, ...)
\]

\[
[Dn, Ds, Vardn, Vards] = \text{dnds}(\text{SeqNT1, SeqNT2}, ...'Window', \text{WindowValue}, ...)
\]

\[
[Dn, Ds, Vardn, Vards] = \text{dnds}(\text{SeqNT1, SeqNT2}, ...'AdjustStops', \text{AdjustStopsValue}, ...)
\]

\[
[Dn, Ds, Vardn, Vards] = \text{dnds}(\text{SeqNT1, SeqNT2}, ...'Verbose', \text{VerboseValue}, ...)
\]

Input Arguments

<table>
<thead>
<tr>
<th>SeqNT1, SeqNT2</th>
<th>Nucleotide sequences. Enter a character vector, string, or a structure with the field Sequence.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeneticCodeValue</td>
<td>Property to specify a genetic code. Enter a Code Number, a character vector, or string with a Code Name from the table Genetic Code. If you use a Code Name, you can truncate it to the first two characters. Default is 1 or Standard.</td>
</tr>
<tr>
<td>MethodValue</td>
<td>Character vector or string specifying the method for calculating substitution rates. Choices are:</td>
</tr>
<tr>
<td>WindowValue</td>
<td>Integer specifying the sliding window size, in codons, for calculating substitution rates and variances.</td>
</tr>
<tr>
<td>AdjustStopsValue</td>
<td>Controls whether stop codons are excluded from calculations. Choices are true (default) or false.</td>
</tr>
</tbody>
</table>
VerboseValue | Property to control the display of the codons considered in the computations and their amino acid translations. Choices are true or false (default).

**Tip** Specify true to use this display to manually verify the codon alignment of the two input sequences. The presence of stop codons (*) in the amino acid translation can indicate that SeqNT1 and SeqNT2 are not codon-aligned.

**Output Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dn</td>
<td>Nonsynonymous substitution rate(s).</td>
</tr>
<tr>
<td>Ds</td>
<td>Synonymous substitution rate(s).</td>
</tr>
<tr>
<td>Vardn</td>
<td>Variance for the nonsynonymous substitution rate(s).</td>
</tr>
<tr>
<td>Vards</td>
<td>Variance for the synonymous substitutions rate(s).</td>
</tr>
</tbody>
</table>

**Description**

\[ [Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2) \] estimates the synonymous and nonsynonymous substitution rates per site between the two homologous nucleotide sequences, SeqNT1 and SeqNT2, by comparing codons using the Nei-Gojobori method.

dnds returns:

- \( Dn \) — Nonsynonymous substitution rate(s).
- \( Ds \) — Synonymous substitution rate(s).
- \( Vardn \) — Variance for the nonsynonymous substitution rate(s).
- \( Vards \) — Variance for the synonymous substitutions rate(s).

This analysis:

- Assumes that the nucleotide sequences, SeqNT1 and SeqNT2, are codon-aligned, that is, do not have frame shifts.

**Tip** If your sequences are not codon-aligned, use the nt2aa function to convert them to amino acid sequences, use the nalign function to globally align them, then use the seqinsertgaps function to recover the corresponding codon-aligned nucleotide sequences. For an example, see “Estimate synonymous and nonsynonymous substitution rates between two nucleotide sequences” on page 1-561.

- Excludes codons that include ambiguous nucleotide characters or gaps
- Considers the number of codons in the shorter of the two nucleotide sequences

**Caution** If SeqNT1 and SeqNT2 are too short or too divergent, saturation can be reached, and dnds returns NaNs and a warning message.

\[ [Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ...'PropertyName', PropertyValue, ...) \] calls dnds with optional properties that use property name/property value.
pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

\[
[Dn, Ds, Vardn, Vards] = \text{dnds}((\text{SeqNT1, SeqNT2, ...'), 'GeneticCode', GeneticCodeValue, ...})
\]
calculates synonymous and nonsynonymous substitution rates using the specified genetic code. Enter a Code Number, a character vector or string with a Code Name from the table Genetic Code. If you use a Code Name, you can truncate it to the first two characters. Default is 1 or Standard.

\[
[Dn, Ds, Vardn, Vards] = \text{dnds}((\text{SeqNT1, SeqNT2, ...'), 'Method', MethodValue, ...})
\]
allows you to calculate synonymous and nonsynonymous substitution rates using the following algorithms:

- NG (default) — Nei-Gojobori method (1986) uses the number of synonymous and nonsynonymous substitutions and the number of potentially synonymous and nonsynonymous sites. Based on the Jukes-Cantor model.
- LWL — Li-Wu-Luo method (1985) uses the number of transitional and transversional substitutions at three different levels of degeneracy of the genetic code. Based on Kimura's two-parameter model.
- PBL — Pamilo-Bianchi-Li method (1993) is similar to the Li-Wu-Luo method, but with bias correction. Use this method when the number of transitions is much larger than the number of transversions.

\[
[Dn, Ds, Vardn, Vards] = \text{dnds}((\text{SeqNT1, SeqNT2, ...'), 'Window', WindowValue, ...})
\]
performs the calculations over a sliding window, specified in codons. Each output is an array containing a rate or variance for each window.

\[
[Dn, Ds, Vardn, Vards] = \text{dnds}((\text{SeqNT1, SeqNT2, ...'), 'AdjustStops', AdjustStopsValue, ...})
\]
controls whether stop codons are excluded from calculations. Choices are true (default) or false.

Tip When the 'AdjustStops' property is set to true, the following are true:

- Stop codons are excluded from frequency tables.
- Paths containing stop codons are not counted in the Nei-Gojobori method.

\[
[Dn, Ds, Vardn, Vards] = \text{dnds}((\text{SeqNT1, SeqNT2, ...'), 'Verbose', VerboseValue, ...})
\]
controls the display of the codons considered in the computations and their amino acid translations. Choices are true or false (default).

Tip Specify true to use this display to manually verify the codon alignment of the two input sequences, \text{SeqNT1} and \text{SeqNT2}. The presence of stop codons (*) in the amino acid translation can indicate that \text{SeqNT1} and \text{SeqNT2} are not codon-aligned.

Examples
Estimate synonymous and nonsynonymous substitution rates between two nucleotide sequences

This example shows how to estimate synonymous and nonsynonymous substitution rates between two nucleotide sequences that are not codon-aligned.

This example uses two nucleotide sequences representing the human HEXA gene (accession number: NM_000520) and mouse HEXA gene (accession number: AK080777).

If you have live internet connection, you can use `getgenbank` function to retrieve the sequence information from the NCBI data repository and load the data into MATLAB®.

```matlab
humanHEXA = getgenbank('NM_000520');
mouseHEXA = getgenbank('AK080777');
```

For your convenience, MATLAB provides these two sequences in the following mat file. Note that data in public databases are frequently updated and curated, and the results in this example may slightly differ if you use the latest data.

```matlab
load hexosaminidase.mat
```

Extract the coding regions from the two nucleotide sequences.

```matlab
humanHEXA_cds = featureparse(humanHEXA,'feature','CDS','Sequence',true); mouseHEXA_cds = featureparse(mouseHEXA,'feature','CDS','Sequence',true);
```

Align the amino acid sequences converted from the nucleotide sequences.

```matlab
[sc,al] = nwalign(nt2aa(humanHEXA_cds),nt2aa(mouseHEXA_cds),'extendgap',1);
```

Use the `seqinsertgaps` function to copy the gaps from the aligned amino acid sequences to their corresponding nucleotide sequences, thus codon-aligning them.

```matlab
humanHEXA_aligned = seqinsertgaps(humanHEXA_cds,al(1,:))
mouseHEXA_aligned = seqinsertgaps(mouseHEXA_cds,al(3,:))
```

Estimate the synonymous and nonsynonymous substitutions rates of the codon-aligned nucleotide sequences and also display the codons considered in the computations and their amino acid translations.

```matlab
[nonsynSubRate,synSubRate] = dnds(humanHEXA_aligned,mouseHEXA_aligned,'verbose',true)
```

DNDS:

Codons considered in the computations:

| ATGACAAGCTCAGGCGCTTGGTTTTTCGCTGTGCTGCTGGCGGCAGCGTTCGCAGGGACGGCCCTCTGGCCCTGGCCCTGAAGTCCAGTACCATCAGCACACCCAGTAC | 0.0933 |

Translations:


nonsynSubRate = 0.0933
References


See Also
featureparse | nwalign | seqinsertgaps

Topics
dndsml on page 1-564
geneticcode on page 1-678
nt2aa on page 1-1219
seqpdist on page 1-1522

Introduced before R2006a
dndsml

Estimate synonymous and nonsynonymous substitution rates using maximum likelihood method

Syntax

\[
[Dn, Ds, Like] = \text{dndsml}(\text{SeqNT1, SeqNT2})
\]

\[
[Dn, Ds, Like] = \text{dndsml}(\text{SeqNT1, SeqNT2, ...'GeneticCode', GeneticCodeValue, ...})
\]

\[
[Dn, Ds, Like] = \text{dndsml}(\text{SeqNT1, SeqNT2, ...'Verbose', VerboseValue, ...})
\]

Input Arguments

<table>
<thead>
<tr>
<th>SeqNT1, SeqNT2</th>
<th>Nucleotide sequences. Enter a character vector, string, or a structure with the field Sequence.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeneticCodeValue</td>
<td>Property to specify a genetic code. Enter a Code Number, a character vector, or string with a Code Name from the table Genetic Code. If you use a Code Name, you can truncate it to the first two characters. Default is 1 or Standard.</td>
</tr>
<tr>
<td>VerboseValue</td>
<td>Property to control the display of the codons considered in the computations and their amino acid translations. Choices are true or false (default).</td>
</tr>
</tbody>
</table>

Tip Specify true to use this display to manually verify the codon alignment of the two input sequences. The presence of stop codons (*) in the amino acid translation can indicate that SeqNT1 and SeqNT2 are not codon-aligned.

Output Arguments

| Dn | Nonsynonymous substitution rate(s). |
| Ds | Synonymous substitution rate(s). |
| Like | Likelihood of estimate of substitution rates. |

Description

\([Dn, Ds, Like] = \text{dndsml}(\text{SeqNT1, SeqNT2})\) estimates the synonymous and nonsynonymous substitution rates between the two homologous sequences, SeqNT1 and SeqNT2, using the Goldman-Yang method (1994) on page 1-567. This maximum likelihood method estimates an explicit model for codon substitution that accounts for transition/transversion rate bias and base/codon frequency bias. Then it uses the model to correct synonymous and nonsynonymous counts to account for multiple substitutions at the same site. The maximum likelihood method is best suited when the sample size is significant (larger than 100 bases) and when the sequences being compared can have transition/transversion rate biases and base/codon frequency biases.

dndsml returns:
• **Dn** — Nonsynonymous substitution rate(s).
• **Ds** — Synonymous substitution rate(s).
• **Like** — Likelihood of this estimate.

This analysis:

- Assumes that the nucleotide sequences, *SeqNT1* and *SeqNT2*, are codon-aligned, that is, do not have frame shifts.

**Tip** If your sequences are not codon-aligned, use the `nt2aa` function to convert them to amino acid sequences, use the `nwalign` function to globally align them, then use the `seqinsertgaps` function to recover the corresponding codon-aligned nucleotide sequences. For an example, see “Estimate synonymous and nonsynonymous substitution rates between two nucleotide sequences using maximum likelihood method” on page 1-565.

- Excludes any ambiguous nucleotide characters or codons that include gaps.
- Considers the number of codons in the shorter of the two nucleotide sequences.

**Caution** If *SeqNT1* and *SeqNT2* are too short or too divergent, saturation can be reached, and `dndsml` returns NaNs and a warning message.

```matlab
[Dn, Ds, Like] = dndsml(SeqNT1, SeqNT2, ...'PropertyName', PropertyValue, ...)```
calls `dnds` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

```matlab
[Dn, Ds, Like] = dndsml(SeqNT1, SeqNT2, ...'GeneticCode', GeneticCodeValue, ...)```
calculates synonymous and nonsynonymous substitution rates using the specified genetic code. Enter a Code Number, a character vector or string with a Code Name from the table Genetic Code. If you use a Code Name, you can truncate it to the first two characters. Default is 1 or Standard.

```matlab
[Dn, Ds, Like] = dndsml(SeqNT1, SeqNT2, ...'Verbose', VerboseValue, ...)```
controls the display of the codons considered in the computations and their amino acid translations. Choices are `true` or `false` (default).

**Tip** Specify `true` to use this display to manually verify the codon alignment of the two input sequences, *SeqNT1* and *SeqNT2*. The presence of stop codons (*) in the amino acid translation can indicate that *SeqNT1* and *SeqNT2* are not codon-aligned.

### Examples

**Estimate synonymous and nonsynonymous substitution rates between two nucleotide sequences using maximum likelihood method**

This example shows how to estimate synonymous and nonsynonymous substitution rates between two nucleotide sequences that are not codon-aligned using maximum likelihood method.
This example uses two nucleotide sequences representing the human HEXA gene (accession number: NM_000520) and mouse HEXA gene (accession number: AK080777).

If you have live internet connection, you can use `getgenbank` function to retrieve the sequence information from the NCBI data repository and load the data into MATLAB®.

```matlab
humanHEXA = getgenbank('NM_000520');
mouseHEXA = getgenbank('AK080777');
```

For your convenience, MATLAB provides these two sequences in the following mat file. Note that data in public databases are frequently updated and curated, and the results in this example may slightly differ if you use the latest data.

```matlab
load hexosaminidase.mat
```

Extract the coding regions from the two nucleotide sequences.

```matlab
humanHEXA_cds = featureparse(humanHEXA,'feature','CDS','Sequence',true);
mouseHEXA_cds = featureparse(mouseHEXA,'feature','CDS','Sequence',true);
```

Align the amino acid sequences converted from the nucleotide sequences.

```matlab
[sc,al] = nwalign(nt2aa(humanHEXA_cds),nt2aa(mouseHEXA_cds),'extendgap',1);
```

Use the `seqinsertgaps` function to copy the gaps from the aligned amino acid sequences to their corresponding nucleotide sequences, thus codon-aligning them.

```matlab
humanHEXA_aligned = seqinsertgaps(humanHEXA_cds,al(1,:));
mouseHEXA_aligned = seqinsertgaps(mouseHEXA_cds,al(3,:));
```

Estimate the synonymous and nonsynonymous substitutions rates of the codon-aligned nucleotide sequences and also display the codons considered in the computations and their amino acid translations.

```matlab
[nonsynSubRate,synSubRate] = dndsml(humanHEXA_aligned,mouseHEXA_aligned,'verbose',true)
```

**DNDSML:**
Codons considered in the computations:
ATGACAAGCTCCAGGCTTTGCTGCTGGCGCGAGCGTTGCAGGCGACGCCCTCTGGCCCTGTCATGAACCTTCCAGCTCCAGGCTTTGCTGCTGGCGCGAGCGTTGCAGGCGACGCCCTCTGGCCCTGTCATGAACCTTCCAGCT:
ATGGCCGGCTGCAGGCTCTGGGTTTCGCTGCTGGCGCGGGCGAGCGTTGCAGGCGACGCCCTCTGGCCCTGTCATGAACCTTCCAGCT:
Translations:

Initial estimates: Kappa=3.301203, dn=0.093274, ds=0.518095, t=0.353716
ML estimates: Kappa=2.498253, omega(dn/ds)=0.185577, t=0.602465

nonsynSubRate = 0.0943
synSubRate = 0.5080
References


See Also
featureparse | nwallign | seqinsertgaps

Topics
dnds on page 1-559
geneticcode on page 1-678
nt2aa on page 1-1219
seqpdist on page 1-1522

Introduced before R2006a
dolayout (biograph)

Calculate node positions and edge trajectories

Syntax

dolayout(BGobj)

dolayout(BGobj, 'Paths', PathsOnlyValue)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGobj</td>
<td>Biograph object created by the biograph function (object constructor).</td>
</tr>
<tr>
<td>PathsOnlyValue</td>
<td>Controls the calculation of only the edge paths, leaving the nodes at their current positions. Choices are true or false (default).</td>
</tr>
</tbody>
</table>

Description

dolayout(BGobj) calls the layout engine to calculate the optimal position for each node so that its 2-D rendering is clean and uncluttered, and then calculates the best curves to represent the edges. The layout engine uses the following properties of the biograph object:

- LayoutType — Specifies the layout engine as 'hierarchical', 'equilibrium', or 'radial'.
- LayoutScale — Rescales the sizes of the node before calling the layout engine. This gives more space to the layout and reduces the overlapping of nodes.
- NodeAutoSize — Controls precalculating the node size before calling the layout engine. When NodeAutoSize is set to 'on', the layout engine uses the node properties FontSize and Shape, and the biograph object property LayoutScale to precalculate the actual size of each node. When NodeAutoSize is set to 'off', the layout engine uses the node property Size.

For more information on the above properties, see Properties of a Biograph Object. For an example of accessing and specifying the above properties of a biograph object, see “Create a Biograph object and specify its properties” on page 1-190.

dolayout(BGobj, 'Paths', PathsOnlyValue) controls the calculation of only the edge paths, leaving the nodes at their current positions. Choices are true or false (default).

Examples

Create a Biograph Object and Calculate Node Positions and Edge Trajectories

This example shows how to create a biograph object and calculate node positions and edge trajectories.

Create a biograph object.
\[
\begin{bmatrix}
0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\]

\texttt{cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];}

\texttt{bg = biograph(cm)}

Biograph object with 5 nodes and 9 edges.

Nodes do not have positions yet.

\texttt{bg.nodes(1).Position}

\texttt{ans =}

\[
\begin{bmatrix}
\end{bmatrix}
\]

Call the layout engine and render the graph.

\texttt{dolayout(bg);}

\texttt{bg.nodes(1).Position}

\texttt{ans = 1x2}

\[
100 \\ 206
\]

\texttt{view(bg)}
Manually modify a node position and recalculate the paths only.

```matlab
bg.nodes(1).Position = [150 150];
dolayout(bg, 'Pathonly', true);
view(bg)
```
See Also
biograph | dolayout | get | getancestors | getdescendants | getedgesbynodeid | getnodesbyid | getrelatives | set | view

Topics
biograph object on page 1-185

Introduced before R2006a
**double (DataMatrix)**

Convert DataMatrix object to double-precision array

**Syntax**

```matlab
B = double(DMObj)
B = double(DMObj, Rows)
B = double(DMObj, Rows, Cols)
```

**Input Arguments**

<table>
<thead>
<tr>
<th>DMObj</th>
<th>DataMatrix object, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows, Cols</td>
<td>Row(s) or column(s) in DMObj, specified by one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Scalar</td>
</tr>
<tr>
<td></td>
<td>• Vector of positive integers</td>
</tr>
<tr>
<td></td>
<td>• Character vector specifying a row or column name</td>
</tr>
<tr>
<td></td>
<td>• Cell array of row or column names</td>
</tr>
<tr>
<td></td>
<td>• Logical vector</td>
</tr>
</tbody>
</table>

**Output Arguments**

<table>
<thead>
<tr>
<th>B</th>
<th>MATLAB numeric array.</th>
</tr>
</thead>
</table>

**Description**

`B = double(DMObj)` converts `DMObj`, a DataMatrix object, to a double-precision array, which it returns in `B`.

`B = double(DMObj, Rows)` converts a subset of `DMObj`, a DataMatrix object, specified by `Rows`, to a double-precision array, which it returns in `B`. `Rows` can be a positive integer, vector of positive integers, character vector specifying a row name, cell array of row names, or a logical vector.

`B = double(DMObj, Rows, Cols)` converts a subset of `DMObj`, a DataMatrix object, specified by `Rows` and `Cols`, to a double-precision array, which it returns in `B`. `Cols` can be a positive integer, vector of positive integers, character vector specifying a column name, cell array of column names, or a logical vector.

**See Also**

DataMatrix | single

**Topics**

DataMatrix object on page 1-532

**Introduced in R2008b**
**elementData**

**Class:** bioma.ExpressionSet  
**Package:** bioma

Retrieve or set data element (DataMatrix object) in ExpressionSet object

**Syntax**

```markdown
DMObj = elementData(ESObj, Element)  
NewESObj = elementData(ESObj, Element, NewDMObj)
```

**Description**

`DMObj = elementData(ESObj, Element)` returns the DataMatrix object from an ExpressionSet object, specified by `Element`, a positive integer or a character vector specifying an element name.

`NewESObj = elementData(ESObj, Element, NewDMObj)` replaces the DataMatrix object specified by `Element` in `ESObj`, an ExpressionSet object, with `NewDMObj`, a new DataMatrix object, and returns `NewESObj`, a new ExpressionSet object.

**Input Arguments**

**ESObj**

Object of the `bioma.ExpressionSet` class.

**Default:**

**Element**

Element (DataMatrix object) in an ExpressionSet object, specified by either of the following:

- Positive integer
- Character vector specifying the element name

**Default:**

**NewDMObj**

Object of the `DataMatrix` class. The sample names and feature names in `NewDMObj` must match the sample names and feature names in the DataMatrix object specified by `Element`.

**Default:**

**Output Arguments**

**DMObj**

Object of the `DataMatrix` class, returned from the ExptData object of an ExpressionSet object.
NewESObj

Object of the bioma.ExpressionSet class, returned after replacing a specified data element (DataMatrix object).

Examples

Construct an ExpressionSet object, ESOBJ, as described in the “Examples” on page 1-0 section of the bioma.ExpressionSet class reference page. Extract a DataMatrix object from it:

```matlab
% Extract first DataMatrix object
ExtractedDMObj = elementData(ESObj, 1);
```

See Also

DataMatrix | bioma.ExpressionSet | bioma.data.ExptData

Topics

“Managing Gene Expression Data in Objects”
elementData

Class: bioma.data.ExptData
Package: bioma.data

Retrieve or set data element (DataMatrix object) in ExptData object

Syntax

DMObj = elementData(EDObj, Element)
NewEDObj = elementData(EDObj, Element, NewDMObj)

Description

DMObj = elementData(EDObj, Element) returns the DataMatrix object from an ExptData object, specified by Element, a positive integer or character vector specifying an element name.

NewEDObj = elementData(EDObj, Element, NewDMObj) replaces the element (DataMatrix object) specified by Element in EDObj, an ExptData object, with NewDMObj, a new DataMatrix object, and returns NewEDObj, a new ExptData object.

Input Arguments

EDObj

Object of the bioma.data.ExptData class.

Default:

Element

Element (DataMatrix object) in an ExptData object, specified by either of the following:

- Positive integer
- Character vector specifying the element name

Default:

NewDMObj

Object of the DataMatrix on page 1-532 class. The sample names and feature names in NewDMObj must match the sample names and feature names of EDObj.

Default:

Output Arguments

DMObj

Object of the DataMatrix on page 1-532 class, returned from an ExptData object.
NewEDObj

Object of the `bioma.data.ExptData` class, returned after replacing a data element (DataMatrix object).

**Examples**

Construct an ExptData object, and then extract a DataMatrix object from it:

```matlab
% Import bioma.data package to make constructor functions available
import bioma.data.*
% Create DataMatrix object from .txt file containing expression values from microarray experiment
dmObj = DataMatrix('File', 'mouseExprsData.txt');
% Construct ExptData object
EDObj = ExptData(dmObj);
% Extract first DataMatrix object
ExtractedDMObj = elementData(EDObj, 1);
```

**See Also**

`DataMatrix` | `bioma.data.ExptData`

**Topics**

“Representing Expression Data Values in ExptData Objects”
**elementNames**

**Class:** bioma.ExpressionSet  
**Package:** bioma

Retrieve or set element names of DataMatrix objects in ExpressionSet object

**Syntax**

```matlab
ElmtNames = elementNames(ESObj)
ElmtNames = elementNames(ESObj, Subset)
NewESObj = elementNames(ESObj, Subset, NewElmtNames)
```

**Description**

`ElmtNames = elementNames(ESObj)` returns a cell array of character vectors specifying the element names of all the data elements (DataMatrix objects) stored in the ExptData object in an ExpressionSet object.

`ElmtNames = elementNames(ESObj, Subset)` returns a cell array of character vectors specifying the element names of a subset of the data elements (DataMatrix objects) in the ExptData object in an ExpressionSet object.

`NewESObj = elementNames(ESObj, Subset, NewElmtNames)` replaces the element names of the data elements (DataMatrix objects) specified by `Subset` in `ESObj`, an ExpressionSet object, with `NewElmtNames`, and returns `NewESObj`, a new ExpressionSet object.

**Input Arguments**

**ESObj**
Object of the bioma.ExpressionSet class.

**Subset**
One of the following to specify the element names of a subset of the data elements (DataMatrix objects) in the ExptData object of an ExpressionSet object:

- Character vector or string specifying an element name
- Cell array of character vectors or string vector specifying element names
- Positive integer
- Vector of positive integers
- Logical vector

**NewElmtNames**
New element names for specific data elements (DataMatrix objects) within an ExpressionSet object, specified by one of the following:
- Numeric vector
- String vector, or cell array of character vectors
- Character vector or string, which `elementNames` uses as a prefix for the element names, with element numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `elementNames` assigns unique element names using the format `Elmt1`, `Elmt2`, etc.

The number of elements in `NewElmtNames` must equal the number of elements specified by `Subset`.

**Output Arguments**

**ElmtNames**

Cell array of character vectors specifying the element names of all or some of the data elements (DataMatrix objects) in the ExptData object of an ExpressionSet object.

**NewESObj**

Object of the `bioma.ExpressionSet` class, returned after replacing element names of specific data elements (DataMatrix objects).

**Examples**

Construct an ExpressionSet object, `ESObj`, as described in the “Examples” on page 1-0 section of the `bioma.ExpressionSet` class reference page. Retrieve the element names of the DataMatrix objects in it:

```matlab
% Retrieve element names of DataMatrix objects
ENames = elementNames(ESObj);
```

**See Also**

`DataMatrix` | `bioma.ExpressionSet` | `bioma.data.ExptData` | `exptData`

**Topics**

“Managing Gene Expression Data in Objects”
elementNames

Class: bioma.data.ExptData
Package: bioma.data

Retrieve or set element names of DataMatrix objects in ExptData object

Syntax

ElmtNames = elementNames(EDObj)
ElmtNames = elementNames(EDObj, Subset)
NewEDObj = elementNames(EDObj, Subset, NewElmtNames)

Description

ElmtNames = elementNames(EDObj) returns a cell array of character vectors specifying the element names of all the data elements (DataMatrix objects) stored in an ExptData object.

ElmtNames = elementNames(EDObj, Subset) returns a cell array of character vectors specifying the element names of a subset of the data elements (DataMatrix objects) stored in an ExptData object.

NewEDObj = elementNames(EDObj, Subset, NewElmtNames) replaces the element names of the data elements (DataMatrix objects) specified by Subset in EDObj, an ExptData object, with NewElmtNames, and returns NewEDObj, a new ExptData object.

Input Arguments

EDObj

Object of the bioma.data.ExptData class.

Default:

Subset

One of the following to specify the element names of a subset of the data elements (DataMatrix objects) in an ExptData object:

• Character vector specifying an element name
• Cell array of character vectors specifying element names
• Positive integer
• Vector of positive integers
• Logical vector

Default:
NewElmtNames

New element names for specific data elements (DataMatrix objects) within an ExptData object, specified by one of the following:

- Numeric vector
- Character vector or cell array of character vectors
- Character vector, which elementNames uses as a prefix for the element names, with element numbers appended to the prefix
- Logical true or false (default). If true, elementNames assigns unique element names using the format Elmt1, Elmt2, etc.

The number of elements in NewElmtNames must equal the number of elements specified by Subset.

Default:

Output Arguments

ElmtNames

Cell array of character vectors specifying the element names of all or some of the data elements (DataMatrix objects) in an ExptData object.

NewEDObj

Object of the bioma.data.ExptData class, returned after replacing element names of specific data elements (DataMatrix objects).

Examples

Construct an ExptData object, and then retrieve the element names of DataMatrix objects from it:

% Import bioma.data package to make constructor functions available
import bioma.data.*
% Create DataMatrix object from .txt file containing expression values from microarray experiment
dmObj = DataMatrix('File', 'mouseExprsData.txt');
% Construct ExptData object
EDObj = ExptData(dmObj);
% Retrieve element names of DataMatrix objects
ENames = elementNames(EDObj);

See Also

DataMatrix | bioma.data.ExptData | dmNames | featureNames | sampleNames

Topics

“Representing Expression Data Values in ExptData Objects”
emblread

Read data from EMBL file

Syntax

```matlab
EMBLData = emblread(File)
EMBLSeq = emblread (File, 'SequenceOnly', SequenceOnlyValue)
```

Input Arguments

<table>
<thead>
<tr>
<th>File</th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is an EMBL-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.</td>
</tr>
<tr>
<td></td>
<td>• Character vector or string that contains the text of an EMBL-formatted file</td>
</tr>
</tbody>
</table>

**Tip** You can use the `getembl` function with the `'ToFile'` property to retrieve data from the European Molecular Biology Laboratory (EMBL) database and create an EMBL-formatted file.

| SequenceOnlyValue | Controls the reading of only the sequence without the metadata. Choices are `true` or `false` (default). |

Output Arguments

<table>
<thead>
<tr>
<th>EMBLData</th>
<th>Structure with fields corresponding to EMBL data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMBLSeq</td>
<td>Character vector representing the sequence.</td>
</tr>
</tbody>
</table>

Description

```matlab
EMBLData = emblread(File) reads data from File, an EMBL-formatted file, and creates EMBLData, a MATLAB structure containing fields corresponding to the EMBL two-character line type code, based on release 107 of the EMBL-Bank flat file format. Each line type code is stored as a separate element in the structure. For a list of the EMBL two-character line type codes, see ftp://ftp.ebi.ac.uk/pub/databases/embl/doc/usrman.txt.
```

**Note** Topology information was not included in EMBL flat files before release 87 of the database. When reading a file created before release 87, EMBLREAD returns an empty `Identification.Topology` field.

**Note** The entry name is no longer displayed in the ID line of EMBL flat files in release 87. When reading a file created in release 87, EMBLREAD returns the accession number in the `Identification.EntryName` field.
EMBLSeq = emlread (File, 'SequenceOnly', SequenceOnlyValue) controls the reading of only the sequence without the metadata. Choices are true or false (default).

Examples

Read sequence information from EMBL file

Download the sequence information from the web and save to a file.

out = getembl('X00558','ToFile','rat_protein.txt');

Read data from the EMBL file.

seqData = emblread('rat_protein.txt')

seqData =

struct with fields:

Identification: [1x1 struct]
   Accession: 'X00558'
   SequenceVersion: 'X00558.1'
   DateCreated: '13-JUN-1985  Rel. 06, Created '
   DateUpdated: '18-APR-2005  Rel. 83, Last updated, Version 4 '
   Description: 'Rat liver apolipoprotein A-I mRNA apoa-i ...
   Keyword: 'apolipoprotein; lipoprotein; signal peptide. ...'
   OrganismSpecies: 'Rattus norvegicus Norway rat ...
   OrganismClassification: [3x75 char]
   Organelle: ''
   Reference: {[1x1 struct]}
   DatabaseCrossReference: [4x75 char]
   Comments: ''
   Assembly: ''
   Feature: [22x75 char]
   BaseCount: [1x1 struct]
   Sequence: 'agctccgggggaggtcgcccacatccttcgggatgaaagctgcag...'

See Also

fastaread|genbankread|genpeptread|getembl|pdbread|seqviewer

Introduced before R2006a
eq (DataMatrix)

Test DataMatrix objects for equality

Syntax

\[ T = \text{eq}(	ext{DMObj1}, \text{DMObj2}) \]
\[ T = \text{DMObj1} == \text{DMObj2} \]
\[ T = \text{eq}(	ext{DMObj1}, B) \]
\[ T = \text{DMObj1} == B \]
\[ T = \text{eq}(B, \text{DMObj1}) \]
\[ T = B == \text{DMObj1} \]

Input Arguments

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>MATLAB numeric or logical array.</td>
</tr>
</tbody>
</table>

Output Arguments

| T | Logical matrix of the same size as DMObj1 and DMObj2 or DMObj1 and B. It contains logical 1 (true) where elements in the first input are equal to the corresponding element in the second input, and logical 0 (false) when they are not equal. |

Description

\[ T = \text{eq}(	ext{DMObj1}, \text{DMObj2}) \] or the equivalent \[ T = \text{DMObj1} == \text{DMObj2} \] compares each element in DataMatrix object \text{DMObj1} to the corresponding element in DataMatrix object \text{DMObj2}, and returns \[ T \], a logical matrix of the same size as \text{DMObj1} and \text{DMObj2}, containing logical 1 (true) where elements in \text{DMObj1} are equal to the corresponding element in \text{DMObj2}, and logical 0 (false) when they are not equal. \text{DMObj1} and \text{DMObj2} must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). \text{DMObj1} and \text{DMObj2} can have different Name properties.

\[ T = \text{eq}(	ext{DMObj1}, B) \] or the equivalent \[ T = \text{DMObj1} == B \] compares each element in DataMatrix object \text{DMObj1} to the corresponding element in \text{B}, a numeric or logical array, and returns \[ T \], a logical matrix of the same size as \text{DMObj1} and \text{B}, containing logical 1 (true) where elements in \text{DMObj1} are equal to the corresponding element in \text{B}, and logical 0 (false) when they are not equal. \text{DMObj1} and \text{B} must have the same size (number of rows and columns), unless one is a scalar.

\[ T = \text{eq}(B, \text{DMObj1}) \] or the equivalent \[ T = B == \text{DMObj1} \] compares each element in \text{B}, a numeric or logical array, to the corresponding element in DataMatrix object \text{DMObj1}, and returns \[ T \], a logical matrix of the same size as \text{B} and \text{DMObj1}, containing logical 1 (true) where elements in \text{B} are equal to the corresponding element in \text{DMObj1}, and logical 0 (false) when they are not equal. \text{B} and \text{DMObj1} must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls \[ T = \text{eq}(X, Y) \] for the syntax \[ T = X == Y \] when \text{X} or \text{Y} is a DataMatrix object.
See Also
DataMatrix | ne

Topics
DataMatrix object on page 1-532

Introduced in R2008b
evalrasmolscript

Send RasMol script commands to Molecule Viewer window

Syntax

evalrasmolscript(FigureHandle, Command)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FigureHandle</td>
<td>Figure handle to a molecule viewer returned by the molviewer function.</td>
</tr>
<tr>
<td>Command</td>
<td>Any of the following:</td>
</tr>
<tr>
<td></td>
<td>• Character vector, string, string vector, or cell array of character vectors</td>
</tr>
<tr>
<td></td>
<td>specifying RasMol script commands. If there are multiple commands, use</td>
</tr>
<tr>
<td></td>
<td>a ; to separate them.</td>
</tr>
<tr>
<td></td>
<td>Note For a complete list of RasMol script commands, see</td>
</tr>
<tr>
<td></td>
<td><a href="https://www.stolaf.edu/academics/chemapps/jmol/docs/">https://www.stolaf.edu/academics/chemapps/jmol/docs/</a></td>
</tr>
<tr>
<td></td>
<td>• Character vector or string specifying a file name or a path and file name</td>
</tr>
<tr>
<td></td>
<td>of a text file containing Jmol script commands. If you specify only a file</td>
</tr>
<tr>
<td></td>
<td>name, that file must be on the MATLAB search path or in the MATLAB</td>
</tr>
<tr>
<td></td>
<td>Current Folder.</td>
</tr>
</tbody>
</table>

Description

evalrasmolscript(FigureHandle, Command) sends the RasMol script commands specified by Command to FigureHandle, the figure handle of a Molecule Viewer window created using the molviewer function.

Examples

Send RasMol script commands to Molecule Viewer app

Display a molecule in the Molecule Viewer app and save the figure handle to the app window.

```matlab
f = molviewer('2DHB');
```
f.HandleVisibility = 'off';

Use evalrasmolscript to send script commands to the molecule viewer that change the background to black and spin the molecule.

evalrasmolscript(f,'background white; spin')
See Also
getpdb | molviewer | pdbread | pdbwrite

Introduced in R2007a
expressions

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set Expressions DataMatrix object from ExpressionSet object

Syntax

ExpressionsDMObj = expressions(ESObj)
NewESObj = expressions(ESObj, NewDMObj)

Description

ExpressionsDMObj = expressions(ESObj) returns the Expressions element (DataMatrix object), which contains expression values, from an ExpressionSet object.

NewESObj = expressions(ESObj, NewDMObj) replaces the Expressions element (DataMatrix object) in ESObj, an ExpressionSet object, with NewDMObj, a new DataMatrix object, and returns NewESObj, a new ExpressionSet object.

Input Arguments

ESObj
Object of the bioma.ExpressionSet class.

Default:

NewDMObj
Object of the DataMatrix on page 1-532 class.

Default:

Output Arguments

ExpressionsDMObj
DataMatrix object containing the expression values from the Expressions DataMatrix object within an ExpressionSet object.

NewESObj
ExpressionSet object returned after replacing the Expressions DataMatrix object.

Examples

Construct an ExpressionSet object, ESObj, as described in the “Examples” on page 1-0 section of the bioma.ExpressionSet class reference page. Extract the Expressions DataMatrix object from it:
% Extract expression values from Expressions DataMatrix object
ExpressionsDMObj = expressions(ESObj);

**See Also**
DataMatrix | bioma.ExpressionSet | bioma.data.ExptData

**Topics**
“Managing Gene Expression Data in Objects”
exprprofrange

Calculate range of gene expression profiles

Syntax

Range = exprprofrange(Data)
[Range, LogRange] = exprprofrange(Data)

... = exprprofrange(Data, 'ShowHist', ShowHistValue)

Arguments

<table>
<thead>
<tr>
<th>Data</th>
<th>DataMatrix object on page 1-532 or numeric matrix of expression values, where each row corresponds to a gene.</th>
</tr>
</thead>
</table>
| ShowHistValue | Controls the display of a histogram with range data. Default is:  
                      • false — When output values are specified.  
                      • true — When output values are not specified. |

Description

Range = exprprofrange(Data) calculates the range of each expression profile in Data, a DataMatrix object on page 1-532 or numeric matrix of expression values, where each row corresponds to a gene.

[Range, LogRange] = exprprofrange(Data) returns the log range, that is, log(max(prof)) - log(min(prof)), of each expression profile. If you do not specify output arguments, exprprofrange displays a histogram bar plot of the range.

... = exprprofrange(Data, 'ShowHist', ShowHistValue) controls the display of a histogram with range data. Choices for ShowHistValue are true or false.

Examples

Calculate range of gene expression profiles

Load microarray data containing gene expression levels of Saccharomyces cerevisiae (yeast).

load yeastdata

This MAT-file includes three variables, which are added to the MATLAB® workspace:

• yeastvalues - A matrix of gene expression data
• genes - A cell array of GenBank® accession numbers for labeling the rows in yeastvalues
• times - A vector of time values for labeling the columns in yeastvalues
Calculate the range of expression profiles for yeast data as gene expression changes during the metabolic shift from fermentation to respiration. And display a histogram of the data.

```matlab
range = exprprofrange(yeastvalues,'ShowHist',true);
```

See Also

`exprprofvar` | `generangefilter`

Introduced before R2006a
exprprofvar

Calculate variance of gene expression profiles

Syntax

\[ \text{Variance} = \text{exprprofvar}(\text{Data}) \]

\[
\text{exprprofvar}(\ldots, '\text{PropertyName}', \text{PropertyValue},\ldots)
\]

\[
\text{exprprofvar}(\ldots, '\text{ShowHist}', \text{ShowHistValue})
\]

Arguments

<table>
<thead>
<tr>
<th><strong>Data</strong></th>
<th>DataMatrix object on page 1-532 or numeric matrix of expression values, where each row corresponds to a gene.</th>
</tr>
</thead>
</table>
| **ShowHistValue** | Controls the display of a histogram with variance data. Default is:  
• false — When output values are specified.  
• true — When output values are not specified. |

Description

\[ \text{Variance} = \text{exprprofvar}(\text{Data}) \] calculates the variance of each expression profile in \( \text{Data} \), a DataMatrix object on page 1-532 or numeric matrix of expression values, where each row corresponds to a gene. If you do not specify output arguments, this function displays a histogram bar plot of the range.

\[ \text{exprprofvar}(\ldots, '\text{PropertyName}', \text{PropertyValue},\ldots) \] defines optional properties using property name/value pairs.

\[ \text{exprprofvar}(\ldots, '\text{ShowHist}', \text{ShowHistValue}) \] controls the display of a histogram with range data. Choices for \( \text{ShowHistValue} \) are true or false.

Examples

**Calculate variance of gene expression profiles**

Load microarray data containing gene expression levels of Saccharomyces cerevisiae (yeast).

load yeastdata

This MAT-file includes three variables, which are added to the MATLAB® workspace:

• \text{yeastvalues} - A matrix of gene expression data  
• \text{genes} - A cell array of GenBank® accession numbers for labeling the rows in \text{yeastvalues}  
• \text{times} - A vector of time values for labeling the columns in \text{yeastvalues}
Calculate the variance of expression profiles for yeast data as gene expression changes during the metabolic shift from fermentation to respiration. And display a histogram of the data.

```matlab
range = exprprofvar(yeastvalues,'ShowHist',true);
```

![Histogram of profile variances](image)

**See Also**
exprprofrange | generangefilter | genevarfilter

**Introduced before R2006a**
**exprWrite**

**Class:** bioma.ExpressionSet  
**Package:** bioma

Write expression values in ExpressionSet object to text file

**Syntax**

```matlab
exprWrite(ESObj, File)  
exprWrite(..., 'Delimiter', DelimiterValue, ...)  
exprWrite(..., 'Precision', PrecisionValue, ...)  
exprWrite(..., 'Header', HeaderValue, ...)  
exprWrite(..., 'Annotated', AnnotatedValue, ...)  
exprWrite(..., 'Append', AppendValue, ...)
```

**Description**

`exprWrite(ESObj, File)` writes the expression values in the `Expressions` element (DataMatrix object) from an ExpressionSet object to a text file, using the delimiter `\t` to separate columns. `exprWrite` writes the data starting at the first column of the first row in the destination file.

`exprWrite(..., 'PropertyName', PropertyValue, ...)` calls `exprWrite` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each `PropertyName` in single quotation marks. Each `PropertyName` is case insensitive. These property name/property value pairs are as follows:

- `exprWrite(..., 'Delimiter', DelimiterValue, ...)`: specifies a delimiter symbol to use as a column separator. Default is `\t`.
- `exprWrite(..., 'Precision', PrecisionValue, ...)`: specifies the precision for writing the data to the text file. Default is 5.
- `exprWrite(..., 'Header', HeaderValue, ...)`: specifies the first line of the text file. Default is the `Name` property for the DataMatrix object.
- `exprWrite(..., 'Annotated', AnnotatedValue, ...)`: controls the writing of row and column names to the text file. Choices are true (default) or false.
- `exprWrite(..., 'Append', AppendValue, ...)`: controls the appending of the expression values to `File` when it is an existing file. Choices are true or false (default). If false, `exprWrite` overwrites `File`.

**Input Arguments**

**ESObj**

Object of the bioma.ExpressionSet class.

**Default:**
File

Character vector specifying either a file name or a path and file name for saving the expression values. If you specify only a file name, exprWrite saves the file to the MATLAB Current Folder.

Default:

DelimiterValue

Character vector specifying a delimiter symbol to use as a matrix column separator. Typical choices are:

- ' ' (default)
- '\t'
- ','
- ':'
- '|'

Default:

PrecisionValue

Precision for writing the data to the text file, specified by either:

- Positive integer specifying the number of significant digits
- C-style format character vector starting with %, such as '%6.5f'

Default: 5

HeaderValue

Character vector specifying the first line of the text file. Default is the Name property for the DataMatrix object.

Default:

AnnotatedValue

Controls the writing of row and column names to the text file. Choices are true (default) or false.

Default:

AppendValue

Controls the appending of the expression values to File when it is an existing file. Choices are true or false (default). If false, exprWrite overwrites File.

Default:

Examples

Construct an ExpressionSet object, ESObj, as described in the “Examples” on page 1-0 section of the bioma.ExpressionSet class reference page. Write the expression values in the ExpressionSet object to a text file:
% Write expression values to text file
exprWrite(ES0b, 'myexpressiondata.txt')

See Also
DataMatrix | bioma.ExpressionSet | bioma.data.ExptData | dmwrite

Topics
“Managing Gene Expression Data in Objects”
**exptData**

**Class:** bioma.ExpressionSet  
**Package:** bioma

Retrieve or set experiment data in ExpressionSet object

**Syntax**

\[ \text{ExptDataObj} = \text{exptData(ESObj)} \]

\[ \text{NewESObj} = \text{exptData(ESObj, NewExptDataObj)} \]

**Description**

\[ \text{ExptDataObj} = \text{exptData(ESObj)} \] returns the ExptData object stored in an ExpressionSet object.

\[ \text{NewESObj} = \text{exptData(ESObj, NewExptDataObj)} \] replaces the ExptData object in \( \text{ESObj} \), an ExpressionSet object, with \( \text{NewExptDataObj} \), a new ExptData object, and returns \( \text{NewESObj} \), a new ExpressionSet object.

**Input Arguments**

\( \text{ESObj} \)

Object of the bioma.ExpressionSet class.

**Default:**

\( \text{NewExptDataObj} \)

Object of the bioma.data.ExptData class.

**Output Arguments**

\( \text{ExptDataObj} \)

Object of the bioma.data.ExptData class.

\( \text{NewESObj} \)

Object of the bioma.ExpressionSet class, returned after replacing the ExptData object.

**Examples**

Construct an ExpressionSet object, \( \text{ESObj} \), as described in the “Examples” on page 1-0 section of the bioma.ExpressionSet class reference page. Retrieve the ExptData object stored in the ExpressionSet object:
% Retrieve the ExptData object
NewEDObj = exptData(ESObj);

See Also
DataMatrix | bioma.ExpressionSet | bioma.data.ExptData | featureData | sampleData

Topics
“Managing Gene Expression Data in Objects”
**exptInfo**

**Class:** bioma.ExpressionSet  
**Package:** bioma

Retrieve or set experiment information in ExpressionSet object

**Syntax**

```matlab
MIAMEObj = exptInfo(ESObj)
NewESObj = exptInfo(ESObj, NewMIAMEObj)
```

**Description**

`MIAMEObj = exptInfo(ESObj)` returns a MIAME object containing experiment information from an ExpressionSet object.

`NewESObj = exptInfo(ESObj, NewMIAMEObj)` replaces the MIAME object in `ESObj`, an ExpressionSet object, with `NewMIAMEObj`, a new MIAME object, and returns `NewESObj`, a new ExpressionSet object.

**Input Arguments**

**ESObj**

Object of the bioma.ExpressionSet class.

**Default:**

**NewMIAMEObj**

Object of the bioma.data.MIAME class.

**Default:**

**Output Arguments**

**MIAMEObj**

Object of the bioma.data.MIAME class.

**NewESObj**

Object of the bioma.ExpressionSet class, returned after replacing the MIAME object.

**Examples**

Construct an ExpressionSet object, `ESObj`, as described in the “Examples” on page 1-0 section of the bioma.ExpressionSet class reference page. Retrieve the MIAME object stored in the ExpressionSet object:
% Retrieve the MIAME object
NewMIAMEObj = exptInfo(E50bj);

References


See Also
bioma.ExpressionSet | bioma.data.MIAME

Topics
“Managing Gene Expression Data in Objects”
fastainfo

Return information about FASTA file

**Syntax**

```matlab
info = fastainfo(file)
```

**Description**

`info = fastainfo(file)` returns a structure `info` containing summary information about the FASTA file.

**Examples**

**Get Summary Information about FASTA File**

```matlab
info = fastainfo('ex1ref.fasta')
```

```matlab
info = struct with fields:
    Filename: 'ex1ref.fasta'
    FilePath: 'B:\matlab\toolbox\bioinfo\bioinfodata'
    FileModDate: '07-May-2010 16:12:06'
    FileSize: 1612
    NumberOfEntries: 1
    Header: 'Reference for ex1'
    Length: 1569
```

**Input Arguments**

`file` — FASTA file
character vector | string | character array

FASTA file, specified as one of the following:

- Character vector or string specifying a FASTA file name, path and name of a FASTA file, or URL pointing to a FASTA file. If you specify only a file name, the file must be on the MATLAB search path or in the current folder.
- Character array containing the text of a FASTA file.

Data Types: char | string

**Output Arguments**

`info` — Summary information about FASTA file
structure
Summary information about the FASTA file, returned as a structure. The structure contains the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>Name of the file.</td>
</tr>
<tr>
<td>FilePath</td>
<td>Path to the file.</td>
</tr>
<tr>
<td>FileModDate</td>
<td>Modification date of the file.</td>
</tr>
<tr>
<td>FileSize</td>
<td>Size of the file in bytes.</td>
</tr>
<tr>
<td>NumberOfEntries</td>
<td>Number of sequence entries in the file.</td>
</tr>
<tr>
<td>Header</td>
<td>If file contains only one sequence, then this is a character vector containing the header information from the FASTA file. Otherwise, this field is empty.</td>
</tr>
<tr>
<td>Length</td>
<td>If file contains only one sequence, then this is a scalar specifying the length of the sequence. Otherwise, this field is empty.</td>
</tr>
</tbody>
</table>

See Also
- fastaread
- fastawrite
- fastqinfo
- fastqread
- fastqwrite

Topics
- “Data Import”

Introduced in R2009b
fastaread

Read data from FASTA file

Syntax

`FASTAData = fastaread(File)`  
`[Header, Sequence] = fastaread(File)`  

`... = fastaread(File, ...'IgnoreGaps', IgnoreGapsValue, ...)`  
`... = fastaread(File, ...'Blockread', BlockreadValue, ...)`  
`... = fastaread(File, ...'TrimHeaders', TrimHeadersValue, ...)`

Input Arguments

- **File**
  Either of the following:
  - Character vector or string specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a FASTA-formatted file (ASCII text file). If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.
  - MATLAB character array that contains the text of a FASTA-formatted file.

- **IgnoreGapsValue**
  Controls the removal of gap symbols. Choices are `true` or `false` (default).

- **BlockreadValue**
  Scalar or vector that controls the reading of a single sequence entry or block of sequence entries from a FASTA-formatted file containing multiple sequences. Enter a scalar `N` to read the `N`th entry in the file. Enter a 1-by-2 vector `[M1, M2]` to read the block of entries starting at the `M1` entry and ending at the `M2` entry. To read all remaining entries in the file starting at the `M1` entry, enter a positive value for `M1` and enter `Inf` for `M2`.

- **TrimHeadersValue**
  Specifies whether to trim the header after the first white space character. White space characters include a space (char(32)) and a tab (char(9)). Choices are `true` or `false` (default).

Output Arguments

- **FASTAData**
  MATLAB structure with the fields `Header` and `Sequence`.

Description

`fastaread` reads data from a FASTA-formatted file into a MATLAB structure with the following fields.
A FASTA-formatted file begins with a right angle bracket (>) and a single line description. Following this description is the sequence as a series of lines with fewer than 80 characters. Sequences must use the standard IUB/IUPAC amino acid and nucleotide letter codes.

For a list of codes, see aminolookup and baselookup.

\[
\text{FASTAData} = \text{fastaread}('\text{File}') \text{ reads a FASTA-formatted file and returns the data in a structure. FASTAData.Header is the header information, while FASTAData.Sequence is the sequence stored as a character vector or string.}
\]

\[
[\text{Header}, \text{Sequence}] = \text{fastaread}('\text{File}') \text{ reads data from a file into separate variables. If the file contains multiple sequences, then Header and Sequence are cell arrays of header and sequence information.}
\]

\[
... = \text{fastaread}('\text{File}, ...'\text{PropertyName}', \text{PropertyValue}, ...') \text{ calls fastaread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. The property name/value pairs can be in any format supported by the function set (for example, name-value pairs and structures). These property name/property value pairs are as follows:}
\]

\[
... = \text{fastaread}('\text{File}, ...'\text{IgnoreGaps}', \text{IgnoreGapsValue}, ...'), \text{when IgnoreGapsValue is true, removes any gap symbol ('-' or '.') from the sequences. Default is false.}
\]

\[
... = \text{fastaread}('\text{File}, ...'\text{Blockread}', \text{BlockreadValue}, ...') \text{ lets you read in a single sequence entry or block of sequence entries from a file containing multiple sequences. If BlockreadValue is a scalar N, then fastaread reads the Nth entry in the file. If BlockreadValue is a 1-by-2 vector [M1, M2], then fastaread reads the block of entries starting at the M1 entry and ending at the M2 entry. To read all remaining entries in the file starting at the M1 entry, enter a positive value for M1 and enter Inf for M2.}
\]

\[
... = \text{fastaread}('\text{File}, ...'\text{TrimHeaders}', \text{TrimHeadersValue}, ...') \text{ specifies whether to trim the header to the first white space.}
\]

## Examples

### Read sequence data from FASTA files

Read the nucleotide sequence information of the human p53 tumor gene.

\[
p53nt = \text{fastaread('p53nt.txt')}
\]

\[
p53nt = \text{struct with fields:}
\]

\[
\text{Header: 'gi|8400737|ref|NM_000546.2| Homo sapiens tumor protein p53 (Li-Fraumeni syndrome)'}
\]

\[
\text{Sequence: 'ACTTGTCATGGCGACTGTCCAGCTTTGTGCCAGGAGGCTGCGACAGGGGTATGGGATGGGGTTTTCCCTCCCTCCTCATGTGCTC'}
\]

Read the amino acid sequence information of p53 protein.

\[
\text{p53} = \text{fastaread('p53nt.txt', }\text{PropertyName}', \text{PropertyValue}, ...')
\]

\[
\text{p53 = struct with fields:}
\]

\[
\text{Header: 'gi|8400737|ref|NM_000546.2| Homo sapiens tumor protein p53 (Li-Fraumeni syndrome)'}
\]

\[
\text{Sequence: 'FQQNYAVYNQFQNYAVYNQFQNYAVYNQ'}
\]
p53aa = fastaread('p53aa.txt')

p53aa = struct with fields:
    Header: 'gi|8400738|ref|NP_000537.2| tumor protein p53 [Homo sapiens]'
    Sequence: 'MEEPQSDPSVEPPLSQETFSDLWKLLPENNVLSPQLPSQAMDDLMLSPDDIEQWFTEDPGPDEAPMPEAAPVAPAPAA...

Read a block of entries from a FASTA file.

pf2_5_10 = fastaread('pf00002.fa', 'blockread', [5 10], ...
                  'ignoregaps',true)

pf2_5_10=6×1 struct array with fields:
    Header
    Sequence

See Also
BioIndexedFile | aminolookup | baselookup | emblread | fastainfo | fastawrite | fastqinfo | fastqread | fastqwrite | genbankread | genpeptread | multialignread | saminfo | samread | seqprofile | seqviewer | sffinfo | sffread

Introduced before R2006a
fastawrite

Write to file using FASTA format

Syntax

fastawrite(File, Data)
fastawrite(File, Header, Sequence)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Character vector or string specifying either a file name or a path and file name for saving the FASTA-formatted data. If you specify only a file name, fastawrite saves the file to the MATLAB Current Folder. If you specify an existing file, fastawrite appends the data to the file, instead of overwriting the file.</td>
</tr>
</tbody>
</table>
| Data     | Any of the following:  
|          | • Character vector or string containing a sequence  
|          | • MATLAB structure containing the fields Header and Sequence  
|          | • MATLAB structure containing sequence information from the GenBank or GenPept database, such as returned by genbankread, getgenbank, genpeptread, or getgenpept.  
|          | • Character array, where each row is a sequence. |
| Header   | Character vector or string containing header information about the sequence. This text appears in the header of the FASTA-formatted file, File. |
| Sequence | Character vector or string containing an amino acid or nucleotide sequence using the standard IUB/IUPAC letter or integer codes. For a list of valid characters, see Amino Acid Lookup or Nucleotide Lookup. |

Description

fastawrite(File, Data) writes the contents of Data to File, a FASTA-formatted file. If you specify an existing FASTA-formatted file, fastawrite appends the data to the file, instead of overwriting the file. For the FASTA-format specifications, visit https://www.ncbi.nlm.nih.gov/BLAST/fasta.shtml.

fastawrite(File, Header, Sequence) writes the specified header and sequence information to File, a FASTA-formatted file.

Tip To append FASTA-formatted data to an existing file, simply specify that file name. fastawrite adds the data to the end of the file.

If you are using fastawrite in a script, you can disable the append warning message by entering the following command lines before the fastawrite command:

```matlab
warnState = warning; % Save the current warning state
warning('off','Bioinfo:fastawrite:AppendToFile');
```
Then enter the following command line after the `fastawrite` command:

```matlab
warning(warnState) %Reset warning state to previous settings
```

---

**Examples**

**Example 1.13. Writing a Coding Region to a FASTA-Formatted File**

1. Retrieve the sequence for the human p53 gene from the GenBank database.
   ```matlab
   seq = getgenbank('NM_000546');
   ```

2. Read the coordinates of the coding region in the CDS line.
   ```matlab
   start = seq.CDS.indices(1)
   start =
   198
   stop = seq.CDS.indices(2)
   stop =
   1379
   ```

3. Extract the coding region.
   ```matlab
   codingSeq = seq.Sequence(start:stop);
   ```

4. Write the coding region to a FASTA-formatted file, specifying Coding region for p53 for the Header in the file, and p53coding.txt for the file name.
   ```matlab
   fastawrite('p53coding.txt','Coding region for p53',codingSeq);
   ```

**Example 1.14. Saving Multiple Sequences to a FASTA-Formatted File**

1. Write two nucleotide sequences to a MATLAB structure containing the fields `Header` and `Sequence`.
   ```matlab
   data(1).Sequence = 'ACACAGGAAA';
   data(1).Header = 'First sequence';
   data(2).Sequence = 'ACGTCAGGTC';
   data(2).Header = 'Second sequence';
   ```

2. Write the sequences to a FASTA-formatted file, specifying `my_sequences.txt` for the file name.
   ```matlab
   fastawrite('my_sequences.txt', data)
   ```

3. Display the FASTA-formatted file, `my_sequences.txt`.
   ```matlab
   type('my_sequences.txt')
   ```
   ```plaintext
   >First sequence
   ACACAGGAAA
   ```
   ```plaintext
   >Second sequence
   ACGTCAGGTC
   ```
Example 1.15. Appending Sequences to a FASTA-Formatted File

1. If you haven't already done so, create the FASTA-formatted file, my_sequences.txt, described previously.

2. Append a third sequence to the file.
   fastawrite('my_sequences.txt','Third sequence','TACTGACTTC')

3. Display the FASTA-formatted file, my_sequences.txt.
   type('my_sequences.txt')

   >First sequence
   ACACAGGAAA

   >Second sequence
   ACGTCAGGTC

   >Third sequence
   TACTGACTTC

See Also
fastainfo | fastaread | fastqinfo | fastqread | fastqwrite | genbankread | genpeptread | getgenbank | getgenpept | multialignwrite | saminfo | samread | seqviewer | sffinfo | sffread

Introduced before R2006a
fastqinfo

Return information about FASTQ file

Syntax

InfoStruct = fastqinfo(File)

Description

InfoStruct = fastqinfo(File) returns a MATLAB structure containing summary information about a FASTQ-formatted file.

Input Arguments

File

Character vector or string specifying a file name or path and file name of a FASTQ-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.

Default:

Output Arguments

InfoStruct

MATLAB structure containing summary information about a FASTQ-formatted file. The structure contains the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>Name of the file.</td>
</tr>
<tr>
<td>FilePath</td>
<td>Path to the file.</td>
</tr>
<tr>
<td>FileModeDate</td>
<td>Modification date of the file.</td>
</tr>
<tr>
<td>FileSize</td>
<td>Size of the file in bytes.</td>
</tr>
<tr>
<td>NumberOfEntries</td>
<td>Number of sequence reads in the file.</td>
</tr>
</tbody>
</table>

Examples

Return a summary of the contents of a FASTQ file:

info = fastqinfo('SRR005164_1_50.fastq')

info =

    Filename: 'SRR005164_1_50.fastq'
    FilePath: 'D:\2010_08_24_h11m43s32_job6027_pass\matlab\toolbox\bioinfo\biodemos'
    FileModeDate: '03-Mar-2009 14:21:51'
See Also
BioIndexedFile | BioRead | fastainfo | fastaread | fastawrite | fastqinfo | fastqread | fastqwrite | saminfo | samread | sffinfo | sffread

Topics
“Working with Illumina®/Solexa Next-Generation Sequencing Data”

External Websites
https://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?cmd=show&f=main&m=main&s=main

Introduced in R2009b
fastqread

Read data from FASTQ file

Syntax

FASTQStruct = fastqread(File)
[Header, Sequence] = fastqread(File)
[Header, Sequence, Qual] = fastqread(File)

fastqread(..., 'Blockread', BlockreadValue, ...)
fastqread(..., 'HeaderOnly', HeaderOnlyValue, ...)
fastqread(..., 'TrimHeaders', TrimHeadersValue, ...)

Description

FASTQStruct = fastqread(File) reads a FASTQ-formatted file and returns the data in a MATLAB array of structures.

[Header, Sequence] = fastqread(File) returns only the header and sequence data in two separate variables.

[Header, Sequence, Qual] = fastqread(File) returns the data in three separate variables.

fastqread(..., 'PropertyName', PropertyValue, ...) calls fastqread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each PropertyName in single quotation marks. Each PropertyName is case insensitive. These property name/property value pairs are as follows:

fastqread(..., 'Blockread', BlockreadValue, ...) reads a single sequence entry or block of sequence entries from a FASTQ-formatted file containing multiple sequences.

fastqread(..., 'HeaderOnly', HeaderOnlyValue, ...) specifies whether to return only the header information.

fastqread(..., 'TrimHeaders', TrimHeadersValue, ...) specifies whether to trim the header to the first white space.

Input Arguments

File

Either of the following:

- Character vector or string specifying a file name or path and file name of a FASTQ-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.
- MATLAB character array that contains the text of a FASTQ-formatted file.

Default:
**BlockreadValue**

Scalar or vector that controls the reading of a single sequence entry or block of sequence entries from a FASTQ-formatted file containing multiple sequences. Enter a scalar \( N \) to read the \( N \)th entry in the file. Enter a 1-by-2 vector \([M1, M2]\) to read a block of entries starting at the \( M1 \) entry and ending at the \( M2 \) entry. To read all remaining entries in the file starting at the \( M1 \) entry, enter a positive value for \( M1 \) and enter \( \text{Inf} \) for \( M2 \).

**Default:**

**HeaderOnlyValue**

Specifies whether to return only the header information. Choices are true or false (default).

**Default:**

**TrimHeadersValue**

Specifies whether to trim the header after the first white space character. White space characters include a space (char(32)) and a tab (char(9)). Choices are true or false (default).

**Default:**

**Output Arguments**

**FASTQStruct**

Array of structures containing information from a FASTQ-formatted file. There is one structure for each sequence read or entry in the file. Each structure contains the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Header information.</td>
</tr>
<tr>
<td>Sequence</td>
<td>Single letter-code representation of a nucleotide sequence.</td>
</tr>
<tr>
<td>Quality</td>
<td>ASCII representation of per-base quality scores for a nucleotide sequence.</td>
</tr>
</tbody>
</table>

**Header**

Variable containing header information or, if the FASTQ-formatted file contains multiple sequences, a cell array containing header information.

**Sequence**

Variable containing sequence information or, if the FASTQ-formatted file contains multiple sequences, a cell array containing sequence information.

**Qual**

Variable containing quality information or, if the FASTQ-formatted file contains multiple sequences, a cell array containing quality information.

**Examples**

Read a FASTQ file into an array of structures:
% Read the contents of a FASTQ-formatted file into
% an array of structures
reads = fastqread('SRR005164_1_50.fastq')

reads =

1x50 struct array with fields:
    Header
    Sequence
    Quality

Read a FASTQ file into three separate variables:

% Read the contents of a FASTQ-formatted file into
% three separate variables
[headers,seqs,quals] = fastqread('SRR005164_1_50.fastq');

Read a block of entries from a FASTQ file:

% Read the contents of reads 5 through 10 into
% an array of structures
reads_5_10 = fastqread('SRR005164_1_50.fastq', 'blockread', [5 10])

1x6 struct array with fields:
    Header
    Sequence
    Quality

More About

FASTQ-file Format

A FASTQ-formatted file contains nucleotide sequence and quality information on four lines:

• **Line 1** — Header information prefixed with an @ symbol
• **Line 2** — Nucleotide sequence
• **Line 3** — Header information prefixed with a + symbol
• **Line 4** — ASCII representation of per-base quality scores for the nucleotide sequence using Phred or Solexa encoding

See Also

BioIndexedFile | BioRead | fastainfo | fastaread | fastawrite | fastqinfo | fastqread | fastqwrite | saminfo | samread | sffinfo | sffread

Topics

“Working with Illumina®/Solexa Next-Generation Sequencing Data”

External Websites

https://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?cmd=show&f=main&m=main&s=main

Introduced in R2009b
fastqwrite

Write to file using FASTQ format

Syntax

fastqwrite(File, FASTQStruct)
fastqwrite(File, Header, Sequence, Qual)

Description

fastqwrite(File, FASTQStruct) writes the contents of a MATLAB structure or array of structures to a FASTQ-formatted file. If you specify an existing FASTQ-formatted file, fastqwrite appends the data to the file, instead of overwriting the file.

fastqwrite(File, Header, Sequence, Qual) writes header, sequence, and quality information to a FASTQ-formatted file.

Tip

To append FASTQ-formatted data to an existing file, simply specify that file name. fastqwrite adds the data to the end of the file.

If you are using fastqwrite in a script, you can disable the append warning message by entering the following command lines before the fastqwrite command:

warnState = warning %Save the current warning state
warning('off','Bioinfo:fastqwrite:AppendToFile');

Then enter the following command line after the fastqwrite command:

warning(warnState) %Reset warning state to previous settings

Input Arguments

File

Character vector or string specifying either a file name or a path and file name for saving the FASTQ-formatted data. If you specify only a file name, fastqwrite saves the file to the MATLAB Current Folder. If you specify an existing file, fastqwrite appends the data to the file, instead of overwriting the file.

Default:

FASTQStruct

MATLAB structure or array of structures containing the fields Header, Sequence, and Quality, such as returned by fastqread.

Default:
Header

Character vector or string containing header information about the nucleotide sequence. This text appears in the header of the FASTQ-formatted file, *File*.

Default:

Sequence

Character vector or string containing a nucleotide sequence using the standard IUB/IUPAC letter or integer codes. For a list of valid characters, see Amino Acid Lookup or Nucleotide Lookup.

Default:

Qual

Character vector or string containing ASCII representation of per-base quality scores for a nucleotide sequence.

Default:

Examples

Write multiple sequences to a FASTQ file from an array of structures:

```matlab
% Read the contents of a FASTQ-formatted file into
% an array of structures
reads = fastqread('SRR005164_1_50.fastq');
% Create another array of structures for the first five reads
reads5 = reads(1:5);
% Write the first five reads to a separate FASTQ-formatted file
fastqwrite('fiveReads.fastq', reads5)
```

Write a single sequence to a FASTQ file from separate variables:

```matlab
% Create separate variables for the header, sequence, and
% quality information of a nucleotide sequence
h = 'MYSEQ-000_1_1_1_953_493';
s = 'GTTACCATGATGTTATTTCTTCATTTGGAGGTAAAA';
q = '))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))RJRZTQLOA';
% Write the information to a FASTQ-formatted file
fastqwrite('oneRead.fastq', h, s, q)
```

More About

FASTQ-file Format

A FASTQ-formatted file contains nucleotide sequence and quality information on four lines:

- **Line 1** — Header information prefixed with an @ symbol
- **Line 2** — Nucleotide sequence
- **Line 3** — Header information prefixed with a + symbol
- **Line 4** — ASCII representation of per-base quality scores for the nucleotide sequence using Phred or Solexa encoding
See Also
BioIndexedFile | BioRead | fastainfo | fastaread | fastawrite | fastqinfo | fastqread |
| fastqwrite | saminfo | samread | sffinfo | sffread

Topics
“Working with Illumina®/Solexa Next-Generation Sequencing Data”

External Websites
https://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?cmd=show&f=main&m=main&s=main

Introduced in R2009b
featurecount

Compute the number of reads mapped to genomic features

Syntax

T = featurecount(GTFfile,Inputfile)
[T,S] = featurecount(__)
[___] = featurecount(__,Name,Value)

Description

T = featurecount(GTFfile,Inputfile) counts the number of reads in the BAM-formatted or SAM-formatted file Inputfile that map onto genomic features as specified in the GTF-formatted file GTFfile. GTFfile specifies the annotation file. Inputfile specifies the names of the BAM or SAM files to consider. The output T is a table where rows correspond to features and columns correspond to the input files. The elements of the table consist of the number of reads mapping to each feature for a given input file.

[T,S] = featurecount(__) returns a table S with a summary of assigned and unassigned alignment entries. If multiple input files are provided, each file is associated with a column.

[___] = featurecount(__,Name,Value) uses additional options specified by one or more Name,Value pair arguments.

Examples

Count the number of reads mapped to genomic features

Count reads from a sample SAM file that map to the features included in a GTF file. By default, featurecount maps the reads to exons, and summarizes the total number of reads at the gene level.

[t,s] = featurecount('Dmel_BDGP5_nohc.gtf','rnaseq_sample1.sam');

Processing GTF file Dmel_BDGP5_nohc.gtf ...
Processing SAM file rnaseq_sample1.sam ...
Processing reference chr2L ...
Processing reference chr2R ...
Processing reference chr3L ...
Processing reference chr3R ...
Processing reference chr4 ...
Processing reference chrX ...
Done.

Display the first 10 rows of count data.

t(1:10,:)
The ID column contains the names of features (genes in this example). The Reference column lists the names of reference sequences for the features. The third column contains the total number of reads mapped to each feature for a given SAM file, that is, rnaseq_sample1.sam. By default, the table shows only those features (rows) and SAM files (columns) with non-zero read counts. Set 'ShowZeroCounts' to true to include those rows and columns with all zero counts in the output table.

s contains the summary statistics of assigned and unassigned reads from each SAM file. For instance, the TotalEntries row indicates the total number of alignment records from the given SAM file, and the Assigned row includes the number of reads that are assigned to features in the GTF file. For details about each row, refer to the Output Arguments section of the reference page.

s

s=9×1 table

<table>
<thead>
<tr>
<th></th>
<th>rnaseq_sample1</th>
</tr>
</thead>
<tbody>
<tr>
<td>TotalEntries</td>
<td>33354</td>
</tr>
<tr>
<td>Assigned</td>
<td>16399</td>
</tr>
<tr>
<td>Unassigned_ambiguous</td>
<td>167</td>
</tr>
<tr>
<td>Unassigned_filtered</td>
<td>0</td>
</tr>
<tr>
<td>Unassigned_lowMappingQuality</td>
<td>0</td>
</tr>
<tr>
<td>Unassigned_multiMapped</td>
<td>0</td>
</tr>
<tr>
<td>Unassigned_noFeature</td>
<td>16788</td>
</tr>
<tr>
<td>Unassigned_supplementary</td>
<td>0</td>
</tr>
<tr>
<td>Unassigned_unmapped</td>
<td>0</td>
</tr>
</tbody>
</table>

Count reads without any summarization and disable displaying the progress messages.

[t2,s2] = featurecount('Dmel_BDGP5_nohc.gtf','rnaseq_sample1.sam', ...
                      'Summarization',false,'Verbose',false);

Notice the ID column of the output table now reports the feature attribute followed by the start and stop positions of each feature, separated by underscores.

t2(1:10,:)
You can choose how to assign a read to a particular feature when the read overlaps with multiple features by setting the 'OverlapMethod' option. For instance, if you want to count a read only if it fully overlaps a feature, use the 'full' option.

```matlab
[tFull, sFull] = featurecount('Dmel_BDGP5_nohc.gtf','rnaseq_sample1.sam', ... 'OverlapMethod','full','Verbose',false);
```

If you have paired-end data, you can count reads as fragments.

```matlab
[tFrag,sFrag] = featurecount('Dmel_BDGP5_nohc.gtf','rnaseq_sample1.sam', ... 'CountFragments',true,'Verbose',false);
```

You can also count fragments from multiple SAM files.

```matlab
[t2,s2] = featurecount('Dmel_BDGP5_nohc.gtf',... {'rnaseq_sample1.sam','rnaseq_sample2.sam'},'CountFragments',true, ... 'Verbose',false);
```

Use the following options to count paired-end reads where at least one of the read mates are above a certain mapping quality threshold.

```matlab
[t3,s3] = featurecount('Dmel_BDGP5_nohc.gtf',... 'rnaseq_sample1.sam','CountFragments',true, 'MinMappingQuality',20, ... 'Verbose',false);
```

If the reads come from any strand-specific assay, you can specify such strand specificity during counting. For instance, if the protocol is stranded, the strand of the feature is compared with the strand of the read. Then only those reads that have the same strand as the overlapped feature are counted.

```matlab
[t4,s4] = featurecount('Dmel_BDGP5_nohc.gtf',... 'rnaseq_sample1.sam','StrandSpecificity','stranded','Verbose',false);
```

### Input Arguments

- **GTFfile** — GTF-formatted file name
  - character vector | string
  - GTF-formatted file name, specified as a character vector or string.
  - Example: `'Dmel_BDGP5_nohc.gtf'`

- **Inputfile** — BAM-formatted or SAM-formatted file name
  - character vector | string | string vector | cell array of character vectors
  - BAM-formatted or SAM-formatted file name, specified as a character vector, string, string vector, or cell array of character vectors.
Example: 'rnaseq_sample1.sam'

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1, Value1,...,NameN, ValueN.

Example: 'CountFragments', true specifies to count reads as pairs of mates.

**Feature — Feature type**

'exon' (default) | character vector | string

Feature type, specified as a character vector or string. This is used to decide what feature to consider from the GTF file. Default is 'exon'.

**Metafeature — Attribute type**

'gene_id' (default) | character vector | string

Attribute type, specified as a character vector or string. This is used to decide what attribute to consider from the GTF file for grouping features into metafeatures and summarizing the read count.

**Summarization — Boolean variable indicating whether to summarize at the metafeature level**

true (default) | false

Boolean variable indicating whether to summarize at the metafeature level, specified as true or false.

Default is true, meaning the function groups features into metafeatures and reports the read counts for metafeatures.

**Alias — Name of file containing aliases of reference names**

character vector | string

Name of file containing aliases of reference names, specified as a character vector or string. The file must be a tab-delimited file where the first column corresponds to the reference names used in the GTF file, and the second column corresponds to the reference names used in the input file(s). The names are case-sensitive. It is necessary to include only the reference names that are different in the GTF file and the input file. The file must contain only one alias term for any reference listed in the input file. By default, the reference names in the GTF file and those in the input files are assumed to be the same.

**CountFragments — Boolean variable indicating whether to count reads as pairs of mates**

false (default) | true

Boolean variable indicating whether to count reads as fragments, specified as true or false. Paired-end reads must have the same ID for the field QNAME in the input file, and the mutual order of mates is inferred by the appropriate bit in the FLAG field within the input file. Reads that have no valid mate either because the mate is unmapped or filtered out by input criteria are still counted if they satisfy the overlapping criteria.

Default is false, that is, the reads are counted as single-end reads, and their pairing information is ignored.
**StrandSpecificity** — Strand specificity of sequencing protocol
'unstranded' (default) | 'stranded' | 'reverse'

Strand specificity of the sequencing protocol, specified as 'unstranded' (default), 'stranded', or 'reverse'.

- If 'unstranded', the strand of the reads (or fragments) is ignored.
- If 'stranded', the strand of the reads (or fragments) is considered, and only those having the same strand as the feature they overlap are counted.
- If 'reverse', the opposite direction of the strand of the reads (or fragments) is considered, and only those having the opposite strand as the feature they overlap are counted.

When counting fragments (paired-end reads), the strand of the first mate is considered as the strand of the whole fragment. The mutual order of mates (first or second) is inferred from the appropriate bit in the FLAG field of the input file.

**MinOverlap** — Minimum number of overlapped bases required
1 (default) | positive integer

Minimum number of overlapped bases required to assign a read to a feature, specified as a positive integer. When counting fragments, the sum of the overlaps from each end is used as the minimum number of overlapped bases.

**MinMappingQuality** — Minimum mapping quality for a given read
0 (default) | non-negative integer

Minimum mapping quality for a given read to be considered for counting, specified as a non-negative integer. This corresponds to the MAPQ field in the input file. If counting fragments, at least one of the read mates must satisfy this criterion in order to be considered for counting.

**CountMultiOverlap** — Boolean variable indicating whether to count reads overlapping multiple features
false (default) | true

Boolean variable indicating whether to count reads overlapping multiple features, specified as true or false (default).

If true, a read (or fragment) overlapping multiple features is counted multiple times. During summarization at the metafeature level, a read (or fragment) is counted only once if it overlaps with multiple features belonging to the same metafeature as long as it does not overlap with other metafeatures.

**CountMultiMapped** — Counting option for reads having multiple mapping locations in the input file
'primary' (default) | 'none' | 'all'

Counting option for reads having multiple mapping locations in the input file, specified as 'primary' (default), 'none', or 'all'.

- If 'primary', only the primary alignment of a multi-mapped read is considered. The appropriate bit in the input file is used to identify primary alignments.
- If 'none', all alignments of a multi-mapped read are ignored. The NH tag is used to identify multi-mapped reads.
• If 'all', all alignments of a multi-mapped read are considered and counted multiple times.

**BothEndsMapped — Boolean variable indicating whether a fragment must have both mates mapped**

false (default) | true

Boolean variable indicating whether a fragment must have both mates mapped, specified as true or false. Mate mapping information is retrieved from the FLAG field in the input file. Default is false.

**ProperlyPaired — Boolean variable indicating whether a fragment must be properly paired**

false (default) | true

Boolean variable indicating whether a fragment must be properly paired, specified as true or false. Mate pairing information is retrieved from the FLAG field in the input file. Default is false.

**ShowZeroCounts — Boolean variable indicating whether to report features or metafeatures with zero count**

false (default) | true

Boolean variable indicating whether to report features or metafeatures with zero count for every input file in the output table, specified as true or false. Default is false, that is, only rows with non-zero counts and columns with non-zero counts are included in the output table.

**OverlapMethod — Method to use when assigning a given read to metafeature**

'partial' (default) | 'full' | 'max' | 'hits'

Method to use when assigning a given read to metafeature, specified as 'partial', 'full', 'max', or 'hits'. If 'Summarization' is set to false, then the reads are assigned to features, instead of metafeatures, based on the specified method.

In the following table, \( R \) refers to a read or fragment, and \( M \) refers to a metafeature.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'partial'</td>
<td>( R ) is assigned to ( M ) if ( R ) overlaps (even partially) only with ( M ). Otherwise ( R ) is considered ambiguous.</td>
</tr>
<tr>
<td>'full'</td>
<td>( R ) is assigned to ( M ) if ( R ) is completely mapped only within ( M ), that is, fully overlapping only ( M ). Otherwise ( R ) is considered ambiguous.</td>
</tr>
<tr>
<td>'max'</td>
<td>( R ) is assigned to ( M ) if ( R ) satisfies the overlapping criteria only with ( M ), or if ( R ) satisfies the overlapping criteria with several metafeatures but overlaps fully only with ( M ).</td>
</tr>
<tr>
<td>'hits'</td>
<td>( R ) is assigned to ( M ) if ( R ) overlaps even partially only ( M ), or if ( M ) is the only metafeature with the highest number of features hit by ( R ); otherwise ( R ) is considered ambiguous.</td>
</tr>
</tbody>
</table>

The following schematic diagram and table illustrate the outcome of these methods in conjunction with the 'CountMultiOverlap' name-value pair argument. In the figure, the read refers to a short-read sequence from an input file, and feature A and feature B refers to features listed in a GTF file.
Each method column lists the feature that the read is assigned to based on the corresponding method. The 'CountMultiOverlap' column indicates whether this name-value pair is set to true or false and if it has any effect in the outcome of each method.

<table>
<thead>
<tr>
<th>Case</th>
<th>'CountMultiOverlap'</th>
<th>'partial'</th>
<th>'full'</th>
<th>'max'</th>
<th>'hits'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No effect since the read maps only to one feature (feature A).</td>
<td>feature A</td>
<td>feature A</td>
<td>feature A</td>
<td>feature A</td>
</tr>
<tr>
<td>2</td>
<td>No effect since the read maps only to one feature (feature A).</td>
<td>feature A</td>
<td>no feature</td>
<td>feature A</td>
<td>feature A</td>
</tr>
<tr>
<td>3</td>
<td>No effect since the read maps only to one feature (feature A).</td>
<td>feature A</td>
<td>no feature</td>
<td>feature A</td>
<td>feature A</td>
</tr>
<tr>
<td>4</td>
<td>No effect since the read maps only to one feature (feature A).</td>
<td>feature A</td>
<td>feature A</td>
<td>feature A</td>
<td>feature A</td>
</tr>
<tr>
<td>5</td>
<td>false</td>
<td>ambiguous</td>
<td>feature A</td>
<td>feature A</td>
<td>ambiguous</td>
</tr>
<tr>
<td></td>
<td>true</td>
<td>feature A, feature B</td>
<td>feature A</td>
<td>feature A</td>
<td>feature A, feature B</td>
</tr>
<tr>
<td>6</td>
<td>false</td>
<td>ambiguous</td>
<td>ambiguous</td>
<td>ambiguous</td>
<td>ambiguous</td>
</tr>
<tr>
<td>7</td>
<td>false</td>
<td>Ambiguous</td>
<td>feature A</td>
<td>feature A</td>
<td>feature A</td>
</tr>
<tr>
<td></td>
<td>true</td>
<td>feature A, feature B</td>
<td>feature A</td>
<td>feature A</td>
<td>feature A</td>
</tr>
</tbody>
</table>

*no feature* means that the read is not assigned to any feature. If you have specified the second output table S, its Unassigned_noFeature row is incremented by one for such occurrence. *ambiguous*
means that the read is not assigned to any feature since it satisfies the overlapping criteria for multiple features, and the Unassigned_ambiguous row is incremented by one for such occurrence.

**UseParallel** — Boolean variable indicating whether to compute in parallel
false (default) | true

Boolean variable indicating whether to compute in parallel, specified as true or false.

In order to execute the computation in parallel, you must have Parallel Computing Toolbox™. If a MATLAB parallel pool does not exist, one is automatically created when the auto-creation option is enabled in your parallel preferences. Otherwise, computation runs in the serial mode.

Default is false, that is, serial mode.

**Verbose** — Boolean variable indicating whether to display the progress of computation
true (default) | false

Boolean variable indicating whether to display the progress of computation, specified as true or false.

**Output Arguments**

**T** — Results containing sequence reads mapped to genomic features

<table>
<thead>
<tr>
<th>Row Names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TotalEntries</td>
<td>Number of records (or alignments) in the input file</td>
</tr>
<tr>
<td>Assigned</td>
<td>Number of reads or fragments that were assigned to features</td>
</tr>
<tr>
<td>Unassigned_ambiguous</td>
<td>Number of unassigned reads or fragments overlapping multiple features or metafeatures</td>
</tr>
<tr>
<td>Unassigned_filtered</td>
<td>Number of alignment records filtered by input criteria</td>
</tr>
<tr>
<td>Unassigned_lowMappingQuality</td>
<td>Number of alignment records filtered out due to low mapping quality</td>
</tr>
</tbody>
</table>

Results containing sequence reads mapped to genomic features, returned as a table. The rows correspond to features, and columns correspond to the input files. The elements of the table consist of the number of reads mapped to each feature for a given input file. The table also reports the ID of each feature and the reference sequence for the feature.

When 'Summarization' is set to false, the ID column of the table reports the metafeature attribute followed by the start and stop positions of each feature, separated by underscores.

**S** — Summary of assigned and unassigned alignment entries

<table>
<thead>
<tr>
<th>Row Names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TotalEntries</td>
<td>Number of records (or alignments) in the input file</td>
</tr>
<tr>
<td>Assigned</td>
<td>Number of reads or fragments that were assigned to features</td>
</tr>
<tr>
<td>Unassigned_ambiguous</td>
<td>Number of unassigned reads or fragments overlapping multiple features or metafeatures</td>
</tr>
<tr>
<td>Unassigned_filtered</td>
<td>Number of alignment records filtered by input criteria</td>
</tr>
<tr>
<td>Unassigned_lowMappingQuality</td>
<td>Number of alignment records filtered out due to low mapping quality</td>
</tr>
</tbody>
</table>

Summary of assigned and unassigned alignment entries, returned as a table. Each column of the table corresponds to each input file provided. The table has the following rows:
<table>
<thead>
<tr>
<th>Row Names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unassigned_multiMapped</td>
<td>Number of alignment records not assigned because of corresponding reads mapped to multiple locations</td>
</tr>
<tr>
<td>Unassigned_noFeature</td>
<td>Number of reads or fragments not assigned to any features</td>
</tr>
<tr>
<td>Unassigned_supplementary</td>
<td>Number of alignment records not assigned because they are flagged as supplementary records for chimeric alignments</td>
</tr>
<tr>
<td>Unassigned_unmapped</td>
<td>Number of alignment records not assigned because corresponding reads are unmapped</td>
</tr>
</tbody>
</table>

**Extended Capabilities**

**Automatic Parallel Support**
Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true`.

For more information, see the `'UseParallel'` name-value pair argument.

**See Also**
bowtie2 | cufflinks

**Topics**
“Count Features from NGS Reads”

**Introduced in R2016a**
**featureData**

**Class:** bioma.ExpressionSet  
**Package:** bioma

Retrieve or set feature metadata in ExpressionSet object

**Syntax**

```
MetaDataObj = featureData(ESObj)
NewESObj = featureData(ESObj, NewMetaDataObj)
```

**Description**

`MetaDataObj = featureData(ESObj)` returns a MetaData object containing the feature metadata from an ExpressionSet object.

`NewESObj = featureData(ESObj, NewMetaDataObj)` replaces the feature metadata in `ESObj`, an ExpressionSet object, with `NewMetaDataObj`, and returns `NewESObj`, a new ExpressionSet object.

**Input Arguments**

**ESObj**

Object of the bioma.ExpressionSet class.

**Default:**

**NewMetaDataObj**

Object of the bioma.data.MetaData class, containing feature metadata, stored in two dataset arrays. The feature names and variable names in `NewMetaDataObj` must match the feature names and variable names in the `MetaDataObj` being replaced in the ExpressionSet object, `ESObj`.

**Default:**

**Output Arguments**

**MetaDataObj**

Object of the bioma.data.MetaData class, containing the feature metadata, stored in two dataset arrays.

**NewESObj**

Object of the bioma.ExpressionSet class, returned after replacing the MetaData object containing the feature metadata.
See Also
bioma.ExpressionSet|bioma.data.MetaData|featureNames|sampleData

Topics
“Managing Gene Expression Data in Objects”
**featureNames**

**Class:** bioma.ExpressionSet  
**Package:** bioma

Retrieve or set feature names in ExpressionSet object

**Syntax**

\[
\text{FeatNames} = \text{featureNames}(\text{ESObj})\\
\text{FeatNames} = \text{featureNames}(\text{ESObj}, \text{Subset})\\
\text{NewESObj} = \text{featureNames}(\text{ESObj}, \text{Subset}, \text{NewFeatNames})
\]

**Description**

\[
\text{FeatNames} = \text{featureNames}(\text{ESObj})\]
returns a cell array of character vectors specifying all feature names in an ExpressionSet object.

\[
\text{FeatNames} = \text{featureNames}(\text{ESObj}, \text{Subset})\]
returns a cell array of character vectors specifying a subset the feature names in an ExpressionSet object.

\[
\text{NewESObj} = \text{featureNames}(\text{ESObj}, \text{Subset}, \text{NewFeatNames})
\]
replaces the feature names specified by \text{Subset} in \text{ESObj}, an ExpressionSet object, with \text{NewFeatNames}, and returns \text{NewESObj}, a new ExpressionSet object.

**Input Arguments**

\(\text{ESObj}\)

Object of the bioma.ExpressionSet class.

\(\text{Subset}\)

One of the following to specify a subset of the feature names in an ExpressionSet object:

- Character vector or string specifying a feature name
- Cell array of character vectors or string vector specifying feature names
- Positive integer
- Vector of positive integers
- Logical vector

\(\text{NewFeatNames}\)

New feature names for specific feature names within an ExpressionSet object, specified by one of the following:

- Numeric vector
- Cell array of character vectors or string vector
- Character vector or string, which \text{featureNames} uses as a prefix for the feature names, with feature numbers appended to the prefix
• Logical true or false (default). If true, featureNames assigns unique feature names using the format Feature1, Feature2, etc.

The number of feature names in NewFeatNames must equal the number of features specified by Subset.

**Output Arguments**

**FeatNames**

Cell array of character vectors specifying all or some of the feature names in an ExpressionSet object. The feature names are the row names in the DataMatrix objects in the ExpressionSet object. The feature names are also the row names of the VarValues dataset array in the MetaData object in the ExpressionSet object.

**NewESObj**

Object of the bioma.ExpressionSet class, returned after replacing specific feature names.

**See Also**

DataMatrix | bioma.ExpressionSet | bioma.data.ExptData | bioma.data.MetaData | sampleNames

**Topics**

"Managing Gene Expression Data in Objects"
**featureNames**

**Class:** bioma.data.ExptData  
**Package:** bioma.data

Retrieve or set feature names in ExptData object

**Syntax**

\[
\text{FeatNames} = \text{featureNames}(\text{EDObj})  
\text{FeatNames} = \text{featureNames}(\text{EDObj}, \text{Subset})  
\text{NewESObj} = \text{featureNames}(\text{EDObj}, \text{Subset}, \text{NewFeatNames})
\]

**Description**

\[
\text{FeatNames} = \text{featureNames}(\text{EDObj}) \text{ returns a cell array of character vectors specifying all feature names in an ExptData object.}
\]

\[
\text{FeatNames} = \text{featureNames}(\text{EDObj}, \text{Subset}) \text{ returns a cell array of character vectors specifying a subset the feature names in an ExptData object.}
\]

\[
\text{NewESObj} = \text{featureNames}(\text{EDObj}, \text{Subset}, \text{NewFeatNames}) \text{ replaces the feature names specified by \text{Subset} in \text{EDObj}, an ExptData object, with \text{NewFeatNames}, and returns \text{NewEDObj}, a new ExptData object.}
\]

**Input Arguments**

**EDObj**

Object of the bioma.data.ExptData class.

**Default:**

**Subset**

One of the following to specify a subset of the feature names in an ExptData object:

- Character vector specifying a feature name
- Cell array of character vectors specifying feature names
- Positive integer
- Vector of positive integers
- Logical vector

**Default:**

**NewFeatNames**

New feature names for specific feature names within an ExptData object, specified by one of the following:
• Numeric vector
• Character vector or cell array of character vectors
• Character vector, which featureNames uses as a prefix for the feature names, with feature numbers appended to the prefix
• Logical true or false (default). If true, featureNames assigns unique feature names using the format Feature1, Feature2, etc.

The number of feature names in NewFeatNames must equal the number of features specified by Subset.

Default:

Output Arguments

FeatNames
Cell array of character vectors specifying all or some of the feature names in an ExptData object. The feature names are the row names in the DataMatrix objects in the ExptData object.

NewEDObj
Object of the bioma.data.ExptData class, returned after replacing specific feature names.

Examples

Construct an ExptData object, and then retrieve the feature names from it:

```matlab
% Import bioma.data package to make constructor functions available
import bioma.data.*
% Create DataMatrix object from .txt file containing expression values from microarray experiment
dmObj = DataMatrix('File', 'mouseExprsData.txt');
% Construct ExptData object
EDObj = ExptData(dmObj);
% Retrieve feature names
FNames = featureNames(EDObj);
```

See Also

DataMatrix | bioma.data.ExptData | dmNames | elementNames | sampleNames

Topics

“Representing Expression Data Values in ExptData Objects”
**featureparse**

Parse features from GenBank, GenPept, or EMBL data

**Syntax**

`FeatStruct = featureparse(Features)`

`FeatStruct = featureparse(Features, ...'Feature', FeatureValue, ...)`

`FeatStruct = featureparse(Features, ...'Sequence', SequenceValue, ...)`

**Input Arguments**

<table>
<thead>
<tr>
<th>Features</th>
<th>Any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• MATLAB structure with fields corresponding to GenBank, GenPept, or EMBL data, such as those returned by genbankread, genpeptread, emblread, getgenbank, getgenpept, or getembl</td>
</tr>
<tr>
<td></td>
<td>• Character vector or character array containing the text from the Features section of a GenBank, GenPept, or EMBL-formatted file</td>
</tr>
</tbody>
</table>

| FeatureValue     | Name of a feature contained in `Features`. When specified, `featureparse` returns only the substructure that corresponds to this feature. If there are multiple features with the same `FeatureValue`, then `FeatStruct` is an array of structures. |

| SequenceValue    | Property to control the extraction, when possible, of the sequences respective to each feature, joining and complementing pieces of the source sequence and storing them in the Sequence field of the returned structure, `FeatStruct`. When extracting the sequence from an incomplete CDS feature, `featureparse` uses the `codon_start` qualifier to adjust the frame of the sequence. Choices are `true` or `false` (default). |

**Output Arguments**

| FeatStruct       | Output structure containing a field for every database feature. Each field name in `FeatStruct` matches the corresponding feature name in the GenBank, GenPept, or EMBL database, with the exceptions listed in the table below. Fields in `FeatStruct` contain substructures with feature qualifiers as fields. In the GenBank, GenPept, and EMBL databases, for each feature, the only mandatory qualifier is its location, which `featureparse` translates to the field `Location`. When possible, `featureparse` also translates this location to numeric indices, creating an `Indices` field. |

**Note** If you use the `Indices` field to extract sequence information, you may need to complement the sequences.
**Description**

`FeatStruct = featureparse(Features)` parses the features from `Features`, which contains GenBank, GenPept, or EMBL features. `Features` can be a:

- Character vector or string containing GenBank, GenPept, or EMBL features
- MATLAB character array including text describing GenBank, GenPept, or EMBL features
- MATLAB structure with fields corresponding to GenBank, GenPept, or EMBL data, such as those returned by `genbankread`, `genpeptread`, `emblread`, `getgenbank`, `getgenpept`, or `getembl`

`FeatStruct` is the output structure containing a field for every database feature. Each field name in `FeatStruct` matches the corresponding feature name in the GenBank, GenPept, or EMBL database, with the following exceptions.

<table>
<thead>
<tr>
<th>Feature Name in GenBank, GenPept, or EMBL Database</th>
<th>Field Name in MATLAB Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10_signal</td>
<td>minus_10_signal</td>
</tr>
<tr>
<td>-35_signal</td>
<td>minus_35_signal</td>
</tr>
<tr>
<td>3'UTR</td>
<td>three_prime_UTR</td>
</tr>
<tr>
<td>3'clip</td>
<td>three_prime_clip</td>
</tr>
<tr>
<td>5'UTR</td>
<td>five_prime_UTR</td>
</tr>
<tr>
<td>5'clip</td>
<td>five_prime_clip</td>
</tr>
<tr>
<td>D-loop</td>
<td>D_loop</td>
</tr>
</tbody>
</table>

Fields in `FeatStruct` contain substructures with feature qualifiers as fields. In the GenBank, GenPept, and EMBL databases, for each feature, the only mandatory qualifier is its location, which `featureparse` translates to the field `Location`. When possible, `featureparse` also translates this location to numeric indices, creating an `Indices` field.

**Note** If you use the `Indices` field to extract sequence information, you may need to complement the sequences.

`FeatStruct = featureparse(Features, ...'PropertyName', PropertyValue, ...) ` calls `featureparse` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`FeatStruct = featureparse(Features, ...'Feature', FeatureValue, ...) ` returns only the substructure that corresponds to `FeatureValue`, the name of a feature contained in `Features`. If there are multiple features with the same `FeatureValue`, then `FeatStruct` is an array of structures.

`FeatStruct = featureparse(Features, ...'Sequence', SequenceValue, ...) ` controls the extraction, when possible, of the sequences respective to each feature, joining and complementing pieces of the source sequence and storing them in the field `Sequence`. When extracting the sequence from an incomplete CDS feature, `featureparse` uses the `codon_start` qualifier to adjust the frame of the sequence. Choices are true or false (default).
Examples

Example 1.16. Obtaining All Features from a GenBank File

The following example obtains all the features stored in the GenBank file nm175642.txt:

```matlab
gbkStruct = genbankread('nm175642.txt');
features = featureparse(gbkStruct)
```

features =

source: [1x1 struct]
gene: [1x1 struct]
CDS: [1x1 struct]

Example 1.17. Obtaining a Subset of Features from a GenBank Record

The following example obtains only the coding sequences (CDS) feature of the Caenorhabditis elegans cosmid record (accession number Z92777) from the GenBank database:

```matlab
worm = getgenbank('Z92777');
CDS = featureparse(worm,'feature','cds')
```

CDS =

1x12 struct array with fields:
Location
Indices
locus_tag
standard_name
note
codon_start
product
protein_id
db_xref
translation

Example 1.18. Extracting Sequences for Each Feature

1 Retrieve two nucleotide sequences from the GenBank database for the neuraminidase (NA) protein of two strains of the Influenza A virus (H5N1).

```matlab
hk01 = getgenbank('AF509094');
vt04 = getgenbank('DQ094287');
```

2 Extract the sequence of the coding region for the neuraminidase (NA) protein from the two nucleotide sequences. The sequences of the coding regions are stored in the Sequence fields of the returned structures, hk01_cds and vt04_cds.

```matlab
hk01_cds = featureparse(hk01,'feature','CDS','Sequence',true);
vt04_cds = featureparse(vt04,'feature','CDS','Sequence',true);
```

3 Once you have extracted the nucleotide sequences, you can use the nt2aa and nwalign functions to align the amino acids sequences converted from the nucleotide sequences.

```matlab
[sc,al]=nwalign(nt2aa(hk01_cds),nt2aa(vt04_cds),'extendgap',1);
```

4 Then you can use the seqinsertgaps function to copy the gaps from the aligned amino acid sequences to their corresponding nucleotide sequences, thus codon-aligning them.
hk01_aligned = seqinsertgaps(hk01_cds,al(1,:))
vt04_aligned = seqinsertgaps(vt04_cds,al(3,:))

Once you have code aligned the two sequences, you can use them as input to other functions such as `dnds`, which calculates the synonymous and nonsynonymous substitutions rates of the codon-aligned nucleotide sequences. By setting `Verbose` to `true`, you can also display the codons considered in the computations and their amino acid translations.

[dn,ds] = dnds(hk01_aligned,vt04_aligned,'verbose',true)

See Also
emblread | genbankread | genpeptread | getgenbank | getgenpept

Introduced in R2006b
featureVarDesc

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set feature variable descriptions in ExpressionSet object

Syntax

DSVarDescriptions = featureVarDesc(ESObj)
NewESObj = featureVarDesc(ESObj, NewDSVarDescriptions)

Description

DSVarDescriptions = featureVarDesc(ESObj) returns a dataset array containing the feature variable names and descriptions from the MetaData object in an ExpressionSet object.

NewESObj = featureVarDesc(ESObj, NewDSVarDescriptions) replaces the feature variable descriptions in ESObj, an ExpressionSet object, with NewDSVarDescriptions, and returns NewESObj, a new ExpressionSet object.

Input Arguments

ESObj
Object of the bioma.ExpressionSet class.

Default:

NewDSVarDescriptions

Descriptions of the feature variable names, specified by either of the following:

• A new dataset array containing the feature variable names and descriptions. In this dataset array, each row corresponds to a variable. The first column contains the variable name, and the second column (VariableDescription) contains a description of the variable. The row names (variable names) must match the row names (variable names) in DSVarDescriptions, the dataset array being replaced in the MetaData object in the ExpressionSet object, ESObj.

• Cell array of character vectors containing descriptions of the feature variables. The number of elements in VarDesc must equal the number of row names (variable names) in DSVarDescriptions, the dataset array being replaced in the MetaData object in the ExpressionSet object, ESObj.

Default:

Output Arguments

DSVarDescriptions

A dataset array containing the feature variable names and descriptions from the MetaData object of an ExpressionSet object. In this dataset array, each row corresponds to a variable. The first column
contains the variable name, and the second column (VariableDescription) contains a description of the variable.

**NewESObj**

Object of the bioma.ExpressionSet class, returned after replacing the dataset array containing the feature variable descriptions.

**See Also**

bioma.ExpressionSet | bioma.data.MetaData | variableDesc

**Topics**

“Managing Gene Expression Data in Objects”
featureVarNames

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set feature variable names in ExpressionSet object

Syntax

FeatVarNames = featureVarNames(ESObj)
FeatVarNames = featureVarNames(ESObj, Subset)
NewESObj = featureVarNames(ESObj, Subset, NewFeatVarNames)

Description

FeatVarNames = featureVarNames(ESObj) returns a cell array of character vectors specifying all feature variable names in an ExpressionSet object.

FeatVarNames = featureVarNames(ESObj, Subset) returns a cell array of character vectors specifying a subset the feature variable names in an ExpressionSet object.

NewESObj = featureVarNames(ESObj, Subset, NewFeatVarNames) replaces the feature variable names specified by Subset in ESObj, an ExpressionSet object, with NewFeatVarNames, and returns NewESObj, a new ExpressionSet object.

Input Arguments

ESObj

Object of the bioma.ExpressionSet class.

Default:

Subset

One of the following to specify a subset of the feature variable names in an ExpressionSet object:

- Character vector or string specifying a feature variable name
- Cell array of character vectors or string vector specifying feature variable names
- Positive integer
- Vector of positive integers
- Logical vector

NewFeatVarNames

New feature variable names for specific feature variable names within an ExpressionSet object, specified by one of the following:

- Numeric vector
• Cell array of character vectors or string vector
• Character array
• Character vector or string, which featureVarNames uses as a prefix for the feature variable names, with feature variable numbers appended to the prefix
• Logical true or false (default). If true, featureVarNames assigns unique feature variable names using the format Var1, Var2, etc.

The number of feature variable names in NewFeatVarNames must equal the number of feature variable names specified by Subset.

## Output Arguments

### FeatVarNames

Cell array of character vectors specifying all or some of the feature variable names in an ExpressionSet object. The feature variable names are the column names of the VarValues dataset array. The feature variable names are also the row names of the VarDescriptions dataset array. Both dataset arrays are in the MetaData object in the ExpressionSet object.

### NewESObj

Object of the bioma.ExpressionSet class, returned after replacing specific feature names.

## See Also

bioma.ExpressionSet | bioma.data.MetaData | featureNames | sampleNames | sampleVarNames

## Topics

“Managing Gene Expression Data in Objects”
**featureVarValues**

Class: bioma.ExpressionSet  
Package: bioma

Retrieve or set feature variable data values in ExpressionSet object

**Syntax**

```matlab
DSVarValues = featureVarValues(ESObj)
NewESObj = featureVarValues(ESObj, NewDSVarValues)
```

**Description**

`DSVarValues = featureVarValues(ESObj)` returns a dataset array containing the measured value of each variable per feature from the MetaData object of an ExpressionSet object.

`NewESObj = featureVarValues(ESObj, NewDSVarValues)` replaces the feature variable values in `ESObj`, an ExpressionSet object, with `NewDSVarValues`, and returns `NewESObj`, a new ExpressionSet object.

**Input Arguments**

*ESObj*
Object of the `bioma.ExpressionSet` class.  
Default:

*NewDSVarValues*
A dataset array containing a value for each variable per feature. In this dataset array, the columns correspond to variables and rows correspond to feature. The row names (feature names) must match the row names (feature names) in `DSVarValues`, the dataset array being replaced in the MetaData object in the ExpressionSet object, `ESObj`.  
Default:

**Output Arguments**

*DSVarValues*
A dataset array containing the measured value of each variable per feature from the MetaData object of an ExpressionSet object. In this dataset array, the columns correspond to variables and rows correspond to features.

*NewESObj*
Object of the `bioma.ExpressionSet` class, returned after replacing the dataset array containing the feature variable values.
See Also
bioma.ExpressionSet | bioma.data.MetaData | variableValues

Topics
“Managing Gene Expression Data in Objects”
**featureview**

Draw linear or circular map of features from GenBank structure

**Syntax**

```
featureview(GBStructure)
featureview(GBStructure, FeatList)
featureview(GBStructure, FeatList, Levels)
featureview(GBStructure, Levels)
[Handles, OutFeatList] = featureview(...)
```

```
featureview(..., 'FontSize', FontSizeValue, ...)
featureview(..., 'ColorMap', ColorMapValue, ...)
featureview(..., 'Qualifiers', QualifiersValue, ...)
featureview(..., 'ShowPositions', ShowPositionsValue, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GBStructure</strong></td>
<td>GenBank structure, typically created using the getgenbank or the genbankread function.</td>
</tr>
<tr>
<td><strong>FeatList</strong></td>
<td>String vector or cell array of character vectors specifying feature names (from the list of all features in the GenBank structure) to include in or exclude from the map.</td>
</tr>
<tr>
<td></td>
<td>• If <strong>FeatList</strong> is a string vector or cell array of character vectors containing feature names, these features are mapped. Any features in <strong>FeatList</strong> not found in the GenBank structure are ignored.</td>
</tr>
<tr>
<td></td>
<td>• If <strong>FeatList</strong> includes <code>-</code> as the first element in the cell array or string vector, then the remaining strings (features) are not mapped.</td>
</tr>
<tr>
<td></td>
<td>By default, <strong>FeatList</strong> is the a list of all features in the GenBank structure.</td>
</tr>
<tr>
<td><strong>Levels</strong></td>
<td>Vector of N integers, where N is the number of features. Each integer represents the level in the map for the corresponding feature. For example, if <strong>Levels</strong> = [1, 1, 2, 3, 3], the first two features would appear on level 1, the third feature on level 2, and the fourth and fifth features on level 3. By default, <strong>Levels</strong> = [1:N].</td>
</tr>
<tr>
<td><strong>FontSizeValue</strong></td>
<td>Scalar that sets the font size (points) for the annotations of the features. Default is 9.</td>
</tr>
</tbody>
</table>
**ColorMapValue**
Three-column matrix, to specify a list of colors to use for each feature. This matrix replaces the default matrix, which specifies the following colors and order: blue, green, red, cyan, magenta, yellow, brown, light green, orange, purple, gold, and silver. In the matrix, each row corresponds to a color, and each column specifies red, green, and blue intensity respectively. Valid values for the RGB intensities are 0.0 to 1.0.

**QualifiersValue**
Cell array of character vectors or string vector to specify an ordered list of qualifiers to search for in the structure and use as annotations. For each feature, the first matching qualifier found from the list is used for its annotation. If a feature does not include any of the qualifiers, no annotation displays for that feature. By default, `QualifiersValue = {'gene', 'product', 'locus_tag', 'note', 'db_xref', 'protein_id'}`. Provide your own `QualifiersValue` to limit or expand the list of qualifiers or change the search order.

**Tip** Set `QualifiersValue = {}` to create a map with no annotations.

**Tip** To determine all qualifiers available for a given feature, do either of the following:
- Create the map, and then click a feature or its annotation to list all qualifiers for that feature.
- Use the `featureparse` command to parse all the features into a new structure, and then use the `fieldnames` command to list the qualifiers for a specific feature.

**ShowPositionsValue**
Property to add the sequence position to the annotation label for each feature. Enter `true` to add the sequence position. Default is `false`.

**Description**

`featureview(GBStructure)` creates a linear or circular map of all features from a GenBank structure, typically created using the `getgenbank` or the `genbankread` function.

`featureview(GBStructure,FeatList)` creates a linear or circular map of a subset of features from a GenBank structure. `FeatList` lets you specify features (from the list of all features in the GenBank structure) to include in or exclude from the map.

- If `FeatList` is a cell array of features, these features are mapped. Any features in `FeatList` not found in the GenBank structure are ignored.
- If `FeatList` includes `'-'` as the first string in the cell array, then the remaining strings (features) are not mapped.

By default, `FeatList` is a list of all features in the GenBank structure.
featureview(GBStructure, FeatList, Levels) or featureview(GBStructure, Levels) indicates which level on the map each feature is drawn. Level 1 is the left-most (linear map) or inner-most (circular map) level, and level N is the right-most (linear map) or outer-most (circular map) level, where N is the number of features.

Levels is a vector of N integers, where N is the number of features. Each integer represents the level in the map for the corresponding feature. For example, if Levels = [1, 1, 2, 3, 3], the first two features would appear on level 1, the third feature on level 2, and the fourth and fifth features on level 3. By default, Levels = [1:N].

[Handles, OutFeatList] = featureview(...) returns a list of handles for each feature in OutFeatList. It also returns OutFeatList, which is a cell array of the mapped features.

Tip Use Handles and OutFeatList with the legend command to create a legend of features.

featureview(..., 'PropertyName', PropertyValue, ...) defines optional properties that use property name/value pairs in any order. These property name/value pairs are as follows:

featureview(..., 'FontSize', FontSizeValue, ...) sets the font size (points) for the annotations of the features. Default FontSizeValue is 9.

featureview(..., 'ColorMap', ColorMapValue, ...) specifies a list of colors to use for each feature. This matrix replaces the default matrix, which specifies the following colors and order: blue, green, red, cyan, magenta, yellow, brown, light green, orange, purple, gold, and silver. ColorMapValue is a three-column matrix, where each row corresponds to a color, and each column specifies red, green, and blue intensity respectively. Valid values for the RGB intensities are 0.0 to 1.0.

featureview(..., 'Qualifiers', QualifiersValue, ...) lets you specify an ordered list of qualifiers to search for and use as annotations. For each feature, the first matching qualifier found from the list is used for its annotation. If a feature does not include any of the qualifiers, no annotation displays for that feature. QualifiersValue is a cell array of character vectors or string vector. By default, QualifiersValue = {'gene', 'product', 'locus_tag', 'note', 'db_xref', 'protein_id'}. Provide your own QualifiersValue to limit or expand the list of qualifiers or change the search order.

Tip Set QualifiersValue = {} to create a map with no annotations.

Tip To determine all qualifiers available for a given feature, do either of the following:

- Create the map, and then click a feature or its annotation to list all qualifiers for that feature.
- Use the featureparse command to parse all the features into a new structure, and then use the fieldnames command to list the qualifiers for a specific feature.

featureview(..., 'ShowPositions', ShowPositionsValue, ...) lets you add the sequence position to the annotation label. If ShowPositionsValue is true, sequence positions are added to the annotation labels. Default is false.
After creating a map:

- Click a feature or annotation to display a list of all qualifiers for that feature.
- Zoom the plot by clicking the following buttons:
Examples

Example 1.19. Creating a Circular Map with a Legend

The following example creates a circular map of five different features mapped on three levels. It also uses outputs from the featureview function as inputs to the legend function to add a legend to the map.

```matlab
GBStructure = getgenbank('J01415');
[Handles, OutFeatList] = featureview(GBStructure, ...
    {'CDS','D_loop','mRNA','tRNA','rRNA'}, [1 2 2 2 3])
legend(Handles, OutFeatList, 'interpreter', 'none', ...
    'location','bestoutside')
title('Human Mitochondrion, Complete Genome')
```

Example 1.20. Creating a Linear Map with Sequence Position Labels and Changed Font Size

The following example creates a linear map showing only the gene feature. It changes the font of the labels to seven points and includes the sequence position in the labels.

```matlab
herpes = getgenbank('NC_001348');
featureview(herpes,{'gene'},'fontsize',7,'showpositions',true)
title('Genes in Human herpesvirus 3 (strain Dumas')
```

Example 1.21. Determining Qualifiers for a Specific Feature

The following example uses the getgenbank function to create a GenBank structure, GBStructure. It then uses the featureparse function to parse the features in the GenBank structure into a new structure, features. It then uses the fieldnames function to return all qualifiers for one of the features, D_loop.

```matlab
GenBankStructure = getgenbank('J01415');
features = featureparse (GenBankStructure)
features =
    source: [1x1 struct]
    D_loop: [1x2 struct]
    rep_origin: [1x3 struct]
    repeat_unit: [1x4 struct]
    misc_signal: [1x1 struct]
    misc_RNA: [1x1 struct]
    variation: [1x17 struct]
    tRNA: [1x22 struct]
    rRNA: [1x2 struct]
    mRNA: [1x10 struct]
    CDS: [1x13 struct]
    conflict: [1x1 struct]

fieldnames(features.D_loop)
ans =
    'Location'
    'Indices'
    'note'
    'citation'
```
See Also
featureparse | genbankread | getgenbank | seqviewer

Introduced in R2006b
galread

Read microarray data from GenePix array list file

Syntax

\[ \text{GALData} = \text{galread}(	ext{File}) \]

Arguments

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>GenePix® array list formatted file (GAL). Enter a character vector or string specifying a file name, or a path and file name.</td>
</tr>
</tbody>
</table>

Description

galread reads data from a GenePix formatted file into a MATLAB structure.

\[ \text{GALData} = \text{galread}(\text{File}) \]
reads in a GenePix array list formatted file (\text{File}) and creates a structure (GALData) containing the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td></td>
</tr>
<tr>
<td>BlockData</td>
<td>An (N)-by-3 array. The columns of this array are the block data, the column data, and the row data respectively. For more information on the GAL format, see here.</td>
</tr>
<tr>
<td>IDs</td>
<td></td>
</tr>
<tr>
<td>Names</td>
<td></td>
</tr>
</tbody>
</table>

See Also

affyread | geoseriestread | geosoftread | gprread | ilmnbsread | imageneread | sptread

Introduced before R2006a
**gcrma**

Perform GC Robust Multi-array Average (GCRMA) background adjustment, quantile normalization, and median-polish summarization on Affymetrix microarray probe-level data

**Syntax**

```matlab
ExpressionMatrix = gcrma(PMMatrix, MMMatrix, ProbeIndices, AffinPM, AffinMM)
ExpressionMatrix = gcrma(PMMatrix, MMMatrix, ProbeIndices, SequenceMatrix)
```

```matlab
ExpressionMatrix = gcrma(..., 'ChipIndex', ChipIndexValue, ...)
ExpressionMatrix = gcrma(..., 'OpticalCorr', OpticalCorrValue, ...)
ExpressionMatrix = gcrma(..., 'CorrConst', CorrConstValue, ...)
ExpressionMatrix = gcrma(..., 'Method', MethodValue, ...)
ExpressionMatrix = gcrma(..., 'TuningParam', TuningParamValue, ...)
ExpressionMatrix = gcrma(..., 'GSBCorr', GSBCorrValue, ...)
ExpressionMatrix = gcrma(..., 'Normalize', NormalizeValue, ...)
ExpressionMatrix = gcrma(..., 'Verbose', VerboseValue, ...)
```

**Input Arguments**

<table>
<thead>
<tr>
<th><strong>Input Argument</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PMMatrix</strong></td>
<td>Matrix of intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)</td>
</tr>
<tr>
<td><strong>MMMMatrix</strong></td>
<td>Matrix of intensity values where each row corresponds to a mismatch (MM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)</td>
</tr>
<tr>
<td><strong>ProbeIndices</strong></td>
<td>Column vector containing probe indices. Probes within a probe set are numbered 0 through ( N - 1 ), where ( N ) is the number of probes in the probe set.</td>
</tr>
<tr>
<td><strong>AffinPM</strong></td>
<td>Column vector of PM probe affinities.</td>
</tr>
</tbody>
</table>

**Tip** You can use the PMIntensities matrix returned by the `celintensityread` function.

**Tip** You can use the MMIntensities matrix returned by the `celintensityread` function.

**Tip** You can use the `affyprobeseqread` function to generate this column vector.

**Tip** You can use the `affyprobeaffinities` function to generate this column vector.
AffinMM | Column vector of MM probe affinities.

**Tip** You can use the `affyprobeaffinities` function to generate this column vector.

SequenceMatrix | An N-by-25 matrix of sequence information for the perfect match (PM) probes on the Affymetrix GeneChip array, where N is the number of probes on the array. Each row corresponds to a probe, and each column corresponds to one of the 25 sequence positions. Nucleotides in the sequences are represented by one of the following integers:

- 0 — None
- 1 — A
- 2 — C
- 3 — G
- 4 — T

**Tip** You can use the `affyprobeseqread` function to generate this matrix. If you have this sequence information in letter representation, you can convert it to integer representation using the `nt2int` function.

ChipIndexValue | Positive integer specifying a column index in `MMMatrinx`, which specifies a chip. This chip intensity data is used to compute probe affinities. Default is 1.

OpticalCorrValue | Controls the use of optical background correction on the PM and MM intensity values in `PMMatrix` and `MMMatrix`. Choices are true (default) or false.

CorrConstValue | Value that specifies the correlation constant, rho, for background intensity for each PM/MM probe pair. Choices are any value \( \geq 0 \) and \( \leq 1 \). Default is 0.7.

MethodValue | Character vector or string that specifies the method to estimate the signal. Choices are ‘MLE’, a faster, ad hoc Maximum Likelihood Estimate method, or ‘EB’, a slower, more formal, empirical Bayes method. Default is ‘MLE’.

TuningParamValue | Value that specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is 5 (MLE) or 0.5 (EB).

**Tip** For information on determining a setting for this parameter, see Wu et al., 2004 on page 1-659.

GSBCorrValue | Specifies whether to perform gene-specific binding (GSB) correction using probe affinity data. Choices are true (default) or false. If there is no probe affinity information, this property is ignored.

NormalizeValue | Controls whether quantile normalization is performed on background adjusted data. Choices are true (default) or false.
**VerboseValue**  
Controls the display of a progress report showing the number of each chip as it is completed. Choices are true (default) or false.

**Output Arguments**

| **ExpressionMatrix** | Matrix of log₂ expression values where each row corresponds to a gene (probe set) and each column corresponds to an Affymetrix CEL file, which represents a single chip. |

**Description**

\[ \text{ExpressionMatrix} = \text{gcrma}(\text{PMMatrix, MMMatrix, ProbeIndices, AffinPM, AffinMM}) \]

performs GCRMA background adjustment, quantile normalization, and median-polish summarization on Affymetrix microarray probe-level data using probe affinity data. \( \text{ExpressionMatrix} \) is a matrix of log₂ expression values where each row corresponds to a gene (probe set) and each column corresponds to an Affymetrix CEL file, which represents a single chip.

**Note**  
There is no column in \( \text{ExpressionMatrix} \) that contains probe set or gene information.

\[ \text{ExpressionMatrix} = \text{gcrma}(\text{PMMatrix, MMMatrix, ProbeIndices, SequenceMatrix}) \]

performs GCRMA background adjustment, quantile normalization, and Robust Multi-array Average (RMA) summarization on Affymetrix microarray probe-level data using probe sequence data to compute probe affinity data. \( \text{ExpressionMatrix} \) is a matrix of log₂ expression values where each row corresponds to a gene (probe set) and each column corresponds to an Affymetrix CEL file, which represents a single chip.

**Note**  
If \( \text{AffinPM} \) and \( \text{AffinMM} \) affinity data and \( \text{SequenceMatrix} \) sequence data are not available, you can still use the gcrma function by entering an empty matrix for these inputs in the syntax.

\[ \text{ExpressionMatrix} = \text{gcrma}( \ldots '\text{PropertyName}', \text{PropertyValue}, \ldots ) \]

calls gcrma with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each \( \text{PropertyName} \) must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

\[ \text{ExpressionMatrix} = \text{gcrma}( \ldots , \text{ChipIndex}', \text{ChipIndexValue}, \ldots ) \]

computes probe affinities from MM probe intensity data from the chip with the specified column index in \( \text{MMMatrix} \). Default \( \text{ChipIndexValue} \) is 1. If \( \text{AffinPM} \) and \( \text{AffinMM} \) affinity data are provided, this property is ignored.

\[ \text{ExpressionMatrix} = \text{gcrma}( \ldots , \text{OpticalCorr}', \text{OpticalCorrValue}, \ldots ) \]

controls the use of optical background correction on the PM and MM intensity values in \( \text{PMMatrix} \) and \( \text{MMMatrix} \). Choices are true (default) or false.

\[ \text{ExpressionMatrix} = \text{gcrma}( \ldots , \text{CorrConst}', \text{CorrConstValue}, \ldots ) \]

specifies the correlation constant, rho, for background intensity for each PM/MM probe pair. Choices are any value \( \geq 0 \) and \( \leq 1 \). Default is 0.7.
ExpressionMatrix = gcrma(..., 'Method', MethodValue, ...) specifies the method to estimate the signal. Choices are MLE, a faster, ad hoc Maximum Likelihood Estimate method, or EB, a slower, more formal, empirical Bayes method. Default is MLE.

ExpressionMatrix = gcrma(..., 'TuningParam', TuningParamValue, ...) specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is 5 (MLE) or 0.5 (EB).

**Tip** For information on determining a setting for this parameter, see Wu et al., 2004 on page 1-659.

ExpressionMatrix = gcrma(..., 'GSBCorr', GSBCorrValue, ...) specifies whether to perform gene specific binding (GSB) correction using probe affinity data. Choices are true (default) or false. If there is no probe affinity information, this property is ignored.

ExpressionMatrix = gcrma(..., 'Normalize', NormalizeValue, ...) controls whether quantile normalization is performed on background adjusted data. Choices are true (default) or false.

ExpressionMatrix = gcrma(..., 'Verbose', VerboseValue, ...) controls the display of a progress report showing the number of each chip as it is completed. Choices are true (default) or false.

**Examples**

1. Load the MAT-file, included with the Bioinformatics Toolbox software, that contains Affymetrix data from a prostate cancer study. The variables in the MAT-file include seqMatrix, a matrix containing sequence information for PM probes, pmMatrix and mmMatrix, matrices containing PM and MM probe intensity values, and probeIndices, a column vector containing probe indexing information.

   load prostatecancerrawdata

2. Compute the Affymetrix PM and MM probe affinities from their sequences and MM probe intensities.

   [apm, amm] = affyprobeaffinities(seqMatrix, mmMatrix(:,1),...  
   'ProbeIndices', probeIndices);

3. Perform GCRMA background adjustment, quantile normalization, and Robust Multi-array Average (RMA) summarization on the Affymetrix microarray probe-level data and create a matrix of expression values.

   expdata = gcrma(pmMatrix, mmMatrix, probeIndices, seqMatrix);

The prostatecancerrawdata.mat file used in this example contains data from Best et al., 2005 on page 1-654.

**References**


See Also
affygcrma | affyprobeseqread | affyread | affyrma | celintensityread | gcrmabackadj | quantilenorm | rmabackadj | rmasummary

Introduced in R2007a
gcrmabackadj

Perform GC Robust Multi-array Average (GCRMA) background adjustment on Affymetrix microarray probe-level data using sequence information.

Syntax

\[
PMMatrix_{\text{Adj}} = \text{gcrmabackadj}(PMMatrix, \text{MMMatrix}, \text{AffinPM}, \text{AffinMM})
\]

\[
[PMMatrix_{\text{Adj}}, \text{nsbStruct}] = \text{gcrmabackadj}(PMMatrix, \text{MMMatrix}, \text{AffinPM}, \text{AffinMM})
\]

\[
... = \text{gcrmabackadj}(...'\text{OpticalCorr}', \text{OpticalCorrValue}, ...)
\]

\[
... = \text{gcrmabackadj}(...'\text{CorrConst}', \text{CorrConstValue}, ...)
\]

\[
... = \text{gcrmabackadj}(...'\text{Method}', \text{MethodValue}, ...)
\]

\[
... = \text{gcrmabackadj}(...'\text{TuningParam}', \text{TuningParamValue}, ...)
\]

\[
... = \text{gcrmabackadj}(...'\text{AddVariance}', \text{AddVarianceValue}, ...)
\]

\[
... = \text{gcrmabackadj}(...'\text{GSBCorr}', \text{GSBCorrValue}, ...)
\]

\[
... = \text{gcrmabackadj}(...'\text{Showplot}', \text{ShowplotValue}, ...)
\]

\[
... = \text{gcrmabackadj}(...'\text{Verbose}', \text{VerboseValue}, ...)
\]

Input Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMMatrix</td>
<td>Matrix of intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)</td>
</tr>
<tr>
<td><strong>Tip</strong> You can use the PMIntensities matrix returned by the celintensityread function.</td>
<td></td>
</tr>
<tr>
<td>MMMatrix</td>
<td>Matrix of intensity values where each row corresponds to a mismatch (MM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)</td>
</tr>
<tr>
<td><strong>Tip</strong> You can use the MMIntensities matrix returned by the celintensityread function.</td>
<td></td>
</tr>
<tr>
<td>AffinPM</td>
<td>Column vector of PM probe affinities, such as returned by the affyprobeaffinities function. Each row corresponds to a probe.</td>
</tr>
<tr>
<td>AffinMM</td>
<td>Column vector of MM probe affinities, such as returned by the affyprobeaffinities function. Each row corresponds to a probe.</td>
</tr>
<tr>
<td>OpticalCorrValue</td>
<td>Controls the use of optical background correction on the PM and MM probe intensity values in PMMatrix and MMMatrix. Choices are true (default) or false.</td>
</tr>
</tbody>
</table>
### Output Arguments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMMatrix_Adj</td>
<td>Matrix of background adjusted PM (perfect match) intensity values.</td>
</tr>
<tr>
<td>nsbStruct</td>
<td>Structure containing nonspecific binding background parameters, estimated from the intensities and affinities of probes on an Affymetrix GeneChip array. nsbStruct includes the following fields:</td>
</tr>
<tr>
<td></td>
<td>• sigma</td>
</tr>
<tr>
<td></td>
<td>• mu_pm</td>
</tr>
<tr>
<td></td>
<td>• mu_mm</td>
</tr>
</tbody>
</table>

### Description

`PMMatrix_Adj = gcrmabackadj(PMMatrix, MMMatrix, AffinPM, AffinMM)` performs GCRMA background adjustment (including optical background correction and nonspecific binding correction) on Affymetrix microarray probe-level data, using probe sequence information and returns `PMMatrix_Adj`, a matrix of background adjusted PM (perfect match) intensity values.
**Note** If AffinPM and AffinMM data are not available, you can still use the `gcrmabackadj` function by entering empty column vectors for both of these inputs in the syntax.

```matlab
[PMMatrix_Adj, nsbStruct] = gcrmabackadj(PMMatrix, MMMatrix, AffinPM, AffinMM)
``` returns `nsbStruct`, a structure containing nonspecific binding background parameters, estimated from the intensities and affinities of probes on an Affymetrix GeneChip array. `nsbStruct` includes the following fields:

- `sigma`
- `mu_pm`
- `mu_mm`

```matlab
... = gcrmabackadj(...'PropertyName', PropertyValue, ...)
``` calls `gcrmabackadj` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

```matlab
... = gcrmabackadj(...'OpticalCorr', OpticalCorrValue, ...)
``` controls the use of optical background correction on the PM and MM probe intensity values in `PMMatrix` and `MMMatrix`. Choices are true (default) or false.

```matlab
... = gcrmabackadj(...'CorrConst', CorrConstValue, ...)
``` specifies the correlation constant, rho, for log background intensity for each PM/MM probe pair. Choices are any value $\geq 0$ and $\leq 1$. Default is 0.7.

```matlab
... = gcrmabackadj(...'Method', MethodValue, ...)
``` specifies the method to estimate the signal. Choices are MLE, a faster, ad hoc Maximum Likelihood Estimate method, or EB, a slower, more formal, empirical Bayes method. Default is MLE.

```matlab
... = gcrmabackadj(...'TuningParam', TuningParamValue, ...)
``` specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is 5 (MLE) or 0.5 (EB).

**Tip** For information on determining a setting for this parameter, see Wu et al., 2004 on page 1-659.

```matlab
... = gcrmabackadj(...'AddVariance', AddVarianceValue, ...)
``` controls whether the signal variance is added to the weight function for smoothing low signal edge. Choices are true or false (default).

```matlab
... = gcrmabackadj(...'GSBCorr', GSBCorrValue, ...)
``` specifies whether to perform gene specific binding (GSB) correction using probe affinity data. Choices are true (default) or false. If there is no probe affinity information, this property is ignored.

```matlab
... = gcrmabackadj(...'Showplot', ShowplotValue, ...)
``` controls the display of a plot showing the log$_2$ of probe intensity values from a specified column (chip) in `MMMatrix`, versus probe affinities in `AffinMM`. Choices are true, false, or I, an integer specifying a column in `MMMatrix`. If set to true, the first column in `MMMatrix` is plotted. Default is:

- **false** — When return values are specified.
• **true** — When return values are not specified.

... = gcrmabackadj(...'Verbose', VerboseValue, ...) controls the display of a progress report showing the number of each chip as it is completed. Choices are **true** (default) or **false**.

**Examples**

1. Load the MAT-file, included with the Bioinformatics Toolbox software, that contains Affymetrix data from a prostate cancer study. The variables in the MAT-file include **seqMatrix**, a matrix containing sequence information for PM probes, **pmMatrix** and **mmMatrix**, matrices containing PM and MM probe intensity values, and **probeIndices**, a column vector containing probe indexing information.

   load prostatecancerrawdata

2. Compute the Affymetrix PM and MM probe affinities from their sequences and MM probe intensities.

   [apm, amm] = affyprobeaffinities(seqMatrix, mmMatrix(:,1), ...
   'ProbeIndices', probeIndices);

3. Perform GCRMA background adjustment on the Affymetrix microarray probe-level data, creating a matrix of background adjusted PM intensity values. Also, display a plot showing the $\log_2$ of probe intensity values from column 3 (chip 3) in **mmMatrix**, versus probe affinities in **amm**.

   pms_adj = gcrmabackadj(pmMatrix, mmMatrix, apm, amm, 'showplot', 3);
Perform GCRMA background adjustment again, using the slower, more formal, empirical Bayes method.

```matlab
pms_adj2 = gcrmabackadj(pmMatrix, mmMatrix, apm, amm, 'method', 'EB');
```

The `prostatecancerrawdata.mat` file used in this example contains data from Best et al., 2005.

### References


### See Also

`affygcrma` | `affyprobeseqread` | `affyread` | `celintensityread` | `probelibraryinfo`
ge (DataMatrix)

Test DataMatrix objects for greater than or equal to

Syntax

\[
T = \text{ge}(\text{DMObj1}, \text{DMObj2}) \\
T = \text{DMObj1} \geq \text{DMObj2} \\
T = \text{ge}(\text{DMObj1}, B) \\
T = \text{DMObj1} \geq B \\
T = \text{ge}(B, \text{DMObj1}) \\
T = B \geq \text{DMObj1}
\]

Input Arguments

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>MATLAB numeric or logical array.</td>
</tr>
</tbody>
</table>

Output Arguments

| T             | Logical matrix of the same size as DMObj1 and DMObj2 or DMObj1 and B. It contains logical 1 (true) where elements in the first input are greater than or equal to the corresponding element in the second input, and logical 0 (false) otherwise. |

Description

\[
T = \text{ge}(\text{DMObj1}, \text{DMObj2}) \text{ or the equivalent } T = \text{DMObj1} \geq \text{DMObj2} \text{ compares each element in DataMatrix object DMObj1 to the corresponding element in DataMatrix object DMObj2, and returns } T, \text{ a logical matrix of the same size as DMObj1 and DMObj2, containing logical 1 (true) where elements in DMObj1 are greater than or equal to the corresponding element in DMObj2, and logical 0 (false) otherwise. DMObj1 and DMObj2 must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). DMObj1 and DMObj2 can have different Name properties.}
\]

\[
T = \text{ge}(\text{DMObj1}, B) \text{ or the equivalent } T = \text{DMObj1} \geq B \text{ compares each element in DataMatrix object DMObj1 to the corresponding element in } B, \text{ a numeric or logical array, and returns } T, \text{ a logical matrix of the same size as DMObj1 and B, containing logical 1 (true) where elements in DMObj1 are greater than or equal to the corresponding element in } B, \text{ and logical 0 (false) otherwise. DMObj1 and B must have the same size (number of rows and columns), unless one is a scalar.}
\]

\[
T = \text{ge}(B, \text{DMObj1}) \text{ or the equivalent } T = B \geq \text{DMObj1} \text{ compares each element in } B, \text{ a numeric or logical array, to the corresponding element in DataMatrix object DMObj1, and returns } T, \text{ a logical matrix of the same size as } B \text{ and DMObj1, containing logical 1 (true) where elements in } B \text{ are greater than or equal to the corresponding element in DMObj1, and logical 0 (false) otherwise. } B \text{ and DMObj1 must have the same size (number of rows and columns), unless one is a scalar.}
\]

MATLAB calls \( T = \text{ge}(X, Y) \) for the syntax \( T = X \geq Y \) when \( X \) or \( Y \) is a DataMatrix object.
See Also
DataMatrix | le

Topics
DataMatrix object on page 1-532

Introduced in R2008b
genbankread

Read data from GenBank file

Syntax

GenBankData = genbankread(File)

Arguments

<table>
<thead>
<tr>
<th>File</th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a GenBank-formatted file (ASCII text file). If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB character array or string vector that contains the text of a GenBank-formatted file.</td>
</tr>
</tbody>
</table>

Tip You can use the getgenbank function with the 'ToFile' property to retrieve sequence information from the GenBank database and create an GenBank-formatted file.

GenBankData MATLAB structure or array of structures containing fields corresponding to GenBank keywords.

Description

GenBankData = genbankread(File) reads a GenBank-formatted file, File, and creates GenBankData, a structure or array of structures, containing fields corresponding to the GenBank keywords. When File contains multiple entries, each entry is stored as a separate element in GenBankData. For a list of the GenBank keywords, see https://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html.

Examples

Retrieve sequence information for the HEXA gene, store the data in a file, and then read into the MATLAB software.

getgenbank('nm_000520', 'ToFile', 'TaySachs_Gene.txt')
s = genbankread('TaySachs_Gene.txt')

s =

    LocusName: 'NM_000520'
    LocusSequenceLength: '2437'
    LocusNumberOfStrands: ''
    LocusTopology: 'linear'
    LocusMoleculeType: 'mRNA'
    LocusGenBankDivision: 'PRI'
Display the source organism for this sequence.

```matlab
s.SourceOrganism
ans =
Homo sapiens
Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
Mammalia; Eutheria; Euarchontoglires; Primates; Haplorrhini;
Catarrhini; Hominidae; Homo.
```

See Also
emblread | fastaread | genpeptread | getgenbank | scfread | seqviewer

Introduced before R2006a
geneentropyfilter

Remove genes with low entropy expression values

Syntax

\[
\text{Mask} = \text{geneentropyfilter}(\text{Data})
\]

\[
[\text{Mask}, \text{FData}] = \text{geneentropyfilter}(\text{Data})
\]

\[
[\text{Mask}, \text{FData}, \text{FNames}] = \text{geneentropyfilter}(\text{Data}, \text{Names})
\]

\[
\text{geneentropyfilter}(\ldots, '\text{Percentile}', \text{PercentileValue})
\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Data}</td>
<td>DataMatrix object on page 1-532 or numeric matrix where each row corresponds to the experimental results for one gene. Each column is the results for all genes from one experiment.</td>
</tr>
<tr>
<td>\text{Names}</td>
<td>Cell array of character vectors or string vector where each element corresponds to the name of a gene for each row of experimental data. \text{Names} has same number of rows as \text{Data} with each row containing the name or ID of the gene in the data set.</td>
</tr>
<tr>
<td>\text{PercentileValue}</td>
<td>Property to specify a percentile below which gene data is removed. Enter a value from 0 to 100.</td>
</tr>
</tbody>
</table>

Description

\text{Mask} = \text{geneentropyfilter}(\text{Data}) identifies gene expression profiles in \text{Data} with entropy values less than the 10th percentile.

\text{Mask} is a logical vector with one element for each row in \text{Data}. The elements of \text{Mask} corresponding to rows with a variance greater than the threshold have a value of 1, and those with a variance less than the threshold are 0.

\text{[Mask, FData]} = \text{geneentropyfilter}(\text{Data}) returns \text{FData}, a filtered data matrix. You can also create \text{FData} using \text{FData} = \text{Data}(\text{Mask},:).

\text{[Mask, FData, FNames]} = \text{geneentropyfilter}(\text{Data, Names}) returns \text{FNames}, a filtered names array, where \text{Names} is a cell array of character vectors or string vector of the names of the genes corresponding to each row of \text{Data}. You can also create \text{FNames} using \text{FNames} = \text{Names}(\text{Mask}).

\text{Note} If \text{Data} is a DataMatrix object with specified row names, you do not need to provide the second input \text{Names} to return the third output \text{FNames}.

geneentropyfilter(..., 'Percentile', \text{PercentileValue}) removes from \text{Data}, the experimental data, gene expression profiles with entropy values less than \text{PercentileValue}, the specified percentile.
Examples
1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: yeastvalues, a matrix of gene expression data, genes, a cell array of GenBank accession numbers for labeling the rows in yeastvalues, and times, a vector of time values for labeling the columns in yeastvalues.

load yeastdata

2 Remove genes with low entropy expression values.

[fyeastvalues, fgenes] = geneentropyfilter(yeastvalues,genes);

References

See Also
exprprofrange | exprprofvar | genelowvalfilter | generangefilter | genevarfilter

Introduced before R2006a
**genelowvalfilter**

Remove gene profiles with low absolute values

**Syntax**

\[
\text{Mask} = \text{genelowvalfilter}(\text{Data})
\]

\[
[\text{Mask},\text{FData}] = \text{genelowvalfilter}(\text{Data})
\]

\[
[\text{Mask},\text{FData},\text{FNames}] = \text{genelowvalfilter}(\text{Data},\text{geneNames})
\]

\[
[\_\_\_] = \text{genelowvalfilter}(\_\_,\text{Name},\text{Value})
\]

**Description**

\[
\text{Mask} = \text{genelowvalfilter}(\text{Data}) \text{ returns a logical vector } \text{Mask} \text{ identifying gene expression profiles in } \text{Data} \text{ that have absolute expression levels in the lowest 10\% of the data set.}
\]

Gene expression profile experiments have data where the absolute values are very low. The quality of this type of data is often bad due to large quantization errors or simply poor spot hybridization. Use this function to filter data.

\[
[\text{Mask},\text{FData}] = \text{genelowvalfilter}(\text{Data}) \text{ also returns } \text{FData}, \text{ a data matrix containing filtered expression profiles.}
\]

\[
[\text{Mask},\text{FData},\text{FNames}] = \text{genelowvalfilter}(\text{Data},\text{geneNames}) \text{ also returns } \text{FNames}, \text{ a cell array of filtered gene names or IDs. You have to specify } \text{geneNames} \text{ to return } \text{FNames} \text{ unless } \text{Data} \text{ is a DataMatrix object with specified row names.}
\]

\[
[\_\_\_] = \text{genelowvalfilter}(\_\_,\text{Name},\text{Value}) \text{ uses additional options specified as one or more optional name-value pair arguments.}
\]

**Examples**

**Filter Out Genes with Low Absolute Expression Levels**

Load the sample yeast data.

\[
\text{load yeastdata;}
\]

Retrieve the genes and corresponding expression data where absolute expression levels exceed the 10th percentile.

\[
[\text{mask},\text{filteredData},\text{filteredGenes}] = \text{genelowvalfilter}(\text{yeastvalues},\text{genes});
\]

Compare the number of filtered genes (filteredGenes) with the number of genes in the original data set (genes).

\[
\text{size (filteredGenes,1)}
\]

\[
\text{ans} = 6394
\]
Filter Out Genes with Low Absolute Expression Levels Using a Logical Vector

Load the sample yeast data.

```matlab
load yeastdata;
```

Mark the genes that have low absolute expression levels below the 10th percentile of the data set.

```matlab
mask = genelowvalfilter(yeastvalues);
```

The variable `genes` contains every gene names in the yeast data set. Use the generated logical vector `mask` to retrieve the genes where expression levels exceed the 10th percentile.

```matlab
filteredGenes = genes(mask);
```

Extract corresponding expression profile data for the selected genes from the variable `yeastvalues`, which contains expression profiles of every gene in the yeast data set.

```matlab
filteredData = yeastvalues(mask,:);
```

Filter Out Genes with Absolute Expression Levels that are Lower Than a User-Defined Threshold

Load the sample yeast data.

```matlab
load yeastdata;
```

Retrieve the genes and corresponding expression data where absolute expression levels exceed the 30th percentile of the data set.

```matlab
[mask,filteredData,filteredGenes] = genelowvalfilter(yeastvalues,genes,'Percentile',30);
```

Compare the number of filtered genes (`filteredGenes`) with the number of genes in the original data set (`genes`).

```matlab
size (filteredGenes,1)
```

ans = 6384

Input Arguments

- **Data** — Input data
  - DataMatrix object | numeric matrix

  Input data, specified as a `DataMatrix` object or numeric matrix. Each row of the matrix corresponds to the experimental results for one gene. Each column represents the results for all genes from one experiment.

- **geneNames** — Gene names or IDs
  - cell array of character vectors | string vector
Gene names or IDs, specified as a cell array of character vectors or string vector. The array has the same number of rows as `Data`. Each row contains the name or ID of the gene in the data set.

**Note** If `Data` is a `DataMatrix` object with specified row names, you do not need to provide the second input `geneNames` to return the third output `FNames`.

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'AbsValue',10.5` specifies `genelowvvalfilter` to remove expression profiles with absolute values less than 10.5.

**Percentile — Percentile value**

10 (default) | scalar value in the range (0,100)

Percentile value, specified as a scalar value in the range (0 to 100). The function `genelowvvalfilter` removes gene expression profiles with absolute values less than the percentile value, which is specified using `'Percentile'`.

Example: `'Percentile',50`

**AbsValue — Absolute expression profile value**

real number

Absolute expression profile value, specified as a real number. The function `genelowvvalfilter` removes gene expression profiles with absolute values less than the absolute value, which is specified using `'AbsValue'`.

Example: `'AbsValue',10.5`

**AnyVal — Logical indicator to select minimum or maximum absolute value**

false (default) | true

Logical indicator to select the minimum or maximum absolute value, specified as `true` or `false`. Set the value to `true` to select the minimum absolute value. Set it to `false` to select the maximum absolute value.

Example: `'AnyVal',true`

**Output Arguments**

**Mask — Logical vector**

vector of 0s and 1s

Logical vector, returned as a vector of 0s and 1s for each row in `Data`. The elements of `Mask` with value 1 correspond to rows with absolute expression levels exceeding the threshold, and those with value 0 correspond to rows with absolute expression levels less than or equal to the threshold.

**FData — Filtered data matrix**

data matrix
Filtered data matrix, returned as a data matrix that contains gene expression profiles with absolute expression levels exceeding the threshold value. You can also create FData using FData = Data(Mask,:).

FNames — Array of filtered gene names
cell array of character vectors | string vector

Array of filtered gene names, returned as a cell array of character vectors or string vector. It contains gene names or IDs corresponding to each row of Data that contains gene expression profiles with absolute expression levels exceeding the threshold value. You can also create FNames using FNames = geneNames(Mask).

References


See Also
exprprofrange | exprprofvar | geneentropyfilter | generangefilter | genevarfilter

Introduced before R2006a
geneont class

Data structure containing Gene Ontology (GO) information

Description

A geneont object is a data structure containing Gene Ontology information. You can explore and traverse Gene Ontology terms using “is_a” and “part_of” relationships.

Construction

geneont Create geneont object and term objects

Methods

getancestors Find terms that are ancestors of specified Gene Ontology (GO) term
getdescendants Find terms that are descendants of specified Gene Ontology (GO) term
getmatrix Convert geneont object into relationship matrix
getrelatives Find terms that are relatives of specified Gene Ontology (GO) term

Properties

date Read-only character vector containing date and time OBO file was last updated
default_namespace Read-only character vector containing namespace to which GO terms are assigned
format_version Read-only character vector containing version of encoding of OBO file
terms Read-only column vector with handles to term objects of geneont object

Instance Hierarchy

A geneont object contains term objects.

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

Indexing

You can use parenthesis () indexing with either GO identifiers (numbers) or by GO terms (term objects) to create a subontology. See “Examples” on page 1-671 below.
Examples

Example 1.22. Indexing into a geneont Object Using the GO Identifier

You can create a subontology by indexing into a geneont object by using the GO identifier.

1. Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.
   ```matlab
   GeneontObj = geneont('LIVE', true)
   ``
   The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.
   ```matlab
   Gene Ontology object with 27769 Terms.
   ```

2. Create a subontology by returning the terms with GO identifiers of GO:000005 through GO:000010.
   ```matlab
   subontology1 = GeneontObj(5:10)
   ``
   Gene Ontology object with 6 Terms.

3. Create a subontology by returning the term with a GO identifier of GO:000100.
   ```matlab
   subontology2 = GeneontObj(100)
   ``
   Gene Ontology object with 1 Terms.

Example 1.23. Indexing into a geneont Object Using the GO Term

You can create a subontology by indexing into a geneont object by using the GO term.

1. Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.
   ```matlab
   GeneontObj = geneont('LIVE', true)
   ``
   The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.
   ```matlab
   Gene Ontology object with 27769 Terms.
   ```

2. Create an array of term objects containing the fifth through tenth terms of the geneont object.
   ```matlab
   termObject = GeneontObj.terms(5:10)
   ``
   6x1 struct array with fields:
   - id
   - name
   - ontology
   - definition
   - comment
   - synonym
   - is_a
   - part_of
   - obsolete

Note: The GO term of 5 is the position of the term object in the geneont object, and is not necessarily the same as the term object with a GO identifier of GO:000005 used in the first
example. This is because there are many terms that are obsolete and are not included as term objects in the geneont object.

3 Create a subontology by returning the fifth through tenth terms of the geneont object.

    subontology3 = GeneontObj(termObject)

    Gene Ontology object with 6 Terms.

**See Also**

  goannotread | num2goid | term
geneont

Class: geneont

Create geneont object and term objects

Syntax

\[
\text{GeneontObj} = \text{geneont} \\
\text{GeneontObj} = \text{geneont}('File', \text{FileValue}) \\
\text{GeneontObj} = \text{geneont}('Live', \text{LiveValue}) \\
\text{GeneontObj} = \text{geneont}('Live', \text{LiveValue}, 'ToFile', \text{ToFileValue})
\]

Description

\text{GeneontObj} = \text{geneont} creates \text{GeneontObj}, a geneont object, from the gene_ontology.obo file in the MATLAB current directory. It also creates multiple term objects, one for each term in the geneont object.

\text{GeneontObj} = \text{geneont}('File', \text{FileValue}) creates \text{GeneontObj}, a geneont object, from \text{FileValue}, a character vector specifying the file name of an Open Biomedical Ontology (OBO)-formatted file that is on the MATLAB search path.

\text{GeneontObj} = \text{geneont}('Live', \text{LiveValue}) controls the creation of \text{GeneontObj}, a geneont object, from the current version of the Gene Ontology database.

Choices are true or false (default).

**Note** The full Gene Ontology database may take several minutes to download when you run this function using the 'Live' property.

\text{GeneontObj} = \text{geneont}('Live', \text{LiveValue}, 'ToFile', \text{ToFileValue}), when \text{LiveValue} is true, creates \text{GeneontObj}, a geneont object, from the most recent version of the Gene Ontology database and saves the contents of this file to \text{ToFileValue}, a character vector specifying a file name or a path and file name.

Input Arguments

- **FileValue** Character vector specifying the file name of an OBO-formatted file that is on the MATLAB search path.
- **LiveValue** Controls the creation of the most up-to-date geneont object. Enter true to create \text{GeneontObj}, a geneont object, from the most recent version of the Gene Ontology database. Default is false.
- **ToFileValue** Character vector specifying a file name or path and file name to which to save the contents of the current version of the Gene Ontology database.
Output Arguments

**GeneontObj**
MATLAB object containing gene ontology information.

Examples

Create Gene Ontology Object and Explore GO Terms

Download the current version of the Gene Ontology database from the Web into a geneont object.

```
GeneontObj = geneont('LIVE', true)
```

The number of term objects associated with the geneont object is displayed.

```
Gene Ontology object with 47386 Terms.
```

Display information about the geneont object.

```
get(GeneontObj)
```

```
default_namespace: 'gene_ontology'
    format_version: '1.2'
data_version: 'releases/2019-04-17'
    version: ''
    date: ''
saved_by: ''
    auto_generated_by: ''
    subsetdef: {15×1 cell}
    import: ''
synonymtypedef: 'systematic_synonym "Systematic synonym" EXACT'
    idspace: ''
default_relationship_id_prefix: ''
id_mapping: ''
    remark: 'Includes Ontology(OntologyID(OntologyIRI(<http://purl.obolibrary.org/obo/go/never_in_taxon.owl>))) [Axioms: 18 Logical Axioms: 0]
typeref: ''
unrecognized_tag: {'ontology'  'go'}
Terms: [47386×1 geneont.term]
```

Search for all GO terms in the geneont object that contain the character vector ribosome in the field name, and use the geneont.terms property to create a MATLAB structure array of term objects containing those terms.

```
comparison = regexpi(get(GeneontObj.terms,'name'),'ribosome');
indices = find(~cellfun('isempty',comparison));
terms_with_ribosome = GeneontObj.terms(indices)
```

```
33×1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
```
Note Although the terms property is a column vector with handles to term objects, in the MATLAB Command Window, it displays as a structure array, with one structure for each GO term in the geneont object.

See Also
geneont.terms | goannotread | num2goid | term
generangefilter

Remove gene profiles with small profile ranges

Syntax

Mask = generangefilter(Data)
[Mask, FData] = generangefilter(Data)
[Mask, FData, FNames] = generangefilter(Data, Names)

generangefilter(..., 'Percentile', PercentileValue, ...)
generangefilter(..., 'AbsValue', AbsValueValue, ...)
generangefilter(..., 'LogPercentile', LogPercentileValue, ...)
generangefilter(..., 'LogValue', LogValueValue, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>DataMatrix object on page 1-532 or numeric matrix where each row corresponds to the experimental results for one gene. Each column is the results for all genes from one experiment.</td>
</tr>
<tr>
<td>Names</td>
<td>Cell array of character vectors or string vector where each element corresponds to the name of a gene for each row of experimental data. Names has same number of rows as Data with each row containing the name or ID of the gene in the data set.</td>
</tr>
<tr>
<td>PercentileValue</td>
<td>Property to specify a percentile below which gene expression profiles are removed. Enter a value from 0 to 100.</td>
</tr>
<tr>
<td>AbsValueValue</td>
<td>Property to specify an absolute value below which gene expression profiles are removed.</td>
</tr>
<tr>
<td>LogPercentileValue</td>
<td>Property to specify the logarithm of a percentile.</td>
</tr>
<tr>
<td>LogValueValue</td>
<td>Property to specify the logarithm of an absolute value.</td>
</tr>
</tbody>
</table>

Description

Mask = generangefilter(Data) calculates the range for each gene expression profile in Data, a DataMatrix object on page 1-532 or matrix of the experimental data, and then identifies the expression profiles with ranges less than the 10th percentile.

Mask is a logical vector with one element for each row in Data. The elements of Mask corresponding to rows with a range greater than the threshold have a value of 1, and those with a range less than the threshold are 0.

[Mask, FData] = generangefilter(Data) returns FData, a filtered data matrix. You can also create FData using FData = Data(Mask,:).

[Mask, FData, FNames] = generangefilter(Data, Names) returns FNames, a filtered names array, where Names is a cell array of character vectors or string vector of the names of the genes corresponding to each row in Data. You can also create FNames using FNames = Names(Mask).
Note If Data is a DataMatrix object with specified row names, you do not need to provide the second input Names to return the third output FNames.

generangefilter(..., 'PropertyName', PropertyValue, ...) calls generangefilter with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

generangefilter(..., 'Percentile', PercentileValue, ...) removes from the experimental data (Data) gene expression profiles with ranges less than a specified percentile (PercentileValue).

generangefilter(..., 'AbsValue', AbsValueValue, ...) removes from Data gene expression profiles with ranges less than AbsValueValue.

generangefilter(..., 'LogPercentile', LogPercentileValue, ...) filters genes with profile ranges in the lowest percent of the log range (LogPercentileValue).

generangefilter(..., 'LogValue', LogValueValue, ...) filters genes with profile log ranges lower than LogValueValue.

Examples

1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: yeastvalues, a matrix of gene expression data, genes, a cell array of GenBank accession numbers for labeling the rows in yeastvalues, and times, a vector of time values for labeling the columns in yeastvalues

    load yeastdata

2 Remove gene profiles with small profile ranges.

    [mask, fyeastvalues, fgenes] = generangefilter(yeastvalues, genes);

References


See Also
eexprprofrange | eexprprofvar | geneentropyfilter | genelowvalfilter | genevarfilter

Introduced before R2006a
**geneticcode**

Return nucleotide codon to amino acid mapping for genetic code

**Syntax**

```plaintext
Map = geneticcode
Map = geneticcode(GeneticCode)
```

**Input Arguments**

<table>
<thead>
<tr>
<th>GeneticCode</th>
<th>Integer, character vector, or string specifying a genetic code number or code name from the table Genetic Code. Default is 1 or 'Standard'.</th>
</tr>
</thead>
</table>

**Tip** If you use a code name, you can truncate the name to the first two letters of the name.

**Output Arguments**

| Map | Structure containing the mapping of nucleotide codons to amino acids for the standard genetic code. The Map structure contains a field for each nucleotide codon. |

**Description**

`Map = geneticcode` returns a structure containing the mapping of nucleotide codons to amino acids for the standard genetic code. The `Map` structure contains a field for each nucleotide codon.

`Map = geneticcode(GeneticCode)` returns a structure containing the mapping of nucleotide codons to amino acids for the specified genetic code. `GeneticCode` is either:

- An integer, character vector, or string specifying a code number or code name from the table Genetic Code
- The `transl_table` (code) number from the NCBI Web page describing genetic codes:
  

**Tip** If you use a code name, you can truncate the name to the first two letters of the name.
Genetic Code

<table>
<thead>
<tr>
<th>Code Number</th>
<th>Code Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Standard</td>
</tr>
<tr>
<td>2</td>
<td>Vertebrate Mitochondrial</td>
</tr>
<tr>
<td>3</td>
<td>Yeast Mitochondrial</td>
</tr>
<tr>
<td>4</td>
<td>Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma/Spiroplasma</td>
</tr>
<tr>
<td>5</td>
<td>Invertebrate Mitochondrial</td>
</tr>
<tr>
<td>6</td>
<td>Ciliate, Dasycladacean, and Hexamita Nuclear</td>
</tr>
<tr>
<td>9</td>
<td>Echinoderm Mitochondrial</td>
</tr>
<tr>
<td>10</td>
<td>Euplotid Nuclear</td>
</tr>
<tr>
<td>11</td>
<td>Bacterial and Plant Plastid</td>
</tr>
<tr>
<td>12</td>
<td>Alternative Yeast Nuclear</td>
</tr>
<tr>
<td>13</td>
<td>Ascidian Mitochondrial</td>
</tr>
<tr>
<td>14</td>
<td>Flatworm Mitochondrial</td>
</tr>
<tr>
<td>15</td>
<td>Blepharisma Nuclear</td>
</tr>
<tr>
<td>16</td>
<td>Chlorophycean Mitochondrial</td>
</tr>
<tr>
<td>21</td>
<td>Trematode Mitochondrial</td>
</tr>
<tr>
<td>22</td>
<td>Scenedesmus Obliquus Mitochondrial</td>
</tr>
<tr>
<td>23</td>
<td>Thraustochytrium Mitochondrial</td>
</tr>
</tbody>
</table>

Examples

Return the mapping of nucleotide codons to amino acids for the Flatworm Mitochondrial genetic code.

wormmap = geneticcode('Flatworm Mitochondrial');

References

[1] NCBI Web page describing genetic codes:

See Also

aa2nt | aminolookup | baselookup | codonbias | dnds | dndsml | nt2aa | revgeneticcode | seqshoworfs | seqviewer

Introduced before R2006a
genevarfilter

Filter genes with small profile variance

Syntax

\[
\text{Mask} = \text{genevarfilter(Data)} \\
[\text{Mask}, \text{FData}] = \text{genevarfilter(Data)} \\
[\text{Mask}, \text{FData}, \text{FNames}] = \text{genevarfilter(Data, Names)}
\]

genevarfilter(..., 'Percentile', \text{PercentileValue}, ...) \\
genevarfilter(..., 'AbsValue', \text{AbsValueValue}, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>DataMatrix object on page 1-532 or numeric matrix where each row corresponds to a gene. If a matrix, the first column is the names of the genes, and each additional column is the results from an experiment.</td>
</tr>
<tr>
<td>Names</td>
<td>Cell array of character vectors or string vector where each element corresponds to the name of a gene for each row of experimental data. Names has same number of rows as Data with each row containing the name or ID of the gene in the data set.</td>
</tr>
<tr>
<td>PercentileValue</td>
<td>Specifies a percentile below which gene expression profiles are removed. Choices are integers from 0 to 100. Default is 10.</td>
</tr>
<tr>
<td>AbsValueValue</td>
<td>Property to specify an absolute value below which gene expression profiles are removed.</td>
</tr>
</tbody>
</table>

Description

Gene profiling experiments typically include genes that exhibit little variation in their profile and are generally not of interest. These genes are commonly removed from the data.

\[
\text{Mask} = \text{genevarfilter(Data)} \text{ calculates the variance for each gene expression profile in Data and returns Mask, which identifies the gene expression profiles with a variance less than the 10th percentile. Mask is a logical vector with one element for each row in Data. The elements of Mask corresponding to rows with a variance greater than the threshold have a value of 1, and those with a variance less than the threshold are 0.}
\]

\[
[\text{Mask}, \text{FData}] = \text{genevarfilter(Data)} \text{ calculates the variance for each gene expression profile in Data and returns FData, a filtered data matrix, in which the low-variation gene expression profiles are removed. You can also create FData using } \text{FData} = \text{Data(Mask,:),}.\]

\[
[\text{Mask}, \text{FData}, \text{FNames}] = \text{genevarfilter(Data, Names)} \text{ returns FNames, a filtered names array, in which the names associated with low-variation gene expression profiles are removed. Names is a cell array of character vectors or string vector of the names of the genes corresponding to each row in Data. You can also create FNames using } \text{FNames} = \text{Names(Mask)}.\]
**Note** If `Data` is a DataMatrix object with specified row names, you do not need to provide the second input `Names` to return the third output `FNames`.

`genevarfilter(..., 'PropertyName', PropertyValue, ...)` calls `genevarfilter` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`genevarfilter(..., 'Percentile', PercentileValue, ...)` removes from `Data`, the experimental data, the gene expression profiles with a variance less than the percentile specified by `PercentileValue`. Choices are integers from 0 to 100. Default is 10.

`genevarfilter(..., 'AbsValue', AbsValueValue, ...)` removes from `Data`, the experimental data, the gene expression profiles with a variance less than `AbsValueValue`.

**Examples**

1. Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: `yeastvalues`, a matrix of gene expression data, `genes`, a cell array of GenBank accession numbers for labeling the rows in `yeastvalues`, and `times`, a vector of time values for labeling the columns in `yeastvalues`.

   ```matlab```
   load yeastdata
   ```

2. Filter genes with a small profile variance.

   ```matlab```
   [fyeastvalues, fgenes] = genevarfilter(yeastvalues,genes);
   ```

**References**


**See Also**

`exprprofrange` | `exprprofvar` | `geneentropyfilter` | `genelowvalfilter` | `generangefilter`

**Topics**

DataMatrix object on page 1-532

**Introduced before R2006a**
**genpeptread**

Read data from GenPept file

**Syntax**

\[ GenPeptData = \text{genpeptread}(\text{File}) \]

**Arguments**

<table>
<thead>
<tr>
<th><strong>File</strong></th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a GenPept-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB character array that contains the text of a GenPept-formatted file.</td>
</tr>
</tbody>
</table>

**Tip** You can use the `getgenpept` function with the 'ToFile' property to retrieve sequence information from the GenPept database and create an GenPept-formatted file.

| **GenPeptData** | MATLAB structure or array of structures containing fields corresponding to GenPept keywords. |

**Description**

**Note** NCBI has changed the name of their protein search engine from GenPept to Entrez Protein. However, the function names in the Bioinformatics Toolbox software (`getgenpept` and `genpeptread`) are unchanged representing the still-used GenPept report format.

\[ GenPeptData = \text{genpeptread}(\text{File}) \] reads a GenPept-formatted file, *File*, and creates *GenPeptData*, a structure or array of structures, containing fields corresponding to the GenPept keywords. When *File* contains multiple entries, each entry is stored as a separate element in *GenPeptData*. For a list of the GenPept keywords, see https://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html.

**Examples**

Retrieve sequence information for the protein coded by the HEXA gene, store the data in a file, and then read into the MATLAB software.

```matlab
getgenpept('p06865', 'ToFile', 'TaySachs_Protein.txt')
genpeptread('TaySachs_Protein.txt')
```
See Also
fastaread | genbankread | getgenpept | pdbread | seqviewer

Introduced before R2006a
# geoseriesread

Read Gene Expression Omnibus (GEO) Series (GSE) format data

## Syntax

```
GEOData = geoseriesread(File)
```

## Input Arguments

- **File**
  - Either of the following:
    - Character vector or string specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a Gene Expression Omnibus (GEO) Series (GSE) format file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB current folder.
    - Character array or column vector of strings that contains the text of a GEO Series (GSE) format file.

**Tip** You can use the `getgeodata` function with the `'ToFile'` property to retrieve GEO Series (GSE) format data from the GEO database and create a GEO Series (GSE) format file.

## Output Arguments

- **GEOData**
  - MATLAB structure containing the following fields:
    - **Header** — Header text from the GEO Series (GSE) format file, typically containing a description of the data or experiment information.
    - **Data** — DataMatrix object on page 1-532 containing the data from a GEO Series (GSE) format file. The columns and rows of the DataMatrix object correspond to the sample IDs and Ref IDs, respectively, from the GEO Series (GSE) format file.

## Description

```
GEOData = geoseriesread(File)
```
reads a Gene Expression Omnibus (GEO) Series (GSE) format file, and then creates a MATLAB structure, **GEOData**, with the following fields.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Header text from the GEO Series (GSE) format file, typically containing a description of the data or experiment information.</td>
</tr>
<tr>
<td>Data</td>
<td>DataMatrix object on page 1-532 containing the data from a GEO Series (GSE) format file. The columns and rows of the DataMatrix object correspond to the sample IDs and Ref IDs, respectively, from the GEO Series (GSE) format file.</td>
</tr>
</tbody>
</table>
Examples

1 Retrieve Series (GSE) data from the GEO Web site and save it to a file.
   geodata = getgeodata('GSE11287','ToFile','GSE11287.txt');

2 In a subsequent MATLAB session, you can access the Series (GSE) data from your local file, instead of retrieving it from the GEO Web site.
   geodata = geoseriesread('GSE11287.txt')
   geodata =  
       Header: [1x1 struct]  
       Data: [45101x6 bioma.data.DataMatrix]

3 Access the sample IDs using the colnames property of a DataMatrix object.
   sampleIDs = geodata.Data.colnames
   sampleIDs =  
       'GSM284935'  'GSM284936'  'GSM284937'  'GSM284938'  'GSM284939'  'GSM284940'

See Also
affyread | agferead | galread | geosoftread | getgeodata | gprread | ilmnbsread | sptread

Topics
DataMatrix object on page 1-532

Introduced in R2008b
**geosoftread**

Read Gene Expression Omnibus (GEO) SOFT format data

**Syntax**

`GEOSOFTData = geosoftread(File)`

**Input Arguments**

<table>
<thead>
<tr>
<th>File</th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a Gene Expression Omnibus (GEO) SOFT format Sample file (GSM), Data Set file (GDS), or Platform (GPL) file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.</td>
</tr>
<tr>
<td></td>
<td>• Character array or column vector of strings that contains the text of a GEO SOFT format file.</td>
</tr>
</tbody>
</table>

**Tip** You can use the `getgeodata` function with the `'ToFile'` property to retrieve GEO SOFT format data from the GEO database and create a GEO SOFT format file.

**Output Arguments**

| GEOSOFTData       | MATLAB structure containing information from a GEO SOFT format file.                                      |

**Description**

`GEOSOFTData = geosoftread(File)` reads a Gene Expression Omnibus (GEO) SOFT format Sample file (GSM), Data Set file (GDS), or Platform (GPL) file, and then creates a MATLAB structure, `GEOSOFTData`, with the following fields.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Type of file read (SAMPLE, DATASET, or PLATFORM)</td>
</tr>
<tr>
<td>Accession</td>
<td>Accession number for record in GEO database.</td>
</tr>
<tr>
<td>Header</td>
<td>Microarray experiment information.</td>
</tr>
<tr>
<td>ColumnDescriptions</td>
<td>Cell array containing descriptions of columns in the data.</td>
</tr>
<tr>
<td>ColumnNames</td>
<td>Cell array containing names of columns in the data.</td>
</tr>
<tr>
<td>Data</td>
<td>Array containing microarray data.</td>
</tr>
<tr>
<td>Identifier (GDS files only)</td>
<td>Cell array containing probe IDs.</td>
</tr>
<tr>
<td>IDRef (GDS files only)</td>
<td>Cell array containing indices to probes.</td>
</tr>
</tbody>
</table>
Note Currently, the geosoftread function supports Sample (GSM), Data Set (GDS), and Platform (GPL) records.

Examples

Retrieve GSM data from the GEO Web site and save it to a file.

```matlab
geodata = getgeodata('GSM3258','ToFile','GSM3258.txt');
```

Use geosoftread to read a local copy of the GSM file, instead of accessing it from the GEO Web site.

```matlab
geodata = geosoftread('GSM3258.txt')
```

```matlab
geodata =
    Scope: 'SAMPLE'
    Accession: 'GSM3258'
    Header: [1x1 struct]
    ColumnDescriptions: {6x1 cell}
    ColumnNames: {6x1 cell}
    Data: {5355x6 cell}
```

Read the GDS file for photosynthesis in proteobacteria.

```matlab
gdsdata = geosoftread('GDS329.soft')
```

```matlab
gdsdata =
    Scope: 'DATASET'
    Accession: 'GDS329'
    Header: [1x1 struct]
    ColumnDescriptions: {6x1 cell}
    ColumnNames: {6x1 cell}
    Data: [5355x6 double]
```

See Also
galread | geoseriesread | getgeodata | gprread | ilmnsread | sptread

Introduced before R2006a
get (biograph)

Retrieve information about biograph object

Syntax

get(BGobj)
BGStruct = get(BGobj)
PropertyValue = get(BGobj, 'PropertyName')
[PropertyValue1, PropertyValue2, ...] = get(BGobj, 'PropertyName1', 'PropertyName2', ...)

Input Arguments

<table>
<thead>
<tr>
<th>BGobj</th>
<th>Biograph object created with the function biograph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(PropertyName)</td>
<td>Property name for a biograph object.</td>
</tr>
</tbody>
</table>

Output Arguments

<table>
<thead>
<tr>
<th>BGStruct</th>
<th>Scalar structure, in which each field name is a property of a biograph object, and each field contains the value of that property.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PropertyValue</td>
<td>Value of the property specified by PropertyName.</td>
</tr>
</tbody>
</table>

Description

get(BGobj) displays all properties and their current values of BGobj, a biograph object.

BGStruct = get(BGobj) returns all properties of BGobj, a biograph object, to BGStruct, a scalar structure, in which each field name is a property of a biograph object, and each field contains the value of that property.

PropertyValue = get(BGobj, 'PropertyName') returns the value of the specified property of BGobj, a biograph object.

[PropertyValue1, PropertyValue2, ...] = get(BGobj, 'PropertyName1', 'PropertyName2', ...) returns the values of the specified properties of BGobj, a biograph object.
## Properties of a Biograph Object

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Character vector to identify the biograph object. Default is ''.</td>
</tr>
<tr>
<td>Label</td>
<td>Character vector to label the biograph object. Default is ''.</td>
</tr>
<tr>
<td>Description</td>
<td>Character vector that describes the biograph object. Default is ''.</td>
</tr>
<tr>
<td>LayoutType</td>
<td>Character vector that specifies the algorithm for the layout engine. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'hierarchical' (default) — Uses a topological order of the graph to assign levels, and then arranges the nodes from top to bottom, while minimizing crossing edges.</td>
</tr>
<tr>
<td></td>
<td>• 'radial' — Uses a topological order of the graph to assign levels, and then arranges the nodes from inside to outside of the circle, while minimizing crossing edges.</td>
</tr>
<tr>
<td></td>
<td>• 'equilibrium' — Calculates layout by minimizing the energy in a dynamic spring system.</td>
</tr>
<tr>
<td>EdgeType</td>
<td>Character vector that specifies how edges display. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'straight'</td>
</tr>
<tr>
<td></td>
<td>• 'curved' (default)</td>
</tr>
<tr>
<td></td>
<td>• 'segmented'</td>
</tr>
</tbody>
</table>

**Note** Curved or segmented edges occur only when necessary to avoid obstruction by nodes. Biograph objects with LayoutType equal to 'equilibrium' or 'radial' cannot produce curved or segmented edges.

<table>
<thead>
<tr>
<th>Scale</th>
<th>Positive number that post-scales the node coordinates. Default is 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LayoutScale</td>
<td>Positive number that scales the size of the nodes before calling the layout engine. Default is 1.</td>
</tr>
<tr>
<td>EdgeTextColor</td>
<td>Three-element numeric vector of RGB values. Default is [0, 0, 0], which defines black.</td>
</tr>
<tr>
<td>EdgeFontSize</td>
<td>Positive number that sets the size of the edge font in points. Default is 8.</td>
</tr>
<tr>
<td>ShowArrows</td>
<td>Controls the display of arrows with the edges. Choices are 'on' (default) or 'off'.</td>
</tr>
<tr>
<td>ArrowSize</td>
<td>Positive number that sets the size of the arrows in points. Default is 8.</td>
</tr>
<tr>
<td>ShowWeights</td>
<td>Controls the display of text indicating the weight of the edges. Choices are 'on' or 'off' (default).</td>
</tr>
</tbody>
</table>
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ShowTextInNodes</strong></td>
<td>Character vector that specifies the node property used to label nodes when you display a biograph object using the <code>view</code> method. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'Label' — Uses the Label property of the node object (default).</td>
</tr>
<tr>
<td></td>
<td>• 'ID' — Uses the ID property of the node object.</td>
</tr>
<tr>
<td></td>
<td>• 'None'</td>
</tr>
<tr>
<td><strong>NodeAutoSize</strong></td>
<td>Controls precalculating the node size before calling the layout engine. Choices are 'on' (default) or 'off'.</td>
</tr>
<tr>
<td><strong>NodeCallback</strong></td>
<td>User-defined callback for all nodes. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the <code>view</code> function to display the biograph object in the Biograph Viewer, you can double-click a node to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, @(node) inspect(node), which displays the Property Inspector dialog box.</td>
</tr>
<tr>
<td><strong>EdgeCallback</strong></td>
<td>User-defined callback for all edges. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the <code>view</code> function to display the biograph object in the Biograph Viewer, you can right-click and select a callback to activate. Default is the anonymous function, @(edge) inspect(edge), which displays the Property Inspector dialog box.</td>
</tr>
</tbody>
</table>
| **CustomNodeDrawFcn** | Function handle to a customized function to draw nodes. Default is [].

#### Examples

1. Create a biograph object and assign the node IDs.
   ```matlab
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
ids = {'M30931','L07625','K03454','M27323','M15390'};
b = biograph(cm,ids);
```
2. Use the `get` function to display the node IDs.
   ```matlab
get(b.nodes,'ID')
```
   ```matlab
ans =
'M30931'
```
See Also
biograph | set

Topics
biograph object on page 1-185

Introduced in R2008b
get

Retrieve property of object

Syntax

\[ \text{S} = \text{get} \left( \text{object} \right) \]
\[ \text{propertyValues} = \text{get} \left( \text{object}, \text{propertyNames} \right) \]

Description

\[ \text{S} = \text{get} \left( \text{object} \right) \] returns a structure containing a field for each property of a BioRead or BioMap object.

\[ \text{propertyValues} = \text{get} \left( \text{object}, \text{propertyNames} \right) \] returns the values of the properties specified by propertyNames.

Examples

Retrieve Sequencing Data from BioRead

Store read data from a SAM-formatted file in a BioRead object.

\[ \text{br} = \text{BioRead} \left( \text{'}ex1.sam\text{'} \right) \]

\[ \text{br} = \]
\[ \text{BioRead with properties:} \]
\[ \text{Quality: \{1501x1 File indexed property\}} \]
\[ \text{Sequence: \{1501x1 File indexed property\}} \]
\[ \text{Header: \{1501x1 File indexed property\}} \]
\[ \text{NSeqs: 1501} \]
\[ \text{Name: \text{''}} \]

Retrieve the data from the object. data is a structure containing a field for each object property, such as Quality, Sequence, Header, NSeqs, and Name.

\[ \text{data} = \text{get} \left( \text{br} \right) \]

\[ \text{data} = \text{struct with fields:} \]
\[ \text{Quality: \{1501x1 cell\}} \]
\[ \text{Sequence: \{1501x1 cell\}} \]
\[ \text{Header: \{1501x1 cell\}} \]
\[ \text{NSeqs: 1501} \]
\[ \text{Name: \text{''}} \]

Retrieve the header data only.

\[ \text{headers} = \text{get} \left( \text{br}, \text{'}Header\text{''} \right) \]
You can also retrieve a subset of properties.

```matlab
subset = get(br,{'Header','NSeqs'});
```

### Input Arguments

- **object** — Object containing read data
  
  BioRead object | BioMap object

  Object containing the read data, specified as a BioRead or BioMap object.
  
  Example: `bioreadObj`

- **propertyNames** — Names of object properties
  
  character vector | string | string vector | cell array of character vectors

  Names of the object properties, specified as a character vector, string, string vector, or cell array of character vectors.
  
  Example: `{'Quality','Sequence'}`

### Output Arguments

- **S** — Object property data
  
  structure

  Object property data, returned as a structure containing a field for each property of the object.

- **propertyValues** — Property values
  
  cell array

  Property values, returned as a cell array.

### See Also

- BioMap | BioRead

### Topics

- "Manage Sequence Read Data in Objects"

### Introduced in R2010a
**get (DataMatrix)**

Retrieve information about DataMatrix object

**Syntax**

```plaintext
get(DMObj)
DMStruct = get(DMObj)
PropertyValue = get(DMObj, 'PropertyName')
[Property1Value, Property2Value, ...] = get(DMObj, 'Property1Name', 'Property2Name', ...)
```

**Input Arguments**

<table>
<thead>
<tr>
<th>DMObj</th>
<th>DataMatrix object, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>PropertyName</td>
<td>Property name of a DataMatrix object.</td>
</tr>
</tbody>
</table>

**Output Arguments**

<table>
<thead>
<tr>
<th>DMStruct</th>
<th>Scalar structure, in which each field name is a property of a DataMatrix object, and each field contains the value of that property.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PropertyValue</td>
<td>Value of the property specified by PropertyName.</td>
</tr>
</tbody>
</table>

**Description**

get(DMObj) displays all properties and their current values of DMObj, a DataMatrix object.

DMStruct = get(DMObj) returns all properties of DMObj, a DataMatrix object, to DMStruct, a scalar structure, in which each field name is a property of a DataMatrix object, and each field contains the value of that property.

PropertyValue = get(DMObj, 'PropertyName') returns the value of the specified property of DMObj, a DataMatrix object.

[Property1Value, Property2Value, ...] = get(DMObj, 'Property1Name', 'Property2Name', ...) returns the values of the specified properties of DMObj, a DataMatrix object.
Properties of a DataMatrix Object

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Character vector that describes the DataMatrix object. Default is ''.</td>
</tr>
<tr>
<td>RowNames</td>
<td>Empty array or cell array of character vectors that specifies the names for the rows, typically gene names or probe identifiers. The number of elements in the cell array must equal the number of rows in the matrix. Default is an empty array.</td>
</tr>
<tr>
<td>ColNames</td>
<td>Empty array or cell array of character vectors that specifies the names for the columns, typically sample identifiers. The number of elements in the cell array must equal the number of columns in the matrix.</td>
</tr>
<tr>
<td>NRows</td>
<td>Positive number that specifies the number of rows in the matrix.</td>
</tr>
<tr>
<td>NCols</td>
<td>Positive number that specifies the number of columns in the matrix.</td>
</tr>
<tr>
<td>NDims</td>
<td>Positive number that specifies the number of dimensions in the matrix.</td>
</tr>
<tr>
<td>ElementClass</td>
<td>Character vector that specifies the class type, such as single or double.</td>
</tr>
</tbody>
</table>

Examples

1. Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: yeastvalues, a matrix of gene expression data, genes, a cell array of GenBank accession numbers for labeling the rows in yeastvalues, and times, a vector of time values for labeling the columns in yeastvalues.

   ```matlab
   load filteredyeastdata
   ```

2. Import the microarray object package so that the DataMatrix constructor function will be available.

   ```matlab
   import bioma.data.*
   ```

3. Create a DataMatrix object from the gene expression data in the first 30 rows of the yeastvalues matrix. Use the genes column vector and times row vector to specify the row names and column names.

   ```matlab
dmo = DataMatrix(yeastvalues(1:30,:),genes(1:30,:),times);
   ```

4. Use the get method to display the properties of the DataMatrix object, dmo.

   ```matlab
   get(dmo)
   ```

   ```matlab
   Name: ''
   RowNames: {30x1 cell}
   ColNames: {'0' '9.5' '11.5' '13.5' '15.5' '18.5' '20.5'}
   NRows: 30
   NCols: 7
   NDims: 2
   ElementClass: 'double'
   ```

See Also

DataMatrix | set
Topics
DataMatrix object on page 1-532

Introduced in R2008b
**get (phytree)**

Retrieve information about phylogenetic tree object

**Syntax**

\[
[\text{Value1, Value2,}...\] = \text{get(}\text{Tree, 'Property1','Property2',}...\text{)}
\]

\[\text{get(}\text{Tree})\]

\[\text{V } = \text{ get(}\text{Tree})\]

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>Phytree object created with the function phytrees.</td>
</tr>
<tr>
<td>Name</td>
<td>Property name for a phytree object.</td>
</tr>
</tbody>
</table>

**Description**

\[[\text{Value1, Value2,}...\] = \text{get(}\text{Tree, 'Property1','Property2',}...\text{)}\] returns the specified properties from a phytree object (Tree).

Properties for a phytree object are listed in the following table.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumLeaves</td>
<td>Number of leaves</td>
</tr>
<tr>
<td>NumBranches</td>
<td>Number of branches</td>
</tr>
<tr>
<td>NumNodes</td>
<td>Number of nodes (NumLeaves + NumBranches)</td>
</tr>
<tr>
<td>Pointers</td>
<td>Branch to leaf/branch connectivity list</td>
</tr>
<tr>
<td>Distances</td>
<td>Edge length for every leaf/branch</td>
</tr>
<tr>
<td>LeafNames</td>
<td>Names of the leaves</td>
</tr>
<tr>
<td>BranchNames</td>
<td>Names of the branches</td>
</tr>
<tr>
<td>NodeNames</td>
<td>Names of all the nodes</td>
</tr>
</tbody>
</table>

\text{get(}\text{Tree})\ displays all property names and their current values for a phytree object (Tree).

\[V = \text{get(}\text{Tree})\] returns a structure where each field name is the name of a property of a phytree object (Tree) and each field contains the value of that property.

**Examples**

1. Read in a phylogenetic tree from a file.

   \[
   \text{tr } = \text{phytreeread('pf00002.tree')}
   \]

   Phylogenetic tree object with 33 leaves (32 branches)

2. Get the names of the leaves.
protein_names = get(tr,'LeafNames')
protein_names =
    'Q9YHC6_RANRI/126-382'
    'VIPR1_RAT/140-397'
    'VIPR_CARAU/100-359'
    ...

See Also
getbyname | phytree | phytreeread | select

Topics
phytree object on page 1-1274

Introduced before R2006a
getancestors (biograph)

Find ancestors of a node in biograph object

Syntax

Node = getancestors(BiographNode)
Node = getancestors(BiographNode, NumGenerations)

Arguments

<table>
<thead>
<tr>
<th>BiographNode</th>
<th>Node in a biograph object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumGenerations</td>
<td>Number of generations. Enter a positive integer.</td>
</tr>
</tbody>
</table>

Description

Node = getancestors(BiographNode) returns a node (BiographNode) and all of its direct ancestors.

Node = getancestors(BiographNode, NumGenerations) finds the node (BiographNode) and its direct ancestors up to a specified number of generations (NumGenerations). If NumGenerations is 0, the function returns the node itself.

Examples

1  Create a biograph object.
   cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
   bg = biograph(cm)
2  Find one generation of ancestors for node 2.
   ancNodes = getancestors(bg.nodes(2));
   set(ancNodes,'Color',[1 .7 .7]);
   bg.view;
3 Find two generations of ancestors for node 2.

```matlab
ancNodes = getancestors(bg.nodes(2),2);
set(ancNodes,'Color',[.7 1 .7]);
bg.view;
```
See Also
biograph | dolayout | get | getancestors | getdescendants | getedgesbynodeid | getnodesbyid | getrelatives | set | view

Topics
biograph object on page 1-185

Introduced before R2006a
getancestors

Class: geneont

Find terms that are ancestors of specified Gene Ontology (GO) term

Syntax

AncestorIDs = getancestors(GeneontObj, ID)
[AncestorIDs, Counts] = getancestors(GeneontObj, ID)

... = getancestors(..., 'Height', HeightValue, ...)
... = getancestors(..., 'Relationtype', RelationtypeValue, ...)
... = getancestors(..., 'Exclude', ExcludeValue, ...)

Description

AncestorIDs = getancestors(GeneontObj, ID) searches GeneontObj, a geneont object, for GO terms that are ancestors of the GO term(s) specified by ID, which is a GO term identifier or vector of identifiers. It returns AncestorIDs, a vector of GO term identifiers including ID. ID is a nonnegative integer or a vector containing nonnegative integers.

[AncestorIDs, Counts] = getancestors(GeneontObj, ID) also returns the number of times each ancestor is found. Counts is a column vector with the same number of elements as terms in GeneontObj.

Tip The Counts return value is useful when you tally counts in gene enrichment studies. For more information, see “Gene Ontology Enrichment in Microarray Data”.

... = getancestors(..., 'PropertyName', PropertyValue, ...) calls getancestors with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = getancestors(..., 'Height', HeightValue, ...) searches up through a specified number of levels, HeightValue, in the gene ontology. HeightValue is a positive integer. Default is Inf.

... = getancestors(..., 'Relationtype', RelationtypeValue, ...) searches for specified relationship types, RelationtypeValue, in the gene ontology. RelationtypeValue is a character vector. Choices are 'is_a', 'part_of', or 'both' (default).

... = getancestors(..., 'Exclude', ExcludeValue, ...) controls excluding ID, the original queried term(s), from the output AncestorIDs, unless the term was reached while searching the gene ontology. Choices are true or false (default).
Input Arguments

**GeneontObj**
A geneont object, such as created by the geneont constructor function.

**ID**
GO term identifier or vector of identifiers.

**HeightValue**
Positive integer specifying the number of levels to search upward in the gene ontology.

**RelationtypeValue**
Character vector specifying the relationship types to search for in the gene ontology. Choices are:
- 'is_a'
- 'part_of'
- 'both' (default)

**ExcludeValue**
Controls excluding ID, the original queried term(s), from the output. Choices are true or false (default).

Output Arguments

**AncestorIDs**
Vector of GO term identifiers including ID.

**Counts**
Column vector with the same number of elements as terms in GeneontObj, indicating the number of times each ancestor is found.

Examples

1. Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.
   
   GO = geneont('LIVE', true)
   
   The MATLAB software creates a geneont object and displays the number of terms in the database.

   Gene Ontology object with 24316 Terms.

2. Retrieve the ancestors of the Gene Ontology term with an identifier of 46680.
   
   ancestors = getancestors(GO, 46680)
   
   ancestors =
   8150
   9636
   17085
   42221
   46680
   50896

3. Create a subordinate Gene Ontology.
   
   subontology = GO(ancestors)
   
   Gene Ontology object with 6 Terms.

4. Create and display a report of the subordinate Gene Ontology terms, that includes the GO identifier and name.
rpt = get(subontology.terms,{'id','name'})

```
[ 8150] 'biological_process'
[ 9636] 'response to toxin'
[17085]   [1x23 char]
[42221]   [1x29 char]
[46680] 'response to DDT'
[50896]   [1x20 char]
```

5 View relationships of the subordinate Gene Ontology by using the `getmatrix` method to create a connection matrix to pass to the `biograph` function.

```matlab
cm = getmatrix(subontology);
BG = biograph(cm, get(subontology.terms, 'name'));
view(BG)
```

See Also

`goannotread` | `num2goid` | `term`
getblast

Retrieve BLAST report from NCBI website

Syntax

\[
\text{blastdata} = \text{getblast}(\text{RID}) \\
\text{blastdata} = \text{getblast}(\text{RID}, \text{Name}, \text{Value})
\]

Description

\text{blastdata} = \text{getblast}(\text{RID}) \text{ retrieves } \text{RID}, \text{ the Request ID for the NCBI BLAST report, and returns the report data in } \text{blastdata}, \text{ a MATLAB structure. The Request ID, } \text{RID}, \text{ must be recent because NCBI purges reports after 36 hours.}

\text{blastdata} = \text{getblast}(\text{RID}, \text{Name}, \text{Value}) \text{ uses additional options specified by one or more name-value pair arguments.}

Examples

Perform BLAST search

Perform a BLAST search on a protein sequence and save the results to an XML file.

Get a sequence from the Protein Data Bank and create a MATLAB structure.

\[
\text{S} = \text{getpdb('1CIV');}
\]

Use the structure as input for the BLAST search with a significance threshold of \(1e{-10}\). The first output is the request ID, and the second output is the estimated time (in minutes) until the search is completed.

\[
[\text{RID1,ROTE}] = \text{blastncbi(S,'blastp','expect',1e-10);}
\]

Get the search results from the report. You can save the XML-formatted report to a file for an offline access. Use ROTE as the wait time to retrieve the results.

\[
\text{report1} = \text{getblast(RID1,'WaitTime',ROTE,'ToFile','1CIV_report.xml')}
\]

Blast results are not available yet. Please wait ...

\[
\text{report1 =}
\]

struct with fields:

\[
\begin{align*}
\text{RID:} & \quad 'R49TJMCF014' \\
\text{Algorithm:} & \quad 'BLASTP 2.6.1+' \\
\text{Database:} & \quad 'nr' \\
\text{QueryID:} & \quad 'Query_224139' \\
\text{QueryDefinition:} & \quad 'unnamed protein product' \\
\text{Hits:} & \quad [1\times100 \text{ struct}] \\
\text{Parameters:} & \quad [1\times1 \text{ struct}]
\end{align*}
\]
Use `blastread` to read BLAST data from the XML-formatted BLAST report file.

```matlab
blastdata = blastread('1CIV_report.xml')
```

```
blastdata =
    struct with fields:
        RID: ''
        Algorithm: 'BLASTP 2.6.1+'
        Database: 'nr'
        QueryID: 'Query_224139'
        QueryDefinition: 'unnamed protein product'
        Hits: [1x100 struct]
        Parameters: [1x1 struct]
        Statistics: [1x1 struct]
```

Alternatively, run the BLAST search with an NCBI accession number.

```matlab
RID2 = blastncbi('AAA59174','blastp','expect',1e-10)
```

```
RID2 =
    'R49WAPMH014'
```

Get the search results from the report.

```matlab
report2 = getblast(RID2)
```

```
report2 =
    struct with fields:
        RID: 'R49WAPMH014'
        Algorithm: 'BLASTP 2.6.1+'
        Database: 'nr'
        QueryID: 'AAA59174.1'
        QueryDefinition: 'insulin receptor precursor [Homo sapiens]
        Hits: [1x100 struct]
        Parameters: [1x1 struct]
        Statistics: [1x1 struct]
```

**Input Arguments**

RID — Request ID for NCBI BLAST search

character vector | string
Request ID for retrieving results from a specific NCBI BLAST search, specified as a character vector or string.

Example: 'GTF033EZ015'

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1, Value1,...,NameN, ValueN.

Example: 'ToFile','report.xml' saves the results to a file named report.xml.

**ToFile — Name of file to save report data to**

character vector | string

Name of the file to save the report data to, specified as the comma-separated pair consisting of 'ToFile' and a character vector or string. The file is XML-formatted by default.

Example: 'ToFile','Report.xml'

**WaitTime — Time to wait for report**

0 (default) | nonnegative integer

Time (in minutes) to wait for the report from NCBI to be ready, specified as the comma-separated pair consisting of 'WaitTime' and a nonnegative integer. If the report is still not ready after the specified time, an error is generated.

The default value is 0, that is, there is no delay in retrieving the report.

**Tip** Use the RTOE, request time of execution, returned by the blastncbi function as the wait time here.

Example: 'WaitTime',2

**Output Arguments**

**blastdata — BLAST report data**

structure

BLAST report data, returned as a structure that contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID</td>
<td>Request ID for retrieving results from a specific NCBI BLAST search</td>
</tr>
<tr>
<td>Algorithm</td>
<td>NCBI algorithm used to perform the BLAST search</td>
</tr>
<tr>
<td>Database</td>
<td>All databases searched</td>
</tr>
<tr>
<td>QueryID</td>
<td>Identifier of the query sequence</td>
</tr>
<tr>
<td>QueryDefinition</td>
<td>Definition of the query sequence</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Hits</td>
<td>Structure containing information on the hit sequences, such as IDs, accession numbers, lengths, and HSPs (high-scoring segment pairs)</td>
</tr>
<tr>
<td>Parameters</td>
<td>Structure containing information on the input parameters used to perform the search</td>
</tr>
<tr>
<td>Statistics</td>
<td>Summary of statistical details about the performed search, such as lambda, kappa, and entropy values</td>
</tr>
</tbody>
</table>

**More About**

**Hits**

This table lists each field of blastdata.Hits.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>ID of the subject sequence that matched the query sequence</td>
</tr>
<tr>
<td>Definition</td>
<td>Description of the subject sequence</td>
</tr>
<tr>
<td>Accession</td>
<td>Accession of the subject sequence</td>
</tr>
<tr>
<td>Length</td>
<td>Length of the subject sequence</td>
</tr>
<tr>
<td>Hsps</td>
<td>Structure containing Information on the high-scoring segment pairs (HSPs)</td>
</tr>
</tbody>
</table>

**Hits.Hsps**

This table summarizes the fields of Hits.Hsps.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>Pairwise alignment score for a high-scoring segment pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>BitScore</td>
<td>Bit score for a high-scoring segment pair.</td>
</tr>
<tr>
<td>Expect</td>
<td>Expectation value for a high-scoring segment pair.</td>
</tr>
<tr>
<td>Identities</td>
<td>Number of identical or similar residues for a high-scoring segment pair between the query sequence and a subject sequence.</td>
</tr>
<tr>
<td>Positives</td>
<td>Number of identical or similar residues for a high-scoring sequence pair between the query sequence and a subject amino acid sequence. This field applies only to translated nucleotide or amino acid query sequences and databases.</td>
</tr>
<tr>
<td>Gaps</td>
<td>Nonaligned residues for a high-scoring segment pair.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlignmentLength</td>
<td>Length of the alignment for a high-scoring segment pair.</td>
</tr>
<tr>
<td>QueryIndices</td>
<td>Indices of the query sequence residue positions for a high-scoring segment pair.</td>
</tr>
<tr>
<td>SubjectIndices</td>
<td>Indices of the subject sequence residue positions for a high-scoring segment pair.</td>
</tr>
<tr>
<td>Frame</td>
<td>Reading frame of the translated nucleotide sequence for a high-scoring segment pair.</td>
</tr>
<tr>
<td>Alignment</td>
<td>3-by-N character array showing the alignment for a high-scoring sequence pair between the query sequence and a subject sequence. The first row is the query sequence, the second row is the alignment, and the third row is the subject sequence.</td>
</tr>
</tbody>
</table>

### Compatibility Considerations

**'Alignments' option has been removed**  
*Errors starting in R2017b*

The 'Alignments' name-value pair has been removed. The number of hits returned in the output is controlled by the number of hits in the input BLAST report.

**'Descriptions' option has been removed**  
*Errors starting in R2017b*

The 'Descriptions' name-value pair has been removed. The number of hits returned in the output is controlled by the number of hits in the input BLAST report.

**'FileFormat' option has been removed**  
*Errors starting in R2017b*

The 'FileFormat' name-value pair has been removed. The file is XML-formatted automatically.

### See Also

blastformat | blastlocal | blastncbi | blastread | blastreadlocal

### Introduced before R2006a
getBowtie2Command

Translate object properties to Bowtie 2 options

Syntax

S = getBowtie2Command(object)
S = getBowtie2Command(object,'IncludeAll',TF)

Description

S = getBowtie2Command(object) returns a character vector S, representing the Bowtie 2 option syntax that corresponds to the modified object properties. By default, the function translates only the modified properties.

getBowtie2Command requires the Bioinformatics Toolbox Interface for Bowtie Aligner support package. If this support package is not installed, then the function provides a download link.

Note getBowtie2Command is supported on Mac and UNIX platforms only.

S = getBowtie2Command(object,'IncludeAll',TF) specifies whether to translate all the object properties (true) or only the modified properties (false).

Examples

Retrieve Bowtie 2 Options

Create a Bowtie2AlignOptions object.

alignOpt = Bowtie2AlignOptions;

Modify the object properties. For example, specify to trim four residues from the 3' and 5' ends before aligning.

alignOpt.Trim3 = 4;
alignOpt.Trim5 = 4;

Retrieve the equivalent Bowtie 2 option syntax for the modified properties.

getbowtie2command(alignOpt)

ans =

' -3 4 -5 4'

Input Arguments

object — Bowtie 2 options object
Bowtie2AlignOptions object | Bowtie2InspectOptions object | Bowtie2BuildOptions object
Bowtie 2 options object, specified as a Bowtie2AlignOptions, Bowtie2InspectOptions, or Bowtie2BuildOptions object.

Example: alignOpt

**TF — Flag to translate all object properties**
```
false (default) | true
```

Flag to translate all object properties, specified as `true` or `false`. If `true`, the function translates all the object properties. If `false`, the function translates only the modified properties.

Example: `true`

Data Types: `logical`

**Output Arguments**

**S — Bowtie 2 option syntax**

character vector

Bowtie 2 option syntax [1], returned as a character vector. The syntax is prefixed by one or two dashes.

**References**


**See Also**

Bowtie2AlignOptions | Bowtie2BuildOptions | Bowtie2InspectOptions | bowtie2 | bowtie2build | bowtie2inspect

**External Websites**

Bowtie 2 manual

**Introduced in R2018a**
getBowtie2Table

Retrieve table with object properties and equivalent Bowtie 2 options

Syntax

tbl = getBowtie2Table(object)

Description

tbl = getBowtie2Table(object) returns a table with all the object properties and corresponding Bowtie 2 options [1].

getBowtie2Table requires the Bioinformatics Toolbox Interface for Bowtie Aligner support package. If this support package is not installed, then the function provides a download link.

Note getBowtie2Table is supported on Mac and UNIX platforms only.

Examples

Retrieve Corresponding Bowtie 2 Options for All Object Properties

Create a Bowtie2AlignOptions object.

alignOpt = Bowtie2AlignOptions;

Retrieve the equivalent Bowtie 2 options for all the object properties.

getbowtie2table(alignOpt)

ans =

47x2 table

<table>
<thead>
<tr>
<th>PropertyName</th>
<th>Bowtie2OptionName</th>
</tr>
</thead>
<tbody>
<tr>
<td>'AllowDovetail'</td>
<td>'--dovetail'</td>
</tr>
<tr>
<td>'AmbiguousPenalty'</td>
<td>'--np'</td>
</tr>
<tr>
<td>'Encoding_Phred33'</td>
<td>[--phred33]</td>
</tr>
<tr>
<td>'Encoding_Phred64'</td>
<td>'--phred64'</td>
</tr>
<tr>
<td>'Encoding_Solexa'</td>
<td>'--solexa-quals'</td>
</tr>
<tr>
<td>'ExcludeContain'</td>
<td>'--no-contain'</td>
</tr>
<tr>
<td>'ExcludeDiscordant'</td>
<td>'--no-discordant'</td>
</tr>
<tr>
<td>'ExcludeMixed'</td>
<td>'--no-mixed'</td>
</tr>
<tr>
<td>'ExcludeOverlap'</td>
<td>'--no-overlap'</td>
</tr>
<tr>
<td>'ExcludeUnaligned'</td>
<td>'--no-unal'</td>
</tr>
<tr>
<td>'IgnoreQuality'</td>
<td>'--ignore-quals'</td>
</tr>
<tr>
<td>'MatchBonus'</td>
<td>'--ma'</td>
</tr>
<tr>
<td>'MaxAmbiguousFunction'</td>
<td>'--n-ceil'</td>
</tr>
<tr>
<td>'MemoryMappedIndex'</td>
<td>'--mm'</td>
</tr>
</tbody>
</table>
Input Arguments

**object** — Bowtie 2 options object
Bowtie2AlignOptions object | Bowtie2InspectOptions object | Bowtie2BuildOptions object

Bowtie 2 options object, specified as a Bowtie2AlignOptions, Bowtie2InspectOptions, or Bowtie2BuildOptions object.

Example: alignOpt

Output Arguments

**tbl** — Object properties and corresponding Bowtie2 options

Table

Object properties and the corresponding Bowtie2 options [1], returned as a table.

References

See Also
Bowtie2AlignOptions | Bowtie2BuildOptions | Bowtie2InspectOptions | bowtie2 | bowtie2build | bowtie2inspect

External Websites
Bowtie 2 manual

Introduced in R2018a
getbyname (phytree)

Branches and leaves from phytree object

Syntax

\[ S = \text{getbyname}(Tree, \text{Expression}) \]
\[ S = \text{getbyname}(Tree, \text{Key}) \]
\[ S = \text{getbyname}(Tree, \text{Key}, \text{'Exact'}, \text{ExactValue}) \]

Arguments

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tree</strong></td>
<td>phytree object created by phytree function (object constructor) or phytreeread function.</td>
</tr>
<tr>
<td><strong>Expression</strong></td>
<td>Regular expression (MATLAB) or cell array of regular expressions to search for in Tree.</td>
</tr>
<tr>
<td><strong>Key</strong></td>
<td>Character vector or cell array of character vectors to search for in Tree.</td>
</tr>
<tr>
<td><strong>ExactValue</strong></td>
<td>Controls whether the full exact node name must match the character vector(s), ignoring case. Choices are true or false (default). When true, S is a numeric column vector indicating which node names match a query exactly, in full.</td>
</tr>
</tbody>
</table>

Description

\[ S = \text{getbyname}(Tree, \text{Expression}) \] searches the nodes names in Tree, a phytree object, for the regular expression(s) specified by Expression. It returns \( S \), a logical matrix of size \( \text{NumNodes} \)-by-\( M \), where \( M \) is either 1 or the length of Expression. Each row in \( S \) corresponds to a node, and each column corresponds to a query in Expression. The logical matrix \( S \) indicates the node names that match Expression, ignoring case.

\[ S = \text{getbyname}(Tree, \text{Key}) \] searches the nodes names in Tree, a phytree object, for the character vector(s) specified by Key. It returns \( S \), a logical matrix of size \( \text{NumNodes} \)-by-\( M \), where \( M \) is either 1 or the length of Key. Each row in \( S \) corresponds to a node, and each column corresponds to a query in Key. The logical matrix \( S \) indicates the node names that match Key, ignoring case.

\[ S = \text{getbyname}(Tree, \text{Key}, \text{'Exact'}, \text{ExactValue}) \] specifies whether the full exact node name must match the character vector(s), ignoring case. Choices are true or false (default). When true, \( S \) is a numeric column vector indicating which node names match a query exactly, in full.

Examples

1. Read a phylogenetic tree file created from a protein family into a phytree object.
   \[ \text{tr} = \text{phytreeread('pf00002.tree')}; \]
2. Determine all the mouse and human proteins by searching for nodes that include the character vectors 'mouse' and 'human' in their names.
   \[ \text{sel} = \text{getbyname(tr,{'mouse','human'})}; \] 
   \[ \text{view(tr,any(sel,2))}; \]
See Also
get | phytree | phytreeread | prune | select

Topics
phytree object on page 1-1274

Introduced before R2006a
**getcanonical (phytree)**

Calculate canonical form of phylogenetic tree

**Syntax**

```matlab
Pointers = getcanonical(Tree)
[Pointers, Distances, Names] = getcanonical(Tree)
```

**Arguments**

- `Tree`: phytree object created by `phytree` function (object constructor).

**Description**

- `Pointers = getcanonical(Tree)` returns the pointers for the canonical form of a phylogenetic tree (`Tree`). In a canonical tree the leaves are ordered alphabetically and the branches are ordered first by their width and then alphabetically by their first element. A canonical tree is isomorphic to all the trees with the same skeleton independently of the order of their leaves and branches.

- `[Pointers, Distances, Names] = getcanonical(Tree)` returns, in addition to the pointers described above, the reordered distances (`Distances`) and node names (`Names`).

**Examples**

1. Create two phylogenetic trees with the same skeleton but slightly different distances.
   ```matlab
   b = [1 2; 3 4; 5 6; 7 8; 9 10];
   tr_1 = phytree(b,[.1 .2 .3 .3 .4 ]');
   tr_2 = phytree(b,[.2 .1 .2 .3 .4 ]');
   ```

2. Plot the trees.
   ```matlab
   plot(tr_1)
   plot(tr_2)
   ```

3. Check whether the trees have an isomorphic construction.
   ```matlab
   isequal(getcanonical(tr_1),getcanonical(tr_2))
   ans =
   1
   ```

**See Also**

- `getbyname`
- `phytree`
- `phytreeread`
- `select`
- `subtree`

**Topics**

- `phytree object on page 1-1274`

**Introduced before R2006a**
getCommand
Translate object properties to original options syntax

Syntax

S = getCommand(optionsObject)

Description

S = getCommand(optionsObject) returns a string S representing the modified properties in optionsObject translated into the original syntax (prefixed by one or two dashes) [1]. By default, the function translates only the modified properties. However, if the IncludeAll property of the object is set to true, the function translates all available properties, with default values for unspecified properties.

Examples

Retrieve Options in Original Syntax

Create a CufflinksOptions object.

Note getCommand also works on other cufflinks-related options objects, such as CuffDiffOptions, CuffCompareOptions, and CuffQuantOptions. For a complete list of objects, see “optionsObject” on page 1-0.

```matlab
opt = CufflinksOptions;
Modify the object properties. For this example, specify the minimum average coverage for 3' end trimming and change the seed for the Cufflinks random number generator.
opt.TrimCoverageThreshold = 5;
opt.Seed = 1;
Retrieve the options translated into the original syntax.
s = getCommand(opt)
ans =
"--seed -2.3 --trim-3-avgcov-thresh 5"
```

Input Arguments

optionsObject — Options
CufflinksOptions | CuffCompareOptions | CuffMergeOptions | CuffQuantOptions | CuffDiffOptions | CuffNormOptions | CuffGFFReadOptions
Options, specified as a `CufflinksOptions`, `CuffCompareOptions`, `CuffMergeOptions`, `CuffQuantOptions`, `CuffDiffOptions`, `CuffNormOptions`, or `CuffGFFReadOptions` object.

**Output Arguments**

*S* — Options in original syntax

`string`

Options in the original syntax, returned as a string.

**References**


**See Also**

`CuffCompareOptions` | `CuffDiffOptions` | `CuffGFFReadOptions` | `CuffMergeOptions` | `CuffNormOptions` | `CuffQuantOptions` | `CufflinksOptions`

**External Websites**

Cufflinks manual

**Introduced in R2019a**
getData

Class: GFFAnnotation

Create structure containing subset of data from GFFAnnotation

Syntax

AnnotStruct = getData(AnnotObj)
AnnotStruct = getData(AnnotObj,StartPos,EndPos)
AnnotStruct = getData(AnnotObj,Subset)
AnnotStruct = getData( ___ ,Name,Value)

Description

AnnotStruct = getData(AnnotObj) returns AnnotStruct, an array of structures containing data from all elements in AnnotObj. The fields in the return structures are the same as the elements in the FieldNames property of AnnotObj.

AnnotStruct = getData(AnnotObj,StartPos,EndPos) returns AnnotStruct, an array of structures containing data from a subset of the elements in AnnotObj that falls within each reference sequence range specified by StartPos and EndPos.

AnnotStruct = getData(AnnotObj,Subset) returns AnnotStruct, an array of structures containing subset of data from AnnotObj specified by Subset, a vector of integers.

AnnotStruct = getData( ___ ,Name,Value) returns AnnotStruct, an array of structures, using any of the input arguments in the previous syntaxes and additional options specified by one or more Name,Value pair arguments.

Input Arguments

AnnotObj

Object of the GFFAnnotation class.

Default:

StartPos

Nonnegative integer specifying the start of a range in each reference sequence in AnnotObj. The integer StartPos must be less than or equal to EndPos.

Default:

EndPos

Nonnegative integer specifying the end of a range in each reference sequence in AnnotObj. The integer EndPos must be greater than or equal to StartPos.

Default:
**Subset**

Vector of positive integers less than or equal to the number of entries in the object. Use the vector `Subset` to retrieve any element or subset of data from the object.

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name, Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

**Reference**

Character vector, string, string vector, or cell array of character vectors specifying one or more reference sequences in `AnnotObj`. Only annotations whose reference field matches one of the character vectors or strings are included in `AnnotStruct`.

**Feature**

Character vector, string, string vector, or cell array of character vectors specifying one or more features in `AnnotObj`. Only annotations whose feature field matches one of the character vectors or strings are included in `AnnotStruct`.

**Default:**

**Overlap**

Minimum number of base positions that an annotation must overlap in the range, to be included in `AnnotStruct`. This value can be any of the following:

- Positive integer
- 'full' — An annotation must be fully contained in the range to be included.
- 'start' — An annotation’s start position must lie within the range to be included.

**Default:** 1

**Output Arguments**

**AnnotStruct**

Array of structures containing data from elements in `AnnotObj`. The fields in the return structures are the same as the elements in the `FieldNames` property of `AnnotObj`. Specifically, these fields are:

- Reference
- Start
- Stop
- Feature
- Source
- Score
- Strand
- Frame
• Attributes

Examples

Example 1.24. Retrieve Subsets of Data from a GFFAnnotation Object

Construct a GFFAnnotation object using a GFF-formatted file that is provided with Bioinformatics Toolbox.

GFFAnnotObj = GFFAnnotation('tair8_1.gff');

Extract annotations for positions 10,000 through 20,000 from the reference sequence.

AnnotStruct1 = getData(GFFAnnotObj,10000,20000)

AnnotStruct1 =

9x1 struct array with fields:
    Reference
    Start
    Stop
    Feature
    Source
    Score
    Strand
    Frame
    Attributes

Extract the first five annotations from the object.

AnnotStruct2 = getData(GFFAnnotObj,[1:5])

AnnotStruct2 =

5x1 struct array with fields:
    Reference
    Start
    Stop
    Feature
    Source
    Score
    Strand
    Frame
    Attributes

Tips

Using getData creates a structure, which provides better access to the annotation data than an object.

• You can access all field values in a structure.
• You can extract, assign, and delete field values.
• You can use linear indexing to access field values of specific annotations. For example, you can access the start value of only the fifth annotation.
See Also
GFFAnnotation | GTFAnnotation | getData (GFFAnnotation) | getFeatureNames (GFFAnnotation) | getIndex (GFFAnnotation) | getRange (GFFAnnotation) | getReferenceNames (GFFAnnotation) | getSubset (GFFAnnotation)

Topics
“Store and Manage Feature Annotations in Objects”

External Websites
GFF (General Feature Format) specifications
**getData**

**Class:** GTFAnnotation

Create structure containing subset of data from GTFAnnotation object

**Syntax**

AnnotStruct = getData(AnnotObj)
AnnotStruct = getData(AnnotObj,StartPos,EndPos)
AnnotStruct = getData(AnnotObj,Subset)
AnnotStruct = getData(___,Name,Value)

**Description**

AnnotStruct = getData(AnnotObj) returns AnnotStruct, an array of structures containing data from all elements in AnnotObj. The fields in the return structures are the same as the elements in the FieldNames property of AnnotObj.

AnnotStruct = getData(AnnotObj,StartPos,EndPos) returns AnnotStruct, an array of structures containing data from a subset of the elements in AnnotObj that falls within each reference sequence range specified by StartPos and EndPos.

AnnotStruct = getData(AnnotObj,Subset) returns AnnotStruct, an array of structures containing subset of data from AnnotObj specified by Subset, a vector of integers.

AnnotStruct = getData(___,Name,Value) returns AnnotStruct, an array of structures, using any of the input arguments from the previous syntaxes and additional options specified by one or more Name,Value pair arguments.

**Input Arguments**

**AnnotObj**

Object of the GTFAnnotation class.

**Default:**

**StartPos**

Nonnegative integer specifying the start of a range in each reference sequence in AnnotObj. The integer StartPos must be less than or equal to EndPos.

**Default:**

**EndPos**

Nonnegative integer specifying the end of a range in each reference sequence in AnnotObj. The integer EndPos must be greater than or equal to StartPos.

**Default:**
**Subset**

Vector of positive integers equal or less than the number of entries in the object. Use the vector `Subset` to retrieve any element or subset of data from the object.

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Reference**

Character vector, string, string vector, or cell array of character vectors specifying one or more reference sequences in `AnnotObj`. Only annotations whose reference field matches one of the character vectors or strings are included in `AnnotStruct`.

**Default:**

**Feature**

Character vector, string, string vector, or cell array of character vectors specifying one or more features in `AnnotObj`. Only annotations whose feature field matches one of the character vectors or strings are included in `AnnotStruct`.

**Default:**

**Gene**

Character vector, string, string vector, or cell array of character vectors specifying one or more genes in `AnnotObj`. Only annotations whose gene field matches one of the character vectors or strings are included in `AnnotStruct`.

**Transcript**

Character vector, string, string vector, or cell array of character vectors specifying one or more transcripts in `AnnotObj`. Only annotations whose transcript field matches one of the character vectors or strings are included in `AnnotStruct`.

**Overlap**

Minimum number of base positions that an annotation must overlap in the range, to be included in `AnnotStruct`. This value can be any of the following:

- Positive integer
- `'full'` — An annotation must be fully contained in the range to be included.
- `'start'` — An annotation’s start position must lie within the range to be included.

**Default:** 1
Output Arguments

AnnotStruct

Array of structures containing data from elements in AnnotObj. The fields in the return structures are the same as the elements in the FieldNames property of AnnotObj, and specified by GTF2.2: A Gene Annotation Format. Specifically, these fields are:

- Reference
- Start
- Stop
- Feature
- Gene
- Transcript
- Source
- Score
- Strand
- Frame
- Attributes

Examples

Example 1.25. Retrieve Subsets of Data from a GTFAnnotation Object

Construct a GTFAnnotation object using a GTF-formatted file that is provided with Bioinformatics Toolbox.

GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');

Extract the annotation data for positions 668,000 through 680,000 from the reference sequence.

AnnotStruct1 = getData(GTFAnnotObj,668000,680000)

AnnotStruct1 =

18x1 struct array with fields:
  Reference
  Start
  Stop
  Feature
  Gene
  Transcript
  Source
  Score
  Strand
  Frame
  Attributes

Extract the first five annotations from the object.

AnnotStruct2 = getData(GTFAnnotObj,[1:5])
AnnotStruct2 =

5x1 struct array with fields:
   Reference
   Start
   Stop
   Feature
   Gene
   Transcript
   Source
   Score
   Strand
   Frame
   Attributes

**Tips**

Using `getdata` creates a structure, which provides better access to the annotation data than an object.

- You can access all field values in a structure.
- You can not only extract field values, but also assign and delete values.
- You can use linear indexing to access field values of specific annotations. For example, you can access the start value of only the fifth annotation.

**See Also**

`GFFAnnotation`, `GTFAnnotation`, `getData (GTFAnnotation)`, `getFeatureNames (GTFAnnotation)`, `getGeneNames (GTFAnnotation)`, `getGenes (GTFAnnotation)`, `getIndex (GTFAnnotation)`, `getRange (GTFAnnotation)`, `getReferenceNames (GTFAnnotation)`, `getSegments (GTFAnnotation)`, `getSubset (GTFAnnotation)`, `getTranscripts (GTFAnnotation)`

**Topics**

“Store and Manage Feature Annotations in Objects”

**External Websites**

GTF2.2: A Gene Annotation Format
getdescendants (biograph)

Find descendants of a node in biograph object

Syntax

\[\text{Nodes} = \text{getdescendants}(\text{BiographNode})\]
\[\text{Nodes} = \text{getdescendants}(\text{BiographNode}, \text{NumGenerations})\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiographNode</td>
<td>Node in a biograph object.</td>
</tr>
<tr>
<td>NumGenerations</td>
<td>Number of generations. Enter a positive integer.</td>
</tr>
</tbody>
</table>

Description

\[\text{Nodes} = \text{getdescendants}(\text{BiographNode})\] finds a given node \(\text{(BiographNode)}\) all of its direct descendants.

\[\text{Nodes} = \text{getdescendants}(\text{BiographNode}, \text{NumGenerations})\] finds the node \(\text{(BiographNode)}\) and all of its direct descendants up to a specified number of generations \(\text{(NumGenerations)}\). If the \text{NumGenerations} is 0, the function returns the node itself.

Examples

1. Create a biograph object.
   
   \[
   \begin{bmatrix}
   0 & 1 & 1 & 0 & 0; 1 & 0 & 0 & 1 & 1; 1 & 0 & 0 & 0 & 0; 0 & 0 & 0 & 0 & 1; 1 & 0 & 1 & 0 & 0
   \end{bmatrix}; \\
   \text{bg} = \text{biograph}(\text{cm})
   \]

2. Find one generation of descendants for node 4.
   
   \[
   \text{desNodes} = \text{getdescendants}(\text{bg.nodes(4)});
   \text{set(desNodes,'Color',[1 .7 .7]);}
   \text{bg.view();}
   \]
Find two generations of descendants for node 4.

desNodes = getdescendants(bg.nodes(4),2);
set(desNodes,'Color',[.7 1 .7]);
bg.view;
See Also
biograph | dolayout | get | getancestors | getdescendants | getedgesbynodeid | getnodesbyid | getrelatives | set | view

Topics
biograph object on page 1-185

Introduced before R2006a
getdescendants

Class: geneont

Find terms that are descendants of specified Gene Ontology (GO) term

Syntax

DescendantIDs = getdescendants(GeneontObj, ID)
[DescendantIDs, Counts] = getdescendants(GeneontObj, ID)

... = getdescendants(..., 'Depth', DepthValue, ...)
... = getdescendants(..., 'Relationtype', RelationtypeValue, ...)
... = getdescendants(..., 'Exclude', ExcludeValue, ...)

Description

DescendantIDs = getdescendants(GeneontObj, ID) searches GeneontObj, a geneont object, for GO terms that are descendants of the GO term(s) specified by ID, which is a GO term identifier or vector of identifiers. It returns DescendantIDs, a vector of GO term identifiers including ID. ID is a nonnegative integer or a vector containing nonnegative integers.

[DescendantIDs, Counts] = getdescendants(GeneontObj, ID) also returns the number of times each descendant is found. Counts is a column vector with the same number of elements as terms in GeneontObj.

Tip The Counts return value is useful when you tally counts in gene enrichment studies. For more information, see "Gene Ontology Enrichment in Microarray Data".

... = getdescendants(..., 'PropertyName', PropertyValue, ...) calls getdescendants with optional properties that use property name/property value pairs. You can specify one or more properties in any order. EachPropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = getdescendants(..., 'Depth', DepthValue, ...) searches down through a specified number of levels, DepthValue, in the gene ontology. DepthValue is a positive integer. Default is Inf.

... = getdescendants(..., 'Relationtype', RelationtypeValue, ...) searches for specified relationship types, RelationtypeValue, in the gene ontology. RelationtypeValue is a character vector. Choices are 'is_a', 'part_of', or 'both' (default).

... = getdescendants(..., 'Exclude', ExcludeValue, ...) controls excluding ID, the original queried term(s), from the output DescendantIDs, unless the term was found while searching the gene ontology. Choices are true or false (default).
**Input Arguments**

- **GeneontObj**
  A geneont object, such as created by the `geneont` constructor function.
- **ID**
  GO term identifier or vector of identifiers.
- **DepthValue**
  Positive integer specifying the number of levels to search downward in the gene ontology.
- **RelationtypeValue**
  Character vector specifying the relationship types to search for in the gene ontology. Choices are:
  - 'is_a'
  - 'part_of'
  - 'both' (default)
- **ExcludeValue**
  Controls excluding ID, the original queried term(s), from the output. Choices are `true` or `false` (default).

**Output Arguments**

- **DescendantIDs**
  Vector of GO term identifiers including ID.
- **Counts**
  Column vector with the same number of elements as terms in `GeneontObj`, indicating the number of times each descendant is found.

**Examples**

1. Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.
   ```matlab
   GO = geneont('LIVE', true)
   ```
   The MATLAB software creates a geneont object and displays the number of terms in the database.
   ```matlab
   Gene Ontology object with 27827 Terms.
   ```
2. Retrieve the descendants of the "aldo-keto reductase activity" GO term with a GO identifier of 4033.
   ```matlab
   descendants = getdescendants(GO,4033)
   ```
   ```matlab
   descendants =
   4032
   4033
   8106
   32018
   32866
   32867
   46568
   50112
   50236
   ```
3. Create a subordinate Gene Ontology.
subontology = GO(descendants)

Gene Ontology object with 9 Terms.

Create and display a report of the subordinate Gene Ontology terms, that includes the GO identifier and name.

```matlab
rpt = [num2goid(cell2mat(get(subontology.terms,'id'))) get(subontology.terms,'name')];
disp(sprintf('%s --> %s 
',rpt{:}))
```

```
GO:0004032 --> aldehyde reductase activity
GO:0004033 --> aldo-keto reductase activity
GO:0008106 --> alcohol dehydrogenase (NADP+) activity
GO:0032018 --> 2-methylbutanal reductase activity
GO:0032866 --> xylose reductase activity
GO:0032867 --> arabinose reductase activity
GO:0046568 --> 3-methylbutanal reductase activity
GO:0050112 --> inositol 2-dehydrogenase activity
GO:0050236 --> pyridoxine 4-dehydrogenase activity
```

View relationships of the subordinate Gene Ontology by using the `getmatrix` method to create a connection matrix to pass to the `biograph` function.

```matlab
cm = getmatrix(subontology);
BG = biograph(cm,rpt(1,:));
view(BG)
```

See Also
goannotread | num2goid | term
getDictionary

Class: BioIndexedFile

Retrieve reference sequence names from SAM-formatted source file associated with BioIndexedFile object.

Syntax

Dict = getDictionary(BioIFobj)

Description

Dict = getDictionary(BioIFobj) returns Dict, a cell array of unique character vectors specifying the names of the reference sequences in the SAM-formatted source file associated with BioIFobj, a BioIndexedFile object.

Input Arguments

BioIFobj

Object of the BioIndexedFile class.

Default:

Output Arguments

Dict

Cell array of unique character vectors specifying the reference sequence names in the SAM-formatted source file associated with BioIFobj, a BioIndexedFile object.

See Also

BioIndexedFile | BioMap | getSubset

Topics

“Work with Next-Generation Sequencing Data”
“Manage Sequence Read Data in Objects”
getedgesbynodeid (biograph)

Get handles to edges in biograph object

Syntax

Edges = getedgesbynodeid(BGobj,SourceIDs,SinkIDs)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGobj</td>
<td>Biograph object.</td>
</tr>
<tr>
<td>SourceIDs, SinkIDs</td>
<td>Enter a character vector, cell array of character vectors, or an empty cell array (gets all edges).</td>
</tr>
</tbody>
</table>

Description

Edges = getedgesbynodeid(BGobj,SourceIDs,SinkIDs) gets the handles to the edges that connect the specified source nodes (SourceIDs) to the specified sink nodes (SinkIDs) in a biograph object.

Examples

1. Create a biograph object for the Hominidae family.

   species = {'Homo','Pan','Gorilla','Pongo','Baboon','Macaca','Gibbon'};
   cm = magic(7)>25 & 1-eye(7);
   bg = biograph(cm, species);

2. Find all the edges that connect to the Homo node.

   EdgesIn = getedgesbynodeid(bg,[],'Homo');
   EdgesOut = getedgesbynodeid(bg,'Homo',[]);
   set(EdgesIn,'LineColor',[0 1 0]);
   set(EdgesOut,'LineColor',[1 0 0]);
   bg.view;

3. Find all edges that connect members of the Cercopithecidae family to members of the Hominidae family.

   Cercopithecidae = {'Macaca','Baboon'};
   Hominidae = {'Homo','Pan','Gorilla','Pongo'};
   edgesSel = getedgesbynodeid(bg,Cercopithecidae,Hominidae);
   set(bg.edges,'LineColor',[.5 .5 .5]);
   set(edgesSel,'LineColor',[0 0 1]);
   bg.view;

See Also

biograph | dolayout | get | getancestors | getdescendants | getedgesbynodeid | getnodesbyid | getrelatives | set | view
Topics
biograph object

Introduced before R2006a
getembl

Retrieve sequence information from EMBL database

Syntax

```matlab
EMBLData = getembl(AccessionNumber)

EMBLData = getembl(..., 'ToFile', ToFileValue, ...)
EMBLSeq = getembl(..., 'SequenceOnly', SequenceOnlyValue, ...)
```

Input Arguments

<table>
<thead>
<tr>
<th>AccessionNumber</th>
<th>Unique identifier for a sequence record. Enter a unique combination of letters and numbers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToFileValue</td>
<td>Character vector specifying a file name or a path and file name to which to save the data. If you specify only a file name, the file is stored in the current folder.</td>
</tr>
<tr>
<td>SequenceOnlyValue</td>
<td>Controls the retrieving of only the sequence without the metadata. Choices are true or false (default).</td>
</tr>
</tbody>
</table>

Output Arguments

<table>
<thead>
<tr>
<th>EMBLData</th>
<th>MATLAB structure with fields corresponding to EMBL data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMBLSeq</td>
<td>MATLAB character vector representing the sequence.</td>
</tr>
</tbody>
</table>

Description

getembl retrieves information from the European Molecular Biology Laboratory (EMBL) database for nucleotide sequences. This database is maintained by the European Bioinformatics Institute (EBI). For more details about the EMBL database, see

https://www.ebi.ac.uk/ena/about/formats

```matlab
EMBLData = getembl(AccessionNumber)
```
searches for the accession number in the EMBL database (https://www.ebi.ac.uk/) and returns **EMBLData**, a MATLAB structure with fields corresponding to the EMBL two-character line type code. Each line type code is stored as a separate element in the structure.

**EMBLData** contains the following fields.

<table>
<thead>
<tr>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
</tr>
<tr>
<td>Accession</td>
</tr>
<tr>
<td>SequenceVersion</td>
</tr>
<tr>
<td>DateCreated</td>
</tr>
</tbody>
</table>
EMBLData = getembl(..., 'PropertyName', PropertyValue, ...) calls getembl with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

EMBLData = getembl(..., 'ToFile', ToFileValue, ...) saves the information to an EMBL-formatted file. ToFileValue is a character vector specifying a file name or a path and file name to which to save the data. If you specify only a file name, the file is stored in the current folder.

Tip Read an EMBL-formatted file back into the MATLAB software using the emblread function.

EMBLSeq = getembl(..., 'SequenceOnly', SequenceOnlyValue, ...) controls the retrieving of only the sequence without the metadata. Choices are true or false (default).

Examples

Retrieve data for the rat liver apolipoprotein A-I.
emblout = getembl('X00558')

Retrieve data for the rat liver apolipoprotein A-I and save it to the file rat_protein. If you specify a file name without a path, the file is stored in the current folder.
emblout = getembl('X00558','ToFile','c:\project\rat_protein.txt')

Retrieve only the sequence for the rat liver apolipoprotein A-I.
Seq = getembl('X00558','SequenceOnly',true)

See Also
emblread | getgenbank | getgenpept | getpdb | seqviewer
Introduced before R2006a
**getEntryByIndex**

**Class:** BioIndexedFile

Retrieve entries from source file associated with BioIndexedFile object using numeric index

**Syntax**

\[ Entries = \text{getEntryByIndex}(\text{BioIFobj}, \text{Indices}) \]

**Description**

\( Entries = \text{getEntryByIndex}(\text{BioIFobj}, \text{Indices}) \) extracts entries from the source file associated with \( \text{BioIFobj} \), a BioIndexedFile object. It extracts and concatenates the entries specified by \( \text{Indices} \), a numeric vector of positive integers. It returns \( Entries \), a character vector of concatenated entries. The value of each element in \( \text{Indices} \) must be less than or equal to the number of entries in the source file. A one-to-one relationship exists between the number and order of elements in \( \text{Indices} \) and the output \( Entries \), even if \( \text{Indices} \) has repeated entries.

**Input Arguments**

BioIFobj

Object of the BioIndexedFile class.

**Default:**

Indices

Numeric vector of positive integers. The value of each element must be less than or equal to the number of entries in the source file associated with \( \text{BioIFobj} \), the BioIndexedFile object.

**Default:**

**Output Arguments**

Entries

Character vector of concatenated entries extracted from the source file associated with \( \text{BioIFobj} \), the BioIndexedFile object.

**Examples**

Construct a BioIndexedFile object to access a table containing cross-references between gene names and gene ontology (GO) terms:

\[
\begin{align*}
\% \text{Create variable containing full absolute path of source file}
\text{sourcefile} &= \text{which('yeastgenes.sgd')}; \\
\% \text{Create a BioIndexedFile object from the source file. Indicate}
\text{\% the source file is a tab-delimited file where contiguous rows}
\end{align*}
\]
% with the same key are considered a single entry. Store the
% index file in the Current Folder. Indicate that keys are
% located in column 3 and that header lines are prefaced with !
gene2goObj = BioIndexedFile('mrtab', sourcefile, '.', ...
    'KeyColumn', 3, 'HeaderPrefix', '!')

Return the first, third, and fifth entries from the source file:

% Access 1st, 3rd, and 5th entries
subset_entries = getEntryByIndex(gene2goObj, [1 3 5]);

**Tips**

Use this method to visualize and explore a subset of the entries in the source file for validation purposes.

**See Also**

BioIndexedFile | getEntryByKey | getSubset

**Topics**

“Work with Next-Generation Sequencing Data”
**getEntryByKey**

**Class:** BioIndexedFile

Retrieve entries from source file associated with BioIndexedFile object using alphanumeric key

**Syntax**

\[ Entries = \text{getEntryByKey}(\text{BioIFobj}, \text{Key}) \]

**Description**

\( Entries = \text{getEntryByKey}(\text{BioIFobj}, \text{Key}) \) extracts entries from the source file associated with \( \text{BioIFobj} \), a BioIndexedFile object. It extracts and concatenates the entries specified by \( \text{Key} \), a character vector or cell array of character vectors specifying one or more alphanumeric keys. It returns \( Entries \), a character vector of concatenated entries. If the keys in the source file are not unique, it returns all entries that match a specified key, all at the position of the key in the \( \text{Key} \) cell array. If the keys in the source file are unique, there is a one-to-one relationship between the number and order of elements in \( \text{Key} \) and the output \( Entries \).

**Input Arguments**

- **BioIFobj**
  
  Object of the BioIndexedFile class.
  
  **Default:**

- **Key**
  
  Character vector or cell array of character vectors specifying one or more keys in the source file associated with \( \text{BioIFobj} \), the BioIndexedFile object.
  
  **Default:**

**Output Arguments**

- **Entries**
  
  Character vector of concatenated entries extracted from the source file associated with \( \text{BioIFobj} \), the BioIndexedFile object.

**Examples**

Construct a BioIndexedFile object to access a table containing cross-references between gene names and gene ontology (GO) terms:

\[
\text{% Create variable containing full absolute path of source file}
\text{sourcefile} = \text{which('yeastgenes.sgd');}
\text{% Create a BioIndexedFile object from the source file. Indicate}
\]

1-742
% the source file is a tab-delimited file where contiguous rows
% with the same key are considered a single entry. Store the
% index file in the Current Folder. Indicate that keys are
% located in column 3 and that header lines are prefaced with !
gene2goObj = BioIndexedFile('mrtab', sourcefile, '.', ...
    'KeyColumn', 3, 'HeaderPrefix', '!

Return the entries from the source file that are specified by the keys AAC1 and AAD10:

% Access entries that have the keys AAC1 and AAD10
subset_entries = getEntryByKey(gene2goObj, {'AAC1' 'AAD10'});

Tips

Use this method to visualize and explore a subset of the entries in the source file for validation purposes.

See Also
BioIndexedFile | getEntryByIndex | getKeys | getSubset

Topics
“Work with Next-Generation Sequencing Data”
**getExons**

Class: GTFAnnotation

Return table of exons from GTFAnnotation object

**Syntax**

```matlab
exons = getExons(AnnotObj)
[exons,junctions] = getExons(AnnotObj)
[___] = getExons(AnnotObj,'Reference',R)
[___] = getExons(AnnotObj,'Gene',G)
[___] = getExons(AnnotObj,'Transcript',T)
```

**Description**

`exons = getExons(AnnotObj)` returns `exons`, a table of existing exons in `AnnotObj`.

`[exons,junctions] = getExons(AnnotObj)` also returns `junctions`, a table of spliced junctions for each reference listed in `AnnotObj`.

`[___] = getExons(AnnotObj,'Reference',R)` returns the exons that belong to the reference(s) specified by `R`.

`[___] = getExons(AnnotObj,'Gene',G)` returns the exons that belong to the gene(s) specified by `G`.

`[___] = getExons(AnnotObj,'Transcript',T)` returns the exons that belong to the transcript(s) specified by `T`.

**Input Arguments**

- **AnnotObj** — GTF annotation
  GTFAnnotation object
  GTF annotation, specified as a GTFAnnotation object.

- **R** — Names of reference sequences
  character vector | string | string vector | cell array of character vectors | categorical array
  Names of reference sequences, specified as a character vector, string, string vector, cell array of character vectors, or categorical array.
  The names must come from the `Reference` field of `AnnotObj`. If a name does not exist, the function provides a warning and ignores it.

- **G** — Names of genes
  character vector | string | string vector | cell array of character vectors | categorical array
  Names of genes, specified as a character vector, string, string vector, cell array of character vectors, or categorical array.
The names must come from the Gene field of AnnotObj. If a name does not exist, the function provides a warning and ignores the name.

**T — Names of transcripts**
character vector | string | string vector | cell array of character vectors | categorical array

Names of transcripts, specified as a character vector, string, string vector, cell array of character vectors, or categorical array.

The names must come from the Transcript field of AnnotObj. If a name does not exist, the function gives a warning and ignores the name.

**Output Arguments**

**exons — Exons in AnnotObj**
table

Exons in AnnotObj, returned as a table. The table contains the following variables for each transcript.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transcript</td>
<td>Cell array of character vectors containing transcript IDs, obtained from the Transcript field of AnnotObj.</td>
</tr>
<tr>
<td>GeneName</td>
<td>Cell array of character vectors containing the names of expressed genes, obtained from the Attributes field of AnnotObj. This cell array can contain empty character vectors if the corresponding gene names are not found in Attributes.</td>
</tr>
<tr>
<td>GeneID</td>
<td>Cell array of character vectors containing the expressed gene IDs, obtained from the Gene field of AnnotObj.</td>
</tr>
<tr>
<td>Reference</td>
<td>Categorical array representing the names of reference sequences to which the expressed genes belong. The reference names are from the Reference field of AnnotObj.</td>
</tr>
<tr>
<td>Start</td>
<td>Start location of each exon.</td>
</tr>
<tr>
<td>Stop</td>
<td>Stop location of each exon.</td>
</tr>
<tr>
<td>Strand</td>
<td>Categorical array containing the strand of expressed gene.</td>
</tr>
</tbody>
</table>

**junctions — Spliced junctions for each reference**
table

Spliced junctions for each reference, returned as a table. The table contains the following variables for each junction.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Start location of each junction.</td>
</tr>
<tr>
<td>Stop</td>
<td>Stop location of each junction.</td>
</tr>
<tr>
<td>Reference</td>
<td>Categorical array representing the names of reference sequences to which the junctions belong. The reference names are from the Reference field of AnnotObj.</td>
</tr>
</tbody>
</table>
Examples

Retrieve Exons from a GTF-formatted File

Create a GTFAnnotation object from a GTF-formatted file.

\[
\text{obj} = \text{GTFAnnotation('hum37_2_1M.gtf');}
\]

Get the list of gene names listed in the object.

\[
\text{gNames} = \text{getGeneNames(obj)}
\]

```
gNames = 28x1 cell
    {'uc002qvu.2'}
    {'uc002qvy.2'}
    {'uc002qvw.2'}
    {'uc002qvx.2'}
    {'uc002qvy.2'}
    {'uc002qvz.2'}
    {'uc002qwa.2'}
    {'uc002qwb.2'}
    {'uc002qwc.1'}
    {'uc002qwd.2'}
    {'uc002qwe.3'}
    {'uc002qwf.2'}
    {'uc002qwg.2'}
    {'uc002qwh.2'}
    {'uc002qwi.3'}
    {'uc002qwk.2'}
    {'uc002qwl.2'}
    {'uc002qwm.1'}
    {'uc002qwn.1'}
    {'uc002qwo.1'}
    {'uc002qwp.2'}
    {'uc002qwq.2'}
    {'uc010ewe.2'}
    {'uc010ewf.1'}
    {'uc010ewg.2'}
    {'uc010ewh.1'}
    {'uc010ewi.2'}
    {'uc010yim.1'}
```

Get a table of exons which belong to the first gene uc002qvu.2.

\[
\text{exons} = \text{getExons(obj,'\text{Gene}',gNames{1})}
\]

```
exons=8x7 table
            Transcript    GeneName    GeneID    Reference    Start       Stop      Strand
    ______________    __________    ______________    _________    ______    ______    ______
            {'uc002qvu.2'}    {0x0 char}    {'uc002qvu.2'}      chr2       218138    219001      -
            {'uc002qvu.2'}    {0x0 char}    {'uc002qvu.2'}      chr2       224864    224920      -
            {'uc002qvu.2'}    {0x0 char}    {'uc002qvu.2'}      chr2       229966    230044      -
            {'uc002qvu.2'}    {0x0 char}    {'uc002qvu.2'}      chr2       231023    231191      -
            {'uc002qvu.2'}    {0x0 char}    {'uc002qvu.2'}      chr2       233101    233229      -
            {'uc002qvu.2'}    {0x0 char}    {'uc002qvu.2'}      chr2       234160    234272      -
```

1-746
See Also
GFFAnnotation | GTFAnnotation | getData (GTFAnnotation) | getFeatureNames (GTFAnnotation) | getGeneNames (GTFAnnotation) | getGenes (GTFAnnotation) | getIndex (GTFAnnotation) | getRange (GTFAnnotation) | getReferenceNames (GTFAnnotation) | getSegments (GTFAnnotation) | getSubset (GTFAnnotation) | getTranscripts (GTFAnnotation)

Topics
“Store and Manage Feature Annotations in Objects”

External Websites
GTF2.2: A Gene Annotation Format
getFeatureNames

Class: GFFAnnotation

Retrieve unique feature names from GFFAnnotation object

Syntax

Features = getFeatureNames(AnnotObj)

Description

Features = getFeatureNames(AnnotObj) returns Features, a cell array of character vectors specifying the unique feature names associated with annotations in AnnotObj.

Input Arguments

AnnotObj

Object of the GFFAnnotation class.

Default:

Output Arguments

Features

Cell array of character vectors specifying the unique feature names associated with annotations in AnnotObj.

Examples

Construct a GFFAnnotation object from a GFF-formatted file that is provided with Bioinformatics Toolbox, and then retrieve the feature names from the annotation object:

% Construct a GFFAnnotation object from a GFF file
GFFAnnotObj = GFFAnnotation('tair8_1.gff');
% Retrieve feature names for the annotation object
featureNames = getFeatureNames(GFFAnnotObj)

featureNames =

'CDS'
'exon'
'five_prime_UTR'
'gene'
'mRNA'
'miRNA'
'ncRNA'
'protein'
'pseudogene'


See Also
GFFAnnotation | GTFAnnotation | getData (GFFAnnotation) | getFeatureNames (GFFAnnotation) | getRange (GFFAnnotation) | getReferenceNames (GFFAnnotation) | getSubset (GFFAnnotation)

Topics
“Store and Manage Feature Annotations in Objects”

External Websites
GFF (General Feature Format) specifications
getFeatureNames

Class: GTFAnnotation
Retrieves unique feature names from GTFAnnotation object

Syntax
Features = getFeatureNames(AnnotObj)

Description
Features = getFeatureNames(AnnotObj) returns Features, a cell array of character vectors specifying the unique feature names associated with annotations in AnnotObj.

Input Arguments
AnnotObj
Object of the GTFAnnotation class.

Default:

Output Arguments
Features
Cell array of character vectors specifying the unique feature names associated with annotations in AnnotObj.

Examples
Construct a GTFAnnotation object from a GTF-formatted file that is provided with Bioinformatics Toolbox, and then retrieve the feature names from the annotation object:

% Construct a GTFAnnotation object from a GTF file
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
% Retrieve feature names for the annotation object
featureNames = getFeatureNames(GTFAnnotObj)

featureNames =
  'CDS'
  'exon'
  'start_codon'
  'stop_codon'

See Also
GFFAnnotation|GTFAnnotation|getData (GTFAnnotation)|getFeatureNames (GTFAnnotation)|getGeneNames (GTFAnnotation)|getGenes (GTFAnnotation)|
getIndex (GTFAnnotation) | getRange (GTFAnnotation) | getReferenceNames (GTFAnnotation) | getSegments (GTFAnnotation) | getSubset (GTFAnnotation) | getTranscripts (GTFAnnotation)

**Topics**
“Store and Manage Feature Annotations in Objects”

**External Websites**
GTF2.2: A Gene Annotation Format
getgenbank

Retrieve sequence information from GenBank database

Syntax

Data = getgenbank(AccessionNumber)
gegenbank(AccessionNumber)

Data = getgenbank(..., 'PartialSeq', PartialSeqValue, ...)
Data = getgenbank(..., 'ToFile', ToFileValue, ...)
Data = getgenbank(..., 'FileFormat', FileFormatValue, ...)
Data = getgenbank(..., 'SequenceOnly', SequenceOnlyValue, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessionNumber</td>
<td>Character vector or string specifying a unique alphanumeric identifier for a sequence record.</td>
</tr>
<tr>
<td>PartialSeqValue</td>
<td>Two-element array of integers containing the start and end positions of the subsequence [StartBP, EndBP] that specifies a subsequence to retrieve. StartBP is an integer between 1 and EndBP. EndBP is an integer between StartBP and the length of the sequence.</td>
</tr>
<tr>
<td>ToFileValue</td>
<td>Character vector or string specifying either a file name or a path and file name for saving the GenBank data. If you specify only a file name, the file is saved to the MATLAB Current Folder.</td>
</tr>
<tr>
<td>FileFormatValue</td>
<td>Character vector or string specifying the format for the sequence information. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'GenBank' — Default when SequenceOnlyValue is false.</td>
</tr>
<tr>
<td></td>
<td>• 'FASTA' — Default when SequenceOnlyValue is true.</td>
</tr>
<tr>
<td></td>
<td>When 'FASTA', then Data contains only two fields, Header and Sequence.</td>
</tr>
<tr>
<td>SequenceOnlyValue</td>
<td>Controls the return of only the sequence as a character array. Choices are true or false (default).</td>
</tr>
</tbody>
</table>

Description

getgenbank retrieves nucleotide information from the GenBank database. This database is maintained by the National Center for Biotechnology Information (NCBI). For more details about the GenBank database, see


Data = getgenbank(AccessionNumber) searches for the accession number in the GenBank database and returns Data, a MATLAB structure containing information for the sequence.
**Tip** If an error occurs while retrieving the GenBank-formatted information, try rerunning the query. Errors can occur due to Internet connectivity issues that are unrelated to the GenBank record.

getgenbank(*AccessionNumber*) displays information in the MATLAB Command Window without returning data to a variable. The displayed information is only hyperlinks to the URLs used to search for and retrieve the data.

getgenbank(*..., 'PropertyName', PropertyValue, ...)* calls getgenbank with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

*Data* = getgenbank(*..., 'PartialSeq', PartialSeqValue, ...) returns the specified subsequence in the Sequence field of the MATLAB structure. *PartialSeqValue* is a two-element array of integers containing the start and end positions of the subsequence [*StartBP*, *EndBP*]. *StartBP* is an integer between 1 and *EndBP*. *EndBP* is an integer between *StartBP* and the length of the sequence.

*Data* = getgenbank(*..., 'ToFile', ToFileValue, ...) saves the data returned from the GenBank database to a file. *ToFileValue* is a character vector or string specifying either a file name or a path and file name for saving the GenBank data. If you specify only a file name, the file is saved to the MATLAB Current Folder. The function does not append data to an existing file. Instead, it overwrites the contents of the existing file without warning.

**Tip** You can read a GenBank-formatted file back into MATLAB using the `genbankread` function.

*Data* = getgenbank(*..., 'FileFormat', FileFormatValue, ...) returns the sequence in the specified format. Choices are 'GenBank' or 'FASTA'. When 'FASTA', then *Data* contains only two fields, Header and Sequence. 'GenBank' is the default when *SequenceOnlyValue* is false. 'FASTA' is the default when *SequenceOnlyValue* is true.

*Data* = getgenbank(*..., 'SequenceOnly', SequenceOnlyValue, ...) returns only the sequence in *Data*, a character array. Choices are true or false (default).

**Note** If you use the 'SequenceOnly' and 'ToFile' properties together, the output is always a FASTA-formatted file.

**Examples**

**Example 1.26. Retrieving an RNA Sequence**

To retrieve the sequence from chromosome 19 that codes for the human insulin receptor and store it in a structure, *S*, in the MATLAB Command Window, type:

```matlab
S = getgenbank('M10051')
```

```matlab
S =
```

```matlab
  LocusName: 'HUMINSR'
  LocusSequenceLength: '4723'
  LocusNumberOfStrands: ''
```
Example 1.27. Retrieving a Partial RNA Sequence

By looking at the Features field of the structure returned, you can determine that the coding sequence is positions 139 through 4287. To retrieve only the coding sequence from chromosome 19 that codes for the human insulin receptor and store it in a structure, CDS, in the MATLAB Command Window, type:

\[
\text{CDS} = \text{getgenbank('M10051','PARTIALSEQ', [139,4287]);}
\]

Compatibility Considerations

Appending data to existing file

Behavior changed in R2019a

The function no longer appends data to an existing file. The function now overwrites the contents of the existing file without warning.

See Also

genbankread | getembl | getgenpept | getpdb | seqviewer

Introduced before R2006a
getGenes

Class: GTFAnnotation

Return table of unique genes in GTFAnnotation object

Syntax

genes = getGenes(AnnotObj)
genes = getGenes(AnnotObj,'Reference',R)
genes = getGenes(AnnotObj,'Gene',G)
genes = getGenes(AnnotObj,'Transcript',T)

Description

genes = getGenes(AnnotObj) returns genes, a table of genes referenced by exons in AnnotObj.

genes = getGenes(AnnotObj,'Reference',R) returns the gene(s) that belong to the reference(s) specified by R.

genes = getGenes(AnnotObj,'Gene',G) returns the gene(s) specified by G.

genes = getGenes(AnnotObj,'Transcript',T) returns the gene(s) that contains the transcript(s) specified by T.

Input Arguments

AnnotObj — GTF annotation
GTFAnnotation object

GTF annotation, specified as a GTFAnnotation object.

R — Names of reference sequences
character vector | string | string vector | cell array of character vectors | categorical array

Names of reference sequences, specified as a character vector, string, string vector, cell array of character vectors, or categorical array.

The names must come from the Reference field of AnnotObj. If a name does not exist, the function provides a warning and ignores it.

G — Names of genes
character vector | string | string vector | cell array of character vectors | categorical array

Names of genes, specified as a character vector, string, string vector, cell array of character vectors, or categorical array.

The names must come from the Gene field of AnnotObj. If a name does not exist, the function provides a warning and ignores the name.

T — Names of transcripts
character vector | string | string vector | cell array of character vectors | categorical array
Names of transcripts, specified as a character vector, string, string vector, cell array of character vectors, or categorical array.

The names must come from the Transcript field of AnnotObj. If a name does not exist, the function gives a warning and ignores the name.

**Output Arguments**

genes — Genes referenced by exons in AnnotObj

date

Genes referenced by exons in AnnotObj, returned as a table. The table contains the following variables for each gene.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeneID</td>
<td>Cell array of character vectors containing gene IDs as listed in AnnotObj,</td>
</tr>
<tr>
<td></td>
<td>obtained from the Gene field of AnnotObj.</td>
</tr>
<tr>
<td>GeneName</td>
<td>Cell array of character vectors containing gene names, obtained from the</td>
</tr>
<tr>
<td></td>
<td>Attributes field of AnnotObj. This cell array can contain empty character</td>
</tr>
<tr>
<td></td>
<td>vectors if the corresponding gene names are not found in Attributes.</td>
</tr>
<tr>
<td>Reference</td>
<td>Categorical array representing the names of reference sequences to which the</td>
</tr>
<tr>
<td></td>
<td>genes belong, obtained from the Reference field of AnnotObj.</td>
</tr>
<tr>
<td>Start</td>
<td>Start location of the first exon in each gene.</td>
</tr>
<tr>
<td>Stop</td>
<td>Stop location of the last exon in each gene.</td>
</tr>
<tr>
<td>Strand</td>
<td>Categorical array containing the strand of each gene.</td>
</tr>
<tr>
<td>NumTranscripts</td>
<td>Integer array listing the number of transcripts in each gene.</td>
</tr>
</tbody>
</table>

**Examples**

**Retrieve Genes from a GTF-formatted File**

Create a GTFAnnotation object from a GTF-formatted file.

```matlab
obj = GTFAnnotation('hum37_2_1M.gtf');
```

Retrieve unique reference names. In this case, there is only one reference sequence, which is chromosome 2 (chr2).

```matlab
ref = getReferenceNames(obj)
ref = 1x1 cell array
    {'chr2'}
```

Get a table of all genes which belong to chr2.

```matlab
genes = getGenes(obj,'Reference',ref)
genes=28×7 table
      GeneID    GeneName    Reference    Start    Stop    Strand    NumTranscripts
      ________    ________    ________    ______    ______    ________    ___________
      1          functions  {'chr2'}    1         1         '+'        1
      2          gene.model  {'chr2'}    2         2         '+'        1
      ...        ...        ...        ...      ...      ...        ...
```

1 Functions and Apps
See Also

GFFAnnotation | GTFAnnotation | getData (GTFAnnotation) | getFeatureNames (GTFAnnotation) | getGeneNames (GTFAnnotation) | getGenes (GTFAnnotation) | getIndex (GTFAnnotation) | getRange (GTFAnnotation) | getReferenceNames (GTFAnnotation) | getSegments (GTFAnnotation) | getSubset (GTFAnnotation) | getTranscripts (GTFAnnotation)

Topics

“Store and Manage Feature Annotations in Objects”

External Websites

GTF2.2: A Gene Annotation Format
getGeneNames

Class: GTFAnnotation

Retrieve unique gene names from GTFAnnotation object

Syntax

Genes = getGeneNames(AnnotObj)

Description

Genes = getGeneNames(AnnotObj) returns Genes, a cell array of character vectors specifying the unique gene names associated with annotations in AnnotObj.

Input Arguments

AnnotObj

Object of the GTFAnnotation class.

Default:

Output Arguments

Genes

Cell array of character vectors specifying the unique gene names associated with annotations in AnnotObj.

Examples

Construct a GTFAnnotation object from a GTF-formatted file that is provided with Bioinformatics Toolbox, and then retrieve a list of the unique gene names from the object:

% Construct a GTFAnnotation object from a GTF file
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
% Get gene names from object
geneNames = getGeneNames(GTFAnnotObj)

geneNames =
    'uc002qvu.2'
    'uc002qvv.2'
    'uc002qvw.2'
    'uc002qvx.2'
    'uc002qvy.2'
    'uc002qvz.2'
    'uc002qwa.2'
    'uc002qwb.2'
    'uc002qwc.1'
'uc002qwd.2'
'uc002qwe.3'
'uc002qwf.2'
'uc002qwg.2'
'uc002qwh.2'
'uc002qwi.3'
'uc002qwk.2'
'uc002qwl.2'
'uc002qwm.1'
'uc002qwn.1'
'uc002qwo.1'
'uc002qwp.2'
'uc002qwp.2'
'uc010ewe.2'
'uc010ewf.1'
'uc010ewg.2'
'uc010ewish.1'
'uc010ewi.2'
'uc010yim.1'

**See Also**
GFFAnnotation | GTFAnnotation | getData (GTFAnnotation) | getFeatureNames (GTFAnnotation) | getGeneNames (GTFAnnotation) | getGenes (GTFAnnotation) | getIndex (GTFAnnotation) | getRange (GTFAnnotation) | getReferenceNames (GTFAnnotation) | getSegments (GTFAnnotation) | getSubset (GTFAnnotation) | getTranscripts (GTFAnnotation)

**Topics**
“Store and Manage Feature Annotations in Objects”

**External Websites**
GTF2.2: A Gene Annotation Format
**getgenpept**

Retrieve sequence information from GenPept database

**Syntax**

\[
\text{Data} = \text{getgenpept}(\text{AccessionNumber})
\]

getgenpept(AccessionNumber)

\[
\text{Data} = \text{getgenpept}(..., \text{'PartialSeq'}, \text{PartialSeqValue}, ...)
\]

\[
\text{Data} = \text{getgenpept}(..., \text{'ToFile'}, \text{ToFileValue}, ...)
\]

\[
\text{Data} = \text{getgenpept}(..., \text{'FileFormat'}, \text{FileFormatValue}, ...)
\]

\[
\text{Data} = \text{getgenpept}(..., \text{'SequenceOnly'}, \text{SequenceOnlyValue}, ...)
\]

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AccessionNumber</strong></td>
<td>Character vector specifying a unique alphanumeric identifier for a sequence record.</td>
</tr>
<tr>
<td><strong>PartialSeqValue</strong></td>
<td>Two-element array of integers containing the start and end positions of the subsequence ([\text{StartAA}, \text{EndAA}]) that specifies a subsequence to retrieve. (\text{StartAA}) is an integer between 1 and (\text{EndAA}); (\text{EndAA}) is an integer between (\text{StartAA}) and the length of the sequence.</td>
</tr>
<tr>
<td><strong>ToFileValue</strong></td>
<td>Character vector specifying either a file name or a path and file name for saving the GenPept data. If you specify only a file name, the file is saved to the MATLAB Current Folder.</td>
</tr>
<tr>
<td><strong>FileFormatValue</strong></td>
<td>Character vector specifying the format for the sequence information. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'Genpept' — Default when (\text{SequenceOnlyValue}) is false. (\text{Genpept}) is the still-used GenPept report format.</td>
</tr>
<tr>
<td></td>
<td>• 'FASTA' — Default when (\text{SequenceOnlyValue}) is true. (\text{FASTA}) is a translation of the nucleotide sequences in the GenBank database.</td>
</tr>
<tr>
<td><strong>SequenceOnlyValue</strong></td>
<td>Controls the return of only the sequence as a character array. Choices are true or false (default).</td>
</tr>
</tbody>
</table>

**Description**

getgenpept retrieves a protein (amino acid) sequence information from the GenPept database, which is a translation of the nucleotide sequences in the GenBank database and is maintained by the National Center for Biotechnology Information (NCBI).

**Note** NCBI has changed the name of their protein search engine from GenPept to Entrez Protein. However, the function names in the Bioinformatics Toolbox software (getgenpept and genpeptread) are unchanged representing the still-used GenPept report format. For more information on GenPept data, visit https://www.ncbi.nlm.nih.gov/home/about/policies.shtml.
Data = getgenpept(AccessionNumber) searches for the accession number in the GenPept database and returns Data, a MATLAB structure containing information for the sequence.

**Tip** If an error occurs while retrieving the GenPept-formatted information, try rerunning the query. Errors can occur due to Internet connectivity issues that are unrelated to the GenPept record.

getgenpept(AccessionNumber) displays information in the MATLAB Command Window without returning data to a variable. The displayed information is only hyperlinks to the URLs used to search for and retrieve the data.

getgenpept(...,'PropertyName', PropertyValue, ...) calls getgenpept with optional properties that use property name/property value pairs. You can specify one or more properties in any order. EachPropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Data = getgenpept(...,'PartialSeq', PartialSeqValue, ...) returns the specified subsequence in the Sequence field of the MATLAB structure. PartialSeqValue is a two-element array of integers containing the start and end positions of the subsequence [StartAA, EndAA]. StartAA is an integer between 1 and EndAA; EndAA is an integer between StartAA and the length of the sequence.

Data = getgenpept(...,'ToFile', ToFileValue, ...) saves the data returned from the GenPept database to a file. ToFileValue is a character vector specifying either a file name or a path and file name for saving the GenPept data. If you specify only a file name, the file is saved to the MATLAB Current Folder. The function does not append data to an existing file. Instead, it overwrites the contents of the existing file without warning.

**Tip** You can read a GenPept-formatted file back into MATLAB using the genpeptread function.

Data = getgenpept(...,'FileFormat', FileFormatValue, ...) returns the sequence in the specified format. Choices are 'GenPept' or 'FASTA'. When 'FASTA', then Data contains only two fields, Header and Sequence. 'GenPept' is the default when SequenceOnlyValue is false. 'FASTA' is the default when SequenceOnlyValue is true.

Data = getgenpept(...,'SequenceOnly', SequenceOnlyValue, ...) returns only the sequence in Data, a character array. Choices are true or false (default).

**Note** If you use the 'SequenceOnly' and 'ToFile' properties together, the output is always a FASTA-formatted file.

**Examples**

**Example 1.28. Retrieving a Peptide Sequence**

To retrieve the sequence for the human insulin receptor and store it in a structure, Seq, in the MATLAB Command Window, type:

Seq = getgenpept('AAA59174')
Example 1.29. Retrieving a Partial Peptide Sequence

By looking at the Features field of the structure, you can determine that the furin-like repeats domain is positions 234 through 281. To retrieve only the furin-like repeats domain from the sequence for the human insulin receptor and store it in a structure, Fur, in the MATLAB Command Window, type:

```
Fur = getgenpept('AAA59174','PARTIALSEQ', [234, 281]);
```

Compatibility Considerations

Appending data to existing file

Behavior changed in R2019a

The function no longer appends data to an existing file. The function now overwrites the contents of the existing file without warning.

See Also

`genpeptread | getembl | getgenbank | getpdb`

Introduced before R2006a
getgeodata

Retrieve Gene Expression Omnibus (GEO) format data

Syntax

GEOData = getgeodata(AccessionNumber)

getgeodata(AccessionNumber, 'ToFile', ToFileValue)

Input Arguments

| AccessionNumber | Character vector specifying a unique identifier for a GEO Sample (GSM), Data Set (GDS), Platform (GPL), or Series (GSE) record in the GEO database. Next-Generation Sequencing data cannot be retrieved using this function. |

Tip

• If you are unable to retrieve data for an accession number, increase your Java heap space:
  • If you have MATLAB version 7.10 (R2010a) or later, see “Java Heap Memory Preferences” (MATLAB).
  • If you have MATLAB version 7.9 (R2009b) or earlier, see https://www.mathworks.com/support/solutions/data/1-18I2C.html.
  • Recently submitted data sets may not be available for immediate download. There can be a one- to two-day delay between an experiment being submitted to the GEO database and its availability on the FTP site.

| ToFileValue | Character vector specifying a file name or path and file name for saving the data. If you specify only a file name, that file will be saved in the MATLAB Current Folder. |

Output Arguments

| GEOData | MATLAB structure containing information for a GEO record retrieved from the GEO database. |

Description

GEOData = getgeodata(AccessionNumber) searches the Gene Expression Omnibus database for the specified accession number of a Sample (GSM), Data Set (GDS), Platform (GPL), or Series (GSE) record and returns a MATLAB structure containing the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Type of data retrieved (SAMPLE, DATASET, PLATFORM, or SERIES)</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accession</td>
<td>Accession number for record in GEO database.</td>
</tr>
<tr>
<td>Header</td>
<td>Microarray experiment information.</td>
</tr>
<tr>
<td>ColumnDescriptions</td>
<td>Cell array containing descriptions of columns in the data.</td>
</tr>
<tr>
<td>ColumnNames</td>
<td>Cell array containing names of columns in the data.</td>
</tr>
<tr>
<td>Data</td>
<td>Array containing microarray data.</td>
</tr>
<tr>
<td>Identifier (GDS files only)</td>
<td>Cell array containing probe IDs.</td>
</tr>
<tr>
<td>IDRef (GDS files only)</td>
<td>Cell array containing indices to probes.</td>
</tr>
</tbody>
</table>

**Note** Currently, the `getgeodata` function supports Sample (GSM), Data Set (GDS), Platform (GPL), and Series (GSE) records.

getgeodata(**AccessionNumber**, 'ToFile', **ToFileValue**) saves the data returned from the database to a file.

**Note** You can read a GEO SOFT-formatted file back into the MATLAB software using the `geosoftread` function. You can read a GEO SERIES-formatted file back into the MATLAB software using the `geoseriesread` function.

For more information, see


### Examples

gEOStruct = getgeodata('GSM1768')

### See Also
`geoseriesread` | `geosoftread` | `getgenbank` | `getgenpept`

**Introduced before R2006a**
**getHeader**

Retrieve sequence headers from object

**Syntax**

```matlab
headers = getHeader(object)
subsetHeaders = getHeader(object,subset)
```

**Description**

*headers = getHeader(object)* returns sequence headers (names) from a BioRead or BioMap object.

*subsetHeaders = getHeader(object,subset)* returns the header information *subsetHeaders* for only the object elements specified by *subset*.

**Examples**

**Retrieve Sequence Header Information**

Store read data from a SAM-formatted file in a BioRead object.

```matlab
br = BioRead('ex1.sam')
```

```matlab
br = 
BioRead with properties:

- Quality: [1501x1 File indexed property]
- Sequence: [1501x1 File indexed property]
- Header: [1501x1 File indexed property]
- NSeqs: 1501
- Name: ''
```

Retrieve sequence header information.

```matlab
allHeaders = getHeader(br);
```

Retrieve sequence header information from the first and third elements in the object.

```matlab
subset = getHeader(br,[1 3])
```

```matlab
subset = 2x1 cell
    {'B7_591:4:96:693:509'}
    {'EAS51_64:8:5:734:57'}
```

Use a logical vector to get the same information.

```matlab
subset2 = getHeader(br,[true false true])
```
Access each property of the object by using the dot notation.

```matlab
allHeaders = br.Header;
subset = br.Header([1 3])
```

**Input Arguments**

- `object` — Object containing read data
  BioRead object | BioMap object

  Object containing the read data, specified as a `BioRead` or `BioMap` object.
  Example: `bioreadObj`

- `subset` — Subset of elements in object
  vector of positive integers | logical vector | string vector | cell array of character vectors

  Subset of elements in the object, specified as a vector of positive integers, logical vector, string vector, or cell array of character vectors containing valid sequence headers.
  Example: `[1 3]`

**Output Arguments**

- `headers` — Sequence headers
  cell array of character vectors

  Sequence headers, returned as a cell array of character vectors.

- `subsetHeaders` — Sequence headers for subset of elements
  cell array of character vectors

  Sequence headers for a subset of elements from the object, returned as a cell array of character vectors.
gethmmalignment

Retrieve multiple sequence alignment associated with hidden Markov model (HMM) profile from PFAM database

Syntax

AlignStruct = gethmmalignment(PFAMName)
AlignStruct = gethmmalignment(PFAMAccessNumber)
AlignStruct = gethmmalignment(PFAMNumber)

AlignStruct = gethmmalignment(..., 'ToFile',ToFileValue, ...)
AlignStruct = gethmmalignment(..., 'Type',TypeValue, ...)
AlignStruct = gethmmalignment(..., 'IgnoreGaps', IgnoreGaps, ...)

Input Arguments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFAMName</td>
<td>Character vector specifying a protein family name (unique identifier) of an HMM profile record in the PFAM database. For example, '7tm_2'.</td>
</tr>
<tr>
<td>PFAMAccessNumber</td>
<td>Character vector specifying a protein family accession number of an HMM profile record in the PFAM database. For example, 'PF00002'.</td>
</tr>
<tr>
<td>PFAMNumber</td>
<td>Integer specifying a protein family number of an HMM profile record in the PFAM database. For example, 2 is the protein family number for the protein family PF00002.</td>
</tr>
<tr>
<td>ToFileValue</td>
<td>Character vector specifying a file name or a path and file name for saving the data. If you specify only a file name, that file will be saved in the MATLAB Current Folder.</td>
</tr>
<tr>
<td>TypeValue</td>
<td>Character vector that specifies the set of alignments returned. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'full' — Default. Returns all alignments that fit the HMM profile.</td>
</tr>
<tr>
<td></td>
<td>• 'seed' — Returns only the alignments used to generate the HMM profile.</td>
</tr>
<tr>
<td>IgnoreGapsValue</td>
<td>Controls the removal of the symbols - and . from the sequence. Choices are true or false (default).</td>
</tr>
</tbody>
</table>

Output Arguments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlignStruct</td>
<td>MATLAB structure array containing the multiple sequence alignment associated with an HMM profile.</td>
</tr>
</tbody>
</table>

Description

AlignStruct = gethmmalignment(PFAMName) searches the PFAM database (http://pfam.xfam.org/) for the HMM profile record represented by PFAMName, a protein family name, retrieves the multiple sequence alignment associated with the HMM profile, and returns AlignStruct, a MATLAB structure array, with each structure containing the following fields:
**AlignStruct** = gethmmalignment(*PFAMAccessNumber*) searches the PFAM database for the HMM profile record represented by *PFAMAccessNumber*, a protein family accession number, retrieves the multiple sequence alignment associated with the HMM profile, and returns **AlignStruct**, a MATLAB structure array.

**AlignStruct** = gethmmalignment(*PFAMNumber*) determines a protein family accession number from *PFAMNumber*, an integer, searches the PFAM database for the associated HMM profile record, retrieves the multiple sequence alignment associated with the HMM profile, and returns **AlignStruct**, a MATLAB structure array.

**AlignStruct** = gethmmalignment(*..., 'PropertyName', PropertyValue, ...)*) calls gethmmalignment with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

- **ToFile** = gethmmalignment(*..., 'ToFile', ToFileValue, ...)*) saves the data returned from the PFAM database to a file specified by *ToFileValue*.

**Note** You can read a FASTA-formatted file containing PFAM data back into the MATLAB software using the fastaread function.

**AlignStruct** = gethmmalignment(*..., 'Type', TypeValue, ...)*) specifies the set of alignments returned. Choices are:

- 'full' — Default. Returns all sequences that fit the HMM profile.
- 'seed' — Returns only the sequences used to generate the HMM profile.

**AlignStruct** = gethmmalignment(*..., 'IgnoreGaps', IgnoreGaps, ...)*) controls the removal of the symbols - and . from the sequence. Choices are true or false (default).

**Examples**

To retrieve a multiple alignment of the sequences used to train the HMM profile for global alignment to the 7-transmembrane receptor protein in the secretin family, enter:

```matlab
pfamalign = gethmmalignment('PF00002','Type','seed')
```

```
29×1 struct array with fields:
    Header
    Sequence
```

**See Also**

fastaread | gethmmprof | gethmmtree | multialignread | multialignwrite | pfamhmmread
Introduced before R2006a
gethmmprof

Retrieve hidden Markov model (HMM) profile from PFAM database

Syntax

\[
\text{HMMStruct} = \text{gethmmprof}(\text{PFAMName}) \\
\text{HMMStruct} = \text{gethmmprof}(\text{PFAMNumber}) \\
\text{HMMStruct} = \text{gethmmprof}(..., \text{'ToFile'}, \text{ToFileValue}, ...) \\
\text{HMMStruct} = \text{gethmmprof}(..., \text{'Mode'}, \text{ModeValue}, ...)
\]

Input Arguments

<table>
<thead>
<tr>
<th>PFAMName</th>
<th>Character vector specifying a protein family name (unique identifier) of an HMM profile record in the PFAM database. For example, '7tm_2'.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFAMNumber</td>
<td>Integer specifying a protein family number of an HMM profile record in the PFAM database. For example, 2 is the protein family number for the protein family 'PF00002'.</td>
</tr>
<tr>
<td>ToFileValue</td>
<td>Character vector specifying a file name or a path and file name for saving the data. If you specify only a file name, that file will be saved in the MATLAB Current Folder.</td>
</tr>
</tbody>
</table>
| ModeValue        | Character vector that specifies the returned alignment mode. Choices are:  \[
\begin{align*}
 & \text{'}ls' \quad & \text{— Default. Global alignment mode.} \\
 & \text{'}fs' \quad & \text{— Local alignment mode.}
\end{align*}
\]

Output Arguments

| HMMStruct        | MATLAB structure containing information for an HMM profile retrieved from the PFAM database. |

Description

**Note** gethmmprof retrieves information from PFAM-HMM profiles, from file format version HMMER2.0 to HMMER3/f.

\[
\text{HMMStruct} = \text{gethmmprof}(\text{PFAMName}) \]

searches the PFAM database (http://pfam.xfam.org/) for the record represented by \text{PFAMName} (a protein family name), retrieves the HMM profile information, and stores it in \text{HMMStruct}, a MATLAB structure containing the following fields corresponding to parameters of an HMM profile.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The protein family name (unique identifier) of the HMM profile record in the PFAM database.</td>
</tr>
<tr>
<td>PfamAccessionNumber</td>
<td>The protein family accession number of the HMM profile record in the PFAM database.</td>
</tr>
<tr>
<td>ModelDescription</td>
<td>Description of the HMM profile.</td>
</tr>
<tr>
<td>ModelLength</td>
<td>The length of the profile (number of MATCH states).</td>
</tr>
<tr>
<td>Alphabet</td>
<td>The alphabet used in the model, 'AA' or 'NT'.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> AlphaLength is 20 for 'AA' and 4 for 'NT'.</td>
</tr>
<tr>
<td>MatchEmission</td>
<td>Symbol emission probabilities in the MATCH states.</td>
</tr>
<tr>
<td></td>
<td>The format is a matrix of size ModelLength-by-AlphaLength, where each row corresponds to the emission distribution for a specific MATCH state.</td>
</tr>
<tr>
<td>InsertEmission</td>
<td>Symbol emission probabilities in the INSERT state.</td>
</tr>
<tr>
<td></td>
<td>The format is a matrix of size ModelLength-by-AlphaLength, where each row corresponds to the emission distribution for a specific INSERT state.</td>
</tr>
<tr>
<td>NullEmission</td>
<td>Symbol emission probabilities in the MATCH and INSERT states for the NULL model.</td>
</tr>
<tr>
<td></td>
<td>The format is a 1-by-AlphaLength row vector.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> NULL probabilities are also known as the background probabilities.</td>
</tr>
<tr>
<td>BeginX</td>
<td>BEGIN state transition probabilities.</td>
</tr>
<tr>
<td></td>
<td>Format is a 1-by-(ModelLength + 1) row vector:</td>
</tr>
<tr>
<td></td>
<td>[ B-&gt;D1 \ B-&gt;M1 \ B-&gt;M2 \ B-&gt;M3 \ldots \ B-&gt;Mend ]</td>
</tr>
<tr>
<td>MatchX</td>
<td>MATCH state transition probabilities.</td>
</tr>
<tr>
<td></td>
<td>Format is a 4-by-(ModelLength - 1) matrix:</td>
</tr>
<tr>
<td></td>
<td>[ M1-&gt;M2 \ M2-&gt;M3 \ldots \ M[end-1]-&gt;Mend; \ M1-&gt;I1 \ M2-&gt;I2 \ldots \ M[end-1]-&gt;I[end-1]; \ M1-&gt;D2 \ M2-&gt;D3 \ldots \ M[end-1]-&gt;Dend; \ M1-&gt;E \ M2-&gt;E \ldots \ M[end-1]-&gt;E ]</td>
</tr>
<tr>
<td>InsertX</td>
<td>INSERT state transition probabilities.</td>
</tr>
<tr>
<td></td>
<td>Format is a 2-by-(ModelLength - 1) matrix:</td>
</tr>
<tr>
<td></td>
<td>[ I1-&gt;M2 \ I2-&gt;M3 \ldots \ I[end-1]-&gt;Mend; \ I1-&gt;I1 \ I2-&gt;I2 \ldots \ I[end-1]-&gt;I[end-1] ]</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DeleteX</td>
<td>DELETE state transition probabilities.</td>
</tr>
<tr>
<td></td>
<td>Format is a 2-by-(ModelLength - 1) matrix:</td>
</tr>
<tr>
<td></td>
<td>[ D1→M2 D2→M3 ... D[end-1]→Mend ; D1→D2 D2→D3 ... D[end-1]→Dend ]</td>
</tr>
<tr>
<td>FlankingInsertX</td>
<td>Flanking insert states (N and C) used for LOCAL profile alignment.</td>
</tr>
<tr>
<td></td>
<td>Format is a 2-by-2 matrix:</td>
</tr>
<tr>
<td></td>
<td>[N→B  C→T ; N→N  C→C]</td>
</tr>
<tr>
<td>LoopX</td>
<td>Loop states transition probabilities used for multiple hits alignment.</td>
</tr>
<tr>
<td></td>
<td>Format is a 2-by-2 matrix:</td>
</tr>
<tr>
<td></td>
<td>[E→C  J→B ; E→J  J→J]</td>
</tr>
<tr>
<td>NullX</td>
<td>Null transition probabilities used to provide scores with log-odds values also for state transitions.</td>
</tr>
<tr>
<td></td>
<td>Format is a 2-by-1 column vector:</td>
</tr>
<tr>
<td></td>
<td>[G→F ; G→G]</td>
</tr>
</tbody>
</table>

**HMMStruct** = `gethmmprof(PFAMNumber)` determines a protein family accession number from **PFAMNumber** (an integer), searches the PFAM database for the associated record, retrieves the HMM profile information, and stores it in **HMMStruct**, a MATLAB structure.

**HMMStruct** = `gethmmprof(..., 'PropertyName', PropertyValue, ...)` calls `gethmmprof` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each **PropertyName** must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

**HMMStruct** = `gethmmprof(..., 'ToFile', ToFileValue, ...)` saves the data returned from the PFAM database in a file specified by **ToFileValue**.

**Note** You can read an HMM-formatted file back into the MATLAB software using the `pfamhmmlread` function.

**HMMStruct** = `gethmmprof(..., 'Mode', ModeValue, ...)` specifies the returned alignment mode. Choices are:

- 'ls' (default) — Global alignment mode.
- 'fs' — Local alignment mode.

For more information on HMM profile models, see “HMM Profile Model” on page 1-941.
Examples

To retrieve a hidden Markov model (HMM) profile for the global alignment of the 7-transmembrane receptor protein in the secretin family, enter:

```matlab
hmm = gethmmprof('7tm_2')
```

```matlab
hmm =
```

```
struct with fields:
    Name: '7tm_2'
    PfamAccessionNumber: 'PF00002.21'
    ModelDescription: '7 transmembrane receptor (Secretin family)'
    ModelLength: 241
    Alphabet: 'AA'
    MatchEmission: [241x20 double]
    InsertEmission: [241x20 double]
    NullEmission: [1x20 double]
    BeginX: [242x1 double]
    MatchX: [240x4 double]
    InsertX: [240x2 double]
    DeleteX: [240x2 double]
    FlankingInsertX: [2x2 double]
    LoopX: [2x2 double]
    NullX: [2x1 double]
```

See Also

gethmmalignment | hmmprofalign | hmmprofstruct | pfamhmmread | showhmmprof

Introduced before R2006a
gethmmtree

Retrieve phylogenetic tree data from PFAM database

Syntax

```matlab
Tree = gethmmtree(PFAMName)
Tree = gethmmtree(PFAMAccessionNumber)
Tree = gethmmtree(PFAMNumber)
Tree = gethmmtree(...'ToFile', ToFileValue, ...)
```

Input Arguments

<table>
<thead>
<tr>
<th>PFAMName</th>
<th>Character vector specifying a protein family name (unique identifier) of an HMM profile record in the PFAM database. For example, '7tm_2'.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFAMAccessionNumber</td>
<td>Character vector specifying a protein family accession number of an HMM profile record in the PFAM database. For example, 'PF00002'.</td>
</tr>
<tr>
<td>PFAMNumber</td>
<td>Integer specifying a protein family number of an HMM profile record in the PFAM database. For example, 2 is the protein family number for the protein family PF0002.</td>
</tr>
<tr>
<td>ToFileValue</td>
<td>Property to specify the location and file name for saving data. Enter either a file name or a path and file name supported by your system (ASCII text file).</td>
</tr>
</tbody>
</table>

Output Arguments

| Tree                  | An object containing a phylogenetic tree representative of the protein family. |

Description

`Tree = gethmmtree(PFAMName)` searches the PFAM database for the record represented by `PFAMName`, a protein family name, retrieves information, and returns `Tree`, an object containing a phylogenetic tree representative of the protein family.

`Tree = gethmmtree(PFAMAccessionNumber)` searches the PFAM database for the record represented by `PFAMAccessionNumber`, a protein family accession number, retrieves information, and returns `Tree`, an object containing a phylogenetic tree representative of the protein family.

`Tree = gethmmtree(PFAMNumber)` determines a protein family accession number from `PFAMNumber`, an integer, searches the PFAM database for the associated record, retrieves information, and returns `Tree`, an object containing a phylogenetic tree representative of the protein family.

`Tree = gethmmtree(...'PropertyName', PropertyValue, ...)` calls `gethmmtree` with optional properties that use property name/property value pairs. You can specify one or more
properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

\[ \text{Tree} = \text{gethmmtree}(\ldots \text{'ToFile'}, \text{ToFileValue}, \ldots) \] saves the data returned from the PFAM database in the file ToFileValue.

**Tip** To download the 'seed' tree, use gethmmtree without any extra input arguments. To obtain the 'full' tree, you may use the gethmmalignment function to download the 'full' alignment and build a tree using the seqpdist and seqneighjoin functions as illustrated in the following example.

### Examples

Retrieve phylogenetic tree built from the multiple-aligned sequences used to train the HMM profile model for global alignment. The PFAM accession number PF00002 is for the 7-transmembrane receptor protein in the secretin family.

```matlab
tree = gethmmtree('PF00002');
```

Recover the 'full' tree for the same family by downloading the full multiple sequence alignment and building the tree using the seqpdist and seqneighjoin functions. It may take some considerable amount of time to calculate the tree for large families.

```matlab
seqs = gethmmalignment('PF00002','type','full');
dis = seqpdist(seqs);
tree = seqneighjoin(dis,'equivar',seqs);
```

**See Also**
gethmmalignment | phytreeread

**Introduced before R2006a**
getIndexChanged

Class: GFFAnnotation

Return index array of annotations from GFFAnnotation object

Syntax

Idx = getIndex(AnnotObj)
Idx = getIndex(AnnotObj,StartPos,EndPos)
Idx = getIndex(___,Name,Value)

Description

Idx = getIndex(AnnotObj) returns an index array Idx, an array of integers containing the index of each annotation in AnnotObj.

Idx = getIndex(AnnotObj,StartPos,EndPos) returns an index array Idx for a subset of elements that falls within each reference sequence range specified by StartPos and EndPos.

Idx = getIndex(___,Name,Value) returns an index array Idx, using any of the input arguments from the previous syntaxes and additional options specified by one or more Name,Value pair arguments.

Input Arguments

AnnotObj
Object of the GFFAnnotation class.

Default:

StartPos
Nonnegative integer specifying the start of a range in each reference sequence in AnnotObj. The integer StartPos must be less than or equal to EndPos.

Default:

EndPos
Nonnegative integer specifying the end of a range in each reference sequence in AnnotObj. The integer EndPos must be greater than or equal to StartPos.

Default:

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.
Reference
Character vector or cell array of character vectors specifying one or more reference sequences in AnnotObj. Only indices of annotations whose reference field matches one of the specified references are included in Idx.

Default:

Feature
Character vector or cell array of character vectors specifying one or more features in AnnotObj. Only indices of annotations whose feature field matches one of the specified features are included in Idx.

Default:

Overlap
Minimum number of base positions that an annotation must overlap in the range, to have its index included in Idx. This value can be any of the following:

• Positive integer
• 'full' — An annotation must be fully contained in the range to be included.
• 'start' — An annotation's start position must lie within the range to be included.

Default: 1

Output Arguments

Idx
Array of integers representing indices of elements in AnnotObj.

Examples

Example 1.30. Retrieve Indices of Annotations from a GFFAnnotation Object

Construct a GFFAnnotation object using a GFF-formatted file that is provided with Bioinformatics Toolbox.

GFFAnnotObj = GFFAnnotation('tair8_1.gff');

Extract indices of annotations or features for positions 10,000 through 20,000 from the reference sequence.

Idx = getIndex(GFFAnnotObj,10000,20000)

Idx =

61
62
63
64
65
See Also
GFFAnnotation | GTFAnnotation | getData (GFFAnnotation) | getFeatureNames (GFFAnnotation) | getIndex (GFFAnnotation) | getRange (GFFAnnotation) | getReferenceNames (GFFAnnotation) | getSubset (GFFAnnotation)

Topics
“Store and Manage Feature Annotations in Objects”

External Websites
GFF (General Feature Format) specifications
**getIndex**

*Class:* GTFAnnotation

Return index array of annotations from GTFAnnotation object

**Syntax**

\[
\text{Idx} = \text{getIndex(AnnotObj)}
\]

\[
\text{Idx} = \text{getIndex(AnnotObj,StartPos,EndPos)}
\]

\[
\text{Idx} = \text{getIndex(____,Name,Value)}
\]

**Description**

\[
\text{Idx} = \text{getIndex(AnnotObj)} \text{ returns an index array } \text{Idx}, \text{ an array of integers containing the index of each annotation in AnnotObj.}
\]

\[
\text{Idx} = \text{getIndex(AnnotObj,StartPos,EndPos)} \text{ returns an index array } \text{Idx for a subset of elements that falls within each reference sequence range specified by StartPos and EndPos.}
\]

\[
\text{Idx} = \text{getIndex(____,Name,Value)} \text{ returns an index array } \text{Idx, using any of the input arguments from the previous syntaxes and additional options specified by one or more Name,Value pair arguments.}
\]

**Input Arguments**

**AnnotObj**

Object of the GTFAnnotation class.

*Default:*

**StartPos**

Nonnegative integer specifying the start of a range in each reference sequence in AnnotObj. The integer StartPos must be less than or equal to EndPos.

*Default:*

**EndPos**

Nonnegative integer specifying the end of a range in each reference sequence in AnnotObj. The integer EndPos must be greater than or equal to StartPos.

*Default:*

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.
Reference
Character vector, string, string vector, or cell array of character vectors specifying one or more reference sequences in AnnotObj. Only indices of annotations whose reference field matches one of the specified references are included in Idx.

Default:

Feature
Character vector, string, string vector, or cell array of character vectors specifying one or more features in AnnotObj. Only indices of annotations whose feature field matches one of the specified features are included in Idx.

Default:

Gene
Character vector, string, string vector, or cell array of character vectors specifying one or more genes in AnnotObj. Only annotations whose gene field matches one of the specified genes are included in AnnotStruct.

Transcript
Character vector, string, string vector, or cell array of character vectors specifying one or more transcripts in AnnotObj. Only annotations whose transcript field matches one of the specified transcripts are included in AnnotStruct.

Overlap
Minimum number of base positions that an annotation must overlap in the range, to have its index included in Idx. This value can be any of the following:

• Positive integer
• 'full' — An annotation must be fully contained in the range to be included.
• 'start' — An annotation’s start position must lie within the range to be included.

Default: 1

Output Arguments

Idx
Array of integers representing indices of elements in AnnotObj.

Examples

Example 1.31. Retrieve Indices of Annotations from a GTFAnnotation Object

Construct a GTFAnnotation object using a GTF-formatted file that is provided with Bioinformatics Toolbox.

GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
Extract indices of annotations for positions 210,000 through 220,000 from the reference sequence.

\[
\text{Idx} = \text{getIndex(GTFAnnotObj,210000,220000)}
\]

\[
\text{Idx} =
\begin{align*}
7 \\
15 \\
16 \\
17 \\
36 \\
47 \\
48 \\
49 \\
69 \\
70 \\
71 \\
89 \\
99 \\
111 \\
112 \\
113
\end{align*}
\]

See Also

GFFAnnotation | GTFAnnotation | getData (GTFAnnotation) | getFeatureNames (GTFAnnotation) | getGeneNames (GTFAnnotation) | getGenes (GTFAnnotation) | getIndex (GTFAnnotation) | getRange (GTFAnnotation) | getReferenceNames (GTFAnnotation) | getSegments (GTFAnnotation) | getSubset (GTFAnnotation) | getTranscripts (GTFAnnotation)

Topics

“Store and Manage Feature Annotations in Objects”

External Websites

GTF2.2: A Gene Annotation Format
**getIndexByKey**

**Class:** BioIndexedFile

Retrieve indices from source file associated with BioIndexedFile object using alphanumeric key

**Syntax**

\[
\text{Indices} = \text{getIndexByKey}(\text{BioIFobj, Key})
\]

\[
[\text{Indices, LogicalVals}] = \text{getIndexByKey}(\text{BioIFobj, Key})
\]

**Description**

\[
\text{Indices} = \text{getIndexByKey}(\text{BioIFobj, Key}) \]

returns the indices of entries in the source file associated with \textbf{BioIFobj}, a BioIndexedFile object. It returns the indices of entries that have the keys specified by \textbf{Key}, a character vector or cell array of character vectors specifying one or more alphanumeric keys. It returns \textbf{Indices}, a numeric vector of the indices of entries that have the alphanumeric keys specified by \textbf{Key}. If the keys in the source file are not unique, it returns all indices of entries that match a specified key, all at the position of the key in the \textbf{Key} cell array. If the keys in the source file are unique, there is a one-to-one relationship between the number and order of elements in \textbf{Key} and the output \textbf{Indices}.

\[
[\text{Indices, LogicalVals}] = \text{getIndexByKey}(\text{BioIFobj, Key})
\]

returns a logical vector that indicates only the last match for each key, such that there is a one-to-one relationship between the number and order of elements in \textbf{Key} and \textbf{Indices(LogicalVals)}.

**Input Arguments**

\textbf{BioIFobj}

Object of the BioIndexedFile class.

**Default:**

\textbf{Key}

Character vector or cell array of character vectors specifying one or more keys in the source file associated with \textbf{BioIFobj}, the BioIndexedFile object.

**Default:**

**Output Arguments**

\textbf{Indices}

Numeric vector of the indices of entries in source file that have the alphanumeric keys specified by \textbf{Key}. 

1-782
**LogicalVals**

Logical vector containing the same number of elements as `Indices`. The vector indicates only the last match for each key specified in `Key`, such that there is a one-to-one relationship between the number and order of elements in `Key` and `Indices(LogicalVals)`.

**Tip** Some files contain repeated keys. For example, SAM-formatted files use the same key for entries that are paired end reads. Use the `Indices(LogicalVals)` syntax to return only the last index of a repeated key. For more information, see “Examples” on page 1-783.

**Examples**

Construct a BioIndexedFile object to access a table containing cross-references between gene names and gene ontology (GO) terms:

```matlab
% Create variable containing full absolute path of source file
sourcefile = which('yeastgenes.sgd');
% Create a BioIndexedFile object from the source file. Indicate
% the source file is a tab-delimited file where contiguous rows
% with the same key are considered a single entry. Store the
% index file in the Current Folder. Indicate that keys are
% located in column 3 and that header lines are prefaced with 
% gene2goObj = BioIndexedFile('mrtab', sourcefile, '.', ...
%     'KeyColumn', 3, 'HeaderPrefix','!')
```

Return the indices for the entries in the source file that are specified by the keys AAC1 and AAD10.

```matlab
% Access indices for entries that have the keys AAC1 and AAD10
indices = getIndexByKey(gene2goObj, {'AAC1' 'AAD10'})
indices =
    3
    5
```

Construct a BioIndexedFile object to access a SAM-formatted file that has repeated keys.

```matlab
% Create variable containing full absolute path of source file
samsourcefile = which('ex1.sam');
% Create a BioIndexedFile object from the source file. Store the
% index file in the Current Folder.
% samObj = BioIndexedFile('sam', samsourcefile, '.')
```

Return only the last indices for the entries in the source file that are specified by two keys, 'B7_593:7:15:244:876' and 'EAS56_65:4:296:78:421', both of which are repeated keys.

```matlab
% Return all indices for entries that have two specific keys
[Indices, LogicalVal] = getIndexByKey(samObj, ...
Indices =
    3058
    3238
    3292
    3293
```
LogicalVal =
    0
    1
    0
    1

% Return only the last index for each key
LastIndices = Indices(LogicalVal)

LastIndices =
    3238
    3293

**Tips**

Use this method to determine the indices of specific entries with known keys.

**See Also**

BioIndexedFile | getEntryByKey | getKeys | getSubset

**Topics**

“Work with Next-Generation Sequencing Data”
getKeys

Class: BioIndexedFile

Retrieve alphanumeric keys from source file associated with BioIndexedFile object

Syntax

Keys = getKeys(BioIFobj)

Description

*Keys* = *getKeys*(*BioIFobj*) returns *Keys*, a cell array of character vectors specifying all the keys to the entries in the source file associated with *BioIFobj*, a BioIndexedFile object. The keys appear in the same order as they do in the source file, even if they are not unique.

Input Arguments

*BioIFobj*

Object of the *BioIndexedFile* class.

Default:

Output Arguments

*Keys*

Cell array of character vectors specifying all the keys to the entries in the source file. The keys appear in the same order as they do in the source file, even if they are not unique.

Examples

Construct a BioIndexedFile object to access a table containing cross-references between gene names and gene ontology (GO) terms:

% Create variable containing full absolute path of source file
sourcefile = which('yeastgenes.sgd');
% Create a BioIndexedFile object from the source file. Indicate
% the source file is a tab-delimited file where contiguous rows
% with the same key are considered a single entry. Store the
% index file in the Current Folder. Indicate that keys are
% located in column 3 and that header lines are prefaced with !
gene2goObj = BioIndexedFile('mrtab', sourcefile, '.', ...
  'KeyColumn', 3, 'HeaderPrefix','!')

Retrieve all the keys for the entries in the source file, then view the first 12 keys:

% Retrieve all keys for entries in gene2goObj
keys = getKeys(gene2goObj);
% View the first 12 keys
keys(1:12)

ans =
    '15S_RRNA'
    '21S_RRNA'
    'AAC1'
    'AAC3'
    'AAD10'
    'AAD14'
    'AAD15'
    'AAD16'
    'AAD3'
    'AAD4'
    'AAD6'
    'AAH1'

Tips

Use this method to see a complete list of the alphanumeric keys, in the order they occur in the source file from which the BioIndexedFile object was created.

See Also
BioIndexedFile | getEntryByKey | getIndexByKey | getSubset

Topics
“Work with Next-Generation Sequencing Data”
getmatrix (biograph)

Get connection matrix from biograph object

Syntax

\[
[\text{Matrix, ID, Distances}] = \text{getmatrix}(\text{BGObj})
\]

Arguments

\[
\begin{array}{|l|l|}
\hline
\text{BGObj} & \text{Biograph object created by biograph (object constructor).} \\
\hline
\end{array}
\]

Description

\([\text{Matrix, ID, Distances}] = \text{getmatrix}(\text{BGObj})\) converts the biograph object, \(\text{BiographObj}\), into a logical sparse matrix, \(\text{Matrix}\), in which 1 indicates that a node (row index) is connected to another node (column index). \(\text{ID}\) is a cell array of character vectors listing the ID properties for each node, and corresponds to the rows and columns of \(\text{Matrix}\). \(\text{Distances}\) is a column vector with one entry for every nonzero entry in \(\text{Matrix}\) traversed column-wise and representing the respective \(\text{Weight}\) property for each edge.

Examples

\[
\begin{align*}
\text{cm} &= \begin{bmatrix} 0 & 1 & 0 & 0; & 2 & 0 & 0 & 4; & 4 & 0 & 0 & 0; & 0 & 0 & 0 & 2; & 4 & 0 & 5 & 0 \end{bmatrix}; \\
\text{bg} &= \text{biograph(cm)}; \\
[\text{cm}, \text{IDs}, \text{dist}] &= \text{getmatrix(bg)}
\end{align*}
\]

See Also

\text{biograph | dolayout | getancestors | getdescendants | getedgesbynodeid | getnodesbyid | getrelatives | view}

Topics

\text{biograph object on page 1-185}

Introduced in R2006b
getmatrix

Class: geneont

Convert geneont object into relationship matrix

Syntax

\[
\text{[Matrix, ID, Relationship]} = \text{getmatrix(GeneontObj)}
\]

Description

\[
\text{[Matrix, ID, Relationship]} = \text{getmatrix(GeneontObj)}
\] converts a geneont object, \text{GeneontObj}, into \text{Matrix}, a matrix of relationship values between nodes (row and column indices), in which 0 indicates no relationship, 1 indicates an “is_a” relationship, and 2 indicates a “part_of” relationship. \text{ID} is a column vector listing Gene Ontology IDs that correspond to the rows and columns of \text{Matrix}. \text{Relationship} is a cell array of character vectors defining the types of relationships.

Input Arguments

\text{GeneontObj} \quad \text{A geneont object, such as created by the geneont constructor function.}

Output Arguments

\text{Matrix} \quad \text{Matrix of relationship values between nodes (row and column indices), in which 0 indicates no relationship, 1 indicates an “is_a” relationship, and 2 indicates a “part_of” relationship.}

\text{ID} \quad \text{Column vector listing Gene Ontology IDs that correspond to the rows and columns of Matrix.}

\text{Relationship} \quad \text{Cell array of character vectors defining the types of relationships.}

Examples

1. Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

   \text{GO} = \text{geneont('LIVE',true)}

   The MATLAB software creates a geneont object and displays the number of terms in the database.

   Gene Ontology object with 27595 Terms.

2. Convert this geneont object into a relationship matrix.

   \text{[MATRIX, ID, REL]} = \text{getmatrix(GO)};
See Also

goannotread | num2goid | term
getmatrix (phytree)

Convert phytree object into relationship matrix

Syntax

\[
[\text{Matrix}, \ ID, \ Distances] = \text{getmatrix}(\text{PhytreeObj})
\]

Arguments

| PhytreeObj               | phytree object created by phytree (object constructor). |

Description

\([\text{Matrix}, \ ID, \ Distances] = \text{getmatrix}(\text{PhytreeObj})\) converts a phytree object, \(\text{PhytreeObj}\), into a logical sparse matrix, \(\text{Matrix}\), in which 1 indicates that a branch node (row index) is connected to its child (column index). The child can be either another branch node or a leaf node. \(\text{ID}\) is a column vector of strings listing the labels that correspond to the rows and columns of \(\text{Matrix}\), with the labels from 1 to \(\text{Number of Leaves}\) being the leaf nodes, then the labels from \(\text{Number of Leaves} + 1\) to \(\text{Number of Leaves} + \text{Number of Branches}\) being the branch nodes, and the label for the last branch node also being the root node. \(\text{Distances}\) is a column vector with one entry for every nonzero entry in \(\text{Matrix}\) traversed column-wise and representing the distance between the branch node and the child.

Examples

\[
T = \text{phytreeread('pf00002.tree')}
\]

\[
[\text{MATRIX}, \ ID, \ \text{DIST}] = \text{getmatrix}(T);
\]

See Also

get | pdist | phytree | phytreeviewer | prune

Topics

phytree object on page 1-1274

Introduced in R2006b
getweightmatrix (biograph)

Get connection matrix with weights from biograph object

**Syntax**

\[
\text{Matrix, ID} = \text{getweightmatrix}(\text{BGObj})
\]

**Arguments**

<table>
<thead>
<tr>
<th><strong>BGObj</strong></th>
<th>Biograph object created by biograph (object constructor).</th>
</tr>
</thead>
</table>

**Description**

\[
\text{Matrix, ID} = \text{getweightmatrix}(\text{BGObj})
\] converts the biograph object into a double sparse matrix, where non-zeros indicate the weight from the source node (row index) to the destination node (column index). \text{ID} is a list of the node's 'ID' property and corresponds to the rows and columns of \text{Matrix}.

**Examples**

\[
\begin{array}{cccccccccc}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 \\
\end{array}
\]

bg = biograph(cm);
[cm,IDs] = getweightmatrix(bg);

**See Also**

biograph | dolayout | getancestors | getdescendants | getedgesbynodeid | getnodesbyid | getrelatives | view

**Topics**

biograph object on page 1-185

**Introduced in R2006b**
getnewickstr (phytree)

Create Newick-formatted character vector

Syntax

\[
\text{nwk} = \text{getnewickstr}(	ext{Tree})
\]

\[
\text{getnewickstr}(..., '\text{PropertyName}', \text{PropertyValue}, ...)
\]

\[
\text{getnewickstr}(..., '\text{Distances}', \text{DistancesValue})
\]

\[
\text{getnewickstr}(..., '\text{BranchNames}', \text{BranchNamesValue})
\]

Arguments

<table>
<thead>
<tr>
<th>Tree</th>
<th>Phytree object created with the function phytree.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DistancesValue</td>
<td>Property to control including or excluding distances in the output. Enter either true (include distances) or false (exclude distances). Default is true.</td>
</tr>
<tr>
<td>BranchNamesValue</td>
<td>Property to control including or excluding branch names in the output. Enter either true (include branch names) or false (exclude branch names). Default is false.</td>
</tr>
</tbody>
</table>

Description

\[
\text{nwk} = \text{getnewickstr}(	ext{Tree}) \text{ returns the Newick-formatted character vector of a phylogenetic tree object (Tree).}
\]

\[
\text{getnewickstr}(..., '\text{PropertyName}', \text{PropertyValue}, ... \text{ defines optional properties using property name/value pairs.}
\]

\[
\text{getnewickstr}(..., '\text{Distances}', \text{DistancesValue}), \text{ when DistancesValue is false, excludes the distances from the output.}
\]

\[
\text{getnewickstr}(..., '\text{BranchNames}', \text{BranchNamesValue}), \text{ when BranchNamesValue is true, includes the branch names in the output.}
\]

Examples

1. Create some random sequences.
   \[
   \text{seqs} = \text{int2nt(ceil(rand(10)*4))};
   \]

2. Calculate pairwise distances.
   \[
   \text{dist} = \text{seqpdist(seqs,'alpha','nt')};
   \]

   \[
   \text{tree} = \text{seqlinkage(dist)};
   \]
Get the Newick-formatted character vector.

\[ \text{nwk} = \text{getnewickstr(tree)} \]

References

Information about the Newick tree format.


See Also

get | getbyname | getcanonical | phytree | phytreeread | phytreeviewer | phytreewrite | seqlinkage

Topics

phytree object on page 1-1274

Introduced before R2006a
getnodesbyid (biograph)

Get handles to nodes

Syntax

\[ \text{NodesHandles} = \text{getnodesbyid}(\text{BGobj}, \text{NodeIDs}) \]

Arguments

<table>
<thead>
<tr>
<th>(\text{BGobj})</th>
<th>Biograph object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{NodeIDs})</td>
<td>Enter a character vector or cell array of character vectors containing node identifications.</td>
</tr>
</tbody>
</table>

Description

\(\text{NodesHandles} = \text{getnodesbyid}(\text{BGobj}, \text{NodeIDs})\) gets the handles for the specified nodes \((\text{NodeIDs})\) in a biograph object.

Examples

1. Create a biograph object.
   
   ```
   \text{species} = \text{'Homosapiens', 'Pan', 'Gorilla', 'Pongo', 'Baboon', ...}
   \text{    'Macaca', 'Gibbon'};
   \text{cm} = \text{magic(7)} > 25 \& \text{1-eye(7)};
   \text{bg} = \text{biograph(cm, species)}
   ```

2. Find the handles to members of the Cercopithecidae family and members of the Hominidae family.
   
   ```
   \text{Cercopithecidae} = \text{'Macaca', 'Baboon'};
   \text{Hominidae} = \text{'Homosapiens', 'Pan', 'Gorilla', 'Pongo'};
   \text{CercopithecidaeNodes} = \text{getnodesbyid(bg, Cercopithecidae)};
   \text{HominidaeNodes} = \text{getnodesbyid(bg, Hominidae)};
   ```

3. Color the families differently and draw a graph.

See Also

biograph | dolayout | get | getancestors | getdescendants | getedgesbynodeid | getnodesbyid | getrelatives | set | view

Topics

biograph object on page 1-185

Introduced before R2006a
getOptionsTable

Return table with all properties and equivalent options in original syntax

Syntax

tbl = getOptionsTable(optionsObject)

Description

tbl = getOptionsTable(optionsObject) returns a table with all the object properties and equivalent options in the original syntax [1].

Examples

Retrieve Equivalent Original Options for Object Properties

Create a CufflinksOptions object.

opt = CufflinksOptions;

Retrieve the equivalent original options for all object properties.

getOptionsTable(opt)

ans =

35x3 table

<table>
<thead>
<tr>
<th>PropertyName</th>
<th>FlagName</th>
<th>FlagShortName</th>
</tr>
</thead>
<tbody>
<tr>
<td>EffectiveLengthCorrection</td>
<td>'EffectiveLengthCorrection'</td>
<td>'--no-effective-length-correction'</td>
</tr>
<tr>
<td>FauxReadTiling</td>
<td>'FauxReadTiling'</td>
<td>'--no-faux-reads'</td>
</tr>
<tr>
<td>FragmentBiasCorrection</td>
<td>'FragmentBiasCorrection'</td>
<td>'--frag-bias-correct'</td>
</tr>
<tr>
<td>FragmentLengthMean</td>
<td>'FragmentLengthMean'</td>
<td>'--frag-len-mean'</td>
</tr>
<tr>
<td>FragmentLengthSTD</td>
<td>'FragmentLengthSTD'</td>
<td>'--frag-len-std-dev'</td>
</tr>
<tr>
<td>GTFGuide</td>
<td>'GTFGuide'</td>
<td>'--GTF-guide'</td>
</tr>
<tr>
<td>IntronOverhangTolerance</td>
<td>'IntronOverhangTolerance'</td>
<td>'--intron-overhang-tolerance'</td>
</tr>
<tr>
<td>JunctionAlpha</td>
<td>'JunctionAlpha'</td>
<td>'--junc-alpha'</td>
</tr>
<tr>
<td>LengthCorrection</td>
<td>'LengthCorrection'</td>
<td>'--no-length-correction'</td>
</tr>
<tr>
<td>MaskFile</td>
<td>'MaskFile'</td>
<td>'--mask-file'</td>
</tr>
<tr>
<td>MaxBundleFfrags</td>
<td>'MaxBundleFfrags'</td>
<td>'--max-bundle-frags'</td>
</tr>
<tr>
<td>MaxBundleLength</td>
<td>'MaxBundleLength'</td>
<td>'--max-bundle-length'</td>
</tr>
<tr>
<td>MaxFragAlignments</td>
<td>'MaxFragAlignments'</td>
<td>'--max-frag-multihits'</td>
</tr>
<tr>
<td>MaxIntronLength</td>
<td>'MaxIntronLength'</td>
<td>'--max-intron-length'</td>
</tr>
</tbody>
</table>

Note: getOptionsTable also works on other cufflinks-related options objects, such as CuffDiffOptions, CuffCompareOptions, and CuffQuantOptions. For a complete list of objects, see “optionsObject” on page 1-0.
Input Arguments

optionsObject — Options
CufflinksOptions | CuffCompareOptions | CuffMergeOptions | CuffQuantOptions |
CuffDiffOptions | CuffNormOptions | CuffGFFReadOptions

Options, specified as a CufflinksOptions, CuffCompareOptions, CuffMergeOptions, CuffQuantOptions, CuffDiffOptions, CuffNormOptions, or CuffGFFReadOptions object.

Output Arguments

tbl — Object properties and corresponding original options
table

Object properties and their corresponding original options, returned as a table.

References


See Also

CuffCompareOptions | CuffDiffOptions | CuffGFFReadOptions | CuffMergeOptions | CuffNormOptions | CuffQuantOptions | CufflinksOptions

External Websites

Cufflinks manual
Introduced in R2019a
getpdb
Retrieve protein structure data from Protein Data Bank (PDB) database

Syntax

\[
PDBStruct = \text{getpdb}(PDBid)
\]

\[
PDBStruct = \text{getpdb}(PDBid, ...'ToFile', ToFileValue, ...)
\]

\[
PDBStruct = \text{getpdb}(PDBid, ...'SequenceOnly', SequenceOnlyValue, ...)
\]

Input Arguments

<table>
<thead>
<tr>
<th><strong>PDBid</strong></th>
<th>Character vector or string specifying a unique identifier for a protein structure record in the PDB database.</th>
</tr>
</thead>
</table>

**Note** Each structure in the PDB database is represented by a four-character alphanumeric identifier. For example, 4hhb is the identifier for hemoglobin.

<table>
<thead>
<tr>
<th><strong>ToFileValue</strong></th>
<th>Character vector or string specifying a file name or a path and file name for saving the PDB-formatted data. If you specify only a file name, that file will be saved in the MATLAB Current Folder.</th>
</tr>
</thead>
</table>

**Tip** After you save the protein structure record to a local PDB-formatted file, you can use the \text{pdbread} function to read the file into the MATLAB software offline or use the \text{molviewer} function to display and manipulate a 3-D image of the structure.

<table>
<thead>
<tr>
<th><strong>SequenceOnlyValue</strong></th>
<th>Controls the return of the protein sequence only. Choices are true or false (default).</th>
</tr>
</thead>
</table>

If there is one sequence, it is returned as a character array. If there are multiple sequences, they are returned as a cell array.

Output Arguments

<table>
<thead>
<tr>
<th><strong>PDBStruct</strong></th>
<th>MATLAB structure containing a field for each PDB record.</th>
</tr>
</thead>
</table>

Description

The Protein Data Bank (PDB) database is an archive of experimentally determined 3-D biological macromolecular structure data. getpdb retrieves protein structure data from the Protein Data Bank (PDB) database, which contains 3-D biological macromolecular structure data.

\[
PDBStruct = \text{getpdb}(PDBid)
\]

searches the PDB database for the protein structure record specified by the identifier \text{PDBid} and returns the MATLAB structure \text{PDBStruct}, which contains a field for each PDB record. The following table summarizes the possible PDB records and the corresponding fields in the MATLAB structure \text{PDBStruct}:
<table>
<thead>
<tr>
<th>PDB Database Record</th>
<th>Field in the MATLAB Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEADER</td>
<td>Header</td>
</tr>
<tr>
<td>OBSLTE</td>
<td>Obsolete</td>
</tr>
<tr>
<td>TITLE</td>
<td>Title</td>
</tr>
<tr>
<td>CAVEAT</td>
<td>Caveat</td>
</tr>
<tr>
<td>COMPND</td>
<td>Compound</td>
</tr>
<tr>
<td>SOURCE</td>
<td>Source</td>
</tr>
<tr>
<td>KEYWDS</td>
<td>Keywords</td>
</tr>
<tr>
<td>EXPDTA</td>
<td>ExperimentData</td>
</tr>
<tr>
<td>AUTHOR</td>
<td>Authors</td>
</tr>
<tr>
<td>REVDAT</td>
<td>RevisionDate</td>
</tr>
<tr>
<td>SPRSDE</td>
<td>Superseded</td>
</tr>
<tr>
<td>JRNAL</td>
<td>Journal</td>
</tr>
<tr>
<td>REMARK 1</td>
<td>Remark1</td>
</tr>
<tr>
<td>REMARK (N)</td>
<td>Remark(n)</td>
</tr>
</tbody>
</table>

**Note** \(N\) equals 2 through 999.  

| DBREF               | DBReferences                  |
| SEQADV              | SequenceConflicts             |
| SEQRES              | Sequence                      |
| FTNOTE              | Footnote                      |
| MODRES              | ModifiedResidues              |
| HET                 | Heterogen                     |
| HETNAM              | HeterogenName                 |
| HETSYN              | HeterogenSynonym              |
| FORMUL              | Formula                       |
| HELIX               | Helix                         |
| SHEET               | Sheet                         |
| TURN                | Turn                          |
| SSBOND              | SSBond                        |
| LINK                | Link                          |
| HYDBND              | HydrogenBond                  |
| SLTBRG              | SaltBridge                    |
| CISPEP              | CISPeptides                   |
| SITE                | Site                          |
| CRYST1              | Cryst1                        |
| ORIGXn              | OriginX                       |

getpdb

1-799
PDB Database Record | Field in the MATLAB Structure
--- | ---
SCALEn | Scale
MTRIXn | Matrix
TVECT | TranslationVector
MODEL | Model
ATOM | Atom
SIGATM | AtomSD
ANISOU | AnisotropicTemp
SIGUIJ | AnisotropicTempSD
TER | Terminal
HETATM | HeterogenAtom
CONECT | Connectivity

`PDBStruct = getpdb(PDBid, ...'PropertyName', PropertyValue, ...)` calls `getpdb` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`PDBStruct = getpdb(PDBid, ...'ToFile',ToFileValue, ...)` saves the data returned from the database to a PDB-formatted file, `ToFileValue`.

**Tip** After you save the protein structure record to a local PDB-formatted file, you can use the `pdbread` function to read the file into the MATLAB software offline or use the `molviewer` function to display and manipulate a 3-D image of the structure.

`PDBStruct = getpdb(PDBid, ...'SequenceOnly', SequenceOnlyValue, ...)` controls the return of the protein sequence only. Choices are true or false (default). If there is one sequence, it is returned as a character array. If there are multiple sequences, they are returned as a cell array.

### The Sequence Field

The `Sequence` field is also a structure containing sequence information in the following subfields:

- NumOfResidues
- ChainID
- ResidueNames — Contains the three-letter codes for the sequence residues.
- Sequence — Contains the single-letter codes for the sequence residues.

**Note** If the sequence has modified residues, then the `ResidueNames` subfield might not correspond to the standard three-letter amino acid codes. In this case, the `Sequence` subfield will contain the modified residue code in the position corresponding to the modified residue. The modified residue code is provided in the `ModifiedResidues` field.
The Model Field

The Model field is also a structure or an array of structures containing coordinate information. If the MATLAB structure contains one model, the Model field is a structure containing coordinate information for that model. If the MATLAB structure contains multiple models, the Model field is an array of structures containing coordinate information for each model. The Model field contains the following subfields:

- Atom
- AtomSD
- AnisotropicTemp
- AnisotropicTempSD
- Terminal
- HeterogenAtom

The Atom Field

The Atom field is also an array of structures containing the following subfields:

- AtomSerNo
- AtomName
- altLoc
- resName
- chainID
- resSeq
- iCode
- X
- Y
- Z
- occupancy
- tempFactor
- segID
- element
- charge
- AtomNameStruct — Contains three subfields: chemSymbol, remoteInd, and branch.

Examples

Retrieve the structure information for the electron transport (heme) protein that has a PDB identifier of 5CYT, read the information into a MATLAB structure pdbstruct, and save the information to a PDB-formatted file electron_transport.pdb in the MATLAB Current Folder:

```matlab
pdbstruct = getpdb('5CYT', 'ToFile', 'electron_transport.pdb')
```
See Also
getembl | getgenbank | getgenpept | molviewer | pdbdistplot | pdbread | pdbsuperpose | pdbtransform | pdbwrite

Introduced before R2006a
getQuality

Retrieve sequence quality information from object

Syntax

quality = getQuality(object)
subsetQuality = getQuality(object,subset)

Description

quality = getQuality(object) returns sequence quality information from a BioRead or BioMap object.

subsetQuality = getQuality(object,subset) returns the sequence quality information subsetQuality for only the object elements specified by subset.

Examples

Retrieve Sequence Quality Information

Store read data from a SAM-formatted file in a BioRead object.

br = BioRead('ex1.sam')

br =
    BioRead with properties:
    Quality: [1501x1 File indexed property]
    Sequence: [1501x1 File indexed property]
    Header: [1501x1 File indexed property]
    NSeqs: 1501
    Name: ''

Retrieve sequence quality information.

seqQuals = getQuality(br);

Retrieve sequence quality information from the first and third elements in the object.

seqQuals2 = getQuality(br,[1 3])

seqQuals2 = 2x1 cell
    {'<<<<<<<<<<<<<<<<<;<<<<<<<<<5<<<<;;><;7'}
    {'<<<<<<<<<<<<<<7;71<<;;<;7;<<3;)3*8/5'}

Use a logical vector to get the same information.

seqQuals3 = getQuality(br,[true false true])
seqQuals3 = 2x1 cell
    {'<<<<<<<<<<<;<<<<<<<<<5<<<<<<;:<;7'}
    {'<<<<<<<<<<<7;71<<;<;7;<7<<3;);3*8/5' }

You can use a header to get the quality of the corresponding sequence with that header. If multiple
sequences have the same header, the function returns the quality information of all those sequences.

Get the quality information of the sequences with the header B7_591:4:96:693:509.

seqQuals4 = getQuality(br,{'B7_591:4:96:693:509'})

seqQuals4 = 1x1 cell array
    {'<<<<<<<<<<<;<<<<<<<<<5<<<<<<;:<;7'}

Access each property of the object using the dot notation.

seqQuals = br.Quality;
seqQuals2   = br.Quality([1 3])

seqQuals2 = 2x1 cell
    {'<<<<<<<<<<<;<<<<<<<<<5<<<<<<;:<;7'}
    {'<<<<<<<<<<<7;71<<;<;7;<7<<3;);3*8/5' }

Input Arguments

object — Object containing read data
         BioRead object | BioMap object

Object containing the read data, specified as a BioRead or BioMap object.
Example: bioreadObj

subset — Subset of elements in object
          vector of positive integers | logical vector | string vector | cell array of character vectors

Subset of elements in the object, specified as a vector of positive integers, logical vector, string
vector, or cell array of character vectors containing valid sequence headers.
Example: [1 3]

Tip When you use a sequence header (or a cell array of headers) for subset, a repeated header
specifies all elements with that header.

Output Arguments

quality — Sequence quality information
          cell array of character vectors

Sequence quality information, returned as a cell array of character vectors. Each character is an
ASCII-encoded value of the log probability of a base being incorrect.
subsetQuality — Sequence quality information for subset of elements

Cell array of character vectors

Sequence quality information for a subset of elements from the object, returned as a cell array of character vectors.

See Also

BioMap | BioRead

Topics

“Manage Sequence Read Data in Objects”

Introduced in R2010a
getRange

Class: GFFAnnotation

Retrieve range of annotations from GFFAnnotation object

Syntax

Range = getRange(AnnotObj)

Description

Range = getRange(AnnotObj) returns Range, a 1-by-2 numeric array specifying the minimum and maximum positions in the reference sequence covered by annotations in AnnotObj.

Input Arguments

AnnotObj

Object of the GFFAnnotation class.

Default:

Output Arguments

Range

1-by-2 numeric array specifying the minimum and maximum positions in the reference sequence covered by annotations in AnnotObj

Examples

Construct a GFFAnnotation object from a GFF-formatted file that is provided with Bioinformatics Toolbox, and then return the range of the feature annotations:

% Construct a GFFAnnotation object from a GFF file
GFFAnnotObj = GFFAnnotation('tair8_1.gff');
% Return first and last positions of reference associated with feature annotations
range = getRange(GFFAnnotObj)

range =

    3631    498516

Tips

- Use the getSubset method with the Reference name-value pair to return a GFFAnnotation object containing only one reference sequence. Then use this subsetted object as input to the getRange method.
See Also
GFFAnnotation | GTFAnnotation | getData (GFFAnnotation) | getFeatureNames (GFFAnnotation) | getIndex (GFFAnnotation) | getRange (GFFAnnotation) | getReferenceNames (GFFAnnotation) | getSubset (GFFAnnotation)

Topics
“Store and Manage Feature Annotations in Objects”

External Websites
GFF (General Feature Format) specifications
getRange

Class: GTFAnnotation

Retrieve range of annotations from GTFAnnotation object

Syntax

Range = getRange(AnnotObj)

Description

Range = getRange(AnnotObj) returns Range, a 1-by-2 numeric array specifying the minimum and maximum positions in the reference sequence covered by annotations in AnnotObj.

Input Arguments

AnnotObj

Object of the GTFAnnotation class.

Default:

Output Arguments

Range

1-by-2 numeric array specifying the minimum and maximum positions in the reference sequence covered by annotations in AnnotObj

Examples

Construct a GTFAnnotation object from a GTF-formatted file that is provided with Bioinformatics Toolbox, and then return the range of the feature annotations:

% Construct a GTFAnnotation object from a GTF file
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
% Return first and last positions of reference associated with feature annotations
range = getRange(GTFAnnotObj)

range =

    41609    1371382

Tips

• Use the getSubset method with the Reference name-value pair to return a GFFAnnotation object containing only one reference sequence. Then use this subsetted object as input to the getRange method.
See Also
GFFAnnotation | GTFAnnotation | getData (GTFAnnotation) | getFeatureNames (GTFAnnotation) | getGeneNames (GTFAnnotation) | getGenes (GTFAnnotation) | getIndex (GTFAnnotation) | getRange (GTFAnnotation) | getReferenceNames (GTFAnnotation) | getSegments (GTFAnnotation) | getSubset (GTFAnnotation) | getTranscripts (GTFAnnotation)

Topics
"Store and Manage Feature Annotations in Objects"

External Websites
GTF2.2: A Gene Annotation Format
getReferenceNames

Class: GFFAnnotation
Retrieve reference names from GFFAnnotation object

Syntax
References = getReferenceNames(AnnotObj)

Description
References = getReferenceNames(AnnotObj) returns References, a cell array of character vectors specifying the names of all reference sequences in AnnotObj.

Input Arguments
AnnotObj
Object of the GFFAnnotation class.

Default:

Output Arguments
References
Cell array of character vectors specifying the names of all reference sequences in AnnotObj.

Examples
Construct a GFFAnnotation object from a GFF-formatted file that is provided with Bioinformatics Toolbox, and then return the names of the reference sequences from the annotation object:

% Construct a GFFAnnotation object from a GFF file
GFFAnnotObj = GFFAnnotation('tair8_1.gff');
% Return reference names for the annotation object
refNames = getReferenceNames(GFFAnnotObj)
refNames =
    'Chr1'

See Also
GFFAnnotation | GTFAnnotation | getData (GFFAnnotation) | getFeatureNames (GFFAnnotation) | getIndex (GFFAnnotation) | getRange (GFFAnnotation) |
getReferenceNames (GFFAnnotation) | getSubset (GFFAnnotation)

Topics
“Store and Manage Feature Annotations in Objects”
External Websites
GFF (General Feature Format) specifications
**getReferenceNames**

Class: `GTFAnnotation`

Retrieve reference names from `GTFAnnotation` object

**Syntax**

References = getReferenceNames(AnnotObj)

**Description**

References = getReferenceNames(AnnotObj) returns References, a cell array of character vectors specifying the names of all reference sequences in AnnotObj.

**Input Arguments**

AnnotObj

Object of the `GTFAnnotation` class.

Default:

**Output Arguments**

References

Cell array of character vectors specifying the names of all reference sequences in AnnotObj

**Examples**

Construct a `GTFAnnotation` object from a GTF-formatted file that is provided with Bioinformatics Toolbox, and then return the names of the reference sequences from the annotation object:

```matlab
% Construct a GTFAnnotation object from a GTF file
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
% Return reference names for the annotation object
refNames = getReferenceNames(GTFAnnotObj)
```

refNames =

'chr2'

**See Also**

`GFFAnnotation`, `GTFAnnotation`, `getData (GTFAnnotation)`, `getFeatureNames (GTFAnnotation)`, `getGeneNames (GTFAnnotation)`, `getGenes (GTFAnnotation)`, `getIndex (GTFAnnotation)`, `getRange (GTFAnnotation)`, `getReferenceNames (GTFAnnotation)`, `getSegments (GTFAnnotation)`, `getSubset (GTFAnnotation)`, `getTranscripts (GTFAnnotation)`
Topics
"Store and Manage Feature Annotations in Objects"

External Websites
GTF2.2: A Gene Annotation Format
getrelatives (biograph)

Find relatives of a node in biograph object

Syntax

Nodes = getrelatives(BiographNode)
Nodes = getrelatives(BiographNode,NumGenerations)

Arguments

<table>
<thead>
<tr>
<th>BiographNode</th>
<th>Node in a biograph object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumGenerations</td>
<td>Number of generations. Enter a positive integer.</td>
</tr>
</tbody>
</table>

Description

Nodes = getrelatives(BiographNode) finds all the direct relatives for a given node (BiographNode).

Nodes = getrelatives(BiographNode,NumGenerations) finds the direct relatives for a given node (BiographNode) up to a specified number of generations (NumGenerations). If the NumGenerations is 0, the function returns the node itself.

Examples

1. Create a biograph object.

```matlab
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
bg = biograph(cm)
```

2. Find all nodes interacting with node 1.

```matlab
intNodes = getrelatives(bg.nodes(1));
set(intNodes,'Color',[.7 .7 1]);
bg.view;
```

See Also

biograph | dolayout | get | getancestors | getdescendants | getedgesbynodeid | getnodesbyid | getrelatives | set | view

Topics

biograph object on page 1-185

Introduced before R2006a
getrelatives

Class: geneont

Find terms that are relatives of specified Gene Ontology (GO) term

Syntax

RelativeIDs = getrelatives(GeneontObj, ID)
[RelativeIDs, Counts] = getrelatives(GeneontObj, ID)

... = getrelatives(..., 'Height', HeightValue, ...)
... = getrelatives(..., 'Depth', DepthValue, ...)
... = getrelatives(..., 'Levels', LevelsValue, ...)
... = getrelatives(..., 'Relationtype', RelationtypeValue, ...)
... = getrelatives(..., 'Exclude', ExcludeValue, ...)

Description

RelativeIDs = getrelatives(GeneontObj, ID) searches GeneontObj, a geneont object, for GO terms that are relatives of the GO term(s) specified by ID, which is a GO term identifier or vector of identifiers. It returns RelativeIDs, a vector of GO term identifiers including ID. ID is a nonnegative integer or a vector containing nonnegative integers.

[RelativeIDs, Counts] = getrelatives(GeneontObj, ID) also returns the number of times each relative is found. Counts is a column vector with the same number of elements as terms in GeneontObj.

Tip The Counts return value is useful when you tally counts in gene enrichment studies. For more information, see “Gene Ontology Enrichment in Microarray Data”.

... = getrelatives(..., 'PropertyName', PropertyValue, ...) calls getrelatives with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = getrelatives(..., 'Height', HeightValue, ...) searches up through a specified number of levels, HeightValue, in the gene ontology. HeightValue is a positive integer. Default is 1.

... = getrelatives(..., 'Depth', DepthValue, ...) searches down through a specified number of levels, DepthValue, in the gene ontology. DepthValue is a positive integer. Default is 1.

... = getrelatives(..., 'Levels', LevelsValue, ...) searches up and down through a specified number of levels, LevelsValue, in the gene ontology. LevelsValue is a positive integer. When specified, it overrides HeightValue and DepthValue.
... = getrelatives(..., 'Relationtype', RelationtypeValue, ...) searches for specified relationship types, RelationtypeValue, in the gene ontology. RelationtypeValue is a character vector. Choices are 'is_a', 'part_of', or 'both' (default).

... = getrelatives(..., 'Exclude', ExcludeValue, ...) controls excluding ID, the original queried term(s), from the output RelativeIDs, unless a term was found while searching the gene ontology. Choices are true or false (default).

**Input Arguments**

<table>
<thead>
<tr>
<th>GeneontObj</th>
<th>A geneont object, such as created by the geneont constructor function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>GO term identifier or vector of identifiers.</td>
</tr>
<tr>
<td>HeightValue</td>
<td>Positive integer specifying the number of levels to search upward in the gene ontology.</td>
</tr>
<tr>
<td>DepthValue</td>
<td>Positive integer specifying the number of levels to search downward in the gene ontology.</td>
</tr>
<tr>
<td>LevelsValue</td>
<td>Positive integer specifying the number of levels up and down to search in the gene ontology. When specified, it overrides HeightValue and DepthValue.</td>
</tr>
</tbody>
</table>
| RelationtypeValue| Character vector specifying the relationship types to search for in the gene ontology. Choices are:
  * 'is_a'
  * 'part_of'
  * 'both' (default) |
| ExcludeValue     | Controls excluding ID, the original queried term(s), from the output RelativeIDs, unless the term was reached while searching the gene ontology. Choices are true or false (default). |

**Output Arguments**

<table>
<thead>
<tr>
<th>RelativeIDs</th>
<th>Vector of GO term identifiers including ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counts</td>
<td>Column vector with the same number of elements as terms in GeneontObj, indicating the number of times each relative is found.</td>
</tr>
</tbody>
</table>

**Examples**

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

   GO = geneont('LIVE', true)

   The MATLAB software creates a geneont object and displays the number of terms in the database.

   Gene Ontology object with 27769 Terms.

2 Retrieve the immediate relatives for the mitochondrial membrane GO term with a GO identifier of 31966.
relatives = getrelatives(GO, 31966, 'levels', 1)

relatives =
    5741
    5743
    31090
    31966
    44429

3 Create a subordinate Gene Ontology.

subontology = GO(relatives)

Gene Ontology object with 5 Terms.

4 Create a report of the subordinate Gene Ontology terms, that includes the GO identifier and name.

rpt = get(subontology.terms, {'id', 'name'})

rpt =

    [ 5741]             [1x28 char]
    [ 5743]             [1x28 char]
[31090]    'organelle membrane'
    [31966]             [1x22 char]
[44429]    'mitochondrial part'

5 View relationships of the subordinate Gene Ontology by using the getmatrix method to create a connection matrix to pass to the biograph function, and color the mitochondrial membrane GO term red.

[cm acc rels] = getmatrix(subontology);
BG = biograph(cm, get(subontology.terms, 'name'));
BG.nodes(acc==31966).Color = [1 0 0];
view(BG)
Retrieve all relatives for the mitochondrial outer membrane GO term with an identifier of 5741.

```matlab
relatives = getrelatives(GO,5741,'levels',inf);
```

Create a subordinate Gene Ontology.

```matlab
subontology = GO(relatives)
```

Gene Ontology object with 13 Terms.

View relationships of the subordinate Gene Ontology by using the `getmatrix` method to create a connection matrix to pass to the `biograph` function and methods, and color the mitochondrial outer membrane GO terms red.

```matlab
[cm acc rels] = getmatrix(subontology);
BG = biograph(cm, get(subontology.terms, 'name'));
BG.nodes(acc==5741).Color = [1 0 0];
view(BG)
```
See Also

goannotread | num2goid | term
getSegments

Class: GTFAnnotation

Return table of non-overlapping segments from GTFAnnotation object

Syntax

segments = getSegments(AnnotObj)
[segments,transcriptIDs] = getSegments(AnnotObj)
[___] = getSegments(AnnotObj,'Reference',R)
[___] = getSegments(AnnotObj,'Gene',G)
[___] = getSegments(AnnotObj,'Transcript',T)

Description

segments = getSegments(AnnotObj) returns segments, a table of non-overlapping segments of nucleotide sequences built by flattening the transcripts in AnnotObj. If an exon boundary is not the same in two or more transcripts of a gene, then the function creates two or more non-overlapping segments which cover all exons in the transcript.

[segments,transcriptIDs] = getSegments(AnnotObj) returns transcriptIDs, a cell array of character vectors containing all unique transcript IDs in AnnotObj.

[___] = getSegments(AnnotObj,'Reference',R) returns the segments that belong to the reference(s) specified by R.

[___] = getSegments(AnnotObj,'Gene',G) returns the segments that belong to the gene(s) specified by G.

[___] = getSegments(AnnotObj,'Transcript',T) returns the segments that belong to the transcript(s) specified by T.

Input Arguments

AnnotObj — GTF annotation
GTFAnnotation object

GTF annotation, specified as a GTFAnnotation object.

R — Names of reference sequences
character vector | string | string vector | cell array of character vectors | categorical array

Names of reference sequences, specified as a character vector, string, string vector, cell array of character vectors, or categorical array.

The names must come from the Reference field of AnnotObj. If a name does not exist, the function provides a warning and ignores it.

G — Names of genes
character vector | string | string vector | cell array of character vectors | categorical array
Names of genes, specified as a character vector, string, string vector, cell array of character vectors, or categorical array.

The names must come from the Gene field of AnnotObj. If a name does not exist, the function provides a warning and ignores the name.

**T — Names of transcripts**
character vector | string | string vector | cell array of character vectors | categorical array

Names of transcripts, specified as a character vector, string, string vector, cell array of character vectors, or categorical array.

The names must come from the Transcript field of AnnotObj. If a name does not exist, the function gives a warning and ignores the name.

**Output Arguments**

**segments — Non-overlapping segments**
table

Non-overlapping segments, returned as a table. The table contains the following variables for each segments.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Start location of each segment.</td>
</tr>
<tr>
<td>Stop</td>
<td>Stop location of each segment.</td>
</tr>
<tr>
<td>Reference</td>
<td>Categorical array representing the names of reference sequences to which the segments belong, obtained from the Reference field of AnnotObj.</td>
</tr>
<tr>
<td>ExonIndicator</td>
<td>Logical sparse matrix of segment versus exon. The rows represent segments. The columns are exons. If the i\text{th} segment is part of the j\text{th} exon, the element at position (ij) is 1. Otherwise, it is 0.</td>
</tr>
<tr>
<td>TranscriptIndicator</td>
<td>Logical sparse matrix of segment versus transcript. The rows represent segments and the columns are transcripts. The element at position (ij) is 1 if the i\text{th} segment is part of the j\text{th} transcript, and 0 otherwise.</td>
</tr>
</tbody>
</table>

**transcriptIDs — Unique transcript IDs**
cell array of character vectors

Unique transcript IDs, returned as a cell array of character vectors. The transcript IDs correspond to columns of the TranscriptIndicator variable of segments. For instance, the first element of transcriptIDs is the ID of the first column of TranscriptIndicator matrix.

**Examples**

**Retrieve Segments from a GTF-formatted File**

Create a GTFAntation object from a GTF-formatted file.

```matlab
obj = GTFAntation('hum37_2_1M.gtf');
```
Retrieve unique reference names. In this case, there is only one reference sequence, which is chromosome 2 (chr2).

\[ \text{ref} = \text{getReferenceNames(obj)} \]

\[ \text{ref} = 1x1 \ text{cell array} \]

\[ \{\text{chr2}\} \]

Get a table of all non-overlapping segments of nucleotide sequences which belong to chr2.

\[ \text{segments} = \text{getSegments(obj,'Reference',ref);} \]

See Also
GFFAnnotation | GTFAnnotation | getData (GTFAnnotation) | getFeatureNames (GTFAnnotation) | getGeneNames (GTFAnnotation) | getGenes (GTFAnnotation) | getIndex (GTFAnnotation) | getRange (GTFAnnotation) | getReferenceNames (GTFAnnotation) | getSegments (GTFAnnotation) | getSubset (GTFAnnotation) | getTranscripts (GTFAnnotation)

Topics
“Store and Manage Feature Annotations in Objects”

External Websites
GTF2.2: A Gene Annotation Format
getSequence

Retrieve sequences from object

Syntax

seqs = getSequence(object)
subsetSeqs = getSequence(object,subset)

Description

seqs = getSequence(object) returns nucleotide sequences from a BioRead or BioMap object.
subsetSeqs = getSequence(object,subset) returns the sequences subsetSeqs for only the object elements specified by subset.

Examples

Retrieve Sequences from NGS Data

Store read data from a SAM-formatted file in a BioRead object.

br = BioRead('ex1.sam')
br =
    BioRead with properties:
    
    Quality: [1501x1 File indexed property]
    Sequence: [1501x1 File indexed property]
    Header: [1501x1 File indexed property]
    NSeqs: 1501
    Name: ''

Retrieve the sequences (reads) from the object.

seqs = getSequence(br);

Retrieve the sequences from the first and third elements in the object.

seqs2 = getSequence(br,[1 3])

seqs2 = 2x1 cell

  {'CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG'}
  {'AGTGGCTCATTGTAAATGTGTGGTTTAACTCGTCC'}

Use a logical vector to get the same information.

seqs3 = getSequence(br,[true false true])

seqs3 = 2x1 cell

  {'CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG'}`
You can use a header to get the corresponding sequences with that header. If multiple sequences have the same header, the function returns all those sequences.

Get the sequences with the header B7_591:4:96:693:509.

```matlab
seqs4 = getSequence(br,{'B7_591:4:96:693:509'})
```

```
1x1 cell array
{ 'CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG'}
```

Access each property of the object using the dot notation.

```matlab
seqs = br.Sequence;
seq2 = br.Sequence([1 3])
```

```
2x1 cell
{ 'CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG'}
{ 'AGTGGCTCATTGTAAATGTGTGGTTTAACTCGTCC'}
```

**Input Arguments**

- **object** — Object containing read data
  BioRead object | BioMap object
  Object containing the read data, specified as a BioRead or BioMap object.
  Example: `bioreadObj`

- **subset** — Subset of elements in object
  vector of positive integers | logical vector | string vector | cell array of character vectors
  Subset of elements in the object, specified as a vector of positive integers, logical vector, string vector, or cell array of character vectors containing valid sequence headers.
  Example: `[1 3]`

**Tip** When you use a sequence header (or a cell array of headers) for `subset`, a repeated header specifies all elements with that header.

**Output Arguments**

- **seqs** — Nucleotide sequences
  cell array of character vectors
  Nucleotide sequences from the object, returned as a cell array of character vectors.

- **subsetSeqs** — Nucleotide sequences from subset of elements
  cell array of character vectors
Nucleotide sequences from a subset of elements from the object, returned as a cell array of character vectors.

**See Also**
BioMap | BioRead

**Topics**
“Manage Sequence Read Data in Objects”

**Introduced in R2010a**
getSubsequence

Retrieve partial sequences from object

**Syntax**

\[
\text{subSeqs} = \text{getSubsequence}(	ext{object, subset, positions})
\]

**Description**

\[
\text{subSeqs} = \text{getSubsequence}(	ext{object, subset, positions})
\]

returns the partial sequences \( \text{subSeqs} \) for sequence positions specified by \( \text{positions} \) from only object elements specified by \( \text{subset} \).

**Examples**

**Retrieve Subsequences from NGS Data**

Store read data from a SAM-formatted file in a BioRead object.

\[
br = \text{BioRead}('ex1.sam')
\]

\[
br =
\begin{align*}
\text{BioRead with properties:} \\
\text{Quality: [1501x1 File indexed property]} \\
\text{Sequence: [1501x1 File indexed property]} \\
\text{Header: [1501x1 File indexed property]} \\
\text{NSeqs: 1501} \\
\text{Name: ''}
\end{align*}
\]

Retrieve the sequences (reads) from the object.

\[
\text{seqs} = \text{getSequence}(br);
\]

Retrieve the first, third, and fifth sequences from the object.

\[
\text{seqs2} = \text{getSequence}(br,[1 3 5])
\]

\[
\text{seqs2} = 3x1 \text{ cell} \\
\{ 'CACTAGTGGCTACATTGAAATGTGGTTTAACTCG' \} \\
\{ 'AGTGCTGCTTGAATATGTTGGTTAACTCGG' \} \\
\{ 'GCTCATTGAAATGTGGTTTAACCTGTCAGG' \}
\]

Retrieve the first five positions of those sequences.

\[
\text{seqs3} = \text{getSubsequence}(br,[1 3 5],[1:5])
\]

\[
\text{seqs3} = 3x1 \text{ cell} \\
\{ 'CACTA' \}
\]
You can use a header to get the corresponding sequences with that header. If multiple sequences have the same header, the function returns all of those sequences.

Get the first five positions of the sequences with the header B7_591:4:96:693:509.

```matlab
seqs4 = getSubsequence(br, {'B7_591:4:96:693:509'}, [1:5])
```

```matlab
seqs4 = 1x1 cell array
    {'CACTA'}
```

Retrieve the first, fourth, and sixth positions of the first three sequences.

```matlab
seq5 = getSubsequence(br, [1:3], [1 4 6])
```

```matlab
seq5 = 3x1 cell
    {'CTG'}
    {'CGG'}
    {'AGC'}
```

### Input Arguments

**object** — Object containing read data

BioRead object | BioMap object

Object containing the read data, specified as a BioRead or BioMap object.

Example: `bioreadObj`

**subset** — Subset of elements in object

vector of positive integers | logical vector | string vector | cell array of character vectors

Subset of elements in the object, specified as a vector of positive integers, logical vector, string vector, or cell array of character vectors containing valid sequence headers.

Example: `[1 3]`

**positions** — Sequence positions

vector of positive integers | logical vector

Sequence positions, specified as a vector of positive integers or logical vector. The last position must be within the range of positions for each sequence specified by `subset`.

Example: `[2:10]`
Output Arguments

subSeqs — Subsequences from subset of elements
cell array of character vectors

Subsequences from a subset of elements, returned as a cell array of character vectors.

See Also
BioMap | BioRead

Topics
“Manage Sequence Read Data in Objects”

Introduced in R2010a
getSubset

Class: BioIndexedFile

Create object containing subset of elements from BioIndexedFile object

Syntax

NewObj = getSubset(BioIFObj, Indices)
NewObj = getSubset(BioIFObj, Keys)

Description

NewObj = getSubset(BioIFObj, Indices) returns NewObj, a new BioIndexedFile object that accesses a subset of entries in the source file associated with BioIFObj, a BioIndexedFile object. The entries are specified by Indices, a vector containing unique positive integers.

NewObj = getSubset(BioIFObj, Keys) returns NewObj, a new BioIndexedFile object that accesses a subset of entries in the source file associated with BioIFObj, a BioIndexedFile object. The entries are specified by Keys, a character vector or cell array of unique character vectors specifying keys.

Input Arguments

BioIFObj

Object of the BioIndexedFile class.

Default:

Indices

Vector containing unique positive integers that specify the entries in the source file to access with NewObj. The number of elements in Indices cannot exceed the number of entries indexed by BioIFObj. There is a one-to-one relationship between the elements in Indices and the entries that NewObj accesses.

Keys

Character vector or cell array of unique character vectors specifying keys that specify the entries in the source file to access with NewObj. The number of elements in Keys is less than or equal to the number of entries indexed by BioIFObj. If the keys in the source file are not unique, then all entries that match a given key are indexed by NewObj. In this case, there is not a one-to-one relationship between the elements in Keys and the entries that NewObj accesses. If the keys in the source file are unique, then there is a one-to-one relationship between the elements in Keys and the entries that NewObj accesses.
Output Arguments

NewObj

Object of the `BioIndexedFile` class.

Examples

Construct a `BioIndexedFile` object to access a table containing cross-references between gene names and gene ontology (GO) terms:

```matlab
% Create a variable containing the full absolute path of the source file.
sourcefile = which('yeastgenes.sgd');
% Create a BioIndexedFile object from the source file. Indicate
% the source file is a tab-delimited file where contiguous rows
% with the same key are considered a single entry. Store the
% index file in the Current Folder. Indicate that keys are
% located in column 3 and that header lines are prefaced with !
gene2goObj = BioIndexedFile('mrtab', sourcefile, '.', ...
    'KeyColumn', 3, 'HeaderPrefix','!')
```

Create a new `BioIndexedFile` object that accesses only the first 1,000 cross-references and reuses the same index file as `gene2goObj`:

```matlab
% Create a new BioIndexedFile object.
gene2goSubset = getSubset(gene2goObj,1:1000);
```

Tips

Use this method to create a smaller, more manageable `BioIndexedFile` object.

See Also

`BioIndexedFile` | `getEntryByIndex` | `getEntryByKey` | `getIndexByKey` | `getKeys` | `read`

Topics

“Work with Next-Generation Sequencing Data”
getSubset

Retrieve subset of elements from object

Syntax

subset = getSubset(object,subset)
subset = getSubset(object,subset,Name,Value)

Description

subset = getSubset(object,subset) returns the sequence read data subset for only the
object elements specified by subset.

subset = getSubset(object,subset,Name,Value) uses additional options specified by one or
more name-value pair arguments. For example, you can specify whether to keep the data in memory.

Examples

Retrieve Subset of Elements from NGS Data

Store read data from a SAM-formatted file in a BioRead object. By default, the data remains in the
source file, and BioRead uses an index file to access the data, making the process more memory
efficient.

br = BioRead('ex1.sam')

br =
BioRead with properties:

  Quality: [1501x1 File indexed property]
  Sequence: [1501x1 File indexed property]
    Header: [1501x1 File indexed property]
    NSeqs: 1501
    Name: ''

Set the 'InMemory' name-value pair argument to true to store the data in memory, enabling you to
access the data faster and edit the properties of the object.

brInMemory = BioRead('ex1.sam','InMemory',true)

brInMemory =
BioRead with properties:

  Quality: {1501x1 cell}
  Sequence: {1501x1 cell}
    Header: {1501x1 cell}
    NSeqs: 1501
    Name: ''
Retrieve the second and third elements from the object \texttt{br}. By default, the resulting object \texttt{subset} is not placed in memory if the parent object \texttt{br} is not in memory. If \texttt{br} is already in memory, the resulting subset is placed in memory.

\texttt{subset = getSubset(br,[2 3])}

\texttt{subset = }
\begin{verbatim}
BioRead with properties:
    Quality: [2x1 File indexed property]
    Sequence: [2x1 File indexed property]
    Header: [2x1 File indexed property]
    NSeqs: 2
    Name: ''
\end{verbatim}

Alternatively, you can keep the parent object \texttt{br} in the source file, and load the resulting subset in memory if the subset is small enough. You access the subset faster and update it as needed.

\texttt{subsetInMemory = getSubset(br,[2 3],''InMemory'',true)}

\texttt{subsetInMemory = }
\begin{verbatim}
BioRead with properties:
    Quality: {2x1 cell}
    Sequence: {2x1 cell}
    Header: {2x1 cell}
    NSeqs: 2
    Name: ''
\end{verbatim}

Update the header information of the first element.

\texttt{subsetInMemory.Header(1)}

\texttt{ans = 1x1 cell array}
\begin{verbatim}
{'EAS54_65:7:152:368:113'}
\end{verbatim}

\texttt{subsetInMemory.Header(1) = {'NewHeader'}};
\texttt{subsetInMemory.Header(1)}

\texttt{ans = 1x1 cell array}
\begin{verbatim}
{'NewHeader'}
\end{verbatim}

You can use a header to get the corresponding elements with that header. If multiple elements have the same header, the function returns all those elements.

Get all the elements with the header 'B7_591:4:96:693:509' from the \texttt{br} object stored in memory.

\texttt{subset2 = getSubset(brInMemory,{'B7_591:4:96:693:509'})}

\texttt{subset2 = }
\begin{verbatim}
BioRead with properties:
    Quality: {'<<<<<<<<<<<<<<<;<<<<<<<<<5<<<<<<;:<;7'}
    Sequence: {'CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG'}
\end{verbatim}
Input Arguments

**object — Object containing read data**

BioRead object | BioMap object

Object containing the read data, specified as a BioRead or BioMap object.

Example: `bioreadObj`

**subset — Subset of elements in object**

vector of positive integers | logical vector | string vector | cell array of character vectors

Subset of elements in the object, specified as a vector of positive integers, logical vector, string vector, or cell array of character vectors containing valid sequence headers.

Example: `[1 3]`

**Tip** When you use a sequence header (or a cell array of headers) for `subset`, a repeated header specifies all elements with that header.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'InMemory',true` specifies to save the output object (`subset`) in memory.

**Name — Name of object**

' ' (default) | character vector | string

Name of the object, specified as the comma-separated pair consisting of `'Name'` and a character vector or string. The default is an empty character vector `' ' (no name).

Example: `'Name','newData'`

**InMemory — Logical flag to keep data in memory**

false (default) | true

Logical flag to keep data in memory, specified as the comma-separated pair consisting of `'InMemory'` and `true` or `false`. Keeping the data in memory lets you access the resulting object `subset` faster and update its properties. If the data specified for `subset` is still large and does not fit in memory, set this name-value pair to `false` to use indexed access, which is more memory efficient but does not enable you to modify the properties.

If the parent `object` is already in memory, the resulting object `subset` is automatically placed in memory, and the function ignores this argument.

Example: `'InMemory',true"
SelectReference — References used to create subset of data

References used to create the subset of data with only the reads mapped to those references, specified as the comma-separated pair consisting of 'SelectReference' and a cell array of character vectors, string vector, or vector of positive integers.

**Note** This argument is for the BioMap objects only.

Example: 'SelectReference','{'RefSeq1'}

### Output Arguments

**subset** — Subset of elements

BioRead object | BioMap object

Subset of elements from the object, returned as a BioRead or BioMap object. If object is in memory, then subset is placed in memory. If object is indexed, then subset is indexed unless you set 'InMemory' to true.

**See Also**

BioMap | BioRead

**Topics**

"Manage Sequence Read Data in Objects"

**Introduced in R2010a**
getSubset

Class: GFFAnnotation

Retrieve subset of elements from GFFAnnotation object

Syntax

NewObj = getSubset(AnnotObj,StartPos,EndPos)
NewObj = getSubset(AnnotObj,Subset)
NewObj = getSubset(___ ,Name,Value)

Description

NewObj = getSubset(AnnotObj,StartPos,EndPos) returns NewObj, a new object containing a subset of the elements from AnnotObj that falls within each reference sequence range specified by StartPos and EndPos.

NewObj = getSubset(AnnotObj,Subset) returns NewObj, a new object containing a subset of elements specified by Subset, a vector of integers.

NewObj = getSubset(___ ,Name,Value) returns NewObj, a new object containing a subset of the elements from AnnotObj, using any of the input arguments from the previous syntaxes and additional options specified by one or more Name,Value pair arguments.

Input Arguments

AnnotObj

Object of the GFFAnnotation class.

Default:

StartPos

Nonnegative integer specifying the start of a range in each reference sequence in AnnotObj. The integer StartPos must be less than or equal to EndPos.

Default:

EndPos

Nonnegative integer specifying the end of a range in each reference sequence in AnnotObj. The integer EndPos must be greater than or equal to StartPos.

Default:

Subset

Vector of positive integers equal or less than the number of entries in the object. Use the vector Subset to retrieve any element or subset of the object.
**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

**Reference**

Character vector or cell array of character vectors specifying one or more reference sequences in AnnotObj. Only annotations whose reference field matches one of the character vectors are included in NewObj.

**Default:**

**Feature**

Character vector or cell array of character vectors specifying one or more features in AnnotObj. Only annotations whose feature field matches one of the character vectors are included in NewObj.

**Default:**

**Overlap**

Minimum number of base positions that an annotation must overlap in the range, to be included in NewObj. This value can be any of the following:

- Positive integer
- 'full' — An annotation must be fully contained in the range to be included.
- 'start' — An annotation’s start position must lie within the range to be included.

**Default:** 1

**Output Arguments**

**NewObj**

Object of the GFFAnnotation class.

**Examples**

**Example 1.32. Create a Subset of Data Containing Only Protein Features from a GFF-formatted File**

Construct a GFFAnnotation object using a GFF-formatted file that is provided with Bioinformatics Toolbox.

```matlab
GFFAnnotObj = GFFAnnotation('tair8_1.gff');
```

Create a subset of data containing only protein features.

```matlab
subsetGFF1 = getSubset(GFFAnnotObj,'Feature','protein')
subsetGFF1 =
```
Example 1.33. Retrieve Subsets of Data from a GFFAnnotation Object

Construct a GFFAnnotation object using a GFF-formatted file that is provided with Bioinformatics Toolbox.

GFFAnnotObj = GFFAnnotation('tair8_1.gff');

Retrieve a subset of data from the first to fifth elements of GFFAnnotObj.

subsetGFF2 = getSubset(GFFAnnotObj,[1:5])

subsetGFF2 =

GFFAnnotation with properties:

    FieldNames: {1x9 cell}
    NumEntries: 5

Retrieve only the first, fifth and eighth elements of GFFAnnotObj.

subsetGFF3 = getSubset(GFFAnnotObj,[1 5 8])

subsetGFF3 =

GFFAnnotation with properties:

    FieldNames: {1x9 cell}
    NumEntries: 3

Tips

- The getSubset method selects annotations from the range specified by StartPos and EndPos for all reference sequences in AnnotObj unless you use the Reference name-value pair argument to limit the reference sequences.
- After creating a subsetted object, you can access the number of entries, range of reference sequence covered by annotations, field names, and reference names. To access the values of all fields, create a structure of the data using the getData method.

See Also

GFFAnnotation | GTFAnnotation | getData (GFFAnnotation) | getFeatureNames (GFFAnnotation) | getIndex (GFFAnnotation) | getRange (GFFAnnotation) | getReferenceNames (GFFAnnotation) | getSubset (GFFAnnotation)

Topics

"Store and Manage Feature Annnotations in Objects"

External Websites

GFF (General Feature Format) specifications
getSubset

Class: GTFAnnotation

Create object containing subset of elements from GTFAnnotation object

Syntax

NewObj = getSubset(AnnotObj,StartPos,EndPos)
NewObj = getSubset(AnnotObj,Subset)
NewObj = getSubset(___,Name,Value)

Description

NewObj = getSubset(AnnotObj,StartPos,EndPos) returns NewObj, a new object containing a subset of the elements from AnnotObj that falls within each reference sequence range specified by StartPos and EndPos.

NewObj = getSubset(AnnotObj,Subset) returns NewObj, a new object containing a subset of elements specified by Subset, a vector of integers.

NewObj = getSubset(___,Name,Value) returns NewObj, a new object containing a subset of the elements from AnnotObj, using any of the input arguments from the previous syntaxes and additional options specified by one or more Name,Value pair arguments.

Input Arguments

AnnotObj
Object of the GTFAnnotation class.

Default:

StartPos
Nonnegative integer specifying the start of a range in each reference sequence in AnnotObj. The integer StartPos must be less than or equal to EndPos.

Default:

EndPos
Nonnegative integer specifying the end of a range in each reference sequence in AnnotObj. The integer EndPos must be greater than or equal to StartPos.

Default:

Subset
Vector of positive integers less than or equal to the number of entries in the object. Use the vector Subset to retrieve any element or subset of the object.
**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Reference**

Character vector, string, string vector, or cell array of character vectors specifying one or more reference sequences in `AnnotObj`. Only annotations whose reference field matches one of the character vectors or strings are included in `NewObj`.

**Feature**

Character vector, string, string vector, or cell array of character vectors specifying one or more features in `AnnotObj`. Only annotations whose feature field matches one of the character vectors or strings are included in `NewObj`.

**Gene**

Character vector, string, string vector, or cell array of character vectors specifying one or more genes in `AnnotObj`. Only annotations whose gene field matches one of the character vectors or strings are included in `NewObj`.

**Transcript**

Character vector, string, string vector, or cell array of character vectors specifying one or more transcripts in `AnnotObj`. Only annotations whose transcript field matches one of the character vectors or strings are included in `NewObj`.

**Overlap**

Minimum number of base positions that an annotation must overlap in the range, to be included in `NewObj`. This value can be any of the following:

- Positive integer
- `'full'` — An annotation must be fully contained in the range to be included.
- `'start'` — An annotation’s start position must lie within the range to be included.

**Default:** 1

**Output Arguments**

**NewObj**

Object of the `GTFAnnotation` class.

**Examples**

**Example 1.34. Create a Subset of Data Containing Only CDS Features from a GTF-formatted File**

Construct a `GTFAnnotation` object using a GTF-formatted file that is provided with Bioinformatics Toolbox.
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');

Create a subset of the data containing only CDS features.

subsetGTF = getSubset(GTFAnnotObj,'Feature','CDS')

subsetGTF =

    GTFAnnotation with properties:
        FieldNames: {1x11 cell}
        NumEntries: 92

**Example 1.35. Retrieve Subsets of Data from a GTFAnnotation Object**

Construct a GTFAnnotation object using a GTF-formatted file that is provided with Bioinformatics Toolbox.

GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');

Retrieve a subset of data from the first to fifth elements of GTFAnnotObj.

subsetGTF1 = getSubset(GTFAnnotObj,[1:5])

subsetGTF1 =

    GTFAnnotation with properties:
        FieldNames: {1x11 cell}
        NumEntries: 5

Retrieve only the first, fifth and eighth elements of GTFAnnotObj.

subsetGTF2 = getSubset(GTFAnnotObj,[1 5 8])

subsetGTF2 =

    GTFAnnotation with properties:
        FieldNames: {1x11 cell}
        NumEntries: 3

**Tips**

• The getSubset method selects annotations from the range specified by StartPos and EndPos for each reference sequence in AnnotObj unless you use the 'Reference' name-value pair argument to limit the reference sequences.

• After creating a subsetted object, you can access the number of entries, range of reference sequences covered by annotations, field names, and reference names. To access the values of all fields, create a structure of the data using the getData method.

**See Also**

GFFAnnotation | GTFAnnotation | getData (GTFAnnotation) | getFeatureNames (GTFAnnotation) | getGeneNames (GTFAnnotation) | getGenes (GTFAnnotation) | getIndex (GTFAnnotation) | getRange (GTFAnnotation) | getReferenceNames
(GTFAnnotation) | getSegments (GTFAnnotation) | getSubset (GTFAnnotation) | getTranscripts (GTFAnnotation)

**Topics**

“Store and Manage Feature Annotations in Objects”

**External Websites**

GTF2.2: A Gene Annotation Format
getTranscripts

Class: GTFAnnotation

Return table of unique transcripts in GTFAnnotation object

Syntax

transcriptsTable = getTranscripts(AnnotObj)
transcriptsTable = getTranscripts(AnnotObj,'Reference',R)
transcriptsTable = getTranscripts(AnnotObj,'Gene',G)
transcriptsTable = getTranscripts(AnnotObj,'Transcript',T)

Description

transcriptsTable = getTranscripts(AnnotObj) returns transcriptsTable, a table of transcripts referenced by exons in AnnotObj.

transcriptsTable = getTranscripts(AnnotObj,'Reference',R) returns the transcript(s) that belong to the reference(s) specified by R.

transcriptsTable = getTranscripts(AnnotObj,'Gene',G) returns the transcript(s) that belong to the gene(s) specified by G.

transcriptsTable = getTranscripts(AnnotObj,'Transcript',T) returns the transcript(s) specified by T.

Input Arguments

AnnotObj — GTF annotation
GTFAnnotation object

R — Names of reference sequences
character vector | string | string vector | cell array of character vectors | categorical array

Names of reference sequences, specified as a character vector, string, string vector, cell array of character vectors, or categorical array.

The names must come from the Reference field of AnnotObj. If a name does not exist, the function provides a warning and ignores it.

G — Names of genes
character vector | string | string vector | cell array of character vectors | categorical array

Names of genes, specified as a character vector, string, string vector, cell array of character vectors, or categorical array.

The names must come from the Gene field of AnnotObj. If a name does not exist, the function provides a warning and ignores the name.
**T — Names of transcripts**

character vector | string | string vector | cell array of character vectors | categorical array

Names of transcripts, specified as a character vector, string, string vector, cell array of character vectors, or categorical array.

The names must come from the Transcript field of AnnotObj. If a name does not exist, the function gives a warning and ignores the name.

**Output Arguments**

**transcriptsTable — Transcripts**

*table*

Transcripts, returned as a table. The table contains the following variables for each transcript.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transcript</td>
<td>Cell array of character vectors containing transcript IDs, obtained from the Transcript field of AnnotObj.</td>
</tr>
<tr>
<td>GeneName</td>
<td>Cell array of character vectors containing the names of expressed genes, obtained from the Attributes field of AnnotObj. This cell array can contain empty character vectors if the corresponding gene names are not found in Attributes.</td>
</tr>
<tr>
<td>GeneID</td>
<td>Cell array of character vectors containing the expressed gene IDs, obtained from the Gene field of AnnotObj.</td>
</tr>
<tr>
<td>Reference</td>
<td>Categorical array representing the names of reference sequences to which the expressed genes belong. The reference names are from the Reference field of AnnotObj.</td>
</tr>
<tr>
<td>Start</td>
<td>Start location of the first exon in each transcript.</td>
</tr>
<tr>
<td>Stop</td>
<td>Stop location of the last exon in each transcript.</td>
</tr>
<tr>
<td>Strand</td>
<td>Categorical array containing the strand of expressed gene.</td>
</tr>
</tbody>
</table>

**Examples**

**Retrieve Transcripts from a GTF-formatted File**

Create a GTFAnnotation object from a GTF-formatted file.

```matlab
obj = GTFAnnotation('hum37_2_1M.gtf');
```

Get the list of gene names listed in the object.

```matlab
gNames = getGeneNames(obj)
gNames = 28x1 cell
  {'uc002qvu.2'}
  {'uc002qvv.2'}
  {'uc002qvw.2'}
  {'uc002qvx.2'}
  {'uc002qvy.2'}
```
Get a table of transcripts which belong to the first gene uc002qvu.2.

```matlab
transcripts = getTranscripts(obj,'Gene',gNames{1})
```

```
<table>
<thead>
<tr>
<th>Transcript</th>
<th>GeneName</th>
<th>GeneID</th>
<th>Reference</th>
<th>Start</th>
<th>Stop</th>
<th>Strand</th>
</tr>
</thead>
<tbody>
<tr>
<td>uc002qvu.2</td>
<td>0x0 char</td>
<td>uc002qvu.2</td>
<td>chr2</td>
<td>218138</td>
<td>249852</td>
<td>-</td>
</tr>
</tbody>
</table>
```

See Also

GFFAnnotation | GTFAnnotation | getData (GTFAnnotation) | getFeatureNames (GTFAnnotation) | getGeneNames (GTFAnnotation) | getGenes (GTFAnnotation) | getIndex (GTFAnnotation) | getRange (GTFAnnotation) | getReferenceNames (GTFAnnotation) | getSegments (GTFAnnotation) | getSubset (GTFAnnotation) | getTranscripts (GTFAnnotation)

Topics

“Store and Manage Feature Annotations in Objects”

External Websites

GTF2.2: A Gene Annotation Format
getTranscriptNames

Class: GTFAnnotation

Retrieve unique transcript names from GTFAnnotation object

Syntax

Transcripts = getTranscriptNames(AnnotObj)

Description

Transcripts = getTranscriptNames(AnnotObj) returns Transcripts, a cell array of character vectors specifying the unique transcript names associated with annotations in AnnotObj.

Input Arguments

AnnotObj

Object of the GTFAnnotation class.

Default:

Output Arguments

Transcripts

Cell array of character vectors specifying the unique transcript names associated with annotations in AnnotObj.

Examples

Construct a GTFAnnotation object from a GTF-formatted file that is provided with Bioinformatics Toolbox, and then retrieve a list of the unique transcript names from the object:

% Construct a GTFAnnotation object from a GTF file
GTFAnotObj = GTFAnnotation('hum37_2_1M.gtf');
% Get transcript names from object
transcriptNames = getTranscriptNames(GTFAnotObj)

transcriptNames =
    'uc002qvu.2'
    'uc002qvv.2'
    'uc002qvw.2'
    'uc002qvx.2'
    'uc002qvy.2'
    'uc002qvz.2'
    'uc002qwa.2'
    'uc002qwb.2'
    'uc002qwc.1'
See Also
GFFAnnotation | GTFAnnotation | getData (GTFAnnotation) | getFeatureNames (GTFAnnotation) | getGeneNames (GTFAnnotation) | getGenes (GTFAnnotation) | getIndex (GTFAnnotation) | getRange (GTFAnnotation) | getReferenceNames (GTFAnnotation) | getSegments (GTFAnnotation) | getSubset (GTFAnnotation) | getTranscripts (GTFAnnotation)

Topics
“Store and Manage Feature Annotations in Objects”

External Websites
GTF2.2: A Gene Annotation Format
GFFAnnotation class

Contain General Feature Format (GFF) annotations

Description

The GFFAnnotation class contains annotations for one or more reference sequences, conforming to the GFF file format.

You construct a GFFAnnotation object from a GFF- or GTF-formatted file. Each element in the object represents an annotation. Use the object properties and methods to filter annotations by feature, reference sequence, or reference sequence position. Use object methods to extract data for a subset of annotations into an array of structures.

Construction

Annotobj = GFFAnnotation(File) constructs Annotobj, a GFFAnnotation object, from File, a GFF- or GTF-formatted file.

Input Arguments

File

Character vector or string specifying a GFF- or GTF-formatted file.

Properties

FieldNames

Cell array of Character vectors specifying the names of the available data fields for each annotation in the GFFAnnotation object. This property is read only.

NumEntries

Integer specifying number of annotations in the GFFAnnotation object. This property is read only.

Methods

getdataCreate structure containing subset of data from GFFAnnotation
getFeatureNamesRetrieve unique feature names from GFFAnnotation object
getIndexReturn index array of annotations from GFFAnnotation object
getRangeRetrieve range of annotations from GFFAnnotation object
getReferenceNamesRetrieve reference names from GFFAnnotation object
getSubsetRetrieve subset of elements from GFFAnnotation object
Copy Semantics

Value. To learn how value classes affect copy operations, see Copying Objects (MATLAB).

Indexing

GFFAnnotation objects support dot indexing to extract properties.

Examples

Construct a GFFAnnotation object from a GFF-formatted file that is provided with Bioinformatics Toolbox:

```matlab
GFFAnnotObj = GFFAnnotation('tair8_1.gff')
```

```
GFFAnnotObj =

   GFFAnnotation with properties:

   FieldNames: {'Reference' 'Start' 'Stop' 'Feature' 'Source' 'Score' 'Strand' 'Frame' 'Attributes'}
   NumEntries: 3331
```

Construct a GFFAnnotation object from a GTF-formatted file that is provided with Bioinformatics Toolbox:

```matlab
GFFAnnotObj = GFFAnnotation('hum37_2_1M.gtf')
```

```
GFFAnnotObj =

   GFFAnnotation with properties:

   FieldNames: {'Reference' 'Start' 'Stop' 'Feature' 'Source' 'Score' 'Strand' 'Frame' 'Attributes'}
   NumEntries: 308
```

See Also

GTFAnnotation

Topics

“Store and Manage Feature Annotations in Objects”

External Websites

GFF (General Feature Format) specifications
**goannotread**

Read annotations from Gene Ontology annotated file

**Syntax**

\[
\text{Annotation} = \text{goannotread}(\text{File}) \\
\text{Annotation} = \text{goannotread}(\text{File, ...}'\text{Fields}', \text{FieldsValue}, ...) \\
\text{Annotation} = \text{goannotread}(\text{File, ...}'\text{Aspect}', \text{AspectValue}, ...) 
\]

**Input Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File</strong></td>
<td>Character vector or string specifying a file name of a Gene Ontology (GO) annotated format (GAF) file.</td>
</tr>
<tr>
<td><strong>FieldsValue</strong></td>
<td>Character vector, string, string vector, or cell array of character vectors specifying one or more fields to read from the Gene Ontology annotated file. Default is to read all fields. Valid fields are listed below.</td>
</tr>
<tr>
<td><strong>AspectValue</strong></td>
<td>Character vector or string specifying one or more characters. Valid aspects are: * P — Biological process * F — Molecular function * C — Cellular component Default is 'CFP', which specifies to read all aspects.</td>
</tr>
</tbody>
</table>

**Output Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Annotation</strong></td>
<td>MATLAB array of structures containing annotations from a Gene Ontology annotated file.</td>
</tr>
</tbody>
</table>

**Description**

**Note** The `goannotread` function supports GAF 1.0 and 2.0 file formats.

\[
\text{Annotation} = \text{goannotread}(\text{File}) \text{ converts the contents of } \text{File}, \text{ a Gene Ontology annotated file, into } \text{Annotation}, \text{ an array of structures. Files should have the structure specified by the Gene Ontology consortium, available at:} \\
\text{http://www.geneontology.org/} \\
\text{Annotation} = \text{goannotread}(\text{File, ...}'\text{PropertyName}', \text{PropertyValue}, ...) \text{ calls goannotread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each } \text{PropertyName} \text{ must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:} 
\]

1-849
Annotation = goannotread(File, ...'Fields', FieldsValue, ...) specifies the fields to read from the Gene Ontology annotated file. FieldsValue is a character vector, string, string vector, or cell array of character vectors specifying one or more fields. Default is to read all fields. Valid fields are:

- Database
- DB_Object_ID
- DB_Object_Symbol
- Qualifier
- GOid
- DBReference
- Evidence
- WithFrom
- Aspect
- DB_Object_Name
- Synonym
- DB_Object_Type
- Taxon
- Date
- Assigned_by

Annotation = goannotread(File, ...'Aspect', AspectValue, ...) specifies the aspects to read from the Gene Ontology annotated file. AspectValue is a character vector or string specifying one or more characters. Valid aspects are:

- P — Biological process
- F — Molecular function
- C — Cellular component

Default is 'CFP', which specifies to read all aspects.

**Examples**

**Read Annotations from Gene Ontology Annotated File**

1. Download gene_association.sgd.gz, the file containing GO annotations for the gene products of *Saccharomyces cerevisiae*, from the yeast genome website to your MATLAB current folder.
2. Uncompress the file using the gunzip function.
   ```
   gunzip('gene_association.sgd.gz')
   ```
3. Load the file.
   ```
   SGDGenes = goannotread('gene_association.sgd');
   ```
4. Create a structure with GO annotations and display a list of the first five genes.
   ```
   S = struct2cell(SGDGenes);
   genes = S(3,1:5)'
   ```
genes =
    '15S_RRNA'
    '15S_RRNA'
    '15S_RRNA'
    '15S_RRNA'
    '21S_RRNA'

5 You can limit the annotations to genes related to molecular function (F) and to the fields for the gene symbol and the associated ID, that is, DB_Object_Symbol and GOid.

    sgdSelect = goannotread('gene_association.sgd','Aspect','F','Fields',{'DB_Object_Symbol','GOid'})
    sgdSelect =
        30701×1 struct array with fields:
        DB_Object_Symbol
        GOid

6 Create a list of genes and the associated GO terms.

    selectGenes = {sgdSelect.DB_Object_Symbol};
    selectGO = [sgdSelect.GOid];

See Also
geneont|geneont|getancestors|getdescendants|getmatrix|getrelatives|num2goid

Introduced before R2006a
**gonnet**

Return Gonnet scoring matrix

**Syntax**

gonnet

**Description**

gonnet returns the Gonnet matrix.

The Gonnet matrix is the recommended mutation matrix for initially aligning protein sequences. Matrix elements are ten times the logarithmic of the probability that the residues are aligned divided by the probability that the residues are aligned by chance, and then matrix elements are normalized to 250 PAM units.

Expected score = -0.6152, Entropy = 1.6845 bits, Lowest score = -8, Highest score = 14.2

Order:
A R N D C Q E G H I L K M F P S T W Y V B Z X *

**References**


**See Also**

blosum | dayhoff | localalign | nuc44 | nwalign | pam | swalign

**Introduced before R2006a**
gprread

Read microarray data from GenePix Results (GPR) file

Syntax

GPRData = gprread(File)

gprread(..., 'PropertyName', PropertyValue,...)
gprread(..., 'CleanColNames', CleanColNamesValue)

Arguments

<table>
<thead>
<tr>
<th>File</th>
<th>GenePix Results (GPR) formatted file. Enter a character vector or string specifying a file name, or a path and file name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CleanColNamesValue</td>
<td>Controls the creation of column names that can be used as variable names.</td>
</tr>
</tbody>
</table>

Description

GPRData = gprread(File) reads GenePix results data from File and creates a MATLAB structure (GPRData).

gprread(..., 'PropertyName', PropertyValue,...) defines optional properties using property name/value pairs.

gprread(..., 'CleanColNames', CleanColNamesValue) controls the creation of column names that can be used as variable names. A GPR file may contain column names with spaces and some characters that the MATLAB software cannot use in MATLAB variable names. If CleanColNamesValue is true, gprread returns names in the field ColumnNames that are valid MATLAB variable names and names that you can use in functions. By default, CleanColNamesValue is false and the field ColumnNames may contain characters that are invalid for MATLAB variable names.

The field Indices of the structure contains indices that can be used for plotting heat maps of the data.

For more details on the GPR format, see

https://mdc.custhelp.com/app/answers/detail/a_id/18883/kw/file%20format

The function supports versions 3, 4, and 5 of the GenePix Results Format.

Examples

Read and display data from GenePix® result (GPR) file

This example shows how to read and display data from a GenePix® result (GPR) file.
Read in a sample GPR file.

```matlab
gprStruct = gprread('mouse_alpdl.gpr')
```

```
gprStruct = struct with fields:
    Header: [1x1 struct]
    Data: [9504x38 double]
    Blocks: [9504x1 double]
    Columns: [9504x1 double]
    Rows: [9504x1 double]
    Names: {9504x1 cell}
    IDs: {9504x1 cell}
    ColumnNames: {38x1 cell}
    Indices: [132x72 double]
    Shape: [1x1 struct]
```

Plot the median foreground intensity for the 635 nm channel.

```matlab
maimage(gprStruct,'F635 Median')
```

See Also

affyread | agferead | celintensityread | galread | geoseriesread | geosoftread | ilmnbsread | imagenereserve | magetfield | sptread
Introduced before R2006a
graphallshortestpaths

Find all shortest paths in graph

Syntax

[dist] = graphallshortestpaths(G)

[dist] = graphallshortestpaths(G, ...'Directed', DirectedValue, ...)
[dist] = graphallshortestpaths(G, ...'Weights', WeightsValue, ...)

Arguments

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>N-by-N sparse matrix that represents a graph. Nonzero entries in matrix G represent the weights of the edges.</td>
</tr>
<tr>
<td>DirectedValue</td>
<td>Property that indicates whether the graph is directed or undirected. Enter false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.</td>
</tr>
<tr>
<td>WeightsValue</td>
<td>Column vector that specifies custom weights for the edges in matrix G. It must have one entry for every nonzero value (edge) in matrix G. The order of the custom weights in the vector must match the order of the nonzero values in matrix G when it is traversed column-wise. This property lets you use zero-valued weights. By default, graphallshortestpaths gets weight information from the nonzero entries in matrix G.</td>
</tr>
</tbody>
</table>

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions”.

[dist] = graphallshortestpaths(G) finds the shortest paths between every pair of nodes in the graph represented by matrix G, using Johnson's algorithm. Input G is an N-by-N sparse matrix that represents a graph. Nonzero entries in matrix G represent the weights of the edges.

Output dist is an N-by-N matrix where dist(S,T) is the distance of the shortest path from source node S to target node T. Elements in the diagonal of this matrix are always 0, indicating the source node and target node are the same. A 0 not in the diagonal indicates that the distance between the source node and target node is 0. An Inf indicates there is no path between the source node and the target node.

Johnson's algorithm has a time complexity of \(O(N\log(N)+N^2E)\), where \(N\) and \(E\) are the number of nodes and edges respectively.

[...] = graphallshortestpaths (G, 'PropertyName', PropertyValue, ...) calls graphallshortestpaths with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:
[dist] = graphallshortestpaths(G, ... 'Directed', DirectedValue, ...) indicates whether the graph is directed or undirected. Set DirectedValue to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.

[dist] = graphallshortestpaths(G, ... 'Weights', WeightsValue, ...) lets you specify custom weights for the edges. WeightsValue is a column vector having one entry for every nonzero value (edge) in matrix G. The order of the custom weights in the vector must match the order of the nonzero values in matrix G when it is traversed column-wise. This property lets you use zero-valued weights. By default, graphallshortestpaths gets weight information from the nonzero entries in matrix G.

Examples

Example 1.36. Finding All Shortest Paths in a Directed Graph

1 Create and view a directed graph with 6 nodes and 11 edges.

W = [.41 .99 .51 .32 .15 .45 .38 .32 .36 .29 .21];
DG = sparse([6 1 2 2 3 4 4 5 5 6 1], [2 6 3 5 4 1 6 3 4 3 5], W)

DG =

(4,1) 0.4500
(6,2) 0.4100
(2,3) 0.5100
(5,3) 0.3200
(6,3) 0.2900
(3,4) 0.1500
(5,4) 0.3600
(1,5) 0.2100
(2,5) 0.3200
(1,6) 0.9900
(4,6) 0.3800

view(biograph(DG,[],'ShowWeights','on'))
Find all the shortest paths between every pair of nodes in the directed graph.

\[ \text{graphallshortestpaths(DG)} \]

\[ \begin{array}{ccccccc}
0 & 1.3600 & 0.5300 & 0.5700 & 0.2100 & 0.9500 \\
1.1100 & 0 & 0.5100 & 0.6600 & 0.3200 & 1.0400 \\
0.6000 & 0.9400 & 0 & 0.1500 & 0.8100 & 0.5300 \\
0.4500 & 0.7900 & 0.6700 & 0 & 0.6600 & 0.3800 \\
0.8100 & 1.1500 & 0.3200 & 0.3600 & 0 & 0.7400 \\
0.8900 & 0.4100 & 0.2900 & 0.4400 & 0.7300 & 0 \\
\end{array} \]

The resulting matrix shows the shortest path from node 1 (first row) to node 6 (sixth column) is 0.95. You can see this in the graph by tracing the path from node 1 to node 5 to node 4 to node 6 (0.21 + 0.36 + 0.38 = 0.95).

**Example 1.37. Finding All Shortest Paths in an Undirected Graph**

Create and view an undirected graph with 6 nodes and 11 edges.

\[ \text{UG = tril(DG + DG')} \]

\[ \begin{array}{c}
(4,1) & 0.4500 \\
(5,1) & 0.2100 \\
(6,1) & 0.9900 \\
(3,2) & 0.5100 \\
\end{array} \]
2 Find all the shortest paths between every pair of nodes in the undirected graph.

graphallshortestpaths(UG,'directed',false)

ans =

     0    0.5300    0.5300    0.4500    0.2100    0.8300    0.5300         0
    0.5300         0    0.5100    0.6600    0.3200    0.7000    0.5100         0
    0.4500    0.6600    0.1500         0    0.3600    0.3800    0.3600    0
    0.2100    0.3200    0.3200    0.3600         0    0.7400    0.3200    0
    0.8300    0.7000    0.5300    0.3800    0.7400         0

The resulting matrix is symmetrical because it represents an undirected graph. It shows the shortest path from node 1 (first row) to node 6 (sixth column) is 0.83. You can see this in the graph by tracing the path from node 1 to node 4 to node 6 (0.45 + 0.38 = 0.83). Because UG is an undirected graph, we can use the edge between node 1 and node 4, which we could not do in the directed graph DG.
References


See Also
allshortestpaths | graphconncomp | graphisdag | graphisomorphism | graphisspantree |
graphmaxflow | graphminspantree | graphpred2path | graphshortestpath |
graphtopoorder | graphtraverse

Introduced in R2006b
graphconncomp

Find strongly or weakly connected components in graph

Syntax

[S, C] = graphconncomp(G)

[S, C] = graphconncomp(G, ...'Directed', DirectedValue, ...)
[S, C] = graphconncomp(G, ...'Weak', WeakValue, ...)

Arguments

<table>
<thead>
<tr>
<th>G</th>
<th>N-by-N sparse matrix that represents a graph. Nonzero entries in matrix G indicate the presence of an edge.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectedValue</td>
<td>Property that indicates whether the graph is directed or undirected. Enter false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.</td>
</tr>
<tr>
<td>WeakValue</td>
<td>Property that indicates whether to find weakly connected components or strongly connected components. A weakly connected component is a maximal group of nodes that are mutually reachable by violating the edge directions. Set WeakValue to true to find weakly connected components. Default is false, which finds strongly connected components. The state of this parameter has no effect on undirected graphs because weakly and strongly connected components are the same in undirected graphs. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.</td>
</tr>
</tbody>
</table>

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions”.

[S, C] = graphconncomp(G) finds the strongly connected components of the graph represented by matrix G using Tarjan’s algorithm. A strongly connected component is a maximal group of nodes that are mutually reachable without violating the edge directions. Input G is an N-by-N sparse matrix that represents a graph. Nonzero entries in matrix G indicate the presence of an edge.

The number of components found is returned in S, and C is a vector indicating to which component each node belongs.

Tarjan’s algorithm has a time complexity of \( O(N+E) \), where \( N \) and \( E \) are the number of nodes and edges respectively.
[S, C] = graphconncomp(G, ...'PropertyName', PropertyValue, ...) calls graphconncomp with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[S, C] = graphconncomp(G, ...'Directed', DirectedValue, ...) indicates whether the graph is directed or undirected. Set directedValue to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true. A DFS-based algorithm computes the connected components. Time complexity is \(O(N+E)\), where \(N\) and \(E\) are number of nodes and edges respectively.

[S, C] = graphconncomp(G, ...'Weak', WeakValue, ...) indicates whether to find weakly connected components or strongly connected components. A weakly connected component is a maximal group of nodes that are mutually reachable by violating the edge directions. Set WeakValue to true to find weakly connected components. Default is false, which finds strongly connected components. The state of this parameter has no effect on undirected graphs because weakly and strongly connected components are the same in undirected graphs. Time complexity is \(O(N+E)\), where \(N\) and \(E\) are number of nodes and edges respectively.

**Note** By definition, a single node can be a strongly connected component.

**Note** A directed acyclic graph (DAG) cannot have any strongly connected components larger than one.

**Examples**

1. Create and view a directed graph with 10 nodes and 17 edges.

   \[
   \text{DG} = \text{sparse([1 1 1 2 3 3 4 5 6 7 8 9 9 9 9 9], ..., [2 6 8 3 1 4 2 5 4 7 6 4 9 8 10 5 3],true,10,10)
   \]

   \[
   \text{DG} = \\
   (2,1) 1 \\
   (1,2) 1 \\
   (3,2) 1 \\
   (2,3) 1 \\
   (9,3) 1 \\
   (3,4) 1 \\
   (5,4) 1 \\
   (7,4) 1 \\
   (4,5) 1 \\
   (9,5) 1 \\
   (1,6) 1 \\
   (7,6) 1 \\
   (6,7) 1 \\
   (1,8) 1 \\
   (9,8) 1 \\
   (8,9) 1 \\
   (9,10) 1 \\
   \]

   \[
   \text{h = view(biograph(DG));}
   \]
Find the number of strongly connected components in the directed graph and determine to which component each of the 10 nodes belongs.

\[
[S,C] = \text{graphconncomp}(DG)
\]

\[
S =
\begin{bmatrix}
4 \\
4 \\
4 \\
4 \\
4 \\
4 \\
4 \\
4 \\
4 \\
3
\end{bmatrix}
\]

Color the nodes for each component with a different color.

\[
\text{colors} = \text{jet}(S);
\]

\[
\text{for } i = 1:\text{numel(h.nodes)}
\]

\[
\quad \text{h.Nodes(i).Color} = \text{colors}(C(i),:);
\]

\[
\text{end}
\]
References


See Also
conncomp | graphallshortestpaths | graphisdag | graphisomorphism | graphisspantree | graphmaxflow | graphminspantree | graphpred2path | graphshortestpath | graphtopoorder | graphtraverse
**graphisdag**

Test for cycles in directed graph

**Syntax**

```matlab
graphisdag(G)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix G indicate the presence of an edge.</td>
</tr>
</tbody>
</table>

**Description**

**Tip** For introductory information on graph theory functions, see “Graph Theory Functions”.

`graphisdag(G)` returns logical 1 (`true`) if the directed graph represented by matrix `G` is a directed acyclic graph (DAG) and logical 0 (`false`) otherwise. `G` is an N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix `G` indicate the presence of an edge.

**Examples**

**Example 1.38. Testing for Cycles in Directed Graphs**

1. Create and view a directed acyclic graph (DAG) with six nodes and eight edges.

```matlab
DG = sparse([1 1 1 2 2 3 4 6],[2 4 6 3 5 4 6 5],true,6,6)
```

```
DG =
    (1,2)    1
    (2,3)    1
    (1,4)    1
    (3,4)    1
    (2,5)    1
    (6,5)    1
    (1,6)    1
    (4,6)    1
```

```matlab
view(biograph(DG))
```

2 Test for cycles in the DAG.

```matlab
graphisdag(DG)
```

```matlab
an = 1
```

3 Add an edge to the DAG to make it cyclic, and then view the directed graph.

```matlab
DG(5,1) = true
```

```matlab
DG =
```

```matlab
(5,1) 1
(1,2) 1
(2,3) 1
(1,4) 1
(3,4) 1
(2,5) 1
(6,5) 1
(1,6) 1
(4,6) 1
```

```matlab
view(biograph(DG))
```
Example 1.39. Testing for Cycles in a Very Large Graph (Greater Than 20,000 Nodes and 30,000 Edges)

1. Download the Gene Ontology database to a geneont object.
   
   ```matlab
   GO = geneont('live',true);
   ```

2. Convert the geneont object to a matrix.

   ```matlab
   CM = getmatrix(GO);
   ```

3. Test for cycles in the graph.

   ```matlab
   graphisdag(CM)
   ```

Example 1.40. Creating a Random DAG

1. Create and view a random directed acyclic graph (DAG) with 15 nodes and 20 edges.

   ```matlab
   g = sparse([],[],true,15,15);
   while nnz(g) < 20
       edge = randsample(15*15,1); % get a random edge
   ```
g(edge) = true;
g(edge) = graphisdag(g);
end
view(biograph(g))

Test for cycles in the graph.

graphisdag(g)

References


See Also

graphallshortestpaths | graphconncomp | graphisomorphism | graphissspantree |
graphmaxflow | graphminspantree | graphpred2path | graphshortestpath |
graphtraverse | isdag

Introduced in R2006b
graphisomorphism

Find isomorphism between two graphs

Syntax

[\texttt{Isomorphic, Map}] = \texttt{graphisomorphism}(\texttt{G1, G2})
[\texttt{Isomorphic, Map}] = \texttt{graphisomorphism}(\texttt{G1, G2,'Directed', DirectedValue})

Arguments

| \texttt{G1}  | N-by-N sparse matrix that represents a directed or undirected graph. Nonzero entries in matrix \texttt{G1} indicate the presence of an edge. |
| \texttt{G2}  | N-by-N sparse matrix that represents a directed or undirected graph. \texttt{G2} must be the same (directed or undirected) as \texttt{G1}. |
| \texttt{DirectedValue} | Property that indicates whether the graphs are directed or undirected. Enter \texttt{false} when both \texttt{G1} and \texttt{G2} are undirected graphs. In this case, the upper triangles of the sparse matrices \texttt{G1} and \texttt{G2} are ignored. Default is \texttt{true}, meaning that both graphs are directed. |

Description

\textbf{Tip} For introductory information on graph theory functions, see “Graph Theory Functions”.

[\texttt{Isomorphic, Map}] = \texttt{graphisomorphism}(\texttt{G1, G2}) returns logical 1 (true) in \texttt{Isomorphic} if \texttt{G1} and \texttt{G2} are isomorphic graphs, and logical 0 (false) otherwise. A graph isomorphism is a 1-to-1 mapping of the nodes in the graph \texttt{G1} and the nodes in the graph \texttt{G2} such that adjacencies are preserved. \texttt{G1} and \texttt{G2} are both N-by-N sparse matrices that represent directed or undirected graphs. Return value \texttt{Isomorphic} is Boolean. When \texttt{Isomorphic} is \texttt{true}, \texttt{Map} is a row vector containing the node indices that map from \texttt{G2} to \texttt{G1} for one possible isomorphism. When \texttt{Isomorphic} is \texttt{false}, \texttt{Map} is empty. The worst-case time complexity is \texttt{O(N!)}), where \texttt{N} is the number of nodes.

[\texttt{Isomorphic, Map}] = \texttt{graphisomorphism}(\texttt{G1, G2,'Directed', DirectedValue}) indicates whether the graphs are directed or undirected. Set \texttt{DirectedValue} to \texttt{false} when both \texttt{G1} and \texttt{G2} are undirected graphs. In this case, the upper triangles of the sparse matrices \texttt{G1} and \texttt{G2} are ignored. Default is \texttt{true}, meaning that both graphs are directed.

Examples

1 Create and view a directed graph with 8 nodes and 11 edges.

\begin{verbatim}
m('ABCDEFGH') = [1 2 3 4 5 6 7 8];
g1 = sparse(m('ABDCDCGEFFG'),m('BCBDGEEFHGH'),true,8,8)
g1 =
\end{verbatim}

\begin{verbatim}
 (1,2) 1
\end{verbatim}
Set a random permutation vector and then create and view a new permuted graph.

```matlab
p = randperm(8)
p =
   7  8  2  3  6  4  1  5
g2 = g1(p,p);
view(biograph(g2,'ABCDEFGH'))
```
3 Check if the two graphs are isomorphic.

\[ [F, \text{Map}] = \text{graphisomorphism}(g2, g1) \]

\[ F = \]

\[ 1 \]

\[ \text{Map} = \]

\[ 7 \ 8 \ 2 \ 3 \ 6 \ 4 \ 1 \ 5 \]

Note that the Map row vector containing the node indices that map from g2 to g1 is the same as the permutation vector you created in step 2.

4 Reverse the direction of the D-G edge in the first graph, and then check for isomorphism again.

\[ g1(m(\text{'DG'}), m(\text{'GD'})) = g1(m(\text{'GD'}), m(\text{'DG'})); \]

\[ \text{view(biograph}(g1, \text{'ABCDEFGH'})) \]
Convert the graphs to undirected graphs, and then check for isomorphism.

```matlab
[F,M] = graphisomorphism(g2+g2',g1+g1','directed',false)
F =
    1
M =
    7    8    2    3    6    4    1    5
```
References


See Also

graphallshortestpaths | graphconncomp | graphisdag | graphisspantree | graphmaxflow | graphminspantree | graphpred2path | graphshortestpath | graphtopoorder | graphtraverse | isomorphism

Introduced in R2006b
graphisspantree

Determine if tree is spanning tree

Syntax

\[ TF = \text{graphisspantree}(G) \]

Arguments

- \( G \): N-by-N sparse matrix whose lower triangle represents an undirected graph. Nonzero entries in matrix \( G \) indicate the presence of an edge.

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions”.

\[ TF = \text{graphisspantree}(G) \] returns logical 1 (true) if \( G \) is a spanning tree, and logical 0 (false) otherwise. A spanning tree must touch all the nodes and must be acyclic. \( G \) is an N-by-N sparse matrix whose lower triangle represents an undirected graph. Nonzero entries in matrix \( G \) indicate the presence of an edge.

Examples

1. Create a phytree object from a phylogenetic tree file.
   \[
   tr = \text{phytreeread('pf00002.tree')}
   \]
   Phylogenetic tree object with 33 leaves (32 branches)

2. Create a connection matrix from the phytree object.
   \[
   [\text{CM,labels,dist}] = \text{getmatrix}(tr);
   \]

3. Determine if the connection matrix is a spanning tree.
   \[
   \text{graphisspantree(CM)}
   \]
   \[
   \text{ans} =
   \]
   \[
   1
   \]

4. Add an edge between the root and the first leaf in the connection matrix.
   \[
   \text{CM(end,1)} = 1;
   \]

5. Determine if the modified connection matrix is a spanning tree.
   \[
   \text{graphisspantree(CM)}
   \]
   \[
   \text{ans} =
   \]
   \[
   0
References


See Also

graphallshortestpaths | graphconncomp | graphisdag | graphisomorphism |
graphmaxflow | graphminspantree | graphpred2path | graphshortestpath |
graphtopoorder | graphtraverse | issentree

Introduced in R2006b
graphmaxflow

Calculate maximum flow in directed graph

Syntax


[...] = graphmaxflow(G, SNode, TNode, ...'Capacity', CapacityValue, ...)
[...] = graphmaxflow(G, SNode, TNode, ...'Method', MethodValue, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix G represent the capacities of the edges.</td>
</tr>
<tr>
<td>SNode</td>
<td>Node in G.</td>
</tr>
<tr>
<td>TNode</td>
<td>Node in G.</td>
</tr>
<tr>
<td>CapacityValue</td>
<td>Column vector that specifies custom capacities for the edges in matrix G. It must have one entry for every nonzero value (edge) in matrix G. The order of the custom capacities in the vector must match the order of the nonzero values in matrix G when it is traversed column-wise. By default, graphmaxflow gets capacity information from the nonzero entries in matrix G.</td>
</tr>
<tr>
<td>MethodValue</td>
<td>Character vector or string that specifies the algorithm used to find the minimal spanning tree (MST). Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'Edmonds' — Uses the Edmonds and Karp algorithm, the implementation of which is based on a variation called the labeling algorithm. Time complexity is (O(N^3E^2)), where N and E are the number of nodes and edges respectively.</td>
</tr>
<tr>
<td></td>
<td>• 'Goldberg' — Default algorithm. Uses the Goldberg algorithm, which uses the generic method known as preflow-push. Time complexity is (O(N^2\sqrt{E})), where N and E are the number of nodes and edges respectively.</td>
</tr>
</tbody>
</table>

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions”.

[MaxFlow, FlowMatrix, Cut] = graphmaxflow(G, SNode, TNode) calculates the maximum flow of directed graph G from node SNode to node TNode. Input G is an N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix G represent the capacities of the edges. Output MaxFlow is the maximum flow, and FlowMatrix is a sparse matrix with all the flow values for every edge. FlowMatrix(X,Y) is the flow from node X to node Y. Output Cut is a logical row vector indicating the nodes connected to SNode after calculating the minimum cut between SNode and TNode. If several solutions to the minimum cut problem exist, then Cut is a matrix.
**Tip** The algorithm that determines Cut, all minimum cuts, has a time complexity of $O(2^N)$, where $N$ is the number of nodes. If this information is not needed, use the `graphmaxflow` function without the third output.

`[...] = graphmaxflow(G, SNode, TNode, ...'PropertyName', PropertyValue, ...)` calls `graphmaxflow` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

`[...] = graphmaxflow(G, SNode, TNode, ...'Capacity', CapacityValue, ...)` lets you specify custom capacities for the edges. `CapacityValue` is a column vector having one entry for every nonzero value (edge) in matrix $G$. The order of the custom capacities in the vector must match the order of the nonzero values in matrix $G$ when it is traversed column-wise. By default, `graphmaxflow` gets capacity information from the nonzero entries in matrix $G$.

`[...] = graphmaxflow(G, SNode, TNode, ...'Method', MethodValue, ...)` lets you specify the algorithm used to find the minimal spanning tree (MST). Choices are:

- 'Edmonds' — Uses the Edmonds and Karp algorithm, the implementation of which is based on a variation called the *labeling algorithm*. Time complexity is $O(N^*E^2)$, where $N$ and $E$ are the number of nodes and edges respectively.
- 'Goldberg' — Default algorithm. Uses the Goldberg algorithm, which uses the generic method known as *preflow-push*. Time complexity is $O(N^2*sqrt(E))$, where $N$ and $E$ are the number of nodes and edges respectively.

**Examples**

**Calculate Maximum Flow**

This example shows how to calculate the maximum flow in a directed graph.

Create a directed graph with six nodes and eight edges.

```matlab
cm = sparse([1 1 2 2 3 3 4 5],[2 3 4 5 4 5 6 6],...
             [2 3 3 1 1 2 3],6,6)
```

```matlab
cm =
(1,2)  2
(1,3)  3
(2,4)  3
(3,4)  1
(2,5)  1
(3,5)  1
(4,6)  2
(5,6)  3
```

Calculate the maximum flow in the graph from node 1 to node 6.

```matlab
[M,F,K] = graphmaxflow(cm,1,6)
```

$M = 4$
\[ F = \]
\[
(1,2) \quad 2 \\
(1,3) \quad 2 \\
(2,4) \quad 1 \\
(3,4) \quad 1 \\
(2,5) \quad 1 \\
(3,5) \quad 1 \\
(4,6) \quad 2 \\
(5,6) \quad 2 \\
\]

\[ K = 2x6 \text{ logical array} \]
\[
1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \\
1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \\
\]

Notice that \( K \) is a two-row matrix because there are two possible solutions to the minimum cut problem.

View the graph with the original capacities.

\[ h = \text{view(biograph(cm,[],'ShowWeights','on'))} \]
Biograph object with 6 nodes and 8 edges.

View the graph with the calculated maximum flows.

view(biograph(F,[],'ShowWeights','on'))
Show one solution to the minimum cut problem in the original graph.

```matlab
set(h.Nodes(K(1,:)), 'Color', [1 0 0])
```
References


See Also

graphallshortestpaths | graphconncomp | graphisdag | graphisomorphism | graphisspantree | graphminsptree | graphpred2path | graphshortestpath | graphtopoorder | graphtraverse | maxflow
Introduced in R2006b
graphminspantree

Find minimal spanning tree in graph

Syntax

[Tree, pred] = graphminspantree(G)
[Tree, pred] = graphminspantree(G, R)

[Tree, pred] = graphminspantree(..., 'Method', MethodValue, ...)
[Tree, pred] = graphminspantree(..., 'Weights', WeightsValue, ...)

Arguments

<table>
<thead>
<tr>
<th>G</th>
<th>N-by-N sparse matrix that represents an undirected graph. Nonzero entries in matrix G represent the weights of the edges.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Scalar between 1 and the number of nodes.</td>
</tr>
</tbody>
</table>

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions”.

[Tree, pred] = graphminspantree(G) finds an acyclic subset of edges that connects all the nodes in the undirected graph G and for which the total weight is minimized. Weights of the edges are all nonzero entries in the lower triangle of the N-by-N sparse matrix G. Output Tree is a spanning tree represented by a sparse matrix. Output pred is a vector containing the predecessor nodes of the minimal spanning tree (MST), with the root node indicated by 0. The root node defaults to the first node in the largest connected component. This computation requires an extra call to the graphconncomp function.

[Tree, pred] = graphminspantree(G, R) sets the root of the minimal spanning tree to node R.

[Tree, pred] = graphminspantree(..., 'PropertyName', PropertyValue, ...) calls graphminspantree with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[Tree, pred] = graphminspantree(..., 'Method', MethodValue, ...) lets you specify the algorithm used to find the minimal spanning tree (MST). Choices are:

- 'Kruskal' — Grows the minimal spanning tree (MST) one edge at a time by finding an edge that connects two trees in a spreading forest of growing MSTs. Time complexity is $O(E+X\times \log(N))$, where X is the number of edges no longer than the longest edge in the MST, and N and E are the number of nodes and edges respectively.
- 'Prim' — Default algorithm. Grows the minimal spanning tree (MST) one edge at a time by adding a minimal edge that connects a node in the growing MST with any other node. Time complexity is $O(E \times \log(N))$, where N and E are the number of nodes and edges respectively.
Note When the graph is unconnected, Prim’s algorithm returns only the tree that contains R, while Kruskal’s algorithm returns an MST for every component.

\([\text{Tree, pred}] = \text{graphminspantree}(\ldots, \text{’Weights’}, \text{WeightsValue}, \ldots)\) lets you specify custom weights for the edges. \text{WeightsValue} is a column vector having one entry for every nonzero value (edge) in matrix \(G\). The order of the custom weights in the vector must match the order of the nonzero values in matrix \(G\) when it is traversed column-wise. By default, \text{graphminspantree} gets weight information from the nonzero entries in matrix \(G\).

Examples

1 Create and view an undirected graph with 6 nodes and 11 edges.

\[
W = [.41 .29 .51 .32 .50 .45 .38 .32 .36 .29 .21];
DG = \text{sparse([1 1 2 2 3 4 4 5 5 6 6], [2 6 3 5 4 1 6 3 4 2 5], W)};
UG = \text{tril(DG + DG')} \quad \text{UG =}
\begin{align*}
(2,1) & \quad 0.4100 \\
(4,1) & \quad 0.4500 \\
(6,1) & \quad 0.2900 \\
(3,2) & \quad 0.5100 \\
(5,2) & \quad 0.3200 \\
(6,2) & \quad 0.2900 \\
(4,3) & \quad 0.5000 \\
(5,3) & \quad 0.3200 \\
(5,4) & \quad 0.3600 \\
(6,4) & \quad 0.3800 \\
(6,5) & \quad 0.2100
\end{align*}
view(\text{biograph(UG, [],’ShowArrows’,’off’,’ShowWeights’,’on’)})
Find and view the minimal spanning tree of the undirected graph.

```matlab
[ST,pred] = graphminspantree(UG)
```

```
ST =
(6,1) 0.2900
(6,2) 0.2900
(5,3) 0.3200
(5,4) 0.3600
(6,5) 0.2100

pred =
0 6 5 5 6 1
```

```matlab
view(biograph(ST,[],'ShowArrows','off','ShowWeights','on'))
```
References


See Also
graphallshortestpaths | graphconncomp | graphisdag | graphisomorphism | graphisspantree | graphmaxflow | graphpred2path | graphshortestpath | graphtopoorder | graphtraverse | minspantree

Introduced in R2006b
graphpred2path

Convert predecessor indices to paths

Syntax

path = graphpred2path(pred, D)

Arguments

<table>
<thead>
<tr>
<th>pred</th>
<th>Row vector or matrix of predecessor node indices. The value of the root (or source) node in pred must be 0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Destination node in pred.</td>
</tr>
</tbody>
</table>

Description

*Tip* For introductory information on graph theory functions, see “Graph Theory Functions”.

path = graphpred2path(pred, D) traces back a path by following the predecessor list in pred starting at destination node D.

The value of the root (or source) node in pred must be 0. If a NaN is found when following the predecessor nodes, graphpred2path returns an empty path.

<table>
<thead>
<tr>
<th>If <em>pred</em> is a ...</th>
<th>And <em>D</em> is a ...</th>
<th>Then <em>path</em> is a ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>row vector of predecessor node indices</td>
<td>scalar</td>
<td>row vector listing the nodes from the root (or source) to D.</td>
</tr>
<tr>
<td></td>
<td>row vector</td>
<td>row cell array with every column containing the path to the destination for every element in D.</td>
</tr>
<tr>
<td>matrix</td>
<td>scalar</td>
<td>column cell array with every row containing the path for every row in pred.</td>
</tr>
<tr>
<td></td>
<td>row vector</td>
<td>matrix cell array with every row containing the paths for the respective row in pred, and every column containing the paths to the respective destination in D.</td>
</tr>
</tbody>
</table>

*Note* If D is omitted, the paths to all the destinations are calculated for every predecessor listed in pred.

Examples

1. Create a phytree object from the phylogenetic tree file for the GLR_HUMAN protein.

   tr = phytreeread('pf00002.tree')
   Phylogenetic tree object with 33 leaves (32 branches)
2 View the phytree object.

```matlab
view(tr)
```

![Phylogenetic Tree Tool](image)

3 From the phytree object, create a connection matrix to represent the phylogenetic tree.

```matlab
[CM,labels,dist] = getmatrix(tr);
```

4 Find the nodes from the root to one leaf in the phylogenetic tree created from the phylogenetic tree file for the GLR_HUMAN protein.

```matlab
root_loc = size(CM,1)

root_loc =

65

glr_loc = strncmp('GLR',labels,3);
glr_loc_ind = find(glr_loc)

glr_loc_ind =

12

[T,PRED]=graphminspantree(CM,root_loc);
PATH = graphpred2path(PRED,glr_loc_ind)

PATH =

65 64 53 52 46 45 44 43 12
References


See Also

graphallshortestpaths | graphconncomp | graphisdag | graphisomorphism |
graphisspantree | graphmaxflow | graphminspantree | graphshortestpath |
graphtopoorder | graphtraverse

Introduced in R2006b
graphshortestpath

Solve shortest path problem in graph

Syntax

[dist,path,pred] = graphshortestpath(G,S)
[___] = graphshortestpath(G,S,D)
[___] = graphshortestpath(___,Name,Value)

Description

[dist,path,pred] = graphshortestpath(G,S) determines the shortest paths from the source node S to all other nodes in the graph G. dist contains the distances from the source node to all other nodes. path contains the shortest paths to every node. pred contains the predecessor nodes of the shortest paths.

[___] = graphshortestpath(G,S,D) determines the shortest path from the source node S to the target node D and returns any of the output arguments from the previous syntax.

[___] = graphshortestpath(___,Name,Value) specifies additional options using one or more name-value pair arguments. Specify name-value pair arguments after any of the input argument combinations in the previous syntaxes.

Examples

Find Shortest Paths in Directed Graph

Create a directed graph with 6 nodes and 11 edges.

W = [.41 .99 .51 .32 .15 .45 .38 .32 .36 .29 .21];
DG = sparse([6 1 2 2 3 4 4 5 5 6 1],[2 6 3 5 4 1 6 3 4 3 5],W)
DG =
(4,1) 0.4500
(6,2) 0.4100
(2,3) 0.5100
(5,3) 0.3200
(6,3) 0.2900
(3,4) 0.1500
(5,4) 0.3600
(1,5) 0.2100
(2,5) 0.3200
(1,6) 0.9900
(4,6) 0.3800

Display the graph.

h = view(biograph(DG,[],'ShowWeights','on'))
Biograph object with 6 nodes and 11 edges.

Find the shortest path from node 1 to node 6.

\[
\text{[dist, path, pred] = graphshortestpath(DG,1,6)}
\]

- \( \text{dist} = 0.9500 \)
- \( \text{path} = 1 \times 4 \)
  
  \[
  \begin{bmatrix}
  1 \\
  5 \\
  4 \\
  6 
  \end{bmatrix}
  \]

- \( \text{pred} = 1 \times 6 \)
  
  \[
  \begin{bmatrix}
  0 \\
  6 \\
  5 \\
  5 \\
  1 \\
  4 
  \end{bmatrix}
  \]

Mark the nodes and edges of the shortest path by coloring them red and increasing the line width.

\[
\text{set(h.Nodes(path),'Color',[1 0.4 0.4])}
\]

\[
\text{edges = getedgesbynodeid(h,get(h.Nodes(path),'ID'))};
\]
pred contains predecessor nodes of the shortest paths from node 1, the source node, to all other nodes, not only the specified destination node. You can use pred to query the shortest paths from the source node to any other node in the graph.

For instance, to figure out the shortest path from node 1 to node 4 using the information in pred, query pred with the destination node as the first query. Then use the returned answer to get the next node. Repeat this procedure until you get the query answer as 0, which indicates the source node.

```matlab
next = pred(4)
next = 5
next = pred(next)
next = 1
next = pred(next)
next = 0
```
The results indicate that the shortest path from node 1 to node 4 is 1->5->4.

**Find Shortest Paths in Undirected Graph**

Create an undirected graph with 6 nodes and 11 edges.

\[
W = [.41 .99 .51 .32 .15 .45 .38 .32 .36 .29 .21];
\]

\[
DG = \text{sparse([6 1 2 3 4 5 5 6 1],[2 6 3 5 4 1 6 3 4 3 5],W);
\]

% tril returns the lower triangular part of the matrix.

\[
UG = \text{tril}(DG+DG')
\]

\[
UG =
\begin{array}{ccc}
(4,1) & 0.4500 \\
(5,1) & 0.2100 \\
(6,1) & 0.9900 \\
(3,2) & 0.5100 \\
(5,2) & 0.3200 \\
(6,2) & 0.4100 \\
(4,3) & 0.1500 \\
(5,3) & 0.3200 \\
(6,3) & 0.2900 \\
(5,4) & 0.3600 \\
(6,4) & 0.3800 \\
\end{array}
\]

View the graph.

\[
h = \text{view(biograph(UG,[],'ShowArrows','off','ShowWeights','on'))}
\]
Biograph object with 6 nodes and 11 edges.

Find the shortest path from node 1 to node 6. Set 'Directed' to false to specify that the graph is not a directed graph.

```
[dist,path,pred] = graphshortestpath(UG,1,6,'Directed',false)
dist = 0.8200
path = 1×4
      1     5     3     6
pred = 1×6
      0     5     5     1     1     3
```

Mark the nodes and edges of the shortest path by coloring them red and increasing the line width.

```
set(h.Nodes(path),'Color',[1 0.4 0.4])
fowEdges = getedgesbynameid(h,get(h.Nodes(path),'ID'));
```
revEdges = getedgesbynodeid(h,get(h.Nodes(fliplr(path)),'ID'));
edges = [fowEdges;revEdges];
set(edges,'LineColor',[1 0 0])
set(edges,'LineWidth',1.5)

---

**Input Arguments**

**G — Adjacency matrix**

matrix

Adjacency matrix, specified as an N-by-N sparse matrix that represents a graph. Nonzero entries in the matrix represent the weights of the edges.

Data Types: double

**S — Source node**

numeric node index

Source node, specified as a numeric node index.
Example: 2
Data Types: double

D — Destination node
numeric node index

Destination node, specified as a numeric node index.
Example: 5
Data Types: double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: [dist,path,pred] = graphshortestpath(G,1,5,'Method','Acyclic') assumes G to be a directed acyclic graph when finding the shortest path from node 1 to node 5.

Method — Shortest path algorithm
'Dijkstra' (default) | 'Bellman-Ford' | 'BFS' | 'Acyclic'

Shortest path algorithm, specified as the comma-separated pair consisting of 'Method' and one of these options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Dijkstra' (default)</td>
<td>This algorithm assumes that all edge weights are positive values in G. The time complexity is (O(\log(N)E)), where (N) and (E) are the number of nodes and the number of edges respectively.</td>
</tr>
<tr>
<td>'BFS'</td>
<td>This breadth-first search algorithm assumes that all weights are equal and that edges are nonzero entries in the sparse matrix G. The time complexity is (O(N+E)).</td>
</tr>
<tr>
<td>'Bellman-Ford'</td>
<td>This algorithm assumes that all edge weights are nonzero entries in G. The time complexity is (O(N^2)).</td>
</tr>
<tr>
<td>'Acyclic'</td>
<td>This algorithm assumes that G is a directed acyclic graph and all edge weights are nonzero entries in G. The time complexity is (O(N+E)).</td>
</tr>
</tbody>
</table>

Example: 'Method','Acyclic'
Data Types: char | string

Directed — Directed or undirected graph flag
ture (default) | false

Directed or undirected graph flag, specified as a comma-separated pair consisting of 'Directed' and true or false. If false, the function ignores the upper triangle of the sparse matrix G.
Example: 'Directed',false
Data Types: logical

Weights — Custom weights for edges in G
column vector

Custom weights for edges in the matrix G, specified as a comma-separated pair consisting of 'Weights' and a column vector. The vector must meet the following conditions.

• It must have one entry for every nonzero value (edge) in the matrix G.
• The order of the custom weights in the vector must match the order of the nonzero values in G when it is traversed columnwise.

You can specify zero-valued weights. By default, the function obtains the weight information from the nonzero entries in G.

Example: 'Weights',[1 2.3 1.3 0 4]

Output Arguments

dist — Distances from source node to all other nodes in graph
numeric scalar | numeric vector

Distances from the source node to all other nodes in the graph, returned as a numeric scalar or vector. dist is returned as a scalar if you specify a destination node as the third input argument.

The function returns Inf for nonreachable nodes and 0 for the source node.

path — Shortest paths from source node to all other nodes
vector | cell array

Shortest paths from the source node to all other nodes, returned as a vector or cell array. It is returned as a vector if you specify a destination node. Each number represents a node index in the graph.

pred — Predecessor nodes of shortest paths
vector

Predecessor nodes of the shortest paths, returned as a vector.

You can use pred to determine the shortest paths from the source node to all other nodes. Suppose that you have a directed graph with 6 nodes.
The function finds that the shortest path from node 1 to node 6 is \( \text{path} = [1, 5, 4, 6] \) and \( \text{pred} = [0, 6, 5, 5, 1, 4] \). Now you can determine the shortest paths from node 1 to any other node within the graph by indexing into \( \text{pred} \). For example, to figure out the shortest path from node 1 to node 2, you can query \( \text{pred} \) with the destination node as the first query, then use the returned answer to get the next node. Repeat this procedure until the query answer is 0, which indicates the source node.

\[
\text{pred}(2) = 6; \quad \text{pred}(6) = 4; \quad \text{pred}(4) = 5; \quad \text{pred}(5) = 1; \quad \text{pred}(1) = 0;
\]

The results indicate that the shortest path from node 1 to node 2 is 1->5->4->6->2.

**References**


See Also

graphallshortestpaths | graphconncomp | graphisdag | graphisomorphism |
graphisspantree | graphmaxflow | graphminspantree | graphpred2path | graphtopoorder |
graphtraverse | shortestpath

Topics

“Graph Theory Functions”

Introduced in R2006b
**graphtopoorder**

Perform topological sort of directed acyclic graph

**Syntax**

```plaintext
order = graphtopoorder(G)
```

**Arguments**

- **G**
  - N-by-N sparse matrix that represents a directed acyclic graph. Nonzero entries in matrix `G` indicate the presence of an edge.

**Description**

**Tip** For introductory information on graph theory functions, see “Graph Theory Functions”.

`order = graphtopoorder(G)` returns an index vector with the order of the nodes sorted topologically. In topological order, an edge can exist between a source node `u` and a destination node `v`, if and only if `u` appears before `v` in the vector `order`. `G` is an N-by-N sparse matrix that represents a directed acyclic graph (DAG). Nonzero entries in matrix `G` indicate the presence of an edge.

**Examples**

1. Create and view a directed acyclic graph (DAG) with six nodes and eight edges.

```plaintext
DG = sparse([6 6 6 2 2 3 5 1], [2 5 1 3 4 5 1 4], true, 6, 6)
view(biograph(DG))
```

```plaintext
DG =
    (5,1)   1
    (6,1)   1
    (6,2)   1
    (2,3)   1
    (1,4)   1
    (2,4)   1
    (3,5)   1
    (6,5)   1
view(biograph(DG))
```
2 Find the topological order of the DAG.

order = graphtopoorder(DG)

order =

6 2 3 5 1 4

3 Permute the nodes so that they appear ordered in the graph display.

DG = DG(order,order)

DG =

(1,2) 1
(2,3) 1
(1,4) 1
(3,4) 1
(1,5) 1
(4,5) 1
(2,6) 1
(5,6) 1

view(biograph(DG))
References


See Also
graphallshortestpaths | graphconncomp | graphisdag | graphisomorphism | graphisspantree | graphmaxflow | graphminspantree | graphpred2path | graphshortestpath | graphtraverse | topoorder

Introduced in R2006b
graphtraverse

Traverse graph by following adjacent nodes

Syntax

\[
[\text{disc}, \text{pred}, \text{closed}] = \text{graphtraverse}(\text{G}, \text{S})
\]

\[
[... ...] = \text{graphtraverse}(\text{G}, \text{S}, ...'\text{Depth}', \text{DepthValue}, ...)
\]

\[
[... ...] = \text{graphtraverse}(\text{G}, \text{S}, ...'\text{Directed}', \text{DirectedValue}, ...)
\]

\[
[... ...] = \text{graphtraverse}(\text{G}, \text{S}, ...'\text{Method}', \text{MethodValue}, ...)
\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(G)</td>
<td>N-by-N sparse matrix that represents a directed graph. Nonzero entries in</td>
</tr>
<tr>
<td></td>
<td>matrix (G) indicate the presence of an edge.</td>
</tr>
<tr>
<td>(S)</td>
<td>Integer that indicates the source node in graph (G).</td>
</tr>
<tr>
<td>(\text{DepthValue})</td>
<td>Integer that indicates a node in graph (G) that specifies the depth of</td>
</tr>
<tr>
<td></td>
<td>the search. Default is (\text{Inf}) (infinity).</td>
</tr>
<tr>
<td>(\text{DirectedValue})</td>
<td>Property that indicates whether graph (G) is directed or undirected.</td>
</tr>
<tr>
<td></td>
<td>Enter (\text{false}) for an undirected graph. This results in the upper</td>
</tr>
<tr>
<td></td>
<td>triangle of the sparse matrix being ignored. Default is (\text{true}).</td>
</tr>
<tr>
<td>(\text{MethodValue})</td>
<td>Character vector or string that specifies the algorithm used to traverse</td>
</tr>
<tr>
<td></td>
<td>the graph. Choices are:</td>
</tr>
<tr>
<td></td>
<td>'BFS' — Breadth-first search. Time complexity is (O(N+E)), where (N) and</td>
</tr>
<tr>
<td></td>
<td>(E) are number of nodes and edges respectively.</td>
</tr>
<tr>
<td></td>
<td>'DFS' — Default algorithm. Depth-first search. Time complexity is (O(N+E)),</td>
</tr>
<tr>
<td></td>
<td>where (N) and (E) are number of nodes and edges respectively.</td>
</tr>
</tbody>
</table>

Description

Tip  For introductory information on graph theory functions, see “Graph Theory Functions”.

\([\text{disc}, \text{pred}, \text{closed}] = \text{graphtraverse}(\text{G}, \text{S})\) traverses graph \(G\) starting from the node indicated by integer \(S\). \(G\) is an N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix \(G\) indicate the presence of an edge. \(\text{disc}\) is a vector of node indices in the order in which they are discovered. \(\text{pred}\) is a vector of predecessor node indices (listed in the order of the node indices) of the resulting spanning tree. \(\text{closed}\) is a vector of node indices in the order in which they are closed.

\([... ...] = \text{graphtraverse}(\text{G}, \text{S}, ...'\text{PropertyName}', \text{PropertyValue}, ...)\) calls \(\text{graphtraverse}\) with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each \(\text{PropertyName}\) must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:
[...] = graphtraverse(G, S, ...'Depth', DepthValue, ...) specifies the depth of the search. DepthValue is an integer indicating a node in graph G. Default is Inf (infinity).

[...] = graphtraverse(G, S, ...'Directed', DirectedValue, ...) indicates whether the graph is directed or undirected. Set DirectedValue to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.

[...] = graphtraverse(G, S, ...'Method', MethodValue, ...) lets you specify the algorithm used to traverse the graph. Choices are:

• 'BFS' — Breadth-first search. Time complexity is \(O(N+E)\), where \(N\) and \(E\) are number of nodes and edges respectively.

• 'DFS' — Default algorithm. Depth-first search. Time complexity is \(O(N+E)\), where \(N\) and \(E\) are number of nodes and edges respectively.

**Examples**

1. Create a directed graph with 10 nodes and 12 edges.

   \[
   DG = \text{sparse}([1 \ 2 \ 3 \ 4 \ 5 \ 5 \ 6 \ 7 \ 8 \ 8 \ 9],[2 \ 4 \ 1 \ 5 \ 3 \ 6 \ 7 \ 9 \ 8 \ 1 \ 10 \ 2],\text{true},10,10)
   \]

   \[
   DG =
   \begin{array}{ccc}
   (3,1) & 1 \\
   (8,1) & 1 \\
   (1,2) & 1 \\
   (9,2) & 1 \\
   (5,3) & 1 \\
   (2,4) & 1 \\
   (4,5) & 1 \\
   (5,6) & 1 \\
   (5,7) & 1 \\
   (7,8) & 1 \\
   (6,9) & 1 \\
   (8,10) & 1 \\
   \end{array}
   \]

   \[
h = \text{view(biograph(DG))}
   \]

   Biograph object with 10 nodes and 12 edges.
Traverse the graph to find the depth-first search (DFS) discovery order starting at node 4.

```matlab
order = graphtraverse(DG,4)
```

```matlab
order =

4     5     3     1     2     6     9     7     8    10
```

Label the nodes with the DFS discovery order:

```matlab
for i = 1:10
    h.Nodes(order(i)).Label = sprintf('%s:%d',h.Nodes(order(i)).ID,i);
end
h.ShowTextInNodes = 'label'
dolayout(h)
Traverse the graph to find the breadth-first search (BFS) discovery order starting at node 4.

```matlab
order = graphtraverse(DG,4,'Method','BFS')
```

```
order =
   4     5     3     6     7     1     9     8     2    10
```

Label the nodes with the BFS discovery order.

```matlab
for i = 1:10
    h.Nodes(order(i)).Label = sprintf('%s:%d',h.Nodes(order(i)).ID,i);
end
```

```matlab
h.ShowTextInNodes = 'label'
dolayout(h)
```
Find and color nodes that are close to (within two edges of) node 4.

```matlab
node_idxs = graphtraverse(DG, 4, 'depth', 2)
node_idxs =
    4     5     3     6     7
set(h.nodes(node_idxs), 'Color', [1 0 0])
```
References


See Also

graphallshortestpaths | graphconncomp | graphisdag | graphisomorphism | graphisspantree | graphmaxflow | graphminspantree | graphpred2path | graphshortestpath | graphtopoorder | traverse

Introduced in R2006b
gt (DataMatrix)

Test DataMatrix objects for greater than

Syntax

\[ T = \text{gt}(\text{DMObj1, DMObj2}) \]
\[ T = \text{DMObj1} > \text{DMObj2} \]
\[ T = \text{gt}(\text{DMObj1, B}) \]
\[ T = \text{DMObj1} > B \]
\[ T = \text{gt}(B, \text{DMObj1}) \]
\[ T = B > \text{DMObj1} \]

Input Arguments

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>MATLAB numeric or logical array.</td>
</tr>
</tbody>
</table>

Output Arguments

| T              | Logical matrix of the same size as DMObj1 and DMObj2 or DMObj1 and B. It contains logical 1 (true) where elements in the first input are greater than the corresponding element in the second input, and logical 0 (false) otherwise. |

Description

\[ T = \text{gt}(\text{DMObj1, DMObj2}) \] or the equivalent \[ T = \text{DMObj1} > \text{DMObj2} \] compares each element in DataMatrix object DMObj1 to the corresponding element in DataMatrix object DMObj2, and returns \( T \), a logical matrix of the same size as DMObj1 and DMObj2, containing logical 1 (true) where elements in DMObj1 are greater than the corresponding element in DMObj2, and logical 0 (false) otherwise. DMObj1 and DMObj2 must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). DMObj1 and DMObj2 can have different Name properties.

\[ T = \text{gt}(\text{DMObj1, B}) \] or the equivalent \[ T = \text{DMObj1} > B \] compares each element in DataMatrix object DMObj1 to the corresponding element in B, a numeric or logical array, and returns \( T \), a logical matrix of the same size as DMObj1 and B, containing logical 1 (true) where elements in DMObj1 are greater than the corresponding element in B, and logical 0 (false) otherwise. DMObj1 and B must have the same size (number of rows and columns), unless one is a scalar.

\[ T = \text{gt}(B, \text{DMObj1}) \] or the equivalent \[ T = B > \text{DMObj1} \] compares each element in B, a numeric or logical array, to the corresponding element in DataMatrix object DMObj1, and returns \( T \), a logical matrix of the same size as B and DMObj1, containing logical 1 (true) where elements in B are greater than the corresponding element in DMObj1, and logical 0 (false) otherwise. B and DMObj1 must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls \( T = \text{gt}(X, Y) \) for the syntax \( T = X > Y \) when \( X \) or \( Y \) is a DataMatrix object.
See Also
DataMatrix | lt

Topics
DataMatrix object on page 1-532

Introduced in R2008b
GTFAnnotation class

Contain Gene Transfer Format (GTF) annotations

Description

The GTFAnnotation class contains annotations for one or more reference sequences, conforming to the GTF file format.

You construct a GTFAnnotation object from a GTF-formatted file. Each element in the object represents an annotation. Use the object properties and methods to filter annotations by feature, reference sequence, or reference sequence position. Use object methods to extract data for a subset of annotations into an array of structures.

Construction

Annotobj = GTFAnnotation(File) constructs Annotobj, a GTFAnnotation object, from File, a GTF-formatted file.

Input Arguments

File

Character vector or string specifying a GTF-formatted file.

Properties

FieldNames

Cell array of character vectors specifying the names of the available data fields for each annotation in the GTFAnnotation object. This property is read only.

NumEntries

Integer specifying number of annotations in the GTFAnnotation object. This property is read only.
Methods

getData   Create structure containing subset of data from GTFAnnotation object
getExons  Return table of exons from GTFAnnotation object
getFeatureNames  Retrieve unique feature names from GTFAnnotation object
getGenes   Return table of unique genes in GTFAnnotation object
getGeneNames  Retrieve unique gene names from GTFAnnotation object
getIndex   Return index array of annotations from GTFAnnotation object
getRange   Retrieve range of annotations from GTFAnnotation object
getReferenceNames  Retrieve reference names from GTFAnnotation object
getSegments  Return table of non-overlapping segments from GTFAnnotation object
getSubset   Create object containing subset of elements from GTFAnnotation object
getTranscripts  Return table of unique transcripts in GTFAnnotation object
getTranscriptNames  Retrieve unique transcript names from GTFAnnotation object

Copy Semantics

Value. To learn how value classes affect copy operations, see Copying Objects (MATLAB).

Indexing

GTFAnnotation objects support dot . indexing to extract properties.

Examples

Construct a GTFAnnotation object from a GTF-formatted file that is provided with Bioinformatics Toolbox:

GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf')

GTFAnnotObj =

    GTFAnnotation with properties:
    FieldNames: {1x11 cell}
    NumEntries: 308

See Also

GFFAnnotation

Topics

"Store and Manage Feature Annotations in Objects"

External Websites

GTF2.2: A Gene Annotation Format
HeatMap

Object containing matrix and heatmap display properties

Description

The HeatMap function creates a HeatMap object. You can use the object to display a heatmap (2-D color image) of matrix data.

Creation

Syntax

HeatMap(data)
HeatMap(data,Name,Value)

Description

hmObj = HeatMap(data) displays a heatmap (2-D color image) of data and returns an object containing the data and display properties.

hmObj = HeatMap(data,Name,Value) sets the object properties on page 1-913 using name-value pairs. For example, HeatMap(data,'Annotate',true) displays data values in the heatmap. You can specify multiple name-value pairs. Enclose each property name in quotes.

Input Arguments

data — Heatmap data
DataMatrix object | numeric matrix

Heatmap data, specified as a DataMatrix object on page 1-532 or numeric matrix.

Name-Value Pair Arguments

Use comma-separated name-value pair arguments to set the object properties on page 1-913. Enclose each property name in single quotes.

Example: hm = HeatMap(data,'Colormap',redbluecmap,'Annotate',true)

Properties

Standardize — Dimension for standardizing data values

'none' (default) | 'row' | 'column' | 3 | 2 | 1

Dimension for standardizing data values, specified as a character vector, string, or positive integer. Choices are:

• 'column' or 1 — Standardize along the columns of data.
• 'row' or 2 — Standardize along the rows of data.
• 'none' or 3 — Do not standardize.

If you specify 'column' or 'row', the function transforms the standardized values so that the mean is 0 and the standard deviation is 1 in the specified dimension.

Example: 'column'
Data Types: double | char | string

**Symmetric** — Flag to make the heatmap color scale symmetric around zero
true (default) | false

Flag to make the heatmap color scale symmetric around zero, specified as true or false.

Example: false
Data Types: logical

**DisplayRange** — Display range of standardize values
maximum absolute value in data (default) | positive scalar

Display range of standardize values, specified as a positive scalar. The default is the maximum absolute value in the input data.

For example, if you specify 3, there is a color variation for values between -3 and 3, but values greater than 3 are the same color as 3, and values less than -3 are the same color as -3.

For example, if you specify redgreencmap for the 'Colormap' property, pure red represents values greater than or equal to the specified display range value and pure green represents values less than or equal to the negative of the specified display range value.

Example: 3
Data Types: double

**Colormap** — Heatmap colors
redgreencmap (default) | matrix | name of function handle

heatmap colors, specified as a three-column (M-by-3) matrix of red-green-blue (RGB) values or the name of a function handle that returns a colormap, such as redgreencmap or redbluecmap.

The default colormap is redgreencmap, in which red represents values above the mean, black represents the mean, and green represents values below the mean of a row (gene) across all columns (samples).

Example: redbluecmap
Data Types: double | char

**ImputeFun** — Name of function or function handle to impute missing data
character vector | cell array

Name of a function or function handle to impute missing data, specified as a character vector or cell array. If you specify a cell array, the first element must be the name of a function or function handle, and the remaining elements must be name-value pairs used as inputs to the function. Missing data points are colored gray in the heatmap.
If data points are missing, you can use this property to impute the missing values.

Example: 'func1'
Data Types: char

ColumnLabels — Column labels
[1x0 double] (default) | string vector | cell array of character vectors | numeric vector

Column labels, specified as a string vector, cell array of character vectors, or numeric vector. The size of the vector must match the number of columns in the input data.

Example: ['sample1','sample2','sample3']
Data Types: double | string | cell

RowLabels — Row labels
[] (default) | string vector | cell array of character vectors | numeric vector

Row labels, specified as a string vector, cell array of character vectors, or numeric vector. The size of the vector must match the number of rows in the input data.

Example: ['gene1','gene2','gene3']
Data Types: double | string | cell

ColumnLabelsRotate — Orientation of column labels
90 (default) | numeric scalar

Orientation of column labels, specified as a numeric scalar. Specify the value of rotation in degrees (positive angles cause counterclockwise rotation).

Example: 30
Data Types: double

RowLabelsRotate — Orientation of row labels
0 (default) | numeric scalar

Orientation of row labels, specified as a numeric scalar. Specify the value of rotation in degrees (positive angles cause counterclockwise rotation).

Example: 30
Data Types: double

Annotate — Flag to display data values in heatmap
false (default) | true

Flag to display data values in the heatmap, specified as true or false.

Example: true
Data Types: logical

AnnotPrecision — Display precision of data values
2 (default) | numeric scalar

Display precision of data values in the heatmap, specified as a numeric scalar. The default number of digits of precision is 2.
Example: 3
Data Types: double

AnnotColor — Text color of displayed data values

*w* (default) | character vector | string | three-element numeric vector

Text color of displayed data values in the heatmap, specified as a character vector, string, or three-element numeric vector. For example, to use cyan, you can enter [0 1 1], 'c', "c", "cyan", or 'cyan'. For details, see “Color Options” on page 1-925.

Example: 'red'
Data Types: char | string | double

ColumnLabelsColor — Color information for column labels

[] (default) | structure | structure array

**Warning** This property will be removed in a future release. Set LabelsWithMarkers to true for colored markers instead of colored texts.

Color information for column labels, specified as a structure or structure array.

For a single structure, you must specify the following fields:

- **Labels** — Cell array of character vectors specifying column labels listed in the ColumnLabels property.
- **Colors** — Character vector or string specifying a color for the column labels. If this field is empty, the default color (black) is used.

For a structure array, you must specify a single element in each field for each structure.

- **Labels** — Character vector or string specifying a column label listed in the ColumnLabels property.
- **Colors** — Character vector or string specifying a color for the column labels. If this field is empty, the default color (black) is used.

For more information on specifying colors, see “Color Options” on page 1-925.

Data Types: struct

RowLabelsColor — Color information for row labels

[] (default) | structure | structure array

**Warning** This property will be removed in a future release. Set LabelsWithMarkers to true for colored markers instead of colored texts.

Color information for row labels, specified as a structure or structure array.

For a single structure, it must have following fields.

- **Labels** — Cell array of character vectors specifying row labels listed in the RowLabels property.
• **Colors** – Character vector or string specifying a color for the row labels. If this field is empty, the default color (black) is used.

For a structure array, each structure must have a single element in each field.

• **Labels** – Character vector or string specifying a row label listed in the RowLabels property.

• **Colors** – Character vector or string specifying a color for the row labels. If this field is empty, the default color (black) is used.

For more information on specifying colors, see “Color Options” on page 1-925.

**LabelsWithMarkers** — Flag to display colored markers for row and column labels

false (default) | true

Flag to display colored markers instead of colored text for the row and column labels, specified as true or false.

Example: true

Data Types: logical

**Object Functions**

- **view** — Display heatmap or clustergram
- **plot** — Render heatmap or clustergram
- **addTitle** — Add title to heatmap or clustergram
- **addXLabel** — Label x-axis of heatmap or clustergram
- **addYLabel** — Label y-axis of heatmap or clustergram

**Examples**

**Plot Heatmap of Data Matrix**

Create a matrix of data.

data = gallery('invhess',20);

Display a 2-D color heatmap of the data.

hmo = HeatMap(data);

```matlab
    Standardize: 'column | row | {none}"
    Symmetric: 'true | false'.
    DisplayRange: 'Scalar'.
    Colormap: []
    ImputeFun: 'string -or- function handle -or- cell array'
    ColumnLabels: 'Cell array of strings, or an empty cell array'
    RowLabels: 'Cell array of strings, or an empty cell array'
    ColumnLabelsRotate: []
    RowLabelsRotate: []
    Annotate: 'on | {off}''
    AnnotPrecision: []
    AnnotColor: []
    ColumnLabelsColor: 'A structure array.'
    RowLabelsColor: 'A structure array.'
```
Display the data values in the heatmap.

hmo.Annotate = true;
view(hmo)
Use the `plot` function to display the heatmap in another figure specified by the figure handle `fH`.

```matlab
fH = figure;
hA = plot(hmo,fH);
```
Use the returned axes handle hA to specify the axes properties.

hA.Title.String = 'Inverse of an Upper Hessenberg Matrix';
hA.XTickLabelMode = 'auto';
hA.YTickLabelMode = 'auto';
Add Custom Title and Labels to Heatmap

Load a sample of gene expression data.

load bc_train_filtered

Display a heatmap of the gene expression values for 4918 genes from 78 samples.

hmo = HeatMap(bcTrainData.Log10Ratio);

    Standardize: 'column | row | {none}';
    Symmetric: 'true | false';
    DisplayRange: 'Scalar';
    ColorMap: [];
    ImputeFun: 'string -or- function handle -or- cell array';
    ColumnLabels: 'Cell array of strings, or an empty cell array';
    RowLabels: 'Cell array of strings, or an empty cell array';
    ColumnLabelsRotate: [];
    RowLabelsRotate: [];
    Annotate: 'on | {off}';
    AnnotPrecision: [];
    AnnotColor: [];
    ColumnLabelsColor: 'A structure array';
    RowLabelsColor: 'A structure array';
    LabelsWithMarkers: 'true | false';
Add a title to the heatmap in red.

title = addTitle(hmo,'Gene Expression Data','Color','red');
Change the title font size.

\[ \text{title.FontSize} = 12; \]
Add labels to the x-axis and y-axis.

```matlab
addXLabel(hmo,'Samples','FontSize',12);
addYLabel(hmo,'Genes','FontSize',12);
```
More About

Color Options

The following lists the predefined colors and their RGB triplet equivalents. The short names and long names are character vectors that specify one of eight preset colors. The RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color; the intensities must be in the range [0 1].

<table>
<thead>
<tr>
<th>RGB Triplet</th>
<th>Short Name</th>
<th>Long Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1 1 0]</td>
<td>y</td>
<td>yellow</td>
</tr>
<tr>
<td>[1 0 1]</td>
<td>m</td>
<td>magenta</td>
</tr>
<tr>
<td>[0 1 1]</td>
<td>c</td>
<td>cyan</td>
</tr>
<tr>
<td>[1 0 0]</td>
<td>r</td>
<td>red</td>
</tr>
<tr>
<td>[0 1 0]</td>
<td>g</td>
<td>green</td>
</tr>
<tr>
<td>[0 0 1]</td>
<td>b</td>
<td>blue</td>
</tr>
<tr>
<td>[1 1 1]</td>
<td>w</td>
<td>white</td>
</tr>
<tr>
<td>[0 0 0]</td>
<td>k</td>
<td>black</td>
</tr>
</tbody>
</table>
See Also
clustergram | redbluecmap | redgreencmap

Introduced in R2009b
hmmprofalign

Align query sequence to profile using hidden Markov model alignment

Syntax

\[
\text{Score} = \text{hmmprofalign}(\text{Model}, \text{Seq}) \\
[\text{Score}, \text{Alignment}] = \text{hmmprofalign}(\text{Model}, \text{Seq}) \\
[\text{Score}, \text{Alignment}, \text{Pointer}] = \text{hmmprofalign}(\text{Model}, \text{Seq})
\]

\[\text{hmmprofalign}(\ldots, \text{'ShowScore'}, \text{ShowScoreValue}, \ldots)\]
\[\text{hmmprofalign}(\ldots, \text{'Flanks'}, \text{FlanksValue}, \ldots)\]
\[\text{hmmprofalign}(\ldots, \text{'ScoreFlanks'}, \text{ScoreFlanksValue}, \ldots)\]
\[\text{hmmprofalign}(\ldots, \text{'ScoreNullTransitions'}, \text{ScoreNullTransitionsValue}, \ldots)\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Hidden Markov model created with the function hmmprofstruct.</td>
</tr>
<tr>
<td>Seq</td>
<td>Amino acid or nucleotide sequence. You can also enter a structure with the field Sequence.</td>
</tr>
<tr>
<td>ShowScoreValue</td>
<td>Controls the display of the scoring space and the winning path. Choices are true or false (default).</td>
</tr>
<tr>
<td>FlanksValue</td>
<td>Controls the inclusion of the symbols generated by the FLANKING INSERT states in the output sequence. Choices are true or false (default).</td>
</tr>
<tr>
<td>ScoreFlanksValue</td>
<td>Controls the inclusion of the transition probabilities for the flanking states in the raw score. Choices are true or false (default).</td>
</tr>
<tr>
<td>ScoreNullTransitionsValue</td>
<td>Controls the adjustment of the raw score using the null model for transitions (Model.NullX). Choices are true or false (default).</td>
</tr>
</tbody>
</table>

Description

\[
\text{Score} = \text{hmmprofalign}(\text{Model}, \text{Seq})\] returns the score for the optimal alignment of the query amino acid or nucleotide sequence (Seq) to the profile hidden Markov model (Model). Scores are computed using log-odd ratios for emission probabilities and log probabilities for state transitions.

\[\text{[Score, Alignment]} = \text{hmmprofalign}(\text{Model}, \text{Seq})\] also returns a character vector showing the optimal profile alignment.

Uppercase letters and dashes correspond to MATCH and DELETE states respectively (the combined count is equal to the number of states in the model). Lowercase letters are emitted by the INSERT states. For more information about the HMM profile, see hmmprofstruct.

\[\text{[Score, Alignment, Pointer]} = \text{hmmprofalign}(\text{Model}, \text{Seq})\] also returns a vector of the same length as the profile model with indices pointing to the respective symbols of the query.
sequence. Null pointers (NaN) mean that such states did not emit a symbol in the aligned sequence because they represent model jumps from the BEGIN state of a MATCH state, model jumps from the from a MATCH state to the END state, or because the alignment passed through DELETE states.

hmmprofalign(..., 'PropertyName', PropertyValue, ...) calls hmmprofalign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

hmmprofalign(..., 'ShowScore', ShowScoreValue, ...), when ShowScoreValue is true, displays the scoring space and the winning path.

hmmprofalign(..., 'Flanks', FlanksValue, ...), when FlanksValue is true, includes the symbols generated by the FLANKING INSERT states in the output sequence.

hmmprofalign(..., 'ScoreFlanks', ScoreFlanksValue, ...), when ScoreFlanksValue is true, includes the transition probabilities for the flanking states in the raw score.

hmmprofalign(..., 'ScoreNullTransitions', ScoreNullTransitionsValue, ...), when ScoreNullTransitionsValue is true, adjusts the raw score using the null model for transitions (Model.NullX).

**Note** Multiple target alignment is not supported in this implementation. All the Model.LoopX probabilities are ignored.

### Examples

**Align query sequence to profile using HMM model alignment**

This example shows how to align a query sequence to a HMM model profile using HMM model alignment.

Load the HMM profile structure of the 7 transmembrane receptor (Secretin family).

```matlab
load('hmm_model_examples','model_7tm_2');
```

Load an example sequence and align it to the profile structure using the HMM alignment.

```matlab
load('hmm_model_examples','sequences');
humanSeq = sequences(1).Sequence;
[a,s]=hmmprofalign(model_7tm_2,humanSeq,'showscore',true)
```
\[ a = 483.7231 \]

\[ s = 'YILVKAIYTLGYSVS-LMSLATGSIILCLFRKLHCTRNYIHLNLFLSFI\text{LRAISV}L\text{VDDVLYSS}Sgtlhc\text{pdq}sswvg\text{CKL}LSV\text{FLQY}CIMANFF\text{WLLVEGLYLHTLL-VA---MLPPR...-WDTN-DHSVPWWVIRIPILISIIVNFVLFISIIRILLQKLT----SPDVGGNDQSQYKRLAKSTLLLIPLFGVHYMVFAVFPIS----ISSKYQILFELCLGSFQGLVVAVLYCFLNSEV' \]

**See Also**

gethmmprof | hmmprofestimate | hmmprofgenerate | hmmprofgenerate | hmmprofstruct | multialign | pfamhmmread | profalign | showhmmprof

*Introduced before R2006a*
**hmmprofestimate**

Estimate profile hidden Markov model (HMM) parameters using pseudocounts

**Syntax**

```matlab
hmmprofestimate(Model, MultipleAlignment, 'PropertyName', PropertyValue, ...)
```

```matlab
hmmprofestimate(..., 'A', AValue)
```

```matlab
hmmprofestimate(..., 'Ax', AxValue)
```

```matlab
hmmprofestimate(..., 'BE', BEValue)
```

```matlab
hmmprofestimate(..., 'BDx', BDxValue)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| Model | Hidden Markov model created with the function `hmmprofstruc`.
| MultipleAlignment | Array of sequences. Sequences can also be a structured array with the aligned sequences in a field `Aligned` or `Sequences` and the optional names in a field `Header` or `Name`.
| A | Property to set the pseudocount weight $A$. Default value is 20.
| Ax | Property to set the pseudocount weight $Ax$. Default value is 20.
| BE | Property to set the background symbol emission probabilities. Default values are taken from `Model.NullEmission`.
| BMx | Property to set the background transition probabilities from any MATCH state ($[M \rightarrow M \rightarrow I]$). Default values are taken from `hmmprofstruc`.
| BDx | Property to set the background transition probabilities from any DELETE state ($[D \rightarrow M \rightarrow D]$). Default values are taken from `hmmprofstruc`.

**Description**

`hmmprofestimate(Model, MultipleAlignment, 'PropertyName', PropertyValue...)` returns a structure with the fields containing the updated estimated parameters of a profile HMM. Symbol emission and state transition probabilities are estimated using the real counts and weighted pseudocounts obtained with the background probabilities. Default weight is $A=20$, the default background symbol emission for match and insert states is taken from `Model.NullEmission`, and the default background transition probabilities are the same as default transition probabilities returned by `hmmprofstruc`.

Model Construction: Multiple aligned sequences should contain uppercase letters and dashes indicating the model MATCH and DELETE states agreeing with `Model.ModelLength`. If model state annotation is missing, but `MultipleAlignment` is space aligned, then a “maximum entropy” criteria is used to select `Model.ModelLength` states.
**Note** Insert and flank insert transition probabilities are not estimated, but can be modified afterwards using `hmmprofstruct`.

`hmmprofestimate(..., 'A', AValue)` sets the pseudocount weight $A = Avalue$ when estimating the symbol emission probabilities. Default value is 20.

`hmmprofestimate(..., 'Ax', AxValue)` sets the pseudocount weight $Ax = Axvalue$ when estimating the transition probabilities. Default value is 20.

`hmmprofestimate(..., 'BE', BEValue)` sets the background symbol emission probabilities. Default values are taken from `Model.NullEmission`.

`hmmprofestimate(..., 'BMx', BMxValue)` sets the background transition probabilities from any MATCH state ([M->M M->I M->D]). Default values are taken from `hmmprofstruct`.

`hmmprofestimate(..., 'BDx', BDxValue)` sets the background transition probabilities from any DELETE state ([D->M D->D]). Default values are taken from `hmmprofstruct`.

**See Also**
`hmmprofalign` | `hmmprofstruct` | `showhmmprof`

_Introduced before R2006a_
**hmmprofgenerate**

Generate random sequence drawn from profile hidden Markov model (HMM)

**Syntax**

```
Sequence = hmmprofgenerate(Model)
[Sequence, Profptr] = hmmprofgenerate(Model)
... = hmmprofgenerate(Model, ...'Align', AlignValue, ...)
... = hmmprofgenerate(Model, ...'Flanks', FlanksValue, ...)
... = hmmprofgenerate(Model, ...'Signature', SignatureValue, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Model</th>
<th>Hidden Markov model created with the hmmprofstruct function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlignValue</td>
<td>Controls the use of uppercase letters for matches and lowercase</td>
</tr>
<tr>
<td></td>
<td>letters for inserted letters. Choices are true or false (default).</td>
</tr>
<tr>
<td>FlanksValue</td>
<td>Controls the inclusion of the symbols generated by the FLANKING</td>
</tr>
<tr>
<td></td>
<td>INSERT states in the output sequence. Choices are true or false</td>
</tr>
<tr>
<td></td>
<td>(default).</td>
</tr>
<tr>
<td>SignatureValue</td>
<td>Controls the return of the most likely path and symbols.</td>
</tr>
<tr>
<td></td>
<td>Choices are true or false (default).</td>
</tr>
</tbody>
</table>

**Description**

`Sequence = hmmprofgenerate(Model)` returns a sequence of amino acids or nucleotides drawn from the profile `Model`. The length, alphabet, and probabilities of the `Model` are stored in a structure. For more information about this structure, see `hmmprofstruct`.

`[Sequence, Profptr] = hmmprofgenerate(Model)` returns a vector of the same length as the profile model pointing to the respective states in the output sequence. Null pointers (0) mean that such states do not exist in the output sequence, either because they are never touched (i.e., jumps from the BEGIN state to MATCH states or from MATCH states to the END state), or because DELETE states are not in the output sequence (not aligned output; see below).

`... = hmmprofgenerate(Model, ...'PropertyName', PropertyValue, ...)` calls `hmmprofgenerate` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

... = `hmmprofgenerate(Model, ...'Align', AlignValue, ...)` if `Align` is true, the output sequence is aligned to the model as follows: uppercase letters and dashes correspond to MATCH and DELETE states respectively (the combined count is equal to the number of states in the model). Lowercase letters are emitted by the INSERT or FLANKING INSERT states. If `AlignValue` is false, the output is a sequence of uppercase symbols. The default value is true.
... = hmmprofgenerate(Model, ... 'Flanks', FlanksValue, ...) if Flanks is true, the output sequence includes the symbols generated by the FLANKING INSERT states. The default value is false.

... = hmmprofgenerate(Model, ... 'Signature', SignatureValue, ...) if SignatureValue is true, returns the most likely path and symbols. The default value is false.

Examples

load('hmm_model_examples','model_7tm_2') % load a model example
rand_sequence = hmmprofgenerate(model_7tm_2)

See Also

hmmprofalign | hmmprofstruct | showhmmprof

Introduced before R2006a
hmmprofmerge

Displays a set of HMM profile alignments

Syntax

hmmprofmerge(Sequences)
hmmprofmerge(Sequences, Names)
hmmprofmerge(Sequences, Names, Scores)

Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequences</td>
<td>Array of sequences. Sequences can also be a structured array with the aligned sequences in a field Aligned or Sequences, and the optional names in a field Header or Name.</td>
</tr>
<tr>
<td>Names</td>
<td>Names for the sequences. Enter a cell array of character vectors.</td>
</tr>
<tr>
<td>Scores</td>
<td>Pairwise alignment scores from the function hmmprofalign. Enter a vector of values with the same length as the number of sequences in Sequences.</td>
</tr>
</tbody>
</table>

Description

hmmprofmerge(Sequences) opens your default Web browser and displays a set of prealigned sequences to an HMM model profile. The output is aligned corresponding to the HMM states.

- **Match states** — Uppercase letters
- **Insert states** — Lowercase letters or asterisks (*)
- **Delete states** — Dashes

Periods (.) are added at positions corresponding to inserts in other sequences. The input sequences must have the same number of profile states, that is, the joint count of capital letters and dashes must be the same.

hmmprofmerge(Sequences, Names) labels the sequences with Names.

hmmprofmerge(Sequences, Names, Scores) sorts the displayed sequences using Scores.

Examples

```matlab
load('hmm_model_examples','model_7tm_2') %load model
load('hmm_model_examples','sequences') %load sequences
for ind =1:length(sequences)
    [scores(ind),sequences(ind).Aligned] = ...
        hmmprofalign(model_7tm_2,sequences(ind).Sequence);
end
hmmprofmerge(sequences, scores)
```

See Also

hmmprofalign | hmmprofstruct
Introduced before R2006a
**hmmprofstruct**

Create or edit hidden Markov model (HMM) profile structure

**Syntax**

```
Model = hmmprofstruct(Length)
Model = hmmprofstruct(Length, Field1, Field1Value, Field2, Field2Value, ...)
NewModel = hmmprofstruct(Model, Field1, Field1Value, Field2, Field2Value, ...)
```

**Input Arguments**

- **Length**
  - Number of match states in the model.
- **Model**
  - MATLAB structure containing fields for the parameters of an HMM profile created with the `hmmprofstruct` function.
- **Field**
  - Character vector or string containing a field name in the structure `Model`. See the table below for field names.
- **FieldValue**
  - Value associated with `Field`. See the table below for descriptions.

**Output Arguments**

- **Model**
  - MATLAB structure containing fields for the parameters of an HMM profile.

**Description**

- **Model = hmmprofstruct(Length)** returns `Model`, a MATLAB structure containing fields for the parameters of an HMM profile. `Length` specifies the number of match states in the model. All other required parameters are set to the default values.

- **Model = hmmprofstruct(Length, Field1, Field1Value, Field2, Field2Value, ...)** returns an HMM profile structure using the specified parameters. All other required parameters are set to default values.

- **NewModel = hmmprofstruct(Model, Field1, Field1Value, Field2, Field2Value, ...)** returns an updated HMM profile structure using the specified parameters. All other parameters are taken from the input `Model`.

**HMM Profile Structure**

The MATLAB structure `Model` contains the following fields, which are the required and optional parameters of an HMM profile. All probability values are in the [0 1] range.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelLength</td>
<td>Integer specifying the length of the profile (number of MATCH states).</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Alphabet</td>
<td>Character vector or string specifying the alphabet used in the model. Choices are 'AA' (default) or 'NT'.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>AlphaLength is 20 for 'AA' and 4 for 'NT'.</td>
</tr>
<tr>
<td>MatchEmission</td>
<td>Symbol emission probabilities in the MATCH states.</td>
</tr>
<tr>
<td></td>
<td>Either of the following:</td>
</tr>
<tr>
<td></td>
<td>• A matrix of size ModelLength-by-AlphaLength, where each row corresponds to the emission distribution for a specific MATCH state. Defaults to uniform distributions.</td>
</tr>
<tr>
<td></td>
<td>• A structure containing residue counts, such as returned by aacount or basecount.</td>
</tr>
<tr>
<td>InsertEmission</td>
<td>Symbol emission probabilities in the INSERT state.</td>
</tr>
<tr>
<td></td>
<td>Either of the following:</td>
</tr>
<tr>
<td></td>
<td>• A matrix of size ModelLength-by-AlphaLength, where each row corresponds to the emission distribution for a specific INSERT state. Defaults to uniform distributions.</td>
</tr>
<tr>
<td></td>
<td>• A structure containing residue counts, such as returned by aacount or basecount.</td>
</tr>
<tr>
<td>NullEmission</td>
<td>Symbol emission probabilities in the MATCH and INSERT states for the NULL model.</td>
</tr>
<tr>
<td></td>
<td>Either of the following:</td>
</tr>
<tr>
<td></td>
<td>• A 1-by-AlphaLength row vector. Defaults to a uniform distribution.</td>
</tr>
<tr>
<td></td>
<td>• A structure containing residue counts, such as returned by aacount or basecount.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>The NULL model is used to compute the log-odds ratio at every state and avoid overflow when propagating the probabilities through the model.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>NULL probabilities are also known as the background probabilities.</td>
</tr>
</tbody>
</table>
### Field | Description
--- | ---
**BeginX** | BEGIN state transition probabilities. Format is a 1-by-\((\text{ModelLength} + 1)\) row vector: 
\[
[B->D1 \ B->M1 \ B->M2 \ B->M3 \ldots \ B->\text{Mend}]
\]
**Note** If necessary, `hmmprofstruct` will normalize the data such that the sum of the transition probabilities from the BEGIN state equals 1: 
\[
\text{sum} (\text{Model.BeginX}) = 1
\]
For fragment profiles: 
\[
\text{sum} (\text{Model.BeginX}(3:\text{end})) = 0
\]
Default is \([0.01 \ 0.99 \ 0 \ 0 \ldots \ 0]\).

**MatchX** | MATCH state transition probabilities. Format is a 4-by-\((\text{ModelLength} - 1)\) matrix: 
\[
[M1->M2 \ M2->M3 \ldots \ M[\text{end}-1]->\text{Mend}; 
M1->I1 \ M2->I2 \ldots \ M[\text{end}-1]->I[\text{end}-1]; 
M1->D2 \ M2->D3 \ldots \ M[\text{end}-1]->D\text{end}; 
M1->E \ M2->E \ldots \ M[\text{end}-1]->E ]
\]
**Note** If necessary, `hmmprofstruct` will normalize the data such that the sum of the transition probabilities from every MATCH state equals 1: 
\[
\text{sum} (\text{Model.MatchX}) = [1 \ 1 \ldots \ 1]
\]
For fragment profiles: 
\[
\text{sum} (\text{Model.MatchX}(4,:)) = 0
\]
Default is `repmat([0.998 \ 0.001 \ 0.001 \ 0],\text{ModelLength}-1,1)`.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InsertX</td>
<td>INSERT state transition probabilities.</td>
</tr>
<tr>
<td></td>
<td>Format is a 2-by-((\text{ModelLength} - 1)) matrix:</td>
</tr>
</tbody>
</table>
|         | \[
|         | \begin{bmatrix}
|         | \text{I}1\rightarrow\text{M}2 \quad \text{I}2\rightarrow\text{M}3 \quad \ldots \quad \text{I}[\text{end}\text{-}1]\rightarrow\text{Mend};
|         | \text{I}1\rightarrow\text{I}1 \quad \text{I}2\rightarrow\text{I}2 \quad \ldots \quad \text{I}[\text{end}\text{-}1]\rightarrow\text{I}[\text{end}\text{-}1]
|         | \end{bmatrix}
|         | Note If necessary, \texttt{hmmprofstruct} will normalize the data such that \(\text{sum(}\text{Model.InsertX}\text{)} = [1 \ 1 \ \ldots \ 1]\) |
|         | Default is \texttt{repmat([0.5 0.5],\text{ModelLength}\text{-}1,1)}. |
| DeleteX | DELETE state transition probabilities.           |
|         | Format is a 2-by-\((\text{ModelLength} - 1)\) matrix: |
|         | \[
|         | \begin{bmatrix}
|         | \text{D}1\rightarrow\text{M}2 \quad \text{D}2\rightarrow\text{M}3 \quad \ldots \quad \text{D}[\text{end}\text{-}1]\rightarrow\text{Mend};
|         | \text{D}1\rightarrow\text{D}2 \quad \text{D}2\rightarrow\text{D}3 \quad \ldots \quad \text{D}[\text{end}\text{-}1]\rightarrow\text{Dend}
|         | \end{bmatrix}
<p>|         | Note If necessary, \texttt{hmmprofstruct} will normalize the data such that (\text{sum(}\text{Model.DeleteX}\text{)} = [1 \ 1 \ \ldots \ 1]) |
|         | Default is \texttt{repmat([0.5 0.5],\text{ModelLength}\text{-}1,1)}. |</p>
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlankingInsertX</td>
<td>Flanking insert states (N and C) used for LOCAL profile alignment. Format is a 2-by-2 matrix:</td>
</tr>
<tr>
<td></td>
<td>[N-&gt;B  C-&gt;T ; N-&gt;N  C-&gt;C]</td>
</tr>
</tbody>
</table>

**Note** If necessary, `hmmprofstruct` will normalize the data such that the sum of the transition probabilities from Flanking Insert states equals 1:

\[ \text{sum}(Model.FlankingInsertsX) = [1 \ 1] \]

**Note** To force global alignment use:

\[ Model.FlankingInsertsX = [1 \ 1; 0 \ 0] \]

Default is \[ [0.01 \ 0.01; 0.99 \ 0.99] \].

<table>
<thead>
<tr>
<th>LoopX</th>
<th>Loop states transition probabilities used for multiple hits alignment. Format is a 2-by-2 matrix:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[E-&gt;C  J-&gt;B ; E-&gt;J  J-&gt;J]</td>
</tr>
</tbody>
</table>

**Note** If necessary, `hmmprofstruct` will normalize the data such that the sum of the transition probabilities from Loop states equals 1:

\[ \text{sum}(Model.LoopX) = [1 \ 1] \]

Default is \[ [0.5 \ 0.01; 0.5 \ 0.99] \].

<table>
<thead>
<tr>
<th>NullX</th>
<th>Null transition probabilities used to provide scores with log-odds values also for state transitions. Format is a 2-by-1 column vector:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[G-&gt;F ; G-&gt;G]</td>
</tr>
</tbody>
</table>

**Note** If necessary, `hmmprofstruct` will normalize the data such that the sum of the transition probabilities from Null states equals 1:

\[ \text{sum}(Model.NullX) = 1 \]

Default is \[ [0.01; 0.99] \].
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDNumber</td>
<td>Optional. User-assigned identification number.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional. User-assigned description of the model.</td>
</tr>
</tbody>
</table>

### HMM Profile Model

An HMM profile model is a common statistical tool for modeling structured sequences composed of symbols. These symbols include randomness in both the output (emission of symbols) and the state transitions of the process. Markov models are generally represented by state diagrams.

The following figure is a state diagram for an HMM profile of length four. INSERT, MATCH, and DELETE states are in the center section.

- INSERT state represents the excess of one or more symbols in the target sequence that are not included in the profile.
- MATCH state means that the target sequence is aligned to the profile at the specific location.
- DELETE state represents a gap or symbol absence in the target sequence (also known as a silent state because it does not emit any symbols).

Flanking states (S, N, B, E, C, T) are used for proper modeling of the ends of the sequence, either for global, local or fragment alignment of the profile. S, B, E, and T are silent, while N and C are used to insert symbols at the flanks.

### Examples

#### Example 1.41. Creating an HMM Profile Structure

Create an HMM profile structure with 100 MATCH states, using the amino acid alphabet.

```matlab
hmmProfile = hmmprofstruct(100, 'Alphabet', 'AA')
hmmProfile =
```

```matlab
    ModelLength: 100
    Alphabet: 'AA'
    MatchEmission: [100x20 double]
    InsertEmission: [100x20 double]
    NullEmission: [1x20 double]
    BeginX: [101x1 double]
    MatchX: [99x4 double]
    InsertX: [99x2 double]
    DeleteX: [99x2 double]
    FlankingInsertX: [2x2 double]
```
LoopX: [2x2 double]
NullX: [2x1 double]

Example 1.42. Editing an HMM Profile Structure

1. Use the `pfamhmmread` function to create an HMM profile structure from `pf00002.ls`, a PFAM HMM-formatted file included with the software.

```matlab
hmm02 = pfamhmmread('pf00002.ls');
```

2. Modify the HMM profile structure to force a global alignment by setting the looping transition probabilities in the flanking insert states to zero.

```matlab
hmm02 = hmmprofstruct(hmm02,'FlankingInsertX',[0 0;1 1]);
ans.

hmm02.FlankingInsertX =
   0     0
   1     1
```

See Also

`aacount` | `basecount` | `gethmmprof` | `hmmprofalign` | `hmmprofestimate` | `hmmprofgenerate` | `hmmprofmerge` | `pfamhmmread` | `showhmmprof`

Introduced before R2006a
horzcat (DataMatrix)

Concatenate DataMatrix objects horizontally

Syntax

DMObjNew = horzcat(DMObj1, DMObj2, ...)
DMObjNew = (DMObj1, DMObj2, ...)
DMObjNew = horzcat(DMObj1, B, ...)
DMObjNew = (DMObj1, B, ...)

Input Arguments

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>MATLAB numeric or logical array.</td>
</tr>
</tbody>
</table>

Output Arguments

| DMObjNew | DataMatrix object created by horizontal concatenation. |

Description

DMObjNew = horzcat(DMObj1, DMObj2, ...) or the equivalent DMObjNew = (DMObj1, DMObj2, ...) horizontally concatenates the DataMatrix objects DMObj1 and DMObj2 into DMObjNew, another DataMatrix object. DMObj1 and DMObj2 must have the same number of rows. The row names and the order of rows for DMObjNew are the same as DMObj1. The row names of DMObj2 and any other DataMatrix object input arguments are not preserved. The column names for DMObjNew are the column names of DMObj1, DMObj2, and other DataMatrix object input arguments.

DMObjNew = horzcat(DMObj1, B, ...) or the equivalent DMObjNew = (DMObj1, B, ...) horizontally concatenates the DataMatrix object DMObj1 and a numeric or logical array B into DMObjNew, another DataMatrix object. DMObj1 and B must have the same number of rows. The row names for DMObjNew are the same as DMObj1. The row names of DMObj2 and any other DataMatrix object input arguments are not preserved. The column names for DMObjNew are the column names of DMObj1 and empty for the columns from B.

MATLAB calls DMObjNew = horzcat(X1, X2, X3, ...) for the syntax DMObjNew = [X1, X2, X3, ...] when any one of X1, X2, X3, etc. is a DataMatrix object.

See Also
DataMatrix | vertcat

Topics
DataMatrix object on page 1-532

Introduced in R2008b
**ilmnbslookup**

Look up Illumina BeadStudio target (probe) sequence and annotation information

**Syntax**

```
AnnotStruct = ilmnbslookup(AnnotationFile, ID)
AnnotStruct = ilmnbslookup(AnnotationFile, ID, 'LookUpField', LookUpFieldValue)
```

**Input Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnnotationFile</td>
<td>Character vector or string specifying a file name or a path and file name of an Illumina annotation file (CSV, BGX, or TXT format). If you specify only a file name, that file must be on the MATLAB search path or in the current folder.</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>You can download Illumina annotation files, such as HumanRef-8_V3_0_R0_11282963_A.bgx, from the Illumina Web site.</td>
</tr>
<tr>
<td>ID</td>
<td>Character vector, string, string vector, or cell array of character vectors representing a unique identifier(s) for one or more targets (probes) on an Illumina microarray.</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>By default, ID must match the Search_key field in AnnotationFile. However, you can use an identifier that corresponds to any of the fields in AnnotationFile, then set the 'LookUpField' property appropriately. For example, if you want to look up annotation information for the targets (probes) on chromosome 7 only, set ID to '7', then set LookUpFieldValue to 'Chromosome'. For a list of all fields in AnnotationFile, see the following tables.</td>
</tr>
<tr>
<td>LookUpFieldValue</td>
<td>Field in AnnotationFile where ilmnbslookup looks for the specified ID. Default is the Search_key field.</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>Set this property so that it corresponds to the ID you use as input.</td>
</tr>
</tbody>
</table>

**Output Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnnotStruct</td>
<td>Structure containing the probe sequence and annotation information for one or more targets (probes) specified by ID, and by AnnotationFile, an Illumina annotation file.</td>
</tr>
<tr>
<td><strong>AnnotStruct</strong></td>
<td>contains the same fields as AnnotationFile. The fields are described in the following two tables.</td>
</tr>
</tbody>
</table>
Description

AnnotStruct = ilmnsbslookup(AnnotationFile, ID) returns AnnotStruct, a structure containing probe sequence and annotation information for one or more targets (probes) specified by ID, and by AnnotationFile, an Illumina annotation file (CSV, BGX, or TXT format).

AnnotStruct contains the same fields as AnnotationFile. The fields are described in the following two tables.

Structure Created from Illumina CSV Annotation File

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search_key</td>
<td>Internal identifier for the target, useful for custom design array</td>
</tr>
<tr>
<td>Target</td>
<td>Unique identifier for the target</td>
</tr>
<tr>
<td>ProbeId</td>
<td>Illumina probe identifier</td>
</tr>
<tr>
<td>Gid</td>
<td>GenBank identifier for the gene</td>
</tr>
<tr>
<td>Transcript</td>
<td>Illumina internal transcript identifier</td>
</tr>
<tr>
<td>Accession</td>
<td>GenBank accession number for the gene</td>
</tr>
<tr>
<td>Symbol</td>
<td>Typically, the gene symbol</td>
</tr>
<tr>
<td>Type</td>
<td>Probe type</td>
</tr>
<tr>
<td>Start</td>
<td>Starting position of the probe sequence in the GenBank record</td>
</tr>
<tr>
<td>Probe_Sequence</td>
<td>Sequence of the probe</td>
</tr>
<tr>
<td>Definition</td>
<td>Definition field from the GenBank record</td>
</tr>
<tr>
<td>Ontology</td>
<td>Gene Ontology terms associated with the gene</td>
</tr>
<tr>
<td>Synonym</td>
<td>Synonyms for the gene (from the GenBank record)</td>
</tr>
</tbody>
</table>
## Structure Created from a BGX or TXT Annotation File

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accession</td>
<td>GenBank accession number for the gene</td>
</tr>
<tr>
<td>Array_Address_Id</td>
<td>Decoder identifier</td>
</tr>
<tr>
<td>Chromosome</td>
<td>Chromosome on which the gene is located</td>
</tr>
<tr>
<td>Cytoband</td>
<td>Cytogenetic banding region of the chromosome on which the target is located</td>
</tr>
<tr>
<td>Definition</td>
<td>Definition field from the GenBank record</td>
</tr>
<tr>
<td>Entrez_Gene_ID</td>
<td>Entrez Gene database identifier for the gene</td>
</tr>
<tr>
<td>GI</td>
<td>GenBank identifier for the gene</td>
</tr>
<tr>
<td>ILMN_Gene</td>
<td>Illumina internal gene symbol</td>
</tr>
<tr>
<td>Obsolete_Probe_Id</td>
<td>Probe identifier before BGX annotation files</td>
</tr>
<tr>
<td>Ontology_Component</td>
<td>Gene Ontology cellular components associated with the gene</td>
</tr>
<tr>
<td>Ontology_Function</td>
<td>Gene Ontology molecular functions associated with the gene</td>
</tr>
<tr>
<td>Ontology_Process</td>
<td>Gene Ontology biological processes associated with the gene</td>
</tr>
<tr>
<td>Probe_Chr_Orientation</td>
<td>Orientation of the probe on the NCBI genome build</td>
</tr>
<tr>
<td>Probe_Coordinates</td>
<td>Genomic position of the probe on the NCBI genome build</td>
</tr>
<tr>
<td>Probe_Id</td>
<td>Illumina probe identifier</td>
</tr>
<tr>
<td>Probe_Sequence</td>
<td>Sequence of the probe</td>
</tr>
<tr>
<td>Probe_Start</td>
<td>Start position of the probe relative to the 5’ end of the source transcript sequence</td>
</tr>
<tr>
<td>Probe_Type</td>
<td>Information about what the probe is targeting</td>
</tr>
<tr>
<td>Protein_Product</td>
<td>NCBI protein accession number</td>
</tr>
<tr>
<td>RefSeq_ID</td>
<td>Identifier from the NCBI RefSeq database</td>
</tr>
<tr>
<td>Reporter_Composite_map</td>
<td>Information associated with control probes</td>
</tr>
<tr>
<td>Reporter_Group_Name</td>
<td>Information associated with control probes</td>
</tr>
<tr>
<td>Reporter_Group_id</td>
<td>Information associated with control probes</td>
</tr>
<tr>
<td>Search_Key</td>
<td>Internal identifier for the target, useful for custom design array</td>
</tr>
<tr>
<td>Source</td>
<td>Source from which the transcript sequence was obtained</td>
</tr>
<tr>
<td>Source_Reference_ID</td>
<td>Source’s identifier</td>
</tr>
<tr>
<td>Species</td>
<td>Species associated with the gene</td>
</tr>
<tr>
<td>Symbol</td>
<td>Typically, the gene symbol</td>
</tr>
<tr>
<td>Synonyms</td>
<td>Synonyms for the gene (from the GenBank record)</td>
</tr>
<tr>
<td>Transcript</td>
<td>Illumina internal transcript identifier</td>
</tr>
<tr>
<td>Unigene_ID</td>
<td>Identifier from the NCBI UniGene database</td>
</tr>
</tbody>
</table>

`AnnotStruct = ilmnslookup(AnnotationFile, ID, 'LookUpField', LookUpFieldValue)` looks for ID in the annotation file in the field specified by `LookUpFieldValue`. Default is the `Search_key` field.
Examples

Note The gene expression file, TumorAdjacent-probe-raw.txt, and the annotation file, HumanRef-8_V3_0_R0_11282963_A.bgx, used in the following examples are not provided with the Bioinformatics Toolbox software.

Example 1.43. Look Up Annotation Information for a Single Target (Probe)

1. Read the contents of a tab-delimited file exported from the Illumina BeadStudio™ software into a MATLAB structure.

   ```matlab
   ilmnStruct = ilmnbsread('TumorAdjacent-probe-raw.txt')
   ilmnStruct =
   Header: [1x1 struct]
   TargetID: {22184x1 cell}
   ColumnNames: {1x37 cell}
   Data: [22184x37 double]
   TextColumnNames: {1x23 cell}
   TextData: [22184x23 cell]
   ```

2. Find the number of the Search_key column in the TextColumnNames cell array, which is returned in the ilmnStruct structure by the ilmnbsread function.

   ```matlab
   srchCol = find(strcmpi('Search_Key', ilmnStruct.TextColumnNames))
   srchCol =
   1
   ```

3. Look up the probe sequence and annotation information for the 10th entry in the annotation file, HumanRef-8_V3_0_R0_11282963_A.bgx.

   ```matlab
   annotation = ilmnbslookup('HumanRef-8_V3_0_R0_11282963_A.bgx',
   ilmnStruct.TextData{10,srchCol})
   annotation =
   Accession: 'NM_144670.2'
   Array_Address_Id: '0004050154'
   Chromosome: '12'
   Cytoband: '12p13.31b'
   Definition: 'Homo sapiens alpha-2-macroglobulin-like 1 (A2ML1), mRNA.'
   Entrez_Gene_ID: '144568'
   GI: '74271844'
   ILMN_Gene: 'A2ML1'
   Obsolete_Probe_Id: '
   Ontology_Component: '
   Ontology_Function: 'endopeptidase inhibitor activity [goid 4866] [evidence IEA]'
   Ontology_Process: '
   Probe_Chr_Orientation: '+'
   Probe_Coordinates: '8920412-8920461'
   Probe_ID: 'ILMN_2136495'
   Probe_Sequence: 'TGTAATCGCAGCCCCTTGGAAGGCCAAGGCAGGAGAATCGCCTCAACACT'
   Probe_Start: '4889'
   Probe_Type: 'S'
   Protein_Product: 'NP_653271.2'
   RefSeq_ID: 'NM_144670.2'
   Reporter_Composite_map: '
   Reporter_Group_Name: '
   Reporter_Group_id: '
   Search_Key: 'ILMN_17375'
   Source: 'RefSeq'
   Source_Reference_ID: 'NM_144670.2'
   Species: 'Homo sapiens'
   Symbol: 'A2ML1'
   ```
Example 1.44. Look Up Annotation Information for a Subset of Targets (Probes)

Use the `ilmnbslookup` function with the 'LookUpField' property to look up the annotation information for all targets located on chromosome 12 in the annotation file, `HumanRef-8_V3_0_R0_11282963_A.bgx`.

```matlab
chr12annotation = ilmnbslookup('HumanRef-8_V3_0_R0_11282963_A.bgx', ...
    '12','LookUpField','Chromosome')
```

The output structure indicates that there are 1,186 targets located on chromosome 12.

See Also

`ilmnbsread`

Introduced in R2008a
**ilmnbsread**

Read gene expression data exported from Illumina BeadStudio software

**Syntax**

IlmnStruct = ilmnbsread(File)

IlmnStruct = ilmnbsread(File, ...'Columns', ColumnsValue, ...)

IlmnStruct = ilmnbsread(File, ...'HeaderOnly', HeaderOnlyValue, ...)

IlmnStruct = ilmnbsread(File, ...'CleanColNames', CleanColNamesValue, ...)

**Input Arguments**

<table>
<thead>
<tr>
<th><strong>File</strong></th>
<th>Character vector or string specifying a file name or a path and file name of a tab-delimited file or comma-separated expression data file exported from Illumina BeadStudio software. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ColumnsValue</strong></td>
<td>Cell array that specifies the column names to read. Default is all column names.</td>
</tr>
<tr>
<td><strong>HeaderOnlyValue</strong></td>
<td>Controls the population of only the Header, ColumnNames, and TextColumnNames fields in IlmnStruct. Choices are true or false (default).</td>
</tr>
<tr>
<td><strong>CleanColNamesValue</strong></td>
<td>Controls the conversion of any ColumnNames containing spaces or characters that cannot be used as MATLAB variable names, to valid MATLAB variable names. Choices are true or false (default).</td>
</tr>
</tbody>
</table>

**Output Arguments**

| IlmnStruct | MATLAB structure containing data exported from Illumina BeadStudio software. |

**Description**

IlmnStruct = ilmnbsread(File) reads File, a tab-delimited or comma-separated expression data file exported from the Illumina BeadStudio software, and creates IlmnStruct, a MATLAB structure containing the following fields.

<table>
<thead>
<tr>
<th><strong>Field</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Character vector containing a description of the data.</td>
</tr>
<tr>
<td>TargetID</td>
<td>Cell array containing unique identifiers for targets on an Illumina gene expression microarray.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColumnNames</td>
<td>Cell array containing names of the columns that contain numeric data in the tab-delimited file exported from the Illumina BeadStudio software.</td>
</tr>
<tr>
<td>Data</td>
<td>Matrix containing numeric microarray data for each target on an Illumina gene expression microarray.</td>
</tr>
</tbody>
</table>

**Note** ColumnNames and Data have the same number of columns.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextColumnNames</td>
<td>Cell array containing names of the columns that contain nonnumeric data in the tab-delimited file exported from the Illumina BeadStudio software. This field can be empty.</td>
</tr>
<tr>
<td>TextData</td>
<td>Cell array containing nonnumeric microarray data (such as annotations) for each target on an Illumina gene expression microarray. This field can be empty.</td>
</tr>
</tbody>
</table>

**Note** TextColumnNames and TextData have the same number of columns.

IlmnStruct = ilmnbsread(File, ...'PropertyName', PropertyValue, ...) calls ilmnbsread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

IlmnStruct = ilmnbsread(File, ...'Columns', ColumnsValue, ...) reads the data only from the columns specified by ColumnsValue, a cell array of column names. Default behavior is to read data from all columns.

IlmnStruct = ilmnbsread(File, ...'HeaderOnly', HeaderOnlyValue, ...) controls the population of only the Header, ColumnNames, and TextColumnNames fields in IlmnStruct. Choices are true or false (default).

IlmnStruct = ilmnbsread(File, ...'CleanColNames', CleanColNamesValue, ...) controls the conversion of any ColumnNames containing spaces or characters that cannot be used as MATLAB variable names, to valid MATLAB variable names. Choices are true or false (default).

**Tip** Use the 'CleanColNames' property if you plan to use the ColumnNames field as variable names.

### Examples

**Note** The gene expression file, TumorAdjacent-probe-raw.txt used in the following example is not provided with the Bioinformatics Toolbox software.

Read the contents of a tab-delimited file exported from the Illumina BeadStudio software into a MATLAB structure.
ilmnStruct = ilmnbsread('TumorAdjacent-probe-raw.txt')

ilmnStruct =

    Header: [1x1 struct]
    TargetID: {22184x1 cell}
    ColumnNames: {1x37 cell}
        Data: [22184x37 double]
    TextColumnNames: {1x23 cell}
        TextData: {22184x23 cell}

See Also
affyread|agferead|celintensityread|galread|geoseriesread|geosoftread|
gprread|ilmnbslookup|imageneread|magetfield|sptread

Introduced in R2008a
imageneread

Read microarray data from ImaGene Results file

Syntax

\[ \text{imagenedata} = \text{imageneread}(\text{File}) \]

\[ \text{imagenedata} = \text{imageneread}(\ldots, \text{'}CleanColNames\text{', CleanColNamesValue}, \ldots) \]

Arguments

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>ImaGene® Results formatted file. Enter a character vector or string specifying a file name, or a path and file name.</td>
</tr>
<tr>
<td>CleanColNamesValue</td>
<td>Controls the conversion of any ColumnNames containing spaces or characters that cannot be used as MATLAB variable names, to valid MATLAB variable names. Choices are true or false (default).</td>
</tr>
</tbody>
</table>

Description

\[ \text{imagenedata} = \text{imageneread}(\text{File}) \] reads ImaGene results data from \text{File} and creates \text{imagenedata}, a MATLAB structure containing the following fields.

<table>
<thead>
<tr>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>HeaderAA</td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>Blocks</td>
</tr>
<tr>
<td>Rows</td>
</tr>
<tr>
<td>Columns</td>
</tr>
<tr>
<td>Fields</td>
</tr>
<tr>
<td>IDs</td>
</tr>
<tr>
<td>ColumnNames</td>
</tr>
<tr>
<td>Indices</td>
</tr>
<tr>
<td>Shape</td>
</tr>
</tbody>
</table>

\[ \text{imagenedata} = \text{imageneread}(\ldots, \text{'}Property\text{', PropertyValue}, \ldots) \] calls imageneread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

controls the conversion of any ColumnNames containing spaces or characters that cannot be used as MATLAB variable names, to valid MATLAB variable names. Choices are true or false (default).
The field **Indices** of the structure contains indices that you can use for plotting heat maps of the data with the function `image` or `imagesc`.

For more details on the ImaGene format and example data, see the ImaGene documentation.

**Examples**

In the following example, the file `cy3.txt` is not provided.

1. Read in a sample ImaGene Results file. Note that the example file, `cy3.txt`, is not provided with the Bioinformatics Toolbox software.
   
   ```matlab
cy3Data = imageneread('cy3.txt');
```

2. Plot the signal mean.
   
   ```matlab
maimage(cy3Data,'Signal Mean');
```

3. Read in a sample ImaGene Results file. Note that the example file, `cy5.txt`, is not provided with the Bioinformatics Toolbox software.
   
   ```matlab
cy5Data = imageneread('cy5.txt');
```

4. Create a loglog plot of the signal median from two ImaGene Results files.
   
   ```matlab
sigMedianCol = find(strcmp('Signal Median',cy3Data.ColumnNames));
cy3Median = cy3Data.Data(:,sigMedianCol);
cy5Median = cy5Data.Data(:,sigMedianCol);
maloglog(cy3Median,cy5Median,'title','Signal Median');
```

**See Also**

`gprread` | `ilmnbsread` | `maboxplot` | `maimage` | `sptread`

**Introduced before R2006a**
**int2aa**

Convert amino acid sequence from integer to letter representation

**Syntax**

```
SeqChar = int2aa(SeqInt)
SeqChar = int2aa(SeqInt, 'Case', CaseValue)
```

**Input Arguments**

<table>
<thead>
<tr>
<th>SeqInt</th>
<th>Row vector of integers specifying an amino acid sequence. For valid integers, see the table Mapping Amino Acid Integers to Letter Codes. Integers are arbitrarily assigned to IUB/IUPAC letters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaseValue</td>
<td>Character vector or string specifying the upper or lower case. Choices are 'upper' (default) or 'lower'.</td>
</tr>
</tbody>
</table>

**Output Arguments**

| SeqChar            | Amino acid sequence specified by a character vector of single-letter codes.                                                                                                                   |

**Description**

*SeqChar* = int2aa(*SeqInt*) converts *SeqInt*, a row vector of integers specifying an amino acid sequence, to *SeqChar*, a character vector or string of single-letter codes specifying the same amino acid sequence. For valid integers, see the table Mapping Amino Acid Integers to Letter Codes.

*SeqChar* = int2aa(*SeqInt*, 'Case', *CaseValue*) specifies the upper or lower case. Choices are 'upper' (default) or 'lower'.
### Mapping Amino Acid Integers to Letter Codes

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>Integer</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alanine</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Arginine</td>
<td>2</td>
<td>R</td>
</tr>
<tr>
<td>Asparagine</td>
<td>3</td>
<td>N</td>
</tr>
<tr>
<td>Aspartic acid (Aspartate)</td>
<td>4</td>
<td>D</td>
</tr>
<tr>
<td>Asparagine</td>
<td>5</td>
<td>C</td>
</tr>
<tr>
<td>Glutamine</td>
<td>6</td>
<td>Q</td>
</tr>
<tr>
<td>Glutamic acid (Glutamate)</td>
<td>7</td>
<td>E</td>
</tr>
<tr>
<td>Glycine</td>
<td>8</td>
<td>G</td>
</tr>
<tr>
<td>Histidine</td>
<td>9</td>
<td>H</td>
</tr>
<tr>
<td>Isoleucine</td>
<td>10</td>
<td>I</td>
</tr>
<tr>
<td>Leucine</td>
<td>11</td>
<td>L</td>
</tr>
<tr>
<td>Lysine</td>
<td>12</td>
<td>K</td>
</tr>
<tr>
<td>Methionine</td>
<td>13</td>
<td>M</td>
</tr>
<tr>
<td>Phenylalanine</td>
<td>14</td>
<td>F</td>
</tr>
<tr>
<td>Proline</td>
<td>15</td>
<td>P</td>
</tr>
<tr>
<td>Serine</td>
<td>16</td>
<td>S</td>
</tr>
<tr>
<td>Threonine</td>
<td>17</td>
<td>T</td>
</tr>
<tr>
<td>Tryptophan</td>
<td>18</td>
<td>W</td>
</tr>
<tr>
<td>Tyrosine</td>
<td>19</td>
<td>Y</td>
</tr>
<tr>
<td>Valine</td>
<td>20</td>
<td>V</td>
</tr>
<tr>
<td>Asparagine or Aspartic acid (Aspartate)</td>
<td>21</td>
<td>B</td>
</tr>
<tr>
<td>Glutamine or Glutamic acid (Glutamate)</td>
<td>22</td>
<td>Z</td>
</tr>
<tr>
<td>Unknown amino acid (any amino acid)</td>
<td>23</td>
<td>X</td>
</tr>
<tr>
<td>Translation stop</td>
<td>24</td>
<td>*</td>
</tr>
<tr>
<td>Gap of indeterminate length</td>
<td>25</td>
<td>-</td>
</tr>
<tr>
<td>Unknown (any integer not in table)</td>
<td>0 or ≥ 26</td>
<td>?</td>
</tr>
</tbody>
</table>

### Examples

Convert an amino acid sequence from integer to letter representation.

```matlab
s = int2aa([13 1 17 11 1 21])
s =
```

MATLAB

### See Also

`aa2int | aminolookup | int2nt | isotopicdist | nt2int`
Introduced before R2006a
**int2nt**

Convert nucleotide sequence from integer to letter representation

**Syntax**

```
SeqChar = int2nt(SeqInt)
SeqChar = int2nt(SeqInt, ...'Alphabet', AlphabetValue, ...)
SeqChar = int2nt(SeqInt, ...'Unknown', UnknownValue, ...)
SeqChar = int2nt(SeqInt, ...'Case', CaseValue, ...)
```

**Input Arguments**

- **SeqInt**: Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes. Integers are arbitrarily assigned to IUB/IUPAC letters.
- **AlphabetValue**: Character vector or string specifying a nucleotide alphabet. Choices are:
  - 'DNA' (default) — Uses the symbols A, C, G, and T.
  - 'RNA' — Uses the symbols A, C, G, and U.
- **UnknownValue**: Character to represent unknown nucleotides, that is 0 or integers ≥ 17. Choices are any character other than the nucleotide characters A, C, G, T, and U and the ambiguous nucleotide characters N, R, Y, K, M, S, W, B, D, H, and V. Default is *.
- **CaseValue**: Character vector or string specifying the upper or lower case. Choices are 'upper' (default) or 'lower'.

**Output Arguments**

- **SeqChar**: Nucleotide sequence specified by a character vector of codes.

**Description**

```
SeqChar = int2nt(SeqInt)`` converts `SeqInt`, a row vector of integers specifying a nucleotide sequence, to `SeqChar`, a character vector of codes specifying the same nucleotide sequence. For valid codes, see the table Mapping Nucleotide Integers to Letter Codes.
## Mapping Nucleotide Integers to Letter Codes

<table>
<thead>
<tr>
<th>Nucleotide</th>
<th>Integer</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adenosine</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Cytidine</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>Guanine</td>
<td>3</td>
<td>G</td>
</tr>
<tr>
<td>Thymidine</td>
<td>4</td>
<td>T</td>
</tr>
<tr>
<td>Uridine (if 'Alphabet' set to 'RNA')</td>
<td>4</td>
<td>U</td>
</tr>
<tr>
<td>Purine (A or G)</td>
<td>5</td>
<td>R</td>
</tr>
<tr>
<td>Pyrimidine (T or C)</td>
<td>6</td>
<td>Y</td>
</tr>
<tr>
<td>Keto (G or T)</td>
<td>7</td>
<td>K</td>
</tr>
<tr>
<td>Amino (A or C)</td>
<td>8</td>
<td>M</td>
</tr>
<tr>
<td>Strong interaction (3 H bonds) (G or C)</td>
<td>9</td>
<td>S</td>
</tr>
<tr>
<td>Weak interaction (2 H bonds) (A or T)</td>
<td>10</td>
<td>W</td>
</tr>
<tr>
<td>Not A (C or G or T)</td>
<td>11</td>
<td>B</td>
</tr>
<tr>
<td>Not C (A or G or T)</td>
<td>12</td>
<td>D</td>
</tr>
<tr>
<td>Not G (A or C or T)</td>
<td>13</td>
<td>H</td>
</tr>
<tr>
<td>Not T or U (A or C or G)</td>
<td>14</td>
<td>V</td>
</tr>
<tr>
<td>Any nucleotide (A or C or G or T or U)</td>
<td>15</td>
<td>N</td>
</tr>
<tr>
<td>Gap of indeterminate length</td>
<td>16</td>
<td>-</td>
</tr>
<tr>
<td>Unknown (any integer not in table)</td>
<td>0 or ≥ 17</td>
<td>* (default)</td>
</tr>
</tbody>
</table>

`SeqChar = int2nt(SeqInt, ...PropertyName', PropertyValue, ...)` calls `int2nt` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

- `SeqChar = int2nt(SeqInt, ...'Alphabet', AlphabetValue, ...)` specifies a nucleotide alphabet. `AlphabetValue` can be 'DNA', which uses the symbols A, C, G, and T, or 'RNA', which uses the symbols A, C, G, and U. Default is 'DNA'.

- `SeqChar = int2nt(SeqInt, ...'Unknown', UnknownValue, ...)` specifies the character to represent unknown nucleotides, that is 0 or integers ≥ 17. `UnknownValue` can be any character other than the nucleotide characters A, C, G, T, and U and the ambiguous nucleotide characters N, R, Y, K, M, S, W, B, D, H, and V. Default is *.

- `SeqChar = int2nt(SeqInt, ...'Case', CaseValue, ...)` specifies the upper or lower case. `CaseValue` can be 'upper' (default) or 'lower'.

### Examples

- Convert a nucleotide sequence from integer to letter representation.

  ```matlab
  s = int2nt([1 2 4 3 2 4 1 3 2])
  ```
s = ACTGCTAGC

• Convert a nucleotide sequence from integer to letter representation and define # as the symbol for unknown numbers 17 and greater.

```matlab
si = [1 2 4 20 2 4 40 3 2];
s = int2nt(si, 'unknown', '#')
s = ACT#CT#GC
```

See Also
aa2int | baselookup | int2aa | nt2int

Introduced before R2006a
**isdag (biograph)**

Test for cycles in biograph object

**Syntax**

```matlab
isdag(BGObj)
```

**Arguments**

| BGObj | Biograph object created by biograph (object constructor). |

**Description**

**Tip** For introductory information on graph theory functions, see "Graph Theory Functions".

**isdag**(BGObj) returns logical 1 (true) if an N-by-N adjacency matrix extracted from a biograph object, BGObj, is a directed acyclic graph (DAG) and logical 0 (false) otherwise. In the N-by-N sparse matrix, all nonzero entries indicate the presence of an edge.

**References**


**See Also**

allshortestpaths | biograph | conncomp | graphisdag | isomorphism | isspantree | maxflow | minspantree | shortestpath | topoorder | traverse

**Topics**

biograph object on page 1-185

**Introduced in R2006b**
isempty

Class: bioma.data.ExptData
Package: bioma.data

Determine whether ExptData object is empty

Syntax

\[
TF = \text{isempty}(\text{EDObj})
\]

Description

\[
TF = \text{isempty}(\text{EDObj}) \text{ returns logical 1 (true) if } \text{EDObj} \text{ is an empty ExptData object. Otherwise, it returns logical 0 (false). An empty ExptData object contains no data elements.}
\]

Input Arguments

EDObj

Object of the bioma.data.ExptData class.

Default:

Examples

Construct an ExptData object, and then check to see if it is empty:

\[
\begin{align*}
% \text{Import bioma.data package to make constructor functions } \\
% \text{available} \\
\text{import bioma.data.*} \\
% \text{Create DataMatrix object from .txt file containing} \\
% \text{expression values from microarray experiment} \\
\text{dmObj = DataMatrix('File', 'mouseExprsData.txt');} \\
% \text{Construct ExptData object} \\
\text{EDObj = ExptData(dmObj);} \\
% \text{Determine if ExptData object is empty} \\
\text{isempty(EDObj)}
\end{align*}
\]

See Also

bioma.data.ExptData

Topics

“Representing Expression Data Values in ExptData Objects”
isempty

Class: bioma.data.MetaData
Package: bioma.data

Determine whether MetaData object is empty

Syntax

\[ TF = \text{isempty}(MDObj) \]

Description

\[ TF = \text{isempty}(MDObj) \] returns logical 1 (true) if \( MDObj \) is an empty MetaData object. Otherwise, it returns logical 0 (false). An empty MetaData object contains no variable names, values, or descriptions.

Input Arguments

MDObj

Object of the bioma.data.MetaData class.

Default:

Examples

Construct a MetaData object, and then check to see if it is empty:

% Import bioma.data package to make constructor function available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Determine if MetaData object is empty
isempty(MDObj2)

See Also
bioma.data.MetaData

Topics
"Representing Sample and Feature Metadata in MetaData Objects"
isempty

Class: bioma.data.MIAME
Package: bioma.data

Determine whether MIAME object is empty

Syntax

\[ TF = \text{isempty}(\text{MIAMEObj}) \]

Description

\[ TF = \text{isempty}(\text{MIAMEObj}) \] returns logical 1 (true) if \text{MIAMEObj} is an empty MIAME object. Otherwise, it returns logical 0 (false). All properties are empty in an empty MIAME object.

Input Arguments

\text{MIAMEObj}

Object of the bioma.data.MIAME class.

Default:

Examples

Construct a MIAME object, and then check to see if it is empty:

% Create a MATLAB structure containing GEO Series data
geoStruct = getgeodata('GSE4616');
% Import bioma.data package to make constructor function available
import bioma.data.*
% Construct MIAME object
MIAMEObj = MIAME(geoStruct);
% Determine if MIAME object is empty
isempty(MIAMEObj)

See Also

bioma.data.MIAME

Topics

“Representing Experiment Information in a MIAME Object”
isequal (DataMatrix)

Test DataMatrix objects for equality

Syntax

\[
TF = \text{isequal}(DMObj1, DMObj2) \\
TF = \text{isequal}(DMObj1, DMObj2, DMObj3, \ldots)
\]

Input Arguments

| DMObj1, DMObj2, DMObj3 | DataMatrix objects, such as created by DataMatrix (object constructor). |

Output Arguments

| TF | Logical value indicating if inputs are numerically equal (have the same contents), have the same size (same NRows and NCols properties), and have the same RowNames and ColNames properties. NaNs are not considered equal to each other. |

Description

\[
TF = \text{isequal}(DMObj1, DMObj2) \text{ returns logical } 1 (\text{true}) \text{ if the input DataMatrix objects, } DMObj1 \text{ and } DMObj2, \text{ meet the following:}
\]

- Are numerically equal (have the same contents)
- Have the same size (same NRows and NCols properties)
- Have the same RowNames and ColNames properties

Otherwise, it returns logical 0 (false). DMObj1 and DMObj2 do not have to have the same Name property. NaNs are not considered equal to each other.

\[
TF = \text{isequal}(DMObj1, DMObj2, DMObj3, \ldots) \text{ returns logical } 1 (\text{true}) \text{ if all input DataMatrix objects, } DMObj1, DMObj2, DMObj3, \text{ etc. meet the following:}
\]

- Are numerically equal (have the same contents)
- Have the same size (same NRows and NCols properties)
- Have the same RowNames and ColNames properties

Otherwise, it returns logical 0 (false). The input DataMatrix objects do not have to have the same Name property. NaNs are not considered equal to each other.

See Also

DataMatrix | isequaln

Topics

DataMatrix object on page 1-532
Introduced in R2008b
isequaln (DataMatrix)

Test DataMatrix objects for equality, treating NaNs as equal

Syntax

\[ TF = \text{isequaln}(\text{DMObj1}, \text{DMObj2}) \]
\[ TF = \text{isequaln}(\text{DMObj1}, \text{DMObj2}, \text{DMObj3}, ...) \]

Input Arguments

| DMObj1, DMObj2, DMObj3 | DataMatrix objects, such as created by DataMatrix (object constructor). |

Output Arguments

| TF | Logical value indicating if inputs are numerically equal (have the same contents), have the same size (same NRows and NCols properties), and have the same RowNames and ColNames properties. NaNs are considered equal to each other. |

Description

\[ TF = \text{isequaln}(\text{DMObj1}, \text{DMObj2}) \] returns logical 1 (true) if the input DataMatrix objects, \text{DMObj1} and \text{DMObj2}, meet the following:

• Are numerically equal (have the same contents)
• Have the same size (same NRows and NCols properties)
• Have the same RowNames and ColNames properties

Otherwise, it returns logical 0 (false). \text{DMObj1} and \text{DMObj2} do not need to have the same Name property. NaNs are considered equal to each other.

\[ TF = \text{isequaln}(\text{DMObj1}, \text{DMObj2}, \text{DMObj3}, ...) \] returns logical 1 (true) if all input DataMatrix objects, \text{DMObj1}, \text{DMObj2}, \text{DMObj3}, etc. meet the following:

• Are numerically equal (have the same contents)
• Have the same size (same NRows and NCols properties)
• Have the same RowNames and ColNames properties

Otherwise, it returns logical 0 (false). The input DataMatrix objects do not need to have the same Name property. NaNs are considered equal to each other.

See Also

DataMatrix, isequal

Topics

DataMatrix object on page 1-532
Introduced in R2012b
isequalwithequalnans (DataMatrix)

Test DataMatrix objects for equality, treating NaNs as equal

Syntax

\[ TF = \text{isequalwithequalnans}(\text{DMObj1}, \text{DMObj2}) \]
\[ TF = \text{isequalwithequalnans}(\text{DMObj1}, \text{DMObj2}, \text{DMObj3}, \ldots) \]

Input Arguments

\[
\begin{array}{|c|}
\hline
\text{DMObj1, DMObj2, DMObj3} \\
\hline
\end{array}
\]

DataMatrix objects, such as created by DataMatrix (object constructor).

Output Arguments

\[
\begin{array}{|c|}
\hline
\text{TF} \\
\hline
\end{array}
\]

Logical value indicating if inputs are numerically equal (have the same contents), have the same size (same NRows and NCols properties), and have the same RowNames and ColNames properties. NaNs are considered equal to each other.

Description

\[ TF = \text{isequalwithequalnans}(\text{DMObj1}, \text{DMObj2}) \] returns logical 1 (true) if the input DataMatrix objects, \( \text{DMObj1} \) and \( \text{DMObj2} \), meet the following:

- Are numerically equal (have the same contents)
- Have the same size (same NRows and NCols properties)
- Have the same RowNames and ColNames properties

Otherwise, it returns logical 0 (false). \( \text{DMObj1} \) and \( \text{DMObj2} \) do not have to have the same Name property. NaNs are considered equal to each other.

\[ TF = \text{isequalwithequalnans}(\text{DMObj1}, \text{DMObj2}, \text{DMObj3}, \ldots) \] returns logical 1 (true) if all input DataMatrix objects, \( \text{DMObj1}, \text{DMObj2}, \text{DMObj3}, \ldots \), meet the following:

- Are numerically equal (have the same contents)
- Have the same size (same NRows and NCols properties)
- Have the same RowNames and ColNames properties

Otherwise, it returns logical 0 (false). The input DataMatrix objects do not have to have the same Name property. NaNs are considered equal to each other.

See Also

DataMatrix | isequal

Topics
DataMatrix object on page 1-532
Introduced in R2008b
isoelectric

Estimate isoelectric point for amino acid sequence

Syntax

\[ pI = \text{isoelectric}(\text{SeqAA}) \]
\[ [pI \text{ Charge}] = \text{isoelectric}(\text{SeqAA}) \]

\text{isoelectric}(\ldots, '\text{PropertyName}', \text{PropertyValue},\ldots)
\text{isoelectric}(\ldots, 'PKVals', \text{PKValsValue})
\text{isoelectric}(\ldots, 'Charge', \text{ChargeValue})
\text{isoelectric}(\ldots, 'Chart', \text{ChartValue})

Arguments

| SeqAA | Amino acid sequence. Enter a character vector, string, or a vector of integers from the table Mapping Amino Acid Letter Codes to Integers. Examples: 'ARN' or [1 2 3]. |
| PKValsValue | Character vector or string specifying a file name or path and file name of a PK file containing a table of pK values for amino acids, which \text{isoelectric} uses to estimate the isoelectric point (pI) of an amino acid sequence. For an example of a PK file format, type edit Emboss.pK in the MATLAB command line. |
| ChargeValue | Property to select a specific pH for estimating charge. Enter a number between 0 and 14. Default is 7.2. |
| ChartValue | Controls the plotting a graph of charge versus pH. Enter true or false. |

Description

\( pI = \text{isoelectric}(\text{SeqAA}) \) returns the estimated isoelectric point (pI) for an amino acid sequence using the following pK values:

\[
\begin{array}{ll}
\text{N\_term} & 8.6 \\
\text{K} & 10.8 \\
\text{R} & 12.5 \\
\text{H} & 6.5 \\
\text{D} & 3.9 \\
\text{E} & 4.1 \\
\text{C} & 8.5 \\
\text{Y} & 10.1 \\
\text{C\_term} & 3.6
\end{array}
\]

The isoelectric point is the pH at which the protein has a net charge of zero.

\( [pI \text{ Charge}] = \text{isoelectric}(\text{SeqAA}) \) returns the estimated isoelectric point (pI) for an amino acid sequence and the estimated charge for a given pH (default is typical intracellular pH 7.2).

The estimates are skewed by the underlying assumptions that all amino acids are fully exposed to the solvent, that neighboring peptides have no influence on the pK of any given amino acid, and that the
constitutive amino acids, as well as the N- and C-termini, are unmodified. Cysteine residues participating in disulfide bridges also affect the true pI and are not considered here. By default, isoelectric uses the EMBOSS amino acid pK table, or you can substitute other values using the property PKVals.

- If the sequence contains ambiguous amino acid characters (b z * -), isoelectric ignores the characters and displays a warning message.
  
  Warning: Symbols other than the standard 20 amino acids appear in the sequence.
- If the sequence contains undefined amino acid characters (i j o), isoelectric ignores the characters and displays a warning message.
  
  Warning: Sequence contains unknown characters. These will be ignored.

isolectric(..., 'PropertyName', PropertyValue,...) defines optional properties using property name/value pairs.

isolectric(..., 'PKVals', PKValsValue) uses pK values stored in a PKValValues, a PK file, to estimate the isoelectric point (pI) of an amino acid sequence. For an example of a PK file format, type edit Emboss.pK in the MATLAB command line.

isolectric(..., 'Charge', ChargeValue) returns the estimated charge of a sequence for a given pH (ChargeValue).

isolectric(..., 'Chart', ChartValue) when ChartValue is true, returns a graph plotting the charge of the protein versus the pH of the solvent.

**Examples**

% Get a sequence from PDB.
pdbSeq = getpdb('1CIV', 'SequenceOnly', true)
% Estimate its isoelectric point.
isoelectric(pdbSeq)

% Plot the charge against the pH for a short polypeptide sequence.
isoelectric('PQGGGwqphGGGwqPhGGGGwQGSHGq', 'CHART', true)

% Get the Rh blood group D antigen from NCBI and calculate % its charge at pH 7.3 (typical blood pH).
gpSeq = getgenpept('AAB39602')
[pI Charge] = isoelectric(gpSeq, 'Charge', 7.38)

**See Also**

aacount | molweight

Introduced before R2006a
isomorphism (biograph)

Find isomorphism between two biograph objects

Syntax

\[
\text{[Isomorphic, Map]} = \text{isomorphism}(\text{BGObj1}, \text{BGObj2})
\]

\[
\text{[Isomorphic, Map]} = \text{isomorphism}(\text{BGObj1}, \text{BGObj2},'\text{Directed}', \text{DirectedValue})
\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGObj1</td>
<td>Biograph object created by biograph (object constructor).</td>
</tr>
<tr>
<td>BGObj2</td>
<td>Biograph object created by biograph (object constructor).</td>
</tr>
<tr>
<td>DirectedValue</td>
<td>Property that indicates whether the graphs are directed or undirected. Enter false when both BGObj1 and BGObj2 produce undirected graphs. In this case, the upper triangles of the sparse matrices extracted from BGObj1 and BGObj2 are ignored. Default is true, meaning that both graphs are directed.</td>
</tr>
</tbody>
</table>

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions”.

\[
\text{[Isomorphic, Map]} = \text{isomorphism}(\text{BGObj1}, \text{BGObj2})
\]

returns logical 1 (true) in Isomorphic if two N-by-N adjacency matrices extracted from biograph objects BGObj1 and BGObj2 are isomorphic graphs, and logical 0 (false) otherwise. A graph isomorphism is a 1-to-1 mapping of the nodes in the graph from BGObj1 and the nodes in the graph from BGObj2 such that adjacencies are preserved. Return value Isomorphic is Boolean. When Isomorphic is true, Map is a row vector containing the node indices that map from BGObj2 to BGObj1. When Isomorphic is false, the worst-case time complexity is O(N!), where N is the number of nodes.

\[
\text{[Isomorphic, Map]} = \text{isomorphism}(\text{BGObj1}, \text{BGObj2},'\text{Directed}', \text{DirectedValue})
\]

indicates whether the graphs are directed or undirected. Set DirectedValue to false when both BGObj1 and BGObj2 produce undirected graphs. In this case, the upper triangles of the sparse matrices extracted from BGObj1 and BGObj2 are ignored. The default is true, meaning that both graphs are directed.

References


See Also
allshortestpaths | biograph | conncomp | graphisomorphism | isdag | isspantree |
maxflow | minspantree | shortestpath | topoorder | traverse

Topics
biograph object on page 1-185

Introduced in R2006b
**isotopicdist**

Calculate high-resolution isotope mass distribution and density function

**Syntax**

\[
\text{isotopicdist}(\text{SeqAA})
\]

\[
\text{isotopicdist}(\text{Compound})
\]

\[
\text{isotopicdist}(\text{Formula})
\]

\[
\text{isotopicdist}(..., \text{NTerminal}', \text{NTerminalValue}, ...)
\]

\[
\text{isotopicdist}(..., \text{CTerminal}', \text{CTerminalValue}, ...)
\]

\[
\text{isotopicdist}(..., \text{Resolution}', \text{ResolutionValue}, ...)
\]

\[
\text{isotopicdist}(..., \text{FFTResolution}', \text{FFTResolutionValue}, ...)
\]

\[
\text{isotopicdist}(..., \text{FFTRange}', \text{FFTRangeValue}, ...)
\]

\[
\text{isotopicdist}(..., \text{FFTLocation}', \text{FFTLocationValue}, ...)
\]

\[
\text{isotopicdist}(..., \text{NoiseThreshold}', \text{NoiseThresholdValue}, ...)
\]

\[
\text{isotopicdist}(..., \text{ShowPlot}', \text{ShowPlotValue}, ...)
\]

**Description**

\[
\text{isotopicdist}(\text{SeqAA})
\]

analyzes a peptide sequence and returns a matrix containing the expected mass distribution; a structure containing the monoisotopic mass, average mass, most abundant mass, nominal mass, and empirical formula; and a matrix containing the expected density function.

\[
\text{isotopicdist}(\text{Compound})
\]

analyzes a compound specified by a numeric vector or matrix.

\[
\text{isotopicdist}(\text{Formula})
\]

analyzes a compound specified by an empirical chemical formula represented by the structure Formula. The field names in Formula must be valid element symbols and are case sensitive. The respective values in Formula are the number of atoms for each element. Formula can also be an array of structures that specifies multiple formulas. The field names can be in any order within a structure. However, if there are multiple structures, the order must be the same in each.

\[
\text{isotopicdist}(..., \text{PropertyName}', \text{PropertyValue}, ...)
\]

calls isotopicdist with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each PropertyName in single quotation marks. Each PropertyName is case insensitive. These property name/property value pairs are as follows:

\[
\text{isotopicdist}(..., \text{NTerminal}', \text{NTerminalValue}, ...)
\]

modifies the N-terminal of the peptide.

\[
\text{isotopicdist}(..., \text{CTerminal}', \text{CTerminalValue}, ...)
\]

modifies the C-terminal of the peptide.

\[
\text{isotopicdist}(..., \text{Resolution}', \text{ResolutionValue}, ...)
\]

specifies the approximate resolution of the instrument, given as the Gaussian width (in daltons) at full width at half height (FWHH).
isotopicdist(..., 'FFTResolution', FFTResolutionValue, ...) specifies the number of data points per dalton, to compute the fast Fourier transform (FFT) algorithm.

isotopicdist(..., 'FFTRange', FFTRangeValue, ...) specifies the absolute range (window size) in daltons for the FFT algorithm and output density function.

isotopicdist(..., 'FFTLocation', FFTLocationValue, ...) specifies the location of the FFT range (window) defined by FFTRangeValue. It specifies this location by setting the location of the lower limit of the range, relative to the location of the monoisotopic peak, which is computed by isotopicdist.

isotopicdist(..., 'NoiseThreshold', NoiseThresholdValue, ...) removes points in the mass distribution that are smaller than \(1/\text{NoiseThresholdValue}\) times the most abundant mass.

isotopicdist(..., 'ShowPlot', ShowPlotValue, ...) controls the display of a plot of the mass distribution.

**Input Arguments**

**SeqAA**

Peptide sequence specified by either a:

- Character vector or string of single-letter codes
- Cell array of character vectors or string vector that specifies multiple peptide sequences

**Tip** You can use the `getgenpept` and `genpeptread` functions to retrieve peptide sequences from the GenPept database or a GenPept-formatted file. You can then use the `cleave` function to perform an insilico digestion on a peptide sequence. The `cleave` function creates a cell array of character vectors representing peptide fragments, which you can submit to the `isotopicdist` function.

**Default:**

**Compound**

Compound specified by either a:

- Numeric vector of form \([C \ H \ N \ O \ S]\), where C, H, N, O, and S are nonnegative numbers that represent the number of atoms of carbon, hydrogen, nitrogen, oxygen, and sulfur respectively in a compound.
- M-by-5 numeric matrix that specifies multiple compounds, with each row corresponding to a compound and each column corresponding to an atom.

**Default:**

**Formula**

Chemical formula specified by either a:

- Structure whose field names are valid element symbols and case sensitive. Their respective values are the number of atoms for each element.
• Array of structures that specifies multiple formulas.

**Note** If Formula is a single structure, the order of the fields does not matter. If Formula is an array of structures, then the order of the fields must be the same in each structure.

**Default:**

**NTerminalValue**
Modification for the N-terminal of the peptide, specified by either:

- One of 'none', 'amine' (default), 'formyl', or 'acetyl'
- Custom modification specified by an empirical formula, represented by a structure. The structure must have field names that are valid element symbols and case sensitive. Their respective values are the number of atoms for each element.

**CTerminalValue**
Modification for the C-terminal of the peptide, specified by either:

- One of 'none', 'freeacid' (default), or 'amide'
- Custom modification specified by an empirical formula, represented by a structure. The structure must have field names that are valid element symbols and case sensitive. Their respective values are the number of atoms for each element.

**ResolutionValue**
Value in daltons specifying the approximate resolution of the instrument, given as the Gaussian width at full width half height (FWHH).

**Default:** 1/16 Da

**FFTResolutionValue**
Value specifying the number of data points per dalton, used to compute the FFT algorithm.

**Default:** 1000

**FFTRangeValue**
Value specifying the absolute range (window size) in daltons for the FFT algorithm and output density function. By default, this value is automatically estimated based on the weight of the molecule. The actual FFT range used internally by isotopicdist is further increased such that \( \text{FFTRangeValue} \times \text{FFTResolutionValue} \) is a power of two.

**Tip** Increase the **FFTRangeValue** if the signal represented by the DF output appears to be truncated.

**Tip** Ultrahigh resolution allows you to resolve micropeaks that have the same nominal mass, but slightly different exact masses. To achieve ultrahigh resolution, increase **FFTResolutionValue** and reduce **ResolutionValue**, but ensure that \( \text{FFTRangeValue} \times \text{FFTResolutionValue} \) is within the available memory.
Default:

**FFTLocationValue**

Fraction that specifies the location of the FFT range (window) defined by **FFTRangeValue**. It specifies this location by setting the location of the lower limit of the FFT range, relative to the location of the monoisotopic peak, which is computed by `isotopicdist`. The location of the lower limit of the FFT range is set to the mass of the monoisotopic peak - (FFTLocationValue * FFTRangeValue).

**Tip** You may need to shift the FFT range to the left in rare cases where a compound contains an element, such as Iron or Argon, whose most abundant isotope is not the lightest one.

Default: 1/16

**NoiseThresholdValue**

Value that removes points in the mass distribution that are smaller than 1/NoiseThresholdValue times the most abundant mass.

Default: 1e6

**ShowPlotValue**

Controls the display of a plot of the isotopic mass distribution. Choices are true, false, or I, which is an integer specifying a compound. If set to true, the first compound is plotted. Default is:

- false — When you specify return values.
- true — When you do not specify return values.

Default:

**Output Arguments**

**MD**

Mass distribution represented by a two-column matrix in which each row corresponds to an isotope. The first column lists the isotopic mass, and the second column lists the probability for that mass.

**Info**

Structure containing mass information for the peptide sequence or compound in the following fields:

- **NominalMass**
- **MonoisotopicMass**
- **ObservedAverageMass** — Estimated from the DF signal output, using instrument resolution specified by the 'Resolution' property.
- **CalculatedAverageMass** — Calculated directly from the input formula, assuming perfect instrument resolution.
- **MostAbundantMass**
• **Formula** — Structure containing the number of atoms of each element.

**DF**

Density function represented by a two-column matrix in which each row corresponds to an m/z value. The first column lists the mass, and the second column lists the relative intensity of the signal at that mass.

**Examples**

Calculate and display the isotopic mass distribution of the peptide sequence `MATLAP` with an Acetyl N-terminal and an Amide C-terminal:

```matlab
MD = isotopicdist('MATLAP','nterm','Acetyl','cterm','Amide', ... 'showplot',true)
```

```
MD =

643.3363    0.6676
644.3388    0.2306
645.3378    0.0797
646.3386    0.0181
647.3396    0.0033
648.3409    0.0005
649.3423    0.0001
650.3439    0.0000
651.3455    0.0000
```

Calculate and display the isotopic mass distribution of Glutamine \((\text{C}_5\text{H}_{10}\text{N}_2\text{O}_3)\):

```matlab
MD = isotopicdist([5 10 2 3 0],'showplot',true)
```
Display the isotopic mass distribution of the "averagine" model, whose molecular formula represents the statistical occurrences of amino acids from all known proteins:

\texttt{isotopicdist([4.9384 7.7583 1.3577 1.4773 0.0417])}
More About

**Average Mass**

Sum of the average atomic masses of the constituent elements in a molecule.

**Monoisotopic Mass**

Sum of the masses of the atoms in a molecule using the unbound, ground-state, rest mass of the principle (most abundant) isotope for each element instead of the isotopic average mass.

**Most Abundant Mass**

Mass of the molecule with the most-highly represented isotope distribution, based on the natural abundance of the isotopes.

**Nominal Mass**

Sum of the integer masses (ignoring the mass defect) of the most abundant isotope of each element in a molecule.

**References**


1-980


See Also
aminolookup | cleave | cleavelookup | genpeptread | getgenpept | int2aa | molweight | nt2aa

Introduced in R2009b
**isspantree (biograph)**

Determine if tree created from biograph object is spanning tree

**Syntax**

\[ TF = 	ext{isspantree}(	ext{BGObj}) \]

**Arguments**

| \text{BGObj} | Biograph object created by \text{biograph} (object constructor). |

**Description**

**Tip** For introductory information on graph theory functions, see “Graph Theory Functions”.

\[ TF = 	ext{isspantree}(	ext{BGObj}) \] returns logical 1 (true) if the N-by-N adjacency matrix extracted from a biograph object, \text{BGObj}, is a spanning tree, and logical 0 (false) otherwise. A spanning tree must touch all the nodes and must be acyclic. The lower triangle of the N-by-N adjacency matrix represents an undirected graph, and all nonzero entries indicate the presence of an edge.

**Note** The function ignores the direction of the edges in the Biograph object.

**References**


**See Also**

allshortestpaths | biograph | conncomp | graphisspantree | isdag | isomorphism | maxflow | minspantree | shortestpath | topoorder | traverse

**Topics**

biograph object on page 1-185

**Introduced in R2006b**
**jcampread**

Read JCAMP-DX-formatted files

**Syntax**

`JCAMPStruct = jcampread(File)`

**Input Arguments**

<table>
<thead>
<tr>
<th>File</th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a JCAMP-DX-formatted file (ASCII text file). If you specify only a file name, that file must be on the MATLAB search path or in the current folder.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB character array that contains the text of a JCAMP-DX-formatted file.</td>
</tr>
</tbody>
</table>

**Output Arguments**

| `JCAMPStruct` | MATLAB structure containing information from a JCAMP-DX-formatted file. |

**Description**

JCAMP-DX is a file format for infrared, NMR, and mass spectrometry data from the Joint Committee on Atomic and Molecular Physical Data (JCAMP). `jcampread` supports reading data from files saved with Versions 4.24, 5, or 6 of the JCAMP-DX format. For more details, see:

http://www.jcamp-dx.org/

`JCAMPStruct = jcampread(File)` reads data from `File`, a JCAMP-DX-formatted file, and creates `JCAMPStruct`, a MATLAB structure containing the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td></td>
</tr>
<tr>
<td>DataType</td>
<td></td>
</tr>
<tr>
<td>DataClass (version 5.00 and above)</td>
<td></td>
</tr>
<tr>
<td>Origin</td>
<td></td>
</tr>
<tr>
<td>Owner</td>
<td></td>
</tr>
<tr>
<td>Blocks</td>
<td></td>
</tr>
<tr>
<td>Notes</td>
<td></td>
</tr>
</tbody>
</table>

The `Blocks` field of the structure is an array of structures corresponding to each set of data in the file. These structures have the following fields.
2. Download the testdata.zip file to your MATLAB Current Folder.
3. Extract isas_ms1.dx, a JCAMP-DX-formatted file, from the testdata.zip file to your MATLAB Current Folder.
4. Read the data from the JCAMP-DX-formatted file, isas_ms1.dx, into the MATLAB software
   
   ```matlab
   jcampStruct = jcampread('isas_ms1.dx')
   
   jcampStruct =
   
   Title: '2-Chlorphenol'
   DataType: 'MASS SPECTRUM'
   DataClass: 'PEAKTABLE'
   Origin: 'H. Mayer, ISAS Dortmund'
   Owner: 'COPYRIGHT (C) 1993 by ISAS Dortmund, FRG'
   Blocks: [1x1 struct]
   Notes: {8x2 cell}
   
5. Plot the mass spectrum.

   ```matlab
   data = jcampStruct.Blocks(1);
   stem(data.XData, data.YData, '.', 'MarkerEdgeColor', 'w');
   title(jcampStruct.Title);
   xlabel(data.XUnits);
   ylabel(data.YUnits);`
See Also
mslowess | mssgolay | msviewer | mzcdfread | mzxmlread | tgspcread

Introduced before R2006a
joinseq

Join two sequences to produce shortest supersequence

Syntax

\[ \text{SeqNT3} = \text{joinseq} (\text{SeqNT1}, \text{SeqNT2}) \]

Arguments

<table>
<thead>
<tr>
<th>( \text{SeqNT1, SeqNT2} )</th>
<th>Nucleotide sequences. Enter a character vector or string for each sequence.</th>
</tr>
</thead>
</table>

Description

\( \text{SeqNT3} = \text{joinseq}(\text{SeqNT1}, \text{SeqNT2}) \) creates a new sequence that is the shortest supersequence of \( \text{SeqNT1} \) and \( \text{SeqNT2} \). If there is no overlap between the sequences, then \( \text{SeqNT2} \) is concatenated to the end of \( \text{SeqNT1} \). If the length of the overlap is the same at both ends of the sequence, then the overlap at the end of \( \text{SeqNT1} \) and the start of \( \text{SeqNT2} \) is used to join the sequences.

If \( \text{SeqNT1} \) is a subsequence of \( \text{SeqNT2} \), then \( \text{SeqNT2} \) is returned as the shortest supersequence and vice versa.

Examples

Join two sequences that contain an overlap.

```matlab
seq1 = 'ACGTAAA';
seq2 = 'AAATGCA';
joined = joinseq(seq1,seq2)
```

\[
\text{joined} = \text{ACGTAAATGCA}
\]

See Also

cat | strcat | strfind

Introduced before R2006a
**knnimpute**

Impute missing data using nearest-neighbor method

**Syntax**

```matlab
imputedData = knnimpute(data)
imputedData = knnimpute(data,k)
imputedData = knnimpute(data,k,Name,Value)
```

**Description**

`imputedData = knnimpute(data)` returns `imputedData` after replacing NaNs in the input data with the corresponding value from the nearest-neighbor column. If the corresponding value from the nearest-neighbor column is also NaN, the next nearest column is used. The function calculates the Euclidean distance between observation columns by using only the rows with no NaN values. Thus, the data must have at least one row that contains no NaN.

`imputedData = knnimpute(data,k)` replaces NaNs in `Data` with a weighted mean of the k nearest-neighbor columns. The weights are inversely proportional to the distances from the neighboring columns.

`imputedData = knnimpute(data,k,Name,Value)` uses additional options specified by one or more name-value pair arguments. For example, `imputedData = knnimpute(data,k,'Distance','mahalanobis')` uses the Mahalanobis distance to compute the nearest-neighbor columns.

**Examples**

**Impute Missing Data Using KNN**

The function `knnimpute` replaces NaNs in the input data with the corresponding value from the nearest-neighbor column. Consider the following matrix.

```matlab
A = [1 2 5;4 5 7;NaN -1 8;7 6 0]
```

A = 4×3

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>NaN</td>
<td>-1</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

A(3,1) is NaN, and because column 2 is the closest column to column 1 in the Euclidean distance, `knnimpute` replaces the (3,1) entry of column 1 with the corresponding entry from column 2, which is -1.

```matlab
results = knnimpute(A)
```

results = 4×3

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>NaN</td>
<td>-1</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>
The data must have at least one row without any NaN values for knnimpute to work. If all rows have NaN values, you can add a row where every observation (column) has identical values and call knnimpute on the updated matrix to replace the NaN values with the average of all column values for a given row.

```matlab
B = [NaN 2 1; 3 NaN 1; 1 8 NaN]
B = 3×3
    NaN   2   1
    3  NaN   1
    1   8  NaN

B(4,:) = ones(1,3)
B = 4×3
    NaN   2   1
    3  NaN   1
    1   8  NaN
    1   1   1

imputed = knnimpute(B)
imputed = 4×3
    1.5000   2.0000   1.0000
    3.0000   2.0000   1.0000
    1.0000   8.0000   4.5000
    1.0000   1.0000   1.0000
```

You can then remove the added row.

```matlab
imputed(4,:) = []
imputed = 3×3
    1.5000   2.0000   1.0000
    3.0000   2.0000   1.0000
    1.0000   8.0000   4.5000
```

Load a sample biological data set and imputes missing values in yeastvalues, where each row represents each gene and each column represents an experimental condition or observation.

```matlab
load yeastdata
Remove data for empty spots where gene labels are set to 'EMPTY'.
emptySpots = strcmp('EMPTY',genes);
yeastvalues(emptySpots,:) = [];
```
knnimpute uses the next nearest column if the corresponding value from the nearest-neighbor column is also NaN. However, if all columns are NaNs, the function generates a warning for each row and keeps the rows instead of deleting the whole row in the returned output. The sample data contains some rows with all NaNs. Remove those rows to avoid the warnings.

```matlab
eyeastvalues(~any(~isnan(yeastvalues),2),:) = [];
```

Impute missing values.

```matlab
imputedData1 = knnimpute(yeastvalues);
```

Check if there any NaN left after imputing data.

```matlab
sum(any(isnan(imputedData1),2))
an = 0
```

Use the 5-nearest neighbor search to get the nearest column.

```matlab
imputedData2 = knnimpute(yeastvalues,5);
```

Change the distance metric to use the Minknowski distance.

```matlab
imputedData3 = knnimpute(yeastvalues,5,'Distance','minkowski');
```

You can also specify the parameter for the distance metric. For instance, specify a different exponent (say 5) for the Minkowski distance.

```matlab
imputedData4 = knnimpute(yeastvalues,5,'Distance','minkowski','DistArgs',5);
```

### Input Arguments

- **data** — Input data matrix
  
  Input data, specified as a matrix. The data must have at least one row that contains no NaN because the function calculates the Euclidean distance between observation columns by using only the rows with no NaN values.
  
  Data Types: double

- **k** — Number of nearest neighbors positive integer
  
  Number of nearest neighbors, specified as a positive integer.
  
  Data Types: double

### Name-Value Pair Arguments

Specify optional comma-separated pairs of **Name,Value** arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as **Name1,Value1,...,NameN,ValueN**.

Example: `imputedData = knnimpute(data,k,'Distance','mahalanobis')`

1-989
Distance — Distance metric

class character vector | string | function handle

Distance metric, specified as a character vector, string, or function handle, as described in the following table.

Use the 'DistArgs' name-value pair in conjunction to specify parameters for the distance function. For instance, to specify a different exponent (say 5) for the Minkowski distance, use: `output = knnimpute(data,3,'Distance','minkowski','DistArgs',5).

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'euclidean'</td>
<td>Euclidean distance (default).</td>
</tr>
<tr>
<td>'squaredeuclidean'</td>
<td>Squared Euclidean distance. (This option is provided for efficiency only. It does not satisfy the triangle inequality.)</td>
</tr>
<tr>
<td>'seuclidean'</td>
<td>Standardized Euclidean distance. Each coordinate difference between observations is scaled by dividing by the corresponding element of the standard deviation, ( S = \text{nanstd}(X) ). Use 'DistArgs' to specify another value for ( S ).</td>
</tr>
<tr>
<td>'mahalanobis'</td>
<td>Mahalanobis distance using the sample covariance of ( X, C = \text{nancov}(X) ). Use 'DistArgs' to specify another value for ( C ), where the matrix ( C ) is symmetric and positive definite.</td>
</tr>
<tr>
<td>'cityblock'</td>
<td>City block distance.</td>
</tr>
<tr>
<td>'minkowski'</td>
<td>Minkowski distance. The default exponent is 2. Use DistParameter to specify a different exponent ( P ), where ( P ) is a positive scalar value of the exponent.</td>
</tr>
<tr>
<td>'chebychev'</td>
<td>Chebychev distance (maximum coordinate difference).</td>
</tr>
<tr>
<td>'cosine'</td>
<td>One minus the cosine of the included angle between points (treated as vectors).</td>
</tr>
<tr>
<td>'correlation'</td>
<td>One minus the sample correlation between points (treated as sequences of values).</td>
</tr>
<tr>
<td>'hamming'</td>
<td>Hamming distance, which is the percentage of coordinates that differ.</td>
</tr>
<tr>
<td>'jaccard'</td>
<td>One minus the Jaccard coefficient, which is the percentage of nonzero coordinates that differ.</td>
</tr>
<tr>
<td>'spearman'</td>
<td>One minus the sample Spearman's rank correlation between observations (treated as sequences of values).</td>
</tr>
</tbody>
</table>
Custom distance function handle. A distance function has the form

```matlab
function D2 = distfun(ZI,ZJ)
% calculation of distance
...
```

where

- `ZI` is a 1-by-\(n\) vector containing a single observation.
- `ZJ` is an \(m_2\)-by-\(n\) matrix containing multiple observations. `distfun` must accept a matrix `ZJ` with an arbitrary number of observations.
- `D2` is an \(m_2\)-by-1 vector of distances, and \(D2(k)\) is the distance between observations `ZI` and `ZJ(k,:)`.

If your data is not sparse, you can generally compute distance more quickly by using a built-in distance instead of a function handle.

### DistArgs — Distance metric parameter values

Distance metric parameter values, specified as a positive scalar or cell array of values. Use `'DistArgs'` together with `'Distance'` to specify parameters for the distance function. For instance, to specify a different exponent (say 5) for the Minkowski distance, use:

```matlab
output = knnimpute(data,3,'Distance','minkowski','DistArgs',5)
```

Example: `'DistArgs',3`

Data Types: `double` | `cell`

### Weights — Weights used in weighted mean calculation

Numeric vector of length \(k\)

Weights used in the weighted mean calculation, specified as a numeric vector of length \(k\).

Example: `'Weights',[0.3 0.5 0.2]`

Data Types: `double`

### Median — Flag to use median of \(k\) nearest neighbors

`true` | `false`

Flag to use the median of \(k\) nearest neighbors instead of the weighted mean, specified as `true` or `false`.

Example: `'Median',true`

Data Types: `logical`
Output Arguments

**imputedData** — Results after replacing NaNs
numeric matrix

Results after replacing NaNs from the input data with the corresponding value from the nearest-neighbor column, returned as a numeric matrix.

References


See Also

`isnan` | `nanmean` | `nanmedian` | `pdist`

Introduced before R2006a
Idivide (DataMatrix)

Left array divide DataMatrix objects

Syntax

DMObjNew  = ldivide(DMObj1, DMObj2)
DMObjNew  = DMObj1 ./ DMObj2
DMObjNew  = ldivide(DMObj1, B)
DMObjNew  = DMObj1 ./ B
DMObjNew  = ldivide(B, DMObj1)
DMObjNew  = B ./ DMObj1

Input Arguments

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>MATLAB numeric or logical array.</td>
</tr>
</tbody>
</table>

Output Arguments

| DMObjNew      | DataMatrix object created by left array division.                      |

Description

DMObjNew  = ldivide(DMObj1, DMObj2) or the equivalent DMObjNew  = DMObj1 ./ DMObj2
performs an element-by-element left array division of the DataMatrix objects DMObj1 and DMObj2
and places the results in DMObjNew, another DataMatrix object. In other words, ldivide divides
each element in DMObj2 by the corresponding element in DMObj1. DMObj1 and DMObj2 must have
the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The
size (number of rows and columns), row names, and column names for DMObjNew are the same as
DMObj1, unless DMObj1 is a scalar; then they are the same as DMObj2.

DMObjNew  = ldivide(DMObj1, B) or the equivalent DMObjNew  = DMObj1 ./ B
performs an element-by-element left array division of the DataMatrix object DMObj1 and B, a numeric or logical
array, and places the results in DMObjNew, another DataMatrix object. In other words, ldivide divides
each element in B by the corresponding element in DMObj1. DMObj1 and B must have
the same size (number of rows and columns), unless B is a scalar. The size (number of rows and columns),
row names, and column names for DMObjNew are the same as DMObj1.

DMObjNew  = ldivide(B, DMObj1) or the equivalent DMObjNew  = B ./ DMObj1
performs an element-by-element left array division of B, a numeric or logical array, and the DataMatrix object
DMObj1, and places the results in DMObjNew, another DataMatrix object. In other words, ldivide divides
each element in DMObj1 by the corresponding element in B. DMObj1 and B must have
the same size (number of rows and columns), unless B is a scalar. The size (number of rows and columns),
row names, and column names for DMObjNew are the same as DMObj1.

Note Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not
supported.
MATLAB calls `DMObjNew = ldivide(X, Y)` for the syntax `DMObjNew = X .\ Y` when `X` or `Y` is a DataMatrix object.

**See Also**
DataMatrix | rdivide | times

**Topics**
DataMatrix object on page 1-532

**Introduced in R2008b**
le (DataMatrix)

Test DataMatrix objects for less than or equal to

Syntax

\[
T = \text{le}(\text{DMObj1}, \text{DMObj2}) \\
T = \text{DMObj1} \leq \text{DMObj2} \\
T = \text{le}(\text{DMObj1}, B) \\
T = \text{DMObj1} \leq B \\
T = \text{le}(B, \text{DMObj1}) \\
T = B \leq \text{DMObj1}
\]

Input Arguments

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>MATLAB numeric or logical array.</td>
</tr>
</tbody>
</table>

Output Arguments

| T                     | Logical matrix of the same size as DMObj1 and DMObj2 or DMObj1 and B. It contains logical 1 (true) where elements in the first input are less than or equal to the corresponding element in the second input, and logical 0 (false) otherwise. |

Description

\[T = \text{le}(\text{DMObj1}, \text{DMObj2})\] or the equivalent \[T = \text{DMObj1} \leq \text{DMObj2}\] compares each element in DataMatrix object \(\text{DMObj1}\) to the corresponding element in DataMatrix object \(\text{DMObj2}\), and returns \(T\), a logical matrix of the same size as \(\text{DMObj1}\) and \(\text{DMObj2}\), containing logical 1 (true) where elements in \(\text{DMObj1}\) are less than or equal to the corresponding element in \(\text{DMObj2}\), and logical 0 (false) otherwise. \(\text{DMObj1}\) and \(\text{DMObj2}\) must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). \(\text{DMObj1}\) and \(\text{DMObj2}\) can have different Name properties.

\[T = \text{le}(\text{DMObj1}, B)\] or the equivalent \[T = \text{DMObj1} \leq B\] compares each element in DataMatrix object \(\text{DMObj1}\) to the corresponding element in \(B\), a numeric or logical array, and returns \(T\), a logical matrix of the same size as \(\text{DMObj1}\) and \(B\), containing logical 1 (true) where elements in \(\text{DMObj1}\) are less than or equal to the corresponding element in \(B\), and logical 0 (false) otherwise. \(\text{DMObj1}\) and \(B\) must have the same size (number of rows and columns), unless one is a scalar.

\[T = \text{le}(B, \text{DMObj1})\] or the equivalent \[T = B \leq \text{DMObj1}\] compares each element in \(B\), a numeric or logical array, to the corresponding element in DataMatrix object \(\text{DMObj1}\), and returns \(T\), a logical matrix of the same size as \(B\) and \(\text{DMObj1}\), containing logical 1 (true) where elements in \(B\) are less than or equal to the corresponding element in \(\text{DMObj1}\), and logical 0 (false) otherwise. \(B\) and \(\text{DMObj1}\) must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls \(T = \text{le}(X, Y)\) for the syntax \(T = X \leq Y\) when \(X\) or \(Y\) is a DataMatrix object.
See Also
DataMatrix | ge

Topics
DataMatrix object on page 1-532

Introduced in R2008b
**localalign**

Return local optimal and suboptimal alignments between two sequences

**Syntax**

```
AlignStruct = localalign(Seq1, Seq2)
AlignStruct = localalign(Seq1, Seq2, ...'NumAln', NumAlnValue, ...)
AlignStruct = localalign(Seq1, Seq2, ...'MinScore', MinScoreValue, ...)
AlignStruct = localalign(Seq1, Seq2, ...'Percent', PercentValue, ...)
AlignStruct = localalign(Seq1, Seq2, ...'DoAlignment', DoAlignmentValue, ...)
AlignStruct = localalign(Seq1, Seq2, ...'Alphabet', AlphabetValue, ...)
AlignStruct = localalign(Seq1, Seq2, ...'ScoringMatrix', ScoringMatrixValue, ...)
AlignStruct = localalign(Seq1, Seq2, ...'Scale', ScaleValue, ...)
AlignStruct = localalign(Seq1, Seq2, ...'GapOpen', GapOpenValue, ...)
```

**Description**

`AlignStruct = localalign(Seq1, Seq2)` returns information about the first optimal (highest scoring) local alignment between two sequences in a MATLAB structure.

`AlignStruct = localalign(Seq1, Seq2, ...'PropertyName', PropertyValue, ...)` calls `localalign` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each `PropertyName` in single quotation marks. Each `PropertyName` is case insensitive. These property name/property value pairs are as follows:

- `AlignStruct = localalign(Seq1, Seq2, ...'NumAln', NumAlnValue, ...)` returns information about one or more nonintersecting, local alignments (optimal and suboptimal). It limits the number of alignments to return by specifying the number of local alignments to return. It returns the alignments in decreasing order according to their score.

- `AlignStruct = localalign(Seq1, Seq2, ...'MinScore', MinScoreValue, ...)` returns information about nonintersecting, local alignments (optimal and suboptimal), whose score is greater than `MinScoreValue`.

- `AlignStruct = localalign(Seq1, Seq2, ...'Percent', PercentValue, ...)` returns information about one or more nonintersecting local alignments (optimal and suboptimal), whose scores are within `PercentValue` percent of the highest score. It returns the alignments in decreasing order according to their score.

- `AlignStruct = localalign(Seq1, Seq2, ...'DoAlignment', DoAlignmentValue, ...)` specifies whether to include the pairwise alignments in the Alignment field of the output structure. Choices are true (default) or false.

- `AlignStruct = localalign(Seq1, Seq2, ...'Alphabet', AlphabetValue, ...)` specifies the type of sequences. Choices are 'AA' (default) or 'NT'.

- `AlignStruct = localalign(Seq1, Seq2, ...'ScoringMatrix', ScoringMatrixValue, ...)` specifies the scoring matrix to use for the local alignment.
\( \text{AlignStruct} = \text{localalign}(\text{Seq1}, \text{Seq2}, \ldots \text{'}\text{Scale}'\text{, ScaleValue}, \ldots) \) specifies a scale factor applied to the output scores, thereby controlling the units of the output scores. Choices are any positive value. Default is 1, which does not change the units of the output score.

\( \text{AlignStruct} = \text{localalign}(\text{Seq1}, \text{Seq2}, \ldots \text{'}\text{GapOpen}'\text{, GapOpenValue}, \ldots) \) specifies the penalty for opening a gap in the alignment. Choices are any positive value. Default is 8.

**Input Arguments**

**Seq1**

First amino acid or nucleotide sequence specified by any of the following:

- Character vector or string of letters representing amino acids or nucleotides, such as returned by int2aa or int2nt
- Vector of integers representing amino acids or nucleotides, such as returned by aa2int or nt2int
- MATLAB structure containing a `Sequence` field, such as returned by fastaread, fastqread, emblread, getembl, genbankread, getgenbank, getgenpept, genpeptread, getpdb, pdbread, or sffread

**Tip** For help with letter and integer representations of amino acids and nucleotides, see Amino Acid Lookup or Nucleotide Lookup.

**Default:**

**Seq2**

Second amino acid or nucleotide sequence, which localalign aligns with `Seq1`.

**Default:**

**NumAlnValue**

Positive scalar (< or \( \leq 2^{12} \)) specifying the number of alignments to return. localalign returns the top `NumAlnValue` local, nonintersecting alignments (optimal and suboptimal). If the number of optimal alignments is greater than `NumAlnValue`, then localalign returns the first `NumAlnValue` alignments based on their order in the trace back matrix.

**Note** If you specify a `NumAlnValue`, you cannot specify a `MinScoreValue` or `PercentValue`.

**Tip** Use `NumAlnValue` to return multiple alignments when you are aligning low complexity sequences and must consider several local alignments.

**Default:** 1

**MinScoreValue**

Positive scalar specifying the minimum score of local, nonintersecting alignments (optimal and suboptimal) to return.
**Note** If you specify a `MinScoreValue`, you cannot specify a `NumAlnValue` or `PercentValue`.

**Tip** Use `MinScoreValue` to return suboptimal alignments, for example when you are interested in accounting for sequencing errors or imperfect scoring matrices.

**Default:**

**PercentValue**

Positive scalar between 0 and 100 that limits the return of local, nonintersecting alignments (optimal and suboptimal) to those alignments with a score within `PercentValue` percent of the highest score. For example, if the highest score is 10.5 and you specify 5 for `PercentValue`, then `localalign` determines a minimum score of $10.5 - (10.5 * 0.05) = 9.975$. It returns all alignments with a score of 9.975 or higher.

**Note** If you specify a `PercentValue`, you cannot specify a `NumAlnValue` or `MinScoreValue`.

**Tip** Use `PercentValue` to return optimal and suboptimal alignments when you do not know how similar the two sequences are or how well they score against a given scoring matrix.

**Default:**

**DoAlignmentValue**

Controls the inclusion of the pairwise alignments in the `Alignment` field of the output structure. Choices are `true` (default) or `false`.

**Default:**

**AlphabetValue**

Character vector or string specifying the type of sequences. Choices are `'AA'` (default) or `'NT'`.

**Default:**

**ScoringMatrixValue**

Either of the following:

- Character vector or string specifying the scoring matrix to use for the local alignment. Choices for amino acid sequences are:
  - `'BLOSUM62'`
  - `'BLOSUM30'` increasing by 5 up to `'BLOSUM90'`
  - `'BLOSUM100'`
  - `'PAM10'` increasing by 10 up to `'PAM500'`
  - `'DAYHOFF'`
  - `'GONNET'`

Default is: 1-999
• ‘BLOSUM50’ — When AlphabetValue equals ‘AA’
• ‘NUC44’ — When AlphabetValue equals ‘NT’

Note: The previous scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the ‘Scale’ property to specify an additional scale factor to convert the output score from bits to another unit.

• Matrix representing the scoring matrix to use for the local alignment, such as returned by the blosum, pam, dayhoff, gonnet, or nuc44 function.

Note: If you use a scoring matrix that you created or was created by one of the previous functions, the matrix does not include a scale factor. The output score is returned in the same units as the scoring matrix. You can use the ‘Scale’ property to specify a scale factor to convert the output score to another unit.

Note: If you need to compile localalign into a stand-alone application or software component using MATLAB Compiler™, use a matrix instead of a character vector or string for ScoringMatrixValue.

Default:

ScaleValue

Positive value that specifies a scale factor that is applied to the output scores, thereby controlling the units of the output scores.

For example, if the output score is initially determined in bits, and you enter log(2) for ScaleValue, then localalign returns Score in nats.

Default is 1, which does not change the units of the output score.

Note: If the ‘ScoringMatrix’ property also specifies a scale factor, then localalign uses it first to scale the output score. It then applies the scale factor specified by ScaleValue to rescale the output score.

Tip: Before comparing alignment scores from multiple alignments, ensure that the scores are in the same units. Use the ‘Scale’ property to control the units of the output scores.

Default:

GapOpenValue

Positive value specifying the penalty for opening a gap in the alignment.

Default: 8
Output Arguments

AlignStruct

MATLAB structure or array of structures containing information about the local optimal and suboptimal alignments between two sequences. Each structure represents an optimal or suboptimal alignment and contains the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>Score for the local optimal or suboptimal alignment.</td>
</tr>
<tr>
<td>Start</td>
<td>1-by-2 vector of indices indicating the starting point in each sequence for the alignment.</td>
</tr>
<tr>
<td>Stop</td>
<td>1-by-2 vector of indices indicating the stopping point in each sequence for the alignment.</td>
</tr>
<tr>
<td>Alignment</td>
<td>3-by-N character array showing the two sequences, Seq1 and Seq2, in the first and third rows. It also shows symbols representing the optimal or suboptimal local alignment between the two sequences in the second row.</td>
</tr>
</tbody>
</table>

Examples

Limit the number of alignments to return between two sequences by specifying the number of alignments:

% Create variables containing two amino acid sequences.
Seq1 = 'VSPAGMASGYDPGKA';
Seq2 = 'IPGKATREYDVSPAG';

% Use the NumAln property to return information about the top three local alignments.
struct1 = localalign(Seq1, Seq2, 'numaln', 3)

struct1 =

    Score: [3x1 double]
    Start: [3x2 double]
    Stop: [3x2 double]
    Alignment: {3x1 cell}

% View the scores of the first and second alignments.
struct1.Score(1:2)

ans =

    11.0000
    9.6667

% View the first alignment.
struct1.Alignment{1}

ans =

    VSPAG
    |||||
    VSPAG
Limit the number of alignments to return between two sequences by specifying a minimum score:

```matlab
% Create variables containing two amino acid sequences.
Seq1 = 'VSPAGMASGYDPGKA';
Seq2 = 'IPGKATREYDVSPAG';

% Use the MinScore property to return information about
% only local alignments with a score greater than 8.
% Use the DoAlignment property to exclude the actual alignments.
struct2 = localalign(Seq1,Seq2,'minscore',8,'doalignment',false)
```

```matlab
struct2 =
    Score: [2x1 double]
    Start: [2x2 double]
    Stop: [2x2 double]
```

Limit the number of alignments to return between two sequences by specifying a percentage from the maximum score:

```matlab
% Create variables containing two amino acid sequences.
Seq1 = 'VSPAGMASGYDPGKA';
Seq2 = 'IPGKATREYDVSPAG';

% Use the Percent property to return information about only
% local alignments with a score within 15% of the maximum score.
struct3 = localalign(Seq1, Seq2, 'percent', 15)
```

```matlab
struct3 =
    Score: [2x1 double]
    Start: [2x2 double]
    Stop: [2x2 double]
    Alignment: {2x1 cell}
```

Specify a scoring matrix and gap opening penalty when aligning two sequences:

```matlab
% Create variables containing two nucleotide sequences.
Seq1 = 'CCAATCTACTACTGCTTGCAGTAC';
Seq2 = 'AGTCCGAGGGCTACTCTACTGAAC';

% Create a scoring matrix with a match score of 10 and a mismatch
% score of -9
sm = [10 -9 -9 -9;
     -9 10 -9 -9;
     -9 -9 10 -9;
     -9 -9 -9 10];

% Use the ScoringMatrix and GapOpen properties when returning
% information about the top three local alignments.
struct4 = localalign(Seq1, Seq2, 'alpha', 'nt', ...
    'scoringmatrix', sm, 'gapopen', 20, 'numaln', 3)
```

```matlab
struct4 =
    Score: [3x1 double]
    Start: [3x2 double]
    Stop: [3x2 double]
    Alignment: {3x1 cell}
```
More About

Nonintersecting Alignments
Alignments having no matches or mismatches in common.

Optimal Alignment
An alignment with the highest score.

Suboptimal Alignment
An alignment with a score less than the highest score.

References


See Also
blossum | dayhoff | fastaread | gethmmalignment | gonnet | localalign | multialign | multialignread | nuc44 | nwallign | pam | seqalignviewer | showalignment | swalign

Topics
“Aligning Pairs of Sequences”
“Retrieve Sequence Information from a Public Database”
“View and Align Multiple Sequences”
Amino Acid Lookup
Nucleotide Lookup

External Websites
https://www.rcsb.org/pdb/home/home.do
https://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?cmd=show&f=main&m=main&s=main

Introduced in R2009b
It (DataMatrix)

Test DataMatrix objects for less than

Syntax

\[
T = \text{lt}(\text{DMObj1}, \text{DMObj2}) \\
T = \text{DMObj1} < \text{DMObj2} \\
T = \text{lt}(\text{DMObj1}, B) \\
T = \text{DMObj1} < B \\
T = \text{lt}(B, \text{DMObj1}) \\
T = B < \text{DMObj1}
\]

Input Arguments

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>MATLAB numeric or logical array.</td>
</tr>
</tbody>
</table>

Output Arguments

| T               | Logical matrix of the same size as DMObj1 and DMObj2 or DMObj1 and B. It contains logical 1 (true) where elements in the first input are less than the corresponding element in the second input, and logical 0 (false) otherwise. |

Description

\[
T = \text{lt}(\text{DMObj1}, \text{DMObj2}) \quad \text{or the equivalent } T = \text{DMObj1} < \text{DMObj2} \quad \text{compares each element in DataMatrix object DMObj1 to the corresponding element in DataMatrix object DMObj2, and returns } T, \quad \text{a logical matrix of the same size as DMObj1 and DMObj2, containing logical 1 (true) where elements in DMObj1 are less than the corresponding element in DMObj2, and logical 0 (false) otherwise. DMObj1 and DMObj2 must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). DMObj1 and DMObj2 can have different Name properties.}
\]

\[
T = \text{lt}(\text{DMObj1}, B) \quad \text{or the equivalent } T = \text{DMObj1} < B \quad \text{compares each element in DataMatrix object DMObj1 to the corresponding element in a numeric or logical array, and returns } T, \quad \text{a logical matrix of the same size as DMObj1 and B, containing logical 1 (true) where elements in DMObj1 are less than the corresponding element in B, and logical 0 (false) otherwise. DMObj1 and B must have the same size (number of rows and columns), unless one is a scalar.}
\]

\[
T = \text{lt}(B, \text{DMObj1}) \quad \text{or the equivalent } T = B < \text{DMObj1} \quad \text{compares each element in B, a numeric or logical array, to the corresponding element in DataMatrix object DMObj1, and returns } T, \quad \text{a logical matrix of the same size as B and DMObj1, containing logical 1 (true) where elements in B are less than the corresponding element in DMObj1, and logical 0 (false) otherwise. B and DMObj1 must have the same size (number of rows and columns), unless one is a scalar.}
\]

MATLAB calls \( T = \text{lt}(X, Y) \) for the syntax \( T = X < Y \) when \( X \) or \( Y \) is a DataMatrix object.
See Also
DataMatrix | gt

Topics
DataMatrix object on page 1-532

Introduced in R2008b
**maboxplot**

Create box plot for microarray data

**Syntax**

\[
\text{maboxplot}(\text{MAData}) \\
\text{maboxplot}(\text{MAData}, \text{ColumnName}) \\
\text{maboxplot}(\text{MAStruct}, \text{FieldName}) \\
H = \text{maboxplot}(...) \\
[H, HLines] = \text{maboxplot}(...) \\
\text{maboxplot}(..., 'Title', \text{TitleValue}, ...) \\
\text{maboxplot}(..., 'Notch', \text{NotchValue}, ...) \\
\text{maboxplot}(..., 'Symbol', \text{SymbolValue}, ...) \\
\text{maboxplot}(..., 'Orientation', \text{OrientationValue}, ...) \\
\text{maboxplot}(..., 'WhiskerLength', \text{WhiskerLengthValue}, ...) \\
\text{maboxplot}(..., 'BoxPlot', \text{BoxPlotValue}, ...) \\
\]

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAData</td>
<td>DataMatrix object on page 1-532, numeric array, or a structure containing a field called Data. The values in the columns of MAData will be used to create box plots. If a DataMatrix object, the column names are used as labels in the box plot.</td>
</tr>
<tr>
<td>ColumnName</td>
<td>An array of column names corresponding to the data in MAData used as labels in the box plot.</td>
</tr>
<tr>
<td>MAStruct</td>
<td>A microarray data structure.</td>
</tr>
<tr>
<td>FieldName</td>
<td>A field within the microarray data structure, MAStruct. The values in the field FieldName will be used to create box plots.</td>
</tr>
<tr>
<td>TitleValue</td>
<td>Character vector or string to use as the title for the plot. The default title is FieldName.</td>
</tr>
<tr>
<td>NotchValue</td>
<td>Logical specifying the type of boxes drawn. Choices are:</td>
</tr>
<tr>
<td></td>
<td>- true — Notched boxes</td>
</tr>
<tr>
<td></td>
<td>- false — Square boxes</td>
</tr>
<tr>
<td></td>
<td>Default is false.</td>
</tr>
<tr>
<td>OrientationValue</td>
<td>Character vector or string specifying the orientation of the box plot. Choices are:</td>
</tr>
<tr>
<td></td>
<td>- 'Vertical'</td>
</tr>
<tr>
<td></td>
<td>- 'Horizontal' (default)</td>
</tr>
</tbody>
</table>
**WhiskerLengthValue**

Value specifying the maximum length of the whiskers as a function of the interquartile range (IQR). The whisker extends to the most extreme data value within \( WhiskerLengthValue \times IQR \) of the box. Default = 1.5. If \( WhiskerLengthValue \) equals 0, then maboxplot displays all data values outside the box, using the plotting symbol Symbol.

**BoxPlotValue**

A cell array of property name/property value pairs to pass to the Statistics and Machine Learning Toolbox™ boxplot function, which creates the box plot. For valid pairs, see the boxplot function.

---

**Description**

maboxplot(MAData) displays a box plot of the values in the columns of MAData. MAData can be a DataMatrix object on page 1-532, numeric array, or a structure containing a field called Data, containing microarray data.

maboxplot(MAData, ColumnName) labels the box plot column names.

maboxplot(MAStruct, FieldName) displays a box plot of the values in the field FieldName in the microarray data structure MAStruct. If MAStruct is block based, maboxplot creates a box plot of the values in the field FieldName for each block.

**Note** If you provide MAStruct, without providing FieldName, maboxplot uses the Signal element in the ColumnNames field of MAStruct, if Affymetrixdata, or the first element in the ColumnNames field of MAStruct, otherwise.

\[ H = \text{maboxplot}(...) \] returns the handle of the box plot axes.

\[ [H, HLines] = \text{maboxplot}(...) \] returns the handles of the lines used to separate the different blocks in the image.

maboxplot(..., 'PropertyName', PropertyValue, ...) calls maboxplot with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

maboxplot(..., 'Title', TitleValue, ...) allows you to specify the title of the plot. The default TitleValue is FieldName.

maboxplot(..., 'Notch', NotchValue, ...) if NotchValue is true, draws notched boxes. The default is false to show square boxes.

maboxplot(..., 'Symbol', SymbolValue, ...) allows you to specify the symbol used for outlier values. The default Symbol is '+'.

maboxplot(..., 'Orientation', OrientationValue, ...) allows you to specify the orientation of the box plot. The choices are 'Vertical' and 'Horizontal'. The default is 'Vertical'.

maboxplot(..., 'WhiskerLength', WhiskerLengthValue, ...) allows you to specify the whisker length for the box plot. WhiskerLengthValue defines the maximum length of the whiskers...
as a function of the interquartile range (IQR) (default = 1.5). The whisker extends to the most extreme data value within WhiskerLength*IQR of the box. If WhiskerLengthValue equals 0, then maboxplot displays all data values outside the box, using the plotting symbol Symbol.

maboxplot(..., 'BoxPlot', BoxPlotValue, ...) allows you to specify arguments to pass to the boxplot function, which creates the box plot. BoxPlotValue is a cell array of property name/property value pairs. For valid pairs, see the boxplot function.

Examples

Display Box Plots for Microarray Data

This example shows how to display box plots for microarray data.

Load the MAT-file, provided with the Bioinformatics Toolbox™ software, that contains yeast data. This MAT-file includes three variables: yeastvalues, a matrix of gene expression data, genes, a cell array of GenBank® accession numbers for labeling the rows in yeastvalues, and times, a vector of time values for labeling the columns in yeastvalues.

load yeastdata

Show the box plot of gene expression data.

maboxplot(yeastvalues, times);
xlabel('Sample Times');
Use the `gprread` function to create a structure containing microarray data, and plot the data using name-value pair arguments of the `maboxplot` function.

```matlab
madata = gprread('mouse_a1wt.gpr');
maboxplot(madata,'F635 Median - B635','TITLE', 'Cy5 Channel FG - BG');
```

See Also

`boxplot` | `magetfield` | `maimage` | `mairplot` | `maloglog` | `malowess` | `manorm` | `mavolcanoplot`

Introduced before R2006a
mafdr

Estimate positive false discovery rate for multiple hypothesis testing

Syntax

FDR = mafdr(PValues)
FDR = mafdr(PValues,Name,Value)
[FDR,Q] = mafdr(PValues,___)
[FDR,Q,aPrioriProb] = mafdr(PValues,___)
[FDR,Q,aPrioriProb,R_squared] = mafdr(PValues,'Method','polynomial',___)

Description

FDR = mafdr(PValues) returns FDR that contains a positive false discovery rate (pFDR) for each entry in PValues using the procedure introduced by Storey (2002) [1]. PValues contains one p-value for each feature (for example, a gene) in a data set.

FDR = mafdr(PValues,Name,Value) uses additional options specified by one or more name-value pair arguments. For example, 'Showplot',true displays diagnostic plots of calculated results.

[FDR,Q] = mafdr(PValues,___) also returns hypothesis testing error measures Q for all p-values. Optionally, you can specify one or more name-value pair arguments.

[FDR,Q,aPrioriProb] = mafdr(PValues,___) also returns aPrioriProb, the estimated a priori probability that the null hypothesis $\pi_0$ is true.

[FDR,Q,aPrioriProb,R_squared] = mafdr(PValues,'Method','polynomial',___) also returns R_squared, the square of correlation coefficient. Use the polynomial method to get the R-squared value.

Examples

Estimate Positive False Discovery Rate for Multiple Hypothesis Testing

Estimate the positive FDR using data from a prostate cancer study (Best et al., 2005). The data contains probe intensity data from Affymetrix® HG-U133A GeneChip® arrays.

Load the gene expression data. It contains two variables, dependentData and independentData that are two matrices of gene expression values from two experimental conditions.

load prostatecancerexpdata

Use mattest to calculate the p-values for gene expression values in the two matrices.

pvalues = mattest(dependentData,independentData,'permute',true);

Use mafdr to calculate the positive FDR values.

fdr = mafdr(pvalues);
Calculate the q-values, \textit{a priori} probability (that the null hypothesis is true), and R-squared value. You must use the polynomial method to get the R-squared value. Plot the data by setting 'Showplot' to true.

\[
[fdr,q,priori,R2] = \text{mafdr}(pvalues,'Method','polynomial','Showplot',true);
\]

**Input Arguments**

\textbf{PValues — P-values for all features}
column vector | DataMatrix object

P-values for all features in a data set, specified as a column vector or a DataMatrix object on page 1-532. You can use the first output of the \text{mattest} function.

Data Types: double

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of \textit{Name}, \textit{Value} arguments. \textit{Name} is the argument name and \textit{Value} is the corresponding value. \textit{Name} must appear inside quotes. You can specify several name and value pair arguments in any order as \text{Name1,Value1,...,NameN,ValueN}.

Example: \text{fdr = mafdr(pvals,'Lambda',0.5,'Showplot',true)} specifies the tuning parameter value of 0.5 to estimate a prior probability and displays the quality statistics plots.
**BHFDR — Flag to use linear step-up procedure**

flag to use the linear step-up procedure introduced by Benjamini and Hochberg (1995) [2], specified as the comma-separated pair consisting of 'BHFDR' and true or false. The default value is false, that is, the function uses the procedure introduced by Storey (2002) [1].

If true:

- The function uses the Benjamini and Hochberg method.
- The function ignores the 'Method' and 'Lambda' name-value pair arguments.
- Specify only one output argument, that is, FDR.
- If you also set 'Showplot' to true, then the function plots only the q-values versus p-values. For details, see “Showplot” on page 1-0.

Example: 'BHFDR',true

Data Types: logical

**Lambda — Tuning parameter**

[0.01:0.01:0.95] (default) | positive scalar | vector

Tuning parameter used to estimate the a priori probability that the null hypothesis is true, specified as the comma-separated pair consisting of 'Lambda' and a positive scalar or vector with four or more values. The scalar value or each value in the vector must be between 0 and 1.

- If you specify a single value, then the function ignores the 'Method' name-value pair argument.
- If you specify a vector of values, then the function chooses the optimal value using the method specified by the 'Method' name-value pair argument.

Example: 'Lambda',[0.01:0.1:0.95]

Data Types: double

**Method — Method to choose Lambda value**

'bootstrap' (default) | 'polynomial'

Method to choose the Lambda value from a range of values, specified as the comma-separated pair consisting of 'Method' and 'bootstrap' or 'polynomial'.

Example: 'Method','polynomial'

Data Types: char | string

**Showplot — Flag to display diagnostic plots**

false (default) | true

Flag to display two diagnostic plots, specified as the comma-separated pair consisting of 'Showplot' and true or false.

If true, the function displays two plots:

- Estimated a priori probability that the null hypothesis \( \tilde{f}_0(\lambda) \) is true versus the tuning parameter \( \lambda \) with a cubic polynomial fitting curve.
• q-values versus p-values

If you also set ‘BHFD’ to true, the function displays only the second plot.

Example: ‘Showplot’,true

Data Types: logical

Output Arguments

FDR — Positive FDR values
vector | DataMatrix object

Positive FDR values, returned as a vector or DataMatrix object.

If PValues is a column vector, then FDR is a column vector.

If PValues is a DataMatrix object, then FDR is a DataMatrix object.

Q — Q-values
column vector

Q-values, returned as a column vector. Q contains the measures of hypothesis testing error for all observations in PValues.

aPrioriProb — Estimated a priori probability
positive scalar

Estimated a priori probability that the null hypothesis \( \pi_0 \) is true, returned as a positive scalar.

R_squared — Square of correlation coefficient
positive scalar

Square of the correlation coefficient, returned as a positive scalar. Specify 'Method' as 'polynomial' to get this fourth output.

References


See Also
affygcrma | affyrma | gcrma | mairplot | maloglog | mapcaplot | mattest | mavolcanoplot | rmasummary

Introduced in R2007a
magetfield

Extract data from microarray structure

Syntax

magetfield(MAstruct, FieldName)

Arguments

<table>
<thead>
<tr>
<th>MAstruct</th>
<th>Microarray structure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FieldName</td>
<td>A column in MAstruct.</td>
</tr>
</tbody>
</table>

Description

magetfield(MAstruct, FieldName) extracts data for FieldName, a column in MAstruct, microarray structure.

The benefit of this function is to hide the details of extracting a column of data from a structure created with one of the microarray reader functions (gprread, agferead, sptread, imageneread).

Examples

maStruct = gprread('mouse_a1wt.gpr');
cy5data = magetfield(maStruct,'F635 Median');
cy3data = magetfield(maStruct,'F532 Median');
mairplot(cy5data,cy3data,'title','R vs G IR plot');

See Also

agferead | gprread | ilmnbsread | imageneread | maboxplot | mairplot | maloglog | malowess | sptread

Introduced before R2006a
**maimage**

Spatial image for microarray data

**Syntax**

\[
\text{maimage}(X, \text{FieldName}) \\
H = \text{maimage}(\ldots) \\
[H, Hlines] = \text{maimage}(\ldots) \\
\text{maimage}(\ldots, '{\text{PropertyName}}', {\text{PropertyValue}},\ldots) \\
\text{maimage}(\ldots, '{\text{Title}}', {\text{TitleValue}}) \\
\text{maimage}(\ldots, '{\text{ColorBar}}', {\text{ColorBarValue}}) \\
\text{maimage}(\ldots, '{\text{HandleGraphicsPropertyName}}' {\text{PropertyValue}},\ldots)
\]

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X)</td>
<td>A microarray data structure.</td>
</tr>
<tr>
<td>FieldName</td>
<td>A field in the microarray data structure (X).</td>
</tr>
<tr>
<td>TitleValue</td>
<td>A character vector or string to use as the title for the plot. The default title is FieldName.</td>
</tr>
<tr>
<td>ColorBarValue</td>
<td>Property to control displaying a color bar in the Figure window. Enter either true or false. The default value is false.</td>
</tr>
</tbody>
</table>

**Description**

\[
\text{maimage}(X, \text{FieldName}) \text{ displays an image of field } \text{FieldName} \text{ from microarray data structure } X. \\
\text{Microarray data can be GenPix Results (GPR) format. After creating the image, click a data point to display the value and ID, if known.} \\
H = \text{maimage}(\ldots) \text{ returns the handle of the image.} \\
[H, Hlines] = \text{maimage}(\ldots) \text{ returns the handles of the lines used to separate the different blocks in the image.}
\]

\[
\text{maimage}(\ldots, '{\text{PropertyName}}', {\text{PropertyValue}},\ldots) \text{ defines optional properties using property name/value pairs.} \\
\text{maimage}(\ldots, '{\text{Title}}', {\text{TitleValue}}) \text{ allows you to specify the title of the plot. The default title is FieldName.} \\
\text{maimage}(\ldots, '{\text{ColorBar}}', {\text{ColorBarValue}}), \text{ when } \text{ColorBarValue} \text{ is true, a color bar is shown. If } \text{ColorBarValue} \text{ is false, no color bar is shown. The default is for the color bar to be shown.} \\
\text{maimage}(\ldots, '{\text{HandleGraphicsPropertyName}}' {\text{PropertyValue}},\ldots) \text{ allows you to pass optional Handle Graphics® property name/value pairs to the function. For example, a name/value pair for color could be } \text{maimage}(\ldots, '{\text{color}}' \ 'r').
\]
Examples

Generate Spatial Image for Microarray Data

This example shows how to generate spatial images for microarray data.

Read in a sample GPR file.

```matlab
data = gprread('mouse_alwt.gpr');
```

Plot the median foreground intensity for the 635 nm channel.

```matlab
maimage(data,'F635 Median')
```

Alternatively, create a similar plot using more basic graphics commands.

```matlab
F635Median = magetfield(data,'F635 Median');
figure
imagesc(F635Median(data.Indices));
```
Change the colormap and add a color bar.

```matlab
colormap bone
colorbar
```
See Also
imagesc | maboxplot | magetfield | mairplot | maloglog | malowess

Introduced before R2006a
mainvarsetnorm

Perform rank invariant set normalization on gene expression values from two experimental conditions or phenotypes

Syntax

\[ \text{NormDataY} = \text{mainvarsetnorm}(\text{DataX}, \text{DataY}) \]
\[ \text{NormDataY} = \text{mainvarsetnorm}(..., \text{'Thresholds'}, \text{ThresholdsValue}, ...) \]
\[ \text{NormDataY} = \text{mainvarsetnorm}(..., \text{'Exclude'}, \text{ExcludeValue}, ...) \]
\[ \text{NormDataY} = \text{mainvarsetnorm}(..., \text{'Percentile'}, \text{PercentileValue}, ...) \]
\[ \text{NormDataY} = \text{mainvarsetnorm}(..., \text{'Iterate'}, \text{IterateValue}, ...) \]
\[ \text{NormDataY} = \text{mainvarsetnorm}(..., \text{'Method'}, \text{MethodValue}, ...) \]
\[ \text{NormDataY} = \text{mainvarsetnorm}(..., \text{'Span'}, \text{SpanValue}, ...) \]
\[ \text{NormDataY} = \text{mainvarsetnorm}(..., \text{'Showplot'}, \text{ShowplotValue}, ...) \]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataX</td>
<td>Vector of gene expression values from a single experimental condition or phenotype, where each row corresponds to a gene. These data points are used as the baseline.</td>
</tr>
<tr>
<td>DataY</td>
<td>Vector of gene expression values from a single experimental condition or phenotype, where each row corresponds to a gene. These data points will be normalized using the baseline.</td>
</tr>
<tr>
<td>ThresholdsValue</td>
<td>Vector that sets the thresholds for the lowest average rank and the highest average rank between the two data sets. The average rank for each data point is determined by first converting the values in DataX and DataY to ranks, then averaging the two ranks for each data point. Then, the threshold for each data point is determined by interpolating between the threshold for the lowest average rank and the threshold for the highest average rank.</td>
</tr>
</tbody>
</table>

Note: These individual thresholds are used to determine the rank invariant set, which is a set of data points, each having a proportional rank difference (prd) smaller than its predetermined threshold. For more information on the rank invariant set, see “Description” on page 1-1021.

\[ \text{ThresholdsValue} \] is a 1-by-2 vector \([LT, HT]\), where \(LT\) is the threshold for the lowest average rank and \(HT\) is threshold for the highest average rank. Select these two thresholds empirically to limit the spread of the invariant set, but allow enough data points to determine the normalization relationship. Values must be between 0 and 1. Default is \([0.03, 0.07]\).
ExcludeValue

Property to filter the invariant set of data points, by excluding the data points whose average rank (between DataX and DataY) is in the highest $N$ ranked averages or lowest $N$ ranked averages.

PercentileValue

Property to stop the iteration process when the number of data points in the invariant set reaches $N$ percent of the total number of input data points. Default is 1.

**Note** If you do not use this property, the iteration process continues until no more data points are eliminated.

IterateValue

Property to control the iteration process for determining the invariant set of data points. Enter *true* to repeat the process until either no more data points are eliminated, or a predetermined percentage of data points (PercentileValue) is reached. Enter *false* to perform only one iteration of the process. Default is true.

**Tip** Select *false* for smaller data sets, typically less than 200 data points.

MethodValue

Property to select the smoothing method used to normalize the data. Enter *'lowess'* or *'runmedian'* . Default is *'lowess'* .

SpanValue

Property to set the window size for the smoothing method. If SpanValue is less than 1, the window size is that percentage of the number of data points. If SpanValue is equal to or greater than 1, the window size is of size SpanValue . Default is 0.05, which corresponds to a window size equal to 5% of the total number of data points in the invariant set.

ShowplotValue

Property to control the plotting of a pair of M-A scatter plots (before and after normalization). M is the ratio between DataX and DataY. A is the average of DataX and DataY. Enter *true* to create the pair of M-A scatter plots. Default is *false*.

### Description

$\text{NormDataY} = \text{mainvarsetnorm(DataX, DataY)}$ normalizes the values in DataY, a vector of gene expression values, to a reference vector, DataX, using the invariant set method. NormDataY is a vector of normalized gene expression values from DataY.

Specifically, mainvarsetnorm:

- Determines the proportional rank difference ($prd$) for each pair of ranks, RankX and RankY, from the two vectors of gene expression values, DataX and DataY.

\[
prd = \text{abs}(\text{RankX} - \text{RankY})
\]

- Determines the invariant set of data points by selecting data points whose proportional rank differences ($prd$) are below threshold, which is a predetermined threshold for a given data point (defined by the ThresholdsValue property). It optionally repeats the process until either no more data points are eliminated, or a predetermined percentage of data points is reached.

The invariant set is data points with a $prd < \text{threshold}$.
• Uses the invariant set of data points to calculate the lowess or running median smoothing curve, which is used to normalize the data in DataY.

**Note** If DataX or DataY contains NaN values, then NormDataY will also contain NaN values at the corresponding positions.

**Tip** mainvarsetnorm is useful for correcting for dye bias in two-color microarray data.

_normDataY = mainvarsetnorm(..., 'PropertyName', PropertyValue, ...)_ calls mainvarsetnorm with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

_normDataY = mainvarsetnorm(..., 'Thresholds', ThresholdsValue, ...)_ sets the thresholds for the lowest average rank and the highest average rank between the two data sets. The average rank for each data point is determined by first converting the values in DataX and DataY to ranks, then averaging the two ranks for each data point. Then, the threshold for each data point is determined by interpolating between the threshold for the lowest average rank and the threshold for the highest average rank.

**Note** These individual thresholds are used to determine the rank invariant set, which is a set of data points, each having a proportional rank difference (prd) smaller than its predetermined threshold. For more information on the rank invariant set, see “Description” on page 1-1021.

*ThresholdsValue* is a 1-by-2 vector [LT, HT], where LT is the threshold for the lowest average rank and HT is threshold for the highest average rank. Select these two thresholds empirically to limit the spread of the invariant set, but allow enough data points to determine the normalization relationship. Values must be between 0 and 1. Default is [0.03, 0.07].

_normDataY = mainvarsetnorm(..., 'Exclude', ExcludeValue, ...)_ filters the invariant set of data points, by excluding the data points whose average rank (between DataX and DataY) is in the highest N ranked averages or lowest N ranked averages.

_normDataY = mainvarsetnorm(..., 'Percentile', PercentileValue, ...)_ stops the iteration process when the number of data points in the invariant set reaches N percent of the total number of input data points. Default is 1.

**Note** If you do not use this property, the iteration process continues until no more data points are eliminated.

_normDataY = mainvarsetnorm(..., 'Iterate', IterateValue, ...)_ controls the iteration process for determining the invariant set of data points. When IterateValue is true, mainvarsetnorm repeats the process until either no more data points are eliminated, or a predetermined percentage of data points (PercentileValue) is reached. When IterateValue is false, performs only one iteration of the process. Default is true.
Tip Select false for smaller data sets, typically less than 200 data points.

\[ \text{NormDataY} = \text{mainvarsetnorm(..., 'Method', MethodValue, ...)} \]

selects the smoothing method for normalizing the data. When \textit{MethodValue} is 'lowess', \texttt{mainvarsetnorm} uses the lowess method. When \textit{MethodValue} is 'runmedian', \texttt{mainvarsetnorm} uses the running median method. Default is 'lowess'.

\[ \text{NormDataY} = \text{mainvarsetnorm(..., 'Span', SpanValue, ...)} \]

sets the window size for the smoothing method. If \textit{SpanValue} is less than 1, the window size is that percentage of the number of data points. If \textit{SpanValue} is equal to or greater than 1, the window size is of size \textit{SpanValue}. Default is 0.05, which corresponds to a window size equal to 5% of the total number of data points in the invariant set.

\[ \text{NormDataY} = \text{mainvarsetnorm(..., 'Showplot', ShowplotValue, ...)} \]

determines whether to plot a pair of M-A scatter plots (before and after normalization). M is the ratio between DataX and DataY. A is the average of DataX and DataY. When \textit{ShowplotValue} is true, \texttt{mainvarsetnorm} plots the M-A scatter plots. Default is false.

Examples

Normalize Microarray Data

This example illustrates how to correct for dye bias or scanning differences between two channels of data from a two-color microarray experiment.

Read microarray data from a sample GPR file.

\begin{verbatim}
maStruct = gprread('mouse_a1wt.gpr');
\end{verbatim}

Extract gene expression values from two different experimental conditions.

\begin{verbatim}
cy5data = magetfield(maStruct, 'F635 Median');
cy3data = magetfield(maStruct, 'F532 Median');
\end{verbatim}

Normalize \texttt{cy3data} using \texttt{cy5data} as reference and plot the results.

\begin{verbatim}
Normcy3data = mainvarsetnorm(cy5data, cy3data, 'showplot', true);
\end{verbatim}
Under perfect experimental conditions, data points with equal expression values would fall along the $M = 0$ line, which represents a gene expression ratio of 1. However, dye bias caused the measured values in one channel to be higher than the other channel, as seen in the Before normalization plot. Normalization corrected the variance, as seen in the After normalization plot.

References


See Also

affyinvarsetnorm | malowess | manorm | quantilenorm

Introduced in R2006a
mairplot

Create intensity versus ratio scatter plot of microarray data

Syntax

mairplot(DataX, DataY)
[Intensity, Ratio] = mairplot(DataX, DataY)
[Intensity, Ratio, H] = mairplot(DataX, DataY)
...
= mairplot(..., 'Type', TypeValue, ...)
= mairplot(..., 'LogTrans', LogTransValue, ...)
= mairplot(..., 'FactorLines', FactorLinesValue, ...)
= mairplot(..., 'Title', TitleValue, ...)
= mairplot(..., 'Labels', LabelsValue, ...)
= mairplot(..., 'Normalize', NormalizeValue, ...)
= mairplot(..., 'LowessOptions', LowessOptionsValue, ...)
= mairplot(..., 'Showplot', ShowplotValue, ...)
= mairplot(..., 'PlotOnly', PlotOnlyValue, ...)

Input Arguments

<p>| <strong>DataX, DataY</strong> | DataMatrix object on page 1-532 or vector of gene expression values where each row corresponds to a gene. For example, in a two-color microarray experiment, DataX could be cy3 intensity values and DataY could be cy5 intensity values. |
| <strong>TypeValue</strong> | Character vector or string that specifies the plot type. Choices are 'IR' (plots log_{10} of the product of the DataX and DataY intensities versus log_2 of the intensity ratios) or 'MA' (plots (1/2)log_2 of the product of the DataX and DataY intensities versus log_2 of the intensity ratios). Default is 'IR'. |
| <strong>LogTransValue</strong> | Controls the conversion of data in X and Y from natural scale to log_2 scale. Set LogTransValue to false, when the data is already log_2 scale. Default is true, which assumes the data is natural scale. |
| <strong>FactorLinesValue</strong> | Adds lines to the plot showing a factor of N change. Default is 2, which corresponds to a level of 1 and -1 on a log_2 scale. |
| <strong>Tip</strong> | You can also change the factor lines interactively, after creating the plot. |
| <strong>TitleValue</strong> | Character vector or string that specifies a title for the plot. |
| <strong>LabelsValue</strong> | Cell array of character vectors or string vector containing labels for the data. If labels are defined, then clicking a point on the plot shows the label corresponding to that point. |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NormalizeValue</td>
<td>Controls the display of lowess normalized ratio values. Enter <code>true</code> to display lowess normalized ratio values. Default is <code>false</code>.</td>
</tr>
<tr>
<td>Tip</td>
<td>You can also normalize the data from the MAIR Plot window, after creating the plot.</td>
</tr>
<tr>
<td>LowessOptionsValue</td>
<td>Cell array of one, two, or three property name/value pairs in any order that affect the lowess normalization. Choices for property name/value pairs are:</td>
</tr>
<tr>
<td></td>
<td>• 'Order', <code>OrderValue</code></td>
</tr>
<tr>
<td></td>
<td>• 'Robust', <code>RobustValue</code></td>
</tr>
<tr>
<td></td>
<td>• 'Span', <code>SpanValue</code></td>
</tr>
<tr>
<td></td>
<td>For more information on the preceding property name/value pairs, see <code>malowess</code>.</td>
</tr>
<tr>
<td>ShowplotValue</td>
<td>Controls the display of the scatter plot. Choices are <code>true</code> (default) or <code>false</code>.</td>
</tr>
<tr>
<td>PlotOnlyValue</td>
<td>Controls the display of the scatter plot without user interface components. Choices are <code>true</code> or <code>false</code> (default).</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> If you set the 'PlotOnly' property to <code>true</code>, you can still display labels for data points by clicking a data point, and you can still adjust the horizontal fold change lines by click-dragging the lines.</td>
</tr>
</tbody>
</table>

### Output Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intensity</td>
<td>DataMatrix object on page 1-532 or vector containing intensity values for the microarray gene expression data, calculated as:</td>
</tr>
<tr>
<td></td>
<td>• $\log_{10}$ of the product of the <code>DataX</code> and <code>DataY</code> intensities (when <code>Type</code> is 'IR')</td>
</tr>
<tr>
<td></td>
<td>• $(1/2)\log_2$ of the product of the <code>DataX</code> and <code>DataY</code> intensities (when <code>Type</code> is 'MA')</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> If <code>DataX</code> or <code>DataY</code> is a DataMatrix object, then <code>Intensity</code> is also a DataMatrix object with the same properties.</td>
</tr>
<tr>
<td>Ratio</td>
<td>DataMatrix object on page 1-532 or vector containing ratios of the microarray gene expression data, calculated as $\log_2(DataX/\text{DataY})$.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> If <code>DataX</code> or <code>DataY</code> is a DataMatrix object, then <code>Ratio</code> is also a DataMatrix object with the same properties.</td>
</tr>
<tr>
<td>H</td>
<td>Handle of the plot.</td>
</tr>
</tbody>
</table>
**Description**

`mairplot(DataX, DataY)` creates a scatter plot that plots log\(_{10}\) of the product of the `DataX` and `DataY` intensities versus log\(_2\) of the intensity ratios.

\[ [\text{Intensity, Ratio}] = \text{mairplot}(\text{DataX, DataY}) \] returns the intensity and ratio values. If you set 'Normalize' to true, the returned ratio values are normalized.

\[ [\text{Intensity, Ratio, H}] = \text{mairplot}(\text{DataX, DataY}) \] returns the handle of the plot.

`... = \text{mairplot}(..., 'PropertyName', PropertyValue, ...)` calls `mairplot` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

- `... = \text{mairplot}(..., 'Type', TypeValue, ...)` specifies the plot type. Choices are 'IR' (plots log\(_{10}\) of the product of the `DataX` and `DataY` intensities versus log\(_2\) of the intensity ratios) or 'MA' (plots (1/2)log\(_2\) of the product of the `DataX` and `DataY` intensities versus log\(_2\) of the intensity ratios). Default is 'IR'.

- `... = \text{mairplot}(..., 'LogTrans', LogTransValue, ...)` controls the conversion of data in `X` and `Y` from natural to log\(_2\) scale. Set `LogTransValue` to false, when the data is already log\(_2\) scale. Default is true, which assumes the data is natural scale.

- `... = \text{mairplot}(..., 'FactorLines', FactorLinesValue, ...)` adds lines to the plot showing a factor of `N` change. Default is 2, which corresponds to a level of 1 and -1 on a log\(_2\) scale.

**Tip** You can also change the factor lines interactively, after creating the plot.

- `... = \text{mairplot}(..., 'Title', TitleValue, ...)` specifies a title for the plot.

- `... = \text{mairplot}(..., 'Labels', LabelsValue, ...)` specifies a cell array of character vectors or string vector of labels for the data. If labels are defined, then clicking a point on the plot shows the label corresponding to that point.

- `... = \text{mairplot}(..., 'Normalize', NormalizeValue, ...)` controls the display of lowess normalized ratio values. Enter true to display to lowess normalized ratio values. Default is false.

**Tip** You can also normalize the data from the MAIR Plot window, after creating the plot.

- `... = \text{mairplot}(..., 'LowessOptions', LowessOptionsValue, ...)` lets you specify up to three property name/value pairs (in any order) that affect the lowess normalization. Choices for property name/value pairs are:
  - 'Order', `OrderValue`
  - 'Robust', `RobustValue`
  - 'Span', `SpanValue`

For more information on the previous three property name/value pairs, see the `malowess` function.
... = mairplot(..., 'Showplot', ShowplotValue, ...) controls the display of the scatter plot. Choices are true (default) or false.

... = mairplot(..., 'PlotOnly', PlotOnlyValue, ...) controls the display of the scatter plot without user interface components. Choices are true or false (default).

**Note** If you set the 'PlotOnly' property to true, you can still display labels for data points by clicking a data point, and you can still adjust the horizontal fold change lines by click-dragging the lines.

Following is an IR plot of normalized data.

Following is an MA plot of unnormalized data.
The intensity versus ratio scatter plot displays the following:

- \( \log_{10}(\text{Intensity}) \) versus \( \log_2(\text{Ratio}) \) scatter plot of genes.
- Two horizontal fold change lines at a fold change level of 2, which corresponds to a ratio of 1 and -1 on a \( \log_2(\text{Ratio}) \) scale. (Lines will be at different fold change levels, if you used the 'FactorLines' property.)
- Data points for genes that are considered differentially expressed (outside of the fold change lines) appear in orange.

After you display the intensity versus ratio scatter plot, you can interactively do the following:

- Adjust the horizontal fold change lines by click-dragging one line or entering a value in the Fold Change text box, then clicking Update.
- Display labels for data points by clicking a data point.
- Select a gene from the Up Regulated or Down Regulated list to highlight the corresponding data point in the plot. Press and hold Ctrl or Shift to select multiple genes.
- Zoom the plot by selecting Tools > Zoom In or Tools > Zoom Out.
- View lists of significantly up-regulated and down-regulated genes, and optionally, export the gene labels and indices to a structure in the MATLAB Workspace by clicking Export.
• Normalize the data by clicking the **Normalize** button, then selecting whether to show the normalized plot in a separate window. If you show the normalized plot in a separate window, the **Show smooth curve** check box becomes available in the original (unnormalized) plot.

**Tip** To select different lowess normalization options before normalizing, select **Tools > Set LOWESS Normalization Options**, then enter options in the Options for LOWESS dialog box.

**Examples**

1. Use the `gprread` function to create a structure containing microarray data.

   ```matlab```
   maStruct = gprread('mouse_a1wt.gpr');
   ```matlab```

2. Use the `magetfield` function to extract the green (cy3) and red (cy5) signals from the structure.

   ```matlab```
   cy5data = magetfield(maStruct,'F635 Median');
   cy3data = magetfield(maStruct,'F532 Median');
   ```matlab```

3. Create an intensity versus ratio scatter plot of the cy3 and cy5 data. Normalize the data and add a title and labels:

   ```matlab```
   mairplot(cy5data, cy3data, 'Normalize', true, ...
   'Title','Normalized R vs G IR plot', ...
   'Labels', maStruct.Names)
4 Return intensity values and ratios without displaying the plot.

   [intensities, ratios] = mairplot(cy5data, cy3data, 'Showplot', false);

5 Create a normalized MA plot of the cy3 and cy5 data without the user interface components.

   mairplot(cy5data, cy3data, 'Normalize', true, ...
            'Type','MA','PlotOnly',true)
References


See Also
maboxplot | magetfield | maimage | mainvarsetnorm | maloglog | malowess | manorm | mattest | mavolcanoplot

Introduced before R2006a
maloglog

Create loglog plot of microarray data

Syntax

maloglog(X, Y)

maloglog(X, Y, ...'FactorLines', N, ...)
maloglog(X, Y, ...'Title', TitleValue, ...)
maloglog(X, Y, ...'Labels', LabelsValues, ...)
maloglog(X, Y, ...'HandleGraphicsName', HGValue, ...)
H = maloglog(...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X, Y</td>
<td>DataMatrix object on page 1-532s or numeric array of microarray expression</td>
</tr>
<tr>
<td>N</td>
<td>Property to add two lines to the plot showing a factor of N change.</td>
</tr>
<tr>
<td>TitleValue</td>
<td>A character vector or string to use as the title for the plot.</td>
</tr>
<tr>
<td>LabelsValues</td>
<td>A cell array of character vectors or string vector containing labels for</td>
</tr>
<tr>
<td></td>
<td>the data in X and Y. If you specify LabelsValues, then clicking a data</td>
</tr>
<tr>
<td></td>
<td>point in the plot shows the label corresponding to that point.</td>
</tr>
</tbody>
</table>

Description

maloglog(X, Y) creates a loglog scatter plot of X versus Y. X and Y are DataMatrix object on page 1-532s or numeric arrays of microarray expression values from two different experimental conditions.

maloglog(X, Y, ...'PropertyName', PropertyValue, ...) calls maloglog with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

maloglog(X, Y, ...'FactorLines', N, ...) adds two lines to the plot showing a factor of N change.

maloglog(X, Y, ...'Title', TitleValue, ...) allows you to specify a title for the plot.

maloglog(X, Y, ...'Labels', LabelsValues, ...) allows you to specify a cell array of character vectors or string vector containing labels for the data. If LabelsValues is defined, then clicking a data point in the plot shows the label corresponding to that point.

maloglog(X, Y, ...'HandleGraphicsName', HGValue, ...) allows you to pass optional Handle Graphics property name/property value pairs to the function.

H = maloglog(...) returns the handle to the plot.
Examples

maStruct = gprread('mouse_a1wt.gpr');
Red = magetfield(maStruct,'F635 Median');
Green = magetfield(maStruct,'F532 Median');
maloglog(Red,Green,'title','Red vs Green');
% Add factorlines and labels
figure
maloglog(Red,Green,'title','Red vs Green',...'
FactorLines',2,'LABELS',maStruct.Names);  
% Now create a normalized plot
figure
maloglog(manorm(Red),manorm(Green),'title',...'
Normalised Red vs Green','FactorLines',2,...'
LABELS',maStruct.Names);

See Also
loglog | maboxplot | magetfield | mairplot | malowess | manorm | mattest | mavolcanoplot

Introduced before R2006a
malowess

Smooth microarray data using Lowess method

Syntax

YSmooth = malowess(X, Y)
YSmooth = malowess(X, Y, ...'Order', OrderValue, ...)
YSmooth = malowess(X, Y, ...'Robust', RobustValue, ...)
YSmooth = malowess(X, Y, ...'Span', SpanValue, ...)

Arguments

<table>
<thead>
<tr>
<th>X, Y</th>
<th>DataMatrix object on page 1-532 or numeric vector containing scatter data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrderValue</td>
<td>Property to select the order of the algorithm. Enter either 1 (linear fit) or 2 (quadratic fit). The default order is 1.</td>
</tr>
<tr>
<td>RobustValue</td>
<td>Property to select a robust fit. Enter either true or false.</td>
</tr>
<tr>
<td>SpanValue</td>
<td>Property to specify the window size. The default value is 0.05 (5% of total points in X)</td>
</tr>
</tbody>
</table>

Description

YSmooth = malowess(X, Y) smooths scatter data in X and Y using the Lowess smoothing method. The default window size is 5% of the length of X. YSmooth is a numeric vector or, if Y is a DataMatrix object, also a DataMatrix object with the same properties as Y.

YSmooth = malowess(X, Y, ...'PropertyName', PropertyValue, ...) calls malowess with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

YSmooth = malowess(X, Y, ...'Order', OrderValue, ...) chooses the order of the algorithm. Note that the Curve Fitting Toolbox™ software refers to Lowess smoothing of order 2 as Loess smoothing.

YSmooth = malowess(X, Y, ...'Robust', RobustValue, ...) uses a robust fit when RobustValue is set to true. This option can take a long time to calculate.

YSmooth = malowess(X, Y, ...'Span', SpanValue, ...) modifies the window size for the smoothing function. If SpanValue is less than 1, the window size is taken to be a fraction of the number of points in the data. If SpanValue is greater than 1, the window is of size SpanValue.

Examples

maStruct = gprread('mouse_a1wt.gpr');
cy5data = magetfield(maStruct, 'F635 Median');
cy3data = magetfield(maStruct, 'F532 Median');
[x,y] = mairplot(cy5data, cy3data);
drawnow
ysmooth = malowess(x,y);
hold on;
plot(x, ysmooth, 'rx')
ynorm = y - ysmooth;

See Also
affyinvarsetnorm | maboxplot | magetfield | maimage | mainvarsetnorm | mairplot |
maloglog | manorm | quantilenorm | robustfit

Introduced before R2006a
manorm

Normalize microarray data

Syntax

\[ X_{\text{Norm}} = \text{manorm}(X) \]
\[ X_{\text{Norm}} = \text{manorm}(\text{MAStruct}, \text{FieldName}) \]
\[ [X_{\text{Norm}}, \text{ColVal}] = \text{manorm}(...) \]

manorm(..., 'Method', MethodValue, ...)
manorm(..., 'Extra_Args', Extra_ArgsValue, ...)
manorm(..., 'LogData', LogDataValue, ...)
manorm(..., 'Percentile', PercentileValue, ...)
manorm(..., 'Global', GlobalValue, ...)
manorm(..., 'StructureOutput', StructureOutputValue, ...)
manorm(..., 'NewColumnName', NewColumnNameValue, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Numeric array or DataMatrix object on page 1-532 of microarray data.</td>
</tr>
<tr>
<td>MASTruct</td>
<td>Microarray structure.</td>
</tr>
<tr>
<td>FieldName</td>
<td>Field.</td>
</tr>
</tbody>
</table>

Description

\( X_{\text{Norm}} = \text{manorm}(X) \) scales the values in each column of \( X \), a numeric array or DataMatrix object on page 1-532 of microarray data, by dividing by the mean column intensity. \( X_{\text{Norm}} \) is a vector, matrix, or DataMatrix object on page 1-532 of normalized microarray data.

\( X_{\text{Norm}} = \text{manorm}(\text{MAStruct}, \text{FieldName}) \) scales the data in \( \text{MAStruct} \), a microarray structure, for a field specified by \( \text{FieldName} \), for each block or print-tip by dividing each block by the mean column intensity. The output is a matrix with each column corresponding to the normalized data for each block.

\([X_{\text{Norm}}, \text{ColVal}] = \text{manorm}(...)\) returns the values used to normalize the data.

manorm(..., 'PropertyName', PropertyValue, ...) calls manorm with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

- manorm(..., 'Method', MethodValue, ...) allows you to choose the method for scaling or centering the data. MethodValue can be 'Mean' (default), 'Median', 'STD' (standard deviation), 'MAD' (median absolute deviation), or a function handle. If you pass a function handle, then the function should ignore NaNs and must return a single value per column of the input data.

- manorm(..., 'Extra_Args', Extra_ArgsValue, ...) allows you to pass extra arguments to the function MethodValue. Extra_ArgsValue must be a cell array.
manorm(..., 'LogData', LogDataValue, ...), when LogDataValue is true, works with log ratio data in which case the mean (or MethodValue) of each column is subtracted from the values in the columns, instead of dividing the column by the normalizing value.

manorm(..., 'Percentile', PercentileValue, ...) only uses the percentile (PercentileValue) of the data preventing large outliers from skewing the normalization. If PercentileValue is a vector containing two values, then the range from the PercentileValue(1) percentile to the PercentileValue(2) percentile is used. The default value is 100, that is to use all the data in the data set.

manorm(..., 'Global', GlobalValue, ...) when GlobalValue is true, normalizes the values in the data set by the global mean (or MethodValue) of the data, as opposed to normalizing each column or block of the data independently.

manorm(..., 'StructureOutput', StructureOutputValue, ...), when StructureOutputValue is true, the input data is a structure returns the input structure with an additional data field for the normalized data.

manorm(..., 'NewColumnName', NewColumnNameValue, ...), when using StructureOutput, allows you to specify the name of the column that is appended to the list of ColumnNames in the structure. The default behavior is to prefix 'Block Normalized' to FieldName.

Examples

maStruct = gprread('mouse_a1wt.gpr');
% Extract some data of interest.
Red = magetfield(maStruct,'F635 Median');
Green = magetfield(maStruct,'F532 Median');
% Create a log-log plot.
maloglog(Red,Green,'factorlines',true)
% Center the data.
normRed = manorm(Red);
normGreen = manorm(Green);
% Create a log-log plot of the centered data.
figure
maloglog(normRed,normGreen,'title','Normalized','factorlines',true)

% Alternatively, you can work directly with the structure
normRedBs = manorm(maStruct,'F635 Median - B635');
normGreenBs = manorm(maStruct,'F532 Median - B532');
% Create a log-log plot of the centered data. This includes some
% zero values so turn off the warning.
figure
w = warning('off','Bioinfo:maloglog:ZeroValues');
warning('off','Bioinfo:maloglog:NegativeValues');
maloglog(normRedBs,normGreenBs,'title',
'Normalized Background-Subtracted Median Values',...
'factorlines',true)
warning(w);

See Also
affyinvarsetnorm | maboxplot | magetfield | mainvarsetnorm | mairplot | maloglog | malowess | quantilenorm | rmasummary

Introduced before R2006a
mapcaplot

Create Principal Component Analysis (PCA) plot of microarray data

Syntax

mapcaplot(data)
mapcaplot(data,labels)

Description

mapcaplot(data) creates 2-D scatter plots of principal components of data. Once you plot the principal components, you can:

- Select principal components for the x and y axes from the drop-down list below each scatter plot.
- Click a data point to display its label.
- Select a subset of data points by dragging a box around them. Points in the selected region and the corresponding points in the other axes are then highlighted. The labels of the selected data points appear in the list box.
- Select a label in the list box to highlight the corresponding data point in the plot. Press and hold Ctrl or Shift to select multiple data points.
- Export the gene labels and indices to the MATLAB workspace.

mapcaplot(data,labels) labels the data points in the PCA plots using labels, instead of the row numbers.

Examples

Create PCA Plot of Microarray Data

Create a PCA plot to visualize genes involved during the metabolic shift from fermentation to respiration of yeast (Saccharomyces cerevisiae).

Load the data file that contains filtered yeast microarray data. The data comes from an experiment (DeRisi et al., 1997) that used DNA microarrays to study temporal gene expression of these genes. Expression levels were measured at seven time points during the diauxic shift.

load filteredyeastdata

This MAT-file includes three variables:

- yeastvalues — A matrix of gene expression data from Saccharomyces cerevisiae (yeast) during the metabolic shift from fermentation to respiration
- genes — A cell array of GenBank® accession numbers for labeling the rows in yeastvalues
- times — A vector of time values for labeling the columns in yeastvalues

Perform PCA on the expression data and plot the result.

mapcaplot(yeastvalues, genes)
Select a subset of data points by dragging a box around them. The data points are highlighted and their corresponding labels appear in **Selected Data**. You can then export the selected data to the workspace by selecting **Export**.
Input Arguments

data — Microarray expression profile data
numeric array | DataMatrix object

Microarray expression profile data, specified as a numeric array or DataMatrix object.

Data Types: double
labels — Data point labels

*cell array of character vectors | string vector*

Data point labels, specified as a cell array of character vectors or string vector.

Data Types: `string` | `cell`

**References**


**See Also**

`clustergram` | `mattest` | `mavolcanoplot` | `pca`

**Introduced before R2006a**
mattest

Perform two-sample t-test to evaluate differential expression of genes from two experimental conditions or phenotypes.

Syntax

- `PValues = mattest(DataX, DataY)`
- `[PValues, TScores] = mattest(DataX, DataY)`
- `[PValues, TScores, DFs] = mattest(DataX, DataY)`
- `... = mattest(..., 'VarType', VarTypeValue, ...)`
- `... = mattest(..., 'Permute', PermuteValue, ...)`
- `... = mattest(..., 'Bootstrap', BootstrapValue, ...)`
- `... = mattest(..., 'Showhist', ShowhistValue, ...)`
- `... = mattest(..., 'Showplot', ShowplotValue, ...)`
- `... = mattest(..., 'Labels', LabelsValue, ...)`

Input Arguments

- **DataX, DataY**: DataMatrix object or a matrix of gene expression values where each row corresponds to a gene and each column corresponds to a replicate. `DataX` and `DataY` must have the same number of rows and are assumed to be normally distributed in each class with equal variances.

  `DataX` contains data from one experimental condition and `DataY` contains data from a different experimental condition. For example, `DataX` could be expression values from cancer cells, and `DataY` could be expression values from normal cells.

- **VarTypeValue**: Character vector that specifies the variance type of the test. `VarTypeValue` can be 'equal' or 'unequal' (default). If set to 'equal', `mattest` performs the test assuming the two samples have equal variances. If set to 'unequal', `mattest` performs the test assuming the two samples have unknown and unequal variances.

- **PermuteValue**: Controls whether permutation tests are run, and if so, how many. Choices are `true`, `false` (default), or any integer greater than 2. If set to `true`, the number of permutations is 1000.

- **BootstrapValue**: Controls whether bootstrap tests are run, and if so, how many. Choices are `true`, `false` (default), or any integer greater than 2. If set to `true`, the number of bootstrap tests is 1000.

- **ShowhistValue**: Controls the display of histograms of t-score distributions and p-value distributions. Choices are `true` or `false` (default).

- **ShowplotValue**: Controls the display of a normal t-score quantile plot. Choices are `true` or `false` (default). In the t-score quantile plot, data points with t-scores $> (1 - 1/(2N))$ or $< 1/(2N)$ display with red circles. N is the total number of genes.
**LabelsValue**

Cell array of character vectors or string vector containing labels (typically gene names or probe set IDs) for each row in DataX and DataY. The labels display if you click a data point in the t-score quantile plot.

### Output Arguments

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
</table>
| **PValues** | One of the following:  
  - Column vector of p-values for each gene in DataX and DataY (if both inputs are matrices).  
  - DataMatrix object on page 1-532 with row names the same as the first input DataMatrix object and a column name of p-values (if at least one input is a DataMatrix object). |
| **TScores** | Column vector of t-scores for each gene in DataX and DataY. |
| **DFs** | Column vector containing the degree of freedom for each gene in DataX and DataY. |

### Description

**PValues** = `mattest(DataX, DataY)` performs an unpaired t-test for differential expression with a standard two-tailed and two-sample t-test on every gene in DataX and DataY and returns a p-value for each gene. DataX and DataY are either a DataMatrix object on page 1-532 or a matrix of gene expression values, in which each row corresponds to a gene, and each column corresponds to a replicate. DataX contains data from one experimental condition and DataY contains data from another experimental condition. DataX and DataY must have the same number of rows and are assumed to be normally distributed in each class. **PValues** is a column vector of p-values for each gene, or, if at least one of the inputs is a DataMatrix object, a DataMatrix object with row names the same as the first input DataMatrix object and a column name of p-values.

**[PValues, TScores]** = `mattest(DataX, DataY)` also returns a t-score for each gene in DataX and DataY. **TScores** is a column vector of t-scores for each gene.

**[PValues, TScores, DFs]** = `mattest(DataX, DataY)` also returns **DFs**, a column vector containing the degree of freedom for each gene across both data sets, DataX and DataY.

... = `mattest(..., 'PropertyName', PropertyValue, ...)` calls `mattest` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each **PropertyName** must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = `mattest(..., 'VarType', VarTypeValue, ...)` specifies the variance type of the test. **VarTypeValue** can be 'equal' or 'unequal' (default). If set to 'equal', mattest performs the test assuming the two samples have equal variances. If set to 'unequal', mattest performs the test assuming the two samples have unknown and unequal variances.

... = `mattest(..., 'Permute', PermuteValue, ...)` controls whether permutation tests are run, and if so, how many. **PermuteValue** can be true, false (default), or any integer greater than 2. If set to true, the number of permutations is 1000.
... = mattest(..., 'Bootstrap', BootstrapValue, ...) controls whether bootstrap tests are run, and if so, how many. BootstrapValue can be true, false (default), or any integer greater than 2. If set to true, the number of bootstrap tests is 1000.

... = mattest(..., 'Showhist', ShowhistValue, ...) controls the display of histograms of t-score distributions and p-value distributions. When ShowhistValue is true, mattest displays histograms. Default is false.

... = mattest(..., 'Showplot', ShowplotValue, ...) controls the display of a normal t-score quantile plot. When ShowplotValue is true, mattest displays a quantile-quantile plot. Default is false. In the t-score quantile plot, the black diagonal line represents the sample quantile being equal to the theoretical quantile. Data points of genes considered to be differentially expressed lie farther away from this line. Specifically, data points with t-scores > \((1 - 1/(2N))\) or < \(1/(2N)\) display with red circles. N is the total number of genes.
... = mattest(..., 'Labels', LabelsValue, ...) controls the display of labels when you click a data point in the t-score quantile plot. LabelsValue is a cell array of character vectors or string vector containing labels (typically gene names or probe set IDs) for each row in DataX and DataY.

**Examples**

1. Load the MAT-file, included with the Bioinformatics Toolbox software, that contains Affymetrix data from a prostate cancer study, specifically probe intensity data from Affymetrix HG-U133A GeneChip arrays. The two variables in the MAT-file, dependentData and independentData, are two matrices of gene expression values from two experimental conditions.

   load prostatecancerexpdata

2. Calculate the p-values and t-scores for the gene expression values in the two matrices and display a normal t-score quantile plot.

   [pvalues,tscores] = mattest(dependentData, independentData,...
   'showplot',true);

3. Calculate the p-values and t-scores again using permutation tests (1000 permutations) and displaying histograms of t-score distributions and p-value distributions.

   [pvalues,tscores] = mattest(dependentData,independentData,...
   'permute',true,'showhist',true,...
   'showplot',true);

4. Calculate the p-values and t-scores again using bootstrap tests (2000 tests) and displaying histograms of t-score distributions and p-value distributions.
[pvalues, tscores] = mattest(dependentData, independentData,...
    'bootstrap', 2000, 'showhist', true,...
    'showplot', true);
The prostatecancerexpdata.mat file used in this example contains data from Best et al., 2005.

References


See Also

affygcma | affyrma | maboxplot | mafdr | mainvarsetnorm | mairplot | maloglog | malowess | manorm | mavolcanoplot | rmasummary

Introduced in R2006a
mavolcanoplot

Create significance versus gene expression ratio (fold change) scatter plot of microarray data

Syntax

mavolcanoplot(DataX, DataY, PValues)
SigStructure = mavolcanoplot(DataX, DataY, PValues)

... mavolcanoplot(..., 'Labels', LabelsValue, ...)
... mavolcanoplot(..., 'LogTrans', LogTransValue, ...)
... mavolcanoplot(..., 'PCutoff', PCutoffValue, ...)
... mavolcanoplot(..., 'Foldchange', FoldchangeValue, ...)
... mavolcanoplot(..., 'PlotOnly', PlotOnlyValue, ...)

Input Arguments

<table>
<thead>
<tr>
<th>DataX, DataY</th>
<th>DataMatrix object on page 1-532, matrix, or vector of gene expression values from a single experimental condition. If a DataMatrix object or a matrix, each row is a gene, each column is a sample, and an average expression value is calculated for each gene.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note</td>
<td>If the values in DataX or DataY are natural scale, use the LogTrans property to convert them to log₂ scale.</td>
</tr>
</tbody>
</table>
| PValues     | Either of the following:  
|             | • Column vector of p-values for each feature (for example, gene) in a data set, such as returned by mattest.  
|             | • DataMatrix object on page 1-532 containing p-values for each feature (for example, gene) in a data set, such as returned by mattest.                                                               |
| LabelsValue | Cell array of character vectors or string vector containing labels (typically gene names or probe set IDs) for the data. After creating the plot, you can click a data point to display the label associated with it. If you do not provide a LabelsValue, data points are labeled with row numbers from DataX and DataY. |
| LogTransValue | Property to control the conversion of data in DataX and DataY from natural scale to log₂ scale. Enter true to convert data to log₂ scale, or false. Default is false, which assumes data is already log₂ scale. |
| PCutoffValue | Lets you specify a cutoff p-value to define data points that are statistically significant. This value is displayed graphically as a horizontal line on the plot. Default is 0.05, which is equivalent to 1.3010 on the –log₁₀ (p-value) scale. The value must be between 0 and 1. |
| Note        | You can also change the p-value cutoff interactively after creating the plot.                                                                                                                        |
**FoldchangeValue**

Lets you specify a ratio fold change to define data points that are differentially expressed. Default is 2, which corresponds to a ratio of 1 and -1 on a log_2 (ratio) scale.

**Note** You can also change the fold change interactively after creating the plot.

**PlotOnlyValue**

Controls the display of the volcano plot without user interface components. Choices are true or false (default).

**Note** If you set the 'PlotOnly' property to true, you can still display labels for data points by clicking a data point, and you can still adjust vertical fold change lines and the horizontal p-value cutoff line by click-dragging the lines.

**Output Arguments**

**SigStructure**

Structure containing information for genes that are considered to be both statistically significant (above the p-value cutoff) and significantly differentially expressed (outside of the fold change values). The fields are listed below.

**Description**

`mavolcanoplot(DataX, DataY, PValues)` creates a scatter plot of gene expression data, plotting significance versus fold change of gene expression ratios of two data sets, *DataX* and *DataY*. It plots significance as the -log_{10} (p-value) from the input, *PValues*. *DataX* and *DataY* can be vectors, matrices, or DataMatrix object on page 1-532s. *PValues* is a column vector or DataMatrix object on page 1-532.

`SigStructure = mavolcanoplot(DataX, DataY, PValues)` returns a structure containing information for genes that are considered to be both statistically significant (above the p-value cutoff) and significantly differentially expressed (outside of the fold change values). The fields within `SigStructure` are sorted by p-value and include:

- Name
- PCutoff
- FCThreshold
- GeneLabels
- PValues
- FoldChanges

**Note** The fields PValues and FoldChanges will be either vectors or DataMatrix objects depending on the type of input PValues.

... `mavolcanoplot(..., 'PropertyName', PropertyValue, ...)` defines optional properties that use property name/value pairs in any order. These property name/value pairs are as follows:
... `mavolcanoplot(..., 'Labels', LabelsValue, ...)` lets you provide a cell array of character vectors or string vector containing labels (typically gene names or probe set IDs) for the data. After creating the plot, you can click a data point to display the label associated with it. If you do not provide a `LabelsValue`, data points are labeled with row numbers from `DataX` and `DataY`.

... `mavolcanoplot(..., 'LogTrans', LogTransValue, ...)` controls the conversion of data from `DataX` and `DataY` to log₂ scale. When `LogTransValue` is `true`, `mavolcanoplot` converts data from natural to log₂ scale. Default is `false`, which assumes the data is already log₂ scale.

... `mavolcanoplot(..., 'PCutoff', PCutoffValue, ...)` lets you specify a p-value cutoff to define data points that are statistically significant. This value displays graphically as a horizontal line on the plot. Default is 0.05, which is equivalent to 1.3010 on the -log₁₀ (p-value) scale.

**Note** You can also change the p-value cutoff interactively after creating the plot.

... `mavolcanoplot(..., 'Foldchange', FoldchangeValue, ...)` lets you specify a ratio fold change to define data points that are differentially expressed. Fold changes display graphically as two vertical lines on the plot. Default is 2, which corresponds to a ratio of 1 and -1 on a log₂ (ratio) scale.

**Note** You can also change the fold change interactively after creating the plot.

... `mavolcanoplot(..., 'PlotOnly', PlotOnlyValue, ...)` controls the display of the volcano plot without user interface components. Choices are `true` or `false` (default).

**Note** If you set the `PlotOnly` property to `true`, you can still display labels for data points by clicking a data point, and you can still adjust vertical fold change lines and the horizontal p-value cutoff line by click-dragging the lines.
The volcano plot displays the following:

- \(-\log_{10}(p\text{-value})\) versus \(\log_2\) (ratio) scatter plot of genes
- Two vertical fold change lines at a fold change level of 2, which corresponds to a ratio of 1 and \(-1\) on a \(\log_2\) (ratio) scale. (Lines will be at different fold change levels, if you used the \('\text{Foldchange}'\) property.)
- One horizontal line at the 0.05 \(p\text{-value}\) level, which is equivalent to 1.3010 on the \(-\log_{10}(p\text{-value})\) scale. (The line will be at a different \(p\text{-value}\) level, if you used the \('\text{p-value Cutoff}'\) property.)

After you display the volcano scatter plot, you can interactively:

- Adjust the vertical fold change lines by click-dragging one line or entering a value in the \(\text{Fold Change}\) text box.
- Adjust the horizontal \(p\text{-value}\) cutoff line by click-dragging or entering a value in the \(\text{p-value Cutoff}\) text box.
- Display labels for data points by clicking a data point.
- Select a gene from the \(\text{Up Regulated}\) or \(\text{Down Regulated}\) list to highlight the corresponding data point in the plot. Press and hold \(\text{Ctrl}\) or \(\text{Shift}\) to select multiple genes.
• Zoom the plot by selecting **Tools > Zoom In** or **Tools > Zoom Out**.
• View lists of significantly up-regulated and down-regulated genes and their associated p-values, and optionally, export the labels, p-values, and fold changes to a structure in the MATLAB Workspace by clicking **Export**.

**Examples**

1. Load a MAT-file, included with the Bioinformatics Toolbox software, which contains Affymetrix data variables, including `dependentData` and `independentData`, two matrices of gene expression values from two experimental conditions.
   ```matlab
   load prostatecancerexpdata
   ```

2. Use the `mattest` function to calculate p-values for the gene expression values in the two matrices.
   ```matlab
   pvalues = mattest(dependentData, independentData);
   ```

3. Using the two matrices, the `pvalues` calculated by `mattest`, and the `probesetIDs` column vector of labels provided, use `mavolcanoplot` to create a significance versus gene expression ratio scatter plot of the microarray data from the two experimental conditions.
   ```matlab
   mavolcanoplot(dependentData, independentData, pvalues,...
   'Labels', probesetIDs)
   ```

4. View the volcano plot without the user interface components.
   ```matlab
   mavolcanoplot(dependentData, independentData, pvalues,...
   'Labels', probesetIDs,'Plotonly', true)
   ```
The *prostatecancerexpdata.mat* file used in the previous example contains data from Best et al., 2005.

**References**


**See Also**

`maboxplot` | `maimage` | `mainvarsetnorm` | `mairplot` | `maloglog` | `malowess` | `manorm` | `mapcaplot` | `mattest`
Introduced in R2006a
max (DataMatrix)

Return maximum values in DataMatrix object

Syntax

\[
M = \text{max}(\text{DMObj1})
\]
\[
[M, \text{Indices}] = \text{max}(\text{DMObj1})
\]
\[
[M, \text{Indices}, \text{Names}] = \text{max}(\text{DMObj1})
\]
\[
... = \text{max}(\text{DMObj1}, [], \text{Dim})
\]
\[
\text{MA} = \text{max}(\text{DMObj1}, \text{DMObj2})
\]

Input Arguments

\begin{tabular}{|l|l|}
\hline
\textit{DMObj1, DMObj2} & DataMatrix objects, such as created by \texttt{DataMatrix} (object constructor).
\hline
\text{Note} \textit{DMObj1} and \textit{DMObj2} must be the same size, unless one is a scalar.
\hline
\textit{Dim} & Scalar specifying the dimension of \textit{DMObj} to return the maximum values. Choices are:
\begin{itemize}
  \item 1 — Default. Returns a row vector containing a maximum value for each column.
  \item 2 — Returns a column vector containing a maximum value for each row.
\end{itemize}
\hline
\end{tabular}

Output Arguments

\begin{tabular}{|l|l|}
\hline
\textit{M} & One of the following:
\begin{itemize}
  \item Scalar specifying the maximum value in \textit{DMObj} when it contains vector of data
  \item Row vector containing the maximum value for each column in \textit{DMObj} (when \textit{Dim} = 1)
  \item Column vector containing the maximum value for each row in \textit{DMObj} (when \textit{Dim} = 2)
\end{itemize}
\hline
\textit{Indices} & Either of the following:
\begin{itemize}
  \item Positive integer specifying the index of the maximum value in a DataMatrix object containing a vector of data
  \item Vector containing the indices for the maximum value in each column (if \textit{Dim} = 1) or row (if \textit{Dim} = 2) in a DataMatrix object containing a matrix of data
\end{itemize}
\hline
\textit{Names} & Vector of the row names (if \textit{Dim} = 1) or column names (if \textit{Dim} = 2) corresponding to the maximum value in each column or each row of a DataMatrix object.
\hline
\end{tabular}
**Description**

\[ M = \max(DMObj1) \] returns the maximum value(s) in \( DMObj1 \), a DataMatrix object. If \( DMObj1 \) contains a vector of data, \( M \) is a scalar. If \( DMObj1 \) contains a matrix of data, \( M \) is a row vector containing a maximum value in each column.

\[ [M, Indices] = \max(DMObj1) \] returns \( Indices \), the indices of the maximum value(s) in \( DMObj1 \), a DataMatrix object. If \( DMObj1 \) contains a vector of data, \( Indices \) is a positive integer. If \( DMObj1 \) contains a matrix of data, \( Indices \) is a vector containing the indices for the maximum value in each column (if \( \text{Dim} = 1 \)) or row (if \( \text{Dim} = 2 \)). If there are multiple maximum values in a column or row, the index for the first value is returned.

\[ [M, Indices, Names] = \max(DMObj1) \] returns \( Names \), a vector of the row names (if \( \text{Dim} = 1 \)) or column names (if \( \text{Dim} = 2 \)) corresponding to the maximum value in each column or each row of \( DMObj1 \), a DataMatrix object. If there are multiple maximum values in a column or row, the row or column name for the first value is returned.

\( \ldots = \max(DMObj1, [], \text{Dim}) \) specifies which dimension to return the maximum values for; that is each column or each row in a DataMatrix object. If \( \text{Dim} = 1 \), returns \( M \), a row vector containing the maximum value in each column. If \( \text{Dim} = 2 \), returns \( M \), a column vector containing the maximum value in each row. Default \( \text{Dim} = 1 \).

\[ MA = \max(DMObj1, DMObj2) \] returns \( MA \), a numeric array containing the larger of the two values from each position of \( DMObj1 \) and \( DMObj2 \). \( DMObj1 \) and \( DMObj2 \) can both be DataMatrix objects, or one can be a DataMatrix object and the other a numeric array. They must be the same size, unless one is a scalar. \( MA \) has the same size (number of rows and columns) as the first nonscalar input.

**See Also**

DataMatrix | min | sum

**Topics**

DataMatrix object on page 1-532

**Introduced in R2008b**
maxflow (biograph)

Calculate maximum flow in biograph object

Syntax

\[ \text{MaxFlow, FlowMatrix, Cut} = \text{maxflow(BGObj, } S\text{Node, } T\text{Node}) \]

\[ \ldots = \text{maxflow(BGObj, } S\text{Node, } T\text{Node, } \ldots'Capacity', \text{CapacityValue, } \ldots) \]

\[ \ldots = \text{maxflow(BGObj, } S\text{Node, } T\text{Node, } \ldots'Method', \text{MethodValue, } \ldots) \]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BGObj</strong></td>
<td>Biograph object created by biograph (object constructor).</td>
</tr>
<tr>
<td><strong>SNode</strong></td>
<td>Node in a directed graph represented by an N-by-N adjacency matrix extracted from biograph object, <strong>BGObj</strong>.</td>
</tr>
<tr>
<td><strong>TNode</strong></td>
<td>Node in a directed graph represented by an N-by-N adjacency matrix extracted from biograph object, <strong>BGObj</strong>.</td>
</tr>
<tr>
<td><strong>CapacityValue</strong></td>
<td>Column vector that specifies custom capacities for the edges in the N-by-N adjacency matrix. It must have one entry for every nonzero value (edge) in the N-by-N adjacency matrix. The order of the custom capacities in the vector must match the order of the nonzero values in the N-by-N adjacency matrix when it is traversed column-wise. By default, <strong>maxflow</strong> gets capacity information from the nonzero entries in the N-by-N adjacency matrix.</td>
</tr>
<tr>
<td><strong>MethodValue</strong></td>
<td>Character vector or string that specifies the algorithm used to find the minimal spanning tree (MST). Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'Edmonds' — Uses the Edmonds and Karp algorithm, the implementation of which is based on a variation called the <strong>labeling algorithm</strong>. Time complexity is ( O(N^2E^2) ), where ( N ) and ( E ) are the number of nodes and edges respectively.</td>
</tr>
<tr>
<td></td>
<td>• 'Goldberg' — Default algorithm. Uses the Goldberg algorithm, which uses the generic method known as preflow-push. Time complexity is ( O(N^2*sqrt(E)) ), where ( N ) and ( E ) are the number of nodes and edges respectively.</td>
</tr>
</tbody>
</table>

Description

**Tip** For introductory information on graph theory functions, see “Graph Theory Functions”.

\[ \text{MaxFlow, FlowMatrix, Cut} = \text{maxflow(BGObj, } S\text{Node, } T\text{Node}) \] calculates the maximum flow of a directed graph represented by an N-by-N adjacency matrix extracted from a biograph object, **BGObj**, from node **SNode** to node **TNode**. Nonzero entries in the matrix determine the capacity of the edges. Output **MaxFlow** is the maximum flow, and **FlowMatrix** is a sparse matrix with all the flow values for every edge. **FlowMatrix**(X,Y) is the flow from node X to node Y. Output
Cut is a logical row vector indicating the nodes connected to SNode after calculating the minimum cut between SNode and TNode. If several solutions to the minimum cut problem exist, then Cut is a matrix.

**Tip** The algorithm that determines Cut, all minimum cuts, has a time complexity of $O(2^N)$, where $N$ is the number of nodes. If this information is not needed, use the maxflow method without the third output.

```
[...] = maxflow(BGObj, SNode, TNode, ...'PropertyName', PropertyValue, ...)
```
calls maxflow with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

```
[...] = maxflow(BGObj, SNode, TNode, ...'Capacity', CapacityValue, ...)
```
lets you specify custom capacities for the edges. CapacityValue is a column vector having one entry for every nonzero value (edge) in the N-by-N adjacency matrix. The order of the custom capacities in the vector must match the order of the nonzero values in the matrix when it is traversed column-wise. By default, graphmaxflow gets capacity information from the nonzero entries in the matrix.

```
[...] = maxflow(BGObj, SNode, TNode, ...'Method', MethodValue, ...)
```
lets you specify the algorithm used to find the minimal spanning tree (MST). Choices are:

- *'Edmonds'* — Uses the Edmonds and Karp algorithm, the implementation of which is based on a variation called the *labeling algorithm*. Time complexity is $O(N^2E^2)$, where $N$ and $E$ are the number of nodes and edges respectively.
- *'Goldberg'* — Default algorithm. Uses the Goldberg algorithm, which uses the generic method known as *preflow-push*. Time complexity is $O(N^2sqrt(E))$, where $N$ and $E$ are the number of nodes and edges respectively.

### References


### See Also

allshortestpaths | biograph | conncomp | graphmaxflow | isdag | isomorphism |  
isspantree | mspantree | shortestpath | topoorder | traverse

### Topics

biograph object on page 1-185

### Introduced in R2006b
mean (DataMatrix)

Return average or mean values in DataMatrix object

Syntax

\[ M = \text{mean}(\text{DMObj}) \]
\[ M = \text{mean}(\text{DMObj}, \text{Dim}) \]
\[ M = \text{mean}(\text{DMObj}, \text{Dim}, \text{IgnoreNaN}) \]

Input Arguments

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMObj</td>
<td>DataMatrix object, such as created by DataMatrix (object constructor).</td>
</tr>
<tr>
<td>Dim</td>
<td>Scalar specifying the dimension of DMObj to calculate the means. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 1 — Default. Returns mean values for elements in each column.</td>
</tr>
<tr>
<td></td>
<td>• 2 — Returns mean values for elements in each row.</td>
</tr>
<tr>
<td>IgnoreNaN</td>
<td>Specifies if NaNs should be ignored. Choices are true (default) or false.</td>
</tr>
</tbody>
</table>

Output Arguments

<table>
<thead>
<tr>
<th>M</th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Row vector containing the mean values from elements in each column in DMObj (when Dim = 1)</td>
</tr>
<tr>
<td></td>
<td>• Column vector containing the mean values from elements in each row in DMObj (when Dim = 2)</td>
</tr>
</tbody>
</table>

Description

\[ M = \text{mean}(\text{DMObj}) \] returns the mean values of the elements in the columns of a DataMatrix object, treating NaNs as missing values. \( M \) is a row vector containing the mean values for elements in each column in DMObj.

\[ M = \text{mean}(\text{DMObj}, \text{Dim}) \] returns the mean values of the elements in the columns or rows of a DataMatrix object, as specified by Dim. If Dim = 1, returns \( M \), a row vector containing the mean values for elements in each column in DMObj. If Dim = 2, returns \( M \), a column vector containing the mean values for elements in each row in DMObj. Default Dim = 1.

\[ M = \text{mean}(\text{DMObj}, \text{Dim}, \text{IgnoreNaN}) \] specifies if NaNs should be ignored. IgnoreNaN can be true (default) or false.

See Also

DataMatrix | max | median | min | sum

Topics

DataMatrix object on page 1-532
Introduced in R2008b
median (DataMatrix)

Return median values in DataMatrix object

Syntax

\[ Med = \text{median}(\text{DMObj}) \]
\[ Med = \text{median}(\text{DMObj}, \text{Dim}) \]
\[ Med = \text{median}(\text{DMObj}, \text{Dim}, \text{IgnoreNaN}) \]

Input Arguments

<table>
<thead>
<tr>
<th>DMObj</th>
<th>DataMatrix object, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dim</td>
<td>Scalar specifying the dimension of DMObj to calculate the medians.</td>
</tr>
<tr>
<td></td>
<td>Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 1 — Default. Returns median values for elements in each column.</td>
</tr>
<tr>
<td></td>
<td>• 2 — Returns median values for elements in each row.</td>
</tr>
<tr>
<td>IgnoreNaN</td>
<td>Specifies if NaNs should be ignored. Choices are true (default) or false.</td>
</tr>
</tbody>
</table>

Output Arguments

| Med                                | Either of the following:                                               |
|                                    | • Row vector containing the median values from elements in each column in DMObj (when Dim = 1) |
|                                    | • Column vector containing the median values from elements in each row in DMObj (when Dim = 2) |

Description

\[ Med = \text{median}(\text{DMObj}) \] returns the median values of the elements in the columns of a DataMatrix object, treating NaNs as missing values. \textit{Med} is a row vector containing the median values for elements in each column in DMObj.

\[ Med = \text{median}(\text{DMObj}, \text{Dim}) \] returns the median values of the elements in the columns or rows of a DataMatrix object, as specified by \textit{Dim}. If \textit{Dim} = 1, returns \textit{Med}, a row vector containing the median values for elements in each column in DMObj. If \textit{Dim} = 2, returns \textit{Med}, a column vector containing the median values for elements in each row in DMObj. Default \textit{Dim} = 1.

\[ Med = \text{median}(\text{DMObj}, \text{Dim}, \text{IgnoreNaN}) \] specifies if NaNs should be ignored. \textit{IgnoreNaN} can be true (default) or false.

See Also

DataMatrix | max | mean | min | sum

Topics

DataMatrix object on page 1-532
Introduced in R2008b
metafeatures

Attractor metagene algorithm for feature engineering using mutual information-based learning

Syntax

```
M = metafeatures(X)
[M,W] = metafeatures(X)
[M,W,GSorted] = metafeatures(X,G)
[M,W,GSorted,GSortedInd] = metafeatures( ___ )
[ ___ ] = metafeatures( ___ ,Name,Value)
[ ___ ] = metafeatures(T)
[ ___ ] = metafeatures(T,Name,Value)
```

Description

`M = metafeatures(X)` returns the weighted sums of features `M` in `X` using the attractor metagene algorithm described in [1].

`M` is a `r`-by-`n` matrix. `r` is the number of metafeatures identified during each repetition of the algorithm. The default number of repetitions is 1. By default, only unique metafeatures are returned in `M`. If multiple repetitions result in the same metafeature, then just one copy is returned in `M`. `n` is the number of samples (patients or time points).

`X` is a `p`-by-`n` numeric matrix. `p` is the number of variables, features, or genes. In other words, rows of `X` correspond to variables, such as measurements of gene expression for different genes. Columns correspond to different samples, such as patients or time points.

```
[M,W] = metafeatures(X) returns a `p`-by-`r` matrix `W` containing metafeatures weights. `M = W'*X`. `p` is the number of variables. `r` is the number of unique metafeatures or the number of times the algorithm is repeated (the default is 1).

[M,W,GSorted] = metafeatures(X,G) uses a `p`-by-1 cell array of character vectors or string vector `G` containing the variable names and returns a `p`-by-`r` cell array of variable names `GSorted` sorted by the decreasing weight.

The `i`th column of `GSorted` lists the feature (variable) names in order of their contributions to the `i`th metafeature.

```
[M,W,GSorted,GSortedInd] = metafeatures( ___ ) returns the indices `GSortedInd` such that `GSorted = G(GSortedInd)`.

[ ___ ] = metafeatures( ___ ,Name,Value) uses additional options specified by one or more `Name,Value` pair arguments.

[ ___ ] = metafeatures(T) uses a `p`-by-`n` table `T`. Gene names are the row names of the table. `M = W'*T{:,:}`.

[ ___ ] = metafeatures(T,Name,Value) uses additional options specified by one or more `Name,Value` pair arguments.
Note It is possible that the number of metafeatures \((r)\) returned in \(M\) can be fewer than the number of replicates (repetitions). Even though you may have set the number of replicates to a positive integer greater than 1, if each repetition returns the same metafeature, then \(r\) is 1, and \(M\) is 1-by-\(n\). This is because, by default, the function returns only unique metafeatures. If you prefer to get all metafeatures, set 'ReturnUnique' to false. A metafeature is considered unique if the Pearson correlation between it and all previously found metafeatures is less than the 'UniqueTolerance' value (the default value is 0.98).

Examples

Apply Attractor Metagene Algorithm to Gene Expression Data

Load the breast cancer gene expression data. The data was retrieved from the Cancer Genome Atlas (TCGA) on May 20, 2014 and contains gene expression data of 17814 genes for 590 different patients.

The expression data is stored in the variable geneExpression. The gene names are stored in the variable geneNames.

load TCGA_Breast_Gene_Expression

The data has several NaN values.

sum(sum(isnan(geneExpression)))

ans =

1695

Use the \(k\)-nearest neighbor imputation method to replace missing data with the corresponding value from an average of the \(k\) columns that are nearest.

geneExpression = knnImpute(geneExpression,3);

There are three common drivers of breast cancer: ERBB2, estrogen, and progesterone. metafeatures allows you to seed the starting weights to focus on the genes of interest. In this case, set the weight for each of these genes to 1 in three different rows of startValues. Each row corresponds to initial values for a different replicate (repetition).

erbb = find(strcmp('ERBB2',geneNames));
estrogen = find(strcmp('ESR1',geneNames));
progesterone = find(strcmp('PGR',geneNames));

startValues = zeros(size(geneExpression,1),3);
startValues(erbb,1) = 1;
startValues(estrogen,2) = 1;
startValues(progesterone,3) = 1;

Apply the attractor metagene algorithm to the imputed data.

[meta, weights, genes_sorted] = metafeatures(geneExpression,geneNames,'start',startValues);

The variable meta has the value of three metagenes discovered for each sample. Plot these three metagenes to gain insight into the nature of gene regulation across different phenotypes of breast cancer.

plot3(meta(1,:),meta(2,:),meta(3,:),'o')
xlabel('ERBB2 metagene')
Based on the plot, observe the following.

- There is a group of points clustered together with low values for all three metagenes. Based on mRNA levels, these could be triple-negative or basal type breast cancer.
- There is a group of points that have high estrogen receptor metagene expression and span across both high and low progestrone metagene expression. There are no points with high progestrone metagene expression and low estrogen metagene expression. This is consistent with the observation that ER-/PR+ breast cancers are extremely rare [3].
- The remaining points are the ERBB2 positive cancers. They have less representation in this data set than the hormone-driven and triple negative cancers.

**Input Arguments**

**X — Data**  
numeric matrix

Data, specified as a numeric matrix. Rows of X correspond to variables, such as measurements of gene expression. Columns correspond to different samples, such as patients or time points.

**G — Variable names**  
cell array of character vectors | string vector
Variable names, specified as a cell array of character vectors or string vector.

**T — Data**

table

Data, specified as a table. The row names of the table correspond to the names of features or genes, and the columns represent different samples, such as patients or time points.

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: 'Replicates',5 specifies to repeat the algorithm five times.

**Alpha — Tuning parameter for the number of metafeatures**

5 (default) | positive scalar

Tuning parameter for the number of metafeatures, specified as the comma-separated pair consisting of 'Alpha' and a positive number. This parameter controls the nonlinearity of the function that calculates the weights as described in the “Attractor Metagene Algorithm” on page 1-1069. As alpha increases, the number of metafeatures tends to increase. This parameter is often the most important parameter to adjust in the analysis of a data set.

Example: 'Alpha',3

**Start — Option for choosing initial weights**

'random' (default) | 'robust' | matrix

Option for choosing initial weights, specified as the comma-separated pair consisting of 'Start' and a character vector, string, or matrix. This table summarizes the available options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'random'</td>
<td>Initialize the weights to a vector of positive weights chosen uniformly at random and scaled such that they sum to 1. Choose a different initial weight vector for each replicate. This option is the default.</td>
</tr>
<tr>
<td>'robust'</td>
<td>If X or T has n columns, run the algorithm n times. On the ith evaluation of the algorithm, the weights are initialized to all zeros with the exception of the ith weight, which is set to 1. This option is useful when you are attempting to find all metafeatures of a data set.</td>
</tr>
<tr>
<td>matrix</td>
<td>n-by-r matrix of initial weights. The algorithm runs r times. The weights in the ith run of the algorithm are initialized to the ith column of the matrix.</td>
</tr>
</tbody>
</table>

Example: 'Start','robust'

**Replicates — Number of times to repeat the algorithm**

1 (default) | positive integer

Number of times to repeat the algorithm, specified as the comma-separated pair consisting of 'Replicates' and a positive integer. This option is valid only with the 'random' start option. The default is 1.

Example: 'Replicates',2
ReturnUnique — Unique metafeatures flag
truedefault|false

Unique metafeatures flag, specified as the comma-separated pair consisting of 'ReturnUnique' and true or false. If true, then only the unique metafeatures are returned. The default is true.

This option is useful when the algorithm is repeated multiple times. By setting this option to true, you choose to look at just the unique metafeatures since the same set of metafeatures can be discovered for different initializations.

A metafeature is considered unique if the Pearson correlation between it and all previously found metafeatures is less than the 'UniqueTolerance' value (the default value is 0.98).

To run the algorithm multiple times, set the 'Replicates' name-value pair argument or the 'Start' option to 'robust' or a matrix with more than 1 row.

Example: 'ReturnUnique',false

UniqueTolerance — Tolerance for metafeature uniqueness
0.98(default) | real number between 0 and 1

Tolerance for metafeature uniqueness, specified as the comma-separated pair consisting of 'UniqueTolerance' and a real number between 0 and 1.

A metafeature is considered unique if the Pearson correlation between it and all previously found metafeatures is less than the 'UniqueTolerance' value.

Example: 'UniqueTolerance',0.90

Options — Options for controlling the algorithm
[](default)|structure

Options for controlling the algorithm, specified as the comma-separated pair consisting of 'Options' and a structure. This table summarizes these options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display</td>
<td>Level of output display. Choices are 'off' or 'iter'. The default is 'off'.</td>
</tr>
<tr>
<td>MaxIter</td>
<td>Maximum number of iterations allowed. The default is 100.</td>
</tr>
<tr>
<td>Tolerance</td>
<td>If M changes by less than the tolerance in an iteration, then the algorithm stops. The default is 1e-6.</td>
</tr>
<tr>
<td>Streams</td>
<td>A RandStream object. If you do not specify any streams, metafeatures uses the default random stream.</td>
</tr>
<tr>
<td>UseParallel</td>
<td>Logical value indicating whether to perform calculations in parallel if a parallel pool and Parallel Computing Toolbox are available. For problems with large data sets relative to the available system memory, running in parallel can degrade performance. The default is false.</td>
</tr>
</tbody>
</table>

Example: 'Options',struct('Display','iter')

Output Arguments

M — Metafeatures
numeric matrix
Metafeatures, returned as a numeric matrix. It is an $r$-by-$n$ matrix containing the weighted sums of the features in $X$. $r$ is the number of replicates performed by the algorithm. $n$ is the number of different samples such as time points or patients.

**Note** It is possible that the number of metafeatures ($r$) returned in $M$ can be fewer than the number of replicates (repetitions). Even though you may have set the number of replicates to a positive integer greater than 1, if each repetition returns the same metafeature, then $r$ is 1, and $M$ is 1-by-$n$. This is because, by default, the function returns only unique metafeatures. If you prefer to get all metafeatures, set 'ReturnUnique' to false. A metafeature is considered unique if the Pearson correlation between it and all previously found metafeatures is less than the 'UniqueTolerance' value (the default value is 0.98).

$W$ — Metafeatures weights
numeric matrix

Metafeatures weights, returned as a numeric matrix. It is a $p$-by-$r$ matrix. $p$ is the number of variables. $r$ is the number of replicates performed by the algorithm.

$GSorted$ — Sorted variable names
cell array of character vectors

Sorted variable names, returned as a cell array of character vectors. It is a $p$-by-$r$ cell array. The names are sorted by decreasing weight. The $i$th column of the $GSorted$ lists the variable names in order of their contributions to $i$th metafeature.

If $GSorted$ is requested without $G$ or if $T.Properties.RowNames$ is empty, then the algorithm names each variable (feature) as $Var_i$, which corresponds to the $i$th row of $X$.

$GSortedInd$ — Index to $GSorted$
matrix

Index to $GSorted$, returned as a matrix of indices. It is a $p$-by-$r$ matrix. The indices satisfy $GSorted = G(GSortedInd)$ or $GSorted = T.Properties.RowNames(GSortedInd)$.

**More About**

**Attractor Metagene Algorithm**

The attractor metagene algorithm [1] is an iterative algorithm that converges to metagenes with important features. A metagene is defined as any weighted sum of gene expression using a nonlinear distance metric. The distance metric is a nonlinear variant of mutual information using binning and splines as described in [2]. In fact, the use of mutual information as a distance metric is one of major benefits of this algorithm since mutual information is a robust information theoretic approach to determine the statistical dependence between variables. Therefore, it is useful for analyzing relationships among gene expression. Another advantage is that the results of the algorithm tend to be more clearly linked with a phenotype defined by gene expression.

The algorithm is initialized by either random or user-specified weights and proceeds in these steps.

1. The estimate of a metagene during the $i$th iteration of the algorithm is $M_i = W_i^\ast G$, where $W_i$ is a vector of weights of size 1-by-$p$ (number of genes), and $G$ is the gene expression matrix of size $p$-by-$n$ (number of samples).
2 Update the weights by $W_{j,i+1} = J(M_i, G_j)$, where $W_{j,i+1}$ is the $j$th element of $W_{i+1}$, $G_j$ is the $j$th row of $G$, and $J$ is a similarity metric, which is defined as follows.

- If the Pearson correlation between $M_i$ and $G_j$ is greater than 0, then $J(M_i, G_j) = I(M_i, G_j)^\alpha$, where $I(M_i, G_j)$ is the measure of mutual information between two genes with minimum value 0 and maximum value 1, and $\alpha$ is any nonnegative number.
- If the correlation is less than or equal to 0, then $J(M_i, G_j) = 0$.

The algorithm iterates until the change in $W_i$ between iterations is less than the defined tolerance, that is, $\|W_i - W_{i-1}\| < \text{tolerance}$ or the maximum number of iterations is reached.

The Role of $\alpha$

In the similarity metric of the algorithm, the parameter $\alpha$ controls the degree of nonlinearity. As $\alpha$ increases, the number of metagenes tends to increase. If $\alpha$ is sufficiently large, then each gene approximately becomes an attractor metagene. If $\alpha$ is zero, then all weights remain equal to each other. Therefore, there is only one attractor metagene representing the average of all genes.

Therefore, adjusting $\alpha$ for the data set under consideration is a key step in fine tuning the algorithm. In the case of [1], using the TCGA data from several types of cancer to identify attractor metagenes, $\alpha$ value of 5 resulted in between 50 and 150 attractor metagenes discovered from the data.

References


Extended Capabilities

Automatic Parallel Support

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set the 'UseParallel' option to true.

Set the 'UseParallel' field of the options structure to true and specify the 'Options' name-value pair argument in the call to this function.

For example: 'Options', struct('UseParallel', true)

For more information, see the 'Options' name-value pair argument.

See Also

randfeatures | rankfeatures | relieff | sequentialfs
Topics
“Identifying Biomolecular Subgroups Using Attractor Metagenes”

Introduced in R2014b
**microplateplot**

Display visualization of microtiter plate

**Syntax**

```matlab
microplateplot(Data)
Handle = microplateplot(...)
```

```matlab
microplateplot(Data, ...'RowLabels', RowLabelsValue, ...)
microplateplot(Data, ...'ColumnLabels', ColumnLabelsValue, ...)
microplateplot(Data, ...'TextLabels', TextLabelsValue, ...)
microplateplot(Data, ...'TextFontSize', TextFontSizeValue, ...)
microplateplot(Data, ...'MissingValueColor', MissingValueColorValue, ...)
microplateplot(Data, ...'ToolTipFormat', ToolTipFormatValue, ...)
```

**Description**

`microplateplot(Data)` displays an image of a microtiter plate with each well colored according to intensity values, such as from a plate reader.

`Handle = microplateplot(...)` returns the handle to the axes of the plot.

`microplateplot(..., 'PropertyName', PropertyValue, ...)` calls `microplateplot` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each `PropertyName` in single quotation marks. Each `PropertyName` is case insensitive. These property name/property value pairs are as follows:

- `microplateplot(Data, ...'RowLabels', RowLabelsValue, ...)`: lets you specify labels for the rows of data.
- `microplateplot(Data, ...'ColumnLabels', ColumnLabelsValue, ...)`: lets you specify labels for the columns of data.
- `microplateplot(Data, ...'TextLabels', TextLabelsValue, ...)`: lets you specify text to overlay of the wells in the image.
- `microplateplot(Data, ...'TextFontSize', TextFontSizeValue, ...)`: lets you specify the font size of the text you specify with the 'TextLabels' property.
- `microplateplot(Data, ...'MissingValueColor', MissingValueColorValue, ...)`: lets you specify the color of wells with missing values (NaN values).
- `microplateplot(Data, ...'ToolTipFormat', ToolTipFormatValue, ...)`: lets you specify the format of the text used in the well tooltips. The well tooltips display the actual value from the input matrix when you click a well. `ToolTipFormatValue` is a format string, such as used by the `sprintf` function. Default is 'Value: %.3f', which specifies including three digits to the right of the decimal in fixed-point notation.
**Input Arguments**

**Data**

DataMatrix object on page 1-532 or matrix containing intensity values, such as from a plate reader.

**Tip** For help importing data from a spreadsheet or data file into a MATLAB matrix, see “Import Text Files” (MATLAB).

**Note** The microplateplot function converts any nonnumeric symbols or characters in the matrix to NaN values.

**Default:**

**RowLabelsValue**

Cell array of character vectors or string vector that specifies labels for the rows of data. Default is the first N letters of the alphabet, where N is the number of rows in Data. If there are more than 26 rows in Data, then the default is AA, AB, ..., ZZ. If Data is a DataMatrix object, then the default is the row labels of Data.

**Default:**

**ColumnLabelsValue**

Cell array of character vectors or string vector that specifies labels for the columns of data. Default is 1, 2, ..., M, where M is the number of columns in Data. If Data is a DataMatrix object, then the default is the column labels of Data.

**Default:**

**TextLabelsValue**

Cell array of character vectors or string vector the same size as Data that specifies text to overlay on the wells of the image.

**Default:**

**TextFontSizeValue**

Positive integer specifying the font size of the text you specify with the 'TextLabels' property. Default font size is determined automatically based on the size of the Figure window.

**Default:**

**MissingValueColorValue**

Three-element numeric vector of RGB values that specifies the color of wells with missing values (NaN values). Default is [0, 0, 0], which defines black.

**Default:**
ToolTipFormatValue

Format string, such as used by the sprintf function, that specifies the format of the text used in the well tooltips. The well tooltips display the actual value from the input matrix when you click a well.

Default: 'Value: %.3f', which specifies including three digits to the right of the decimal in fixed-point notation.

Output Arguments

Handle

Handle to the axes of the plot.

Tip Use the Handle output with the set function and the 'YDir' or 'XDir' property to reverse the order of the A through H labels or 1 through 12 labels respectively. Note that in the microplate plot, the default order for the A through H labels, or 'YDir' property, is 'reverse' (top to bottom), and the default order for the 1 through 12 labels, or 'XDir' property, is 'normal' (left to right). For more information on the 'XDir' and 'YDir' properties, see Axes.

Examples

Example 1.45. Creating a Plot of a Microplate, Changing the Colormap, Viewing Well Values, and Adding Text Labels

1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains two variables: assaydata, an 8-by-12 matrix of data values from a microtiter plate, and whiteToRed, a 64-by-3 matrix that defines a colormap.

load microPlateAssay

2 Create a visualization of the data from the microtiter plate.

microplateplot(assaydata)

3 Change the visualization to use a white-to-red colormap, and then view a tooltip displaying the value of well D6 by clicking the well.

colormap(whiteToRed)
Notice that all wells in column 12 are black, indicating missing data.

4 Overlay an X on well E8.
   a Create an empty cell array.
      \[
      \text{mask} = \text{cell}(8,12);
      \]
   b Add the string 'X' to the cell in the fifth row and eighth column of the array.
      \[
      \text{mask}\{5,8\} = 'X';
      \]
   c Pass the cell array to the microplateplot function using the 'TextLabels' property.
      \[
      \text{microplateplot(assaydata,'TEXTLABELS',mask)};
      \]
Example 1.46. Changing the Order of Row Labels in the Plot

1. If you have not already done so, create a plot of a microplate as described previously.
2. Return a handle to the axes of the plot, and then reverse the order of the row letter labels.

```matlab
h = microplateplot(assaydata);
set(h,'YDir','normal')
```
Example 1.47. Adding a Title and Axis Labels to the Plot

For information on adding a title and x-axis and y-axis labels to your plot, see “Add Title and Axis Labels to Chart” (MATLAB).

Example 1.48. Printing and Exporting the Plot

For information on printing or exporting your plot, see “Printing and Saving” (MATLAB).

See Also
imagesc | set | sprintf

Topics
“Printing and Saving” (MATLAB)

Introduced in R2009a
**min (DataMatrix)**

Return minimum values in DataMatrix object

**Syntax**

\[ M = \text{min}(\text{DMObj}1) \]

\[ [M, \text{Indices}] = \text{min}(\text{DMObj}1) \]

\[ [M, \text{Indices}, \text{Names}] = \text{min}(\text{DMObj}1) \]

... = \text{min}(\text{DMObj}1, [], \text{Dim})

\[ \text{MA} = \text{min}(\text{DMObj}1, \text{DMObj}2) \]

**Input Arguments**

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
</table>

**Note** DMObj1 and DMObj2 must be the same size, unless one is a scalar.

<table>
<thead>
<tr>
<th>Dim</th>
<th>Scalar specifying the dimension of DMObj to return the minimum values. Choices are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Default. Returns a row vector containing a minimum value for each column.</td>
</tr>
<tr>
<td>2</td>
<td>Returns a column vector containing a minimum value for each row.</td>
</tr>
</tbody>
</table>

**Output Arguments**

<table>
<thead>
<tr>
<th>M</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scalar specifying the minimum value in DMObj when it contains vector of data</td>
</tr>
<tr>
<td></td>
<td>Row vector containing the minimum value for each column in DMObj (when Dim = 1)</td>
</tr>
<tr>
<td></td>
<td>Column vector containing the minimum value for each row in DMObj (when Dim = 2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indices</th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive integer specifying the index of the minimum value in a DataMatrix object containing a vector of data</td>
</tr>
<tr>
<td></td>
<td>Vector containing the indices for the minimum value in each column (if Dim = 1) or row (if Dim = 2) in a DataMatrix object containing a matrix of data</td>
</tr>
</tbody>
</table>

| Names | Vector of the row names (if Dim = 1) or column names (if Dim = 2) corresponding to the minimum value in each column or each row of a DataMatrix object. |
Description

\( M = \min(\text{DMObj1}) \) returns the minimum value(s) in \( \text{DMObj1} \), a DataMatrix object. If \( \text{DMObj1} \) contains a vector of data, \( M \) is a scalar. If \( \text{DMObj1} \) contains a matrix of data, \( M \) is a row vector containing a minimum value in each column.

\[ [M, \text{Indices}] = \min(\text{DMObj1}) \]
returns \( \text{Indices} \), the indices of the minimum value(s) in \( \text{DMObj1} \), a DataMatrix object. If \( \text{DMObj1} \) contains a vector of data, \( \text{Indices} \) is a positive integer. If \( \text{DMObj1} \) contains a matrix of data, \( \text{Indices} \) is a vector containing the indices for the minimum value in each column (if \( \text{Dim} = 1 \)) or row (if \( \text{Dim} = 2 \)). If there are multiple minimum values in a column or row, the index for the first value is returned.

\[ [M, \text{Indices}, \text{Names}] = \min(\text{DMObj1}) \]
returns \( \text{Names} \), a vector of the row names (if \( \text{Dim} = 1 \)) or column names (if \( \text{Dim} = 2 \)) corresponding to the minimum value in each column or each row of \( \text{DMObj1} \), a DataMatrix object. If there is more than one minimum value in a column or row, the row or column name for the first value is returned.

\( \ldots = \min(\text{DMObj1}, [], \text{Dim}) \)
specifies which dimension to return the minimum values for, that is each column or each row in a DataMatrix object. If \( \text{Dim} = 1 \), returns \( M \), a row vector containing the minimum value in each column. If \( \text{Dim} = 2 \), returns \( M \), a column vector containing the minimum value in each row. Default \( \text{Dim} = 1 \).

\( M\text{A} = \min(\text{DMObj1}, \text{DMObj2}) \)
returns \( M\text{A} \), a numeric array containing the smaller of the two values from each position of \( \text{DMObj1} \) and \( \text{DMObj2} \). \( \text{DMObj1} \) and \( \text{DMObj2} \) can both be DataMatrix objects, or one can be a DataMatrix object and the other a numeric array. They must be the same size, unless one is a scalar. \( M\text{A} \) has the same size (number of rows and columns) as the first nonscalar input.

See Also
Datamatrix | max | sum

Topics
Datamatrix object on page 1-532

Introduced in R2008b
**minspantree (biograph)**

Find minimal spanning tree in biograph object

**Syntax**

```latex
[Tree, pred] = minspantree(BGObj)
[Tree, pred] = minspantree(BGObj, R)
[Tree, pred] = minspantree(..., 'Method', MethodValue, ...)
[Tree, pred] = minspantree(..., 'Weights', WeightsValue, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGObj</td>
<td>Biograph object created by <code>biograph</code> (object constructor).</td>
</tr>
<tr>
<td>R</td>
<td>Scalar between 1 and the number of nodes.</td>
</tr>
</tbody>
</table>

**Description**

**Tip** For introductory information on graph theory functions, see “Graph Theory Functions”.

[minspantree(BGobj)] finds an acyclic subset of edges that connects all the nodes in the undirected graph represented by an N-by-N adjacency matrix extracted from a biograph object, `BGobj`, and for which the total weight is minimized. Weights of the edges are all nonzero entries in the lower triangle of the N-by-N sparse matrix. Output `Tree` is a spanning tree represented by a sparse matrix. Output `pred` is a vector containing the predecessor nodes of the minimal spanning tree (MST), with the root node indicated by 0. The root node defaults to the first node in the largest connected component. This computation requires an extra call to the `graphconncomp` function.

**Note** The function ignores the direction of the edges in the Biograph object.

[minspantree(BGobj, R)] sets the root of the minimal spanning tree to node R.

[minspantree(..., 'PropertyName', PropertyValue, ...)] calls minspantree with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[minspantree(..., 'Method', MethodValue, ...)] lets you specify the algorithm used to find the minimal spanning tree (MST). Choices are:

- ‘Kruskal’ — Grows the minimal spanning tree (MST) one edge at a time by finding an edge that connects two trees in a spreading forest of growing MSTs. Time complexity is \(O(E+X*\log(N))\), where \(X\) is the number of edges no longer than the longest edge in the MST, and \(N\) and \(E\) are the number of nodes and edges respectively.
• 'Prim' — Default algorithm. Grows the minimal spanning tree (MST) one edge at a time by adding a minimal edge that connects a node in the growing MST with any other node. Time complexity is \(O(E \cdot \log(N))\), where \(N\) and \(E\) are the number of nodes and edges respectively.

**Note** When the graph is unconnected, Prim's algorithm returns only the tree that contains \(R\), while Kruskal's algorithm returns an MST for every component.

\[
\text{Tree, pred} = \text{minspantree}(..., \text{'Weights'}, \text{WeightsValue}, ...)\]

lets you specify custom weights for the edges. \textit{WeightsValue} is a column vector having one entry for every nonzero value (edge) in the \(N\)-by-\(N\) sparse matrix. The order of the custom weights in the vector must match the order of the nonzero values in the \(N\)-by-\(N\) sparse matrix when it is traversed column-wise. By default, \textit{minspantree} gets weight information from the nonzero entries in the \(N\)-by-\(N\) sparse matrix.

### References


### See Also

allshortestpaths | biograph | conncomp | graphminspantree | isdag | isomorphism | isspantree | maxflow | shortestpath | topoorder | traverse

### Topics

biograph object on page 1-185

**Introduced in R2006b**
minus (DataMatrix)

Subtract DataMatrix objects

Syntax

\[
\text{DMObjNew} = \text{minus}(\text{DMObj1}, \text{DMObj2}) \\
\text{DMObjNew} = \text{DMObj1} - \text{DMObj2} \\
\text{DMObjNew} = \text{minus}(\text{DMObj1}, B) \\
\text{DMObjNew} = \text{DMObj1} - B \\
\text{DMObjNew} = \text{minus}(B, \text{DMObj1}) \\
\text{DMObjNew} = B - \text{DMObj1}
\]

Input Arguments

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>MATLAB numeric or logical array.</td>
</tr>
</tbody>
</table>

Output Arguments

| DMObjNew   | DataMatrix object created by subtraction.                           |

Description

\[
\text{DMObjNew} = \text{minus}(\text{DMObj1}, \text{DMObj2}) \quad \text{or} \quad \text{DMObjNew} = \text{DMObj1} - \text{DMObj2}
\]

performs an element-by-element subtraction of the DataMatrix object \(\text{DMObj2}\) from the DataMatrix object \(\text{DMObj1}\) and places the results in \(\text{DMObjNew}\), another DataMatrix object. \(\text{DMObj1}\) and \(\text{DMObj2}\) must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for \(\text{DMObjNew}\) are the same as \(\text{DMObj1}\), unless \(\text{DMObj1}\) is a scalar; then they are the same as \(\text{DMObj2}\).

\[
\text{DMObjNew} = \text{minus}(\text{DMObj1}, B) \quad \text{or} \quad \text{DMObjNew} = \text{DMObj1} - B
\]

performs an element-by-element subtraction of \(B\), a numeric or logical array, from the DataMatrix object \(\text{DMObj1}\), and places the results in \(\text{DMObjNew}\), another DataMatrix object. \(\text{DMObj1}\) and \(B\) must have the same size (number of rows and columns), unless \(B\) is a scalar. The size (number of rows and columns), row names, and column names for \(\text{DMObjNew}\) are the same as \(\text{DMObj1}\).

\[
\text{DMObjNew} = \text{minus}(B, \text{DMObj1}) \quad \text{or} \quad \text{DMObjNew} = B - \text{DMObj1}
\]

performs an element-by-element subtraction of the DataMatrix object \(\text{DMObj1}\) from \(B\), a numeric or logical array, and places the results in \(\text{DMObjNew}\), another DataMatrix object. \(\text{DMObj1}\) and \(B\) must have the same size (number of rows and columns), unless \(B\) is a scalar. The size (number of rows and columns), row names, and column names for \(\text{DMObjNew}\) are the same as \(\text{DMObj1}\).

\[\text{Note}\quad \text{Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.}\]
MATLAB calls `DMObjNew = minus(X, Y)` for the syntax `DMObjNew = X - Y` when `X` or `Y` is a `DataMatrix` object.

**See Also**
`DataMatrix` | `plus`

**Topics**
`DataMatrix` object on page 1-532

**Introduced in R2008b**
molweight

Calculate molecular weight of amino acid sequence

Syntax

molweight(SeqAA)

Arguments

| SeqAA | Amino acid sequence. Enter a character vector, string, or a vector of integers from the tableAmino Acid Lookup. Examples: 'ARN', [1 2 3]. You can also enter a structure with the field Sequence. |

Description

molweight(SeqAA) calculates the molecular weight for the amino acid sequence SeqAA.

Examples

1. Retrieve an amino acid sequence from the NCBI GenPept database.
   
rhodopsin = getgenpept('NP_000530');

2. Calculate the molecular weight of the sequence.
   
rhodopsinMW = molweight(rhodopsin)
   
rhodopsinMW =
   
3.8892e+004

See Also

aacount|atomiccomp|isoelectric|isotopicdist|proteinplot

Introduced before R2006a
# molviewer

Display and manipulate 3-D molecule structure

## Syntax

```
molviewer
molviewer(File)
molviewer(pdbID)
molviewer(pdbStruct)
FigureHandle = molviewer(…)
```

## Input Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| `File` | Character vector or string specifying one of the following:  
- File name of a file on the MATLAB search path or in the MATLAB Current Folder  
- Path and file name  
- URL pointing to a file (URL must begin with a protocol such as http://, ftp://, or file://)  
  The referenced file is a molecule model file, such as a Protein Data Bank (PDB)-formatted file (ASCII text file). Valid file types include:  
  - PDB  
  - MOL (MDL)  
  - SDF  
  - XYZ  
  - SMOL  
  - JVXL  
  - CIF/mmCIF |
| `pdbID` | Character vector or string specifying a unique identifier for a protein structure record in the PDB database.  
  **Note**: Each structure in the PDB database is represented by a four-character alphanumeric identifier. For example, 4hhb is the identifier for hemoglobin. |
| `pdbStruct` | A structure containing a field for each PDB record, such as returned by the `getpdb` or `pdbread` function. |

## Output Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FigureHandle</code></td>
<td>Figure handle to the Molecule Viewer.</td>
</tr>
</tbody>
</table>
Description

`molviewer` opens the Molecule Viewer app. You can display 3-D molecular structures by selecting `File > Open`, `File > Load PDB ID`, or `File > Open URL`.

`molviewer(File)` reads the data in a molecule model file, `File`, and opens the Molecule Viewer app displaying the 3-D molecular structure for viewing and manipulation.

`molviewer(pdbID)` retrieves the structural data of a protein, `pdbID`, from the PDB database and opens the Molecule Viewer app displaying the 3-D molecular structure for viewing and manipulation.

`molviewer(pdbStruct)` reads the data from `pdbStruct`, a structure containing a field for each PDB record, and opens the Molecule Viewer app displaying a 3-D molecular structure for viewing and manipulation.

`FigureHandle = molviewer(...)` returns the figure handle to the Molecule Viewer window.

**Tip** You can pass the `FigureHandle` to the `evalrasmolscript` function, which sends RasMol script commands to the Molecule Viewer window.

**Tip** If you receive any errors related to memory or Java heap space, try increasing your Java heap space as described at https://www.mathworks.com/support/solutions/en/data/1-18I2C/.
After displaying the 3-D molecule structure, you can:

- Hover the mouse over a subcomponent of the molecule to display an identification label for it.
- Spin and rotate the molecule at different angles by click-dragging it.
- Spin the molecule in the x-z plane by clicking 🔄.
- Spin the molecule in the x-y plane by pressing and holding the **Shift** key, then click-dragging left and right.
- Zoom in a stepless fashion by pressing and holding the **Shift** key, then click-dragging up and down.
- Zoom in a stepwise fashion by clicking the figure, then turning the mouse scroll wheel, or by clicking the following buttons:

  📸 or 📷
• Move the molecule by pressing and holding Ctrl + Alt, then click-dragging.
• Change the background color between black and white by clicking
• Reset the molecule position by clicking
• Show or hide the Control Panel by clicking
• Manipulate and annotate the 3-D structure by selecting options in the Control Panel or, for a complete list of options, by right-clicking the Molecule Viewer window to select commands:

- Display the Jmol Script Console by clicking.
**Note** There is a known bug with the Open button of the script editor that prevents loading a Rasmol script interactively. Instead use the `evalrasmolscript` function which sends RasMol script commands to the Molecule Viewer app. Also, you can copy and paste the script commands into the script console.

**Examples**

View the acetylsalicylic acid (aspirin) molecule, whose structural information is contained in the Elsevier MDL molecule file aspirin.mol.

```
molviewer('aspirin.mol')
```

View the H5N1 influenza virus hemagglutinin molecule, whose structural information is located at `www.rcsb.org/pdb/files/2FK0.pdb.gz`.

```
molviewer('http://www.rcsb.org/pdb/files/2FK0.pdb.gz')
```

View the molecule with a PDB identifier of 2DHB.

```
molviewer('2DHB')
```

View the molecule with a PDB identifier of 4hhb, and create a figure handle for the Molecule Viewer.

```
FH = molviewer('4hhb')
```

Use the `getpdb` function to retrieve protein structure data from the PDB database and create a MATLAB structure. Then view the protein molecule.

```
pdbstruct = getpdb('1vqx')
molviewer(pdbstruct)
```

**See Also**

`evalrasmolscript` | `getpdb` | `pdbread` | `pdbsuperpose` | `pdbtransform` | `pdbwrite`
Introduced in R2007a
msalign

Align peaks in signal to reference peaks

Syntax

\[ \text{IntensitiesOut} = \text{msalign}(X, \text{Intensities}, \text{RefX}) \]

\[ ... = \text{msalign}(..., 'Rescaling', \text{RescalingValue}, ...) \]
\[ ... = \text{msalign}(..., 'Weights', \text{WeightsValue}, ...) \]
\[ ... = \text{msalign}(..., 'MaxShift', \text{MaxShiftValue}, ...) \]
\[ ... = \text{msalign}(..., 'WidthOfPulses', \text{WidthOfPulsesValue}, ...) \]
\[ ... = \text{msalign}(..., 'WindowSizeRatio', \text{WindowSizeRatioValue}, ...) \]
\[ ... = \text{msalign}(..., 'Iterations', \text{IterationsValue}, ...) \]
\[ ... = \text{msalign}(..., 'GridSteps', \text{GridStepsValue}, ...) \]
\[ ... = \text{msalign}(..., 'SearchSpace', \text{SearchSpaceValue}, ...) \]
\[ ... = \text{msalign}(..., 'ShowPlot', \text{ShowPlotValue}, ...) \]
\[ \text{[IntensitiesOut, RefXOut]} = \text{msalign}(..., 'Group', \text{GroupValue}, ...) \]

Input Arguments

<table>
<thead>
<tr>
<th>(X)</th>
<th>Vector of separation-unit values for a set of signals with peaks. The number of elements in the vector equals the number of rows in the matrix (\text{Intensities}). The separation unit can quantify wavelength, frequency, distance, time, or m/z depending on the instrument that generates the signal data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{Intensities})</td>
<td>Matrix of intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals the number of elements in vector (X).</td>
</tr>
<tr>
<td>(\text{RefX})</td>
<td>Vector of separation-unit values of known reference masses in a sample signal.</td>
</tr>
<tr>
<td>(\text{RescalingValue})</td>
<td>Controls the rescaling of (X). Choices are \text{true} (default) or \text{false}. When \text{false}, the output signal is aligned only to the reference peaks by using constant shifts. By default, \text{msalign} estimates a rescaling factor, unless (\text{RefX}) contains only one reference peak.</td>
</tr>
<tr>
<td>(\text{WeightsValue})</td>
<td>Vector of positive values, with the same number of elements as (\text{RefX}). The default vector is \text{ones(size(RefX))}.</td>
</tr>
</tbody>
</table>

Tip For reference peaks, select compounds that are not expected to have significant shifts among the different signals. For example, in mass spectrometry, select compounds that do not undergo structural transformation, such as phosphorylation. Doing so increases the accuracy of your alignment and lets you detect compounds that exhibit structural transformations among the sample signal.
<table>
<thead>
<tr>
<th><strong>MaxShiftValue</strong></th>
<th>Two-element vector, in which the first element is negative and the second element is positive, that specifies the lower and upper limits of a range, in separation units, relative to each peak. No peak shifts beyond these limits. Default is ([-100 \ 100]).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WidthOfPulsesValue</strong></td>
<td>Positive value that specifies the width, in separation units, for all the Gaussian pulses used to build the correlating synthetic signal. The point of the peak where the Gaussian pulse reaches 60.65% of its maximum is set to the width specified by <strong>WidthOfPulsesValue</strong>. Default is 10.</td>
</tr>
<tr>
<td><strong>WindowSizeRatioValue</strong></td>
<td>Positive value that specifies a scaling factor that determines the size of the window around every alignment peak. The synthetic signal is compared to the input signal only within these regions, which saves computation time. The size of the window is given in separation-units by <strong>WidthOfPulsesValue</strong> * <strong>WindowSizeRatioValue</strong>. Default is 2.5, which means at the limits of the window, the Gaussian pulses have a value of 4.39% of their maximum.</td>
</tr>
<tr>
<td><strong>IterationsValue</strong></td>
<td>Positive integer that specifies the number of refining iterations. At every iteration, the search grid is scaled down to improve the estimates. Default is 5.</td>
</tr>
<tr>
<td><strong>GridStepsValue</strong></td>
<td>Positive integer that specifies the number of steps for the search grid. At every iteration, the search area is divided by <strong>GridStepsValue</strong>^2. Default is 20.</td>
</tr>
<tr>
<td><strong>SearchSpaceValue</strong></td>
<td>Character vector or string that specifies the type of search space. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'regular' — Default. Evenly spaced lattice.</td>
</tr>
<tr>
<td></td>
<td>• 'latin' — Random Latin hypercube with <strong>GridStepsValue</strong>^2 samples.</td>
</tr>
<tr>
<td><strong>ShowPlotValue</strong></td>
<td>Controls the display of a plot of an original and aligned signal over the reference masses specified by <strong>RefX</strong>. Choices are <strong>true</strong>, <strong>false</strong>, or <strong>I</strong>, an integer specifying the index of a signal in <strong>Intensities</strong>. If you set to <strong>true</strong>, the first signal in <strong>Intensities</strong> is plotted. Default is:</td>
</tr>
<tr>
<td></td>
<td>• <strong>false</strong> — When return values are specified.</td>
</tr>
<tr>
<td></td>
<td>• <strong>true</strong> — When return values are not specified.</td>
</tr>
</tbody>
</table>
Controls the creation of RefXOut, a new vector of separation-unit values to be used as reference masses for aligning the peaks. This vector is created by adjusting the values in RefX, based on the sample data from multiple signals in Intensities, such that the overall shifting and scaling of the peaks is minimized. Choices are true or false (default).

**Tip** Set GroupValue to true only if Intensities contains data for a large number of signals, and you are not confident of the separation-unit values used for your reference peaks in RefX. Leave GroupValue set to false if you are confident of the separation-unit values used for your reference peaks in RefX.

### Output Arguments

<table>
<thead>
<tr>
<th>IntensitiesOut</th>
<th>Matrix of intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The intensity values represent a shifting and scaling of the data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RefXOut</td>
<td>Vector of separation-unit values of reference masses, calculated from RefX and the sample data from multiple signals in Intensities, when you set GroupValue to true.</td>
</tr>
</tbody>
</table>

### Description

**Tip** Use the following syntaxes with data from any separation technique that produces signal data, such as spectroscopy, NMR, electrophoresis, chromatography, or mass spectrometry.

\[
\text{IntensitiesOut} = \text{msalign} (X, \text{Intensities}, \text{RefX})
\]

aligns the peaks in raw, noisy signal data, represented by Intensities and X, to reference peaks, provided by RefX. First, it creates a synthetic signal from the reference peaks using Gaussian pulses centered at the separation-unit values specified by RefX. Then, it shifts and scales the separation-unit scale to find the maximum alignment between the input signals and the synthetic signal. (It uses an iterative multiresolution grid search until it finds the best scale and shift factors for each signal.) Once the new separation-unit scale is determined, the corrected signals are created by resampling their intensities at the original separation-unit values, creating IntensitiesOut, a vector or matrix of corrected intensity values. The resampling method preserves the shape of the peaks.

**Tip** The msalign function works best with three to five reference peaks that you know will appear in the signal. If you use a single reference peak (internal standard), there is a possibility of aligning sample peaks to the incorrect reference peaks as msalign both scales and shifts the X vector. If using a single reference peak, you might need to only shift the X vector. To do this, use

\[
\text{IntensitiesOut} = \text{interp1}(X, \text{Intensities}, X-(\text{ReferencePeak}-\text{ExperimentalPeak})).
\]

... = msalign(..., 'PropertyName', PropertyValue, ...) calls msalign with optional properties that use property name/property value pairs. You can specify one or more properties in
any order. EachPropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = msalign(..., 'Rescaling', RescalingValue, ...) controls the rescaling of X. Choices are true (default) or false. When false, the output signal is aligned only to the reference peaks by using constant shifts. By default, msalign estimates a rescaling factor, unless RefX contains only one reference peak.

... = msalign(..., 'Weights', WeightsValue, ...) specifies the relative weight for each mass in RefX, the vector of reference separation-unit values. WeightsValue is a vector of positive values, with the same number of elements as RefX. The default vector is ones(size(RefX)), which means each reference peak is weighted equally, so that more intense reference peaks have a greater effect in the alignment algorithm. If you have a less intense reference peak, you can increase its weight to emphasize it more in the alignment algorithm.

... = msalign(..., 'MaxShift', MaxShiftValue, ...) specifies the lower and upper limits of the range, in separation units, relative to each peak. No peak shifts beyond these limits. MaxShiftValue is a two-element vector, in which the first element is negative and the second element is positive. Default is [-100 100].

**Note** Use these values to tune the robustness of the algorithm. Ideally, you should keep the range within the maximum expected shift. If you try to correct larger shifts by increasing the limits, you increase the possibility of picking incorrect peaks to align to the reference masses.

... = msalign(..., 'WidthOfPulses', WidthOfPulsesValue, ...) specifies the width, in separation units, for all the Gaussian pulses used to build the correlating synthetic signal. The point of the peak where the Gaussian pulse reaches 60.65% of its maximum is set to the width you specify with WidthOfPulsesValue. Choices are any positive value. Default is 10. WidthOfPulsesValue may also be a function handle. The function is evaluated at the respective separation-unit values and returns a variable width for the pulses. Its evaluation should give reasonable values from 0 to max(abs(Range)); otherwise, the function returns an error.

**Note** Tuning the spread of the Gaussian pulses controls a tradeoff between robustness (wider pulses) and precision (narrower pulses). However, the spread of the pulses is unrelated to the shape of the observed peaks in the signal. The purpose of the pulse spread is to drive the optimization algorithm.

... = msalign(..., 'WindowSizeRatio', WindowSizeRatioValue, ...) specifies a scaling factor that determines the size of the window around every alignment peak. The synthetic signal is compared to the sample signal only within these regions, which saves computation time. The size of the window is given in separation units by WidthOfPulsesValue * WindowSizeRatioValue. Choices are any positive value. Default is 2.5, which means at the limits of the window, the Gaussian pulses have a value of 4.39% of their maximum.

... = msalign(..., 'Iterations', IterationsValue, ...) specifies the number of refining iterations. At every iteration, the search grid is scaled down to improve the estimates. Choices are any positive integer. Default is 5.

... = msalign(..., 'GridSteps', GridStepsValue, ...) specifies the number of steps for the search grid. At every iteration, the search area is divided by GridStepsValue^2. Choices are any positive integer. Default is 20.
... = msalign(..., 'SearchSpace', SearchSpaceValue, ...) specifies the type of search space. Choices are:

- 'regular' — Default. Evenly spaced lattice.
- 'latin' — Random Latin hypercube with GridStepsValue^2 samples.

... = msalign(..., 'ShowPlot', ShowPlotValue, ...) controls the display of a plot of an original and aligned signal over the reference masses specified by RefX. Choices are true, false, or I, an integer specifying the index of a signal in Intensities. If set to true, the first signal in Intensities is plotted. Default is:

- false — When return values are specified.
- true — When return values are not specified.

[IntensitiesOut, RefXOut] = msalign(..., 'Group', GroupValue, ...) controls the creation of RefXOut, a new vector of separation-unit values to use as reference masses for aligning the peaks. This vector is created by adjusting the values in RefX, based on the sample data from multiple signals in Intensities, such that the overall shifting and scaling of the peaks is minimized. Choices are true or false (default).

**Tip** Set GroupValue to true only if Intensities contains data for a large number of signals, and you are not confident of the separation-unit values used for your reference peaks in RefX. Leave GroupValue set to false if you are confident of the separation-unit values used for your reference peaks in RefX.

**Examples**

**Example 1.49. Aligning a Mass Spectrum with Three or More Reference Peaks**

1. Load a MAT-file, included with the Bioinformatics Toolbox software, that contains sample data, reference masses, and parameter data for synthetic peak width.

   load sample_lo_res
   R = [3991.4 4598 7964 9160];
   W = [60 100 60 100];

2. Display a color image of the mass spectra before alignment.

   msh heatmap(MZ_lo_res,Y_lo_res,'markers',R,'range',[3000 10000])
   title('before alignment')
Align spectra with reference masses and display a color image of mass spectra after alignment.

\[ YA = \text{msalign}(MZ\_lo\_res,Y\_lo\_res,R,'weights',W); \]
\[ \text{msheatmap}(MZ\_lo\_res,YA,'markers',R,'range',[3000 10000]) \]
\[ \text{title}('after\_alignment') \]
Example 1.50. Aligning a Mass Spectrum with One Reference Peak

It is not recommended to use the `msalign` function if you have only one reference peak. Instead, use the following procedure, which shifts the X input vector, but does not scale it.

1. Load sample data and view the first sample spectrum.

```matlab
load sample_lo_res
MZ = MZ_lo_res;
Y = Y_lo_res(:,1);
msviewer(MZ, Y)
```
2 Use the tall peak around 4000 m/z as the reference peak. To determine the reference peak's m/z value, click and then click-drag to zoom in on the peak. Right-click in the center of the peak, and then click Add Marker to label the peak with its m/z value.
3 Shift a spectrum by the difference between RP, the known reference mass of 4000 m/z, and SP, the experimental mass of 4051.14 m/z.

\[
\begin{align*}
RP &= 4000; \\
SP &= 4051.14; \\
YOut &= \text{interp1}(MZ, Y, MZ-(RP-SP));
\end{align*}
\]

4 Plot the original spectrum in red and the shifted spectrum in blue and zoom in on the reference peak.

```matlab
plot(MZ,Y,'r',MZ,YOut,'b:');
xlabel('Mass/Charge (M/Z)');
ylabel('Relative Intensity');
legend('Y','YOut');
axis([3600 4800 -2 60])
```

References


See Also

msalign | msbackadj | msdotplot | msheatmap | mslowess | msnorm | mspalign | mspeaks | msppresample | msresample | mssgolay | msviewer
Topics
“Mass Spectrometry and Bioanalytics”
“Preprocessing Raw Mass Spectrometry Data”
“Visualizing and Preprocessing Hyphenated Mass Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling”
“Differential Analysis of Complex Protein and Metabolite Mixtures using Liquid Chromatography/Mass Spectrometry (LC/MS)”

Introduced before R2006a
**msbackadj**

Correct baseline of signal with peaks

**Syntax**

\[
yOut = msbackadj(X,\text{Intensities})
\]

\[
yOut = msbackadj(X,\text{Intensities},\text{Name},\text{Value})
\]

**Description**

\(yOut = msbackadj(X,\text{Intensities})\) adjusts the variable baseline of a raw signal with peaks by performing the following steps.

1. Estimate the baseline within multiple shifted windows of width 200 separation units.
2. Regress the varying baseline to the window points using a spline approximation.
3. Adjust the baseline of the peak signals supplied by the input \text{Intensities}.
4. Return the adjusted intensity values in the output matrix \(yOut\).

\(yOut = msbackadj(X,\text{Intensities},\text{Name},\text{Value})\) sets additional options specified by one or more name-value pair arguments. For example, \(msbackadj(X,\text{Intensities},'\text{WindowSize}',300)\) sets the width of the shifting window to 300 separation units.

**Examples**

**Adjust Baseline of Mass Spectrometry Data**

Load a sample mass spec data including \text{MZ\_lo\_res}, a vector of m/z values, and \text{Y\_lo\_res}, a matrix of intensity values.

\[
\text{load sample\_lo\_res}
\]

Adjust the baseline of a group of spectrograms and show only the third spectrum and its estimated background.

\[
YB = msbackadj(MZ\_lo\_res,Y\_lo\_res,'\text{ShowPlot}',3);\]
Estimate the baseline for every spectrum in \( Y_{\text{lo res}} \) using an anonymous function to describe an m/z dependent parameter. Then plot the estimated background for the fourth spectrum.

\[
w_f = @(mz) 200 + 0.001 \cdot mz;
\]

\[
\text{msbackadj}(MZ_{\text{lo res}}, Y_{\text{lo res}}, '\text{StepSize}', w_f, '\text{ShowPlot}', 4);
\]
**Input Arguments**

**X — Separation-unit values**  
vector

Separation-unit values for a set of signals with peaks, specified as a vector.

The number of elements in the vector equals the number of rows in *Intensities*. The separation unit can quantify wavelength, frequency, distance, time, or m/z ratio depending on the instrument that generates the signal data.

Data Types: double

**Intensities — Intensity values for set of peaks**  
numeric matrix

Intensity values for a set of peaks that share separation-unit range, specified as a numeric matrix.

Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals the number of elements in *X*. The signal data can come from any separation technique, such as spectroscopy, NMR, electrophoresis, chromatography, or mass spectrometry.

Data Types: double
**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example:

**WindowSize — Shifting window size**

200 (default) | positive scalar | function handle

Shifting window size, specified as a positive scalar or function handle. By default, msbackadj estimates baseline points for windows with a width of 200 separation units.

If you specify a function handle, the function is evaluated at the respective X values and returns a variable width for the window. Specifying a function handle is useful when the resolution of the signal is dissimilar at different regions.

The result of msbackadj depends on the window size and step size. Define the parameters based on the width of your peaks in the signal and the presence of possible drifts. If you have wider peaks towards the end of the signal, consider using variable window sizes and/or step sizes.

Example: 'WindowSize',300
Data Types: double | function_handle

**StepSize — Step size for shifting window**

200 (default) | positive scalar | function handle

Step size for the shifting window, specified as a positive scalar or function handle. By default, msbackadj estimates baseline points for windows placed every 200 separation units.

If you specify a function handle, the function is evaluated at the respective separation-unit values and returns the distance between adjacent windows.

Example: 'StepSize',150
Data Types: double | function_handle

**RegressionMethod — Method to regress window estimated points**

'pchip' (default) | 'linear' | 'spline'

Method to regress the window estimated points to a soft curve, specified as one of the following:

- 'pchip' — Shape-preserving piecewise cubic interpolation. The interpolated value at a query point is based on a shape-preserving piecewise cubic interpolation of the values at neighboring grid points.
- 'linear' — Linear interpolation. The interpolated value at a query point is based on linear interpolation of the values at neighboring grid points in each respective dimension.
- 'spline' — Spline interpolation. The interpolated value at a query point is based on a cubic interpolation of the values at neighboring grid points in each respective dimension.

Example: 'RegressionMethod','linear'
Data Types: char | string
EstimationMethod — Method to find likely baseline value
'vequantile' (default) | 'em'

Method to find likely baseline (background) value in every window, specified as one of the following:

- 'quantile' — Quantile value is set to 10%.
- 'em' — Every sample is the independent and identically distributed (i.i.d) draw of any of two normal distributed classes (background or peaks). Because the class label is hidden, the distributions are estimated with an Expectation-Maximization algorithm. The ultimate baseline value is the mean of the background class.

Example: 'EstimationMethod','em'
Data Types: char | string

SmoothMethod — Method to smooth curve of estimated points
'venone' (default) | 'lowess' | 'loess' | 'rlowess' | 'rloess'

Method to smooth the curve of estimated points, specified as one of the following:

- 'none' — No smoothing.
- 'lowess' — Linear fit.
- 'loess' — Quadratic fit.
- 'rlowess' — Robust linear fit.
- 'rloess' — Robust quadratic fit.

Example: 'SmoothMethod','lowess'
Data Types: char | string

QuantileValue — Quantile value
0.10 (default) | positive scalar between 0 and 1

Quantile value, specified as a positive scalar between 0 and 1.

Example: 'QuantileValue',0.2
Data Types: double

PreserveHeights — Flag to preserve height of tallest peak
false (default) | true

Flag to preserve the height of the tallest peak in the signal, specified as true or false. By default, peak heights are not preserved.

Example: 'PreserveHeights',true
Data Types: logical

ShowPlot — Flag to plot regressed baseline, original signal, and estimated baseline points
false | true | positive integer

Flag to plot the regressed baseline, original signal, and estimated baseline points, specified as true, false, or a positive integer.

The default behavior is as follows:
• When you call `msbackadj` without an output argument, the plot is shown. Only the first signal from the input `Intensities` is plotted.
• When you call `msbackadj` with an output argument, the plot is not shown. But you can get the plot by also setting `'ShowPlot'` to true.

You can also specify an index to one of the signals (columns) in `Intensities` to show the corresponding plot of that signal.

Example: `'ShowPlot',5`

Data Types: `double` | `logical`

**Output Arguments**

`yOut` — Adjusted intensity values matrix

Adjusted intensity values, returned as a matrix.

**See Also**

`msalign`, `msbackadj`, `msdotplot`, `msheatmap`, `mslowess`, `msnorm`, `mspalign`, `mspeaks`, `msppresample`, `msresample`, `mssgolay`, `msviewer`

**Topics**

“Mass Spectrometry and Bioanalytics”
“Preprocessing Raw Mass Spectrometry Data”
“Visualizing and Preprocessing Hyphenated Mass Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling”
“Differential Analysis of Complex Protein and Metabolite Mixtures using Liquid Chromatography/Mass Spectrometry (LC/MS)”

**Introduced before R2006a**
msdotplot

Plot set of peak lists from LC/MS or GC/MS data set

Syntax

msdotplot(Peaklist, Times)
msdotplot(FigHandle, Peaklist, Times)
msdotplot(..., 'Quantile', QuantileValue)
PlotHandle = msdotplot(...)

Input Arguments

<table>
<thead>
<tr>
<th>Peaklist</th>
<th>Cell array of peak lists, where each element is a two-column matrix with m/z values in the first column and ion intensity values in the second column. Each element corresponds to a spectrum or retention time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tip</td>
<td>You can use the mzxml2peaks function to create the Peaklist cell array.</td>
</tr>
<tr>
<td>Times</td>
<td>Vector of retention times associated with an LC/MS or GC/MS data set. The number of elements in Times equals the number of elements in the cell array Peaklist.</td>
</tr>
<tr>
<td>Tip</td>
<td>You can use the mzxml2peaks function to create the Times vector.</td>
</tr>
<tr>
<td>FigHandle</td>
<td>Handle to an open Figure window such as one created by the msheatmap function.</td>
</tr>
<tr>
<td>QuantileValue</td>
<td>Value that specifies a percentage. When peaks are ranked by intensity, only those that rank above this percentage are plotted. Choices are any value ≥ 0 and ≤ 1. Default is 0. For example, setting QuantileValue = 0 plots all peaks, and setting QuantileValue = 0.8 plots only the 20% most intense peaks.</td>
</tr>
</tbody>
</table>

Output Arguments

| PlotHandle     | Handle to the line series object (figure plot). |

Description

msdotplot(Peaklist, Times) plots a set of peak lists from a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set represented by Peaklist, a cell array of peak lists, where each element is a two-column matrix with m/z values in the first column and ion intensity values in the second column, and Times, a vector of retention times associated with the spectra. Peaklist and Times have the same number of elements. The data is plotted into any existing figure generated by the msheatmap function; otherwise, the data is plotted into a new Figure window.
msdotplot(FigHandle, Peaklist, Times) plots the set of peak lists into the axes contained in an open Figure window with the handle FigHandle.

**Tip** This syntax is useful to overlay a dot plot on top of a heat map of mass spectrometry data created with the msheatmap function.

msdotplot(..., 'Quantile', QuantileValue) plots only the most intense peaks, specifically those in the percentage above the specified QuantileValue. Choices are any value \( \geq 0 \) and \( \leq 1 \). Default is 0. For example, setting QuantileValue = 0 plots all peaks, and setting QuantileValue = 0.8 plots only the 20% most intense peaks.

PlotHandle = msdotplot(...) returns a handle to the line series object (figure plot). You can use this handle as input to the get function to display a list of the plot's properties. You can use this handle as input to the set function to change the plot's properties, including showing and hiding points.

**Examples**

1. Load a MAT-file, included with the Bioinformatics Toolbox software, which contains LC/MS data variables, including peaks and ret_time. peaks is a cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time. ret_time is a column vector of retention times associated with the LC/MS data set.

   ```matlab
   load lcmsdata
   ```

2. Create a dot plot with only the 5% most intense peaks.

   ```matlab
   msdotplot(ms_peaks,ret_time,'Quantile',0.95)
   ```
3 Resample the data, then create a heat map of the LC/MS data.

\[
[MZ,Y] = 
\text{msppresample}(ms\_peaks,5000);
\text{msheatmap}(MZ,\text{ret\_time},\log(Y))
\]
Overlay the dot plot on the heat map, and then zoom in to see the detail.

```matlab
msdotplot(ms_peaks, ret_time)
axis([480 532 375 485])
```
See Also
msalign | msbackadj | msdotplot | msheatmap | mslowess | msnorm | mspalign | mspeaks | msppresample | msresample | mssgolay | msviewer

Topics
“Mass Spectrometry and Bioanalytics”
“Preprocessing Raw Mass Spectrometry Data”
“Visualizing and Preprocessing Hyphenated Mass Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling”
“Differential Analysis of Complex Protein and Metabolite Mixtures using Liquid Chromatography/Mass Spectrometry (LC/MS)”

Introduced in R2007a
**msheatmap**

Create pseudocolor image of set of mass spectra

**Syntax**

```matlab
msheatmap(MZ, Intensities)
msheatmap(MZ, Times, Intensities)
msheatmap(..., 'Midpoint', MidpointValue, ...)
msheatmap(..., 'Range', RangeValue, ...)
msheatmap(..., 'Markers', MarkersValue, ...)
msheatmap(..., 'SpecIdx', SpecIdxValue, ...)
msheatmap(..., 'Group', GroupValue, ...)
msheatmap(..., 'Resolution', ResolutionValue, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MZ</strong></td>
<td>Column vector of common mass/charge (m/z) values for a set of spectra. The number of elements in the vector equals the number of rows in the matrix <strong>Intensities</strong>.</td>
</tr>
<tr>
<td><strong>Times</strong></td>
<td>Column vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. The number of elements in the vector equals the number of columns in the matrix <strong>Intensities</strong>. The retention times are used to label the y-axis of the heat map.</td>
</tr>
<tr>
<td><strong>Intensities</strong></td>
<td>Matrix of intensity values for a set of mass spectra that share the same m/z range. Each row corresponds to an m/z value, and each column corresponds to a spectrum or retention time. The number of rows equals the number of elements in vector <strong>MZ</strong>. The number of columns equals the number of elements in vector <strong>Times</strong>.</td>
</tr>
</tbody>
</table>

**Note** You can use the `msppresample` function to create the **MZ** vector.

**Tip** You can use the `mzxml2peaks` function to create the **Times** vector.

**Note** You can use the `msppresample` function to create the **Intensities** matrix.
**MidpointValue**

Value specifying a quantile of the ion intensity values to fall below the midpoint of the colormap, meaning they do not represent peaks. *

msheatmap* uses a custom colormap where cool colors represent nonpeak regions, white represents the midpoint, and warm colors represent peaks. Choices are any value \( \geq 0 \) and \( \leq 1 \). Default is:

- 0.99 — For LC/MS or GC/MS data or when input \( T \) is provided. This means that 1% of the pixels are warm colors and represent peaks.
- 0.95 — For non-LC/MS or non-GC/MS data or when input \( T \) is not provided. This means that 5% of the pixels are warm colors and represent peaks.

**Tip** You can also change the midpoint interactively after creating the heat map by right-clicking the color bar, selecting **Interactive Colormap Shift**, and then click-dragging the cursor vertically on the color bar. This technique is useful when comparing multiple heat maps.

**RangeValue**

1-by-2 vector specifying the m/z range for the x-axis of the heat map. *RangeValue* must be within \([\text{min}(MZ) \ \text{max}(MZ)]\). Default is the full range \([\text{min}(MZ) \ \text{max}(MZ)]\).

**MarkersValue**

Vector of m/z values to mark on the top horizontal axis of the heat map. Default is [ ].

**SpecIdxValue**

either of the following:

- Vector of values with the same number of elements as columns (spectra) in the matrix *Intensities*.

- Cell array of character vectors or string vector with the same number of elements as columns (spectra) in the matrix *Intensities*.

Each value or character vector or string specifies a label for the corresponding spectrum. These values or character vectors or strings are used to label the y-axis of the heat map.

**Note** If input *Times* is provided, it is assumed that *Intensities* contains LC/MS or GC/MS data, and *SpecIdxValue* is ignored.

**GroupValue**

Either of the following:

- Vector of values with the same number of elements as rows in the matrix *Intensities*

- Cell array of character vectors or string vector with the same number of elements as rows (spectra) in the matrix *Intensities*

Each value, character vector, or string specifies a group to which the corresponding spectrum belongs. The spectra are sorted and combined into groups along the y-axis in the heat map.

**Note** If input *Times* is provided, it is assumed that *Intensities* contains LC/MS or GC/MS data, and *GroupValue* is ignored.
**ResolutionValue**

Value specifying the horizontal resolution of the heat map image. Increase this value to enhance details. Decrease this value to reduce memory usage. Default is:

- **0.5** — When \( MZ \) contains > 2,500 elements.
- **0.05** — When \( MZ \) contains \( \leq 2,500 \) elements.

### Description

\( \text{msheatmap}(MZ, \text{Intensities}) \) displays a pseudocolor heat map image of the intensities for the spectra in matrix \( \text{Intensities} \).

\( \text{msheatmap}(MZ, \text{Times}, \text{Intensities}) \) displays a pseudocolor heat map image of the intensities for the spectra in matrix \( \text{Intensities} \), using the retention times in vector \( \text{Times} \) to label the \( y \)-axis.

\( \text{msheatmap}(..., '\text{PropertyName}', \text{PropertyValue}, ...) \) calls \( \text{msheatmap} \) with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each \( \text{PropertyName} \) must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

- \( \text{msheatmap}(..., '\text{Midpoint}', \text{MidpointValue}, ...) \) specifies a quantile of the ion intensity values to fall below the midpoint of the colormap, meaning they do not represent peaks. \( \text{msheatmap} \) uses a custom colormap where cool colors represent nonpeak regions, white represents the midpoint, and warm colors represent peaks. Choices are any value between 0 and 1. Default is:
  - **0.99** — For LC/MS or GC/MS data or when input \( T \) is provided. This means that 1% of the pixels are warm colors and represent peaks.
  - **0.95** — For non-LC/MS or non-GC/MS data or when input \( T \) is not provided. This means that 5% of the pixels are warm colors and represent peaks.

**Tip** You can also change the midpoint interactively after creating the heat map by right-clicking the color bar, selecting Interactive Colormap Shift, then click-dragging the cursor vertically on the color bar. This technique is useful when comparing multiple heat maps.

- \( \text{msheatmap}(..., '\text{Range}', \text{RangeValue}, ...) \) specifies the m/z range for the \( x \)-axis of the heat map. \( \text{RangeValue} \) is a 1-by-2 vector that must be within \([\min(MZ) \ max(MZ)]\). Default is the full range \([\min(MZ) \ max(MZ)]\).

- \( \text{msheatmap}(..., '\text{Markers}', \text{MarkersValue}, ...) \) places markers along the top horizontal axis of the heat map for the m/z values specified in the vector \( \text{MarkersValue} \). Default is \([\]\).

- \( \text{msheatmap}(..., '\text{SpecIdx}', \text{SpecIdxValue}, ...) \) labels the spectra along the \( y \)-axis in the heat map. The labels are specified by \( \text{SpecIdxValue} \), a vector of values, cell array of character vectors, or string vector. The number of values or character vectors or strings is the same as the number of columns (spectra) in the matrix \( \text{Intensities} \). Each value or character vector or string specifies a label for the corresponding spectrum.

- \( \text{msheatmap}(..., '\text{Group}', \text{GroupValue}, ...) \) sorts and combines spectra into groups along the \( y \)-axis in the heat map. The groups are specified by \( \text{GroupValue} \), a vector of values, cell array of character vectors, or string vector. The number of values, character vectors, or strings is the same as
the number of rows in the matrix \textit{Intensities}. Each value or character vector or string specifies a group to which the corresponding spectrum belongs. Default is \([1:\text{numSpectra}].\)

\texttt{msheatmap(..., 'Resolution', \textit{ResolutionValue}, ...)} specifies the horizontal resolution of the heat map image. Increase this value to enhance details. Decrease this value to reduce memory usage. Default is:

- \(0.5\) — When \textit{MZ} contains \(> 2,500\) elements.
- \(0.05\) — When \textit{MZ} contains \(\leq 2,500\) elements.

\textbf{Examples}

\textbf{Example 1.51. SELDI-TOF Data}

1. Load SELDI-TOF sample data.

   \texttt{load sample_lo_res}

2. Create a vector of four \(m/z\) values to mark along the top horizontal axis of the heat map.

   \(M = [3991.4\ 4598\ 7964\ 9160];\)

3. Display the heat map with \(m/z\) markers and a limited \(m/z\) range.

   \texttt{msheatmap(MZ_lo_res,Y_lo_res,'markers',M,'range',[3000 10000])}

4. Display the heat map again grouping each spectrum into one of two groups.

   \texttt{TwoGroups = [1 1 2 2 1 1 2 2];}

   \texttt{msheatmap(MZ_lo_res,Y_lo_res,'markers',M,'group',TwoGroups)}
Example 1.52. Liquid Chromatography/Mass Spectrometry (LC/MS) Data

1. Load LC/MS sample data.
   ```matlab
   load lcmsdata
   ```

2. Resample the peak lists to create a vector of m/z values and a matrix of intensity values.
   ```matlab
   [MZ, Intensities] = msppresample(ms_peaks, 5000);
   ```

3. Display the heat map showing mass spectra at different retention times.
   ```matlab
   msheatmap(MZ, ret_time, log(Intensities))
   ```
See Also
msalign | msbackadj | msdotplot | msheatmap | mslowess | msnorm | msalign | mspeaks |
msppresample | msresample | mssgolay | msvviewer

Topics
“Mass Spectrometry and Bioanalytics”
“Preprocessing Raw Mass Spectrometry Data”
“Visualizing and Preprocessing Hyphenated Mass Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling”
“Differential Analysis of Complex Protein and Metabolite Mixtures using Liquid Chromatography/Mass Spectrometry (LC/MS)”

Introduced before R2006a
mslowess

Smooth signal with peaks using nonparametric method

Syntax

\[ Y_{out} = \text{mslowess}(X, \text{Intensities}) \]

\[ \text{mslowess}(..., \text{'Order'}, \text{OrderValue}, ...) \]
\[ \text{mslowess}(..., \text{'Span'}, \text{SpanValue}, ...) \]
\[ \text{mslowess}(..., \text{'Kernel'}, \text{KernelValue}, ...) \]
\[ \text{mslowess}(..., \text{'RobustIterations'}, \text{RobustIterationsValue}, ...) \]
\[ \text{mslowess}(..., \text{'ShowPlot'}, \text{ShowPlotValue}, ...) \]

Arguments

<table>
<thead>
<tr>
<th>( X )</th>
<th>Vector of separation-unit values for a set of signals with peaks. The number of elements in the vector equals the number of rows in the matrix ( \text{Intensities} ). The separation unit can quantify wavelength, frequency, distance, time, or m/z depending on the instrument that generates the signal data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Intensities} )</td>
<td>Matrix of intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals the number of elements in vector ( X ).</td>
</tr>
</tbody>
</table>

Description

Tip Use the following syntaxes with data from any separation technique that produces signal data, such as spectroscopy, NMR, electrophoresis, chromatography, or mass spectrometry.

\[ Y_{out} = \text{mslowess}(X, \text{Intensities}) \] smooths raw noisy signal data, \( \text{Intensities} \), using a locally weighted linear regression (Lowess) method with a default span of 10 samples.

Note mslowess assumes the input vector, \( X \), may not have uniformly spaced separation units. Therefore, the sliding window for smoothing is centered using the closest samples in terms of the \( X \) value and not in terms of the \( X \) index.

Note When the input vector, \( X \), does not have repeated values or NaN values, the algorithm is approximately twice as fast.

\[ \text{mslowess}(X, \text{Intensities}, ..., \text{'PropertyName'}, \text{PropertyValue}, ...) \] calls \text{mslowess} with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each \text{PropertyName} must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:
mslowess(..., 'Order', OrderValue, ...) specifies the order (OrderValue) of the Lowess smoother. Enter 1 (linear polynomial fit or Lowess), 2 (quadratic polynomial fit or Loess), or 0 (equivalent to a weighted local mean estimator and presumably faster because only a mean computation is performed instead of a least-squares regression). The default value is 1.

**Note** Curve Fitting Toolbox software also refers to Lowess smoothing of order 2 as Loess smoothing.

mslowess(..., 'Span', SpanValue, ...) specifies the window size for the smoothing kernel. If SpanValue is greater than 1, the window is equal to SpanValue number of samples independent of the separation-unit vector, X. The default value is 10 samples. Higher values will smooth the signal more at the expense of computation time. If SpanValue is less than 1, the window size is taken to be a fraction of the number of points in the data. For example, when SpanValue is 0.005, the window size is equal to 0.50% of the number of points in X.

mslowess(..., 'Kernel', KernelValue, ...) selects the function specified by KernelValue for weighting the observed intensities. Samples close to the separation-unit location being smoothed have the most weight in determining the estimate. KernelValue can be any of the following character vectors (or strings):

- 'tricubic' (default) — \((1 - (\text{dist}/\text{dmax}).^3).^3\)
- 'gaussian' — \(\exp(-2*(\text{dist}/\text{dmax}).^2)\)
- 'linear' — \(1-\text{dist}/\text{dmax}\)

mslowess(..., 'RobustIterations', RobustIterationsValue, ...) specifies the number of iterations (RobustValue) for a robust fit. If RobustIterationsValue is 0 (default), no robust fit is performed. For robust smoothing, small residual values at every span are outweighed to improve the new estimate. 1 or 2 robust iterations are usually adequate, while larger values might be computationally expensive.

**Note** For an X vector that has uniformly spaced separation units, a nonrobust smoothing with OrderValue equal to 0 is equivalent to filtering the signal with the kernel vector.

mslowess(..., 'ShowPlot', ShowPlotValue, ...) plots the smoothed signal over the original signal. When you call mslowess without output arguments, the signals are plotted unless ShowPlotValue is false. When ShowPlotValue is true, only the first signal in Intensities is plotted. ShowPlotValue can also contain an index to one of the signals in Intensities.

**Examples**

1. Load a MAT-file, included with the Bioinformatics Toolbox software, that contains some sample data.

   ```matlab
   load sample_lo_res
   ```

2. Smooth the spectra and draw a figure of the first spectrum with original and smoothed signals.

   ```matlab
   YS = mslowess(MZ_lo_res,Y_lo_res,'Showplot',true);
   ```
Zoom in on a region of the figure to see the difference in the original and smoothed signals.

\[ \text{axis}([7350 \ 7550 \ 0.1 \ 1.0]) \]

**See Also**

msalign | msbackadj | msdotplot | msheatmap | mslowess | msnorm | mspalign | mspeaks | msppresample | msresample | mssgolay | msviewer
Topics

“Mass Spectrometry and Bioanalytics”
“Preprocessing Raw Mass Spectrometry Data”
“Visualizing and Preprocessing Hyphenated Mass Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling”
“Differential Analysis of Complex Protein and Metabolite Mixtures using Liquid Chromatography/Mass Spectrometry (LC/MS)”

Introduced before R2006a
msnorm

Normalize set of signals with peaks

**Syntax**

\[
yOut = \text{msnorm}(X, \text{Intensities})
\]

\[
[yOut, \text{normParams}] = \text{msnorm}(X, \text{Intensities})
\]

\[
yOut = \text{msnorm}(X, \text{Intensities}, \text{NormParameters})
\]

\[
[\_\_\_] = \text{msnorm}(X, \text{Intensities}, \text{Name,Value})
\]

**Description**

\[
yOut = \text{msnorm}(X, \text{Intensities})\]

normalizes a group of signals with peaks by standardizing the area under the curve (AUC) to the group median and returns the normalized data \(yOut\).

\[
[yOut, \text{normParams}] = \text{msnorm}(X, \text{Intensities})\]

also returns the normalization parameters \(\text{normParams}\), which you can use to normalize another group of signals.

\[
yOut = \text{msnorm}(X, \text{Intensities}, \text{NormParameters})\]

uses the parameter information \(\text{NormParameters}\) from a previous normalization to normalize a new set of signals. The function uses the same parameters to select the separation-unit positions and output scale from the previous normalization. If you specified a consensus proportion using the 'Consensus' name-value pair argument in the previous normalization, the function selects no new separation-unit positions and performs normalization using the same separation-unit positions.

\[
[\_\_\_] = \text{msnorm}(X, \text{Intensities}, \text{Name,Value})\]

uses additional options specified by one or more name-value pair arguments and returns any of the output arguments in previous syntaxes. For example, \(\text{out} = \text{msnorm}(X,Y,'\text{Quantile}',[0.9 1])\) sets the lower (0.9) and upper (1) quantile limit to use only the largest 10% of intensities in each signal to compute the AUC.

**Examples**

**AUC Normalization**

This example shows how to normalize the area under the curve of every mass spectrum from the mass spec data.

Load a MAT-file, included with the Bioinformatics Toolbox™ software, that contains sample mass spec data, including \(\text{MZ\_lo\_res}\), a vector of m/z values, and \(\text{Y\_lo\_res}\), a matrix of intensity values.

\text{load sample\_lo\_res}

Create a subset (four signals) of the data.

\[
\text{MZ} = \text{MZ\_lo\_res};
\]

\[
\text{Y} = \text{Y\_lo\_res}(:,[1 2 5 6]);
\]

Plot the four spectra.
Normalize the area under the curve (AUC) of every spectrum to the median, eliminating low-mass (m/z < 1,000) noise, and post-rescaling such that the maximum intensity is 100. Plot the four spectra.

Y1 = msnorm(MZ,Y,'Limits', [1000 inf], 'Max', 100);
plot(MZ, Y1)
axis([-1000 20000 -20 105])
xlabel('Mass-charge Ratio')
ylabel('Relative Ion Intensities')
title('AUC Normalized Spectra')
Maximum Intensity Normalization

This example shows how to normalize the ion intensity of every spectrum from the mass spec data.

Load a MAT-file, included with the Bioinformatics Toolbox™ software, that contains sample mass spec data, including `MZ_lo_res`, a vector of m/z values, and `Y_lo_res`, a matrix of intensity values.

```matlab
load sample_lo_res
```

Create a subset (four signals) of the data.

```matlab
MZ = MZ_lo_res;
Y = Y_lo_res(:,[1 2 5 6]);
```

Normalize the ion intensity of every spectrum to the maximum intensity of the single highest peak from any of the spectra in the range above 1000 m/z. Plot the four spectra.

```matlab
Y2 = msnorm(MZ,Y,'QUANTILE', [1 1], 'LIMITS', [1000 inf]);
plot(MZ, Y2)
axis([-1000 20000 -20 105])
xlabel('Mass-charge Ratio')
ylabel('Relative Ion Intensities')
title('Maximum-Intensity Normalized Spectra')
```
Quantile Normalization

This example shows how to perform quantile normalization for mass spec data.

Load a MAT-file, included with the Bioinformatics Toolbox™ software, that contains sample mass spec data, including `MZ_lo_res`, a vector of m/z values, and `Y_lo_res`, a matrix of intensity values.

```matlab
load sample_lo_res
```

Create a subset (four signals) of the data.

```matlab
MZ = MZ_lo_res;
Y = Y_lo_res(:,[1 2 5 6]);
```

Normalize using the data in the m/z regions where the intensities are within the fourth quartile in at least 90% of the spectrograms. Note that you can use the normalization parameters in the second output to normalize another set of data in the same m/z regions. Plot the four spectra.

```matlab
[Y3,S] = msnorm(MZ,Y, 'Quantile', [0.75 1], 'Consensus', 0.9);
area(MZ,S.Xh.*1000,'LineStyle','None','FaceColor', [.8 .8 .8])
hold on
plot(MZ, Y3)
hold off
axis([-1000 20000 -20 105])
xlabel('Mass-charge Ratio')
```
Use the normalization parameters in the second output of the previous step to normalize a different subset of data (four signals) using the data in the same m/z regions as the previous data set. Plot the four spectra.

\[
Y4 = \text{msnorm}(MZ, Y\_lo\_res(:,[3 4 7 8]), S);
\]

\[
\text{area}(MZ, S.Xh.*1000, 'LineStyle', 'None', 'FaceColor', [.8 .8 .8])
\]

\[
\text{hold on}
\]

\[
\text{plot}(MZ, Y4)
\]

\[
\text{hold off}
\]

\[
\text{axis([-1000 20000 -20 105])}
\]

\[
\text{xlabel('Mass-charge Ratio')}
\]

\[
\text{ylabel('Relative Ion Intensities')}
\]

\[
\text{title('Fourth-quartile Normalized Spectra')}
\]
Input Arguments

**X — Vector of separation-unit values for signals with peaks**

vector

Vector of separation-unit values for a set of signals with peaks, specified as a vector.

Data Types: double

**Intensities — Intensity values for set of peaks**

matrix

Intensity values for a set of peaks that share the same separation-unit range, specified as a matrix. Each row is a separation-unit value and each column is either a set of signals with peaks or a retention time. The number of rows in `Intensities` must equal the number of elements in the input vector `X`.

Data Types: double

**NormParameters — Normalization parameters**

structure

Normalization parameters to normalize another group of signals, specified as a structure. `NormParameters` is a structure returned by `msnorm` from a previous normalization call.
Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1, Value1,...,NameN, ValueN.

Example: out = msnorm(X,Y,'Quantile',[0.9 1]) sets the lower (0.9) and upper (1) quantile limit to use only the largest 10% of intensities in each signal to compute the AUC.

Quantile — Quantile limits to reduce separation-unit values in X
[0 1] (default) | 1-by-2 vector | scalar between 0 and 1

Quantile limits to reduce the set of separation-unit values in X, specified as a 1-by-2 vector or a scalar between 0 and 1.

If you specify a vector, the first element is the lower limit and the second element is the upper limit. For example, [0.9 1] means that the function uses only the largest 10% of intensities in each signal to compute the AUC. The default value [0 1] means that the function uses the whole AUC, instead of limiting the intensities to a particular quantile.

If you specify a scalar value, it represents the lower quantile limit. The upper quantile limit is automatically set to 1.

Example: 'Quantile',[0.8 1]
Data Types: double

Limits — Separation-unit range to pick normalization points
[min(X) max(X)] (default) | 1-by-2 vector

Separation-unit range to pick normalization points, specified as a 1-by-2 vector. The default value [min(X) max(X)] selects all available points from X. If you specify a lower or upper limit as a value that is not within the available range [min(X) max(X)], the function sets the lower limit to min(X) and the upper limit to max(X).

This parameter is useful to eliminate noise from the AUC calculation. For instance, you can exclude the matrix noise that appears in the low-mass region (m/z values less than 1000) of a SELDI mass spectrometer by setting the limit to [1000 max(X)].

Example: 'Limits',[900 max(X)]
Data Types: double

Consensus — Minimal percentage of intensity values within quantile limits
scalar between 0 and 1

Minimal percentage of intensity values within the quantile limits that a separation-unit position must have to be included in the AUC calculation, specified as a scalar between 0 and 1. The same separation-unit positions are then used to normalize all the signals. Use this parameter to eliminate low-intensity peaks and noise from the normalization.

For instance, to select m/z regions whose intensities are within the third quantile in at least 90% of the spectrograms, set 'Quantile' and 'Consensus' as follows: yOut = msnorm(MZ,Y,'Quantile',[0.5 0.75], 'Consensus', 0.9).

Example: 'Consensus', 0.8
Data Types: double
Method — Method for normalizing AUC of every signal
'Median' (default) | 'Mean'

Method for normalizing the AUC of every signal, specified as 'Median' or 'Mean'.
Example: 'Method', 'Mean'
Data Types: char | string

Max — Overall maximum intensity to scale to after normalization
scalar

Overall maximum intensity to scale to after normalizing each signal individually, specified as a scalar. If you do not specify this parameter, no postscaling is performed.

**Note** If you specify this value and also set 'Quantile' to [1 1], then a single point (peak height of the tallest peak) is normalized to the specified maximum value.

Example: 'Max'
Data Types: double

### Output Arguments

**yOut — Normalized intensity values**
matrix

Normalized intensity values, returned as a matrix.

**normParams — Normalization parameters**
structure

Normalization parameters that you can use to normalize another group of signals, returned as a structure.

### See Also

msalign | msbackadj | msdotplot | msheatmap | mslowess | msnorm | mspalign | mspeaks | msppresample | msresample | mssgolay | msvviewer

### Topics

“Mass Spectrometry and Bioanalytics”
“Preprocessing Raw Mass Spectrometry Data”
“Visualizing and Preprocessing Hyphenated Mass Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling”
“Differential Analysis of Complex Protein and Metabolite Mixtures using Liquid Chromatography/Mass Spectrometry (LC/MS)”

### Introduced before R2006a
mspalign

Align mass spectra from multiple peak lists from LC/MS or GC/MS data set

Syntax

\[
\text{[CMZ, AlignedPeaks] = mspalign(Peaklist)}
\]

\[
\text{[CMZ, AlignedPeaks] = mspalign(Peaklist, ...'Quantile', QuantileValue, ...)}
\]

\[
\text{[CMZ, AlignedPeaks] = mspalign(Peaklist, ...'EstimationMethod', EstimationMethodValue, ...)}
\]

\[
\text{[CMZ, AlignedPeaks] = mspalign(Peaklist, ...'CorrectionMethod', CorrectionMethodValue, ...)}
\]

\[
\text{[CMZ, AlignedPeaks] = mspalign(Peaklist, ...'ShowEstimation', ShowEstimationValue, ...)}
\]

Input Arguments

<table>
<thead>
<tr>
<th>Peaklist</th>
<th>Cell array of peak lists from a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. Each element in the cell array is a two-column matrix with m/z values in the first column and ion intensity values in the second column. Each element corresponds to a spectrum or retention time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuantileValue</td>
<td>Value that determines which peaks are selected by the estimation method to create CMZ, the vector of common m/z values. Choices are any value ( \geq 0 ) and ( \leq 1 ). Default is 0.95.</td>
</tr>
</tbody>
</table>
| EstimationMethodValue | Character vector or string specifying the method to estimate CMZ, the vector of common mass/charge (m/z) values. Choices are:

  - histogram — Default method. Peak locations are clustered using a kernel density estimation approach. The peak ion intensity is used as a weighting factor. The center of all the clusters conform to the CMZ vector.
  - regression — Takes a sample of the distances between observed significant peaks and regresses the inter-peak distance to create the CMZ vector with similar inter-element distances. |
**CorrectionMethodValue**

Character vector or string specifying the method to align each peak list to the CMZ vector. Choices are:

- **nearest-neighbor** — Default method. For each common peak in the CMZ vector, its counterpart in each peak list is the peak that is closest to the common peak's m/z value.
- **shortest-path** — For each common peak in the CMZ vector, its counterpart in each peak list is selected using the shortest path algorithm.

**ShowEstimationValue**

Controls the display of an assessment plot relative to the estimation method and the vector of common mass/charge (m/z) values. Choices are **true** or **false**. Default is either:

- **false** — When return values are specified.
- **true** — When return values are not specified.

### Output Arguments

<table>
<thead>
<tr>
<th>CMZ</th>
<th>Vector of common mass/charge (m/z) values estimated by the mspalign function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlignedPeaks</td>
<td>Cell array of peak lists, with the same form as Peaklist, but with corrected m/z values in the first column of each matrix.</td>
</tr>
</tbody>
</table>

### Description

\[
[CMZ, \text{AlignedPeaks}] = \text{mspalign} (\text{Peaklist}) \]

aligns mass spectra from multiple peak lists (centroided data), by first estimating CMZ, a vector of common mass/charge (m/z) values estimated by considering the peaks in all spectra in Peaklist, a cell array of peak lists, where each element corresponds to a spectrum or retention time. It then aligns the peaks in each spectrum to the values in CMZ, creating AlignedPeaks, a cell array of aligned peak lists.

\[
[CMZ, \text{AlignedPeaks}] = \text{mspalign} (\text{Peaklist}, ...'\text{PropertyName}', \text{PropertyValue}, ...) 
\]

calls mspalign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

\[
[CMZ, \text{AlignedPeaks}] = \text{mspalign} (\text{Peaklist}, ...'\text{Quantile}', \text{QuantileValue}, ...) 
\]
determines which peaks are selected by the estimation method to create CMZ, the vector of common m/z values. Choices are a scalar between 0 and 1. Default is 0.95.

\[
[CMZ, \text{AlignedPeaks}] = \text{mspalign} (\text{Peaklist}, ...'\text{EstimationMethod}', \text{EstimationMethodValue}, ...) 
\]

specifies the method used to estimate CMZ, the vector of common mass/charge (m/z) values. Choices are:

- **histogram** — Default method. Peak locations are clustered using a kernel density estimation approach. The peak ion intensity is used as a weighting factor. The center of all the clusters conform to the CMZ vector.
- **regression** — Takes a sample of the distances between observed significant peaks and regresses the inter-peak distance to create the CMZ vector with similar inter-element distances.
\([\text{CMZ}, \text{AlignedPeaks}] = \text{mspalign} (\text{Peaklist}, \ldots \text{'CorrectionMethod'}, \text{CorrectionMethodValue}, \ldots)\) specifies the method used to align each peak list to the \text{CMZ} vector. Choices are:

- nearest-neighbor — Default method. For each common peak in the \text{CMZ} vector, its counterpart in each peak list is the peak that is closest to the common peak's m/z value.
- shortest-path — For each common peak in the \text{CMZ} vector, its counterpart in each peak list is selected using the shortest path algorithm.

\([\text{CMZ}, \text{AlignedPeaks}] = \text{mspalign} (\text{Peaklist}, \ldots \text{'ShowEstimation'}, \text{ShowEstimationValue}, \ldots)\) controls the display of an assessment plot relative to the estimation method and the estimated vector of common mass/charge (m/z) values. Choices are true or false. Default is either:

- false — When return values are specified.
- true — When return values are not specified.

**Examples**

1. Load a MAT-file, included with the Bioinformatics Toolbox software, which contains liquid chromatography/mass spectrometry (LC/MS) data variables, including \text{peaks} and \text{ret_time}. \text{peaks} is a cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time. \text{ret_time} is a column vector of retention times associated with the LC/MS data set.

   \text{load lcmsdata}

2. Resample the unaligned data, display it in a heat map, and then overlay a dot plot.

   \([\text{MZ},\text{Y}] = \text{msppresample} (\text{ms_peaks},5000);\)
   \text{msheatmap(MZ,ret_time,log(Y))}
msdotplot(ms_peaks, ret_time)

3 Click the Zoom In button, and then click the dot plot two or three times to zoom in and see how the dots representing peaks overlay the heat map image.
Align the peak lists from the mass spectra using the default estimation and correction methods.

```
[CMZ, aligned_peaks] = mspalign(ms_peaks);
```

Resample the unaligned data, display it in a heat map, and then overlay a dot plot.

```
[MZ2,Y2] = msppresample(aligned_peaks,5000);
msheatmap(MZ2,ret_time,log(Y2))
```
msdotplot(aligned_peaks, ret_time)

6 Link the axes of the two heat plots and zoom in to observe the detail to compare the unaligned and aligned LC/MS data sets.

linkaxes(findobj(0, 'Tag', 'MSHeatMap'))
axis([480 532 375 485])
References


See Also
msalign | msbackadj | msdotplot | msheatmap | mslowess | msnorm | mspalign | mspeaks | msppresample | msresample | mssgolay | msviewer

Topics
“Mass Spectrometry and Bioanalytics”
“Preprocessing Raw Mass Spectrometry Data”
“Visualizing and Preprocessing Hyphenated Mass Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling”
“Differential Analysis of Complex Protein and Metabolite Mixtures using Liquid Chromatography/Mass Spectrometry (LC/MS)”

Introduced in R2007a
mspeaks

Convert raw peak data to peak list (centroided data)

Syntax

Peaklist = mspeaks(X, Intensities)
[Peaklist, PFWHH] = mspeaks(X, Intensities)
[Peaklist, PFWHH, PExt] = mspeaks(X, Intensities)
mspeaks(X, Intensities, ...'Base', BaseValue, ...)
mspeaks(X, Intensities, ...'Levels', LevelsValue, ...)
mspeaks(X, Intensities, ...'NoiseEstimator', NoiseEstimatorValue, ...)
mspeaks(X, Intensities, ...'Multiplier', MultiplierValue, ...)
mspeaks(X, Intensities, ...'PeakLocation', PeakLocationValue, ...)
mspeaks(X, Intensities, ...'FWHHFilter', FWHHFilterValue, ...)
mspeaks(X, Intensities, ...'OverSegmentationFilter', OverSegmentationFilterValue, ...)
mspeaks(X, Intensities, ...'HeightFilter', HeightFilterValue, ...)
mspeaks(X, Intensities, ...'ShowPlot', ShowPlotValue, ...)
mspeaks(X, Intensities, ...'Style', StyleValue, ...)

Description

Peaklist = mspeaks(X, Intensities) finds relevant peaks in raw, noisy peak signal data, and creates Peaklist, a two-column matrix, containing the separation-axis value and intensity for each peak. X is a vector of separation-unit values for a set of signals with peaks. Intensities is a matrix of intensity values for a set of peaks that share the same separation-unit range.

[Peaklist, PFWHH] = mspeaks(X, Intensities) returns PFWHH, a two-column matrix indicating the left and right locations of the full width at half height (FWHH) markers for each peak. For any peak not resolved at FWHH, mspeaks returns the peak shape extents instead. When Intensities includes multiple signals, then PFWHH is a cell array of matrices.

[Peaklist, PFWHH, PExt] = mspeaks(X, Intensities) returns PExt, a two-column matrix indicating the left and right locations of the peak shape extents determined after wavelet denoising. When Intensities includes multiple signals, then PExt is a cell array of matrices.

mspeaks(X, Intensities, ...'PropertyName', PropertyValue, ...) calls mspeaks with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each PropertyName in single quotation marks. Each PropertyName is case insensitive. These property name/property value pairs are as follows:

mspeaks(X, Intensities, ...'Base', BaseValue, ...) specifies the wavelet base.
mspeaks(X, Intensities, ...'Levels', LevelsValue, ...) specifies the number of levels for the wavelet decomposition.
mspeaks(X, Intensities, ...'NoiseEstimator', NoiseEstimatorValue, ...) specifies the method to estimate the threshold, T, to filter out noisy components in the first high-band decomposition (y_h).
mspeaks($X$, $Intensities$, ...'Multiplier', $MultiplierValue$, ...) specifies the threshold multiplier constant.

mspeaks($X$, $Intensities$, ...'Denoising', $DenoisingValue$, ...) controls the use of wavelet denoising to smooth the signal. Choices are true (default) or false.

mspeaks($X$, $Intensities$, ...'PeakLocation', $PeakLocationValue$, ...) specifies the proportion of the peak height to use to select the points used to compute the centroid separation-axis value of the respective peak. $PeakLocationValue$ must be a value $\geq 0$ and $\leq 1$. Default is 1.0.

mspeaks($X$, $Intensities$, ...'FWHHFilter', $FWHHFilterValue$, ...) specifies the minimum full width at half height (FWHH), in separation units, for reported peaks. Peaks with FWHH below this value are excluded from the output list $Peaklist$.

mspeaks($X$, $Intensities$, ...'OverSegmentationFilter', $OverSegmentationFilterValue$, ...) specifies the minimum distance, in separation units, between neighboring peaks. When the signal is not smoothed appropriately, multiple maxima can appear to represent the same peak. Increase this filter value to join oversegmented peaks into a single peak.

mspeaks($X$, $Intensities$, ...'HeightFilter', $HeightFilterValue$, ...) specifies the minimum height for reported peaks. Peaks with heights below this value are excluded from the output list $Peaklist$.

mspeaks($X$, $Intensities$, ...'ShowPlot', $ShowPlotValue$, ...) controls the display of a plot of the original and the smoothed signal, with the peaks included in the output matrix $Peaklist$ marked.

mspeaks($X$, $Intensities$, ...'Style', $StyleValue$, ...) specifies the style for marking the peaks in the plot.

mspeaks finds peaks in data from any separation technique that produces signal data, such as spectroscopy, nuclear magnetic resonance (NMR), electrophoresis, chromatography, or mass spectrometry.

**Input Arguments**

$X$

Vector of separation-unit values for a set of signals with peaks. The number of elements in the vector equals the number of rows in the matrix $Intensities$. The separation unit can quantify wavelength, frequency, distance, time, or m/z depending on the instrument that generates the signal data.

Default:

$Intensities$

Matrix of intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals the number of elements in vector $X$.

Default:
**BaseValue**

Integer from 2 to 20 that specifies the wavelet base.

**Default:** 4

**LevelsValue**

Integer from 1 to 12 that specifies the number of levels for the wavelet decomposition.

**Default:** 10

**NoiseEstimatorValue**

Character vector, string, or scalar that specifies the method to estimate the threshold, $T$, to filter out noisy components in the first high-band decomposition ($y_h$). Choices are:

- `mad` — Default. Median absolute deviation, which calculates $T = \sqrt{2 \log(n)} \times \text{mad}(y_h) / 0.6745$, where $n$ = the number of rows in the Intensities matrix.
- `std` — Standard deviation, which calculates $T = \text{std}(y_h)$.
- A positive real value.

**Default:**

**MultiplierValue**

Positive real value that specifies the threshold multiplier constant.

**Default:** 1.0

**DenoisingValue**

Controls the use of wavelet denoising to smooth the signal. Choices are `true` (default) or `false`.

**Tip** If your data was previously smoothed, for example, with the mslowess or mssgolay function, you do not need to use wavelet denoising. Set this property to `false`.

**Default:**

**PeakLocationValue**

Value that specifies the proportion of the peak height to use to select the points to compute the centroid separation-axis value of the respective peak. The value must be $\geq 0$ and $\leq 1$.

**Note** When `PeakLocationValue` = 1.0, the peak location is at the maximum of the peak. When `PeakLocationValue` = 0, mspeaks computes the peak location with all the points from the closest minimum to the left of the peak to the closest minimum to the right of the peak.

**Default:** 1.0
**FWHHFilterValue**

Positive real value that specifies the minimum full width at half height (FWHH), in separation units, for reported peaks. Peaks with FWHH below this value are excluded from the output list `Peaklist`.

**Default:** 0

**OverSegmentationFilterValue**

Positive real value that specifies the minimum distance, in separation units, between neighboring peaks. When the signal is not smoothed appropriately, multiple maxima can appear to represent the same peak. Increase this filter value to join oversegmented peaks into a single peak.

**Default:** 0

**HeightFilterValue**

Positive real value that specifies the minimum height for reported peaks.

**Default:** 0

**ShowPlotValue**

Controls the display of a plot of the original signal and the smoothed signal, with the peaks included in the output matrix `Peaklist` marked. Choices are `true`, `false`, or `I`, an integer specifying the index of a spectrum in `Intensities`. If set to `true`, the first spectrum in `Intensities` is plotted. Default is:

- `false` — When you specify return values.
- `true` — When you do not specify return values.

**Default:**

**StyleValue**

Character vector or string specifying the style for marking the peaks in the plot. Choices are:

- `'peak'` (default) — Places a marker at the peak crest.
- `'exttriangle'` — Draws a triangle using the peak crest and the extents.
- `'fwhhtriangle'` — Draws a triangle using the peak crest and the FWHH points.
- `'extline'` — Places a marker at the peak crest and vertical lines at the extents.
- `'fwhhline'` — Places a marker at the peak crest and a horizontal line at FWHH.

**Default:**

**Output Arguments**

**Peaklist**

Two-column matrix where each row corresponds to a peak. The first column contains separation-unit values (indicating the location of peaks along the separation axis). The second column contains intensity values. When `Intensities` includes multiple signals, then `Peaklist` is a cell array of matrices, each containing a peak list.
PFW HH

Two-column matrix indicating the left and right locations of the full width at half height (FWHH) markers for each peak. For any peak not resolved at FWHH, mspeaks returns the peak shape extents instead. When Intensities includes multiple signals, then PFW HH is a cell array of matrices.

PExt

Two-column matrix indicating the left and right locations of the peak shape extents determined after wavelet denoising. When Intensities includes multiple signals, then PExt is a cell array of matrices.

Examples

1 Load a MAT-file, included with the Bioinformatics Toolbox software, that contains two mass spectrometry data variables, MZ_lo_res and Y_lo_res. MZ_lo_res is a vector of m/z values for a set of spectra. Y_lo_res is a matrix of intensity values for a set of mass spectra that share the same m/z range.

   load sample_lo_res

2 Adjust the baseline of the eight spectra stored in Y_lo_res.

   YB = msbackadj(MZ_lo_res,Y_lo_res);

3 Convert the raw mass spectrometry data to a peak list by finding the relevant peaks in each spectrum.

   P = mspeaks(MZ_lo_res,YB);

4 Plot the third spectrum in YB, the matrix of baseline-corrected intensity values, with the detected peaks marked.

   P = mspeaks(MZ_lo_res,YB,'SHOWPLOT',3);
Smooth the signal using the `mslowess` function. Then convert the smoothed data to a peak list by finding relevant peaks and plot the third spectrum.

```matlab
YS = mslowess(MZ_lo_res,YB,'SHOWPLOT',3);
```

![Smoothed signal](image1.png)

```matlab
P = mspeaks(MZ_lo_res,YS,'DENOISING',false,'SHOWPLOT',3);
```

![Peak list](image2.png)
6 Use the `cellfun` function to remove all peaks with m/z values less than 2000 from the eight peaks listed in output `P`. Then plot the peaks of the third spectrum (in red) over its smoothed signal (in blue).

```matlab
Q = cellfun(@(p) p(p(:,1)>2000,:),P,'UniformOutput',false);
figure
plot(MZ_lo_res,YS(:,3),'b',Q{3}(:,1),Q{3}(:,2),'rx')
xlabel('Mass/Charge (M/Z)')
ylabel('Relative Intensity')
axis([0 20000 -5 95])
```

![Image](image.png)

### Algorithms

`mspeaks` converts raw peak data to a peak list (centroided data) by:

1. Smoothing the signal using undecimated wavelet transform with Daubechies coefficients
2. Assigning peak locations
3. Estimating noise
4. Eliminating peaks that do not satisfy specified criteria

### References


See Also
msalign, msbackadj, msdotplot, mshatmap, mslowess, msnorm, mspalign, mspeaks, msppresample, msresample, mssgolay, msvviewer

Topics
“Mass Spectrometry and Bioanalytics”
“Preprocessing Raw Mass Spectrometry Data”
“Visualizing and Preprocessing Hyphenated Mass Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling”
“Differential Analysis of Complex Protein and Metabolite Mixtures using Liquid Chromatography/Mass Spectrometry (LC/MS)”

Introduced in R2007a
## msppresample

Resample signal with peaks while preserving peaks

### Syntax

\[[X, \text{Intensities}] = \text{msppresample}(\text{Peaklist}, N)\]

\[
\text{msppresample}(\text{Peaklist}, N, \ldots 'Range', \text{RangeValue}, \ldots)
\]

\[
\text{msppresample}(\text{Peaklist}, N, \ldots 'FWHH', \text{FWHHValue}, \ldots)
\]

\[
\text{msppresample}(\text{Peaklist}, N, \ldots 'ShowPlot', \text{ShowPlotValue}, \ldots)
\]

### Input Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Peaklist</strong></td>
<td>Either of the following:</td>
</tr>
<tr>
<td></td>
<td>• Two-column matrix, where the first column contains separation-unit values and the second column contains intensity values. The separation unit can quantify wavelength, frequency, distance, time, or m/z depending on the instrument that generates the signal data.</td>
</tr>
<tr>
<td></td>
<td>• Cell array of peak lists, where each element is a two-column matrix of separation-unit values and intensity values, and each element corresponds to a signal or retention time.</td>
</tr>
</tbody>
</table>

**Tip** You can use the `mzxml2peaks` function or the `mspeaks` function to create the `Peaklist` matrix or cell array.

<table>
<thead>
<tr>
<th><strong>N</strong></th>
<th>Integer specifying the number of equally spaced points (separation-unit values) in the resampled signal.</th>
</tr>
</thead>
</table>

| **RangeValue** | 1-by-2 vector specifying the minimum and maximum separation-unit values for the output matrix `Intensities`. `RangeValue` must be within \[\min(\text{inputSU}) \text{ max}(\text{inputSU})\], where `inputSU` is the concatenated separation-unit values from the input `Peaklist`. Default is the full range \[\min(\text{inputSU}) \text{ max}(\text{inputSU})\]. |

| **FWHHValue** | Value that specifies the full width at half height (FWHH) in separation units. The FWHH is used to convert each peak to a Gaussian shaped curve. Default is \(\text{median}(\text{diff( inputSU)})/2\), where `inputSU` is the concatenated separation-unit values from the input `Peaklist`. The default is a rough approximation of resolution observed in the input data, `Peaklist`. |

**Tip** To ensure that the resolution of the peaks is preserved, set `FWHHValue` to half the distance between the two peaks of interest that are closest to each other.
ShowPlotValue | Controls the display of a plot of an original and resampled signal. Choices are true, false, or I, an integer specifying the index of a signal in Intensities. If you set to true, the first signal in Intensities is plotted. Default is:

- false — When return values are specified.
- true — When return values are not specified.

Output Arguments

<table>
<thead>
<tr>
<th>X</th>
<th>Vector of equally spaced, common separation-unit values for a set of signals with peaks. The number of elements in the vector equals ( N ), or the number of rows in matrix Intensities.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intensities</td>
<td>Matrix of reconstructed intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals ( N ), or the number of elements in vector X.</td>
</tr>
</tbody>
</table>

Description

**Tip** Use the following syntaxes with data from any separation technique that produces signal data, such as spectroscopy, NMR, electrophoresis, chromatography, or mass spectrometry.

\[
[X, \text{Intensities}] = \text{msppresample}(\text{Peaklist}, N) \]

resamples Peaklist, a peak list, by converting centroided peaks to a semicontinuous, raw signal that preserves peak information. The resampled signal has \( N \) equally spaced points. Output \( X \) is a vector of \( N \) elements specifying the equally spaced, common separation-unit values for the set of signals with peaks. Output Intensities is a matrix of reconstructed intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals \( N \).

\( \text{msppresample} \) uses a Gaussian kernel to reconstruct the signal. The intensity at any given separation-unit value is taken from the maximum intensity of any contributing (overlapping) peaks.

**Tip** \( \text{msppresample} \) is useful to prepare a set of signals for imaging functions such as \( \text{msheatmap} \) and preprocessing functions such as \( \text{msbackadj} \) and \( \text{msnorm} \).

\( \text{msppresample}(\text{Peaklist}, N, ... \text{'PropertyName'}, \text{PropertyValue}, ...) \) calls \( \text{msppresample} \) with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

\( \text{msppresample}(\text{Peaklist}, N, ... \text{'Range'}, \text{RangeValue}, ...) \) specifies a separation-unit range for the output matrix Intensities using the minimum and maximum separation values specified in the 1-by-2 vector RangeValue. RangeValue must be within \( \min(\text{inputSU}) \).
max(inputSU), where inputSU is the concatenated separation-unit values from the input Peaklist. Default is the full range \([\min(inputSU) \ max(inputSU)]\)

msppresample(Peaklist, N, ...'FWHH', FWHHValue, ...) sets the full width at half height (FWHH) in separation units. The FWHH is used to convert each peak to a Gaussian shaped curve. Default is \(\text{median}(\text{diff(inputSU)})/2\), where inputSU is the concatenated separation-unit values from the input Peaklist. The default is a rough approximation of resolution observed in the input data, Peaklist.

**Tip** To ensure that the resolution of the peaks is preserved, set FWHHValue to half the distance between the two peaks of interest that are closest to each other.

msppresample(Peaklist, N, ...'ShowPlot', ShowPlotValue, ...) controls the display of a plot of an original and resampled signal. Choices are true, false, or I, an integer specifying the index of a signal in Intensities. If you set to true, the first signal in Intensities is plotted. Default is:

- false — When return values are specified.
- true — When return values are not specified.

**Examples**

1 Load a MAT-file, included with the Bioinformatics Toolbox software, that contains liquid chromatography/mass spectrometry (LC/MS) data variables. It includes peaks, a cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time.

   load lcmsdata

2 Resample the data, specifying 5000 m/z values in the resampled signal. Then create a heat map of the LC/MS data.

   \([\text{MZ,Y}] = \text{msppresample}(\text{ms_peaks},5000)\);
   \text{msheatmap(MZ,ret_time,log(Y))}\)
Plot the reconstructed profile spectra between two retention times.

```matlab
figure
t1 = 3370;
t2 = 3390;
h = find(ret_time>t1 & ret_time<t2);
[MZ,Y] = msppresample(ms_peaks(h),10000);
plot3(repmat(MZ,1,numel(h)),repmat(ret_time(h)',10000,1),Y)
xlabel('Mass/Charge (M/Z)')
ylabel('Retention Time')
zlabel('Relative Intensity')
```
4 Resample the data to plot the Total Ion Chromatogram (TIC).

```matlab
figure
[MZ,Y] = msppresample(ms_peaks,5000);
plot(ret_time,sum(Y))
title('Total Ion Chromatogram (TIC)')
xlabel('Retention Time')
ylabel('Relative Intensity')
```
Resample the data to plot the Extracted Ion Chromatogram (XIC) in the 450 to 500 m/z range.

```matlab
figure
[MZ,Y] = msppresample(ms_peaks,5000,'Range',[450 500]);
plot(ret_time,sum(Y))
title('Extracted Ion Chromatogram (XIC) from 450 to 500 M/Z')
xlabel('Retention Time')
ylabel('Relative Intensity')
```

See Also
- msalign
- msbackadj
- msdplot
- msheatmap
- mslowess
- msnorm
- mspalign
- mspeaks
- msppresample
- msresample
- mssgolay
- msvviewer

Topics
- "Mass Spectrometry and Bioanalytics"
- "Preprocessing Raw Mass Spectrometry Data"
- "Visualizing and Preprocessing Hyphenated Mass Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling"
- "Differential Analysis of Complex Protein and Metabolite Mixtures using Liquid Chromatography/Mass Spectrometry (LC/MS)"

Introduced in R2007a
msresample

Resample signal with peaks

Syntax

\[
[Xout, \text{Intensitiesout}] = \text{msresample}(X, \text{Intensities}, N)
\]

\[
\text{msresample}(..., \text{Uniform}', \text{UniformValue}, ...)
\]

\[
\text{msresample}(..., \text{Range}', \text{RangeValue}, ...)
\]

\[
\text{msresample}(..., \text{RangeWarnOff}', \text{RangeWarnOffValue}, ...)
\]

\[
\text{msresample}(..., \text{Missing}', \text{MissingValue}, ...)
\]

\[
\text{msresample}(..., \text{Window}', \text{WindowValue}, ...)
\]

\[
\text{msresample}(..., \text{Cutoff}', \text{CutoffValue}, ...)
\]

\[
\text{msresample}(..., \text{ShowPlot}', \text{ShowPlotValue}, ...)
\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X)</td>
<td>Vector of separation-unit values for a set of signals with peaks. The number of elements in the vector equals the number of rows in the matrix (\text{Intensities}). The separation unit can quantify wavelength, frequency, distance, time, or m/z depending on the instrument that generates the signal data.</td>
</tr>
<tr>
<td>(\text{Intensities})</td>
<td>Matrix of intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals the number of elements in vector (X).</td>
</tr>
<tr>
<td>(N)</td>
<td>Positive integer specifying the total number of samples.</td>
</tr>
</tbody>
</table>

Description

**Tip** Use the following syntaxes with data from any separation technique that produces signal data, such as spectroscopy, NMR, electrophoresis, chromatography, or mass spectrometry.

\[
[Xout, \text{Intensitiesout}] = \text{msresample}(X, \text{Intensities}, N)
\] resamples raw noisy signal data, \(\text{Intensities}\). The output signal has \(N\) samples with a spacing that increases linearly within the range \([\min(X) \, \max(X)]\). \(X\) can be a linear or a quadratic function of its index. When you set input arguments such that down-sampling takes place, \text{msresample} applies a lowpass filter before resampling to minimize aliasing.

For the antialias filter, \text{msresample} uses a linear-phase FIR filter with a least-squares error minimization. The cutoff frequency is set by the largest down-sampling ratio when comparing the same regions in the \(X\) and \(Xout\) vectors.

**Tip** \text{msresample} is particularly useful when you have signals with different separation-unit vectors and you want to match the scales.
msresample(..., 'PropertyName', PropertyValue, ...) calls msresample with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

msresample(..., 'Uniform', UniformValue, ...), when UniformValue is true, it forces the vector X to be uniformly spaced. The default value is false.

msresample(..., 'Range', RangeValue, ...) specifies a 1-by-2 vector with the separation-unit range for the output signal, Intensitiesout. RangeValue must be within [min(X) max(X)]. Default value is the full range [min(X) max(X)]. When RangeValue values exceed the values in X, msresample extrapolates the signal with zeros and returns a warning message.

msresample(..., 'RangeWarnOff', RangeWarnOffValue, ...) controls the return of a warning message when RangeValue values exceed the values in X. RangeWarnOffValue can be true or false (default).

msresample(..., 'Missing', MissingValue, ...), when MissingValue is true, analyzes the input vector, X, for dropped samples. The default value is false. If the down-sample factor is large, checking for dropped samples might not be worth the extra computing time. Dropped samples can only be recovered if the original separation-unit values follow a linear or a quadratic function of the X vector index.

msresample(..., 'Window', WindowValue, ...) specifies the window used when calculating parameters for the lowpass filter. Enter 'Flattop', 'Blackman', 'Hamming', or 'Hanning'. The default value is 'Flattop'.

msresample(..., 'Cutoff', CutoffValue, ...) specifies the cutoff frequency. Enter a scalar value from 0 to 1 (Nyquist frequency or half the sampling frequency). By default, msresample estimates the cutoff value by inspecting the separation-unit vectors, X and XOut. However, the cutoff frequency might be underestimated if X has anomalies.

msresample(..., 'ShowPlot', ShowPlotValue, ...) plots the original and the resampled signal. When msresample is called without output arguments, the signals are plotted unless ShowPlotValue is false. When ShowPlotValue is true, only the first signal in Intensities is plotted. ShowPlotValue can also contain an index to one of the signals in Intensities.

**Tip** LC/MS data analysis requires extended amounts of memory from the operating system.

- If you receive errors related to memory, try the following:
  - Increase the virtual memory (swap space) for your operating system as described in “Resolve “Out of Memory” Errors” (MATLAB).
- If you receive errors related to Java heap space, increase your Java heap space:
  - If you have MATLAB version 7.10 (R2010a) or later, see “Java Heap Memory Preferences” (MATLAB).
  - If you have MATLAB version 7.9 (R2009b) or earlier, see https://www.mathworks.com/support/solutions/en/data/1-18I2C/.
Examples

Resample Mass Spectrometry Data

This example shows how to resample mass spec data.

Load a MAT-file, included with Bioinformatics Toolbox™, that contains mass spectrometry data, and then extract m/z and intensity value vectors.

```matlab
load sample_hi_res;
mz = MZ_hi_res;
y = Y_hi_res;
```

Plot the original data.

```matlab
plot(mz, y, '.')
```

Resample the spectrogram to have 10000 samples between 2000 and maximum m/z value in the data set, and show both the resampled and original data.

```matlab
[mz1,y1] = msresample(mz, y, 10000, 'range',[2000 max(mz)],'SHOWPLOT',true);
```
See Also
msalign | msbackadj | msdotplot | msheetmap | mslowess | msnorm | mspalign | mspeaks |
msppresample | msresample | mssgolay | msviewer

Topics
“Mass Spectrometry and Bioanalytics”
“Preprocessing Raw Mass Spectrometry Data”
“Visualizing and Preprocessing Hyphenated Mass Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling”
“Differential Analysis of Complex Protein and Metabolite Mixtures using Liquid Chromatography/Mass Spectrometry (LC/MS)”

Introduced before R2006a
mssgolay

Smooth signal with peaks using least-squares polynomial

Syntax

\[ \text{Yout} = \text{mssgolay}(X, \text{Intensities}) \]

\[ \text{mssgolay}(X, \text{Intensities}, ...'\text{Span}', \text{SpanValue}, ...) \]
\[ \text{mssgolay}(X, \text{Intensities}, ...'\text{Degree}', \text{DegreeValue}, ...) \]
\[ \text{mssgolay}(X, \text{Intensities}, ...'\text{ShowPlot}', \text{ShowPlotValue}, ...) \]

Arguments

<table>
<thead>
<tr>
<th>( X )</th>
<th>Vector of separation-unit values for a set of signals with peaks. The number of elements in the vector equals the number of rows in the matrix ( \text{Intensities} ). The separation unit can quantify wavelength, frequency, distance, time, or m/z depending on the instrument that generates the signal data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Intensities} )</td>
<td>Matrix of intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals the number of elements in vector ( X ).</td>
</tr>
</tbody>
</table>

Description

Tip Use the following syntaxes with data from any separation technique that produces signal data, such as spectroscopy, NMR, electrophoresis, chromatography, or mass spectrometry.

\[ \text{Yout} = \text{mssgolay}(X, \text{Intensities}) \] smooths raw noisy signal data, \( \text{Intensities} \), using a least-squares digital polynomial filter (Savitzky and Golay filters). The default span or frame is 15 samples.

\[ \text{mssgolay}(X, \text{Intensities}, ...'\text{PropertyName}', \text{PropertyValue}, ...) \] calls mssgolay with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each \( \text{PropertyName} \) must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

\[ \text{mssgolay}(X, \text{Intensities}, ...'\text{Span}', \text{SpanValue}, ...) \] modifies the frame size for the smoothing function. If \( \text{SpanValue} \) is greater than 1, the window is the size of \( \text{SpanValue} \) in samples independent of the \( X \) vector. Higher values smooth the signal more with an increase in computation time. If \( \text{SpanValue} \) is less than 1, the window size is a fraction of the number of points in the input data, \( X \). For example, if \( \text{SpanValue} \) is 0.05, the window size is equal to 5% of the number of points in \( X \).

Note The original algorithm by Savitzky and Golay assumes the input vector, \( X \), has uniformly spaced separation units, while mssgolay also allows one that is not uniformly spaced. Therefore, the sliding
frame for smoothing is centered using the closest samples in terms of the $X$ value and not in terms of the $X$ index.

When the input vector, $X$, does not have repeated values or NaN values, the algorithm is approximately twice as fast.

When the input vector, $X$, is evenly spaced, the least-squares fitting is performed once so that the signal is filtered with the same coefficients, and the speed of the algorithm increases considerably.

If the input vector, $X$, is evenly spaced and $SpanValue$ is even, span is incremented by 1 to include both edge samples in the frame.

\[ \text{mssgolay}(X, \text{Intensities}, ...'\text{Degree}', \text{DegreeValue}, ...) \]

specifies the degree of the polynomial ($\text{DegreeValue}$) fitted to the points in the moving frame. The default value is 2. $\text{DegreeValue}$ must be smaller than $SpanValue$.

\[ \text{mssgolay}(X, \text{Intensities}, ...'\text{ShowPlot}', \text{ShowPlotValue}, ...) \]

plots smoothed signals over the original. When \text{mssgolay} is called without output arguments, the signals are plotted unless $\text{ShowPlotValue}$ is false. When $\text{ShowPlotValue}$ is true, only the first signal in $\text{Intensities}$ is plotted. $\text{ShowPlotValue}$ can also contain an index to one of the signals in $\text{Intensities}$.

**Examples**

**Smooth Mass Spectrometry Data**

This example shows how to smooth mass spectrometry data using least-squares polynomial approach.

Load a MAT-file, included with Bioinformatics Toolbox™, that contains mass spectrometry data including $\text{MZ_lo_res}$, a vector of m/z values for a set of spectra, and $\text{Y_lo_res}$, a matrix of intensity values for a set of mass spectra that share the same m/z charge.

\[ \text{load sample_lo_res} \]

Apply least-squares polynomial smoothing to the data.

\[ \text{YS} = \text{mssgolay(MZ_lo_res, Y_lo_res)}; \]

Plot the third sample/spectrogram in $\text{Y_lo_res}$, and its smoothed signal.

\[ \text{mssgolay(MZ_lo_res,Y_lo_res,}'\text{SHOWPLOT}',3)\]


See Also
msalign | msbackadj | msdotplot | msheatmap | mslowess | msnorm | msalign | mspeaks |
mspresample | msresample | mssgolay | msvviewer

Topics
“Mass Spectrometry and Bioanalytics”
“Preprocessing Raw Mass Spectrometry Data”
“Visualizing and Preprocessing Hyphenated Mass Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling”
“Differential Analysis of Complex Protein and Metabolite Mixtures using Liquid Chromatography/Mass Spectrometry (LC/MS)”

Introduced before R2006a
**msviewer**

Explore mass spectrum or set of mass spectra

**Syntax**

msviewer(MZ , Intensities)
msviewer(..., 'Markers', MarkersValue)
msviewer(..., 'Group', GroupValue)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MZ</td>
<td>Column vector of common mass/charge (m/z) values for a set of spectra. The number of elements in the vector equals the number of rows in the matrix Intensities.</td>
</tr>
<tr>
<td>Intensities</td>
<td>Matrix of intensity values for a set of mass spectra that share the same m/z range. Each row corresponds to an m/z value, and each column corresponds to a spectrum or retention time. The number of rows equals the number of elements in vector MZ.</td>
</tr>
<tr>
<td>MarkersValue</td>
<td>Column vector to specify a list of marker positions from the mass/charge vector MZ.</td>
</tr>
<tr>
<td>GroupValue</td>
<td>Either of the following:</td>
</tr>
<tr>
<td></td>
<td>• Vector of values with the same number of elements as rows in the matrix Intensities</td>
</tr>
<tr>
<td></td>
<td>• Cell array of character vectors or string vector with the same number of elements as rows (spectra) in the matrix Intensities</td>
</tr>
<tr>
<td></td>
<td>Each value, character vector, or string specifies a group to which the corresponding spectrum belongs. Spectra from the same group are plotted with the same color. Default is [1:numSpectra].</td>
</tr>
</tbody>
</table>

**Description**

msviewer(MZ , Intensities) displays the MS Viewer, which lets you view and explore a mass spectrum defined by MZ and Intensities.

msviewer(..., 'Markers', MarkersValue) specifies a list of marker positions from the mass/charge vector, MZ, for exploration and easy navigation. Enter a column vector with MZ values.

msviewer(..., 'Group', GroupValue) specifies a group to which the spectra belong. The groups are specified by GroupValue, a vector of values or cell array of character vectors or string vector. The number of values or character vectors or strings is the same as the number of rows in the matrix Intensities. Each value or character vector specifies a group to which the corresponding spectrum belongs. Spectra from the same group are plotted with the same color. Default is [1:numSpectra].

The MS Viewer includes the following features:
• Plot mass spectra. The spectra are plotted with different colors according to their group labels.
• An overview displays a full spectrum, and a box indicates the region that is currently displayed in the main window.
• Five different zoom in options, one zoom out option, and a reset view option resize the spectrum.
• Add/focus/move/delete marker operations
• Import/Export markers from/to MATLAB workspace
• Print and preview the spectra plot
• Print the spectra plot to a MATLAB Figure window

MSViewer has five components:
• Menu bar: File, Tools, Window, and Help
• Toolbar: Move marker, Zoom XY, Zoom X, Zoom Y, Zoom out, Reset view, and Help
• Main window: display the spectra
• Overview window: display the overview of a full spectrum (the average of all spectra in display)
• Marker control panel: a list of markers, Add Marker, Delete Marker, up and down buttons

**Examples**

**Plot Mass Spectra Data**

This example shows how to plot mass spectra data.

Load and plot a sample mass spectra data.

```matlab
load sample_lo_res
msviewer(MZ_lo_res, Y_lo_res)
```
Add a marker by pointing to a mass peak, right-clicking, and then clicking **Add Marker**.

The **File** menu has the following options.

- **Import Markers from Workspace** - Opens the Import Markers From MATLAB® Workspace dialog. The dialog displays a list of double Mx1 or 1xM variables. If the selected variable is out of range, the viewer displays an error message.

- **Export Markers to Workspace** - Opens the Export Markers to MATLAB® Workspace dialog. Enter a variable name for the markers. All markers are saved. If there is no marker available, this menu item is disabled.

- **Print to Figure** - Prints the spectra plot in the main display to a MATLAB® figure window.

The **Tools** menu has the following options.

- **Add Marker** - Opens the Add Marker dialog where you can enter an m/z marker.
- **Delete Marker** - Removes the currently selected m/z marker from the **Markers** (m/z) list.
- **Next Marker** or **Previous Marker** - Moves the selection up and down the **Markers** list.
• **Zoom XY, Zoom X, Zoom Y, or Zoom Out** - Changes the cursor from an arrow to a crosshair. Left-click and drag a rectangle box over an area and then release it. The display zooms the area covered by the box.

From the range window at the bottom, move the view box to a new location.

**See Also**
msalign | msbackadj | msdotplot | msheatmap | mslowess | msnorm | mspalign | mspeaks | msppresample | msresample | mssgolay | msviewer

**Topics**
“Mass Spectrometry and Bioanalytics”
“Preprocessing Raw Mass Spectrometry Data”
“Visualizing and Preprocessing Hyphenated Mass Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling”
“Differential Analysis of Complex Protein and Metabolite Mixtures using Liquid Chromatography/Mass Spectrometry (LC/MS)”

**Introduced before R2006a**
multialign

Align multiple sequences using progressive method

Syntax

\[
\text{SeqsMultiAligned} = \text{multialign}(\text{Seqs}) \\
\text{SeqsMultiAligned} = \text{multialign}(\text{Seqs}, \text{Tree})
\]

multialign(..., 'PropertyName', PropertyValue,...)
multialign(..., 'Weights', WeightsValue)
multialign(..., 'ScoringMatrix', ScoringMatrixValue)
multialign(..., 'SMInterp', SMInterpValue)
multialign(..., 'GapOpen', GapOpenValue)
multialign(..., 'ExtendGap', ExtendGapValue)
multialign(..., 'DelayCutoff', DelayCutoffValue)
multialign(..., 'UseParallel', UseParallelValue)
multialign(..., 'Verbose', VerboseValue)
multialign(..., 'ExistingGapAdjust', ExistingGapAdjustValue)
multialign(..., 'TerminalGapAdjust', TerminalGapAdjustValue)

Input Arguments

\text{Seqs} \\
\text{Vector of structures with the fields 'Sequence' for the residues and 'Header' or 'Name' for the labels.} \\
\text{Seqs} can also be a string vector, cell array of character vectors, or character array.

\text{Tree} \\
\text{Phylogenetic tree calculated with the seqlinkage on page 1-1508 or seqneighjoin on page 1-1518 function.}

\text{WeightsValue} \\
\text{Property to select the sequence weighting method. Enter 'THG' (default) or 'equal'.}
ScoringMatrixValue

Either of the following:

- Character vector or string specifying the scoring matrix to use for the alignment. Choices for amino acid sequences are:
  - 'BLOSUM62'
  - 'BLOSUM30' increasing by 5 up to 'BLOSUM90'
  - 'BLOSUM100'
  - 'PAM10' increasing by 10 up to 'PAM500'
  - 'DAYHOFF'
  - 'GONNET'

Default is:

- 'BLOSUM80' to 'BLOSUM30' series — When AlphabetValue equals 'AA'
- 'NUC44' — When AlphabetValue equals 'NT'

Note The above scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the 'Scale' property to specify an additional scale factor to convert the output score from bits to another unit.

- Matrix representing the scoring matrix to use for the alignment. It can be a matrix, such as returned by the blosum, pam, dayhoff, gonnet, or nuc44 function. It can also be an $M$-by-$M$ matrix or $M$-by-$M$-by-$N$ array of matrices with $N$ user-defined scoring matrices.

Note If you use a scoring matrix that you created or was created by one of the above functions, the matrix does not include a scale factor. The output score will be returned in the same units as the scoring matrix. When passing your own series of scoring matrices, ensure they share the same scale.

SMInterpValue

Property to specify whether linear interpolation of the scoring matrices is on or off. When false, the scoring matrix is assigned to a fixed range depending on the distances between the two profiles (or sequences) being aligned. Default is true.
**GapOpenValue**
Scalar or a function specified using @. If you enter a function, `multialign` passes four values to the function: the average score for two matched residues (sm), the average score for two mismatched residues (sx), and the length of both profiles or sequences (len1, len2). Default is @(sm,sx,len1,len2) 5*sm.

**ExtendGapValue**
Scalar or a function specified using @. If you enter a function, `multialign` passes four values to the function: the average score for two matched residues (sm), the average score for two mismatched residues (sx), and the length of both profiles or sequences (len1, len2). Default is @(sm,sx,len1,len2) sm/4.

**DelayCutoffValue**
Property to specify the threshold delay of divergent sequences. Default is unity where sequences with the closest sequence farther than the median distance are delayed.

**UseParallelValue**
Controls the computation of the pairwise alignments using parfor-loops. When true, and Parallel Computing Toolbox is installed and a parpool is open, computation occurs in parallel. If there are no open parpool, but automatic creation is enabled in the Parallel Preferences, the default pool will be automatically open and computation occurs in parallel. If Parallel Computing Toolbox is installed, but there are no open parpool and automatic creation is disabled, then computation uses parfor-loops in serial mode. If Parallel Computing Toolbox is not installed, then computation uses parfor-loops in serial mode. Default is false, which uses for-loops in serial mode.

**VerboseValue**
Property to control displaying the sequences with sequence information. Default is false.

**ExistingGapAdjustValue**
Property to control automatic adjustment based on existing gaps. Default is true.

**TerminalGapAdjustValue**
Property to adjust the penalty for opening a gap at the ends of the sequence. Default is false.

**Output Arguments**

<table>
<thead>
<tr>
<th><strong>SeqsMultiAligned</strong></th>
<th>Vector of structures (same as Seqs) but with the field 'Sequence' updated with the alignment.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>When <code>Seqs</code> is a cell or char array, <code>SeqsMultiAligned</code> is a char array with the output alignment following the same order as the input.</td>
</tr>
</tbody>
</table>

**Description**

`SeqsMultiAligned = multialign(Seqs)` performs a progressive multiple alignment for a set of sequences (Seqs). Pairwise distances between sequences are computed after pairwise alignment with the Gonnet scoring matrix and then by counting the proportion of sites at which each pair of
sequences are different (ignoring gaps). The guide tree is calculated by the neighbor-joining method assuming equal variance and independence of evolutionary distance estimates.

\[ \text{SeqsMultiAligned} = \text{multialign}(\text{Seqs}, \text{Tree}) \] uses a tree (\text{Tree}) as a guide for the progressive alignment. The sequences (\text{Seqs}) should have the same order as the leaves in the tree (\text{Tree}) or use a field ('Header' or 'Name') to identify the sequences.

\text{multialign}(\ldots, \text{PropertyName}, \text{PropertyValue},\ldots) \] enters optional arguments as property name/property value pairs. Specify one or more properties in any order. Enclose each \text{PropertyName} in single quotation marks. Each \text{PropertyName} is case insensitive. These property name/property value pairs are as follows:

\text{multialign}(\ldots, \text{'Weights'}, \text{WeightsValue}) \] selects the sequence weighting method. Weights emphasize highly divergent sequences by scaling the scoring matrix and gap penalties. Closer sequences receive smaller weights.

Values of the property \text{Weights} are:

- 'THG' (default) — Thompson-Higgins-Gibson method using the phylogenetic tree branch distances weighted by their thickness.
- 'equal' — Assigns the same weight to every sequence.

\text{multialign}(\ldots, \text{'ScoringMatrix'}, \text{ScoringMatrixValue}) \] selects the scoring matrix (\text{ScoringMatrixValue}) for the progressive alignment. Match and mismatch scores are interpolated from the series of scoring matrices by considering the distances between the two profiles or sequences being aligned. The first matrix corresponds to the smallest distance, and the last matrix to the largest distance. Intermediate distances are calculated using linear interpolation.

\text{multialign}(\ldots, \text{'SMInterp'}, \text{SMInterpValue}), \text{when SMInterpValue is false}, turns off the linear interpolation of the scoring matrices. Instead, each supplied scoring matrix is assigned to a fixed range depending on the distances between the two profiles or sequences being aligned.

\text{multialign}(\ldots, \text{'GapOpen'}, \text{GapOpenValue}) \] specifies the initial penalty for opening a gap.

\text{multialign}(\ldots, \text{'ExtendGap'}, \text{ExtendGapValue}) \] specifies the initial penalty for extending a gap.

\text{multialign}(\ldots, \text{'DelayCutoff'}, \text{DelayCutoffValue}) \] specifies a threshold to delay the alignment of divergent sequences whose closest neighbor is farther than

\[ (\text{DelayCutoffValue}) \times (\text{median patristic distance between sequences}) \]

\text{multialign}(\ldots, \text{'UseParallel'}, \text{UseParallelValue}) \] specifies whether to use \text{parfor}-loops when computing the pairwise alignments. When \text{true}, and Parallel Computing Toolbox is installed and a \text{parpool} is open, computation occurs in parallel. If there are no open \text{parpool}, but automatic creation is enabled in the Parallel Preferences, the default pool will be automatically open and computation occurs in parallel. If Parallel Computing Toolbox is installed, but there are no open \text{parpool} and automatic creation is disabled, then computation uses \text{parfor}-loops in serial mode. If Parallel Computing Toolbox is not installed, then computation uses \text{parfor}-loops in serial mode. Default is \text{false}, which uses for-loops in serial mode.

\text{multialign}(\ldots, \text{'Verbose'}, \text{VerboseValue}), \text{when VerboseValue is true}, turns on verbosity.
The remaining input optional arguments are analogous to the function `profalign` on page 1-1340 and are used through every step of the progressive alignment of profiles.

`multialign(..., 'ExistingGapAdjust', ExistingGapAdjustValue),` when `ExistingGapAdjustValue` is false, turns off the automatic adjustment based on existing gaps of the position-specific penalties for opening a gap.

When `ExistingGapAdjustValue` is true, for every profile position, `profalign` proportionally lowers the penalty for opening a gap toward the penalty of extending a gap based on the proportion of gaps found in the contiguous symbols and on the weight of the input profile.

`multialign(..., 'TerminalGapAdjust', TerminalGapAdjustValue),` when `TerminalGapAdjustValue` is true, adjusts the penalty for opening a gap at the ends of the sequence to be equal to the penalty for extending a gap.

**Examples**

**Align multiple sequences**

This example shows how to align multiple protein sequences.

Use the `fastaread` function to read `p53samples.txt`, a FASTA-formatted file included with Bioinformatics Toolbox™, which contains p53 protein sequences of seven species.

```matlab
p53 = fastaread('p53samples.txt')
p53 = 7×1 struct array with fields:
  Header
  Sequence
```

Compute the pairwise distances between each pair of sequences using the 'GONNET' scoring matrix.

```matlab
dist = seqpdist(p53,'ScoringMatrix','GONNET');
dist
```

Build a phylogenetic tree using an unweighted average distance (UPGMA) method. This tree will be used as a guiding tree in the next step of progressive alignment.

```matlab
tree = seqlinkage(dist,'average',p53)
tree
```

Perform progressive alignment using the PAM family scoring matrices.

```matlab
ma = multialign(p53,tree,'ScoringMatrix',...{'pam150','pam200','pam250'})
ma
```

```matlab
ma = 7×1 struct array with fields:
  Header
  Sequence
```

```matlab
showalignment(ma)
```
Align Nucleotide Sequences

1 Enter an array of sequences.

```matlab
seqs = {'CACGTAACATCTC','ACGACGTAACATCTTCT','AAACGTAACATCTCGC'};
```

2 Promote terminations with gaps in the alignment.

```matlab
multialign(seqs,'terminalGapAdjust',true)
ans =
--CACGTAACATCTC--
ACGACGTAACATCTTCT
-AAACGTAACATCTCGC
```

3 Compare the alignment without termination gap adjustment.

```matlab
multialign(seqs)
ans =
CA--CGTAACATCT--C
Extended Capabilities

**Automatic Parallel Support**
Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true`.

For more information, see the `'UseParallel'` name-value pair argument.

See Also
align2cigar | hmmprofalign | multialignread | multialignwrite | nalign | profalign | seqconsensus | seqneighjoin | seqprofile | showalignment

Introduced before R2006a
**multialignread**

Read multiple sequence alignment file

**Syntax**

\[
S = \text{multialignread}(\text{File})
\]

\[
[\text{Headers, Sequences}] = \text{multialignread}(\text{File})
\]

\[
... = \text{multialignread}(\text{File}, '\text{IgnoreGaps}', \text{IgnoreGapsValue})
\]

**Input Arguments**

*File*  
Multiple sequence alignment file specified by one of the following:

- File name or path and file name
- URL pointing to a file
- MATLAB character array that contains the text of a multiple sequence alignment file

You can read common multiple sequence alignment file types, such as ClustalW (.aln), GCG (.msf), and PHYLIP.

*IgnoreGapsValue*  
Controls removing gap symbols, such as `-' or `'.'`, from the sequences. Choices are true or false (default).

**Output Arguments**

<table>
<thead>
<tr>
<th>S</th>
<th>MATLAB structure array containing the following fields:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Header — Header information from the file.</td>
</tr>
<tr>
<td></td>
<td>• Sequence — Amino acid or nucleotide sequences.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Headers</th>
<th>Cell array containing the header information from the file.</th>
</tr>
</thead>
</table>

| Sequences       | Cell array containing the amino acid or nucleotide sequences. |

**Description**

\[
S = \text{multialignread}(\text{File})
\]

reads a multiple sequence alignment file. The file contains multiple sequence lines that start with a sequence header followed by an optional number (not used by multialignread) and a section of the sequence. The multiple sequences are broken into blocks with the same number of blocks for every sequence. To view an example multiple sequence alignment file, type `open aagag.aln` at the MATLAB command line.

The output, `S`, is a structure array where `S.Header` contains the header information and `S.Sequence` contains the amino acid or nucleotide sequences.

\[
[\text{Headers, Sequences}] = \text{multialignread}(\text{File})
\]

reads the file into separate variables, *Headers* and *Sequences*, which are cell arrays containing header information and amino acid or nucleotide sequences, respectively.
... = multialignread(File, 'IgnoreGaps', IgnoreGapsValue) controls the removal of any gap symbol, such as '-' or '.', from the sequences. Choices are true or false (default).

**Examples**

Read a multiple sequence alignment of the gag polyprotein for several HIV strains.

```
gagaa = multialignread('aagag.aln')
gagaa =
```

1x16 struct array with fields:
- Header
- Sequence

**See Also**

fastaread | gethmmalignment | multialign | multialignwrite | seqalignviewer | seqconsensus | seqdisp | seqprofile

**Introduced before R2006a**
multialignwrite

Write multiple alignment to file

Syntax

multialignwrite(Alignment)
multialignwrite(..., 'Format', FormatValue, ...)
multialignwrite(..., 'Header', HeaderValue, ...)
multialignwrite(..., 'WriteCount', WriteCountValue, ...)

Description

multialignwrite(Alignment) writes the contents of an alignment to a ClustalW ALN-formatted (default) or MSF-formatted file.

multialignwrite(..., 'PropertyName', PropertyValue, ...) calls multialignwrite with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each PropertyValue in single quotation marks. EachPropertyName is case insensitive. These property name/property value pairs are as follows:

multialignwrite(..., 'Format', FormatValue, ...) specifies the format of the file. FormatValue can be 'ALN' (default) or 'MSF'.

multialignwrite(..., 'Header', HeaderValue, ...) specifies the first line of the file. The default for HeaderValue is 'MATLAB multiple sequence alignment'.

multialignwrite(..., 'WriteCount', WriteCountValue, ...) specifies whether to add the residue counts to the end of each line. WriteCountValue can be true (default) or false.

Input Arguments

Alignment

An alignment, such as returned by the multialign function, represented by a vector of structures, each containing the fields Header and Sequence.

File

Character vector or string specifying either a file name or a path and file name for saving the data. If you specify only a file name, the file is saved to the MATLAB Current Folder browser.

Tip If you use an .msf extension when supplying a file name for File, the data is written to an MSF-formatted file. Otherwise, the data is written to a ClustalW ALN-formatted file.

Below the columns of the ClustalW ALN-formatted file, symbols can appear that denote:
• * — Residues or nucleotides in the column are identical in all sequences in the alignment.
• : — Conserved substitutions exist in the column for all sequences in the alignment.
• . — Semiconserved substitutions exist in the column for all sequences in the alignment.

For more information on these symbols and the groups of residues considered conserved and semiconserved, see section 12 in “Changes since version 1.6” at https://web.mit.edu/seven/src/clustalw-1.82/README.

Default:

**FormatValue**

Character vector or string that specifies the format of File. Choices are 'ALN' (default) or 'MSF'.

**Tip** You can also write to an MSF-formatted file by using an .msf extension when supplying a file name for File.

Default:

**HeaderValue**

Character vector or string that specifies the first line of the file.

**Tip** Use the 'Header' property if your file header must be a specific format for a third-party software application.

Default: 'MATLAB multiple sequence alignment'

**WriteCountValue**

Specifies whether to add the residue counts to the end of each line. Choices are true (default) or false.

Default:

**Examples**

1 Use the **fastaread** function to read **p53samples.txt**, a FASTA-formatted file included with the Bioinformatics Toolbox software, which contains seven cellular tumor antigen p53 sequences.

```
p53 = fastaread('p53samples.txt')
p53 =
7x1 struct array with fields:
  Header
  Sequence
```

2 Use the **multialign** function to align the seven cellular tumor antigen p53 sequences.

```
ma = multialign(p53,'verbose',true);
```
Write the alignment to a file named p53.aln.

`multialignwrite('p53.aln', ma)`

See Also
`fastaread`, `fastawrite`, `gethmmalignment`, `multialign`, `multialignread`, `phytreewrite`, `seqalignviewer`, `seqconsensus`, `seqdisp`, `seqprofile`

Introduced in R2008b
mzcdf2peaks

Convert mzCDF structure to peak list

Syntax

\[
\text{[Peaklist, Times] = mzcdf2peaks(mzCDFStruct)}
\]

Input Arguments

\textit{mzCDFStruct}  
MATLAB structure containing information from a netCDF file, such as one created by the \texttt{mzcdfread} function. Its fields correspond to the variables and global attributes in a netCDF file. If a netCDF variable contains local attributes, an additional field is created, with the name of the field being the variable name appended with \_attributes. The number and names of the fields will vary, depending on the mass spectrometer software, but typically there are mass\_values and intensity\_values fields.

Output Arguments

\textit{Peaklist}  
Either of the following:

- Two-column matrix, where the first column contains mass/charge (m/z) values and the second column contains ion intensity values.
- Cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time.

\textit{Times}  
Scalar of vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. If \textit{Times} is a vector, the number of elements equals the number of peak lists contained in \textit{Peaklist}.

Description

\text{[Peaklist, Times] = mzcdf2peaks(mzCDFStruct)} extracts peak information from \textit{mzCDFStruct}, a MATLAB structure containing information from a netCDF file, such as one created by the \texttt{mzcdfread} function, and creates \textit{Peaklist}, a single matrix or a cell array of matrices containing mass/charge (m/z) values and ion intensity values, and \textit{Times}, a scalar or vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set.

\textit{mzCDFStruct} contains fields that correspond to the variables and global attributes in a netCDF file. If a netCDF variable contains local attributes, an additional field is created, with the name of the field being the variable name appended with \_attributes. The number and names of the fields will vary, depending on the mass spectrometer software, but typically there are mass\_values and intensity\_values fields.
Examples

In the following example, the file `results.cdf` is not provided.

1 Use the `mzcdfread` function to read a netCDF file into the MATLAB software as a structure. Then extract the peak information from the structure.

   ```matlab
   mzcdf_struct = mzcdfread('results.cdf');
   [peaks,time] = mzcdf2peaks(mzcdf_struct)
   
   peaks =
   [7008x2 single]
   [7008x2 single]
   [7008x2 single]
   [7008x2 single]
   
   time =
   8.3430
   12.6130
   16.8830
   21.1530
   
2 Create a color map containing a color for each peak list (retention time).

   ```matlab
   colors = hsv(numel(peaks));
   ```

3 Create a 3-D figure of the peaks and add labels to it.

   ```matlab
   figure
   hold on
   for i = 1:numel(peaks)
       t = repmat(time(i),size(peaks{i},1),1);
       plot3(t,peaks{i}(:,1),peaks{i}(:,2),'color',colors(i,:))
   end
   view(70,60)
   xlabel('Time')
   ylabel(mzcdf_struct.mass_axis_label)
   zlabel(mzcdf_struct.intensity_axis_label)
   ```
See Also
msdotplot | mspalign | msppresample | mzcdfread

Introduced in R2008b
mzcdfinfo

Return information about netCDF file containing mass spectrometry data

Syntax

InfoStruct = mzcdfinfo(File)

Input Arguments

| File        | Character vector or string containing a file name, or a path and file name, of a netCDF file that contains mass spectrometry data and conforms to the ANDI/MS or the ASTM E2077-00 (2005) standard specification or earlier specifications. If you specify only a file name, that file must be on the MATLAB search path or in the current folder. |

Output Arguments

| InfoStruct | MATLAB structure containing information from a netCDF file. It includes the fields in the following table. |

Description

InfoStruct = mzcdfinfo(File) returns a MATLAB structure, InfoStruct, containing summary information about a netCDF file, File.

File is a character vector or string containing a file name, or a path and file name, of a netCDF file that contains mass spectrometry data. The file must conform to the ANDI/MS or the ASTM E2077-00 (2005) standard specification or earlier specifications.

InfoStruct includes the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>Name of the netCDF file.</td>
</tr>
<tr>
<td>FileTimeStamp</td>
<td>Date time stamp of the netCDF file.</td>
</tr>
<tr>
<td>FileSize</td>
<td>Size of the file in bytes.</td>
</tr>
<tr>
<td>NumberOfScans</td>
<td>Number of scans in the file.</td>
</tr>
<tr>
<td>StartTime</td>
<td>Run start time.</td>
</tr>
<tr>
<td>EndTime</td>
<td>Run end time.</td>
</tr>
<tr>
<td>TimeUnits</td>
<td>Units for time.</td>
</tr>
<tr>
<td>GlobalMassMin</td>
<td>Minimum m/z value in all scans.</td>
</tr>
<tr>
<td>GlobalMassMax</td>
<td>Maximum m/z value in all scans.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlobalIntensityMin</td>
<td>Minimum intensity value in all scans.</td>
</tr>
<tr>
<td>GlobalIntensityMax</td>
<td>Maximum intensity value in all scans.</td>
</tr>
<tr>
<td>ExperimentType</td>
<td>Indicates if data is raw or centroided.</td>
</tr>
</tbody>
</table>

**Note** If any of the associated attributes are not in the netCDF file (because they are optional in the specifications), the value for that field will be set to N/A or NaN.

### Examples

In the following example, the file `results.cdf` is not provided.

Return a MATLAB structure containing summary information about a netCDF file.

```matlab
info = mzcdfinfo('results.cdf')
```

```matlab
info =
    Filename: 'results.cdf'
    FileTimeStamp: '19930703134354-700'
    FileSize: 339892
    NumberOfScans: 4
    StartTime: 8.3430
    EndTime: 21.1530
    TimeUnits: 'N/A'
    GlobalMassMin: 399.9990
    GlobalMassMax: 1.8000e+003
    GlobalIntensityMin: NaN
    GlobalIntensityMax: NaN
    ExperimentType: 'Continuum Mass Spectrum'
```

### See Also

`mzcdfread`

introduced in R2008b
**mzCdfread**

Read mass spectrometry data from netCDF file

**Syntax**

\[
mzCDFStruct = mzCdfread(File)
\]

\[
mzCDFStruct = mzCdfread(File, ...'TimeRange', TimeRangeValue, ...)
\]

\[
mzCDFStruct = mzCdfread(File, ...'ScanIndices', ScanIndicesValue, ...)
\]

\[
mzCDFStruct = mzCdfread(File, ...'Verbose', VerboseValue, ...)
\]

**Input Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File</strong></td>
<td>Character vector or string containing a file name, or a path and file name, of a netCDF file that contains mass spectrometry data and conforms to the ANDI/MS or the ASTM E2077-00 (2005) standard specification or earlier specifications. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.</td>
</tr>
<tr>
<td><strong>TimeRangeValue</strong></td>
<td>Two-element numeric array ([Start _ End]) that specifies the time range in File for which to read spectra. Default is to read spectra from all times ([0 \ Inf]). Tip Time units are indicated in the netCDF global attributes. For summary information about the time ranges in a netCDF file, use the mzcdfinfo function.</td>
</tr>
<tr>
<td><strong>ScanIndicesValue</strong></td>
<td>Positive integer, vector of integers, or a two-element numeric array ([Start_Ind \ End_Ind]) that specifies a scan, multiple scans, or a range of scans in File to read. Start_Ind and End_Ind are each positive integers indicating a scan index number. Start_Ind must be less than End_Ind. Default is to read all scans. Tip For information about the scan indices in a netCDF file, check the NumberOfScans field in the structure returned by the mzcdfinfo function. Note If you specify a ScanIndicesValue, you cannot specify a TimeRangeValue.</td>
</tr>
<tr>
<td><strong>VerboseValue</strong></td>
<td>Controls the display of the progress of the reading of File. Choices are true (default) or false.</td>
</tr>
</tbody>
</table>
Output Arguments

| mzCDFStruct | MATLAB structure containing mass spectrometry information from a netCDF file. Its fields correspond to the variables and global attributes in a netCDF file. If a netCDF variable contains local attributes, an additional field is created, with the name of the field being the variable name appended with _attributes. The number and names of the fields will vary, depending on the mass spectrometer software, but typically there are mass_values and intensity_values fields. |

Description

*mzCDFStruct = mzdcdread(File)* reads a netCDF file, *File*, and then creates a MATLAB structure, *mzCDFStruct*.

*File* is a character vector or string containing a file name, or a path and file name, of a netCDF file that contains mass spectrometry data. The file must conform to the ANDI/MS or the ASTM E2077-00 (2005) standard specification or earlier specifications.

*mzCDFStruct* contains fields that correspond to the variables and global attributes in a netCDF file. If a netCDF variable contains local attributes, an additional field is created, with the name of the field being the variable name appended with _attributes. The number and names of the fields will vary, depending on the mass spectrometer software, but typically there are mass_values and intensity_values fields.

**Tip** LC/MS data analysis requires extended amounts of memory from the operating system.

- If you receive errors related to memory, try the following:
  - Increase the virtual memory (swap space) for your operating system as described in “Resolve “Out of Memory” Errors” (MATLAB).
  - If you receive errors related to Java heap space, increase your Java heap space:
    - If you have MATLAB version 7.10 (R2010a) or later, see “Java Heap Memory Preferences” (MATLAB).
    - If you have MATLAB version 7.9 (R2009b) or earlier, see https://www.mathworks.com/support/solutions/en/data/1-18I2C/.

*mzCDFStruct = mzdcdread(File, ...'PropertyName', PropertyValue, ...)* calls *mzdcdread* with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

- *mzCDFStruct = mzdcdread(File, ...'TimeRange', TimeRangeValue, ...)* specifies the range of time in *File* to read. *TimeRangeValue* is a two-element numeric array [Start End]. Default is to read spectra from all times [0 Inf].

**Tip** Time units are indicated in the netCDF global attributes. For summary information about the time ranges in a netCDF file, use the *mzdcdinfo* function.
**Note** If you specify a *TimeRangeValue*, you cannot specify *ScanIndicesValue*.

\[
mzCDFStruct = \text{mzcdfread}(	ext{File}, \ldots \text{'}ScanIndices\text{', }\text{ScanIndicesValue}, \ldots)\]

specifies a scan, multiple scans, or range of scans in *File* to read. *ScanIndicesValue* is a positive integer, vector of integers, or a two-element numeric array [\text{Start\_Ind} \text{ End\_Ind}]. \text{Start\_Ind} and \text{End\_Ind} are each positive integers indicating a scan index number. \text{Start\_Ind} must be less than \text{End\_Ind}. Default is to read all scans.

**Tip** For information about the scan indices in a netCDF file, check the *NumberOfScans* field in the structure returned by the *mzcdfinfo* function.

**Note** If you specify a *ScanIndicesValue*, you cannot specify a *TimeRangeValue*.

\[
mzCDFStruct = \text{mzcdfread}(	ext{File}, \ldots \text{'}Verbose\text{', }\text{VerboseValue}, \ldots)\]

controls the progress display when reading *File*. Choices are true (default) or false.

**Examples**

In the following example, the file *results.cdf* is not provided.

1. Read a netCDF file into the MATLAB software as a structure.

   \[
   \text{out} = \text{mzcdfread}('results.cdf');
   \]

2. View the second scan in the netCDF file by creating separate variables containing the intensity and m/z values, and then plotting these values. Add a title and x- and y-axis labels using fields in the output structure.

   \[
   \text{idx1} = \text{out.scan\_index}(2)+1;
   \text{idx2} = \text{out.scan\_index}(3);
   \text{y} = \text{out.intensity\_values}(\text{idx1} : \text{idx2});
   \text{z} = \text{out.mass\_values}(\text{idx1} : \text{idx2});
   \text{stem(z,y,'marker','none')}
   \]

   \[
   \text{title(sprintf('Time: %f',out.scan\_acquisition\_time(2)))}
   \text{xlabel(out.mass\_axis\_units)}
   \text{ylabel(out.intensity\_axis\_units)}
   \]
See Also
jcampread | mzcdf2peaks | mzcdfinfo | mzxmlread | tgspcread

Introduced in R2008b
**mzxml2peaks**

Convert mzXML structure to peak list

**Syntax**

\[
[\text{Peaklist}, \text{Times}] = \text{mzxml2peaks}(	ext{mzXMLStruct})
\]

\[
[\text{Peaklist}, \text{Times}] = \text{mzxml2peaks}(	ext{mzXMLStruct}, 'Levels', \text{LevelsValue})
\]

**Input Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mzXMLStruct</code></td>
<td>MATLAB structure containing information from an mzXML file, such as one created by the <code>mzxmlread</code> function. It includes the fields shown in the table below.</td>
</tr>
<tr>
<td><code>LevelsValue</code></td>
<td>Positive integer or vector of integers that specifies the level(s) of spectra in <code>mzXMLStruct</code> to convert, assuming the spectra are from tandem MS data sets. Default is 1, which converts only the first-level spectra, that is, spectra containing precursor ions. Setting <code>LevelsValue</code> to 2 converts only the second-level spectra, which are the fragment spectra (created from a precursor ion).</td>
</tr>
</tbody>
</table>

**Output Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Peaklist</code></td>
<td>Either of the following:</td>
</tr>
<tr>
<td></td>
<td>• Two-column matrix, where the first column contains mass/charge (m/z) values and the second column contains ion intensity values.</td>
</tr>
<tr>
<td></td>
<td>• Cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time.</td>
</tr>
<tr>
<td><code>Times</code></td>
<td>Vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. The number of elements in <code>Times</code> equals the number of elements in <code>Peaklist</code>.</td>
</tr>
</tbody>
</table>

**Description**

\[
[\text{Peaklist}, \text{Times}] = \text{mzxml2peaks}(	ext{mzXMLStruct})
\]

extracts peak information from `mzXMLStruct`, a MATLAB structure containing information from an mzXML file, such as one created by the `mzxmlread` function, and creates `Peaklist`, a cell array of matrices containing mass/charge (m/z) values and ion intensity values, and `Times`, a vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. `mzXMLStruct` includes the following fields:
### Field | Description
--- | ---
scan | Structure array containing the data pertaining to each individual scan, such as mass spectrometry level, total ion current, polarity, precursor mass (when it applies), and the spectrum data.
index | Structure containing indices to the positions of scan elements in the XML document.
mzXML | Structure containing:
• Information in the root element of the mzXML schema, such as instrument details, experiment details, and preprocessing method
• URLs pointing to schemas for the individual scans
• Indexing approach
• Digital signature calculated for the current instance of the document

$[\text{Peaklist, Times}] = \text{mzxml2peaks}(\text{mzXMLStruct, 'Levels', LevelsValue})$ specifies the level(s) of the spectra in $\text{mzXMLStruct}$ to convert, assuming the spectra are from tandem MS data sets. Default is 1, which converts only the first-level spectra, that is, spectra containing precursor ions. Setting $\text{LevelsValue}$ to 2 converts only the second-level spectra, which are the fragment spectra (created from a precursor ion).

### Examples

**Note** In the following example, the file `results.mzxml` is not provided. Sample mzXML files can be found at:

- The Sashimi Project
- Peptide Atlas Repository at the Institute for Systems Biology (ISB)

1. Use the `mzxmlread` function to read an mzXML file into the MATLAB software as structure. Then extract the peak information of only the first-level ions from the structure.

   ```matlab
   mzxml_struct = mzxmlread('results.mzxml');
   [peaks,time] = mzxml2peaks(mzxml_struct);
   ```

2. Create a dot plot of the LC/MS data.

   ```matlab
   msdotplot(peaks,time)
   ```

### See Also

`msdotplot` | `mspalign` | `msppresample` | `mzxmlread`

**Introduced in R2007a**
mzxmlinfo

Return information about mzXML file

Syntax

InfoStruct = mzxmlinfo(File)

InfoStruct = mzxmlinfo(File, 'NumOfLevels', NumOfLevelsValue)

Input Arguments

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Character vector or string containing a file name, or a path and file name, of an mzXML file that conforms to the mzXML 2.1 specification or earlier specifications. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.</td>
</tr>
<tr>
<td>NumOfLevelsValue</td>
<td>Controls the return of NumOfLevels, an additional field in InfoStruct, that contains the number of mass spectrometry (MS) levels of spectra in File. Choices are true or false (default).</td>
</tr>
</tbody>
</table>

Output Arguments

| InfoStruct        | MATLAB structure containing information from an mzXML file. It includes the fields shown in the table below. |

Description

InfoStruct = mzxmlinfo(File) returns a MATLAB structure, InfoStruct, containing summary information about an mzXML file, File.

File is a character vector or string containing a file name, or a path and file name, of an mzXML file. The file must conform to the mzXML 2.1 specification or earlier specifications. You can view the mzXML 2.1 specification at:


InfoStruct includes the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>Name of the mzXML file.</td>
</tr>
<tr>
<td>FileModDate</td>
<td>Modification date of the file.</td>
</tr>
<tr>
<td>FileSize</td>
<td>Size of the file in bytes.</td>
</tr>
<tr>
<td>NumberOfScans</td>
<td>Number of scans in the file.*</td>
</tr>
<tr>
<td>StartTime</td>
<td>Run start time.*</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndTime</td>
<td>Run end time.*</td>
</tr>
<tr>
<td>DataProcessingIntensityCutoff</td>
<td>Minimum mass/charge (m/z) intensity value.*</td>
</tr>
<tr>
<td>DataProcessingCentroided</td>
<td>Indicates if data is centroided.*</td>
</tr>
<tr>
<td>DataProcessingDeisotoped</td>
<td>Indicates if data is deisotoped.*</td>
</tr>
<tr>
<td>DataProcessingChargeDeconvoluted</td>
<td>Indicates if data is deconvoluted.*</td>
</tr>
<tr>
<td>DataProcessingSpotIntegration</td>
<td>For LC/MALDI experiments, indicates if peaks eluting over multiple spots have been integrated into a single spot.*</td>
</tr>
</tbody>
</table>

* — These fields contain N/A if the mzXML file does not include the associated attributes. The associated attributes are optional in the mzXML file, per the mzXML 2.1 specification.

**InfoStruct = mzxmlinfo(File, 'NumOfLevels', NumOfLevelsValue)** controls the return of NumOfLevels, an additional field in mzXMLInfo, that contains the number of mass spectrometry levels of spectra in File. Choices are true or false (default).

### Examples

**Note** In the following example, the file results.mzxml is not provided. Sample mzXML files can be found at:

- The Sashimi Project
- Peptide Atlas Repository at the Institute for Systems Biology (ISB)

Return a MATLAB structure containing summary information about an mzXML file.

```matlab
info = mzxmlinfo('results.mzxml');
info =
    Filename: 'results.mzxml'
    FileModDate: '07-May-2008 13:39:12'
    FileSize: 10607
    NumberOfScans: 2
    StartTime: 'PT0.00683333S'
    EndTime: 'PT200.036S'
    DataProcessingIntensityCutoff: 'N/A'
    DataProcessingCentroided: 'false'
    DataProcessingDeisotoped: 'N/A'
    DataProcessingChargeDeconvoluted: 'N/A'
    DataProcessingSpotIntegration: 'N/A'
```

Return a MATLAB structure containing summary information, including the number of mass spectrometry levels, about an mzXML file.

```matlab
info = mzxmlinfo('results.mzxml','numoflevels',true);
```
info =

    Filename: 'results.mzxml'
    FileModDate: '07-May-2008 13:39:12'
    FileSize: 10607
    NumberOfScans: 2
    StartTime: 'PT0.00683333S'
    EndTime: 'PT200.036S'
    DataProcessingIntensityCutoff: 'N/A'
    DataProcessingCentroided: 'false'
    DataProcessingDeisotoped: 'N/A'
    DataProcessingChargeDeconvoluted: 'N/A'
    DataProcessingSpotIntegration: 'N/A'
    NumberOfMSLevels: 2

See Also
mzxmlread

Introduced in R2008b
mzxmlread

Read data from mzXML file

Syntax

mzXMLStruct = mzxmlread(myFile)
mzXMLStruct = mzxmlread(myFile,Name,Value)

Description

mzXMLStruct = mzxmlread(myFile) returns a MATLAB structure, mzXMLStruct, from an mzXML file, myFile.

mzXMLStruct = mzxmlread(myFile,Name,Value) reads an mzXML file, myFile, and then returns a MATLAB structure, mzXMLStruct, using additional options specified by one or more Name,Value pair arguments.

Examples

Create a MATLAB Structure from an mzXML File

In this example, the file results_1.mzxml is not provided. You can find sample mzXML files at:

- The Sashimi Project
- Peptide Atlas Repository at the Institute for Systems Biology (ISB)

Read an mzXML file into a MATLAB structure.

out = mzxmlread('results_1.mzxml')

out =

    scan: [2000x1 struct]
    mzXML: [1x1 struct]
    index: [1x1 struct]

View the first scan in the mzXML file by creating separate variables containing the mass-to-charge ratio (mz_ratio) and intensity (Y) values respectively. Then plot these values.

mz_ratio = out.scan(1).peaks.mz(1:2:end);
Y = out.scan(1).peaks.mz(2:2:end);
stem(mz_ratio,Y,'marker','none')
Extract One or Multiple Scans from an mzXML Structure

In this example, the file `results_2.mzxml` is not provided. You can find sample mzXML files at:

- The Sashimi Project
- Peptide Atlas Repository at the Institute for Systems Biology (ISB)

Read an mzXML file into a MATLAB structure, extracting a scan at index 1000.

```matlab
out1 = mzxmlread('results_2.mzxml','ScanIndices',1000)
```

```matlab
time = [1 2 3 4 5];
out1 =
    scan: [1x1 struct]
    mzXML: [1x1 struct]
    index: [1x1 struct]
```

Read an mzXML file into a MATLAB structure, extracting multiple scans at indices 1000, 1500, and 2000.

```matlab
out2 = mzxmlread('results_2.mzxml','ScanIndices',[1000 1500 2000])
```

```matlab
out2 =
    scan: [3x1 struct]
    mzXML: [1x1 struct]
    index: [1x1 struct]
```

Read an mzXML file into a MATLAB structure, extracting a range of scans from indices 1000 to 2000.

```matlab
out3 = mzxmlread('results_2.mzxml','ScanIndices',[1000:2000])
```
out3 =
    scan: [1001x1 struct]
mzXML: [1x1 struct]
index: [1x1 struct]

**Input Arguments**

myFile — Input file  
character vector | string

Input file, specified as a character vector or string containing an mzXML file name. The file must conform to the mzXML 2.1 or earlier specifications. You can read the mzXML 2.1 specification here:


**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: ‘Levels’,3,’TimeRange’,[5.0 10.0]

**Levels — Spectra levels**  
positive integer | vector of integers

Spectra levels, specified as a positive integer or vector of integers indicating which scans to extract scans from myFile. By default, mzxmlread reads all spectra levels.

For summary information about the levels of spectra in an mzXML file, use the mzxmlinfo function.

If you are using the 'Levels' name-value pair argument, then you cannot use 'TimeRange' or 'ScanIndices'.

Example: ‘Levels’,5

**TimeRange — Range of time**  
Two-element numeric array

Range of time, specified as a two-element numeric array, such as [Start End] indicating which scans to extract from myFile. The Start and End scalar values must be between the startTime and endTime attributes of the msRun element in myFile. The Start scalar value must be less than End. By default, mzxmlread reads all scans.

For summary information about the time ranges in an mzXML file, use the mzxmlinfo function.

If you are using 'TimeRange' name-value pair argument, then you cannot use 'Levels' or 'ScanIndices'.

Example: ‘TimeRange’,[5.1 10.2]

**ScanIndices — Scan indices**  
positive integer | vector of positive integers
Scan indices, specified as a positive integer or vector of positive integers indicating which scans to extract from `myFile`. Use an integer to specify a single scan, or a vector of integers to specify multiple scans. By default, `mzxmlread` reads all scans.

For summary information about the time ranges in an mzXML file, use the `mzxmlinfo` function.

If you are using the `ScanIndices` name-value pair argument, then you cannot use `Levels` or `TimeRange`.

Example: `ScanIndices`, 7000

**Verbose** — Verbose mode
```matlab
type (default) | 1 | false | 0
```

Verbose mode, specified as `true` (1), or `false` (0). When `Verbose` is set to `true`, `mzxmlread` displays the progress while reading `myFile`.

Example: `Verbose`, `false`

## Output Arguments

**mzXMLStruct** — Structure from mzXML file

MATLAB structure

Structure from an mzXML file, returned as a MATLAB structure. `mzXMLStruct` has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scan</code></td>
<td>Structure array containing the data pertaining to each individual scan, such as mass spectrometry level, total ion current, polarity, precursor mass (when it applies), and the spectrum data.</td>
</tr>
<tr>
<td><code>index</code></td>
<td>Structure containing indices to the positions of scan elements in the XML document.</td>
</tr>
<tr>
<td><code>mzXML</code></td>
<td>Structure containing all of the following:</td>
</tr>
<tr>
<td></td>
<td>• Information in the root element of the mzXML schema, such as instrument details, experiment details, and preprocessing methods</td>
</tr>
<tr>
<td></td>
<td>• URLs pointing to schemas for each scan</td>
</tr>
<tr>
<td></td>
<td>• Indexing approach</td>
</tr>
<tr>
<td></td>
<td>• Digital signature calculated for the current instance of the document</td>
</tr>
</tbody>
</table>

## Tips

LC/MS data analysis requires extended amounts of memory from the operating system.

- If you receive errors related to memory, try the following:
  
  - Increase the virtual memory (swap space) for your operating system as described in “Resolve “Out of Memory” Errors” (MATLAB).

- If you receive errors related to Java heap space, increase your Java heap space:
  
  - If you have MATLAB version 7.10 (R2010a) or later, see “Java Heap Memory Preferences” (MATLAB).
• If you have MATLAB version 7.9 (R2009b) or earlier, see https://www.mathworks.com/support/solutions/en/data/1-18I2C/.

See Also
jcampread | mxml2peaks | mxmlinfo | tgspcread | xmlread

Introduced in R2006b
nbintest

Unpaired hypothesis test for count data with small sample sizes

Syntax

test = nbintest(X,Y)
test = nbintest(X,Y,Name,Value)

Description

test = nbintest(X,Y) performs a hypothesis test that two independent samples of short-read count data, in each row of X and Y, come from distributions with equal means under the assumptions that:

• Short-read counts are modeled using the negative binomial distribution.
• Variance and mean of data in each row are linked through a regression function along all the rows.

X and Y must have the same number of rows and at least 2 columns, but not necessarily the same number of columns. Rows of X and Y correspond to variables, features, or genes, such as measurements of gene expression for different genes. Columns are usually time points or patients.

test is a NegativeBinomialTest object with two-sided p-values stored in the pValue property.

Use this function when you want to perform an unpaired hypothesis test for short-read count data (from high-throughput assays such as RNA-Seq or ChIP-Seq) with small sample sizes (in the order of tens at most). For instance, use this function to decide if observed differences in read counts between two conditions are significant for given genes.

test = nbintest(X,Y,Name,Value) uses additional options specified by one or more Name,Value pair arguments.

Note It is recommended that you use the diagnostic plots of the NegativeBinomialTest object returned by nbintest before interpreting the p-values. These plots allow you to see if the model assumption is correct, and the variance link used is appropriate for the data.

Examples

Perform unpaired hypothesis test for short-read count data

This example shows how to perform an unpaired hypothesis test for synthetic short-read count data from two different biological conditions.

The data in this example contains synthetic gene count data for 5000 genes, representing two different biological conditions, such as diseased and normal cells. For each condition, there are five samples. Only 10% of the genes (500 genes) are differentially expressed. Specifically, half of them (250 genes) are exactly 3-fold overexpressed. The other 250 genes are 3-fold underexpressed.
The rest of the gene expression data is generated from the same negative binomial distribution for both conditions. Each sample also has a different size factor (that is, the coverage or sampling depth).

Load the data.

```matlab
clear all
load('nbintest_data.mat','K','H0');
```

The variable `K` contains gene count data. The rows represent genes, and the columns represent samples. In this case, the first five columns represent samples from the first condition. The other five columns represent samples from the second condition. Display the first few rows of `K`.

```matlab
K(1:5,:) ans = 5×10
13683 14140 8281 14309 12208 8045 9446 11317
16028 16805 9813 16486 14076 9901 10927 13348
814 862 492 910 758 521 573 753 936
15870 16453 9857 16454 14267 9671 10997 13624
9422 9393 5734 9598 8174 5381 6315 9869 9795
```

In this example, the null hypothesis is true when the gene is not differentially expressed. The variable `H0` contains boolean indicators that indicate for which genes the null hypothesis is true (marked as 1). In other words, `H0` contains known labels that you will use later to compare with predicted results.

```matlab
sum(H0)
```

```matlab
ans = 4500
```

Out of 5000 genes, 4500 are not differentially expressed in this synthetic data.

Run an unpaired hypothesis test for samples from two conditions using `nbintest`. The assumption is that the data came from a negative binomial distribution, where the variance is linked to the mean via a locally-regressed smooth function of the mean as described in [1] by setting `'VarianceLink'` to `'LocalRegression'`.

```matlab
tLocal = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','LocalRegression');
```

Use `plotVarianceLink` to plot a scatter plot for each experimental condition (for X and Y conditions), with the sample variance on the common scale versus the estimate of the condition-dependent mean. Use a linear scale for both axes. Include curves for all other linkage options by setting `'Compare'` to true.

```matlab
plotVarianceLink(tLocal,'Scale','linear','Compare',true)
```
The Identity line represents the Poisson model, where the variance is identical to the mean as described in [3]. Observe that the data seems to be overdispersed (that is, most points are above the Identity line). The Constant line represents the negative binomial model, where the variance is the sum of the shot noise term (mean) and a constant multiplied by the squared mean as described in [2]. The Local Regression and Constant linkage options appear to fit better with the overdispersed data.

Use plotChiSquaredFit to assess the goodness-of-fit for variance regression. It plots the empirical CDF (ecdf) of the chi-squared probabilities. The probabilities are the ratio between the observed and the estimated variance stratified by short-read count levels into five equal-sized bins.

plotChiSquaredFit(tLocal)
Each figure shows five ecdf curves. Each curve represents one of the five short-read count levels. For instance, the blue line represents the ecdf curve for a low short-read counts between 0 and 1264. The red line represents high counts (more than 11438).

One way to interpret the curves is to check if the ecdf curves are above the diagonal line. If they are above the line, then the variance is overestimated. If they are below the line, then the variance is underestimated. In both figures, the variance seems to be correctly estimated for higher counts (that is, the red line follows the diagonal line), but slightly overestimated for lower count levels.

To assess the performance of the hypothesis test, construct a confusion matrix using the known labels and the predicted p-values.

```matlab
confusionmat(H0,(tLocal.pValue > .001))
```

```
ans = 2x2

493    7
 5  4495
```

Out of 500 differentially expressed genes, 493 are correctly predicted (true positives) and 7 of them are incorrectly predicted as not-differentially expressed genes (false negatives). Out of 4500 genes that are not differentially expressed, 4495 are correctly predicted (true negatives) and 5 of them are incorrectly predicted as differentially expressed genes (false positives).

For a comparison, run the hypothesis test again assuming that counts are modeled by the Poisson distribution, where the variance is identical to the mean.
tPoisson = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','Identity');

Plot the ecdf curves. Observe that all the curves are below the diagonal line, implying that the variance is underestimated. Therefore, the negative binomial model fits the data better.

plotChiSquaredFit(tPoisson)
Input Arguments

X — Gene expression values from the first experimental condition
matrix | table

Gene expression values from the first experimental condition, specified as a matrix or table. For instance, X can represent gene expression values from cancer cells.

**Note** X and Y must have the same number of rows and at least 2 columns, but not necessarily the same number of columns. Rows of X and Y correspond to genes (or features), such as measurements of gene expression for different genes. Columns are usually time points or patients.

Y — Gene expression values from the second experimental condition
matrix | table

Gene expression values from the second experimental condition, specified as a matrix or table. For instance, Y can represent gene expression values from normal cells.

**Note** X and Y must have the same number of rows and at least 2 columns, but not necessarily the same number of columns. Rows of X and Y correspond to genes (or features), such as measurements of gene expression for different genes. Columns are usually time points or patients.
**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: 'VarianceLink','Identity' specifies that the variance is equal to the mean when defining the linkage between the two.

**VarianceLink — Linkage type between the variance and mean**

'LocalRegression' (default) | 'Constant' | 'Identity'

Linkage type between the variance and mean, specified as a comma-separated pair consisting of 'VarianceLink' and a character vector or string. This table summarizes the available linkage options.

<table>
<thead>
<tr>
<th>Linkage Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'LocalRegression'</td>
<td>The variance is the sum of the shot noise term (mean) and a locally regressed nonparametric smooth function of the mean as described in [1]. This option is the default. Use this option if your data is overdispersed and has more than 1000 rows (genes).</td>
</tr>
<tr>
<td>'Constant'</td>
<td>The variance is the sum of the shot noise term (mean) and a constant multiplied by the squared mean as described in [2]. This method uses all the rows in the data to estimate the constant. Use this option if your data is overdispersed and has less than 1000 rows.</td>
</tr>
<tr>
<td>'Identity'</td>
<td>The variance is equal to the mean as described in [3]. Counts are therefore modeled by the Poisson distribution individually for each row of X and Y. Use this option if your data has few genes and the regression between the variance and mean is not possible because of very small number of samples or replicates. This option is not recommended for overdispersed data.</td>
</tr>
</tbody>
</table>

Example: 'VarianceLink','Constant'

**PooledVariance — Logical flag to pool variance across both conditions**

false (default) | true

Logical flag to pool variance across both conditions, specified as true or false. By default, the variance is estimated separately for each condition.

Example: 'PooledVariance',true

**SizeFactor — Size (scaling) factor of each column in X and Y**

[] (default) | cell array of two vectors

Size (scaling) factor of each column in X and Y, specified as a cell array of two vectors such as {SX,SY}. SX and SY are numeric vectors with sizes equal to size(X,2) and size(Y,2). SX, SY, or both can be a scalar indicating that all columns share the same size factor.

In a high-throughput sequencing library, the size factor is an estimation of the coverage or the sampling depth. The default is an empty array [], meaning the size factor is estimated as the median of the ratio of the sample’s counts to the geometric mean of each row in X or Y. Rows with zero geometric mean are ignored.

Example: 'SizeFactor',{[1.2,0.5,0.8],[0.8,1.1,1.5]}
Output Arguments

test — Hypothesis test results
NegativeBinomialTest object

Hypothesis test results, returned as a NegativeBinomialTest object. Use this object to create
diagnostic plots and access p-values.

References

Genome Biology, 11(10):R106.


Assessment of Technical Reproducibility and Comparison with Gene Expression Arrays.
Genome Research, 16:1509-1517.

See Also
NegativeBinomialTest | mattest | plotChiSquaredFit | plotVarianceLink

Topics
“Negative Binomial Distribution” (Statistics and Machine Learning Toolbox)

Introduced in R2014b
**NegativeBinomialTest**

Unpaired hypothesis test result

**Description**

A `NegativeBinomialTest` object, returned by the `nbintest` function, contains the results of an unpaired hypothesis test for short-read count data with small sample sizes. Use this object to access p-values of the test or to create diagnostic plots.

**Creation**

`nbintest` returns the unpaired hypothesis test result as a `NegativeBinomialTest` object. You cannot construct this object directly.

**Properties**

- **pValue — Two-sided p-values**
  column vector

  Two-sided p-values, specified as a column vector, for every row of the inputs to `nbintest`.

- **VarianceLink — Linkage type between the variance and mean**
  'LocalRegression' (default) | 'Constant' | 'Identity'

  This property is read-only.

  Linkage type between the variance and mean, specified as a character vector or string. This table summarizes the available linkage options.

<table>
<thead>
<tr>
<th>Linkage Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'LocalRegression'</td>
<td>The variance is the sum of the shot noise term (mean) and a locally regressed nonparametric smooth function of the mean as described in [1]. This option is the default. Use this option if your data contains several rows (genes), such as more than 1000 rows.</td>
</tr>
<tr>
<td>'Constant'</td>
<td>The variance is the sum of the shot noise term (mean) and a constant multiplied by the squared mean as described in [2]. This method uses all the rows in the data to estimate the constant. Use this option if your data has fewer rows, that is, less than 1000 rows, and is overdispersed.</td>
</tr>
<tr>
<td>'Identity'</td>
<td>The variance is equal to the mean as described in [3]. Counts are therefore modeled by the Poisson distribution individually for each row of X and Y. Use this option to compare the results of the other two options.</td>
</tr>
</tbody>
</table>

- **PooledVariance — Logical flag to pool variance between both conditions**
  0 (default) | 1

  This property is read-only.
Logical flag to pool variance between both conditions, specified as 1 (true) or 0 (false). The default is 0, meaning the variance is estimated separately for each condition.

**SizeFactors** — Size (scaling) factor of each column in X and Y

cell array of two vectors

This property is read-only.

Size (scaling) factor of each column in X and Y, specified as a cell array of two vectors, such as `{SX,SY}`. SX and SY are numeric vectors with sizes equal to `size(X,2)` and `size(Y,2)`.

**Note** These properties are read-only. Run `nbintest` to change them.

**Object Functions**

- `plotVarianceLink`  Plot the sample variance versus the estimate of the condition-dependent mean
- `plotChiSquaredFit`  Plot goodness-of-fit for variance regression

**Examples**

**Perform unpaired hypothesis test for short-read count data**

This example shows how to perform an unpaired hypothesis test for synthetic short-read count data from two different biological conditions.

The data in this example contains synthetic gene count data for 5000 genes, representing two different biological conditions, such as diseased and normal cells. For each condition, there are five samples. Only 10% of the genes (500 genes) are differentially expressed. Specifically, half of them (250 genes) are exactly 3-fold overexpressed. The other 250 genes are 3-fold underexpressed. The rest of the gene expression data is generated from the same negative binomial distribution for both conditions. Each sample also has a different size factor (that is, the coverage or sampling depth).

Load the data.

```matlab
clear all
load('nbintest_data.mat','K','H0');
```

The variable `K` contains gene count data. The rows represent genes, and the columns represent samples. In this case, the first five columns represent samples from the first condition. The other five columns represent samples from the second condition. Display the first few rows of `K`.

```matlab
K(1:5,:)
ans = 5×10
    13683     14140     8281    14309    12208     8045    9446   11317
    16028    16805     9813    16486    14076    9901    10927   13348
     814     862     492     910     758     521     573     753
   15870    16453     9857    16454    14267    9671    10997   13624
    9422    9393     5734    9598    8174    5381    6315    7752
```

In this example, the null hypothesis is true when the gene is not differentially expressed. The variable `H0` contains boolean indicators that indicate for which genes the null hypothesis is true (marked as 1). In other words, `H0` contains known labels that you will use later to compare with predicted results.
Out of 5000 genes, 4500 are not differentially expressed in this synthetic data.

Run an unpaired hypothesis test for samples from two conditions using `nbintest`. The assumption is that the data came from a negative binomial distribution, where the variance is linked to the mean via a locally-regressed smooth function of the mean as described in [1] by setting `'VarianceLink'` to `'LocalRegression'`.

```matlab
htLocal = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','LocalRegression');
```

Use `plotVarianceLink` to plot a scatter plot for each experimental condition (for X and Y conditions), with the sample variance on the common scale versus the estimate of the condition-dependent mean. Use a linear scale for both axes. Include curves for all other linkage options by setting `'Compare'` to `true`.

```matlab
plotVarianceLink(htLocal,'Scale','linear','Compare',true)
```
The **Identity** line represents the Poisson model, where the variance is identical to the mean as described in [3]. Observe that the data seems to be overdispersed (that is, most points are above the **Identity** line). The **Constant** line represents the negative binomial model, where the variance is the sum of the shot noise term (mean) and a constant multiplied by the squared mean as described in [2]. The **Local Regression** and **Constant** linkage options appear to fit better with the overdispersed data.

Use `plotChiSquaredFit` to assess the goodness-of-fit for variance regression. It plots the empirical CDF (ecdf) of the chi-squared probabilities. The probabilities are the ratio between the observed and the estimated variance stratified by short-read count levels into five equal-sized bins.

`plotChiSquaredFit(tLocal)`
Each figure shows five ecdf curves. Each curve represents one of the five short-read count levels. For instance, the blue line represents the ecdf curve for a low short-read counts between 0 and 1264. The red line represents high counts (more than 11438).

One way to interpret the curves is to check if the ecdf curves are above the diagonal line. If they are above the line, then the variance is overestimated. If they are below the line, then the variance is underestimated. In both figures, the variance seems to be correctly estimated for higher counts (that is, the red line follows the diagonal line), but slightly overestimated for lower count levels.

To assess the performance of the hypothesis test, construct a confusion matrix using the known labels and the predicted p-values.

\[
\text{confusionmat}(\text{H0}, (\text{tLocal.pValue} > .001))
\]

\[
\begin{array}{cc}
493 & 7 \\
5 & 4495 \\
\end{array}
\]

Out of 500 differentially expressed genes, 493 are correctly predicted (true positives) and 7 of them are incorrectly predicted as not-differentially expressed genes (false negatives). Out of 4500 genes that are not differentially expressed, 4495 are correctly predicted (true negatives) and 5 of them are incorrectly predicted as differentially expressed genes (false positives).

For a comparison, run the hypothesis test again assuming that counts are modeled by the Poisson distribution, where the variance is identical to the mean.
tPoisson = nbintest(K(:,1:5),K(:,6:10), 'VarianceLink', 'Identity');

Plot the ecdf curves. Observe that all the curves are below the diagonal line, implying that the variance is underestimated. Therefore, the negative binomial model fits the data better.

plotChiSquaredFit(tPoisson)
References


See Also
mattest | nbintest

Topics
“Negative Binomial Distribution” (Statistics and Machine Learning Toolbox)

Introduced in R2014b
**ndims (DataMatrix)**

Return number of dimensions in DataMatrix object

**Syntax**

\[ N = \text{ndims}(DMObj) \]

**Input Arguments**

| DMObj          | DataMatrix object, such as created by DataMatrix (object constructor). |

**Output Arguments**

| N              | Positive integer representing the number of dimensions in DMObj. The number of dimensions in a DataMatrix object is always 2. |

**Description**

\[ N = \text{ndims}(DMObj) \] returns the number of dimensions in DMObj, a DataMatrix object. The number of dimensions in a DataMatrix object is always 2.

**See Also**

DataMatrix

**Topics**

DataMatrix object on page 1-532

**Introduced in R2008b**


**ne (DataMatrix)**

Test DataMatrix objects for inequality

**Syntax**

\[
T = \text{ne}(\text{DMObj1, DMObj2}) \\
T = \text{DMObj1} \sim= \text{DMObj2} \\
T = \text{ne}(\text{DMObj1, B}) \\
T = \text{DMObj1} \sim= B \\
T = \text{ne}(\text{B, DMObj1}) \\
T = B \sim= \text{DMObj1}
\]

**Input Arguments**

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>MATLAB numeric or logical array.</td>
</tr>
</tbody>
</table>

**Output Arguments**

| T               | Logical matrix of the same size as DMObj1 and DMObj2 or DMObj1 and B. It contains logical 1 (true) where elements in the first input are not equal to the corresponding element in the second input, and logical 0 (false) when they are equal. |

**Description**

\[
T = \text{ne}(\text{DMObj1, DMObj2}) \text{ or the equivalent } T = \text{DMObj1} \sim= \text{DMObj2} \text{ compares each element in DataMatrix object } \text{DMObj1} \text{ to the corresponding element in DataMatrix object } \text{DMObj2}, \text{ and returns } T, \text{ a logical matrix of the same size as } \text{DMObj1} \text{ and } \text{DMObj2}, \text{ containing logical 1 (true) where elements in } \text{DMObj1} \text{ are not equal to the corresponding element in } \text{DMObj2}, \text{ and logical 0 (false) when they are equal. DMObj1 and DMObj2 must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). DMObj1 and DMObj2 can have different Name properties.} \\
\]

\[
T = \text{ne}(\text{DMObj1, B}) \text{ or the equivalent } T = \text{DMObj1} \sim= B \text{ compares each element in DataMatrix object } \text{DMObj1} \text{ to the corresponding element in } \text{B}, \text{ a numeric or logical array, and returns } T, \text{ a logical matrix of the same size as } \text{DMObj1} \text{ and } \text{B}, \text{ containing logical 1 (true) where elements in } \text{DMObj1} \text{ are not equal to the corresponding element in } \text{B}, \text{ and logical 0 (false) when they are equal. DMObj1 and B must have the same size (number of rows and columns), unless one is a scalar.} \\
\]

\[
T = \text{ne}(\text{B, DMObj1}) \text{ or the equivalent } T = \text{B} \sim= \text{DMObj1} \text{ compares each element in } \text{B}, \text{ a numeric or logical array, to the corresponding element in DataMatrix object } \text{DMObj1}, \text{ and returns } T, \text{ a logical matrix of the same size as } \text{B} \text{ and } \text{DMObj1}, \text{ containing logical 1 (true) where elements in } \text{B} \text{ are not equal to the corresponding element in } \text{DMObj1}, \text{ and logical 0 (false) when they are equal. B and DMObj1 must have the same size (number of rows and columns), unless one is a scalar.} \\
\]

MATLAB calls \(T = \text{ne}(X, Y)\) for the syntax \(T = X \sim= Y\) when \(X\) or \(Y\) is a DataMatrix object.
See Also
DataMatrix | eq

Topics
DataMatrix object on page 1-532

Introduced in R2008b
ngsbrowser

(To be removed) Open NGS Browse to visualize and explore alignments

**Note** ngsbrowser will be removed in a future release. Use Genomics Viewer instead.

**Syntax**

```matlab
ngsbrowser
ngsbrowser(bmObj)
```

**Description**

ngsbrowser opens the NGS Browser app.

```matlab
ngsbrowser(bmObj)
```
imports the BioMap object `bmObj` and opens it in the NGS Browser app.

**Input Arguments**

`bmObj` — Object containing sequence information or short-read sequence alignment data

BioMap object

Object containing sequence information or short-read sequence alignment data, specified as a BioMap object.

**Examples**

**View Sequence Data from BioMap object in NGS Browser**

Create a BioMap object from a SAM-formatted file.

```matlab
b = BioMap('ex1.sam');
```

Display the object in the NGS Browser.

```matlab
ngsbrowser(b)
```

**Tips**

- Use the NGS Browser to compare the alignment of multiple data sets to a common reference sequence.
- Use the NGS Browser to investigate regions of interest in the short-read alignment determined by various analyses, such as RNA-Seq, ChIP-Seq, and genetic variation analyses.
Compatibility Considerations

`ngsbrowser` will be removed
Warns starting in R2019b

`ngsbrowser` will be removed in a future release. Use `Genomics Viewer` instead.

See Also

**Functions**
BioMap | `seqalignviewer` | `seqviewer`

**Apps**
Genomics Viewer

**Topics**
“Visualize and Investigate Sequence Read Alignments”

**External Websites**
NCBI Genome Database

Introduced in R2011b
**nmercount**

Count n-mers in nucleotide or amino acid sequence

**Syntax**

\[
Nmer = nmercount(Seq, Length)
\]

\[
Nmer = nmercount(Seq, Length, C)
\]

**Input Arguments**

<table>
<thead>
<tr>
<th>Seq</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Character vector or string specifying a nucleotide sequence or amino acid sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers or the table Mapping Amino Acid Letter Codes to Integers.</td>
</tr>
<tr>
<td></td>
<td>- MATLAB structure containing a <code>Sequence</code> field that contains a nucleotide sequence or an amino acid sequence, such as returned by <code>fastaread</code>, <code>fastqread</code>, <code>emblread</code>, <code>getembl</code>, <code>genbankread</code>, <code>getgenbank</code>, <code>getgenpept</code>, <code>genpeptread</code>, <code>getpdb</code>, or <code>pdbread</code>.</td>
</tr>
</tbody>
</table>

| Length  | Integer specifying the length of n-mer to count. |

**Output Arguments**

| Nmer | Cell array containing the n-mer counts in `Seq`. |

**Description**

\[
Nmer = nmercount(Seq, Length)
\]

counts the n-mers or patterns of a specific length in `Seq`, a nucleotide sequence or amino acid sequence, and returns the n-mer counts in a cell array.

\[
Nmer = nmercount(Seq, Length, C)
\]

returns only the n-mers with cardinality of at least `C`.

**Examples**

1. Use the `getgenpept` function to retrieve the amino acid sequence for the human insulin receptor.

   ```matlab
   S = getgenpept('AAA59174','SequenceOnly',true);
   ```

2. Count the number of four-mers in the amino acid sequence and display the first 20 rows in the cell array.

   ```matlab
   nmers = nmercount(S,4);
nmers(1:20,:)
   ans =
   'APES'  [2]
   ```
See Also
aaccount | basecount | codoncount | dimercount

Introduced before R2006a
nt2aa

Convert nucleotide sequence to amino acid sequence

Syntax

```
SeqAA = nt2aa(SeqNT)

SeqAA = nt2aa(..., 'Frame', FrameValue, ...)
SeqAA = nt2aa(..., 'GeneticCode', GeneticCodeValue, ...)
SeqAA = nt2aa(..., 'AlternativeStartCodons', AlternativeStartCodonsValue, ...)
SeqAA = nt2aa(..., 'ACGTOnly', ACGTOnlyValue, ...)
```

Input Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| SeqNT    | One of the following:  
  
  • Character vector or string containing single-letter codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers.  
  
  • Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes.  
  
  • MATLAB structure containing a Sequence field that contains a nucleotide sequence, such as returned by fastaread, fastqread, emblread, getembl, genbankread, or getgenbank.  

  **Note** Hyphens are valid only if the codon to which it belongs represents a gap, that is, the codon contains all hyphens. Example: ACT---TGA

  **Tip** Do not use a sequence with hyphens if you specify ‘all’ for **FrameValue**.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| FrameValue | Integer, character vector, or string specifying a reading frame in the nucleotide sequence. Choices are 1, 2, 3, or ‘all’. Default is 1.  
  
  If **FrameValue** is ‘all’, then **SeqAA** is a 3-by-1 cell array. |
**Output Arguments**

| SeqAA          | Amino acid sequence specified by a character vector of single-letter codes. |

**Description**

\[ SeqAA = \text{nt2aa}(\text{SeqNT}) \]

converts a nucleotide sequence, specified by \( \text{SeqNT} \), to an amino acid sequence, returned in \( \text{SeqAA} \), using the standard genetic code.

\[ SeqAA = \text{nt2aa}(\text{SeqNT}, \ldots, '\text{PropertyName}', \text{PropertyValue}, \ldots) \]

calls `nt2aa` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

- `FrameValue`: Controls the translation of alternative codons.
  Choices are `true` (default) or `false`.

  If `true`, then the function errors if any of these characters are present.
  If `false`, then the function tries to resolve ambiguities. If it cannot, it returns X for the affected codon.

- `GeneticCodeValue`: Controls the behavior of ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, and N) and unknown characters. `ACGTOnlyValue` can be `true` (default) or `false`.

  - If `true`, then the function errors if any of these characters are present.
  - If `false`, then the function tries to resolve ambiguities. If it cannot, it returns X for the affected codon.

**Tip** If you use a code name, you can truncate the name to the first two letters of the name.
default, AlternativeStartCodonsValue is set to true, and if the first codon of a sequence is a known alternative start codon, the codon is translated to methionine.

If this option is set to false, then an alternative start codon at the start of a sequence is translated to its corresponding amino acid in the genetic code that you specify, which might not necessarily be methionine. For example, in the human mitochondrial genetic code, AUA and AUU are known to be alternative start codons. For more information on alternative start codons, visit https://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=t#SG1.

For more information about alternative start codons, see:
www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=t#SG1

Genetic Code

<table>
<thead>
<tr>
<th>Code Number</th>
<th>Code Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Standard</td>
</tr>
<tr>
<td>2</td>
<td>Vertebrate Mitochondrial</td>
</tr>
<tr>
<td>3</td>
<td>Yeast Mitochondrial</td>
</tr>
<tr>
<td>4</td>
<td>Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma/Spiroplasma</td>
</tr>
<tr>
<td>5</td>
<td>Invertebrate Mitochondrial</td>
</tr>
<tr>
<td>6</td>
<td>Ciliate, Dasycladacean, and Hexamita Nuclear</td>
</tr>
<tr>
<td>9</td>
<td>Echinoderm Mitochondrial</td>
</tr>
<tr>
<td>10</td>
<td>Euplotid Nuclear</td>
</tr>
<tr>
<td>11</td>
<td>Bacterial and Plant Plastid</td>
</tr>
<tr>
<td>12</td>
<td>Alternative Yeast Nuclear</td>
</tr>
<tr>
<td>13</td>
<td>Ascidian Mitochondrial</td>
</tr>
<tr>
<td>14</td>
<td>Flatworm Mitochondrial</td>
</tr>
<tr>
<td>15</td>
<td>Blepharisma Nuclear</td>
</tr>
<tr>
<td>16</td>
<td>Chlorophycean Mitochondrial</td>
</tr>
<tr>
<td>21</td>
<td>Trematode Mitochondrial</td>
</tr>
<tr>
<td>22</td>
<td>Scenedesmus Obliquus Mitochondrial</td>
</tr>
<tr>
<td>23</td>
<td>Thraustochytrium Mitochondrial</td>
</tr>
</tbody>
</table>
### Standard Genetic Code

<table>
<thead>
<tr>
<th>Amino Acid Name</th>
<th>Amino Acid Code</th>
<th>Nucleotide Codon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alanine</td>
<td>A</td>
<td>GCT GCC GCA GCG</td>
</tr>
<tr>
<td>Arginine</td>
<td>R</td>
<td>CGT CGC CGA CGG AGA AGG</td>
</tr>
<tr>
<td>Asparagine</td>
<td>N</td>
<td>AAT AAC</td>
</tr>
<tr>
<td>Aspartic acid (Aspartate)</td>
<td>D</td>
<td>GAT GAC</td>
</tr>
<tr>
<td>Cysteine</td>
<td>C</td>
<td>TGT TGC</td>
</tr>
<tr>
<td>Glutamine</td>
<td>Q</td>
<td>CAA CAG</td>
</tr>
<tr>
<td>Glutamic acid (Glutamate)</td>
<td>E</td>
<td>GAA GAG</td>
</tr>
<tr>
<td>Glycine</td>
<td>G</td>
<td>GGT GGC GGA GGG</td>
</tr>
<tr>
<td>Histidine</td>
<td>H</td>
<td>CAT CAC</td>
</tr>
<tr>
<td>Isoleucine</td>
<td>I</td>
<td>ATT ATC ATA</td>
</tr>
<tr>
<td>Leucine</td>
<td>L</td>
<td>TTA TTG CTT CTC CTA CTG</td>
</tr>
<tr>
<td>Lysine</td>
<td>K</td>
<td>AAA AAG</td>
</tr>
<tr>
<td>Methionine</td>
<td>M</td>
<td>ATG</td>
</tr>
<tr>
<td>Phenylalanine</td>
<td>F</td>
<td>TTT TTC</td>
</tr>
<tr>
<td>Proline</td>
<td>P</td>
<td>CCT CCC CCA CCG</td>
</tr>
<tr>
<td>Serine</td>
<td>S</td>
<td>TCT TCC TCA TCG AGT AGC</td>
</tr>
<tr>
<td>Threonine</td>
<td>T</td>
<td>ACT ACC ACA ACG</td>
</tr>
<tr>
<td>Tryptophan</td>
<td>W</td>
<td>TGG</td>
</tr>
<tr>
<td>Tyrosine</td>
<td>Y</td>
<td>TAT, TAC</td>
</tr>
<tr>
<td>Valine</td>
<td>V</td>
<td>GTT GTC GTA GTG</td>
</tr>
<tr>
<td>Asparagine or Aspartic acid (Aspartate)</td>
<td>B</td>
<td>Random codon from D and N</td>
</tr>
<tr>
<td>Glutamine or Glutamic acid (Glutamate)</td>
<td>Z</td>
<td>Random codon from E and Q</td>
</tr>
<tr>
<td>Unknown amino acid (any amino acid)</td>
<td>X</td>
<td>Random codon</td>
</tr>
<tr>
<td>Translation stop</td>
<td>*</td>
<td>TAA TAG TGA</td>
</tr>
<tr>
<td>Gap of indeterminate length</td>
<td>-</td>
<td>---</td>
</tr>
<tr>
<td>Unknown character (any character or symbol not in table)</td>
<td>?</td>
<td>???</td>
</tr>
</tbody>
</table>

`SeqAA = nt2aa(..., 'ACGTOnly', ACGTOnlyValue, ...)` controls the behavior of ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, and N) and unknown characters. `ACGTOnlyValue` can be `true` (default) or `false`. If `true`, then the function errors if any of these characters are present. If `false`, then the function tries to resolve ambiguities. If it cannot, it returns X for the affected codon.
Examples

Example 1.53. Converting the ND1 Gene

1 Use the `getgenbank` function to retrieve genomic information for the human mitochondrion from the GenBank database and store it in a MATLAB structure.

```matlab
mitochondria = getgenbank('NC_012920')
```

```matlab
mitochondria =
    LocusName: 'NC_012920'
    LocusSequenceLength: '16569'
    LocusNumberOfStrands: ''
    LocusTopology: 'circular'
    LocusMoleculeType: 'DNA'
    LocusGenBankDivision: 'PRI'
    LocusModificationDate: '05-MAR-2010'
    Definition: 'Homo sapiens mitochondrion, complete genome.'
    Accession: 'NC_012920 AC_000021'
    Version: 'NC_012920.1'
    GI: '251831106'
    Project: []
    DBLink: 'Project:30353'
    Keywords: []
    Segment: []
    Source: 'mitochondrion Homo sapiens (human)'
    SourceOrganism: [4x65 char]
    Reference: {1x7 cell}
    Comment: [24x67 char]
    Features: [933x74 char]
    CDS: [1x13 struct]
    Sequence: [1x16569 char]
    SearchURL: [1x70 char]
    RetrieveURL: [1x104 char]
```

2 Determine the name and location of the first gene in the human mitochondrion.

```matlab
mitochondria.CDS(1).gene
ans =
    ND1
mitochondria.CDS(1).location
ans =
    3307..4262
```

3 Extract the sequence for the ND1 gene from the nucleotide sequence.

```matlab
ND1gene = mitochondria.Sequence(3307:4262);
```

4 Convert the ND1 gene on the human mitochondria genome to an amino acid sequence using the Vertebrate Mitochondrial genetic code.

```matlab
protein1 = nt2aa(ND1gene,'GeneticCode', 2);
```

5 Use the `getgenpept` function to retrieve the same amino acid sequence from the GenPept database.
protein2 = getgenpept('YP_003024026', 'SequenceOnly', true);
6 Use the isequal function to compare the two amino acid sequences.
isequal (protein1, protein2)
ans =
    1

Example 1.54. Converting the ND2 Gene

1 Use the getgenbank function to retrieve the nucleotide sequence for the human mitochondrion from the GenBank database.
mitochondria = getgenbank('NC_012920');
2 Determine the name and location of the second gene in the human mitochondrion.
mitochondria.CDS(2).gene
ans =
    ND2
mitochondria.CDS(2).location
ans =
    4470..5511
3 Extract the sequence for the ND2 gene from the nucleotide sequence.
ND2gene = mitochondria.Sequence(4470:5511);
4 Convert the ND2 gene on the human mitochondria genome to an amino acid sequence using the Vertebrate Mitochondrial genetic code.
protein1 = nt2aa(ND2gene,'GeneticCode', 2);

Note In the ND2gene nucleotide sequence, the first codon is ATT, which is translated to M, while the subsequent ATT codons are translated to I. If you set 'AlternativeStartCodons' to false, then the first ATT codon is translated to I, the corresponding amino acid in the Vertebrate Mitochondrial genetic code.

5 Use the getgenpept function to retrieve the same amino acid sequence from the GenPept database.
protein2 = getgenpept('YP_003024027', 'SequenceOnly', true);
6 Use the isequal function to compare the two amino acid sequences.
isequal (protein1, protein2)
ans =
    1

Example 1.55. Converting a Sequence with Ambiguous Characters

If you have a sequence with ambiguous or unknown nucleotide characters, you can set the 'ACGTOnly' property to false to have the nt2aa function try to resolve them:
nt2aa('agttgccgacgcgcncar','ACGTOonly', false)
ans =
SCRRAQ

See Also
aa2nt | aminolookup | baselookup | codonbias | dnds | dndsml | geneticcode | isotopicdist | revgeneticcode | seqviewer

Introduced before R2006a
nt2int

Convert nucleotide sequence from letter to integer representation

Syntax

\[ \text{SeqInt} = \text{nt2int}(\text{SeqChar}) \]

\[ \text{SeqInt} = \text{nt2int}(\text{SeqChar}, \ldots\text{'Unknown'}, \text{UnknownValue}, \ldots) \]

\[ \text{SeqInt} = \text{nt2int}(\text{SeqChar}, \ldots\text{'ACGTOnly'}, \text{ACGTOnlyValue}, \ldots) \]

Input Arguments

<table>
<thead>
<tr>
<th>SeqChar</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers. Integers are arbitrarily assigned to IUB/IUPAC letters.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that contains a nucleotide sequence, such as returned by fastaread, fastqread, emblread, getembl, genbankread, or getgenbank.</td>
</tr>
</tbody>
</table>

| UnknownValue     | Integer to represent unknown nucleotides. Choices are integers \( \geq 0 \) and \( \leq 255 \). Default is 0. |

| ACGTOnlyValue    | Controls the prohibition of ambiguous nucleotides. Choices are true or false (default). If ACGTOnlyValue is true, you can enter only the characters A, C, G, T, and U. |

Output Arguments

| SeqInt           | Nucleotide sequence specified by a row vector of integers. |

Description

\[ \text{SeqInt} = \text{nt2int}(\text{SeqChar}) \] converts \( \text{SeqChar} \), a character vector or string specifying a nucleotide sequence, to \( \text{SeqInt} \), a row vector of integers specifying the same nucleotide sequence. For valid codes, see the table Mapping Nucleotide Letter Codes to Integers. Unknown characters (characters not in the table) are mapped to 0. Gaps represented with hyphens are mapped to 16.

\[ \text{SeqInt} = \text{nt2int}(\text{SeqChar}, \ldots\text{'PropertyName'}, \text{PropertyValue}, \ldots) \] calls nt2int with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

\[ \text{SeqInt} = \text{nt2int}(\text{SeqChar}, \ldots\text{'Unknown'}, \text{UnknownValue}, \ldots) \] specifies an integer to represent unknown nucleotides. UnknownValue can be an integer \( \geq 0 \) and \( \leq 255 \). Default is 0.
\( \text{SeqInt} = \text{nt2int}(\text{SeqChar}, ..., 'ACGTOOnly', \text{ACGTOOnlyValue}, ...) \) controls the prohibition of ambiguous nucleotides (N, R, Y, K, M, S, W, B, D, H, and V). Choices are true or false (default). If \( \text{ACGTOOnlyValue} \) is true, you can enter only the characters A, C, G, T, and U.

### Mapping Nucleotide Letter Codes to Integers

<table>
<thead>
<tr>
<th>Nucleotide</th>
<th>Code</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adenosine</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>Cytidine</td>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>Guanine</td>
<td>G</td>
<td>3</td>
</tr>
<tr>
<td>Thymidine</td>
<td>T</td>
<td>4</td>
</tr>
<tr>
<td>Uridine (if 'Alphabet' set to 'RNA')</td>
<td>U</td>
<td>4</td>
</tr>
<tr>
<td>Purine (A or G)</td>
<td>R</td>
<td>5</td>
</tr>
<tr>
<td>Pyrimidine (T or C)</td>
<td>Y</td>
<td>6</td>
</tr>
<tr>
<td>Keto (G or T)</td>
<td>K</td>
<td>7</td>
</tr>
<tr>
<td>Amino (A or C)</td>
<td>M</td>
<td>8</td>
</tr>
<tr>
<td>Strong interaction (3 H bonds) (G or C)</td>
<td>S</td>
<td>9</td>
</tr>
<tr>
<td>Weak interaction (2 H bonds) (A or T)</td>
<td>W</td>
<td>10</td>
</tr>
<tr>
<td>Not A (C or G or T)</td>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>Not C (A or G or T)</td>
<td>D</td>
<td>12</td>
</tr>
<tr>
<td>Not G (A or C or T)</td>
<td>H</td>
<td>13</td>
</tr>
<tr>
<td>Not T or U (A or C or G)</td>
<td>V</td>
<td>14</td>
</tr>
<tr>
<td>Any nucleotide (A or C or G or T or U)</td>
<td>N</td>
<td>15</td>
</tr>
<tr>
<td>Gap of indeterminate length</td>
<td>-</td>
<td>16</td>
</tr>
<tr>
<td>Unknown (any character not in table)</td>
<td>*</td>
<td>0 (default)</td>
</tr>
</tbody>
</table>

### Examples

**Example 1.56. Converting a Simple Sequence**

Convert a nucleotide sequence from letters to integers.

```matlab
s = nt2int('ACTGCTAGC')
```

\[ s = \begin{pmatrix} 1 & 2 & 4 & 3 & 2 & 4 & 1 & 3 & 2 \end{pmatrix} \]

**Example 1.57. Converting a Random Sequence**

1. Create a random character vector to represent a nucleotide sequence.

```matlab
SeqChar = randseq(20)
```

```matlab
SeqChar = TTATGACGTATTCATCTTT
```

2. Convert the nucleotide sequence from letter to integer representation.
SeqInt = nt2int(SeqChar)

SeqInt =

Columns 1 through 13
   4    4    1    4    3    1    2    3    4    4    1    4    4

Columns 14 through 20
   2    4    1    2    4    4    4

See Also
aa2int | baselookup | int2aa | int2nt

Introduced before R2006a
ntdensity

Plot density of nucleotides along sequence

Syntax

ntdensity(SeqNT)
Density = ntdensity(SeqNT)

... = ntdensity(..., 'Window', WindowValue, ...)
[Density, HighCG] = ntdensity(..., 'CGThreshold', CGThresholdValue, ...)

Arguments

<table>
<thead>
<tr>
<th>SeqNT</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers.</td>
</tr>
<tr>
<td></td>
<td>• Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that contains a nucleotide sequence, such as returned by emblread, fastaread, fastqread, genbankread, getembl, or getgenbank.</td>
</tr>
</tbody>
</table>

Note Although you can submit a sequence with nucleotides other than A, C, G, and T, ntdensity plots only A, C, G, and T.

<table>
<thead>
<tr>
<th>WindowValue</th>
<th>Value that specifies the window length for the density calculation. Default is length(SeqNT)/20.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGThresholdValue</td>
<td>Controls the return of indices for regions where the CG content of SeqNT is greater than CGThresholdValue. Default is 5.</td>
</tr>
</tbody>
</table>

Description

ntdensity(SeqNT) plots the density of nucleotides A, C, G, and T in sequence SeqNT.

Density = ntdensity(SeqNT) returns a MATLAB structure with the density of nucleotides A, C, G, and T.

... = ntdensity(SeqNT, ...'PropertyName', PropertyValue, ...) calls ntdensity with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = ntdensity(..., 'Window', WindowValue, ...) uses a window of length WindowValue for the density calculation. Default WindowValue is length(SeqNT)/20.
[Density, HighCG] = ntdensity(..., 'CGThreshold', CGThresholdValue, ...) returns indices for regions where the CG content of SeqNT is greater than CGThresholdValue. Default CGThresholdValue is 5.

Examples

1  Create a random character vector to represent a nucleotide sequence.
   
   s = randseq(1000, 'alphabet', 'dna');

2  Plot the density of nucleotides along the sequence.
   
   ntdensity(s)

See Also

basecount | codoncount | cpgisland | dimercount | filter

Introduced before R2006a
**nuc44**

Return NUC44 scoring matrix for nucleotide sequences

**Syntax**

`ScoringMatrix = nuc44`

`[ScoringMatrix, MatrixInfo] = nuc44`

**Description**

`ScoringMatrix = nuc44` returns the scoring matrix. The `nuc44` scoring matrix uses ambiguous nucleotide codes and probabilities rounded to the nearest integer.

Scale = 0.277316

Expected score = -1.7495024, Entropy = 0.5164710 bits

Lowest score = -4, Highest score = 5

Order: A C G T R Y K M S W B D H V N

`[ScoringMatrix, MatrixInfo] = nuc44` returns a structure with information about the matrix with fields Name and Order.

**Note** The NUC44 scoring matrix is supplied by NCBI and is used by the BLAST suite of programs. For more information, see ftp://ftp.ncbi.nih.gov/blast/matrices/.

**See Also**

blosum | dayhoff | gonnet | localalign | nalign | pam | swalign

**Introduced before R2006a**
num2goid

Convert numbers to Gene Ontology IDs

Syntax

\[ \text{GOIDs} = \text{num2goid}(X) \]

Description

\[ \text{GOIDs} = \text{num2goid}(X) \] converts the numbers in \( X \) to a cell array of character vectors with Gene Ontology IDs. IDs are seven-digit numbers preceded by the prefix \( \text{GO:} \), which is the standard used by the Gene Ontology database.

Examples

Get the Gene Ontology IDs of the following numbers.

\[
t = [5575 \ 5622 \ 5623 \ 5737 \ 5840 \ 30529 \ 43226 \ 43228 \ 43229 \ 43232 \ 43234];
\]
\[
\text{id} = \text{num2goid}(t)
\]
Columns 1 through 4

\[
\text{'GO:0005575'} \quad \text{'GO:0005622'} \quad \text{'GO:0005623'} \quad \text{'GO:0005737'}
\]
Columns 5 through 8

\[
\text{'GO:0005840'} \quad \text{'GO:0030529'} \quad \text{'GO:0043226'} \quad \text{'GO:0043228'}
\]
Columns 9 through 11

\[
\text{'GO:0043229'} \quad \text{'GO:0043232'} \quad \text{'GO:0043234'}
\]

See Also

geneont | geneont | getancestors | getdescendants | getmatrix | getrelatives | goannotread

Introduced before R2006a
**numel (DataMatrix)**

Return number of elements in DataMatrix object

**Syntax**

\[
N = numel(DMObj) \\
Ns = numel(DMObj, Index1, Index2)
\]

**Input Arguments**

<table>
<thead>
<tr>
<th><strong>DMObj</strong></th>
<th>DataMatrix object, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index1</strong></td>
<td>A row or range of rows in DMObj specified by a positive integer or a range using the format (x:y), where (x) is the first row and (y) is the last row.</td>
</tr>
<tr>
<td><strong>Index2</strong></td>
<td>A column or range of columns in DMObj specified by a positive integer or a range using the format (x:y), where (x) is the first column and (y) is the last column.</td>
</tr>
</tbody>
</table>

**Output Arguments**

<table>
<thead>
<tr>
<th><strong>N</strong></th>
<th>Positive integer representing the number of elements in DMObj, a DataMatrix object.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ns</strong></td>
<td>Positive integer representing the number of subscripted elements in DMObj, a DataMatrix object.</td>
</tr>
</tbody>
</table>

**Description**

\(N = numel(DMObj)\) returns 1. To find the number of elements in DMObj, a DataMatrix object, use either of the following syntaxes:

\[
\text{prod(size(DMObj))} \\
numel(DMObj, ':',':')
\]

\(Ns = numel(DMObj, Index1, Index2)\) returns the number of subscripted elements in DMObj, a DataMatrix object. Index1 specifies a row or range of rows in DMObj. Index2 specifies a column or range of columns in DMObj.

**See Also**

DataMatrix

**Topics**

DataMatrix object on page 1-532

**Introduced in R2008b**
nwalign

Globally align two sequences using Needleman-Wunsch algorithm

Syntax

Score = nwalign(Seq1,Seq2)
[Score, Alignment] = nwalign(Seq1,Seq2)
[Score, Alignment, Start] = nwalign(Seq1,Seq2)

... = nwalign(Seq1,Seq2, ...'Alphabet', AlphabetValue, ...)
... = nwalign(Seq1,Seq2, ...'ScoringMatrix', ScoringMatrixValue, ...)
... = nwalign(Seq1,Seq2, ...'Scale', ScaleValue, ...)
... = nwalign(Seq1,Seq2, ...'GapOpen', GapOpenValue, ...)
... = nwalign(Seq1,Seq2, ...'ExtendGap', ExtendGapValue, ...)
... = nwalign(Seq1,Seq2, ...'Glocal', GlocalValue, ...)
... = nwalign(Seq1,Seq2, ...'Showscore', ShowscoreValue, ...)

Input Arguments

<table>
<thead>
<tr>
<th>Seq1, Seq2</th>
<th>Amino acid or nucleotide sequences. Enter any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string of letters representing amino acids or nucleotides, such as returned by int2aa or int2nt</td>
</tr>
<tr>
<td></td>
<td>• Vector of integers representing amino acids or nucleotides, such as returned by aa2int or nt2int</td>
</tr>
<tr>
<td></td>
<td>• Structure containing a Sequence field</td>
</tr>
<tr>
<td>AlphabetValue</td>
<td>Character vector or string specifying the type of sequence. Choices are 'AA' (default) or 'NT'.</td>
</tr>
</tbody>
</table>

Tip For help with letter and integer representations of amino acids and nucleotides, see Amino Acid Lookup or Nucleotide Lookup.
<table>
<thead>
<tr>
<th><strong>ScoringMatrixValue</strong></th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Character vector or string specifying the scoring matrix to use for the global alignment. Choices for amino acid sequences are:</td>
</tr>
<tr>
<td></td>
<td>- 'BLOSUM62'</td>
</tr>
<tr>
<td></td>
<td>- 'BLOSUM30' increasing by 5 up to 'BLOSUM90'</td>
</tr>
<tr>
<td></td>
<td>- 'BLOSUM100'</td>
</tr>
<tr>
<td></td>
<td>- 'PAM10' increasing by 10 up to 'PAM500'</td>
</tr>
<tr>
<td></td>
<td>- 'DAYHOFF'</td>
</tr>
<tr>
<td></td>
<td>- 'GONNET'</td>
</tr>
<tr>
<td></td>
<td>Default is:</td>
</tr>
<tr>
<td></td>
<td>- 'BLOSUM50' — When <em>AlphabetValue</em> equals 'AA'</td>
</tr>
<tr>
<td></td>
<td>- 'NUC44' — When <em>AlphabetValue</em> equals 'NT'</td>
</tr>
</tbody>
</table>

**Note** The above scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the 'Scale' property to specify an additional scale factor to convert the output score from bits to another unit.

- Matrix representing the scoring matrix to use for the global alignment, such as returned by the `blosum`, `pam`, `dayhoff`, `gonnet`, or `nuc44` function.

**Note** If you use a scoring matrix that you created or was created by one of the above functions, the matrix does not include a scale factor. The output score will be returned in the same units as the scoring matrix. You can use the 'Scale' property to specify a scale factor to convert the output score to another unit.

**Note** If you need to compile `nwalign` into a stand-alone application or software component using MATLAB Compiler, use a matrix instead of a character vector or string for `ScoringMatrixValue`.
**ScaleValue**
Positive value that specifies a scale factor that is applied to the output score.

For example, if the output score is initially determined in bits, and you enter \( \log(2) \) for **ScaleValue**, then `nwalign` returns **Score** in nats.

Default is 1, which does not change the units of the output score.

**Note** If the 'ScoringMatrix' property also specifies a scale factor, then `nwalign` uses it first to scale the output score, then applies the scale factor specified by **ScaleValue** to rescale the output score.

**GapOpenValue**
Positive value specifying the penalty for opening a gap in the alignment. Default is 8.

**ExtendGapValue**
Positive value specifying the penalty for extending a gap using the affine gap penalty scheme.

**Note** If you specify this value, `nwalign` uses the affine gap penalty scheme, that is, it scores the first gap using the **GapOpenValue** and scores subsequent gaps using the **ExtendGapValue**. If you do not specify this value, `nwalign` scores all gaps equally, using the **GapOpenValue** penalty.

**GlocalValue**
Controls the return of a semiglobal or “glocal” alignment. In a semiglobal alignment, gap penalties at the end of the sequences are null. Choices are `true` or `false` (default).

**ShowscoreValue**
Controls the display of the scoring space and the winning path of the alignment. Choices are `true` or `false` (default).

### Output Arguments

<table>
<thead>
<tr>
<th><strong>Score</strong></th>
<th>Optimal global alignment score in bits.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alignment</strong></td>
<td>3-by-N character array showing the two sequences, <em>Seq1</em> and <em>Seq2</em>, in the first and third rows, and symbols representing the optimal global alignment for them in the second row.</td>
</tr>
<tr>
<td><strong>Start</strong></td>
<td>2-by-1 vector of indices indicating the starting point in each sequence for the alignment. Because this is a global alignment, <strong>Start</strong> is always [1;1].</td>
</tr>
</tbody>
</table>

### Description

Score = `nwalign(Seq1,Seq2)` returns the optimal global alignment score in bits. The scale factor used to calculate the score is provided by the scoring matrix.
[Score, Alignment] = nwalign(Seq1,Seq2) returns a 3-by-N character array showing the two sequences, Seq1 and Seq2, in the first and third rows, and symbols representing the optimal global alignment for them in the second row. The symbol | indicates amino acids or nucleotides that match exactly. The symbol : indicates amino acids or nucleotides that are related as defined by the scoring matrix (nonmatches with a zero or positive scoring matrix value).

[Score, Alignment, Start] = nwalign(Seq1,Seq2) returns a 2-by-1 vector of indices indicating the starting point in each sequence for the alignment. Because this is a global alignment, Start is always [1;1].

... = nwalign(Seq1,Seq2, ...'PropertyName', PropertyValue, ...) calls nwalign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = nwalign(Seq1,Seq2, ...'Alphabet', AlphabetValue, ...) specifies the type of sequences. Choices are 'AA' (default) or 'NT'.

... = nwalign(Seq1,Seq2, ...'ScoringMatrix', ScoringMatrixValue, ...) specifies the scoring matrix to use for the global alignment. Default is:

• 'BLOSUM50' — When AlphabetValue equals 'AA'
• 'NUC44' — When AlphabetValue equals 'NT'

... = nwalign(Seq1,Seq2, ...'Scale', ScaleValue, ...) specifies a scale factor that is applied to the output score, thereby controlling the units of the output score. Choices are any positive value.

... = nwalign(Seq1,Seq2, ...'GapOpen', GapOpenValue, ...) specifies the penalty for opening a gap in the alignment. Choices are any positive value. Default is 8.

... = nwalign(Seq1,Seq2, ...'ExtendGap', ExtendGapValue, ...) specifies the penalty for extending a gap using the affine gap penalty scheme. Choices are any positive value.

... = nwalign(Seq1,Seq2, ...'Glocal', GlocalValue, ...) controls the return of a semiglobal or “glocal” alignment. In a semiglobal alignment, gap penalties at the end of the sequences are null. Choices are true or false (default).

... = nwalign(Seq1,Seq2, ...'Showscore', ShowscoreValue, ...) controls the display of the scoring space and winning path of the alignment. Choices are true or false (default).
The scoring space is a heat map displaying the best scores for all the partial alignments of two sequences. The color of each \((n_1,n_2)\) coordinate in the scoring space represents the best score for the pairing of subsequences \(\text{Seq1}(1:n_1)\) and \(\text{Seq2}(1:n_2)\), where \(n_1\) is a position in \(\text{Seq1}\) and \(n_2\) is a position in \(\text{Seq2}\). The best score for a pairing of specific subsequences is determined by scoring all possible alignments of the subsequences by summing matches and gap penalties.

The winning path is represented by black dots in the scoring space, and it illustrates the pairing of positions in the optimal global alignment. The color of the last point (lower right) of the winning path represents the optimal global alignment score for the two sequences and is the \(\text{Score}\) output returned by \texttt{nwalign}.

\textbf{Note} The scoring space visually indicates if there are potential alternate winning paths, which is useful when aligning sequences with big gaps. Visual patterns in the scoring space can also indicate a possible sequence rearrangement.

\textbf{Examples}

1. Globally align two amino acid sequences using the \texttt{BLOSUM50} (default) scoring matrix and the default values for the \texttt{GapOpen} and \texttt{ExtendGap} properties. Return the optimal global alignment score in bits and the alignment character array.

\begin{verbatim}
[Score, Alignment] = nwalign('VSPAGMASGYD','IPGKASYD')
Score =
    7.3333
\end{verbatim}
Globally align two amino acid sequences specifying the PAM250 scoring matrix and a gap open penalty of 5.

```matlab
[Score, Alignment] = nwalign('IGRHRYHIGG','SRYIGRG',...
    'scoringmatrix','pam250',...
    'gapopen',5)
```

Score = 2.3333

Alignment = 
VSPAGMASGYD  : || || || | I-P-GKAS-YD

Globally align two amino acid sequences returning the Score in nat units (nats) by specifying a scale factor of log(2).

```matlab
[Score, Alignment] = nwalign('HEAGAWGHEE','PAWHEAE','Scale',log(2))
```

Score = 0.2310

Alignment = 
HEAGAWGHE-E  || || |
-P-AW-HEAE

References


See Also

aa2int | aminolookup | baselookup | blosum | dayhoff | gonnet | int2aa | int2nt | localalign | multialign | nt2aa | nt2int | nuc44 | pam | profalign | seqdotplot | showalignment | swalign

Introduced before R2006a
oligoprop

Calculate sequence properties of DNA oligonucleotide

Syntax

SeqProperties = oligoprop(SeqNT)

SeqProperties = oligoprop(SeqNT, ...'Salt', SaltValue, ...)
SeqProperties = oligoprop(SeqNT, ...'Temp', TempValue, ...)
SeqProperties = oligoprop(SeqNT, ...'Primerconc', PrimerconcValue, ...)
SeqProperties = oligoprop(SeqNT, ...'HPBase', HPBaseValue, ...)
SeqProperties = oligoprop(SeqNT, ...'HPLoop', HPLoopValue, ...)
SeqProperties = oligoprop(SeqNT, ...'Dimerlength', DimerlengthValue, ...)

Input Arguments

<table>
<thead>
<tr>
<th>SeqNT</th>
<th>DNA oligonucleotide sequence represented by any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string containing the letters A, C, G, T, or N</td>
</tr>
<tr>
<td></td>
<td>• Vector of integers containing the integers 1, 2, 3, 4, or 15</td>
</tr>
<tr>
<td></td>
<td>• Structure containing a Sequence field that contains a nucleotide sequence</td>
</tr>
<tr>
<td>SaltValue</td>
<td>Value that specifies a salt concentration in moles/liter for melting temperature calculations. Default is 0.05 moles/liter.</td>
</tr>
<tr>
<td>TempValue</td>
<td>Value that specifies the temperature in degrees Celsius for nearest-neighbor calculations of free energy. Default is 25 degrees Celsius.</td>
</tr>
<tr>
<td>PrimerconcValue</td>
<td>Value that specifies the concentration in moles/liter for melting temperature calculations. Default is 50e-6 moles/liter.</td>
</tr>
<tr>
<td>HPBaseValue</td>
<td>Value that specifies the minimum number of paired bases that form the neck of the hairpin. Default is 4 base pairs.</td>
</tr>
<tr>
<td>HPLoopValue</td>
<td>Value that specifies the minimum number of bases that form the loop of a hairpin. Default is 2 bases.</td>
</tr>
<tr>
<td>DimerlengthValue</td>
<td>Value that specifies the minimum number of aligned bases between the sequence and its reverse. Default is 4 bases.</td>
</tr>
</tbody>
</table>

Output Arguments

| SeqProperties | Structure containing the sequence properties for a DNA oligonucleotide. |

Description

SeqProperties = oligoprop(SeqNT) returns the sequence properties for a DNA oligonucleotide as a structure with the following fields:
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GC</td>
<td>Percent GC content for the DNA oligonucleotide. Ambiguous N characters in $\text{SeqNT}$ are considered to potentially be any nucleotide. If $\text{SeqNT}$ contains ambiguous N characters, GC is the midpoint value, and its uncertainty is expressed by GCdelta.</td>
</tr>
<tr>
<td>GCdelta</td>
<td>The difference between GC (midpoint value) and either the maximum or minimum value GC could assume. The maximum and minimum values are calculated by assuming all N characters are G/C or not G/C, respectively. Therefore, GCdelta defines the possible range of GC content.</td>
</tr>
<tr>
<td>Hairpins</td>
<td>$H$-by-length($\text{SeqNT}$) matrix of characters displaying all potential hairpin structures for the sequence $\text{SeqNT}$. Each row is a potential hairpin structure of the sequence, with the hairpin forming nucleotides designated by capital letters. $H$ is the number of potential hairpin structures for the sequence. Ambiguous N characters in $\text{SeqNT}$ are considered to potentially complement any nucleotide.</td>
</tr>
<tr>
<td>Dimers</td>
<td>$D$-by-length($\text{SeqNT}$) matrix of characters displaying all potential dimers for the sequence $\text{SeqNT}$. Each row is a potential dimer of the sequence, with the self-dimerizing nucleotides designated by capital letters. $D$ is the number of potential dimers for the sequence. Ambiguous N characters in $\text{SeqNT}$ are considered to potentially complement any nucleotide.</td>
</tr>
<tr>
<td>MolWeight</td>
<td>Molecular weight of the DNA oligonucleotide. Ambiguous N characters in $\text{SeqNT}$ are considered to potentially be any nucleotide. If $\text{SeqNT}$ contains ambiguous N characters, MolWeight is the midpoint value, and its uncertainty is expressed by MolWeightdelta.</td>
</tr>
<tr>
<td>MolWeightdelta</td>
<td>The difference between MolWeight (midpoint value) and either the maximum or minimum value MolWeight could assume. The maximum and minimum values are calculated by assuming all N characters are G or C, respectively. Therefore, MolWeightdelta defines the possible range of molecular weight for $\text{SeqNT}$.</td>
</tr>
<tr>
<td>Tm</td>
<td>A vector with melting temperature values, in degrees Celsius, calculated by six different methods, listed in the following order:</td>
</tr>
<tr>
<td></td>
<td>• Basic (Marmur et al., 1962 on page 1-1244)</td>
</tr>
<tr>
<td></td>
<td>• Salt adjusted (Howley et al., 1979 on page 1-1244)</td>
</tr>
<tr>
<td></td>
<td>• Nearest-neighbor (Breslauer et al., 1986 on page 1-1244)</td>
</tr>
<tr>
<td></td>
<td>• Nearest-neighbor (SantaLucia Jr. et al., 1996 on page 1-1244)</td>
</tr>
<tr>
<td></td>
<td>• Nearest-neighbor (SantaLucia Jr., 1998 on page 1-1244)</td>
</tr>
<tr>
<td></td>
<td>• Nearest-neighbor (Sugimoto et al., 1996 on page 1-1244)</td>
</tr>
<tr>
<td></td>
<td>Ambiguous N characters in $\text{SeqNT}$ are considered to potentially be any nucleotide. If $\text{SeqNT}$ contains ambiguous N characters, Tm is the midpoint value, and its uncertainty is expressed by Tmdelta.</td>
</tr>
<tr>
<td>Tmdelta</td>
<td>A vector containing the differences between Tm (midpoint value) and either the maximum or minimum value Tm could assume for each of the six methods. Therefore, Tmdelta defines the possible range of melting temperatures for $\text{SeqNT}$.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Thermo</td>
<td>4-by-3 matrix of thermodynamic calculations.</td>
</tr>
<tr>
<td></td>
<td>The rows correspond to nearest-neighbor parameters from:</td>
</tr>
<tr>
<td></td>
<td>• Breslauer et al., 1986 on page 1-1244</td>
</tr>
<tr>
<td></td>
<td>• SantaLucia Jr. et al., 1996 on page 1-1244</td>
</tr>
<tr>
<td></td>
<td>• SantaLucia Jr., 1998 on page 1-1244</td>
</tr>
<tr>
<td></td>
<td>• Sugimoto et al., 1996 on page 1-1244</td>
</tr>
<tr>
<td></td>
<td>The columns correspond to:</td>
</tr>
<tr>
<td></td>
<td>• $\Delta H$ — Enthalpy in kilocalories per mole, kcal/mol</td>
</tr>
<tr>
<td></td>
<td>• $\Delta S$ — Entropy in calories per mole-degrees Kelvin, cal/(K)(mol)</td>
</tr>
<tr>
<td></td>
<td>• $\Delta G$ — Free energy in kilocalories per mole, kcal/mol</td>
</tr>
<tr>
<td></td>
<td>Ambiguous N characters in SeqNT are considered to potentially be any</td>
</tr>
<tr>
<td></td>
<td>nucleotide. If SeqNT contains ambiguous N characters, Thermo is the</td>
</tr>
<tr>
<td></td>
<td>midpoint value, and its uncertainty is expressed by Thermodelta.</td>
</tr>
<tr>
<td>Thermodelta</td>
<td>4-by-3 matrix containing the differences between Thermo (midpoint value)</td>
</tr>
<tr>
<td></td>
<td>and either the maximum or minimum value Thermo could assume for each</td>
</tr>
<tr>
<td></td>
<td>calculation and method. Therefore, Thermodelta defines the possible range</td>
</tr>
<tr>
<td></td>
<td>of thermodynamic values for SeqNT.</td>
</tr>
</tbody>
</table>

SeqProperties = oligoprop(SeqNT, ...'PropertyName', PropertyValue, ...) calls
oligoprop with optional properties that use property name/property value pairs. You can specify
one or more properties in any order. Each PropertyName must be enclosed in single quotation
marks and is case insensitive. These property name/property value pairs are as follows:

SeqProperties = oligoprop(SeqNT, ...'Salt', SaltValue, ...) specifies a salt
cation concentration in moles/liter for melting temperature calculations. Default is 0.05 moles/liter.

SeqProperties = oligoprop(SeqNT, ...'Temp', TempValue, ...) specifies the
temperature in degrees Celsius for nearest-neighbor calculations of free energy. Default is 25
degrees Celsius.

SeqProperties = oligoprop(SeqNT, ...'Primerconc', PrimerconcValue, ...) specifies the concentration in moles/liter for melting temperatures. Default is 50e-6 moles/liter.

SeqProperties = oligoprop(SeqNT, ...'HPBase', HPBaseValue, ...) specifies the minimum number of paired bases that form the neck of the hairpin. Default is 4 base pairs.

SeqProperties = oligoprop(SeqNT, ...'HPLoop', HPLoopValue, ...) specifies the minimum number of bases that form the loop of a hairpin. Default is 2 bases.

SeqProperties = oligoprop(SeqNT, ...'Dimerlength', DimerlengthValue, ...) specifies the minimum number of aligned bases between the sequence and its reverse. Default is 4 bases.
Examples

Example 1.58. Calculating Properties for a DNA Sequence

1. Create a random sequence.
   ```
   seq = randseq(25)
   seq =
   TAGCTTCATCGTTGACTTCTACTAA
   ```

2. Calculate sequence properties of the sequence.
   ```
   S1 = oligoprop(seq)
   S1 =
   GC: 36
   GCdelta: 0
   Hairpins: [0x25 char]
   Dimers: 'tAGCTtcatcgttgacctactaa'
   MolWeight: 7.5820e+003
   MolWeightdelta: 0
   Tm: [52.7640 60.8629 62.2493 55.2870 54.0293 61.0614]
   Tmdelta: [0 0 0 0 0 0]
   Thermodelta: [4x3 double]
   ```

3. List the thermodynamic calculations for the sequence.
   ```
   S1.Thermo
   ans =
   -178.5000 -477.5700 -36.1125
   -182.1000 -497.8000 -33.6809
   -190.2000 -522.9000 -34.2974
   -191.9000 -516.9000 -37.7863
   ```

Example 1.59. Calculating Properties for a DNA Sequence with Ambiguous Characters

1. Calculate sequence properties of the sequence ACGTAGAGGACGTN.
   ```
   S2 = oligoprop('ACGTAGAGGACGTN')
   S2 =
   GC: 53.5714
   GCdelta: 3.5714
   Hairpins: 'ACGTagaggACGTn'
   Dimers: [3x14 char]
   MolWeight: 4.3329e+003
   MolWeightdelta: 20.0150
   Tm: [38.8357 42.2958 57.7880 52.4180 49.9633 55.1330]
   Tmdelta: [1.4643 1.4643 10.3885 3.4633 0.2829 3.8074]
   ```

2. List the potential dimers for the sequence.
   ```
   S2.Dimers
   ans =
   ACGTagaggacgtn
   ```
References


See Also

palindromes

Topics

isoelectric on page 1-970
molweight on page 1-1084
ntdensity on page 1-1229
randseq on page 1-1371

Introduced before R2006a
**palindromes**

Find palindromes in sequence

**Syntax**

\[
\begin{align*}
[\text{Position, Length}] &= \text{palindromes}(\text{SeqNT}) \\
[\text{Position, Length, Pal}] &= \text{palindromes}(\text{SeqNT}) \\
\ldots &= \text{palindromes}(\text{SeqNT}, \ldots, \text{'Length'}, \text{LengthValue}, \ldots) \\
\ldots &= \text{palindromes}(\text{SeqNT}, \ldots, \text{'Complement'}, \text{ComplementValue}, \ldots)
\end{align*}
\]

**Arguments**

<table>
<thead>
<tr>
<th>SeqNT</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers.</td>
</tr>
<tr>
<td></td>
<td>• Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that contains a nucleotide sequence, such as returned by emblread, fastaread, fastqread, genbankread, getembl, or getgenbank.</td>
</tr>
</tbody>
</table>

| LengthValue    | Integer specifying a minimum length for palindromes. Default is 6. |
| ComplementValue| Controls the return of complementary palindromes, that is, where the elements match their complementary pairs A-T (or U) and C-G instead of an exact nucleotide match. Choices are true or false (default). |

**Description**

\[
\begin{align*}
[\text{Position, Length}] &= \text{palindromes}(\text{SeqNT}) \text{ finds all palindromes in sequence } \text{SeqNT} \text{ with a length greater than or equal to 6, and returns the starting indices, Position, and the lengths of the palindromes, Length. }
\end{align*}
\]

\[
\begin{align*}
[\text{Position, Length, Pal}] &= \text{palindromes}(\text{SeqNT}) \text{ also returns a cell array, Pal, of the palindromes. }
\end{align*}
\]

\[
\begin{align*}
\ldots &= \text{palindromes}(\text{SeqNT}, \ldots, \text{'PropertyName'}, \text{PropertyValue}, \ldots) \text{ calls palindromes with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows: }
\end{align*}
\]

\[
\begin{align*}
\ldots &= \text{palindromes}(\text{SeqNT}, \ldots, \text{'Length'}, \text{LengthValue}, \ldots) \text{ finds all palindromes longer than or equal to LengthValue. Default is 6. }
\end{align*}
\]

\[
\begin{align*}
\ldots &= \text{palindromes}(\text{SeqNT}, \ldots, \text{'Complement'}, \text{ComplementValue}, \ldots) \text{ controls the return of complementary palindromes, that is, where the elements match their complementary pairs }
\end{align*}
\]
A-T (or A-U) and C-G instead of an exact nucleotide match. Choices for `ComplementValue` are true or false (default).

**Examples**

Find the palindromes in a simple nucleotide sequence.

```matlab
[p,l,s] = palindromes('GCTGTAACGTATATAAT')
```

```
p =
    11
    12
l =
    7
    7
s =
    'TATATAT'
    'ATATATA'
```

Find the complementary palindromes in a simple nucleotide sequence.

```matlab
[pc,lc,sc] = palindromes('TAGCTTGTCACTGAGGCCA', ...  
    'Complement',true)
```

```
pc =
    8
lc =
    7
sc =
    'TCACTGA'
```

Find the palindromes in a random nucleotide sequence.

```matlab
a = randseq(100)
```

```
a =
    TAGCTTATCGTTGACTTCTACTAA
    AAGCAAGCTCCTGAGTAGCTGGCCA
    AGCGAGCTTGCTTGTGCCCGGCTGC
    GGCGGTTGTATCTGAATACGCCAT
[ pos, len, pal ] = palindromes( a )
```

```
pos =
    74
len =
    6
pal =
    'GCGGCG'
```

**See Also**

`regexp` | `seqcomplement` | `seqrcomplement` | `seqreverse` | `seqshowwords` | `strfind`

Introduced before R2006a
pam

Return Point Accepted Mutation (PAM) scoring matrix

Syntax

ScoringMatrix = pam(N)
[ScoringMatrix, MatrixInfo] = pam(N)

... = pam(N, ...'Extended', ExtendedValue, ...)
... = pam(N, ...'Order', OrderValue, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Integer specifying the PAM scoring matrix to return. Choices are 10:10:500.</td>
</tr>
<tr>
<td>ExtendedValue</td>
<td>Controls the return of the ambiguous characters (B, Z, and X), and the stop character (*), in addition to the 20 standard amino acid characters. Choices are true or false (default).</td>
</tr>
<tr>
<td>OrderValue</td>
<td>Character vector or string that controls the order of amino acids in the returned scoring matrix. Choices are a character vector or string with at least the 20 standard amino acids. The default order of the output is A R N D C Q E G H I L K M F P S T W Y V B Z X *. If OrderValue does not contain the characters B, Z, X, and *, then these characters are not returned.</td>
</tr>
</tbody>
</table>

Description

ScoringMatrix = pam(N) returns the PAMN scoring matrix for amino acid sequences.

[ScoringMatrix, MatrixInfo] = pam(N) returns a structure with information about the PAM matrix. The fields in the structure are Name, Scale, Entropy, Expected, and Order.

... = pam(N, ...'PropertyName', PropertyValue, ...) calls pam with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = pam(N, ...'Extended', ExtendedValue, ...) controls the return of the ambiguous characters (B, Z, and X), and the stop character (*), in addition to the 20 standard amino acid characters. Choices are true or false (default).

... = pam(N, ...'Order', OrderValue, ...) controls the order of amino acids in the returned scoring matrix. Choices are a character vector or string with at least the 20 standard amino acid characters.
acids. The default ordering of the output is A R N D C Q E G H I L K M F P S T W Y V B Z X *. If OrderValue does not contain the extended characters B, Z, X, and *, then these characters are not returned.

PAM50 substitution matrix in 1/2 bit units, Expected score = -3.70, Entropy = 2.00 bits, Lowest score = -13, Highest score = 13.

PAM250 substitution matrix in 1/3 bit units, Expected score = -0.844, Entropy = 0.354 bits, Lowest score = -8, Highest score = 17.

**Examples**

Return the PAM50 matrix.

```matlab
PAM50 = pam(50)
```

Return the PAM250 matrix and specify the order of amino acids in the matrix.

```matlab
PAM250 = pam(250,'Order','CSTPAGNDEQHRKMILVFYW')
```

**See Also**

blosum | dayhoff | gonnet | localalign | nuc44 | nwalign | swalign

**Introduced before R2006a**
pdbdistplot

Visualize intermolecular distances in Protein Data Bank (PDB) file

Syntax

pdbdistplot(PDBid)
pdbdistplot(PDBid, Distance)
pdbdistplot(__,'Chain',ChainID)
pdbdistplot(__,'Model',ModelNum)
pdbdistplot(__,'Hetero',TF)

Arguments

<table>
<thead>
<tr>
<th>PDBid</th>
<th>Any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a unique identifier for a protein structure record.</td>
</tr>
<tr>
<td></td>
<td>• Name of a variable for a MATLAB structure containing PDB information for a molecular structure, such as returned by getpdb or pdbread.</td>
</tr>
<tr>
<td></td>
<td>• Name of file containing PDB information for a molecular structure, such as created by getpdb with the 'ToFile' property.</td>
</tr>
</tbody>
</table>

**Note** Each structure in the PDB database is represented by a four-character alphanumeric identifier. For example, 4hbb is the identification code for hemoglobin.

<table>
<thead>
<tr>
<th>Distance</th>
<th>Threshold distance in angstroms shown on a spy plot. Default is 7.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>ChainID</th>
<th>Any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying the chain ID to consider.</td>
</tr>
<tr>
<td></td>
<td>• Cell arrays of character vectors or string vector specifying the list of chain IDs to consider.</td>
</tr>
</tbody>
</table>

*ChainID* is case-sensitive. By default, all chains included in the model are considered.

| ModelNum    | Positive integer specifying which model to consider. Default is 1. |

Description

pdbdistplot displays the distances between atoms and between residues in a PDB structure.

pdbdistplot(PDBid) retrieves the structure specified by *PDBid* from the PDB database and creates a heat map showing inter-residue distances and a spy plot showing the residues where the minimum distances apart are less than 7 angstroms. If multiple chains are present in *PDBid*, separate plots are created.

pdbdistplot(PDBid, Distance) specifies the threshold distance shown on a spy plot. Default is 7.
pdbdistplot( ___ , 'Chain', ChainID) specifies the chains to consider. By default, all chains included in the model are considered.

pdbdistplot( ___ , 'Model', ModelNum) specifies which PDB structural model to consider. Default is 1.

pdbdistplot( ___ , 'Hetero', TF) specifies whether to include hetero atoms in the plot of residue interactions. TF is logical, true or false. Default is false.

Examples

Display a heat map of the inter-residue distances and a spy plot at 7 angstroms of the protein cytochrome C from albacore tuna.

pdbdistplot('5CYT');
Display a spy plot at 10 angstroms of the same structure.

`pdbdistplot('5CYT',10);`
See Also
getpdb | molviewer | pdbread | proteinplot | ramachandran

Introduced before R2006a
**pdbread**

Read data from Protein Data Bank (PDB) file

**Syntax**

```
PDBStruct = pdbread(File)
PDBStruct = pdbread(File, 'ModelNum', ModelNumValue)
```

**Input Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>File</code></td>
<td>Either of the following:</td>
</tr>
<tr>
<td></td>
<td>• Character vector or string specifying a file name, a path and file name,</td>
</tr>
<tr>
<td></td>
<td>or a URL pointing to a file. The referenced file is a Protein Data Bank</td>
</tr>
<tr>
<td></td>
<td>(PDB)-formatted file (ASCII text file). If you specify only a file name,</td>
</tr>
<tr>
<td></td>
<td>that file must be on the MATLAB search path or in the MATLAB Current</td>
</tr>
<tr>
<td></td>
<td>Folder.</td>
</tr>
<tr>
<td></td>
<td>• Character array or column vector of strings that contains the text of a</td>
</tr>
<tr>
<td></td>
<td>PDB-formatted file.</td>
</tr>
<tr>
<td><code>ModelNumValue</code></td>
<td>Positive integer specifying a model in a PDB-formatted file.</td>
</tr>
</tbody>
</table>

**Tip** You can use the `getpdb` function with the `ToFile` property to retrieve protein structure data from the PDB database and create a PDB-formatted file.

**Output Arguments**

```
PDBStruct = pdbread(File)
```

MATLAB structure containing a field for each PDB record.

**Description**

The Protein Data Bank (PDB) database is an archive of experimentally determined 3-D biological macromolecular structure data. For more information about the PDB format, see:

https://www.wwpdb.org/documentation/file-format

```
PDBStruct = pdbread(File) reads the data from PDB-formatted text file File and stores the data in the MATLAB structure, PDBStruct, which contains a field for each PDB record. The following table summarizes the possible PDB records and the corresponding fields in the MATLAB structure PDBStruct:
```

<table>
<thead>
<tr>
<th>PDB Database Record</th>
<th>Field in the MATLAB Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEADER</td>
<td>Header</td>
</tr>
<tr>
<td>OBSLITE</td>
<td>Obsolete</td>
</tr>
<tr>
<td>PDB Database Record</td>
<td>Field in the MATLAB Structure</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>TITLE</td>
<td>Title</td>
</tr>
<tr>
<td>CAVEAT</td>
<td>Caveat</td>
</tr>
<tr>
<td>COMPND</td>
<td>Compound</td>
</tr>
<tr>
<td>SOURCE</td>
<td>Source</td>
</tr>
<tr>
<td>KEYWDS</td>
<td>Keywords</td>
</tr>
<tr>
<td>EXPDTA</td>
<td>ExperimentData</td>
</tr>
<tr>
<td>AUTHOR</td>
<td>Authors</td>
</tr>
<tr>
<td>REVDAT</td>
<td>RevisionDate</td>
</tr>
<tr>
<td>SPRSDE</td>
<td>Superseded</td>
</tr>
<tr>
<td>JRNL</td>
<td>Journal</td>
</tr>
<tr>
<td>REMARK 1</td>
<td>Remark1</td>
</tr>
<tr>
<td>REMARK N</td>
<td>Remarkn</td>
</tr>
</tbody>
</table>

**Note** N equals 2 through 999.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBREF</td>
<td>DBReferences</td>
</tr>
<tr>
<td>SEQADV</td>
<td>SequenceConflicts</td>
</tr>
<tr>
<td>SEQRES</td>
<td>Sequence</td>
</tr>
<tr>
<td>FTNOTE</td>
<td>Footnote</td>
</tr>
<tr>
<td>MODRES</td>
<td>ModifiedResidues</td>
</tr>
<tr>
<td>HET</td>
<td>Heterogen</td>
</tr>
<tr>
<td>HETNAM</td>
<td>HeterogenName</td>
</tr>
<tr>
<td>HETSYN</td>
<td>HeterogenSynonym</td>
</tr>
<tr>
<td>FORMUL</td>
<td>Formula</td>
</tr>
<tr>
<td>HELIX</td>
<td>Helix</td>
</tr>
<tr>
<td>SHEET</td>
<td>Sheet</td>
</tr>
<tr>
<td>TURN</td>
<td>Turn</td>
</tr>
<tr>
<td>SSBOND</td>
<td>SSBond</td>
</tr>
<tr>
<td>LINK</td>
<td>Link</td>
</tr>
<tr>
<td>HYDBND</td>
<td>HydrogenBond</td>
</tr>
<tr>
<td>SLTBRG</td>
<td>SaltBridge</td>
</tr>
<tr>
<td>CISPEP</td>
<td>CISPeptides</td>
</tr>
<tr>
<td>SITE</td>
<td>Site</td>
</tr>
<tr>
<td>CRYST1</td>
<td>Cryst1</td>
</tr>
<tr>
<td>ORIGXn</td>
<td>OriginX</td>
</tr>
<tr>
<td>SCALEn</td>
<td>Scale</td>
</tr>
<tr>
<td>MTRIXn</td>
<td>Matrix</td>
</tr>
</tbody>
</table>
### PDB Database Record Field in the MATLAB Structure

<table>
<thead>
<tr>
<th>PDB Database Record</th>
<th>Field in the MATLAB Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVECT</td>
<td>TranslationVector</td>
</tr>
<tr>
<td>MODEL</td>
<td>Model</td>
</tr>
<tr>
<td>ATOM</td>
<td>Atom</td>
</tr>
<tr>
<td>SIGATM</td>
<td>AtomSD</td>
</tr>
<tr>
<td>ANISOU</td>
<td>AnisotropicTemp</td>
</tr>
<tr>
<td>SIGUIJ</td>
<td>AnisotropicTempSD</td>
</tr>
<tr>
<td>TER</td>
<td>Terminal</td>
</tr>
<tr>
<td>HETATM</td>
<td>HeterogenAtom</td>
</tr>
<tr>
<td>CONECT</td>
<td>Connectivity</td>
</tr>
</tbody>
</table>

`PDBStruct = pdbread(File, 'ModelNum', ModelNumValue)` reads only the model specified by `ModelNumValue` from the PDB-formatted text file `File` and stores the data in the MATLAB structure `PDBStruct`. If `ModelNumValue` does not correspond to an existing mode number in `File`, then `pdbread` reads the coordinate information of all the models.

**The Sequence Field**

The Sequence field is also a structure containing sequence information in the following subfields:

- `NumOfResidues`
- `ChainID`
- `ResidueNames` — Contains the three-letter codes for the sequence residues.
- `Sequence` — Contains the single-letter codes for the sequence residues.

**Note** If the sequence has modified residues, then the `ResidueNames` subfield might not correspond to the standard three-letter amino acid codes. In this case, the `Sequence` subfield will contain the modified residue code in the position corresponding to the modified residue. The modified residue code is provided in the `ModifiedResidues` field.

**The Model Field**

The Model field is also a structure or an array of structures containing coordinate information. If the MATLAB structure contains one model, the Model field is a structure containing coordinate information for that model. If the MATLAB structure contains multiple models, the Model field is an array of structures containing coordinate information for each model. The Model field contains the following subfields:

- `Atom`
- `AtomSD`
- `AnisotropicTemp`
- `AnisotropicTempSD`
- `Terminal`
- `HeterogenAtom`
The Atom Field

The Atom field is also an array of structures containing the following subfields:

- AtomSerNo
- AtomName
- altLoc
- resName
- chainID
- resSeq
- iCode
- X
- Y
- Z
- occupancy
- tempFactor
- segID
- element
- charge
- AtomNameStruct — Contains three subfields: chemSymbol, remoteInd, and branch.

Examples

1. Use the `getpdb` function to retrieve structure information from the Protein Data Bank (PDB) for the nicotinic receptor protein with identifier 1abt, and then save the data to the PDB-formatted file `nicotinic_receptor.pdb` in the MATLAB Current Folder.

   ```matlab
getpdb('1abt', 'ToFile', 'nicotinic_receptor.pdb');
```

2. Read the data from the `nicotinic_receptor.pdb` file into a MATLAB structure `pdbstruct`.

   ```matlab
   pdbstruct = pdbread('nicotinic_receptor.pdb');
   ```

3. Read only the second model from the `nicotinic_receptor.pdb` file into a MATLAB structure `pdbstruct_Model2`.

   ```matlab
   pdbstruct_Model2 = pdbread('nicotinic_receptor.pdb', 'ModelNum', 2);
   ```

4. View the atomic coordinate information in the model fields of both MATLAB structures `pdbstruct` and `pdbstruct_Model2`.

   ```matlab
   pdbstruct.Model
   ans =
   1x4 struct array with fields:
       MDLSerNo
       Atom
       Terminal
   ```

   ```matlab
   pdbstruct_Model2.Model
   ```
ans =

    MDLSerNo: 2
    Atom: [1x1205 struct]
    Terminal: [1x2 struct]

5  Read the data from a URL into a MATLAB structure, gfl_pdbstruct.

    gfl_pdbstruct = pdbread('http://www.rcsb.org/pdb/files/1gfl.pdb');

See Also

    genpeptread | getpdb | molviewer | pdbdistplot | pdbsuperpose | pdbtransform | pdbwrite

Introduced before R2006a
pdbsuperpose

Superpose 3-D structures of two proteins

Syntax

```matlab
pdbsuperpose(PDB1, PDB2)
Dist = pdbsuperpose(PDB1, PDB2)
[Dist, RMSD] = pdbsuperpose(PDB1, PDB2)
[Dist, RMSD, Transf] = pdbsuperpose(PDB1, PDB2)
[Dist, RMSD, Transf, PBD2TX] = pdbsuperpose(PDB1, PDB2)
...
= pdbsuperpose(..., 'ModelNum', ModelNumValue, ...)
... = pdbsuperpose(..., 'Scale', ScaleValue, ...)
... = pdbsuperpose(..., 'Translate', TranslateValue, ...)
... = pdbsuperpose(..., 'Reflection', ReflectionValue, ...)
... = pdbsuperpose(..., 'SeqAlign', SeqAlignValue, ...)
... = pdbsuperpose(..., 'Segment', SegmentValue, ...)
... = pdbsuperpose(..., 'Apply', ApplyValue, ...)
... = pdbsuperpose(..., 'Display', DisplayValue, ...)
```

Input Arguments

<table>
<thead>
<tr>
<th>PDB1, PDB2</th>
<th>Protein structures represented by any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a unique identifier for a protein structure record in the Protein Data Bank (PDB) database.</td>
</tr>
<tr>
<td></td>
<td>• Variable containing a PDB-formatted MATLAB structure, such as returned by <code>getpdb</code> or <code>pdbread</code>.</td>
</tr>
<tr>
<td></td>
<td>• Character vector or string specifying a file name or, a path and file name. The referenced file is a PDB-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.</td>
</tr>
</tbody>
</table>

| ModelNumValue | Two-element numeric array whose elements correspond to models in PDB1 and PDB2 respectively when PDB1 or PDB2 contains multiple models. It specifies the models to consider in the superposition. By default, the first model in each structure is considered. |

| ScaleValue | Specifies whether to include a scaling component in the linear transformation. Choices are `true` or `false` (default). |

| TranslateValue | Specifies whether to include a translation component in the linear transformation. Choices are `true` (default) or `false`. |
| **ReflectionValue** | Specifies whether to include a reflection component in the linear transformation. Choices are:
| | • `true` — Include reflection component.
| | • `false` — Exclude reflection component.
| | • 'best' — Default. May or may not include the reflection component, depending on the best fit solution. |

| **SeqAlignValue** | Specifies whether to perform a local sequence alignment and then use only the portions of the structures corresponding to the segments that align to compute the linear transformation. Choices are `true` (default) or `false`.  
| **Note** If you set the 'SeqAlign' property to `true`, you can also specify the following properties used by the `swalign` function:
| | • 'ScoringMatrix'
| | • 'GapOpen'
| | • 'ExtendGap'
| | For more information on these properties, see `swalign`. |

| **SegmentValue** | Specifies the boundaries and the chain of two subsequences to consider for computing the linear transformation. `SegmentValue` is a cell array of character vectors with the following format:
| | ```
| {'start1-stop1:chain1', 'start2-stop2:chain2'}
| ```
| | You can omit the boundaries to indicate the entire chain, such as in `{'chain1', 'start2-stop2:chain2'}`. You can specify only one pair of segments at any given time, and the specified segments are assumed to contain the same number of alpha carbon atoms. |

| **ApplyValue** | Specifies the extent to which the linear transformation should be applied. Choices are:
| | • 'all' — Default. Apply the linear transformation to the entire PDB2 structure.
| | • 'chain' — Apply the linear transformation to the specified chain only.
| | • 'segment' — Apply the linear transformation to the specified segment only. |

| **DisplayValue** | Specifies whether to display the original PDB1 structure and the resulting transformed PDB2TX structure in the Molecule Viewer window using the `molviewer` function. Each structure is represented as a separate model. Choices are `true` (default) or `false`. |
**Output Arguments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dist</strong></td>
<td>Value representing a dissimilarity measure given by the sum of the squared errors between <strong>PDB1</strong> and <strong>PDB2</strong>. For more information, see <strong>procrustes</strong> in the Statistics and Machine Learning Toolbox documentation.</td>
</tr>
<tr>
<td><strong>RMSD</strong></td>
<td>Scalar representing the root mean square distance between the coordinates of the <strong>PDB1</strong> structure and the transformed <strong>PDB2</strong> structure, considering only the atoms used to compute the linear transformation.</td>
</tr>
<tr>
<td><strong>Transf</strong></td>
<td>Linear transformation computed to superpose the chain of <strong>PDB2</strong> to the chain of <strong>PDB1</strong>. <strong>Transf</strong> is a MATLAB structure with the following fields:</td>
</tr>
<tr>
<td></td>
<td>• <strong>T</strong> — Orthogonal rotation and reflection component.</td>
</tr>
<tr>
<td></td>
<td>• <strong>b</strong> — Scale component.</td>
</tr>
<tr>
<td></td>
<td>• <strong>c</strong> — Translation component.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>Only alpha carbon atom coordinates are used to compute the linear transformation.</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>You can use the <strong>Transf</strong> output as input to the <strong>pdbtransform</strong> function.</td>
</tr>
<tr>
<td><strong>PDB2TX</strong></td>
<td>PDB-formatted MATLAB structure that represents the coordinates in the transformed <strong>PDB2</strong> protein structure.</td>
</tr>
</tbody>
</table>

**Description**

The function **pdbsuperpose**(**PDB1**, **PDB2**) computes and applies a linear transformation to superpose the coordinates of the protein structure represented in **PDB2** to the coordinates of the protein structure represented in **PDB1**. **PDB1** and **PDB2** are protein structures represented by any of the following:

- Character vector or string specifying a unique identifier for a protein structure record in the PDB database.
- Variable containing a PDB-formatted MATLAB structure, such as returned by **getpdb** or **pdbread**.
- Character vector or string specifying a file name or a path and file name. The referenced file is a PDB-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.

Alpha carbon atom coordinates of single chains for each structure are considered to compute the linear transformation (translation, reflection, orthogonal rotation, and scaling). By default, the first chain in each structure is considered to compute the transformation, and the transformation is applied to the entire molecule. By default, the original **PDB1** structure and the resulting transformed **PDB2** structure are displayed as separate models in the Molecule Viewer window using the **molviewer** function.

The function **Dist** = **pdbsuperpose**(**PDB1**, **PDB2**) returns a dissimilarity measure given by the sum of the squared errors between **PDB1** and **PDB2**. For more information, see **procrustes**.
\[ \text{Dist, RMSD} = \text{pdblsuperpose}(\text{PDB1}, \text{PDB2}) \] also returns \text{RMSD}, the root mean square distance between the coordinates of the \text{PDB1} structure and the transformed \text{PDB2} structure, considering only the atoms used to compute the linear transformation.

\[ \text{Dist, RMSD, Transf} = \text{pdblsuperpose}(\text{PDB1}, \text{PDB2}) \] also returns \text{Transf}, the linear transformation computed to superpose the chain of \text{PDB2} to the chain of \text{PDB1}. \text{Transf} is a MATLAB structure with the following fields:

- **T** — Orthogonal rotation and reflection component.
- **b** — Scale component.
- **c** — Translation component.

**Note** Only alpha carbon atom coordinates are used to compute the linear transformation.

\[ \text{Dist, RMSD, Transf, PBD2TX} = \text{pdblsuperpose}(\text{PDB1}, \text{PDB2}) \] also returns \text{PBD2TX}, a PDB-formatted MATLAB structure that represents the coordinates in the transformed \text{PDB2} protein structure.

... = pdbsuperpose(..., 'PropertyName', PropertyValue, ...) calls pdbsuperpose with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each \text{PropertyName} must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = pdbsuperpose(..., 'ModelNum', ModelNumValue, ...) specifies the models to consider in the superposition when \text{PDB1} or \text{PDB2} contains multiple models. \text{ModelNumValue} is a two-element numeric array whose elements correspond to the models in \text{PDB1} and \text{PDB2} respectively. By default, the first model in each structure is considered.

... = pdbsuperpose(..., 'Scale', ScaleValue, ...) specifies whether to include a scaling component in the linear transformation. Choices are true or false (default).

... = pdbsuperpose(..., 'Translate', TranslateValue, ...) specifies whether to include a translation component in the linear transformation. Choices are true (default) or false.

... = pdbsuperpose(..., 'Reflection', ReflectionValue, ...) specifies whether to include a reflection component in the linear transformation. Choices are true (include reflection component), false (exclude reflection component), or 'best' (may or may not include the reflection component, depending on the best fit solution). Default is 'best'.

... = pdbsuperpose(..., 'SeqAlign', SeqAlignValue, ...) specifies whether to perform a local sequence alignment and then use only the portions of the structures corresponding to the segments that align to compute the linear transformation. Choices are true (default) or false.

**Note** If you set the 'SeqAlign' property to true, you can also specify the following properties used by the swalign function:

- 'ScoringMatrix'
- 'GapOpen'
- 'ExtendGap'
... = pdbsuperpose(..., 'Segment', SegmentValue, ...) specifies the boundaries and the chain of two subsequences to consider for computing the linear transformation. SegmentValue is a cell array of character vectors with the following format: {'start1-stop1:chain1', 'start2-stop2:chain2'}. You can omit the boundaries to indicate the entire chain, such as in {'chain1', 'start2-stop2:chain2'}. You can specify only one pair of segments at any given time, and the specified segments are assumed to contain the same number of alpha carbon atoms.

... = pdbsuperpose(..., 'Apply', ApplyValue, ...) specifies the extent to which the linear transformation should be applied. Choices are 'all' (apply the linear transformation to the entire PDB2 structure), 'chain' (apply the linear transformation to the specified chain only), or 'segment' (apply the linear transformation to the specified segment only). Default is 'all'.

... = pdbsuperpose(..., 'Display', DisplayValue, ...) specifies whether to display the original PDB1 structure and the resulting transformed PDB2TX structure in the Molecule Viewer window using the molviewer function. Each structure is represented as a separate model. Choices are true (default) or false.

Examples

Example 1.60. Superposing Two Hemoglobin Structures

1 Use the getpdb function to retrieve protein structure data from the Protein Data Bank (PDB) database for two hemoglobin structures.

   str1 = getpdb('1dke');
   str2 = getpdb('4hhb');

2 Superpose the first model of the two hemoglobin structures, applying the transformation to the entire molecule.

   d = pdbsuperpose(str1, str2, 'model', [1 1], 'apply', 'all');

3 Superpose the two hemoglobin structures (each containing four chains), computing and applying the linear transformation chain by chain. Do not display the structures.

   strtx = str2;
   chainList1 = {str1.Sequence.ChainID};
   chainList2 = {str2.Sequence.ChainID};
   for i = 1:4
       [d(i), rmsd(i), tr(i), strtx] = pdbsuperpose(str1, strtx, ...
         'segment', {chainList1{i}; chainList2{i}}, ...
         'apply', 'chain', 'display', false);
   end

Example 1.61. Superposing Two Chains of a Thioredoxin Structure

Superpose chain B on chain A of a thioredoxin structure (PDBID = 2trx), and then apply the transformation only to chain B.

[d, rmsd, tr] = pdbsuperpose('2trx', '2trx', 'segment', {'A', 'B'}, ...
  'apply', 'chain')

   d =
      0.0028

   rmsd =
Example 1.62. Superposing Two Calmodulin Structures

Superpose two calmodulin structures according to the linear transformation obtained using two 20 residue-long segments.

\[
\text{pdbsuperpose('1a29', '1cll', 'segment', {'10-30:A', '10-30:A'})}
\]

\[
\text{ans = 0.1945}
\]
**pdbtransform**

Apply linear transformation to 3-D structure of molecule

**Syntax**

```matlab
pdbtransform(PDB, Transf)
PDBTX = pdbtransform(PDB, Transf)
...
= pdbtransform(..., 'ModelNum', ModelNumValue, ...)
... = pdbtransform(..., 'Segment', SegmentValue, ...)
```

**Input Arguments**

<table>
<thead>
<tr>
<th><strong>PDB</strong></th>
<th>Protein structure represented by any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a unique identifier for a protein structure record in the Protein Data Bank (PDB) database.</td>
</tr>
<tr>
<td></td>
<td>• Variable containing a PDB-formatted MATLAB structure, such as returned by <code>getpdb</code> or <code>pdbread</code>.</td>
</tr>
<tr>
<td></td>
<td>• Character vector or string specifying a file name or a path and file name. The referenced file is a PDB-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Transf</strong></th>
<th>MATLAB structure representing a linear transformation, which is applied to the coordinates of the molecule represented by <code>PDB</code>. <code>Transf</code> contains the following fields:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• T — Orthogonal rotation and reflection component.</td>
</tr>
<tr>
<td></td>
<td>• b — Scale component.</td>
</tr>
<tr>
<td></td>
<td>• c — Translation component.</td>
</tr>
</tbody>
</table>

**Tip** You can use the `Transf` structure returned by the `pdbsuperpose` function as input.

<table>
<thead>
<tr>
<th><strong>ModelNumValue</strong></th>
<th>Positive integer that specifies the model to which to apply the transformation, when <code>PDB</code> contains multiple models. By default, the first model is considered.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>SegmentValue</strong></th>
<th>Specifies the extent to which the linear transformation is applied. <code>SegmentValue</code> can be either:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• 'all' — The transformation is applied to the entire PDB input.</td>
</tr>
<tr>
<td></td>
<td>• Character vector or string specifying the boundaries and the chain to consider. It uses either of the following formats: 'start-stop:chain' or 'chain'. Omitting the boundaries indicates the entire chain.</td>
</tr>
</tbody>
</table>
Output Arguments

| PDBTX | Transformed PDB-formatted MATLAB structure. |

Description

dbtransform(PDB, Transf) applies the linear transformation specified in Transf, a MATLAB structure representing a linear transformation, to the coordinates of the molecule represented by PDB, which can be any of the following:

- Character vector or string specifying a unique identifier for a protein structure record in the PDB database.
- Variable containing a PDB-formatted MATLAB structure, such as returned by getpdb or pdbread.
- Character vector or string specifying a file name or a path and file name. The referenced file is a PDB-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.

PDBTX = pdbtransform(PDB, Transf) returns PDBTX, the transformed PDB-formatted MATLAB structure.

... = pdbtransform(...'PropertyName', PropertyValue, ...) calls pdbtransform with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = pdbtransform(..., 'ModelNum', ModelNumValue, ...) specifies the model to which to apply the transformation, when PDB contains multiple models. ModelNumValue is a positive integer. By default, the first model is considered.

... = pdbtransform(..., 'Segment', SegmentValue, ...) specifies the extent to which the linear transformation is applied. SegmentValue can be either:

- 'all' — The transformation is applied to the entire PDB input.
- Character vector or string specifying the boundaries and the chain to consider. It uses either of the following formats: 'start-stop:chain' or 'chain'. Omitting the boundaries indicates the entire chain.

Examples

1. Create a MATLAB structure that defines a linear transformation.
   
   transf.T = eye(3);  transf.b = 1;  transf.c = [11.8 -2.8 -32.3];

2. Apply the linear transformation to chain B in the thioredoxin structure, with a PDB identifier of 2trx.
   
   pdbtx = pdbtransform('2trx', transf, 'segment', 'B');

See Also

gtpdb | molviewer | pdbread | pdbsuperpose | procrustes

1-1265
Introduced in R2008b
pdbwrite

Write to file using Protein Data Bank (PDB) format

Syntax

pdbwrite(File, PDBStruct)
PDBArray = pdbwrite(File, PDBStruct)

Input Arguments

<table>
<thead>
<tr>
<th>File</th>
<th>Character vector or string specifying either a file name or a path and file name for saving the PDB-formatted data. If you specify only a file name, the file is saved to the MATLAB Current Folder.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDBStruct</td>
<td>MATLAB structure containing 3-D protein structure coordinate data, created initially by using the getpdb or pdbread functions. Note: You can edit this structure to modify its 3-D protein structure data. The coordinate information is stored in the Model field of PDBStruct.</td>
</tr>
</tbody>
</table>

Output Arguments

| PDBArray   | Character array in which each row corresponds to a line in a PDB record. |

Description

pdbwrite(File, PDBStruct) writes the contents of the MATLAB structure PDBStruct to a PDB-formatted file (ASCII text file) whose path and file name are specified by File. In the output file, File, the atom serial numbers are preserved. The atomic coordinate records are ordered according to their atom serial numbers.

Tip After you save the MATLAB structure to a local PDB-formatted file, you can use the molviewer function to display and manipulate a 3-D image of the structure.

PDBArray = pdbwrite(File, PDBStruct) saves the formatted PDB record, converted from the contents of the MATLAB structure PDBStruct, to PDBArray, a character array in which each row corresponds to a line in a PDB record.

Note You can edit PDBStruct to modify its 3-D protein structure data. The coordinate information is stored in the Model field of PDBStruct.
Examples

1 Use the `getpdb` function to retrieve structure information from the Protein Data Bank (PDB) for the green fluorescent protein with identifier 1GFL, and store the data in the MATLAB structure `gflstruct`.

   ```matlab
gflstruct = getpdb('1GFL');
```

2 Find the x-coordinate of the first atom.

   ```matlab
gflstruct.Model.Atom(1).X
   ans =
     -14.0930
```

3 Edit the x-coordinate of the first atom.

   ```matlab
gflstruct.Model.Atom(1).X = -18;
```

   **Note** Do not add or remove any Atom fields, because the `pdbwrite` function does not allow the number of elements in the structure to change.

4 Write the modified MATLAB structure `gflstruct` to a new PDB-formatted file `modified_gfl.pdb` in the Work folder on your C drive.

   ```matlab
   pdbwrite('C:\work\modified_gfl.pdb', gflstruct);
```

5 Use the `pdbread` function to read the modified PDB file into a MATLAB structure, then confirm that the x-coordinate of the first atom has changed.

   ```matlab
   modified_gflstruct = pdbread('C:\work\modified_gfl.pdb')
   modified_gflstruct.Model.Atom(1).X
   ans =
     -18
```

See Also
`getpdb` | `molviewer` | `pdbread`

Introduced in R2007a
pdist (phytree)

Calculate pairwise patristic distances in phytree object

Syntax

\[ D = \text{pdist}(\text{Tree}) \]

\[ [D, C] = \text{pdist}(\text{Tree}) \]

\[ \text{pdist}(\ldots, '\text{Nodes}', \text{NodesValue}, \ldots) \]

\[ \text{pdist}(\ldots, '\text{Squareform}', \text{SquareformValue}, \ldots) \]

\[ \text{pdist}(\ldots, '\text{Criteria}', \text{CriteriaValue}, \ldots) \]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Tree}</td>
<td>phytree object created by \text{phytree} function (object constructor) or \text{phytreeread} function.</td>
</tr>
<tr>
<td>\text{NodesValue}</td>
<td>Character vector or string specifying the nodes included in the computation. Choices are 'leaves' (default) or 'all'.</td>
</tr>
<tr>
<td>\text{SquareformValue}</td>
<td>Controls the creation of a square matrix. Choices are true or false (default).</td>
</tr>
<tr>
<td>\text{CriteriaValue}</td>
<td>Character vector or string specifying the criteria used to relate pairs. Choices are 'distance' (default) or 'levels'.</td>
</tr>
</tbody>
</table>

Description

\[ D = \text{pdist}(\text{Tree}) \] returns \( D \), a vector containing the patristic distances between every possible pair of leaf nodes of \( \text{Tree} \), a phylogenetic tree object. The patristic distances are computed by following paths through the branches of the tree and adding the patristic branch distances originally created with the seqlinkage function.

The output vector \( D \) is arranged in the order \((2,1), (3,1), \ldots, (M,1), (3,2), \ldots, (M,2), \ldots, (M,M-1)\) (the lower-left triangle of the full \( M \)-by-\( M \) distance matrix). To get the distance between the \( I \)th and \( J \)th nodes \((I > J)\), use the formula \( D((J-1)*(M-J/2)+I-J) \). \( M \) is the number of leaves.

\[ [D, C] = \text{pdist}(\text{Tree}) \] returns \( C \), the index of the closest common parent nodes for every possible pair of query nodes.

\text{pdist}(\ldots, '\text{PropertyName}', \text{PropertyValue}, \ldots) \) calls \text{pdist} with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each \text{PropertyName} must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

\text{pdist}(\ldots, '\text{Nodes}', \text{NodesValue}, \ldots) \) specifies the nodes included in the computation. Choices are 'leaves' (default) or 'all'. When \text{NodesValue} is 'leaves', the output is ordered as before, but \( M \) is the total number of nodes in the tree (\text{NumLeaves}+\text{NumBranches}).
pdist(..., 'Squareform', SquareformValue, ...) controls the creation of a square matrix. Choices are true or false (default). When SquareformValue is true, pdist converts the output into a square-formatted matrix, so that D(I,J) denotes the distance between the Ith and the Jth nodes. The output matrix is symmetric and has a zero diagonal.

pdist(..., 'Criteria', CriteriaValue, ...) changes the criteria used to relate pairs. CriteriaValue can be 'distance' (default) or 'levels'.

**Examples**

1. Read a phylogenetic tree file into a phytree object.
   
   tr = phytreeread('pf00002.tree')

2. Calculate the tree distances between pairs of leaves.
   
   dist = pdist(tr,'nodes','leaves','squareform',true)

**See Also**

phytree | phytreeread | phytreeviewer | seqlinkage | seqpdist

**Topics**

phytree object on page 1-1274

**Introduced before R2006a**
pfamhmmread

Read data from PFAM HMM-formatted file

**Syntax**

\[ HMMStruct = pfamhmmread(File) \]

**Input Arguments**

<table>
<thead>
<tr>
<th>File</th>
<th>Character vector or string specifying a file name, a path and file name, a URL pointing to a file, or the text of a PFAM-HMM-formatted file. The referenced file is a PFAM HMM-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.</th>
</tr>
</thead>
</table>

**Tip** You can use the gethmmprof function with the 'ToFile' property to retrieve HMM profile information from the PFAM database and create a PFAM HMM-formatted file.

**Output Arguments**

<table>
<thead>
<tr>
<th>HMMStruct</th>
<th>MATLAB structure containing information from a PFAM HMM-formatted file.</th>
</tr>
</thead>
</table>

**Description**

**Note** pfamhmmread reads PFAM-HMM formatted files, from file format version HMMER2.0 to HMMER3/f.

\[ HMMStruct = pfamhmmread(File) \] reads File, a PFAM HMM-formatted file, and converts it to HMMStruct, a MATLAB structure containing the following fields corresponding to parameters of an HMM profile:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The protein family name (unique identifier) of the HMM profile record in the PFAM database.</td>
</tr>
<tr>
<td>PfamAccessionNumber</td>
<td>The protein family accession number of the HMM profile record in the PFAM database.</td>
</tr>
<tr>
<td>ModelDescription</td>
<td>Description of the HMM profile.</td>
</tr>
<tr>
<td>ModelLength</td>
<td>The length of the profile (number of MATCH states).</td>
</tr>
<tr>
<td>Alphabet</td>
<td>The alphabet used in the model, 'AA' or 'NT'.</td>
</tr>
</tbody>
</table>

**Note** AlphaLength is 20 for 'AA' and 4 for 'NT'.

1-1271
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatchEmission</td>
<td>Symbol emission probabilities in the MATCH states. The format is a matrix of size ( \text{ModelLength} \times \text{AlphaLength} ), where each row corresponds to the emission distribution for a specific MATCH state.</td>
</tr>
<tr>
<td>InsertEmission</td>
<td>Symbol emission probabilities in the INSERT state. The format is a matrix of size ( \text{ModelLength} \times \text{AlphaLength} ), where each row corresponds to the emission distribution for a specific INSERT state.</td>
</tr>
<tr>
<td>NullEmission</td>
<td>Symbol emission probabilities in the MATCH and INSERT states for the NULL model. The format is a 1-by-( \text{AlphaLength} ) row vector.</td>
</tr>
<tr>
<td>Note</td>
<td>NULL probabilities are also known as the background probabilities.</td>
</tr>
<tr>
<td>BeginX</td>
<td>BEGIN state transition probabilities. Format is a 1-by-(( \text{ModelLength} + 1 )) row vector: ([B -&gt; D1 \ B -&gt; M1 \ B -&gt; M2 \ B -&gt; M3 \ldots \ B -&gt; \text{Mend}])</td>
</tr>
<tr>
<td>MatchX</td>
<td>MATCH state transition probabilities. Format is a 4-by-(( \text{ModelLength} - 1 )) matrix: ([M1 -&gt; M2 \ M2 -&gt; M3 \ldots M[\text{end-1}] -&gt; \text{Mend}; \ M1 -&gt; I1 \ M2 -&gt; I2 \ldots M[\text{end-1}] -&gt; I[\text{end-1}]; \ M1 -&gt; D2 \ M2 -&gt; D3 \ldots M[\text{end-1}] -&gt; D\text{end}; \ M1 -&gt; E \ M2 -&gt; E \ldots M[\text{end-1}] -&gt; E ])</td>
</tr>
<tr>
<td>InsertX</td>
<td>INSERT state transition probabilities. Format is a 2-by-(( \text{ModelLength} - 1 )) matrix: ([I1 -&gt; M2 \ I2 -&gt; M3 \ldots I[\text{end-1}] -&gt; \text{Mend}; \ I1 -&gt; I1 \ I2 -&gt; I2 \ldots I[\text{end-1}] -&gt; I[\text{end-1}] ])</td>
</tr>
<tr>
<td>DeleteX</td>
<td>DELETE state transition probabilities. Format is a 2-by-(( \text{ModelLength} - 1 )) matrix: ([D1 -&gt; M2 \ D2 -&gt; M3 \ldots D[\text{end-1}] -&gt; \text{Mend}; \ D1 -&gt; D2 \ D2 -&gt; D3 \ldots D[\text{end-1}] -&gt; \text{Dend} ])</td>
</tr>
<tr>
<td>FlankingInsertX</td>
<td>Flanking insert states (N and C) used for LOCAL profile alignment. Format is a 2-by-2 matrix: ([N -&gt; B \ C -&gt; T; \ N -&gt; N \ C -&gt; C])</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LoopX</td>
<td>Loop states transition probabilities used for multiple hits alignment.</td>
</tr>
<tr>
<td></td>
<td>Format is a 2-by-2 matrix:</td>
</tr>
<tr>
<td></td>
<td>[E-&gt;C \ J-&gt;B ; E-&gt;J \ J-&gt;J]</td>
</tr>
<tr>
<td>NullX</td>
<td>Null transition probabilities used to provide scores with log-odds values</td>
</tr>
<tr>
<td></td>
<td>also for state transitions.</td>
</tr>
<tr>
<td></td>
<td>Format is a 2-by-1 column vector:</td>
</tr>
<tr>
<td></td>
<td>[G-&gt;F ; G-&gt;G]</td>
</tr>
</tbody>
</table>

For more information on HMM profile models, see “HMM Profile Model” on page 1-941.

**Examples**

Read a locally saved PFAM HMM-formatted file into a MATLAB structure.

```matlab
pfamhmmread('pf00002.ls')
```

```matlab
an =
```

```
Name: '7tm 2'
PfamAccessionNumber: 'PF00002.15'
ModelDescription: '7 transmembrane receptor (Secretin family)'
ModelLength: 293
Alphabet: 'AA'
MatchEmission: [293x20 double]
InsertEmission: [293x20 double]
NullEmission: [1x20 double]
BeginX: [294x1 double]
MatchX: [292x4 double]
InsertX: [292x2 double]
DeleteX: [292x2 double]
FlankingInsertX: [2x2 double]
LoopX: [2x2 double]
NullX: [2x1 double]
```

**See Also**
gethmmalignment | gethmmprof | hmmprofalign | hmmprofstruct | showhmmprof

**Introduced before R2006a**
phytree object

Data structure containing phylogenetic tree

Description

A phytree object is a data structure containing a phylogenetic tree. Phylogenetic trees are binary rooted trees, which means that each branch is the parent of two other branches, two leaves, or one branch and one leaf. A phytree object can be ultrametric or nonultrametric.

Method Summary

Following are methods of a phytree object:

- `cluster (phytree)`: Validate clusters in phylogenetic tree
- `get (phytree)`: Retrieve information about phylogenetic tree object
- `getbyname (phytree)`: Branches and leaves from phytree object
- `getcanonical (phytree)`: Calculate canonical form of phylogenetic tree
- `getmatrix (phytree)`: Convert phytree object into relationship matrix
- `getnewickstr (phytree)`: Create Newick-formatted character vector
- `pdist (phytree)`: Calculate pairwise patristic distances in phytree object
- `plot (phytree)`: Draw phylogenetic tree
- `prune (phytree)`: Remove branch nodes from phylogenetic tree
- `reorder (phytree)`: Reorder leaves of phylogenetic tree
- `reroot (phytree)`: Change root of phylogenetic tree
- `select (phytree)`: Select tree branches and leaves in phytree object
- `subtree (phytree)`: Extract phylogenetic subtree
- `view (phytree)`: View phylogenetic tree
- `weights (phytree)`: Calculate weights for phylogenetic tree

Property Summary

Note: You cannot modify these properties directly. You can access these properties using the `get` method.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumLeaves</td>
<td>Number of leaves</td>
</tr>
<tr>
<td>NumBranches</td>
<td>Number of branches</td>
</tr>
<tr>
<td>NumNodes</td>
<td>Number of nodes ((\text{NumLeaves} + \text{NumBranches}))</td>
</tr>
<tr>
<td>Pointers</td>
<td>Branch to leaf/branch connectivity list</td>
</tr>
</tbody>
</table>
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distances</td>
<td>Edge length for every leaf/branch</td>
</tr>
<tr>
<td>LeafNames</td>
<td>Names of the leaves</td>
</tr>
<tr>
<td>BranchNames</td>
<td>Names of the branches</td>
</tr>
<tr>
<td>NodeNames</td>
<td>Names of all the nodes</td>
</tr>
</tbody>
</table>

### See Also

cluster | get | getbyname | getcanonical | getmatrix | getnewickstr | pdist | phytree | phytreeread | phytreeviewer | phytreewrite | plot | prune | reroott | select | seqlinkage | seqneighjoin | seqpdist | subtree | view | weights

**Introduced in R2006b**
**phytree**

Create phytree object

**Syntax**

\[
\begin{align*}
\text{Tree} &= \text{phytree}(B) \\
\text{Tree} &= \text{phytree}(B, D) \\
\text{Tree} &= \text{phytree}(B, C) \\
\text{Tree} &= \text{phytree}(BC) \\
\text{Tree} &= \text{phytree}(\ldots, N) \\
\text{Tree} &= \text{phytree}
\end{align*}
\]

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B)</td>
<td>Numeric array of size ([\text{NUMBRANCHES} \times 2]) in which every row represents a branch of the tree. It contains two pointers to the branch or leaf nodes, which are its children.</td>
</tr>
<tr>
<td>(C)</td>
<td>Column vector with distances for every branch.</td>
</tr>
<tr>
<td>(D)</td>
<td>Column vector with distances from every node to their parent branch.</td>
</tr>
<tr>
<td>(BC)</td>
<td>Combined matrix with pointers to branches or leaves, and distances of branches.</td>
</tr>
<tr>
<td>(N)</td>
<td>Cell array with the names of leaves and branches.</td>
</tr>
</tbody>
</table>

**Description**

\(\text{Tree} = \text{phytree}(B)\) creates an ultrametric phylogenetic tree object. In an ultrametric phylogenetic tree object, all leaves are the same distance from the root.

\(B\) is a numeric array of size \([\text{NUMBRANCHES} \times 2]\) in which every row represents a branch of the tree and it contains two pointers to the branch or leaf nodes, which are its children.

Leaf nodes are numbered from 1 to \(\text{NUMLEAVES}\) and branch nodes are numbered from \(\text{NUMLEAVES} + 1\) to \(\text{NUMLEAVES} + \text{NUMBRANCHES}\). Note that because only binary trees are allowed, \(\text{NUMLEAVES} = \text{NUMBRANCHES} + 1\).

Branches are defined in chronological order (for example, \(B(i,:) > \text{NUMLEAVES} + i\)). As a consequence, the first row can only have pointers to leaves, and the last row must represent the root branch. Parent-child distances are set to 1, unless the child is a leaf and to satisfy the ultrametric condition of the tree its distance is increased.

Given a tree with three leaves and two branches as an example.

```
    5
   /|
  4 / |
 /  | \
 2   1
```

```
5
/|
4/ |
/  |
2 1
```
In the MATLAB Command Window, type

```
B = [1 2 ; 3 4]
B =

1   2
3   4

tree = phytree(B)
```

Phylogenetic tree object with 3 leaves (2 branches)

```
view(tree)
```

```
Tree = phytree(B, D) creates an additive (ultrametric or nonultrametric) phylogenetic tree object with branch distances defined by D. D is a numeric array of size [NUMNODES X 1] with the distances of every child node (leaf or branch) to its parent branch equal to NUMNODES = NUMLEAVES + NUMBRANCHES. The last distance in D is the distance of the root node and is meaningless.

```
b = [1 2 ; 3 4 ]
b =

1   2
3   4
```

```
d = [1; 2; 1.5; 1; 0]
d =
```
Tree = phytree(B, C) creates an ultrametric phylogenetic tree object with distances between branches and leaves defined by C. C is a numeric array of size [NUMBRANCHES X 1], which contains the distance from each branch to the leaves. In ultrametric trees, all of the leaves are at the same location (same distance to the root).

\[
b = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}
\]

\[
b =
\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}
\]

\[
c = [1 4]'
\]

\[
c =
\begin{bmatrix} 1 \\ 4 \end{bmatrix}
\]
view(phytree(b,c))

Tree = phytree(BC) creates an ultrametric phylogenetic binary tree object with branch pointers in BC(:,[1 2]) and branch coordinates in BC(:,3). Same as phytree(B,C).

Tree = phytree(..., N) specifies the names for the leaves and/or the branches. N is a string vector or cell array of character vectors. If NUMEL(N)==NUMLEAVES, then the names are assigned chronologically to the leaves. If NUMEL(N)==NUMBRANCHES, the names are assigned to the branch nodes. If NUMEL(N)==NUMLEAVES + NUMBRANCHES, all the nodes are named. Unassigned names default to 'Leaf #' and/or 'Branch #' as required.

Tree = phytree creates an empty phylogenetic tree object.

Examples

Create a Phylogenetic Tree

This example shows how to create a phylogenetic tree from a multiple sequence alignment file.

Read a multiple sequence alignment file.

Sequences = multialignread('aagag.aln');

Calculate the distance between each pair of sequences.
distances = seqpdist(Sequences);

Construct a phylogenetic tree object from the pairwise distances calculated previously.
tree = seqlinkage(distances);

View the phylogenetic tree.
phytreeviewer(tree)

See Also
cluster | get | getbyname | getcanonical | getmatrix | getnewickstr | pdist | phytreeread | phytreeviewer | phytreewrite | plot | prune | reroot | select | seqlinkage | seqneighjoin | seqpdist | subtree | view | weights

Topics
phytree object on page 1-1274
Introduced in R2006a
phytreeread

Read phylogenetic tree file

Syntax

\[
Tree = \text{phytreeread}(\text{File}) \\
[Tree, Boot] = \text{phytreeread}(\text{File})
\]

Input Arguments

<table>
<thead>
<tr>
<th><strong>File</strong></th>
<th>Character vector or string specifying a Newick- or Nexus-formatted tree file (ASCII text file) name, a path and file name, or a URL pointing to a file.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>For the Nexus tree format, only one tree from the first TREES block is read. This is either the last default tree (marked by an asterisk *) or the first tree, if no default trees exist.</td>
</tr>
</tbody>
</table>

Output Arguments

<table>
<thead>
<tr>
<th><strong>Tree</strong></th>
<th>phytree object created with the function phytree.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Boot</strong></td>
<td>Column vector of bootstrap values for each tree node specified in File. If File does not specify a bootstrap value for a node, it returns a NaN value for that node. phytreeread considers the following values in File to be bootstrap values:</td>
</tr>
<tr>
<td></td>
<td>• Values within square brackets ([]) after the branch or leaf node lengths (for Newick-formatted trees only)</td>
</tr>
<tr>
<td></td>
<td>• Values that appear instead of branch or leaf node labels (for both Newick- and Nexus-formatted trees)</td>
</tr>
</tbody>
</table>

Description

\[
Tree = \text{phytreeread}(\text{File})
\] reads a Newick- or Nexus-formatted tree file and returns a phytree object on page 1-1274 containing data from the file.

The NEWICK tree format can be found at:


The NEXUS tree format can be found at:

https://wiki.christophchamp.com/index.php/NEXUS_file_format

**Note** This implementation allows only binary trees. Non-binary trees are translated into a binary tree with extra branches of length 0.
[Tree, Boot] = phytreeread(File) returns Boot, a column vector of bootstrap values for each tree node specified in File. If File does not specify a bootstrap value for a node, it returns a NaN value for that node. phytreeread considers the following values in File to be bootstrap values:

- Values within square brackets ([ ]) after the branch or leaf node lengths (for Newick-formatted trees only)
- Values that appear instead of branch or leaf node labels (for both Newick- and Nexus-formatted trees)

**Examples**

tr = phytreeread('pf00002.tree')

Phylogenetic tree object with 33 leaves (32 branches)

tr2 = phytreeread('pf00002.nex')

Phylogenetic tree object with 33 leaves (32 branches)

**See Also**

gethmmtree | phytree | phytreeviewer | phytreewrite

**Topics**

phytree object on page 1-1274

**Introduced before R2006a**
**phytreeviewer**

Visualize, edit, and explore phylogenetic tree data

**Syntax**

```matlab
phytreeviewer
phytreeviewer(Tree)
phytreeviewer(File)
```

**Description**

`phytreeviewer` opens the Phylogenetic Tree app that allows you to view, edit, and explore phylogenetic tree data.

`phytreeviewer(Tree)` loads a `phytree` object `Tree` into the app.

`phytreeviewer(File)` loads data from a Newick or ClustalW tree formatted file into the app.

**Input Arguments**

- **Tree** — Phylogenetic tree
  Phytree object

  Phylogenetic tree, specified as a `Phytree` object created with the functions `phytree` or `phytreeread`.

- **File** — Newick or ClustalW tree formatted file
  character vector | string

  Newick or ClustalW tree formatted file, specified as a character vector or string containing the file name, a path and file name, or a URL pointing to the file.

**Examples**

**View a Phylogenetic Tree**

This example shows how to view a phylogenetic tree.

Load a sample phylogenetic tree.

```matlab
tr = phytreeread('pf00002.tree')

Phylogenetic tree object with 33 leaves (32 branches)
```

View the phylogenetic tree.

```matlab
phytreeviewer(tr)
```
Alternatively, you can click Phylogenetic Tree on the Apps tab to open the app, and view the phylogenetic tree object `tr`.

See Also
cluster | phytree | phytreeread | phytreewrite | plot | view

Introduced in R2012b
### phytreewrite

Write phylogenetic tree object to Newick-formatted file

#### Syntax

```matlab
phytreewrite(File, Tree)
phytreewrite(Tree)
phytreewrite(..., 'Distances', DistancesValue, ...)
phytreewrite(..., 'BranchNames', BranchNamesValue, ...)
```

#### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>File</code></td>
<td>Character vector or string specifying a Newick-formatted file (ASCII text file) name, a path and file name, or a URL pointing to a file.</td>
</tr>
<tr>
<td><code>Tree</code></td>
<td>Phylogenetic tree object, either created with <code>phytree</code> (object constructor function) or imported using the <code>phytreeread</code> function.</td>
</tr>
</tbody>
</table>

#### Description

`phytreewrite(File, Tree)` copies the contents of a `phytree` object from the MATLAB workspace to a file. Data in the file uses the Newick format for describing trees.

`phytreewrite(Tree)` opens the Save Phylogenetic Tree As dialog box for you to enter or select a file name.

`phytreewrite(..., 'PropertyName', PropertyValue, ...)` calls `phytreewrite` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each `PropertyName` in single quotation marks. Each `PropertyName` is case insensitive. These property name/property value pairs are as follows:

- **Distances**: Specifies whether to exclude the distances from the output. `DistancesValue` can be `true` (default) or `false`.
- **BranchNames**: Specifies whether to exclude the branch names from the output. `BranchNamesValue` can be `true` (default) or `false`.

#### Examples

Read tree data from a Newick-formatted file.

```matlab
tr = phytreeread('pf00002.tree')
```

Phylogenetic tree object with 33 leaves (32 branches)

Remove all the mouse proteins and view the pruned tree.
ind = getbyname(tr, 'mouse');
tr = prune(tr, ind);
view(tr)

Write pruned tree data to a file.
phytreewrite('newtree.tree', tr)

See Also
multialignwrite | phytree | phytree object | phytreeread | phytreewriter | seqlinkage

Topics
getnewickstr on page 1-792

Introduced before R2006a
plot (DataMatrix)

Draw 2-D line plot of DataMatrix object

Syntax

plot(DMObj1)
plot(DMObj1, DMObj2)
plot(..., LineSpec)

Arguments

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note</td>
<td>If both DMObj1 and DMObj2 are input arguments, one of the inputs can be a MATLAB numeric array.</td>
</tr>
<tr>
<td>LineSpec</td>
<td>Character vector specifying a line style, marker symbol, and color of the plotted lines. For more information on these specifiers, see LineSpec.</td>
</tr>
</tbody>
</table>

Description

plot(DMObj1) plots the columns of a DataMatrix object DMObj1 versus their index.

plot(DMObj1, DMObj2) plots the data from DMObj1 and DMObj2, two DataMatrix objects, or one DataMatrix object and one MATLAB numeric array.

• If DMObj1 and DMObj2 are both vectors, they must have the same number of elements, and plot plots one vector versus the other vector, creating a single line.
• If one is a vector and one a scalar, plot plots discrete points vertically or horizontally, at the scalar value.
• If one is a vector and one a matrix, the number of elements in the vector must equal either the number of rows or the number of columns in the matrix, and plot plots the vector versus each row or column in the matrix.
• If both are matrices, they must have the same size (number of rows and columns), and plot plots each column in DMObj1 versus the corresponding column in DMObj2.

plot(..., LineSpec) plots all lines as defined by LineSpec, a character vector specifying a line style, marker symbol, and/or color.

Note For a list of line style, marker, and color specifiers, see LineSpec.

See Also

DataMatrix | plot

Topics

DataMatrix object on page 1-532
Introduced in R2008b
plot

Render heatmap or clustergram

Syntax

plot(hm_cg_object)
plot(hm_cg_object,hFig)
hAxes = plot(__)

Description

plot(hm_cg_object) renders a heatmap or clustergram of hm_cg_object.

plot(hm_cg_object,hFig) displays a heatmap or clustergram in a MATLAB figure specified by the figure handle hFig.

hAxes = plot(__) returns the handle to the axes of the heatmap or clustergram figure using any of the input argument combinations from the previous syntaxes.

Examples

Plot Heatmap of Data Matrix

Create a matrix of data.

data = gallery('invhess',20);

Display a 2-D color heatmap of the data.

hmo = HeatMap(data);

    Standardize: '[column | row | {none}]'
    Symmetric: '[true | false].'
    DisplayRange: 'Scalar.'
    Colormap: []
    ImputeFun: 'string -or- function handle -or- cell array'
    ColumnLabels: 'Cell array of strings, or an empty cell array'
    RowLabels: 'Cell array of strings, or an empty cell array'
    ColumnLabelsRotate: []
    RowLabelsRotate: []
    Annotate: '[on | {off}]'
    AnnotPrecision: []
    AnnotColor: []
    ColumnLabelsColor: 'A structure array.'
    RowLabelsColor: 'A structure array.'
    LabelsWithMarkers: '[true | false].'
    ColumnLabelsLocation: '[ top | {bottom} ]'
    RowLabelsLocation: '[ {left} | right ]'
Display the data values in the heatmap.

hmo.Annotate = true;
view(hmo)
Use the `plot` function to display the heatmap in another figure specified by the figure handle `fH`.

```matlab
fH = figure;
hA = plot(hmo,fH);
```
Use the returned axes handle \( h_A \) to specify the axes properties.

\[
\begin{align*}
  h_A.Title.String &= \text{'Inverse of an Upper Hessenberg Matrix'}; \\
  h_A.XTickLabelMode &= \text{'auto'}; \\
  h_A.YTickLabelMode &= \text{'auto'};
\end{align*}
\]
Input Arguments

hm_cg_object — Heatmap or clustergram object
HeatMap object | clustergram object

Heatmap or clustergram object, specified as a HeatMap object or clustergram object.

hFig — Handle to figure
figure handle

Handle to a figure to display the heatmap or clustergram, specified as a figure handle.

Output Arguments

hAxes — Axes of heatmap or clustergram figure
axes object

Axes of the heatmap or clustergram figure, returned as an axes object.

See Also
Axes | HeatMap | clustergram
Introduced in R2009b
plotChiSquaredFit

Plot goodness-of-fit for variance regression

Syntax

plotChiSquaredFit(test)
plotChiSquaredFit(test,Name,Value)
H = plotChiSquaredFit(___)

Description

plotChiSquaredFit(test) plots the empirical CDF of the chi-squared probabilities of the ratio between the observed and the estimated variance stratified by count levels into five equal-sized bins. Use this plot to assess the goodness-of-fit.

test, an output of the nbintest function, is a NegativeBinomialTest object. It contains results from an unpaired hypothesis test for two independent samples.

Note If the 'VarianceLink' name-value pair argument was set to 'Identity' when you ran nbintest, then the chi-squared probability is computed using the ratio between the observed variance to the mean.

plotChiSquaredFit(test,Name,Value) uses a name-value pair argument.

H = plotChiSquaredFit(____) returns handles to axes.

Examples

Perform unpaired hypothesis test for short-read count data

This example shows how to perform an unpaired hypothesis test for synthetic short-read count data from two different biological conditions.

The data in this example contains synthetic gene count data for 5000 genes, representing two different biological conditions, such as diseased and normal cells. For each condition, there are five samples. Only 10% of the genes (500 genes) are differentially expressed. Specifically, half of them (250 genes) are exactly 3-fold overexpressed. The other 250 genes are 3-fold underexpressed. The rest of the gene expression data is generated from the same negative binomial distribution for both conditions. Each sample also has a different size factor (that is, the coverage or sampling depth).

Load the data.

clear all
load('nbintest_data.mat','K','H0');

The variable K contains gene count data. The rows represent genes, and the columns represent samples. In this case, the first five columns represent samples from the first condition. The other five columns represent samples from the second condition. Display the first few rows of K.
In this example, the null hypothesis is true when the gene is not differentially expressed. The variable \( H_0 \) contains boolean indicators that indicate for which genes the null hypothesis is true (marked as 1). In other words, \( H_0 \) contains known labels that you will use later to compare with predicted results.

```matlab
sum(H0)
ans = 4500
```

Out of 5000 genes, 4500 are not differentially expressed in this synthetic data.

Run an unpaired hypothesis test for samples from two conditions using `nbintest`. The assumption is that the data came from a negative binomial distribution, where the variance is linked to the mean via a locally-regressed smooth function of the mean as described in [1] by setting 'VarianceLink' to 'LocalRegression'.

```matlab
tLocal = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','LocalRegression');
```

Use `plotVarianceLink` to plot a scatter plot for each experimental condition (for X and Y conditions), with the sample variance on the common scale versus the estimate of the condition-dependent mean. Use a linear scale for both axes. Include curves for all other linkage options by setting 'Compare' to `true`.

```matlab
plotVarianceLink(tLocal,'Scale','linear','Compare',true)
```
The **Identity** line represents the Poisson model, where the variance is identical to the mean as described in [3]. Observe that the data seems to be overdispersed (that is, most points are above the **Identity** line). The **Constant** line represents the negative binomial model, where the variance is the sum of the shot noise term (mean) and a constant multiplied by the squared mean as described in [2]. The **Local Regression** and **Constant** linkage options appear to fit better with the overdispersed data.

Use `plotChiSquaredFit` to assess the goodness-of-fit for variance regression. It plots the empirical CDF (ecdf) of the chi-squared probabilities. The probabilities are the ratio between the observed and the estimated variance stratified by short-read count levels into five equal-sized bins.

`plotChiSquaredFit(tLocal)`
Each figure shows five ecdf curves. Each curve represents one of the five short-read count levels. For instance, the blue line represents the ecdf curve for a low short-read counts between 0 and 1264. The red line represents high counts (more than 11438).

One way to interpret the curves is to check if the ecdf curves are above the diagonal line. If they are above the line, then the variance is overestimated. If they are below the line, then the variance is underestimated. In both figures, the variance seems to be correctly estimated for higher counts (that is, the red line follows the diagonal line), but slightly overestimated for lower count levels.

To assess the performance of the hypothesis test, construct a confusion matrix using the known labels and the predicted p-values.

```matlab
confusionmat(H0,(tLocal.pValue > .001))
```

```matlab
ans = 2x2
    493     7
     5  4495
```

Out of 500 differentially expressed genes, 493 are correctly predicted (true positives) and 7 of them are incorrectly predicted as not-differentially expressed genes (false negatives). Out of 4500 genes that are not differentially expressed, 4495 are correctly predicted (true negatives) and 5 of them are incorrectly predicted as differentially expressed genes (false positives).

For a comparison, run the hypothesis test again assuming that counts are modeled by the Poisson distribution, where the variance is identical to the mean.
tPoisson = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','Identity');

Plot the ecdf curves. Observe that all the curves are below the diagonal line, implying that the variance is underestimated. Therefore, the negative binomial model fits the data better.

plotChiSquaredFit(tPoisson)
Input Arguments

test — Unpaired hypothesis test result
NegativeBinomialTest object (default)

Unpaired hypothesis test results, specified as a NegativeBinomialTest object. test is returned by the nbintest function.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: 'NumBins', 4

NumBins — Number of equal-sized bins
5 (default) | positive integer

Number of equal-sized bins, specified as a comma-separated pair consisting of 'NumBins' and a positive integer.

Example: 'NumBins', 3
**Output Arguments**

**H — Handles to axes**  
vector of handles

Handles to axes, specified as a vector of handles.

**See Also**

NegativeBinomialTest | mattest | nbintest | plotVarianceLink

**Introduced in R2014b**
**plot (phytree)**

Draw phylogenetic tree

**Syntax**

```matlab
plot(Tree)
pplot(Tree, ActiveBranches)
H = plot(...)
pplot(..., 'Type', TypeValue, ...)
pplot(..., 'Orientation', OrientationValue, ...)
pplot(..., 'Rotation', RotationValue, ...)
pplot(..., 'BranchLabels', BranchLabelsValue, ...)
pplot(..., 'LeafLabels', LeafLabelsValue, ...)
pplot(..., 'TerminallLabels', TerminallLabelsValue, ...)
pplot(..., 'LLRotation', LLRotationValue, ...)
```

**Input Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>Phylogenetic tree object created, such as created with the <code>phytree</code> constructor function.</td>
</tr>
<tr>
<td>ActiveBranches</td>
<td>Logical array of size <code>numBranches</code>-by-1 indicating the active branches, which are displayed in the Figure window.</td>
</tr>
<tr>
<td>TypeValue</td>
<td>Character vector or string specifying a method for drawing the phylogenetic tree. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'square' (default)</td>
</tr>
<tr>
<td></td>
<td>• 'angular'</td>
</tr>
<tr>
<td></td>
<td>• 'radial'</td>
</tr>
<tr>
<td></td>
<td>• 'equalangle'</td>
</tr>
<tr>
<td></td>
<td>• 'equaldaylight'</td>
</tr>
<tr>
<td>OrientationValue</td>
<td>Character vector or string specifying the position of the root node, and hence the orientation of a phylogram or cladogram tree, when the 'Type' property is 'square' or 'angular'. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'left' (default)</td>
</tr>
<tr>
<td></td>
<td>• 'right'</td>
</tr>
<tr>
<td></td>
<td>• 'top'</td>
</tr>
<tr>
<td></td>
<td>• 'bottom'</td>
</tr>
<tr>
<td>RotationValue</td>
<td>Scalar between 0 (default) and 360 specifying rotation angle (in degrees) of the phylogenetic tree in the Figure window, when the 'Type' property is 'radial', 'equalangle', or 'equaldaylight'.</td>
</tr>
<tr>
<td>BranchLabelsValue</td>
<td>Controls the display of branch labels next to branch nodes. Choices are true or false (default).</td>
</tr>
</tbody>
</table>
### LeafLabelsValue
Controls the display of leaf labels next to leaf nodes. Choices are true or false. Default is:

- **true** — When the 'Type' property is 'radial', 'equalangle', or 'equaldaylight'
- **false** — When the 'Type' property is 'square' or 'angular'

### TerminalLabels
Controls the display of terminal labels over the axis tick labels, when the 'Type' property is 'square' or 'angular'. Choices are true (default) or false.

### LLRotationValue
Controls the rotation of leaf labels so that the text aligns to the root node, when the 'Type' property is 'radial', 'equalangle', or 'equaldaylight'. Choices are true or false (default).

### Output Arguments

**H**
Structure with handles to seven graph elements. The structure includes the following fields:

- axes
- BranchLines
- BranchDots
- LeafDots
- branchNodeLabels
- leafNodeLabels
- terminalNodeLabels

**Tip** Use the set function with the handles in this structure and their related properties to modify the plot. For more information on the properties you can modify using the axes handle, see Axes. For more information on the properties you can modify using the BranchLines, BranchDots, or LeafDots handle, see Primitive Line. For more information on the properties you can modify using the branchNodeLabels, leafNodeLabels, or terminalNodeLabels handle, see Text.

### Description

**plot(Tree)** draws a phylogenetic tree object into a figure as a phylogram. The significant distances between branches and nodes are in the horizontal direction. Vertical distances are arbitrary and have no significance.

**plot(Tree, ActiveBranches)** hides the nonactive branches and all of their descendants in the Figure window. **ActiveBranches** is a logical array of size numBranches-by-1 indicating the active branches.

**H = plot(...)** returns a structure with handles to seven graph elements.
plot(..., 'Type', TypeValue, ...) specifies a method for rendering the phylogenetic tree. Choices are as follows.

<table>
<thead>
<tr>
<th>Rendering Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'square' (default)</td>
<td>![Square Diagram]</td>
</tr>
<tr>
<td>'angular'</td>
<td>![Angular Diagram]</td>
</tr>
<tr>
<td>Rendering Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>'radial'</td>
<td><img src="image1" alt="Diagram" /></td>
</tr>
</tbody>
</table>

**Tip** This rendering type hides the significance of the root node and emphasizes clusters, thereby making it useful for visually assessing clusters and detecting outliers.

<table>
<thead>
<tr>
<th>'equalangle'</th>
<th><img src="image2" alt="Diagram" /></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>'equaldaylight'</th>
<th><img src="image3" alt="Diagram" /></th>
</tr>
</thead>
</table>

**Tip** This rendering type hides the significance of the root node and emphasizes clusters, thereby making it useful for visually assessing clusters and detecting outliers.
plot(..., 'Orientation', OrientationValue, ...) specifies the orientation of the root node, and hence the orientation of a phylogram or cladogram phylogenetic tree in the Figure window, when the 'Type' property is 'square' or 'angular'.

plot(..., 'Rotation', RotationValue, ...) specifies the rotation angle (in degrees) of the phylogenetic tree in the Figure window, when the 'Type' property is 'radial', 'equalangle', or 'equaldaylight'. Choices are any scalar between 0 (default) and 360.

plot(..., 'BranchLabels', BranchLabelsValue, ...) hides or displays branch labels next to the branch nodes. Choices are true or false (default).

plot(..., 'LeafLabels', LeafLabelsValue, ...) hides or displays leaf labels next to the leaf nodes. Choices are true or false. Default is:

- true — When the 'Type' property is 'radial', 'equalangle', or 'equaldaylight'
- false — When the 'Type' property is 'square' or 'angular'

plot(..., 'TerminalLabels', TerminalLabelsValue, ...) hides or displays terminal labels over the axis tick labels, when the 'Type' property is 'square' or 'angular'. Choices are true (default) or false.

plot(..., 'LLRotation', LLRotationValue, ...) controls the rotation of leaf labels so that the text aligns to the root node, when the 'Type' property is 'radial', 'equalangle', or 'equaldaylight'. Choices are true or false (default).

**Examples**

% Create a phytree object from a file
tr = phytreeread('pf00002.tree')
% Plot the tree and return a structure with handles to the
% graphic elements of the phytree object
h = plot(tr,'Type','radial')

% Modify the font size and color of the leaf node labels
% by using one of the handles in the return structure
set(h.leafNodeLabels,'FontSize',6,'Color',[1 0 0])

**See Also**
cluster | phytree | phytreeread | phytreeviewer | seqlinkage | seqneighjoin | view

**Topics**
phytree object on page 1-1274

**Introduced before R2006a**
plotVarianceLink

Plot the sample variance versus the estimate of the condition-dependent mean

Syntax

plotVarianceLink(test)
plotVarianceLink(test,Name,Value)
H = plotVarianceLink(___)

Description

plotVarianceLink(test) displays one scatter plot for each experimental condition with the sample variance on the common scale versus the estimate of the condition-dependent mean.

test, an output of the nbintest function, is a NegativeBinomialTest object, containing results from an unpaired hypothesis test for two independent samples.

If the 'PooledVariance' name-value pair argument was set to true when you ran nbintest, then plotVarianceLink plots only one scatter plot. The function also plots the variance regression according to the model specified by the 'VarianceLink' name-value pair argument of nbintest.

plotVarianceLink(test,Name,Value) uses one or more name-value pair arguments.

H = plotVarianceLink(___ ) returns handles to axes.

Examples

Perform unpaired hypothesis test for short-read count data

This example shows how to perform an unpaired hypothesis test for synthetic short-read count data from two different biological conditions.

The data in this example contains synthetic gene count data for 5000 genes, representing two different biological conditions, such as diseased and normal cells. For each condition, there are five samples. Only 10% of the genes (500 genes) are differentially expressed. Specifically, half of them (250 genes) are exactly 3-fold overexpressed. The other 250 genes are 3-fold underexpressed. The rest of the gene expression data is generated from the same negative binomial distribution for both conditions. Each sample also has a different size factor (that is, the coverage or sampling depth).

Load the data.

clear all
load('nbintest_data.mat','K','H0');

The variable K contains gene count data. The rows represent genes, and the columns represent samples. In this case, the first five columns represent samples from the first condition. The other five columns represent samples from the second condition. Display the first few rows of K.

K(1:5,:)
In this example, the null hypothesis is true when the gene is not differentially expressed. The variable \( H_0 \) contains boolean indicators that indicate for which genes the null hypothesis is true (marked as 1). In other words, \( H_0 \) contains known labels that you will use later to compare with predicted results.

\[
\text{sum}(H_0)
\]
\[
\text{ans} = 4500
\]

Out of 5000 genes, 4500 are not differentially expressed in this synthetic data.

Run an unpaired hypothesis test for samples from two conditions using \( \text{nbintest} \). The assumption is that the data came from a negative binomial distribution, where the variance is linked to the mean via a locally-regressed smooth function of the mean as described in [1] by setting 'VarianceLink' to 'LocalRegression'.

\[
tLocal = \text{nbintest}(K(:,1:5),K(:,6:10),'VarianceLink','LocalRegression');
\]

Use \( \text{plotVarianceLink} \) to plot a scatter plot for each experimental condition (for X and Y conditions), with the sample variance on the common scale versus the estimate of the condition-dependent mean. Use a linear scale for both axes. Include curves for all other linkage options by setting 'Compare' to true.

\[
\text{plotVarianceLink}(tLocal,'Scale','linear','Compare',true)
\]
Variance Link on X

- Observed
- Local Regression
- Constant
- Identity

Common Scale Variance vs. Common Scale Mean

$\times 10^4$
The **Identity** line represents the Poisson model, where the variance is identical to the mean as described in [3]. Observe that the data seems to be overdispersed (that is, most points are above the **Identity** line). The **Constant** line represents the negative binomial model, where the variance is the sum of the shot noise term (mean) and a constant multiplied by the squared mean as described in [2]. The **Local Regression** and **Constant** linkage options appear to fit better with the overdispersed data.

Use `plotChiSquaredFit` to assess the goodness-of-fit for variance regression. It plots the empirical CDF (ecdf) of the chi-squared probabilities. The probabilities are the ratio between the observed and the estimated variance stratified by short-read count levels into five equal-sized bins.

`plotChiSquaredFit(tLocal)`
Each figure shows five ecdf curves. Each curve represents one of the five short-read count levels. For instance, the blue line represents the ecdf curve for a low short-read counts between 0 and 1264. The red line represents high counts (more than 11438).

One way to interpret the curves is to check if the ecdf curves are above the diagonal line. If they are above the line, then the variance is overestimated. If they are below the line, then the variance is underestimated. In both figures, the variance seems to be correctly estimated for higher counts (that is, the red line follows the diagonal line), but slightly overestimated for lower count levels.

To assess the performance of the hypothesis test, construct a confusion matrix using the known labels and the predicted p-values.

```
confusionmat(H0,(tLocal.pValue > .001))
```

```
ans = 2x2

493     7
5  4495
```

Out of 500 differentially expressed genes, 493 are correctly predicted (true positives) and 7 of them are incorrectly predicted as not-differentially expressed genes (false negatives). Out of 4500 genes that are not differentially expressed, 4495 are correctly predicted (true negatives) and 5 of them are incorrectly predicted as differentially expressed genes (false positives).

For a comparison, run the hypothesis test again assuming that counts are modeled by the Poisson distribution, where the variance is identical to the mean.
tPoisson = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','Identity');

Plot the ecdf curves. Observe that all the curves are below the diagonal line, implying that the variance is underestimated. Therefore, the negative binomial model fits the data better.

plotChiSquaredFit(tPoisson)
Input Arguments

test — Unpaired hypothesis test result
NegativeBinomialTest object (default)

Unpaired hypothesis test results, specified as a `NegativeBinomialTest` object. `test` is returned by the `nbintest` function.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `Compare',true,'Scale','linear'

Compare — Logical flag to add a curve with the variance link for other models
false (default) | true

Logical flag to add a curve with the variance link for other models, specified as a comma-separated pair consisting of `'Compare'` and `true` or `false`. When it is set to `true`, the plot shows curves for all the available linkage options, that is, `'LocalRegression'`, `'Constant'`, and `'Identity'`.

Example: `'Compare',true`
Scale — Scale for both axes
'log' (default) | 'linear'

Scale for both axes, specified as a comma-separated pair consisting of 'Scale' and 'log' or 'linear'.
Example: 'Scale','linear'

Output Arguments

H — Handles to axes
vector of handles

Handles to axes, specified as a vector of handles.

See Also

NegativeBinomialTest | mattest | nbintest | plotChiSquaredFit

Introduced in R2014b
plus (DataMatrix)

Add DataMatrix objects

Syntax

\[ DMObjNew = \text{plus}(DMObj1, DMObj2) \]
\[ DMObjNew = DMObj1 + DMObj2 \]
\[ DMObjNew = \text{plus}(DMObj1, B) \]
\[ DMObjNew = DMObj1 + B \]
\[ DMObjNew = \text{plus}(B, DMObj1) \]
\[ DMObjNew = B + DMObj1 \]

Input Arguments

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>MATLAB numeric or logical array.</td>
</tr>
</tbody>
</table>

Output Arguments

| DMObjNew        | DataMatrix object created by addition.                                 |

Description

\[ DMObjNew = \text{plus}(DMObj1, DMObj2) \] or the equivalent \[ DMObjNew = DMObj1 + DMObj2 \] performs an element-by-element addition of the DataMatrix objects \( DMObj1 \) and \( DMObj2 \) and places the results in \( DMObjNew \), another DataMatrix object. \( DMObj1 \) and \( DMObj2 \) must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for \( DMObjNew \) are the same as \( DMObj1 \), unless \( DMObj1 \) is a scalar; then they are the same as \( DMObj2 \).

\[ DMObjNew = \text{plus}(DMObj1, B) \] or the equivalent \[ DMObjNew = DMObj1 + B \] performs an element-by-element addition of \( DMObj1 \), a DataMatrix object, and \( B \), a numeric or logical array, and places the results in \( DMObjNew \), another DataMatrix object. \( DMObj1 \) and \( B \) must have the same size (number of rows and columns), unless \( B \) is a scalar. The size (number of rows and columns), row names, and column names for \( DMObjNew \) are the same as \( DMObj1 \).

\[ DMObjNew = \text{plus}(B, DMObj1) \] or the equivalent \[ DMObjNew = B + DMObj1 \] performs an element-by-element addition of \( B \), a numeric or logical array, and \( DMObj1 \), a DataMatrix object, and places the results in \( DMObjNew \), another DataMatrix object. \( DMObj1 \) and \( B \) must have the same size (number of rows and columns), unless \( B \) is a scalar. The size (number of rows and columns), row names, and column names for \( DMObjNew \) are the same as \( DMObj1 \).

Note: Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.
MATLAB calls `DMObjNew = plus(X, Y)` for the syntax `DMObjNew = X + Y` when `X` or `Y` is a DataMatrix object.

**See Also**
DataMatrix | minus

**Topics**
DataMatrix object on page 1-532

**Introduced in R2008b**
power (DataMatrix)

Array power DataMatrix objects

Syntax

DMObjNew = power(DMObj1, DMObj2)
DMObjNew = DMObj1 .^ DMObj2
DMObjNew = power(DMObj1, B)
DMObjNew = DMObj1 .^ B
DMObjNew = power(B, DMObj1)
DMObjNew = B .^ DMObj1

Input Arguments

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>MATLAB numeric or logical array.</td>
</tr>
</tbody>
</table>

Output Arguments

| DMObjNew       | DataMatrix object created by array power.                             |

Description

DMObjNew = power(DMObj1, DMObj2) or the equivalent DMObjNew = DMObj1 .^ DMObj2 performs an element-by-element power of the DataMatrix objects DMObj1 and DMObj2 and places the results in DMObjNew, another DataMatrix object. In other words, power raises each element in DMObj1 by the corresponding element in DMObj2. DMObj1 and DMObj2 must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for DMObjNew are the same as DMObj1, unless DMObj1 is a scalar; then they are the same as DMObj2.

DMObjNew = power(DMObj1, B) or the equivalent DMObjNew = DMObj1 .^ B performs an element-by-element power of the DataMatrix object DMObj1 and B, a numeric or logical array, and places the results in DMObjNew, another DataMatrix object. In other words, power raises each element in DMObj1 by the corresponding element in B. DMObj1 and B must have the same size (number of rows and columns), unless B is a scalar. The size (number of rows and columns), row names, and column names for DMObjNew are the same as DMObj1.

DMObjNew = power(B, DMObj1) or the equivalent DMObjNew = B .^ DMObj1 performs an element-by-element power of B, a numeric or logical array, and the DataMatrix object DMObj1, and places the results in DMObjNew, another DataMatrix object. In other words, power raises each element in B by the corresponding element in DMObj1. DMObj1 and B must have the same size (number of rows and columns), unless B is a scalar. The size (number of rows and columns), row names, and column names for DMObjNew are the same as DMObj1.

Note Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.
MATLAB calls `DMObjNew = power(X, Y)` for the syntax `DMObjNew = X .^ Y` when `X` or `Y` is a DataMatrix object.

**See Also**
DataMatrix | times

**Topics**
DataMatrix object on page 1-532

**Introduced in R2008b**
preset

Set combination of alignment options

Syntax

preset(object,P)

Description

preset(object,P) sets a combination of alignment options (object properties) to predefined values for typical tradeoffs between execution time and sensitivity. The function sets the following object properties: NumReseedings, NumSeedExtensions, NumSeedMismatches, SeedIntervalFunction, SeedLength, and Mode.

preset requires the Bioinformatics Toolbox Interface for Bowtie Aligner support package. If this support package is not installed, then the function provides a download link.

Note preset is supported on Mac and UNIX platforms only.

Examples

Map Reads to Reference Sequence Using Fast Alignment Options

Build a set of index files for the Drosophila genome. An error message appears if you do not have the Bioinformatics Toolbox Interface for Bowtie Aligner support package installed when you run the function. Click the provided link to download the package from the Add-on menu.

For this example, the reference sequence Dmel_chr4.fa is already provided with the toolbox.

status = bowtie2build('Dmel_chr4.fa', 'Dmel_chr4_index');

If the index build is successful, the function returns 0 and creates the index files (*.bt2) in the current folder. The files have the prefix 'Dmel_chr4_index'.

Once the index is ready, map the read sequences to the reference. The paired-end read files (SRR6008575_10k_1.fq and SRR6008575_10k_2.fq) are already provided with the toolbox.

Create an options object.

alignOpt = Bowtie2AlignOptions;

Some preset options define a combination of values for several alignment parameters at the same time. Use an empty string to see the list of all the preset options.

preset(alignOpt,'')

ans =

8×6 table
Align using the 'Fast' option, which makes the alignment process faster but less sensitive and less accurate.

```matlab
preset(alignOpt,'Fast');
```

Map reads to the reference using the specified alignment options.

```matlab
flag = run(alignOpt,'Dmel_chr4','SRR6008575_10k_1.fq','SRR6008575_10k_2.fq','SRR6008575_10k_chr4.sam');
```

The output is a SAM-formatted file that contains the mapping results.

### Input Arguments

- **object** — Alignment options
  
  Bowtie2AlignOptions object

  Alignment options, specified as a Bowtie2AlignOptions object.

  Example: `alignOpt`

- **P** — Preset option
  
  `'Sensitive'` object (default) | `'VerySensitive'` | `'Fast'` | `'VeryFast'` | `'LocalFast'` | `'LocalVeryFast'` | `'LocalSensitive'` | `'LocalVerySensitive'`

  Preset option, specified as a character vector. Valid options are `'Sensitive'`, `'VerySensitive'`, `'Fast'`, `'VeryFast'`, `'LocalFast'`, `'LocalVeryFast'`, `'LocalSensitive'`, `'LocalVerySensitive'`, and `''. Use an empty character vector `''` to display the predefined values for each preset option.

  Example: `'LocalFast'`

### References


### See Also

- Bowtie2AlignOptions
- Bowtie2BuildOptions
- Bowtie2InspectOptions
- bowtie2
- bowtie2build
- bowtie2inspect

### External Websites

- Bowtie 2 manual
Introduced in R2018a
**probelibraryinfo**

Create table of probe set library information

**Syntax**

\[ \text{ProbeInfo} = \text{probelibraryinfo}(\text{CELStruct}, \text{CDFStruct}) \]

**Input Arguments**

<table>
<thead>
<tr>
<th>\text{CELStruct}</th>
<th>Structure created by the \text{affyread} function from an Affymetrix CEL file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{CDFStruct}</td>
<td>Structure created by the \text{affyread} function from an Affymetrix CDF library file associated with the CEL file.</td>
</tr>
</tbody>
</table>

**Output Arguments**

<table>
<thead>
<tr>
<th>\text{ProbeInfo}</th>
<th>Three-column matrix with the same number of rows as the \text{Probes} field of the \text{CELStruct}.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Column 1 — Probe set ID/name to which the probe belongs. (Probes that do not belong to a probe set in the CDF library file have probe set ID/name equal to 0.)</td>
</tr>
<tr>
<td></td>
<td>• Column 2 — Contains the probe pair number.</td>
</tr>
<tr>
<td></td>
<td>• Column 3 — Indicates if the probe is a perfect match (1) or mismatch (-1) probe.</td>
</tr>
</tbody>
</table>

**Description**

\[ \text{ProbeInfo} = \text{probelibraryinfo}(\text{CELStruct}, \text{CDFStruct}) \] creates a table of information linking the probe data from \text{CELStruct}, a structure created from an Affymetrix CEL file, with probe set information from \text{CDFStruct}, a structure created from an Affymetrix CDF file.

**Note** Affymetrix probe pair indexing is 0-based, while MATLAB software indexing is 1-based. The output from \text{probelibraryinfo} is 1-based.

**Examples**

**Retrieve Probe Set Library Information**

This example shows how to extract probe set library information from Affymetrix® GeneChip® microarray data.

This example uses sample data from the \text{E. coli} Antisense Genome Array. Download the data from Demo_Data_Ecoli-antisense.zip. Extract the data files from the DTT archive using the Data Transfer Tool.
You also need to download Ecoli_ASv2.CDF library file for the E. coli Antisense Genome Array. You may already have these files if you have any Affymetrix GeneChip software installed on your machine. If not, get the library files by downloading and unzipping the E. coli Antisense Genome Array zip file.

Read the contents of a CEL file into a MATLAB structure.

celStruct = affyread('Ecoli-antisense-121502.CEL');

Read the contents of a CDF file into a MATLAB structure.

cdfStruct = affyread('C:\LibFiles\Ecoli_ASv2.CDF');

Extract probe set library information.

probeInfo = probelibraryinfo(celStruct, cdfStruct);

Determine the probe set to which the 1104th probe belongs.

cdfStruct.ProbeSets(probeInfo(1104,1)).Name

ans =
    'thrA_b0002_at'

See Also

affyread | celintensityread | probesetlink | probesetlookup | probesetplot | probesetvalues

Topics

“Working with Affymetrix® Data”

Introduced before R2006a
probesetlink

Display probe set information on NetAffx Web site

Syntax

probesetlink(AffyStruct, PS)
URL = probesetlink(AffyStruct, PS)

probesetlink(AffyStruct, PS, ...'Source', SourceValue, ...)
probesetlink(AffyStruct, PS, ...'Browser', BrowserValue, ...)
URL = probesetlink(AffyStruct, PS, ...'NoDisplay', NoDisplayValue, ...)

Input Arguments

| AffyStruct | Structure created by the affyread function from an Affymetrix CHP file or an Affymetrix CDF library file. |
| PS | Probe set index or the probe set ID/name. |
| SourceValue | Controls the linking to the data source (for example, GenBank or Flybase) for the probe set (instead of linking to the NetAffx™ Web site). Choices are true or false (default). |

**Note** This property requires the GIN library file associated with the CHP or CDF file to be located in the same folder as the CDF library file.

| BrowserValue | Controls the display of the probe set information in your system's default Web browser. Choices are true or false (default). |

| NoDisplayValue | Controls the return of URL without opening a Web browser. Choices are true or false (default). |

Output Arguments

| URL | URL for the probe set information. |

Description

probesetlink(AffyStruct, PS) opens a Web Browser window displaying information on the NetAffx Web site about a probe set specified by PS, a probe set index or the probe set ID/name, and AffyStruct, a structure created from an Affymetrix CHP file or Affymetrix CDF library file.

URL = probesetlink(AffyStruct, PS) also returns the URL (linking to the NetAffx Web site) for the probe set information.

probesetlink(AffyStruct, PS, ...'PropertyName', PropertyValue, ...) calls probesetLink with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:
probesetlink(AffyStruct, PS, ...'Source', SourceValue, ...) controls the linking to the data source (for example, GenBank or Flybase) for the probe set (instead of linking to the NetAffx Web site). Choices are true or false (default).

**Note** The 'Source' property requires the GIN library file associated with the CHP or CDF file to be located in the same folder as the CDF library file.

probesetlink(AffyStruct, PS, ...'Browser', BrowserValue, ...) controls the display of the probe set information in your system's default Web browser. Choices are true or false (default).

URL = probesetlink(AffyStruct, PS, ...'NoDisplay', NoDisplayValue, ...) controls the return of the URL without opening a Web browser. Choices are true or false (default).

**Note** The NetAffx Web site requires you to register and provide a user name and password.

**Examples**

**Display Probe Set Information on NetAffx™ Web Site**

This example uses sample data from the *E. coli* Antisense Genome Array. Download the data from Demo_Data_E-coli-antisense.zip. Extract the data files from the DTT archive using the Data Transfer Tool.

You also need to download Ecoli_ASv2.CDF and Ecoli-antisense-121502.CHP files for the *E. coli* Antisense Genome Array. You may already have these files if you have any Affymetrix GeneChip software installed on your machine. If not, get the library files by downloading and unzipping the *E. coli* Antisense Genome Array zip file.

Read the contents of a CHP file into a MATLAB structure, assuming the corresponding CDF file is stored at C:\LibFiles.

chpStruct = affyread('Ecoli-antisense-121502.CHP','C:\LibFiles');

Display information from the NetAffx Web site for the argG_b3172_at probe set.

probesetlink(chpStruct,'argG_b3172_at')

**See Also**

affyread | celintensityread | probelibraryinfo | probesetlookup | probesetplot | probesetvalues

**Topics**

“Working with Affymetrix® Data”

**Introduced before R2006a**
probesetlookup

Look up information for Affymetrix probe set

**Syntax**

```matlab
PSStruct = probesetlookup(AffyStruct, ID)
```

**Input Arguments**

<table>
<thead>
<tr>
<th>AffyStruct</th>
<th>Structure created by the <code>affyread</code> function from an Affymetrix CHP file or an Affymetrix CDF library file for expression assays.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Character vector, string, string vector, or cell array of character vectors specifying one or more probe set IDs/names or gene IDs.</td>
</tr>
</tbody>
</table>

**Output Arguments**

<table>
<thead>
<tr>
<th>PSStruct</th>
<th>Structure or array of structures containing the following fields for a probe set:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier</td>
<td>Gene ID associated with the probe set</td>
</tr>
<tr>
<td>_probeSetName</td>
<td>Probe set ID/name</td>
</tr>
<tr>
<td>CDFIndex</td>
<td>Index into the CDF structure for the probe set</td>
</tr>
<tr>
<td>GINIndex</td>
<td>Index into the GIN structure for the probe set</td>
</tr>
<tr>
<td>Description</td>
<td>Description of the probe set</td>
</tr>
<tr>
<td>Source</td>
<td>Source(s) of the probe set</td>
</tr>
<tr>
<td>SourceURL</td>
<td>Source URL(s) for the probe set</td>
</tr>
</tbody>
</table>

**Description**

`PSStruct = probesetlookup(AffyStruct, ID)` returns a structure or an array of structures containing information for an Affymetrix probe set specified by ID, a character vector, string, string vector, or cell array of character vectors specifying one or more probe set IDs/names or gene IDs, and by `AffyStruct`, a structure created from an Affymetrix CHP file or Affymetrix CDF library file for expression assays.

**Note** This function works with CHP files and CDF files for expression assays only. It requires that the GIN library file associated with the CHP file or CDF file to be located in the same folder as the CDF library file.

**Examples**

The following example uses the CDF library file from the `E. coli` Antisense Genome array, which you can download from:

https://www.affymetrix.com/support/technical/sample_data/demo_data.affx
The following example assumes that the `Ecoli_ASv2.CDF` library file is stored at `D:\Affymetrix\LibFiles\Ecoli`.

1. Read the contents of a CDF library file into a MATLAB structure.
   ```matlab
cdfStruct = affyread('D:\Affymetrix\LibFiles\Ecoli\Ecoli_ASv2.CDF');
```

2. Look up the gene ID (Identifier) associated with the `argG_b3172_at` probe set.
   ```matlab
   probesetlookup(cdfStruct,'argG_b3172_at')
   ans =
   
   Identifier: '3315278'
   ProbeSetName: 'argG_b3172_at'
   CDFIndex: 5213
   GINIndex: 3074
   Description: [1x82 char]
   Source: 'NCBI EColi Genome'
   SourceURL: [1x74 char]
   ``

See Also

`affyread` | `celintensityread` | `probelibraryinfo` | `probesetlink` | `probesetplot` | `probesetvalues` | `rmabackadj`

Introduced before R2006a
probesetplot

Plot Affymetrix probe set intensity values

Syntax

probesetplot(CELStruct, CDFStruct, PS)
probesetplot(CELStruct, CDFStruct, PS, ...'GeneName', GeneNameValue, ...)
probesetplot(CELStruct, CDFStruct, PS, ...'Field', FieldValue, ...)
probesetplot(CELStruct, CDFStruct, PS, ...'ShowStats', ShowStatsValue, ...)

Arguments

<table>
<thead>
<tr>
<th>CELStruct</th>
<th>Structure created by the affyread function from an Affymetrix CEL file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDFStruct</td>
<td>Structure created by the affyread function from an Affymetrix CDF library file associated with the CEL file.</td>
</tr>
<tr>
<td>PS</td>
<td>Probe set index or the probe set ID/name.</td>
</tr>
<tr>
<td>GeneNameValue</td>
<td>Controls whether the probe set name or the gene name is used for the title of the plot. Choices are true or false (default).</td>
</tr>
<tr>
<td>Note</td>
<td>The 'GeneName' property requires the GIN library file associated with the CEL and CDF files to be located in the same folder as the CDF library file from which CDFStruct was created.</td>
</tr>
<tr>
<td>FieldValue</td>
<td>Character vector or string specifying the type of data to plot. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'Intensity' (default)</td>
</tr>
<tr>
<td></td>
<td>• 'StdDev'</td>
</tr>
<tr>
<td></td>
<td>• 'Background'</td>
</tr>
<tr>
<td></td>
<td>• 'Pixels'</td>
</tr>
<tr>
<td></td>
<td>• 'Outlier'</td>
</tr>
<tr>
<td>ShowStatsValue</td>
<td>Controls whether the mean and standard deviation lines are included in the plot. Choices are true or false (default).</td>
</tr>
</tbody>
</table>

Description

probesetplot(CELStruct, CDFStruct, PS) plots the PM (perfect match) and MM (mismatch) intensity values for a specified probe set. CELStruct is a structure created by the affyread function from an Affymetrix CEL file. CDFStruct is a structure created by the affyread function from an Affymetrix CDF library file associated with the CEL file. PS is the probe set index or the probe set ID/name.
Note MATLAB software uses 1-based indexing for probe set numbers, while the Affymetrix CDF file uses 0-based indexing for probe set numbers. For example, CDFStruct.ProbeSets(1) has a ProbeSetNumber of 0 in the ProbePairs field.

probesetplot(CELStruct, CDFStruct, PS, ...'PropertyName', PropertyValue, ...) calls probesetplot with optional properties that use property name/property value pairs. You can specify one or more properties in any order. EachPropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

probesetplot(CELStruct, CDFStruct, PS, ...'GeneName', GeneNameValue, ...) controls whether the probe set name or the gene name is used for the title of the plot. Choices are true or false (default).

Note The 'GeneName' property requires the GIN library file associated with the CEL and CDF files to be located in the same folder as the CDF library file from which CDFStruct was created.

probesetplot(CELStruct, CDFStruct, PS, ...'Field', FieldValue, ...) specifies the type of data to plot. Choices are:

- 'Intensity' (default)
- 'StdDev'
- 'Background'
- 'Pixels'
- 'Outlier'

probesetplot(CELStruct, CDFStruct, PS, ...'ShowStats', ShowStatsValue, ...) controls whether the mean and standard deviation lines are included in the plot. Choices are true or false (default).

Examples

Plot Affymetrix™ Probe Set Intensity Values

This example uses sample data from the E. coli Antisense Genome Array. Download the data from Demo_Data_E-coli-antisense.zip. Extract the data files from the DTT archive using the Data Transfer Tool.

You also need to download Ecoli_ASv2.CDF library file for the E. coli Antisense Genome Array. You may already have these files if you have any Affymetrix GeneChip software installed on your machine. If not, get the library files by downloading and unzipping the E. coli Antisense Genome Array zip file.

Read the contents of a CEL file into a MATLAB structure.

celStruct = affyread('Ecoli-antisense-121502.CEL');

Read the contents of a CDF file into a MATLAB structure.

cdfStruct = affyread('C:\LibFiles\Ecoli_ASv2.CDF');
Plot the PM and MM intensity values of the \texttt{argG\_b3172\_at} probe set, including the mean and standard deviation.

\begin{verbatim}
probesetplot(celStruct, cdfStruct, 'argG\_b3172\_at', 'showstats', true)
\end{verbatim}

See Also

\texttt{affyread} | \texttt{celintensityread} | \texttt{probesetlink} | \texttt{probesetlookup} | \texttt{probesetvalues}

Topics

“Working with Affymetrix® Data”

Introduced before R2006a
probesetvalues

Create table of Affymetrix probe set intensity values

Syntax

`PSValues = probesetvalues(CELStruct, CDFStruct, PS)`  
`PSValues = probesetvalues(CELStruct, CDFStruct, PS, 'Background', BackgroundValue)`  
`ColumnNames = probesetvalues`

Input Arguments

<table>
<thead>
<tr>
<th>CELStruct</th>
<th>Structure created by the affyread function from an Affymetrix CEL file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDFStruct</td>
<td>Structure created by the affyread function from an Affymetrix CDF library file associated with the CEL file.</td>
</tr>
<tr>
<td>PS</td>
<td>Probe set index or the probe set ID/name.</td>
</tr>
<tr>
<td>BackgroundValue</td>
<td>Controls the background correction in the calculation. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• true (default) — Background values from the Background field in the PSValues matrix are used to calculate the probe intensity values.</td>
</tr>
<tr>
<td></td>
<td>• false — Background values are not calculated.</td>
</tr>
<tr>
<td></td>
<td>• A vector of precalculated background values (such as returned by the zonebackadj function) whose length is equal to the number of probes in CELStruct. These background values are used to calculate the probe intensity values.</td>
</tr>
</tbody>
</table>

**Tip** Including background correction in the calculation of the probe intensity values can be slow. Therefore, setting 'Background' to false can speed up the calculation. However, the values returned in the 'Background' field of the PSValues matrix will be zero.

Output Arguments

<table>
<thead>
<tr>
<th>PSValues</th>
<th>Twenty-column matrix with one row for each probe pair in the probe set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColumnNames</td>
<td>Cell array of character vectors containing the column names of the PSValues matrix. This is returned only when you call probesetvalues with no input arguments.</td>
</tr>
</tbody>
</table>

Description

`PSValues = probesetvalues(CELStruct, CDFStruct, PS)` creates a table of intensity values for PS, a probe set, from the probe-level data in CELStruct, a structure created by the affyread
function from an Affymetrix CEL file. PS is a probe set index or probe set ID/name from CDFStruct, a structure created by the affyread function from an Affymetrix CDF library file associated with the CEL file. PSValues is a twenty-column matrix with one row for each probe pair in the probe set. The columns correspond to the following fields.

<table>
<thead>
<tr>
<th>Column</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'ProbeSetNumber'</td>
<td>Number identifying the probe set to which the probe pair belongs.</td>
</tr>
<tr>
<td>2</td>
<td>'ProbePairNumber'</td>
<td>Index of the probe pair within the probe set.</td>
</tr>
<tr>
<td>3</td>
<td>'UseProbePair'</td>
<td>This field is for backward compatibility only and is not currently used.</td>
</tr>
<tr>
<td>4</td>
<td>'Background'</td>
<td>Estimated background of probe intensity values of the probe pair.</td>
</tr>
<tr>
<td>5</td>
<td>'PMPosX'</td>
<td>x-coordinate of the perfect match probe.</td>
</tr>
<tr>
<td>6</td>
<td>'PMPosY'</td>
<td>y-coordinate of the perfect match probe.</td>
</tr>
<tr>
<td>7</td>
<td>'PMIntensity'</td>
<td>Intensity value of the perfect match probe.</td>
</tr>
<tr>
<td>8</td>
<td>'PMStdDev'</td>
<td>Standard deviation of intensity value of the perfect match probe.</td>
</tr>
<tr>
<td>9</td>
<td>'PMPixels'</td>
<td>Number of pixels in the cell containing the perfect match probe.</td>
</tr>
<tr>
<td>10</td>
<td>'PMOutlier'</td>
<td>True/false flag indicating if the perfect match probe was marked as an outlier.</td>
</tr>
<tr>
<td>11</td>
<td>'PMMasked'</td>
<td>True/false flag indicating if the perfect match probe was masked.</td>
</tr>
<tr>
<td>12</td>
<td>'MMPosX'</td>
<td>x-coordinate of the mismatch probe.</td>
</tr>
<tr>
<td>13</td>
<td>'MMPosY'</td>
<td>y-coordinate of the mismatch probe.</td>
</tr>
<tr>
<td>14</td>
<td>'MMIntensity'</td>
<td>Intensity value of the mismatch probe.</td>
</tr>
<tr>
<td>15</td>
<td>'MMStdDev'</td>
<td>Standard deviation of intensity value of the mismatch probe.</td>
</tr>
<tr>
<td>16</td>
<td>'MMPixels'</td>
<td>Number of pixels in the cell containing the mismatch probe.</td>
</tr>
<tr>
<td>17</td>
<td>'MMOutlier'</td>
<td>True/false flag indicating if the mismatch probe was marked as an outlier.</td>
</tr>
<tr>
<td>18</td>
<td>'MMMasked'</td>
<td>True/false flag indicating if the mismatch probe was masked.</td>
</tr>
<tr>
<td>19</td>
<td>'GroupNumber'</td>
<td>Number identifying the group to which the probe pair belongs. For expression arrays, this is always 1. For genotyping arrays, this is typically 1 (allele A, sense), 2 (allele B, sense), 3 (allele A, antisense), or 4 (allele B, antisense).</td>
</tr>
<tr>
<td>20</td>
<td>'Direction'</td>
<td>Number identifying the direction of the probe pair. 1 = sense and 2 = antisense.</td>
</tr>
</tbody>
</table>
Note MATLAB software uses 1-based indexing for probe set numbers, while the Affymetrix CDF file uses 0-based indexing for probe set numbers. For example, CDFStruct.ProbeSets(1) has a ProbeSetNumber of 0 in the ProbePairs field.

\[
\text{PSValues} = \text{probesetvalues}(\text{CELStruct, CDFStruct, PS, 'Background', BackgroundValue})
\]
controls the background correction in the calculation. BackgroundValue can be:

- true (default) — Background values from the Background field in the PSValues matrix are used to calculate the probe intensity values.
- false — Background values are not calculated.
- A vector of precalculated background values (such as returned by the zonebackadj function) whose length is equal to the number of probes in CELStruct. These background values are used to calculate the probe intensity values.

Tip Including background correction in the calculation of the probe intensity values can be slow. Therefore, setting 'Background' to false can speed up the calculation. However, the values returned in the 'Background' field of the PSValues matrix will be zero.

ColumnNames = probesetvalues returns a cell array of character vectors containing the column names of the PSValues matrix. ColumnNames is returned only when you call probesetvalues without input arguments. The information contained in ColumnNames is common to all Affymetrix GeneChip arrays.

Examples

Retrieve Affymetrix™ Probe Set Intensity Values

This example uses sample data from the E. coli Antisense Genome Array. Download the data from Demo_Data_Ecoli-antisense.zip. Extract the data files from the DTT archive using the Data Transfer Tool.

You also need to download Ecoli_ASv2.CDF library file for the E. coli Antisense Genome Array. You may already have these files if you have any Affymetrix GeneChip software installed on your machine. If not, get the library files by downloading and unzipping the E. coli Antisense Genome Array zip file.

Read the contents of a CEL file into a MATLAB structure.

celStruct = affyread('Ecoli-antisense-121502.CEL');

Read the contents of a CDF file into a MATLAB structure.

cdfStruct = affyread('C:\LibFiles\Ecoli_ASv2.CDF');

Use the zonebackadj function to return a matrix or cell array of vectors containing the estimated background values for each probe.

[baData,zones,background] = zonebackadj(celStruct,'cdf',cdfStruct);

Create a table of intensity values for the argG_b3172_at probe set.
```matlab
psvals = probesetvalues(celStruct, cdfStruct, 'argG_b3172_at', ...
    'background', background)

psvals =
    1.0e+03 *

Columns 1 through 7

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2120</td>
<td>0</td>
<td>0</td>
<td>0.0454</td>
<td>0.4300</td>
<td>0.1770</td>
<td>0.1690</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0010</td>
<td>0</td>
<td>0.0455</td>
<td>0.4310</td>
<td>0.1770</td>
<td>0.1273</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0020</td>
<td>0</td>
<td>0.0455</td>
<td>0.4320</td>
<td>0.1770</td>
<td>0.1270</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0030</td>
<td>0</td>
<td>0.0455</td>
<td>0.4330</td>
<td>0.1770</td>
<td>0.1333</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0040</td>
<td>0</td>
<td>0.0455</td>
<td>0.4340</td>
<td>0.1770</td>
<td>0.2123</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0050</td>
<td>0</td>
<td>0.0455</td>
<td>0.4350</td>
<td>0.1770</td>
<td>0.1495</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0060</td>
<td>0</td>
<td>0.0455</td>
<td>0.4360</td>
<td>0.1770</td>
<td>0.1495</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0070</td>
<td>0</td>
<td>0.0456</td>
<td>0.4370</td>
<td>0.1770</td>
<td>0.1525</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0080</td>
<td>0</td>
<td>0.0456</td>
<td>0.4380</td>
<td>0.1770</td>
<td>0.1645</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0090</td>
<td>0</td>
<td>0.0456</td>
<td>0.4390</td>
<td>0.1770</td>
<td>0.1260</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0100</td>
<td>0</td>
<td>0.0456</td>
<td>0.4400</td>
<td>0.1770</td>
<td>0.0540</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0110</td>
<td>0</td>
<td>0.0456</td>
<td>0.4410</td>
<td>0.1770</td>
<td>0.0833</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0120</td>
<td>0</td>
<td>0.0457</td>
<td>0.4420</td>
<td>0.1770</td>
<td>0.0955</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0130</td>
<td>0</td>
<td>0.0457</td>
<td>0.4430</td>
<td>0.1770</td>
<td>0.1100</td>
</tr>
<tr>
<td>5.2120</td>
<td>0.0140</td>
<td>0</td>
<td>0.0457</td>
<td>0.4440</td>
<td>0.1770</td>
<td>0.2510</td>
</tr>
</tbody>
</table>

Columns 8 through 14

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0354</td>
<td>0.0250</td>
<td>0</td>
<td>0</td>
<td>0.4300</td>
<td>0.1780</td>
<td>0.1635</td>
<td></td>
</tr>
<tr>
<td>0.0218</td>
<td>0.0300</td>
<td>0</td>
<td>0</td>
<td>0.4310</td>
<td>0.1780</td>
<td>0.1063</td>
<td></td>
</tr>
<tr>
<td>0.0237</td>
<td>0.0300</td>
<td>0</td>
<td>0</td>
<td>0.4320</td>
<td>0.1780</td>
<td>0.1750</td>
<td></td>
</tr>
<tr>
<td>0.0259</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.4330</td>
<td>0.1780</td>
<td>0.0940</td>
<td></td>
</tr>
<tr>
<td>0.0433</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.4340</td>
<td>0.1780</td>
<td>0.1718</td>
<td></td>
</tr>
<tr>
<td>0.0275</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.4350</td>
<td>0.1780</td>
<td>0.1540</td>
<td></td>
</tr>
<tr>
<td>0.0112</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.4360</td>
<td>0.1780</td>
<td>0.0460</td>
<td></td>
</tr>
<tr>
<td>0.0377</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.4370</td>
<td>0.1780</td>
<td>0.1070</td>
<td></td>
</tr>
<tr>
<td>0.0312</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.4380</td>
<td>0.1780</td>
<td>0.0973</td>
<td></td>
</tr>
<tr>
<td>0.0234</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.4390</td>
<td>0.1780</td>
<td>0.1213</td>
<td></td>
</tr>
<tr>
<td>0.0112</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.4400</td>
<td>0.1780</td>
<td>0.0540</td>
<td></td>
</tr>
<tr>
<td>0.0174</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.4410</td>
<td>0.1780</td>
<td>0.0623</td>
<td></td>
</tr>
<tr>
<td>0.0171</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.4420</td>
<td>0.1780</td>
<td>0.0840</td>
<td></td>
</tr>
<tr>
<td>0.0196</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.4430</td>
<td>0.1780</td>
<td>0.0925</td>
<td></td>
</tr>
<tr>
<td>0.0460</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.4440</td>
<td>0.1780</td>
<td>0.1118</td>
<td></td>
</tr>
</tbody>
</table>

Columns 15 through 20

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0241</td>
<td>0.0300</td>
<td>0</td>
<td>0</td>
<td>0.0010</td>
<td>0.0020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0146</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.0010</td>
<td>0.0020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0286</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.0010</td>
<td>0.0020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0227</td>
<td>0.0300</td>
<td>0</td>
<td>0</td>
<td>0.0010</td>
<td>0.0020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0365</td>
<td>0.0300</td>
<td>0</td>
<td>0</td>
<td>0.0010</td>
<td>0.0020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0303</td>
<td>0.0300</td>
<td>0</td>
<td>0</td>
<td>0.0010</td>
<td>0.0020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0098</td>
<td>0.0250</td>
<td>0</td>
<td>0</td>
<td>0.0010</td>
<td>0.0020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0210</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.0010</td>
<td>0.0020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0219</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.0010</td>
<td>0.0020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0253</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.0010</td>
<td>0.0020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0129</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.0010</td>
<td>0.0020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0125</td>
<td>0.0360</td>
<td>0</td>
<td>0</td>
<td>0.0010</td>
<td>0.0020</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

0.0186  0.0300    0    0  0.0010  0.0020
0.0220  0.0360    0    0  0.0010  0.0020
0.0207  0.0360    0    0  0.0010  0.0020

See Also
affyread | celintensityread | probelibraryinfo | probesetlink | probesetlookup | probesetplot | rmabackadj | zonebackadj

Topics
“Working with Affymetrix® Data”

Introduced before R2006a
profalign

Align two profiles using Needleman-Wunsch global alignment

Syntax

\[
\text{Prof} = \text{profalign}(\text{Prof}1, \text{Prof}2)
\]

\[
[\text{Prof}, H1, H2] = \text{profalign}(\text{Prof}1, \text{Prof}2)
\]

\[
\text{profalign}(..., '\text{PropertyName}', \text{PropertyValue}, ...)
\]

Description

\text{Prof} = \text{profalign}(\text{Prof}1, \text{Prof}2) \quad \text{returns a new profile (Prof) for the optimal global alignment of two profiles (Prof}1, \text{Prof}2). \text{The profiles (Prof}1, \text{Prof}2) \text{are numeric arrays of size [((4 or 5 or 20 or 21) x Profile Length)] with counts or weighted profiles. Weighted profiles are used to down-weight similar sequences and up-weight divergent sequences. The output profile is a numeric matrix of size [(5 or 21) x New Profile Length] where the last row represents gaps. Original gaps in the input profiles are preserved. The output profile is the result of adding the aligned columns of the input profiles.}

\[
[\text{Prof}, H1, H2] = \text{profalign}(\text{Prof}1, \text{Prof}2) \quad \text{returns pointers that indicate how to rearrange the columns of the original profiles into the new profile.}
\]

\text{profalign}(..., '\text{PropertyName}', \text{PropertyValue}, ...) \quad \text{calls profalign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:}

\text{profalign}(..., '\text{PropertyName}', \text{PropertyValue}, ...) \quad \text{defines the scoring matrix to be used for the alignment.}

\text{ScoringMatrixValue} \quad \text{can be either of the following:}

- \text{Character vector or string specifying the scoring matrix to use for the alignment. Choices for amino acid sequences are:}
  - 'BLOSUM62'
  - 'BLOSUM30' increasing by 5 up to 'BLOSUM90'
  - 'BLOSUM100'
  - 'PAM10' increasing by 10 up to 'PAM500'
  - 'DAYHOFF'
  - 'GONNET'
Default is:

- ‘BLOSUM50’ — When AlphabetValue equals ‘AA’
- ‘NUC44’ — When AlphabetValue equals ‘NT’

**Note** The above scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the ‘Scale’ property to specify an additional scale factor to convert the output score from bits to another unit.

- Matrix representing the scoring matrix to use for the alignment, such as returned by the blosum, pam, dayhoff, gonnet, or nuc44 function.

**Note** If you use a scoring matrix that you created or was created by one of the above functions, the matrix does not include a scale factor. The output score will be returned in the same units as the scoring matrix.

**Note** If you need to compile profalign into a stand-alone application or software component using MATLAB Compiler, use a matrix instead of a character vector or string for ScoringMatrixValue.

profalign(..., 'GapOpen', {G1Value, G2Value}, ...) sets the penalties for opening a gap in the first and second profiles respectively. G1Value and G2Value can be either scalars or vectors. When using a vector, the number of elements is one more than the length of the input profile. Every element indicates the position specific penalty for opening a gap between two consecutive symbols in the sequence. The first and the last elements are the gap penalties used at the ends of the sequence. The default gap open penalties are {10, 10}.

profalign(..., 'ExtendGap', {E1Value, E2Value}, ...) sets the penalties for extending a gap in the first and second profile respectively. E1Value and E2Value can be either scalars or vectors. When using a vector, the number of elements is one more than the length of the input profile. Every element indicates the position specific penalty for extending a gap between two consecutive symbols in the sequence. The first and the last elements are the gap penalties used at the ends of the sequence. If ExtendGap is not specified, then extensions to gaps are scored with the same value as GapOpen.

profalign(..., 'ExistingGapAdjust', ExistingGapAdjustValue, ...), if ExistingGapAdjustValue is false, turns off the automatic adjustment based on existing gaps of the position-specific penalties for opening a gap. When ExistingGapAdjustValue is true (default), for every profile position, profalign proportionally lowers the penalty for opening a gap toward the penalty of extending a gap based on the proportion of gaps found in the contiguous symbols and on the weight of the input profile.

profalign(..., 'TerminalGapAdjust', TerminalGapAdjustValue, ...), when TerminalGapAdjustValue is true, adjusts the penalty for opening a gap at the ends of the sequence to be equal to the penalty for extending a gap. Default is false.

profalign(..., 'ShowScore', ShowScoreValue, ...), when ShowScoreValue is true, displays the scoring space and the winning path.
Examples

1. Read in sequences and create profiles.

   ```
   ma1 = ['RGTANCDMQDA';'RGTAHCDMQDA';'RRRAPCDL-DA'];
   ma2 = ['RGTHCDLADAT';'RGTACDMADAA'];
   p1 = seqprofile(ma1,'gaps','all','counts',true);
   p2 = seqprofile(ma2,'counts',true);
   ```

2. Merge two profiles into a single one by aligning them.

   ```
   p = profalign(p1,p2);
   seqlogo(p)
   ```

3. Use the output pointers to generate the multiple alignment.

   ```
   [p, h1, h2] = profalign(p1,p2);
   ma = repmat('-',5,12);
   ma(1:3,h1) = ma1;
   ma(4:5,h2) = ma2;
   disp(ma)
   ```

4. Increase the gap penalty before cysteine in the second profile.

   ```
   gapVec = 10 + [p2(aa2int('C'),:) 0] * 10
   p3 = profalign(p1,p2,'gapopen',{10,gapVec});
   seqlogo(p3)
   ```

5. Add a new sequence to a profile without inserting new gaps into the profile.

   ```
   gapVec = [0 inf(1,11) 0];
   p4 = profalign(p3,seqprofile('PLHFMSVLWDVQQWP'),...
                 'gapopen',{gapVec,10});
   seqlogo(p4)
   ```

See Also

multialign | seqconsensus

Topics

hmmprofalign on page 1-927
nwalign on page 1-1234
seqprofile on page 1-1530

Introduced before R2006a
proteinplot

Open Protein Plot window to investigate properties of amino acid sequence

**Syntax**

```matlab
proteinplot
proteinplot (SeqAA)
```

**Arguments**

<table>
<thead>
<tr>
<th>SeqAA</th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string containing single-letter codes specifying an amino acid sequence. For valid letter codes, see the table Mapping Amino Acid Letter Codes to Integers. Unknown characters are mapped to 0.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that contains an amino acid sequence, such as returned by <code>fastaread</code>, <code>getgenpept</code>, <code>genpeptread</code>, <code>getpdb</code>, or <code>pdbread</code>.</td>
</tr>
</tbody>
</table>

**Description**

The Protein Plot window lets you analyze and compare properties of a single amino acid sequence. It displays smoothed line plots of various properties such as the hydrophobicity of the amino acids in the sequence.

`proteinplot` opens the Protein Plot window.

`proteinplot (SeqAA)` opens the Protein Plot window and loads `SeqAA`, an amino acid sequence, into the window.

**Tip** You can analyze and compare properties of an amino acid sequence from the MATLAB command line also by using the `proteinpropplot` function.

**Examples**

**Example 1.63. Importing Sequences into the Protein Plot Window**

You can import a sequence into the Protein Plot window from the MATLAB command line.

1. Retrieve an amino acid sequence from the Protein Data Bank (PDB) database.
   ```matlab
   prion = getpdb('1HJM', 'SEQUENCEONLY', true);
   ```
2. Load the amino acid sequence into the Protein Plot window.
   ```matlab
   proteinplot(prion)
   ```

   The Protein Plot window opens, and the sequence appears in the **Sequence** text box.
You can import a sequence after the Protein Plot window is open by doing either of the following:

- Type or paste an amino acid sequence into the **Sequence** text box.
- Click the **Import Sequence** button to open the Import dialog box. From the **Import From** list, select one of the following:
  - **Workspace** — To select a variable from the MATLAB Workspace
  - **Text File** — To select a text file
  - **FASTA File** — To select a FASTA-formatted file
  - **GenPept File** — To select a GenPept-formatted file
  - **GenPept Database** — To specify an accession number in the GenPept database

**Example 1.64. Viewing Properties of Amino Acids**

Select a property from the **Properties** drop-down list box to display a smoothed plot of the property values along the sequence. You can select multiple properties from the list by holding down **Shift** or **Ctrl** while selecting properties. When you select two properties, the plots are displayed using two y-axes, with one y-axis on the left and one on the right. For all other selections, a single y-axis is displayed. When displaying one or two properties, the y values displayed are the actual property values. When displaying three or more properties, the values are normalized to the range 0-1.
**Example 1.65. Accessing Information About the Properties**

You can access information about the properties from the Help menu.

1. Select Help > References. The Help browser opens with a list of properties and references.
2. Scroll down to locate the property of interest.

**Example 1.66. Using Other Features in the Protein Plot Window**

The Terminal Selection boxes (N and C) let you choose to plot only part of the sequence. By default, all of the sequence is plotted.

You can add your own properties by clicking on the Add button next to the Properties list. This opens a Property dialog box that lets you specify the value for each of the amino acids. The Display Text box lets you specify the text that will be displayed in the Properties list on the main Protein Plot window. You can also save the property values to a file for future use by typing a file name in the Filename text box.

The default smoothing method is an unweighted linear moving average with a window length of five residues. You can change this by selecting Edit > Filter Window Options. The dialog box lets you select the Window Size from 5 to 29 residues. Increasing the window size produces a smoother plot. You can modify the shape of the smoothing window by changing the Edge Weight factor. And you can choose the smoothing function to be a linear moving average, an exponential moving average or a linear Lowess smoothing.

The File menu lets you import a sequence, save the plot that you have created to a Figure file, export the data values in the figure to a workspace variable or to a MAT-file, export the figure to a normal Figure window for customizing, or print the figure.

The Edit menu lets you create a new property, to reset the property values to the default values, and to modify the smoothing parameters with the Configuration Values menu item.

The View menu lets you turn the toolbar on and off, and to add a legend to the plot.

The Tools menu lets you zoom in and zoom out of the plot, to view Data Statistics such as mean, minimum and maximum values of the plot, and to normalize the values of the plot from 0 to 1.

The Help menu lets you view this document and to see the references for the sequence properties included with the Protein Plot window.

**See Also**

aacount | atomiccomp | molviewer | molweight | pdbdistplot | proteinpropplot | seqviewer | yyaxis

**Introduced before R2006a**
proteinpropplot

Plot properties of amino acid sequence

Syntax

proteinpropplot (SeqAA)

proteinpropplot(SeqAA, ...'PropertyTitle', PropertyTitleValue, ...)
proteinpropplot(SeqAA, ...'Startat', StartatValue, ...)
proteinpropplot(SeqAA, ...'Endat', EndatValue, ...)
proteinpropplot(SeqAA, ...'Smoothing', SmoothingValue, ...)
proteinpropplot(SeqAA, ...'EdgeWeight', EdgeWeightValue, ...)
proteinpropplot(SeqAA, ...'WindowLength', WindowLengthValue, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeqAA</td>
<td>Amino acid sequence. Enter any of the following:</td>
</tr>
<tr>
<td></td>
<td>• Character vector or string containing letters representing an amino acid</td>
</tr>
<tr>
<td></td>
<td>• Vector of integers representing an amino acid, such as returned by aa2int</td>
</tr>
<tr>
<td></td>
<td>• Structure containing a Sequence field that contains an amino acid sequence, such as returned by getembl, getgenpept, or getpdb</td>
</tr>
<tr>
<td>PropertyTitleValue</td>
<td>Character vector or string that specifies the property to plot. Default is Hydrophobicity (Kyte &amp; Doolittle). To display a list of properties to plot, enter a empty character vector or empty string for PropertyTitleValue. For example, type:</td>
</tr>
<tr>
<td></td>
<td>proteinpropplot(sequence, 'propertytitle', '')</td>
</tr>
<tr>
<td></td>
<td>Tip To access references for the properties, view the proteinpropplot file.</td>
</tr>
<tr>
<td>StartatValue</td>
<td>Integer that specifies the starting point for the plot from the N-terminal end of the amino acid sequence SeqAA. Default is 1.</td>
</tr>
<tr>
<td>EndatValue</td>
<td>Integer that specifies the ending point for the plot from the N-terminal end of the amino acid sequence SeqAA. Default is length(SeqAA).</td>
</tr>
<tr>
<td>SmoothingValue</td>
<td>Character vector or string the specifies the smoothing method. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• linear (default)</td>
</tr>
<tr>
<td></td>
<td>• exponential</td>
</tr>
<tr>
<td></td>
<td>• lowess</td>
</tr>
</tbody>
</table>
**Description**

`proteinpropplot` (SeqAA) displays a plot of the hydrophobicity (Kyte and Doolittle, 1982 on page 1-1349) of the residues in sequence SeqAA.

`proteinpropplot(SeqAA, ...'PropertyName', PropertyValue, ...)` calls `proteinpropplot` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

- `proteinpropplot(SeqAA, ...'PropertyTitle', PropertyTitleValue, ...)` specifies a property to plot for the amino acid sequence SeqAA. Default is Hydrophobicity (Kyte & Doolittle). To display a list of possible properties to plot, enter an empty character vector or empty string for `PropertyTitleValue`. For example, type:

  ```
  proteinpropplot(sequence, 'propertytitle', '')
  ```

  **Tip** To access references for the properties, view the `proteinpropplot` file.

- `proteinpropplot(SeqAA, ...'Startat', StartatValue, ...)` specifies the starting point for the plot from the N-terminal end of the amino acid sequence SeqAA. Default is 1.

- `proteinpropplot(SeqAA, ...'Endat', EndatValue, ...)` specifies the ending point for the plot from the N-terminal end of the amino acid sequence SeqAA. Default is `length(SeqAA)`.

- `proteinpropplot(SeqAA, ...'Smoothing', SmoothingValue, ...)` specifies the smoothing method. Choices are:
  - linear (default)
  - exponential
  - lowess

- `proteinpropplot(SeqAA, ...'EdgeWeight', EdgeWeightValue, ...)` specifies the edge weight used for linear and exponential smoothing methods. Decreasing this value emphasizes peaks in the plot. Choices are any value \( \geq 0 \) and \( \leq 1 \). Default is 1.

- `proteinpropplot(SeqAA, ...'WindowLength', WindowLengthValue, ...)` specifies the window length for the smoothing method. Increasing this value gives a smoother plot that shows less detail. Default is 11.
Examples

Example 1.67. Plotting Hydrophobicity

1. Use the `getpdb` function to retrieve a protein sequence.
   ```matlab
   prion = getpdb('1HJM', 'SEQUENCEONLY', true);
   ```
2. Plot the hydrophobicity (Kyte and Doolittle, 1982 on page 1-1349) of the residues in the sequence.
   ```matlab
   proteinpropplot(prion)
   ```

Example 1.68. Plotting Parallel Beta Strand

1. Use the `getgenpept` function to retrieve a protein sequence.
   ```matlab
   s = getgenpept('aad50640');
   ```
2. Plot the conformational preference for parallel beta strand for the residues in the sequence.
   ```matlab
   proteinpropplot(s,'propertytitle','Parallel beta strand')
   ```
References


See Also
aaccount | atomiccomp | molviewer | molweight | pdbdistplot | plotyy | proteinplot | ramachandran | seqviewer

Introduced in R2007a
prune (phytree)

Remove branch nodes from phylogenetic tree

**Syntax**

\[
T2 = \text{prune}(T1, \text{Nodes})
\]
\[
T2 = \text{prune}(T1, \text{Nodes}, \text{Mode}', \text{Exclusive}')
\]

**Arguments**

<table>
<thead>
<tr>
<th>(T1)</th>
<th>Phylogenetic object created with the <code>phytree</code> constructor function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{Nodes})</td>
<td>Nodes to remove from tree.</td>
</tr>
<tr>
<td>(\text{Mode})</td>
<td>Property to control the method of pruning. Enter either 'Inclusive' or 'Exclusive'. The default value is 'Inclusive'.</td>
</tr>
</tbody>
</table>

**Description**

\(T2 = \text{prune}(T1, \text{Nodes})\) removes the nodes listed in the vector \(\text{Nodes}\) from the tree \(T1\). \(\text{prune}\) removes any branch or leaf nodes listed in \(\text{Nodes}\) and all their descendants from the tree \(T1\), and returns the modified tree \(T2\). The parent nodes are connected to the 'brothers' as required. Nodes in the tree are labeled as \([1: \text{numLeaves}]\) for the leaves and as \([\text{numLeaves}+1: \text{numLeaves}+\text{numBranches}]\) for the branches. \(\text{Nodes}\) can also be a logical array of size \([\text{numLeaves}+\text{numBranches} \times 1]\) indicating the nodes to be removed.

\(T2 = \text{prune}(T1, \text{Nodes}, \text{Mode}', \text{Exclusive}')\) changes the \(\text{Mode}\) property for pruning to 'Exclusive' and removes only the descendants of the nodes listed in the vector \(\text{Nodes}\). Nodes that do not have a predecessor become leaves in the list \(\text{Nodes}\). In this case, pruning is the process of reducing a tree by turning some branch nodes into leaf nodes, and removing the leaf nodes under the original branch.

**Examples**

Load a phylogenetic tree created from a protein family

```matlab
tr = phytreeread('pf00002.tree');
view(tr)
```
Remove all the 'mouse' proteins

\[
\text{ind} = \text{getbyname}(\text{tr}, \text{'mouse'});
\]
\[
\text{tr} = \text{prune}(\text{tr}, \text{ind});
\]
\[
\text{view}(\text{tr})
\]
Remove potential outliers in the tree

\[ \text{[sel, sel_leaves]} = \text{select(tr,'criteria','distance',...}
\quad \text{'threshold', 3, ...}
\quad \text{'reference', 'leaves', ...}
\quad \text{'exclude', 'leaves', ...}
\quad \text{'propagate', 'tolleaves');} \]

\text{tr = prune(tr, ~sel_leaves)}
\text{view(tr)}
See Also
get | phytree | phytreeviewer | select

Topics
phytree object on page 1-1274

Introduced before R2006a
pubMedID

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set PubMed IDs in ExpressionSet object

Syntax

PMIDs = pubMedID(ESObj)
NewESObj = pubMedID(ESObj, NewPMIDs)

Description

PMIDs = pubMedID(ESObj) returns a character vector or cell array of character vectors containing the PubMed IDs from a MIAME object in an ExpressionSet object.

NewESObj = pubMedID(ESObj, NewPMIDs) replaces the PubMed IDs in the MIAME object in ESObj, an ExpressionSet object, with NewPMIDs, a character vector or cell array of character vectors specifying new PubMed IDs, and returns NewESObj, a new ExpressionSet object.

Input Arguments

ESObj
Object of the bioma.ExpressionSet class.

Default:

NewPMIDs
Character vector or cell array of character vectors containing new PubMed IDs.

Default:

Output Arguments

PMIDs
Character vector or cell array of character vectors containing the PubMed IDs from a MIAME object in an ExpressionSet object.

NewESObj
Object of the bioma.ExpressionSet class, returned after replacing the PubMed IDs.

Examples

Construct an ExpressionSet object, ESObj, as described in the “Examples” on page 1-0 section of the bioma.ExpressionSet class reference page. Retrieve the PubMed identifiers stored in the MIAME object stored in the ExpressionSet object:
% Retrieve PubMed IDs from the MIAME object
PMIDs = pubMedID(ESObj)

See Also
bioma.ExpressionSet | bioma.data.MIAME

Topics
“Managing Gene Expression Data in Objects”

External Websites
quantilenorm

Quantile normalization over multiple arrays

Syntax

\[
\text{NormData} = \text{quantilenorm}(\text{Data})
\]
\[
\text{NormData} = \text{quantilenorm}(\ldots, '\text{MEDIAN}', \text{true})
\]
\[
\text{NormData} = \text{quantilenorm}(\ldots, '\text{DISPLAY}', \text{true})
\]

Description

\[
\text{NormData} = \text{quantilenorm}(\text{Data})
\]
where the columns of Data correspond to separate chips, normalizes the distributions of the values in each column.

\underline{Note} If Data contains NaN values, then NormData will also contain NaN values at the corresponding positions.

\[
\text{NormData} = \text{quantilenorm}(\ldots, '\text{MEDIAN}', \text{true})
\]
takes the median of the ranked values instead of the mean.

\[
\text{NormData} = \text{quantilenorm}(\ldots, '\text{DISPLAY}', \text{true})
\]
plots the distributions of the columns and of the normalized data.

Examples

\[
\text{load yeastdata}
\]
\[
\text{normYeastValues} = \text{quantilenorm(yeastvalues, 'display', 1)};
\]

See Also

affygcrma | affyrma | malowess | manorm | rmabackadj | rmasummary

Introduced before R2006a
**ramachandran**

Draw Ramachandran plot for Protein Data Bank (PDB) data

**Syntax**

```matlab
ramachandran(PDBid)
ramachandran(File)
ramachandran(PDBStruct)
RamaStruct = ramachandran(...)

ramachandran(..., 'Chain', ChainValue, ...)
ramachandran(..., 'Plot', PlotValue, ...)
ramachandran(..., 'Model', ModelValue, ...)
ramachandran(..., 'Glycine', GlycineValue, ...)
ramachandran(..., 'Regions', RegionsValue, ...)
ramachandran(..., 'RegionDef', RegionDefValue, ...)
```

**Input Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDBid</td>
<td>Character vector or string specifying a unique identifier for a protein</td>
</tr>
<tr>
<td></td>
<td>structure record in the PDB database.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> Each structure in the PDB database is represented by a four-</td>
</tr>
<tr>
<td></td>
<td>character alphanumeric identifier. For example, 4hhb is the identifier</td>
</tr>
<tr>
<td></td>
<td>for hemoglobin.</td>
</tr>
<tr>
<td>File</td>
<td>Character vector or string specifying a file name or a path and file name.</td>
</tr>
<tr>
<td></td>
<td>The referenced file is a Protein Data Bank (PDB)-formatted file. If you</td>
</tr>
<tr>
<td></td>
<td>specify only a file name, that file must be on the MATLAB search path or</td>
</tr>
<tr>
<td></td>
<td>in the MATLAB Current Directory.</td>
</tr>
<tr>
<td>PDBStruct</td>
<td>MATLAB structure containing PDB-formatted data, such as returned by</td>
</tr>
<tr>
<td></td>
<td>getpdb or pdbread.</td>
</tr>
<tr>
<td>ChainValue</td>
<td>Character vector, string, string vector, or cell array of character vectors</td>
</tr>
<tr>
<td></td>
<td>that specifies the chain(s) to compute the torsion angles for and plot.</td>
</tr>
<tr>
<td></td>
<td><strong>Choices are:</strong></td>
</tr>
<tr>
<td></td>
<td>• ‘All’ (default) — Torsion angles for all chains are computed and plotted.</td>
</tr>
<tr>
<td></td>
<td>• A character vector or string specifying the chain ID, which is case</td>
</tr>
<tr>
<td></td>
<td>sensitive.</td>
</tr>
<tr>
<td></td>
<td>• A cell array of character vectors or string vector specifying chain</td>
</tr>
<tr>
<td></td>
<td>IDs, which are case sensitive.</td>
</tr>
<tr>
<td><strong>PlotValue</strong></td>
<td>Character vector or string specifying how to plot chains. Choices are:</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------</td>
</tr>
<tr>
<td>• 'None' — Plots nothing.</td>
<td></td>
</tr>
<tr>
<td>• 'Separate' — Plots torsion angles for all specified chains in separate plots.</td>
<td></td>
</tr>
<tr>
<td>• 'Combined' (default) — Plots torsion angles for all specified chains in one combined plot.</td>
<td></td>
</tr>
</tbody>
</table>

| **ModelValue** | Integer that specifies the structure model to consider. Default is 1. |
| **GlycineValue** | Controls the highlighting of glycine residues with a circle in the plot. Choices are true or false (default). |

| **RegionsValue** | Controls the drawing of Ramachandran reference regions in the plot. Choices are true or false (default). |

The default regions are core right-handed alpha, core beta, core left-handed alpha, and allowed, with the core regions corresponding to data points of preferred values of psi/phi angle pairs, and the allowed regions corresponding to possible, but disfavored values of psi/phi angle pairs, based on simple energy considerations. The boundaries of these default regions are based on the calculations by Morris et al., 1992.

**Note** If using the default colormap, red = right-handed core alpha, core beta, and core left-handed alpha, while yellow = allowed.

| **RegionDefValue** | MATLAB structure or array of structures (if specifying multiple regions) containing information (name, color, and boundaries) for custom reference regions in a Ramachandran plot. Each structure must contain the following fields: |

- **Name** — Character vector or string specifying a name for the region.
- **Color** — Character vector or string or three-element numeric vector of RGB values specifying a color for the region in the plot.
- **Patch** — A 2-by-N matrix of values, the first row containing torsion angle phi (Φ) values, and the second row containing torsion angle psi (Ψ) values. When psi/phi angle pairs are plotted, the data points specify boundaries for the region. N is the number of data points needed to define the region.

**Tip** If you specify custom reference regions in which a smaller region is contained or covered by a larger region, list the structure for the smaller region first in the array so that it is plotted last and visible in the plot.
Output Arguments

<table>
<thead>
<tr>
<th>RamaStruct</th>
<th>MATLAB structure or array of structures (if protein contains multiple chains). Each structure contains the following fields:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Angles</td>
</tr>
<tr>
<td></td>
<td>• ResidueNum</td>
</tr>
<tr>
<td></td>
<td>• ResidueName</td>
</tr>
<tr>
<td></td>
<td>• Chain</td>
</tr>
<tr>
<td></td>
<td>• HPoints</td>
</tr>
<tr>
<td></td>
<td>For descriptions of the fields, see the following table.</td>
</tr>
</tbody>
</table>

Description

A Ramachandran plot is a plot of the torsion angle phi, Φ, (torsion angle between the C-N-CA-C atoms) versus the torsion angle psi, Ψ, (torsion angle between the N-CA-C-N atoms) for each residue of a protein sequence.

`ramachandran(PDBid)` generates the Ramachandran plot for the protein specified by the PDB database identifier `PDBid`.

`ramachandran(File)` generates the Ramachandran plot for the protein specified by `File`, a PDB-formatted file.

`ramachandran(PDBStruct)` generates the Ramachandran plot for the protein stored in `PDBStruct`, a MATLAB structure containing PDB-formatted data, such as returned by `getpdb` or `pdbread`.

`RamaStruct = ramachandran(...)` returns a MATLAB structure or array of structures (if protein contains multiple chains). Each structure contains the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angles</td>
<td>Three-column matrix containing the torsion angles phi (Φ), psi (Ψ), and omega (ω) for each residue in the sequence, ordered by residue sequence number. The number of rows in the matrix is equal to the number of rows in the ResidueNum column vector, which can be used to determine which residue corresponds to each row in the Angles matrix.</td>
</tr>
</tbody>
</table>

**Note** The Angles matrix contains a row for each number in the range of residue sequence numbers, including residue sequence numbers missing from the PDB file. Rows corresponding to residue sequence numbers missing from the PDB file contain the value NaN.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResidueNum</td>
<td>Column vector containing the residue sequence numbers from the PDB file.</td>
</tr>
</tbody>
</table>
|               | **Note** The ResidueNum vector starts with one of the following:  
|               | • The lowest residue sequence number (if the lowest residue sequence number is negative or zero)  
|               | • The number 1 (if the lowest residue sequence number is positive)  
|               | The ResidueNum vector ends with the highest residue sequence number and includes all numbers in the range, including residue sequence numbers missing from the PDB file. |
| ResidueName   | Column vector containing the residue names for the protein.                                                                                   |
| Chain         | A character vector or string specifying the chains in the protein.                                                                             |
| HPoints       | Handle to the data points in the plot.                                                                                                       |

`ramachandran(..., 'PropertyName', PropertyValue, ...)` calls `ramachandran` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`ramachandran(..., 'Chain', ChainValue, ...)` specifies the chain(s) to compute the torsion angles for and plot. Choices are:
- 'All' (default) — Torsion angles for all chains are computed and plotted.
- A character vector or string specifying the chain ID, which is case sensitive.
- A cell array of character vectors or string vector specifying chain IDs, which are case sensitive.

`ramachandran(..., 'Plot', PlotValue, ...)` specifies how to plot chains. Choices are:
- 'None' — Plots nothing.
- 'Separate' — Plots torsion angles for all specified chains in separate plots.
- 'Combined' (default) — Plots torsion angles for all specified chains in one combined plot.

`ramachandran(..., 'Model', ModelValue, ...)` specifies the structure model to consider. Default is 1.

`ramachandran(..., 'Glycine', GlycineValue, ...)` controls the highlighting of glycine residues with a circle in the plot. Choices are true or false (default).

`ramachandran(..., 'Regions', RegionsValue, ...)` controls the drawing of Ramachandran reference regions in the plot. Choices are true or false (default).
The default regions are core right-handed alpha, core beta, core left-handed alpha, and allowed, with the core regions corresponding to data points of preferred values of psi/phi angle pairs, and the allowed regions corresponding to possible, but disfavored values of psi/phi angle pairs, based on simple energy considerations. The boundaries of these default regions are based on the calculations by Morris et al., 1992.

**Note** If using the default colormap, then red = core right-handed alpha, core beta, and core left-handed alpha, while yellow = allowed.

`ramachandran(..., 'RegionDef', RegionDefValue, ...)` specifies information (name, color, and boundary) for custom reference regions in a Ramachandran plot. `RegionDefValue` is a MATLAB structure or array of structures containing the following fields:

- **Name** — Character vector or string specifying a name for the region.
- **Color** — Character vector or string or three-element numeric vector of RGB values specifying a color for the region in the plot.
- **Patch** — A 2-by-N matrix of values, the first row containing torsion angle phi (Φ) values, and the second row containing torsion angle psi (Ψ) values. When psi/phi angle pairs are plotted, the data points specify a boundary for the region. N is the number of data points needed to define the region.

**Tip** If you specify custom reference regions in which a smaller region is contained or covered by a larger region, list the structure for the smaller region first in the array so that it is plotted last and visible in the plot.

**Examples**

**Example 1.69. Drawing a Ramachandran Plot**

Draw the Ramachandran plot for the human serum albumin complexed with octadecanoic acid, which has a PDB database identifier of `1E7I`.

```matlab
ramachandran('1E7I')
```
Example 1.70. Drawing a Ramachandran Plot for a Specific Chain

1. Use the `getpdb` function to retrieve protein structure data for the human growth hormone from the PDB database, and save the information to a file.

   ```matlab
getpdb('1a22','ToFile','1a22.pdb');
```

2. Compute the torsion angles and draw the Ramachandran plot for chain A of the human growth hormone, represented in the pdb file, `1a22.pdb`.

   ```matlab
   ChainA1a22Struct = ramachandran('1a22.pdb','chain','A')
   ChainA1a22Struct =
   Angles: [191x3 double]
   ResidueNum: [191x1 double]
   ResidueName: {191x1 cell}
   Chain: 'A'
   HPoints: 370.0012
Example 1.71. Drawing Ramachandran Plots with Highlighted Glycine Residues and Ramachandran Regions

1. Use the `getpdb` function to retrieve protein structure data for the human growth hormone from the PDB database, and store the information in a structure.

   ```matlab
   Struct1a22 = getpdb('1a22');
   ```

2. Draw a combined Ramachandran plot for all chains of the human growth hormone, represented in the pdb structure, `1a22Struct`. Highlight the glycine residues (with a circle), and draw the reference Ramachandran regions in the plot.

   ```matlab
   ramachandran(Struct1a22,'glycine',true,'regions',true);
   ```
Tip Click a data point to display a data tip with information about the residue. Click a region to display a data tip defining the region. Press and hold the Alt key to display multiple data tips.

3 Draw a separate Ramachandran plot for each chain of the human growth hormone, represented in the pdb structure, 1a22Struct. Highlight the glycine residues (with a circle) and draw the reference Ramachandran regions in the plot.

ramachandran(Struct1a22,'plot','separate','chain','all',... 'glycine',true,'regions',true)
Example 1.72. Writing a Tab-Delimited Report File from a Ramachandran Structure

1 Create an array of two structures containing torsion angles for chains A and D in the Calcium/Calmodulin-dependent protein kinase, which has a PDB database identifier of 1hkx.

```matlab
a = ramachandran('1hkx', 'chain', {'A', 'D'})
```
a =

1x2 struct array with fields:
Angles
ResidueNum
ResidueName
Chain
HPoints

Write a tab-delimited report file containing torsion angles phi (Φ) and psi (Ψ) for chains A and D in the Calcium/Calmodulin-dependent protein kinase.

fid = fopen('rama_1hkx_report.txt', 'wt');

for c = 1:numel(a)
    for i = 1:length(a(c).Angles)
        if ~all(isnan(a(c).Angles(i,:)))
            fprintf(fid, '%s	%d	%s	%f	%f
', a(c).Chain, ...
                    a(c).ResidueNum(i), a(c).ResidueName{i}, ...
                    a(c).Angles(i,1:2));
        end
    end
end
fclose(fid);

View the file you created in the MATLAB Editor.

edit rama_1hkx_report.txt

References


See Also
getpdb | molviewer | pdbdistplot | pdbread | proteinpropplot
Introduced before R2006a
randfeatures

Generate randomized subset of features

Syntax

[IDX, Z] = randfeatures(X, Group, 'PropertyName', PropertyValue...)
randfeatures(..., 'Classifier', C)
randfeatures(..., 'ClassOptions', CO)
randfeatures(..., 'PerformanceThreshold', PT)
randfeatures(..., 'ConfidenceThreshold', CT)
randfeatures(..., 'SubsetSize', SS)
randfeatures(..., 'PoolSize', PS)
randfeatures(..., 'NumberOfIndices', N)
randfeatures(..., 'CrossNorm', CN)
randfeatures(..., 'Verbose', VerboseValue)

Description

[IDX, Z] = randfeatures(X, Group, 'PropertyName', PropertyValue...) performs a randomized subset feature search reinforced by classification. randfeatures randomly generates subsets of features used to classify the samples. Every subset is evaluated with the apparent error. Only the best subsets are kept, and they are joined into a single final pool. The cardinality for every feature in the pool gives the measurement of the significance.

X contains the training samples. Every column of X is an observed vector. Group contains the class labels. Group can be a numeric vector, a cell array of character vectors or string vector; numel(Group) must be the same as the number of columns in X, and numel(unique(Group)) must be greater than or equal to 2. Z is the classification significance for every feature. IDX contains the indices after sorting Z; i.e., the first one points to the most significant feature.

randfeatures(..., 'Classifier', C) sets the classifier. Options are

'da' (default) Discriminant analysis
'knn' K nearest neighbors

randfeatures(..., 'ClassOptions', CO) is a cell with extra options for the selected classifier. When you specify the discriminant analysis model ('da') as a classifier, randfeatures uses the classify function with its default parameters. For the KNN classifier, randfeatures uses fitcknn with the following default options. {'Distance','correlation','NumNeighbors',5}.

randfeatures(..., 'PerformanceThreshold', PT) sets the correct classification threshold used to pick the subsets included in the final pool. For the 'da' model, the default is 0.8. For the 'knn' model, the default is 0.7.

randfeatures(..., 'ConfidenceThreshold', CT) uses the posterior probability of the discriminant analysis to invalidate classified subvectors with low confidence. When using the 'da' model, the default is 0.95.^{(number of classes)}. When using the 'knn' model, the default is 1, meaning any classified subvector must have all k neighbors classified to the same class in order to be kept in the pool.
randfeatures(..., 'SubsetSize', SS) sets the number of features considered in every subset. Default is 20.

randfeatures(..., 'PoolSize', PS) sets the targeted number of accepted subsets for the final pool. Default is 1000.

randfeatures(..., 'NumberOfIndices', N) sets the number of output indices in IDX. Default is the same as the number of features.

randfeatures(..., 'CrossNorm', CN) applies independent normalization across the observations for every feature. Cross-normalization ensures comparability among different features, although it is not always necessary because the selected classifier properties might already account for this. Options are

- 'none' (default) Intensities are not cross-normalized.
- 'meanvar' \( x_{\text{new}} = (x - \text{mean}(x))/\text{std}(x) \)
- 'softmax' \( x_{\text{new}} = (1+\exp((\text{mean}(x)-x)/\text{std}(x)))^{-1} \)
- 'minmax' \( x_{\text{new}} = (x - \text{min}(x))/(\text{max}(x)-\text{min}(x)) \)

randfeatures(..., 'Verbose', VerboseValue), when Verbose is true, turns off verbosity. Default is true.

**Examples**

Find a reduced set of genes that is sufficient for classification of all the cancer types in the t-matrix NCI60 data set. Load sample data.

```matlab
load NCI60tmatrix
```

Select features.

```matlab
I = randfeatures(X,GROUP,'SubsetSize',15,'Classifier','da');
```

Test features with a linear discriminant classifier.

```matlab
C = classify(X(I(1:25),:)',X(I(1:25),:)',GROUP);
cp = classperf(GROUP,C);
cp.CorrectRate
```

ans =

1

**References**


See Also
classify | classperf | crossvalind | rankfeatures | sequentialfs

Introduced before R2006a
randseq

Generate random sequence from finite alphabet

Syntax

Seq = randseq(SeqLength)

Seq = randseq(SeqLength, ...'Alphabet', AlphabetValue, ...)
Seq = randseq(SeqLength, ...'Weights', WeightsValue, ...)
Seq = randseq(SeqLength, ...'FromStructure', FromStructureValue, ...)
Seq = randseq(SeqLength, ...'Case', CaseValue, ...)
Seq = randseq(SeqLength, ...'DataType', DataTypeValue, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeqLength</td>
<td>Integer that specifies the number of nucleotides or amino acids in the random sequence.</td>
</tr>
<tr>
<td>AlphabetValue</td>
<td>Character vector or string that specifies the alphabet for the sequence. Choices are 'dna'(default), 'rna', or 'amino'.</td>
</tr>
<tr>
<td>WeightsValue</td>
<td>Property to specify a weighted random sequence.</td>
</tr>
<tr>
<td>FromStructureValue</td>
<td>Property to specify a weighted random sequence using output structures from the functions from basecount, dimercount, codoncount, or aacount.</td>
</tr>
<tr>
<td>CaseValue</td>
<td>Character vector or string that specifies the case of letters in a sequence when Alphabet is 'char'. Choices are 'upper'(default) or 'lower'.</td>
</tr>
<tr>
<td>DataTypeValue</td>
<td>Character vector or string that specifies the data type for a sequence. Choices are 'char'(default) for letter sequences, and 'uint8' or 'double' for numeric sequences. Creates a sequence as an array of DataType.</td>
</tr>
</tbody>
</table>

Description

Seq = randseq(SeqLength) creates a random sequence with a length specified by SeqLength.

Seq = randseq(SeqLength, ...'PropertyName', PropertyValue, ...) calls randseq with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Seq = randseq(SeqLength, ...'Alphabet', AlphabetValue, ...) generates a sequence from a specific alphabet.

Seq = randseq(SeqLength, ...'Weights', WeightsValue, ...) creates a weighted random sequence where the ith letter of the sequence alphabet is selected with weight W(i). The
weight vector is usually a probability vector or a frequency count vector. Note that the ith element of the nucleotide alphabet is given by int2nt(i), and the ith element of the amino acid alphabet is given by int2aa(i).

\[ \text{Seq} = \text{randseq}(\text{SeqLength}, \ldots \text{'FromStructure'}, \text{FromStructureValue}, \ldots) \]
creates a weighted random sequence with weights given by the output structure from basecount, dimercount, codoncount, or aacount.

\[ \text{Seq} = \text{randseq}(\text{SeqLength}, \ldots \text{'Case'}, \text{CaseValue}, \ldots) \]
specifies the case for a letter sequence.

\[ \text{Seq} = \text{randseq}(\text{SeqLength}, \ldots \text{'DataType'}, \text{DataTypeValue}, \ldots) \]
specifies the data type for the sequence array.

**Examples**

Generate a random DNA sequence.

```
randseq(20)
```

```
an = TAGCTGGCCAAGCGAGCTTG
```

Generate a random RNA sequence.

```
rndseq(20, 'alphabet', 'rna')
```

```
an = GCUGCGGCGGUUGUAUCUG
```

Generate a random protein sequence.

```
rndseq(20, 'alphabet', 'amino')
```

```
an = DYKMCLEYFEGFHFTGHKK
```

**See Also**
hmmgenerate | rand | randperm | randsample

**Introduced before R2006a**
rankfeatures

Rank key features by class separability criteria

Syntax

\[
[IDX, Z] = \text{rankfeatures}(X, \text{Group})
\]

\[
[IDX, Z] = \text{rankfeatures}(X, \text{Group}, \ldots \text{'Criterion'}, \text{CriterionValue}, \ldots)
\]

\[
[IDX, Z] = \text{rankfeatures}(X, \text{Group}, \ldots \text{'CCWeighting'}, \text{ALPHA}, \ldots)
\]

\[
[IDX, Z] = \text{rankfeatures}(X, \text{Group}, \ldots \text{'NWeighting'}, \text{BETA}, \ldots)
\]

\[
[IDX, Z] = \text{rankfeatures}(X, \text{Group}, \ldots \text{'NumberOfIndices'}, \text{N}, \ldots)
\]

\[
[IDX, Z] = \text{rankfeatures}(X, \text{Group}, \ldots \text{'CrossNorm'}, \text{CN}, \ldots)
\]

Description

\[
[IDX, Z] = \text{rankfeatures}(X, \text{Group})
\]

ranks the features in \(X\) using an independent evaluation criterion for binary classification. \(X\) is a matrix where every column is an observed vector and the number of rows corresponds to the original number of features. \(\text{Group}\) contains the class labels.

\(IDX\) is the list of indices to the rows in \(X\) with the most significant features. \(Z\) is the absolute value of the criterion used (see below).

\(\text{Group}\) can be a numeric vector, a cell array of character vectors or string vector. \numel(\text{Group})\ is the same as the number of columns in \(X\), and \(\text{Group}\) must have only two unique values. If it contains any NaN values, the function ignores the corresponding observation vector in \(X\).

\[
[IDX, Z] = \text{rankfeatures}(X, \text{Group}, \ldots \text{'PropertyName'}, \text{PropertyValue}, \ldots)
\]

calls \(\text{rankfeatures}\) with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each \text{PropertyName} must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

\[
[IDX, Z] = \text{rankfeatures}(X, \text{Group}, \ldots \text{'Criterion'}, \text{CriterionValue}, \ldots)
\]

sets the criterion used to assess the significance of every feature for separating two labeled groups. Choices are:

- \text{'ttest'} (default) — Absolute value two-sample t-test with pooled variance estimate.
- \text{'entropy'} — Relative entropy, also known as Kullback-Leibler distance or divergence.
- \text{'bhattacharyya'} — Minimum attainable classification error or Chernoff bound.
- \text{'roc'} — Area between the empirical receiver operating characteristic (ROC) curve and the random classifier slope.
- \text{'wilcoxon'} — Absolute value of the standardized u-statistic of a two-sample unpaired Wilcoxon test, also known as Mann-Whitney.

**Note** \text{'ttest'}, \text{'entropy'}, and \text{'bhattacharyya'} assume normal distributed classes while \text{'roc'} and \text{'wilcoxon'} are nonparametric tests. All tests are feature independent.
\[ \text{IDX}, \text{Z} = \text{rankfeatures}(X, \text{Group}, \ldots \text{CCWeighting'}, \text{ALPHA}, \ldots) \] uses correlation information to outweigh the \text{Z} value of potential features using \( \text{Z} \times (1 - \text{ALPHA} \times (\text{RHO})) \), where \text{RHO} is the average of the absolute values of the cross-correlation coefficient between the candidate feature and all previously selected features. \text{ALPHA} sets the weighting factor. It is a scalar value between 0 and 1. When \text{ALPHA} is 0 (default) potential features are not weighted. A large value of \text{RHO} (close to 1) outweighs the significance statistic; this means that features that are highly correlated with the features already picked are less likely to be included in the output list.

\[ \text{IDX}, \text{Z} = \text{rankfeatures}(X, \text{Group}, \ldots \text{NWeighting'}, \text{BETA}, \ldots) \] uses regional information to outweigh the \text{Z} value of potential features using \( \text{Z} \times (1 - \text{exp}(-(\text{DIST}/\text{BETA}).^2)) \), where \text{DIST} is the distance (in rows) between the candidate feature and previously selected features. \text{BETA} sets the weighting factor. It is greater than or equal to 0. When \text{BETA} is 0 (default) potential features are not weighted. A small \text{DIST} (close to 0) outweighs the significance statistics of only close features. This means that features that are close to already picked features are less likely to be included in the output list. This option is useful for extracting features from time series with temporal correlation.

\text{BETA} can also be a function of the feature location, specified using @ or an anonymous function. In both cases \text{rankfeatures} passes the row position of the feature to \text{BETA()} and expects back a value greater than or equal to 0.

\textbf{Note} You can use \text{'CCWeighting'} and \text{'NWeighting'} together.

\[ \text{IDX}, \text{Z} = \text{rankfeatures}(X, \text{Group}, \ldots \text{NumberOfIndices'}, \text{N}, \ldots) \] sets the number of output indices in \text{IDX}. Default is the same as the number of features when \text{ALPHA} and \text{BETA} are 0, or 20 otherwise.

\[ \text{IDX}, \text{Z} = \text{rankfeatures}(X, \text{Group}, \ldots \text{CrossNorm'}, \text{CN}, \ldots) \] applies independent normalization across the observations for every feature. Cross-normalization ensures comparability among different features, although it is not always necessary because the selected criterion might already account for this. Choices are:

- \text{'none'} (default) — Intensities are not cross-normalized.
- \text{'meanvar'} — \( x_{\text{new}} = (x - \text{mean}(x))/\text{std}(x) \)
- \text{'softmax'} — \( x_{\text{new}} = (1 + \text{exp}((\text{mean}(x) - x)/\text{std}(x)))^{-1} \)
- \text{'minmax'} — \( x_{\text{new}} = (x - \text{min}(x))/(\text{max}(x) - \text{min}(x)) \)

\textbf{Examples}

**Find a reduced set of genes to differentiate breast cancer cells**

Find a reduced set of genes that is sufficient for differentiating breast cancer cells from all other types of cancer in the t-matrix NCI60 data set. Load sample data.

load NCI60tmatrix

Get a logical index vector to the breast cancer cells.

BC = GROUP == 8;

Select features.
I = rankfeatures(X,BC,'NumberOfIndices',12);

Test features with a linear discriminant classifier.

C = classify(X(I,:),X(I,:),double(BC));
cp = classperf(BC,C);
cp.CorrectRate

ans =

1

Use cross-correlation weighting to further reduce the required number of genes.

I = rankfeatures(X,BC,'CCWeighting',0.7,'NumberOfIndices',8);
C = classify(X(I,:),X(I,:),double(BC));
cp = classperf(BC,C);
cp.CorrectRate

ans =

1

Find the discriminant peaks of two groups of signals with Gaussian pulses modulated by two different sources.

load GaussianPulses
f = rankfeatures(y,grp,'NWeighting',@(x) x/10+5,'NumberOfIndices',5);
plot(t,y(grp==1,:),'b',t,y(grp==2,:),'g',t(f),1.35,'vr')

References


See Also
classify | classperf | crossvalind | randfeatures | sequentialfs

Introduced before R2006a
rdivide (DataMatrix)

Right array divide DataMatrix objects

Syntax

\[ \text{DMObjNew} = \text{rdivide} (\text{DMObj1}, \text{DMObj2}) \]
\[ \text{DMObjNew} = \text{DMObj1} ./ \text{DMObj2} \]
\[ \text{DMObjNew} = \text{rdivide} (\text{DMObj1}, B) \]
\[ \text{DMObjNew} = \text{DMObj1} ./ B \]
\[ \text{DMObjNew} = \text{rdivide} (B, \text{DMObj1}) \]
\[ \text{DMObjNew} = B ./ \text{DMObj1} \]

Input Arguments

<table>
<thead>
<tr>
<th>DMObj1, DMObj2</th>
<th>DataMatrix objects, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>MATLAB numeric or logical array.</td>
</tr>
</tbody>
</table>

Output Arguments

| DMObjNew       | DataMatrix object created by right array division.                     |

Description

\[ \text{DMObjNew} = \text{rdivide} (\text{DMObj1}, \text{DMObj2}) \] or the equivalent \[ \text{DMObjNew} = \text{DMObj1} ./ \text{DMObj2} \] performs an element-by-element right array division of the DataMatrix objects \text{DMObj1} and \text{DMObj2} and places the results in \text{DMObjNew}, another DataMatrix object. In other words, rdivide divides each element in \text{DMObj1} by the corresponding element in \text{DMObj2}. \text{DMObj1} and \text{DMObj2} must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for \text{DMObjNew} are the same as \text{DMObj1}, unless \text{DMObj1} is a scalar; then they are the same as \text{DMObj2}.

\[ \text{DMObjNew} = \text{rdivide} (\text{DMObj1}, B) \] or the equivalent \[ \text{DMObjNew} = \text{DMObj1} ./ B \] performs an element-by-element right array division of the DataMatrix object \text{DMObj1} and \text{B}, a numeric or logical array, and places the results in \text{DMObjNew}, another DataMatrix object. In other words, rdivide divides each element in \text{DMObj1} by the corresponding element in \text{B}. \text{DMObj1} and \text{B} must have the same size (number of rows and columns), unless \text{B} is a scalar. The size (number of rows and columns), row names, and column names for \text{DMObjNew} are the same as \text{DMObj1}.

\[ \text{DMObjNew} = \text{rdivide} (B, \text{DMObj1}) \] or the equivalent \[ \text{DMObjNew} = B ./ \text{DMObj1} \] performs an element-by-element right array division of \text{B}, a numeric or logical array, and the DataMatrix object \text{DMObj1}, and places the results in \text{DMObjNew}, another DataMatrix object. In other words, rdivide divides each element in \text{B} by the corresponding element in \text{DMObj1}. \text{DMObj1} and \text{B} must have the same size (number of rows and columns), unless \text{B} is a scalar. The size (number of rows and columns), row names, and column names for \text{DMObjNew} are the same as \text{DMObj1}.

**Note** Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.
MATLAB calls $DM_{Obj\text{New}} = \text{rdivide}(X, Y)$ for the syntax $DM_{Obj\text{New}} = X ./ Y$ when $X$ or $Y$ is a DataMatrix object.

See Also
DataMatrix | ldivide | times

Topics
DataMatrix object on page 1-532

Introduced in R2008b
read

Class: BioIndexedFile

Read one or more entries from source file associated with BioIndexedFile object

Syntax

\[
\text{Output} = \text{read}(\text{BioIFobj}, \text{Indices}) \\
\text{Output} = \text{read}(\text{BioIFobj}, \text{Key})
\]

Description

\(\text{Output} = \text{read}(\text{BioIFobj}, \text{Indices})\) reads the entries specified by \(\text{Indices}\) from the source file associated with \(\text{BioIFobj}\), a BioIndexedFile object. \(\text{Indices}\) is a vector of positive integers specifying indices to entries in the source file. The read method reads and parses the entries using the function specified by the Interpreter property of the BioIndexedFile object. A one-to-one relationship exists between the number and order of elements in \(\text{Indices}\) and \(\text{Output}\), even if \(\text{Indices}\) has repeated entries. \(\text{Output}\) is a structure or an array of structures containing the parsed data returned by the interpreter function.

\(\text{Output} = \text{read}(\text{BioIFobj}, \text{Key})\) reads the entries specified by \(\text{Key}\) from the source file associated with \(\text{BioIFobj}\), a BioIndexedFile object. \(\text{Key}\) is a character vector or cell array of character vectors specifying one or more keys to entries in the source file. The read method reads and parses the entries using the function specified by the Interpreter property of the BioIndexedFile object. If the keys in the source file are not unique, the read method reads all entries that match a specified key, all at the position of the key in the \(\text{Key}\) cell array. If the keys in the source file are unique, there is a one-to-one relationship between the number and order of elements in \(\text{Key}\) and \(\text{Output}\).

Input Arguments

BioIFobj

Object of the BioIndexedFile class.

Default:

Indices

Vector of positive integers specifying indices to entries in the source file associated with \(\text{BioIFobj}\), the BioIndexedFile object. The number of elements in \(\text{Indices}\) must be less than or equal to the number of entries in the source file. There is a one-to-one relationship between the number and order of elements in \(\text{Indices}\) and \(\text{Output}\), even if \(\text{Indices}\) has repeated entries.

Default:

Key

Character vector or cell array of character vectors specifying one or more keys in the source file.
Default:

**Output Arguments**

**Output**

Structure or an array of structures containing the parsed data returned by the interpreter function.

**Examples**

Construct a BioIndexedFile object to access a table containing cross-references between gene names and gene ontology (GO) terms:

```matlab
% Create variable containing full absolute path of source file
sourcefile = which('yeastgenes.sgd');
% Create a BioIndexedFile object from the source file. Indicate
% the source file is a tab-delimited file where contiguous rows
% with the same key are considered a single entry. Store the
% index file in the Current Folder. Indicate that keys are
% located in column 3 and that header lines are prefaced with!
gene2goObj = BioIndexedFile('mrtab', sourcefile, '.', ...
    'KeyColumn', 3, 'HeaderPrefix','!')

% Access entries that have the string YAT2 in their keys
YAT2_entries = getEntryByKey(gene2goObj, 'YAT2');
% Adjust the object interpreter to return only the column
% containing the GO term
gene2goObj.Interpreter = @(x) regexp(x,'GO:\d+','match')
% Parse the entries with a key of YAT2 and return all GO terms
% from those entries
GO_YAT2_entries = read(gene2goObj, 'YAT2')

GO_YAT2_entries =
    'GO:0004092'  'GO:0005737'  'GO:0006066'  'GO:0006066'  'GO:0009437'
```

**Tips**

Before using the `read` method, make sure the `Interpreter` property of the BioIndexedFile object is set appropriately. The `Interpreter` property is a handle to a function that parses the entries in the source file. The interpreter function must accept a character vector of one or more concatenated entries and return a structure or an array of structures containing the interpreted data.

If the BioIndexedFile object was created from a source file with an application-specific format such as `SAM`, `FASTQ`, or `FASTA`, the default `Interpreter` property is a handle to a function appropriate for that file type and typically does not require you to change it. If the BioIndexedFile object was created from a source file with a `TABLE`, `MRTAB`, or `FLAT` format, then the default `Interpreter` property is `[]`, which means the interpreter is an anonymous function in which the output is equivalent to the input.

For information on setting the `Interpreter` property, see `BioIndexedFile` class.
See Also
BioIndexedFile | getSubset

Topics
“Work with Next-Generation Sequencing Data”
rebasecuts

Find restriction enzymes that cut nucleotide sequence

Syntax

[Enzymes, Sites] = rebasecuts(SeqNT)
rebasecuts(SeqNT, Group)
rebasecuts(SeqNT, [Q, R])
rebasecuts(SeqNT, S)

Input Arguments

<table>
<thead>
<tr>
<th>SeqNT</th>
<th>Nucleotide sequence.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>Cell array of character vectors or string vector representing the names of valid restriction enzymes.</td>
</tr>
<tr>
<td>Q, R</td>
<td>Base positions that limit the search to all sites between base Q and base R.</td>
</tr>
<tr>
<td>S</td>
<td>Base position that limits the search to all sites after base S.</td>
</tr>
</tbody>
</table>

Output Arguments

<table>
<thead>
<tr>
<th>Enzymes</th>
<th>Cell array of character vectors containing the names of restriction enzymes from REBASE®, the Restriction Enzyme Database.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sites</td>
<td>Vector of cut sites identified with the base position number before every cut.</td>
</tr>
</tbody>
</table>

Description

[Enzymes, Sites] = rebasecuts(SeqNT) finds all the restriction enzymes that cut SeqNT, a nucleotide sequence.

rebasecuts(SeqNT, Group) limits the search to Group, a list of enzymes.

rebasecuts(SeqNT, [Q, R]) limits the search to those enzymes that cut after the base position specified by Q and before the base position specified by R.

rebasecuts(SeqNT, S) limits the search to those enzymes that cut just after the base position specified by S.

REBASE, the Restriction Enzyme Database, is a collection of information about restriction enzymes and related proteins. For more information about REBASE, see:

http://rebase.neb.com/rebase/rebase.html

Examples

1. Create a nucleotide sequence.

   ```matlab
   seq = 'AGAGGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA';
   ```
2 Find all possible enzymes and cleavage sites in the sequence.

    [enzymes, sites] = rebasecuts(seq)

3 Find where restriction enzymes CfoI and Tru9I cut the sequence.

    [enzymes, sites] = rebasecuts(seq, {'CfoI','Tru9I'})

    enzymes =
    'CfoI'
    'CfoI'
    'Tru9I'

    sites =
    13
    39
    45

4 Find all possible enzymes that cut after base 7.

    enzymes  = rebasecuts(seq, 7)

    enzymes =
    'Csp6I'
    'CviQI'
    'RsaNI'

5 Find all possible enzymes that cut between bases 11 and 37.

    enzymes  = rebasecuts(seq, [11 37])

    enzymes =
    'AccII'
    'AspLEI'
    'BmiI'
    'Bsh1236I'
    'BspFNI'
    'BspLII'
    'BstFNI'
    'BstHHI'
    'BstUI'
    'CfoI'
    'FnuDII'
    'GlaI'
    'HhaI'
    'Hin6I'
    'HinP1I'
    'Hpvl88I'
    'HspAI'
    'MvnI'
    'NlaIV'
    'PspN4I'
    'SetI'
References


See Also
cleave|cleavelookup|regexp|restrict|seq2regexp|seqshowwords

Introduced before R2006a
redbluecmap

Create red and blue colormap

Syntax

redbluecmap(Length)

Arguments

| Length | Positive integer that specifies the length of (or the number of colors in) the colormap. Choices are positive integers ≥ 3 or ≤ 11. Default is 11. |

Description

redbluecmap(Length) returns a Length-by-3 matrix containing a red and blue diverging color palette. Low values are dark blue, values in the center of the map are white, and high values are dark red. Length is a positive integer ≥ 3 and ≤ 11, which determines the number of colors in the colormap. Default is 11.

Examples

Perform hierarchical clustering on gene expression data

Load microarray data containing gene expression levels of Saccharomyces cerevisiae (yeast) during the metabolic shift from fermentation to respiration (Derisi, J. et al., 1997).

load filteredyeastdata

This MAT-file includes three variables, which are added to the MATLAB® workspace:

• yeastvalues - A matrix of gene expression data from Saccharomyces cerevisiae during the metabolic shift from fermentation to respiration
• genes - A cell array of GenBank® accession numbers for labeling the rows in yeastvalues
• times - A vector of time values for labeling the columns in yeastvalues

Create a clustergram object and display the heat map from the gene expression data in the first 30 rows of the yeastvalues matrix and standardize along the rows of data.

cgo = clustergram(yeastvalues(1:30,:), 'Standardize','Row')

Clustergram object with 30 rows of nodes and 7 columns of nodes.
Use the `set` method and the `genes` and `times` vectors to add meaningful row and column labels to the clustergram.

```matlab
set(cgo,'RowLabels',genes(1:30),'ColumnLabels',times)
```
Add a color bar to the clustergram by clicking the Insert Colorbar button on the toolbar.

View a data tip containing the intensity value, row label, and column label for a specific area of the heat map by clicking the Data Cursor button on the toolbar, then clicking an area in the heat map. To delete this data tip, right-click it, then select Delete Current Datatip.

Display intensity values for each area of the heat map by clicking the Annotate button on the toolbar. Click the Annotate button again to remove the intensity values.

Tip: If the amount of data is large enough, the cells within the clustergram are too small to display the intensity annotations. Zoom in to see the intensity annotations.

Remove the dendrogram tree diagrams from the figure by clicking the Show Dendrogram button on the toolbar. Click it again to display the dendrograms.

Use the get method to display the properties of the clustergram object, cgo.

get(cgo)

    Cluster: 'ALL'
    RowPDist: {'Euclidean'}
Change the clustering parameters by changing the linkage method and changing the color of the groups of nodes in the dendrogram whose linkage is less than a threshold of 3.

```matlab
set(cgo,'Linkage','complete','Dendrogram',3)
```
Place the cursor on a branch node in the dendrogram to highlight (in blue) the group associated with it. Press and hold the mouse button to display a data tip listing the group number and the nodes (genes or samples) in the group.
Right-click a branch node in the dendrogram to display a menu of options.
The following options are available:

- **Set Group Color** - Change the cluster group color.
- **Print Group to Figure** - Print the group to a Figure window.
- **Copy Group to New Clustergram** - Copy the group to a new Clustergram window.
- **Export Group to Workspace** - Create a clustergram object of the group in the MATLAB Workspace.
- **Export Group Info to Workspace** - Create a structure containing information about the group in the MATLAB Workspace. The structure contains these fields:

1. **GroupNames** - Cell array of character vectors containing the names of the row or column groups.
2. **RowNodeNames** - Cell array of character vectors containing the names of the row nodes.
3. **ColumnNodeNames** - Cell array of text character vectors containing the names of the column nodes.
4. **ExprValues** - An M-by-N matrix of intensity values, where M and N are the number of row nodes and of column nodes respectively. If the matrix contains gene expression data, typically each row corresponds to a gene and each column corresponds to sample.
Create a clustergram object in the MATLAB Workspace of Group 18 by right-clicking it, then selecting Export Group to Workspace. In the **Export to Workspace** dialog box, type Group18, then click OK.

Use the **view** method to view the clustergram object, **Group18**.

```matlab
view(Group18)
```

![Clustergram Object](image)

View all the gene expression data using a diverging red and blue colormap and standardize along the rows of data.

```matlab
cgo_all = clustergram(yeastvalues,'Colormap','redbluecmap','Standardize','Row')
```

Clustergram object with 614 rows of nodes and 7 columns of nodes.
Create structure arrays to specify marker colors and annotations for two groups of rows (510 and 593) and two groups of columns (4 and 5).

```matlab
rm = struct('GroupNumber',{510,593},'Annotation',{'A','B'},... 'Color',{'b','m'});

cm = struct('GroupNumber',{4,5},'Annotation',{'Time1','Time2'},... 'Color',{{1 1 0},{0.6 0.6 1}});
```

Use the ‘RowGroupMarker’ and ‘ColumnGroupMarker’ properties to add the color markers and annotations to the clustergram.

```matlab
set(cgo_all,'RowGroupMarker',rm,'ColumnGroupMarker',cm)
```
References


See Also
Colormap Editor | clustergram | colormap | redgreenmap

Introduced in R2008a
**redgreencmap**

Create red and green colormap

**Syntax**

redgreencmap(Length)

redgreencmap(Length, 'Interpolation', InterpolationValue)

**Arguments**

<table>
<thead>
<tr>
<th><strong>Length</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Length of the colormap. Enter either 256 or 64. Default is the length of the colormap of the current figure.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>InterpolationValue</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InterpolationValue</td>
<td>Property that lets you set the algorithm for color interpolation. Choices are:</td>
</tr>
<tr>
<td></td>
<td>- 'linear'</td>
</tr>
<tr>
<td></td>
<td>- 'quadratic'</td>
</tr>
<tr>
<td></td>
<td>- 'cubic'</td>
</tr>
<tr>
<td></td>
<td>- 'sigmoid' (default)</td>
</tr>
</tbody>
</table>

**Note** The sigmoid interpolation is tanh.

**Description**

redgreencmap(Length) returns a Length-by-3 matrix containing a red and green colormap. Low values are bright green, values in the center of the map are black, and high values are red. Enter either 256 or 64 for Length. If Length is empty, the length of the map will be the same as the length of the colormap of the current figure.

redgreencmap(Length, 'Interpolation', InterpolationValue) lets you set the algorithm for color interpolation. Choices are:

- 'linear'
- 'quadratic'
- 'cubic'
- 'sigmoid' (default)

**Note** The sigmoid interpolation is tanh.

**Examples**

1. Create a MATLAB structure from the microarray data in a GenePix Results (GPR) file, then display an image of the 'F635 Median' field.
pd = gprread('mouse_alpd.gpr');
maimage(pd,'F635 Median')

2 Reset the colormap of the current figure.

colormap(redgreencmap)
See Also
Colormap Editor|clustergram|colormap|redbluecmap

Introduced before R2006a
reorder (phytree)

Reorder leaves of phylogenetic tree

Syntax

Tree1Reordered = reorder(Tree1, Order)
[Tree1Reordered, OptimalOrder] = reorder(Tree1, Order, 'Approximate', ApproximateValue)
[Tree1Reordered, OptimalOrder] = reorder(Tree1, Tree2)

Input Arguments

<table>
<thead>
<tr>
<th>Tree1, Tree2</th>
<th>Phytree objects.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td>Vector with position indices for each leaf.</td>
</tr>
<tr>
<td>ApproximateValue</td>
<td>Controls the use of the optimal leaf-ordering calculation to find the closest order possible to the suggested one without dividing the clades or producing crossing branches. Enter true to use the calculation. Default is false.</td>
</tr>
</tbody>
</table>

Output Arguments

<table>
<thead>
<tr>
<th>Tree1Reordered</th>
<th>Phytree object with reordered leaves.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OptimalOrder</td>
<td>Vector of position indices for each leaf in Tree1Reordered, determined by the optimal leaf-ordering calculation.</td>
</tr>
</tbody>
</table>

Description

Tree1Reordered = reorder(Tree1, Order) reorders the leaves of the phylogenetic tree Tree1, without modifying its structure and distances, creating a new phylogenetic tree, Tree1Reordered. Order is a vector of position indices for each leaf. If Order is invalid, that is, if it divides the clades (or produces crossing branches), then reorder returns an error message.

[Tree1Reordered, OptimalOrder] = reorder(Tree1, Order, 'Approximate', ApproximateValue) controls the use of the optimal leaf-ordering calculation, which finds the best approximate order closest to the suggested one, without dividing the clades or producing crossing branches. Enter true to use the calculation and return Tree1Reordered, the reordered tree, and OptimalOrder, a vector of position indices for each leaf in Tree1Reordered, determined by the optimal leaf-ordering calculation. Default is false.

[Tree1Reordered, OptimalOrder] = reorder(Tree1, Tree2) uses the optimal leaf-ordering calculation to reorder the leaves in Tree1 such that it matches the order of leaves in Tree2 as closely as possible, without dividing the clades or producing crossing branches. Tree1Reordered is the reordered tree, and OptimalOrder is a vector of position indices for each leaf in Tree1Reordered, determined by the optimal leaf-ordering calculation.
Examples

Example 1.73. Reordering Leaves Using a Valid Order

1  Create and view a phylogenetic tree.
   
   ```matlab
   b = [1 2; 3 4; 5 6; 7 8; 9 10];
   tree = phytree(b)
   view(tree)
   ``

2  Reorder the leaves on the phylogenetic tree, and then view the reordered tree.
   
   ```matlab
   treeReordered = reorder(tree, [5, 6, 3, 4, 1, 2])
   view(treeReordered)
   ```

Example 1.74. Finding Best Approximate Order When Using an Invalid Order

1  Create a phylogenetic tree by reading a Newick-formatted tree file (ASCII text file).
   
   ```matlab
   tree = phytreeread('pf00002.tree')
   view(tree)
   ``

2  Create a row vector of the leaf names in alphabetical order.
   
   ```matlab
   [dummy,order] = sort(get(tree,'LeafNames'));
   ``

3  Reorder the phylogenetic tree to match as closely as possible the row vector of alphabetically ordered leaf names, without dividing the clades or having crossing branches.
   
   ```matlab
   treeReordered = reorder(tree,order,'approximate',true)
   view(treeReordered)
   ``

Example 1.75. Reordering Leaves to Match Leaf Order in Another Phylogenetic Tree

1  Create a phylogenetic tree by reading sequence data from a FASTA file, calculating the pairwise distances between sequences, and then using the neighbor-joining method.
   
   ```matlab
   seqs = fastaread('pf00002.fa')
   
   seqs =
   33x1 struct array with fields:
       Header
       Sequence
   
   dist = seqpdist(seqs,'method','jukes-cantor','indels','pair');
   NJtree = seqneighjoin(dist,'equivar',seqs)
   view(NJtree)
   ``

2  Create another phylogenetic tree from the same sequence data and pairwise distances between sequences, using the single linkage method.
   
   ```matlab
   HCTree = seqlinkage(dist,'single',seqs)
   view(HCTree)
   ```
3 Use the optimal leaf-ordering calculation to reorder the leaves in HCTree such that it matches the order of leaves in NJtree as closely as possible, without dividing the clades or having crossing branches.

    HCTree_reordered = reorder(HCTree,NJtree)
    Phylogenetic tree object with 33 leaves (32 branches)

4 View the reordered phylogenetic tree and the tree used to reorder it.

    view(HCTree_reordered)
    view(NJtree)

See Also
get | getbyname | phytree | prune

Topics
phytree object on page 1-1274

Introduced in R2007a
reroot (phytree)

Change root of phylogenetic tree

Syntax

\[ \text{Tree2} = \text{reroot(} \text{Tree1} \text{)} \]
\[ \text{Tree2} = \text{reroot(} \text{Tree1, Node} \text{)} \]
\[ \text{Tree2} = \text{reroot(} \text{Tree1, Node, Distance} \text{)} \]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree1</td>
<td>Phylogenetic tree (phytree object) created with the function phytree.</td>
</tr>
<tr>
<td>Node</td>
<td>Node index returned by the phytree object method getbyname.</td>
</tr>
<tr>
<td>Distance</td>
<td>Distance from the reference branch.</td>
</tr>
</tbody>
</table>

Description

\[ \text{Tree2} = \text{reroot(} \text{Tree1} \text{)} \] changes the root of a phylogenetic tree (\text{Tree1}) using a midpoint method. The midpoint is the location where the mean values of the branch lengths, on either side of the tree, are equalized. The original root is deleted from the tree.

\[ \text{Tree2} = \text{reroot(} \text{Tree1, Node} \text{)} \] changes the root of a phylogenetic tree (\text{Tree1}) to a branch node using the node index (\text{Node}). The new root is placed at half the distance between the branch node and its parent.

\[ \text{Tree2} = \text{reroot(} \text{Tree1, Node, Distance} \text{)} \] changes the root of a phylogenetic tree (\text{Tree1}) to a new root at a given distance (\text{Distance}) from the reference branch node (\text{Node}) toward the original root of the tree. Note: The new branch representing the root in the new tree (\text{Tree2}) is labeled 'Root'.

Examples

1. Create an ultrametric tree.

\[
\begin{align*}
\text{tr}_1 &= \text{phytree}([5 \ 7; 8 \ 9; 6 \ 11; 1 \ 2; 3 \ 4; 10 \ 12; \ldots \\
& \quad 14 \ 16; 15 \ 17; 13 \ 18]) \\
\text{plot}(\text{tr}_1, 'branchlabels', \text{true})
\end{align*}
\]

A figure with the phylogenetic tree displays.
2 Place the root at 'Branch 7'.

```matlab
sel = getbyname(tr_1,'Branch 7');
tr_2 = reroot(tr_1,sel)
plot(tr_2,'branchlabels',true)
```

A figure of a phylogenetic tree displays with the root moved to the center of branch 7.

3 Move the root to a branch that makes the tree as ultrametric as possible.

```matlab
tr_3 = reroot(tr_2)
plot(tr_3,'branchlabels',true)
```

A figure of the new tree displays with the root moved from the center of branch 7 to branch 8.
See Also
get | getbyname | phytree | prune | select | seqneighjoin

Topics
phytree object on page 1-1274

Introduced before R2006a
restrict

Split nucleotide sequence at restriction site

Syntax

\[
\text{Fragments} = \text{restrict} (\text{SeqNT}, \text{Enzyme})
\]
\[
\text{Fragments} = \text{restrict} (\text{SeqNT}, \text{NTPattern}, \text{Position})
\]
\[
[\text{Fragments}, \text{CuttingSites}] = \text{restrict} (...) 
\]
\[
[\text{Fragments}, \text{CuttingSites}, \text{Lengths}] = \text{restrict} (...)
\]
\[
... = \text{restrict} (...,'\text{PartialDigest}', \text{PartialDigestValue})
\]

Arguments

<table>
<thead>
<tr>
<th>SeqNT</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers.</td>
</tr>
<tr>
<td></td>
<td>• Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that contains a nucleotide sequence, such as returned by fastaread, fastqread, emblread, getembl, genbankread, or getgenbank.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enzyme</th>
<th>Character vector or string specifying a name of a restriction enzyme from REBASE, the Restriction Enzyme Database.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tip</strong></td>
<td>Some enzymes specify cutting rules for both a strand and its complement strand. restrict applies the cutting rule only for the 5' —&gt; 3' strand. For a workaround to applying an enzyme cutting rule for both strands, see “Examples” on page 1-1405.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NTPattern</th>
<th>Short nucleotide sequence recognition pattern to search for in SeqNT, a larger sequence. NTPattern can be either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string</td>
</tr>
<tr>
<td></td>
<td>• Regular expression (MATLAB)</td>
</tr>
</tbody>
</table>
### Description

**Fragments** = \texttt{restrict(SeqNT, Enzyme)} cuts \texttt{SeqNT}, a nucleotide sequence, into fragments at the restriction sites of \texttt{Enzyme}, a restriction enzyme. The \texttt{restrict} function stores the return values in \texttt{Fragments}, a cell array of sequences.

\texttt{Fragments} = \texttt{restrict(SeqNT, NTPattern, Position)} cuts \texttt{SeqNT}, a nucleotide sequence, into fragments at restriction sites specified by \texttt{NTPattern}, a nucleotide recognition pattern, and \texttt{Position}.

\texttt{[Fragments, CuttingSites]} = \texttt{restrict(...)} returns a numeric vector with the indices representing the cutting sites. The \texttt{restrict} function adds a 0 to the beginning of the \texttt{CuttingSites} vector so that the number of elements in \texttt{CuttingSites} equals the number of elements in \texttt{Fragments}. You can use \texttt{CuttingSites} + 1 to point to the first base of every fragment respective to the original sequence.

\texttt{[Fragments, CuttingSites, Lengths]} = \texttt{restrict(...)} returns a numeric vector with the lengths of every fragment.

\texttt{... = restrict(..., 'PartialDigest', PartialDigestValue)} simulates a partial digest where each restriction site in the sequence has a \texttt{PartialDigestValue} or probability of being cut.

REBASE, the Restriction Enzyme Database, is a collection of information about restriction enzymes and related proteins. For more information about REBASE or to search REBASE for the name of a restriction enzyme, see:

http://rebase.neb.com/rebase/rebase.html

### Examples

**Example 1.76. Splitting a Nucleotide Sequence by Specifying an Enzyme**

1. Enter a nucleotide sequence.
   \begin{verbatim}
   Seq = 'AGAGGGTACGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA';
   \end{verbatim}
2. Use the restriction enzyme HspAI (which specifies a recognition sequence of GCGC and a cleavage position of 1) to cleave the nucleotide sequence.
   \begin{verbatim}
   fragmentsEnzyme = restrict(Seq,'HspAI')
   \end{verbatim}
MATLAB returns:

\[
\text{fragmentsEnzyme} =
\begin{align*}
'AGAGGGGTACG' \\
'CGCTCTGAAAAGCGGGAACCTCGTGG' \\
'CGCTTTATTAA'
\end{align*}
\]

**Example 1.77. Splitting a Nucleotide Sequence by Specifying a Pattern and Position**

1. Enter a nucleotide sequence.
   
   ```matlab
   Seq = 'AGAGGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA';
   ```

2. Use the sequence pattern GCGC with the point of cleavage at position 3 to cleave the nucleotide sequence.
   
   ```matlab
   fragmentsPattern = restrict(Seq,'GCGC',3)
   ```

   MATLAB returns:
   
   ```matlab
   fragmentsPattern =
   'AGAGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA'
   ```

**Example 1.78. Splitting a Nucleotide Sequence by Specifying a Regular Expression for the Pattern**

1. Enter a nucleotide sequence.
   
   ```matlab
   Seq = 'AGAGGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA';
   ```

2. Use a regular expression to specify the sequence pattern.
   
   ```matlab
   fragmentsRegExp = restrict(Seq,'GCG[^C]',3)
   ```

   MATLAB returns:
   
   ```matlab
   fragmentsRegExp =
   'AGAGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA'
   ```

**Example 1.79. Returning the Cutting Sites and Fragment Lengths**

1. Enter a nucleotide sequence.
   
   ```matlab
   Seq = 'AGAGGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA';
   ```

2. Capture the cutting sites and fragment lengths as well as the fragments.
   
   ```matlab
   [fragments, cut_sites, lengths] = restrict(Seq,'HspAI')
   ```

   MATLAB returns:
   
   ```matlab
   fragments =
   'AGAGGGTACG' \\
   'CGCTCTGAAAAGCGGGAACCTCGTGG' \\
   'CGCTTTATTAA'
   ```
**Example 1.80. Splitting a Double-Stranded Nucleotide Sequence**

Some enzymes specify cutting rules for both a strand and its complement strand. `restrict` applies the cutting rule only for the 5' → 3' strand. You can apply this rule manually for the complement strand.

1. Enter a nucleotide sequence.
   ```matlab
   seq = 'CCCGCNNNNNNN';
   ```

2. Use the `seqcomplement` function to determine the complement strand, which is in the 3' → 5' direction.
   ```matlab
   seqc = seqcomplement(seq)
   MATLAB returns:
   seqc =
   GGGCGNNNNNNNN
   ```

3. Cut the first strand using the restriction enzyme **FauI** (which specifies a recognition sequence pattern of CCCGC and a cleavage position of 9).
   ```matlab
   cuts_strand1 = restrict(seq, 'FauI')
   MATLAB returns:
   cuts_strand1 =
   'CCCGCNNNN'  
   'NNN'
   ```

4. Cut the complement strand according to the rule specified by **FauI** (which specifies a recognition sequence pattern of GGGCG with the point of cleavage at position 11).
   ```matlab
   cuts_strand2 = restrict(seqc, 'GGGCG', 11)
   MATLAB returns:
   cuts_strand2 =
   'GGGCGNNNNNNN'
   'N'
   ```
References


See Also
cleave|cleavelookup|rebasecuts|regexp|seq2regexp|seqcomplement|seqshowwords

Introduced before R2006a
revgeneticcode

Return reverse mapping (amino acid to nucleotide codon) for genetic code

Syntax

\[
\text{Map} = \text{revgeneticcode} \\
\text{Map} = \text{revgeneticcode}(\text{GeneticCode}) \\
\text{Map} = \text{revgeneticcode}(..., \text{'Alphabet'}, \text{AlphabetValue}, ...) \\
\text{Map} = \text{revgeneticcode}(..., \text{'ThreeLetterCodes'}, \text{ThreeLetterCodesValue}, ...)
\]

Input Arguments

<table>
<thead>
<tr>
<th>GeneticCode</th>
<th>Integer, character vector, or string specifying a genetic code number or code name from the table Genetic Code. Default is 1 or 'Standard'.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Tip</strong> If you use a code name, you can truncate the name to the first two letters of the name.</td>
</tr>
<tr>
<td>AlphabetValue</td>
<td>Character vector or string specifying the nucleotide alphabet to use in the map. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'DNA' (default) — Uses the symbols A, C, G, and T.</td>
</tr>
<tr>
<td></td>
<td>• 'RNA' — Uses the symbols A, C, G, and U.</td>
</tr>
<tr>
<td>ThreeLetterCodesValue</td>
<td>Controls the use of three-letter amino acid codes as field names in the return structure Map. Choices are true for three-letter codes or false for one-letter codes. Default is false.</td>
</tr>
</tbody>
</table>

Output Arguments

| Map | Structure containing the reverse mapping of amino acids to nucleotide codons for the standard genetic code. The Map structure contains a field for each amino acid. |

Description

\[
\text{Map} = \text{revgeneticcode} \text{ returns a structure containing the reverse mapping of amino acids to nucleotide codons for the standard genetic code. The Map structure contains a field for each amino acid.}
\]

\[
\text{Map} = \text{revgeneticcode}(\text{GeneticCode}) \text{ returns a structure containing the reverse mapping of amino acids to nucleotide codons for the specified genetic code. GeneticCode is either:}
\]

- An integer, character vector, or string specifying a code number or code name from the table Genetic Code
• The `transl_table` (code) number from the NCBI Web page describing genetic codes:

**Tip** If you use a code name, you can truncate the name to the first two letters of the name.

\[ Map = \text{revgeneticcode}(\ldots, '\text{PropertyName}', \text{PropertyValue}, \ldots) \]

\[ \text{Map} = \text{revgeneticcode}(\ldots, \text{'Alphabet'}, \text{AlphabetValue}, \ldots) \]

\[ \text{Map} = \text{revgeneticcode}(\ldots, \text{'ThreeLetterCodes'}, \text{ThreeLetterCodesValue}, \ldots) \]

*Genetic Code*

<table>
<thead>
<tr>
<th>Code Number</th>
<th>Code Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Standard</td>
</tr>
<tr>
<td>2</td>
<td>Vertebrate Mitochondrial</td>
</tr>
<tr>
<td>3</td>
<td>Yeast Mitochondrial</td>
</tr>
<tr>
<td>4</td>
<td>Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma/</td>
</tr>
<tr>
<td></td>
<td>Spiroplasma</td>
</tr>
<tr>
<td>5</td>
<td>Invertebrate Mitochondrial</td>
</tr>
<tr>
<td>6</td>
<td>Ciliate, Dasycladacean, and Hexamita Nuclear</td>
</tr>
<tr>
<td>9</td>
<td>Echinoderm Mitochondrial</td>
</tr>
<tr>
<td>10</td>
<td>Euplotid Nuclear</td>
</tr>
<tr>
<td>11</td>
<td>Bacterial and Plant Plastid</td>
</tr>
<tr>
<td>12</td>
<td>Alternative Yeast Nuclear</td>
</tr>
<tr>
<td>13</td>
<td>Ascidian Mitochondrial</td>
</tr>
<tr>
<td>14</td>
<td>Flatworm Mitochondrial</td>
</tr>
<tr>
<td>15</td>
<td>Blepharisma Nuclear</td>
</tr>
<tr>
<td>16</td>
<td>Chlorophycean Mitochondrial</td>
</tr>
<tr>
<td>21</td>
<td>Trematode Mitochondrial</td>
</tr>
<tr>
<td>22</td>
<td>Scenedesmus Obliquus Mitochondrial</td>
</tr>
<tr>
<td>23</td>
<td>Thraustochytrium Mitochondrial</td>
</tr>
</tbody>
</table>
Examples

- Return the reverse mapping of amino acids to nucleotide codons for the Standard genetic code.

  ```matlab
  map = revgeneticcode
  map =
  Name: 'Standard'
  A: {'GCT' 'GCC' 'GCA' 'GCG'}
  R: {'CGT' 'CGC' 'CGA' 'CGG' 'AGA' 'AGG'}
  N: {'AAT' 'AAC'}
  D: {'GAT' 'GAC'}
  C: {'TGT' 'TGC'}
  Q: {'CAA' 'CAG'}
  E: {'GAA' 'GAG'}
  G: {'GTT' 'GTC' 'GTA' 'GTG'}
  H: {'CAT' 'CAC'}
  I: {'ATT' 'ATC' 'ATA'}
  L: {'TTA' 'TTG' 'CTT' 'CTC' 'CTA' 'CTG'}
  K: {'AAA' 'AAG'}
  M: {'ATG'}
  F: {'TTT' 'TTC'}
  P: {'CCU' 'CCC' 'CCA' 'CCG'}
  S: {'UCU' 'UCC' 'UCA' 'UCG'}
  T: {'ACU' 'ACC' 'ACA' 'ACG'}
  W: {'TGG'}
  Y: {'TAT' 'TAC'}
  V: {'GTV' 'GTC' 'GTA' 'GTG'}
  Stops: {'TAA' 'TAG' 'TGA'}
  Starts: {'TTG' 'CTG' 'ATG'}
  ```

- Return the reverse mapping of amino acids to nucleotide codons for the Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma/Spiroplasma genetic code, using the rna alphabet.

  ```matlab
  moldmap = revgeneticcode(4,'Alphabet','rna');
  ```

- Return the reverse mapping of amino acids to nucleotide codons for the Flatworm Mitochondrial genetic code, using three-letter codes for the field names in the return structure.

  ```matlab
  wormmap = revgeneticcode('Flatworm Mitochondrial',... 'ThreeLetterCodes',true);
  ```

References

[1] NCBI Web page describing genetic codes:


See Also

aa2nt | aminolookup | baselookup | geneticcode | nt2aa

Introduced before R2006a
**rmabackadj**

Perform background adjustment on Affymetrix microarray probe-level data using Robust Multi-array Average (RMA) procedure

**Syntax**

BackAdjustedMatrix = rmabackadj(PMData)

BackAdjustedMatrix = rmabackadj(..., 'Method', MethodValue, ...)
BackAdjustedMatrix = rmabackadj(..., 'Truncate', TruncateValue, ...)
BackAdjustedMatrix = rmabackadj(..., 'Showplot', ShowplotValue, ...)

**Input Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMData</td>
<td>Matrix of intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)</td>
</tr>
<tr>
<td>MethodValue</td>
<td>Specifies the estimation method for the background adjustment model parameters. Enter either 'RMA' (to use estimation method described by Bolstad, 2005) or 'MLE' (to estimate the parameters using maximum likelihood). Default is 'RMA'.</td>
</tr>
<tr>
<td>TruncateValue</td>
<td>Specifies the background noise model. Enter either true (use a truncated Gaussian distribution) or false (use a nontruncated Gaussian distribution). Default is true.</td>
</tr>
<tr>
<td>ShowplotValue</td>
<td>Controls the plotting of a histogram showing the distribution of PM probe intensity values (blue) and the convoluted probability distribution function (red), with estimated parameters mu, sigma and alpha. Enter either 'all' (plot a histogram for each column or chip) or specify a subset of columns (chips) by entering the column number, list of numbers, or range of numbers.</td>
</tr>
</tbody>
</table>

**Output Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackAdjustedMatrix</td>
<td>Matrix of background-adjusted probe intensity values.</td>
</tr>
</tbody>
</table>

**Description**

BackAdjustedMatrix = rmabackadj(PMData) returns the background adjusted values of probe intensity values in the matrix, PMData. Note that each row in PMData corresponds to a perfect match (PM) probe and each column in PMData corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.) Details on the background adjustment are described by Bolstad, 2005.

BackAdjustedMatrix = rmabackadj(..., 'PropertyName', PropertyValue, ...) calls rmabackadj with optional properties that use property name/property value pairs. You can specify
one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

BackAdjustedMatrix = rmabackadj(..., 'Method', MethodValue, ...) specifies the estimation method for the background adjustment model parameters. When MethodValue is 'RMA', rmabackadj implements the estimation method described by Bolstad, 2005. When MethodValue is 'MLE', rmabackadj estimates the parameters using maximum likelihood. Default is 'RMA'.

BackAdjustedMatrix = rmabackadj(..., 'Truncate', TruncateValue, ...) specifies the background noise model used. When TruncateValue is false, rmabackadj uses nontruncated Gaussian as the background noise model. Default is true.

BackAdjustedMatrix = rmabackadj(..., 'Showplot', ShowplotValue, ...) lets you plot a histogram showing the distribution of PM probe intensity values (blue) and the convoluted probability distribution function (red), with estimated parameters mu, sigma and alpha. When ShowplotValue is 'all', rmabackadj plots a histogram for each column or chip. When ShowplotValue is a number, list of numbers, or range of numbers, rmabackadj plots a histogram for the indicated column number (chip).

For example:

- (...,'Showplot', 3,...) plots the intensity values in column 3 of PMData.
- (...,'Showplot', [3,5,7],...) plots the intensity values in columns 3, 5, and 7 of PMData.
- (...,'Showplot', 3:9,...) plots the intensity values in columns 3 to 9 of PMData.
Examples

1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains Affymetrix probe-level data, including `pmMatrix`, a matrix of PM probe intensity values from multiple CEL files.

```matlab
load prostatecancerrawdata
```

2 Perform background adjustment on the PM probe intensity values in the matrix, `pmMatrix`, creating a new matrix, `BackgroundAdjustedMatrix`.

```matlab
BackgroundAdjustedMatrix = rmabackadj(pmMatrix);
```

3 Perform background adjustment on the PM probe intensity values in only column 3 of the matrix, `pmMatrix`, creating a new matrix, `BackgroundAdjustedChip3`.

```matlab
BackgroundAdjustedChip3 = rmabackadj(pmMatrix(:,3));
```

The `prostatecancerrawdata.mat` file used in the previous example contains data from Best et al., 2005.

References


See Also

`affyinvarsetnorm` | `affyread` | `affyrma` | `celintensityread` | `probelibraryinfo` | `probesetlink` | `probesetlookup` | `probesetvalues` | `quantilenorm` | `rmasummary`

Introduced in R2006a
rmasummary

Calculate gene expression values from Affymetrix microarray probe-level data using Robust Multi-array Average (RMA) procedure

Syntax

ExpressionMatrix = rmasummary(ProbeIndices, Data)

ExpressionMatrix = rmasummary(ProbeIndices, Data, 'Output', OutputValue)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProbeIndices</td>
<td>Column vector of probe indices. The convention for probe indices is, for each probe set, to label each probe 0 to ( N - 1 ), where ( N ) is the number of probes in the probe set.</td>
</tr>
<tr>
<td>Data</td>
<td>Matrix of natural-scale intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)</td>
</tr>
<tr>
<td>OutputValue</td>
<td>Specifies the scale of the returned gene expression values. OutputValue can be:</td>
</tr>
<tr>
<td></td>
<td>• 'log'</td>
</tr>
<tr>
<td></td>
<td>• 'log2'</td>
</tr>
<tr>
<td></td>
<td>• 'log10'</td>
</tr>
<tr>
<td></td>
<td>• 'linear'</td>
</tr>
<tr>
<td></td>
<td>• @functionname</td>
</tr>
</tbody>
</table>

In the last instance, the data is transformed as defined by the function functionname. Default is 'log2'.

Tip Use the ProbeIndices field in the structure returned by celintensityread as the ProbeIndices input.

Tip Using a single-precision matrix for Data decreases memory usage.

Tip You can use the matrix from the PMIntensities field in the structure returned by celintensityread as the Data input. However, first ensure the matrix has been background adjusted, using the rmabackadj or grmabackadj function, and normalized, using the quantilenorm function.
Description

ExpressionMatrix = rmasummary(ProbeIndices, Data) returns gene (probe set) expression values after calculating them from natural-scale probe intensities in the matrix Data, using the column vector of probe indices, ProbeIndices. Note that each row in Data corresponds to a perfect match (PM) probe, and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.) Note that the column vector ProbeIndices designates probes within each probe set by labeling each probe 0 to N - 1, where N is the number of probes in the probe set. Note that each row in ExpressionMatrix corresponds to a gene (probe set) and each column in ExpressionMatrix corresponds to an Affymetrix CEL file, which represents a single chip.

For a given probe set n, with J probe pairs, let $Y_{ijn}$ denote the background-adjusted, base 2 log transformed and quantile-normalized PM probe intensity value of chip $i$ and probe $j$. $Y_{ijn}$ follows a linear additive model:

$$Y_{ijn} = U_{in} + A_{jn} + E_{ijn}; \quad i = 1, \ldots, I; \quad j = 1, \ldots, J; \quad n = 1, \ldots, N$$

where:

$U_{in}$ = Gene expression of the probe set $n$ on chip $i$

$A_{jn}$ = Probe affinity effect for the $j$th probe in the probe set

$E_{ijn}$ = Residual for the $j$th probe on the $i$th chip

The RMA method assumes $A_1 + A_2 + \ldots + A_J = 0$ for all probe sets. A robust procedure, median polish, estimates $U_i$ as the log scale measure of expression.

**Note** There is no column in ExpressionMatrix that contains probe set or gene information.

ExpressionMatrix = rmasummary(..., 'PropertyName', PropertyValue, ...) calls rmasummary with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

ExpressionMatrix = rmasummary(ProbeIndices, Data, 'Output', OutputValue)

specifies the scale of the returned gene expression values. OutputValue can be:

- 'log'
- 'log2'
- 'log10'
- 'linear'
- @functionname

In the last instance, the data is transformed as defined by the function functionname. Default is 'log2'.
Examples

1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains Affymetrix data variables, including pmMatrix, a matrix of PM probe intensity values from multiple CEL files.

load prostatecancerrawdata

2 Perform background adjustment on the PM probe intensity values in the matrix, pmMatrix, using the rmabackadj function, thereby creating a new matrix, BackgroundAdjustedMatrix.

BackgroundAdjustedMatrix = rmabackadj(pmMatrix);

3 Normalize the data in BackgroundAdjustedMatrix, using the quantilenorm function.

NormMatrix = quantilenorm(BackgroundAdjustedMatrix);

4 Calculate gene expression values from the probe intensities in NormMatrix, creating a new matrix, ExpressionMatrix. (Use the probeIndices column vector provided to supply information on the probe indices.)

ExpressionMatrix = rmasummary(probeIndices, NormMatrix);

The prostatecancerrawdata.mat file used in the previous example contains data from Best et al., 2005.

References


See Also

affygcrrma|affyinvarsetnorm|affyrma|celintensityread|gcrmabackadj|
mainvarsetnorm|malowess|manorm|quantilenorm|rmabackadj

Introduced in R2006a
rna2dna
Convert RNA sequence to DNA sequence

Syntax

\[ \text{SeqDNA} = \text{rna2dna(\text{SeqRNA})} \]

Arguments

<table>
<thead>
<tr>
<th>SeqRNA</th>
<th>RNA sequence specified by any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string with the characters A, C, G, U, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N,</td>
</tr>
<tr>
<td></td>
<td>• Row vector of integers from the table Mapping Nucleotide Integers to Letter Codes.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that contains an RNA sequence, such as returned by fastaread, fastqread, emblread, getembl, genbankread, or getgenbank.</td>
</tr>
</tbody>
</table>

Description

\[ \text{SeqDNA} = \text{rna2dna(\text{SeqRNA})} \] converts an RNA sequence to a DNA sequence by converting any uracil nucleotides (U) in the RNA sequence to thymine nucleotides (T). The DNA sequence is returned in the same format as the RNA sequence. For example, if \( \text{SeqRNA} \) is a vector of integers, then so is \( \text{SeqDNA} \).

Examples

Convert an RNA sequence to a DNA sequence.

\[
\begin{align*}
\text{rna2dna('ACGAUGACAUAGCUU')} \\
\text{ans = ACGATGAGTCATGCTT}
\end{align*}
\]

See Also
dna2rna | regexp | strrep

Introduced before R2006a
**rnaconvert**

Convert secondary structure of RNA sequence between bracket and matrix notations

**Syntax**

\[
RNA\text{Struct}2 = \text{rnaconvert}(RNA\text{Struct})
\]

**Input Arguments**

<table>
<thead>
<tr>
<th><strong>RNA\text{Struct}</strong></th>
<th>Secondary structure of an RNA sequence represented by either:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Bracket notation</td>
</tr>
<tr>
<td></td>
<td>• Connectivity matrix</td>
</tr>
</tbody>
</table>

**Tip** Use the `rnafold` function to create `RNA\text{Struct}`.

**Output Arguments**

<table>
<thead>
<tr>
<th><strong>RNA\text{Struct}2</strong></th>
<th>Secondary structure of an RNA sequence represented by either:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• <strong>Bracket notation</strong> — Character vector or string containing dots and brackets, where each dot represents an unpaired base, while a pair of equally nested, opening and closing brackets represents a base pair.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Connectivity matrix</strong> — Binary, upper-triangular matrix, where ( RNA\text{matrix}(i, j) = 1 ) if and only if the ( i )th residue in the RNA sequence ( Seq ) is paired with the ( j )th residue of ( Seq ).</td>
</tr>
</tbody>
</table>

**Description**

\( RNA\text{Struct}2 = \text{rnaconvert}(RNA\text{Struct}) \) returns `RNA\text{Struct}2`, the secondary structure of an RNA sequence, in matrix notation (if `RNA\text{Struct}` is in bracket notation), or in bracket notation (if `RNA\text{Struct}` is in matrix notation).

**Examples**

**Example 1.81. Converting from Bracket to Matrix Notation**

1. Create a character vector representing a secondary structure of an RNA sequence in bracket notation.
   
   \[
   \text{Bracket} = '(((((............)))))(........));
   \]

2. Convert the secondary structure to a connectivity matrix representation.

   \[
   \text{Matrix} = \text{rnaconvert}('\text{Bracket}');
   \]
Example 1.82. Converting from Matrix to Bracket Notation

1. Create a connectivity matrix representing a secondary structure of an RNA sequence.

```matlab
Matrix2 = zeros(12);
Matrix2(1,12) = 1;
Matrix2(2,11) = 1;
Matrix2(3,10) = 1;
Matrix2(4,9) = 1;
```

2. Convert the secondary structure to bracket notation.

```matlab
Bracket2 = rnaconvert(Matrix2)

Bracket2 =

((((....))))
```

See Also
rnafold | rnaplot

Introduced in R2007b
rnafold

Predict minimum free-energy secondary structure of RNA sequence

Syntax

rnafold(Seq)
RNAbracket = rnafold(Seq)
[RNAbracket, Energy] = rnafold(Seq)
[RNAbracket, Energy, RNAmatrix] = rnafold(Seq)

... = rnafold(Seq, ...'MinLoopSize', MinLoopSizeValue, ...)
... = rnafold(Seq, ...'NoGU', NoGUValue, ...)
... = rnafold(Seq, ...'Progress', ProgressValue, ...)

Input Arguments

<table>
<thead>
<tr>
<th>Seq</th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying an RNA sequence.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that specifies an RNA sequence.</td>
</tr>
<tr>
<td>MinLoopSizeValue</td>
<td>Integer specifying the minimum size of the loops (in bases) to be considered when computing the free energy. Default is 3.</td>
</tr>
<tr>
<td>NoGUValue</td>
<td>Controls whether GU or UG pairs are forbidden to form. Choices are true or false (default).</td>
</tr>
<tr>
<td>ProgressValue</td>
<td>Controls the display of a progress bar during the computation of the minimum free-energy secondary structure. Choices are true or false (default).</td>
</tr>
</tbody>
</table>

Output Arguments

<table>
<thead>
<tr>
<th>RNAbracket</th>
<th>Character vector of dots and brackets indicating the bracket notation for the minimum-free energy secondary structure of an RNA sequence. In the bracket notation, each dot represents an unpaired base, while a pair of equally nested, opening and closing brackets represents a base pair.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy</td>
<td>Value specifying the energy (in kcal/mol) of the minimum free-energy secondary structure of an RNA sequence.</td>
</tr>
<tr>
<td>RNAmatrix</td>
<td>Connectivity matrix representing the minimum free-energy secondary structure of an RNA sequence. A binary, upper-triangular matrix where ( RNAmatrix(i, j) = 1 ) if and only if the ith residue in the RNA sequence Seq is paired with the jth residue of Seq.</td>
</tr>
</tbody>
</table>
Description

`rnafold(Seq)` predicts and displays the secondary structure (in bracket notation) associated with the minimum free energy for the RNA sequence, `Seq`, using the thermodynamic nearest-neighbor approach.

**Note** For long sequences, this prediction can be time consuming. For example, a 600-nucleotide sequence can take several minutes, and sequences greater than 1000 nucleotides can take over 1 hour, depending on your system.

`RNAbracket = rnafold(Seq)` predicts and returns the secondary structure associated with the minimum free energy for the RNA sequence, `Seq`, using the thermodynamic nearest-neighbor approach. The returned structure, `RNAbracket`, is in bracket notation, that is a vector of dots and brackets, where each dot represents an unpaired base, while a pair of equally nested, opening and closing brackets represents a base pair.

`[RNAbracket, Energy] = rnafold(Seq)` also returns `Energy`, the energy value (in kcal/mol) of the minimum free-energy secondary structure of the RNA sequence.

`[RNAbracket, Energy, RNAmatrix] = rnafold(Seq)` also returns `RNAmatrix`, a connectivity matrix representing the secondary structure associated with the minimum free energy. `RNAmatrix` is an upper triangular matrix where `RNAmatrix(i, j) = 1` if and only if the `i`th residue in the RNA sequence `Seq` is paired with the `j`th residue of `Seq`.

`... = rnafold(Seq, ...'PropertyName', PropertyValue, ...)` calls `rnafold` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`... = rnafold(Seq, ...'MinLoopSize', MinLoopSizeValue, ...)` specifies the minimum size of the loops (in bases) to be considered when computing the free energy. Default is 3.

`... = rnafold(Seq, ...'NoGU', NoGUValue, ...)` controls whether GU or UG pairs are forbidden to form. Choices are true or false (default).

`... = rnafold(Seq, ...'Progress', ProgressValue, ...)` controls the display of a progress bar during the computation of the minimum free-energy secondary structure. Choices are true or false (default).

Examples

Determine the minimum free-energy secondary structure (in both bracket and matrix notation) and the energy value of the following RNA sequence:

```matlab
seq = 'ACCCCCUCCUCCUUGGAUCAAGGGGCUCAA';
[bracket, energy, matrix] = rnafold(seq);bracket
```

```matlab
bracket =
..(((((.....)))))).....
```
References


See Also

rnaconvert | rnaplot

Introduced in R2007b
**rnaplot**

Draw secondary structure of RNA sequence

**Syntax**

```matlab
rnaplot(RNA2ndStruct)
ha = rnaplot(RNA2ndStruct)
[ha, H] = rnaplot(RNA2ndStruct)
```

```matlab
rnaplot(RNA2ndStruct, ...'Sequence', SequenceValue, ...)
rnaplot(RNA2ndStruct, ...'Format', FormatValue, ...)
rnaplot(RNA2ndStruct, ...'Selection', SelectionValue, ...)
rnaplot(RNA2ndStruct, ...'ColorBy', ColorByValue, ...)
```

**Input Arguments**

<table>
<thead>
<tr>
<th><strong>RNA2ndStruct</strong></th>
<th>Secondary structure of an RNA sequence represented by either:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string specifying bracket notation</td>
</tr>
<tr>
<td></td>
<td>• Connectivity matrix</td>
</tr>
</tbody>
</table>

**Tip** Use the `rnafold` function to create `RNA2ndStruct`.

<table>
<thead>
<tr>
<th><strong>SequenceValue</strong></th>
<th>Sequence of the RNA secondary structure being plotted, specified by either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string</td>
</tr>
<tr>
<td></td>
<td>• Structure containing a <code>Sequence</code> field that contains an RNA sequence</td>
</tr>
</tbody>
</table>

This information is used in the data tip displayed by clicking a base in the plot of the RNA secondary structure `RNA2ndStruct`. This information is required if you specify the 'Diagram' format or if you specify to highlight any of the following paired selections: 'AU', 'UA', 'GC', 'CG', 'GU' or 'UG'.

<table>
<thead>
<tr>
<th><strong>FormatValue</strong></th>
<th>Character vector or string specifying the format of the plot. Choices are:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• 'Circle' (default)</td>
</tr>
<tr>
<td></td>
<td>• 'Diagram'</td>
</tr>
<tr>
<td></td>
<td>• 'Dotdiagram'</td>
</tr>
<tr>
<td></td>
<td>• 'Graph'</td>
</tr>
<tr>
<td></td>
<td>• 'Mountain'</td>
</tr>
<tr>
<td></td>
<td>• 'Tree'</td>
</tr>
</tbody>
</table>

**Note** If you specify 'Diagram', you must also use the 'Sequence' property to provide the RNA sequence.
| **SelectionValue** | Either of the following:
| | • Numeric array specifying the indices of residues to highlight in the plot.
| | • Character vector or string specifying the subset of residues to highlight in the plot. Choices are:
| | • 'Paired'
| | • 'Unpaired'
| | • 'AU' or 'UA'
| | • 'GC' or 'CG'
| | • 'GU' or 'UG'

**Note** If you specify 'AU', 'UA', 'GC', 'CG', 'GU', or 'UG', you must also use the 'Sequence' property to provide the RNA sequence.

| **ColorByValue** | Character vector or string specifying a color scheme for the plot. Choices are:
| | • 'State' (default) — Color by pair state: paired bases and unpaired bases.
| | • 'Residue' — Color by residue type (A, C, G, and U).
| | • 'Pair' — Color by pair type (AU/UA, GC/CG, and GU/UG).

**Note** If you specify 'residue' or 'pair', you must also use the 'Sequence' property to provide the RNA sequence.

**Note** Because internal nodes of a tree correspond to paired residues, you cannot specify 'residue' if you specify 'Tree' for the 'Format' property.

### Output Arguments

| **ha** | Handle to the figure axis. |
**Description**

`rnaplot(RNA2ndStruct)` draws the RNA secondary structure specified by `RNA2ndStruct`, the secondary structure of an RNA sequence represented by a character vector or string specifying bracket notation or a connectivity matrix.

`ha = rnaplot(RNA2ndStruct)` returns `ha`, a handle to the figure axis.

`[ha, H] = rnaplot(RNA2ndStruct)` also returns `H`, a structure of handles, which you can use to graph elements in a MATLAB Figure window.

**Tip** Use the handles returned in `H` to change properties of the graph elements, such as color, marker size, and marker type.

`H` contains a subset of the following fields, based on what you specify for the 'Selection' and 'ColorBy' properties.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paired</td>
<td>Handles to all paired residues</td>
</tr>
<tr>
<td>Unpaired</td>
<td>Handles to all unpaired residues</td>
</tr>
<tr>
<td>A</td>
<td>Handles to all A residues</td>
</tr>
<tr>
<td>C</td>
<td>Handles to all C residues</td>
</tr>
<tr>
<td>G</td>
<td>Handles to all G residues</td>
</tr>
<tr>
<td>U</td>
<td>Handles to all U residues</td>
</tr>
<tr>
<td>AU</td>
<td>Handles to all AU or UA pairs</td>
</tr>
<tr>
<td>GC</td>
<td>Handles to all GC or CG pairs</td>
</tr>
<tr>
<td>GU</td>
<td>Handles to all GU or UG pairs</td>
</tr>
<tr>
<td>Selected</td>
<td>Handles to all selected residues</td>
</tr>
</tbody>
</table>
rnaplot(RNA2ndStruct, ...'PropertyName', PropertyValue, ...) calls rnaplot with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

rnaplot(RNA2ndStruct, ...'Sequence', SequenceValue, ...) draws the RNA secondary structure specified by RNA2ndStruct, and annotates it with the sequence positions supplied by SequenceValue, the RNA sequence specified by a character vector, string, or a structure containing a Sequence field.

rnaplot(RNA2ndStruct, ...'Format', FormatValue, ...) draws the RNA secondary structure specified by RNA2ndStruct, using the format specified by FormatValue.

FormatValue is a character vector or string specifying the format of the plot. Choices are as follows.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Circle'</td>
<td>Each base is represented by a dot on the circumference of a circle of arbitrary size. Lines connect bases that pair with each other.</td>
</tr>
<tr>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>'Diagram'</td>
<td>Two-dimensional representation of the RNA secondary structure. Each base is represented and identified by a letter. The backbone and hydrogen bonds between base pairs are represented by lines.</td>
</tr>
<tr>
<td></td>
<td><img src="image1" alt="Diagram Image" /></td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> If you specify 'Diagram', you must also use the 'Sequence' property to provide the RNA sequence.</td>
</tr>
<tr>
<td>'Dotdiagram'</td>
<td>Two-dimensional representation of the RNA secondary structure. Each base is represented and identified by a dot. The backbone and hydrogen bonds between base pairs are represented by lines.</td>
</tr>
<tr>
<td></td>
<td><img src="image2" alt="Dotdiagram Image" /></td>
</tr>
<tr>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>'Graph'</td>
<td>Bases are displayed in their sequence position along the abscissa (x-axis) of a graph. Semi-elliptical lines connect bases that pair with each other. The height of the lines is proportional to the distance between paired bases.</td>
</tr>
<tr>
<td></td>
<td><img src="image1" alt="Graph Format Diagram" /></td>
</tr>
<tr>
<td>'Mountain'</td>
<td>Each base is represented by a dot in a two-dimensional plot, where the base position is in the abscissa (x-axis) and the number of base pairs enclosing a given base is in the ordinate (y-axis).</td>
</tr>
<tr>
<td></td>
<td><img src="image2" alt="Mountain Format Diagram" /></td>
</tr>
<tr>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>'Tree'</td>
<td>Each base is represented by a node in a tree graph. Leaf nodes indicate unpaired bases, while each internal node indicates a base pair. The tree root is a fictitious node, not associated with any base in the secondary structure.</td>
</tr>
</tbody>
</table>

*rnaplot*(RNA2ndStruct, ...'Selection', SelectionValue, ...) draws the RNA secondary structure specified by RNA2ndStruct, highlighting a subset of residues specified by SelectionValue. SelectionValue can be either:

- Numeric array specifying the indices of residues to highlight in the plot.
- Character vector or string specifying the subset of residues to highlight in the plot. Choices are:
  - 'Paired'
  - 'Unpaired'
  - 'AU' or 'UA'
  - 'GC' or 'CG'
  - 'GU' or 'UG'

**Note**  If you specify 'AU', 'UA', 'GC', 'CG', 'GU', or 'UG', you must also use the 'Sequence' property to provide the RNA sequence.

*rnaplot*(RNA2ndStruct, ...'ColorBy', ColorByValue, ...) draws the RNA secondary structure specified by RNA2ndStruct, using a color scheme specified by ColorByValue, a character vector or string indicating a color scheme. Choices are:

- 'State' (default) — Color by pair state: paired bases and unpaired bases.
- 'Pair' — Color by pair type (AU/UA, GC/CG, and GU/UG).

**Note**  If you specify 'Residue' or 'Pair', you must also use the 'Sequence' property to provide the RNA sequence.

**Note**  Because internal nodes of a tree correspond to paired residues, you cannot specify 'Residue' if you specify 'Tree' for the 'Format' property.
Examples

1 Determine the minimum free-energy secondary structure of an RNA sequence and plot it in circle format:

```matlab
seq = 'GCCCGCCGCUAGCUACUAUGAGAGCUACGGAUACAAAGGUGUGGUCACUGUCUGCCCGCGCG';
ss = rnafold(seq);
rnaplot(ss)
```

![Figure 1](image)

2 Plot the RNA sequence secondary structure in graph format and color it by pair type.

```matlab
rnaplot(ss, 'sequence', seq, 'format', 'graph', 'colorby', 'pair')
```
3 Plot the RNA sequence secondary structure in mountain format and color it by residue type. Use the handle to add a title to the plot.

```matlab
ha = rnaplot(ss, 'sequence', seq, 'format', 'mountain', ...
              'colorby', 'residue');
title(ha, 'Bacillus halodurans, tRNA Arg');
```
Mutate the first six positions in the sequence and observe the effect the change has on the secondary structure by highlighting the first six residues.

\[
\text{seqMut} = \text{seq};
\text{seqMut}(1:6) = 'AAAAAA';
\text{ssMut} = \text{rnafold(seqMut)};
\text{rnaplot(ss, 'sequence', seq, 'format', 'dotdiagram', 'selection', 1:6)};
\text{rnaplot(ssMut, 'sequence', seqMut, 'format', 'dotdiagram', 'selection', 1:6)};
\]
Tip If necessary, click-drag the legend to prevent it from covering the plot. Click a base in the plot to display a data tip with information on that base.

See Also
rnaconvert | rnafold

Introduced in R2007b
rownames (DataMatrix)

Retrieve or set row names of DataMatrix object

Syntax

\[
\text{ReturnRowNames} = \text{rownames}(\text{DMObj})
\]

\[
\text{ReturnRowNames} = \text{rownames}(\text{DMObj}, \text{RowIndices})
\]

\[
\text{DMObjNew} = \text{rownames}(\text{DMObj}, \text{RowIndices}, \text{RowNames})
\]

Input Arguments

| \( \text{DMObj} \) | DataMatrix object, such as created by \texttt{DataMatrix} (object constructor). |
| \( \text{RowIndices} \) | One or more rows in \( \text{DMObj} \), specified by any of the following: |
| | • Positive integer |
| | • Vector of positive integers |
| | • Character vector specifying a row name |
| | • Cell array of character vectors |
| | • Logical vector |
| \( \text{RowNames} \) | Row names specified by any of the following: |
| | • Numeric vector |
| | • Cell array of character vectors |
| | • Character array |
| | • Single character vector, which is used as a prefix for row names, with row numbers appended to the prefix |
| | • Logical \texttt{true} or \texttt{false} (default). If \texttt{true}, unique row names are assigned using the format \texttt{row1}, \texttt{row2}, \texttt{row3}, etc. If \texttt{false}, no row names are assigned. |

\textbf{Note} The number of elements in \( \text{RowNames} \) must equal the number of elements in \( \text{RowIndices} \).

Output Arguments

| \( \text{ReturnRowNames} \) | Character vector or cell array of character vectors containing row names in \( \text{DMObj} \). |
| \( \text{DMObjNew} \) | DataMatrix object created with names specified by \( \text{RowIndices} \) and \( \text{RowNames} \). |

Description

\( \text{ReturnRowNames} = \text{rownames}(\text{DMObj}) \) returns \( \text{ReturnRowNames} \), a cell array of character vectors specifying the row names in \( \text{DMObj} \), a DataMatrix object.
ReturnRowNames = rownames(DMObj, RowIndices) returns the row names specified by RowIndices. RowIndices can be a positive integer, vector of positive integers, character vector specifying a row name, cell array of character vectors, or a logical vector.

DMObjNew = rownames(DMObj, RowIndices, RowNames) returns DMObjNew, a DataMatrix object with rows specified by RowIndices set to the names specified by RowNames. The number of elements in RowIndices must equal the number of elements in RowNames.

See Also
DataMatrix | colnames

Topics
DataMatrix object on page 1-532

Introduced in R2008b
**run**

Map sequence reads to reference sequence using Bowtie 2

**Syntax**

```plaintext
run(object,indexBaseName,reads1,reads2,outputFileName)
run(___,'IncludeAll',TF)
flag = run(__)
```

**Description**

`run(object,indexBaseName,reads1,reads2,outputFileName)` maps the sequencing reads from `reads1` and `reads2` against the reference sequence and writes the results to the output file `outputFileName`. The input `indexBaseName` represents the base name (prefix) of the reference index files. `object` is a `Bowtie2AlignOptions` object.

`run` requires the Bioinformatics Toolbox Interface for Bowtie Aligner support package. If this support package is not installed, then the function provides a download link.

**Note** `run` is supported on Mac and UNIX platforms only.

`run(___,'IncludeAll',TF)` specifies whether to use all object properties and their corresponding values when running `bowtie2`. Specify this option after all other input arguments. By default, only the modified properties are used to run `bowtie2`.

`flag = run(__)` returns an exit flag of the function using any of the input arguments in the previous syntaxes.

**Examples**

**Align Reads to Reference Sequence Using Bowtie 2**

First build a set of index files for the Drosophila genome. An error message appears if you do not have the Bioinformatics Toolbox Interface for Bowtie Aligner support package installed when you run the function. Click the provided link to download the package from the Add-on menu.

For this example, the reference sequence `Dmel_chr4.fa` is already provided with the toolbox.

```plaintext
status = bowtie2build('Dmel_chr4.fa', 'Dmel_chr4_index');
```

If the index build is successful, the function returns 0 and creates the index files (*.bt2) in the current folder. The files have the prefix `'Dmel_chr4_index'`.

Once the index is ready, map the read sequences to the reference. The pair-end read files (SRR6008575_10k_1.fq and SRR6008575_10k_2.fq) are already provided with the toolbox.

Create an options object.
alignOpt = Bowtie2AlignOptions;
Trim four residues from the 3' end before aligning.
alignOpt.Trim3 = 4;
Map reads to the reference using the specified alignment option.
flag = run(alignOpt,'Dmel_chr4','SRR6008575_10k_1.fq','SRR6008575_10k_2.fq','SRR6008575_10k_chr4_trimmed.sam');
The output is a SAM-formatted file that contains the mapping results.

**Input Arguments**

*object* — Alignment options  
Bowtie2AlignOptions object  
Alignment options, specified as a Bowtie2AlignOptions object.
Example: 'alignOpt'

*indexBaseName* — Base name of reference index files  
character vector | string  
Base name (prefix) of the reference index files, specified as a character vector or string. The file extension for index files is either *_.bt2* or *_.bt21.*
Example: 'Dmel_chr4'
Data Types: char | string

*read1* — Names of files with first mate reads  
character vector | string  
Names of files with the first mate reads, specified as a character vector or string.
Example: 'SRR6008575_10k_1.fq'
Data Types: char | string

*read2* — Names of files with second mate reads  
character vector | string  
Names of files with the second mate reads, specified as a character vector or string.
Example: 'SRR6008575_10k_2.fq'
Data Types: char | string

*outputFileName* — Output file name  
character vector | string  
Output file name, specified as a character vector or string. This file contains the mapping results.
Example: 'SRR6008575_10k_chr4.sam'
Data Types: char | string

*TF* — Flag to use all object properties and their corresponding values  
false (default) | true
Flag to use all object properties and their corresponding values when you run the function, specified as true or false. By default, only the modified properties are used.

Example: true

**Output Arguments**

flag — Exit status

type: integer

Exit status of the function, returned as an integer. flag is 0 if the function runs without errors or warning. Otherwise, it is nonzero.

**References**


**See Also**

Bowtie2AlignOptions | Bowtie2BuildOptions | Bowtie2InspectOptions | bowtie2 | bowtie2build | bowtie2inspect

**External Websites**

Bowtie 2 manual

**Introduced in R2018a**
**run**

Build Bowtie 2 index files

**Syntax**

run(object,referenceFileNames,indexBaseName)
run(____,'IncludeAll',TF)
flag = run(____)

**Description**

run(object,referenceFileNames,indexBaseName) builds Bowtie 2 index files from the reference sequence information saved in the FASTA files specified by referenceFileNames.

run requires the Bioinformatics Toolbox Interface for Bowtie Aligner support package. If this support package is not installed, then the function provides a download link.

**Note** run is supported on Mac and UNIX platforms only.

run(____,'IncludeAll',TF) specifies whether to use all object properties and their corresponding values when running bowtie2build. Specify this option after all other input arguments. By default, only the modified properties are used to run the function.

flag = run(____) returns an exit flag of the function using any of the input arguments in the previous syntaxes.

**Examples**

**Build Bowtie 2 Index Files for Reference Sequence**

Build a set of index files for the Drosophila genome. An error message appears if you do not have the Bioinformatics Toolbox Interface for Bowtie Aligner support package installed when you run the function. Click the provided link to download the package from the Add-on menu.

Create an options object.

buildOpt = Bowtie2BuildOptions;

Build the index files using the run function. For this example, the reference sequence Dmel_chr4.fa is already provided with the toolbox.

flag = run(buildOpt,'Dmel_chr4.fa', 'Dmel_chr4_index');
If the index build is successful, the function returns 0 and creates the index files (*.bt2) in the current folder. The files have the prefix 'Dmel_chr4_index'.

**Input Arguments**

- **object** — Options to build index files
  Bowtie2BuildOptions object
  Options to build the index files, specified as a Bowtie2BuildOptions object.
  Example: 'buildOpt'

- **referenceFileNames** — Names of files with reference sequence information
  string | character vector | string array | cell array of character vectors
  Names of files with reference sequence information, specified as a string, character vector, string array, or cell array of character vectors.
  Example: 'Dmel_chr4.fa'
  Data Types: char | string | cell

- **indexBaseName** — Base name of reference index files
  character vector | string
  Base name (prefix) of the reference index files, specified as a character vector or string. The file extension for index files is either *.bt2 or *.bt21.
  Example: 'Dmel_chr4'
  Data Types: char | string

- **TF** — Flag to use all object properties and their corresponding values
  false (default) | true
  Flag to use all object properties and their corresponding values when you run the function, specified as true or false. By default, only the modified properties are used.
  Example: true

**Output Arguments**

- **flag** — Exit status
  integer
  Exit status of the function, returned as an integer. flag is 0 if the function runs without errors or warning. Otherwise, it is nonzero.

**References**

See Also
Bowtie2AlignOptions | Bowtie2BuildOptions | Bowtie2InspectOptions | bowtie2 | bowtie2build | bowtie2inspect

External Websites
Bowtie 2 manual

Introduced in R2018a
run

Inspect Bowtie 2 index files

Syntax

run(object,indexBaseName,outputFileName)
run(___,'IncludeAll',TF)
flag = run(___)

Description

run(object,indexBaseName,outputFileName) inspects Bowtie 2 index files with the prefix indexBaseName, checks the original reference sequences used to build the index, and saves the reference sequences in the output file outputFileName. The argument object is a Bowtie2InspectOptions object.

run requires the Bioinformatics Toolbox Interface for Bowtie Aligner support package. If this support package is not installed, then the function provides a download link.

Note run is supported on Mac and UNIX platforms only.

run(___,'IncludeAll',TF) specifies whether to use all object properties and their corresponding values when you run bowtie2inspect. Specify this option after all other input arguments. By default, only the modified properties are used to run bowtie2inspect.

flag = run(___) returns an exit flag of the function using any of the input arguments in the previous syntaxes.

Examples

Inspect Bowtie 2 Index and Retrieve Reference Sequence Information

Get information about the reference sequence used to build the corresponding index files. An error message appears if you do not have the Bioinformatics Toolbox Interface for Bowtie Aligner support package installed when you run the function. Click the provided link to download the package from the Add-on menu.

Create an options object.

inspectOpt = Bowtie2InspectOptions;

Inspect the index and get information about the reference sequence. By default, the output file Dmel_chr4_retrieved.fa contains the actual sequence of the reference used to build the index. If the function runs without any warning or error, it returns 0.
flag = run(inspectOpt,'Dmel_chr4', 'Dmel_chr4_retrieved.fa');

**Input Arguments**

*object* — Options to inspect index files

Bowtie2InspectOptions object

Options to inspect the index files, specified as a Bowtie2InspectOptions object.

Example: 'inspectOpt'

*indexBaseName* — Base name of reference index files

character vector | string

Base name (prefix) of the reference index files, specified as a character vector or string. The file extension for index files is either *.bt2* or *.bt21.*

Example: 'Dmel_chr4'

Data Types: char | string

*outputFileName* — Name of output file

string | character vector

Name of an output file, specified as a string or character vector. By default, the output file contains the reference sequences that are used to build the index files.

Example: 'refSeq.fa'

Data Types: char | string

*TF* — Flag to use all object properties and their corresponding values

false (default) | true

Flag to use all object properties and their corresponding values when you run the function, specified as true or false. By default, only the modified properties are used.

Example: true

**Output Arguments**

*flag* — Exit status

integer

Exit status of the function, returned as an integer. flag is 0 if the function runs without errors or warning. Otherwise, it is nonzero.

**References**


**See Also**

Bowtie2AlignOptions | Bowtie2BuildOptions | Bowtie2InspectOptions | bowtie2 | bowtie2build | bowtie2inspect
External Websites
Bowtie 2 manual

Introduced in R2018a
**saminfo**

Return information about SAM file

**Syntax**

\[ \text{InfoStruct} = \text{saminfo}(	ext{File}) \]
\[ \text{InfoStruct} = \text{saminfo}(	ext{File}, \text{Name}, \text{Value}) \]

**Description**

\[ \text{InfoStruct} = \text{saminfo}(	ext{File}) \] returns a MATLAB structure containing summary information about a SAM-formatted file.

\[ \text{InfoStruct} = \text{saminfo}(	ext{File}, \text{Name}, \text{Value}) \] returns a MATLAB structure with additional options specified by one or more Name,Value pair arguments.

**Input Arguments**

**File**

Character vector or string specifying a file name or path and file name of a SAM-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the Current Folder.

**Default:**

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

**NumOfReads**

Logical that controls the inclusion of a NumReads field in InfoStruct, the output structure.

**Note** Setting NumOfReads to true can significantly increase the time to create the output structure.

**Default:** false

**ScanDictionary**

Logical that controls the scanning of the SAM-formatted file to determine the reference names and the number of reads aligned to each reference. If true, the ScannedDictionary and ScannedDictionaryCount fields contain this information.

**Default:** false
### Output Arguments

**InfoStruct**

MATLAB structure containing summary information about a SAM-formatted file. The structure contains these fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>Name of the SAM-formatted file.</td>
</tr>
<tr>
<td>FilePath</td>
<td>Path to the file.</td>
</tr>
<tr>
<td>FileSize</td>
<td>Size of the file in bytes.</td>
</tr>
<tr>
<td>FileModDate</td>
<td>Modification date of the file.</td>
</tr>
<tr>
<td>NumReads*</td>
<td>Number of sequence reads in the file.</td>
</tr>
<tr>
<td>ScannedDictionary*</td>
<td>Cell array of character vectors specifying the names of the reference sequences in the SAM-formatted file.</td>
</tr>
<tr>
<td>ScannedDictionaryCount*</td>
<td>Cell array specifying the number of reads aligned to each reference sequence.</td>
</tr>
<tr>
<td>Header**</td>
<td>Structure containing the file format version, sort order, and group order.</td>
</tr>
<tr>
<td>SequenceDictionary**</td>
<td>Structure containing the:</td>
</tr>
<tr>
<td></td>
<td>• Sequence name</td>
</tr>
<tr>
<td></td>
<td>• Sequence length</td>
</tr>
<tr>
<td></td>
<td>• Genome assembly identifier</td>
</tr>
<tr>
<td></td>
<td>• MD5 checksum of sequence</td>
</tr>
<tr>
<td></td>
<td>• URI of sequence</td>
</tr>
<tr>
<td></td>
<td>• Species</td>
</tr>
<tr>
<td>ReadGroup**</td>
<td>Structure containing the:</td>
</tr>
<tr>
<td></td>
<td>• Read group identifier</td>
</tr>
<tr>
<td></td>
<td>• Sample</td>
</tr>
<tr>
<td></td>
<td>• Library</td>
</tr>
<tr>
<td></td>
<td>• Description</td>
</tr>
<tr>
<td></td>
<td>• Platform unit</td>
</tr>
<tr>
<td></td>
<td>• Predicted median insert size</td>
</tr>
<tr>
<td></td>
<td>• Sequencing center</td>
</tr>
<tr>
<td></td>
<td>• Date</td>
</tr>
<tr>
<td></td>
<td>• Platform</td>
</tr>
<tr>
<td>Program**</td>
<td>Structure containing the:</td>
</tr>
<tr>
<td></td>
<td>• Program name</td>
</tr>
<tr>
<td></td>
<td>• Version</td>
</tr>
<tr>
<td></td>
<td>• Command line</td>
</tr>
</tbody>
</table>
* — The NumReads field is empty if you do not set the NumOfReads name-value pair argument to true. The ScannedDictionary and ScannedDictionaryCount fields are empty if you do not set the ScanDictionary name-value pair argument to true.

** — These structures and their fields appear in the output structure only if they are in the SAM file. The information in these structures depends on the information in the SAM file.

**Examples**

Return information about the ex1.sam file included with Bioinformatics Toolbox:

```matlab
info = saminfo('ex1.sam')
```

```matlab
info =

Filename: 'ex1.sam'
FilePath: [1x89 char]
FileSize: 254270
FileModDate: '12-May-2011 14:23:25'
Header: [1x1 struct]
SequenceDictionary: [1x1 struct]
ReadGroup: [1x2 struct]
NumReads: {}
ScannedDictionary: {0x1 cell}
ScannedDictionaryCount: [0x1 uint64]
```

Return information about the ex1.sam file including the number of sequence reads:

```matlab
info = saminfo('ex1.sam','numofreads', true)
```

```matlab
info =

Filename: 'ex1.sam'
FilePath: [1x89 char]
FileSize: 254270
FileModDate: '12-May-2011 14:23:25'
Header: [1x1 struct]
SequenceDictionary: [1x1 struct]
ReadGroup: [1x2 struct]
NumReads: 1501
ScannedDictionary: {0x1 cell}
ScannedDictionaryCount: [0x1 uint64]
```

**Tips**

Use saminfo to investigate the size and content of a SAM file before using the samread function to read the file contents into a MATLAB structure.

**See Also**

BioIndexedFile | BioMap | fastainfo | fastaread | fastawrite | fastqinfo | fastqread | fastqwrite | samread | sffinfo | sffread

**Topics**

"Manage Sequence Read Data in Objects"
External Websites
Sequence Read Archive
SAM format specification

Introduced in R2010a
samplealign

Align two data sets containing sequential observations by introducing gaps

**Syntax**

\[
[I, J] = \text{samplealign}(X, Y)
\]

\[
[I, J] = \text{samplealign}(X, Y, ...'Band', \text{BandValue}, ...)
\]

\[
[I, J] = \text{samplealign}(X, Y, ...'Width', \text{WidthValue}, ...)
\]

\[
[I, J] = \text{samplealign}(X, Y, ...'Gap', \text{GapValue}, ...)
\]

\[
[I, J] = \text{samplealign}(X, Y, ...'Quantile', \text{QuantileValue}, ...)
\]

\[
[I, J] = \text{samplealign}(X, Y, ...'Distance', \text{DistanceValue}, ...)
\]

\[
[I, J] = \text{samplealign}(X, Y, ...'Weights', \text{WeightsValue}, ...)
\]

\[
[I, J] = \text{samplealign}(X, Y, ...'ShowConstraints', \text{ShowConstraintsValue}, ...)
\]

\[
[I, J] = \text{samplealign}(X, Y, ...'ShowNetwork', \text{ShowNetworkValue}, ...)
\]

\[
[I, J] = \text{samplealign}(X, Y, ...'ShowAlignment', \text{ShowAlignmentValue}, ...)
\]

**Input Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>X, Y</strong></td>
<td>Matrices of data where rows correspond to observations or samples, and columns correspond to features or dimensions. X and Y can have a different number of rows, but they must have the same number of columns. The first column is the reference dimension and must contain unique values in ascending order. The reference dimension could contain sample indices of the observations or a measurable value, such as time.</td>
</tr>
</tbody>
</table>
| **BandValue** | Either of the following:  
  - Scalar.  
  - Function specified using @(z), where z is the mid-point between a given observation in one data set and a given observation in the other data set.  
  
  *BandValue* specifies a maximum allowable distance between observations (along the reference dimension only) in the two data sets, thus limiting the number of potential matches between observations in two data sets. If \( S \) is the value in the reference dimension for a given observation (row) in one data set, then that observation is matched only with observations in the other data set whose values in the reference dimension fall within \( S \pm \text{BandValue} \). Then, only these potential matches are passed to the algorithm for further scoring. Default *BandValue* is Inf. |
### WidthValue

Either of the following:

- Two-element vector, \([U, V]\)
- Scalar that is used for both \(U\) and \(V\)

**WidthValue** limits the number of potential matches between observations in two data sets; that is, each observation in \(X\) is scored to the closest \(U\) observations in \(Y\), and each observation in \(Y\) is scored to the closest \(V\) observations in \(X\). Then, only these potential matches are passed to the algorithm for further scoring. Closeness is measured using only the first column (reference dimension) in each data set. Default is \(\text{Inf}\) if `Band` is specified; otherwise default is 10.

### GapValue

Any of the following:

- Cell array, \([G, H]\), where \(G\) is either a scalar or a function handle specified using @\((X)\), and \(H\) is either a scalar or a function handle specified using @\((Y)\). The functions @\((X)\) and @\((Y)\) must calculate the penalty for each observation (row) when it is matched to a gap in the other data set. The functions @\((X)\) and @\((Y)\) must return a column vector with the same number of rows as \(X\) or \(Y\), containing the gap penalty for each observation (row).
- Single function handle specified using @\((Z)\), which is used for both \(G\) and \(H\). The function @\((Z)\) must calculate the penalty for each observation (row) in both \(X\) and \(Y\) when it is matched to a gap in the other data set. The function @\((Z)\) must take as arguments \(X\) and \(Y\). The function @\((Z)\) must return a column vector with the same number of rows as \(X\) or \(Y\), containing the gap penalty for each observation (row).
- Scalar that is used for both \(G\) and \(H\).

**GapValue** specifies the position-dependent terms for assigning gap penalties. The calculated value, \(GP_X\), is the gap penalty for matching observations from the first data set \(X\) to gaps inserted in the second data set \(Y\), and is the product of two terms: \(GP_X = G * QMS\). The term \(G\) takes its value as a function of the observations in \(X\). Similarly, \(GP_Y\) is the gap penalty for matching observations from \(Y\) to gaps inserted in \(X\), and is the product of two terms: \(GP_Y = H * QMS\). The term \(H\) takes its value as a function of the observations in \(Y\). By default, the term \(QMS\) is the 0.75 quantile of the score for the pairs of observations that are potential matches (that is, pairs that comply with the 'Band' and 'Width' constraints). Default **GapValue** is 1.

### QuantileValue

Scalar between 0 and 1 that specifies the quantile value used to calculate the term \(QMS\), which is used by the 'Gap' property to calculate gap penalties. Default is 0.75.
**DistanceValue**  
Function handle specified using @(R,S). The function @(R,S) must:

- Calculate the distance between pairs of observations that are potential matches.
- Take as arguments, R and S, matrices that have the same number of rows and columns, and whose paired rows represent all potential matches of observations in X and Y respectively.
- Return a column vector of positive values with the same number of elements as rows in R and S.

Default is the Euclidean distance between the pairs.

**Caution** All columns in X and Y, including the reference dimension, are considered when calculating distances. If you do not want to include the reference dimension in the distance calculations, use the 'Weight' property to exclude it.

**WeightsValue**  
Either of the following:

- Logical row vector with the same number of elements as columns in X and Y, that specifies columns in X and Y.
- Numeric row vector with the same number of elements as columns in X and Y, that specifies the relative weights of the columns (features).

This property controls the inclusion/exclusion of columns (features) or the emphasis of columns (features) when calculating the distance score between observations that are potential matches, that is, when using the 'Distance' property. Default is a logical row vector with all elements set to true.

**Tip** Using a numeric row vector for WeightsValue and setting some values to 0 can simplify the distance calculation when the data sets have many columns (features).

**Note** The weight values are not considered when using the 'Band', 'Width', or 'Gap' property.

**ShowConstraintsValue**  
Controls the display of the search space constrained by the specified 'Band' and 'Width' input parameters, thereby giving an indication of the memory required to run the algorithm with the specific 'Band' and 'Width' parameters on your data sets. Choices are true or false (default).

**ShowNetworkValue**  
Controls the display of the dynamic programming network, the match scores, the gap penalties, and the winning path. Choices are true or false (default).
ShowAlignmentValue | Controls the display of the first and second columns of the \(X\) and \(Y\) data sets in the abscissa and the ordinate respectively, of a two-dimensional plot. Choices are true, false (default), or an integer specifying a column of the \(X\) and \(Y\) data sets to plot as the ordinate.

**Output Arguments**

| \(I\) | Column vector containing indices of rows (observations) in \(X\) that match to a row (observation) in \(Y\). Missing indices indicate that row (observation) is matched to a gap. |
| \(J\) | Column vector containing indices of rows (observations) in \(Y\) that match to a row (observation) in \(X\). Missing indices indicate that row (observation) is matched to a gap. |

**Description**

\([I, J] = \text{samplealign}(X, Y)\) aligns the observations in two matrices of data, \(X\) and \(Y\), by introducing gaps. \(X\) and \(Y\) are matrices of data where rows correspond to observations or samples, and columns correspond to features or dimensions. \(X\) and \(Y\) can have different number of rows, but must have the same number of columns. The first column is the reference dimension and must contain unique values in ascending order. The reference dimension could contain sample indices of the observations or a measurable value, such as time. The \text{samplealign} function uses a dynamic programming algorithm to minimize the sum of positive scores resulting from pairs of observations that are potential matches and the penalties resulting from the insertion of gaps. Return values \(I\) and \(J\) are column vectors containing indices that indicate the matches for each row (observation) in \(X\) and \(Y\) respectively.

**Tip** If you do not specify return values, \text{samplealign} does not run the dynamic programming algorithm. Running \text{samplealign} without return values, but setting the 'ShowConstraints', 'ShowNetwork', or 'ShowAlignment' property to true, lets you explore the constrained search space, the dynamic programming network, or the aligned observations, without running into potential memory problems.

\([I, J] = \text{samplealign}(X, Y, ...'PropertyName', PropertyValue, ...)\) calls \text{samplealign} with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each \(PropertyName\) must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

\([I, J] = \text{samplealign}(X, Y, ...'Band', BandValue, ...)\) specifies a maximum allowable distance between observations (along the reference dimension only) in the two data sets, thus limiting the number of potential matches between observations in the two data sets. If \(S\) is the value in the reference dimension for a given observation (row) in one data set, then that observation is matched only with observations in the other data set whose values in the reference dimension fall within \(S \pm BandValue\). Then, only these potential matches are passed to the algorithm for further scoring. \(BandValue\) can be a scalar or a function specified using @(\(z\)), where \(z\) is the mid-point between a given observation in one data set and a given observation in the other data set. Default \(BandValue\) is Inf.
This constraint reduces the time and memory complexity of the algorithm from $O(MN)$ to $O(\sqrt{MN})^K$, where $M$ and $N$ are the number of observations in $X$ and $Y$ respectively, and $K$ is a small constant such that $K<<M$ and $K<<N$. Adjust BandValue to the maximum expected shift between the reference dimensions in the two data sets, that is, between $X(:,1)$ and $Y(:,1)$.

$$[I, J] = \text{samplealign}(X, Y, \ldots \text{'Width'}, \text{WidthValue}, \ldots)$$

limits the number of potential matches between observations in two data sets; that is, each observation in $X$ is scored to the closest $U$ observations in $Y$, and each observation in $Y$ is scored to the closest $V$ observations in $X$. Then, only these potential matches are passed to the algorithm for further scoring. WidthValue is either a two-element vector, $[U, V]$ or a scalar that is used for both $U$ and $V$. Closeness is measured using only the first column (reference dimension) in each data set. Default is Inf if 'Band' is specified; otherwise default is 10.

This constraint reduces the time and memory complexity of the algorithm from $O(MN)$ to $O(\sqrt{MN})^K \sqrt{UUV}$, where $M$ and $N$ are the number of observations in $X$ and $Y$ respectively, and $U$ and $V$ are small such that $U<<M$ and $V<<N$.

**Note** If you specify both 'Band' and 'Width', only pairs of observations that meet both constraints are considered potential matches and passed to the algorithm for scoring.

**Tip** Specify 'Width' when you do not have a good estimate for the 'Band' property. To get an indication of the memory required to run the algorithm with specific 'Band' and 'Width' parameters on your data sets, run samplealign, but do not specify return values and set 'ShowConstraints' to true.

$$[I, J] = \text{samplealign}(X, Y, \ldots \text{'Gap'}, \text{GapValue}, \ldots)$$

specifies the position-dependent terms for assigning gap penalties.

**GapValue** is any of the following:

- Cell array, $\{G, H\}$, where $G$ is either a scalar or a function handle specified using $@(X)$, and $H$ is either a scalar or a function handle specified using $@(Y)$. The functions $g(X)$ and $@Y$ must calculate the penalty for each observation (row) when it is matched to a gap in the other data set. The functions $g(X)$ and $@Y$ must return a column vector with the same number of rows as $X$ or $Y$, containing the gap penalty for each observation (row).
- Single function handle specified using $@(Z)$, that is used for both $G$ and $H$. The function $@Z$ must calculate the penalty for each observation (row) in both $X$ and $Y$ when it is matched to a gap in the other data set. The function $@Z$ must take as arguments $X$ and $Y$. The function $@Z$ must return a column vector with the same number of rows as $X$ or $Y$, containing the gap penalty for each observation (row).
- Scalar that is used for both $G$ and $H$.

The calculated value, $GPX$, is the gap penalty for matching observations from the first data set $X$ to gaps inserted in the second data set $Y$, and is the product of two terms: $GPX = G \ast QMS$. The term $G$ takes its value as a function of the observations in $X$. Similarly, $GPY$ is the gap penalty for matching observations from $Y$ to gaps inserted in $X$, and is the product of two terms: $GPY = H \ast QMS$. The term $H$ takes its value as a function of the observations in $Y$. By default, the term $QMS$ is the 0.75 quantile of the score for the pairs of observations that are potential matches (that is, pairs that comply with the 'Band' and 'Width' constraints).
If \( G \) and \( H \) are positive scalars, then \( GPX \) and \( GPY \) are independent of the observation where the gap is being inserted.

Default \( \text{GapValue} \) is 1, that is, both \( G \) and \( H \) are 1, which indicates that the default penalty for gap insertions in both sequences is equivalent to the quantile (set by the 'Quantile' property, default = 0.75) of the score for the pairs of observations that are potential matches.

**Note** \( \text{GapValue} \) defaults to a relatively safe value. However, the success of the algorithm depends on the fine tuning of the gap penalties, which is application dependent. When the gap penalties are large relative to the score of the correct matches, \( \text{samplealign} \) returns alignments with fewer gaps, but with more incorrectly aligned regions. When the gap penalties are smaller, the output alignment contains longer regions with gaps and fewer matched observations. Set 'ShowNetwork' to true to compare the gap penalties to the score of matched observations in different regions of the alignment.

\[
[I, J] = \text{samplealign}(X, Y, ...'Quantile', \text{QuantileValue}, ...) \]

specifies the quantile value used to calculate the term \( QMS \), which is used by the 'Gap' property to calculate gap penalties. \( \text{QuantileValue} \) is a scalar between 0 and 1. Default is 0.75.

**Tip** Set \( \text{QuantileValue} \) to an empty array ([[]]) to make the gap penalties independent of \( QMS \), that is, \( GPX \) and \( GPY \) are functions of only the \( G \) and \( H \) input parameters respectively.

\[
[I, J] = \text{samplealign}(X, Y, ...'Distance', \text{DistanceValue}, ...) \]

specifies a function to calculate the distance between pairs of observations that are potential matches. \( \text{DistanceValue} \) is a function handle specified using \( @() \). The function \( @() \) must take as arguments, \( R \) and \( S \), matrices that have the same number of rows and columns, and whose paired rows represent all potential matches of observations in \( X \) and \( Y \) respectively. The function \( @() \) must return a column vector of positive values with the same number of elements as rows in \( R \) and \( S \). Default is the Euclidean distance between the pairs.

**Caution** All columns in \( X \) and \( Y \), including the reference dimension, are considered when calculating distances. If you do not want to include the reference dimension in the distance calculations, use the 'Weight' property to exclude it.

\[
[I, J] = \text{samplealign}(X, Y, ...'Weights', \text{WeightsValue}, ...) \]

controls the inclusion/exclusion of columns (features) or the emphasis of columns (features) when calculating the distance score between observations that are potential matches, that is when using the 'Distance' property. \( \text{WeightsValue} \) can be a logical row vector that specifies columns in \( X \) and \( Y \). \( \text{WeightsValue} \) can also be a numeric row vector with the same number of elements as columns in \( X \) and \( Y \), that specifies the relative weights of the columns (features). Default is a logical row vector with all elements set to true.

**Tip** Using a numeric row vector for \( \text{WeightsValue} \) and setting some values to 0 can simplify the distance calculation when the data sets have many columns (features).
Note The weight values are not considered when computing the constrained alignment space, that is when using the 'Band' or 'Width' properties, or when calculating the gap penalties, that is when using the 'Gap' property.

\[ I, J \] = samplealign(\( X, Y \), ...) 'ShowConstraints', ShowConstraintsValue, ...) controls the display of the search space constrained by the input parameters 'Band' and 'Width', giving an indication of the memory required to run the algorithm with specific 'Band' and 'Width' on your data sets. Choices are true or false (default).

\[ I, J \] = samplealign(\( X, Y \), ...) 'ShowNetwork', ShowNetworkValue, ...) controls the display of the dynamic programming network, the match scores, the gap penalties, and the winning path. Choices are true or false (default).

\[ I, J \] = samplealign(\( X, Y \), ...) 'ShowAlignment', ShowAlignmentValue, ...) controls the display of the first and second columns of the \( X \) and \( Y \) data sets in the abscissa and the ordinate respectively, of a two-dimensional plot. Links between all the potential matches that meet the constraints are displayed, and the matches belonging to the output alignment are highlighted. Choices are true, false (default), or an integer specifying a column of the \( X \) and \( Y \) data sets to plot as the ordinate.

Examples

Example 1.83. Warping a sine wave with a smooth function to more closely follow cyclical sunspot activity

1 Load sunspot.dat, a data file included with the MATLAB software, that contains the variable sunspot, which is a two-column matrix containing variations in sunspot activity over the last 300 years. The first column is the reference dimension (years), and the second column contains sunspot activity values. Sunspot activity is cyclical, reaching a maximum about every 11 years.

\[
\text{load sunspot.dat}
\]

2 Create a sine wave with a known period of sunspot activity.

\[
\text{years} = (1700:1990)';
\]
\[
\text{T} = 11.038;
\]
\[
\text{f} = @(y) 60 + 60 * \sin(y*(2*pi/T));
\]

3 Align the observations between the sine wave and the sunspot activity by introducing gaps.

\[
[i,j] = \text{samplealign([years f(years)],sunspot,'weights',...}
[0 1],'\text{showalignment}',true);
\]
Estimate a smooth function to warp the sine wave.

```matlab
[p,s,mu] = polyfit(years(i),years(j),15);
wy = @(y) polyval(p,(y-mu(1))./mu(2));
```

Plot the sunspot cycles, unwarped sine wave, and warped sine wave.

```matlab
years = (1700:1/12:1990)';
figure
plot(sunspot(:,1),sunspot(:,2),years,f(years),wy(years),...
    f(years))
legend('Sunspots','Unwarped Sine Wave','Warped Sine Wave')
title('Smooth Warping Example')
```
Example 1.84. Recovering a nonlinear warping between two signals containing noisy Gaussian peaks

1 Create two signals with noisy Gaussian peaks.

```matlab
rng('default')
peakLoc = [30 60 90 130 150 200 230 300 380 430];
peakInt = [7 1 3 10 3 6 1 8 3 10];
time = 1:450;
comp = exp(-(bsxfun(@minus,time,peakLoc')./5).^2);
sig_1 = (peakInt + rand(1,10)) * comp + rand(1,450);
sig_2 = (peakInt + rand(1,10)) * comp + rand(1,450);
```

2 Define a nonlinear warping function.

```matlab
wf = @(t) 1 + (t<=100).*0.01.*(t.^2) + (t>100).*... (310+150*tanh(t./100-3));
```

3 Warp the second signal to distort it.

```matlab
sig_2 = interp1(time,sig_2,wf(time),'pchip');
```

4 Align the observations between the two signals by introducing gaps.

```matlab
[i,j] = samplealign([time;sig_1]',[time;sig_2]',...
  'weights',[0,1],'band',35,'quantile',.5);
```

5 Plot the reference signal, distorted signal, and warped (corrected) signal.

```matlab
figure
sig_3 = interp1(time,sig_2,interp1(i,j,time,'pchip'),'pchip');
```
6 Plot the real and the estimated warping functions.

```matlab
figure
plot(time, wf(time), time, interp1(j, i, time, 'pchip'))
legend('Distorting Function', 'Estimated Warping')
```
Note For examples of using function handles for the Band, Gap, and Distance properties, see “Visualizing and Preprocessing Hyphenated Mass Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling”.

References


See Also
msalign | msheatmap | mspalign | msppresample | msresample

Introduced in R2007b
sampleData

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set sample metadata in ExpressionSet object

Syntax

MetaDataObj = sampleData(ESObj)
NewESObj = sampleData(ESObj, NewMetaDataObj)

Description

MetaDataObj = sampleData(ESObj) returns a MetaData object containing the sample metadata from an ExpressionSet object.

NewESObj = sampleData(ESObj, NewMetaDataObj) replaces the sample metadata in \texttt{ESObj}, an ExpressionSet object, with \texttt{NewMetaDataObj}, and returns \texttt{NewESObj}, a new ExpressionSet object.

Input Arguments

ESObj
Object of the bioma.ExpressionSet class.

Default:

NewMetaDataObj
Object of the bioma.data.MetaData class, containing sample metadata, stored in two dataset arrays. The sample names and variable names in \texttt{NewMetaDataObj} must match the sample names and variable names in the \texttt{MetaDataObj} being replaced in the ExpressionSet object, \texttt{ESObj}.

Default:

Output Arguments

MetaDataObj
Object of the bioma.data.MetaData class, containing the sample metadata, stored in two dataset arrays.

NewESObj
Object of the bioma.ExpressionSet class, returned after replacing the MetaData object containing the sample metadata.
Examples

Construct an ExpressionSet object, ESObj, as described in the “Examples” on page 1-0 section of the bioma.ExpressionSet class reference page. Retrieve the MetaData object that contains sample metadata, stored in the ExpressionSet object:

```matlab
% Retrieve the sample data
NewMDObj = sampleData(ESObj);
```

See Also

bioma.ExpressionSet | bioma.data.ExptData | featureData | sampleNames

Topics

“Managing Gene Expression Data in Objects”
sampleNames

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set sample names in ExpressionSet object

Syntax

`SamNames = sampleNames(ESObj)`
`SamNames = sampleNames(ESObj, Subset)`
`NewESObj = sampleNames(ESObj, Subset, NewSamNames)`

Description

`SamNames = sampleNames(ESObj)` returns a cell array of strings specifying all sample names in an ExpressionSet object.

`SamNames = sampleNames(ESObj, Subset)` returns a cell array of strings specifying a subset the sample names in an ExpressionSet object.

`NewESObj = sampleNames(ESObj, Subset, NewSamNames)` replaces the sample names specified by `Subset` in `ESObj`, an ExpressionSet object, with `NewSamNames`, and returns `NewESObj`, a new ExpressionSet object.

Input Arguments

`ESObj`

Object of the `bioma.ExpressionSet` class.

`Subset`

One of the following to specify a subset of the sample names in an ExpressionSet object:

- String or character vector specifying a sample name
- Cell array of character vectors or string vector specifying sample names
- Positive integer
- Vector of positive integers
- Logical vector

`NewSamNames`

New sample names for specific sample names within an ExpressionSet object, specified by one of the following:

- Numeric vector
- Cell array of character vectors or string vector
- Character vector or string, which `sampleNames` uses as a prefix for the sample names, with sample numbers appended to the prefix
• Logical true or false (default). If true, sampleNames assigns unique sample names using the format Sample1, Sample2, etc.

The number of sample names in NewSamNames must equal the number of samples specified by Subset.

Output Arguments

SamNames

Cell array of strings specifying all or some of the sample names in an ExpressionSet object. The sample names are the column names in the DataMatrix objects in the ExpressionSet object. The sample names are also the row names of the VarValues dataset array in the MetaData object in the ExpressionSet object.

NewESObj

Object of the bioma.ExpressionSet class, returned after replacing specific sample names.

Examples

Construct an ExpressionSet object, ESObj, as described in the “Examples” on page 1-0 section of the bioma.ExpressionSet class reference page. Retrieve the sample names from it:

```matlab
% Retrieve the sample names
SNames = sampleNames(ESObj);
```

See Also

DataMatrix | bioma.ExpressionSet | bioma.data.ExptData | bioma.data.MetaData | featureNames

Topics

“Managing Gene Expression Data in Objects”
sampleNames

Class: bioma.data.ExptData
Package: bioma.data

Retrieve or set sample names in ExptData object

Syntax

SamNames = sampleNames(EDObj)
SamNames = sampleNames(EDObj, Subset)
NewEDObj = sampleNames(EDObj, Subset, NewSamNames)

Description

SamNames = sampleNames(EDObj) returns a cell array of character vectors specifying all sample names in an ExptData object.

SamNames = sampleNames(EDObj, Subset) returns a cell array of character vectors specifying a subset the sample names in an ExptData object.

NewEDObj = sampleNames(EDObj, Subset, NewSamNames) replaces the sample names specified by Subset in EDObj, an ExptData object, with NewSamNames, and returns NewEDObj, a new ExptData object.

Input Arguments

EDObj

Object of the bioma.data.ExptData class.

Default:

Subset

One of the following to specify a subset of the sample names in an ExptData object:

• Character vector specifying a sample name
• Cell array of character vectors specifying sample names
• Positive integer
• Vector of positive integers
• Logical vector

Default:

NewSamNames

New sample names for specific sample names within an ExptData object, specified by one of the following:
• Numeric vector
• Character vector or cell array of character vectors
• Character vector, which `sampleNames` uses as a prefix for the sample names, with sample numbers appended to the prefix
• Logical `true` or `false` (default). If `true`, `sampleNames` assigns unique sample names using the format `Sample1`, `Sample2`, etc.

The number of sample names in `NewSamNames` must equal the number of samples specified by `Subset`.

**Default:**

**Output Arguments**

*SamNames*

Cell array of character vectors specifying all or some of the sample names in an ExptData object. The sample names are the column names in the DataMatrix objects in the ExptData object.

*NewEDObj*

Object of the `bioma.data.ExptData` class, returned after replacing specific sample names.

**Examples**

Construct an ExptData object, and then retrieve the sample names from it:

```matlab
% Import bioma.data package to make constructor functions available
import bioma.data.*
% Create DataMatrix object from .txt file containing expression values from microarray experiment
dmObj = DataMatrix('File', 'mouseExprsData.txt');
% Construct ExptData object
EDObj = ExptData(dmObj);
% Retrieve sample names
SNames = sampleNames(EDObj);
```

**See Also**

`DataMatrix`, `bioma.data.ExptData`, `dmNames`, `elementNames`, `featureNames`

**Topics**

“Representing Expression Data Values in ExptData Objects”
sampleNames

Class: bioma.data.MetaData
Package: bioma.data

Retrieve or set sample names in MetaData object

Syntax

\[
\text{SamFeatNames} = \text{sampleNames}(\text{MDObj})
\]
\[
\text{SamFeatNames} = \text{sampleNames}(\text{MDObj}, \text{Subset})
\]
\[
\text{NewMDObj} = \text{sampleNames}(\text{MDObj}, \text{Subset}, \text{NewSamFeatNames})
\]

Description

\text{SamFeatNames} = \text{sampleNames}(\text{MDObj}) \text{ returns a cell array of character vectors specifying all sample names in a MetaData object.}

\text{SamFeatNames} = \text{sampleNames}(\text{MDObj}, \text{Subset}) \text{ returns a cell array of character vectors specifying a subset the sample names in a MetaData object.}

\text{NewMDObj} = \text{sampleNames}(\text{MDObj}, \text{Subset}, \text{NewSamFeatNames}) \text{ replaces the sample names specified by \text{Subset} in \text{MDObj}, a MetaData object, with \text{NewSamFeatNames}, and returns \text{NewMDObj}, a new MetaData object.}

Input Arguments

\text{MDObj}

Object of the bioma.data.MetaData class.

\text{Subset}

One of the following to specify a subset of the sample names in a MetaData object:

- Character vector specifying a sample name
- Cell array of character vectors specifying sample names
- Positive integer
- Vector of positive integers
- Logical vector

\text{NewSamFeatNames}

New sample names for specific names within a MetaData object, specified by one of the following:

- Numeric vector
- Cell array of character vectors or character array
- Character vector which \text{sampleNames} uses as a prefix for the sample or feature names, with numbers appended to the prefix
• Logical true or false (default). If true, sampleNames assigns unique names using the format Sample1, Sample2, etc.

The number of names in NewSamFeatNames must equal the number of samples specified by Subset.

**Output Arguments**

**SamFeatNames**

Cell array of character vectors specifying all or some of the sample names in a MetaData object. The sample names are also the row names of the VarValues dataset array in the MetaData object.

**NewMDObj**

Object of the bioma.data.MetaData class, returned after replacing specific sample names.

**Examples**

Construct a MetaData object, and then retrieve the sample names from it:

```matlab
% Import bioma.data package to make constructor function
% available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Retrieve the sample names
SNames = sampleNames(MDObj2)
```

**See Also**

bioma.data.MetaData | variableDesc | variableNames | variableValues

**Topics**

“Representing Sample and Feature Metadata in MetaData Objects”
sampleVarDesc

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set sample variable descriptions in ExpressionSet object

Syntax

\[
DSVarDescriptions = \text{sampleVarDesc}(ESObj) \\
NewESObj = \text{sampleVarDesc}(ESObj, \text{NewDSVarDescriptions})
\]

Description

\[
DSVarDescriptions = \text{sampleVarDesc}(ESObj)
\]
returns a dataset array containing the sample variable names and descriptions from the MetaData object in an ExpressionSet object.

\[
NewESObj = \text{sampleVarDesc}(ESObj, \text{NewDSVarDescriptions})
\]
replaces the sample variable descriptions in \(ESObj\), an ExpressionSet object, with \(NewDSVarDescriptions\), and returns \(NewESObj\), a new ExpressionSet object.

Input Arguments

ESObj
Object of the bioma.ExpressionSet class.

Default:

NewDSVarDescriptions

Descriptions of the sample variable names, specified by either of the following:

- A new dataset array containing the sample variable names and descriptions. In this dataset array, each row corresponds to a variable. The first column contains the variable name, and the second column (VariableDescription) contains a description of the variable. The row names (variable names) must match the row names (variable names) in \(DSVarDescriptions\), the dataset array being replaced in the MetaData object in the ExpressionSet object, \(ESObj\).

- Cell array of character vectors containing descriptions of the sample variables. The number of elements in VarDesc must equal the number of row names (variable names) in \(DSVarDescriptions\), the dataset array being replaced in the MetaData object in the ExpressionSet object, \(ESObj\).

Default:

Output Arguments

DSVarDescriptions

A dataset array containing the sample variable names and descriptions from the MetaData object of an ExpressionSet object. In this dataset array, each row corresponds to a variable. The first column
contains the variable name, and the second column (VariableDescription) contains a description of the variable.

**NewESObj**

Object of the bioma.ExpressionSet class, returned after replacing the dataset array containing the sample variable descriptions.

**Examples**

Construct an ExpressionSet object, ESObj, as described in the “Examples” on page 1-0 section of the bioma.ExpressionSet class reference page. Retrieve the sample variable descriptions in the ExpressionSet object:

```matlab
% Retrieve the sample variable descriptions
SVarDescriptions = sampleVarDesc(ESObj)
```

**See Also**

bioma.ExpressionSet | bioma.data.MetaData | variableDesc

**Topics**

“Managing Gene Expression Data in Objects”
sampleVarNames

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set sample variable names in ExpressionSet object

Syntax

SamVarNames = sampleVarNames(ESObj)
SamVarNames = sampleVarNames(ESObj, Subset)
NewESObj = sampleVarNames(ESObj, Subset, NewSamVarNames)

Description

SamVarNames = sampleVarNames(ESObj) returns a cell array of character vectors specifying all sample variable names in an ExpressionSet object.

SamVarNames = sampleVarNames(ESObj, Subset) returns a cell array of character vectors specifying a subset the sample variable names in an ExpressionSet object.

NewESObj = sampleVarNames(ESObj, Subset, NewSamVarNames) replaces the sample variable names specified by Subset in ESObj, an ExpressionSet object, with NewSamVarNames, and returns NewESObj, a new ExpressionSet object.

Input Arguments

ESObj
Object of the bioma.ExpressionSet class.

Subset

One of the following to specify a subset of the sample variable names in an ExpressionSet object:

- Character vector or string specifying a sample variable name
- Cell array of character vectors or string vector specifying sample variable names
- Positive integer
- Vector of positive integers
- Logical vector

NewSamVarNames

New sample variable names for specific sample variable names within an ExpressionSet object, specified by one of the following:

- Numeric vector
- String vector or cell array of character vectors
- Character vector or string, which sampleVarNames uses as a prefix for the sample variable names, with sample variable numbers appended to the prefix
• Logical true or false (default). If true, sampleVarNames assigns unique sample variable names using the format Var1, Var2, etc.

The number of sample variable names in NewSamVarNames must equal the number of sample variable names specified by Subset.

Output Arguments

SamVarNames

Cell array of character vectors specifying all or some of the sample variable names in an ExpressionSet object. The sample variable names are the column names of the VarValues dataset array. The sample variable names are also the row names of the VarDescriptions dataset array. Both dataset arrays are in the MetaData object in the ExpressionSet object.

NewESObj

Object of the bioma.ExpressionSet class, returned after replacing specific sample names.

Examples

Construct an ExpressionSet object, ESObj, as described in the “Examples” on page 1-0 section of the bioma.ExpressionSet class reference page. Retrieve the sample variable names from the ExpressionSet object:

% Retrieve the sample variable names
VNames = sampleVarNames(ESObj)

See Also

bioma.ExpressionSet | bioma.data.MetaData | featureNames | featureVarNames | sampleNames

Topics

“Managing Gene Expression Data in Objects”
sampleVarValues

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set sample variable values in ExpressionSet object

Syntax

\[ DSVarValues = \text{sampleVarValues}(ESObj) \]
\[ NewESObj = \text{sampleVarValues}(ESObj, \text{NewDSVarValues}) \]

Description

\[ DSVarValues = \text{sampleVarValues}(ESObj) \] returns a dataset array containing the measured value of each variable per sample from the MetaData object of an ExpressionSet object.

\[ NewESObj = \text{sampleVarValues}(ESObj, \text{NewDSVarValues}) \] replaces the sample variable values in \( ESObj \), an ExpressionSet object, with \( \text{NewDSVarValues} \), and returns \( NewESObj \), a new ExpressionSet object.

Input Arguments

\( ESObj \)

Object of the \text{bioma.ExpressionSet} class.

Default:

\( \text{NewDSVarValues} \)

A new dataset array containing a value for each variable per sample. In this dataset array, the columns correspond to variables and rows correspond to samples. The row names (sample names) must match the row names (sample names) in \( DSVarValues \), the dataset array being replaced in the MetaData object in the ExpressionSet object, \( ESObj \).

Default:

Output Arguments

\( DSVarValues \)

A dataset array containing the measured value of each variable per sample from the MetaData object of an ExpressionSet object. In this dataset array, the columns correspond to variables and rows correspond to samples.

\( NewESObj \)

Object of the \text{bioma.ExpressionSet} class, returned after replacing the dataset array containing the sample variable values.
Examples

Construct an ExpressionSet object, ESObj, as described in the “Examples” on page 1-0 section of the bioma.ExpressionSet class reference page. Retrieve the sample variable values in ExpressionSet object:

```matlab
% Retrieve the sample variable values
SVarValues = sampleVarValues(ESObj);
```

See Also
bioma.ExpressionSet | bioma.data.MetaData | variableValues

Topics
“Managing Gene Expression Data in Objects”
**samread**

Read data from SAM file

**Syntax**

```matlab
SAMStruct = samread(File)
[SAMStruct, HeaderStruct] = samread(File)
... = samread(File,'ParameterName',ParameterValue)
```

**Description**

`SAMStruct = samread(File)` reads a SAM-formatted file and returns the data in a MATLAB array of structures.

`[SAMStruct, HeaderStruct] = samread(File)` returns the alignment and header data in two separate variables.

`... = samread(File,'ParameterName',ParameterValue)` accepts one or more comma-separated parameter name/value pairs. Specify `ParameterName` inside single quotes.

**Input Arguments**

**File**

Character vector or string specifying a file name, path and file name of a SAM-formatted file, or the text of a SAM-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.

**Parameter Name/Value Pairs**

**Tags**

Controls the reading of the optional tags in addition to the first 11 fields for each alignment in the SAM-formatted file. Choices are `true` (default) or `false`.

Default:

**ReadGroup**

Character vector or string specifying the read group ID for which to read alignment records from. Default is to read records from all groups.

**Tip** For a list of the read groups (if present), return the header information in a separate `Header` structure and view the `ReadGroup` field in this structure.

Default:
BlockRead

Scalar or vector that controls the reading of a single sequence entry or block of sequence entries from a SAM-formatted file containing multiple sequences. Enter a scalar \( N \), to read the \( N \)th entry in the file. Enter a 1-by-2 vector \([M_1, M_2]\), to read a block of entries starting at the \( M_1 \) entry and ending at the \( M_2 \) entry. To read all remaining entries in the file starting at the \( M_1 \) entry, enter a positive value for \( M_1 \) and enter \( \text{Inf} \) for \( M_2 \).

Default:

Output Arguments

SAMStruct

An \( N \)-by-1 array of structures containing sequence alignment and mapping information from a SAM-formatted file, where \( N \) is the number of alignment records stored in the SAM-formatted file. Each structure contains the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryName</td>
<td>Name of read sequence (if unpaired) or name of sequence pair (if paired).</td>
</tr>
<tr>
<td>Flag</td>
<td>Integer indicating the bit-wise information that specifies the status of each of 11 flags described by the SAM format specification.</td>
</tr>
<tr>
<td>ReferenceName</td>
<td>Name of the reference sequence.</td>
</tr>
<tr>
<td>Position</td>
<td>Position (one-based offset) of the forward reference sequence where the left-most base of the alignment of the read sequence starts.</td>
</tr>
<tr>
<td>MappingQuality</td>
<td>Integer specifying the mapping quality score for the read sequence.</td>
</tr>
<tr>
<td>CigarString</td>
<td>CIGAR-formatted character vector representing how the read sequence aligns with the reference sequence.</td>
</tr>
<tr>
<td>MateReferenceName</td>
<td>Name of the reference sequence associated with the mate. If this name is the same as ReferenceName, then this value is ( = ). If there is no mate, then this value is ( * ).</td>
</tr>
<tr>
<td>MatePosition</td>
<td>Position (one-based offset) of the forward reference sequence where the left-most base of the alignment of the mate of the read sequence starts.</td>
</tr>
</tbody>
</table>
### Field | Description
--- | ---
**InsertSize** | The number of base positions between the read sequence and its mate, when both are mapped to the same reference sequence. Otherwise, this value is 0.

**Sequence** | Character vector containing the letter representations of the read sequence. It is the reverse-complement if the read sequence aligns to the reverse strand of the reference sequence.

**Quality** | Character vector containing the ASCII representation of the per-base quality score for the read sequence. The quality score is reversed if the read sequence aligns to the reverse strand of the reference sequence.

**Tags** | List of applicable SAM tags and their values.

### HeaderStruct
Structure containing header information for the SAM-formatted file in the following fields.

| Field | Description |
--- | --- |
**Header** | Structure containing the file format version, sort order, and group order. |
**SequenceDictionary** | Structure containing the:
- Sequence name
- Sequence length
- Genome assembly identifier
- MD5 checksum of sequence
- URI of sequence
- Species |

**ReadGroup** | Structure containing the:
- Read group identifier
- Sample
- Library
- Description
- Platform unit
- Predicted median insert size
- Sequencing center
- Date
- Platform |
Field | Description
--- | ---
Program* | Structure containing the:
* Program name
* Version
* Command line

— These structures and their fields appear in the output structure only if they are present in the SAM file. The information in these structures depends on the information present in the SAM file.

**Examples**

Read the header information and the alignment data from the `ex1.sam` file included with Bioinformatics Toolbox, and then return the information in two separate variables:

```matlab
[data header] = samread('ex1.sam');
```

Read a block of entries, excluding the tags, from the `ex1.sam` file, and then return the information in an array of structures:

```matlab
% Read entries 5 through 10 and do not include the tags
data = samread('ex1.sam','blockread', [5 10], 'tags', false);
```

**Tips**

- Use the `saminfo` function to investigate the size and content of a SAM-formatted file before using the `samread` function to read the file contents into a MATLAB array of structures.
- If your SAM-formatted file is too large to read using available memory, try one of the following:
  - Use the `BlockRead` parameter with the `samread` function to read a subset of entries.
  - Create a BioIndexedFile object from the SAM-formatted file, then access the entries using methods of the BioIndexedFile class.
  - Use the `SAMStruct` output argument that `samread` returns to create a BioMap object, which lets you explore, access, filter, and manipulate all or a subset of the data, before doing subsequent analyses or viewing the data.

**See Also**

BioIndexedFile | BioMap | bamindexread | baminfo | bamread | fastainfo | fastaread | fastawrite | fastqinfo | fastqread | fastqwrite | saminfo | sffinfo | sffread | soapread

**Topics**

“Manage Sequence Read Data in Objects”
“Work with Next-Generation Sequencing Data”

**External Websites**

Sequence Read Archive
SAM format specification

**Introduced in R2010a**
samsort

Sort SAM files

Syntax

\[ \text{sortedFile} = \text{samsort}(\text{inFile}) \]
\[ \text{samsort}(\text{inFile},\text{outFile}) \]

Description

\[ \text{sortedFile} = \text{samsort}(\text{inFile}) \] sorts a SAM file \( \text{inFile} \) and returns the name of the sorted SAM file \( \text{sortedFile} \). The function sorts the alignment records by the reference sequence name first, and then by position within the reference.

\[ \text{samsort}(\text{inFile},\text{outFile}) \] sorts \( \text{inFile} \) and produces a sorted SAM file named \( \text{outFile} \).

Examples

Sort SAM File

Sort a sample SAM file. The sorted file has the same base name as the input file, but with the extension "\.sorted.sam". By default, the sorted file is saved in the current directory.

\[ \text{sortedFile} = \text{samsort}("\text{ex1.sam}) \]
\[ \text{sortedFile} = "\text{ex1.sorted.sam}" \]

You can change the name of output file by specifying it as the second input.

\[ \text{samsort}("\text{ex1.sam},"\text{sortedEx1.sam}) \]
\[ \text{ans} = "\text{sortedEx1.sam}" \]

You can also save the output file to an existing directory by providing the file path information.

\[ \text{mkdir}("./\text{OutputEx1SAM})\); \]
\[ \text{samsort}("\text{ex1.sam},"./\text{OutputEx1SAM/sortedEx1.sam}) \]
\[ \text{ans} = "./\text{OutputEx1SAM/sortedEx1.sam}" \]

Input Arguments

\text{inFile} — Name of input SAM file to sort
character vector | string

Name of the input SAM file to sort, specified as a string or character vector. You can specify a file name or a path and file name. The file name must have the extension \text{.sam}.
Example: "./InputData/ex1.sam"
Data Types: char | string

**outFile — Name of output SAM file**
character vector | string

Name of the output SAM file, specified as a string or character vector. You can specify a file name or a path and file name. The file name must have the extension .sam. The file is saved in the current directory by default unless you specify the path information. If you specify the file path, the listed directories must exist before you run the function.

Example: "./OutputData/ex1Sorted.sam"
Data Types: char | string

**Output Arguments**

**sortedFile — Name of output SAM file**
string

Name of the output SAM file, returned as a string. `sortedFile` has the same base name as `inFile` but with the extension .sorted.sam. The file is saved in the current directory by default.

**See Also**

BioMap | bamread | saminfo | samread

**Topics**

“Data Import”
“Manage Sequence Read Data in Objects”

**External Websites**

Sequence Read Archive
SAM format specification

**Introduced in R2019b**
**scfread**

Read trace data from SCF file

**Syntax**

```matlab
Sample = scfread(File)
[Sample, Probability] = scfread(File)
[Sample, Probability, Comments] = scfread(File)
[A, C, G, T] = scfread (File)
```

**Arguments**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File</strong></td>
<td>Character vector or string specifying the file name or a path and file name of an SCF formatted file.</td>
</tr>
</tbody>
</table>

**Description**

scfread reads data from an SCF formatted file into MATLAB structures.

`Sample = scfread(File)` reads an SCF formatted file and returns the sample data in the structure `Sample`, which contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Column vector containing intensity of A fluorescence tag</td>
</tr>
<tr>
<td>C</td>
<td>Column vector containing intensity of C fluorescence tag</td>
</tr>
<tr>
<td>G</td>
<td>Column vector containing intensity of G fluorescence tag</td>
</tr>
<tr>
<td>T</td>
<td>Column vector containing intensity of T fluorescence tag</td>
</tr>
</tbody>
</table>

`[Sample, Probability] = scfread(File)` also returns the probability data in the structure `Probability`, which contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>peak_index</td>
<td>Column vector containing the position in the SCF file for the start of the data for each peak</td>
</tr>
<tr>
<td>prob_A</td>
<td>Column vector containing the probability of each base in the sequence being an A</td>
</tr>
<tr>
<td>prob_C</td>
<td>Column vector containing the probability of each base in the sequence being a C</td>
</tr>
<tr>
<td>prob_G</td>
<td>Column vector containing the probability of each base in the sequence being a G</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>prob_T</td>
<td>Column vector containing the probability of each base in the sequence being a T</td>
</tr>
<tr>
<td>base</td>
<td>Column vector containing the called bases for the sequence</td>
</tr>
</tbody>
</table>

\[
\text{[Sample, Probability, Comments]} = \text{scfread(File)}
\]
also returns the comment information from the SCF file in a character array Comments.

\[
\text{[A, C, G, T]} = \text{scfread (File)}
\]
returns the sample data for the four bases in separate variables.

\[
\text{[A, C, G, T, ProbA, ProbC, ProbG, ProbT]} = \text{scfread (File)}
\]
also returns the probabilities data for the four bases in separate variables.

\[
\text{[A, C, G, T, ProbA, ProbC, ProbG, ProbT, Comments, PkIndex, Base]} = \text{scfread (File)}
\]
also returns the peak indices and called bases in separate variables.

SCF files store data from DNA sequencing instruments. Each file includes sample data, sequence information, and the relative probabilities of each of the four bases.

**Examples**

\[
\text{[sampleStruct, probStruct, Comments]} = \text{scfread('sample.scf')}
\]

```matlab
sampleStruct =

    A: [10827x1 double]
    C: [10827x1 double]
    G: [10827x1 double]
    T: [10827x1 double]
```

```matlab
probStruct =

    peak_index: [742x1 double]
    prob_A: [742x1 double]
    prob_C: [742x1 double]
    prob_G: [742x1 double]
    prob_T: [742x1 double]
    base: [742x1 char]
```

```matlab
Comments =

SIGN=A=121,C=103,G=119,T=82
SPAC= 16.25
PRIM=0
MACH=Arkansas_SN312
DYEP=DT3700POP5{BD}v2.mob
NAME=HCIUP1D61207
LANE=6
GELN=
PROC= 
RTRK=
CONV=phred version=0.990722.h
```
COMM=
SRCE=ABI 373A or 377

See Also
genbankread | traceplot

Introduced before R2006a
select (phytree)

Select tree branches and leaves in phytree object

Syntax

\[ S = \text{select}(\text{Tree}, \text{N}) \]
\[ [S, \text{Selleaves}, \text{Selbranches}] = \text{select}(...) \]

\text{select}(..., \text{'Reference'}, \text{ReferenceValue}, ...) \]
\text{select}(..., \text{'Criteria'}, \text{CriteriaValue}, ...) \]
\text{select}(..., \text{'Threshold'}, \text{ThresholdValue}, ...) \]
\text{select}(..., \text{'Exclude'}, \text{ExcludeValue}, ...) \]
\text{select}(..., \text{'Propagate'}, \text{PropagateValue}, ...) \]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Tree}</td>
<td>Phylogenetic tree (phytree object) created with the function \text{phytree}.</td>
</tr>
<tr>
<td>\text{N}</td>
<td>Number of closest nodes to the root node.</td>
</tr>
<tr>
<td>\text{ReferenceValue}</td>
<td>Property to select a reference point for measuring distance.</td>
</tr>
<tr>
<td>\text{CriteriaValue}</td>
<td>Property to select a criteria for measuring distance.</td>
</tr>
<tr>
<td>\text{ThresholdValue}</td>
<td>Property to select a distance value. Nodes with distances below this value are selected.</td>
</tr>
<tr>
<td>\text{ExcludeValue}</td>
<td>Property to remove (exclude) branch or leaf nodes from the output. Enter 'none', 'branches', or 'leaves'. The default value is 'none'.</td>
</tr>
<tr>
<td>\text{PropagateValue}</td>
<td>Property to select propagating nodes toward the leaves or the root.</td>
</tr>
<tr>
<td>\text{S}</td>
<td>Logical vector for all selected nodes.</td>
</tr>
<tr>
<td>\text{Selleaves}</td>
<td>Logical vector for selected leaves.</td>
</tr>
<tr>
<td>\text{Selbranches}</td>
<td>Logical vector for selected branches.</td>
</tr>
</tbody>
</table>

Description

\[ S = \text{select}(\text{Tree}, \text{N}) \] returns a logical vector \( S \) of size \([\text{NumNodes} \times 1]\) indicating the \( N \) closest nodes to the root node of a phytree object \( \text{Tree} \) where \( \text{NumNodes} = \text{NumLeaves} + \text{NumBranches} \). The first criterion used is branch levels, then patristic distance (also known as tree distance). By default, \text{select} uses Inf as the value of \text{N}, and \text{select}(\text{Tree}) \) returns a vector with values of true.

\[ [S, \text{Selleaves}, \text{Selbranches}] = \text{select}(...) \] returns two additional logical vectors, one for the selected leaves and one for the selected branches.

\text{select}(..., \text{'PropertyName'}, \text{PropertyValue}, ...) \) uses additional options specified as one or more name-value pair arguments. Each \text{PropertyName} must be enclosed in single quotation marks and is case insensitive. These name-value pairs are as follows:
select(..., 'Reference', ReferenceValue, ...) changes the reference point(s) to measure the closeness. ReferenceValue can be 'root' (default) or 'leaves' or an index that points to any node of the tree. When using 'leaves', a node can have multiple distances to its descendant leaves (nonultrametric tree). If so, select considers the minimum distance to any descendant leaf.

select(..., 'Criteria', CriteriaValue, ...) changes the criteria used to measure closeness. If CriteriaValue = 'levels' (default), the first criterion is branch levels and then patristic distance. If CriteriaValue = 'distance', the first criterion is patristic distance and then branch levels.

select(..., 'Threshold', ThresholdValue, ...) selects all the nodes where closeness is less than or equal to the threshold value (ThresholdValue). You can use either 'Criteria' or 'Reference' in conjunction with this name-value pair. If N is not specified, then N = Inf. Otherwise you can limit the number of selected nodes by N.

select(..., 'Exclude', ExcludeValue, ...) sets a postfilter which excludes all the branch nodes from S when ExcludeValue = 'branches' or excludes all the leave nodes when ExcludeValue = 'leaves'. The default is 'none'.

select(..., 'Propagate', PropagateValue, ...) activates a postfunctionality that propagates the selected nodes to the leaves when PropagateValue is set to 'toleaves' or toward the root finding a common ancestor when PropagateValue is set to 'toroot'. The default value is 'none'. PropagateValue may also be 'both'. The 'Propagate' property acts after the 'Exclude' name-value pair.

Examples

% Load a phylogenetic tree created from a protein family:
tr = phytreeread('pf00002.tree');

% To find close products for a given protein (e.g. vipr2_human):
ind = getbyname(tr,'vipr2_human');
[sel,sel_leaves] = select(tr,'criteria','distance',...  
    'threshold',0.6,'reference',ind);
view(tr,sel_leaves)

% To find potential outliers in the tree, use
[sel,sel_leaves] = select(tr,'criteria','distance',...  
    'threshold',3,...  
    'reference','leaves',...  
    'exclude','leaves',...  
    'propagate','toleaves');
view(tr,-sel_leaves)

See Also
gt | pdist | phytree | phytreeviewer | prune

Topics
phytree object on page 1-1274

Introduced before R2006a
seq2regexp

Convert sequence with ambiguous characters to regular expression

Syntax

```
RegExp = seq2regexp(Seq)
RegExp = seq2regexp(Seq, ...'Alphabet', AlphabetValue, ...)
RegExp = seq2regexp(Seq, ...'Ambiguous', AmbiguousValue, ...)
```

Input Arguments

<table>
<thead>
<tr>
<th>Seq</th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string containing codes specifying an amino acid or nucleotide sequence.</td>
</tr>
<tr>
<td></td>
<td>• Structure containing a Sequence field that contains an amino acid or nucleotide sequence, such as returned by fastaread, fastqread, getembl, getgenbank, getgenpept, or getpdb.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AlphabetValue</th>
<th>Character vector or string specifying the sequence alphabet. Choices are:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• 'NT' (default) — Nucleotide</td>
</tr>
<tr>
<td></td>
<td>• 'AA' — Amino acid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AmbiguousValue</th>
<th>Controls whether ambiguous characters are included in RegExp, the regular expression return value. Choices are:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• true (default) — Include ambiguous characters in the return value</td>
</tr>
<tr>
<td></td>
<td>• false — Return only unambiguous characters</td>
</tr>
</tbody>
</table>

Output Arguments

| RegExp       | Character vector of codes specifying an amino acid or nucleotide sequence in regular expression format using IUB/IUPAC codes.                                                                                                     |

Description

```
RegExp = seq2regexp(Seq) converts ambiguous amino acid or nucleotide symbols in a sequence to a regular expression format using IUB/IUPAC codes.

RegExp = seq2regexp(Seq, ...'PropertyName', PropertyValue, ...)) calls seq2regexp with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

RegExp = seq2regexp(Seq, ...'Alphabet', AlphabetValue, ...) specifies the sequence
```
alphabet. *AlphabetValue* can be either 'NT' for nucleotide sequences or 'AA' for amino acid sequences. Default is 'NT'.

\[\text{RegExp} = \text{seq2regexp(} \text{Seq}, \ldots, \text{Ambiguous}', \text{AmbiguousValue}, \ldots)\] controls whether ambiguous characters are included in *RegExp*, the regular expression return value. Choices are true (default) or false. For example:

- If *Seq* = 'ACGTK', and *AmbiguousValue* is true, the MATLAB software returns ACGT[GTK] with the unambiguous characters G and T and the ambiguous character K.
- If *Seq* = 'ACGTK', and *AmbiguousValue* is false, the MATLAB software returns ACGT[GT] with only the unambiguous characters.

### Nucleotide Conversion

<table>
<thead>
<tr>
<th>Nucleotide Code</th>
<th>Nucleotide</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Adenosine</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>Cytosine</td>
<td>C</td>
</tr>
<tr>
<td>G</td>
<td>Guanine</td>
<td>G</td>
</tr>
<tr>
<td>T</td>
<td>Thymidine</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>Uridine</td>
<td>U</td>
</tr>
<tr>
<td>R</td>
<td>Purine</td>
<td>[AG]</td>
</tr>
<tr>
<td>Y</td>
<td>Pyrimidine</td>
<td>[TC]</td>
</tr>
<tr>
<td>K</td>
<td>Keto</td>
<td>[GT]</td>
</tr>
<tr>
<td>M</td>
<td>Amino</td>
<td>[AC]</td>
</tr>
<tr>
<td>S</td>
<td>Strong interaction (3 H bonds)</td>
<td>[GC]</td>
</tr>
<tr>
<td>W</td>
<td>Weak interaction (2 H bonds)</td>
<td>[AT]</td>
</tr>
<tr>
<td>B</td>
<td>Not A</td>
<td>[CGT]</td>
</tr>
<tr>
<td>D</td>
<td>Not C</td>
<td>[AGT]</td>
</tr>
<tr>
<td>H</td>
<td>Not G</td>
<td>[ACT]</td>
</tr>
<tr>
<td>V</td>
<td>Not T or U</td>
<td>[ACG]</td>
</tr>
<tr>
<td>N</td>
<td>Any nucleotide</td>
<td>[ACGT]</td>
</tr>
<tr>
<td>-</td>
<td>Gap of indeterminate length</td>
<td>-</td>
</tr>
<tr>
<td>?</td>
<td>Unknown</td>
<td>?</td>
</tr>
</tbody>
</table>

### Amino Acid Conversion

<table>
<thead>
<tr>
<th>Amino Acid Code</th>
<th>Amino Acid</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Asparagine or Aspartic acid (Aspartate)</td>
<td>[DN]</td>
</tr>
<tr>
<td>Z</td>
<td>Glutamine or Glutamic acid (Glutamate)</td>
<td>[EQ]</td>
</tr>
<tr>
<td>X</td>
<td>Any amino acid</td>
<td>[ARNDCEGHIKLMFPSTWYV]</td>
</tr>
</tbody>
</table>
Examples

1  Convert a nucleotide sequence to a regular expression.

    seq2regexp('ACwTMan')

    ans =

2  Convert the same nucleotide sequence, but remove ambiguous characters from the regular expression.

    seq2regexp('ACwTMan', 'ambiguous', false)

    ans =

See Also
regexp | regexpi | restrict | seqwordcount

Introduced before R2006a
seqalignviewer

Visualize and edit multiple sequence alignment

Syntax

seqalignviewer
seqalignviewer(Alignment)
seqalignviewer(Alignment,Name,Value)

seqalignviewer('close')

Description

seqalignviewer opens the Sequence Alignment app, where you can display and interactively adjust multiple sequence alignments.

seqalignviewer(Alignment) loads a group of previously multiply aligned sequences into the app, where you can view and interactively adjust the alignment.

seqalignviewer(Alignment,Name,Value) opens the app with additional options specified by one or more Name,Value pair arguments.

seqalignviewer('close') closes the Sequence Alignment app.

Tip If gaps are available after you have selected a block from aligned sequences, then there are three regions that you can drag and move horizontally:

- Selected block
- Block on the left of the selection
- Block on the right of the selection

Examples

View a Multiple Sequence Alignment File

Load and view a multiple sequence alignment file.

seqalignviewer('aagag.aln')
Alternatively, you can click Sequence Alignment on the Apps tab to open the app, and view the alignment data.

You can also generate a phylogenetic tree from aligned sequences from within the app. Select Display > View Tree.

**Input Arguments**

**Alignment — Multiple sequence alignment (MSA) data**

structure | character array | character vector | string vector | 3-by-N character array

Multiple sequence alignment (MSA) data, specified as:

- MATLAB structure containing a Sequence field, such as returned by fastaread, gethmmalignment, multialign, or multialignread
- Character array or column vector of strings containing MSA data, such as returned by multialign
- Character vector specifying a file or URL containing MSA data
- 3-by-N character array showing the pairwise alignment of two sequences, such as returned by nwalign or swalign.
Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: 'Alphabet', 'AA' specifies that the aligned sequences are amino acid sequences.

Alphabet — Type of aligned sequences

`'AA' | 'NT'`

Type of aligned sequences, specified as 'AA' for amino acid sequences or 'NT' for nucleotide sequences. If you do not specify the type, `seqalignviewer` attempts to determine the correct type. If it cannot, it defaults to 'AA'.

Example: 'Alphabet', 'AA'

SeqHeaders — List of names to label sequences in alignment window

Array of structures containing a Header or Name field | cell array of character vectors | string vector

List of names to label the sequences in the alignment window, specified as a MATLAB array of structures containing a Header or Name field, cell array of character vectors, or string vector. The number of elements in either array must be the same as the number of sequences in the alignment data Alignment.

Example: 'SeqHeaders', names

Alternatives

You can also display a color-coded multiple or pairwise sequence alignment using the `showalignment` function. However, the alignment displays in a MATLAB Figure window, where you cannot interact with it.

Compatibility Considerations

'R2012b' option in `seqalignviewer` has been removed

Errors starting in R2017b

The 'R2012b' name-value pair has been removed. The default version of `seqalignviewer` runs more robustly than the previous version (R2012b).

See Also

cigar2align | fastaread | gethmmalignment | multialign | multialignread |
multialignwrite | nwalkin | seqviewer | showalignment | swalign

Topics

“View and Align Multiple Sequences”
“Investigating the Bird Flu Virus”

Introduced in R2012b
seqcomplement

Calculate complementary strand of nucleotide sequence

Syntax

SeqC = seqcomplement(SeqNT)

Arguments

<table>
<thead>
<tr>
<th>SeqNT</th>
<th>Nucleotide sequence specified by any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string containing the letters A, C, G, T, U, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N.</td>
</tr>
<tr>
<td></td>
<td>• Row vector of integers from the table Mapping Nucleotide Integers to Letter Codes.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that contains a nucleotide sequence, such as returned by emblread, fastaread, fastqread, genbankread, getembl, or getgenbank.</td>
</tr>
</tbody>
</table>

Description

SeqC = seqcomplement(SeqNT) calculates the complementary strand of a DNA or RNA nucleotide sequence. The return sequence, SeqC, is in the same format as SeqNT. For example, if SeqNT is a vector of integers, then so is SeqC.

<table>
<thead>
<tr>
<th>Nucleotide in SeqNT</th>
<th>Converts to This Nucleotide in SeqC</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T or U</td>
</tr>
<tr>
<td>C</td>
<td>G</td>
</tr>
<tr>
<td>G</td>
<td>C</td>
</tr>
<tr>
<td>T or U</td>
<td>A</td>
</tr>
</tbody>
</table>

Examples

Return the complement of a DNA nucleotide sequence.

s = 'ATCG';
seqcomplement(s)

ans =
TAGC

See Also
codoncount | palindromes | seqrcomplement | seqreverse | seqviewer

Introduced before R2006a
seqconsensus

Calculate consensus sequence

Syntax

\[ CSeq = \text{seqconsensus}(\text{Seqs}) \]
\[ [CSeq, \text{Score}] = \text{seqconsensus}(\text{Seqs}) \]
\[ CSeq = \text{seqconsensus}(\text{Profile}) \]

seqconsensus(..., 'PropertyName', PropertyValue,...)
seqconsensus(..., 'ScoringMatrix', ScoringMatrixValue)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seqs</td>
<td>Set of multiply aligned amino acid or nucleotide sequences. Enter a character array, string vector, cell array of character vectors, or an array of structures with the field Sequence.</td>
</tr>
<tr>
<td>Profile</td>
<td>Sequence profile. Enter a profile from the function seqprofile. Profile is a matrix of size [20 (or 4) x Sequence Length] with the frequency or count of amino acids (or nucleotides) for every position. Profile can also have 21 (or 5) rows if gaps are included in the consensus.</td>
</tr>
</tbody>
</table>
**ScoringMatrixValue** | Either of the following:
--- | ---
- Character vector or string specifying the scoring matrix to use for the alignment. Choices for amino acid sequences are:
  - 'BLOSUM62'
  - 'BLOSUM30' increasing by 5 up to 'BLOSUM90'
  - 'BLOSUM100'
  - 'PAM10' increasing by 10 up to 'PAM500'
  - 'DAYHOFF'
  - 'GONNET'
- A $21 \times 21$, $5 \times 5$, $20 \times 20$, or $4 \times 4$ numeric array. For the gap-included cases, gap scores (last row/column) are set to mean(diag(ScoringMatrix)) for a gap matching with another gap, and set to mean(nodiag(ScoringMatrix)) for a gap matching with another symbol.

**Note** The above scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the 'Scale' property to specify an additional scale factor to convert the output score from bits to another unit.

- A $21 \times 21$, $5 \times 5$, $20 \times 20$, or $4 \times 4$ numeric array. For the gap-included cases, gap scores (last row/column) are set to mean(diag(ScoringMatrix)) for a gap matching with another gap, and set to mean(nodiag(ScoringMatrix)) for a gap matching with another symbol.

**Note** If you use a scoring matrix that you created, the matrix does not include a scale factor. The output score will be returned in the same units as the scoring matrix.

**Note** If you need to compile seqconsensus into a stand-alone application or software component using MATLAB Compiler, use a matrix instead of a character vector or string for **ScoringMatrixValue**.

## Description

$CSeq = \text{seqconsensus}(\text{Seqs})$, for a multiply aligned set of sequences ($\text{Seqs}$), returns a character vector with the consensus sequence ($CSeq$). The frequency of symbols (20 amino acids, 4 nucleotides) in the set of sequences is determined with the function `seqprofile`. For ambiguous nucleotide or amino acid symbols, the frequency or count is added to the standard set of symbols.

$[CSeq, Score] = \text{seqconsensus}(\text{Seqs})$ returns the conservation score of the consensus sequence. Scores are computed with the scoring matrix BLOSUM50 for amino acids or NUC44 for nucleotides. Scores are the average euclidean distance between the scored symbol and the M-
dimensional consensus value. \( M \) is the size of the alphabet. The consensus value is the profile weighted by the scoring matrix.

\[
CSeq = \text{seqconsensus}(\text{Profile})
\]

returns a character vector with the consensus sequence \((CSeq)\) from a sequence profile \((\text{Profile})\).

\[
\text{seqconsensus}(\ldots, \text{'PropertyName'}, \text{PropertyValue}, \ldots)
\]
defines optional properties using property name/value pairs.

\[
\text{seqconsensus}(\ldots, \text{'ScoringMatrix'}, \text{ScoringMatrixValue})
\]
specifies the scoring matrix.

The following input parameters are analogous to the function \text{seqprofile} when the alphabet is restricted to 'AA' or 'NT'.

\[
\text{seqconsensus}(\ldots, \text{'Alphabet'}, \text{AlphabetValue})
\]

\[
\text{seqconsensus}(\ldots, \text{'Gaps'}, \text{GapsValue})
\]

\[
\text{seqconsensus}(\ldots, \text{'Ambiguous'}, \text{AmbiguousValue})
\]

\[
\text{seqconsensus}(\ldots, \text{'Limits'}, \text{LimitsValue})
\]

**Examples**

\[
\text{seqs} = \text{fastaread('pf00002.fa')};
\]

\[
[C,S] = \text{seqconsensus}\left(\text{seqs}, \text{'limits'}, [50 60], \text{'gaps'}, \text{'all'}\right)
\]

**See Also**

\text{fastaread} | \text{multialignread} | \text{multialignwrite} | \text{profalign} | \text{seqdisp} | \text{seqprofile}

**Introduced before R2006a**
seqdisp

Format long sequence output for easy viewing

Syntax

seqdisp(Seq)
seqdisp(Seq, ...'Row', RowValue, ...)
seqdisp(Seq, ...'Column', ColumnValue, ...)
seqdisp(Seq, ...'ShowNumbers', ShowNumbersValue, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq</td>
<td>Nucleotide or amino acid sequence represented by any of the following:</td>
</tr>
<tr>
<td></td>
<td>• Character array</td>
</tr>
<tr>
<td></td>
<td>• String vector</td>
</tr>
<tr>
<td></td>
<td>• Character vector containing the FASTA file name</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure with the field Sequence</td>
</tr>
<tr>
<td></td>
<td>Multiply aligned sequences are allowed.</td>
</tr>
<tr>
<td></td>
<td>FASTA files can have the file extension fa, fasta, fas, fsa, or fst.</td>
</tr>
<tr>
<td>RowValue</td>
<td>Integer that specifies the length of each row. Default is 60.</td>
</tr>
<tr>
<td>ColumnValue</td>
<td>Integer that specifies the column width or number of symbols before displaying a space. Default is 10.</td>
</tr>
<tr>
<td>ShowNumbersValue</td>
<td>Controls the display of numbers at the start of each row. Choices are true (default) to show numbers, or false to hide numbers.</td>
</tr>
</tbody>
</table>

Description

seqdisp(Seq) displays a sequence in rows, with a default row length of 60 and a default column width of 10.

seqdisp(Seq, ...'PropertyName', PropertyValue, ...) calls seqdisp with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

seqdisp(Seq, ...'Row', RowValue, ...) specifies the length of each row for the displayed sequence.

seqdisp(Seq, ...'Column', ColumnValue, ...) specifies the number of letters to display before adding a space. RowValue must be larger than and evenly divisible by ColumnValue.

seqdisp(Seq, ...'ShowNumbers', ShowNumbersValue, ...) controls the display of numbers at the start of each row. Choices are true (default) to show numbers, or false to hide numbers.
Examples

Read sequence information from the GenBank database. Display the sequence in rows with 50 letters, and within a row, separate every 10 letters with a space.

```matlab
mouseHEXA = getgenbank('AK080777');
seqdisp(mouseHEXA, 'Row', 50, 'Column', 10)
```

Create and save a FASTA file with two sequences, and then display it.

```matlab
hdr = ['Sequence A'; 'Sequence B'];
seq = ['TAGCTGRCGAAGCCAAGCGAGCTTN'; 'ATCGACYGTTCCGTTGCTCAGAAN']
fastawrite('local.fa', hdr, seq);
seqdisp('local.fa', 'ShowNumbers', false')
```

```matlab
ans =
>Sequence A
 1  TAGCTGRCGAAGCCAAGCGAGCTTN
>Sequence B
 1  ATCGACYGTTCCGTTGCTCAGAAN
```

See Also

- `getgenbank`
- `multialignread`
- `multialignwrite`
- `seqconsensus`
- `seqlogo`
- `seqprofile`
- `seqshoworfs`
- `seqshowwords`
- `seqviewer`

Introduced before R2006a
seqdotplot

Create dot plot of two sequences

Syntax

seqdotplot(Seq1, Seq2)
seqdotplot(Seq1, Seq2, Window, Number)
Matches = seqdotplot(...)  
[Matches, Matrix] = seqdotplot(...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq1, Seq2</td>
<td>Nucleotide or amino acid sequences. Enter a character vector or string for each sequence. Do not enter a vector of integers. You can also enter a structure with the field Sequence.</td>
</tr>
<tr>
<td>Window</td>
<td>Enter an integer for the size of a window.</td>
</tr>
<tr>
<td>Number</td>
<td>Enter an integer for the number of characters within the window that match.</td>
</tr>
</tbody>
</table>

Description

seqdotplot(Seq1, Seq2) plots a figure that visualizes the match between two sequences.

seqdotplot(Seq1, Seq2, Window, Number) plots sequence matches when there are at least Number matches in a window of size Window.

When plotting nucleotide sequences, start with a Window of 11 and Number of 7.

Matches = seqdotplot(...) returns the number of dots in the dot plot matrix.

[Matches, Matrix] = seqdotplot(...) returns the dot plot as a sparse matrix.

Examples

This example shows the similarities between the prion protein (PrP) nucleotide sequences of two ruminants, the moufflon and the golden takin.

moufflon = getgenbank('AB060288', 'Sequence', true);
takin = getgenbank('AB060290', 'Sequence', true);
seqdotplot(moufflon, takin, 11, 7)
Note For the correct interpretation of a dot plot, your monitor's display resolution must be able to contain the sequence lengths. If the resolution is not adequate, seqdotplot resizes the image and returns a warning.

Matches = seqdotplot(moufflon,takin,11,7)
Matches =
  5552

[Matches, Matrix] = seqdotplot(moufflon,takin,11,7)

See Also
nwalign | swalign

Introduced before R2006a
seqfilter

Filter out sequences based on specified criterion

Syntax

seqfilter(fastqFile)
seqfilter(fastqFile,Name,Value)
[outFiles,nSeqIn,nSeqOut] = seqfilter( ___ )

Description

seqfilter(fastqFile) applies a filtering criterion to the sequences in fastqFile and saves the sequences that meet the criterion in a new FASTQ file. By default, the sequences that pass the criterion are saved under file names with the suffix '_filtered' appended. If you do not specify any criterion, the function filters sequences using the default.

seqfilter(fastqFile,Name,Value) uses additional options specified by one or more Name,Value pair arguments.

[outFiles,nSeqIn,nSeqOut] = seqfilter( ___ ) returns a cell array outFiles with the names of output files. nSeqIn and nSeqOut represent the numbers of sequences included and excluded from each input file, respectively.

Examples

Filter next-generation sequencing data

Filter out sequences with more than 10% of low quality bases, where a base is considered low quality when its quality score is less than 20.

[outFile,in,out] = seqfilter('SRR005164_1_50.fastq',...
'Method','MaxPercentLowQualityBases',...
'Threshold',[10 20]);

Check the number of sequences saved in the output file.

in

in = 39

Check the number of sequences filtered out.

out

out = 11

Filter out sequences having an average quality score of below 20.

[outFile,in,out] = seqfilter('SRR005164_1_50.fastq',...
'Method','MeanQuality',...
'Threshold',20);
Apply the filtering criterion to every 10 bases as a sliding window.

[outFile,in,out] = seqfilter('SRR005164_1_50.fastq',...
    'Method','MeanQuality',...
    'Threshold',20,'WindowSize',10);

Filter out sequences with less than 100 bases.

[outFile,in,out] = seqfilter('SRR005164_1_50.fastq',...
    'Method','MinLength',...
    'Threshold',100);

**Input Arguments**

fastqFile — Names of FASTQ files with sequence and quality information
character vector | string | string vector | cell array of character vectors

Names of FASTQ-formatted files with sequence and quality information, specified as a character vector, string, string vector, or cell array of character vectors.

Example: 'SRR005164_1_50.fastq'

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: 'Method','MaxNumberLowQualityBases','Threshold',[5 15]

**Method — Criterion to filter sequences**

'MaxNumberLowQualityBases' (default) | 'MaxPercentLowQualityBases' | 'MeanQuality' | 'MinLength'

Criterion to filter sequences, specified as one of the following options. Specify only one filtering criterion per function call.

- 'MaxNumberLowQualityBases'— applies a maximum threshold on the number of low-quality bases allowed.
- 'MaxPercentLowQualityBases'— applies a maximum threshold on the percentage of low-quality bases allowed.
- 'MeanQuality'— applies a minimum threshold on the average base quality across each sequence.
- 'MinLength'— applies a minimum threshold on the sequence length.

Use this name-value pair argument together with 'Threshold' to specify the appropriate threshold value. Depending on the filtering criterion, the corresponding value for 'Threshold' can be a scalar or two-element vector. See the 'Threshold' option for the default values. If you do not specify 'Threshold', then the function uses the default threshold value of the specified method. For each filtering criterion, the function uses the base quality encoding format specified by the 'Encoding' name-value pair argument.

Example: 'Method','MaxNumberLowQualityBases','Threshold',[5 15]
**Threshold** — Threshold value for filtering criterion

<table>
<thead>
<tr>
<th>'Method'</th>
<th>'Threshold'</th>
<th>Default 'Threshold' value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'MaxNumberLowQualityBases'</td>
<td>Two-element vector ([V1 \ V2]). (V1) is a nonnegative integer that specifies the maximum number of low-quality bases allowed. (V2) specifies the minimum base quality. Any base with quality less than (V2) is considered a low-quality base. Any sequence containing a number of low-quality bases greater than (V1) is filtered out and not saved in the output file.</td>
<td>[0 10]</td>
</tr>
<tr>
<td>'MaxPercentLowQualityBases'</td>
<td>Two-element vector ([V1 \ V2]). (V1) is a scalar between 0 and 100 that specifies the maximum percentage of low-quality bases allowed. (V2) specifies the minimum base quality. Any base with quality less than (V2) is considered a low-quality base. Any sequence containing a percentage of low-quality bases greater than (V1) is filtered out and not saved in the output file.</td>
<td>[0 10]</td>
</tr>
<tr>
<td>'MeanQuality'</td>
<td>Positive scalar that specifies the minimum threshold on the average base quality across each sequence. Any sequence with average base quality less than this value is filtered out.</td>
<td>0</td>
</tr>
<tr>
<td>'MinLength'</td>
<td>Nonnegative integer that specifies the minimum threshold on the sequence length allowed. Any sequence with length less than this value is filtered out.</td>
<td>1</td>
</tr>
</tbody>
</table>

Example: 'Method','MaxPercentLowQualityBases','Threshold',[10 20]

**WindowSize** — Size of sliding window to apply filtering criterion to sequence

Size of the sliding window to apply the filtering criterion to a sequence, specified as a positive integer. The size of the window corresponds to the number of bases that the function uses at one time to apply the criterion. If any window fails the criterion, the whole sequence is discarded.
The default is \texttt{Inf}, that is, the filtering criterion is applied to the whole sequence.

Example: 'WindowSize',100

**Encoding — Base quality encoding format**

'Illumina18' (default) \| 'Sanger' \| 'Solexa' \| 'Illumina13' \| 'Illumina15'

Base quality encoding format, specified as a character vector or string.

Example: 'Encoding','Sanger'

**OutputDir — Relative or absolute path to output file directory**

colorbox

Relative or absolute path to the output file directory, specified as a character vector or string. The default is the current directory.

Example: 'OutputDir','F:\results'

**OutputSuffix — Suffix to use in output file name**

'\_filtered' (default) \| character vector \| string

Suffix to use in the output file name, specified as a character vector or string. It is inserted after the input file name and before the file extension. The default is '\_filtered'.

Example: 'OutputSuffix','\_WindowSize100\_filtered'

**PairedFiles — Whether to consider input files as pairs for paired-end sequence data**

false (default) \| true

Whether to consider the input files as pairs for paired-end sequence data, specified as true or false.

If true, the input files are read as pairs, and the sequence data is maintained in sync between the files. That is, if a sequence is filtered out in the first file, the corresponding sequence in the paired file is also filtered out.

Example: 'PairedFiles',true

**WriteSingleton — Whether to save singleton sequences in a separate output file**

false (default) \| true

Whether to save singleton sequences in a separate output file, specified as true or false. To set this to true, the 'PairedFiles' option must also be set to true.

A singleton sequence is the sequence that pass the filtering criterion but its corresponding sequence in the paired file does not. If true, singleton sequences are saved in a separate file with the suffix '\_singleton'. The default is false, meaning that, only sequences that pass the filtering criterion in both input files of a given pair are saved in the output files.

Example: 'PairedFiles',true,'WriteSingleton',true

**UseParallel — Boolean indicating whether to perform computation in parallel**

false (default) \| true

Boolean indicating whether to perform computation in parallel, specified as true or false.
For parallel computing, you must have Parallel Computing Toolbox. If a parallel pool does not exist, one is created automatically when the auto-creation option is enabled in your parallel preferences. Otherwise, computation runs in serial mode.

**Note** There is a cost associated with sharing large input files across workers in a distributed environment. In some cases, running in parallel may not be beneficial in terms of performance.

Example: `'UseParallel',true`

**Output Arguments**

- **outFiles** — Output file names
  
  cell array of character vectors
  
  Output file names, returned as a cell array of character vectors.

- **nSeqIn** — Number of sequences selected from each input file
  
  scalar | vector
  
  Number of sequences selected from each input file, returned as a scalar or an \( n \)-by-1 vector where \( n \) is the number of input files. If there are multiple input files, the order within `nSeqIn` corresponds to the order of the input files.

- **nSeqOut** — Number of sequences excluded from each input file
  
  scalar | vector
  
  Number of sequences excluded from each input file, returned as a scalar or an \( n \)-by-1 vector where \( n \) is the number of input files. If there are multiple input files, the order within `nSeqOut` corresponds to the order of the input files.

**Extended Capabilities**

**Automatic Parallel Support**

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true`.

For more information, see the `'UseParallel'` name-value pair argument.

**See Also**

- `seqsplit`
- `seqsplitpe`
- `seqtrim`

*Introduced in R2016b*
**seqinsertgaps**

Insert gaps into nucleotide or amino acid sequence

**Syntax**

```plaintext
NewSeq = seqinsertgaps(Seq, Positions)
NewSeq = seqinsertgaps(Seq, GappedSeq)
NewSeq = seqinsertgaps(Seq, GappedSeq, Relationship)
```

**Input Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| Seq        | Either of the following:  
|            | • Character vector or string specifying a nucleotide or amino acid sequence  
|            | • MATLAB structure containing a Sequence field |
| Positions  | Vector of integers to specify the positions in Seq before which to insert a gap. |
| GappedSeq  | Either of the following:  
|            | • Character vector or string specifying a nucleotide or amino acid sequence  
|            | • MATLAB structure containing a Sequence field |
| Relationship | Integer specifying the relationship between Seq and GappedSeq. Choices are:  
|            | • 1 — Both sequences use the same alphabet, that is both are nucleotide sequences or both are amino acid sequences.  
|            | • 3 — Seq contains nucleotides representing codons and GappedSeq contains amino acids (default). |

**Output Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewSeq</td>
<td>Sequence with gaps inserted, represented by a character vector specifying a nucleotide or amino acid sequence.</td>
</tr>
</tbody>
</table>

**Description**

*NewSeq = seqinsertgaps(Seq, Positions)* inserts gaps in the sequence *Seq* before the positions specified by the integers in the vector *Positions*.

*NewSeq = seqinsertgaps(Seq, GappedSeq)* finds the gap positions in the sequence *GappedSeq*, then inserts gaps in the corresponding positions in the sequence *Seq*.

*NewSeq = seqinsertgaps(Seq, GappedSeq, Relationship)* specifies the relationship between *Seq* and *GappedSeq*. Enter 1 for *Relationship* when both sequences use the same alphabet, that is both are nucleotide sequences or both are amino acid sequences. Enter 3 for *Relationship* when *Seq* contains nucleotides representing codons and *GappedSeq* contains amino acids. Default is 3.
Examples

1. Retrieve two nucleotide sequences from the GenBank database for the neuraminidase (NA) protein of two strains of the Influenza A virus (H5N1).

   ```matlab
   hk01 = getgenbank('AF509094');
   vt04 = getgenbank('DQ094287');
   ```

2. Extract the coding region from the two nucleotide sequences.

   ```matlab
   hk01_cds = featureparse(hk01,'feature','CDS','Sequence',true);
   vt04_cds = featureparse(vt04,'feature','CDS','Sequence',true);
   ```

3. Align the amino acids sequences converted from the nucleotide sequences.

   ```matlab
   [sc,al]=nwalign(nt2aa(hk01_cds),nt2aa(vt04_cds),'extendgap',1);
   ```

4. Use the seqinsertgaps function to copy the gaps from the aligned amino acid sequences to their corresponding nucleotide sequences, thus codon-aligning them.

   ```matlab
   hk01_aligned = seqinsertgaps(hk01_cds,al(1,:))
   vt04_aligned = seqinsertgaps(vt04_cds,al(3,:))
   ```

5. Once you have code aligned the two sequences, you can use them as input to other functions such as dnds, which calculates the synonymous and nonsynonymous substitutions rates of the codon-aligned nucleotide sequences. By settingVerbose to true, you can also display the codons considered in the computations and their amino acid translations.

   ```matlab
   [dn,ds] = dnds(hk01_aligned,vt04_aligned,'verbose',true)
   ```

See Also

dnds | dndsml | featureparse | int2aa | int2nt

Introduced in R2007a
seqlinkage

Construct phylogenetic tree from pairwise distances

Syntax

PhyloTree = seqlinkage(Distances)
PhyloTree = seqlinkage(Distances, Method)
PhyloTree = seqlinkage(Distances, Method, Names)

Arguments

<table>
<thead>
<tr>
<th>Distances</th>
<th>Matrix or vector of pairwise distances, such as returned by the seqpdist function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>Character vector or string that specifies a distance method. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'single'</td>
</tr>
<tr>
<td></td>
<td>• 'complete'</td>
</tr>
<tr>
<td></td>
<td>• 'average' (default)</td>
</tr>
<tr>
<td></td>
<td>• 'weighted'</td>
</tr>
<tr>
<td></td>
<td>• 'centroid'</td>
</tr>
<tr>
<td></td>
<td>• 'median'</td>
</tr>
<tr>
<td>Names</td>
<td>Specifies alternative labels for leaf nodes. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• Vector of structures, each with a Header or Name field</td>
</tr>
<tr>
<td></td>
<td>• Cell array of character vectors or string vector</td>
</tr>
<tr>
<td></td>
<td>The elements must be unique. The number of elements must comply with the number of samples used to generate the pairwise distances in Dist.</td>
</tr>
</tbody>
</table>

Description

PhyloTree = seqlinkage(Distances) returns a phylogenetic tree object from the pairwise distances, Distances, between the species or products. Distances is a matrix or vector of pairwise distances, such as returned by the seqpdist function.

PhyloTree = seqlinkage(Distances, Method) creates a phylogenetic tree object using a specified patristic distance method. The available methods are:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'single'</td>
<td>Nearest distance (single linkage method)</td>
</tr>
<tr>
<td>'complete'</td>
<td>Furthest distance (complete linkage method)</td>
</tr>
<tr>
<td>'average' (default)</td>
<td>Unweighted Pair Group Method Average (UPGMA, group average).</td>
</tr>
<tr>
<td>'weighted'</td>
<td>Weighted Pair Group Method Average (WPGMA)</td>
</tr>
<tr>
<td>'centroid'</td>
<td>Unweighted Pair Group Method Centroid (UPGMC)</td>
</tr>
</tbody>
</table>
PhyloTree = seqlinkage(Distances, Method, Names) passes a list of unique names to label the leaf nodes (for example, species or products) in a phylogenetic tree object.

### Examples

**Build Phylogenetic Tree from Pairwise Distances**

Create an array of structures representing a multiple alignment of amino acids:

```matlab
seqs = fastaread('pf00002.fa');
```

Measure the Jukes-Cantor pairwise distances between sequences:

```matlab
distances = seqpdist(seqs,'method','jukes-cantor','indels','pair');
```

Build the phylogenetic tree for the multiple sequence alignment from calculated pairwise distances. Specify the method to compute the distances of the new nodes to all other nodes. Provide leaf names:

```matlab
phylotree = seqlinkage(distances,'single',seqs)
```

View the phylogenetic tree:

```matlab
view(phylotree)
```
Compatibility Considerations

`seqlinkage` correctly computes the input pairwise distances

*Behavior changed in R2017b*

For the R2017a or earlier versions, `seqlinkage` incorrectly doubled the input pairwise distances when building a tree. This bug has been fixed in R2017b.

If you have been previously selecting a subset of the tree returned by `seqlinkage` with a distance threshold, consider dividing the threshold by 2.

Note that the tree topology has always been computed correctly and not affected by this bug.
See Also
cluster | phytree | phytreewrite | plot | seqneighjoin | seqpdist | view

Introduced before R2006a
seqlogo

Display sequence logo for nucleotide or amino acid sequences

Syntax

seqlogo(Seqs)
seqlogo(Profile)
WgtMatrix = seqlogo(...)
[WgtMatrix, Handle] = seqlogo(...)

seqlogo(..., 'Displaylogo', DisplaylogoValue, ...)
seqlogo(..., 'Alphabet', AlphabetValue, ...)
seqlogo(..., 'Startat', StartatValue, ...)
seqlogo(..., 'Endat', EndatValue, ...)
seqlogo(..., 'SSCorrection', SSCorrectionValue, ...)

Input Arguments

<table>
<thead>
<tr>
<th>Seqs</th>
<th>Set of pairwise or multiply aligned nucleotide or amino acid sequences, represented by any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character array</td>
</tr>
<tr>
<td></td>
<td>• Cell array of character vectors</td>
</tr>
<tr>
<td></td>
<td>• String vector</td>
</tr>
<tr>
<td></td>
<td>• Array of structures containing a Sequence field</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Profile</th>
<th>Sequence profile distribution matrix with the frequency of nucleotides or amino acids for every column in the multiple alignment, such as returned by the seqprofile function.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The size of the frequency distribution matrix is:</td>
</tr>
<tr>
<td></td>
<td>• For nucleotides — [4 x sequence length]</td>
</tr>
<tr>
<td></td>
<td>• For amino acids — [20 x sequence length]</td>
</tr>
<tr>
<td></td>
<td>If gaps were included, Profile may have 5 rows (for nucleotides) or 21 rows (for amino acids), but seqlogo ignores gaps.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DisplaylogoValue</th>
<th>Controls the display of a sequence logo. Choices are true (default) or false.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>AlphabetValue</th>
<th>Character vector or string specifying the type of sequence (nucleotide or amino acid). Choices are 'NT' (default) or 'AA'.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>StartatValue</th>
<th>Positive integer that specifies the starting position for the sequences in Seqs. Default starting position is 1.</th>
</tr>
</thead>
</table>

| EndatValue | Positive integer that specifies the ending position for the sequences in Seqs. Default ending position is the maximum length of the sequences in Seqs. |
**SSCorrectionValue**

Controls the use of small sample correction in the estimation of the number of bits. Choices are `true` (default) or `false`.

---

**Output Arguments**

<table>
<thead>
<tr>
<th>WgtMatrix</th>
<th>Cell array containing the symbol list in <code>Seqs</code> or <code>Profile</code> and the weight matrix used to graphically display the sequence logo.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handle</td>
<td>Handle to the sequence logo figure.</td>
</tr>
</tbody>
</table>

**Description**

`seqlogo(Seqs)` displays a sequence logo for `Seqs`, a set of aligned sequences. The logo graphically displays the sequence conservation at a particular position in the alignment of sequences, measured in bits. The maximum sequence conservation per site is \( \log_2(4) \) bits for nucleotide sequences and \( \log_2(20) \) bits for amino acid sequences. If the sequence conservation value is zero or negative, no logo is displayed in that position.

`seqlogo(Profile)` displays a sequence logo for `Profile`, a sequence profile distribution matrix with the frequency of nucleotides or amino acids for every column in the multiple alignment, such as returned by the `seqprofile` function.

**Color Code for Nucleotides**

<table>
<thead>
<tr>
<th>Nucleotide</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Green</td>
</tr>
<tr>
<td>C</td>
<td>Blue</td>
</tr>
<tr>
<td>G</td>
<td>Yellow</td>
</tr>
<tr>
<td>T, U</td>
<td>Red</td>
</tr>
<tr>
<td>Other</td>
<td>Purple</td>
</tr>
</tbody>
</table>

**Color Code for Amino Acids**

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>Chemical Property</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>G S T Y C Q N</td>
<td>Polar</td>
<td>Green</td>
</tr>
<tr>
<td>A V L I P W F M</td>
<td>Hydrophobic</td>
<td>Orange</td>
</tr>
<tr>
<td>D E</td>
<td>Acidic</td>
<td>Red</td>
</tr>
<tr>
<td>K R H</td>
<td>Basic</td>
<td>Blue</td>
</tr>
<tr>
<td>Other</td>
<td>—</td>
<td>Tan</td>
</tr>
</tbody>
</table>

\( WgtMatrix = \text{seqlogo}(\ldots) \) returns a cell array of unique symbols in the sequence `Seqs` or `Profile`, and the information weight matrix used to graphically display the logo.

\([WgtMatrix, \text{Handle}] = \text{seqlogo}(\ldots) \) returns a handle to the sequence logo figure.

`seqlogo(Seqs, ...'PropertyName', PropertyValue, ...)` calls `seqpdist` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:
seqlogo(..., 'Displaylogo', DisplaylogoValue, ...) controls the display of a sequence logo. Choices are true (default) or false.

seqlogo(..., 'Alphabet', AlphabetValue, ...) specifies the type of sequence (nucleotide or amino acid). Choices are 'NT' (default) or 'AA'.

**Note** If you provide amino acid sequences to seqlogo, you must set Alphabet to 'AA'.

seqlogo(..., 'Startat', StartatValue, ...) specifies the starting position for the sequences in Seqs. Default starting position is 1.

seqlogo(..., 'Endat', EndatValue, ...) specifies the ending position for the sequences in Seqs. Default ending position is the maximum length of the sequences in Seqs.

seqlogo(..., 'SSCorrection', SSCorrectionValue, ...) controls the use of small sample correction in the estimation of the number of bits. Choices are true (default) or false.

**Note** A simple calculation of bits tends to overestimate the conservation at a particular location. To compensate for this overestimation, when SSCorrection is set to true, a rough estimate is applied as an approximate correction. This correction works better when the number of sequences is greater than 50.

### Examples

**Display a Sequence Logo for Aligned Nucleotide Sequences**

This example shows how to display a sequence logo for a set of aligned nucleotide sequences.

Create a series of aligned nucleotide sequences.

```plaintext
S = {'ATTATAGCAAACTA', 'AACATGCCAAAGTA', 'ATCATGCAAAAGGA'}
```

Display the sequence logo.

```plaintext
seqlogo(S)
```
Display a Sequence Logo for Aligned Amino Acid Sequences

This example shows how to display a sequence logo for a set of aligned amino acid sequences.

Create a series of aligned amino acid sequences.

```matlab
S2 = {'LSGGQRQRVAIARALAL',
      'LSGGEKQRVAIARALMN',
      'LSGGQIQRVLLARALAA',
      'LSGGERRRLEIACVLAL',
      'FSGGEKKKNELWQMLAL',
      'LSGGERRRLEIACVLAL'};
```

Display the sequence logo, specifying an amino acid sequence and limiting the logo to sequence positions 2 through 10.

```matlab
seqlogo(S2, 'alphabet', 'aa', 'startAt', 2, 'endAt', 10)
```
References


See Also
seqconsensus | seqdisp | seqprofile

Introduced before R2006a
seqmatch

Find matches for every character vector in library

Syntax

Index = seqmatch(keyword, Library)

Description

Index = seqmatch(keyword, Library) looks through Library to find character vectors or string that begin with keyword. Index contains the index to the first occurrence for every character vector or string in the query. keyword and Library must be cell arrays of character vectors or string vector.

Examples

lib = {'VIPS_HUMAN', 'SCCR_RABIT', 'CALR_PIG', 'VIPR_RAT', 'PACR_MOUSE'};
query = {'CALR', 'VIP'};
h = seqmatch(query, lib);
lib(h)

ans =
    'CALR_PIG'    'VIPS_HUMAN'

See Also

regexp | strncmp

Introduced before R2006a
**seqneighjoin**

Construct phylogenetic tree using neighbor-joining method

**Syntax**

PhyloTree = seqneighjoin(Distances)
PhyloTree = seqneighjoin(Distances, Method)
PhyloTree = seqneighjoin(Distances, Method, Names)
PhyloTree = seqneighjoin(..., 'Reroot', RerootValue)

**Input Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distances</td>
<td>Matrix or vector containing biological distances between pairs of sequences, such as returned by the seqpdist function.</td>
</tr>
<tr>
<td>Method</td>
<td>Character vector or string specifying a method to compute the distances between nodes. Choices are 'equivar' (default) or 'firstorder'.</td>
</tr>
</tbody>
</table>
| Names          | Either of the following:  
|                | • Vector of structures with the fields Header and Name  
|                | • Cell array of character vectors or string vector  
|                | The number of elements must equal the number of samples used to generate the pairwise distances in Distances. |

**Description**

PhyloTree = seqneighjoin(Distances) computes PhyloTree, a phylogenetic tree object, from Distances, pairwise distances between the species or products, using the neighbor-joining method.

PhyloTree = seqneighjoin(Distances, Method) specifies Method, a method to compute the distances of the new nodes to all other nodes at every iteration. The general expression to calculate the distances between the new node, n, after joining i and j and all other nodes (k), is given by

\[ D(n,k) = aD(i,k) + (1-a)D(j,k) - aD(n,i) - (1-a)D(n,j) \]

This expression is guaranteed to find the correct tree with additive data (minimum variance reduction).

Choices for Method are:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equivar (default)</td>
<td>Assumes equal variance and independence of evolutionary distance estimates (a = 1/2), such as in the original neighbor-joining algorithm by Saitou and Nei, JMBE (1987) or as in Studier and Keppler, JMBE (1988).</td>
</tr>
</tbody>
</table>
### Method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstorder</td>
<td>Assumes a first-order model of the variances and covariances of evolutionary distance estimates, with 'a' being adjusted at every iteration to a value between 0 and 1, such as in Gascuel, JMBE (1997).</td>
</tr>
</tbody>
</table>

**PhyloTree** = seqneighjoin(Distances, Method, Names) passes Names, a list of names (such as species or products), to label the leaf nodes in the phylogenetic tree object.

**PhyloTree** = seqneighjoin(..., ‘Reroot’, RerootValue) specifies whether to reroot PhyloTree. Choices are true (default) or false. When RerootValue is false, seqneighjoin excludes rerooting the resulting tree, which is useful for observing the original linkage order followed by the algorithm. By default seqneighjoin reroots the resulting tree using the midpoint method.

### Examples

**Build Phylogenetic Tree using Neighbor Joining Method**

Create an array of structures representing a multiple alignment of amino acids:

```matlab
seqs = fastaread('pf00002.fa');
```

Measure the Jukes-Cantor pairwise distances between sequences.

```matlab
distances = seqpdist(seqs,'method','jukes-cantor','indels','pair');
```

Use the output argument distances, a vector containing biological distances between each pair of sequences, as an input argument to seqneighjoin.

Build the phylogenetic tree for the multiple sequence alignment using the neighbor-joining algorithm. Specify the method to compute the distances of the new nodes to all other nodes.

```matlab
phylotree = seqneighjoin(distances,'equivar',seqs)
```

Phylogenetic tree object with 32 leaves (31 branches)

View the phylogenetic tree:

```matlab
view(phylotree)
```
References


See Also

cluster | multialign | phytree | plot | reroot | seqlinkage | seqpdist | view
Introduced before R2006a
seqpdist

Calculate pairwise distance between sequences

Syntax

\[ D = \text{seqpdist}(\text{Seqs}) \]

\[ D = \text{seqpdist}(\text{Seqs}, ... \text{'PropertyName'}, \text{PropertyValue}, ...) \]

\[ D = \text{seqpdist}(\text{Seqs}, ... \text{'Method'}, \text{MethodValue}, ...) \]

\[ D = \text{seqpdist}(\text{Seqs}, ... \text{'Indels'}, \text{IndelsValue}, ...) \]

\[ D = \text{seqpdist}(\text{Seqs}, ... \text{'OptArgs'}, \text{OptArgsValue}, ...) \]

\[ D = \text{seqpdist}(\text{Seqs}, ... \text{'PairwiseAlignment'}, \text{PairwiseAlignmentValue}, ...) \]

\[ D = \text{seqpdist}(\text{Seqs}, ... \text{'UseParallel'}, \text{UseParallelValue}, ...) \]

\[ D = \text{seqpdist}(\text{Seqs}, ... \text{'SquareForm'}, \text{SquareFormValue} ...) \]

\[ D = \text{seqpdist}(\text{Seqs}, ... \text{'Alphabet'}, \text{AlphabetValue}, ...) \]

\[ D = \text{seqpdist}(\text{Seqs}, ... \text{'ScoringMatrix'}, \text{ScoringMatrixValue}, ...) \]

\[ D = \text{seqpdist}(\text{Seqs}, ... \text{'Scale'}, \text{ScaleValue}, ...) \]

\[ D = \text{seqpdist}(\text{Seqs}, ... \text{'GapOpen'}, \text{GapOpenValue}, ...) \]

\[ D = \text{seqpdist}(\text{Seqs}, ... \text{'ExtendGap'}, \text{ExtendGapValue}, ...) \]

Input Arguments

<table>
<thead>
<tr>
<th>Seqs</th>
<th>Any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Cell array of character vectors or string vector containing nucleotide or amino acid</td>
</tr>
<tr>
<td></td>
<td>sequences</td>
</tr>
<tr>
<td></td>
<td>• Vector of structures containing a Sequence field</td>
</tr>
<tr>
<td></td>
<td>• Matrix of characters, in which each row corresponds to a nucleotide or amino acid</td>
</tr>
<tr>
<td>MethodValue</td>
<td>Character vector or string that specifies the method to calculate pairwise distances.</td>
</tr>
<tr>
<td></td>
<td>Default is 'Jukes-Cantor'.</td>
</tr>
<tr>
<td>IndelsValue</td>
<td>Character vector or string that specifies how to treat sites with gaps. Default is</td>
</tr>
<tr>
<td></td>
<td>'score'.</td>
</tr>
<tr>
<td>OptArgsValue</td>
<td>Character vector or cell array that specifies one or more input arguments required or</td>
</tr>
<tr>
<td></td>
<td>accepted by the distance method specified by the Method property.</td>
</tr>
</tbody>
</table>
| **PairwiseAlignmentValue** | Controls the global pairwise alignment of input sequences (using the nwalign function), while ignoring the multiple alignment of the input sequences (if any). Choices are true or false. Default is:
  - true — When all input sequences do not have the same length.
  - false — When all input sequences have the same length.

**Tip** If your input sequences are the same length, seqpdist assumes they are aligned. If they are not aligned, do one of the following:
  - Align the sequences before passing them to seqpdist, for example, using the multialign function.
  - Set PairwiseAlignment to true when using seqpdist.

| **UseParallelValue** | Controls the calculation of the pairwise distances using parfor-loops. When true, and Parallel Computing Toolbox is installed and a parpool is open, computation occurs in parallel. If there are no open parpool, but automatic creation is enabled in the Parallel Preferences, the default pool will be automatically open and computation occurs in parallel. If Parallel Computing Toolbox is installed, but there are no open parpool and automatic creation is disabled, then computation uses parfor-loops in serial mode. If Parallel Computing Toolbox is not installed, then computation uses parfor-loops in serial mode. Default is false, which uses for-loops in serial mode.

| **SquareFormValue** | Controls the conversion of the output into a square matrix. Choices are true or false (default).

<p>| <strong>AlphabetValue</strong> | Character vector or string specifying the type of sequence (nucleotide or amino acid). Choices are 'NT' or 'AA' (default). |</p>
<table>
<thead>
<tr>
<th><strong>ScoringMatrixValue</strong></th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Character vector or string specifying the scoring matrix to use for the alignment. Choices for amino acid sequences are:</td>
</tr>
<tr>
<td></td>
<td>- 'BLOSUM62'</td>
</tr>
<tr>
<td></td>
<td>- 'BLOSUM30' increasing by 5 up to 'BLOSUM90'</td>
</tr>
<tr>
<td></td>
<td>- 'BLOSUM100'</td>
</tr>
<tr>
<td></td>
<td>- 'PAM10' increasing by 10 up to 'PAM500'</td>
</tr>
<tr>
<td></td>
<td>- 'DAYHOFF'</td>
</tr>
<tr>
<td></td>
<td>- 'GONNET'</td>
</tr>
<tr>
<td>Default is:</td>
<td>- 'BLOSUM50' — When <strong>AlphabetValue</strong> equals 'AA'</td>
</tr>
<tr>
<td></td>
<td>- 'NUC44' — When <strong>AlphabetValue</strong> equals 'NT'</td>
</tr>
</tbody>
</table>

**Note** The above scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the 'Scale' property to specify an additional scale factor to convert the output score from bits to another unit.

- Matrix representing the scoring matrix to use for the alignment, such as returned by the `blosum`, `pam`, `dayhoff`, `gonnet`, or `nuc44` function.

**Note** If you use a scoring matrix that you created or was created by one of the above functions, the matrix does not include a scale factor. The output score will be returned in the same units as the scoring matrix. You can use the 'Scale' property to specify a scale factor to convert the output score to another unit.

**Note** If you need to compile `seqpdist` into a stand-alone application or software component using MATLAB Compiler, use a matrix instead of a character vector or string for `ScoringMatrixValue`.

<table>
<thead>
<tr>
<th><strong>ScaleValue</strong></th>
<th>Positive value that specifies the scale factor used to return the score in arbitrary units. If the scoring matrix information also provides a scale factor, then both are used.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>GapOpenValue</strong></th>
<th>Positive integer that specifies the penalty for opening a gap in the alignment. Default is 8.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>ExtendedGapValue</strong></th>
<th>Positive integer that specifies the penalty for extending a gap. Default is equal to <strong>GapOpenValue</strong>.</th>
</tr>
</thead>
</table>
Output Arguments

| D        | Vector that contains biological distances between each pair of sequences stored in the M elements of Seqs. |

Description

\( D = \text{seqpdist}(\text{Seqs}) \) returns \( D \), a vector containing biological distances between each pair of sequences stored in the \( M \) sequences of \( \text{Seqs} \), a cell array of sequences, a vector of structures, or a matrix or sequences.

\( D \) is a \( 1 \)-by-\((M \times (M-1)/2) \) row vector corresponding to the \( M \times (M-1)/2 \) pairs of sequences in \( \text{Seqs} \). The output \( D \) is arranged in the order \(((2,1),(3,1),\ldots,(M,1),(3,2),\ldots,(M,2),\ldots,(M,M-1))\). This is the lower-left triangle of the full \( M \)-by-\( M \) distance matrix. To get the distance between the \( I \)th and the \( J \)th sequences for \( I > J \), use the formula \( D((J-1)\times(M-J/2)+I-J) \).

\( D = \text{seqpdist}(\text{Seqs}, ...'\text{PropertyName}', \text{PropertyValue}, ...) \) calls seqpdist with optional properties that use property name/property value pairs. Specify one or more properties in any order. Enclose each \( \text{PropertyName} \) in single quotation marks. Each \( \text{PropertyName} \) is case insensitive. These property name/property value pairs are as follows:

\( D = \text{seqpdist}(\text{Seqs}, ...'\text{Method}', \text{MethodValue}, ...) \) specifies a method to compute distances between each sequence pair. Choices are shown in the following tables.
### Methods for Nucleotides and Amino Acids

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>p-distance</strong></td>
<td>Proportion of sites at which the two sequences are different. $p$ is close to 1 for poorly related sequences, and $p$ is close to 0 for similar sequences.</td>
</tr>
<tr>
<td></td>
<td>$d = p$</td>
</tr>
<tr>
<td><strong>Jukes-Cantor (default)</strong></td>
<td>Maximum likelihood estimate of the number of substitutions between two sequences. $p$ is described with the method p-distance.</td>
</tr>
<tr>
<td></td>
<td>For nucleotides: $d = -3/4 \log(1-p \times 4/3)$</td>
</tr>
<tr>
<td></td>
<td>For amino acids: $d = -19/20 \log(1-p \times 20/19)$</td>
</tr>
<tr>
<td><strong>alignment-score</strong></td>
<td>Distance ($d$) between two sequences ($1, 2$) is computed from the pairwise alignment score between the two sequences ($\text{score12}$), and the pairwise alignment score between each sequence and itself ($\text{score11, score22}$) as follows: $d = (1-\text{score12}/\text{score11}) \times (1-\text{score12}/\text{score22})$</td>
</tr>
<tr>
<td></td>
<td>This option does not imply that prealigned input sequences will be realigned, it only scores them. Use with care; this distance method does not comply with the ultrametric condition. In the rare case where the score between sequences is greater than the score when aligning a sequence with itself, then $d = 0$</td>
</tr>
</tbody>
</table>

### Methods with No Scoring of Gaps (Nucleotides Only)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tajima-Nei</strong></td>
<td>Maximum likelihood estimate considering the background nucleotide frequencies. It can be computed from the input sequences or given by setting $\text{OptArgs}$ to $[gA \ gC \ gG \ gT]$. $gA, gC, gG, gT$ are scalar values for the nucleotide frequencies.</td>
</tr>
<tr>
<td><strong>Kimura</strong></td>
<td>Considers separately the transitional nucleotide substitution and the transversional nucleotide substitution.</td>
</tr>
<tr>
<td><strong>Tamura</strong></td>
<td>Considers separately the transitional nucleotide substitution, the transversional nucleotide substitution, and the GC content. GC content can be computed from the input sequences or given by setting $\text{OptArgs}$ to the proportion of GC content (scalar value from 0 to 1).</td>
</tr>
<tr>
<td><strong>Hasegawa</strong></td>
<td>Considers separately the transitional nucleotide substitution, the transversional nucleotide substitution, and the background nucleotide frequencies. Background frequencies can be computed from the input sequences or given by setting the $\text{OptArgs}$ property to $[gA \ gC \ gG \ gT]$.</td>
</tr>
<tr>
<td><strong>Nei-Tamura</strong></td>
<td>Considers separately the transitional nucleotide substitution between purines, the transitional nucleotide substitution between pyrimidines, the transversional nucleotide substitution, and the background nucleotide frequencies. Background frequencies can be computed from the input sequences or given by setting the $\text{OptArgs}$ property to $[gA \ gC \ gG \ gT]$.</td>
</tr>
</tbody>
</table>
Methods with No Scoring of Gaps (Amino Acids Only)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poisson</td>
<td>Assumes that the number of amino acid substitutions at each site has a Poisson distribution.</td>
</tr>
<tr>
<td>Gamma</td>
<td>Assumes that the number of amino acid substitutions at each site has a Gamma distribution with parameter $a$. Set $a$ using the OptArgs property. Default is 2.</td>
</tr>
</tbody>
</table>

You can also specify a user-defined distance function using @, for example, @distfun. The distance function must have the form:

```matlab
function D = distfun(S1, S2, OptArgsValue)
```

The `distfun` function takes the following arguments:

- $S1, S2$ — Two sequences of the same length (nucleotide or amino acid).
- `OptArgsValue` — Optional problem-dependent arguments.

The `distfun` function returns a scalar that represents the distance between $S1$ and $S2$.

$D = seqpdist(Seqs, ...'Indels', IndelsValue, ...)$ specifies how to treat sites with gaps. Choices are:

- `score` (default) — Scores these sites either as a point mutation or with the alignment parameters, depending on the method selected.
- `pairwise-del` — For every pairwise comparison, it ignores the sites with gaps.
- `complete-del` — Ignores all the columns in the multiple alignment that contain a gap. This option is available only if you provided a multiple alignment as the input `Seqs`.

$D = seqpdist(Seqs, ...'OptArgs', OptArgsValue, ...)$ passes one or more arguments required or accepted by the distance method specified by the Method property. Use a character vector or cell array to pass one or more input arguments. For example, provide the nucleotide frequencies for the Tajima-Nei distance method, instead of computing them from the input sequences.

$D = seqpdist(Seqs, ...'PairwiseAlignment', PairwiseAlignmentValue, ...)$ controls the global pairwise alignment of input sequences (using the `nwalign` function), while ignoring the multiple alignment of the input sequences (if any). Default is:

- `true` — When all input sequences do not have the same length.
- `false` — When all input sequences have the same length.

**Tip** If your input sequences have the same length, `seqpdist` assumes they are aligned. If they are not aligned, do one of the following:

- Align the sequences before passing them to `seqpdist`, for example, using the `multialign` function.
- Set `PairwiseAlignment` to `true` when using `seqpdist`.

$D = seqpdist(Seqs, ...'UseParallel', UseParallelValue, ...)$ specifies whether to use `parfor`-loops when calculating the pairwise distances. When `true`, and Parallel Computing
Toolbox is installed and a parpool is open, computation occurs in parallel. If there are no open parpool, but automatic creation is enabled in the Parallel Preferences, the default pool will be automatically open and computation occurs in parallel. If Parallel Computing Toolbox is installed, but there are no open parpool and automatic creation is disabled, then computation uses parfor-loops in serial mode. If Parallel Computing Toolbox is not installed, then computation uses parfor-loops in serial mode. Default is false, which uses for-loops in serial mode.

\[ D = \text{seqpdist}(\text{Seqs}, \ldots \text{'SquareForm'}, \text{SquareFormValue} \ldots) \]

controls the conversion of the output into a square matrix such that \( D(I,J) \) denotes the distance between the \( I \)th and \( J \)th sequences. The square matrix is symmetric and has a zero diagonal. Choices are true or false (default). Setting Squareform to true is the same as using the squareform function in Statistics and Machine Learning Toolbox.

\[ D = \text{seqpdist}(\text{Seqs}, \ldots \text{'Alphabet'}, \text{AlphabetValue}, \ldots) \]

specifies the type of sequence (nucleotide or amino acid). Choices are 'NT' or 'AA' (default).

The remaining input properties are available when the Method property equals 'alignment-score' or the PairwiseAlignment property equals true.

\[ D = \text{seqpdist}(\text{Seqs}, \ldots \text{'ScoringMatrix'}, \text{ScoringMatrixValue}, \ldots) \]

specifies the scoring matrix to use for the global pairwise alignment. Default is:
- 'NUC44' — When AlphabetValue equals 'NT'.
- 'BLOSUM50' — When AlphabetValue equals 'AA'.

\[ D = \text{seqpdist}(\text{Seqs}, \ldots \text{'Scale'}, \text{ScaleValue}, \ldots) \]

specifies the scale factor used to return the score in arbitrary units. Choices are any positive value. If the scoring matrix information also provides a scale factor, then both are used.

\[ D = \text{seqpdist}(\text{Seqs}, \ldots \text{'GapOpen'}, \text{GapOpenValue}, \ldots) \]

specifies the penalty for opening a gap in the alignment. Choices are any positive integer. Default is 8.

\[ D = \text{seqpdist}(\text{Seqs}, \ldots \text{'ExtendGap'}, \text{ExtendGapValue}, \ldots) \]

specifies the penalty for extending a gap in the alignment. Choices are any positive integer. Default is equal to GapOpenValue.

**Examples**

1. Read amino acid alignment data into a MATLAB structure.

   ```matlab
   seqs = fastaread('pf00002.fa');
   ```

2. For every possible pair of sequences in the multiple alignment, ignore sites with gaps and score with the scoring matrix PAM250.

   ```matlab
   dist = seqpdist(seqs,'Method','alignment-score',...
                   'Indels','pairwise-delete',...
                   'ScoringMatrix','pam250');
   ```

3. Force the realignment of each sequence pair ignoring the provided multiple alignment.

   ```matlab
   dist = seqpdist(seqs,'Method','alignment-score',...
                   'Indels','pairwise-delete',...
                   'ScoringMatrix','pam250',...
                   'PairwiseAlignment',true);
   ```
Measure the Jukes-Cantor pairwise distances after realigning each sequence pair, counting the
gaps as point mutations.

```matlab
dist = seqpdist(seqs,'Method','jukes-cantor',...  
    'Indels','score',...  
    'Scoringmatrix','pam250',...  
    'PairwiseAlignment',true);
```

**Extended Capabilities**

**Automatic Parallel Support**
Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true`.

For more information, see the `'UseParallel'` name-value pair argument.

**See Also**
`dnds | dndsml | fastaread | multialign | nalign | pdist | phytree | seqlinkage`

**Topics**
`phytree object`

**Introduced before R2006a**
seqprofile

Calculate sequence profile from set of multiply aligned sequences

Syntax

Profile = seqprofile(Seqs)
[Profile, Symbols] = seqprofile(Seqs)

seqprofile(Seqs, ...'Alphabet', AlphabetValue, ...)
seqprofile(Seqs, ...'Counts', CountsValue, ...)
seqprofile(Seqs, ...'Gaps', GapsValue, ...)
seqprofile(Seqs, ...'Ambiguous', AmbiguousValue, ...)
seqprofile(Seqs, ...'Limits', LimitsValue, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seqs</td>
<td>Set of multiply aligned sequences represented by any of the following:</td>
</tr>
<tr>
<td></td>
<td>• Character array</td>
</tr>
<tr>
<td></td>
<td>• Cell array of character vectors</td>
</tr>
<tr>
<td></td>
<td>• String vector</td>
</tr>
<tr>
<td></td>
<td>• Array of structures containing the field Sequence</td>
</tr>
<tr>
<td>AlphabetValue</td>
<td>Character vector or string specifying the sequence alphabet. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'NT' — Nucleotides</td>
</tr>
<tr>
<td></td>
<td>• 'AA' — Amino acids (default)</td>
</tr>
<tr>
<td></td>
<td>• 'none' — No alphabet</td>
</tr>
<tr>
<td></td>
<td>When Alphabet is 'none', the symbol list is based on the observed symbols. Each character can be any symbol, except for a hyphen (-) and a period (.), which are reserved for gaps.</td>
</tr>
<tr>
<td>CountsValue</td>
<td>Controls returning frequency (ratio of counts/total counts) or counts. Choices are true (counts) or false (frequency). Default is false.</td>
</tr>
<tr>
<td>GapsValue</td>
<td>Character vector or string that controls the counting of gaps in a sequence. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'all' — Counts all gaps</td>
</tr>
<tr>
<td></td>
<td>• 'noflanks' — Counts all gaps except those at the flanks of every sequence</td>
</tr>
<tr>
<td></td>
<td>• 'none' — Default. Counts no gaps.</td>
</tr>
<tr>
<td>AmbiguousValue</td>
<td>Controls counting ambiguous symbols. Enter 'Count' to add partial counts to the standard symbols.</td>
</tr>
<tr>
<td>LimitsValue</td>
<td>Specifies whether to use part of the sequence. Enter a [1x2] vector with the first position and the last position to include in the profile. Default is [1, SeqLength].</td>
</tr>
</tbody>
</table>
Description

Profile = seqprofile(Seqs) returns Profile, a matrix of size [20 (or 4) x SequenceLength] with the frequency of amino acids (or nucleotides) for every column in the multiple alignment. The order of the rows is given by

- 4 nucleotides — A C G T/U
- 20 amino acids — A R N D C Q E G H I L K M F P S T W Y V

[Profile, Symbols] = seqprofile(Seqs) returns Symbols, a unique symbol list where every symbol in the list corresponds to a row in Profile, the profile.

seqprofile(Seqs, ...) returns a unique symbol list where every symbol in the list corresponds to a row in Profile, the profile.

Examples

Calculate Sequence Profile

Create an array of structures representing a multiple alignment of amino acids:

seqs = fastaread('pf00002.fa');

Return the sequence profile and symbol list from position 50 through 55 of the set of multiply aligned sequences, counting all gaps.

[Profile2, Symbols2] = seqprofile(seqs,'limits',[50 55],'gaps','all')

Profile2 = 21x6

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0313</td>
<td>0.0313</td>
<td>0.1563</td>
<td>0.4375</td>
<td>0.1250</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.3750</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.0938</td>
<td>0.1563</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0313</td>
<td>0</td>
</tr>
</tbody>
</table>
Symbols2 =
'ARNDCQEGHILKMFPSTWYV-'

See Also
fastaread | multialignread | multialignwrite | seqconsensus | seqdisp | seqlogo

Introduced before R2006a
**seqqcplot**

Create quality control plots for sequence and quality data

**Syntax**

```matlab
seqqcplot(dataSource)
seqqcplot(dataSource,type)
seqqcplot(dataSource,type,encoding)
seqqcplot(___,Name,Value)
H = seqqcplot(___)
```

**Description**

`seqqcplot(dataSource)` generates a figure with quality control (QC) plots of sequence and quality data from `dataSource`. The figure contains the following types of QC plots.

- Box plot for the average quality score at each sequence position
- Bar plot for the sequence base composition at each sequence position
- Histogram of the average sequence quality score distribution
- Histogram of the GC-content distribution
- Histogram of the sequence length distribution

In the figure, you can click a specific plot to open it in a separate window.

`seqqcplot(dataSource,type)` generates a QC plot specified by `type`.

`seqqcplot(dataSource,type,encoding)` also specifies the encoding format of the base quality in the input file.

`seqqcplot(___,Name,Value)` uses any of the input arguments in the previous syntaxes and additional options specified by one or more `Name,Value` pair arguments.

`H = seqqcplot(___)` returns the figure handle `H` of the output figure.

**Examples**

**Create Quality Control Plots for Sequence and Quality Data**

Plot quality control plots for sequence statistics and quality data from a FASTQ file.

```matlab
seqqcplot('SRR005164_1_50.fastq');
```
Plot only the box plot of average quality score for each sequence position.

```
seqqcplot('SRR005164_1_50.fastq','QualityBoxplot');
```

![Quality Boxplot](image)

Plot the quality data of sequences with a minimum mean quality of 25.

```
seqqcplot('SRR005164_1_50.fastq','MeanQuality',25);
```
Plot the data of sequences having a minimum mean quality of 25 and a minimum sequence length of 100.

seqqcplot('SRR005164_1_50.fastq','MeanQuality',25,'MinLength',100);
Functions and Apps

Quality Boxplot

Base Composition

Quality Distribution

GC Distribution

Length Distribution

Base Positions: 1, Inf;  Minimum Length: 100;  Minimum Mean Quality: 25
Produce QC plots for the quality data corresponding to the subsequences from base position 10 to 100.

seqqcplot('SRR005164_1_50.fastq','BasePositions',[10 100]);
Base Positions: 10, 100; Minimum Length: 0; Minimum Mean Quality: -inf
Input Arguments

dataSource — Sequence and quality information
BioMap object | BioRead object | character vector | string | string vector | cell array of character vector

Sequence and quality information, specified as a BioMap object, BioRead object, character vector, string, string vector, or cell array of character vectors representing the names of FASTQ, SAM, or BAM files.

seqqcplot uses the read quality data, instead of the alignment quality, if you specify SAM or BAM files, a BioRead or BioMap object.

Example: 'SRR005164_1_50.fastq'

type — Name of QC plot to generate
'Summary' (default) | 'QualityBoxplot' | 'CompositionLine' | 'CompositionBar' | 'QualityDistribution' | 'GCDistribution' | 'LengthDistribution'

Name of the QC plot to generate, specified as one of the following:

<table>
<thead>
<tr>
<th>Name of QC Plot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'QualityBoxplot'</td>
<td>Box plot for the average quality score at each sequence position.</td>
</tr>
<tr>
<td>'CompositionLine'</td>
<td>Line plot for the sequence base composition at each sequence position.</td>
</tr>
<tr>
<td>'CompositionBar'</td>
<td>Bar plot for the sequence base composition at each sequence position.</td>
</tr>
<tr>
<td>'QualityDistribution'</td>
<td>Histogram of the average sequence quality score distribution.</td>
</tr>
<tr>
<td>'GCDistribution'</td>
<td>Histogram of the GC-content distribution.</td>
</tr>
<tr>
<td>'LengthDistribution'</td>
<td>Histogram of the sequence length distribution.</td>
</tr>
<tr>
<td>'Summary'</td>
<td>Summary figure containing all available QC plots, except the 'CompositionLine' plot. The figure also shows the values of name-value pairs that were used to generate the plots. If name-value pairs were not specified, it shows the corresponding default values instead.</td>
</tr>
</tbody>
</table>

By default, all available QC plots are plotted as subplots in a figure. To open a specific subplot in a separate figure window, click the subplot.

Example: 'QualityBoxplot'

encoding — Encoding format of base quality
'Illumina18' (default) | 'Sanger' | 'Solexa' | 'Illumina13' | 'Illumina15' | 'Illumina19'

Encoding format of the base quality, specified as one of the following:

- 'Sanger'
- 'Solexa'
- 'Illumina13'

1-1541
• 'Illumina15'
• 'Illumina18'
• 'Illumina19'

Example: 'Sanger'

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: 'MeanQuality', 5

MeanQuality — Minimum threshold on average base quality across each sequence

-Inf (default) | numeric scalar

Minimum threshold on the average base quality across each sequence, specified as a numeric scalar. The function considers only sequences with average quality score equal to or greater than the threshold. The threshold value is interpreted according to the specified encoding format. Default is -Inf, that is, any sequence is considered.

Example: 'MeanQuality', 5

MinLength — Minimum threshold on sequence length

0 (default) | nonnegative numeric scalar

Minimum threshold on the sequence length, specified as a nonnegative numeric scalar. The function considers only sequences with length equal to or greater than the threshold.

Example: 'MinLength', 100

BasePositions — Base position range for subsequences

[1 Inf] (default) | two-element vector

Base position range for subsequences, specified as a two-element vector. The function considers only the subsequences in the specified position range. Default is [1 Inf], that is, the entire length of each sequence is considered.

Example: 'BasePositions', [5 50]

Output Arguments

H — Handle to output figure

figure handle

Handle to the output figure, returned as a figure handle.

See Also

seqfilter | seqsplit | seqsplitpe | seqtrim

Introduced in R2017a
seqrcomplement

Calculate reverse complementary strand of nucleotide sequence

Syntax

\[ SeqRC = \text{seqrcomplement}(\text{SeqNT}) \]

Arguments

<table>
<thead>
<tr>
<th>SeqNT</th>
<th>Nucleotide sequence specified by any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string with the letters A, C, G, T, U, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N.</td>
</tr>
<tr>
<td></td>
<td>• Row vector of integers from the table Mapping Nucleotide Integers to Letter Codes.</td>
</tr>
<tr>
<td></td>
<td>• MATLAB structure containing a Sequence field that contains a nucleotide sequence, such as returned by emblread, fastaread, fastqread, genbankread, getembl, or getgenbank.</td>
</tr>
</tbody>
</table>

Description

\[ SeqRC = \text{seqrcomplement}(\text{SeqNT}) \] calculates the reverse complementary strand of a DNA or RNA nucleotide sequence. The return sequence, *SeqRC*, reads from 3' --> 5' and is in the same format as *SeqNT*. For example, if *SeqNT* is a vector of integers, then so is *SeqRC*.

<table>
<thead>
<tr>
<th>Nucleotide in <em>SeqNT</em></th>
<th>Converts to This Nucleotide in <em>SeqRC</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T or U</td>
</tr>
<tr>
<td>C</td>
<td>G</td>
</tr>
<tr>
<td>G</td>
<td>C</td>
</tr>
<tr>
<td>T or U</td>
<td>A</td>
</tr>
</tbody>
</table>

Examples

Return the reverse complement of a DNA nucleotide sequence.

s = 'ATCG'
seqrcomplement(s)

ans =
CGAT

See Also

codoncount | palindromes | seqcomplement | seqreverse | seqviewer

Introduced before R2006a
seqreverse

Calculate reverse strand of nucleotide sequence

Syntax

\[ SeqR = \text{seqreverse}(SeqNT) \]

Arguments

<table>
<thead>
<tr>
<th>( SeqNT )</th>
<th>Nucleotide sequence specified by any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Character vector or string with the letters A, C, G, T, U, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N.</td>
<td></td>
</tr>
<tr>
<td>• Row vector of integers from the table Mapping Nucleotide Integers to Letter Codes.</td>
<td></td>
</tr>
<tr>
<td>• MATLAB structure containing a Sequence field that contains a nucleotide sequence, such as returned by emblread, fastaread, fastqread, genbankread, getembl, or getgenbank.</td>
<td></td>
</tr>
</tbody>
</table>

Description

\( SeqR = \text{seqreverse}(SeqNT) \) calculates the reverse strand of a DNA or RNA nucleotide sequence. The return sequence, \( SeqR \), reads from 3' -> 5' and is in the same format as \( SeqNT \). For example, if \( SeqNT \) is a vector of integers, then so is \( SeqR \).

Examples

Return the reverse strand of a DNA nucleotide sequence.

\[
\begin{align*}
s &= \text{‘ATCG’} \\
\text{seqreverse}(s) \\
\text{ans} &= \\
\text{GCTA}
\end{align*}
\]

See Also

codoncount | fliplr | palindromes | seqcomplement | seqrcomplement | seqviewer

Introduced before R2006a
seqshoworfs

Display open reading frames in sequence

Syntax

seqshoworfs(SeqNT)

seqshoworfs(SeqNT, ...'Frames', FramesValue, ...)
seqshoworfs(SeqNT, ...'GeneticCode', GeneticCodeValue, ...)
seqshoworfs(SeqNT, ...'MinimumLength', MinimumLengthValue, ...)
seqshoworfs(SeqNT, ...'AlternativeStartCodons', AlternativeStartCodonsValue, ...)
seqshoworfs(SeqNT, ...'Color', ColorValue, ...)
seqshoworfs(SeqNT, ...'Columns', ColumnsValue, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeqNT</td>
<td>Nucleotide sequence. Enter either a character vector or string with A, T (U), G, C, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N, or a vector of integers. You can also enter a structure with the field Sequence.</td>
</tr>
<tr>
<td>FramesValue</td>
<td>Property to select the frame. Enter 1, 2, 3, -1, -2, -3, enter a vector with integers, or 'all'. The default value is the vector [1 2 3]. Frames -1, -2, and -3 correspond to the first, second, and third reading frames for the reverse complement.</td>
</tr>
<tr>
<td>GeneticCodeValue</td>
<td>Genetic code name. Enter a code number or a code name from the table Genetic Code.</td>
</tr>
<tr>
<td>MinimumLengthValue</td>
<td>Property to set the minimum number of codons in an ORF.</td>
</tr>
<tr>
<td>AlternativeStartCodonsValue</td>
<td>Property to control using alternative start codons. Enter either true or false. The default value is false.</td>
</tr>
</tbody>
</table>
**ColorValue**

Color to highlight the reading frame. Specify one of the following:
- Three-element numeric vector of RGB values
- Character vector or string containing a predefined single-letter color code
- Character vector or string containing a predefined color name

For example, to use cyan, enter `[0 1 1]`, `’c’`, or `’cyan’`. For more information on specifying colors, see “Color Options” on page 1-1548.

To specify different colors for the three reading frames, use a 1-by-3 cell array of color values. If you are displaying reverse complement reading frames, then use a 1-by-6 cell array of color values.

The default color scheme is blue for the first reading frame, red for the second, and green for the third.

**ColumnsValue**

Property to specify the number of columns in the output.

### Genetic Code

<table>
<thead>
<tr>
<th>Code Number</th>
<th>Code Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Standard</td>
</tr>
<tr>
<td>2</td>
<td>Vertebrate Mitochondrial</td>
</tr>
<tr>
<td>3</td>
<td>Yeast Mitochondrial</td>
</tr>
<tr>
<td>4</td>
<td>Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma/Spiroplasma</td>
</tr>
<tr>
<td>5</td>
<td>Invertebrate Mitochondrial</td>
</tr>
<tr>
<td>6</td>
<td>Ciliate, Dasycladacean, and Hexamita Nuclear</td>
</tr>
<tr>
<td>9</td>
<td>Echinoderm Mitochondrial</td>
</tr>
<tr>
<td>10</td>
<td>Euplotid Nuclear</td>
</tr>
<tr>
<td>11</td>
<td>Bacterial and Plant Plastid</td>
</tr>
<tr>
<td>12</td>
<td>Alternative Yeast Nuclear</td>
</tr>
<tr>
<td>13</td>
<td>Ascidian Mitochondrial</td>
</tr>
<tr>
<td>14</td>
<td>Flatworm Mitochondrial</td>
</tr>
<tr>
<td>15</td>
<td>Blepharisma Nuclear</td>
</tr>
<tr>
<td>16</td>
<td>Chlorophycean Mitochondrial</td>
</tr>
<tr>
<td>21</td>
<td>Trematode Mitochondrial</td>
</tr>
<tr>
<td>22</td>
<td>Scenedesmus Obliquus Mitochondrial</td>
</tr>
<tr>
<td>23</td>
<td>Thraustochytrium Mitochondrial</td>
</tr>
</tbody>
</table>
Description

seqshoworfs identifies and highlights all open reading frames using the standard or an alternative genetic code.

seqshoworfs(SeqNT) displays the sequence with all open reading frames highlighted, and it returns a structure of start and stop positions for each ORF in each reading frame. The standard genetic code is used with start codon 'AUG' and stop codons 'UAA', 'UAG', and 'UGA'.

seqshoworfs(SeqNT, ...'PropertyName', PropertyValue, ...) calls seqshoworfs with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

seqshoworfs(SeqNT, ...'Frames', FramesValue, ...) specifies the reading frames to display. The default is to display the first, second, and third reading frames with ORFs highlighted in each frame.

seqshoworfs(SeqNT, ...'GeneticCode', GeneticCodeValue, ...) specifies the genetic code to use for finding open reading frames.

seqshoworfs(SeqNT, ...'MinimumLength', MinimumLengthValue, ...) sets the minimum number of codons for an ORF to be considered valid. The default value is 10.

seqshoworfs(SeqNT, ...'AlternativeStartCodons', AlternativeStartCodonsValue, ...) uses alternative start codons if AlternativeStartCodons is set to true. For example, in the human mitochondrial genetic code, AUA and AUU are known to be alternative start codons. For more details on alternative start codons, see https://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=t#SG1

seqshoworfs(SeqNT, ...'Color', ColorValue, ...) specifies the color used to highlight the open reading frames in the output display. The default color scheme is blue for the first reading frame, red for the second, and green for the third.

seqshoworfs(SeqNT, ...'Columns', ColumnsValue, ...) specifies how many columns per line to use in the output. The default value is 64.

Examples

Display the open reading frames in a random nucleotide sequence.

s = randseq(200, 'alphabet', 'dna');
seqshoworfs(s);
Display the open reading frames in a GenBank sequence.

```matlab
HLA_DQB1 = getgenbank('NM_002123');
seqshoworfs(HLA_DQB1.Sequence);
```

### More About

#### Color Options

The following lists the predefined colors and their RGB triplet equivalents. The short names and long names are character vectors that specify one of eight preset colors. The RGB triplet is a three-
element row vector whose elements specify the intensities of the red, green, and blue components of the color; the intensities must be in the range $[0\ 1]$. 

<table>
<thead>
<tr>
<th>RGB Triplet</th>
<th>Short Name</th>
<th>Long Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1 1 0]</td>
<td>y</td>
<td>yellow</td>
</tr>
<tr>
<td>[1 0 1]</td>
<td>m</td>
<td>magenta</td>
</tr>
<tr>
<td>[0 1 1]</td>
<td>c</td>
<td>cyan</td>
</tr>
<tr>
<td>[1 0 0]</td>
<td>r</td>
<td>red</td>
</tr>
<tr>
<td>[0 1 0]</td>
<td>g</td>
<td>green</td>
</tr>
<tr>
<td>[0 0 1]</td>
<td>b</td>
<td>blue</td>
</tr>
<tr>
<td>[1 1 1]</td>
<td>w</td>
<td>white</td>
</tr>
<tr>
<td>[0 0 0]</td>
<td>k</td>
<td>black</td>
</tr>
</tbody>
</table>

See Also

codoncount | cpgisland | geneticcode | regexp | seqdisp | seqshowwords | seqviewer | seqwordcount

Introduced before R2006a
seqshowwords

(To be removed) Graphically display words in sequence

**Note** seqshowwords will be removed in a future release.

**Syntax**

```matlab
Struct = seqshowwords(Seq, Word)
seqshowwords(Seq, Word, ...'Color', ColorValue, ...)
seqshowwords(Seq, Word, ...'Columns', ColumnsValue, ...)
seqshowwords(Seq, Word, ...'Alphabet', AlphabetValue, ...)
```

**Description**

`Struct = seqshowwords(Seq, Word)` opens a separate window displaying a sequence with all occurrences of one or more words highlighted. It also returns a structure containing the start and stop positions for all occurrences of the words in the sequence.

`seqshowwords(Seq, Word, ...'PropertyName', PropertyValue, ...)` calls `seqshowwords` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each `PropertyName` in single quotation marks. Each `PropertyName` is case insensitive. These property name/property value pairs are as follows:

- `seqshowwords(Seq, Word, ...'Color', ColorValue, ...)` specifies the color to highlight the words in the output display of the sequence. Default is red.
- `seqshowwords(Seq, Word, ...'Columns', ColumnsValue, ...)` specifies how many columns or characters per line in the output display of the sequence. Default is 64.
- `seqshowwords(Seq, Word, ...'Alphabet', AlphabetValue, ...)` specifies the alphabet for the sequence and the word or words. Choices are 'AA' or 'NT' (default).

**Input Arguments**

- **Seq**
  Amino acid or nucleotide sequence specified by any of the following:
  - Character vector or string containing letters representing amino acids or nucleotides, such as returned by `int2aa` or `int2nt`.
  - MATLAB structure containing a `Sequence` field, such as returned by `fastaread`, `fastqread`, `emblread`, `getembl`, `genbankread`, `getgenbank`, `getgenpept`, `genpeptread`, `getpdb`, `pdbread`, or `sffread`.

  Default:

- **Word**
  One or more short amino acid or nucleotide sequences specified by any of the following:
• Character vector or string of letters
• Regular expression (MATLAB)
• Cell array of character vectors, string vector, or regular expressions

Note If the search word or words contain IUB/IUPAC amino acid or nucleotide symbols that map to multiple nucleotides or amino acids, then `seqshowwords` shows all possible matches. For details about how symbols are interpreted, see Nucleotide Conversion and Amino Acid Conversion. For example, the symbol R represents either G or A. If `Word` is ‘ART’, then `seqshowwords` shows occurrences of both ‘AAT’ and ‘AGT’.

Tip If `Word` contains a repeating pattern, such as ‘TATA’, then `seqshowwords` does not highlight overlapping patterns of TA in the sequence. To highlight multiple repeats of TA in a sequence, use a regular expression, such as ‘TA(TA)*TA’, for `Word`. For more information, see “Examples” on page 1-1552.

Default:

ColorValue

Color to highlight all occurrences of one or more words in the sequence. Specify the color with one of the following:

• Three-element numeric vector of RGB values
• Character vector or string containing a predefined single-letter color code
• Character vector or string containing a predefined color name

For example, to use cyan, enter `[0 1 1]`, ‘c’, or ‘cyan’. For more information on specifying colors, see “Color Options” on page 1-1553.

Default: Red, which is specified by `[1 0 0]`, ‘r’, or ‘red’

ColumnsValue

Positive integer specifying how many columns or characters per line in the output display of the sequence.

Default: 64

AlphabetValue

Character vector or string specifying the type of sequences. Choices are ‘AA’ or ‘NT’ (default).

Output Arguments

Struct

MATLAB structure containing the start and stop positions of all occurrences or the word or words in the sequence. It includes two fields.
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start</strong></td>
<td>Row vector containing the start position of each occurrence of the search word or words.</td>
</tr>
<tr>
<td><strong>Stop</strong></td>
<td>Row vector containing the stop position of each occurrence of the search word or words.</td>
</tr>
</tbody>
</table>

### Examples

Search for a word containing multiple symbols:

% Highlight the word 'BART' which represents 'TAGT' and 'TAAT'
seqshowwords('GCTAGTAACGTATATAAT','BART')

ans =
   Start: [3 17]
   Stop: [6 20]

Search for a word that repeats, excluding overlaps:

% Highlight all occurrences of 'TATA', excluding those that are % already part of another matched word.
seqshowwords('GCTATAACGTATATATAAT','TATA')

ans =
   Start: [3 10 14]
   Stop: [6 13 17]

Search for a word that repeats, including overlaps:

% Use the regular expression 'TA(TA)*TA' to highlight all multiple % repeats of 'TA'
seqshowwords('GCTATAACGTATATATAAT','TA(TA)*TA')

ans =
   Start: [3 10]
   Stop: [6 19]
Search for multiple words:

% Use a cell array as input to highlight both the words
% 'CG' and 'GC'
seqshowwords('GCTATAACGTATATATATA', {'CG', 'GC'})

ans =

    Start: [1 8]
    Stop: [2 9]

More About

Color Options

The following lists the predefined colors and their RGB triplet equivalents. The short names and long
names are character vectors that specify one of eight preset colors. The RGB triplet is a three-
element row vector whose elements specify the intensities of the red, green, and blue components of
the color; the intensities must be in the range [0 1].

<table>
<thead>
<tr>
<th>RGB Triplet</th>
<th>Short Name</th>
<th>Long Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1 1 0]</td>
<td>y</td>
<td>yellow</td>
</tr>
<tr>
<td>[1 0 1]</td>
<td>m</td>
<td>magenta</td>
</tr>
<tr>
<td>[0 1 1]</td>
<td>c</td>
<td>cyan</td>
</tr>
<tr>
<td>[1 0 0]</td>
<td>r</td>
<td>red</td>
</tr>
<tr>
<td>[0 1 0]</td>
<td>g</td>
<td>green</td>
</tr>
<tr>
<td>[0 0 1]</td>
<td>b</td>
<td>blue</td>
</tr>
<tr>
<td>[1 1 1]</td>
<td>w</td>
<td>white</td>
</tr>
<tr>
<td>[0 0 0]</td>
<td>k</td>
<td>black</td>
</tr>
</tbody>
</table>
Alternatives

The `seqviewer` function opens the Biological Sequence Viewer, where you search for words in a sequence by selecting **Sequence > Find Word**. The Biological Sequence Viewer does not:

- Allow searching for multiple words in one step
- Return a structure containing the start and stop positions for all occurrences of the word in the sequence

Compatibility Considerations

`seqshowwords` will be removed

*Not recommended starting in R2020a*

`seqshowwords` will be removed in a future release.

See Also

clean | palindromes | regexp | restrict | seqdisp | seqviewer | seqwordcount | strfind

Topics

“Exploring a Nucleotide Sequence Using the Sequence Viewer App”

Regular Expressions (MATLAB)

**Introduced before R2006a**
seqsplit

Split sequences into separate files based on barcodes

Syntax

seqsplit(fastqFile, barcodeFile)
seqsplit(___ ,Name,Value)
[outFiles,N] = seqsplit( ___ )

Description

seqsplit(fastqFile, barcodeFile) splits sequences in fastqFile according to the barcodes in barcodeFile and saves the sequences in separate files. By default, the output file name consists of the input file name followed by the barcode identifier. Sequences that do not match any provided barcodes, or that match multiple barcodes ambiguously, are saved in a file with the suffix '_unmatched' instead of the barcode identifier.

seqsplit(___,Name,Value) uses additional options specified by one or more Name,Value pair arguments.

[outFiles,N] = seqsplit( ___ ) returns the names of output files in a cell array outFiles. N represents a vector containing the numbers of sequences saved in each output file.

Examples

Split sequences into separate files based on barcodes

Create a tab-delimited file with barcode IDs and barcode sequences.

```matlab
barcodeInfo = {'ID1', 'AAAAC'; 'ID2', 'AGATT'; 'ID3', 'GACTT'};
writetable(cell2table(barcodeInfo), 'barcodeExample.txt', ...
    'Delimiter', '	', 'WriteVariableNames', false);
```

Split sequences into separate output files based on the barcode sequences. By default, the function assumes that the barcode is located at the 5' end of each sequence, and no mismatches are allowed during barcode matching.

```matlab
[outFiles, N] = seqsplit('SRR005164_1_50.fastq', 'barcodeExample.txt');
```

Check the number of sequences in each output file after splitting.

```matlab
N
```

```matlab
N = 3x1
```

```matlab
2
1
1
```

Allow up to two mismatches during the barcode matching.
[outFiles, N] = seqsplit('SRR005164_1_50.fastq', 'barcodeExample.txt', ...
'MaxMismatches',2,'OutputSuffix','_MM2_split');

N
N = 3×1
 5
 9
 5

Input Arguments

fastqFile — Names of FASTQ files with sequence and quality information
character vector | string | string vector | cell array of character vectors

Names of FASTQ-formatted files with sequence and quality information, specified as a character vector, string, string vector, or cell array of character vectors.
Example: 'SRR005164_1_50.fastq'

barcodeFile — Name of barcode files with barcode information
character vector | string

Name of barcode file with barcode information, specified as a character vector or string. The file must be tab-formatted, containing barcode IDs and barcode sequences. Each ID must be followed by a barcode sequence, and all barcode sequences must have the same length.
Example: 'barcodeExample.txt'

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: 'MaxMismatches',2 specifies to allow up to 2 mismatches during barcode matching.

MaxMismatches — Maximum number of mismatches allowed during barcode matching
0 (default) | nonnegative integer

Maximum number of mismatches allowed during barcode matching, specified as a nonnegative integer. The default is 0, that is, no mismatches are allowed.
Example: 'MaxMismatches',2

BarcodeFormat — Type of barcode to match
5 (default) | 3

Type of barcode to match, specified as 3 or 5. A value of 5 corresponds to the barcode located at the 5' end of each sequence, and 3 corresponds to the 3' end.
Example: 'BarcodeFormat',3

RemoveBarcode — Whether to remove the barcode
true (default) | false
Whether to remove the barcode and corresponding quality information from the matched sequences, specified as true or false. The default is true.

Example: 'RemoveBarcode',false

**WriteUnmatched** — Whether to save unmatched sequences
false (default) | true

Whether to save unmatched sequences and corresponding quality information in a separate output file, specified as true or false. The output file name has the suffix '_unmatched' instead of the barcode ID.

Example: 'WriteUnmatched',true

**OutputDir** — Relative or absolute path to output file directory
character vector | string

Relative or absolute path to the output file directory, specified as a character vector or string. The default is the current directory.

Example: 'OutputDir','F:\results'

**OutputSuffix** — Suffix to use in output file name
'_split' (default) | character vector | string

Suffix to use in the output file name, specified as a character vector or string. It is inserted after the input file name and before the barcode ID. The default is '_split'.

Example: 'OutputSuffix','_MisMatches2_split'

**UseParallel** — Whether to perform computation in parallel
false (default) | true

Whether to perform computation in parallel, specified as true or false.

For parallel computing, you must have Parallel Computing Toolbox. If a parallel pool does not exist, one is created automatically when the auto-creation option is enabled in your parallel preferences. Otherwise, computation runs in serial mode.

**Note** There is a cost associated with sharing large input files across workers in a distributed environment. In some cases, running in parallel may not be beneficial in terms of performance.

Example: 'UseParallel',true

**Output Arguments**

<table>
<thead>
<tr>
<th><strong>outFiles</strong> — Output file names</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell array of character vectors</td>
</tr>
</tbody>
</table>

Output file names, returned as a cell array of character vectors. By default, the name of each output file consists of the input file name followed by the output suffix ('_split') and the barcode identifier.

<table>
<thead>
<tr>
<th><strong>N</strong> — Numbers of sequences saved in each output file</th>
</tr>
</thead>
<tbody>
<tr>
<td>scalar</td>
</tr>
</tbody>
</table>

1-1557
Numbers of sequences saved in each output file, returned as a scalar or an \( n \)-by-1 vector, where \( n \) is the number of output files. If there are multiple output files, the order within \( N \) corresponds to the order of the output files.

**Extended Capabilities**

**Automatic Parallel Support**
Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true`.

For more information, see the `'UseParallel'` name-value pair argument.

**See Also**
`seqfilter` | `seqsplitpe` | `seqtrim`

**Introduced in R2016b**
seqsplitpe

Split merged paired-end sequences into separate files

Syntax

seqsplitpe(fastqFile)
seqsplitpe(__,Name,Value)
[outFiles,N] = seqsplitpe(__)

Description

seqsplitpe(fastqFile) splits merged paired-end sequences from fastqFile into two separate files. Each sequence is split in the middle. The first half of the sequence is saved in the first output file and the other half in the second output file. By default, each output file name consists of the input file name appended with a suffix '_1' or '_2' before the file extension.

seqsplitpe(__,Name,Value) uses additional options specified by one or more Name,Value pair arguments.

[outFiles,N] = seqsplitpe(__) returns the names of output files in a cell array outFiles. N represents a vector containing the numbers of sequences saved in each output file.

Examples

Split merged paired-end sequences into separate files

Split each of the paired-end sequences in half, and store each half in separate output files.

[outFiles, N] = seqsplitpe('SXX123456_merged.fastq');

Check the number of sequences in each output file.

N

N = 2x1

50
50

Input Arguments

fastqFile — Names of FASTQ files with sequence and quality information
character vector | string | string vector | cell array of character vectors

Names of FASTQ files with sequence and quality information, specified as a character vector, string, string vector, or cell array of character vectors.

Example: 'SRR005164_1_50.fastq'
**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `OutputSuffix',PairedEnd_split'` specifies to use the custom suffix in the output file names.

**OutputDir — Relative or absolute path to output file directory**

cell array of character vectors

Relative or absolute path to the output file directory, specified as a character vector or string. The default is the current directory.

Example: `OutputDir','F:\results`

**OutputSuffix — Custom suffix to use in output file names**

`''` (default) | character vector | string

Custom suffix to use in the output file names, specified as a character vector or string. It is inserted after the input file name and before the suffix '_1' or '_2'. The default is ''. 

Example: `OutputSuffix','_MisMatches2`

**UseParallel — Boolean indicating whether to perform computation in parallel**

`false` (default) | `true`

Boolean indicating whether to perform computation in parallel, specified as `true` or `false`.

For parallel computing, you must have Parallel Computing Toolbox. If a parallel pool does not exist, one is created automatically when the auto-creation option is enabled in your parallel preferences. Otherwise, computation runs in serial mode.

**Note** There is a cost associated with sharing large input files across workers in a distributed environment. In some cases, running in parallel may not be beneficial in terms of performance.

Example: `UseParallel',true`

**Output Arguments**

**outFiles — Output file names**

cell array of character vectors

Output file names, returned as a cell array of character vectors. By default, the name of each output file consists of the input file name appended with a suffix '_1' or '_2' before the file extension.

**N — Number of sequences saved in each output file**

vector

Number of sequences saved in each output file, returned as an n-by-1 vector where n is the number of output files. If there are multiple output files, the order within N corresponds to the order of the output files.
Extended Capabilities

Automatic Parallel Support
Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set 'UseParallel' to true.

For more information, see the 'UseParallel' name-value pair argument.

See Also
seqfilter | seqsplit | seqtrim

Introduced in R2016b
seqtrim

Trim sequences based on specified criterion

Syntax

seqtrim(fastqFile)
seqtrim(fastqFile,Name,Value)
[outFiles,nSeqTrimmed,nSeqUntrimmed] = seqtrim( ___ )

Description

seqtrim(fastqFile) trims the sequences in fastqFile and saves the trimmed sequences in new
FASTQ files. By default, the trimmed sequences are saved under file names with the suffix
'.trimmed' appended. If you do not specify any trimming criterion, the function trims sequences
using the default.

seqtrim(fastqFile,Name,Value) uses additional options specified by one or more Name,Value
pair arguments.

[outFiles,nSeqTrimmed,nSeqUntrimmed] = seqtrim( ___ ) returns a cell array outFiles
with the names of output files. nSeqTrimmed and nSeqUntrimmed represent the numbers of
sequences trimmed and untrimmed from each input file, respectively.

Examples

Trim sequences in a FASTQ file

Trim each sequence when the number of bases with quality below 20 is greater than 3 within a
sliding window of size 25.

[outFile,nt,unt] = seqtrim('SRR005164_1_50.fastq', 'Method', 'MaxNumberLowQualityBases', ...'
'Threshold', [3 20], 'WindowSize', 25);

Check the number of sequences that were trimmed.

nt
nt = 36

Check the number of sequences that were untrimmed.

unt
unt = 14

Trim the first 10 bases of each sequence.

[outfile,nt] = seqtrim('SRR005164_1_50.fastq','Method','Termini', ...'
'Threshold',[10 0]);

Trim the last 5 bases.
[outfile,nt] = seqtrim('SRR005164_1_50.fastq','Method','Termini', ...  
'Threshold',[0 5]);

Trim each sequence at position 50.

[outfile,nt] = seqtrim('SRR005164_1_50.fastq','Method','BasePositions', ...  
'Threshold',[1 50]);

Trim each sequence when the running average base quality becomes less than 20.

[outFile,nt,unt] = seqtrim('SRR005164_1_50.fastq','Method','MeanQuality', ...  
'Threshold',20)

Trim each sequence when the percentage of bases with quality below 10 is more than 15.

[outFile,nt,unt] = seqtrim('SRR005164_1_50.fastq','Method','MaxPercentLowQualityBases', ...  
'Threshold',[15 10])

Input Arguments

fastqFile — Names of FASTQ files with sequence and quality information
character vector | string | string vector | cell array of character vectors

Names of FASTQ-formatted files with sequence and quality information, specified as a character vector, string, string vector, or cell array of character vectors.

Example: ‘SRR005164_1_50.fastq’

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: ‘Method’, 'MaxNumberLowQualityBases', 'Threshold', [3 20] specifies to trim each sequence when the number of bases with quality below 20 is greater than 3.

Method — Criterion to trim sequences
'MaxNumberLowQualityBases' (default) | 'MaxPercentLowQualityBases' | 'MeanQuality' | 'BasePositions' | 'Termini'

Criterion to trim sequences, specified as one of the following options. Specify only one trimming criterion per function call.

- 'MaxNumberLowQualityBases' - applies a maximum threshold on the number of low-quality bases allowed before trimming a sequence starting at the 5' end.
- 'MaxPercentLowQualityBases' - applies a maximum threshold on the percentage of low-quality bases allowed before trimming a sequence starting at the 5' end.
- 'MeanQuality' - applies a minimum threshold on the running average base quality allowed before trimming a sequence starting at the 5' end.
- 'BasePositions' - trims each sequence according to the base positions (first base and last base) starting at the 5' end.
- 'Termini' - trims each sequence from either the 5' or 3' end or from both ends.
Use this name-value pair argument together with 'Threshold' to specify the appropriate threshold value. Depending on the trimming criterion, the corresponding value for 'Threshold' varies. See the 'Threshold' option for the default values.

**Note** Sequences resulting in empty sequences after trimming are saved in the output files as empty sequences. To remove empty sequences from files, use the seqfilter function with the 'MinLength' option set to the value of 1.

Example: 'Method','MaxNumberLowQualityBases','Threshold',[5 15]

**Threshold — Threshold value for trimming criterion**

**scalar | vector**

Threshold value for the trimming criterion, specified as a scalar or vector. Use this name-value pair to define the threshold value for the trimming criterion specified by 'Method'.

Depending on the trimming criterion, the corresponding value for 'Threshold' can be a scalar or two-element vector. If you do not specify 'Threshold', then the function uses the default threshold value of the corresponding method. For each trimming criterion, the function uses the encoding format of the base quality specified by the 'Encoding' name-value pair argument.

<table>
<thead>
<tr>
<th>'Method'</th>
<th>'Threshold'</th>
<th>Default 'Threshold' value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'MaxNumberLowQualityBases'</td>
<td>Two-element vector [V1 V2]. V1 is a nonnegative integer that specifies the maximum number of low-quality bases allowed before trimming. V2 specifies the minimum base quality. Any base with quality less than V2 is considered a low-quality base.</td>
<td>[0 10]</td>
</tr>
<tr>
<td>'MaxPercentLowQualityBases'</td>
<td>Two-element vector [V1 V2]. V1 is a scalar between 0 and 100 that specifies the maximum percentage of low quality bases allowed before trimming. V2 specifies the minimum base quality. Any base with quality less than V2 is considered a low-quality base.</td>
<td>[0 10]</td>
</tr>
<tr>
<td>'MeanQuality'</td>
<td>Positive scalar that specifies the minimum threshold on the running average base quality allowed before trimming a sequence starting at the 5' end.</td>
<td>0</td>
</tr>
</tbody>
</table>
Method

`'Threshold'

Default 'Threshold' value

<table>
<thead>
<tr>
<th>Method</th>
<th>Threshold</th>
<th>Default 'Threshold' value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>'BasePositions'</code></td>
<td>Two-element vector ([V1 \ V2]), where (V1) and (V2) are positive integers specifying the base positions to start trimming at the 5' end and 3' end, respectively. To trim only the 5' end of each sequence before position (V1), use ([V1 \ Inf]). To trim only the 3' end of each sequence after position (V2), use ([1 \ V2]).</td>
<td>([1 \ Inf]), that is, each sequence is left untrimmed.</td>
</tr>
<tr>
<td><code>'Termini'</code></td>
<td>Two-element vector ([V1 \ V2]), where (V1) and (V2) are nonnegative integers specifying the number of bases to trim at the 5' end and the 3' end, respectively. To trim (V1) bases at the 5' end only, use ([V1 \ 0]). To trim (V2) bases at the 3' end only, use ([0 \ V2]).</td>
<td>([0 \ 0]), that is, each sequence is left untrimmed.</td>
</tr>
</tbody>
</table>

Example: `'Method','MaxPercentLowQualityBases','Threshold',[10 20]`

**WindowSize — Size of sliding window to apply filtering criterion to sequence**

Inf (default) | positive integer

Size of the sliding window to apply the trimming criterion to a sequence, specified as a positive integer. The size of the window corresponds to the number of bases that the function uses at one time to apply the criterion. Any given sequence is trimmed before the first base of the window that violates the given criterion.

The sliding window can be applied to the following methods:

- `'MaxNumberLowQualityBases'`,
- `'MaxPercentLowQualityBases'`, and
- `'MeanQuality'`.

**Note** Sequences shorter than the size of the window are saved in the output file as empty sequences. To remove empty sequences from files, use the seqfilter function with the 'MinLength' option set to the value of 1.

Example: `'WindowSize',10`

**Encoding — Base quality encoding format**

'Illumina18' (default) | 'Sanger' | 'Solexa' | 'Illumina13' | 'Illumina15'
Base quality encoding format, specified as a character vector or string.
Example: 'Encoding','Sanger'

OutputDir — Relative or absolute path to output file directory
character vector | string

Relative or absolute path to the output file directory, specified as a character vector or string. The default is the current directory.
Example: 'OutputDir','F:\results'

OutputSuffix — Suffix to use in output file name
'_{trimmed}' (default) | character vector | string

Suffix to use in the output file name, specified as a character vector or string. It is inserted after the input file name and before the file extension. The default is '_trimmed'.
Example: 'OutputSuffix','_WindowSize10_trimmed'

UseParallel — Boolean indicating whether to perform computation in parallel
false (default) | true

Boolean indicating whether to perform computation in parallel, specified as true or false.

For parallel computing, you must have Parallel Computing Toolbox. If a parallel pool does not exist, one is created automatically when the auto-creation option is enabled in your parallel preferences. Otherwise, computation runs in serial mode.

Note

- There is a cost associated with sharing large input files across workers in a distributed environment. In some cases, running in parallel may not be beneficial in terms of performance.
- During parallel computations, the work is divided by files, not by sequences, meaning that, for a single large file, running in parallel does not make a difference.

Example: 'UseParallel',true

Output Arguments

outFiles — Output file names
cell array of character vectors

Output file names, returned as a cell array of character vectors.

nSeqTrimmed — Number of sequences trimmed from each input file
scalar | vector

Number of sequences trimmed from each input file, returned as a scalar or an n-by-1 vector where n is the number of input files. If there are multiple input files, the order within nSeqTrimmed corresponds to the order of the input files.
nSeqUntrimmed — Number of sequences untrimmed from each input file

scalar | vector

Number of sequences untrimmed from each input file, returned as a scalar or an \( n \)-by-1 vector where \( n \) is the number of input files. If there are multiple input files, the order within nSeqUntrimmed corresponds to the order of the input files.

Extended Capabilities

Automatic Parallel Support
Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set 'UseParallel' to true.

For more information, see the 'UseParallel' name-value pair argument.

See Also
seqfilter | seqsplit | seqsplitpe

Introduced in R2016b
seqviewer

Visualize and interactively explore biological sequences

Syntax

seqviewer
seqviewer(Seq)
seqviewer(Seq,Name,Value)
seqviewer('close')

Description

seqviewer opens the Sequence Viewer app.

seqviewer(Seq) loads a sequence Seq into the app, where you can view and interactively explore the sequence.

seqviewer(Seq,Name,Value) opens the app with additional options specified by one or more Name,Value pair arguments.

seqviewer('close') closes the Sequence Viewer app.

Input Arguments

Seq — Amino acid or nucleotide sequence
character vector | string | row vector of integers | structure

Amino acid or nucleotide sequence, specified as:

- Character vector or string containing single-letter codes or a file name with an extension of .gbk, .gpt, .fasta, .fa, or .ebi.
- Row vector of integers
- MATLAB structure containing a Sequence field that contains an amino acid or nucleotide sequence, such as returned by fastaread, fastqread, getgenpept, genpeptread, getpdb, pdbread, emblread, getembl, genbankread, or getgenbank

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: ‘Alphabet’,‘AA’ specifies that the aligned sequences are amino acid sequences.

Alphabet — Type of aligned sequences

‘AA’ | ‘NT’

Type of aligned sequences, specified as 'AA' for amino acid sequences or 'NT' for nucleotide sequences.
Example: ‘Alphabet’, 'AA'

Examples

Open and View a Biological Sequence

Retrieve a sequence from the GenBank database.

\[ S = \text{getgenbank('M10051')}; \]

Load the sequence into the Sequence Viewer app.

\[ \text{seqviewer}(S) \]

Alternatively, you can click Sequence Viewer on the Apps tab to open the app, and view the biological sequence \( S \).
Close the app.
seqviewer('close')

See Also
aa2nt | aaccount | aminolookup | basecount | baselookup | dimercount | emblread |
fastaread | fastawrite | genbankread | geneticcode | genpeptread | getembl | getgenbank |
getgenpept | nt2aa | proteinplot | seqcomplement | seqdisp | seqrcomplement |
seqreverse | seqshoworfs | seqshowwords | seqwordcount

Topics
“Exploring a Nucleotide Sequence Using the Sequence Viewer App”

Introduced before R2006a
seqwordcount

Count number of occurrences of word in sequence

Syntax

seqwordcount(Seq, Word)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq</td>
<td>Character vector or string containing a nucleotide or amino acid sequence. You can also enter a structure with the field Sequence.</td>
</tr>
<tr>
<td>Word</td>
<td>Enter a short sequence of characters.</td>
</tr>
</tbody>
</table>

Description

seqwordcount(Seq, Word) counts the number of times that a word appears in a sequence, and then returns the number of occurrences of that word.

If Word contains nucleotide or amino acid symbols that represent multiple possible symbols (ambiguous characters), then seqwordcount counts all matches. For example, the symbol R represents either G or A (purines). For another example, if word equals 'ART', then seqwordcount counts occurrences of both 'AAT' and 'AGT'.

Examples

seqwordcount does not count overlapping patterns multiple times. In the following example, seqwordcount reports three matches. TATATATA is counted as two distinct matches, not three overlapping occurrences.

seqwordcount('GCTATAACGTATATAT','TATA')

ans = 3

The following example reports two matches (TAGT and TAAT). B is the ambiguous code for G, T, or C, while R is an ambiguous code for G and A.

seqwordcount('GCTAGTAACGTATATAAT','BART')

ans = 2

See Also
codoncount | seq2regexp | seqshoworfs | seqshowwords | seqviewer | strfind

Introduced before R2006a
set (biograph)

Set property of biograph object

Syntax

set(BGobj)
set(BGobj, 'PropertyName')
set(BGobj, 'PropertyName', PropertyValue)
set(BGobj, 'Property1Name', Property1Value, 'Property2Name', Property2Value, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGobj</td>
<td>Biograph object created with the function biograph.</td>
</tr>
<tr>
<td>PropertyName</td>
<td>Property name for a biograph object.</td>
</tr>
<tr>
<td>PropertyValue</td>
<td>Value of the property specified by PropertyName.</td>
</tr>
</tbody>
</table>

Description

set(BGobj) displays possible values for all properties that have a fixed set of property values in BGobj, a biograph object.

set(BGobj, 'PropertyName') displays possible values for a specific property that has a fixed set of property values in BGobj, a biograph object.

set(BGobj, 'PropertyName', PropertyValue) sets the specified property of BGobj, a biograph object.

set(BGobj, 'Property1Name', Property1Value, 'Property2Name', Property2Value, ...) sets the specified properties of BGobj, a biograph object.
## Properties of a Biograph Object

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Character vector to identify the biograph object. Default is ''.</td>
</tr>
<tr>
<td>Label</td>
<td>Character vector to label the biograph object. Default is ''.</td>
</tr>
<tr>
<td>Description</td>
<td>Character vector that describes the biograph object. Default is ''.</td>
</tr>
<tr>
<td>LayoutType</td>
<td>Character vector that specifies the algorithm for the layout engine. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'hierarchical' (default) — Uses a topological order of the graph to assign levels, and then arranges the nodes from top to bottom, while minimizing crossing edges.</td>
</tr>
<tr>
<td></td>
<td>• 'radial' — Uses a topological order of the graph to assign levels, and then arranges the nodes from inside to outside of the circle, while minimizing crossing edges.</td>
</tr>
<tr>
<td></td>
<td>• 'equilibrium' — Calculates layout by minimizing the energy in a dynamic spring system.</td>
</tr>
<tr>
<td>EdgeType</td>
<td>Character vector that specifies how edges display. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'straight'</td>
</tr>
<tr>
<td></td>
<td>• 'curved' (default)</td>
</tr>
<tr>
<td></td>
<td>• 'segmented'</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> Curved or segmented edges occur only when necessary to avoid obstruction by nodes. Biograph objects with LayoutType equal to 'equilibrium' or 'radial' cannot produce curved or segmented edges.</td>
</tr>
<tr>
<td>Scale</td>
<td>Positive number that post-scales the node coordinates. Default is 1.</td>
</tr>
<tr>
<td>LayoutScale</td>
<td>Positive number that scales the size of the nodes before calling the layout engine. Default is 1.</td>
</tr>
<tr>
<td>EdgeTextColor</td>
<td>Three-element numeric vector of RGB values. Default is [0, 0, 0], which defines black.</td>
</tr>
<tr>
<td>EdgeFontSize</td>
<td>Positive number that sets the size of the edge font in points. Default is 8.</td>
</tr>
<tr>
<td>ShowArrows</td>
<td>Controls the display of arrows with the edges. Choices are 'on' (default) or 'off'.</td>
</tr>
<tr>
<td>ArrowSize</td>
<td>Positive number that sets the size of the arrows in points. Default is 8.</td>
</tr>
<tr>
<td>ShowWeights</td>
<td>Controls the display of text indicating the weight of the edges. Choices are 'on' or 'off' (default).</td>
</tr>
</tbody>
</table>
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShowTextInNodes</td>
<td>Character vector that specifies the node property used to label nodes when you display a biograph object using the view method. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'Label' — Uses the Label property of the node object (default).</td>
</tr>
<tr>
<td></td>
<td>• 'ID' — Uses the ID property of the node object.</td>
</tr>
<tr>
<td></td>
<td>• 'None'</td>
</tr>
<tr>
<td>NodeAutoSize</td>
<td>Controls precalculating the node size before calling the layout engine. Choices are 'on' (default) or 'off'.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> Set it to off if you want to apply different node sizes by changing the Size property.</td>
</tr>
<tr>
<td>NodeCallback</td>
<td>User-defined callback for all nodes. Enter the name of a function, a function handle, or a cell array with multiple function handles. After</td>
</tr>
<tr>
<td></td>
<td>using the view function to display the biograph object in the Biograph Viewer, you can double-click a node to activate the first callback, or</td>
</tr>
<tr>
<td></td>
<td>right-click and select a callback to activate. Default is the anonymous function, @(node) inspect(node), which displays the Property</td>
</tr>
<tr>
<td></td>
<td>Inspector dialog box.</td>
</tr>
<tr>
<td>EdgeCallback</td>
<td>User-defined callback for all edges. Enter the name of a function, a function handle, or a cell array with multiple function handles. After</td>
</tr>
<tr>
<td></td>
<td>using the view function to display the biograph object in the Biograph Viewer, you can right-click and select a callback to activate. Default is</td>
</tr>
<tr>
<td></td>
<td>the anonymous function, @(edge) inspect(edge), which displays the Property Inspector dialog box.</td>
</tr>
<tr>
<td>CustomNodeDrawFcn</td>
<td>Function handle to a customized function to draw nodes. Default is []</td>
</tr>
<tr>
<td>Nodes</td>
<td>Read-only column vector with handles to node objects of a biograph object. The size of the vector is the number of nodes. For properties of</td>
</tr>
<tr>
<td></td>
<td>node objects, see Properties of a Node Object.</td>
</tr>
<tr>
<td>Edges</td>
<td>Read-only column vector with handles to edge objects of a biograph object. The size of the vector is the number of edges. For properties of</td>
</tr>
<tr>
<td></td>
<td>edge objects, see Properties of an Edge Object.</td>
</tr>
</tbody>
</table>

### Examples

1. Create a biograph object with default node IDs.
   ```matlab
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
bg = biograph(cm)
Biograph object with 5 nodes and 9 edges.
```
2. Use the view method to display the biograph object.
   ```matlab
view(bg)
```
3 Use the `set` method to change the edge lines from curved to straight.

```matlab
set(bg, 'EdgeType', 'straight')
```

4 Display the biograph object again.

```matlab
view(bg)
```
See Also
biograph | get

Topics
biograph object on page 1-185

Introduced in R2008b
set

Set property of object

Syntax

newObject = set(object,Name,Value)
set(object,propertyName)
set(object)
allProperties = set(object)

Description

newObject = set(object,Name,Value) returns a new object that is a copy of object with
properties set to the values specified by using one or more name-value pairs. Use single quotes
around the property name. For example, newObj = set(brObj,'Sequence',
{'ACTCAG','GTCATG'}) specifies the Sequence property of brObj. You can specify any property
name, except NSeqs. See BioRead or BioMap for their properties.

set(object,propertyName) displays all possible values for the specified property PropName of
the object.

set(object) displays all properties of the object and their possible values.

allProperties = set(object) returns the structure allProperties containing all properties
of object and their possible values.

Examples

Modify NGS Data

Store read data from a SAM-formatted file in a BioRead object. Set 'InMemory' to true to load the
object into memory so that you can modify its properties.

br = BioRead('SRR005164_1_50.fastq','InMemory',true)

br =
    BioRead with properties:
        Quality: {50x1 cell}
        Sequence: {50x1 cell}
        Header: {50x1 cell}
        NSeqs: 50
        Name: ''

Check the list of object properties and their possible values. For example, the Header property takes
a cell array of strings as its value.

allProperties = set(br)
Specify custom header information that follows the pattern Header_1, Header_2, ... , Header_50.

```matlab
headers = cell(50,1);
for i = 1:50
    headers(i) = {'Header_' int2str(i)};
end
```

Set the header property of the `br` object. Use the same object as the output to update an existing object.

```matlab
br = set(br,'Header',headers)
```

Alternatively, you can set the property by using the dot notation.

```matlab
br.Header = headers
```

### Input Arguments

- **object** — Object containing read data
  - `BioRead` object | `BioMap` object

Object containing the read data, specified as a `BioRead` or `BioMap` object. If the object is not stored in memory, you cannot modify its properties, except the `Name` property.
Example: `readData`

**propertyName — Name of object property**
character vector | string

Name of the object property, specified as a character vector or string.
Example: 'Sequence'

**Output Arguments**

**newObject — New object with updated properties**
BioRead object | BioMap object

New object with updated properties, returned as a BioRead or BioMap object.

**allProperties — Structure containing all properties and values**
struct

Structure containing all properties and their possible values, returned as a struct. The field names are the property names, and the values are cell arrays of possible property values.

**See Also**

BioMap | BioRead

**Topics**

“Manage Sequence Read Data in Objects”

**Introduced in R2010a**
set (DataMatrix)

Set property of DataMatrix object

Syntax

set(DMObj)
set(DMObj, 'PropertyName')
DMObj = set(DMObj, 'PropertyName', PropertyValue)
DMObj = set(DMObj, 'Property1Name', Property1Value, 'Property2Name', Property2Value, ...)

Arguments

<table>
<thead>
<tr>
<th>DMObj</th>
<th>DataMatrix object, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>PropertyName</td>
<td>Property name of a DataMatrix object.</td>
</tr>
<tr>
<td>PropertyValue</td>
<td>Value of the property specified by PropertyName.</td>
</tr>
</tbody>
</table>

Description

set(DMObj) displays possible values for all properties that have a fixed set of property values in DMObj, a DataMatrix object.

set(DMObj, 'PropertyName') displays possible values for a specific property that has a fixed set of property values in DMObj, a DataMatrix object.

DMObj = set(DMObj, 'PropertyName', PropertyValue) sets the specified property of DMObj, a DataMatrix object.

DMObj = set(DMObj, 'Property1Name', Property1Value, 'Property2Name', Property2Value, ...) sets the specified properties of DMObj, a DataMatrix object.
Properties of a DataMatrix Object

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Character vector that describes the DataMatrix object. Default is ''.</td>
</tr>
<tr>
<td>RowNames</td>
<td>Empty array or cell array of character vectors that specifies the names for the rows, typically gene names or probe identifiers. The number of elements in the cell array must equal the number of rows in the matrix. Default is an empty array.</td>
</tr>
<tr>
<td>ColNames</td>
<td>Empty array or cell array of character vectors that specifies the names for the columns, typically sample identifiers. The number of elements in the cell array must equal the number of columns in the matrix.</td>
</tr>
<tr>
<td>NRows</td>
<td>Positive number that specifies the number of rows in the matrix.</td>
</tr>
<tr>
<td>NCols</td>
<td>Positive number that specifies the number of columns in the matrix.</td>
</tr>
<tr>
<td>NDims</td>
<td>Positive number that specifies the number of dimensions in the matrix.</td>
</tr>
<tr>
<td>ElementClass</td>
<td>Character vector that specifies the class type, such as single or double.</td>
</tr>
</tbody>
</table>

**Note** You cannot modify this property directly. You can access it using the `get` method.

Examples

1. Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: `yeastvalues`, a matrix of gene expression data, `genes`, a cell array of GenBank accession numbers for labeling the rows in `yeastvalues`, and `times`, a vector of time values for labeling the columns in `yeastvalues`.

```matlab
load filteredyeastdata
```

2. Import the microarray object package so that the `DataMatrix` constructor function will be available.

```matlab
import bioma.data.*
```

3. Create a DataMatrix object from the gene expression data in the first 30 rows of the `yeastvalues` matrix.

```matlab
dmo = DataMatrix(yeastvalues(1:30,:));
```
4 Use the `get` method to display the properties of the DataMatrix object, `dmo`.

    get(dmo)

    Name: '
    RowNames: []
    ColNames: []
    NRows: 30
    NCols: 7
    NDims: 2
    ElementClass: 'double'

    Notice that the `RowNames` and `ColNames` fields are empty.

5 Use the `set` method and the `genes` and `times` variables to specify row names and column names for the DataMatrix object, `dmo`.

    dmo = set(dmo,'RowNames',genes(1:30),'ColNames',times)

6 Use the `get` method to display the properties of the DataMatrix object, `dmo`.

    get(dmo)

    Name: '
    RowNames: {30x1 cell}
    ColNames: {' 0'  ' 9.5'  '11.5'  '13.5'  '15.5'  '18.5'  '20.5'}
    NRows: 30
    NCols: 7
    NDims: 2
    ElementClass: 'double'

See Also
- `DataMatrix` | `get`

Topics
- `DataMatrix` object on page 1-532

Introduced in R2008b
**setHeader**

Update header information of reads

**Syntax**

```
newObject = setHeader(object,headerInfo)
newObject = setHeader(object,headerInfo,subset)
```

**Description**

`newObject = setHeader(object,headerInfo)` returns a new object that is a copy of `object` with the `Header` property set to `headerInfo`.

`newObject = setHeader(object,headerInfo,subset)` returns a new object that is a copy of `object` with the `Header` property of a subset of elements set to `headerInfo`. A one-to-one relationship must exist between the number and order of elements in `headerInfo` and `subset`.

**Examples**

**Update Header Information**

Store read data from a SAM-formatted file in a BioRead object. Set 'InMemory' to true to load the object into memory so that you can modify its properties.

```matlab
br = BioRead('SRR005164_1_50.fastq','InMemory',true)
```

```
br =
   BioRead with properties:
     Quality: {50x1 cell}
     Sequence: {50x1 cell}
     Header: {50x1 cell}
     NSeqs: 50
     Name: ''
```

Check the header information for the first three elements of the object.

```matlab
br.Header(1:3)
```

```
ans = 3x1 cell
    {'SRR005164.1'}
    {'SRR005164.2'}
    {'SRR005164.3'}
```

Define custom headers for the first three elements.

```matlab
headers = {'Header1','Header2','Header3'};
```

Set the header information of the first three elements. `br2` is a copy of `br` with the headers updated. If you need to update the `br` object itself, set it as the output of the function.
br2 = setHeader(br,headers,[1:3]);
br2.Header(1:3)

ans = 3x1 cell
    {'Header1'}
    {'Header2'}
    {'Header3'}

You can also update the headers of the br object directly using dot notation.

br.Header(1:3) = headers;
br.Header(1:3)

ans = 3x1 cell
    {'Header1'}
    {'Header2'}
    {'Header3'}

Input Arguments

object — Object containing read data
    BioRead object | BioMap object

Object containing the read data, specified as a BioRead or BioMap object. If the object is not stored in memory, you cannot modify its properties, except the Name property.

Example: readData

headerInfo — Header information
    cell array of character vectors | string vector

Header information of the reads, specified as a cell array of character vectors or string vector.

Example: {'H1','H2','H3'}

subset — Subset of elements in object
    vector of positive integers | logical vector | string vector | cell array of character vectors

Subset of elements in the object, specified as a vector of positive integers, logical vector, string vector, or cell array of character vectors containing valid sequence headers.

Example: [1 3]

Tip When you use a sequence header (or a cell array of headers) for subset, a repeated header specifies all elements with that header.

Output Arguments

newObject — New object with updated properties
    BioRead object | BioMap object

New object with updated properties, returned as a BioRead or BioMap object.
See Also
BioMap | BioRead

Topics
“Manage Sequence Read Data in Objects”

Introduced in R2010a
**setQuality**

Update quality information

**Syntax**

```
newObject = setQuality(object,qualityInfo)
newObject = setQuality(object,qualityInfo,subset)
```

**Description**

`newObject = setQuality(object,qualityInfo)` returns a new object that is a copy of `object` with the `Quality` property set to `qualityInfo`.

```
newObject = setQuality(object,qualityInfo,subset) returns a new object that is a copy of 
object with the `Quality` property of a subset of elements set to `qualityInfo`. A one-to-one
relationship must exist between the number and order of elements in `qualityInfo` and `subset`.
```

**Examples**

**Update Quality Information**

Store read data from a SAM-formatted file in a BioRead object. Set 'InMemory' to true to load the
object into memory so that you can modify its properties.

```
br = BioRead('SRR005164_1_50.fastq','InMemory',true)
```

```
br =
   BioRead with properties:
      Quality: {50x1 cell}
      Sequence: {50x1 cell}
      Header: {50x1 cell}
      NSeqs: 50
      Name: '
```

Check the quality information for the first three elements of the object.

```
br.Quality(1:3)
```

```
ans = 3x1 cell
   {'<A<<@=+><'<<<<>8<>8<<<<;&<=7>8=9#<;<?9<<<<A;<<A<<<@:<<<<>7<?)<;A;'
   {'A<<<;<<<<@:<<<;;A;;A;@=*<<<<A>+<<<A;<<<<<<;;<;<@9<=<B<B?.@9<@:A>,<?]<<<<A<<B@81*"<<<<A
   {'<<<<<<;A;<<<<@=*:?9CA90(<6;9;<=;A<<<;B<<B@5&<<A;:A<<B@/A;<<;<<@;<<;<<<<;?9<<<<<<<<;<<:<<1<

Generate random quality scores for the first three reads. Assume that the quality scores are between
0 (equivalent to the ASCII code of 33) and 60 (equivalent to the ASCII code of 93).

```
qualities = cell(3,1);
rng('default');
```
for i = 1:3
    qualities{i} = char(floor((93 - 33) * rand(1,length(br.Quality{i}))) + 33);
end
qualities
qualities = 3x1 cell
    {'QW(WF&1AZZ*[Z>Q):WPZH#SYINM8H+K"1#&RJ4Z#:7NP,>,;GKN1IH*\(>Z5D.N0?JVZA)0S0'
    {'Q/X5,0E=6RDAXZNN7%@@QY(C=15*P3@E0HJM<.&.W*RA\%;'Z!0RU&80Q:W+0))UCA)TF6?9%/(/,:#WY>>5W7'08/
    {'?RPG7Q@6YUBFD-3=.S,+.3X;W[;"9D0EK.(24:7&0Q"XL>C/<ZA.@.F187\#VWP&051,L'H>OKWVSJ,"M?=WEF'...

Update the quality information of the first three elements. br2 is a copy of br with the updated quality scores. If you need to update the br object itself, set it as the output of the function.

br2 = setQuality(br,qualities,[1:3]);
br2.Quality(1:3)
ans = 3x1 cell
    {'QW(WF&1AZZ*[Z>Q):WPZH#SYINM8H+K"1#&RJ4Z#:7NP,>,;GKN1IH*\(>Z5D.N0?JVZA)0S0'
    {'Q/X5,0E=6RDAXZNN7%@@QY(C=15*P3@E0HJM<.&.W*RA\%;'Z!0RU&80Q:W+0))UCA)TF6?9%/(/,:#WY>>5W7'08/
    {'?RPG7Q@6YUBFD-3=.S,+.3X;W[;"9D0EK.(24:7&0Q"XL>C/<ZA.@.F187\#VWP&051,L'H>OKWVSJ,"M?=WEF'...

You can update the quality scores of the br object directly by using dot notation.

br.Quality(1:3) = qualities;
br.Quality(1:3)
ans = 3x1 cell
    {'QW(WF&1AZZ*[Z>Q):WPZH#SYINM8H+K"1#&RJ4Z#:7NP,>,;GKN1IH*\(>Z5D.N0?JVZA)0S0'
    {'Q/X5,0E=6RDAXZNN7%@@QY(C=15*P3@E0HJM<.&.W*RA\%;'Z!0RU&80Q:W+0))UCA)TF6?9%/(/,:#WY>>5W7'08/
    {'?RPG7Q@6YUBFD-3=.S,+.3X;W[;"9D0EK.(24:7&0Q"XL>C/<ZA.@.F187\#VWP&051,L'H>OKWVSJ,"M?=WEF'...

**Input Arguments**

**object** — Object containing read data  
BioRead object | BioMap object

Object containing the read data, specified as a BioRead or BioMap object. If the object is not stored in memory, you cannot modify its properties, except the Name property.

Example: readData

**qualityInfo** — Quality information  
cell array of character vectors

Quality information of reads, specified as a cell array of character vectors or string vector.

Example: 
    {'<A<<@=+>','A<<<;<<'}

**subset** — Subset of elements in object  

vector of positive integers | logical vector | string vector | cell array of character vectors

Subset of elements in the object, specified as a vector of positive integers, logical vector, string vector, or cell array of character vectors containing valid sequence headers.

1-1587
Example: [1 3]

**Tip** When you use a sequence header (or a cell array of headers) for `subset`, a repeated header specifies all elements with that header.

**Output Arguments**

`newObject` — New object with updated properties  
`BioRead` object | `BioMap` object

New object with updated properties, returned as a `BioRead` or `BioMap` object.

**See Also**

`BioMap` | `BioRead`

**Topics**

“Manage Sequence Read Data in Objects”

**Introduced in R2010a**
setSequence

Update read sequences

Syntax

newObject = setSequence(object,sequenceInfo)
newObject = setSequence(object,sequenceInfo,subset)

Description

newObject = setSequence(object,sequenceInfo) returns a new object that is a copy of object with the Sequence property set to sequenceInfo.

newObject = setSequence(object,sequenceInfo,subset) returns a new object that is a copy of object with the Sequence property of a subset of elements set to sequenceInfo. A one-to-one relationship must exist between the number and order of elements in sequenceInfo and subset.

Examples

Update Read Sequences

Store read data from a SAM-formatted file in a BioRead object. Set 'InMemory' to true to load the object into memory so that you can modify its properties.

br = BioRead('SRR005164_1_50.fastq','InMemory',true)

br =
    BioRead with properties:
        Quality: [50x1 cell]
        Sequence: [50x1 cell]
        Header: [50x1 cell]
        NSeqs: 50
        Name: '

Check the read sequences of the first three elements of the object.

br.Sequence(1:3)

ans = 3x1 cell
    {'TGGCTTTAAAGCAGAACTTGTGAAAGAAGGAAAGCATTATGATTATCTGGCTAAGCTTAGCATTGTTTAGAA'}
    {'TTACACTATCCTCTGATTACCAAAGACGTTTCTCGGTCATACAGACAGTCCTTGAGCAAGGGAAGAATTTATTTGCAGGCAAAAAAGTGTCCAACCGTATCGTGAGTATCGACCGGCATTACCTT'}
    {'CACGAGCGGTATATTTGCCTTTTTGTGCTGTGATTCGATTCTTTTCTCTCCTCCACCCCAAGCGAGCT TTGCTCAGAAAGTGAGTACGCTTC'}

Generate random sequences for the first three reads. Use the randseq function to generate random sequences with the same length as the original sequences.

sequenceInfo = cell(3,1);
rng('default');
for i = 1:3
    sequenceInfo{i} = randseq(length(br.Quality{i}));
end

sequenceInfo

sequenceInfo = 3x1 cell
    { 'TTATGACGTTATTTCTACTTTGATTGTCGAGACATGCTACCTACCAGTGCTCGGAACTCAGCTCGGGTTAACC'
    { 'TATCACGCTGGCTGGTCTTCACTAGACATCGAGCGGGATATTGACATATGTACATATTTACCTCTACAATGGATGCGCAAAAACATTCCCTCATCACAATTGAACTAAAGGGCGCGAGACGTATTCCCCG'
    { 'GTTGCTGCTGTTGGGACCATAAAACCTCATACGGCGGAACCGGCGACTATCGGAACGGCCCTAAAATTTACCGGAGAGCTGTTCGAAGGCTGGTTGAATACATGGCAGAAGATTGAGGTGTCCTAAACTTACCGGCCATAACACCTAGCCGTCTCGGGGGAATAAGTGACCTAT'

Update the sequences of the first three elements. br2 is a copy of br with updated read sequences. If you need to update the br object itself, set it as the output of the function.

br2 = setSequence(br,sequenceInfo,[1:3]);
br2.Sequence(1:3)
ans = 3x1 cell
    { 'TTATGACGTTATTTCTACTTTGATTGTCGAGACATGCTACCTACCAGTGCTCGGAACTCAGCTCGGGTTAACC'
    { 'TATCACGCTGGCTGGTCTTCACTAGACATCGAGCGGGATATTGACATATGTACATATTTACCTCTACAATGGATGCGCAAAAACATTCCCTCATCACAATTGAACTAAAGGGCGCGAGACGTATTCCCCG'
    { 'GTTGCTGCTGTTGGGACCATAAAACCTCATACGGCGGAACCGGCGACTATCGGAACGGCCCTAAAATTTACCGGAGAGCTGTTCGAAGGCTGGTTGAATACATGGCAGAAGATTGAGGTGTCCTAAACTTACCGGCCATAACACCTAGCCGTCTCGGGGGAATAAGTGACCTAT'

You can also update the sequences of the br object directly using dot notation.

br.Sequence(1:3) = sequenceInfo;
br.Sequence(1:3)
ans = 3x1 cell
    { 'TTATGACGTTATTTCTACTTTGATTGTCGAGACATGCTACCTACCAGTGCTCGGAACTCAGCTCGGGTTAACC'
    { 'TATCACGCTGGCTGGTCTTCACTAGACATCGAGCGGGATATTGACATATGTACATATTTACCTCTACAATGGATGCGCAAAAACATTCCCTCATCACAATTGAACTAAAGGGCGCGAGACGTATTCCCCG'
    { 'GTTGCTGCTGTTGGGACCATAAAACCTCATACGGCGGAACCGGCGACTATCGGAACGGCCCTAAAATTTACCGGAGAGCTGTTCGAAGGCTGGTTGAATACATGGCAGAAGATTGAGGTGTCCTAAACTTACCGGCCATAACACCTAGCCGTCTCGGGGGAATAAGTGACCTAT'

Input Arguments

object — Object containing read data
         BioRead object | BioMap object

Object containing the read data, specified as a BioRead or BioMap object. If the object is not stored in memory, you cannot modify its properties, except the Name property.

Example: readData

sequenceInfo — Read sequences
cell array of character vectors | string vector

Read sequences, specified as a cell array of character vectors or string vector containing nucleotide sequences.

Example: {'TGCTTGC','AAAGACGTACG'}

subset — Subset of elements in object
         vector of positive integers | logical vector | string vector | cell array of character vectors

Subset of elements in the object, specified as a vector of positive integers, logical vector, string vector, or cell array of character vectors containing valid sequence headers.
Example: [1 3]

**Tip** When you use a sequence header (or a cell array of headers) for `subset`, a repeated header specifies all elements with that header.

### Output Arguments

- **newObject** — New object with updated properties  
  BioRead object | BioMap object  
  
  New object with updated properties, returned as a BioRead or BioMap object.

### See Also

- BioMap | BioRead

### Topics

- “Manage Sequence Read Data in Objects”

### Introduced in R2010a
setSubsequence

Update partial sequences

Syntax

newObject = setSubsequence(object,subsequences,subset,positions)

Description

newObject = setSubsequence(object,subsequences,subset,positions) returns a new object that is a copy of object with the partial sequences of a subset of elements set to subsequences. The positions argument specifies the sequence positions to be updated by subsequences. A one-to-one relationship must exist between the number and order of elements in subsequences and subset.

Examples

Update Partial Sequences

Store read data from a SAM-formatted file in a BioRead object. Set 'InMemory' to true to load the object into memory so that you can modify its properties.

br = BioRead('SRR005164_1_50.fastq','InMemory',true)

br =

BioRead with properties:

    Quality: {50x1 cell}
    Sequence: {50x1 cell}
    Header: {50x1 cell}
    NSeqs: 50
    Name: ''

Assume that you want to update the sequences of the first two reads partially (for example, the first five positions). First check the existing sequences.

br.Sequence(1:2)

ans = 2x1 cell

    {'TGGCTTTAAAGCAAGAACTTGTGAAAGAAGGAAAGCATTATGATTATCTGGCTAAGCTTAGCATTGTTTAGAA'}
    {'TTACACTATCTGATTACCAAAGACGTTTCTCGGTCATACAGACAGTCCTTGAGCAAGGGAAGAATTTATTTGCAGGCAAAAAAGTGTCCAACCGTATCGTGAGTATCGACCGGCATTACCTT'}

Define the subsequences. Each subsequence must have the same length.

subSequences = {'ATTCG','TACTA'}

subSequences = 1x2 cell

    {'ATTCG'}    {'TACTA'}
Update the first five positions of the first two reads. The number of positions must equal the length of each subsequence. In this example, the total number of positions is five, as is the length of each subsequence. br2 is a copy of br with updated read sequences. If you need to update the br object itself, set it as the output of the function.

```matlab
positions   = [1:5];
subset      = [1 2];
br2         = setSubsequence(br,subSequences,subset,positions);
br2.Sequence(1:2)
```

ans = 2x1 cell

```
{'ATTCGTAAAAGCAGAACTTGTGAAGAAGGAAGCAATTATGATTATCTGGCTAAGCTTAGCATTGTTAGAA'
{'TACTACTATCCTCTGATTACCAAAGACGTTTCTCGGTCATACAGACAGTCCTTGAGCAAGGAAGAATTTATTTGCAGGCAAAAAAGTGTCCAACCGTATCGTGAGTATCGACCGGCATTACCTT'
```

### Input Arguments

**object** — Object containing read data  
`BioRead object | BioMap object`

Object containing the read data, specified as a `BioRead` or `BioMap` object. If the object is not stored in memory, you cannot modify its properties, except the `Name` property.

Example: `readData`

**subsequences** — Partial read sequences  
`cell array of character vectors | string vector`

Partial read sequences, specified as a cell array of character vectors or string vector. Each character vector or string (that is, each sequence) must be the same length.

Example: `{'TGGCTTC','AAAGCAG'}`

**subset** — Subset of elements in object  
`vector of positive integers | logical vector | string vector | cell array of character vectors`

Subset of elements in the object, specified as a vector of positive integers, logical vector, string vector, or cell array of character vectors containing valid sequence headers.

Example: `[1 3]`

**Tip**  
When you use a sequence header (or a cell array of headers) for `subset`, a repeated header specifies all elements with that header.

**positions** — Sequence positions  
`vector of positive integers | logical vector`

Sequence positions, specified as a vector of positive integers or logical vector. The number of positions must equal the length of every character vector or string in `subsequences`.

Example: `[1:5]`
Output Arguments

newObject — New object with updated properties
BioRead object | BioMap object

New object with updated properties, returned as a BioRead or BioMap object.

See Also
BioMap | BioRead

Topics
“Manage Sequence Read Data in Objects”

Introduced in R2010a
**setSubset**

Update elements of object

**Syntax**

$newObject = setSubset(object,elements,subset)$

**Description**

$newObject = setSubset(object,elements,subset)$ returns a new object that is a copy of object with a subset of elements set to elements. A one-to-one relationship must exist between the number and order of elements in elements and subset.

**Examples**

**Update Elements of BioRead Object**

Construct two BioRead objects, one with 10 elements, and one with 2 elements. Trim the headers to the first white space.

```matlab
struct1 = fastqread('SRR005164_1_50.fastq',... 'blockread', [1 10], 'trimheaders', true);
struct2 = fastqread('SRR005164_1_50.fastq',... 'blockread', [11 12], 'trimheaders', true);
brObj1  = BioRead(struct1)
brObj1 = BioRead with properties:
    Quality: {10x1 cell}
    Sequence: {10x1 cell}
    Header: {10x1 cell}
    NSeqs: 10
    Name: ''

brObj2  = BioRead(struct2)
brObj2 = BioRead with properties:
    Quality: {2x1 cell}
    Sequence: {2x1 cell}
    Header: {2x1 cell}
    NSeqs: 2
    Name: ''
```

Replace the first two elements in brObj1 with the elements in brObj2. The object brObj2 must contain the same number of elements as the number of elements in subset (in this case, 2).
subset = [1:2];
brObj1 = setSubset(brObj1,brObj2,subset)

brObj1 = BioRead with properties:
    Quality: {10x1 cell}
    Sequence: {10x1 cell}
    Header: {10x1 cell}
    NSeqs: 10
    Name: ''

Input Arguments

object — Object containing read data
BioRead object | BioMap object

Object containing the read data, specified as a BioRead or BioMap object. If the object is not stored in memory, you cannot modify its properties, except the Name property.
Example: readData

elements — Object containing information related to read data
BioRead object | BioMap object

Object containing information related to the read data, specified as a BioRead or BioMap object. The object must contain the same number of elements as the number of elements in subset.
Example: brObject

subset — Subset of elements in object
vector of positive integers | logical vector | string vector | cell array of character vectors

Subset of elements in the object, specified as a vector of positive integers, logical vector, string vector, or cell array of character vectors containing valid sequence headers.
Example: [1 3]

Tip When you use a sequence header (or a cell array of headers) for subset, a repeated header specifies all elements with that header.

Output Arguments

newObject — New object with updated properties
BioRead object | BioMap object

New object with updated properties, returned as a BioRead or BioMap object.

See Also
BioMap | BioRead
Topics
"Manage Sequence Read Data in Objects"

Introduced in R2010a
sffinfo

Return information about SFF file

Syntax

\[ \text{InfoStruct} = \text{sffinfo}(\text{File}) \]

Description

\[ \text{InfoStruct} = \text{sffinfo}(\text{File}) \] returns a MATLAB structure containing summary information about a Standard Flowgram Format (SFF) file.

Input Arguments

File

Character vector or string specifying a file name or path and file name of an SFF file produced by version 1.0 of the Genome Sequencer System data analysis software from 454 Life Sciences®. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.

Output Arguments

InfoStruct

MATLAB structure containing summary information about an SFF file. The structure contains the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>Name of the file.</td>
</tr>
<tr>
<td>FileModDate</td>
<td>Modification date of the file.</td>
</tr>
<tr>
<td>FileSize</td>
<td>Size of the file in bytes.</td>
</tr>
<tr>
<td>Version</td>
<td>Version number of the file.</td>
</tr>
<tr>
<td>FlowgramCode</td>
<td>Code of the format used to encode flowgram values.</td>
</tr>
<tr>
<td>NumberOfReads</td>
<td>Number of sequence reads in the file.</td>
</tr>
<tr>
<td>NumberOfFlowsPerRead</td>
<td>Number of flows for each read.</td>
</tr>
<tr>
<td>FlowChars</td>
<td>Bases used in each flow.</td>
</tr>
<tr>
<td>KeySequence</td>
<td>Character vector of bases in the key sequence.</td>
</tr>
</tbody>
</table>

Examples

The SFF file, SRR013472.sff, used in this example is not provided with the Bioinformatics Toolbox software. You can download sample SFF files from:


Return a summary of the contents of an SFF file:
info = sffinfo('SRR013472.sff')

info =

    Filename: 'SRR013472.sff'
    FileModDate: '23-Feb-2009 15:14:36'
    FileSize: 6632392
    Version: [0 0 0 1]
    FlowgramCode: 1
    NumberOfReads: 3546
    NumberOfFlowsPerRead: 440
    FlowChars: [1x440 char]
    KeySequence: 'TCAG'

See Also
fastainfo | fastaread | fastawrite | fastqinfo | fastqread | fastqwrite | saminfo | samread | sffread

External Websites

Introduced in R2009b
sffread

Read data from SFF file

Syntax

SFFStruct = sffread(File)

sffread(..., 'Blockread', BlockreadValue, ...)

sffread(..., 'Feature', FeatureValue, ...)

Description

SFFStruct = sffread(File) reads a Standard Flowgram Format (SFF) file and returns the data in a MATLAB array of structures.

sffread(..., 'PropertyName', PropertyValue, ...) calls sffread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each PropertyName in single quotation marks. Each PropertyName is case insensitive. These property name/property value pairs are as follows:

sffread(..., 'Blockread', BlockreadValue, ...) reads a single sequence entry or block of sequence entries from an SFF file containing multiple sequences.

sffread(..., 'Feature', FeatureValue, ...) specifies the information to include in the return structure.

Input Arguments

File

Character vector or string specifying a file name or path and file name of an SFF file produced by version 1.0 of the Genome Sequencer System data analysis software from 454 Life Sciences. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.

Default:

BlockreadValue

Scalar or vector that controls the reading of a single sequence entry or block of sequence entries from an SFF file containing multiple sequences. Enter a scalar N, to read the Nth entry in the file. Enter a 1-by-2 vector [M1, M2], to read a block of entries starting at the M1 entry and ending at the M2 entry. To read all remaining entries in the file starting at the M1 entry, enter a positive value for M1 and enter Inf for M2.

Default:

FeatureValue

Character vector or string specifying the information to include in the output structure. The character vector or string includes letters from the alphabet H, S, Q, C, F, and I, which represent the
fields Header, Sequence, Quality, Clipping, FlowgramValue, and FlowgramIndex, respectively.

**Default:** 'HSQ'

## Output Arguments

**SFFStruct**

Array of structures containing information from an SFF file. There is one structure for each read or entry in the file. Each structure contains one or more of the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Universal accession number.</td>
</tr>
<tr>
<td>Sequence</td>
<td>Numeric representation of nucleotide sequence.</td>
</tr>
<tr>
<td>Quality</td>
<td>Per-base quality scores.</td>
</tr>
<tr>
<td>Clipping</td>
<td>Clipping boundary positions.</td>
</tr>
<tr>
<td>FlowgramValue</td>
<td>Sequence of flowgram intensity values.</td>
</tr>
<tr>
<td>FlowgramIndex</td>
<td>Sequence of flowgram intensity indices.</td>
</tr>
</tbody>
</table>

### Examples

The SFF file, `SRR013472.sff`, used in these examples is not provided with the Bioinformatics Toolbox software. You can download sample SFF files from:


Read an entire SFF file:

```matlab
% Read the contents of an entire SFF file into an
% array of structures
reads = sffread('SRR013472.sff')

reads =

3546x1 struct array with fields:
   Header
   Sequence
   Quality
```

Read a block of entries from an SFF file:

```matlab
% Read only the header and sequence information of the
% first five reads from an SFF file into an array of structures
reads5 = sffread('SRR013472.sff', 'block', [1 5], 'feature', 'hs')

reads5 =

5x1 struct array with fields:
   Header
   Sequence
```
See Also
fastainfo | fastaread | fastawrite | fastqinfo | fastqread | fastqwrite | saminfo | samread | sffinfo

External Websites

Introduced in R2009b
**shortestpath (biograph)**

Solve shortest path problem in biograph object

**Syntax**

```matlab
[dist, path, pred] = shortestpath(BGObj, S)
[dist, path, pred] = shortestpath(BGObj, S, T)
```

```matlab
[...] = shortestpath(..., 'Directed', DirectedValue, ...)
[...] = shortestpath(..., 'Method', MethodValue, ...)
[...] = shortestpath(..., 'Weights', WeightsValue, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BGObj</strong></td>
<td>Biograph object created by biograph (object constructor).</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td>Node in graph represented by an N-by-N adjacency matrix extracted from a biograph object, <strong>BGObj</strong>.</td>
</tr>
<tr>
<td><strong>T</strong></td>
<td>Node in graph represented by an N-by-N adjacency matrix extracted from a biograph object, <strong>BGObj</strong>.</td>
</tr>
<tr>
<td><strong>DirectedValue</strong></td>
<td>Property that indicates whether the graph represented by the N-by-N adjacency matrix extracted from a biograph object, <strong>BGObj</strong>, is directed or undirected. Enter false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.</td>
</tr>
<tr>
<td><strong>MethodValue</strong></td>
<td>Character vector or string that specifies the algorithm used to find the shortest path. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'Bellman-Ford' — Assumes weights of the edges to be nonzero entries in the N-by-N adjacency matrix. Time complexity is (O(N^2E)), where (N) and (E) are the number of nodes and edges respectively.</td>
</tr>
<tr>
<td></td>
<td>• 'BFS' — Breadth-first search. Assumes all weights to be equal, and nonzero entries in the N-by-N adjacency matrix to represent edges. Time complexity is (O(N+E)), where (N) and (E) are the number of nodes and edges respectively.</td>
</tr>
<tr>
<td></td>
<td>• 'Acyclic' — Assumes the graph represented by the N-by-N adjacency matrix extracted from a biograph object, <strong>BGObj</strong>, to be a directed acyclic graph and that weights of the edges are nonzero entries in the N-by-N adjacency matrix. Time complexity is (O(N+E)), where (N) and (E) are the number of nodes and edges respectively.</td>
</tr>
<tr>
<td></td>
<td>• 'Dijkstra' — Default algorithm. Assumes weights of the edges to be positive values in the N-by-N adjacency matrix. Time complexity is (O(\log(N)\cdot E)), where (N) and (E) are the number of nodes and edges respectively.</td>
</tr>
</tbody>
</table>
WeightsValue | Column vector that specifies custom weights for the edges in the N-by-N adjacency matrix extracted from a biograph object, BGObj. It must have one entry for every nonzero value (edge) in the N-by-N adjacency matrix. The order of the custom weights in the vector must match the order of the nonzero values in the N-by-N adjacency matrix when it is traversed columnwise. This property lets you use zero-valued weights. By default, shortestpaths gets weight information from the nonzero entries in the N-by-N adjacency matrix.

**Description**

**Tip** For introductory information on graph theory functions, see “Graph Theory Functions”.

\[ \text{[dist, path, pred]} = \text{shortestpath(BGObj, S)} \] determines the single-source shortest paths from node \( S \) to all other nodes in the graph represented by an N-by-N adjacency matrix extracted from a biograph object, BGObj. Weights of the edges are all nonzero entries in the N-by-N adjacency matrix. \( \text{dist} \) are the \( N \) distances from the source to every node (using \( \text{Inf} \)s for nonreachable nodes and \( 0 \) for the source node). \( \text{path} \) contains the winning paths to every node. \( \text{pred} \) contains the predecessor nodes of the winning paths.

\[ \text{[dist, path, pred]} = \text{shortestpath(BGObj, S, T)} \] determines the single source-single destination shortest path from node \( S \) to node \( T \).

\[ [...] = \text{shortestpath(..., 'PropertyName', PropertyValue, ...)} \] calls shortestpath with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each \( \text{PropertyName} \) must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

\[ [...] = \text{shortestpath(..., 'Directed', DirectedValue, ...)} \] indicates whether the graph represented by the N-by-N adjacency matrix extracted from a biograph object, BGObj, is directed or undirected. Set \( \text{DirectedValue} \) to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.

\[ [...] = \text{shortestpath(..., 'Method', MethodValue, ...)} \] lets you specify the algorithm used to find the shortest path. Choices are:

- 'Bellman-Ford' — Assumes weights of the edges to be nonzero entries in the N-by-N adjacency matrix. Time complexity is \( O(N*E) \), where \( N \) and \( E \) are the number of nodes and edges respectively.
- 'BFS' — Breadth-first search. Assumes all weights to be equal, and nonzero entries in the N-by-N adjacency matrix to represent edges. Time complexity is \( O(N+E) \), where \( N \) and \( E \) are the number of nodes and edges respectively.
- 'Acyclic' — Assumes the graph represented by the N-by-N adjacency matrix extracted from a biograph object, BGObj, to be a directed acyclic graph and that weights of the edges are nonzero entries in the N-by-N adjacency matrix. Time complexity is \( O(N+E) \), where \( N \) and \( E \) are the number of nodes and edges respectively.
- 'Dijkstra' — Default algorithm. Assumes weights of the edges to be positive values in the N-by-N adjacency matrix. Time complexity is \( O(\log(N)*E) \), where \( N \) and \( E \) are the number of nodes and edges respectively.
[...] = shortestpath(..., 'Weights', WeightsValue, ...) lets you specify custom weights for the edges. WeightsValue is a column vector having one entry for every nonzero value (edge) in the N-by-N adjacency matrix extracted from a biograph object, BGObj. The order of the custom weights in the vector must match the order of the nonzero values in the N-by-N adjacency matrix when it is traversed column-wise. This property lets you use zero-valued weights. By default, shortestpath gets weight information from the nonzero entries in the N-by-N adjacency matrix.

References


See Also
allshortestpaths | biograph | conncomp | graphshortestpath | isdag | isomorphism | isspantree | maxflow | minspantree | topoorder | traverse

Topics
biograph object on page 1-185

Introduced in R2006b
showalignment

(To be removed) Display color-coded sequence alignment

Note showalignment will be removed in a future release.

Syntax

showalignment(Alignment)

showalignment(..., 'MatchColor', MatchColorValue, ...)
showalignment(..., 'SimilarColor', SimilarColorValue, ...)
showalignment(..., 'StartPointers', StartPointersValue, ...)
showalignment(..., 'Columns', ColumnsValue, ...)
showalignment(..., 'TerminalGap', TerminalGapValue, ...)

Description

showalignment(Alignment) displays a color-coded sequence alignment in a MATLAB Figure window.

showalignment(..., 'PropertyName', PropertyValue, ...) calls showalignment with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each PropertyName in single quotation marks. Each PropertyName is case insensitive. These property name/property value pairs are as follows:

showalignment(..., 'MatchColor', MatchColorValue, ...) specifies the color to highlight matching characters in the output display.

showalignment(..., 'SimilarColor', SimilarColorValue, ...) specifies the color to highlight similar characters in the output display.

showalignment(..., 'StartPointers', StartPointersValue, ...) specifies the starting indices in the original sequences of a local pairwise alignment.

showalignment(..., 'Columns', ColumnsValue, ...) specifies the number of characters to display in one row when displaying a pairwise alignment, and labels the start of each row with the sequence positions.

showalignment(..., 'TerminalGap', TerminalGapValue, ...) controls the inclusion or exclusion of terminal gaps from the count of matches and similar residues when displaying a pairwise alignment. TerminalGapValue can be true (default) or false.

Input Arguments

Alignment

Pairwise or multiple sequence alignment specified by one of the following:
3-by-N character array showing the pairwise alignment of two sequences, such as returned by `nwalign` or `swalign`

MATLAB structure containing a `Sequence` field, such as returned by `fastaread`, `gethmalignment`, `multialign`, or `multialignread`

Character array or string vector that contains a multiple sequence alignment, such as returned by `multialign`

**MatchColorValue**

Color to highlight matching characters in the output display. Specify the color with one of the following:

- Three-element numeric vector of RGB values
- Character vector or string containing a predefined single-letter color code
- Character vector or string containing a predefined color name

For example, to use cyan, enter `[0 1 1]`, `'c'`, or `'cyan'`. For more information on specifying colors, see “Color Options” on page 1-1609.

**Default:** Red, which is specified by `[1 0 0]`, `'r'`, or `'red'`

**SimilarColorValue**

Color to highlight similar characters in the output display. Specify the color with one of the following:

- Three-element numeric vector of RGB values
- Character vector or string containing a predefined single-letter color code
- Character vector or string containing a predefined color name

For example, to use cyan, enter `[0 1 1]`, `'c'`, or `'cyan'`. For more information on specifying colors, see “Color Options” on page 1-1609.

**Default:** Magenta, which is specified by `[1 0 1]`, `'m'`, or `'magenta'`

**StartPointersValue**

Two-element vector that specifies the starting indices in the original sequences of a local pairwise alignment.

**Tip** You can use the third output returned by `swalign` as the `StartPointersValue`.

**Default:**

**ColumnsValue**

Scalar that specifies the number of characters to display in one row when displaying a pairwise alignment.

**Default:** 64
**TerminalGapValue**

Specifies whether to include or exclude terminal gaps from the count of matches and similar residues when displaying a pairwise alignment. Choices are **true** (default) or **false**.

**Default:**

**Examples**

Display a pairwise sequence alignment:

```matlab
% Globally align two amino acid sequences
[Score, Alignment] = nwalign('VSPAGMASGYD', 'IPGKASYD');
% Display the color-coded alignment
showalignment(Alignment);
```

Notice that for pairwise sequence alignments, matching and similar characters appear in red and magenta respectively.

Display a multiple sequence alignment

```matlab
% Read a multiple-sequence alignment file
gag = multialignread('aagag.aln');
% Display the color-coded alignment
showalignment(gag)
```
Notice that for multiple sequence alignments, highly conserved positions appear in red and conserved positions appear in magenta.

Tip To view a multiple-sequence alignment and interact with it, use the `seqalignviewer` function.

More About

Color Options

The following lists the predefined colors and their RGB triplet equivalents. The short names and long names are character vectors that specify one of eight preset colors. The RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color; the intensities must be in the range [0 1].

<table>
<thead>
<tr>
<th>RGB Triplet</th>
<th>Short Name</th>
<th>Long Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1 1 0]</td>
<td>y</td>
<td>yellow</td>
</tr>
<tr>
<td>[1 0 1]</td>
<td>m</td>
<td>magenta</td>
</tr>
<tr>
<td>[0 1 1]</td>
<td>c</td>
<td>cyan</td>
</tr>
<tr>
<td>[1 0 0]</td>
<td>r</td>
<td>red</td>
</tr>
<tr>
<td>[0 1 0]</td>
<td>g</td>
<td>green</td>
</tr>
<tr>
<td>[0 0 1]</td>
<td>b</td>
<td>blue</td>
</tr>
<tr>
<td>[1 1 1]</td>
<td>w</td>
<td>white</td>
</tr>
<tr>
<td>[0 0 0]</td>
<td>k</td>
<td>black</td>
</tr>
</tbody>
</table>
Alternatives

You can also display a multiple or pairwise sequence alignment using the seqalignviewer function. The alignment displays in the Biological Sequence Alignment window, where you can view and interactively adjust a sequence alignment.

Compatibility Considerations

showalignment will be removed
Not recommended starting in R2020a

showalignment will be removed in a future release.

See Also

blosum | dayhoff | fastaread | gethmmalignment | gonnet | localalign | multialign |
multialignread | nuc44 | nalign | pam | seqalignviewer | showalignment | swalign

Topics

“Aligning Pairs of Sequences”
“Retrieve Sequence Information from a Public Database”
“View and Align Multiple Sequences”
Amino Acid Lookup
Nucleotide Lookup

External Websites

https://www.rcsb.org/pdb/home/home.do
https://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?cmd=show&f=main&m=main&s=main

Introduced before R2006a
showhmmprof

Plot hidden Markov model (HMM) profile

Syntax

showhmmprof(Model)

showhmmprof(Model, ...'Scale', ScaleValue, ...)

showhmmprof(Model, ...'Order', OrderValue, ...)

Arguments

<table>
<thead>
<tr>
<th>Model</th>
<th>Hidden Markov model created by the function gethmmprof or pfamhmread.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScaleValue</td>
<td>Property to select a probability scale. Enter one of the following values:</td>
</tr>
<tr>
<td>OrderValue</td>
<td>Property to specify the order of the amino acid alphabet. Enter a character vector or string with the 20 standard amino acids characters A R N D C Q E G H I L K M F P S T W Y V. The ambiguous characters B Z X are not allowed.</td>
</tr>
</tbody>
</table>

Description

showhmmprof(Model) plots a profile hidden Markov model described by the structure Model.

showhmmprof(..., 'PropertyName', PropertyValue, ...) calls showhmmprof with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

showhmmprof(Model, ...'Scale', ScaleValue, ...) specifies the scale to use. If log probabilities (ScaleValue='logprob'), probabilities (ScaleValue='prob'), or log-odd ratios (ScaleValue='logodds'). To compute the log-odd ratios, the null model probabilities are used for symbol emission and equally distributed transitions are used for the null transition probabilities. The default ScaleValue is 'logprob'.

showhmmprof(Model, ...'Order', OrderValue, ...) specifies the order in which the symbols are arranged along the vertical axis. This option allows you reorder the alphabet and group the symbols according to their properties.

Examples

1 Load a model example.

    model = pfamhmread('pf00002.ls');
2 Plot the profile.

```matlab
showhmmprof(model, 'Scale', 'logodds')
```
3 Order the alphabet by hydrophobicity.

```matlab```
hydrophobic = 'IVLFCMAGTSWYPHNDQEKR';
```

4 Plot the profile.

```matlab```
showhmmpref(model, 'Order', hydrophobic)
```

See Also
gethmmprof | hmmproalign | hmmprofestimate | hmmprofgenerate | hmmprofstruct | pfamhmmread

Introduced before R2006a
single (DataMatrix)

Convert DataMatrix object to single-precision array

**Syntax**

\[
B = \text{single}(\text{DMObj}) \\
B = \text{single}(\text{DMObj}, \text{Rows}) \\
B = \text{single}(\text{DMObj}, \text{Rows}, \text{Cols})
\]

**Input Arguments**

<table>
<thead>
<tr>
<th>( \text{DMObj} )</th>
<th>DataMatrix object, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Rows, Cols} )</td>
<td>Row(s) or column(s) in ( \text{DMObj} ), specified by one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Scalar</td>
</tr>
<tr>
<td></td>
<td>• Vector of positive integers</td>
</tr>
<tr>
<td></td>
<td>• Character vector specifying a row or column name</td>
</tr>
<tr>
<td></td>
<td>• Cell array of row or column names</td>
</tr>
<tr>
<td></td>
<td>• Logical vector</td>
</tr>
</tbody>
</table>

**Output Arguments**

| \( B \) | MATLAB numeric array. |

**Description**

\( B = \text{single}(\text{DMObj}) \) converts \( \text{DMObj} \), a DataMatrix object, to a single-precision array, which it returns in \( B \).

\( B = \text{single}(\text{DMObj}, \text{Rows}) \) converts a subset of \( \text{DMObj} \), a DataMatrix object, specified by \( \text{Rows} \), to a single-precision array, which it returns in \( B \). \( \text{Rows} \) can be a positive integer, vector of positive integers, character vector specifying a row name, cell array of row names, or a logical vector.

\( B = \text{single}(\text{DMObj}, \text{Rows}, \text{Cols}) \) converts a subset of \( \text{DMObj} \), a DataMatrix object, specified by \( \text{Rows} \) and \( \text{Cols} \), to a single-precision array, which it returns in \( B \). \( \text{Cols} \) can be a positive integer, vector of positive integers, character vector specifying a column name, cell array of column names, or a logical vector.

**See Also**

DataMatrix | double

**Topics**

DataMatrix object on page 1-532

**Introduced in R2008b**
size

Class: bioma.ExpressionSet
Package: bioma

Return size of ExpressionSet object

Syntax

NFeatSam = size(ESObj)
[NFeatures, NSamples] = size(ESObj)
DimLength = size(ESObj, Dim)

Description

NFeatSam = size(ESObj) returns a two-element row vector containing the number of features and number of samples in an ExpressionSet object.

[NFeatures, NSamples] = size(ESObj) returns the number of features and number of samples in an ExpressionSet object as separate variables.

DimLength = size(ESObj, Dim) returns the length of the dimension specified by Dim.

Input Arguments

ESObj

Object of the bioma.ExpressionSet class.

Default:

Dim

Scalar specifying the dimension of the ExpressionSet object. Choices are:

- 1 — Features
- 2 — Samples

Default:

Examples

Construct an ExpressionSet object, ESObj, as described in the “Examples” on page 1-0 section of the bioma.ExpressionSet class reference page. Determine the number of features and samples in the ExpressionSet object:

% Retrieve the number of features and samples
NumFeatSam = size(ESObj)

See Also

DataMatrix | bioma.ExpressionSet | bioma.data.ExptData
Topics
"Managing Gene Expression Data in Objects"
size

Class: bioma.data.ExptData
Package: bioma.data

Return size of ExptData object

Syntax

\[ N\text{FeatSam} = \text{size}(\text{EDObj}) \]
\[ [\text{NFeatures}, \text{NSamples}] = \text{size}(\text{EDObj}) \]
\[ \text{DimLength} = \text{size}(\text{EDObj}, \text{Dim}) \]

Description

\[ N\text{FeatSam} = \text{size}(\text{EDObj}) \] returns a two-element row vector containing the number of features and number of samples in an ExptData object.

\[ [\text{NFeatures}, \text{NSamples}] = \text{size}(\text{EDObj}) \] returns the number of features and number of samples in an ExptData object as separate variables.

\[ \text{DimLength} = \text{size}(\text{EDObj}, \text{Dim}) \] returns the length of the dimension specified by \text{Dim}.

Input Arguments

\text{EDObj}

Object of the bioma.data.ExptData class.

Default:

\text{Dim}

Scalar specifying the dimension of the ExptData object. Choices are:

- 1 — Features
- 2 — Samples

Default:

Examples

Construct an ExptData object, and then determine the number of features and samples in it:

% Import bioma.data package to make constructor functions available
import bioma.data.*
% Create DataMatrix object from .txt file containing expression values from microarray experiment
% dmo = DataMatrix('File', 'mouseExprsData.txt');
% Construct ExptData object
```matlab
EDObj = ExptData(dmObj);
% Retrieve the number of features and samples
NumFeatSam = size(EDObj)
```

**See Also**
bioma.data.ExptData

**Topics**
“Representing Expression Data Values in ExptData Objects”
size

Class: bioma.data.MetaData
Package: bioma.data

Return size of MetaData object

Syntax

\[ NSamVar = \text{size}(MDObj) \]
\[ [\text{NSamples, NVariables}] = \text{size}(MDObj) \]
\[ \text{DimLength} = \text{size}(MDObj, Dim) \]

Description

\( NSamVar = \text{size}(MDObj) \) returns a two-element row vector containing the number of samples or features and number of variables in a MetaData object.

\[ [\text{NSamples, NVariables}] = \text{size}(MDObj) \] returns the number of samples or features and the number of variables in a MetaData object as separate variables.

\( \text{DimLength} = \text{size}(MDObj, Dim) \) returns the length of the dimension specified by \( Dim \).

Input Arguments

\( MDObj \)

Object of the \texttt{bioma.data.MetaData} class.

Default:

\( Dim \)

Scalar specifying the dimension of the MetaData object. Choices are:

- 1 — Samples
- 2 — Variables

Default:

Examples

Construct a MetaData object, and then determine the number of samples and variables in it:

```matlab
% Import bioma.data package to make constructor function available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Retrieve the number of samples and variables
NumSamVar = size(MDObj2)
```
See Also
bioma.data.MetaData

Topics
“Representing Sample and Feature Metadata in MetaData Objects”
**soapread**

Read data from Short Oligonucleotide Analysis Package (SOAP) file

**Syntax**

SOAPStruct = soapread(File)
SOAPStruct = soapread(File,Name,Value)

**Description**

SOAPStruct = soapread(File) reads File, a SOAP-formatted file (version 2.15) and returns the data in SOAPStruct, a MATLAB array of structures.

SOAPStruct = soapread(File,Name,Value) reads a SOAP-formatted file with additional options specified by one or more Name,Value pair arguments.

**Input Arguments**

**File**

Character vector or string specifying a file name, path and file name, or the text of a SOAP-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the Current Folder.

The soapread function reads SOAP-formatted files (version 2.15).

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

**BlockRead**

Scalar or vector that controls the reading of a single sequence entry or block of sequence entries from a SOAP-formatted file containing multiple sequences. Enter a scalar \( N \), to read the \( N \)th entry in the file. Enter a 1-by-2 vector \([M1, M2]\), to read a block of entries starting at the \( M1 \) entry and ending at the \( M2 \) entry. To read all remaining entries in the file starting at the \( M1 \) entry, enter a positive value for \( M1 \) and enter Inf for \( M2 \).

**Default:**

**AlignDetails**

Logical specifying whether or not to include the AlignDetails field in the SOAPStruct output argument. The AlignDetails field includes information on mismatches, insertions, and deletions in the alignment. Choices are true (default) or false.

**Default:** true
Output Arguments

SOAPStruct

An N-by-1 array of structures containing sequence alignment and mapping information from a SOAP-formatted file, where N is the number of alignment records stored in the SOAP-formatted file. Each structure contains the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryName</td>
<td>Name of aligned read sequence.</td>
</tr>
<tr>
<td>Sequence</td>
<td>Character vector containing the letter representations of the read sequence. It is the reverse-complement if the read sequence aligns to the reverse strand of the reference sequence.</td>
</tr>
<tr>
<td>Quality</td>
<td>Character vector containing the ASCII representation of the per-base quality score for the read sequence. The quality score is reversed if the read sequence aligns to the reverse strand of the reference sequence.</td>
</tr>
<tr>
<td>NumHits</td>
<td>The number of total instances where this read sequence aligned to an identical length of bases on another area of the reference sequence.</td>
</tr>
<tr>
<td>PairedEndSourceFile</td>
<td>Flag (a or b) specifying which source file to which the read sequence belongs. This field applies only to read sequences that are paired in the alignment.</td>
</tr>
<tr>
<td>Length</td>
<td>Scalar specifying the length of the read sequence.</td>
</tr>
<tr>
<td>Strand</td>
<td>+ or − specifying direction (forward or reverse) of reference sequence to which the read sequence aligns.</td>
</tr>
<tr>
<td>ReferenceName</td>
<td>Name or numeric ID of the reference sequence to which the read sequence aligns.</td>
</tr>
<tr>
<td>Position</td>
<td>Position (one-based offset) of the forward reference sequence where the left-most base of the alignment of the read sequence starts.</td>
</tr>
<tr>
<td>AlignDetails</td>
<td>Information on mismatches, insertions, and deletions in the alignment. For SOAP-formatted files v2.15, this field includes CIGAR strings.</td>
</tr>
</tbody>
</table>

Examples

Read the alignment records (entries) from the sample01.soap file into a MATLAB array of structures and access some of the data:

```matlab
% Read the alignment records stored in the file sample01.soap
data = soapread('sample01.soap')
data =
17x1 struct array with fields:
```
QueryName
Sequence
Quality
NumHits
PairedEndSourceFile
Length
Strand
ReferenceName
Position
AlignDetails

% Access the quality score for the 6th entry
data(6).Quality
ans =
<>.

% Determine the strand direction (forward or reverse) of the reference sequence to which the 12th entry aligns
data(12).Strand
ans =
-

Read a block of alignment records (entries) from the sample01.soap file into a MATLAB array of structures:

% Read a block of six entries from a SOAP file
data_5_10 = soapread('sample01.soap','blockread', [5 10])

% 6x1 struct array with fields:
QueryName
Sequence
Quality
NumHits
PairedEndSourceFile
Length
Strand
ReferenceName
Position
AlignDetails

Tips

If your SOAP-formatted file is too large to read using available memory, try either of the following:

- Use the BlockRead name-value pair arguments to read a subset of entries.
- Create a BioIndexedFile object from the SOAP-formatted file (using ‘TABLE’ for the Format), and then access the entries using methods of the BioIndexedFile class.
References


See Also
bamread | fastqread | samread

Topics
“Work with Next-Generation Sequencing Data”

External Websites
Sequence Read Archive

Introduced in R2010b
sortcols (DataMatrix)

Sort columns of DataMatrix object in ascending or descending order

Syntax

```
DMObjNew = sortcols(DMObj1)
DMObjNew = sortcols(DMObj1, Row)
DMObjNew = sortcols(DMObj1, 'ColName')
DMObjNew = sortcols(DMObj1, ..., Mode)
[DMObjNew, Indices] = sortcols(DMObj1, ...)
```

Input Arguments

- **DMObj1**: DataMatrix object, such as created by `DataMatrix` (object constructor).
- **Row**: One or more rows in `DMObj1` by which to sort the columns. Choices are:
  - Positive integer
  - Vector of positive integers
  - Character vector or string specifying a row name
  - Cell array of character vectors or string vector specifying multiple row names
  - Logical vector
- **'ColName'**: Character vector or string that specifies to sort the columns by the column names.
- **Mode**: Character vector or string specifying the order by which to sort the columns. Choices are `'ascend'` (default) or `'descend'`.

Output Arguments

- **DMObjNew**: DataMatrix object created from sorting the columns of another DataMatrix object.
- **Indices**: Index vector that links `DMObj1` to `DMObjNew`. In other words, `DMObjNew = DMObj1(:, idx)`.

Description

```
DMObjNew = sortcols(DMObj1) sorts the columns in `DMObj1` in ascending order based on the elements in the first row. For any columns that have equal elements in a row, sorting is based on the row immediately below.

DMObjNew = sortcols(DMObj1, Row) sorts the columns in `DMObj1` in ascending order based on the elements in the specified row. Any columns that have equal elements in the specified row are sorted based on the elements in the next specified row.

DMObjNew = sortcols(DMObj1, 'ColName') sorts the columns in `DMObj1` in ascending order according to the column names.
```
DMObjNew = sortcols(DMObj1, ..., Mode) specifies the order of the sort. Mode can be 'ascend' (default) or 'descend'.

[DMObjNew, Indices] = sortcols(DMObj1, ...) returns Indices, an index vector that links DMObj1 to DMObjNew. In other words, DMObjNew = DMObj1(:,idx).

See Also
DataMatrix | sortrows

Topics
DataMatrix object on page 1-532

Introduced in R2008b
sortrows (DataMatrix)

Sort rows of DataMatrix object in ascending or descending order

Syntax

\[
\begin{align*}
    DMObjNew &= \text{sortrows}(DMObj1) \\
    DMObjNew &= \text{sortrows}(DMObj1, \text{Column}) \\
    DMObjNew &= \text{sortrows}(DMObj1, 'RowName') \\
    DMObjNew &= \text{sortrows}(DMObj1, ..., \text{Mode}) \\
    [DMObjNew, Indices] &= \text{sortrows}(DMObj1, ...)
\end{align*}
\]

Input Arguments

| DMObj1 | DataMatrix object, such as created by DataMatrix (object constructor). |
| Column | One or more columns in DMObj1 by which to sort the rows. Choices are: |
|        | • Positive integer |
|        | • Vector of positive integers |
|        | • Character vector or string specifying a column name |
|        | • Cell array of character vectors or string vector specifying multiple column names |
|        | • Logical vector |
| 'RowName' | Character vector or string that specifies to sort the rows by the row names. |
| Mode | Character vector or string specifying the order by which to sort the rows. Choices are 'ascend' (default) or 'descend'. |

Output Arguments

| DMObjNew | DataMatrix object created from sorting the rows of another DataMatrix object. |
| Indices | Index vector that links DMObj1 to DMObjNew. In other words, DMObjNew = DMObj1(idx,:). |

Description

\( DMObjNew = \text{sortrows}(DMObj1) \) sorts the rows in DMObj1 in ascending order based on the elements in the first column. For any rows that have equal elements in a column, sorting is based on the column immediately to the right.

\( DMObjNew = \text{sortrows}(DMObj1, \text{Column}) \) sorts the rows in DMObj1 in ascending order based on the elements in the specified column. Any rows that have equal elements in the specified column are sorted based on the elements in the next specified column.

\( DMObjNew = \text{sortrows}(DMObj1, 'RowName') \) sorts the rows in DMObj1 in ascending order according to the row names.
DMObjNew = sortrows(DMObj1, ..., Mode) specifies the order of the sort. Mode can be 'ascend' (default) or 'descend'.

[DMObjNew, Indices] = sortrows(DMObj1, ...) returns Indices, an index vector that links DMObj1 to DMObjNew. In other words, DMObjNew = DMObj1(idx,:).

See Also
DataMatrix | sortcols

Topics
DataMatrix object on page 1-532

Introduced in R2008b
**sptread**

Read data from SPOT file

**Syntax**

```markdown
SPOTData = sptread(File)

SPOTData = sptread(File, 'CleanColNames', CleanColNamesValue)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Character vector or string specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a SPOT-formatted file (ASCII text file). If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.</td>
</tr>
<tr>
<td>CleanColNamesValue</td>
<td>Controls the use of valid MATLAB variable names.</td>
</tr>
</tbody>
</table>

**Description**

`SPOTData = sptread(File)` reads `File`, a SPOT-formatted file, and creates `SPOTData`, a MATLAB structure containing the following fields:

- Header
- Data
- Blocks
- Columns
- Rows
- IDs
- ColumnNames
- Indices
- Shape

`SPOTData = sptread(File, 'CleanColNames', CleanColNamesValue)` controls the use of valid MATLAB variable names. The column names in the SPOT-formatted file contain periods and some characters that cannot be used in MATLAB variable names. If you plan to use the column names as variable names in a function, use this option with `CleanColNames` set to `true` and the function will return the field `ColumnNames` with valid variable names.

The `Indices` field of the structure includes the indices that you can use for plotting heat maps of the data.

**Examples**

1. Read in a sample SPOT file and plot the median foreground intensity for the 635 nm channel. Note that the example file `spotdata.txt` is not provided with the Bioinformatics Toolbox software.
spotStruct = sptread('spotdata.txt')
maimage(spotStruct,'Rmedian');

Alternately, create a similar plot using more basic graphics commands.

Rmedian = magetfield(spotStruct,'Rmedian');
imagesc(Rmedian(spotStruct.Indices));
colormap bone
colorbar

See Also
affyread|agferead|celintensityread|geoseriesread|geosoftread|gprread|
ilmbsread|imageneread|maboxplot|magetfield

Introduced before R2006a
**std (DataMatrix)**

Return standard deviation values in DataMatrix object

**Syntax**

\[ S = \text{std}(\text{DMObj}) \]
\[ S = \text{std}(\text{DMObj}, \text{Flag}) \]
\[ S = \text{std}(\text{DMObj}, \text{Flag}, \text{Dim}) \]
\[ S = \text{std}(\text{DMObj}, \text{Flag}, \text{Dim}, \text{IgnoreNaN}) \]

**Input Arguments**

<table>
<thead>
<tr>
<th><strong>DMObj</strong></th>
<th>DataMatrix object, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
</table>
| **Flag**      | Scalar specifying how to normalize the data. Choices are:
|               | • 0 — Default. Normalizes using a sample size of \(N - 1\), unless \(N = 1\), in which case, normalizes using a sample size of 1.  
|               | • 1 — Normalizes using a sample size of \(N\). 
|               | \(N\) = the number of elements in each column or row, as specified by \(\text{Dim}\). For more information on the normalization equations, see the MATLAB function \text{std}. |
| **Dim**       | Scalar specifying the dimension of \(\text{DMObj}\) to calculate the standard deviations. Choices are:
|               | • 1 — Default. Returns standard deviation values for elements in each column.  
|               | • 2 — Returns standard deviation values for elements in each row. |
| **IgnoreNaN** | Specifies if NaNs should be ignored. Choices are true (default) or false. |

**Output Arguments**

| **S** | Either of the following:
|-------|-----------------------------------------------------------------------|
|       | • Row vector containing the standard deviation values from elements in each column in \(\text{DMObj}\) (when \(\text{Dim} = 1\))  
|       | • Column vector containing the standard deviation values from elements in each row in \(\text{DMObj}\) (when \(\text{Dim} = 2\)) |

**Description**

\(S = \text{std}(\text{DMObj})\) returns the standard deviation values of the elements in the columns of a DataMatrix object, treating NaNs as missing values. The data is normalized using a sample size of \(N - 1\), where \(N\) = the number of elements in each column. \(S\) is a row vector containing the standard deviation values for elements in each column in \(\text{DMObj}\).
$S = \text{std}(\text{DMObj}, \text{Flag})$ specifies how to normalize the data. If $\text{Flag} = 0$, normalizes using a sample size of $N - 1$. If $\text{Flag} = 1$, normalizes using a sample size of $N$. $N$ = the number of elements in each column or row, as specified by $\text{Dim}$. For more information on the normalization equations, see the MATLAB function std. Default $\text{Flag} = 0$.

$S = \text{std}(\text{DMObj}, \text{Flag}, \text{Dim})$ returns the standard deviation values of the elements in the columns or rows of a DataMatrix object, as specified by $\text{Dim}$. If $\text{Dim} = 1$, returns $S$, a row vector containing the standard deviation values for elements in each column in $\text{DMObj}$. If $\text{Dim} = 2$, returns $S$, a column vector containing the standard deviation values for elements in each row in $\text{DMObj}$. Default $\text{Dim} = 1$.

$S = \text{std}(\text{DMObj}, \text{Flag}, \text{Dim}, \text{IgnoreNaN})$ specifies if NaNs should be ignored. $\text{IgnoreNaN}$ can be true (default) or false.

**See Also**
DataMatrix | mean | median | var

**Topics**
DataMatrix object on page 1-532

**Introduced in R2008b**
subtree (phytree)

Extract phylogenetic subtree

Syntax

\[ Tree2 = \text{subtree}(Tree1, \text{Nodes}) \]

Description

\[ Tree2 = \text{subtree}(Tree1, \text{Nodes}) \] extracts a new subtree (\( Tree2 \)) where the new root is the first common ancestor of the \( \text{Nodes} \) vector from \( Tree1 \). Nodes in the tree are indexed as \([1:\text{NUMLEAVES}]\) for the leaves and as \([\text{NUMLEAVES}+1:\text{NUMLEAVES}+\text{NUMBRANCHES}]\) for the branches. Nodes can also be a logical array of following sizes \([\text{NUMLEAVES}+\text{NUMBRANCHES} \times 1], [\text{NUMLEAVES} \times 1] \) or \([\text{NUMBRANCHES} \times 1]\).

Examples

1. Load a phylogenetic tree created from a protein family.
   
   ```
   tr = phytreeread('pf00002.tree');
   ```

2. Get the subtree that contains the VIPR2 and GLR human proteins.
   
   ```
   sel = getbyname(tr,{'vipr2_human','glr_human'});
   sel = any(sel,2);
   tr = subtree(tr,sel);
   view(tr)
   ```
See Also
get | getbyname | phytree | prune | select

Topics
phytree object on page 1-1274

Introduced before R2006a
sum (DataMatrix)

Return sum of elements in DataMatrix object

Syntax

\[ S = \text{sum}(\text{DMObj}) \]
\[ S = \text{sum}(\text{DMObj}, \text{Dim}) \]
\[ S = \text{sum}(\text{DMObj}, \text{Dim}, \text{IgnoreNaN}) \]

Input Arguments

<table>
<thead>
<tr>
<th>DMObj</th>
<th>DataMatrix object, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dim</td>
<td>Scalar specifying the dimension of DMObj to calculate the sums. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 1 — Default. Returns sum of elements in each column.</td>
</tr>
<tr>
<td></td>
<td>• 2 — Returns sum of elements in each row.</td>
</tr>
<tr>
<td>IgnoreNaN</td>
<td>Specifies if NaNs should be ignored. Choices are true (default) or false.</td>
</tr>
</tbody>
</table>

Output Arguments

<table>
<thead>
<tr>
<th>S</th>
<th>Either of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Row vector containing the sums of the elements in each column in DMObj (when Dim = 1)</td>
</tr>
<tr>
<td></td>
<td>• Column vector containing the sums of the elements in each row in DMObj (when Dim = 2)</td>
</tr>
</tbody>
</table>

Description

\[ S = \text{sum}(\text{DMObj}) \] returns the sum of the elements in the columns of a DataMatrix object, treating NaNs as missing values. \( S \) is a row vector containing the sums of the elements in each column in DMObj. If the values in DMObj are singles, then \( S \) is a single; otherwise, \( S \) is a double.

\[ S = \text{sum}(\text{DMObj}, \text{Dim}) \] returns the sum of the elements in the columns or rows of a DataMatrix object, as specified by \( \text{Dim} \). If \( \text{Dim} = 1 \), returns \( S \), a row vector containing the sums of the elements in each column in DMObj. If \( \text{Dim} = 2 \), returns \( S \), a column vector containing the sums of the elements in each row in DMObj. Default \( \text{Dim} = 1 \).

\[ S = \text{sum}(\text{DMObj}, \text{Dim}, \text{IgnoreNaN}) \] specifies if NaNs should be ignored. IgnoreNaN can be true (default) or false.

See Also

DataMatrix | max | min

Topics

DataMatrix object on page 1-532
sum (DataMatrix)

Introduced in R2008b
swalign

Locally align two sequences using Smith-Waterman algorithm

Syntax

Score = swalign(Seq1, Seq2)
[Score, Alignment] = swalign(Seq1, Seq2)
[Score, Alignment, Start] = swalign(Seq1, Seq2)

... = swalign(Seq1, Seq2, ...'Alphabet', AlphabetValue)
... = swalign(Seq1, Seq2, ...'ScoringMatrix', ScoringMatrixValue, ...)
... = swalign(Seq1, Seq2, ...'Scale', ScaleValue, ...)
... = swalign(Seq1, Seq2, ...'GapOpen', GapOpenValue, ...)
... = swalign(Seq1, Seq2, ...'ExtendGap', ExtendGapValue, ...)
... = swalign(Seq1, Seq2, ...'Showscore', ShowscoreValue, ...)

Input Arguments

<table>
<thead>
<tr>
<th>Seq1, Seq2</th>
<th>Amino acid or nucleotide sequences. Enter any of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Character vector or string of letters representing amino acids or nucleotides, such as returned by int2aa or int2nt</td>
</tr>
<tr>
<td></td>
<td>• Vector of integers representing amino acids or nucleotides, such as returned by aa2int or nt2int</td>
</tr>
<tr>
<td></td>
<td>• Structure containing a Sequence field</td>
</tr>
</tbody>
</table>

**Tip** For help with letter and integer representations of amino acids and nucleotides, see Amino Acid Lookup or Nucleotide Lookup.

| AlphabetValue | Character vector or string specifying the type of sequence. Choices are 'AA' (default) or 'NT'. |
### ScoringMatrixValue

Either of the following:

- Character vector or string specifying the scoring matrix to use for the local alignment. Choices for amino acid sequences are:
  - 'BLOSUM62'
  - 'BLOSUM30' increasing by 5 up to 'BLOSUM90'
  - 'BLOSUM100'
  - 'PAM10' increasing by 10 up to 'PAM500'
  - 'DAYHOFF'
  - 'GONNET'

Default is:

- 'BLOSUM50' — When AlphabetValue equals 'AA'
- 'NUC44' — When AlphabetValue equals 'NT'

**Note** The above scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the 'Scale' property to specify an additional scale factor to convert the output score from bits to another unit.

- Matrix representing the scoring matrix to use for the local alignment, such as returned by the blosum, pam, dayhoff, gonnet, or nuc44 function.

**Note** If you use a scoring matrix that you created or was created by one of the above functions, the matrix does not include a scale factor. The output score will be returned in the same units as the scoring matrix. You can use the 'Scale' property to specify a scale factor to convert the output score to another unit.

**Note** If you need to compile swalign into a stand-alone application or software component using MATLAB Compiler, use a matrix instead of a character vector or string for ScoringMatrixValue.
**ScaleValue**

Positive value that specifies a scale factor that is applied to the output score.

For example, if the output score is initially determined in bits, and you enter \( \log(2) \) for `ScaleValue`, then `swalign` returns `Score` in nats.

Default is 1, which does not change the units of the output score.

**Note** If the 'ScoringMatrix' property also specifies a scale factor, then `swalign` uses it first to scale the output score, then applies the scale factor specified by `ScaleValue` to rescale the output score.

**GapOpenValue**

Positive value specifying the penalty for opening a gap in the alignment. Default is 8.

**ExtendGapValue**

Positive value specifying the penalty for extending a gap using the affine gap penalty scheme.

**Note** If you specify this value, `swalign` uses the affine gap penalty scheme, that is, it scores the first gap using the `GapOpenValue` and scores subsequent gaps using the `ExtendGapValue`. If you do not specify this value, `swalign` scores all gaps equally, using the `GapOpenValue` penalty.

**ShowscoreValue**

Controls the display of the scoring space and the winning path of the alignment. Choices are `true` or `false` (default).

### Output Arguments

<table>
<thead>
<tr>
<th><strong>Score</strong></th>
<th>Optimal local alignment score in bits.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alignment</strong></td>
<td>3-by-N character array showing the two sequences, <code>Seq1</code> and <code>Seq2</code>, in the first and third rows, and symbols representing the optimal local alignment between them in the second row.</td>
</tr>
<tr>
<td><strong>Start</strong></td>
<td>2-by-1 vector of indices indicating the starting point in each sequence for the alignment.</td>
</tr>
</tbody>
</table>

### Description

`Score = swalign(Seq1, Seq2)` returns the optimal local alignment score in bits. The scale factor used to calculate the score is provided by the scoring matrix.

`[Score, Alignment] = swalign(Seq1, Seq2)` returns a 3-by-N character array showing the two sequences, `Seq1` and `Seq2`, in the first and third rows, and symbols representing the optimal local alignment between them in the second row. The symbol `|` indicates amino acids or nucleotides.
that match exactly. The symbol \( : \) indicates amino acids or nucleotides that are related as defined by
the scoring matrix (nonmatches with a zero or positive scoring matrix value).

\( \text{[Score, Alignment, Start]} = \text{swalign(} \text{Seq1, Seq2)} \) returns a 2-by-1 vector of indices
indicating the starting point in each sequence for the alignment.

\( \ldots = \text{swalign(} \text{Seq1,Seq2, \ldots'PropertyName', PropertyValue, \ldots)} \) calls swalign
with optional properties that use property name/property value pairs. You can specify one or more
properties in any order. Each \( \text{PropertyName} \) must be enclosed in single quotation marks and is case
insensitive. These property name/property value pairs are as follows:

\( \ldots = \text{swalign(} \text{Seq1,Seq2, \ldots'Alphabet', AlphabetValue)} \) specifies the type of
sequences. Choices are 'AA' (default) or 'NT'.

\( \ldots = \text{swalign(} \text{Seq1,Seq2, \ldots'ScoringMatrix', ScoringMatrixValue, \ldots)} \) specifies
the scoring matrix to use for the local alignment. Default is:

- 'BLOSUM50' — When \( \text{AlphabetValue} \) equals 'AA'
- 'NUC44' — When \( \text{AlphabetValue} \) equals 'NT'

\( \ldots = \text{swalign(} \text{Seq1,Seq2, \ldots'Scale', ScaleValue, \ldots)} \) specifies a scale factor that is
applied to the output score, thereby controlling the units of the output score. Choices are any positive
value.

\( \ldots = \text{swalign(} \text{Seq1,Seq2, \ldots'GapOpen', GapOpenValue, \ldots)} \) specifies the penalty for
opening a gap in the alignment. Choices are any positive value. Default is 8.

\( \ldots = \text{swalign(} \text{Seq1,Seq2, \ldots'ExtendGap', ExtendGapValue, \ldots)} \) specifies the penalty
for extending a gap using the affine gap penalty scheme. Choices are any positive value.

\( \ldots = \text{swalign(} \text{Seq1,Seq2, \ldots'Showscore', ShowscoreValue, \ldots)} \) controls the display
of the scoring space and winning path of the alignment. Choices are \( \text{true} \) or \( \text{false} \) (default).
The scoring space is a heat map displaying the best scores for all the partial alignments of two sequences. The color of each \((n_1, n_2)\) coordinate in the scoring space represents the best score for the pairing of subsequences \(\text{Seq1}(s_1:n_1)\) and \(\text{Seq2}(s_2:n_2)\), where \(n_1\) is a position in \(\text{Seq1}\), \(n_2\) is a position in \(\text{Seq2}\), \(s_1\) is any position in \(\text{Seq1}\) between \(1:n_1\), and \(s_2\) is any position in \(\text{Seq2}\) between \(1:n_2\). The best score for a pairing of specific subsequences is determined by scoring all possible alignments of the subsequences by summing matches and gap penalties.

The winning path is represented by black dots in the scoring space, and it illustrates the pairing of positions in the optimal local alignment. The color of the last point (lower right) of the winning path represents the optimal local alignment score for the two sequences and is the \texttt{Score} output returned by \texttt{swalign}.

\textbf{Note} The scoring space visually shows tandem repeats, small segments that potentially align, and partial alignments of domains from rearranged sequences.

\textbf{Examples}

1. Locally align two amino acid sequences using the \texttt{BLOSUM50} (default) scoring matrix and the default values for the \texttt{GapOpen} and \texttt{ExtendGap} properties. Return the optimal local alignment score in bits and the alignment character array.

\[
[\text{Score}, \text{Alignment}] = \text{swalign}('\text{VSPAGMSGYD}', '\text{IPGKASYD}')
\]

\texttt{Score} =
Locally align two amino acid sequences specifying the PAM250 scoring matrix and a gap open penalty of 5.

```matlab
[Score, Alignment] = swalign('HEAGAWGHEE','PAWHEAE',
    'ScoringMatrix', 'pam250',
    'GapOpen',5)
```

Score =

-8.6667

Alignment =

```
PAGMASGYD
| | | | |
P-GKAS-YD
```

Locally align two amino acid sequences returning the Score in nat units (nats) by specifying a scale factor of \( \log(2) \).

```matlab
[Score, Alignment] = swalign('HEAGAWGHEE','PAWHEAE','Scale',log(2))
```

Score =

6.4694

Alignment =

```
GAWGHE
:|| ||
PAW-HE
```

References


See Also

- aa2int
- aminolookup
- baselookup
- blosum
- dayhoff
- gonnet
- int2aa
- int2nt
- localalign
- multialign
- nt2aa
- nt2int
- nuc44
- nwalkign
- pam
- pdbsuperpose
- seqdotplot
- showalignment

Introduced before R2006a
term class

Data structure containing information about Gene Ontology (GO) term

Description

A term object is a data structure containing information about a Gene Ontology (GO) term. You can explore and traverse Gene Ontology terms using “is_a” and “part_of” relationships.

Construction

geneont Create geneont object and term objects

Properties

definition Read-only character vector that defines GO term
id Read-only numeric value that corresponds to GO identifier of GO term
is_a Read-only numeric array containing GO identifiers of GO terms that have an “is a” relationship with this GO term
name Read-only character vector representing name of GO term
obsolete Read-only Boolean value that indicates whether a GO term is obsolete
ontology Read-only character vector describing the ontology of GO term
part_of Read-only numeric array containing GO identifiers of GO terms that have a “part of” relationship with this GO term
synonym Read-only array containing GO terms that are synonyms of this GO term

Instance Hierarchy

A geneont object contains term objects.

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects (MATLAB) in the MATLAB Programming Fundamentals documentation.

Indexing

You can use parenthesis () indexing to access the terms in an array of handles to term objects. See “Examples” on page 1-1644 below.

Examples

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.
GeneontObj = geneont('LIVE', true)

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

Gene Ontology object with 27827 Terms.

2 Use the terms property to create a variable containing an array of handles to the term objects of the geneont object.

array_of_terms = GeneontObj.terms

27827x1 struct array with fields:
id
name
ontology
definition
comment
synonym
is_a
part_of
obsolete

Note Although the terms property is an array of handles to term objects, in the MATLAB Command Window, it displays as a structure array, with one structure for each GO term in the geneont object.

3 Return the fifth term (term object) of the geneont object.

fifth_term = array_of_terms(5)

    id: 6
    name: [1x60 char]
    ontology: 'molecular function'
    definition: [1x321 char]
    comment: ''
    synonym: []
    is_a: 5385
    part_of: [0x1 double]
    obsolete: 0

See Also
geneont
tgspcinfo

Return information about SPC file

Syntax

\[ \text{InfoStruct} = \text{tgspcinfo}(\text{File}) \]

Description

\[ \text{InfoStruct} = \text{tgspcinfo}(\text{File}) \]
returns a MATLAB structure containing summary information about a Galactic SPC file from Thermo Scientific®.

Input Arguments

File

Character vector or string specifying a file name or path and file name of an SPC file. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.

Default:

Output Arguments

InfoStruct

MATLAB structure containing the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>Name of the SPC file.</td>
</tr>
<tr>
<td>FileSize</td>
<td>Size of the SPC file in bytes.</td>
</tr>
<tr>
<td>ExperimentType</td>
<td>Experimental technique used to create the data.</td>
</tr>
<tr>
<td>NumDataPoints</td>
<td>Number of data points (y data values) in the SPC file.</td>
</tr>
<tr>
<td>XFirst</td>
<td>First x data value in the SPC file.</td>
</tr>
<tr>
<td>XLast</td>
<td>Last x data value in the SPC file.</td>
</tr>
<tr>
<td>NumScans</td>
<td>Number of scans or subfiles in the SPC file.</td>
</tr>
<tr>
<td>XLabel</td>
<td>Label for the x data values.</td>
</tr>
<tr>
<td>YLabel</td>
<td>Label for the y data values.</td>
</tr>
<tr>
<td>ZLabel</td>
<td>Label for the z data values.</td>
</tr>
<tr>
<td>CollectionTime</td>
<td>Date and time the scans were collected.</td>
</tr>
<tr>
<td>CollectionTimeDatenum</td>
<td>Date and time the scans were collected in serial date number format. For more information, see datenum.</td>
</tr>
<tr>
<td>Resolution</td>
<td>Instrument resolution.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SourceInstrument</td>
<td>Name or model of the instrument used to collect data.</td>
</tr>
<tr>
<td>InterferogramPeakPointNumber</td>
<td>Peak point number for interferograms. It is 0 for scans that are not interferograms.</td>
</tr>
<tr>
<td>Comment</td>
<td>User-provided comments.</td>
</tr>
<tr>
<td>CustomAxisUnitLabel</td>
<td>User-provided labels for the axis units.</td>
</tr>
<tr>
<td>SubScanHeaders</td>
<td>Header information for subfiles or scans, including scan index, next scan index, and w data value.</td>
</tr>
<tr>
<td>ZValues</td>
<td>Vector containing the z data values of all scans in the SPC file.</td>
</tr>
</tbody>
</table>

**Examples**

This example assumes that you already have an SPC file to use. sample.spc file is not provided with the Bioinformatics Toolbox software.

Return information about an SPC file:

```matlab
% Return information about an SPC file named sample.spc
info = tgspcinfo('sample.spc')
```

Reading header for file: SAMPLE.SPC
File contains 1 scans

```
info =

    Filename: 'SAMPLE.SPC'
    FileSize: 48380
    ExperimentType: 'General SPC'
    NumDataPoints: 12031
        XFirst: 6.2998e+003
        XLast: 499.9531
    NumScans: 1
        XLabel: 'Wavenumber (cm-1)'
        YLabel: 'Absorbance'
        ZLabel: 'Arbitrary'
    CollectionTime: '08-Mar-1993 15:13:00'
    CollectionTimeDatenum: 7.2800e+005
    Resolution: '  .00   '
    SourceInstrument: ''
    InterferogramPeakPointNumber: 0
    Comment: [1x74 char]
    CustomAxisUnitLabel: ''
    SubScanHeaders: [1x1 struct]
    ZValues: 0
```

**See Also**
datenum | tgspcread

**Introduced in R2009b**
**tgspcread**

Read data from SPC file

**Syntax**

```matlab
SPCStruct = tgspcread(File)
tgspcread(..., 'ZRange', ZRangeValue, ...)
tgspcread(..., 'ScanIndices', ScanIndicesValue, ...)
tgspcread(..., 'Verbose', VerboseValue, ...)
```

**Description**

`SPCStruct = tgspcread(File)` reads a Galactic SPC file from Thermo Scientific, and returns the data in a MATLAB structure.

tgspcread(..., 'PropertyName', PropertyValue, ...) calls `tgspcread` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each `PropertyName` in single quotation marks. Each `PropertyName` is case insensitive. These property name/property value pairs are as follows:

- `tgspcread(..., 'ZRange', ZRangeValue, ...)` specifies a range of z data values in the SPC file from which to extract scans.
- `tgspcread(..., 'ScanIndices', ScanIndicesValue, ...)` specifies a scan, multiple scans, or range of scans in the SPC file to read.
- `tgspcread(..., 'Verbose', VerboseValue, ...)` controls the display of the progress of the reading of the SPC file. Choices are `true` (default) or `false`.

**Input Arguments**

- **File**
  Character vector or string specifying a file name or path and file name of an SPC file that conforms to the Thermo Scientific Universal Data Format Specification. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.

  **Default:**

- **ZRangeValue**
  Two-element numeric array `[Start End]` that specifies the range of z data values in `File` to read. `Start` and `End` must be positive scalars, and `Start` must be less than `End`. Default is to extract all scans.

  **Tip**  For summary information about the z data values in an SPC file, use the `tgspcinfo` function.

  **Note**  If you specify a `ZRangeValue`, you cannot specify a `ScanIndicesValue`. 
Default:

**ScanIndicesValue**

Positive integer, vector of integers, or a two-element numeric array \([Start\_Ind: End\_Ind]\) that specifies a scan, multiple scans, or a range of scans in *File* to read. *Start\_Ind* and *End\_Ind* are each positive integers indicating a scan index. *Start\_Ind* must be less than *End\_Ind*. Default is to read all scans.

**Tip** For summary information about the scan indices in an SPC file, check the *NumScans* field in the structure returned by the `tgspcinfo` function.

**Note** If you specify a *ScanIndicesValue*, you cannot specify a *ZRangeValue*.

Default:

**VerboseValue**

Controls the display of the progress of the reading of *File*. Choices are true (default) or false.

Default:

**Output Arguments**

**SPCStruct**

Structure containing information from an SPC file. The structure contains the following fields.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Structure containing the following fields:</td>
</tr>
<tr>
<td></td>
<td>• Filename — Name of the SPC file.</td>
</tr>
<tr>
<td></td>
<td>• FileSize — Size of the SPC file in bytes.</td>
</tr>
<tr>
<td></td>
<td>• ExperimentType — Experimental technique used to create the data.</td>
</tr>
<tr>
<td></td>
<td>• NumDataPoints — Number of data points (y data values) in the SPC file.</td>
</tr>
<tr>
<td></td>
<td>• XFirst — First x data value in the SPC file.</td>
</tr>
<tr>
<td></td>
<td>• XLast — Last x data value in the SPC file.</td>
</tr>
<tr>
<td></td>
<td>• NumScans — Number of scans or subfiles in the SPC file.</td>
</tr>
<tr>
<td></td>
<td>• XLabel — Label for the x data values.</td>
</tr>
<tr>
<td></td>
<td>• YLabel — Label for the y data values.</td>
</tr>
<tr>
<td></td>
<td>• ZLabel — Label for the z data values.</td>
</tr>
<tr>
<td></td>
<td>• CollectionTime — Date and time the scan data were collected.</td>
</tr>
<tr>
<td></td>
<td>• CollectionTimeDatenum — Date and time the scan data were collected in serial date number format. For more information, see datenum.</td>
</tr>
<tr>
<td></td>
<td>• Resolution — Instrument resolution.</td>
</tr>
<tr>
<td></td>
<td>• SourceInstrument — Name or model of instrument used to collect data.</td>
</tr>
<tr>
<td></td>
<td>• InterferogramPeakPointNumber — Peak point number for interferograms. It is 0 for scans that are not interferograms.</td>
</tr>
<tr>
<td></td>
<td>• Comment — User-provided comments.</td>
</tr>
<tr>
<td></td>
<td>• CustomAxisUnitLabel — User-provided labels for the axis units.</td>
</tr>
<tr>
<td></td>
<td>• SubScanHeaders — Header information for subfiles or scans, including scan index, next scan index, and w data value.</td>
</tr>
<tr>
<td></td>
<td>• ZValues — Vector containing the z data values of all scans in the SPC file.</td>
</tr>
<tr>
<td>X</td>
<td>Vector or cell array containing the x data values.</td>
</tr>
<tr>
<td></td>
<td>If all scans share the same x data values, then X is a vector. If scans have different x data values, then X is a cell array.</td>
</tr>
<tr>
<td>Y</td>
<td>Vector, matrix, or cell array containing the y data values.</td>
</tr>
<tr>
<td></td>
<td>If there is only one scan, then Y is a vector. If there are multiple scans that share the same x data values, then Y is a matrix.</td>
</tr>
<tr>
<td></td>
<td>If there are multiple scans having different x data values, then Y is a cell array.</td>
</tr>
<tr>
<td>Z</td>
<td>Vector containing the z data values of scans read from the SPC file.</td>
</tr>
</tbody>
</table>

**Examples**

This example assumes that you already have an SPC file to use. sample.spc file is not provided with the Bioinformatics Toolbox software.
Read an SPC file:

% Read the contents of an SPC file into a MATLAB structure
out = tgspcread('results.spc')

File contains 1 scans

out =
    Header: [1x1 struct]
        X: [12031x1 single]
        Y: [12031x1 double]
        Z: 0

Plot an SPC file:

% Plot the first scan in the SPC file:
plot(out.X,out.Y(:,1));

See Also
datenum | jcampread | mzcdf2peaks | mzcdfinfo | mzcdfread | mzxml2peaks | mzxmlinfo | mzxmlread | tgspcinfo

Introduced in R2009b
times (DataMatrix)

Multiply DataMatrix objects

Syntax

\[
\text{DMObjNew} = \text{times}(\text{DMObj1, DMObj2}) \\
\text{DMObjNew} = \text{DMObj1} .* \text{DMObj2} \\
\text{DMObjNew} = \text{times}(\text{DMObj1, B}) \\
\text{DMObjNew} = \text{DMObj1} .* \text{B} \\
\text{DMObjNew} = \text{times}(\text{B, DMObj1}) \\
\text{DMObjNew} = \text{B} .* \text{DMObj1}
\]

Input Arguments

- **DMObj1, DMObj2**: DataMatrix objects, such as created by `DataMatrix` (object constructor).
- **B**: MATLAB numeric or logical array.

Output Arguments

- **DMObjNew**: DataMatrix object created by multiplication.

Description

\[
\text{DMObjNew} = \text{times}(\text{DMObj1, DMObj2}) \text{ or the equivalent } \text{DMObjNew} = \text{DMObj1} .* \text{DMObj2}
\]

performs an element-by-element multiplication of the DataMatrix objects \(\text{DMObj1}\) and \(\text{DMObj2}\) and places the results in \(\text{DMObjNew}\), another DataMatrix object. \(\text{DMObj1}\) and \(\text{DMObj2}\) must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for \(\text{DMObjNew}\) are the same as \(\text{DMObj1}\), unless \(\text{DMObj1}\) is a scalar; then they are the same as \(\text{DMObj2}\).

\[
\text{DMObjNew} = \text{times}(\text{DMObj1, B}) \text{ or the equivalent } \text{DMObjNew} = \text{DMObj1} .* \text{B}
\]

performs an element-by-element multiplication of the DataMatrix object \(\text{DMObj1}\) and \(\text{B}\), a numeric or logical array, and places the results in \(\text{DMObjNew}\), another DataMatrix object. \(\text{DMObj1}\) and \(\text{B}\) must have the same size (number of rows and columns), unless \(\text{B}\) is a scalar. The size (number of rows and columns), row names, and column names for \(\text{DMObjNew}\) are the same as \(\text{DMObj1}\).

\[
\text{DMObjNew} = \text{times}(\text{B, DMObj1}) \text{ or the equivalent } \text{DMObjNew} = \text{B} .* \text{DMObj1}
\]

performs an element-by-element multiplication of \(\text{B}\), a numeric or logical array, and the DataMatrix object \(\text{DMObj1}\), and places the results in \(\text{DMObjNew}\), another DataMatrix object. \(\text{DMObj1}\) and \(\text{B}\) must have the same size (number of rows and columns), unless \(\text{B}\) is a scalar. The size (number of rows and columns), row names, and column names for \(\text{DMObjNew}\) are the same as \(\text{DMObj1}\).

**Note**: Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.
MATLAB calls \texttt{DMObjNew = times(X, Y)} for the syntax \texttt{DMObjNew = X.* Y} when \( X \) or \( Y \) is a DataMatrix object.

**See Also**

“Arithmetic” (MATLAB) | DataMatrix | \texttt{minus} | \texttt{plus}

**Topics**

DataMatrix object on page 1-532

**Introduced in R2008b**
**topoorder (biograph)**

Perform topological sort of directed acyclic graph extracted from biograph object

**Syntax**

`order = topoorder(BGObj)`

**Arguments**

| BGObj | Biograph object created by `biograph` (object constructor). |

**Description**

**Tip** For introductory information on graph theory functions, see “Graph Theory Functions”.

`order = topoorder(BGObj)` returns an index vector with the order of the nodes sorted topologically. In topological order, an edge can exist between a source node `u` and a destination node `v`, if and only if `u` appears before `v` in the vector `order`. `BGObj` is a biograph object from which an N-by-N adjacency matrix is extracted and represents a directed acyclic graph (DAG). In the N-by-N sparse matrix, all nonzero entries indicate the presence of an edge.

**References**


**See Also**

allshortestpaths | biograph | conncomp | graphtopoorder | isdag | isomorphism | isspantree | maxflow | minspantree | shortestpath | traverse

**Topics**

biograph object on page 1-185

**Introduced in R2006b**
**traceplot**

Draw nucleotide trace plots

**Syntax**

```matlab
traceplot(TraceStructure)
traceplot(A, C, G, T)
h = traceplot(...)
```

**Description**

traceplot(TraceStructure) creates a trace plot from data in a structure with fields A, C, G, and T.

traceplot(A, C, G, T) creates a trace plot from data in vectors A, C, G, and T.

h = traceplot(...) returns a structure with the handles of the lines corresponding to A, C, G, T.

**Examples**

1. Read trace data from an SCF-formatted file into a MATLAB structure.

   ```matlab
tstruct = scfread('sample.scf')
tstruct =
   
   A: [10827x1 double]
   C: [10827x1 double]
   G: [10827x1 double]
   T: [10827x1 double]
```

2. Draw a nucleotide trace plot of the data.

   ```matlab
   traceplot(tstruct)
   ```
See Also

scfread

Introduced before R2006a
traverse (biograph)

Traverse biograph object by following adjacent nodes

Syntax

\[
\text{[disc, pred, closed] = traverse(BGObj, S)}
\]

\[
[...\] = \text{traverse(BGObj, S, 'Depth', DepthValue, ...)}
\]

\[
[...\] = \text{traverse(BGObj, S, 'Directed', DirectedValue, ...)}
\]

\[
[...\] = \text{traverse(BGObj, S, 'Method', MethodValue, ...)}
\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGObj</td>
<td>Biograph object created by biograph (object constructor).</td>
</tr>
<tr>
<td>S</td>
<td>Integer that indicates the source node in BGObj.</td>
</tr>
<tr>
<td>DepthValue</td>
<td>Integer that indicates a node in BGObj that specifies the depth of the search. Default is Inf (infinity).</td>
</tr>
<tr>
<td>DirectedValue</td>
<td>Property that indicates whether graph represented by an N-by-N adjacency matrix extracted from a biograph object, BGObj is directed or undirected. Enter false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.</td>
</tr>
<tr>
<td>MethodValue</td>
<td>Character vector that specifies the algorithm used to traverse the graph. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 'BFS' — Breadth-first search. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.</td>
</tr>
<tr>
<td></td>
<td>• 'DFS' — Default algorithm. Depth-first search. Time complexity is O(N +E), where N and E are number of nodes and edges respectively.</td>
</tr>
</tbody>
</table>

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions”.

\[
\text{[disc, pred, closed] = traverse(BGObj, S)}
\] traverses the directed graph represented by an N-by-N adjacency matrix extracted from a biograph object, BGObj, starting from the node indicated by integer S. In the N-by-N sparse matrix, all nonzero entries indicate the presence of an edge. disc is a vector of node indices in the order in which they are discovered. pred is a vector of predecessor node indices (listed in the order of the node indices) of the resulting spanning tree. closed is a vector of node indices in the order in which they are closed.

\[
[...\] = \text{traverse(BGObj, S, 'PropertyName', PropertyValue, ...)}
\] calls traverse with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:
[...] = traverse(BGObj, S, ...'Depth', DepthValue, ...) specifies the depth of the search. DepthValue is an integer indicating a node in the graph represented by the N-by-N adjacency matrix extracted from a biograph object, BGObj. Default is Inf (infinity).

[...] = traverse(BGObj, S, ...'Directed', DirectedValue, ...) indicates whether the graph represented by the N-by-N adjacency matrix extracted from a biograph object, BGObj is directed or undirected. Set DirectedValue to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.

[...] = traverse(BGObj, S, ...'Method', MethodValue, ...) lets you specify the algorithm used to traverse the graph represented by the N-by-N adjacency matrix extracted from a biograph object, BGObj. Choices are:

- 'BFS' — Breadth-first search. Time complexity is $O(N+E)$, where N and E are number of nodes and edges respectively.
- 'DFS' — Default algorithm. Depth-first search. Time complexity is $O(N+E)$, where N and E are number of nodes and edges respectively.

References


See Also

allshortestpaths | biograph | conncomp | graphtraverse | isdag | isomorphism | isspantree | maxflow | minspantree | shortestpath | topoorder

Topics

biograph object on page 1-185

Introduced in R2006b
### var (DataMatrix)

Return variance values in DataMatrix object

**Syntax**

\[ V = \text{var}(\text{DMObj}) \]
\[ V = \text{var}(\text{DMObj}, \text{Flag}) \]
\[ V = \text{var}(\text{DMObj}, \text{Wgt}) \]
\[ V = \text{var}(\ldots, \text{Dim}) \]
\[ V = \text{var}(\ldots, \text{Dim}, \text{IgnoreNaN}) \]

**Input Arguments**

<table>
<thead>
<tr>
<th>DMObj</th>
<th>DataMatrix object, such as created by DataMatrix (object constructor).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag</td>
<td>Scalar specifying how to normalize the data. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 0 — Default. Normalizes using a sample size of ( N - 1 ), unless ( N = 1 ), in which case, normalizes using a sample size of 1.</td>
</tr>
<tr>
<td></td>
<td>• 1 — Normalizes using a sample size of ( N ).</td>
</tr>
<tr>
<td></td>
<td>( N ) = the number of elements in each column or row, as specified by Dim.</td>
</tr>
<tr>
<td></td>
<td>For more information on the normalization equations, see the MATLAB function std.</td>
</tr>
<tr>
<td>Wgt</td>
<td>Weight vector equal in length to the dimension over which var operates (specified by Dim. It is used to compute the variance.</td>
</tr>
<tr>
<td>Dim</td>
<td>Scalar specifying the dimension of DMObj to calculate the variances. Choices are:</td>
</tr>
<tr>
<td></td>
<td>• 1 — Default. Returns variance values for elements in each column.</td>
</tr>
<tr>
<td></td>
<td>• 2 — Returns variance values for elements in each row.</td>
</tr>
<tr>
<td>IgnoreNaN</td>
<td>Specifies if NaNs should be ignored. Choices are true (default) or false.</td>
</tr>
</tbody>
</table>

**Output Arguments**

| V | An unbiased estimator of the variance within the columns or rows of a DataMatrix object. It can be either of the following: |
|   | • Row vector containing the variance values from elements in each column in DMObj (when Dim = 1) |
|   | • Column vector containing the variance values from elements in each row in DMObj (when Dim = 2) |

**Description**

\( V = \text{var}(\text{DMObj}) \) returns the variance values of the elements in the columns of a DataMatrix object, treating NaNs as missing values. The data is normalized using a sample size of \( N - 1 \), where \( N \) = the
number of elements in each column. \( V \) is a row vector containing the variance values for elements in each column in \( DMObj \). The variance is the square of the standard deviation.

\[
V = \text{var}(DMObj, \ Flag)\]

specifies how to normalize the data. If \( Flag = 0 \), normalizes using a sample size of \( N - 1 \). If \( Flag = 1 \), normalizes using a sample size of \( N \). \( N \) = the number of elements in each column or row, as specified by \( Dim \). For more information on the normalization equations, see the MATLAB function \( \text{std} \). Default \( Flag = 0 \).

\[
V = \text{var}(DMObj, Wgt)\]

computes the variance using \( Wgt \), a weight vector whose length must equal the length of the dimension over which \( \text{var} \) operates (specified by \( Dim \)). All elements in \( Wgt \) must be nonnegative. The \( \text{var} \) function normalizes \( Wgt \) to sum of 1.

\[
V = \text{var}(..., \ Dim)\]

returns the variance values of the elements in the columns or rows of a DataMatrix object, as specified by \( Dim \). If \( Dim = 1 \), returns \( V \), a row vector containing the variance values for elements in each column in \( DMObj \). If \( Dim = 2 \), returns \( V \), a column vector containing the variance values for elements in each row in \( DMObj \). Default \( Dim = 1 \).

\[
V = \text{var}(..., \ Dim, \ IgnoreNaN)\]

specifies if NaNs should be ignored. \( IgnoreNaN \) can be true (default) or false.

**See Also**
DataMatrix | mean | median | std

**Topics**
DataMatrix object on page 1-532

**Introduced in R2008b**
variableDesc

Class: bioma.data.MetaData  
Package: bioma.data

Retrieve or set variable descriptions for samples in MetaData object

Syntax

\[
\text{DSVarDescriptions} = \text{variableDesc}(\text{MDObj})  \\
\text{NewMDObj} = \text{variableDesc}(\text{MDObj}, \text{NewDSVarDescriptions})
\]

Description

\(\text{DSVarDescriptions} = \text{variableDesc}(\text{MDObj})\) returns a dataset array containing the variable names and descriptions for samples from a MetaData object.

\(\text{NewMDObj} = \text{variableDesc}(\text{MDObj}, \text{NewDSVarDescriptions})\) replaces the sample variable descriptions in \(\text{MDObj}\), a MetaData object, with \(\text{NewDSVarDescriptions}\), and returns \(\text{NewMDObj}\), a new MetaData object.

Input Arguments

\(\text{MDObj}\)

Object of the bioma.data.MetaData class.

Default:

\(\text{NewDSVarDescriptions}\)

Descriptions of the sample variable names, specified by one of the following:

- A new dataset array containing the variable names and descriptions for samples. In this dataset array, each row corresponds to a variable. The first column contains the variable name, and the second column (VariableDescription) contains a description of the variable. The row names (variable names) must match the row names (variable names) in \(\text{DSVarDescriptions}\), the dataset array being replaced in the MetaData object, \(\text{MDObj}\).

- Cell array of character vectors containing descriptions of the variables. The number of elements in \(\text{VarDesc}\) must equal the number of row names (variable names) in \(\text{DSVarDescriptions}\), the dataset array being replaced in the MetaData object, \(\text{MDObj}\).

Default:

Output Arguments

\(\text{DSVarDescriptions}\)

A dataset array containing the variable names and descriptions from a MetaData object. In this dataset array, each row corresponds to a sample variable. The first column contains the variable name, and the second column (VariableDescription) contains a description of the variable.
NewMDObj

Object of the `bioma.data.MetaData` class, returned after replacing the dataset array containing the sample variable descriptions.

**Examples**

Construct a MetaData object, and then retrieve the sample variable descriptions from it:

```matlab
% Import bioma.data package to make constructor function available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Retrieve the sample variable descriptions
VarDescriptions = variableDesc(MDObj2)
```

**See Also**

`bioma.data.MetaData | sampleNames | variableNames | variableValues`

**Topics**

“Representing Sample and Feature Metadata in MetaData Objects”
variableNames

Class: bioma.data.MetaData
Package: bioma.data

Retrieve or set variable names for samples in MetaData object

Syntax

VarNames = variableNames(MDObj)
VarNames = variableNames(MDObj, Subset)
NewMDObj = variableNames(MDObj, Subset, NewVarNames)

Description

VarNames = variableNames(MDObj) returns a cell array of character vectors specifying all variable names in a MetaData object.

VarNames = variableNames(MDObj, Subset) returns a cell array of character vectors specifying a subset the variable names in a MetaData object.

NewMDObj = variableNames(MDObj, Subset, NewVarNames) replaces the variable names specified by Subset in MDObj, a MetaData object, with NewVarNames, and returns NewMDObj, a new MetaData object.

Input Arguments

MDObj

Object of the bioma.data.MetaData class.

Subset

One of the following to specify a subset of the variable names in a MetaData object:

- Character vector or string specifying a variable name
- Cell array of character vectors or string vector specifying variable names
- Positive integer
- Vector of positive integers
- Logical vector

NewVarNames

New variable names for specific sample or feature variable names within a MetaData object, specified by one of the following:

- Numeric vector
- Cell array of character vectors or character array
- Character vector, which variableNames uses as a prefix for the variable names, with variable numbers appended to the prefix
• Logical true or false (default). If true, variableNames assigns unique variable names using the format Var1, Var2, etc.

The number of variable names in NewVarNames must equal the number of variable names specified by Subset.

**Output Arguments**

VarNames

Cell array of character vectors specifying all variable names in a MetaData object.

NewMDObj

Object of the bioma.data.MetaData class, returned after replacing the variable names.

**Examples**

Construct a MetaData object, and then retrieve the sample variable names from it:

```matlab
% Import bioma.data package to make constructor function available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Retrieve the sample variable names
VNames = variableNames(MDObj2)
```

**See Also**

bioma.data.MetaData | sampleNames | variableDesc | variableValues

**Topics**

“Representing Sample and Feature Metadata in MetaData Objects”
variableValues

Class: bioma.data.MetaData
Package: bioma.data

Retrieve or set variable values for samples in MetaData object

Syntax

\[DSVarValues = \text{variableValues}(\text{MDObj})\]
\[\text{NewMDObj} = \text{variableValues}(\text{MDObj}, \text{NewDSVarValues})\]

Description

\[DSVarValues = \text{variableValues}(\text{MDObj})\] returns a dataset array containing the measured value of each variable per sample from a MetaData object.

\[\text{NewMDObj} = \text{variableValues}(\text{MDObj}, \text{NewDSVarValues})\] replaces the sample variable values in \(\text{MDObj}\), a MetaData object, with \(\text{NewDSVarValues}\), and returns \(\text{NewMDObj}\), a new MetaData object.

Input Arguments

\textbf{MDObj}

Object of the bioma.data.MetaData class.

Default:

\textbf{NewDSVarValues}

A new dataset array containing a value for each variable per sample. In this dataset array, the columns correspond to variables and rows correspond to samples. The row names (sample names) must match the row names (sample names) in \(\text{DSVarValues}\), the dataset array being replaced in the MetaData object, \(\text{MDObj}\).

Default:

Output Arguments

\textbf{DSVarValues}

A dataset array containing the measured value of each variable per sample from a MetaData object. In this dataset array, the columns correspond to variables and rows correspond to samples.

\textbf{NewMDObj}

Object of the bioma.data.MetaData class, returned after replacing the dataset array containing the sample variable values.
Examples

Construct a MetaData object, and then retrieve the sample variable values from it:

% Import bioma.data package to make constructor function
% available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Retrieve the sample variable values
VarValues = variableValues(MDObj2)

See Also
bioma.data.MetaData | sampleNames | variableDesc | variableNames

Topics
“Representing Sample and Feature Metadata in MetaData Objects”
varValuesTable

Class: bioma.data.MetaData
Package: bioma.data

Create 2-D graphic table GUI of variable values in MetaData object

Syntax

Handle = varValuesTable(MDObj)
Handle = varValuesTable(MDObj, ParentHandle)

Description

Handle = varValuesTable(MDObj) creates a 2-D graphic table containing variable data from a MetaData object and returns a uitable handle to the table.

Handle = varValuesTable(MDObj, ParentHandle) specifies the parent handle to the table. The parent can be a figure or uipanel handle.

Input Arguments

MDObj

Object of the bioma.data.MetaData class.

Default:

ParentHandle

Figure or uipanel handle to be the parent handle to the table.

Default:

Examples

Construct a MetaData object, and then create a 2-D table of the variable values from it:

```matlab
% Import bioma.data package to make constructor function available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Retrieve the sample variable values in a table
handle = varValuesTable(MDObj2)
```
See Also
bioma.data.MetaData

Topics
“Representing Sample and Feature Metadata in MetaData Objects”
vertcat (DataMatrix)

Concateenate DataMatrix objects vertically

Syntax

\[
\begin{align*}
DMObjNew &= \text{vertcat}(DMObj1, DMObj2, \ldots) \\
DMObjNew &= (DMObj1; DMObj2; \ldots) \\
DMObjNew &= \text{vertcat}(DMObj1, B, \ldots) \\
DMObjNew &= (DMObj1, B, \ldots)
\end{align*}
\]

Input Arguments

| DMObj1, DMObj2 | DataMatrix objects, such as created by DataMatrix (object constructor). |
| B              | MATLAB numeric or logical array. |

Output Arguments

| DMObjNew | DataMatrix object created by vertical concatenation. |

Description

\(DMObjNew = \text{vertcat}(DMObj1, DMObj2, \ldots)\) or the equivalent \(DMObjNew = (DMObj1; DMObj2; \ldots)\) vertically concatenates the DataMatrix objects \(DMObj1\) and \(DMObj2\) into \(DMObjNew\), another DataMatrix object. \(DMObj1\) and \(DMObj2\) must have the same number of columns. The column names and the order of columns for \(DMObjNew\) are the same as \(DMObj1\). The column names of \(DMObj2\) and any other DataMatrix object input arguments are not preserved. The row names for \(DMObjNew\) are the row names of \(DMObj1, DMObj2,\) and other DataMatrix object input arguments.

\(DMObjNew = \text{vertcat}(DMObj1, B, \ldots)\) or the equivalent \(DMObjNew = (DMObj1, B, \ldots)\) vertically concatenates the DataMatrix object \(DMObj1\) and a numeric or logical array \(B\) into \(DMObjNew\), another DataMatrix object. \(DMObj1\) and \(B\) must have the same number of columns. The column names for \(DMObjNew\) are the same as \(DMObj1\). The column names of \(DMObj2\) and any other DataMatrix object input arguments are not preserved. The row names for \(DMObjNew\) are the row names of \(DMObj1\) and empty for the rows from \(B\).

MATLAB calls \(DMObjNew = \text{vertcat}(X1, X2, X3, \ldots)\) for the syntax \(DMObjNew = [X1; X2; X3; \ldots]\) when any one of \(X1, X2, X3,\) etc. is a DataMatrix object.

See Also

DataMatrix | horzcat

Topics

DataMatrix object on page 1-532

Introduced in R2008b
**view (biograph)**

Draw figure from biograph object

**Syntax**

```matlab
view(BGobj)
BGobjHandle = view(BGobj)
```

**Arguments**

| **BGobj**       | Biograph object created with the function biograph. |

**Description**

`view(BGobj)` opens a Figure window and draws a graph represented by a biograph object (`BGobj`). When the biograph object is already drawn in the Figure window, this function only updates the graph properties.

`BGobjHandle = view(BGobj)` returns a handle to a deep copy of the biograph object (`BGobj`) in the Figure window. When updating an existing figure, you can use the returned handle to change object properties programmatically or from the command line. When you close the Figure window, the handle is no longer valid. The original biograph object (`BGobj`) is left unchanged.

**Examples**

1. Create a biograph object.
   ```matlab
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
bg = biograph(cm)
```
2. Render the biograph object into a Handles Graphic figure and get back a handle.
   ```matlab
h = view(bg)
```
3. Change the color of all nodes and edges.
   ```matlab
set(h.Nodes,'Color',[.5 .7 1])
set(h.Edges,'LineColor',[0 0 0])
```

**See Also**

biograph | dolayout | get | getancestors | getdescendants | getedgesbynodeid | getnodesbyid | getrelatives | set | view

**Topics**

biograph object on page 1-185

**Introduced before R2006a**
**view**

Display heatmap or clustergram

**Syntax**

`view(hm_cg_object)`

**Description**

`view(hm_cg_object)` displays a heatmap or clustergram of `hm_cg_object`.

**Examples**

**Plot Heatmap of Data Matrix**

Create a matrix of data.

```matlab
data = gallery('invhess',20);
```

Display a 2-D color heatmap of the data.

```matlab
hmo = HeatMap(data);
```

```matlab
Standardize: [column | row | {none}]
Symmetric: [true | false].'
DisplayRange: 'Scalar.'
Colormap: []
ImputeFun: 'string -or- function handle -or- cell array'
ColumnLabels: 'Cell array of strings, or an empty cell array'
RowLabels: 'Cell array of strings, or an empty cell array'
ColumnLabelsRotate: []
RowLabelsRotate: []
Annotate: [on | {off}]
AnnotPrecision: []
AnnotColor: []
ColumnLabelsColor: 'A structure array.'
RowLabelsColor: 'A structure array.'
LabelsWithMarkers: [true | false].'
ColumnLabelsLocation: ['top | {bottom} ']
RowLabelsLocation: ['{left} | right ]'
```
Display the data values in the heatmap.

hmo.Annotate = true;
view(hmo)
Use the `plot` function to display the heatmap in another figure specified by the figure handle `fH`.

```matlab
fH = figure;
hA = plot(hmo,fH);
```
Use the returned axes handle `hA` to specify the axes properties.

```matlab
hA.Title.String = 'Inverse of an Upper Hessenberg Matrix';
hA.XTickLabelMode = 'auto';
hA.YTickLabelMode = 'auto';
```
Input Arguments

hm_cg_object — Heatmap or clustergram object
HeatMap object | clustergram object

Heatmap or clustergram object, specified as a HeatMap object or clustergram object.

See Also
HeatMap | clustergram

Introduced in R2009b
view (phytree)

View phylogenetic tree

Syntax

view(Tree)
view(Tree, IntNodes)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>Phylogenetic tree (phytree object) created with the function phytree.</td>
</tr>
<tr>
<td>IntNodes</td>
<td>Nodes from the phytree object to initially display in the Tree.</td>
</tr>
</tbody>
</table>

Description

view(Tree) opens the Phylogenetic Tree window and draws a tree from data in a phytree object (Tree). The significant distances between branches and nodes are in the horizontal direction. Vertical distances have no significance and are selected only for display purposes. You can access tools to edit and analyze the tree from the Phylogenetic Tree menu bar or by using the left and right mouse buttons.

view(Tree, IntNodes) opens the Phylogenetic Tree window with an initial selection of nodes specified by IntNodes. IntNodes can be a logical array of any of the following sizes: NumLeaves + NumBranches x 1, NumLeaves x 1, or NumBranches x 1. IntNodes can also be a list of indices.

Examples

View a Phylogenetic Tree

Load and view a sample phylogenetic tree.

```matlab
tr = phytreeread('pf00002.tree');
view(tr)
```
See Also
cluster | phytree | phytreeread | phytreeviewer | plot | seqlinkage | seqneighjoin

Topics
phytree object on page 1-1274

Introduced before R2006a
weights (phytree)

Calculate weights for phylogenetic tree

Syntax

\[ W = \text{weights}(\text{Tree}) \]

Arguments

| Tree | Phylogenetic tree (phytree object) created with the function phyltree. |

Description

\[ W = \text{weights}(\text{Tree}) \] calculates branch proportional weights for every leaf in a tree (Tree) using the Thompson-Higgins-Gibson method. The distance of every segment of the tree is adjusted by dividing it by the number of leaves it contains. The sequence weights are the result of normalizing to unity the new patristic distances between every leaf and the root.

Examples

1. Create an ultrametric tree with specified branch distances.

   \[
   \text{bd} = [1 \ 2 \ 3]';
   \text{tr}_1 = \text{phytree}([1 \ 2; 3 \ 4; 5 \ 6], \text{bd})
   \]

2. View the tree.

   \[
   \text{view}(\text{tr}_1)
   \]

3. Display the calculated weights.

   \[
   \text{weights}(\text{tr}_1)
   \]

   \[
   \text{ans} =
   \]
References


See Also
multialign | phytree | profalign | seqlinkage

Topics
phytree object on page 1-1274

Introduced before R2006a
**write**

Write contents of BioRead or BioMap object to file

**Syntax**

```plaintext
write(object,fName)
write(object,fName,Name,Value)
```

**Description**

`write(object,fName)` writes the contents of a BioRead or BioMap object to a file named `fName`.

`write(object,fName,Name,Value)` uses additional options specified by one or more name-value pair arguments. For example, `write(object,'data','Format','FASTQ')` saves the contents of the object in the FASTQ-formatted file `data.fastq`.

**Examples**

**Save NGS Data to a File**

Create a BioRead object from an FASTQ file.

```plaintext
BRObj = BioRead('SRR005164_1_50.fastq');
```

Extract the first 10 elements from `BRObj` and store them in a new object.

```plaintext
subsetBRObj = getSubset(BRObj, [1:10]);
```

Write the contents of the subset data to a file named `subsetData.fastq` in the local C drive. By default, the file is FASTQ-formatted because the object contains the quality data.

```plaintext
write(subsetBRObj, 'C:\subsetData');
```

**Input Arguments**

- **object** — Object containing read data  
  BioRead object | BioMap object  
  
  Object containing the read data, specified as a **BioRead** or **BioMap** object.
  
  Example: `bioreadObj`

- **fName** — Name of file  
  character vector | string  
  
  Name of the file where the contents of `object` are written, specified as a character vector or string.

  The function adds the file extension automatically depending on the type of data the object contains. If you provide the extension, the function checks for consistency between the extension and the data format of the object. The file name can be prefixed by a file path. If the path is missing, the function
writes the file to the same folder where the source file is located or to the current folder if the data is in memory.

Example: 'output'

Data Types: char

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: write(object,'data','Format','FASTQ') saves the contents of the object in the FASTQ-formatted file data.fastq.

Format — File format

'FASTQ' | 'FASTA' | 'SAM' | 'BAM'

File format, specified as a comma-separated pair consisting of 'Format' and a character vector or string.

For BioRead objects, available formats are 'FASTA' and 'FASTQ'. The default format is 'FASTA' if the object does not contain qualities, that is, the Quality property is empty. Otherwise, the default format is 'FASTQ'.

For BioMap objects, available formats are 'FASTA', 'FASTQ', 'SAM', and 'BAM' (default).

Example: 'Format','FASTA'

Data Types: char

Overwrite — Boolean indicator to overwrite existing file

false (default) | true

Boolean indicator to overwrite an existing file, specified as a comma-separated pair consisting of 'Overwrite' and true or false. If true, the function overwrites the file and deletes any respective index file (*.idx,*.bai,*linearindex) or ordered file (*.ordered.bam, *.ordered.sam) that is no longer needed.

Example: 'Overwrite',true

Data Types: logical

See Also

BioMap | BioRead

Topics

"Manage Sequence Read Data in Objects"

Introduced in R2010a
zonebackadj

Perform background adjustment on Affymetrix microarray probe-level data using zone-based method

Syntax

BackAdjustedData = zonebackadj(Data)
[BackAdjustedData, ZoneStruct] = zonebackadj(Data)
[BackAdjustedData, ZoneStruct, Background] = zonebackadj(Data)

... = zonebackadj(Data, ...'NumZones', NumZonesValue, ...)
... = zonebackadj(Data, ...'Percent', PercentValue, ...)
... = zonebackadj(Data, ...'SmoothFactor', SmoothFactorValue, ...)
... = zonebackadj(Data, ...'NoiseFrac', NoiseFracValue, ...)
... = zonebackadj(Data, ...'CDF', CDFValue, ...)
... = zonebackadj(Data, ...'Mask', MaskValue, ...)
... = zonebackadj(Data, ...'Showplot', ShowplotValue, ...)

Input Arguments

Data | Either of the following:
- MATLAB structure containing probe intensities from an Affymetrix CEL file, such as returned by affyread when used to read a CEL file.
- Array of MATLAB structures containing probe intensities from multiple Affymetrix CEL files.

NumZonesValue | Scalar or two-element vector that specifies the number of zones to use in the background adjustment. If a scalar, it must be a square number. If a two-element vector, the first element specifies the number of rows and the second element specifies the number of columns in a nonsquare grid. Default is 16.

PercentValue | Value that specifies a percentage, P, such that the lowest P percent of ranked intensity values from each zone is used to estimate the background for that zone. Default is 2.

SmoothFactorValue | Value that specifies the smoothing factor used in the calculation of the weighted average of the contributions of each zone to the background of a point. Default is 100.

NoiseFracValue | Value that specifies the noise fraction, NF, such that the background-adjusted value is given by \( \max((Intensity - WeightedBackground), NF*LocalNoiseEstimate) \). Default is 0.5.
CDFValue

Either of the following:

- Character vector or string specifying a file name or path and file name of an Affymetrix CDF library file. If you specify only a file name, the file must be on the MATLAB search path or in the current folder.
- MATLAB structure containing information from an Affymetrix CDF library file, such as returned by affyread when used to read a CDF file.

The CDF library file or structure specifies control cells, which are not used in the background estimates.

MaskValue

Logical vector that specifies which cells to mask and not use in the background estimates. In the vector, 0 = not masked and 1 = masked. Defaults are the values in the Masked column of the Probes field of the CEL file.

ShowplotValue

Controls the plotting of an image of the background estimates. Choices are true or false (default).

Output Arguments

| BackAdjustedData | Matrix or cell array of vectors containing background-adjusted probe intensity values. |
| ZoneStruct       | MATLAB structure containing the centers of the zones used to perform the background adjustment and the estimates of the background values at the center of each zone. |
| Background       | Matrix or cell array of vectors containing the estimated background values for each probe. |

Description

BackAdjustedData = zonebackadj(Data) returns the background-adjusted probe intensities from Data, which contains probe intensities from Affymetrix CEL files. Details of the background adjustment are described in Statistical Algorithms Description Document.

[BackAdjustedData, ZoneStruct] = zonebackadj(Data) also returns a structure containing the centers of the zones used to perform the background adjustment and the estimates of the background values at the center of each zone.

[BackAdjustedData, ZoneStruct, Background] = zonebackadj(Data) also returns a matrix or cell array of vectors containing the estimated background values for each probe.

... = zonebackadj(Data, ...'PropertyName', PropertyValue, ...) calls zonebackadj with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = zonebackadj(Data, ...'NumZones', NumZonesValue, ...) specifies the number of zones to use in the background adjustment. NumZonesValue can be either a scalar that is a square
number or a two-element array in which the first element specifies the number of rows and the second element specifies the number of columns in a nonsquare grid. Default is 16.

\[ \ldots = \text{zonebackadj}(\text{Data}, \ldots, 'Percent', \text{PercentValue}, \ldots) \] specifies a percentage, \( P \), such that the lowest \( P \) percent of ranked intensity values from each zone is used to estimate the background for that zone. Default is 2.

\[ \ldots = \text{zonebackadj}(\text{Data}, \ldots, 'SmoothFactor', \text{SmoothFactorValue}, \ldots) \] specifies the smoothing factor used in the calculation of the weighted average of the contributions of each zone to the background of a point, thus providing a smooth transition between zones. Default is 100.

\[ \ldots = \text{zonebackadj}(\text{Data}, \ldots, 'NoiseFrac', \text{NoiseFracValue}, \ldots) \] specifies the noise fraction, such that the background-adjusted value is given by \( \max((\text{Intensity} - \text{WeightedBackground}), \text{NF} \times \text{LocalNoiseEstimate}) \), where NF is NoiseFracValue. Default is 0.5.

\[ \ldots = \text{zonebackadj}(\text{Data}, \ldots, 'CDF', \text{CDFValue}, \ldots) \] specifies an Affymetrix CDF library file or structure, which specifies control cells, which are not used in the background estimates.

\[ \ldots = \text{zonebackadj}(\text{Data}, \ldots, 'Mask', \text{MaskValue}, \ldots) \] specifies a logical vector of that specifies which cells to mask and not use in the background estimates. In the vector, 0 = not masked and 1 = masked. Defaults are the values in the Masked column of the Probes field of the CEL file.

\[ \ldots = \text{zonebackadj}(\text{Data}, \ldots, 'Showplot', \text{ShowplotValue}, \ldots) \] plots an image of the background estimates. Choices are true or false (default).
Examples

Retrieve Affymetrix™ Probe Set Intensity Values

This example uses sample data from the E. coli Antisense Genome Array. Download the data from Demo_Data_Ecoli-antisense.zip. Extract the data files from the DTT archive using the Data Transfer Tool.

You also need to download Ecoli_ASv2.CDF library file for the E. coli Antisense Genome Array. You may already have these files if you have any Affymetrix GeneChip software installed on your machine. If not, get the library files by downloading and unzipping the E. coli Antisense Genome Array zip file.

Read the contents of a CEL file into a MATLAB structure.

celStruct = affyread('Ecoli-antisense-121502.CEL');

Read the contents of a CDF file into a MATLAB structure.

cdfStruct = affyread('C:\LibFiles\Ecoli_ASv2.CDF');

Use the zonebackadj function to return a matrix or cell array of vectors containing the estimated background values for each probe.

[baData,zones,background] = zonebackadj(celStruct,'cdf',cdfStruct);

Create a table of intensity values for the argG_b3172_at probe set.

psvals = probesetvalues(celStruct, cdfStruct, 'argG_b3172_at', ...
                          'background',background)

pvals =

    1.0e+03 *

    Columns 1 through 7
         5.2120    0.0010         0         0         0.0454    0.4300    0.1770    0.1690
         5.2120    0.0020         0         0         0.0455    0.4310    0.1770    0.1273
         5.2120    0.0030         0         0         0.0455    0.4320    0.1770    0.1270
         5.2120    0.0040         0         0         0.0455    0.4330    0.1770    0.1333
         5.2120    0.0050         0         0         0.0455    0.4340    0.1770    0.2123
         5.2120    0.0060         0         0         0.0455    0.4350    0.1770    0.1495
         5.2120    0.0070         0         0         0.0455    0.4360    0.1770    0.0503
         5.2120    0.0080         0         0         0.0456    0.4370    0.1770    0.1525
         5.2120    0.0090         0         0         0.0456    0.4380    0.1770    0.1645
         5.2120    0.0100         0         0         0.0456    0.4390    0.1770    0.1260
         5.2120    0.0110         0         0         0.0456    0.4400    0.1770    0.0540
         5.2120    0.0120         0         0         0.0456    0.4410    0.1770    0.0833
         5.2120    0.0130         0         0         0.0457    0.4420    0.1770    0.0955
         5.2120    0.0140         0         0         0.0457    0.4430    0.1770    0.1100

    Columns 8 through 14
           0.0354    0.0250         0         0         0.4300    0.1780    0.1635
           0.0218    0.0300         0         0         0.4310    0.1780    0.1083
           0.0237    0.0300         0         0         0.4320    0.1780    0.1750
0.0259    0.0360    0    0    0.4330    0.1780    0.0940
0.0433    0.0360    0    0    0.4340    0.1780    0.1718
0.0275    0.0360    0    0    0.4350    0.1780    0.1540
0.0112    0.0300    0    0    0.4360    0.1780    0.0460
0.0377    0.0360    0    0    0.4370    0.1780    0.1070
0.0312    0.0360    0    0    0.4380    0.1780    0.0973
0.0234    0.0360    0    0    0.4390    0.1780    0.1213
0.0112    0.0360    0    0    0.4400    0.1780    0.0540
0.0174    0.0360    0    0    0.4410    0.1780    0.0623
0.0171    0.0300    0    0    0.4420    0.1780    0.0840
0.0196    0.0360    0    0    0.4430    0.1780    0.0925
0.0460    0.0360    0    0    0.4440    0.1780    0.1118

Columns 15 through 20
0.0241    0.0300    0    0    0.0010    0.0020
0.0146    0.0360    0    0    0.0010    0.0020
0.0286    0.0360    0    0    0.0010    0.0020
0.0227    0.0300    0    0    0.0010    0.0020
0.0365    0.0300    0    0    0.0010    0.0020
0.0303    0.0300    0    0    0.0010    0.0020
0.0098    0.0250    0    0    0.0010    0.0020
0.0210    0.0360    0    0    0.0010    0.0020
0.0219    0.0360    0    0    0.0010    0.0020
0.0253    0.0360    0    0    0.0010    0.0020
0.0129    0.0360    0    0    0.0010    0.0020
0.0125    0.0360    0    0    0.0010    0.0020
0.0186    0.0300    0    0    0.0010    0.0020
0.0220    0.0360    0    0    0.0010    0.0020
0.0207    0.0360    0    0    0.0010    0.0020

References


See Also

affyinvarsetnorm | affyread | celintensityread | gcrma | gcrmabackadj |
probelibraryinfo | probesetlink | probesetlookup | probesetvalues | quantilenorm |
rmbackadj | rmam summary

Topics

“Working with Affymetrix® Data”

Introduced in R2007b
**date property**

**Class:** geneont

Read-only character vector containing date and time OBO file was last updated

**Description**

date is a read-only property of the geneont class. date is a character vector containing the date and time the OBO file was last updated. The OBO file is the Open Biomedical Ontology file from which the geneont object was created.

**Values**

Possible values are any date and time the OBO file was updated. Use this date information to compare the dates associated with ontologies used to create various geneont objects.

**Examples**

1. Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.
   
   ```matlab
   GeneontObj = geneont('LIVE', true)
   ```
   
   The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.
   
   **Gene Ontology object with 27769 Terms.**

2. Display the date and time associated with the OBO file used to create the geneont object.

   ```matlab
   GeneontObj.date
   ```

   ```matlab
   ans =
   02:12:2008 19:30
   ```

**See Also**
geneont.format_version
default_namespace property

Class: geneont

Read-only character vector containing namespace to which GO terms are assigned

Description

default_namespace is a read-only property of the geneont class. default_namespace is a character vector containing the ontology namespace to which the GO terms are assigned.

Values

Currently, gene_ontology is the only possible namespace. However, other namespaces may be used in the future. Use this namespace information to determine the ontology namespace to which the GO terms in a geneont object are assigned.

Examples

1  Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.
    GeneontObj = geneont('LIVE', true)
    The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.
    Gene Ontology object with 27769 Terms.

2  Display the namespace associated with the GO terms of the geneont object.
    GeneontObj.default_namespace
    ans =
    gene_ontology
**format_version property**

Class: geneont

Read-only character vector containing version of encoding of OBO file

**Description**

format_version is a read-only property of the geneont class. format_version is a character vector containing the version of the encoding of the OBO file. The OBO file is the Open Biomedical Ontology file from which the geneont object was created.

**Values**

Possible values are the current or previous versions of the OBO file. Use this version information to compare the version associated with OBO file used to create various geneont objects.

**Examples**

1. Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

   GeneontObj = geneont('LIVE', true)

   The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

   Gene Ontology object with 27769 Terms.

2. Display the version of the OBO file used to create the geneont object.

   GeneontObj.format_version

   ans =

   1.0
terms property

Class: geneont

Read-only column vector with handles to term objects of geneont object

Description

terms is a read-only property of the geneont class. terms is a column vector with handles to the term objects of a geneont object.

Note  Although terms is a column vector with handles to term objects, in the MATLAB Command Window, it displays as a structure array, with one structure for each GO term in the geneont object.

Values

Use the information in this structure to access (by GO ID) the terms of a geneont object and to view the properties of individual terms.

Examples

1  Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

   GeneontObj = geneont('LIVE', true)

   The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

   Gene Ontology object with 27786 Terms.

2  Use the terms property to display the MATLAB structure array containing 27,786 term objects associated with the geneont object.

   GeneontObj.terms

   27786x1 struct array with fields:
   id
   name
   ontology
   definition
   comment
   synonym
   is_a
   part_of
   obsolete

Note  Although the terms property is an array of handles to term objects, in the MATLAB Command Window, it displays as a structure array, with one structure for each GO term in the geneont object.
3 Use the terms property to view the properties of the term object in the 14,723rd position in the geneont object.

```
GeneontObj.terms(14723)
```

```
id: 31655
    name: 'negative regulation of heat dissipation'
    ontology: 'biological process'
    definition: [1x113 char]
    comment: ''
    synonym: {4x2 cell}
    is_a: [3x1 double]
    part_of: 31653
    obsolete: 0
```

4 Create a cell array of character vectors that list the ontology property for each term in the geneont object.

```
ontologies = get(GeneontObj.terms,'ontology');
```

5 Create a logical mask that identifies all the terms with an ontology property of cellular component.

```
mask = strcmp(ontologies,'cellular component');
```

6 Apply the logical mask to all the terms in the GeneontObj geneont object to return a MATLAB structure array of term objects, containing only terms with an ontology property of cellular component.

```
cell_comp_terms = GeneontObj.terms(mask)
```

```
2362x1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
    part_of
    obsolete
```

There are 2,362 terms with an ontology property of cellular component.

7 Create a subontology of all the cellular component terms by indexing into the GeneontObj geneont object with the masked term objects.

```
subontology = GeneontObj(cell_comp_terms)
```

```
Gene Ontology object with 2367 Terms.
```

See Also

term
definition property

Class: term

Read-only character vector that defines GO term

Description

definition is a read-only property of the term class. definition is a character vector that defines the GO term.

Values

Possible values are any definition used for a term in the Gene Ontology database. Use the definition property to determine definitions of term objects, or to access or filter term objects by definition.

Examples

Example 1.85. Using the definition Property to Determine the Definition of a term Object

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

   GeneontObj = geneont('LIVE', true)

   The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

   Gene Ontology object with 27769 Terms.

2 Display the definition of the term object in the 287th position in the geneont object, GeneontObj.

   GeneontObj.terms(287).name

   ans =

   "The smaller of the two subunits of an organellar ribosome." [GOC:mcc]

Tip If you know the GO identifier (for example, 314) of a term object, instead of its index or position number (for example, 287), you can use the following syntax to display the definition of a term object:

   GeneontObj(314).terms.definition

For help converting the index or position number of a term object to its GO identifier, see the term.id property.

Example 1.86. Filtering term Objects by Text in Their Definitions

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.
GeneontObj = geneont('LIVE', true)

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

Gene Ontology object with 27769 Terms.

Display the structure array containing 27,786 term objects associated with the geneont object.

GeneontObj.terms

27786x1 struct array with fields:
  id
  name
  ontology
  definition
  comment
  synonym
  is_a
  part_of
  obsolete

3 Find term objects whose definitions include the phrase “ceramide oligosaccharides” by first creating a cell array of character vectors that list the definition property for each term in the geneont object.

definitions = get(GeneontObj.terms,'definition');

4 Use the regexpi function to search these character vectors for ‘ceramide oligosaccharides’.

matches = regexpi(definitions,'ceramide oligosaccharides','once');

5 Create a logical mask that identifies all the terms with a definition property that includes the phrase “ceramide oligosaccharides.”

mask = ~cellfun(@isempty,matches);

6 Apply the logical mask to all the terms in the GeneontObj geneont object to return a structure containing the GO identifiers of terms with a definition that includes the phrase “ceramide oligosaccharides.”

get(GO.terms(mask),'id')

ans =
  [1573]
  [1574]

7 Apply the logical mask to all the terms in the GeneontObj geneont object to return a structure containing the full definitions of terms with a definition that includes the phrase “ceramide oligosaccharides.”

char(get(GO.terms(mask),'definition'))
id property

Class: term

Read-only numeric value that corresponds to GO identifier of GO term

Description

id is a property of the term class. id is a read-only numeric value that corresponds to the GO identifier of the GO term.

Tip You can use the num2goid function to convert id to a GO ID character vector formatted as a 7-digit number preceded by the prefix GO:, which is the standard used by the Gene Ontology database.

Values

Any value from 1 to N, where N is the largest value for an identifier of a term object in a geneont object. Use the id property to determine GO identifiers of term objects, or to access term objects by their GO identifier.

Tip You can use the id property for a GO term as input to methods of a geneont object, such as getancestors, getdescendants, and getrelatives.

Examples

Example 1.87. Displaying and Formatting the GO Identifier of a term Object

1  Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

   GeneontObj = geneont('LIVE', true)

   The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

   Gene Ontology object with 27769 Terms.

2  Display the GO identifier of the term object in the 183rd position in the geneont object, GeneontObj.

   GeneontObj.terms(183).id

   ans =

   207

Note The index or position (183 in this example) of the term object in the geneont object is not the same as the GO identifier (207 in this example) for the term object. This is because there are many terms that are obsolete and are not included as term objects in the geneont object.
Number the GO identifier into a character array.

\texttt{num2goid(GeneontObj.terms(183).id)}

\texttt{ans =}

\texttt{'GO:0000207'}

**Example 1.88. Using the GO Identifier with Methods of a geneont Object**

1. Find the index or position number of the term object whose \texttt{name} property is \texttt{'membrane'}.

\texttt{membrane_index = find(strcmp(get(GeneontObj.terms,'name'),'membrane'))}

\texttt{membrane_index =}

\texttt{9556}

2. Use this index or position number and the \texttt{id} property to determine the GO identifier of the term object.

\texttt{membrane_goid = GeneontObj.terms(membrane_index).id}

\texttt{membrane_goid =}

\texttt{16020}

3. Use this GO identifier as input to the \texttt{getrelatives} method to find the GO identifiers of other term objects that are immediate relatives of the term object whose \texttt{name} property is \texttt{'membrane'}.

\texttt{relative_ids = getrelatives(GeneontObj,membrane_goid)}

\texttt{relative_ids =}

\texttt{5628}
\texttt{5886}
\texttt{16020}
\texttt{19867}
\texttt{30673}
\texttt{31090}
\texttt{34045}
\texttt{34357}
\texttt{42175}
\texttt{42622}
\texttt{42734}
\texttt{44464}
\texttt{45211}
\texttt{48475}
\texttt{60342}

4. List the \texttt{name} properties of these term objects.

\texttt{get(GeneontObj(relative_ids).terms,'name')}

\texttt{ans =}

\texttt{'prospore membrane'}
\texttt{'plasma membrane'}
\texttt{'membrane'}
\texttt{'outer membrane'}
'axolemma'
'organelle membrane'
'pre-autophagosomal structure membrane'
'photosynthetic membrane'
'nuclear envelope-endoplasmic reticulum network'
'photoreceptor outer segment membrane'
'presynaptic membrane'
'cell part'
'postsynaptic membrane'
'coated membrane'
'photoreceptor inner segment membrane'

See Also
getancestors | getdescendants | getrelatives | num2goid
is_a property

Class: term

Read-only numeric array containing GO identifiers of GO terms that have an “is a” relationship with this GO term.

Description

is_a is a read-only property of the term class. is_a is a column vector containing GO identifiers. These GO identifiers specify other term objects to which the term object has an “is a” relationship. The term object is an example of the term objects specified by its is_a property.

Values

Possible values are identifiers of GO terms from the Gene Ontology database. Use the is_a property to determine GO identifiers of GO terms that have an “is a” relationship with a specific GO term.

Examples

Example 1.89. Using the is_a Property to Determine term Objects with an “is a” Relationship

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

GeneontObj = geneont('LIVE', true)

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

Gene Ontology object with 27769 Terms.

2 Display the term objects to which the term object in the 18,703rd position has an “is a” relationship.

GeneontObj.terms(18703).is_a

ans =

    42754
    45187
    48521
    51241

Tip You can also use the getancestors method of a geneont object with the 'Relationtype' property set to 'is_a' to determine term objects with an "is a" relationship.

Tip If you know the GO identifier (for example, 42321) of a term object, instead of its index or position number (for example, 18703), you can use the following syntax to display the is_a property of a term object:
GeneontObj(42321).terms.is_a

For help converting the index or position number of a term object to its GO identifier, see the term.id property.

See Also
getancestors | getdescendants | getrelatives
name property

Class: term

Read-only character vector representing name of GO term

Description

name is a read-only property of the term class. name is a character vector representing the name of the GO term.

Values

Possible values are any name used for a term in the Gene Ontology database. Use the name property to determine names of term objects, or to access or filter term objects by name.

Examples

Example 1.90. Using the name Property to Determine the Name of a term Object

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

    GeneontObj = geneont('LIVE', true)

    The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

    Gene Ontology object with 27769 Terms.

2 Display the name of the term object in the 157th position in the geneont object, GeneontObj.

    GeneontObj.terms(157).name

    ans =

        cytosolic small ribosomal subunit

Tip If you know the GO identifier (for example, 181) of a term object, instead of its index or position number (for example, 157), you can use the following syntax to display the name of a term object:

    GeneontObj(181).terms.name

For help converting the index or position number of a term object to its GO identifier, see the term.id property.

Example 1.91. Using the name Property to Find and Display Specific term Objects

1 Find the index or position number of the term object whose name property is 'membrane'.

    membrane_index = find(strcmp(get(GeneontObj.terms,'name'),'membrane'))
membrane_index = 9556

2 Use this index or position number and the id property to determine the GO identifier of the term object.

    membrane_goid = GeneontObj.terms(membrane_index).id
    membrane_goid = 16020

3 Use this GO identifier as input to the getrelatives method to find the GO identifiers of other term objects that are immediate relatives of the term object whose name property is 'membrane'.

    relative_ids = getrelatives(GeneontObj, membrane_goid)
    relative_ids =
        5628
        5886
        16020
        19867
        30673
        31090
        34045
        34357
        42175
        42622
        42734
        44464
        45211
        48475
        60342

4 List the name properties of these term objects.

    get(GeneontObj(relative_ids).terms,'name')
    ans =
        'prospore membrane'
        'plasma membrane'
        'membrane'
        'outer membrane'
        'axolemma'
        'organelle membrane'
        'pre-autophagosomal structure membrane'
        'photosynthetic membrane'
        'nuclear envelope-endoplasmic reticulum network'
        'photoreceptor outer segment membrane'
        'presynaptic membrane'
        'cell part'
        'postsynaptic membrane'
        'coated membrane'
        'photoreceptor inner segment membrane'
obsolete property

Class: term

Read-only Boolean value that indicates whether a GO term is obsolete

Description

obsolete is a read-only property of the term class. obsolete is a Boolean value that indicates if the GO term is obsolete (1) or not obsolete (0).

Values

1 — Obsolete
0 — Not obsolete

Use the obsolete property to determine whether a term object is obsolete, or to access or filter term objects by obsolete value.

Examples

Example 1.92. Using the obsolete Property to Determine the Obsolete Status of a term Object

1. Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.
   
   ```matlab
   GeneontObj = geneont('LIVE', true)
   ```
   
   The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.
   
   Gene Ontology object with 27769 Terms.

2. Display the obsolete status of the term object in the third and seventh positions in the geneont object, G0
   
   ```matlab
   GeneontObj.terms(3).obsolete
   ans =
   0
   GeneontObj.terms(7).obsolete
   ans =
   1
   ```

   **Tip** If you know the GO identifier (for example, 8) of a term object, instead of its index or position number (for example, 7), you can use the following syntax to display the obsolete status of a term object:

   ```matlab
   GeneontObj(8).terms.obsolete
   ```
For help converting the index or position number of a term object to its GO identifier, see the term.id property.

Example 1.93. Filtering term Objects by Obsolete Status

1. Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.
   
   ```matlab
   GeneontObj = geneont('LIVE', true)
   ```
   
   The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.
   
   Gene Ontology object with 27769 Terms.

2. Display the structure array containing 27,786 term objects associated with the geneont object.
   
   ```matlab
   GeneontObj.terms
   ```
   
   27786x1 struct array with fields:
   - id
   - name
   - ontology
   - definition
   - comment
   - synonym
   - is_a
   - part_of
   - obsolete

3. Create a cell array of logicals that list the obsolete property for each term in the geneont object.
   
   ```matlab
   obsolescence = get(GeneontObj.terms,'obsolete');
   ```

4. Create a logical mask from the cell array that identifies all the nonobsolete terms.
   
   ```matlab
   mask = ~cell2mat(obsolescence);
   ```

5. Apply the logical mask to all the terms in the GeneontObj geneont object to return a structure containing only terms that are not obsolete.
   
   ```matlab
   nonobsolete_terms = GeneontObj.terms(mask)
   ```
   
   26424x1 struct array with fields:
   - id
   - name
   - ontology
   - definition
   - comment
   - synonym
   - is_a
   - part_of
   - obsolete

   There are 26,424 terms that are not obsolete.
ontology property

Class: term

Read-only character vector describing the ontology of GO term

Description

ontology is a read-only property of the term class. ontology is a character vector describing the ontology of the GO term.

Values

'molecular function'
'biological process'
'cellular component'

Use the ontology property to determine the ontology of term objects, or to access or filter term objects by ontology.

Examples

Example 1.94. Using the ontology Property to Determine the Ontology of a term Object

1  Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

   GeneontObj = geneont('LIVE', true)

   The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

   Gene Ontology object with 27769 Terms.

2  Display the ontology of the term object in the 155th position in the geneont object, GeneontObj.

   GeneontObj.terms(155).ontology
   ans =
   molecular function

Tip  If you know the GO identifier (for example, 179) of a term object, instead of its index or position number (for example, 155), you can use the following syntax to display the ontology of a term object:

   GeneontObj(179).terms.ontology

For help converting the index or position number of a term object to its GO identifier, see the term.id property.
Example 1.95. Filtering term Objects by Cellular Component Ontology

1. Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

   GeneontObj = geneont('LIVE', true)

   The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

   Gene Ontology object with 27769 Terms.

2. Display the structure array containing 27,786 term objects associated with the geneont object.

   GeneontObj.terms

27786x1 struct array with fields:
   id
   name
   ontology
   definition
   comment
   synonym
   is_a
   part_of
   obsolete

3. View the properties of the term object in the 14,723rd position in the geneont object.

   GeneontObj.terms(14723)

   id: 31655
    name: 'negative regulation of heat dissipation'
    ontology: 'biological process'
    definition: [1x113 char]
    comment: ''
    synonym: {4x2 cell}
    is_a: [3x1 double]
    part_of: 31653
    obsolete: 0

4. Create a cell array of character vectors that list the ontology property for each term in the geneont object.

   ontologies = get(GeneontObj.terms,'ontology');

5. Create a logical mask that identifies all the terms with an ontology property of cellular component.

   mask = strcmp(ontologies,'cellular component');

6. Apply the logical mask to all the terms in the GeneontObj geneont object to return a structure containing only terms with an ontology property of cellular component.

   cell_comp_terms = GeneontObj.terms(mask)

   2362x1 struct array with fields:
   id
   name
   ontology
   definition
comment
synonym
is_a
part_of
obsolete

There are 2,362 terms with an ontology property of cellular component.
part_of property

Class: term

Read-only numeric array containing GO identifiers of GO terms that have a “part of” relationship with this GO term

Description

part_of is a read-only property of the term class. part_of is a column vector containing GO identifiers. These GO identifiers specify other term objects to which the term object has a “part_of” relationship. The term object is a subset of the term objects specified by its part_of property.

Values

Possible values are identifiers of GO terms from the Gene Ontology database. Use the part_of property to determine GO identifiers of GO terms that have a “part of” relationship with a specific GO term.

Examples

Example 1.96. Using the part_of Property to Determine term Objects with a “part of” Relationship

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

   GeneontObj = geneont('LIVE', true)

   The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

   Gene Ontology object with 27769 Terms.

2 Display the term objects to which the term object in the 18,703rd position has a “part of” relationship.

   GeneontObj.terms(18703).part_of

   ans =

         50802

Tip You can also use the getancestors method of a geneont object with the 'Relationtype' property set to 'part_of' to determine term objects with a “part of” relationship.

Tip If you know the GO identifier (for example, 42321) of a term object, instead of its index or position number (for example, 18703), you can use the following syntax to display the part_of property of a term object:

   GeneontObj(42321).terms.part_of
For help converting the index or position number of a term object to its GO identifier, see the term.id property.

**See Also**
getancestors | getdescendants | getrelatives
synonym property

Class: term

Read-only array containing GO terms that are synonyms of this GO term

Description

synonym is a read-only property of the term class. synonym is a two-column cell array containing GO terms that are synonyms of this GO term. The first column contains a character vector specifying the type of synonym, such as 'exact_synonym', 'related_synonym', 'broad_synonym', 'narrow_synonym', or 'alt_id'. The second column contains the GO identifier of the synonymous term or a character vector describing the synonymous term.

Values

Possible values are identifiers of GO terms from the Gene Ontology database. Use the synonym property to determine GO identifiers of synonymous term objects.

Examples

Example 1.97. Using the synonym Property to Determine Synonymous term Objects

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

GeneontObj = geneont('LIVE', true)

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

Gene Ontology object with 27769 Terms.

2 Display the term objects that are synonymous to the term object in the third position in the geneont object, GeneontObj.

synonyms = GeneontObj.terms(3).synonym

synonyms =

'alt_id'           'GO:0019952'
'alt_id'           'GO:0050876'
'exact_synonym'     [1x39 char]

3 Because the exact synonym does not have a GO identifier listed, display the text of this synonym.

synonyms(3,2)

ans =

""reproductive physiological process" []"

4 Display the term objects that are synonymous to the term object in the 352nd position in the geneont object, GeneontObj.
GeneontObj.terms(352).synonym

ans =

'alt_id' 'GO:0006374'
'alt_id' 'GO:0006375'
'related_synonym' [1x26 char]
'related_synonym' [1x26 char]
'narrow_synonym' [1x51 char]
'narrow_synonym' [1x50 char]
'broad_synonym' "mRNA splicing" []
'broad_synonym' [1x22 char]

Tip If you know the GO identifier (for example, 398) of a term object, instead of its index or position number (for example, 352), you can use the following syntax to display the synonym of a term object:

GeneontObj(398).terms.synonym

For help converting the index or position number of a term object to its GO identifier, see the term.id property.
Genomics Viewer

View NGS sequences and annotations

Description

The Genomics Viewer app lets you view and explore integrated genomic data with an embedded version of the Integrative Genomics Viewer (IGV) [1][2]. The genomic data include NGS read alignments, genome variants, and segmented copy number data.

Using the app, you can:

• Visualize short-read data (.BAM or .CRAM) aligned to a reference sequence and compare multiple data sets aligned to a common reference sequence.
• View coverage of different regions of the reference sequence.
• Investigate quality and other details of aligned reads.
• Display nonquantitative genome annotations (.BED, .GFF, .GFF3, and .GTF).
• Load structural variants (.VCF) and visualize genetic alterations, such as insertions and deletions.
• View segmented copy number data (.SEG) and quantitative genomic data (.WIG, .BIGWIG, and .BEDGRAPH), such as ChIP peaks and alignment coverage.

The app requires an internet connection.

Open the Genomics Viewer App

• MATLAB Toolstrip: On the Apps tab, under Computational Biology, click the app icon.
• MATLAB command prompt: Enter genomicsViewer.

Examples

• “Visualize NGS Data Using Genomics Viewer App”

References


See Also

Apps
Sequence Alignment | Sequence Viewer
Topics
“Visualize NGS Data Using Genomics Viewer App”

Introduced in R2019b
Molecule Viewer
Display and manipulate 3-D molecule structure

Description
The Molecule Viewer app lets you display and manipulate 3-D molecular structures.
You can:

• Import structural information directly from the Protein Data Bank (PDB) database or other supported files.
• Measure distances and dihedral angles.
• Display molecular surfaces, such as van der Waals or solvent-accessible surfaces.
• Select different visualization and color schemes to display a molecule, such as the ribbon or backbone representation.
• Run RasMol script commands from within the app.

Open the Molecule Viewer App

• MATLAB Toolstrip: On the Apps tab, under Computational Biology, click the app icon.
• MATLAB command prompt: Enter molviewer.

Examples

View 3-D structure of molecule
Display the 3-D structure of an acetylsalicylic acid (aspirin) molecule.

f = molviewer('aspirin.mol');
f.HandleVisibility = 'off';
Programmatic Use

molviewer opens the Molecule Viewer app.

`molviewer(file)` reads the structural information from `file` and shows the 3-D molecular structure in the Molecule Viewer app.

`molviewer(pdbID)` retrieves the structural data for a protein from the PDB database using its `pdbID` and shows the 3-D molecular structure in the Molecule Viewer app.

See Also

Functions

molviewer

Introduced in R2007a
NGS Browser

(To be removed) Visualize and explore alignments

Note  The app will be removed in a future release. Use Genomics Viewer instead.

Description
The NGS Browser app lets you visualize and explore the alignment of reads to a reference sequence.

You can:
- Visualize short-read data aligned to a nucleotide reference sequence, and compare multiple data sets aligned against a common reference sequence.
- View coverage of different regions of the reference sequence.
- Investigate quality and other details of aligned reads.
- Identify polymorphisms and visualize insertions and deletions.
- Retrieve feature annotations relative to a specific region of the reference sequence.

Open the NGS Browser App
- MATLAB Toolstrip: On the Apps tab, under Computational Biology, click the app icon.
- MATLAB command prompt: Enter ngsbrowser.

Examples

View Sequence Data from BioMap object in NGS Browser
Create a BioMap object from a SAM-formatted file.

b = BioMap('ex1.sam');

Display the object in the NGS Browser.

ngsbrowser(b)

“Visualize and Investigate Sequence Read Alignments”

Programmatic Use
ngsbrowser opens the NGS Browser app.

ngsbrowser(biomapObj) imports a BioMap object biomapObj containing sequence alignment information and opens it in the app.
Compatibility Considerations

**NGS Browser will be removed**

*Warns starting in R2019b*

The app will be removed in a future release. Use **Genomics Viewer** instead.

See Also

**Apps**

Genomics Viewer | Sequence Alignment | Sequence Viewer

**Topics**

“Visualize and Investigate Sequence Read Alignments”

**Introduced in R2011b**
Sequence Alignment

Visualize and edit multiple sequence alignments

Description

The Sequence Alignment app lets you visualize and edit multiple sequence alignments.

You can:

• Inspect the sequence alignment and make manual adjustments.
• View the consequence sequence information and export it to a file or MATLAB workspace.
• Generate a phylogenetic tree from aligned sequences.

Open the Sequence Alignment App

• MATLAB Toolstrip: On the Apps tab, under Computational Biology, click the app icon.
• MATLAB command prompt: Enter seqalignviewer.

Examples

View a Multiple Sequence Alignment File

Load and view a multiple sequence alignment file.

seqalignviewer('aagag.aln')
Alternatively, you can click Sequence Alignment on the **Apps** tab to open the app, and view the alignment data.

You can also generate a phylogenetic tree from aligned sequences from within the app. Select **Display > View Tree**.

- “View and Align Multiple Sequences”

**Programmatic Use**

`seqalignviewer` opens the **Sequence Alignment** app.

`seqalignviewer(Alignment)` loads multiple sequence alignment data `Alignment` into the app. Alignment can be one of the following:

- A MATLAB structure containing a `Sequence` field, such as returned by `fastaread`, `gethmmalignment`, `multialign`, or `multialignread`
- A MATLAB character array containing MSA data, such as returned by `multialign`
- A string specifying a file or URL that contains MSA data
• A 3-by-N character array showing the pairwise alignment of two sequences, such as the array returned by `nwalign` or `swalign`.

`seqalignviewer('close')` closes the app.

**See Also**

<table>
<thead>
<tr>
<th>Apps</th>
<th>NGS Browser</th>
<th>Sequence Viewer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions</td>
<td><code>seqalignviewer</code></td>
<td></td>
</tr>
<tr>
<td>Topics</td>
<td>“View and Align Multiple Sequences”</td>
<td></td>
</tr>
<tr>
<td><strong>Introduced in R2012b</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Phylogenetic Tree

Visualize, edit, and explore phylogenetic tree data

Description
The Phylogenetic Tree app lets you visualize and explore phylogenetic tree data.

You can:
• Prune, reorder, and rename branches.
• Collapse and expand branches.
• Inspect and measure the path length between leaf nodes.

Open the Phylogenetic Tree App
• MATLAB Toolstrip: On the Apps tab, under Computational Biology, click the app icon.
• MATLAB command prompt: Enter phytreeviewer.

Examples
• “Using the Phylogenetic Tree App”

Programmatic Use

phytreeviewer opens the Phylogenetic Tree app.

phytreeviewer(Tree) loads a phytree object into the app.

phytreeviewer(File) loads data from a Newick or ClustalW file into the app.

See Also

Apps
NGS Browser | Sequence Alignment | Sequence Viewer

Functions
phytreeviewer

Topics
“Using the Phylogenetic Tree App”

Introduced in R2012b
Sequence Viewer

Visualize and interactively explore biological sequences

Description
The Sequence Viewer app lets you visualize and explore amino acid or nucleotide sequences.

You can:
- Import sequences from the NCBI or EMBL databases directly.
- View and explore various sequence information (such as ORFs and CDSs) and basic statistics (such as percentages of base counts or amino acid counts).
- View the complement and reverse complement sequences, and other sequence features such as genes and exons.
- Extract protein coding sections of a nucleotide sequence and export them to the MATLAB workspace.
- Search for characteristic words or sequence patterns using regular expressions.

Open the Sequence Viewer App
- MATLAB Toolstrip: On the Apps tab, under Computational Biology, click the app icon.
- MATLAB command prompt: Enter seqviewer.

Examples

Open and View a Biological Sequence

Retrieve a sequence from the GenBank database.

S = getgenbank('M10051');

Load the sequence into the Sequence Viewer app.

seqviewer(S)

Alternatively, you can click Sequence Viewer on the Apps tab to open the app, and view the biological sequence S.
Close the app.

seqviewer('close')

- “Exploring a Nucleotide Sequence Using the Sequence Viewer App”

**Programmatic Use**

`seqviewer` opens the Sequence Viewer app.

`seqviewer(Seq)` loads a amino acid or nucleotide sequence `Seq` into the app. `Seq` can be one of the following:

- A string of single-letter codes
- A row vector of integers
A MATLAB structure containing a Sequence field that contains an amino acid or nucleotide sequence, such as the structure returned by `fastaread`, `fastqread`, `getgenpept`, `genpeptread`, `getpdb`, `pdbread`, `emblread`, `getembl`, `genbankread`, or `getgenbank`

A string specifying a file name with an extension of `.gbk`, `.gpt`, `.fasta`, `.fa`, or `.ebi.

See Also

Apps
NGS Browser | Sequence Alignment

Functions
`seqviewer`

Topics
“Exploring a Nucleotide Sequence Using the Sequence Viewer App”

Introduced before R2006a