Advanced Programming Techniques in MATLAB

Loren Shure

MathWorks, Inc.

Agenda

- MATLAB and memory
  - What you as a programmer should know
    - Passing arrays
    - How structures use memory

- Functions of all types
  - Introduction/Review of MATLAB function types
  - Applications of new nested functions
    - Solving optimization problems
    - Building a graphical user interface for volume visualization
    - Building 2-figure GUIs (Optional)

MATLAB and Memory

- Passing arrays to functions
  - When does MATLAB copy memory?

  ```
  function y = foo(x,a,b)
  a(1) = a(1) + 12;
  y = a*x+b;
  ```

- Calling `foo`
  ```
  y = foo(1:3,2,4)
  ```

  - `i.e.,` **x = 1:3, a = 2, b = 4**

```
>> edit foo.m
```

In-place Optimizations

- When does MATLAB do calculations "**in-place**"?

```
x = 2*x + 3;
y = 2*x + 3;
```

In-place Optimizations

When does MATLAB do calculations "**in-place**"?

```
function testInPlace
x = randn(n,1);
x = myfunc(x);          % vs. y = myfunc(x)
x = myfuncInPlace(x); % vs. y = myfuncInPlace(x)

function x = myfuncInPlace(x)
x = sin(2*x.^2+3*x+4);

function y = myfunc(x)
y = sin(2*x.^2+3*x+4);
```

```
>> edit myfuncInPlace myfunc testInPlace
%separate functions, separate files
```
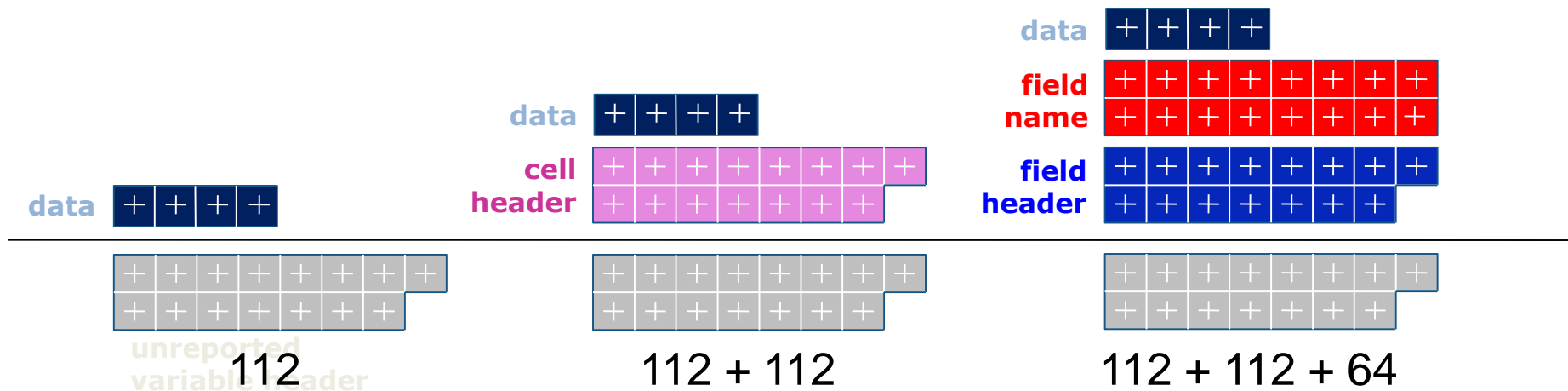
Memory Used for Different Array Types

```
d = [1 2]          % Double array
dcell  = {d}       % Cell array containing
dstruct.d = d

whos
```

```
>> edit overhead.m
```

Container Overhead

```
>> x   = [1 2]      % regular array, 16 bytes
>> y   = {[1 2]}    % cell array, 128 bytes
>> z.x = [1 2]      % structure, 192 bytes
>> whos
```


112

112 + 112

112 + 112 + 64

MATLAB and Memory

How does MATLAB store structures?

```
s.A = rand(3000,3000);
s.B = rand(3000,3000);


sNew = s;


s.A(1,1) = 17;


>> edit structmem1.m
```

MATLAB and Memory

How does MATLAB store structures?

```
im1.red = redPlane;    % each plane is m x n
im1.green = greenPlane;
im1.blue = bluePlane;
```
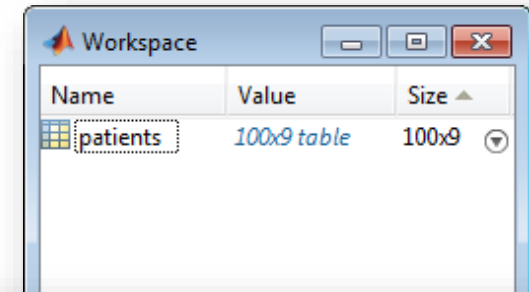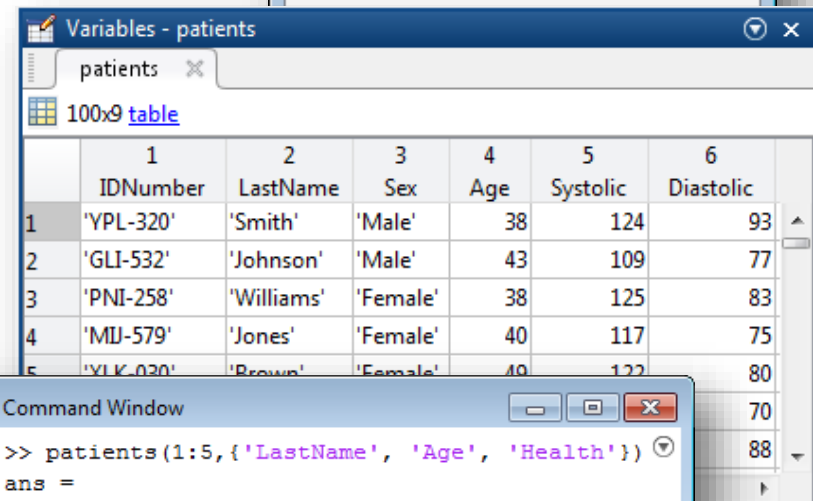
versus

```
% each 1x3
im2(1,1).pixel = [red(1) green(1) blue(1)];
im2(2,1).pixel = [red(2) green(2) blue(2)];
...
im2(m,n).pixel = [red(m*n) green(m*n) …
                  blue(m*n)];
```
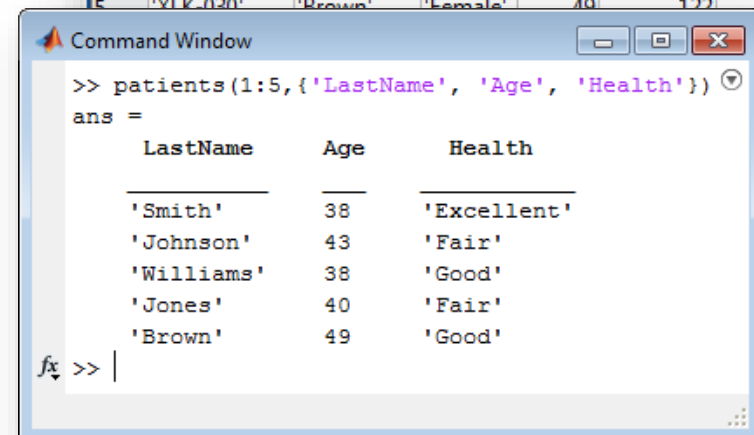
```
>> edit structmem2.m
```

Tables

- New fundamental data type

- For mixed-type tabular data
  - Holds both data and metadata

- Supports flexible indexing

- Built-in functionality (merge, sort, etc.)
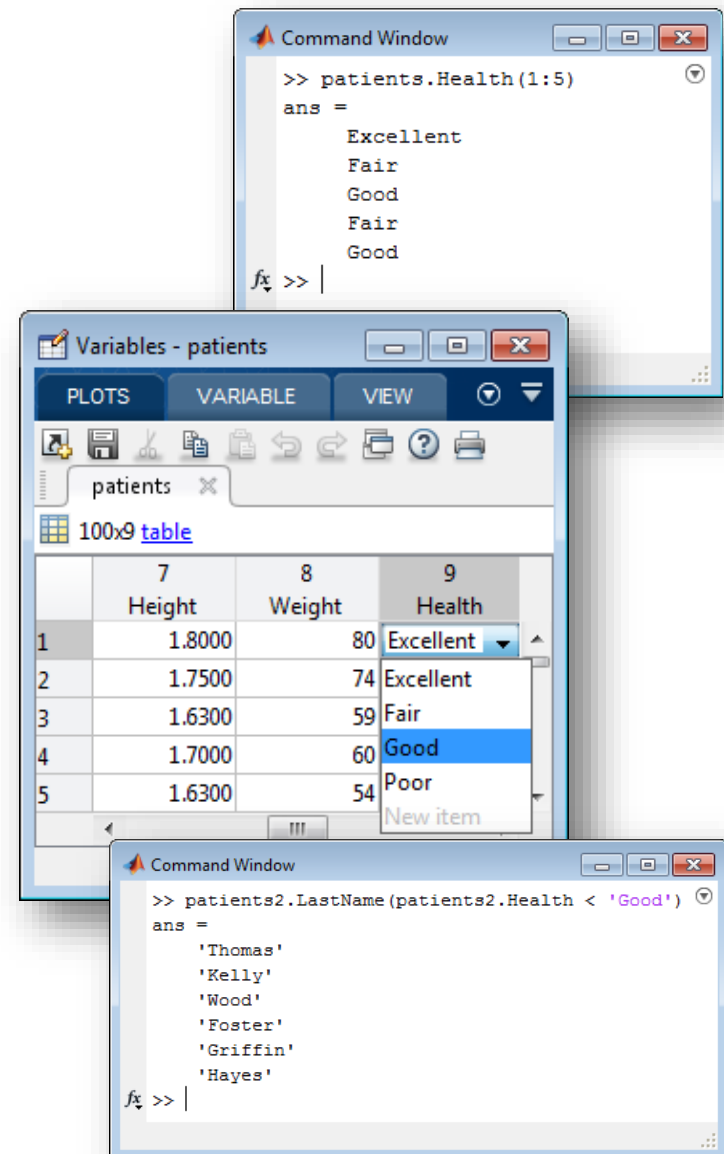
Categorical Arrays

- **New fundamental data type**

- **For discrete non-numeric data**
  - Values drawn from a finite set of possible values ("categories")

- **More memory efficient than a cell array of strings**

- **Can be compared using logical operators**
  - Similar to numeric arrays

```
>> edit showCategorical.m
```

Summary of MATLAB and Memory

- How MATLAB passes arrays to functions
  - By value, with "lazy" copy or copy on write
  - In-place optimization code pattern

- Memory use in array storage
  - Atomic types vs. cell arrays and structures
  - Array of structs vs. struct arrays
    - i.e., s(300,300).red vs. s.red(300,300)
  - Categorical arrays and tables

MATLAB and Memory: Additional Resources

- Recorded Webinar:
  - Handling Large Data Sets Efficiently in MATLAB
  - mathworks.com -> Recorded Webinars

- Memory Management Links from doc page for `memory`
    - Memory Allocation
    - Strategies for Efficient Use of Memory
    - Resolving "Out of Memory" Errors

Functions

Everything you wanted to know, but were afraid to ask

A comp.soft-sys.matlab Post

Question:

With the new function types introduced in R14, I am curious as to which methods would be preferable from a speed standpoint. We have:

1. Functions defined in separate files

2. Subfunctions

3. Inline functions

4. Anonymous functions

5. Nested functions

Other than scope, persistence, or program structure, is there some difference that

makes MATLAB work better with the type?

comp.soft-sys.matlab Post Answered

Cleve's Answer

Good question, and a hard one to answer. As MATLAB evolves, the answer will change. The function call mechanism in the current version of MATLAB is pretty expensive, for any kind of function. One of the most important tasks facing our Accelerator/JIT team today is to make function calls faster. This is why we have introduced the @ notation for both function handles and anonymous functions. Future versions of MATLAB should have improved speed for anything involving the @ sign.

Inline functions were an experiment in the use of the overloading mechanism that turned out to be useful. But we've always regarded them as a bit of a hack. So our advice now is to phase out their use over time. Other than that, your choice today among the other kinds of functions should be based on style and convenience.

Function Quiz

- Let's see how much *you* know about functions in MATLAB

- This quiz covers
  - Anonymous functions
  - Nested functions
  - Function handles
  - Regular ("simple") functions
  - Subfunctions

- This quiz does not cover
  - MEX functions, private functions, …

Function Quiz Review

- Regular ("simple") functions

- Function handles

- Anonymous functions

- Subfunctions

- Nested functions

- Q: Which of these function types can be combined in a single file?

Advice

- You can use the function `functions` to explore details of the function referred to by a particular function handle.

- NOTE: Changing the values of the struct output from functions does NOT alter the function handle details!

Nested Function Applications

- Solving optimization problems

- Building a graphical user interface for volume visualization

- Building 2-figure GUIs (Optional)

Application 1:
Solving Optimization Problems

- We get many posts on comp.soft-sys.matlab about optimization. The problems fall into several catgories. The one we will address to today is:

    – How to include extra parameters to define the objective function

Optimization Example (unconstrained)

Objective function:

`a x1^2 + b x1 x2 + c x2^2 + d abs(x2 – x1)+ e*randn`

`a,b,c` – "Regular" parameters

`d,e` – Additional parameters that might alter the type of problem by making the objective function either

- non-smooth (`d`) or
- stochastic (`e`)

```
>> edit optimS.m optimAnon.m
```

Application 2:
Building a Graphical User Interface for Volume Visualization

- Application: Building a custom tool for volume visualization

- This example illustrates:
  - Using function handles to export nested functions
  - Using nested functions for object callbacks
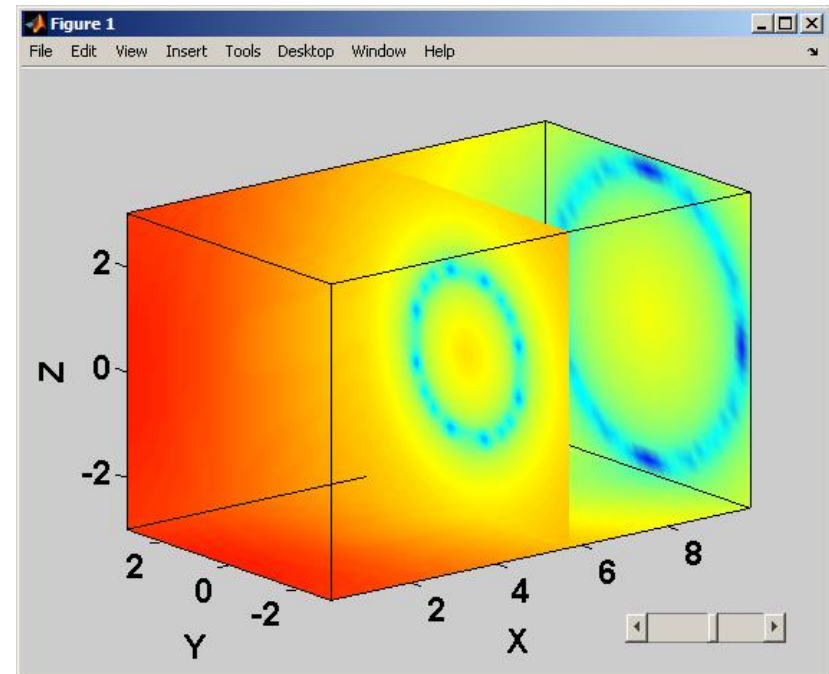
Application 3 (Optional): Creating Two-Figure GUIs

- This is a frequent comp.soft-sys.matlab request (several times each month).

- Challenges:
  - Managing handles
  - Sharing information between figures

- Nested functions can simplify and improve performance.

```
>> edit redBlueN.m
```

Red-Blue Exercise

- Start with code in `redBlueN2.m`

- Change so there are 2 figures,
  - "Blue" button in one figure
  - "Red" in the other

- Have each button change the color of BOTH figures

Why Use Nested and Anonymous Functions?

Benefits of nested and anonymous functions

- More robust

- Changes in path cause fewer problems with function handles

- Data sharing and number of copies of data

  – Memory savings, especially for large shared data

- Program structure

- Over time, higher performance

Additional benefits of nested (and sub) functions

- Scope and persistence

- Reduce variable and function namespace clutter

Function Usage Advice

- You can now (as of Release 14) call functions without inputs using this notation:

  ```
  y = foo();
  ```

- Put all nested functions at the bottom of the enclosing function. You may prefer to have them defined right where you will use them, but this can become unwieldy.

- Place a comment next to a call to a nested function stating what workspace variables are altered by the call (since the nested function can see the enclosing workspace).

More Function Usage Advice

- Nested functions are good when there is significant shared information (many pieces or large arrays).

- Nested functions are very good when there is much large data to share for reading and writing.

- Subfunctions (or private functions) are useful when users don't need these directly. Those who need them can see.

- Data sharing inside subfunctions is only reading large arrays, not making copies and changing them.

MATLAB Function Types: Additional Resources

- **MATLAB Digest - September 2005**
  - Dynamic Function Creation with Anonymous and Nested Functions
  - http://www.mathworks.com/company/newsletters/digest/2005/sept/dynfunctions.html

- **Examples**
  - Anonymous functions
  - http://www.mathworks.com/products/matlab/demos.html?file=/products/demos/shipping/matlab/anondemo.html
  - Nested Functions
  - http://www.mathworks.com/products/matlab/demos.html?file=/products/demos/shipping/matlab/nesteddemo.html

- **The Art of MATLAB Blog**
  - Look at Category: Function Handles
  - http://blogs.mathworks.com/loren/

Summary

- MATLAB memory usage
  - When is data copied

- Nested and anonymous functions
  - Very good at data encapsulation
  - Efficient for sharing memory between functions
  - Another choice for name / variable scope
  - Less clutter on MATLAB path