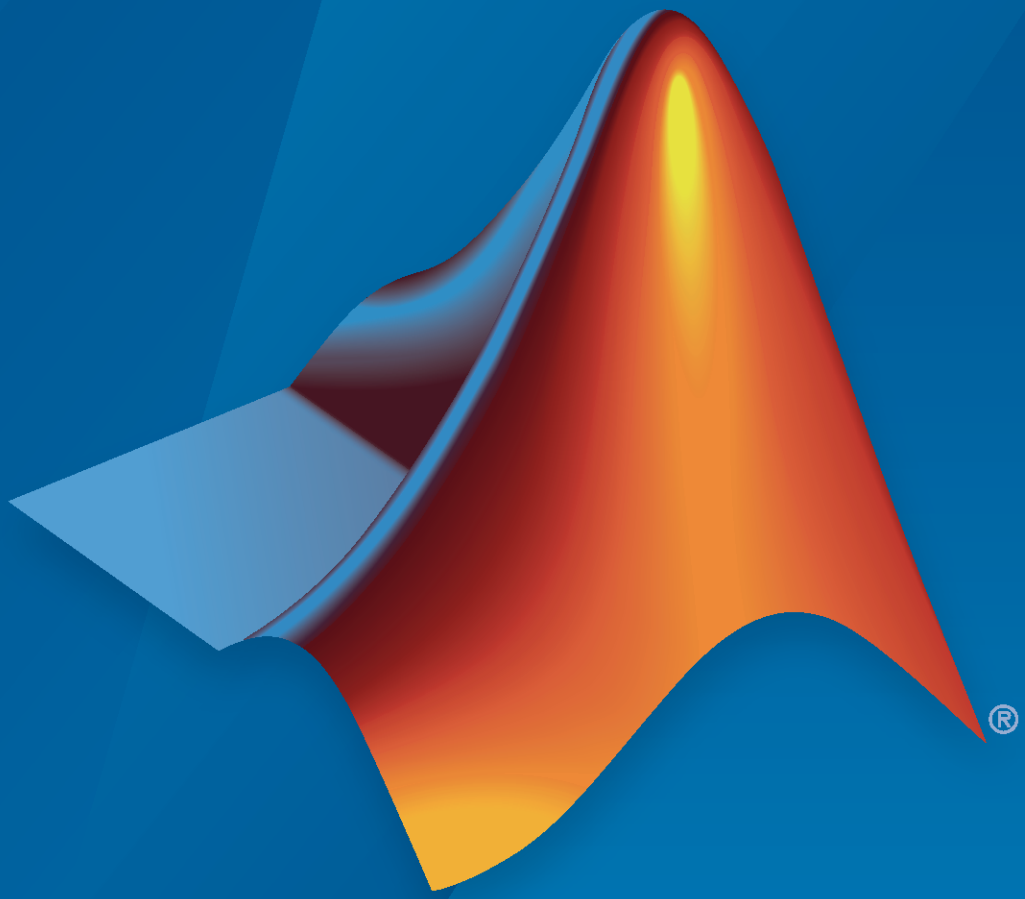
Signal Processing Toolbox™ Release Notes



MATLAB®

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Signal Processing Toolbox™ Release Notes

Contents

R2026a

Filter Designer App: New and improved algorithm- and specification-based filter design **1-2**

digitalFilter Object: Use digital filter cascades and CTF format **1-3**

filt2block Function: Create Simulink blocks from CTF coefficients and digital filter cascades **1-3**

Filter Design: Design complex FIR filters and use custom window functions **1-3**

Filter Analyzer App: Add data tips and copy displays **1-3**

Spectral and Time-Frequency Analysis: Plot in axes, panels, and apps . . **1-3**

Feature Extraction: Extract time-frequency features from specified signal representation **1-4**

Signal Feature Extractor App: Import audio signals and extract audio features **1-4**

Signal Labeling: Use time-frequency masks and get STFT map **1-4**

Spectral Measurements: Estimate band power from multiple bands **1-4**

signalDatastore Object: Read file data in parallel **1-4**

filtfilt Function: New options to perform zero-phase digital filtering **1-5**

Enhanced accuracy for digital filter design, spectral windows, and waveform generation **1-5**

EDF File Analyzer App: New character encoding formats to read annotations **1-5**

AI Application Examples: Coexecute PyTorch models in MATLAB **1-5**

Code Generation Example: Generate deployable code that identifies faults in gearbox **1-5**

Deep Learning: Code generation support for inverse short-time Fourier transform **1-6**

GPU support to compute average order-magnitude spectrum for vibration analysis **1-6**

Single-precision support for spectral windows and digital filter analysis **1-6**

Functionality being removed or changed **1-6**
Filter Designer has changed **1-6**
signalTimeFrequencyFeatureExtractor has changed **1-7**
filterBuilder will be removed **1-7**
dspfwiz will be removed **1-7**
FVTool will be removed **1-7**

R2025b

Quality and stability improvements **2-2**

R2025a

Signal Feature Extractor App: Extract signal features in time and frequency domain **3-2**

Filter Analyzer App: Add legend strings, set sample rates, and visualize filter stages **3-2**

Signal Labeler App: Label signals in time-frequency domain **3-3**

Signal Labeling: Label signals in time-frequency domain **3-3**

spectrogram function: Plot spectrograms in axes and apps **3-4**

Create Plot Live Editor Task: Visualize function outputs **3-4**

spectralEntropy function: Specify time information, frequency range, time limits, and obtain time vector associated with spectral entropy output **3-4**

spectralKurtosis function: Specify time information and obtain frequency vector associated with spectral kurtosis output **3-5**

stftLayer and istftLayer objects: Initialize layer weights through dlnetwork initialization **3-5**

Experiment Manager Templates: Enhancements **3-5**

AI Application Examples: Analyze, classify, and estimate radar and wireless data for spectrum sensing applications **3-6**

Deep Learning: Code generation support for short-time Fourier transform **3-6**

GPU support for digital filter design, signal transforms, and vibration analysis **3-6**

Functionality being removed or changed **3-6**
Signal Labeler has changed **3-6**
New default value for stftLayer and istftLayer learnable parameters **3-7**

R2024b

Digital Filter Design: Design classic IIR filters using cascaded transfer functions **4-2**

Digital Filter Analysis: Visualize and analyze filters using cascaded transfer functions **4-2**

Digital Filtering: Filter signals using cascaded transfer functions **4-3**

Peak Finding: Refine peak value and location estimates **4-4**

Feature Extraction: Extract scalar features and integrate with datastore objects **4-4**

spectralEntropy and spectralKurtosis functions: Customize feature extraction and input timetables **4-5**

signalDatastore object: Cast data in single precision and output GPU arrays **4-5**

Filter Analyzer App: Drag filters from Filters table to displays **4-5**

Signal Labeler App: Export labeled signal sets to Diagnostic Feature Designer **4-5**

Experiment Manager Templates: Quickly set up transfer learning signal classification experiments **4-6**

AI Application Examples: Detect anomalies, denoise signals, extract features, and label signals **4-6**

Feature Extraction Examples: Accelerate feature extraction using datastores, GPUs, and parallel computing **4-6**

Application Example: Denoise signals using Savitzky-Golay filters with weighting vectors **4-6**

C/C++ code generation support for descriptive statistics, signal modeling, vibration analysis, and waveform generation **4-6**

GPU support for time-frequency analysis and feature extraction **4-7**

GPU code generation support for feature extraction and spectral analysis functions **4-7**

Single-precision support for spectral windows and waveform generation **4-7**

Functionality being removed or changed **4-7**
FVTool will be removed **4-7**
pentropy will be removed **4-9**
pkurtosis will be removed **4-9**
digitalFilter has changed **4-11**

R2024a

Filter Analyzer App: View, analyze, and compare filters **5-2**

Filter Design: Use cascaded transfer functions **5-3**

Feature Extraction: Extract time-frequency features of signals **5-3**

deepSignalAnomalyDetector Object: Perform streaming anomaly detection and save models **5-3**

Partition signals and label sequences into frames **5-3**

Signal Analyzer App: Listen to audio signals **5-4**

Signal Analyzer App: Fill missing data **5-4**

Signal Labeler App: Compute and compare more types of spectra and spectrograms **5-4**

Signal Labeler App: Label more easily **5-4**

Deep Learning: Inverse short-time Fourier transform layer **5-4**

dlistft Function: Compute deep learning inverse short-time Fourier transforms **5-4**

Experiment Manager Templates: Quickly set up deep learning experiments with signals **5-5**

AI Application Examples: Detect anomalies, denoise signals, extract features, and label signals **5-5**

Application Example: Perform graph signal processing **5-5**

Generate C/C++ code for signal generation and spectral analysis **5-5**

GPU code generation support for preprocessing and spectral analysis functions **5-6**

Functionality being removed or changed **5-6**
Signal Labeler has changed **5-6**

R2023b

Signal Labeler App: Automatically label bounded signal regions **6-2**

Signal Labeler App: Automatically classify sounds in audio signals and label them **6-2**

Signal Analyzer App: Compute and compare more types of spectra **6-2**

stftmag2sig Function: Recover phase information from spectrogram magnitudes using gradient descent **6-2**

resize, paddata, and trimdata Functions: Change the size of data by adding or removing elements **6-2**

Application Example: Compute Shock Response Spectra **6-3**

Single-precision support for spectral analysis and multirate signal processing **6-3**

C/C++ Code Generation Support: Code generation for spectral analysis and signal modeling **6-3**

GPU support for spectral analysis and order analysis **6-3**

GPU code generation support for time-frequency analysis functions **6-4**

Functionality being removed or changed **6-4**
dlstft combines real and imaginary parts of transform into one output argument **6-4**
stftLayer weights initialized to analysis window **6-4**

R2023a

Signal Anomaly Detection: Detect signal anomalies using deep learning autoencoders **7-2**

Signal Labeling: Label signal points within a range **7-2**

Signal Analyzer App: Find and annotate signal peaks **7-2**

Signal Analyzer App: Rename and delete signals in preprocessing mode **7-2**

Create Plot Live Editor Task: Visualize function outputs **7-2**

Application Examples: Detect anomalies in signals and perform fatigue analysis **7-2**

Single-precision support for digital filtering and multirate signal processing **7-3**

C/C++ Code Generation Support: Code generation for digital filter design, multirate signal processing, and waveform generation **7-3**

GPU support for feature extraction, statistics, and spectral measurements **7-3**

R2022b

Labeling Convenience Function: Generate labels from file names **8-2**

alignsignals Function: Align signals based on rising edge or peak locations **8-2**

Signal Analyzer App: Interactively preprocess signals with various functions and actions **8-2**

Signal Analyzer and Signal Labeler Apps: View frequency axes in log scale and toggle decibel display **8-2**

Spectrogram Computation: Compare available functions **8-2**

AI Workflows: Discover datastores, functions, and other resources for AI tasks **8-2**

Deep Learning Examples: Recover signals, monitor human health, and perform signal source separation **8-3**

C/C++ Code Generation Support: Code generation for signal generation and preprocessing, time-frequency analysis, and vibration analysis . . **8-3**

GPU support for spectral and time-frequency analysis **8-3**

Functionality being removed or changed **8-4**
alignsignals syntax changes **8-4**
stftLayer: OutputMode property will be removed in a future release **8-4**

R2022a

Signal Analyzer App: Calculate signal statistics **9-2**

Signal Analyzer App: Interactively edit signals **9-2**

Signal Analyzer App: Analyze signals with nonfinite data **9-2**

Signal Labeler App: Automatically detect speech regions in audio signals and label spoken words **9-2**

Signal Labeler App: Extract features from signals **9-2**

Signal Label Definitions: Define label definitions for generated features **9-2**

Labeled Signal Sets: Generate feature data and get label indices from the command line **9-3**

Labeled Signal Sets: Access source datastore on different machines **9-3**

Deep Learning Examples: Use adversarial learning denoiser model and perform sequence-to-sequence classification **9-3**

C/C++ Code Generation Support: Code generation for filtering, feature extraction, preprocessing, and signal measurements **9-3**

GPU support for feature extraction, spectral analysis, spectral measurements, and transforms **9-3**

Functionality being removed or changed **9-4**
Filter Visualization Tool (fvtool) has changed **9-4**
sptool has been removed **9-4**

R2021b

Design Filter Live Editor Task: Design digital filter interactively **10-2**

Signal Labeler App: Inspect distribution of label counts on heatmap . . **10-2**

Signal Labeler App: Show outliers in Dashboard **10-2**

Signal Labeler App: Listen to audio signals while annotating them interactively **10-2**

Signal Analyzer App: Denoise signals interactively using wavelet methods **10-2**

Feature Extraction: Extract time-domain and frequency-domain features of signals **10-2**

Feature Extraction: Compute zero-crossing rates of signals **10-3**

Deep Learning: Short-time Fourier transform layer **10-3**

Deep Learning Examples: Use short-time Fourier transform layer and perform deep learning regression **10-3**

Signal Datastores: Specify FileSet objects as data locations **10-3**

poctave Function: Improved octave smoothing and filter design **10-3**

C/C++ Code Generation Support: Code generation for filtering, spectral analysis, and vibration analysis **10-3**

GPU support for digital filtering, feature extraction, signal processing, transforms, and waveform generation **10-4**

Run functions in a thread-based environment **10-4**

MATLAB Online support for Signal Analyzer and Signal Labeler **10-4**

Functionality being removed or changed **10-4**

designfilt no longer assists in correcting incomplete or erroneous function calls **10-4**

sptool will be removed **10-5**

R2021a

Signal Labeler App: Analyze labeling progress and distribution of labels in your labeled signal set **11-2**

Signal Labeler App: Label complex-valued signals **11-2**

Signal Labeler App: View spectra and spectrograms in Fast Navigation mode **11-2**

Signal Labeler App: Speed up labeling and inspection by panning and zooming through signals and labels **11-2**

Signal Labeler App: Import and label audio files **11-2**

EDF File Analyzer App: View EDF or EDF+ files **11-2**

European Data Format Files: Create and modify EDF or EDF+ files **11-2**

Labeling Convenience Functions: Split data sets by label value and assign attributes based on file location **11-2**

Signal Labeling: Count label values and create datastores from labeledSignalSet objects **11-3**

dlstft Function: Compute short-time Fourier transforms of deep learning arrays **11-3**

Signal Datastores: Write data from datastore to files using writeall . . . **11-3**

Time-Frequency Analysis: Compute instantaneous signal bandwidth . . **11-3**

poctave Function: Compute and visualize octave spectrograms **11-3**

Signal Resampling: Change sample rates of MATLAB timetables or resample them to uniform grids **11-3**

Deep Learning Examples: Classify signals using neural networks and a custom log spectrogram layer **11-4**

Signal Processing Onramp: Interactive introduction to practical signal processing methods **11-4**

C/C++ Code Generation Support: Code generation for filtering, signal modeling, spectral analysis, and statistics **11-4**

GPU support for signal labeling, time-frequency analysis, transforms, digital filtering, and waveform generation **11-4**

R2026a

Version: 26.1

New Features

Bug Fixes

Compatibility Considerations

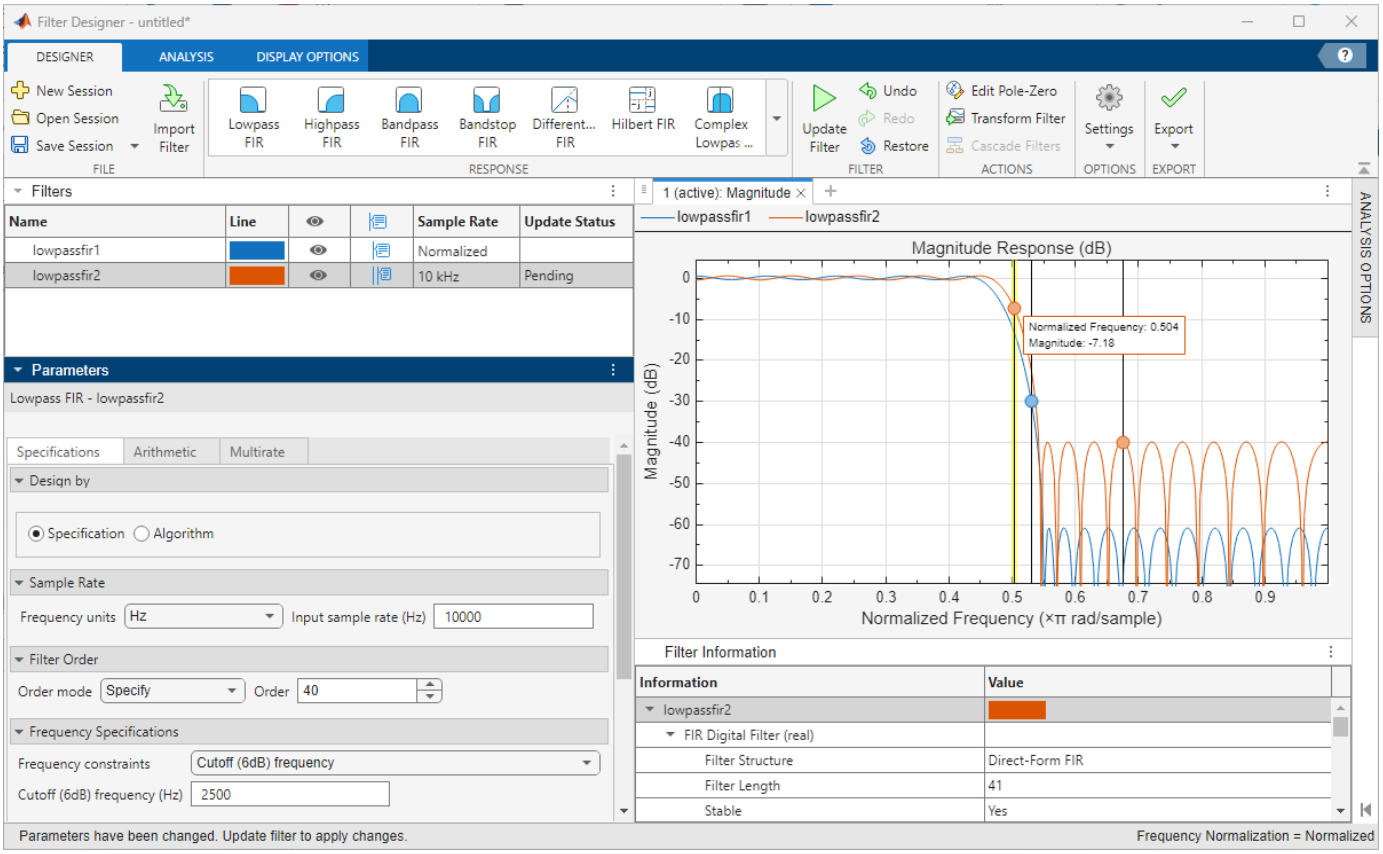
Filter Designer App: New and improved algorithm- and specification-based filter design

This release introduces a new version of the Filter Designer app. The app enables you to design IIR, FIR, and multirate filters using any one of these options:

- Start with a list of specifications and then select an algorithm that best satisfies those parameters.
- Start with an algorithm and then adjust the algorithm parameters.
- Add, remove, or modify transfer-function poles and zeros.
- Import and modify a `digitalFilter` object, a filter System object™, or a design made with a previous version of the app.

You can manage, analyze, and compare your filters in the same display or in side-by-side displays using several analysis plots. After you finish designing a filter, you can:

- Export your design to the MATLAB® workspace or to Simulink®.
- Generate a C header or COE file using coefficients.
- Generate MATLAB code to recreate your design or filter a signal.
- Export filters to DSP HDL IP Designer, if you have a DSP HDL Toolbox™ license.



Compatibility Considerations

For more information, see “Filter Designer has changed” on page 1-6.

digitalFilter Object: Use digital filter cascades and CTF format

Starting this release, `digitalFilter` objects can also represent “Digital Filter Cascades”. A digital filter cascade is a representation of several digital filters connected in series.

These new object functions enable you to create, analyze and modify a digital filter cascade:

- `cascade` — Cascade two or more digital filter objects.
- `ctf` — Return filter coefficients in cascaded-transfer-function (CTF) format. If the filter is a cascade of digital filters, it returns the coefficients of all the stages in CTF format.
- `getNumStages` — Get the number of stages of a digital filter or digital filter cascade.
- `setSampleRate` — Set the sample rate of a digital filter or digital filter cascade.

You can now create a `digitalFilter` object by specifying a set of filter coefficients in CTF format.

filt2block Function: Create Simulink blocks from CTF coefficients and digital filter cascades

The `filt2block` function now supports creating Simulink blocks from CTF coefficients, from `digitalFilter` objects that contain CTF coefficients, or from `digitalFilter` objects that represent digital filter cascades.

Filter Design: Design complex FIR filters and use custom window functions

Starting this release, the `designfilt` function supports these additional equiripple-based filter design responses.

- `'complexlowpassfir'` — Complex lowpass FIR filter
- `'complexhighpassfir'` — Complex highpass FIR filter

In the Design Filter Live Editor task, you can now specify a custom window function for window-based filter designs.

Filter Analyzer App: Add data tips and copy displays

This release enables you to add data tips on filter analysis plots in the Filter Analyzer app and copy displays to the clipboard.

Spectral and Time-Frequency Analysis: Plot in axes, panels, and apps

Signal Processing Toolbox™ introduces a `Parent` name-value argument that enables you to specify a target container in which to plot output for these functions:

- Spectral Analysis — `cpsd`, `mscohere`, `periodogram`, `poctave`, `pwelch`, and `tfestimate`

- Time-Frequency Analysis — `pspectrum`, `stft`, and `xspectrogram`

Specify the target container as one of these objects: `Axes`, `UIAxes`, or `Panel`.

The “Plot Spectral Representations of Signal in App Designer” example shows how to create and run an app that plots Welch's PSD estimate, the octave spectrum, and the scalogram of a signal.

⚠Feature Extraction: Extract time-frequency features from specified signal representation

The `signalTimeFrequencyFeatureExtractor` object now supports feature extraction from a signal representation that you specify in the `Transform` property.

- This release introduces the `timeFrequencyFeatureTransformOptions` object, which lets you set the signal representation associated with each time-frequency feature that you aim to extract.
- The `Transform` property now supports a `struct` array or a `timeFrequencyFeatureTransformOptions` object.

⚠Compatibility Considerations

For more information, see “signalTimeFrequencyFeatureExtractor has changed” on page 1-7.

Signal Feature Extractor App: Import audio signals and extract audio features

Starting this release, you can import audio signals and extract audio features in the Signal Feature Extractor app. You must have a license for Audio Toolbox™ to use this functionality.

★Signal Labeling: Use time-frequency masks and get STFT map

This release introduces the `timeFrequencyMask` object, which enables you to plot time-frequency regions of interest and convert time-frequency masks for machine learning workflows

This release also introduces two functions for the `labelSpectrogramOptions` object.

- The `getTFMap` function enables you to obtain the short-time Fourier transform (STFT) map of a signal based on the spectrogram options set on the `labelSpectrogramOptions` object.
- The `getTFMapSize` function enables you to determine the size of the STFT map that would result if computed using the spectrogram options set on the `labelSpectrogramOptions` object.

Spectral Measurements: Estimate band power from multiple bands

The `bandpower` function now returns the average power at multiple frequency ranges.

signalDatastore Object: Read file data in parallel

The new `UseParallel` name-value argument in the `readall` function enables you to read file data in parallel from a `signalDatastore` object. You must have a Parallel Computing Toolbox™ license to use this functionality.

★filtfilt Function: New options to perform zero-phase digital filtering

Starting this release, the `filtfilt` function supports new name-value arguments that enable you to zero-phase filter signals with additional options.

- `InitialStatesMethod` — Choose an algorithm to estimate the initial conditions of the filter states.
- `TransientLength` — Specify a number of samples or choose an estimation method to suppress the transient response from the filtered signal.
- `PaddingPattern` — Choose a pattern to pad the input signal on both ends before filtering.

Enhanced accuracy for digital filter design, spectral windows, and waveform generation

This release updates the method to compute the sine and cosine terms used in several Signal Processing Toolbox functions. The `sinpi` and `cospi` functions provide more accurate results than `sin` and `cos` for inputs that represent multiple-of-π angles.

These functions now yield enhanced accuracy:

- Digital Filter Design — `fir1`, `fircls`, `fircls1`, `firls`, and `rcosdesign`
- Spectral Windows — `barthannwin`, `blackman`, `bohmanwin`, `flattopwin`, `hamming`, `hann`, `taylorwin`, and `tukeywin`
- Waveform Generation — `sinc`

EDF File Analyzer App: New character encoding formats to read annotations

The EDF File Analyzer app, the `edfread` function, and the `edfinfo` object now support reading annotations in any of these character encoding formats: UTF-8, ASCII, and Latin-1.

AI Application Examples: Coexecute PyTorch models in MATLAB

This release introduces examples that apply signal processing techniques and deep learning models using Python® coexecution to the analysis of signals.

- “Use Signal Feature Extraction to Train PyTorch Fault Detection Model” demonstrates how to coexecute a PyTorch® long short-term memory neural network in MATLAB to identify faulty bearing signals.
- “Signal Imputation in Signal Analyzer Using Time-Series Foundation Model” demonstrates how to coexecute a PyTorch time-series foundation model in Signal Analyzer to restore missing samples in signals.
- “Anomaly Detection in Signal Labeler Using Time-Series Foundation Model” demonstrates how to coexecute a PyTorch time-series foundation model in Signal Labeler to detect anomalies in signals.

Code Generation Example: Generate deployable code that identifies faults in gearbox

This release introduces an example that demonstrates how to generate deployable code for vibration analysis. “Generate C++ Code That Analyzes Vibration Signals from Rotating Machinery” shows how

to generate C++ code that analyzes the vibration signal from an accelerometer attached to a gearbox and identifies the local faults on the gear or pinion teeth.

Deep Learning: Code generation support for inverse short-time Fourier transform

The `dlistft` function now supports C/C++ and GPU code generation.

You must have MATLAB Coder™ to generate standalone C and C++ code. You must have a MATLAB Coder and GPU Coder™ license to generate CUDA® code for graphical processor units (GPU).

GPU support to compute average order-magnitude spectrum for vibration analysis

The `orderspectrum` function now supports `gpuArray` objects.

You must have a Parallel Computing Toolbox license to use `gpuArray` objects with the supported functions. For more details, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox). To find which GPUs are supported, see “GPU Computing Requirements” (Parallel Computing Toolbox).

Single-precision support for spectral windows and digital filter analysis

These Signal Processing Toolbox functions now support single precision:

- Spectral Windows — `gausswin`, `hamming`, `kaiser`, `nuttallwin`, `rectwin`, `taylorwin`, and `tukeywin`
- Digital Filter Analysis — `filternorm`

⚠Functionality being removed or changed

Filter Designer has changed

Behavior change

The new version of the Filter Designer app contains most of the functionality found in previous versions plus new and updated features. On the other hand, there are some differences to keep in mind:

- In Filter Designer, you can open any design sessions that you saved as `.fda` files using previous versions of the app. This feature has some limitations:

  1. In some cases, the new Filter Designer cannot infer the metadata that stores the specifications used to design a given filter. In those cases, the app imports only the filter coefficients. You cannot modify designs specified as coefficients, but you can use them as reference to compare to new designs. You can also export these designs back to the workspace or use them to create Simulink models.
  2. There are some design options that the new Filter Designer does not support. If you used these unsupported options when designing your filter and set them to nondefault values, the filter specification metadata inferred by the new Filter Designer may not be a perfect match to the filter specification metadata of the original design.

3. The algorithm used by the new Filter Designer to determine the order of a minimum-order design is an enhanced version of the one used by previous versions. For that reason, the order of your filter might change when you redesign the filter with the same specification parameters.

   For more information, see “Version History”.

- Filter Designer no longer imports or exports filters as `dfilt` objects. Instead, the app supports filters as coefficients in CTF format, `digitalFilter` objects, or filter System objects.
- You can still specify the coefficient quantization in the new Filter Designer. To set other internal fixed-point properties, export the filter and adjust the settings in the System object, Simulink block, or DSP HDL IP Designer session.

signalTimeFrequencyFeatureExtractor has changed
Behavior change

The `Transform` property of `signalTimeFrequencyFeatureExtractor` objects will no longer support string scalars or character vectors in a future release. Instead, specify `Transform` as a `struct` array or a `timeFrequencyFeatureTransformOptions` object.

If you create or edit a `signalTimeFrequencyFeatureExtractor` object whose `Transform` property is specified as a string scalar or character vector, `signalTimeFrequencyFeatureExtractor` automatically sets `Transform` to a `timeFrequencyFeatureTransformOptions` object.

filterBuilder will be removed
Still runs

`filterBuilder` will be removed in a future release. Use the Filter Designer app or the `designfilt` function with no input arguments instead.

dspfwiz will be removed
Still runs

`dspfwiz` will be removed in a future release. To create Simulink blocks from your filter designs, use Filter Designer.

FVTool will be removed
Warns

FVTool will be removed in a future release. Use Filter Analyzer instead. There are differences that require updates to your code.

R2025b

Version: 25.2

Bug Fixes

Quality and stability improvements

R2025b delivers quality and stability improvements, building on the new features introduced in R2025a.

R2025a

Version: 25.1

New Features

Bug Fixes

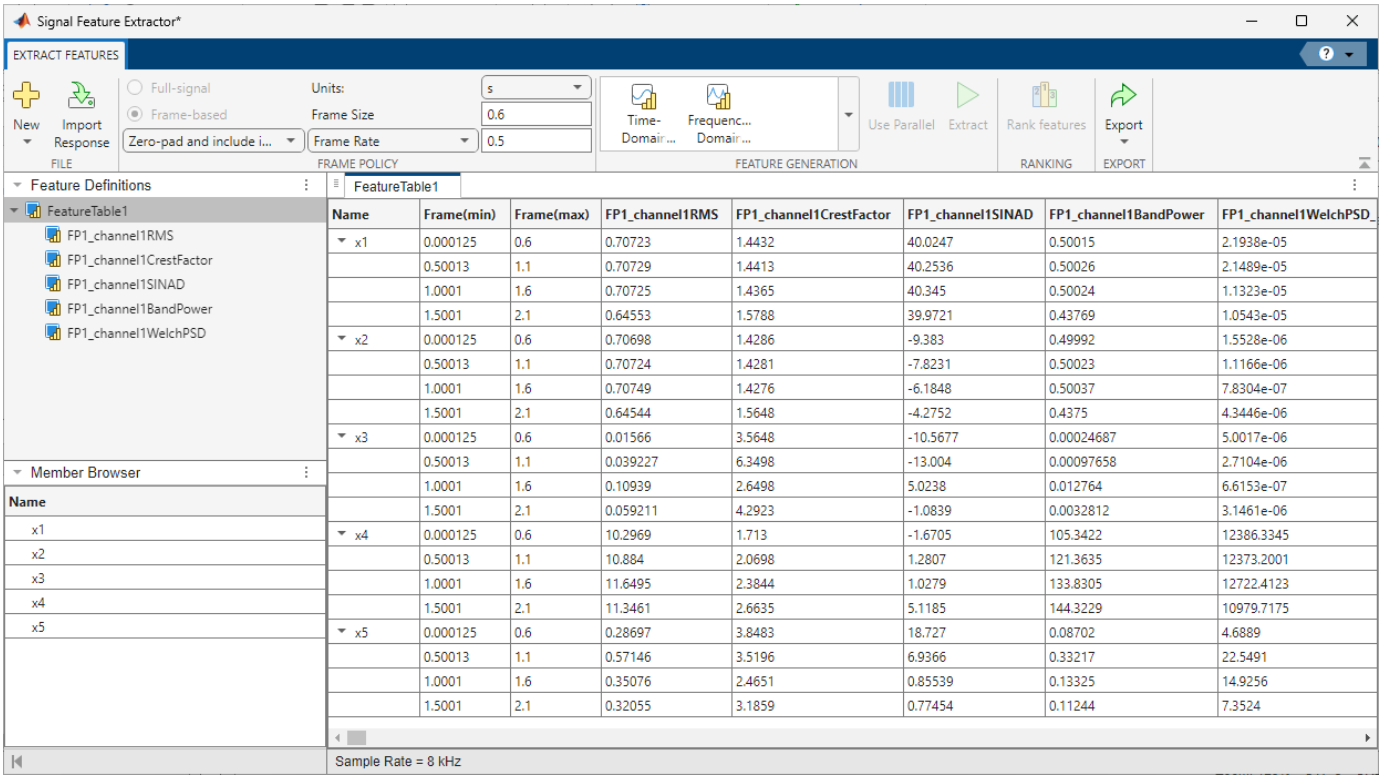
Compatibility Considerations

★Signal Feature Extractor App: Extract signal features in time and frequency domain

This release introduces the Signal Feature Extractor app. Signal Feature Extractor enables you to extract and process signal features in the time and frequency domain.

- Import signals from MATLAB workspace or local file and categorical response labels from MATLAB workspace.
- Extract time-domain and frequency-domain features from entire signals or per frame.
- Use parallel computation to accelerate feature extraction.
- Rank signal features with feature-ranking algorithms for classification workflows, including MRMR, ANOVA, and Kruskal Wallis.
- Export labeled signal sets to the MATLAB workspace or a local file.
- Export ranked features or all the extracted features to the MATLAB workspace or to the Classification Learner (Statistics and Machine Learning Toolbox) app.

You must have a Parallel Computing Toolbox license to use parallel computation. You must have a Statistics and Machine Learning Toolbox™ license to rank signal features.



Filter Analyzer App: Add legend strings, set sample rates, and visualize filter stages

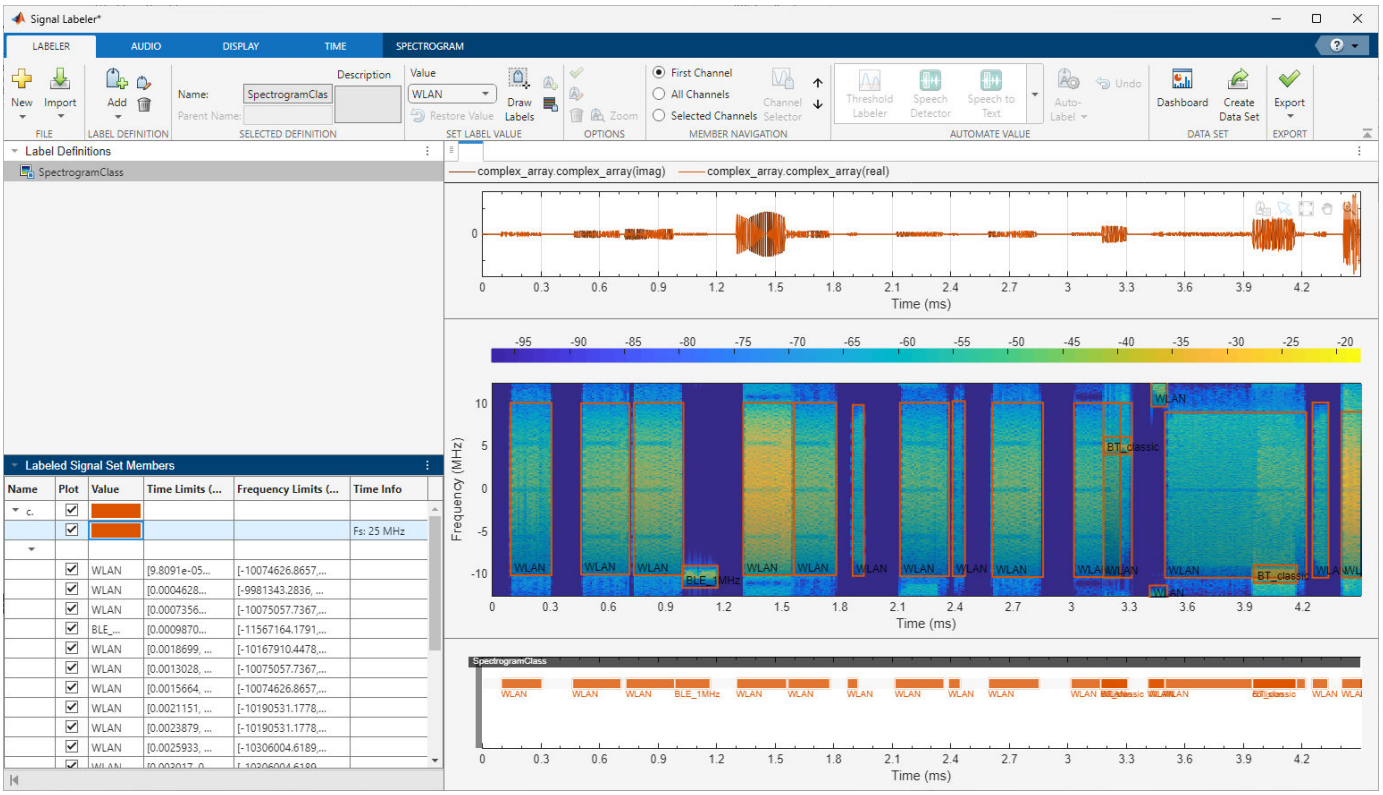
This release introduces enhancements for filter visualization and measurement using the Filter Analyzer app.

- You can now visualize the filter stages in a tree hierarchy and analyze the response of each stage as well as the overall filter response.
- The updated `filterAnalyzer`, `addFilters`, and `replaceFilters` functions include the new `LegendStrings` name-value argument, which enables you to append strings to the default legend strings for all the filters added to Filter Analyzer.

★Signal Labeler App: Label signals in time-frequency domain

Starting this release, you can label signals in the time-frequency domain using the Signal Labeler app.

- Create time-frequency region-of-interest label definitions.
- Specify spectrogram options to calculate the time-frequency signal representation to be labeled.
- Label time-frequency regions in signal spectrograms manually or automatically by defining your autolabeling functions.
- Create data sets that include spectrograms and label masks and use them to train detection and segmentation deep-learning models.
- Export time-frequency labeled signal sets and label definitions to the MATLAB workspace or a file.



Signal Labeling: Label signals in time-frequency domain

This release introduces enhancements to the `signalLabelDefinition` and `labeledSignalSet` objects, and a new signal labeling object to support signal labeling in time-frequency domain.

- The new `labelSpectrogramOptions` object enables you to store spectrogram options for a region-of-interest (ROI) time-frequency label definition.
- The `signalLabelDefinition` object includes these enhancements:
  - You can now set the `LabelType` property to `"roiTimeFrequency"` to indicate an ROI time-frequency label type.
  - The new `TimeFrequencyOptions` and `MemberChannel` name-value arguments enable you to specify options and a member channel for time-frequency map computation.
- The `labeledSignalSet` object supports ROI time-frequency label definitions. You can now specify label definitions that include `"roiTimeFrequency"` labels.
  - Create datastores that point to spectrograms and label masks and use them to train segmentation and detection deep-learning models.
  - Edit time-frequency labels programmatically. Set label values, get label definitions, names, indices, and labeled signals, and remove region values.

spectrogram function: Plot spectrograms in axes and apps

Starting this release, the `spectrogram` function supports a new `Parent` name-value argument that enables you to specify a target over which to plot the spectrogram. Specify the target as one of these objects: `Axes`, `UIAxes`, or `Panel`.

Create Plot Live Editor Task: Visualize function outputs

You can now use the Create Plot Live Editor task to interactively visualize filter responses represented with cascaded transfer functions (CTF).

The Create Plot Live Editor task supports CTF inputs in these Signal Processing Toolbox functions:

- `freqz`
- `grpdelay`
- `impz`
- `zplane`

⚠spectralEntropy function: Specify time information, frequency range, time limits, and obtain time vector associated with spectral entropy output

The `spectralEntropy` function includes enhancements and supports new arguments that enable you to specify time information, frequency range, and time limits to calculate the spectral entropy from a signal or spectrogram. You can also obtain the vector of time values associated with the spectral entropy output.

⚠Compatibility Considerations

The `pentropy` function will be removed in a future release. Use `spectralEntropy` instead. You must update your code to use `spectralEntropy`.

- You can now use the `Range` name-value argument in `spectralEntropy` just like `FrequencyLimits` in `pentropy`.
- You can now use the `TimeLimits` name-value argument in `spectralEntropy` just like in `pentropy`.
- You can now specify the time information corresponding to a spectrogram. Specify the spectrogram with frequencies and times in `spectralEntropy` just like in `pentropy`.
- You can now obtain the time vector associated with the spectral entropy output. Specify the second output argument in `spectralEntropy` just like in `pentropy`.

⚠spectralKurtosis function: Specify time information and obtain frequency vector associated with spectral kurtosis output

The `spectralKurtosis` function now supports specifying a sample time for the input signal as a `duration` scalar. You can also obtain the vector of frequencies at which the spectral kurtosis is computed.

⚠Compatibility Considerations

The `pkurtosis` function will be removed in a future release. Use `spectralKurtosis` instead. You must update your code to use `spectralKurtosis`.

You can now obtain the frequency vector associated with the spectral kurtosis output. Specify the fifth output argument in `spectralKurtosis` just like the second output argument in `pkurtosis`.

⚠stftLayer and istftLayer objects: Initialize layer weights through dlnetwork initialization

Starting this release, you can initialize `stftLayer` and `istftLayer` layer weights in `dlnetwork` (Deep Learning Toolbox) objects through the `initialize` (Deep Learning Toolbox) function.

⚠Compatibility Considerations

For `stftLayer` and `istftLayer`, the default value of the `Weights` property is now `[]`. For more information, see “New default value for stftLayer and istftLayer learnable parameters” on page 3-7.

Experiment Manager Templates: Enhancements

This release introduces updates to the Signal Classification Using Transfer Learning template in the Experiment Manager (Deep Learning Toolbox) app to leverage initialization support for signal classification workflows using transfer learning. This template also incorporates suggested hyperparameters, so you can now experiment with more values from the list of suggested hyperparameters.

The Experiment Manager app enables you to train deep learning networks under various initial conditions and compare the results. You can experiment with different hyperparameter values, training sets, or network architectures. You must have a Deep Learning Toolbox™ license to train a neural network with the app.

Use Experiment Manager Templates for Signal Processing Workflows provides an overview of the Experiment Manager templates you can use for signal processing workflows.

★AI Application Examples: Analyze, classify, and estimate radar and wireless data for spectrum sensing applications

This release introduces examples that apply signal processing techniques and deep learning to the analysis, classification, and estimation of radar and wireless data for spectrum sensing applications.

- Automated Labeling of Time-Frequency Regions for AI-Based Spectrum Sensing Applications uses rule-based methods or unsupervised learning techniques to help automate time-frequency data labeling.
- Direction-of-Arrival Estimation Using Deep Learning estimates direction of arrival using a deep learning estimator that outperforms the traditional MUSIC algorithm in challenging scenarios.
- CBRS Band Radar Parameter Estimation Using YOLOX detects radar pulses in noise and estimates the pulse parameters using a combination of time-frequency maps and a deep-learning object detector.
- Export Labeled Data from Signal Labeler for AI-Based Spectrum Sensing Applications uses deep learning networks and the Signal Labeler app to identify frames from the Bluetooth® and Wi-Fi® wireless standards.

Deep Learning: Code generation support for short-time Fourier transform

The `dlstft` function, the `stftLayer` object, and the `istftLayer` object now support C/C++ and GPU code generation.

You must have MATLAB Coder to generate standalone C and C++ code for the supported function and objects. You must have a MATLAB Coder and GPU Coder license to generate CUDA code for graphical processor units (GPU).

For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

GPU support for digital filter design, signal transforms, and vibration analysis

These Signal Processing Toolbox functions now support `gpuArray` objects: `fir1`, `envelope`, and `envspectrum`.

You must have a Parallel Computing Toolbox license to use `gpuArray` objects with the supported functions. For more details, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox). To find which GPUs are supported, see GPU Computing Requirements (Parallel Computing Toolbox).

⚠Functionality being removed or changed

Signal Labeler has changed

Behavior change

Signal Labeler no longer supports the feature extraction mode, which is now available as an app. To extract signal features, open Signal Feature Extractor from the MATLAB Toolstrip or the Command Window.

Signal Feature Extractor does not support importing labeled signal sets with feature labels created in Signal Labeler. If you used Signal Labeler to create labeled signal sets with features, you must regenerate the features in Signal Feature Extractor from signal members or from a labeled signal set without feature labels.

New default value for stftLayer and istftLayer learnable parameters
Behavior change

For `stftLayer` and `istftLayer`, the default value of the `Weights` property is now `[]`. Prior to R2025a:

- `stftLayer` set the default value to the analysis window used to compute the transform.
- `istftLayer` set the default value to the synthesis window used to compute the transform.

R2024b

Version: 24.2

New Features

Bug Fixes

Compatibility Considerations

★Digital Filter Design: Design classic IIR filters using cascaded transfer functions

Starting this release, you can design IIR digital filters and output coefficients with Cascaded Transfer Functions (CTF) using the `butter`, `cheby1`, `cheby2`, and `ellip` functions.

The functions return matrices that list denominator and numerator polynomial coefficients of the filter transfer function, represented as a cascade of filter sections. That way, you can generate IIR filters that offer better numerical stability than with single-section transfer functions.

```
[B,A] = butter(7,[0.3 0.5],"bandpass","ctf")
```

```
B = 4×5
    0.0975    0.1950    0.0975         0         0
    0.0975    0.3900    0.5850    0.3900    0.0975
    0.0975   -0.1950         0    0.1950   -0.0975
    0.0975   -0.3900    0.5850   -0.3900    0.0975

A = 4×5
    1.0000   -0.4905    0.5095         0         0
    1.0000   -0.9935    1.3076   -0.5436    0.3076
    1.0000   -1.0346    1.4636   -0.6860    0.4636
    1.0000   -1.1148    1.7687   -0.9644    0.7687
```

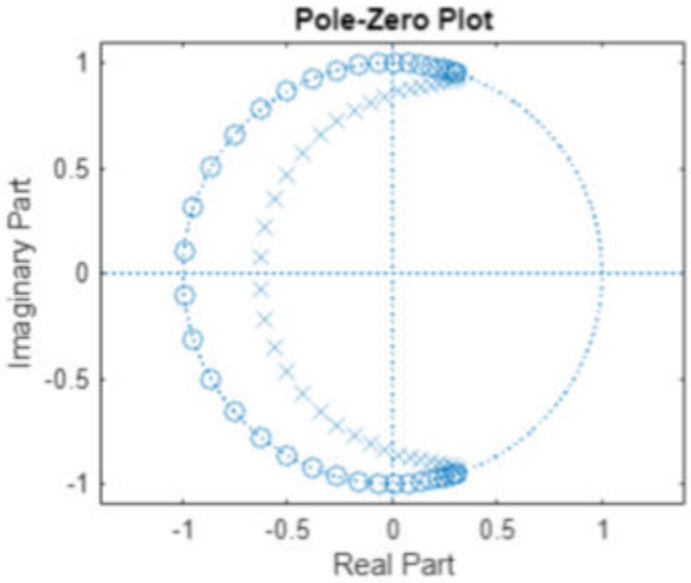
This release also introduces the `ctf2zp` function, which enables you to convert filter systems in the CTF format to their zero-pole-gain representation.

Digital Filter Analysis: Visualize and analyze filters using cascaded transfer functions

Starting this release, these Signal Processing Toolbox filter analysis and visualization functions support Cascaded Transfer Functions:

- Time-Domain Responses — `impzlength`, `impz`, and `stepz`
- Frequency-Domain Responses — `freqz`, `grpdelay`, `phasedelay`, `phasez`, `zerophase`, and `zplane`
- Filter Exploration — `filtord`, `islinphase`, `ismaxphase`, `isminphase`, and `isstable`

```
[B,A] = cheby2(40,50,0.4,"ctf");
zplane(B,A,"ctf")
```

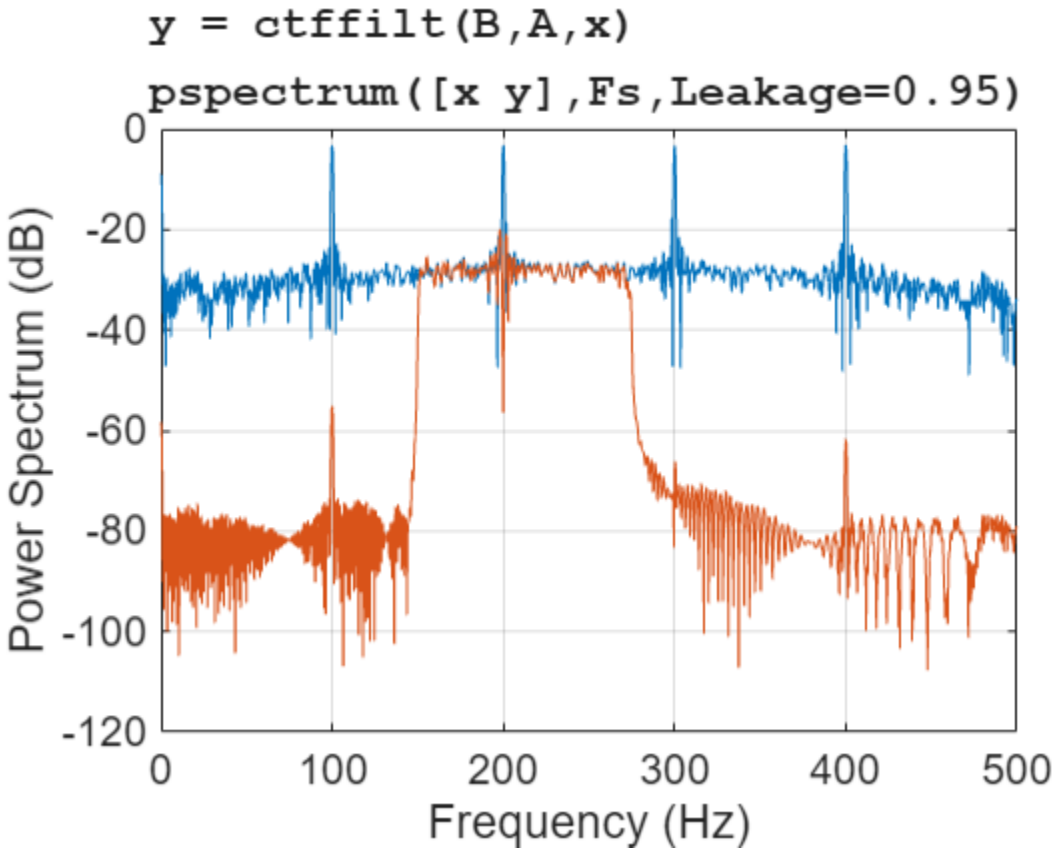


Digital Filtering: Filter signals using cascaded transfer functions

This release introduces the `ctffilt` function, which enables you to filter signals using transfer functions represented with Cascaded Transfer Functions (CTF).

- Filter signal vectors, matrices, multidimensional arrays, or MATLAB timetables, and choose the dimension over which to perform the filtering.
- Perform successive signal filtering by specifying initial and final conditions for the filter states.

Starting this release, `filtfilt` supports filters in the CTF format.



★Peak Finding: Refine peak value and location estimates

This release introduces the `refinepeaks` function, which enables you to refine peak value and location estimates using quadratic or nonlinear least-squares processing.

★Feature Extraction: Extract scalar features and integrate with datastore objects

The `signalTimeFeatureExtractor`, `signalFrequencyFeatureExtractor`, and `signalTimeFrequencyFeatureExtractor` objects incorporate new functionalities that enable you to extract signal features for use in AI models.

- Extract scalar features, such as peak value, kurtosis, and skewness, from signal features in the time domain, the frequency domain, and the time-frequency domain.
- Extract features from signals stored in datastores:
  - The `extract` function now accepts `signalDatastore` and `audioDatastore` (Audio Toolbox) objects.
  - The new `UseParallel` name-value argument in the `extract` function enables you to accelerate code by automatically running computations in parallel. You must have a Parallel Computing Toolbox license to use this functionality.

spectralEntropy and spectralKurtosis functions: Customize feature extraction and input timetables

The `spectralEntropy` and `spectralKurtosis` functions support new input arguments to customize the computation of spectral entropy and spectral kurtosis from signals and spectrograms.

- `spectralEntropy` and `spectralKurtosis` now support MATLAB timetables.
- `spectralEntropy` introduces the `Scaled` and `Instantaneous` name-value arguments.
  - If `Scaled` is `true`, the function scales the spectral entropy of the input signal or spectrogram by the spectral entropy of the corresponding white noise.
  - If `Instantaneous` is `true`, the function computes the instantaneous spectral entropy as a time series. Otherwise, the function computes the spectral entropy value of the entire input signal or spectrogram as a scalar.
- `spectralKurtosis` introduces the `Scaled` and `ConfidenceLevel` name-value arguments.
  - If `Scaled` is `true`, the function returns the frequency-averaged spectral kurtosis. If `Scaled` is `false`, the function returns the numeric kurtosis of the signal spectrogram.
  - `ConfidenceLevel` lets you specify the width of the range within which the input signal is determined to be Gaussian and stationary.

signalDatastore object: Cast data in single precision and output GPU arrays

The `signalDatastore` object can now cast signal data in single precision and output `gpuArray` objects.

- Use the `OutputDataType` name-value argument to specify the output type of the signal data as double precision or single precision.
- Use the `OutputEnvironment` name-value argument to specify whether to return the data as numeric arrays or `gpuArray` objects.

You must have a Parallel Computing Toolbox license to use `gpuArray` objects. For more details, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox). To see which GPUs are supported, see GPU Computing Requirements (Parallel Computing Toolbox).

Filter Analyzer App: Drag filters from Filters table to displays

You can now drag filters from the **Filters** table onto any display for visualization using the Filter Analyzer app.

Signal Labeler App: Export labeled signal sets to Diagnostic Feature Designer

Starting this release, you can export signal sets labeled using Signal Labeler directly to Diagnostic Feature Designer (Predictive Maintenance Toolbox). You must have a Predictive Maintenance Toolbox™ license to use Diagnostic Feature Designer.

Experiment Manager Templates: Quickly set up transfer learning signal classification experiments

The Experiment Manager (Deep Learning Toolbox) app now includes a preconfigured template that supports signal classification workflows using transfer learning.

The Experiment Manager app enables you to train deep learning networks under various initial conditions and compare the results. You can experiment with different hyperparameter values, training sets, or network architectures. You must have a Deep Learning Toolbox license to use this functionality.

AI Application Examples: Detect anomalies, denoise signals, extract features, and label signals

This release introduces examples that apply signal processing techniques and deep learning to the analysis of biomedical, geological, and wireless network data.

- Wireless Resource Allocation Using Graph Neural Network demonstrates the use of graph neural networks for power allocation in wireless networks.
- Denoise Signals with Generative Adversarial Networks uses autoencoders and generative adversarial networks to denoise signals.

Feature Extraction Examples: Accelerate feature extraction using datastores, GPUs, and parallel computing

This release introduces and updates examples that showcase computational acceleration in multidomain signal feature extraction using datastores, graphical processing units (GPUs), and parallel computing to identify bearing faults in mechanical systems.

- Machine Learning and Deep Learning Classification Using Signal Feature Extraction Objects shows how to extract multidomain features from a `signalDatastore` object and trains a machine learning model and a deep learning network using the extracted signal features.
- Accelerate Signal Feature Extraction and Classification Using a GPU uses a GPU to speed up feature extraction for AI-assisted classification of vibration data.
- Accelerate Signal Feature Extraction and Classification Using a Parallel Pool of Workers uses parallel computing to speed up feature extraction for AI-assisted classification of vibration data.

Application Example: Denoise signals using Savitzky-Golay filters with weighting vectors

This release introduces an example that applies signal processing techniques to the filtering of synthetic signals with additive noise. Design and Analyze Savitzky-Golay Filters shows how to design, analyze, and apply Savitzky-Golay filters using weighting vectors and optimization for best smoothness.

C/C++ code generation support for descriptive statistics, signal modeling, vibration analysis, and waveform generation

These Signal Processing Toolbox functions now support C/C++ code generation:

- Descriptive Statistics — `seqperiod`
- Signal Modeling — `ac2poly`, `ac2rc`, `invfreqs`, `invfreqz`, `is2rc`, `lar2rc`, `lpc`, `rc2is`, `rc2lar`, and `schurrc`
- Vibration Analysis — `modalfit` and `modalfrf`
- Waveform Generation — `udecode`, `shiftdata`, and `unshiftdata`

These Signal Processing Toolbox functions now support MATLAB timetables for code generation:

- Spectral Analysis — `pspectrum`
- Time-Frequency Analysis — `instbw`, `instfreq`, and `kurtogram`

You must have MATLAB Coder to generate standalone C and C++ code for the supported functions. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

GPU support for time-frequency analysis and feature extraction

These Signal Processing Toolbox functions now support `gpuArray` objects: `hht`, `signalTimeFrequencyFeatureExtractor`, and `tfridge`.

You must have a Parallel Computing Toolbox license to use `gpuArray` objects with the supported functions. For more details, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox). To see which GPUs are supported, see GPU Computing Requirements (Parallel Computing Toolbox).

GPU code generation support for feature extraction and spectral analysis functions

The `pwelch` function now supports code generation for graphical processing units (GPUs).

The `chebwin`, `findpeaks`, and `kaiser` functions now have enhanced performance for code generation.

You must have a MATLAB Coder and GPU Coder license to generate CUDA code.

Single-precision support for spectral windows and waveform generation

These Signal Processing Toolbox functions now support single precision:

- Spectral Windows — `barthannwin`, `bartlett`, `blackman`, `blackmanharris`, `bohmanwin`, `chebwin`, `flattopwin`, `hann`, `parzenwin` and `triang`
- Waveform Generation — `chirp`

⚠Functionality being removed or changed

FVTool will be removed
Still runs

FVTool will be removed in a future release. Use Filter Analyzer instead. There are differences that require updates to your code.

Consider these filters:

```
[b,a] = ellip(5,5,60,[0.2 0.45]);
dFd = designfilt("bandpassfir", ...
    SampleRate=2e3,PassbandRipple=5, ...
    StopbandFrequency1=500,PassbandFrequency1=600, ...
    StopbandAttenuation1=80, ...
    PassbandFrequency2=750,StopbandFrequency2=900, ...
    StopbandAttenuation2=40);
```

Given the filter numerator coefficients `b`, denominator coefficients `a`, and the `digitalFilter` object `dFd`, you must make the following updates to your code.

Original Code in R2024a or Earlier	Updated Code in R2024b
`fvtool(b,a,dFd)`	`filterAnalyzer(b,a,dFd)`
`fvtool(b,a,dFd,Analysis="freq")`	`filterAnalyzer(b,a,dFd, ...` `  Analysis="magnitude",Overlay="phase")`
`fvtool(b,a,Fs=1000)`	`filterAnalyzer(b,a,SampleRates=1000)`

`fvtool(b,a,dFd,NumberofPoints=512, ...`
`  FrequencyRange="[0, 2pi)", ...`
`  FrequencyScale="Log")`
	`filterAnalyzer(b,a,dFd,NFFT=512, ...`
	`  FrequencyRange="twosided", ...`
	`  FrequencyScale="log")`

`hfvt = fvtool(dFd);` `addfilter(hfvt,dfilt.df1(b,a))`	`fa = filterAnalyzer(dFd);` `addFilters(fa,b,a)`
`hfvt = fvtool(dFd);` `setfilter(hfvt,dfilt.df1(b,a))`	`fa = filterAnalyzer(dFd,FilterNames="df");` `replaceFilters(fa,b,a,FilterNames="df")`

`hfvt = fvtool(b,a,dFd);`
`deletefilter(hfvt,2)`
	`fa = filterAnalyzer(b,a,dFd, ...`
	`  FilterNames=["ba" "dFd"]);`
	`  deleteFilters(fa,FilterNames="dFd")`

`hfvt = fvtool(b,a,dFd);` `legend(hfvt,"ba","dFd")`	`filterAnalyzer(b,a,dFd, ...` `  FilterNames=["ba" "dFd"])`
`hfvt = fvtool(b,a,dFd);` `zoom(hfvt,[0.4 0.7 -30 0])`	`fa = filterAnalyzer(b,a,dFd);` `zoom(fa,"xy",[0.4 0.7 -30 0])`
`fvtool(b,a,dFd,Analysis="noisepower")`	`filterAnalyzer(b,a,dFd,Analysis="noisepsd")`

The second order sections (SOS) format is not supported in Filter Analyzer. Use the Cascaded Transfer Functions format instead. Given a filter specified as an SOS matrix `sos`, you must make the following updates to your code.

Original Code in R2024a or Earlier	Updated Code in R2024b

`fvtool(sos)`	`filterAnalyzer(sos(:,1:3),sos(:,4:6))`
	
	or
	
	`[ctfNum,ctfDen] = sos2ctf(sos);`
	`filterAnalyzer(ctfNum,ctfDen)`

`hfvt = fvtool(sos);`
`set(hfvt.SOSViewSettings, ...`
`  View="cumulative")`
	`[ctfNum,ctfDen] = sos2ctf(sos);`
	`fa = filterAnalyzer(ctfNum,ctfDen, ...`
	`  CTFAnalysisMode="cumulative");`

Original Code in R2024a or Earlier	Updated Code in R2024b

`hfvt = fvtool(sos);`
`set(hfvt.SOSViewSettings, ...`
`    View="userdefined", ...`
`    UserDefined={3,1})`
	`[ctfNum,ctfDen] = sos2ctf(sos);`
	`fa = filterAnalyzer(ctfNum,ctfDen, ...`
	`    CTFAnalysisMode="specify", ...`
	`    CTFAnalysisSections={3,1});`

pentropy will be removed
Still runs

The `pentropy` function will be removed in a future release. Use `spectralEntropy` instead. You must update your code to use `spectralEntropy`.

Consider a signal `x` sampled at a rate `fs`:

```
fs = 1000;
Ts = seconds(1/fs);
t = (0:1/fs:10)';

x = chirp(t,300,10,400);
xT = timetable(x,SampleRate=fs);
```

To get the spectral entropy with `spectralEntropy`, make these updates to your code.

Original Code in R2024a or Earlier	Updated Code in R2024b
`se = pentropy(xT);`	`[S,f,T] = pspectrum(xT,"spectrogram");` `sE = timetable(T,spectralEntropy(S,f));`
`se = pentropy(x,fs);`	`[S,f] = pspectrum(x,fs,"spectrogram");` `sE = spectralEntropy(S,f);`
`se = pentropy(x,Ts);`	`[S,f] = pspectrum(x,Ts,"spectrogram");` `sE = spectralEntropy(S,f);`
`se = pentropy(x,t);`	`[S,f] = pspectrum(x,t,"spectrogram");` `sE = spectralEntropy(S,f);`
`[S,f,T] = pspectrum(x,fs,"spectrogram");` `se = pentropy(S,f,T);`	`[S,f] = pspectrum(x,fs,"spectrogram");` `sE = spectralEntropy(S,f);`
`[S,f,T] = pspectrum(x,Ts,"spectrogram");` `se = pentropy(S,f,T);`	`[S,f] = pspectrum(x,Ts,"spectrogram");` `sE = spectralEntropy(S,f);`
`[S,f,T] = pspectrum(x,t,"spectrogram");` `se = pentropy(S,f,T);`	`[S,f] = pspectrum(x,t,"spectrogram");` `sE = spectralEntropy(S,f);`

- You can use the `Scaled` name-value argument in `spectralEntropy` just like in `pentropy`.
- You can use the `Instantaneous` name-value argument in `spectralEntropy` just like in `pentropy`.
- `spectralEntropy` does not support the `FrequencyLimits` name-value argument of `pentropy`.
- `spectralEntropy` does not support the `TimeLimits` name-value argument of `pentropy`.

pkurtosis will be removed
Still runs

The `pkurtosis` function will be removed in a future release. Use `spectralKurtosis` instead. You must update your code to use `spectralKurtosis`.

Consider a signal `x` sampled at a rate `fs`:

```
fs = 1000;
Ts = seconds(1/fs);
t = (0:1/fs:10)';

x = chirp(t,300,10,400);
xT = timetable(x,SampleRate=fs);

function y = wlen(x)
    wdiv = 2.^[1 3:7];
    y = ceil(x/wdiv(find(x < 2.^[6 11:14 Inf],1)));
end

wnd = 500;

[Sstft,Fstft] = stft(x,fs, ...
    FrequencyRange="onesided", ...
    Window=triang(600),FFTLength=1512,OverlapLength=421);
```

To get the spectral kurtosis with `spectralKurtosis`, make these updates to your code.

Original Code in R2024a or Earlier	Updated Code in R2024b
Time-Domain Responses	

`sk = pkurtosis(xT);`	`fres = xT.Properties.SampleRate/winlen(height(xT));`
	`[S,F] = pspectrum(xT,"spectrogram", ...`
	`  FrequencyResolution=fres, ...`
	`  OverlapPercent=80);`
	`sK = spectralKurtosis(S,F,Scaled=false);`
	`sK(F<=fres|F>=xT.Properties.SampleRate/2-fres) = 0;`

`sk = pkurtosis(x);`	`fres = 1/winlen(length(x));`
	`[S,F] = pspectrum(x,1,"spectrogram", ...`
	`  FrequencyResolution=fres, ...`
	`  OverlapPercent=80);`
	`sK = spectralKurtosis(S,F,Scaled=false);`
	`sK(F<=fres|F>=1/2-fres) = 0;`

`sk = pkurtosis(x,fs);`	`fres = fs/winlen(length(x));`
	`[S,F] = pspectrum(x,fs,"spectrogram", ...`
	`  FrequencyResolution=fres, ...`
	`  OverlapPercent=80);`
	`sK = spectralKurtosis(S,F,Scaled=false);`
	`sK(F<=fres|F>=fs/2-fres) = 0;`

`sk = pkurtosis(x,Ts);`	`fres = 1/(seconds(Ts)*winlen(length(x)));`
	`[S,F] = pspectrum(x,Ts,"spectrogram", ...`
	`  FrequencyResolution=fres, ...`
	`  OverlapPercent=80);`
	`sK = spectralKurtosis(S,F,Scaled=false);`
	`sK(F<=fres|F>=1/2/seconds(Ts)-fres) = 0;`

`sk = pkurtosis(x,t);`	`fres = 1/(mean(diff(t))*winlen(length(x)));`
	`[S,F] = pspectrum(x,t,"spectrogram", ...`
	`  FrequencyResolution=fres, ...`
	`  OverlapPercent=80);`
	`sK = spectralKurtosis(S,F,Scaled=false);`
	`sK(F<=fres|F>=1/2/mean(diff(t))-fres) = 0;`

Time-Domain Responses with Specified Window Resolution	

Original Code in R2024a or Earlier	Updated Code in R2024b

`sk = pkurtosis(xT,wnd);`	`fres = xT.Properties.SampleRate/wnd;`
	`[S,F] = pspectrum(xT,"spectrogram", ...`
	`    FrequencyResolution=fres, ...`
	`    OverlapPercent=80);`
	`sK = spectralKurtosis(S,F,Scaled=false);`
	`sK(F<=fres|F>=xT.Properties.SampleRate/2-fres) = 0;`

`sk = pkurtosis(x,[],wnd);`	`fres = 1/wnd;`
	`[S,F] = pspectrum(x,1,"spectrogram", ...`
	`    FrequencyResolution=fres, ...`
	`    OverlapPercent=80);`
	`sK = spectralKurtosis(S,F,Scaled=false);`
	`sK(F<=fres|F>=1/2-fres) = 0;`

`sk = pkurtosis(x,fs,wnd);`	`fres = fs/wnd;`
	`[S,F] = pspectrum(x,fs,"spectrogram", ...`
	`    FrequencyResolution=fres, ...`
	`    OverlapPercent=80);`
	`sK = spectralKurtosis(S,F,Scaled=false);`
	`sK(F<=fres|F>=fs/2-fres) = 0;`

`sk = pkurtosis(x,Ts,wnd);`	`fres = 1/(seconds(Ts)*wnd);`
	`[S,F] = pspectrum(x,Ts,"spectrogram", ...`
	`    FrequencyResolution=fres, ...`
	`    OverlapPercent=80);`
	`sK = spectralKurtosis(S,F,Scaled=false);`
	`sK(F<=fres|F>=1/seconds(Ts)/2-fres) = 0;`

`sk = pkurtosis(x,t,wnd);`	`fres = 1/(mean(diff(t))*wnd);`
	`[S,F] = pspectrum(x,t,"spectrogram", ...`
	`    FrequencyResolution=fres, ...`
	`    OverlapPercent=80);`
	`sK = spectralKurtosis(S,F,Scaled=false);`
	`sK(F<=fres|F>=1/mean(diff(t))/2-fres) = 0;`

Frequency-Domain Responses	

`sk = pkurtosis(Sstft,fs,Fstft,wnd);`	`sK = spectralKurtosis(abs(Sstft).^2,Fstft, ...`
	`    Scaled=false);`
	`sK(Fstft<=fs/wnd|Fstft>=fs*(1/2-1/wnd)) = 0;`

`sk = pkurtosis(Sstft,fs,Fstft,wnd);`	`sK = spectralKurtosis(x,fs, ...`
	`    Window=triang(600),FFTLength=1512, ...`
	`    OverlapLength=421,Scaled=false);`
	`sK(Fstft<=fs/wnd|Fstft>=fs*(1/2-1/wnd)) = 0;`

- You can use the `ConfidenceLevel` name-value argument in `spectralKurtosis` just like in `pkurtosis`.

digitalFilter has changed
Behavior change

The coefficients behavior of the `digitalFilter` object has changed. Obtain the numerator and denominator coefficients of digital filters designed with Cascaded Transfer Functions. Starting this release, the `Numerator` and `Denominator` properties replace the `Coefficients` property.

Considering a `digitalFilter` filter object `filt`, you will require the following updates to your code to get the filter coefficients `B` (numerator) and `A` (denominator).

Impulse Response Type	Original Code in R2024a or Earlier	Updated Code in R2024b
IIR	`B = filt.Coefficients(:,1:3);` `A = filt.Coefficients(:,4:6);`	`B = filt.Numerator;` `A = filt.Denominator;`
FIR	`B = filt.Coefficients;`	`B = filt.Numerator;`

R2024a

Version: 24.1

New Features

Bug Fixes

Compatibility Considerations

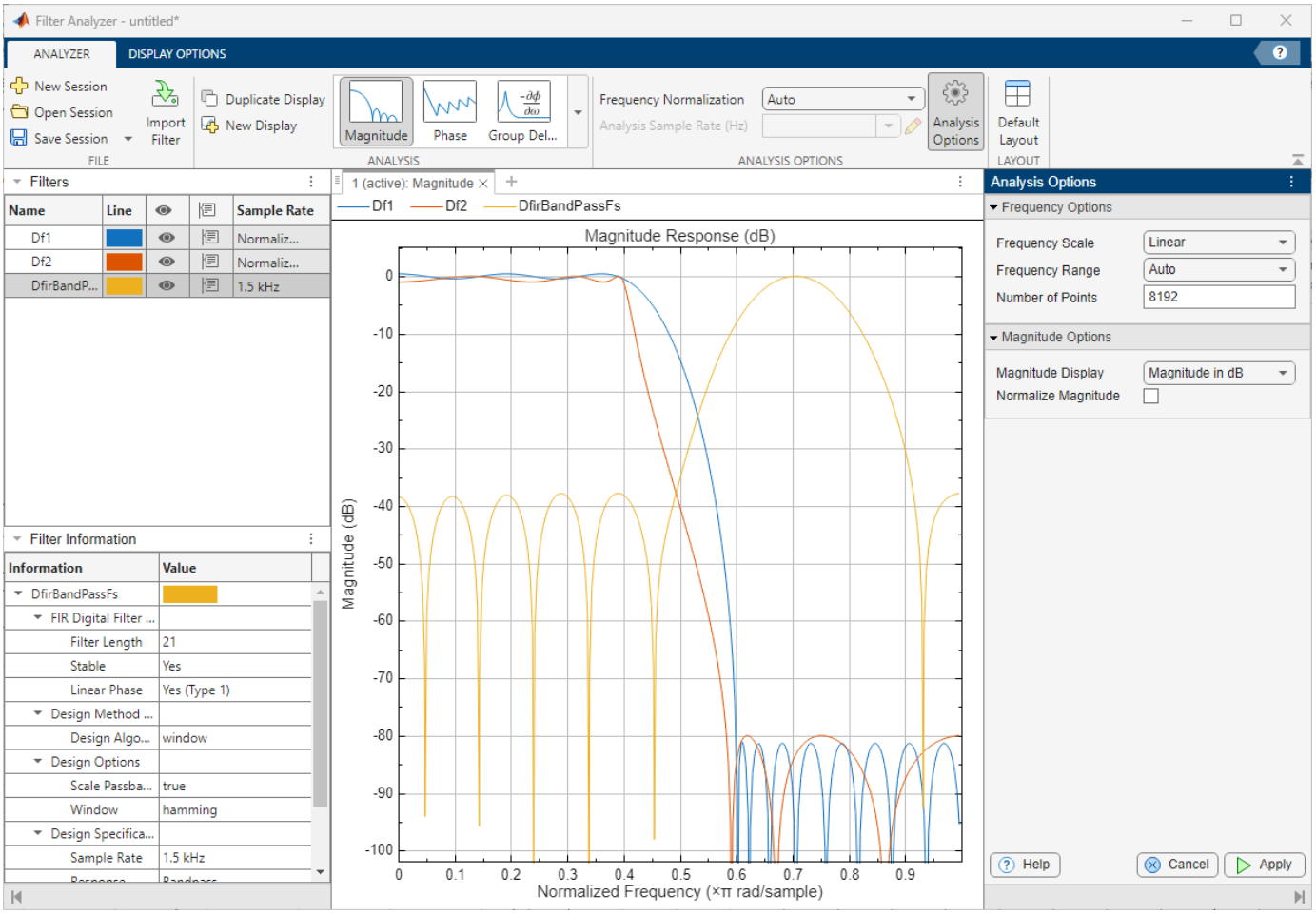
★Filter Analyzer App: View, analyze, and compare filters

The new Filter Analyzer app enables you to:

- Import filter objects or filter coefficients.
- View, analyze, and compare responses of multiple digital filters.
- View a list of filters in the Filters table and the details for each filter in the Filter Information table.
- Plot a new filter analysis plot in a separate display window.
- Specify filter sample rate and analysis sample rate separately.
- Save the state of the current session, including filters, displays, and analysis options, for use in a future session of the app.

You can use the command-line interface to add, remove, and update filters, displays, and analysis options.

This app provides you with greater interactivity and flexibility while you visualize and analyze your digital filters.



Filter Design: Use cascaded transfer functions

This release introduces cascaded transfer functions (CTF) as a new filter format to Signal Processing Toolbox.

- The new Filter Analyzer app uses the format for IIR filters.
- The `sos2ctf` and `zp2ctf` functions enable you to convert second-order section or zero-pole-gain representations of filters to cascaded transfer function form.
- The `scaleFilterSections` function lets you specify a vector of scale factors or a scalar overall factor to weight the different sections of a filter in cascaded transfer function form.

★Feature Extraction: Extract time-frequency features of signals

The new `signalTimeFrequencyFeatureExtractor` object generates time-frequency feature tables or matrices for use in AI models.

- Extract features such as instantaneous bandwidth, instantaneous frequency, spectral entropy, spectral kurtosis, and time-frequency ridges.
- Extract features starting from signal spectrograms, continuous and nondecimated discrete wavelet-based decompositions, or data-adaptive methods.
- Extract clearance factor, energy, skewness, among other scalar features, from the signal features.

You can also generate a MATLAB function based on the object that is compatible with C/C++ code generation. You must have MATLAB Coder to generate C/C++ code.

You must have Wavelet Toolbox™ to use wavelet-based functionality.

deepSignalAnomalyDetector Object: Perform streaming anomaly detection and save models

The `deepSignalAnomalyDetector` object now has a third mode, `deepSignalAnomalyDetectorLSTMForecaster`, that uses a long short-term memory network to detect anomalies in streaming workflows and thus in real time.

The new `saveModel` object function creates a MAT file containing the trained network and parameters corresponding to any object generated by `deepSignalAnomalyDetector`. Use the generated file with the Deep Signal Anomaly Detector (DSP System Toolbox) Simulink block.

You must have a Deep Learning Toolbox license to use this functionality.

Partition signals and label sequences into frames

Use the `framesig` and `framelbl` functions to partition signals and label sequences into frames.

- `framesig` enables you to divide each channel of a signal into a matrix of overlapping or underlapping frames and window each frame using a window of your choice.
- `framelbl` enables you to keep track of attribute, region-of-interest, and point labels of signals when these signals are partitioned into frames.
    - Divide label sequences of categorical, string, numeric, logical, or table types.

- Resolve ambiguous regions in logical matrix inputs and region-of-interest tables.
- Consolidate labels within a frame into a single value.
- Support overlap, underlap, and successive framing.

★Signal Analyzer App: Listen to audio signals

This release enhances the Signal Analyzer app with audio playback controls that play signals.

Signal Analyzer App: Fill missing data

Starting this release, you can preprocess signals in the Signal Analyzer app by finding and filling gaps in the data. You can fill missing data with constant values, combinations of neighboring samples, or moving averages. You can also interpolate or apply an autoregressive model.

Signal Labeler App: Compute and compare more types of spectra and spectrograms

Starting this release, you have more flexibility when computing spectra or spectrograms using the Signal Labeler app. You can control the window length, the number of discrete Fourier transform points, and the resolution bandwidth. You can also target a specific resolution bandwidth.

⚠Signal Labeler App: Label more easily

Starting this release, you can label more easily using the Signal Labeler app. You can navigate between members more easily, hide and show signals, access zoom controls close to the axes, move legends, and resize axes for easier labeling.

⚠Compatibility Considerations

The behavior of Signal Labeler has changed. For more information, see Signal Labeler has changed.

Deep Learning: Inverse short-time Fourier transform layer

This release introduces a learnable signal processing layer that enables you to compute the inverse short-time Fourier transform within a deep learning network. The `istftLayer` object is available from the command line and from the Deep Network Designer (Deep Learning Toolbox) app. You can use this layer with the `dlnetwork` (Deep Learning Toolbox) architecture. You must have Deep Learning Toolbox to use the `istftLayer` object. For a list of available layers, see List of Deep Learning Layers (Deep Learning Toolbox).

dlistft Function: Compute deep learning inverse short-time Fourier transforms

This release introduces the `dlistft` function, which enables you to compute the deep-learning inverse short-time Fourier transform. The function outputs the inverse transforms as `dlarray` objects that enable automatic differentiation and can be used in custom training loops. You must have Deep Learning Toolbox to use the `dlistft` function.

Experiment Manager Templates: Quickly set up deep learning experiments with signals

Starting this release, the Experiment Manager (Deep Learning Toolbox) app includes preconfigured templates that support signal processing workflows. Using these preconfigured templates, you can perform:

- Signal segmentation
- Signal classification
- Signal regression

The Experiment Manager app enables you to train deep learning networks under various initial conditions and compare the results. You can experiment with different hyperparameter values, custom training functions, training sets, or network architectures. You must have a Deep Learning Toolbox license to use this functionality.

AI Application Examples: Detect anomalies, denoise signals, extract features, and label signals

This release introduces examples that apply signal processing techniques and deep learning to the analysis of synthetic, acoustic, and mechanical data.

- Anomaly Detection Using Convolutional Autoencoder with Wavelet Scattering Sequences shows how to detect anomalies in acoustic data using the `deepSignalAnomalyDetector` and `waveletScattering` (Wavelet Toolbox) objects.
- Denoise EEG Signals Using Differentiable Signal Processing Layers shows how to remove electro-oculogram (EOG) noise from electroencephalogram (EEG) signals using deep learning regression.
- Machine and Deep Learning Classification Using Signal Feature Extraction Objects uses signal feature extraction objects to extract multidomain features to identify faulty bearing signals in mechanical systems.
- Feature Selection Based on Deep Learning Interpretability for Signal Classification Applications uses the locally interpretable model-agnostic explanation technique to interpret decision-making processes of a deep learning network and performs feature selection to reduce training time and network complexity.
- Export Labeled Data from Signal Labeler for Deep Learning Classification shows how to label signals and export data using the Signal Labeler app to train a deep learning classifier.

Application Example: Perform graph signal processing

This release introduces an example that applies signal processing techniques to the analysis of medical data. Graph Signal Processing and Brain Signal Analysis processes a brain graph and decomposes brain signals into aligned and liberal components to analyze brain activity.

Generate C/C++ code for signal generation and spectral analysis

These Signal Processing Toolbox functions now support C/C++ code generation:

- Waveform Generation — `demod` and `vco`

- Spectral Analysis — `window`

The `pentropy` function now supports MATLAB timetables, `datetime` arrays, and `duration` arrays for code generation. The `rpmtrack` function supports MATLAB timetables. The `resample` function supports `datetime` arrays.

These Signal Processing Toolbox functions now support single-precision variable-size windows for code generation:

- Spectral Analysis — `cpsd`, `mscohere`, `periodogram`, `pwelch`, `spectrogram`, `tfestimate`, and `xspectrogram`

You must have MATLAB Coder to generate standalone C and C++ code for the supported functions. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

GPU code generation support for preprocessing and spectral analysis functions

The `hamming`, `blackman`, `flattopwin`, `hann`, `periodogram`, `upsample`, `downsample`, `hilbert`, and `square` functions now support code generation for graphical processing units (GPUs).

You must have MATLAB Coder and GPU Coder to generate CUDA code.

⚠Functionality being removed or changed

Signal Labeler has changed

Behavior change

The behavior of the Signal Labeler app has changed.

- Member-by-member navigation has been moved to the main app. There is no need to enter a specialized mode.
- You can now show or hide a signal in the display by clicking its name in the time-domain view legend. You can also move the legend by clicking and dragging.
- You can now resize the label viewer and the time-domain, frequency-domain, and time-frequency views of the signals you are labeling.
- The time-domain view no longer supports the **Show Markers** and **Normalize Y Axis** options.
- To zoom in and out or switch to draw mode, use the axes toolbar. The only zooming action that still supports context menu and keyboard shortcuts is Zoom to label.
- To select labels in the label viewer, you must be in select mode.
- When labeling regions of interest or points in draw mode, use the mouse to adjust location or size or type a location value. Fine-tuning location value using keyboard shortcuts is no longer supported.

R2023b

Version: 23.2

New Features

Bug Fixes

Compatibility Considerations

Signal Labeler App: Automatically label bounded signal regions

The Signal Labeler app now enables you to locate and label signal regions with values greater or smaller than a given threshold or inside or outside a given set of ranges. You can specify a minimum region length to filter out false positives.

Signal Labeler App: Automatically classify sounds in audio signals and label them

The Signal Labeler app now provides automatic classification of sounds in audio signals. To perform the classification, the app uses the YAMNet pretrained neural network as implemented in the `classifySound` (Audio Toolbox) function. You can choose to include or exclude specific sounds. You can also specify the minimum duration of detected sound regions, the minimum separation between consecutive regions of the same sound, and the confidence threshold for reporting sounds.

You must have Audio Toolbox and Deep Learning Toolbox to use this functionality.

Signal Analyzer App: Compute and compare more types of spectra

Starting this release, you have more flexibility when computing spectra using the Signal Analyzer app. You can control the window length, the number of discrete Fourier transform points, and the resolution bandwidth.

stftmag2sig Function: Recover phase information from spectrogram magnitudes using gradient descent

The `stftmag2sig` function now enables you to use the gradient descent algorithm to reconstruct a signal starting with the magnitude of its time-frequency representation. The new functionality mitigates the edge effects that affect the reconstruction estimates given by the Griffin-Lim algorithm and does not require the constant overlap-add constraint demanded by `istft`.

You must have a Deep Learning Toolbox license to use the new functionality.

resize, paddata, and trimdata Functions: Change the size of data by adding or removing elements

Change the size of array or tabular data by using the `resize`, `paddata`, and `trimdata` functions. You can specify the dimensions to operate along, the fill value or pattern for padding, and the side of the input data for resizing.

- `resize` adds or removes elements depending on if the length of the input data is less than or greater than the target length. The resized data matches the target length.
- `paddata` only adds elements. If the length of the input data is greater than the target length, the output data is the same as the input data.
- `trimdata` only removes elements. If the length of the input data is less than the target length, the output data is the same as the input data.

Application Example: Compute Shock Response Spectra

This release introduces an example that applies signal processing techniques to the analysis of vibration data. Practical Introduction to Shock Waveform and Shock Response Spectrum introduces the concept of a shock response spectrum, surveys different types of shock response spectra, and computes shock response spectra of synthetic and measured signals.

Single-precision support for spectral analysis and multirate signal processing

These Signal Processing Toolbox functions now support single precision:

- Spectral Analysis — `cpsd`, `mscohere`, `periodogram`, `pwelch`, `spectrogram`, `tfestimate`, and `xspectrogram`.
- Multirate Signal Processing — `decimate`. The `buffer` function now also supports non-double input arguments.

C/C++ Code Generation Support: Code generation for spectral analysis and signal modeling

These Signal Processing Toolbox functions now support C/C++ code generation:

- Spectral Analysis — `dpss` and `pmtm`
- Signal Modeling — `stmcb`
- Waveform Generation — `modulate`

These Signal Processing Toolbox functions now support MATLAB timetables for code generation:

- Spectral Analysis — `pkurtosis` and `pspectrum`
- Time-Frequency Analysis — `instbw`, `instfreq`, and `kurtogram`

The `firls` function now has enhanced performance for code generation.

You must have MATLAB Coder to generate standalone C and C++ code for the supported functions. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

GPU support for spectral analysis and order analysis

These Signal Processing Toolbox functions now support `gpuArray` objects:

- Spectral Analysis — `ifsst` and `tfestimate`
- Order Analysis — `rpmfreqmap` and `rpmordermap`

You must have Parallel Computing Toolbox to use `gpuArray` objects with the supported functions. For more details, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox). To see which GPUs are supported, see GPU Computing Requirements (Parallel Computing Toolbox).

GPU code generation support for time-frequency analysis functions

The `spectrogram`, `wvd`, and `xspectrogram` functions now support code generation for graphical processing units (GPUs).

You must have MATLAB Coder and GPU Coder to generate CUDA code.

⚠Functionality being removed or changed

dlstft combines real and imaginary parts of transform into one output argument
Behavior change

The `dlstft` function now combines the real and imaginary parts of the short-time Fourier transform (STFT) into a single output argument. Previously, the function returned the real part of the transform as the first output argument and the imaginary part as the second. Starting this release, the function returns the frequencies and times at which the STFT is computed as second and third output arguments, respectively. Function calls with four output arguments error out.

Original Code in R2023a or Earlier	Result	Updated Code in R2023b
`[yr,yi] = dlstft(x);`	Runs, but the second output argument has a different interpretation	`y = dlstft(x); yr = real(y); yi = imag(y);`
`[yr,yi,f,t] = dlstft(x);`	Errors	`[y,f,t] = dlstft(x); yr = real(y); yi = imag(y);`

stftLayer weights initialized to analysis window
Behavior change

The short-time Fourier transform (STFT) layer `stftLayer` now initializes the `Weights` learnable parameter to the analysis window used to compute the transform. Previously, the parameter was initialized to an array containing the Gabor atoms for the STFT.

R2023a

Version: 9.2

New Features

Bug Fixes

Signal Anomaly Detection: Detect signal anomalies using deep learning autoencoders

This release introduces the `deepSignalAnomalyDetector` object, which you can use to detect signal anomalies. The detector can implement an autoencoder using a 1-D convolutional neural network (CNN) or a long short-term memory (LSTM) network. You can train the detector object and then use the trained model to detect and plot anomalies in signal data.

You must have a Deep Learning Toolbox license to use this functionality.

Signal Labeling: Label signal points within a range

The new `sigrangebinmask` function locates and labels signal points with values within a target range. You can specify a minimum region length to filter out false positives.

Signal Analyzer App: Find and annotate signal peaks

Starting this release, perform peak measurements for time-domain signals in the Signal Analyzer app. View signal peak values in the **Measurements** table and as annotated markers on the display.

Signal Analyzer App: Rename and delete signals in preprocessing mode

This release enhances the Signal Analyzer app to enable you to delete or rename signals in the preprocessing mode. You can also change the line color of signals in the mode.

Create Plot Live Editor Task: Visualize function outputs

You can now use the Create Plot Live Editor task to interactively visualize filter responses and other outputs for several signal processing functions. You can select different chart types and set optional parameters. The task also automatically generates code that becomes part of your live script.

The Create Plot Live Editor task supports these Signal Processing Toolbox functions:

- `freqz`
- `grpdelay`
- `impz`
- `periodogram`
- `pspectrum`
- `pwelch`
- `spectrogram`
- `zplane`

Application Examples: Detect anomalies in signals and perform fatigue analysis

This release introduces examples that apply signal processing techniques to the analysis of medical and mechanical data.

- Detect Anomalies In Signals Using `deepSignalAnomalyDetector` uses autoencoders to detect abnormal points or segments in time-series data.
- Detect Anomalies in Machinery Using LSTM Autoencoder uses the `deepSignalAnomalyDetector` object to detect anomalies in data from an industrial machine.
- Practical Introduction to Fatigue Analysis Using Rainflow Counting finds the total damage on a mechanical component due to cyclic stress.

Single-precision support for digital filtering and multirate signal processing

The `filtfilt`, `upfirdn`, and `resample` functions now support single-precision data.

C/C++ Code Generation Support: Code generation for digital filter design, multirate signal processing, and waveform generation

These Signal Processing Toolbox functions now support C/C++ code generation:

- Digital Filter Design — `gaussdesign`
- Multirate Signal Processing — `decimate` and `interp`
- Waveform Generation — `buffer`

The `rcosdesign` function now supports variable-size inputs during code generation.

You must have MATLAB Coder to generate standalone C and C++ code for the supported functions. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

GPU support for feature extraction, statistics, and spectral measurements

These Signal Processing Toolbox functions now support `gpuArray` objects:

- Descriptive Statistics — `rssq`
- Feature Extraction — `signalFrequencyFeatureExtractor` and `signalTimeFeatureExtractor`
- Spectral Measurements — `bandpower`, `medfreq`, `obw`, `powerbw`, `sinad`, `snr`, and `thd`

You must have Parallel Computing Toolbox to use `gpuArray` objects with the supported functions. For more details, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox). To see which GPUs are supported, see GPU Computing Requirements (Parallel Computing Toolbox).

R2022b

Version: 9.1

New Features

Bug Fixes

Compatibility Considerations

Labeling Convenience Function: Generate labels from file names

The new `filenames2labels` function creates a list of labels based on the names of files in a specified location or datastore. Use this function to simplify and streamline labeling processes for deep learning workflows with large data sets.

alignsignals Function: Align signals based on rising edge or peak locations

Starting this release, you can use the `alignsignals` function to align two signals based on the locations of their peaks or rising edges.

Signal Analyzer App: Interactively preprocess signals with various functions and actions

The new preprocessing mode of the Signal Analyzer app enables you to preprocess and edit signals and prepare them for further analysis. In the **Preprocess** mode, you can:

- Filter, denoise, detrend, smooth, and resample signals
- Extract, crop, trim, clip, or split signals based on drawn regions of interest or cursor locations
- Create and apply custom preprocessing functions

Signal Analyzer and Signal Labeler Apps: View frequency axes in log scale and toggle decibel display

Starting this release, the Signal Analyzer and Signal Labeler apps support logarithmic frequency axes in the spectrum and spectrogram views. You can also display the power spectrum in decibels or linear scale.

Spectrogram Computation: Compare available functions

Spectrogram Computation with Signal Processing Toolbox presents and compares different ways of computing short-time Fourier transforms and spectrograms of nonstationary signals using the `spectrogram`, `stft`, and `pspectrum` functions.

AI Workflows: Discover datastores, functions, and other resources for AI tasks

Manage Data Sets for Machine Learning and Deep Learning Workflows presents available datastores, functions, and other resources you can use for different AI tasks. Learn about:

- Data organization for signal classification, sequence-to-sequence classification, and regression tasks
- Data preprocessing and feature extraction
- Data set resources

Deep Learning Examples: Recover signals, monitor human health, and perform signal source separation

This release introduces examples that combine signal processing techniques and deep learning networks:

- Signal Recovery with Differentiable Scalograms and Spectrograms uses differentiable time-frequency transforms to recover a time-domain signal without the need for phase information or transform inversion.
- Signal Source Separation Using W-Net Architecture uses a deep learning network to discern fetal and maternal electrocardiogram (ECG) signals present in noninvasive abdominal measurements taken on pregnant patients.
- Human Health Monitoring Using Continuous Wave Radar and Deep Learning uses a deep learning network to reconstruct electrocardiograms from continuous-wave radar signals.

C/C++ Code Generation Support: Code generation for signal generation and preprocessing, time-frequency analysis, and vibration analysis

These Signal Processing Toolbox functions now support C/C++ code generation:

- Smoothing and Denoising — `hampel` and `medfilt1`
- Waveform Generation — `uencode`

These Signal Processing Toolbox functions now support timetables for C/C++ code generation:

- Time-Frequency Analysis — `hht`, `stft`, and `vmd`
- Vibration Analysis — `envspectrum`, `rainflow`, and `tsa`

The Generate Optimized Code on Raspberry Pi Target example shows how to generate optimized C++ code for the `resample` function that can be deployed on a Raspberry Pi® target (ARM®-based device).

You must have MATLAB Coder to generate standalone C and C++ code for the supported functions. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

GPU support for spectral and time-frequency analysis

These Signal Processing Toolbox functions now support `gpuArray` objects:

- Spectral Analysis and Measurements — `meanfreq` and `poctave`
- Time-Frequency Analysis — `instbw` and `instfreq`

The Classify ECG Signals Using Long Short-Term Memory Networks with GPU Acceleration example accelerates feature extraction and network training with a GPU.

You must have Parallel Computing Toolbox to use `gpuArray` objects with the supported functions. For more details, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox). To see which GPUs are supported, see GPU Computing Requirements (Parallel Computing Toolbox).

Functionality being removed or changed

alignsignals syntax changes

Still runs

The `alignsignals` function syntax has changed.

Functionality	Result	Use Instead
`alignsignals(x,y, ... maxlag)`	Runs. This syntax will be removed in a future release.	`alignsignals(x,y, ... Method="xcorr", ... MaxLag=maxlag)`
`alignsignals(x,y, ... maxlag,"truncate")`	Runs. This syntax will be removed in a future release.	`alignsignals(x,y, ... Method="xcorr", ... MaxLag=maxlag, ... Truncate=true)`

stftLayer: OutputMode property will be removed in a future release

Still runs

The `OutputMode` property of `stftLayer` will be removed in a future release. Update your code and networks to make them compatible with `stftLayer` output in `"SCBT"` format. For more information, see Layer Output Format.

R2022a

Version: 9.0

New Features

Bug Fixes

Compatibility Considerations

Signal Analyzer App: Calculate signal statistics

This release introduces **Measurements** in the Signal Analyzer app. You can calculate statistics such as minimum, maximum, median, mean, and peak-to-peak values for a full signal or a region of interest (ROI).

Signal Analyzer App: Interactively edit signals

The Signal Analyzer app has a new **Edit Signals** mode for interactive signal preprocessing. You can use clip and trim actions to remove unwanted signal data, or use a one-click crop action to include only data from a selected ROI.

Signal Analyzer App: Analyze signals with nonfinite data

This release enhances the Signal Analyzer app to support signals containing `NaN` and `Inf` values.

Signal Labeler App: Automatically detect speech regions in audio signals and label spoken words

The Signal Labeler app now provides automatic labeling of detected regions of speech and speech-to-text transcription. You must have Audio Toolbox to use this functionality.

- To automate the detection of speech content, the app uses the `detectSpeech` (Audio Toolbox) function.
- To perform speech-to-text transcription, the app uses the `speech2text` function available on File Exchange. The function interfaces with third-party speech-to-text APIs including:
  - Google® Speech API
  - IBM® Watson Speech API
  - Microsoft® Azure Speech API

The `speech2text` entry on File Exchange includes a tutorial to help get you started.

Signal Labeler App: Extract features from signals

This release introduces feature extraction in the Signal Labeler app. You can extract time and spectral features from signals and save generated features as labels. You can also export features to the MATLAB workspace or the Classification Learner (Statistics and Machine Learning Toolbox) app for machine learning and deep learning workflows.

Signal Label Definitions: Define label definitions for generated features

The `signalLabelDefinition` object now has label types to define signal features generated in the Signal Labeler app or by using a feature extractor object and `setLabelValue`. You can define full signal features using attribute feature labels and frame-based features using ROI feature labels.

Labeled Signal Sets: Generate feature data and get label indices from the command line

The new `createFeatureData` function creates a table or matrix of attribute and ROI feature labels in a `labeledSignalSet` object. Use `getLabelNames`, `getLabelDefinitions`, and the new `getLabelIndices` function to retrieve feature label definitions in a labeled signal set.

Labeled Signal Sets: Access source datastore on different machines

Starting this release, you can use the `getAlternateFileSystemRoots` and `setAlternateFileSystemRoots` functions to view and change the path to source data in a `labeledSignalSet`.

Deep Learning Examples: Use adversarial learning denoiser model and perform sequence-to-sequence classification

This release introduces examples that use signal processing techniques and deep learning networks:

- Denoise Signals with Adversarial Learning Denoiser Model uses a denoiser object with an adversarial learning architecture to remove noise from noisy electrocardiogram and electroencephalogram signals.
- Classify Arm Motions Using EMG Signals and Deep Learning performs sequence-to-sequence classification of forearm motions using labeled electromyography signals and a long short-term memory (LSTM) network.

C/C++ Code Generation Support: Code generation for filtering, feature extraction, preprocessing, and signal measurements

These Signal Processing Toolbox functions now support C/C++ code generation:

- Digital and Analog Filters — `filtic`, `grpdelay`, and `isstable`
- Preprocessing and Feature Extraction — `fillgaps` and `findchangepts`
- Pulse and Transition Metrics — `dutycycle`, `midcross`, `overshoot`, `pulseperiod`, `pulsesep`, `pulsewidth`, `settlingtime`, `slewrate`, and `undershoot`

You must have MATLAB Coder to generate standalone C and C++ code for the supported functions. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

GPU support for feature extraction, spectral analysis, spectral measurements, and transforms

These Signal Processing Toolbox functions now support `gpuArray` objects:

- Feature Extraction — `findpeaks` and `zerocrossrate`
- Spectral Analysis — `db2pow` and `pow2db`
- Spectral Measurements — `pentropy` and `pkurtosis`
- Transforms — `hilbert`

You must have Parallel Computing Toolbox to use `gpuArray` objects with the supported functions. For more details, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox). To see which GPUs are supported, see GPU Support by Release (Parallel Computing Toolbox).

Functionality being removed or changed

Filter Visualization Tool (fvtool) has changed

Behavior change

The behavior of the Filter Visualization Tool has changed. In previous releases, fvtool had toolbars containing plot editing and analysis controls. Starting this release, fvtool plot editing, analysis, and integration with Filter Designer controls are provided on the toolstrip. To edit a plot, first use **Send to Figure** to plot to a figure window and then use the plot editing toolbar.

sptool has been removed

Errors

The `sptool` function has been removed.

- For signal and spectral analysis, use the Signal Analyzer app.
- For filter design, use the Filter Designer app.

R2021b

Version: 8.7

New Features

Bug Fixes

Compatibility Considerations

Design Filter Live Editor Task: Design digital filter interactively

This release introduces the Design Filter Live Editor task, which lets you design and analyze a digital filter interactively. The task automatically generates MATLAB code for your live script.

Signal Labeler App: Inspect distribution of label counts on heatmap

This release introduces a member count chart in the Signal Labeler Dashboard. You can inspect the distribution of label counts simultaneously across members and across region durations or point locations.

Signal Labeler App: Show outliers in Dashboard

Starting this release, the box plots for region duration or point location distributions show outliers in the Signal Labeler Dashboard.

Signal Labeler App: Listen to audio signals while annotating them interactively

This release enhances the Signal Labeler app with new audio playback controls that play signals and regions of interest. You must have Audio Toolbox to use this functionality.

Signal Analyzer App: Denoise signals interactively using wavelet methods

The Signal Analyzer app now enables wavelet denoising. Use this feature to denoise signals in the app using wavelet methods. You must have Wavelet Toolbox to use this functionality.

Feature Extraction: Extract time-domain and frequency-domain features of signals

This release introduces the `signalTimeFeatureExtractor` and `signalFrequencyFeatureExtractor` objects. These feature extractor objects generate feature tables or matrices that you can use to train a machine learning model or a deep learning network.

- Use the `signalTimeFeatureExtractor` object to extract time-domain features of signals. Extract mean, RMS level, standard deviation, shape factor, signal-to-noise ratio, total harmonic distortion, signal to noise and distortion ratio, peak value, crest factor, clearance factor, and impulse factor.
- Use the `signalFrequencyFeatureExtractor` object to extract frequency-domain features of signals. Extract mean and median frequency, band power, occupied and power bandwidth, power spectral density, and spectral peak amplitude and location.

You can use the generated MATLAB code to generate C/C++ code that you can use to extract features for your entire data set.

Feature Extraction: Compute zero-crossing rates of signals

The new `zerocrossrate` function returns the rate, count, and location of zero crossings for a signal.

Deep Learning: Short-time Fourier transform layer

This release introduces a learnable signal processing layer that computes the short-time Fourier transform within a deep learning network. The `stftLayer` object is available from the command line and from the Deep Network Designer (Deep Learning Toolbox) app. You can use this layer with both `DAGNetwork` (Deep Learning Toolbox) and `dlnetwork` (Deep Learning Toolbox) architectures. You must have Deep Learning Toolbox to use the `stftLayer` object. For a list of available layers, see List of Deep Learning Layers (Deep Learning Toolbox).

Deep Learning Examples: Use short-time Fourier transform layer and perform deep learning regression

This release introduces examples that use signal processing techniques and deep learning networks:

- Denoise EEG Signals Using Deep Learning Regression uses deep learning regression to remove electro-oculogram (EOG) noise from electroencephalogram (EEG) signals.
- Hand Gesture Classification Using Radar Signals and Deep Learning classifies ultra-wideband impulse radar signal data using a convolutional neural network.
- Human Activity Recognition Using Mobile Phone Data classifies human activity using features extracted from smartphone sensor signals.
- Anomaly Detection Using Autoencoder and Wavelets uses wavelet-extracted features to detect arc signals in a DC system.
- Learn Pre-Emphasis Filter Using Deep Learning shows how to use a convolutional deep network and a short-time Fourier transform learnable layer to learn a pre-emphasis filter for speech recognition.

Signal Datastores: Specify FileSet objects as data locations

Starting this release, you can use `FileSet` objects to provide file locations for `signalDatastore` objects. `FileSet` objects provide increased performance compared to file paths or `DsFileSet` objects. For more information, see `matlab.io.datastore.FileSet`.

poctave Function: Improved octave smoothing and filter design

This release improves the robustness of the filter designs in `poctave`. The function also has an improved algorithm for octave smoothing that attenuates power levels at band edges.

C/C++ Code Generation Support: Code generation for filtering, spectral analysis, and vibration analysis

These Signal Processing Toolbox functions now support C/C++ code generation:

- Digital Filtering — `ss2sos`, `tf2sos`, and `zp2sos`
- Digital Filter Analysis — `filternorm` and `phasez`

- Spectral Analysis — `db` and `poctave`
- Vibration Analysis — `rpmtrack`

You must have MATLAB Coder to generate standalone C and C++ code for the supported functions. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

GPU support for digital filtering, feature extraction, signal processing, transforms, and waveform generation

These Signal Processing Toolbox functions now support `gpuArray` objects:

- Digital Filtering — `filtfilt`
- Feature Extraction — `pspectrum`
- Multirate Signal Processing — `downsample`, `resample`, `upfirdn`, and `upsample`
- Transforms — `xspectrogram`
- Waveform Generation — `shiftdata` and `unshiftdata`

You must have Parallel Computing Toolbox to use `gpuArray` objects with the supported functions. For more details, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox). To see which GPUs are supported, see GPU Support by Release (Parallel Computing Toolbox).

Run functions in a thread-based environment

You can now run these Signal Processing Toolbox functions in the background using MATLAB `backgroundPool`:

- Descriptive Statistics — `meanfreq`, `medfreq`, `peak2peak`, `peak2rms`, `rms`, and `rssq`
- Pulse and Transition Metrics — `dutycycle`
- Spectral Estimation — `pwelch`
- Spectral Measurements — `bandpower`
- Time-Frequency Analysis — `fsst`, `istft`, `spectrogram`, `stft`, and `xspectrogram`
- Transforms — `czt` and `dct`
- Waveform Generation — `chirp` and `sinc`

For more information, see Run MATLAB Functions in Thread-Based Environment.

MATLAB Online support for Signal Analyzer and Signal Labeler

Starting this release, the Signal Analyzer and Signal Labeler apps are supported in MATLAB Online™.

⚠Functionality being removed or changed

designfilt no longer assists in correcting incomplete or erroneous function calls

Behavior change

Starting this release, the `designfilt` function no longer assists in correcting calls to `designfilt` in a script or function. If you specify an incomplete or inconsistent set of input arguments in calls to

`designfilt` within a script or function, the function issues an error with a link to open the Filter Design Assistant. Users can use the assistant to generate a filter and display the corresponding code on the command line. The generated filter is saved to the MATLAB workspace. For more information, see Filter Design Assistant. In previous releases, the assistant automatically corrected the script or function.

sptool will be removed
Warns

The `sptool` function will be removed in a future release.

- For signal and spectral analysis, use the Signal Analyzer app.
- For filter design, use the Filter Designer app.

R2021a

Version: 8.6

New Features

Bug Fixes

Signal Labeler App: Analyze labeling progress and distribution of labels in your labeled signal set

This release introduces the Dashboard in the Signal Labeler app that lets users track their labeling progress. You can analyze distributions of each label to uncover biases in your data for machine learning and artificial intelligence applications.

Signal Labeler App: Label complex-valued signals

The Signal Labeler app now supports complex-valued signals.

Signal Labeler App: View spectra and spectrograms in Fast Navigation mode

The Signal Labeler app now shows signal spectra and spectrograms in **Fast Navigation** mode to aid in the labeling process. You can now draw region-of-interest or point labels on the spectrogram or the time plot in this mode.

Signal Labeler App: Speed up labeling and inspection by panning and zooming through signals and labels

The Signal Labeler app now lets users navigate through long signals using predefined panning windows. It also lets you quickly inspect labeled regions or points of interest using new zoom-to-label functionality.

Signal Labeler App: Import and label audio files

Starting this release, you can import and label audio files in the Signal Labeler app. You must have Audio Toolbox to use this functionality.

EDF File Analyzer App: View EDF or EDF+ files

The new EDF File Analyzer app enables users to view header and data records stored in EDF or EDF + files. Users can visualize annotated signals to aid analysis.

European Data Format Files: Create and modify EDF or EDF+ files

This release introduces the `edfwrite` object that lets users create new or modify existing EDF or EDF+ files with signal data, header information, and annotations.

Labeling Convenience Functions: Split data sets by label value and assign attributes based on file location

This release introduces three functions that aid and streamline labeling processes for deep learning workflows with large data sets.

- `countlabels` counts the number of unique label values in an array, a table, or a datastore.

- `splitlabels` finds indices to split labeled datasets into multiple datasets with equal specified label proportions.
- `folders2labels` creates a list of labels that can be associated with data files based on the names of the directories that house them.

Signal Labeling: Count label values and create datastores from labeledSignalSet objects

This release enhances `labeledSignalSet` objects by adding two functions that aid in machine learning and deep learning training workflows.

- `countLabelValues` counts the number of members that have a given attribute or the number of members that have at least one instance of a given ROI or point label.
- `createDatastores` takes data from a `labeledSignalSet` object and creates two datastores: a `signalDatastore` containing the signal data and an `arrayDatastore` containing the labels.

dlstft Function: Compute short-time Fourier transforms of deep learning arrays

This release introduces the `dlstft` function, which computes short-time Fourier transforms of `dlarray` (Deep Learning Toolbox) objects. The function outputs the transforms as `dlarray` objects that enable automatic differentiation and can be used in custom training loops.

Signal Datastores: Write data from datastore to files using writeall

This release enhances `signalDatastore` objects by introducing the `writeall` function. Use `writeall` to write data from a datastore to files on disk.

Time-Frequency Analysis: Compute instantaneous signal bandwidth

The new `instbw` function calculates the instantaneous bandwidth of a signal as the second conditional spectral moment of a time-frequency distribution. The function also accepts time-frequency distributions as input. Use `instbw` for signal analysis or machine learning applications.

poctave Function: Compute and visualize octave spectrograms

Starting this release, the `poctave` function computes the octave spectrogram of a nonstationary signal or timetable.

Signal Resampling: Change sample rates of MATLAB timetables or resample them to uniform grids

The `resample` function now accepts MATLAB timetables as input.

Deep Learning Examples: Classify signals using neural networks and a custom log spectrogram layer

This release introduces two examples that use signal processing techniques and deep learning networks:

- Spoken Digit Recognition with Custom Log Spectrogram Layer and Deep Learning uses a deep convolutional neural network and a custom log spectrogram layer to classify speech recordings.
- Labeling Radar Signals with Signal Labeler (Radar Toolbox) uses Signal Labeler to label time and frequency features of noisy pulse radar signals.

Signal Processing Onramp: Interactive introduction to practical signal processing methods

See the Signal Processing Onramp for an introduction to signal processing methods including preprocessing, filtering, and spectral analysis.

C/C++ Code Generation Support: Code generation for filtering, signal modeling, spectral analysis, and statistics

These Signal Processing Toolbox functions now support C/C++ code generation:

- Digital and Analog Filters — `freqz`, `impz`, `impzlength`, `lp2bp`, `lp2bs`, `lp2hp`, `lp2lp`, `sos2ss`, `sos2zp`, `ss2zp`, `stepz`, `tf2zp`, and `tf2zpk`
- Signal Modeling — `arburg`, `arcov`, `armcov`, `aryule`, and `prony`
- Spectral Analysis — `pburg`, `pcov`, `peig`, `pmcov`, `pmusic`, `pyulear`, `rooteig`, and `rootmusic`
- Time-Frequency Analysis — `instbw`
- Waveform Generation — `marcumq`

You must have MATLAB Coder to generate standalone C and C++ code for supported functions. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

GPU support for signal labeling, time-frequency analysis, transforms, digital filtering, and waveform generation

These Signal Processing Toolbox functions now support `gpuArray` objects:

- Signal Labeling — `extendsigroi`, `binmask2sigroi`, `mergesigroi`, `removesigroi`, `shortensigroi`, and `sigroi2binmask`
- Time-Frequency Analysis — `dlstft` and `stftmag2sig`
- Digital Filtering — `sosfilt`
- Transforms — `bitrevorder` and `digitrevorder`
- Waveform Generation — `buffer`

You must have Parallel Computing Toolbox to use `gpuArray` objects with supported functions. For more details, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox). To see which GPUs are supported, see GPU Support by Release (Parallel Computing Toolbox).