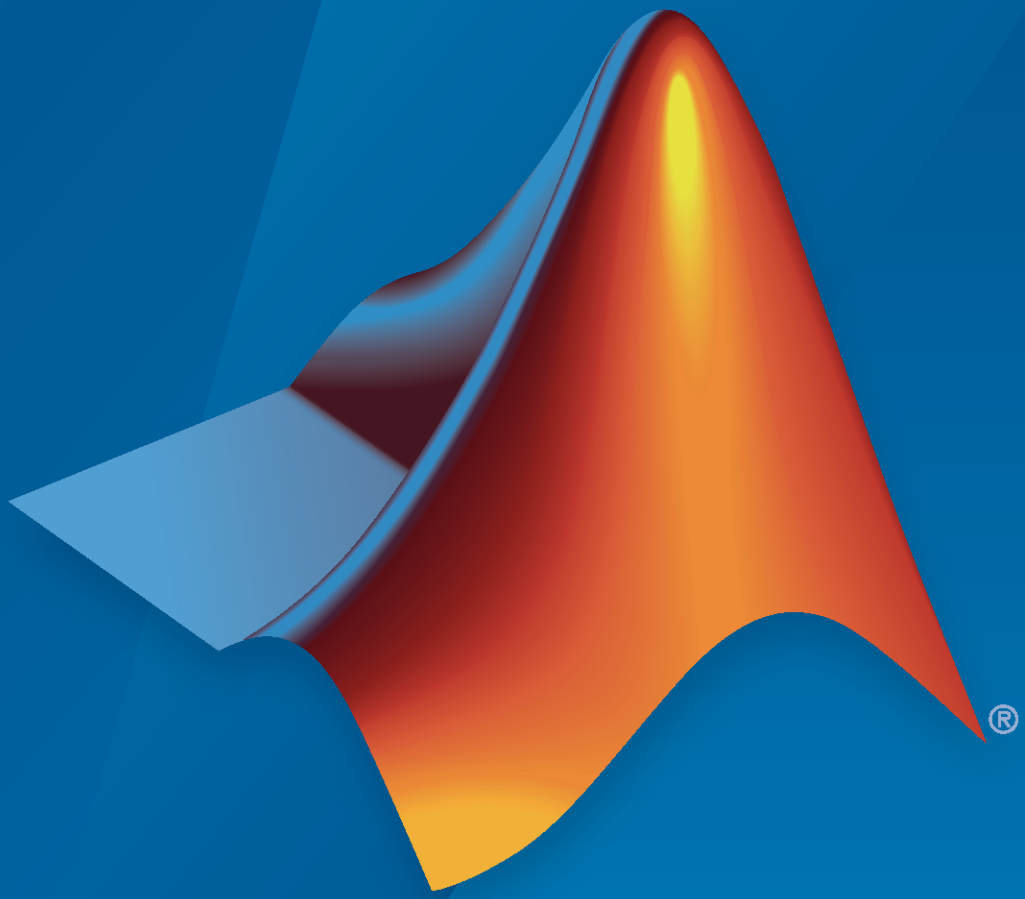
Signal Processing Toolbox™ Release Notes



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

R2024a

R2023b

R2023a

R2022b

R2022a

R2021b

R2021a

R2020b

R2020a

R2019b

R2019a

R2018b

R2024a

★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 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 also can 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” on page 1-5.

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 app. You can use this layer with the `dlnetwork` 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 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` 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

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

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

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

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

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).

R2020b

Signal Labeler App: Perform faster labeling

Starting this release, the Signal Labeler app lets users perform faster labeling by drawing regions and points of interest. The app also has a new fast navigation mode that enables users to navigate quickly through datasets and label members.

Signal Labeler App: View spectra and spectrograms

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

Signal Labeler App: Import data from files

The Signal Labeler app now lets users import data from files.

Signal Segmentation: Extract and convert signal regions of interest in preparation for deep learning

This release introduces the `signalMask` object and the `binmask2sigroi`, `sigroi2binmask`, `extendsigroi`, `extractsigroi`, `mergesigroi`, `removesigroi`, and `shortensigroi` functions. With the new functionality, you can:

- Express region-of-interest (ROI) signal masks as tables of ROI limits, as categorical sequences, or as matrices of binary sequences, and convert between formats
- Manipulate masks: Extend, remove, or merge regions of interest
- Extract signal regions defined by masks to prepare data for training machine learning or deep learning models
- Plot signals with color-coded regions

European Data Format Files: Read EDF and EDF+ files and obtain information about them

The European Data Format (EDF) is a widely used storage and file exchange format for biological and medical signals. This release introduces the `edfread` and `edfinfo` functions.

- `edfread` enables users to read data stored in EDF and EDF+ files into the MATLAB workspace.
- `edfinfo` returns information about the header and contents of an EDF file.

Short-Time Fourier Transform: Reconstruct signals from their STFT magnitudes and compute one-sided estimates

This release introduces the `stftmag2sig` function, which allows users to reconstruct a signal time-domain waveform starting from the magnitude of its short-time Fourier transform.

Starting this release, the `stft` and `istft` functions can compute one-sided forward and inverse short-time Fourier transforms of real-valued signals.

Signal Labeling: Point to signal collections in the workspace or in files using signalDatastore objects

Starting this release, the `labeledSignalSet` object accepts input specified by `signalDatastore` objects.

⚠Signal Resampling: Change sample rates of N-D arrays or resample them to uniform grids

This release enhances the `resample` function to accept *N*-D arrays as input. You can use the function to resample multidimensional arrays or to interpolate nonuniformly sampled *N*-D arrays to uniform grids.

⚠Compatibility Considerations

In previous releases, `resample` accepted arrays with more than two dimensions but returned two-dimensional matrices. As a result of this behavior, the resampled output was correct in some cases but not all. Starting this release, the function has a `dim` argument that allows users to specify the dimension along which to operate. If `dim` is not specified, `resample` operates along the first array dimension with size greater than 1. The output has the same number of dimensions as the input.

chirp Function: Generate complex-valued swept-frequency cosine signals

The `chirp` function now generates complex-valued chirps. You can now specify negative initial or final chirp frequencies.

pmtm Function: Perform spectral analysis using sine tapers

Starting this release, the `pmtm` function lets you compute multitaper power spectral density estimates of signals using either Slepian tapers or sine tapers.

Deep Learning Examples: Use generative adversarial network and generate Raspberry Pi code

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

- Generate Synthetic Signals Using Conditional Generative Adversarial Network uses a conditional generative adversarial network to produce synthetic signals.
- Deploy Signal Segmentation Deep Network on Raspberry Pi generates a MEX function and a standalone executable that perform waveform segmentation on a Raspberry Pi.

C/C++ Code Generation Support: Generate code for feature extraction, signal measurements, and vibration analysis

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

- **Feature Extraction** — `cusum`, `edr`, `findsignal`, and `rssq`
- **Spectral Measurements** — `instfreq`, `sinad`, `pentropy`, `pkurtosis`, `snr`, `thd`, and `toi`
- **Transition Measurements** — `falltime`, `risetime`, and `statelevels`
- **Vibration Analysis** — `orderspectrum`, `rpmfreqmap`, `rpmordermap`, and `tachorpm`

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 acceleration for spectral analysis and time-frequency analysis functions

The `cpsd`, `fsst`, `goertzel`, `istft`, `mscohere`, `periodogram`, and `pwelch` functions now support `gpuArray` objects. The `czt` function now supports 3-D input for both numeric arrays and `gpuArray` objects. Starting this release, you also can use the `spectrogram` function on a GPU to compute short-time Fourier transforms at nonuniformly spaced frequencies.

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).

GPU code generation support for zero-phased filtering and Fourier synchrosqueezed transform functions

The `filtfilt`, `fsst`, and `ifsst` functions now support code generation for graphical processing units (GPUs).

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

tall Array Support: Operate on tall arrays with the pwelch function

The `pwelch` function now accepts tall arrays as input.

R2020a

Signal Labeler App: Perform interactive or automated signal labeling

The Signal Labeler app can now be opened from the **Apps** tab on the MATLAB Toolstrip or from the Command Window. The app enables you to import data from the MATLAB workspace and export labeled data to the workspace or to a file. The app's label viewer is now enhanced to let you edit labels interactively.

Signal Labeler now enables you to autolabel plotted signals and inspect labeling results before committing. When in autolabeling mode, you can run any function that labels attributes, regions of interest, or points of interest. You can then inspect the labeling, modify the autolabeling function, edit labels, and save the labeling when it is satisfactory. For easier label inspection, the autolabeling mode displays only the labels generated by the most recently called function.

Signal Datastores: Work with signal collections that exist in the workspace or in files

This release introduces datastores that make it possible to read, preprocess, and transform signal collections that exist in the MATLAB workspace or in files. `signalDatastore` objects enable users to work with large data collections for easier processing in machine learning and artificial intelligence applications. For an example of `signalDatastore` usage, see Waveform Segmentation Using Deep Learning.

Time-Frequency Analysis: Use variational mode decomposition to extract intrinsic modes

This release introduces the `vmd` function, which performs variational mode decomposition. VMD decomposes a real signal into a number of narrowband mode functions whose envelopes and instantaneous frequencies vary much more slowly than their central frequencies. The algorithm determines all mode waveforms and central frequencies simultaneously and thus distributes errors among them in a balanced way. Variational mode decomposition is suitable for the study of nonstationary or nonlinear signals.

Deep Learning Examples: Use time-frequency analysis and neural networks for classification and labeling

This release introduces three examples that employ signal processing techniques and deep learning:

- Iterative Approach for Creating Labeled Signal Sets with Reduced Human Effort uses a train-as-you-label iterative method for deep learning classifier training.
- Pedestrian and Bicyclist Classification Using Deep Learning (Phased Array System Toolbox) uses a deep learning network to classify pedestrians and bicyclists based on their micro-Doppler characteristics.
- Modulation Classification with Deep Learning (Communications Toolbox) performs modulation classification using a convolutional neural network.

tall Arrays: Operate on tall arrays with the spectrogram and stft functions

The `spectrogram` and `stft` functions now support tall arrays as inputs. For more information on usage and limitations, see the Extended Capabilities section at the bottom of each reference page.

GPU code generation support for fftfilt, stft, and istft functions

The `fftfilt`, `stft`, and `istft` functions now support code generation for graphical processing units (GPUs). You must have MATLAB Coder and GPU Coder to generate CUDA code.

GPU acceleration for spectrogram, czt, stft, and wvd functions

The `spectrogram`, `czt`, `stft`, and `wvd` functions now support `gpuArray` objects. 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).

C/C++ Code Generation Support: Generate code for time-frequency analysis, feature extraction, spectral analysis, multirate signal processing, and filter design

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

- **Time-Frequency Analysis** — `hht`, `kurtogram`, `pspectrum`, `spectrogram`, and `xspectrogram`
- **Feature Extraction** — `dtw`
- **Spectral Analysis** — `tfestimate`
- **Spectral Measurements** — `bandpower`, `enbw`, `meanfreq`, `medfreq`, `obw`, `powerbw`, and `sfdr`
- **Multirate Signal Processing** — `resample` and `upfirdn`
- **Filter Design** — `bilinear`, `butter`, `lp2bp`, `lp2bs`, `lp2hp`, `lp2lp`, and `zp2ss`
- **Transforms** — `cceps`, `icceps`, `fwht`, and `ifwht`
- **Linear Predictive Coding** — `levinson`

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.

⚠ Functionality being removed or changed

Label button removed from Signal Analyzer

Behavior change

Signal Analyzer no longer opens Signal Labeler, which is now available as an app. If you want to label signals, open Signal Labeler from the MATLAB Toolstrip or the Command Window.

R2019b

Version: 8.3

New Features

Bug Fixes

Signal Labeling: Perform automated labeling using user-defined functions

The Signal Labeler now enables you to label your data using custom automated labeling functions. You can label signal attributes, regions of interest, or points of interest.

Signal Labeling: Automatically find and label signal peaks and valleys

The Signal Labeler now enables you to find and label local maxima and minima of signals. Signal Labeler uses the MATLAB functions `islocalmax` and `islocalmin` to search for the peaks and valleys.

Signal Analyzer App: Analyze complex signals

The Signal Analyzer app now accepts complex data. You can view real (inphase) and imaginary (quadrature) parts and estimate two-sided spectra and spectrograms of complex signals.

Tall Array Support: Compute spectrograms of signals too large to fit in memory

The `stft` and `spectrogram` functions now accept tall arrays as input.

stft and istft Functions: Compute and invert short-time Fourier transforms of multichannel signals

The `stft` and `istft` functions now accept multichannel signals as input.

Time-Frequency Gallery: Examine features and limitations of time-frequency analysis methods

Use the new Time-Frequency Gallery to examine the features and limitations of the different time-frequency analysis methods provided by Signal Processing Toolbox and Wavelet Toolbox. The Gallery presents the potential application of specific time-frequency methods to the analysis of seismic data, music and speech signals, biomedical data, and vibration measurements.

C/C++ Code Generation Support: Generate code for time-frequency analysis, spectral analysis of nonuniformly sampled signals, and digital filtering

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

- **Time-Frequency Analysis** — `fsst`, `ifsst`, `tfridge`, `wvd`, and `xwvd`
- **Spectral Analysis of Nonuniformly Sampled Signals** — `plomb`
- **Transforms** — `dftmtx` and `rceps`
- **Digital Filtering** — `eqtflength`, `fftfilt`, and `tf2ss`
- **Waveform Generation** — `chirp`, `diric`, `gmonopuls`, and `sawtooth`

- **Spectral Windows** — `chebwin`

You must have MATLAB Coder to generate standalone C and C++ code for supported functions.

R2019a

Version: 8.2

New Features

Bug Fixes

Signal Labeling: Label signals interactively and visualize labeled signals

The Signal Labeler enables you to label signals interactively and visualize labeled signals. You can annotate signals and prepare signal datasets for machine learning and deep learning classification and regression tasks. You can access the Signal Labeler from the Signal Analyzer app.

Time-Frequency Analysis: Compute short-time Fourier transforms and inverse short-time Fourier transforms

This release introduces a set of functions that provide enhanced support for the short-time Fourier transform. The short-time Fourier transform is the most widely used tool for time-frequency analysis. The transform has applications in all fields that involve nonstationary signals.

- The `stft` function computes the short-time Fourier transform of a signal.
- The `istft` function reconstructs a signal from its short-time Fourier transform.
- The `iscola` function checks whether a window-overlap combination satisfies a necessary condition for perfect reconstruction.

All three functions support C/C++ code generation.

Signal Analyzer App: Remove trends from signals and estimate their envelopes

The Signal Analyzer app now enables you to compute the upper and lower envelopes of a waveform. You can find the envelopes using an FFT-based analytic function, an FIR Hilbert filter, the function peaks, or the signal RMS values.

This release also introduces functionality to remove trends from signals. You can remove constant trends, linear trends, and piecewise linear trends.

Signal Analyzer App: Enhanced management of multichannel signals

Starting this release, Signal Analyzer displays an expandable hierarchy of any multichannel signal that you import. The app maintains the hierarchy as you work with the different channels, enabling better and easier signal management and export.

C/C++ Code Generation Support: Generate code for filter design, spectral analysis, and spectral windowing

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

- **Filter Design and Filtering:**

  `buttap`, `filtfilt`, `filtord`, `fir1`, `firls`, `kaiserord`, and `sos2tf`
- **Spectral Analysis:**

  `cpsd`, `czt`, `goertzel`, `mscohere`, `periodogram`, and `pwelch`
- **Spectral Windows:**

`barthannwin`, `bartlett`, `blackman`, `blackmanharris`, `bohmanwin`, `flattopwin`, `gausswin`, `nuttallwin`, `parzenwin`, `rectwin`, `taylorwin`, `triang`, and `tukeywin` now accept variable input.

- **Waveform Generation:**

  `gauspuls`, `pulstran`, `rectpuls`, `square`, and `tripuls`

- **Linear Predictive Coding:**

  `lsf2poly`, `poly2ac`, `poly2lsf`, `poly2rc`, `rc2ac`, `rc2poly`, and `rlevinson`

You must have MATLAB Coder to generate standalone C and C++ code for supported functions.

R2018b

Version: 8.1

New Features

Bug Fixes

Signal Analyzer App: Preprocess signals using user-defined functions

The Signal Analyzer app now enables you to preprocess your data within the app itself, using custom preprocessing functions.

Signal Analyzer App: Change sample rates of signals and convert nonuniformly sampled signals to uniformly sampled signals

The Signal Analyzer app now enables you to resample signals. You can interpolate nonuniformly sampled signals onto uniform grids. You can also change the sample rate of uniformly sampled signals. Generate MATLAB functions to resample any number of signals according to your specifications.

Time-Frequency Analysis: Analyze signals using the Wigner-Ville distribution

This release adds support for the Wigner-Ville distribution, which provides a high-resolution time-frequency representation of a signal. The distribution has applications in signal visualization, detection, and estimation.

- `wvd` computes the Wigner-Ville distribution of a signal. The function also computes the smoothed pseudo Wigner-Ville distribution, which uses independent windows to smooth in time and frequency.
- `xwvd` computes the cross Wigner-Ville distribution of two signals. The function also computes the cross smoothed pseudo Wigner-Ville distribution, which uses independent windows to smooth in time and frequency.

Deep Learning Example: Identify morphological features of signals using recurrent neural networks

This release introduces an example, Waveform Segmentation using Deep Learning, that shows one way to combine signal processing and long short-term memory (LSTM) networks for the analysis of signals.

Signal Labeling: Define labels and create sets of labeled signals

This release introduces functionality to define labels for signals and to create sets of labeled signals. You can store signal values and annotations in a form that keeps all data together.

- `signalLabelDefinition` enables users to create signal label definitions. The definitions can be for attributes, regions, or points of interest.
- `labeledSignalSet` enables users to group signals, label definitions, and label values that can be used in learning algorithms.