

Signal Processing Toolbox™ Release Notes



MATLAB®



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

© COPYRIGHT 2004–2022 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

R2022b

Labeling Convenience Function: Generate labels from file names	1-2
alignsignals Function: Align signals based on rising edge or peak locations	1-2
Signal Analyzer App: Interactively preprocess signals with various functions and actions	1-2
Signal Analyzer and Signal Labeler Apps: View frequency axes in log scale and toggle decibel display	1-2
Spectrogram Computation: Compare available functions	1-2
AI Workflows: Discover datastores, functions, and other resources for AI tasks	1-2
Deep Learning Examples: Recover signals, monitor human health, and perform signal source separation	1-3
C/C++ Code Generation Support: Code generation for signal generation and preprocessing, time-frequency analysis, and vibration analysis . .	1-3
GPU support for spectral and time-frequency analysis	1-3
Functionality being removed or changed	1-4
alignsignals syntax changes	1-4
stftLayer: OutputMode property will be removed in a future release	1-4

R2022a

Signal Analyzer App: Calculate signal statistics	2-2
Signal Analyzer App: Interactively edit signals	2-2
Signal Analyzer App: Analyze signals with nonfinite data	2-2
Signal Labeler App: Automatically detect speech regions in audio signals and label spoken words	2-2

Signal Labeler App: Extract features from signals	2-2
Signal Label Definitions: Define label definitions for generated features	2-2
Labeled Signal Sets: Generate feature data and get label indices from the command line	2-3
Labeled Signal Sets: Access source datastore on different machines	2-3
Deep Learning Examples: Use adversarial learning denoiser model and perform sequence-to-sequence classification	2-3
C/C++ Code Generation Support: Code generation for filtering, feature extraction, preprocessing, and signal measurements	2-3
GPU support for feature extraction, spectral analysis, spectral measurements, and transforms	2-3
Functionality being removed or changed	2-4
Filter Visualization Tool (fvtool) has changed	2-4
sptool has been removed	2-4

R2021b

Design Filter Live Editor Task: Design digital filter interactively	3-2
Signal Labeler App: Inspect distribution of label counts on heatmap ...	3-2
Signal Labeler App: Show outliers in Dashboard	3-2
Signal Labeler App: Listen to audio signals while annotating them interactively	3-2
Signal Analyzer App: Denoise signals interactively using wavelet methods	3-2
Feature Extraction: Extract time-domain and frequency-domain features of signals	3-2
Feature Extraction: Compute zero-crossing rates of signals	3-3
Deep Learning: Short-time Fourier transform layer	3-3
Deep Learning Examples: Use short-time Fourier transform layer and perform deep learning regression	3-3
Signal Datastores: Specify FileSet objects as data locations	3-3
poctave Function: Improved octave smoothing and filter design	3-3

C/C++ Code Generation Support: Code generation for filtering, spectral analysis, and vibration analysis	3-3
GPU support for digital filtering, feature extraction, signal processing, transforms, and waveform generation	3-4
Run functions in a thread-based environment	3-4
MATLAB Online support for Signal Analyzer and Signal Labeler	3-4
Functionality being removed or changed	3-4
designfilt no longer assists in correcting incomplete or erroneous function calls	3-4
sptool will be removed	3-5

R2021a

Signal Labeler App: Analyze labeling progress and distribution of labels in your labeled signal set	4-2
Signal Labeler App: Label complex-valued signals	4-2
Signal Labeler App: View spectra and spectrograms in Fast Navigation mode	4-2
Signal Labeler App: Speed up labeling and inspection by panning and zooming through signals and labels	4-2
Signal Labeler App: Import and label audio files	4-2
EDF File Analyzer App: View EDF or EDF+ files	4-2
European Data Format Files: Create and modify EDF or EDF+ files	4-2
Labeling Convenience Functions: Split data sets by label value and assign attributes based on file location	4-2
Signal Labeling: Count label values and create datastores from labeledSignalSet objects	4-3
dlstft Function: Compute short-time Fourier transforms of deep learning arrays	4-3
Signal Datastores: Write data from datastore to files using writeall	4-3
Time-Frequency Analysis: Compute instantaneous signal bandwidth	4-3
p octave Function: Compute and visualize octave spectrograms	4-3

Signal Resampling: Change sample rates of MATLAB timetables or resample them to uniform grids	4-3
Deep Learning Examples: Classify signals using neural networks and a custom log spectrogram layer	4-4
Signal Processing Onramp: Interactive introduction to practical signal processing methods	4-4
C/C++ Code Generation Support: Code generation for filtering, signal modeling, spectral analysis, and statistics	4-4
GPU support for signal labeling, time-frequency analysis, transforms, digital filtering, and waveform generation	4-4

R2020b

Signal Labeler App: Perform faster labeling	5-2
Signal Labeler App: View spectra and spectrograms	5-2
Signal Labeler App: Import data from files	5-2
Signal Segmentation: Extract and convert signal regions of interest in preparation for deep learning	5-2
European Data Format Files: Read EDF and EDF+ files and obtain information about them	5-2
Short-Time Fourier Transform: Reconstruct signals from their STFT magnitudes and compute one-sided estimates	5-2
Signal Labeling: Point to signal collections in the workspace or in files using signalDatastore objects	5-3
Signal Resampling: Change sample rates of N-D arrays or resample them to uniform grids	5-3
chirp Function: Generate complex-valued swept-frequency cosine signals	5-3
pmtm Function: Perform spectral analysis using sine tapers	5-3
Deep Learning Examples: Use generative adversarial network and generate Raspberry Pi code	5-3
C/C++ Code Generation Support: Generate code for feature extraction, signal measurements, and vibration analysis	5-3

GPU acceleration for spectral analysis and time-frequency analysis functions	5-4
GPU code generation support for zero-phased filtering and Fourier synchrosqueezed transform functions	5-4
tall Array Support: Operate on tall arrays with the pwelch function	5-4

R2020a

Signal Labeler App: Perform interactive or automated signal labeling ..	6-2
Signal Datastores: Work with signal collections that exist in the workspace or in files	6-2
Time-Frequency Analysis: Use variational mode decomposition to extract intrinsic modes	6-2
Deep Learning Examples: Use time-frequency analysis and neural networks for classification and labeling	6-2
tall Arrays: Operate on tall arrays with the spectrogram and stft functions	6-3
GPU code generation support for fftfilt, stft, and istft functions	6-3
GPU acceleration for spectrogram, czt, stft, and wvd functions	6-3
C/C++ Code Generation Support: Generate code for time-frequency analysis, feature extraction, spectral analysis, multirate signal processing, and filter design	6-3
Functionality being removed or changed	6-3
Label button removed from Signal Analyzer	6-3

R2019b

Signal Labeling: Perform automated labeling using user-defined functions	7-2
Signal Labeling: Automatically find and label signal peaks and valleys	7-2
Signal Analyzer App: Analyze complex signals	7-2

Tall Array Support: Compute spectrograms of signals too large to fit in memory	7-2
stft and istft Functions: Compute and invert short-time Fourier transforms of multichannel signals	7-2
Time-Frequency Gallery: Examine features and limitations of time-frequency analysis methods	7-2
C/C++ Code Generation Support: Generate code for time-frequency analysis, spectral analysis of nonuniformly sampled signals, and digital filtering	7-2

R2019a

Signal Labeling: Label signals interactively and visualize labeled signals	8-2
Time-Frequency Analysis: Compute short-time Fourier transforms and inverse short-time Fourier transforms	8-2
Signal Analyzer App: Remove trends from signals and estimate their envelopes	8-2
Signal Analyzer App: Enhanced management of multichannel signals ..	8-2
C/C++ Code Generation Support: Generate code for filter design, spectral analysis, and spectral windowing	8-2

R2018b

Signal Analyzer App: Preprocess signals using user-defined functions	9-2
Signal Analyzer App: Change sample rates of signals and convert nonuniformly sampled signals to uniformly sampled signals	9-2
Time-Frequency Analysis: Analyze signals using the Wigner-Ville distribution	9-2
Deep Learning Example: Identify morphological features of signals using recurrent neural networks	9-2
Signal Labeling: Define labels and create sets of labeled signals	9-2

Signal Analyzer App: Preprocess signals by smoothing and filtering . . .	10-2
Signal Analyzer App: Detect transients and perform time-frequency analysis using scalogram view	10-2
One-Step Filtering: Filter signals using lowpass, highpass, bandpass, and bandstop responses	10-2
Time-Frequency Analysis: Perform empirical mode decomposition, Hilbert-Huang transform, and instantaneous frequency estimation	10-2
Time-Frequency Analysis: Estimate kurtogram, spectral kurtosis, and spectral entropy	10-2
ooctave Function: Compute 1/N octave spectra and perform octave smoothing	10-3
Rotating Machinery: Estimate and track rotational speed from vibration signals	10-3
Deep Learning Example: Classify signals using long short-term memory networks	10-3
Functionality being removed or changed	10-3

Signal Analyzer App: Analyze sporadic signals with persistence spectrum and sharpen time-frequency estimates using reassignment	11-2
Signal Analyzer App: Extract and export signal regions of interest	11-2
Signal Analyzer App: Generate MATLAB scripts to automate analysis	11-2
pspectrum Function: Analyze power spectrum, spectrogram, and persistence spectrum of signals	11-2
Rotating Machinery: Remove noise coherently with time-synchronous averaging and analyze wear using envelope spectra	11-2
Modal Analysis: Use parametric methods for FRF and modal parameter estimation	11-2
Fatigue Analysis: Perform high-cycle rainflow counting	11-3

findchangepts Function: Find changepoints in spectrograms and other multivariate signals	11-3
Functionality being removed or changed	11-3

R2017a

Signal Analyzer App: Perform time-frequency analysis using spectrogram view	12-2
Signal Analyzer App: Analyze timestamped signals	12-2
Modal Analysis: Estimate frequency-response functions and modal parameters of mechanical systems	12-2
Cross-Spectrogram: Compare time-frequency content of nonstationary signals	12-2
MIMO Spectral Analysis: Estimate cross-spectral density and coherence for multi-input/multi-output systems	12-2
Transfer Function Estimation: Compute unbiased estimates for SISO and MIMO systems containing additive input noise	12-2
dct and idct Functions: Compute four standard types of discrete cosine transform	12-3
Code Generation Support: Generate code for an expanded set of Signal Processing Toolbox functions	12-3
findpeaks Function: Single-precision support and improved code generation functionality	12-3

R2016b

Signal Analyzer App: Perform time- and frequency-domain analysis of multiple time series	13-2
Similarity Matching: Find patterns in data using edit distance or dynamic time warping	13-2
Order Analysis: Track orders and extract waveforms to analyze rotational machinery	13-2
Fourier Synchrosqueezing Transform: Obtain sharp time-frequency estimates and extract signal modes	13-2

Distortion Measurement Functions: Measure aliased harmonics in undersampled signals	13-2
GPU acceleration: Enhance performance of dct, idct, and sinc functions	13-3
Filter Design and Analysis Tool renamed to Filter Designer	13-3
Window Design and Analysis Tool renamed to Window Designer	13-3
Functionality being removed or changed	13-3

R2016a

Gap Filling: Reconstruct missing samples using autoregressive modeling	14-2
Changepoint Detection: Find abrupt changes and statistical shifts in signals	14-2
Dynamic Time Warping: Stretch, align, and compare signals with different time scales	14-2
Reassigned Periodogram: Sharpen the frequency localization of spectral estimates	14-2
Signal Analyzer App: Visualize and compare multiple time series	14-2
datetime Support: Use datetime arrays in Signal Processing Toolbox functions	14-2
xcorr Function: Generate faster code for long input vectors	14-2
chebwin Function: Compute Dolph-Chebyshev windows faster	14-3

R2015b

Reassigned Spectrogram: Sharpen the time-frequency localization of spectral estimates	15-2
Hampel Filter and Improved Median Filtering: Detect outliers and remove them from data	15-2
Order Analysis: Analyze vibrations in rotational machinery with order and frequency maps	15-2

Envelope Detection: Extract analytic, peak, and RMS envelopes	15-2
Signal Alignment: Measure delay and align signals in time	15-2

R2015a

Frequency Measurements: Compute mean and median frequencies using spectral estimates	16-2
Bandwidth Measurements: Compute occupied bandwidth and bandwidth at specified power levels	16-2
resample function accepts signals with nonuniform sampling or missing data	16-2
pwelch function computes maximum-hold and minimum-hold spectra	16-2
spectrogram function computes two-sided, centered, power, and PSD spectra	16-2
Fast chirp Z-transform support for spectral analysis functions	16-2

R2014b

Spectral estimation of signals with nonuniform sampling or missing data	17-2
Multichannel support for spectral analysis functions	17-2
Peak finding with visualization and enhanced peak selection	17-2

R2014a

Simplified workflow for specification-based filter design	18-2
Visualization of harmonic distortion measurements	18-2
Changes in name-value pair arguments of measurement functions	18-2

R2013b

Distortion, intermodulation, and SNR measurement functions	19-2
Design raised cosine and Gaussian pulse-shaping filters	19-2
Functionality Being Removed or Changed	19-2

R2013a

Functions to measure equivalent noise bandwidth, band power, and spurious-free dynamic range	20-2
Function interface to compute power spectrum with confidence intervals and DC-centered spectra	20-2
Function interface for analysis and implementation of single-precision filters	20-2
Function interface for analysis of second-order section (biquad) filters	20-2
Code generation support for Signal Processing Toolbox functions	20-2
Functionality Being Removed or Changed	20-3

R2012b

Signal Browser in SPTool	21-2
GPU acceleration for xcorr, xcorr2, fftfilt, xcov, and cconv functions	21-3

R2012a

Measurements for Bilevel Pulse Waveforms	22-2
Signal Statistics	22-2

R2011b

Passband and Stopband Weights for Fixed-Order Unconstrained Partial Band Differentiator Filters	23-2
Numerator and Denominator Order Specifications Added to filterbuilder for Lowpass and Highpass Butterworth Designs	23-2
Conversion of Error and Warning Message Identifiers	23-2

R2011a

Enhancements to filtfilt	24-2
Symmetric Window Option for Blackman-Harris Windows	24-2
rectpuls Returns Double-Precision Vector	24-2
Code Generation from MATLAB and Fixed-Point MEX-File Generation	24-2

R2010b

Embedded MATLAB Support for Additional Signal Processing Toolbox Functions	25-2
---	------

R2010a

Single-Precision Support Added for dfilt Objects	26-2
Embedded MATLAB Support for Additional Signal Processing Toolbox Functions	26-2
Functions, Objects, Object Methods, and Object Properties Being Removed	26-2
Warning for Filter Designer in SPTool	26-3

R2009b

Embedded MATLAB Support Added to Signal Processing Toolbox Functions	27-2
Ability to Export Filter Coefficients Added to realizemdl	27-2

R2009a

New filter design approach using fdesign and filterbuilder	28-2
New dfilt method to specify filter coefficients at block ports	28-2

R2008b

New Walsh-Hadamard Transform functions	29-2
---	------

R2008a

New Marcum Q Function	30-2
Conversion Between Magnitude and dB Added	30-2
PMTM Function Enhanced with Ability to Keep or Drop Last Taper	30-2

R2007b

Confidence Interval Estimation Added	31-2
Spurious-Free Dynamic Range (SFDR) Measurement Added	31-2
Local Maxima/Peak Finder Added	31-2
Conversions Between Power and dB Added	31-2

R2007a

lsf2poly and latcfilter Multi-Channel Input Support Added	32-2
Circular Convolution (cconv) Function Added	32-2
Spectrum Objects Partial Frequency Range Input Support Added	32-2
cceps Factorize Algorithm Information Clarified	32-2
dfilt.statespace Now Supports realizemdl Method	32-2
ellip and ellipap Functions Enhanced	32-2

R2006b

Frequency Vector Input Added to Spectral Analysis Functions	33-2
FFT Length in Spectral Analyses Changed	33-2
sosfilt and dfilt filter Method Support Multidimensional Array Input	33-2
dfilt block Method Supports Target Subsystem Destination and Link Between Command Line and Model	33-2
gaussfir Algorithm Updated	33-2

R2006a

Taylor Window Function Added	34-2
SPTool Filter Designer Replaced by FDATool	34-2
sgolay Example Improved	34-2
zp2sos zeroflag Parameter Added	34-2
Help for Objects Changed	34-2

dfilt (Discrete-Time Filters) Delay Structure Added	35-2
WinTool/WVTool Normalize Magnitude Added	35-2
FDATool/FVTool Plot Displays Improved	35-2
FVTool Passband Zoom Added	35-2

FDATool and FVTool Changes	36-2
FDATool Spectral Rejection Masks Added	36-2
FDATool Generated C Header File Complex Filter Support	36-2
FDATool Tip of the Day Added	36-2
FDATool State Space Filters Support Removed	36-2
FDATool/FVTool New Analysis Parameters Magnitude Response Options	36-2
FVTool SOS Filter Coefficients Display Enhancement	36-2
FVTool Default Phase Units Changed	36-2
dfilt Changes	36-3
dfilt Coefficients Method Changed	36-3
dfilt Filter States Changed to Use States Property	36-3
Spectral Analysis Changes	36-3
spectrogram Function Replaces specgram	36-3
Spectral Analysis Functions Inputs Changed	36-3
PSD Objects and Function Output Plots Changed	36-3
Other Changes	36-4
gaussfir Function Replaces firgauss	36-4
firpm and cfirpm Inputs Changed	36-4
New Demos	36-4
Filter Wizard Product Dependency Removed	36-4

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
<code>alignsignals(x,y,maxlag)</code>	Runs. This syntax will be removed in a future release.	<code>alignsignals(x,y,Method="xcorr",MaxLag=maxlag)</code>
<code>alignsignals(x,y,maxlag,"t")</code>	Runs. This syntax will be removed in a future release.	<code>alignsignals(x,y,Method="xcorr",MaxLag=maxlag)</code>

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`.

p octave Function: Improved octave smoothing and filter design

This release improves the robustness of the filter designs in `p octave`. 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.

p octave Function: Compute and visualize octave spectrograms

Starting this release, the `p octave` 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

Version: 8.5

New Features

Bug Fixes

Compatibility Considerations

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

Version: 8.4

New Features

Bug Fixes

Compatibility Considerations

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 `sfd`
- **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 `stfft` function computes the short-time Fourier transform of a signal.
- The `istfft` function reconstructs a signal from its short-time Fourier transform.
- The `iscole` 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`, `mcohore`, `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.

R2018a

Version: 8.0

New Features

Bug Fixes

Compatibility Considerations

Signal Analyzer App: Preprocess signals by smoothing and filtering

The **Signal Analyzer** app now enables you to perform basic preprocessing of your data within the app itself. Generate MATLAB scripts to apply the same preprocessing steps to any number of signals.

- You can lowpass, highpass, bandpass, or bandstop filter the signals.
- You can smooth signals using any of several available methods: Savitzky-Golay, Gaussian, moving mean or median, linear or quadratic regression, or robust linear or quadratic regression. The app uses the MATLAB function `smoothdata` to perform the smoothing.

Signal Analyzer App: Detect transients and perform time-frequency analysis using scalogram view

The **Signal Analyzer** app now computes scalograms. Scalograms enable you to detect transients and perform time-frequency analysis. You must have a Wavelet Toolbox license to view scalograms.

One-Step Filtering: Filter signals using lowpass, highpass, bandpass, and bandstop responses

This release introduces four functions that enable users to filter single- and multichannel signals and timetables in one step and without having to design the filter: `lowpass`, `highpass`, `bandpass`, and `bandstop`.

Each function designs and applies a minimum-order filter and compensates for the delay.

Time-Frequency Analysis: Perform empirical mode decomposition, Hilbert-Huang transform, and instantaneous frequency estimation

This release introduces the `emd`, `hht`, and `instfreq` functions.

- `emd` performs empirical mode decomposition. Use `emd` to decompose a nonlinear/nonstationary process into its intrinsic modes of oscillation. `emd` iterates on an input signal to extract the natural AM-FM modes, or intrinsic mode functions, contained in the data.
- `hht` implements the Hilbert-Huang transform. Use `hht` to obtain a time-frequency representation of a signal similar but complementary to the spectrogram or continuous wavelet transform. `hht` uses the data-adaptive intrinsic mode functions obtained from the empirical mode decomposition to obtain instantaneous frequency estimates of a multicomponent nonlinear/nonstationary signal.
- `instfreq` computes the instantaneous frequency of a signal, either as the first conditional spectral moment of a time-frequency distribution or as the derivative of the phase of an analytic signal.

Time-Frequency Analysis: Estimate kurtogram, spectral kurtosis, and spectral entropy

This release introduces three functions that perform spectral measurements.

- `pkurtosis` computes the spectral kurtosis of a signal. Spectral kurtosis is a statistical tool that indicates and pinpoints nonstationary or non-Gaussian behavior in the frequency domain.

- `kurtogram` computes and displays the fast kurtogram of a signal. The kurtogram is a tool for detecting and characterizing signal nonstationarities that provides key information for performing spectral kurtosis analysis.
- `pentropy` computes the spectral entropy of a signal. Spectral entropy measures how spiky or flat the spectrum of a signal is. A signal with a spiky spectrum, like a sum of sinusoids, has low spectral entropy. A signal with a flat spectrum, like white noise, has high spectral entropy.

p octave Function: Compute 1/N octave spectra and perform octave smoothing

The `p octave` function enables you to compute the octave spectrum of a single- or multichannel signal or timetable. The function determines the average power over octave bands defined by the ANSI S1.11 standard.

Rotating Machinery: Estimate and track rotational speed from vibration signals

The `rpmtrack` function enables users to measure and track time-dependent rotational speed. The function displays an interactive plot on which you can select ridge points using the mouse.

Deep Learning Example: Classify signals using long short-term memory networks

This release introduces an example that shows one way to combine signal processing and deep learning for the analysis of physiologic signals. `Classify ECG Signals Using Long Short-Term Memory Networks` classifies electrocardiogram data using time-frequency analysis and a type of recurrent neural network that is well-suited to study time series.

Functionality being removed or changed

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

Compatibility Considerations

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
Signal Browser	Still runs but issues a warning	Signal Analyzer app	Replace relevant instances of <code>sptool</code> with <code>signalAnalyzer</code> .
Filter Design and Analysis Tool	Still runs but issues a warning	Filter Designer app	Replace relevant instances of <code>sptool</code> with <code>filterDesigner</code> .
Filter Visualization Tool	Still runs but issues a warning	<code>fvtool</code>	Replace relevant instances of <code>sptool</code> with <code>fvtool</code> .

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
Spectrum Viewer	Still runs but issues a warning	Signal Analyzer app	Replace relevant instances of <code>sptool</code> with <code>signalAnalyzer</code> .

R2017b

Version: 7.5

New Features

Bug Fixes

Compatibility Considerations

Signal Analyzer App: Analyze sporadic signals with persistence spectrum and sharpen time-frequency estimates using reassignment

The **Signal Analyzer** app now computes persistence spectra and reassigned spectrograms.

- The persistence spectrum contains time-dependent probabilities of occurrence of signals at given frequency locations and power levels. This type of spectrum is useful for detecting brief events.
- The reassignment technique sharpens the time and frequency localization of spectrograms and makes them easier to read and interpret.

Signal Analyzer App: Extract and export signal regions of interest

The **Signal Analyzer** app now enables you to extract regions of interest from signals. You can analyze the regions of interest in more depth in the app itself, export them to the MATLAB workspace, or save them to a MAT-file.

Signal Analyzer App: Generate MATLAB scripts to automate analysis

The **Signal Analyzer** app now generates MATLAB scripts to extract signal regions of interest and to compute power spectrum, spectrogram, or persistence spectrum estimates.

pspectrum Function: Analyze power spectrum, spectrogram, and persistence spectrum of signals

The `pspectrum` function enables users to compute, display, and analyze power spectra, spectrograms, reassigned spectrograms, and persistence spectra of signals. The function's smart defaults provide reasonable spectral estimates without the need to specify many parameters.

Rotating Machinery: Remove noise coherently with time-synchronous averaging and analyze wear using envelope spectra

The `tssa` and `envspectrum` functions enable you to analyze rotating machinery.

- Use the `tssa` function to remove noise coherently. The function interpolates a signal and averages the values occurring at equal values of rotational phase.
- The `envspectrum` function enables you to study wear in bearings. The function finds the envelope of a vibration signal and computes the spectrum of the envelope.

For an example that uses these functions, see [Vibration Analysis of Rotating Machinery](#).

Modal Analysis: Use parametric methods for FRF and modal parameter estimation

This release enhances the `modalfrf`, `modalfit`, and `modalsd` functions.

- `modalfrf` and `modalsd` now enable you to estimate frequency-response functions using a state-space model.
- `modalfit` can use a least-squares rational function method to estimate modal parameters.

- `modalfrf` and `modalfit` accept identified systems as input. You must have a System Identification Toolbox™ license to use this functionality.

Fatigue Analysis: Perform high-cycle rainflow counting

This release introduces the `rainflow` function for fatigue analysis.

`findchangepts` Function: Find changepoints in spectrograms and other multivariate signals

The `findchangepts` function can now find changepoints of spectrograms and multivariate signals.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>spectrum</code>	Errors	<code>periodogram</code> or <code>pspectrum</code>	Replace all instances of <code>spectrum</code> with <code>periodogram</code> or <code>pspectrum</code> .
<code>psd</code>	Errors	<code>periodogram</code> or <code>pwelch</code>	Replace all instances of <code>psd</code> with <code>periodogram</code> or <code>pwelch</code> .

R2017a

Version: 7.4

New Features

Bug Fixes

Signal Analyzer App: Perform time-frequency analysis using spectrogram view

The **Signal Analyzer** app now enables time-frequency analysis of any signals in the MATLAB workspace.

Signal Analyzer App: Analyze timestamped signals

The **Signal Analyzer** app now accepts signal time values defined using vectors, `timeseries` objects, or MATLAB timetables.

Modal Analysis: Estimate frequency-response functions and modal parameters of mechanical systems

The `modalfrf`, `modalfit`, and `modalsd` functions perform experimental modal analysis of signals associated with mechanical systems.

- `modalfrf` uses measured input and output signals to estimate H_1 , H_2 , and H_v frequency-response functions. The function accepts multiple inputs and outputs.
- `modalfit` extracts natural frequencies, damping ratios, and mode-shape vectors from a set of frequency-response functions.
- `modalsd` plots the stabilization diagram of a SISO or MIMO system starting from its frequency-response functions.

For an example that uses these functions, see [Modal Analysis of a Simulated System and a Wind Turbine Blade](#).

Cross-Spectrogram: Compare time-frequency content of nonstationary signals

The `xspectrogram` function uses the short-time Fourier transform to compute cross-spectra of signals whose frequency content changes with time. Use the cross-spectrogram to identify time-frequency regions where two different signals have energy, or to measure time-varying phase shifts.

MIMO Spectral Analysis: Estimate cross-spectral density and coherence for multi-input/multi-output systems

The `cpsd` and `mscohere` functions now support systems with multiple inputs and outputs.

Transfer Function Estimation: Compute unbiased estimates for SISO and MIMO systems containing additive input noise

The `tfestimate` function now returns the H_2 estimate of the transfer function in addition to the already available H_1 estimate. It also supports systems with multiple inputs and outputs.

dct and idct Functions: Compute four standard types of discrete cosine transform

The `dct` and `idct` functions now compute the DCT-1, DCT-3, and DCT-4 variants of the discrete cosine transform, in addition to the already available DCT-2. You can also choose the array dimension along which to transform a multidimensional array.

Code Generation Support: Generate code for an expanded set of Signal Processing Toolbox functions

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

- `alignsignals`
- `cconv`
- `convmtx`
- `corrmtx`
- `envelope`
- `finddelay`
- `hilbert`
- `sgolayfilt`
- `sinc`
- `xcorr2`
- `xcov`

You must have MATLAB Coder and Signal Processing Toolbox software to generate standalone C and C++ code for supported functions. See Signal Processing Toolbox (MATLAB Coder) for a complete list of Signal Processing Toolbox functions with code generation support.

findpeaks Function: Single-precision support and improved code generation functionality

The `findpeaks` function now accepts single-precision input. Its code generation functionality has improved performance and supports plotting.

R2016b

Version: 7.3

New Features

Bug Fixes

Compatibility Considerations

Signal Analyzer App: Perform time- and frequency-domain analysis of multiple time series

The Signal Analyzer app now enables plotting, alignment, and comparison of any signals in the MATLAB workspace, both in the time domain and in the frequency domain. The new panner enables you to zoom in on and scroll through signals. The app also now has a command-line interface.

Similarity Matching: Find patterns in data using edit distance or dynamic time warping

The `edr` function measures how similar two signals are by computing the minimum number of insert, delete, or replace operations needed to convert one of the signals into the other. This measure, called *edit distance*, is similar to the measure used in `dtw` but is robust to outliers.

The `findsignal` function finds the region of a data set that best matches a specified pattern. You can search by minimizing a distance metric, by dynamic time warping, or by using edit distance.

Order Analysis: Track orders and extract waveforms to analyze rotational machinery

This release introduces four functions for order analysis of rotational machinery.

- To extract an RPM signal from a series of tachometer pulses, use the `tachorpm` function.
- To estimate the average spectrum of a vibration signal as a function of order, use the `orderspectrum` function.
- To track and extract orders in a vibration signal, use the `ordertrack` function.
- To extract time-domain order waveforms from a vibration signal, use the `orderwaveform` function.

Fourier Synchrosqueezing Transform: Obtain sharp time-frequency estimates and extract signal modes

This release adds support for the FFT-based synchrosqueezed transform and mode extraction for 1-D signals. Synchrosqueezing is a time-frequency reassignment technique that enables you to reconstruct a signal from the reassigned transform. This technique enables you to extract and visualize oscillatory modes in the signal.

- To obtain the Fourier synchrosqueezed transform of a signal, use the `fsst` function.
- To invert a Fourier synchrosqueezed transform, use the `ifsst` function.
- To extract time-frequency ridges from a signal, use the `tfridge` function.

Distortion Measurement Functions: Measure aliased harmonics in undersampled signals

The `snr` and `thd` functions now incorporate any harmonics of the fundamental that are aliased into the Nyquist range.

- By default, `snr` treats aliased harmonics as part of the noise. Now it is possible to treat them as part of the signal.

- By default, `thd` treats aliased harmonics as part of the noise. Now it is possible to treat them as harmonics.

GPU acceleration: Enhance performance of `dct`, `idct`, and `sinc` functions

This release introduces GPU acceleration for `dct`, `idct`, and `sinc`. GPU acceleration for these functions requires Parallel Computing Toolbox software. See GPU Computing for details. The supported Signal Processing Toolbox functions accept `gpuArray` objects as inputs.

Filter Design and Analysis Tool renamed to Filter Designer

The Filter Design and Analysis Tool (FDATool) has been renamed to Filter Designer. To open the **Filter Designer** app from the command line, use `filterDesigner` instead of `fdatool`.

Compatibility Considerations

Change all calls to `fdatool`, which opens the app, to the new `filterDesigner` command. The functionality remains unchanged.

Window Design and Analysis Tool renamed to Window Designer

The Window Design and Analysis Tool (WinTool) has been renamed to Window Designer. To open the **Window Designer** app from the command line, use `windowDesigner` instead of `wintool`.

Compatibility Considerations

Change all calls to `wintool`, which opens the app, to the new `windowDesigner` command. The functionality remains unchanged.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>fdatool</code>	Still opens the Filter Designer app.	<code>filterDesigner</code>	Update instances of <code>fdatool</code> to use the new <code>filterDesigner</code> command.
<code>wintool</code>	Still opens the Window Designer app.	<code>windowDesigner</code>	Update instances of <code>wintool</code> to use the new <code>windowDesigner</code> command.

R2016a

Version: 7.2

New Features

Bug Fixes

Gap Filling: Reconstruct missing samples using autoregressive modeling

The `fillgaps` function uses autoregressive modeling to interpolate missing large contiguous portions of signals. For an example that uses this function, see [Reconstructing Missing Data](#).

Changepoint Detection: Find abrupt changes and statistical shifts in signals

The `findchangepts` function locates the points at which a signal significantly changes its mean, its variance, or both. The `cusum` function detects small incremental changes in the mean of a process by monitoring where the cumulative sum of the signal drifts beyond a target mean. For an example that uses these functions, see [Detecting Outbreaks and Significant Changes in Signals](#).

Dynamic Time Warping: Stretch, align, and compare signals with different time scales

The `dtw` function stretches two nonlinearly related signals onto a common time axis such that a global measure of distance between the signals is smallest. The process highlights similarities between signals by making equivalent features appear at the same location on the common axis. For an example that uses this function, see [Extracting Classification Features from Physiological Signals](#).

Reassigned Periodogram: Sharpen the frequency localization of spectral estimates

The `periodogram` function now computes reassigned spectra. The reassignment technique sharpens the frequency localization of spectral estimates and produces periodograms that are easier to read and interpret. For an example that uses this feature, see [Measuring the Power of Deterministic Periodic Signals](#).

Signal Analyzer App: Visualize and compare multiple time series

The Signal Analyzer app enables plotting, alignment, and comparison of any signals in the MATLAB workspace.

datetime Support: Use datetime arrays in Signal Processing Toolbox functions

The `findpeaks`, `plomb`, and `resample` functions now accept `datetime` arrays as input arguments.

xcorr Function: Generate faster code for long input vectors

For long input vectors, code generation for `xcorr` now uses a frequency-domain calculation instead of a time-domain calculation. The resulting code can be faster than in previous releases.

chebwin Function: Compute Dolph-Chebyshev windows faster

The `chebwin` function has been rewritten and exhibits improved performance for windows of any size.

R2015b

Version: 7.1

New Features

Bug Fixes

Reassigned Spectrogram: Sharpen the time-frequency localization of spectral estimates

The `spectrogram` function now computes reassigned spectra. The reassignment technique sharpens the time and frequency localization of spectral estimates and produces spectrograms that are easier to read and interpret. See the `spectrogram` reference page for examples. The enhanced function also enables you to specify a threshold such that spectrum values with less power are set to zero.

Hampel Filter and Improved Median Filtering: Detect outliers and remove them from data

The new `hampel` function detects and removes outliers from data. The function computes a moving median and considers as outliers those data points that deviate from the median by more than an adjustable number of standard deviations.

The `medfilt1` function now uses a faster algorithm and enables you to choose how to treat signals with NaNs and how to filter signal edges.

Order Analysis: Analyze vibrations in rotational machinery with order and frequency maps

The new `rpmordermap` and `rpmfreqmap` functions perform order analysis on rotational machinery. `rpmordermap` computes order maps of rotating systems, helping you visualize vibrations that vary linearly with rotational speed. `rpmfreqmap` computes frequency maps. You can also use these functions to plot the maps on interactive figures.

Envelope Detection: Extract analytic, peak, and RMS envelopes

The new `envelope` function computes the upper and lower envelopes of a waveform. You can compute the envelopes using an FFT-based analytic function, an FIR Hilbert filter, or the function `peaks` or `RMS` values.

Signal Alignment: Measure delay and align signals in time

The new `finddelay` and `alignsignals` functions compute the delay between two signals and align signals in time.

R2015a

Version: 7.0

New Features

Bug Fixes

Frequency Measurements: Compute mean and median frequencies using spectral estimates

This release introduces two functions that use spectral estimates to measure signal frequency:

- `meanfreq` estimates the mean frequency of the spectrum of a signal.
- `medfreq` estimates the median frequency of the spectrum of a signal.

These functions accept a time-domain signal, a power spectral density, or a power spectrum as input. They have visualization functionality and support multichannel signals.

Bandwidth Measurements: Compute occupied bandwidth and bandwidth at specified power levels

This release introduces two functions that measure signal bandwidth:

- `obw` returns the occupied bandwidth of a signal at a specified percentage.
- `powerbw` returns the power bandwidth of a signal at a specified level.

These functions accept a time-domain signal, a power spectral density, or a power spectrum as input. They have visualization functionality and support multichannel signals.

resample function accepts signals with nonuniform sampling or missing data

This release enhances the `resample` function to accept nonuniformly sampled signals and reconstruct them on a regular grid. The function also accepts NaNs in the input signal and treats them as missing data.

pwelch function computes maximum-hold and minimum-hold spectra

The `pwelch` function now computes the maximum-hold and minimum-hold power spectral density (PSD) estimates of a signal.

spectrogram function computes two-sided, centered, power, and PSD spectra

The `spectrogram` function now computes two-sided or centered spectra and can output the mean-square spectrum instead of the power spectral density. The enhanced function also plots a color bar with units.

Fast chirp Z-transform support for spectral analysis functions

All the spectral analysis functions offered by the Signal Processing Toolbox product let you input a vector with the frequencies at which to compute PSD estimates. The functions traditionally have used the Goertzel algorithm in those cases. Starting with this release, the functions might switch to a faster method based on the chirp Z-transform when the input frequency vector has many points and the points are uniformly spaced. The change applies to the following functions:

-
- Nonparametric methods: cpsd, mscohere, periodogram, pmtm, pwelch, spectrogram, tfestimate
 - Parametric methods: pburg, pcov, pmcov, pyulear

R2014b

Version: 6.22

New Features

Bug Fixes

Spectral estimation of signals with nonuniform sampling or missing data

This release introduces a new spectral estimation function, `plomb`. The function can compute spectra of nonuniformly sampled signals or signals with missing samples by using the Lomb-Scargle algorithm. `plomb` also provides visualization functionality and support for multichannel data.

Multichannel support for spectral analysis functions

This release enhances several parametric and nonparametric spectral analysis functions by offering support for multichannel data. The functions continue to process vector data as single channels. For matrix input, the functions process the data column by column and return a matrix of spectra.

- Nonparametric methods: `bandpower`, `cpsd`, `mscohere`, `periodogram`, `pmtm`, `pwelch`, `tfestimate`.
- Parametric methods: `arburg`, `arccov`, `armcov`, `aryule`, `pburg`, `pcov`, `pmcov`, `pyulear`.

Peak finding with visualization and enhanced peak selection

The function `findpeaks` now lets you estimate the width of any peak and limit the results of peak searches by width or prominence. When called with no output arguments, the function plots the signal and annotates the value, width, and prominence of every peak.

R2014a

Version: 6.21

New Features

Bug Fixes

Compatibility Considerations

Simplified workflow for specification-based filter design

This release introduces a new filter design function, `designfilt`. A single command allows you to design lowpass, highpass, bandpass, bandstop, differentiator, and Hilbert filters, including minimum-order designs, in one step. The `designfilt` interface leads to readable and self-documenting code that is easy to maintain.

Use `filter` to filter signals with the `digitalFilter` objects generated by `designfilt`.

Use `fvtool` to visualize digital filters created using `designfilt`.

All the filter analysis and data filtering functions offered by the Signal Processing Toolbox product can be used with digital filters designed with `designfilt`.

`designfilt` also features a Filter Design Assistant—a smart run-time error-recovery mechanism that helps you correct faulty syntax and generates MATLAB code at the Command Window or in the Editor.

Visualization of harmonic distortion measurements

This release enhances SNR, distortion, intermodulation, and spurious free dynamic range measurement functions to provide visualization functionality.

The function `sfdr` plots the spectrum of the input signal and labels its fundamental component and largest spur. The plot shades the spurious free dynamic range and displays its value; it shows the fundamental, the DC value, and the rest of the signal in different colors. `sfdr` now uses a Kaiser window by default.

The function `sinad` plots the spectrum of the input signal, labels its fundamental component, and displays its signal to noise and distortion ratio. The plot shows the fundamental component, the DC value, and the noise in different colors.

The function `snr` plots the spectrum of the input signal, labels its fundamental component and higher harmonics, and displays its signal-to-noise ratio. The plot shows the fundamental, the noise, and the DC value and harmonics using different colors.

The function `thd` plots the spectrum of the input signal, labels its fundamental component and harmonics, and displays its total harmonic distortion. The plot shows the fundamental, the harmonics, and the DC level and noise using different colors.

The function `toi` plots the spectrum of the input signal, annotates its lower and upper fundamentals and intermodulation products, and displays its third-order intercept point.

Changes in name-value pair arguments of measurement functions

The R2014a release changes the name-value pair arguments 'MidPct' and 'PctRefLevels' to 'MidPercentReferenceLevel' and 'PercentReferenceLevels' for the following signal measurement functions: `dutycycle`, `falltime`, `midcross`, `pulseperiod`, `pulsesep`, `pulsewidth`, `risetime`, `settlingtime`, `slewrate`, `overshoot`, and `undershoot`.

Compatibility Considerations

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
'MidPct' name-value pair of measurement functions	Still runs	'MidPercentReferenceLevel'	Replace all instances of 'MidPct' with 'MidPercentReferenceLevel'
'PctRefLevels' name-value pair of measurement functions	Still runs	'PercentReferenceLevels'	Replace all instances of 'PctRefLevels' with 'PercentReferenceLevels'

R2013b

Version: 6.20

New Features

Bug Fixes

Compatibility Considerations

Distortion, intermodulation, and SNR measurement functions

This release introduces four frequency-domain measurement functions that allow you to characterize the nonlinearity of a system. Quantitative measures of system linearity are important in a number of applications including audio system analysis, power electronics, and radio-frequency (RF) network analysis.

Use `thd` to measure the total harmonic distortion (THD) of a sinusoidal signal. THD is appropriate for signals with discrete spectra consisting of a fundamental frequency and one or more harmonics.

Use `sinad` to measure the signal to noise and distortion ratio (SINAD) of a signal. SINAD is appropriate for signals with mixed spectra consisting of a fundamental frequency with one or more harmonics and additive noise.

Use `toi` to measure the third-order intercept (TOI) point. TOI is used to quantify the intermodulation distortion of a system in response to a two-tone input.

Use `snr` to measure the signal-to-noise ratio (SNR) of a signal. SNR is appropriate when you want to measure the power of the fundamental frequency in comparison to the variance of additive noise.

Design raised cosine and Gaussian pulse-shaping filters

This release introduces two new functions for raised cosine and Gaussian pulse-shaping filter design. The new functions provide a uniform interface for transmit and receive pulse-shaping filters that is designed specifically for communication engineers.

Raised cosine filters are used to minimize intersymbol interference (ISI) by shaping pulses with a filter that satisfies the Nyquist ISI criterion. Use `rcosdesign` to design a raised cosine or square-root raised cosine finite impulse response pulse-shaping filter.

Gaussian pulse-shaping filters are used in modulation schemes such as Gaussian minimum shift keying (GMSK) to limit the bandwidth of pulse waveforms. Use `gaussdesign` to design a Gaussian finite impulse response pulse-shaping filter.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>firrcos</code>	Still runs	<code>rcosdesign</code>	Replace all instances of <code>firrcos</code> with <code>rcosdesign</code> . See Migrate from <code>firrcos</code> to <code>rcosdesign</code> .
<code>gaussfir</code>	Still runs	<code>gaussdesign</code>	Replace all instances of <code>gaussfir</code> with <code>gaussdesign</code> . See Migrate from <code>gaussfir</code> to <code>gaussdesign</code> .

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
fdesign.pulseshaping	Still runs	rcosdesign or gaussdesign	Replace all instances of fdesign.pulseshaping with rcosdesign or gaussdesign, as appropriate. See Migrate from fdesign.pulseshaping to rcosdesign or gaussdesign.

Migrate from firrcos to rcosdesign

When you update legacy code using firrcos to use rcosdesign instead, keep in mind that rcosdesign designs a square-root raised cosine filter by default. If you want a normal raised cosine filter you must set the optional parameter, shape, to 'normal'. firrcos has the opposite default behavior.

The number of samples per symbol must be an integer. Equivalently, the ratio $F_s/(2*F_c)$, where F_s is the sampling frequency and F_c is the cutoff frequency of the filter, must be an integer.

firrcos and rcosdesign use different normalizations for the filter coefficients. firrcos normalizes the coefficient so that the nominal passband gain is 1. rcosdesign sets the filter energy to 1.

firrcos	rcosdesign
<pre>Fs = 7000; N = 16; Fc = 500; df = 375; ag = firrcos(N,Fc,df,Fs) ug = firrcos(N,Fc,df,Fs,'sqrt')</pre>	<pre>R = df/2/Fc; sps = Fs/2/Fc; ga = rcosdesign(R,N/sps,sps,'normal'); ga = ga/max(ga)/sps gu = rcosdesign(R,N/sps,sps); gu = gu/max(gu)*(1+R*(4/pi-1))/sps</pre>
<pre>N = 16; Fc = 1000; R = 0.25; Fs = 8000; b1 = firrcos(N,Fc,R,Fs, ... 'rolloff','normal')</pre>	<pre>beta = R; sps = Fs/(2*Fc); span = N / sps; b1n = rcosdesign(beta,span,sps,'normal'); b1n = b1n / max(b1n) / sps</pre>
<pre>N = 16; Fc = 1000; R = 0.25; Fs = 8000; b2 = firrcos(N,Fc,R,Fs, ... 'rolloff','sqrt')</pre>	<pre>beta = R; sps = Fs/(2*Fc); span = N / sps; b2n = rcosdesign(beta,span,sps,'sqrt'); b2n = b2n / max(b2n) / (pi*sps) ... * (pi*(1-beta) + 4*beta)</pre>

Migrate from `gaussfir` to `gaussdesign`

When you update legacy code using `gaussfir` to use `gaussdesign` instead, keep in mind the differences between the arguments in each function. In `gaussfir`, the second argument, `nt`, is the number of symbol periods between the start of the impulse response of the filter and its peak. It is thus equal to *half* the second argument, `span`, of `gaussdesign`. Moreover, `gaussdesign` has no default values for its second and third arguments. `gaussfir` respectively has 3 and 2.

gaussfir	gaussdesign
<code>bt = 0.3; h = gaussfir(bt)</code>	<code>hn = gaussdesign(bt,6,2)</code>
<code>nt = 5; h = gaussfir(bt,nt)</code>	<code>span = 2*nt; hn = gaussdesign(bt,span,2)</code>
<code>of = 3; h = gaussfir(bt,nt,of)</code>	<code>sps = of; hn = gaussdesign(bt,span,sps)</code>

Migrate from `fdesign.pulseshaping` to `rcosdesign` or `gaussdesign`

The use of `fdesign.pulseshaping` is not recommended. Use `rcosdesign` or `gaussdesign`, as appropriate.

fdesign.pulseshaping	rcosdesign, gaussdesign
<code>sps = 6; span = 4; Beta = 0.25; f1 = fdesign.pulseshaping(sps, ... 'Square Root Raised Cosine', ... 'Nsym,Beta',span,Beta); d1 = design(f1); n1 = d1.Numerator</code>	<code>n1n = rcosdesign(Beta,span,sps); n1n = n1n / max(n1n) * (-1/(pi*sps) ... * (pi*(Beta-1) - 4*Beta))</code>
<code>g1 = fdesign.pulseshaping(sps, ... 'Square Root Raised Cosine', ... 'N,Beta',sps*span,Beta); h1 = design(g1); k1 = h1.Numerator</code>	<code>k1n = rcosdesign(Beta,span,sps); k1n = k1n / max(k1n) * (-1/(pi*sps) ... * (pi*(Beta-1) - 4*Beta))</code>
<code>f2 = fdesign.pulseshaping(sps, ... 'Raised Cosine', ... 'Nsym,Beta',span,Beta); d2 = design(f2); n2 = d2.Numerator</code>	<code>n2n = rcosdesign(Beta,span,sps,'normal'); n2n = n2n/max(abs(n2n))/sps</code>
<code>g2 = fdesign.pulseshaping(sps, ... 'Raised Cosine', ... 'N,Beta',sps*span,Beta); h2 = design(g2); k2 = h2.Numerator</code>	<code>k2n = rcosdesign(Beta,span,sps,'normal'); k2n = k2n/max(abs(k2n))/sps</code>
<code>BT = 0.3; f3 = fdesign.pulseshaping(sps, ... 'Gaussian', ... 'Nsym,BT',span,BT); d3 = design(f3); n3 = d3.Numerator</code>	<code>n3n = gaussdesign(BT,span,sps)</code>

R2013a

Version: 6.19

New Features

Bug Fixes

Compatibility Considerations

Functions to measure equivalent noise bandwidth, band power, and spurious-free dynamic range

This release introduces three new functions for power measurements in the frequency domain. To measure the average power contained in a specified frequency interval, use `bandpower`. You can measure the spurious-free dynamic range using `sfdR`. The spurious-free dynamic range is the ratio of powers for the fundamental, or carrier frequency, to the next largest component. The function `enbw` measures the equivalent noise bandwidth of a window. The equivalent noise bandwidth is the width of an ideal rectangular filter with peak power equal to the peak power of the window. The product of the peak power and the equivalent noise bandwidth is equal to the power obtained by integrating under the magnitude-squared Fourier transform of the window function.

Function interface to compute power spectrum with confidence intervals and DC-centered spectra

This release enhances functions for nonparametric and parametric power spectral density (PSD) function estimation. In R2013a, nonparametric PSD estimators provide confidence intervals based on the chi-square probability distribution. Parametric PSD estimators provide confidence intervals based on the Gaussian probability distribution. All nonparametric and parametric PSD estimators include the ability to obtain a two-sided PSD estimate with 0 frequency (DC) in the center. See `periodogram`, `pwelch`, and `pmtm` for information on obtaining confidence intervals and DC centered spectra for nonparametric PSD estimation. See `pburg`, `pcov`, `pmcov`, and `pyulear` for information on obtaining confidence intervals and DC centered spectra for parametric PSD estimation.

Function interface for analysis and implementation of single-precision filters

This release enhances a number of filter analysis and implementation functions to accept single-precision floating point inputs. In R2013a, you can use single-precision filter coefficients with filter analysis functions, such as `freqz`, `fvtool`, `grpdelay`, `impz`, `phasez`, and `zerophase`. If you input single-precision filter coefficients, the analysis function outputs are single precision.

In R2013a, the filtering function, `sosfilt`, accepts single-precision inputs for both the filter coefficients and data. If either the second-order section matrix input or the data is single-precision, `sosfilt` outputs single-precision data.

The filtering functions `fftfilt`, `filtfilt`, and `latcfilt` still require double-precision inputs.

Function interface for analysis of second-order section (biquad) filters

The R2013a release adds SOS matrix input support for filter analysis functions, such as `freqz`, `fvtool`, `grpdelay`, `impz`, `phasez`, and `zerophase`.

Code generation support for Signal Processing Toolbox functions

This release removes the requirement for DSP System Toolbox™ to generate standalone C and C++ code for supported Signal Processing Toolbox functions. You must have MATLAB Coder and Signal Processing Toolbox software to generate standalone C and C++ code for supported functions. See

Functions Supported for Code Generation for a list of Signal Processing Toolbox functions with code generation support if you have MATLAB Coder software.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>dspdata.avgpower</code>	Still runs	<code>bandpower</code>	Replace all instances of <code>dspdata.avgpower</code> with <code>bandpower</code>
<code>dspdata.msspectrum</code>	Still runs	<code>periodogram</code> , <code>pwelch</code>	Replace all instances of <code>dspdata.msspectrum</code> with the appropriate function interface
<code>dspdata.psd</code>	Still runs	<code>pburg</code> , <code>pcov</code> , <code>periodogram</code> , <code>pmcov</code> , <code>pmtm</code> , <code>pwelch</code> , <code>pyulear</code>	Replace all instances of <code>dspdata.psd</code> with the appropriate function interface
<code>dspdata.pseudospectrum</code>	Still runs	<code>peig</code> , <code>pmusic</code>	Replace all instances of <code>dspdata.pseudospectrum</code> with the appropriate function interface
<code>dspdata.sfdr</code>	Still runs	<code>sfdr</code>	Replace all instances of <code>dspdata.sfdr</code> with <code>sfdr</code>
<code>sigwin.barthannwin</code>	Still runs	<code>barthannwin</code>	Replace all instances of <code>sigwin.barthannwin</code> with <code>barthannwin</code>
<code>sigwin.bartlett</code>	Still runs	<code>bartlett</code>	Replace all instances of <code>sigwin.bartlett</code> with <code>bartlett</code>
<code>sigwin.blackman</code>	Still runs	<code>blackman</code>	Replace all instances of <code>sigwin.blackman</code> with <code>blackman</code>
<code>sigwin.blackmanharris</code>	Still runs	<code>blackmanharris</code>	Replace all instances of <code>sigwin.blackmanharris</code> with <code>blackmanharris</code>
<code>sigwin.bohmanwin</code>	Still runs	<code>bohmanwin</code>	Replace all instances of <code>sigwin.bohmanwin</code> with <code>bohmanwin</code>
<code>sigwin.chebwin</code>	Still runs	<code>chebwin</code>	Replace all instances of <code>sigwin.chebwin</code> with <code>chebwin</code>
<code>sigwin.flattopwin</code>	Still runs	<code>flattopwin</code>	Replace all instances of <code>sigwin.flattopwin</code> with <code>flattopwin</code>

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
sigwin.gausswin	Still runs	gausswin	Replace all instances of sigwin.gausswin with gausswin
sigwin.hamming	Still runs	hamming	Replace all instances of sigwin.hamming with hamming
sigwin.hann	Still runs	hann	Replace all instances of sigwin.hann with hann
sigwin.kaiser	Still runs	kaiser	Replace all instances of sigwin.kaiser with kaiser
sigwin.nuttallwin	Still runs	nuttallwin	Replace all instances of sigwin.nuttallwin with nuttallwin
sigwin.parzenwin	Still runs	parzenwin	Replace all instances of sigwin.parzenwin with parzenwin
sigwin.rectwin	Still runs	rectwin	Replace all instances of sigwin.rectwin with rectwin
sigwin.taylorwin	Still runs	taylorwin	Replace all instances of sigwin.taylorwin with taylorwin
sigwin.triang	Still runs	triang	Replace all instances of sigwin.triang with triang
sigwin.tukeywin	Still runs	tukeywin	Replace all instances of sigwin.tukeywin with tukeywin
spectrum.burg	Still runs	pburg	Replace all instances of spectrum.burg with pburg
spectrum.cov	Still runs	pcov	Replace all instances of spectrum.cov with pcov
spectrum.eig	Still runs	peig	Replace all instances of spectrum.eig with peig
spectrum.mcov	Still runs	pmcov	Replace all instances of spectrum.mcov with pmcov
spectrum.mtm	Still runs	pmtm	Replace all instances of spectrum.mtm with pmtm
spectrum.music	Still runs	pmusic	Replace all instances of spectrum.music with pmusic

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
<code>spectrum.periodogram</code>	Still runs	<code>periodogram</code>	Replace all instances of <code>spectrum.periodogram</code> with <code>periodogram</code>
<code>spectrum.welch</code>	Still runs	<code>pwelch</code>	Replace all instances of <code>spectrum.welch</code> with <code>pwelch</code>

R2012b

Version: 6.18







New Features

Bug Fixes

Compatibility Considerations

Signal Browser in SPTool

The R2012b release introduces a revamped Signal Browser GUI. Whenever you select one or more signals in the **Signals** list box and click the **View** button, The Signal Browser GUI launches from the SPTool GUI. The Signal Browser allows you to perform the following additional operations:

- Time Domain Measurements Panels — The Signal Browser includes new side panels labeled **Trace Selection**, **Cursor Measurements**, **Signal Statistics**, **Bilevel Measurements**, and **Peak Finder**.
 - The **Trace Selection** panel allows you to choose which signal to make the active signal when you select multiple input signals in the **Signals** list box in SPTool. This panel replaces the **Select Trace** toolbar button () and **Select a trace** dialog box that were used in previous releases to choose an active signal. In the Signal Browser menu, select **Tools > Measurements > Trace Selection**.
 - The **Cursor Measurements** panel displays screen cursors. This panel replaces the **Markers** menu and all the corresponding toolbar buttons that were used in previous releases to show and configure markers. In the Signal Browser toolbar, click the Cursor Measurements button (). Alternatively, in the Signal Browser menu, select **Tools > Measurements > Cursor Measurements**.
 - The **Signal Statistics** panel displays the maximum, minimum, peak-to-peak difference, mean, median, and RMS values of a selected signal. It also displays the times at which the maximum and minimum values occur. In the Signal Browser toolbar, click the Signal Statistics button (). Alternatively, in the Signal Browser menu, select **Tools > Measurements > Signal Statistics**.
 - The **Bilevel Measurements** panel displays information about a selected signal's transitions, overshoots or undershoots, and cycles. In the Signal Browser toolbar, click the Bilevel Measurements button (). Alternatively, in the Signal Browser menu, select **Tools > Measurements > Bilevel Measurements**.
 - The **Peak Finder** panel displays maxima and the times at which they occur. These displays allow you to modify the settings for peak threshold, maximum number of peaks, and peak excursion. In the Signal Browser toolbar, click the Peak Finder button (). Alternatively, in the Signal Browser menu, select **Tools > Measurements > Peak Finder**.
- Multiple Display Support — R2012b enhances the Signal Browser by allowing you to choose to have multiple displays. This feature allows you to tile your screen into a number of separate displays, up to a grid of 4 rows and 4 columns. You may find multiple displays useful when you select multiple input signals in the **Signals** list box in SPTool. To set the number of displays, in the Signal Browser toolbar, click the Layout button (). Alternatively, in the Signal Browser menu, select **View > Layout**.
- Style dialog box — R2012b enhances the Signal Browser by allowing you to customize the style of displays using a Style dialog box. You can change the color of the figure containing the displays, the background and foreground colors of display axes, and properties of lines in a display. The Style dialog box replaces the **Line Properties** toolbar button and **Edit Line** dialog box that were used in previous releases for customizing line properties. To view or modify the line style of the active signal, in the Signal Browser menu, select **View > Style**. You can also right-click on the display and select **Style**. The Signal Browser — Style dialog box opens.

- **Sampled data as stairs** — The Signal Browser can display a signal as a staircase graph. A staircase graph includes only horizontal lines and vertical lines. Each horizontal line represents the signal value for a discrete sample period and is connected to two vertical lines. Each vertical line represents a change in values occurring at a sample. Using this approach is equivalent to using the MATLAB `stairs` function. Stairstep graphs are useful for drawing time history graphs of digitally sampled data. To display a sampled signal as a Stairstep graph, in the Signal Browser — Style dialog box, set the **Plot type** parameter to **Stairs**.
- **Properties dialog box** — R2012b enhances the Signal Browser by providing a central location where you can modify the properties of a display. To change options for a display, in the Signal Browser menu, select **View > Properties**. You can also right-click on the display and select **Properties**. The Signal Browser — Visuals:Time Domain Options dialog box opens.
 - **Complex data support** — The Signal Browser accepts complex-valued input signals and can visualize them in two distinct fashions. By default, the complex data is displayed in real and imaginary form as different-colored lines on the same axes. Alternately, you can display the magnitude and phase of the signal on separate axes in the same display. To change the complex data options, in the Signal Browser — Visuals:Time Domain Options dialog box, select the **Display** tab. Then, select or clear the **Plot signal(s) as magnitude and phase** check box.
 - **Ability to change the time units of the display** — The Signal Browser allows you to label the *time*-axis in three different ways. First, you can confirm the default operation, in which the Signal Browser displays time in metric units. The Signal Browser chooses the appropriate metric units, based on the minimum *time*-axis limit and the maximum *time*-axis limit of the scope window. Second, you can verify that the *time*-axis is always labeled as **Time (seconds)** and that the appropriate power of 10 appears in the bottom-right corner of the Time Scope display. Finally, you remove the units in the *time*-axis label entirely. To change the manner in which the time units are displayed, in the Signal Browser — Visuals:Time Domain Options dialog box, select the **Main** tab. Then, set the **Time Units** parameter to either **Metric** (based on **Time Span**), **Seconds**, or **None**, respectively.

See the Signal Browser reference topic for more information.

Compatibility Considerations

The R2012b Signal Browser GUI replaces the tool available in previous releases that was also named Signal Browser. In R2012b, you can still use the Signal Browser from previous releases, which is hereafter referred to as the Legacy Browser. To do so, execute the following steps.

- 1 In the SPTool menu, select **File > Preferences**.
- 2 In the list at the left side, select **Signal Browser**.
- 3 Under **Signal Browser**, select the **Use Legacy Browser (to be removed)** check box.
- 4 Click **OK**.

Now when you select one or more signals from the **Signals** list and click the **View** button, the Legacy Browser opens.

GPU acceleration for `xcorr`, `xcorr2`, `fftfilt`, `xcov`, and `cconv` functions

The R2012b release introduces GPU acceleration for `xcorr`, `xcorr2`, `fftfilt`, `xcov`, and `cconv`. GPU acceleration for these functions requires Parallel Computing Toolbox software.

If you have the Parallel Computing Toolbox, you can use `gpuArray` to create a GPUArray object. The supported Signal Processing Toolbox functions accept GPUArray objects as inputs.

R2012a

Version: 6.17

New Features

Measurements for Bilevel Pulse Waveforms

In R2012a, you can perform a number of basic measurements on bilevel pulse waveforms. These measurements include:

- State levels — You can estimate the state levels of a bilevel waveform using the histogram method with `statelevels`.
- Transition metrics — You can measure the rise time, fall time, and mid-reference level instants of waveform transitions. See the help for `midcross`, `risetime`, `falltime`, and `slewrates` for details. Additionally, R2012a introduces pulse metrics to measure bilevel waveform behavior in pretransition and posttransition regions including, `overshoot`, `undershoot`, and `settlingtime`.
- Duration metrics — See the help for `pulsewidth`, `pulseperiod`, `pulsesep`, and `dutycycle` for details.

Signal Statistics

The R2012a release introduces a number of basic signal statistics to augment the statistical measurements available in base MATLAB. See the function reference pages for `rms`, `rssq`, `peak2peak`, and `peak2rms` for details.

R2011b

Version: 6.16

New Features

Bug Fixes

Compatibility Considerations

Passband and Stopband Weights for Fixed-Order Unconstrained Partial Band Differentiator Filters

In R2011b, you can specify passband and stopband weights for a fixed-order unconstrained partial band differentiator filter design. You can access this capability through `filterbuilder` and `fdesign.differentiator`. With `fdesign.differentiator`, use the specification string `'N,Fp,Fst'`, and set the design method to `'equiripple'`. The following example shows you how to see the passband weight, `Wpass`, and stopband weight, `Wstop`, design options.

```
d = fdesign.differentiator('N,Fp,Fst',30,0.25,0.5);  
designopts(d,'equiripple')
```

Specify the passband or stopband weight values when you design your equiripple filter. For example:

```
Hd = design(d,'equiripple','Wstop',4);
```

Numerator and Denominator Order Specifications Added to filterbuilder for Lowpass and Highpass Butterworth Designs

In R2011b, you can specify different numerator and denominator orders in `filterbuilder` for lowpass and highpass Butterworth (maxflat) designs.

Access this option in `filterbuilder` by setting **Impulse response** under **Filter specifications** to **IIR** and **Order mode** to **Specify**. The default is equal order for both the numerator and denominator. Check **Denominator order** to specify a different denominator order.

Conversion of Error and Warning Message Identifiers

For R2011b, Signal Processing Toolbox error and warning message identifiers have changed.

Compatibility Considerations

If you have scripts or functions that use these changed message identifiers, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages. You can also use them in code that uses a `try/catch` statement and performs an action based on a specific error identifier.

If your code checks for a message identifier in a warning or error, you must update it to check for the new warning or error instead. To determine the identifier for a warning, run the following command just after you see the warning:

```
[MSG,MSGID] = lastwarn;
```

This command saves the message identifier to the variable `MSGID`.

To determine the identifier for an error, run the following command just after you see the error:

```
exception = MException.last;  
MSGID = exception.identifier;
```

Note Warning messages indicate a potential issue with your code. While you can turn off a warning, a suggested alternative is to change your code so that your code does not generate warnings.

R2011a

Version: 6.15

New Features

Bug Fixes

Compatibility Considerations

Enhancements to `filtfilt`

In R2011a, there are two major enhancements to `filtfilt`:

- 1 `filtfilt` has been completely rewritten to improve performance. Actual performance improvement depends on your hardware, filter length, signal length, and number of channels.
- 2 `filtfilt` now accepts IIR filters in second-order section (biquad) form.

Symmetric Window Option for Blackman-Harris Windows

In R2011a, `blackmanharris` and `nuttallwin` have a symmetric window design option. The input argument `SFLAG` controls the window option and defaults to `'symmetric'`. `'periodic'` returns a N-periodic window. The default symmetric option is preferred in FIR filter design because it results in linear phase. In spectral analysis applications, the periodic option is preferred.

Compatibility Considerations

In releases previous to R2011a, `blackmanharris` and `nuttallwin` only return N-periodic windows. To reproduce behavior in R2011a consistent with behavior in previous releases, use:

```
win = blackman(N,'periodic'); % N is the window length
```

or

```
win = nuttallwin(N,'periodic'); % N is the window length
```

`rectpuls` Returns Double-Precision Vector

In R2011a, `rectpuls` returns a double-precision vector instead of a logical vector.

Compatibility Considerations

In previous releases, `rectpuls` returns a logical vector. To produce behavior in R2011a consistent with previous releases, cast the output of `rectpuls` to a logical vector.

```
t = linspace(0,1,0.01);  
y = logical(rectpuls(t));
```

Code Generation from MATLAB and Fixed-Point MEX-File Generation

In R2011a, MathWorks® is no longer using the term *Embedded MATLAB* to refer to the language subset that supports code generation from MATLAB algorithms. This nomenclature incorrectly implies that the generated code is used in embedded systems only.

The new term is *code generation from MATLAB*. This terminology better reflects the full extent of the capability for translating MATLAB algorithms into readable, efficient, and compact MEX and C/C++ code for deployment to both desktop and embedded systems.

Signal Processing Toolbox users who have the DSP System Toolbox and MATLAB Coder software can generate deployable C/C++ code and MEX files using supported functions in the Signal Processing Toolbox.

You can find material on using Code Generation from MATLAB with the Signal Processing Toolbox software in Code Generation from MATLAB Support in Signal Processing Toolbox.

Users who have the DSP System Toolbox and Fixed-Point Toolbox™ can accelerate MEX-files for fixed-point applications using `fiaccel`.

Compatibility Considerations

The functionality associated with C/C++ and MEX code generation from MATLAB has changed in R2011a. These changes include:

- The compiler flag `#codegen` replaces `#eml`.
- `codegen` replaces `emlc` and `emlmex` for generating deployable C/C++ code and MEX-files from MATLAB algorithms. You must have the DSP System Toolbox and the MATLAB Coder software to use `codegen`.
- `fiaccel` replaces `emlmex` for generating fixed-point MEX code from MATLAB algorithms. To use `fiaccel`, you must have the DSP System Toolbox and Fixed-Point Toolbox software and your MATLAB code must satisfy the conditions described on the `fiaccel` reference page.

R2010b

Version: 6.14

New Features

Bug Fixes

Embedded MATLAB Support for Additional Signal Processing Toolbox Functions

In R2010b, Embedded MATLAB® supports `upsample` and `downsample` in the Signal Processing Toolbox.

R2010a

Version: 6.13

New Features

Bug Fixes

Compatibility Considerations

Single-Precision Support Added for `dfilt` Objects

In R2010a, users can construct `dfilt` objects with single-precision floating point arithmetic. Set the `Arithmetic` property to `'single'` to obtain a single-precision floating point representation of the filter coefficients.

Embedded MATLAB Support for Additional Signal Processing Toolbox Functions

In R2010a, Embedded MATLAB supports additional functions in the Signal Processing Toolbox. You can find a comprehensive list of supported functions in the Function Library Reference. You can find examples of using supported functions with the Signal Processing Toolbox software at Code Generation from MATLAB Support in Signal Processing Toolbox.

Functions, Objects, Object Methods, and Object Properties Being Removed

Name	What Happens When you use the Function, Object, Object Method, or Object Property	Use Instead	Compatibility Considerations
<code>addsection</code> (Method for <code>dfilt.cascade</code> and <code>dfilt.parallel</code> objects)	Errors	<code>addstage</code> method	Replace all instances of the <code>addsection</code> method with the <code>addstage</code> method for <code>dfilt.cascade</code> and <code>dfilt.parallel</code> objects.
<code>FFTLength</code> (Property of spectrum objects)	Errors	<code>'NFFT'</code> parameter in <code>psd</code> or <code>msspectrum</code> methods	Replace all instances of the <code>FFTLength</code> property in spectrum objects by the <code>'NFFT'</code> parameter in the <code>psd</code> or <code>msspectrum</code> methods.
<code>freqzplot</code>	Warns	<code>fvtool</code>	Replace all instances of <code>freqzplot</code> with <code>fvtool</code> .
<code>removesection</code> (Method for <code>dfilt.cascade</code> and <code>dfilt.parallel</code> objects)	Errors	<code>removestage</code> method	Replace all instances of the <code>removesection</code> method with the <code>removestage</code> method for <code>dfilt.cascade</code> and <code>dfilt.parallel</code> objects.

Name	What Happens When you use the Function, Object, Object Method, or Object Property	Use Instead	Compatibility Considerations
Section (Property of <code>dfilt.cascade</code> and <code>dfilt.parallel</code> objects)	Errors	Stage property	Replace all instances of the Section property with the Stage property for <code>dfilt.cascade</code> and <code>dfilt.parallel</code> objects.
Sidelobe_atten (Property of <code>sigwin.chebwin</code> object)	Errors	SidelobeAtten property	Replace all instances of the Sidelobe_atten property with the SidelobeAtten property for <code>sigwin.chebwin</code> objects.
specplot	Warns	plot method with DSPDATA object	Replace all instances of specplot with the plot method for DSPDATA objects.

Warning for Filter Designer in SPTool

In R2010a, the use of Filter Designer in `sptool` is not recommended. Use `fdatool` instead. Under File → Preferences → Filter Designer in SPTool, you may still select to use Filter Designer, but you will receive a warning that Filter Designer will be removed in a future release. If you select Filter Designer, you will be prompted to change your preferences in the `sigprefs.mat` and `startup.spt` files when you exit SPTool. Changing your preferences to use Filter Designer results in a warning each time SPTool starts. See Setting Preferences and Saving and Loading Sessions for details.

Compatibility Considerations

Because a future release will remove Filter Designer, use `fdatool` instead. Filters created in Filter Designer are not compatible with FDATool. Under File → Preferences → Filter Designer in `sptool`, you can select to use FDATool. You receive a prompt to convert filters created in Filter Designer to a format compatible with FDATool. When you exit `sptool` after changing your preferences, you receive another prompt, instructing you to update your preferences to use FDATool.

R2009b

Version: 6.12

New Features

Bug Fixes

Embedded MATLAB Support Added to Signal Processing Toolbox Functions

In R2009b, Embedded MATLAB supports the generation of embeddable C code for a subset of Signal Processing Toolbox filter design and window generation functions. You must install both the Signal Processing Toolbox and Signal Processing Blockset™ software to use this feature. Depending on which Embedded MATLAB feature you wish to use, additional products are required. The generated C code meets the strict memory and data type requirements of embedded target environments. See Code Generation from MATLAB Support in Signal Processing Toolbox for a list of supported Signal Processing Toolbox functions and examples.

Ability to Export Filter Coefficients Added to `realizemdl`

If you use Simulink®, you can now use the new `MapCoeffstoPorts` property with `realizemdl` to map filter coefficients from `dfilt` objects to constant blocks. The coefficients also appear in the MATLAB workspace providing tunability to the realized Simulink model. See `dfilt` for a list of supported filter structures and any restrictions.

R2009a

Version: 6.11

New Features

Bug Fixes

New filter design approach using `fdesign` and `filterbuilder`

A new and more robust way to design filters has been added to the toolbox — `fdesign` objects and the `filterbuilder` GUI. The following filter responses are supported: lowpass, highpass, bandpass, bandstop, Hilbert, differentiator, pulse-shaping (including FIR Gaussian) and arbitrary magnitude. Advanced design methods and additional responses are available in Filter Design Toolbox™.

New `dfilt` method to specify filter coefficients at block ports

A new option has been added to the `dfilt` block method to specify filter coefficients via Simulink block ports.

R2008b

Version: 6.10

New Features

Bug Fixes

New Walsh-Hadamard Transform functions

A new fast Walsh-Hadamard transform `fwht` function and an inverse fast Walsh-Hadamard transform `ifwht` have been added to the toolbox. An associated demo has also been added.

R2008a

Version: 6.9

New Features

Bug Fixes

New Marcum Q Function

A new `marcumq` function, which implements the generalized Marcum Q function, has been added to the toolbox.

Conversion Between Magnitude and dB Added

The new utility functions `mag2db` and `db2mag` have been added for converting from magnitude to dB and dB to magnitude, respectively.

PMTM Function Enhanced with Ability to Keep or Drop Last Taper

You can now specify whether to keep or drop the last taper for calculating the `pmtm`, which returns the PSD using the Thomson multitaper method. By default, the last taper is dropped. If you set the `DropLastTaper` property false, the last taper is included.

R2007b

Version: 6.8

New Features

Bug Fixes

Confidence Interval Estimation Added

The `spectrum` object has been enhanced with a new method for calculating confidence intervals for PSDs and mean-squared spectra.

Spurious-Free Dynamic Range (SFDR) Measurement Added

The `dspdata` object has been enhanced with a new method to measure spurious-free dynamic range (SFDR) for mean-squared spectra.

Local Maxima/Peak Finder Added

A new function `findpeaks` has been added to identify local maxima in a data vector. You can specify the minimum peak height and distance from its neighbors to limit the results. A `findpeaks` method has also been added to the `dspdata` object.

Conversions Between Power and dB Added

The new utility functions `pow2db` and `db2pow` have been added for converting from power to dB and dB to power, respectively.

R2007a

Version: 6.7

New Features

Bug Fixes

lsf2poly and latcfilter Multi-Channel Input Support Added

`lsf2poly` and `latcfilter` now support N-D array input where each column represents a separate input channel.

Circular Convolution (cconv) Function Added

A new function (`cconv`) that computes circular convolution has been added to the toolbox.

Spectrum Objects Partial Frequency Range Input Support Added

spectrum objects now support computing the spectrum and pseudospectrum on a user-specified vector of frequencies. This vector identifies the frequencies at which the spectrum or pseudospectrum is calculated.

cceps Factorize Algorithm Information Clarified

A more detailed explanation of the factorize algorithm and an example have been added to the `cceps` reference page.

dfilt.statespace Now Supports realizemdl Method

You can now create a Simulink block from `dfilt.statespace` objects with the `dfilt realizemdl` method.

ellip and ellipap Functions Enhanced

Both `ellip` and `ellipap` have been enhanced so that they are able to handle filters with more stringent requirements.

R2006b

Version: 6.6

New Features

Bug Fixes

Compatibility Considerations

Frequency Vector Input Added to Spectral Analysis Functions

The spectral analysis command line functions (`pburg`, `pcov`, `peig`, `periodogram`, `pmcov`, `pmtm`, `pmusic`, `pwelch`, and `pyulear`) now accept a frequency vector as an input parameter. This vector identifies the frequencies at which the spectral analysis function returns an estimate.

For functions that use the Goertzel algorithm (`periodogram`, `pmtm`, `pwelch`, and `spectrogram`), the frequency inputs are rounded to match the nearest bin value used by the algorithm.

FFT Length in Spectral Analyses Changed

The `FFTLength` parameter has been removed from all `spectrum` objects and you now specify the number of FFT points (NFFT) via the `psd`, `msspectrum`, or `pseudospectrum` estimation method. The NFFT value can be an integer or a string (either `'Nextpow2'` or `'Auto'`). `'Nextpow2'` is the default and sets the number of FFT points to the next power of 2 greater than the input signal length (or the segment length for `spectrum.welch` objects). `'Auto'` sets the number of FFT points to be equal to the input signal or segment length.

Compatibility Considerations

You should update any existing code that specifies the `FFTLength` parameter and instead use the NFFT parameter associated with an estimation method.

You should also verify that any `spectrum.welch` objects are using the desired FFT length, since the FFT length is now based on the segment length instead of the input signal length.

sosfilt and dfilt filter Method Support Multidimensional Array Input

You can now input a multidimensional array to `sosfilt` and to the `dfilt` filter method.

dfilt block Method Supports Target Subsystem Destination and Link Between Command Line and Model

The `dfilt` block method now allows you to specify a target subsystem in your Simulink model where you want to place the block. Two new parameters implement this: `'Destination'` and `'Link2obj'`. The `'Destination'` specifies where to place the block and `'Link2obj'` creates a link between the block in your model and inputs from the command line.

gaussfir Algorithm Updated

The alpha parameter in the formula used in `gaussfir` has been updated to match the formula in Rappaport T.S., *Wireless Communications Principles and Practice*, 2nd Edition, Prentice Hall, 2001.

R2006a

Version: 6.5

New Features

Bug Fixes

Compatibility Considerations

Taylor Window Function Added

A new function `taylorwin` for generating Taylor windows has been added.

SPTool Filter Designer Replaced by FDATool

FDATool has replaced the SPTool Filter Designer as the preferred method for designing filters for use in SPTool. For details, see FDATool in the Signal Processing Toolbox documentation.

Compatibility Considerations

The format in which filters are saved differs between SPTool Filter Designer and FDATool. When you load an SPTool session with saved filters, you are prompted to upgrade your filters to use FDATool format.

sgolay Example Improved

The example for `sgolay` has been improved and expanded.

zp2sos zeroflag Parameter Added

A new parameter `zeroflag` has been added to `zp2sos`. This parameter is used for real zeros that are the negatives of each other. For these zeros it specifies whether to keep them together instead of ordering them according to proximity to poles.

Help for Objects Changed

To obtain help for objects, use `help object.constructor` instead of the old `help object/constructor`. Note that to obtain help for methods, you still use `help object/method`.

R14SP3

Version: 6.4

New Features

Bug Fixes

dfilt (Discrete-Time Filters) Delay Structure Added

A new delay structure (`dfilt.delay`) has been added to `dfilt` objects. This structure adds latency to any signal filtered with it.

WinTool/WVTool Normalize Magnitude Added

A `Normalize magnitude` option has been added to the **Analysis Parameters** of WinTool and WVTool magnitude plots.

FDATool/FVTool Plot Displays Improved

The default plots for filter responses have been improved. The y-axis autoscaling includes buffer regions around the data and shows only the significant data. To see all of the data without any buffer regions, select **Full view** from the **View** menu.

FVTool Passband Zoom Added

If you have a filter in FVTool that was created in FDATool or from a Filter Design Toolbox `fdesign` object, you can use **Passband** on the **View** menu to zoom the passband region.

R14SP2

Version: 6.3

New Features

Bug Fixes

Compatibility Considerations

FDATool and FVTool Changes

FDATool Spectral Rejection Masks Added

You can draw lines on your filter response in FDATool to indicate rejection areas.

FDATool Generated C Header File Complex Filter Support

FDATool now supports generating C header files for complex filters.

FDATool Tip of the Day Added

A new Tip of the Day dialog displays when you start FDATool. It contains tips and hints for using FDATool.

FDATool State Space Filters Support Removed

FDATool no longer supports state space filters.

Compatibility Considerations

If you load a saved FDATool session that contains a state space filter, it is converted to a direct-form II transposed filter.

FDATool/FVTool New Analysis Parameters Magnitude Response Options

Three new options have been added to the Analysis Parameters for magnitude response displays.

- **Normalize Magnitude to 1 (0 dB)** — displays the magnitude so that the maximum magnitude value occurs at 0 dB
- **Autoscale axes** — automatically scales the response data y-axis
- **dB Display Range** — If you are not using autoscale and the magnitude display is in dB, this allows you to specify the y-axis limits, .

FVTool SOS Filter Coefficients Display Enhancement

The coefficient view in FVTool now displays each section of a second-order section filter as a separate filter with its own numerator, denominator, and gain.

FVTool Default Phase Units Changed

The default units for the phase response in FVTool have been changed to radians. This is consistent with the phasez function.

Compatibility Considerations

You should verify that the FVTool phase plots generated by existing code display the desired units.

dfilt Changes

dfilt Coefficients Method Changed

The `dfilt` `coefficients` method has been changed to the `coeffs` method, which returns a structure. See the `Methods` section of `dfilt` for information.

Compatibility Considerations

You should update any code that uses the `coefficients` method to use the new `coeffs` method and its returned structure.

dfilt Filter States Changed to Use States Property

You cannot pass filter states (initial and final conditions) via the `dfilt` `filter` method. You must use the `states` property. See `dfilt` for more information.

Compatibility Considerations

You should update any code that passed filter states via the `dfilt` `filter` method to use the new `states` property.

Spectral Analysis Changes

spectrogram Function Replaces specgram

`spectrogram` has been added to replace the grandfathered `specgram` function. If you use this function with no outputs, a surface plot is displayed, instead of an image.

Compatibility Considerations

You should update any code that references `specgram` to use the new `spectrogram` function. `spectrogram` uses different default values than `specgram` and the order of the inputs has changed.

Spectral Analysis Functions Inputs Changed

`pwelch` (and the other spectrum analysis functions) no longer accept `'half'` or `'whole'`. You must use `'onesided'` or `'twosided'` to indicate the type of analysis you want.

Compatibility Considerations

You should update any code that uses `'half'` or `'whole'` with spectral analysis functions and instead use `'onesided'` or `'twosided'`, respectively.

PSD Objects and Function Output Plots Changed

The following functions and methods now generate standard MATLAB plots, instead of launching an interactive plot. Refer to the MATLAB documentation for information on plots.

- `dspdata` `plot` method
- `spectrum` `psd`, `pseudospectrum`, and `msspectrum` methods

- `pburg`
- `pcov`
- `periodogram`
- `pmcov`
- `pmtm`
- `pwelch`
- `pyulear`

Other Changes

gaussfir Function Replaces firgauss

`gaussfir` has been added to replace the grandfathered `firgauss` function. `gaussfir` uses parameters that are common to communications systems.

Compatibility Considerations

You should update any code that references `firgauss` to use the new `gaussfir`.

firpm and cfirpm Inputs Changed

The `firpm` and `cfirpm` functions now take function handles as inputs instead of strings.

Compatibility Considerations

You should update any code that uses `firpm` or `cfirpm` so that it will work correctly with function handle inputs instead of string input.

New Demos

Signal Processing Toolbox demos have been reorganized and a new demo on the analysis of a numerically controlled oscillator (NCO) has been added.

Filter Wizard Product Dependency Removed

The Filter Wizard no longer requires Filter Design Toolbox software. You can use the Filter Wizard if you have Signal Processing Toolbox software and Simulink installed. If you have the Filter Design Toolbox software installed, more options are available. See `dspfwiz` for more information.