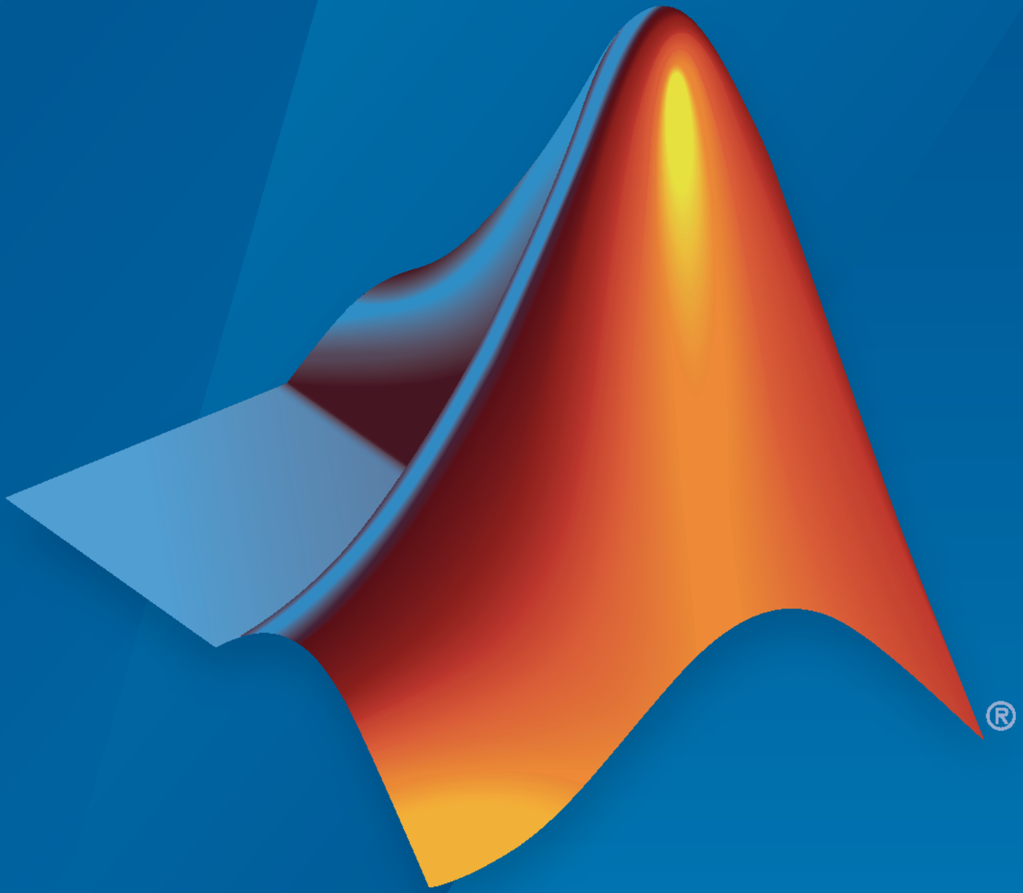
Wavelet Toolbox™ Release Notes



MATLAB®

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

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

Wavelet Toolbox™ Release Notes

Contents

R2023a

Wavelet Signal Analyzer App: Visualize and compress signals using the nondecimated discrete wavelet transform **1-2**

Wavelet Image Analyzer App: Visualize and synthesize wavelet decompositions of images **1-2**

waverec2 Function: Apply gains to lowpass coefficients and wavelet coefficient subbands **1-2**

Wavelet Scattering: Gather properties from the GPU **1-3**

New AI examples: Signal classification and hardware deployment **1-3**

Maximal Overlap Discrete Wavelet Packet Transforms: Analyze single-precision data **1-3**

dwtleader Function: Analyze single-precision data **1-3**

GPU Computing: Accelerate wavelet functions on your GPU **1-3**

C/C++ Code Generation: Automatically generate code for wavelet functions **1-3**

Functionality being removed or changed **1-4**
Wavelet Analyzer App has been removed from the MATLAB Apps tab and will be removed in R2023b **1-4**

R2022b

New Wavelet Scattering block: Model wavelet scattering network in Simulink **2-2**

dlcwt Function: Compute continuous wavelet transforms of deep learning arrays **2-2**

Deep Learning: Continuous wavelet transform layer **2-2**

Deep Learning: Maximal overlap discrete wavelet transform layer **2-3**

Orthogonal Wavelets: New families **2-3**

Wavelet Entropy: Updated and enhanced control of entropy parameters and wavelet decomposition **2-4**

isorthwfb and isbiorthwfb Functions: Test wavelet filter banks for perfect reconstruction **2-5**

Deep learning and differentiable signal processing examples **2-5**

Maximal Overlap Discrete Wavelet Transform: Time align wavelet and scaling coefficients with a signal **2-6**

C/C++ Code Generation: Automatically generate code for wavelet functions **2-6**

Functionality being removed or changed **2-6**
Wavelet Analyzer App will be removed **2-6**
dlmodwt accepts empty filter pair **2-6**

R2022a

Wavelet Time-Frequency Analyzer App: Visualize scalograms **3-2**

Signal Multiresolution Analyzer App: Support for additional decomposition methods **3-2**

sensingDictionary: Basis pursuit and matching pursuit for sparse signal recovery for 1-D signals **3-2**

dlmodwt Function: Compute MRA of deep learning arrays using the maximal overlap discrete wavelet transform **3-2**

Continuous Wavelet Transform: Invert using approximate synthesis filters **3-3**

Continuous Wavelet Transform: Enhanced bump wavelet support **3-3**

Continuous Wavelet Transform: Morse wavelet parameters and voices per octave enhancements **3-3**

CWT Filter Bank: Obtain lowpass filter frequency responses **3-3**

C/C++ Code Generation: Automatically generate code for the inverse continuous wavelet transform **3-3**

GPU Computing: Accelerate maximal overlap discrete wavelet transform on your GPU **3-3**

Deep Learning Example: Use wavelet scattering to develop an alert system for predictive maintenance 3-3

Sparse Signal Recovery Example: Use pursuit algorithms to recover data and remove impulse noise 3-4

Functionality being removed or changed 3-4
icwt syntax has changed 3-4
Data type of wavelet and scaling coefficients must match for icwt 3-4
wmpdictionary will be removed 3-5
wmpalg no longer supports plotting 3-7
wmpalg is no longer recommended 3-7
Some tools in the Wavelet Analyzer App have been removed 3-7

R2021b

Tunable Q-Factor Wavelet Transform: Specify your own Q-factor 4-2

2-D Lifting: Analyze SSCB data using lifting 4-2

Laurent Polynomials and Laurent Matrices: Operate on Laurent functions and study liftingScheme properties 4-3

MATLAB Online support for Wavelet Signal Denoiser 4-3

Signal Multiresolution Analyzer App: Analyze single-precision data 4-3

Denoising: Denoise signals using wavelet methods with Signal Analyzer 4-3

C/C++ Code Generation: Automatically generate code for wavelet functions 4-3

Machine Learning and Deep Learning Examples: Use wavelet-derived features for classification and fault detection 4-3

Wavelet Packets Example: Remove harmonic interference components from a signal 4-4

Functionality being removed or changed 4-4
Some Laurent and lifting functions will be removed 4-4
Lifting Function Syntax Changes 4-4
CustomLowpassFilter name-value argument in liftingScheme must be a cell array 4-6

R2021a

1-D Lifting: Analyze signals using lifting **5-2**

Wavelet Time Scattering: Accelerate and deploy automatic feature extraction of T×C×B data **5-2**

Signal Multiresolution Analyzer App: Performance improvements and MATLAB Online support **5-3**

Discrete Decimated Wavelet Analysis: Obtain wavelet transforms for complex-valued data **5-3**

Maximal Overlap Discrete Wavelet Transform: Obtain MODWT of multichannel signals **5-3**

Haar Transform: Obtain Haar transform of SSCB data **5-3**

GPU Computing: Accelerate wavelet functions on your GPU **5-3**

GPU Code Generation: Automatically generate GPU code for wavelet functions **5-3**

C/C++ Code Generation: Automatically generate code for wavelet functions **5-3**

Deep Learning Example: Perform modulation classification using wavelet-derived features and deploy onto hardware **5-4**

Functionality being removed or changed **5-4**
- 1-D Lifting Scheme Changes **5-4**
- Wavelet Time Scattering: waveletScattering property Decimate has been removed **5-6**
- Wavelet Time Scattering: featureMatrix function syntax will be deprecated **5-6**
- Wavelet Time Scattering: Highest center frequency calculated using geometric mean **5-6**

R2020b

Empirical Wavelet Transform: Perform adaptive signal decomposition using fully-automated spectrum segmentation **6-2**

CWT Marginals: Obtain and visualize time-averaged and scale-averaged wavelet spectrum **6-2**

Deep Learning Examples: Classify signals using wavelet-derived features and deploy onto hardware **6-2**

GPU Computing: Accelerate wavelet functions on your GPU 6-3

GPU Code Generation: Automatically generate GPU code for wavelet functions 6-3

C/C++ Code Generation: Automatically generate code for wavelet functions 6-3

R2020a

GPU Computing: Accelerate continuous wavelet transform on your GPU 7-2

Time-Frequency Analysis: Use variational mode decomposition to extract intrinsic modes 7-2

1-D Multisignal Discrete Wavelet Packet Transforms: Automatically perform wavelet packet analysis of multichannel signals 7-2

Kingsbury Q-shift Dual-Tree Complex Wavelet Transforms: Perform shift-invariant and directionally sensitive discrete multiresolution analysis with minimal redundancy 7-2

New Examples: Hands-on introductions to continuous wavelet analysis and multiresolution analysis 7-3

GPU Computing: Accelerate Wigner-Ville distribution 7-3

GPU Code Generation: Generate single precision code for cwt 7-3

C/C++ Code Generation: Generate single precision code for cwtfilterbank 7-3

wcoherence Function: Compute wavelet coherence over user-specified frequency or period range 7-4

C/C++ Code Generation: Automatically generate code for discrete wavelet analysis, time-frequency analysis, denoising, and multiscale variance estimation 7-4

Functionality being removed or changed 7-4

'NumOctaves' name-value pair argument in wcoherence will be removed 7-4

Wavelet decomposition types realdt and cplxdt for dddtree and dddtree2 functions will be removed 7-5

Some tools in the Wavelet Analyzer App have been removed 7-5

R2019b

Shearlets: Generate sparse representations of images automatically for deep learning and image processing . . . 8-2

Time-Frequency Gallery: Examine features and limitations of time-frequency analysis methods . . . 8-2

GPU Computing: Accelerate automatic feature extraction using wavelet scattering on GPUs . . . 8-2

Machine and Deep Learning Examples: Classify signals using wavelet-derived features . . . 8-2

C/C++ Code Generation: Automatically generate code for multisignal discrete wavelet analysis . . . 8-3

Functionality being removed . . . 8-3
Some tools in the Wavelet Analyzer app will be removed . . . **8-3**

R2019a

Wavelet Scattering for Images: Generate compact invariant feature representations automatically for image classification . . . 9-2

Image Denoising: Automatically denoise images while preserving sharp features . . . 9-2

GPU and C/C++ Code Generation: Automatically generate GPU or C/C++ code for the continuous wavelet transform . . . 9-2

Wavelet Scattering Examples: Classify images and time series using wavelet scattering and deep learning . . . 9-2

1-D Wavelet Time Scattering: Enable variable downsampling of coefficients . . . 9-3

Functionality being removed or changed . . . 9-3
waveletScattering property Decimate will be removed . . . **9-3**

R2018b

Wavelet Scattering: Generate invariant representations of time series data automatically for classification and regression . . . 10-2

Signal Multiresolution Analyzer App: Visualize and synthesize wavelet and empirical mode decompositions of signals 10-2

Time-Frequency Analysis: Analyze signals using the Wigner-Ville distribution 10-2

Scattering Examples: Classify signals using wavelet time scattering and machine learning 10-2

Signal Labeling: Define labels and create sets of labeled signals 10-3

Functionality being removed or changed 10-3

R2018a

Filter Banks for Continuous Wavelet Transform: Create, visualize, and use filter banks for time-frequency analysis 11-2

Filter Banks for Discrete Wavelet Transform: Explore time-frequency characteristics of wavelets and scaling functions 11-2

Constant-Q Transform: Perform adaptive time-frequency analysis using nonstationary Gabor frames 11-2

Scalogram View: Detect transients and perform time-frequency analysis with the Signal Analyzer App 11-3

Empirical Mode Decomposition and Hilbert-Huang Transform: Perform data-adaptive time-frequency analysis of nonlinear and nonstationary processes 11-3

Continuous Wavelet Transform: Improved control of time-frequency parameters and timetable support 11-3

Machine and Deep Learning Examples: Classify signals using wavelet-based feature extraction and deep learning 11-3

Functionality Being Removed or Changed 11-3

R2017b

Wavelet Signal Denoiser App: Visualize and denoise time-series data 12-2

Wavelet Denoising: Denoise time-series data with improved automatic selection of input parameters 12-2

Continuous Wavelet Transform: Perform time-frequency analysis of complex-valued time-series data 12-2

Functionality Being Removed or Changed 12-2

R2017a

Multiscale Local Polynomial Transform: Perform smoothing of nonuniformly sampled signals 13-2

3-D Dual-Tree Complex Wavelet Transform: Perform directionally selective wavelet analysis of volumetric data 13-2

Length-16 Q-shift filter 13-2

Modified treatment of boundary conditions in Wavelet Analyzer 13-2

R2016b

Code Generation: Generate C code for DWT, wavelet packets, and denoising using MATLAB Coder 14-2

Continuous Wavelet Transform: Analyze signals with improved automatic selection of wavelet and scales 14-2

Morse Wavelets: Family of analytic wavelets for continuous wavelet analysis 14-3

Wavelet Leaders and Wavelet Transform Modulus Maxima: Characterize fractal data and singularities 14-3

Haar Lifting Transforms: Perform multiresolution analysis of images and multichannel signals 14-3

Compression data files default to uint64 data 14-3

Wavelet Design and Analysis App renamed to Wavelet Analyzer 14-3

wavedemo function removed 14-3

Functionality being removed or changed 14-4

R2016a

Maximal Overlap Discrete Wavelet Packet Transform: Perform nondecimated wavelet packet analysis on arbitrary-length signals 15-2

Wavelet Synchrosqueezing: Sharpen time-frequency estimates and extract signal modes 15-2

Wavelet Coherence: Compare time-varying frequency content between signals 15-2

Fejer-Korovkin filter with 18 coefficients 15-2

Maximum derivative output of gauswavf and cgauwavf 15-2

Functionality being removed or changed 15-3

R2015b

Maximal Overlap Discrete Wavelet Transform: Perform nondecimated analysis on arbitrary-length signals and obtain multiscale variance and correlation estimates 16-2

Frequency-Localized Wavelets: Perform more frequency-localized discrete and continuous wavelet analysis 16-2

Time-Frequency Analysis: Convert scale to frequency to interpret the continuous wavelet transform as a time-frequency transform 16-2

Application examples: Analyze financial data and physiologic signals using wavelets 16-2

R2015a

Functionality being removed or changed 17-2

R2014b

Bug Fixes

R2014a

Dual-Tree Wavelet Transforms 19-2

R2013b

Two-Dimensional Continuous Wavelet Transform (2-D CWT) 20-2

Dual-Tree Transforms and Double-Density Transforms 20-2

R2013a

Bug Fixes

R2012b

Bug Fixes

R2012a

Matching Pursuit 23-2

R2011b

Fourier Transform Based Continuous Wavelet Transform GUI 24-2

Inverse Continuous Wavelet Transform Using Linear Scales 24-2

MATLAB Code Generation Support for Denoising and Compression GUIs 24-2

Signal Reconstruction from Continuous Wavelet Transform Coefficients Demo 24-2

Changes in Fourier Transform Based Continuous Wavelet Transform Defaults for Derivative of Gaussian (DOG) and Paul Wavelets 24-2

R2011a

Inverse Continuous Wavelet Transform 25-2

FFT-based Continuous Wavelet Transform 25-2

Pattern-adapted Wavelets for Signal Detection Demo 25-2

R2010b

Cone of Influence for Continuous Wavelet Transform 26-2

Wavelet Cross Spectrum and Coherence 26-2

Wavelet Packet Spectrum 26-2

Natural and Frequency Ordering of Wavelet Packet Terminal Nodes . . . 26-2

Image and Signal Approximation Quality Metrics 26-2

R2010a

3-D Discrete Wavelet Transform 27-2

Nondecimated Wavelet Transform 27-2

New Denoising Function 27-2

R2009b

Bug Fixes

R2009a

New Demos 29-2

R2008b

True Image Compression Support 30-2

New Demo 30-2

R2008a

True Color Images Support 31-2

New Extension Modes for Continuous Wavelets 31-2

New Norms Calculation 31-2

Wavelet Families Display 31-2

Single Data Type Support 31-2

New Demos 31-2

R2007b

Importing and Exporting between GUIs and Workspace . . . 32-2

Scalograms for Continuous Wavelet Transforms 32-2

Constructing Clusters from Hierarchical Cluster Trees . . . 32-2

R2007a

1D Multisignal Analysis, Compression, and Denoising Added . . . 33-2

1D Multisignal Wavelet and Clustering Added 33-2

Wavelet 1D Multisignal Analysis GUI Added 33-2

R2006b

Multivariate De-noising Added 34-2

Multiscale Principal Component Analysis Added 34-2

New Demos 34-2

R2006a

No New Features or Changes

R14SP3

No New Features or Changes

R14SP2

No New Features or Changes

R2023a

Version: 6.3

New Features

Bug Fixes

Compatibility Considerations

Wavelet Signal Analyzer App: Visualize and compress signals using the nondecimated discrete wavelet transform

The **Wavelet Signal Analyzer** app enables visualization, analysis, and compression of 1-D signals using the nondecimated discrete wavelet transform. The app plots the decomposition of the signal and its corresponding reconstruction. The app also shows statistics of the decomposition, including the approximate frequency band of each component. With the **Wavelet Signal Analyzer** app, you can:

- Access all single-channel, real- and complex-valued 1-D signals in the MATLAB® workspace
- Compare reconstructions from different analyses by varying the wavelet or the decomposition level
- Visualize the time-aligned coefficients
- Extend the signal periodically or by reflection before computing the wavelet transform
- Apply a threshold to the wavelet coefficients to compress the signal
- Plot the energy for all decomposition levels and display histograms of the original and compressed coefficients at a specific level
- Export decomposition coefficients, compressed coefficients, and compressed signals to the MATLAB workspace
- Generate MATLAB scripts to reproduce results in your workspace

The **Wavelet Signal Analyzer** app supports single- and double-precision data.

Wavelet Image Analyzer App: Visualize and synthesize wavelet decompositions of images

The **Wavelet Image Analyzer** app enables you to visualize the discrete wavelet decomposition of images. With the **Wavelet Image Analyzer** app, you can:

- Import images from your MATLAB workspace or from a file
- Specify the orthogonal or biorthogonal wavelet to use in the decomposition
- Change the decomposition level
- Reconstruct an image with the wavelet coefficient subbands you specify
- Easily compare different reconstructions
- Export the image decompositions to your MATLAB workspace
- Generate MATLAB scripts to reproduce results in your workspace

The **Wavelet Image Analyzer** app supports grayscale and RGB images.

waverec2 Function: Apply gains to lowpass coefficients and wavelet coefficient subbands

You can now use the `waverec2` function to set the gain for the lowpass coefficients and the wavelet coefficient subbands to use in image reconstruction.

Wavelet Scattering: Gather properties from the GPU

You can use the new `waveletScattering` object function `gather` to collect all the properties of a `waveletScattering` object stored in GPU memory into your workspace.

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

New AI examples: Signal classification and hardware deployment

This release introduces examples that employ wavelet techniques in AI applications.

- “Time-Frequency Feature Embedding with Deep Metric Learning” shows how to use deep metric learning with a supervised contrastive loss function to construct feature embeddings based on a time-frequency analysis of electroencephaligraphic (EEG) signals.
- “Time-Frequency Convolutional Network for EEG Data Classification” shows how to classify EEG time series from persons with and without epilepsy using a time-frequency convolutional network.
- “Deep Learning Code Generation on ARM for Fault Detection Using Wavelet Scattering and Recurrent Neural Networks” demonstrates code generation for acoustic-based machine fault detection using a wavelet scattering network paired with a recurrent neural network.

Maximal Overlap Discrete Wavelet Packet Transforms: Analyze single-precision data

The functions `modwpt`, `imodwpt`, and `modwptdetails` now support single-precision data.

dwtleader Function: Analyze single-precision data

The `dwtleader` function now supports single-precision data.

GPU Computing: Accelerate wavelet functions on your GPU

These Wavelet Toolbox functions now support `gpuArray` objects:

- `modwpt`
- `wentropy`

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

C/C++ Code Generation: Automatically generate code for wavelet functions

These Wavelet Toolbox functions now support C/C++ code generation:

- `dwtleader`
- `wentropy`

The “Generate and Deploy Optimized Code for Wavelet Time Scattering on ARM Targets” example shows how to generate optimized C++ code for the `waveletScattering` object that can be deployed on a Raspberry Pi® target (ARM®-based device).

You must have MATLAB Coder™ to generate C/C++ code.

Functionality being removed or changed

Wavelet Analyzer App has been removed from the MATLAB Apps tab and will be removed in R2023b

Warns

You can no longer launch **Wavelet Analyzer** app from the MATLAB **Apps** tab. In R2023b, **Wavelet Analyzer** will be removed. Use one of these apps instead:

- **Signal Multiresolution Analyzer** — Perform signal multiresolution analysis using wavelet and data-adaptive techniques.
- **Wavelet Image Analyzer** — Analyze images using the discrete wavelet transform.
- **Wavelet Signal Analyzer** — Analyze and compress signals using the nondecimated discrete wavelet transform.
- **Wavelet Signal Denoiser** — Analyze and denoise signals using the discrete wavelet transform.
- **Wavelet Time-Frequency Analyzer** — Perform time-frequency analysis of signals using the continuous wavelet transform.

R2022b

Version: 6.2

New Features

Bug Fixes

Compatibility Considerations

New Wavelet Scattering block: Model wavelet scattering network in Simulink

This release introduces the first Simulink® block that supports designing machine learning systems for signal processing applications.

The new Wavelet Scattering (DSP System Toolbox) block creates a framework for wavelet time scattering in the Simulink environment. Use this block to derive low-variance features from real-valued data and then use those features in machine learning and deep learning applications. The block uses predefined wavelet filters to compute the scalogram and applies an averaging filter to the scalogram for feature extraction. For more information, see Wavelet Scattering.

The Wavelet Scattering block requires DSP System Toolbox™.

dlcwt Function: Compute continuous wavelet transforms of deep learning arrays

This release introduces the `dlcwt` function, which computes the continuous wavelet transform (CWT) of `dlarray` (Deep Learning Toolbox) objects using a CWT filter bank that you create from `cwtfilterbank`. The `dlcwt` function outputs the transforms as `dlarray` objects that enable automatic differentiation. You can use the objects in custom training loops or inside a custom layer. You must have Deep Learning Toolbox™ to use the `dlcwt` function.

The new `cwtfilters2array` function converts the filter bank you create using `cwtfilterbank` into a reduced-weight tensor for deep learning. `cwtfilters2array` allows you to specify a threshold for controlling the number of filter values. By thresholding, you can significantly reduce the number of learnable parameters in the filter bank. You can use the new `array2cwtfilters` function to convert the tensor into a filter bank matrix.

`dlarray` inputs must be compatible with the channel-by-batch-by-time (CBT) format. The underlying data is real-valued and can be in single- or double-precision. For a list of functions that support `dlarray` objects, see List of Functions with dlarray Support (Deep Learning Toolbox).

Deep Learning: Continuous wavelet transform layer

This release introduces the `cwtLayer` object, which computes the continuous wavelet transform (CWT) within a deep learning network. The learnable layer can output the CWT magnitude, CWT squared magnitude, and the CWT real and imaginary parts concatenated along the channel dimension. In the `cwtLayer` object, you can:

- Use the analytic Morse, analytic Morlet, or bump wavelet
- Choose to include the lowpass (scaling) filter in the CWT
- Choose whether to update the weights with training
- Specify the number of voices (wavelets) per octave
- Specify the frequency limits for the CWT

The `cwtLayer` 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 `cwtLayer` object. For a list of available layers, see List of Deep Learning Layers (Deep Learning Toolbox).

Deep Learning: Maximal overlap discrete wavelet transform layer

This release introduces the `modwtLayer` object, which computes the maximal overlap discrete wavelet transform (MODWT) and MODWT multiresolution analysis (MRA) within a deep learning network. `modwtLayer` parameters you can specify include:

- Which orthogonal wavelet to use in the MODWT
- Lowpass and highpass filters that correspond to an orthogonal wavelet
- Which resolution levels (scales) to output
- Whether to update the weights with training
- Whether to include the lowpass (scaling) coefficients in the MODWT, or the level smooth in the MODWTMRA
- Choice of boundary conditions

The `modwtLayer` 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 `modwtLayer` object. For a list of available layers, see List of Deep Learning Layers (Deep Learning Toolbox).

Orthogonal Wavelets: New families

The Wavelet Toolbox now supports for five new orthogonal wavelet families. You can use `wfilters` to obtain the filters associated with the new families.

Family	Family Short Name	Additional Information
Best-localized Daubechies	`"bl"`	To obtain the scaling filter, you can also use the new `blscalf` function. You can then use `orthfilt` to obtain the four filters associated with the wavelet.
Beylkin	`"beyl"`	
Vaidyanathan	`"vaid"`	
Han real orthogonal scaling filters with sum and linear-phase moments	`"han"`	To obtain the scaling filter, you can also use the new `hanscalf` function. You can then use `orthfilt` to obtain the four filters associated with the wavelet.

Family	Family Short Name	Additional Information
Morris minimum bandwidth	`"mb"`	To obtain the scaling filter, you can also use the new `mbscalf` function. You can then use `orthfilt` to obtain the four filters associated with the wavelet.

To obtain information on each family, enter `waveinfo(sn)`, where `sn` is the wavelet family short name, at the MATLAB command prompt. For example, `waveinfo("bl")`.

You can specify the new wavelet filters in all discrete wavelet and wavelet packet command-line functions using the short name with a valid filter number. For example, `wavedec(data,N,"bl7")` or `modwt(data,"bl7")`. The **Signal Multiresolution Analyzer** and **Wavelet Signal Denoiser** apps do not support the new families currently.

The functions `blscalf`, `hanscalf`, and `mbscalf` support C/C++ code generation. You must have MATLAB Coder to generate C/C++ code.

Wavelet Entropy: Updated and enhanced control of entropy parameters and wavelet decomposition

This release provides an updated and enhanced version of the `wentropy` function. The function supports Shannon, Renyi, and Tsallis entropies. The function also provides a simple interface to a variety of entropy- and wavelet-related parameters. You can use `wentropy` to extract features for machine learning applications. The `wentropy` function works on raw data as well as wavelet coefficients. To obtain wavelet entropy estimates, you can use the default parameter values or adjust the values for your use case. With `wentropy`, you can:

- Specify the exponent used in the Renyi and Tsallis entropies.
- Obtain wavelet entropy estimates normalized by scale or across scales.
- Scale the wavelet entropy by the factor corresponding to a uniform distribution for the specified entropy.
- Choose the type of wavelet transform to perform: discrete wavelet, discrete wavelet packet, maximal overlap discrete wavelet, or maximal overlap discrete wavelet packet.
- Specify the wavelet and level of the wavelet transform.
- You can threshold the wavelet or wavelet packet coefficients. This prevents the function from treating the coefficients with nonsignificant energy as a sequence with high entropy.
- Obtain the relative wavelet energies.

The `wentropy` function supports real-valued single- or double-precision data.

Compatibility Considerations

The syntax used in the old version of `wentropy` continues to work, but is no longer recommended. The old version provides you minimal control over how to estimate the entropy. The `wentropy` function automatically determines from the input syntax which version to use.

You can specify the Shannon entropy in both versions of `wentropy`. However, because the old version makes no assumptions about the input data, reproducing the same results as the new version can require extensive effort.

Old Version

```
load wecg
n = numel(wecg);
lev = 3;
wt = modwt(wecg,lev);
energy = sum(abs(wt).^2,2);
wt2 = abs(wt)./sqrt(energy);
ent = zeros(lev+1,1);
for k=1:lev+1
    ent(k) = wentropy(wt2(k,:),'shannon')/log(n);
end
ent

ent =

    0.3925
    0.6512
    0.6985
    0.9329
```

New Version

```
load wecg
ent = wentropy(wecg,Level=3)

ent =

    0.3925
    0.6512
    0.6985
    0.9329
```

isorthwfb and isbiorthwfb Functions: Test wavelet filter banks for perfect reconstruction

This release introduces two functions you can use to evaluate two-channel filter banks formed by filters you specify:

- `isorthwfb` — Test if a two-channel filter bank satisfies the necessary and sufficient conditions to be an orthogonal perfect reconstruction wavelet filter bank.
- `isbiorthwfb` — Test if a two-channel filter bank satisfies the necessary and sufficient conditions to be a biorthogonal perfect reconstruction wavelet filter bank.

Each test consists of a number of orthogonality and biorthogonality checks. You can also adjust the tolerance of the checks.

The functions `isorthwfb` and `isbiorthwfb` support C/C++ code generation. You must have MATLAB Coder to generate C/C++ code.

Deep learning and differentiable signal processing examples

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

- Signal Recovery with Differentiable Scalograms and Spectrograms shows how to use differentiable scalograms and spectrograms to recover a time-domain signal without the need for phase information.
- Human Health Monitoring Using Continuous Wave Radar and Deep Learning shows how to reconstruct electrocardiogram signals using a bidirectional long short-term memory network and wavelet multiresolution analysis.

Maximal Overlap Discrete Wavelet Transform: Time align wavelet and scaling coefficients with a signal

The `modwt` function can now output approximately time-aligned, or shifted, wavelet and scaling coefficients. Shifting the coefficients is useful if you want to align features in the signal with the coefficients. To correct for the delay in the wavelet and scaling filters, the function circularly shifts the wavelet coefficients (at all levels) and scaling coefficients.

C/C++ Code Generation: Automatically generate code for wavelet functions

The functions `orthfilt` and `biorfilt` now support C/C++ code generation. You must have MATLAB Coder to generate C/C++ code.

Functionality being removed or changed

Wavelet Analyzer App will be removed

Warns

The **Wavelet Analyzer** app is no longer recommended and will be removed in a future release.

- For time-frequency analysis, use the **Wavelet Time-Frequency Analyzer** app.
- For wavelet signal denoising, use the **Wavelet Signal Denoiser** app.
- For signal multiresolution analysis, use the **Signal Multiresolution Analyzer** app.

dlmodwt accepts empty filter pair

Behavior change

You can now specify a pair of empty inputs for the lowpass and highpass filters. The `dlmodwt` function continues to generate an error if one filter input is empty and the other filter input is nonempty.

Functionality	Previous Behavior	New Behavior
`w = dlmodwt(x,[],[])`	Errors	`w = dlmodwt(x,[],[])` is equivalent to `w = dlmodwt(x)`
`w = dlmodwt(x,[],[],level)`	Errors	`w = dlmodwt(x,[],[],level)` is equivalent to `w = dlmodwt(x,Lo,Hi,level)`, where `[~,~,Lo,Hi] = wfilters('sym4')`

R2022a

Version: 6.1

New Features

Bug Fixes

Compatibility Considerations

Wavelet Time-Frequency Analyzer App: Visualize scalograms

The **Wavelet Time-Frequency Analyzer** enables computation and visualization of the scalogram. The app provides an initial display of the scalogram using `cwt` with default settings. You can modify the settings, such as choosing a different wavelet, or adjusting the parameters for Morse wavelets. With the app, you can access all 1-D signals in your MATLAB workspace. You can import multiple signals simultaneously. You can export the scalogram and generate a MATLAB script to reproduce the wavelet analysis in your workspace. The signal can be:

- A real- or complex-valued vector.
- A single-variable regularly sampled timetable.
- Single or double precision.

Signal Multiresolution Analyzer App: Support for additional decomposition methods

With the **Signal Multiresolution Analyzer** app, you can now decompose signals using these methods:

- Empirical wavelet transform (see `ewt`)
- Variational mode decomposition (see `vmd`)
- Tunable Q-factor wavelet transform (see `tqwt`)

sensingDictionary: Basis pursuit and matching pursuit for sparse signal recovery for 1-D signals

With the new `sensingDictionary` function, you create a sensing dictionary for sparse approximations of 1-D signals. The function `sensingDictionary` provides built-in support for a variety of options, including wavelet, discrete cosine transform, Fourier, and Gaussian and Bernoulli random distributions. You can also create and use custom dictionaries.

You can apply your dictionary for signal sparse recovery using matching pursuit or basis pursuit. Additionally, the basis pursuit algorithm supports custom dictionaries created using tall arrays. You can apply these custom dictionaries to tall array inputs.

The pursuit algorithms work with real- or complex-valued signals having single or double precision.

dlmodwt Function: Compute MRA of deep learning arrays using the maximal overlap discrete wavelet transform

This release introduces the `dlmodwt` function, which computes the maximal overlap discrete wavelet transform (MODWT) and MODWT multiresolution analysis (MRA) of `dlarray` (Deep Learning Toolbox) objects. The function outputs the wavelet analysis as `dlarray` objects that enable automatic differentiation and can be used in custom training loops or inside a custom layer.

`dlarray` inputs must be compatible with channel-by-batch-by-time (CBT) format. The underlying data can be real- or complex-valued, in single- or double-precision.

Continuous Wavelet Transform: Invert using approximate synthesis filters

You can now use `icwt` to invert the CWT using the approximate synthesis filters associated with the analysis filter bank obtained from `cwtfilterbank` or `cwt`. The filter bank is an optional output of `cwt`.

Continuous Wavelet Transform: Enhanced bump wavelet support

You can now obtain the scaling coefficients for the CWT when using `cwt` with the bump wavelet.

Continuous Wavelet Transform: Morse wavelet parameters and voices per octave enhancements

With `cwt`, `icwt`, `cwtfreqbounds`, and `cwtfilterbank`, you can now:

- Set an odd number of voices per octave
- Set the Morse wavelet time-bandwidth product equal to the symmetry parameter

CWT Filter Bank: Obtain lowpass filter frequency responses

You can now use the `freqz` method to obtain the lowpass or scaling filter frequency response from the CWT filter bank you created using `cwtfilterbank`. You can also obtain the full two-sided frequency responses of the filter bank.

C/C++ Code Generation: Automatically generate code for the inverse continuous wavelet transform

The `icwt` function now supports C/C++ code generation. You must have MATLAB Coder to generate C/C++ code.

GPU Computing: Accelerate maximal overlap discrete wavelet transform on your GPU

The `modwt`, `imodwt`, and `modwtmra` functions now support `gpuArray` objects. This support requires Parallel Computing Toolbox. For more information, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox).

Deep Learning Example: Use wavelet scattering to develop an alert system for predictive maintenance

This release introduces an example, Detect Anomalies Using Wavelet Scattering with Autoencoders, that shows how to use the wavelet scattering transform with both LSTM and convolutional networks to develop an alert system for predictive maintenance. The example compares wavelet scattering transform+deep network and raw data+deep network approaches.

Sparse Signal Recovery Example: Use pursuit algorithms to recover data and remove impulse noise

This release introduces an example, Signal Deconvolution and Impulse Denoising Using Pursuit Methods, that shows how to:

- Use orthogonal matching pursuit to recover ground profile information from noisy seismic signal measurements.
- Use basis pursuit to remove impulse noise from power system current measurements.

Functionality being removed or changed

icwt syntax has changed

Behavior change

The behavior of `icwt` has changed. If you invert the CWT over a specified frequency range or range of periods, you must precede those inputs either by a wavelet name or an empty input for the default Morse wavelet.

You do not have to specify the default Morse wavelet if you are only setting name-value arguments. For example, `xrec = icwt(cfs,TimeBandwidth=40)`.

Functionality	Result	Use Instead
`xrec = icwt(cfs,f,freqrange)`	Errors	`xrec = icwt(cfs,[],f,freqrange)` or `xrec = icwt(cfs,"morse",f,freqrange)`
`xrec = icwt(cfs,f,freqrange,Name=Value)`	Errors	`xrec = icwt(cfs,[],f,freqrange,Name=Value)` or `xrec = icwt(cfs,"morse",f,freqrange,Name=Value)`
`xrec = icwt(cfs,period,periodrange)`	Errors	`xrec = icwt(cfs,[],period,periodrange)` or `xrec = icwt(cfs,"morse",period,periodrange)`
`xrec = icwt(cfs,period,periodrange,Name=Value)`	Errors	`xrec = icwt(cfs,[],period,periodrange,Name=Value)` or `xrec = icwt(cfs,"morse",period,periodrange,Name=Value)`

Data type of wavelet and scaling coefficients must match for icwt

Behavior change

Starting this release, `icwt` requires that the wavelet and scaling coefficient inputs have the same data type. Both sets of coefficients must be either single or double precision.

Note that the wavelet and scaling coefficient outputs of `cwt` and the `wt` method of `cwtfilterbank` always have the same data type.

wmpdictionary will be removed
Still runs

The `wmpdictionary` function will be removed in a future release. Use `sensingDictionary` instead.

Functionality	Result	Use Instead	Compatibility Considerations

`MPDICT = wmpdictionary(N)`	Still runs	Execute these steps:
		1 Create a `sensingDictionary` using pre-built support for wavelet and DCT frames:
		`A1 = sensingDictionary('Size',N,...`
		`'Type',{'dwt','dct'},...`
		`'Name',{'sym4'},...`
		`'Level',[5]);`
		2 Create a custom `sensingDictionary`:
		`T = linspace(0,1,N)';`
		`K = 1:ceil(N/2);`
		`T1 = repmat(T,1,numel(K));`
		`K1 = repmat(K,numel(T),1);`
		`Amat = sin(2*pi*(K1.*T1));`
		`A2 = sensingDictionary('CustomDictionary',Amat);`
		3 Concatenate the results:
		`MPDICT = [A1 A2];`
			• `sensingDictionary` provides pre-built support for a variety of frames, including Fourier, Gaussian and Bernoulli random distributions, and Walsh code.
			• `sensingDictionary` does not currently support wavelet packet bases.

Functionality	**Result**	**Use Instead**	**Compatibility Considerations**

`MPDICT = wmpdictionary(N,'lstcpt',dtypes)`, where `dtypes` is a cell array of cell arrays with valid subdictionaries	Still runs	Each cell array in `dtypes` describes one subdictionary. To specify a subdictionary in `sensingDictionary`, use the `Type`, `Name`, and `Level` name-value arguments.
		
		For example:
		
		• Replace
		`mpdict = wmpdictionary(100,'lstcpt',{'dct','RnIden`
		with
		`D = sensingDictionary('Size',100,'Type',{'dct','ey`
		
		• Replace
		`mpdict = wmpdictionary(100,...`
		`'lstcpt',{{'db4',3},{'dct'}});`
		with
		`x = sensingDictionary('Size',100,...`
		`'Type',{'dwt','dct'},...`
		`'Name',{'db4'},...`
		`'Level',[3 0])`
			For the wavelet option, `sensingDictionary` and `wmpdictionary` behave differently.
			
			• `wmpdictionary` returns the wavelets at all levels and the scaling functions at the final level.
			• `sensingDictionary` returns the wavelets at only the final level.
			
			For example,
			`mpdict = wmpdictionary(100,'lstcpt',{{'db1',2}});`
			returns the scaling functions for level 2, the wavelets for level 2, and the wavelets for level 1, whereas
			`A = sensingDictionary('Size',100,'Type',{'dwt'},'Name',{'db1'},'Level',2);`
			only returns the wavelets at level 2.

Functionality	**Result**	**Use Instead**	**Compatibility Considerations**
`[~,NBVECT] = wmpdictionary(N)` or `[~,NBVECT] = wmpdictionary(N,'lstcpt')`	Still runs	`NBVECT` is the number of vectors in each subdictionary. The number of vectors in a subdictionary of a `sensingDictionary` object depends on the associated basis type. • For a non-random basis type, the number of vectors is `N`. • For a random basis type, the number of vectors is the column size you specified when you created the `sensingDictionary` object. • For a custom `sensingDictionary`, the number of vectors is the column size you specified when you created the `sensingDictionary` object.	You can also use the `subdict` method of `sensingDictionary` to extract the vectors.

wmpalg no longer supports plotting
Errors

The `wmpalg` function no longer supports the name-value arguments `stepplot` and `typeplot`. Remove all instances from your code. Instead, use MATLAB plotting commands.

wmpalg is no longer recommended
Still runs

The `wmpalg` function is no longer recommended. Instead, use `sensingDictionary` with `matchingPursuit` and `basisPursuit`.

Some tools in the Wavelet Analyzer App have been removed

These tools in the **Wavelet Analyzer** app have been removed.

Tools	Replacement
Continuous Wavelet 1-D	To visualize the scalogram, use the new **Wavelet Time-Frequency Analyzer** app or the `cwt` function. With the app, you can select the wavelet to use as well as adjust Morse wavelet parameters. The app also supports single-variable regularly sampled timetables and real- or complex-valued single- or double-precision data.
Complex Continuous Wavelet 1-D	Use the new **Wavelet Time-Frequency Analyzer** app or the `cwt` function. With the app, you can also export the scalogram and generate a script to reproduce the wavelet analysis to your workspace.

R2021b

Version: 6.0

New Features

Bug Fixes

Compatibility Considerations

Tunable Q-Factor Wavelet Transform: Specify your own Q-factor

With the new tunable Q-factor wavelet transform (TQWT) functions, `tqwt`, `itqwt`, and `tqwtmra`, you can perform discrete multiresolution analyses of signals using Q-factors you specify. Signal data can be a vector, a matrix, or a 3-D array. For matrices and 3-D arrays, the first dimension is interpreted as time. The Q-factor of a wavelet transform is the ratio of the center frequency to the bandwidth of the filters used to obtain the wavelet coefficients. Higher Q-factors produce narrower filters, which are more appropriate for analyzing oscillatory signals. For signals with transient components, you can specify lower Q-factors. The TQWT provides a Parseval frame decomposition where energy is partitioned among components.

Use the `tqwt` function to obtain decompositions of the input signals. The `tqwt` function supports time-by-channel-by-batch (T×C×B) inputs. With the `itqwt` function, you can invert the transform for perfect reconstruction. You can use `tqwtmra` to obtain the multiresolution analysis (MRA) from the decomposition. The `tqwt`, `itqwt`, and `tqwtmra` functions:

- Support real- and complex-valued data.
- Support single- and double-precision data.
- Support C/C++ code generation.
- Support GPU acceleration.

You must have MATLAB Coder to generate C/C++ code. You must have Parallel Computing Toolbox for `gpuArray` support. For more information, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox).

2-D Lifting: Analyze SSCB data using lifting

You can now use a lifting scheme object to analyze spatial-by-spatial-by-channel-by-batch (SSCB) data with the enhanced 2-D lifting analysis and synthesis functions `lwt2` and `ilwt2`. The enhanced functions:

- Support real- and complex-valued data.
- Support single- and double-precision data.
- Enable you to obtain integer-to-integer wavelet transforms.
- Support C/C++ code generation.

You must have MATLAB Coder to generate C/C++ code.

Compatibility Considerations

In previous releases, you used `lwt2` or `ilwt2` with a lifting scheme cell array. You used `liftwave` to create the cell array. Starting with R2021b, to perform 2-D lifting analysis or synthesis, you use `lwt2` and `ilwt2` either with default settings or by specifying a lifting scheme object that you create with `liftingScheme`.

Laurent Polynomials and Laurent Matrices: Operate on Laurent functions and study liftingScheme properties

With the new `laurentPolynomial` and `laurentMatrix` objects, you can perform mathematical and logical operations on Laurent polynomials and Laurent matrices. You can use the object functions to study characteristics of a `liftingScheme` created from a given set of filters.

The `laurentPolynomial`, `laurentMatrix`, and `liftingScheme` objects and object functions now all support C/C++ code generation. You must have MATLAB Coder to generate C/C++ code.

Compatibility Considerations

`laurpoly` and `laurmat` will be removed in a future release.

MATLAB Online support for Wavelet Signal Denoiser

Starting this release, the **Wavelet Signal Denoiser** app is now supported in MATLAB Online™.

Signal Multiresolution Analyzer App: Analyze single-precision data

The **Signal Multiresolution Analyzer** app now supports single-precision data.

Denoising: Denoise signals using wavelet methods with Signal Analyzer

The **Signal Analyzer** (Signal Processing Toolbox) app now enables wavelet denoising. Use this feature to interactively denoise signals in the app using wavelet methods. You must have a Signal Processing Toolbox™ license to use **Signal Analyzer**.

C/C++ Code Generation: Automatically generate code for wavelet functions

These Wavelet Toolbox functions now support C/C++ code generation:

- **Discrete Multiresolution Analysis** — `tqwt`, `itqwt`, and `tqwtmra`
- **Lifting** — `lwt`, `ilwt`, `lwtcoef`, `lwt2`, `ilwt2`, `lwtcoef2`, `liftingScheme`, and `ls2filt`
- **Laurent** — `laurentPolynomial`, `laurentMatrix`, `filters2lp`, and `liftfilt`

You must have MATLAB Coder to generate C/C++ code.

Machine Learning and Deep Learning Examples: Use wavelet-derived features for classification and fault detection

This release introduces examples of classification and fault detection using wavelet-derived features in machine learning and deep learning workflows.

- Anomaly Detection Using Autoencoder and Wavelets shows how to detect arc faults in a DC system using features extracted by the lifting wavelet transform.

- Air Compressor Fault Detection Using Wavelet Scattering shows how to classify faults in acoustic recordings of air compressors using a wavelet scattering network and a support vector machine.
- Fault Detection Using Wavelet Scattering and Recurrent Deep Networks shows how to classify faults in acoustic recordings of air compressors using a wavelet scattering network paired with a recurrent neural network.
- Parasite Classification Using Wavelet Scattering and Deep Learning shows how to classify parasitic infections in Giemsa stain images using wavelet image scattering and deep learning.

Wavelet Packets Example: Remove harmonic interference components from a signal

A new featured example, Wavelet Packet Harmonic Interference Removal, shows how to use wavelet packets to remove harmonic interference, or sinusoidal, components from a signal without adversely affecting the frequency content of the primary signal in the neighborhood of these harmonics.

Functionality being removed or changed

Some Laurent and lifting functions will be removed

Still runs

- `laurpoly` will be removed in a future release. Use `laurentPolynomial` instead.
- `laurmat` will be removed in a future release. Use `laurentMatrix` instead.
- `biorlift`, `cdflift`, `coiflift`, and `dblift` will be removed in a future release. Set the `Wavelet` property of `liftingScheme` instead.

Functionality	Result	Use Instead	Compatibility Considerations
`P = laurpoly(C,d)` and `P = laurpoly(C,'dmax',d)`	Still runs	`P = laurentPolynomial(Coefficients=C,MaxOrder=d)`	You can also create a lifting scheme associated with a pair of Laurent polynomials.
`P = laurpoly(C,'dmin',d)`	Still runs	`P = laurentPolynomial(Coefficients=C,MaxOrder=`*N*`+d-1)`, where *N* is the length of `C`.	
`M = laurmat(V)`	Still runs	`M = laurentMatrix(Elements=V)`	You can also perform mathematical operations on the matrices.

Lifting Function Syntax Changes

Behavior change

The syntax of 2-D lifting functions has changed. The new syntax uses name-value arguments. The `liftfilt` function syntax has also changed.

Functionality	Result	Use Instead	Compatibility Considerations
`[CA,CH,CV,CD] = lwt2(X,W)`	Errors	`[CA,CH,CV,CD] = lwt2(X,Wavelet=W)`	You can also obtain the lifting wavelet transform (LWT) using a lifting scheme by setting the `LiftingScheme` name-value argument.
`[CA,CH,CV,CD] = lwt2(X,W,LEVEL)`	Errors	`[CA,CH,CV,CD] = lwt2(X,Wavelet=W,Level=LEVEL)`	You can also specify the extension mode by setting the `Extension` name-value argument.
`X = ilwt2(CA,CH,CV,CD,W)`	Errors	`X = ilwt2(CA,CH,CV,CD,Wavelet=W)`	You can also set the `LiftingScheme` name-value argument to obtain the inverse LWT.
`X = ilwt2(CA,CH,CV,CD,W,LEVEL)`	Errors	`X = ilwt2(CA,CH,CV,CD,Wavelet=W,Level=LEVEL)`	You can also set the `Extension` and `Int2Int` name-value arguments.
`Y = lwtcoef2(TYPE,XDEC,LS,LEVEL,LEVEXT)`	Errors	`Y = lwtcoef2(CA,CH,CV,CD,Name=Value)` with the lifting decomposition `CA`, `CH`, `CV`, and `CD` in place of `XDEC`, and the following name-value arguments: • Replace `LS` with `LiftingScheme`, where `LiftingScheme` is a `liftingScheme` object. • Replace `LEVEXT` with `Level`. • Replace `TYPE` with the `Type` and `OutputType` name-value arguments. • `LEVEL` is no longer needed.	According to the value of `TYPE`, set the `Type` and `OutputType` name-value arguments as listed: • `'a'` — `Type="approximation"` and `OutputType="projection"` • `'ca'` — `Type="approximation"` and `OutputType="coefficients"` • `'d'` — `Type="detail"` and `OutputType="projection"` • `'cd'` — `Type="detail"` and `OutputType="coefficients"`

Functionality	Result	Use Instead	Compatibility Considerations
`[CA,CD] = lwt2(X,W,LEVEL,'typeDEC','wp')`	Errors	Not applicable	The wavelet packet decomposition option is no longer provided.
`X_InPlace = lwt2(X,LS)`	Errors	Not applicable	In-place transforms are no longer supported.
`X = ilwt2(AD_In_Place,W)`	Errors	Not applicable	In-place transforms are no longer supported.
`[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,HiD,LoR,HiR,ELS)`	Errors	`[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,LoR,LiftingSteps=ELS)`, where `ELS` is a structure array consisting of elementary lifting steps.	You can also scale the filters by a normalization factor. For more information about elementary lifting steps, see `liftingStep`.
`liftfilt(LoD,HiD,LoR,HiR,ELS,TYPE,VALUE)`	Errors	NA	This syntax is no longer supported.

CustomLowpassFilter name-value argument in liftingScheme must be a cell array
Behavior change

Starting this release, to use a lowpass filter to create a lifting scheme associated with an orthogonal wavelet, you must specify `CustomLowpassFilter` as a cell array. If you specify `CustomLowpassFilter` as a vector, `liftingScheme` will generate an error.

To update your code, change instances of `'CustomLowpassFilter',lpass`, where `lpass` is the vector, to `'CustomLowpassFilter',{lpass}`.

R2021a

Version: 5.6

New Features

Bug Fixes

Compatibility Considerations

1-D Lifting: Analyze signals using lifting

The new `liftingScheme` function enables you to efficiently implement the 1-D critically sampled discrete wavelet transform on signals using lifting. The `liftingScheme` function provides built-in lifting schemes for the orthogonal Daubechies extremal phase and least asymmetric wavelets. The `liftingScheme` function also provides built-in support for several biorthogonal B-spline wavelets.

The `liftingScheme` function paired with the enhanced 1-D lifting analysis and synthesis functions, `lwt` and `ilwt`:

- Supports real- and complex-valued signals
- Supports single- and double-precision data
- Supports multichannel signals
- Enables you to obtain integer-to-integer wavelet transforms
- Enables you to build your own lifting schemes with predict, update, and normalization steps
- Enables you to modify existing lifting schemes by adding or deleting lifting steps

You can use the `ls2filt` function to extract the analysis and synthesis filters from the lifting scheme. With the `lwtcoef` function, you can extract or reconstruct the 1-D wavelet coefficients from a lifting decomposition of a signal.

Compatibility Considerations

In previous releases, you used `liftwave` to create a lifting scheme structure. Starting with R2021a, use `liftingScheme` to create a lifting scheme object for signal analysis. To perform a 1-D lifting analysis or synthesis, you can use the `lwt` and `ilwt` functions either with default settings or by specifying the lifting scheme object. You can use the `addlift` and `deletelift` object functions to easily add and delete steps to the lifting scheme. With the `ls2filt` object function, you can extract the analysis and synthesis filters from the lifting scheme.

Wavelet Time Scattering: Accelerate and deploy automatic feature extraction of T×C×B data

You can now use `waveletScattering` to create a time scattering network that supports time × channel × batch (T×C×B) inputs. The `waveletScattering` object and the following object functions now also support C/C++ code generation and accept `gpuArray` inputs:

- **C/C++ code generation** — `scatteringTransform`, `featureMatrix`, `filterbank`, `littlewoodPaleySum`, `log`, `centerFrequencies`, `numorders`, `numfilterbanks`, `numCoefficients`, and `paths`
- **GPU acceleration** — `scatteringTransform`, `featureMatrix`, `log`, and `scattergram`

You must have MATLAB Coder to generate C/C++ code. You must have Parallel Computing Toolbox for `gpuArray` support. For more information, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox).

Signal Multiresolution Analyzer App: Performance improvements and MATLAB Online support

Starting this release, the **Signal Multiresolution Analyzer** app provides faster visualization and synthesis of wavelet and empirical mode decompositions of signals. You can also now run **Signal Multiresolution Analyzer** on MATLAB Online.

Discrete Decimated Wavelet Analysis: Obtain wavelet transforms for complex-valued data

The `dwt`, `idwt`, `wavedec`, and `waverec` functions now support complex-valued and single-precision data.

Maximal Overlap Discrete Wavelet Transform: Obtain MODWT of multichannel signals

The `modwt`, `imodwt`, and `modwtmra` functions now support multichannel signals. The three functions now also support complex-valued and single-precision data.

Haar Transform: Obtain Haar transform of SSCB data

The `haart2` and `ihaart2` functions now support spatial-by-spatial-by-channel-by-batch (SSCB) data. Both functions now also support single-precision data.

GPU Computing: Accelerate wavelet functions on your GPU

These Wavelet Toolbox functions now accept `gpuArray` inputs:

- **Discrete Wavelet Analysis** — `haart`, `ihaart`, `haart2`, `ihaart2`, `idwt`, `idwt2`, `waverec`, and `waverec2`

This support requires Parallel Computing Toolbox. For more information, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox).

GPU Code Generation: Automatically generate GPU code for wavelet functions

These Wavelet Toolbox functions now support CUDA® code generation:

- **Denoising** — `wdenoise` and `wdenoise2`
- **Discrete Wavelet Transforms** — `waverec` and `waverec2`

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

C/C++ Code Generation: Automatically generate code for wavelet functions

These Wavelet Toolbox functions now support C/C++ code generation:

- **Time-Frequency Analysis** — `cqt` and `icqt`

You must have MATLAB Coder to generate C/C++ code.

Deep Learning Example: Perform modulation classification using wavelet-derived features and deploy onto hardware

This release introduces an example, Modulation Classification Using Wavelet Analysis on NVIDIA Jetson, that shows how to generate and deploy CUDA code for modulation classification.

Functionality being removed or changed

1-D Lifting Scheme Changes

Still runs

The following lifting functions will be removed in a future release:

- `liftwave` — create a lifting scheme
- `addlift` — add a lifting step to a lifting scheme created using `liftwave`
- `ls2filt` — extract filters from a lifting scheme created using `liftwave`
- `filt2ls` — create a lifting scheme structure identical in type to a lifting scheme created using `liftwave`
- `lsinfo` — display information of a lifting scheme created using `liftwave`
- `displs` — display a lifting scheme created using `liftwave`
- `wavenames` — display wavelet names supported by `liftwave`

The syntax of 1-D lifting functions has also changed. The new syntax uses name-value arguments.

Functionality	Result	Use Instead	Compatibility Considerations
`LS = liftwave(WNAME)`	Still runs	`LS = liftingScheme('Wavelet',WNAME)`	You can also use `liftingScheme` to create a lifting scheme by specifying lowpass filter coefficients or customized lifting steps.

`LS = liftwave(WNAME,'Int2Int')`	Still runs	`LS = liftingScheme('Wavelet',WNAME)`
		
		 `[CA,CD] = lwt(X,'LiftingScheme',LS,'Int2Int',true)`
			To preserve integer-valued data, set the `Int2Int` name-value pair of the `lwt` function to `true`.

Functionality	Result	Use Instead	Compatibility Considerations
`[CA,CD] = lwt(X,W)`	Errors	`[CA,CD] = lwt(X,'Wavelet',W)`	You can also obtain the lifting wavelet transform (LWT) of a 1-D signal using a lifting scheme by setting the `LiftingScheme` name-value argument.
`[CA,CD] = lwt(X,W,LEVEL)`	Errors	`[CA,CD] = lwt(X,'Wavelet',W, 'Level',LEVEL)`	You can also specify the extension mode by setting the `ExtensionMode` name-value argument.
`X = ilwt(CA,CD,W)`	Errors	`X = ilwt(CA,CD,'Wavelet',W)`	You can also set the `LiftingScheme` name-value argument to obtain the inverse LWT.
`X = ilwt(CA,CD,W,LEVEL)`	Errors	`X = ilwt(CA,CD,'Wavelet',W,'Level',LEVEL)`	You can also set the `ExtensionMode` and `Int2Int` name-value arguments.
`Y = lwtcoef(TYPE,XDEC, LS,LEVEL,LEVEXT)`	Errors	`Y = lwtcoef(CA,CD,Name,Value)` with the lifting decomposition `CA` and `CD` in place of `XDEC`, and the following name-value arguments: • Replace `LS` with `'LiftingScheme'`, where `'LiftingScheme'` is a `liftingScheme` object. • Replace `LEVEXT` with `'Level'`. • Replace `TYPE` with the `Type` and `OutputType` name-value arguments. • `LEVEL` is no longer needed.	According to the value of `TYPE`, set the `Type` and `OutputType` name-value arguments as listed: • `'a'` — `'Type','approximation'` and `'OutputType','projection'` • `'ca'` — `'Type','approximation'` and `'OutputType','coefficients'` • `'d'` — `'Type','detail'` and `'OutputType','projection'` • `'cd'` — `'Type','detail'` and `'OutputType','coefficients'`

Functionality	Result	Use Instead	Compatibility Considerations
`[CA,CD] = lwt(X,W,LEVEL,'typeDEC','wp')`	Errors	NA	The wavelet packet decomposition option is no longer provided.
`X_InPlace = lwt(X,W)`	Errors	NA	In-place transforms are no longer supported.
`X = ilwt(AD_In_Place,W)`	Errors	NA	In-place transforms are no longer supported.

Wavelet Time Scattering: waveletScattering property Decimate has been removed
Errors

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`waveletScattering` property `Decimate`	Errors	`OversamplingFactor`	• Replace all instances of `'Decimate',true` with `'OversamplingFactor',0`. • Replace all instances of `'Decimate',false` with `'OversamplingFactor',Inf`.

Wavelet Time Scattering: featureMatrix function syntax will be deprecated
Still runs

One of the syntaxes for the `waveletScattering` object function `featureMatrix` will be deprecated in a future release.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`smat = featureMatrix(scatternet,s)`, where `s` is the cell array of scattering coefficients obtained from `scatteringTransform`	Still runs	`smat = featureMatrix(scatternet,x)` where `x` is the input data	Replace all instances of `smat = featureMatrix(scatternet,s)` with `smat = featureMatrix(scatternet,x)`.

Wavelet Time Scattering: Highest center frequency calculated using geometric mean
Behavior change

Starting in R2021a, `waveletScattering` uses the geometric mean to compute the highest wavelet center frequency in the time scattering network. The method for determining how to space linearly those frequencies lower than the invariance scale has also changed. These changes improve the Littlewood-Paley sums of the resulting filter banks.

Center frequencies are logarithmically spaced from the highest frequency to the frequency that corresponds to the invariance scale. Starting in R2021a, depending on scattering network parameters such as the invariance scale, the *number* of filters you obtain may be different than in previous releases. The method for applying the filters to compute the scattering and scalogram coefficients has not changed.

R2020b

Version: 5.5

New Features

Bug Fixes

Empirical Wavelet Transform: Perform adaptive signal decomposition using fully-automated spectrum segmentation

With the new `ewt` function, you can decompose a real- or complex-valued signal using an adaptive wavelet subdivision scheme. The empirical wavelet transform (EWT) determines the wavelet filter passbands based on peaks in the signal spectrum. You can control the number and width of passbands through a number of options including frequency resolution of the spectral estimate, maximum number of peaks, peak threshold, and segmentation method. You can specify the maximum number of segments through a number of options including the maximum number of detected peaks. Like all multiresolution techniques, EWT provides a perfect reconstruction of the input signal. Additionally, the EWT coefficients (or analysis) partition the energy of the input signal into the separate passbands.

- You can obtain the peak normalized frequencies identified in the signal and the approximate frequency passbands of the wavelet filter bank.
- The `ewt` function optionally outputs the data-adaptive wavelet filter bank along with information on the segmentation.
- You can visualize the MRA components in the signal.
- For a real-valued signal, you can use the MRA components with the `hht` function to visualize the Hilbert spectrum of the signal.
- The `ewt` function supports single and double precision.
- You can generate C/C++ code for workflows that include empirical wavelet transforms. You must have MATLAB Coder to generate C/C++ code.

CWT Marginals: Obtain and visualize time-averaged and scale-averaged wavelet spectrum

Use the new `timeSpectrum` and `scaleSpectrum` functions to obtain the time- or scale-averaged wavelet power spectrum of a signal using `cwtfilterbank`. The CWT marginals can be obtained from either the signal or the CWT coefficients.

- You have a variety of ways to normalize the power of the scale- and time-averaged wavelet spectrum. For example, you can normalize as a PDF.
- You can visualize the CWT marginals alongside the scalogram, enabling you to see how the marginal data is derived.
- `timeSpectrum` and `scaleSpectrum` both accept `gpuArray` inputs, enabling you to compute the scale- and time-averaged wavelet spectrum on your GPU. This support requires Parallel Computing Toolbox. For more information, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox).
- You can generate C/C++ code for workflows that include CWT marginals. You must have MATLAB Coder to generate C/C++ code.
- You can obtain the scale-averaged wavelet spectrum over specific frequency limits.
- You can obtain the time-averaged wavelet spectrum over specific time limits.

Deep Learning Examples: Classify signals using wavelet-derived features and deploy onto hardware

This release introduces examples that employ wavelet techniques and deep learning:

- Crack Identification From Accelerometer Data shows how to detect transverse pavement cracks and localize their position.
- Deploy Signal Classifier on NVIDIA Jetson Using Wavelet Analysis and Deep Learning shows how to generate and deploy CUDA code that classifies human electrocardiogram (ECG) signals.
- Deploy Signal Classifier Using Wavelets and Deep Learning on Raspberry Pi shows how to generate and deploy C++ code for ECG signal prediction.

GPU Computing: Accelerate wavelet functions on your GPU

These Wavelet Toolbox functions now accept `gpuArray` inputs:

- **Discrete Wavelet Analysis** — `dwt`, `dwt2`, `wavedec`, and `wavedec2`
- **Time-Frequency Analysis** — `timeSpectrum`, `scaleSpectrum`, and `wcoherence`
- **Upsampling and Downsampling** — `dyaddown` and `dyadup`
- **Data Extension** — `wextend`

This support requires Parallel Computing Toolbox. For more information, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox).

GPU Code Generation: Automatically generate GPU code for wavelet functions

These Wavelet Toolbox functions now support CUDA code generation:

- **Discrete Wavelet Transforms** — `dwt`, `idwt`, `dwt2`, `idwt2`, `wavedec`, and `wavedec2`
- **Nondecimated Discrete Wavelet Transforms** — `modwt`, `imodwt`, and `modwtmra`
- **Multisignal Discrete Wavelet Analysis** — `mdwtdec`

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

C/C++ Code Generation: Automatically generate code for wavelet functions

These Wavelet Toolbox functions now support C/C++ code generation:

- **Discrete Wavelet Analysis** — `swt`, `iswt`, `swt2`, and `iswt2`
- **Time-Frequency Analysis** — `ewt`, `timeSpectrum`, `scaleSpectrum`, and `wcoherence`
- **2-D Denoising** — `wdenoise2`

You must have MATLAB Coder to generate C/C++ code.

R2020a

Version: 5.4

New Features

Bug Fixes

Compatibility Considerations

GPU Computing: Accelerate continuous wavelet transform on your GPU

Convert your data into a GPU array and perform continuous wavelet transforms (CWTs) on your GPU. The `cwtfilterbank` object and `cwt` function now accept `gpuArray` inputs. In many cases, execution on the GPU is faster than on the CPU, so this feature might offer improved performance. To take full advantage of the GPU when performing multiple CWTs, first create a `cwtfilterbank` object and then use the `wt` object function. This workflow minimizes overhead and maximizes performance. The following examples show how to accelerate the computation of wavelet-derived features using `gpuArray` in deep learning workflows.

- Wavelet Time Scattering with GPU Acceleration — Music Genre Classification
- Wavelet Time Scattering with GPU Acceleration — Spoken Digit Recognition
- GPU Acceleration of Scalograms for Deep Learning

This support requires Parallel Computing Toolbox. For more information, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox).

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.

1-D Multisignal Discrete Wavelet Packet Transforms: Automatically perform wavelet packet analysis of multichannel signals

You can use the new `dwpt` and `idwpt` functions to obtain wavelet packet transforms of multichannel signals.

- You can specify wavelets or wavelet filters.
- The functions support single- and double-precision inputs.
- You can obtain either the full wavelet packet tree or just the terminal nodes.
- You can generate C/C++ code for workflows that include wavelet packet transforms. You must have MATLAB Coder to generate C/C++ code.

Kingsbury Q-shift Dual-Tree Complex Wavelet Transforms: Perform shift-invariant and directionally sensitive discrete multiresolution analysis with minimal redundancy

This release introduces Kingsbury Q-shift dual-tree complex wavelet transforms (DTCWT) `dualtree`, `idualtree`, `dualtree2`, and `idualtree2`. DTCWT overcomes many of the limitations of the critically downsampled discrete wavelet transform, including shift variance and lack of directional sensitivity. DTCWT does this with a minimal increase in redundancy of 2^d for d-dimensional data. Use the `dualtree` and `dualtree2` functions to obtain decompositions of 1-D and 2-D data, respectively. With the `idualtree` and `idualtree2` functions, you can invert the transforms for perfect reconstruction.

For workflows that involve dual-tree complex wavelet transforms, use the new functions `dualtree`, `idualtree`, `dualtree2`, and `idualtree2`.

- The functions support arbitrary input data sizes.
- The functions support single- and double-precision inputs.
- You can easily reconstruct subband-limited approximations. You can apply different gains to different transform levels.
- You can generate C/C++ code for workflows that include dual-tree transforms. You must have MATLAB Coder to generate C/C++ code.

Compatibility Considerations

The wavelet decomposition `typetree` input argument values `'realdt'` and `'cplxdt'` for `dddtree` and `dddtree2` are no longer recommended and will be removed in a future release. Use the new functions `dualtree` and `dualtree2` instead.

New Examples: Hands-on introductions to continuous wavelet analysis and multiresolution analysis

This release introduces two examples that show how to perform and interpret basic wavelet analysis.

- Practical Introduction to Continuous Wavelet Analysis
- Practical Introduction to Multiresolution Analysis

GPU Computing: Accelerate Wigner-Ville distribution

The `wvd` function now accepts `gpuArray` inputs. This support requires Parallel Computing Toolbox. For more information, see Run MATLAB Functions on a GPU (Parallel Computing Toolbox).

GPU Code Generation: Generate single precision code for cwt

You can now generate CUDA code from the `cwt` function that supports single-precision input data. You must have MATLAB Coder and GPU Coder to generate CUDA code.

C/C++ Code Generation: Generate single precision code for cwtfilterbank

You can now generate C/C++ code from the `cwtfilterbank` object to obtain the continuous wavelet transform of single-precision data. You must have MATLAB Coder to generate C/C++ code.

wcoherence Function: Compute wavelet coherence over user-specified frequency or period range

The `wcoherence` function now accepts frequency limits and period limits as input.

C/C++ Code Generation: Automatically generate code for discrete wavelet analysis, time-frequency analysis, denoising, and multiscale variance estimation

These Wavelet Toolbox functions now support C/C++ code generation:

- **Discrete Wavelet Analysis** — `dualtree`, `idualtree`, `dualtree2`, `idualtree2`, `haart`, `ihaart`, `haart2`, and `ihaart2`
- **Time-Frequency Analysis** — `hht`
- **1-D Wavelet Packet Transforms** — `dwpt`, and `idwpt`
- **1-D Denoising** — `wdenoise`
- **1-D Statistical Analysis** — `modwtvar`

You must have MATLAB Coder to generate C/C++ code.

Functionality being removed or changed

'NumOctaves' name-value pair argument in wcoherence will be removed

Still runs

The `'NumOctaves'` name-value pair argument in `wcoherence` will be removed in a future release.

Functionality	Result	Use Instead	Compatibility Considerations
Name-value pair argument `'NumOctaves'` for `wcoherence`. For example: `[ ___ ] = wcoherence( ___ ,'NumOctaves',24)`	Still runs	In `wcoherence`, set either the: • Name-value pair argument `'FrequencyLimits'` to modify the frequency range of wavelet coherence. • Name-value pair argument `'PeriodLimits'` to modify the period range of wavelet coherence. See `cwtfreqbounds` for details.	Replace all instances of the `'NumOctaves'` name-value pair argument with either the `'FrequencyLimits'` or `'PeriodLimits'` name-value pair argument.

Wavelet decomposition types realdt and cplxdt for dddtree and dddtree2 functions will be removed
Still runs

The wavelet decomposition `typetree` input argument values `'realdt'` and `'cplxdt'` for `dddtree` and `dddtree2` are no longer recommended and will be removed in a future release.

Functionality	Result	Use Instead	Compatibility Considerations
`dddtree` function with `typetree` input argument value `'cplxdt'`	Still runs	`dualtree`	Update all instances of `dddtree` with `treetype` input argument value `'cplxdt'` to use the new `dualtree` function.
`dddtree2` function with `typetree` input argument value `'realdt'` or `'cplxdt'`	Still runs	`dualtree2`	Update all instances of `dddtree2` with `treetype` input argument values `'realdt'` or `'cplxdt'` to use the new `dualtree2` function.

Some tools in the Wavelet Analyzer App have been removed

These tools in the **Wavelet Analyzer App** have been removed.

Tools	Replacement
Continuous Wavelet 1-D (Using FFT)	• To take the CWT of a single time series, use `cwt`. • To take the CWT of multiple time series, the recommended procedure is to precompute a CWT filter bank with `cwtfilterbank` and apply the filter bank to multiple time series. See Using CWT Filter Bank on Multiple Time Series. • To visualize the scalogram, use `cwt`. • To visualize wavelets in time and frequency, use `cwtfilterbank`.
New Wavelet for CWT	• To tune the generalized Morse wavelet to your needs, vary the time-bandwidth and symmetry parameters of `cwtfilterbank` or `cwt`. • To create a custom DWT filter bank, use `dwtfilterbank`. See Add Quadrature Mirror and Biorthogonal Wavelet Filters.
Fractional Brownian Generation 1-D	To synthesize fractional Brownian motion, use `wfbm`.

Tools	Replacement
Wavelet Display, **Wavelet Packet Display**	• To visualize the analytic Morse, Morlet, and bump wavelets in time and frequency, use `cwtfilterbank`. • To visualize orthogonal and biorthogonal wavelets in time and frequency, use `dwtfilterbank`. • To visualize in time other wavelets such as the Meyer, Morlet, Gaussian, Mexican hat, and Shannon wavelets, use `wavefun`. • To display wavelet packets, use `wpfun`.
Signal Extension, **Image Extension**	To extend real-valued vectors or matrices, use `wextend`.

R2019b

Version: 5.3

New Features

Bug Fixes

Compatibility Considerations

Shearlets: Generate sparse representations of images automatically for deep learning and image processing

The new `shearletSystem` function creates a cone-adapted shearlet system that can be applied to real-valued 2-D images. The system uses bandlimited shearlets and provides a translation-covariant transform. You can use the shearlet system to obtain directionally sensitive sparse representations of images with anisotropic features. The representations can be used in areas such as image classification, image denoising, feature extraction, and compressed sensing. To learn more about shearlets, see Shearlet Systems. You can use `shearletSystem` to create a shearlet system specific to your requirements:

- You can specify real- or complex-valued shearlets.
- You can normalize the shearlet system to be a Parseval frame.
- The shearlet system supports single and double precision.
- You can generate C/C++ code for workflows that include shearlet transforms. You must have MATLAB Coder to generate C/C++ code.

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.

GPU Computing: Accelerate automatic feature extraction using wavelet scattering on GPUs

Perform wavelet time scattering transforms on your GPU using a two-filter-bank wavelet scattering framework. This release includes examples that demonstrate how to use this new capability for signal classification.

This functionality requires Parallel Computing Toolbox and a CUDA-enabled NVIDIA® GPU with compute capability 3.0 or higher.

Machine and Deep Learning Examples: Classify signals using wavelet-derived features

This release introduces examples of signal classification using wavelet-derived features in machine learning and deep learning workflows.

- Wavelet Time Scattering with GPU Acceleration — Music Genre Classification
- Wavelet Time Scattering with GPU Acceleration — Spoken Digit Recognition

C/C++ Code Generation: Automatically generate code for multisignal discrete wavelet analysis

The Wavelet Toolbox functions `mdwtdec` and `mdwtrec` now support C/C++ code generation. You must have MATLAB Coder to generate C/C++ code.

Functionality being removed

Some tools in the Wavelet Analyzer app will be removed

Still runs

The following tools in the **Wavelet Analyzer** app will be removed in a future release.

Tools	Recommended Replacement
Continuous Wavelet 1-D (Using FFT)	• To take the CWT of a single time series, use `cwt`. • To take the CWT of multiple time series, the recommended procedure is to precompute a CWT filter bank with `cwtfilterbank` and apply the filter bank to multiple time series. See Using CWT Filter Bank on Multiple Time Series. • To visualize the scalogram, use `cwt`. • To visualize wavelets in time and frequency, use `cwtfilterbank`.
New Wavelet for CWT	• To tune the generalized Morse wavelet to your needs, vary the time-bandwidth and symmetry parameters of `cwtfilterbank` or `cwt`. • To create a custom DWT filter bank, use `dwtfilterbank`. See Add Quadrature Mirror and Biorthogonal Wavelet Filters.
Fractional Brownian Generation 1-D	To synthesize fractional Brownian motion, use `wfbm`.
Wavelet Display, **Wavelet Packet Display**	• To visualize the analytic Morse, Morlet, and bump wavelets in time and frequency, use `cwtfilterbank`. • To visualize orthogonal and biorthogonal wavelets in time and frequency, use `dwtfilterbank`. • To visualize in time other wavelets such as the Meyer, Morlet, Gaussian, Mexican hat, and Shannon wavelets, use `wavefun`. • To display wavelet packets, use `wpfun`.
Signal Extension, **Image Extension**	To extend real-valued vectors or matrices, use `wextend`.

R2019a

Version: 5.2

New Features

Bug Fixes

Compatibility Considerations

Wavelet Scattering for Images: Generate compact invariant feature representations automatically for image classification

The new `waveletScattering2` function creates a framework for wavelet image scattering. The framework uses complex-valued 2-D Morlet wavelets to automatically generate features from RGB or grayscale images. Wavelet image scattering yields representations insensitive to image translations and rotations without sacrificing class discriminability. As with 1-D wavelet time scattering, you do not need a large dataset to train filters for accurate classification. You can use `waveletScattering2` to create a framework specific to your requirements:

- Specify the number of linearly spaced rotations per wavelet filter and the amount of translation invariance.
- Specify how much the scattering coefficients are oversampled with respect to the critically downsampled values.
- Optimize the scattering transform based on the wavelet bandwidths and the number of wavelets per octave.

You can also examine characteristics of the framework filter banks, including center spatial frequencies and frequency supports.

Image Denoising: Automatically denoise images while preserving sharp features

The `wdenoise2` function provides a simple interface to a variety of denoising methods that can be applied to RGB or grayscale images. You can use `wdenoise2` with preset default values or specify a variety of denoising methods, including empirical Bayesian, false discovery rate, and Donoho-Johnstone methods. You can use cycle spinning for translationally invariant denoising with all the supported denoising methods. You can also denoise an RGB image in its PCA color space.

GPU and C/C++ Code Generation: Automatically generate GPU or C/C++ code for the continuous wavelet transform

You can now generate code for workflows that include continuous wavelet transforms.

- The `cwt` function supports CUDA code generation.
- The `cwtfilterbank` object supports C/C++ code generation.

You must have MATLAB Coder to generate C/C++ code. CUDA code generation requires MATLAB Coder and GPU Coder.

Wavelet Scattering Examples: Classify images and time series using wavelet scattering and deep learning

This release introduces several examples of using wavelet scattering frameworks for image and time series classification. Texture Classification with Wavelet Image Scattering and Digit Classification with Wavelet Scattering use wavelet image scattering to classify textures and digits, respectively. Acoustic Scene Recognition Using Late Fusion and Spoken Digit Recognition with Wavelet Scattering and Deep Learning classify audio data using wavelet time scattering and a deep convolutional neural network based on mel-frequency spectrograms.

1-D Wavelet Time Scattering: Enable variable downsampling of coefficients

When you create a framework with `waveletScattering`, you can use the new `OversamplingFactor` property to control the amount of downsampling in the scattering transform. By default, the scattering coefficients are critically downsampled by the maximum amount possible.

Functionality being removed or changed

waveletScattering property Decimate will be removed
Still runs

The `waveletScattering` property `Decimate` will be removed in a future release. Use the new property `OversamplingFactor` instead.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`Decimate`	Still runs	Use `OversamplingFactor`	• Replace all instances of `'Decimate',true` with `'OversamplingFactor',0`. • Replace all instances of `'Decimate',false` with `'OversamplingFactor',Inf`.

R2018b

Version: 5.1

New Features

Bug Fixes

Compatibility Considerations

Wavelet Scattering: Generate invariant representations of time series data automatically for classification and regression

The new `waveletScattering` function creates a framework for wavelet time scattering. The framework uses wavelets and a lowpass scaling function to automatically generate low-variance representations of real-valued time series data. Wavelet time scattering yields representations insensitive to translations in the input signal without sacrificing class discriminability. `waveletScattering` uses predefined wavelet and lowpass scaling filters for feature extraction. You do not need a large dataset to train filters for accurate classification. You can use `waveletScattering` to create a framework specific for your requirements:

- Specify the duration of translation invariance.
- Set the number of filter banks in the framework, and the number of wavelet filters per octave in each filter bank
- Generate the feature matrix to use in a classifier.

You can also return the filter banks used in the framework, and inspect their characteristics including the wavelet filter center frequencies and frequency standard deviations.

Signal Multiresolution Analyzer App: Visualize and synthesize wavelet and empirical mode decompositions of signals

The **Signal Multiresolution Analyzer** app enables visualization of wavelet and empirical mode decompositions of signals. The relative energy and frequency band of the individual components in the decomposition are provided. You can synthesize signals from components you choose that best capture features of interest. The app enables easy comparison of multiple reconstructions. You can export reconstructions and generate MATLAB scripts to reproduce results in your workspace.

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.
- `xwvd` computes the cross Wigner-Ville distribution of two signals.

Scattering Examples: Classify signals using wavelet time scattering and machine learning

This release introduces three examples of feature extraction using wavelet time scattering for machine learning workflows.

- Wavelet Time Scattering for ECG Signal Classification
- Wavelet Time Scattering Classification of Phonocardiogram Data
- Music Genre Classification Using Wavelet Time Scattering

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.

Functionality being removed or changed

The `BPfrequencies` and `BPperiods` object functions of `cwtfilterbank` have been renamed `centerFrequencies` and `centerPeriods`, respectively. The functionality remains unchanged. `BPfrequencies` and `BPperiods` will be removed in a future release.

Compatibility Considerations

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`BPfrequencies`	Still runs	Use `centerFrequencies`	Replace all instances of `BPfrequencies` with `centerFrequencies`.
`BPperiods`	Still runs	Use `centerPeriods`	Replace all instances of `BPperiods` with `centerPeriods`.

R2018a

Version: 5.0

New Features

Bug Fixes

Compatibility Considerations

Filter Banks for Continuous Wavelet Transform: Create, visualize, and use filter banks for time-frequency analysis

Use the `cwtfilterbank` function to create a continuous wavelet transform (CWT) filter bank.

- Visualize wavelets in time and frequency.
- Create filter banks with specific frequency or period ranges, and with a specific number of wavelet filters per octave.
- Measure 3-dB bandwidths and q-factor.
- Use the precomputed filter bank with `cwt` to provide a more efficient implementation of the CWT for multiple signals.

Filter Banks for Discrete Wavelet Transform: Explore time-frequency characteristics of wavelets and scaling functions

Use the `dwtfilterbank` function to create a discrete wavelet transform (DWT) filter bank.

- Visualize wavelets and scaling functions in time and frequency.
- Measure the 3-dB bandwidths of the wavelet and scaling functions. You can also measure energy concentration of the wavelet and scaling functions in the theoretical DWT passbands.
- Create a DWT filter bank using your own custom filters. You can determine whether the filter bank is orthogonal or biorthogonal.
- Determine the frame bounds of the filter bank.

Constant-Q Transform: Perform adaptive time-frequency analysis using nonstationary Gabor frames

Use the `cqt` function to take the constant-Q transform of signals. With the `icqt` function, you can invert the transform for perfect reconstruction. The constant-Q transform, which is based on nonstationary Gabor frames, constructs and applies adaptive, compact bandwidth windows directly in the frequency domain.

- Perform minimally redundant and maximally redundant constant-Q transforms.
- Visualize the constant-Q transform of a signal.
- Obtain the Gabor frames used in the analysis, and the frequency shifts in the discrete Fourier transform bins between the passbands.
- Use `cqt` with a number of different windows for the nonstationary Gabor frames, including the Hann window and Hamming window.
- Set the frequency limits over which the constant-Q transform has a logarithmic frequency response. You can also set the number of frequency bins per octave.
- Use `cqt` on multichannel signals.

Scalogram View: Detect transients and perform time-frequency analysis with the Signal Analyzer App

The **Signal Analyzer** app can now compute scalograms using Morse wavelets. Scalograms enable you to detect transients and perform time-frequency analysis. You must have a Signal Processing Toolbox license to use **Signal Analyzer**.

Empirical Mode Decomposition and Hilbert-Huang Transform: Perform data-adaptive time-frequency analysis of nonlinear and nonstationary processes

Use `emd` to decompose a nonlinear or nonstationary process into its intrinsic modes of oscillation. `emd` iterates on an input signal to extract natural AM-FM modes, also known as intrinsic mode functions, contained in the data.

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 or nonstationary signal.

Continuous Wavelet Transform: Improved control of time-frequency parameters and timetable support

The `cwt` and `icwt` functions now provide more control over the forward and inverse continuous wavelet transforms.

- The Morse and analytic Morlet scaling functions are now part of the CWT. With the scaling coefficients, you can obtain a more accurate signal reconstruction.
- You can now specify a frequency range and period range to use in the CWT. With the new `cwtfreqbounds` function, you can determine the appropriate frequency limits and customize the behavior of the CWT at low-frequency and high-frequency bounds.
- The `cwt` function now supports a single-variable, uniformly sampled timetable.

Machine and Deep Learning Examples: Classify signals using wavelet-based feature extraction and deep learning

This release introduces two examples for classifying signals using wavelets. ECG Classification Using Wavelet Features uses wavelet-based features and a support vector machine to classify human electrocardiogram (ECG) waveforms. The example demonstrates how wavelet features can significantly reduce the size of the data and yet still retain the salient differences between waveforms. Signal Classification with Wavelet Analysis and Convolutional Neural Networks uses continuous wavelet analysis and transfer learning to classify ECG waveforms using a deep convolutional neural network (CNN). The example creates time-frequency images of the ECG waveforms using the continuous wavelet transform and leverages a pretrained deep CNN for image classification.

Functionality Being Removed or Changed

The `'NumOctaves'` name-value pair argument in `cwt` will be removed in a future release.

Functionality	Result	Use Instead	Compatibility Considerations
Name-value pair argument `'NumOctaves'` for `cwt`. For example: `[ ___ ] = cwt( ___ ,'NumOctaves',24)`	Still runs	In `cwt`, set either the: • Name-value pair argument `'FrequencyLimits'` to modify the frequency range of the CWT. • Name-value pair argument `'PeriodLimits'` to modify the period range of the CWT. See `cwtfreqbounds` for details.	Replace all instances of the `'NumOctaves'` name-value pair argument with either the `'FrequencyLimits'` or `'PeriodLimits'` name-value pair argument.
Default regression weights for `dwtleader`	Still runs	In `dwtleader`, use the name-value pair argument `'RegressionWeight'` to set the weight option used in the least-squares regression model. See `dwtleader` for details.	To duplicate the behavior of `dwtleader` found in releases prior to R2018a, update all instances of `dwtleader` to include the name-value pair argument `'RegressionWeight'` set to `'scale'`.

R2017b

Version: 4.19

New Features

Bug Fixes

Compatibility Considerations

Wavelet Signal Denoiser App: Visualize and denoise time-series data

The **Wavelet Signal Denoiser** app enables visualization and denoising of 1-D signals in the MATLAB workspace. The app provides an initial denoised version of your data using default parameters. You can duplicate, add, and modify the denoising settings to compare multiple denoised versions of your signal to determine the optimal settings. After you determine the optimal settings for your signal, you can export the denoised data and generate MATLAB scripts to reproduce the results.

Wavelet Denoising: Denoise time-series data with improved automatic selection of input parameters

The `wdenoise` function provides a simple interface to a variety of denoising methods that can be applied to 1-D signals. Input parameters are automatically chosen for quick and easy use. `wdenoise` supports empirical Bayesian methods, false discovery rate, and James-Stein block thresholding. The function also supports the denoising of multichannel time-series data and MATLAB timetables.

Continuous Wavelet Transform: Perform time-frequency analysis of complex-valued time-series data

The `cwt` function now supports complex-valued data for analysis and synthesis. With this enhancement, you can perform time-frequency analysis of rotary components. A visualization affordance has also been added so that you can more easily discern regions where edge effects become significant.

Functionality Being Removed or Changed

Functionality	Result	Use Instead	Compatibility Considerations
`swt2`	Still runs	Not applicable	To distinguish a single-level decomposition of a truecolor image from a multilevel decomposition of an indexed image, the approximation and detail coefficient arrays of truecolor images are 4-D arrays. See Migrate from R2017b to Previous Release or Migrate from Previous Release to R2017b. `swt2` uses double-precision arithmetic internally and returns double-precision coefficient matrices.

Functionality	Result	Use Instead	Compatibility Considerations
`iswt2`	Still runs	Not applicable	To distinguish a single-level decomposition of a truecolor image from a multilevel decomposition of an indexed image, the approximation and detail coefficient arrays of truecolor images are 4-D arrays. See Migrate from R2017b to Previous Release or Migrate from Previous Release to R2017b.

Migrate from Previous Releases to R2017b

Depending on the original input data type and level of wavelet decomposition, you might have to take different steps to make `swt2` coefficient arrays from previous releases compatible with R2017b coefficient arrays. The steps depend on whether you have a single coefficient array or separate approximation and detail coefficient arrays.

Single Coefficient Array	Multiple Coefficient Arrays
Input: Index image • Single-level: No compatibility issues • Multi-level: No compatibility issues	Input: Index image • Single-level: No compatibility issues • Multi-level: No compatibility issues

Input: Truecolor image
• Single-level: If `swc` is the output of `swt2` from a previous release, execute:
`swc1 = double(swc);`
• Multi-level: If `swc` is the output of `swt2` from a previous release, execute:
`swc1 = double(swc);`
	Input: Truecolor image
	• Single-level: If `ca`, `chd`, `cvd`, and `cdd` are outputs of `swt2` from a previous release, execute:
	`ca1 = double(ca);`
	`chd1 = double(chd);`
	`cvd1 = double(cvd);`
	`cdd1 = double(cdd);`
	`ca2 = reshape(ca1,[m,n,1,3]);`
	`chd2 = reshape(chd1,[m,n,1,3]);`
	`cvd2 = reshape(cvd1,[m,n,1,3]);`
	`cdd2 = reshape(cdd1,[m,n,1,3]);`
	• Multi-level: If `ca`, `chd`, `cvd`, and `cdd` are outputs of `swt2` from a previous release, execute:
	`ca1 = double(ca);`
	`chd1 = double(chd);`
	`cvd1 = double(cvd);`
	`cdd1 = double(cdd);`

Migrate from R2017b to Previous Releases

Depending on the original input data type and level of wavelet decomposition, you might have to take different steps to make R2017b `swt2` coefficient arrays compatible with the coefficient arrays from previous releases. The steps depend on whether you have a single coefficient array or separate approximation and detail coefficient arrays.

Single Coefficient Array	Multiple Coefficient Arrays
Input: Index image • Single-level: No compatibility issues • Multi-level: No compatibility issues	Input: Index image • Single-level: No compatibility issues • Multi-level: No compatibility issues

Input: Truecolor image
• Single-level: No compatibility issues
• Multi-level: No compatibility issues
	Input: Truecolor image
	• Single-level: If `ca`, `chd`, `cvd`, and `cdd` are outputs of `swt2` from R2017b, execute:
	`ca1 = single(squeeze(ca));`
	`chd1 = single(squeeze(chd));`
	`cvd1 = single(squeeze(cvd));`
	`cdd1 = single(squeeze(cdd));`
	• Multi-level: No compatibility issues

R2017a

Version: 4.18

New Features

Bug Fixes

Compatibility Considerations

Multiscale Local Polynomial Transform: Perform smoothing of nonuniformly sampled signals

The `mlpt`, `imlpt`, `mlptdenoise`, and `mlptrecon` functions use local polynomial lifting schemes to perform multiresolution analysis, denoising, and reconstruction of nonuniformly sampled single-channel and multichannel signals.

3-D Dual-Tree Complex Wavelet Transform: Perform directionally selective wavelet analysis of volumetric data

The `dualtree3` and `idualtree3` functions perform the 3-D complex dual-tree transform. This transform is selective to orientation and provides perfect reconstruction. For an example demonstrating how the dual-tree complex discrete transform provides advantages over the critically sampled discrete wavelet transform for signal, image, and volume processing, see the featured example, Dual-Tree Wavelet Transforms.

Length-16 Q-shift filter

The `dtfilters` function now includes a Q-shift filter of length 16 that is used in dual-tree transforms. To use this filter, specify `'dtf4'` as the filter name.

Compatibility Considerations

The previous version of `dtfilters` included filters of lengths 6, 10, 14, and 18. These lengths corresponded to `'dtf1'`, `'dtf2'`, `'dtf3'`, and `'dtf4'`, respectively. To specify the new filter of length 16, use `'dtf4'`. The filter of length 18 now corresponds to `'dtf5'`. If you have any code that specifies `dtfilters` with a `'dtf4'` filter name, change the filter name to `'dtf5'` to continue using a filter of length 18.

Modified treatment of boundary conditions in Wavelet Analyzer

In previous releases, the **Wavelet Analyzer** app reset the DWT extension mode to the default `'sym'` behavior, overriding any previous setting. Starting with this release, the app uses the current setting. To display or change the current setting, use `dwtmode`.

R2016b

Version: 4.17

New Features

Bug Fixes

Compatibility Considerations

Code Generation: Generate C code for DWT, wavelet packets, and denoising using MATLAB Coder

You can now generate code for workflows that support:

- 1-D and 2-D critically sampled discrete wavelet transforms (DWT)
- 1-D maximal overlap discrete wavelet transforms (MODWT)
- Wavelet packet transforms (MODWPT)
- 1-D and 2-D wavelet denoising

You must have MATLAB Coder to generate code. The following functions support code generation:

- `dwt`, `idwt`
- `dwt2`, `idwt2`
- `modwt`, `imodwt`, `modwtmra`
- `modwpt`, `imodwpt`, `modwptdetails`
- `appcoef`, `appcoef2`
- `detcoef`, `detcoef2`
- `wavedec`, `wavedec2`
- `waverec`, `waverec2`
- `wden`, `wdencmp`, `ddencmp`
- `qmf`
- `dyadup`
- `wextend`
- `thselect`
- `wthresh`, `wthcoef`, `wthcoef2`

Continuous Wavelet Transform: Analyze signals with improved automatic selection of wavelet and scales

This release provides an updated version of the continuous wavelet transform, `cwt`, and a new inverse transform, `icwt`, for reconstructing the original signal. These functions are easier to use because they have simple interfaces and include default values for the wavelet and scales and frequency and period ranges are easy to specify. When you use the updated `cwt`, which use analytic wavelets and L1 normalization, `icwt` produce a more accurate reconstruction.

Compatibility Considerations

The old version of `cwt` continues to work, however, updating existing code to use the new version of `cwt` is recommended. Both the old and updated versions use the same function name. The inputs to the function determine automatically which version is used.

`icwt` is recommended instead of `icwtft` and `icwtlin`.

Morse Wavelets: Family of analytic wavelets for continuous wavelet analysis

The Morse family of analytic wavelets are ideal for continuous wavelet analysis. These exactly analytic wavelets are characterized by two parameters. You can vary these parameters to change the shape and duration of the wavelet as needed to analyze your signal or image.

Wavelet Leaders and Wavelet Transform Modulus Maxima: Characterize fractal data and singularities

The `wtmm` function returns an estimate of the global Holder exponent, which characterizes multifractal behavior. You can use `wtmm` to characterize cusp-like singularities. The `dwtleader` function distinguishes monofractal from multifractal behavior. You can use WTMM and wavelet leaders to analyze turbulence, electrophysical signals, and financial time series. Both functions accept 1-D data.

Haar Lifting Transforms: Perform multiresolution analysis of images and multichannel signals

Four Haar lifting transform functions have been added to the toolbox: `haart` and `ihaart` for 1-D signals, and `haart2` and `ihaart2` for 2-D signals. The Haar wavelet, though not continuous, is the simplest possible wavelet and is a special case of the Daubechies wavelet, `db1`. These new Haar lifting transforms are computationally efficient.

Compression data files default to uint64 data

By default, `wcompress` now writes `.wtc` files using `uint64` precision data. A new `'legacy'` flag has been added to write `.wtc` files using the earlier `uint32` precision data format. You do not need to include the `'legacy'` flag for reading a `.wtc` file because `wcompress` automatically detects and correctly reads the data format.

Compatibility Considerations

To share a data file with someone using a previous release, use the `'legacy'` flag to create a `.wtc` file of `uint32` data. For example, `wcompress('c',x,'comp_data.wtc','legacy')`.

Wavelet Design and Analysis App renamed to Wavelet Analyzer

The Wavelet Design and Analysis app has been renamed to Wavelet Analyzer. To open the Wavelet Analyzer from the command line, use `waveletAnalyzer` instead of `wavemenu`.

Compatibility Considerations

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

wavedemo function removed

The `wavedemo` function has been removed. Use Wavelet Toolbox Examples instead.

Compatibility Considerations

Change all calls and links to `wavedemo` to point to Wavelet Toolbox Examples.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
Old `cwt`	Old functionality is detected based on the input syntax and the old `cwt` still runs.	Updated `cwt`	Update all instances of `cwt` to use the updated `cwt` syntax.
`wcompress`	An error occurs if you read `.wtc` files saved using `uint32` data.	Add a `'legacy'` flag to use `uint32` data instead of default `uint64` data	Add a `'legacy'` flag to instances of `wcompress` commands that read `.wtc` files containing `uint32` data.
`wavemenu`	Still opens Wavelet app	`waveletAnalyzer`	Update instances of `wavemenu` to use the new `waveletAnalyzer` function.
`wavedemo`	An error occurs .	Wavelet Toolbox Examples	Update instances of `wavedemo` to point to the Examples page.

R2016a

Version: 4.16

New Features

Bug Fixes

Compatibility Considerations

Maximal Overlap Discrete Wavelet Packet Transform: Perform nondecimated wavelet packet analysis on arbitrary-length signals

This release adds support for the maximal overlap discrete wavelet packet transform (MODWPT) for 1-D signals. You can decompose signals using `modwpt` and invert the transform using `imodwpt`. Also, you can obtain MODWPT details using `modwptdetails`. For an example of using wavelet packets, see Wavelet Packets: Decomposing the Details.

Wavelet Synchrosqueezing: Sharpen time-frequency estimates and extract signal modes

This release adds support for the wavelet synchrosqueezed transform and mode extraction for 1-D signals. Wavelet synchrosqueezing is a time-frequency reassignment technique that enables you to reconstruct the signal from the reassigned transform. This technique enables you to extract and visualize oscillatory modes in the signal. To obtain the synchrosqueezed transform of a signal, use `wsst`. To invert the transform, use `iwsst`. You can determine or extract time-frequency ridges in the synchrosqueezed transform with `wsstridge`. For an example of synchrosqueezing, see Time-Frequency Reassignment and Mode Extraction with Synchrosqueezing.

Wavelet Coherence: Compare time-varying frequency content between signals

This release adds the `wcoherence` function, which computes the magnitude-squared wavelet coherence of two input signals. The `wcoherence` function also computes the wavelet cross spectrum. Wavelet coherence is useful for detecting common time-localized oscillations in nonstationary, bivariate signals. `wcoherence` also provides visualizations that show the magnitude-squared coherence, cross-spectrum phase, and the cone of influence. The phase plot is helpful in determining the lead-lag relationships between the signals. The cone of influence demonstrates where edge effects become significant. For an example of using `wcoherence`, see Compare Time-Frequency Content in Signals with Wavelet Coherence.

Compatibility Considerations

`wcoher` is not recommended. Update code that uses `wcoher` to use `wcoherence` instead.

Fejer-Korovkin filter with 18 coefficients

This release adds support for Fejer-Korovkin (`fejerkorovkin`) scaling and wavelet filters with 18 coefficients. The valid short name is `'fk18'`.

Maximum derivative output of gauswavf and cgauwavf

As of R2016a, the highest order derivative supported for the Gaussian (`gauswavf`) and complex Gaussian wavelet (`cgauwavf`) is 8.

Compatibility Considerations

Specifying a derivative order greater than 8 produces an error. In code that uses `gauswavf` or `cgauwavf`, update these functions to use a derivative value from 1 to 8. The requirement to have Symbolic Math Toolbox™ has been removed.

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`wcoher`	Still runs	`wcoherence`	Replace all instances of `wcoher` with `wcoherence`.
`gauswavf` and `cgauwavf`	Errors when the order of the derivative is greater than 8		Update instances of `gauswavf` and `cgauwavf` to use a maximum derivative value of 8.

R2015b

Version: 4.15

New Features

Bug Fixes

Maximal Overlap Discrete Wavelet Transform: Perform nondecimated analysis on arbitrary-length signals and obtain multiscale variance and correlation estimates

This release adds support for the maximal overlap discrete wavelet transform (MODWT) for 1-D signals. You can decompose signals using `modwt` and invert the transform using `imodwt`. Additionally, you can obtain a MODWT-based multiresolution analysis using `modwtmra`. You can also obtain wavelet variance, correlation, and cross-correlation sequence estimates with confidence intervals using `modwtvar`, `modwtcorr`, and `modwtxcorr`.

Frequency-Localized Wavelets: Perform more frequency-localized discrete and continuous wavelet analysis

This release adds new frequency-localized wavelets for continuous and discrete wavelet analysis. The bump wavelet is a frequency-localized wavelet with an adjustable center frequency and bandwidth.

You can use the bump wavelet with `cwtft`. For discrete decimated and nondecimated wavelet or wavelet packet analysis, use the new Fejer-Korovkin family of frequency-localized orthogonal wavelets. To obtain information on the Fejer-Korovkin wavelets, enter `waveinfo('fk')` at the MATLAB command prompt. To obtain the Fejer-Korovkin filters, use `wfilters` or `fejerkorovkin`. You can specify the Fejer-Korovkin filters in all discrete wavelet and wavelet packet command line and interactive applications using the short name, `'fk'` with a valid filter number. For example, `wavedec(data,N,'fk8')` or `modwt(data,'fk8')`.

Time-Frequency Analysis: Convert scale to frequency to interpret the continuous wavelet transform as a time-frequency transform

This release adds scale-to-frequency conversion for `cwtft` and `cwt`. `cwtft` returns the scale-to-frequency conversions as a field in the structure array output. `cwt` accepts an optional sampling interval input, which enables you to output scale-to-frequency conversions. This release also adds a featured example, Time-Frequency Analysis with the Continuous Wavelet Transform.

Application examples: Analyze financial data and physiologic signals using wavelets

This release introduces new examples for analyzing financial data and physiologic signals using wavelets. The financial example, Wavelet Analysis of Financial Data, shows how you can use wavelets to analyze multiscale volatility in financial time series data and explore multiscale correlation in bivariate time series data. The physiologic signal analysis example, Wavelet Analysis of Physiologic Signals, showcases QRS detection in the electrocardiogram using wavelets, wavelet coherence, and time-frequency analysis.

R2015a

Version: 4.14.1

Bug Fixes

Compatibility Considerations

Functionality being removed or changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`ndwt`, `indwt`, `ndwt2`, `indwt2`	Errors	`swt`, `iswt`, `swt2`, `iswt2`	Replace all instances of `ndwt`, `indwt`, `ndwt2`, and `indwt2` with the corresponding function for the stationary wavelet transform.

R2014b

Version: 4.14

Bug Fixes

R2014a

Version: 4.13

New Features

Bug Fixes

Dual-Tree Wavelet Transforms

This release introduces a new example, Dual-Tree Wavelet Transforms, which demonstrates the advantages of the dual-tree discrete wavelet transform (DWT) over the critically sampled DWT. The example illustrates the approximate shift invariance and directional selectivity of the complex dual-tree wavelet transform. These properties enable the dual-tree wavelet transform to outperform the critically sampled DWT in a number of applications.

R2013b

Version: 4.12

New Features

Bug Fixes

Two-Dimensional Continuous Wavelet Transform (2-D CWT)

This release introduces the 2-D continuous wavelet transform (CWT) for images. The 2-D CWT provides information about images at specified scales, rotation angles, and positions in the plane. Applications of the 2-D CWT include:

- Fault detection in images
- Object recognition
- Fringe pattern profilometry

For information on how to implement the 2-D CWT at the MATLAB command line, see `cwtft2`.

To use `cwtft2` in the Wavelet Toolbox interactive tool, enter

```
>> wavemenu
```

Then, from the **Two-dimensional** tools section, select **Continuous Wavelet Transform 2-D**. See 2-D Continuous Wavelet Transform App for more information on the 2-D CWT app.

`cwtft2` supports both isotropic and anisotropic 2-D wavelets. Use isotropic wavelets to perform pointwise analysis in images or when oriented features are not relevant. Use anisotropic wavelets when your goal is to detect oriented features.

`cwtft2` implements the 2-D CWT using the 2-D discrete Fourier transform. Use `cwtftinfo2` to obtain the 2-D Fourier transforms of the supported analyzing wavelets.

Dual-Tree Transforms and Double-Density Transforms

This release introduces two types of 1-D and 2-D oversampled (frame) wavelet perfect reconstruction filter banks. For 1-D wavelet analysis, use `dddtree` to obtain the following wavelet transforms:

- Complex dual-tree
- Double-density
- Complex dual-tree double-density

For 2-D wavelet analysis, use `dddtree2` to obtain the following wavelet transforms:

- Double-density
- Real oriented dual-tree
- Complex oriented dual-tree
- Real oriented double-density dual-tree
- Complex oriented double-density dual-tree

The dual-tree and double-density transforms mitigate a number of shortcomings of the critically sampled discrete wavelet transform. The double-density and dual-tree transforms achieve directional selectivity and approximate shift invariance with significantly less computational cost than the undecimated discrete wavelet transform.

R2013a

Version: 4.11

Bug Fixes

R2012b

Version: 4.10

Bug Fixes

R2012a

Version: 4.9

New Features

Matching Pursuit

In R2012a you can decompose a 1-D signal in a dictionary of time/frequency or time/scale atoms with matching pursuit.

Representing a signal in a union of time-frequency/time-scale bases can provide sparser signal representations than attainable with any single basis. Matching pursuit uses iterative greedy algorithms to reduce the computational complexity of searching through a redundant dictionary.

Wavelet Toolbox software supports basic matching pursuit, orthogonal matching pursuit, and weak orthogonal matching pursuit at the command line with `wmpdictionary` and `wmpalg`. You can also perform matching pursuit with the interactive `wavemenu` tool.

You can build dictionaries using several internally supported options or provide your own custom dictionaries. See Matching Pursuit for background information and examples.

R2011b

Version: 4.8

New Features

Bug Fixes

Compatibility Considerations

Fourier Transform Based Continuous Wavelet Transform GUI

In R2011b, you can compute the Fourier transform based continuous wavelet transform (CWT) and inverse CWT using the Wavelet Toolbox graphical user interface `wavemenu`. To access these graphical tools, enter `wavemenu` at the command line, and select **Continuous Wavelet 1-D (using FFT)**.

Inverse Continuous Wavelet Transform Using Linear Scales

In R2011b, you can compute the inverse continuous wavelet transform (CWT) for a wider class of analyzing wavelets using `icwtlin`. `icwtlin` returns the inverse for CWT coefficients obtained at linearly spaced scales. `icwtlin` supports the output of `cwtft` and the output of `cwt` for a select number of wavelets. See `icwtlin` for detailed information.

MATLAB Code Generation Support for Denoising and Compression GUIs

In R2011b, you can generate MATLAB code for 1-D and 2-D discrete wavelet transforms (DWT), stationary wavelet transforms (SWT), and wavelet packet transforms. You can denoise or compress a signal or image in the GUI and export the MATLAB code to implement that operation at the command line. This approach allows you to set denoising thresholds or compression ratios aided by visualization tools and save the commands to reproduce those operations at the command line. See Generating MATLAB Code from Wavelet Toolbox GUI for examples.

Signal Reconstruction from Continuous Wavelet Transform Coefficients Demo

R2011b includes a new demo illustrating signal reconstruction using the continuous wavelet transform (CWT). The demo emphasizes the use of the CWT to analyze a signal and reconstruct a time- and scale-based approximation with select coefficients using the inverse CWT. See `Signal Reconstruction from Continuous Wavelet Transform Coefficients` for details.

Changes in Fourier Transform Based Continuous Wavelet Transform Defaults for Derivative of Gaussian (DOG) and Paul Wavelets

In R2011b, the default values for the smallest scale, scale increment, and number of scales have changed in `cwtft` for the derivative of Gaussian (DOG) and Paul wavelets. The change in the defaults also affects the Mexican hat wavelet, which is a special case of the DOG wavelet. In R2011b, the default value of the smallest scale for the Paul and DOG wavelets is `2*dt`, where `dt` is the sampling period. The default scale increment, `ds`, is 0.4875. The default number of scales is `fix(log2(length(sig))/ds)+1` for the Paul wavelet and `max([fix(log2(length(sig))/ds),1])` for the DOG wavelets, where `sig` is the input signal.

Compatibility Considerations

`cwtft` was introduced in R2011a. In that release, the default smallest scales for the DOG and Paul wavelets are `dt/8` and `dt` respectively, where `dt` is the sampling interval. The default scale increment is 0.5. The default number of scales is `fix(1.5*log2(length(sig))/ds)+1` for the Paul wavelet. For DOG wavelets, the default number of scales is `fix(1.25*log2(length(sig))/ds)+1`, where `sig` is the input signal. You can obtain results in

R2011b using `cwtft` with the DOG and Paul wavelets identical to results in R2011a with the default values. To do so, specify the smallest scale, scale increment, and number of scales in a structure or cell array. See `cwtft` for details.

R2011a

Version: 4.7

New Features

Bug Fixes

Inverse Continuous Wavelet Transform

In R2011a, you can compute the inverse continuous wavelet transform (CWT) using an FFT-based algorithm. The inverse CWT allows you to synthesize approximations to your 1D signal based on selected scales. The inverse CWT is only supported for coefficients obtained using the FFT-based CWT. See `icwtft` and `cwtft` for details.

FFT-based Continuous Wavelet Transform

In R2011a, you can compute the continuous wavelet transform (CWT) using an FFT-based algorithm with `cwtft`. The CWT computed using an FFT algorithm supports the computation of the inverse CWT. See `cwtft` and `icwtft` for details. Only select wavelets are valid for use with `cwtft`. See `cwtftinfo` for a list of supported wavelets.

Pattern-adapted Wavelets for Signal Detection Demo

In R2011a there is a new demo using pattern adapted wavelets for signal detection. You can view this demo here: `Pattern adapted wavelets for signal detection`. The Wavelet Toolbox software enables you to design admissible wavelets based on the pattern you wish to detect. Designing a valid wavelet based on your desired pattern allows you to exploit the optimality of matched filtering in the framework of the CWT. The demo illustrates this process on simulated data and human EEG recordings.

R2010b

Version: 4.6

New Features

Bug Fixes

Cone of Influence for Continuous Wavelet Transform

In R2010b, you can compute the cone of influence (COI) for the continuous wavelet transform (CWT) of a signal. At each scale, the COI determines the set of CWT coefficients influenced by the value of the signal at a specified position. The COI provides an important visual aid in interpreting the CWT. By overlaying the cone of influence on the CWT image, you can determine which CWT coefficients each value of the signal affects at every scale. See `conofinf` for details.

Wavelet Cross Spectrum and Coherence

In R2010b, you can estimate the wavelet cross spectrum and wavelet coherence of two time series. The wavelet cross spectrum and coherence provide wavelet-based alternatives for the Fourier-based cross spectrum and coherence. These wavelet estimators are suitable for nonstationary signals. Using a complex-valued analyzing wavelet, you can also examine intervals in the time-scale plane where the two time series exhibit common phase behavior. See `wcoher` and the new demo `Wavelet Coherence` for details.

Wavelet Packet Spectrum

In R2010b, you can compute the wavelet packet spectrum with `wpspectrum`. The wavelet packet spectrum provides a time-frequency analysis of a time series. The wavelet packet spectrum is useful as wavelet-based counterpart of the short-time Fourier transform.

Natural and Frequency Ordering of Wavelet Packet Terminal Nodes

In R2010b, you can order the wavelet packet transform terminal nodes by natural (Payley) or frequency (sequency) order. See `otnodes` for details.

Image and Signal Approximation Quality Metrics

In R2010b, you can measure the quality of your signal or image approximation using a number of widely-used quality metrics. These metrics include: the peak signal-to-noise ratio (PSNR), the mean square error (MSE), the maximum absolute error, and the energy ratio of the approximation to the original. See `measerr` for details.

R2010a

Version: 4.5

New Features

Bug Fixes

3-D Discrete Wavelet Transform

This release adds new functions and a GUI to support the 3-D discrete wavelet transform. This new functionality lets you decompose, analyze, and display a 3-D object using a different wavelet for each dimension. The new functions are: `dwt3`, `idwt3`, `wavedec3`, and `waverec3`. A demo (`wavelet3ddemo`) is also included.

Nondecimated Wavelet Transform

New nondecimated wavelet transform functions support signals of arbitrary size and different extension modes. Previous functionality had two limitations: signal length had to equal a power of 2 and the only allowable extension mode was periodized. The new functions are: `ndwt`, `indwt`, `ndwt2`, and `indwt2`. A demo (`ndwtdemo`) is also included.

New Denoising Function

The new `cmddenoise` function uses interval-dependent denoising to compute the denoised signal and coefficients. This allows you to apply different denoising thresholds to different portions of the signal, which is typically nonuniform. You can also export thresholds from the GUI and use them in the `cmddenoise` function. The toolbox includes a denoising demo (`cmddenoise`).

R2009b

Version: 4.4.1

Bug Fixes

R2009a

Version: 4.4

New Features

Bug Fixes

New Demos

The toolbox now includes three new demos:

`Adding a New Wavelet`

`Wavelet Interval-dependant Denoising`

`Wavelet Scalograms`

R2008b

Version: 4.3

New Features

Bug Fixes

True Image Compression Support

The new `wcompress` functions lets you compress 2D image data. You can also interactively compress images using the new Two-Dimensional Images Compression GUI.

New Demo

The toolbox now includes a new codepad demo on image compression.

R2008a

Version: 4.2

New Features

Bug Fixes

True Color Images Support

The toolbox can now process true color images. All major toolbox GUIs and all of the 2D-oriented command line functions have been also updated and support true color images.

New Extension Modes for Continuous Wavelets

The new `cwtext` function lets you calculate 1D continuous wavelet parameters using extension parameters.

New Norms Calculation

The Multisignal 1D GUI and other related GUIs now include 1-norm, 2-norm, and inf-norm calculations.

Wavelet Families Display

A new function, `waveletfamilies`, displays all the available wavelet families and their properties.

Single Data Type Support

The `swt2` and `iswt2` functions now support single data types.

New Demos

The toolbox now includes the following new codepad demos:

- Multiscale Principal Component Analysis
- Multivariate Denoising

R2007b

Version: 4.1

New Features

Bug Fixes

Importing and Exporting between GUIs and Workspace

You can now import data from the workspace to all toolbox GUIs and export data from all toolbox GUIs to the workspace. Use **Import from Workspace** and **Export to Workspace**, respectively, on the GUI's **File** menu.

Scalograms for Continuous Wavelet Transforms

The ability to compute scalograms of the wavelet coefficients in continuous wavelet analysis has been added as an option to the `cwt` function. You can also pass the structure produced by `cwt` directly to the new `wscalogram` function. Scalograms show the percentage of energy in each wavelet coefficient.

Constructing Clusters from Hierarchical Cluster Trees

You can now construct clusters from hierarchical cluster trees in multisignal analysis using the new `mdwtcluster` function.

R2007a

Version: 4.0

New Features

Bug Fixes

1D Multisignal Analysis, Compression, and Denoising Added

The following command-line functions for 1D multisignal analysis, compression, and denoising have been added to the toolbox:

`mswcmp`	Multisignal 1D compression using wavelets.
`mswcmpscr`	Multisignal 1D wavelet compression scores.
`mswcmptp`	Multisignal 1D compression thresholds and perf.
`mswden`	Multisignal 1D denoising using wavelets.
`mswthresh`	Performs Multisignal 1D thresholding.

1D Multisignal Wavelet and Clustering Added

The following command-line functions for 1D multisignal wavelets and clustering have been added to the toolbox:

`chgwdeccfs`	Change Multisignal 1D decomposition coeffs
`mdwtdec`	Multisignal 1D wavelet decomposition
`mdwtrec`	Multisignal 1D wavelet reconstruction.
`wdecenergy`	Multisignal 1D decomposition energy repartition

Note Clustering analyses require that Statistics Toolbox™ is installed.

Wavelet 1D Multisignal Analysis GUI Added

A graphical user interface for 1-D multisignal analysis has been added. To start this GUI, select **Multisignal Analysis 1D** from the `wavemenu` dialog.

R2006b

Version: 3.1

New Features

Bug Fixes

Multivariate De-noising Added

A new command-line function (`wmulden`) and a new GUI (**Multivariate Denoising** from the `wavemenu` initial window) for de-noising a matrix of signals have been added. Both the function and GUI take into account the signals themselves and the correlations between the signals. A two-step process is used. First, a change of basis is performed to deal with noise spatial correlation de-noising in the new basis. Then, a principal component analysis is performed to take advantage of the deterministic relationships between the signals, leading to an additional de-noising effect.

Multiscale Principal Component Analysis Added

A new command-line function (`wmspca`) and a new GUI (**Multiscale Princ. Comp. Analysis** from the `wavemenu` initial window) for simplifying a matrix of signals have been added. Both the function and GUI take into account the signals themselves and the correlations between the signals. The multiscale principal component analysis mixes wavelet decompositions and principal component analysis.

New Demos

The following new demos are added:

- Continuous and Discrete Wavelet Analysis
- Detecting Discontinuities and Breakdown Points
- De-Noising Signals and Images
- Data Compression using 2D Wavelet Analysis
- Image Fusion
- Detecting Self-Similarity
- Wavelet Packets: Decomposing the Details

R2006a

Version: 3.0.4

No New Features or Changes

R14SP3

Version: 3.0.3

No New Features or Changes

R14SP2

Version: 3.0.2

No New Features or Changes