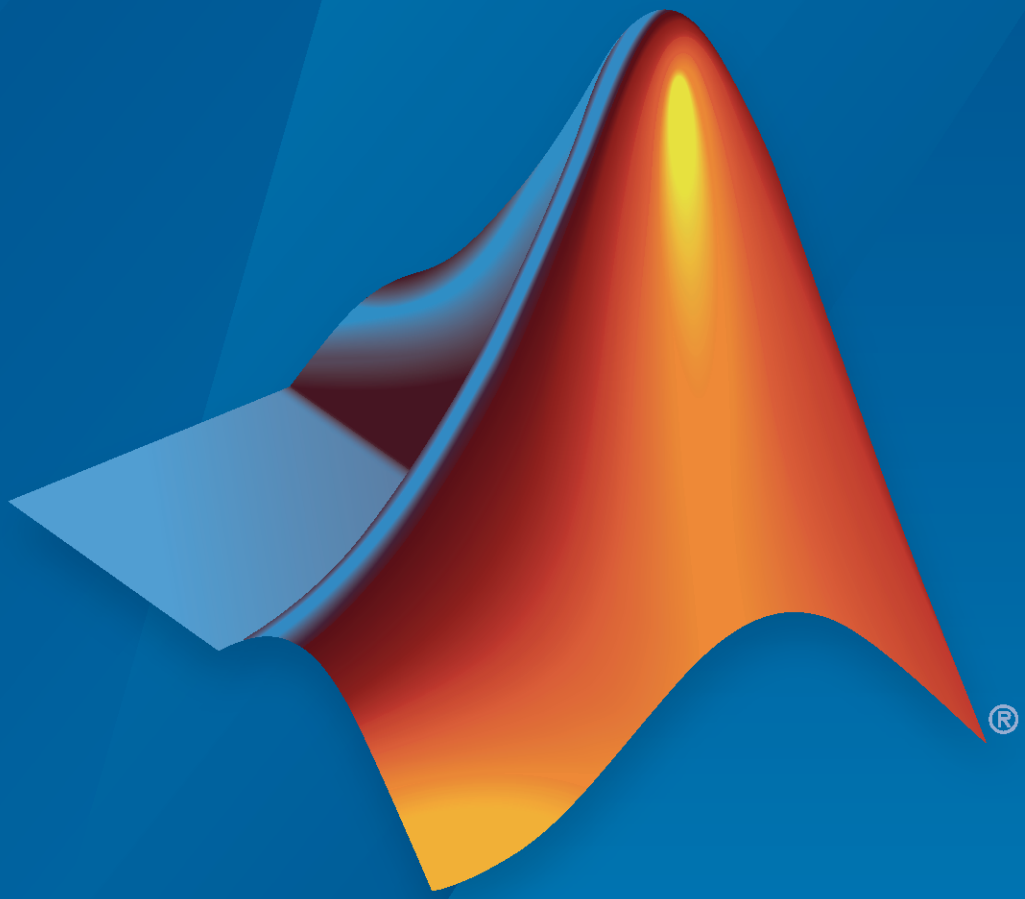
Wavelet Toolbox™ Release Notes



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

R2024a

R2023b

R2023a

R2022b

R2022a

R2021b

R2021a

R2020b

R2020a

R2019b

R2019a

R2024a

★Wavelet Signal Analyzer: Plot variance of nondecimated wavelet coefficients

For nondecimated wavelet decompositions, you can use Wavelet Signal Analyzer to plot the variance estimates of the coefficients at all levels. You can visualize how the variance and confidence intervals change as you compress the signal. You can:

- Specify a biased or unbiased estimator.
- Compute wavelet variance confidence intervals with specified confidence level based on a chi-square or Gaussian distribution.

Wavelet Synchrosqueezing: Support for analytic Morse wavelets

The functions `wsst`, `iwsst` and `wsstridge` now support analytic Morse wavelets. To learn more about Morse wavelets, see “Morse Wavelets”.

You can also now obtain, as a structure array, the continuous wavelet transform (CWT) filter bank parameters the `wsst` function uses to compute the wavelet synchrosqueezed transform. To analyze filter bank properties such as the filter frequency responses, you can use the parameters to create a `cwtfilterbank` object. The `wsstridge` function accepts the parameters as an input argument.

★Feature Extraction: Extract time-frequency features of signals

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

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

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

You must have Signal Processing Toolbox™ to use `signalTimeFrequencyFeatureExtractor`.

GPU Computing: Accelerate wavelet synchrosqueezing on your GPU

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

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).

Multisignal 1-D Discrete Decimated Wavelet Transforms: Support for single-precision and complex-valued data

The `mdwtdec` and `mdwtrec` functions now support single-precision and complex-valued data.

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

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

- `waveletScattering2`
- `wdcbm2`
- `wsst`

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

Deep Learning Example: Detect anomalies using wavelet scattering with deepSignalAnomalyDetector

This release introduces an example, “Anomaly Detection Using Convolutional Autoencoder with Wavelet Scattering Sequences”, that shows how to detect anomalies in acoustic data using the `deepSignalAnomalyDetector` and `waveletScattering` objects.

Relative-Velocity Changes Example: Use time-frequency resolution properties of the CWT to determine frequency-dependent delays in signals

This release introduces an example, “Relative Velocity Changes in Seismic Waves Using Time-Frequency Analysis”, that shows how to use wavelet coherence and the wavelet cross spectrum to characterize frequency-dependent delays in nonstationary signals.

⚠Functionality being removed or changed

littlewoodPaleySum syntax has changed

Behavior change

The `littlewoodPaleySum` function syntax has changed. The object function of the `waveletScattering2` object now returns the spatial frequencies of the Littlewood-Paley sum in the *x*- and *y*-directions as separate vectors. Previously, the function returned the spatial frequencies as a two-column matrix, where the first and second columns were the spatial frequencies in the *x*- and *y*-directions, respectively. Because of this change, the function now supports 2-D wavelet scattering networks configured with a non-square image size.

Original Code in R2023b or Earlier	Result	Updated Code in R2024a
`[lpsum,f] = littlewoodPaleySum(sf)`	Runs but the second output argument has a different interpretation.	`[lpsum,fx,fy] = littlewoodPaleySum(sf)`

R2023b

Wavelet Signal Analyzer App: Support for new transforms

The Wavelet Signal Analyzer app now supports these transforms:

- Decimated discrete wavelet transform
- Decimated discrete wavelet packet transform
- Maximal overlap discrete wavelet packet transform

Wavelet Signal Analyzer App: Display autocorrelation of wavelet coefficients

For all decomposition methods, the Wavelet Signal Analyzer app now enables you to plot the autocorrelation of the wavelet coefficients at the decomposition level or terminal node that you specify. You can vary the order of the moving average filter and the number of lags. The plot includes the upper and lower bounds of the 95% confidence interval.

Wavelet Signal Analyzer App: Apply level- and node-specific thresholds for compression

For all decomposition methods, the Wavelet Signal Analyzer app now enables you to apply different thresholds to individual decomposition levels or terminal nodes. You can then export the results to your workspace. You can also generate a script to reproduce the compressed signal in your workspace.

Wavelet Image Analyzer App: Decompose images using the continuous wavelet transform

You can now use the Wavelet Image Analyzer app to visualize the continuous wavelet transform (CWT) decomposition of images. With the app, you can:

- Change the wavelet and normalization.
- Specify scales based on the maximum desired scale or use custom scales.
- Change the voices per octave.
- Specify the number of desired angles for anisotropic wavelets or specify custom angles.
- View the decomposition at all scales for a specific angle or at all angles for a specific scale.
- Display either the magnitude and phase or real and imaginary parts of the decomposition.

Continuous Wavelet Transform: Reconstruct signals from their scalograms

This release introduces the `cwtmag2sig` function, which allows you to reconstruct a signal starting from the magnitude of its continuous wavelet transform. The `cwtmag2sig` function uses the gradient descent method with an optimizer that you select from Deep Learning Toolbox™.

You must have Deep Learning Toolbox to use this functionality.

The `cwtmag2sig` function also supports `gpuArray` objects, enabling you to accelerate reconstruction on your GPU.

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).

1-D Discrete Wavelet Packet Transforms: Compute wavelet transforms of complex-valued data

These functions now support complex-valued data:

- Decimated Discrete Wavelet Packet Transforms — `dwpt` and `idwpt`
- Maximal Overlap Discrete Wavelet Packet Transforms — `modwpt`, `imodwpt`, and `modwptdetails`

1-D Discrete Wavelet Transforms: Specify signal extension mode

You can specify the signal extension mode for the `wavedec`, `waverec`, and `appcoef` functions. The extension modes provide options for dealing with the problem of border distortion in signal analysis. For a list of extension modes, see `dwtmode`.

GPU Code Generation: Automatically generate GPU code for time-frequency analysis functions

The `wcoherence` and `wvd` functions now support CUDA® code generation.

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

GPU Computing: Accelerate dwtleader function on your GPU

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

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).

⚠Functionality being removed or changed

Wavelet Analyzer App has been removed

Errors

The Wavelet Analyzer app has been removed.

- To perform signal multiresolution analysis using wavelet and data-adaptive techniques, use the Signal Multiresolution Analyzer app.
- To analyze images using the discrete wavelet transform or the continuous wavelet transform, use the Wavelet Image Analyzer app.
- To analyze and compress signals using discrete wavelet or wavelet packet transforms, use the Wavelet Signal Analyzer app.
- To analyze and denoise signals using the discrete wavelet transform, use the Wavelet Signal Denoiser app.

- To perform time-frequency analysis of signals using the continuous wavelet transform, use the Wavelet Time-Frequency Analyzer app.

cwtft2 Function: Plotting is not recommended
Still runs

Plotting continues to work in `cwtft2`, but is no longer recommended. Use the Wavelet Image Analyzer app instead.

R2023a

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

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	New 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`
	`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

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

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

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

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

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

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`.