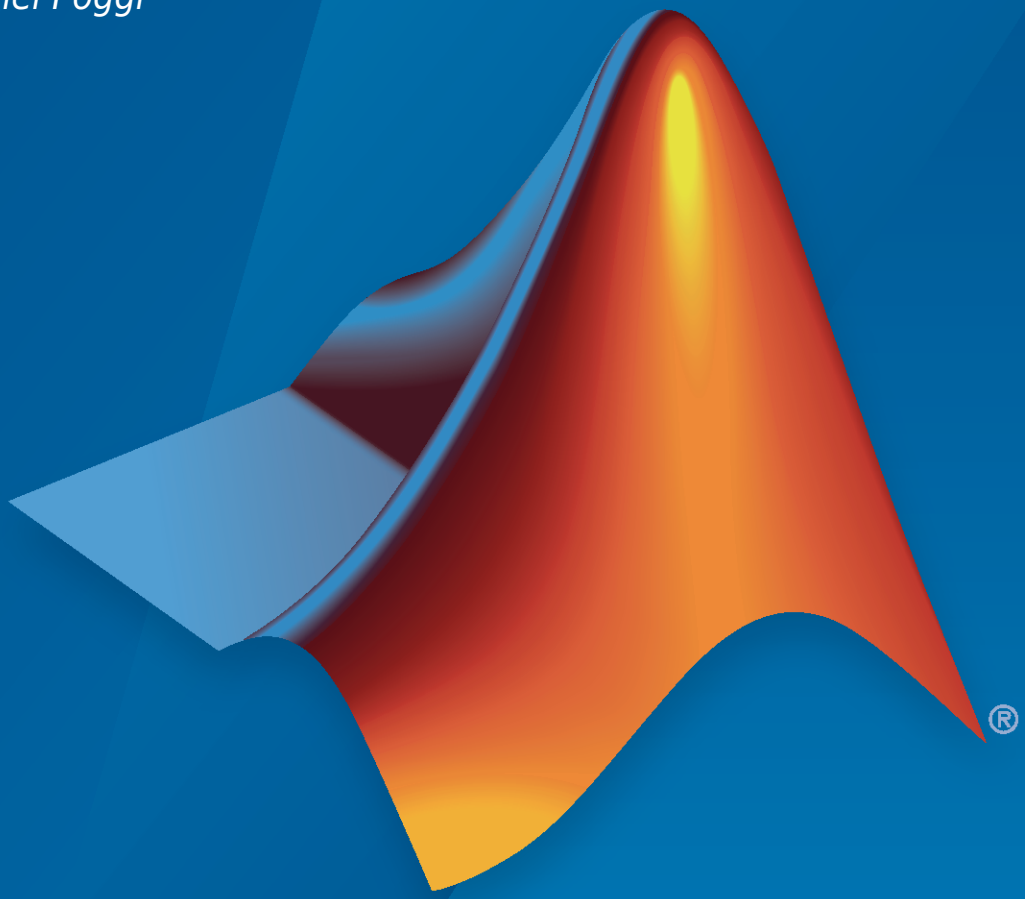
Wavelet Toolbox™

Getting Started Guide

Michel Misiti
Yves Misiti
Georges Oppenheim
Jean-Michel Poggi



MATLAB®

R2020a

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™ Getting Started Guide

© COPYRIGHT 1997–2020 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

March 1997	First printing	New for Version 1.0
September 2000	Second printing	Revised for Version 2.0 (Release 12)
June 2001	Online only	Revised for Version 2.1 (Release 12.1)
July 2002	Online only	Revised for Version 2.2 (Release 13)
June 2004	Online only	Revised for Version 3.0 (Release 14)
July 2004	Third printing	Revised for Version 3.0
October 2004	Online only	Revised for Version 3.0.1 (Release 14SP1)
March 2005	Online only	Revised for Version 3.0.2 (Release 14SP2)
June 2005	Fourth printing	Minor revision for Version 3.0.2
September 2005	Online only	Minor revision for Version 3.0.3 (Release R14SP3)
March 2006	Online only	Minor revision for Version 3.0.4 (Release 2006a)
September 2006	Online only	Revised for Version 3.1 (Release 2006b)
March 2007	Online only	Revised for Version 4.0 (Release 2007a)
September 2007	Online only	Revised for Version 4.1 (Release 2007b)
October 2007	Fifth printing	Revised for Version 4.1
March 2008	Online only	Revised for Version 4.2 (Release 2008a)
October 2008	Online only	Revised for Version 4.3 (Release 2008b)
March 2009	Online only	Revised for Version 4.4 (Release 2009a)
September 2009	Online only	Minor revision for Version 4.4.1 (Release 2009b)
March 2010	Online only	Revised for Version 4.5 (Release 2010a)
September 2010	Online only	Revised for Version 4.6 (Release 2010b)
April 2011	Online only	Revised for Version 4.7 (Release 2011a)
September 2011	Online only	Revised for Version 4.8 (Release 2011b)
March 2012	Online only	Revised for Version 4.9 (Release 2012a)
September 2012	Online only	Revised for Version 4.10 (Release 2012b)
March 2013	Online only	Revised for Version 4.11 (Release 2013a)
September 2013	Online only	Revised for Version 4.12 (Release 2013b)
March 2014	Online only	Revised for Version 4.13 (Release 2014a)
October 2014	Online only	Revised for Version 4.14 (Release 2014b)
March 2015	Online only	Revised for Version 4.14.1 (Release 2015a)
September 2015	Online only	Revised for Version 4.15 (Release 2015b)
March 2016	Online only	Revised for Version 4.16 (Release 2016a)
September 2016	Online only	Revised for Version 4.17 (Release 2016b)
March 2017	Online only	Revised for Version 4.18 (Release 2017a)
September 2017	Online only	Revised for Version 4.19 (Release 2017b)
March 2018	Online only	Revised for Version 5.0 (Release 2018a)
September 2018	Online only	Revised for Version 5.1 (Release 2018b)
March 2019	Online only	Revised for Version 5.2 (Release 2019a)
September 2019	Online only	Revised for Version 5.3 (Release 2019b)
March 2020	Online only	Revised for Version 5.4 (Release 2020a)

Contents

1 Getting Started with Wavelet Toolbox Software

Wavelet Toolbox Product Description **1-2**
Key Features **1-2**

Installing Wavelet Toolbox Software **1-3**
System Recommendations **1-3**
Platform-Specific Details **1-3**

What is a Wavelet? **1-5**
Localize Discontinuity in Sine Wave **1-5**

Choose a Wavelet **1-10**
Time-Frequency Analysis **1-10**
Multiresolution Analysis **1-13**
Denoising **1-17**
Compression **1-17**
General Considerations **1-18**

Extremal Phase **1-21**

Effect of Wavelet Support on Noisy Data **1-23**

Comparing MODWT and MODWTMRA **1-27**

Image Reconstruction with Biorthogonal Wavelets **1-32**

Wavelets and Vanishing Moments **1-35**

Continuous and Discrete Wavelet Transforms **1-38**
Guidelines for Continuous Wavelet Transform vs. Discrete Wavelet Transform **1-40**

Nonstationary Gabor Frames and the Constant-Q Transform **1-42**
Decomposing the Time-Frequency Plane **1-42**
Constant-Q Transform **1-43**
References **1-44**

Haar Transforms for Time Series Data and Images **1-46**

From Fourier Analysis to Wavelet Analysis **1-58**
Inner Products **1-58**
Fourier Transform **1-59**
Short-Time Fourier Transform **1-61**

Continuous Wavelet Transform and Scale-Based Analysis **1-63**
Definition of the Continuous Wavelet Transform **1-63**
Scale **1-64**
Shifting **1-66**
CWT as a Windowed Transform **1-66**

Continuous Wavelet Transform as a Bandpass Filter **1-68**
CWT as a Filtering Technique **1-68**
DFT-Based Continuous Wavelet Transform **1-70**

Inverse Continuous Wavelet Transform **1-71**

Interpreting Continuous Wavelet Coefficients **1-73**

Critically-Sampled Discrete Wavelet Transform **1-85**
One-Stage Filtering: Approximations and Details **1-85**
Multiple-Level Decomposition **1-87**

Critically-Sampled Wavelet Reconstruction **1-88**
Reconstruction Filters **1-88**
Reconstructing Approximations and Details **1-89**
Wavelets From Conjugate Mirror Filters **1-90**

Wavelet Synchrosqueezing **1-92**
What is Wavelet Synchrosqueezing? **1-92**
Algorithm **1-92**
Required Component Separation **1-93**
Examples **1-94**

Introduction to Wavelet Families **1-99**
Haar **1-99**
Daubechies **1-99**
Biorthogonal **1-100**
Coiflets **1-101**
Symlets **1-102**
Morlet **1-102**
Mexican Hat **1-103**
Meyer **1-103**
Other Real Wavelets **1-103**
Complex Wavelets **1-104**

References **1-105**

2 Using Wavelets

Introduction to Wavelet Toolbox App and Functions **2-2**

Wavelets: Working with Images **2-3**
Understanding Images in the MATLAB Environment **2-3**
Indexed Images **2-3**
Wavelet Decomposition of Indexed Images **2-4**

RGB (Truecolor) Images 2-5
Wavelet Decomposition of Truecolor Images 2-5
Other Images 2-5
Image Conversion 2-5

Density Estimation Using Wavelets 2-8
1-D Estimation Using the Wavelet Analyzer App 2-8
Importing and Exporting Information from the Wavelet Analyzer App . . . 2-11

1-D Wavelet Coefficient Selection Using the Wavelet Analyzer App 2-13

2-D Wavelet Coefficient Selection Using the Wavelet Analyzer App 2-20

1-D Extension Using the Command Line 2-25

2-D Extension Using the Command Line 2-26

Image Fusion 2-27
Image Fusion Using the Command Line 2-27
Image Fusion Using the Wavelet Analyzer App 2-29
Saving the Synthesized Image 2-32

1-D Fractional Brownian Motion Synthesis 2-33

New Wavelet for CWT 2-35

Additional Wavelet GUIs 2-38
1-D Extension Using the Signal Extension Tool 2-38
2-D Extension Using the Image Extension Tool 2-41
Fractional Brownian Motion Synthesis Tool 2-42
New Wavelet for CWT Using the Pattern Adapted Admissible Wavelet Design Tool 2-45

3 Getting Started with Wavelet Analysis

Wavelet Families and Properties 3-2

Visualizing Wavelets, Wavelet Packets, and Wavelet Filters 3-4

Continuous Wavelet Analysis 3-7

Continuous Wavelet Transform and Inverse Continuous Wavelet Transform 3-9

Discrete Wavelet Analysis 3-13
1-D Wavelet Denoising 3-13
2-D Decimated Discrete Wavelet Analysis 3-15
Nondecimated Discrete Wavelet Analysis 3-17

Lifting 3-20

Critically Sampled Wavelet Packet Analysis **3-25**

Getting Started with Wavelet Toolbox Software

- “Wavelet Toolbox Product Description” on page 1-2
- “Installing Wavelet Toolbox Software” on page 1-3
- “What is a Wavelet?” on page 1-5
- “Choose a Wavelet” on page 1-10
- “Extremal Phase” on page 1-21
- “Effect of Wavelet Support on Noisy Data” on page 1-23
- “Comparing MODWT and MODWTMRA” on page 1-27
- “Image Reconstruction with Biorthogonal Wavelets” on page 1-32
- “Wavelets and Vanishing Moments” on page 1-35
- “Continuous and Discrete Wavelet Transforms” on page 1-38
- “Nonstationary Gabor Frames and the Constant-Q Transform” on page 1-42
- “Haar Transforms for Time Series Data and Images” on page 1-46
- “From Fourier Analysis to Wavelet Analysis” on page 1-58
- “Continuous Wavelet Transform and Scale-Based Analysis” on page 1-63
- “Continuous Wavelet Transform as a Bandpass Filter” on page 1-68
- “Inverse Continuous Wavelet Transform” on page 1-71
- “Interpreting Continuous Wavelet Coefficients” on page 1-73
- “Critically-Sampled Discrete Wavelet Transform” on page 1-85
- “Critically-Sampled Wavelet Reconstruction” on page 1-88
- “Wavelet Synchrosqueezing” on page 1-92
- “Introduction to Wavelet Families” on page 1-99
- “References” on page 1-105

Wavelet Toolbox Product Description

Analyze and synthesize signals and images using wavelets

Wavelet Toolbox provides functions and apps for analyzing and synthesizing signals and images. The toolbox includes algorithms for continuous wavelet analysis, wavelet coherence, synchrosqueezing, and data-adaptive time-frequency analysis. The toolbox also includes apps and functions for decimated and nondecimated discrete wavelet analysis of signals and images, including wavelet packets and dual-tree transforms.

Using continuous wavelet analysis, you can study the way spectral features evolve over time, identify common time-varying patterns in two signals, and perform time-localized filtering. Using discrete wavelet analysis, you can analyze signals and images at different resolutions to detect changepoints, discontinuities, and other events not readily visible in raw data. You can compare signal statistics on multiple scales, and perform fractal analysis of data to reveal hidden patterns.

With Wavelet Toolbox you can obtain a sparse representation of data, useful for denoising or compressing the data while preserving important features. Many toolbox functions support C/C++ code generation for desktop prototyping and embedded system deployment.

Key Features

- Time-frequency analysis using continuous wavelet transform, wavelet coherence, constant-Q transform, empirical mode decomposition, and Hilbert-Huang transform
- Wavelet Signal Denoiser app for denoising time-series data
- Decimated wavelet packet and wavelet transforms, including wavelet leaders for fractal analysis
- Nondecimated techniques, including dual-tree, stationary wavelet, maximal overlap discrete wavelet, and wavelet packet transforms
- Signal, image denoising, and compression, including matching pursuit
- Lifting method for constructing custom wavelets

Installing Wavelet Toolbox Software

To install this toolbox on your computer, see the appropriate platform-specific MATLAB® installation guide. To determine if the Wavelet Toolbox software is already installed on your system, check for a subfolder named `wavelet` within the main toolbox folder.

Wavelet Toolbox software can perform signal or image analysis. For indexed images or truecolor images (represented by `m`-by-`n`-by-3 arrays of `uint8`), all wavelet functions use floating-point operations. To avoid `Out of Memory` errors, be sure to allocate enough memory to process various image sizes.

The memory can be real RAM or can be a combination of RAM and virtual memory. See your operating system documentation for how to configure virtual memory.

System Recommendations

While not a requirement, we recommend you obtain Signal Processing Toolbox™ and Image Processing Toolbox™ software to use in conjunction with the Wavelet Toolbox software. These toolboxes provide complementary functionality that give you maximum flexibility in analyzing and processing signals and images.

This manual makes no assumption that your computer is running any other MATLAB toolboxes.

Platform-Specific Details

Some details of the use of the Wavelet Toolbox software may depend on your hardware or operating system.

Windows Fonts

We recommend you set your operating system to use “Small Fonts.” Set this option by clicking the Display icon in your desktop's Control Panel (accessible through the **Settings > Control Panel** submenu). Select the **Configuration** option, and then use the **Font Size** menu to change to **Small Fonts**. You'll have to restart Windows® for this change to take effect.

Fonts for Non-Windows Platforms

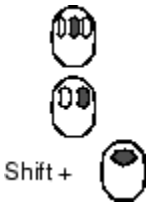
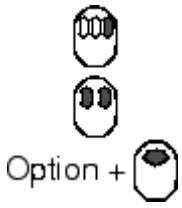
We recommend you set your operating system to use standard default fonts.

However, for all platforms, if you prefer to use large fonts, some of the labels in the Wavelet Analyzer app figures may be illegible when using the default display mode of the toolbox. To change the default mode to accept large fonts, use the `wtbxmngr` function. (For more information, see either the `wtbxmngr` help or its reference page.)

Mouse Compatibility

Wavelet Toolbox software was designed for three distinct types of mouse control.

Left Mouse Button	Middle Mouse Button	Right Mouse Button
Make selections. Activate controls.	Display cross-hairs to show position-dependent information.	Translate plots up and down, and left and right.

Left Mouse Button	Middle Mouse Button	Right Mouse Button
		

Note The functionality of the middle mouse button and the right mouse button can be inverted depending on the platform.

What is a Wavelet?

A wavelet is a waveform of effectively limited duration that has an average value of zero and nonzero norm.

Many signals and images of interest exhibit piecewise smooth behavior punctuated by transients. Speech signals are characterized by short bursts encoding consonants followed by steady-state oscillations indicative of vowels. Natural images have edges. Financial time series exhibit transient behavior, which characterize rapid upturns and downturns in economic conditions. Unlike the Fourier basis, wavelet bases are adept at sparsely representing piecewise regular signals and images, which include transient behavior.

Compare wavelets with sine waves, which are the basis of Fourier analysis. Sinusoids do not have limited duration — they extend from minus to plus infinity. While sinusoids are smooth and predictable, wavelets tend to be irregular and asymmetric.

Fourier analysis consists of breaking up a signal into sine waves of various frequencies. Similarly, wavelet analysis is the breaking up of a signal into shifted and scaled versions of the original (or *mother*) wavelet.

Just looking at pictures of wavelets and sine waves, you can see intuitively that signals with sharp changes might be better analyzed with an irregular wavelet than with a smooth sinusoid.

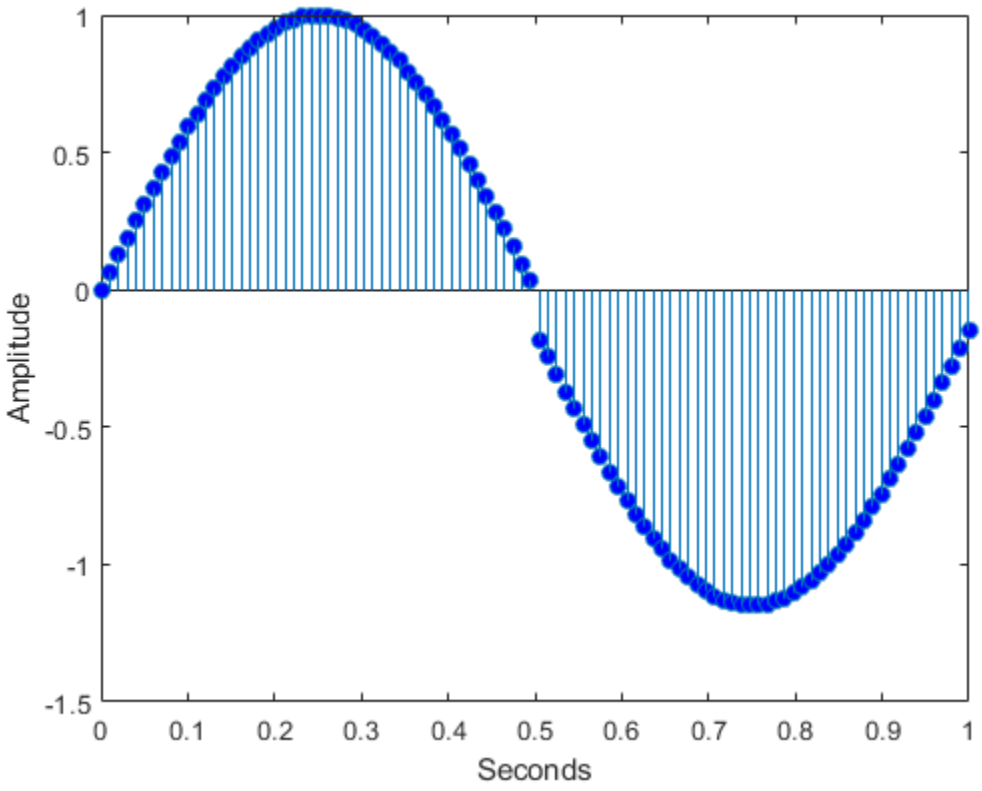
It also makes sense that local features can be described better with wavelets that have local extent. The following example illustrates this for a simple signal consisting of a sine wave with a discontinuity.

Localize Discontinuity in Sine Wave

This example shows wavelet analysis can localize a discontinuity in a sine wave.

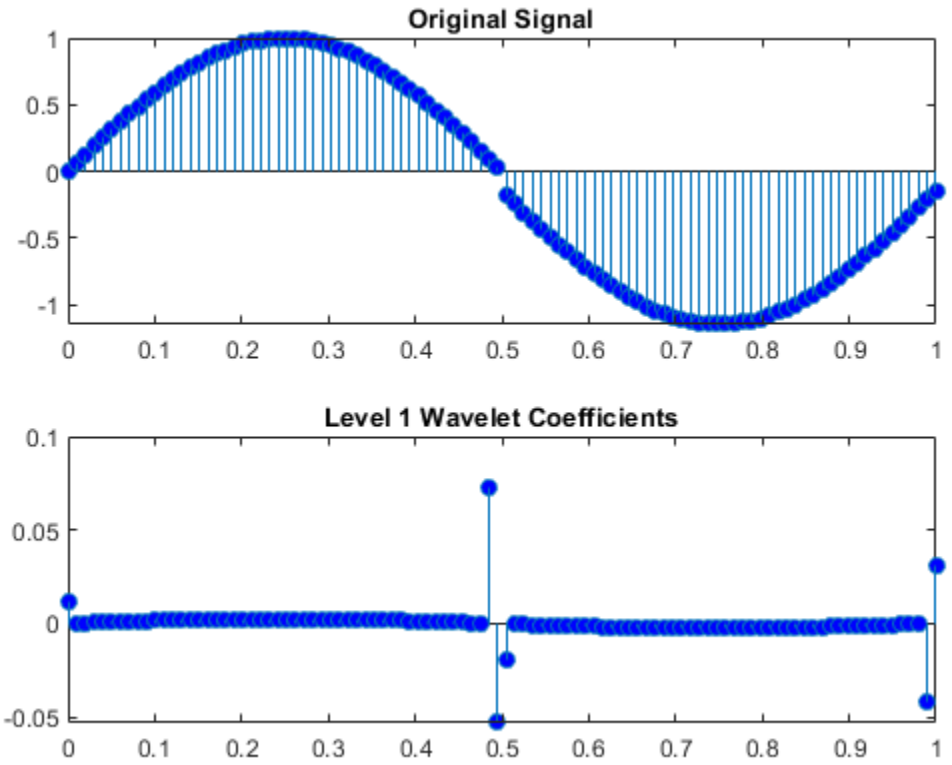
Create a 1-Hz sine wave sampled at 100 Hz. The duration of the sine wave is one second. The sine wave has a discontinuity at $t = 0.5$ seconds.

```
t = linspace(0,1,100)';
x = sin(2*pi*t);
x1 = x-0.15;
y = zeros(size(x));
y(1:length(y)/2) = x(1:length(y)/2);
y(length(y)/2+1:end) = x1(length(y)/2+1:end);
stem(t,y,'markerfacecolor',[0 0 1]);
xlabel('Seconds');
ylabel('Amplitude');
```



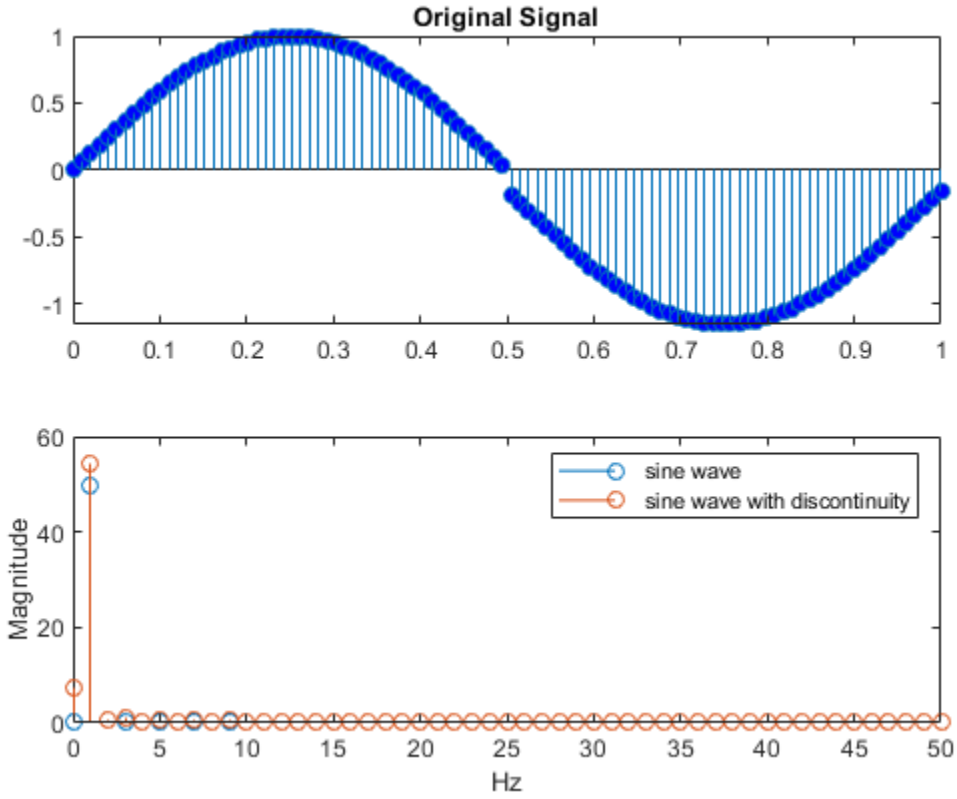
Obtain the nondecimated discrete wavelet transform of the sine wave using the `'sym2'` wavelet and plot the wavelet (detail) coefficients along with the original signal.

```
[swa,swd] = swt(y,1,'sym2');
subplot(211)
stem(t,y,'markerfacecolor',[0 0 1]);
title('Original Signal');
subplot(212)
stem(t,swd,'markerfacecolor',[0 0 1]);
title('Level 1 Wavelet Coefficients');
```



Compare the Fourier coefficient magnitudes for the 1-Hz sine wave with and without the discontinuity.

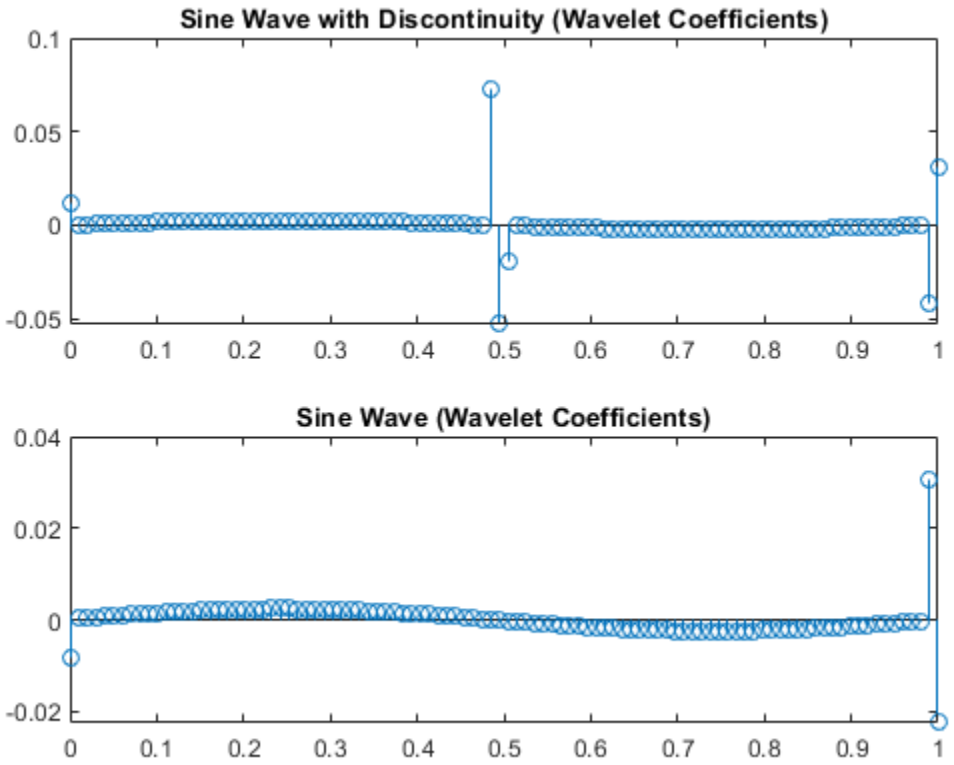
```
dftsig = fft([x y]);
dftsig = dftsig(1:length(y)/2+1,:);
df = 100/length(y);
freq = 0:df:50;
stem(freq,abs(dftsig));
xlabel('Hz'); ylabel('Magnitude');
legend('sine wave','sine wave with discontinuity');
```



There is minimal difference in the magnitudes of the Fourier coefficients. Because the discrete Fourier basis vectors have support over the entire time interval, the discrete Fourier transform does not detect the discontinuity as efficiently as the wavelet transform.

Compare the level 1 wavelet coefficients for the sine wave with and without the discontinuity.

```
[swax,swdx] = swt(x,1,'sym2');
subplot(211)
stem(t,swd); title('Sine Wave with Discontinuity (Wavelet Coefficients)');
subplot(212)
stem(t,swdx); title('Sine Wave (Wavelet Coefficients)');
```



The wavelet coefficients of the two signals demonstrate a significant difference. Wavelet analysis is often capable of revealing characteristics of a signal or image that other analysis techniques miss, like trends, breakdown points, discontinuities in higher derivatives, and self-similarity. Furthermore, because wavelets provide a different view of data than those presented by Fourier techniques, wavelet analysis can often significantly compress or denoise a signal without appreciable degradation.

Choose a Wavelet

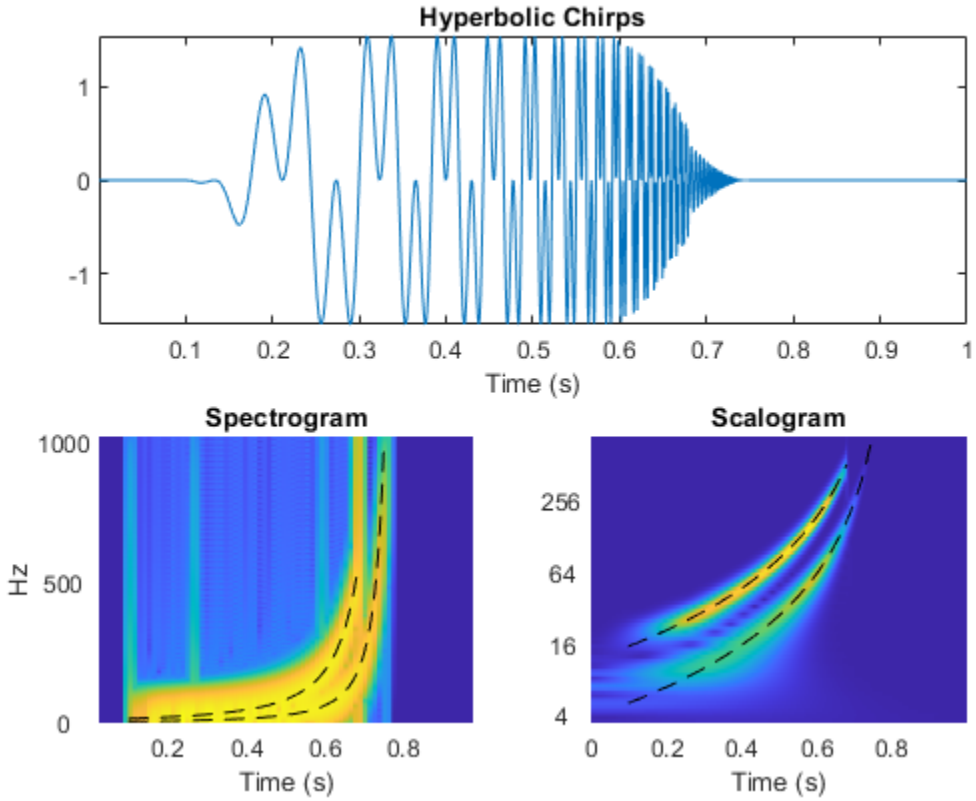
There are two types of wavelet analysis: continuous and multiresolution. The type of wavelet analysis best suited for your work depends on what you want to do with the data. This topic focuses on 1-D data, but you can apply the same principles to 2-D data. To learn how to perform and interpret each type of analysis, see “Practical Introduction to Continuous Wavelet Analysis” and “Practical Introduction to Multiresolution Analysis”.

Time-Frequency Analysis

If your goal is to perform a detailed time-frequency analysis, choose the continuous wavelet transform (CWT). In terms of implementation, scales are discretized more finely in the CWT than in the discrete wavelet transform (DWT). For additional information, see “Continuous and Discrete Wavelet Transforms” on page 1-38.

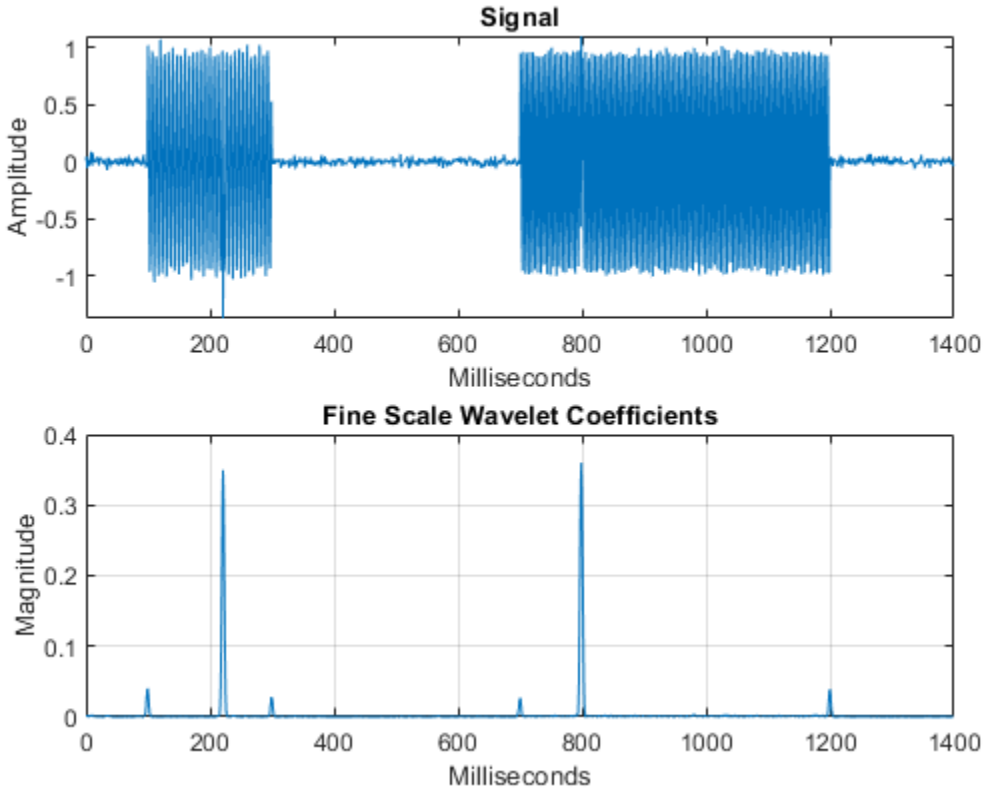
Instantaneous Frequency

The CWT is superior to the short-time Fourier transform (STFT) for signals in which the instantaneous frequency grows rapidly. In the following figure, the instantaneous frequencies of the hyperbolic chirp are plotted as dashed lines in the spectrogram and CWT-derived scalogram. For additional information, see “Time-Frequency Analysis and Continuous Wavelet Transform”.



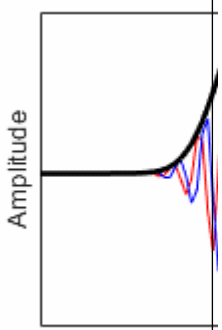
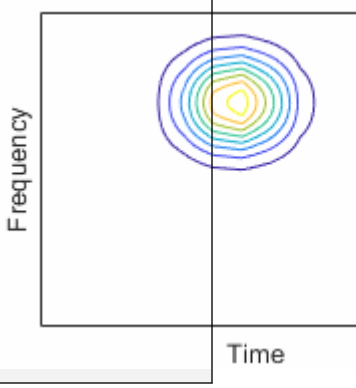
Localizing Transients

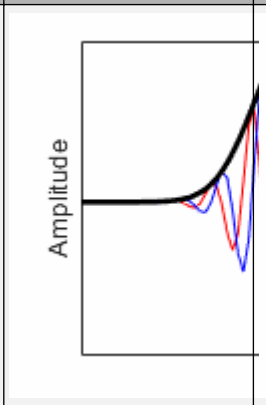
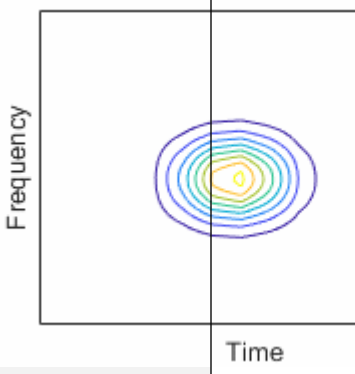
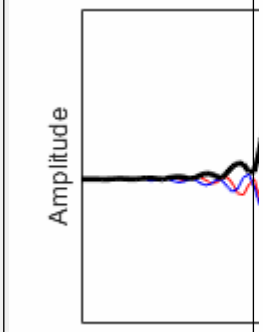
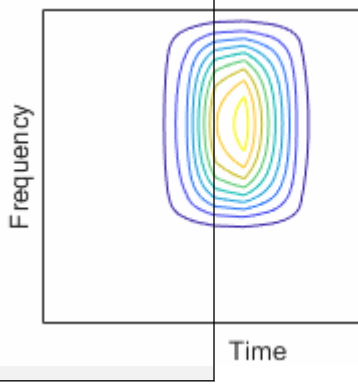
The CWT is good at localizing transients in nonstationary signals. In the following figure, observe how well the wavelet coefficients align with the abrupt changes that occur in the signal. For additional information, see “Practical Introduction to Continuous Wavelet Analysis”.



Supported Wavelets

To obtain the continuous wavelet transform of your data, use `cwt` and `cwtfilterbank`. Both functions support the analytic wavelets listed in the following table. By default, `cwt` and `cwtfilterbank` use the generalized Morse wavelet family. This family is defined by two parameters. You can vary the parameters to recreate many commonly used wavelets. In the time-domain plots, the red line and blue lines are the real and imaginary parts, respectively, of the wavelet. The contour plots show the wavelet spread in time and frequency. For additional information, see “Morse Wavelets” and “Generalized Morse and Analytic Morlet Wavelets”.

Wavelet	Features	Name	Time Domain	Time-Frequency Domain
Generalized Morse Wavelet	Can vary two parameters to change time and frequency spread	`'morse'` (default)		

Wavelet	Features	Name	Time Domain	Time-Frequency Domain
Analytic Morlet (Gabor) Wavelet	Equal variance in time and frequency	`'amor'`		
Bump Wavelet	Wider variance in time, narrower variance in frequency	`'bump'`		

All the wavelets in the table are analytic. Analytic wavelets are wavelets with one-sided spectra and are complex valued in the time domain. These wavelets are a good choice for obtaining a time-frequency analysis using the CWT. Because the wavelet coefficients are complex valued, the CWT provides phase information. `cwt` and `cwtfilterbank` support analytic and anti-analytic wavelets. For additional information, see “CWT-Based Time-Frequency Analysis”.

Multiresolution Analysis

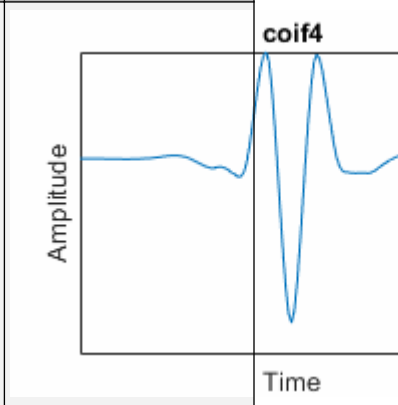
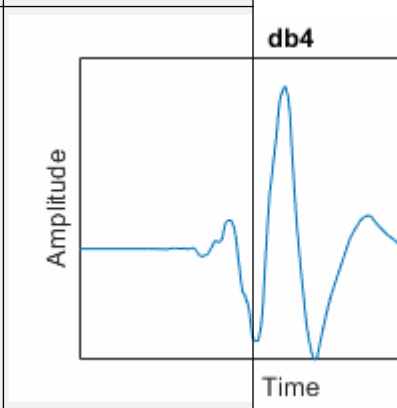
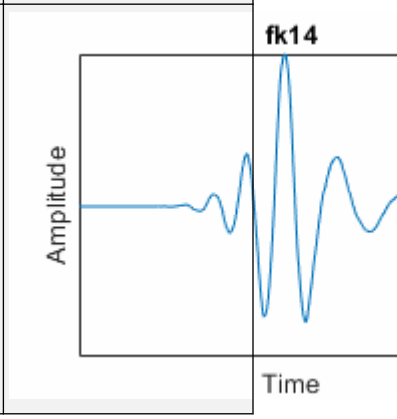
In a multiresolution analysis (MRA), you approximate a signal at progressively coarser scales while recording the differences between approximations at consecutive scales. You create the approximations and the differences by taking the discrete wavelet transform (DWT) of the signal. The DWT provides a sparse representation for many natural signals. Approximations are formed by comparing the signal with scaled and translated copies of a scaling function. Differences between consecutive scales, also known as details, are captured using scaled and translated copies of a wavelet. On a $\log_2$ scale, the difference between consecutive scales is always 1. In the case of the CWT, differences between consecutive scales are finer.

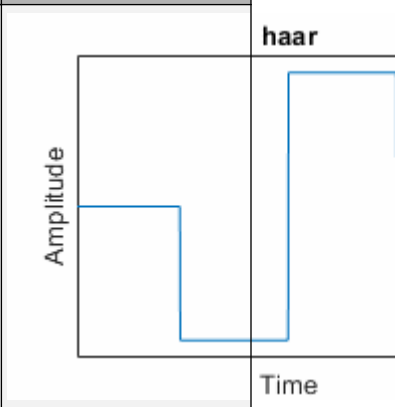
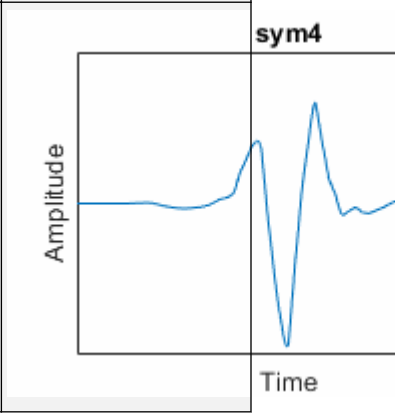
When generating the MRA, you can either subsample (decimate) the approximation by a factor of 2 every time you increase the scale or not. Each option offers advantages and disadvantages. If you subsample, you end up with the same number of wavelet coefficients as the original signal. In the decimated DWT, translations are integer multiples of scale. For the nondecimated DWT, translations are integer shifts. A nondecimated DWT provides a redundant representation of the original data, but

not as redundant as the CWT. Your application not only influences your choice of wavelet, but also which version of the DWT to use.

Energy Preservation

If preserving energy in the analysis stage is important, you must use an orthogonal wavelet. An orthogonal transform preserves energy. Consider using an orthogonal wavelet with compact support. Keep in mind that except for the Haar wavelet, orthogonal wavelets with compact support are not symmetric. The associated filters have nonlinear phase. This table lists supported orthogonal wavelets. See `wavemngr('read')` for all wavelet family names.

Orthogonal Wavelet	Features	Name	See Also	Representative
Coiflet	Scaling function and wavelets have same number of vanishing moments	`'coifN'` for N = 1, 2, ..., 5	`coifwavf`, `waveinfo`	
Daubechies	Nonlinear phase; energy concentrated near the start of their support	`'dbN'` for N = 1, 2, ..., 45	`dbaux`, `waveinfo`, Extremal Phase Wavelet Coefficients on page 1-21	
Fejér-Korovkin	Filters constructed to minimize the difference between a valid scaling filter and the ideal sinc lowpass filter; are especially useful in discrete (decimated and undecimated) wavelet packet transforms.	`'fkN'` for N = 4, 6, 8, 14, 18, 22	`fejerkorovkin`, `waveinfo`	

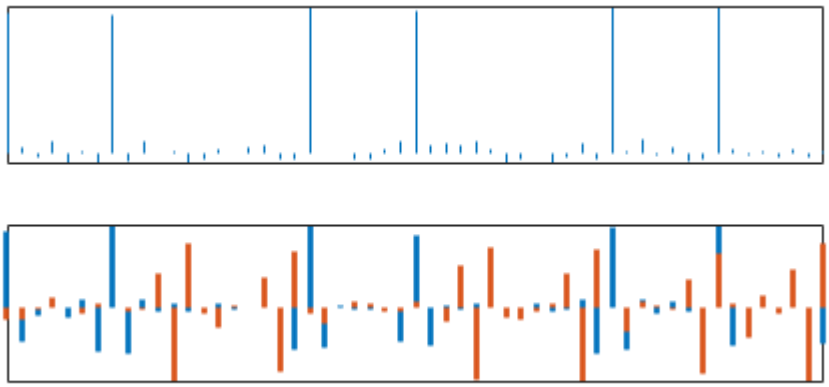
Orthogonal Wavelet	Features	Name	See Also	Representative
Haar	Symmetric; special case of Daubechies; useful for edge detection	`'haar'` (`'db1'`)	`waveinfo`	
Symlet	Least asymmetric; nearly linear phase	`'symN'` for N = 2, 3, ..., 45	`symaux`, `waveinfo`, “Least Asymmetric Wavelet and Phase”	

Use `waveinfo` to learn more about individual wavelet families. For example, `waveinfo('db')`.

Depending on how you address border distortions, the DWT might not conserve energy in the analysis stage. For more information, see “Border Effects”. The maximal overlap discrete wavelet transform `modwt` and maximal overlap discrete wavelet packet transform `modwpt` do conserve energy. The wavelet packet decomposition `dwpt` does not conserve energy.

Feature Detection

If you want to find closely spaced features, choose wavelets with smaller support, such as `haar`, `db2`, or `sym2`. The support of the wavelet should be small enough to separate the features of interest. Wavelets with larger support tend to have difficulty detecting closely spaced features. Using wavelets with large support can result in coefficients that do not distinguish individual features. In the following figure, the upper plot shows a signal with spikes. The lower plot shows the first-level MRA details of a maximal overlap DWT using the `haar` (thick blue lines) and `db6` (thick red lines) wavelets.



If your data has sparsely spaced transients, you can use wavelets with larger support.

Analysis of Variance

If your goal is to conduct an analysis of variance, the maximal overlap discrete wavelet transform (MODWT) is suited for the task. The MODWT is a variation of the standard DWT.

- The MODWT conserves energy in the analysis stage.
- The MODWT partitions variance across scales. For examples, see “Wavelet Analysis of Financial Data” and “Wavelet Changepoint Detection”.
- The MODWT requires an orthogonal wavelet, such as a Daubechies wavelet or Symlet.
- The MODWT is a shift-invariant transform. Shifting the input data shifts the wavelet coefficients by an identical amount. The decimated DWT is not shift invariant. Shifting the input changes the coefficients and can redistribute energy across scales.

See `modwt`, `modwtmra`, and `modwtvar` for more information. See also “Comparing MODWT and MODWTMRA” on page 1-27.

Redundancy

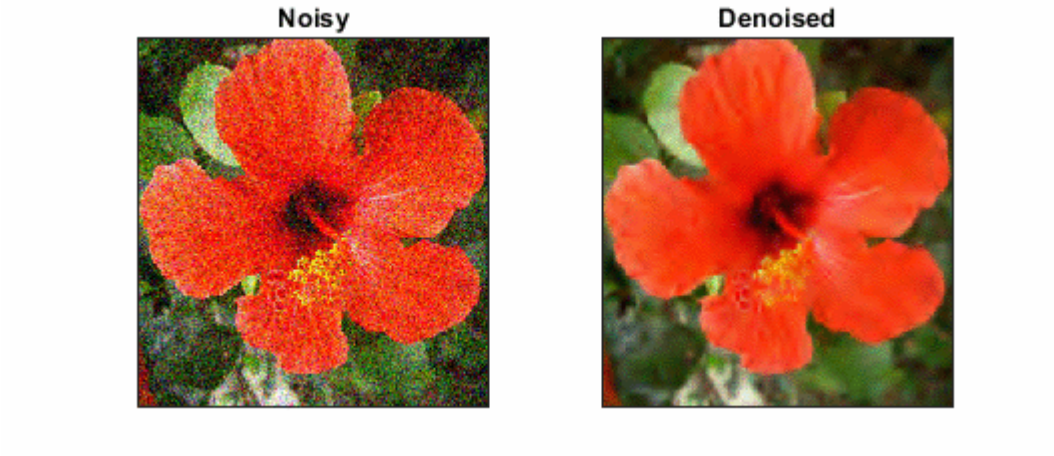
Taking the decimated DWT, `wavedec`, of a signal using an orthonormal family of wavelets provides a minimally redundant representation of the signal. There is no overlap in wavelets within and across scales. The number of coefficients equals the number of signal samples. Minimally redundant representations are a good choice for compression, when you want to remove features that are not perceived.

The CWT of a signal provides a highly redundant representation of a signal. There is significant overlap between wavelets within and across scales. Also, given the fine discretization of the scales, the cost to compute the CWT and store the wavelet coefficients is significantly greater than the DWT. The MODWT `modwt` is also a redundant transform but the redundancy factor is usually significantly less than the CWT. Redundancy tends to reinforce signal characteristics and features you want to examine, such as frequency breaks or other transient events.

If your work requires representing a signal with minimal redundancy, use `wavedec`. If your work requires a redundant representation, use `modwt` or `modwpt`.

Denoising

An orthogonal wavelet, such as a Symlet or Daubechies wavelet, is a good choice for denoising signals. A biorthogonal wavelet can also be good for image processing. Biorthogonal wavelet filters have linear phase which is very critical for image processing. Using a biorthogonal wavelet will not introduce visual distortions in the image.

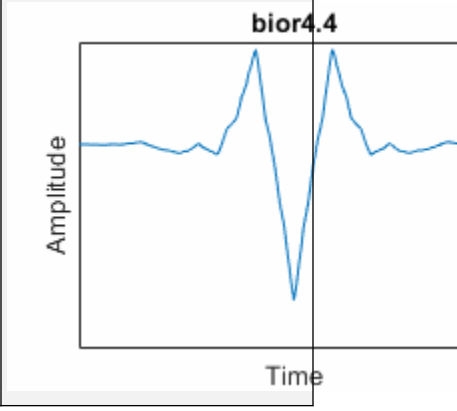


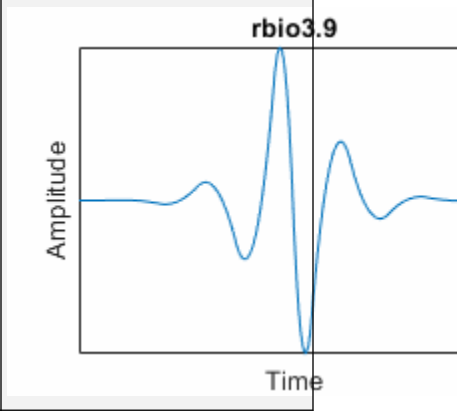
- An orthogonal transform does not color white noise. If white noise is provided as input to an orthogonal transform, the output is white noise. Performing a DWT with a biorthogonal wavelet colors white noise.
- An orthogonal transform preserves energy.

The `sym4` wavelet is the default wavelet used in `wdenoise` and **Wavelet Signal Denoiser** app. The `bior4.4` biorthogonal wavelet is the default wavelet in `wdenoise2`.

Compression

If your work involves signal or image compression, consider using a biorthogonal wavelet. This table lists the supported biorthogonal wavelets with compact support.

Biorthogonal Wavelet	Features	Name	Representative
Biorthogonal Spline	Compact support; symmetric filters; linear phase	`'bior`*`Nr`*`.`*`Nd`*`'` where *Nr* and *Nd* are the numbers of vanishing moments for the reconstruction and decomposition filters, respectively; see `waveinfo('bior')` for supported values	

Biorthogonal Wavelet	Features	Name	Representative
Reverse Biorthogonal Spline	Compact support; symmetric filters; linear phase	`'rbio`*`Nd.Nr`*`'` where *Nr* and *Nd* are the numbers of vanishing moments for the reconstruction and decomposition filters, respectively; see `waveinfo('rbio')` for supported values	

Having two scaling function-wavelet pairs, one pair for analysis and another for synthesis, is useful for compression.

- Biorthogonal wavelet filters are symmetric and have linear phase. (See .)
- The wavelets used for analysis can have many vanishing moments. A wavelet with *N* vanishing moments is orthogonal to polynomials of degree *N*-1. Using a wavelet with many vanishing moments results in fewer significant wavelet coefficients. Compression is improved.
- The dual wavelets used for synthesis can have better regularity. The reconstructed signal is smoother.

Using an analysis filter with fewer vanishing moments than a synthesis filter can adversely affect compression. For an example, see “Image Reconstruction with Biorthogonal Wavelets” on page 1-32.

When using biorthogonal wavelets, energy is not conserved at the analysis stage. See “Orthogonal and Biorthogonal Filter Banks” for additional information.

General Considerations

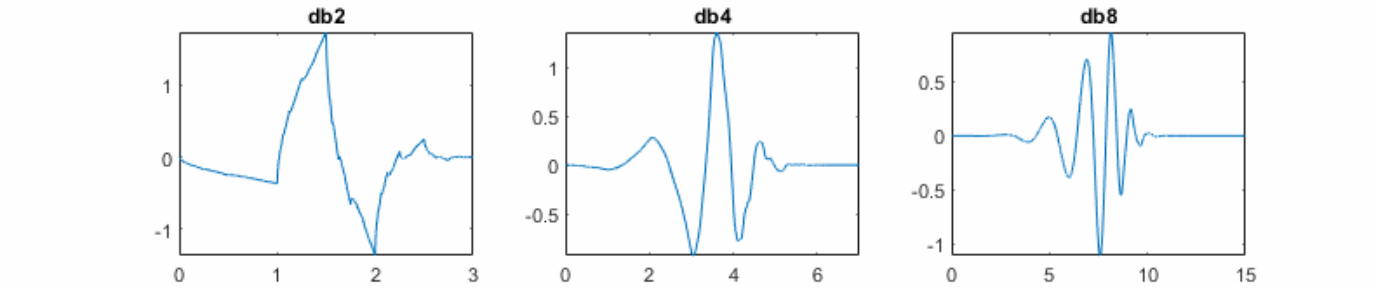
Wavelets have properties that govern their behavior. Depending on what you want to do, some properties can be more important.

Orthogonality

If a wavelet is orthogonal, the wavelet transform preserves energy. Except for the Haar wavelet, no orthogonal wavelet with compact support is symmetric. The associated filter has nonlinear phase.

Vanishing Moments

A wavelet with *N* vanishing moments is orthogonal to polynomials of degree $N-1$. For an example, see “Wavelets and Vanishing Moments” on page 1-35. The number of vanishing moments and the oscillation of the wavelet have a loose relationship. As the number of vanishing moments grows, the greater the wavelet oscillates.



The number of vanishing moments also affects the support of a wavelet. Daubechies proved that a wavelet with *N* vanishing moments must have a support of at least length 2*N*-1.

Names for many wavelets are derived from the number of vanishing moments. For example, `db6` is the Daubechies wavelet with six vanishing moments, and `sym3` is the Symlet with three vanishing moments. For coiflet wavelets, `coif3` is the coiflet with six vanishing moments. For Fejér-Korovkin wavelets, `fk8` is the Fejér-Korovkin wavelet with a length 8 filter. Biorthogonal wavelet names are derived from the number of vanishing moments the analysis wavelet and synthesis wavelet each have. For instance, `bior3.5` is the biorthogonal wavelet with three vanishing moments in the synthesis wavelet and five vanishing moments in the analysis wavelet. To learn more, see `waveinfo` and `wavemngr`.

If the number of vanishing moments *N* is equal to 1, 2, or 3, then `db`*N* and `sym`*N* are identical.

Regularity

Regularity is related to how many continuous derivatives a function has. Intuitively, regularity can be considered a measure of smoothness. To detect an abrupt change in the data, a wavelet must be sufficiently regular. For a wavelet to have *N* continuous derivatives, the wavelet must have at least *N* +1 vanishing moments. See “Detecting Discontinuities and Breakdown Points” for an example. If your data is relatively smooth with few transients, a more regular wavelet might be a better fit for your work.

References

[1] Daubechies, Ingrid. *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, 1992.

See Also

Apps

Signal Multiresolution Analyzer | **Wavelet Signal Denoiser**

Functions

`cwt` | `cwtfilterbank` | `dwtfilterbank` | `wavedec` | `wavedec2` | `waveinfo` | `wavemngr` | `wdenoise` | `wdenoise2`

Related Examples

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

More About

- Understanding Wavelets, Part 1: What Are Wavelets
- Understanding Wavelets, Part 2: Types of Wavelet Transforms
- Understanding Wavelets, Part 3: An Example Application of the Discrete Wavelet Transform
- Understanding Wavelets, Part 4: An Example Application of the Continuous Wavelet Transform

Extremal Phase

This example demonstrates that for a given support, the cumulative sum of the squared coefficients of a scaling filter increase more rapidly for an extremal phase wavelet than other wavelets.

Generate the scaling filter coefficients for the `db15` and `sym15` wavelets. Both wavelets have support of width $2 \times 15 - 1 = 29$.

```
[~,~,LoR_db,~] = wfilters('db15');
[~,~,LoR_sym,~] = wfilters('sym15');
```

Next, generate the scaling filter coefficients for the `coif5` wavelet. This wavelet also has support of width $6 \times 5 - 1 = 29$.

```
[~,~,LoR_coif,~] = wfilters('coif5');
```

Confirm the sum of the coefficients for all three wavelets equals $\sqrt{2}$.

```
sqrt(2)-sum(LoR_db)
```

```
ans = 2.2204e-16
```

```
sqrt(2)-sum(LoR_sym)
```

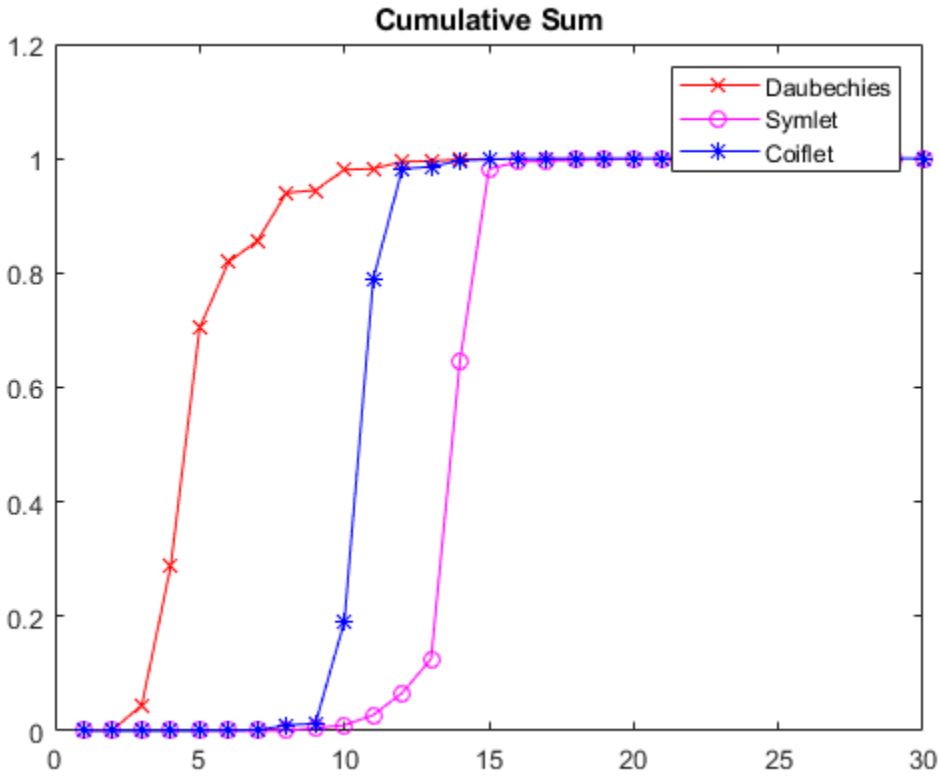
```
ans = 0
```

```
sqrt(2)-sum(LoR_coif)
```

```
ans = 2.2204e-16
```

Plot the cumulative sums of the squared coefficients. Note how rapidly the Daubechies sum increases. This is because its energy is concentrated at small abscissas. Since the Daubechies wavelet has extremal phase, the cumulative sum of its squared coefficients increases more rapidly than the other two wavelets.

```
plot(cumsum(LoR_db.^2),'rx-')
hold on
plot(cumsum(LoR_sym.^2),'mo-')
plot(cumsum(LoR_coif.^2),'b*-')
legend('Daubechies','Symlet','Coiflet')
title('Cumulative Sum')
```



See Also

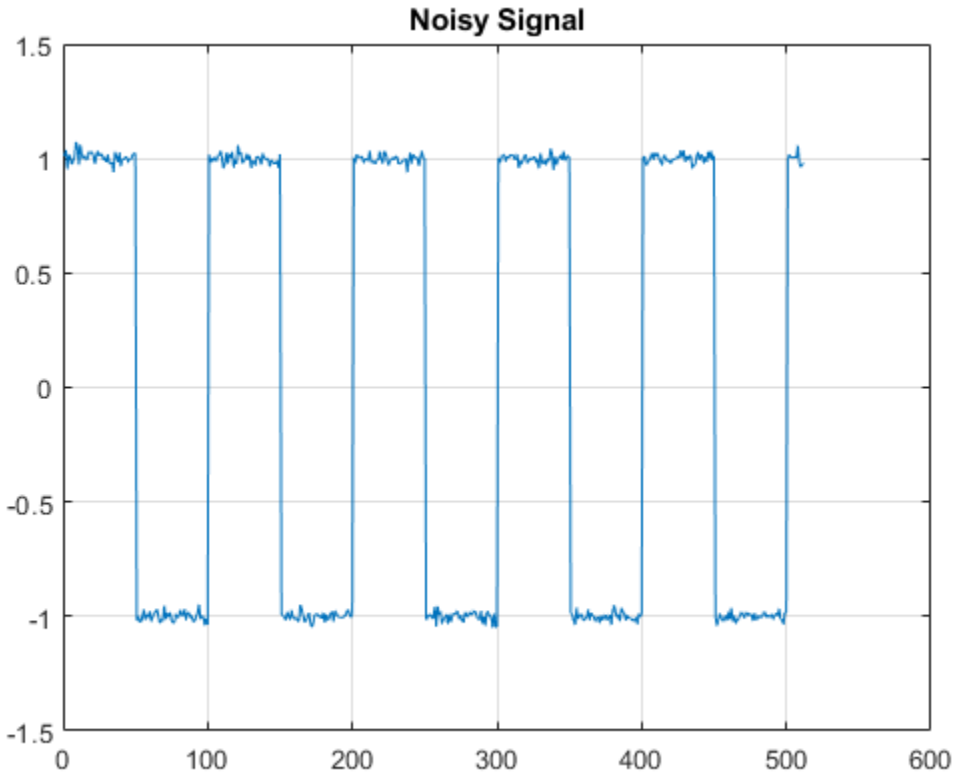
dbaux | symaux

Effect of Wavelet Support on Noisy Data

In this example you demonstrate an instance of discontinuities in noisy data being represented more sparsely using a Haar wavelet than when using a wavelet with larger support.

Create a noisy square wave with 512 samples. Plot the square wave.

```
n = 512;
t = 0:0.001:(n*0.001)-0.001;
yn = square(2*pi*10*t)+0.02*randn(size(t));
plot(yn)
grid on
title('Noisy Signal')
```

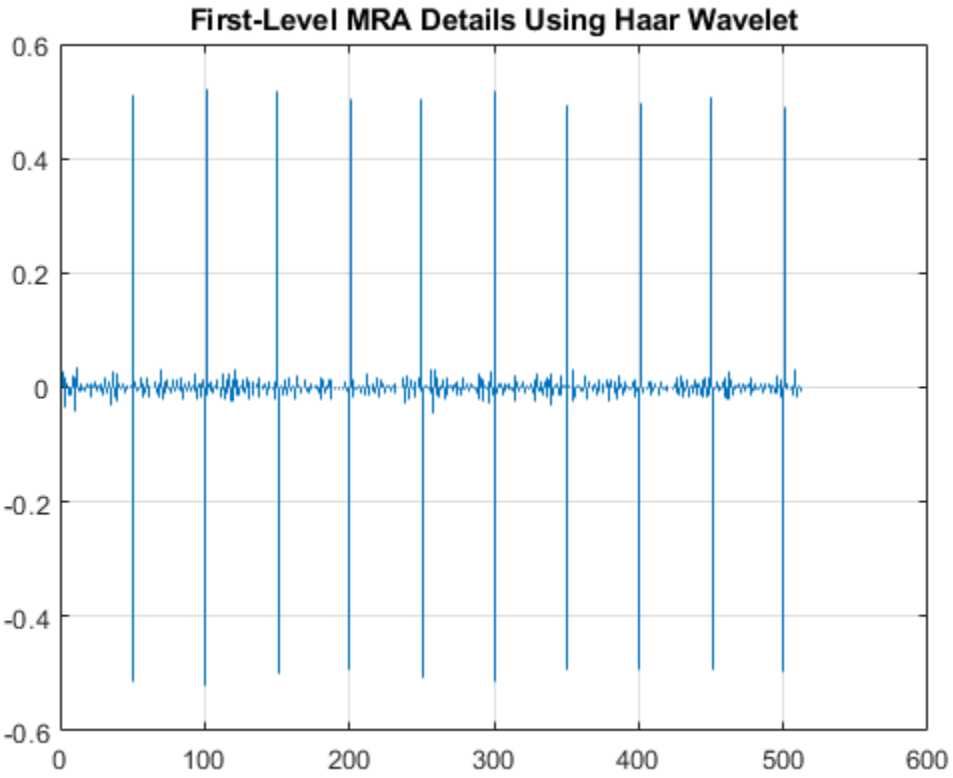


Obtain the maximal overlap discrete wavelet transform (MODWT) of the signal using the `haar` wavelet. The `haar` wavelet has a support of length equal to 1

```
modhaar = modwt(yn,'haar');
```

Obtain the multiresolution analysis from the `haar` MODWT matrix and plot the first-level details.

```
mrahaar = modwtmra(modhaar,'haar');
stem(mrahaar(1,:),'Marker','none','ShowBaseLine','off');
grid on
title('First-Level MRA Details Using Haar Wavelet')
```

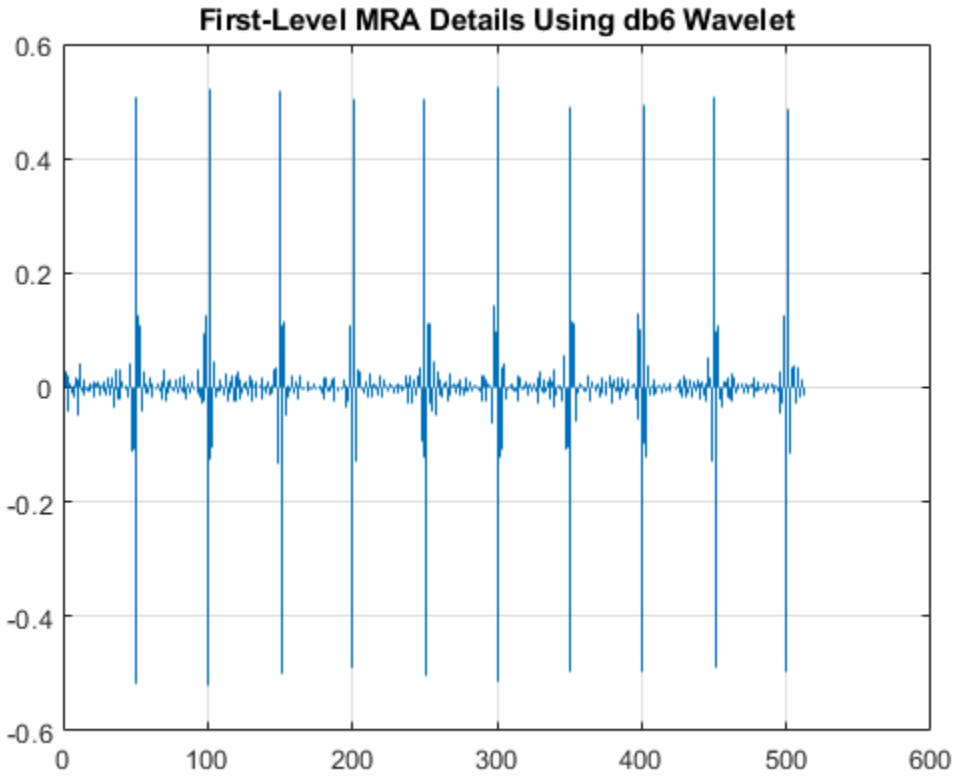


Obtain the MODWT of the signal using the `db6` wavelet. The `db6` wavelet has a support of length equal to 11.

```
moddb6 = modwt(yn,'db6');
```

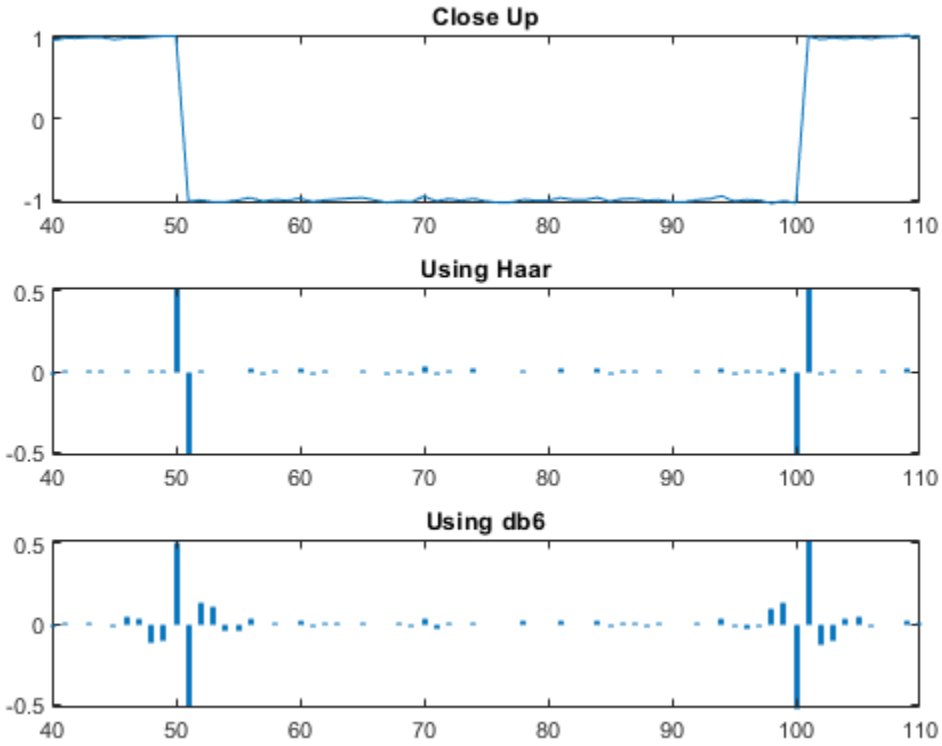
Obtain the multiresolution analysis from the `db6` MODWT matrix and plot the first-level details. The discontinuities are represented less sparsely using the `db6` wavelet than the `haar` wavelet.

```
mradb6 = modwtmra(moddb6,'db6');
stem(mradb6(1,:),'Marker','none','ShowBaseLine','off');
grid on
title('First-Level MRA Details Using db6 Wavelet')
```



Zoom in. Using the Haar wavelet results in fewer coefficients necessary to identify the change in the signal.

```
ind = 40:110;
subplot(311)
plot(ind,yn(ind))
title('Close Up')
subplot(312)
stem(ind,mrahaar(1,ind),'Marker','none','ShowBaseLine','off','LineWidth',2)
title('Using Haar')
subplot(313)
stem(ind,mradb6(1,ind),'Marker','none','ShowBaseLine','off','LineWidth',2)
title('Using db6')
```



See Also

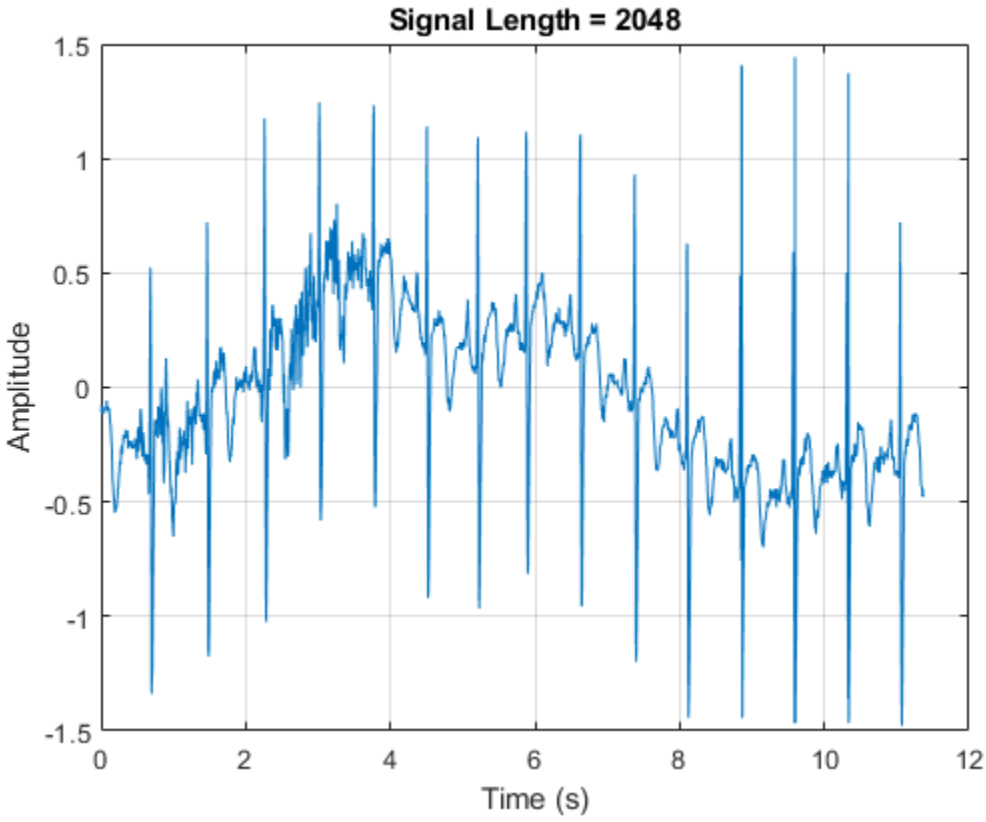
`modwt` | `modwtmra` | `waveinfo`

Comparing MODWT and MODWTMRA

This example demonstrates the differences between the functions MODWT and MODWTMRA. The MODWT partitions a signal's energy across detail coefficients and scaling coefficients. The MODWTMRA projects a signal onto wavelet subspaces and a scaling subspace.

Choose the `sym6` wavelet. Load and plot an electrocardiogram (ECG) signal. The sampling frequency for the ECG signal is 180 hertz. The data are taken from Percival and Walden (2000), p.125 (data originally provided by William Constantine and Per Reinhall, University of Washington).

```
load wecg
t = (0:numel(wecg)-1)/180;
wv = 'sym6';
plot(t,wecg)
grid on
title(['Signal Length = ',num2str(numel(wecg))])
xlabel('Time (s)')
ylabel('Amplitude')
```



Take the MODWT of the signal.

```
wtecg = modwt(wecg,wv);
```

The input data are samples of a function $f(x)$ evaluated at N-many time points. The function can be expressed as a linear combination of the scaling function $\phi(x)$ and wavelet $\psi(x)$at varying scales and

translations: $f(x) = \sum_{k=0}^{N-1} c_k 2^{-J_0/2} \phi(2^{-J_0}x - k) + \sum_{j=1}^{J_0} f_j(x)$ where $f_j(x) = \sum_{k=0}^{N-1} d_{j,k} 2^{-j/2} \psi(2^{-j}x - k)$ and J_0 is the number of levels of wavelet decomposition. The first sum is the coarse scale approximation of the signal, and the $f_j(x)$ are the details at successive scales. MODWT returns the N-many coefficients $\{c_k\}$and the $(J_0 \times N)$-many detail coefficients $\{d_{j,k}\}$ of the expansion. Each row in `wtecg` contains the coefficients at a different scale.

When taking the MODWT of a signal of length N, there are floor($\log_2(N)$)-many levels of decomposition (by default). Detail coefficients are produced at each level. Scaling coefficients are returned only for the final level. In this example, since N = 2048, J_0 = floor(log2(2048)) = 11 and the number of rows in `wtecg` is $J_0 + 1 = 11 + 1 = 12$.

The MODWT partitions the energy across the various scales and scaling coefficients: $||X||^2 = \sum_{j=1}^{J_0} ||W_j||^2 + ||V_{J_0}||^2$ where X is the input data, W_j are the detail coefficients at scale j, and V_{J_0} are the final-level scaling coefficients.

Compute the energy at each scale, and evaluate their sum.

```
energy_by_scales = sum(wtecg.^2,2);
Levels = {'D1';'D2';'D3';'D4';'D5';'D6';'D7';'D8';'D9';'D10';'D11';'A11'};
energy_table = table(Levels,energy_by_scales);
disp(energy_table)
```

```
    Levels     energy_by_scales
    _______    ________________

    {'D1' }          14.063
    {'D2' }          20.612
    {'D3' }          37.716
    {'D4' }          25.123
    {'D5' }          17.437
    {'D6' }          8.9852
    {'D7' }          1.2906
    {'D8' }          4.7278
    {'D9' }          12.205
    {'D10'}          76.428
    {'D11'}          76.268
    {'A11'}          3.4192
```

```
energy_total = varfun(@sum,energy_table(:,2))
```

```
energy_total=table
    sum_energy_by_scales
    ____________________

           298.28
```

Confirm the MODWT is energy-preserving by computing the energy of the signal and comparing it with the sum of the energies over all scales.

```
energy_ecg = sum(wecg.^2);
max(abs(energy_total.sum_energy_by_scales-energy_ecg))
```

```
ans = 7.4357e-10
```

Take the MODWTMRA of the signal.

```
mraecg = modwtmra(wtecg,wv);
```

MODWTMRA returns the projections of the function $f(x)$ onto the various wavelet subspaces and final scaling space. That is, MODWTMRA returns $\sum_{k=0}^{N-1} c_k 2^{-J_0/2}\phi(2^{-J_0}x-k)$and the J_0-many $\{f_j(x)\}$ evaluated at N-many time points. Each row in `mraecg` is a projection of $f(x)$ onto a different subspace. This means the original signal can be recovered by adding all the projections. This is *not* true in the case of the MODWT. Adding the coefficients in `wtecg` will *not* recover the original signal.

Choose a time point, add the projections of $f(x)$ evaluated at that time point and compare with the original signal.

```
time_point = 1000;
abs(sum(mraecg(:,time_point))-wecg(time_point))
```

```
ans = 3.0849e-13
```

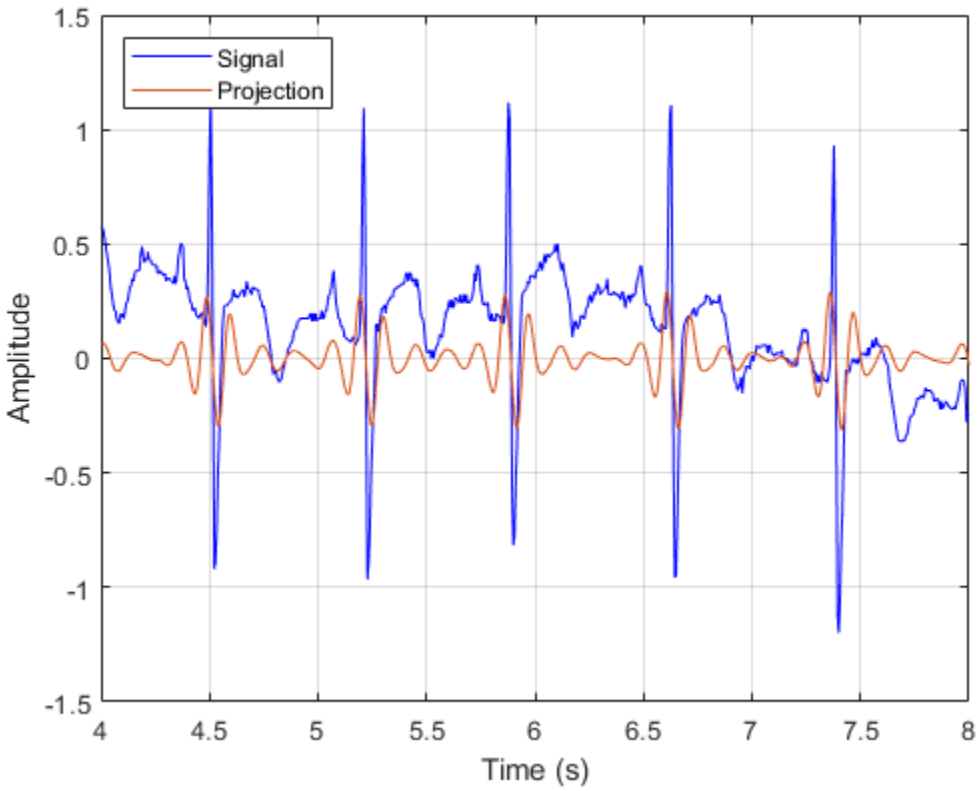
Confirm that, unlike MODWT, MODWTMRA is not an energy-preserving transform.

```
energy_ecg = sum(wecg.^2);
energy_mra_scales = sum(mraecg.^2,2);
energy_mra = sum(energy_mra_scales);
max(abs(energy_mra-energy_ecg))
```

```
ans = 115.7053
```

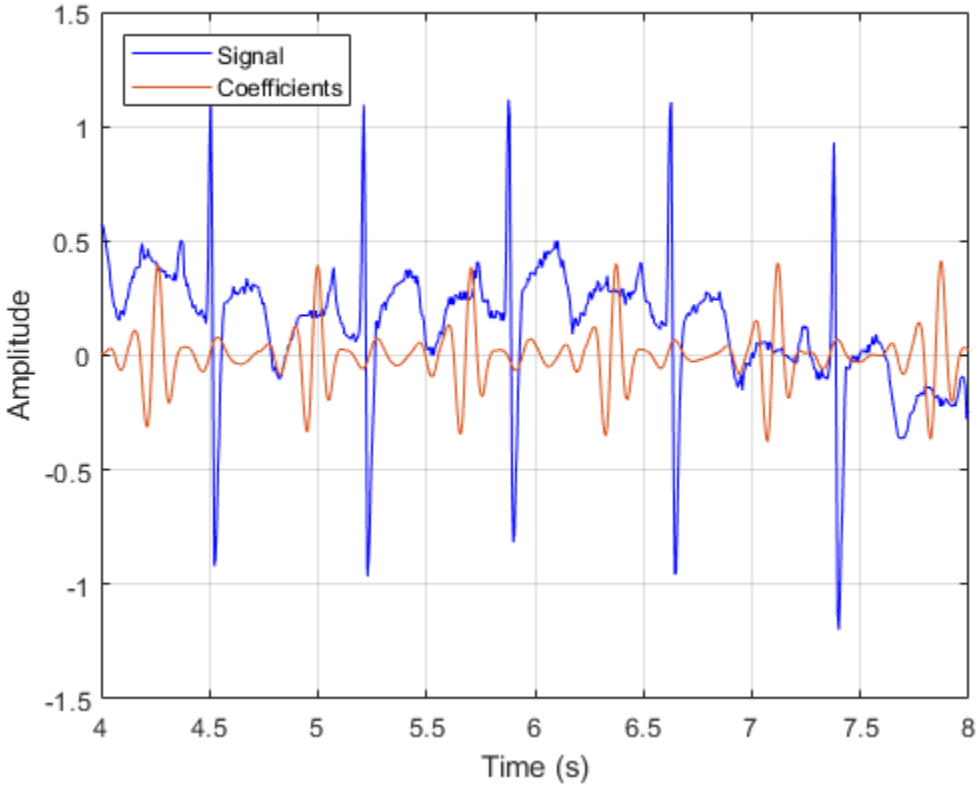
The MODWTMRA is a zero-phase filtering of the signal. Features will be time-aligned. Demonstrate this by plotting the original signal and one of its projections. To better illustrate the alignment, zoom in.

```
plot(t,wecg,'b')
hold on
plot(t,mraecg(4,:),'-')
hold off
grid on
xlim([4 8])
legend('Signal','Projection','Location','northwest')
xlabel('Time (s)')
ylabel('Amplitude')
```



Make a similar plot using the MODWT coefficients at the same scale. Note that features will not be time-aligned. The MODWT is *not* a zero-phase filtering of the input.

```
plot(t,wecg,'b')
hold on
plot(t,wtecg(4,:),'-')
hold off
grid on
xlim([4 8])
legend('Signal','Coefficients','Location','northwest')
xlabel('Time (s)')
ylabel('Amplitude')
```



References

[1] Percival, D. B., and A. T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge, UK: Cambridge University Press, 2000.

See Also

`modwt` | `modwtmra`

Image Reconstruction with Biorthogonal Wavelets

This example shows how applying the order biorthogonal wavelet filters can affect image reconstruction.

Generate the analysis and synthesis filters for the `bior3.5` wavelet. Load in and display an image.

```
[LoD,HiD,LoR,HiR] = wfilters('bior3.5');
load woman
imagesc(X)
colormap gray
```

The analysis filters, `LoD` and `HiD`, have 5 vanishing moments. The synthesis filters, `LoR` and `HiR`, have 3 vanishing moments. Do a five-level wavelet decomposition of the image using the analysis filters.

```
[c1,s1] = wavedec2(X,5,LoD,HiD);
```

Find the threshold that keeps only those wavelet coefficients with magnitudes in the top 10 percent. Use the threshold to set the bottom 90 percent of coefficients to 0.

```
frac = 0.1;
c1sort = sort(abs(c1),'desc');
num = numel(c1);
thr = c1sort(floor(num*frac));
c1new = c1.*(abs(c1)>=thr);
```

Reconstruct the image using the synthesis filters and the thresholded coefficients. Display the reconstruction.

```
X1 = waverec2(c1new,s1,LoR,HiR);
figure
imagesc(X1)
colormap gray
```

Do a five-level wavelet decomposition of the image using the synthesis filters.

```
[c2,s2] = wavedec2(X,5,LoR,HiR);
```

Find the threshold that keeps only those wavelet coefficients with magnitudes in the top 10 percent. Use the threshold to set the bottom 90 percent of coefficients to 0

```
frac = 0.1;
c2sort = sort(abs(c2),'desc');
num = numel(c2sort);
thr = c2sort(floor(num*frac));
c2new = c2.*(abs(c2)>=thr);
```

Reconstruct the image using the synthesis filters and the thresholded coefficients. Display the reconstruction. Decomposing with a filter that has 3 vanishing moments and reconstructing with a filter that has 5 vanishing moments results in poor reconstruction.

```
X2 = waverec2(c2new,s2,LoD,HiD);
figure
```

```
imagesc(X2)
colormap gray
```

See Also

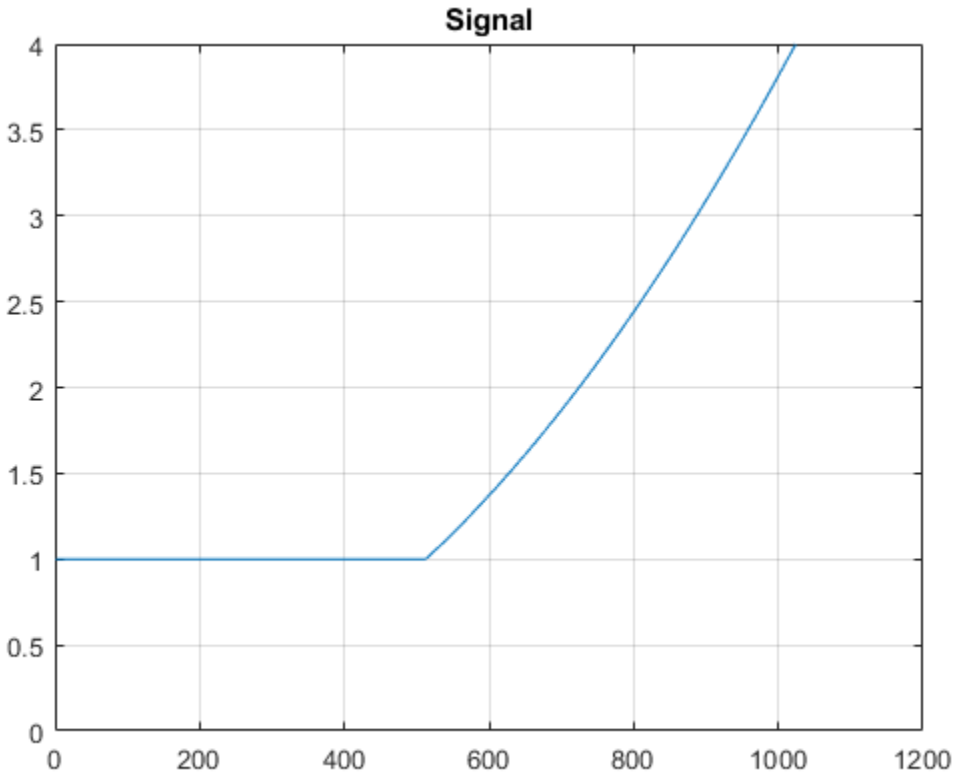
`biorfilt` | `biorwavf` | `wavedec2` | `waverec2` | `wfilters`

Wavelets and Vanishing Moments

This example shows how the number of vanishing moments can affect wavelet coefficients.

Create a signal defined over the interval $0 \le x \le 2$. The signal is constant over the interval $0 \le x < 1$ and quadratic over the interval $1 \le x \le 2$. Plot the signal.

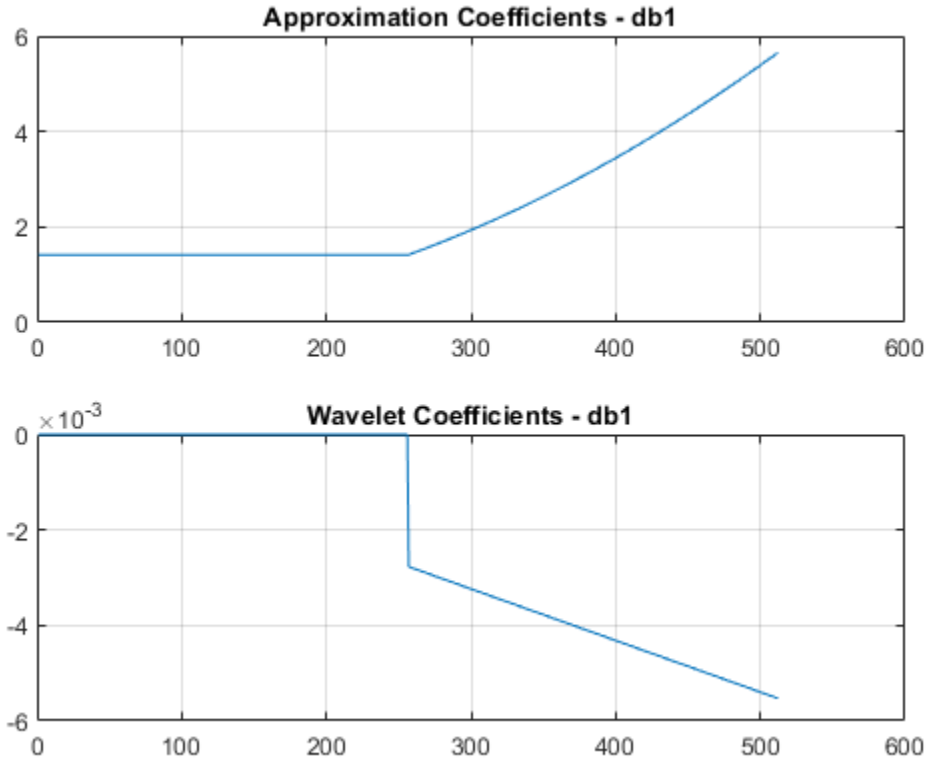
```
n = 1024;
x = linspace(0,2,n);
sig = zeros(1,n);
ind0 = (0<=x)&(x<1);
ind1 = (1<=x)&(x<=2);
sig(ind0) = 1;
sig(ind1) = x(ind1).^2;
plot(sig)
ylim([0 4])
grid on
title('Signal')
```



Compute a single-level wavelet decomposition of the signal using the `db1` wavelet. This wavelet has one vanishing moment. Plot the approximation coefficients and wavelet coefficients.

```
[a1,d1] = dwt(sig,'db1');
figure
subplot(2,1,1)
plot(a1)
ylim([0 6])
```

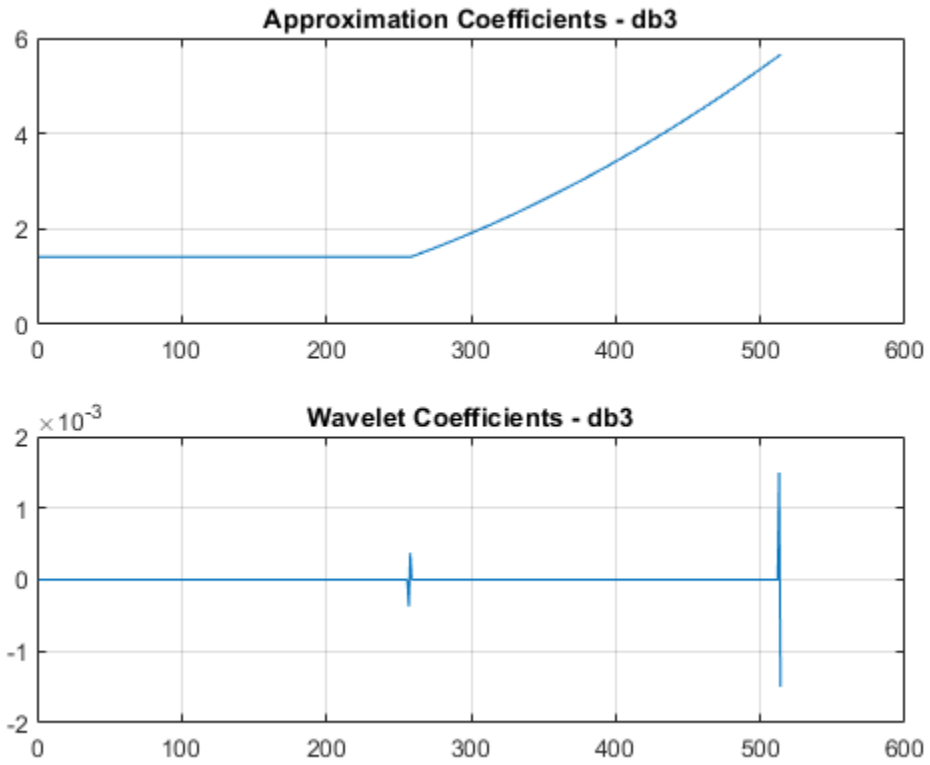
```
grid on
title('Approximation Coefficients - db1')
subplot(2,1,2)
plot(d1)
ylim([-6e-3 0])
grid on
title('Wavelet Coefficients - db1')
```



The wavelet coefficients corresponding with the constant portion of the signal are approximately 0. The magnitude of the wavelet coefficients corresponding with the quadratic portion of the signal are increasing. Because the `db1` wavelet has one vanishing moment, the wavelet is not orthogonal to the quadratic portion of the signal.

Compute a single-level wavelet decomposition of the signal using the `db3` wavelet. This wavelet has three vanishing moments. Plot the approximation coefficients and wavelet coefficients.

```
[a2,d2] = dwt(sig,'db3');
figure
subplot(2,1,1)
plot(a2)
ylim([0 6])
grid on
title('Approximation Coefficients - db3')
subplot(2,1,2)
plot(d2)
grid on
title('Wavelet Coefficients - db3')
```



The wavelet coefficients corresponding with the constant portion of the signal are approximately 0. The spike in the middle corresponds to where the constant and quadratic pieces of the signal meet. The spike at the end is a boundary effect. The magnitude of the wavelet coefficients corresponding with the quadratic portion of the signal are approximately 0. Because the `db3` wavelet has three vanishing moments, the wavelet is orthogonal to the quadratic part of the signal.

See Also

`waveinfo`

Continuous and Discrete Wavelet Transforms

This topic describes the major differences between the continuous wavelet transform (CWT) and the discrete wavelet transform (DWT) – both decimated and nondecimated versions. `cwt` is a discretized version of the CWT so that it can be implemented in a computational environment. This discussion focuses on the 1-D case, but is applicable to higher dimensions.

The `cwt` wavelet transform compares a signal with shifted and scaled (stretched or shrunk) copies of a basic wavelet. If $\psi(t)$ is a wavelet centered at t=0 with time support on [-T/2, T/2], then $\frac{1}{s}\psi(\frac{t-u}{s})$ is centered at t = u with time support [-sT/2+u, sT/2+u]. The `cwt` function uses L1 normalization so that all frequency amplitudes are normalized to the same value. If 0<s<1, the wavelet is contracted (shrunk) and if s>1, the wavelet is stretched. The mathematical term for this is dilation. See “Continuous Wavelet Transform and Scale-Based Analysis” on page 1-63 for examples of how this operation extracts features in the signal by matching it against dilated and translated wavelets.

The major difference between the CWT and discrete wavelet transforms, such as the `dwt` and `modwt`, is how the scale parameter is discretized. The CWT discretizes scale more finely than the discrete wavelet transform. In the CWT, you typically fix some base which is a fractional power of two, for example, $2^{1/v}$ where v is an integer greater than 1. The v parameter is often referred to as the number of “voices per octave”. Different scales are obtained by raising this base scale to positive integer powers, for example $2^{j/v} \quad j = 1, 2, 3, \ldots$. The translation parameter in the CWT is discretized to integer values, denoted here by m. The resulting discretized wavelets for the CWT are

$$\frac{1}{2^{j/v}}\psi(\frac{n-m}{2^{j/v}}).$$

The reason v is referred to as the number of voices per octave is because increasing the scale by an octave (a doubling) requires v intermediate scales. Take for example $2^{v/v} = 2$ and then increase the numerator in the exponent until you reach 4, the next octave. You move from $2^{v/v} = 2$ to $2^{2v/v} = 4$. There are v intermediate steps. Common values for v are 10,12,14,16, and 32. The larger the value of v, the finer the discretization of the scale parameter, s. However, this also increases the amount of computation required because the CWT must be computed for every scale. The difference between scales on a $\log_2$ scale is $1/v$. See “CWT-Based Time-Frequency Analysis” and “Continuous Wavelet Analysis of Modulated Signals” for examples of scale vectors with the CWT.

In the discrete wavelet transform, the scale parameter is always discretized to integer powers of 2, 2^j, j=1,2,3,..., so that the number of voices per octave is always 1. The difference between scales on a $\log_2$ scale is always 1 for discrete wavelet transforms. Note that this is a much coarser sampling of the scale parameter, s, than is the case with the CWT. Further, in the decimated (downsampled) discrete wavelet transform (DWT), the translation parameter is always proportional to the scale. This means that at scale, 2^j, you always translate by 2^jm where m is a nonnegative integer. In nondecimated discrete wavelet transforms like `modwt` and `swt`, the scale parameter is restricted to powers of two, but the translation parameter is an integer like in the CWT. The discretized wavelet for the DWT takes the following form

$$\frac{1}{\sqrt{2^j}}\psi(\frac{1}{2^j}(n-2^j m)).$$

The discretized wavelet for the nondecimated discrete wavelet transform, such as the MODWT, is

$$\frac{1}{\sqrt{2^j}}\psi\left(\frac{n-m}{2^j}\right).$$

To summarize:

- The CWT and the discrete wavelet transforms differ in how they discretize the scale parameter. The CWT typically uses exponential scales with a base smaller than 2, for example $2^{1/12}$. The discrete wavelet transform always uses exponential scales with the base equal to 2. The scales in the discrete wavelet transform are powers of 2. Keep in mind that the physical interpretation of scales for both the CWT and discrete wavelet transforms requires the inclusion of the signal's sampling interval if it is not equal to one. For example, assume you are using the CWT and you set your base to $s_0 = 2^{1/12}$. To attach physical significance to that scale, you must multiply by the sampling interval Δt, so a scale vector covering approximately four octaves with the sampling interval taken into account is $s_0^j \Delta t \quad j = 1, 2, \cdots 48$. Note that the sampling interval multiplies the scales, it is not in the exponent. For discrete wavelet transforms the base scale is always 2.
- The decimated and nondecimated discrete wavelet transforms differ in how they discretize the translation parameter. The decimated discrete wavelet transform (DWT), always translates by an integer multiple of the scale, 2^jm . The nondecimated discrete wavelet transform translates by integer shifts.

These differences in how scale and translation are discretized result in advantages and disadvantages for the two classes of wavelet transforms. These differences also determine use cases where one wavelet transform is likely to provide superior results. Some important consequences of the discretization of the scale and translation parameter are:

- The DWT provides a sparse representation for many natural signals. In other words, the important features of many natural signals are captured by a subset of DWT coefficients that is typically much smaller than the original signal. This "compresses" the signal. With the DWT, you always end up with the same number of coefficients as the original signal, but many of the coefficients may be close to zero in value. As a result, you can often throw away those coefficients and still maintain a high-quality signal approximation. With the CWT, you go from N samples for an N-length signal to a M-by-N matrix of coefficients with M equal to the number of scales. The CWT is a highly redundant transform. There is significant overlap between wavelets at each scale and between scales. The computational resources required to compute the CWT and store the coefficients is much larger than the DWT. The nondecimated discrete wavelet transform is also redundant but the redundancy factor is usually significantly less than the CWT, because the scale parameter is not discretized so finely. For the nondecimated discrete wavelet transform, you go from N samples to an L+1-by-N matrix of coefficients where L is the level of the transform.
- The strict discretization of scale and translation in the DWT ensures that the DWT is an orthonormal transform (when using an orthogonal wavelet). There are many benefits of orthonormal transforms in signal analysis. Many signal models consist of some deterministic signal plus white Gaussian noise. An orthonormal transform takes this kind of signal and outputs the transform applied to the signal plus white noise. In other words, an orthonormal transform takes in white Gaussian noise and outputs white Gaussian noise. The noise is uncorrelated at the input and output. This is important in many statistical signal processing settings. In the case of the DWT, the signal of interest is typically captured by a few large-magnitude DWT coefficients, while the noise results in many small DWT coefficients that you can throw away. If you have studied linear algebra, you have no doubt learned many advantages to using orthonormal bases in the analysis and representation of vectors. The wavelets in the DWT are like orthonormal vectors. Neither the CWT nor the nondecimated discrete wavelet transform are orthonormal transforms.

The wavelets in the CWT and nondecimated discrete wavelet transform are technically called frames, they are linearly-dependent sets.

- The DWT is not shift-invariant. Because the DWT downsamples, a shift in the input signal does not manifest itself as a simple equivalent shift in the DWT coefficients at all levels. A simple shift in a signal can cause a significant realignment of signal energy in the DWT coefficients by scale. The CWT and nondecimated discrete wavelet transform are shift-invariant. There are some modifications of the DWT such as the dual-tree complex discrete wavelet transform that mitigate the lack of shift invariance in the DWT, see “Critically Sampled and Oversampled Wavelet Filter Banks” for some conceptual material on this topic and “Dual-Tree Complex Wavelet Transforms” for an example.
- The discrete wavelet transforms are equivalent to discrete filter banks. Specifically, they are tree-structured discrete filter banks where the signal is first filtered by a lowpass and a highpass filter to yield lowpass and highpass subbands. Subsequently, the lowpass subband is iteratively filtered by the same scheme to yield narrower octave-band lowpass and highpass subbands. In the DWT, the filter outputs are downsampled at each successive stage. In the nondecimated discrete wavelet transform, the outputs are not downsampled. The filters that define the discrete wavelet transforms typically only have a small number of coefficients so the transform can be implemented very efficiently. For both the DWT and nondecimated discrete wavelet transform, you do not actually require an expression for the wavelet. The filters are sufficient. This is not the case with the CWT. The most common implementation of the CWT requires you have the wavelet explicitly defined. Even though the nondecimated discrete wavelet transform does not downsample the signal, the filter bank implementation still allows for good computational performance, but not as good as the DWT.
- The discrete wavelet transforms provide perfect reconstruction of the signal upon inversion. This means that you can take the discrete wavelet transform of a signal and then use the coefficients to synthesize an exact reproduction of the signal to within numerical precision. You can implement an inverse CWT, but it is often the case that the reconstruction is not perfect. Reconstructing a signal from the CWT coefficients is a much less stable numerical operation.
- The finer sampling of scales in the CWT typically results in a higher-fidelity signal analysis. You can localize transients in your signal, or characterize oscillatory behavior better with the CWT than with the discrete wavelet transforms.

For additional information on wavelet transforms and applications, see

- “From Fourier Analysis to Wavelet Analysis” on page 1-58
- “Continuous Wavelet Transform and Scale-Based Analysis” on page 1-63
- “Continuous Wavelet Transform as a Bandpass Filter” on page 1-68
- “Inverse Continuous Wavelet Transform” on page 1-71
- “Interpreting Continuous Wavelet Coefficients” on page 1-73
- “Critically-Sampled Discrete Wavelet Transform” on page 1-85
- “Wavelet Packets: Decomposing the Details”
- “Compare Time-Frequency Content in Signals with Wavelet Coherence”

Guidelines for Continuous Wavelet Transform vs. Discrete Wavelet Transform

Based on the previous section, here are some basic guidelines for deciding on whether to use a discrete or continuous wavelet transform.

- If your application is to obtain the sparsest possible signal representation for compression, denoising, or signal transmission, use the DWT with `wavedec`.
- If your application requires an orthonormal transform, use the DWT with one of the orthogonal wavelet filters. The orthogonal families in the Wavelet Toolbox are designated as type 1 wavelets in the wavelet manager, `wavemngr`. Valid built-in orthogonal wavelet families are `'haar'`, `'dbN'`, `'fkN'`, `'coifN'`, or `'symN'` where N is the number of vanishing moments for all families except `'fk'`. For `'fk'`, N is the number of filter coefficients. See `waveinfo` for more detail.
- If your application requires a shift-invariant transform but you still need perfect reconstruction and some measure of computational efficiency, try a nondecimated discrete wavelet transform like `modwt` or a dual-tree transform like `dualtree`.
- If your primary goal is a detailed time-frequency (scale) analysis or precise localization of signal transients, use `cwt`. For an example of time-frequency analysis with the CWT, see “CWT-Based Time-Frequency Analysis”.
- For denoising a signal by thresholding wavelet coefficients, use the `wdenoise` function or the **Wavelet Signal Denoiser** app. `wdenoise` and **Wavelet Signal Denoiser** provide default settings that can be applied to your data, as well as a simple interface to a variety of denoising methods. With the app, you can visualize and denoise signals, and compare results. For examples of denoising a signal, see “Denoise A Signal Using Default Values” and “Denoise a Signal with the Wavelet Signal Denoiser”. For denoising images, use `wdenoise2`. For an example, see “Denoising Signals and Images”.
- If your application requires that you have a solid understanding of the statistical properties of the wavelet coefficients, use a discrete wavelet transform. There is active work in understanding the statistical properties of the CWT, but currently there are many more distributional results for the discrete wavelet transforms. The success of the DWT in denoising is largely due to our understanding of its statistical properties. For an example of estimation and hypothesis testing using a nondecimated discrete wavelet transform see “Wavelet Analysis of Financial Data”.

See Also

More About

- Understanding Wavelets, Part 1: What Are Wavelets
- Understanding Wavelets, Part 2: Types of Wavelet Transforms
- Understanding Wavelets, Part 3: An Example Application of the Discrete Wavelet Transform
- Understanding Wavelets, Part 4: An Example Application of the Continuous Wavelet Transform

Nonstationary Gabor Frames and the Constant-Q Transform

In this section...
“Decomposing the Time-Frequency Plane” on page 1-42
“Constant-Q Transform” on page 1-43
“References” on page 1-44

Nonstationary Gabor frames enable you to implement time-adaptive or frequency-adaptive analysis of signals. The functions `cqt` and `icqt` use nonstationary Gabor frames to obtain a constant-Q (frequency-adaptive) transform (CQT) of a signal. A notable strength of nonstationary Gabor frames is that they enable the construction of stable inverses, yielding perfect reconstruction.

The theory of nonstationary Gabor frames and efficient algorithms for their implementation are due to Dörfler, Holighaus, Grill, and Velasco [1][2]. The algorithms in [1] and [2] implement a phase-locked version of the CQT that does not preserve the same phases that would be obtained by naïve convolution. In [3], Schörkhuber, Klapuri, Holighaus, and Dörfler develop efficient algorithms for the CQT and inverse CQT that do mimic the coefficients obtained by naïve convolution. The Large Time-Frequency Analysis Toolbox [4] provides an extensive set of algorithms for nonstationary Gabor analysis and synthesis.

In standard Gabor analysis, a window of fixed size tiles the time-frequency plane. A nonstationary Gabor frame is a collection of windowing functions of various sizes that are used to tile the time-frequency plane. Wavelet analysis tiles the time-frequency plane in a similar manner. You have the flexibility to change the sampling density in time or frequency. Nonstationary Gabor frames are useful in areas such as audio signal processing, where fixed-sized time-frequency windows are not optimal. Unlike the short-time Fourier transform, the windows used in the constant-Q transform have adaptable bandwidth and sampling density. In frequency space, the windows are centered at logarithmically spaced center frequencies.

Decomposing the Time-Frequency Plane

The Fourier transform of *f(t)* is the correlation of *f(t)* with $e^{j\omega t}$:

$$F(\omega) = \int_{\infty}^{\infty} f(t) e^{-j\omega t} dt \,.$$

Since $e^{j\omega t}$ does not have compact support, the Fourier transform is not an ideal choice for studying nonstationary signals. If the frequency content of a signal changes over time, the Fourier transform does not capture what those changes are or when those changes occur. The partition of the time-frequency plane shown here represents this Fourier transform behavior.

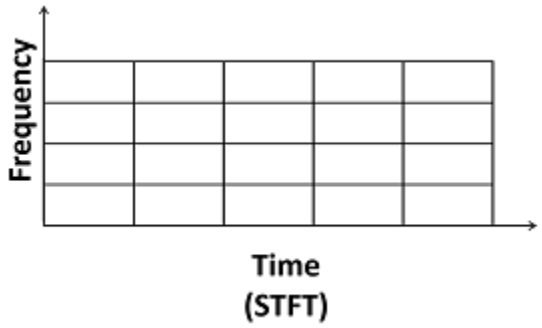
To perform a time-frequency analysis of a nonstationary signal, start with a real-valued even windowing function, $g(t)$, which is effectively nonzero over only a finite interval and has norm equal to one. In addition, the Fourier transform of $g(t)$ is centered at zero and is lowpass. Next, window *f(t)* with translates of $g(t)$. Then take the Fourier transform of the result

$$SF(u,\zeta) = \int f(t)g(t-u)e^{-j\zeta t}dt\,.$$

Correlating *f(t)* with the Gabor atoms, $g(t-u)e^{j\zeta t}$, is standard Gabor analysis. By varying u, you consider only values of *f(t)* near time u. The support of $g(t)$ determines the size of the neighborhood near time u. The Fourier transform of $g_{u,\zeta}(t) = g(t-u)e^{\zeta t}$ is the translation by ζ of the Fourier transform of $g(t)$ and is given by

$$\widehat{g}_{u,\zeta}(\omega) = e^{-(\omega-\zeta)}\widehat{g}(\omega-\zeta)\,.$$

The energy concentration of $\widehat{g}_{u,\zeta}(\omega)$ has variance σ_ω and is centered at ζ. If the window, $g_{u,\zeta}(t) = g(t-u)e^{\zeta t}$, shifts on a regular grid, the Fourier transform of the product of the shifted window and f(t) is the short-time Fourier transform (STFT). The STFT tiling of the time-frequency plane can be represented as a grid of boxes, each centered at (u, ζ):



The set of functions $\{g_{u,\zeta}\}$ is known as a *Gabor frame*. The elements of this set are called *Gabor atoms*. A frame is a set of functions, $\{h_k(t)\}$, that satisfy the following condition: there exist constants $0 < A \le B < \infty$ such that for any function *f(t)*,

$$A\|f\|^2 \le \Sigma_k |\langle f, h_k\rangle|^2 \le B\|f\|^2\,.$$

The energy concentration of $g(t)$, in time, has variance σ_t. The energy concentration of $\widehat{g}(\omega)$, in frequency, has variance σ_ω. The energy concentration determines how well the window localizes the signal in time and frequency. By the time-frequency uncertainty principle, there is a limit as to how well you can simultaneously localize in both time and frequency domains, as indicated by

$$\sigma_t\sigma_\omega \ge \frac{1}{2}\,.$$

Narrowing the window in one domain results in poorer localization in the other domain. Gabor showed that the area of the window is minimal when $g(t)$ is Gaussian.

Constant-Q Transform

In the CQT, the bandwidth and sampling density in frequency are varied. The windows are constructed and applied directly in the frequency domain. Different windows have different center

frequencies and bandwidths, but the ratio of the center frequency to bandwidth remains constant. Maintaining a constant ratio implies:

- Resolution in time improves at higher frequencies.
- Resolution in frequency improves at lower frequencies.

The time shifts for each window depend on the bandwidth, due to the uncertainty principle.

The CQT depends on:

- The window functions g_k are real-valued, even functions. In the frequency domain, the Fourier transform of g_k is defined on the interval, *[-Fs/2, Fs/2]*.
- The sampling rate, ζ_s.
- The number of bins per octave, b.
- The minimum and maximum frequencies, ζ_{min} and ζ_{max}.

Choose a minimum frequency ζ_{min} and number of bins per octave b. Next, form a sequence of geometrically spaced frequencies,

$$\zeta_k = \zeta_{min} \times 2^{k/b}$$

for $k = 0,\ldots,K$ where K is an integer such that ζ_K is the largest frequency strictly less than the Nyquist frequency $\zeta_s/2$. The bandwidth at the kth frequency is set to $\Omega_k = \zeta_{k+1}-\zeta_{k-1}$. Given this sampling, the ratio of the kth center frequency to the window bandwidth is independent of k:

$$Q = \zeta_k/\Delta_k = (2^{1/b}-2^{-1/b})^{-1}.$$

To ensure perfect reconstruction, the DC component and Nyquist frequency are prepended and appended, respectively, to the sequence.

$W(\omega)$ forms the window functions g_k. $W(\omega)$ is a real-valued, even continuous function that is centered at 0, positive in the interval [-½,½], and 0 elsewhere. $W(\omega)$ is translated to each center frequency ζ_k then scaled. Evaluating a scaled and translated version of $W(\omega)$ yields the filter coefficients $g_k[m]$, given by

$$g_k[m] = W((m\ \zeta_s/L - \zeta_k)/\Omega_k)$$

for $m = 0, \ldots, L-1$, where L is the signal length. By default, `cqt` uses the `'hann'` window.

By the uncertainty principle, the size of the bandwidth constrains the value of the time shifts. To satisfy the frame inequality, the shift a_k of g_k must satisfy

$$a_k \le \zeta_k/\Omega_k.$$

As mentioned previously, the window is applied in the frequency domain. The filters, g_k, centered at ζ_k, are formed and applied to the Fourier transform of the signal. Taking the inverse transform obtains the constant-Q coefficients.

References

[1] Holighaus, N., M. Dörfler, G.A. Velasco, and T. Grill. "A framework for invertible real-time constant-Q transforms." *IEEE Transactions on Audio, Speech, and Language Processing*. Vol. 21, No. 4, 2013, pp. 775–785.

[2] Velasco, G. A., N. Holighaus, M. Dörfler, and T. Grill. "Constructing an invertible constant-Q transform with nonstationary Gabor frames." In *Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11)*. Paris, France: 2011.

[3] Schörkhuber, C., A. Klapuri, N. Holighaus, and M. Dörfler. "A Matlab Toolbox for Efficient Perfect Reconstruction Time-Frequency Transforms with Log-Frequency Resolution." Submitted to the *AES 53rd International Conference on Semantic Audio*. London, UK: 2014.

[4] Průša, Z., P. L. Søndergaard, N. Holighaus, C. Wiesmeyr, and P. Balazs. *The Large Time-Frequency Analysis Toolbox 2.0*. Sound, Music, and Motion, Lecture Notes in Computer Science 2014, pp 419-442. https://github.com/ltfat

See Also

`cqt` | `icqt`

More About

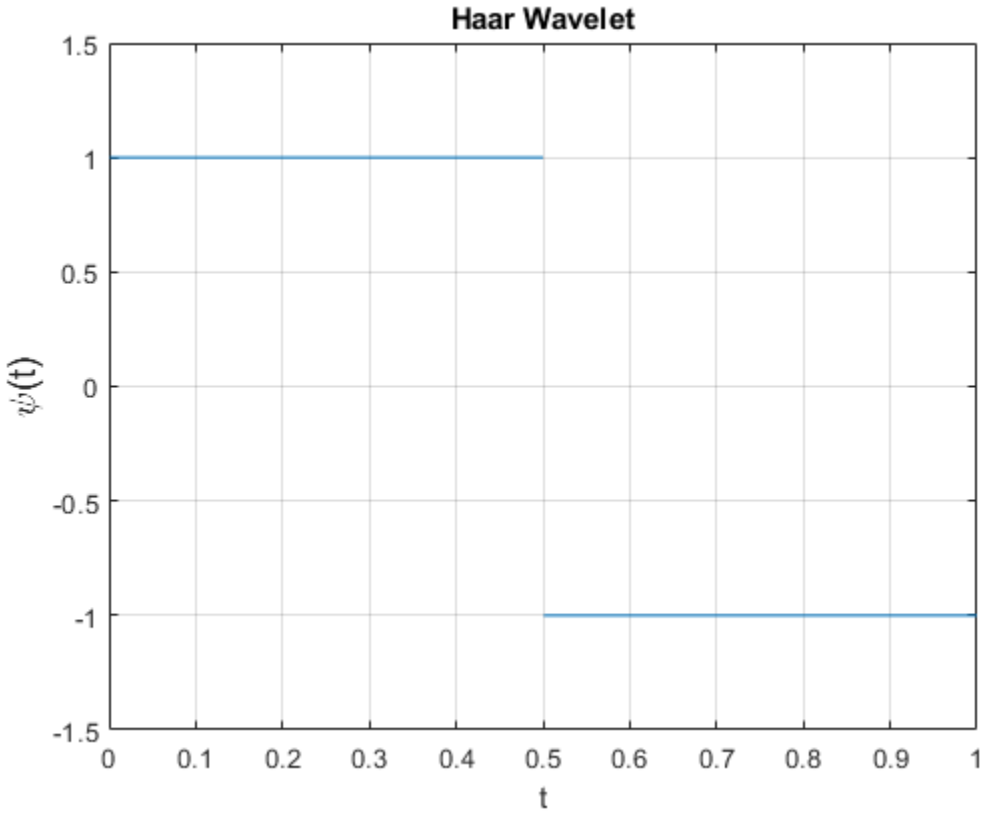
- “Short-Time Fourier Transform” on page 1-61

Haar Transforms for Time Series Data and Images

This example shows how to use Haar transforms to analyze time series data and images. To run all the code in this example, you must have Signal Processing Toolbox™ and Image Processing Toolbox™.

First, visualize the Haar wavelet.

```
[~,psi,x] = wavefun('haar',10);
x = x(2:end-1);
psi = psi(2:end-1);
hl = plot(x(1:512),psi(1:512));
grid on
hold on
line(x(513:end),psi(513:end))
xlabel('t')
ylabel('\psi(t)','fontsize',14)
title('Haar Wavelet')
```

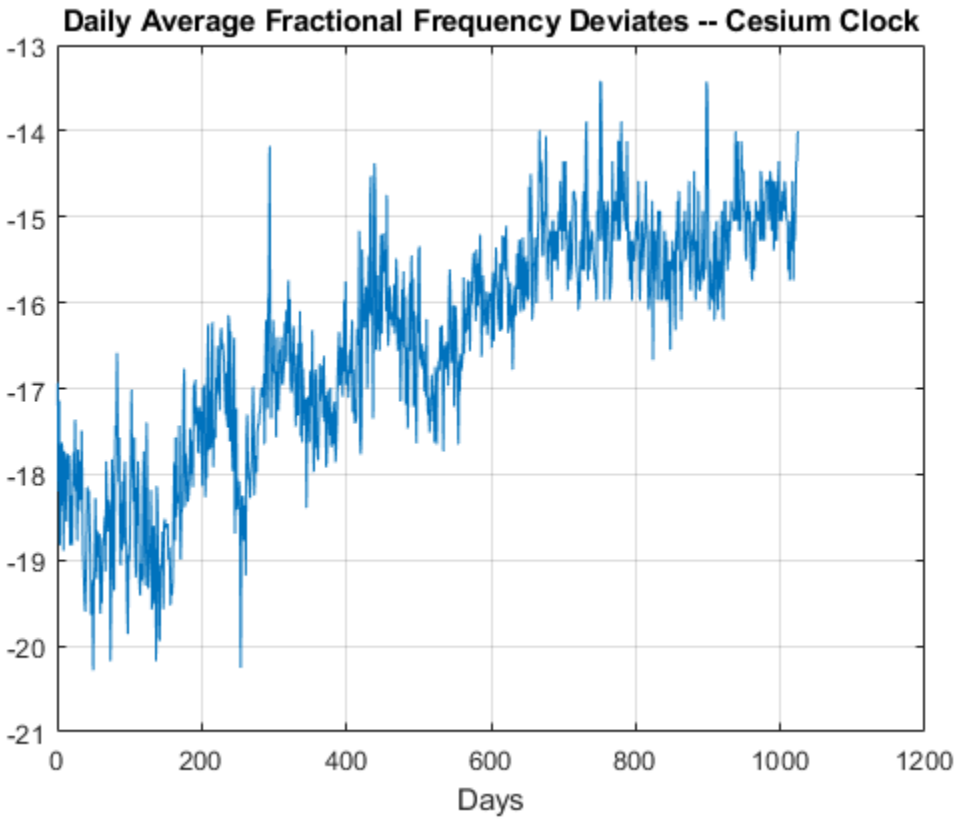


The Haar wavelet is discontinuous. As a result, it is typically not used in denoising or compression applications where smoothness of the reconstruction wavelet is an important consideration. However, Haar transforms are useful in a number of applications due to their superior time (spatial) localization and computational efficiency. The Wavelet Toolbox™ supports Haar analysis in most of the discrete wavelet analysis tools. This example features Haar lifting implementations which support integer-to-integer wavelet transforms for both 1-D and 2-D data and multichannel (multivariate) 1-D data.

Analyze Signal Variability By Scale

Load and plot the `clock_571` dataset. This example is essentially a recreation of the analysis presented in Percival & Walden [3], pp 13–16.

```
load clock_571;
figure;
plot(clock_571)
xlabel('Days')
grid on;
title('Daily Average Fractional Frequency Deviates -- Cesium Clock');
```

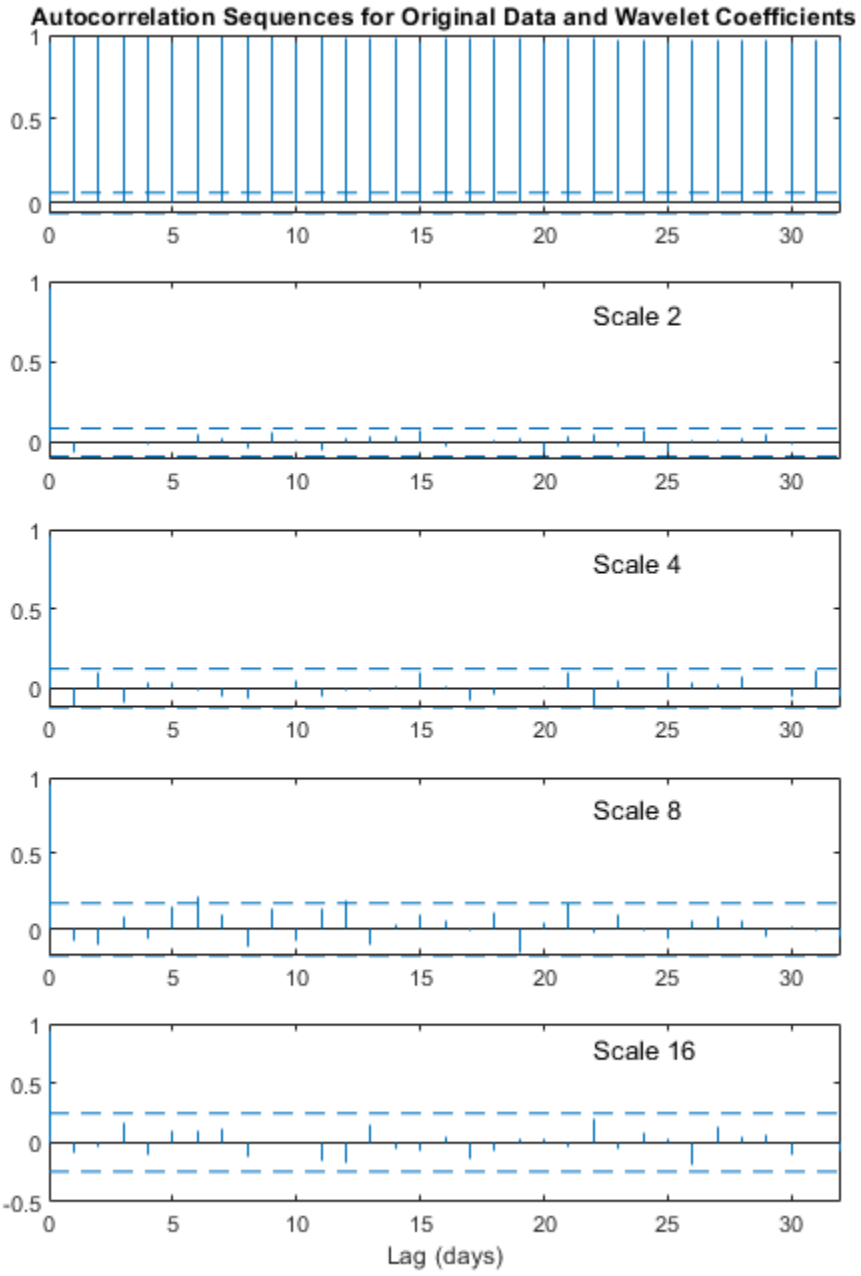


The data are daily average fractional frequency deviates for a particular cesium beam atomic clock with respect to the U.S. Naval Observatory master clock. If the value of the time series is 0, that means the cesium clock has neither lost nor gained time with respect to the master clock over the duration of the day. If the value is negative, the clock has lost time that day, a positive value means that the clock has gained time. For this data, the values are all negative. For certain applications, like geodesy, it is important to know whether there are certain time scales where the clock's deviation from the master clock is at its lowest value. In other words, are there certain scales where the clock agrees most closely with the master clock? The Haar transform is useful here because it possesses two important properties: It decorrelates data by scale and it partitions signal energy among scale.

To illustrate the decorrelating property, obtain the Haar transform down to level 6. Plot the autocorrelation sequence of the original data along with the autocorrelation of the wavelet coefficients by scale for scales of 2,4,8,16, and 32 days. Dashed lines on the plots delineate 95%

confidence intervals for white noise inputs. Values exceeding those lines are indicative of significant autocorrelation in the data.

```
[s,w] = haart(clock_571,6);
helperAutoCorr(clock_571,w);
```



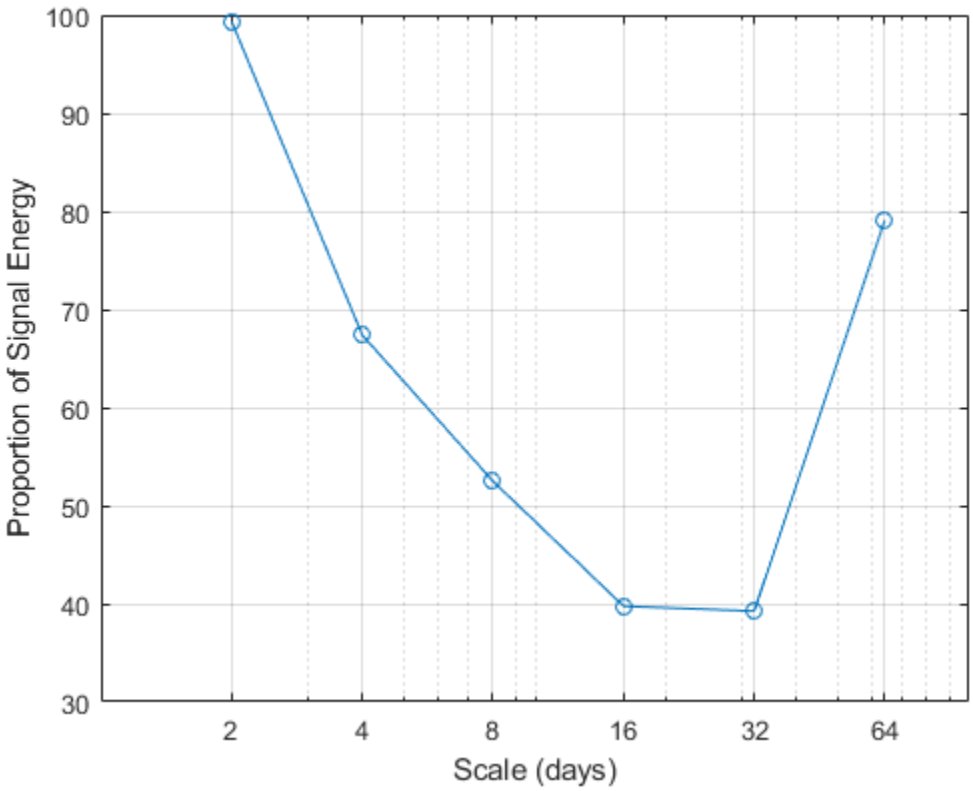
The top plot shows the autocorrelation sequence for the original data. Subsequent plots show the autocorrelation sequences for wavelet coefficients at increasingly coarser scales. It is clear that the autocorrelation sequence of the original data exhibits correlation at all lags while the Haar transform coefficients are decorrelated. Next, demonstrate energy conservation.

```
sigenergy = norm(clock_571,2)^2
energyByScale = cellfun(@(x)norm(x,2)^2,w);
haarenergy = norm(s,2)^2+sum(energyByScale)
```

```
sigenergy =

   2.7964e+05


haarenergy =

   2.7964e+05
```

The total signal energy is preserved by the Haar transform. Because of these properties, you can make meaningful inferences based on the proportion of signal energy captured by the wavelet coefficients at each scale.

```
scales = 2.^(1:6);
figure
plot(scales,energyByScale,'-o')
xlabel('Scale (days)')
set(gca,'xscale','log')
set(gca,'xtick',2.^(1:6))
ylabel('Proportion of Signal Energy')
grid on
```



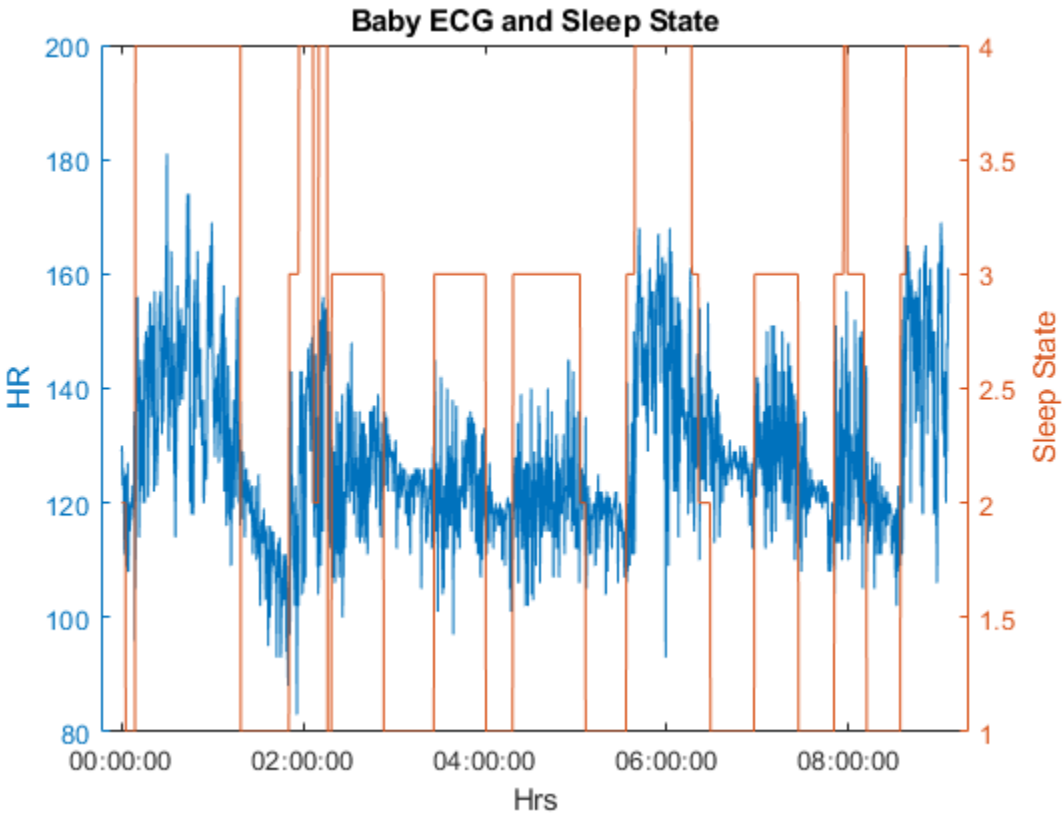
You see that the energy is at a minimum for scales of 16 and 32 days. For the Haar wavelet (and all Daubechies wavelets), the wavelet coefficients at a given scale represent differences between weighted averages of the data over a duration 1/2 the length of the scale. This plot indicates scales over which the cesium clock is in best agreement with the master clock. This means that considering data over approximately two-week or even one-month periods is more accurate than data on smaller or longer scales. As previously mentioned, this has important implications for geodesy where extremely precise time measurements are critical.

Although the Haar wavelet is discontinuous, it is still effective at representing various kinds of time series. Examples include count data and data where values of a time series are tied to some specific state, which affects the level of the time series. As an example, consider the relationship between heart rate and sleep state.

Create Signal Approximations

The data consist of two time series. One time series is the heart rate of a 66-day old infant sampled every 16 seconds for just over 9 hours. The heart rate time series is integer-valued. The other time series is the expertly scored sleep state of the same infant over the same period with the same sampling rate. The sleep state data was scored based on the infant's EEG and EOG (eye movement) data, not based on the heart rate. The sleep state codes are 1=quiet sleep, 2=between quiet and active sleep, 3=active sleep, and 4=awake. Both time series were recorded by Prof. Peter Fleming, Dr Andrew Sawczenko, and Jeanine Young of the Institute of Child Health, Royal Hospital for Sick Children, Bristol and kindly provided for use in this example. Plot the heart rate data along with the sleep states.

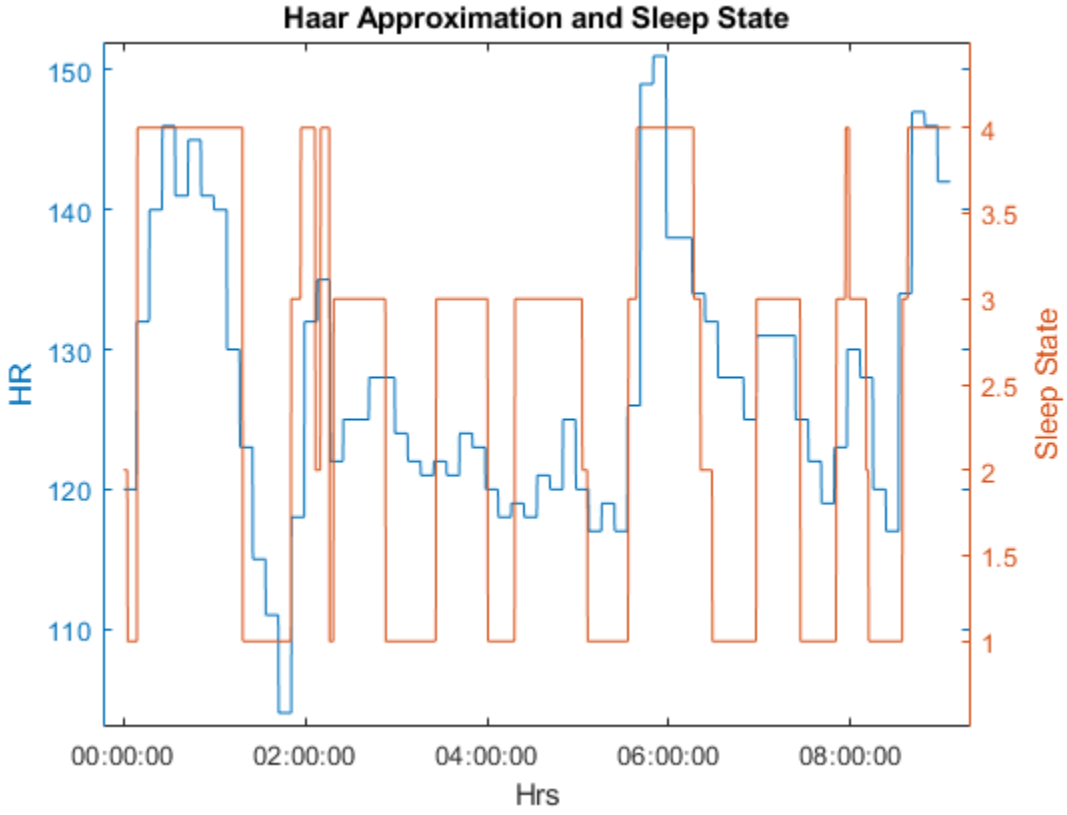
```
load BabyECGData
figure
yyaxis left
plot(times,HR)
ylabel('HR')
xlabel('Hrs')
YLim = [min(HR)-1 max(HR)+1];
yyaxis right
plot(times,SS)
ylabel('Sleep State')
YLim = [0.5 4.5];
title('Baby ECG and Sleep State')
```



An inspection of the data reveals an apparent correlation between sleep state and heart rate, but the data is extremely noisy. Because the Haar transform provides a staircase approximation to a signal, it is often useful in situations where a response is dependent on a predictor variable with a small number of discrete states. Here the discrete states are the four sleep stages. Obtain the Haar approximation of the heart rate data using a level 5 approximation. Because the heart rate data is integer-valued, use the `'integer'` flag to ensure that integer-valued data is returned. Plot the result.

```
[S,W] = haart(HR,'integer');
HaarHR = ihaart(S,W,5,'integer');
figure
hL = plotyy(times,HaarHR,times,SS);
Ax1 = hL(1);
Ax2 = hL(2);
```

```
Ax1.YLim = [min(HaarHR)-1 max(HaarHR)+1];
Ax1.YLabel.String = 'HR';
Ax2.YLim = [0.5 4.5];
Ax2.YLabel.String = 'Sleep State';
xlabel('Hrs')
title('Haar Approximation and Sleep State')
```



The Haar approximation more clearly shows the relationship between the sleep state and the heart rate data. You can assess this change by looking at the correlation between the raw data and the sleep state time series.

```
corr(SS,HR)
```

```
ans =

    0.5576
```

Now compare the value of 0.56 with the correlation between the sleep state data and the Haar approximation

```
corr(SS,HaarHR)
```

```
ans =
```

```
    0.6907
```

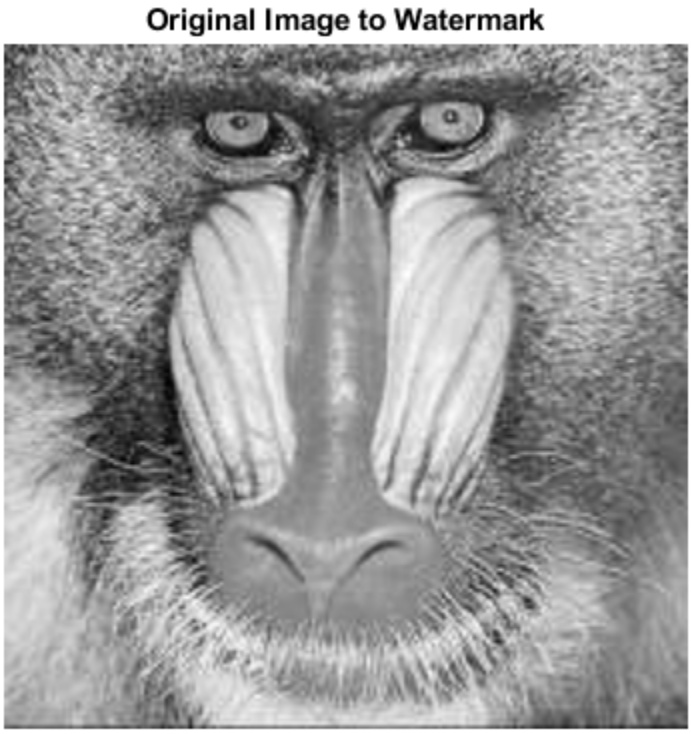
The correlation has increased from 0.56 to 0.69. More advanced wavelet analysis and modeling of this data is presented in Nason, von Sachs, & Kroisandt [1] and Nason, Sapatinas, & Sawczenko [2].

Digital Watermarking of Images

Watermarking is an important data protection tool. It is a passive protection technique where a marker is covertly inserted in some data in order to verify the authenticity or integrity of the data. Wavelet techniques in general and the Haar transform in particular are frequently employed in watermarking images. This example illustrates the use of the Haar transform in watermarking an image and recovering the watermark. The example employs a simple watermarking scheme chosen for ease of illustration. In this scheme, the watermark is inserted in the approximation coefficients at level 3.

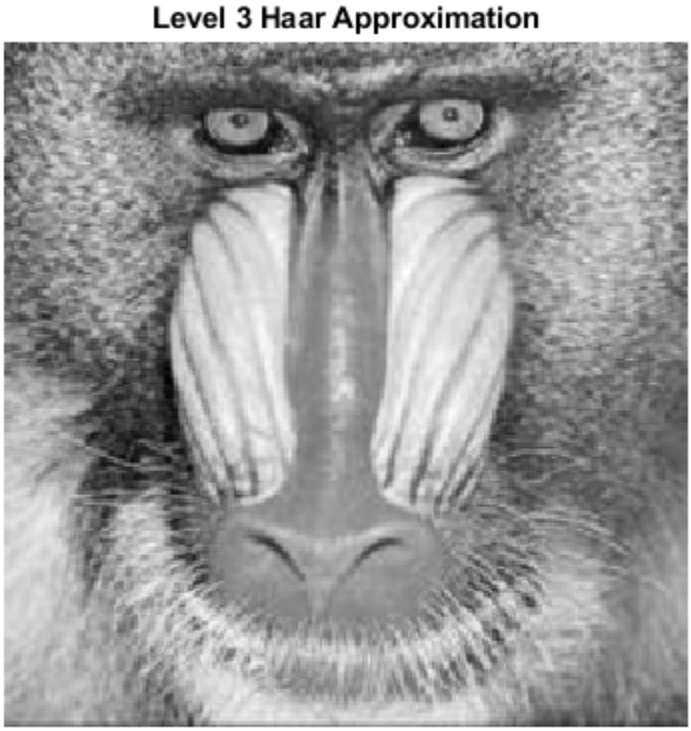
Watermark an image of a Mandril with one of a honey badger. Read in the Mandrill image. Resize it to 2048x2048 and display the result.

```
coverIM = imread('mandrill.jpg');
coverIM = rgb2gray(coverIM);
coverIM = imresize(im2double(coverIM),[2048 2048]);
imagesc(coverIM)
colormap gray
title('Original Image to Watermark')
axis off
axis square
```



Obtain the Haar transform of the Mandrill image down to level 3.

```
[LLorig,LHorig,HLorig,HHorig] = haart2(coverIM,3);
imagesc(LLorig)
title('Level 3 Haar Approximation')
axis off
axis square
```



Read in watermark image and resize it.

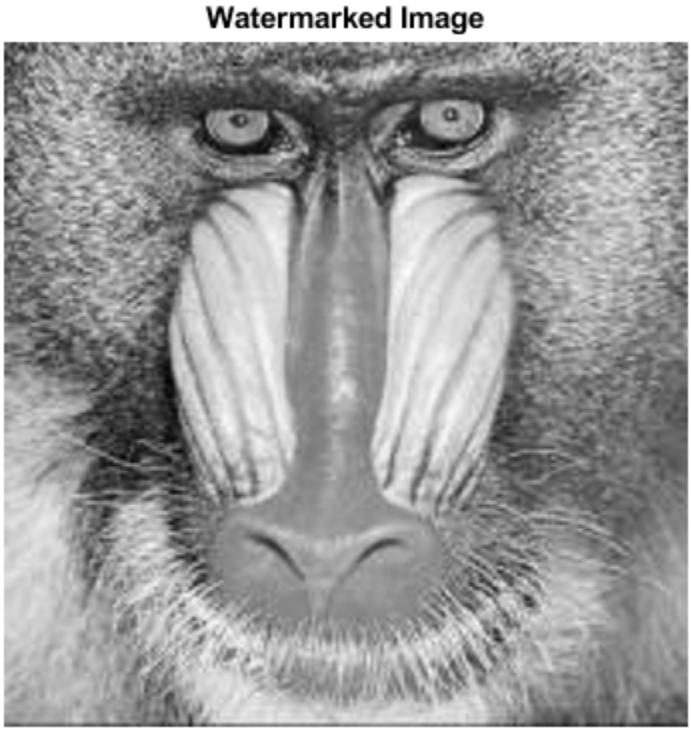
```
watermark = imread('honeybadger.jpg');
watermark = im2double(rgb2gray(watermark));
watermark = imresize(watermark,[2048 2048]);
```

Obtain the Haar transform of the watermark image down to level 3.

```
[LLw,LHw,HLw,HHw] = haart2(watermark,3);
imagesc(LLw)
colormap gray
title('Level 3 Haar Approximation--Watermark')
axis off
axis square
```

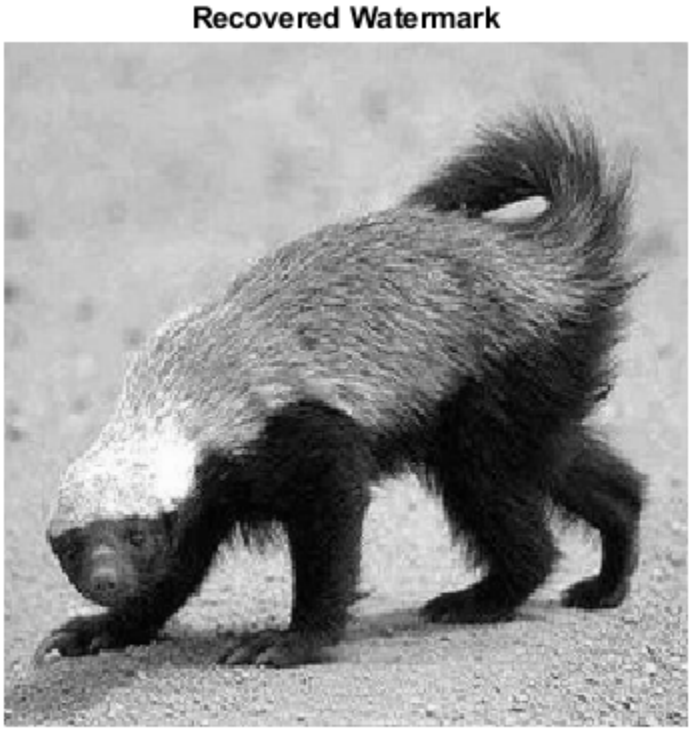
Add the honey badger watermark to the Mandril image by attenuating the level-3 approximation coefficients of the watermark and inserting the attenuated coefficients into the level-3 Mandrill approximation coefficients.

```
LLwatermarked = LLorig+1e-4*LLw;
markedIM = ihaart2(LLwatermarked,LHorig,HLorig,HHorig);
imagesc(markedIM)
title('Watermarked Image')
axis off
axis square
colormap gray
```



The watermark (honey badger) is not visible in the watermarked image. Because you know what algorithm was used to insert the watermark, you can recover the watermark using the Haar transform.

```
[LLr,LHr,HLr,HHr] = haart2(markedIM,3);
LLmarked = (LLr-LLorig).*1e4;
imagesc(LLmarked)
title('Recovered Watermark')
colormap gray
axis off
axis square
```


Recovered Watermark

From Fourier Analysis to Wavelet Analysis

In this section...
“Inner Products” on page 1-58
“Fourier Transform” on page 1-59
“Short-Time Fourier Transform” on page 1-61

Inner Products

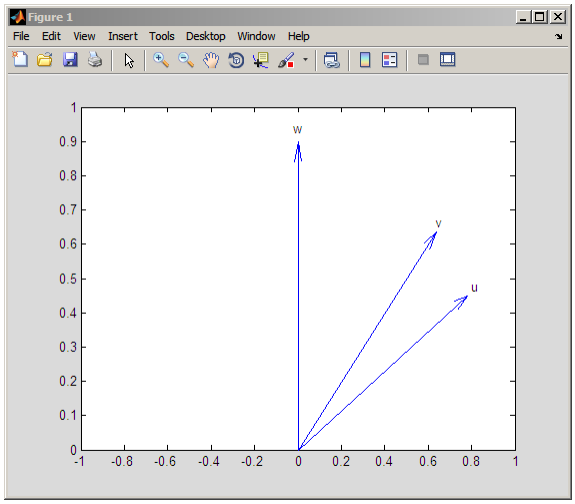
Both the Fourier and wavelet transforms measure similarity between a signal and an *analyzing* function. Both transforms use a mathematical tool called an *inner product* as this measure of similarity. The two transforms differ in their choice of analyzing function. This results in the different way the two transforms represent the signal and what kind of information can be extracted.

As a simple example of the inner product as a measure of similarity, consider the inner product of vectors in the plane. The following MATLAB example calculates the inner product of three unit vectors, $\{u, v, w\}$, in the plane:

$$\left\{\begin{pmatrix}\sqrt{3}/2\\ 1/2\end{pmatrix}, \begin{pmatrix}1/\sqrt{2}\\ 1/\sqrt{2}\end{pmatrix}, \begin{pmatrix}0\\ 1\end{pmatrix}\right\}$$

```
u = [sqrt(3)/2 1/2];
v = [1/sqrt(2) 1/sqrt(2)];
w = [0 1];
% Three unit vectors in the plane
quiver([0 0 0],[0 0 0],[u(1) v(1) w(1)],[u(2) v(2) w(2)]);
axis([-1 1 0 1]);
text(-0.020,0.9371,'w');
text(0.6382,0.6623,'v');
text(0.7995,0.4751,'u');
% Compute inner products and print results
fprintf('The inner product of u and v is %1.2f\n', dot(u,v))
fprintf('The inner product of v and w is %1.2f\n', dot(w,v))
fprintf('The inner product of u and w is %1.2f\n', dot(u,w))
```

Looking at the figure, it is clear that u and v are most similar in their orientation, while u and w are the most dissimilar.



The inner products capture this geometric fact. Mathematically, the inner product of two vectors, u and v is equal to the product of their norms and the cosine of the angle, θ, between them:

$$< u, v > \; = \; | \, |u| \, | \, | \, |v| \, | \cos(\theta)$$

For the special case when both u and v have unit norm, or unit energy, the inner product is equal to cos(θ) and therefore lies between [-1,1]. In this case, you can interpret the inner product directly as a correlation coefficient. If either u or v does not have unit norm, the inner product may exceed 1 in absolute value. However, the inner product still depends on the cosine of the angle between the two vectors making it interpretable as a kind of correlation. Note that the absolute value of the inner product is largest when the angle between them is either 0 or π radians (0 or 180 degrees). This occurs when one vector is a real-valued scalar multiple of the other.

While inner products in higher-dimensional spaces like those encountered in the Fourier and wavelet transforms do not exhibit the same ease of geometric interpretation as the previous example, they measure similarity in the same way. A significant part of the utility of these transforms is that they essentially summarize the correlation between the signal and some basic functions with certain physical properties, like frequency, scale, or position. By summarizing the signal in these constituent parts, we are able to better understand the mechanisms that produced the signal.

Fourier Transform

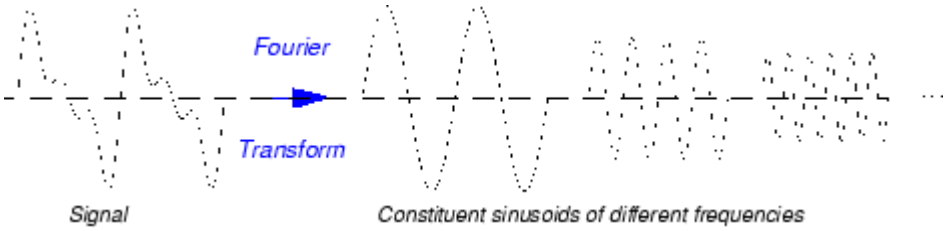
Fourier analysis is used as a starting point to introduce the wavelet transforms, and as a benchmark to demonstrate cases where wavelet analysis provides a more useful characterization of signals than Fourier analysis.

Mathematically, the process of Fourier analysis is represented by the *Fourier transform*:

$$F(\omega) = \int_{\infty}^{\infty} f(t) e^{-j\omega t} dt \, .$$

which is the integral (sum) over all time of the signal f(t) multiplied by a complex exponential. Recall that a complex exponential can be broken down into real and imaginary sinusoidal components. Note that the Fourier transform maps a function of a single variable into another function of a single variable.

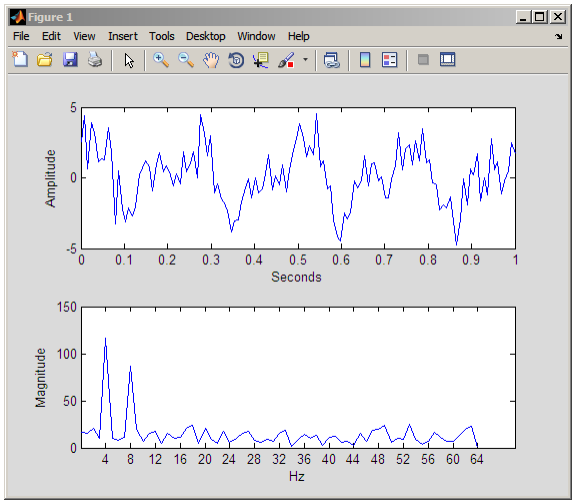
The integral defining the Fourier transform is an *inner product*. See “Inner Products” on page 1-58 for an example of how inner products measure of similarity between two signals. For each value of ω, the integral (or sum) over all values of time produces a scalar, F(ω), that summarizes how similar the two signals are. These complex-valued scalars are the *Fourier coefficients*. Conceptually, multiplying each Fourier coefficient, F(ω), by a complex exponential (sinusoid) of frequency ω yields the constituent sinusoidal components of the original signal. Graphically, the process looks like



Because $e^{j\omega t}$ is complex-valued, F(ω) is, in general, complex-valued. If the signal contains significant oscillations at an angular frequency of ω_0, the absolute value of $F(\omega_0)$ will be large. By examining a plot of $|F(\omega)|$ as a function of angular frequency, it is possible to determine what frequencies contribute most to the variability of f(t).

To illustrate how the Fourier transform captures similarity between a signal and sinusoids of different frequencies, the following MATLAB code analyzes a signal consisting of two sinusoids of 4 and 8 Hertz (Hz) corrupted by additive noise using the discrete Fourier transform.

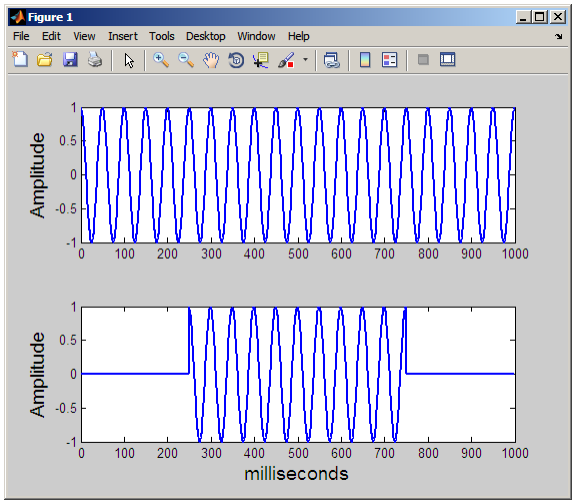
```
rng(0,'twister');
Fs = 128;
t = linspace(0,1,128);
x = 2*cos(2*pi*4*t)+1.5*sin(2*pi*8*t)+randn(size(t));
xDFT = fft(x);
Freq = 0:64;
subplot(211);
plot(t,x); xlabel('Seconds'); ylabel('Amplitude');
subplot(212);
plot(Freq,abs(xDFT(1:length(xDFT)/2+1)))
set(gca,'xtick',[4:4:64]);
xlabel('Hz'); ylabel('Magnitude');
```



Viewed as a time signal, it is difficult to determine what significant oscillations are present in the data. However, looking at the absolute value of the Fourier transform coefficients as function of frequency, the dominant oscillations at 4 and 8 Hz are easy to detect.

Short-Time Fourier Transform

The Fourier transform summarizes the similarity between a signal and a sinusoid with a single complex number. The magnitude of the complex number captures the degree to which oscillations at a particular frequency contribute to the signal's energy, while the argument of the complex number captures phase information. Note that the Fourier coefficients have no time dependence. The Fourier coefficients are obtained by integrating, or summing, over all time, so it is clear that this information is lost. Consider the following two signals:



Both signals consist of a single sine wave with a frequency of 20 Hz. However, in the top signal, the sine wave lasts the entire 1000 milliseconds. In the bottom plot, the sine wave starts at 250 and ends at 750 milliseconds. The Fourier transform detects that the two signals have the same frequency content, but has no way of capturing that the duration of the 20 Hz oscillation differs between the two signals. Further, the Fourier transform has no mechanism for marking the beginning and end of the intermittent sine wave.

In an effort to correct this deficiency, Dennis Gabor (1946) adapted the Fourier transform to analyze only a small section of the signal at a time -- a technique called *windowing* the signal. Gabor's adaptation is called the short-time Fourier transform (STFT). The technique works by choosing a time function, or window, that is essentially nonzero only on a finite interval. As one example consider the following Gaussian window function:

$$w(t) = \sqrt{\frac{\alpha}{\pi}} e^{-\alpha t^2}$$

The Gaussian function is centered around t=0 on an interval that depends on the value of α. Shifting the Gaussian function by τ results in:

$$w(t-\tau) = \sqrt{\frac{\alpha}{\pi}} e^{-\alpha (t-\tau)^2},$$

which centers the Gaussian window around τ. Multiplying a signal by $w(t-\tau)$ selects a portion of the signal centered at τ. Taking the Fourier transform of these windowed segments for different values of τ, produces the STFT. Mathematically, this is:

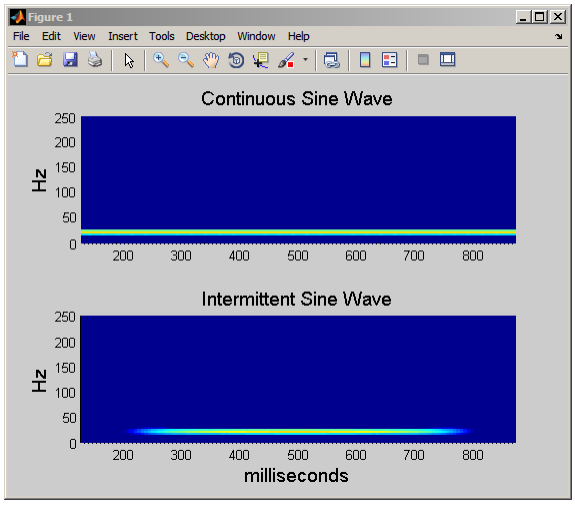
$$F(\omega,\tau)=\int f(t)w(t-\tau)e^{-j\omega t}dt$$

The STFT maps a function of one variable into a function of two variables, ω and τ. This 2-D representation of a 1-D signal means that there is redundancy in the STFT. The following figure demonstrates how the STFT maps a signal into a time-frequency representation.

The STFT represents a sort of compromise between time- and frequency-based views of a signal. It provides some information about both when and at what frequencies a signal event occurs. However, you can only obtain this information with limited precision, and that precision is determined by the size of the window.

While the STFT compromise between time and frequency information can be useful, the drawback is that once you choose a particular size for the time window, that window is the same for all frequencies. Many signals require a more flexible approach -- one where you can vary the window size to determine more accurately either time or frequency.

Instead of plotting the STFT in three dimensions, the convention is to code $|F(\omega,\tau)|$ as intensity on some color map. Computing and displaying the STFT of the two 20-Hz sine waves of different duration shown previously:



By using the STFT, you can see that the intermittent sine wave begins near 250 msec and ends around 750 msec. Additionally, you can see that the signal's energy is concentrated around 20 Hz.

Continuous Wavelet Transform and Scale-Based Analysis

In this section...
“Definition of the Continuous Wavelet Transform” on page 1-63
“Scale” on page 1-64
“Shifting” on page 1-66
“CWT as a Windowed Transform” on page 1-66

Definition of the Continuous Wavelet Transform

Like the Fourier transform, the *continuous wavelet transform* (CWT) uses inner products to measure the similarity between a signal and an analyzing function. In the Fourier transform, the analyzing functions are complex exponentials, $e^{j\omega t}$. The resulting transform is a function of a single variable, ω. In the short-time Fourier transform, the analyzing functions are windowed complex exponentials, $w(t)e^{j\omega t}$, and the result is a function of two variables. The STFT coefficients, $F(\omega, \tau)$, represent the match between the signal and a sinusoid with angular frequency ω in an interval of a specified length centered at τ.

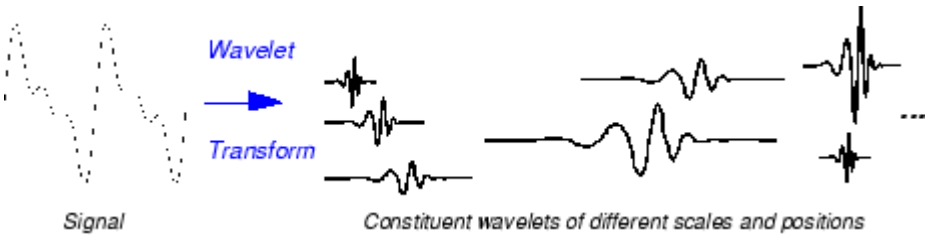
In the CWT, the analyzing function is a wavelet, ψ. The CWT compares the signal to shifted and compressed or stretched versions of a wavelet. Stretching or compressing a function is collectively referred to as *dilation* or *scaling* and corresponds to the physical notion of *scale*. By comparing the signal to the wavelet at various scales and positions, you obtain a function of two variables. The 2-D representation of a 1-D signal is redundant. If the wavelet is complex-valued, the CWT is a complex-valued function of scale and position. If the signal is real-valued, the CWT is a real-valued function of scale and position. For a scale parameter, a>0, and position, b, the CWT is:

$$C(a,b;f(t),\psi(t)) = \int_{-\infty}^{\infty} f(t)\frac{1}{a}\psi^*\left(\frac{t-b}{a}\right)dt$$

where * denotes the complex conjugate. Not only do the values of scale and position affect the CWT coefficients, the choice of wavelet also affects the values of the coefficients.

By continuously varying the values of the scale parameter, a, and the position parameter, b, you obtain the *cwt coefficients C(a,b)*. Note that for convenience, the dependence of the CWT coefficients on the function and analyzing wavelet has been suppressed.

Multiplying each coefficient by the appropriately scaled and shifted wavelet yields the constituent wavelets of the original signal.



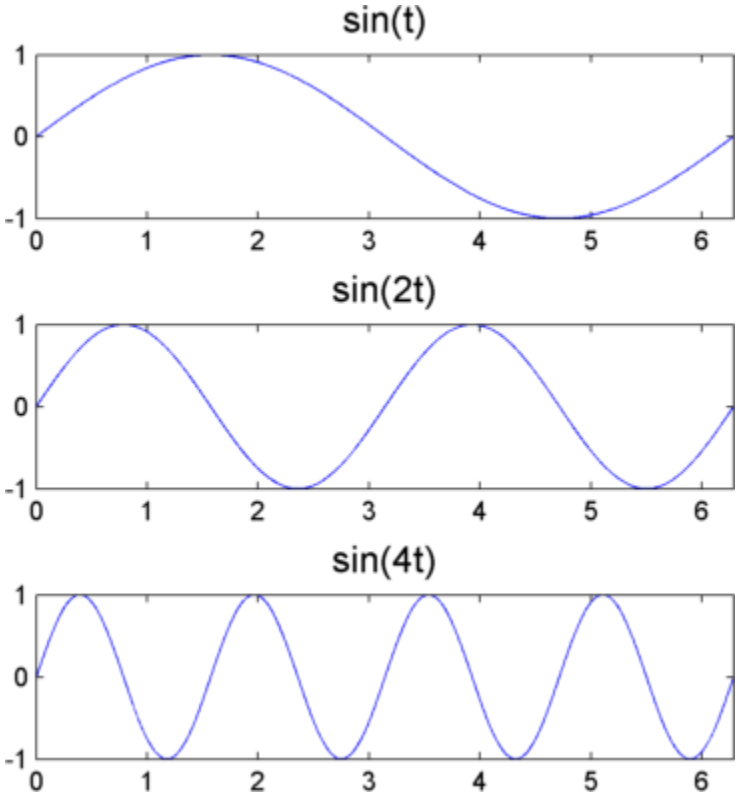
There are many different admissible wavelets that can be used in the CWT. While it may seem confusing that there are so many choices for the analyzing wavelet, it is actually a strength of wavelet analysis. Depending on what signal features you are trying to detect, you are free to select a wavelet

that facilitates your detection of that feature. For example, if you are trying to detect abrupt discontinuities in your signal, you may choose one wavelet. On the other hand, if you are interested in finding oscillations with smooth onsets and offsets, you are free to choose a wavelet that more closely matches that behavior.

Scale

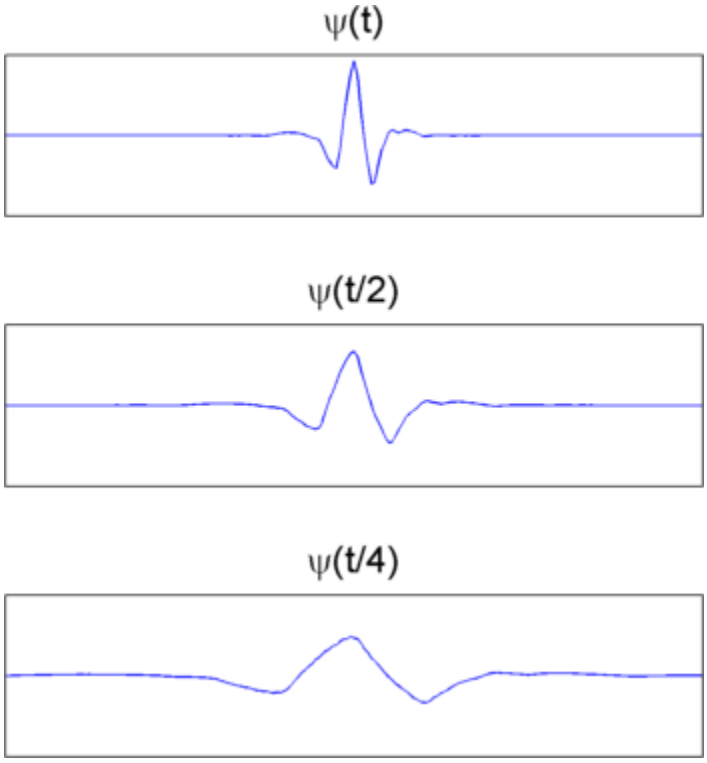
Like the concept of frequency, *scale* is another useful property of signals and images. For example, you can analyze temperature data for changes on different scales. You can look at year-to-year or decade-to-decade changes. Of course, you can examine finer (day-to-day), or coarser scale changes as well. Some processes reveal interesting changes on long time, or spatial scales that are not evident on small time or spatial scales. The opposite situation also happens. Some of our perceptual abilities exhibit *scale invariance*. You recognize people you know regardless of whether you look at a large portrait, or small photograph.

To go beyond colloquial descriptions such as “stretching” or “shrinking” we introduce the *scale factor*, often denoted by the letter *a*. The scale factor is an inherently positive quantity, $a>0$. For sinusoids, the effect of the scale factor is very easy to see.



In sin(at), the scale is the inverse of the radian frequency, *a*.

The scale factor works exactly the same with wavelets. The smaller the scale factor, the more “compressed” the wavelet. Conversely, the larger the scale, the more stretched the wavelet. The following figure illustrates this for wavelets at scales 1,2, and 4.



This general inverse relationship between scale and frequency holds for signals in general.

Not only is a time-scale representation a different way to view data, it is a very natural way to view data derived from a great number of natural phenomena.

Scale and Frequency

There is clearly a relationship between scale and frequency. Recall that longer scales correspond to the most “stretched” wavelets. The more stretched the wavelet, the longer the portion of the signal with which it is being compared, and therefore the coarser the signal features measured by the wavelet coefficients.

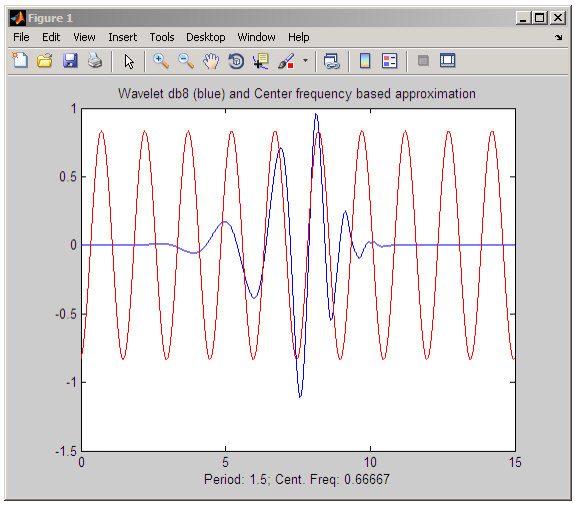
To summarize, the general correspondence between scale and frequency is:

- Small scale a ⇒ Compressed wavelet ⇒ Rapidly changing details ⇒ High frequency ω.
- Long scale a ⇒ Stretched wavelet ⇒ Slowly changing, coarse features ⇒ Low frequency ω.

While there is a general relationship between scale and frequency, no precise relationship exists. Users familiar with Fourier analysis often want to define a mapping between a wavelet at a given scale with a specified sampling period to a frequency in hertz. You can only do this in a general sense. Therefore, it is better to talk about the pseudo-frequency corresponding to a scale. The Wavelet

Toolbox software provides two functions `centfrq` and `scal2frq`, which enable you to find these approximate scale-frequency relationships for specified wavelets and scales.

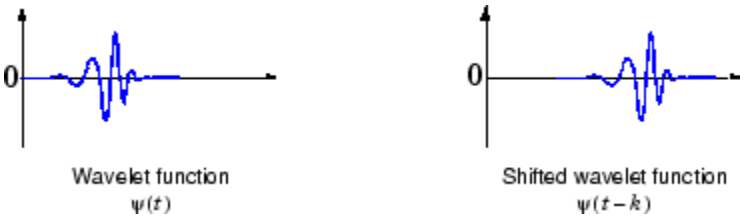
The basic approach identifies the peak power in the Fourier transform of the wavelet as its center frequency and divides that value by the product of the scale and the sampling interval. See `scal2frq` for details. The following example shows the match between the estimated center frequency of the `db8` wavelet and a sinusoid of the same frequency.



The relationship between scale and frequency in the CWT is also explored in “Continuous Wavelet Transform as a Bandpass Filter” on page 1-68.

Shifting

Shifting a wavelet simply means delaying (or advancing) its onset. Mathematically, delaying a function f(t) by k is represented by f(t – k):

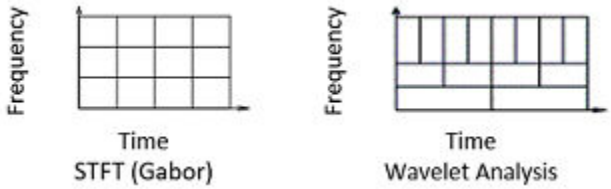


CWT as a Windowed Transform

In “Short-Time Fourier Transform” on page 1-61, the STFT is described as a windowing of the signal to create a local frequency analysis. A shortcoming of the STFT approach is that the window size is constant. There is a trade off in the choice of window size. A longer time window improves frequency resolution while resulting in poorer time resolution because the Fourier transform loses all time resolution over the duration of the window. Conversely, a shorter time window improves time localization while resulting in poorer frequency resolution.

Wavelet analysis represents the next logical step: a windowing technique with variable-sized regions. Wavelet analysis allows the use of long time intervals where you want more precise low-frequency information, and shorter regions where you want high-frequency information.

The following figure contrasts the different ways the STFT and wavelet analysis decompose the time-frequency plane.



Continuous Wavelet Transform as a Bandpass Filter

In this section...
“CWT as a Filtering Technique” on page 1-68 “DFT-Based Continuous Wavelet Transform” on page 1-70

CWT as a Filtering Technique

The continuous wavelet transform (CWT) computes the inner product of a signal, $f(t)$, with translated and dilated versions of an analyzing wavelet, $\psi(t)$. The definition of the CWT is:

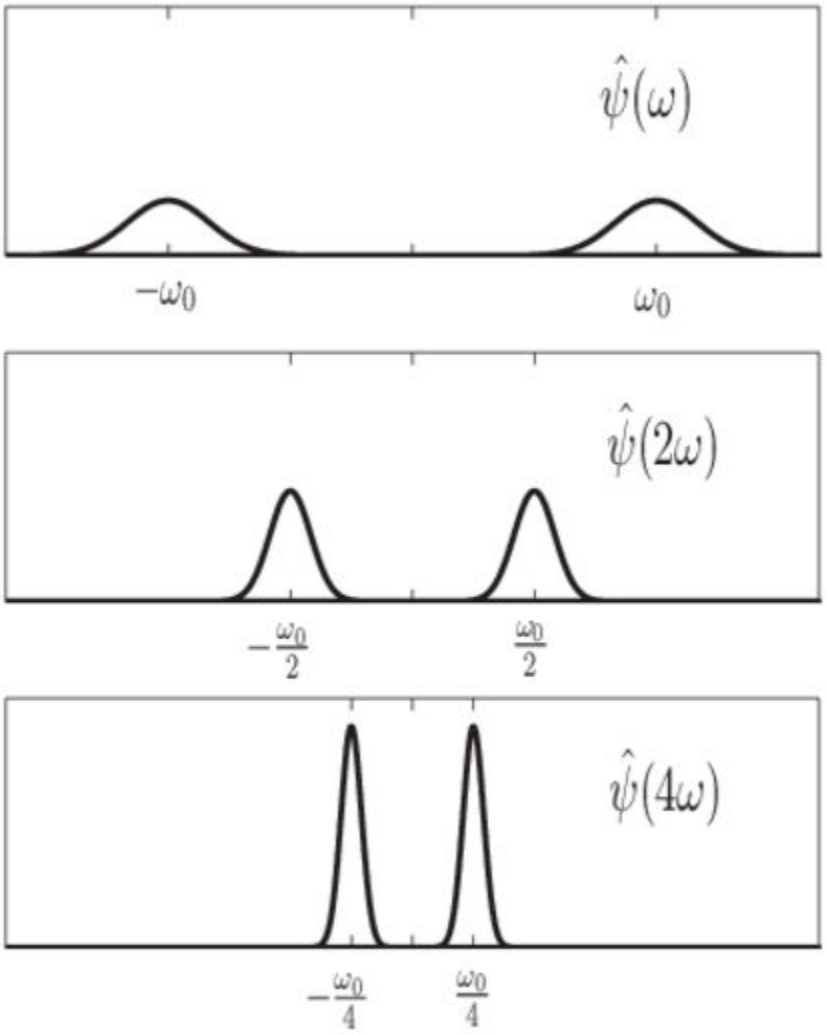
$$C(a,b;f(t),\psi(t)) = \int_{-\infty}^{\infty} f(t)\frac{1}{a}\psi^*\left(\frac{t-b}{a}\right)dt$$

You can also interpret the CWT as a frequency-based filtering of the signal by rewriting the CWT as an inverse Fourier transform.

$$C(a,b;f(t),\psi(t)) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \widehat{f}\,(\omega)\overline{\widehat{\psi}}(a\omega)e^{i\omega b}d\omega$$

where $\widehat{f}\,(\omega)$ and $\widehat{\psi}(\omega)$ are the Fourier transforms of the signal and the wavelet.

From the preceding equations, you can see that stretching a wavelet in time causes its support in the frequency domain to shrink. In addition to shrinking the frequency support, the center frequency of the wavelet shifts toward lower frequencies. The following figure demonstrates this effect for a hypothetical wavelet and scale (dilation) factors of 1,2, and 4.



This depicts the CWT as a bandpass filtering of the input signal. CWT coefficients at lower scales represent energy in the input signal at higher frequencies, while CWT coefficients at higher scales represent energy in the input signal at lower frequencies. However, unlike Fourier bandpass filtering, the width of the bandpass filter in the CWT is inversely proportional to scale. The width of the CWT *filters* decreases with increasing scale. This follows from the *uncertainty* relationships between the time and frequency support of a signal: the broader the support of a signal in time, the narrower its support in frequency. The converse relationship also holds.

In the wavelet transform, the scale, or dilation operation is defined to preserve energy. To preserve energy while shrinking the frequency support requires that the peak energy level increases. The implementation of `cwt` in Wavelet Toolbox uses L1 normalization. The *quality factor*, or *Q factor* of a filter is the ratio of its peak energy to bandwidth. Because shrinking or stretching the frequency support of a wavelet results in commensurate increases or decreases in its peak energy, wavelets are often referred to as constant-Q filters.

DFT-Based Continuous Wavelet Transform

The equation in the preceding section defined the CWT as the inverse Fourier transform of a product of Fourier transforms.

$$C(a,b;f(t),\psi(t)) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{f}(\omega)\widehat{\psi}^{*}(a\omega)e^{j\omega b}d\omega$$

The *time* variable in the inverse Fourier transform is the translation parameter, b.

This suggests that you can compute the CWT with the inverse Fourier transform. Because there are efficient algorithms for the computation of the discrete Fourier transform and its inverse, you can often achieve considerable savings by using `fft` and `ifft` when possible.

To obtain a picture of the CWT in the Fourier domain, start with the definition of the wavelet transform:

$$< f(t), \psi_{a,b}(t) > \; = \frac{1}{a}\int_{-\infty}^{\infty} f(t)\psi^{*}(\tfrac{t-b}{a})dt$$

If you define:

$$\widetilde{\psi}_{a}(t) = \frac{1}{a}\psi^{*}(-t/a)$$

you can rewrite the wavelet transform as

$$(f * \widetilde{\psi}_{a})(b) = \int_{-\infty}^{\infty} f(t)\widetilde{\psi}_{a}(b-t)dt$$

which explicitly expresses the CWT as a convolution.

To implement the discretized version of the CWT, assume that the input sequence is a length N vector, *x[n]*. The discrete version of the preceding convolution is:

$$W_{a}[b] = \sum_{n=0}^{N-1} x[n]\,\widetilde{\psi}_{a}[b-n]$$

To obtain the CWT, it appears you have to compute the convolution for each value of the shift parameter, b, and repeat this process for each scale, a.

However, if the two sequences are circularly-extended (periodized to length N), you can express the circular convolution as a product of discrete Fourier transforms. The CWT is the inverse Fourier transform of the product

$$W_{a}(b) = \frac{1}{N}\sqrt{\frac{2\pi}{\Delta t}}\sum_{k=0}^{N-1} \hat{X}(2\pi k/N\Delta t)\hat{\psi}^{*}(a2\pi k/N\Delta t)e^{j2\pi kb/N}$$

where Δt is the sampling interval (period).

Expressing the CWT as an inverse Fourier transform enables you to use the computationally-efficient `fft` and `ifft` algorithms to reduce the cost of computing convolutions.

The `cwt` function implements the CWT.

Inverse Continuous Wavelet Transform

The `icwt` function implements the inverse CWT. Using `icwt` requires that you obtain the CWT from `cwt`.

Because the CWT is a redundant transform, there is not a unique way to define the inverse. The inverse CWT implemented in the Wavelet Toolbox uses the analytic Morse wavelet and L1 normalization.

The inverse CWT is classically presented in the double-integral form. Assume you have a wavelet with a Fourier transform that satisfies the admissibility condition:

$$C_\psi = \int_{-\infty}^{\infty} \frac{\left|\hat{\psi}(\omega)\right|^2}{|\omega|} d\omega < \infty$$

For wavelets satisfying the admissibility condition and finite-energy functions, *f(t)*, you can define the inverse CWT as:

$$f(t) = \frac{1}{C_\psi} \int_a \int_b < f(t), \psi_{a,b}(t) > \psi_{a,b}(t)\, db\, \frac{da}{a^2}$$

For analyzing wavelets and functions satisfying the following conditions, a single integral formula for the inverse CWT exists. These conditions are:

- The analyzed function, *f(t)*, is real-valued and the analyzing wavelet has a real-valued Fourier transform.
- The analyzed function, *f(t)*, is real-valued and the Fourier transform of the analyzing wavelet has support only on the set of nonnegative frequencies. This is referred to as an *analytic* wavelet. A function whose Fourier transform only has support on the set of nonnegative frequencies must be complex-valued.

The preceding conditions constrain the set of possible analyzing wavelets. Wavelets supported by `cwt` are analytic. Because the toolbox only supports the analysis of real-valued functions, the real-valued condition on the analyzed function is always satisfied.

To motivate the single integral formula, let ψ_1 and ψ_2 be two *wavelets* that satisfy the following two-*wavelet* admissibility condition:

$$\int \frac{\left|\hat{\psi}_1^*(\omega)\right| \left|\hat{\psi}_2(\omega)\right|}{|\omega|} d\omega < \infty$$

Define the constant:

$$C_{\psi_1,\psi_2} = \int \frac{\hat{\psi}_1^*(\omega)\hat{\psi}_2(\omega)}{|\omega|} d\omega$$

The above constant may be complex-valued. Let *f(t)* and *g(t)* be two finite energy functions. If the two-*wavelet* admissibility condition is satisfied, the following equality holds:

$$C_{\psi_1,\psi_2} < f, g > \; = \int \int < f, \psi_1 > \; < g, \psi_2 >^* db \frac{da}{a}$$

where < , > denotes the inner product, * denotes the complex conjugate, and the dependence of ψ_1 and ψ_2 on scale and position has been suppressed for convenience.

The key to the single integral formula for the inverse CWT is to recognize that the two-*wavelet* admissibility condition can be satisfied even if one of the *wavelets* is not admissible. In other words, it is not necessary that both ψ_1 and ψ_2 be separately admissible. You can also relax the requirements further by allowing one of the *functions* and *wavelets* to be distributions. By first letting *g(t)* be the Dirac delta function (a distribution) and also allowing ψ_2 to be the Dirac delta function, you can derive the single integral formula for the inverse CWT

$$f(t) = 2\ Re\left\{\frac{1}{C_{\psi_1,\delta}}\int_0^\infty < f(t), \psi_1(t) > \frac{da}{a}\right\}$$

where *Re{ }* denotes the real part.

The preceding equation demonstrates that you can reconstruct the signal by summing the scaled CWT coefficients over all scales.

By summing the scaled CWT coefficients from select scales, you obtain an approximation to the original signal. This is useful in situations where your phenomenon of interest is localized in scale.

`icwt` implements a discretized version of the above integral.

Interpreting Continuous Wavelet Coefficients

Because the CWT is a redundant transform and the CWT coefficients depend on the wavelet, it can be challenging to interpret the results.

To help you in interpreting CWT coefficients, it is best to start with a simple signal to analyze and an analyzing wavelet with a simple structure.

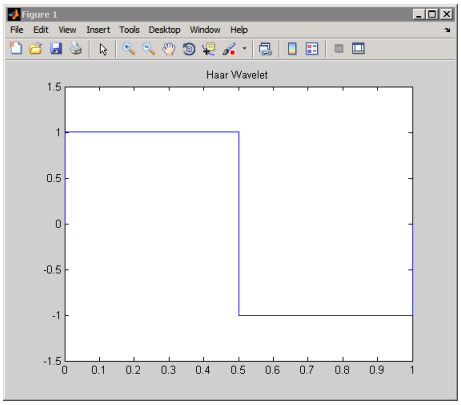
A signal feature that wavelets are very good at detecting is a discontinuity, or singularity. Abrupt transitions in signals result in wavelet coefficients with large absolute values.

For the signal create a shifted impulse. The impulse occurs at point 500.

```
x = zeros(1000,1);
x(500) = 1;
```

For the wavelet, pick the Haar wavelet.

```
[~,psi,xval] = wavefun('haar',10);
plot(xval,psi); axis([0 1 -1.5 1.5]);
title('Haar Wavelet');
```



To compute the CWT using the Haar wavelet at scales 1 to 128, enter:

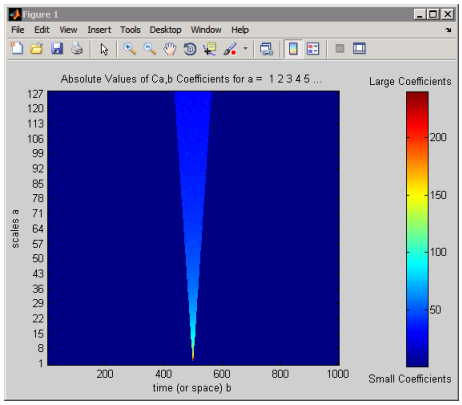
```
CWTcoeffs = cwt(x,1:128,'haar');
```

`CWTcoeffs` is a 128-by-1000 matrix. Each row of the matrix contains the CWT coefficients for one scale. There are 128 rows because the `SCALES` input to `cwt` is `1:128`. The column dimension of the matrix matches the length of the input signal.

Recall that the CWT of a 1D signal is a function of the scale and position parameters. To produce a plot of the CWT coefficients, plot position along the *x*-axis, scale along the *y*-axis, and encode the magnitude, or size of the CWT coefficients as color at each point in the *x*-*y*, or time-scale plane.

You can produce this plot using `cwt` with the optional input argument `'plot'`.

```
cwt(x,1:128,'haar','plot');
colormap jet; colorbar;
```



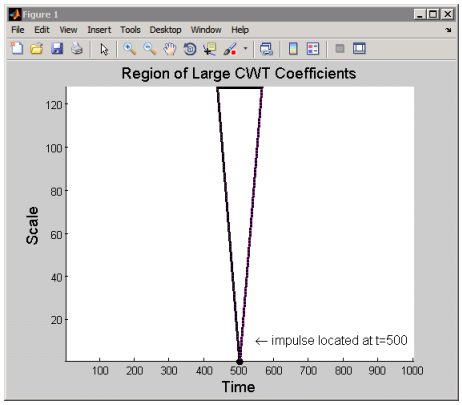
The preceding figure was modified with text labels to explicitly show which colors indicate large and small CWT coefficients.

You can also plot the size of the CWT coefficients in 3D with

```
cwt(x,1:64,'haar','3Dplot'); colormap jet;
```

where the number of scales has been reduced to aid in visualization.

Examining the CWT of the shifted impulse signal, you can see that the set of large CWT coefficients is concentrated in a narrow region in the time-scale plane at small scales centered around point 500. As the scale increases, the set of large CWT coefficients becomes wider, but remains centered around point 500. If you trace the border of this region, it resembles the following figure.



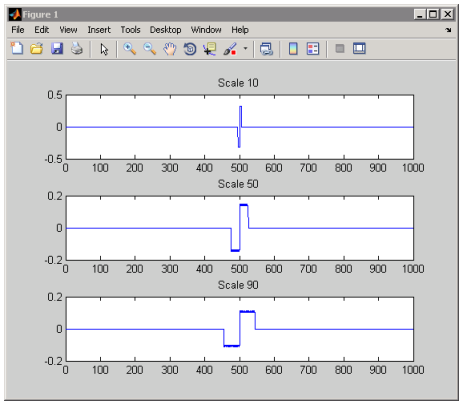
This region is referred to as the *cone of influence* of the point t=500 for the Haar wavelet. For a given point, the cone of influence shows you which CWT coefficients are affected by the signal value at that point.

To understand the cone of influence, assume that you have a wavelet supported on [-C, C]. Shifting the wavelet by b and scaling by a results in a wavelet supported on [-Ca+b, Ca+b]. For the simple case of a shifted impulse, $\delta(t-\tau)$, the CWT coefficients are only nonzero in an interval around τ equal to the support of the wavelet at each scale. You can see this by considering the formal expression of the CWT of the shifted impulse.

$$C(a,b;\delta(t-\tau),\psi(t)) = \int_{\infty}^{\infty}\delta(t-\tau)\frac{1}{\sqrt{a}}\psi^*(\frac{t-b}{a})dt = \frac{1}{\sqrt{a}}\psi^*(\frac{\tau-b}{a})$$

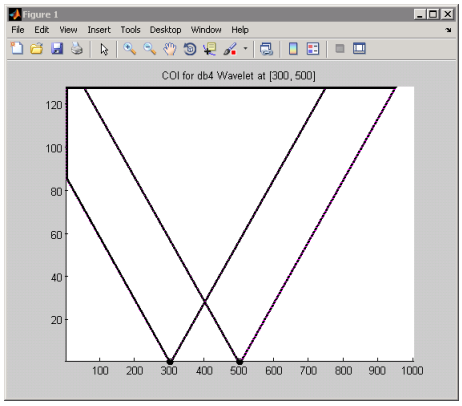
For the impulse, the CWT coefficients are equal to the conjugated, time-reversed, and scaled wavelet as a function of the shift parameter, b. You can see this by plotting the CWT coefficients for a select few scales.

```
subplot(311)
plot(CWTcoeffs(10,:)); title('Scale 10');
subplot(312)
plot(CWTcoeffs(50,:)); title('Scale 50');
subplot(313)
plot(CWTcoeffs(90,:)); title('Scale 90');
```



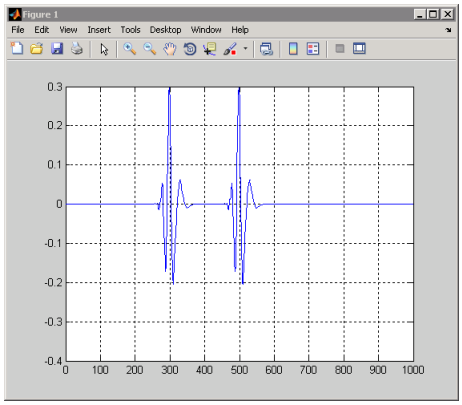
The cone of influence depends on the wavelet. You can find and plot the cone of influence for a specific wavelet with `conofinf`.

The next example features the superposition of two shifted impulses, $\delta(t-300)+\delta(t-500)$. In this case, use the Daubechies' extremal phase wavelet with four vanishing moments, `db4`. The following figure shows the cone of influence for the points 300 and 500 using the `db4` wavelet.



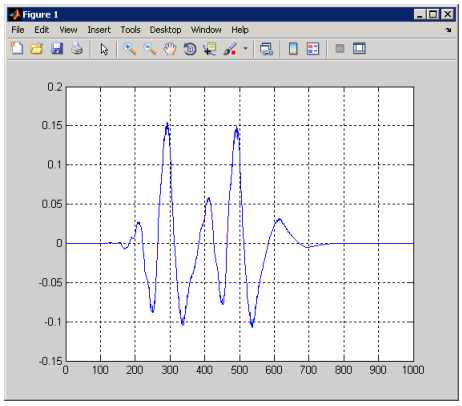
Look at point 400 for scale 20. At that scale, you can see that neither cone of influence overlaps the point 400. Therefore, you can expect that the CWT coefficient will be zero at that point and scale. The signal is only nonzero at two values, 300 and 500, and neither cone of influence for those values includes the point 400 at scale 20. You can confirm this by entering:

```
x = zeros(1000,1);
x([300 500]) = 1;
CWTcoeffs = cwt(x,1:128,'db4');
plot(CWTcoeffs(20,:)); grid on;
```



Next, look at the point 400 at scale 80. At scale 80, the cones of influence for both points 300 and 500 include the point 400. Even though the signal is zero at point 400, you obtain a nonzero CWT coefficient at that scale. The CWT coefficient is nonzero because the support of the wavelet has become sufficiently large at that scale to allow signal values 100 points above and below to affect the CWT coefficient. You can confirm this by entering:

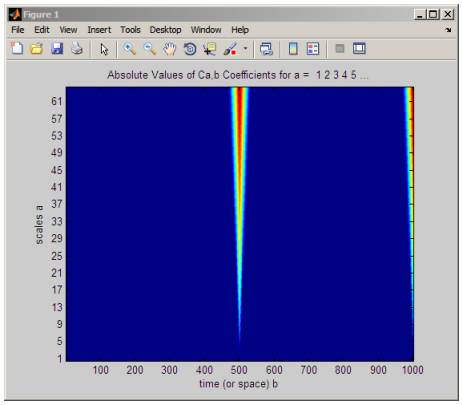
```
plot(CWTcoeffs(80,:));
grid on;
```



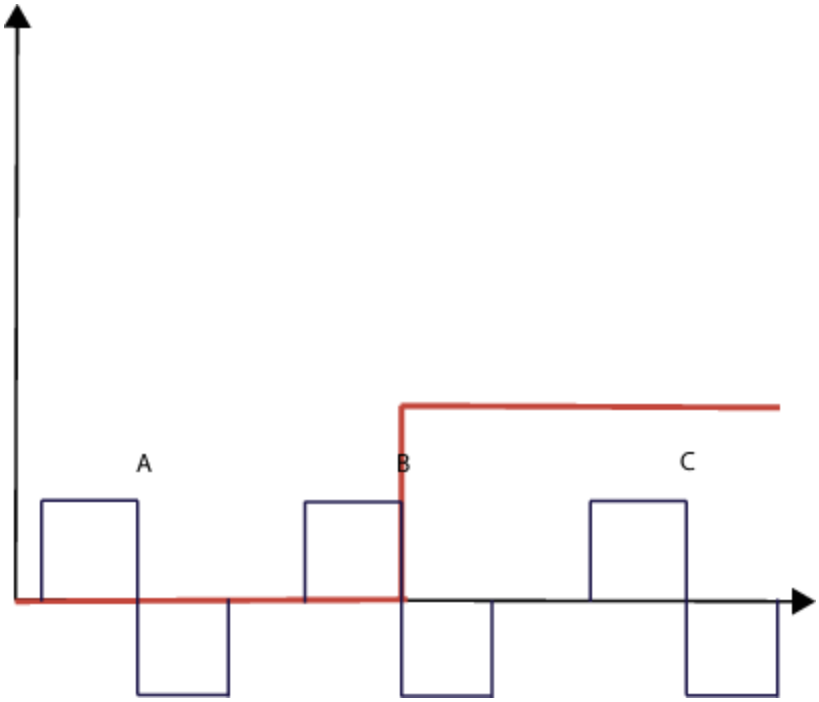
In the preceding example, the CWT coefficients became large in the vicinity of an abrupt change in the signal. This ability to detect discontinuities is a strength of the wavelet transform. The preceding example also demonstrated that the CWT coefficients localize the discontinuity best at small scales. At small scales, the small support of the wavelet ensures that the singularity only affects a small set of wavelet coefficients.

To demonstrate why the wavelet transform is so adept at detecting abrupt changes in the signal, consider a shifted Heaviside, or unit step signal.

```
x = [zeros(500,1); ones(500,1)];
CWTcoeffs = cwt(x,1:64,'haar','plot'); colormap jet;
```



Similar to the shifted impulse example, the abrupt transition in the shifted step function results in large CWT coefficients at the discontinuity. The following figure illustrates why this occurs.



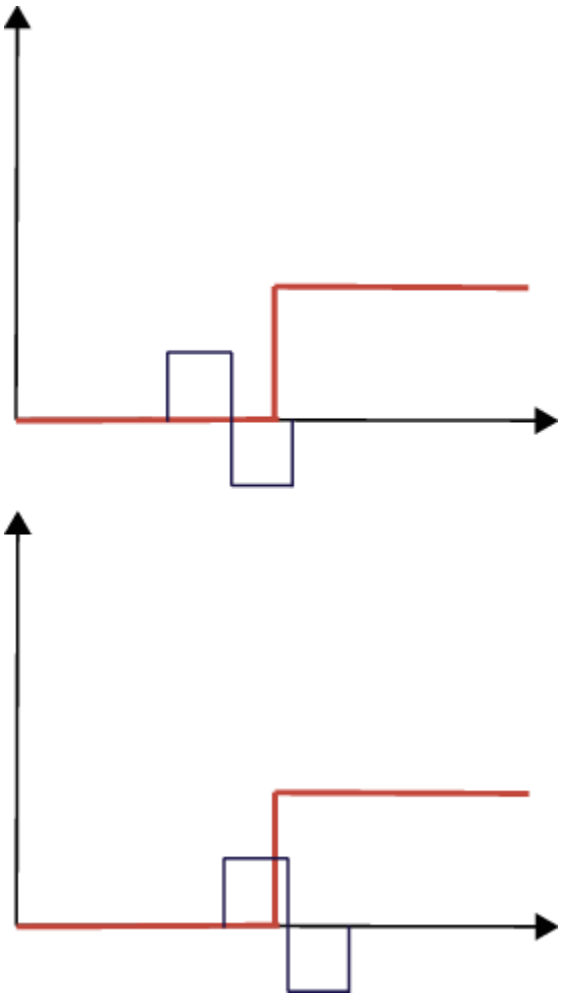
In the preceding figure, the red function is the shifted unit step function. The black functions labeled A, B, and C depict Haar wavelets at the same scale but different positions. You can see that the CWT coefficients around position A are zero. The signal is zero in that neighborhood and therefore the wavelet transform is also zero because any wavelet integrates to zero.

Note the Haar wavelet centered around position B. The negative part of the Haar wavelet overlaps with a region of the step function that is equal to 1. The CWT coefficients are negative because the product of the Haar wavelet and the unit step is a negative constant. Integrating over that area yields a negative number.

Note the Haar wavelet centered around position C. Here the CWT coefficients are zero. The step function is equal to one. The product of the wavelet with the step function is equal to the wavelet. Integrating any wavelet over its support is zero. This is the zero moment property of wavelets.

At position B, the Haar wavelet has already shifted into the nonzero portion of the step function by 1/2 of its support. As soon as the support of the wavelet intersects with the unity portion of the step function, the CWT coefficients are nonzero. In fact, the situation illustrated in the previous figure coincides with the CWT coefficients achieving their largest absolute value. This is because the entire negative deflection of the wavelet oscillation overlaps with the unity portion of the unit step while none of the positive deflection of the wavelet does. Once the wavelet shifts to the point that the positive deflection overlaps with the unit step, there will be some positive contribution to the integral. The wavelet coefficients are still negative (the negative portion of the integral is larger in area), but they are smaller in absolute value than those obtained at position B.

The following figure illustrates two other positions where the wavelet intersects the unity portion of the unit step.

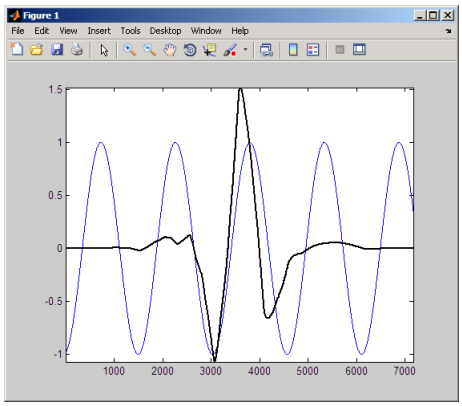


In the top figure, the wavelet has just begun to overlap with the unity portion of the unit step. In this case, the CWT coefficients are negative, but not as large in absolute value as those obtained at position B. In the bottom figure, the wavelet has shifted past position B and the positive deflection of the wavelet begins to contribute to the integral. The CWT coefficients are still negative, but not as large in absolute value as those obtained at position B.

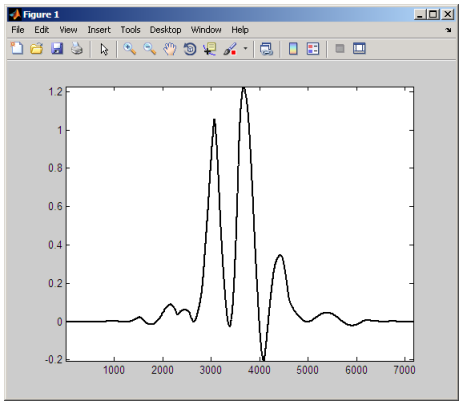
You can now visualize how the wavelet transform is able to detect discontinuities. You can also visualize in this simple example exactly why the CWT coefficients are negative in the CWT of the shifted unit step using the Haar wavelet. Note that this behavior differs for other wavelets.

```
x = [zeros(500,1); ones(500,1)];
CWTcoeffs = cwt(x,1:64,'haar','plot'); colormap jet;
% plot a few scales for visualization
subplot(311);
plot(CWTcoeffs(5,:)); title('Scale 5');
subplot(312);
plot(CWTcoeffs(10,:)); title('Scale 10');
subplot(313);
plot(CWTcoeffs(50,:)); title('Scale 50');
```

Next consider how the CWT represents smooth signals. Because sinusoidal oscillations are a common phenomenon, this section examines how sinusoidal oscillations in the signal affect the CWT coefficients. To begin, consider the `sym4` wavelet at a specific scale superimposed on a sine wave.

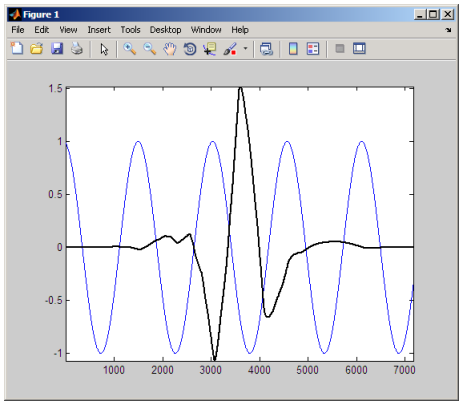


Recall that the CWT coefficients are obtained by computing the product of the signal with the shifted and scaled analyzing wavelet and integrating the result. The following figure shows the product of the wavelet and the sinusoid from the preceding figure.

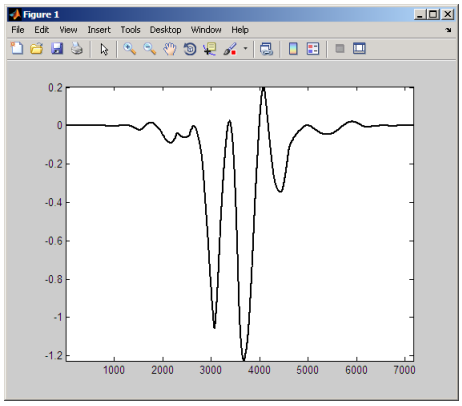


You can see that integrating over this product produces a positive CWT coefficient. That results because the oscillation in the wavelet approximately matches a period of the sine wave. The wavelet is *in phase* with the sine wave. The negative deflections of the wavelet approximately match the negative deflections of the sine wave. The same is true of the positive deflections of both the wavelet and sinusoid.

The following figure shifts the wavelet 1/2 of the period of the sine wave.

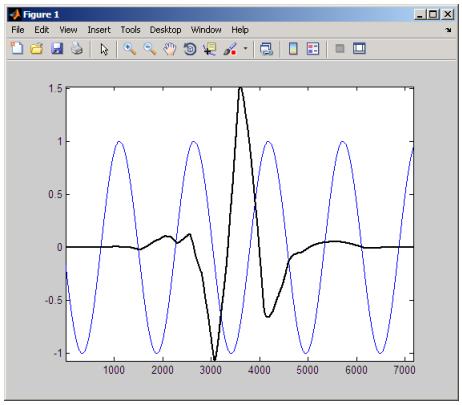


Examine the product of the shifted wavelet and the sinusoid.

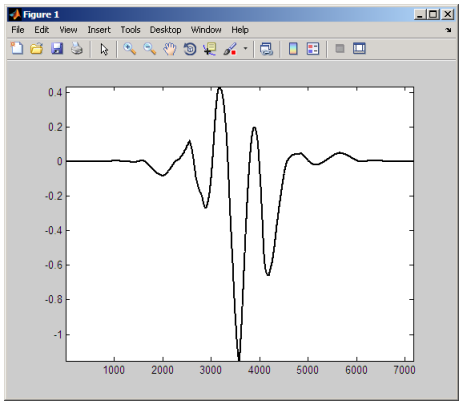


You can see that integrating over this product produces a negative CWT coefficient. That results because the wavelet is 1/2 cycle out of phase with the sine wave. The negative deflections of the wavelet approximately match the positive deflections of the sine wave. The positive deflections of the wavelet approximately match the negative deflections of the sinusoid.

Finally, shift the wavelet approximately one quarter cycle of the sine wave.

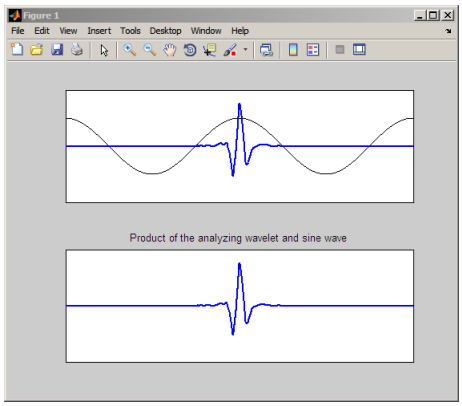


The following figure shows the product of the shifted wavelet and the sinusoid.



Integrating over this product produces a CWT coefficient much smaller in absolute value than either of the two previous examples. That results because the negative deflection of the wavelet approximately aligns with a positive deflection of the sine wave. Also, the main positive deflection of the wavelet approximately aligns with a positive deflection of the sine wave. The resulting product looks much more like a wavelet than the other two products. If it looked exactly like a wavelet, the integral would be zero.

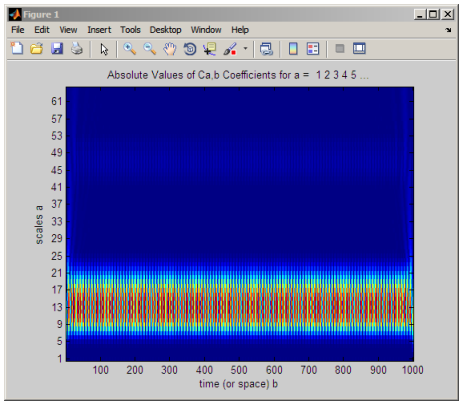
At scales where the oscillation in the wavelet occurs on either a much larger or smaller scale than the period of the sine wave, you obtain CWT coefficients near zero. The following figure illustrates the case where the wavelet oscillates on a much smaller scale than the sinusoid.



The product shown in the bottom pane closely resembles the analyzing wavelet. Integrating this product results in a CWT coefficient near zero.

The following example constructs a 60-Hz sine wave and obtains the CWT using the `sym8` wavelet.

```
t = linspace(0,1,1000);
x = cos(2*pi*60*t);
CWTcoeffs = cwt(x,1:64,'sym8','plot'); colormap jet;
```

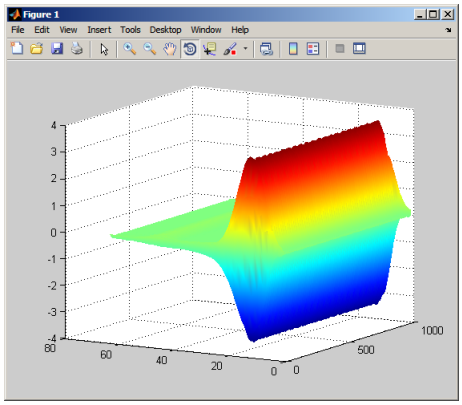


Note that the CWT coefficients are large in absolute value around scales 9 to 21. You can find the pseudo-frequencies corresponding to these scales using the command:

```
freq = scal2frq(9:21,'sym8',1/1000);
```

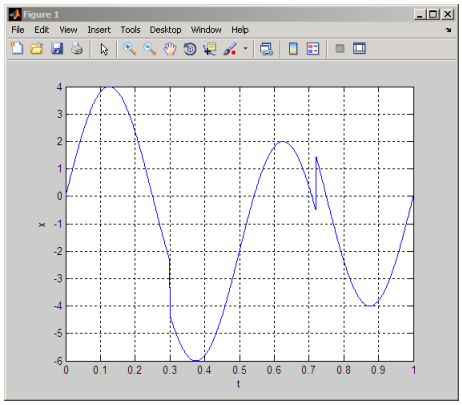
Note that the CWT coefficients are large at scales near the frequency of the sine wave. You can clearly see the sinusoidal pattern in the CWT coefficients at these scales with the following code.

```
surf(CWTcoeffs); colormap jet;
shading('interp'); view(-60,12);
```



The final example constructs a signal consisting of both abrupt transitions and smooth oscillations. The signal is a 2-Hz sinusoid with two introduced discontinuities.

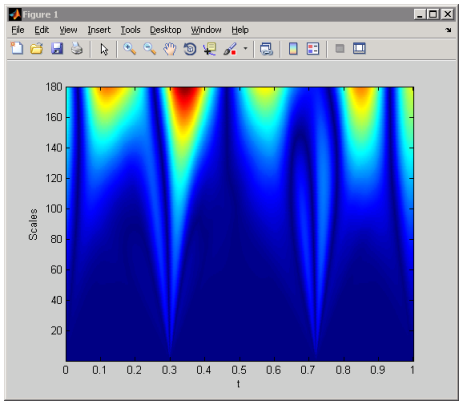
```
N = 1024;
t = linspace(0,1,1024);
x = 4*sin(4*pi*t);
x = x - sign(t - .3) - sign(.72 - t);
plot(t,x); xlabel('t'); ylabel('x');
grid on;
```



Note the discontinuities near t=0.3 and t=0.7.

Obtain and plot the CWT using the `sym4` wavelet.

```
CWTcoeffs = cwt(x,1:180,'sym4');
imagesc(t,1:180,abs(CWTcoeffs));
colormap jet; axis xy;
xlabel('t'); ylabel('Scales');
```



Note that the CWT detects both the abrupt transitions and oscillations in the signal. The abrupt transitions affect the CWT coefficients at all scales and clearly separate themselves from smoother signal features at small scales. On the other hand, the maxima and minima of the 2–Hz sinusoid are evident in the CWT coefficients at large scales and not apparent at small scales.

The following general principles are important to keep in mind when interpreting CWT coefficients.

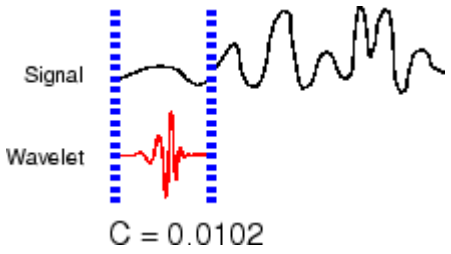
- **Cone of influence**— Depending on the scale, the CWT coefficient at a point can be affected by signal values at points far removed. You have to take into account the support of the wavelet at specific scales. Use `conofinf` to determine the cone of influence. Not all wavelets are equal in their support. For example, the Haar wavelet has smaller support at all scales than the `sym4` wavelet.
- **Detecting abrupt transitions**— Wavelets are very useful for detecting abrupt changes in a signal. Abrupt changes in a signal produce relatively large wavelet coefficients (in absolute value) centered around the discontinuity at all scales. Because of the support of the wavelet, the set of CWT coefficients affected by the singularity increases with increasing scale. Recall this is the definition of the cone of influence. The most precise localization of the discontinuity based on the CWT coefficients is obtained at the smallest scales.

- **Detecting smooth signal features**— Smooth signal features produce relatively large wavelet coefficients at scales where the oscillation in the wavelet correlates best with the signal feature. For sinusoidal oscillations, the CWT coefficients display an oscillatory pattern at scales where the oscillation in the wavelet approximates the period of the sine wave.

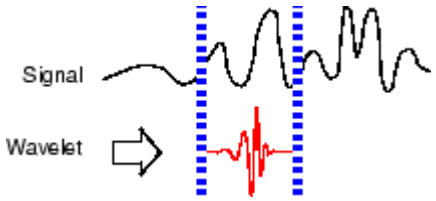
The basic algorithm for the continuous wavelet transform (CWT) is:

1. Take a wavelet and compare it to a section at the start of the original signal.
2. Calculate a number, `C`, that represents how closely correlated the wavelet is with this section of the signal. The larger the number `C` is in absolute value, the more the similarity. This follows from the fact the CWT coefficients are calculated with an inner product. See “Inner Products” on page 1-58 for more information on how inner products measure similarity. If the signal energy and the wavelet energy are equal to one, `C` may be interpreted as a correlation coefficient. Note that, in general, the signal energy does not equal one and the CWT coefficients are not directly interpretable as correlation coefficients.

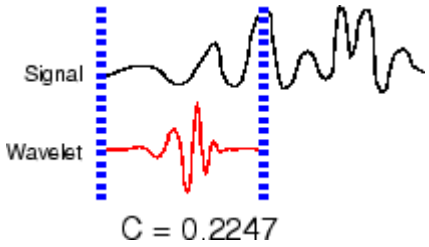
   As described in “Continuous and Discrete Wavelet Transforms” on page 1-38, the CWT coefficients explicitly depend on the analyzing wavelet. Therefore, the CWT coefficients are different when you compute the CWT for the same signal using different wavelets.



3. Shift the wavelet to the right and repeat steps 1 and 2 until you've covered the whole signal.



4. Scale (stretch) the wavelet and repeat steps 1 through 3.



5. Repeat steps 1 through 4 for all scales.

Critically-Sampled Discrete Wavelet Transform

Calculating wavelet coefficients at every possible scale is a fair amount of work, and it generates an awful lot of data. What if we choose only a subset of scales and positions at which to make our calculations?

It turns out, rather remarkably, that if we choose scales and positions based on powers of two — so-called *dyadic* scales and positions — then our analysis will be much more efficient and just as accurate. We obtain such an analysis from the *discrete wavelet transform* (DWT). For more information on DWT, see “Algorithms” in the *Wavelet Toolbox User's Guide*.

An efficient way to implement this scheme using filters was developed in 1988 by Mallat (see [Mal89] in “References” on page 1-105). The Mallat algorithm is in fact a classical scheme known in the signal processing community as a *two-channel subband coder* (see page 1 of the book *Wavelets and Filter Banks, by Strang and Nguyen* [StrN96]*).*

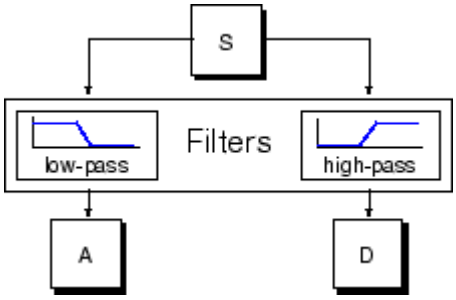
This very practical filtering algorithm yields a *fast wavelet transform* — a box into which a signal passes, and out of which wavelet coefficients quickly emerge. Let's examine this in more depth.

One-Stage Filtering: Approximations and Details

For many signals, the low-frequency content is the most important part. It is what gives the signal its identity. The high-frequency content, on the other hand, imparts flavor or nuance. Consider the human voice. If you remove the high-frequency components, the voice sounds different, but you can still tell what's being said. However, if you remove enough of the low-frequency components, you hear gibberish.

In wavelet analysis, we often speak of *approximations* and *details*. The approximations are the high-scale, low-frequency components of the signal. The details are the low-scale, high-frequency components.

The filtering process, at its most basic level, looks like this.

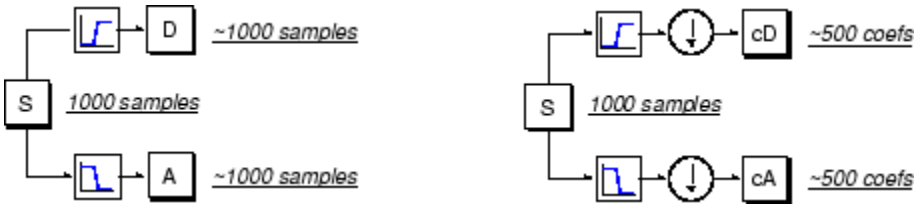


The original signal, S, passes through two complementary filters and emerges as two signals.

Unfortunately, if we actually perform this operation on a real digital signal, we wind up with twice as much data as we started with. Suppose, for instance, that the original signal S consists of 1000 samples of data. Then the resulting signals will each have 1000 samples, for a total of 2000.

These signals A and D are interesting, but we get 2000 values instead of the 1000 we had. There exists a more subtle way to perform the decomposition using wavelets. By looking carefully at the computation, we may keep only one point out of two in each of the two 2000-length samples to get

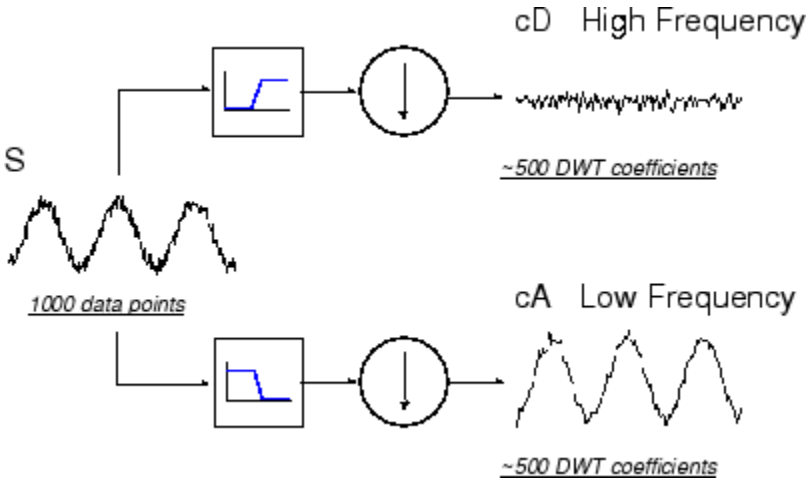
the complete information. This is the notion of *downsampling*. We produce two sequences called `cA` and `cD`.



The process on the right, which includes downsampling, produces DWT coefficients.

To gain a better appreciation of this process, let's perform a one-stage discrete wavelet transform of a signal. Our signal will be a pure sinusoid with high-frequency noise added to it.

Here is our schematic diagram with real signals inserted into it.



The MATLAB code needed to generate `s`, `cD`, and `cA` is

```
s
= sin(20.*linspace(0,pi,1000)) + 0.5.*rand(1,1000);
[cA,cD] = dwt(s,'db2');
```

where `db2` is the name of the wavelet we want to use for the analysis.

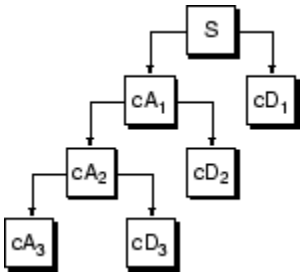
Notice that the detail coefficients `cD` are small and consist mainly of a high-frequency noise, while the approximation coefficients `cA` contain much less noise than does the original signal.

```
[length(cA) length(cD)]

ans =
   501  501
```

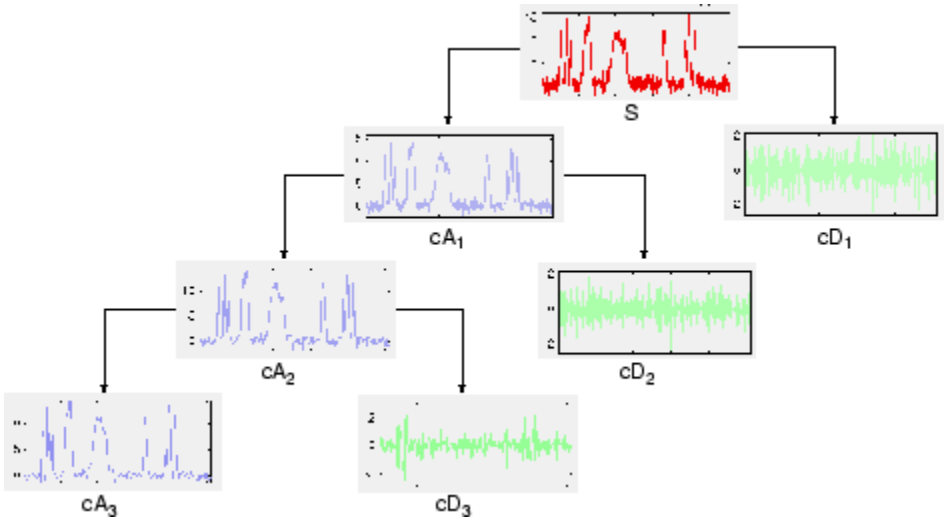
You may observe that the actual lengths of the detail and approximation coefficient vectors are slightly *more* than half the length of the original signal. This has to do with the filtering process, which is implemented by convolving the signal with a filter. The convolution “smears” the signal, introducing several extra samples into the result.

Multiple-Level Decomposition

The decomposition process can be iterated, with successive approximations being decomposed in turn, so that one signal is broken down into many lower resolution components. This is called the *wavelet decomposition tree*.



Looking at a signal's wavelet decomposition tree can yield valuable information.



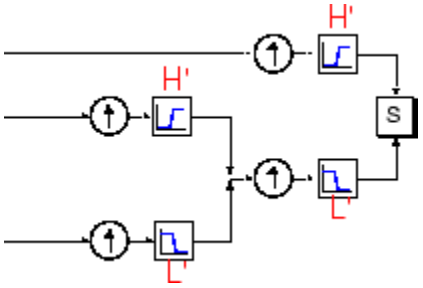
Number of Levels

Since the analysis process is iterative, in theory it can be continued indefinitely. In reality, the decomposition can proceed only until the individual details consist of a single sample or pixel. In practice, you'll select a suitable number of levels based on the nature of the signal, or on a suitable criterion such as *entropy* (see “Choosing the Optimal Decomposition” in the *Wavelet Toolbox User's Guide*).

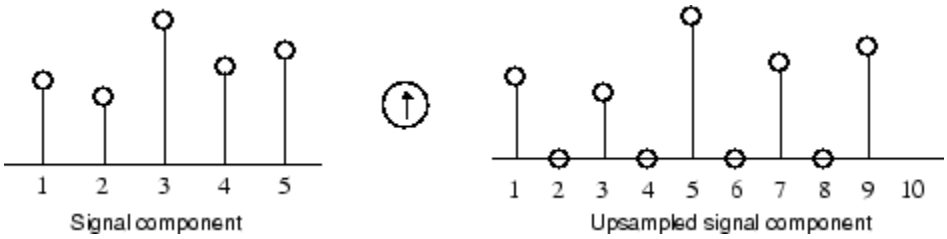
Critically-Sampled Wavelet Reconstruction

We've learned how the discrete wavelet transform can be used to analyze, or decompose, signals and images. This process is called *decomposition* or *analysis*. The other half of the story is how those components can be assembled back into the original signal without loss of information. This process is called *reconstruction*, or *synthesis*. The mathematical manipulation that effects synthesis is called the *inverse discrete wavelet transform* (IDWT).

To synthesize a signal using Wavelet Toolbox software, we reconstruct it from the wavelet coefficients.



Where wavelet analysis involves filtering and downsampling, the wavelet reconstruction process consists of upsampling and filtering. Upsampling is the process of lengthening a signal component by inserting zeros between samples.



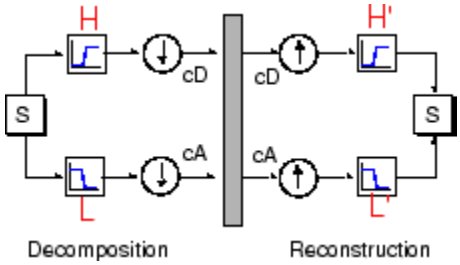
The toolbox includes commands, like `idwt` and `waverec`, that perform single-level or multilevel reconstruction, respectively, on the components of 1-D signals. These commands have their 2-D and 3-D analogs, `idwt2`, `waverec2`, `idwt3`, and `waverec3`.

Reconstruction Filters

The filtering part of the reconstruction process also bears some discussion, because it is the choice of filters that is crucial in achieving perfect reconstruction of the original signal.

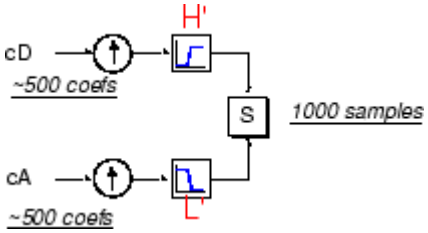
The downsampling of the signal components performed during the decomposition phase introduces a distortion called aliasing. It turns out that by carefully choosing filters for the decomposition and reconstruction phases that are closely related (but not identical), we can “cancel out” the effects of aliasing.

A technical discussion of how to design these filters is available on page 347 of the book *Wavelets and Filter Banks*, by Strang and Nguyen. The low- and high-pass decomposition filters (`L` and `H`), together with their associated reconstruction filters (`L'` and `H'`), form a system of what is called *quadrature mirror filters:*



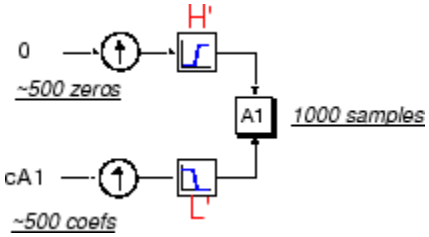
Reconstructing Approximations and Details

We have seen that it is possible to reconstruct our original signal from the coefficients of the approximations and details.



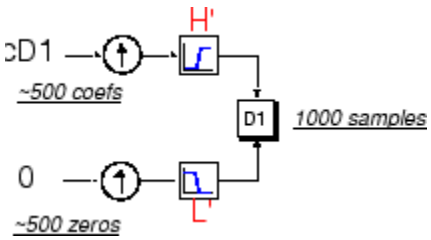
It is also possible to reconstruct the approximations and details themselves from their coefficient vectors. As an example, let's consider how we would reconstruct the first-level approximation `A1` from the coefficient vector `cA1`.

We pass the coefficient vector `cA1` through the same process we used to reconstruct the original signal. However, instead of combining it with the level-one detail `cD1`, we feed in a vector of zeros in place of the detail coefficients vector:



The process yields a reconstructed *approximation* `A1`, which has the same length as the original signal `S` and which is a real approximation of it.

Similarly, we can reconstruct the first-level detail `D1`, using the analogous process:

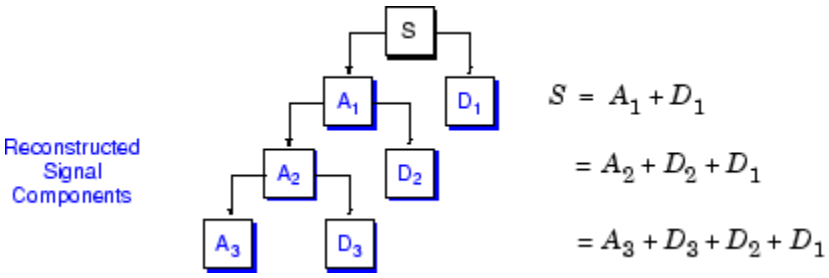


The reconstructed details and approximations are true constituents of the original signal. In fact, we find when we combine them that

$$A_1 + D_1 = S.$$

Note that the coefficient vectors `cA1` and `cD1` — because they were produced by downsampling and are only half the length of the original signal — cannot directly be combined to reproduce the signal. *It is necessary to reconstruct the approximations and details before combining them.*

Extending this technique to the components of a multilevel analysis, we find that similar relationships hold for all the reconstructed signal constituents. That is, there are several ways to reassemble the original signal:



Wavelets From Conjugate Mirror Filters

In the section “Reconstruction Filters” on page 1-88, we spoke of the importance of choosing the right filters. In fact, the choice of filters not only determines whether perfect reconstruction is possible, it also determines the shape of the wavelet we use to perform the analysis.

To construct a wavelet of some practical utility, you seldom start by drawing a waveform. Instead, it usually makes more sense to design the appropriate quadrature mirror filters, and then use them to create the waveform. Let's see how this is done by focusing on an example.

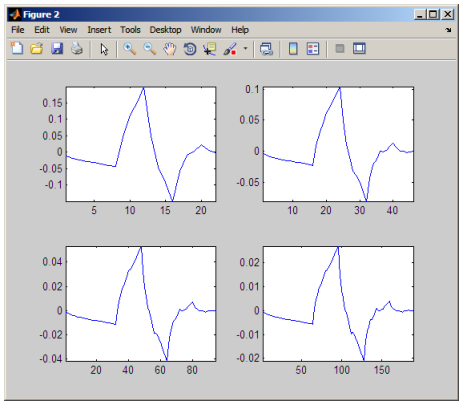
Consider the low-pass reconstruction filter (`L'`) for the `db2` wavelet.

The filter coefficients can be obtained from the `dbaux` function. By reversing the order of the scaling filter vector and multiplying every even element (indexing from 1) by (-1), you obtain the high-pass filter.

Repeatedly upsampling by two and convolving the output with the scaling filter produces the Daubechies' extremal phase wavelet.

```
L = dbaux(2);
H = wrev(L).*[1 -1 1 -1];
HU = dyadup(H,0);
HU = conv(HU,L);
plot(HU); title('1st Iteration');
H1 = conv(dyadup(HU,0),L);
H2 = conv(dyadup(H1,0),L);
H3 = conv(dyadup(H2,0),L);
H4 = conv(dyadup(H3,0),L);
figure;
for k =1:4
subplot(2,2,k);
eval(['plot(H' num2str(k) ')']);
```

```
axis tight;
end
```



The curve begins to look progressively more like the db2 wavelet. This means that the wavelet's shape is determined entirely by the coefficients of the reconstruction filters.

This relationship has profound implications. It means that you cannot choose just any shape, call it a wavelet, and perform an analysis. At least, you can't choose an arbitrary wavelet waveform if you want to be able to reconstruct the original signal accurately. You are compelled to choose a shape determined by quadrature mirror decomposition filters.

Scaling Function

We've seen the interrelation of wavelets and quadrature mirror filters. The wavelet function ψ is determined by the high-pass filter, which also produces the details of the wavelet decomposition.

There is an additional function associated with some, but not all, wavelets. This is the so-called *scaling function*, ϕ. The scaling function is very similar to the wavelet function. It is determined by the low-pass quadrature mirror filters, and thus is associated with the approximations of the wavelet decomposition.

In the same way that iteratively upsampling and convolving the high-pass filter produces a shape approximating the wavelet function, iteratively upsampling and convolving the low-pass filter produces a shape approximating the scaling function.

Wavelet Synchrosqueezing

What is Wavelet Synchrosqueezing?

The wavelet synchrosqueezed transform is a time-frequency analysis method that is useful for analyzing multicomponent signals with oscillating modes. Examples of signals with oscillating modes include speech waveforms, machine vibrations, and physiologic signals. Many of these real-world signals with oscillating modes can be written as a sum of amplitude-modulated and frequency-modulated components. A general expression for these types of signals with summed components is

$$\sum_{k=1}^{K} A_k(t)\cos(2\pi\phi_k(t)),$$

where $A_k(t)$ is the slowly varying amplitude and $\phi_k(t)$ is the instantaneous phase. A truncated Fourier series, where the amplitude and frequency do not vary with time, is a special case of these signals.

The wavelet transform and other linear time-frequency analysis methods decompose these signals into their components by correlating the signal with a dictionary of time-frequency atoms [1]. The wavelet transform uses translated and scaled versions of a mother wavelet as its time-frequency atom. Some time-frequency spreading is associated with all of these time-frequency atoms, which affects the sharpness of the signal analysis.

The wavelet synchrosqueezed transform is a time-frequency method that reassigns the signal energy in frequency. This reassignment compensates for the spreading effects caused by the mother wavelet. Unlike other time-frequency reassignment methods, synchrosqueezing reassigns the energy only in the frequency direction, which preserves the time resolution of the signal. By preserving the time, the inverse synchrosqueezing algorithm can reconstruct an accurate representation of the original signal. To use synchrosqueezing, each term in the summed components signal expression must be an intrinsic mode type (IMT) function. For details on the criteria that constitute IMTs, see [2].

Algorithm

The synchrosqueezing algorithm uses these steps.

1. Obtain the CWT of the input signal. For use with synchrosqueezing, the CWT must use an analytic wavelet to capture instantaneous frequency information.
2. Extract the instantaneous frequencies from the CWT output, W_f, using a phase transform, ω_f. This phase transform is proportional to the first derivative of the CWT with respect to the translation, u. In this definition of the phase transform, s is the scale.

   $$\omega_f(s,u) = \frac{\partial t W_f(s,u)}{2\pi i W_f(s,u)}.$$

   The scales are defined as $s = \frac{f_\chi}{f}$, where f_χ is the peak frequency and f is the frequency.

   To extract the instantaneous frequency, consider a simple sine wave, $e^{i2\pi f_0 t}$.

   a. Obtain the wavelet transform,

   $$W_f\left(e^{i2\pi f_0 t}\right) = e^{i2\pi f_0 u},$$

where $\widehat{\chi}(sf_0)$ is the Fourier transform of the wavelet at sf_0.

b Take the partial derivative of the previous equation with respect to the translation, u:

$$\frac{\partial}{\partial u}W_f\left(e^{i2\pi f_0 t}\right) = i2\pi f_0 \widehat{\chi}\left(f_\chi\right)e^{i2\pi f_0 u}$$

c Divide the partial derivative by the wavelet transform and $i2\pi$ to obtain the instantaneous frequency, f_0.

3 "Squeeze" the CWT over regions where the phase transform is constant. The resulting instantaneous frequency value is reassigned to a single value at the centroid of the CWT time-frequency region. This reassignment results in sharpened output from the synchrosqueezed transform when compared to the CWT.

As described, synchrosqueezing uses the continuous wavelet transform (CWT) and its first derivative with respect to translation. The CWT is invertible and since the synchrosqueezed transform inherits the CWT invertibility property, the signal can be reconstructed.

Required Component Separation

With synchrosqueezing the signal components must be IMTs that are well separated in the time-frequency plane. If this requirement is met, you can track the trajectory of the instantaneous frequencies along a curve. The curves show the location of the maximum energy as it varies over time for each signal mode. See `wsstridge` for a description of the trajectory curves algorithm.

This inequality defines the required separation criteria:

$$|\phi'_k(t) - \phi'_{k-1}(t)| \geq \frac{1}{4}|\phi'_k(t) + \phi'_{k-1}(t)|$$

where ϕ' is the instantaneous frequency and d is a positive separation constant [2]. To determine this required separation, suppose a bump wavelet, x, has a Fourier transform with support in the range $[\varepsilon_x - \Delta, \varepsilon_x + \Delta]$. Because the bump wavelet has a center frequency of $\frac{5}{2\pi}$ Hz, use $\left[\frac{5}{2}\pi - \frac{1}{2}, \frac{5}{2}\pi + \frac{1}{2}\right]$ as the interval. Then solve $\Delta < \varepsilon_x\frac{d}{1+d}$ for d to get $d > \frac{1}{4}$ for the bump wavelet.

To show this separation requirement for the bump wavelet, consider a signal composed of $\cos(2\pi(0.1t)) + \sin((2\pi(0.2t))$. Using the bump wavelet to obtain the CWT, the instantaneous phase of the cosine is $\phi_1(t) = 0.1t$, and the instantaneous frequency is the first derivative, 0.1. Likewise, for the sine component, the instantaneous frequency is 0.2. The separation inequality, $|0.1| \geq \frac{1}{4}|0.3|$, is true. Therefore, the two signal components are IMT functions and are separated enough to use the synchrosqueezed transform.

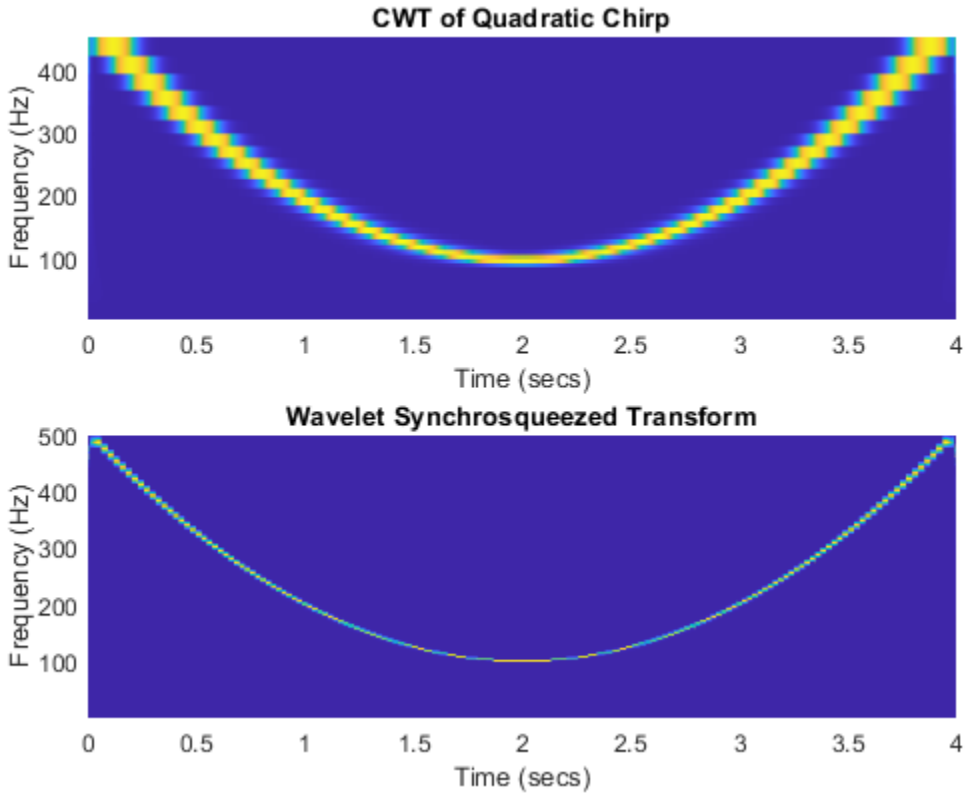
If you use higher frequencies, such as 0.3 and 0.4 for the instantaneous frequencies, the inequality is $|0.1| \geq \frac{1}{4}|0.7|$, which is not true. Because these signal components are not well-separated IMTs the signal, $\cos(2\pi(0.3t)) + \sin((2\pi(0.4t))$, is not appropriate for use with the synchrosqueezed transform.

Examples

CWT vs Synchrosqueezed Transform Smearing

Comparing the CWT with the synchrosqueezed transform of a quadratic chirp shows reduced energy smearing for the synchrosqueezed transform result.

```
load quadchirp;
Fs = 1000;
[wt,f] = cwt(quadchirp,'bump',Fs);
subplot(2,1,1)
hp = pcolor(tquad,f,abs(wt));
hp.EdgeColor = 'none';
xlabel('Time (secs)')
ylabel('Frequency (Hz)')
title('CWT of Quadratic Chirp')
subplot(2,1,2)
wsst(quadchirp,Fs,'bump')
```

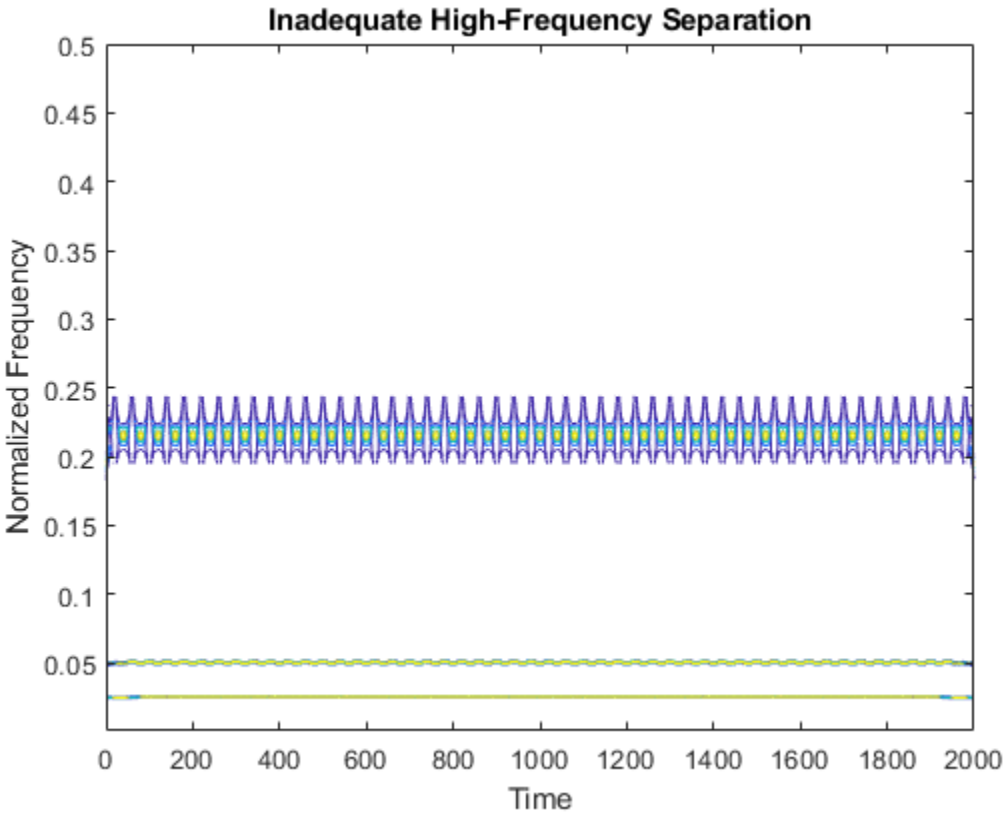


Low-Frequency vs. High-Frequency Component Separation

This example shows the separation needed between signal components to obtain usable results from the synchrosqueezed transform. The signal components are 0.025, 0.05, 0.20, and 0.225 cycles per sample. The high-frequency components, 0.20 and 0.225, do not have enough separation, so you cannot express the whole signal as a sum of well-separated IMTs.

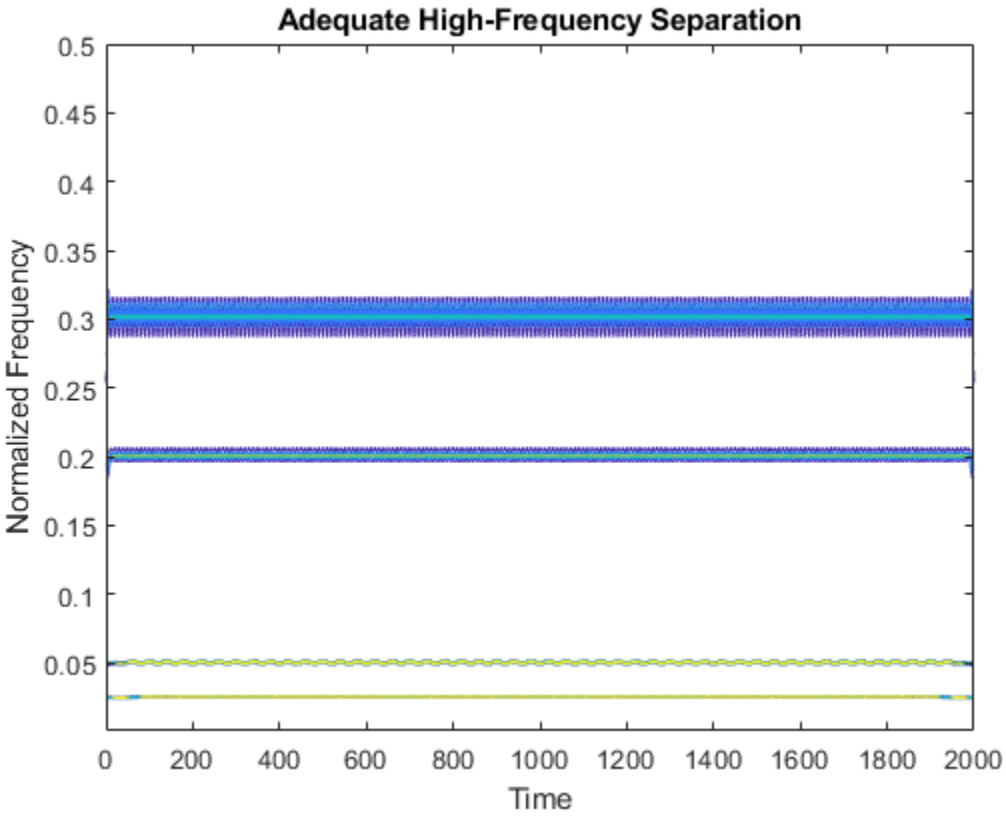
Define the signal and plot the synchrosqueezed components.

```
t = 0:2000;
x1 = cos(2*pi*.025*t);
x2 = cos(2*pi*.05*t);
x3 = cos(2*pi*.20*t);
x4 = cos(2*pi*.225*t);
x =x1+x2+x3+x4;
[sst,f] = wsst(x);
contour(t,f,abs(sst))
xlabel('Time')
ylabel('Normalized Frequency')
title('Inadequate High-Frequency Separation')
```



Increase the separation of the high-frequency components, and then plot the synchrosqueezed components again.

```
x4 = cos(2*pi*.3*t);
x =x1+x2+x3+x4;
[sst,f] = wsst(x);
figure
contour(t,f,abs(sst))
xlabel('Time')
ylabel('Normalized Frequency')
title('Adequate High-Frequency Separation')
```



All the signal components are now well-separated IMTs and are separated enough to distinguish from each other. This signal is appropriate for use with the synchrosqueezing algorithm.

Region With Inadequate Separation

This example shows a signal with two linear chirps. A linear chirp is defined as

$$f(t) = \cos\left(\phi + 2\pi\left(f_0 t + \frac{mt^2}{2}\right)\right).$$

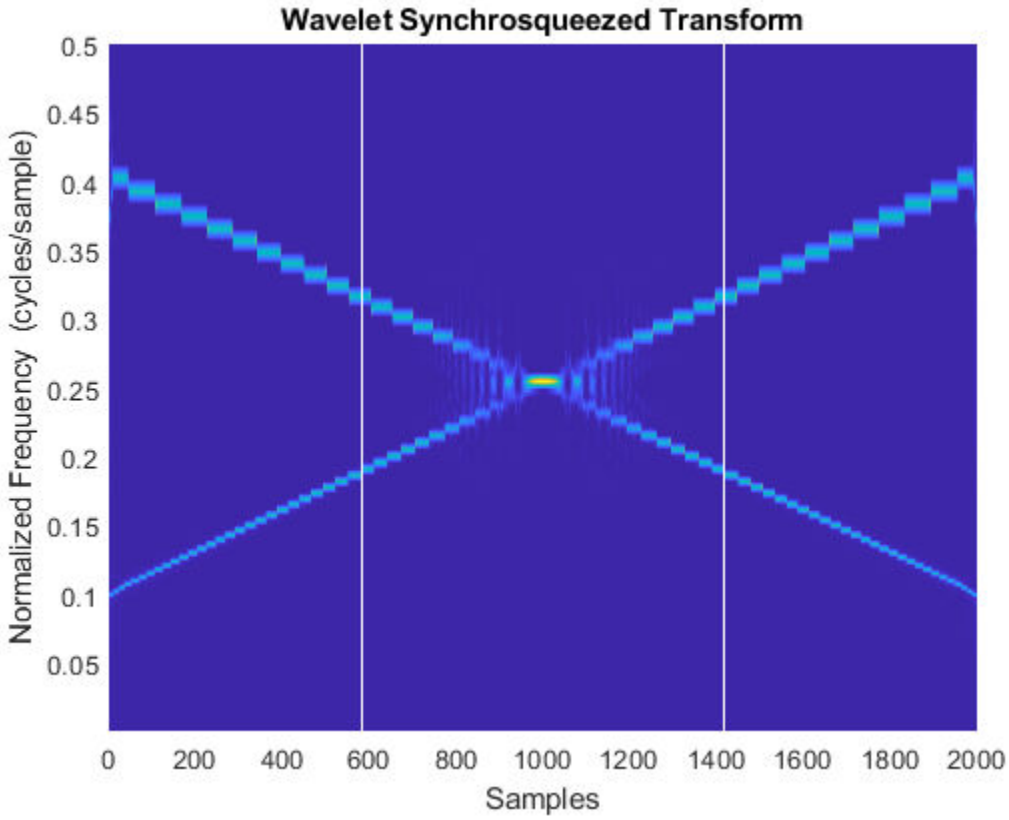
Its first derivative, $f_0 + mt$, defines the instantaneous frequency line. Use the bump wavelet and its separation constant of 0.25. To determine the region where the two chirp signals with instantaneous frequencies of 0.4 and 0.1 cycles per sample are not separated enough, solve this equation:

$$|y_1 - y_2| = 0.25|y_1 + y_2|.$$

$y_1 = \frac{-0.15}{1000}x + 0.4$ and $y_2 = \frac{0.15}{1000}x + 0.1$ are the instantaneous frequency lines of the chirps.

```
t = 0:2000;
y1 = chirp(t,0.4,1000,0.25);
y2 = chirp(t,0.1,1000,0.25);
y = y1+y2;
wsst(y,'bump')
xlabel('Samples')
```

```
h1 = line([583 583], [0 0.5]);
h2 = line([1417 1417], [0 0.5]);
h1.Color='white';
h2.Color='white';
```



The vertical lines are the bounds of the region. They indicate that not enough separation occurs at sample 583 and sample 1417. In the region between the vertical lines, the signal does not consist of well-separated IMTs. In the regions outside the vertical lines, the signal has well-separated IMTs. You can obtain good results from the synchrosqueezed transform in these regions.

References

[1] Mallet, S. *A Wavelet Tour of Signal Processing*. San Diego, CA: Academic Press, 2008, p. 89.

[2] Daubechies, I., J. Lu, and H. T. Wu. "Synchrosqueezed Wavelet Transforms: an empirical mode decomposition-like tool." *Applied and Computational Harmonic Analysis*. Vol. 30(2), pp. 243–261.

[3] Thakur, G., E. Brevdo, N. S. Fučkar, and H. T. Wu. "The synchrosqueezing algorithm for time-varying spectral analysis: robustness properties and new paleoclimate applications." *Signal Processing*. Vol. 93, pp. 1079–1094.

See Also

Related Examples

- “Time-Frequency Reassignment and Mode Extraction with Synchrosqueezing”

Introduction to Wavelet Families

Several families of wavelets that have proven to be especially useful are included in this toolbox. What follows is an introduction to some wavelet families.

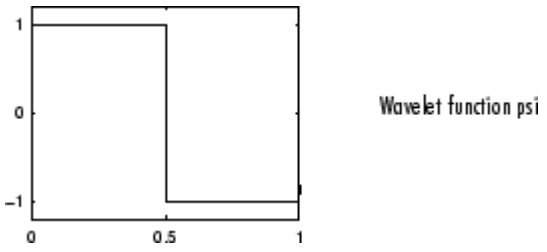
- “Haar” on page 1-99
- “Daubechies” on page 1-99
- “Biorthogonal” on page 1-100
- “Coiflets” on page 1-101
- “Symlets” on page 1-102
- “Morlet” on page 1-102
- “Mexican Hat” on page 1-103
- “Meyer” on page 1-103
- “Other Real Wavelets” on page 1-103
- “Complex Wavelets” on page 1-104

To explore all wavelet families on your own, check out the **Wavelet Display** tool:

1. Type `wvdtool` at the MATLAB command line. The **Wavelet Display** tool appears.
2. Select a family from the **Wavelet** drop-down list at the top right of the tool.
3. Click the **Display** button. Pictures of the wavelets and their associated filters appear.
4. Obtain more information by clicking the information buttons located at the right.

Haar

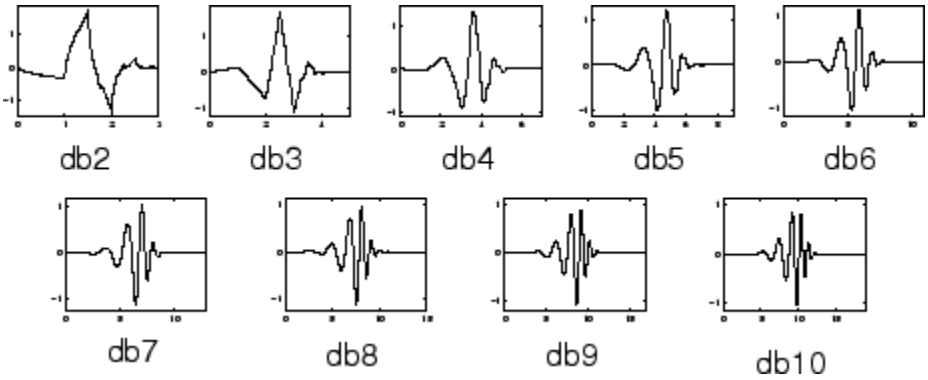
Any discussion of wavelets begins with Haar wavelet, the first and simplest. The Haar wavelet is discontinuous, and resembles a step function. It represents the same wavelet as Daubechies `db1`.



Daubechies

Ingrid Daubechies, one of the brightest stars in the world of wavelet research, invented what are called compactly supported orthonormal wavelets — thus making discrete wavelet analysis practicable.

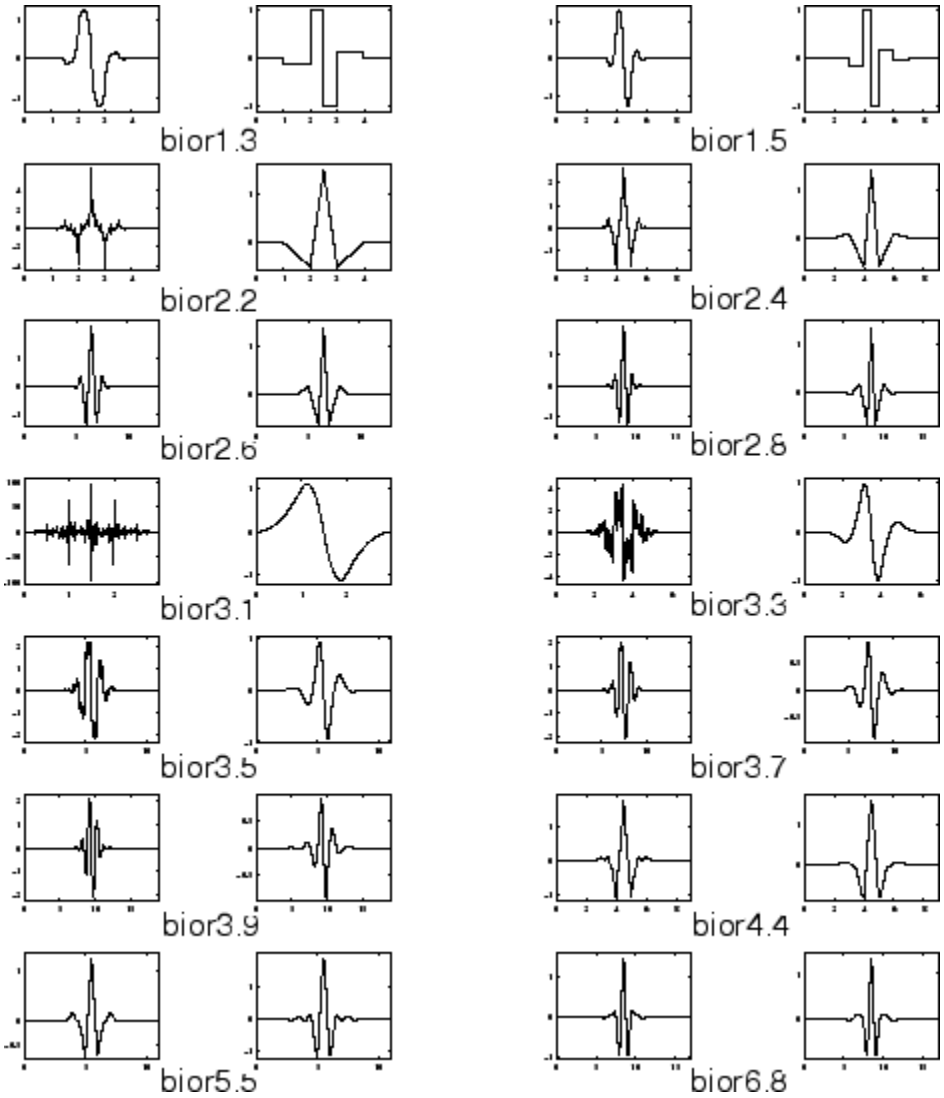
The names of the Daubechies family wavelets are written `dbN`, where `N` is the order, and `db` the “surname” of the wavelet. The `db1` wavelet, as mentioned above, is the same as `Haar` wavelet. Here are the wavelet functions psi of the next nine members of the family:



You can obtain a survey of the main properties of this family by typing `waveinfo('db')` from the MATLAB command line. See “Daubechies Wavelets: dbN” in the *Wavelet Toolbox User's Guide* for more detail.

Biorthogonal

This family of wavelets exhibits the property of linear phase, which is needed for signal and image reconstruction. By using two wavelets, one for decomposition (on the left side) and the other for reconstruction (on the right side) instead of the same single one, interesting properties are derived.



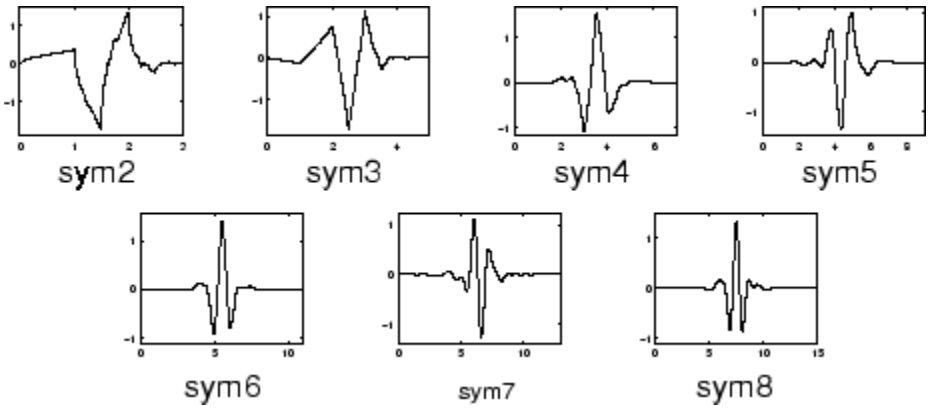
You can obtain a survey of the main properties of this family by typing `waveinfo('bior')` from the MATLAB command line. See “Biorthogonal Wavelet Pairs: biorNr.Nd” in the *Wavelet Toolbox User's Guide* for more detail.

Coiflets

Built by I. Daubechies at the request of R. Coifman. The wavelet function has 2*N* moments equal to 0 and the scaling function has 2*N*-1 moments equal to 0. The two functions have a support of length 6*N*-1. You can obtain a survey of the main properties of this family by typing `waveinfo('coif')` from the MATLAB command line. See “Coiflet Wavelets: coifN” in the *Wavelet Toolbox User's Guide* for more detail.

Symlets

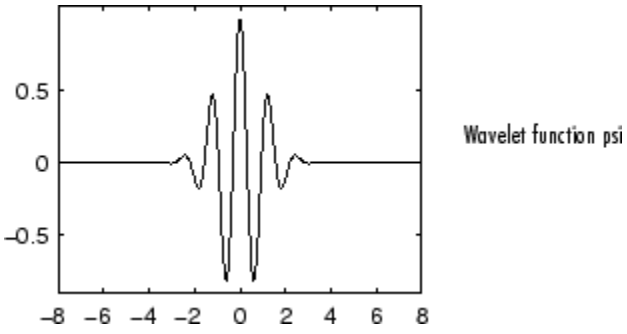
The symlets are nearly symmetrical wavelets proposed by Daubechies as modifications to the `db` family. The properties of the two wavelet families are similar. Here are the wavelet functions psi.



You can obtain a survey of the main properties of this family by typing `waveinfo('sym')` from the MATLAB command line. See “Symlet Wavelets: symN” in the *Wavelet Toolbox User's Guide* for more detail.

Morlet

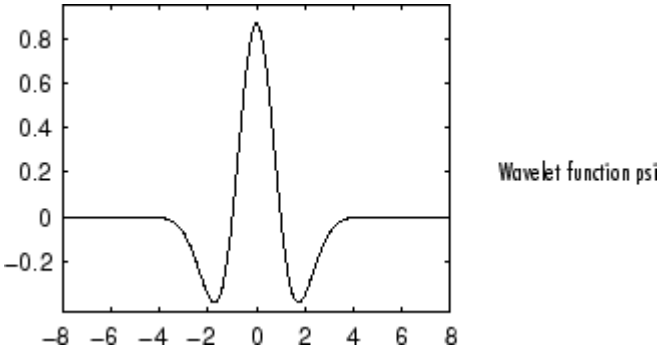
This wavelet has no scaling function, but is explicit.



You can obtain a survey of the main properties of this family by typing `waveinfo('morl')` from the MATLAB command line. See “Morlet Wavelet: morl” in the *Wavelet Toolbox User's Guide* for more detail.

Mexican Hat

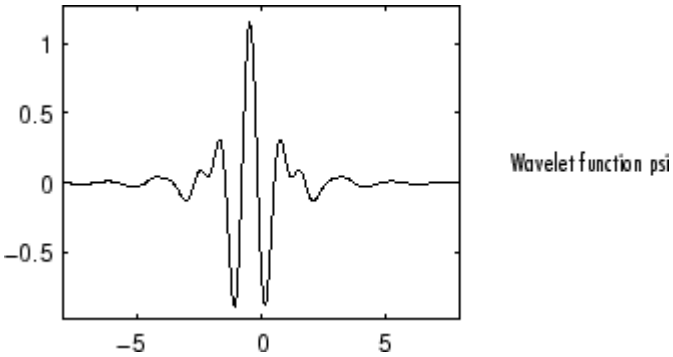
This wavelet has no scaling function and is derived from a function that is proportional to the second derivative function of the Gaussian probability density function. It is also knows as the Ricker wavelet.



You can obtain a survey of the main properties of this family by typing `waveinfo('mexh')` from the MATLAB command line. See “Mexican Hat Wavelet: mexh” in the *Wavelet Toolbox User's Guide* for more information.

Meyer

The Meyer wavelet and scaling function are defined in the frequency domain.



You can obtain a survey of the main properties of this family by typing `waveinfo('meyer')` from the MATLAB command line. See “Meyer Wavelet: meyr” in the *Wavelet Toolbox User's Guide* for more detail.

Other Real Wavelets

Some other real wavelets are available in the toolbox:

- Reverse Biorthogonal
- Gaussian derivatives family
- FIR based approximation of the Meyer wavelet

See “Additional Real Wavelets” in the *Wavelet Toolbox User's Guide* for more information.

Complex Wavelets

Some complex wavelet families are available in the toolbox:

- Gaussian derivatives
- Morlet
- Frequency B-Spline
- Shannon

See “Complex Wavelets” in the *Wavelet Toolbox User's Guide* for more information.

References

[Abr97] Abry, P. (1997), *Ondelettes et turbulence. Multirésolutions, algorithmes de décomposition, invariance d'échelles*, Diderot Editeur, Paris.

[Abr03]Abry, P., P. Flandrin, M.S. Taqqu, D. Veitch (2003), "Self-similarity and long-range dependence through the wavelet lens," *Theory and applications of long-range dependence*, Birkhäuser, pp. 527–556.

[Ant94] Antoniadis, A. (1994), "Smoothing noisy data with coiflets," *Statistica Sinica* 4 (2), pp. 651–678.

[AntO95] Antoniadis, A., G. Oppenheim, Eds. (1995), *Wavelets and statistics*, Lecture Notes in Statistics 103, Springer Verlag.

[AntP98] Antoniadis A., D.T. Pham (1998), "Wavelet regression for random or irregular design," *Comp. Stat. and Data Analysis*, 28, pp. 353–369.

[AntG99] Antoniadis, A., G. Gregoire (1999), "Density and Hazard rate estimation for right-censored data using wavelet methods," *J. R. Statist. Soc. B*, 61, 1, pp. 63–84.

[ArnABEM95]Arneodo, A., F. Argoul, E. Bacry, J. Elezgaray, J.F. Muzy (1995), *Ondelettes, multifractales et turbulence*, Diderot Editeur, Paris.

[Bak95] Bakshi, B. (1998), "Multiscale PCA with application to MSPC monitoring," *AIChE J.* 44, pp. 1596–1610.

[BarJM03] Bardet, J.-M., G. Lang, G. Oppenheim, A. Philippe, S. Stoev, M.S. Taqqu (2003), "Generators of long-range dependence processes: a survey" *Theory and applications of long-range dependence*, Birkhäuser Boston, pp. 579–623.

[BirM97] Birgé, L., P. Massart (1997), "From model selection to adaptive estimation," in D. Pollard (ed.), *Festschrift for L. Le Cam*, Springer, pp. 55–88.

[Bri95] Brislawn, C.M. (1995), "Fingerprints to digital," *Notices of the AMS*. Vol. 42, pp. 1278–1283.

[Bur96] Burke Hubbard, B. (1996), *The world according to wavelets*, AK Peters, Wellesley. The French original version is titled *Ondes et Ondelettes. La saga d'un outil mathématique*, Pour la Science, (1995).

[Chr06] Christophe, E., C. Mailhes, P. Duhamel (2006), "Adaptation of zerotrees using signed binary digit representations for 3 dimensional image coding," EURASIP *Journal on Image and Video Processing*, 2007, to appear in the special issue on Wavelets in Source Coding, Communications, and Networks, Paper ID 54679.

[Chu92a] Chui, C.K. (1992a), *Wavelets: a tutorial in theory and applications*, Academic Press.

[Chu92b] Chui, C.K. (1992b), *An introduction to wavelets*, Academic Press.

[Coh92] Cohen, A. (1992), "Ondelettes, analyses multirésolution et traitement numérique du signal," Ph.D. thesis, University of Paris IX, Dauphine.

[Coh95] Cohen, A. (1995), *Wavelets and multiscale signal processing*, Chapman and Hall.

[CohDF92] Cohen, A., I. Daubechies, J.C. Feauveau (1992), "Biorthogonal basis of compactly supported wavelets," *Comm. Pure Appli. Math.* , vol. 45, pp. 485–560.

[CohDJV93] Cohen, A., I. Daubechies, B. Jawerth, P. Vial (1993), "Multiresolution analysis, wavelets and fast wavelet transform on an interval," *CRAS Paris, Ser. A*, t. 316, pp. 417–421.

[CoiD95] Coifman, R.R., D.L. Donoho (1995), "Translation invariant de-noising," *Lecture Notes in Statistics*, 103, pp. 125–150.

[CoiMW92] Coifman, R.R., Y. Meyer, M.V. Wickerhauser (1992), "Wavelet analysis and signal processing," in *Wavelets and their applications*, M.B. Ruskai et al. (Eds.), pp. 153–178, Jones and Bartlett.

[CoiW92] Coifman, R.R., M.V Wickerhauser (1992), "Entropy-based algorithms for best basis selection," *IEEE Trans. on Inf. Theory*, vol. 38, 2, pp. 713–718.

[Dau92] Daubechies, I. (1992), *Ten lectures on wavelets*, SIAM.

[DevJL92] DeVore, R.A., B. Jawerth, B.J. Lucier (1992), "Image compression through wavelet transform coding," *IEEE Trans. on Inf. Theory*, vol. 38, 2, pp. 719–746.

[Don93] Donoho, D.L. (1993), "Progress in wavelet analysis and WVD: a ten minute tour," in *Progress in wavelet analysis and applications*, Y. Meyer, S. Roques, pp. 109–128. Frontières Ed.

[Don95] Donoho, D.L. (1995), "De-Noising by soft-thresholding," *IEEE Trans. on Inf. Theory*, vol. 41, 3, pp. 613–627.

[DonJ94a] Donoho, D.L., I.M. Johnstone (1994),"Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, pp. 425–455.

[DonJ94b] Donoho, D.L., I.M. Johnstone (1994), "Ideal de-noising in an orthonormal basis chosen from a library of bases," *CRAS Paris, Ser I*, t. 319, pp. 1317–1322.

[DonJKP95] Donoho, D.L., I.M. Johnstone, G. Kerkyacharian, D. Picard (1995), "Wavelet shrinkage: asymptopia," *Jour. Roy. Stat. Soc., series B*, vol. 57, no. 2, pp. 301–369.

[DonJKP96] Donoho, D.L., I.M. Johnstone, G. Kerkyacharian, D. Picard (1996), "Density estimation by wavelet thresholding," *Annals of Stat.*, 24, pp. 508–539.

[Fla92] Flandrin, P. (1992), "Wavelet analysis and synthesis of fractional Brownian motion," *IEEE Trans. on Inf. Th.*, 38, pp. 910–917.

[HalPKP97] Hall, P., S. Penev, G. Kerkyacharian, D. Picard (1997), "Numerical performance of block thresholded wavelet estimators," *Stat. and Computing*, 7, pp. 115–124.

[HarKPT98] Hardle, W., G. Kerkyacharian, D. Picard, A. Tsybakov (1998), *Wavelets, approximation and statistical applications*, Lecture Notes in Statistics, 129, Springer Verlag.

[Ist94] Istas, J., G. Lang (1994), "Quadratic variations and estimation of the local Hölder index of a Gaussian process," *Ann. Inst. Poincaré*, 33, pp. 407–436.

[KahL95] Kahane, J.P., P.G Lemarié (1995), *Fourier series and wavelets*, Gordon and Research Publishers, Studies in the Development of Modern Mathematics, vol 3.

[Kai94] Kaiser, G. (1994), *A friendly guide to wavelets*, Birkhäuser.

[Lav99] Lavielle, M. (1999), "Detection of multiple changes in a sequence of dependent variables," *Stoch. Proc. and their Applications*, 83, 2, pp. 79–102.

[Lem90] Lemarié, P.G., Ed., (1990), *Les ondelettes en 1989*, *Lecture Notes in Mathematics*, Springer Verlag.

[Mal89] Mallat, S. (1989), "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, pp. 674–693.

[Mal98] Mallat, S. (1998), *A wavelet tour of signal processing*, Academic Press.

[Mey90] Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*), Cambridge Univ. Press, 1993.

[Mey93] Meyer, Y. (1993), *Les ondelettes. Algorithmes et applications*, Colin Ed., Paris, 2nd edition. (English translation: *Wavelets: algorithms and applications*, SIAM).

[MeyR93] Meyer, Y., S. Roques, Eds. (1993), *Progress in wavelet analysis and applications*, Frontières Ed.

[MisMOP93a] Misiti, M., Y. Misiti, G. Oppenheim, J.M. Poggi (1993a), "Analyse de signaux classiques par décomposition en ondelettes," *Revue de Statistique Appliquée*, vol. XLI, no. 4, pp. 5–32.

[**MisMOP93b]** Misiti, M., Y. Misiti, G. Oppenheim, J.M. Poggi (1993b), "Ondelettes en statistique et traitement du signal," *Revue de Statistique Appliquée*, vol. XLI, no. 4, pp. 33–43.

[MisMOP94] Misiti, M., Y. Misiti, G. Oppenheim, J.M. Poggi (1994), "Décomposition en ondelettes et méthodes comparatives: étude d'une courbe de charge électrique," *Revue de Statistique Appliquée*, vol. XLII, no. 2, pp. 57–77.

[MisMOP03] Misiti, M., Y. Misiti, G. Oppenheim, J.-M. Poggi (2003), "Les ondelettes et leurs applications," Hermes.

[MisMOP07] Misiti, M., Y. Misiti, G. Oppenheim, J.-M. Poggi (2007), *Wavelets and their applications*, ISTE DSP Series.

[NasS95] Nason, G.P., B.W. Silverman (1995), "The stationary wavelet transform and some statistical applications," *Lecture Notes in Statistics*, 103, pp. 281–299.

[Ogd97] Ogden, R.T. (1997), *Essential wavelets for statistical applications and data analysis*, Birkhäuser.

[PesKC96] Pesquet, J.C., H. Krim, H. Carfatan (1996), "Time-invariant orthonormal wavelet representations," *IEEE Trans. Sign. Proc.*, vol. 44, 8, pp. 1964–1970.

[Sai96] Said A., W.A. Pearlman (1996), "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 6, No. 3, pp. 243–250.

[Sha93] Shapiro J.M. (1993), "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Proc.*, Vol. 41, No. 12, pp. 3445–3462.

[StrN96] Strang, G., T. Nguyen (1996), *Wavelets and filter banks*, Wellesley-Cambridge Press.

[Swe98] Sweldens, W. (1998), "The Lifting Scheme: a Construction of Second Generation of Wavelets," *SIAM J. Math. Anal.*, 29 (2), pp. 511–546.

[Teo98] Teolis, A. (1998), *Computational signal processing with wavelets*, Birkhäuser.

[VetK95] Vetterli, M., J. Kovacevic (1995), *Wavelets and subband coding*, Prentice Hall.

[Wal99] Walker, J.S. (1999), "Wavelet-Based Image Compression," University of Wisconsin, Eau Claire, Wisconsin, USA, , Sub-chapter of CRC Press book: *Transform and Data Compression. A Primer on Wavelets and Their Scientific Applications*. A second edition is published in 2008.

[Wic91] Wickerhauser, M.V. (1991), "INRIA lectures on wavelet packet algorithms," *Proceedings ondelettes et paquets d'ondes*, 17–21 June, Rocquencourt France, pp. 31–99.

[Wic94] Wickerhauser, M.V. (1994), *Adapted wavelet analysis from theory to software algorithms*, A.K. Peters.

[Zee98] Zeeuw, P.M. (1998), "Wavelet and image fusion," CWI, Amsterdam, March 1998, `http://www.cwi.nl/~pauldz/`

Using Wavelets

This chapter takes you step-by-step through examples that teach you how to use the graphical tools and command-line functions.

- “Introduction to Wavelet Toolbox App and Functions” on page 2-2
- “Wavelets: Working with Images” on page 2-3
- “Density Estimation Using Wavelets” on page 2-8
- “1-D Wavelet Coefficient Selection Using the Wavelet Analyzer App” on page 2-13
- “2-D Wavelet Coefficient Selection Using the Wavelet Analyzer App” on page 2-20
- “1-D Extension Using the Command Line” on page 2-25
- “2-D Extension Using the Command Line” on page 2-26
- “Image Fusion” on page 2-27
- “1-D Fractional Brownian Motion Synthesis” on page 2-33
- “New Wavelet for CWT” on page 2-35
- “Additional Wavelet GUIs” on page 2-38

Introduction to Wavelet Toolbox App and Functions

Wavelet Toolbox software contains graphical tools and command-line functions that let you

- Examine and explore properties of individual wavelets and wavelet packets
- Examine statistics of signals and signal components
- Perform a continuous wavelet transform of a 1-D signal
- Perform discrete analysis and synthesis of 1-D and 2-D signals
- Perform wavelet packet analysis of 1-D and 2-D signals
- Compress and remove noise from signals and images

In addition to the above, the toolbox makes it easy to customize the presentation and visualization of your data. You choose

- Which signals to display
- A region of interest to magnify
- A coloring scheme for display of wavelet coefficient details

Note All the Wavelet Analyzer app tools described here let you import information from and export information to either the disk or workspace.

Wavelets: Working with Images

This section provides additional information about working with images in the Wavelet Toolbox software. It describes the types of supported images and how the MATLAB environment represents them, as well as techniques for analyzing color images.

Understanding Images in the MATLAB Environment

The basic data structure in MATLAB is the rectangular *matrix*, an ordered set of real or complex elements. This object is naturally suited to the representation of *images*, which are real-valued, ordered sets of color or intensity data. (This toolbox does not support complex-valued images.)

The word *pixel* is derived from *picture element* and usually denotes a single dot on a computer display, or a single element in an image matrix. You can select a single pixel from an image matrix using normal matrix subscripting. For example:

```
I(2,15)
```

returns the value of the pixel at row 2 and column 15 of the image `I`. By default, MATLAB scales images to fill the display axes; therefore, an image pixel may use more than a single pixel on the screen.

Indexed Images

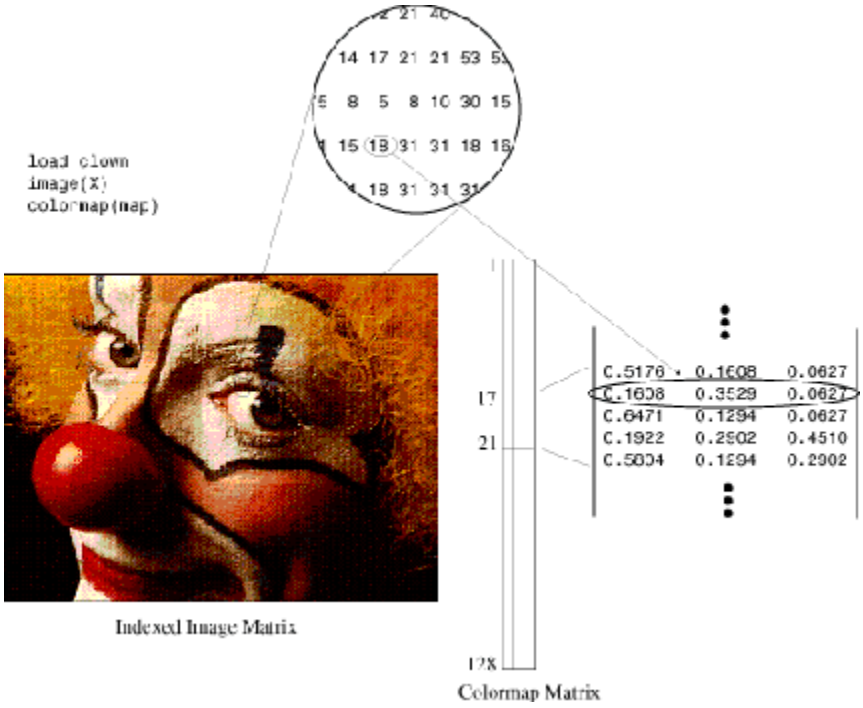
A typical color image requires two matrices: a colormap and an image matrix. The *colormap* is an ordered set of values that represent the colors in the image. For each image pixel, the *image matrix* contains a corresponding index into the colormap. (The elements of the image matrix are floating-point integers, or *flints*, which MATLAB stores as double-precision values.)

The size of the colormap matrix is `n`-by-3 for an image containing `n` colors. Each row of the colormap matrix is a 1-by-3 red, green, blue (RGB) color vector

```
color = [R G B]
```

that specifies the intensity of the red, green, and blue components of that color. `R`, `G`, and `B` are real scalars that range from 0.0 (black) to 1.0 (full intensity). MATLAB translates these values into display intensities when you display an image and its colormap.

When MATLAB displays an indexed image, it uses the values in the image matrix to look up the desired color in the colormap. For instance, if the image matrix contains the value 18 in matrix location (86,198), the color for pixel (86,198) is the color from row 18 of the colormap.



Outside MATLAB, indexed images with `n` colors often contain values from 0 to n–1. These values are indices into a colormap with 0 as its first index. Since MATLAB matrices start with index 1, you must increment each value in the image, or *shift up* the image, to create an image that you can manipulate with toolbox functions.

Wavelet Decomposition of Indexed Images

Indexed images can be thought of as scaled intensity images, with matrix elements containing only integers from 1 to `n`, where `n` is the number of discrete shades in the image.

If the colormap is not provided, the Wavelet Analyzer app displays the image and processing results using a monotonic colormap with `max(max(X))-min(min(X))+1` colors.

Since the image colormap is only used for display purposes, some indexed images may need to be preprocessed to achieve the correct results from the wavelet decomposition.

In general, color indexed images do not have linear, monotonic colormaps and need to be converted to the appropriate gray-scale indexed image before performing a wavelet decomposition.

How Decompositions Are Displayed

Note that the coefficients, approximations, and details produced by wavelet decomposition are not indexed image matrices.

To display these images in a suitable way, the Wavelet Analyzer app follows these rules:

- Reconstructed approximations are displayed using the colormap `map`.
- The coefficients and the reconstructed details are displayed using the colormap `map` applied to a rescaled version of the matrices.

RGB (Truecolor) Images

An RGB image, sometimes referred to as a truecolor image, is stored in MATLAB as an *m*-by-*n*-by-3 data array that defines red, green, and blue color components for each individual pixel. RGB images do not use a palette. The color of each pixel is determined by the combination of the red, green, and blue intensities stored in each color plane at the pixel's location. Graphics file formats store RGB images as 24-bit images, where the red, green, and blue components are 8 bits each. This yields a potential of 16 million colors.

The precision with which a real-life image can be replicated led to the nickname “truecolor image.” An RGB MATLAB array can be of class `double`, `single`, `uint8`, or `uint16`. In an RGB array of class `double`, each color component is a value between 0 and 1.

The color components of an 8-bit RGB image are integers in the range [0, 255] rather than floating-point values in the range [0, 1].

Wavelet Decomposition of Truecolor Images

The truecolor images analyzed are *m*-by-*n*-by-3 arrays of `uint8`. Each of the three-color components is a matrix that is decomposed using the 2-D wavelet decomposition scheme.

Other Images

Wavelet Toolbox software lets you work with some other types of images. Using the `imread` function, the various tools using images try to load indexed images from files that are not MAT files (for example, PCX files).

These tools are:

- 2-D Discrete Wavelet Analysis
- 2-D Wavelet Packet Analysis
- 2-D Stationary Wavelet Analysis

For more information on the supported file types, type `help imread`.

Use the `imfinfo` function to find the type of image stored in the file. If the file does not contain an indexed image, the load operation fails.

Image Conversion

Image Processing Toolbox software provides a comprehensive set of functions that let you easily convert between image types. If you do not have Image Processing Toolbox software, the examples below demonstrate how this conversion may be performed using basic MATLAB commands.

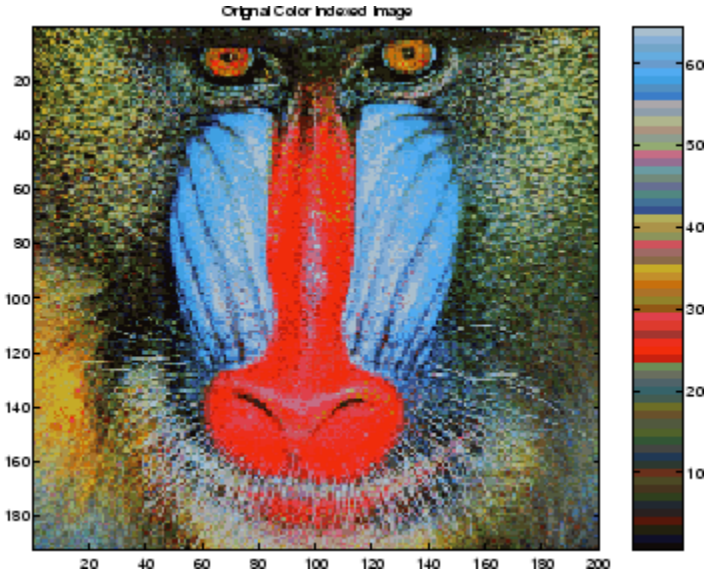
Example 1: Converting Color Indexed Images

```
load xpmndrll
whos
```

Name	Size	Bytes	Class
X2	192x200	307200	double array

Name	Size	Bytes	Class
map	64x3	1536	double array

```
image(X2)
title('Original Color Indexed Image')
colormap(map); colorbar
```



The color bar to the right of the image is not smooth and does not monotonically progress from dark to light. This type of indexed image is not suitable for direct wavelet decomposition with the toolbox and needs to be preprocessed.

First, separate the color indexed image into its RGB components:

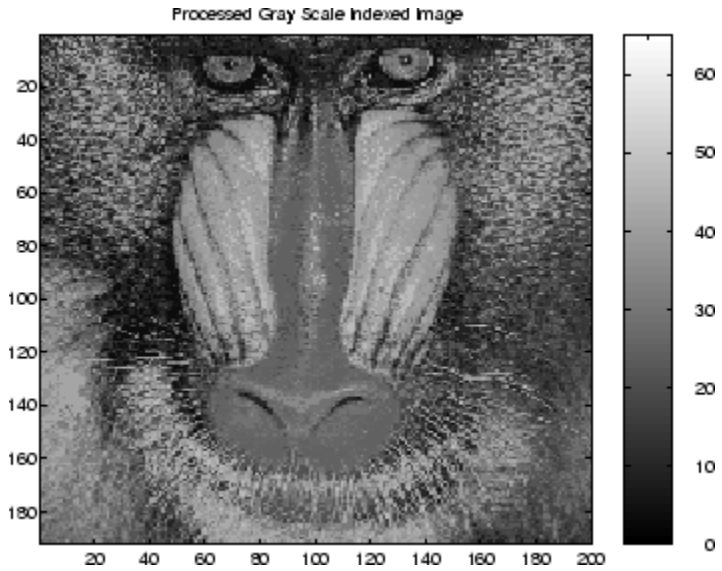
```
R = map(X2,1); R = reshape(R,size(X2));
G = map(X2,2); G = reshape(G,size(X2));
B = map(X2,3); B = reshape(B,size(X2));
```

Next, convert the RGB matrices into a gray-scale intensity image, using the standard perceptual weightings for the three-color components:

```
Xrgb = 0.2990*R + 0.5870*G + 0.1140*B;
```

Then, convert the gray-scale intensity image back to a gray-scale indexed image with 64 distinct levels and create a new colormap with 64 levels of gray:

```
n = 64;            % Number of shades in new indexed image
X = round(Xrgb*(n-1)) + 1;
map2 = gray(n);
figure
image(X), title('Processed
Gray Scale Indexed Image')
colormap(map2), colorbar
```



The color bar of the converted image is now linear and has a smooth transition from dark to light. The image is now suitable for wavelet decomposition.

Finally, save the converted image in a form compatible with the Wavelet Toolbox Wavelet Analyzer app:

```
baboon = X;
map = map2;
save baboon baboon map
```

Example 2: Converting an RGB TIF Image

Suppose the file `myImage.tif` contains an RGB image (noncompressed) of size `S1xS2`. Use the following commands to convert this image:

```
A = imread('myImage.tif');
% A is an S1xS2x3 array of uint8.

A = double(A);
Xrgb  = 0.2990*A(:,:,1) + 0.5870*A(:,:,2) + 0.1140*A(:,:,3);
NbColors = 255;
X = wcodemat(Xrgb,NbColors);
map = pink(NbColors);
```

The same program can be used to convert BMP or JPEG files.

Density Estimation Using Wavelets

This section takes you through the features of 1-D wavelet density estimation using one of the Wavelet Toolbox specialized tools.

The toolbox provides Wavelet Analyzer app to estimate the density of a sample and complement well known tools like the histogram (available from the MATLAB core) or kernel based estimates.

For the examples in this section, switch the extension mode to symmetric padding, using the command

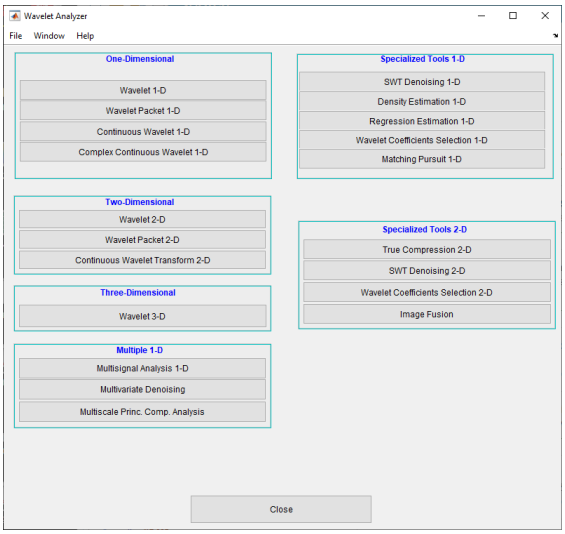
```
dwtmode('sym')
```

1-D Estimation Using the Wavelet Analyzer App

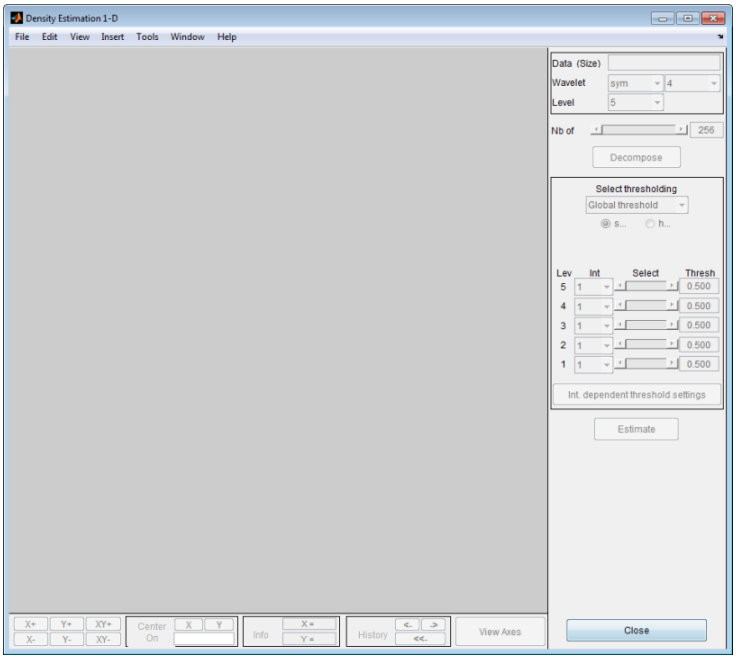
1 Start the Density Estimation 1-D Tool.

From the MATLAB prompt, type `waveletAnalyzer`.

The **Wavelet Analyzer** appears.



Click the **Density Estimation 1-D** menu item. The discrete wavelet analysis tool for 1-D density estimation appears.



2 Load data.

At the MATLAB command line, type

```
load ex1cusp1
```

In the **Density Estimation 1-D** tool, choose **File > Import from Workspace**.

When the **Import from Workspace** dialog box appears, select `ex1cusp1`. Click **OK** to import the noisy cusp data.

The sample, a 64-bin histogram, and the processed data obtained after a binning are displayed. In this example, we'll accept the default value for the number of bins (250). The binned data, suitably normalized, will be processed by wavelet decomposition.

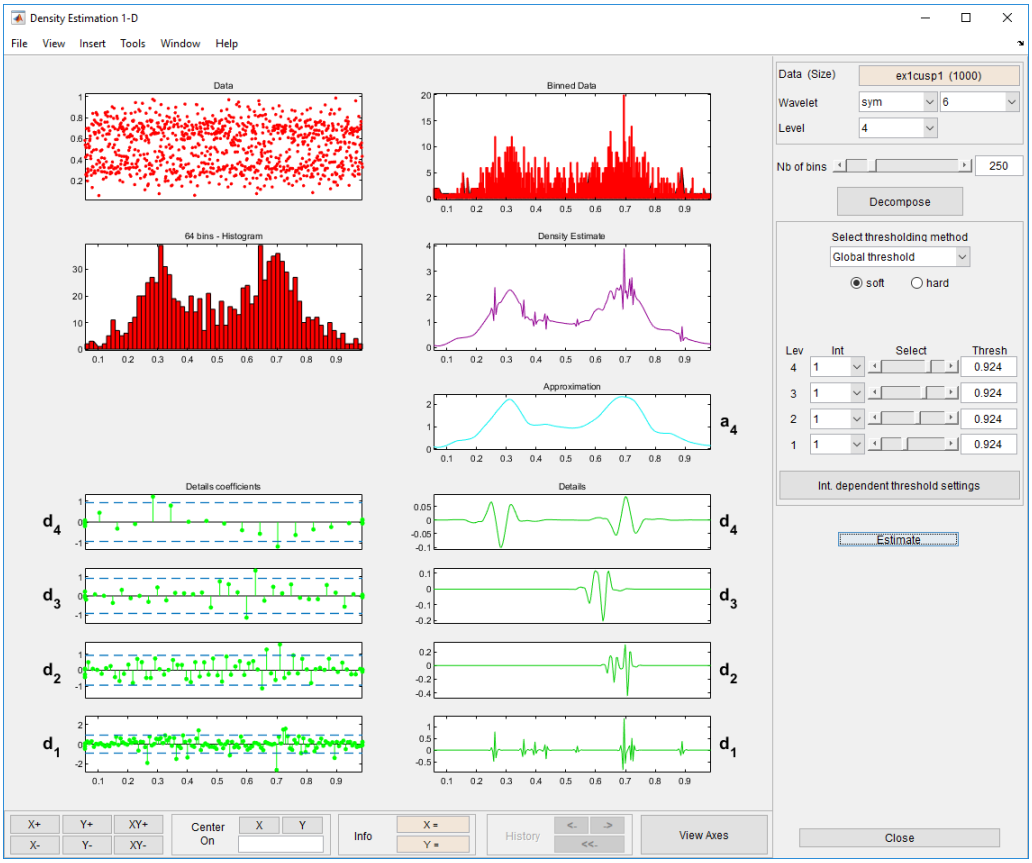
3 Perform a Wavelet Decomposition of the binned data.

Select the `sym6` wavelet from the **Wavelet** menu and select **4** from the **Level** menu, and click the **Decompose** button. After a pause for computation, the tool displays the detail coefficients of the decomposition of the binned data.

4 Perform a density estimation.

Accept the defaults of global soft thresholding. The sliders located on the right of the window control the level dependent thresholds, indicated by dashed blue lines running horizontally through the graphs on the left of the window.

Continue by clicking the **Estimate** button.

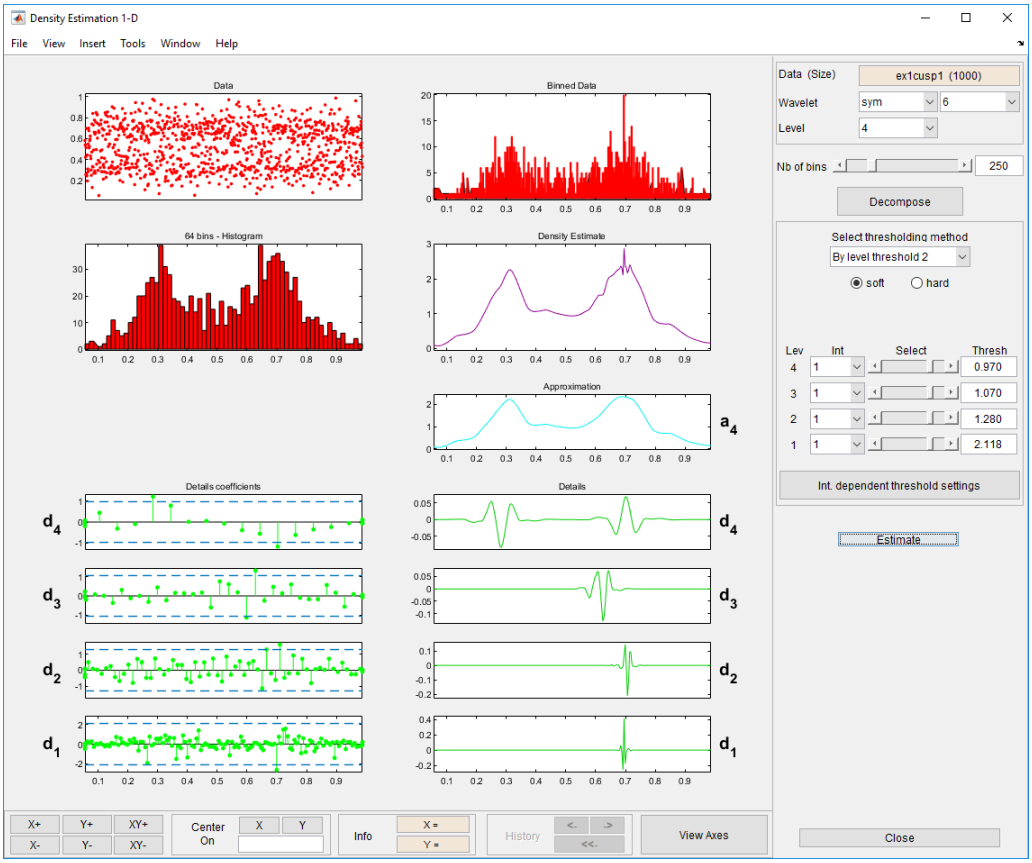


You can see that the estimation process delivers a very irregular resulting density. The density estimate (in purple) is the normalized sum of the signals located below it: the approximation `a4` and the reconstructed details after coefficient thresholding.

5 Perform thresholding.

You can experiment with the various predefined thresholding strategies by selecting the appropriate options from the menu located on the right of the window or directly by dragging the dashed blue lines with the left mouse button. Let's try another estimation method.

From the menu **Select thresholding method**, select the item **By level threshold 2**. Next, click the **Estimate** button.



The estimated density is more satisfactory. It correctly identifies the smooth part of the density and the cusp at 0.7.

Importing and Exporting Information from the Wavelet Analyzer App

The tool lets you save the estimated density to disk. The toolbox creates a MAT-file in the current folder with a name you choose.

To save the estimated density, use the menu option **File > Save Density**. A dialog box appears that lets you specify a folder and filename for storing the density. Type the name `dex1cusp`. After saving the density data to the file `dex1cusp.mat`, load the variables into your workspace:

```
load dex1cusp
whos
```

Name	Size	Bytes	Class
ex1cusp1	1x1000	8000	double array
thrParams	1x4	544	cell array
wname	1x4	8	char array
xdata	1x250	2000	double array
ydata	1x250	2000	double array

The original noisy cusp data `ex1cusp1` has 1000 samples. The variables `thrParams`, `wname`, `xdata`, and `ydata` are stored in `dex1cusp.mat`. The estimated density is given by `xdata` and `ydata`. The length of these vectors is equal to the number of bins you choose in step 4. In addition, the parameters of the estimation process are given by the wavelet name in `wname`.

```
wname
```

```
wname =
    sym6
```

and the level dependent thresholds contained in `thrParams`, which is a cell array of length 4 (the level of the decomposition). For i from 1 to 4, `thrParams{i}` contains the lower and upper bounds of the interval of thresholding and the threshold value (since interval dependent thresholds are allowed). For more information, see “1-D Adaptive Thresholding of Wavelet Coefficients”. For example, for level 1,

```
thrParams{1}
ans =
    0.0560    0.9870    2.1179
```

Note When you load data from a file using the menu option **File > Load Data for Density Estimate**, the first 1-D variable encountered in the file is considered the signal. Variables are inspected in alphabetical order.

At the end of this section, turn the extension mode back to zero padding using

```
dwtmode('zpd')
```

1-D Wavelet Coefficient Selection Using the Wavelet Analyzer App

This section takes you through the features of 1-D selection of wavelet coefficients using one of the Wavelet Toolbox specialized tools. The toolbox provides the Wavelet Analyzer app to explore some reconstruction schemes based on various wavelet coefficients selection strategies:

- Global selection of biggest coefficients (in absolute value)
- By level selection of biggest coefficients
- Automatic selection of biggest coefficients
- Manual selection of coefficients

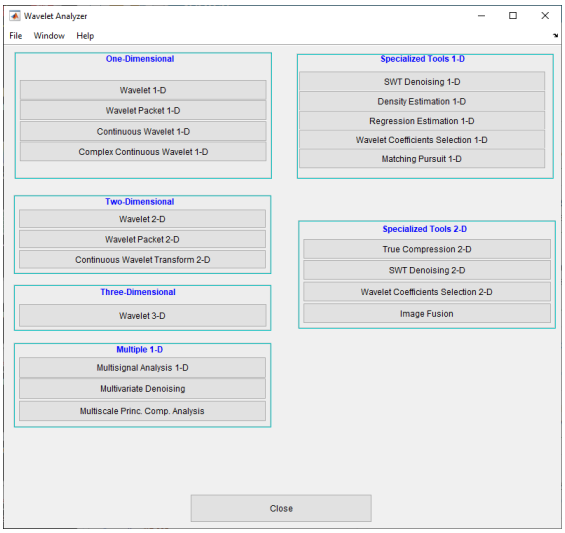
For this section, switch the extension mode to symmetric padding using the command

```
dwtmode('sym')
```

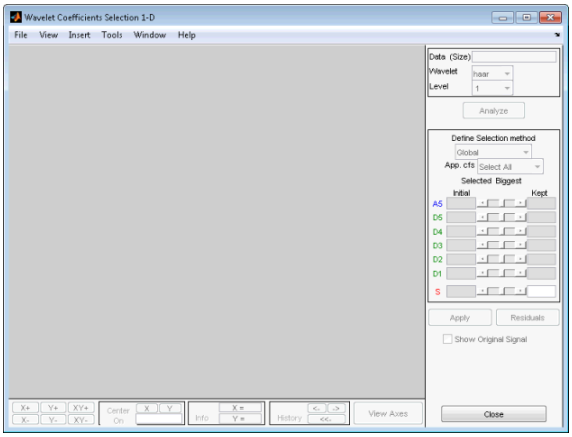
1 Start the Wavelet Coefficients Selection 1-D Tool.

From the MATLAB prompt, type `waveletAnalyzer`.

The **Wavelet Analyzer** appears.



Click the **Wavelet Coefficients Selection 1-D** menu item. The discrete wavelet coefficients selection tool for 1-D signals appears.



2 Load data.

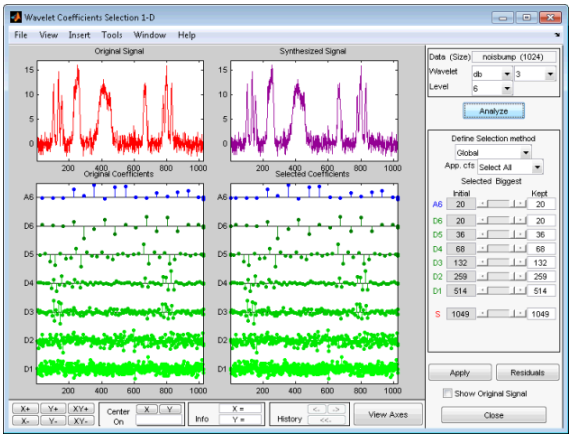
At the MATLAB command prompt, type

```
load noisbump
```

In the **Wavelet Coefficients Selection 1-D** tool, select **File > Import Signal from Workspace**. When the **Import from Workspace** dialog box appears, select the `noisbump` variable. Click **OK** to import the noisy bumps data

3 Perform a Wavelet Decomposition.

Select the `db3` wavelet from the **Wavelet** menu and select 6 from the **Level** menu, and then click the **Analyze** button.

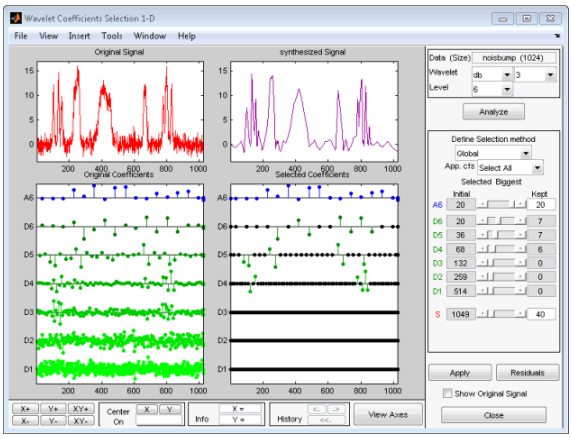


The tool displays below the original signal (on the left) its wavelet decomposition: the approximation coefficients `A6` and detail coefficients from `D6` at the top to `D1` at the bottom. In the middle of the window, below the synthesized signal (which at this step is the same, since all the wavelet coefficients are kept) it displays the selected coefficients.

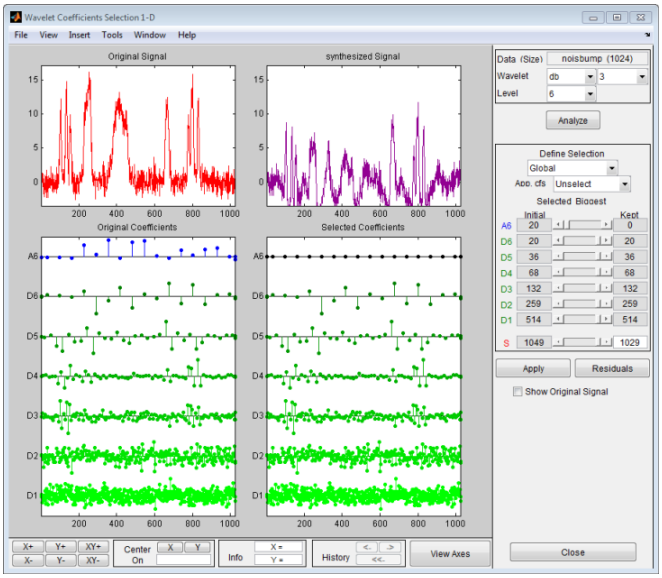
Selecting Biggest Coefficients Globally

On the right of the window, find a column labeled **Kept**. The last line shows the total number of coefficients: 1049. This is a little bit more than the number of observations, which is 1024. You can choose the number of selected biggest coefficients by typing a number instead of 1049 or by using the slider. Type `40` and press **Enter**. The numbers of selected biggest coefficients level by

level are updated (but cannot be modified since **Global** is the current selection method). Then click the **Apply** button. The resulting coefficients are now displayed.

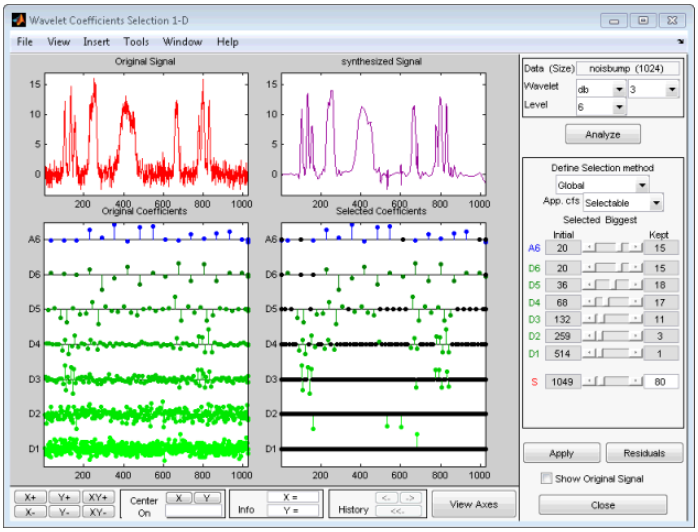


In the previous trial, the approximation coefficients were all kept. It is possible to relax this constraint by selecting another option from the **App. cfs** menu (Approximation Coefficients abbreviation). Choose the **Unselect** option and click the **Apply** button.



None of the approximation coefficients are kept.

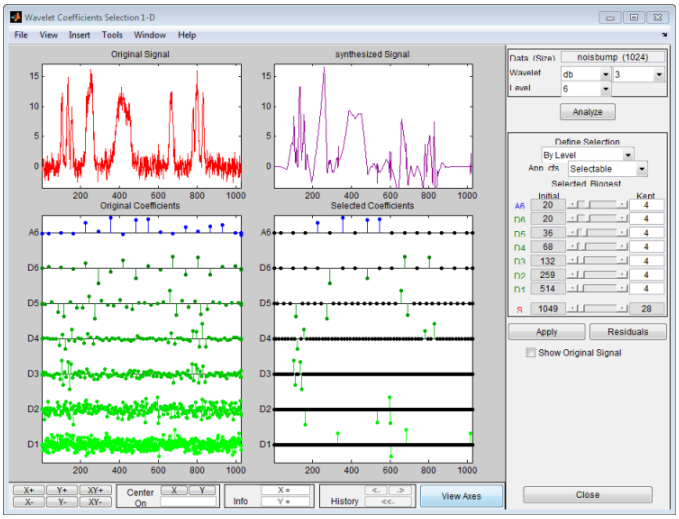
From the **App. cfs** menu, select the **Selectable** option. Type `80` for the number of selected biggest coefficients and press **Enter**. Then, click the **Apply** button.



Some of the approximation coefficients (15) have been kept.

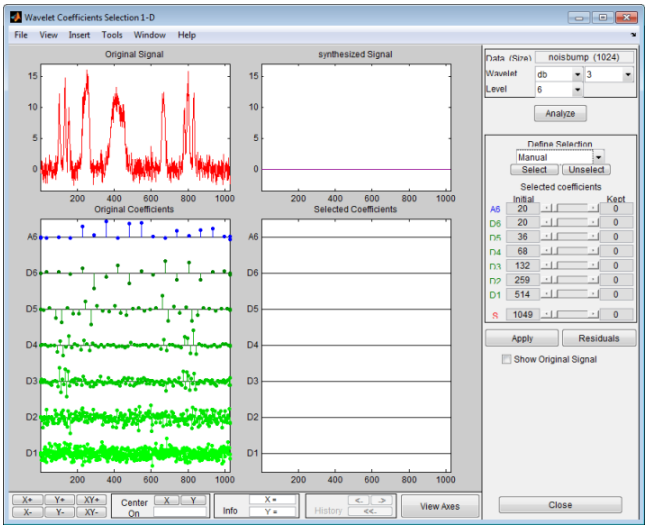
Selecting Biggest Coefficients by Level

From the **Define Selection method** menu, select the **By Level** option. You can choose the number of selected biggest coefficients by level or select it using the sliders. Type 4 for the approximation and each detail, and then click the **Apply** button.

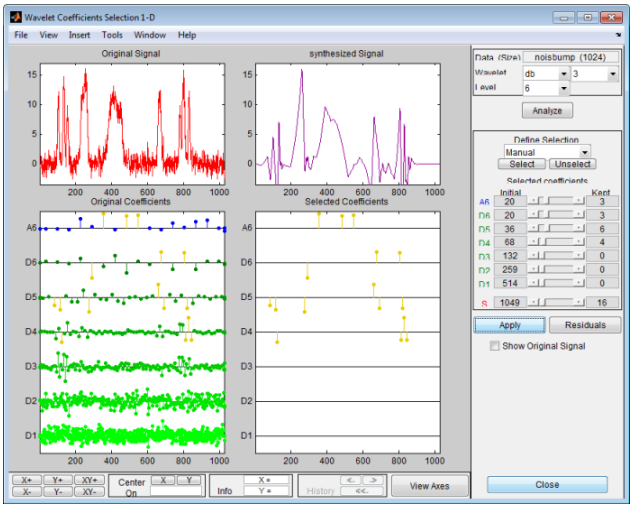


Selecting Coefficients Manually

From the **Define Selection method** menu, select the **Manual** option. The tool displays on the left part, below the original signal, its wavelet decomposition. At the beginning, no coefficients are kept so no selected coefficient is visible and the synthesized signal is null.



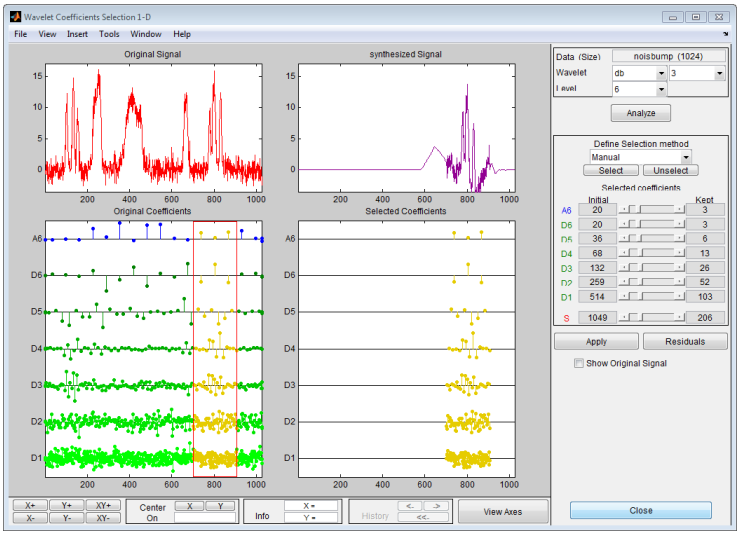
Select 16 coefficients individually by double clicking each of them using the left mouse button. The color of selected coefficients switches from green to yellow for the details and from blue to yellow for the approximation, which appear on the left of the window and appear in yellow on the middle part. Click the **Apply** button.



You can deselect the currently selected coefficients by double clicking each of them. Another way to select or deselect a set of coefficients is to use the selection box. Drag a rubber band box (hold down the left mouse button) over a portion of the coefficient axes (original or selected) containing all the currently selected coefficients. Click the **Unselect** button located on the right of the window. Click the **Apply** button. The tool displays the null signal again.

Note that when the coefficients are very close, it is easier to zoom in before selecting or deselecting them.

Drag a rubber band box over the portion of the coefficient axes around the position 800 and containing all scales and click the **Select** button. Click the **Apply** button.

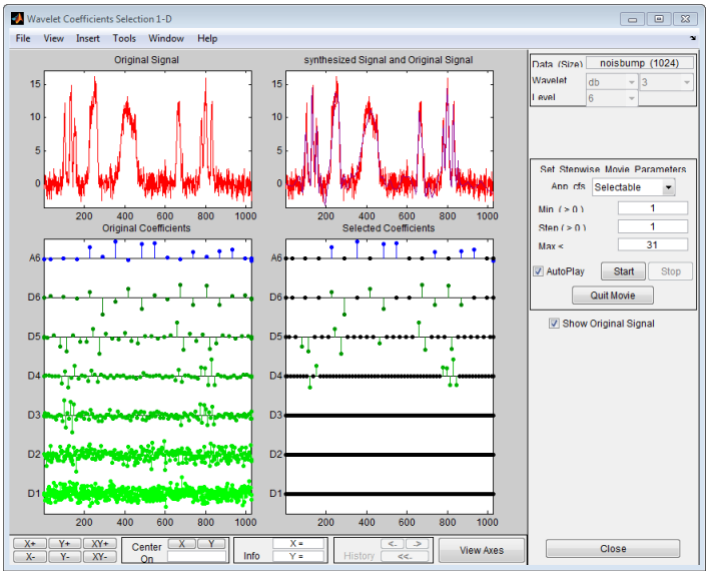


This illustrates that wavelet analysis is a local analysis since the signal is perfectly reconstructed around the position 800. Check the **Show Original Signal** to magnify it.

Selecting Coefficients Automatically

From the **Define Selection method** menu, select the **Stepwise movie** option. The tool displays the same initial window as in the manual selection mode, except for the left part of it.

Let's perform the stepwise movie using the k biggest coefficients, from `k = 1` to `k = 31` in steps of 1, click the **Start** button. As soon as the result is satisfactory, click the **Stop** button.



4 Save the synthesized signal.

The tool lets you save the synthesized signal to disk. The toolbox creates a MAT-file in the current folder with a name you choose.

To save the synthesized signal from the present selection, use the menu option **File > Save Synthesized Signal**. A dialog box appears that lets you specify a folder and filename for storing the signal and the wavelet name.

At the end of this section, turn back the extension mode to zero padding using the command

```
dwtmode('zpd')
```

2-D Wavelet Coefficient Selection Using the Wavelet Analyzer App

This section takes you through the features of 2-D selection of wavelet coefficients using one of the Wavelet Toolbox specialized tools. The toolbox provides the Wavelet Analyzer app to explore some reconstruction schemes based on various wavelet coefficient selection strategies:

- Global selection of biggest coefficients (in absolute value)
- By level selection of biggest coefficients
- Automatic selection of biggest coefficients.

This section will be short since the functionality are similar to the 1-D ones examined in the previous section.

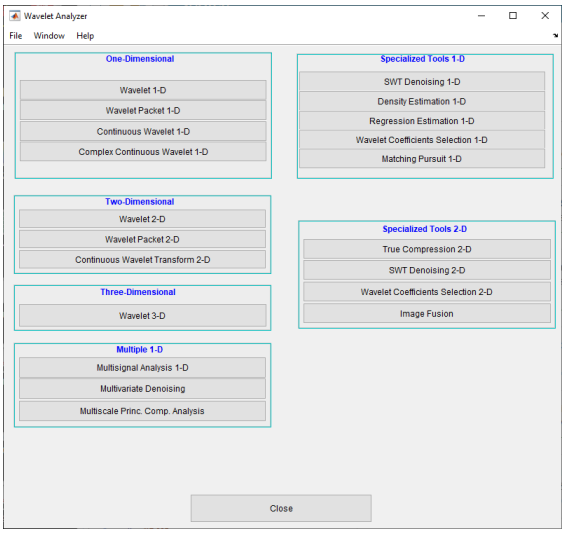
For this section, switch the extension mode to symmetric padding using the command

```
dwtmode('sym')
```

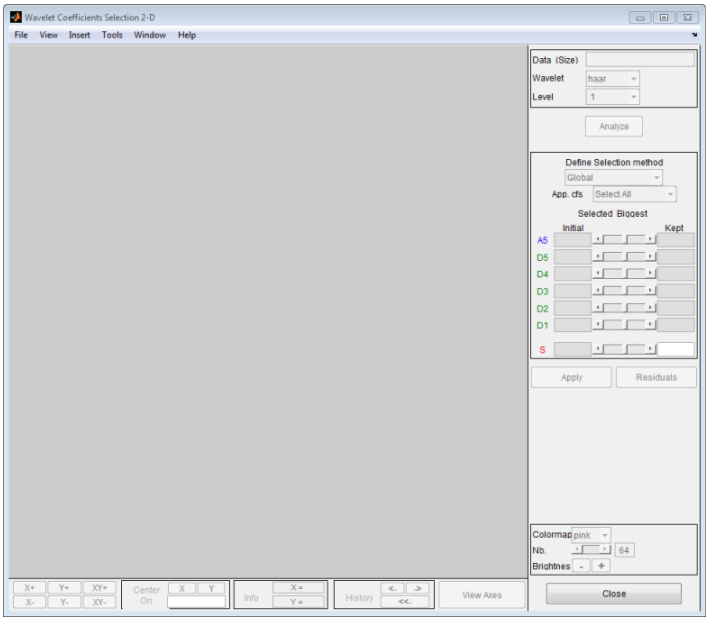
1 Start the Wavelet Coefficients Selection 2-D Tool.

From the MATLAB prompt, type `waveletAnalyzer`.

The **Wavelet Analyzer** appears.



Click the **Wavelet Coefficients Selection 2-D** menu item. The discrete wavelet coefficients selection tool for images appears.



2 Load data.

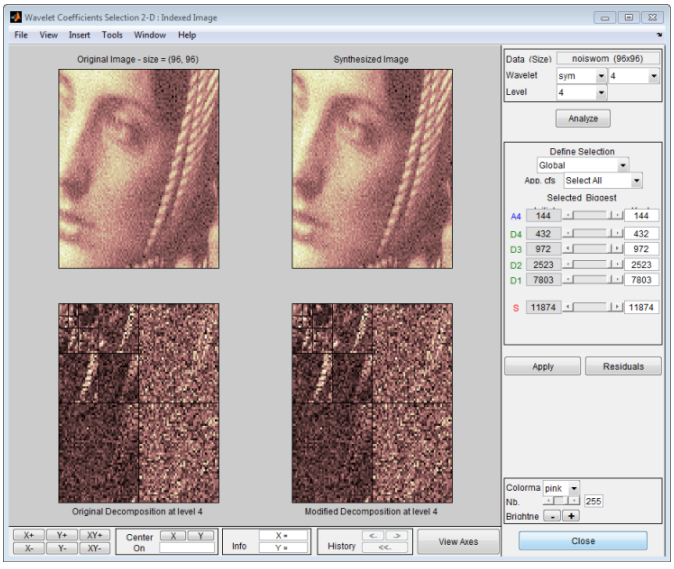
At the MATLAB command prompt, type

```
load noiswom;
```

In the **Wavelet Coefficients Selection 2-D** tool, select **File > Import from Workspace**. When the **Import from Workspace** dialog box appears, select the X variable. Click **OK** to import the image.

3 Perform a Wavelet Decomposition.

Select the `sym4` wavelet from the **Wavelet** menu and select **4** from the **Level** menu, and then click the **Analyze** button.



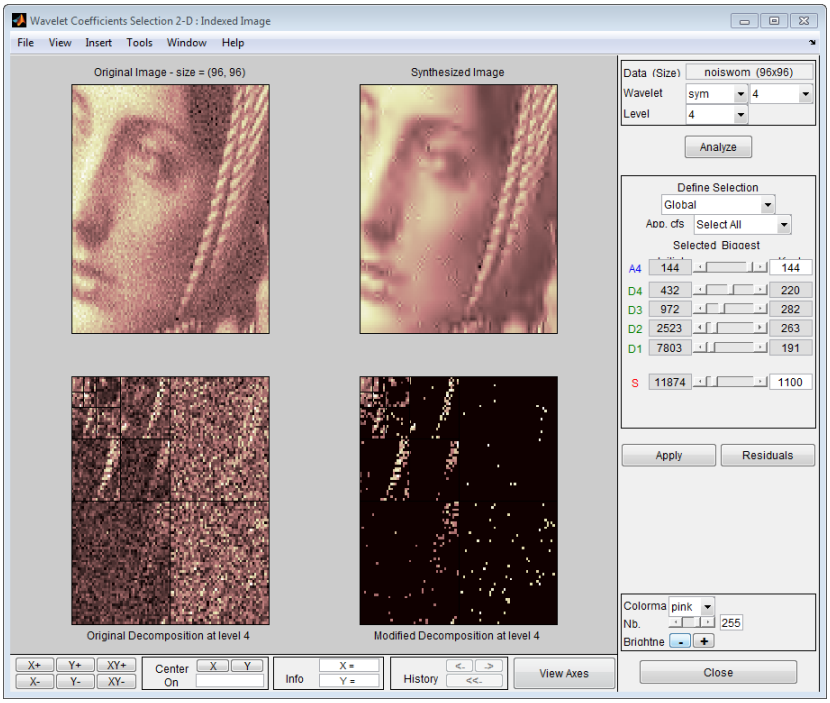
The tool displays its wavelet decomposition below the original image (on the left). The selected coefficients are displayed in the middle of the window, below the synthesized image (which, at this step, is the same since all the wavelet coefficients are kept). There are 11874 coefficients, a little bit more than the original image number of pixels, which is 96x96 = 9216.

Note The difference between 9216 and 11874 comes from the extra coefficients generated by the redundant DWT using the current extension mode (symmetric, `'sym'`). Because 96 is divisible by $2^4 = 16$, using the periodic extension mode (`'per'`) for the DWT, you obtain for each level the minimum number of coefficients. More precisely, if you type `dwtmode('per')` and repeat steps **2** to **5**, you obtain 9216 coefficients.

Selecting Biggest Coefficients Globally

On the right of the window, find a column labeled **Kept**. The last line shows the total number of coefficients: 11874. This is a little bit more than the original image number of pixels. You can choose the number of selected biggest coefficients by typing a number instead of 11874, or by using the slider. Type `1100` and press **Enter**. The numbers of selected biggest coefficients level by level are updated (but cannot be modified, since **Global** is the current selection method).

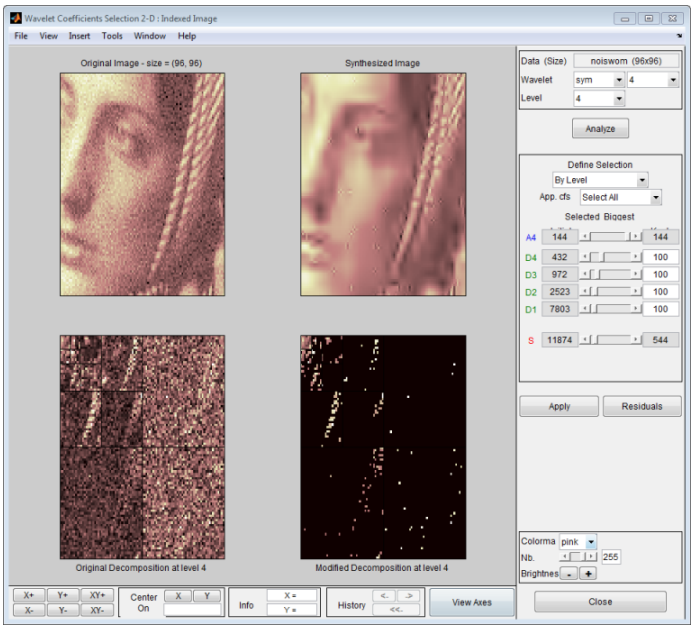
Then click the **Apply** button.



In the previous operation, all the approximation coefficients were kept. It is possible to relax this constraint by selecting another option from the **App. cfs** menu (see “1-D Wavelet Coefficient Selection Using the Wavelet Analyzer App” on page 2-13).

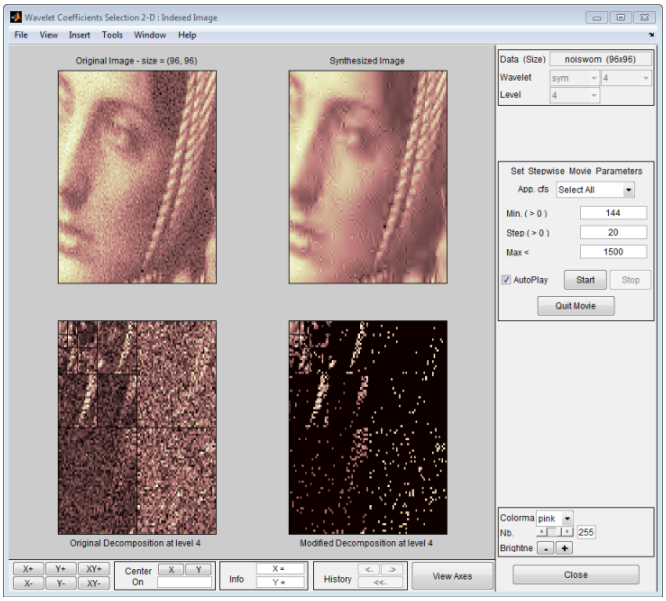
Selecting Biggest Coefficients by Level

Selecting Biggest Coefficients by Level. From the **Define Selection method** menu, select the **By Level** option. You can choose the number of selected biggest coefficients by level, or select it using the sliders. Type `100` for each detail, and then click the **Apply** button.



Selecting Coefficients Automatically

From the **Define Selection method** menu, select the **Stepwise movie** option. The tool displays its wavelet decomposition on the left, below the original image. At the beginning, no coefficients are kept so the synthesized image is null. Perform the stepwise movie using the `k` biggest coefficients, from `k = 144` to `k = 1500`, in steps of `20`. Click the **Start** button. As soon as the result is satisfactory, click the **Stop** button.



We've stopped the movie at 864 coefficients (including the number of approximation coefficients).

4 Save the synthesized image.

This tool lets you save the synthesized image to disk. The toolbox creates a MAT-file in the current folder with a name you choose.

To save the synthesized image from the present selection, use the menu option **File > Save Synthesized Image**. A dialog box appears that lets you specify a folder and filename for storing the image and, in addition, the colormap and the wavelet name.

At the end of this section, turn back the extension mode to zero padding using the command

```
dwtmode('zpd')
```

1-D Extension Using the Command Line

The function `wextend` performs signal extension. For more information, see its reference page.

2-D Extension Using the Command Line

The function `wextend` performs image extension. For more information, see its reference page.

Image Fusion

For the example, switch the extension mode to symmetric padding, using the command:

```
dwtmode('sym')
```

The toolbox requires only one function for image fusion: `wfusimg`. You'll find full information about this function in its reference page. For more details on fusion methods see the `wfusmat` function.

In this section, you will learn how to:

- Load images
- Perform decompositions
- Merge images from their decompositions
- Restore images from their decompositions
- Save image after fusion

The principle of image fusion using wavelets is to merge the wavelet decompositions of the two original images using fusion methods applied to approximations coefficients and details coefficients (see [MisMOP03] and [Zee98] in “References” on page 1-105).

The two images must be of the same size and are supposed to be associated with indexed images on a common colormap (see `wextend` to resize images).

Two examples are examined: the first one merges two different images leading to a new image and the second restores an image from two fuzzy versions of an original image.

Image Fusion Using the Command Line

Example 1: Fusion of Two Different Images

1 Load two original images: a mask and a bust.

```
load mask; X1 = X;
load bust; X2 = X;
```

2 Merge the two images from wavelet decompositions at level 5 using `db2` by taking two different fusion methods: fusion by taking the mean for both approximations and details,

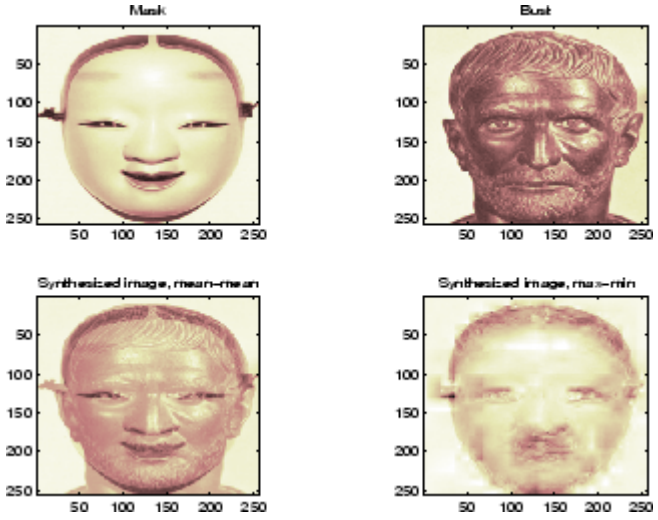
```
XFUSmean = wfusimg(X1,X2,'db2',5,'mean','mean');
```

and fusion by taking the maximum for approximations and the minimum for the details.

```
XFUSmaxmin = wfusimg(X1,X2,'db2',5,'max','min');
```

3 Plot original and synthesized images.

```
colormap(map);
subplot(221), image(X1), axis square, title('Mask')
subplot(222), image(X2), axis square, title('Bust')
subplot(223), image(XFUSmean), axis square,
title('Synthesized image, mean-mean')
subplot(224), image(XFUSmaxmin), axis square,
title('Synthesized image, max-min')
```



Example 2: Restoration by Fusion from Fuzzy Images

1. Load two fuzzy versions of an original image.

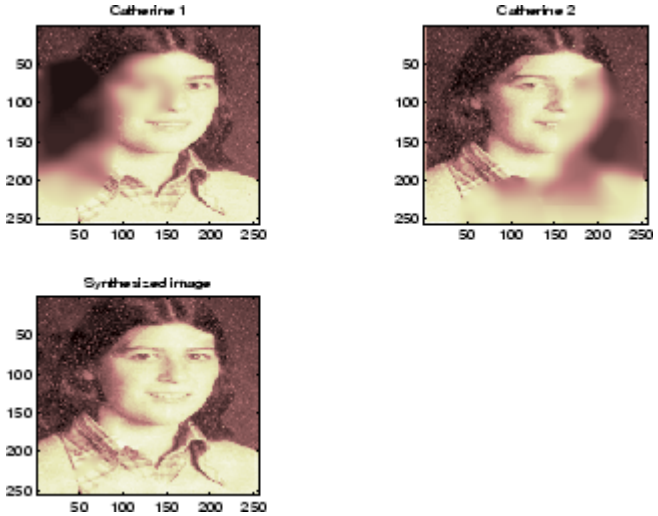
```
load cathe_1; X1 = X;
load cathe_2; X2 = X;
```

2. Merge the two images from wavelet decompositions at level 5 using `sym4` by taking the maximum of absolute value of the coefficients for both approximations and details.

```
XFUS = wfusimg(X1,X2,'sym4',5,'max','max');
```

3. Plot original and synthesized images.

```
colormap(map);
subplot(221), image(X1), axis square,
title('Catherine 1')
subplot(222), image(X2), axis square,
title('Catherine 2')
subplot(223), image(XFUS), axis square,
title('Synthesized image')
```



The synthesized image is a restored version of good quality of the common underlying original image.

Image Fusion Using the Wavelet Analyzer App

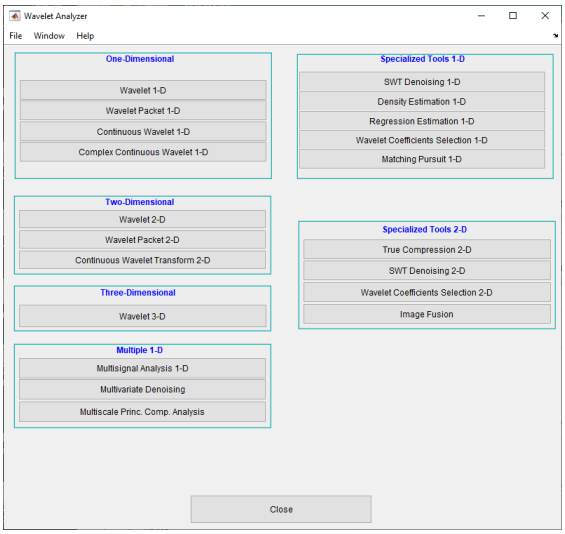
The principle of image fusion using wavelets is to merge the wavelet decompositions of the two original images using fusion methods applied to approximations coefficients and details coefficients (see [MisMOP03] and [Zee98] in “References” on page 1-105).

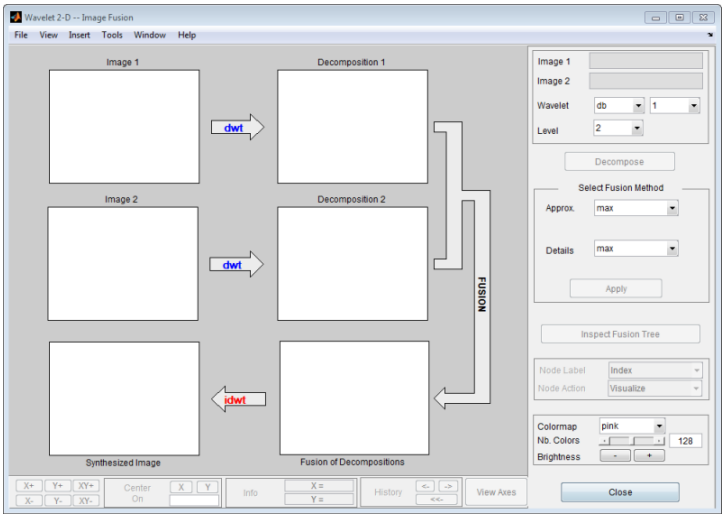
The two images must be of the same size and are supposed to be associated with indexed images on a common colormap (see `wextend` to resize images).

Two examples are examined: the first one merges two different images leading to a new image and the second restores an image from two fuzzy versions of an original image.

1 Start the **Wavelet Analyzer** App.

From the MATLAB prompt, type `waveletAnalyzer` to display the **Wavelet Analyzer** and then click the **Image Fusion** menu item to display the **Image Fusion** Tool.





2 Load the original images: a mask and a bust.

```
load mask; X1 = X;
load bust; X2 = X;
```

In the **Image Fusion** tool, select **File > Load or Import Image 1 > Import from Workspace**. When the **Import from Workspace** dialog box appears, select the `X1` variable, which loads the mask image.

Perform the same sequence choosing the `X2` variable to load the bust image.

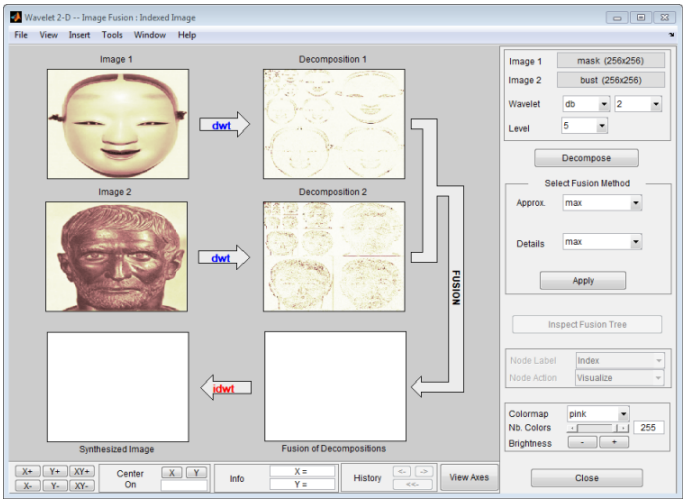
3 Perform wavelet decompositions.

Using the **Wavelet** and **Level** menus located to the upper right, determine the wavelet family, the wavelet type, and the number of levels to be used for the analysis.

For this analysis, select the `db2` wavelet at level 5.

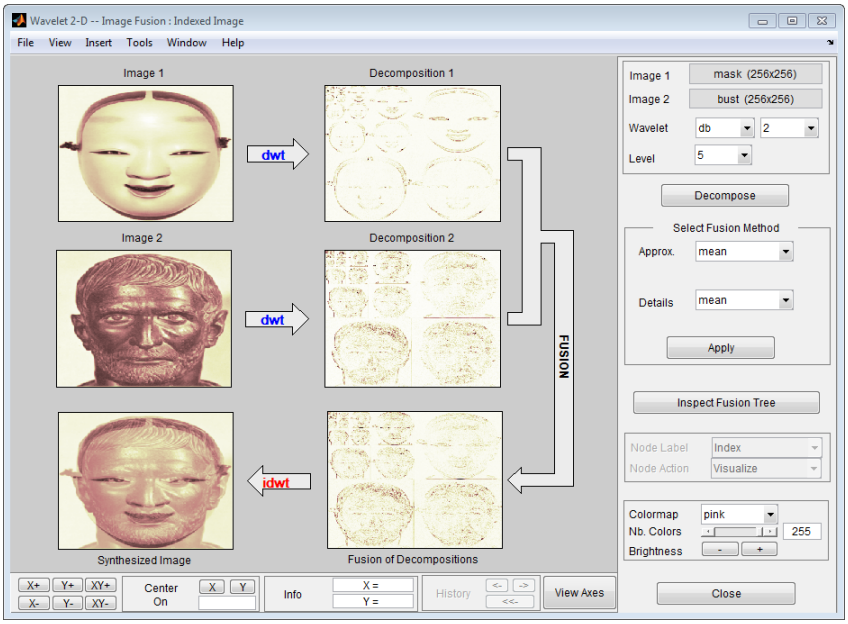
Click the **Decompose** button.

After a pause for computation, the tool displays the two analyses.



4 Merge two images from their decompositions.

From **Select Fusion Method** frame, select the item `mean` for both **Approx.** and **Details**. Next, click the **Apply** button.



The synthesized image and its decomposition (which is equal to the fusion of the two decompositions) appear. The new image produced by fusion clearly exhibits features from the two original ones.

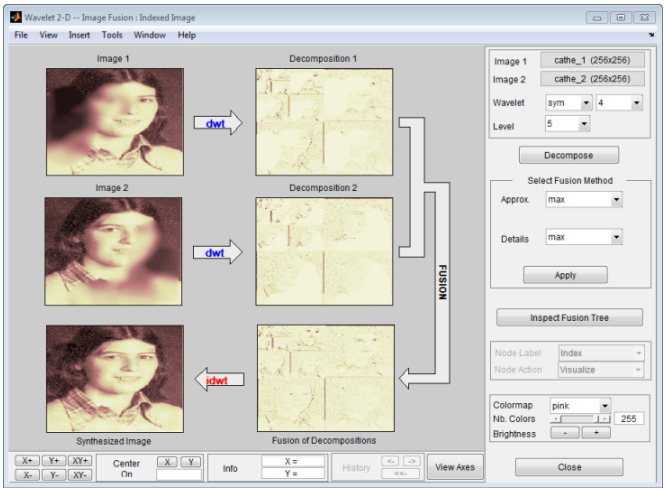
Let us now examine another example illustrating restoration using image fusion.

5 Restore the image using image fusion.

From the **File** menu, load Image 1 by selecting the MAT-file `cathe_1.mat`, and Image 2 by selecting the MAT-file `cathe_2.mat`.

6 Using the **Wavelet** and **Level** menus, select the `sym4` wavelet at level 5. Click the **Decompose** button.

7 From **Select Fusion Method** frame, select the item `max` for both **Approx.** and **Details**. Next, click the **Apply** button.



The synthesized image is a restored version of good quality of the common underlying original image.

Saving the Synthesized Image

The Image Fusion Tool lets you save the synthesized image to disk. The toolbox creates a MAT-file in the current folder with a name you choose.

To save the synthesized image from the present selection, use the menu option **File > Save Synthesized Image**.

A dialog box appears that lets you specify a folder and filename for storing the image. After you save the image data to the file `rescathe.mat`, the synthesized image is given by `X` and the colormap by `map`.

1-D Fractional Brownian Motion Synthesis

This example shows how to generate a fractional Brownian motion signal using the `wfbm` function.

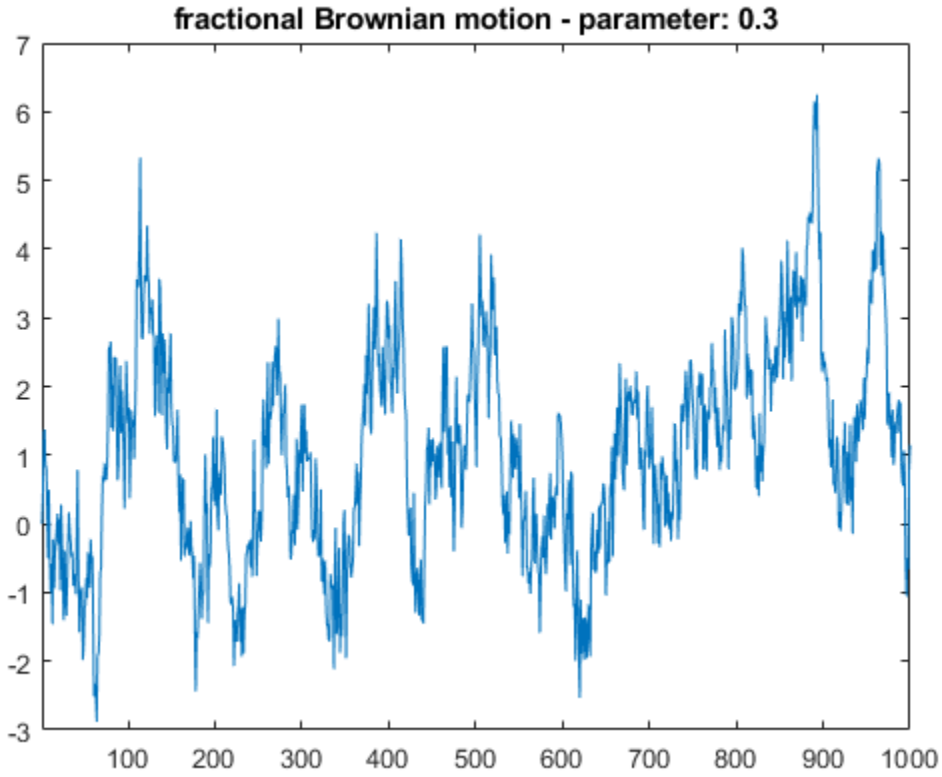
A fractional Brownian motion (`fBm`) is a continuous-time Gaussian process depending on the Hurst parameter $0 < H < 1$. It generalizes the ordinary Brownian motion corresponding to $H = 0.5$ and whose derivative is white noise. The `fBm` is self-similar in distribution and the variance of the increments is given by

```
Var(fBm(t) - fBm(s)) = v abs(t-s)^(2H),
```

where v is a positive constant. According to the value of `H`, the `fBm` exhibits for $H < 0.5$, long-range dependence, and for $H < 0.5$, short or intermediate dependence.

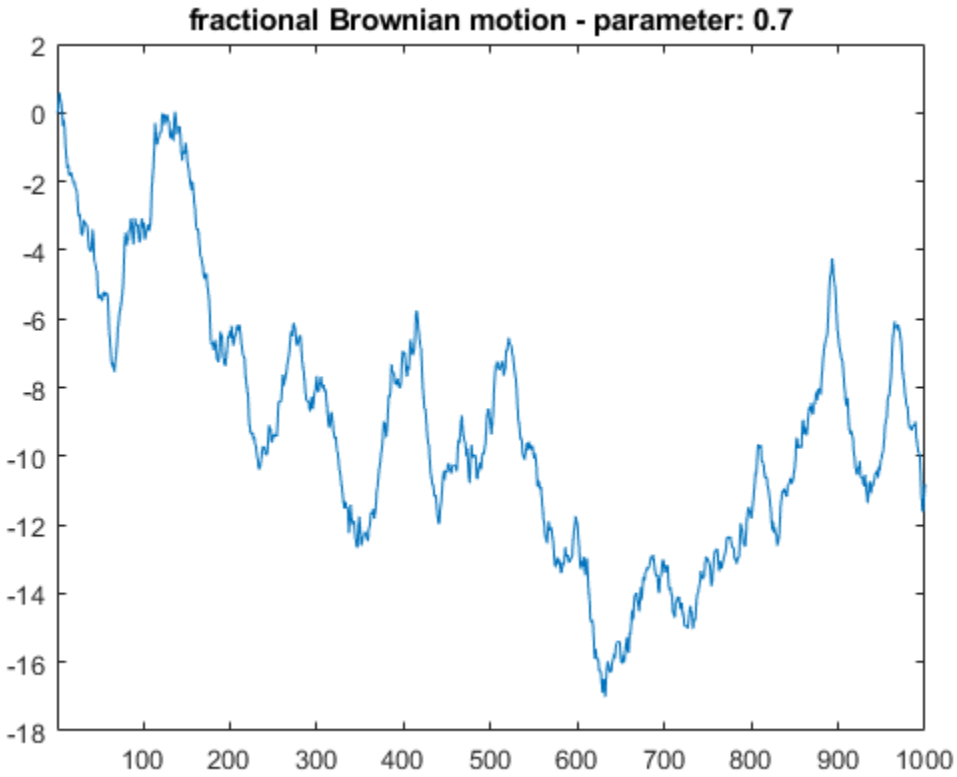
For purposes of reproducibility, set the random seed to the default value. Generate fractional Brownian motion with length 1000 for $H = 0.3$. Plot the result.

```
rng default
H = 0.3;
len = 1000;
fBm03 = wfbm(H,len,'plot');
```



Generate fractional Brownian motion with length 1000 for $H = 0.7$. Plot the result. Because $H > 0.5$, the fractional Brownian motion exhibits a stronger low-frequency component and has, locally, less irregular behavior.

```
rng default
H = 0.7;
fBm07 = wfbm(H,len,'plot');
```



Confirm the previous syntax is equivalent to generating fractional Brownian motion using the orthogonal `db10` wavelet and six reconstruction steps.

```
rng default
w = 'db10';
ns = 6;
fBm07x = wfbm(H,len,w,ns);
max(abs(fBm07-fBm07x))
```

```
ans = 0
```

New Wavelet for CWT

This example illustrates how to generate a new wavelet starting from a pattern.

The principle for designing a new wavelet for CWT is to approximate a given pattern using least squares optimization under constraints leading to an admissible wavelet well suited for the pattern detection using the continuous wavelet transform [1].

Load an original pattern: a pseudo sine.

```
load ptpssin1
who
```

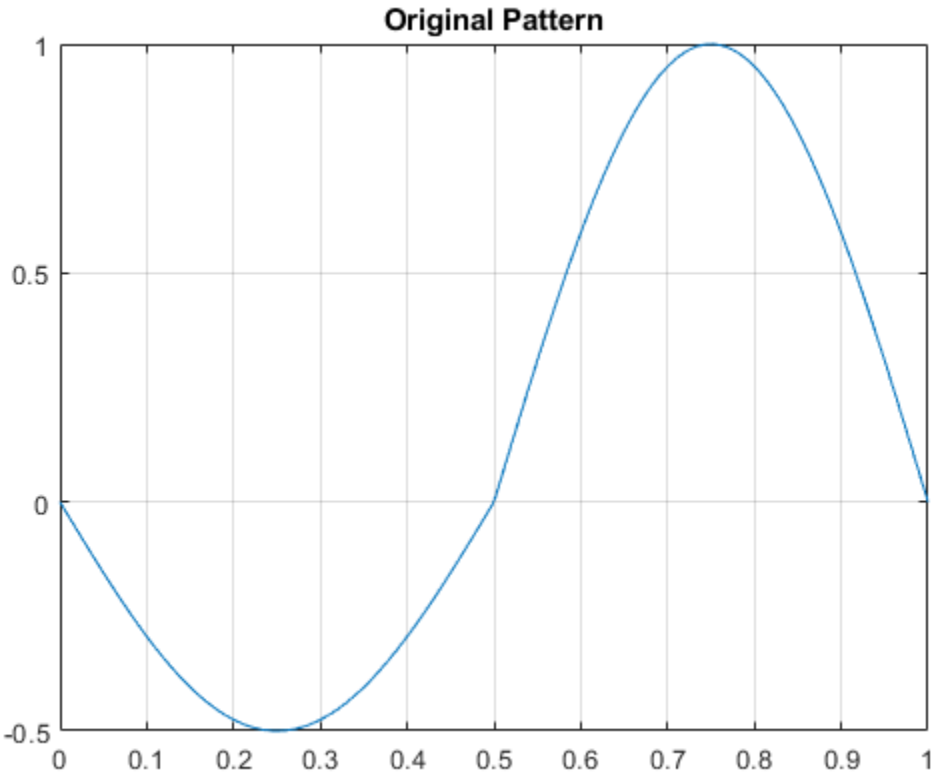
```
Your variables are:

IntVAL   X        Y        caption
```

The variables X and Y contain the pattern. Integrate the pattern over the interval [0, 1]. Plot the pattern.

```
dX = max(diff(X));
patternInt = dX*sum(Y);
disp(['Integral of pattern = ',num2str(patternInt)]);
```

```
Integral of pattern = 0.15915
```

```
plot(X,Y)
title('Original Pattern')
grid on
```



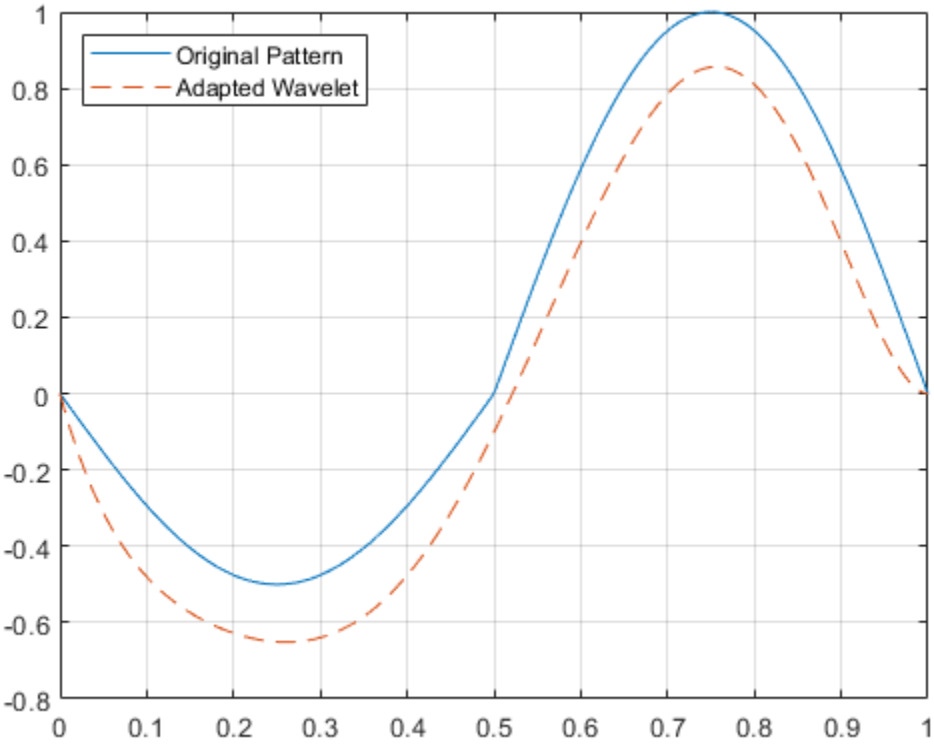
The pattern on the interval [0, 1] integrates to 0.15915. So it is not a wavelet but it is a good candidate since it oscillates like a wavelet.

To synthesize a new wavelet adapted to the given pattern, use a least squares polynomial approximation of degree 6 with constraints of continuity at the beginning and the end of the pattern.

```
[psi,xval,nc] = pat2cwav(Y, 'polynomial',6, 'continuous');
```

The new wavelet is given by `xval` and `nc*psi`.

```
figure
plot(X,Y,'-',xval,nc*psi,'--')
grid on
legend('Original Pattern','Adapted Wavelet','Location','NorthWest')
```



Check that `psi` satisfies the definition of a wavelet by confirming that it integrates to zero and has L_2 norm is equal to 1.

```
dxval = max(diff(xval));
newWaveletIntegral = dxval*sum(psi);
disp(['Integral of new wavelet = ',num2str(newWaveletIntegral)])
```

```
Integral of new wavelet = 1.9626e-05
```

```
newWaveletSqN = dxval*sum(psi.^2);
disp(['New wavelet has L2-norm = ',num2str(newWaveletSqN)])
```

```
New wavelet has L2-norm = 1
```

References

[1] Misiti, M., Y. Misiti, G. Oppenheim, and J.-M. Poggi. *Les ondelettes et leurs applications*. France: Hermes Science/Lavoisier, 2003.

See Also

`pat2cwav`

Additional Wavelet GUIs

1-D Extension Using the Signal Extension Tool

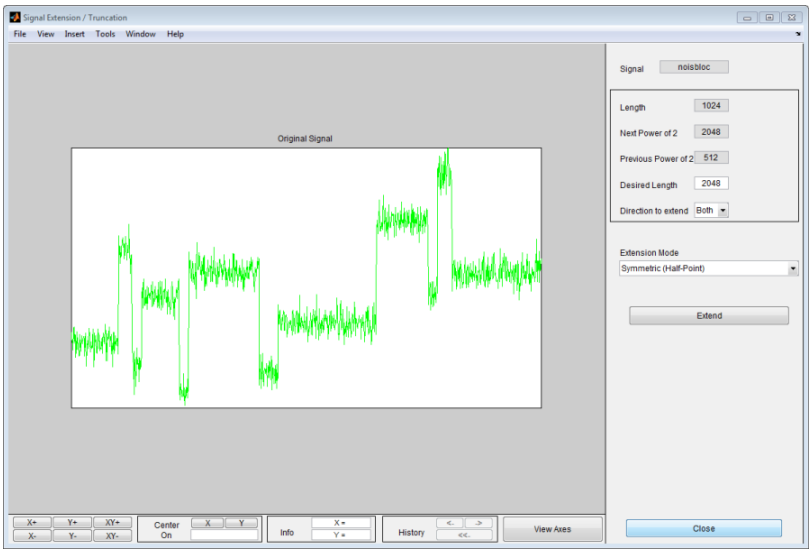
1. Load data. At the MATLAB command prompt, type

   ```
   load noisbloc;
   ```

2. Start the **Signal Extension** Tool.

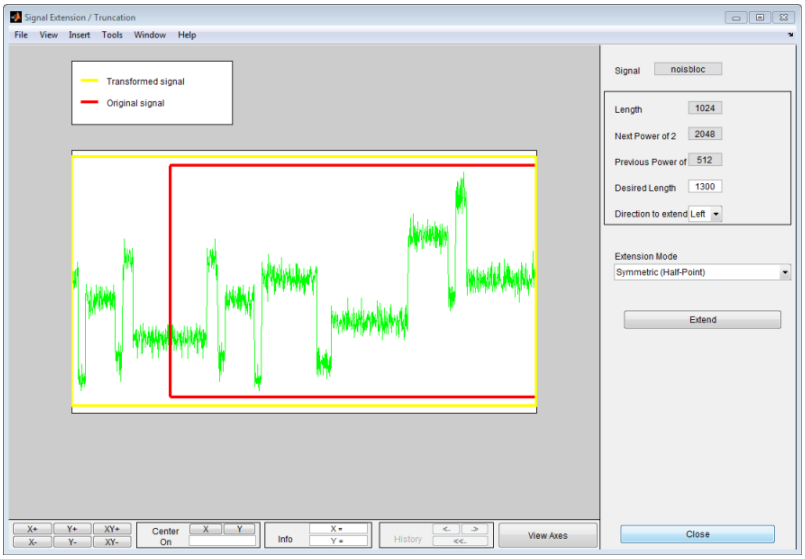
   From the MATLAB prompt, type `sigxtool`. The **Signal Extension** tool appears.

3. In the **Signal Extension** tool, select **File > Import from Workspace**. When the **Import from Workspace** dialog box appears, select the `noisbloc` variable. Click **OK** to import the data



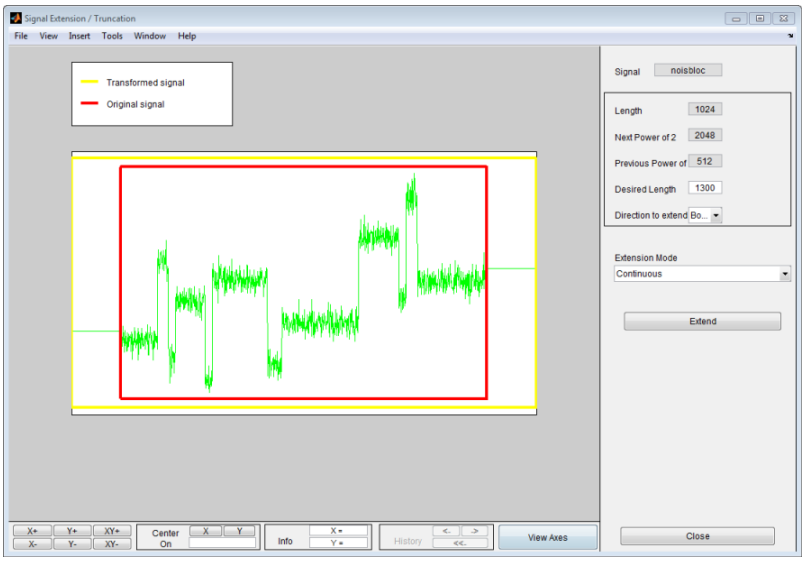
4. Extend the signal.

   Enter `1300` in the **Desired Length** box of the extended signal, and select the **Left** option from the **Direction to extend** menu. Then accept the default **Symmetric** for the **Extension mode**, and click the **Extend** button.



The tool displays the original signal delimited by a red box and the transformed signal delimited by a yellow box. The signal has been extended by left symmetric boundary values replication.

Select the **Both** option from the **Direction to extend** menu and select the **Continuous** option from the **Extension mode** menu. Click the **Extend** button.



The signal is extended in both directions by replicating the first value to the left and the last value to the right, respectively.

Extending Signal for SWT

Since the decomposition at level k of a signal using SWT requires that 2^k divides evenly into the length of the signal, the tool provides a special option dedicated to this kind of extension.

Select the **For SWT** option from the **Extension mode** menu. Click the **Extend** button.

This tool displays the pattern detection performed with the adapted wavelet on the left and `db1` wavelet (default) on the right. The two positions are perfectly detected in both cases but scales are slightly underestimated by the `db1` wavelet.

The tool allows you to generate various running signals and choose the wavelet to be compared with the adapted one.

Click the **Close** button to get back to the main window.

Saving the New Wavelet

The New Wavelet for CWT Tool lets you save the synthesized wavelet. The toolbox creates a MAT-file in the current folder with a name you choose.

To save the new wavelet from the present selection, use the option **File > Save Adapted Wavelet**. A dialog box appears that lets you specify a folder and filename for storing the data. After you save the wavelet data to the file `newwavel.mat`, the adapted wavelet is given by `X` and `Y`.

Note that the version of the saved wavelet is correctly defined to be used in the CWT algorithm and is such that its square norm is equal to 1.

See Also

Signal Multiresolution Analyzer | Wavelet Signal Denoiser

Getting Started with Wavelet Analysis

- “Wavelet Families and Properties” on page 3-2
- “Visualizing Wavelets, Wavelet Packets, and Wavelet Filters” on page 3-4
- “Continuous Wavelet Analysis” on page 3-7
- “Continuous Wavelet Transform and Inverse Continuous Wavelet Transform” on page 3-9
- “Discrete Wavelet Analysis” on page 3-13
- “Lifting” on page 3-20
- “Critically Sampled Wavelet Packet Analysis” on page 3-25

Wavelet Families and Properties

This example shows how to find and display information about available wavelets. The Wavelet Toolbox software contains an extensive selection of the most commonly-used wavelets and orthogonal and biorthogonal wavelet filters. You also have the ability to add your own filters to the toolbox.

Determine the existing wavelet families. Display the wavelet family names in the command window.

```
waveletfamilies('f')
```

Display the names of all available wavelets in each family.

```
waveletfamilies('a')
```

You can also use `wavemngr` to display the available wavelet families.

```
wavemngr('read')
```

Use the wavelet family short name to determine what analysis an existing wavelet supports.

The wavelet family short name for the Daubechies extremal-phase wavelets is `'db'`.

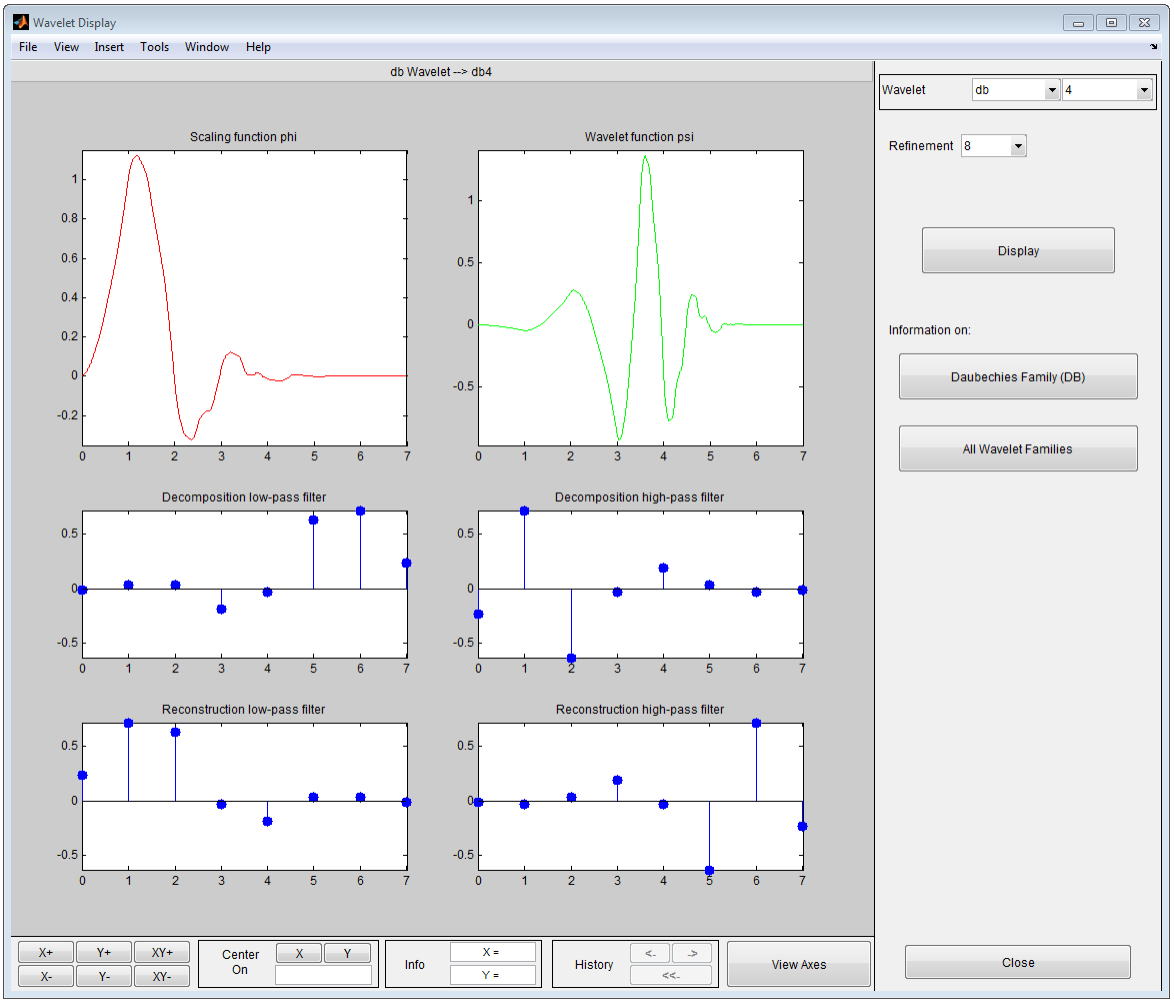
```
waveinfo('db')
```

Determine what analysis the Morlet wavelet supports. The wavelet family short name is `'morl'`.

```
waveinfo('morl')
```

Use the Wavelet Toolbox **Wavelet Display** tool to investigate wavelet families. To start the interactive tool, enter `wvdtool` at the command line.

Select the db4 wavelet and click **Display**.



See Also

More About

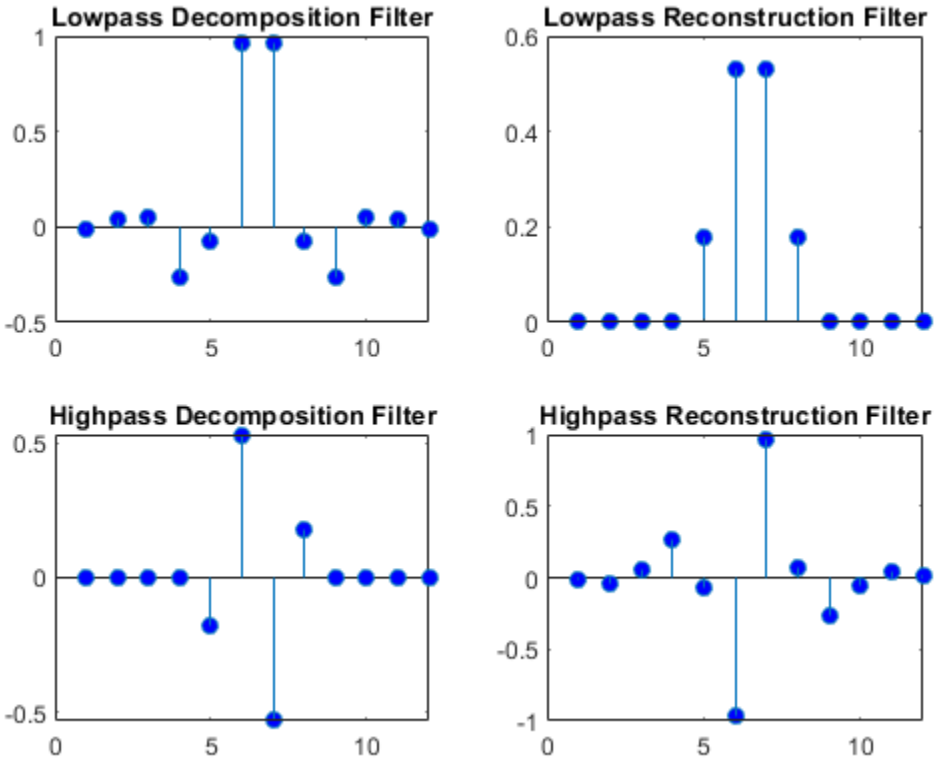
- “Visualizing Wavelets, Wavelet Packets, and Wavelet Filters” on page 3-4

Visualizing Wavelets, Wavelet Packets, and Wavelet Filters

This example shows how to use `wfilters`, `wavefun`, and `wpfun` to obtain the filters, wavelet, or wavelet packets corresponding to a particular wavelet family. You can visualize 2-D separable wavelets with `wavefun2`.

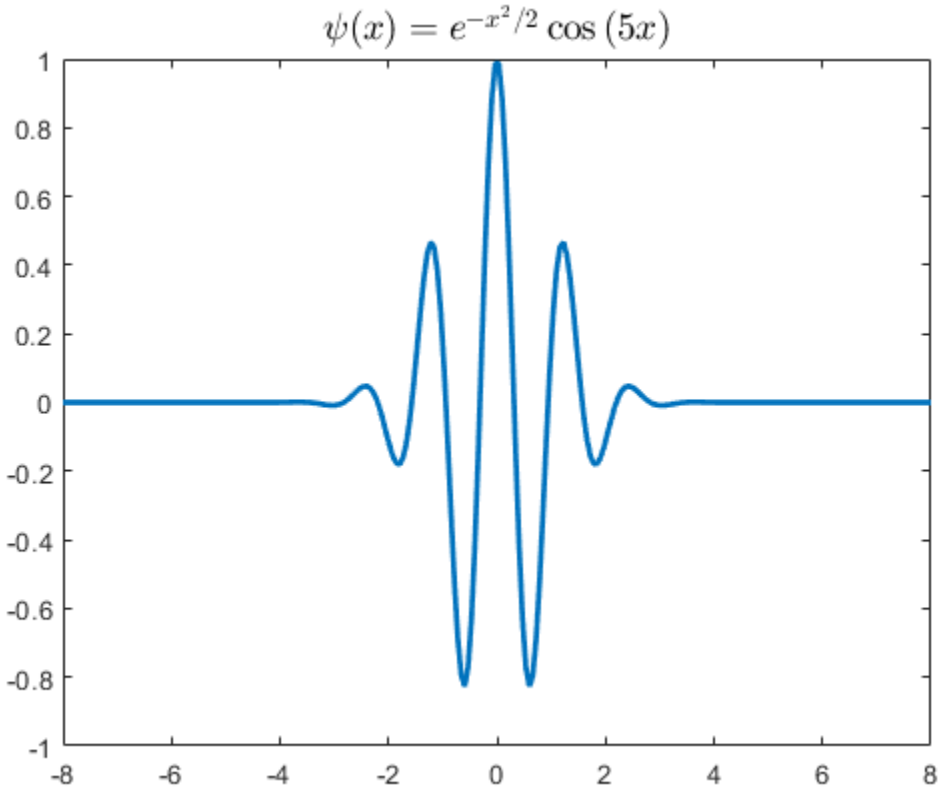
Obtain the decomposition (analysis) and reconstruction (synthesis) filters for the biorthogonal spline wavelet filters with 3 vanishing moments in the reconstruction filter and 5 vanishing moments in the decomposition filter.

```
[LoD,HiD,LoR,HiR] = wfilters('bior3.5');
subplot(2,2,1)
stem(LoD,'markerfacecolor',[0 0 1]); title('Lowpass Decomposition Filter');
subplot(2,2,2)
stem(LoR,'markerfacecolor',[0 0 1]); title('Lowpass Reconstruction Filter');
subplot(2,2,3)
stem(HiD,'markerfacecolor',[0 0 1]); title('Highpass Decomposition Filter');
subplot(2,2,4)
stem(HiR,'markerfacecolor',[0 0 1]); title('Highpass Reconstruction Filter');
```



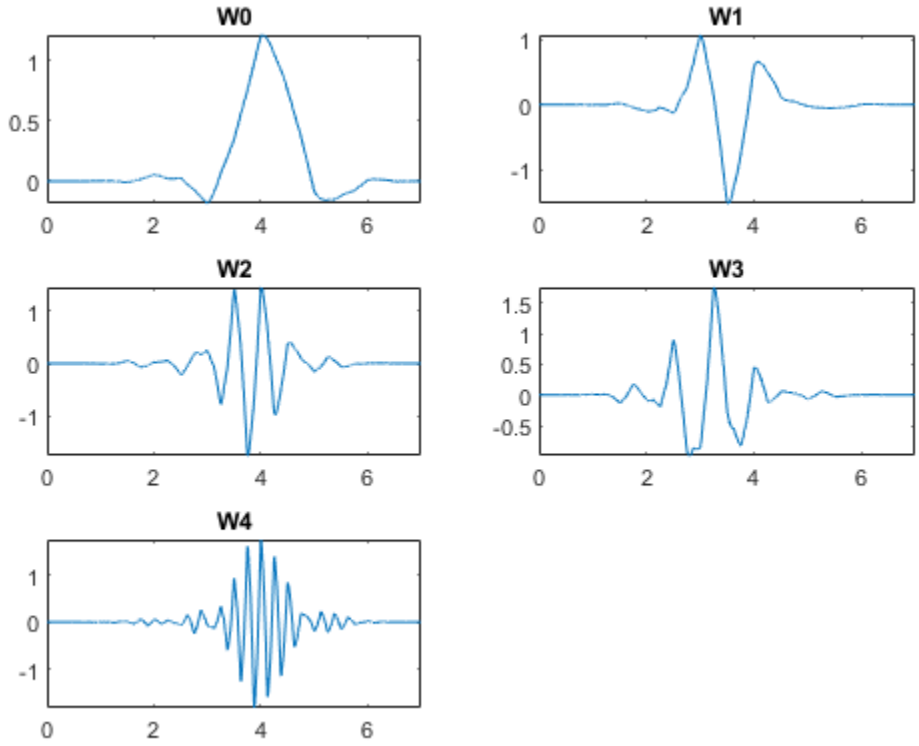
Visualize the real-valued Morlet wavelet. There is no associated scaling function.

```
figure
[psi,xval] = wavefun('morl');
plot(xval,psi,'linewidth',2)
title('$\psi(x) = e^{-x^2/2} \cos{(5x)}$','Interpreter','latex',...
     'fontsize',14);
```



Obtain the first 4 wavelet packets for the Daubechies least-asymmetric wavelet with 4 vanishing moments, `sym4`.

```
[wpws,x] = wpfun('sym4',4,10);
for nn = 1:size(wpws,1)
    subplot(3,2,nn)
    plot(x,wpws(nn,:))
    axis tight
    title(['W',num2str(nn-1)]);
end
```



See Also

`cwtfilterbank` | `dwtfilterbank` | `wavefun` | `wpfun`

Continuous Wavelet Analysis

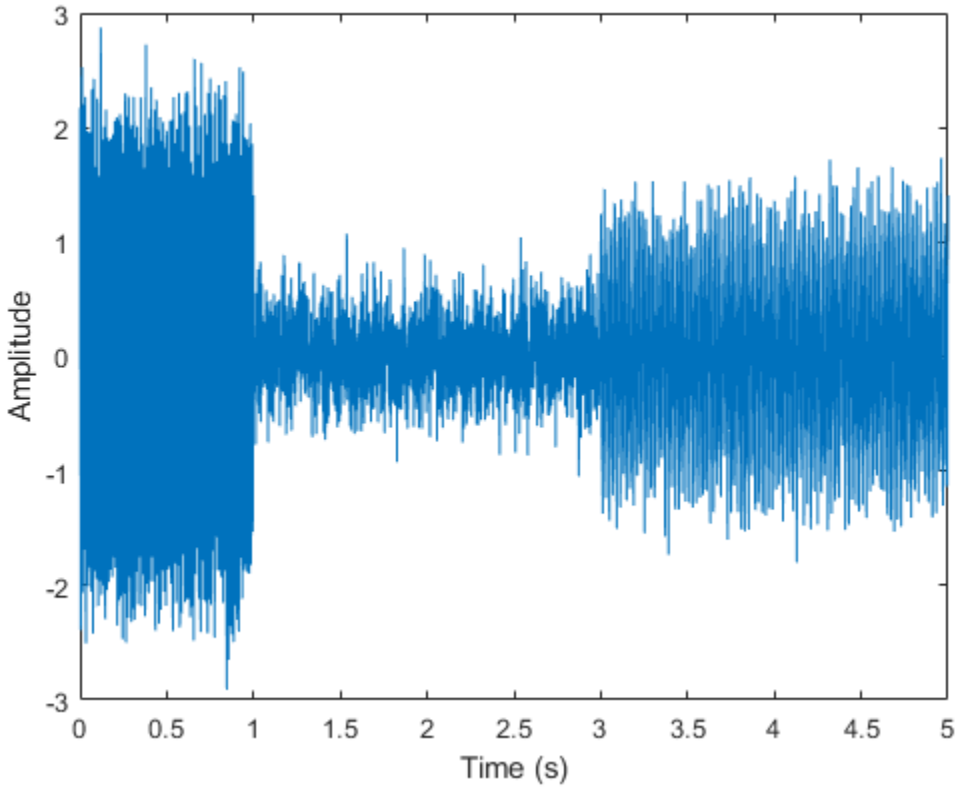
This example shows how to perform time-frequency analysis using the continuous wavelet transform (CWT). Continuous wavelet analysis provides a time-scale/time-frequency analysis of signals and images. The Wavelet Toolbox™ software has both command line and interactive functionality to support continuous wavelet analysis of 1-D signals.

Construct a signal consisting of two sinusoids with frequencies of 100 and 50 Hz, and white noise. The support of the two sinusoids is disjoint. The 100-Hz sine wave begins at t = 0 and has a duration of 1 second. The 100-Hz sinusoid has an amplitude of 2. The 50-Hz sinusoid begins at three seconds and has a duration of two seconds. The 50-Hz sinusoid has an amplitude of 1. The sampling frequency is 1 kHz. The signal length is 5000 samples.

```
Fs = 1000;
t = linspace(0,5,5e3);
x = 2*cos(2*pi*100*t).*(t<1)+cos(2*pi*50*t).*(3<t)+0.3*randn(size(t));
```

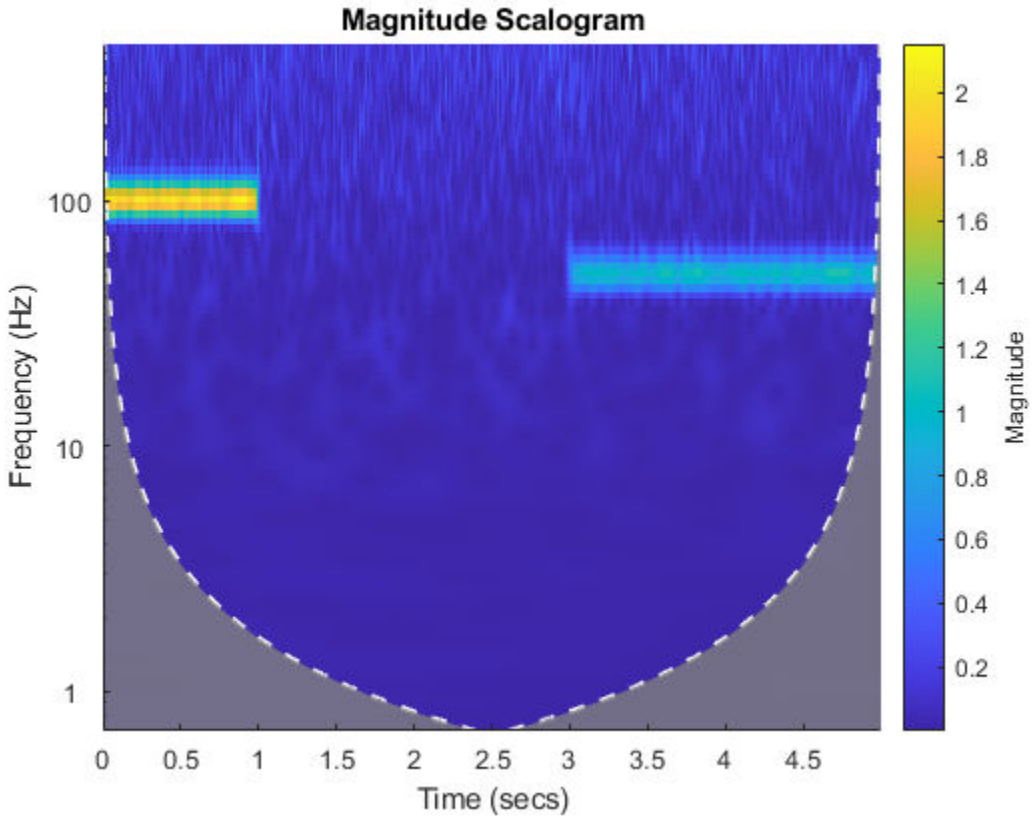
Plot the signal.

```
plot(t,x)
xlabel('Time (s)')
ylabel('Amplitude')
```



Use `cwt` to obtain the CWT of the signal and plot its scalogram. The magnitudes of the sinusoid components in the colorbar are essentially their amplitudes even though they are at different scales.

```
cwt(x,Fs)
```



See Also

`cwt` | `cwtfilterbank`

More About

- “CWT-Based Time-Frequency Analysis”

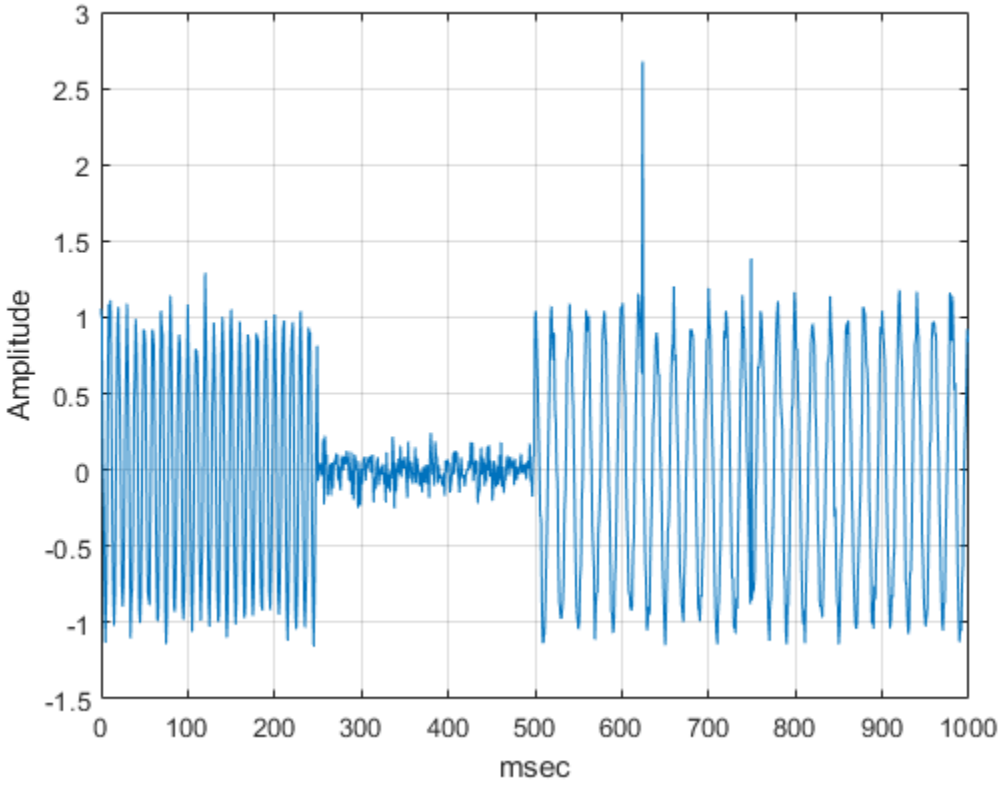
Continuous Wavelet Transform and Inverse Continuous Wavelet Transform

This example shows how to use the continuous wavelet transform (CWT) and inverse CWT.

CWT of Sine Waves and Impulses

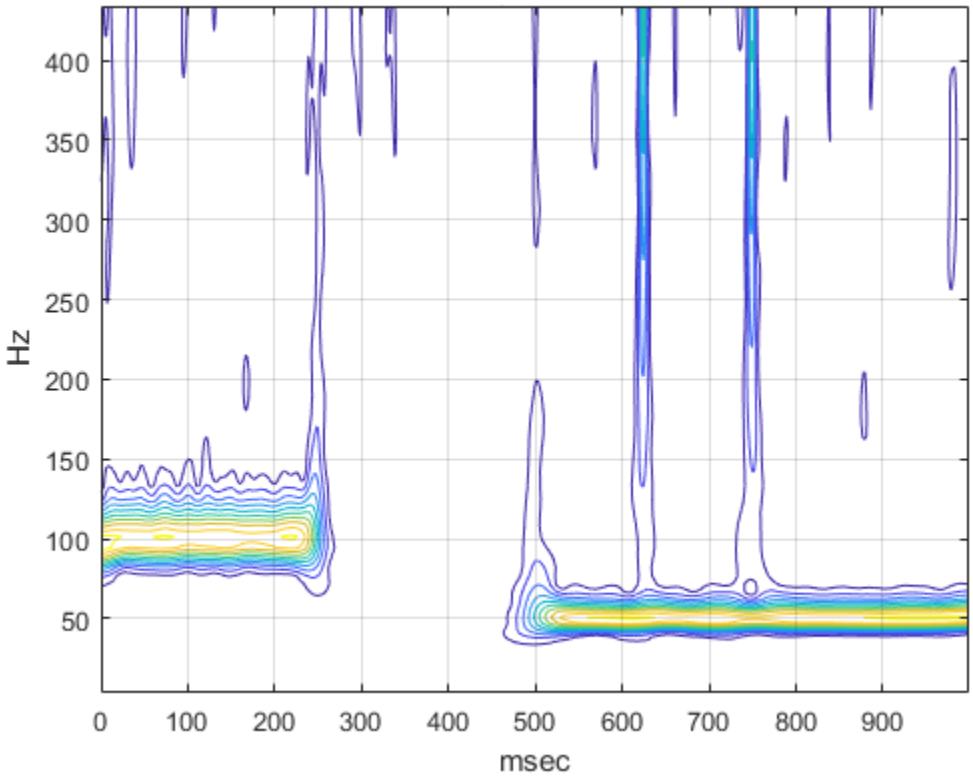
Create and plot a signal consisting of two disjoint sine waves with frequencies of 100 and 50 Hz punctuated by two impulses. The sampling frequency is 1 kHz and the total signal duration is one second. The 100-Hz sine wave occurs over the first 250 milliseconds of the data. The 50-Hz sinusoid occurs over the last 500 milliseconds. The impulses occur at 650 and 750 milliseconds. The signal also has $N(0, 0.1^2)$ additive white Gaussian noise. The impulse at 650 milliseconds is visible, but the impulse at 750 milliseconds is not clearly evident in the time-domain data.

```
Fs = 1000;
t = 0:1/Fs:1-1/Fs;
x = zeros(size(t));
x([625,750]) = 2.5;
x = x+ cos(2*pi*100*t).*(t<0.25)+cos(2*pi*50*t).*(t>=0.5)+...
    0.1*randn(size(t));
plot(t.*1000,x)
grid on;
xlabel('msec'); ylabel('Amplitude');
```



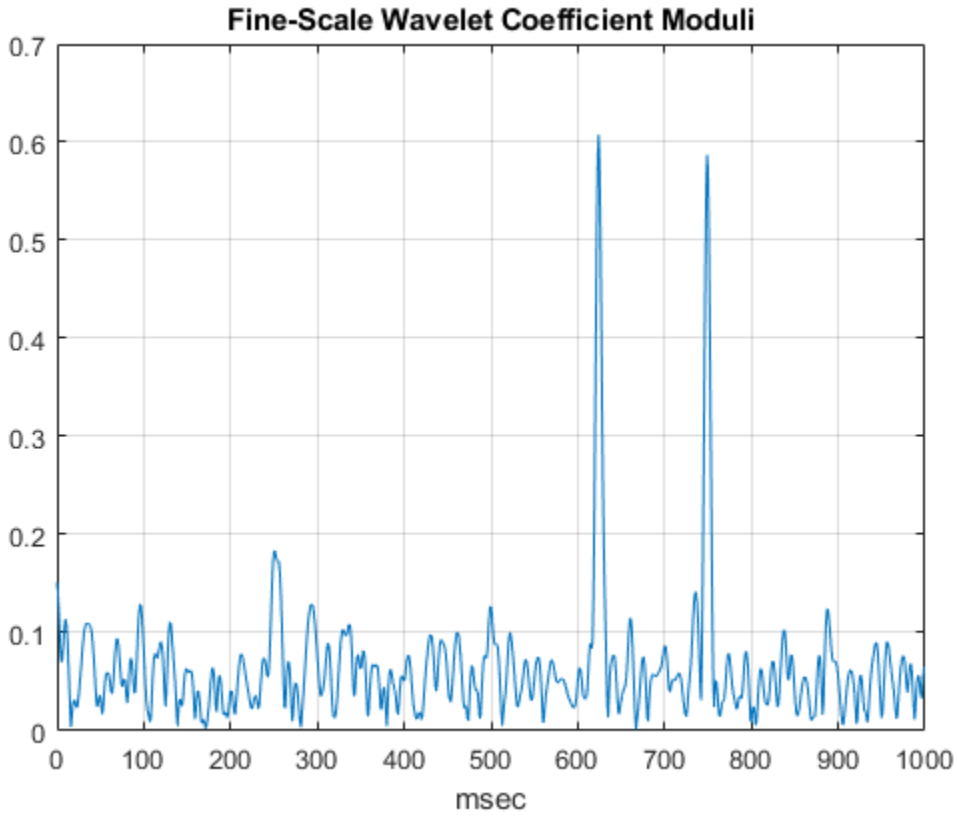
Obtain and plot the CWT using the default analytic Morse wavelet.

```
[cfs,f] = cwt(x,1000);
contour(t.*1000,f,abs(cfs));
xlabel('msec'); ylabel('Hz');
grid on;
```



The CWT moduli correctly show the supports of the disjoint sinusoids and the locations of the impulses at 650 and 750 milliseconds. In the CWT moduli, the impulse at 750 milliseconds is clearly visible. This is especially true if you plot just the finest-scale wavelet coefficients.

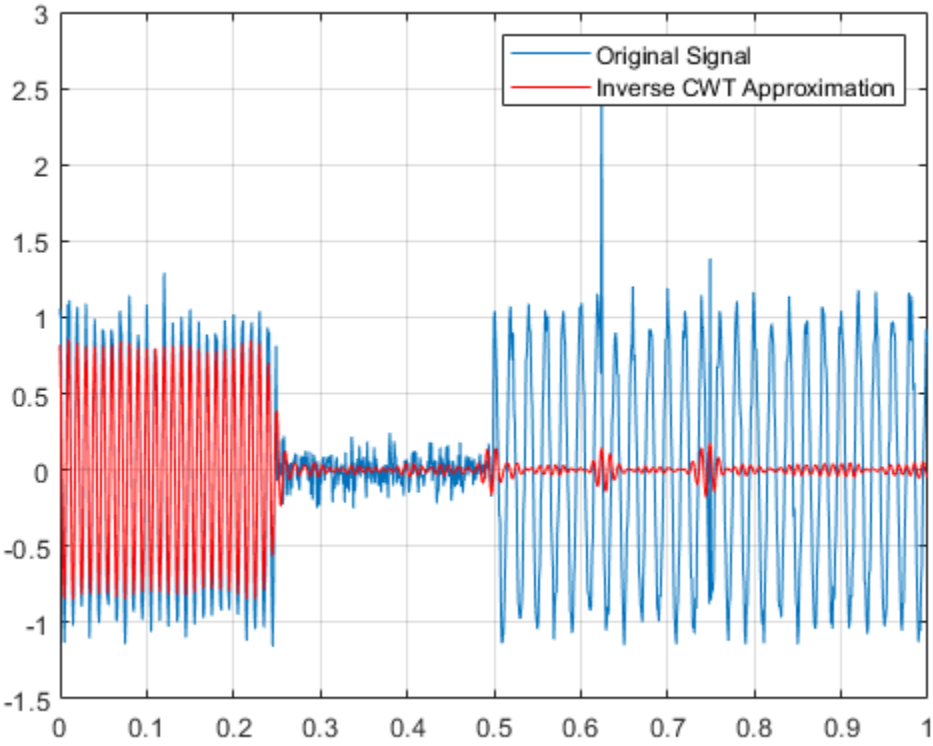
```
plot(t.*1000,abs(cfs(1,:)))
grid on
title('Fine-Scale Wavelet Coefficient Moduli')
xlabel('msec')
```



Frequency-Localized Inverse CWT

Using the inverse CWT you can construct frequency-localized approximations to events in your time series. Use the inverse CWT to obtain an approximation to the 100-Hz sinusoid in the previous example.

```
xrec = icwt(cfs,f,[90 110]);
plot(t,x);
hold on;
plot(t,xrec,'r');
legend('Original Signal','Inverse CWT Approximation',...
    'Location','NorthEast');
grid on;
```



If you zoom in on the plot, you see the 100-Hz component is well approximated but the 50-Hz component has been removed.

Discrete Wavelet Analysis

Wavelet Toolbox software enables you to analyze signals, images, and 3-D data using orthogonal and biorthogonal critically-sampled discrete wavelet analysis. Critically-sampled discrete wavelet analysis is also known as *decimated* discrete wavelet analysis. Decimated discrete wavelet analysis is most appropriate for data compression, denoising, and the sparse representation of certain classes of signals and images.

In decimated discrete wavelet analysis, the scales and translations are dyadic.

You can perform 1-D, 2-D, and 3-D decimated discrete wavelet analysis using the interactive tool by entering **`waveletAnalyzer`** at the command line and clicking **Wavelet 1-D**, **Wavelet 2-D**, or **Wavelet 3-D**.

1-D Wavelet Denoising

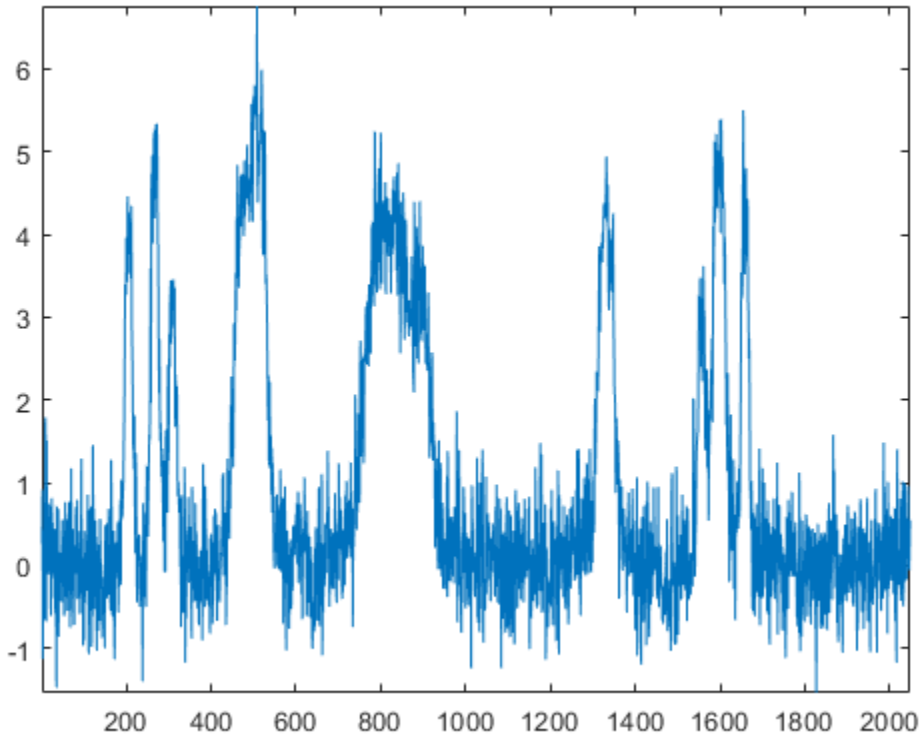
This example shows how to denoise a signal using discrete wavelet analysis.

Create a reference signal.

```
len = 2^11;
h = [4  -5  3  -4  5  -4.2   2.1   4.3  -3.1   5.1  -4.2];
t = [0.1  0.13  0.15  0.23  0.25  0.40  0.44  0.65  0.76  0.78  0.81];
h  = abs(h);
w  = 0.01*[0.5 0.5 0.6 1 1 3 1 1 0.5 0.8 0.5];
tt = linspace(0,1,len);
xref = zeros(1,len);
for j=1:11
    xref = xref+(h(j)./(1+((tt-t(j))/w(j)).^4));
end
```

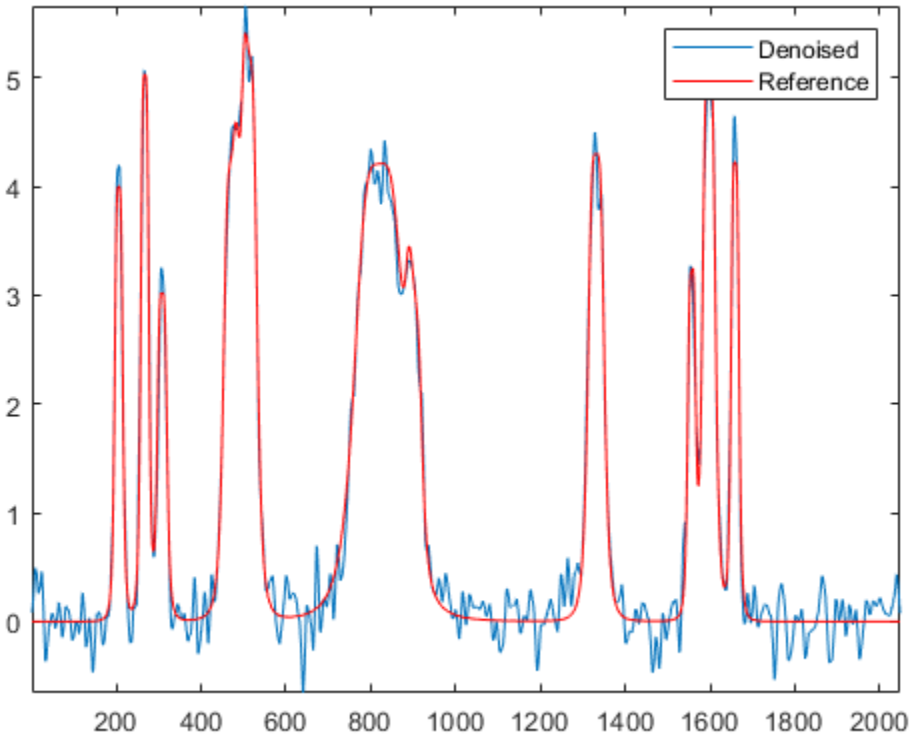
Add zero-mean white Gaussian noise with a variance of 0.25.

```
rng default
x = xref + 0.5*randn(size(xref));
plot(x)
axis tight
```



Denoise the signal down to level 3 using the Daubechies least asymmetric wavelet with 4 vanishing moments. Use the universal threshold selection rule of Donoho and Johnstone with soft thresholding based on the DWT coefficients at level 1. Use the periodization signal extension mode — `dwtmode('per')`. Plot the result along with the reference signal for comparison.

```
dwtmode('per');


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  WARNING: Change DWT Extension Mode  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

****************************************
**  DWT Extension Mode: Periodization  **
****************************************


xd = wdenoise(x,3,'Wavelet','sym4',...
    'DenoisingMethod','UniversalThreshold','NoiseEstimate','LevelIndependent');
plot(xd)
axis tight
hold on
plot(xref,'r')
legend('Denoised','Reference')
```

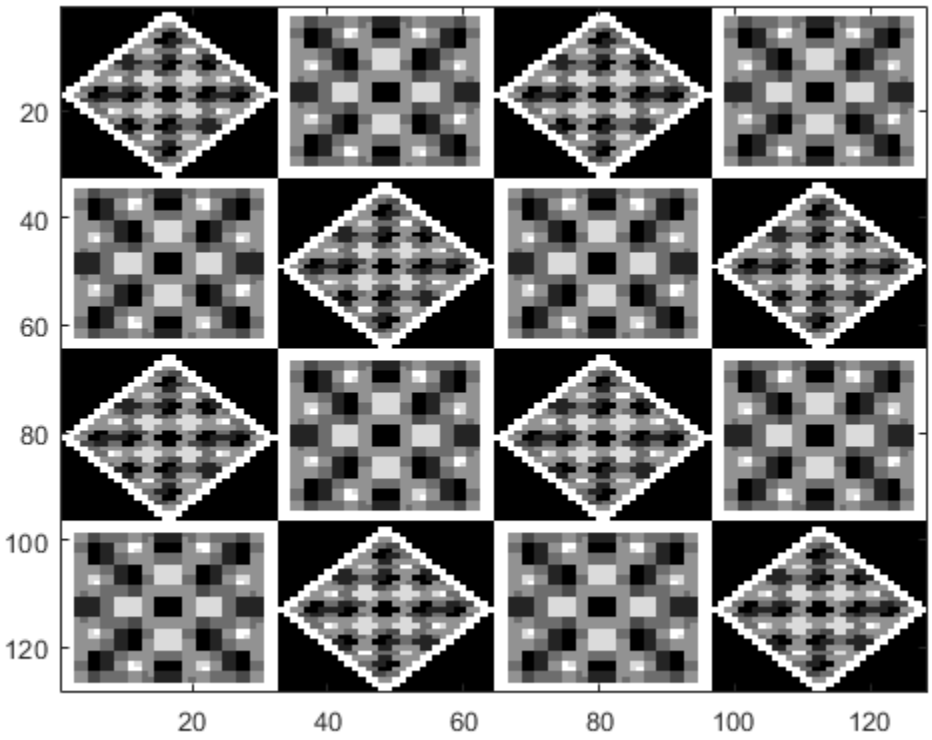


2-D Decimated Discrete Wavelet Analysis

This example shows how to obtain the 2-D DWT of an input image.

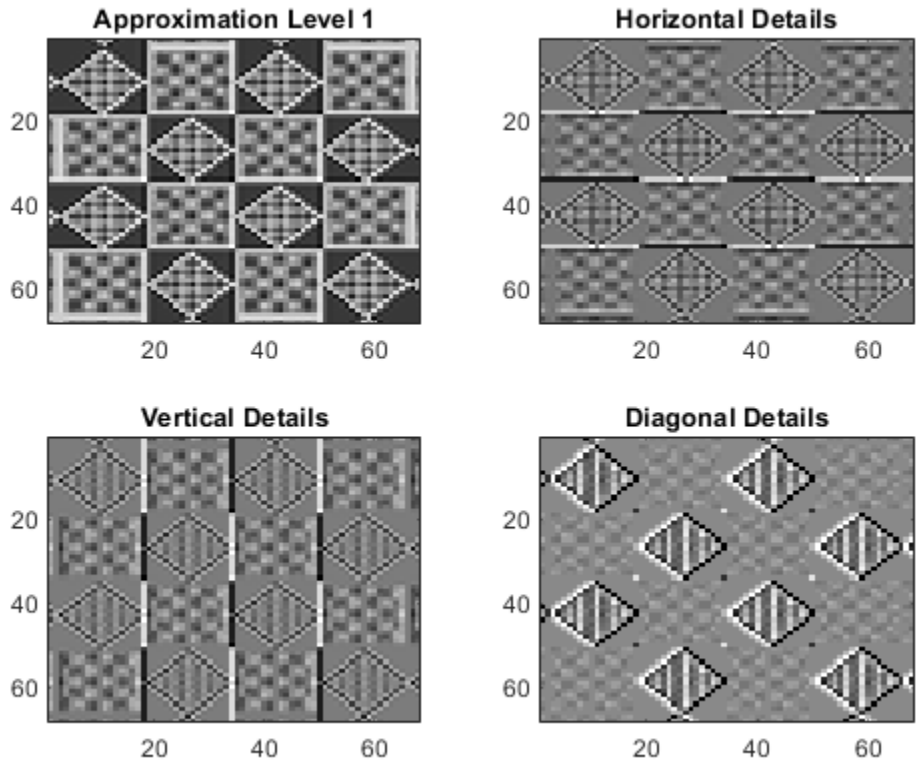
Load and display the image. The image consists of vertical, horizontal, and diagonal patterns.

```
load tartan;
imagesc(X); colormap(gray);
```



Obtain the 2-D DWT at level 1 using the biorthogonal B-spline wavelet and scaling filters with 2 vanishing moments in the analysis filters and 4 vanishing moments in the synthesis filters. Extract the horizontal, vertical, and diagonal wavelet coefficients and the approximation coefficients. Display the results.

```
[C,S] = wavedec2(X,1,'bior2.4');
[H,V,D] = detcoef2('all',C,S,1);
A = appcoef2(C,S,'bior2.4');
subplot(221);
imagesc(A); title('Approximation Level 1');
colormap(gray);
subplot(222);
imagesc(H); title('Horizontal Details');
subplot(223);
imagesc(V); title('Vertical Details');
subplot(224);
imagesc(D); title('Diagonal Details');
```



You see that the wavelet details are sensitive to particular orientations in the input image. The approximation coefficients are a lowpass approximation to the original image.

Nondecimated Discrete Wavelet Analysis

This example shows how to obtain the nondecimated (stationary) wavelet transform of a noisy frequency-modulated signal.

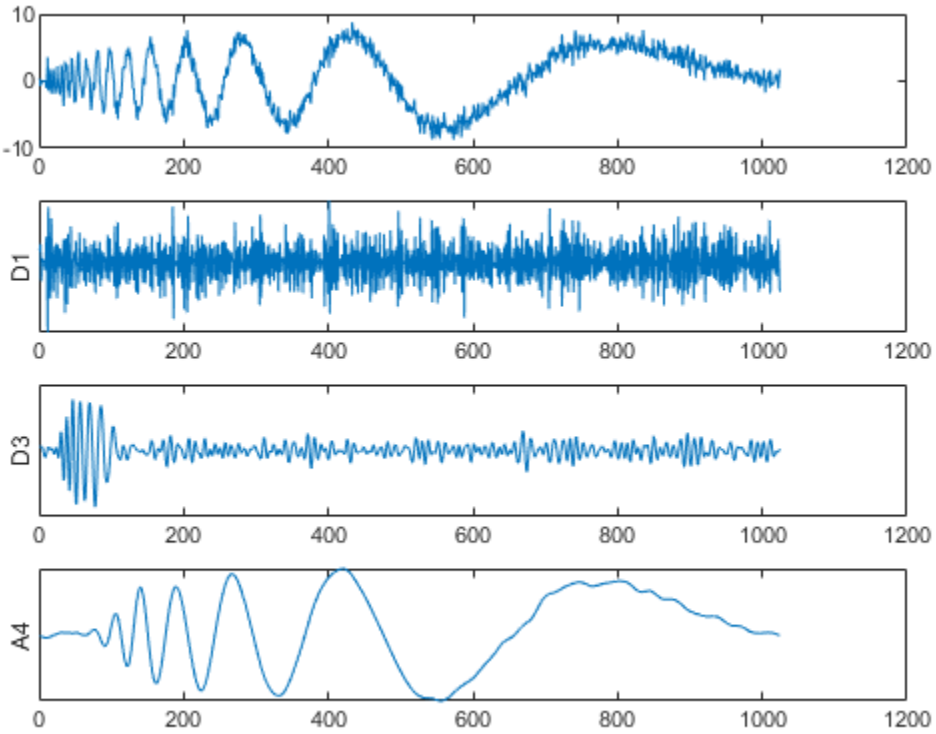
Load the noisy Doppler signal and obtain the stationary wavelet transform down to level 4.

```
load noisdopp
swc = swt(noisdopp,4,'sym8');
```

Plot the original signal and the level 1 and 3 wavelet coefficients. Plot the level 4 approximation.

```
subplot(4,1,1)
plot(noisdopp)
subplot(4,1,2)
plot(swc(1,:))
ylabel('D1')
set(gca,'ytick',[])
subplot(4,1,3)
plot(swc(3,:))
ylabel('D3')
set(gca,'ytick',[])
subplot(4,1,4)
```

```
plot(swc(5,:))
ylabel('A4')
set(gca,'ytick',[])
```



The wavelet and approximation coefficients at each level are equal in length to the input signal. The additive noise is almost entirely localized in the level one detail coefficients. The level 3 detail coefficients capture the high-frequency oscillations at the beginning of the Doppler signal. The level 4 approximation coefficients are a lowpass approximation to the Doppler signal.

Obtain the 2-D nondecimated wavelet transform of an image. Use the Daubechies least asymmetric wavelet, `sym4`, and obtain the multiresolution analysis down to level 3. Load the image. Use `wcodemat` to scale the matrix for display.

```
load tartan
nbcol = size(map,1);
cod_X = wcodemat(X,nbcol);
```

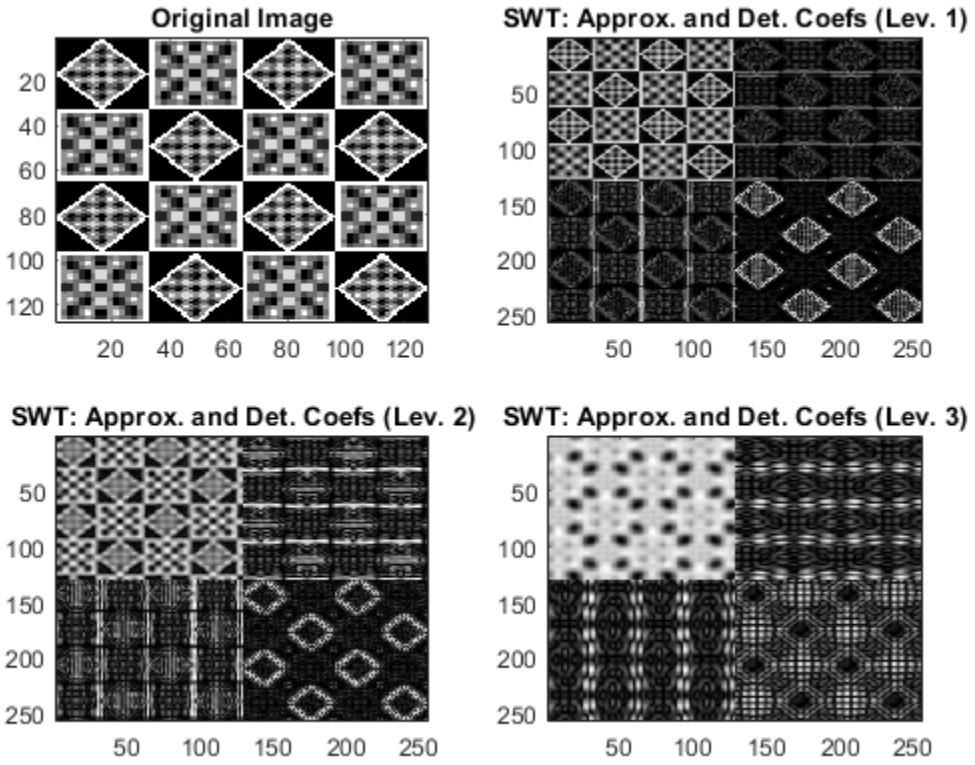
Obtain the nondecimated multiresolution analysis down to level 3.

```
[ca,chd,cvd,cdd] = swt2(X,3,'sym4');
```

Display the original image and the approximation and detail coefficients at each level.

```
figure
subplot(2,2,1)
image(cod_X)
title('Original Image')
```

```
colormap(map)

for k = 1:3
    cod_ca  = wcodemat(ca(:,:,k),nbcol);
    cod_chd = wcodemat(chd(:,:,k),nbcol);
    cod_cvd = wcodemat(cvd(:,:,k),nbcol);
    cod_cdd = wcodemat(cdd(:,:,k),nbcol);
    decl = [cod_ca,cod_chd;cod_cvd,cod_cdd];

    subplot(2,2,k+1)
    image(decl)

    title(['SWT: Approx. ', ...
        'and Det. Coefs (Lev. ',num2str(k),')'])
    colormap(gray)
end
```



See Also

dwtfilterbank | modwt | modwtmra | swt | swt2 | wavedec | wavedec2 | wdenoise | wdenoise2

Lifting

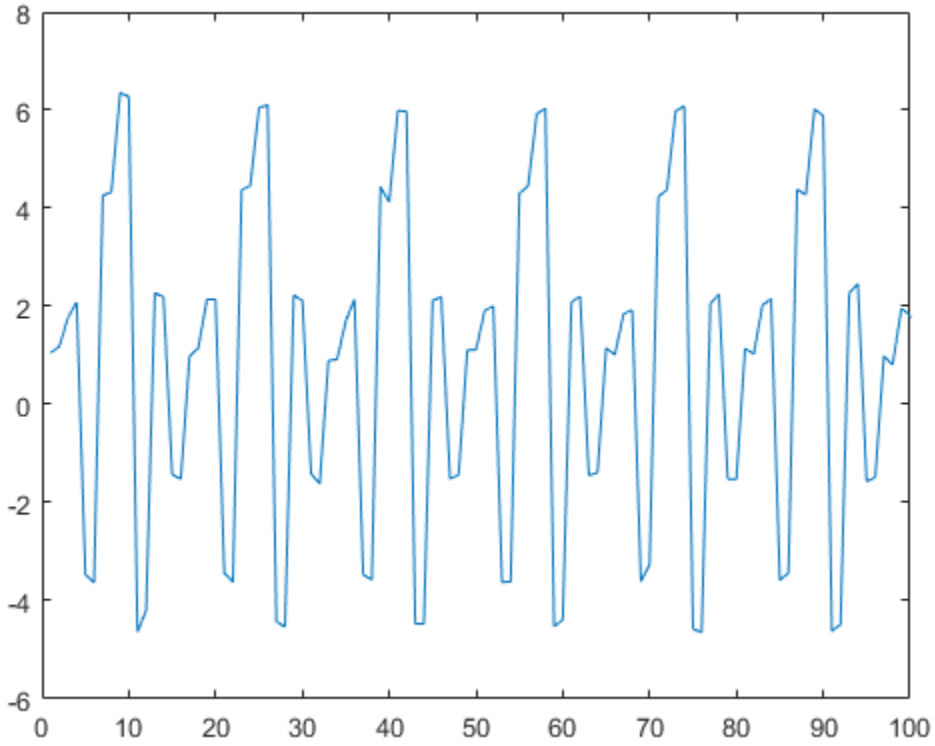
This example shows how to use lifting on a 1-D signal.

Create a 1-D signal that is piecewise constant over 2 samples. Add $N(0, 0.1^2)$ noise to the signal.

```
x = [1 1 2 2 -3.5 -3.5 4.3 4.3 6 6 -4.5 -4.5 2.2 2.2 -1.5 -1.5];
x = repmat(x,1,64);
rng default
x = x+ 0.1*randn(size(x));
```

Plot the signal and zoom in on the first 100 samples to visualize the correlation in neighboring samples.

```
plot(x)
xlim([0 100])
```



Use the *lazy* wavelet to obtain the even and odd polyphase components of the signal.

```
LS = liftwave('lazy');
[A,D] = lwt(x,LS);
```

If you plot the detail (wavelet) coefficients in `D`, you see that this transform has not decorrelated the signal. The wavelet coefficients look very much like the signal.

Add a dual lifting step that subtracts the even-indexed coefficient from the odd-coefficient one sample later, $x(2n + 1) - x(2n)$.

```
els = {'d',-1,0};
LSnew = addlift(LS,els);
```

Because the signal is piecewise constant over consecutive samples with additive noise, the new dual lifting step should result in wavelet coefficients small in absolute value. In this case, the wavelet transform does decorrelate the data. Verify this by finding the approximation and detail coefficients with the new dual lifting step.

```
[A,D] = lwt(x,LSnew);
```

If you plot the detail (wavelet) coefficients, you see that the wavelet coefficients no longer resemble the original signal.

The approximation coefficients, `A`, of the previous transform constitute the even polyphase component of the signal. Therefore, the coefficients are affected by aliasing. Use a primal lifting step to update the approximation coefficients and reduce aliasing. The primal step replaces the approximation coefficients by $x(2n) + 1/2(x(2n+1) - x(2n))$, which is equal to the average of $x(2n)$ and $x(2n+1)$. The averaging is a lowpass filtering, which helps to reduce aliasing.

```
els = {'p',1/2, 0};
LSnew = addlift(LSnew,els);
```

Use the updated lifting scheme to obtain the wavelet transform of the input signal.

```
[A,D] = lwt(x,LSnew);
```

Add the appropriate scaling to ensure perfect reconstruction. Obtain the approximation and wavelet coefficients using lifting and reconstruct the signal using `ilwt`. Verify perfect reconstruction.

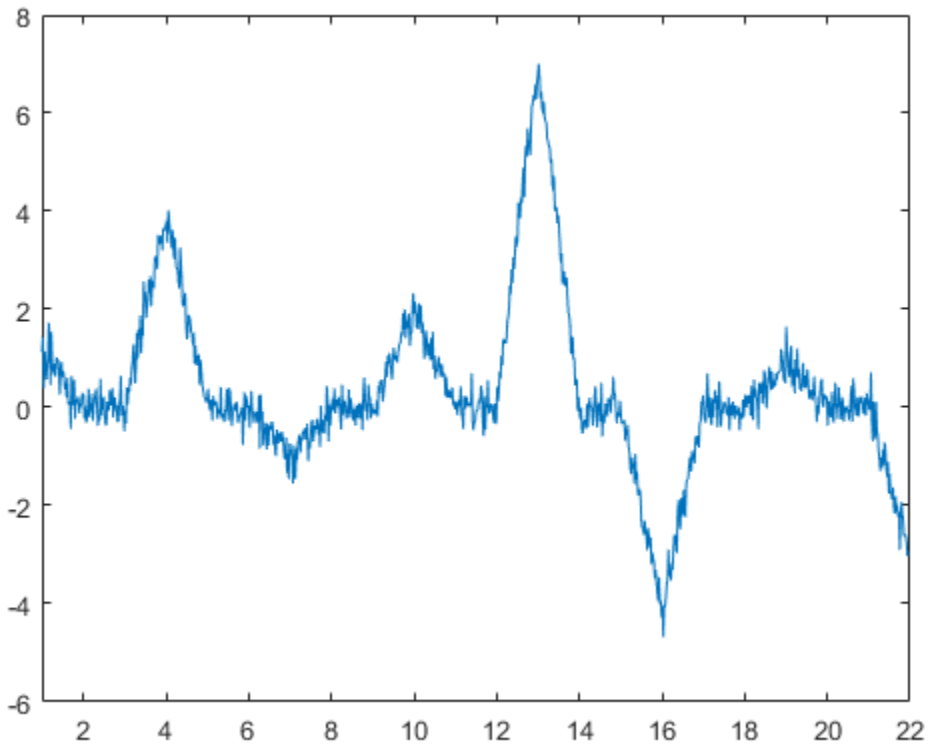
```
LSnew(end,:) = {sqrt(2),sqrt(2)/2,[]};
[A,D] = lwt(x,LSnew);
x1 = ilwt(A,D,LSnew);
max(abs(x1-x))
```

```
ans = 1.7764e-15
```

The preceding example designed a wavelet, which effectively removed a zeroth order polynomial (constant). If the behavior of the signal is better represented by a higher-order polynomial, you can design a dual wavelet with the appropriate number of vanishing moments to decorrelate the signal.

Use the lifting scheme to design a wavelet with 2 vanishing moments. A dual wavelet with 2 vanishing moments decorrelates a signal with local behavior approximated by a first-order polynomial. Create a signal characterized by first-order polynomial behavior with additive $N(0, 0.25^2)$ noise.

```
y = [1 0 0 4 0 0 -1 0 0 2 0 0 7 0 0 -4 0 0 1 0 0 -3];
x1 = 1:(21/1024):22-(21/1024);
y1 = interp1(1:22,y,x1,'linear');
rng default
y1 = y1+0.25*randn(size(y1));
plot(x1,y1)
xlim([1 22])
```



In this case, the wavelet coefficients should remove a first-order polynomial. If the signal value at an odd index, $x(2n + 1)$, is well approximated by a first-order polynomial fitted to the surrounding sample values, then $1/2(x(2n) + x(2n + 2))$ should provide a good fit for $x(2n + 1)$. In other words, $x(2n + 1)$ should be the midpoint between $x(2n)$ and $x(2n + 2)$.

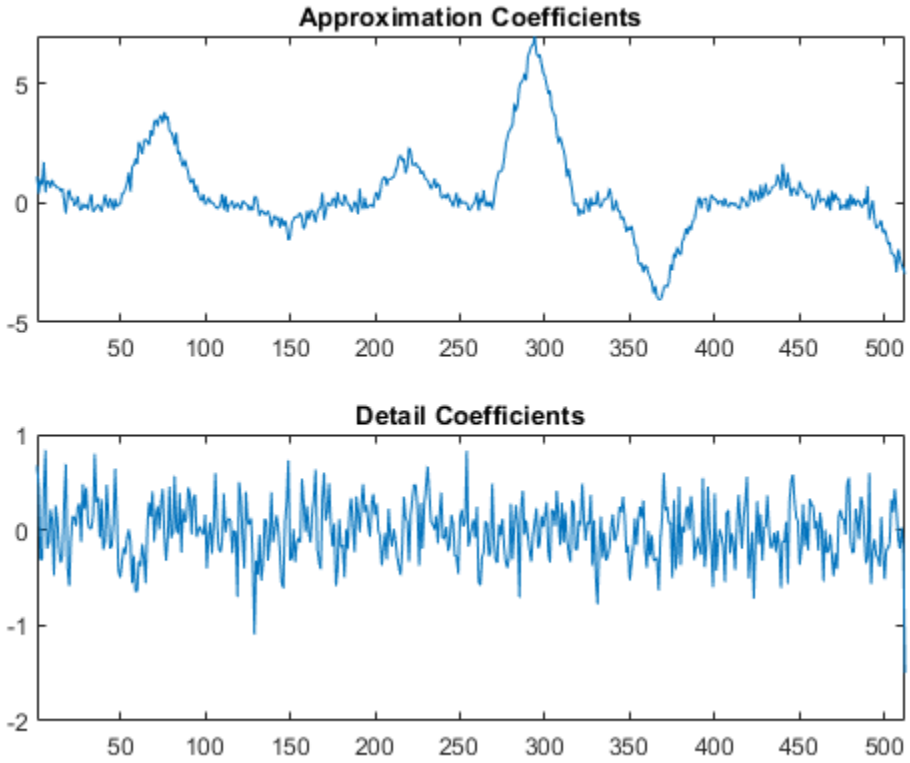
It follows that $x(2n + 1) - 1/2(x(2n) + x(2n + 2))$ should decorrelate the signal.

Start with the lazy wavelet transform and add a dual lifting step which models the preceding equation.

```
LS = liftwave('lazy');
els = {'d',[-1/2 -1/2],1};
LSnew = addlift(LS,els);
```

Use the lifting scheme to obtain the approximation and detail coefficients and plot the result.

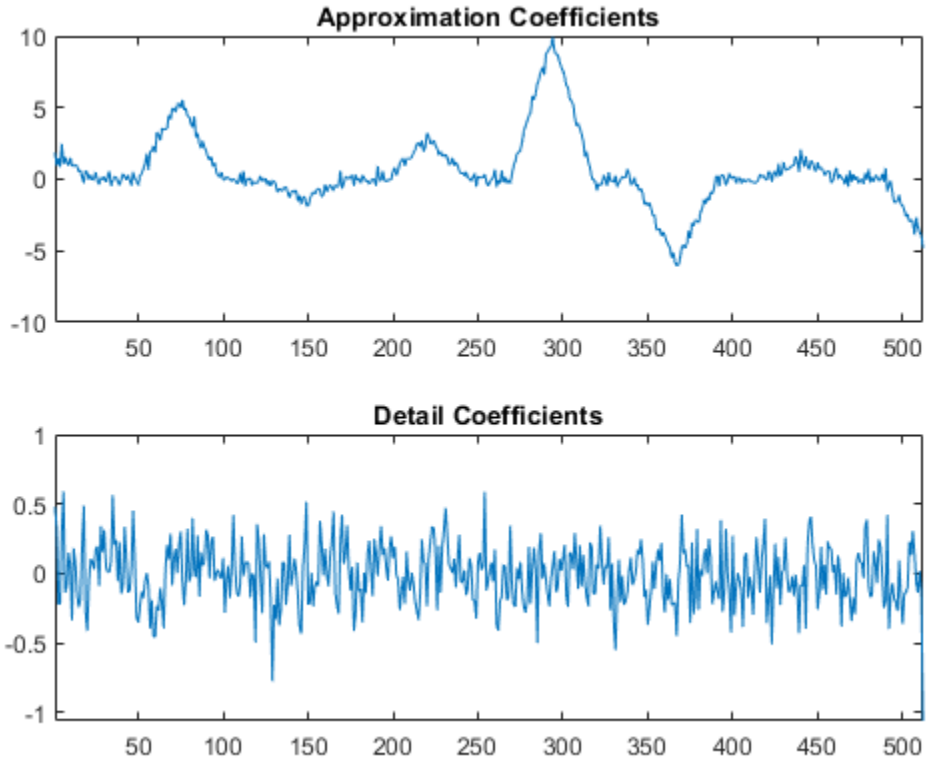
```
[A,D] = lwt(y1,LSnew);
subplot(2,1,1)
plot(A)
xlim([1 512])
title('Approximation Coefficients')
subplot(2,1,2)
plot(D)
xlim([1 512])
title('Detail Coefficients')
```



You see that the wavelet coefficients appear to only contain noise, while the approximation coefficients represent a denoised version of the original signal. Because the preceding transform uses only the even polyphase component for the approximation coefficients, you can reduce aliasing by adding a primal lifting step. Finally, add the normalization constants to produce a perfect reconstruction filter bank.

Obtain the discrete wavelet transform with the new lifting scheme and plot the results.

```
els = {'p',[1/4 1/4],0};
LSnew = addlift(LSnew,els);
LSnew(end,:) = {sqrt(2),sqrt(2)/2,[]};
[A,D] = lwt(y1,LSnew);
subplot(2,1,1)
plot(A)
xlim([1 512])
title('Approximation Coefficients')
subplot(2,1,2)
plot(D)
xlim([1 512])
title('Detail Coefficients')
```



Demonstrate that you have designed a perfect reconstruction filter bank.

```
y2 = ilwt(A,D,LSnew);
max(abs(y2-y1))
```

```
ans = 8.8818e-16
```

Critically Sampled Wavelet Packet Analysis

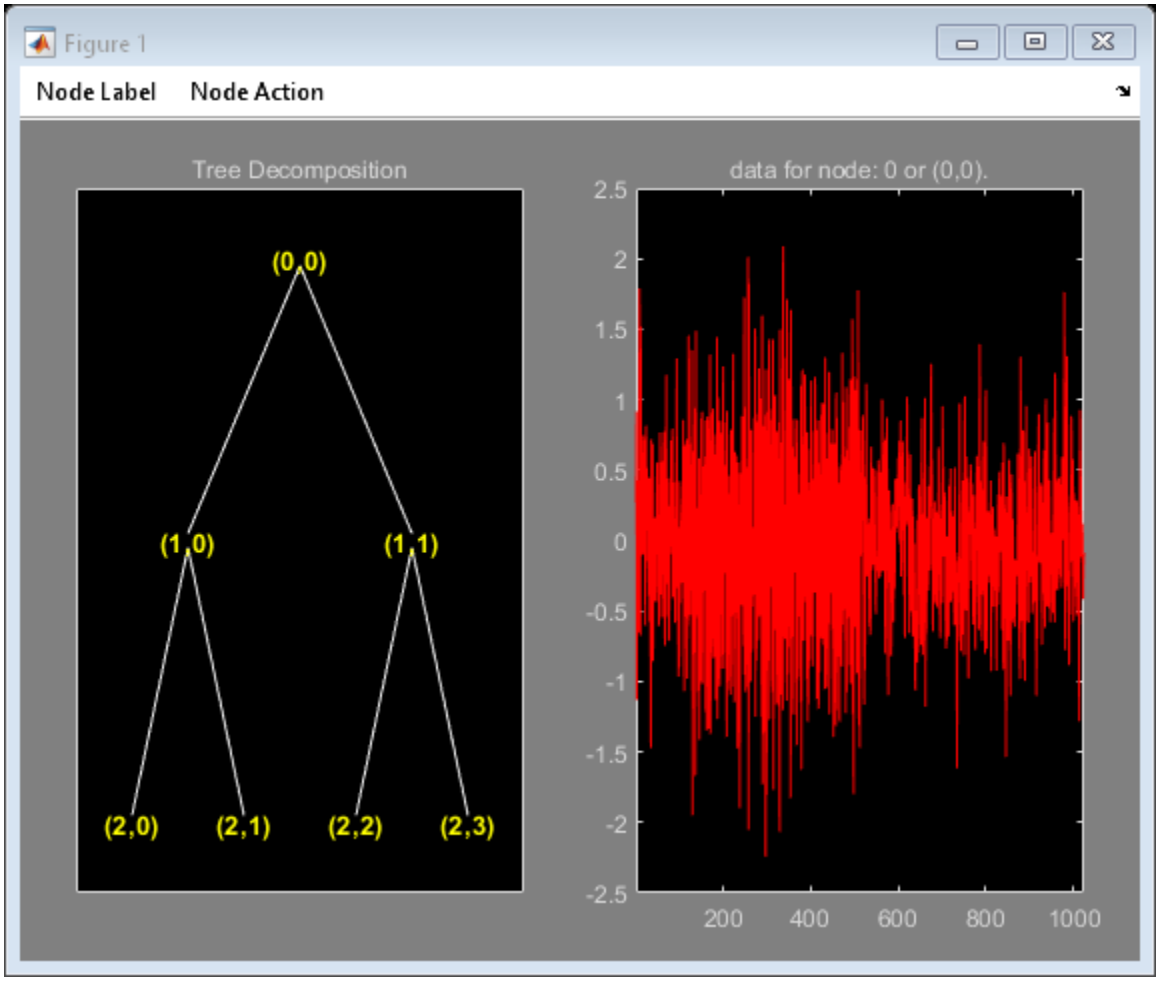
This example shows how to obtain the wavelet packet transform of a 1-D signal. The example also demonstrates that frequency ordering is different from Paley ordering.

Create a signal consisting of a sine wave with a frequency of $7\pi/8$ radians/sample in additive white Gaussian N(0,1/4) noise. The sine wave occurs between samples 128 and 512 of the signal. Set the `dwtmode` to periodization and return it to your original setting at the end of the example.

```
rng default
st = dwtmode('status','nodisplay');
dwtmode('per','nodisp');

n = 0:1023;
indices = (n>127 & n<=512);
x = cos(7*pi/8*n).*indices+0.5*randn(size(n));
```

Obtain the wavelet packet transform down to level 2 using the Daubechies least asymmetric wavelet with 4 vanishing moments. Plot the wavelet packet tree.

```
T = wpdec(x,2,'sym4');
plot(T)
```

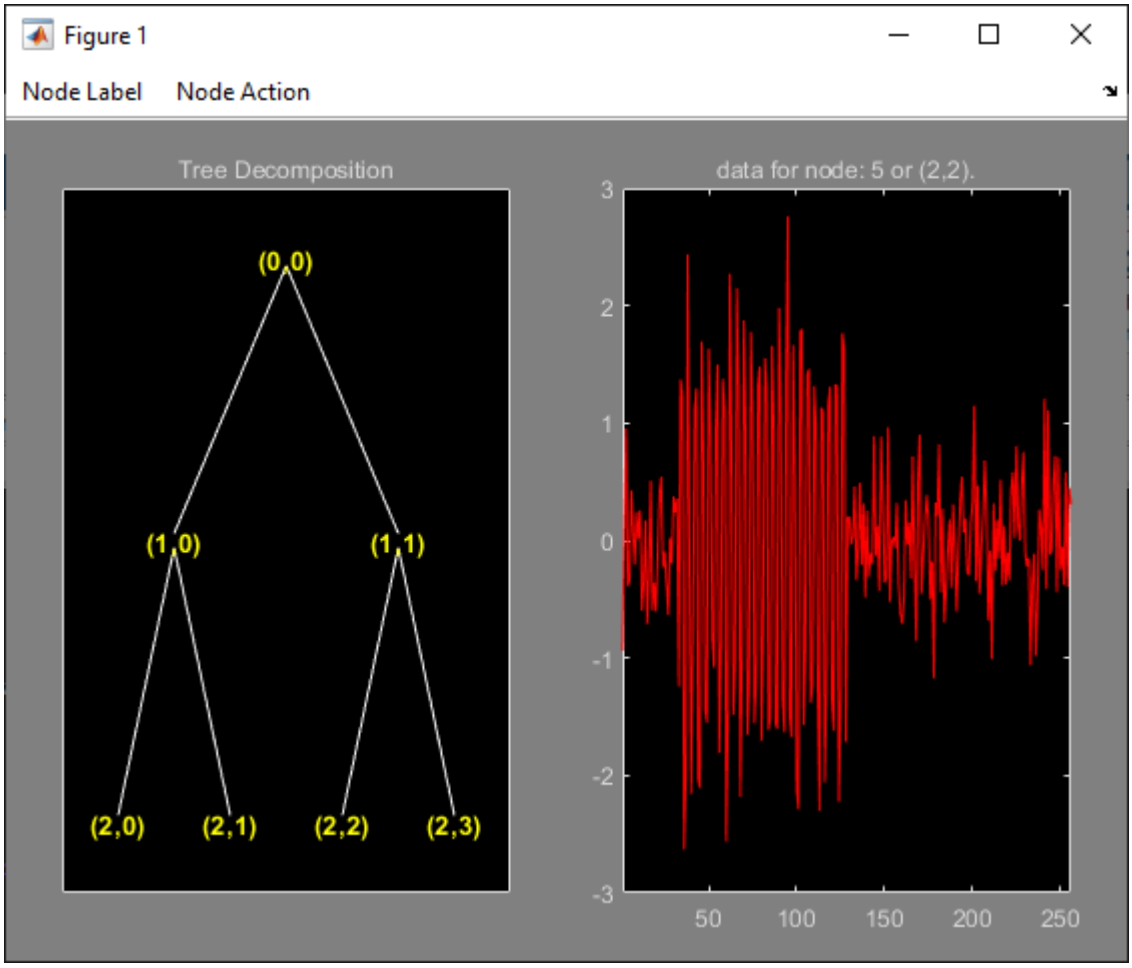


Find the Paley and frequency ordering of the terminal nodes.

```
[tn_pal,tn_freq] = otnodes(T);
```

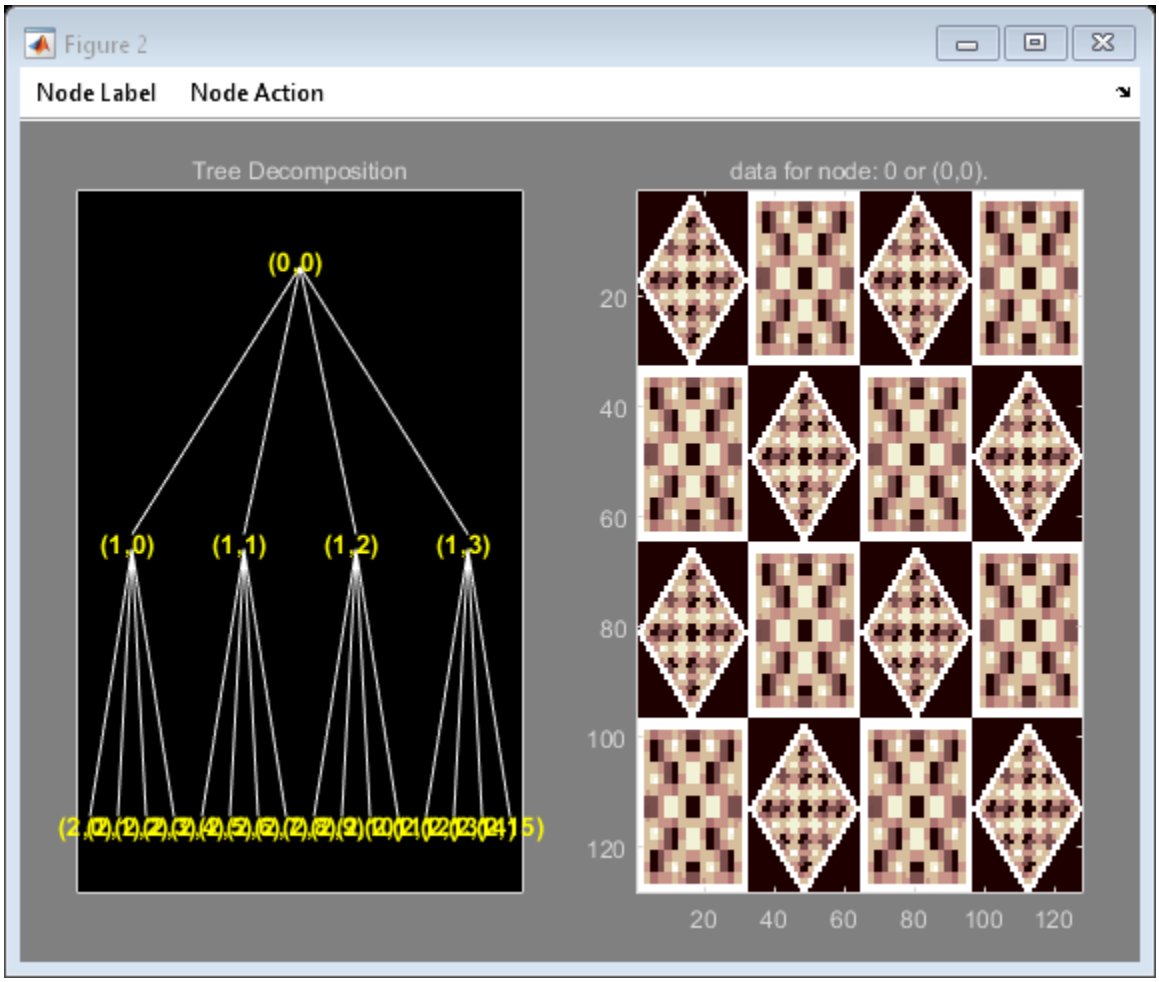
`tn_freq` contains the vector `[3 4 6 5]`, which shows that the highest frequency interval, $[3\pi/4, \pi)$, is actually node 5 in the Paley-ordered wavelet packet tree.

Click on node (2,2) in the wavelet packet tree to see that the frequency ordering correctly predicts the presence of the sine wave.



The wavelet packet transform of a 2-D image yields a quaternary wavelet packet tree. Load an example image. Use the biorthogonal B-spline wavelet with 3 vanishing moments in the reconstruction wavelet and 5 vanishing moments in the decomposition wavelet. Plot the resulting quaternary wavelet packet tree.

```
load tartan
T = wpdec2(X,2,'bior3.5');
plot(T)
```




```
dwtmode(st,'nodisplay')
```