Wavelet Toolbox™
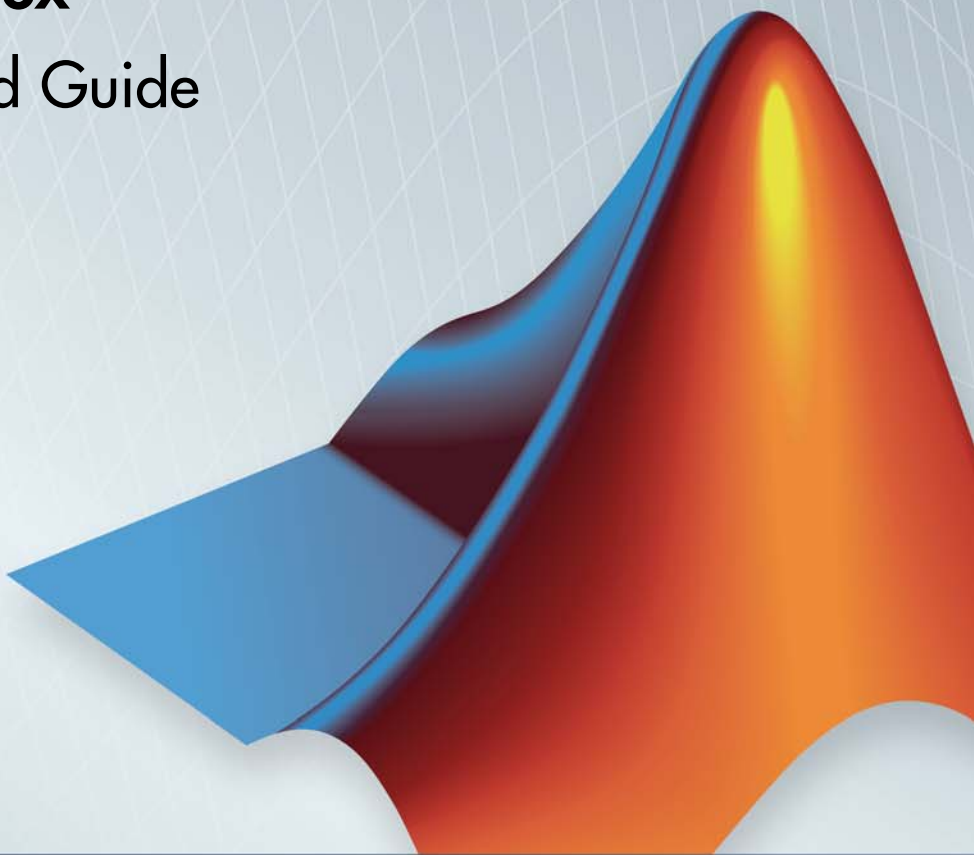# Getting Started Guide

R2013a

Michel Misiti
Yves Misiti
Georges Oppenheim
Jean-Michel Poggi

MATLAB®

MathWorks®

How to Contact MathWorks

www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.com/contact_TS.html	Technical Support

suggest@mathworks.com	Product enhancement suggestions
bugs@mathworks.com	Bug reports
doc@mathworks.com	Documentation error reports
service@mathworks.com	Order status, license renewals, passcodes
info@mathworks.com	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

For contact information about worldwide offices, see the MathWorks Web site.

Wavelet Toolbox™ Getting Started Guide

Trademarks

Patents

Contents

Getting Started with Wavelet Toolbox Software

1

Using Wavelets

2

Getting Started with Wavelet Analysis

3

1

Getting Started with Wavelet Toolbox Software

Product Description

Analyze and synthesize signals and images using wavelet techniques

Wavelet Toolbox™ software extends the MATLAB® technical computing environment with graphical tools and command-line functions for developing wavelet-based algorithms for the analysis, synthesis, denoising, and compression of signals and images.

The Wavelet Toolbox supports the interactive exploration of wavelet properties and applications. It is useful for speech and audio processing, image and video processing, biomedical imaging, and 1-D and 2-D applications in communications and geophysics.

Key Features

- Standard wavelet families, including Daubechies wavelet filters, complex Morlet and Gaussian, real reverse biorthogonal, and discrete Meyer

- Wavelet and signal processing utilities, including a function to convert scale to frequency

- Methods for adding wavelet families

- Lifting methods for constructing wavelets

- Customizable presentation and visualization of data

- Interactive tools for continuous and discrete wavelet analysis

- Wavelet packets, implemented as MATLAB objects

- One-dimensional multisignal analysis, compression, and denoising

- Multiscale principal component analysis

- Multivariate denoising

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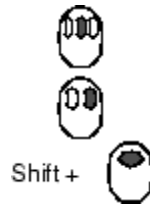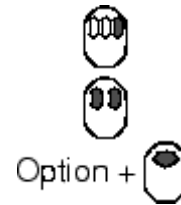
Note The functionality of the middle mouse button and the right mouse button can be inverted depending on the platform.

Wavelets: Tools for Sparse Representation

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.

What is a Wavelet?

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

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.

Sine Wave   Wavelet (db10)
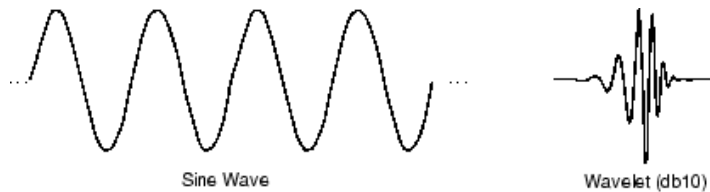
Fourier analysis consists of breaking up a signal into sine waves of various frequencies. Similarly, wavelet analysis is the breaking up of a signal into shifted and scaled versions of the original (or *mother*) wavelet.

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.

What Can Wavelet Analysis Do?

One major advantage afforded by wavelets is the ability to perform *local analysis.* Because wavelets are localized in time and scale, wavelet coefficients are able to localize abrupt changes in smooth signals and images.

This example shows you how wavelet coefficients localize a discontinuity in a sine wave.

Create a 1-Hz sine wave sampled at 100 Hz. 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')
```
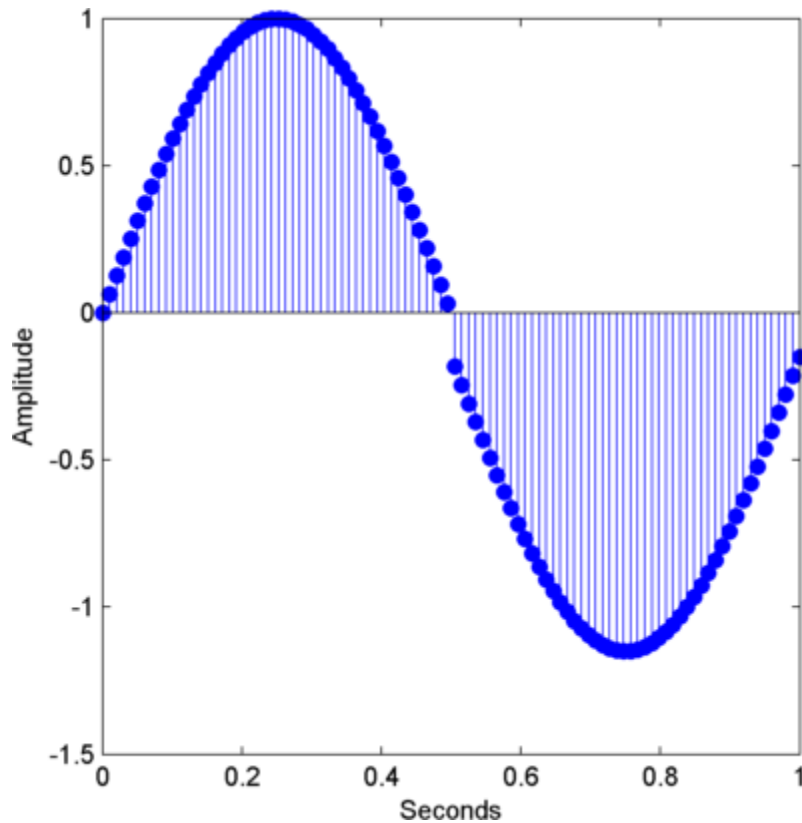
Obtain the nondecimated discrete wavelet transform of the sine wave using the 'sym2' wavelet and plot the wavelet (detail) coefficients along with the original signal.

```
[swa,swd] = swt(y,1,'sym2');
subplot(211)
stem(t,y,'markerfacecolor',[0 0 1]); title('Orignal Signal');
subplot(212)
stem(t,swd,'markerfacecolor',[0 0 1]); title('Level 1 Wavelet Coefficients'
```
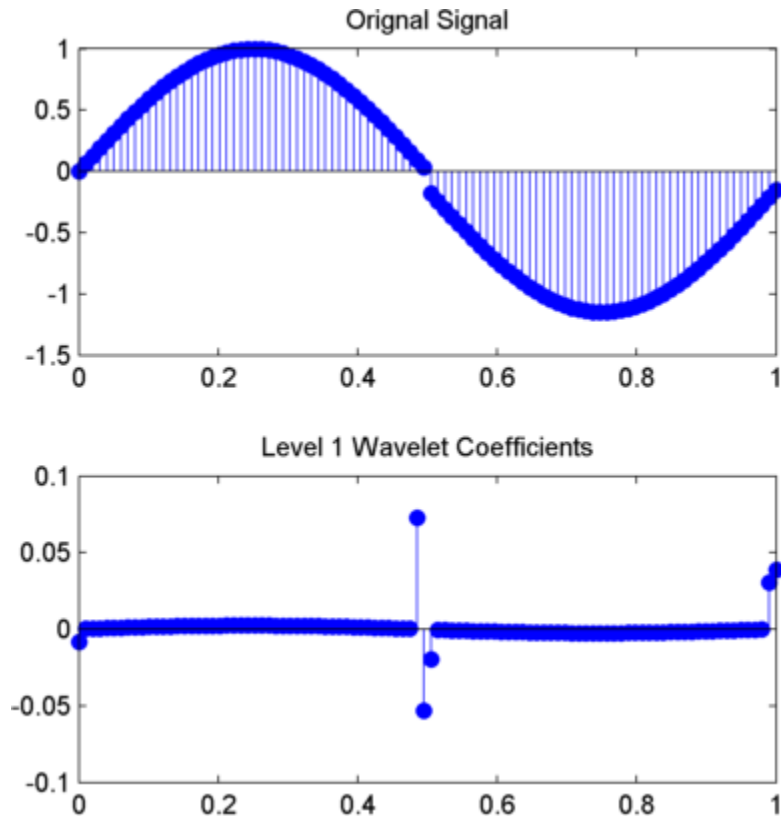
Compare the Fourier coefficient magnitudes for the 1-Hz sine wave with and without the discontinuity.

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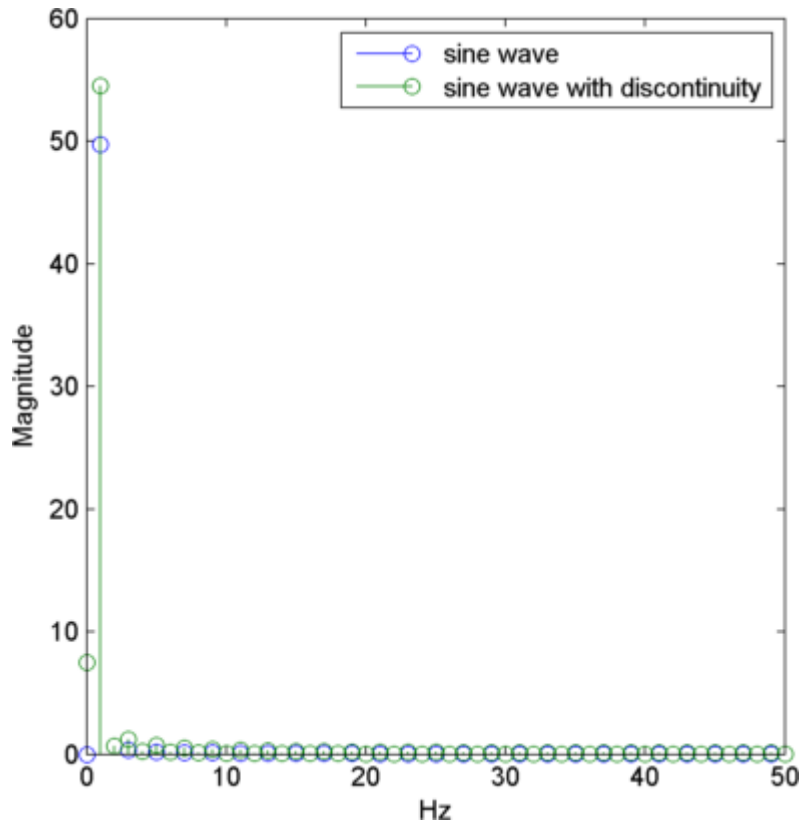
There is minimal difference in the magnitudes of the Fourier coefficients. Because the discrete Fourier basis vectors have support over the entire time interval, the discrete Fourier transform does not detect the discontinuity as efficiently as the wavelet transform.

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

```
[~,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)');
```
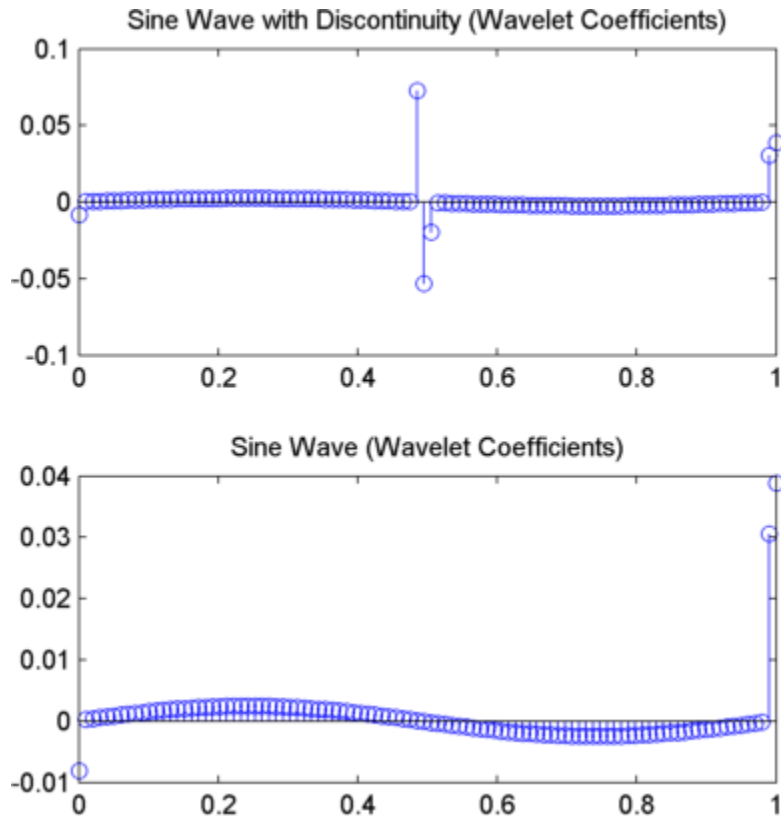
The wavelet coefficients of the two signals demonstrate a significant difference.

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.

From Fourier Analysis to Wavelet Analysis

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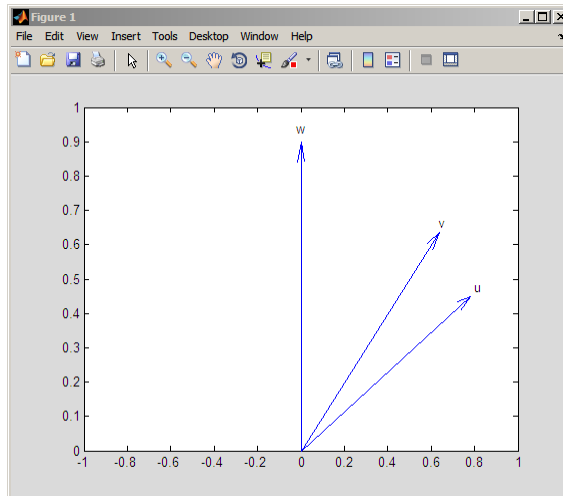
The inner products capture this geometric fact. Mathematically, the inner product of two vectors, u and v is equal to the product of their norms and the cosine of the angle, θ, between them:

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

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 1-11 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

Signal   Constituent sinusoids of different frequencies
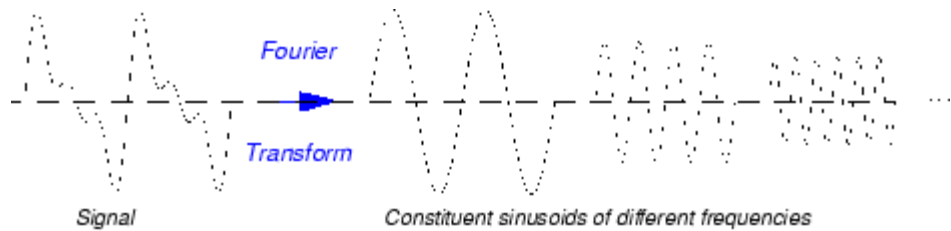
Because $e^{j\omega t}$ is complex-valued, F(ω) is, in general, complex-valued. If the signal contains significant oscillations at an angular frequency of ω_0, the absolute value of $F(\omega_0)$ will be large. By examining a plot of $|F(\omega)|$ as a function of angular frequency, it is possible to 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');
```
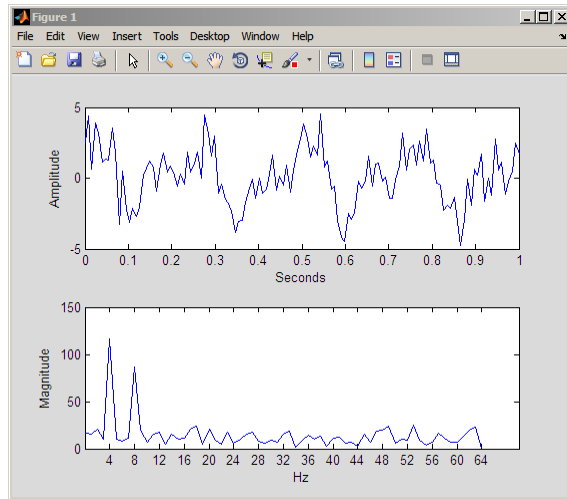
Viewed as a time signal, it is difficult to determine what significant oscillations are present in the data. However, looking at the absolute value of the Fourier transform coefficients as function of frequency, the dominant oscillations at 4 and 8 Hz are easy to detect.

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:
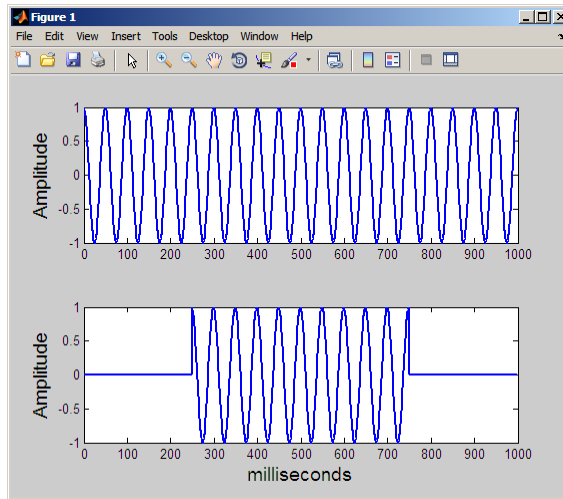
Both signals consist of a single sine wave with a frequency of 20 Hz. However, in the top signal, the sine wave lasts the entire 1000 milliseconds. In the bottom plot, the sine wave starts at 250 and ends at 750 milliseconds. The Fourier transform detects that the two signals have the same frequency 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 two-dimensional representation of a one-dimensional signal means that there is redundancy in the STFT. The following figure demonstrates how the STFT maps a signal into a time-frequency representation.
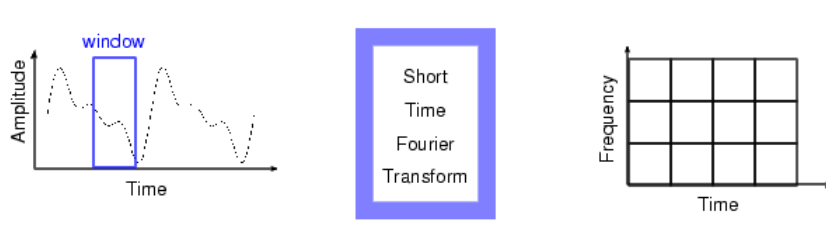
The STFT represents a sort of compromise between time- and frequency-based views of a signal. It provides some information about both when and at what frequencies a signal event occurs. However, you can only obtain this information with limited precision, and that precision is determined by the size 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:
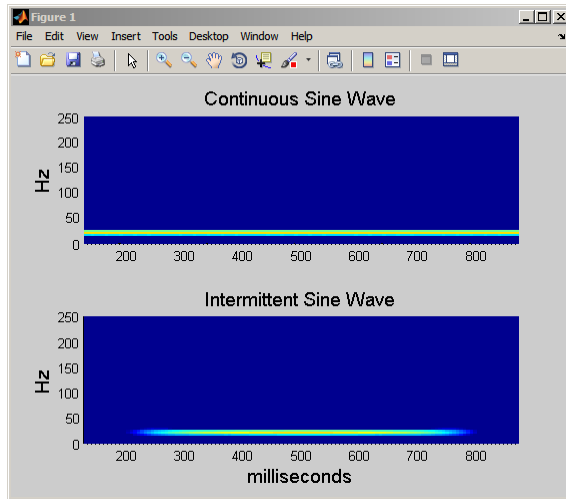
By using the STFT, you can see that the intermittent sine wave begins near 250 msec and ends around 750 msec. Additionally, you can see that the signal's energy is concentrated around 20 Hz.

Continuous Wavelet Transform

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 in 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 two-dimensional representation of a one-dimensional 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:

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

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.

Signal   Constituent wavelets of different scales and positions
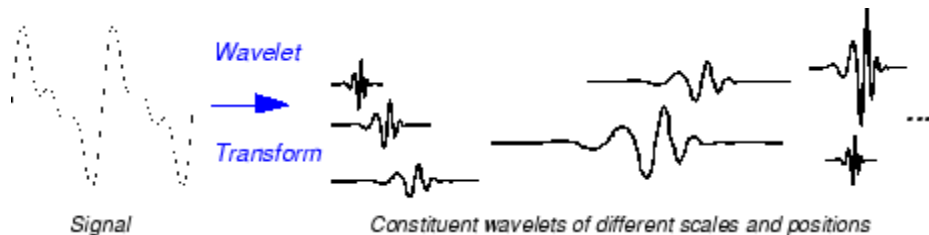
There are many different admissible wavelets that can be used in the CWT. While it may seem confusing that there are so many choices for the analyzing wavelet, it is actually a strength of wavelet analysis. Depending on what signal features you are trying to detect, you are free to select a wavelet that 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 interesting 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 a inherently positive quantity, $a>0$. For sinusoids, the effect of the scale factor is very easy to see.
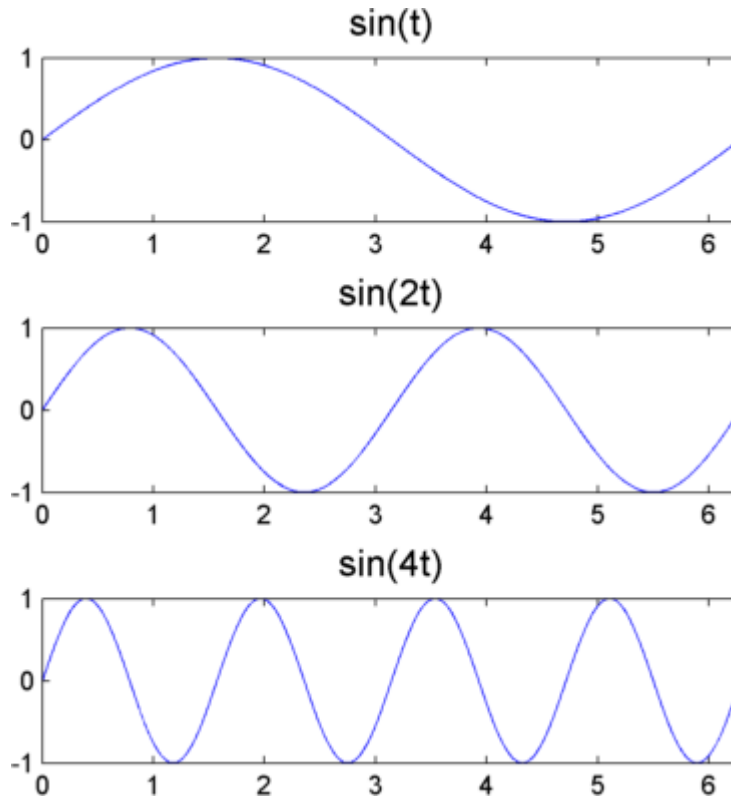
In sin(at), the scale is the inverse of the radian frequency, a.

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.

$$\psi(t)$$

$$\psi(t/2)$$

$$\psi(t/4)$$
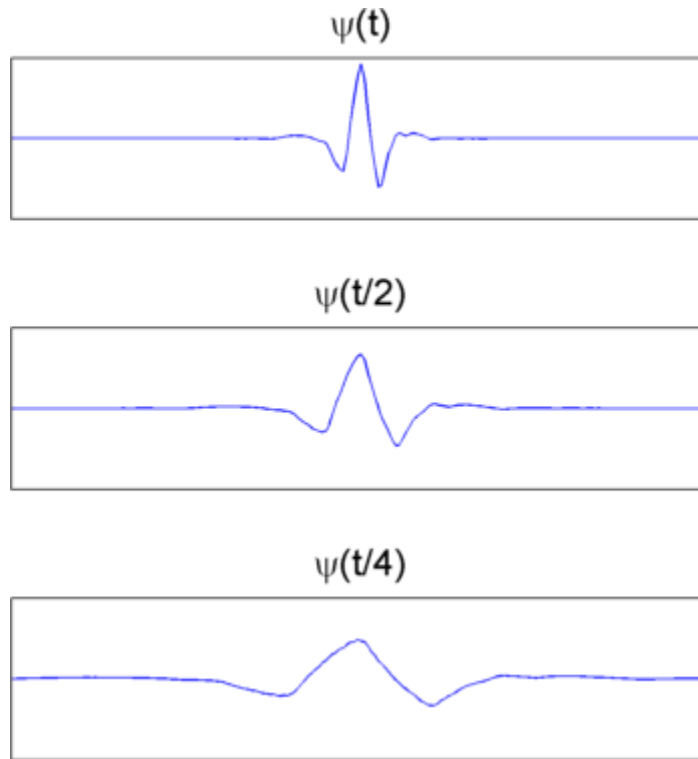
This general inverse relationship between scale and frequency holds for signals in general. See "CWT as a Filtering Technique" on page 1-26 and "Scale and Frequency" on page 1-23 for more information on the relationship between scale and frequency.

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 higher 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:

• Low scale a ⇒ Compressed wavelet ⇒ Rapidly changing details ⇒ High frequency ω.

• High 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.
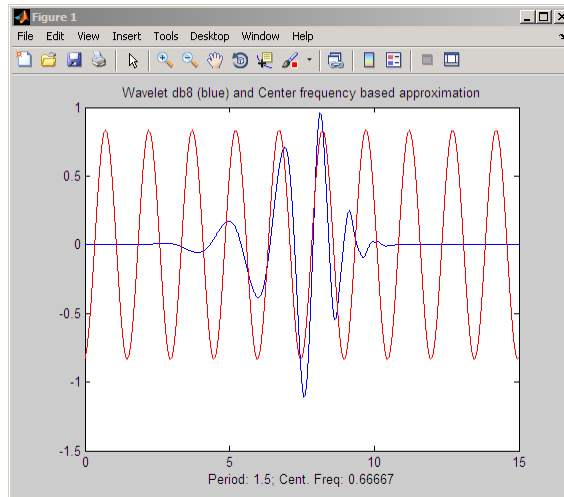
The relationship between scale and frequency in the CWT is also explored in "CWT as a Filtering Technique" on page 1-26.

Shifting

Shifting a wavelet simply means delaying (or advancing) its onset. Mathematically, delaying a function f(*t*) by *k* is represented by f($t - k$):
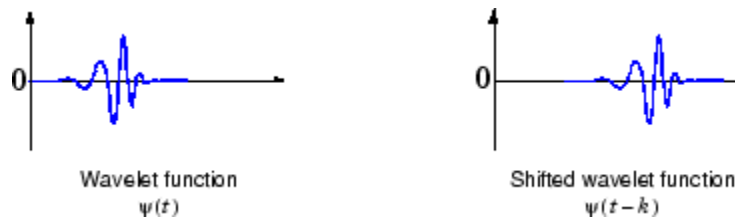
CWT as a Windowed Transform

In "Short-Time Fourier Transform" on page 1-15, 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 time, frequency, time-frequency, and time-scale representations of a signal.
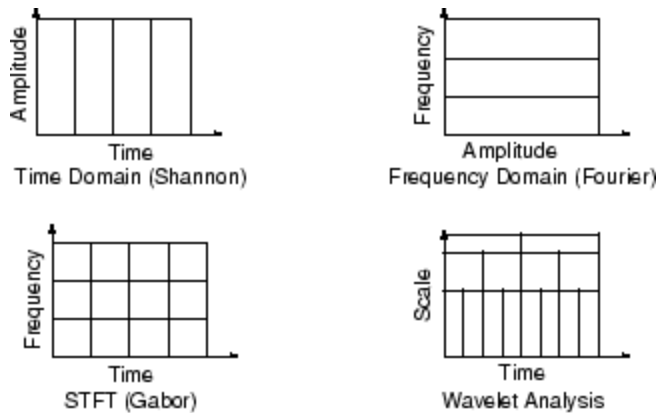
CWT as a Filtering Technique

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:

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

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

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

where $\hat{f}(\omega)$ and $\hat{\psi}(\omega)$ 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.
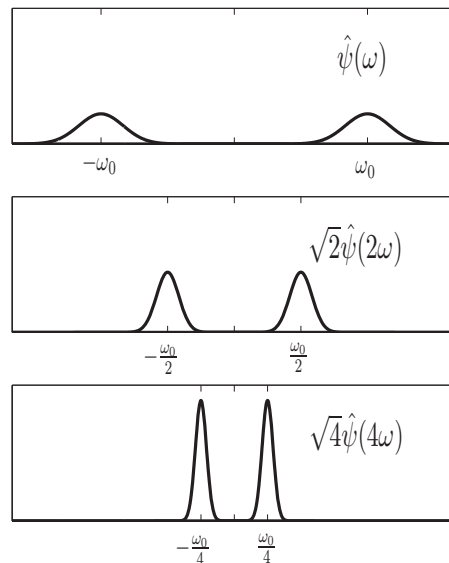
This depicts the CWT as a bandpass filtering of the input signal. CWT coefficients at lower scales represent energy in the input signal at higher frequencies, while CWT coefficients at higher scales represent energy in the input signal at lower frequencies. However, unlike Fourier bandpass filtering, 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 *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.

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

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:

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

If you define:

$$\tilde{\psi}_a(t) = \frac{1}{\sqrt{a}}\psi^*(-t\,/\,a)$$

you can rewrite the wavelet transform as

$$(f * \tilde{\psi}_a)(b) = \int_{-\infty}^{\infty} f(t)\tilde{\psi}_a(b-t)dt$$

which explicitly expresses the CWT as a convolution.

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

$$W_a[b] = \sum_{n=0}^{N-1} x[n]\tilde{\psi}_a[b-n]$$

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

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

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 `cwtft` function implements the CWT using an FFT-based algorithm. See `cwtftinfo` for information pertaining to the supported analyzing wavelets.

Inverse Continuous Wavelet Transform

The icwtft function implements the inverse CWT. Using icwtft requires that you obtain the CWT from cwtft. The Wavelet Toolbox does not support the inverse CWT for a general CWT obtained using 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 utilizes a discrete version of the single integral formula due to Morlet.

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:

$$C_\psi = \int_{-\infty}^{\infty} \frac{|\overset{\wedge}{\psi}(\omega)|^2}{|\omega|} d\omega < \infty$$

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. If you inspect the list of wavelets supported by cwtft, each wavelet is either analytic or has a real-valued Fourier transform. 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:

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

Define the constant:

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

INote that 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:

$$C_{\psi_1,\psi_2} < f,g >= \iint <f,\psi_1><g,\psi_2>^* db\frac{da}{a^2}$$

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

$$f(t) = 2\, Re\,\{\frac{1}{C_{\psi_1,\delta}} \int_0^\infty <f(t),\psi_1(t)> \frac{da}{a^{3/2}}\}$$

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.

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

Continuous Wavelet Transform Algorithm

The following outlines the basic algorithm for the CWT:

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 1-11 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 "Definition of the Continuous Wavelet Transform" on page 1-19, 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.

C = 0.0102
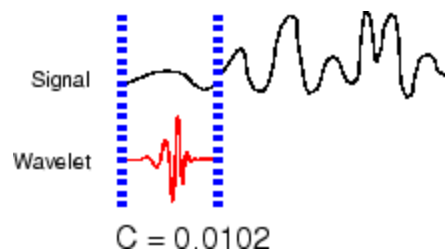
3 Shift the wavelet to the right and repeat steps 1 and 2 until you've covered the whole signal.
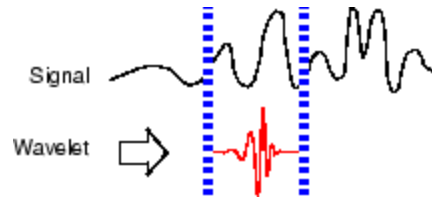
4 Scale (stretch) the wavelet and repeat steps 1 through 3.

C = 0.2247
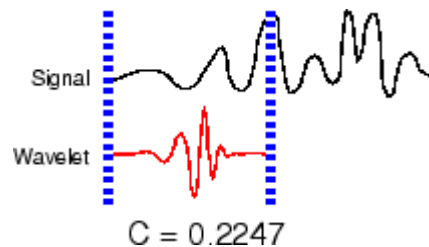
5 Repeat steps 1 through 4 for all scales.

Interpreting CWT 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');
```
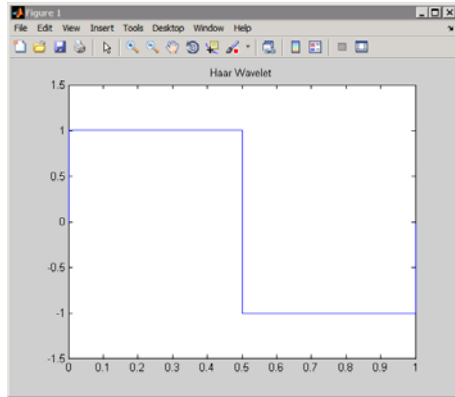
To compute the CWT using the Haar wavelet at scales 1 to 128, enter:

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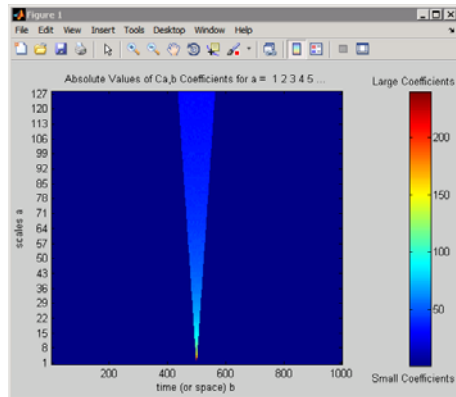
The preceding figure was modified with text labels to explicitly show which colors indicate large and small CWT coefficients.

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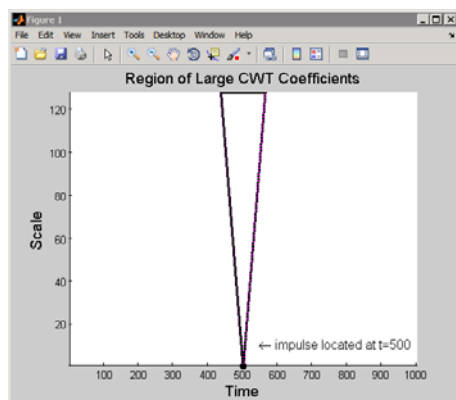
This region is referred to as the *cone of influence* of the point *t*=500 for the Haar wavelet. For a given point, the cone of influence shows you which CWT coefficients are affected by the signal value at that point.

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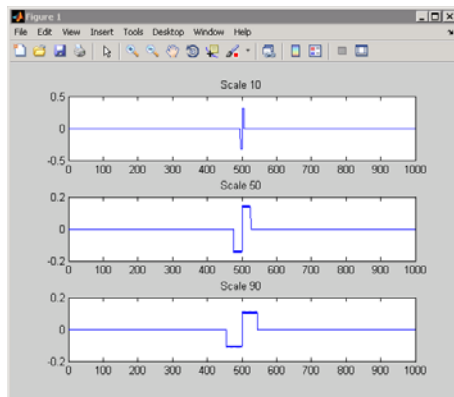
The cone of influence depends on the wavelet. You can find and plot the cone of influence for a specific wavelet with conofinf.

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.
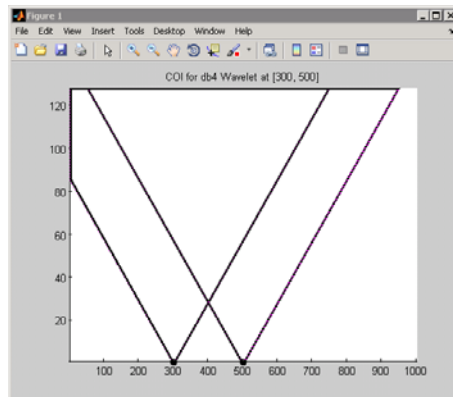
Look at point 400 for scale 20. At that scale, you can see that neither cone of influence overlaps the point 400. Therefore, you can expect that the CWT coefficient will be zero at that point and scale. The signal is only nonzero at two values, 300 and 500, and neither cone of influence for those values 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;
```
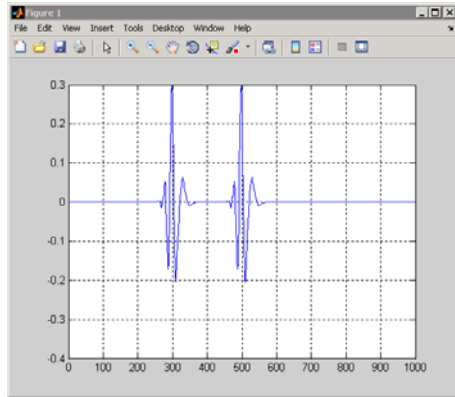
Next, look at the point 400 at scale 80. At scale 80, the cones of influence for both points 300 and 500 include the point 400. Even though the signal is zero at point 400, you obtain a nonzero CWT coefficient at that scale. The CWT coefficient is nonzero because the support of the wavelet has become 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;
```
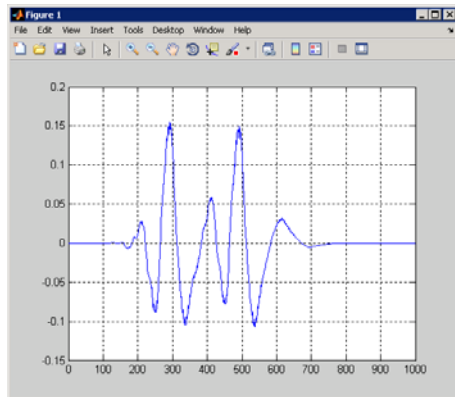

In the preceding example, the CWT coefficients became large in the vicinity of an abrupt change in the signal. This ability to detect discontinuities is a strength of the wavelet transform. The preceding example also demonstrated

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;
```
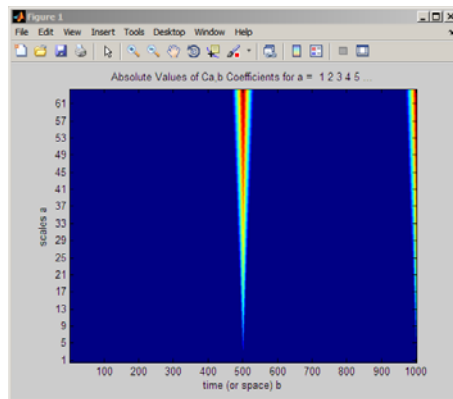

Similar to the shifted impulse example, the abrupt transition in the shifted step function results in large CWT coefficients at the discontinuity. The following figure illustrates why this occurs.
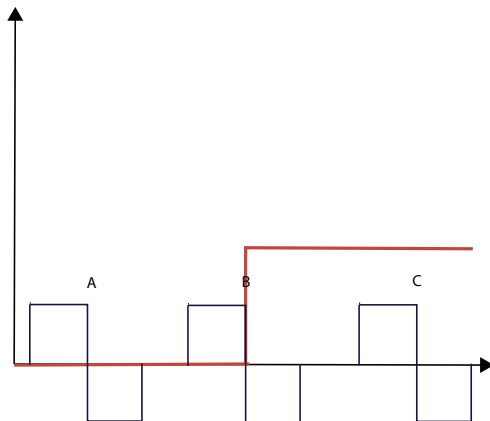
In the preceding figure, the red function is the shifted unit step function. The black functions labeled A, B, and C depict Haar wavelets at the same scale but different positions. You can see that the CWT coefficients around position A are zero. The signal is zero in that neighborhood and therefore 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.
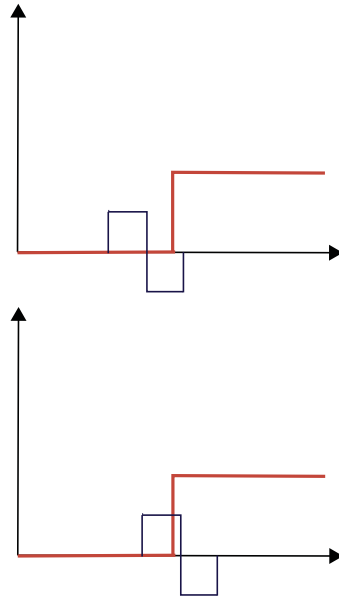
In the top figure, the wavelet has just begun to overlap with the unity portion of the unit step. In this case, the CWT coefficients are negative, but not as large in absolute value as those obtained at position B. In the bottom figure, the wavelet has shifted past position B and the positive deflection 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.
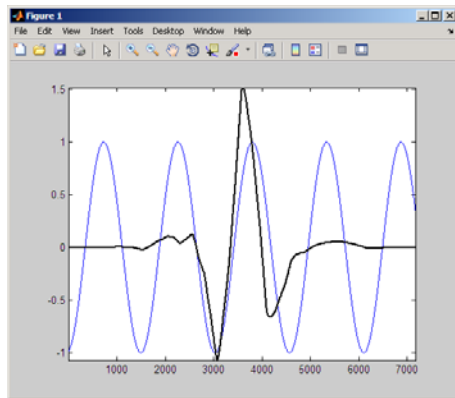
Recall that the CWT coefficients are obtained by computing the product of the signal with the shifted and scaled analyzing wavelet and integrating the result. The following figure shows the product of the wavelet and the sinusoid from the preceding figure.
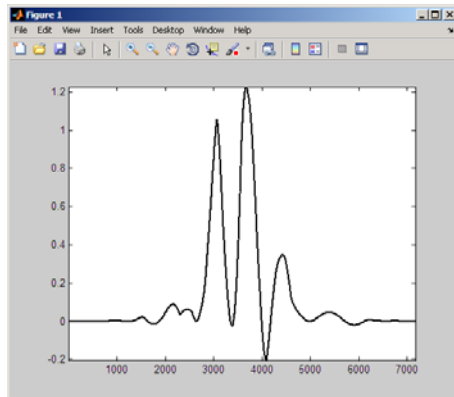
You can see that integrating over this product produces a positive CWT coefficient. That results because the oscillation in the wavelet approximately matches a period of the sine wave. The wavelet is *in phase* with the sine wave. The negative deflections of the wavelet approximately match the negative 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.
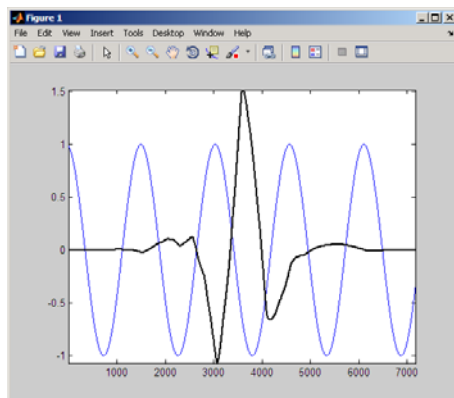
Examine the product of the shifted wavelet and the sinusoid.
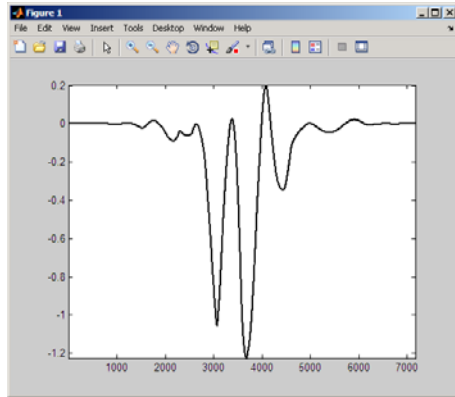
You can see that integrating over this product produces a negative CWT coefficient. That results because the wavelet is 1/2 cycle out of phase with the sine wave. The negative deflections of the wavelet approximately match the positive deflections of the sine wave. The positive deflections of the wavelet approximately match the negative deflections of the sinusoid.

Finally, shift the wavelet approximately one quarter cycle of the sine wave.
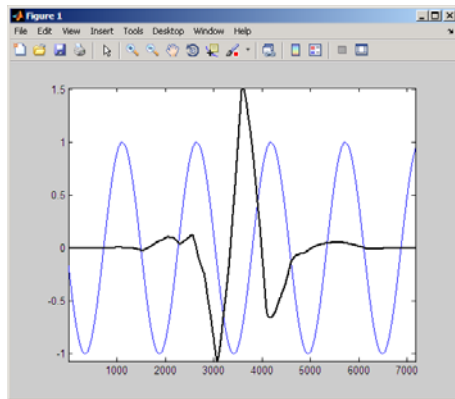
The following figure shows the product of the shifted wavelet and the sinusoid.
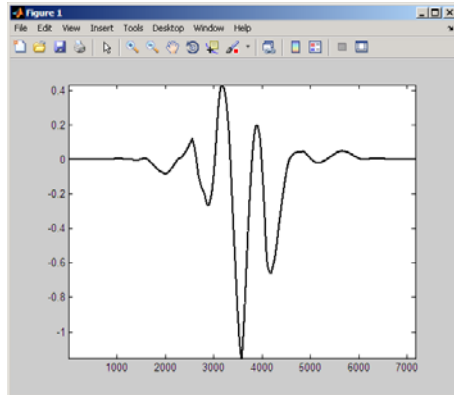
Integrating over this product produces a CWT coefficient much smaller in absolute value than either of the two previous examples. That results because the negative deflection of the wavelet approximately aligns with a positive deflection of the sine wave. Also, the main positive deflection of the wavelet 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.
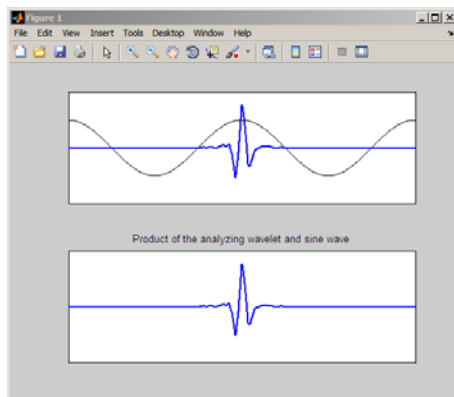
The product shown in the bottom pane closely resembles the analyzing wavelet. Integrating this product results in a CWT coefficient near zero.

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;
```
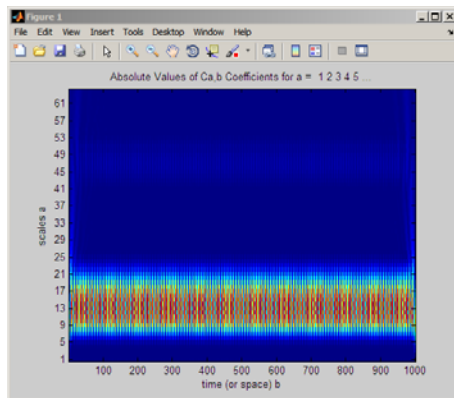

Note that the CWT coefficients are large in absolute value around scales 9 to 21. You can find the pseudo-frequencies corresponding to these scales using the command:

```
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);
```
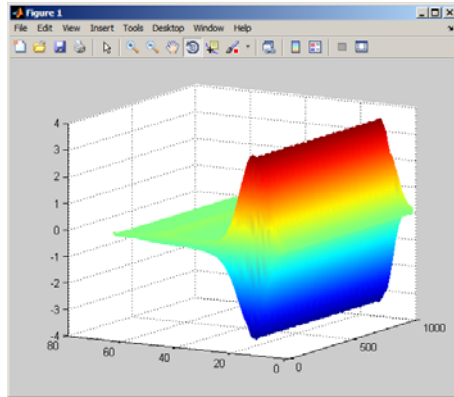
The final example constructs a signal consisting of both abrupt transitions and smooth oscillations. The signal is a 4-Hz sinusoid with two introduced discontinuities.

```
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;
```
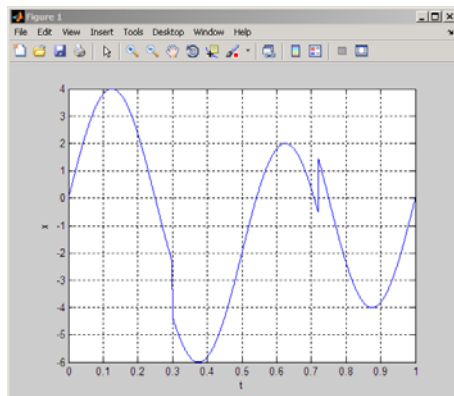

Note the discontinuities near t=0.3 and t=0.7.

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');
```
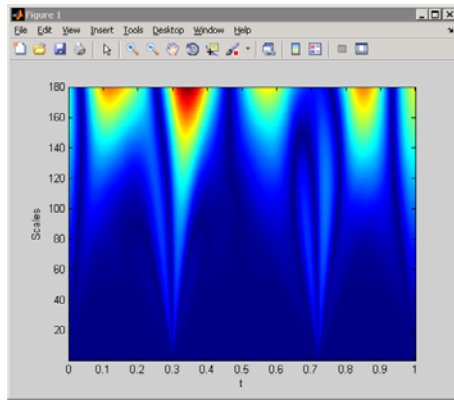

Note that the CWT detects both the abrupt transitions and oscillations in the signal. The abrupt transitions affect the CWT coefficients at all scales and clearly separate themselves from smoother signal features at small scales. On the other hand, the maxima and minima of the 2–Hz sinusoid are evident 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.

Redundancy in the Continuous Wavelet Transform

Any signal processing performed on a computer using real-world data must be performed on a discrete signal — that is, on a signal that has been measured at discrete time. So what exactly is "continuous" about the CWT?

What's "continuous" about the CWT, and what distinguishes it from the discrete wavelet transform (to be discussed in the following section), is the set of scales and positions at which it operates.

Unlike the discrete wavelet transform, the CWT can operate at every scale, from that of the original signal up to some maximum scale that you determine by trading off your need for detailed analysis with available computational horsepower.

The CWT is also continuous in terms of shifting: during computation, the analyzing wavelet is shifted smoothly over the full domain of the analyzed function.
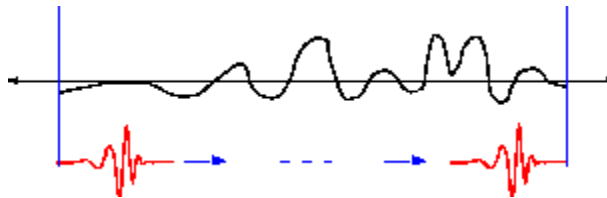
Critically-Sampled Discrete Wavelet Transform

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 1-71). 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.
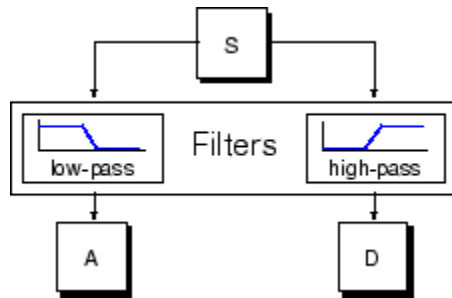
The original signal, S, passes through two complementary filters and emerges as two signals.

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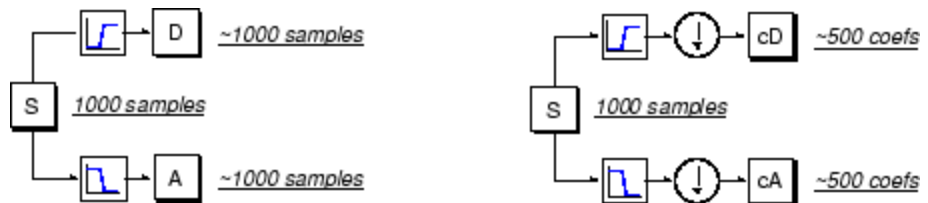
The process on the right, which includes downsampling, produces DWT coefficients.

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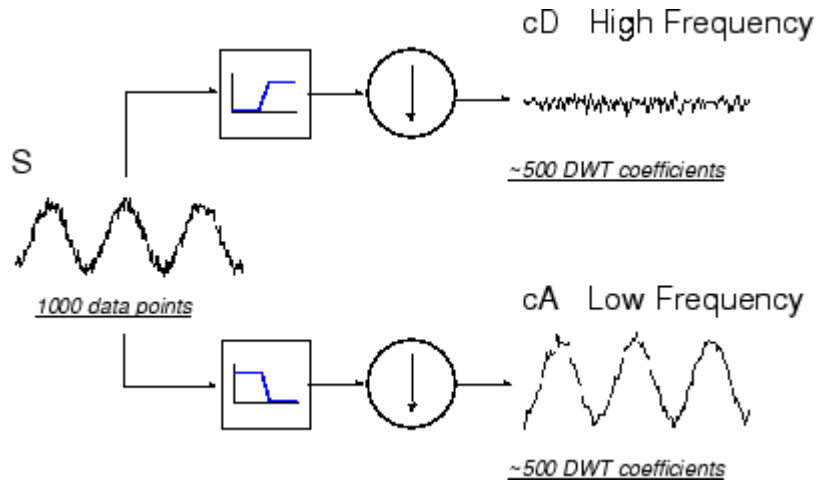
The MATLAB code needed to generate s, cD, and cA is

```
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*.
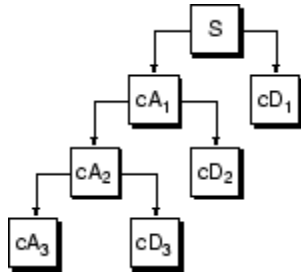
Looking at a signal's wavelet decomposition tree can yield valuable information.
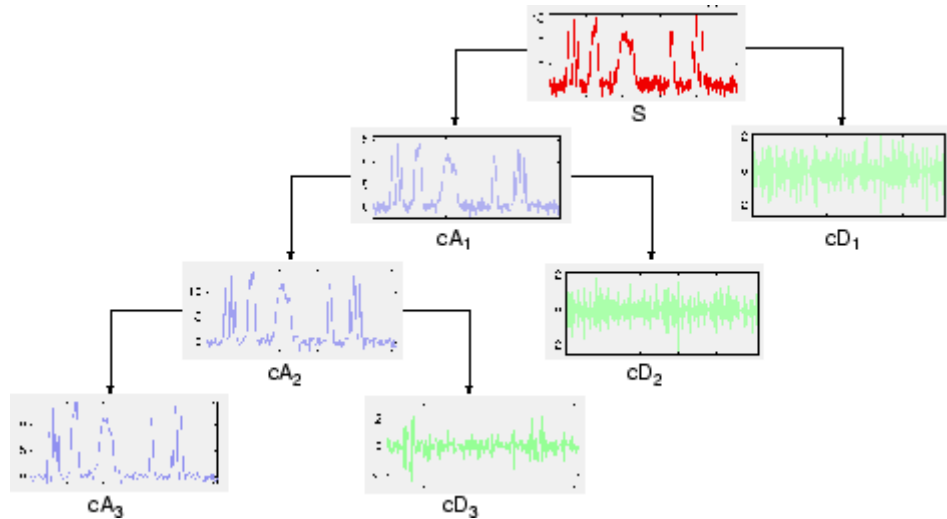
Number of Levels

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.
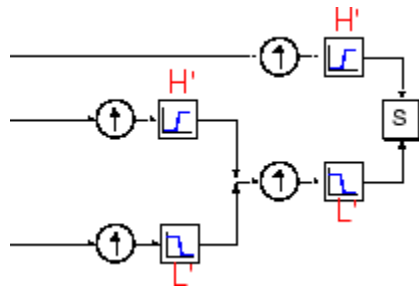
Where wavelet analysis involves filtering and downsampling, the wavelet reconstruction process consists of upsampling and filtering. Upsampling is the process of lengthening a signal component by inserting zeros between samples.
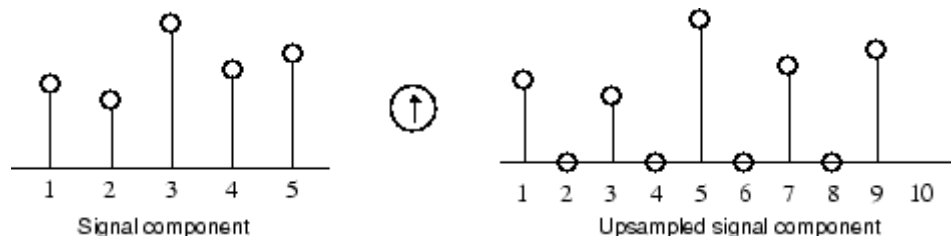
The toolbox includes commands, like `idwt` and `waverec`, that perform single-level or multilevel reconstruction, respectively, on the components of one-dimensional signals. These commands have their two-dimensional and three-dimensional analogs, `idwt2`, `waverec2`, `idwt3`, and `waverec3`.

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:*

Decomposition   Reconstruction
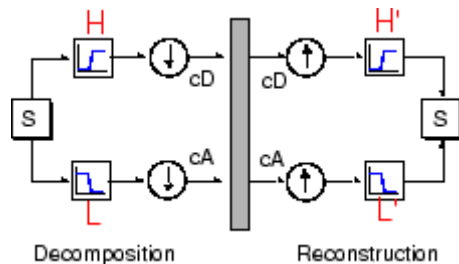
Reconstructing Approximations and Details

We have seen that it is possible to reconstruct our original signal from the coefficients of the approximations and details.
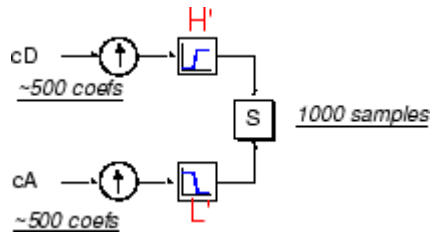
It is also possible to reconstruct the approximations and details themselves from their coefficient vectors. As an example, let's consider how we would reconstruct the first-level approximation A1 from the coefficient vector cA1.

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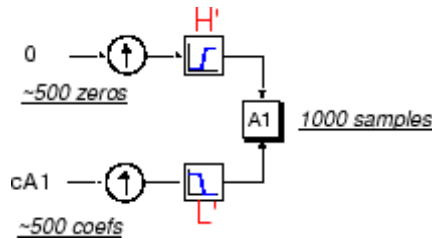
The process yields a reconstructed *approximation* A1, which has the same length as the original signal S and which is a real approximation of it.

Similarly, we can reconstruct the first-level detail D1, using the analogous process:
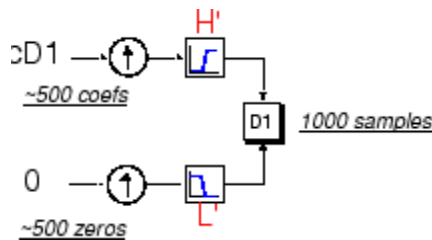
The reconstructed details and approximations are true constituents of the original signal. In fact, we find when we combine them that

$$A_1 + D_1 = S.$$

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:

$$S = A_1 + D_1$$
$$= A_2 + D_2 + D_1$$
$$= A_3 + D_3 + D_2 + D_1$$
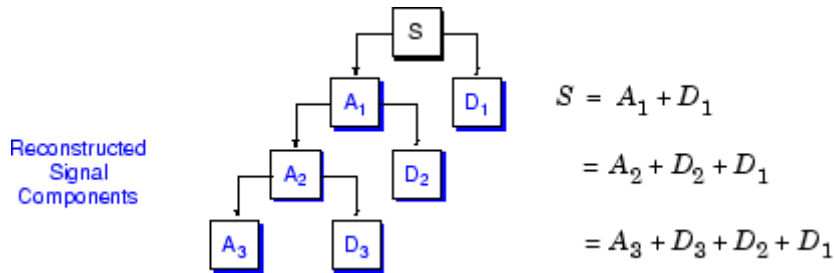
Wavelets From Conjugate Mirror Filters

In the section "Reconstruction Filters" on page 1-56, 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
```
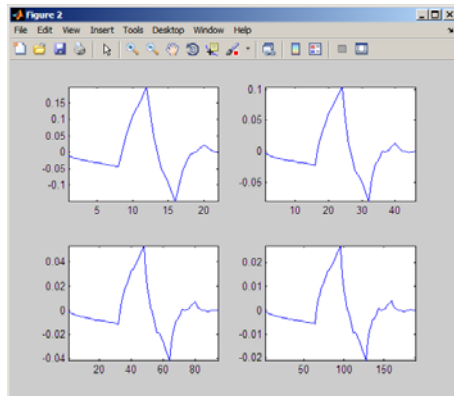

The curve begins to look progressively more like the db2 wavelet. This means that the wavelet's shape is determined entirely by the coefficients of the reconstruction filters.

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 Packet Analysis

The *wavelet packet* method is a generalization of wavelet decomposition that offers a richer range of possibilities for signal analysis.

In wavelet analysis, a signal is split into an approximation and a detail. The approximation is then itself split into a second-level approximation and detail, and the process is repeated. For an n-level decomposition, there are n+1 possible ways to decompose or encode the signal.

$$S = A_1 + D_1$$
$$= A_2 + D_2 + D_1$$
$$= A_3 + D_3 + D_2 + D_1$$
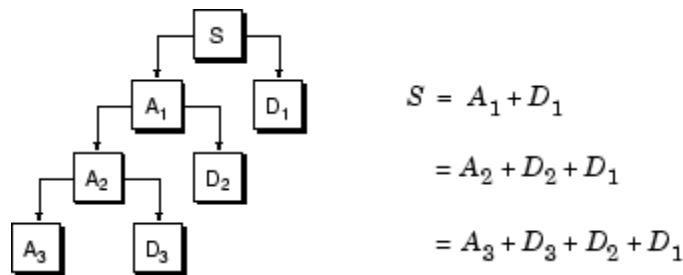
In wavelet packet analysis, the details as well as the approximations can be split.

This yields more than $2^{2^{n-1}}$ different ways to encode the signal. This is the *wavelet packet decomposition tree*.
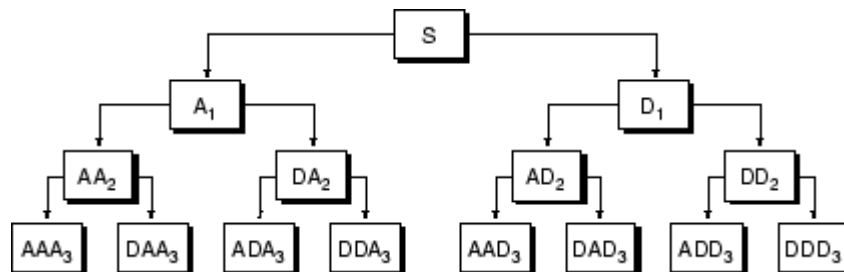
The wavelet decomposition tree is a part of this complete binary tree.

For instance, wavelet packet analysis allows the signal S to be represented as A1 + AAD3 + DAD3 + DD2. This is an example of a representation that is not possible with ordinary wavelet analysis.

Choosing one out of all these possible encodings presents an interesting problem. In this toolbox, we use an *entropy-based criterion* to select the most suitable decomposition of a given signal. This means we look at each node of the decomposition tree and quantify the information to be gained by performing each split.

Simple and efficient algorithms exist for both wavelet packet decomposition and optimal decomposition selection. This toolbox uses an adaptive filtering algorithm, based on work by Coifman and Wickerhauser (see [CoiW92] in "References" on page 1-71), with direct applications in optimal signal coding and data compression.

Such algorithms allow the **Wavelet Packet 1-D** and **Wavelet Packet 2-D** tools to include "Best Level" and "Best Tree" features that optimize the decomposition both globally and with respect to each node.

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 1-64
- "Daubechies" on page 1-65
- "Biorthogonal" on page 1-65
- "Coiflets" on page 1-67
- "Symlets" on page 1-67
- "Morlet" on page 1-68
- "Mexican Hat" on page 1-68
- "Meyer" on page 1-69
- "Other Real Wavelets" on page 1-69
- "Complex Wavelets" on page 1-69

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

1 Type `wavemenu` at the MATLAB command line. The **Wavelet Toolbox Main Menu** appears.

2 Click the **Wavelet Display** menu item. The **Wavelet Display** tool appears.

3 Select a family from the **Wavelet** menu at the top right of the tool.

4 Click the **Display** button. Pictures of the wavelets and their associated filters appear.

5 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.

Wavelet function psi
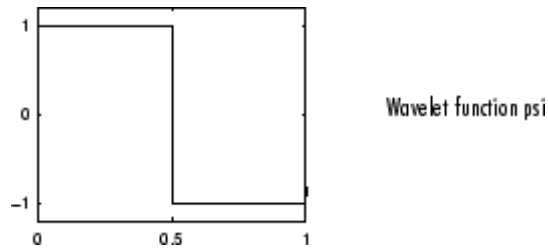
Daubechies

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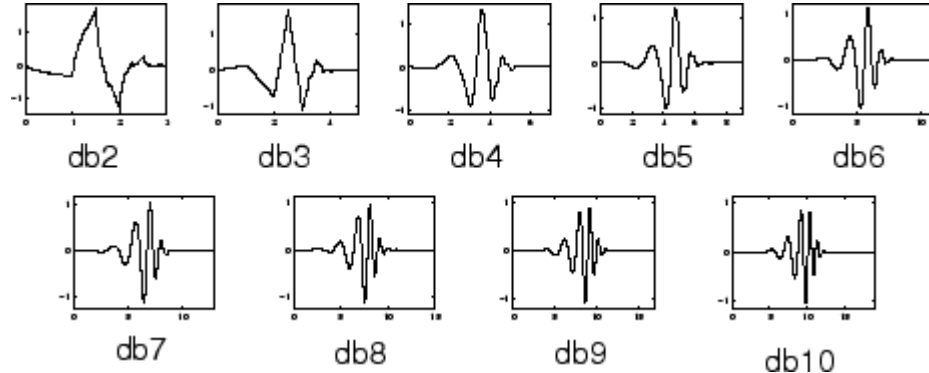
You can obtain a survey of the main properties of this family by typing waveinfo('db') from the MATLAB command line. See "Daubechies Wavelets: dbN" in the *Wavelet Toolbox User's Guide* for more detail.

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 $2N$ moments equal to 0 and the scaling function has $2N$-1 moments equal to 0. The two functions have a support of length $6N$-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.

coif1   coif2   coif3   coif4   coif5
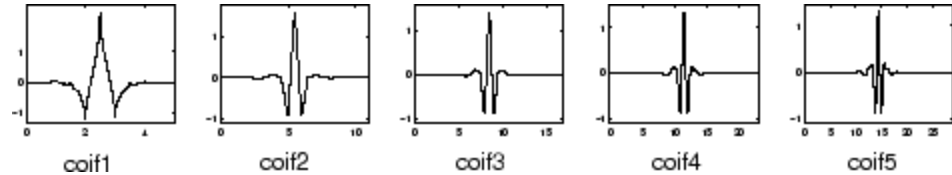
Symlets

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.

sym2   sym3   sym4   sym5

sym6   sym7   sym8
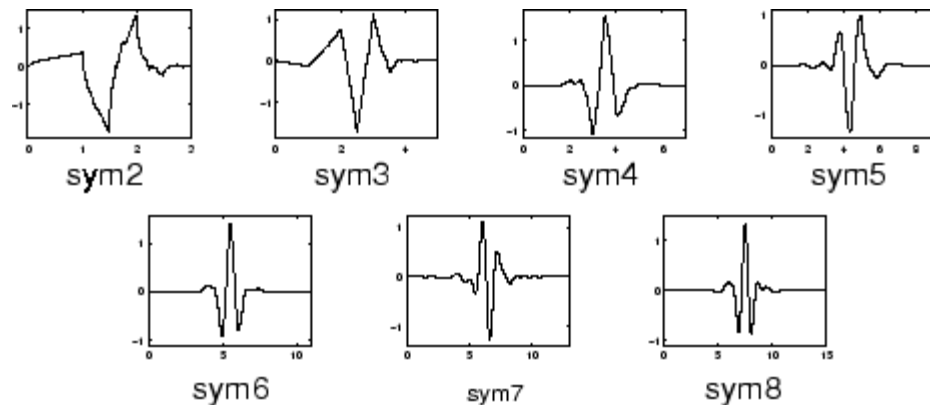
You can obtain a survey of the main properties of this family by typing `waveinfo('sym')` from the MATLAB command line. See "Symlet Wavelets: symN" in the *Wavelet Toolbox User's Guide* for more detail.

Morlet

This wavelet has no scaling function, but is explicit.

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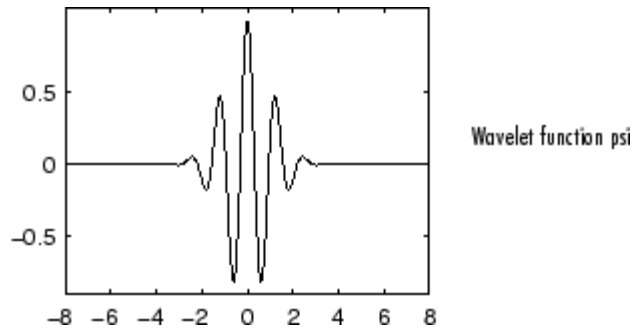
You can obtain a survey of the main properties of this family by typing `waveinfo('morl')` from the MATLAB command line. See "Morlet Wavelet: morl" in the *Wavelet Toolbox User's Guide* for more detail.

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.

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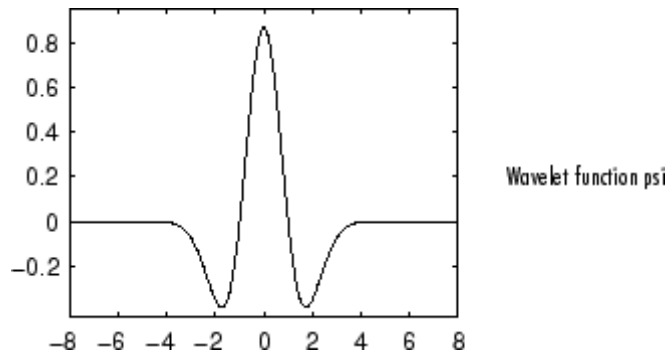
You can obtain a survey of the main properties of this family by typing `waveinfo('mexh')` from the MATLAB command line. See "Mexican Hat Wavelet: mexh" in the *Wavelet Toolbox User's Guide* for more information.

Meyer

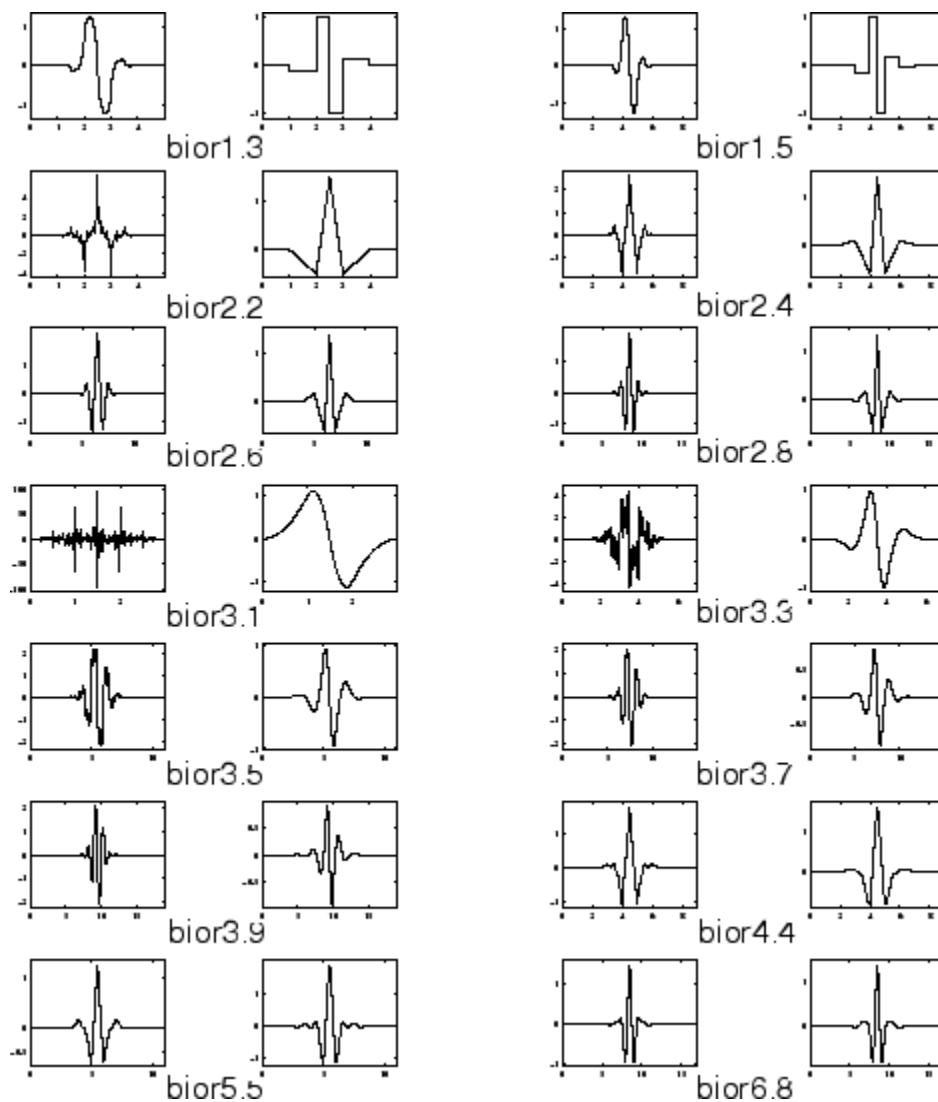The Meyer wavelet and scaling function are defined in the frequency domain.

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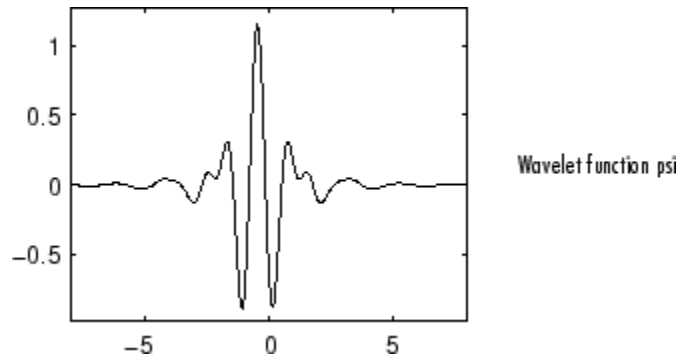
You can obtain a survey of the main properties of this family by typing `waveinfo('meyer')` from the MATLAB command line. See "Meyer Wavelet: meyr" in the *Wavelet Toolbox User's Guide* for more detail.

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.), *Festchrift 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 thesholding," *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 opéateurs*, 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.

[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/`

2

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 GUIs 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 one-dimensional signal
- Perform discrete analysis and synthesis of one- and two-dimensional signals
- Perform wavelet packet analysis of one- and two-dimensional 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 graphical user interface tools described in this chapter 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.
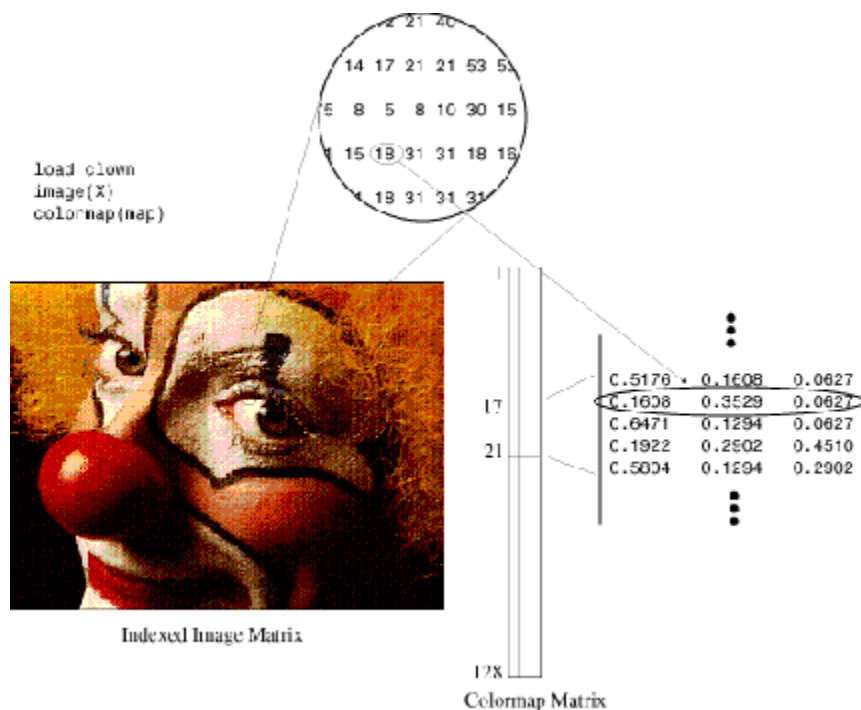
Outside MATLAB, indexed images with n colors often contain values from 0 to n–1. These values are indices into a colormap with 0 as its first index. Since MATLAB matrices start with index 1, you must increment each value in the image, or *shift up* the image, to create an image that you can manipulate 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 graphical user interface tools display 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 graphical user interface tools follow 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 two-dimensional 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:

- Two-Dimensional Discrete Wavelet Analysis
- Two-Dimensional Wavelet Packet Analysis
- Two-Dimensional Stationary Wavelet Analysis
- Two-Dimensional Extension tool

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
map	64x3	1536	double array

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


Original Color Indexed Image

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
```


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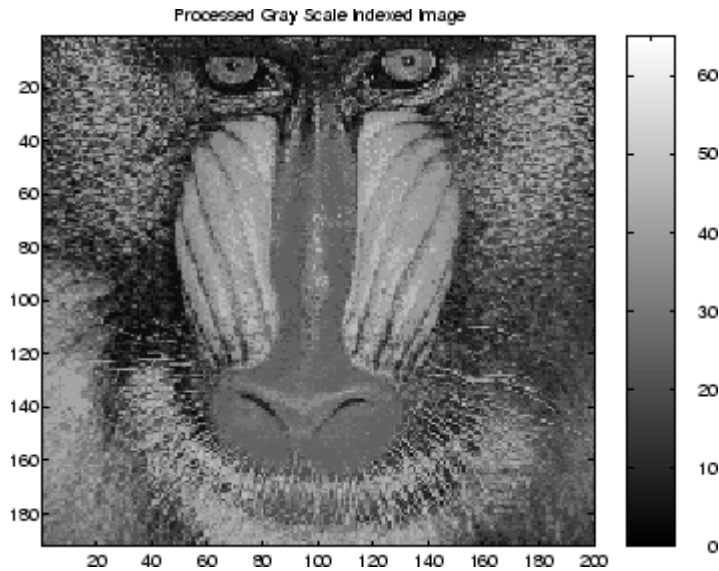
The color bar of the converted image is now linear and has a smooth transition from dark to light. The image is now suitable for wavelet decomposition.

Finally, save the converted image in a form compatible with the Wavelet Toolbox graphical user interface:

```
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.

One-Dimensional Wavelet Density Estimation

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

The toolbox provides a graphical interface tool 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')
```

One-Dimensional Estimation Using the Graphical Interface

1 Start the Density Estimation 1-D Tool.

From the MATLAB prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.

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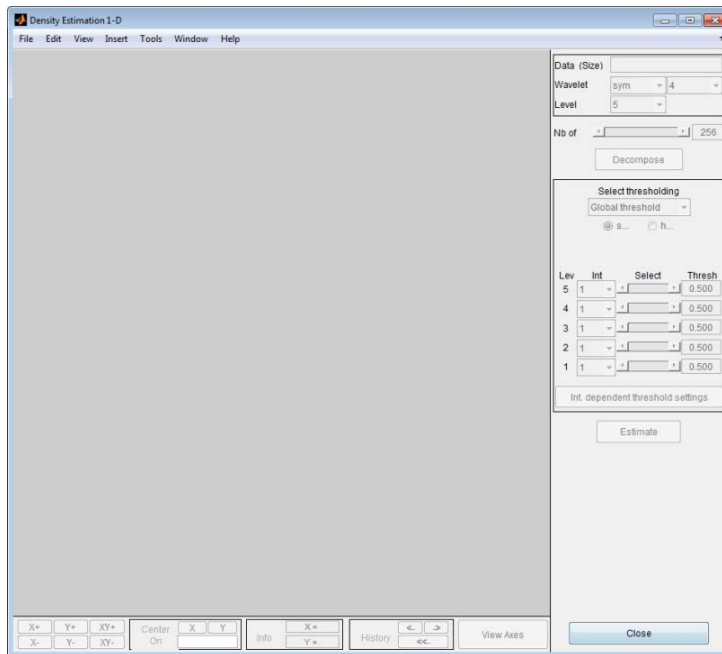
2 Load data.

From the **File** menu, choose the **Load > Data for Density Estimate** option.

When the **Load data for Density Estimate** dialog box appears, select the MAT-file ex1cusp1.mat from the MATLAB folder toolbox/wavelet/wavedemo. Click **OK**. The noisy cusp data is loaded into the **Density Estimation 1-D** tool.

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 yellow dotted lines running horizontally through the graphs on the left of the window.

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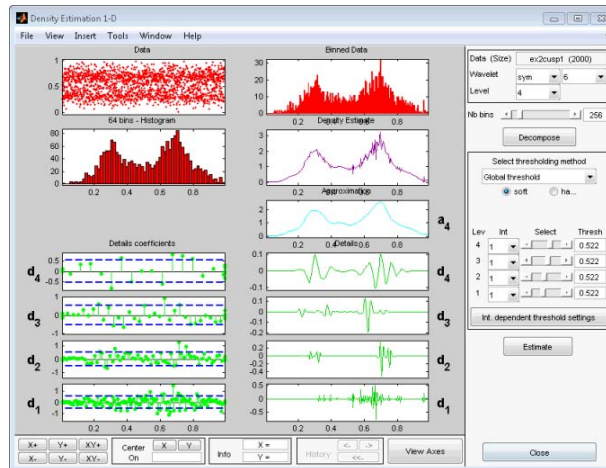
You can see that the estimation process delivers a very irregular resulting density. The density estimate (in yellow) is the normalized sum of the signals located below it: the approximation a4 and the reconstructed details after coefficient thresholding.

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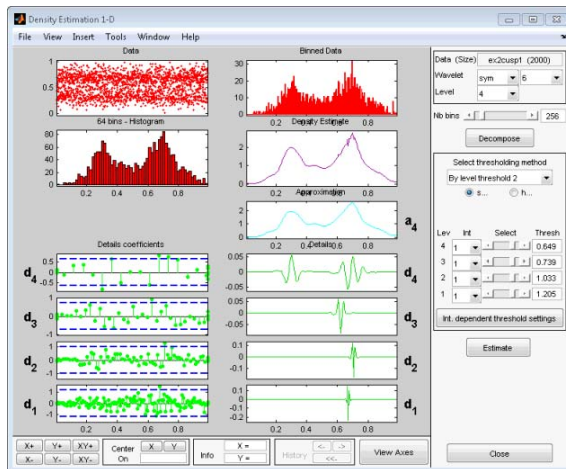
The estimated density is more satisfactory. It correctly identifies the smooth part of the density and the cusp at 0.7.

Importing and Exporting Information from the Graphical Interface

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
thrParams	1x4	464	cell array
wname	1x4	8	char array

Name	Size	Bytes	Class
xdata	1x250	2000	double array
ydata	1x250	2000	double array

The estimated density is given by xdata and ydata. The length of these vectors is of the same as 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 "One-Dimensional 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 one-dimensional 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')
```

Interactive 1-D Wavelet Coefficient Selection

This section takes you through the features of one-dimensional selection of wavelet coefficients using one of the Wavelet Toolbox specialized tools. The toolbox provides a graphical interface tool 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

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.

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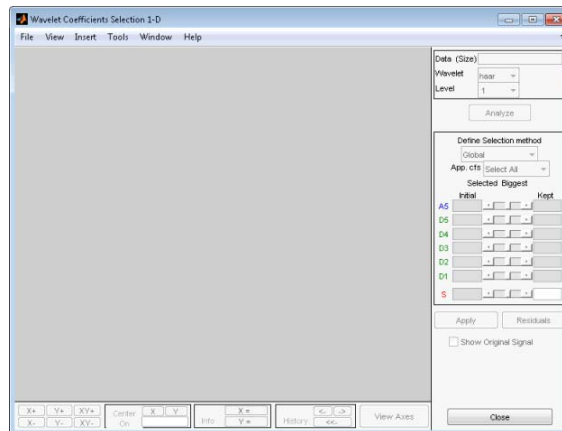
2 Load data.

From the **File** menu, choose the **Load Signal** option.

When the **Load Signal** dialog box appears, select the MAT-file
noisbump.mat, which should reside in the MATLAB folder
toolbox/wavelet/wavedemo. Click the **OK** button. The noisy bumps data
is loaded into the **Wavelet Coefficients Selection 1-D** tool.

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.
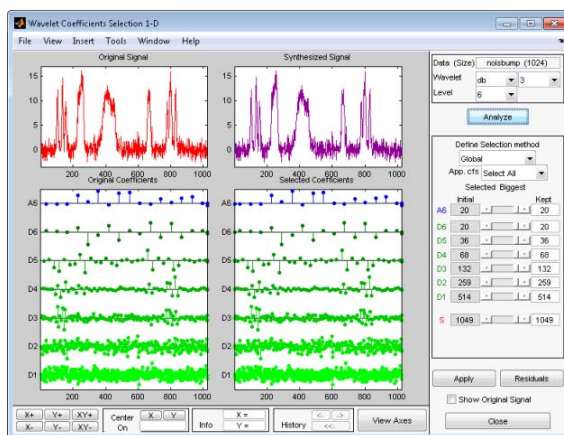
The tool displays below the original signal (on the left) its wavelet
decomposition: the approximation coefficients A6 and detail coefficients
from D6 at the top to D1 at the bottom. In the middle of the window, below
the synthesized signal (which at this step is the same, since all the wavelet
coefficients 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.
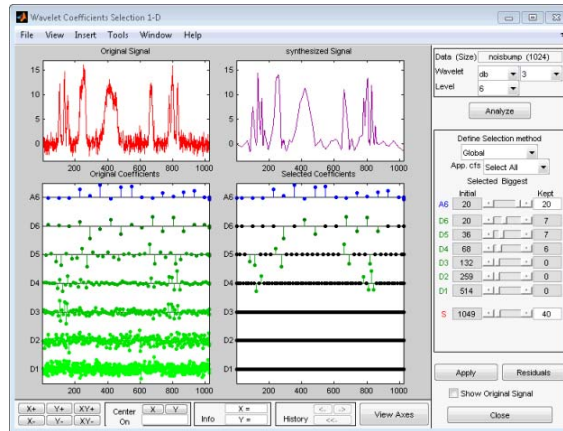
In the previous trial, the approximation coefficients were all kept. It is possible to relax this constraint by selecting another option from the **App. cfs** menu (Approximation Coefficients abbreviation). Choose the **Unselect** option and click the **Apply** button.
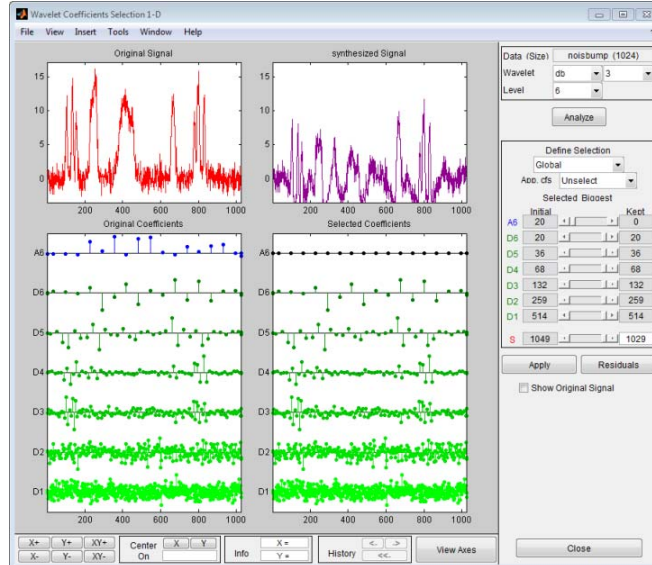

None of the approximation coefficients are kept.

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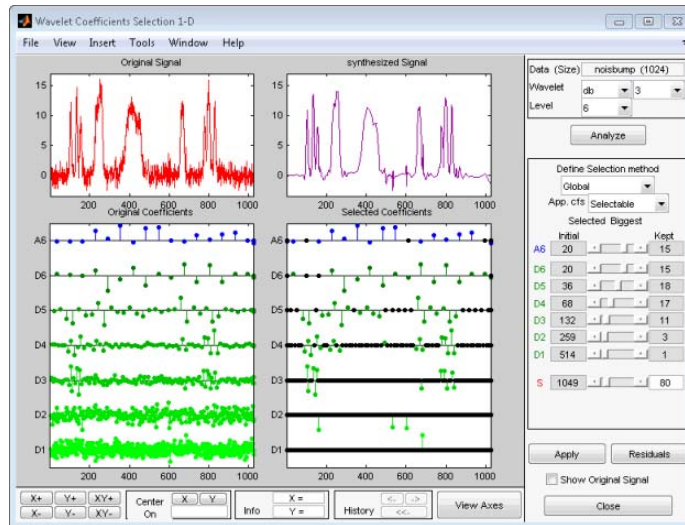
Some of the approximation coefficients (15) have been kept.

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.
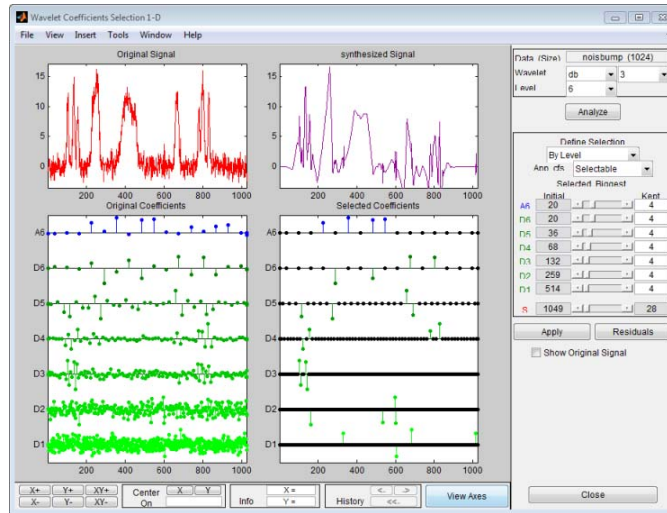
Selecting Coefficients Manually

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
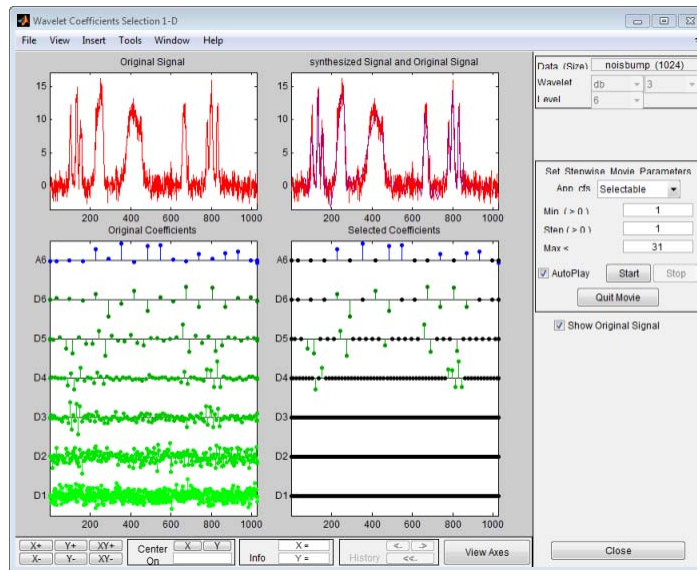coefficient is visible and the synthesized signal is null.
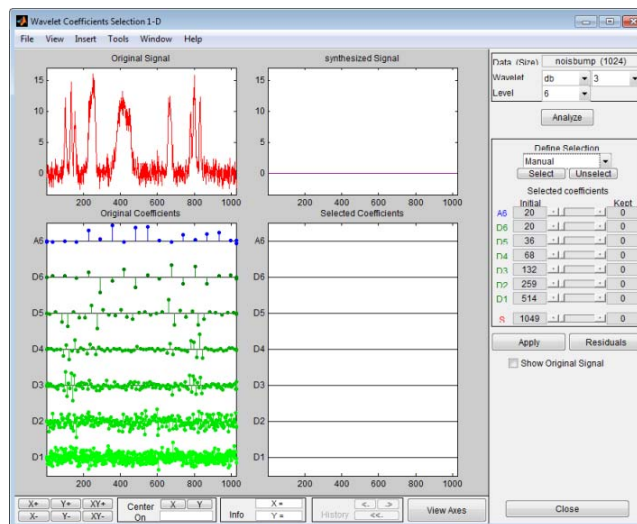
Select 16 coefficients individually by double clicking each of them using the left mouse button. The color of selected coefficients switches from green to yellow for the details and from blue to yellow for the approximation, which appear on the left of the window and appear in yellow on the middle part. Click the **Apply** button.
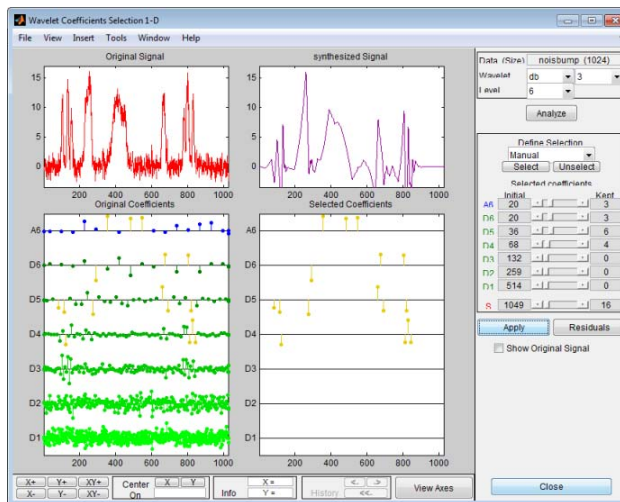
You can deselect the currently selected coefficients by double clicking each of them. Another way to select or deselect a set of coefficients is to use the selection box. Drag a rubber band box (hold down the left mouse button) over a portion of the coefficient axes (original or selected) containing 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')
```

Interactive 2-D Wavelet Coefficient Selection

This section takes you through the features of two-dimensional selection of wavelet coefficients using one of the Wavelet Toolbox specialized tools. The toolbox provides a graphical interface tool 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 one-dimensional 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

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.

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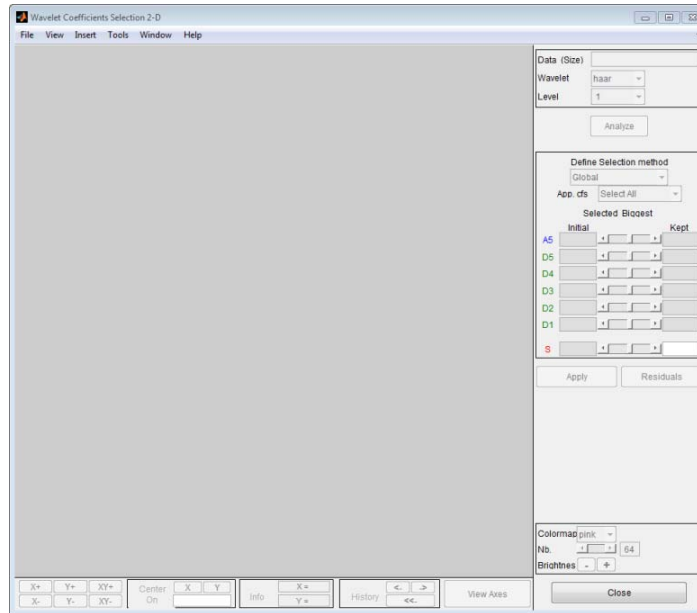
2 Load data.

From the **File** menu, choose the **Load Image** option.

When the **Load Image** dialog box appears, select the MAT-file
`noiswom.mat`, which should reside in the MATLAB folder
`toolbox/wavelet/wavedemo`. Click the **NO** to load the grayscale 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.
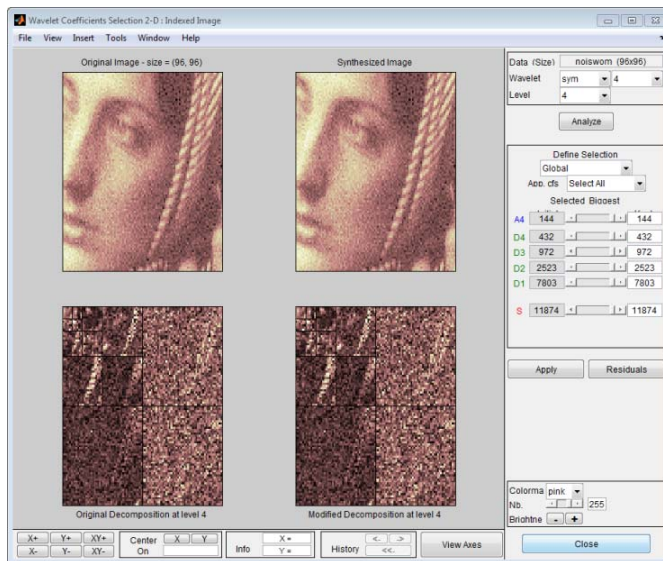
The tool displays its wavelet decomposition below the original image (on the left). The selected coefficients are displayed in the middle of the window, below the synthesized image (which, at this step, is the same since all the wavelet coefficients are kept). There are 11874 coefficients, a little bit 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.
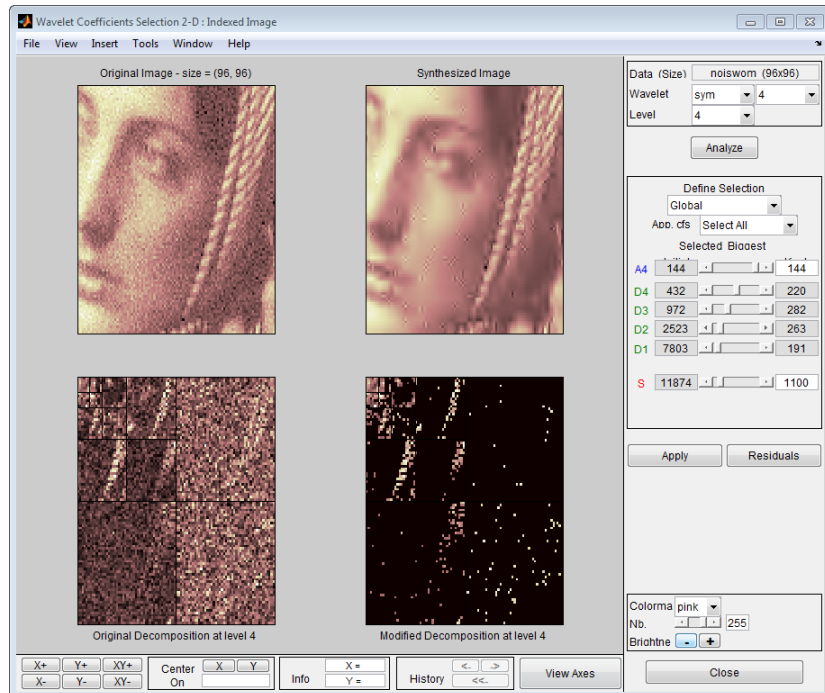
In the previous operation, all the approximation coefficients were kept. It is possible to relax this constraint by selecting another option from the **App. cfs** menu (see "Interactive 1-D Wavelet Coefficient Selection" on page 2-16).

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.
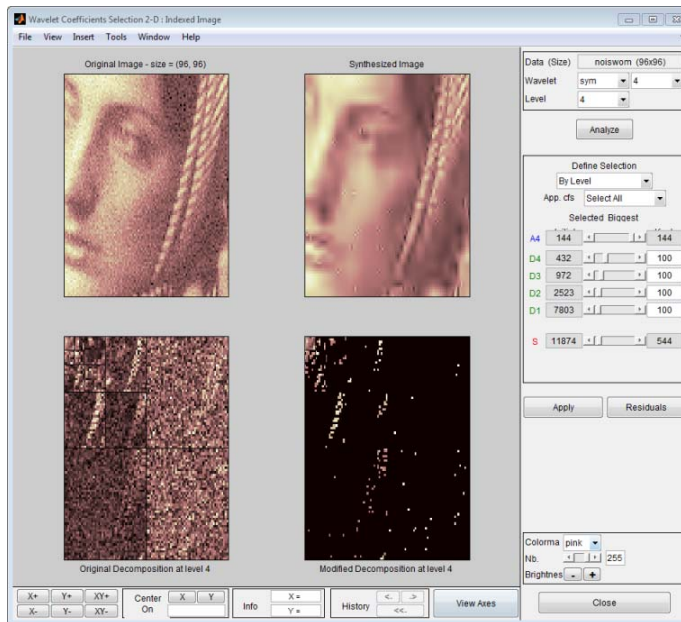
Selecting Coefficients Automatically

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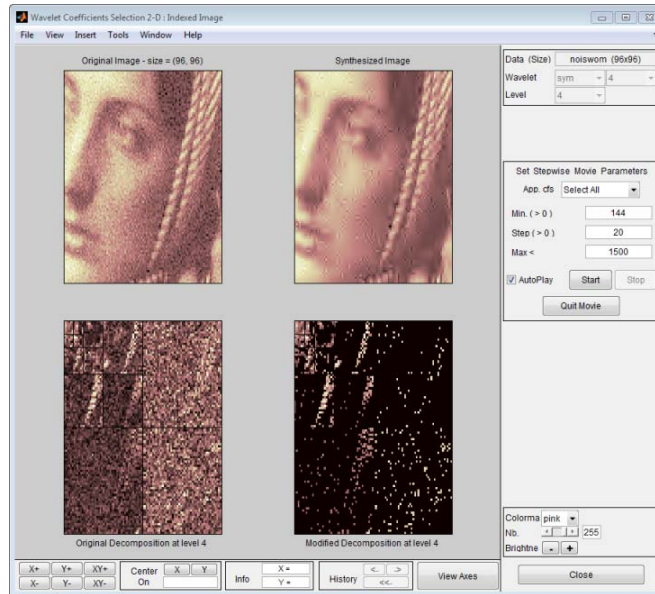
We've stopped the movie at 864 coefficients (including the number of approximation coefficients).

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')
```

One-Dimensional Extension

This section takes you through the features of one-dimensional extension or truncation using one of the Wavelet Toolbox utilities.

One-Dimensional Extension Using the Command Line

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

One-Dimensional Extension Using the Graphical Interface

1 Start the Signal Extension Tool.

From the MATLAB prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.

Click the **Signal Extension** menu item.

2 Load data.

From the **File** menu, choose the **Load Signal** option.

When the **Load Signal** dialog box appears, select the MAT-file `noisbloc.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`. Click the **OK** button. The noisy blocks data is loaded into the **Signal Extension** tool.
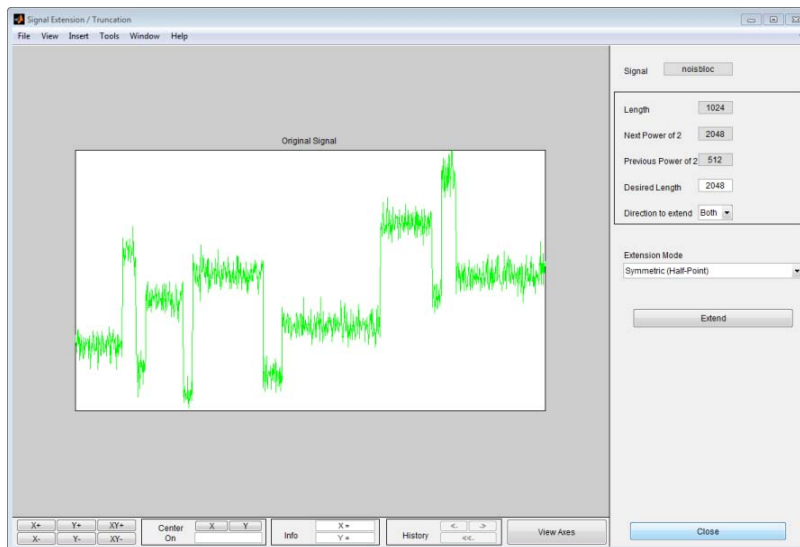
3 Extend the signal.

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.
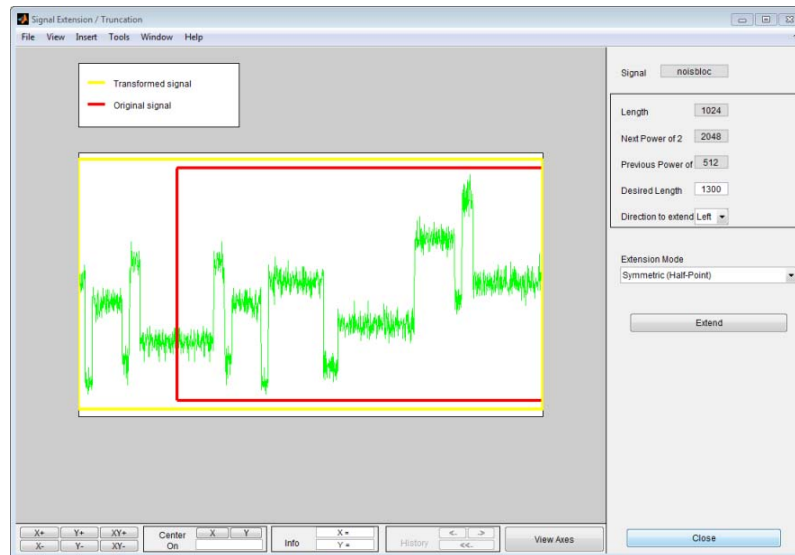
The tool displays the original signal delimited by a red box and the transformed signal delimited by a yellow box. The signal has been extended by left symmetric boundary values replication.

Select the **Both** option from the **Direction to extend** menu and select the **Continuous** option from the **Extension mode** menu. Click the **Extend** button.
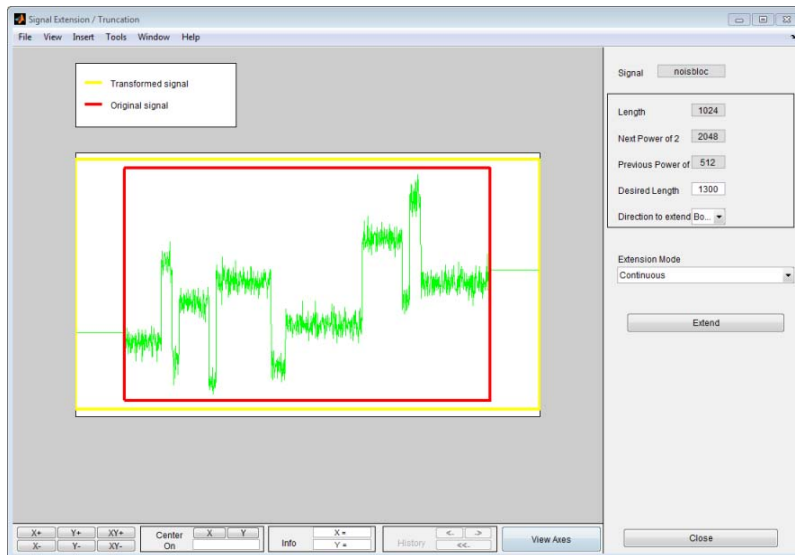
The signal is extended in both directions by replicating the first value to the left and the last value to the right, respectively.

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.
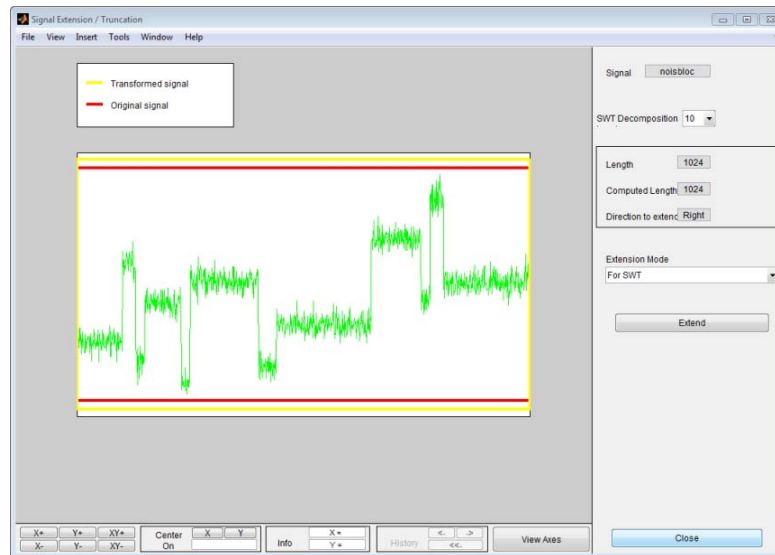
Since the signal is of length 1024 = 2^10, no extension is needed so the **Extend** button is ineffective.

From the **File** menu, choose the **Example Extension** option and select the last item of the list.
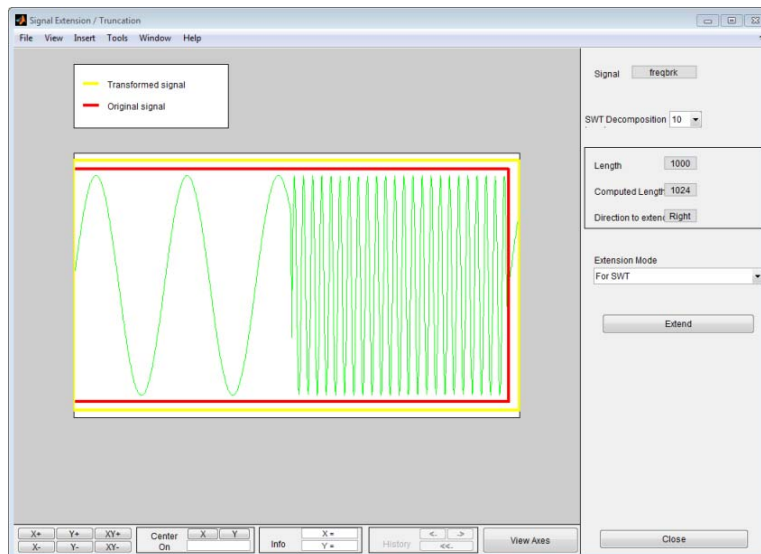
Since the signal is of length 1000 and the decomposition level needed for SWT is 10, the tool performs a minimal right periodic extension. The extended signal is of length 1024.

Select **4** from the **SWT Decomposition Level** menu, and then click the **Extend** button. The tool performs a minimal right periodic extension leading to an extended signal of length 1008 (because 1008 is the smallest integer greater than 1000 divisible by $2^4 = 16$).

Select **2** from the **SWT Decomposition Level** menu. Since 1000 is divisible by 4, no extension is needed.

Truncating Signal

The same tool allows you to truncate a signal.

Since truncation is not allowed for the special mode **For SWT**, select the **Periodic** option from the **Extension mode** menu. Type 900 for the desired length and press **Enter**. Click the **Truncate** button.
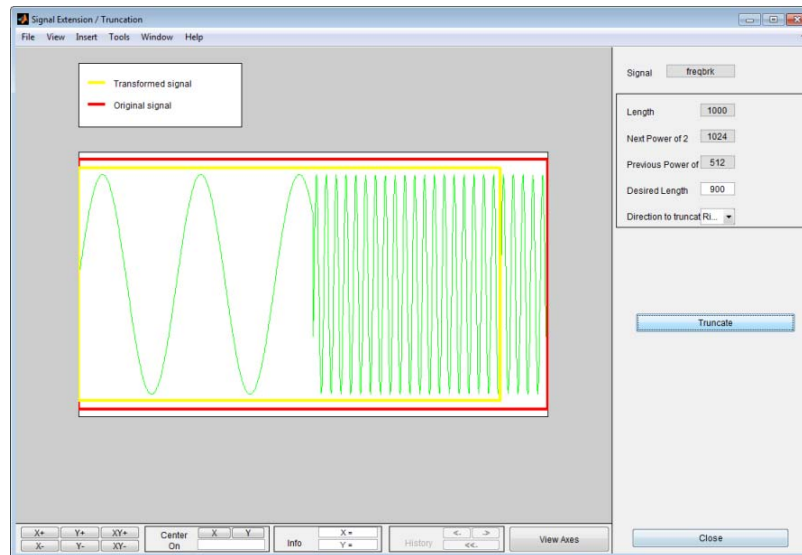
The tool displays the original signal delimited by a red box and the truncated signal delimited by a yellow box. The signal has been truncated by deleting 100 values on the right side.

Importing and Exporting Information from the Graphical Interface

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

To save the transformed signal, use the menu option **File > Save Transformed Signal**. A dialog box appears that lets you specify a folder and filename for storing the image. Type the name tfrqbrk. After saving the signal data to the file tfrqbrk.mat, load the variable into your workspace:

```
load tfrqbrk
whos
```

Name	Size	Bytes	Class
tfrqbrk	1x900	7200	double array

Two-Dimensional Extension

This section takes you through the features of two-dimensional extension or truncation using one of the Wavelet Toolbox utilities. This section is short since it is very similar to "One-Dimensional Extension" on page 2-32.

Two-Dimensional Extension Using the Command Line

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

Two-Dimensional Extension Using the Graphical Interface

1 Start the Image Extension Tool.

From the MATLAB prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.

Click the **Image Extension** menu item.

2 Extend (or truncate) the image.

From the **File** menu, choose the **Example Extension** option and select the first item of the list.
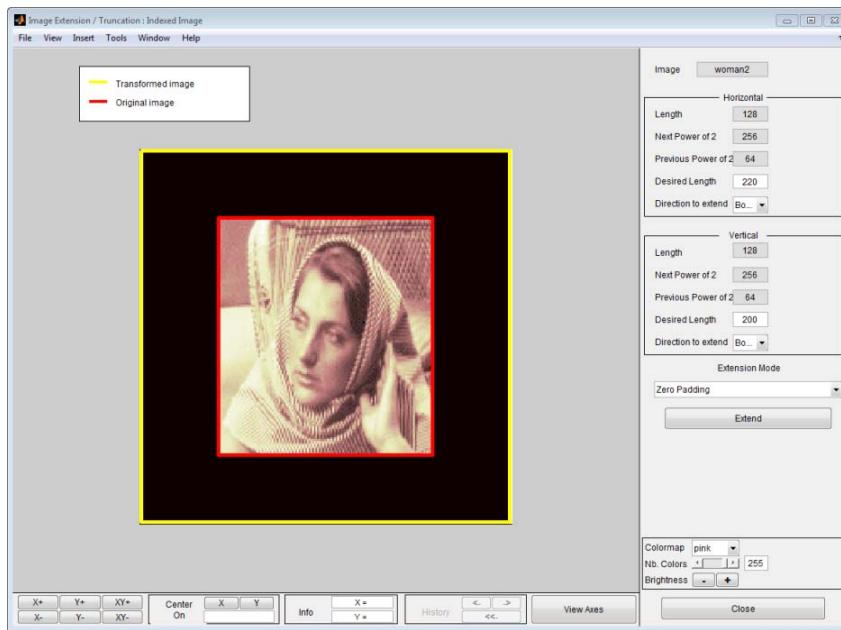
The tool displays the original image delimited by a red box and the transformed image delimited by a yellow box. The image has been extended by zero padding. The right part of the window allows you to control the parameters of the extension/truncation process for the vertical and horizontal directions, respectively. The possibilities are similar to the one-dimensional ones described in "One-Dimensional Extension" on page 2-32.

To see some more extension cases, look at the examples of the toolbox (using the `wavedemo` command).

Importing and Exporting Information from the Graphical Interface

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

To save the transformed image, use the menu option **File > Save Transformed Image**.

A dialog box appears that lets you specify a folder and filename for storing the image. Type the name woman2. After saving the image data to the file woman2.mat, load the variable into your workspace:

```
load woman2
whos
```

Name	Size	Bytes	Class
woman2	200x220	352000	double array
map	253x3	6120	double array

The transformed image is stored together with its colormap.

Image Fusion

This section takes you through the features of Image Fusion, one of the Wavelet Toolbox specialized tools.

For the examples in this section, 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'll learn how to

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

Since you can perform analyses either from the command line or using the graphical interface tools, this section has subsections covering each method.

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 1-71).

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')
```
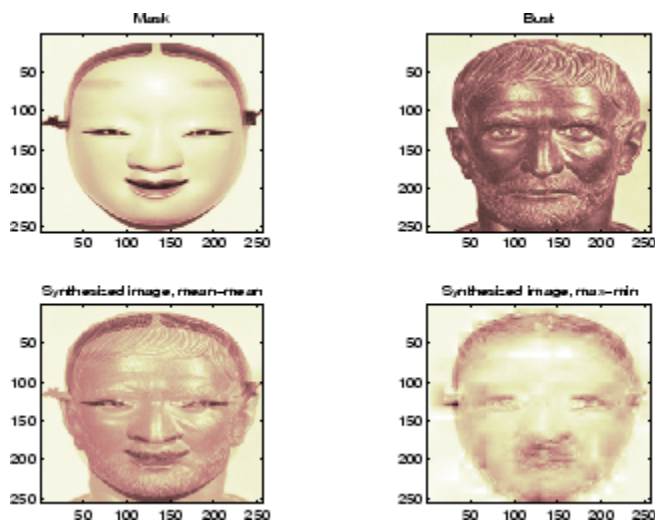
Example 2: Restoration by Fusion from Fuzzy Images

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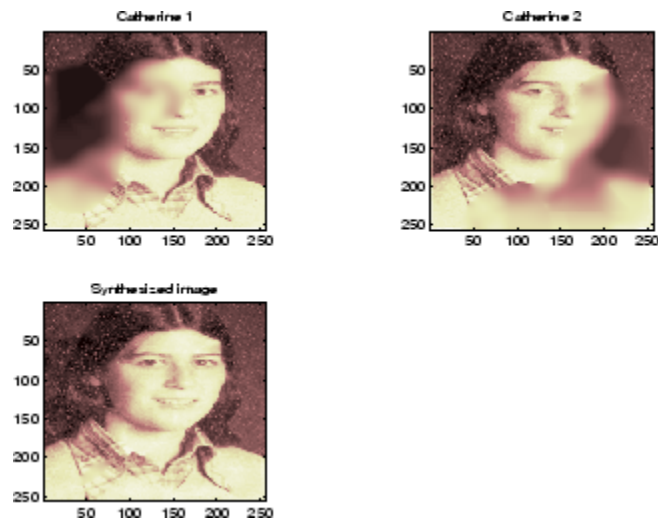
The synthesized image is a restored version of good quality of the common underlying original image.

Image Fusion Using the Graphical Interface

1 Start the Image Fusion Tool.

From the MATLAB prompt, type

```
wavemenu
```

to display the **Wavelet Toolbox Main Menu** and then click the **Image Fusion** menu item to display the Image Fusion Tool.
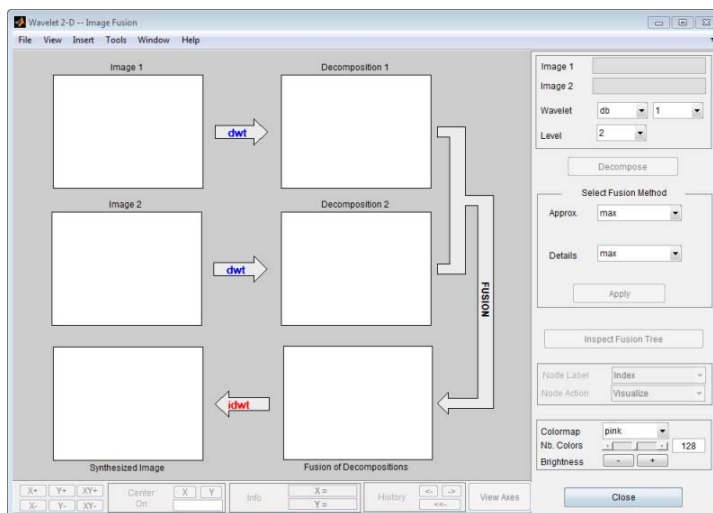
2 Load original images.

From the **File** menu, choose the **Load Image 1** option.

When the **Load Image 1** dialog box appears, select the MAT-file `mask.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`. Click **NO** to load as a grayscale image.

Perform the same sequence choosing the **Load Image 2** option and selecting the MAT-file `bust.mat`. Click **NO** to load as a grayscale 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.
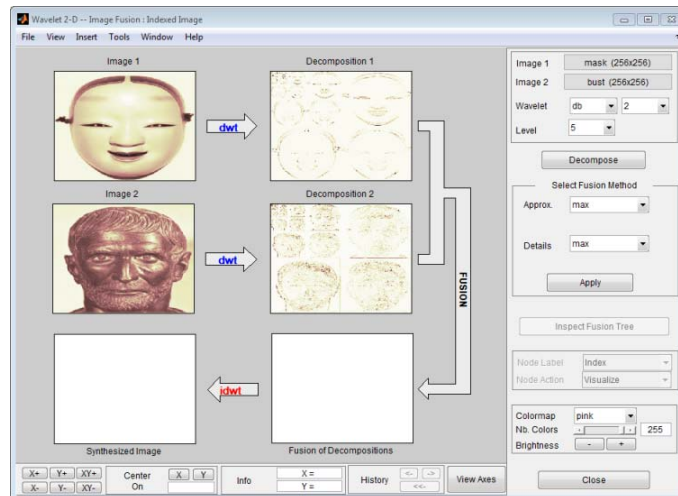
4 Merge two images from their decompositions.

From **Select Fusion Method** frame, select the item mean for both **Approx.** and **Details**. Next, click the **Apply** button.
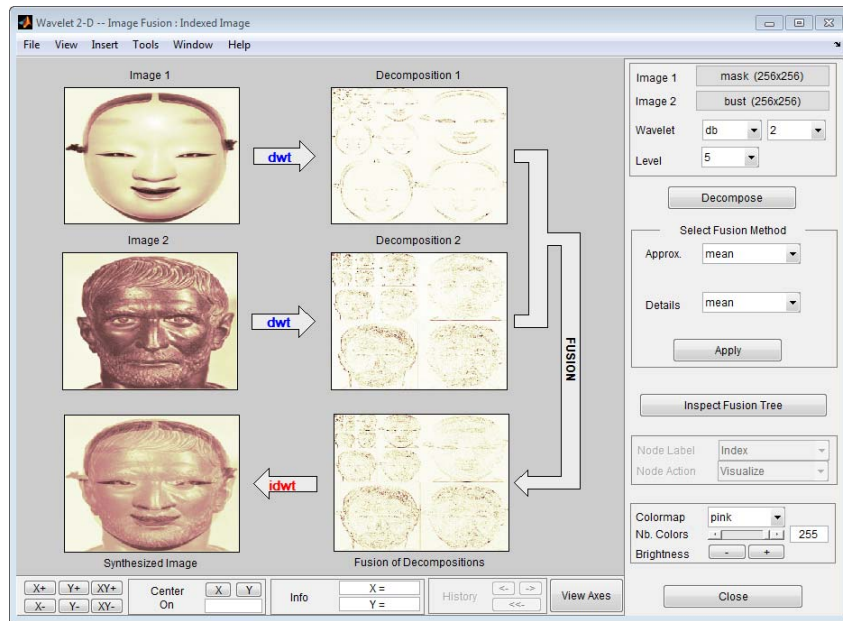
The synthesized image and its decomposition (which is equal to the fusion of the two decompositions) appear. The new image produced by fusion clearly exhibits features from the two original ones.

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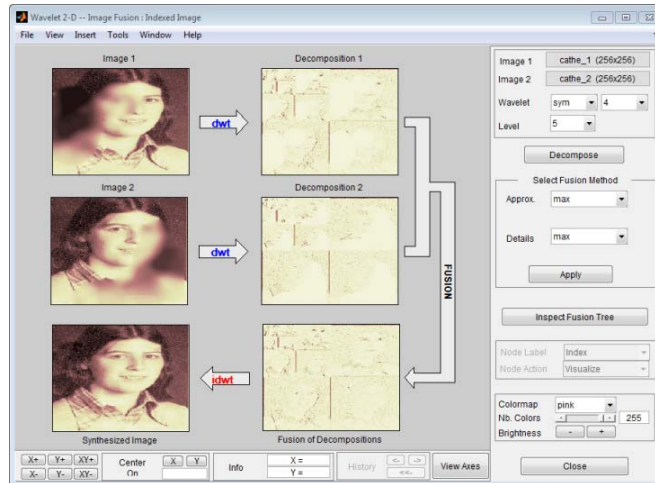
The synthesized image is a restored version of good quality of the common underlying original image.

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.

One-Dimensional Fractional Brownian Motion Synthesis

This section takes you through the features of One-Dimensional Fractional Brownian Motion Synthesis using one of the Wavelet Toolbox specialized tools.

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

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

The toolbox requires only one function to generate a fractional Brownian motion signal: wfbm. You'll find full information about this function in its reference page.

In this section, you'll learn how to

- Generate a fractional Brownian motion signal
- Look at its main properties
- Save the synthesized signal

Since you can perform the generation either from the command line or using the graphical interface tools, this section has subsections covering each method.

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 the white noise. The fBm is self-similar in distribution and the variance of the increments is given by

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

where v is a positive constant.

Fractional Brownian Motion Synthesis Using the Command Line

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.

Let us give an example of each situation using the wfbm file, which generates a sample path of this process.

```
% Generate fBm for H = 0.3 and H = 0.7

% Set the parameter H and the sample length
H = 0.3; lg = 1000;
% Generate and plot wavelet-based fBm for H = 0.3
fBm03 = wfbm(H,lg,'plot');

% Generate and plot wavelet-based fBm for H = 0.7
fBm07 = wfbm(H,lg,'plot');

% The last step is equivalent to
% Define wavelet and level of decomposition
% w = ' db10'; ns = 6;
% Generate
% fBm07 = wfbm(H,lg,'plot',w,ns);
```

It appears that fBm07 clearly exhibits a stronger low-frequency component and has, locally, a less irregular behavior.

Fractional Brownian Motion Synthesis Using the Graphical Interface

1 Start the Fractional Brownian Motion Synthesis Tool.

From the MATLAB prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears. Click **Fractional Brownian Generation 1-D** to display the One-Dimensional Fractional Brownian Motion Synthesis Tool.
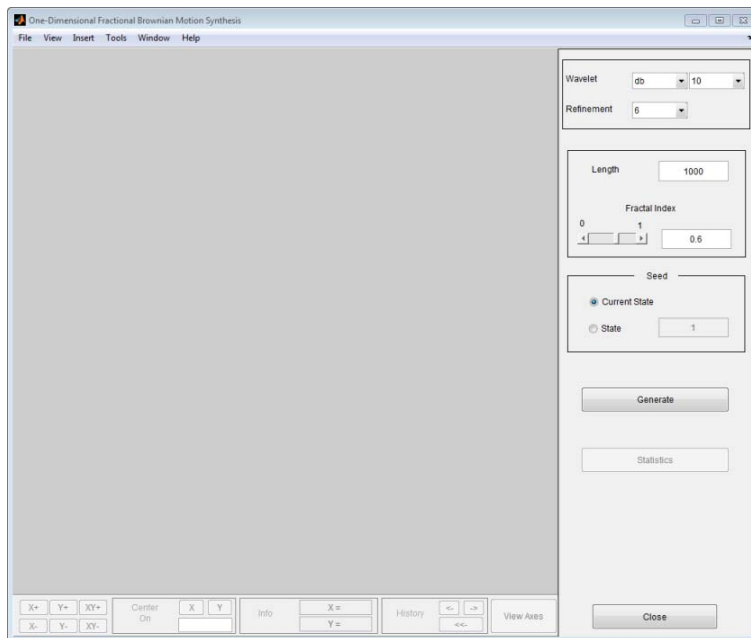
2 Generate fBm.

From the **Fractal Index** edit button, type 0.3 and from the **Seed** frame, select the item **State** and set the value to 0. Next, click the **Generate** button.
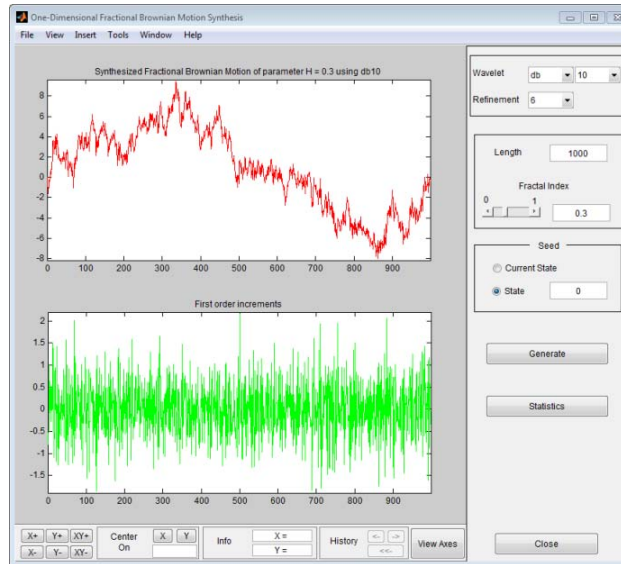
The synthesized signal exhibits a locally highly irregular behavior.

3 Now let us try another value for the fractal index. From the **Fractal Index** edit button, type 0.7 and from the **Seed** frame, select the item **State** and set the value to 0. Next, click the **Generate** button.
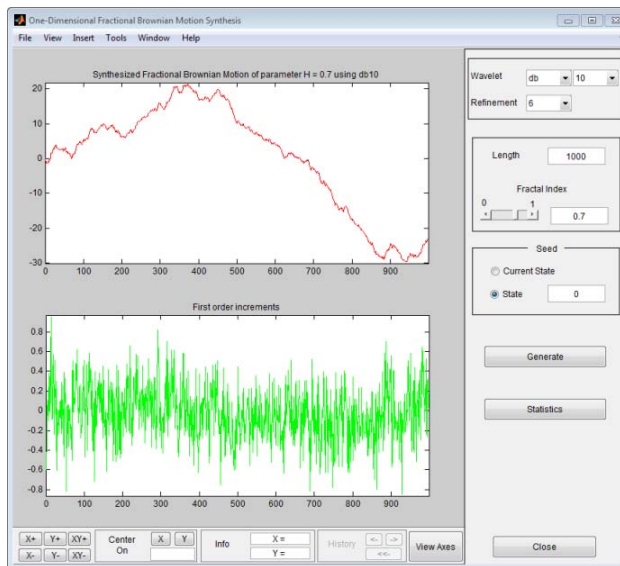
The synthesized signal clearly exhibits a stronger low-frequency component and has locally a less irregular behavior. These properties can be investigated by clicking the **Statistics** button.

Saving the Synthesized Signal

The Fractional Brownian Motion Synthesis 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 option **File > Save Synthesized Signal**. A dialog box appears that lets you specify a folder and filename for storing the signal. After saving the signal data to the file fbm07.mat, load the variables into workspace.

```
load fbm07
whos
```

Name	Size	Bytes	Class
FBM_PARAMS	1x1	668	struct array
fbm07	1x1000	8000	double array

```
FBM_PARAMS

FBM_PARAMS =
          SEED: [2x1 double]
           Wav: 'db10'
        Length: 1000
             H: 0.7000
    Refinement: 6
```

The synthesized signal is given by fbm07. In addition, the parameters of the generation are given by FBM_PARAMS, which is a cell array of length 5.

New Wavelet for CWT

This section takes you through the features of New Wavelet for CWT, one of the Wavelet Toolbox specialized tools.

The toolbox requires only one function to design a new wavelet adapted to a given pattern for CWT: `pat2cwav`. You'll find full information about this function in its reference page.

In this section, you'll learn how to

- Load a pattern
- Synthesize a new wavelet adapted to the given pattern
- Detect patterns by CWT using the adapted wavelet
- Compare the detection using both the adapted wavelet and well-known wavelets
- Save the synthesized wavelet

Since you can perform the design of the new wavelet for CWT either from the command line or using the graphical interface tools, this section has subsections covering each method.

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 (see [MisMOP03] in "References" on page 1-71).

New Wavelet for CWT Using the Command Line

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

```
% Load original pattern: a pseudo sine one.
load ptpssin1;

% Variables X and Y contain the pattern.
whos
```

Name	Size	Bytes	Class
IntVAL	1x1	8	double array
X	1x256	2048	double array
Y	1x256	2048	double array
caption	1x35	70	char array

```
IntVAL

IntVAL =
    0.1592
```

```
% The pattern on the interval [0,1] integrates to 0.1592.
% So it is not a wavelet but it is a good candidate since it
% oscillates like a wavelet.
plot(X,Y), title('Original Pattern')
```


Original Pattern
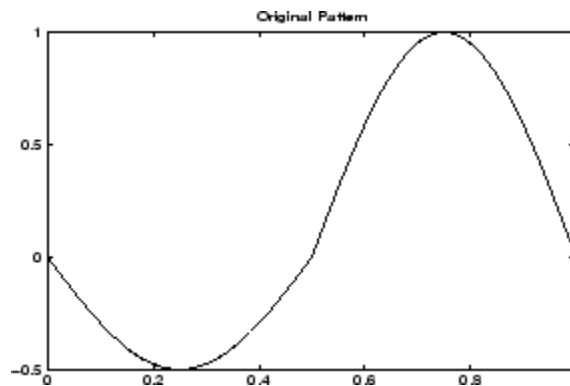
```
% To synthesize a new wavelet adapted to the given pattern, use
% a least squares polynomial approximation of degree 6 with
% constraints of continuity at the beginning and the end of the
% pattern.
[psi,xval,nc] = pat2cwav(Y, 'polynomial',6, 'continuous') ;

% The new wavelet is given by xval and nc*psi.
plot(X,Y,'-',xval,nc*psi,'--'),
title('Original Pattern and Adapted Wavelet (dashed line)')
```

Original Pattern and Adapted Wavelet (dashed line)
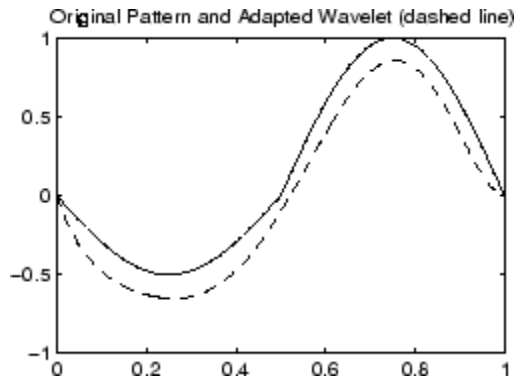

```
% Let us notice that the version of the wavelet correctly
% defined in order to be used in the CWT algorithm must be of
% square norm equal to 1. It is simply given by xval and psi.
```

New Wavelet for CWT Using the Graphical Interface

1 Start the New Wavelet for CWT Tool.

From the MATLAB prompt, type

wavemenu

The **Wavelet Toolbox Main Menu** appears. Click the **New Wavelet for CWT** menu item to display the Pattern Adapted Admissible Wavelet Design Tool.

.
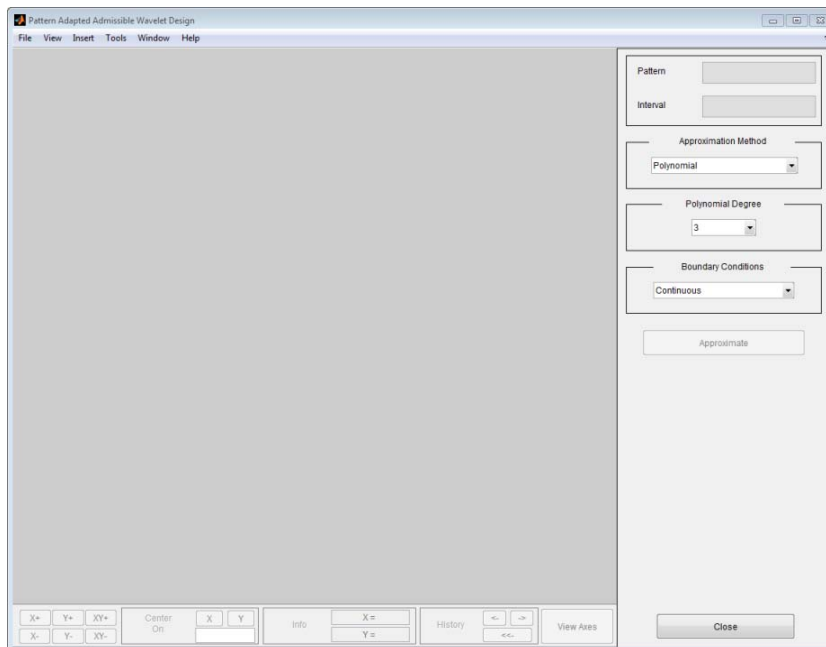
2 Load the original pattern.

The MAT-file defining the pattern can contain more than one variable. In that case, the variable Y is considered if it exists; otherwise, the first variable is considered.

3 From the **File** menu, choose the **Load Pattern** option.

When the **Load Pattern** dialog box appears, select the MAT-file ptpssin1.mat, which should reside in the MATLAB folder toolbox/wavelet/wavedemo. Click the **OK** button.
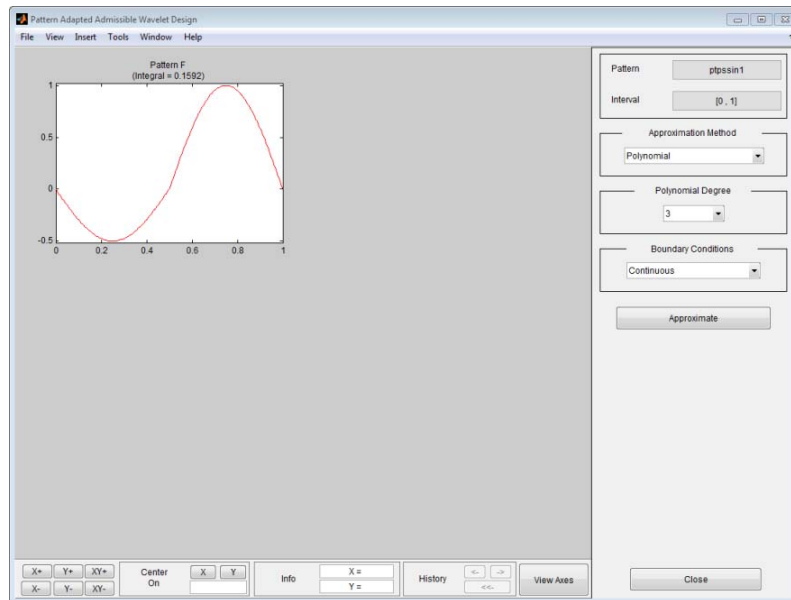
The selected pattern denoted by F is defined on the interval [0,1] and is of integral 0.1592. It is not a wavelet, but it is a good candidate because it oscillates like a wavelet.

4 Perform pattern approximation.

Accept the default parameters leading to use a polynomial of degree 3 with constraints of continuity at the borders 0 and 1, to approximate the pattern F. Click the **Approximate** button.

After a pause for computation, the tool displays the new wavelet in green superimposed with the original pattern in red.
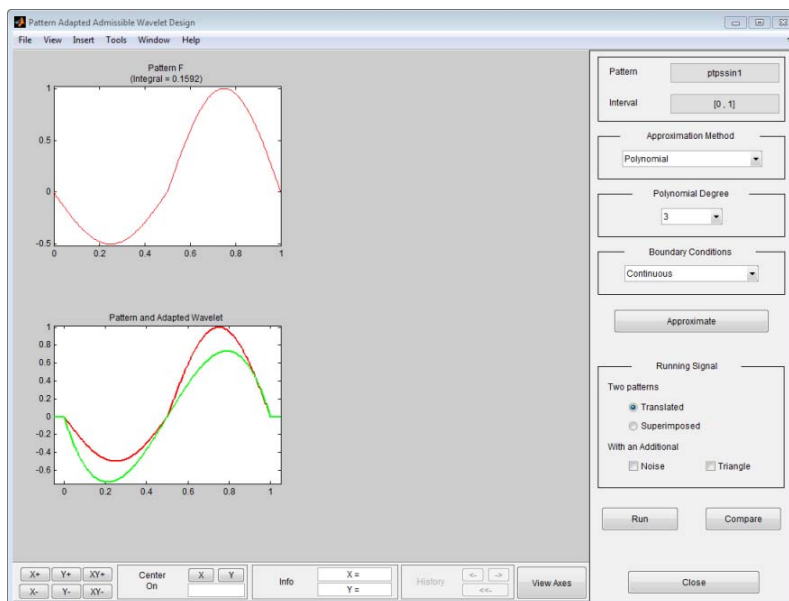
The result is not really satisfactory. A solution is to increase the polynomial degree to fit better the pattern.

5 Using the **Polynomial Degree** menu, increase the degree by selecting 6. Then click the **Approximate** button again.
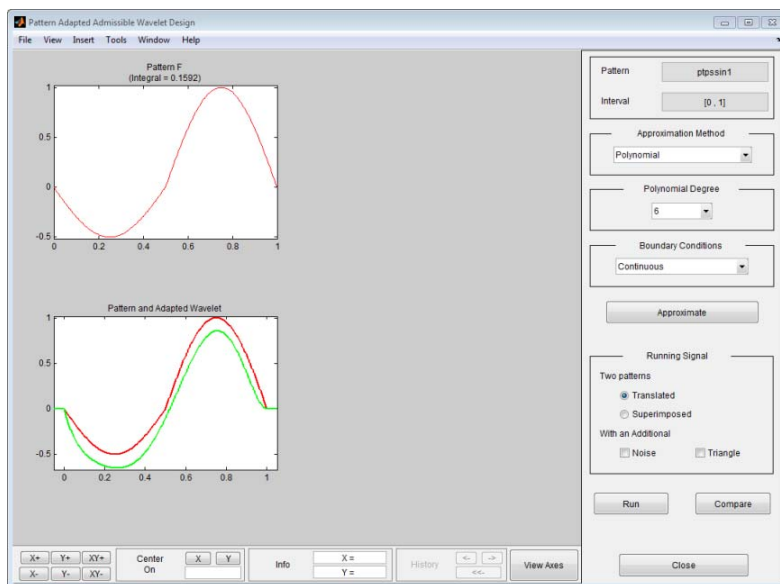
The result is now of good quality and can be used for pattern detection.

6 Pattern detection using the new wavelet.

Click the **Run** button.

After a pause for computation, the tool displays the running signal and the pattern detection by CWT using the adapted wavelet.
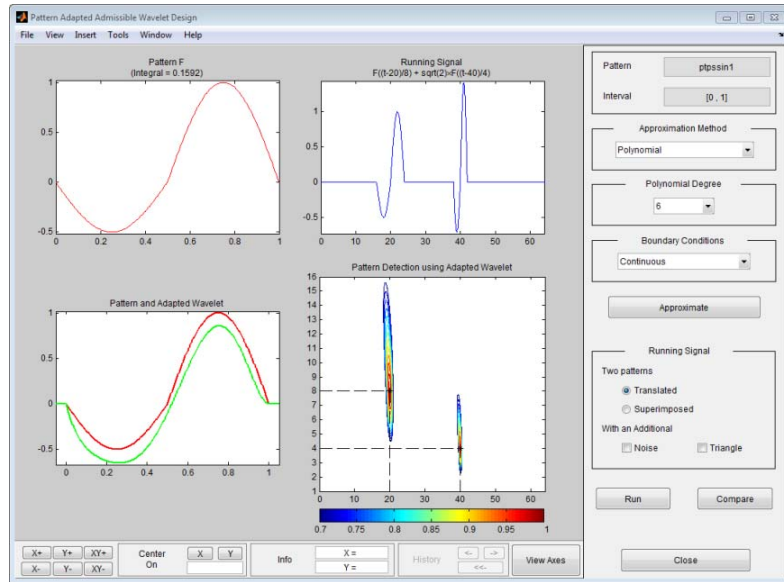
The running signal is the superimposition of two dilated and translated versions of the pattern F, namely F((t-20)/8) and F((t-40)/4). The two pairs (position, scale) to be detected are given by (20,8) and (40,4) and are materialized by dashed lines in the lower right graph of the contour plot of the CWT. The detection is perfect because the two local maxima of the absolute values of the continuous wavelet coefficients fit perfectly.

7 Using the **Running signal** frame, select the **Noise** check box to add an additive noise to the previous signal. Click the **Run** button again.
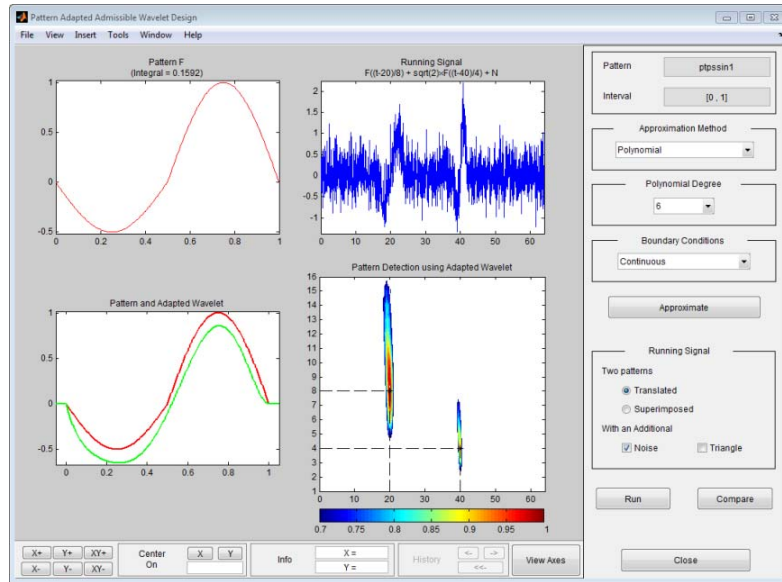
The quality of the detection is not altered at all.

8 Compare the adapted wavelet and well-known wavelets.

Let us now compare the performance for pattern detection of the adapted wavelet versus well-known wavelets. Click the **Compare** button. A new window appears.
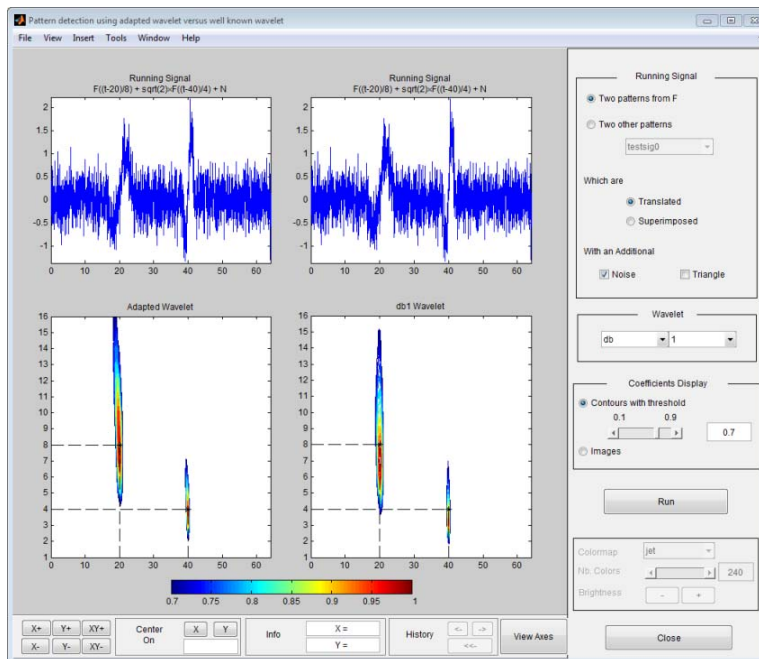
This tool displays the pattern detection performed with the adapted wavelet on the left and db1 wavelet (default) on the right. The two positions are perfectly detected in both cases but scales are slightly underestimated by the db1 wavelet.

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.

3

Getting Started with Wavelet Analysis

Wavelet Families and Properties

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')
```

Not all the wavelets listed support discrete wavelet analysis. In order to support discrete wavelet analysis, there must be decomposition and reconstruction filters that implicitly define the wavelet.

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')
```

The Daubechies extremal-phase wavelets are orthogonal wavelets that permit discrete wavelet analysis.

Determine what analysis the Morlet wavelet supports. The wavelet family short name is 'morl'.

```
waveinfo('morl')
```

The Morlet wavelet is not suitable for discrete analysis. You can do continuous wavelet analysis using this wavelet.

Use the Wavelet Toolbox interactive tool, `wavemenu`, to investigate wavelet families.

Almost anything you can do in the Wavelet Toolbox software at the command line, you can accomplish in `wavemenu`.

To start the interactive tool, enter `wavemenu` at the command line.

Click **Wavelet Display**. Select the db4 wavelet and click **Display**.
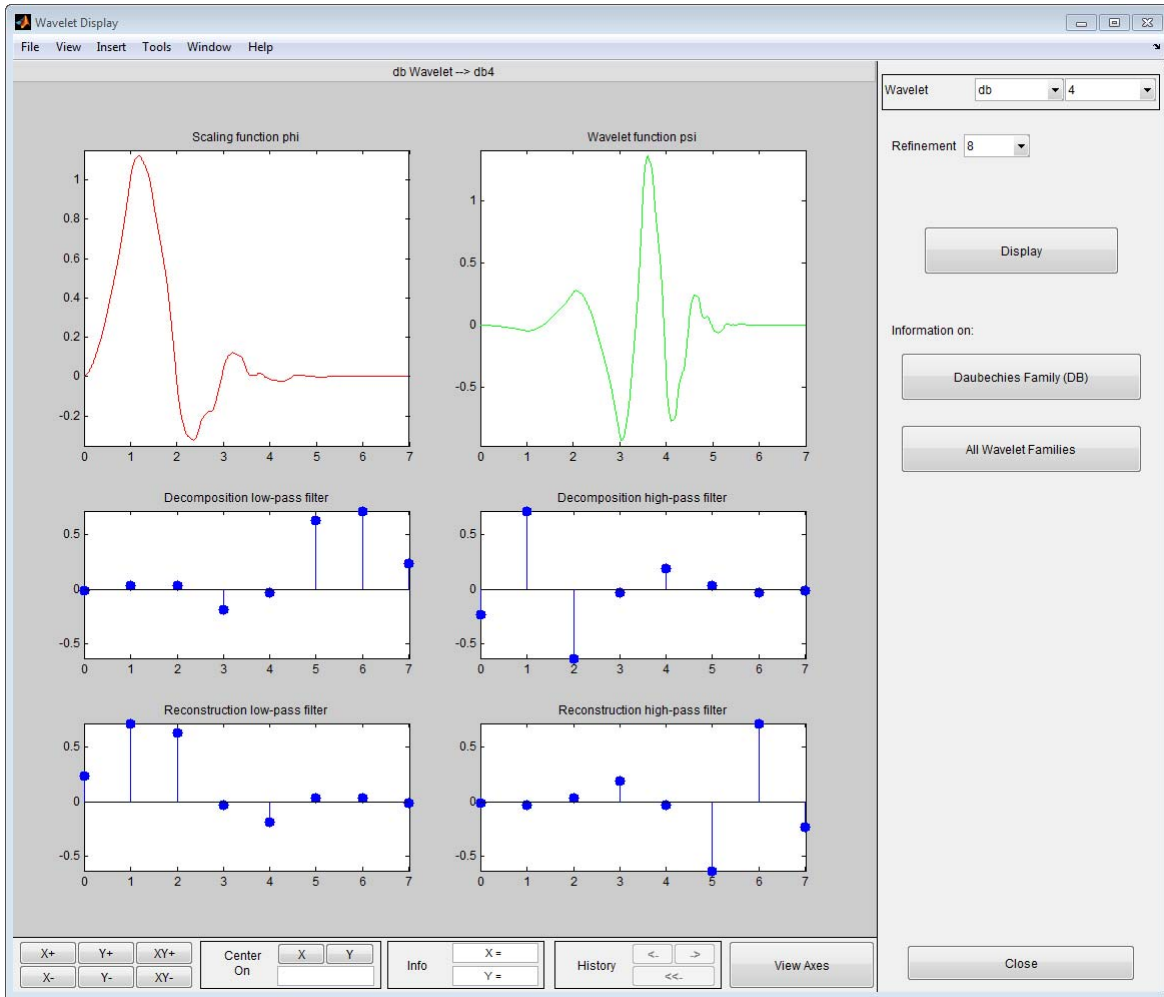
Click **Information on: Daubechies Family (DB)** to obtain specific information on this wavelet family.

Visualizing Wavelets, Wavelet Packets, and Wavelet Filters

You can use `wfilters`, `wavefun`, and `wpfun` to obtain the filters, wavelet, or wavelet packets correspond 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(221);
stem(LoD,'markerfacecolor',[O O 1]); title('Lowpass Decomposition Filter');
subplot(222);
stem(LoR,'markerfacecolor',[O O 1]); title('Lowpass Reconstruction Filter')
subplot(223);
stem(HiD,'markerfacecolor',[O O 1]); title('Highpass Decomposition Filter')
subplot(224);
stem(HiR,'markerfacecolor',[O O 1]); title('Highpass Reconstruction Filter'
```
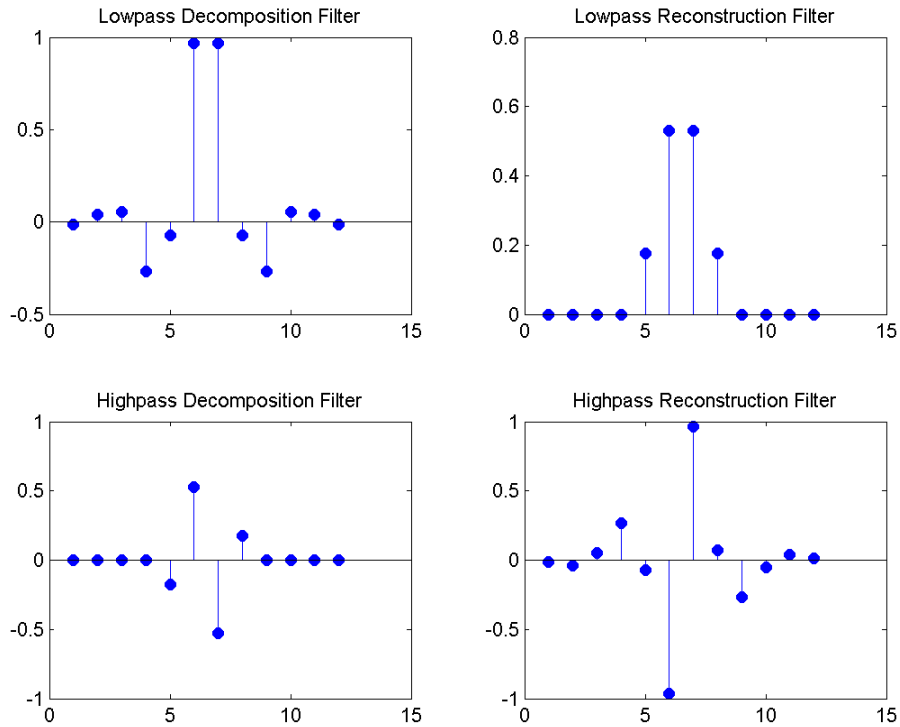
Visualize the real-valued Morlet wavelet. There is no associated scaling function.

```
[psi,xval] = wavefun('morl');
plot(xval,psi,'linewidth',2);
title('$\psi(x) = e^{-x^2/2} \cos{(5x)}$','Interpreter','latex',...
    'fontsize',14);
```

$$\psi(x) = e^{-x^2/2}\cos(5x)$$
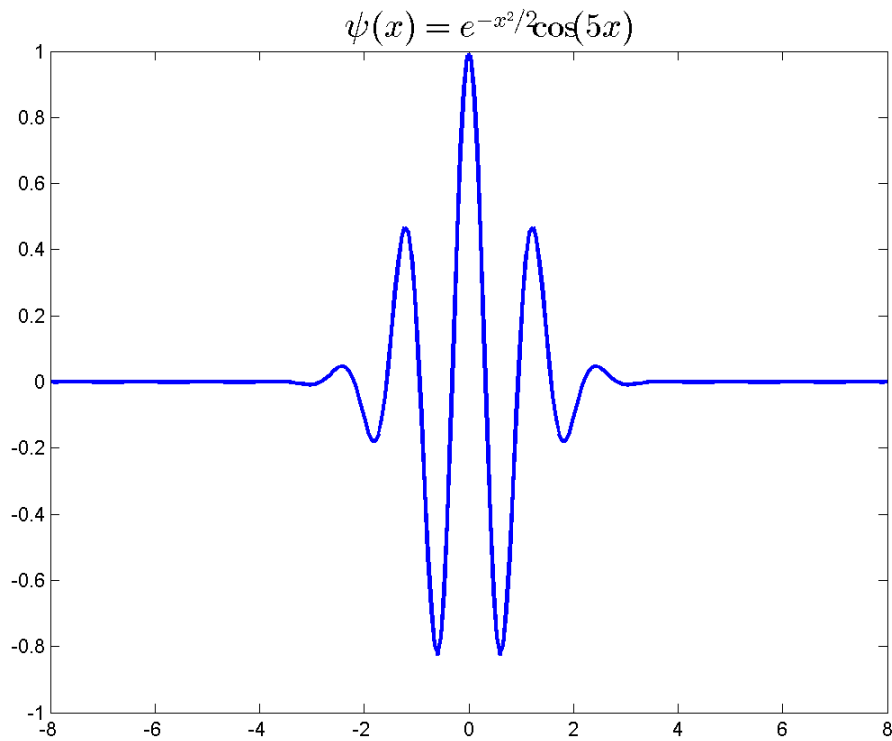
Obtain the first 4 wavelet packets for the Daubechies least-asymmetric wavelet with 4 vanishing moments, sym4.

```
[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
```
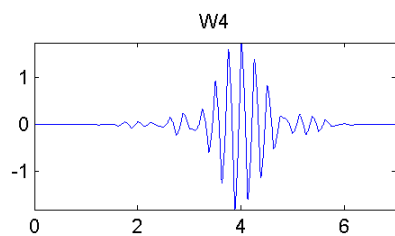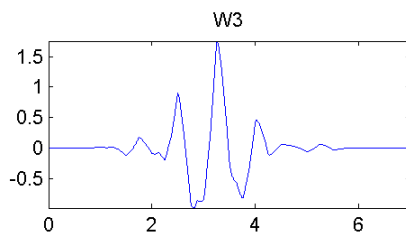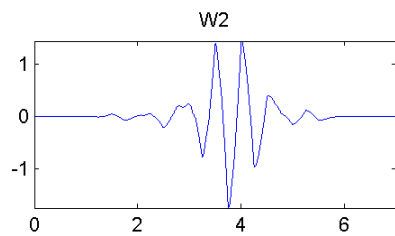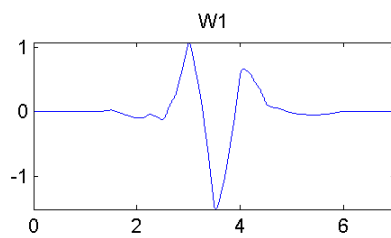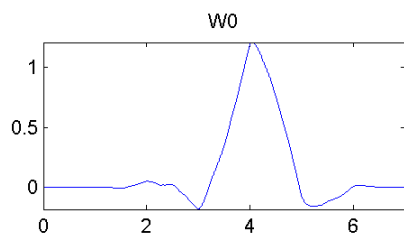
Continuous Wavelet Analysis

Continuous wavelet analysis provides a time-scale/time-frequency analysis of signals. The Wavelet Toolbox software has both command line and interactive functionality to support continuous wavelet analysis. To perform continuous wavelet analysis with the interactive tool, wavemenu, enter

wavemenu

at the command line and click one of the following choices.
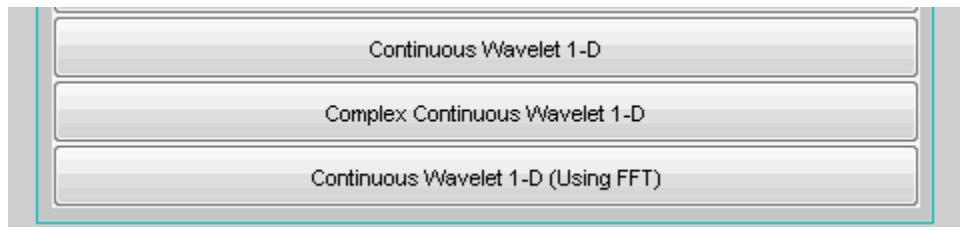
This example shows you how to perform a time-frequency analysis using the continuous wavelet transform (CWT). Construct a signal consisting of two sinusoids with frequencies of 100 and 50 Hz. The data is sampled at 1 kHz. The support of the two sinusoids is disjoint. The 100-Hz sine wave begins at t=0 and has a duration of 1 second. The 50-Hz sinusoid begins at three seconds and has a duration of two seconds.

Use the complex-valued (nonanalytic) Morlet wavelet, cmor1-1. To determine the scales of interest, assume you are interested in the frequency region from 10 to 125 Hz. To determine the range of scales corresponding to [10,125], use centfrq.

```
Fs = 1000;
fc = centfrq('cmor1-1');
% a = fc/(freq*dt)
freqrange = [20 150];
scalerange = fc./(freqrange*(1/Fs));
```

With your scales of interest, obtain a scalogram analysis.

```
t = linspace(0,5,5e3);
x = cos(2*pi*100*t).*(t<1)+cos(2*pi*50*t).*(3<t)+0.3*randn(size(t));
```

```
scales = scalerange(end):0.2:scalerange(1);
Coeffs = cwt(x,scales,'cmor1-1');
SCImg = wscalogram('image',Coeffs,'scales',scales,'ydata',x,'xdata',t);
```
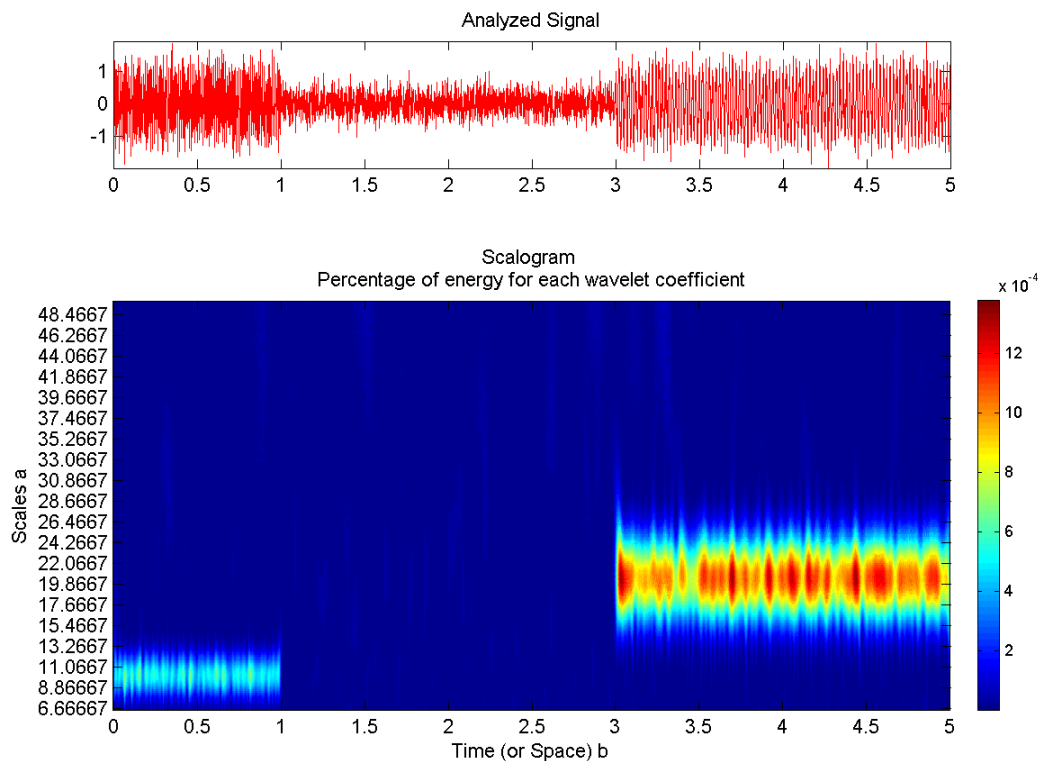

The Wavelet Toolbox software also provides a CWT implementation based on the discrete Fourier transform (DFT). See "DFT-Based Continuous Wavelet Transform" on page 1-28 for the mathematical motivation behind using the inverse DFT to compute the CWT. You can implement a DFT-based continuous wavelet analysis using the interactive tool, wavemenu, by entering

wavemenu

at the command prompt and clicking

Continuous Wavelet 1-D (Using FFT)

This example shows you how to use the DFT-based CWT. Create 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 $N(0,0.15^2)$ additive white Gaussian noise.

Create and plot the signal. The impulse at 650 milliseconds is visible, but the impulse at 750 milliseconds is not 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)+0.15*randn(size(t
plot(t,x);
```
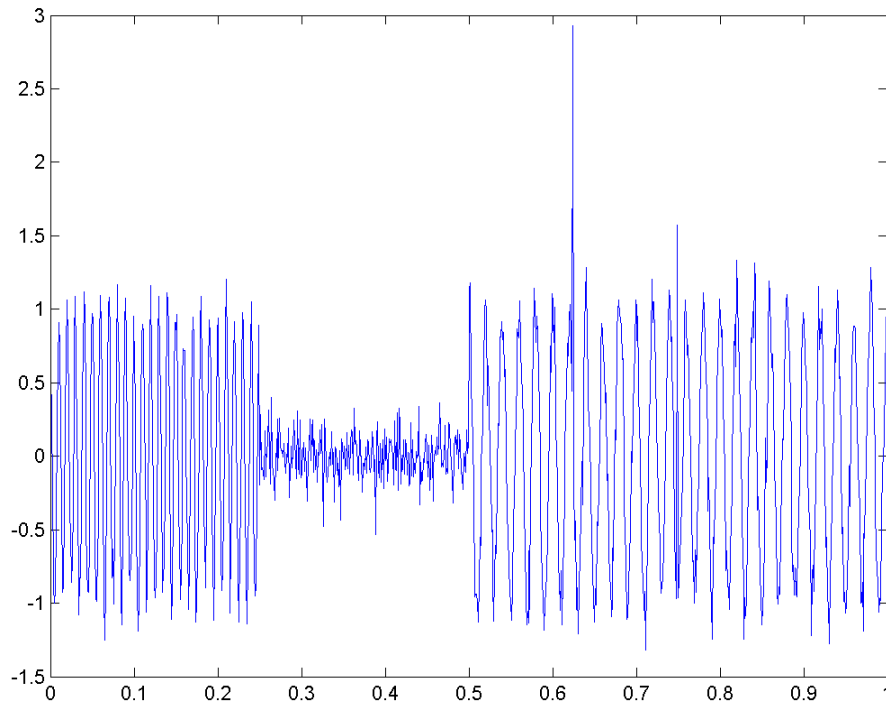
First constuct a logarithmically-spaced (base 2) scale vector that yields 47 scales from 2 milliseconds up to approximately 240 milliseconds. Set the scale resolution for the CWT at 0.15.

```
ds = 0.15;
J = fix((1/ds)*log2(length(x)/8));
dt = 1/Fs;
scales = 2*dt*2.^((0:J).*ds);
```

Obtain the CWT using an inverse DFT algorithm with cwtft. Use the analytic Morlet wavelet. Use the Fourier factor for the analytic Morlet wavelet given on the reference page for cwtft to obtain a more accurate scale-to-period conversion. Plot the result.

CWT Coefficient Moduli
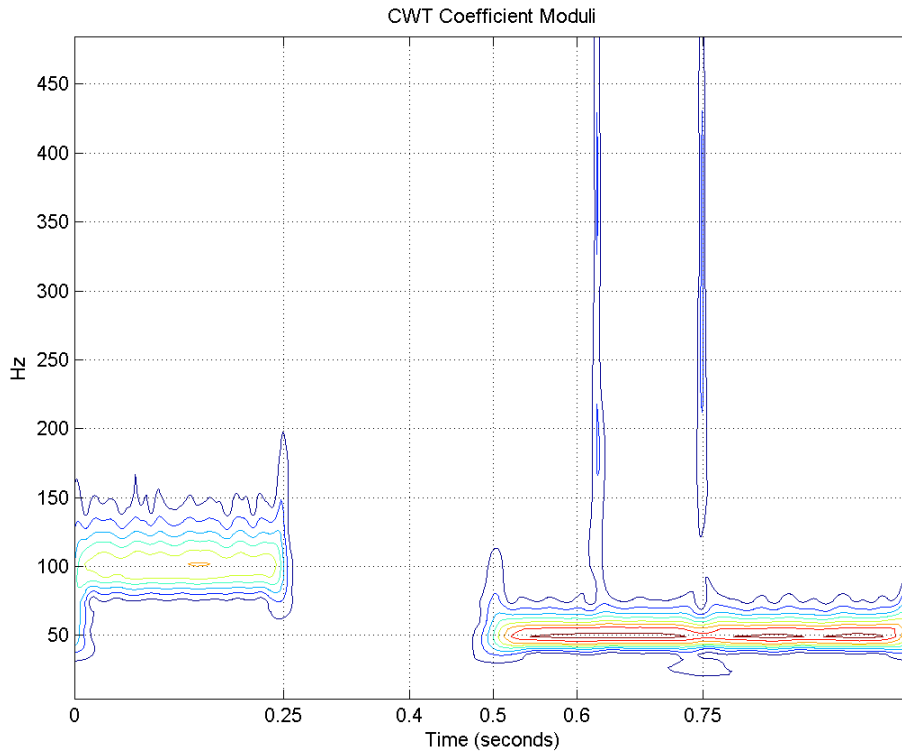
The CWT moduli correctly show the supports of the disjoint sinusoids and the locations of the impulses at 650 and 750 milliseconds. In the CWT moduli, the impulse at 750 milliseconds is clearly visible.

The DFT-based CWT enables you to approximate the inverse CWT. The approximate inverse CWT allows you to construct scale- and time-localized approximations to events in your time series. See "Inverse Continuous Wavelet Transform" on page 1-30 for a mathematical motivation.

Use the inverse CWT to obtain a scale-localized approximation to the 100-Hz sinusoid in the previous example. The signal consists 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 $N(0,0.15^2)$ additive white Gaussian noise.

Recreate the signal and continuous wavelet analysis for convenience. If you have run the previous examples and have the output in your MATLAB workspace, you do not need to repeat this step.

Create the signal.

```
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)+0.15*randn(size(t
plot(t,x);
```

Construct the scale vector for the CWT.

```
ds = 0.15;
J = fix((1/ds)*log2(length(x)/8));
dt = 1/Fs;
scales = 2*dt*2.^((0:J).*ds);
```

Obtain the CWT.

```
cwtstruct = cwtft({x,0.001},'Scales',scales,'Wavelet','morl');
periods = cwtstruct.scales.*(4*pi)/(6+sqrt(38));
freq  = 1./periods;
cfs = cwtstruct.cfs;
contour(t,freq,abs(cfs));
set(gca,'xtick',[0 0.25 0.4 0.5 0.6 0.75 1]); grid on;
xlabel('Time (seconds)'); ylabel('Hz'); title('CWT Coefficient Moduli');
```

The 100-Hz sine wave has a scale of 0.01 seconds, but the CWT analysis is not perfectly localized at that scale. Choose scales from 0.007 to 0.014 seconds to accurately capture the 100-Hz component.

```
indices = find(scales>=0.007 & scales<=0.014);
```

Copy the structure array, `cwtstruct`, from the continuous wavelet analysis of the signal. Initialize a new CWT coefficient array of zeros and extract the relevant scales from the original coefficient array. Reconstruct the signal approximation based on those scales using `icwtft`.

```
icwtsin = cwtstruct;
icwtsin.cfs = zeros(size(cwtstruct.cfs));
icwtsin.cfs(indices,:) = cwtstruct.cfs(indices,:);
xrec = icwtft(icwtsin);
plot(t,x);
hold on;
plot(t,xrec,'r');
set(gca,'xlim',[O O.4]);
legend('Original Signal','Inverse CWT Approximation',...
    'Location','NorthEast');
```
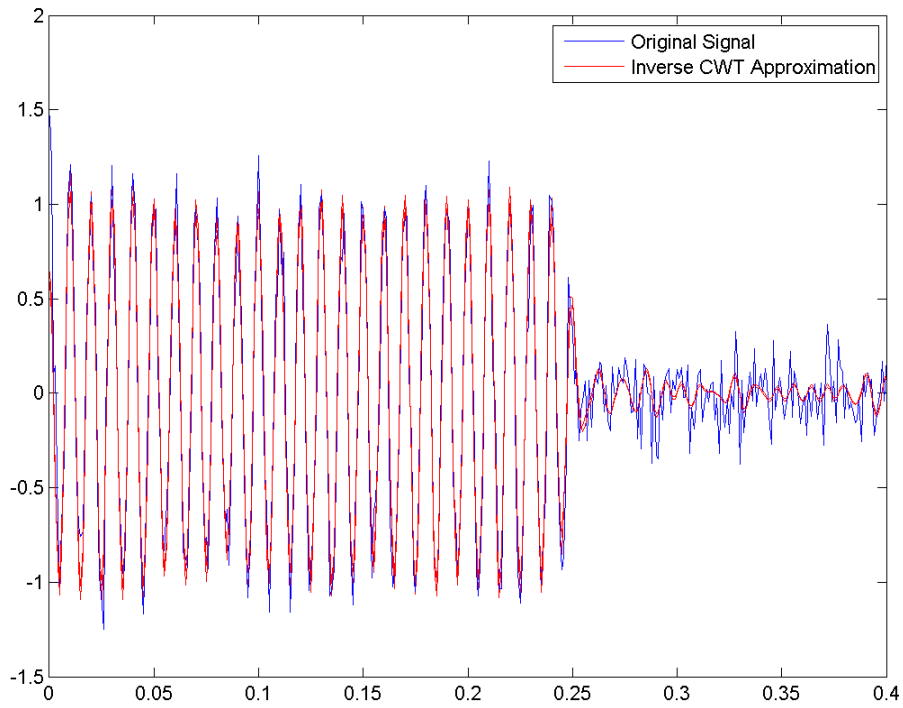
Critically-Sampled Discrete Wavelet Analysis

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, wavemenu, by entering

```
wavemenu
```

at the command line and clicking **Wavelet 1-D**, **Wavelet 2-D**, or **Wavelet 3-D**.

1-D Wavelet Denoising

This example shows you 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
```
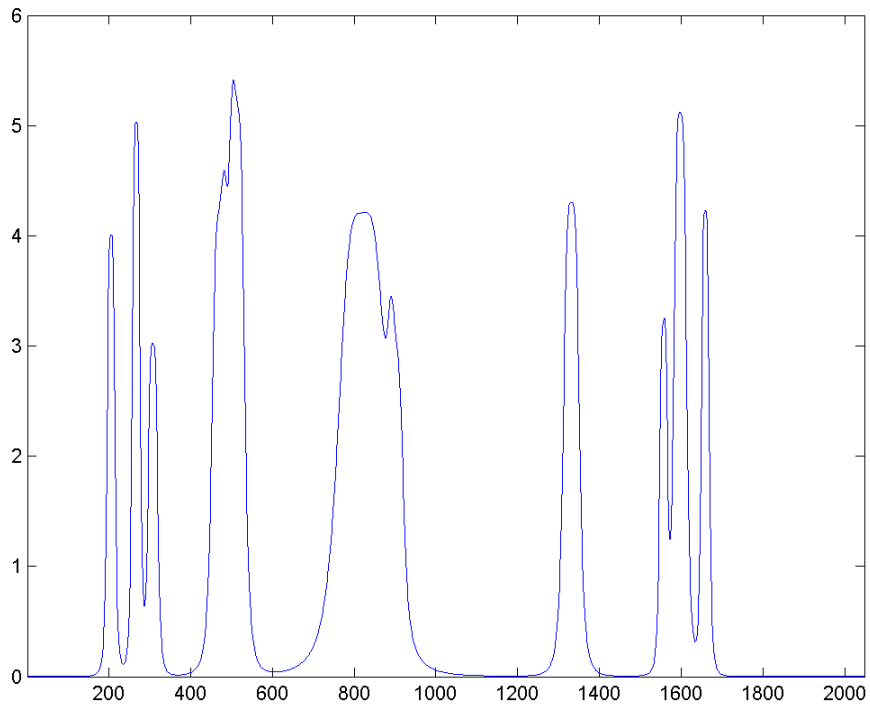
Add zero-mean white Gaussian noise with a variance of 0.25.

```
rng default;
x = xref + 0.5*randn(size(xref));
plot(x); set(gca,'xlim',[1 2048]);
```
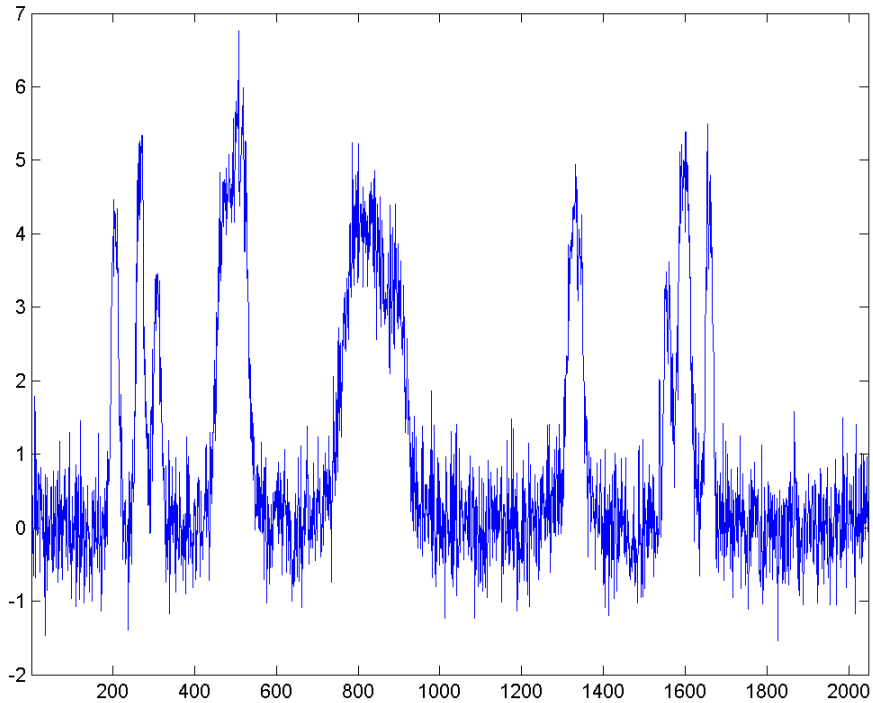
Denoise the signal down to level 3 using the Daubechies least asymmetric wavelet with 4 vanishing moments. Use the universal threshold selection rule of Donoho and Johnstone with soft thresholding based on the DWT coefficients at level 1. Use the periodization signal extension mode — dwtmode('per'). Plot the result along with the reference signal for comparision.

```
dwtmode('per');
[xd,cxd,lxd] = wden(x,'sqtwolog','s','sln',4,'sym4');
plot(xd);
set(gca,'xlim',[1 2048]); hold on;
plot(xref,'r');
```
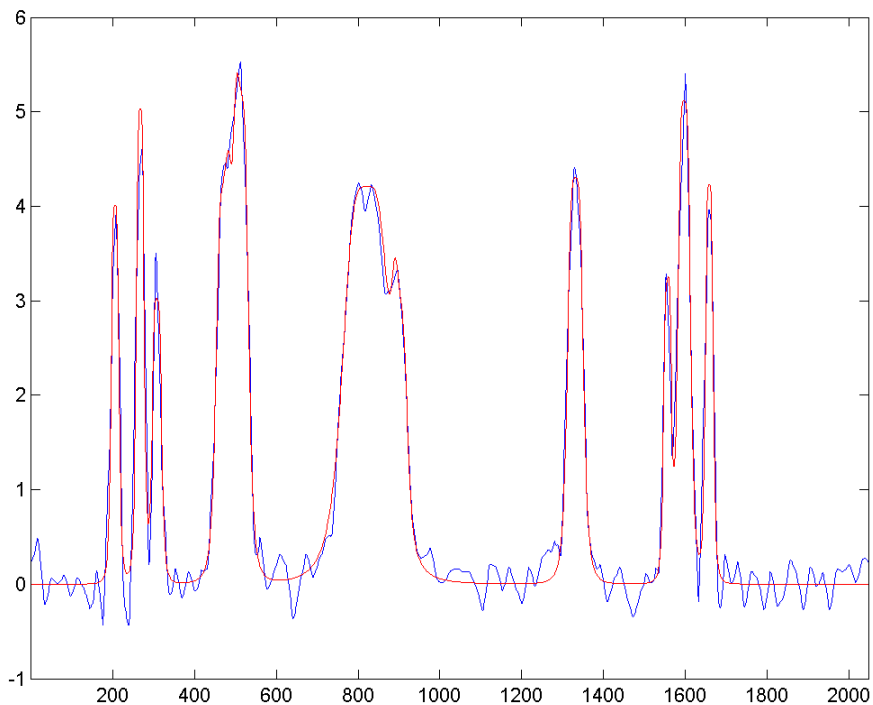
2-D Decimated Discrete Wavelet Analysis

You can analyze images with the DWT by creating 2-D separable scaling and wavelet filters. The Wavelet Toolbox has both command line and interactive functionality for 2-D decimated discrete wavelet analysis.

This example shows you 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);
```
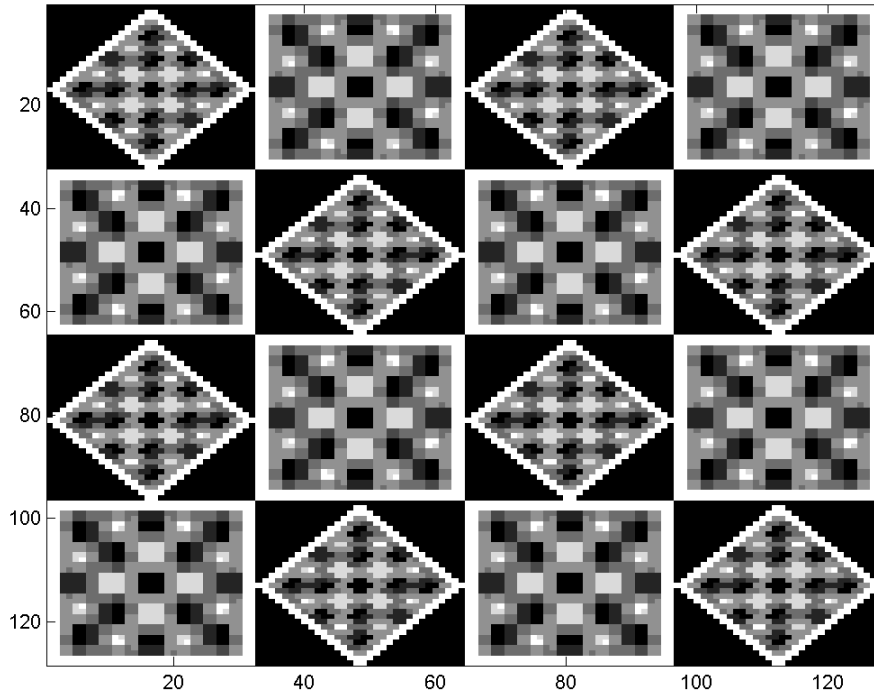
Obtain the 2-D DWT at level 1 using the biorthogonal B-spline wavelet and scaling filters with 2 vanishing moments in the analysis filters and 4 vanishing moments in the synthesis filters. Extract the horizontal, vertical, and diagonal wavelet coefficients and the approximation coefficients. Display 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');
```
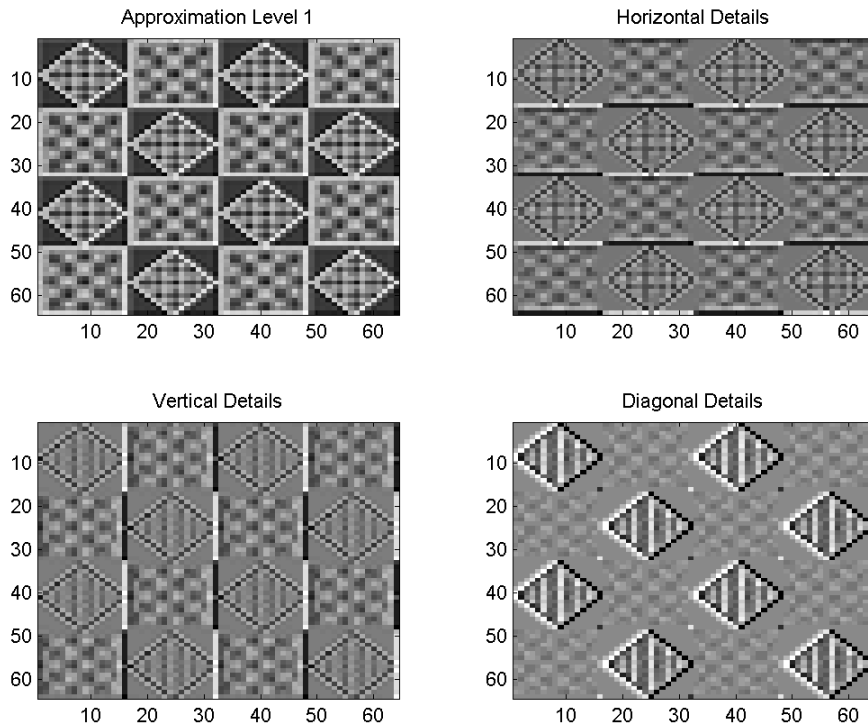

You see that the wavelet details are sensitive to particular orientations in the input image. The approximation coefficients are a lowpass approximation to the original image.

Lifting

The lifting scheme is both an alternative description of the discrete wavelet transform and an alternative way to build wavelets.

Lifting provides several advantages including:

- In-place calculation of the wavelet coefficients
- Inverse wavelet transform is easily obtained
- Ability to perform integer-to-integer wavelet transforms
- Extension to domains, which are not shift-invariant
- Extension to irregularly-sampled data

See "Lifting Method for Constructing Wavelets" for more detail on the lifting scheme.

This example shows you the principles of 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);
set(gca,'xlim',[0 100]);
```
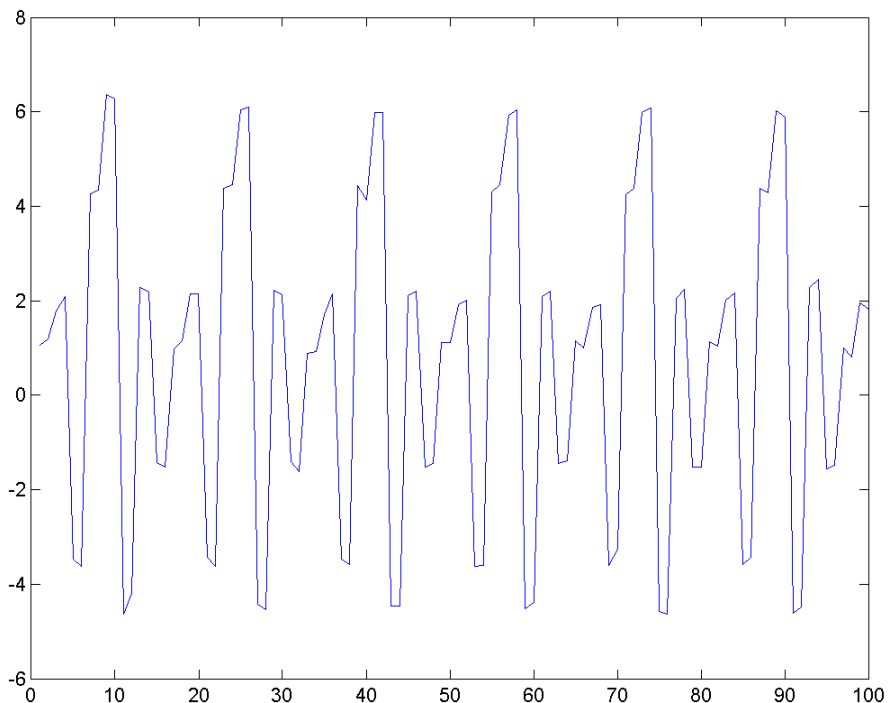
Use the *lazy* wavelet to obtain the even and odd polyphase components of the signal.

```
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))
```

The preceding example designed a wavelet, which effectively removed a zero-th 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.

This example shows you how to 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); set(gca,'xlim',[1 22]);
```
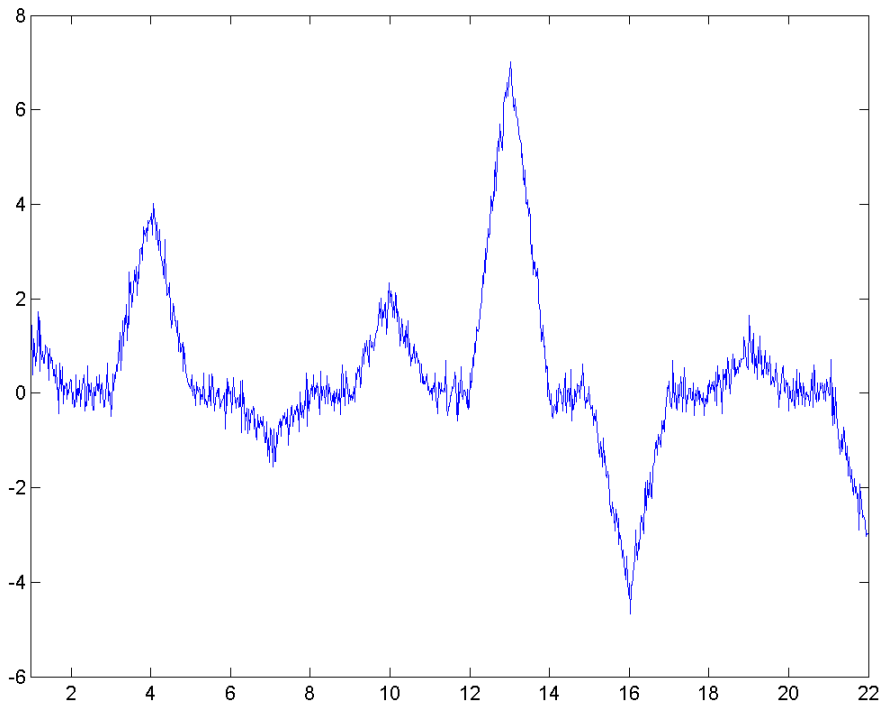
In this case, the wavelet coefficients should remove a first-order polynomial. If the signal value at an odd index, *x(2n+1)*, is well approximated by a first-order polynomial fitted to the surrounding sample values, then 1/2(*x(2n)+x(2n+2)*) should provide a good fit for *x(2n+1)*. In other words, *x(2n+1)* 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(211)
plot(A); set(gca,'xlim',[1 512]);
title('Approximation Coefficients');
subplot(212)
plot(D); set(gca,'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(211)
```

```
plot(A); set(gca,'xlim',[1 512]);
title('Approximation Coefficients');
subplot(212)
plot(D); set(gca,'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))
```

Nondecimated Discrete Wavelet Analysis

The critically-sampled discrete wavelet transform is not shift-invariant. This means that a shift in the input signal does not produce a simple shift in the DWT coefficients. The lack of shift invariance is a result of the downsampling at each level of the DWT.

The nondecimated discrete wavelet transform does not downsample at each level. The shift-invariance of the nondecimated discrete wavelet transform comes at the price of increased computation and redundancy.

The Wavelet Toolbox provides functions for 1-D and 2-D nondecimated discrete wavelet analysis. The functions swt and ndwt implement the 1-D nondecimated transform, and swt2 and ndwt2 are the 2-D versions.

You can perform nondecimated discrete wavelet analysis using the interactive tool, wavemenu, by entering

```
wavemenu
```

at the command line and clicking **SWT Denoising 1-D** or **SWT Denoising 2-D**.

This example shows you 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(411)
plot(noisdopp);
subplot(412);
plot(swc(1,:)); ylabel('D1');
set(gca,'ytick',[]);
subplot(413)
```

```
plot(swc(3,:)); ylabel('D3');
set(gca,'ytick',[]);
subplot(414);
plot(swc(5,:)); ylabel('A4');
set(gca,'ytick',[]);
```
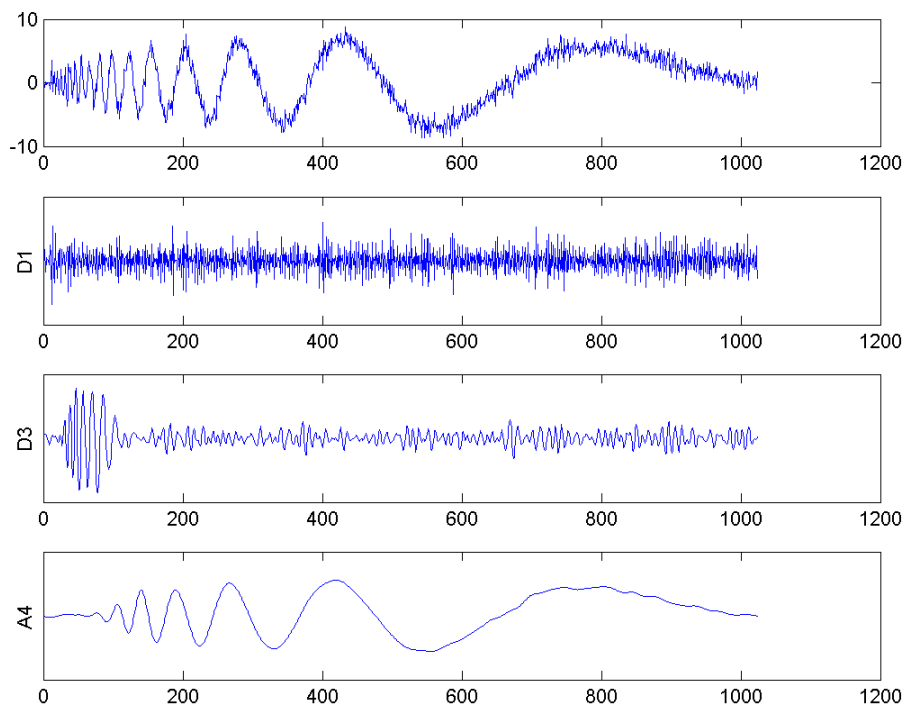

The wavelet and approximation coefficients at each level are equal in length to the input signal. The additive noise is almost entirely localized in the level one detail coefficients. The level 3 details coefficients captures the high-frequency oscillations at the beginning of the Doppler signal. The level 4 approximation coefficients are a lowpass approximation to the Doppler signal.

This example shows you how to 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.

```
subplot(221)
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 dec.: approx. ', ...
   'and det. coefs (lev. ',num2str(k),')']);
    colormap(gray)
end
```

Original image

SWT dec.: approx. and det. coefs (lev. 1)

SWT dec.: approx. and det. coefs (lev. 2)

SWT dec.: approx. and det. coefs (lev. 3)
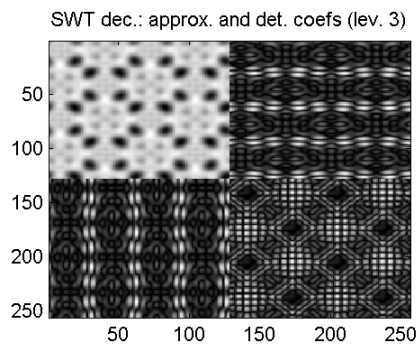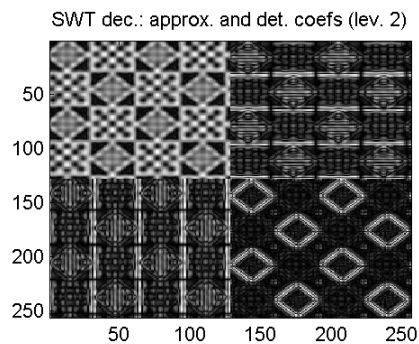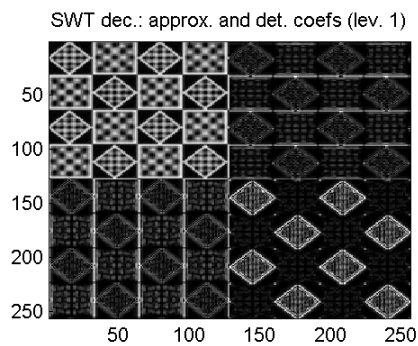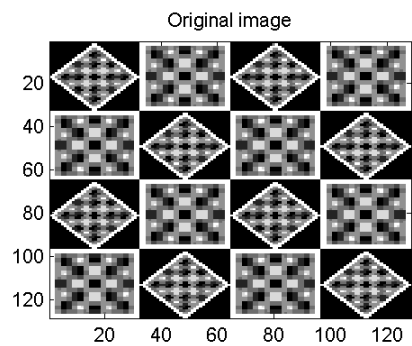
Critically-Sampled Wavelet Packet Analysis

The decimated wavelet transform filters an input, X, with a lowpass filter, H, and highpass filter, G. The output of both filtering operations is downsampled by two. The downsampled output of the lowpass filter is the input to the next level in the wavelet transform. The discrete wavelet transform does not iterate on the output of the highpass filter. The following figure illustrates the wavelet tree.
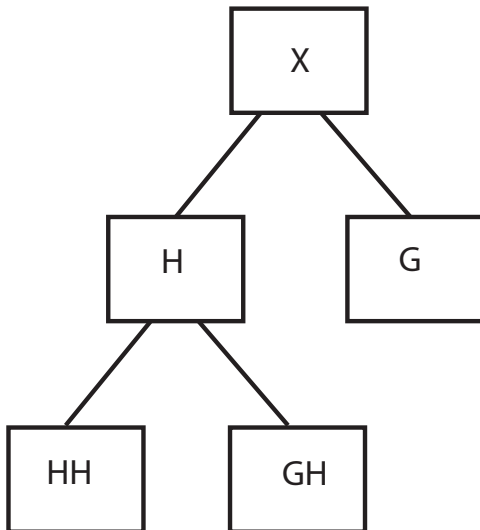
The wavelet packet transform can also operate on the output on the highpass filter. The following figure illustrates one *admissible* binary wavelet packet tree.
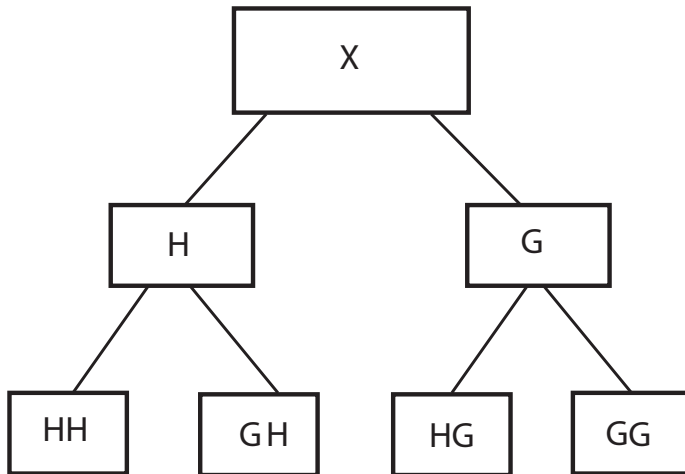
Nodes at each level of the wavelet packet tree can be *parent* nodes for *children* nodes at lower levels. An admissible wavelet packet tree has either 0 or 2 children nodes for each parent node.

There are actually a number of wavelet packet transforms.

The wavelet packet tree shown in the preceding figure goes down to level 2. At level 2, the wavelet packet transform divides the frequency support of the signal into approximately four equal intervals: {[0,п/4), [п/4, п/2), [п/2, 3п/4), [3п/4, п)}.

The wavelet packet tree in the preceding figure shows the terminal nodes in *Paley*, or natural order. The actual frequency, or *sequency*, ordering of the four terminal nodes is: {[0,п/4), [п/4, п/2), [3п/4, п), [п/2, 3п/4),)}.

You can use otnodes to determine the frequency ordering of terminal nodes.

The wavelet tree at level 2 has two nodes, which divide the frequency axis into equal intervals. The wavelet packet tree has four, which allows a finer frequency resolution with the wavelet packet transform.

You can use wpdec to obtain the wavelet packet transform of a 1-D signal and wpdec2 to obtain the wavelet packet transform of an image.

You can also perform wavelet packet analysis using the interactive tool
`wavemenu`. Enter

```
wavemenu
```

at the command line and click **Wavelet Packet 1-D** or **Wavelet Packet 2-D**.

This example shows you 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.

```
rng default;
dwtmode('per');
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, [$3\pi/4$, π), 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.
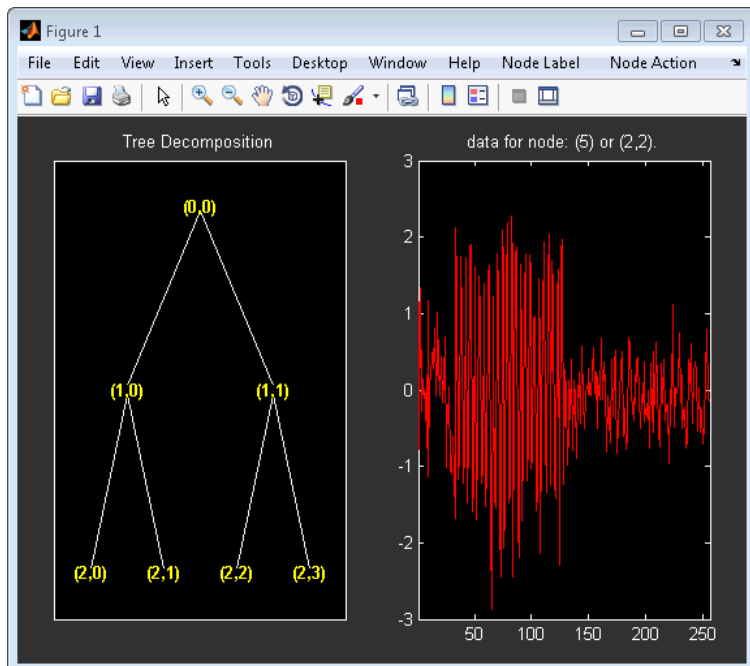
The wavelet packet transform of a 2-D image yields a quarternary wavelet packet tree.

This example shows you how to obtain the wavelet packet transform of an image.

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 quartenary wavelet packet tree.

```
load tartan;
T = wpdec2(X,2,'bior3.5');
plot(T);
```

Matching Pursuit

Matching pursuit allows you to represent a 1-D signal in an overcomplete dictionary. See "Sparse Representation in Redundant Dictionaries" for a more detailed motivation for signal representations in an overcomplete dictionary.

Wavelet Toolbox software provides both command line and interactive tools for matching pursuit. You can construct a time-frequency, or time-scale dictionary consisting of wavelet, wavelet packet, and non-wavelet bases. The non-wavelet options include cosine, sine, discrete cosine transform (DCT-II), shifted impulses, and polynomial bases. You can also include your own custom subdictionaries. See "Matching Pursuit — Command Line" and "Matching Pursuit — Interactive Analysis" for examples.

This example shows you how to perform matching pursuit on a 1-D input signal.

Load the cuspamax signal. Construct a dictionary consisting of Daubechies least asymmetric wavelet packets at level 4, Daubechies extremal phase wavelets at level 2, the DCT-II basis, the sin basis, and the shifted Kronecker delta basis.

```
load cuspamax;
lstcpt = {{'wpsym4',1},{'db4',2},'dct','sin','RnIdent'};
mpdict = wmpdictionary(length(cuspamax),'LstCpt',lstcpt);
```

Use orthogonal matching pursuit to obtain an approximation of the signal in the overcomplete dictionary, mpdict, with 25 iterations. Plot the result as a movie, updating every 5 iterations.

```
[yfit,r,coeff,iopt,qual] = wmpalg('OMP',cuspamax,mpdict,...
    'typeplot','movie','stepplot',5);
```

Index