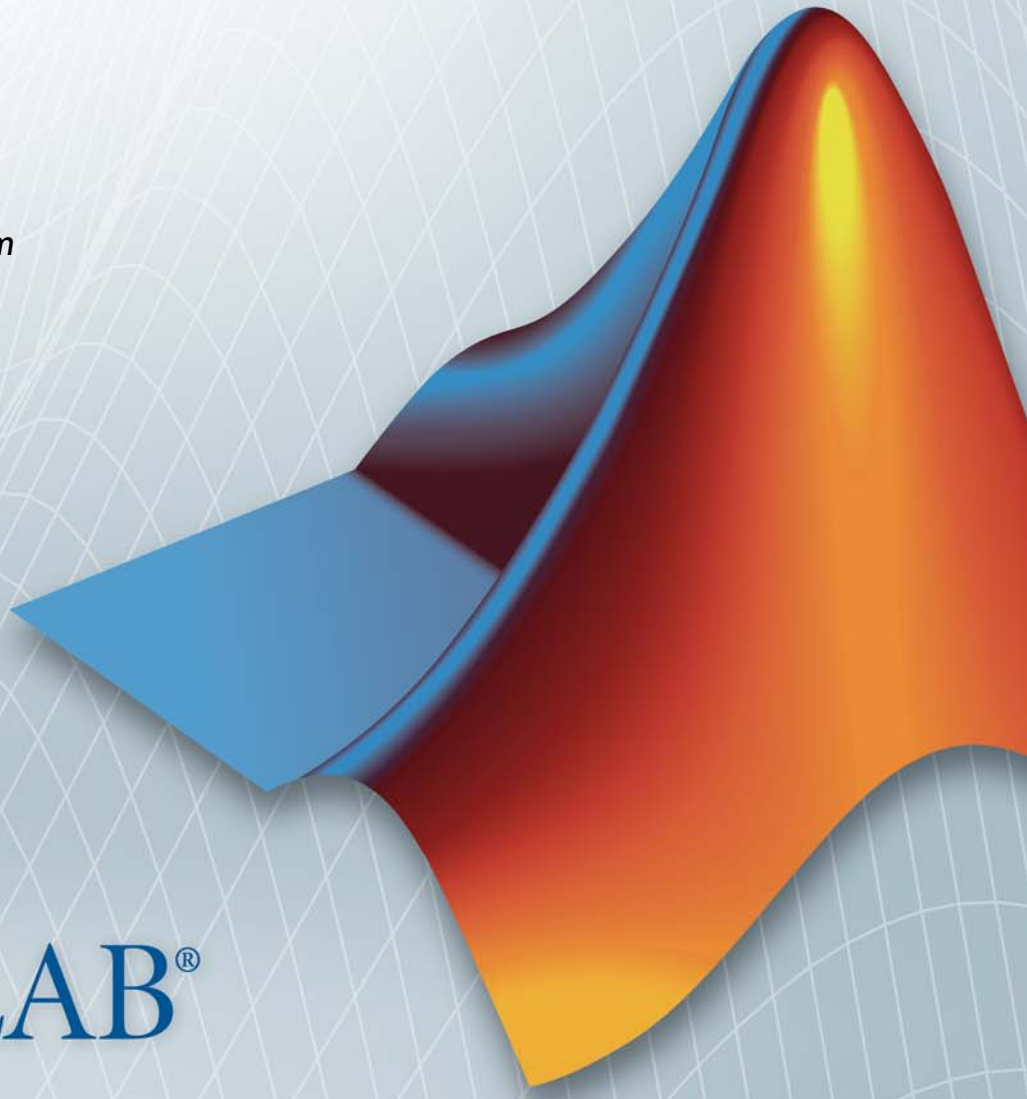
Wavelet Toolbox™

Getting Started Guide

R2011b

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

Revision History

March 1997	First printing	New for Version 1.0
September 2000	Second printing	Revised for Version 2.0 (Release 12)
June 2001	Online only	Revised for Version 2.1 (Release 12.1)
July 2002	Online only	Revised for Version 2.2 (Release 13)
June 2004	Online only	Revised for Version 3.0 (Release 14)
July 2004	Third printing	Revised for Version 3.0
October 2004	Online only	Revised for Version 3.0.1 (Release 14SP1)
March 2005	Online only	Revised for Version 3.0.2 (Release 14SP2)
June 2005	Fourth printing	Minor revision for Version 3.0.2
September 2005	Online only	Minor revision for Version 3.0.3 (Release R14SP3)
March 2006	Online only	Minor revision for Version 3.0.4 (Release 2006a)
September 2006	Online only	Revised for Version 3.1 (Release 2006b)
March 2007	Online only	Revised for Version 4.0 (Release 2007a)
September 2007	Online only	Revised for Version 4.1 (Release 2007b)
October 2007	Fifth printing	Revised for Version 4.1
March 2008	Online only	Revised for Version 4.2 (Release 2008a)
October 2008	Online only	Revised for Version 4.3 (Release 2008b)
March 2009	Online only	Revised for Version 4.4 (Release 2009a)
September 2009	Online only	Minor revision for Version 4.4.1 (Release 2009b)
March 2010	Online only	Revised for Version 4.5 (Release 2010a)
September 2010	Online only	Revised for Version 4.6 (Release 2010b)
April 2011	Online only	Revised for Version 4.7 (Release 2011a)
September 2011	Online only	Revised for Version 4.8 (Release 2011b)

Contents

1 Wavelets: A New Tool for Signal Analysis

2 Using Wavelets

Index

1

Wavelets: A New Tool for Signal Analysis

Product Overview

Everywhere around us are signals that can be analyzed. For example, there are seismic tremors, human speech, engine vibrations, medical images, financial data, music, and many other types of signals. Wavelet analysis is a new and promising set of tools and techniques for analyzing these signals.

The Wavelet Toolbox™ software is a collection of functions built on the MATLAB® technical computing environment. It provides tools for the analysis and synthesis of deterministic and random signals and images using wavelets and wavelet packets using the MATLAB language.

MathWorks® provides several other products that complement the signal and image analysis tasks you can perform with the Wavelet Toolbox software. These products include the Signal Processing Toolbox™ and the Image Processing Toolbox™ to mention just two examples. For more information about these and other MathWorks products, see www.mathworks.com/products/.

The Wavelet Toolbox software provides two categories of tools:

- Command-line functions
- Graphical interactive tools

The command-line functions are MATLAB programs that you can call directly from the command line or from your own applications. Most of these functions are in separate files using a `.m` extension. You can view the code for these functions using the following statement:

```
edit function_name
```

You can view command-line help for each function with:

```
help function_name
```

To view detailed reference pages for each function, enter:

```
doc function_name
```

A summary list of the Wavelet Toolbox functions is available to you by typing

```
help wavelet
```

You can also view a list of Wavelet Toolbox functions by viewing the “Function Reference” and “Function Reference” files in the documentation.

You can change the way any toolbox function works by copying and renaming the MATLAB file and modifying your copy. You can also extend the toolbox by adding your own MATLAB programs.

The second category of tools is a collection of graphical interface tools that afford access to extensive functionality. Access these tools from the command line by typing

```
wavemenu
```

Note The examples in this guide are generated using Wavelet Toolbox software with the DWT extension mode set to `'zpd'` (for zero padding), except when it is explicitly mentioned. So if you want to obtain exactly the same numerical results, type `dwtmode('zpd')`, before to execute the example code.

In most of the command-line examples, figures are displayed. To clarify the presentation, the plotting commands are partially or completely omitted. To reproduce the displayed figures exactly, you would need to insert some graphical commands in the example code.

Background Reading

Wavelet Toolbox software provides a complete introduction to wavelets and assumes no previous knowledge of the area. The toolbox allows you to use wavelet techniques on your own data immediately and develop new insights.

It is our hope that, through the use of these practical tools, you may want to explore the beautiful underlying mathematics and theory.

Excellent supplementary texts provide complementary treatments of wavelet theory and practice (see "References") in the *Wavelet Toolbox User's Guide*. For instance:

- Burke-Hubbard [Bur96] is an historical and up-to-date text presenting the concepts using everyday words.
- Daubechies [Dau92] is a classic for the mathematics.
- Kaiser [Kai94] is a mathematical tutorial, and a physics-oriented book.
- Mallat [Mal98] is a 1998 book, which includes recent developments, and consequently is one of the most complete.
- Meyer [Mey93] is the "father" of the wavelet books.
- Strang-Nguyen [StrN96] is especially useful for signal processing engineers. It offers a clear and easy-to-understand introduction to two central ideas: filter banks for discrete signals, and for wavelets. It fully explains the connection between the two. Many exercises in the book are drawn from Wavelet Toolbox software.

The Wavelet Digest Internet site (http://www.wavelet.org/) provides much useful and practical information.

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.

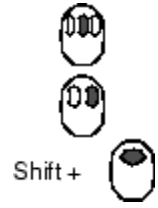
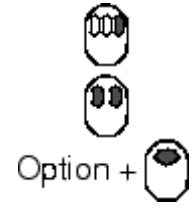
Fonts for Non-Windows Platforms

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.

For more information, see "Using the Mouse" in the *Wavelet Toolbox User's Guide*.

Wavelet Applications

Wavelets are characterized by scale and position. As a result, wavelets are useful in analyzing variations in signals and images, which are best characterized in terms of scale and position. To clarify them we try to untangle the aspects somewhat arbitrarily.

For scale aspects, we present one idea around the notion of local regularity. For time aspects, we present a list of domains. When the decomposition is taken as a whole, the de-noising and compression processes are center points.

Scale Aspects

As a complement to the spectral signal analysis, new signal forms appear. They are less regular signals than the usual ones.

The cusp signal presents a very quick local variation. Its equation is t^r with t close to 0 and $0 < r < 1$. The lower r the sharper the signal.

To illustrate this notion physically, imagine you take a piece of aluminum foil; The surface is very smooth, very regular. You first crush it into a ball, and then you spread it out so that it looks like a surface. The asperities are clearly visible. Each one represents a two-dimension cusp and analog of the one dimensional cusp. If you crush again the foil, more tightly, in a more compact ball, when you spread it out, the roughness increases and the regularity decreases.

Several domains use the wavelet techniques of regularity study:

- Biology for cell membrane recognition, to distinguish the normal from the pathological membranes
- Metallurgy for the characterization of rough surfaces
- Finance for the analysis of nonstationary time series
- In Internet traffic description, for designing the services size

Position (or Time) Aspects

Let's switch to position aspects. The main goals are:

- Rupture and edges detection
- Study of short-time phenomena as transient processes

As domain applications, we get:

- Industrial supervision of gear-wheel
- Checking undue noises in craned or dented wheels, and more generally in nondestructive control quality processes
- Detection of short pathological events as epileptic crises or normal ones as evoked potentials in EEG (medicine)
- SAR imagery
- Automatic target recognition
- Intermittence in physics

Wavelet Decomposition as a Whole

Many applications use the wavelet decomposition taken as a whole. The common goals concern the signal or image clearance and simplification, which are parts of de-noising or compression.

We find many published papers in oceanography and earth studies.

One of the most popular successes of the wavelets is the compression of FBI fingerprints.

When trying to classify the applications by domain, it is almost impossible to sum up several thousand papers written within the last 15 years. Moreover, it is difficult to get information on real-world industrial applications from companies. They understandably protect their own information.

Some domains are very productive. Medicine is one of them. We can find studies on micro-potential extraction in EKGs, on time localization of His bundle electrical heart activity, in ECG noise removal. In EEGs, a quick transitory signal is drowned in the usual one. The wavelets are able to determine if a quick signal exists, and if so, can localize it. There are attempts to enhance mammograms to discriminate tumors from calcifications.

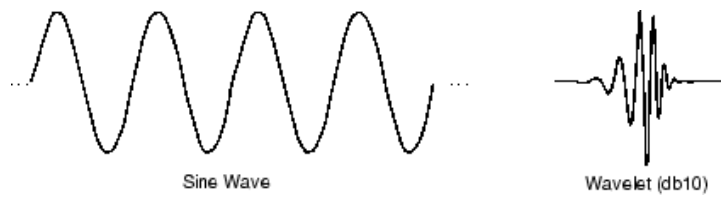
Another prototypical application is a classification of Magnetic Resonance Spectra. The study concerns the influence of the fat we eat on our body fat. The type of feeding is the basic information and the study is intended to avoid taking a sample of the body fat. Each Fourier spectrum is encoded by some of its wavelet coefficients. A few of them are enough to code the most interesting features of the spectrum. The classification is performed on the coded vectors.

What is Wavelet Analysis?

Now that we know some situations when wavelet analysis is useful, it is worthwhile asking "What is wavelet analysis?" and even more fundamentally, "What is a wavelet?"

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

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. And where sinusoids are smooth and predictable, wavelets tend to be irregular and asymmetric.



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

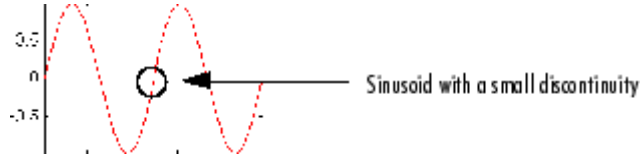
Just looking at pictures of wavelets and sine waves, you can see intuitively that signals with sharp changes might be better analyzed with an irregular wavelet than with a smooth sinusoid, just as some foods are better handled with a fork than a spoon.

It also makes sense that local features can be described better with wavelets that have local extent.

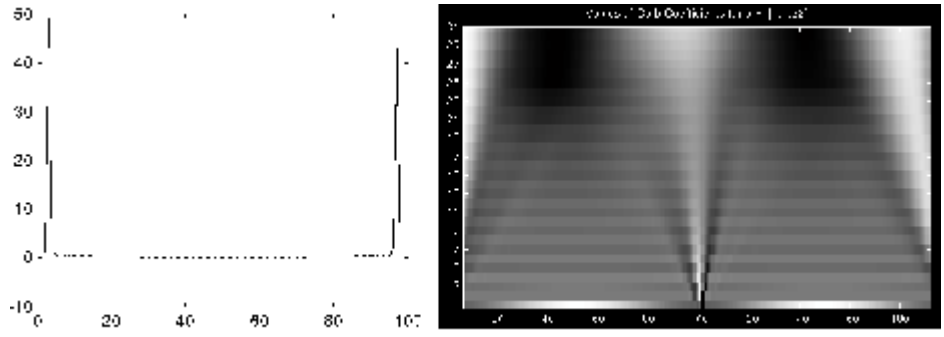
What Can Wavelet Analysis Do?

One major advantage afforded by wavelets is the ability to perform *local analysis* — that is, to analyze a localized area of a larger signal.

Consider a sinusoidal signal with a small discontinuity — one so tiny as to be barely visible. Such a signal easily could be generated in the real world, perhaps by a power fluctuation or a noisy switch.



A plot of the Fourier coefficients (as provided by the `fft` command) of this signal shows nothing particularly interesting: a flat spectrum with two peaks representing a single frequency. However, a plot of wavelet coefficients clearly shows the exact location in time of the discontinuity.



Wavelet analysis is capable of revealing aspects of data that other signal analysis techniques miss, aspects like trends, breakdown points, discontinuities in higher derivatives, and self-similarity. Furthermore, because it affords a different view of data than those presented by traditional techniques, wavelet analysis can often compress or de-noise a signal without appreciable degradation.

Indeed, in their brief history within the signal processing field, wavelets have already proven themselves to be an indispensable addition to the analyst's collection of tools and continue to enjoy a burgeoning popularity today.

From Fourier Analysis to Wavelet Analysis

In this section...

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.

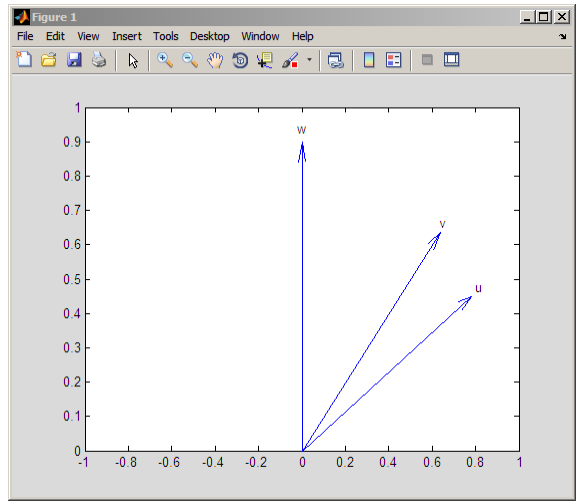
As a simple example of the inner product as a measure of similarity, consider the inner product of vectors in the plane. The following MATLAB example calculates the inner product of three unit vectors, $\{u,v,w\}$, in the plane:

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

```
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

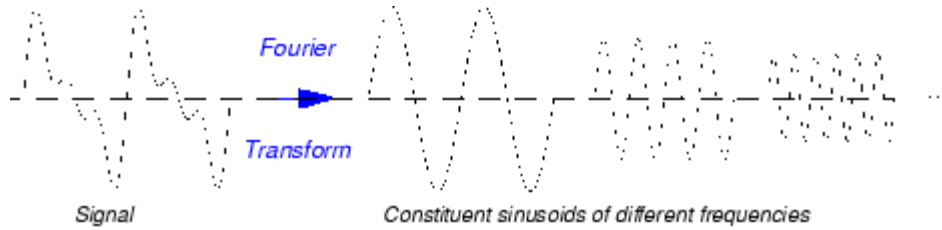
Fourier analysis is used as a starting point to introduce the wavelet transforms, and as a benchmark to demonstrate cases where wavelet analysis provides a more useful characterization of signals than Fourier analysis.

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.

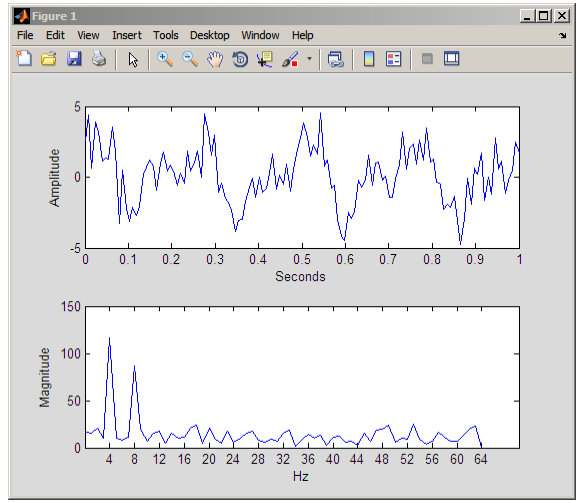
The integral defining the Fourier transform is an *inner product*. See “Inner Products” on page 1-12 for an example of how inner products measure of similarity between two signals. For each value of ω, the integral (or sum) over all values of time produces a scalar, F(ω), that summarizes how similar the two signals are. These complex-valued scalars are the *Fourier coefficients*. Conceptually, multiplying each Fourier coefficient, F(ω), by a complex exponential (sinusoid) of frequency ω yields the constituent sinusoidal components of the original signal. Graphically, the process looks like



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

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

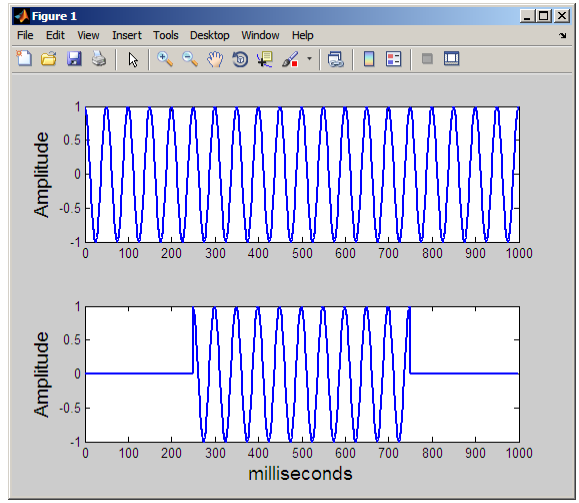
```
rng(0,'twister');
Fs = 128;
t = linspace(0,1,128);
x = 2*cos(2*pi*4*t)+1.5*sin(2*pi*8*t)+randn(size(t));
xDFT = fft(x);
Freq = 0:64;
subplot(211);
plot(t,x); xlabel('Seconds'); ylabel('Amplitude');
subplot(212);
plot(Freq,abs(xDFT(1:length(xDFT)/2+1)))
set(gca,'xtick',[4:4:64]);
xlabel('Hz'); ylabel('Magnitude');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



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

Short-Time Fourier Transform

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



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

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

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

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

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

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

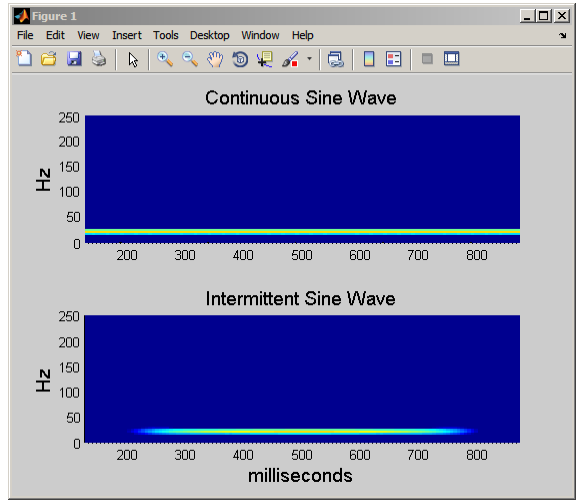
$$F(\omega,\tau) = \int f(t) w(t-\tau) e^{-j\omega t} dt$$

The STFT maps a function of one variable into a function of two variables, ω and τ. This 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

In this section...

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

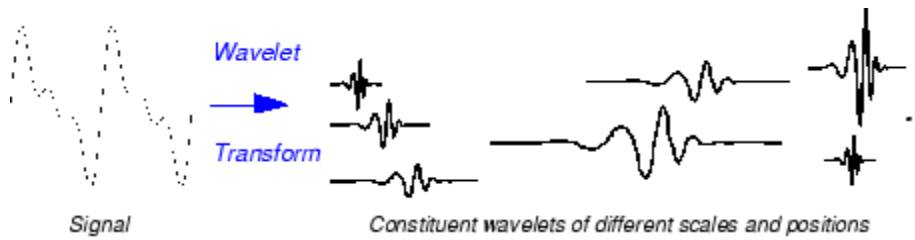
variables. The two-dimensional representation of a one-dimensional signal is redundant. If the wavelet is complex-valued, the CWT is a complex-valued function of scale and position. If the signal is real-valued, the CWT is a real-valued function of scale and position. For a scale parameter, a>0, and position, b, the CWT is:

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

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.

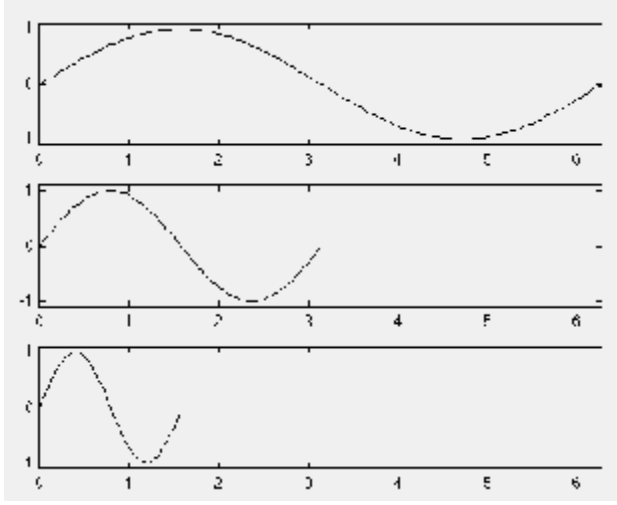


There are many different admissible wavelets that can be used in the CWT. While it may seem confusing that there are so many choices for the analyzing wavelet, it is actually a strength of wavelet analysis. Depending on what signal features you are trying to detect, you are free to select a wavelet that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

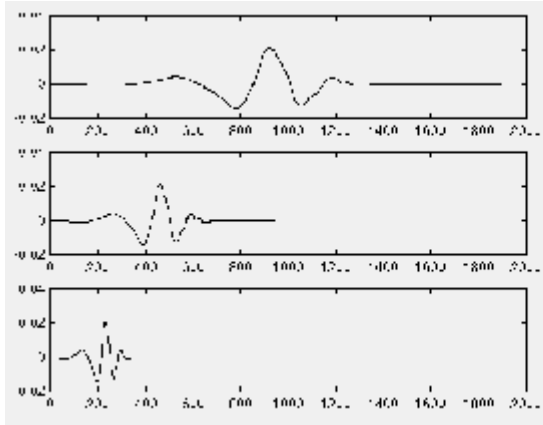
Scale

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.



The scale factor works exactly the same with wavelets. The smaller the scale factor, the more “compressed” the wavelet.



$$f(t) = \psi(t) \quad ; \quad a = 1$$

$$f(t) = \psi(2t) \; ; \quad a = \frac{1}{2}$$

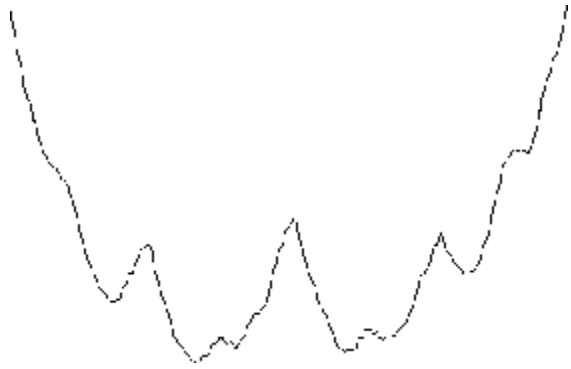
$$f(t) = \psi(4t) \; ; \quad a = \frac{1}{4}$$

It is clear from the diagrams that, for a sinusoid sin(ω*t*), the scale factor *a* is related (inversely) to the radian frequency ω. This general inverse relationship between scale and frequency holds for signals in general. See "CWT as a Filtering Technique" on page 1-27 and "Scale and Frequency" on page 1-24 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.

Consider a lunar landscape, whose ragged surface (simulated below) is a result of centuries of bombardment by meteorites whose sizes range from gigantic boulders to dust specks.

If we think of this surface in cross section as a one-dimensional signal, then it is reasonable to think of the signal as having components of different scales — large features carved by the impacts of large meteorites, and finer features created by small meteorites.



Here is a case where thinking in terms of scale makes much more sense than thinking in terms of frequency.

Even though this signal is artificial, many natural phenomena — from the intricate branching of blood vessels and trees, to the jagged surfaces of mountains and fractured metals — lend themselves to an analysis of scale.

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.

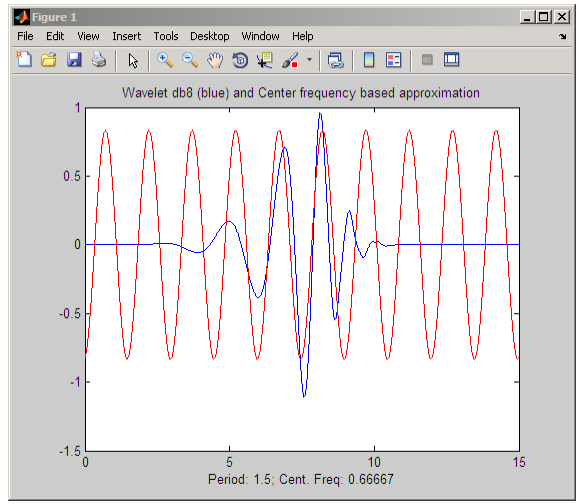
To summarize, the general correspondence between scale and frequency is:

- Low scale $a \Rightarrow$ Compressed wavelet $\Rightarrow$ Rapidly changing details $\Rightarrow$ High frequency ω.

- High scale $a \Rightarrow$ Stretched wavelet $\Rightarrow$ Slowly changing, coarse features $\Rightarrow$ 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.

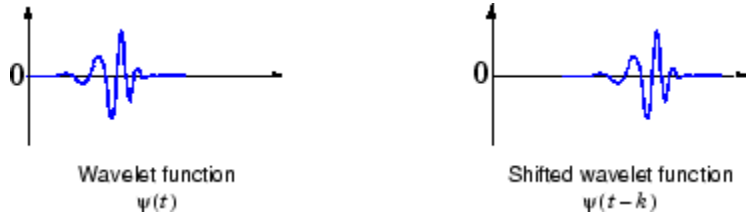
The basic approach identifies the peak power in the Fourier transform of the wavelet as its center frequency and divides that value by the product of the scale and the sampling interval. See `scal2frq` for details. The following example shows the match between the estimated center frequency of the `db8` 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 1-27.

Shifting

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

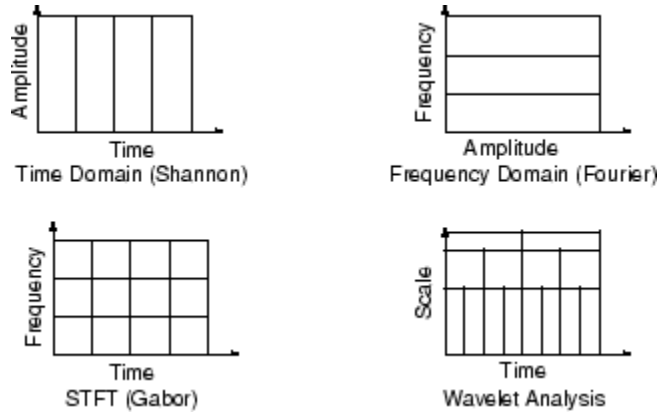


CWT as a Windowed Transform

In "Short-Time Fourier Transform" on page 1-16, 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:

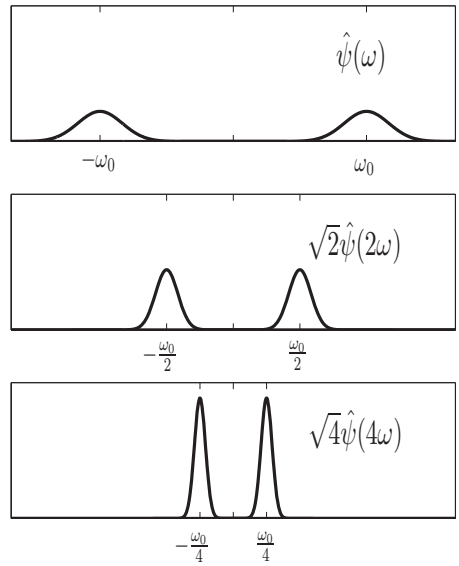
$$C(a,b;f(t),\psi(t)) = \int_{-\infty}^{\infty} f(t)\frac{1}{\sqrt{a}}\psi^{*}(\tfrac{t-b}{a})dt$$

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.

Continuous Wavelet Transform via the Inverse Discrete Fourier 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:

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

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:

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

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.

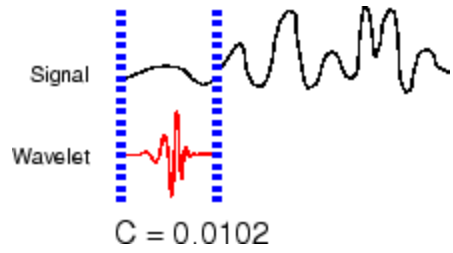
Five Easy Steps to a Continuous Wavelet Transform

Here are the five steps of an easy recipe for creating a CWT:

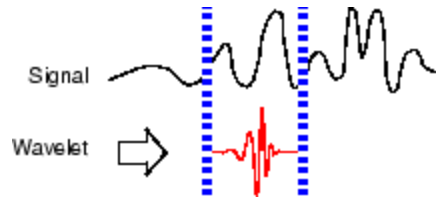
1 Take a wavelet and compare it to a section at the start of the original signal.

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 1-12 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.

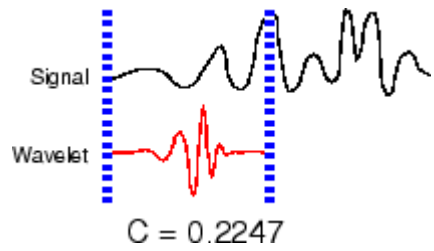
   As described in "Definition of the Continuous Wavelet Transform" on page 1-20, the CWT coefficients explicitly depend on the analyzing wavelet. Therefore, the CWT coefficients are different when you compute the CWT for the same signal using different wavelets.



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



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



5 Repeat steps 1 through 4 for all scales.

Interpreting the 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.

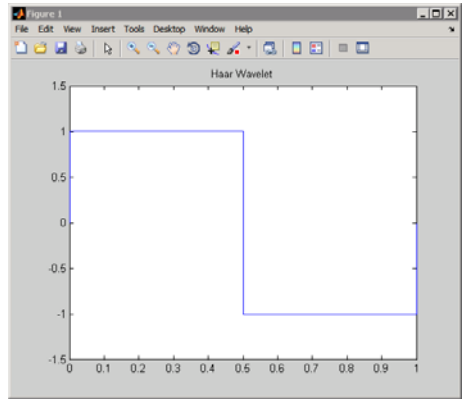
A signal feature that wavelets are very good at detecting is a discontinuity, or singularity. Abrupt transitions in signals result in wavelet coefficients with large absolute values.

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:

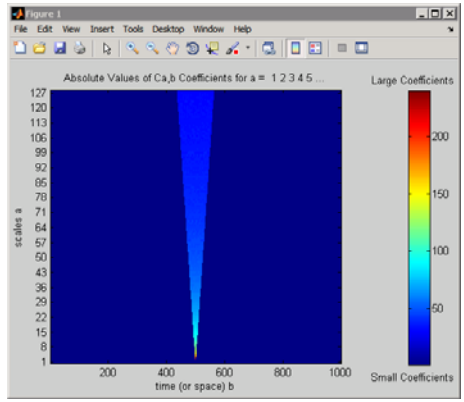
```
CWTcoeffs = cwt(x,1:128,'haar');
```

`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;
```



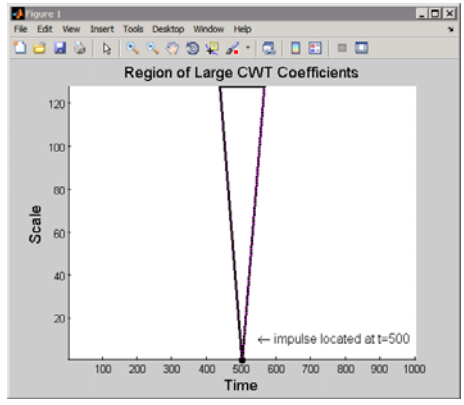
The preceding figure was modified with text labels to explicitly show which colors indicate large and small CWT coefficients.

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.



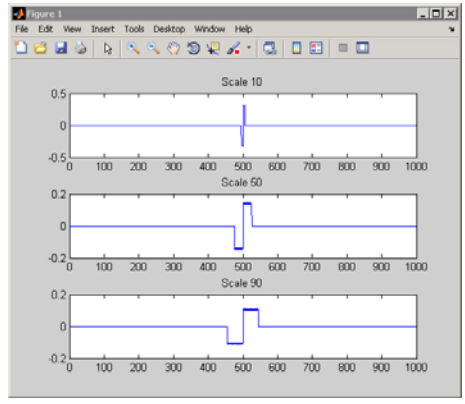
This region is referred to as the *cone of influence* of the point t=500 for the Haar wavelet. For a given point, the cone of influence shows you which CWT coefficients are affected by the signal value at that point.

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.

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

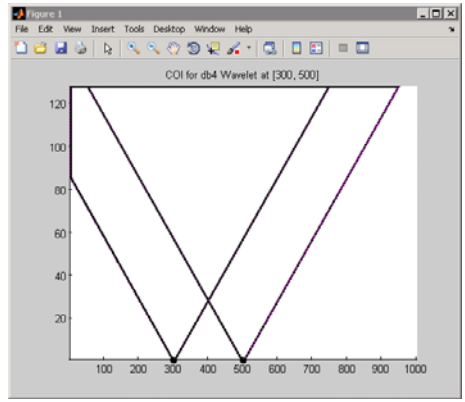
For the impulse, the CWT coefficients are equal to the conjugated, time-reversed, and scaled wavelet as a function of the shift parameter, b. You can see this by plotting the CWT coefficients for a select few scales.

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



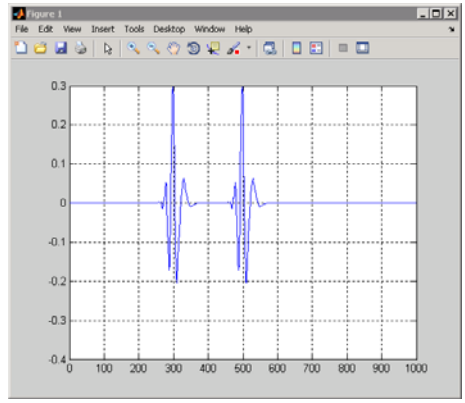
The cone of influence depends on the wavelet. You can find and plot the cone of influence for a specific wavelet with `conofinf`.

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.



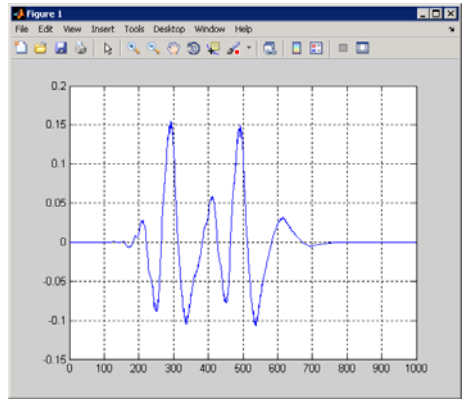
Look at point 400 for scale 20. At that scale, you can see that neither cone of influence overlaps the point 400. Therefore, you can expect that the CWT coefficient will be zero at that point and scale. The signal is only nonzero at two values, 300 and 500, and neither cone of influence for those values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

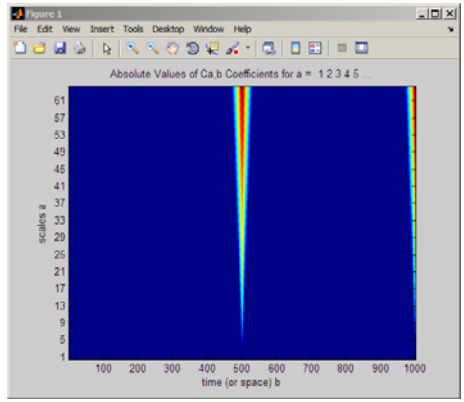


In the preceding example, the CWT coefficients became large in the vicinity of an abrupt change in the signal. This ability to detect discontinuities is a strength of the wavelet transform. The preceding example also demonstrated

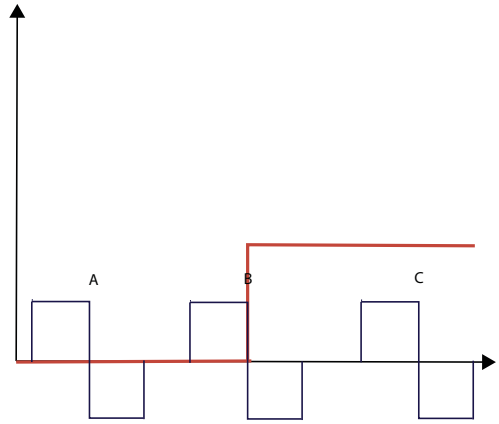
that the CWT coefficients localize the discontinuity best at small scales. At small scales, the small support of the wavelet ensures that the singularity only affects a small set of wavelet coefficients.

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.

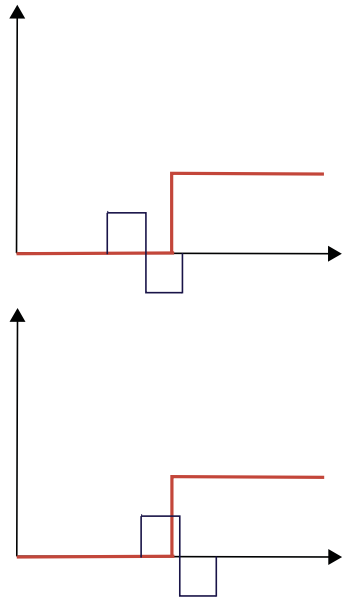
Note the Haar wavelet centered around position B. The negative part of the Haar wavelet overlaps with a region of the step function that is equal to 1. The CWT coefficients are negative because the product of the Haar wavelet and the unit step is a negative constant. Integrating over that area yields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



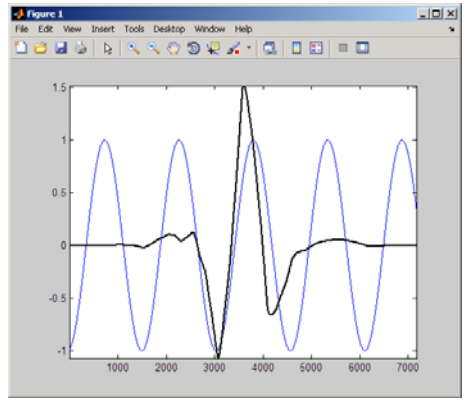
In the top figure, the wavelet has just begun to overlap with the unity portion of the unit step. In this case, the CWT coefficients are negative, but not as large in absolute value as those obtained at position B. In the bottom figure, the wavelet has shifted past position B and the positive deflection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

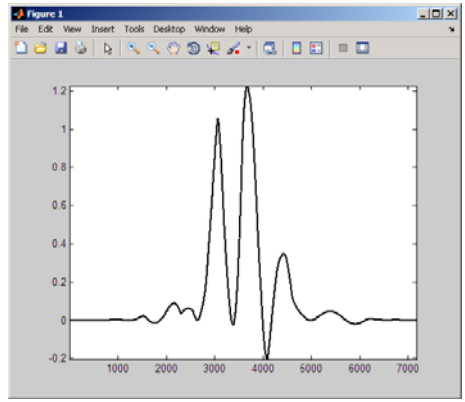
```
x = [zeros(500,1); ones(500,1)];
```

```
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.

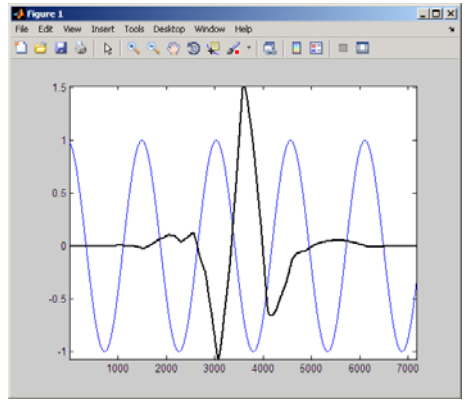


Recall that the CWT coefficients are obtained by computing the product of the signal with the shifted and scaled analyzing wavelet and integrating the result. The following figure shows the product of the wavelet and the sinusoid from the preceding figure.

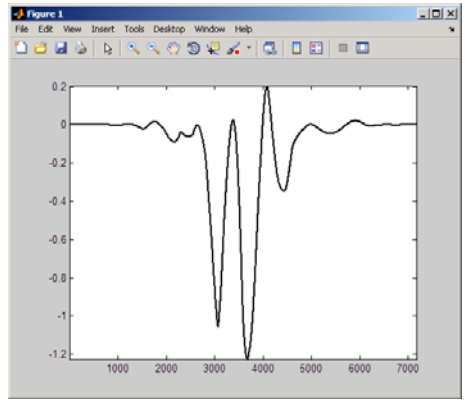


You can see that integrating over this product produces a positive CWT coefficient. That results because the oscillation in the wavelet approximately matches a period of the sine wave. The wavelet is *in phase* with the sine wave. The negative deflections of the wavelet approximately match the negative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

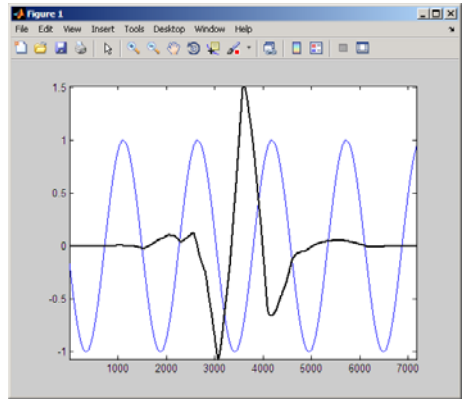


Examine the product of the shifted wavelet and the sinusoid.

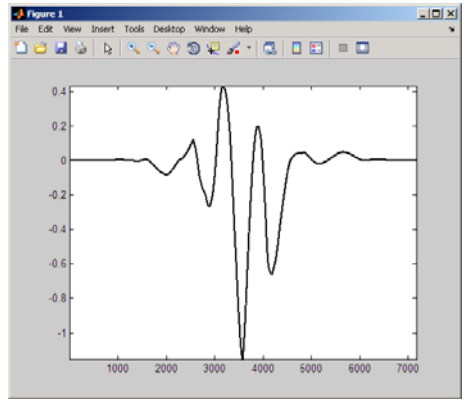


You can see that integrating over this product produces a negative CWT coefficient. That results because the wavelet is 1/2 cycle out of phase with the sine wave. The negative deflections of the wavelet approximately match the positive deflections of the sine wave. The positive deflections of the wavelet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

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

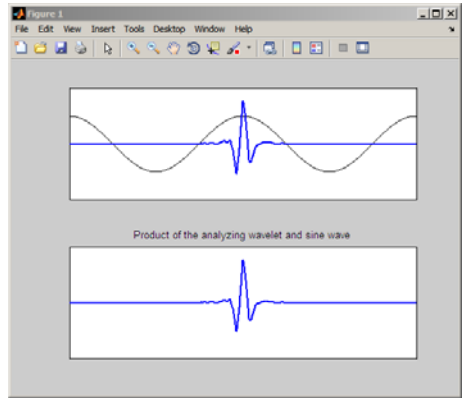


The following figure shows the product of the shifted wavelet and the sinusoid.



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.

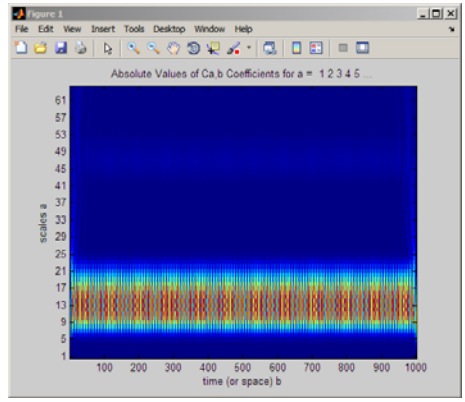
At scales where the oscillation in the wavelet occurs on either a much larger or smaller scale than the period of the sine wave, you obtain CWT coefficients near zero. The following figure illustrates the case where the wavelet oscillates on a much smaller scale than the sinusoid.



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;
```

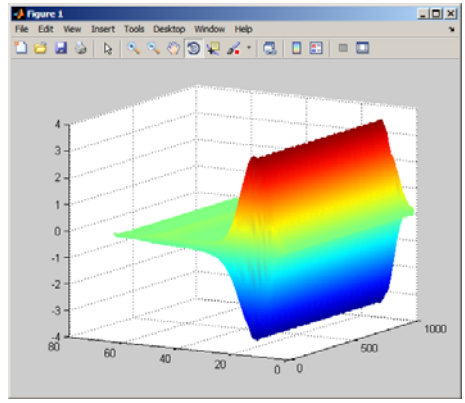


Note that the CWT coefficients are large in absolute value around scales 9 to 21. You can find the pseudo-frequencies corresponding to these scales using the command:

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

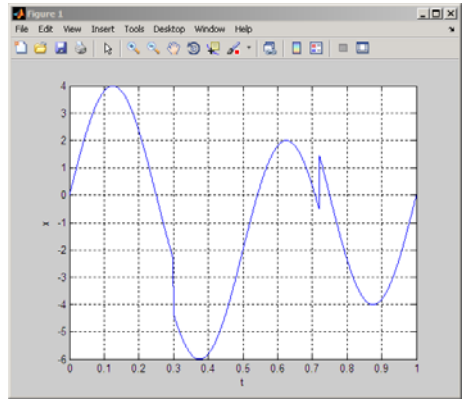
Note that the CWT coefficients are large at scales near the frequency of the sine wave. You can clearly see the sinusoidal pattern in the CWT coefficients at these scales with the following code.

```
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.

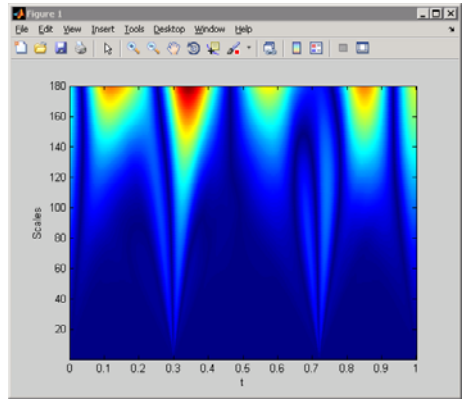
```
N = 1024;
t = linspace(0,1,1024);
x = 4*sin(4*pi*t);
x = x - sign(t - .3) - sign(.72 - t);
plot(t,x); xlabel('t'); ylabel('x');
grid on;
```



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.

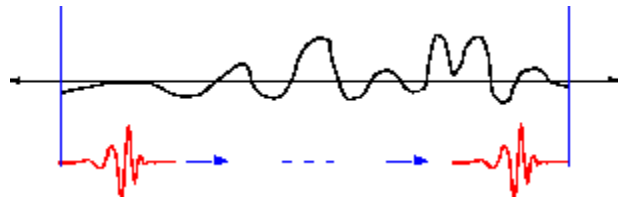
What's Continuous About the Continuous Wavelet Transform?

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 it?

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.



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"). 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]*)*.

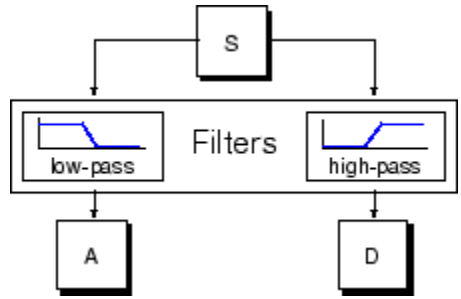
This very practical filtering algorithm yields a *fast wavelet transform* — a box into which a signal passes, and out of which wavelet coefficients quickly emerge. Let's examine this in more depth.

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.

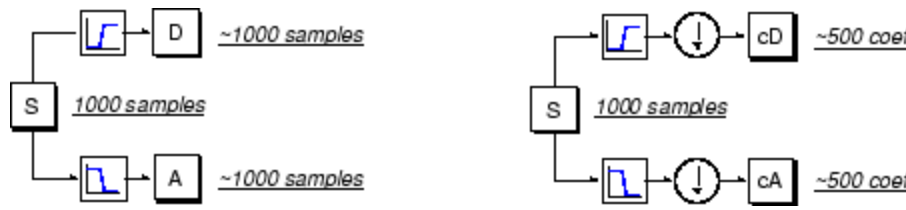
The filtering process, at its most basic level, looks like this.



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.

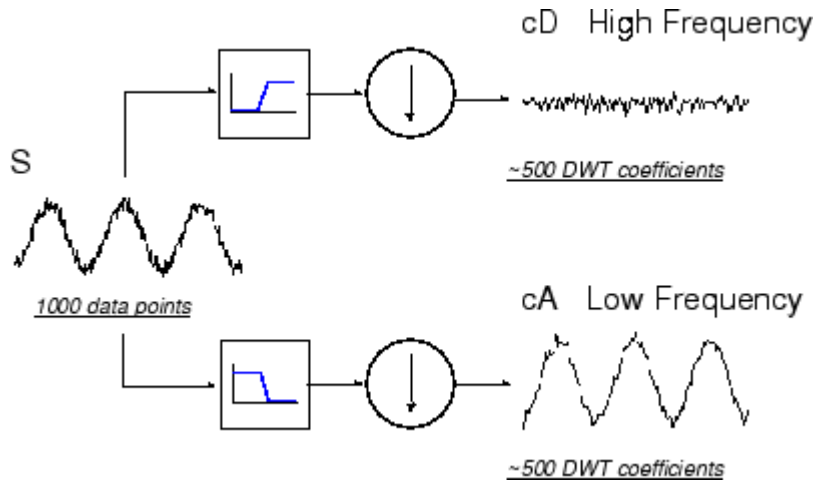
These signals A and D are interesting, but we get 2000 values instead of the 1000 we had. There exists a more subtle way to perform the decomposition using wavelets. By looking carefully at the computation, we may keep only one point out of two in each of the two 2000-length samples to get the complete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

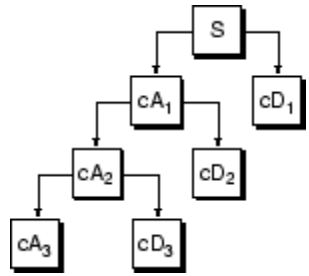
Notice that the detail coefficients `cD` are small and consist mainly of a high-frequency noise, while the approximation coefficients `cA` contain much less noise than does the original signal.

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

ans =
   501   501
```

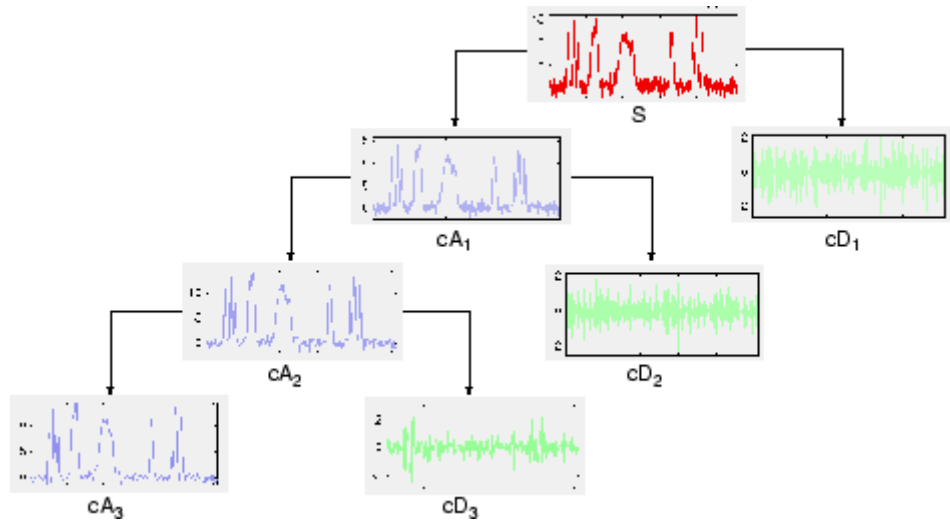
You may observe that the actual lengths of the detail and approximation coefficient vectors are slightly *more* than half the length of the original signal. This has to do with the filtering process, which is implemented by convolving the signal with a filter. The convolution “smears” the signal, introducing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

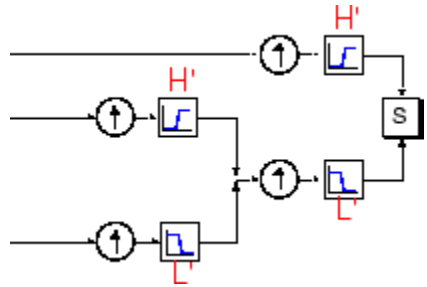
Since the analysis process is iterative, in theory it can be continued indefinitely. In reality, the decomposition can proceed only until the individual details consist of a single sample or pixel. In practice, you'll select

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*).

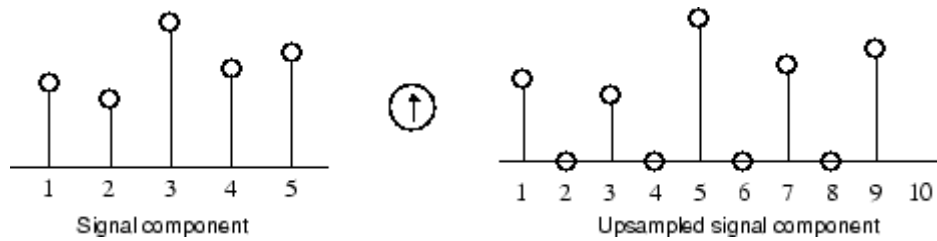
Wavelet Reconstruction

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.



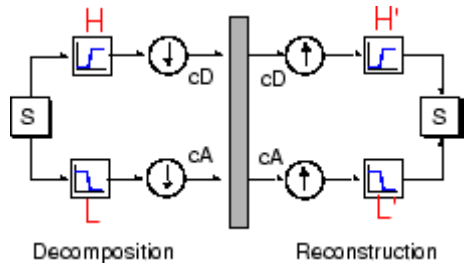
The toolbox includes commands, like `idwt` and `waverec`, that perform single-level or multilevel reconstruction, respectively, on the components of one-dimensional signals. These commands have their two-dimensional and three-dimensional analogs, `idwt2`, `waverec2`, `idwt3`, and `waverec3`.

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.

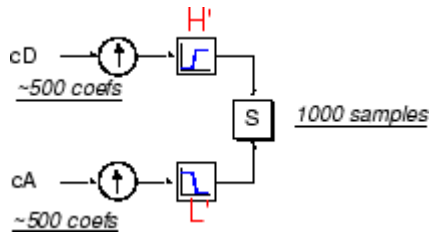
The downsampling of the signal components performed during the decomposition phase introduces a distortion called aliasing. It turns out that by carefully choosing filters for the decomposition and reconstruction phases that are closely related (but not identical), we can "cancel out" the effects of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



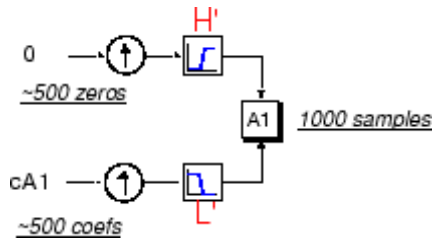
Reconstructing Approximations and Details

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



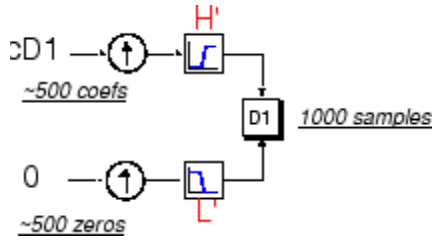
It is also possible to reconstruct the approximations and details themselves from their coefficient vectors. As an example, let's consider how we would reconstruct the first-level approximation `A1` from the coefficient vector `cA1`.

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:



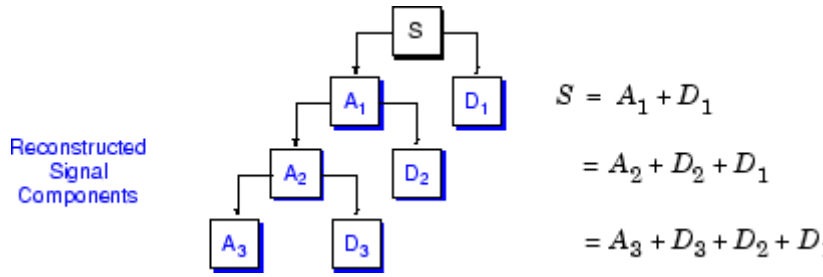
The reconstructed details and approximations are true constituents of the original signal. In fact, we find when we combine them that

$$A_1 + D_1 = S.$$

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:



Relationship of Filters to Wavelet Shapes

In the section "Reconstruction Filters" on page 1-57, 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.

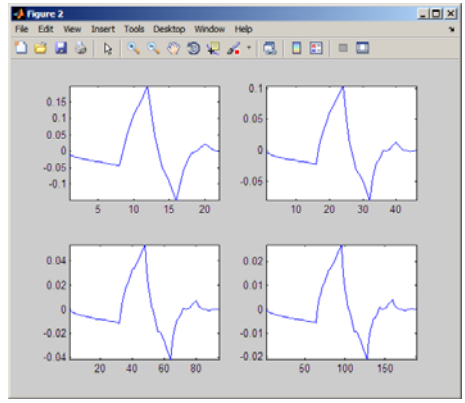
Consider the low-pass reconstruction filter (`L'`) for the `db2` wavelet.

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

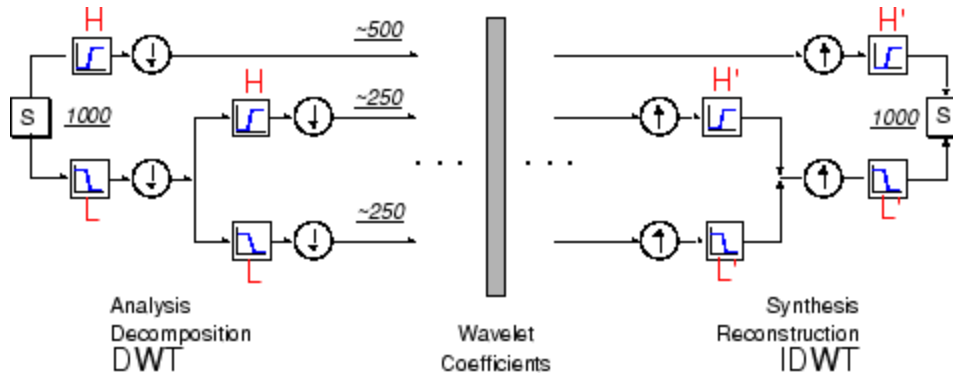
We've seen the interrelation of wavelets and quadrature mirror filters. The wavelet function ψ is determined by the high-pass filter, which also produces the details of the wavelet decomposition.

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.

Multistep Decomposition and Reconstruction

A multistep analysis-synthesis process can be represented as



This process involves two aspects: breaking up a signal to obtain the wavelet coefficients, and reassembling the signal from the coefficients.

We've already discussed decomposition and reconstruction at some length. Of course, there is no point breaking up a signal merely to have the satisfaction of immediately reconstructing it. We may modify the wavelet coefficients before performing the reconstruction step. We perform wavelet analysis

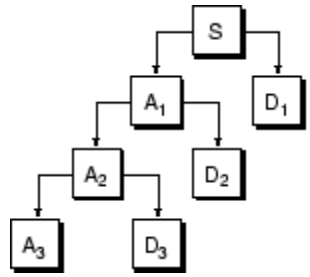
because the coefficients thus obtained have many known uses, de-noising and compression being foremost among them.

But wavelet analysis is still a new and emerging field. No doubt, many uncharted uses of the wavelet coefficients lie in wait. The toolbox can be a means of exploring possible uses and hitherto unknown applications of wavelet analysis. Explore the toolbox functions and see what you discover.

Wavelet Packet Analysis

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

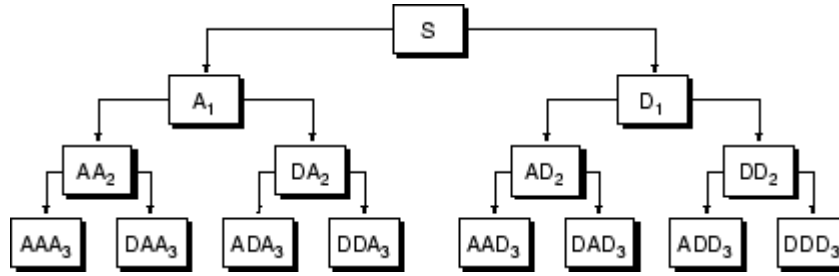
In wavelet analysis, a signal is split into an approximation and a detail. The approximation is then itself split into a second-level approximation and detail, and the process is repeated. For an `n`-level decomposition, there are `n+1` possible ways to decompose or encode the signal.



$$S = A_1 + D_1$$
$$= A_2 + D_2 + D_1$$
$$= A_3 + D_3 + D_2 + D_1$$

In wavelet packet analysis, the details as well as the approximations can be split.

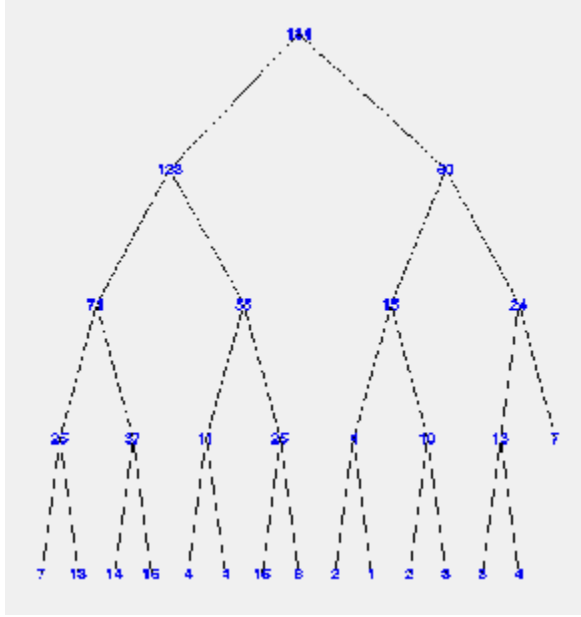
This yields more than $2^{2^{n-1}}$ different ways to encode the signal. This is the *wavelet packet decomposition tree*.



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"), 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.

History of Wavelets

From an historical point of view, wavelet analysis is a new method, though its mathematical underpinnings date back to the work of Joseph Fourier in the nineteenth century. Fourier laid the foundations with his theories of frequency analysis, which proved to be enormously important and influential.

The attention of researchers gradually turned from frequency-based analysis to scale-based analysis when it started to become clear that an approach measuring average fluctuations at different scales might prove less sensitive to noise.

The first recorded mention of what we now call a “wavelet” seems to be in 1909, in a thesis by Alfred Haar.

The concept of wavelets in its present theoretical form was first proposed by Jean Morlet and the team at the Marseille Theoretical Physics Center working under Alex Grossmann in France.

The methods of wavelet analysis have been developed mainly by Y. Meyer and his colleagues, who have ensured the methods’ dissemination. The main algorithm dates back to the work of Stephane Mallat in 1988. Since then, research on wavelets has become international. Such research is particularly active in the United States, where it is spearheaded by the work of scientists such as Ingrid Daubechies, Ronald Coifman, and Victor Wickerhauser.

Barbara Burke Hubbard describes the birth, the history, and the seminal concepts in a very clear text. See *The World According to Wavelets,* A.K. Peters, Wellesley, 1996.

The wavelet domain is growing up very quickly. A lot of mathematical papers and practical trials are published every month.

Introduction to the 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.

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.

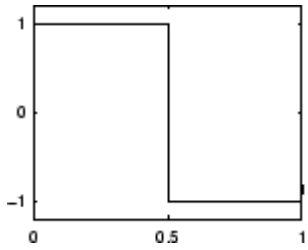
3 Select a family from the **Wavelet** menu at the top right of the tool.

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. Haar wavelet is discontinuous, and resembles a step function. It represents the same wavelet as Daubechies `db1`. See “Haar” in the *Wavelet Toolbox User’s Guide* for more detail.

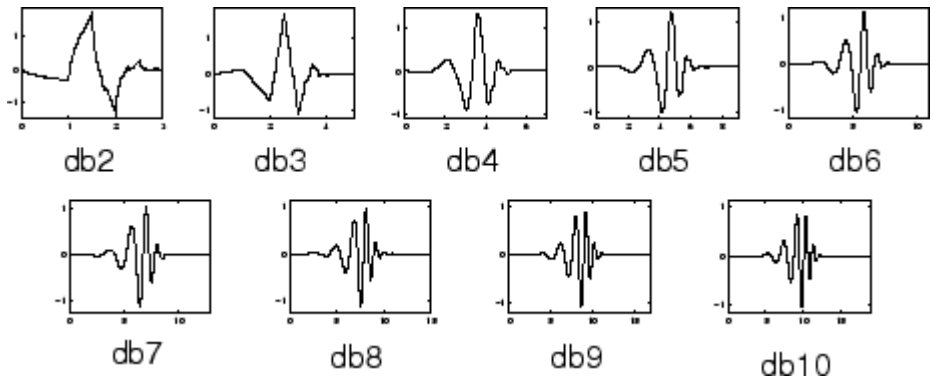


Wavelet function psi

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:

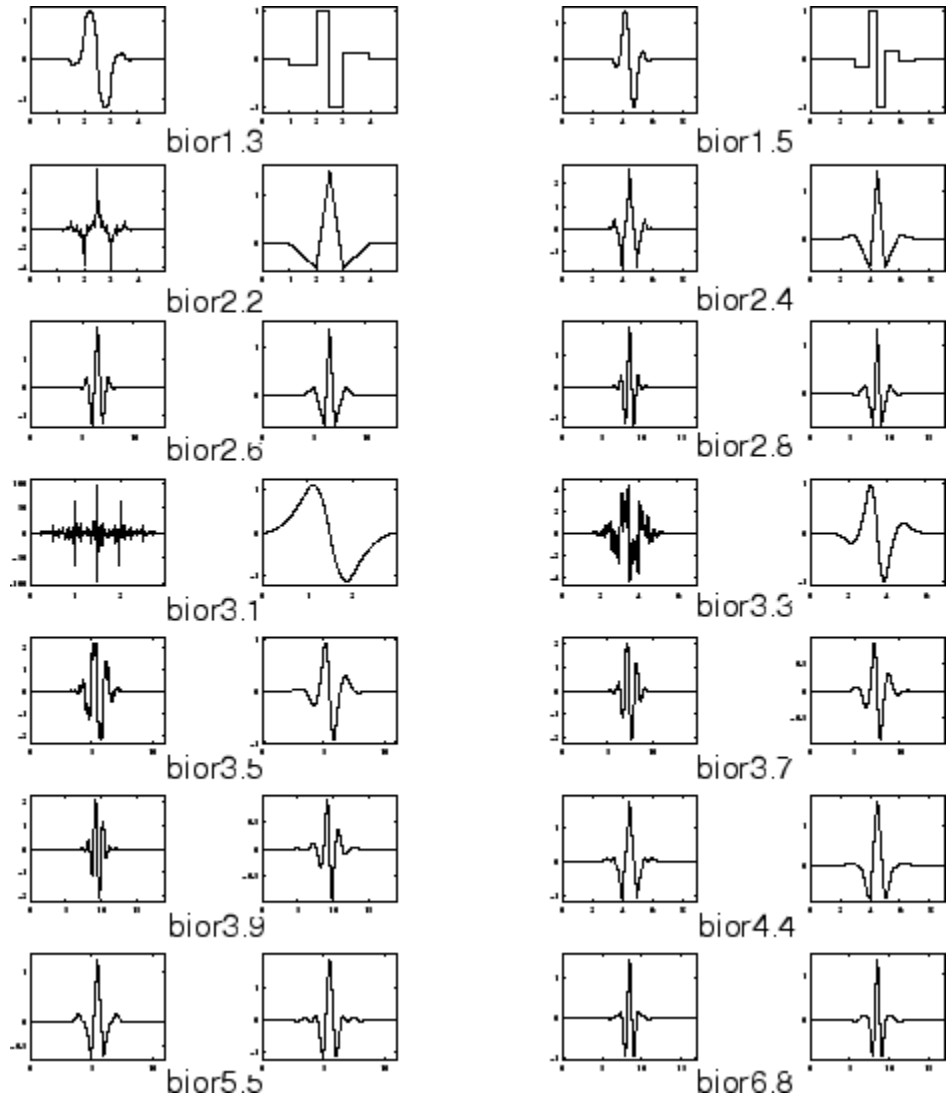


You can obtain a survey of the main properties of this family by typing `waveinfo('db')` from the MATLAB command line. See "Daubechies Wavelets: dbN" in the *Wavelet Toolbox User's Guide* for more detail.

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.



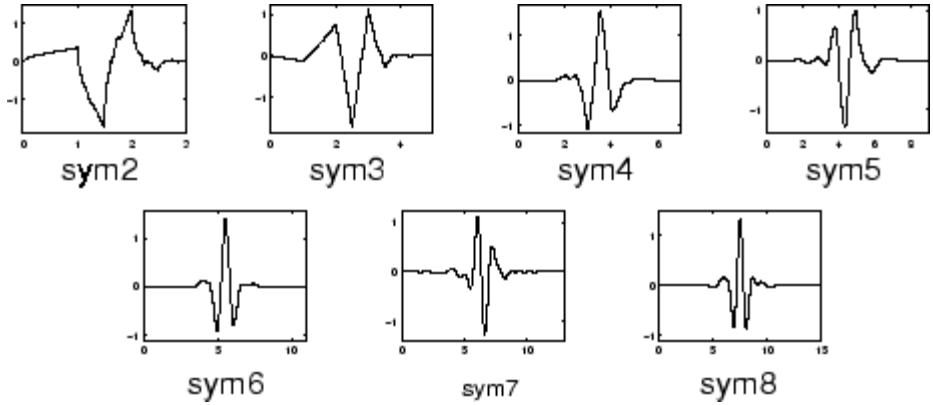
You can obtain a survey of the main properties of this family by typing `waveinfo('bior')` from the MATLAB command line. See "Biorthogonal Wavelet Pairs: biorNr.Nd" in the *Wavelet Toolbox User's Guide* for more detail.

Coiflets

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

Symlets

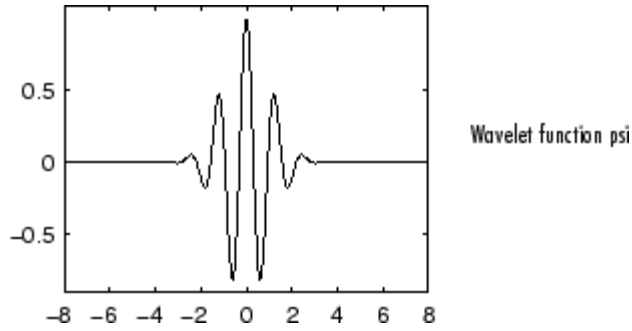
The symlets are nearly symmetrical wavelets proposed by Daubechies as modifications to the `db` family. The properties of the two wavelet families are similar. Here are the wavelet functions psi.



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

Morlet

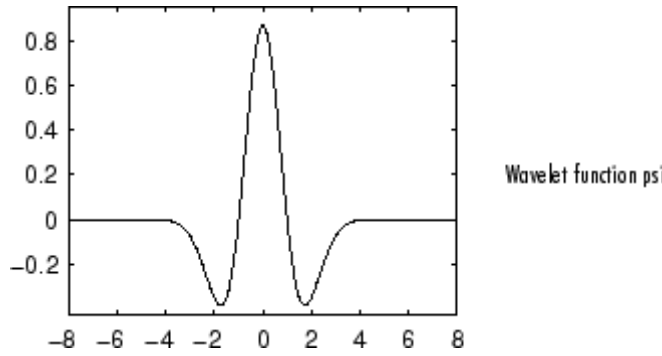
This wavelet has no scaling function, but is explicit.



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

Mexican Hat

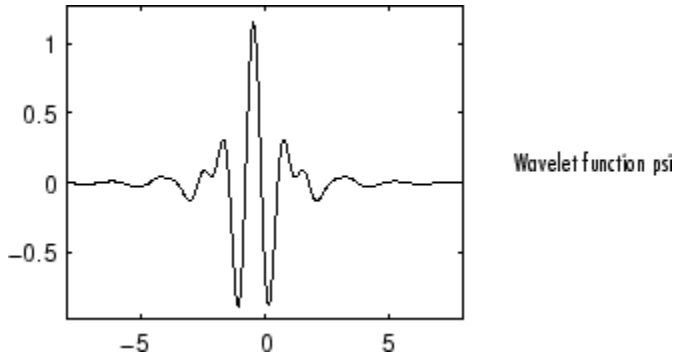
This wavelet has no scaling function and is derived from a function that is proportional to the second derivative function of the Gaussian probability density function.



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

Meyer

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



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

Other Real Wavelets

Some other real wavelets are available in the toolbox:

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

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

Complex Wavelets

Some complex wavelet families are available in the toolbox:

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

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

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 (see "Using Wavelet Packets" in the *Wavelet Toolbox User's Guide*)
- 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 disk or workspace. For more information, see "File Menu Options" in the *Wavelet Toolbox User's Guide*.

One-Dimensional Continuous Wavelet Analysis

This section takes you through the features of continuous wavelet analysis using Wavelet Toolbox software.

The toolbox requires only one function for continuous wavelet analysis: `cwt`. You'll find full information about this function in its reference page.

In this section, you'll learn how to

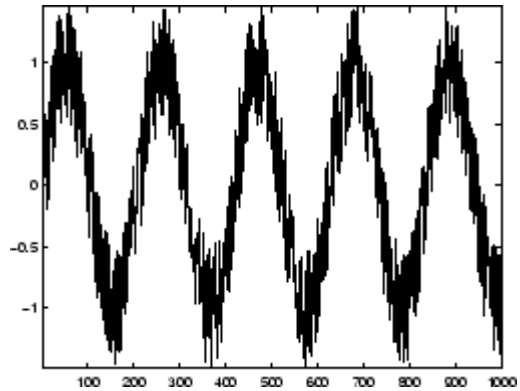
- Load a signal
- Perform a continuous wavelet transform of a signal
- Produce a plot of the coefficients
- Produce a plot of coefficients at a given scale
- Produce a plot of local maxima of coefficients across scales
- Select the displayed plots
- Switch from scale to pseudo-frequency information
- Zoom in on detail
- Display coefficients in normal or absolute mode
- Choose the scales at which analysis is performed

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

The final subsection discusses how to exchange signal and coefficient information between the disk and the graphical tools.

Continuous Analysis Using the Command Line

This example involves a noisy sinusoidal signal.



1 Load a signal.

From the MATLAB prompt, type

```
load noissin;
```

You now have the signal `noissin` in your workspace:

```
whos
```

Name	Size	Bytes	Class
noissin	1x1000	8000	double array

2 Perform a Continuous Wavelet Transform.

Use the `cwt` command. Type

```
c = cwt(noissin,1:48,'db4');
```

The arguments to `cwt` specify the signal to be analyzed, the scales of the analysis, and the wavelet to be used. The returned argument `c` contains the coefficients at various scales. In this case, `c` is a 48-by-1000 matrix with each row corresponding to a single scale.

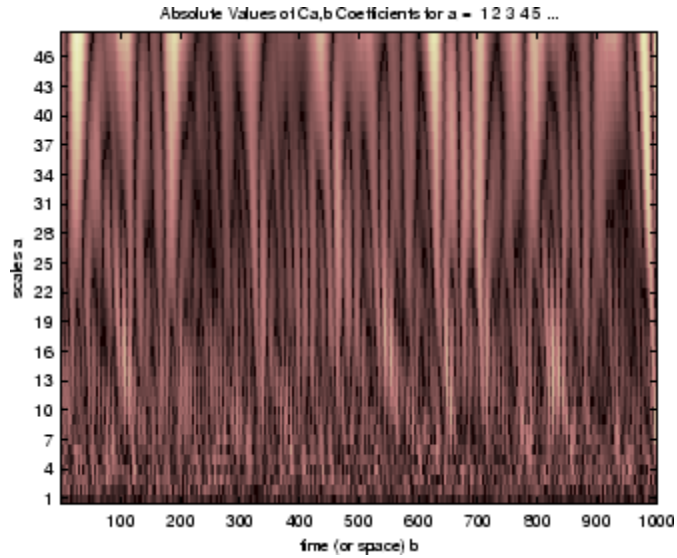
3 Plot the coefficients.

The `cwt` command accepts a fourth argument. This is a flag that, when present, causes `cwt` to produce a plot of the absolute values of the continuous wavelet transform coefficients.

The `cwt` command can accept more arguments to define the different characteristics of the produced plot. For more information, see the `cwt` reference page.

```
c = cwt(noissin,1:48,'db4','plot');
```

A plot appears.



Of course, coefficient plots generated from the command line can be manipulated using ordinary MATLAB graphics commands.

4 Choose scales for the analysis.

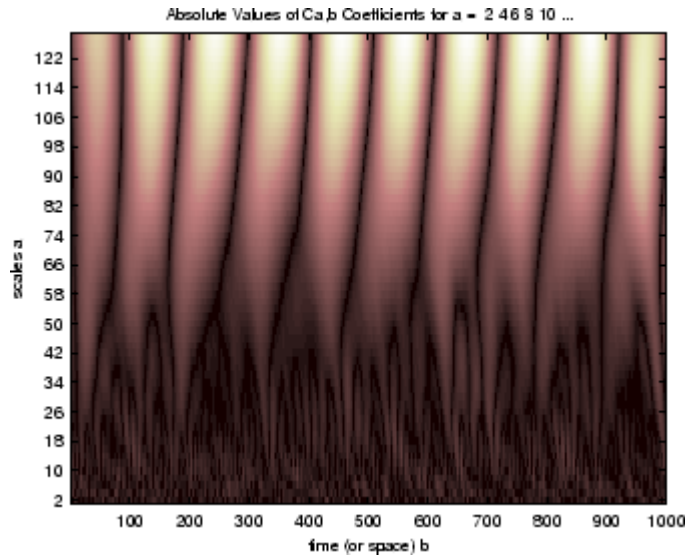
The second argument to `cwt` gives you fine control over the scale levels on which the continuous analysis is performed. In the previous example, we used all scales from 1 to 48, but you can construct any scale vector subject to these constraints:

- All scales must be real positive numbers.
- The scale increment must be positive.
- The highest scale cannot exceed a maximum value depending on the signal.

Let's repeat the analysis using every other scale from 2 to 128. Type

```
c = cwt(noissin,2:2:128,'db4','plot');
```

A new plot appears:



This plot gives a clearer picture of what's happening with the signal, highlighting the periodicity.

Continuous Analysis Using the Graphical Interface

We now use the **Continuous Wavelet 1-D** tool to analyze the same noisy sinusoidal signal we examined earlier using the command line interface in "Continuous Analysis Using the Command Line" on page 2-5.

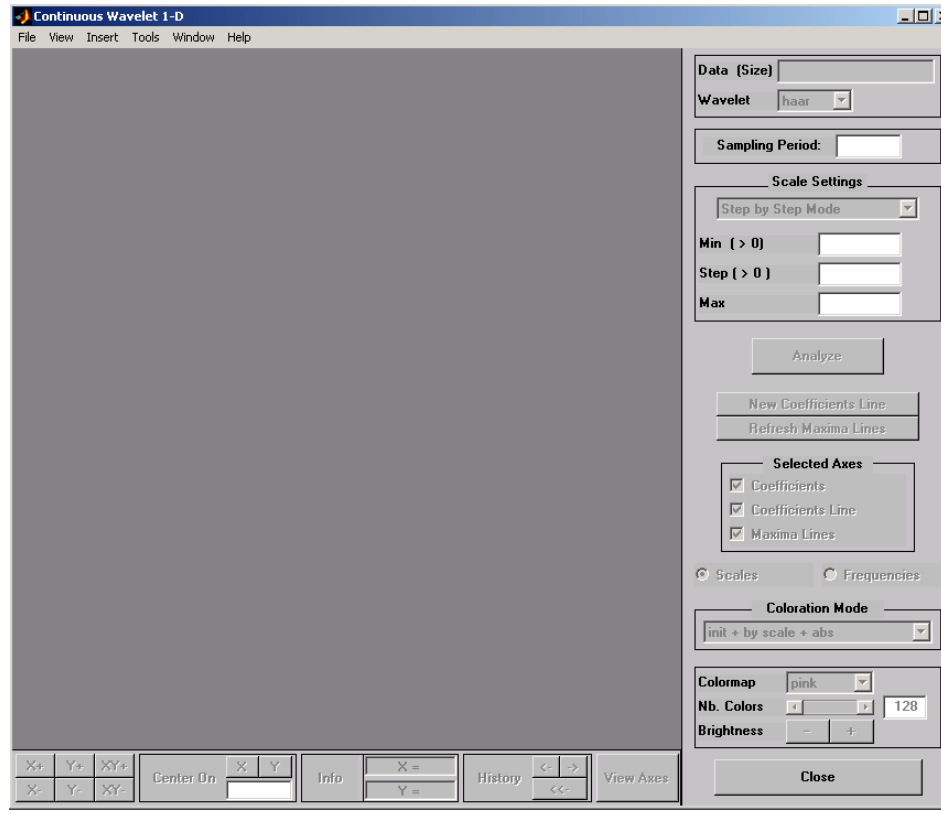
1 Start the Continuous Wavelet 1-D Tool. From the MATLAB prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.

Click the **Continuous Wavelet 1-D** menu item.

The continuous wavelet analysis tool for one-dimensional signal data appears.



2 Load a signal.

Choose the **File > Load Signal** menu option.

When the **Load Signal** dialog box appears, select the demo MAT-file `noissin.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`. Click the **OK** button.

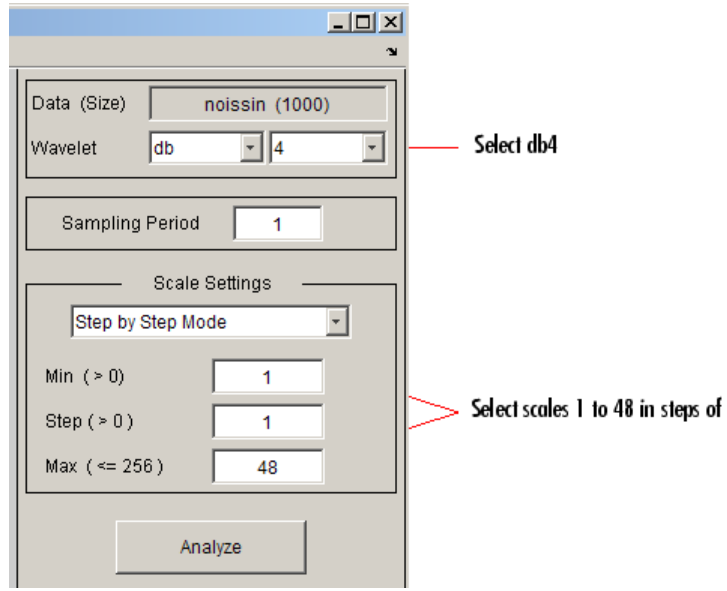
The noisy sinusoidal signal is loaded into the **Continuous Wavelet 1-D** tool.

The default value for the sampling period is equal to 1 (second).

3 Perform a Continuous Wavelet Transform.

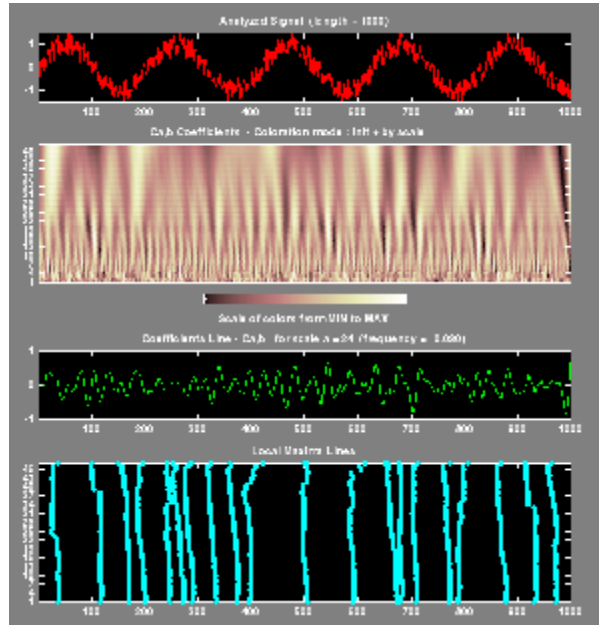
To start our analysis, let's perform an analysis using the `db4` wavelet at scales 1 through 48, just as we did using command line functions in the previous section.

In the upper right portion of the **Continuous Wavelet 1-D** tool, select the `db4` wavelet and scales 1–48.



4 Click the **Analyze** button.

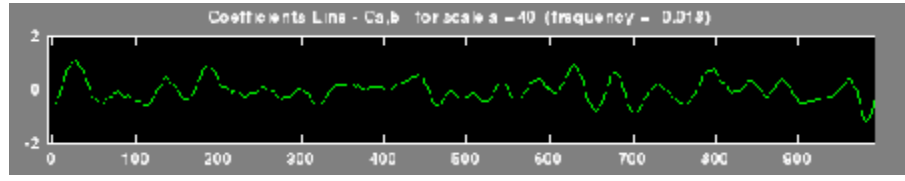
After a pause for computation, the tool displays the coefficients plot, the coefficients line plot corresponding to the scale a = 24, and the local maxima plot, which displays the chaining across scales (from a = 48 down to a = 1) of the coefficients local maxima.



5 View Wavelet Coefficients Line.

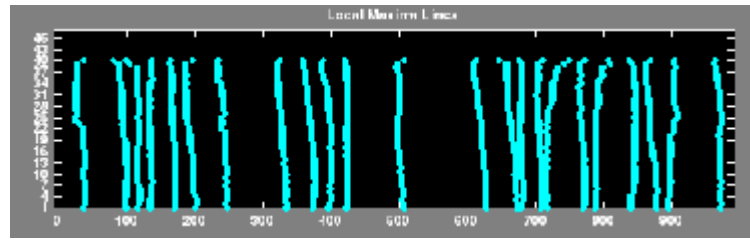
Select another scale a = 40 by clicking in the coefficients plot with the right mouse button. See step 9 to know, more precisely, how to select the desired scale.

Click the **New Coefficients Line** button. The tool updates the plot.

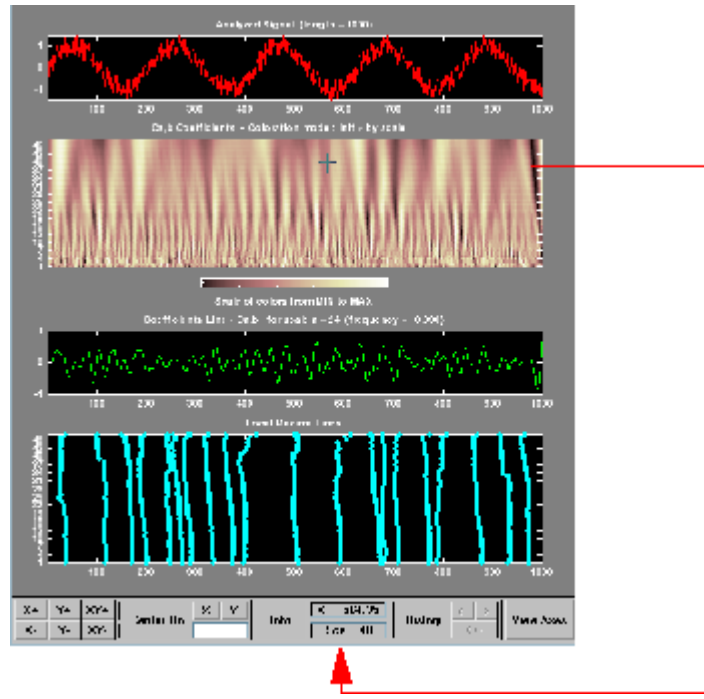


6 View Maxima Line.

Click the **Refresh Maxima Line** button. The local maxima plot displays the chaining across scales of the coefficients local maxima from a = 40 down to a = 1.



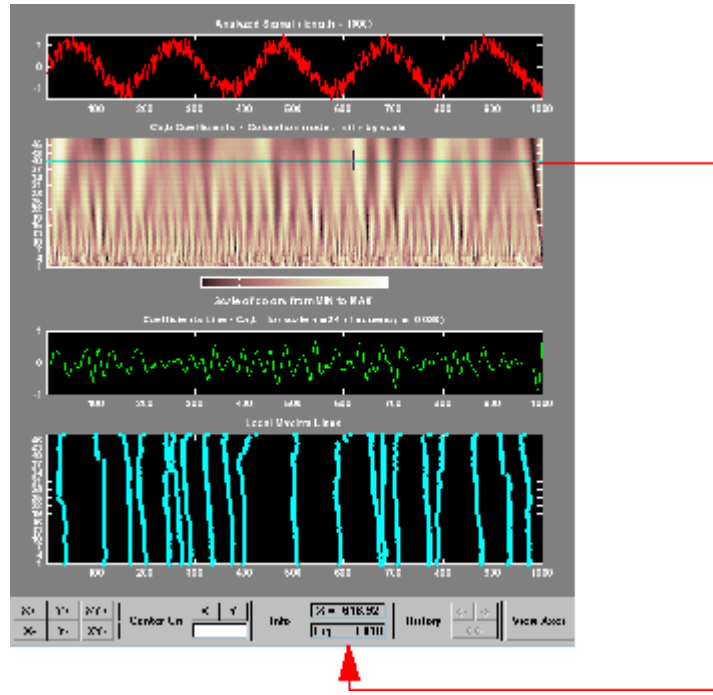
Hold down the right mouse button over the coefficients plot. The position of the mouse is given by the **Info** frame (located at the bottom of the screen) in terms of location (**X**) and scale (**Sca**).



7 Switch from scale to Pseudo-Frequency Information.

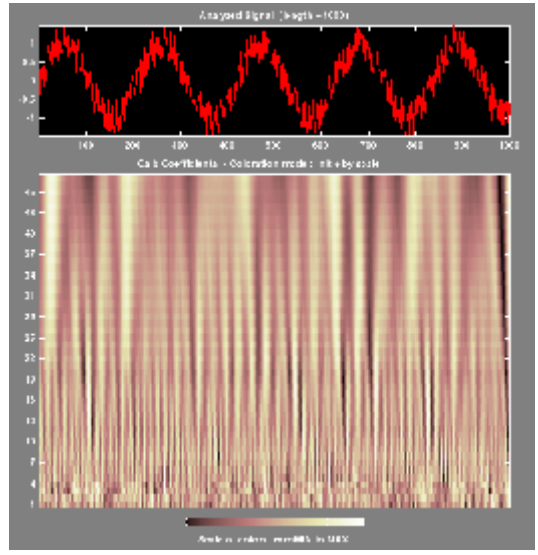
Using the option button on the right part of the screen, select **Frequencies** instead of **Scales**. Again hold down the right mouse button over the

coefficients plot, the position of the mouse is given in terms of location (**X**) and frequency (**Frq**) in Hertz.



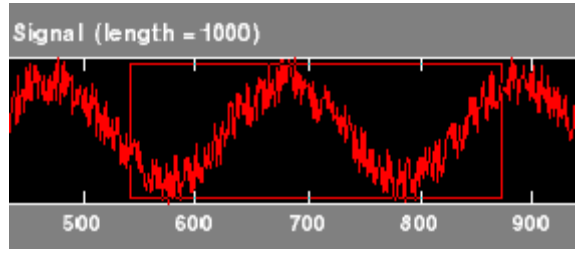
This facility allows you to interpret scale in terms of an associated pseudo-frequency, which depends on the wavelet and the sampling period. For more information on the connection between scale and frequency, see "How to Connect Scale to Frequency?" in the *Wavelet Toolbox User's Guide*.

8 Deselect the last two plots using the check boxes in the **Selected Axes** frame.



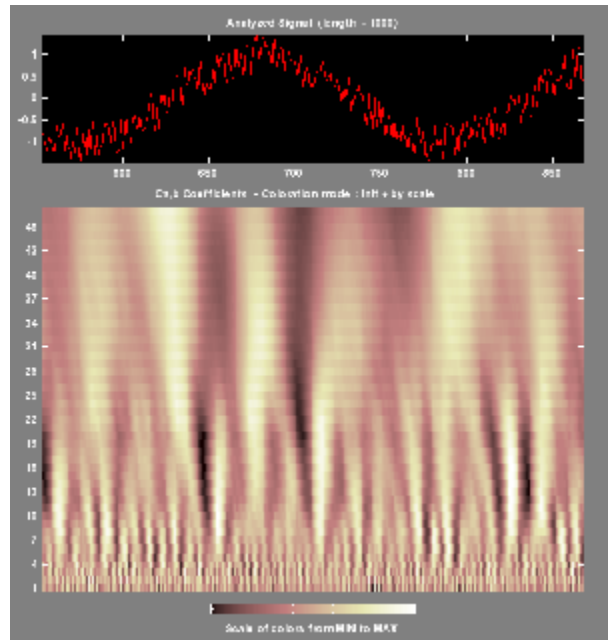
9 Zoom in on detail.

Drag a rubber band box (by holding down the left mouse button) over the portion of the signal you want to magnify.



10 Click the **X+** button (located at the bottom of the screen) to zoom horizontally only.

The **Continuous Wavelet 1-D** tool enlarges the displayed signal and coefficients plot (for more information on zooming, see "Connection of Plots" in the *Wavelet Toolbox User's Guide*).



As with the command line analysis on the preceding pages, you can change the scales or the analyzing wavelet and repeat the analysis. To do this, just edit the necessary fields and click the **Analyze** button.

11 View normal or absolute coefficients.

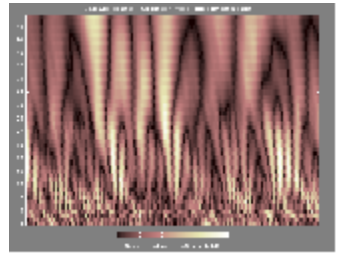
The **Continuous Wavelet 1-D** tool allows you to plot either the absolute values of the wavelet coefficients, or the coefficients themselves.

More generally, the coefficients coloration can be done in several different ways. For more details on the Coloration Mode, see "Controlling the Coloration Mode".

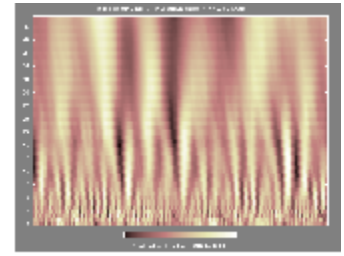
Choose either one of the absolute modes or normal modes from the **Coloration Mode** menu. In normal modes, the colors are scaled between the minimum and maximum of the coefficients. In absolute modes, the

colors are scaled between zero and the maximum absolute value of the coefficients.

The coefficients plot is redisplayed in the mode you select.



Absolute Mode



Normal Mode

Importing and Exporting Information from the Graphical Interface

The Continuous Wavelet 1-D graphical interface tool lets you import information from and export information to disk.

You can

- Load signals from disk into the **Continuous Wavelet 1-D** tool.
- Save wavelet coefficients from the **Continuous Wavelet 1-D** tool to disk.

Loading Signals into the Continuous Wavelet 1-D Tool

To load a signal you've constructed in your MATLAB workspace into the **Continuous Wavelet 1-D** tool, save the signal in a MAT-file (with extension `mat` or other).

For instance, suppose you've designed a signal called `warma` and want to analyze it in the **Continuous Wavelet 1-D** tool.

```
save warma warma
```

The workspace variable `warma` must be a vector.

```
sizwarma = size(warma)
```

```
sizwarma =
           1        1000
```

To load this signal into the **Continuous Wavelet 1-D** tool, use the menu option **File > Load Signal**. A dialog box appears that lets you select the appropriate MAT-file to be loaded.

Note The first one-dimensional variable encountered in the file is considered the signal. Variables are inspected in alphabetical order.

Saving Wavelet Coefficients

The Continuous Wavelet 1-D tool lets you save wavelet coefficients to disk. The toolbox creates a MAT-file in the current folder with the extension `wc1` and a name you give it.

To save the continuous wavelet coefficients from the present analysis, use the menu option **File > Save > Coefficients**.

A dialog box appears that lets you specify a folder and filename for storing the coefficients.

Consider the example analysis:

File > Example Analysis > with haar at scales [1:1:64] → Cantor curve.

After saving the continuous wavelet coefficients to the file `cantor.wc1`, load the variables into your workspace:

```
load cantor.wc1 -mat
whos
```

Name	Size	Bytes	Class
coeff	64x2188	1120256	double array
scales	1x64	512	double array
wname	1x4	8	char array

Variables `coefs` and `scales` contain the continuous wavelet coefficients and the associated scales. More precisely, in the above example, `coefs` is a 64-by-2188 matrix, one row for each scale; and `scales` is the 1-by-64 vector `1:64`. Variable `wname` contains the wavelet name.

One-Dimensional Complex Continuous Wavelet Analysis

This section takes you through the features of complex continuous wavelet analysis using the Wavelet Toolbox software and focuses on the differences between the real and complex continuous analysis.

You can refer to the section "One-Dimensional Continuous Wavelet Analysis" on page 2-4 if you want to learn how to

- Zoom in on detail
- Display coefficients in normal or absolute mode
- Choose the scales at which the analysis is performed
- Switch from scale to pseudo-frequency information
- Exchange signal and coefficient information between the disk and the graphical tools

Wavelet Toolbox software requires only one function for complex continuous wavelet analysis of a real valued signal: `cwt`. You'll find full information about this function in its reference page.

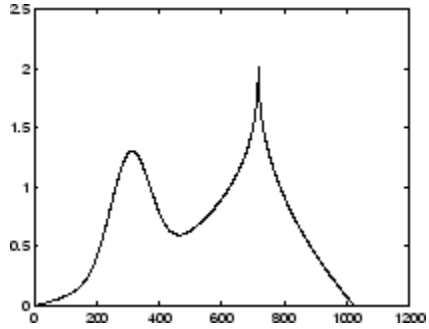
In this section, you'll learn how to

- Load a signal
- Perform a complex continuous wavelet transform of a signal
- Produce plots of the coefficients

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

Complex Continuous Analysis Using the Command Line

This example involves a cusp signal.



1 Load a signal.

From the MATLAB prompt, type

```
load cuspamax;
```

You now have the signal `cuspamax` in your workspace:

```
whos
```

Name	Size	Bytes	Class
caption	1x71	142	char array
cuspamax	1x1024	8192	double array

```
caption
```

```
caption =
       x = linspace(0,1,1024);
       y = exp(-128*((x-0.3).^2))-3*(abs(x-0.7).^0.4);
```

`caption` is a string that contains the signal definition.

2 Perform a Continuous Wavelet Transform.

Use the `cwt` command. Type

```
c = cwt(cuspamax,1:2:64,'cgau4');
```

The arguments to `cwt` specify the signal to be analyzed, the scales of the analysis, and the wavelet to be used. The returned argument `c` contains the coefficients at various scales. In this case, `c` is a complex 32-by-1024 matrix, each row of which corresponds to a single scale.

3 Plot the coefficients.

The `cwt` command accepts a fourth argument. This is a flag that, when present, causes `cwt` to produce four plots related to the complex continuous wavelet transform coefficients:

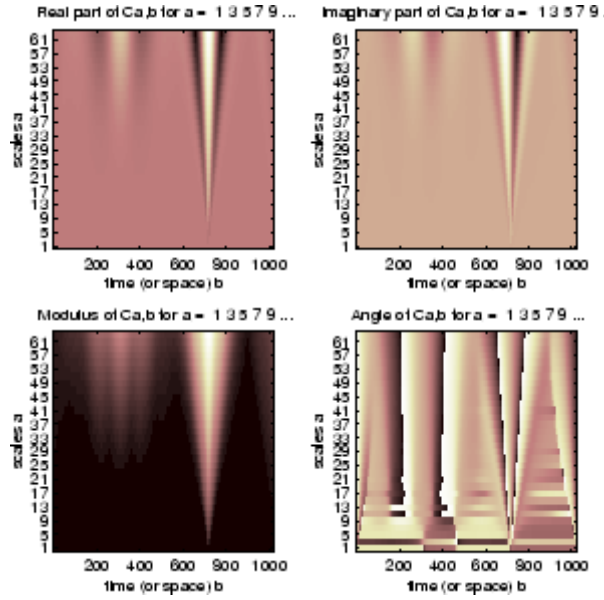
- Real and imaginary parts
- Modulus and angle

The `cwt` command can accept more arguments to define the different characteristics of the produced plots. For more information, see the `cwt` reference page.

Type

```
c = cwt(cuspamax,1:2:64,'cgau4','plot');
```

A plot appears:



Of course, coefficient plots generated from the command line can be manipulated using ordinary MATLAB graphics commands.

Complex Continuous Analysis Using the Graphical Interface

We now use the **Complex Continuous Wavelet 1-D** tool to analyze the same cusp signal we examined using the command line interface in the previous section.

1 Start the Complex Continuous Wavelet 1-D Tool.

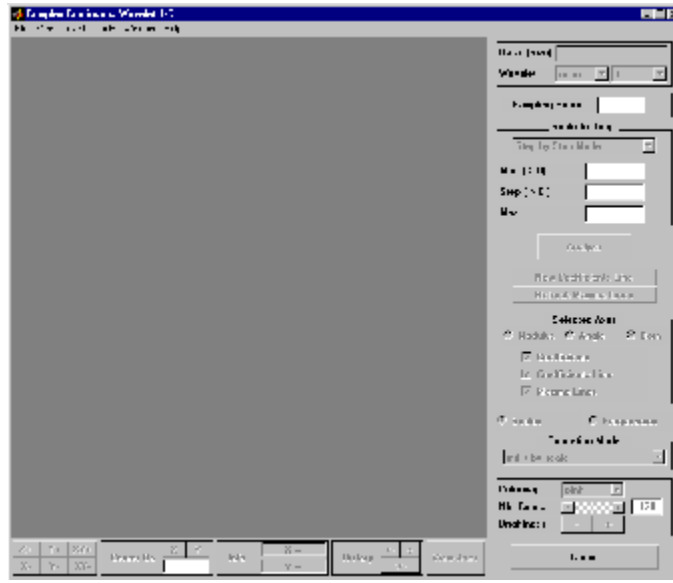
From the MATLAB prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.

Click the **Complex Continuous Wavelet 1-D** menu item.

The continuous wavelet analysis tool for one-dimensional signal data appears.



2 Load a signal.

Choose the **File > Load Signal** menu option.

When the **Load Signal** dialog box appears, select the demo MAT-file `cuspamax.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`. Click the **OK** button.

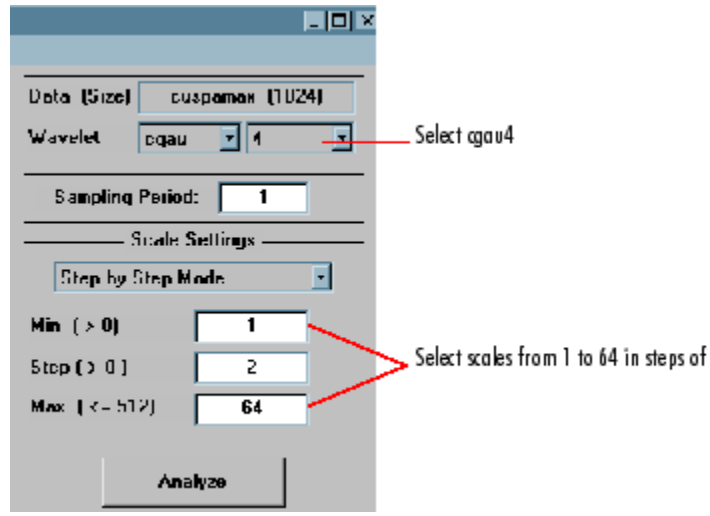
The cusp signal is loaded into the **Complex Continuous Wavelet 1-D** tool.

The default value for the sampling period is equal to 1 (second).

3 Perform a Complex Continuous Wavelet Transform

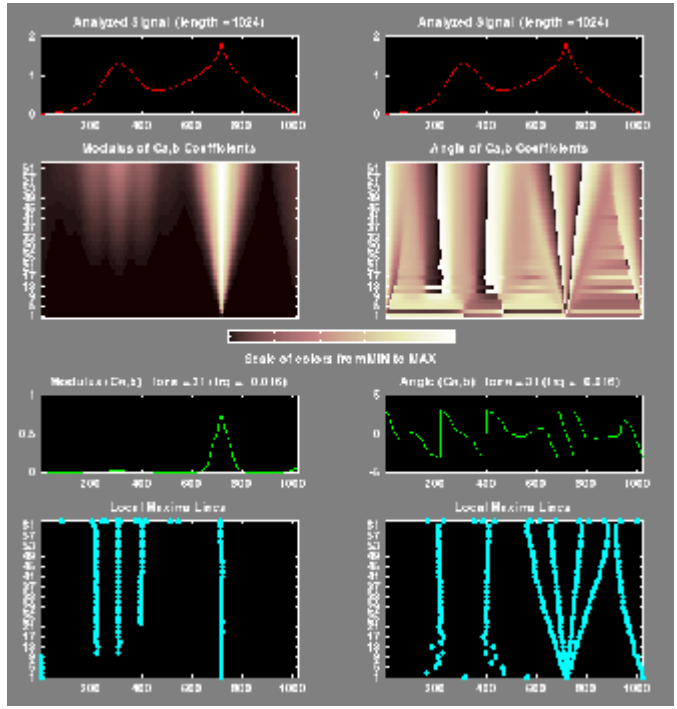
To start our analysis, let's perform an analysis using the `cgau4` wavelet at scales 1 through 64 in steps of 2, just as we did using command-line functions in "Complex Continuous Analysis Using the Command Line" on page 2-20.

In the upper-right portion of the **Complex Continuous Wavelet 1-D** tool, select the `cgau4` wavelet and scales 1–64 in steps of 2.



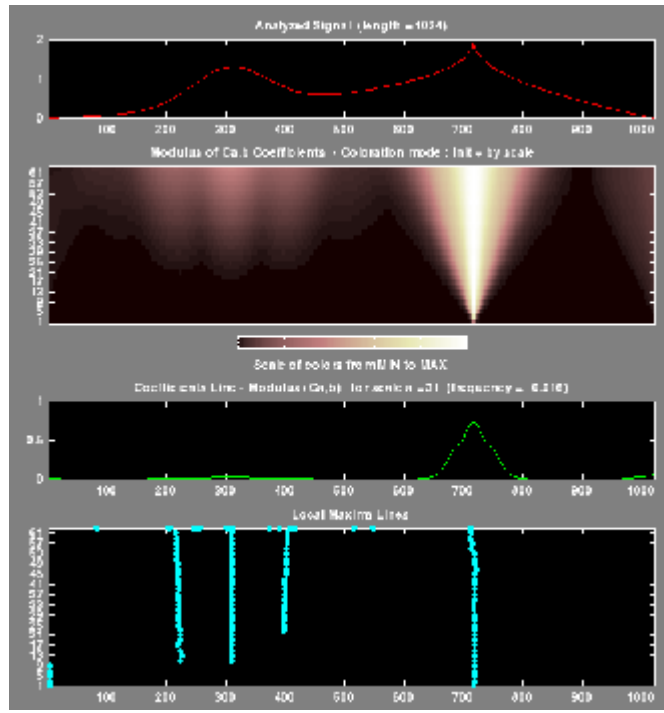
Click the **Analyze** button.

After a pause for computation, the tool displays the usual plots associated to the modulus of the coefficients on the left side, and the angle of the coefficients on the right side.



Each side has exactly the same representation that we found in “Continuous Analysis Using the Graphical Interface” on page 2-7.

Select the plots related to the modulus of the coefficients using the **Modulus** option button in the **Selected Axes** frame.



The figure now looks like the one in the real **Continuous Wavelet 1-D** tool.

Importing and Exporting Information from the Graphical Interface

To know how to import and export information from the Complex Continuous Wavelet Graphical Interface, see the corresponding paragraph in "One-Dimensional Continuous Wavelet Analysis" on page 2-4.

The only difference is that the variable `coefs` is a complex matrix (see "Saving Wavelet Coefficients" on page 2-17).

One-Dimensional Discrete Wavelet Analysis

This section takes you through the features of one-dimensional discrete wavelet analysis using the Wavelet Toolbox software.

The toolbox provides these functions for one-dimensional signal analysis. For more information, see the reference pages.

Analysis-Decomposition Functions

Function Name	Purpose
`dwt`	Single-level decomposition
`wavedec`	Decomposition
`wmaxlev`	Maximum wavelet decomposition level

Synthesis-Reconstruction Functions

Function Name	Purpose
`idwt`	Single-level reconstruction
`waverec`	Full reconstruction
`wrcoef`	Selective reconstruction
`upcoef`	Single reconstruction

Decomposition Structure Utilities

Function Name	Purpose
`detcoef`	Extraction of detail coefficients
`appcoef`	Extraction of approximation coefficients
`upwlev`	Recomposition of decomposition structure

De-noising and Compression

Function Name	Purpose
`ddencmp`	Provide default values for de-noising and compression
`wbmpen`	Penalized threshold for wavelet 1-D or 2-D de-noising
`wdcbm`	Thresholds for wavelet 1-D using Birgé-Massart strategy
`wdencmp`	Wavelet de-noising and compression
`wden`	Automatic wavelet de-noising
`wthrmngr`	Threshold settings manager

In this section, you'll learn how to

- Load a signal
- Perform a single-level wavelet decomposition of a signal
- Construct approximations and details from the coefficients
- Display the approximation and detail
- Regenerate a signal by inverse wavelet transform
- Perform a multilevel wavelet decomposition of a signal
- Extract approximation and detail coefficients
- Reconstruct the level 3 approximation
- Reconstruct the level 1, 2, and 3 details
- Display the results of a multilevel decomposition
- Reconstruct the original signal from the level 3 decomposition
- Remove noise from a signal
- Refine an analysis
- Compress a signal
- Show a signal's statistics and histograms

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

The final subsection discusses how to exchange signal and coefficient information between the disk and the graphical tools.

One-Dimensional Analysis Using the Command Line

This example involves a real-world signal — electrical consumption measured over the course of 3 days. This signal is particularly interesting because of noise introduced when a defect developed in the monitoring equipment as the measurements were being made. Wavelet analysis effectively removes the noise.

1 Load a signal.

From the MATLAB prompt, type

```
load leleccum;
```

Set the variables. Type

```
s = leleccum(1:3920);
l_s = length(s);
```

2 Perform a single-level wavelet decomposition of a signal.

Perform a single-level decomposition of the signal using the `db1` wavelet. Type

```
[cA1,cD1] = dwt(s,'db1');
```

This generates the coefficients of the level 1 approximation (`cA1`) and detail (`cD1`).

3 Construct approximations and details from the coefficients.

To construct the level 1 approximation and detail (`A1` and `D1`) from the coefficients `cA1` and `cD1`, type

```
A1 = upcoef('a',cA1,'db1',1,l_s);
D1 = upcoef('d',cD1,'db1',1,l_s);
```

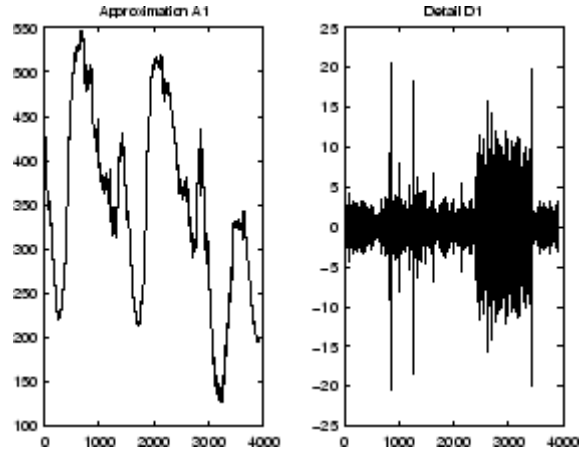
or

```
A1 = idwt(cA1,[],'db1',l_s);
D1 = idwt([],cD1,'db1',l_s);
```

4 Display the approximation and detail.

To display the results of the level-one decomposition, type

```
subplot(1,2,1); plot(A1); title('Approximation A1')
subplot(1,2,2); plot(D1); title('Detail D1')
```



5 Regenerate a signal by using the Inverse Wavelet Transform.

To find the inverse transform, type

```
A0 = idwt(cA1,cD1,'db1',l_s);
err = max(abs(s-A0))
err =
```

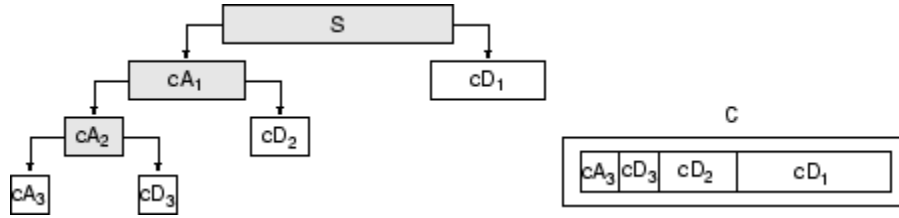
```
    2.2737e-013
```

6 Perform a multilevel wavelet decomposition of a signal.

To perform a level 3 decomposition of the signal (again using the `db1` wavelet), type

```
[C,L] = wavedec(s,3,'db1');
```

The coefficients of all the components of a third-level decomposition (that is, the third-level approximation and the first three levels of detail) are returned concatenated into one vector, C. Vector L gives the lengths of each component.



7 Extract approximation and detail coefficients.

To extract the level 3 approximation coefficients from `C`, type

```
cA3 = appcoef(C,L,'db1',3);
```

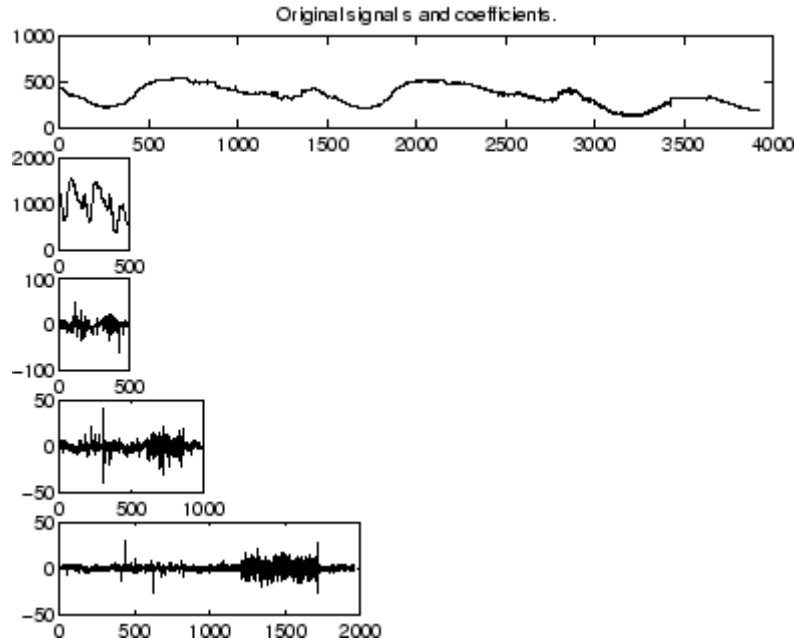
To extract the levels 3, 2, and 1 detail coefficients from `C`, type

```
cD3 = detcoef(C,L,3);
cD2 = detcoef(C,L,2);
cD1 = detcoef(C,L,1);
```

or

```
[cD1,cD2,cD3] = detcoef(C,L,[1,2,3]);
```

Results are displayed in the figure below, which contains the signal `s`, the approximation coefficients at level 3 (`cA3`), and the details coefficients from level 3 to 1 (`cD3`, `cD2` and `cD1`) from the top to the bottom.



8 Reconstruct the Level 3 approximation and the Level 1, 2, and 3 details.

To reconstruct the level 3 approximation from `C`, type

```
A3 = wrcoef('a',C,L,'db1',3);
```

To reconstruct the details at levels 1, 2, and 3, from `C`, type

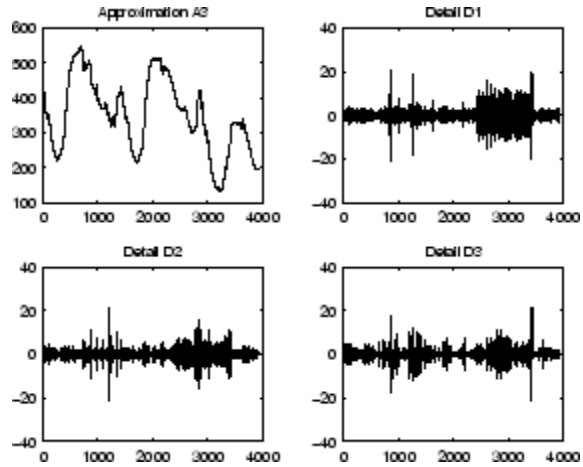
```
D1 = wrcoef('d',C,L,'db1',1);
D2 = wrcoef('d',C,L,'db1',2);
D3 = wrcoef('d',C,L,'db1',3);
```

9 Display the results of a multilevel decomposition.

To display the results of the level 3 decomposition, type

```
subplot(2,2,1); plot(A3);
title('Approximation A3')
subplot(2,2,2); plot(D1);
title('Detail D1')
```

```
subplot(2,2,3); plot(D2);
title('Detail D2')
subplot(2,2,4); plot(D3);
title('Detail D3')
```



10 Reconstruct the original signal from the Level 3 decomposition.

To reconstruct the original signal from the wavelet decomposition structure, type

```
A0 = waverec(C,L,'db1');

err = max(abs(s-A0))

err =
      4.5475e-013
```

11 Crude de-noising of a signal.

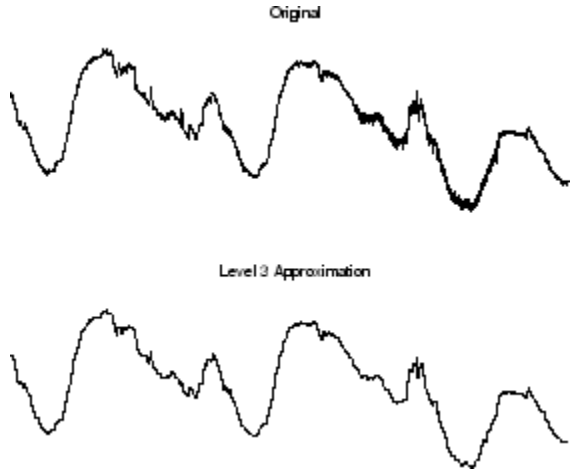
Using wavelets to remove noise from a signal requires identifying which component or components contain the noise, and then reconstructing the signal without those components.

In this example, we note that successive approximations become less and less noisy as more and more high-frequency information is filtered out of the signal.

The level 3 approximation, `A3`, is quite clean as a comparison between it and the original signal.

To compare the approximation to the original signal, type

```
subplot(2,1,1);plot(s);title('Original'); axis off
subplot(2,1,2);plot(A3);title('Level 3 Approximation');
axis off
```



Of course, in discarding all the high-frequency information, we've also lost many of the original signal's sharpest features.

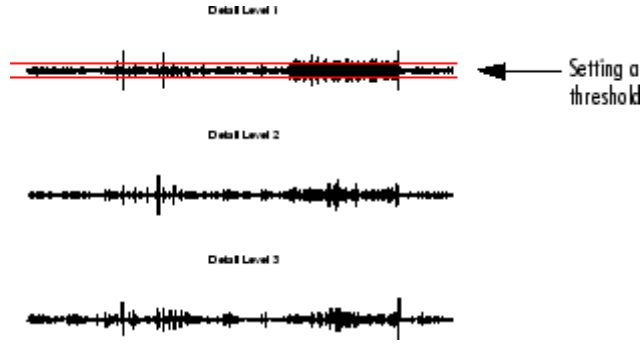
Optimal de-noising requires a more subtle approach called *thresholding*. This involves discarding only the portion of the details that exceeds a certain limit.

12 Remove noise by thresholding.

Let's look again at the details of our level 3 analysis.

To display the details `D1`, `D2`, and `D3`, type

```
subplot(3,1,1); plot(D1); title('Detail Level 1'); axis off
subplot(3,1,2); plot(D2); title('Detail Level 2'); axis off
subplot(3,1,3); plot(D3); title('Detail Level 3'); axis off
```



Most of the noise occurs in the latter part of the signal, where the details show their greatest activity. What if we limited the strength of the details by restricting their maximum values? This would have the effect of cutting back the noise while leaving the details unaffected through most of their durations. But there's a better way.

Note that `cD1`, `cD2`, and `cD3` are just MATLAB vectors, so we could directly manipulate each vector, setting each element to some fraction of the vectors' peak or average value. Then we could reconstruct new detail signals `D1`, `D2`, and `D3` from the thresholded coefficients.

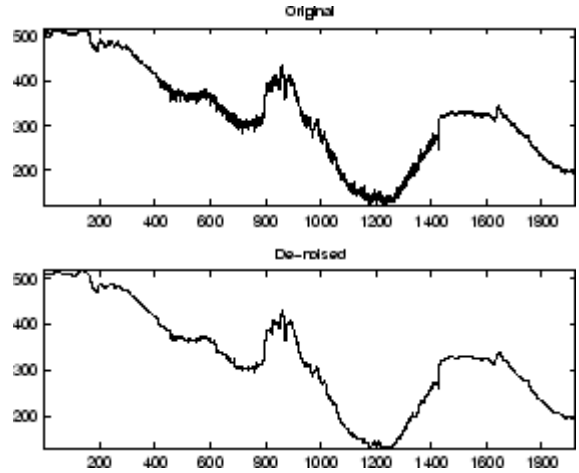
To denoise the signal, use the `ddencmp` command to calculate the default parameters and the `wdencmp` command to perform the actual de-noising, type

```
[thr,sorh,keepapp] = ddencmp('den','wv',s);
clean = wdencmp('gbl',C,L,'db1',3,thr,sorh,keepapp);
```

Note that `wdencmp` uses the results of the decomposition (`C` and `L`) that we calculated in step 6. We also specify that we used the `db1` wavelet to perform the original analysis, and we specify the global thresholding option `'gbl'`. See `ddencmp` and `wdencmp` in the reference pages for more information about the use of these commands.

To display both the original and denoised signals, type

```
subplot(2,1,1); plot(s(2000:3920)); title('Original')
subplot(2,1,2); plot(clean(2000:3920)); title('denoised')
```



We've plotted here only the noisy latter part of the signal. Notice how we've removed the noise without compromising the sharp detail of the original signal. This is a strength of wavelet analysis.

While using command line functions to remove the noise from a signal can be cumbersome, the software's graphical interface tools include an easy-to-use de-noising feature that includes automatic thresholding.

More information on the de-noising process can be found in the following sections:

- Remove noise from a signal
- "De-Noising" in the *Wavelet Toolbox User's Guide*
- "One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients" on page 2-145
- "One-Dimensional Variance Adaptive Thresholding of Wavelet Coeffiients" in the *Wavelet Toolbox User's Guide*

One-Dimensional Analysis Using the Graphical Interface

In this section, we explore the same electrical consumption signal as in the previous section, but we use the graphical interface tools to analyze the signal.

1 Start the 1-D Wavelet Analysis Tool.

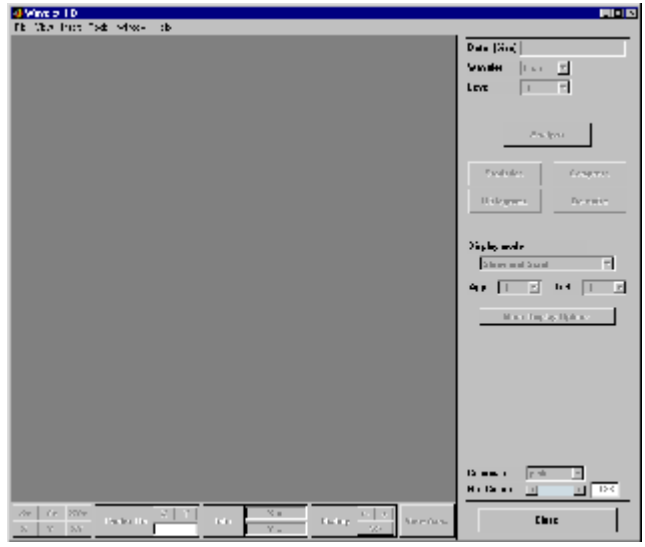
From the MATLAB prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.

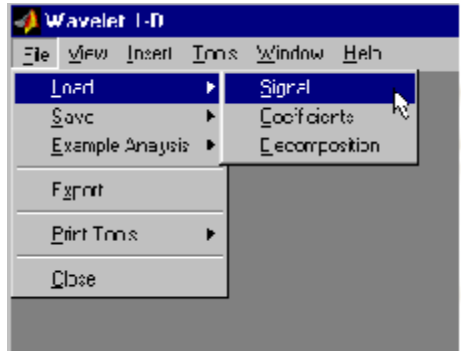
Click the **Wavelet 1-D** menu item.

The discrete wavelet analysis tool for one-dimensional signal data appears.

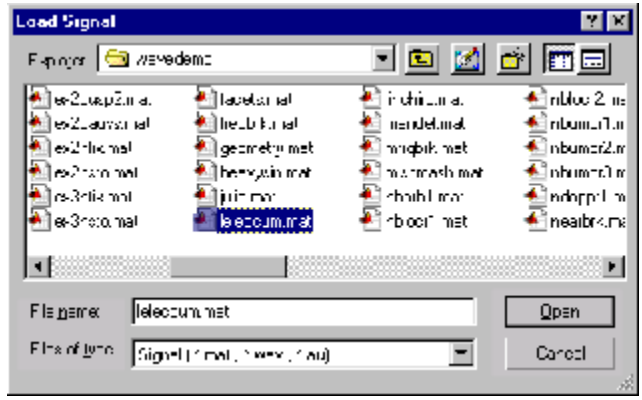


2 Load a signal.

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



When the **Load Signal** dialog box appears, select the demo MAT-file `leleccum.mat`, which is in the MATLAB folder `toolbox/wavelet/wavedemo`. Click the **OK** button.

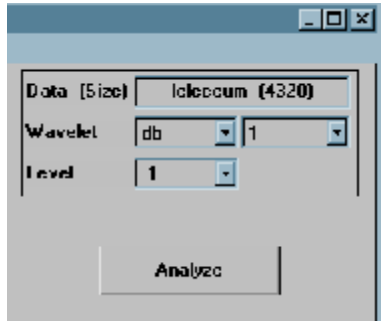


The electrical consumption signal is loaded into the **Wavelet 1-D** tool.

3 Perform a single-level wavelet decomposition.

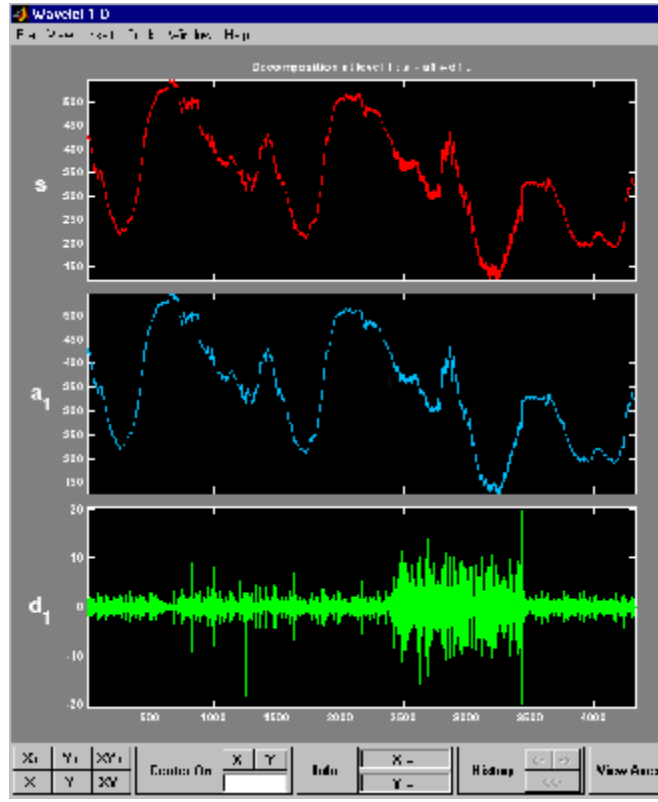
To start our analysis, let's perform a single-level decomposition using the `db1` wavelet, just as we did using the command-line functions in "One-Dimensional Analysis Using the Command Line" on page 2-30.

In the upper right portion of the **Wavelet 1-D** tool, select the `db1` wavelet and single-level decomposition.



Click the **Analyze** button.

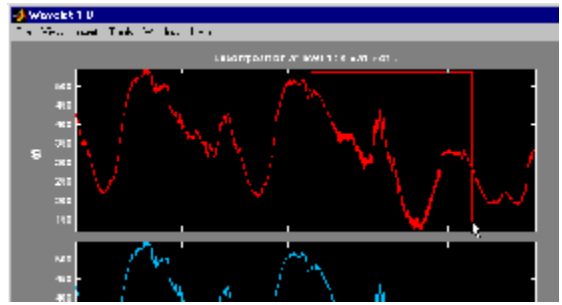
After a pause for computation, the tool displays the decomposition.



4 Zoom in on relevant detail.

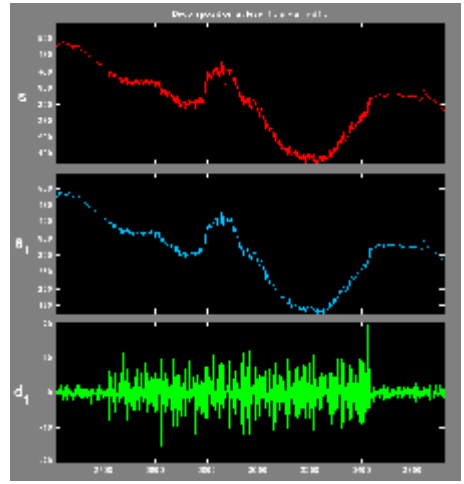
One advantage of using the graphical interface tools is that you can zoom in easily on any part of the signal and examine it in greater detail.

Drag a rubber band box (by holding down the left mouse button) over the portion of the signal you want to magnify. Here, we've selected the noisy part of the original signal.



Click the **X+** button (located at the bottom of the screen) to zoom horizontally.

The **Wavelet 1-D** tool zooms all the displayed signals.



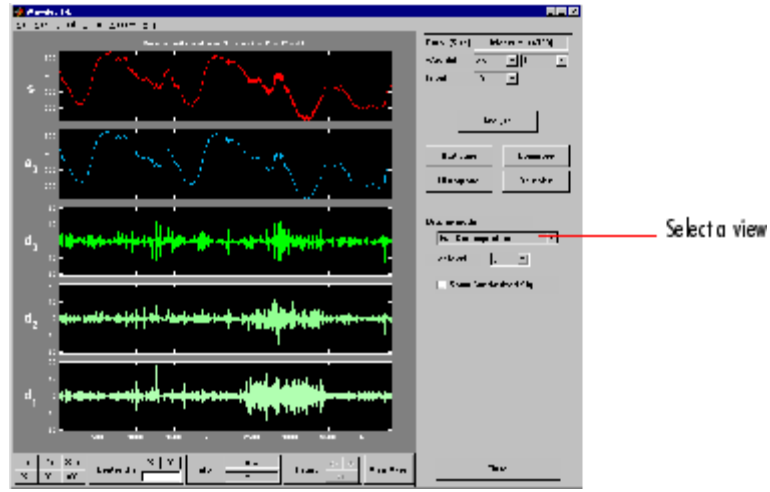
The other zoom controls do more or less what you'd expect them to. The **X-** button, for example, zooms out horizontally. The history function keeps track of all your views of the signal. Return to a previous zoom level by clicking the left arrow button.

5 Perform a multilevel decomposition.

Again, we'll use the graphical tools to emulate what we did in the previous section using command line functions. To perform a level 3 decomposition of the signal using the `db1` wavelet:

Select **3** from the **Level** menu at the upper right, and then click the **Analyze** button again.

After the decomposition is performed, you'll see a new analysis appear in the **Wavelet 1-D** tool.

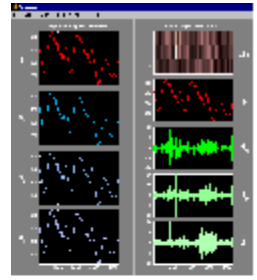


Selecting Different Views of the Decomposition

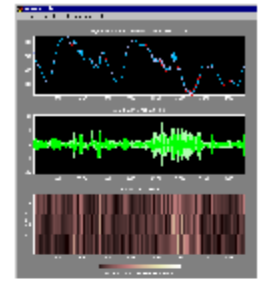
The **Display mode** menu (middle right) lets you choose different views of the wavelet decomposition.

The default display mode is called "Full Decomposition Mode." Other alternatives include:

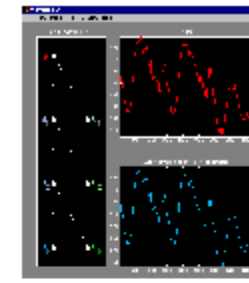
- "Separate Mode," which shows the details and the approximations in separate columns.
- "Superimpose Mode," which shows the details on a single plot superimposed in different colors. The approximations are plotted similarly.
- "Tree Mode," which shows the decomposition tree, the original signal, and one additional component of your choice. Click on the decomposition tree to select the signal component you'd like to view.
- "Show and Scroll Mode," which displays three windows. The first shows the original signal superimposed on an approximation you select. The second window shows a detail you select. The third window shows the wavelet coefficients.
- "Show and Scroll Mode (Stem Cfs)" is very similar to the "Show and Scroll Mode" except that it displays, in the third window, the wavelet coefficients as stem plots instead of colored blocks.



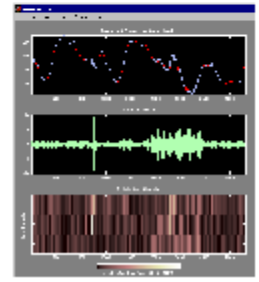
Separate Mode



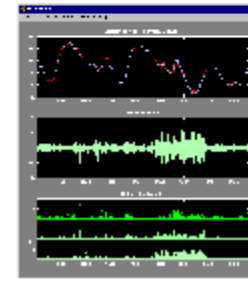
Superimpose Mode



Tree Mode



Show & Scroll Mode



Show & Scroll Mode (Stem Cfs)

You can change the default display mode on a per-session basis. Select the desired mode from the **View > Default Display Mode** submenu.

Note The **Compression** and **De-noising** windows opened from the **Wavelet 1-D** tool will inherit the current coefficient visualization attribute (stems or colored blocks).

Depending on which display mode you select, you may have access to additional display options through the **More Display Options** button (for more information, see "More Display Options" in the *Wavelet Toolbox User's Guide*).

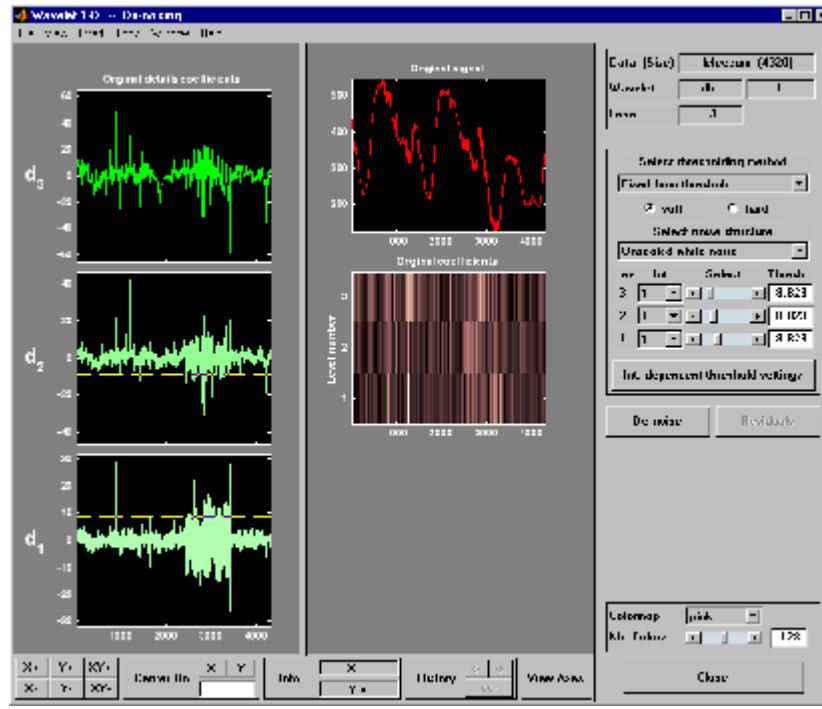
These options include the ability to suppress the display of various components, and to choose whether or not to display the original signal along with the details and approximations.

6 Remove noise from a signal.

The graphical interface tools feature a de-noising option with a predefined thresholding strategy. This makes it very easy to remove noise from a signal.

Bring up the de-noising tool: click the **denoise** button, located in the middle right of the window, underneath the **Analyze** button.

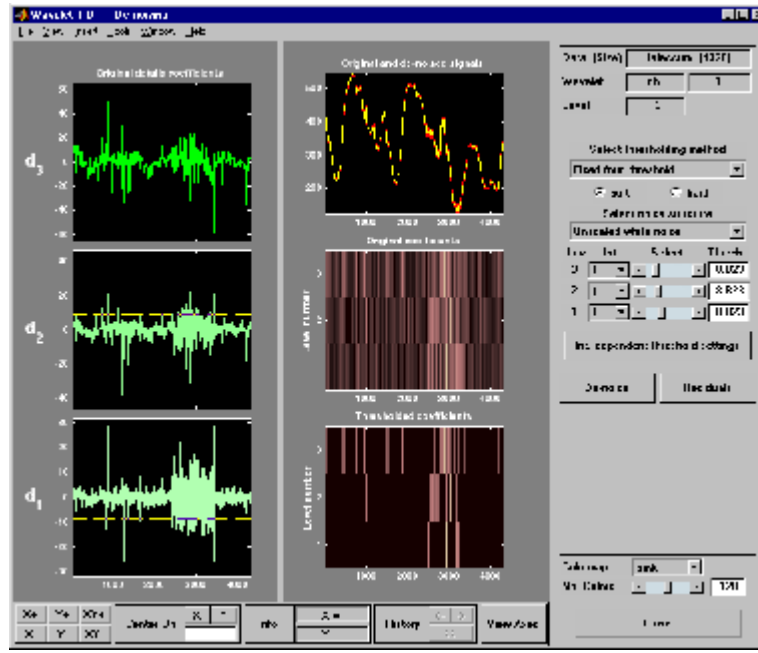
The **Wavelet 1-D De-noising** window appears.



While a number of options are available for fine-tuning the de-noising algorithm, we'll accept the defaults of soft fixed form thresholding and unscaled white noise.

Continue by clicking the **denoise** button.

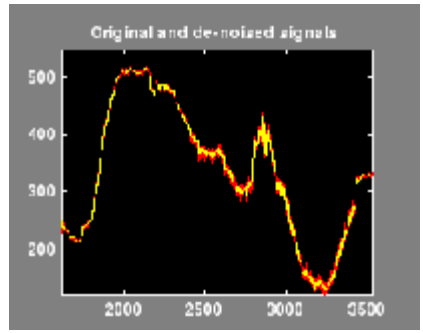
The denoised signal appears superimposed on the original. The tool also plots the wavelet coefficients of both signals.



Zoom in on the plot of the original and denoised signals for a closer look.

Drag a rubber band box around the pertinent area, and then click the **XY+** button.

The **denoise** window magnifies your view. By default, the original signal is shown in red, and the denoised signal in yellow.



Dismiss the **Wavelet 1-D De-noising** window: click the **Close** button.

You cannot have the **denoise** and **Compression** windows open simultaneously, so close the **Wavelet 1-D De-noising** window to continue. When the **Update Synthesized Signal** dialog box appears, click **No**. If you click **Yes**, the **Synthesized Signal** is then available in the **Wavelet 1-D** main window.

7 Refine the analysis.

The graphical tools make it easy to refine an analysis any time you want to. Up to now, we've looked at a level 3 analysis using `db1`. Let's refine our analysis of the electrical consumption signal using the `db3` wavelet at level 5.

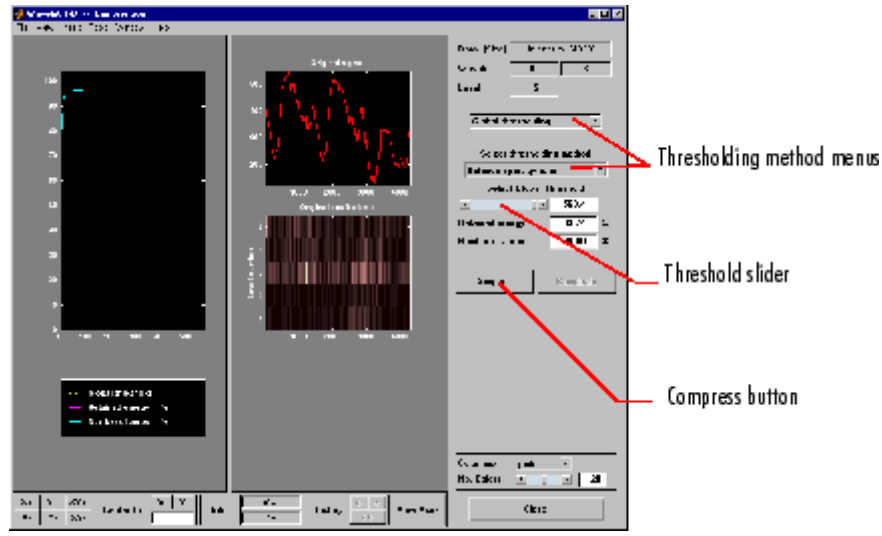
Select 5 from the **Level** menu at the upper right, and select the `db3` from the **Wavelet** menu. Click the **Analyze** button.

8 Compress the signal.

The graphical interface tools feature a compression option with automatic or manual thresholding.

Bring up the **Compression** window: click the **Compress** button, located in the middle right of the window, underneath the **Analyze** button.

The **Compression** window appears.

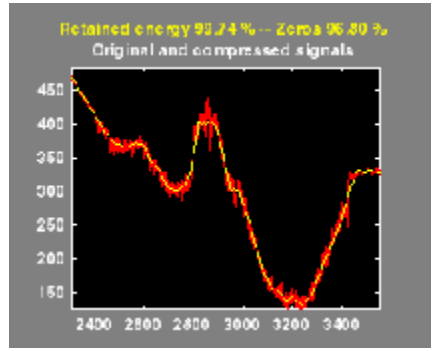


While you always have the option of choosing by level thresholding, here we'll take advantage of the global thresholding feature for quick and easy compression.

Note If you want to experiment with manual thresholding, choose the **By Level thresholding** option from the menu located at the top right of the **Wavelet 1-D Compression** window. The sliders located below this menu then control the level-dependent thresholds, indicated by yellow dotted lines running horizontally through the graphs on the left of the window. The yellow dotted lines can also be dragged directly using the left mouse button.

Click the **Compress** button, located at the center right.

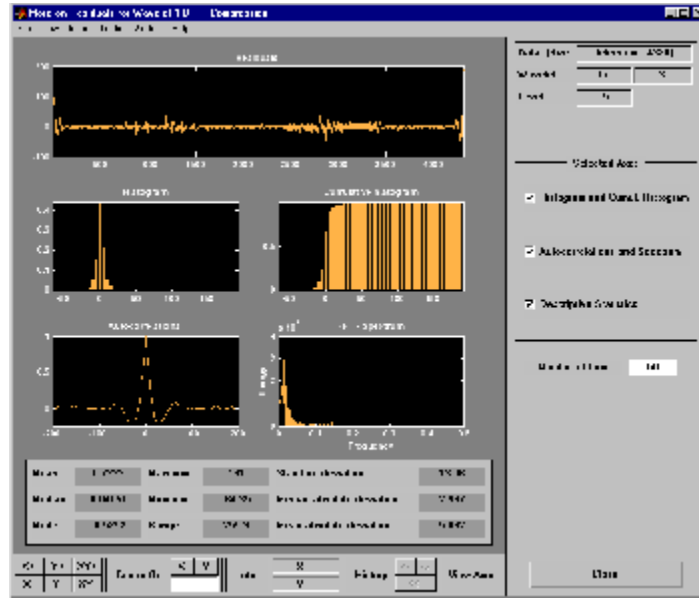
After a pause for computation, the electrical consumption signal is redisplayed in red with the compressed version superimposed in yellow. Below, we've zoomed in to get a closer look at the noisy part of the signal.



You can see that the compression process removed most of the noise, but preserved 99.74% of the energy of the signal. The automatic thresholding was very efficient, zeroing out all but 3.2% of the wavelet coefficients.

9 Show the residuals.

From the **Wavelet 1-D Compression** tool, click the **Residuals** button. The **More on Residuals for Wavelet 1-D Compression** window appears.



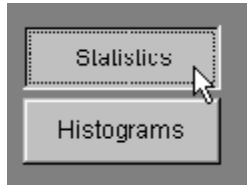
Displayed statistics include measures of tendency (mean, mode, median) and dispersion (range, standard deviation). In addition, the tool provides frequency-distribution diagrams (histograms and cumulative histograms), as well as time-series diagrams: autocorrelation function and spectrum. The same feature exists for the **Wavelet 1-D De-noising** tool.

Dismiss the **Wavelet 1-D Compression** window: click the **Close** button. When the **Update Synthesized Signal** dialog box appears, click **No.**

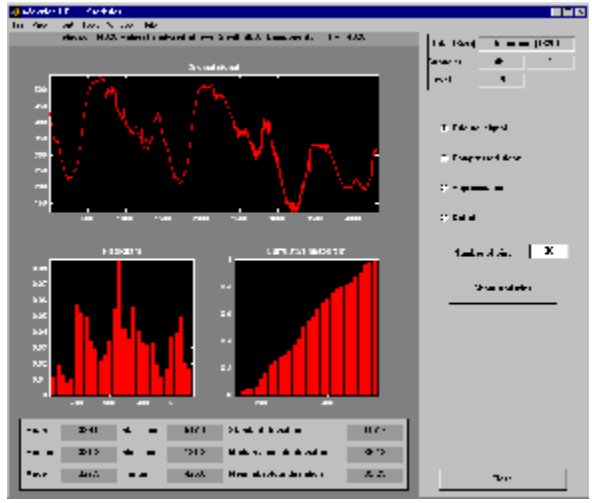
10 Show statistics.

You can view a variety of statistics about your signal and its components.

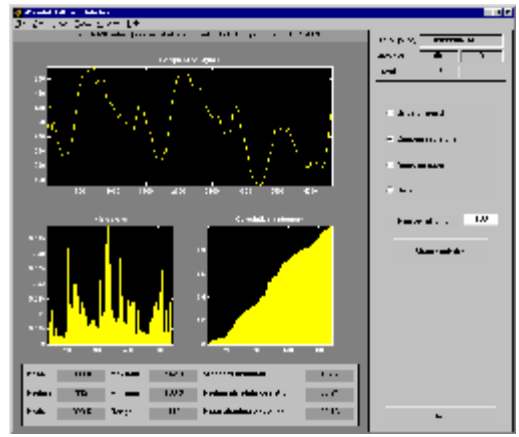
From the **Wavelet 1-D** tool, click the **Statistics** button.



The **Wavelet 1-D Statistics** window appears displaying by default statistics on the original signal.



Select the synthesized signal or signal component whose statistics you want to examine. Click the appropriate option button, and then click the **Show Statistics** button. Here, we've chosen to examine the compressed signal using more 100 bins instead of 30, which is the default:

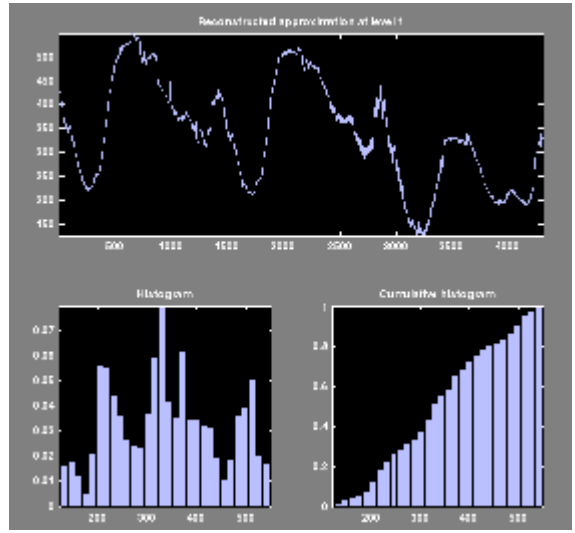


Displayed statistics include measures of tendency (mean, mode, median) and dispersion (range, standard deviation).

In addition, the tool provides frequency-distribution diagrams (histograms and cumulative histograms). You can plot these histograms separately using the **Histograms** button from the **Wavelets 1-D** window.

Click the **Approximation** option button. A menu appears from which you choose the level of the approximation you want to examine.

Select Level 1 and again click the **Show Statistics** button. Statistics appear for the level 1 approximation.

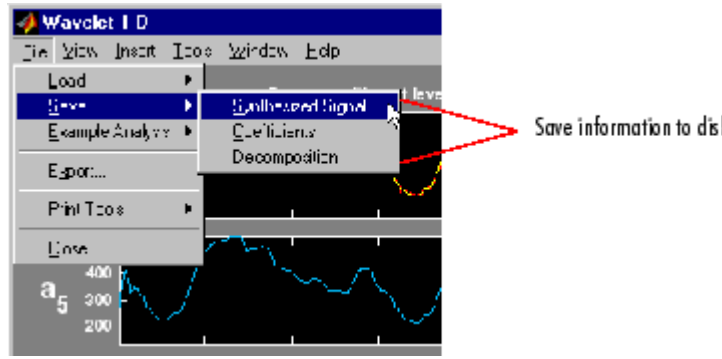


Importing and Exporting Information from the Graphical Interface

The **Wavelet 1-D** graphical interface tool lets you import information from and export information to disk.

Saving Information to Disk

You can save synthesized signals, coefficients, and decompositions from the **Wavelet 1-D** tool to the disk, where the information can be manipulated and later reimported into the graphical tool.



Saving Synthesized Signals. You can process a signal in the **Wavelet 1-D** tool and then save the processed signal to a MAT-file (with extension `mat` or other).

For example, load the example analysis: **File > Example Analysis > Basic Signals > with db3 at level 5 → Sum of sines**, and perform a compression or de-noising operation on the original signal. When you close the **De-noising** or **Compression** window, update the synthesized signal by clicking **Yes** in the dialog box.

Then, from the **Wavelet 1-D** tool, select the **File > Save > Synthesized Signal** menu option.

A dialog box appears allowing you to select a folder and filename for the MAT-file. For this example, choose the name `synthsig`.

To load the signal into your workspace, simply type

```
load synthsig
```

When the synthesized signal is obtained using any thresholding method except a global one, the saved structure is

```
whos
```

Name	Size	Bytes	Class
synthsig	1x1000	8000	double array
thrParams	1x5	580	cell array
wname	1x3	6	char array

The synthesized signal is given by the variable `synthsig`. In addition, the parameters of the de-noising or compression process are given by the wavelet name (`wname`) and the level dependent thresholds contained in the `thrParams` variable, which is a cell array of length 5 (same as the level of the decomposition).

For `i` from 1 to 5, `thrParams{i}` contains the lower and upper bounds of the thresholding interval and the threshold value (since interval dependent thresholds are allowed, see "One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients" on page 2-145).

For example, for level 1,

```
thrParams{1}

ans =
    1.0e+03 *
    0.0010    1.0000    0.0014
```

When the synthesized signal is obtained using a global thresholding method, the saved structure is

Name	Size	Bytes	Class
synthsig	1x1000	8000	double array
valTHR	1x1	8	double array
wname	1x3	6	char array

where the variable `valTHR` contains the global threshold:

```
valTHR
```

```
valTHR =
    1.2922
```

Saving Discrete Wavelet Transform Coefficients. The **Wavelet 1-D** tool lets you save the coefficients of a discrete wavelet transform (DWT) to disk. The toolbox creates a MAT-file in the current folder with a name you choose.

To save the DWT coefficients from the present analysis, use the menu option **File > Save > Coefficients**.

A dialog box appears that lets you specify a folder and filename for storing the coefficients.

Consider the example analysis:

File > Example Analysis > Basic Signals > with db1 at level 5 → Cantor curve.

After saving the wavelet coefficients to the file `cantor.mat`, load the variables into your workspace:

```
load cantor
whos
```

Name	Size	Bytes	Class
coefs	1x2190	17520	double array
longs	1x7	56	double array
thrParams	0x0	0	double array
wname	1x3	6	char array

Variable `coefs` contains the discrete wavelet coefficients. More precisely, in the above example `coefs` is a 1-by-2190 vector of concatenated coefficients, and `longs` is a vector giving the lengths of each component of `coefs`.

Variable `wname` contains the wavelet name and `thrParams` is empty since the synthesized signal does not exist.

Saving Decompositions. The **Wavelet 1-D** tool lets you save the entire set of data from a discrete wavelet analysis to disk. The toolbox creates a MAT-file in the current folder with a name you choose, followed by the extension `wa1` (wavelet analysis 1-D).

Open the **Wavelet 1-D** tool and load the example analysis:

File > Example Analysis > Basic Signals > with db3 at level 5 → Sum of sines

To save the data from this analysis, use the menu option **File > Save > Decomposition**.

A dialog box appears that lets you specify a folder and filename for storing the decomposition data. Type the name `wdecex1d`.

After saving the decomposition data to the file `wdecex1d.wa1`, load the variables into your workspace:

```
load wdecex1d.wa1 -mat
whos
```

Name	Size	Bytes	Class
coefs	1x1023	8184	double array
data_name	1x6	12	char array
longs	1x7	56	double array
thrParams	0x0	0	double array
wave_name	1x3	6	char array

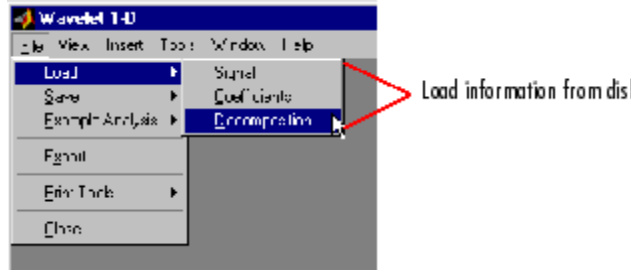
Note Save options are also available when performing de-noising or compression inside the **Wavelet 1-D** tool. In the **Wavelet 1-D De-noising** window, you can save denoised signal and decomposition. The same holds true for the **Wavelet 1-D Compression** window. This way, you can save many different trials from inside the De-noising and Compression windows without going back to the main **Wavelet 1-D** window during a fine-tuning process.

Note When saving a synthesized signal, a decomposition or coefficients to a MAT-file, the `mat` file extension is not necessary. You can save approximations individually for each level or save them all at once.

Loading Information into the Wavelet 1-D Tool

You can load signals, coefficients, or decompositions into the graphical interface. The information you load may have been previously exported from the graphical interface, and then manipulated in the workspace, or it may have been information you generated initially from the command line.

In either case, you must observe the strict file formats and data structures used by the **Wavelet 1-D** tool, or else errors will result when you try to load information.



Loading Signals. To load a signal you've constructed in your MATLAB workspace into the **Wavelet 1-D** tool, save the signal in a MAT-file (with extension `mat` or other).

For instance, suppose you've designed a signal called `warma` and want to analyze it in the **Wavelet 1-D** tool.

```
save warma warma
```

The workspace variable warma must be a vector.

```
sizwarma = size(warma)
sizwarma =
           1        1000
```

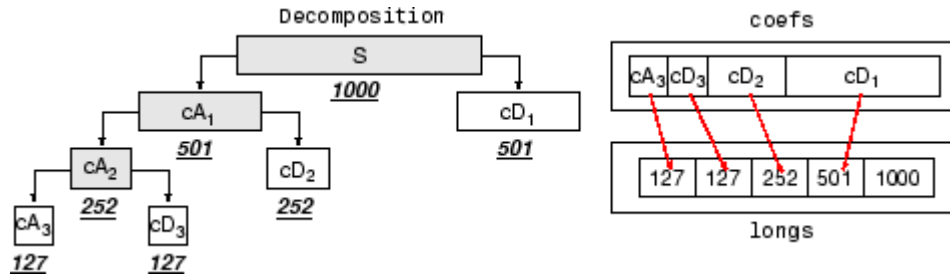
To load this signal into the **Wavelet 1-D** tool, use the menu option **File > Load > Signal**.

A dialog box appears that lets you select the appropriate MAT-file to be loaded.

Note The first one-dimensional variable encountered in the file is considered the signal. Variables are inspected in alphabetical order.

Loading Discrete Wavelet Transform Coefficients. To load discrete wavelet transform coefficients into the **Wavelet 1-D** tool, you must first save the appropriate data in a MAT-file, which must contain at least the two variables `coefs` and `longs`.

Variable `coefs` must be a vector of DWT coefficients (concatenated for the various levels), and variable `longs` a vector specifying the length of each component of `coefs`, as well as the length of the original signal.



After constructing or editing the appropriate data in your workspace, type

```
save myfile coefs longs
```

Use the **File > Load > Coefficients** menu option from the **Wavelet 1-D** tool to load the data into the graphical tool.

A dialog box appears, allowing you to choose the folder and file in which your data reside.

Loading Decompositions. To load discrete wavelet transform decomposition data into the **Wavelet 1-D** graphical interface, you must first save the appropriate data in a MAT-file (with extension `wa1` or other).

The MAT-file contains the following variables.

Variable	Status	Description
`coefs`	Required	Vector of concatenated DWT coefficients
`longs`	Required	Vector specifying lengths of components of `coefs` and of the original signal
`wave_name`	Required	String specifying name of wavelet used for decomposition (e.g., `db3`)
`data_name`	Optional	String specifying name of decomposition

After constructing or editing the appropriate data in your workspace, type

```
save myfile coefs longs wave_name
```

Use the **File > Load > Decomposition** menu option from the **Wavelet 1-D** tool to load the decomposition data into the graphical tool.

A dialog box appears, allowing you to choose the folder and file in which your data reside.

Note When loading a signal, a decomposition or coefficients from a MAT-file, the extension of this file is free. The `mat` extension is not necessary.

Two-Dimensional Discrete Wavelet Analysis

This section takes you through the features of two-dimensional discrete wavelet analysis using the Wavelet Toolbox software. The toolbox provides these functions for image analysis. For more information, see the function reference pages.

Note In this section the presentation and examples use two-dimensional arrays corresponding to indexed image representations. However, the functions described are also available when using truecolor images, which are represented by `m`-by-`n`-by-3 arrays of `uint8`. For more information on image formats, see "Wavelets: Working with Images" on page 2-89.

Analysis-Decomposition Functions

Function Name	Purpose
`dwt2`	Single-level decomposition
`wavedec2`	Decomposition
`wmaxlev`	Maximum wavelet decomposition level

Synthesis-Reconstruction Functions

Function Name	Purpose
`idwt2`	Single-level reconstruction
`waverec2`	Full reconstruction
`wrcoef2`	Selective reconstruction
`upcoef2`	Single reconstruction

Decomposition Structure Utilities

Function Name	Purpose
`detcoef2`	Extraction of detail coefficients
`appcoef2`	Extraction of approximation coefficients
`upwlev2`	Recomposition of decomposition structure

De-Noising and Compression

Function Name	Purpose
`ddencmp`	Provide default values for de-noising and compression
`wbmpen`	Penalized threshold for wavelet 1-D or 2-D de-noising
`wdcbm2`	Thresholds for wavelet 2-D using Birgé-Massart strategy
`wdencmp`	Wavelet de-noising and compression
`wthrmngr`	Threshold settings manager

In this section, you'll learn

- How to load an image
- How to analyze an image
- How to perform single-level and multilevel image decompositions and reconstructions (command line only)
- How to use Square and Tree mode features (GUI only)
- How to zoom in on detail (GUI only)
- How to compress an image

Two-Dimensional Analysis Using the Command Line

In this example we'll show how you can use two-dimensional wavelet analysis to compress an image efficiently without sacrificing its clarity.

Note Instead of directly using `image(I)` to visualize the image `I`, we use `image(wcodemat(I))`, which displays a rescaled version of `I` leading to a clearer presentation of the details and approximations (see `wcodemat` reference page).

1 Load an image.

From the MATLAB prompt, type

```
load wbarb;
whos
```

Name	Size	Bytes	Class
X	256x256	524288	double array
map	192x3	4608	double array

2 Display the image. Type

```
image(X); colormap(map); colorbar;
```

3 Convert an indexed image to a grayscale image.

If the colormap is smooth, the wavelet transform can be directly applied to the indexed image; otherwise the indexed image should be converted to

grayscale format. For more information, see "Wavelets: Working with Images" on page 2-89.

Since the colormap is smooth in this image, you can now perform the decomposition.

4 Perform a single-level wavelet decomposition.

To perform a single-level decomposition of the image using the `bior3.7` wavelet, type

```
[cA1,cH1,cV1,cD1] = dwt2(X,'bior3.7');
```

This generates the coefficient matrices of the level-one approximation (`cA1`) and horizontal, vertical and diagonal details (`cH1`,`cV1`,`cD1`, respectively).

5 Construct and display approximations and details from the coefficients.

To construct the level-one approximation and details (A_1, H_1, V_1, and D_1) from the coefficients cA_1, cH_1, cV_1, and cD_1, type

```
A1 = upcoef2('a',cA1,'bior3.7',1);
H1 = upcoef2('h',cH1,'bior3.7',1);
V1 = upcoef2('v',cV1,'bior3.7',1);
D1 = upcoef2('d',cD1,'bior3.7',1);
```

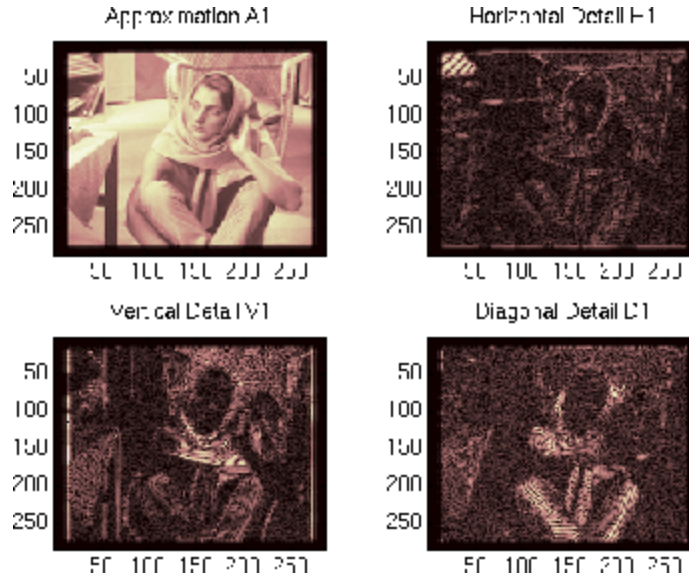
or

```
sx = size(X);
A1 = idwt2(cA1,[],[],[],'bior3.7',sx);
H1 = idwt2([],cH1,[],[],'bior3.7',sx);
V1 = idwt2([],[],cV1,[],'bior3.7',sx);
D1 = idwt2([],[],[],cD1,'bior3.7',sx);
```

To display the results of the level 1 decomposition, type

```
colormap(map);
subplot(2,2,1); image(wcodemat(A1,192));
title('Approximation A1')
subplot(2,2,2); image(wcodemat(H1,192));
title('Horizontal Detail H1')
subplot(2,2,3); image(wcodemat(V1,192));
```

```
title('Vertical Detail V1')
subplot(2,2,4); image(wcodemat(D1,192));
title('Diagonal Detail D1')
```



6 Regenerate an image by single-level Inverse Wavelet Transform.

To find the inverse transform, type

```
Xsyn = idwt2(cA1,cH1,cV1,cD1,'bior3.7');
```

This reconstructs or synthesizes the original image from the coefficients of the level 1 approximation and details.

7 Perform a multilevel wavelet decomposition.

To perform a level 2 decomposition of the image (again using the `bior3.7` wavelet), type

```
[C,S] = wavedec2(X,2,'bior3.7');
```

where `X` is the original image matrix, and 2 is the level of decomposition.

The coefficients of all the components of a second-level decomposition (that is, the second-level approximation and the first two levels of detail) are returned concatenated into one vector, `C`. Argument `S` is a bookkeeping matrix that keeps track of the sizes of each component.

8 Extract approximation and detail coefficients.

To extract the level 2 approximation coefficients from `C`, type

```
cA2 = appcoef2(C,S,'bior3.7',2);
```

To extract the first- and second-level detail coefficients from `C`, type

```
cH2 = detcoef2('h',C,S,2);
cV2 = detcoef2('v',C,S,2);
cD2 = detcoef2('d',C,S,2);
cH1 = detcoef2('h',C,S,1);
cV1 = detcoef2('v',C,S,1);
cD1 = detcoef2('d',C,S,1);
```

or

```
[cH2,cV2,cD2] = detcoef2('all',C,S,2);
[cH1,cV1,cD1] = detcoef2('all',C,S,1);
```

where the first argument (`'h'`, `'v'`, or `'d'`) determines the type of detail (horizontal, vertical, diagonal) extracted, and the last argument determines the level.

9 Reconstruct the Level 2 approximation and the Level 1 and 2 details.

To reconstruct the level 2 approximation from `C`, type

```
A2 = wrcoef2('a',C,S,'bior3.7',2);
```

To reconstruct the level 1 and 2 details from `C`, type

```
H1 = wrcoef2('h',C,S,'bior3.7',1);
V1 = wrcoef2('v',C,S,'bior3.7',1);
D1 = wrcoef2('d',C,S,'bior3.7',1);
H2 = wrcoef2('h',C,S,'bior3.7',2);
V2 = wrcoef2('v',C,S,'bior3.7',2);
```

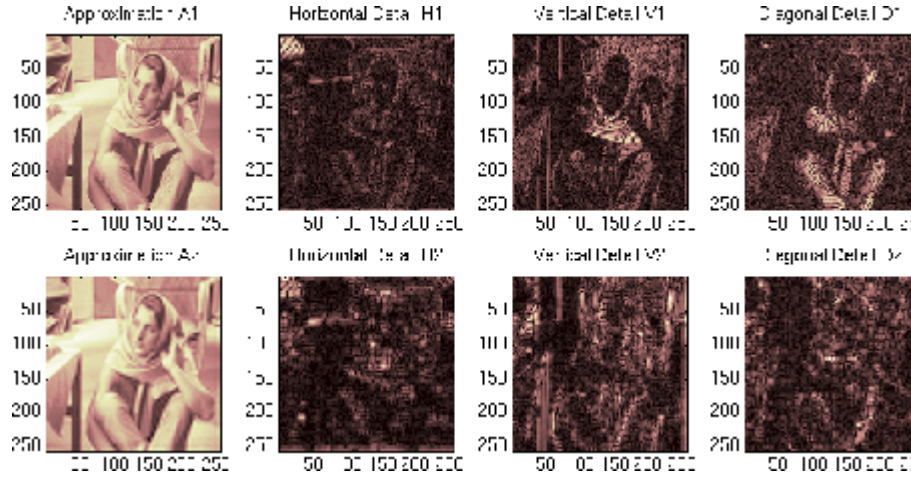
```
D2 = wrcoef2('d',C,S,'bior3.7',2);
```

10 Display the results of a multilevel decomposition.

Note With all the details involved in a multilevel image decomposition, it makes sense to import the decomposition into the **Wavelet 2-D** graphical tool in order to more easily display it. For information on how to do this, see "Loading Decompositions" on page 2-87.

To display the results of the level 2 decomposition, type

```
colormap(map);
subplot(2,4,1);image(wcodemat(A1,192));
title('Approximation A1')
subplot(2,4,2);image(wcodemat(H1,192));
title('Horizontal Detail H1')
subplot(2,4,3);image(wcodemat(V1,192));
title('Vertical Detail V1')
subplot(2,4,4);image(wcodemat(D1,192));
title('Diagonal Detail D1')
subplot(2,4,5);image(wcodemat(A2,192));
title('Approximation A2')
subplot(2,4,6);image(wcodemat(H2,192));
title('Horizontal Detail H2')
subplot(2,4,7);image(wcodemat(V2,192));
title('Vertical Detail V2')
subplot(2,4,8);image(wcodemat(D2,192));
title('Diagonal Detail D2')
```



11 Reconstruct the original image from the multilevel decomposition.

To reconstruct the original image from the wavelet decomposition structure, type

```
X0 = waverec2(C,S,'bior3.7');
```

This reconstructs or synthesizes the original image from the coefficients `C` of the multilevel decomposition.

12 Compress the image and display it.

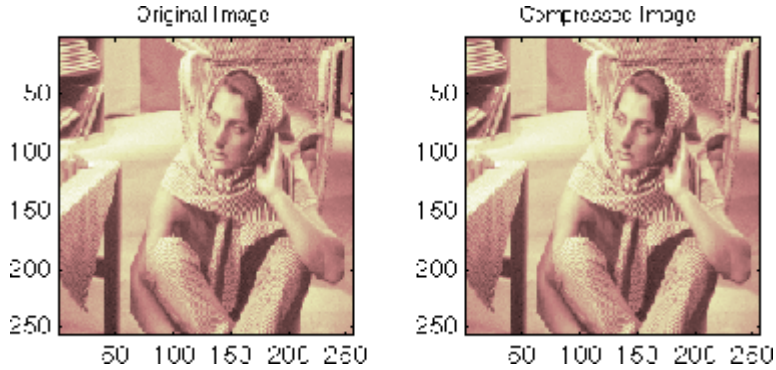
To compress the original image `X`, use the `ddencmp` command to calculate the default parameters and the `wdencmp` command to perform the actual compression. Type

```
[thr,sorh,keepapp]= ddencmp('cmp','wv',X);
[Xcomp,CXC,LXC,PERF0,PERFL2] = ...
wdencmp('gbl',C,S,'bior3.7',2,thr,sorh,keepapp);
```

Note that we pass in to `wdencmp` the results of the decomposition (`C` and `S`) we calculated in step 7. We also specify the `bior3.7` wavelets, because we used this wavelet to perform the original analysis. Finally, we specify the global thresholding option `'gbl'`. See `ddencmp` and `wdencmp` reference pages for more information about the use of these commands.

To view the compressed image side by side with the original, type

```
colormap(map);
subplot(121); image(X); title('Original Image');
axis square
subplot(122); image(Xcomp); title('Compressed Image');
axis square
```




```
PERF0
   PERF0 =
       49.8076

PERFL2
   PERFL2 =
       99.9817
```

These returned values tell, respectively, what percentage of the wavelet coefficients was set to zero and what percentage of the image's energy was preserved in the compression process.

Note that, even though the compressed image is constructed from only about half as many nonzero wavelet coefficients as the original, there is almost no detectable deterioration in the image quality.

Two-Dimensional Analysis Using the Graphical Interface

In this section we explore the same image as in the previous section, but we use the graphical interface tools to analyze the image.

1 Start the 2-D Wavelet Analysis Tool.

From the MATLAB prompt, type

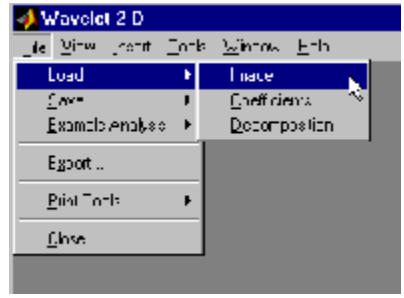
```
wavemenu
```

The **Wavelet Tool Main Menu** appears.

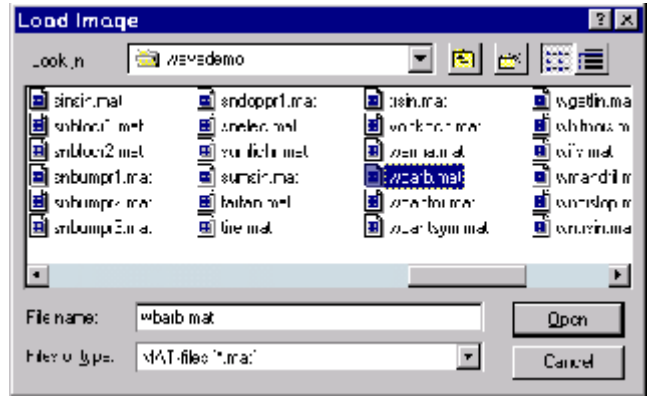
Click the **Wavelet 2-D** menu item. The discrete wavelet analysis tool for two-dimensional image data appears.

2 Load an image.

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



When the **Load Image** dialog box appears, select the demo MAT-file `wbarb.mat`, which is in the MATLAB folder `toolbox/wavelet/wavedemo`. Click the **OK** button.

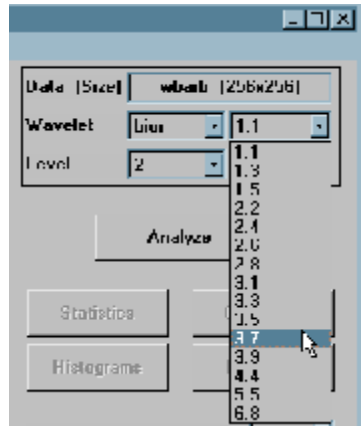


The image is loaded into the **Wavelet 2-D** tool.

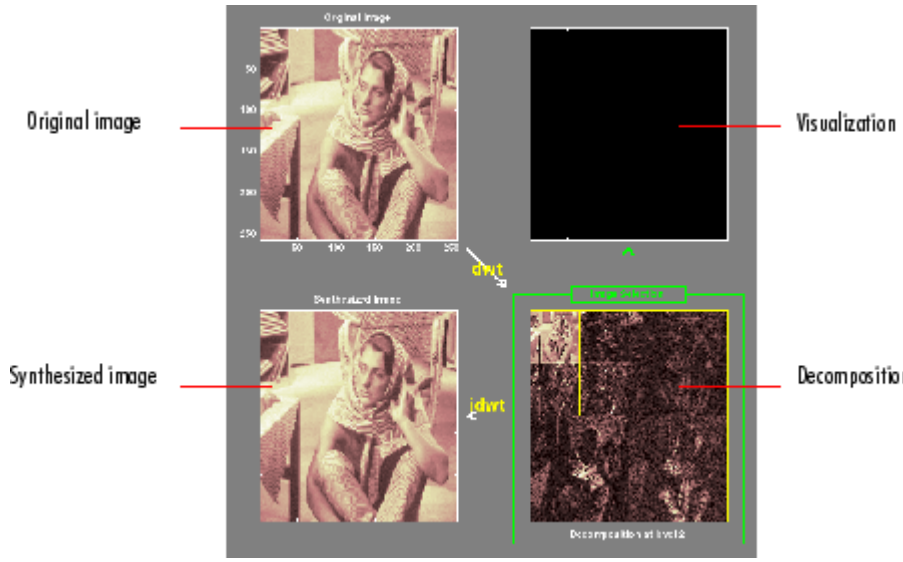
3 Analyze the image.

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 `bior3.7` wavelet at level 2.



Click the **Analyze** button. After a pause for computation, the **Wavelet 2-D** tool displays its analysis.

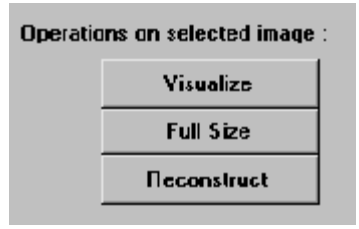


Using Square Mode Features

By default, the analysis appears in "Square Mode." This mode includes four different displays. In the upper left is the original image. Below that is the image reconstructed from the various approximations and details. To the lower right is a decomposition showing the coarsest approximation coefficients and all the horizontal, diagonal, and vertical detail coefficients. Finally, the visualization space at the top right displays any component of the analysis that you want to look at more closely.

Click on any decomposition component in the lower right window.

A green border highlights the selected component. At the lower right of the **Wavelet 2-D** window, there is a set of three buttons labeled "Operations on selected image." Note that if you click again on the same component, you'll deselect it and the green border disappears.



Click the **Visualize** button.

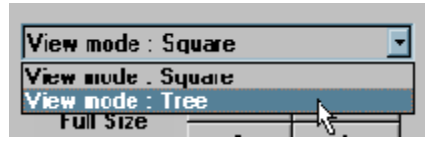
The selected image is displayed in the visualization area. You are seeing the raw, unreconstructed two-dimensional wavelet coefficients. Using the other buttons, you can display the reconstructed version of the selected image component, or you can view the selected component at full screen resolution.

Approximation coefficients cA2

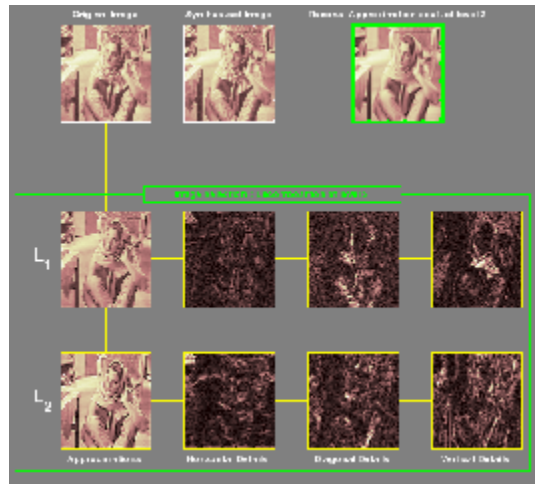
Reconstructed Approximation A2

Using Tree Mode Features

Choose **Tree** from the **View Mode** menu.



Your display changes to reveal the following.



This is the same information shown in square mode, with in addition all the approximation coefficients, but arranged to emphasize the tree structure of the decomposition. The various buttons and menus work just the same as they do in square mode.

Zooming in on Detail

Drag a rubber band box (by holding down the left mouse button) over the portion of the image you want to magnify.

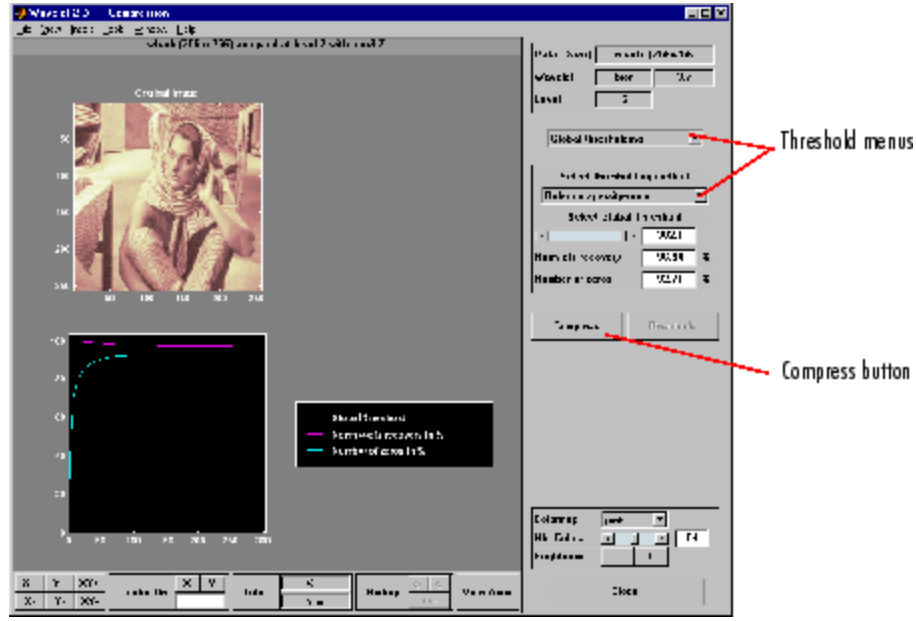
Click the **XY+** button (located at the bottom of the screen) to zoom horizontally and vertically.

The **Wavelet 2-D** tool enlarges the displayed images.

To zoom back to original magnification, click the **History** <<- button.

4 Compress the image

Click the **Compress** button, located to the upper right of the **Wavelet 2-D** window. The **Wavelet 2-D Compression** window appears.

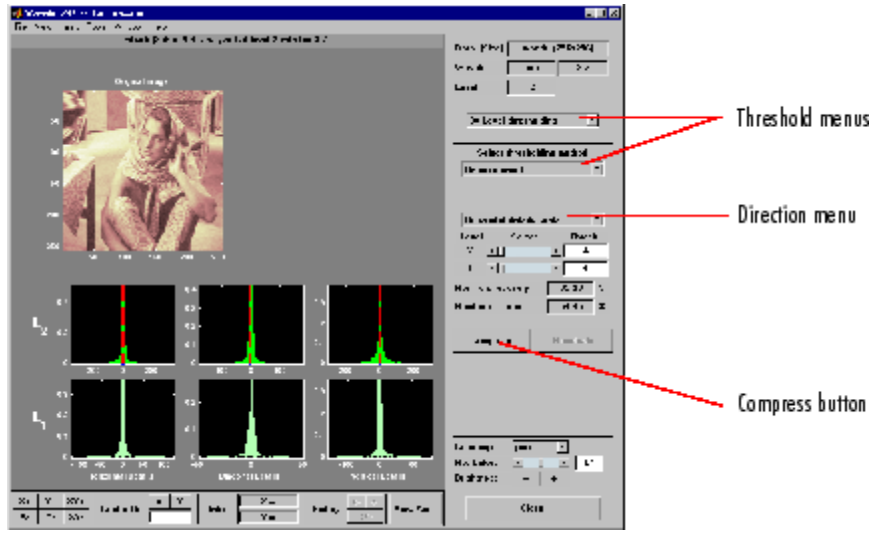


The tool automatically selects thresholding levels to provide a good initial balance between retaining the image's energy while minimizing the number of coefficients needed to represent the image.

However, you can also adjust thresholds manually using the **By Level thresholding** option, and then the sliders or edits corresponding to each level.

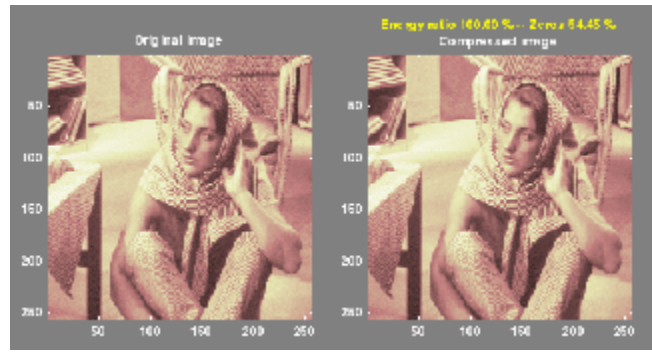
For this example, select the **By Level thresholding** option and select the **Remove near 0** method from the **Select thresholding method** menu.

The following window is displayed.



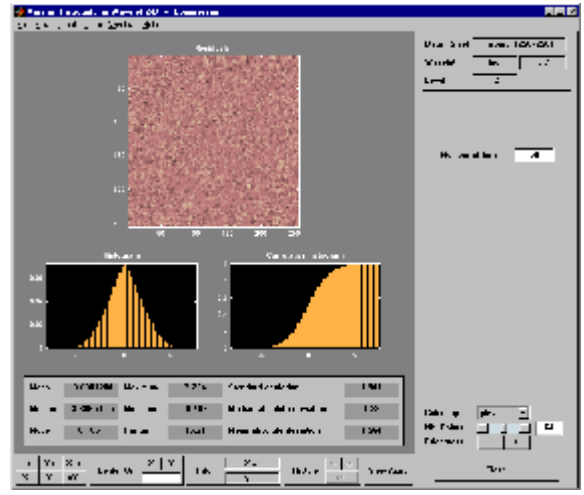
Select from the direction menu whether you want to adjust thresholds for horizontal, diagonal or vertical details. To make the actual adjustments for each level, use the sliders or use the left mouse button to directly drag the yellow vertical lines.

To compress the original image, click the **Compress** button. After a pause for computation, the compressed image is displayed beside the original. Notice that compression eliminates almost half the coefficients, yet no detectable deterioration of the image appears.



5 Show the residuals.

From the **Wavelet 2-D Compression** tool, click the **Residuals** button. The **More on Residuals for Wavelet 2-D Compression** window appears.



Displayed statistics include measures of tendency (mean, mode, median) and dispersion (range, standard deviation). In addition, the tool provides frequency-distribution diagrams (histograms and cumulative histograms). The same tool exists for the **Wavelet 2-D De-noising** tool.

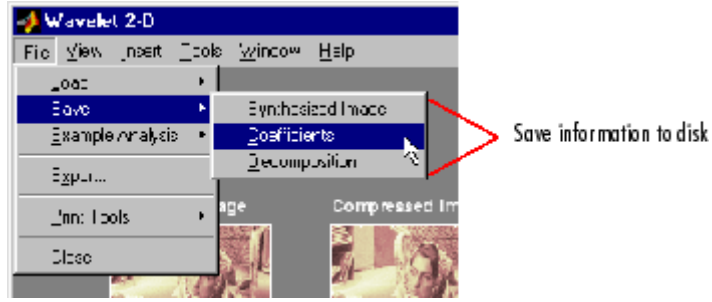
Note The statistics displayed in the above figure are related to the displayed image but not to the original one. Usually this information is the same, but in some cases, edge effects may cause the original image to be cropped slightly. To see the exact statistics, use the command line functions to get the desired image and then apply the desired MATLAB statistical function(s).

Importing and Exporting Information from the Graphical Interface

The **Wavelet 2-D** graphical tool lets you import information from and export information to disk, if you adhere to the proper file formats.

Saving Information to Disk

You can save synthesized images, coefficients, and decompositions from the **Wavelet 2-D** tool to disk, where the information can be manipulated and later reimported into the graphical tool.



Saving Synthesized Images. You can process an image in the **Wavelet 2-D** tool, and then save the processed image to a MAT-file (with extension `mat` or other).

For example, load the example analysis:

File > Example Analysis > at level 3, with sym4 → detail Durer

and perform a compression on the original image. When you close the **Wavelet 2-D Compression** window, update the synthesized image by clicking **Yes** in the dialog box that appears.

Then, from the **Wavelet 2-D** tool, select the **File > Save > Synthesized Image** menu option. A dialog box appears allowing you to select a folder and filename for the MAT-file (with extension `mat` or other). For this example, choose the name `symage`.

To load the image into your workspace, type

```
load symage
whos
```

Name	Size	Bytes	Class
`X`	`359x371`	`1065512`	`double array`
`map`	`64x3`	`1536`	`double array`
`valTHR`	`1x1`	`8`	`double array`
`wname`	`1x4`	`8`	`char array`

The synthesized image is given by `X` and `map` contains the colormap. In addition, the parameters of the de-noising or compression process are given by the wavelet name (`wname`) and the global threshold (`valTHR`).

Saving Discrete Wavelet Transform Coefficients. The **Wavelet 2-D** tool lets you save the coefficients of a discrete wavelet transform (DWT) to disk. The toolbox creates a MAT-file in the current folder with a name you choose.

To save the DWT coefficients from the present analysis, use the menu option **File > Save > Coefficients**.

A dialog box appears that lets you specify a folder and filename for storing the coefficients.

Consider the example analysis:

File > Example Analysis > at level 3, with sym4 → Detail Durer

After saving the discrete wavelet coefficients to the file `cfsdurer.mat`, load the variables into your workspace:

```
load cfsdurer
whos
```

Name	Size	Bytes	Class
coefs	1x142299	1138392	double array
map	64x3	1536	double array
sizes	5x2	80	double array
valTHR	0x0	0	double array
wname	1x4	8	char array

Variable `map` contains the colormap. Variable `wname` contains the wavelet name and `valTHR` is empty since the synthesized image is the same as the original one.

Variables `coefs` and `sizes` contain the discrete wavelet coefficients and the associated matrix sizes. More precisely, in the above example, `coefs` is a 1-by-142299 vector of concatenated coefficients, and `sizes` gives the length of each component.

Saving Decompositions. The **Wavelet 2-D** tool lets you save the entire set of data from a discrete wavelet analysis to disk. The toolbox creates a MAT-file in the current folder with a name you choose, followed by the extension `wa2` (wavelet analysis 2-D).

Open the **Wavelet 2-D** tool and load the example analysis:

File > Example Analysis > at level 3, with sym4 → Detail Durer.

To save the data from this analysis, use the menu option **File > Save > Decomposition**.

A dialog box appears that lets you specify a folder and filename for storing the decomposition data. Type the name `decdurer`.

After saving the decomposition data to the file `decdurer.wa2`, load the variables into your workspace:

```
load decdurer.wa2 -mat
whos
```

Name	Size	Bytes	Class
coefs	1x142299	1138392	double array
data_name	1x6	12	char array
map	64x3	1536	double array
sizes	5x2	80	double array
valTHR	0x0	0	double array
wave_name	1x4	8	char array

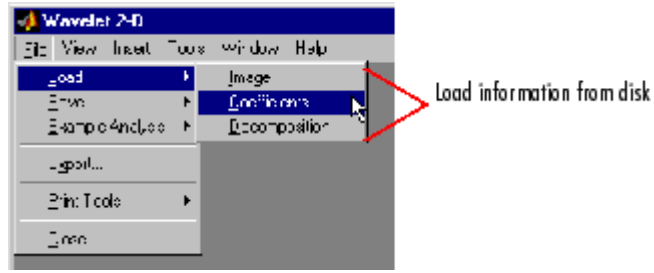
Variables `coefs` and `sizes` contain the wavelet decomposition structure. Other variables contain the wavelet name, the colormap, and the filename containing the data. Variable `valTHR` is empty since the synthesized image is the same as the original one.

Note Save options are also available when performing de-noising or compression inside the **Wavelet 2-D** tool. In the **Wavelet 2-D De-noising** window, you can save denoised image and decomposition. The same holds true for the **Wavelet 2-D Compression** window. This way, you can save many different trials from inside the De-noising and Compression windows without going back to the main **Wavelet 2-D** window during a fine-tuning process. When saving a synthesized signal, a decomposition or coefficients to a MAT-file, the `mat` file extension is not necessary. You can save approximations individually for each level or save them all at once.

Loading Information into the Wavelet 2-D Tool

You can load images, coefficients, or decompositions into the graphical interface. The information you load may have been previously exported from the graphical interface, and then manipulated in the workspace; or it may have been information you generated initially from the command line.

In either case, you must observe the strict file formats and data structures used by the **Wavelet 2-D** tool, or else errors will result when you try to load information.



Loading Images. This toolbox supports only *indexed images*. An indexed image is a matrix containing only integers from 1 to `n`, where `n` is the number of colors in the image.

This image may optionally be accompanied by an `n`-by-3 matrix called `map`. This is the colormap associated with the image. When MATLAB displays such an image, it uses the values of the matrix to look up the desired color in this colormap. If the colormap is not given, the **Wavelet 2-D** tool uses a monotonic colormap with `max(max(X))min(min(X))+1` colors.

To load an image you've constructed in your MATLAB workspace into the **Wavelet 2-D** tool, save the image (and optionally, the variable `map`) in a MAT-file (with extension `mat` or other).

For instance, suppose you've created an image called brain and want to analyze it in the **Wavelet 2-D** tool. Type

```
X = brain;
map = pink(256);
save myfile X map
```

To load this image into the **Wavelet 2-D** tool, use the menu option **File > Load > Image**.

A dialog box appears that lets you select the appropriate MAT-file to be loaded.

Note The graphical tools allow you to load an image that does not contain integers from 1 to n. The computations are correct because they act directly on the matrix, but the display of the image is strange. The values less than 1 are evaluated as 1, the values greater than n are evaluated as n, and a real value within the interval [1,n] is evaluated as the closest integer.

The coefficients, approximations, and details produced by wavelet decomposition are not indexed image matrices.

To display these images in a suitable way, the **Wavelet 2-D** tool follows these rules:

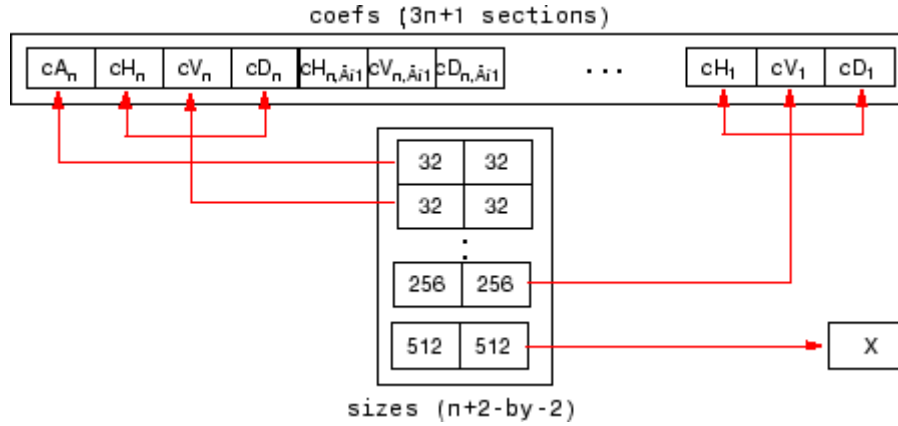
- Reconstructed approximations are displayed using the colormap `map`.
- The coefficients and the reconstructed details are displayed using the colormap `map` applied to a rescaled version of the matrices.

Note The first two-dimensional variable encountered in the file (except the variable `map`, which is reserved for the colormap) is considered the image. Variables are inspected in alphabetical order.

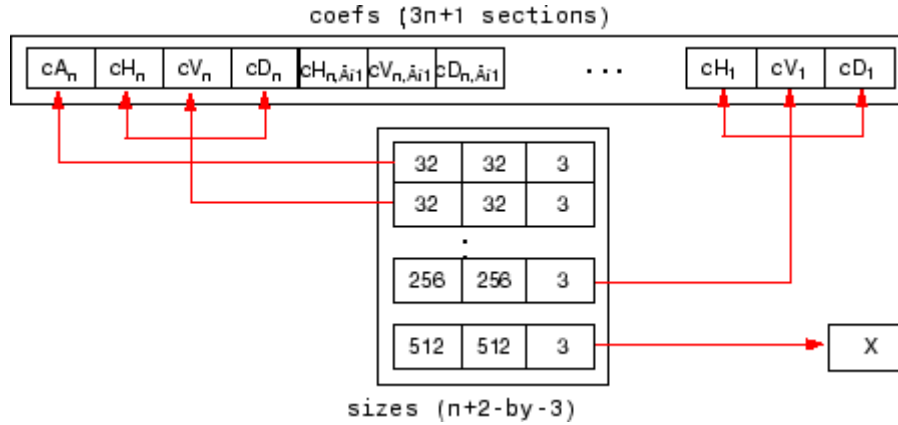
Loading Discrete Wavelet Transform Coefficients. To load discrete wavelet transform (DWT) coefficients into the **Wavelet 2-D** tool, first save the appropriate data in a MAT-file, which must contain at least the two variables:

- `coefs`, the coefficients vector
- `sizes`, the bookkeeping matrix

For an indexed image the matrix `sizes` is a `(n+2-by-2)` array:



For a truecolor image, the matrix `sizes` is a `(n+2-by-3)`:



Variable `coefs` must be a vector of concatenated DWT coefficients. The `coefs` vector for an `n`-level decomposition contains `3n+1` sections, consisting of the level-`n` approximation coefficients, followed by the horizontal, vertical, and diagonal detail coefficients, in that order, for each level. Variable `sizes` is a matrix, the rows of which specify the size of cA_n, the size of cH_n (or cV_n, or cD_n),..., the size of cH_1 (or cV_1, or cD_1), and the size of the original image `X`. The sizes of vertical and diagonal details are the same as the horizontal detail.

After constructing or editing the appropriate data in your workspace, type

```
save myfile coefs sizes
```

Use the **File > Load > Coefficients** menu option from the **Wavelet 2-D** tool to load the data into the graphical tool.

A dialog box appears, allowing you to choose the folder and file in which your data reside.

Loading Decompositions. To load discrete wavelet transform decomposition data into the **Wavelet 2-D** tool, you must first save the appropriate data in a MAT-file (with extension `wa2` or other).

The MAT-file contains these variables.

Variable	Status	Description
`coefs`	Required	Vector of concatenated DWT coefficients
`sizes`	Required	Matrix specifying sizes of components of `coefs` and of the original image
`wave_name`	Required	String specifying name of wavelet used for decomposition (e.g., `db3`)
`map`	Optional	`n`-by-3 colormap matrix.
`data_name`	Optional	String specifying name of decomposition

After constructing or editing the appropriate data in your workspace, type

```
save myfile.wa2 coefs sizes wave_name
```

Use the **File > Load > Decomposition** menu option from the **Wavelet 2-D** tool to load the image decomposition data.

A dialog box appears, allowing you to choose the folder and file in which your data reside.

Note When loading an image, a decomposition, or coefficients from a MAT-file, the extension of this file is free. The `mat` extension is not necessary.

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.)

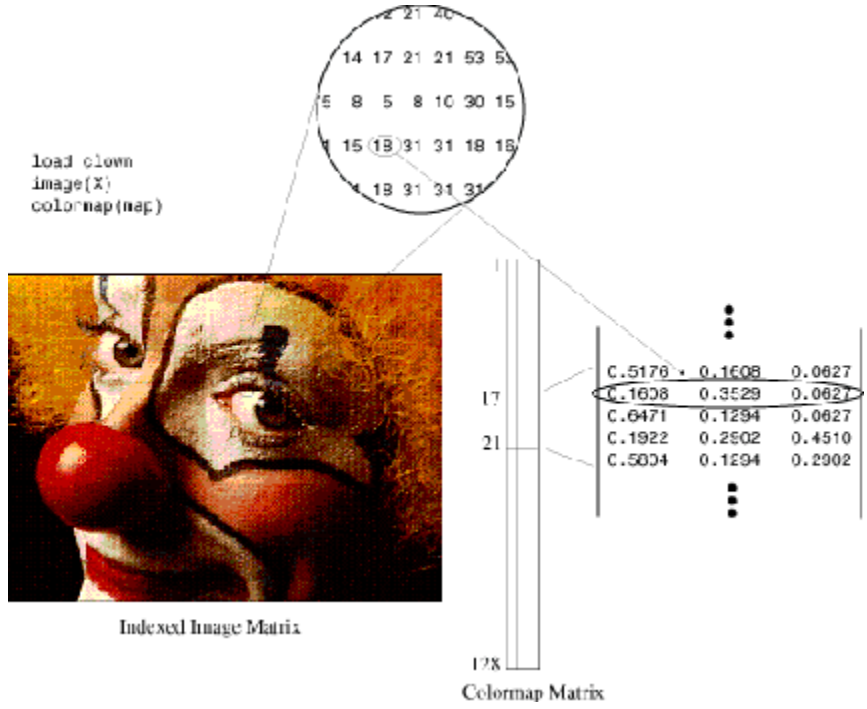
The size of the colormap matrix is `n`-by-3 for an image containing `n` colors. Each row of the colormap matrix is a 1-by-3 red, green, blue (RGB) color vector

```
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

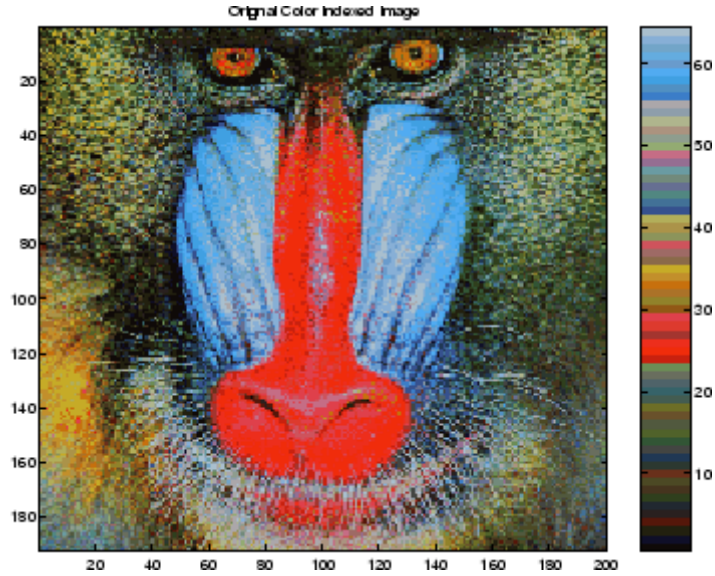
Image Processing Toolbox software provides a comprehensive set of functions that let you easily convert between image types. If you do not have Image Processing Toolbox software, the examples below demonstrate how this conversion may be performed using basic MATLAB commands.

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
```



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

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

```
R = map(X2,1); R = reshape(R,size(X2));
```

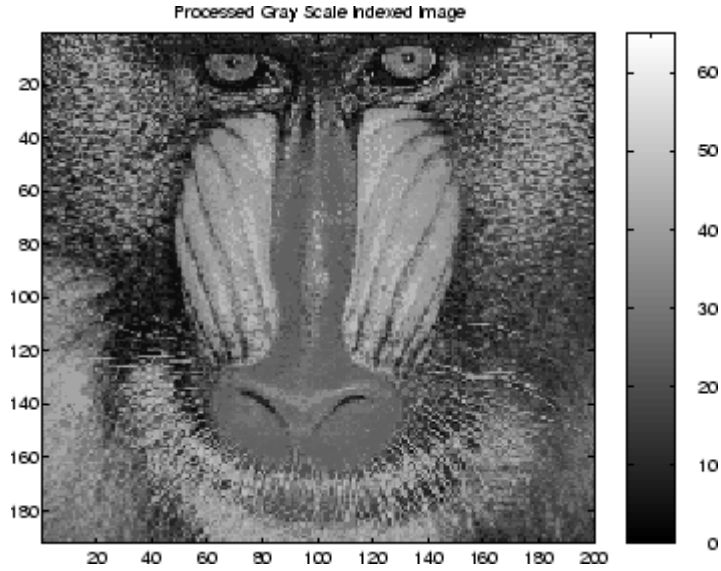
```
G = map(X2,2); G = reshape(G,size(X2));
B = map(X2,3); B = reshape(B,size(X2));
```

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

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

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

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



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

Finally, save the converted image in a form compatible with the Wavelet Toolbox 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 Discrete Stationary Wavelet Analysis

This section takes you through the features of one-dimensional discrete stationary wavelet analysis using the Wavelet Toolbox software. For more information see "Discrete Stationary Wavelet Transform (SWT)" in the *Wavelet Toolbox User's Guide*.

The toolbox provides these functions for one-dimensional discrete stationary wavelet analysis. For more information on the functions, see the reference pages.

Analysis-Decomposition Functions

Function Name	Purpose
`swt`	Decomposition

Synthesis-Reconstruction Functions

Function Name	Purpose
`iswt`	Reconstruction

The stationary wavelet decomposition structure is more tractable than the wavelet one. So the utilities, useful for the wavelet case, are not necessary for the stationary wavelet transform (SWT).

In this section, you'll learn to

- Load a signal
- Perform a stationary wavelet decomposition of a signal
- Construct approximations and details from the coefficients
- Display the approximation and detail at level 1
- Regenerate a signal by using inverse stationary wavelet transform
- Perform a multilevel stationary wavelet decomposition of a signal
- Reconstruct the level 3 approximation

- Reconstruct the level 1, 2, and 3 details
- Reconstruct the level 1 and 2 approximations
- Display the results of a decomposition
- Reconstruct the original signal from the level 3 decomposition
- Remove noise from a signal

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

The final subsection discusses how to exchange signal and coefficient information between the disk and the graphical tools.

One-Dimensional Analysis Using the Command Line

This example involves a noisy Doppler test signal.

1 Load a signal.

From the MATLAB prompt, type

```
load noisdopp
```

2 Set the variables. Type

```
s = noisdopp;
```

For the SWT, if a decomposition at level `k` is needed, `2^k` must divide evenly into the length of the signal. If your original signal does not have the correct length, you can use the **Signal Extension** GUI tool or the `wextend` function to extend it.

3 Perform a single-level Stationary Wavelet Decomposition.

Perform a single-level decomposition of the signal using the `db1` wavelet. Type

```
[swa,swd] = swt(s,1,'db1');
```

This generates the coefficients of the level 1 approximation (swa) and detail (swd). Both are of the same length as the signal. Type

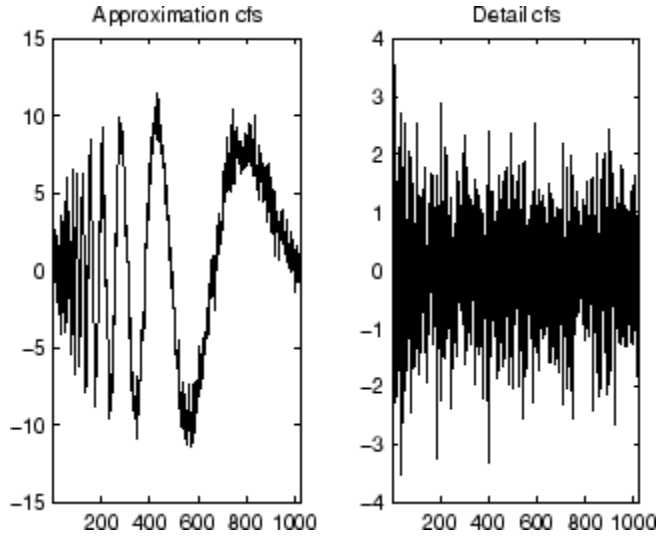
```
whos
```

Name	Size	Bytes	Class
noisdopp	1x1024	8192	double array
s	1x1024	8192	double array
swa	1x1024	8192	double array
swd	1x1024	8192	double array

4 Display the coefficients of approximation and detail.

To display the coefficients of approximation and detail at level 1, type

```
subplot(1,2,1), plot(swa); title('Approximation cfs')
subplot(1,2,2), plot(swd); title('Detail cfs')
```



5 Regenerate the signal by Inverse Stationary Wavelet Transform.

To find the inverse transform, type

```
A0 = iswt(swa,swd,'db1');
```

To check the perfect reconstruction, type

```
err = norm(s-A0)
err =
 2.1450e-14
```

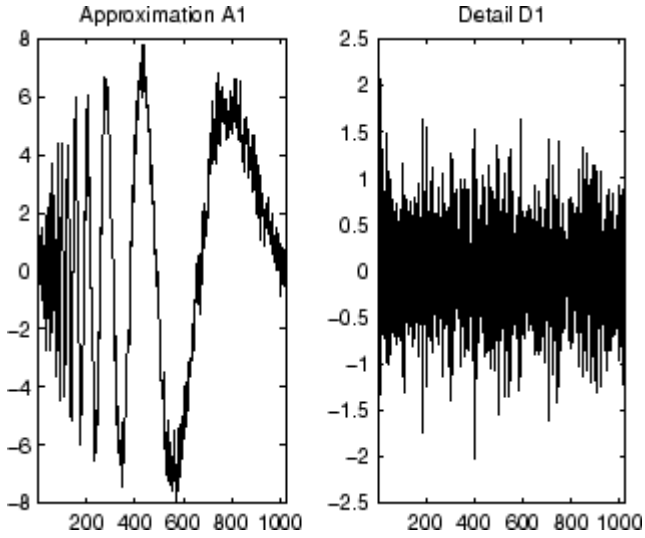
6 Construct and display approximation and detail from the coefficients.

To construct the level 1 approximation and detail (`A1` and `D1`) from the coefficients `swa` and `swd`, type

```
nulcfs = zeros(size(swa));
A1 = iswt(swa,nulcfs,'db1');
D1 = iswt(nulcfs,swd,'db1');
```

To display the approximation and detail at level 1, type

```
subplot(1,2,1), plot(A1); title('Approximation A1');
subplot(1,2,2), plot(D1); title('Detail D1');
```



7 Perform a multilevel Stationary Wavelet Decomposition.

To perform a decomposition at level 3 of the signal (again using the `db1` wavelet), type

```
[swa,swd] = swt(s,3,'db1');
```

This generates the coefficients of the approximations at levels 1, 2, and 3 (`swa`) and the coefficients of the details (`swd`). Observe that the rows of `swa` and `swd` are the same length as the signal length. Type

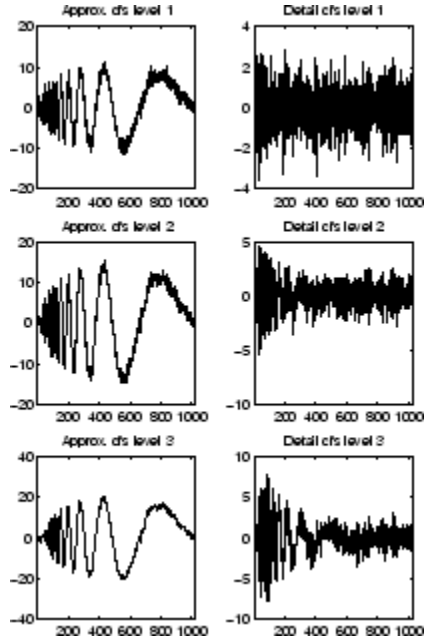
```
clear A0 A1 D1 err nulcfs
whos
```

Name	Size	Bytes	Class
noisdopp	1x1024	8192	double array
s	1x1024	8192	double array
swa	3x1024	24576	double array
swd	3x1024	24576	double array

8 Display the coefficients of approximations and details.

To display the coefficients of approximations and details, type

```
kp = 0;
for i = 1:3
    subplot(3,2,kp+1), plot(swa(i,:));
    title(['Approx. cfs level ',num2str(i)])
    subplot(3,2,kp+2), plot(swd(i,:));
    title(['Detail cfs level ',num2str(i)])
    kp = kp + 2;
end
```



9 Reconstruct approximation at Level 3 From coefficients.

To reconstruct the approximation at level 3, type

```
mzero = zeros(size(swd));
A = mzero;
A(3,:) = iswt(swa,mzero,'db1');
```

10 Reconstruct details from coefficients.

To reconstruct the details at levels 1, 2 and 3, type

```
D = mzero;
for i = 1:3
    swcfs = mzero;
    swcfs(i,:) = swd(i,:);
    D(i,:) = iswt(mzero,swcfs,'db1');
end
```

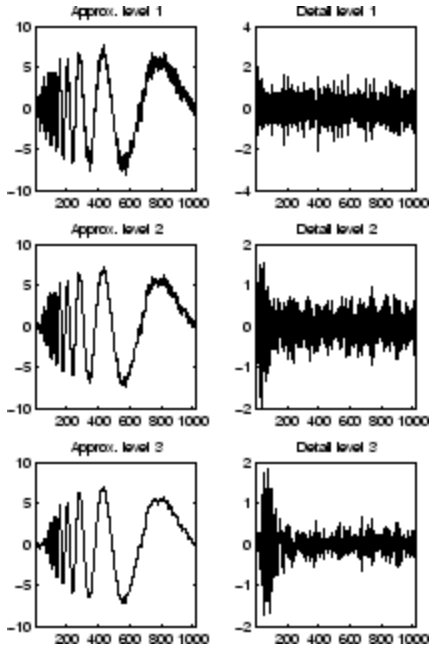
11 Reconstruct and display approximations at Levels 1 and 2 from approximation at Level 3 and details at Levels 2 and 3.

To reconstruct the approximations at levels 2 and 3, type

```
A(2,:) = A(3,:) + D(3,:);
A(1,:) = A(2,:) + D(2,:);
```

To display the approximations and details at levels 1, 2 and 3, type

```
kp = 0;
for i = 1:3
    subplot(3,2,kp+1), plot(A(i,:));
    title(['Approx. level ',num2str(i)])
    subplot(3,2,kp+2), plot(D(i,:));
    title(['Detail level ',num2str(i)])
    kp = kp + 2;
end
```



12 Remove noise by thresholding.

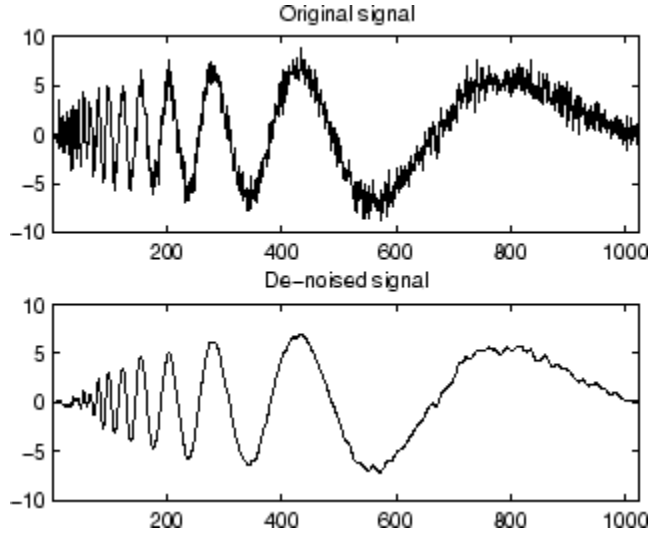
To denoise the signal, use the `ddencmp` command to calculate a default global threshold. Use the `wthresh` command to perform the actual thresholding of the detail coefficients, and then use the `iswt` command to obtain the denoised signal.

Note All methods for choosing thresholds in the 1-D Discrete Wavelet Transform case are also valid for the 1-D Stationary Wavelet Transform, which are also those used by the GUI tools. This is also true for the 2-D transforms.

```
[thr,sorh] = ddencmp('den','wv',s);
dswd = wthresh(swd,sorh,thr);
clean = iswt(swa,dswd,'db1');
```

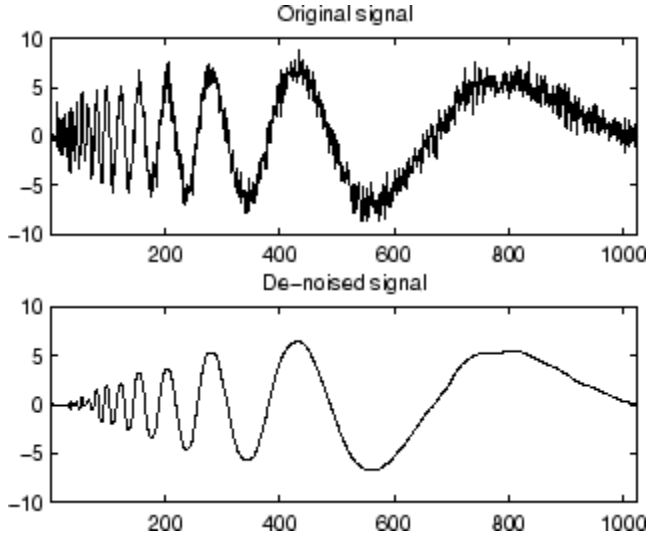
To display both the original and denoised signals, type

```
subplot(2,1,1), plot(s);
title('Original signal')
subplot(2,1,2), plot(clean);
title('denoised signal')
```



The obtained signal remains a little bit noisy. The result can be improved by considering the decomposition of `s` at level 5 instead of level 3, and repeating steps 14 and 15. To improve the previous de-noising, type

```
[swa,swd] = swt(s,5,'db1');
[thr,sorh] = ddencmp('den','wv',s);
dswd = wthresh(swd,sorh,thr);
clean = iswt(swa,dswd,'db1');
subplot(2,1,1), plot(s); title('Original signal')
subplot(2,1,2), plot(clean); title('denoised signal')
```



A second syntax can be used for the `swt` and `iswt` functions, giving the same results:

```
lev = 5; swc = swt(s,lev,'db1');
swcden = swc;
swcden(1:end-1,:) = wthresh(swcden(1:end-1,:),sorh,thr);
clean = iswt(swcden,'db1');
```

You can obtain the same plot by using the same plot commands as in step 16 above.

One-Dimensional Analysis for De-Noising Using the Graphical Interface

Now we explore a strategy to denoise signals, based on the one-dimensional stationary wavelet analysis using the graphical interface tools. The basic idea is to average many slightly different discrete wavelet analyses.

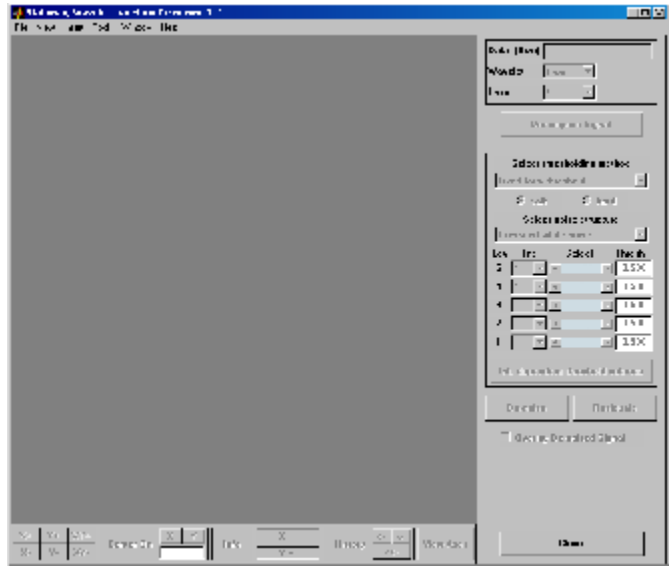
1 Start the Stationary Wavelet Transform De-Noising 1-D Tool.

From the MATLAB prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.

Click the **SWT De-noising 1-D** menu item. The discrete stationary wavelet transform de-noising tool for one-dimensional signals appears.

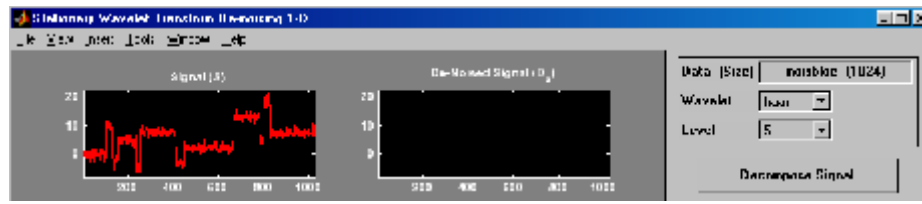


2 Load data.

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

When the **Load Signal** dialog box appears, select the MAT-file `noisbloc.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`.

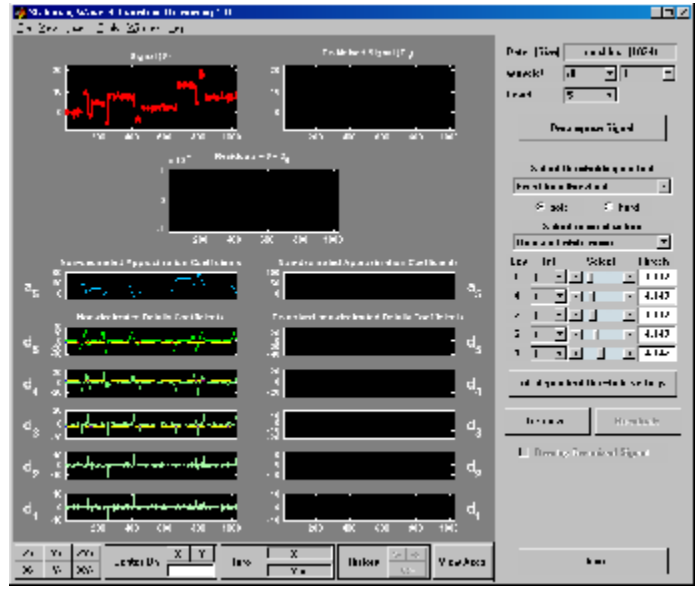
Click the **OK** button. The noisy blocks signal is loaded into the **SWT De-noising 1-D** tool.



3 Perform a Stationary Wavelet Decomposition.

Select the `db1` wavelet from the **Wavelet** menu and select **5** from the **Level** menu, and then click the **Decompose Signal** button. After a pause for

computation, the tool displays the stationary wavelet approximation and detail coefficients of the decomposition. These are also called nondecimated coefficients since they are obtained using the same scheme as for the DWT, but omitting the decimation step (see "Fast Wavelet Transform (FWT) Algorithm" in the *Wavelet Toolbox User's Guide*).

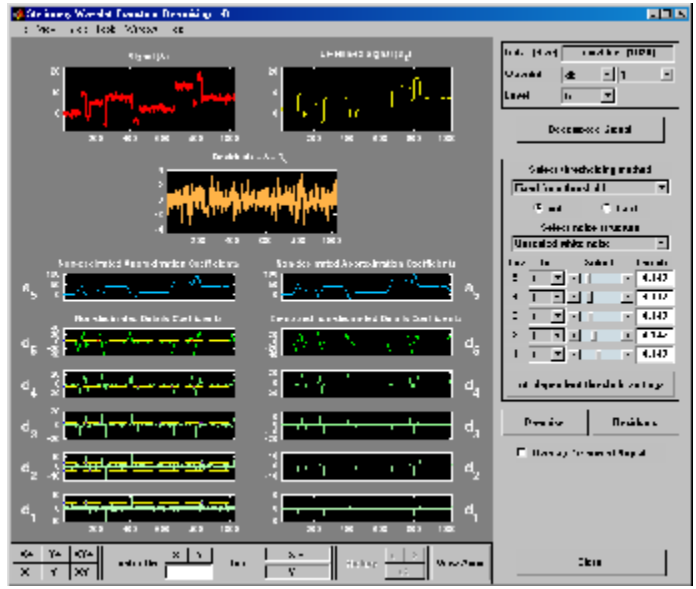


4 denoise the signal using the Stationary Wavelet Transform.

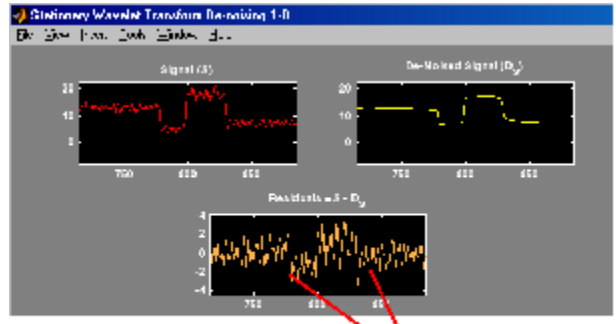
While a number of options are available for fine-tuning the de-noising algorithm, we'll accept the defaults of fixed form soft thresholding and unscaled white noise. The sliders located on the right part of the window control the level-dependent thresholds, indicated by yellow dotted lines running horizontally through the graphs of the detail coefficients to the left of the window. The yellow dotted lines can also be dragged directly using the left mouse button over the graphs.

Note that the approximation coefficients are not thresholded.

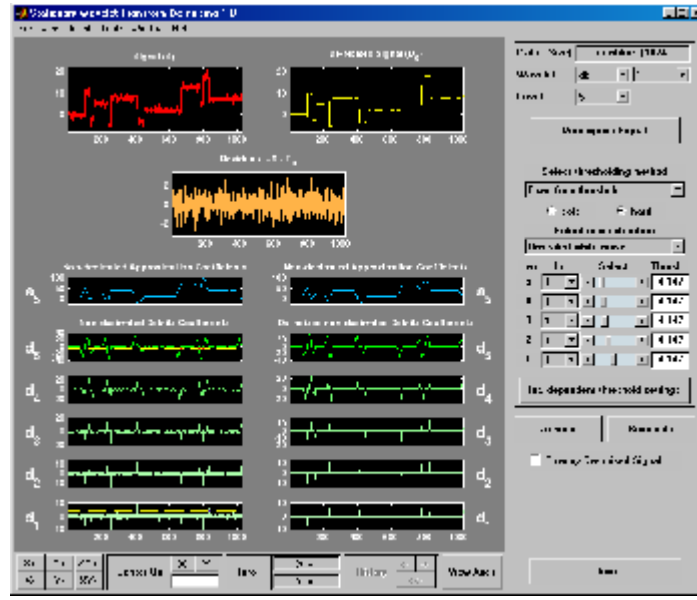
Click the **denoise** button.



The result is quite satisfactory, but seems to be oversmoothed around the discontinuities of the signal. This can be seen by looking at the residuals, and zooming on a breakdown point, for example around position 800.



Selecting a Thresholding Method. Select **hard** for the thresholding mode instead of **soft**, and then click the **denoise** button.



The result is of good quality and the residuals look like a white noise sample. To investigate this last point, you can get more information on residuals by clicking the **Residuals** button.

Importing and Exporting from the GUI

The tool lets you save the denoised signal to disk. The toolbox creates a MAT-file in the current folder with a name of your choice.

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

```
load dnoibloc
whos
```

Name	Size	Bytes	Class
dnoibloc	1x1024	8192	double array
thrParams	1x5	580	cell array
wname	1x3	6	char array

The denoised signal is given by `dnoibloc`. In addition, the parameters of the de-noising process are available. The wavelet name is contained in `wname`:

```
wname

wname =
    db1
```

and the level dependent thresholds are encoded in `thrParams`, which is a cell array of length 5 (the level of the decomposition). For i from 1 to 5, `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 Variance Adaptive Thresholding of Wavelet Coefficients" on page 2-145.

For example, for level 1,

```
thrParams{1}
 ans =
      1.0e+03 *
      0.0010 1.0240 0.0041
```

Here the lower bound is 1, the upper bound is 1024, and the threshold value is 4.1. The total time-interval is not segmented and the procedure does not use the interval dependent thresholds.

Two-Dimensional Discrete Stationary Wavelet Analysis

This section takes you through the features of two-dimensional discrete stationary wavelet analysis using the Wavelet Toolbox software. For more information, see "Available Methods for De-Noising, Estimation, and Compression Using GUI Tools" in the *Wavelet Toolbox User's Guide*.

The toolbox provides these functions for image analysis. For more information, see the reference pages.

Analysis-Decomposition Function

Function Name	Purpose
`swt2`	Decomposition

Synthesis-Reconstruction Function

Function Name	Purpose
`iswt2`	Reconstruction

The stationary wavelet decomposition structure is more tractable than the wavelet one. So, the utilities useful for the wavelet case are not necessary for the Stationary Wavelet Transform (SWT).

In this section, you'll learn to

- Load an image
- Analyze an image
- Perform single-level and multilevel image decompositions and reconstructions (command line only)
- denoise an image

Two-Dimensional Analysis Using the Command Line

In this example, we'll show how you can use two-dimensional stationary wavelet analysis to denoise an image.

Note Instead of using `image(I)` to visualize the image `I`, we use `image(wcodemat(I))`, which displays a rescaled version of `I` leading to a clearer presentation of the details and approximations (see the `wcodemat` reference page).

This example involves a image containing noise.

1 Load an image.

From the MATLAB prompt, type

```
load noiswom
whos
```

Name	Size	Bytes	Class
X	96x96	73728	double array
map	255x3	6120	double array

For the SWT, if a decomposition at level `k` is needed, `2^k` must divide evenly into `size(X,1)` and `size(X,2)`. If your original image is not of correct size, you can use the **Image Extension** GUI tool or the function `wextend` to extend it.

2 Perform a single-level Stationary Wavelet Decomposition.

Perform a single-level decomposition of the image using the `db1` wavelet. Type

```
[swa,swh,swv,swd] = swt2(X,1,'db1');
```

This generates the coefficients matrices of the level-one approximation (`swa`) and horizontal, vertical and diagonal details (`swh`, `swv`, and `swd`, respectively). Both are of size-the-image size. Type

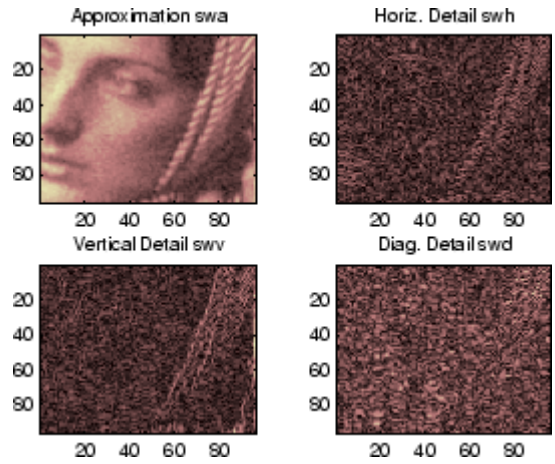
```
whos
```

Name	Size	Bytes	Class
X	96x96	73728	double array
map	255x3	6120	double array
swa	96x96	73728	double array
swh	96x96	73728	double array
swv	96x96	73728	double array
swd	96x96	73728	double array

3 Display the coefficients of approximation and details.

To display the coefficients of approximation and details at level 1, type

```
map = pink(size(map,1)); colormap(map)
subplot(2,2,1), image(wcodemat(swa,192));
title('Approximation swa')
subplot(2,2,2), image(wcodemat(swh,192));
title('Horiz. Detail swh')
subplot(2,2,3), image(wcodemat(swv,192));
title('Vertical Detail swv')
subplot(2,2,4), image(wcodemat(swd,192));
title('Diag. Detail swd').
```



4 Regenerate the image by Inverse Stationary Wavelet Transform.

To find the inverse transform, type

```
A0 = iswt2(swa,swh,swv,swd,'db1');
```

To check the perfect reconstruction, type

```
err = max(max(abs(X-A0)))

err =
     1.1369e-13
```

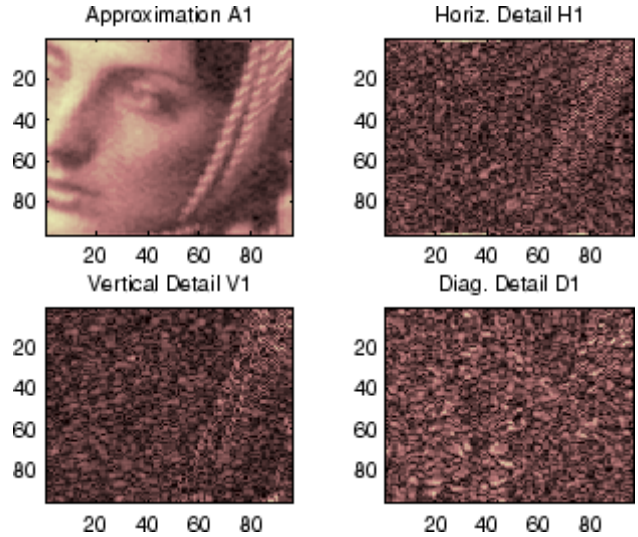
5 Construct and display approximation and details from the coefficients.

To construct the level 1 approximation and details (`A1`, `H1`, `V1` and `D1`) from the coefficients `swa`, `swh`, `swv` and `swd`, type

```
nulcfs = zeros(size(swa));
A1 = iswt2(swa,nulcfs,nulcfs,nulcfs,'db1');
H1 = iswt2(nulcfs,swh,nulcfs,nulcfs,'db1');
V1 = iswt2(nulcfs,nulcfs,swv,nulcfs,'db1');
D1 = iswt2(nulcfs,nulcfs,nulcfs,swd,'db1');
```

To display the approximation and details at level 1, type

```
colormap(map)
subplot(2,2,1), image(wcodemat(A1,192));
title('Approximation A1')
subplot(2,2,2), image(wcodemat(H1,192));
title('Horiz. Detail H1')
subplot(2,2,3), image(wcodemat(V1,192));
title('Vertical Detail V1')
subplot(2,2,4), image(wcodemat(D1,192));
title('Diag. Detail D1')
```



6 Perform a multilevel Stationary Wavelet Decomposition.

To perform a decomposition at level 3 of the image (again using the `db1` wavelet), type

```
[swa,swh,swv,swd] = swt2(X,3,'db1');
```

This generates the coefficients of the approximations at levels 1, 2, and 3 (`swa`) and the coefficients of the details (`swh`, `swv` and `swd`). Observe that the matrices `swa(:,:,i)`, `swh(:,:,i)`, `swv(:,:,i)`, and `swd(:,:,i)` for a given level `i` are of size-the-image size. Type

```
clear A0 A1 D1 H1 V1 err nulcfs
whos
```

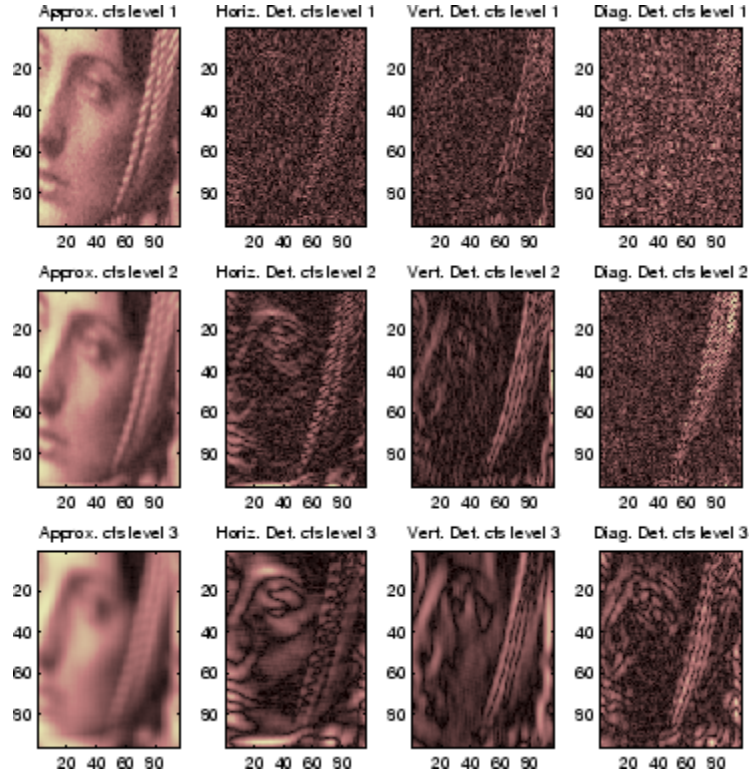
Name	Size	Bytes	Class
X	96x96	73728	double array
map	255x3	6120	double array
swa	96x96x3	221184	double array
swh	96x96x3	221184	double array

Name	Size	Bytes	Class
swv	96x96x3	221184	double array
swd	96x96x3	221184	double array

7 Display the coefficients of approximations and details.

To display the coefficients of approximations and details, type

```
colormap(map)
kp = 0;
for i = 1:3
    subplot(3,4,kp+1), image(wcodemat(swa(:,:,i),192));
    title(['Approx. cfs level ',num2str(i)])
    subplot(3,4,kp+2), image(wcodemat(swh(:,:,i),192));
    title(['Horiz. Det. cfs level ',num2str(i)])
    subplot(3,4,kp+3), image(wcodemat(swv(:,:,i),192));
    title(['Vert. Det. cfs level ',num2str(i)])
    subplot(3,4,kp+4), image(wcodemat(swd(:,:,i),192));
    title(['Diag. Det. cfs level ',num2str(i)])
    kp = kp + 4;
end
```



8 Reconstruct approximation at Level 3 and details from coefficients.

To reconstruct the approximation at level 3, type

```
mzero = zeros(size(swd));
A = mzero;
A(:,:,3) = iswt2(swa,mzero,mzero,mzero,'db1');
```

To reconstruct the details at levels 1, 2 and 3, type

```
H = mzero; V = mzero;
D = mzero;
for i = 1:3
    swcfs = mzero; swcfs(:,:,i) = swh(:,:,i);
    H(:,:,i) = iswt2(mzero,swcfs,mzero,mzero,'db1');
    swcfs = mzero; swcfs(:,:,i) = swv(:,:,i);
```

```
    V(:,:,i) = iswt2(mzero,mzero,swcfs,mzero,'db1');
    swcfs = mzero; swcfs(:,:,i) = swd(:,:,i);
    D(:,:,i) = iswt2(mzero,mzero,mzero,swcfs,'db1');
end
```

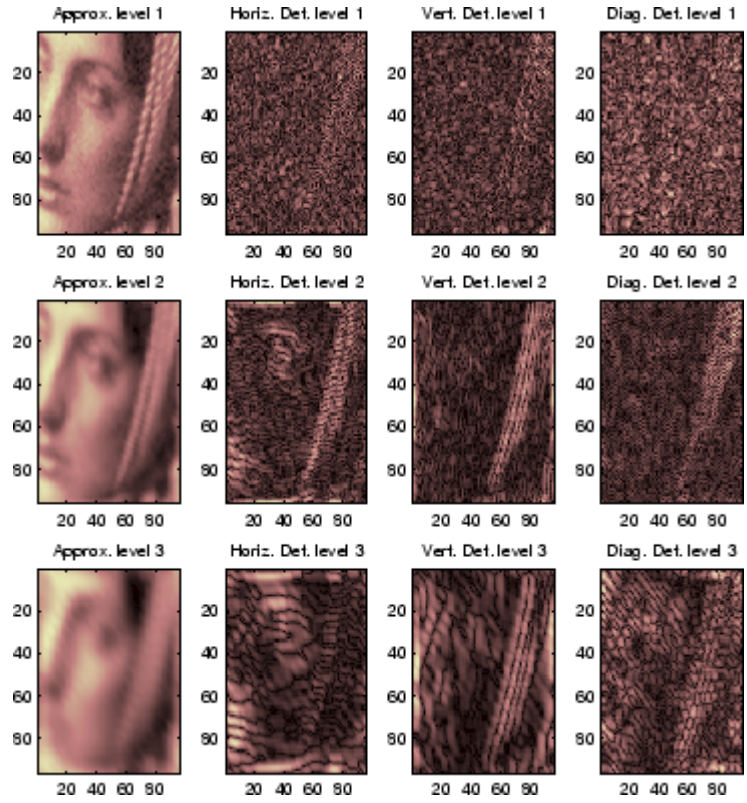
9 Reconstruct and display approximations at Levels 1, 2 from approximation at Level 3 and details at Levels 1, 2, and 3.

To reconstruct the approximations at levels 2 and 3, type

```
A(:,:,2) = A(:,:,3) + H(:,:,3) + V(:,:,3) + D(:,:,3);
A(:,:,1) = A(:,:,2) + H(:,:,2) + V(:,:,2) + D(:,:,2);
```

To display the approximations and details at levels 1, 2, and 3, type

```
colormap(map)
kp = 0;
for i = 1:3
    subplot(3,4,kp+1), image(wcodemat(A(:,:,i),192));
    title(['Approx. level ',num2str(i)])
    subplot(3,4,kp+2), image(wcodemat(H(:,:,i),192));
    title(['Horiz. Det. level ',num2str(i)])
    subplot(3,4,kp+3), image(wcodemat(V(:,:,i),192));
    title(['Vert. Det. level ',num2str(i)])
    subplot(3,4,kp+4), image(wcodemat(D(:,:,i),192));
    title(['Diag. Det. level ',num2str(i)])
    kp = kp + 4;
end
```



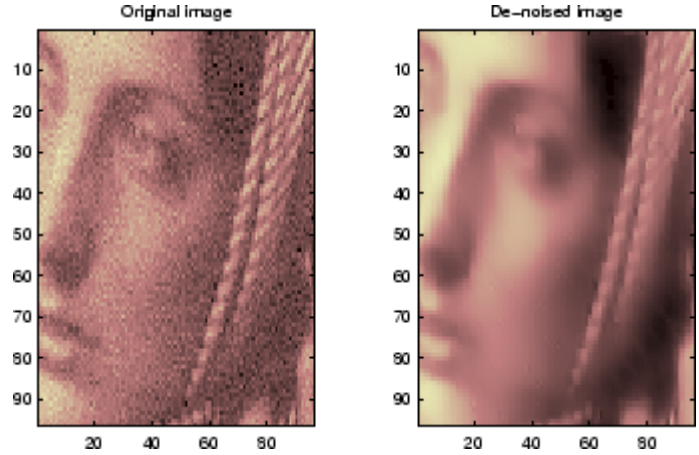
10 Remove noise by thresholding.

To denoise an image, use the threshold value we find using the GUI tool (see the next section), use the `wthresh` command to perform the actual thresholding of the detail coefficients, and then use the `iswt2` command to obtain the denoised image.

```
thr = 44.5;
sorh = 's'; dswh = wthresh(swh,sorh,thr);
dswv = wthresh(swv,sorh,thr);
dswd = wthresh(swd,sorh,thr);
clean = iswt2(swa,dswh,dswv,dswd,'db1');
```

To display both the original and denoised images, type

```
colormap(map)
subplot(1,2,1), image(wcodemat(X,192));
title('Original image')
subplot(1,2,2), image(wcodemat(clean,192));
title('denoised image')
```



A second syntax can be used for the `swt2` and `iswt2` functions, giving the same results:

```
lev= 4;
swc = swt2(X,lev,'db1');
swcden = swc;
swcden(:,:,1:end-1) =
wthresh(swcden(:,:,1:end-1),sorh,thr);
clean = iswt2(swcden,'db1');
```

You obtain the same plot by using the plot commands in step 9 above.

Two-Dimensional Analysis for De-Noising Using the Graphical Interface

In this section, we explore a strategy for de-noising images based on the two-dimensional stationary wavelet analysis using the graphical interface tools. The basic idea is to average many slightly different discrete wavelet analyses.

1 Start the Stationary Wavelet Transform De-Noising 2-D Tool.

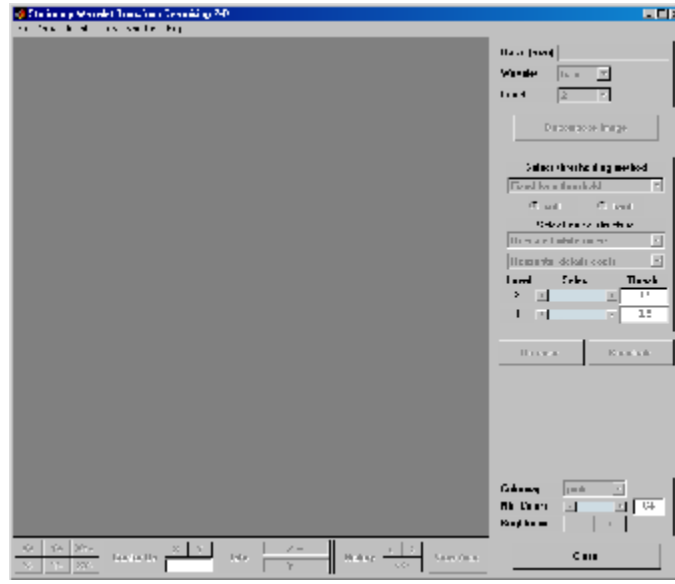
From the MATLAB prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears:

Click the **SWT De-noising 2-D** menu item.

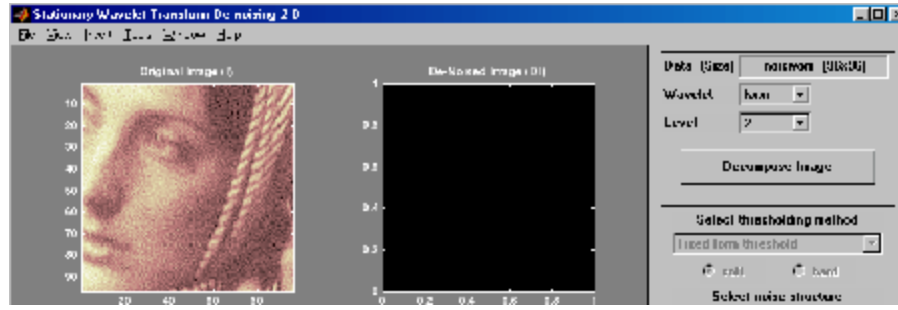
The discrete stationary wavelet transform de-noising tool for images appears.



2 Load data.

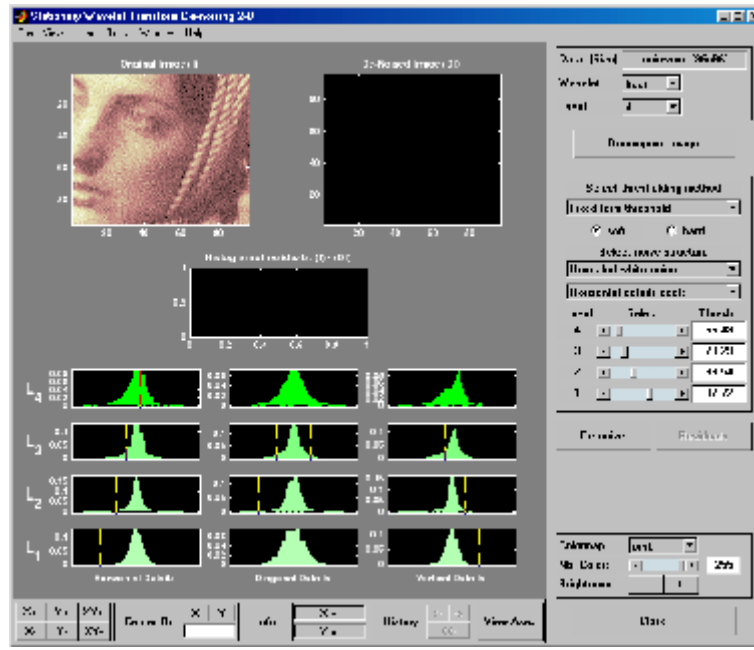
From the **File** menu, choose the **Load Image** option.

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 **OK** button. The noisy woman image is loaded into the **SWT De-noising 2-D** tool.



3 Perform a Stationary Wavelet Decomposition.

Select the `haar` wavelet from the **Wavelet** menu, select **4** from the **Level** menu, and then click the **Decompose Image** button.

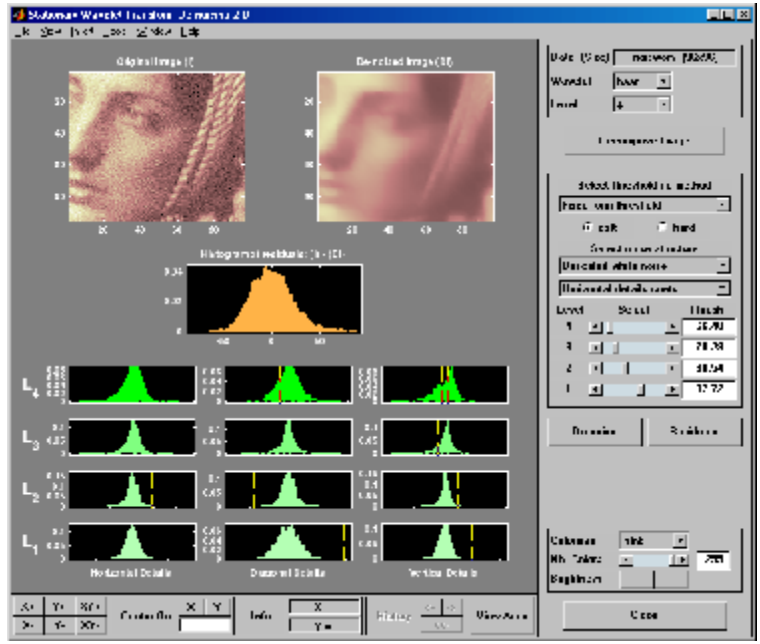


The tool displays the histograms of the stationary wavelet detail coefficients of the image on the left of the window. These histograms are organized as follows:

- From the bottom for level 1 to the top for level 4
- On the left horizontal coefficients, in the middle diagonal coefficients, and on the right vertical coefficients

4 denoise the image using the Stationary Wavelet Transform.

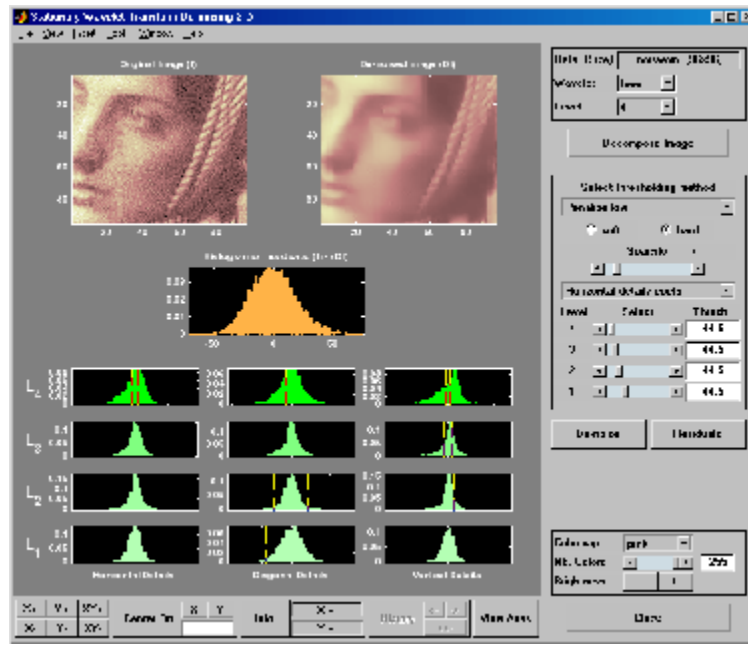
While a number of options are available for fine-tuning the de-noising algorithm, we'll accept the defaults of fixed form soft thresholding and unscaled white noise. The sliders located to the right of the window control the level dependent thresholds indicated by yellow dotted lines running vertically through the histograms of the coefficients on the left of the window. Click the **denoise** button.



The result seems to be oversmoothed and the selected thresholds too aggressive. Nevertheless, the histogram of the residuals is quite good since it is close to a Gaussian distribution, which is the noise introduced to produce the analyzed image `noiswom.mat` from a piece of the original image `woman.mat`.

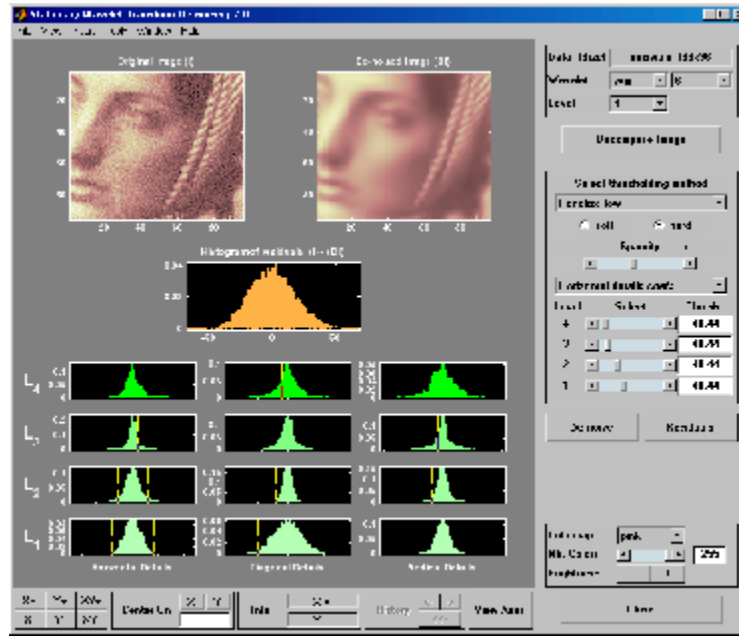
5 Selecting a thresholding method.

From the **Select thresholding method** menu, choose the **Penalize low** item. The associated default for the thresholding mode is automatically set to **hard**; accept it. Use the **Sparsity** slider to adjust the threshold value close to 44.5, and then click the **denoise** button.



The result is quite satisfactory, although it is possible to improve it slightly.

Select the `sym6` wavelet and click the **Decompose Image** button. Use the **Sparsity** slider to adjust the threshold value close to 40.44, and then click the **denoise** button.



Importing and Exporting Information from the Graphical Interface

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

To save the denoised image from the present de-noising process, use the menu **File > Save denoised Image**. A dialog box appears that lets you specify a folder and filename for storing the image. Type the name `dnoiswom`. After saving the image data to the file `dnoiswom.mat`, load the variables into your workspace:

```
load dnoiswom
whos
```

Name	Size	Bytes	Class
X	96x96	73728	double array
map	255x3	6120	double array

Name	Size	Bytes	Class
valTHR	3x4	96	double array
wname	1x4	8	char array

The denoised image is X and map is the colormap. In addition, the parameters of the de-noising process are available. The wavelet name is contained in wname, and the level dependent thresholds are encoded in valTHR. The variable valTHR has four columns (the level of the decomposition) and three rows (one for each detail orientation).

One-Dimensional Wavelet Regression Estimation

This section takes you through the features of one-dimensional wavelet regression estimation using one of the Wavelet Toolbox specialized tools. The toolbox provides a graphical interface tool to explore some de-noising schemes for equally or unequally sampled data.

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

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

One-Dimensional Estimation Using the GUI for Equally Spaced Observations (Fixed Design)

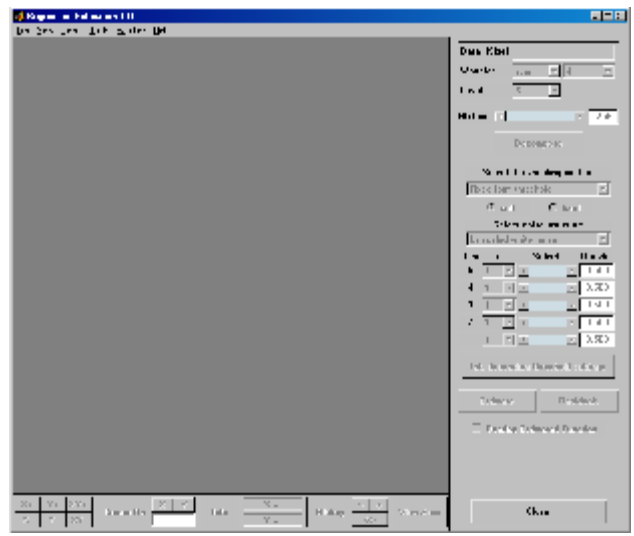
1 Start the Regression Estimation 1-D Tool.

From the MATLAB prompt, type

```
wavemenu
```

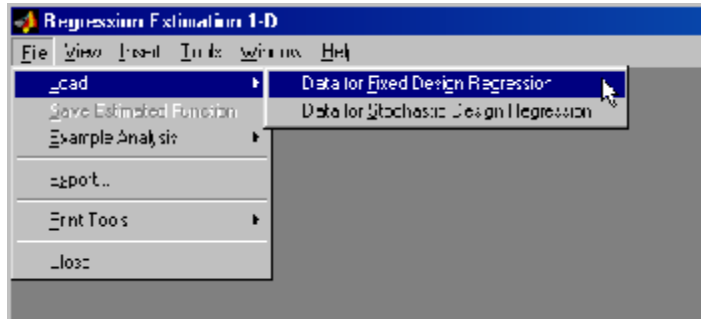
The **Wavelet Toolbox Main Menu** appears.

Click the **Regression Estimation 1-D** menu item. The discrete wavelet analysis tool for one-dimensional regression estimation appears.



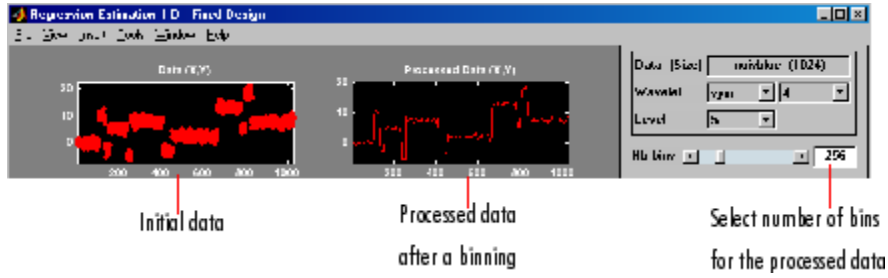
2 Load data.

From the **File** menu, choose the **Load Data for Fixed Design Regression** option.



When the **Load data for Fixed Design Regression** dialog box appears, select the demo MAT-file `noisbloc.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`.

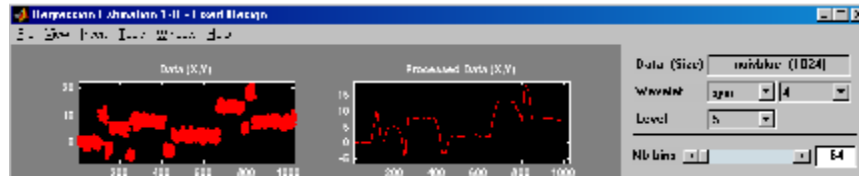
Click the **OK** button. The noisy blocks data is loaded into the **Regression Estimation 1-D - Fixed Design** tool.



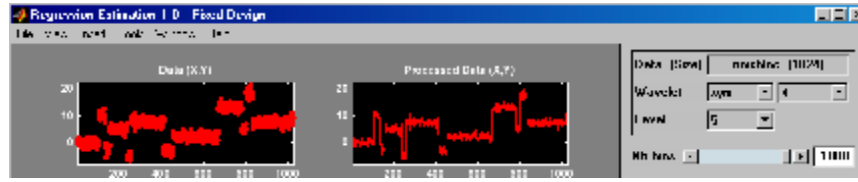
The loaded data denoted (X,Y) and the processed data obtained after a binning, are displayed.

3 Choose the processed data.

The default value for the number of bins is 256 for this example. Enter 64 in the **Nb bins** (number of bins) edit box, or use the slider to adjust the value. The new binned data to be processed appears.



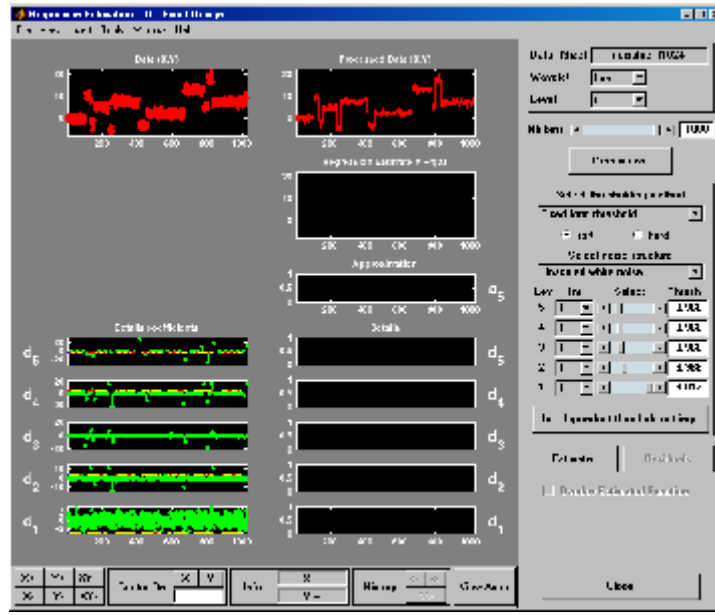
The binned data appears to be very smoothed. Select 1000 from the **Nb bins** edit and press **Enter** or use the slider. The new data to be processed appears.



The binned data appears to be very close to the initial data, since `noisbloc` is of length 1024.

4 Perform a Wavelet Decomposition of the processed data.

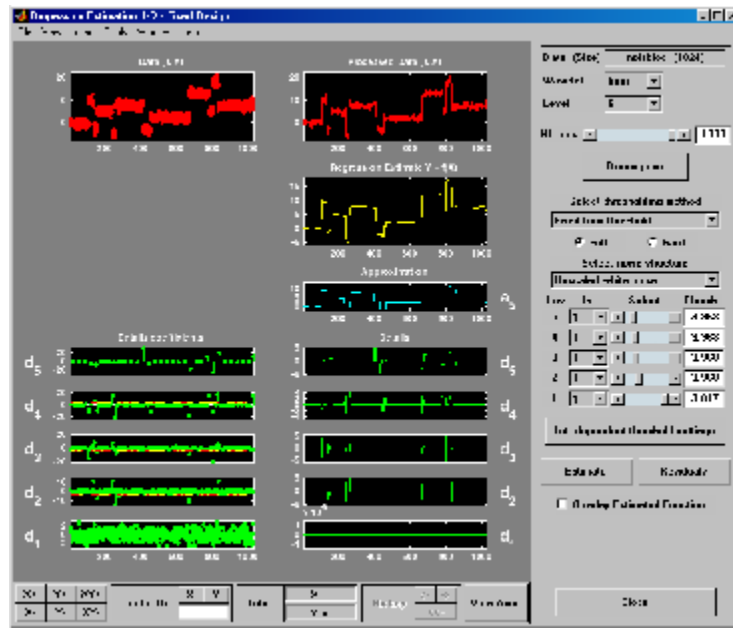
Select the `haar` wavelet from the **Wavelet** menu and select **5** from the **Level** menu, and then click the **Decompose** button. After a pause for computation, the tool displays the detail coefficients of the decomposition.



5 Perform a regression estimation.

While a number of options are available for fine-tuning the estimation algorithm, we'll accept the defaults of fixed form soft thresholding and unscaled white noise. The sliders located to the right of the window control the level dependent thresholds, indicated by yellow dotted lines running horizontally through the graphs on the left part of the window.

Continue by clicking the **Estimate** button.



You can see that the process removed the noise and that the blocks are well reconstructed. The regression estimate (in yellow) is the sum of the signals located below it: the approximation a5 and the reconstructed details after coefficient thresholding.

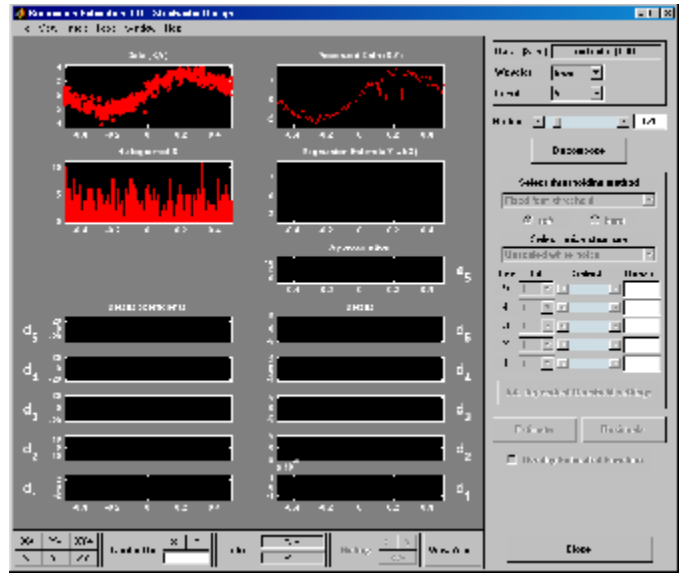
You can experiment with the various predefined thresholding strategies by selecting the appropriate options from the menu located on the right part of the window or directly by dragging the yellow horizontal lines with the left mouse button.

Let us now illustrate the regression estimation using the graphical interface for randomly or irregularly spaced observations, focusing on the differences from the previous situation.

One-Dimensional Estimation Using the GUI for Randomly Spaced Observations (Stochastic Design)

1 From the **File** menu, choose the **Load > Data for Stochastic Design Regression** option. When the **Load data for Stochastic Design**

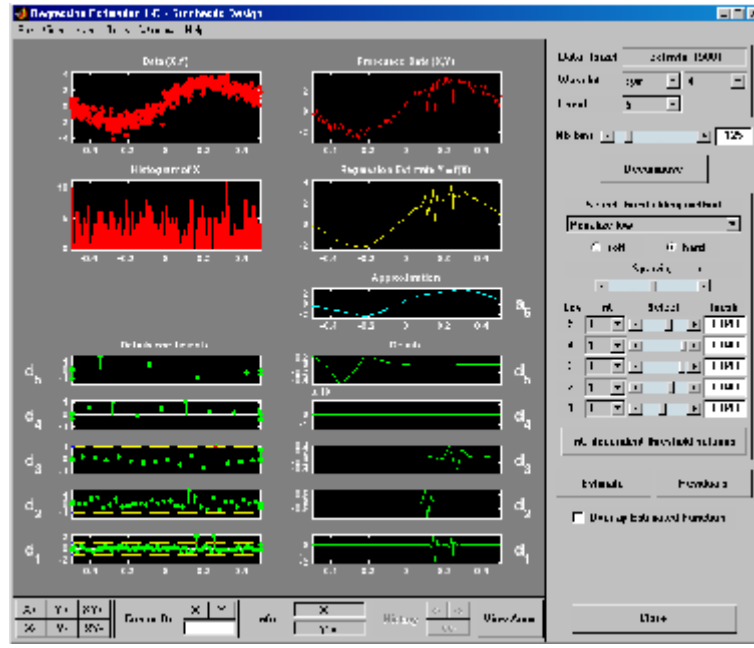
Regression dialog box appears, select the demo MAT-file `ex1nsto.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`. Click the **OK** button. This short set of data (of size 500) is loaded into the **Regression Estimation 1-D -- Stochastic Design** tool.



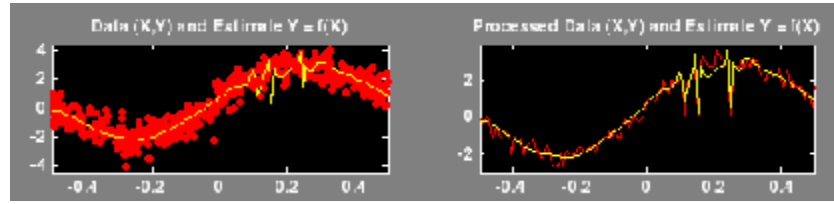
The loaded data denoted (X,Y), the histogram of X, and the processed data obtained after a binning are displayed. The histogram is interesting, because the values of X are randomly distributed. The binning step is essential since it transforms a problem of regression estimation for irregularly spaced X data into a classical fixed design scheme for which fast wavelet transform can be used.

2 Select the `sym4` wavelet from the **Wavelet** menu, select **5** from the **Level** menu, and enter 125 in the **Nb bins** edit box. Click the **Decompose** button. The tool displays the detail coefficients of the decomposition.

3 From the **Select thresholding method** menu, select the item **Penalize low** and click the **Estimate** button.



4 Check **Show Estimated Function** to validate the fit of the original data.



Importing and Exporting Information from the Graphical Interface

Saving Function

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

To save the estimated function from the present estimation, use the menu option **File > Save Estimated Function**. A dialog box appears that lets you specify a folder and filename for storing the function. Type the name `fex1nsto`. After saving the function data to the file `fex1nsto.mat`, load the variables into your workspace:

```
load fex1nsto
whos
```

Name	Size	Bytes	Class
thrParams	1x5	580	cell array
wname	1x4	8	char array
xdata	1x125	1000	double array
ydata	1x125	1000	double array

The estimated function is given by `xdata` and `ydata`. The length of these vectors is equal to the number of bins you choose in step 2. In addition, the parameters of the estimation process are given by the wavelet name contained in `wname`:

```
wname

wname =
     sym4
```

and the level dependent thresholds contained in `thrParams`, which is a cell array of length 5 (the level of the decomposition). For `i` from 1 to 5, `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 Variance Adaptive Thresholding of Wavelet Coefficients" on page 2-145 in the *Wavelet Toolbox User's Guide*.

For example, for level 1,

```
thrParams{1}

ans =
```

```
    -0.4987 0.4997 1.0395
```

Loading Data

To load data for regression estimation, your file must contain at least one vector. If your file contains only one vector, this vector is considered as `ydata` and an `xdata` vector is automatically generated.

If your file contains at least two vectors, they must be called `xdata` and `ydata` or `x` and `y`.

These vectors must be the same length.

For example, load the file containing the data considered in the previous example:

```
clear
load ex1nsto
whos
```

Name	Size	Bytes	Class
x	1x500	4000	double array
y	1x500	4000	double array

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

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

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. For more information, see "Function Estimation: Density and Regression" in the *Wavelet Toolbox User's Guide*.

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

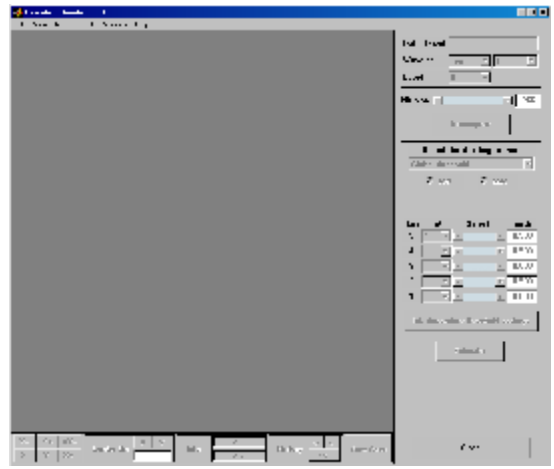
1. Start the Density Estimation 1-D Tool.

   From the MATLAB prompt, type

   ```
   wavemenu
   ```

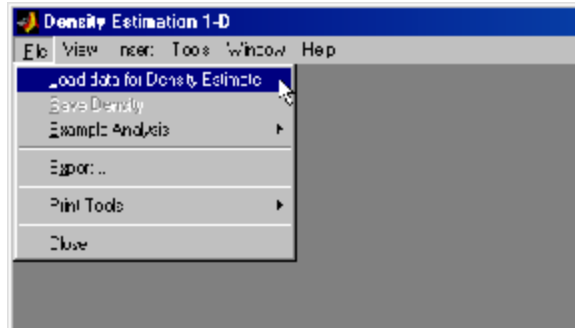
   The **Wavelet Toolbox Main Menu** appears.

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

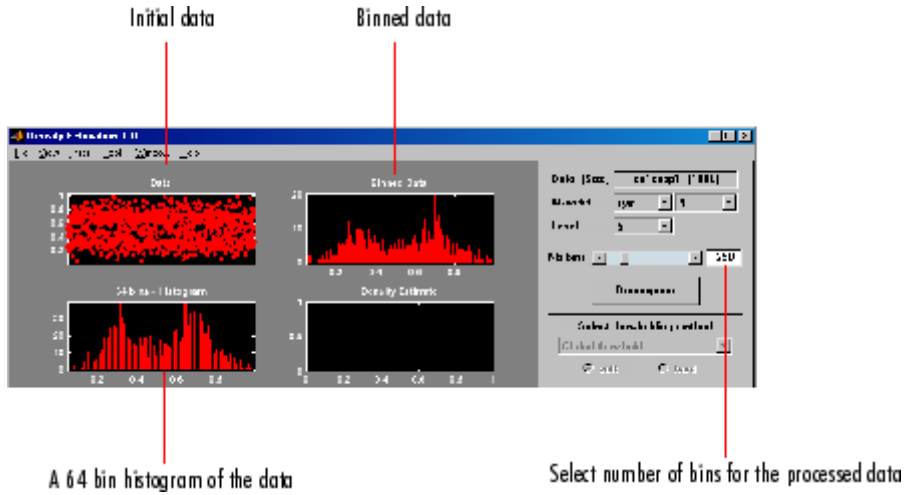


2 Load data.

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



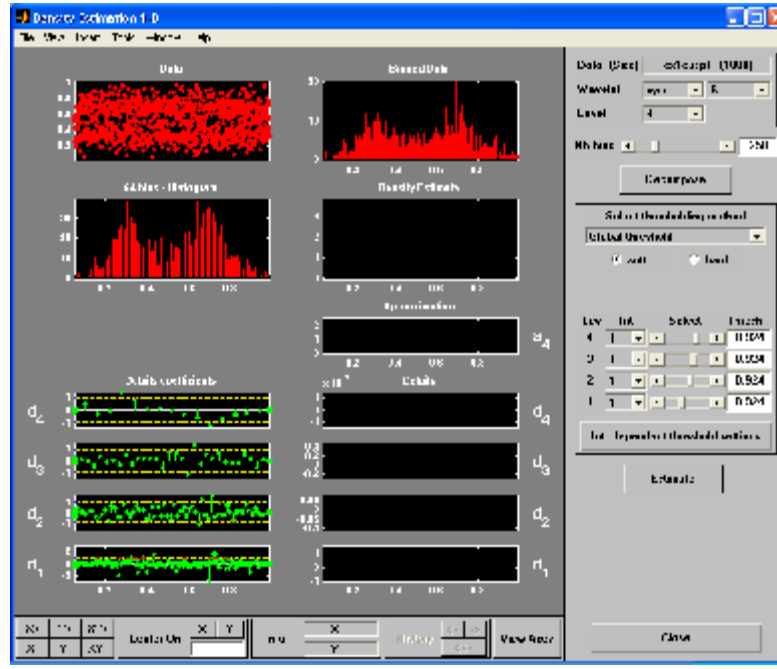
When the **Load data for Density Estimate** dialog box appears, select the demo MAT-file `ex1cusp1.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`. Click the **OK** button. The noisy cusp data is loaded into the **Density Estimation 1-D** tool.



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.

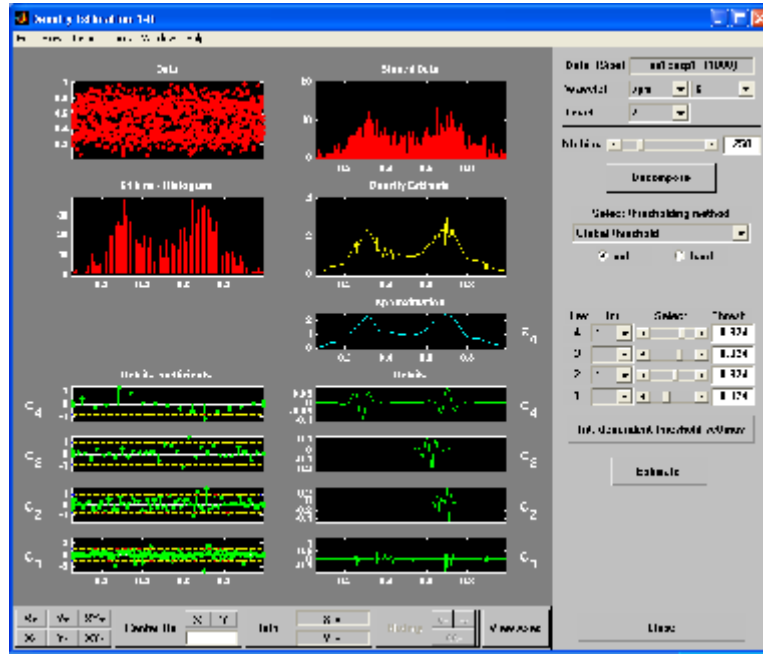
Select the `sym6` wavelet from the **Wavelet** menu and select **4** from the **Level** menu, and then click the **Decompose** button. After a pause for computation, the tool displays the detail coefficients of the decomposition of the binned data.



4 Perform a density estimation.

We'll 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.

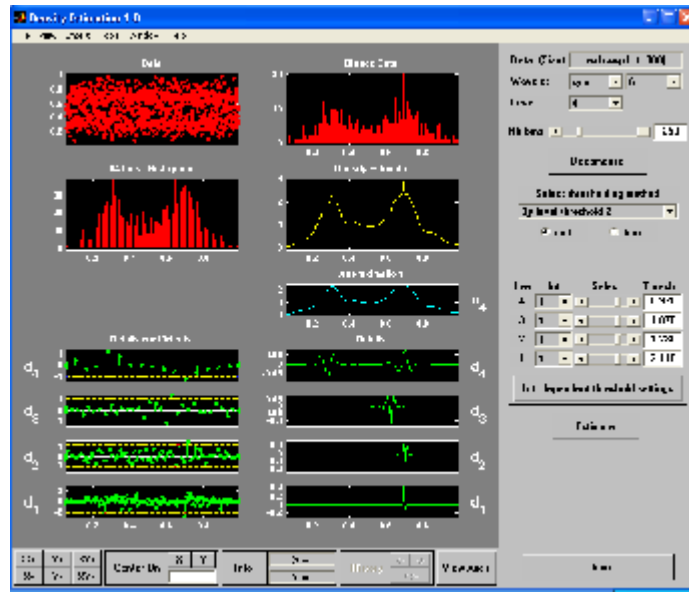


You can see that the estimation process delivers a very irregular resulting density. The density estimate (in yellow) is the normalized sum of the signals located below it: the approximation `a4` and the reconstructed details after coefficient thresholding.

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**. For more information about these methods, see "Function Estimation: Density and Regression" in the *Wavelet Toolbox User's Guide*. 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 Variance Adaptive Thresholding of Wavelet Coefficients" on page 2-145. 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')
```

One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients

This section takes you through the features of local thresholding of wavelet coefficients for one-dimensional signals or data. This capability is available through graphical interface tools throughout the Wavelet Toolbox software:

- Wavelet De-noising 1-D
- Wavelet Compression 1-D
- SWT De-noising 1-D
- Regression Estimation 1-D
- Density Estimation 1-D

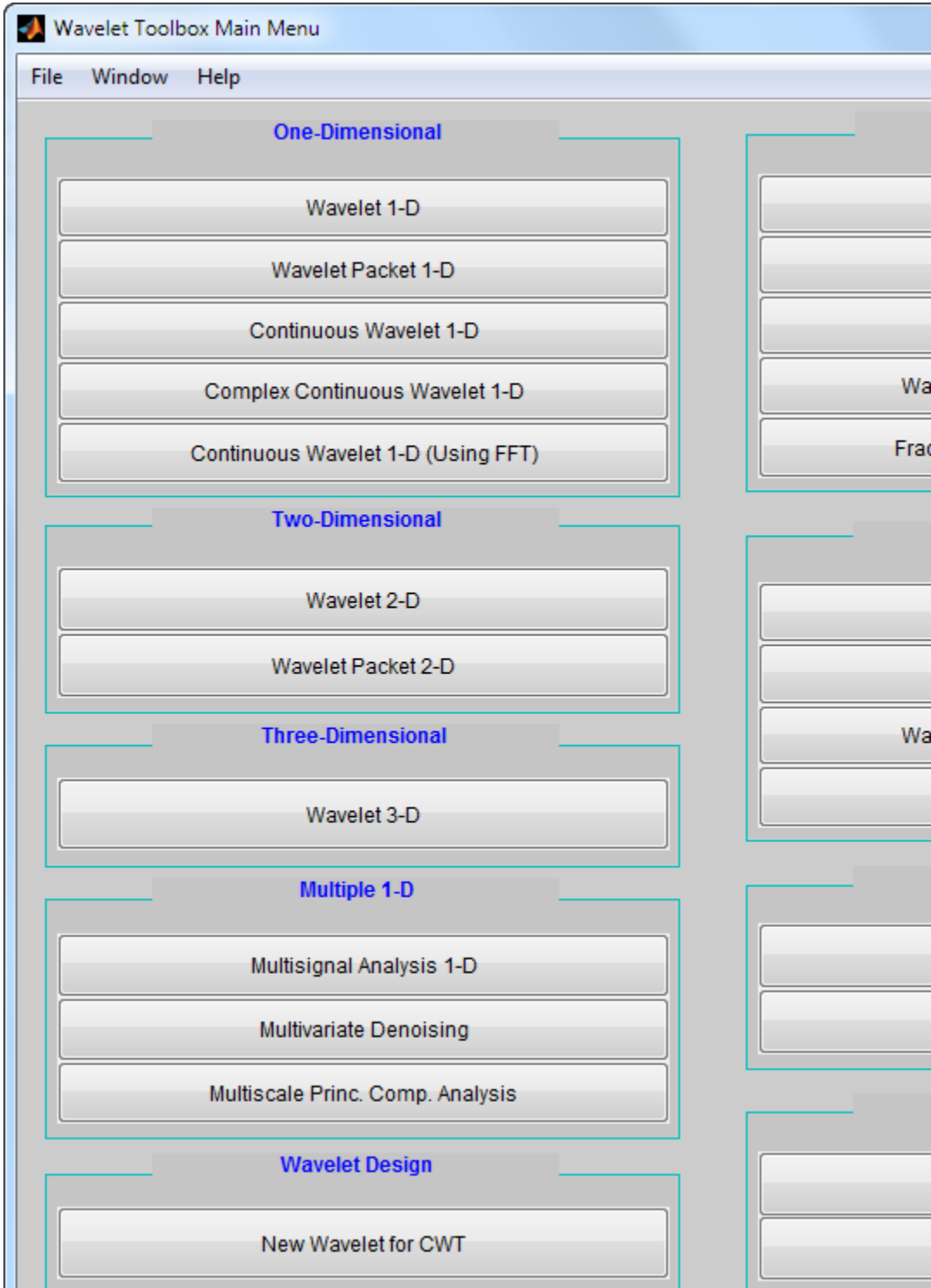
This tool allows you to define, level by level, time-dependent (x-axis-dependent) thresholds, and then increase the capability of the de-noising strategies handling nonstationary variance noise. More precisely, the model assumes that the observation is equal to the interesting signal superimposed on noise. The noise variance can vary with time. There are several different variance values on several time intervals. The values as well as the intervals are unknown. This section will use one of the graphical interface tool (**SWT De-noising 1-D**) to illustrate this capability. The behavior of all the above-mentioned tools is similar.

One-Dimensional Local Thresholding for De-Noising Using the Graphical Interface

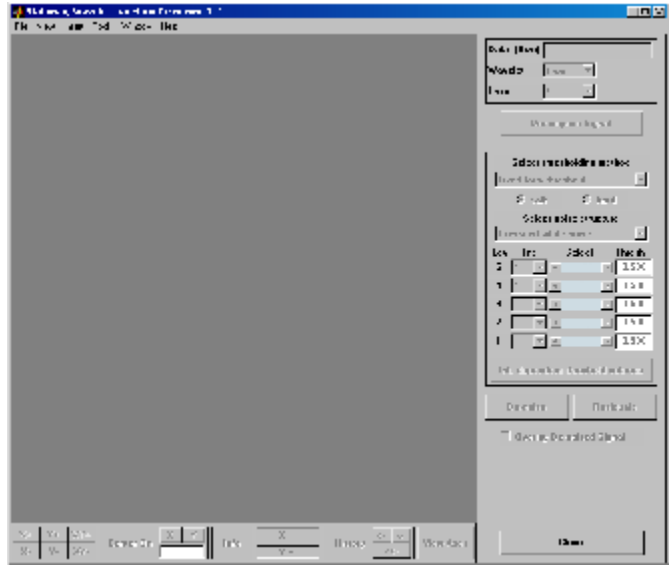
1. From the MATLAB prompt, type

   ```
   wavemenu
   ```

   The **Wavelet Toolbox Main Menu** appears.


Wavelet Toolbox Main Menu
File
Window
Help
One-Dimensional
Wavelet 1-D
Wavelet Packet 1-D
Continuous Wavelet 1-D
Complex Continuous Wavelet 1-D
Continuous Wavelet 1-D (Using FFT)
Two-Dimensional
Wavelet 2-D
Wavelet Packet 2-D
Three-Dimensional
Wavelet 3-D
Multiple 1-D
Multisignal Analysis 1-D
Multivariate Denoising
Multiscale Princ. Comp. Analysis
Wavelet Design
New Wavelet for CWT

The discrete stationary wavelet transform de-noising tool for one-dimensional signals appears.



2 Load data.

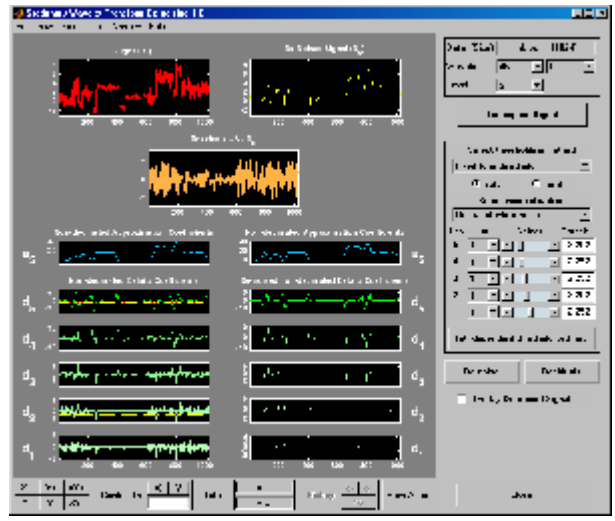
From the **File** menu, choose the **Load Signal** option.

When the **Load Signal** dialog box appears, select the MAT-file `nblocr1.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`. Click the **OK** button. The noisy blocks signal with two change points in the noise variance located at positions 200 and 600, is loaded into the **SWT De-noising 1-D** tool.

3 Perform signal decomposition.

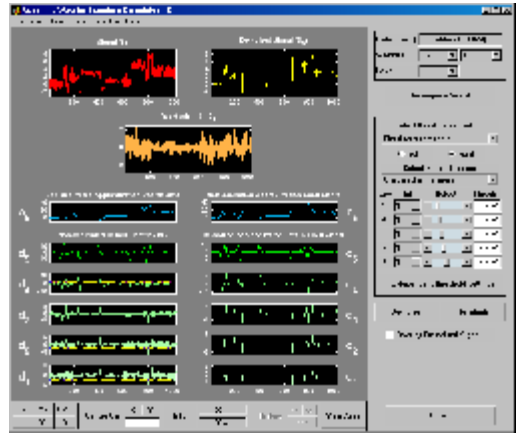
Select the `db1` wavelet from the **Wavelet** menu and select **5** from the **Level** menu, and then click the **Decompose Signal** button. After a pause for computation, the tool displays the stationary wavelet approximation and detail coefficients of the decomposition.

Accept the defaults of **Fixed form soft** thresholding and **Unscaled white noise**. Click the **denoise** button.



The result is quite satisfactory, but seems to be oversmoothed when the signal is irregular.

Select **hard** for the thresholding mode instead of **soft**, and then click the **denoise** button.

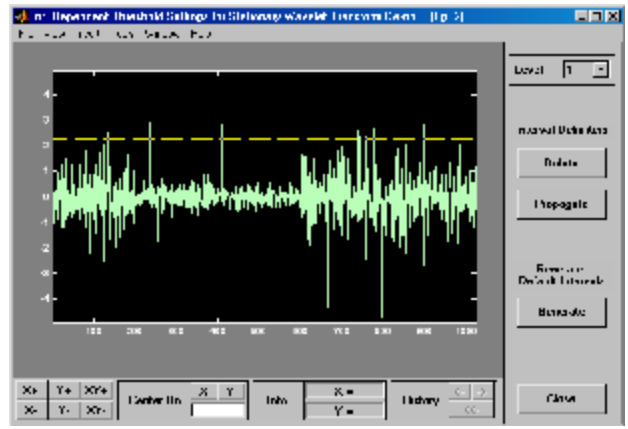


The result is not satisfactory. The denoised signal remains noisy before position 200 and after position 700. This illustrates the limits of the

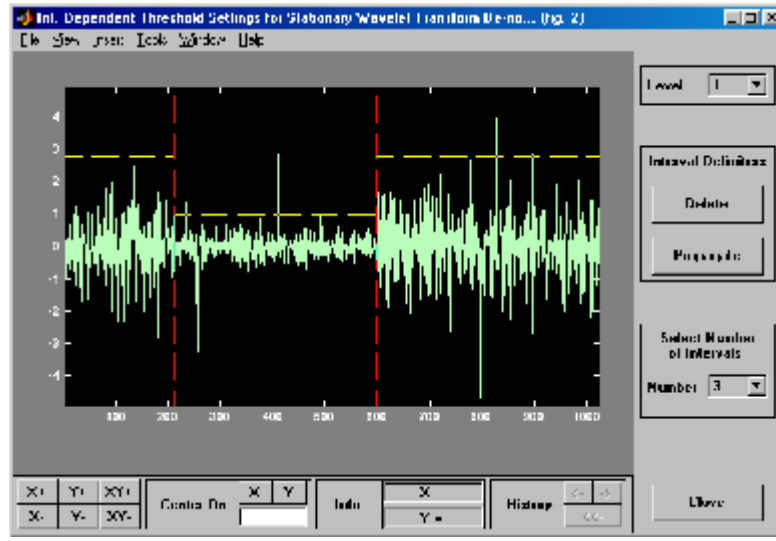
classical de-noising strategies. In addition, the residuals obtained during the last trials clearly suggest to try a local thresholding strategy.

4 Generate interval-dependent thresholds.

Click the **Int. dependent threshold Settings** button located at the bottom of the thresholding method frame. A new window titled **Int. Dependent Threshold Settings for figure ...** appears.

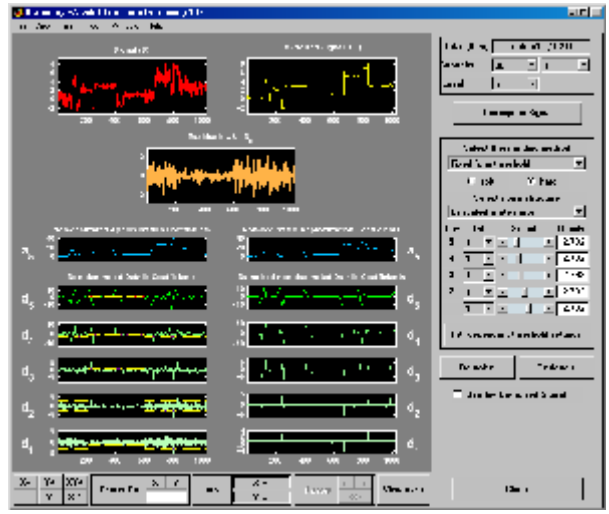


Click the **Generate** button. After a pause for computation, the tool displays the default intervals associated with adapted thresholds.



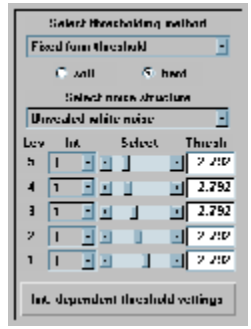
Three intervals are proposed. Since the variances for the three intervals are very different, the optimization program easily detects the correct structure. Nevertheless, you can visualize the intervals proposed for a number of intervals from 1 to 6 using the **Select Number of Intervals** menu (which replaces the **Generate** button). Using the default intervals automatically propagates the interval delimiters and associated thresholds to all levels.

denoise with Interval-Dependent Thresholds. Click the **Close** button in the **Int. Dependent Threshold Settings for ...** window. When the **Update thresholds** dialog box appears, click **Yes**. The **SWT De-noising 1-D** main window is updated. The sliders located to the right of the window control the level and interval dependent thresholds. For a given interval, the threshold is indicated by yellow dotted lines running horizontally through the graphs on the left of the window. The red dotted lines running vertically through the graphs indicate the interval delimiters. Next click the **denoise** button.



The result is quite satisfactory, but some unpleasant coefficients remain.

Modifying Interval Dependent Thresholds. The thresholds can be increased to keep only the highest values of the wavelet coefficients at each level. Do this by dragging the yellow lines directly on the graphs on the left of the window, or using the **View Axes** button (located at the bottom of the screen near the **Close** button), which allows you to see each axis in full size. Another way is to edit the thresholds by selecting the interval number located near the sliders and typing the desired value.



Note that you can also change the interval limits by holding down the left mouse button over the vertical dotted red lines, and dragging them.

You can also define your own interval dependent strategy. Click the **Int. dependent threshold settings** button. The **Int. Dependent Threshold Settings for ...** window appears again. We shall explore this window for a little while. Click the **Delete** button, so that the interval delimiters disappear. Double click the left mouse button to define new interval delimiters; for example at positions 300 and 500 and adjust the thresholds manually. Each level must be considered separately using the **Level** menu for adjusting the thresholds. The current interval delimiters can be propagated to all levels by clicking the **Propagate** button. So click the **Propagate** button. Adjust the thresholds for each level, one by one. At the end, click the **Close** button of the **Int. Dependent Threshold settings for ...** window. When the **Update thresholds** dialog box appears, click **Yes**. Then click the **denoise** button.

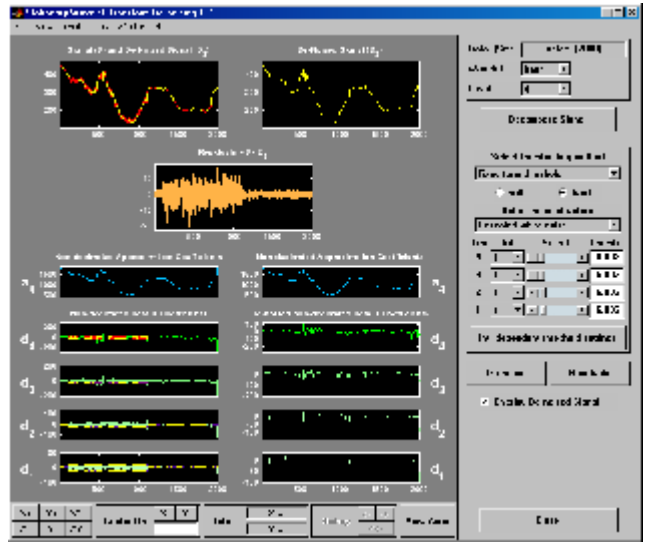
Note that

- By double-clicking again on an interval delimiter with the left mouse button, you delete it.
- You can move the interval delimiters (vertical red dotted lines) and the threshold levels (horizontal yellow dotted lines) by holding down the left mouse button over these lines and dragging them.
- The maximum number of interval delimiters at each level is 10.

Some Examples of De-noising with Interval Dependent Thresholds.

From the **File** menu, choose the **Example Analysis > Noisy Signals - Dependent Noise Variance** option. The proposed items contain, in addition to the usual information, the "true" number of intervals. You can then experiment with various signals for which local thresholding is needed.

For example, choose the last item, which is a real-world electrical signal. The entire process performed on steps **4**, **5**, **8**, **9**, and **10** is demonstrated.



Importing and Exporting Information from the Graphical Interface

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

To save the denoised signal from the present de-noising process, use the menu option **File > Save denoised Signal**. A dialog box appears that lets you specify a folder and filename for storing the signal. Type the name `dnelec`. After saving the signal data to the file `dnelec.mat`, load the variables into your workspace:

```
load dnelec
whos
```

Name	Size	Bytes	Class
`dnelec`	`1x2000`	`16000`	`double array`
`thrParams`	`1x4`	`656`	`cell array`
`wname`	`1x4`	`8`	`char array`

The denoised signal is given by `dnelec`. In addition, the parameters of the de-noising process are given by the wavelet name contained in wname:

```
wname

wname =
      haar
```

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}` is an array `nbintx3` (where `nbint` is the number of intervals, here 3), and each row contains the lower and upper bounds of the interval of thresholding and the threshold value. For example, for level 1,

```
thrParams{1}
ans =
    1.0e+03 *

    0.0010 0.0980 0.0060
    0.0980 1.1240 0.0204
    1.1240 2.0000 0.0049
```

One-Dimensional Selection of Wavelet Coefficients Using the Graphical Interface

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')
```

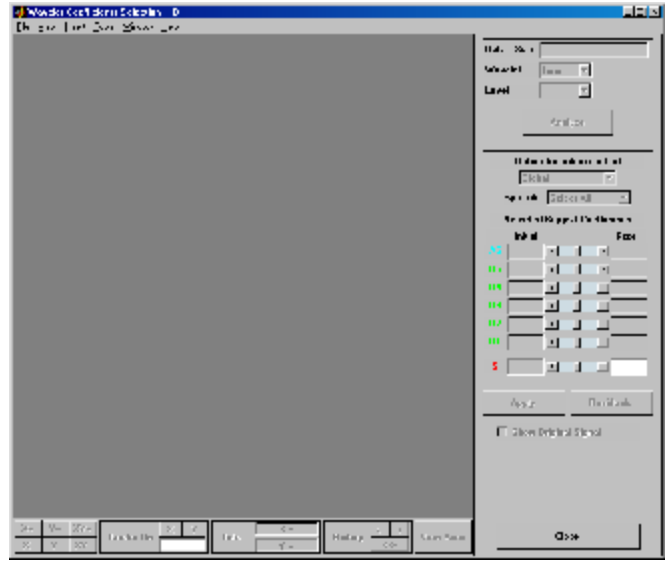
1. Start the Wavelet Coefficients Selection 1-D Tool.

   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.



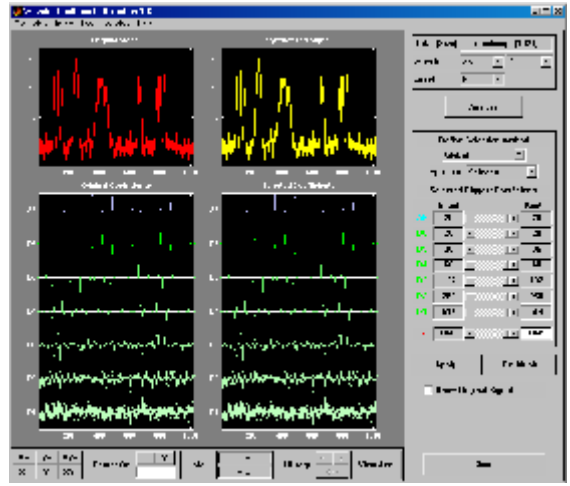
2 Load data.

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

When the **Load Signal** dialog box appears, select the demo 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.

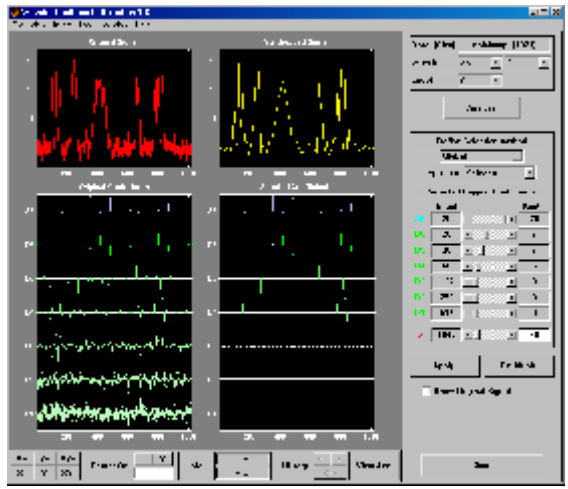


The tool displays below the original signal (on the left) its wavelet decomposition: the approximation coefficients `A6` and detail coefficients from `D6` at the top to `D1` at the bottom. In the middle of the window, below the synthesized signal (which at this step is the same, since all the wavelet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

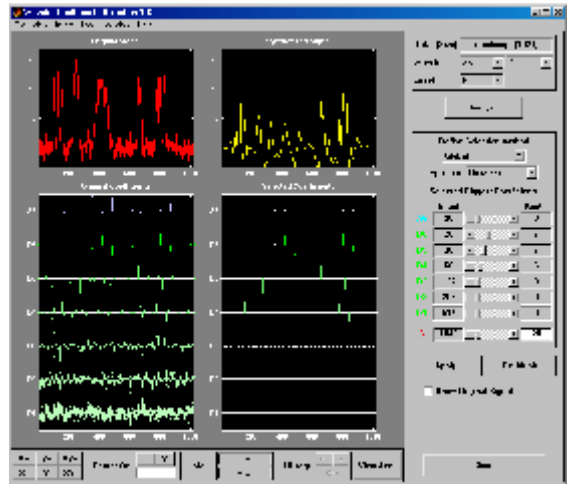
Selecting Biggest Coefficients Globally

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.

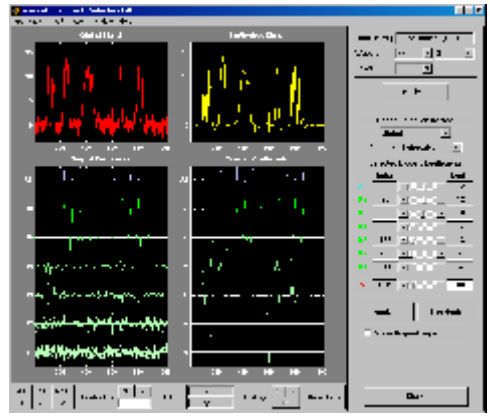


In the previous trial, the approximation coefficients were all kept. It is possible to relax this constraint by selecting another option from the **App. cfs** menu (Approximation Coefficients abbreviation). Choose the **Unselect** option and click the **Apply** button.



None of the approximation coefficients are kept.

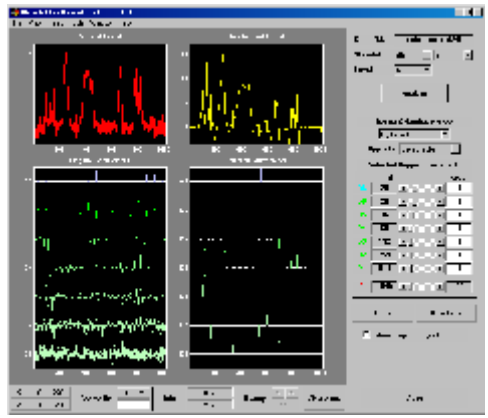
From the **App. cfs** menu, select the **Selectable** option. Type 80 for the number of selected biggest coefficients and press **Enter**. Then, click the **Apply** button.



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

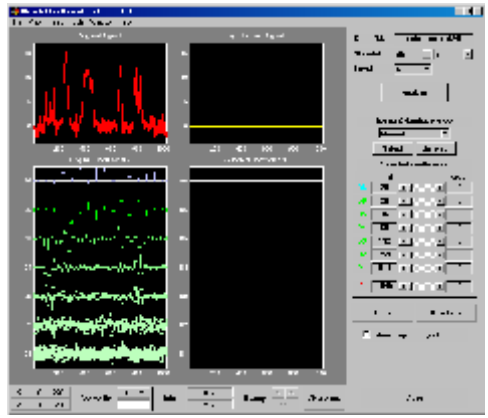
Selecting Biggest Coefficients by Level

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.

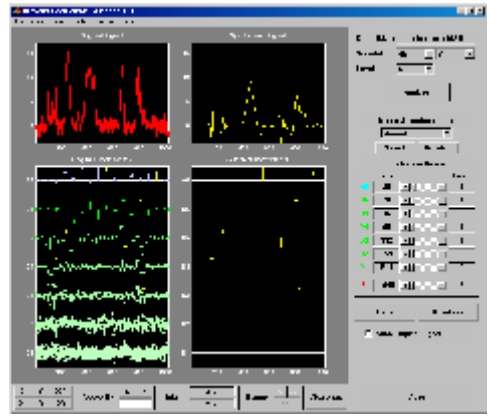


Selecting Coefficients Manually

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.



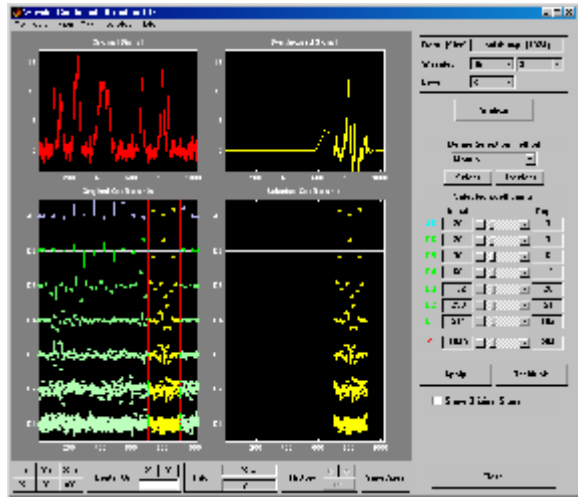
Select seven coefficients individually by double clicking each of them using the left mouse button. The color of selected coefficients switches from green to yellow for the details and from blue to yellow for the approximation, which appear on the left of the window and appear in yellow on the middle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

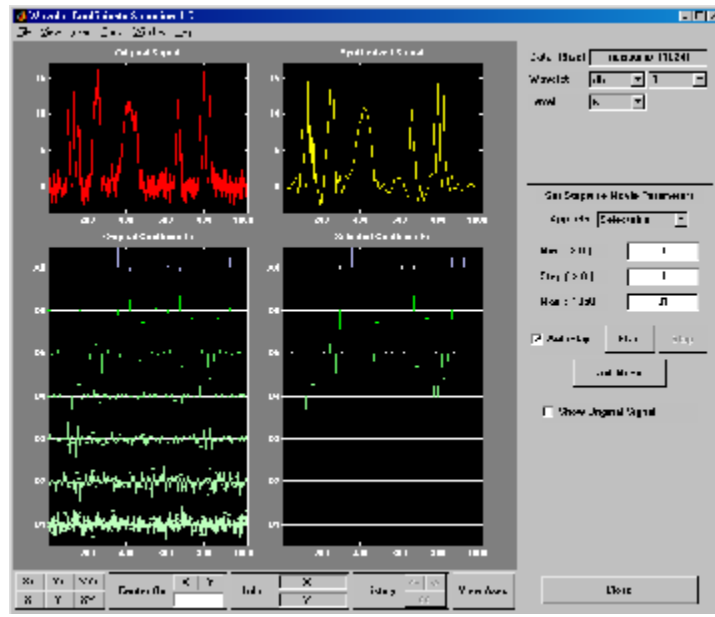


This illustrates that wavelet analysis is a local analysis since the signal is perfectly reconstructed around the position 800. Check the **Show Original Signal** to magnify it.

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')
```

Two-Dimensional Selection of Wavelet Coefficients Using the Graphical Interface

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')
```

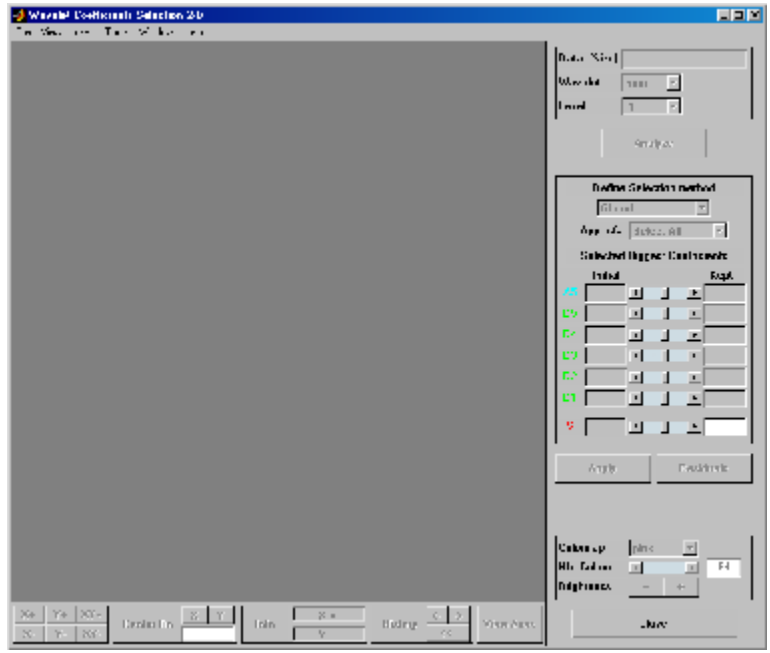
1 Start the Wavelet Coefficients Selection 2-D Tool.

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.



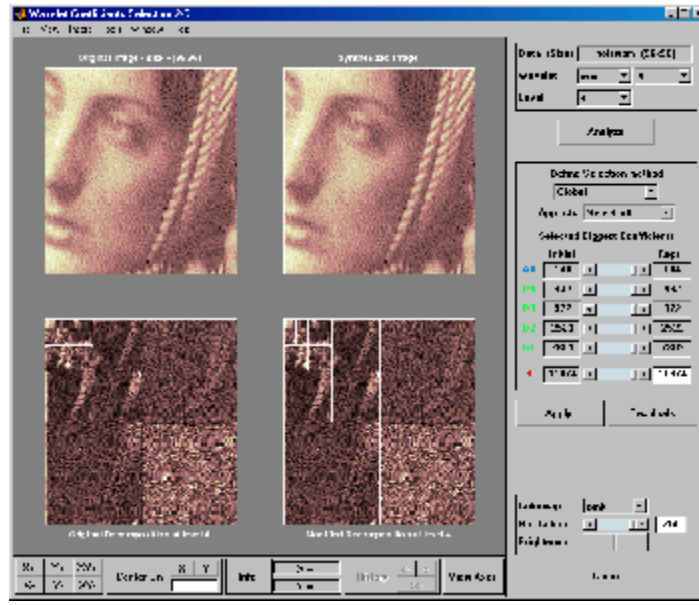
2 Load data.

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

When the **Load Image** dialog box appears, select the demo MAT-file `noiswom.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`. Click the **OK** button. The noisy woman data is loaded into the **Wavelet Coefficients Selection 2-D** tool.

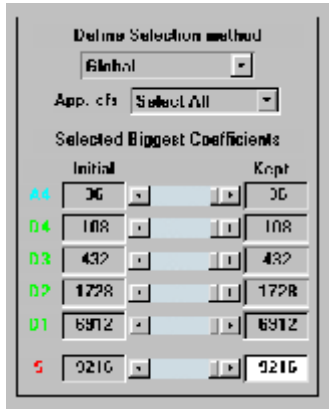
3 Perform a Wavelet Decomposition.

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.

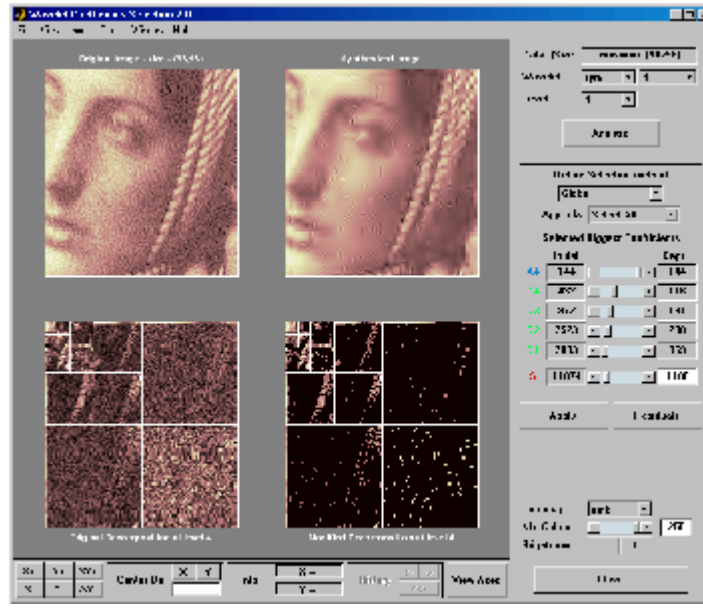
Note The difference between 9216 and 11874 comes from the extra coefficients generated by the redundant DWT using the current extension mode (symmetric, `'sym'`). Let us mention that since 96 is divisible evenly into $2^4 = 16$, using the periodic extension mode (`'per'`) for the DWT, you obtain for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



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).

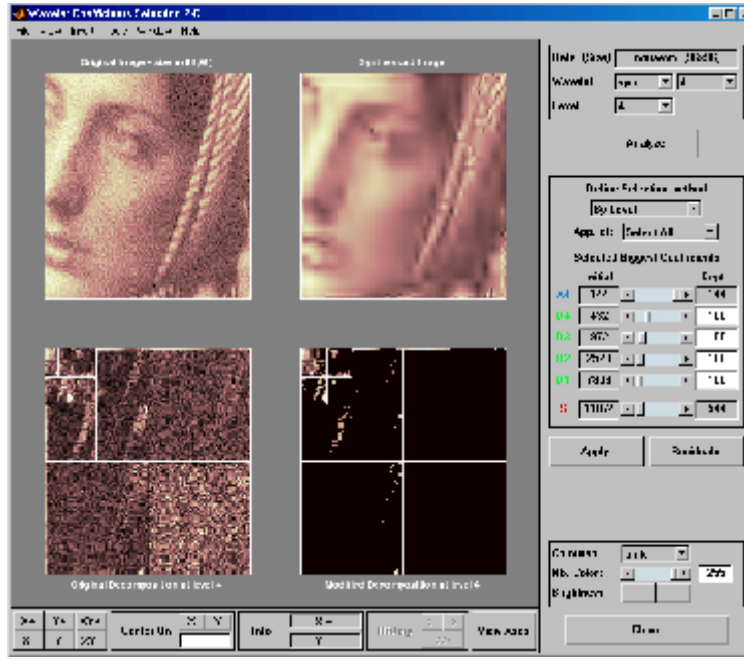
Then click the **Apply** button.



In the previous trial, the approximation coefficients were all kept. It is possible to relax this constraint by selecting another option from the **App. cfs** menu (see “One-Dimensional Selection of Wavelet Coefficients Using the Graphical Interface” on page 2-155).

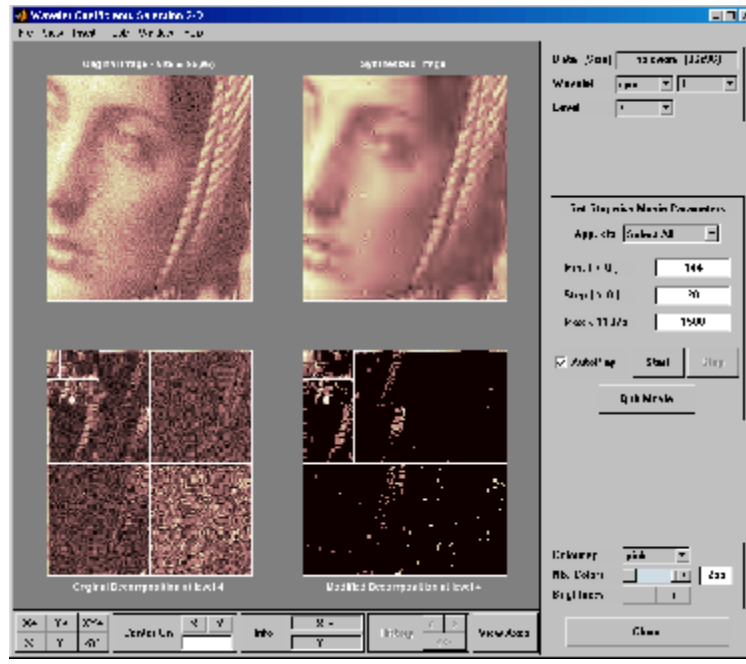
Selecting Biggest Coefficients by Level

Selecting Biggest Coefficients by LevelFrom 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
```

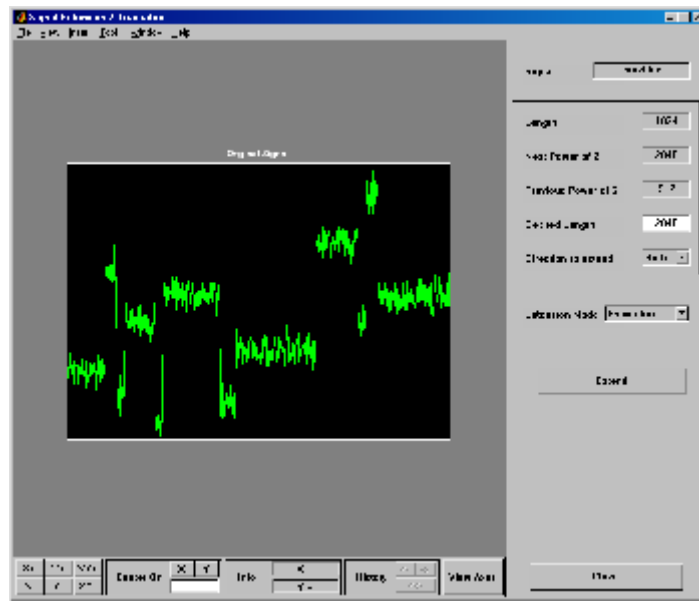
The **Wavelet Toolbox Main Menu** appears.

Click the **Signal Extension** menu item.

2 Load data.

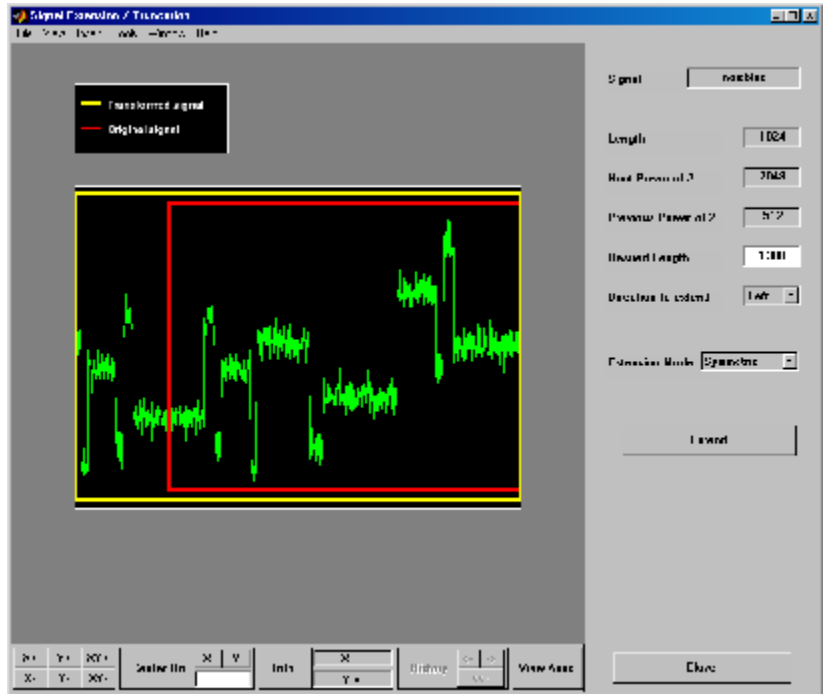
From the **File** menu, choose the **Load Signal** option.

When the **Load Signal** dialog box appears, select the demo 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.



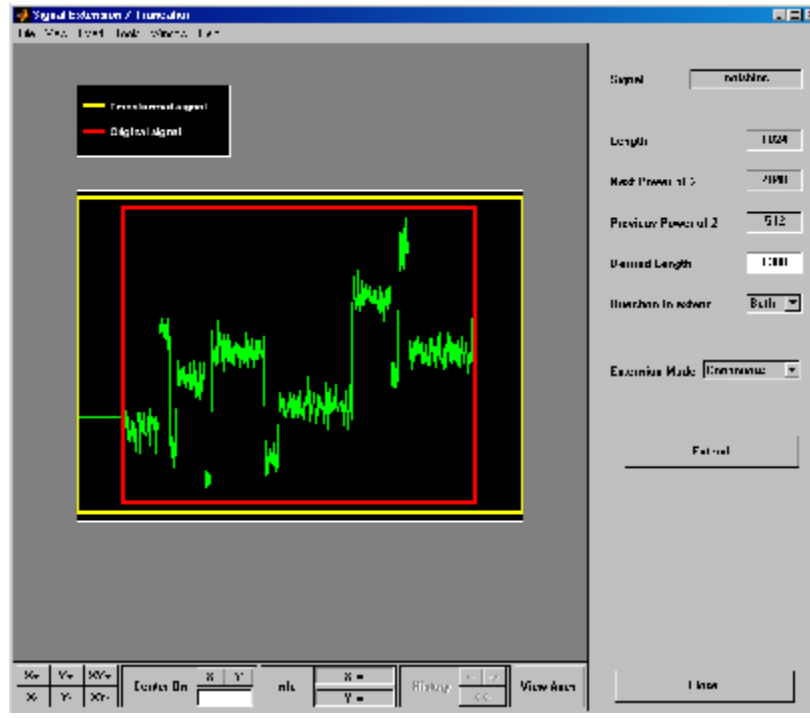
3 Extend the signal.

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.

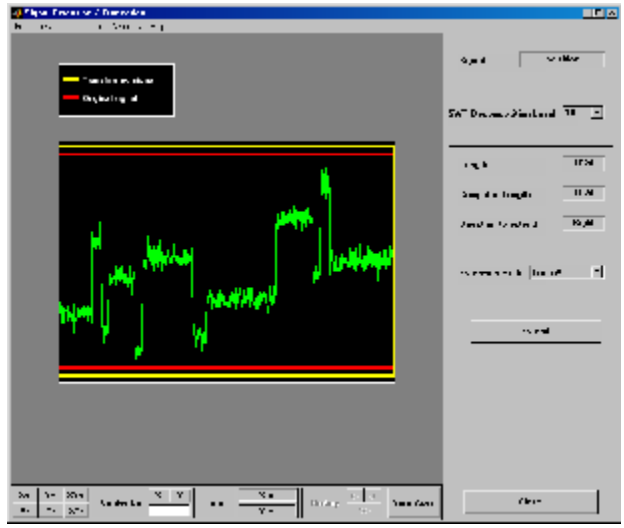


The signal is extended in both directions by replicating the first value to the left and the last value to the right, respectively.

Extending Signal for SWT

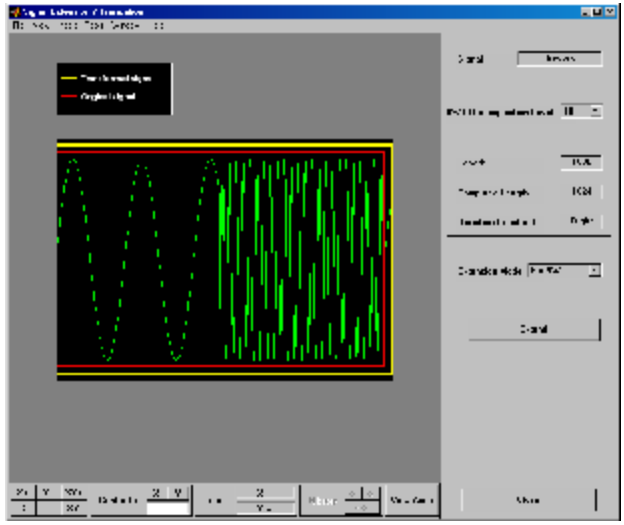
Since the decomposition at level `k` of a signal using SWT requires that `2^k` divides evenly into the length of the signal, the tool provides a special option dedicated to this kind of extension.

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.

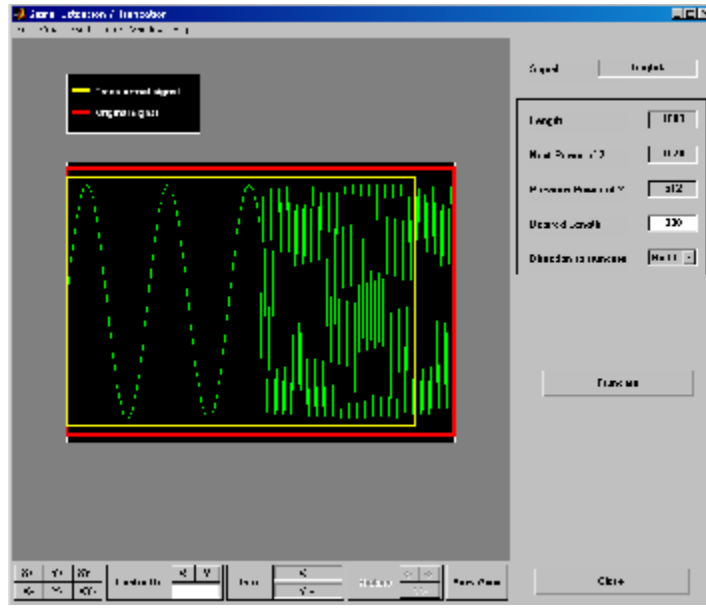
Select **4** from the **SWT Decomposition Level** menu, and then click the **Extend** button. The tool performs a minimal right periodic extension leading to an extended signal of length 1008 (because 1008 is the smallest integer greater than 1000 divisible by 2^4 = 16).

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 2-172.

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

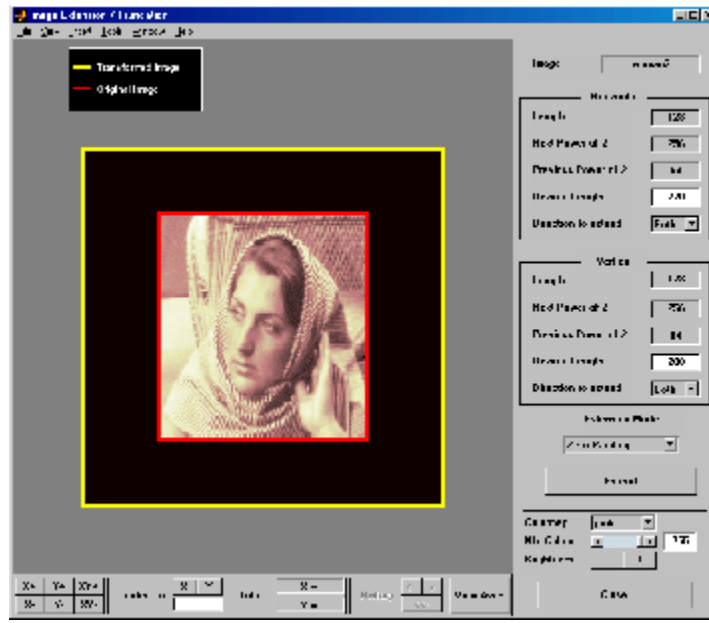
```
wavemenu
```

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 2-172.

To see some more extension cases, look at the general demos 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").

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,

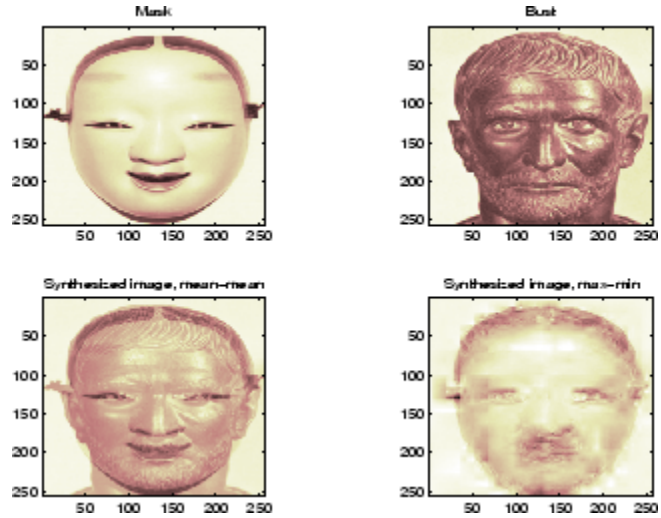
```
XFUSmean = wfusimg(X1,X2,'db2',5,'mean','mean');
```

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.

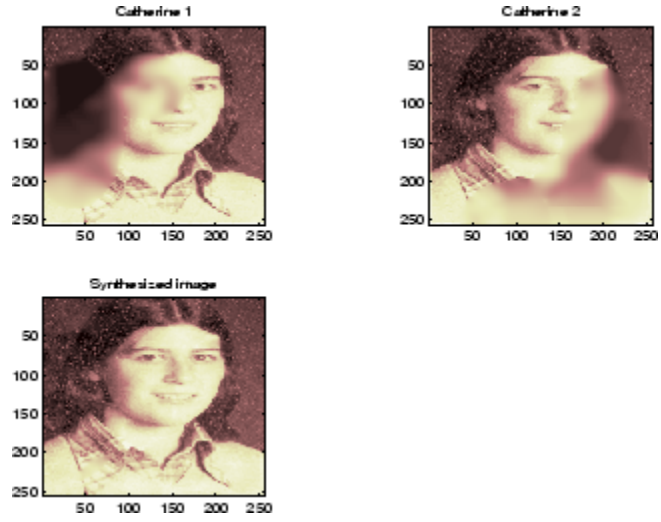
```
load cathe_1; X1 = X;
load cathe_2; X2 = X;
```

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

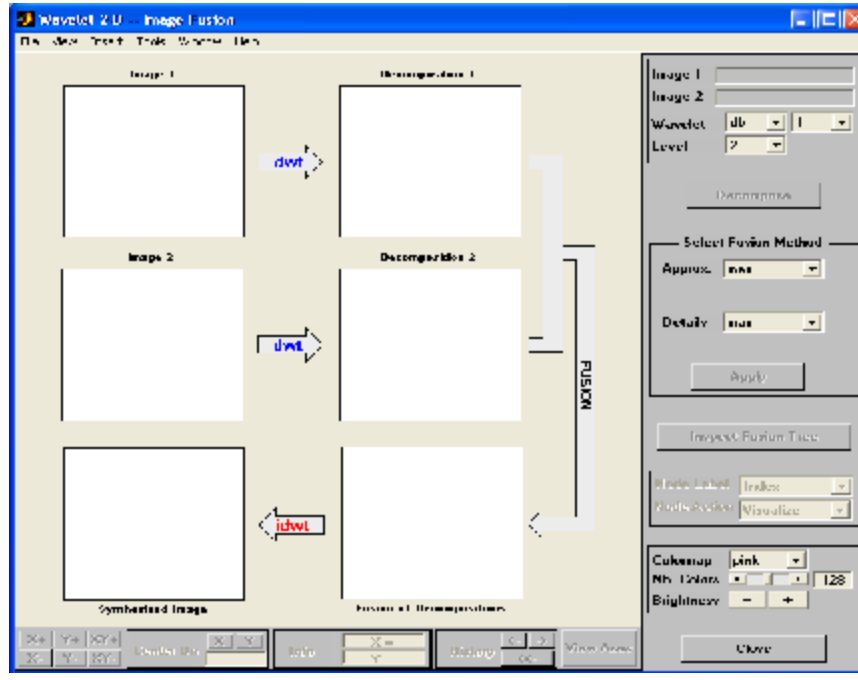
1 Start the Image Fusion Tool.

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.


Wavelet Toolbox Main Menu
File Window Help
One-Dimensional
Wavelet 1-D
Wavelet Packet 1-D
Continuous Wavelet 1-D
Complex Continuous Wavelet 1-D
Continuous Wavelet 1-D (Using FFT)
Two-Dimensional
Wavelet 2-D
Wavelet Packet 2-D
Three-Dimensional
Wavelet 3-D
Multiple 1-D
Multisignal Analysis 1-D
Multivariate Denoising
Multiscale Princ. Comp. Analysis
Wavelet Design
New Wavelet for CWT
Specialized Tools 1-D
SWT Denoising 1-D
Density Estimation 1-D
Regression Estimation 1-D
Wavelet Coefficients Selection 1-D
Fractional Brownian Generation 1-D
Specialized Tools 2-D
True Compression 2-D
SWT Denoising 2-D
Wavelet Coefficients Selection 2-D
Image Fusion
Display
Wavelet Display
Wavelet Packet Display
Extension
Signal Extension
Image Extension
Close


dwt
dwt
idwt
FUSION
Image 1
Image 2
Wavelet
Level
Select Fusion Method
Approx.
Detail
Apply
Colormap
Brightness
Close

2 Load original images.

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

When the **Load Image 1** dialog box appears, select the demo MAT-file `mask.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`. Click the **OK** button.

Perform the same sequence choosing the **Load Image 2** option and selecting the demo MAT-file `bust.mat`.

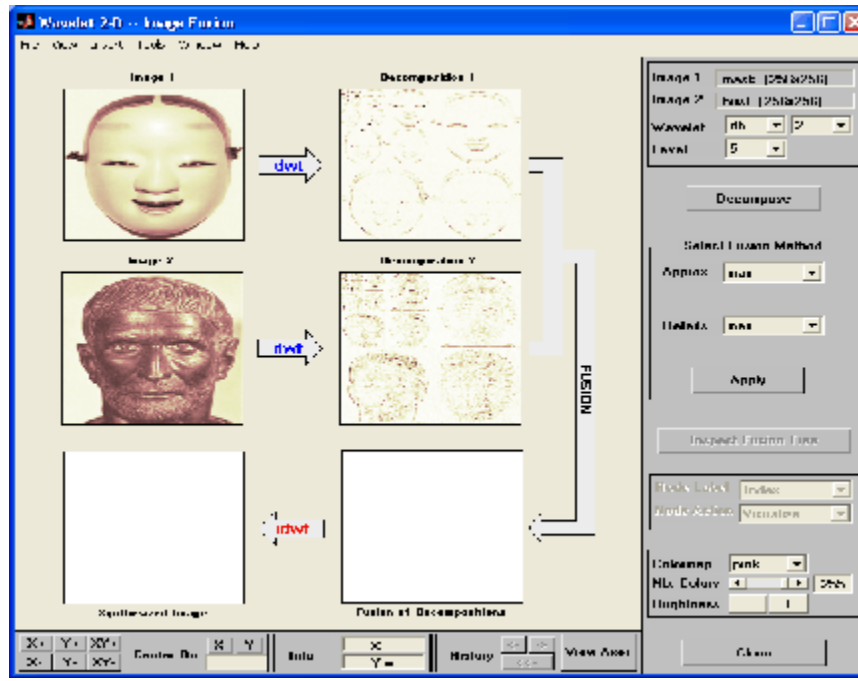
3 Perform wavelet decompositions.

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.

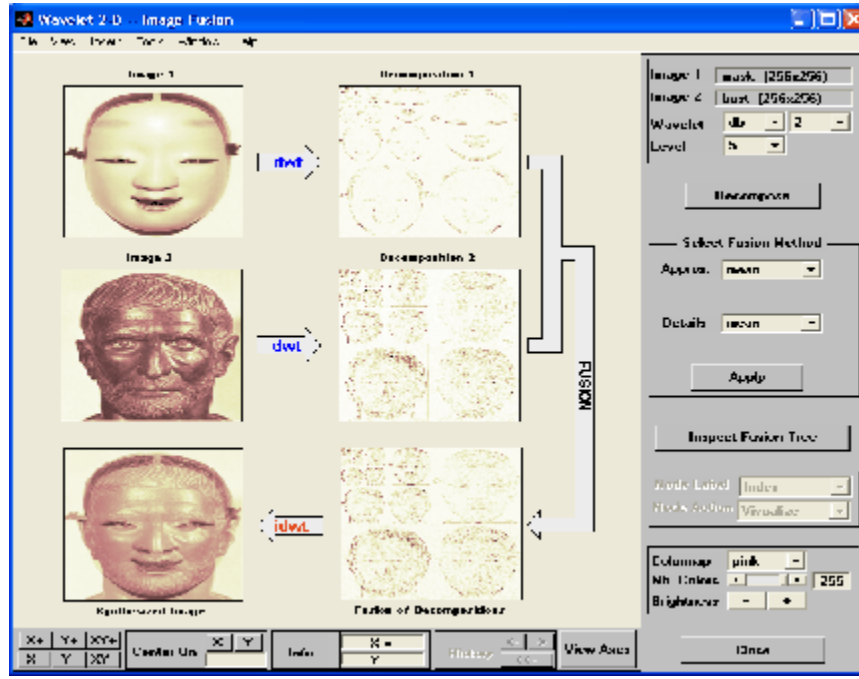
Click the **Decompose** button.

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.

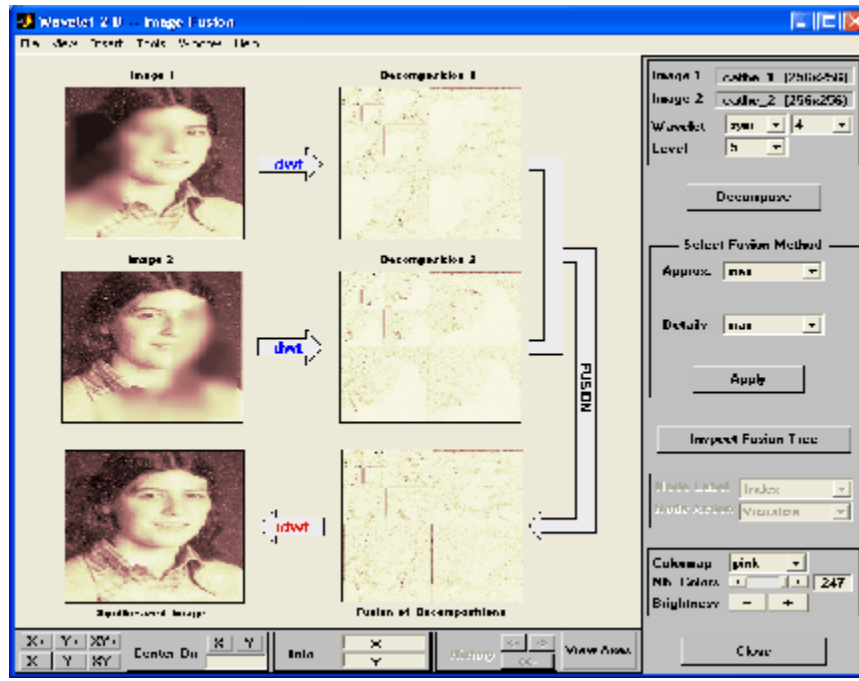
Let us now examine another example illustrating restoration using image fusion.

5 Restore the image using image fusion.

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

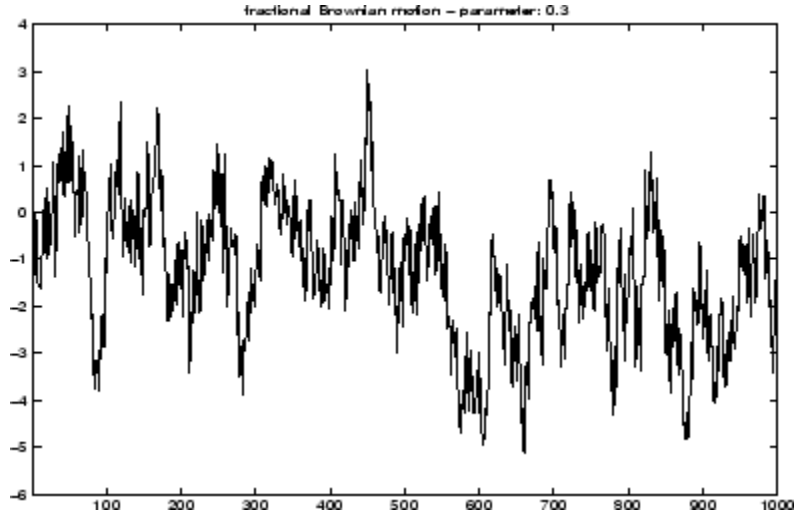
where `v` is a positive constant.

Fractional Brownian Motion Synthesis Using the Command Line

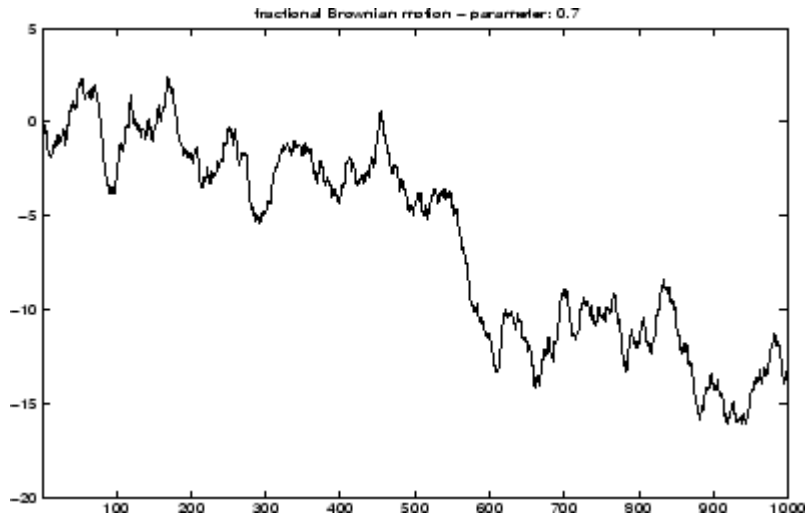
According to the value of `H`, the `fBm` exhibits for `H > 0.5`, long-range dependence and for `H < 0.5`, short or intermediate dependence.

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

% Initialize the randn generator
randn('state',1)

% 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');
```




```
% Reset randn generator and parameter H
randn('state',1); H = 0.7;
% 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.

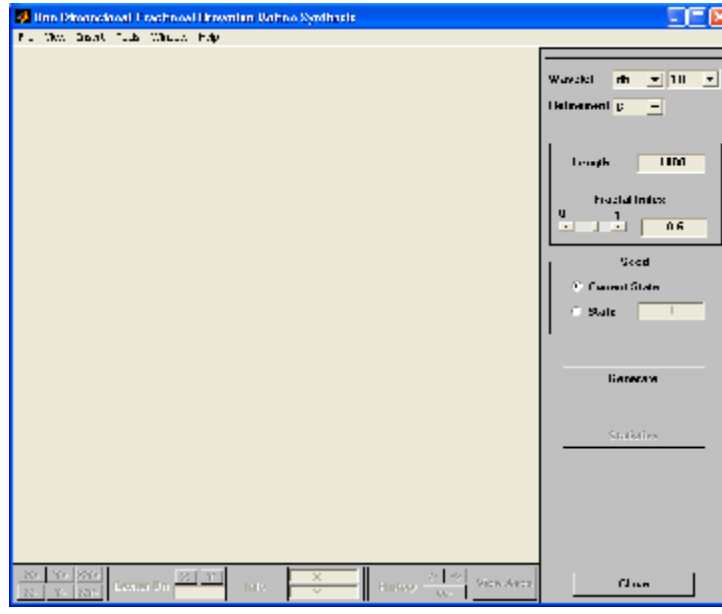
Fractional Brownian Motion Synthesis Using the Graphical Interface

1 Start the Fractional Brownian Motion Synthesis Tool.

From the MATLAB prompt, type

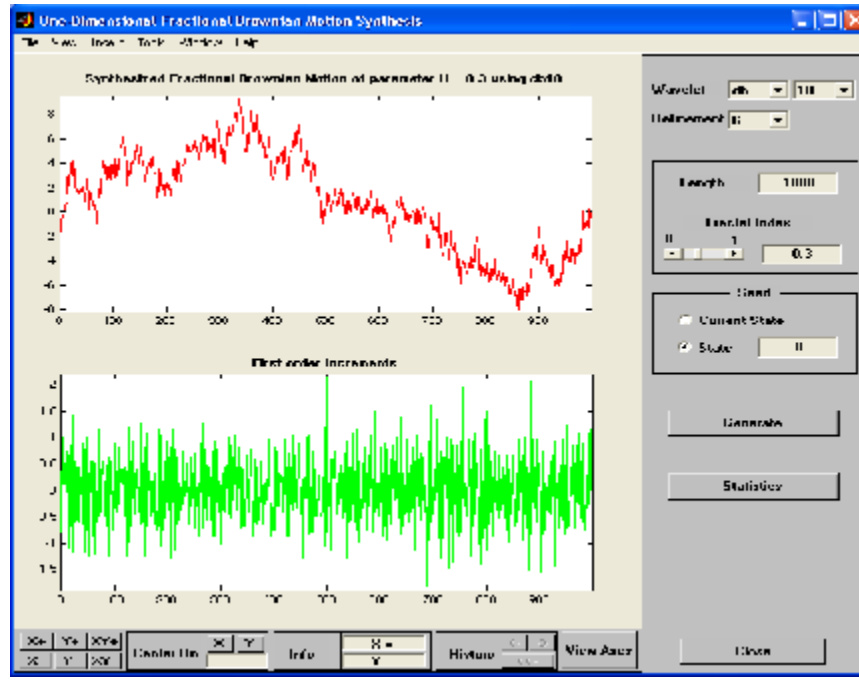
```
wavemenu
```

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



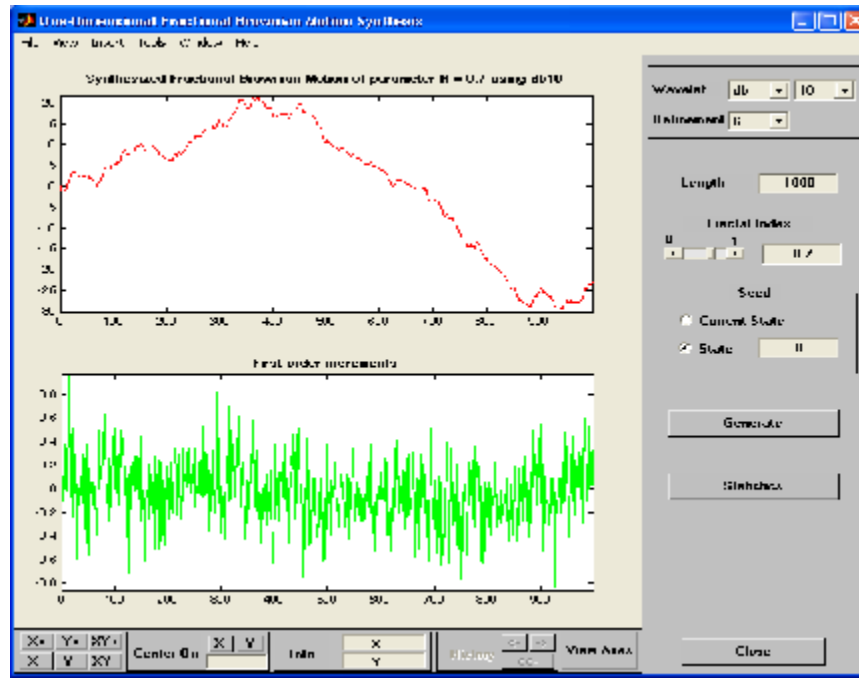
2 Generate `fBm`.

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").

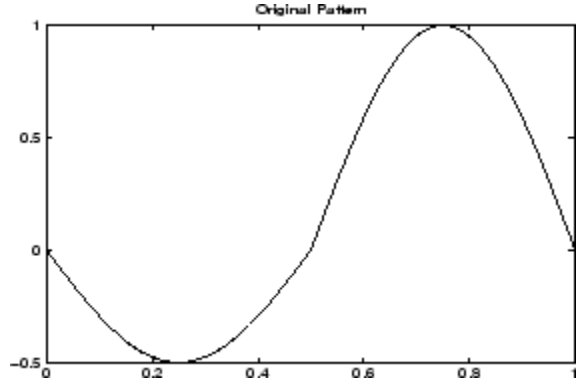
New Wavelet for CWT Using the Command Line

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
```

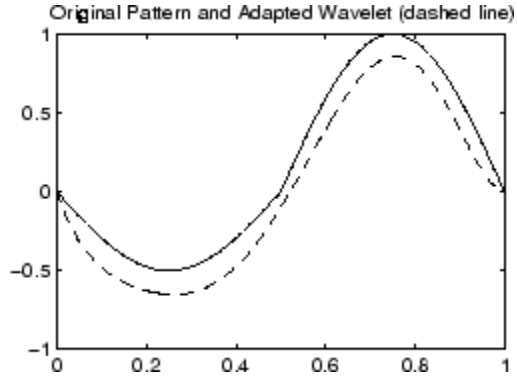
Name	Size	Bytes	Class
IntVAL	1x1	8	double array
X	1x256	2048	double array
Y	1x256	2048	double array
caption	1x35	70	char array

```
% This example is a demo-example, so we have the value of the
% integral of the pattern as well as the details about its
% construction in the caption variable.

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')
```




```
% 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)')
```




```
% 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.

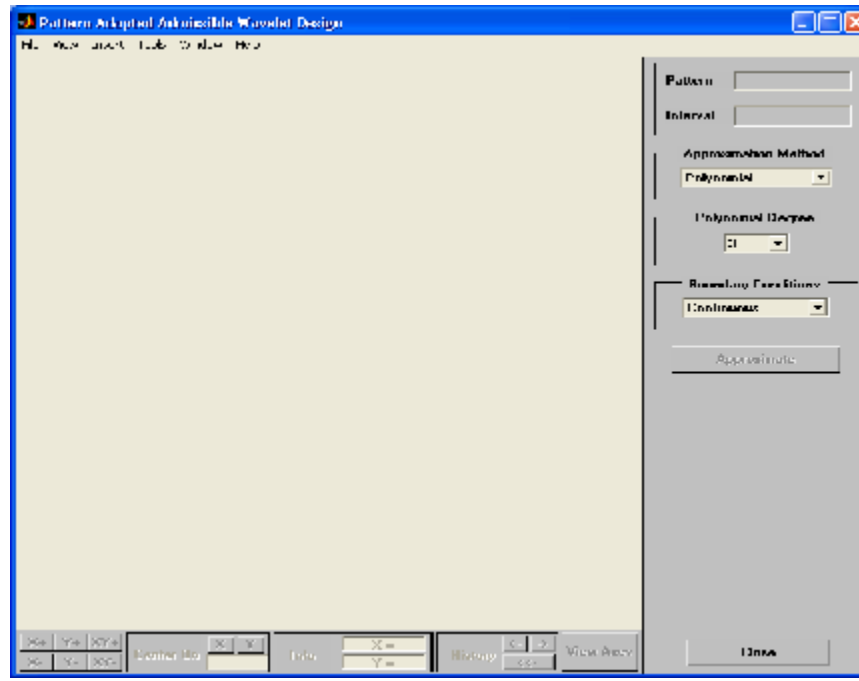
From the MATLAB prompt, type

```
wavemenu
```

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


Wavelet Toolbox Main Menu
File Window Help
One-Dimensional
Wavelet 1-D
Wavelet Packet 1-D
Continuous Wavelet 1-D
Complex Continuous Wavelet 1-D
Continuous Wavelet 1-D (Using FFT)
Two-Dimensional
Wavelet 2-D
Wavelet Packet 2-D
Three-Dimensional
Wavelet 3-D
Multiple 1-D
Multisignal Analysis 1-D
Multivariate Denoising
Multiscale Princ. Comp. Analysis
Wavelet Design
New Wavelet for CWT
Specialized Tools 1-D
SWT Denoising 1-D
Density Estimation 1-D
Regression Estimation 1-D
Wavelet Coefficients Selection 1-D
Fractional Brownian Generation 1-D
Specialized Tools 2-D
True Compression 2-D
SWT Denoising 2-D
Wavelet Coefficients Selection 2-D
Image Fusion
Display
Wavelet Display
Wavelet Packet Display
Extension
Signal Extension
Image Extension
Close

.

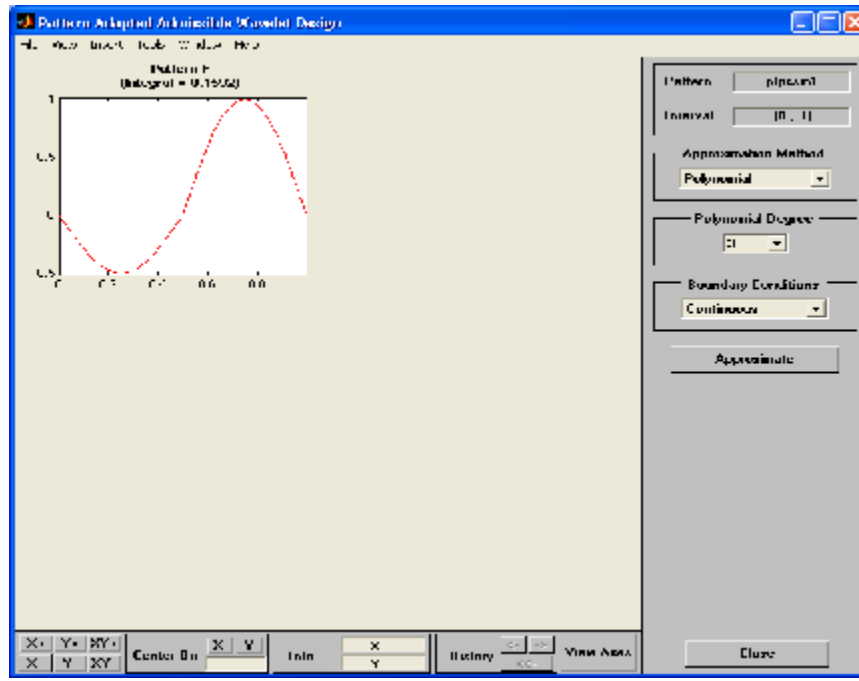


2 Load the original pattern.

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 demo MAT-file `ptpssin1.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`. Click the **OK** button.

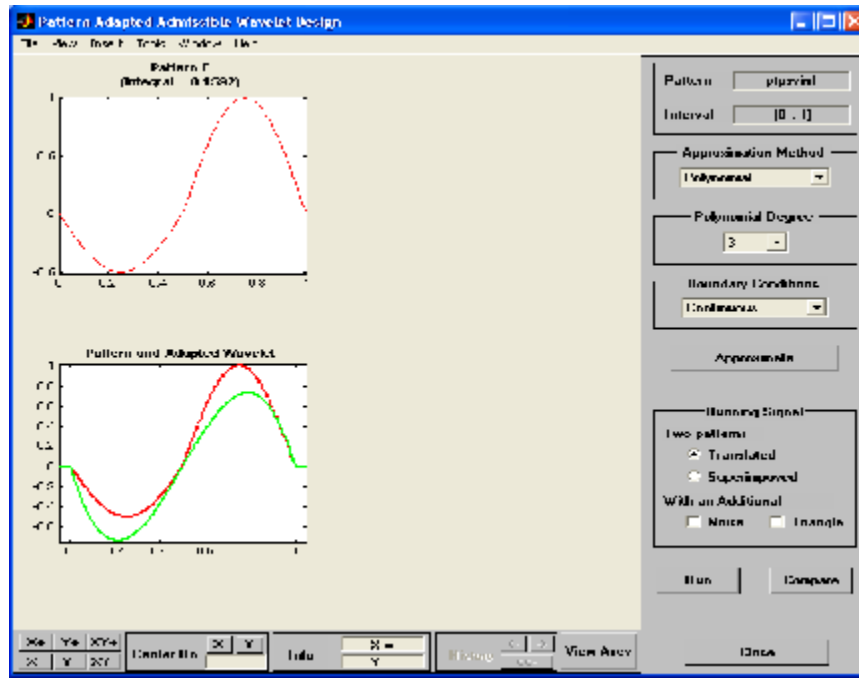


The selected pattern denoted by `F` is defined on the interval `[0,1]` and is of integral 0.1592. It is not a wavelet, but it is a good candidate because it oscillates like a wavelet.

4 Perform pattern approximation.

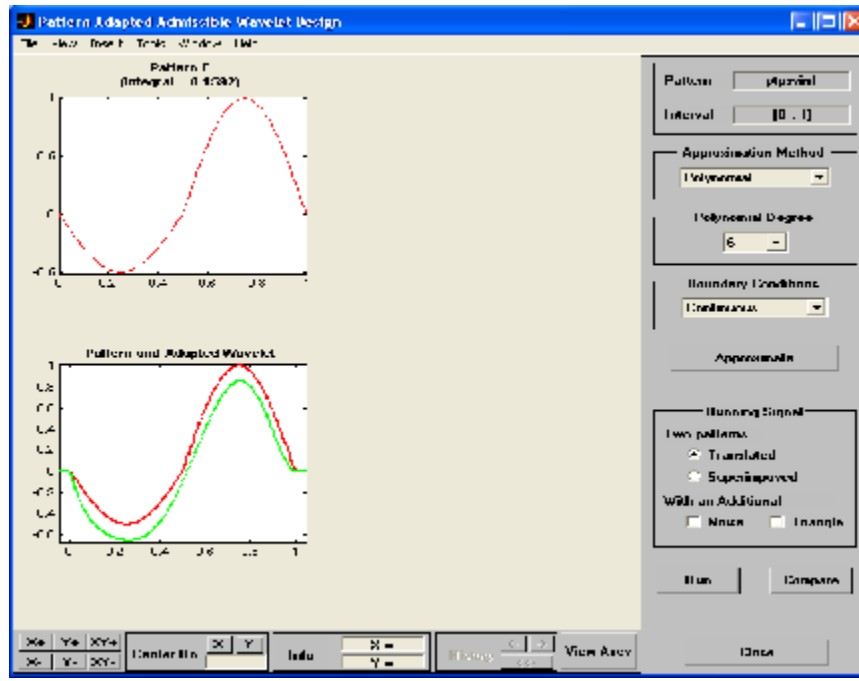
Accept the default parameters leading to use a polynomial of degree 3 with constraints of continuity at the borders 0 and 1, to approximate the pattern `F`. Click the **Approximate** button.

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.

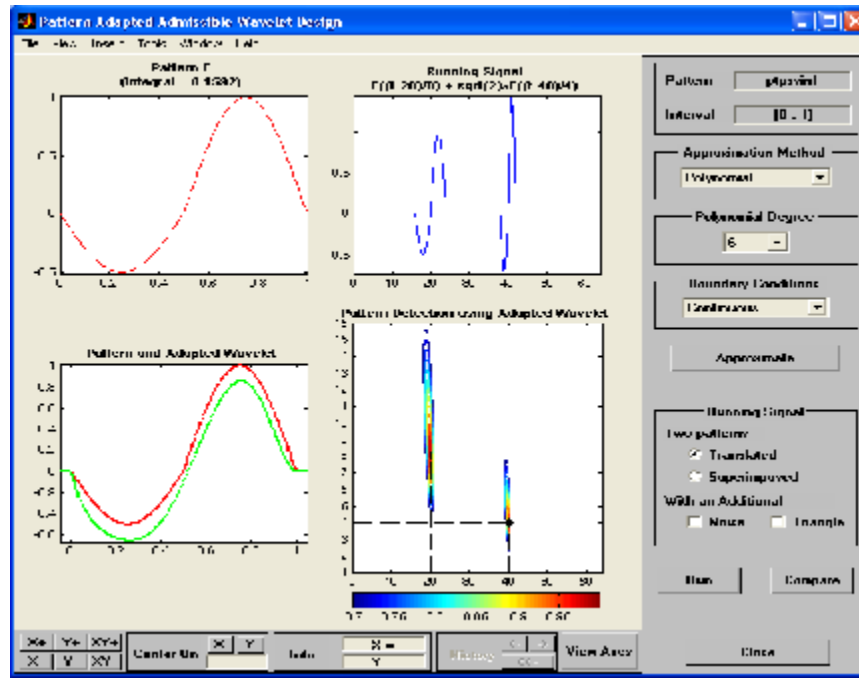


The result is now of good quality and can be used for pattern detection.

6 Pattern detection using the new wavelet.

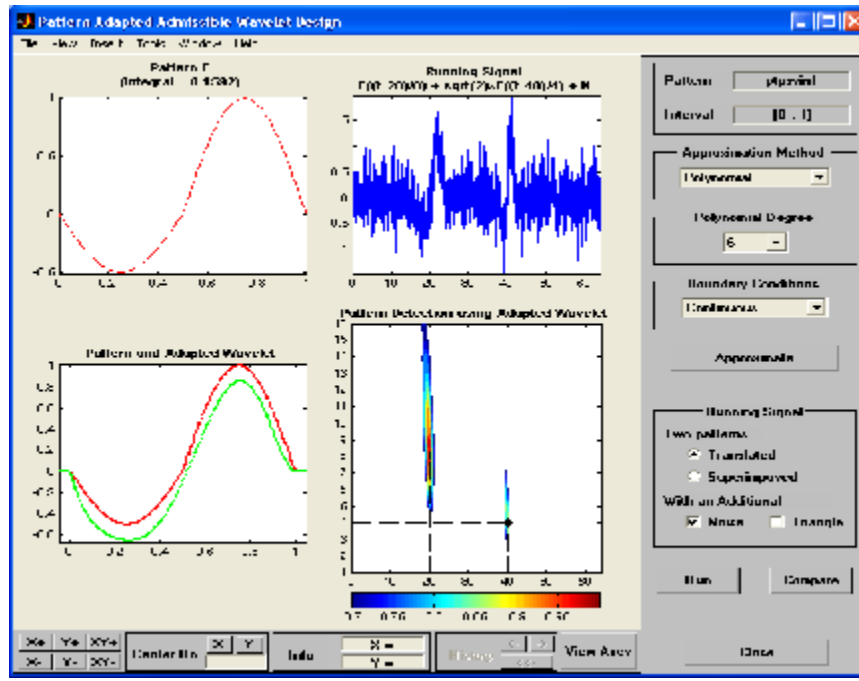
Click the **Run** button.

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.

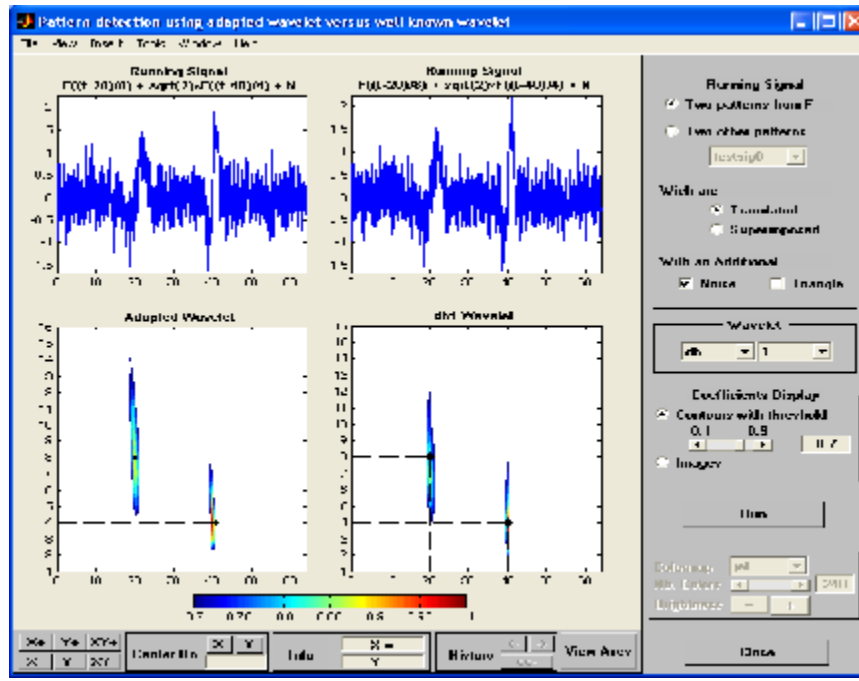
7 Using the **Running signal** frame, select the **Noise** check box to add an additive noise to the previous signal. Click the **Run** button again.



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.

Multivariate Wavelet De-Noising

This section demonstrates the features of multivariate de-noising provided in the Wavelet Toolbox software. The toolbox includes the `wmulden` function and a graphical user interface (GUI) tool available from `wavemenu`. This section also describes the command-line and GUI methods and includes information about transferring signal and parameter information between the disk and the GUI.

This multivariate wavelet de-noising problem deals with models of the form $X(t) = F(t) + e(t)$, where the observation X is p-dimensional, F is the deterministic signal to be recovered, and e is a spatially correlated noise signal. This kind of model is well suited for situations for which such additive, spatially correlated noise is realistic.

Multivariate Wavelet De-Noising Using the Command Line

This example uses noisy test signals. In this section, you will

- Load a multivariate signal.
- Display the original and observed signals.
- Remove noise by a simple multivariate thresholding after a change of basis.
- Display the original and denoised signals.
- Improve the obtained result by retaining less principal components.
- Display the number of retained principal components.
- Display the estimated noise covariance matrix.

1 Load a multivariate signal by typing the following at the MATLAB prompt:

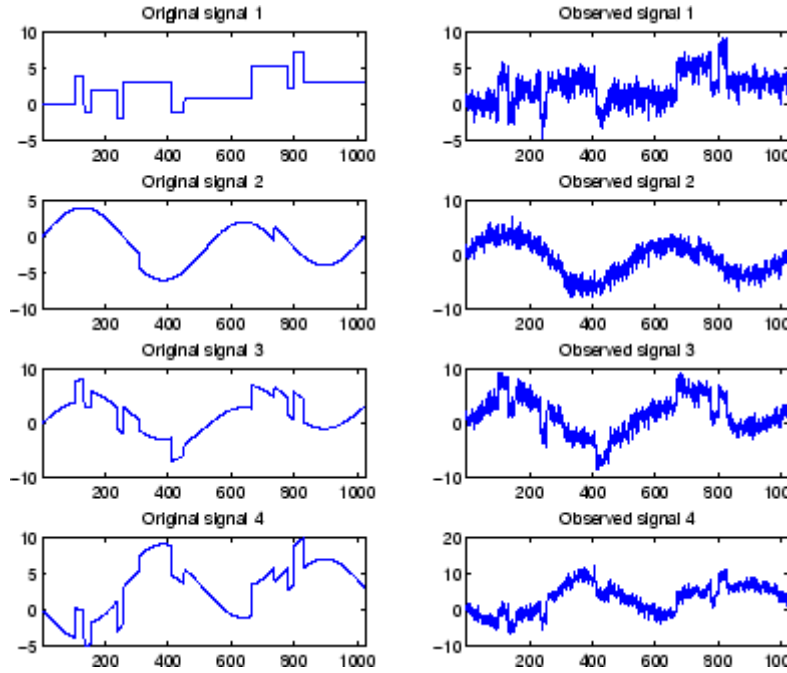
```
load ex4mwden
whos
```

Name	Size	Bytes	Class
covar	4x4	128	double array
x	1024x4	32768	double array
x_orig	1024x4	32768	double array

Usually, only the matrix of data `x` is available. Here, we also have the true noise covariance matrix (`covar`) and the original signals (`x_orig`). These signals are noisy versions of simple combinations of the two original signals. The first one is "Blocks" which is irregular, and the second is "HeavySine," which is regular except around time 750. The other two signals are the sum and the difference of the two original signals. Multivariate Gaussian white noise exhibiting strong spatial correlation is added to the resulting four signals, which leads to the observed data stored in `x`.

2 Display the original and observed signals by typing

```
kp = 0;
for i = 1:4
    subplot(4,2,kp+1), plot(x_orig(:,i));
    title(['Original signal ',num2str(i)])
    subplot(4,2,kp+2), plot(x(:,i));
    title(['Observed signal ',num2str(i)])
    kp = kp + 2;
end
```



The true noise covariance matrix is given by

```
covar

covar =
    1.0000    0.8000    0.6000    0.7000
    0.8000    1.0000    0.5000    0.6000
    0.6000    0.5000    1.0000    0.7000
    0.7000    0.6000    0.7000    1.0000
```

3 Remove noise by simple multivariate thresholding.

The de-noising strategy combines univariate wavelet de-noising in the basis where the estimated noise covariance matrix is diagonal with noncentered Principal Component Analysis (PCA) on approximations in the wavelet domain or with final PCA.

First, perform univariate de-noising by typing the following to set the de-noising parameters:

```
level = 5;
wname = 'sym4';
tptr  = 'sqtwolog';
sorh  = 's';
```

Then, set the PCA parameters by retaining all the principal components:

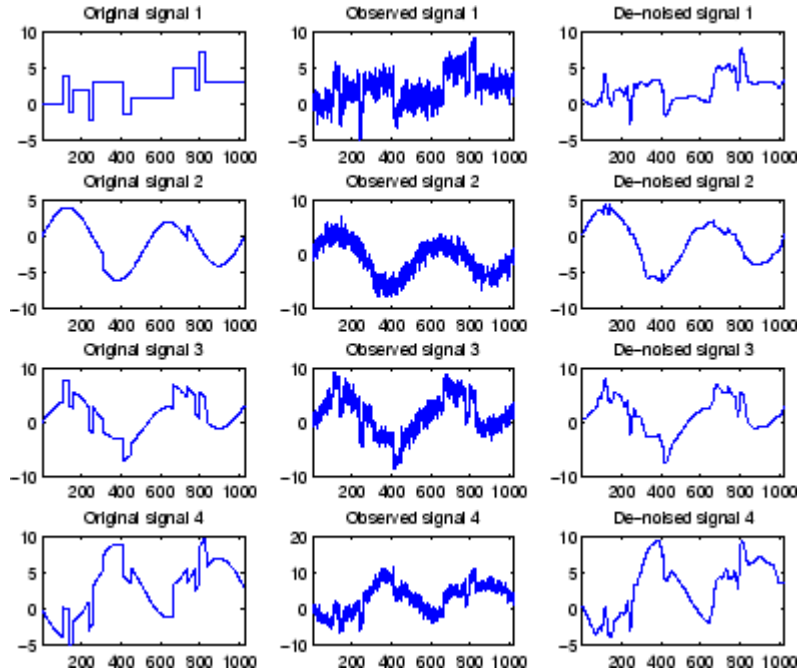
```
npc_app = 4;
npc_fin = 4;
```

Finally, perform multivariate de-noising by typing

```
x_den = wmulden(x, level, wname, npc_app, npc_fin, tptr, sorh);
```

4 Display the original and denoised signals by typing

```
kp = 0;
for i = 1:4
    subplot(4,3,kp+1), plot(x_orig(:,i));
    title(['Original signal ',num2str(i)])
    subplot(4,3,kp+2), plot(x(:,i));
    title(['Observed signal ',num2str(i)])
    subplot(4,3,kp+3), plot(x_den(:,i));
    title(['denoised signal ',num2str(i)])
    kp = kp + 3;
end
```



5 Improve the first result by retaining fewer principal components.

The results are satisfactory. Focusing on the two first signals, note that they are correctly recovered, but the result can be improved by taking advantage of the relationships between the signals, leading to an additional de-noising effect.

To automatically select the numbers of retained principal components by Kaiser's rule (which keeps the components associated with eigenvalues exceeding the mean of all eigenvalues), type

```
npc_app = 'kais';
npc_fin = 'kais';
```

Perform multivariate de-noising again by typing

```
[x_den, npc, nestco] = wmulden(x, level, wname, npc_app, ...
     npc_fin, tptr, sorh);
```

6 Display the number of retained principal components.

The second output argument gives the numbers of retained principal components for PCA for approximations and for final PCA.

```
npc

npc =
    2     2
```

As expected, since the signals are combinations of two initial ones, Kaiser's rule automatically detects that only two principal components are of interest.

7 Display the estimated noise covariance matrix.

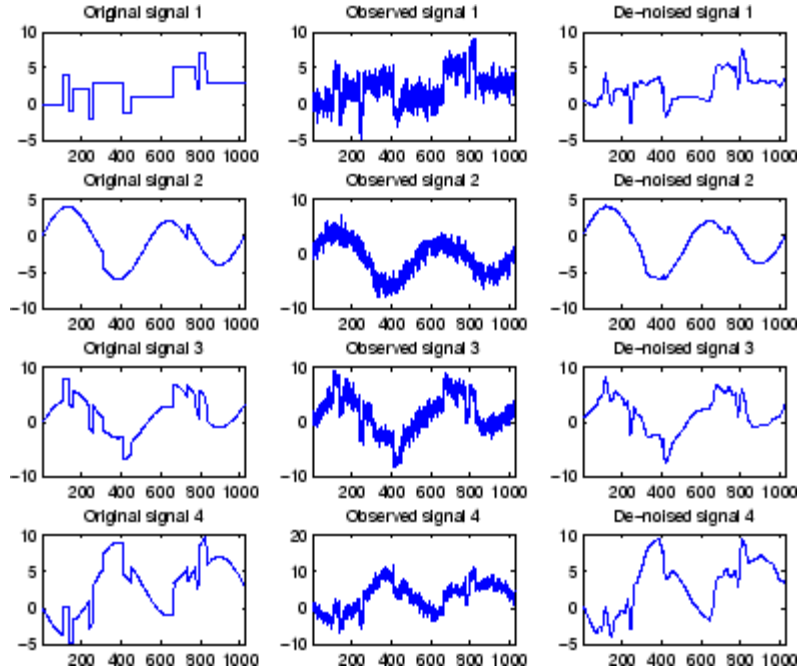
The third output argument contains the estimated noise covariance matrix:

```
nestco

nestco =
    1.0784    0.8333    0.6878    0.8141
    0.8333    1.0025    0.5275    0.6814
    0.6878    0.5275    1.0501    0.7734
    0.8141    0.6814    0.7734    1.0967
```

As you can see by comparing with the true matrix covar given previously, the estimation is satisfactory.

8 Display the original and final denoised signals by typing

```
kp = 0;
for i = 1:4
    subplot(4,3,kp+1), plot(x_orig(:,i));
    title(['Original signal ',num2str(i)])
    subplot(4,3,kp+2), plot(x(:,i));
    title(['Observed signal ',num2str(i)])
    subplot(4,3,kp+3), plot(x_den(:,i));
    title(['denoised signal ',num2str(i)])
    kp = kp + 3;
end
```



The results are better than those previously obtained. The first signal, which is irregular, is still correctly recovered, while the second signal, which is more regular, is denoised better after this second stage of PCA.

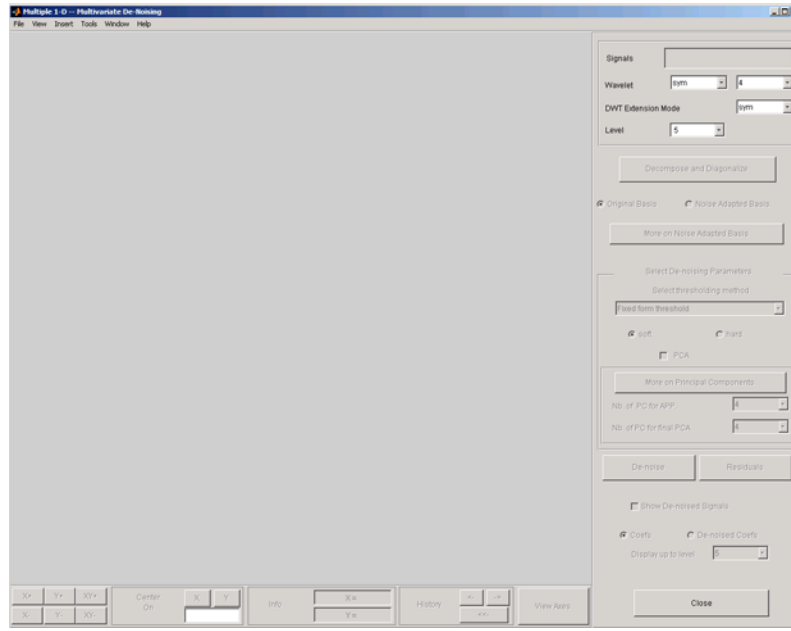
Multivariate Wavelet De-Noising Using the Graphical Interface

This section explores a de-noising strategy for multivariate signals using the graphical interface tools.

1 Start the Multivariate De-noising Tool by first opening the **Wavelet Toolbox Main Menu**.

```
wavemenu
```

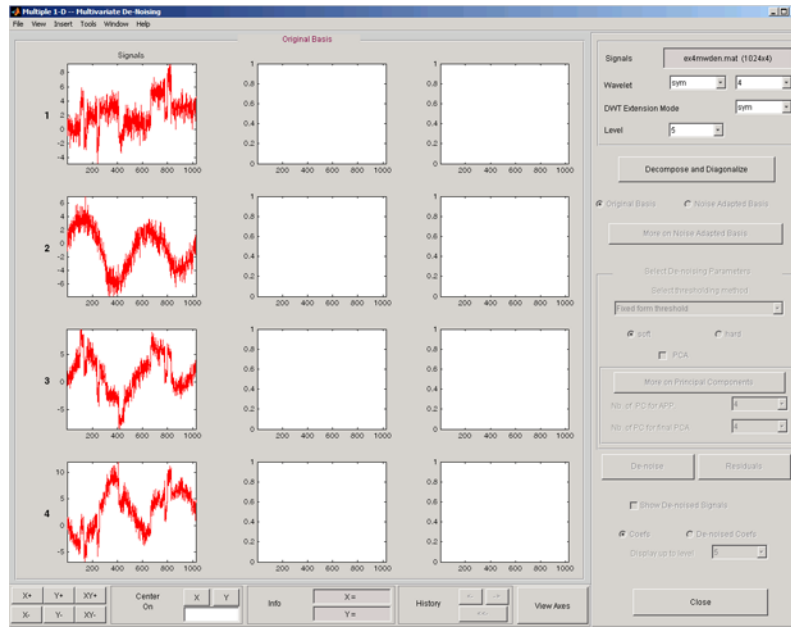
2 Click **Multivariate Denoising** to open the **Multivariate De-Noising** GUI.



3 Load data.

Select **File > Load Signals**. In the **Select** dialog box, select the MAT-file `ex4mwden.mat` from the MATLAB folder `toolbox/wavelet/wmultsig1d`.

Click **Open** to load the noisy multivariate signal into the GUI. The signal is a matrix containing four columns, where each column is a signal to be denoised.

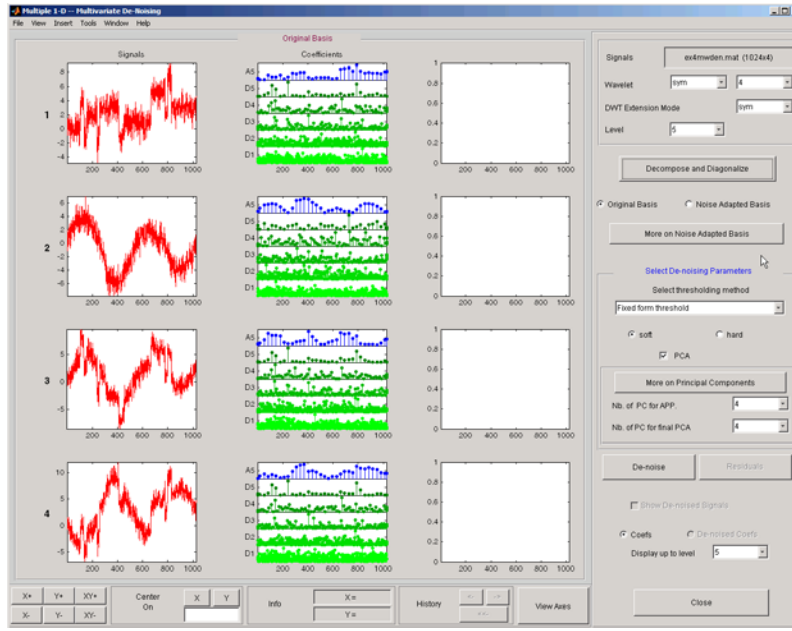


These signals are noisy versions from simple combinations of the two original signals. The first one is “Blocks” which is irregular and the second is “HeavySine” which is regular except around time 750. The other two signals are the sum and the difference between the original signals. Multivariate Gaussian white noise exhibiting strong spatial correlation is added to the resulting four signals.

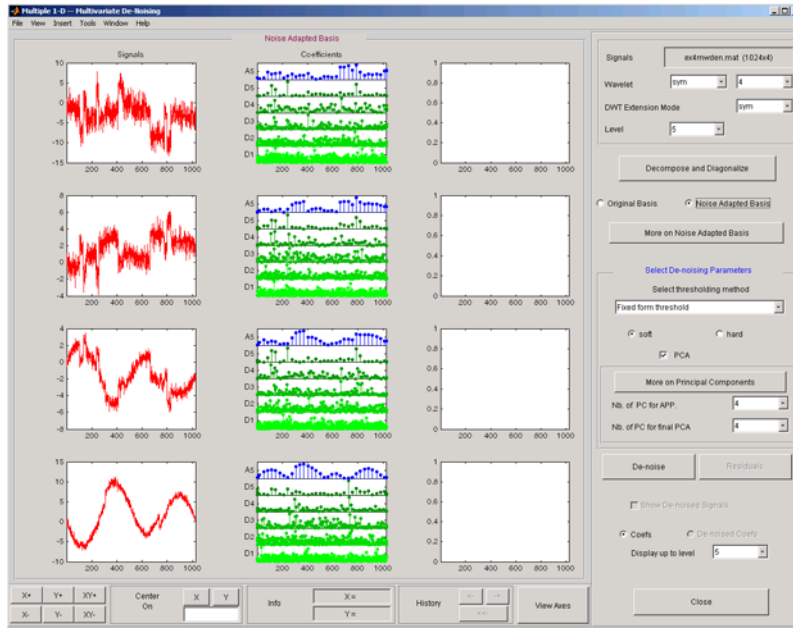
The following example illustrates the two different aspects of the proposed de-noising method. First, perform a convenient change of basis to cope with spatial correlation and denoise in the new basis. Then, use PCA to take advantage of the relationships between the signals, leading to an additional de-noising effect.

4 Perform a wavelet decomposition and diagonalize the noise covariance matrix.

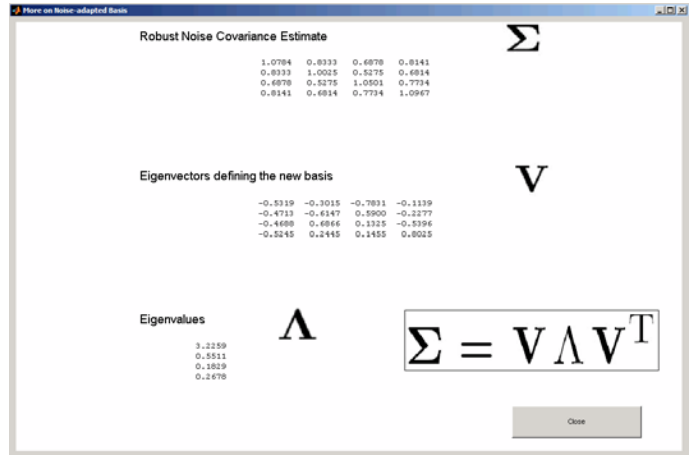
Use the displayed default values for the **Wavelet**, the **DWT Extension Mode**, and the decomposition **Level**, and then click **Decompose and Diagonalize**. The tool displays the wavelet approximation and detail coefficients of the decomposition of each signal in the original basis.



Select **Noise Adapted Basis** to display the signals and their coefficients in the noise-adapted basis.



To see more information about this new basis, click **More on Noise Adapted Basis**. A new figure displays the robust noise covariance estimate matrix and the corresponding eigenvectors and eigenvalues.



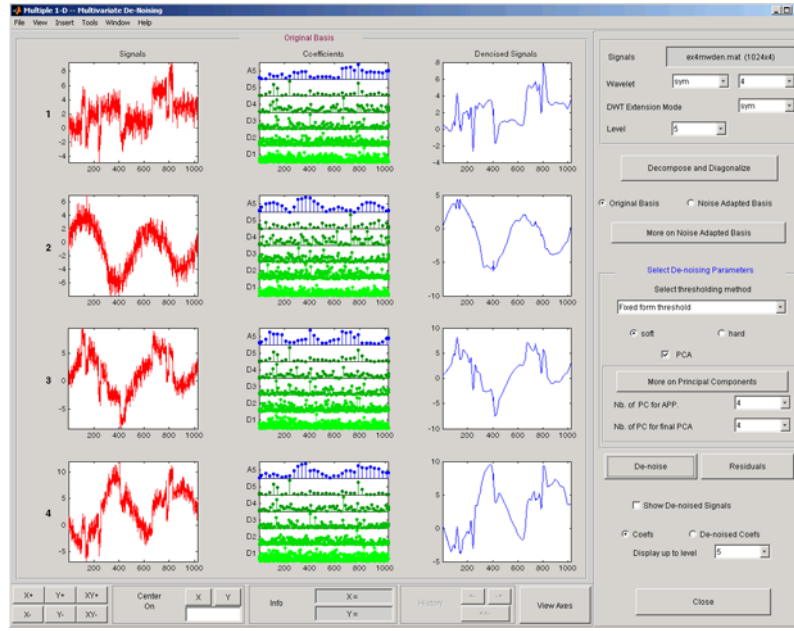
Eigenvectors define the change of basis, and eigenvalues are the variances of uncorrelated noises in the new basis.

The multivariate de-noising method proposed below is interesting if the noise covariance matrix is far from diagonal exhibiting spatial correlation, which, in this example, is the case.

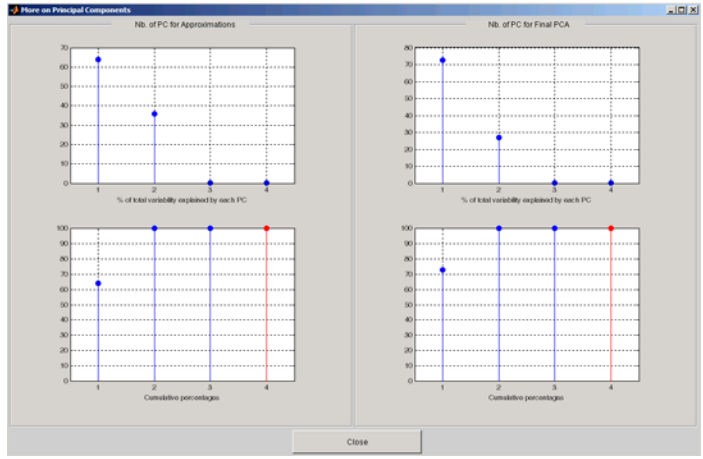
5 denoise the multivariate signal.

A number of options are available for fine-tuning the de-noising algorithm. However, we will use the defaults: fixed form soft thresholding, scaled white noise model, and the proposed numbers of retained principal components. In this case, the default values for PCA lead to retaining all the components.

Select **Original Basis** to return to the original basis and then click **denoise**.

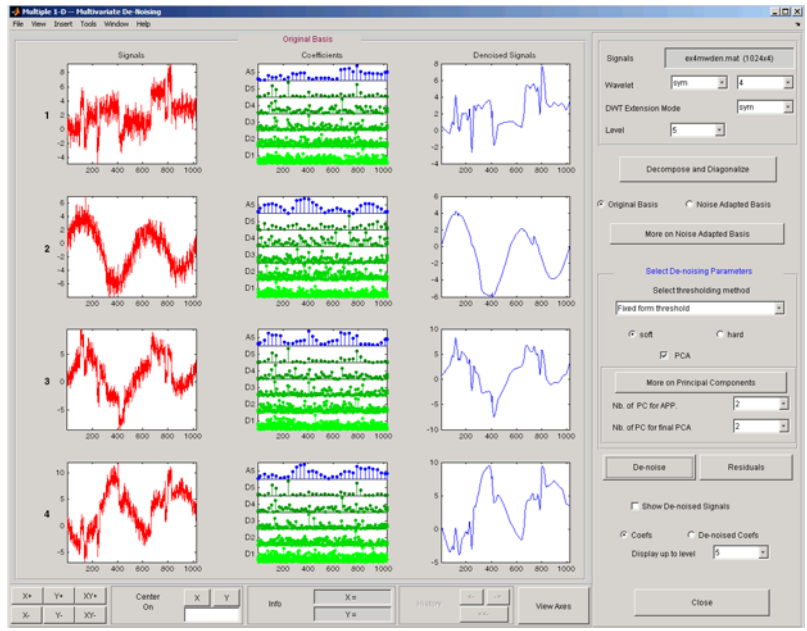


The results are satisfactory. Both of the two first signals are correctly recovered, but they can be improved by getting more information about the principal components. Click **More on Principal Components**.

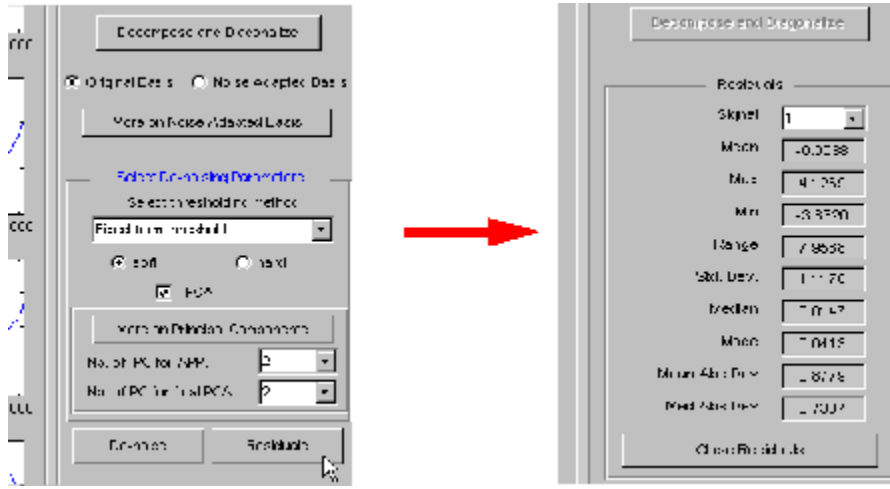


A new figure displays information to select the numbers of components to keep for the PCA of approximations and for the final PCA after getting back to the original basis. You can see the percentages of variability explained by each principal component and the corresponding cumulative plot. Here, it is clear that only two principal components are of interest.

Close the **More on Principal Components** window. Select 2 as the **Nb. of PC for APP**. Select 2 as the **Nb. of PC for final PCA**, and then click **denoise**.



The results are better than those previously obtained. The first signal, which is irregular, is still correctly recovered. The second signal, which is more regular, is denoised better after this second stage of PCA. You can get more information by clicking **Residuals**.



Importing and Exporting from the GUI

The tool lets you save denoised signals to disk by creating a MAT-file in the current folder with a name of your choice.

To save the signal denoised in the previous section,

1. Select **File > Save denoised Signals**.
2. Select **Save denoised Signals and Parameters**. A dialog box appears that lets you specify a folder and filename for storing the signal.
3. Type the name `s_ex4mwden` and click **OK** to save the data.
4. Load the variables into your workspace:

```
load s_ex4mwdent
whos
```

Name	Size	Bytes	Class
DEN_Params	1x1	430	struct array
PCA_Params	1x1	1536	struct array
x	1024x4	32768	struct array

The denoised signals are in matrix `x`. The parameters (`PCA_Params` and `DEN_Params`) of the two-stage de-noising process are also available.

- PCA_Params are the change of basis and PCA parameters:

```
PCA_Params

PCA_Params =
    NEST: {[4x4 double]  [4x1 double]  [4x4 double]}
     APP: {[4x4 double]  [4x1 double]  [2]}
     FIN: {[4x4 double]  [4x1 double]  [2]}
```

`PCA_Params.NEST{1}` contains the change of basis matrix. `PCA_Params.NEST{2}` contains the eigenvalues, and `PCA_Params.NEST{3}` is the estimated noise covariance matrix.

`PCA_Params.APP{1}` contains the change of basis matrix, `PCA_Params.APP{2}` contains the eigenvalues, and `PCA_Params.APP{3}` is the number of retained principal components for approximations.

The same structure is used for `PCA_Params.FIN` for the final PCA.

- `DEN_Params` are the de-noising parameters in the diagonal basis:

```
DEN_Params

DEN_Params =
     thrVAL: [4.8445 2.0024 1.1536 1.3957 0]
    thrMETH: 'sqtwolog'
    thrTYPE: 's'
```

The thresholds are encoded in `thrVAL`. For `j` from `1` to `5`, `thrVAL(j)` contains the value used to threshold the detail coefficients at level `j`. The thresholding method is given by `thrMETH` and the thresholding mode is given by `thrTYPE`.

Multiscale Principal Components Analysis

This section demonstrates the features of multiscale principal components analysis provided in the Wavelet Toolbox software. The toolbox includes the `wmspca` function and a graphical user interface (GUI) available from `wavemenu`. This section describes the command-line and GUI methods, and information about transferring signal and parameter information between the disk and the GUI.

The aim of multiscale PCA is to reconstruct, starting from a multivariate signal and using a simple representation at each resolution level, a simplified multivariate signal. The multiscale principal components generalizes the normal PCA of a multivariate signal represented as a matrix by performing a PCA on the matrices of details of different levels simultaneously. A PCA is also performed on the coarser approximation coefficients matrix in the wavelet domain as well as on the final reconstructed matrix. By selecting the numbers of retained principal components, interesting simplified signals can be reconstructed.

For a more detailed introduction and use of multiscale PCA for Statistical Process Control see the paper written by B. Bakshi (see [Bak95] in "References").

Since you can perform multiscale PCA either from the command line or using the GUI, this section has subsections covering each method.

Multiscale Principal Components Analysis Using the Command Line

This example uses noisy test signals. In this section, you will:

- Load a multivariate signal.
- Perform a simple multiscale PCA.
- Display the original and simplified signals.
- Improve the obtained result by retaining less principal components.

1 Load a multivariate signal by typing at the MATLAB prompt:

```
load ex4mwden
whos
```

Name	Size	Bytes	Class
covar	4x4	128	double array
x	1024x4	32768	double array
x_orig	1024x4	32768	double array

The data stored in matrix `x` comes from two test signals, Blocks and HeavySine, and from their sum and difference, to which multivariate Gaussian white noise has been added.

2 Perform a simple multiscale PCA.

The multiscale PCA combines noncentered PCA on approximations and details in the wavelet domain and a final PCA. At each level, the most significant principal components are selected.

First, set the wavelet parameters:

```
level= 5;
wname = 'sym4';
```

Then, automatically select the number of retained principal components using Kaiser's rule by typing

```
npc = 'kais';
```

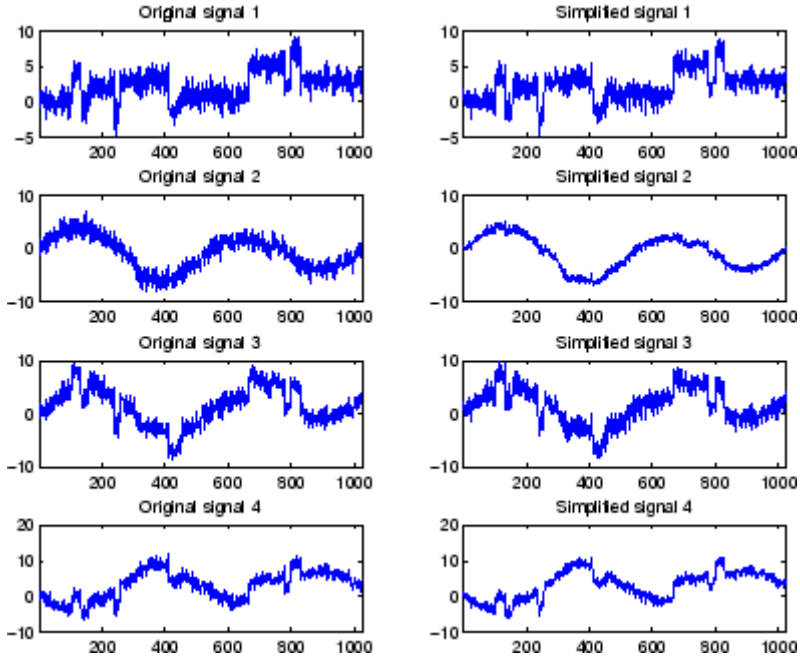
Finally, perform multiscale PCA:

```
[x_sim, qual, npc] = wmspca(x ,level, wname, npc);
```

3 Display the original and simplified signals:

```
kp = 0;
for i = 1:4
    subplot(4,2,kp+1), plot(x (:,i));
    title(['Original signal ',num2str(i)])
    subplot(4,2,kp+2), plot(x_sim(:,i));
    title(['Simplified signal ',num2str(i)])
```

```
    kp = kp + 2;
end
```



The results from a compression perspective are good. The percentages reflecting the quality of column reconstructions given by the relative mean square errors are close to 100%.

```
qual

qual =

   98.0545   93.2807   97.1172   98.8603
```

4 Improve the first result by retaining fewer principal components.

The results can be improved by suppressing noise, because the details at levels 1 to 3 are composed essentially of noise with small contributions from the signal. Removing the noise leads to a crude, but large, denoising effect.

The output argument npc contains the numbers of retained principal components selected by Kaiser's rule:

```
npc

npc =
     1     1     1     1     1     2     2
```

For `d` from 1 to 5, `npc(d)` is the number of retained noncentered principal components (PCs) for details at level d. The number of retained noncentered PCs for approximations at level 5 is `npc(6)`, and `npc(7)` is the number of retained PCs for final PCA after wavelet reconstruction. As expected, the rule keeps two principal components, both for the PCA approximations and the final PCA, but one principal component is kept for details at each level.

To suppress the details at levels 1 to 3, update the `npc` argument as follows:

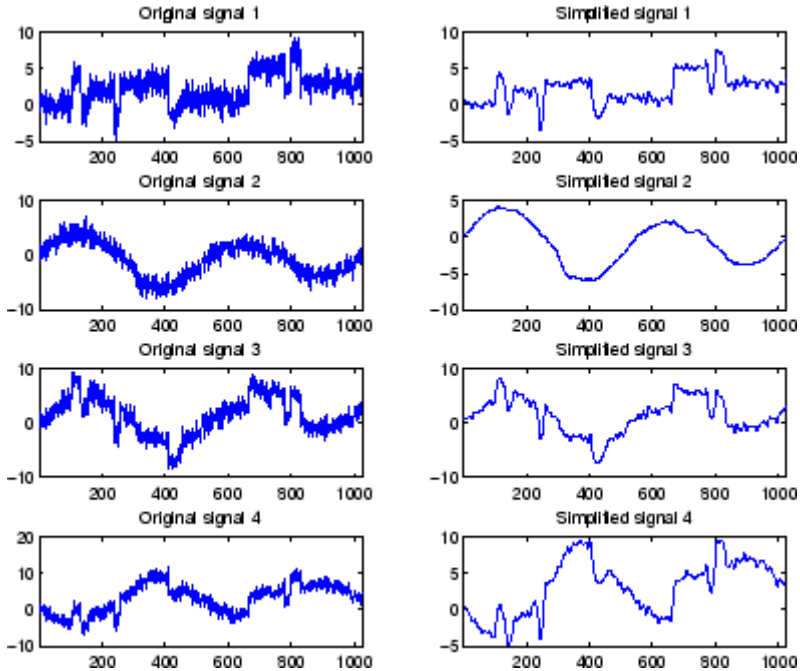
```
npc(1:3) = zeros(1,3);

npc

npc =
0     0     0     1     1     2     2
```

Then, perform multiscale PCA again:

```
[x_sim, qual, npc] = wmspca(x, level, wname, npc);
```

5 Display the original and final simplified signals:

```
kp = 0;
for i = 1:4
    subplot(4,2,kp+1), plot(x (:,i));
    title(['Original signal ',num2str(i)])
    subplot(4,2,kp+2), plot(x_sim(:,i));
    title(['Simplified signal ',num2str(i)])
    kp = kp + 2;
end
```



As shown, the results are improved.

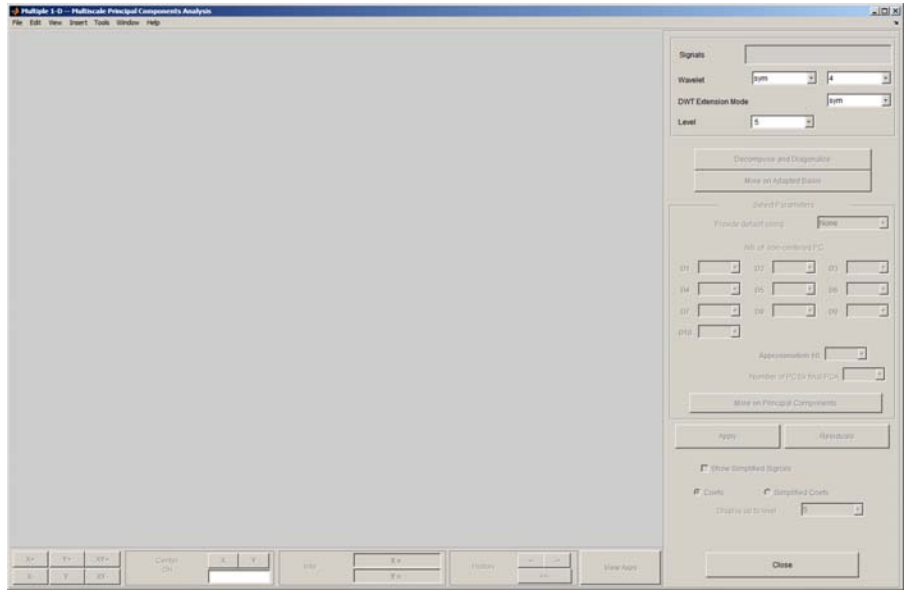
Multiscale Principal Components Analysis Using the Graphical Interface

This section explores multiscale PCA using the GUIs.

1 Start the Multiscale Princ. Comp. Analysis tool by first opening the Wavelet Toolbox Main Menu:

```
wavemenu
```

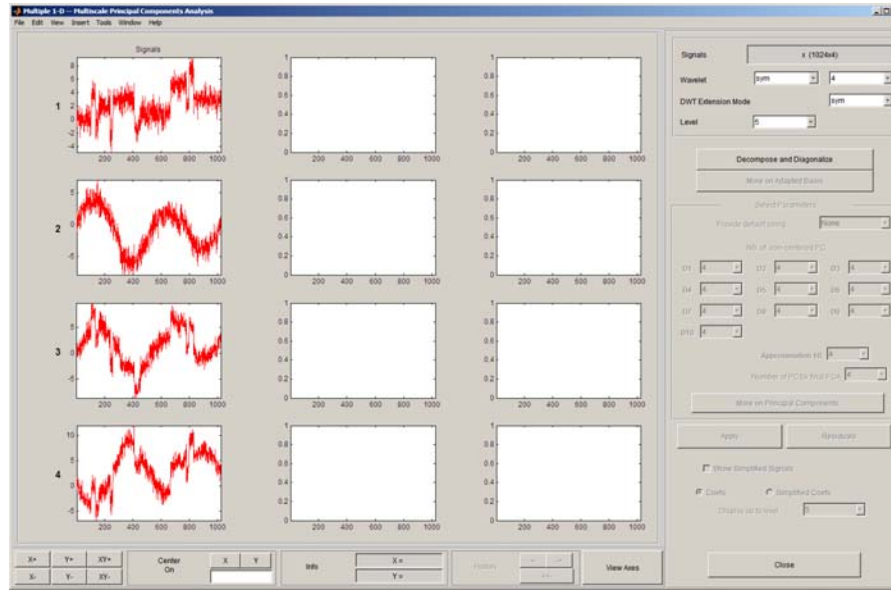
2 Click **Multiscale Princ. Comp. Analysis** to open the Multiscale Principal Components Analysis GUI.



3 Load data.

Select **File > Load Signals**. In the **Select** dialog box, select the MAT-file `ex4mwden.mat` from the MATLAB folder `toolbox/wavelet/wmultsig1d`.

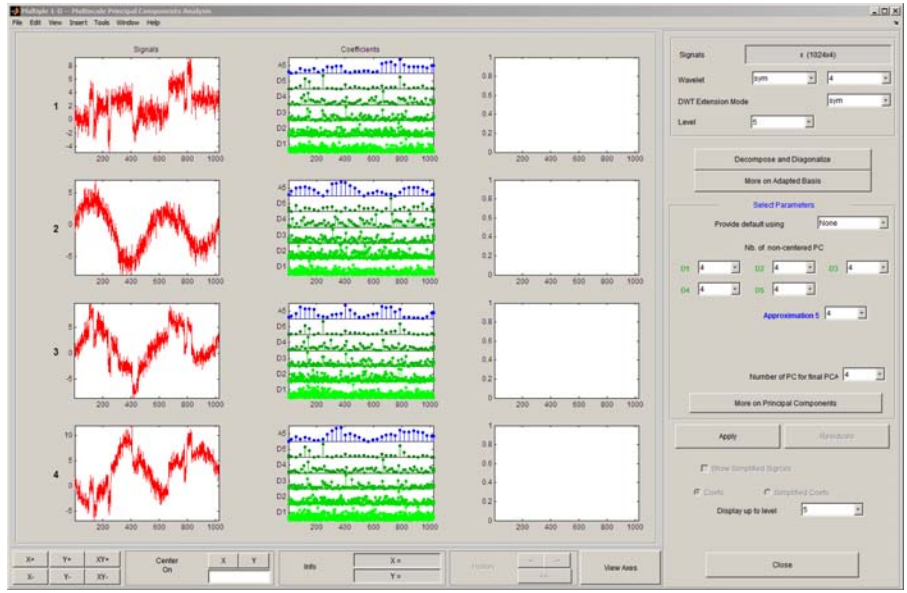
Click **Open** to load the multivariate signal into the GUI. The signal is a matrix containing four columns, where each column is a signal to be simplified.



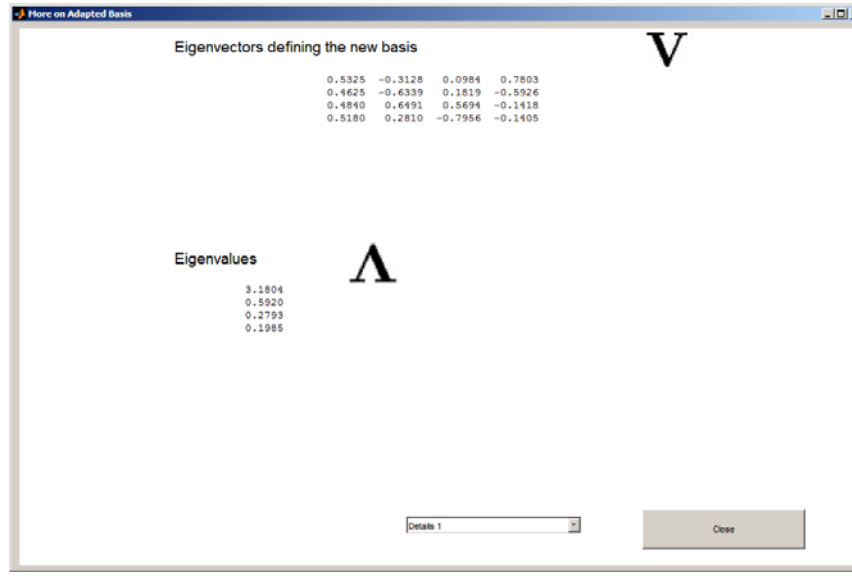
These signals are noisy versions from simple combinations of the two original signals, Blocks and HeavySine and their sum and difference, each with added multivariate Gaussian white noise.

4 Perform a wavelet decomposition and diagonalize each coefficients matrix.

Use the default values for the **Wavelet**, the **DWT Extension Mode**, and the decomposition **Level**, and then click **Decompose and Diagonalize**. The tool displays the wavelet approximation and detail coefficients of the decomposition of each signal in the original basis.



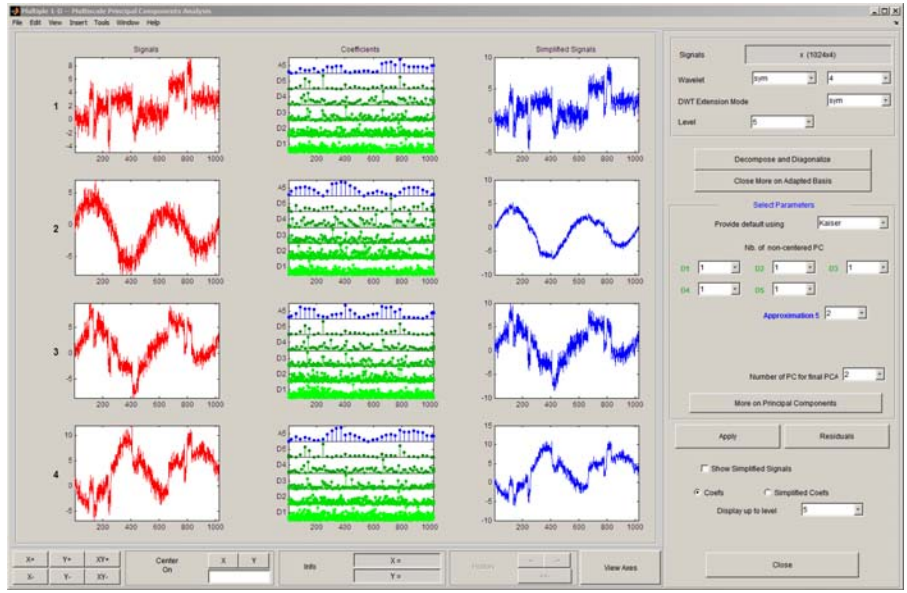
To get more information about the new bases allowed for performing a PCA for each scale, click **More on Adapted Basis**. A new figure displays the corresponding eigenvectors and eigenvalues for the matrix of the detail coefficients at level 1.



You can change the level or select the coarser approximations or the reconstructed matrix to investigate the different bases. When you finish, click **Close**.

5 Perform a simple multiscale PCA.

The initial values for PCA lead to retaining all the components. Select `Kaiser` from the **Provide default using** drop-down list, and click **Apply**.

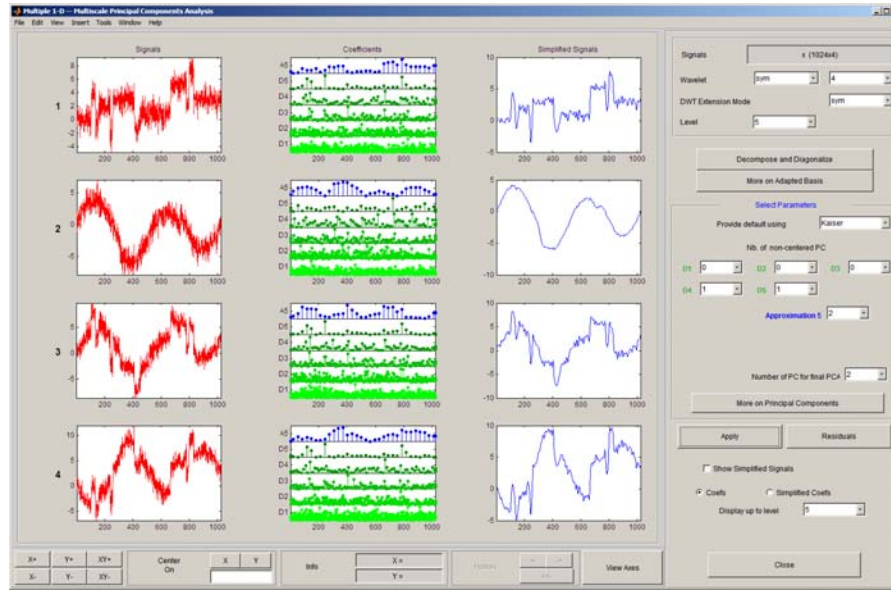


The results are good from a compression perspective.

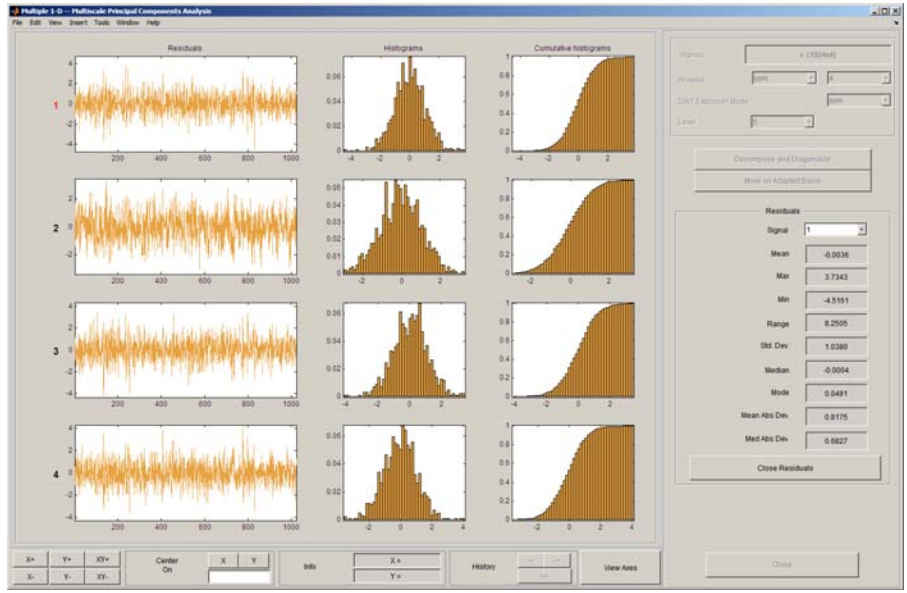
6 Improve the obtained result by retaining fewer principal components.

The results can be improved by suppressing the noise, because the details at levels 1 to 3 are composed essentially of noise with small contributions from the signal, as you can see by careful inspection of the detail coefficients. Removing the noise leads to a crude, but large, de-noising effect.

For **D1**, **D2** and **D3**, select `0` as the **Nb. of non-centered PC** and click **Apply**.



The results are better than those previously obtained. The first signal, which is irregular, is still correctly recovered, while the second signal, which is more regular, is denoised better after this second stage of PCA. You can get more information by clicking **Residuals**.



Importing and Exporting from the GUI

The Multiscale Principal Components Analysis tool lets you save the simplified signals to disk. The toolbox creates a MAT-file in the current folder with a name of your choice.

To save the simplified signals from the previous section:

1. Select **File > Save Simplified Signals**.
2. Select **Save Simplified Signals and Parameters**. A dialog box appears that lets you specify a folder and file name for storing the signal.
3. Type the name `s_ex4mwden` and click **OK** to save the data.
4. Load the variables into your workspace:

```
load s_ex4mwden
whos
```

Name	Size	Bytes	Class
PCA_Params	1x7	2628	struct array
x	1024x4	32768	double array

The simplified signals are in matrix `x`. The parameters of multiscale PCA are available in `PCA_Params`:

```
PCA_Params

PCA_Params =
1x7 struct array with fields:
    pc
    variances
    npc
```

`PCA_Params` is a structure array of length `d+2` (here, the maximum decomposition level `d=5`) such that `PCA_Params(d).pc` is the matrix of principal components. The columns are stored in descending order of the variances. `PCA_Params(d).variances` is the principal component variances vector, and `PCA_Params(d).npc` is the vector of selected numbers of retained principal components.

One-Dimensional Multisignal Analysis

This section takes you through the features of one-dimensional multisignal wavelet analysis, compression and denoising using the Wavelet Toolbox software. The rationale for each topic is the same as in the 1-D single signal case (for details, see "Advanced Concepts" in the *Wavelet Toolbox User's Guide*).

The toolbox provides the following functions for multisignal analysis.

Analysis-Decomposition and Synthesis-Reconstruction Functions

Function Name	Purpose
`mdwtdec`	Multisignal wavelet decomposition
`mdwtrec`	Multisignal wavelet reconstruction and extraction of approximation and detail coefficients

Decomposition Structure Utilities

Function Name	Purpose
`chgwdeccfs`	Change multisignal 1-D decomposition coefficients
`wdcenergy`	Multisignal 1-D decomposition energy repartition

Compression and Denoising Functions

Function Name	Purpose
`mswcmp`	Multisignal 1-D compression using wavelets
`mswcmpscr`	Multisignal 1-D wavelet compression scores
`mswcmptp`	Multisignal 1-D compression thresholds and performance
`mswden`	Multisignal 1-D denoising using wavelets
`mswthresh`	Perform multisignal 1-D thresholding

You can perform analyses from the MATLAB command line or by using the graphical interface tools. This section describes each method. The last section discusses how to exchange signal and coefficient information between the disk and the graphical tools.

One-Dimensional Multisignal Analysis Using the Command Line

1 Load a file, from the MATLAB prompt, by typing

```
load thinker
```

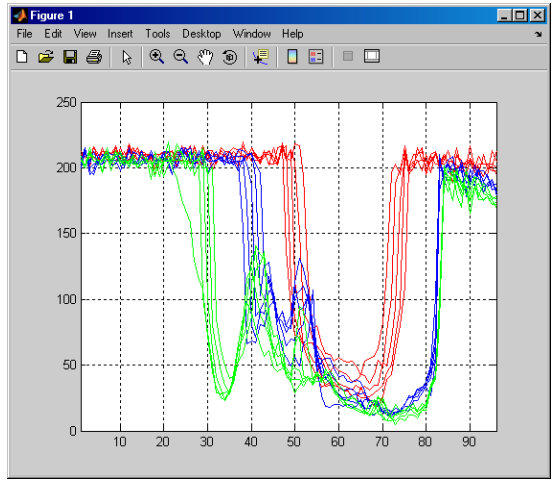
The file `thinker.mat` contains a single variable `X`. Use `whos` to show information about `X`.

```
whos
```

Name	Size	Bytes	Class
X	192x96	147456	double array

2 Plot some signals.

```
figure;
plot(X(1:5,:)','r');   hold on
plot(X(21:25,:)','b'); plot(X(31:35,:)','g')
set(gca,'Xlim',[1,96])
grid
```



3 Perform a wavelet decomposition of signals at level 2 of row signals using the `db2` wavelet.

```
dec = mdwtdec('r',X,2,'db2')
```

This generates the decomposition structure `dec`:

```
dec =
          dirDec: 'r'
           level: 2
           wname: 'db2'
      dwtFilters: [1x1 struct]
         dwtEXTM: 'sym'
        dwtShift: 0
        dataSize: [192 96]
              ca: [192x26 double]
              cd: {[192x49 double]  [192x26 double]}
```

4 Change wavelet coefficients.

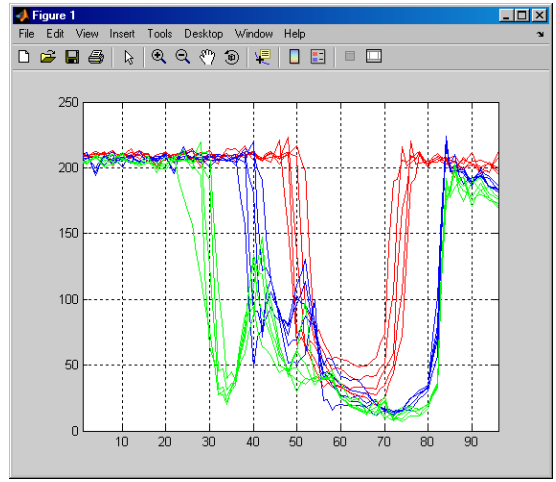
For each signal change the wavelet coefficients by setting all the coefficients of the detail of level 1 to zero.

```
decBIS = chgwdeccfs(dec,'cd',0,1);
```

This generates a new decomposition structure `decBIS`.

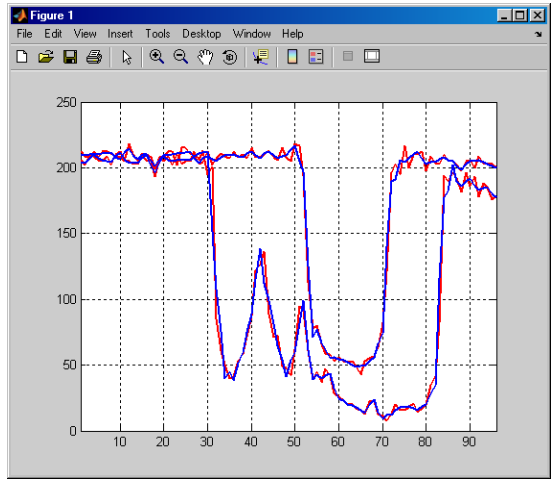
5 Perform a wavelet reconstruction of signals and plot some of the new signals.

```
Xbis = mdwtrec(decBIS);
figure;
plot(Xbis(1:5,:)','r');   hold on
plot(Xbis(21:25,:)','b');
plot(Xbis(31:35,:)','g')
grid; set(gca,'Xlim',[1,96])
```



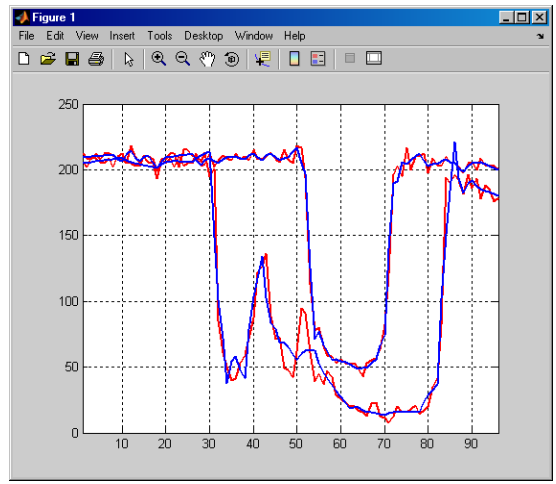
Compare old and new signals by plotting them together.

```
figure; idxSIG = [1 31];
plot(X(idxSIG,:)','r','linewidth',2);   hold on
plot(Xbis(idxSIG,:)','b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
```



6 Set the wavelet coefficients at level 1 and 2 for signals 31 to 35 to the value zero, perform a wavelet reconstruction of signal 31, and compare some of the old and new signals.

```
decTER = chgwdeccfs(dec,'cd',0,1:2,31:35);
Y = mdwtrec(decTER,'a',0,31);
figure;
plot(X([1 31],:)','r','linewidth',2);   hold on
plot([Xbis(1,:)
; Y]','b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
```



7 Compute the energy of signals and the percentage of energy for wavelet components.

```
[E,PEC,PECFS] = wdecenergy(dec);
```

Energy of signals 1 and 31:

```
Ener_1_31 = E([1 31])
Ener_1_31 =

  1.0e+006 *
    3.7534
    2.2411
```

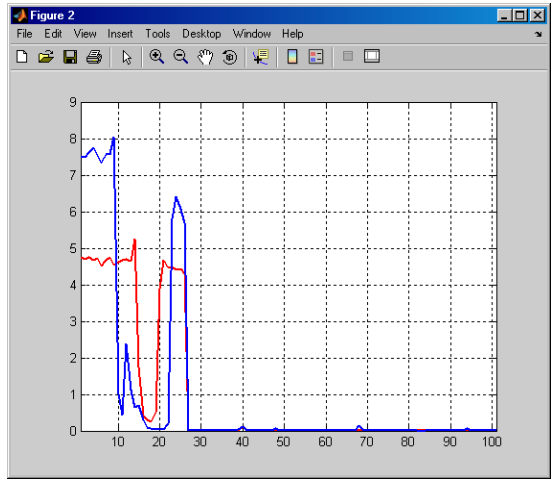
8 Compute the percentage of energy for wavelet components of signals 1 and 31.

```
PEC_1_31 = PEC([1 31],:)

PEC_1_31 =
   99.7760    0.1718    0.0522
   99.3850    0.2926    0.3225
```

The first column shows the percentage of energy for approximations at level 2. Columns 2 and 3 show the percentage of energy for details at level 2 and 1, respectively.

9 Display the percentage of energy for wavelet coefficients of signals 1 and 31. As we can see in the `dec` structure, there are 26 coefficients for the approximation and the detail at level 2, and 49 coefficients for the detail at level 1.

```
PECFS_1 = PECFS(1,:); PECFS_31 = PECFS(31,:);
figure;
plot(PECFS_1,'r','linewidth',2); hold on
plot(PECFS_31,'b','linewidth',2);
grid; set(gca,'Xlim',[1,size(PECFS,2)])
```



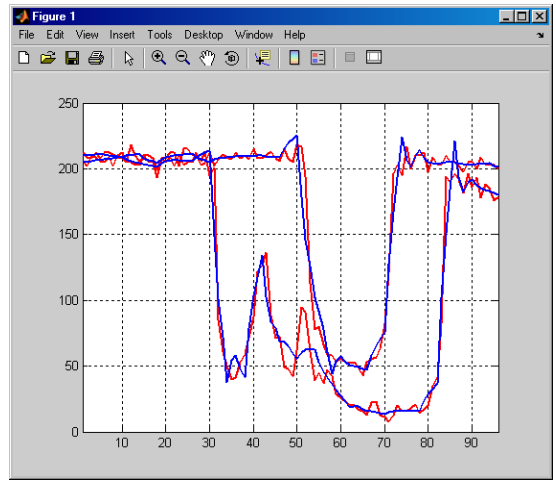
10 Compress the signals to obtain a percentage of zeros near 95% for the wavelet coefficients.

```
[XC,decCMP,THRESH] = mswcmp('cmp',dec,'NO_perf',95);
[Ecmp,PECcmp,PECFScmp] = wdecenergy(decCMP);
```

Plot the original signals 1 and 31, and the corresponding compressed signals.

```
figure;
```

```
plot(X([1 31],:)','r','linewidth',2);   hold on
plot(XC([1 31],:)','b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
```



Compute thresholds, percentage of energy preserved and percentage of zeros associated with the `L2_perf` method preserving at least 95% of energy.

```
[THR_VAL,L2_Perf,NO_Perf] = mswcmptp(dec,'L2_perf',95);
idxSIG = [1,31];

Thr   = THR_VAL(idxSIG)
Thr =
  256.1914
  158.6085

L2per = L2_Perf(idxSIG)
L2per =
  96.5488
  94.7197

NOper = NO_Perf(idxSIG)
NOper =
  79.2079
```

```
    86.1386
```

Compress the signals to obtain a percentage of zeros near 60% for the wavelet coefficients.

```
[XC,decCMP,THRESH] = mswcmp('cmp',dec,'NO_perf',60);
```

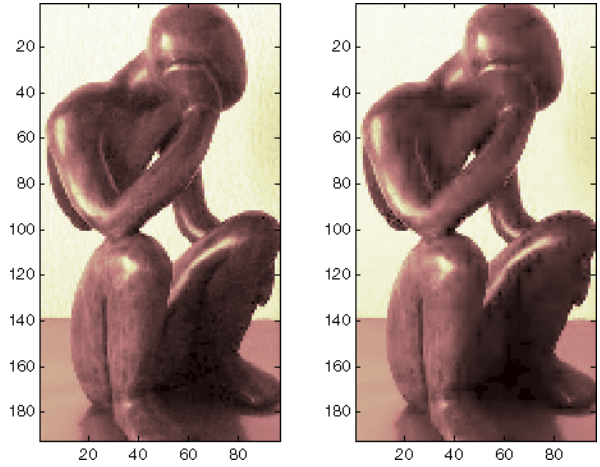
`XC` signals are the compressed versions of the original signals in the row direction.

Compress the `XC` signals in the column direction

```
XX = mswcmp('cmpsig','c',XC,'db2',2,'NO_perf',60);
```

Plot original signals `X` and the compressed signals `XX` as images.

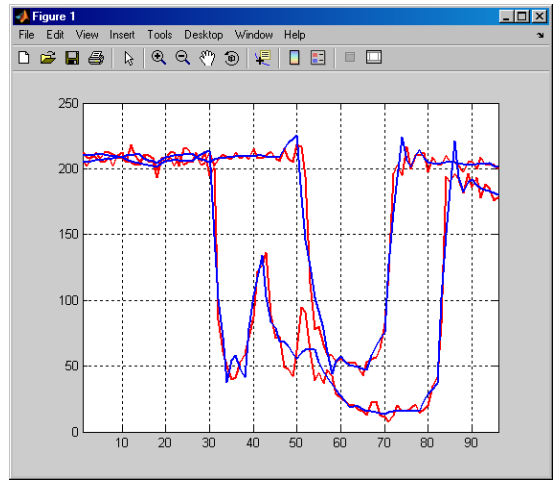
```
figure;
subplot(1,2,1); image(X)
subplot(1,2,2); image(XX)
colormap(pink(222))
```



11 Denoise the signals using the universal threshold:

```
[XD,decDEN,THRESH] = mswden('den',dec,'sqtwolog','sln'); figure;
plot(X([1 31],:)','r','linewidth',2); hold on
plot(XD([1 31],:)','b','linewidth',2);
```

```
grid; set(gca,'Xlim',[1,96])
```



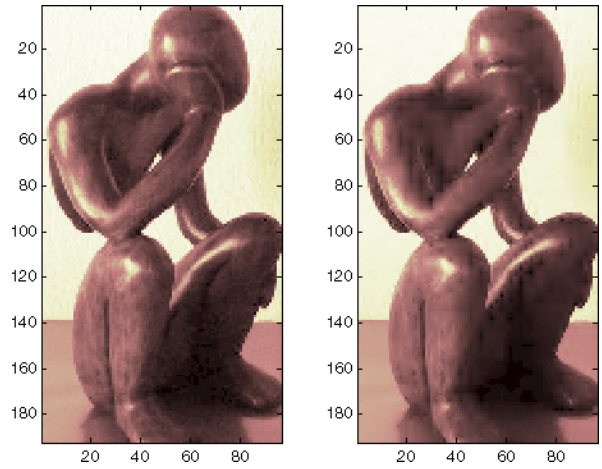
`XD` signals are the denoised versions of the original signals in the row direction.

Denoise the `XD` signals in column direction

```
XX = mswden('densig','c',XD,'db2',2,'sqtwolog','sln');
```

Plot original signals `X` and the denoised signals `XX` as images.

```
figure;
subplot(1,2,1); image(X)
subplot(1,2,2); image(XX)
colormap(pink(222))
```



One-Dimensional Multisignal Analysis Using the Graphical Interface

In this section, we explore the same signal as in the previous section, but use the graphical interface tools to analyze it.

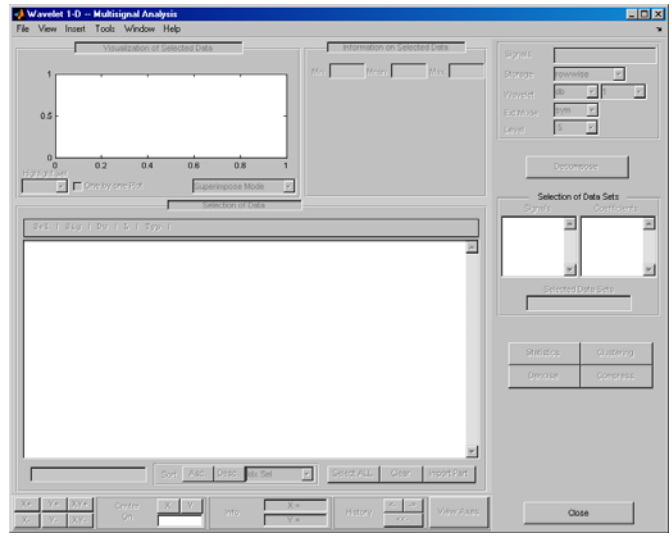
1 Start the Wavelet 1-D Multisignal Analysis Tool.

From the MATLAB prompt, type:

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.

Click **Multisignal Analysis 1-D** to open the Wavelet 1-D Multisignal Analysis tool.



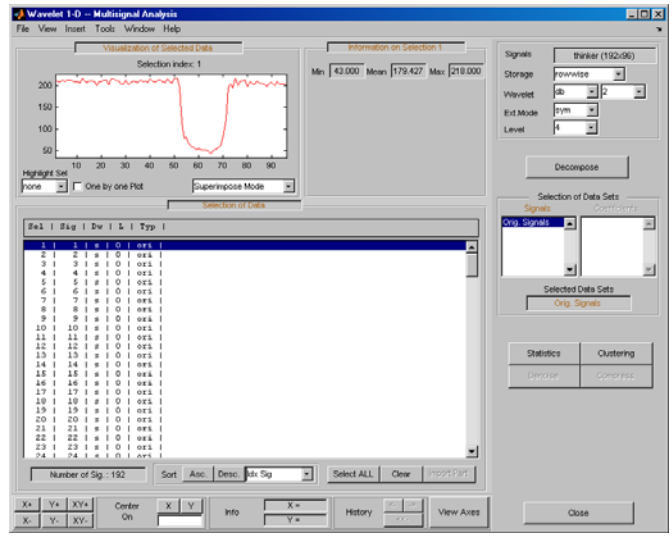
The tool is divided into five panes. Two of them are the same as in all Wavelet Toolbox GUIs: the Command Frame on the right side of the figure and the Dynamic Visualization tool at the bottom. The Command Frame contains a special component found in all multisignal tools: the **Selection of Data Sets** pane which is used to manage two lists.

The three new panes are the **Visualization of Selected Data** pane, the **Information on Selected Data** pane, and the **Selection of Data** pane.

2 Load the signals.

From the **File** menu, select **Load > Signals**. When the Load Signal dialog box appears, select the demo MAT-file `thinker.mat` from the MATLAB folder `toolbox/wavelet/wmultisig1d` and click **OK**.

The data matrix loads in the Wavelet 1-D Multisignal Analysis **tool**, and the first signal appears.



The **Selection of Data** pane contains a list of selectable signals. At the beginning, only the originally loaded signals are available. You can generate and add new signals to the list by decomposing, compressing, or denoising original signals.

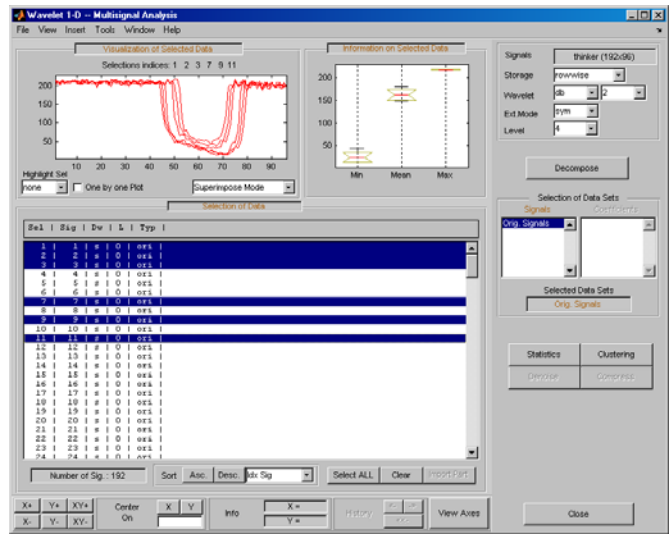
Each row of the list displays the index of selectable signal (`Idx Sel`), the index of original signal (`Idx Sig`) and three wavelet transform attributes describing the process used to obtain the selectable signal from the original one.

3 View the signals and signal information.

With signal 1 highlighted, Shift-click the mouse on signal 3 to select signals 1, 2, and 3.

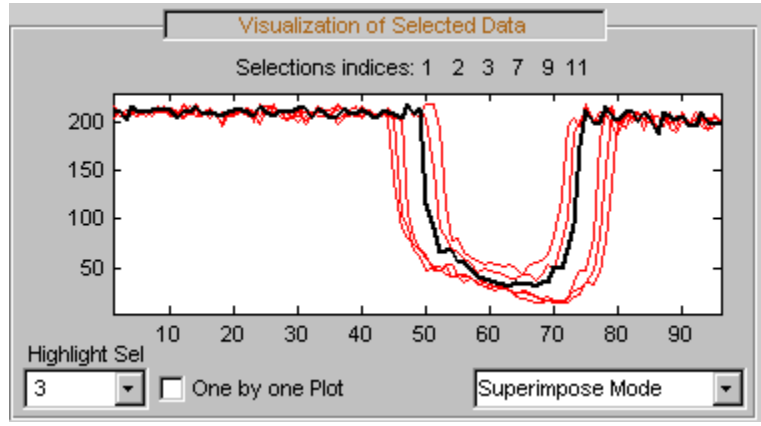
Ctrl-click the mouse on signals 7, 9, and 11. (The **Select ALL** button at the bottom of the **Selection of Data** pane selects all signals and the **Clear** button deselects all signals.)

The selected signals (1, 2, 3, 7, 9 and 11) appear in the **Visualization of Selected Data** pane. The **Information on Selected Data** pane contains the box plots of the minimums, the means, and the maximums of these signals.



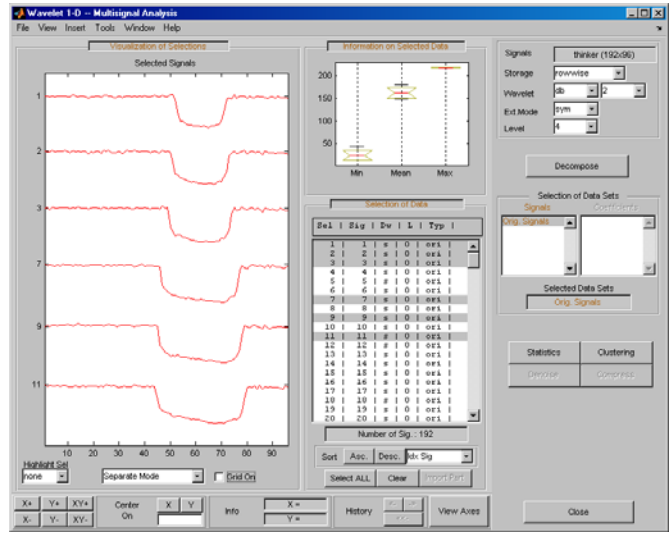
4 Highlight a signal.

Using the **Highlight Sel** button in the lower-left corner of the **Visualization of Selected Data** pane, select signal 3.



5 Select Different Views.

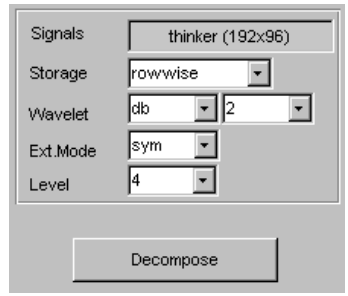
In the **Visualization of Selected Data** pane, change the view mode using the pop-up in the lower-right corner. Choose **Separate Mode**. The selected signals appear.



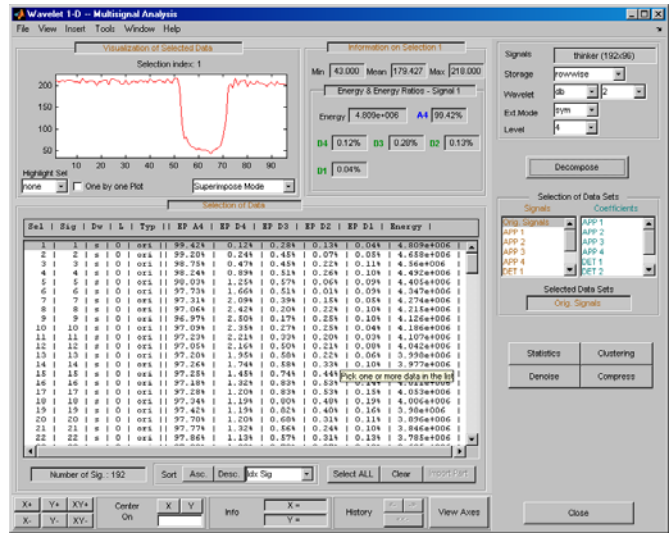
6 Decompose a multisignal.

Perform an analysis at level 4 using the `db2` wavelet and the same file used in the command line section: `thinker.mat`.

In the upper right portion of the Wavelet 1-D Multisignal Analysis tool, select `db2` and level 4 in the Wavelet fields.

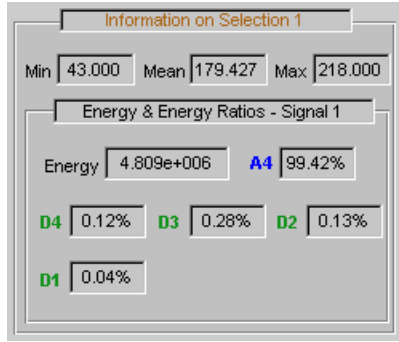


Click **Decompose**. After a pause for computation, all the original signals are decomposed and signal 1 is automatically selected



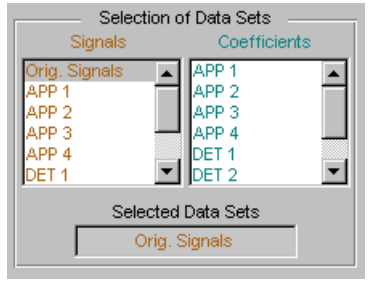
In the **Selection of Data** pane, new information is added for each original signal — the percentage of energy of the wavelet components (**D1**,...,**D4** and **A4**) and the total energy. The **Information on Selected Data** pane

contains information on the single selected signal: **Min**, **Mean**, **Max** and the energy distribution of the signal.



Since the original signals are decomposed, new objects appear and the **Selection of Data Sets** pane in the Command Frame updates.

The **Selection of Data Sets** pane defines the available signals that are now selectable from the **Selection of Data** pane.

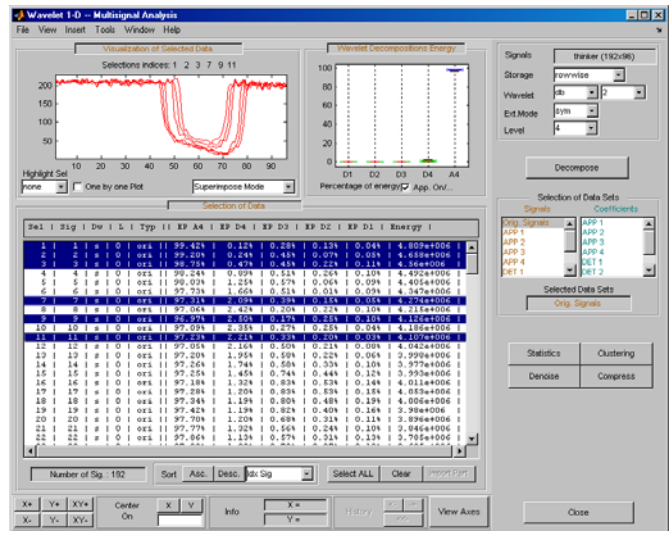


The list on the left allows you to select sets of signals and the right list allows you to select sets of corresponding coefficients: original signals (`Orig. Signals`), approximations (`APP 1`,...) and details from levels 1 to 4 (`DET 1`,...).

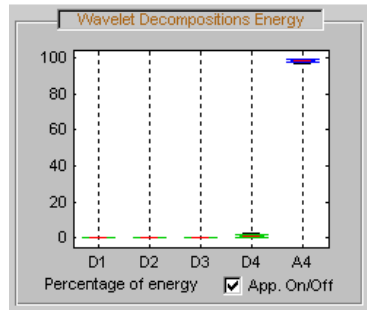
In the list on the right, the coefficients vectors can be of different lengths, but only components of the same length can be selected together.

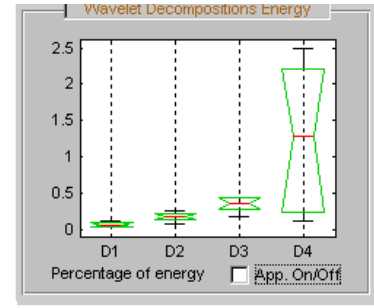
After a decomposition the original signals (`Orig. Signals`) data set appears automatically selected.

Select signals 1, 2, 3, 7, 9 and 11.



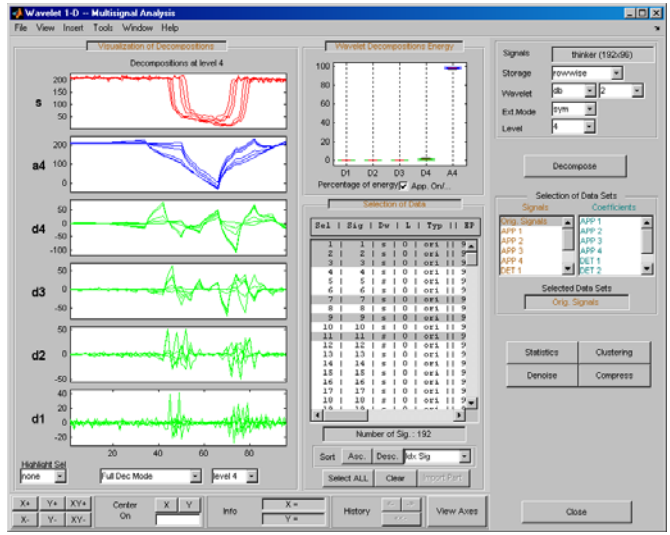
The energy of selected signals is primarily concentrated in the approximation `A4`, so the box plot is crushed (see following figure on the left). Deselect **App. On/Off** to see a better representation of details energy (see following figure on the right).





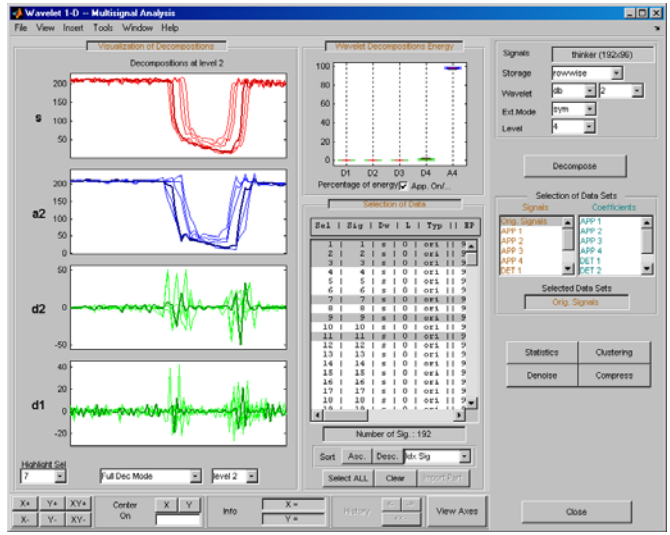
7 Display multisignal decompositions.

In the **Visualization of Selected Data** pane, change the view mode using the pop-up below the plots and select `Full Dec Mode`. The decompositions of the selected signals display.



Change the **Level** to 2.

Select the signal 7 in **Highlight Sel.**

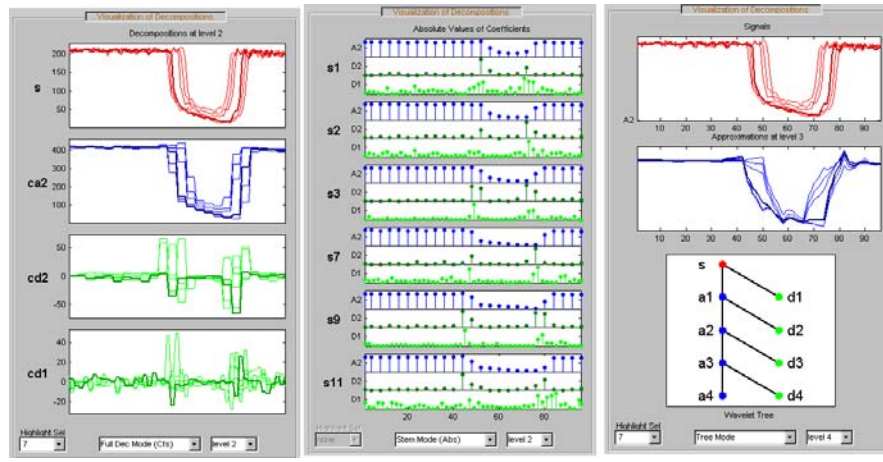


8 Change the visualization modes.

Using the second pop-up from the left at the bottom of the pane, select `Full Dec Mode (Cfs)`. The coefficients of the decompositions of the selected signals display. At level k, coefficients are duplicated 2k times.

Change the view mode to `Stem Mode (Abs)`, and then, change to `Tree Mode`. The wavelet tree corresponding to the decompositions of the selected signals displays.

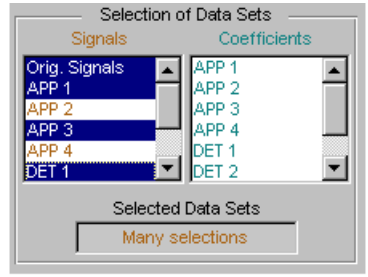
Select the level 4 and click the node a3. Then highlight signal 7.



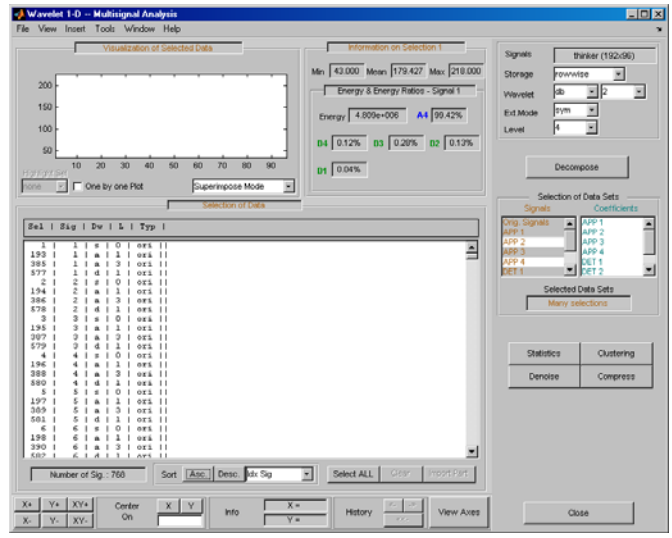
9 Select Different Wavelet Components.

Ctrl-click **Orig. Signals**, **APP 1**, **APP 3** and **DET 1** to select these four sets of signals from the list on the left in the **Selection of Data Sets** pane.

The total number of selected data (`Number of Sig.`) appears in the **Selection of Data Sets** pane: four sets of 192 signals each is a total of 768 signals.

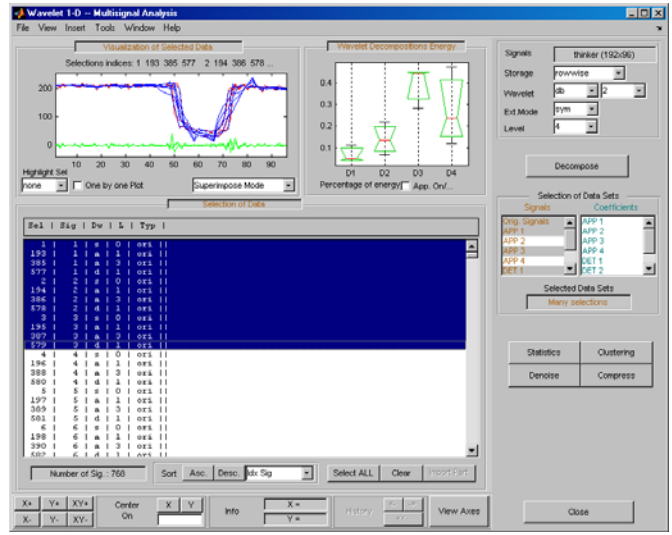


Click the **Asc**. button in the **Sort** pane. The selected data are sorted in ascending order with respect to the `Idx Sig` parameter

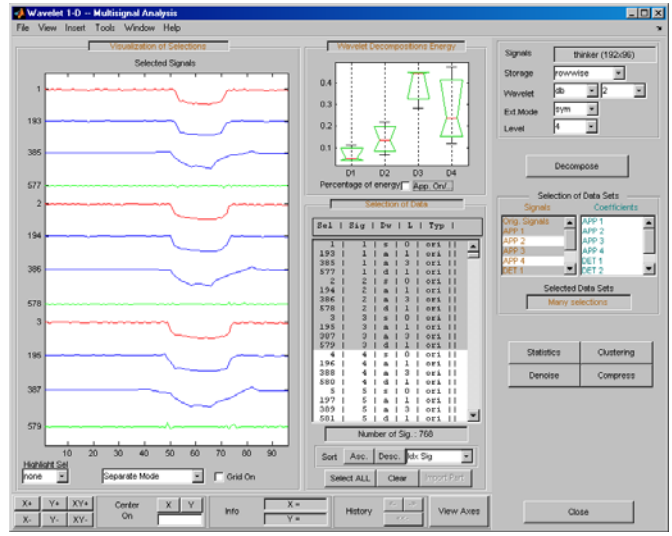


Note that DWT attributes of each selectable signal have been updated where **a** stands for approximation, **d** for detail and **s** for signal.

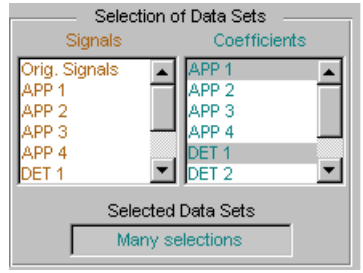
Click the `Idx Sel 1` signal and then shift-click the `Idx Sel 579` signal.



Choose `Separate Mode`.



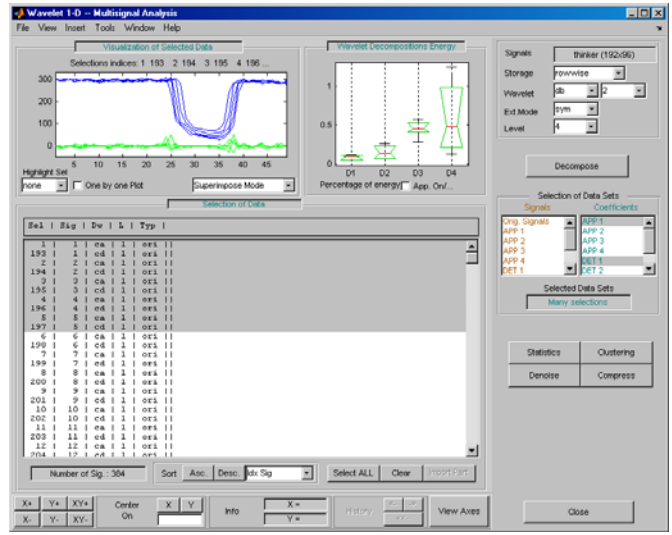
Ctrl-click to select two sets of signals from the right-most list of the **Selection of Data Sets** pane: `APP 1` and `DET 1`.



Note that in this list of coefficients sets, the selected vectors must be of same length, which means that you must select components of the same level.

Click the **Asc.** button in the Sort pane. The selected data are sorted in ascending order with respect to Idx Sig parameter.

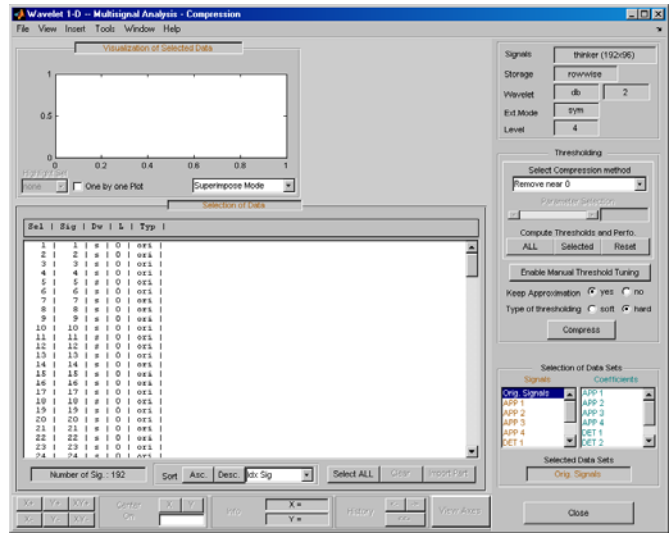
Select the ten first signals.



10 Compress a multisignal.

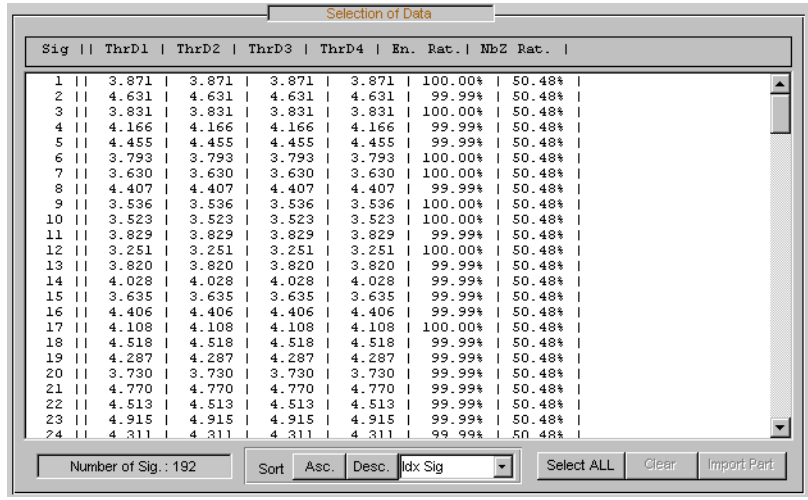
The graphical interface tools feature a compression option with automatic or manual thresholding.

Cick **Compress**, located in the lower-right side of the window. This displays the Compression window.



Note The tool always compresses all the original signals when you click the **Compress** button.

Before compressing, choose the particular strategy for computing the thresholds. Select the adapted parameters in the **Select Compression Method** frame. Then, apply this strategy to compute the thresholds according to the current method, either to the current selected signals by clicking the **Selected** button, or to all signals by clicking the **ALL** button. For this example, accept the defaults and click the **ALL** button.



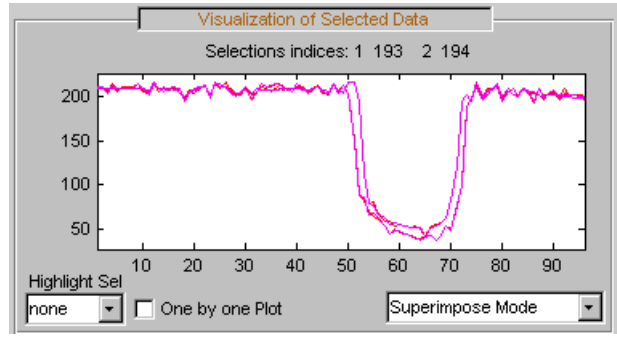
The thresholds for each level (`ThrD1` to `ThrD4`), the energy ratio (`En. Rat.`) and the sparsity ratio (`NbZ Rat.`) are displayed in the **Selection of Data** pane.

Click the **Compress** button at the bottom of the **Thresholding** pane. Now you can select new data sets: compressed Signals, the corresponding approximations, details and coefficients.

Press the **Ctrl** key and click the **Compressed** item in the left list of the **Selection of Data Sets** pane. The original signals and their compressed versions are selected (2 x 192 = 384 signals).

Click the **Asc.** button at the bottom of the **Selection of Data** pane to sort the signals using `Idx Sig` number.

With the mouse, select the first four signals. They correspond to the original signals 1, 2 and the corresponding compressed signals 193, 194.

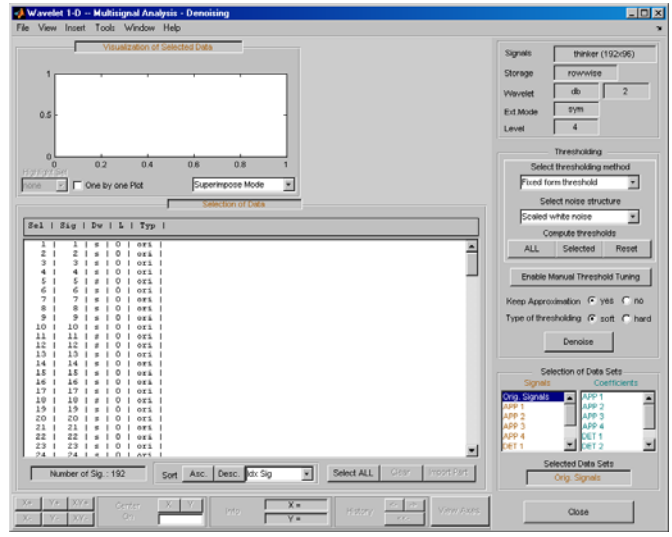


Click the **Close** button to close the Compression window.

11 Denoise a multisignal.

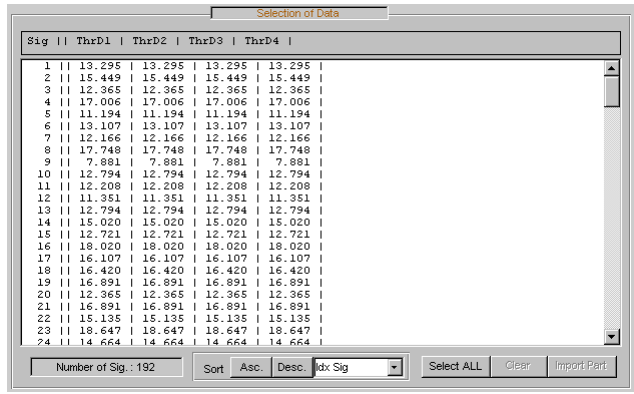
The graphical user interface offers a denoising option with either a predefined thresholding strategy or a manual thresholding method. Using this tool makes very easy to remove noise from many signals in one step.

Display the Denoising window by clicking the **Denoise** button located in the bottom part of the Command Frame on the right of the window.



A number of options are available for fine-tuning the denoising algorithm. For this example, accept the defaults: `soft` type of thresholding, `Fixed form threshold` method, and `Scaled white noise` as noise structure.

Click the **ALL** button in the Thresholding pane. The threshold for each level (`ThrD1`, ..., `ThrD4`) computes and displays in the **Selection of Data** pane.

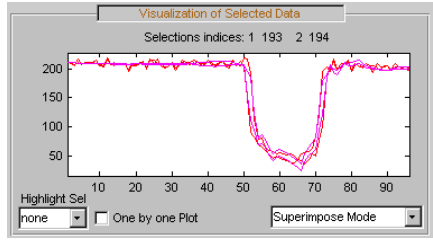


Then click the **Denoise** button at the bottom of the **Thresholding** pane.

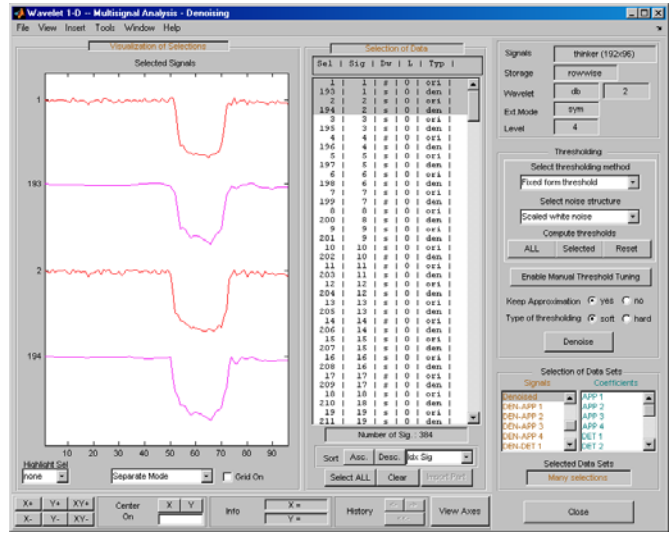
Ctrl-click the **Denoised** item in the list on the left of the **Selection of Data Sets** pane. The original signals and the corresponding denoised ones are selected (2 x 192 = 384 signals).

Click the **Asc.** button at the bottom of the **Selection of Data** pane to sort the signals according to the Idx Sig parameter.

With the mouse, select the first four signals. They correspond to the original signals 1, 2 and the corresponding denoised signals 193, 194



Choose `Separate Mode`.

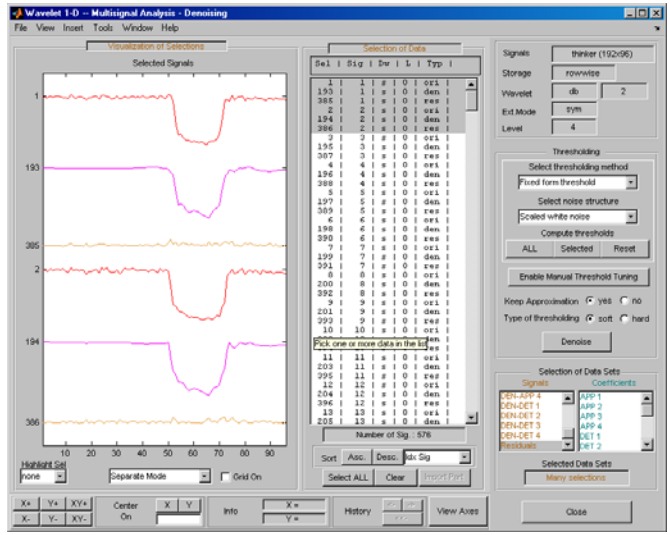


12 To view residuals, Ctrl-click the `Orig. Signal`, the `Denoised` and the `Residuals` items in the list on the left of the **Selection of Data Sets** pane. Original, denoised and residual signals are selected (3 x 192 = 576 signals).

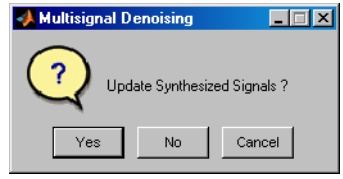
Click the **Asc.** button at the bottom of the **Selection of Data** pane to sort the signals using the `Idx Sig` parameter.

With the mouse, select the first six signals. They correspond to the original signals 1, 2, the corresponding denoised signals 193, 194 and the residuals 385, 386.

Then, choose `Separate Mode`.

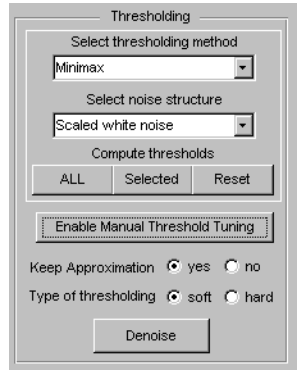


13 Click **Close** to close the denoising tool. Then, click the **Yes** button to update the synthesized signals.

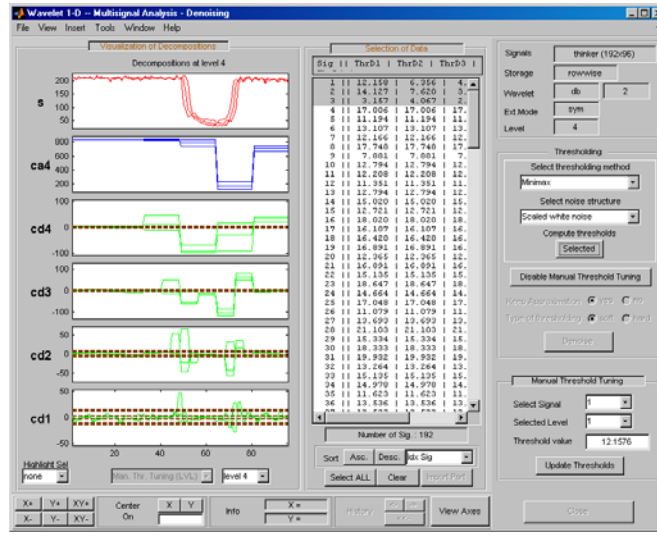


Manual Threshold Tuning

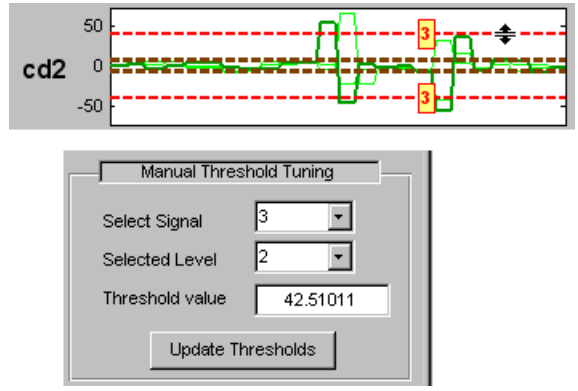
1 Choose a method, select one or several signals in the **Selection of Data** pane using the mouse and keys. Then click the **Selected** button. You can select another group of signals using the same method. Press the **Denoise** button to denoise the selected signal(s).



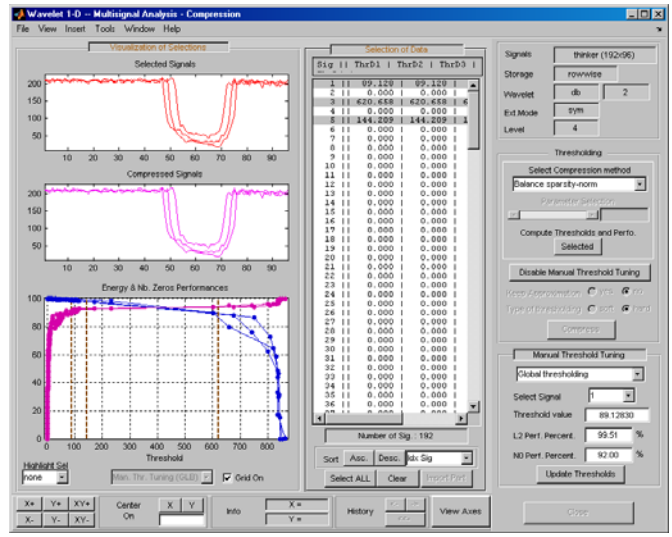
You can also use manual threshold tuning. Click the **Enable Manual Thresholding Tuning** button.



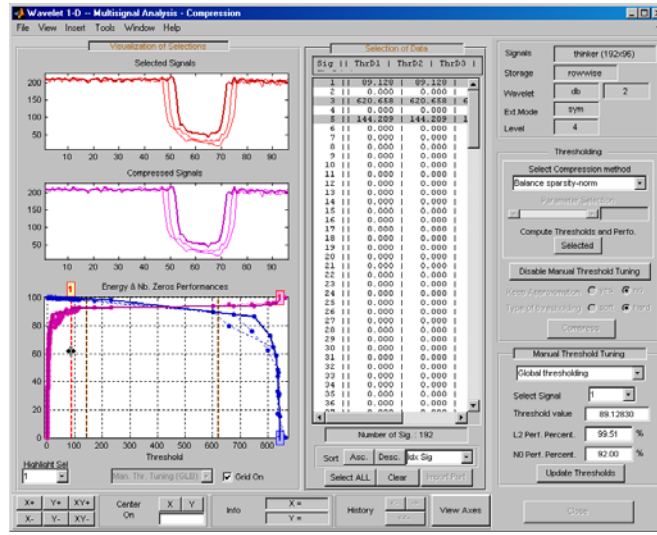
The horizontal lines in the wavelet coefficient axes (`cd1`, ..., `cd4`) can be dragged using the mouse. This may be done individually, by group or all together depending on the values in the **Select Signal** and **Selected Level** fields in the **Manual Threshold Tuning** pane.



2 In the Wavelet 1-D Multisignal Analysis Compression tool, you can use two methods for threshold tuning: the **By level thresholding** method which is used in the Wavelet 1-D Multisignal Analysis Denoising tool, and the **Global thresholding method**.



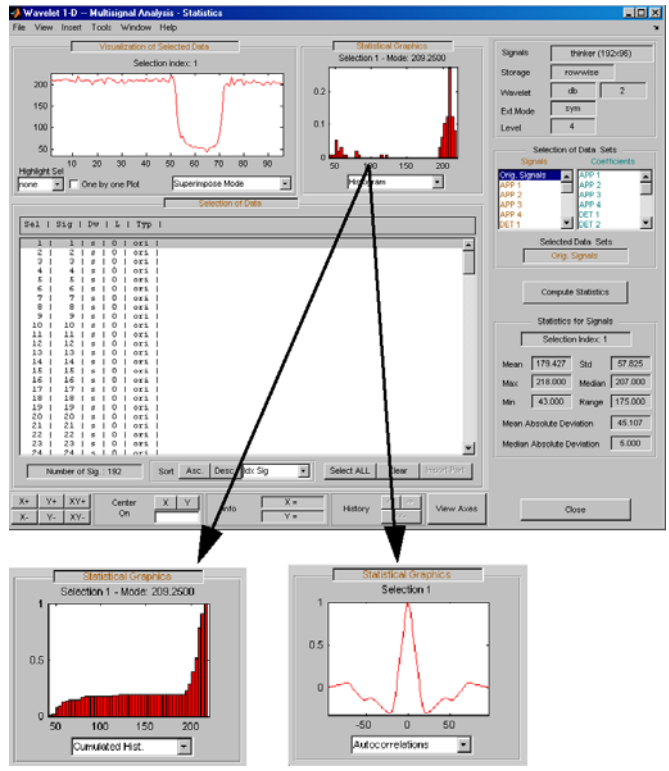
You can drag the vertical lines in the **Energy and Nb. Zeros Performances** axes using the mouse. This can be done individually or all together depending on the values of **Select Signal** in the **Manual Threshold Tuning** pane.



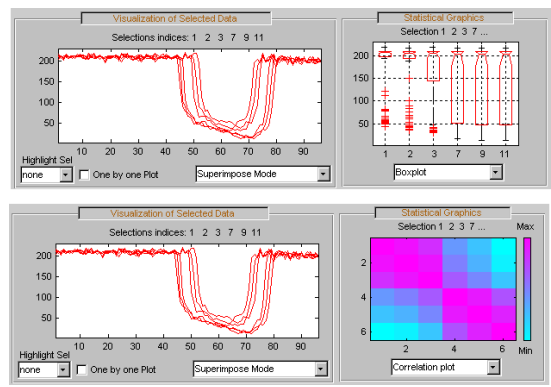
The threshold value, L2 performance, and number of zeros performance are updated in the corresponding edit buttons in the **Manual Threshold Tuning** pane.

Statistics on Signals

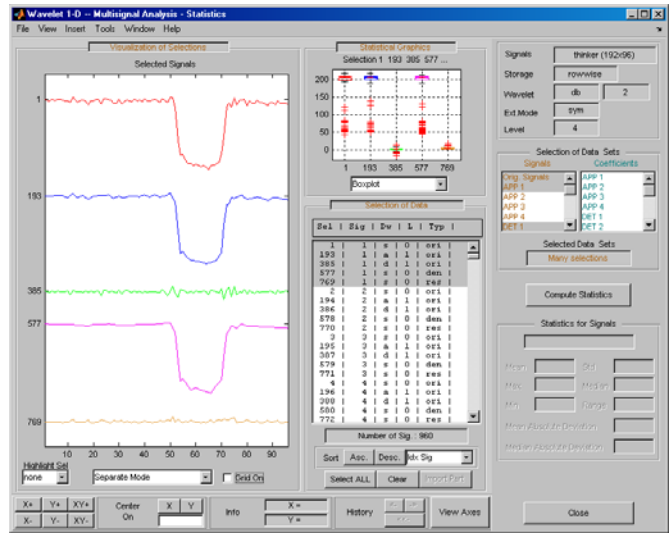
1 You can display various statistical parameters related to the signals and their components. From the Wavelet 1-D Multisignal Analysis tool, click the **Statistics** button. Then select the signal 1 in the **Selection of Data Sets** pane.



Select the signals 1, 2, 3, 7, 9 and 11 in the **Selection of Data** pane, and display the corresponding boxplots and correlation plots.



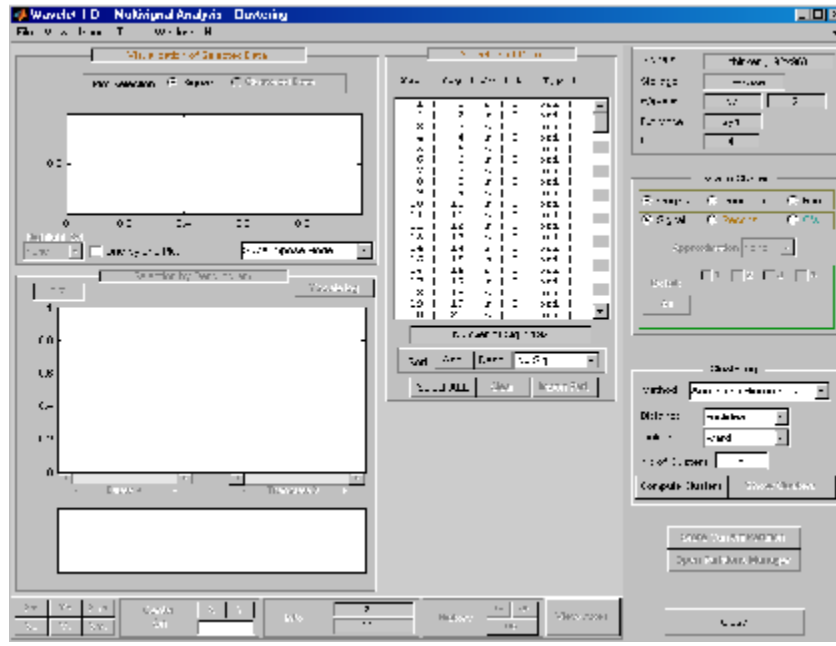
2 To display statistics on many wavelet components, in the **Selection Data Sets** pane, in the left column, select `Orig. Signals`, `APP 1`, `DET 1`, Denoised and `Residuals` signals. Then choose `Separate Mode`, and click the **Asc.** button in the **Sort** pane. The selected data are sorted in ascending order with respect to `Idx Sig` parameter. In the **Selection of Data** pane, select data related to signal 1.



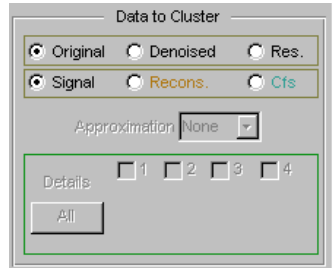
Clustering Signals

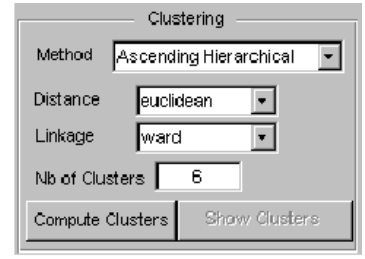
Note To use clustering, you must have Statistics Toolbox™ software installed.

1 Click the **Clustering** button located in the Command Frame, which is in the lower right of the Wavelet 1-D Multisignal Analysis window to open the Clustering tool.



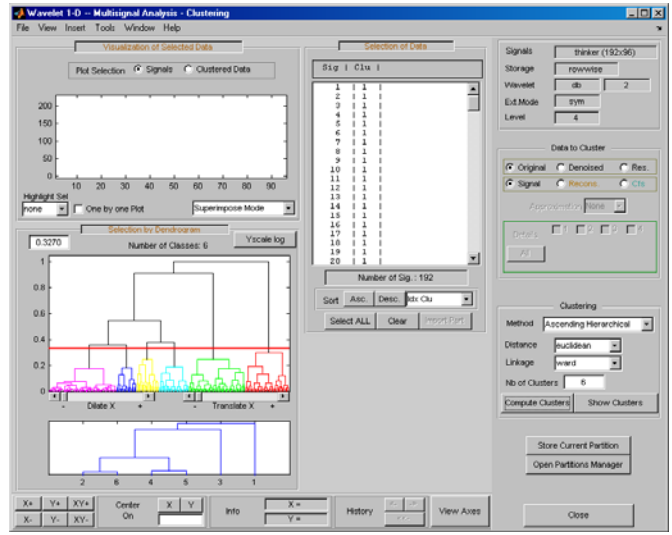
You can cluster various type of signals and wavelet components: original, denoised or compressed, residuals, and approximations or details (reconstructed or coefficients). Similarly, there are several methods for constructing partitions of data.





Use the default parameters (`Original` and `Signal` in Data to Cluster, and in `Ascending Hierarchical, euclidean, ward`, and `6` in Culstering) and click the **Compute Clusters** button.

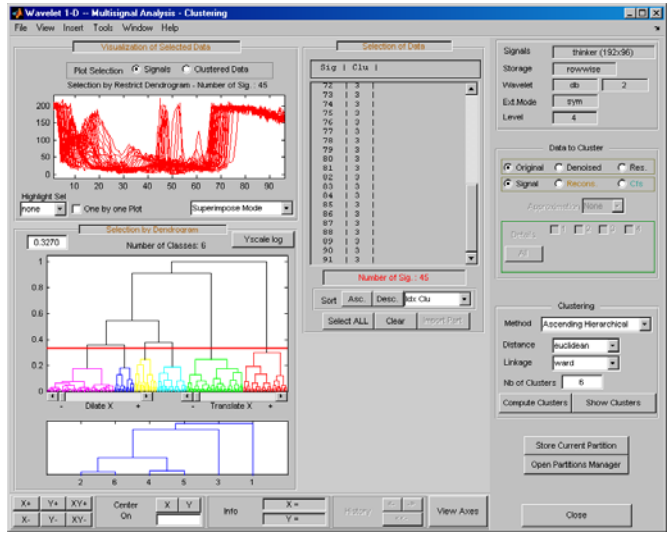
A full dendrogram and a restricted dendrogram display in the **Selection by Dendrogram** pane. For each signal, the cluster number displays in the **Selection of Data** pane.



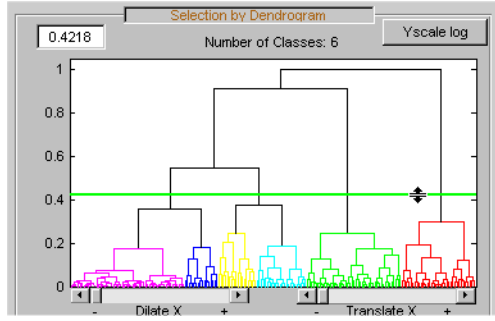
2 Select one cluster, several clusters, or a part of a cluster.

Click the `xticklabel 3` at the bottom of the restricted dendrogram. The links of the third cluster blink in the full dendrogram and the 24 signals of this class display in the **Visualization of Selected Data** pane. You can see their numbers in the **Selection of Data** pane.

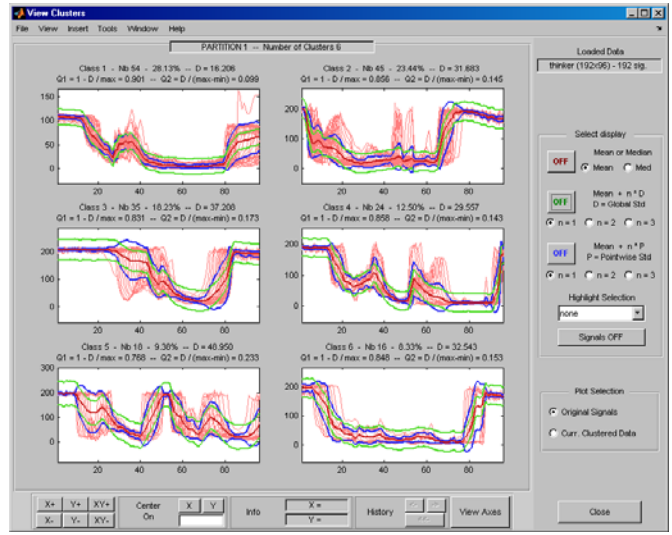
Clicking the line in the restricted or in the full dendrogram lets you select one cluster, several linked clusters, or a part of a cluster. For a more accurate selection, use the `Dilate X` and the `Translate X` sliders under the full dendrogram. You can also use the **Yscale** button located above the full dendrogram. The corresponding signals display in the **Visualization of Selected Data** pane and in the list of the **Selection of Data** pane.



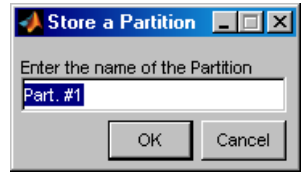
You can use the horizontal line in the full dendrogram to change the number of clusters. Use the left mouse button to drag the line up or down.



3 Use the **Show Clusters** button to examine the clusters of the current partition. You can display the mean (or the median) of each cluster, the global standard deviation and the pointwise standard deviation distance around the mean (or the median). The number of the cluster, the number of elements, the percentage of signals, and two indices of quality display for each cluster.

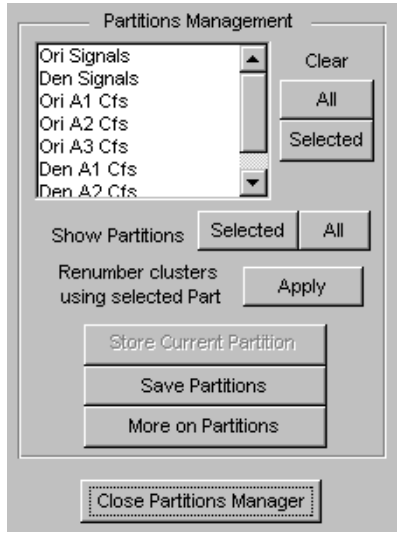


4 Click the **Store Current Partition** button below the **Clustering** pane to store the current partition for further comparisons. A default name is suggested. Note that the 1-D Wavelet Multisignal Analysis tool stores the partitions and they are not saved on the disk.



Partitions

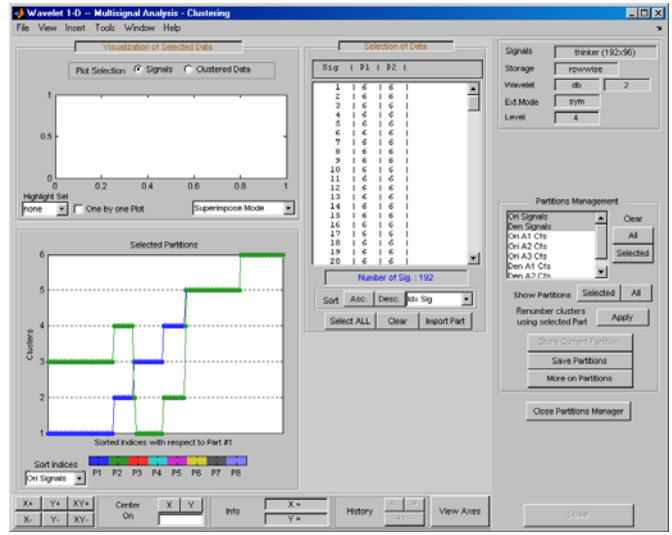
1 Build and store several partitions (for example, partitions with signals, denoised signals approximations at level 1, 2 and 3, and denoised signals). Then, click the **Open Partition Manager** button below the **Store Current Partition** button. The **Partitions Management** pane appears. The names of all stored partitions are listed.



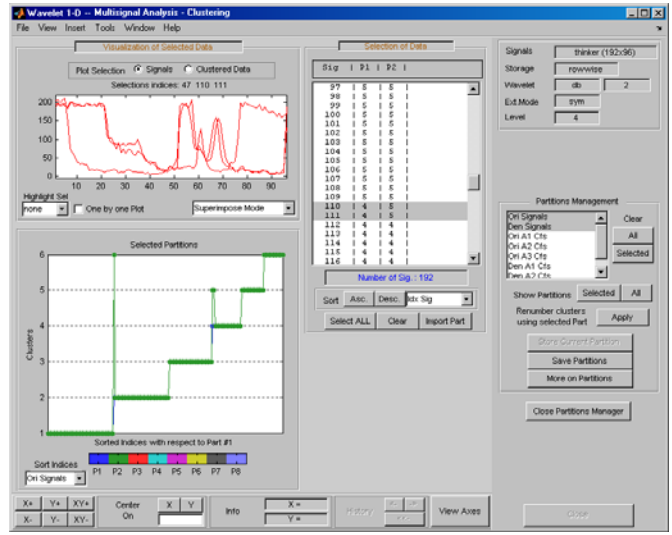
Now, you can show, clear, or save the partitions (individually, selected ones, or all together).

2 To display partitions, select the `Ori Signals` and the `Den Signals` partitions, and click the **Selected** button next to the **Show Partitions** label.

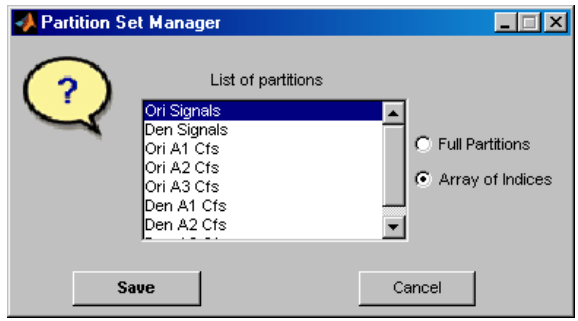
The clusters are almost the same, but it is difficult to see this on the **Selected Partitions** axis, due to the scaling difference. Press the **Apply** button to renumber the clusters (starting from the selected partition as basic numbering) to compare the two partitions.



Only three signals are not classified in the same cluster for the two considered partitions.

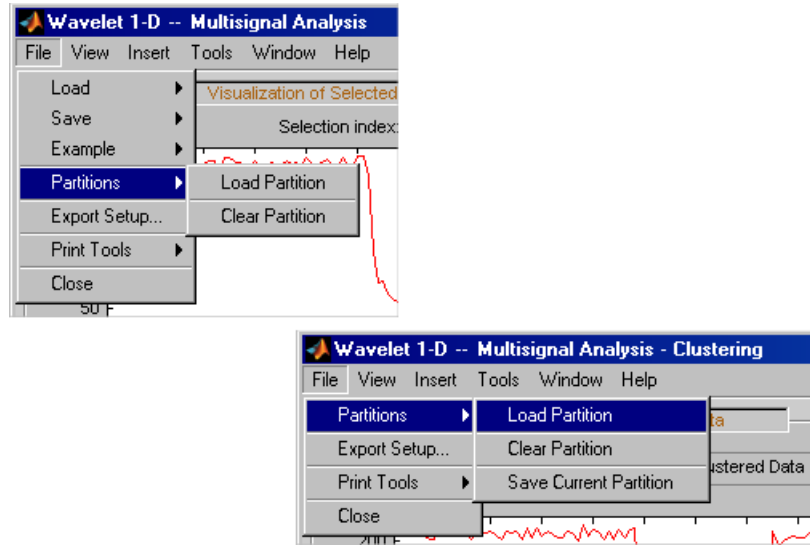


3 Select the partitions you want to save and click the **Save Partitions** button below the **Store Current Partition** button in the **Partitions Management** pane.

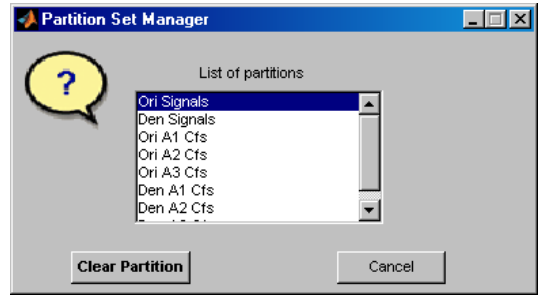


Partitions are saved as an array of integers, where each column corresponds to one partition and contains the indices of clusters. When you choose the `Full Partitions` option, an array object (`wpartobj`) is saved.

4 To load or clear stored partitions use **File > Partitions** in the Wavelet 1-D Multisignal Analysis tool. (**File > Partitions** is also available in the Wavelet 1-D Multisignal Analysis Clustering tool and you can also save the current partition.)



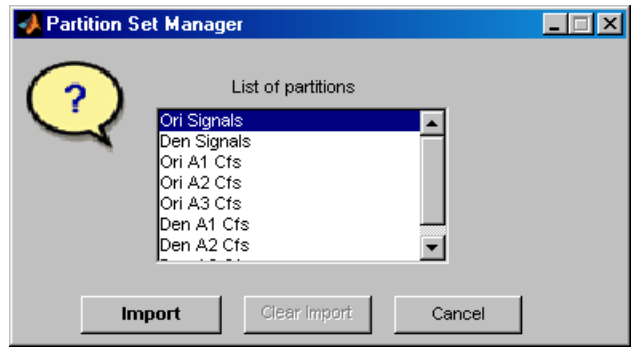
To clear one or more stored partitions, select **File > Partitions > Clear Partition**.



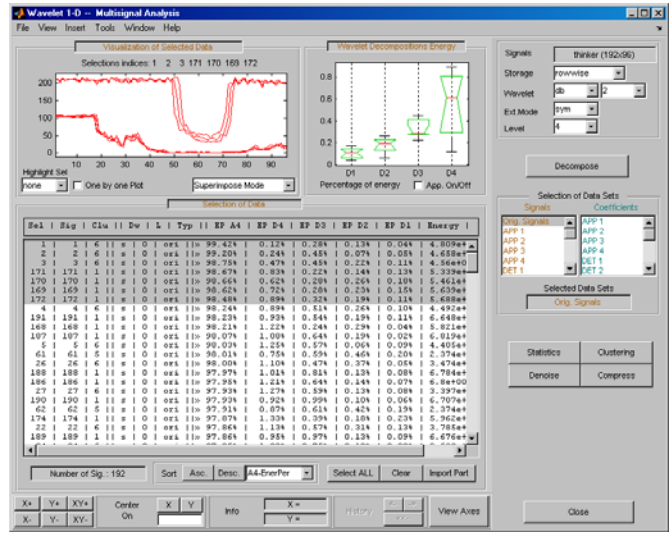
Select **File > Partitions > Load Partition** to load one or several partitions from the disk. The loaded partitions are stored in Wavelet 1-D Multisignal Analysis tool with any previously stored partitions. A partition can also be a manually created column vector.

Note The number of signals in loaded partitions must be equal to the number of signals in the Wavelet 1-D Multisignal Analysis tool. A warning appears if this condition is not true.

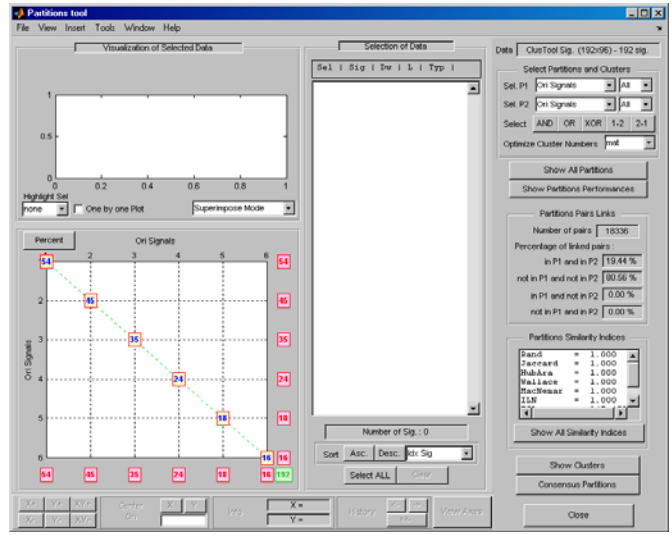
5 In each subcomponent of the Wavelet 1-D Multisignal Analysis tool (main, statistics, denoising, compression, clustering), you can import a stored partition from the list in the **Selection of Data** pane. Click the **Import Part** button at the bottom of the **Selection of Data** pane, the Partition Set Manager window appears. Select one partition and click the **Import** button.



For this example, go back to the main window, import the `Ori Signals` partition and sort the signals in descending order with respect to **A4** energy percentage.

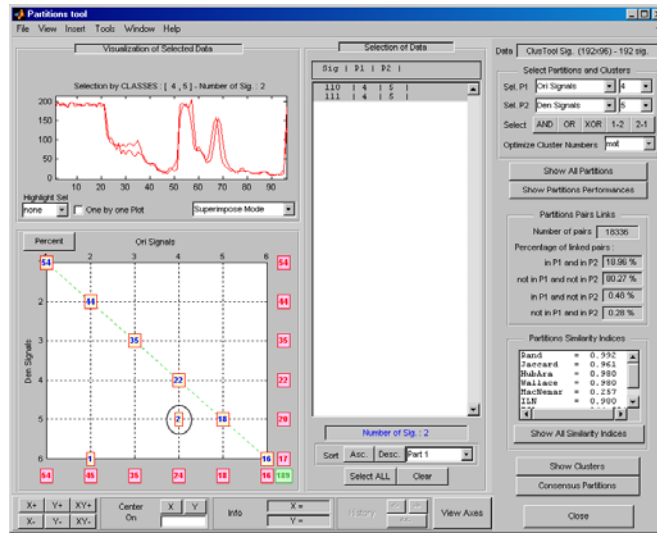


6 Click the **More on Partitions** button at the bottom of the Partitions Management pane to display the Partition tool.



7 Select the `Den Signals` in **Sel P2** in the upper-right corner of the window. Then, in the lower left axis, click the yellow text containing the value 2 (the

coordinates of the corresponding point are (4,5)). The corresponding signals are displayed together with all related information.

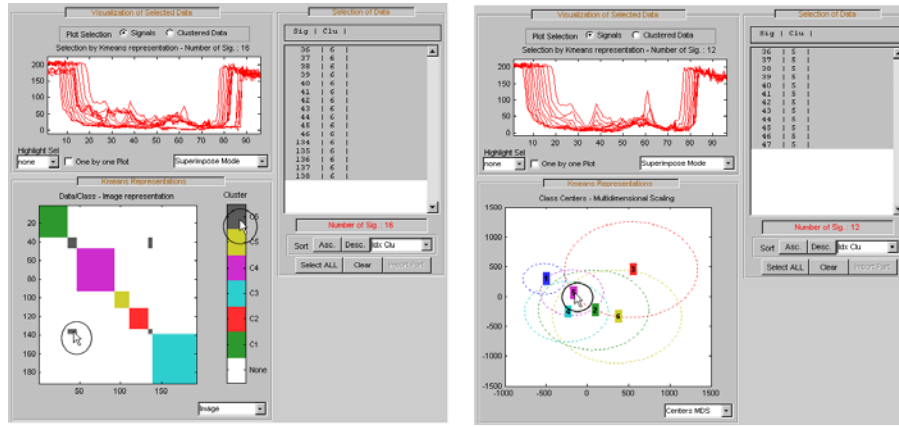


More on Clustering

Instead of the Ascending Hierarchical Tree clustering method, you can use the K-means method. For this case, the partition cannot be represented by a dendrogram and other representations should be used.

In the image representation (see figure below on the left), you can select a cluster by clicking on the corresponding color on the colorbar. You can also select a cluster or part of a cluster by clicking on the image.

In the center representation (see figure below on the right) you can select a cluster by clicking on the corresponding colored center.

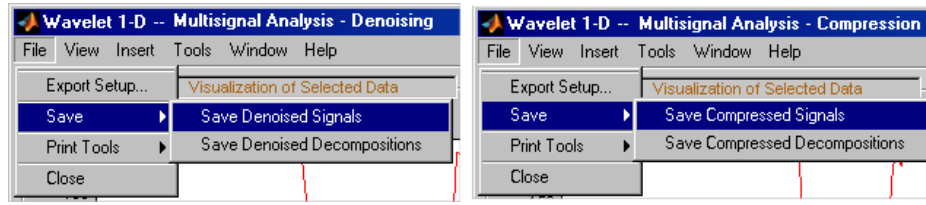


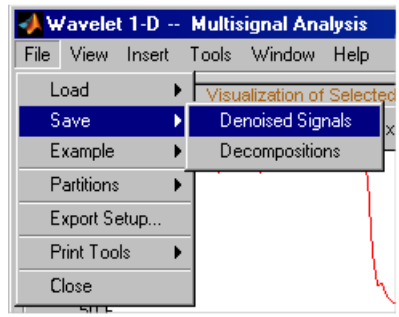
Importing and Exporting Information from the Graphical Interface

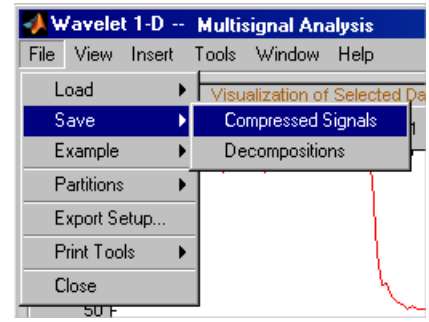
The Wavelet 1-D Multisignal Analysis tool lets you move data to and from disk.

Saving Information to Disk

You can save decompositions and denoised or compressed signals (including the corresponding decompositions from Wavelet 1-D Multisignal Analysis tools) to disk. You then can manipulate the data and later import it again into the graphical tools.







Saving Decompositions

The Wavelet 1-D Multisignal Analysis main tool lets you save the entire set of data from a wavelet analysis to disk. The toolbox creates a MAT-file in the current folder with a name you choose.

1. Open the Wavelet 1-D Multisignal Analysis main tool and load the example analysis by selecting **File > Example > Ex 21: Thinker (rows)**.

2. Save the data from this analysis, using the menu option **File > Save Decompositions**.

   A dialog box appears that lets you specify a folder and filename for storing the decomposition data. For this example, use the name `decORI.mat`.

3. Type the name `decORI`.

4. After saving the decomposition data to the file `decORI.mat`, load the variables into your workspace:

```
load decORI
whos
```

Name	Size	Bytes	Class
dec	1x1	163306	struct

```
dec
dec =
        dirDec: 'r'
```

```
          level: 4
          wname: 'db2'
     dwtFilters: [1x1 struct]
        dwtEXTM: 'sym'
       dwtShift: 0
       dataSize: [192 96]
             ca: [192x8 double]
             cd: {1x4 cell}
```

The field `ca` of the structure `dec` gives the coefficients of approximation at level 4, the field `cd` is a cell array which contains the coefficients of details.

```
size(dec.cd{1})
ans =
   192    49
size(dec.cd{2})
ans =
   192    26
size(dec.cd{3})
ans =
   192    14
size(dec.cd{4})
ans =
   192     8
```

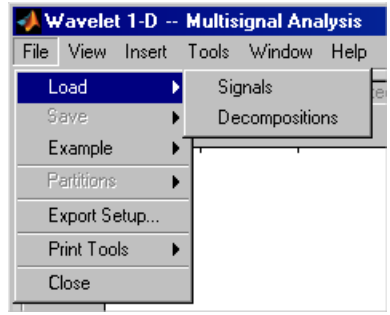
You can change the coefficients using the `chgwdeccfs` function.

Note For a complete description of the `dec` structure, see "Loading Decompositions" on page 2-291.

Loading Information into the Wavelet 1-D Multisignal Analysis Tool

You can load signals or decompositions into the graphical interface. The information you load may be previously exported from the graphical interface, and then manipulated in the workspace; or it may be information you initially generated from the command line. In either case, you must observe the strict

file formats and data structures used by the Wavelet 1-D Multisignal Analysis tools or errors will occur when you try to load information.



Loading Signals. To load a signal you constructed in your MATLAB workspace into the Wavelet 1-D Multisignal Analysis tool, save the signal in a MAT-file (with extension `.mat`).

For example, if you design a signal called `magic128` and want to analyze it in the Wavelet 1-D Multisignal Analysis tool, type

```
save magic128 magic128
```

Note The workspace variable `magic128` must be a matrix and the number of rows and columns must be greater than 1.

```
sizmag = size(magic128)

sizmag =
    128   128
```

To load this signal into the Wavelet 1-D Multisignal Analysis tool, use the **File > Load Signal** menu item. A dialog box appears in which you select the appropriate MAT-file to be loaded.

Note When you load a matrix of signals from the disk, the name of 2-D variables are inspected in the following order: `x`, `X`, `sigDATA`, and `signals`. Then, the 2-D variables encountered in the file are inspected in alphabetical order.

Loading Decompositions. To load decompositions that you constructed in the MATLAB workspace into the Wavelet 1-D Multisignal Analysis tool, save the signal in a MAT-file (with extension `mat`).

For instance, if you design a signal called `magic128` and want to analyze it in the Wavelet 1-D Multisignal Analysis too, the structure `dec` must have the following fields:

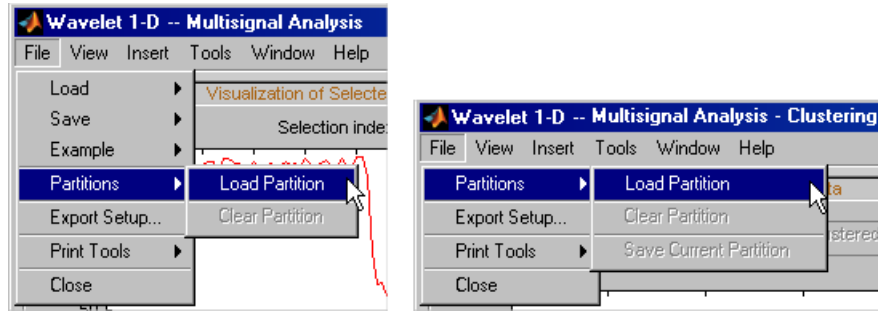
`'dirDec'`	Direction indicator with `'r'` for row or `'c'` for column
`'level'`	Level of DWT decomposition
`'wname'`	Wavelet name
`'dwtFilters'`	Structure with four fields: `LoD`, `HiD`, `LoR`, `HiR`
`'dwtEXTM'`	DWT extension mode (see `dwtmode`)
`'dwtShift'`	DWT shift parameter (0 or 1)
`'dataSize'`	Size of original matrix `X`
`'ca'`	Approximation coefficients at level `dec.level`
`'cd'`	Cell array of detail coefficients, from 1 to `dec.level`

The coefficients `cA` and `cD{k}`, for (`k = 1` to `dec.level`), are matrices and are stored rowwise if `dec.dirDec` is equal to `'r'` or columnwise if `dec.dirDec` is equal to `'c'`.

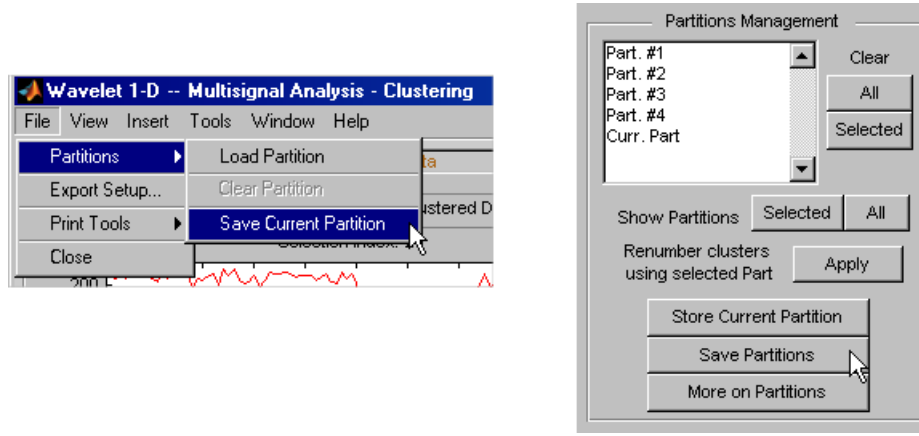
Note The fields `'wname'` and `'dwtFilters'` have to be compatible (see the `wfilters` function). The sizes of `cA` and `cD{k}`, (for `k = 1` to `dec.level`) must be compatible with the direction, the level of the decomposition, and the extension mode.

Loading and Saving Partitions.

Loading. The Wavelet 1-D Multisignal Analysis main tool and clustering tool let you load a set of partitions from disk.



Saving Partitions. The Wavelet 1-D Multisignal Analysis clustering tool lets you save a set of partitions to disk.



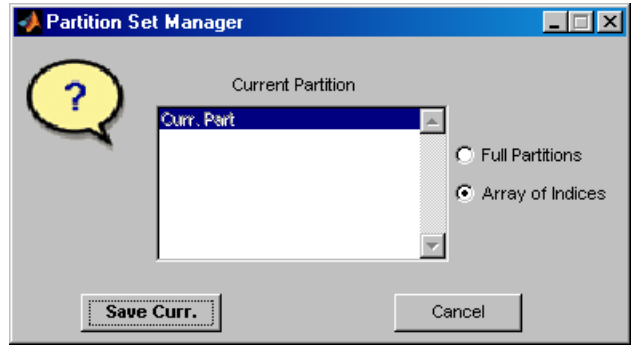
For example:

1. Open the Wavelet 1-D Multisignal Analysis main tool and load the example analysis using **File > Example > Ex 21: Thinker (rows)**.

2. Click the **Clustering** button. The Wavelet 1-D Multisignal Analysis Clustering window appears.

3. Click the **Compute Clusters** button, and then save the current partition using menu option **File > Partitions > Save Current Partition**. A dialog box appears that lets you specify the type of data to save.

4. Click the **Save Curr.** button.



5. Another dialog box appears that lets you specify a folder and filename for storing the partition data. Type the name `curPART`.

6. After saving the partition data to the file `curPART.mat`, load the variables into your workspace:

```
load curPART
whos
```

Name	Size	Bytes	Class
tab_IdxCLU	192x1	1536	double

7 You can modify the array `tab_IdxCLU` in the workspace, and save the partition data in a file. For example to create two new partitions with four and two clusters, type the following lines:

```
tab_IdxCLU(:,2) = rem((1:192)',4) + 1;
tab_IdxCLU(:,3) = double((1:192)'>96) + 1;
save newpart tab_IdxCLU
```

Now you can use three partitions for the example Ex 21: Thinker (rows). Then, you can load the partitions stored in the file `newPART.mat` in the Wavelet 1-D Multisignal Analysis main tool and clustering tool.

Note A partition is a column vector of integers. The values must vary from `1` to `NbClusters` (`NbClusters > 1`), and each cluster must contain at least one element. The number of rows must be equal to the number of signals.

Two-Dimensional True Compression

This section takes you through the features of two-dimensional true compression using the Wavelet Toolbox software.

For more information on the compression methods see "True Compression for Images" in the *Wavelet Toolbox User's Guide*.

For more information on the main function available when using command-line mode, see the `wcompress` reference pages.

Starting from a given image, the goal of the true compression is to minimize the length of the sequence of bits needed to represent it, while preserving information of acceptable quality. Wavelets contribute to effective solutions for this problem.

The complete chain of compression includes phases of quantization, coding and decoding in addition of the wavelet processing itself.

The purpose of this section is to show how to compress and uncompress a grayscale or truecolor image using various compression methods.

In this section, you'll learn to

- Compress using global thresholding and Huffman encoding
- Uncompress
- Compress using progressive methods
- Handle truecolor images

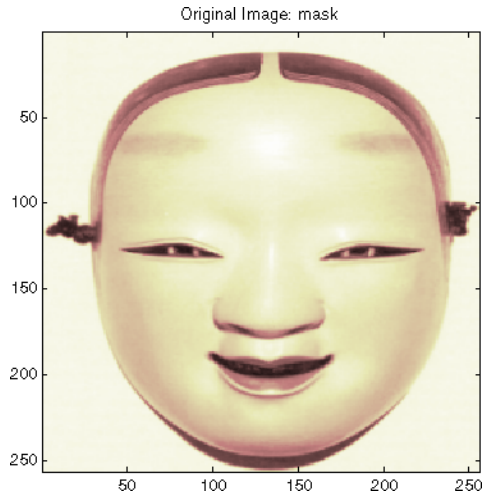
Two-Dimensional True Compression Using the Command Line

Compression by Global Thresholding and Huffman Encoding

First load and display the grayscale image mask.

```
load mask;
image(X)
```

```
axis square;
colormap(pink(255))
title('Original Image: mask')
```



A synthetic performance of the compression is given by the compression ratio and the Bit-Per-Pixel ratio which are equivalent.

The compression ratio CR means that the compressed image is stored using only CR% of the initial storage size.

The Bit-Per-Pixel ratio BPP gives the number of bits used to store one pixel of the image.

For a grayscale image, the initial BPP is 8 while for a truecolor image the initial BPP is 24 because 8 bits are used to encode each of the three colors (RGB color space).

The challenge of compression methods is to find the best compromise between a weak compression ratio and a good perceptual result.

Let us begin with a simple method cascading global coefficients thresholding and Huffman encoding. We use the default wavelet *bior4.4* and the default level which is the maximum possible level (see the `wmaxlev` function) divided by 2.

The desired Bit-Per-Pixel ratio BPP is set to 0.5 and the compressed image will be stored in the file named 'mask.wtc'.

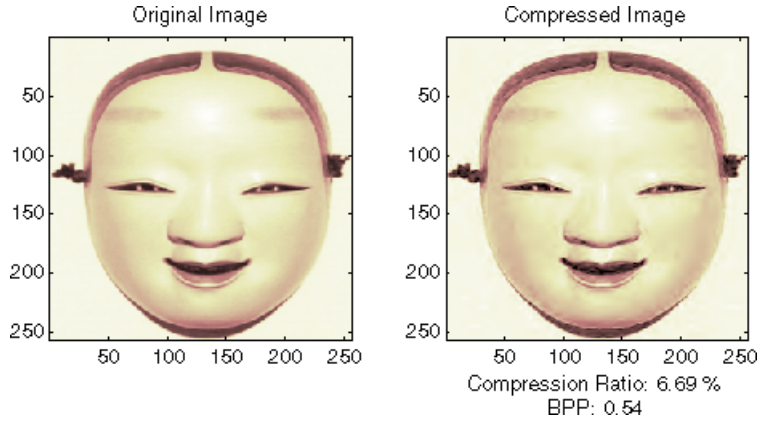
```
meth = 'gbl_mmc_h'; % Method name
option = 'c';         % 'c' stands for compression
[CR,BPP] = wcompress(option,X,'mask.wtc',meth,'bpp',0.5)

CR =

  6.6925

BPP =

  0.5354
```

The achieved Bit-Per-Pixel ratio is actually about 0.53 (closed to the desired one) for a compression ratio of 6.7%.

Uncompression

Let us uncompress the image retrieved from the file 'mask.wtc' and compare it to the original image.

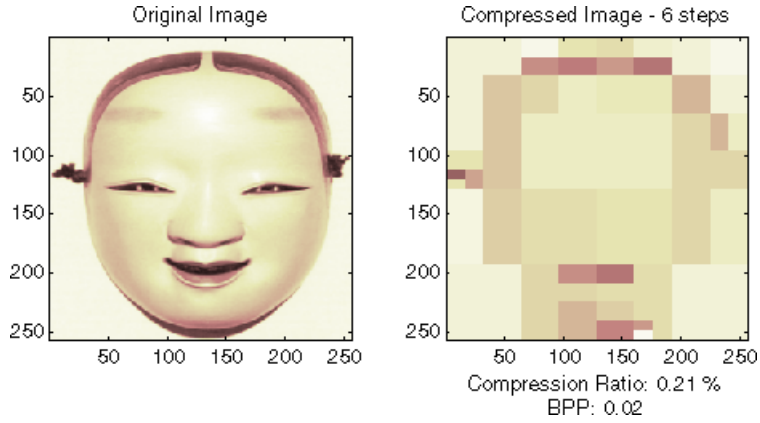
```
option = 'u';  % 'u' stands for uncompression
Xc = wcompress(option,'mask.wtc');
colormap(pink(255))
subplot(1,2,1); image(X);
axis square;
title('Original Image')
subplot(1,2,2); image(Xc);
axis square;
title('Compressed Image')
xlabel({['Compression Ratio: ' num2str(CR,'%1.2f %%')], ...
        ['BPP: ' num2str(BPP,'%3.2f')]})
```



Compression by Progressive Methods

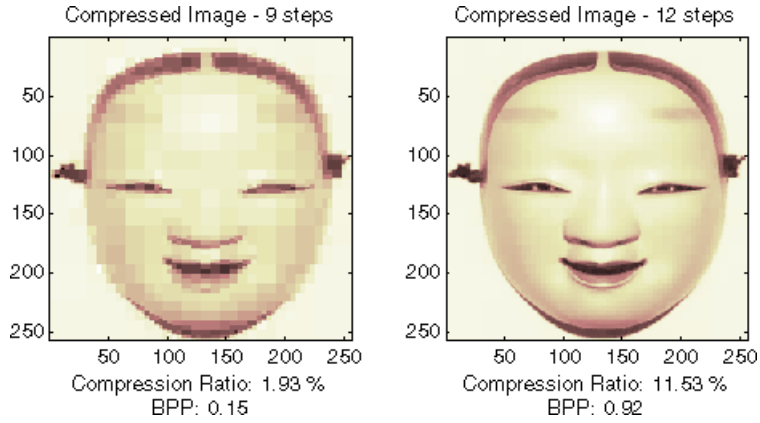
Let us now illustrate the use of progressive methods starting with the well known EZW algorithm using the Haar wavelet. The key parameter is the number of loops. Increasing it, leads to better recovery but worse compression ratio.

```
meth = 'ezw';     % Method name
wname  = 'haar';  % Wavelet name
nbloop = 6;       % Number of loops
[CR,BPP] = wcompress('c',X,'mask.wtc',meth, ...
                     'maxloop',nbloop,'wname',wname);
Xc = wcompress('u','mask.wtc');
colormap(pink(255))
subplot(1,2,1); image(X);
axis square;
title('Original Image')
subplot(1,2,2); image(Xc);
axis square; title('Compressed Image - 6 steps')
xlabel({['Compression Ratio: ' num2str(CR,'%1.2f %%')], ...
        ['BPP: ' num2str(BPP,'%3.2f')]})
```



A too small number of steps (here 6) produces a very coarse compressed image. So let us examine a little better result for 9 steps and a satisfactory result for 12 steps.

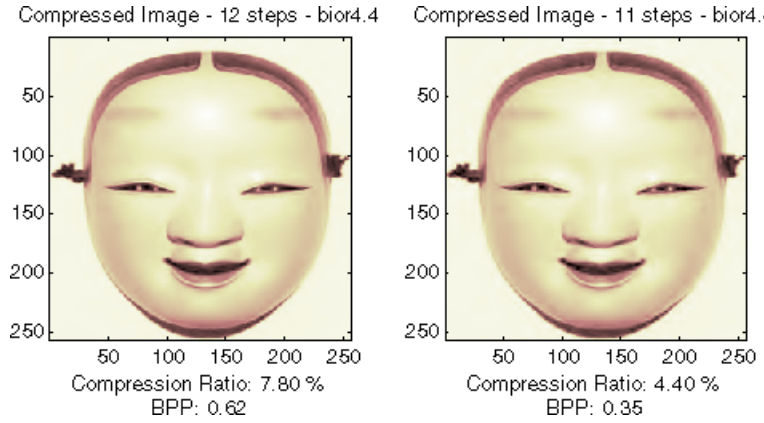
```
[CR,BPP]= wcompress('c',X,'mask.wtc',meth,'maxloop',9, ...
                    'wname','haar');
Xc = wcompress('u','mask.wtc');
colormap(pink(255))
subplot(1,2,1); image(Xc);
axis square; title('Compressed Image - 9 steps')
xlabel({['Compression Ratio: ' num2str(CR,'%1.2f %%')],...
        ['BPP: ' num2str(BPP,'%3.2f')]})

[CR,BPP] = wcompress('c',X,'mask.wtc',meth,'maxloop',12, ...
                     'wname','haar');
Xc = wcompress('u','mask.wtc');
subplot(1,2,2); image(Xc);
axis square;
title('Compressed Image - 12 steps')
xlabel({['Compression Ratio: ' num2str(CR,'%1.2f %%')],...
        ['BPP: ' num2str(BPP,'%3.2f')]})
```



As can be seen, the reached BPP ratio is about 0.92 when using 12 steps.

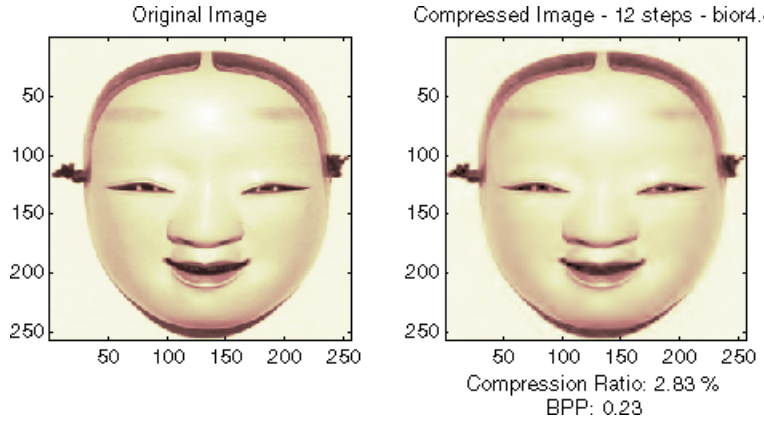
Let us try to improve it by using the wavelet *bior4.4* instead of *haar* and looking at obtained results for steps 12 and 11.

```
[CR,BPP] = wcompress('c',X,'mask.wtc','ezw','maxloop',12, ...
                     'wname','bior4.4');
Xc = wcompress('u','mask.wtc');
colormap(pink(255))
subplot(1,2,1); image(Xc);
axis square;
title('Compressed Image - 12 steps - bior4.4')
xlabel({['Compression Ratio: ' num2str(CR,'%1.2f %%')], ...
        ['BPP: ' num2str(BPP,'%3.2f')]})
[CR,BPP] = wcompress('c',X,'mask.wtc','ezw','maxloop',11, ...
                     'wname','bior4.4');
Xc = wcompress('u','mask.wtc');
subplot(1,2,2); image(Xc);
axis square;
title('Compressed Image - 11 steps - bior4.4')
xlabel({['Compression Ratio: ' num2str(CR,'%1.2f %%')], ...
        ['BPP: ' num2str(BPP,'%3.2f')]})
```



Starting from the eleventh loop, the result can be considered satisfactory. The reached BPP ratio is now about 0.35. It can even be slightly improved by using a more recent method: SPIHT (Set Partitioning In Hierarchical Trees).

```
[CR,BPP] = wcompress('c',X,'mask.wtc','spiht','maxloop',12, ...
                     'wname','bior4.4');
Xc = wcompress('u','mask.wtc');
colormap(pink(255))
subplot(1,2,1); image(X);
axis square;
title('Original Image')
subplot(1,2,2); image(Xc);
axis square;
title('Compressed Image - 12 steps - bior4.4')
xlabel({['Compression Ratio: ' num2str(CR,'%1.2f %%')], ...
        ['BPP: ' num2str(BPP,'%3.2f')]})
[psnr,mse] = psnr_mse_maxerr(X,Xc)

delete('mask.wtc')
```



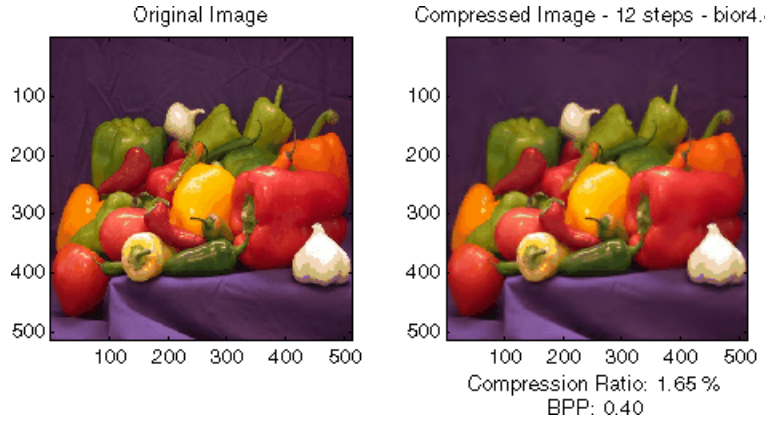
The final compression ratio (2.8%) and the Bit-Per-Pixel ratio (0.23) are very satisfactory. Let us recall that the first ratio means that the compressed image is stored using only 2.8% of the initial storage size.

Handling Truecolor Images

Finally, let us illustrate how to compress the `wpeppers.jpg` truecolor image. Truecolor images can be compressed along the same scheme as the grayscale images by applying the same strategies to each of the three color components.

The progressive compression method used is SPIHT (Set Partitioning In Hierarchical Trees) and the number of encoding loops is set to 12.

```
X = imread('wpeppers.jpg');
[CR,BPP] = wcompress('c',X,'wpeppers.wtc','spiht','maxloop',12);
Xc = wcompress('u','wpeppers.wtc');
subplot(1,2,1); image(X);
axis square;
title('Original Image')
subplot(1,2,2); image(Xc);
axis square;
title('Compressed Image - 12 steps - bior4.4')
xlabel({['Compression Ratio: ' num2str(CR,'%1.2f %%')], ...
        ['BPP: ' num2str(BPP,'%3.2f')]})
delete('wpeppers.wtc')
```



The compression ratio (1.65%) and the Bit-Per-Pixel ratio (0.4) are very satisfactory while maintaining a good visual perception.

Two-Dimensional True Compression Using the Graphical Interface

In this section, we explore the different methods for two-dimensional true compression, using the graphical interface tools.

1 Start the True Compression 2-D Tool.

From the MATLAB prompt, type

```
wavemenu
```

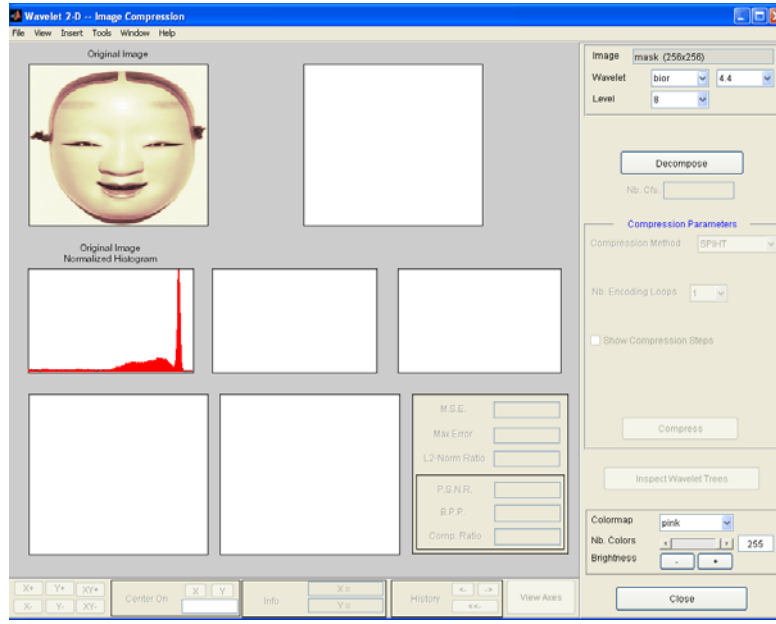
The **Wavelet Toolbox Main Menu** appears. Click the **True Compression 2-D menu** item. The true compression tool for images appears.

2 Load the image.

From the **File** menu, choose the **Load Image** option and select the **Matlab Supported Formats** item.

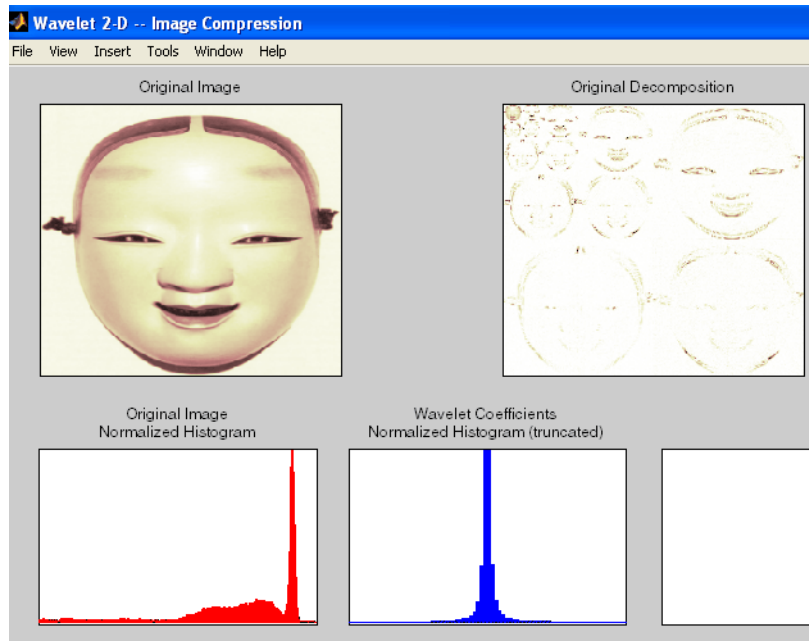
When the **Load Image** dialog box appears, select the MAT-file `mask.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`.

Click the **OK** button. A window appears asking you if you want to consider the loaded image as a truecolor one. Click the **No** button since it is a grayscale image. The mask image is loaded into the **True Compression 2-D** tool. It appears at the top left of the window together with the gray level histogram just below.



3 Perform a Wavelet Decomposition.

Accept the default wavelet **bior4.4** and select **4** from the **Level** menu which is the maximum possible level divided by 2 and then click the **Decompose** button. After a pause for computation, the tool displays the wavelet approximation and details coefficients of the decomposition for the three directions, together with the histogram of the original coefficients.



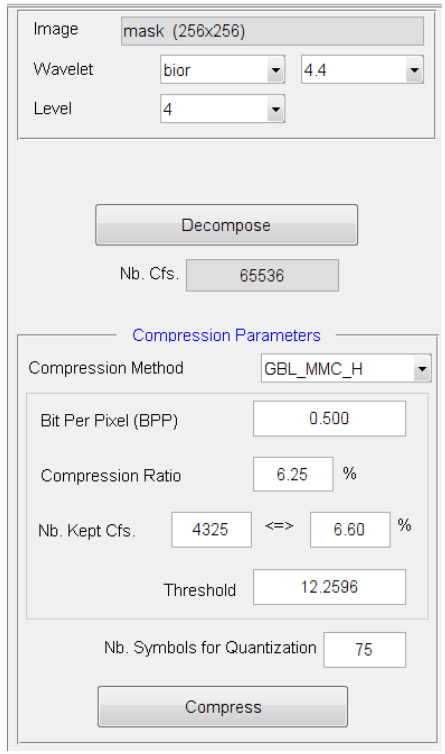
The peak of the bin containing zero illustrates the capability of wavelets to concentrate the image energy into a few nonzero coefficients.

4 Try a simple method.

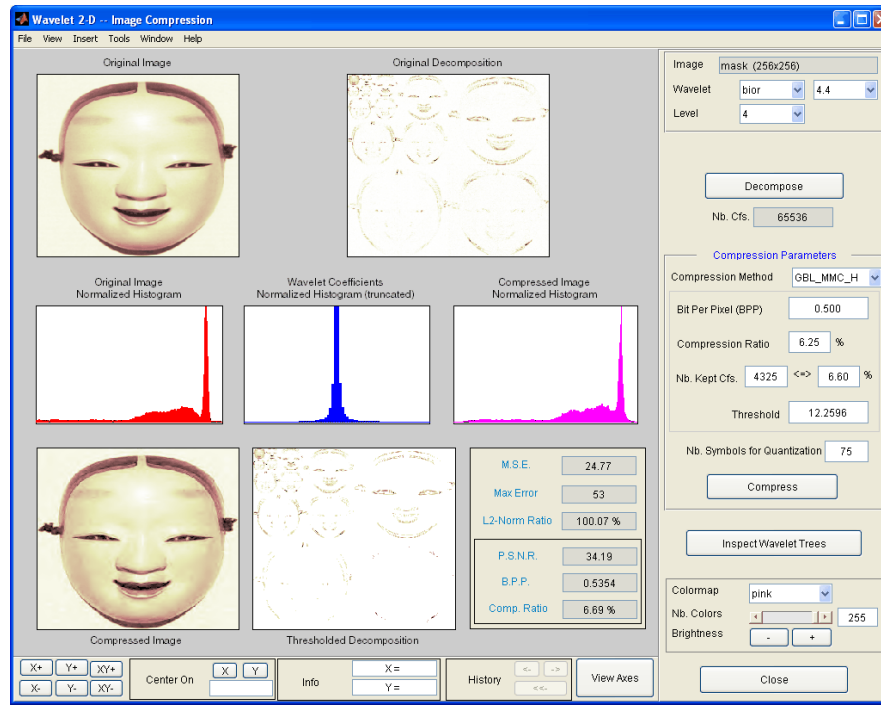
Begin with a simple method cascading global coefficients thresholding and Huffman encoding.

Choose the **GBL_MMC_H** option from the menu **Compression method** located at the top right of the **Compression Parameters** frame. For more information on the compression methods, see "True Compression for Images" in the *Wavelet Toolbox User's Guide*.

Set the desired Bit-Per-Pixel ratio to **0.5**.



Values of the other parameters are automatically updated. Note that these values are only approximations of the true performances since the quantization step cannot be exactly predicted without performing it. Click the **Compress** button.



Synthetic performance is given by the compression ratio and the computed Bit-Per-Pixel (BPP). This last one is actually about 0.53 (close to the desired one 0.5) for a compression ratio of 6.7%.

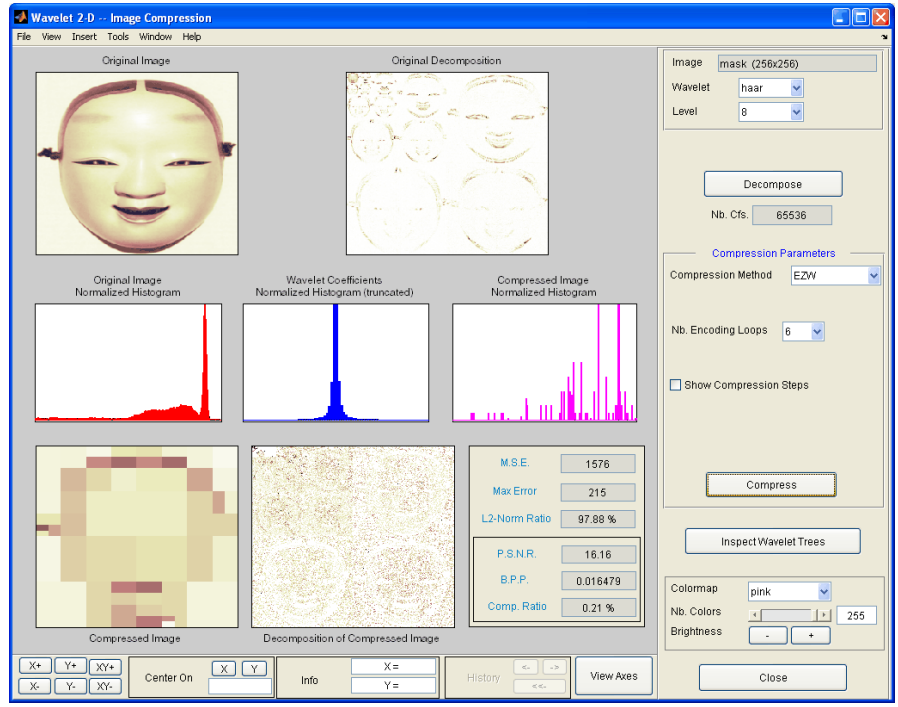
The compressed image, at the bottom left, can be compared with the original image.

The result is satisfactory but a better compromise between compression ratio and visual quality can be obtained using more sophisticated true compression which combines the thresholding and quantization steps.

5 Compress using a first progressive method: EZW.

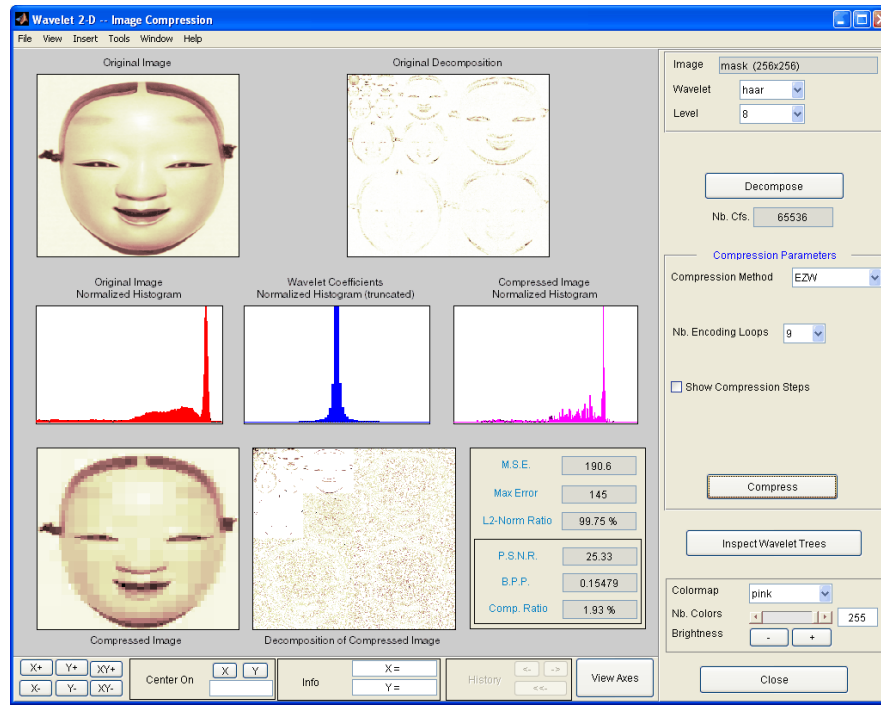
Let us now illustrate the use of progressive methods starting with the well known EZW algorithm. Let us start by selecting the wavelet **haar** from the **Wavelet** menu and select **8** from the **Level** menu. Then click the **Decompose** button.

Choose the **EZW** option from the menu **Compression method**. The key parameter is the number of loops: increasing it leads to a better recovery but worse compression ratio. From the **Nb. Encoding Loops** menu, set the number of encoding loops to **6**, which is a small value. Click the **Compress** button.

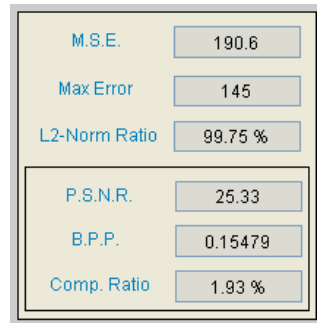


6 Refine the result by increasing the number of loops.

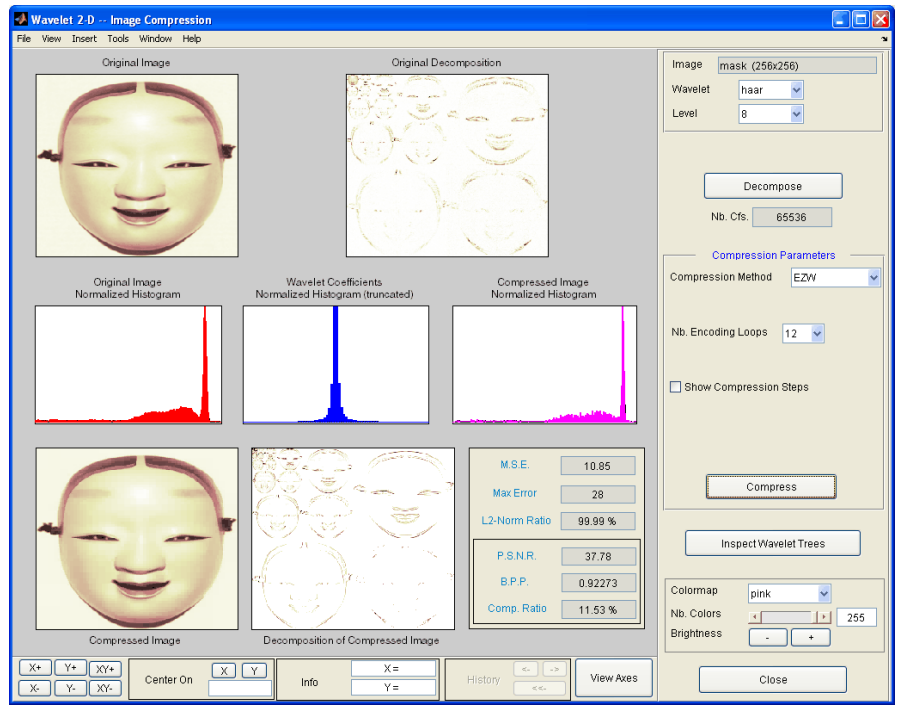
Too few steps produce a very coarse compressed image. So let us examine a little better result for **9** steps. Set the number of encoding loops to 9 and click the **Compress** button.



As can be seen, the result is better but not satisfactory, both by visual inspection and by calculating the Peak Signal to Noise Ratio (PSNR) which is less than 30.



Now try a large enough number of steps to get a satisfactory result. Set the number of encoding loops to **12** and click the **Compress** button.

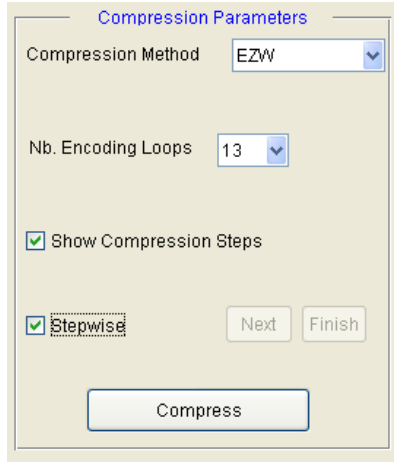


The result is now acceptable. But for 12 steps, we attain a Bit-Per-Pixel ratio about 0.92.

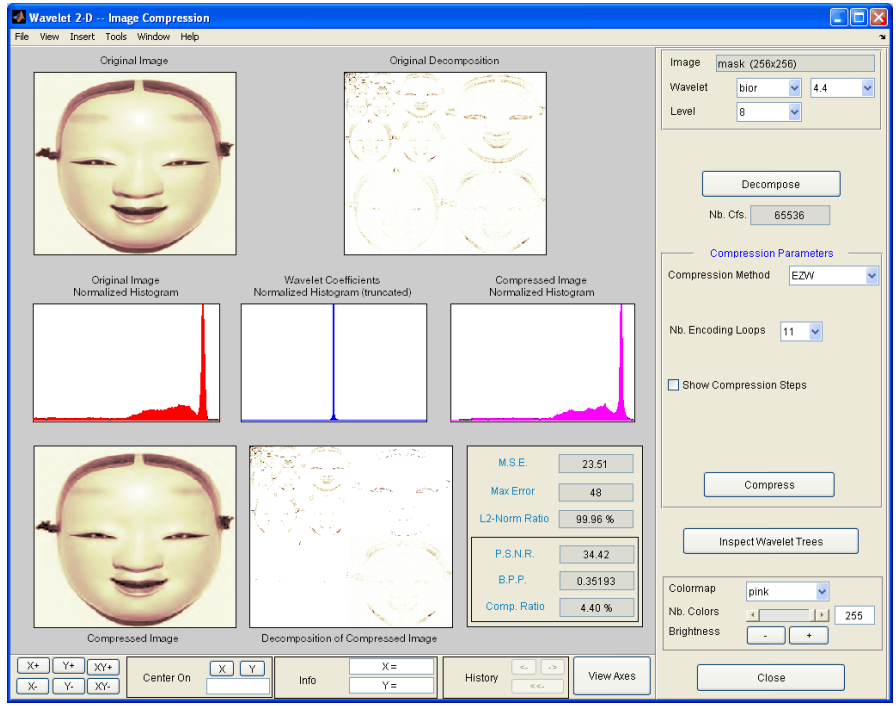
7 Get better compression performance by changing the wavelet and selecting the best adapted number of loops.

Let us try to improve the compression performance by changing the wavelet: select **bior4.4** instead of **haar** and then click the **Decompose** button.

In order to select the number of loops, the GUI tool allows you to examine the successive results obtained by this kind of stepwise procedure. Set the number of encoding loops at a large value, for example **13**, and click the **Show Compression Steps** button. Moreover you could execute the procedure stepwise by clicking the **Stepwise** button.



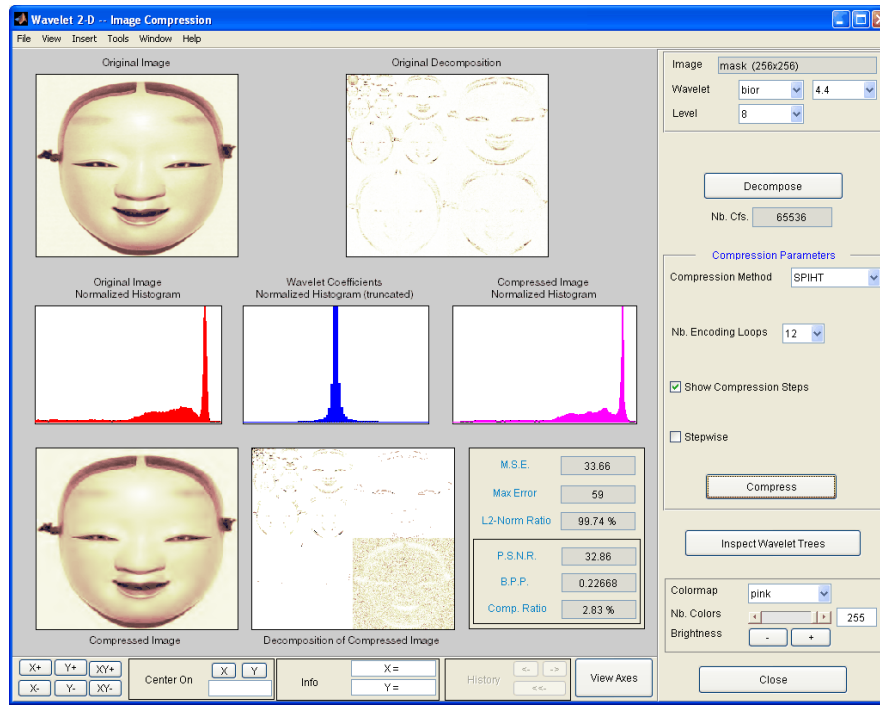
Then, click the **Compress** button. Thirteen progressively more finely compressed images appear, and you can then select visually a convenient value for the number of loops. A satisfactory result seems to be obtained after 11 loops. So, you can set the number of encoding loops to **11** and click the **Compress** button.



The reached BPP ratio is now about 0.35 which is commonly considered a very satisfactory result. Nevertheless, it can be slightly improved by using a more recent method SPIHT (Set Partitioning In Hierarchical Trees).

8 Obtain a final compressed image by using a third method.

Choose the **SPIHT** option from the menu **Compression method**, set the number of encoding loops to **12**, and click the **Compress** button.



The final compression ratio and the Bit-Per-Pixel ratio are very satisfactory: 2.8% and 0.22. Let us recall that the first ratio means that the compressed image is stored using only 2.8% of the initial storage size.

9 Handle truecolor images.

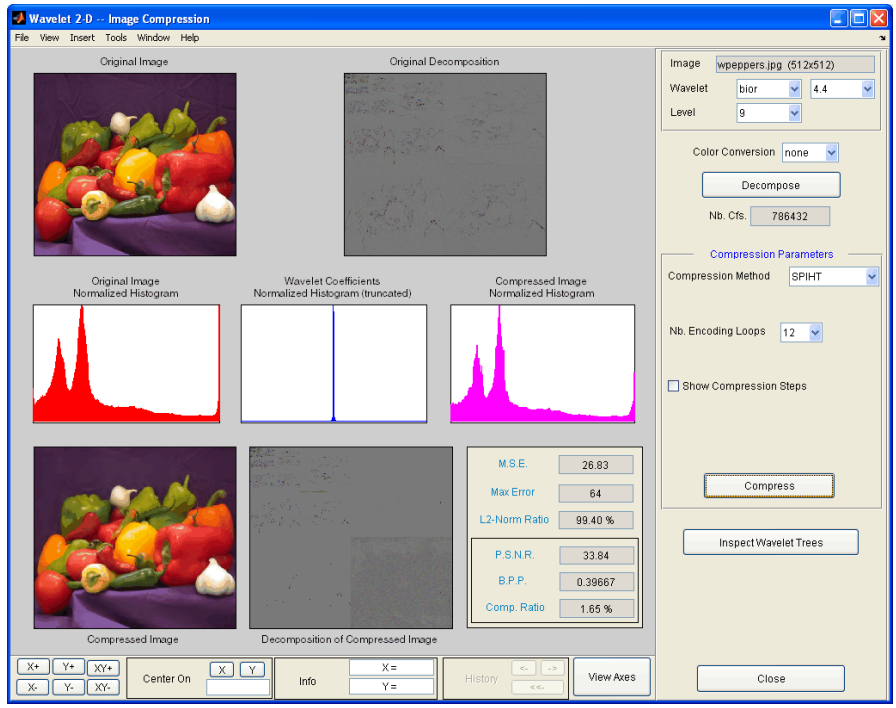
Finally, let us illustrate how to compress truecolor images. The truecolor images can be compressed along the same scheme by applying the same strategies to each of the three-color components.

From the **File** menu, choose the **Load Image** option and select the **Matlab Supported Formats** item.

When the **Load Image** dialog box appears, select the MAT-file `wpeppers.jpg` which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`.

Click the **OK** button. A window appears asking you if you want to consider the loaded image as a truecolor one. Click the **Yes** button. Accept the defaults for wavelet and decomposition level menus and then click the **Decompose** button.

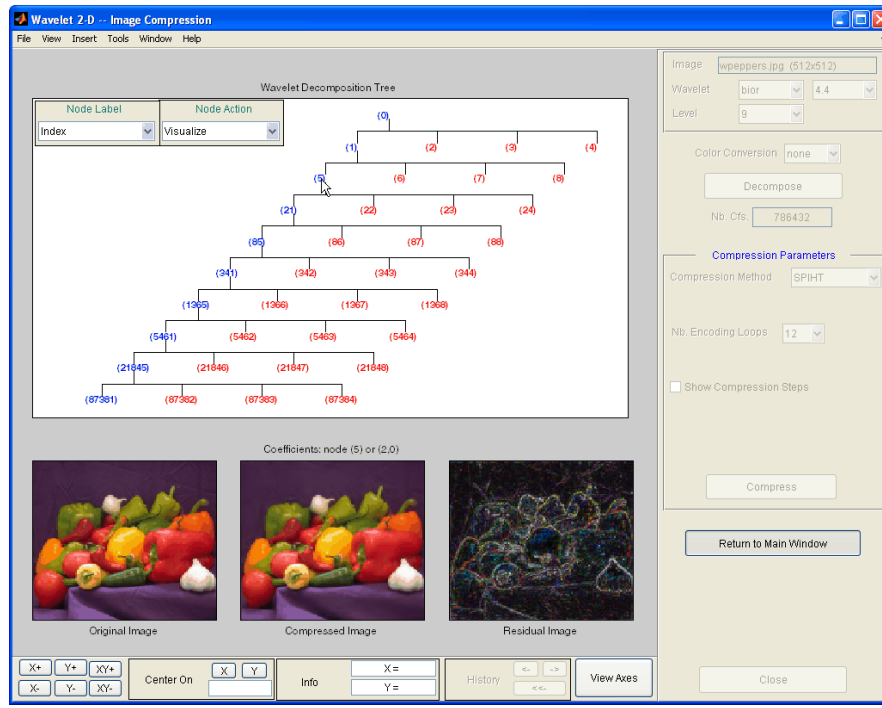
Then, accept the default compression method **SPIHT** and set the number of encoding loops to **12.** Click the **Compress** button.



The compression ratio and Bit-Per-Pixel (BPP) ratio are very satisfactory: 1.65% and 0.4 together with a very good perceptual result.

10 Inspect the wavelet tree.

For both grayscale and truecolor images, more insight on the multiresolution structure of the compressed image can be retrieved by clicking the **Inspect Wavelet Trees** button and then on the various active menus available from the displayed tree.



Importing and Exporting from the GUI

You can save the compressed image to disk in the current folder with a name of your choice.

The Wavelet Toolbox compression tools can create a file using either one of the Matlab Supported Format types or a specific format which can be recognized by the extension of the file: **wtc** (Wavelet Toolbox Compressed).

To save the above compressed image, use the menu option **File > Save Compressed Image > Wavelet Toolbox Compressed Image**. A dialog box appears that lets you specify a folder and filename for storing the image. Of course, the use of the **wtc** format requires you to uncompress the stored image using the Wavelet Toolbox true compression tools.

Three-Dimensional Discrete Wavelet Analysis

This section demonstrates the features of three-dimensional discrete wavelet analysis using the Wavelet Toolbox software. The toolbox provides these functions for 3-D data analysis. (You use the Wavelet 3-D GUI to perform all tasks except the first task.

- Getting information on the command line functions
- Loading 3-D data
- Analyzing a 3-D data
- Selecting and displaying slices
- Creating a slice movie
- Creating true 3-D display
- Importing and exporting information

Performing Three-Dimensional Analysis Using the Command Line

Both the demo (see `wavelet3ddemo`) and the documentation of the *Analysis-Decomposition* and *Synthesis-Reconstruction* functions show how you can analyze 3-D arrays efficiently using command line functions dedicated to the three-dimensional wavelet analysis. For more information, see the function reference pages.

Analysis-Decomposition Functions

Function Name	Purpose
`dwt3`	Single-level decomposition
`wavedec3`	Decomposition

Synthesis-Reconstruction Functions

Function Name	Purpose
`idwt3`	Single-level reconstruction
`waverec3`	Full reconstruction

Performing Three-Dimensional Analysis Using the Graphical Interface

In this section you explore the same 3-D-data as in the `wavelet3ddemo` demo, but you use the graphical interface tools.

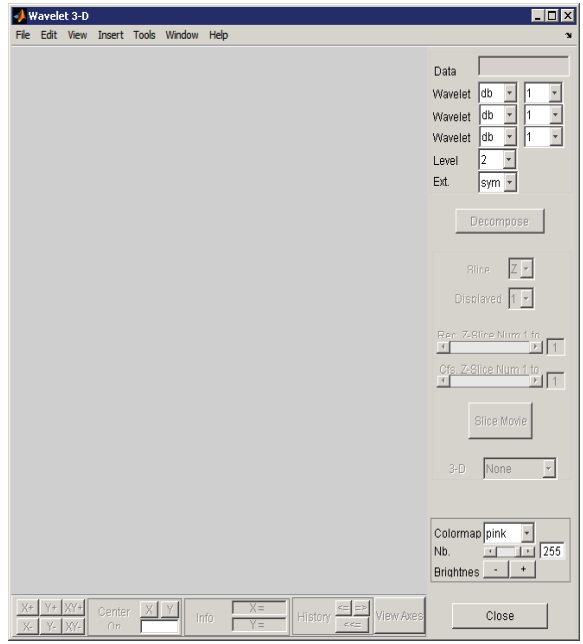
1. Start the 3-D Wavelet Analysis Tool.

   From the MATLAB prompt, type

   ```
   wavemenu
   ```

   The **Wavelet Tool Main Menu** appears.

   Click the **Wavelet 3-D** menu item. The discrete wavelet analysis tool for three-dimensional data opens.



2 Load a 3-D array.

From the **File** menu, choose **Load > Data**.

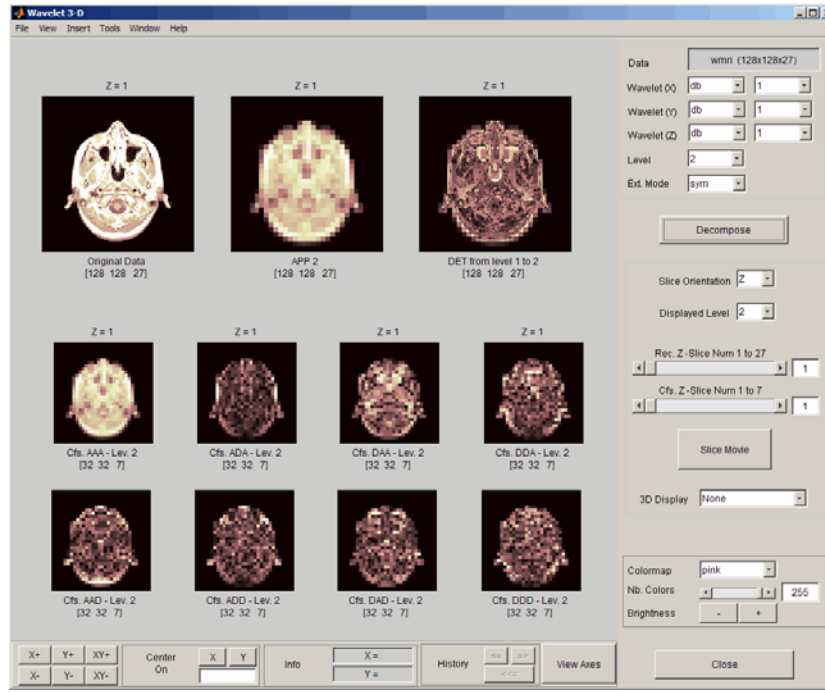
When the **Load Data** dialog box appears, select the demo MAT-file `wmri.mat`, which is in the MATLAB folder `toolbox/wavelet/wavelet`. Click **OK** to load the 3-D data into the **Wavelet 3-D** tool.

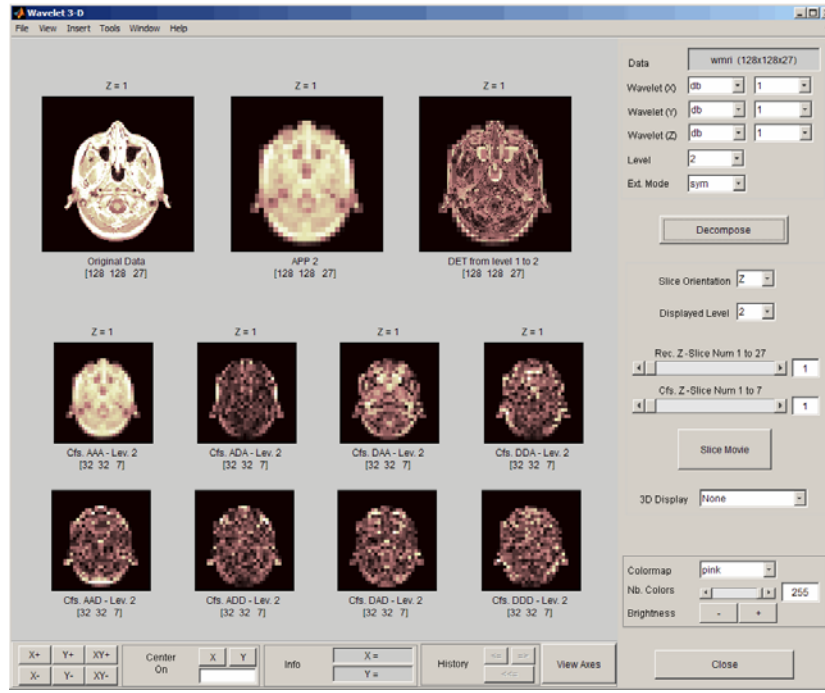
3 Analyze the 3-D array. Using the **Wavelet** and **Level** menus located in the upper part of the tool, specify:

- The wavelet familes (one per direction X, Y and Z)
- The decomposition level and the wavelet extension mode to be used for the analysis

For this analysis, accept the defaults: `db1` wavelet for each direction, decomposition at level `2` and symmetric extension mode (`sym`).

Click **Decompose**. After a pause for computation, the **Wavelet 3-D** tool displays its analysis.


Wavelet 3-D
File View Insert Tools Window Help
Z = 1
Original Data
[128 128 27]
APP 2
[128 128 27]
DET from level 1 to 2
[128 128 27]
Cfs. AAA - Lev. 2
[32 32 7]
Cfs. ADA - Lev. 2
[32 32 7]
Cfs. DAA - Lev. 2
[32 32 7]
Cfs. DDA - Lev. 2
[32 32 7]
Cfs. AAD - Lev. 2
[32 32 7]
Cfs. ADD - Lev. 2
[32 32 7]
Cfs. DAD - Lev. 2
[32 32 7]
Cfs. DDD - Lev. 2
[32 32 7]
Data
wmri (128x128x27)
Wavelet (X)
Wavelet (Y)
Wavelet (Z)
db
Level
Ext. Mode
sym
Decompose
Slice Orientation
Displayed Level
Rec. Z-Slice Num 1 to 27
Cfs. Z-Slice Num 1 to 7
Slice Movie
3D Display
None
Colormap
pink
Nb. Colors
255
Brightness
X+ Y+ XY+
X- Y- XY-
Center On
Info
History
View Axes
Close

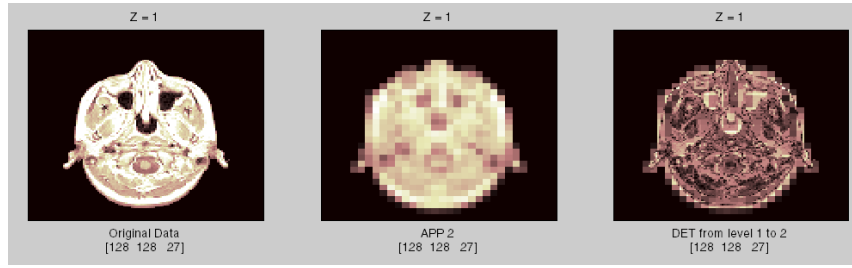


Review the slices of data and wavelet components in the graphical display. These slices are orthogonal to the *z*-direction as indicated by **Slice Orientation** in the command part of the window. This option lets you choose the desired slice orientation.

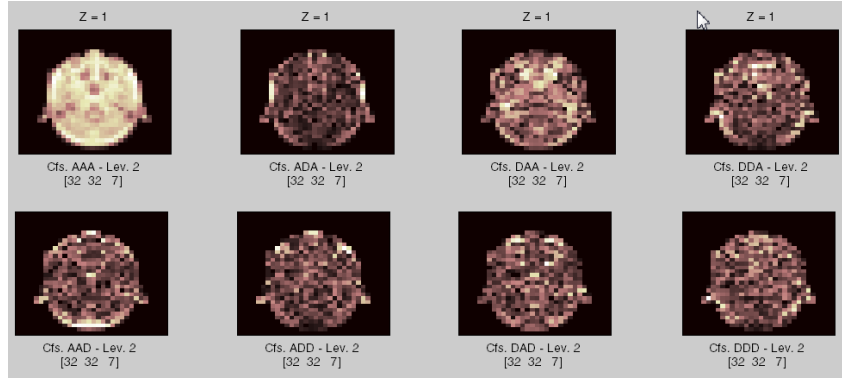
The first row of the graphical display area displays from left to right and for **Z = 1**:

- The original data slice
- The approximation at level 2 slice (low-pass component APP2)
- The slice which is the sum of all the components from level 1 to level 2, different from the low-pass one.

The *x*-labels of the three axes give you the name and the size of the displayed data.

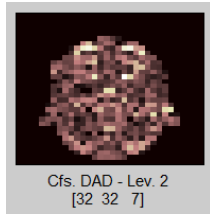


The next two lines of axes, display the wavelet coefficients at level 2, which is the desired level of the analysis. In the first line, the first graph contains the coefficients of approximation at level 2. The remaining seven axes display the seven types of wavelet coefficients at level 2. These coefficients contain the *x*-labels of the eight axes and display the name, type and size of the displayed data.

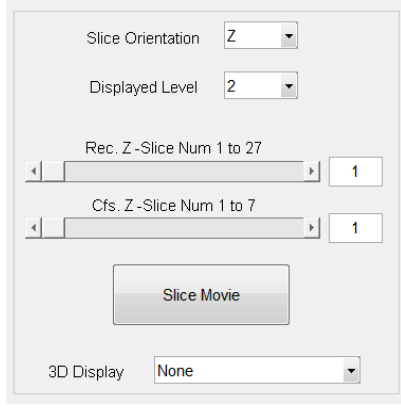


For example, in the third graphic of the bottom line, you can see the `Cfs-DAD` coefficients at level 2, which correspond to an array of size 32 x 32 x 7. The name of the `DAD` coefficients group indicates that it is obtained using

- The high-pass filter in *x* direction (`D` for detail)
- The low-pass filter in *y* direction (`A` for approximation)
- The high-pass filter in Z direction (`D` again for detail), leading to the `DAD` component

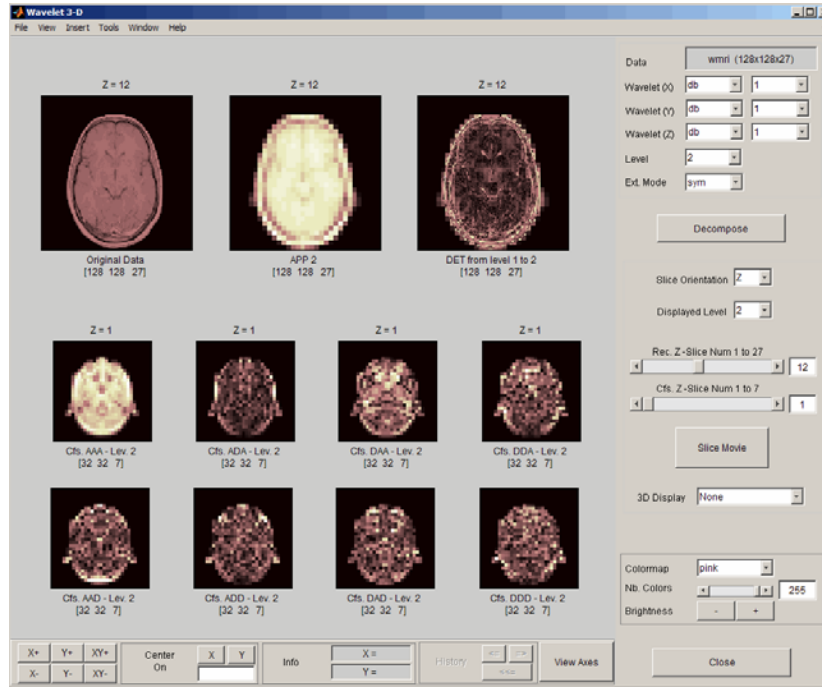


You use the **Displayed Level** in the command part of the window to choose the level of the displayed component, from 1 to the decomposition level.



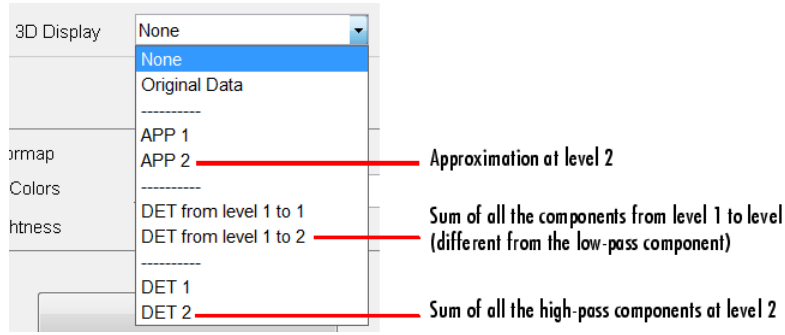
You can modify characteristics of the display using the options in the command part of the window. Each pair of sliders controls part of graphical array, the original and the reconstructed slices with the first pair or the coefficients slices with the second pair. Above each slider you can see the number of slices in the current slice orientation.

Using the slider (or by directly editing the values) of **Rec. Z-Slice**, choose slice number twelve. Similarly, choose slice number two using **Cfs. Z-Slice**.

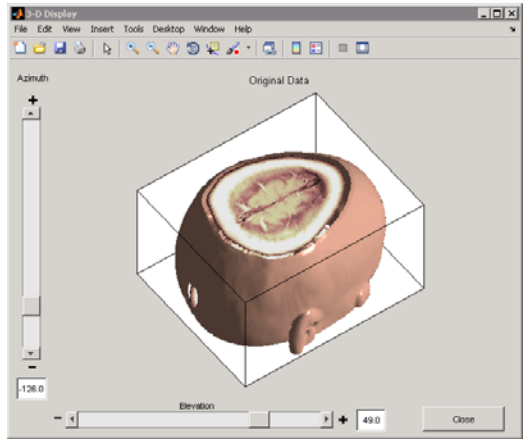


The **Slice Movie** button lets you see a movie of all the slices, first for the reconstructed slices and then for the coefficients slices. In this case, the movie contains 27 reconstructed images and 7 coefficients images.

3D Display lets you examine the original data and the wavelet components in true 3-D mode. Click **3D Display** and select APP1.



A rotated 3-D view of the approximation at level 1 opens in a new window. Use the sliders in the 3-D tool to examine the 3-D data.



Importing and Exporting Information from the Graphical Interface

You can import information from and export information either to disk or to the workspace using the **Wavelet 3-D** graphical tool.

Loading Information into the Wavelet 3-D Tool

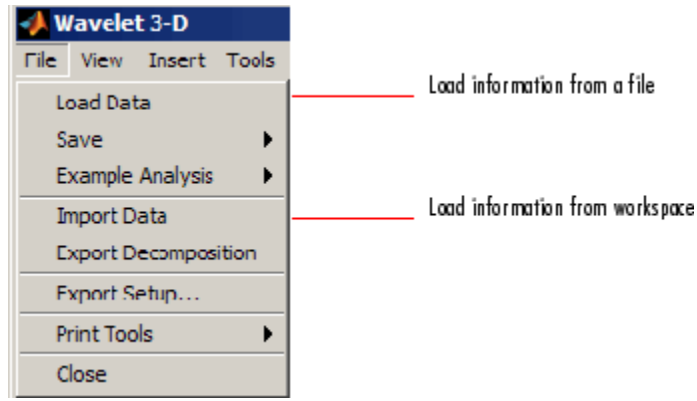
To load 3-D data you have constructed in your MATLAB workspace into the **Wavelet 3-D** tool, save the 3-D data in a MAT-file, using

```
M = magic(8);
X = repmat(M,[1 1 8]);
save magic3d X
whos
```

where `M` and `X` are

Name	Size	Bytes	Class
`M`	`8x8`	`512`	`double`
`X`	`8x8x8`	`4096`	`double`

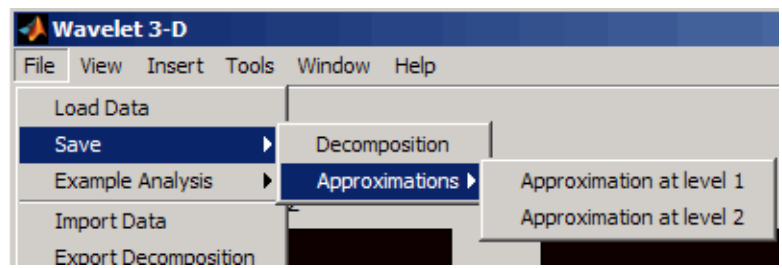
To load this 3-D data into the **Wavelet 3-D** tool, use the menu option **File > Load Data**. You then select the MAT-file to load.



Similarly, you can load information from the workspace using **File > Import Data**. You then select the variable to load.

Saving Information to a File

You can save decompositions and approximations from the **Wavelet 3-D** tool to a file or to the workspace.



Saving Decompositions. The **Wavelet 3-D** tool lets you save the entire set of data from a discrete wavelet analysis to a file. The toolbox creates a MAT-file in the current folder with a name you choose.

1. Open the **Wavelet 3-D** tool with **File > Load Data**, and select magic3d to load the 3-D data file.

2 After analyzing your data, save it by using **File > Save > Decomposition**.

3 In the dialog box that appears, specify a folder and file name for storing the decomposition data. Type the name `dec_magic3d`.

4 After saving the decomposition data to the file `dec_magic3d.mat`, load the variables into your workspace.

```
load dec_magic3d
whos
```

where wdec is

Name	Size	Bytes	Class
wdec	1x1	9182	struct

The variable wdec contains the wavelet decomposition structure.

```
wdec =
     sizeINI: [8 8 8]
       level: 2
     filters: [1x1 struct]
      mode: 'sym'
       dec: {15x1 cell}
     sizes: [3x3 double]
```

Saving Approximations. You can process a 3-D data in the **Wavelet 3-D** tool and then save any desired approximation, depending on the level chosen for the decomposition.

1 Open the **Wavelet 3-D** tool and load the file containing the 3-D data to analyze by using **File > Load Data**

2 Select magic3d.

3 Select the **File > Save > Approximations > Approximation at level 2** menu option.

4. In the dialog box that appears, select a folder and file name for the MAT-file. For this example, choose the name `App2_magic3D`.

5. Load the image data into your workspace.

```
load App2_magic3D
whos
```

where `x` is

Name	Size	Bytes	Class
`x`	`8x8x8`	`4096`	`double`

Index

E

F

G

S

T

U

W

Z