Wavelet Toolbox

For Use with MATLAB®

Michel Misiti
Yves Misiti
Georges Oppenheim
Jean-Michel Poggi

Computation

Visualization

Programming

User's Guide
Version 1

How to Contact The MathWorks:

508-647-7000 Phone

508-647-7001 Fax

The MathWorks, Inc.
24 Prime Park Way
Natick, MA 01760-1500 Mail

`http://www.mathworks.com` Web
`ftp.mathworks.com` Anonymous FTP server
`comp.soft-sys.matlab` Newsgroup

`support@mathworks.com` Technical support
`suggest@mathworks.com` Product enhancement suggestions
`bugs@mathworks.com` Bug reports
`doc@mathworks.com` Documentation error reports
`subscribe@mathworks.com` Subscribing user registration
`service@mathworks.com` Order status, license renewals, passcodes
`info@mathworks.com` Sales, pricing, and general information

Wavlet Toolbox User's Guide

Printing History: March 1996 First printing

Contents

Preface

1 Wavelets: A New Tool for Signal Analysis

3 Wavelet Applications

4 Wavelets in Action: Examples and Case Studies

5 Using Wavelet Packets

6 Advanced Concepts

7 Adding Your Own Wavelets

8 Reference

A GUI Reference

Preface

About the Authors

Michel Misiti, Georges Oppenheim, and Jean-Michel Poggi are mathematics professors at Ecole Centrale de Lyon, University of Marne-La- Vallée and Paris 10 University. Yves Misiti is a research engineer specializing in Computer Sciences at Paris 11 University.

They are members of the “Laboratoire de Mathématique” Orsay-Paris 11 University France.

Their fields of interest are statistical signal processing, stochastic processes, adaptive control, and wavelets.

The authors group, which was constituted more than ten years ago, has published numerous theoretical papers and carried out applications in close collaboration with industrial teams. For instance:

- Robustness of the piloting law for a civilian space launcher for which an expert system was developed
- Forecasting of the electricity consumption by nonlinear methods
- Forecasting of air pollution

Acknowledgments

The authors wish to express their gratitude to all the colleagues who directly or indirectly contributed to the making of the Wavelet Toolbox.

Specifically

- For the wavelet questions to Pierre-Gilles Lemarié-Rieusset (Evry) and Yves Meyer (Paris 9)
- For the statistical questions to Lucien Birgé (Paris 6) and Pascal Massart (Paris 11) whose last result is included as a starting point of the de-noising algorithm
- To David Donoho (Stanford) and to Anestis Antoniadis (Grenoble) who are used to giving generously so many valuable ideas

Colleagues and friends have helped us steadily: Samir Akkouche (Ecole Centrale de Lyon), Mark Asch (Paris 11), Patrice Assouad (Paris 11), Roger Astier (Paris 11), Jean Coursol (Paris 11), Didier Dacunha-Castelle (Paris 11), Claude Deniau (Marseille), Patrick Flandrin (Ecole Normale de Lyon), Eric Galin (Ecole Centrale de Lyon), Christine Graffigne (Paris 5), Anatoli Juditsky (Rennes), Gérard Kerkyacharian (Amiens), Gérard Malgouyres (Paris 11), Olivier Nowak (Ecole Centrale de Lyon), Dominique Picard (Paris 7), and Franck Tarpin-Bernard (Ecole Centrale de Lyon).

Several student groups have tested preliminary versions.

One of our first opportunities to apply the ideas of wavelets connected with signal analysis and its modeling occurred during a close and pleasant cooperation with the team “Analysis and Forecast of the Electrical Consumption” of Electricité de France (Clamart-Paris) directed first by Jean-Pierre Desbrosses, then by Hervé Laffaye and which included Xavier Brossat, Yves Deville, and Marie-Madeleine Martin.

Many thanks to those who tested and helped to refine the software and the printed matter and at last to The MathWorks group and specially to Roy Lurie, Jim Tung, and Bruce Sesnovich.

And finally, apologies to those we may have omitted.

What is the Wavelet Toolbox?

The Wavelet Toolbox is a collection of functions built on the MATLAB® Technical Computing Environment. It provides tools for the analysis and synthesis of signals and images using wavelets and wavelet packets within the framework of MATLAB.

The toolbox provides two categories of tools:

- Command line functions
- Graphical interactive tools

The first category of tools is made up of functions that you can call directly from the command line or from your own applications. Most of these functions are M-files, series of statements that implement specialized wavelet analysis or synthesis algorithms. You can view the code for these functions using the following statement:

```
type function_name
```

You can view the header of the function, the help part, using the statement:

```
help function_name
```

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

```
help wavelet
```

You can change the way any toolbox function works by copying and renaming the M-file, then modifying your copy. You can also extend the toolbox by adding your own M-files.

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

```
wavemenu
```

from the command line.

How to Use This Guide

If you are new to wavelet analysis and synthesis and need an overview of the concepts, read Chapter 1, "Wavelets: A New Tool for Signal Analysis." It presents the main ideas without mathematical complexity.

After this you can refer to Chapter 2 and Chapter 5, for instructions on using the wavelet and wavelet packet analysis tools, respectively; Chapter 3, which discusses practical applications of wavelet analysis; and Chapter 4, which provides examples and a case study.

If you have experience with signal analysis and wavelets, you may want to turn directly to:

- Chapter 2 and Chapter 5, for instructions on using the wavelet and wavelet packet analysis tools, respectively.
- Chapter 6, for a discussion of the technical underpinnings of wavelet analysis.
- Chapter 7, for instructions on extending the Wavelet Toolbox by adding your own wavelets.

All toolbox users should look to Chapter 8 for complete reference information about the Wavelet Toolbox command line functions, and to Appendix A for more detailed information on using the many functions provided by the graphical tools.

For More Background

The Wavelet Toolbox 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.

An excellent supplementary text that presents a complementary treatment of wavelet theory and practice is the book, *Wavelets and Filter Banks* by Gilbert Strang and Truong Nguyen. Signal processing engineers will find this book especially useful. It offers a clear and easy-to-understand introduction to two central ideas - filter banks for discrete signals and wavelets - and fully explains the connection between them. Many exercises in the book are drawn from the Wavelet Toolbox.

Wavelets and Filter Banks
Gilbert Strang and Truong Nguyen
Wellesley-Cambridge Press, 1996
ISBN 0-9614088-7-1

Available from

Wellesley-Cambridge Press
Box 812060, Wellesley
MA 02181, USA.
Phone: (617) 431-8488
Fax: (617) 253-4358
Email: `gs@math.mit.edu`

The homepage for the book is:

http://saigon.ece.wisc.edu/~waveweb/Tutorials/book.html

Installation

To install this toolbox on your computer, see the appropriate platform-specific *MATLAB Installation Guide*. To determine if the Wavelet Toolbox is already installed on your system, check for a subdirectory named `wavelet` within the main toolbox directory or folder.

The Wavelet Toolbox can perform signal or image analysis. Since MATLAB stores most numbers in double precision, even a single image takes up a lot of memory. For instance, one copy of a 512-by-512 image uses 2 MB of memory. To avoid `Out of Memory` errors, it is important 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 set up virtual memory.

System Recommendations

While not a requirement, we recommend you obtain the Signal Processing and Image Processing Toolboxes to use in conjunction with the Wavelet Toolbox. These toolboxes provide complementary functionality that will 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 may depend on your hardware or operating system.

Windows Fonts

We recommend you set the system to use “Small Fonts.” Some of the labels in the GUI windows may be illegible if large fonts are used.

Set this option by clicking the Display icon in your desktop's Control Panel (accessible through the **Settings⇒Control Panel** submenu in Windows 95). Use the **Font Size** menu to change to **Small Fonts**. You'll have to restart Windows for this change to take effect.

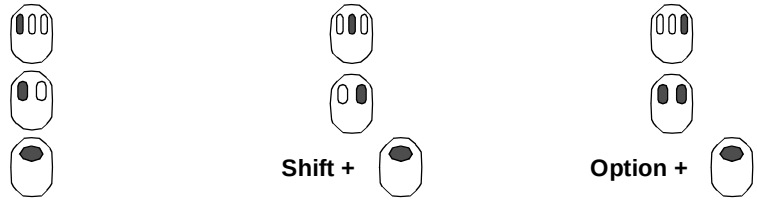
Other Platforms Fonts

We recommend you set the system to use standard default fonts. Some of the labels in the GUI windows may be illegible if other fonts are used.

Mouse Compatibility

The Wavelet Toolbox was designed for three distinct types of mouse control:

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



For more information, see the section “Using the Mouse” on page A-4.

Typographical Conventions

This manual uses certain typographical conventions.

Font	Use for
`Monospace`	Commands, function names, and screen displays; for example, `conv`.
`Monospace Italics`	Names of arguments that are meant to be replaced and not typed literally; for instance: `cd` *`directory`*.
Italics	Book titles, mathematical notation, and the introduction of new terms.
Boldface Initial Cap	Names of keys, such as the **Return** key and menu items, such as the **File** menu.

1

Wavelets: A New Tool for Signal Analysis

Everywhere around us are signals that need to be analyzed. Seismic tremors, human speech, engine vibrations, medical images, financial data, music, and many other types of signals have to be efficiently encoded, compressed, cleaned up, reconstructed, described, simplified, modeled, distinguished, or located.

Wavelet analysis is a new and promising set of tools and techniques for doing this.

Fourier Analysis

Signal analysts already have at their disposal an impressive arsenal of tools. Perhaps the most well-known of these is *Fourier analysis*, which breaks down a signal into constituent sinusoids of different frequencies. Another way to think of Fourier analysis is as a mathematical technique for *transforming* our view of the signal from a time-based one to a frequency-based one.

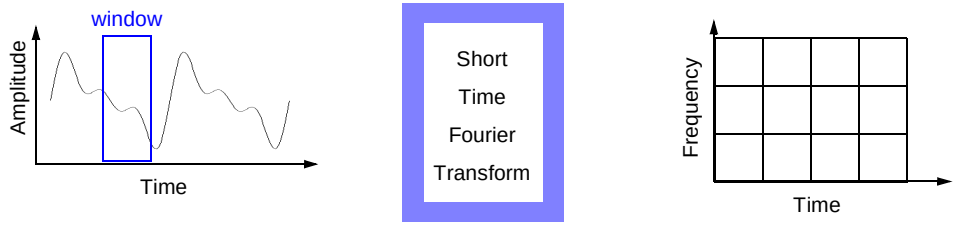
For many signals, Fourier analysis is extremely useful because the signal's frequency content is of great importance. So why do we need other techniques, like wavelet analysis?

Fourier analysis has a serious drawback. In transforming to the frequency domain, time information is lost. When looking at a Fourier transform of a signal, it is impossible to tell *when* a particular event took place.

If a signal doesn't change much over time — that is, if it is what is called a *stationary* signal — this drawback isn't very important. However, most interesting signals contain numerous non-stationary or transitory characteristics: drift, trends, abrupt changes, and beginnings and ends of events. These characteristics are often the most important part of the signal, and Fourier analysis is not suited to detecting them.

Short-Time Fourier Analysis

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, called the *Short-Time Fourier Transform* (STFT), maps a signal into a two-dimensional function of time and frequency.



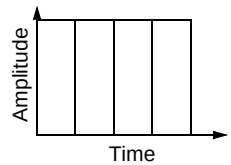
The STFT represents a sort of compromise between the time- and frequency-based views of a signal. It provides some information about both when and at what frequencies a signal event occurs. However, you can only obtain this information with limited precision, and that precision is determined by the size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

While the STFT's 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 we can vary the window size to determine more accurately either time or frequency.

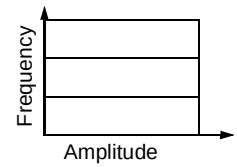
Wavelet Analysis

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

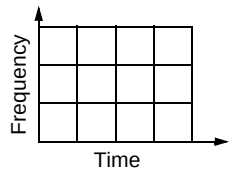
Here's what this looks like in contrast with the time-based, frequency-based, and STFT views of a signal:



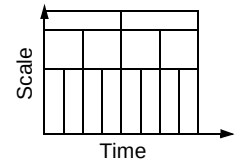
Time Domain (Shannon)



Frequency Domain (Fourier)



STFT (Gabor)



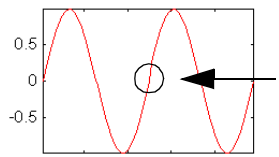
Wavelet Analysis

You may have noticed that wavelet analysis does not use a time-frequency region, but rather a time-*scale* region. We'll have more to say about the concept of scale later.

What Can Wavelet Analysis Do?

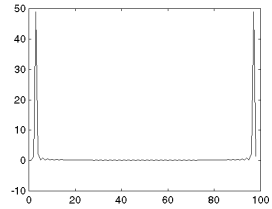
One major advantage afforded by wavelets is the ability to perform *local analysis* — that is, to analyze a localized area of a larger signal.

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.

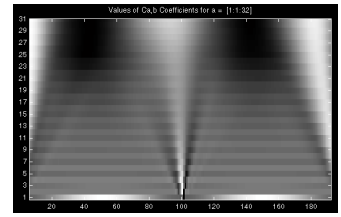


Sinusoid with a small discontinuity

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.



Fourier Coefficients



Wavelet Coefficients

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. Further, 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.

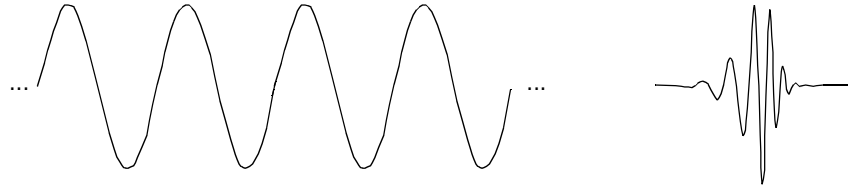
Indeed, in their brief history within the signal processing field, wavelets have already proven themselves to be an indispensable addition to the analyst's collection of tools and continue to enjoy a burgeoning popularity today.

What is Wavelet Analysis?

Now that we know some situations when wavelet analysis is useful, it is worthwhile asking the questions “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.



Sine Wave Wavelet (db10)

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

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

Number of Dimensions

Thus far, we've discussed only one-dimensional data, which encompasses most ordinary signals. However, wavelet analysis can be applied to two-dimensional data — *images*; and, in principle, to higher-dimensional data.

This toolbox uses only one- and two-dimensional analysis techniques.

The Continuous Wavelet Transform

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 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.)

The results of the transform are the *Fourier coefficients* $F(\omega)$, which when multiplied by a sinusoid of appropriate frequency ω, yield the constituent sinusoidal components of the original signal. Graphically, the process looks like:

Similarly, the *continuous wavelet transform* (CWT) is defined as the sum over all time of the signal multiplied by scaled, shifted versions of the wavelet function ψ:

$$C(scale, position) = \int_{-\infty}^{\infty} f(t)\psi(scale, position, t)dt$$

The result of the CWT are many *wavelet coefficients* C, which are a function of scale and position.

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

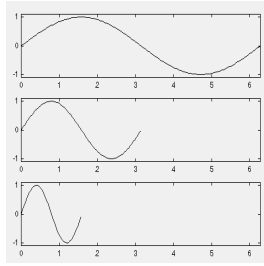
Signal *Constituent wavelets of different scales and positions*

Scaling

We've already alluded to the fact that wavelet analysis produces a time-scale view of a signal, and now we're talking about scaling and shifting wavelets. What exactly do we mean by *scale* in this context?

Scaling a wavelet simply means stretching (or compressing) it.

To go beyond colloquial descriptions such as "stretching," we introduce the *scale factor*, often denoted by the letter a. If we're talking about sinusoids, for example, the effect of the scale factor is very easy to see:

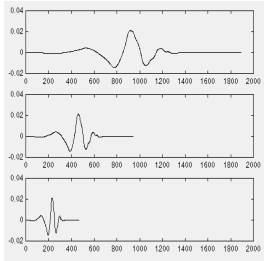


$$f(t) = \sin(t) \quad ; \quad a = 1$$

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

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

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



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

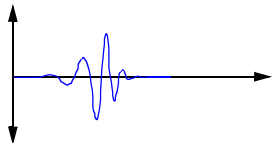
$$f(t) = \psi(2t) \quad ; \quad a = \frac{1}{2}$$

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

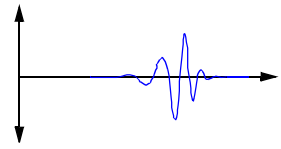
It is clear from the diagrams that, for a sinusoid $\sin(\omega t)$, the scale factor a is related (inversely) to the radian frequency ω. Similarly, with wavelet analysis, the scale is related to the frequency of the signal. We'll return to this topic later.

Shifting

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



Wavelet function
$\psi(t)$



Shifted wavelet function
$\psi(t-k)$

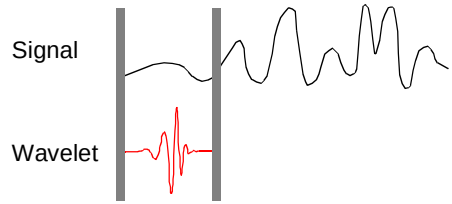
Five Easy Steps to a Continuous Wavelet Transform

The continuous wavelet transform is the sum over all time of the signal multiplied by scaled, shifted versions of the wavelet. This process produces wavelet coefficients that are a function of scale and position.

It's really a very simple process. In fact, here are the five steps of an easy recipe for creating a CWT:

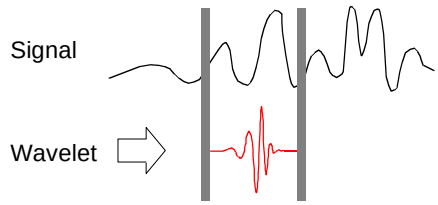
1 Take a wavelet and compare it to a section at the start of the original signal.

2 Calculate a number, C, that represents how closely correlated the wavelet is with this section of the signal. The higher C is, the more the similarity. Note that the results will depend on the shape of the wavelet you choose.

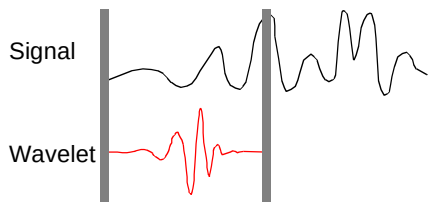


C = 0.0102

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



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

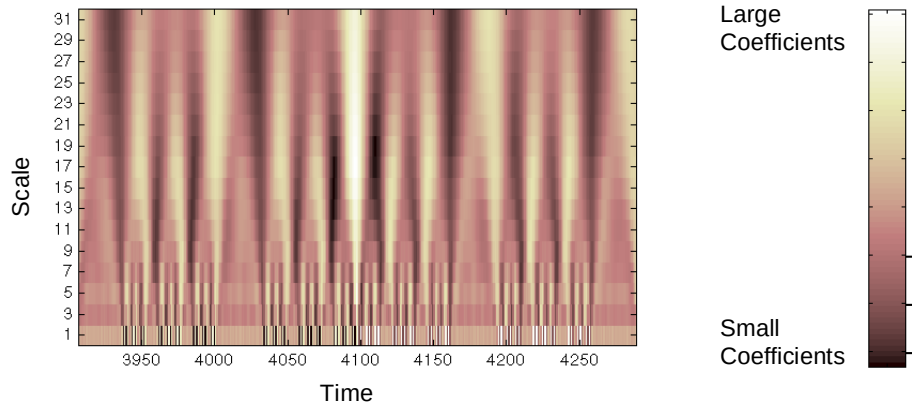


C = 0.2247

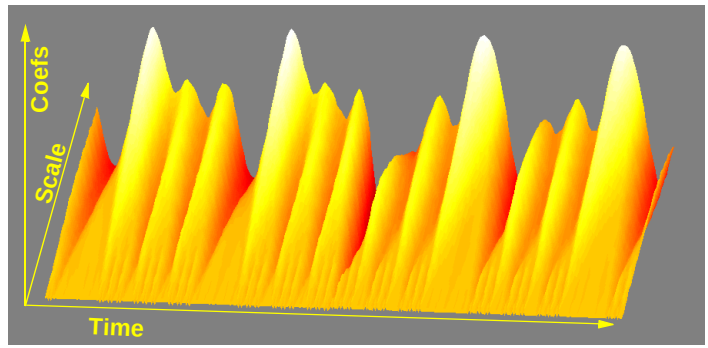
5 Repeat steps 1 through 4 for all scales.

When you're done, you'll have the coefficients produced at different scales by different sections of the signal. The coefficients constitute the results of a regression of the original signal performed on the wavelets.

How to make sense of all these coefficients? You could make a plot on which the *x*-axis represents position along the signal (time), the *y*-axis represents scale, and the color at each *x*-*y* point represents the magnitude of the wavelet coefficient `C`. These are the coefficient plots generated by the graphical tools.



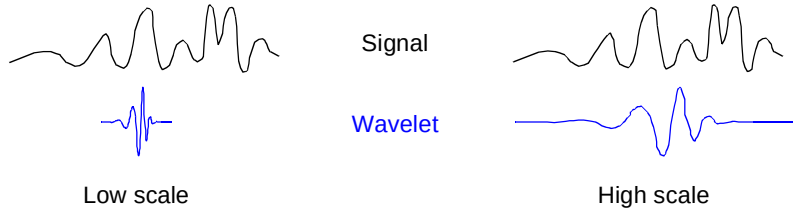
These coefficient plots resemble a bumpy surface viewed from above. If you could look at the same surface from the side, you might see something like this:



The continuous wavelet transform coefficient plots are precisely the time-scale view of the signal we referred to earlier. It is a different view of signal data than the time-frequency Fourier view, but it is not unrelated.

Scale and Frequency

Notice that the scales in the coefficients plot (shown as *y*-axis labels) run from 1 to 31. Recall that the 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 thus the coarser the signal features being measured by the wavelet coefficients.



Thus, there is a correspondence between wavelet scales and frequency as revealed by wavelet analysis:

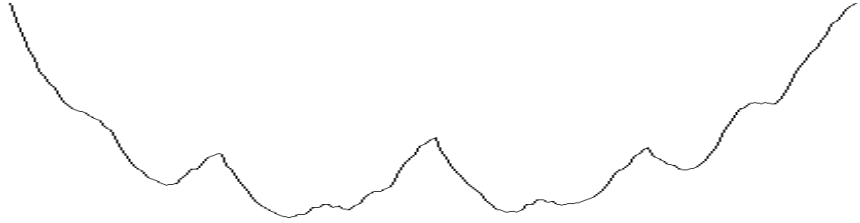
- Low scale $a \Rightarrow$ Compressed wavelet $\Rightarrow$ Rapidly changing details $\Rightarrow$ High frequency ω.
- High scale $a \Rightarrow$ Stretched wavelet $\Rightarrow$ Slowly changing, coarse features $\Rightarrow$ Low frequency ω.

The Scale of Nature

It's important to understand that the fact that wavelet analysis does not produce a time-frequency view of a signal is not a weakness but a strength of the technique.

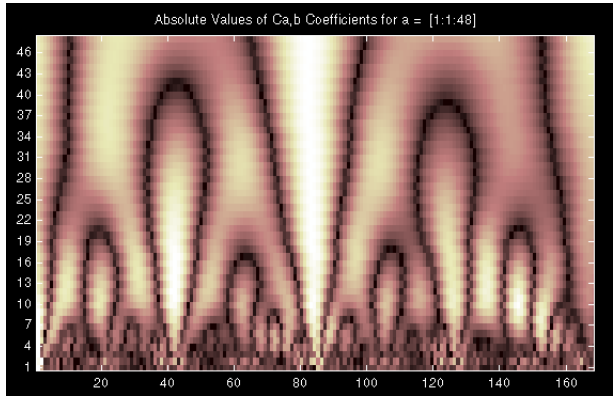
Not only is time-scale a different way to view data, it is a very natural way to view data deriving from a great number of natural phenomena.

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 abraded by small meteorites.

Here is a case where thinking in terms of scale makes much more sense than thinking in terms of frequency. Inspection of the CWT coefficients plot for this signal reveals patterns among scales and shows the signal's possibly fractal nature.



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.

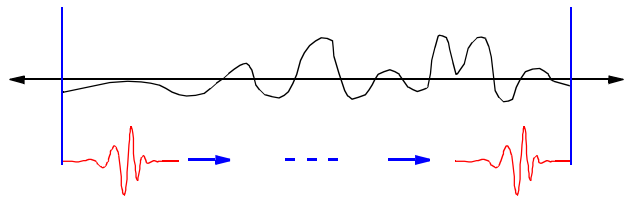
What's Continuous About the Continuous Wavelet Transform?

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 intervals. It is important to remember that the continuous wavelet transform is also operating in 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), are the scales 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 which 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.



The 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 just such an analysis from the *discrete wavelet transform* (DWT).

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

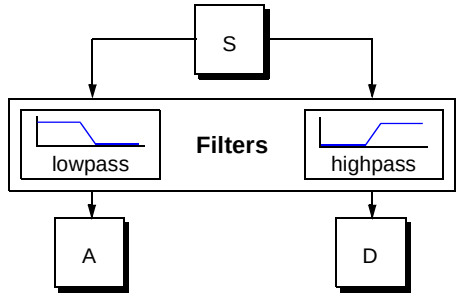
This very practical filtering algorithm yields a *fast wavelet transform* — a box into which a signal passes, and out of which wavelet coefficients quickly emerge. Let's examine this in more depth.

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.

It is for this reason that, in wavelet analysis, we often speak of *approximations* and *details*.

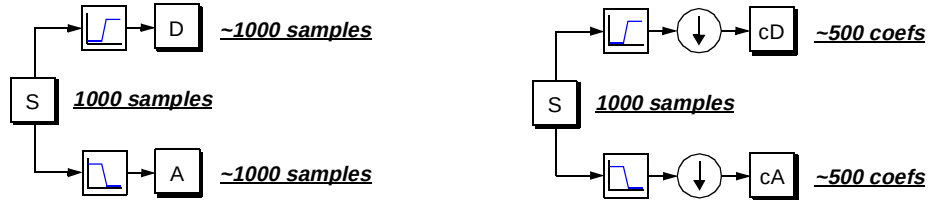
The approximations are the high-scale, low-frequency components of the signal. The details are the low-scale, high-frequency components. The filtering process, at its most basic level, looks like this:



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 approximation and the detail will each have 1000 samples, for a total of 2000.

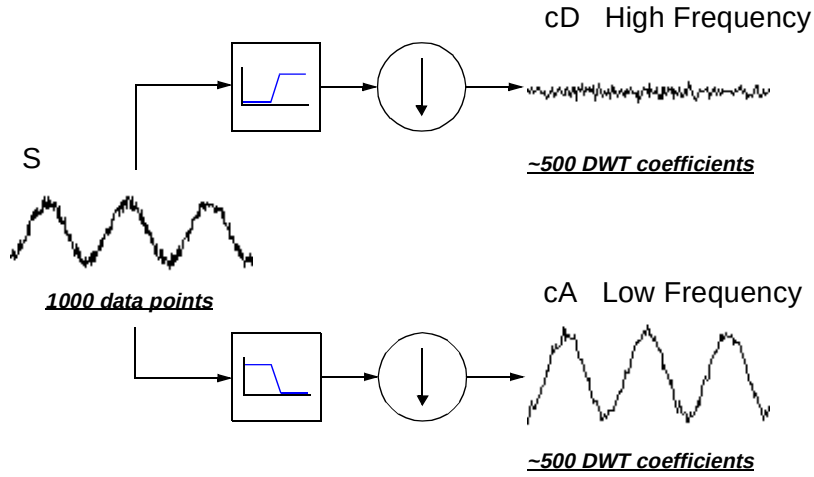
To correct this problem, we introduce the notion of *downsampling*. This simply means throwing away every second data point. While doing this introduces *aliasing* (a type of error, see p. 91 of the book *Wavelets and Filter Banks*, by Strang and Nguyen) in the signal components, it turns out we can account for this later on in the process.



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.

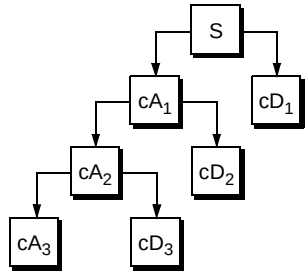
Notice that the detail coefficients `cD` consist mainly of the high-frequency noise, while the approximation coefficients `cA` contain much less noise than does the original signal.

```
» [length(cA) length(cD)]
ans =
   501  501
```

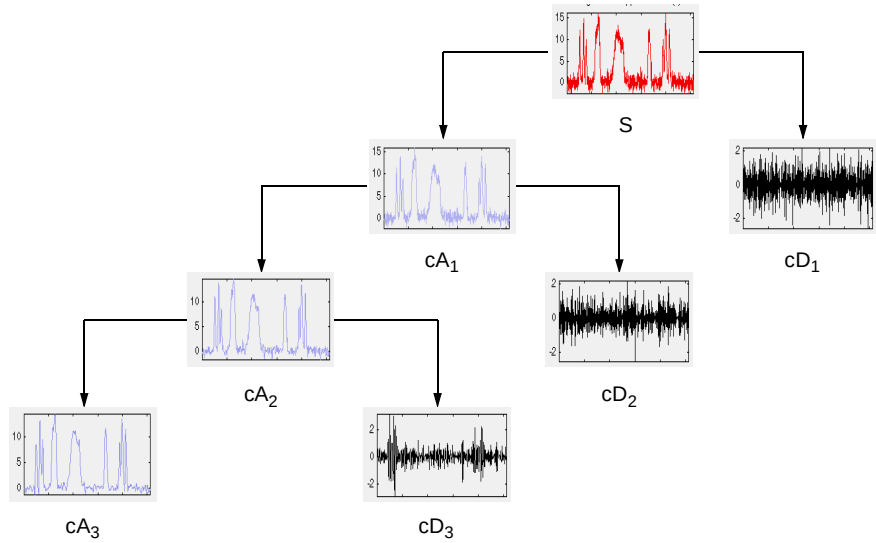
You may observe that the actual lengths of the detail and approximation coefficient vectors are slightly *more* than half the length of the original signal. This has to do with the filtering process, which is implemented by convolving the signal with a filter. The convolution “smears” the signal, introducing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



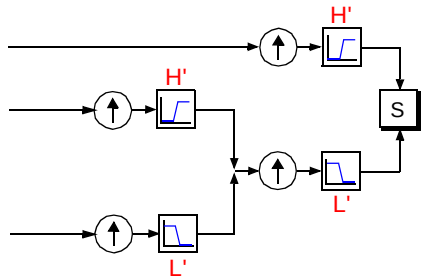
Number of Levels

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 Chapter 6 for details).

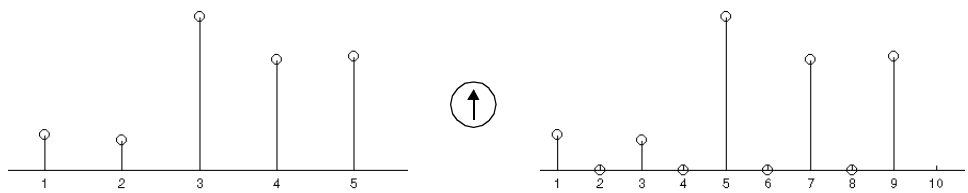
Wavelet Reconstruction

We've learned how the discrete wavelet transform can be used to analyze, or decompose, signals and images. The other half of the story is how those components can be assembled back into the original signal with no 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 in our toolbox, 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:



Signal component

Upsampled signal component

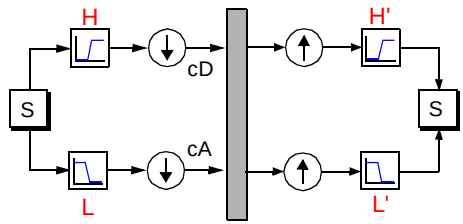
The wavelet toolbox includes commands, like `idwt` and `waverec`, that perform one-level or multi-level reconstruction, respectively, on the components of one-dimensional signals. These commands have their two-dimensional analogues, `idwt2` and `waverec2`.

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.

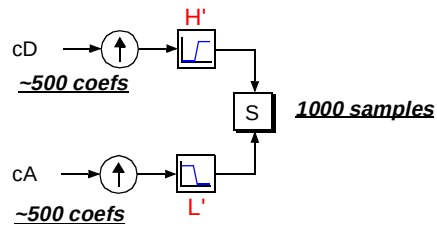
That perfect reconstruction is even possible is noteworthy. Recall that the downsampling of the signal components performed during the decomposition phase introduces a distortion called aliasing. It turns out that by carefully choosing filters for the decomposition and reconstruction phases that are closely related (but not identical), we can "cancel out" the effects of aliasing. This was the breakthrough made possible by the work of Ingrid Daubechies.

A technical discussion of how to design these filters can be found in p. 347 of the book *Wavelets and Filter Banks*, by Strang and Nguyen. The low- and highpass decomposition filters (`L` and `H`), together with their associated reconstruction filters (`L'` and `H'`), form a system of what are called *quadrature mirror filters*:



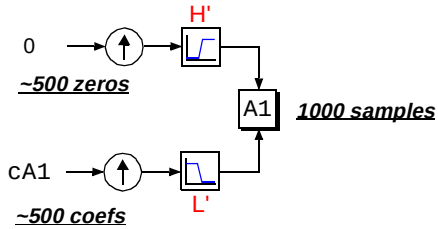
Reconstructing Approximations and Details

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



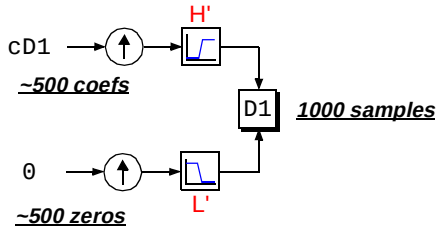
It is also possible to reconstruct the approximations and details themselves from their coefficient vectors. As an example, let's consider how we would reconstruct the first-level approximation `A1` from the coefficient vector `cA1`.

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 details:



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:



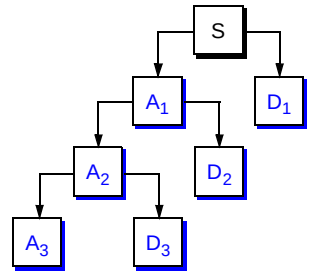
The reconstructed details and approximations are true constituents of the original signal. In fact, we find when we combine them that:

$$A_1 + D_1 = S$$

Note that the coefficient vectors `cA1` and `cD1` — because they were produced by downsampling, contain aliasing distortion, 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 multi-level analysis, we find that similar relationships hold for all the reconstructed signal constituents. That is, there are several ways to reassemble the original signal:

Reconstructed Signal Components



$$S = A_1 + D_1$$

$$= A_2 + D_2 + D_1$$

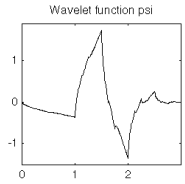
$$= A_3 + D_3 + D_2 + D_1$$

Relationship of Filters to Wavelet Shapes

In the section “Reconstruction Filters” on page 1-21, 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.

In fact, 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 lowpass reconstruction filter (`L'`) for the `db2` wavelet.



db2 wavelet

The filter coefficients can be obtained from the `dbaux` command:

```
» Lprime = dbaux(2)
Lprime =
    0.3415    0.5915    0.1585   -0.0915
```

If we reverse the order of this vector (see `wrev`), and then multiply every second sample by –1, we obtain the highpass filter `H'`:

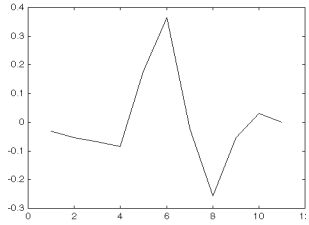
```
Hprime =
   -0.0915    -0.1585     0.5915    -0.3415
```

Next, upsample `Hprime` by two (see `dyadup`), inserting zeros in alternate positions:

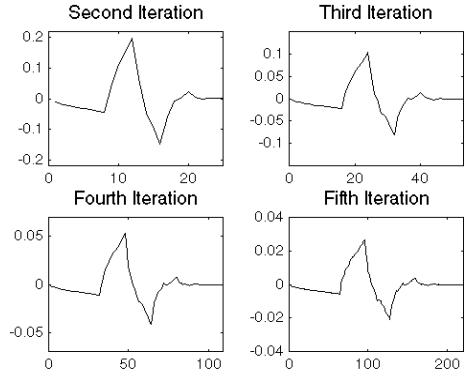
```
HU =
    -0.0915         0   -0.1585         0    0.5915         0   -0.3415         0
```

Finally, convolve the upsampled vector with the original lowpass filter:

```
H2 = conv(HU,Lprime);
plot(H2)
```



If we iterate this process several more times, repeatedly upsampling and convolving the resultant vector with the four-element filter vector `Lprime`, a pattern begins to emerge:



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.

The Scaling Function

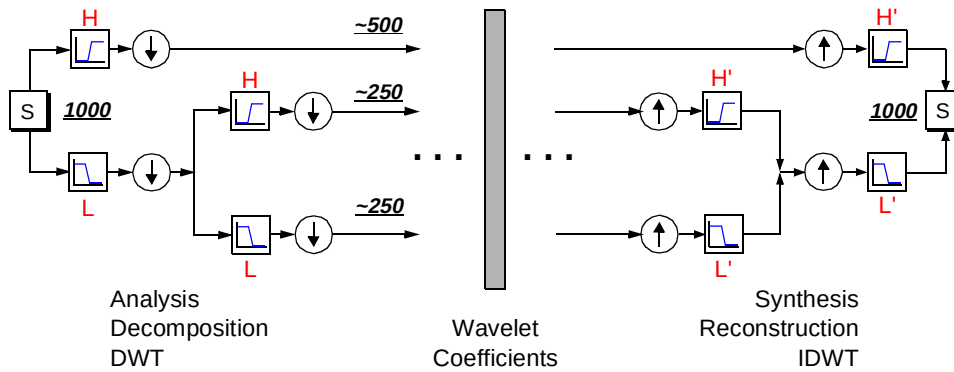
We've seen the interrelation of wavelets and quadrature mirror filters. The wavelet function ψ is determined by the highpass filter, which also produces the details of the wavelet decomposition.

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 lowpass quadrature mirror filters, and thus is associated with the approximations of the wavelet decomposition.

In the same way that iteratively upsampling and convolving the highpass filter produces a shape approximating the wavelet function, iteratively upsampling and convolving the lowpass filter produces a shape approximating the scaling function.

Multistep Decomposition and Reconstruction

A multistep analysis-synthesis process can be represented as:



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

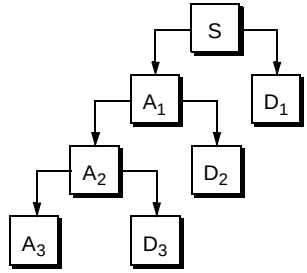
We've already discussed decomposition and reconstruction at some length. Of course, there is no point breaking up a signal merely to have the satisfaction of immediately reconstructing it. We perform wavelet analysis because the coefficients thus obtained have many known uses, de-noising and compression being foremost among them.

But wavelet analysis is still a new and emerging field. Many uncharted uses of the wavelet coefficients no doubt lie in wait. The Wavelet 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

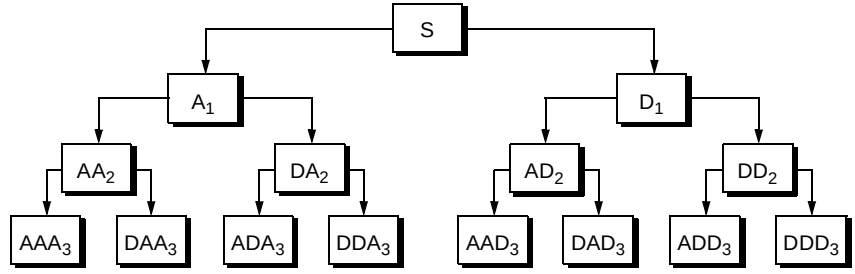
The *wavelet packet* method is a generalization of wavelet decomposition that offers a richer range of possibilities for signal analysis.

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.



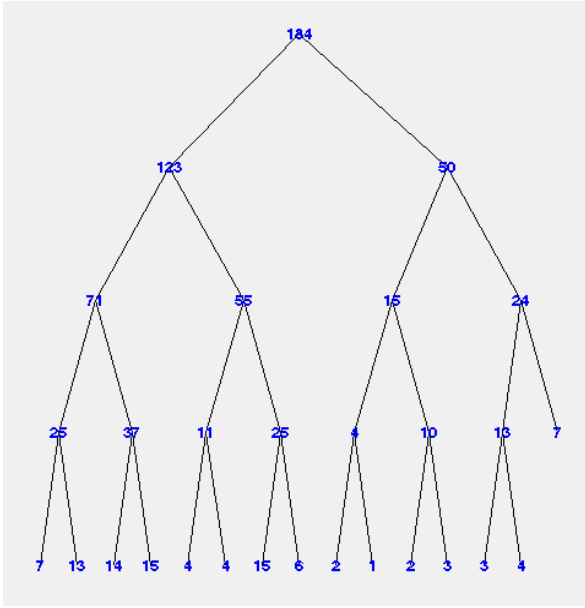
$$S = A_1 + D_1$$
$$= A_2 + D_2 + D_1$$
$$= A_3 + D_3 + D_2 + D_1$$

In wavelet packet analysis, the details as well as the approximations can be split. This yields 2^n different ways to encode the signal. This is the *wavelet packet decomposition 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, 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 the term “wavelet” was in 1909, in a thesis by Alfred Haar.

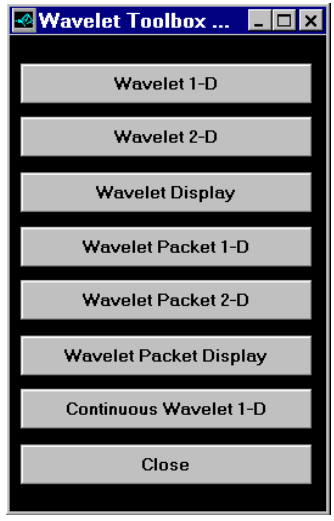
The concept of wavelets in its present theoretical form was first proposed by Jean Morlet and the team at the Marseille Theoretical Physics Center working under Alex Grossmann in France.

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.

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

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



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

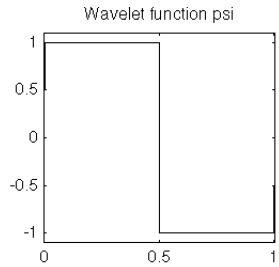
3. Select a family from the **Wavelet** menu at the top right of the tool.

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

5. Obtain more information by clicking on the information buttons located at the right.

Haar

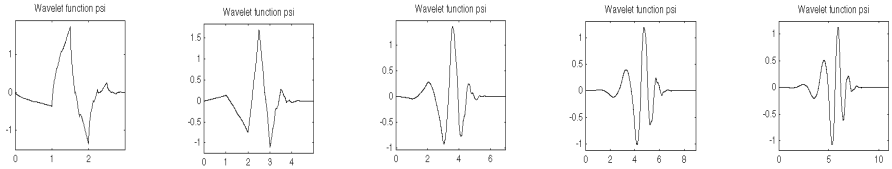
Any discussion of wavelets begins with Haar, the first and simplest. Haar is discontinuous, and resembles a step function. It represents the same wavelet as Daubechies `db1`. See "Haar" on page 64 for more detail.



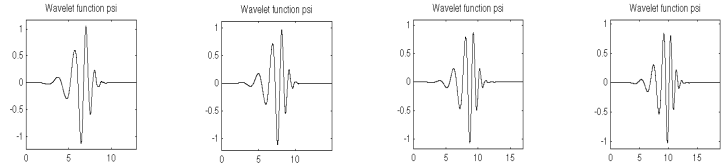
Daubechies

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`. Here are the next nine members of the family:



db2 db3 db4 db5 db6

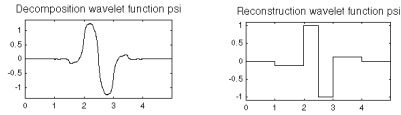


db7 db8 db9 db10

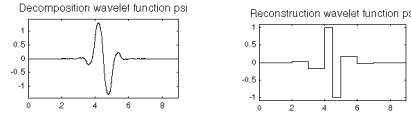
You can obtain a survey of the main properties of this family by typing `waveinfo('db')` from the MATLAB command line. See "Daubechies Wavelets: dbN" on page 6-63 for more detail.

Biorthogonal

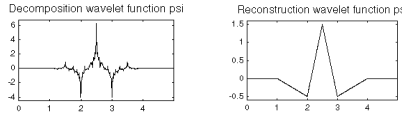
This family of wavelets exhibits the property of linear phase, which is needed for signal and image reconstruction. By using two wavelets, one for decomposition and the other for reconstruction instead of the same single one, interesting properties are derived.



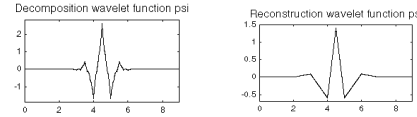
bior1.3



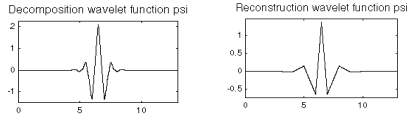
bior1.5



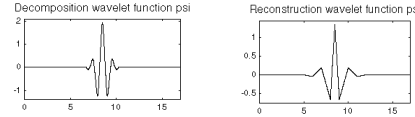
bior2.2



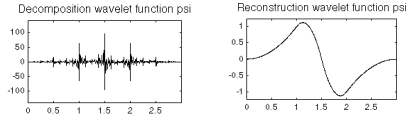
bior2.4



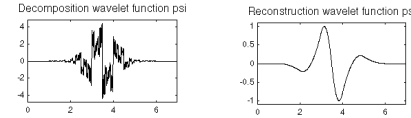
bior2.6



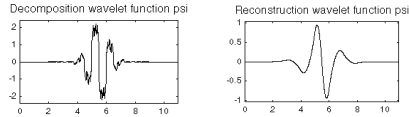
bior2.8



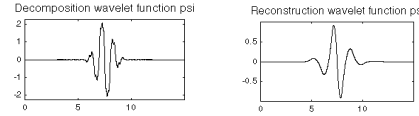
bior3.1



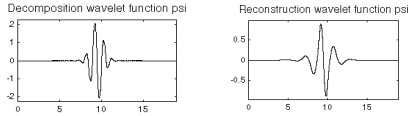
bior3.3



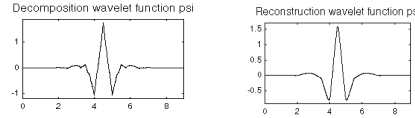
bior3.5



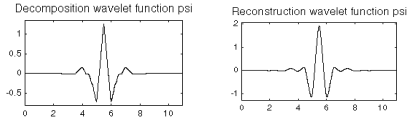
bior3.7



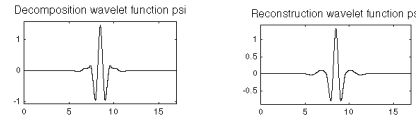
bior3.9



bior4.4



bior5.5

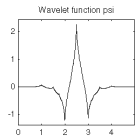


bior6.8

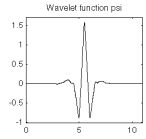
You can obtain a survey of the main properties of this family by typing `waveinfo('bior')` from the MATLAB command line. See “Biorthogonal Wavelet Pairs: biorNr.Nd” on page 6-67 for more detail.

Coiflets

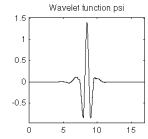
Built by I. Daubechies at the request of R. Coifman. The wavelet function has 2*N* moments equal to 0 and the scaling function has 2*N*-1 moments equal to 0. The two functions have a support of length 6*N*-1. You can obtain a survey of the main properties of this family by typing `waveinfo('coif')` from the MATLAB command line. See “Coiflet Wavelets: coifN” on page 6-66 for more detail.



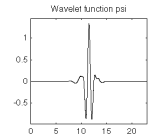
coif1



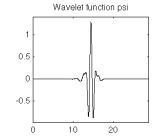
coif2



coif3



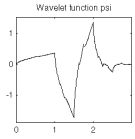
coif4



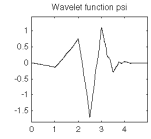
coif5

Symlets

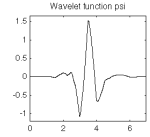
The symlets are nearly symmetrical wavelets proposed by Daubechies as modifications to the db family. The properties of the two wavelet families are similar.



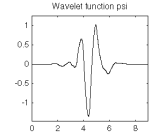
sym2



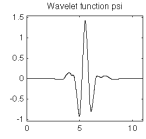
sym3



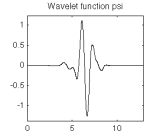
sym4



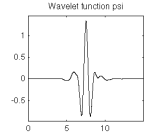
sym5



sym6



sym7

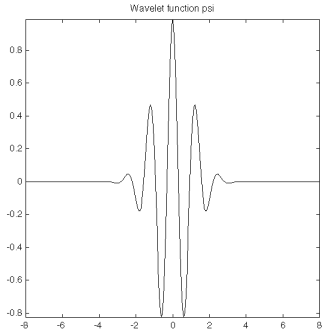


sym8

You can obtain a survey of the main properties of this family by typing `waveinfo('sym')` from the MATLAB command line. See “Symlet Wavelets: symN” on page 6-65 for more detail.

Morlet

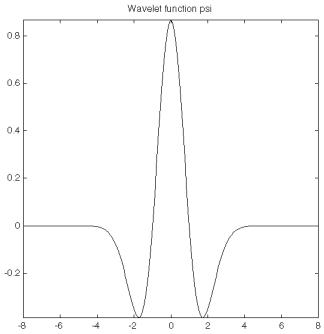
This wavelet has no scaling function, but is explicit.



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” on page 6-72 for more detail.

Mexican Hat

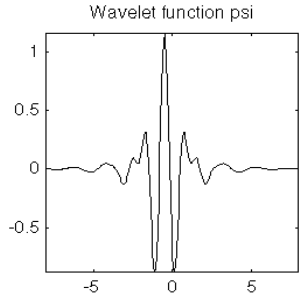
This wavelet has no scaling function and is derived from a function that is proportional to the second derivative function of the Gaussian probability density function.



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” on page 6-71 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” on page 6-69 for more detail.

2

Using Wavelets

The Wavelet Toolbox contains graphical tools and command line functions that let you:

- Examine and explore characteristics of individual wavelets and wavelet packet.
- Examine statistics of signals and signal components.
- Perform a continuous wavelet transform of a one-dimensional signal.
- Perform discrete analysis and synthesis of one- and two-dimensional signals.
- Perform wavelet packet analysis of one- and two-dimensional signals.
- Compress and remove noise from signals and images.

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

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

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

- Continuous Wavelet Analysis (One-Dimensional)
- One-Dimensional Discrete Wavelet Analysis
- Two-Dimensional Discrete Wavelet Analysis

Chapter 5 describes using the toolbox to perform wavelet packet analysis.

Continuous Wavelet Analysis (One-Dimensional)

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

The Wavelet Toolbox requires only one function for continuous wavelet analysis: `cwt`. You'll find full information about this function in the Command Reference (Chapter 8).

In this section, you'll learn how to:

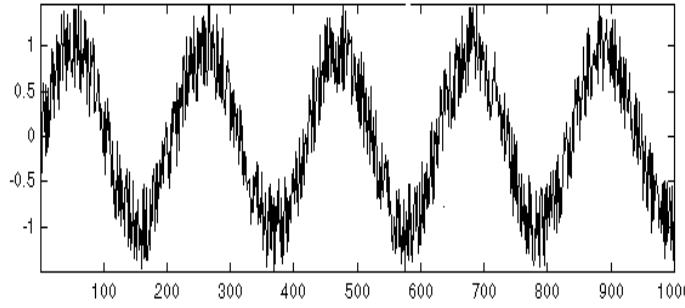
- Load a signal
- Perform a continuous wavelet transform of a signal
- Produce a plot of the coefficients
- Zoom in on detail
- Display coefficients in normal or absolute mode
- Choose the scales at which analysis is performed

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.



Loading a Signal.

1 From the MATLAB prompt, type:

```
» load noissin;
```

You now have the signal noissin in your workspace:

```
» whos
```

```
      Name       Size       Elements   Bytes     Class
      noissin    1 by 1000  1000       8000      double array
```

Performing a Continuous Wavelet Transform.

2 Use the cwt command. Type:

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

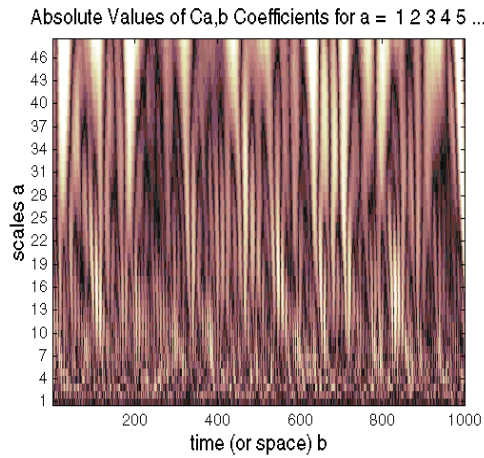
The arguments to cwt specify the signal to be analyzed, the scales of the analysis, and the wavelet to be used. The returned argument c contains the coefficients at various scales. In this case, c is a 48-by-1000 matrix, each row of which corresponds to a single scale.

Plotting 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.

3 Type

```
» 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. The colormap for the picture above was changed to pink from the default `cool` by typing:

```
» colormap(pink)
```

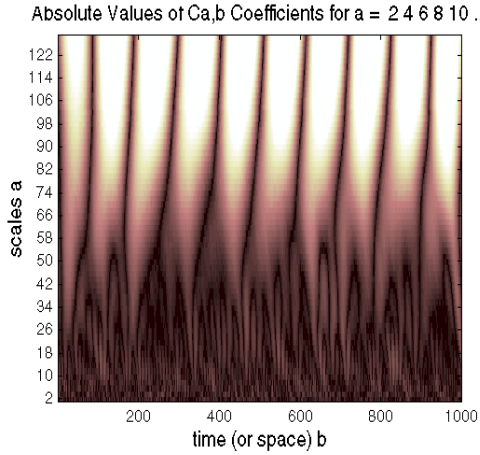
Choosing Scales for the Analysis. The second argument to `cwt` gives you fine control over the scale levels on which the continuous analysis is performed. In the previous example, we used all scales from 1 to 48, but you can construct any scale vector subject to these constraints:

- All scales must be real positive numbers.
- The initial scale must be positive.
- The scale increment must be positive.
- The highest scale cannot exceed a signal-dependent maximum.

4 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.

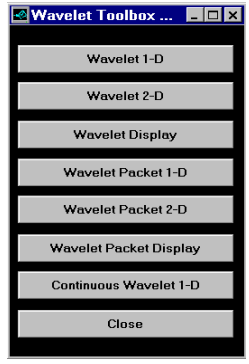
Continuous Analysis Using the Graphical Interface

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

Starting the Continuous Wavelet 1-D Tool.

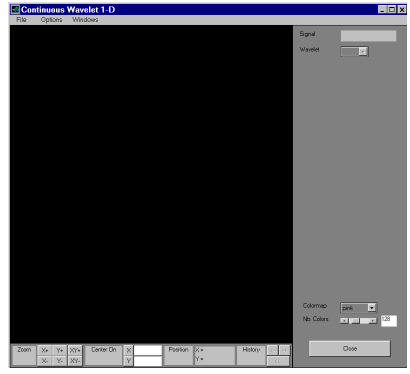
1 From the MATLAB prompt, type `wavemenu`.

The **Wavelet Toolbox Main Menu** appears.



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

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



Loading a Signal.

3 Choose the **File⇒Load Signal** menu option.

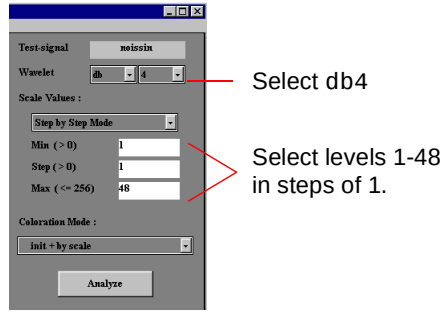
4 When the **Load Signal** dialog box appears, select the demo MAT-file `noissin.mat`, which should reside in the MATLAB directory `toolbox/wavelet/wavedemo`. Click the **OK** button.

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

Performing a Continuous Wavelet Transform.

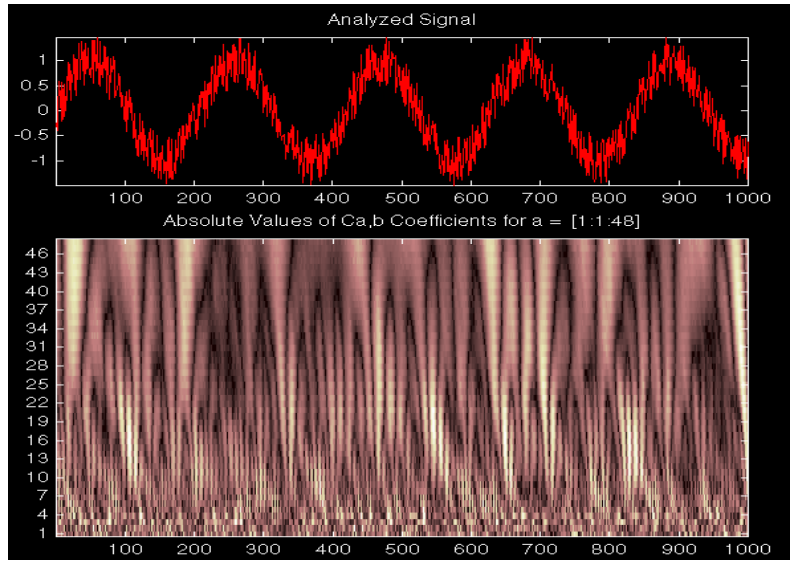
To start our analysis, let's perform an analysis using the `db4` wavelet at scales 1 through 48, just as we did using command line functions in the previous section.

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



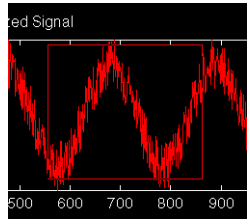
6 Click the **Analyze** button.

After a pause for computation, the tool displays the coefficients plot.



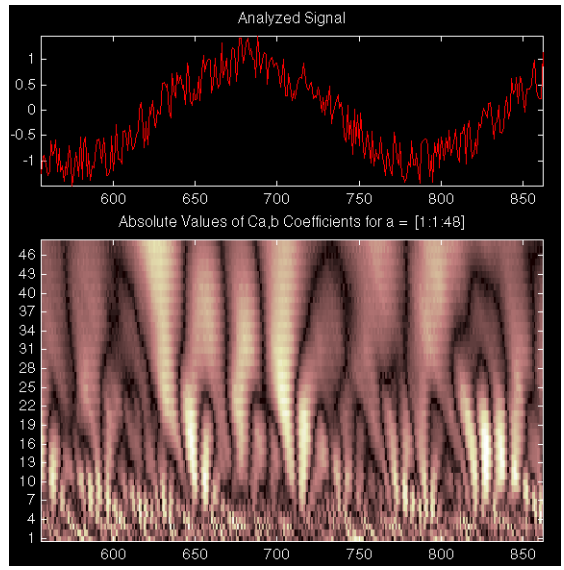
Zooming in on Detail.

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



8 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.

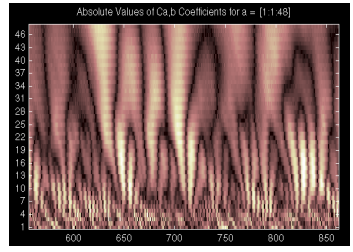


As with the command line analysis on the preceding pages, you can change the scales or the analyzing wavelet and repeat the analysis. To do this, just edit the necessary fields and press the **Analyze** button.

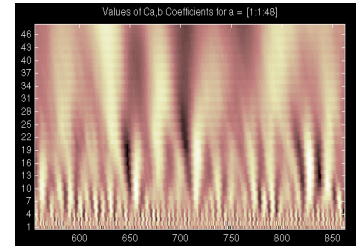
Viewing Normal or Absolute Coefficients. The **Continuous Wavelet 1-D** tool allows you to plot either the absolute values of the wavelet coefficients, or the coefficients themselves.

9 Choose either **Absolute Mode** or **Normal Mode** from the **Coloration Mode** menu, located just above the **Analyze** button. In normal mode, the colors are scaled between the minimum and maximum of the coefficients. In absolute mode, the colors are scaled between zero and the maximum absolute value of the coefficients (for more details on the **Coloration Mode**, (See “Continuous Wavelet Tool Features” on page A-1).

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 your disk.

You can:

- Load signals from your disk into the **Continuous Wavelet 1-D** tool.
- Save wavelet coefficients from the **Continuous Wavelet 1-D** tool into your 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 that has the same name as the signal variable itself.

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

```
» save 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.

Saving Wavelet Coefficients

The Continuous Wavelet 1-D tool lets you save wavelet coefficients to your disk. The toolbox creates a MAT-file in the current directory 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 directory and filename for storing the coefficients.

Consider the demo analysis:

File⇒Demo 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       Elements   Bytes     Class
      coefs     64 by 2188 140032     1120256   double array
      scales    1 by 64    64         512       double 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.

One-Dimensional Discrete Wavelet Analysis

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

The Wavelet Toolbox provides these functions for one-dimensional signal analysis. For more information, see the Command Reference (Chapter 8).

Analysis Decomposition Functions:

Function Name	Purpose
`dwt`	One-step decomposition
`wavedec`	Decomposition

Synthesis Reconstruction Functions:

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

Decomposition Structure Utilities:Analysis Decomposition

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

Functions:

Function Name	Purpose
ddencmp	Provide default values for de-noising and compression
wdencmp	Wavelet de-noising and compression
wden	Automatic wavelet de-noising

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 multi-level 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 multi-level 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 three 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.

Loading a Signal.

1 From the MATLAB prompt, type:

```
» load leleccum;
```

2 Set the variables. Type:

```
» s = leleccum(1:3920);
» ls = length(s);
```

Performing A One-Step Wavelet Decomposition of a Signal.

3 Perform a one-step 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`).

Constructing Approximations and Details from the Coefficients.

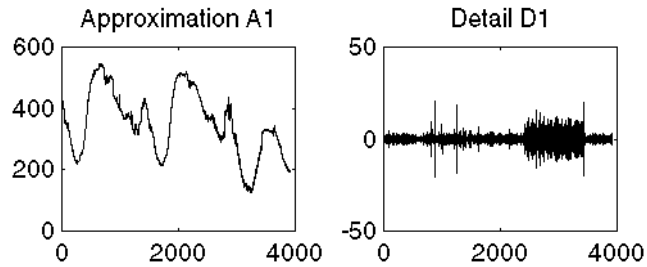
4 To construct the level 1 approximation and detail (`A1` and `D1`) from the coefficients `cA1` and `cD1`, type:

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

Displaying the Approximation and Detail.

5 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')
```



Regenerating a Signal by Inverse Wavelet Transform.

6 To find the inverse transform, type:

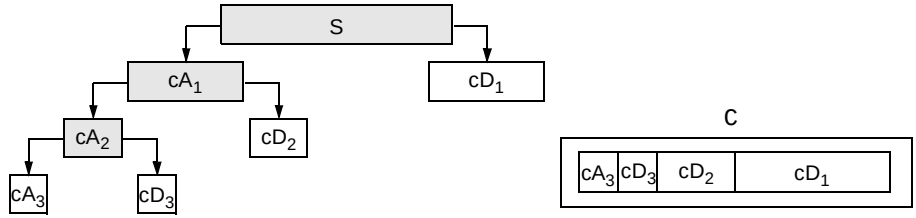
```
» A0 = idwt(cA1,cD1,'db1',ls);
```

Performing a Multilevel Wavelet Decomposition of a Signal.

7 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.



Extracting Approximation and Detail Coefficients.

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

```
» cA3 = appcoef(C,L,'db1',3);
```

9 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);
```

Reconstructing the Level 3 Approximation.

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

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

Reconstructing the Level 1, 2, and 3 Details.

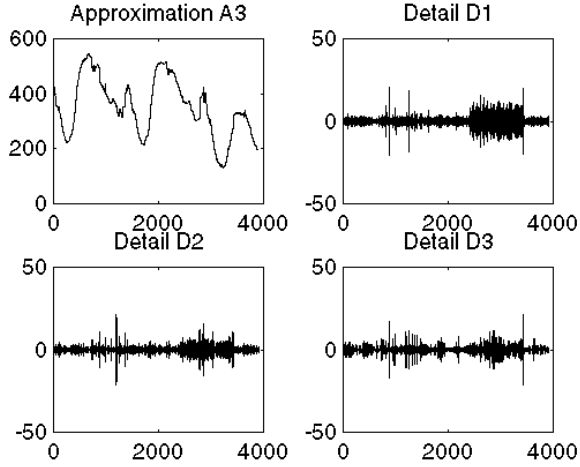
11 To reconstruct the details at levels 1, 2 and 3, from `C`, type:

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

Displaying the Results of a Multilevel Decomposition.

12 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')
```



Reconstructing the Original Signal From the Level 3 Decomposition.

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

```
» A0 = waverec(C,L,'db1');
```

Crude De-noising of a Signal.

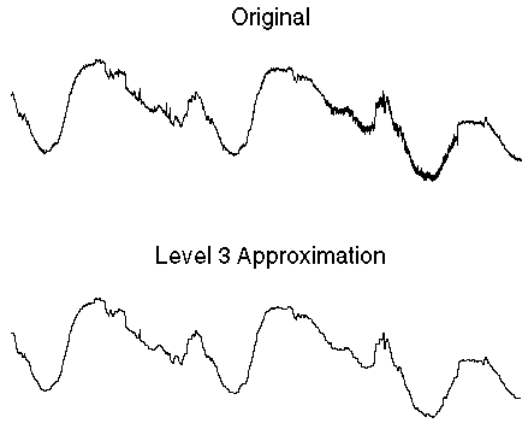
Using wavelets to remove noise from a signal requires identifying which component or components contain the noise and then reconstructing the signal without those components.

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 shows.

14 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.

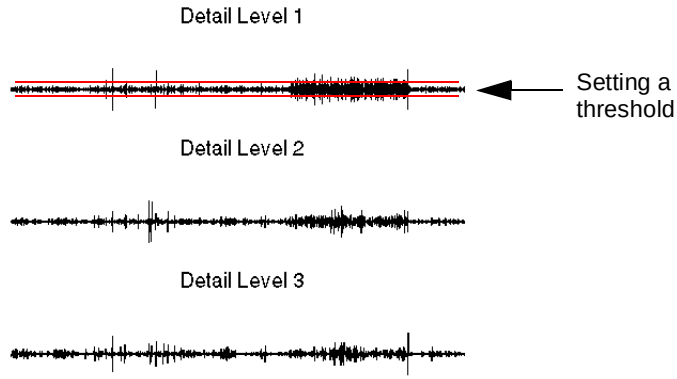
Optimal de-noising requires a more subtle approach called *thresholding*. This involves discarding only the portion of the details that exceeds a certain limit.

Removing Noise by Thresholding.

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

15 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, and 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.

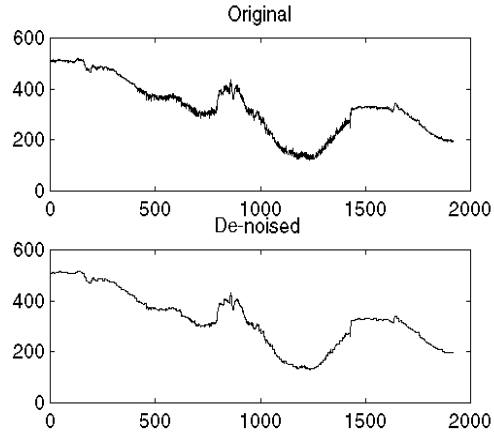
16 To de-noise the signal, use the ddencmp command to calculate the default parameters and the wdencmp command to perform the actual de-noising, type:

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

Note that we pass in to `wdencmp` the results of the decomposition (`C` and `L`) we calculated in Step 7 on page 2-16. We also specify that we used the `db1` wavelet to perform the original analysis, and we specify the global thresholding option `'gbl'`. See the `ddencmp` and `wdencmp` reference entries for more information about the use of these commands.

17 To display both the original and de-noised signals, type:

```
» subplot(2,1,1); plot(s(2000:3920)); title('Original')
» subplot(2,1,2); plot(clean(2000:3920)); title('De-noised')
```



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 Wavelet Toolbox graphical interface tools include an easy-to-use de-noising feature that includes automatic thresholding.

One-Dimensional Analysis Using the Graphical Interface

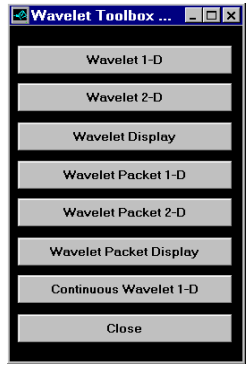
In this section we explore the same electrical consumption signal as in the previous section, but we use the graphical interface tools to analyze the signal.

Starting the 1-D Wavelet Analysis Tool.

1 From the MATLAB prompt, type:

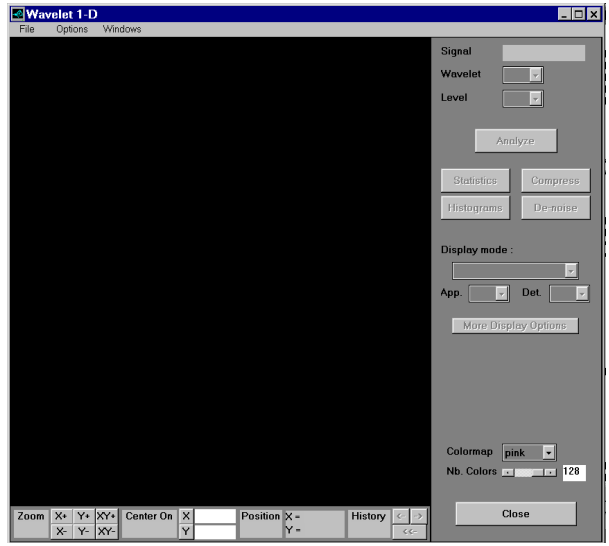
```
» wavemenu.
```

The **Wavelet Toolbox Main Menu** appears.



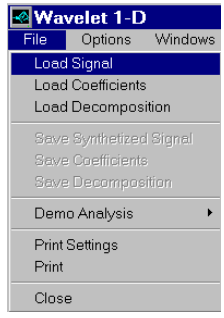
2 Click the **Wavelet 1-D** menu item.

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

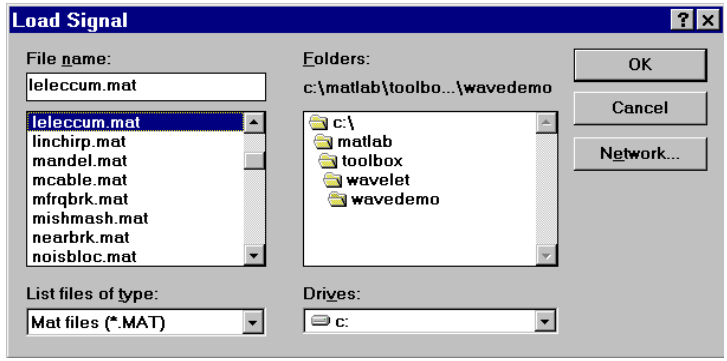


Loading a Signal.

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



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

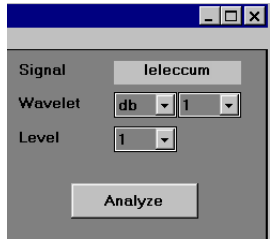


The electrical consumption signal is loaded into the **Wavelet 1-D** tool.

Performing A One-Step Wavelet Decomposition of a Signal.

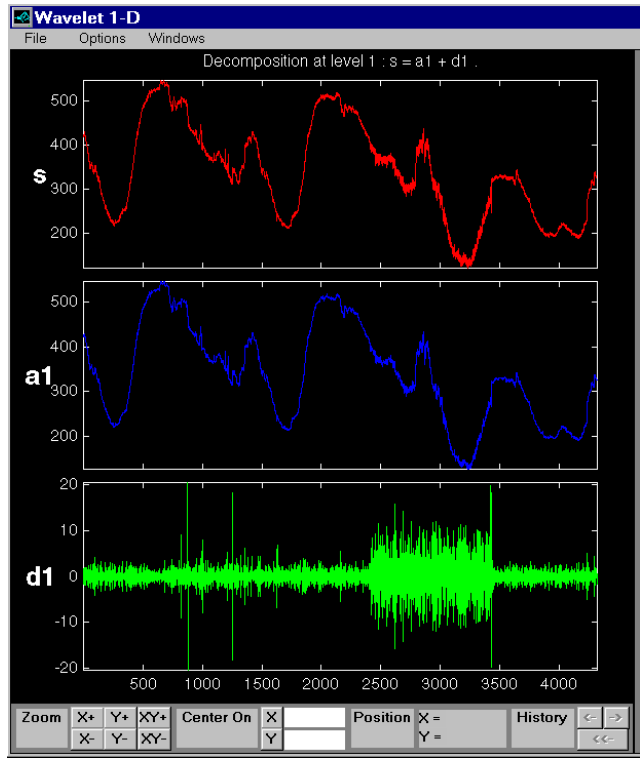
To start our analysis, let's perform a single-level decomposition using the `db1` wavelet, just as we did using command line functions in the previous section.

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



6 Click the **Analyze** button.

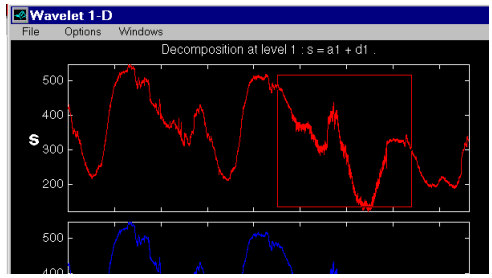
After a pause for computation, the tool displays the decomposition.



Zooming in On Relevant Detail.

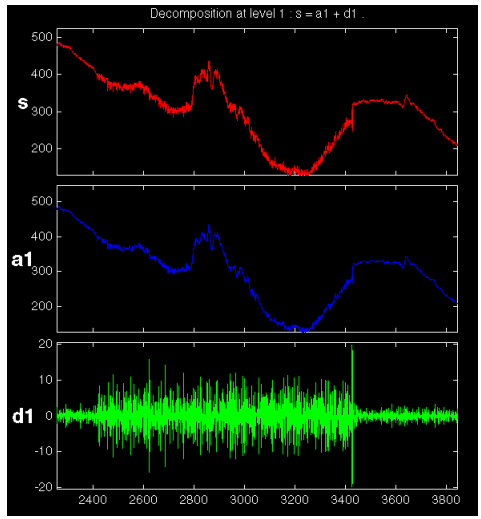
One advantage of using the graphical interface tools is that you can zoom in easily on any part of the signal and examine it in greater detail.

7 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.



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

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

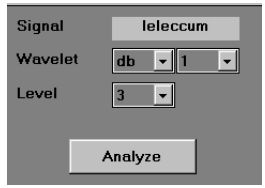


The other zoom controls do more or less what you'd expect them to. The **X-**button, for example, zooms out horizontally. The history function keeps track of all your views of the signal. Return to a previous zoom level by clicking the left arrow button.

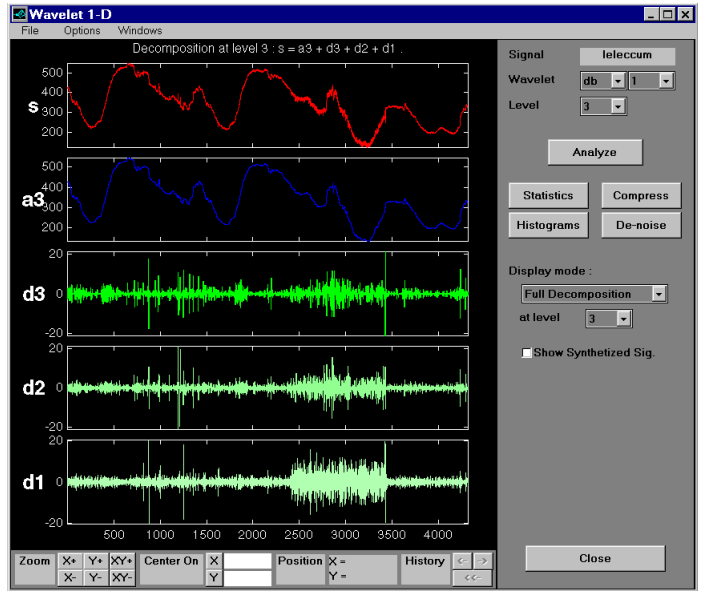
Performing a Multi-Level Decomposition of a Signal.

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:

9 Simply 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.



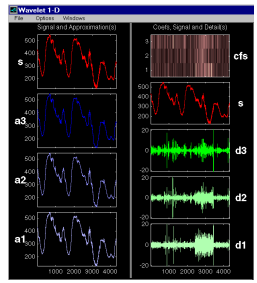
Selecting Different Views of the Decomposition.

The menu at the middle right lets you choose different views of the wavelet decomposition.

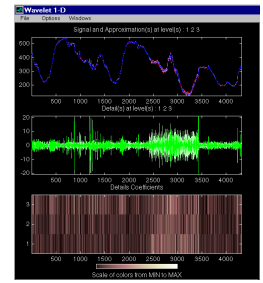
The default display mode is called "Full Decomposition Mode." Other alternatives include:

- "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.

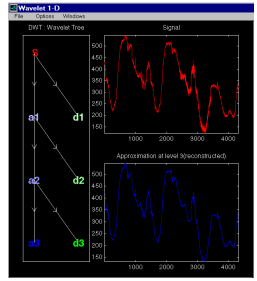
- "Show and Scroll Mode," which displays three windows. The first shows the original signal superimposed on an approximation you select. The second window shows a detail you select. The third window shows the wavelet coefficients.



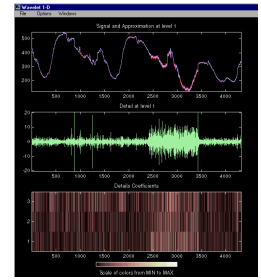
Separate Mode



Superimpose Mode



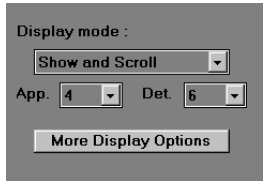
Tree Mode



Show & Scroll Mode

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

Depending on which display mode you select, you may have access to additional display options through the **More Display Options** button.

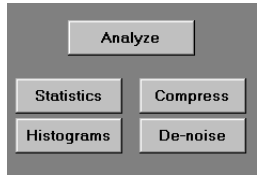


These options include the ability to suppress the display of various components, and to choose whether or not to display the original signal along with the details and approximations.

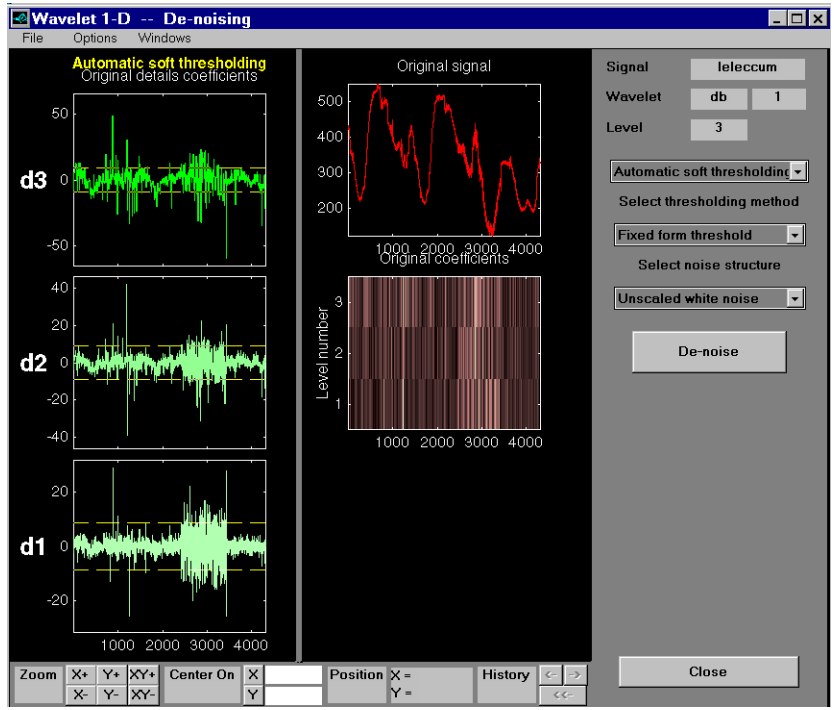
Removing Noise From a Signal.

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

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



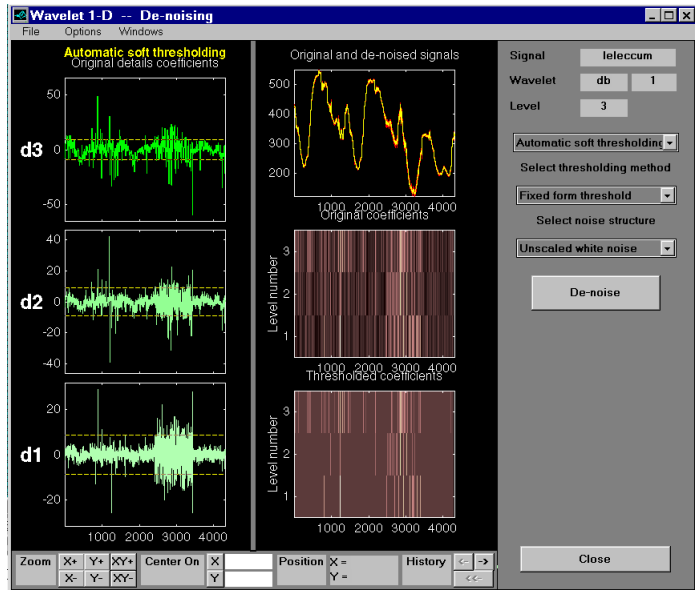
The **Wavelet 1-D De-noising** window appears.



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

11 Continue by clicking the **De-noise** button.

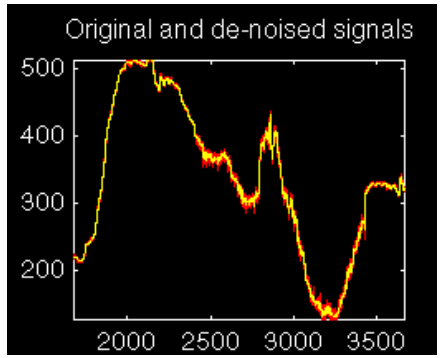
The de-noised signal appears superimposed on the original. The tool also plots the wavelet coefficients of both signals.



Zoom in on the plot of the original and de-noised signals for a closer look.

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

The **De-noise** window magnifies your view. By default, the original signal is shown in red, and the de-noised signal in yellow.



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

You cannot have the **De-noise** 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).

Refining an Analysis.

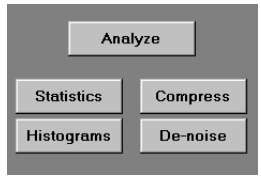
The graphical tools make it easy to refine an analysis any time you want to. Up to now, we've looked at a level 3 analysis using `db1`. Let's refine our analysis of the electrical consumption signal using the `db3` wavelet at level 5.

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

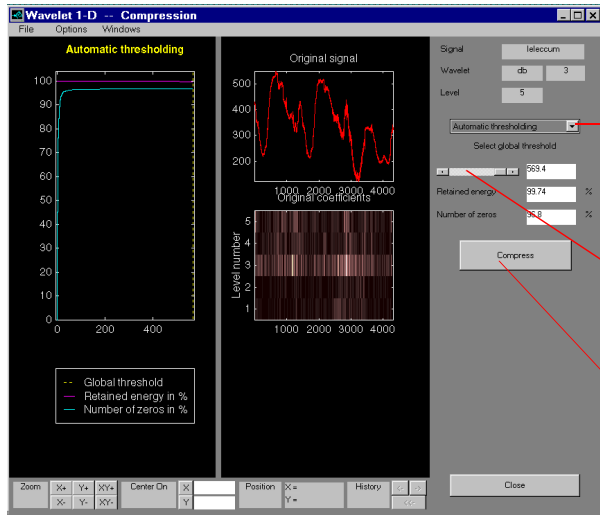
Compressing a Signal.

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

15 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.

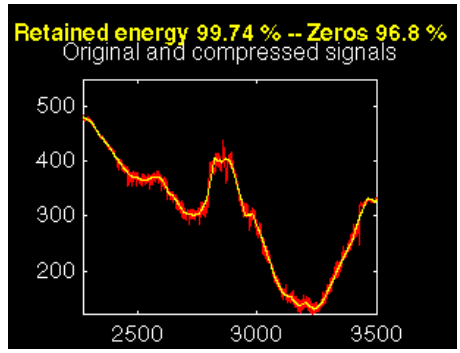


While you always have the option of choosing manual thresholding, here we'll take advantage of the automatic thresholding feature for quick and easy compression.

Note: If you want to experiment with manual thresholding, choose that 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.

16 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.



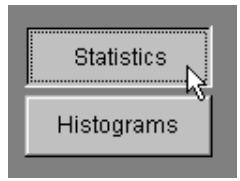
You can see that the compression process removed most of the noise, but preserved 99.74% of the energy of the signal. The automatic thresholding was very efficient, zeroing out all but 3.2% of the wavelet coefficients.

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

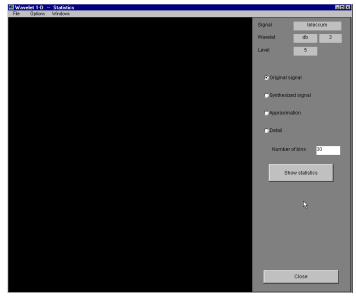
Showing Statistics.

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

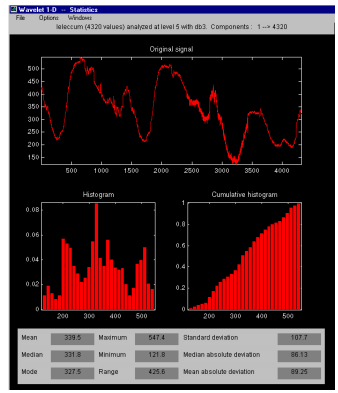
18 From the **Wavelet 1-D** tool, click the **Statistics** button.



The **Wavelet 1-D Statistics** window appears.



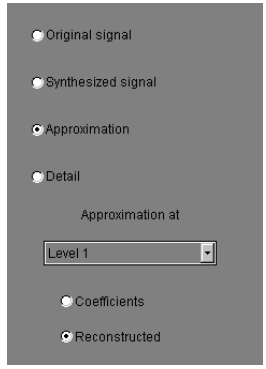
19 Select the signal or signal component whose statistics you want to examine. Click on the appropriate radio button, then press the **Show Statistics** button. Here, we've chosen to examine the original signal:



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

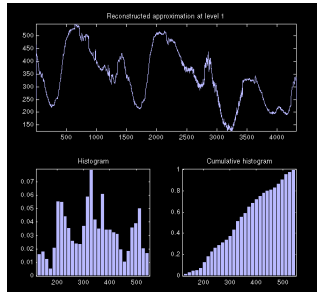
In addition, the tool provides frequency-distribution diagrams (histograms and cumulative histograms). You can plot these histograms separately using the **Histograms** button from the **Wavelets 1-D** window.

20 Select the **Approximation** radio button. A menu appears from which you choose the level of the approximation you want to examine.



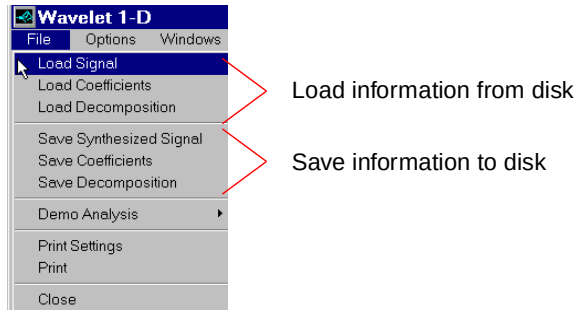
21 Select Level 1 and again click the **Show Statistics** button.

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 your disk.



Saving Information to the 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.

For example, load the demo analysis: **File⇒Demo Analysis⇒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-noise** or **Wavelet 1-D 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 directory and filename for the MAT-file. For this example, choose the name `synthsig`.

To load the signal into your workspace, simply type:

```
» load synthsig
» whos
```

```
      Name      Size       Elements  Bytes     Class
      synthsig  1 by 1000  1000      8000      double array
```

Saving Discrete Wavelet Transform Coefficients.

The **Wavelet 1-D** tool lets you save the coefficients of a discrete wavelet transform (DWT) to your disk. The toolbox creates a MAT-file in the current directory 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 directory and filename for storing the coefficients.

Consider the demo analysis:
File⇒Demo Analysis⇒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       Elements  Bytes     Class
      coefs     1 by 2190  2190      17520     double array
      longs     1 by 7     7         56        double 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`.

Saving Decompositions.

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

Open the **Wavelet 1-D** tool and load the demo analysis:
File⇒Demo Analysis⇒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 directory and filename for storing the decomposition data. Type the name `wdecex`.

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

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

```
      Name      Size       Elements  Bytes     Class
      coefs     1 by 1023  1023      8184      double array
      data_name 1 by 8     8         64        double array
      longs     1 by 64    64        512       double array
      wave_name 1 by 3     3         24        double array
```

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 that has the same name as the signal variable itself.

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

```
» save warma
```

The workspace variable warma must be a vector.

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

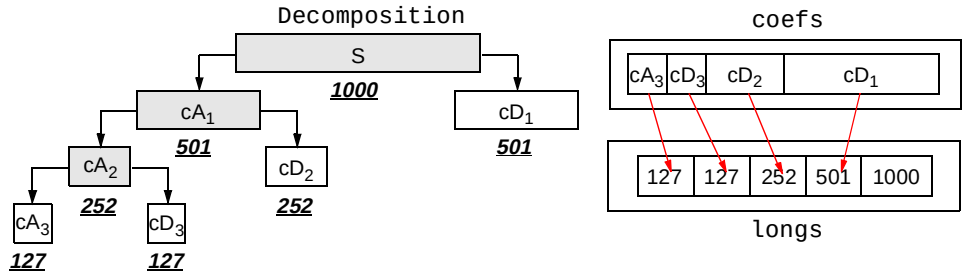
To load this signal into the **Wavelet 1-D** tool, use the menu option **File⇒Load Signal**.

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

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 containing only 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
```

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 directory 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` (wavelet analysis 1-D). The MAT-file must contain these variables:

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

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

```
» save myfile.wa1
```

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 directory and file in which your data reside.

Two-Dimensional Discrete Wavelet Analysis

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

The Wavelet Toolbox provides these functions for image analysis. For more information, see the Command Reference (Chapter 8).

Analysis-Decomposition Functions:

Function Name	Purpose
`dwt2`	One-step decomposition
`wavedec2`	Decomposition

Synthesis-Reconstruction Functions:

Function Name	Purpose
`idwt2`	One-step 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
wdencmp	Wavelet de-noising and compression

In this section, you'll learn:

- How to load an image
- How to analyze an image
- How to perform one-step and multi-level 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 using directly `image(I)` in order 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` in Chapter 8).

Loading an Image.

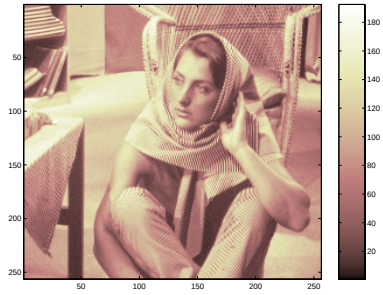
1 From the MATLAB prompt, type:

```
» load wbarb;
» whos
```

```
Name        Size          Elements    Bytes      Class
X           256 by 256    65536       524288     double array
map         192 by 3      576         4608       double array
```

2 Display the image. Type:

```
» image(X); colormap(map)
```



Converting an Indexed Image to a Grayscale Image.

3 If the colorbar is smooth, the wavelet transform can be directly applied to the indexed image, otherwise the indexed image should be converted to grayscale format. See “Working with Indexed Images” at the end of this chapter for more information.

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

Performing A One-Step Wavelet Decomposition of an Image.

4 Perform a one-step 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).

Constructing Approximations and Details from the Coefficients.

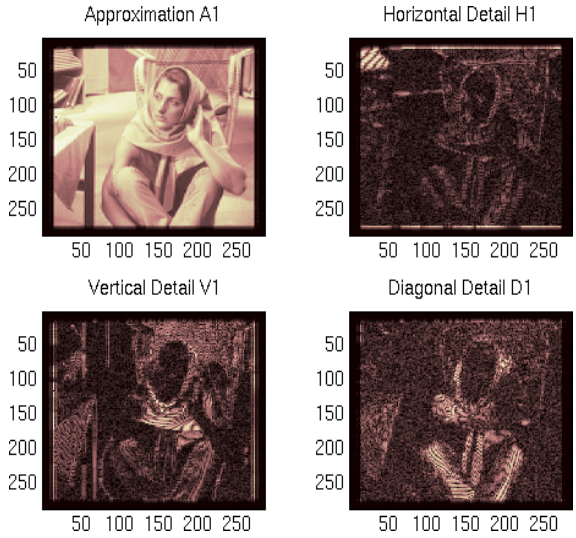
5 To construct the level-one approximation and details (A_1, H_1, V_1, and D_1) from the coefficients cA_1, cH_1, cV_1, and cD_1, type:

```
» 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);
```

Displaying the Approximation and Details.

6 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')
```



Regenerating an Image by One-Step Inverse Wavelet Transform.

7 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.

Performing a Multi-Level Wavelet Decomposition of an Image.

8 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.

Extracting Approximation and Detail Coefficients.

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

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

10 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);
```

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

Reconstructing the Level 2 Approximation.

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

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

Reconstructing the Level 1 and 2 Details.

12 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);
```

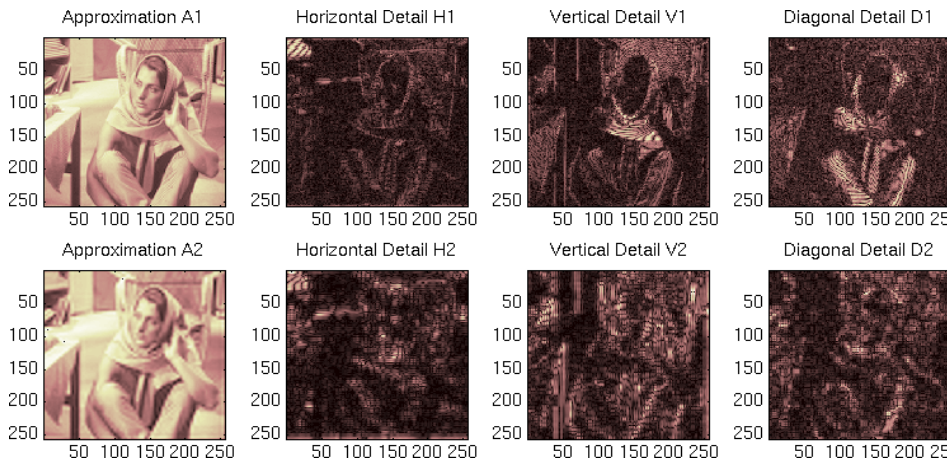
Displaying the Results of a Multi-Level Decomposition.

Note: With all the details involved in a multi-level 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 67.

---

13 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')
```



Reconstructing the Original Image from the Multilevel Decomposition.

14 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 multi-level decomposition.

Compressing an Image.

15 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:

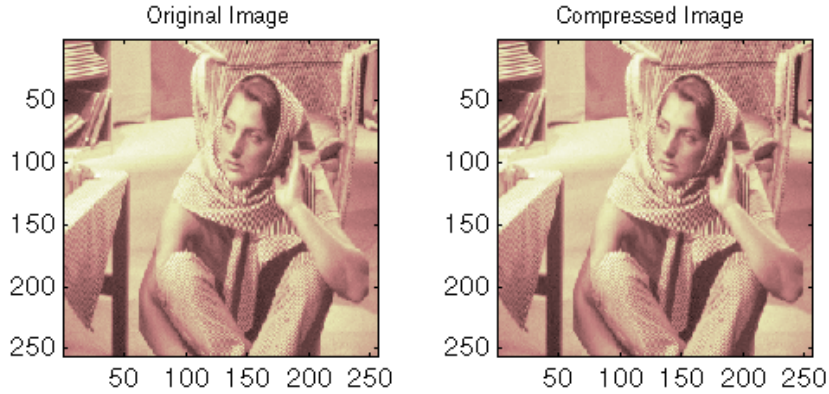
```
» [thr,sorh,keepapp] = ddencmp('cmp','wv',X);
» [Xcomp,CXC,LXC,PERF0,PERFL2] =
» wdencmp('gbl',C,S,'bior3.7',2,thr,sorh,keepapp);
```

Note that we pass in to `wdencmp` the results of the decomposition (`C` and `S`) we calculated in Step 7 on page 2-47. We also specify the `bior3.7` wavelet, because we used this wavelet to perform the original analysis. Finally, we specify the global thresholding option `'gbl'`. See the `ddencmp` and `wdencmp` reference entries for more information about the use of these commands.

Displaying the Compressed Image.

16 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
   86.6550
» PERFL2
   99.9779
```

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

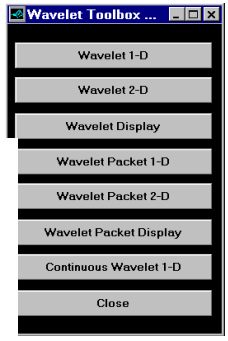
In this section we explore the same image as in the previous section, but we use the graphical interface tools to analyze the image.

Starting the 2-D Wavelet Analysis Tool.

1 From the MATLAB prompt, type

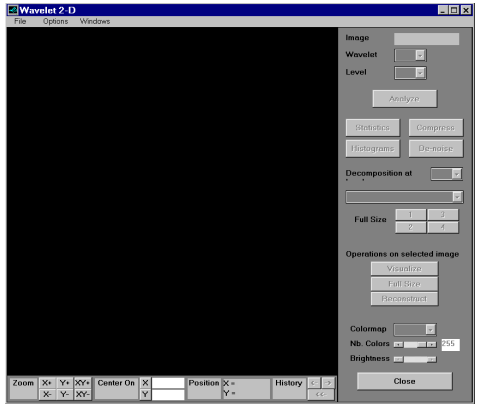
```
» wavemenu.
```

The **Wavelet Tool Main Menu** appears.



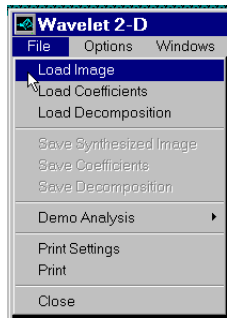
2 Click the **Wavelet 2-D** menu item.

The discrete wavelet analysis tool for two-dimensional image data appears.

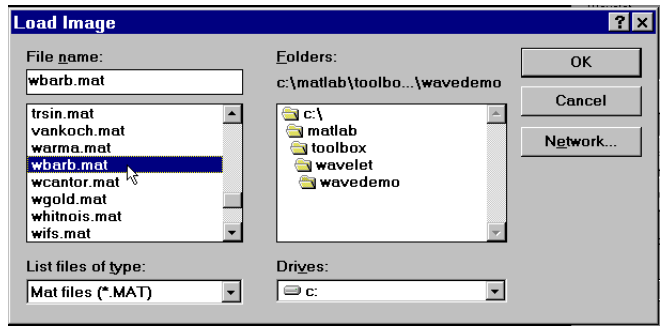


Loading an Image.

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



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

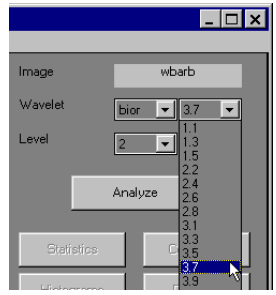


The image is loaded into the **Wavelet 2-D** tool.

Analyzing an Image.

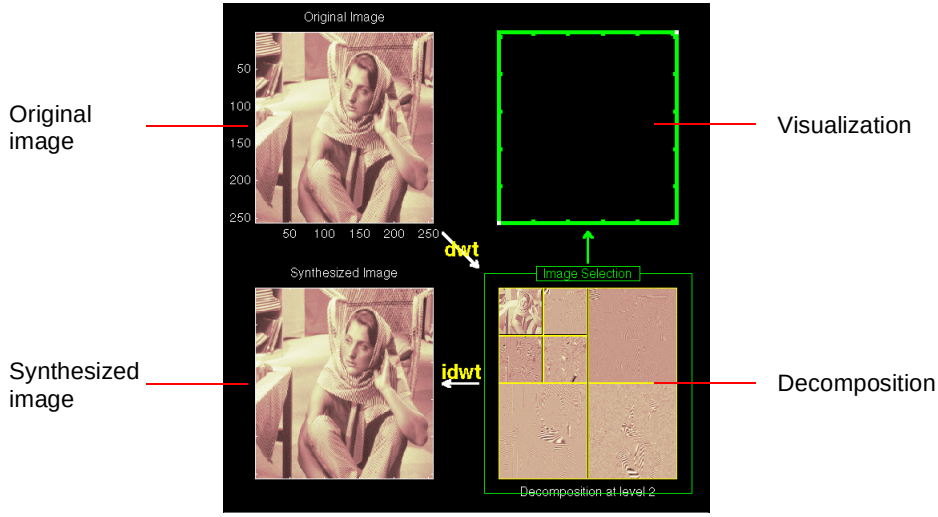
5 Using the **Wavelet** and **Levels** menus located to the upper right, determine the wavelet family and type as well as the number of levels to be used for the analysis.

For this analysis, select the `bior3.7` wavelet at level 2.



6 Click the **Analyze** button.

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

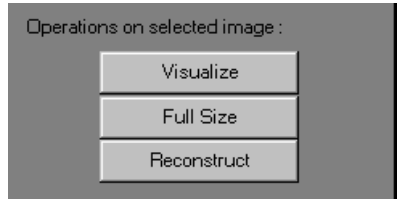


Using Square Mode Features.

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.

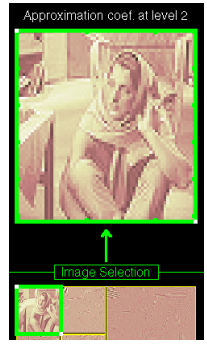
7 Click on any decomposition component in the lower right window.

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."



8 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.

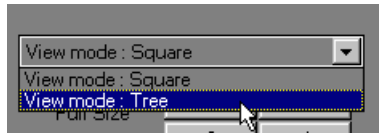


Visualized Approximation A_2

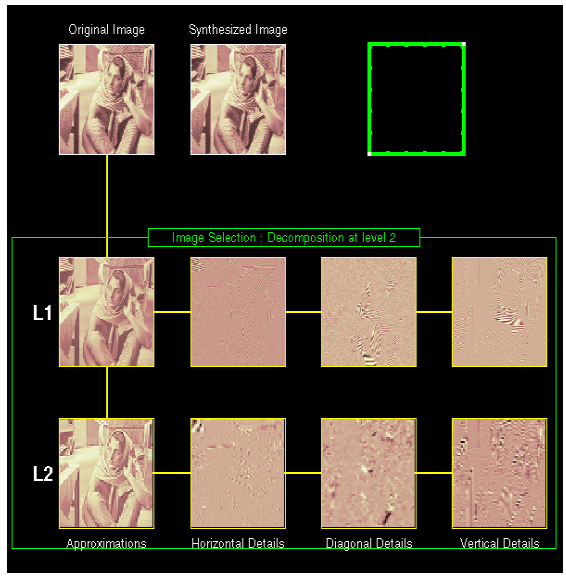
Reconstructed Approximation A_2

Using Tree Mode Features.

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



Your display changes to reveal:

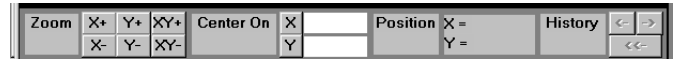


This is the same information shown in square mode, with in addition all the approximation coefficients, but arranged to emphasize the tree structure of the decomposition. The various buttons and menus work just the same as they do in square mode.

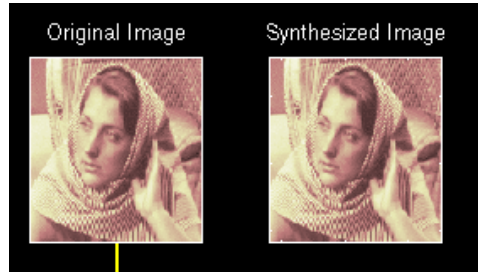
Zooming in on Detail.

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

11 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.

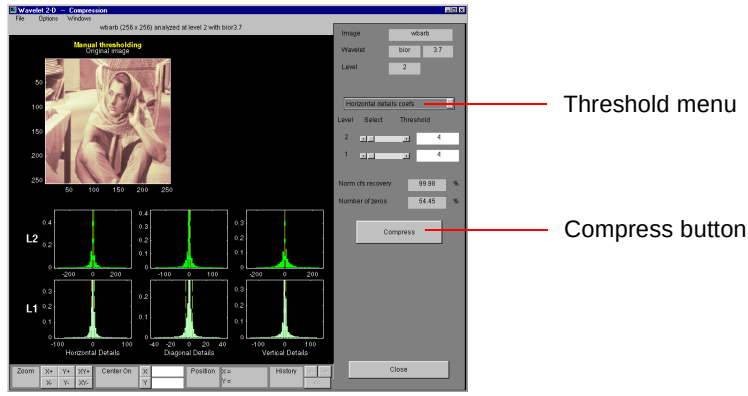


To zoom back to original magnification, click the **History** <— button.

Compressing an Image.

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

The **Wavelet 2-D Compression** window appears.



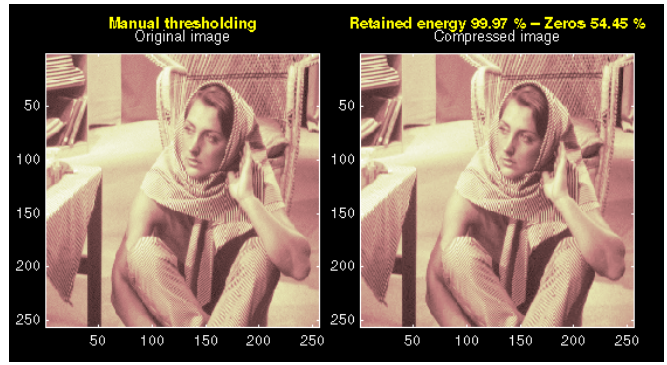
The tool automatically selects thresholding levels to provide a good initial balance between retaining the image's energy while minimizing the number of coefficients needed to represent the image.

However, you can also adjust thresholds manually using the **Thresholding** menu and sliders or corresponding edits. Select from the menu whether you want to adjust thresholds for horizontal, diagonal or vertical details, then use the sliders to make the actual adjustments for each level.

For this example, we'll accept the default thresholds.

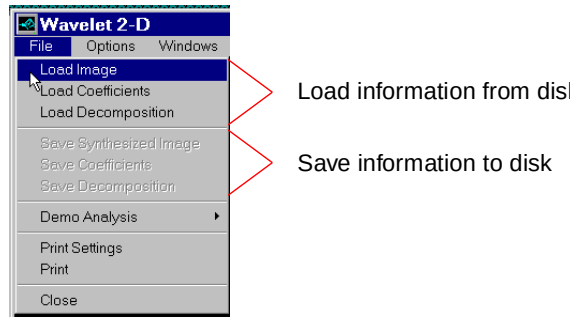
13 To compress the original image, click the **Compress** button.

After a pause for computation, the compressed image appears beside the original. Notice that compression eliminated almost half the coefficients, yet no more than one-half of one percent of image energy was lost in the process.



Importing and Exporting Information from the Graphical Interface

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



Saving Information to the Disk

You can save synthesized images, coefficients, and decompositions from the **Wavelet 2-D** tool to the 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.

For example, load the demo analysis
File⇒Demo 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 directory and filename for the MAT-file. For this example, choose the name symage.

To load the image into your workspace, simply type:

```
» load symage
» whos
```

```
Name      Size       Elements   Bytes     Class
map       64 by 3    192        1536      double array
symage    359 by 371 133189     1065512   double array
```

Saving Discrete Wavelet Transform Coefficients.

The **Wavelet 2-D** tool lets you save the coefficients of a discrete wavelet transform (DWT) to your disk. The toolbox creates a MAT-file in the current directory 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 directory and filename for storing the coefficients.

Consider the demo analysis
File⇒Demo Analysis⇒at level 3, with sym4 –-> Detail Durer.

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

```
» load durer
» whos
```

```
      Name   Size          Elements  Bytes     Class
      coefs  1 by 142299   142299    1138392   double array
      sizes  5 by 2        10        80        double array
```

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 your disk. The toolbox creates a MAT-file in the current directory with a name you choose, followed by the extension `wa2` (wavelet analysis 2-D).

Open the **Wavelet 2-D** tool and load the demo analysis **File⇒Demo 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 directory and filename for storing the decomposition data. Type the name `durer`.

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

```
» load durer.wa2 -mat
» whos
```

```
  Name      Size          Elements  Bytes     Class
  coefs     1 by 142299   142299    1138392   double array
  data_name 1 by 6        6         48        double array
  map       64 by 3       192       1536      double array
  sizes     5 by 2        10        80        double array
  wave_name 1 by 4        4         32        double array
```

Variables `coefs` and `sizes` contain the wavelet decomposition structure. Other variables contain the wavelet name, the colormap, and the filename containing the data.

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 a `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 that has the same name as the image matrix itself.

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

```
» save brain
```

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.

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

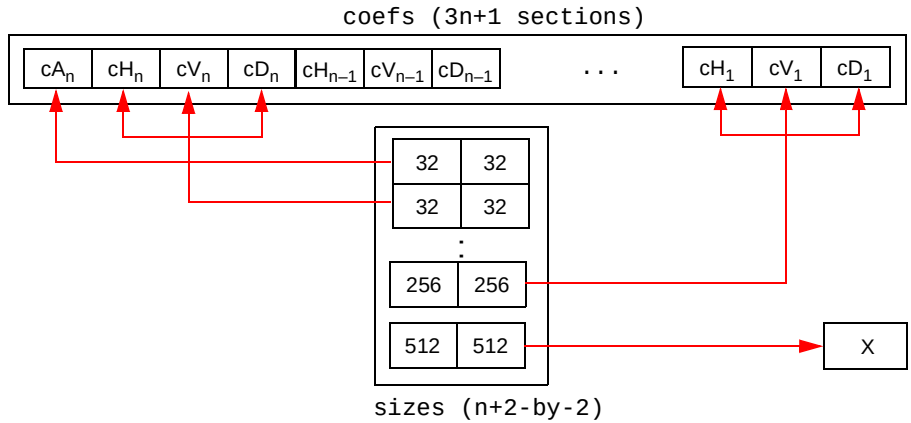
Note that the coefficients, approximations, and details produced by wavelet decomposition are not indexed image matrices.

In order 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.

Loading Discrete Wavelet Transform Coefficients.

To load discrete wavelet transform (DWT) coefficients into the **Wavelet 2-D** tool, you must first save the appropriate data in a MAT-file containing only two variables: coefficients vector `coefs` and bookkeeping matrix `sizes`.



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
```

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 directory 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` (wavelet analysis 2-D).

The MAT-file must contain these variables:

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

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

```
» save myfile.wa2
```

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 directory and file in which your data reside.

Working with Indexed Images

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

Understanding Images in MATLAB

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.)

In this supplement, the word *pixel* denotes 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`. Pixel is derived from *picture element* and usually denotes a single dot on a computer display. 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

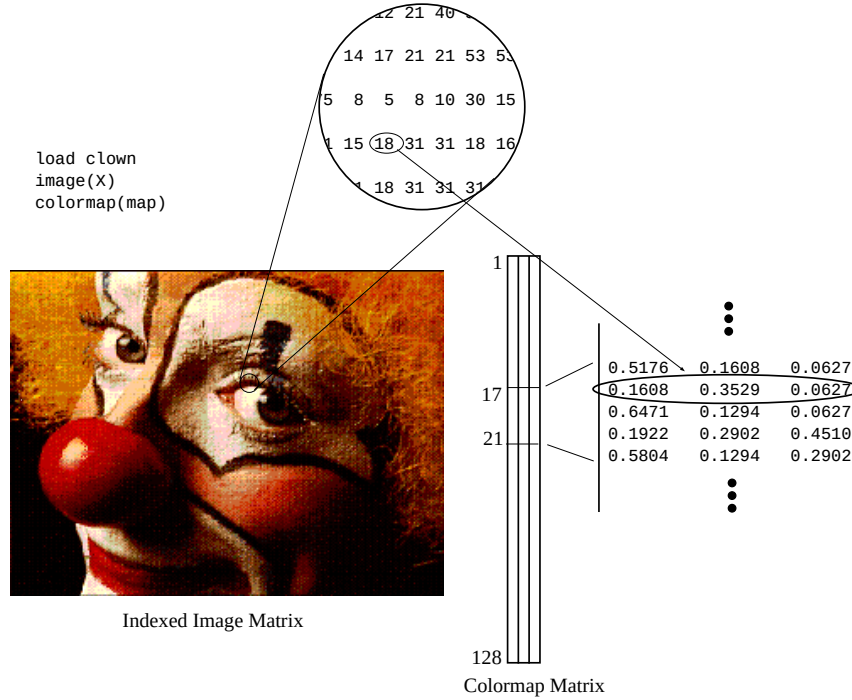
A typical color image requires two matrices: a colormap and an image matrix. The *colormap* is an ordered set of values that represent the colors in the image. For each image pixel, the *image matrix* contains a corresponding index into the colormap. (The elements of the image matrix are floating-point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

The Wavelet Toolbox supports only *indexed images* with linear, *monotonic* colormaps. These 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.

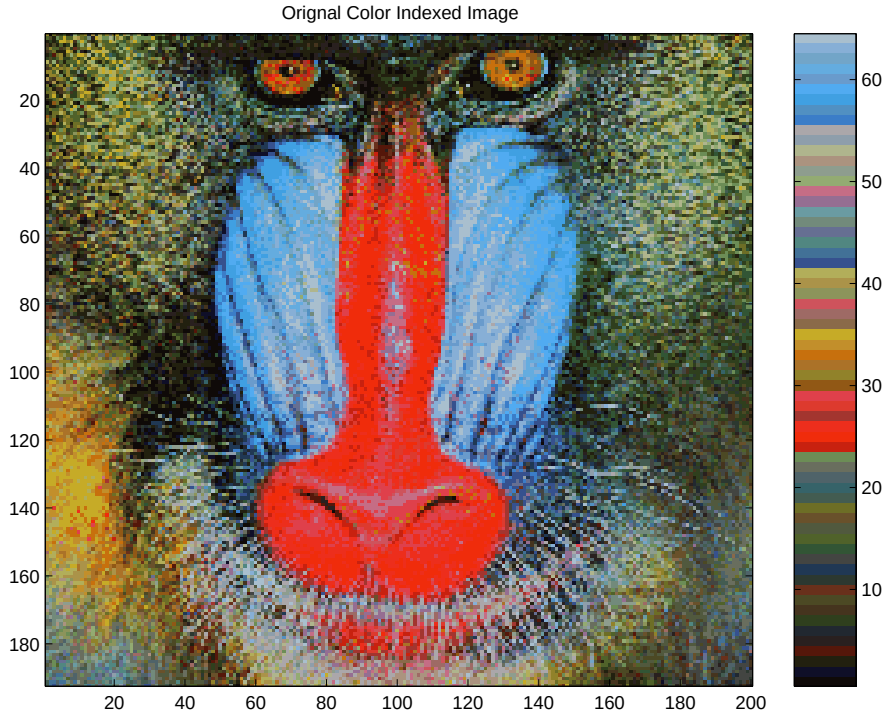
If the colormap is not provided, the graphical user interface tools display the image and processing results using a monotonic colormap with `max(max(X))-min(min(X))+1` colors.

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

In general, color indexed images do not have linear, monotonic colormaps and need to converted into the appropriate gray scale indexed image before performing a wavelet decomposition. The Image Processing Toolbox provides a comprehensive set of functions that let you easily convert between image types.

Should you not have the Image Processing Toolbox, the example below demonstrates how this conversion may be performed, using basic MATLAB commands.

```
» load xpmndrll
» whos
      Name       Size         Elements  Bytes     Class
      X2         192 by 200   38400     307200    double array
      map        64 by 3      192       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 we separate the color indexed image into its RGB components,

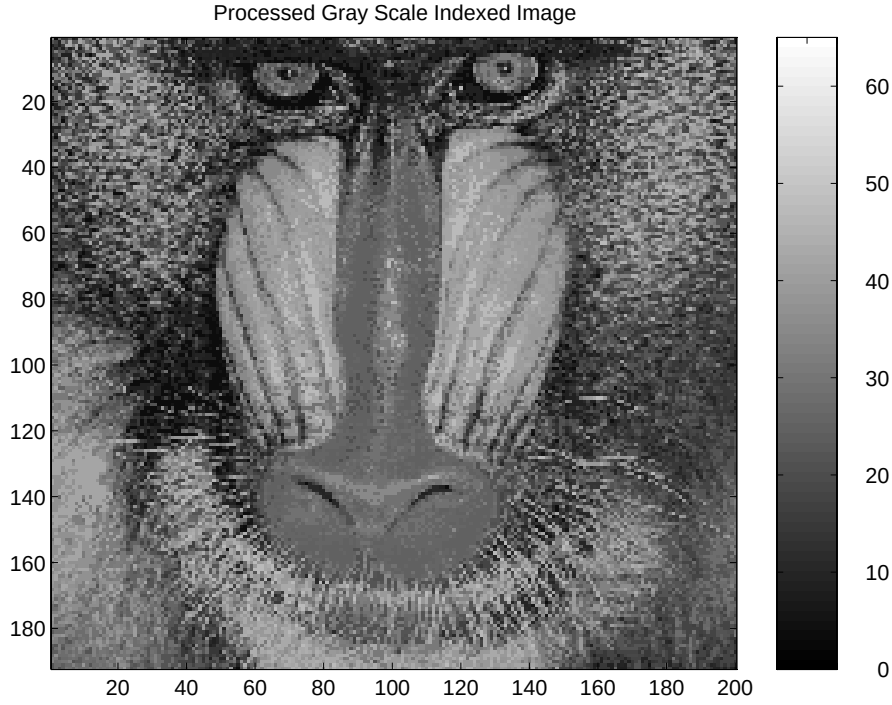
```
» R = map(X2,1); R = reshape(R,size(X2));
» G = map(X2,2); G = reshape(G,size(X2));
» B = map(X2,3); B = reshape(B,size(X2));
```

Now we 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;
```

Next, we 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, we save the converted image in a form compatible with the Wavelet Toolbox graphical user interface,

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

How Decompositions Are Displayed

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

In order 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.

3

Wavelet Applications

This chapter explores various applications of wavelets by presenting a series of sample analyses dealing with:

- Detecting Discontinuities and Breakdown Points (I and II)
- Detecting Long-Term Evolution
- Detecting Self-Similarity
- Identifying Pure Frequencies
- Suppressing Signals
- De-Noising Signals
- Compressing Signals

Each example is followed by a discussion of the usefulness of wavelet analysis for the particular application area under consideration.

Use the graphical interface tools to follow along:

1. From the MATLAB command line, type:

   ```
   » wavemenu.
   ```

2. Click on **Wavelets 1-D** (or other tool as appropriate).

3. Load the sample analysis by selecting the appropriate submenu item from **File⇒Demo Analysis.**

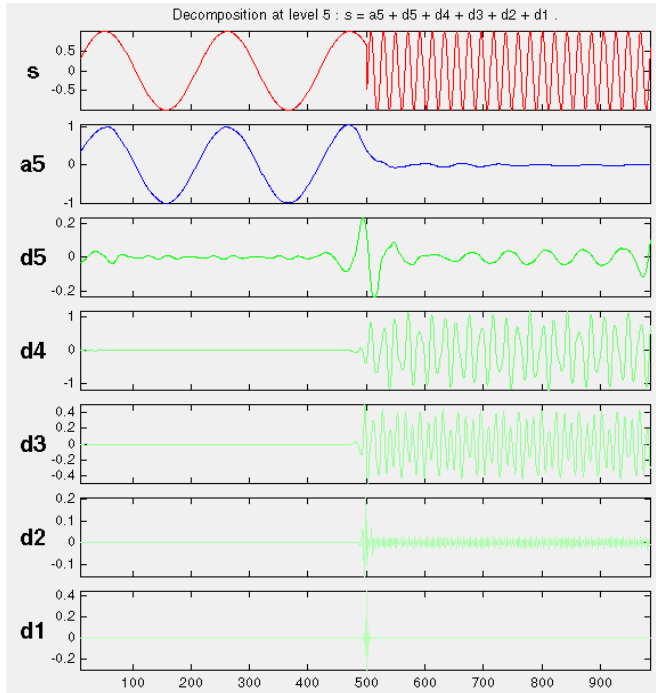
Feel free to explore on your own — use the different options provided in the graphical interface to look at different components of the signal, to compress or de-noise the signal, to examine signal statistics, or to zoom in and out on different signal features.

If you want, try loading the corresponding MAT-file from the MATLAB command line, and use the wavelet toolbox functions to investigate further the sample signals. The MAT-files are located in the directory: `toolbox/wavelet/wavedemo`.

There are also other signals in the `wavedemo` directory that you can analyze on your own.

Detecting Discontinuities and Breakdown Points I

The purpose of this example is to show how analysis by wavelets can detect the exact instant when a signal changes. The discontinuous signal consists of a "slow" sine wave abruptly followed by a "medium" sine wave.



Demo Analysis:
Frequency breakdown
MAT-file:
`freqbrk.mat`
Wavelet:
`db5`
Level:
5

The first- and second-level details (D_1 and D_2) show the discontinuity most clearly, because the rupture contains the high frequency part. Note that if we were *only* interested in identifying the discontinuity, `db1` would be a more useful wavelet to use for the analysis than `db5`.

The discontinuity is localized very precisely: only a small domain around time = 500 contains any large first- or second-level details.

Here is a noteworthy example of an important advantage of wavelet analysis over Fourier. If the same signal had been analyzed by the Fourier transform, we would not have been able to detect the instant when the signal's frequency changed, whereas it is clearly observable here.

Details D_3 and D_4 contain the “medium” sine wave. The “slow” sine is clearly isolated in approximation A_5, from which the higher-frequency information has been filtered.

Discussion

The deterministic part of the signal may undergo abrupt changes such as a jump, or a sharp change in the first or second derivative. In image processing, one of the major problems is edge detection, which also involves detecting abrupt changes. Also in this category, we find signals with very rapid evolutions such as transient signals in dynamic systems.

The main characteristic of these phenomena is that the change is localized in time or in space.

The purpose of the analysis is to determine:

- The site of the change (e.g., time or position),
- The type of change (a rupture of the signal, or an abrupt change in its first or second derivative),
- The amplitude of the change.

The local aspects of wavelet analysis are well adapted for processing this type of event, as the processing scales are linked to the speed of the change.

Guidelines for Detecting Discontinuities

Short wavelets are often more effective than long ones in detecting a signal rupture. In the initial analysis scales, the support is small enough to allow fine analysis. The shapes of discontinuities that can be identified by the smallest wavelets are simpler than those that can be identified by the longest wavelets.

Therefore, to identify:

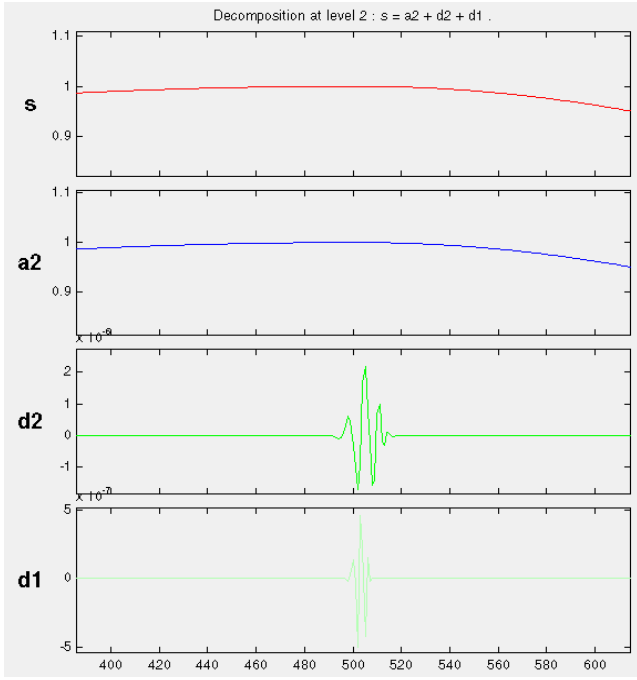
- A signal discontinuity, use the `haar` wavelet.
- A rupture in the *j*-th derivative, select a sufficiently regular wavelet with at least *j* vanishing moments. (See Detecting Discontinuities and Breakdown Points II on page 3-6.)

The presence of noise, which is after all a fairly common situation in signal processing, makes identification of discontinuities more complicated. If the first levels of the decomposition can be used to eliminate a large part of the noise, the rupture is sometimes visible at deeper levels in the decomposition.

Check, for example, the sample analysis **File⇒Demo Analysis⇒ramp + white noise** (MAT-file `wnoislop`). The rupture is visible in the level-six approximation (A_6) of this signal.

Detecting Discontinuities and Breakdown Points II

The purpose of this example is to show how analysis by wavelets can detect a discontinuity in one of a signal's derivatives. The signal, while apparently a single smooth curve, is actually composed of two separate exponentials that are connected at time = 500. The discontinuity occurs only in the second derivative, at time = 500.



Demo Analysis:
Second derivative breakdown
MAT-file:
`scddvbrk.mat`
Wavelet:
db4
Level:
2

We have zoomed in on the middle part of the signal to show more clearly what happens around time = 500. The details are high only in the middle of the signal and are negligible elsewhere. This suggests the presence of high-frequency information — a sudden change or discontinuity — around time = 500.

Discussion

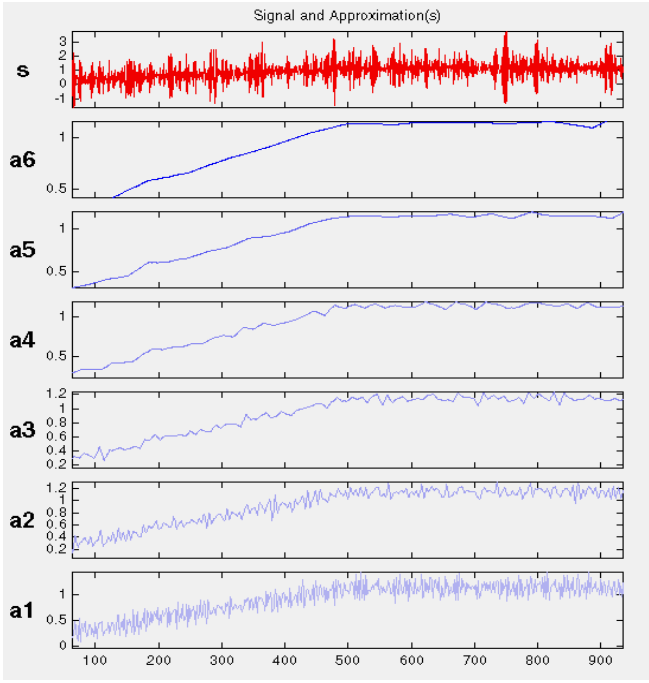
Regularity can be an important criterion in selecting a wavelet. We have chosen to use `db4`, which is sufficiently regular for this analysis. Had we chosen the `haar` wavelet, the discontinuity would not have been detected. If you try repeating this analysis using `haar` at level two, you'll notice that the details are equal to zero at time = 500.

Note that in order to detect a singularity the selected wavelet must be sufficiently regular, which implies a longer filter impulse response.

See Chapter 6, "Advanced Topics" for a discussion of the mathematical meaning of regularity and a comparison of the regularity of various wavelets.

Detecting Long-Term Evolution

The purpose of this example is to show how analysis by wavelets can detect the overall trend of a signal. The signal in this case is a ramp obscured by “colored” (limited-spectrum) noise. (We have zoomed in along the *x*-axis to avoid showing edge effects.)



Demo Analysis:
Ramp + colored noise
MAT-file:
`cnoislop.mat`
Wavelet:
`db3`
Level:
6

There is so much noise in the original signal, `s`, that its overall shape is not apparent upon visual inspection. In this level-six analysis, we note that the trend becomes more and more clear with each approximation, `A1` to `A6`. Why is this?

The trend represents the slowest part of the signal. In wavelet analysis terms, this corresponds to the greatest scale value. As the scale increases, the resolution decreases, producing a better estimate of the unknown trend.

Another way to think of this is in terms of frequency. Successive approximations possess progressively less high-frequency information. With the higher frequencies removed, what's left is the overall trend of the signal.

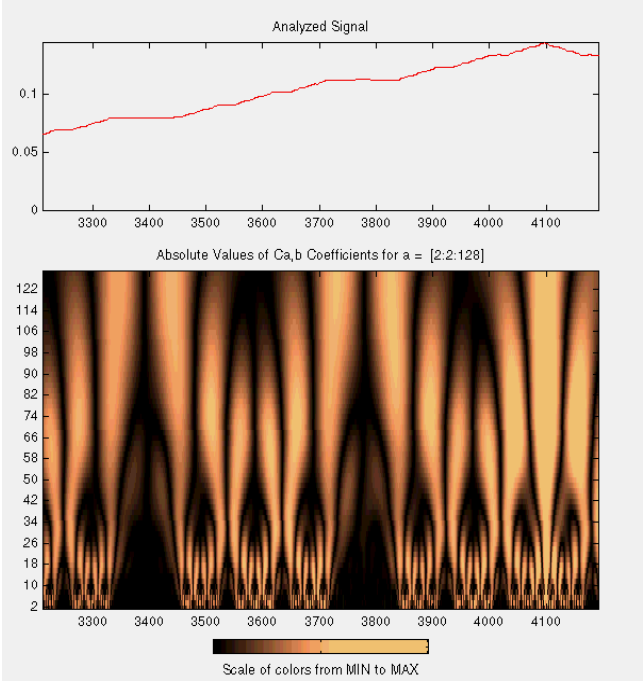
Discussion

Wavelet analysis is useful in revealing signal trends, a goal that is complementary to the one of revealing a signal hidden in noise. It's important to remember that the trend is the slowest part of the signal. If the signal itself includes sharp changes, then successive approximations look less and less similar to the original signal.

Consider the demo analysis **File⇒Demo Analysis⇒Step signal** (MAT-file `wstep.mat`). It is instructive to analyze this signal using the **Wavelet 1-D** tool and to see what happens to the successive approximations. Try it.

Detecting Self-Similarity

The purpose of this example is to show how analysis by wavelets can detect a self-similar, or fractal, signal. The signal here is the Koch curve — a synthetic signal that is built recursively.



Demo Analysis:
Koch curve
MAT-file:
`vonkoch.mat`
Wavelet:
`coif3`
Level:
Continuous, 2:2:128

This analysis was performed with the **Continuous Wavelet 1-D** graphical tool. A repeating pattern in the wavelet coefficients plot is characteristic of a signal that looks similar on many scales.

Wavelet Coefficients and Self-Similarity

From an intuitive point of view, the wavelet decomposition consists of calculating a “resemblance index” between the signal and the wavelet. If the index is large, the resemblance is strong, otherwise it is slight. The indices are the wavelet coefficients.

If a signal is similar to itself at different scales, then the “resemblance index,” or wavelet coefficients also will be similar at different scales. In the coefficients plot, which shows scale on the vertical axis, this self-similarity generates a characteristic pattern.

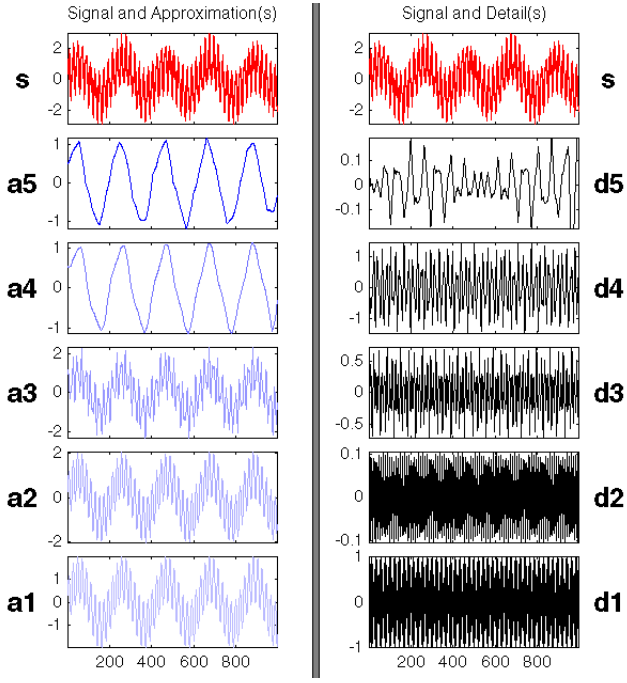
Discussion

The work of many authors and the trials that they have carried out suggest that wavelet decomposition is very well adapted to the study of the fractal properties of signals and images.

When the characteristics of a fractal evolve with time and become local, the signal is what is known as a *multifractal*. The wavelets then are an especially suitable tool for practical analysis and generation.

Identifying Pure Frequencies

The purpose of this example is to show how analysis by wavelets can effectively perform what is thought of as a Fourier-type function — that is, resolving a signal into constituent sinusoids of different frequencies. The signal is a sum of three pure sine waves.



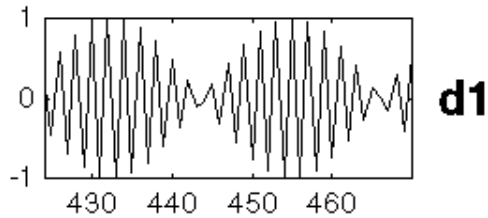
Demo Analysis:
Sum of sines
MAT-file:
`sumsin.mat`
Wavelet:
`db3`
Level:
5

Discussion

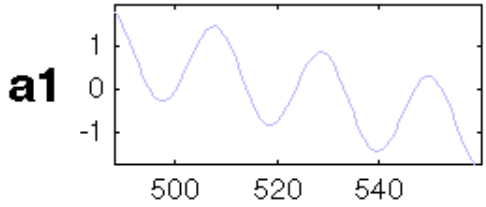
The signal is a sum of three sines: “slow,” “medium,” and “rapid,” which have periods (relative to the sampling period of 1) of 200, 20, and 2, respectively.

The “slow,” “medium,” and “rapid” sinusoids appear most clearly in approximation `A4`, detail `D4`, and detail `D1`, respectively. The slight differences that can be observed on the decompositions can be attributed to the sampling period.

Detail D1 contains primarily the signal components whose period is between 1 and 2 (i.e., the "rapid" sine), but this period is not visible at the scale which is used for the graph. Zooming in on detail D1 (see below) reveals that each "belly" is composed of 10 oscillations, and this can be used to estimate the period. We indeed find that it is close to 2.



The detail D3, and to an even greater extent, detail D4, contain the "medium" sine frequencies. We notice that there is a breakdown between approximations A3 and A4, from which the medium frequency information has been subtracted. We should therefore use approximations A1 to A3 to estimate the period of the "medium" sine. Zooming in on A1 reveals a period of around 20.



Now only the period of the "slow" sine remains to be determined. Examination of approximation A4 (see the figure on the previous page) shows the distance between two successive maximums to be 200.

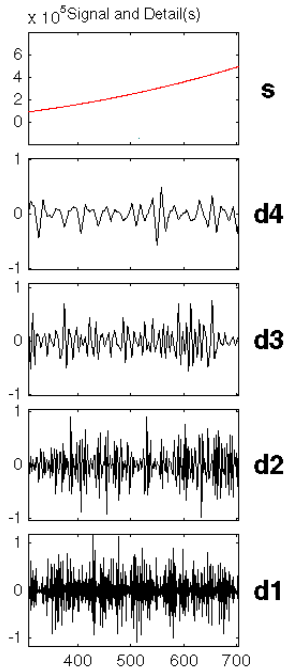
This “slow” sine still is visible in approximation A5, but were we to extend this analysis to further levels, we would find that it disappears from the approximation and move into the details at level 8.

Signal Component	Found in	Period	Frequency
“Slow sine”	Approximation A4	200	0.005
“Medium sine”	Detail D4	20	0.05
“Rapid sine”	Detail D1	2	0.5

In summation, we have used wavelet analysis to determine the frequencies of pure sinusoidal signal components. We were able to do this because the different frequencies predominate at a different scales, and each scale is taken account of by our analysis.

Suppressing Signals

The purpose of this example is to illustrate the property that causes the decomposition of a polynomial to produce null details, provided the number of "vanishing moments" of the wavelet (`N` for a Daubechies wavelet `dbN`) exceeds the degree of the polynomial. The signal here is a second-degree polynomial combined with a small amount of white noise.

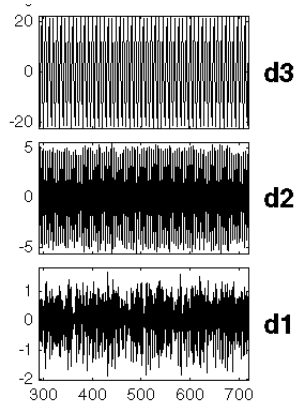


Demo Analysis:
Noisy polynomial
MAT-file:
`noispol.mat`
Wavelet:
db3
Level:
4

Note that only the noise comes through in the details. The peak-to-peak magnitude of the details is about 2, while the amplitude of the polynomial signal is on the order of 10^5.

The `db3` wavelet, which has three vanishing moments, was used for this analysis. Note that a wavelet of the Daubechies family with fewer vanishing moments would fail to suppress the polynomial signal. For more information, see that section, Daubechies Wavelets: dbN on page 6-63.

Here is what the first three details look like when we perform the same analysis with db2.



The peak-to-peak magnitudes of the details D1, D2, and D3 are 2, 10, and 40, respectively. These are much higher detail magnitudes than those obtained using db3.

Discussion

For the db2 analysis, the details for levels 2 to 4 show a periodic form that is very regular, and that increases with the level. This is explained by the fact that the detail for level j takes into account primarily the fluctuations of the polynomial function around its mean value on dyadic intervals that are 2^j long. The fluctuations are periodic and very large in relation to the details of the noise decomposition.

On the other hand, for the db3 analysis, we find the presence of white noise thus indicating that the polynomial does not come into play in any of the details. The wavelet suppresses the polynomial part and analyzes the noise.

Suppressing part of a signal allows us to highlight the remainder.

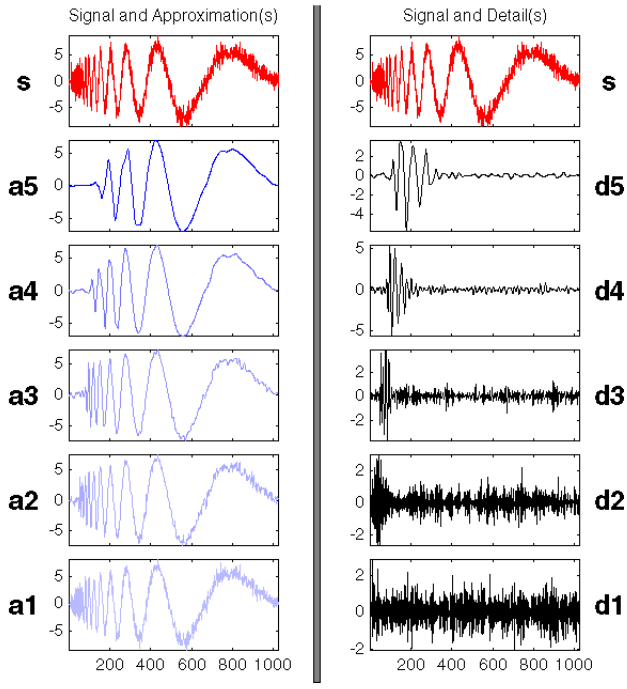
Vanishing Moments

The ability of a wavelet to suppress a polynomial depends on a crucial mathematical characteristic of the wavelet called its number of *vanishing moments*. A technical discussion of vanishing moments appears in Chapter 6, "Advanced Concepts." For the present discussion, it suffices to think of "moment" as an extension of "average." If a wavelet's average value is zero, then it has (at least) one vanishing moment.

More precisely, if the average value of $x^k \psi(x)$ is zero (where $\psi(x)$ is the wavelet function), for $k = 0, \ldots, n$, then the wavelet has $n + 1$ vanishing moments and polynomials of degree n are suppressed by this wavelet.

De-Noising Signals

The purpose of this example is to show how to de-noise a signal using wavelet analysis. This example also gives us an opportunity to demonstrate the automatic thresholding feature of the **Wavelet 1-D** graphical interface tool. The signal to be analyzed is a Doppler-shifted sinusoid with some added noise.

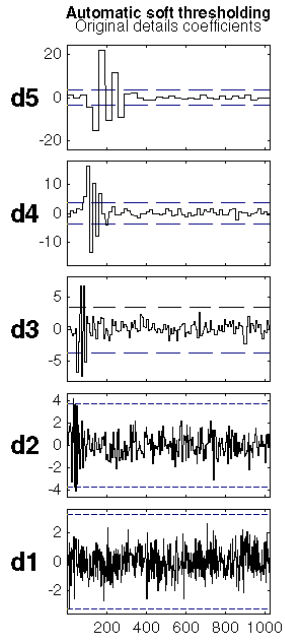


Demo Analysis:
Noisy Doppler
MAT-file:
`noisdopp.mat`
Wavelet:
`sym4`
Level:
5

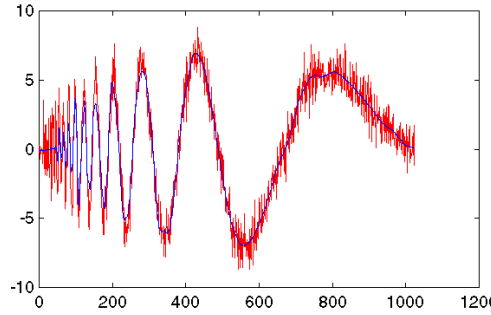
Discussion

We note that the highest frequencies appear at the start of the original signal. The successive approximations appear less and less noisy; however, they also lose progressively more high-frequency information. In approximation `A5`, for example, about the first 20% of the signal is truncated.

Click the **De-noise** button to bring up the **Wavelet 1-D De-noising** window. This window shows each detail along with its automatically set de-noising threshold.



Press the **De-noise** button. On the screen, the original and de-noised signals appear superimposed in red and yellow, respectively. In this figure, the de-noised signal is shown in blue for better contrast.



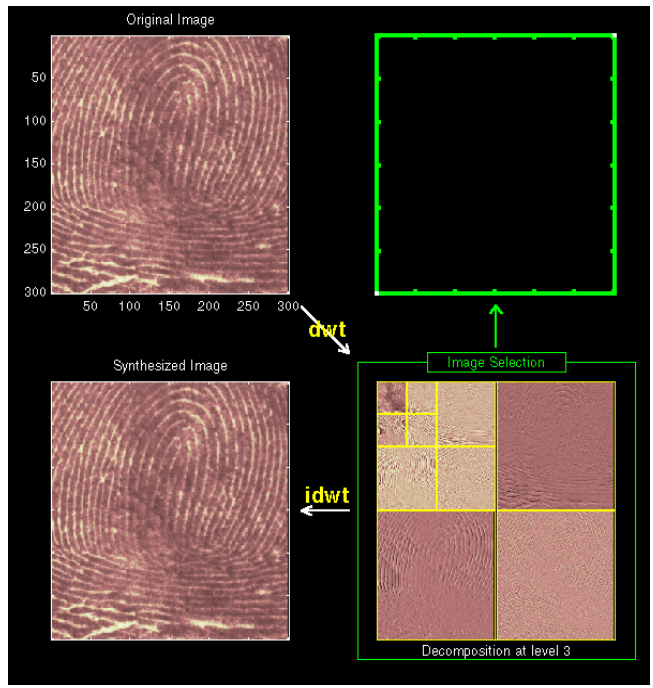
Note that the de-noised signal is flat initially. Some of the highest-frequency signal information was lost during the de-noising process, though less was lost here than in the higher-level approximations A4 and A5.

For this signal, wavelet packet analysis does a better job of removing the noise without compromising the high-frequency information. Explore on your own: try repeating this analysis using the **Wavelet Packet 1-D** tool. Select the menu item **File⇒Demo Analysis⇒noisdopp**.

Compressing Signals

The purpose of this example is to show how to compress an image using two-dimensional wavelet analysis. Compression is one of the most important applications of wavelets. The image to be compressed is a fingerprint.

For this example, open the **Wavelet 2-D** tool and select the menu item **File⇒Demo Analysis⇒at level 3, with haar —> finger**.

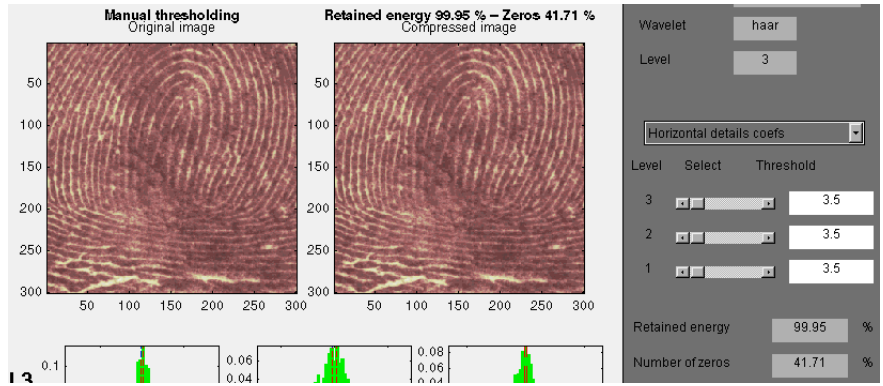


Demo Analysis:
finger
MAT-file:
`detfingr.mat`
Wavelet:
`haar`
Level:
3

The analysis appears in the **Wavelet 2-D** tool. Click the **Compress** button (located at the middle right) to bring up the **Wavelet 2-D Compression** window.

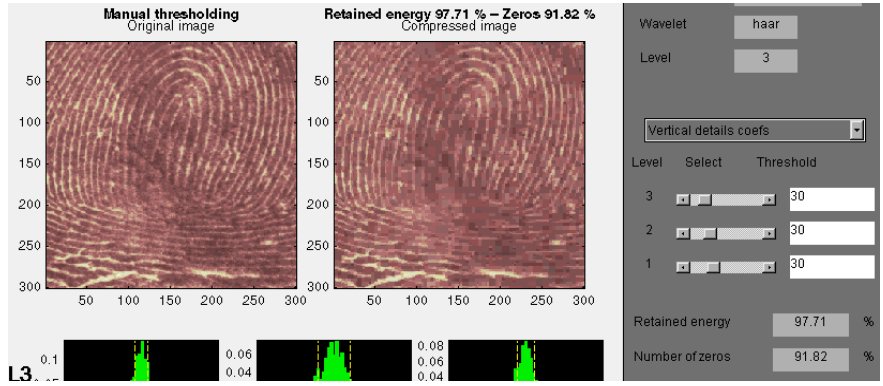
Discussion

The graphical tool provides an automatically-generated threshold, which for this example is 3.5. Values under the threshold are forced to zero, achieving about 42% zeros while retaining almost all (99.95%) the energy of the original image.



The automatic threshold usually achieves a reasonable balance between number of zeros and retained image energy. Depending on your data and your analysis criteria, you may find setting more aggressive thresholds achieves better results.

Here we've set the individual thresholds to around 30. This results in a compressed image consisting of 91.8% zeros with 97.7% retained energy.



4

Wavelets in Action: Examples and Case Studies

This chapter presents the different possibilities offered by wavelet decomposition in the form of examples you can work with on your own. Suggested areas for further exploration follow most examples, along with a summary of the topics addressed by that example.

This chapter also includes a case study that examines the practical uses of wavelet analysis in even greater detail, as well as a demonstration of the application of wavelets for fast multiplication of large matrices.

An extended discussion of many of the topics addressed by the examples can be found in Chapter 6, “Advanced Concepts.”

Illustrated Examples

Fourteen illustrated examples are included in this section, organized as shown:

Figure	Page	Description of the Signal	Signal Name	MAT-file
Figure 4-1:	page 4-9	A sum of sines: $s_1(t) = \sin(3t) + \sin(0.3t) + \sin(0.03t)$	$s_1(t)$	`sumsin`
Figure 4-2:	page 4-11	A frequency breakdown: $1 \le t \le 500, \quad s_2(t) = \sin(0.03t)$ $501 \le t \le 1000, \quad s_2(t) = \sin(0.3t)$	$s_2(t)$	`freqbrk`
Figure 4-3:	page 4-12	A uniform white noise: on the interval [–0.5 0.5]	$b_1(t)$	`whitnois`
Figure 4-4:	page 4-14	A colored AR(3) noise $b_2(t) = -1.5b_2(t-1) - 0.75b_2(t-2) - 0.125b_2(t-3) + b_1(t) + 0.5$	$b_2(t)$	`warma`
Figure 4-5:	page 4-17	A polynomial + a white noise: on the interval [1 1000] $s_3(t) = t^2 - t + 1 + b_1(t)$	$s_3(t)$	`noispol`
Figure 4-6:	page 4-19	A step signal: $1 \le t \le 500, \quad s_4(t) = 0$ $501 \le t \le 1000, \quad s_4(t) = 20$	$s_4(t)$	`wstep`

Figure	Page	Description of the Signal	Signal Name	MAT-file
Figure 4-7:	page 4-21	Two proximal discontinuities: $1 \le t \le 499, \quad s_5(t) = 3t$ $500 \le t \le 510, \quad s_5(t) = 1500$ $511 \le t, \quad s_5(t) = 3t - 30$	s_5(t)	`nearbrk`
Figure 4-8:	page 4-23	A second-derivative discontinuity: $t \in [-0.5 \quad 0.5] \subset R;$ $t < 0, f_3(t) = \exp(-4t^2)$ $t \ge 0, f_3(t) = \exp(-t^2)$ s_6 is f_3 sampled at 10^{-3}	s_6(t)	`scddvbrk`
Figure 4-9:	page 4-25	A ramp + a white noise: $1 \le t \le 499, \quad s_7(t) = \frac{3t}{500} + b_1(t)$ $500 \le t \le 1000, \quad s_7(t) = 3 + b_1(t)$	s_7(t)	`wnoislop`
Figure 4-10:	page 4-28	A ramp + a colored noise: $1 \le t \le 499, \quad s_8(t) = \frac{t}{500} + b_2(t)$ $500 \le t \le 1000, \quad s_8(t) = 1 + b_2(t)$	s_8(t)	`cnoislop`
Figure 4-11:	page 4-30	A sine + a white noise: $s_9(t) = \sin(0.03t) + b_1(t)$	s_9(t)	`noissin`
Figure 4-12:	page 4-21	A triangle + a sine: $1 \le t \le 500, \quad s_{10}(t) = \frac{t-1}{500} + \sin(0.3t)$ $501 \le t \le 1000, \quad s_{10}(t) = \frac{1000-t}{500} + \sin(0.3t)$	s_{10}(t)	`trsin`

Figure	Page	Description of the Signal	Signal Name	MAT-file
Figure 4-13:	page 4-34	A triangle + a sine + a noise: $501 \le t \le 1000,$ $s_{11}(t) = \frac{1000-t}{500} + \sin(0.3t) + b_1(t)$ $1 \le t \le 500,\ s_{11}(t) = \frac{t-1}{500} + \sin(0.3t) + b_1($	s_{11}(t)	`wntrsin`
Figure 4-14:	page 4-36	A real electricity consumption signal	—	`leleccum`

Please note that:

- All the decompositions use Daubechies wavelets.
- The examples show the signal, the approximations, and the details.

The examples include specific comments and feature distinct domains — for instance if the level of decomposition is 5:

- The left column contains the signal and the approximations A_5 to A_1.
- The right column contains the signal and the details D_5 to D_1.
- The approximation A_1 is located under A_2, A_2 under A_3 and so on. The same is true for the details.
- The abscissa axis represents the time. The unit for the ordinate axis for approximations and details is the same as that of the signal.
- When the approximations do not provide enough information, they are replaced by details obtained by changing wavelets.
- The examples include questions for you to think about:
  - What can be seen on the figure?
  - What additional questions can be studied?

Advice to the Reader

You should follow along and process these examples on your own, using either the graphical interface or the command line functions.

Use the graphical interface for immediate signal processing. To execute the analyses included in the figures:

1. To bring up the **Wavelet Toolbox Main Menu** type:
   ```
   » wavemenu
   ```
2. Select the **Wavelet 1-D** menu option to open the **Wavelet 1-D** tool.
3. From the **Wavelet 1-D** tool, choose the **File_Demo Analysis** menu option.
4. From the dialog box, select the sample analysis in question.

This triggers the execution of the examples.

When using the command line, follow the process illustrated in this M-file to conduct calculations:

```
% Load original 1-D signal.
    load sumsin; s = sumsin;

% Perform the decomposition of s at level 5, using coif3.
    w = 'coif3'
    [c,l] = wavedec(s,5,w);

% Reconstruct the approximation signals and detail signals at
% levels 1 to 5, using the wavelet decomposition structure [c,l].
for i = 1:5
    eval(['a(',int2str(i),',:) = wrcoef(''a'',c,l,w,i);']);
    eval(['d(',int2str(i),',:) = wrcoef(''d'',c,l,w,i);']);
end
```

Note: This loop replaces 10 separate `wrcoef` statements defining variables `a1` through `a5`, and `d1` through `d5`.

```
% Plots.
    t = 100:900;
    subplot(621); plot(t,s(t),'r');
    title('Orig. signal and approx. 1 to 5.');
    subplot(622); plot(t,s(t),'r');
    title('Orig. signal and details 1 to 5.');
    for i = 1:5,
        subplot(6,2,2*i+1); plot(t,a(i,t),'b');
        subplot(6,2,2*i+2); plot(t,d(i,t),'g');
    end
```

About Further Exploration

Tip 1: On all figures, visually check that for j = 0, 1, ..., $A_j = A_{j+1} + D_{j+1}$.

Tip 2: Don't forget to change wavelets. Test the shortest ones first.

Tip 3: Identify edge effects. They will create problems for a correct analysis. At present, there is no easy way to avoid them perfectly. You can use tools described in the section Dealing with Border Distortion on page 6-46 and see also the `dwtmode` function in Chapter 8, "Reference". They should eliminate or greatly reduce these effects.

Tip 4: As much as possible, conduct calculations manually to cross-check results with the values in the graphic representations. Manual calculations are possible with the `db1` wavelet.

For the sake of simplicity in the following examples, we use only the `haar` and `db` family wavelets, which are the most frequently used wavelets.

Example #1: A Sum of Sines

Analyzing wavelet: *db3*

Decomposition levels: *5*

The signal is composed of the sum of three sines: "slow", "medium" and "rapid" with regard to the sampling period equal to 1, the periods are approximately 200, 20 and 2 respectively. We should therefore see this later period in D_1, the medium sine in D_4, and the slow sine in A_4. The slight differences that can be observed on the decompositions can be attributed to the sampling period. The scale of the approximation charts is 2, 4 or 10 times larger than that of the details. D_1 contains primarily the components whose period is situated between 1 and 2 (i.e., the "rapid" sine), but this period is not visible at the scale which is used for the graph. Zooming in on D_1 reveals that each "belly" is composed of 10 oscillations, and can be used to estimate the period. We find that the period is close to 2. D_2 is very small. This is also seen in the approximations: the first two resemble one another, since $A_1 = A_2 + D_2$

The detail D_3, and to an even great extent D_4, contain the "medium" sine. We notice that there is a breakdown between approximations 3 and 4. Approximations A_1 to A_3 can be used to estimate the period of the medium sine. Now, only the "slow" sine, which appears in A_4 remains to be determined. The distance between two successive maximums is equal to 200, which is the period of the slow sine. This latter sine is still visible in A_5, but will disappear from the approximation and move into the details at level 8.

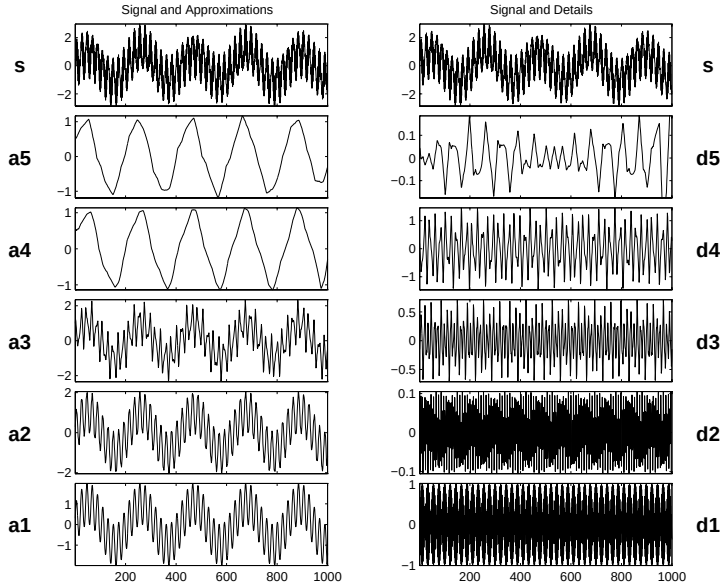


Figure 4-1: A Sum of Sines

	Example #1: A Sum of Sines
Addressed topics	• Detecting breakdown points • Detecting long-term evolution • Identifying pure frequencies • The effect of a wavelet on a sine • Details and approximations: a signal moves from an approximation to a detail • The level at which characteristics appear
Further exploration	• Compare with a Fourier analysis • Change the frequencies. Analyze other linear combinations.

Example #2: A Frequency Breakdown

Analyzing wavelet: *db5*

Decomposition levels: *5*

The signal is formed of a "slow" sine and a "medium" sine, on either side of 500. These two sines are not connected in a continuous manner: D_1 and D_2 can be used to detect this discontinuity. It is localized very precisely: only a small domain around 500 contains large details. This is because the rupture contains the high frequency part; the frequencies in the rest of the signal are not as high. It should be noted that if we are interested only in identifying the discontinuity, `db1` is more useful than `db5`.

D_3 and D_4 contain the "medium" sine as in the previous analysis. The "slow" sine appears clearly alone in A_5. It is more regular than in the s_1 analysis, since `db5` is more regular than `db3`. If the same signal had been analyzed by the Fourier transform, we would not have been able to detect the instant corresponding to the signal's frequency change, whereas it is clearly observable here.

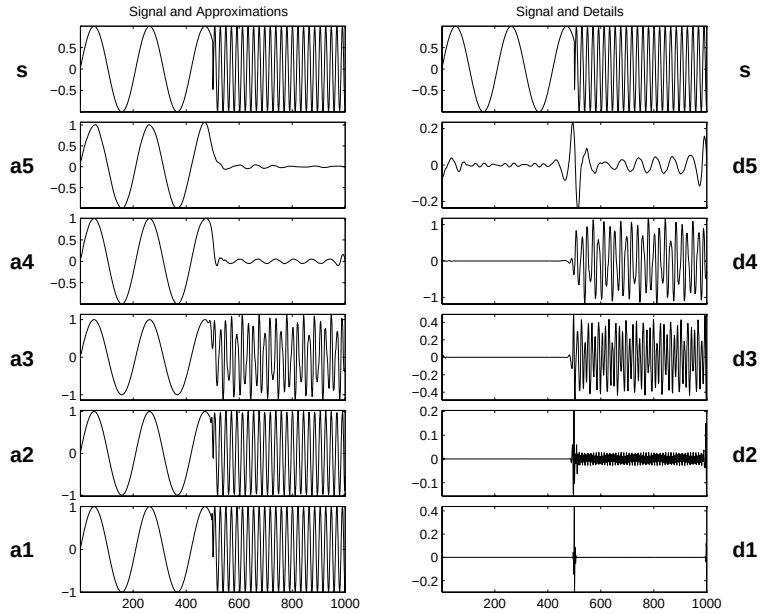


Figure 4-2: A Frequency Breakdown

	Example #2: A Frequency Breakdown
Addressed topics	• Suppressing signals • Detecting long-term evolution
Further exploration	• Compare to the signal s_1 • On a longer signal, have the slow sinusoid moved into the details • Compare with a Fourier analysis • Compare with a windowed Fourier analysis

Example #3: Uniform White Noise

Analyzing wavelet: *db3*

Decomposition levels: *5*

At all levels we encounter noise-type signals, which are clearly irregular. This is due to the fact that all the frequencies carry the same energy. The variances however, decrease regularly between one level and the next as can be seen reading the detail chart (on the right) and the approximations (on the left). The variance decreases two-fold between one level and the next, i.e. variance(D_j) = variance(D_{j-1})/2. Lastly, it should be noted that the details and approximations are not white noises, and that these signals are increasingly interdependent as the resolution decreases. On the other hand, the wavelet coefficients are random, non-correlated variables. This property is not evident on the reconstructed signals shown here, but it can be guessed at from the representation of the coefficients.

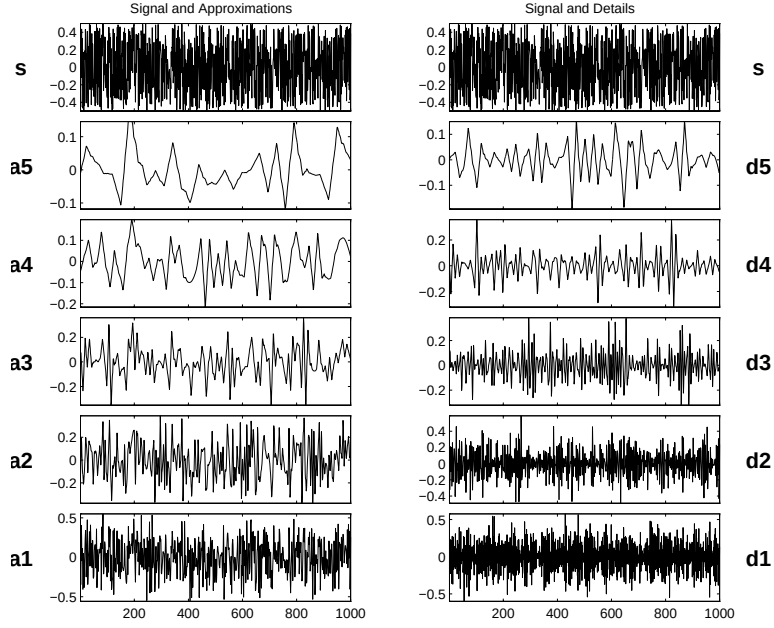


Figure 4-3: Uniform White Noise

	Example #3: Uniform White Noise
Addressed topics	• Processing noise • The shapes of the decomposition values • The evolution of these shapes according to level: the correlation increases, the variance decreases • Compare the frequencies included in the details with those in the approximations
Further exploration	• Study the values of the coefficients and their distribution • On the continuous analysis, identify the chaotic aspect of the colors • Replace the uniform white noise by a Gaussian white noise or other noise.

Example #4: Colored AR(3) Noise

Analyzing wavelet: *db3*

Decomposition levels: *5*

Note: AR(3) means AutoRegressive model of order 3.

This figure can be examined in view of the previous figure, since we are confronted here with a non-white noise whose spectrum is mainly at the higher frequencies. It is therefore found primarily in D_1, which contains the major portion of the signal. In this situation, which is commonly encountered in practice, the effects of the noise on the analysis decrease considerably more rapidly than in the case of white noise. In A_3, A_4 and A_5, we encounter the same scheme as that in the analysis of b_1 (see the table on page 4-3), the noise from which b_2 is built using linear filtering.

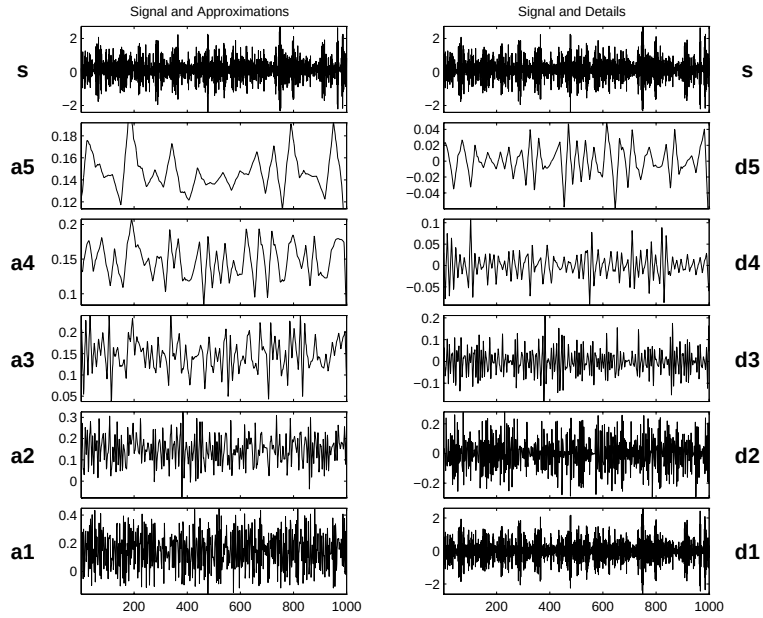


Figure 4-4: Colored AR(3) Noise

	Example #4: Colored AR(3) Noise
Addressed topics	• Processing noise • The relative importance of different details • The comparative importance of D_1 and A_1. • Compare the detail frequencies with those in the approximations.
Further exploration	• Compare approximations A_3, A_4, and A_5 with those shown in Figure 4-3. • Replace AR(3) with an ARMA (AutoRegressive Moving Average) model noise. For instance: $b_3(t) = -1.5b_3(t-1) - 0.75b_3(t-2) - 0.125b_3(t-3) + b_1(t) - 0.7b_1(t-1)$ • Study an ARIMA (Integrated ARMA) model noise. For instance: $b_4(t) = b_4(t-1) + b_3(t)$ • Check that each detail can be modeled by an ARMA process.

Example #5: Polynomial + White Noise

Analyzing wavelets: *db2* and *db3*

Decomposition levels: *4*

The purpose of this analysis is to illustrate the property which causes the decomposition by `dbN` of a `p`-degree polynomial to produce null details as long as `N > p`. In this case, `p=2` and we examine the first four levels of details for two values of N: one is too small, `N=2` on the left, and the other is sufficient, `N=3` on the right. The approximations are left out since they differ very little from the signal itself.

For `db2` (on the left), we obtain the decomposition of $t^2 + b_1(t)$, since the $-t + 1$ part of the signal is suppressed by the wavelet. In fact, with the exception of level 1, where noise-generated irregularities can be seen, the details for levels 2 to 4 show a periodic form that is very regular, and which increases with the level. This is explained by the fact that the detail for level j takes into account primarily the fluctuations of the function around its mean value on dyadic intervals that are long. The fluctuations are periodic and very large in relation to the details of the noise decomposition.

On the other hand, for `db3` (on the right) we again find the presence of white noise thus indicating that the polynomial does not come into play in any of the details. The wavelet suppresses the polynomial part and analyzes the noise.

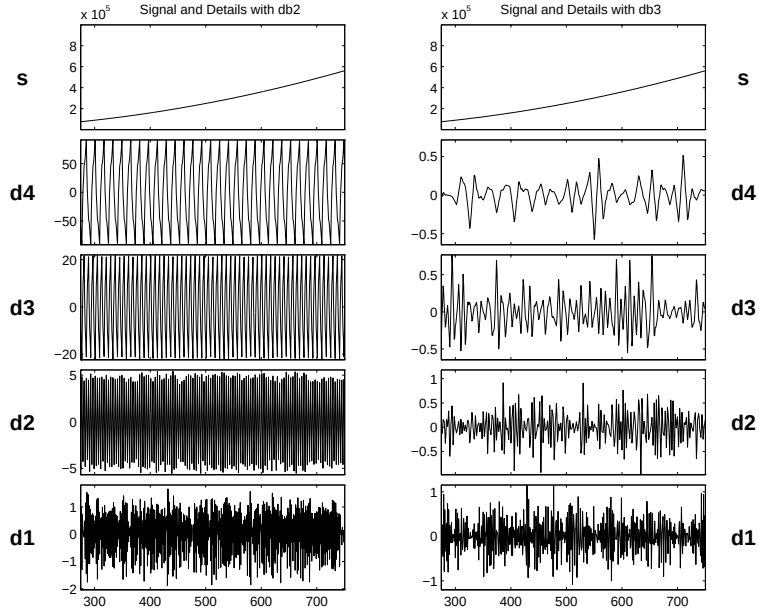


Figure 4-5: Polynomial + White Noise

	Example #5: Polynomial + White Noise
Addressed topics	• Suppressing signals • Compare the results of the processing for the following wavelets: the short db2 and the longer db3. • Explain the regularity that is visible in D_3 and D_4 in the analysis by db2.

Example #6: A Step Signal

Analyzing wavelet: *db2*

Decomposition levels: *5*

In this case we are faced with the simplest example of a rupture (i.e., a step). The time instant when the jump occurs is equal to 500. The break is detected at all levels, but it is obviously detected with greater precision in the higher resolutions (levels 1 and 2) than in the lower ones (levels 4 and 5). It is very precisely localized at level 1, where only a very small zone around the jump time can be seen.

It should be noted that the reconstructed details are primarily composed of the basic wavelet represented in the initial time.

What is more, the rupture is all the more precisely localized when the wavelet corresponds to a short filter.

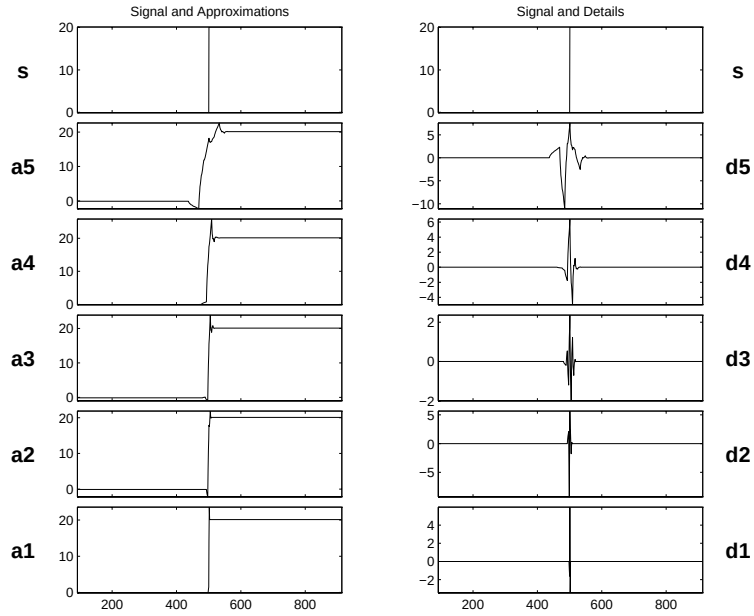


Figure 4-6: A Step Signal

	Example #6: A Step Signal
Addressed topics	• Detecting breakdown points • Suppressing signals • Detecting long-term evolution • Identify the range width of the variations of details and approximations.
Further exploration	• Use the coefficients of the FIR filter associated with the wavelet to check the values of D_1 • Replace the step by an impulse • Add noise to the signal and repeat the analysis

Example #7: Two Proximal Discontinuities

Analyzing wavelet: *db2* and *db7*

Decomposition levels: *5*

The signal is formed of two straight lines with identical slopes, extending across a very short plateau. On the initial signal, the plateau is in fact barely visible to the naked eye. Two analyses are thus carried out, one on a well localized wavelet with the short filter (`db2`) on the left and the other on a wavelet having a longer filter (`db7`) on the right. In both analyses, the plateau is detected clearly; with the exception of a fairly limited domain, D_1 is equal to zero. The regularity of the signal in the plateau however is clearly distinguished for `db2` (for which plateau beginning and end time are distinguished), whereas for `db7` both discontinuities are fused and only the entire plateau can said to be "visible." This example suggests that the selected wavelets should be associated with short filters to distinguish proximal discontinuities of the first derivative. A look at the other detail levels again shows the lack of precision when detecting at low resolutions. The wavelet filters the straight line and analyzes the discontinuities.

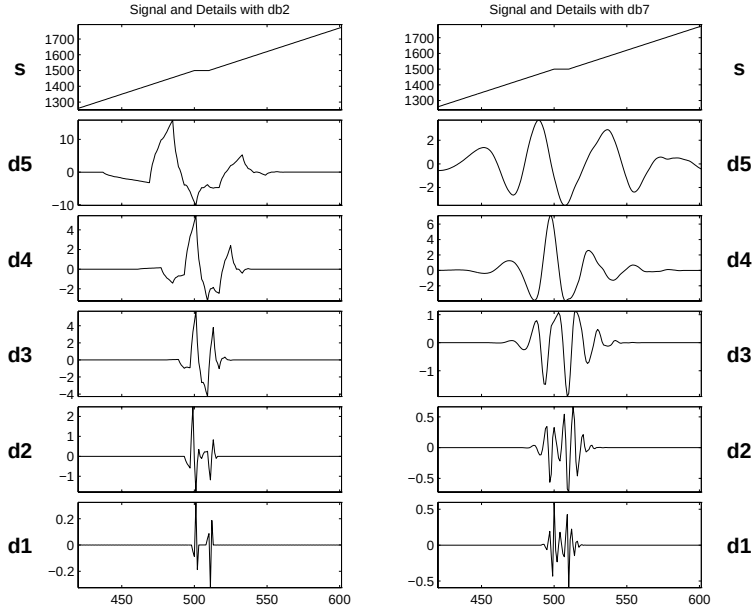


Figure 4-7: Two Proximal Discontinuities

	Example #7: Two Proximal Discontinuities
Addressed topics	• Detecting breakdown points • Move the discontinuities closer together and further apart
Further exploration	• Add noise to the signal until the rupture is no longer visible • Try using other wavelets, *haar* for instance.

Example #8: A Second-Derivative Discontinuity

Analyzing wavelet: *db1* and *db4*

Decomposition levels: *2*

This figure shows that the regularity can be an important criterion in selecting a wavelet. The basic function is composed of two exponentials that are connected at 0, and the analyzed signal is the sampling of the continuous function with increments of 10^{-3}. The sampled signal is analyzed using two different wavelets: Haar which is insufficiently regular, on the left, and `db4` which is sufficiently regular, on the right.

On the left we notice that the singularity has not been detected in the extent that the details are equal to 0 at 0. The black areas correspond to very rapid oscillations of the details. These values are equal to the difference between the function and an approximation using a constant function. Close to 0, the slow decrease of the details absolute values followed by a slow increase is due to the fact that the function derivative is zero and continuous at 0. The value of the details is very small (close to 10^{-3} for Haar and 10^{-4} for `db4`) since the signal is very smooth and does not contain any high frequency. This value is even smaller for `db4`, since the wavelet is more regular than `db1`.

However with `db4` (on the right), the discontinuity is well detected: the details are high only close to 0 and are 0 everywhere else. This is the only element that can be derived from the analysis. In this case, as a conclusion, we notice that the selected wavelet must be sufficiently regular, which thus implies a longer filter impulse response in order to detect the singularity.

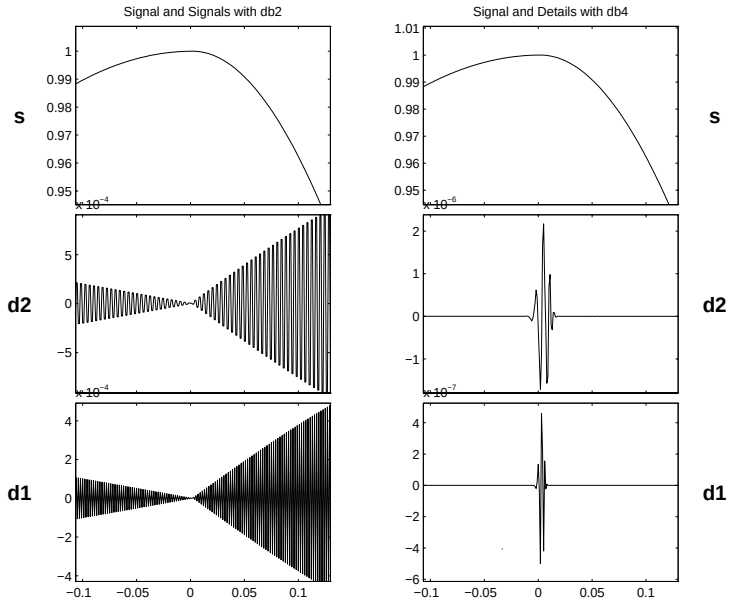


Figure 4-8: A Second-Derivative Discontinuity

	Example #8: A Second-Derivative Discontinuity
Addressed topics	• Detecting breakdown points • Suppressing signals • Identifying a difficult discontinuity • Carefully selecting a wavelet in order to reveal an effect
Further exploration	• Calculate the detail values for the Haar wavelet • Beware of parasitic effects: rapid detail fluctuations may be artifacts • Add noise to the signal until the rupture is no longer visible

Example #9: A Ramp + White Noise

Analyzing wavelet: *db3*

Decomposition levels: *6*

The signal is built from a trend plus noise. The trend is a slow linear rise from 0 to 3, up to t=500 and becoming constant afterwards. The noise is a uniform zero mean white noise, varying between -0.5 and 0.5 (see the analyzed signal b_1).

In the charts on the right, we again find the decomposition of noise in the details. In the charts on the left, the approximations form increasingly precise estimates of the ramp with less and less noise. These approximations are quite acceptable from level 3, and the ramp is well reconstructed at level 6.

We can therefore separate the ramp from the noise. Although the noise affects all scales, its effect decreases sufficiently quickly for the low-resolution approximations to restore the ramp. It should also be noted that the breakdown point of the ramp is shown with good precision. This is due to the fact that the ramp is recovered at too low a resolution.

The uniform noise indicates that the ramp might be best estimated using half sums for the higher and lower portions of the signal. This approach is not applicable for other noises.

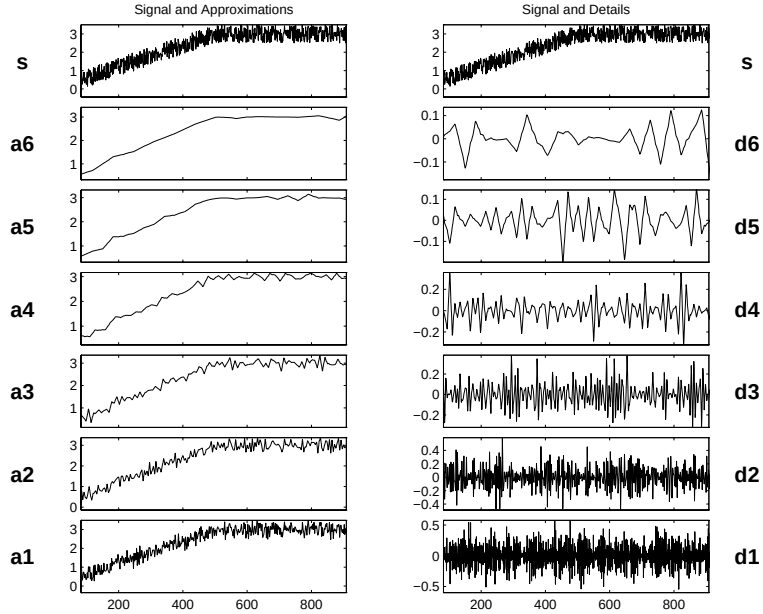


Figure 4-9: A Ramp + White Noise

	Example #9: A Ramp + White Noise
Addressed topics	• Detecting breakdown points • Processing noise • Detecting long-term evolution • Splitting signal components • Identifying noises and approximations • Compare with the white noise $b_1(t)$ shown in Figure 4-3. • Identify the number of levels needed to suppress the noise almost entirely.
Further exploration	• Change the noise

Example #10: A Ramp + Colored Noise

Analyzing wavelet: *db3*

Decomposition levels: *6*

The signal is built in the same manner as the previous example, using a trend plus a noise. The trend is a slow linear increase from 0 to 1, up to t=500. Beyond this time, the value remains constant. The noise is a zero mean AR(3) noise, varying between -3 and 3 (see the analysed signal b_2). The scale of the noise is indeed six times greater than that of the ramp. At first glance, the situation seems a little bit less favorable than in the previous example, in terms of the separation between the ramp and the noise. This is actually a misconception, since the two signal components are more precisely separated in frequency.

The charts on the right show the detail decomposition of the colored noise. The charts on the left show a decomposition that resembles the one in the previous analysis. Starting at level 3, the curves provide satisfactory approximations of the ramp.

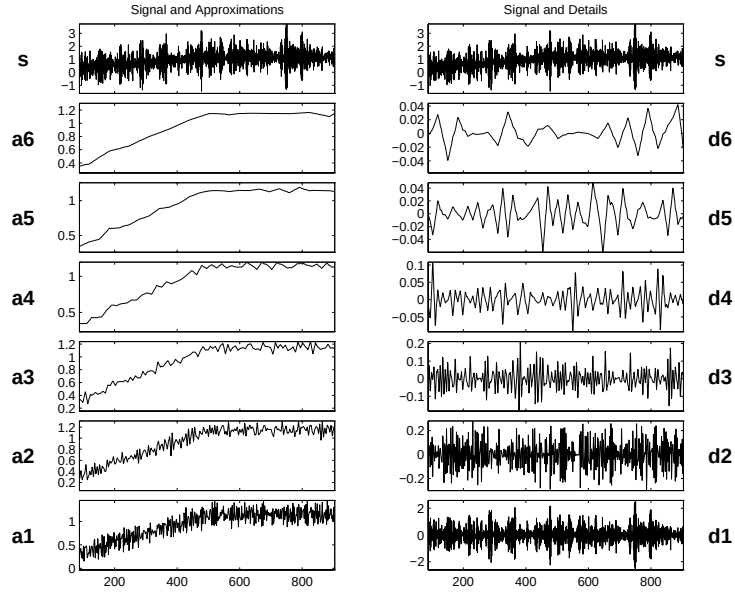


Figure 4-10: A Ramp + Colored Noise

	Example #10: A Ramp + Colored Noise
Addressed topics	• Detecting breakdown points • Processing noise • Detecting long-term evolution • Splitting signal components • Compare with the $s_7(t)$ signal shown in Figure 4-9. • Identify the number of levels needed to suppress the noise almost entirely
Further exploration	• Identify the noise characteristics. Use the coefficients and the command line mode.

Example #11: A Sine + White Noise

Analyzing wavelet: *db5*

Decomposition levels: *5*

The signal is formed of the sum of two previously analyzed signals: the slow sine with a period close to 200 and the uniform white noise b_1. This example is an illustration of the linear property of decompositions: the analysis of the sum of two signals is equal to the sum of analyses.

The details correspond to those obtained during the decomposition of the white noise.

The sine is found in the approximation A_5. This is a high enough level for the effect of the noise to be negligible in relation to the amplitude of the sine.

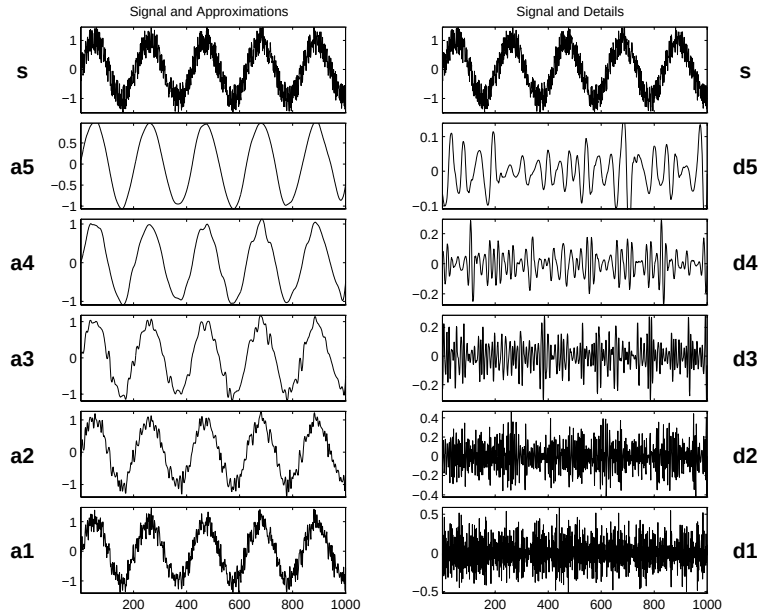


Figure 4-11: A Sine + White Noise

	Example #11: A Sine + White Noise
Addressed topics	• Processing noise • Detecting long-term evolution • Splitting signal components • Identifying the frequency of a sine
Further exploration	• Identify the noise characteristics. Use the coefficients and the command line mode.

Example #12: A Triangle + A Sine

Analyzing wavelet: *db5*

Decomposition levels: *6*

The signal is the sum of a sine having a period of approximately 20 and of a "triangle".

D_1 and D_2 are very small. This suggests that the signal contains no components with periods that are short in relation to the sampling period.

D_3 and especially D_4 can be attributed to the sine. The jump of the sine from A_3 to D_4 is clearly visible.

The details for the higher levels D_5 and D_6 are small, especially D_5. D_6 exhibits some edge effects.

A_6 contains the triangle which includes only low frequencies.

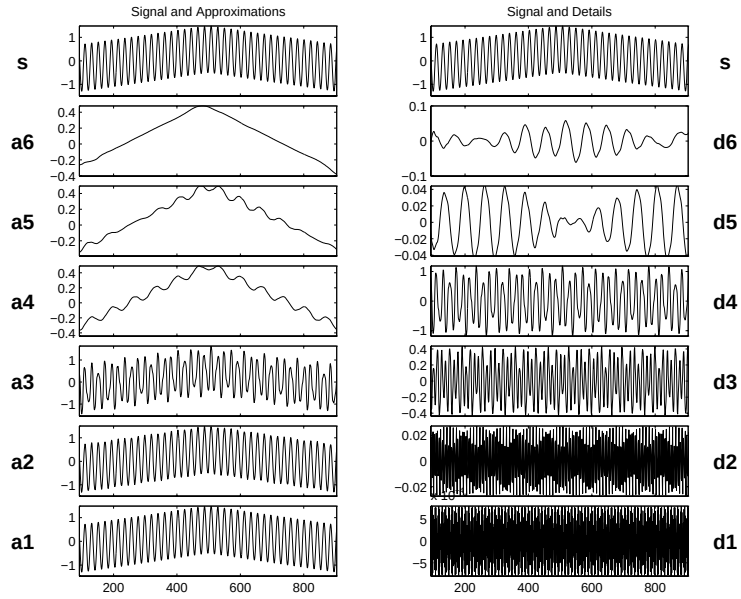


Figure 4-12: A Triangle + A Sine

	Example #12: A Triangle + A Sine
Addressed topics	• Detecting long-term evolution • Splitting signal components • Identifying the frequency of a sine
Further exploration	• Try using sinusoids whose period is a power of 2.

Example #13: A Triangle + A Sine + Noise

Noise Analyzing wavelet: *db5*

Decomposition levels: *7*

The signal examined here is the same as the previous signal plus a uniform white noise divided by 3. The analysis can therefore be compared to the previous analysis. All differences are due to the presence of the noise.

D_1 and D_2 are due to the noise.

D_3 and especially D_4 are due to the sine.

The higher-level details are increasingly low, and originate in the noise.

A_7 contains a triangle, although it is not as well reconstructed as in the previous example.

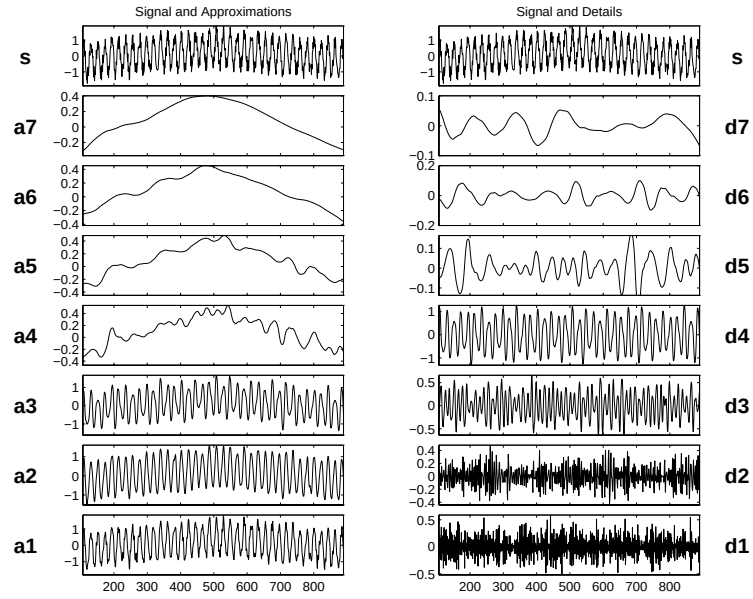


Figure 4-13: A Triangle + A Sine + Noise

	Example #13: A Triangle + A Sine + Noise
Addressed topics	• Detecting long-term evolution • Splitting signal components
Further exploration	• Increase the amplitude of the noise • Replace the triangle by a polynomial • Replace the white noise by an ARMA noise

Example #14: A Real Electricity Consumption Signal

Analyzing wavelet: *db3*

Decomposition levels: *5*

The series presents a peak in the center, followed by two drops, a shallow drop, and then a considerably weaker peak.

The details for levels 1 and 2 are of the same order of magnitude and give a good expression of the local irregularities caused by the noise. The detail for level 3 presents high values in the beginning and at the end of the main "peak," thus allowing us to locate the corresponding drops. The detail D_4 shows coarser morphological aspects for the series i.e., three successive peaks. This fits the shape of the curve remarkably well, and includes the essential signal components for periods of less than 32 time-units. The approximations show this effect clearly: A_1 and A_2 bear a strong resemblance; A_3 forms a reasonably accurate approximation of the original signal. A look at A_4, however, shows that a considerable amount of information has been lost.

In this case, as a conclusion, the multi-scale aspect is the most interesting and the most significant feature: the essential components of the electrical signal used to complete the description at 32 time-units (homogeneous to A_5) are the components with a period between 8 to 16 time-units.

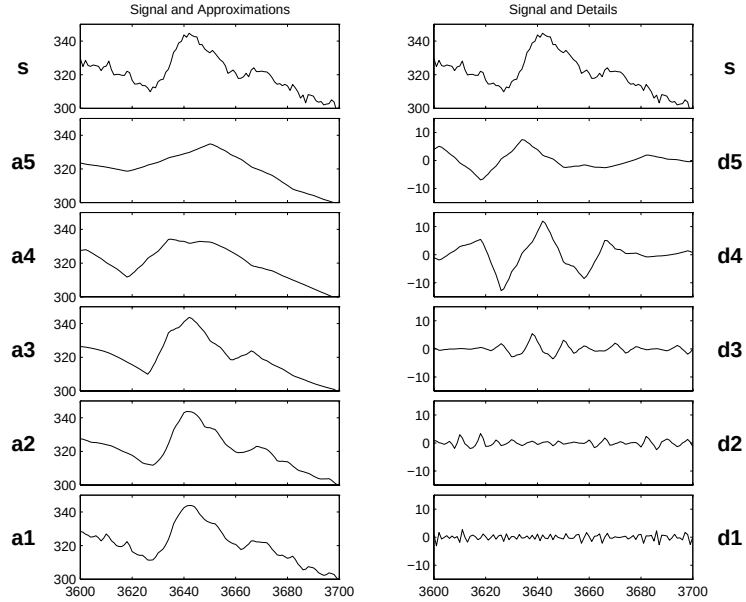


Figure 4-14: A Real Electricity Consumption Signal

	Example #14: A Real Electricity Consumption Signal
Addressed topics	• Detecting long-term evolution • Splitting signal components • Detecting breakdown points • Multiscale analysis
Further exploration	• Try the same analysis on various sections of the signal. Focus on a range other than the `[3600:3700]` shown here.

This signal is explored in much greater detail in A Case Study: An Electrical Signal on page 4-37.

A Case Study: An Electrical Signal

The goal of this section is to provide a statistical description of an electrical load consumption using the wavelet decompositions as a multiscale analysis.

Two problems are addressed. They both deal with signal extraction from the load curve corrupted by noise:

1. What information is contained in the signal, and what pieces of information are useful?

2. Are there various kinds of noises, and can they be distinguished from one another?

The context of the study is the forecast of the electrical load. Currently, short-term forecasts are based on the data sampled over 30 minutes. After eliminating certain components linked to weather conditions, calendar effects, outliers and known external actions, a SARIMA parametric model is developed. The model delivers forecasts from half-an-hour to two days. The quality of the forecasts is very high at least for 90% of all days, but the method fails when working with the data sampled over 1 minute.

Data and the External Information

The data consist of measurement of a complex, highly-aggregated plant: the electrical load consumption, sampled minute by minute, over a 5-week period. This time series of 50,400 points is partly plotted at the top of the Figure 4-17.

External information is given by electrical engineers, and additional indications can be found in several papers. This information, used to define reference situations for the purpose of comparison, includes these points:

- The load curve is the aggregation of hundreds of sensors measurements, thus generating measurement errors.
- Roughly speaking, the consumption is accounted for by industry for 50% and by individual consumers for the other half. The component of the load curve produced by industry has a rather regular profile and exhibits low-frequency changes. On the other hand, the consumption of individual consumers may be highly irregular, leading to high-frequency components.

- There are more than 10 millions individual consumers.
- The fundamental periods are the weekly-daily cycles, linked to economic rhythms.
- Daily consumption patterns also change according to rate changes at different times (e.g. relay-switched water heaters to benefit from special night rates).
- Missing data have been replaced.
- Outliers have not been corrected.
- For the observations 2400 to 3400, the measurement errors are unusually high, due to sensors failures.

From a methodological point of view, the wavelet techniques provide a multiscale analysis of the signal as a sum of orthogonal signals corresponding to different time scales, allowing a kind of time-scale analysis.

Because of the absence of a model for the one minute data, the description strategy proceeds essentially by successive uses of various comparative methods applied to signals obtained by the wavelet decomposition.

Without modeling, it is impossible to define a signal or a noise effect. Nevertheless, we say that any repetitive pattern is due to signal and is meaningful.

Finally it is known that two kinds of noise corrupt the signal: sensors errors and the state noise.

We shall not report here the complete analysis which is included in a paper. Instead, we illustrate the contribution of wavelet transforms to the local description of time series. We choose two small samples: one taken at midday, and the other at the end of the night.

In the first period the signal structure is complex, in the second one, much simpler.The midday period has a complicated structure because the intensity of the electricity consumers activity is high and it presents very large changes. At the end of the night the activity is low and changes slowly.

For the local analysis, the decomposition is taken up to the level j = 5, because $2^5 = 32$ is very close to half an hour. We are then able to study the components of the signal for which the period is less than half an hour.

The analyzing wavelet used here is `db3`.

The results are described similarly for the two periods.

Analysis of the Midday Period

This signal (see Figure 4-14) is also analyzed in Example #14: A Real Electricity Consumption Signal on page 4-35 more crudely.

The shape is a middle mode between 00h30 pm and 01h00 pm, preceded and followed by a hollow off-peak, and next a second smoother mode at 01h15 pm. The approximation A_5 corresponding to the time scale of 32 minutes, is a very crude approximation, particularly for the central mode: there is a peak time lag and an underestimation of the maximum value. So at this level the most essential information is missing. We have to look at lower scales (4 for instance).

Let us examine the corresponding details.

The details D_1 and D_2 have small values and may be considered as local short-period discrepancies caused by the high frequency components of sensor and state noises. In this bandpass, these noises are essentially due to measurement errors and fast variations of the signal induced by millions of state changes of personal electrical appliances.

The detail D_3 exhibits high values at times corresponding to the start and the end of the original middle mode. It allows time localization of the local minima.

The detail D_4 contains the main patterns: three successive modes. It is remarkably close to the shape of the curve. The ratio of the values of this level to the other levels is equal to 5. The detail D_5 does not bear much information. So the contribution of the level 4 is the highest one, both in qualitative and quantitative aspects. It captures the shape of the curve in the concerned period.

In conclusion, with respect to the approximation A_5, the detail D_4 is the main additional correction: the components of period 8 to 16 minutes contain the crucial dynamics.

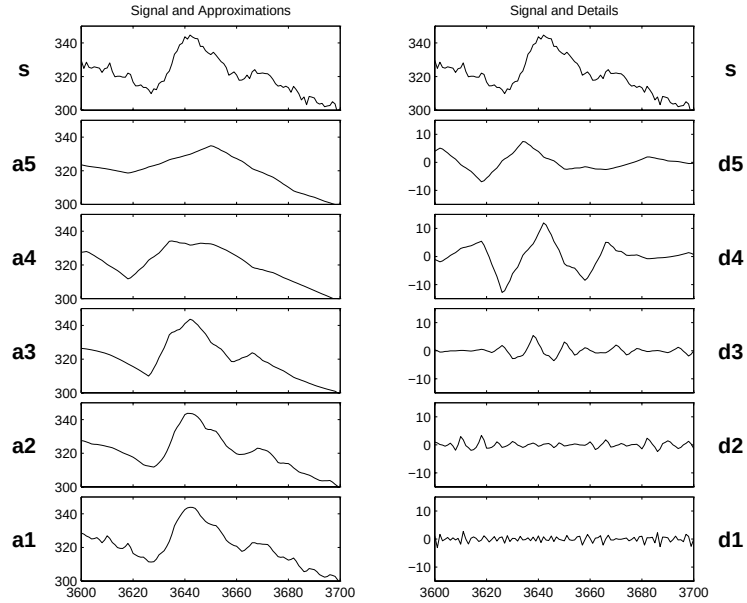


Figure 4-15: Analysis of the Midday Period

Analysis of the End of the Night Period

See Figure 4-16.

The shape of the curve during the end of the night is a slow descent globally smooth but locally highly irregular. One can hardly distinguish two successive local extrema in the vicinity of time t=1600 and t=1625. The approximation A_5 is quite good except at these two modes.

The accuracy of the approximation can be explained by the fact that there remains only a low frequency signal corrupted by noises. The massive and simultaneous changes of personal electric appliances are absent.

The details D_1, D_2, and D_3 show the kind of variation and have, roughly speaking, similar shape and mean value. They contain the local short period irregularities caused by noises and the inspection of D_2 and D_3 allows one to detect the local minimum around t=1625.

The details D_4 and D_5 exhibit the slope changes of the regular part of the signal and A_4 and A_5 are piecewise linear.

In conclusion, none of the time scales brings a significant contribution, sufficiently different from the noise level, and no additional correction is needed. The retained approximation is A_4 or A_5.

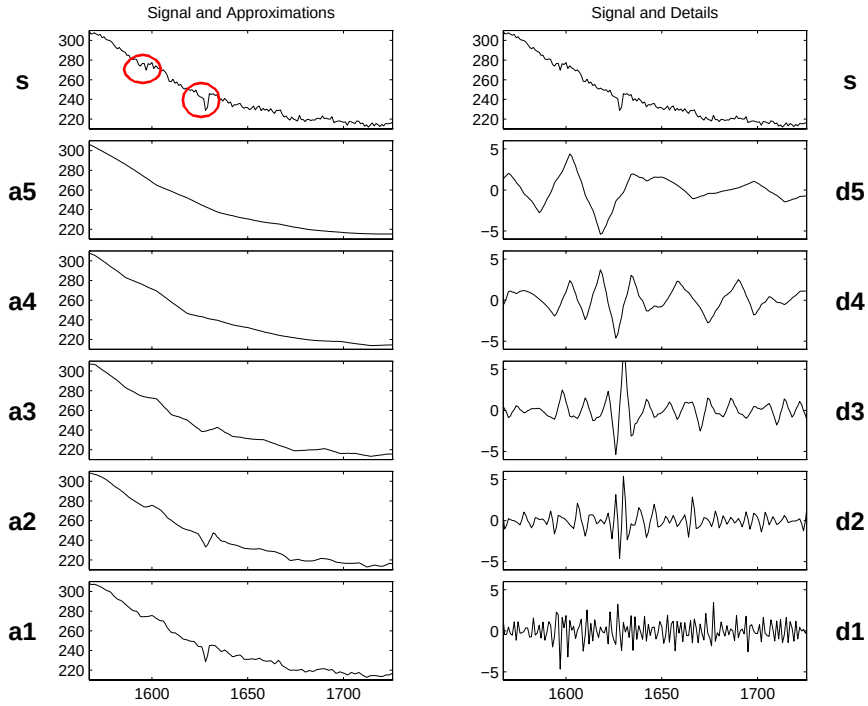


Figure 4-16: Analysis of the End of the Night Period

All the figures in this paragraph have been generated using the Graphical User Interface tools, but the user can also process the analysis using the command line mode. The following example gives some indications and corresponds to a command line equivalent of producing Figure 4-17.

```
% Load the original 1-D signal, decompose, reconstruct details in
% original time and plot.
% load the signal.
load leleccum; s = leleccum;

% Decompose the signal s at level 5 using the wavelet w.
w = 'db3'; [c,l] = wavedec(s,5,w);

% Reconstruct the details using the coefficients.
for i = 1:5
   eval(['d(',int2str(i),',:) = wrcoef(''d'',c,l,w,i);']);
end
```

Note: This loop replaces 5 separate `wrcoef` statements defining variables D_1 through D_5.

```
% Avoid edge effects by suppressing edge values and plot.
tt = 1+100:length(s)-100;
subplot(611); plot(tt,s(tt),'r');
title('Electrical Signal and Details');
for i = 1:5, subplot(6,1,i+1); plot(tt,d(i,tt),'g'); end
```

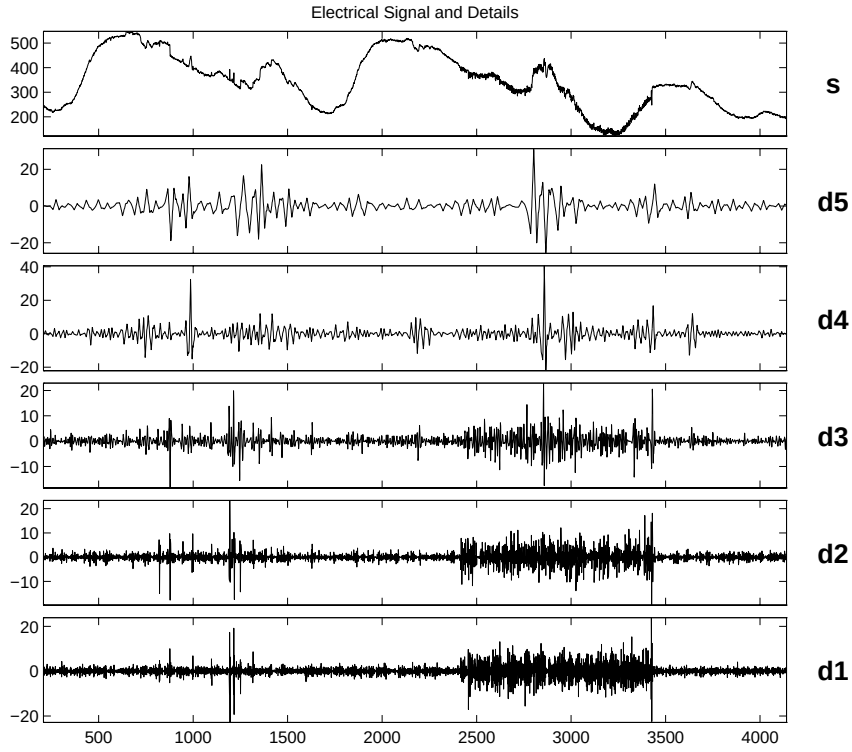


Figure 4-17: Decomposition of Three-Day Electrical Signal at Level 5 Using db3

Suggestions for Further Analysis

Let us now make some suggestions for possible further analysis starting from the details of the decomposition at level 5 of three days (see Figure 4-17).

Identify the Sensor Failure

Focus on the wavelet decomposition and try to identify the sensor failure directly on the details D_1, D_2 and D_3 and not the other ones. Try to identify the other part of the noise.

Indication: see Figure 4-18.

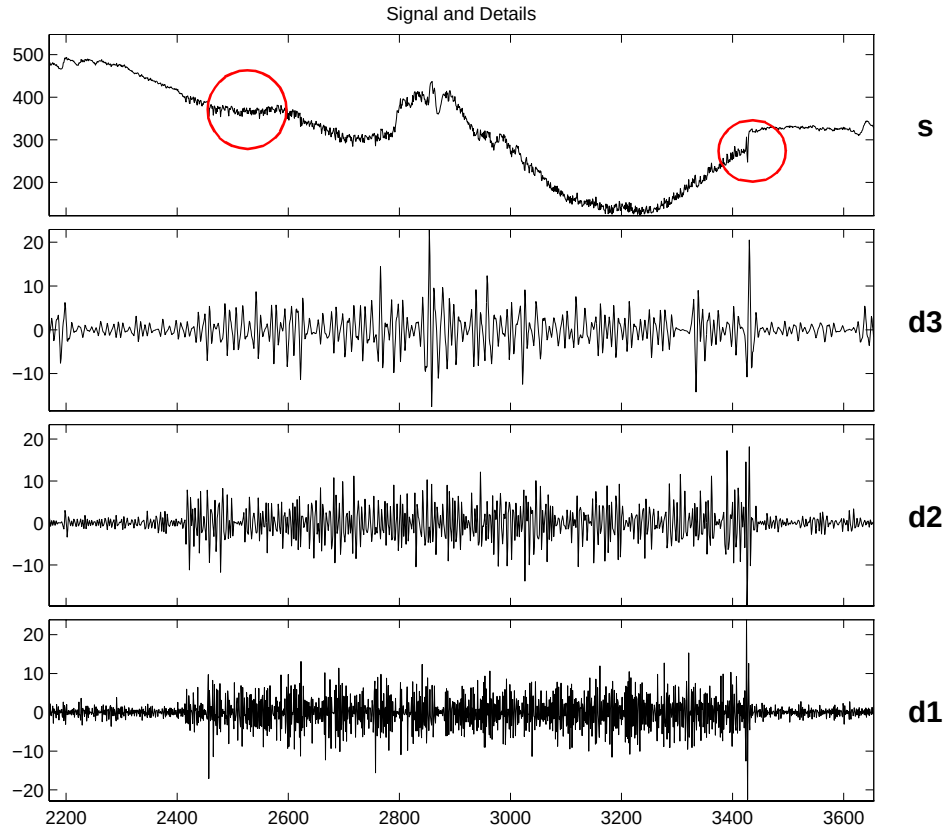


Figure 4-18: Identification of Sensor Failure

Suppress the Noise

Suppress measurement noise. Try by yourself and use, afterwards, the de-noising tools.

Indication: study the approximations and compare two successive days, the first without sensor failure and the second corrupted by failure (see Figure 4-19).

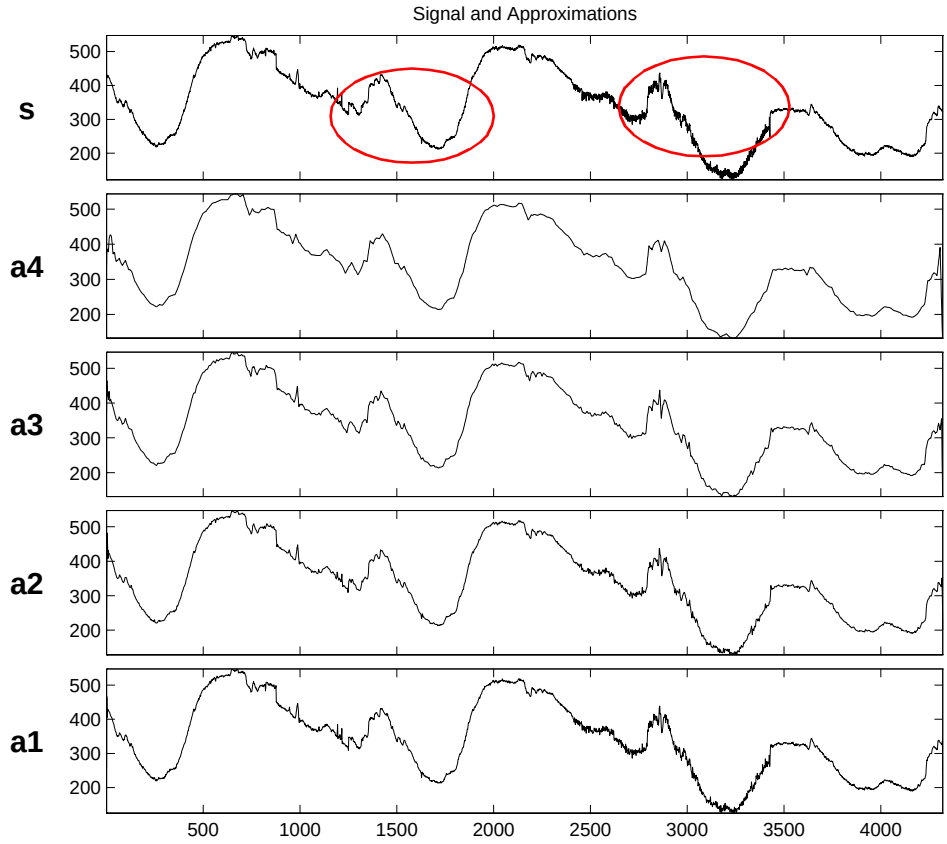


Figure 4-19: Comparison of Smoothed Versions of the Signal

Identify Patterns in the Details

The idea here is to identify a pattern in the details typical of relay-switched water heaters.

Indication: the Figure 4-20 gives an example of such a period. Focus on details D_2, D_3 and D_4 around abscissa 1350, 1383, and 1415 in order to detect abrupt changes of the signal induced by automatic switches.

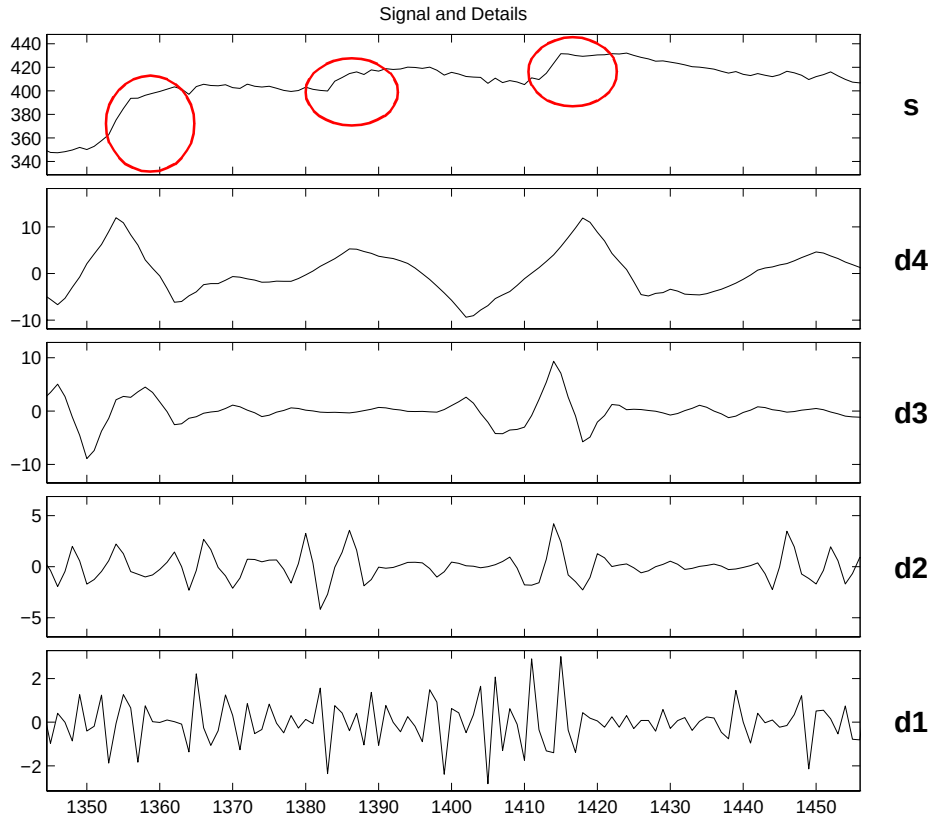


Figure 4-20: Location of the Water Heaters and Identification of the Effects on the Details

Locate and Suppress Outlying Values

Suppress the outliers by setting the corresponding values of the details to 0.

Indication: the Figure 4-21 gives two examples of outliers around t = 1193 and t = 1215, the effect produced on the details is clear when focusing on the low levels. As far as outliers are concerned, D_1 and D_2 are synchronized with s, D_3 shows a delayed effect.

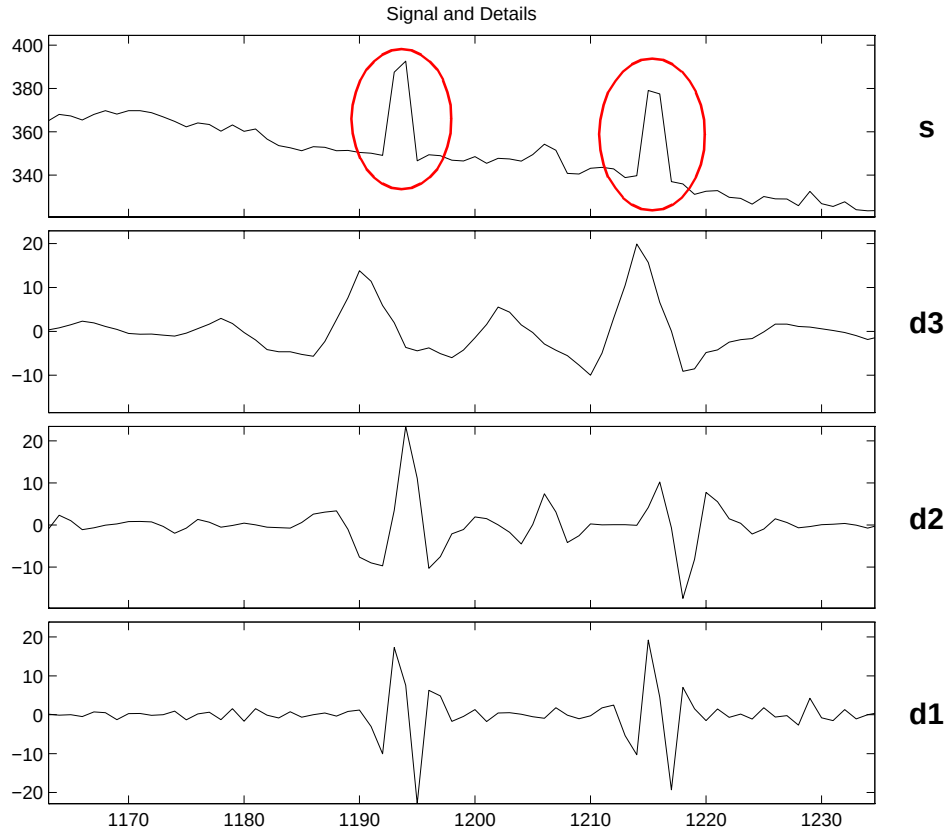


Figure 4-21: Location of the Outliers and Identification of the Effects on the Details

Study Missing Data

Missing data have been crudely substituted (around observation 2870) by estimation of half an hour sampled data and spline smoothing for the intermediate time points. Improve the interpolation by using an approximation and portions of the details taken elsewhere, thus implementing a sort of "graft."

Indication: see Figure 4-22 focusing around time 2870, and use the small variations part of D_1 in order to detect the missing data.

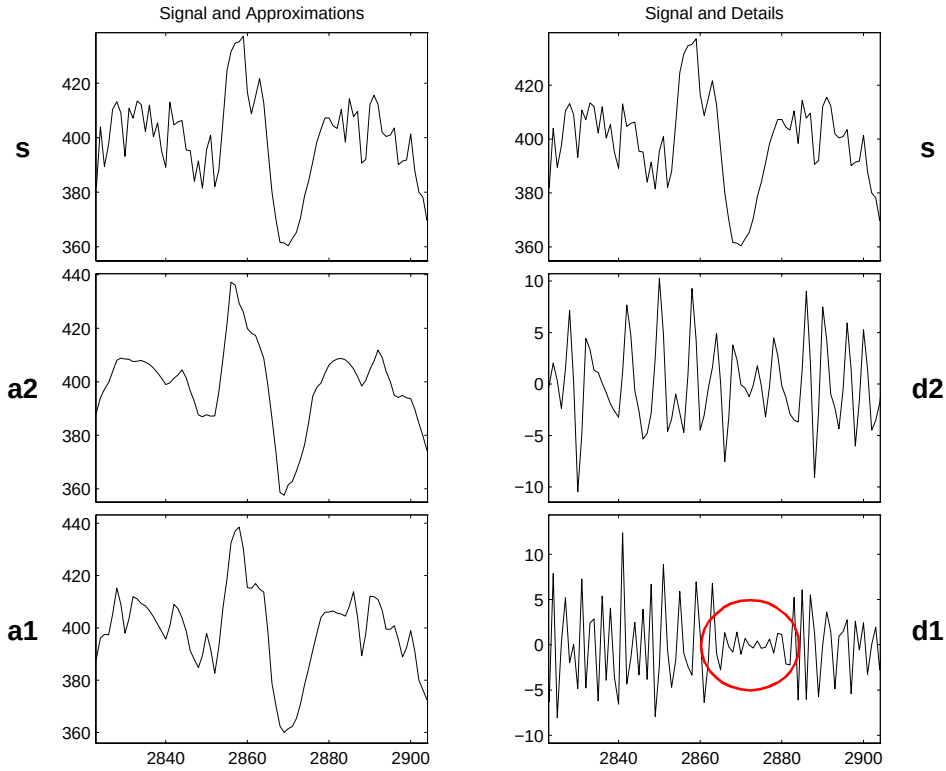


Figure 4-22: Detection of Missing Data Replaced Using Splines

Fast Multiplication of Large Matrices

This section illustrates matrix-vector multiplication in the wavelet domain.

- The problem is:

  let *m* be a dense matrix of large size (*n*, *n*). We want to perform a large number L of multiplications of *m* by vectors *v*.

- The idea is:

  Stage 1: (executed once) compute the matrix approximation *sm* at a suitable level *k*, the matrix being assimilated with an image.

  Stage 2: (executed *L* times) divided in the following three steps:

  1. Compute vector approximation.
  2. Compute multiplication in wavelet domain.
  3. Reconstruct vector approximation.

It is clear that when *sm* is a sufficiently good approximation of *m*, the error with respect to ordinary multiplication can be small. This is the case in the first example below where *m* is a magic square. Conversely, when the wavelet representation of the matrix *m* is dense, for example if all the coefficients have the same order of magnitude, the error will be large. This is the case in the second example below where *m* is a two-dimensional Gaussian white noise. The Figure 4-23 compares for $n = 512$, the number of flops required by wavelet based method and by ordinary method versus *L*.

Example 1: Effective Fast Matrix Multiplication

```
n = 512; lev = 5; wav = 'db1';

% Wavelet based matrix multiplication by a vector:
% a "good" example
% Matrix is magic(512) Vector is (1:512)

m = magic(n); v = (1:n)';
[LoF_D,HiF_D,LoF_R,HiF_R] = wfilters(wav);

% ordinary matrix multiplication by a vector.
flops(0), p = m * v; flomv = flops

flomv =
       524288
% Compute matrix approximation at level 5.
flops(0)
sm = m;
for i = 1:lev
    sm = dyaddown(conv2(sm,LoF_D),'c');
    sm = dyaddown(conv2(sm,LoF_D'),'r');
end
flmapp = flops

flmapp =
        2095104

% The three steps:
% 1. Compute vector approximation.
% 2. Compute multiplication in wavelet domain.
% 3. Reconstruct vector approximation.
```

```
flops(0)
sv = v;
for i = 1:lev, sv = dyaddown(conv(sv,LoF_D)); end
sp = sm * sv;
for i = 1:lev, sp = conv(dyadup(sp),LoF_R); end
sp = wkeep(sp,length(v));
flwmv = flops

flwmv =
        8958

% Plot ordinary versus wavelet based m*v flops in loglog.
```

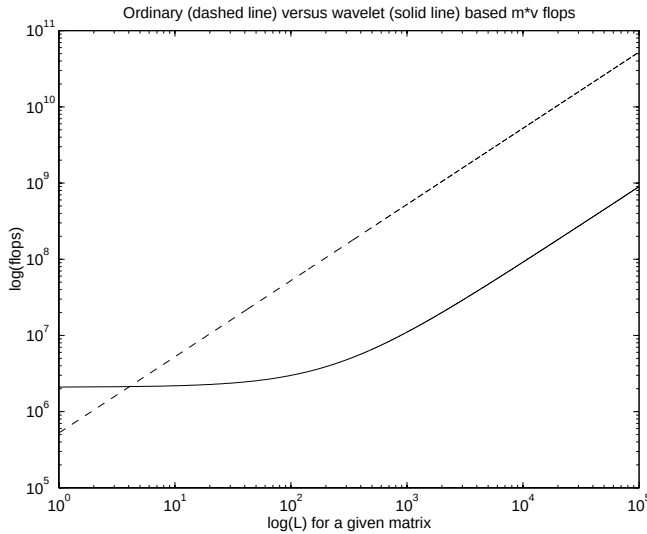


Figure 4-23: Wavelet Based Matrix Multiplication by a Vector

```
% Relative square norm error in percent when using wavelets.
rnrm = 100 * (norm(p-sp)/norm(p))

rnrm =
        2.9744e-06
```

Example 2: Ineffective Fast Matrix Multiplication

The commands used are the same as in Example 1.

```
% Wavelet based matrix multiplication by a vector:
% a "bad" example
% Matrix is randn(512,512) Vector is (1:512)
% Relative square norm error in percent
rnrm = 100 * (norm(p-sp)/norm(p))

rnrm =
        98.8839
```

5

Using Wavelet Packets

The Wavelet Toolbox contains graphical tools and command line functions that let you:

- Examine and explore characteristics of individual wavelet packets.
- Perform wavelet packet analysis of one- and two-dimensional signals.
- Use wavelet packets to compress and remove noise from signals and images.

This chapter takes you step-by-step through examples that teach you how to use the **Wavelet Packet 1-D** and **Wavelet Packet 2-D** graphical tools. The last section discusses how to transfer information from the graphical tools into your disk, and back again.

Because of the inherent complexity of packing and unpacking complete wavelet packet decomposition tree structures, we recommend using the **Wavelet Packet 1-D** and **Wavelet Packet 2-D** graphical tools for performing exploratory analyses.

The command line functions are also available and provide the same capabilities. However, it is most efficient to use the command line only for performing batch processing.

About Wavelet Packet Analysis

This chapter takes you through the features of one- and two-dimensional wavelet packet analysis using the MATLAB Wavelet Toolbox. You'll learn how to:

- Load a signal or image
- Perform a wavelet packet analysis of a signal or image
- Remove noise from a signal
- Compress an image
- Show statistics and histograms

The Wavelet Toolbox provides these functions for wavelet packet analysis. For more information, see the Command Reference (Chapter 8). The reference entries for these functions include examples showing how to perform wavelet packet analysis via the command line.

Analysis-Decomposition Functions:

Function Name	Purpose
`wpdec` and `wpdec2`	Full decomposition
`wpsplt`	Decompose packet

Synthesis-Reconstruction Functions:

Function Name	Purpose
`wprcoef`	Reconstruct coefficients
`wprec`	Full reconstruction
`wpjoin`	Recompose packet

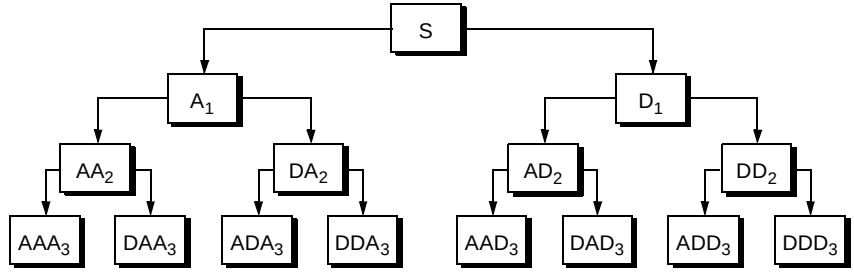
Decomposition Structure Utilities:

Function Name	Purpose
`besttree`	Find best tree
`bestlevt`	Find best level tree
`wentropy`	Entropy
`entrupd`	Update wavelet packets entropy

De-noising and Compression:

Function Name	Purpose
`ddencmp`	Default values for de-noising and compression
`wpthcoef`	Wavelet packets coefficients thresholding
`wpdencmp`	De-noising and compression using wavelet packets

In the wavelet packet framework, compression and de-noising ideas are exactly the same as those developed in the wavelet framework. The only difference is that wavelet packets offer a more complex and flexible analysis, because in wavelet packet analysis, the details as well as the approximations are split:



A single wavelet packet decomposition gives a lot of bases from which you can look for the best representation with respect to a design objective. This can be done by finding the “best tree” based on an entropy criterion.

De-noising and compression are interesting applications of wavelet packet analysis. The wavelet packet de-noising or compression procedure involves four steps:

1 Decomposition

   For a given wavelet, compute the wavelet packet decomposition of signal `x` at level `N`.

2 Computation of the best tree

   For a given entropy, compute the optimal wavelet packet tree. Of course, this step is optional. The graphical tools provide a **Best Tree** button for making this computation quick and easy.

3 Thresholding of wavelet packet coefficients

   For each packet (except for the approximation), select a threshold and apply thresholding to coefficients.

   The graphical tools automatically provide an initial threshold based on balancing the amount of compression and retained energy. This threshold is a reasonable first approximation for most cases. However, in general you will have to refine your threshold by trial and error so as to optimize the results to fit your particular analysis and design criteria.

   The tools facilitate experimentation with different thresholds, and make it easy to alter the trade-off between amount of compression and retained signal energy.

4 Reconstruction

   Compute wavelet packet reconstruction based on the original approximation coefficients at level `N` and the modified coefficients.

In this example we'll show how you can use one-dimensional wavelet packet analysis to compress and to de-noise a signal.

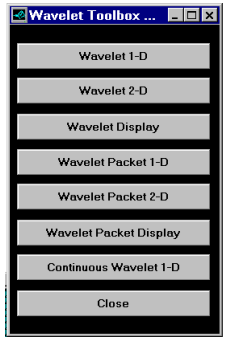
One-Dimensional Wavelet Packet Analysis

We now turn to the **Wavelet Packet 1-D** tool to analyze a synthetic signal that is the sum of two linear chirps.

Starting the Wavelet Packet 1-D Tool.

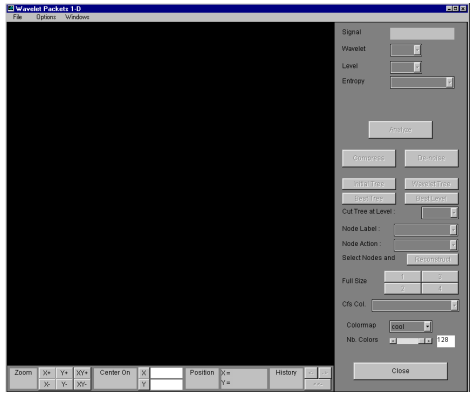
1 From the MATLAB prompt, type `wavemenu`.

The **Wavelet Toolbox Main Menu** appears.



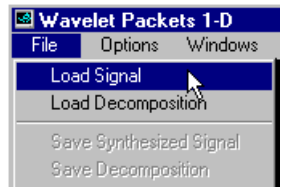
2 Click the **Wavelet Packet 1-D** menu item.

The tool appears on the desktop.

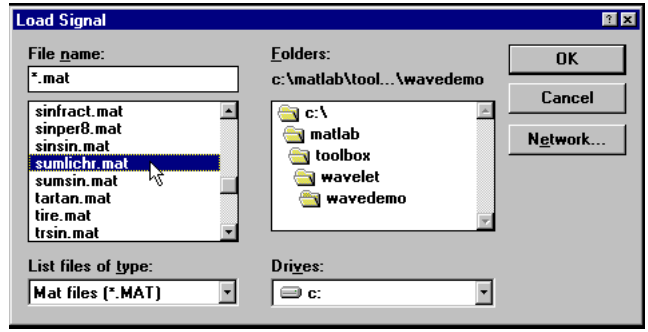


Loading a Signal.

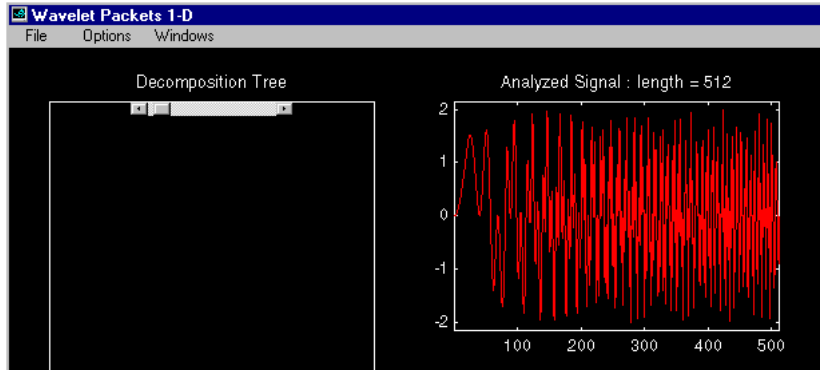
3 From the **File** menu, choose the **Load Signal** option.



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

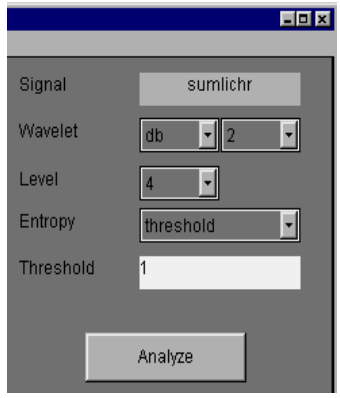


The `sumlichr` signal is loaded into the **Wavelet Packet 1-D** tool.



Analyzing a Signal.

5 Make the appropriate settings for the analysis. Select the `db2` wavelet, level 4, entropy type `threshold`, and threshold parameter 1. Click the **Analyze** button.



The available entropy types are:

Type	Description
Shannon	Non-normalized entropy involving the logarithm of the squared value of each signal sample — or, more formally: $-\sum s_i^2 \log(s_i^2)$
Threshold	The number of samples for which the absolute value of the signal exceeds a threshold ε.
Norm	The concentration in l^p norm with $1 \le p < 2$.
Log Energy	The logarithm of "energy," defined as the sum over all samples: $\sum \log(s_i^2)$

Type	Description
SURE (Stein's Unbiased Risk Estimate)	A threshold-based method in which the threshold equals: $\sqrt{2\log_e(n\log_2(n))}$ where n is the number of samples in the signal.
User	An entropy type criterion you define in an M-file.

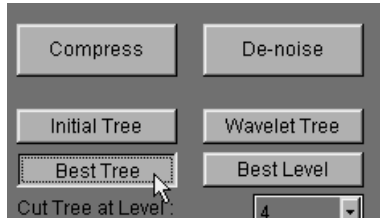
For more information about the available entropy types, user-defined entropy, and threshold parameters, see the reference entry for `wentropy`, and Chapter 6.

Note: Many capabilities are available using the command area on the right of the **Wavelet Packet 1-D** window. Some of them are used in the sequel. Please refer to the Appendix A, "GUI Reference" for a more complete description.

Computing the Best Tree.

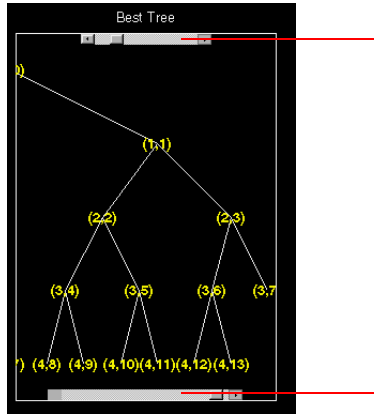
Because there are so many ways to reconstruct the original signal from the wavelet packet decomposition tree, we select the best tree before attempting to compress the signal.

6 Click the **Best Tree** button.



After a pause for computation, the **Wavelet Packet 1-D** tool displays the best tree. Use the top and bottom sliders to spread nodes apart and pan over to particular areas of the tree, respectively.

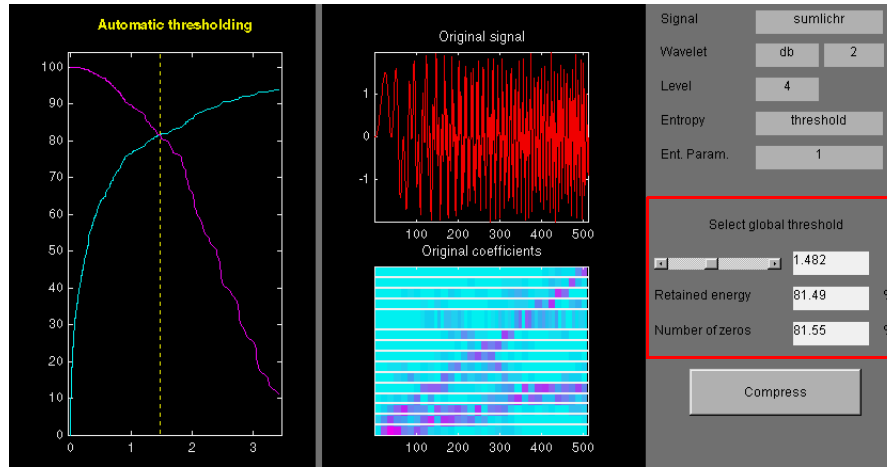
Observe that, for this analysis, the best tree and the initial tree are almost the same. One branch at the far right of the tree was eliminated.



Selecting a Threshold for Compression.

7 Click the **Compress** button.

The **Wavelet Packet 1-D Compression** window appears with an approximate threshold value automatically selected.



The left most graph shows how the threshold (vertical yellow dotted line) has been chosen automatically (1.482) to balance the number of zeros in the compressed signal (blue curve that increases as the threshold increases) with the amount of energy retained in the compressed signal (purple curve that decreases as the threshold increases).

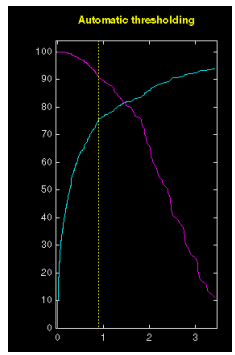
This threshold means that any signal element whose value is less than 1.482 will be set to zero when we perform the compression.

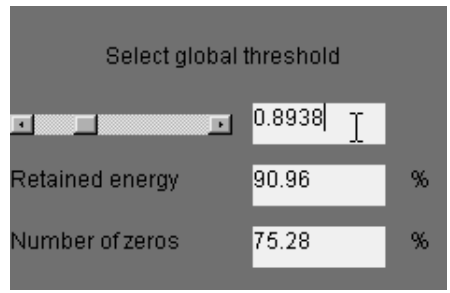
Threshold controls are located to the right (see red box in figure). Note that the automatic threshold of 1.482 results in a retained energy of only 81.49%. This may cause unacceptable amounts of distortion, especially in the peak values of the oscillating signal. Depending on your design criteria, you may want to choose a threshold that retains more of the original signal's energy.

8 Adjust the threshold by typing `0.8938` in the text field opposite the threshold slider, then press the **Enter** key.

The value `0.8938` is a number that we have discovered through trial and error yields more satisfactory results for this analysis.

After a pause, the **Wavelet Packet 1-D Compression** window displays new information.





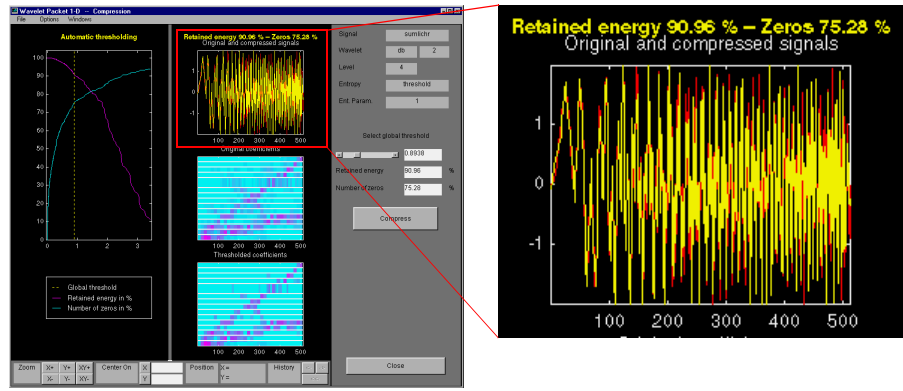
Note that, as we have *reduced* the threshold from 1.482 to 0.8938:

- The vertical yellow dotted line has shifted to the left.
- The retained energy has *increased* from 81.49% to 90.96%.
- The number of zeros (equivalent to the amount of compression) has *decreased* from 81.55% to 75.28%.

Compressing a Signal.

9 Click the **Compress** button.

The **Wavelet Packet 1-D** tool compresses the signal using the thresholding criterion we selected.



The original (red) and compressed (yellow) signals are displayed superimposed. Visual inspection suggests the compression quality is quite good.

Looking more closely at the compressed signal, we see that the number of zeros in the wavelet packets representation of the compressed signal is about 75.3%, and the retained energy about 91%.

If you try to compress the same signal using wavelets with exactly the same parameters, only 89% of the signal energy is retained, and only 59% of the wavelet coefficients set to zero. This illustrates the superiority of wavelet packets for performing compression, at least on certain signals.

You can demonstrate this to yourself by returning to the main **Wavelet Packet 1-D** window, computing the wavelet tree, and then repeating the compression.

De-Noising a Signal Using Wavelet Packet

We now use the **Wavelet Packet 1-D** tool to analyze a noisy chirp signal. This analysis illustrates the use of Stein's Unbiased Estimate of Risk (SURE) as a principle for selecting a threshold to be used for de-noising.

This technique calls for setting the threshold T to

$$T = \sqrt{2\log_e(n\log_2(n))}$$

where n is the length of the signal.

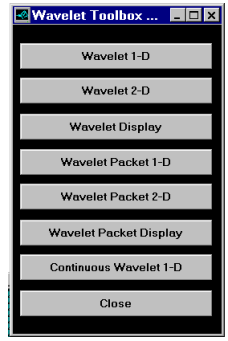
A more thorough discussion of the SURE criterion appears in Chapter 6. For now, suffice it to say that this method works well if your signal is normalized in such a way that the data fit the model $x(t) = f(t) + e(t)$, where $e(t)$ is a Gaussian white noise with zero mean and unit variance.

If you've already started the **Wavelet Packet 1-D** tool and it is active on your computer's desktop, *skip to step 3*.

Starting the Wavelet Packet 1-D Tool.

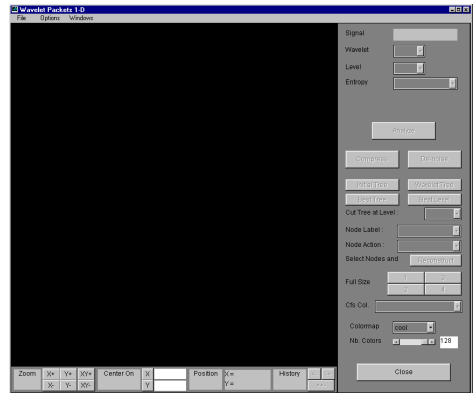
1 From the MATLAB prompt, type `wavemenu`.

The **Wavelet Toolbox Main Menu** appears.



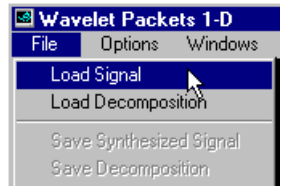
2 Click the **Wavelet Packet 1-D** menu item.

The tool appears on the desktop.

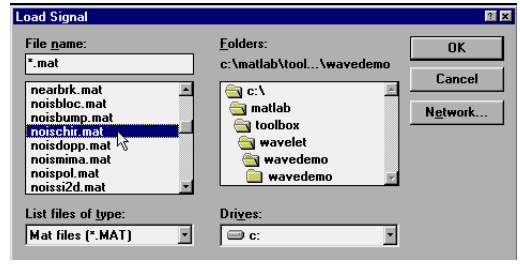


Loading a Signal.

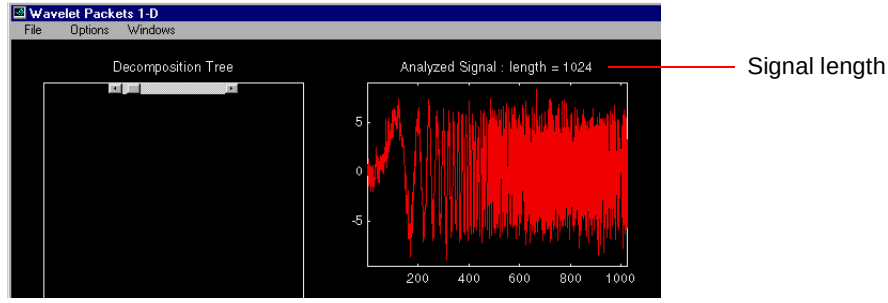
3 From the **File** menu, choose the **Load Signal** option.



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

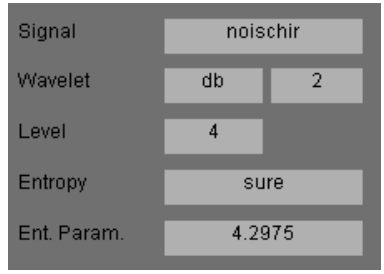


The noischir signal is loaded into the **Wavelet Packet 1-D** tool. Notice that the signal's length is 1024. This means we should set the SURE criterion threshold equal to `sqrt(2.*log(1024.*log2(1024)))`, or 4.2975.



Analyzing a Signal.

5 Make the appropriate settings for the analysis. Select the db2 wavelet, level 4, entropy type sure, and threshold parameter 4.2975. Click the **Analyze** button.



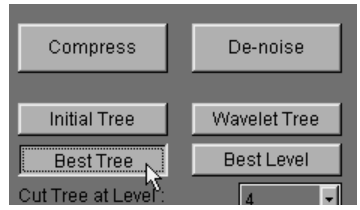
There is a pause while the wavelet packet analysis is computed.

Note: Many capabilities are available using the command area on the right of the **Wavelet Packet 1-D** window. Some of them are used in the sequel. Please refer to the Appendix A, "GUI Reference" for a more complete description.

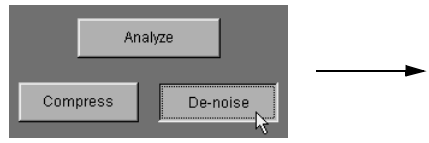
Computing the Best Tree and Performing De-Noising.

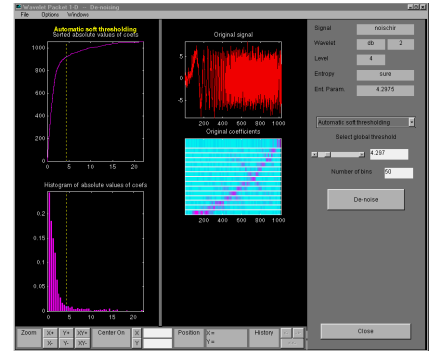
6 Click the **Best Tree** button.

Computing the best tree makes the de-noising calculations more efficient.



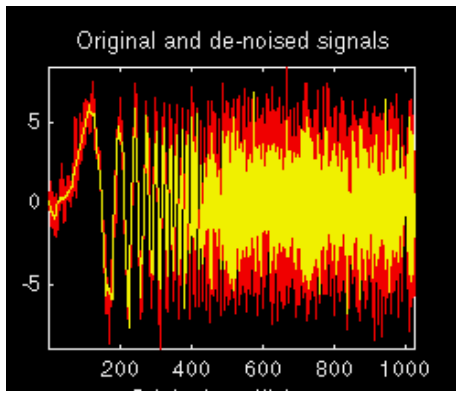
7 Click the **De-noise** button, bringing up the **Wavelet Packet 1-D De-Noising** window.

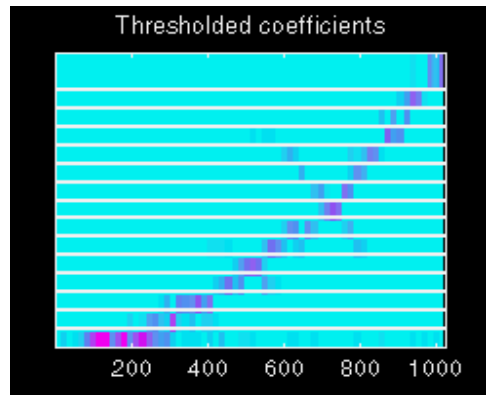




8 Click the **De-noise** button located at the center right side of the **Wavelet Packet 1-D De-Noising** window.

The results of the de-noising operation are quite good, as can be seen by looking at the thresholded coefficients. The frequency of the chirp signal increases quadratically over time, and the thresholded coefficients essentially capture the quadratic curve in the time-frequency plane.





You can also use the M-file `wpdencmp` to perform wavelet packet de-noising or compression from the command line.

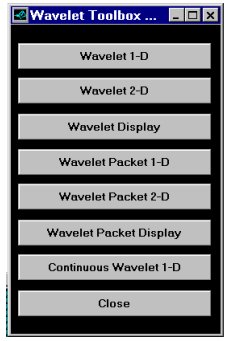
Two-Dimensional Wavelet Packet Analysis

In this section, we employ the **Wavelet Packet 2-D** tool to analyze and compress an image of a fingerprint. This is a real-world problem: the Federal Bureau of Investigation (FBI) maintains a large database of fingerprints — about 30 million sets of them. The cost of storing all this data runs to hundreds of millions of dollars. By turning to wavelets, the FBI has achieved a 15:1 compression ratio. In this application, wavelet compression is better than the more traditional JPEG compression, as it avoids small square artifacts, and is particularly well suited to detect discontinuities (lines) in the fingerprint.

Starting the Wavelet Packet 2-D Tool.

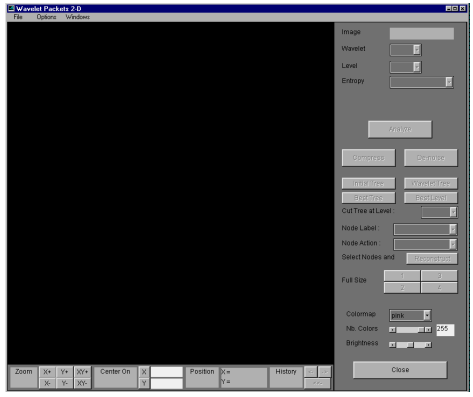
1 From the MATLAB prompt, type:
```
» wavemenu.
```

The **Wavelet Toolbox Main Menu** appears.



2 Click the **Wavelet Packet 2-D** menu item.

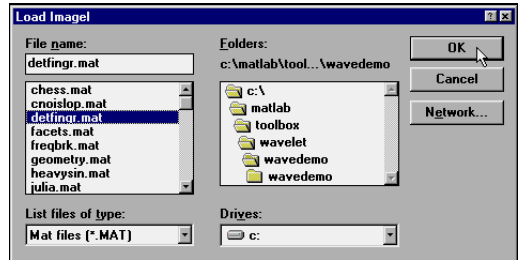
The tool appears on the desktop.



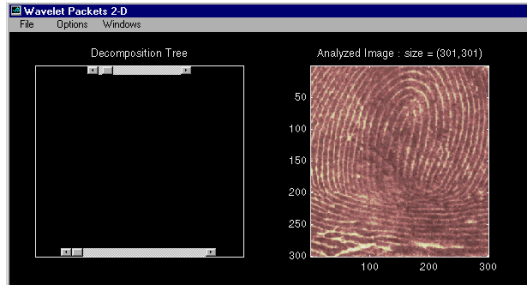
Loading an Image.

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

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



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



Analyzing an Image.

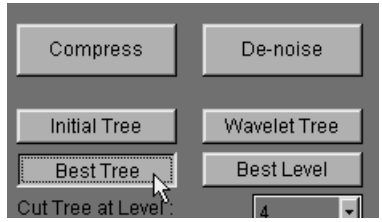
4 Make the appropriate settings for the analysis. Select the haar wavelet, level 3, and entropy type shannon. Click the **Analyze** button.

There is a pause while the wavelet packet analysis is computed.

Note: Many capabilities are available using the command area on the right of the **Wavelet Packet 2-D** window. Some of them are used in the sequel. Please refer to the Appendix A, "GUI Reference" for a more complete description.

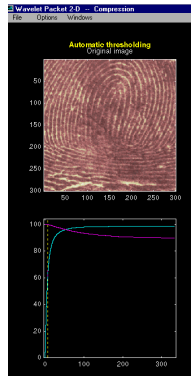
---

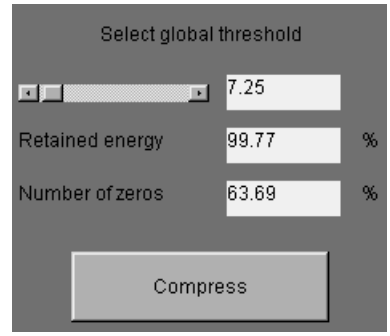
5 Click the **Best Tree** button to compute the best tree before compressing the image.



Compressing an image.

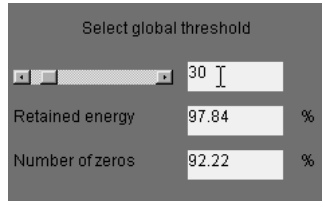
6 Click the **Compress** button to bring up the **Wavelet Packet 2-D Compression** window.





Notice that the default threshold (7.25) provides about 64% compression while retaining virtually all the energy of the original image. Depending on your criteria, it may be worthwhile experimenting with more aggressive thresholds to achieve a higher degree of compression. Recall that we are not doing any quantization of the image, merely setting specific coefficients to zero. This can be considered a pre-compression step in a broader compression system.

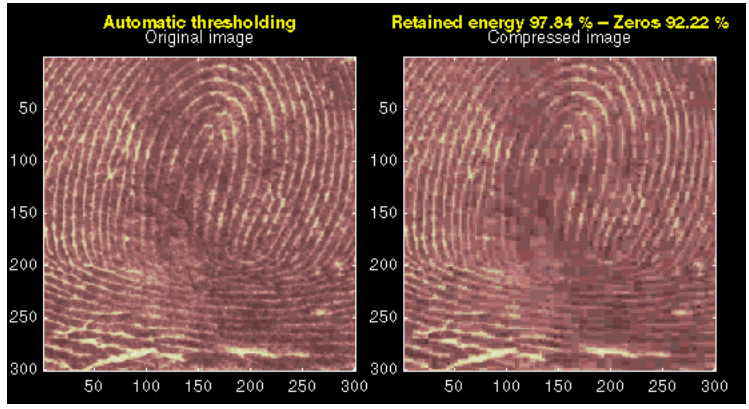
7 Alter the threshold: type the number `30` in the text field opposite the threshold slider located on the right side of the **Wavelet Packet 2-D Compression** window. Then press the **Enter** key.



Setting all wavelet packet coefficients whose value falls below 30 to zero yields much better results. Note that the new threshold achieves a compression ratio of better than 12:1, while still retaining nearly 98% of the image energy. Compare this wavelet packet analysis to the wavelet analysis of the same image in “Compressing Signals” in Chapter 3.

8 Click the **Compress** button to start the compression.

You can see the result obtained by wavelet packet coefficients thresholding and image reconstruction. The visual recovery is correct but not perfect. The compressed image, shown side-by-side with the original, shows some artifacts.

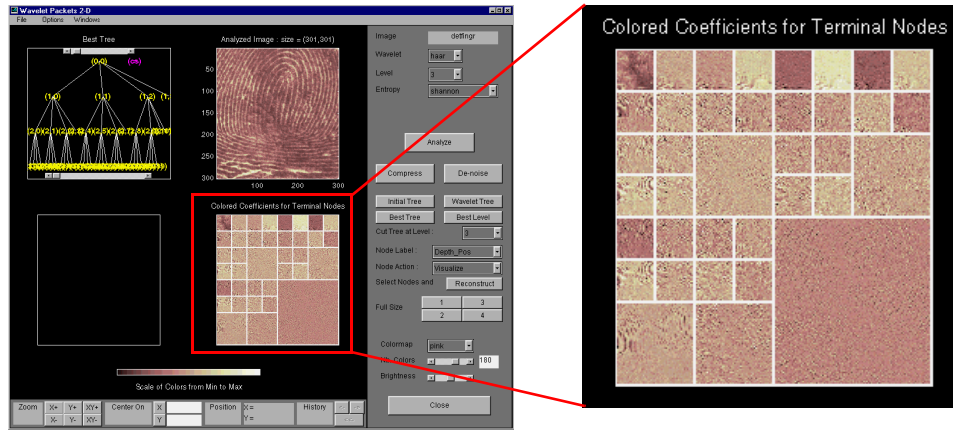


9 Click the **Close** button located at the bottom of the **Wavelet Packet 2-D Compression** window. Update the synthesized image by clicking **Yes** when the dialog box appears.

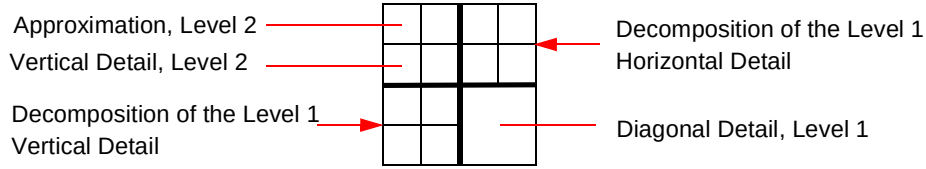
Take this opportunity to try out your own compression strategy. Adjust the threshold value, the entropy function, and the wavelet, and see if you can obtain better results.

Hint: The `bior6.8` wavelet is better suited to this analysis than is `haar`, and can lead to a compression ratio of 24:1.

Before concluding this analysis, it is worth turning our attention to the "colored coefficients for terminal nodes plot" and considering the best tree decomposition for this image.



This plot is shown in the lower right side of the **Wavelet Packet 2-D** tool. The plot shows us which details have been decomposed and which have not. Larger squares represent details that have not been broken down to as many levels as smaller squares. Consider, for example, this level 2 decomposition pattern:



Looking at the pattern of small and large squares in the fingerprint analysis shows that the best tree algorithm has apparently singled out the diagonal details, often sparing these from further decomposition. Why is this?

If we consider the original image, we realize that much of its information is concentrated in the sharp edges that constitute the fingerprint's pattern. Looking at these edges, we see that they are predominantly oriented horizontally and vertically. This explains why the best tree algorithm has "chosen" not to decompose the diagonal details — they do not provide very much information.

Importing and Exporting from Graphical Tools

The **Wavelet Packet 1-D** and **Wavelet Packet 2-D** tools let you import information from and export information to your disk.

If you adhere to the proper file formats, you can:

- Save decompositions as well as synthesized signals and images from the wavelet packet graphical tools into your disk.
- Load signals, images, and one- and two-dimensional decompositions from your disk into the **Wavelet Packet 1-D** and **Wavelet Packet 2-D** graphical tools.

Saving Information to the Disk

The graphical tools' functions let you save synthesized signals or images, as well as one- or two-dimensional wavelet packet decomposition structures, using specific file formats. This feature provides flexibility and allows you to combine command line and graphical interface operations.

Saving Synthesized Signals

You can process a signal in the **Wavelet Packet 1-D** tool and then save the processed signal to a MAT-file.

For example, load the demo analysis: **File⇒Demo Analysis⇒with db3 at level 5 —> Sum of sines**, and perform a compression or de-noising operation on the original signal. When you close the **Wavelet Packet 1-D De-noising** or **Wavelet Packet 1-D Compression** window, update the synthesized signal by clicking **Yes** in the dialog box.

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

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

To load the signal into your workspace, simply type:

```
» load synthsig
» whos
```

```
     Name       Size       Elements   Bytes      Class
     synthsig   1 by 1000  1000       8000       double array
```

Saving Synthesized Images

You can process an image in the **Wavelet Packet 2-D** tool and then save the processed image to a MAT-file.

For example, load the demo analysis **File⇒Demo Analysis⇒db1 – depth: 1 – ent: shannon —> woman**, and perform a compression on the original image. When you close the **Wavelet Packet 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 directory and filename for the MAT-file. For this example, choose the name `wpsymage`.

To load the image into your workspace, simply type:

```
» load wpsymage
» whos
```

```
  Name       Size          Elements   Bytes      Class
  map        255 by 3      765        6120       double array
  wpsymage   256 by 256    65536      524288     double array
```

Saving One-Dimensional Decomposition Structures

The **Wavelet Packet 1-D** tool lets you save an entire wavelet packet decomposition tree and related data to your disk. The toolbox creates a MAT-file in the current directory with a name you choose, followed by the extension `wp1` (wavelet packet 1-D).

Open the **Wavelet Packet 1-D** tool and load the demo analysis **File⇒Demo Analysis⇒db1 – depth: 2 – ent: shannon —> sumsin**.

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

A dialog box appears that lets you specify a directory and file name for storing the decomposition data. Type the name `wpdecex`.

After saving the decomposition data to the file wpdecex1d.wp1, load the variables into your workspace.

```
» load wpdecex1d.wp1 -mat
» whos
```

```
Name          Size       Elements   Bytes     Class
data_name     1 by 6     6          48        double array
data_struct   1 by 1057   1057       8456      double array
tree_struct   2 by 5     3          80        double array
```

Variables tree_struct and data_struct contain the wavelet packet decomposition structure (tree and data). The other variable contains the data name.

Saving Two-Dimensional Decomposition Structures

The file format, variables, and conventions are exactly the same as in the one-dimensional case except for the extension, which is wp2 (wavelet packet 2-D). The variables saved are the same as with the one-dimensional case, with the addition of the colormap matrix map:

```
Name         Size          Elements   Bytes     Class
data_name    1 by 5        5          40        double array
data_struct  1 by 65590    65590      524720    double array
map          255 by 3      765        6120      double array
tree_struct  3 by 5        15         120       double array
```

Loading Information into the Graphical Tools

You can load signals, images, or one- and two-dimensional wavelet packet decompositions into the graphical interface tools. 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 graphical tools, 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 Packet 1-D** tool, save the signal in a MAT-file that has the same name as the signal variable itself.

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

```
» lsave warma
```

The workspace variable `warma` must be a vector.

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

To load this signal into the **Wavelet Packet 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.

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 a `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 Packet 2-D** graphical 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 Packet 2-D** tool, save the image (and optionally, the variable `map`) in a MAT-file that has the same name as the image matrix itself.

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

```
» lsave brain
```

To load this image into the **Wavelet Packet 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.

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

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

In order to display these images in a suitable way, the **Wavelet Packet 2-D** tool follows these rules:

- Reconstructed approximations are displayed using the colormap map. The same holds for the result of the reconstruction of selected nodes.
- The coefficients and the reconstructed details are displayed using the colormap map applied to a rescaled version of the matrices.

Loading Wavelet Packet Decomposition Structures

You can load one- and two-dimensional wavelet packet decompositions into the graphical tools providing you have previously saved the decomposition data in a MAT-file of the appropriate format.

While it is possible to edit data originally created using the graphical tools and then exported, you must be careful about doing so. Wavelet packet data structures are complex, and the graphical tools do not do any consistency checking. This can lead to errors if you try to load improperly formatted data.

One-dimensional data structures must contain the variables:

Variable	Description
data_name	String specifying name of decomposition
data_struct	Vector specifying data in tree structure
tree_struct	Vector specifying tree structure

These variables must be saved in a MAT-file with the extension `.wp1`.

Two-dimensional data structures must contain the variables:

Variable	Description
`data_name`	String specifying name of decomposition
`data_struct`	Vector specifying data in tree structure
`map`	Image map
`tree_struct`	Vector specifying tree structure

These variables must be saved in a MAT-file with the extension `.wp2`.

To load the properly-formatted data, use the menu option **File⇒Load Decomposition Structure** from the appropriate tool, then select the desired MAT-file from the dialog box that appears.

The **Wavelet Packet 1-D** or **2-D** graphical tool then automatically updates its display to show the new analysis.

6

Advanced Concepts

This chapter presents an alternative and more advanced treatment of wavelet methods. It assumes that the reader is comfortable with mathematical ideas. For more detail on the theory, the reader is directed to the book *Wavelets and Filter Banks* by Strang and Nguyen, and to the references at the end of this Chapter.

Mathematical Conventions

This chapter and the reference section use certain mathematical conventions.

General Notation	Interpretation
$a = 2^j, j \in Z$	Dyadic scale. j is the level, $1/a$ or 2^{-j} is the resolution
$b = ka, k \in Z$	Dyadic translation
t	Continuous time
k or n	Discrete time
(i, j)	Pixel
s	Signal or image. The signal is a function defined on R or Z, the image is defined on R^2 or Z^2. A finite-length signal is extended to all R, Z, R^2 or Z^2 using zeros (this is zero-padding).
$\hat{f}$	Fourier transform of the function f or the sequence f.
Continuous time	
$L^2(R)$	Set of signals of finite energy
$\int_R s(x)^2 dx$	Energy of the signal s
$\langle s, s' \rangle = \int_R s(x)s'(x)$	Scalar product of signals s and s'
$L^2(R^2)$	Set of images of finite energy
$\int_R \int_R s(x, y)^2 dxdy$	Energy of the image s
$\langle s, s' \rangle = \int_R \int_R s(x, y)s'(x, y)dxdy$	Scalar product of images s and s'

General Notation	Interpretation
Discrete time	
$l^2(Z)$	Set of signals of finite energy
$\sum_Z s(n)^2$	Energy of the signal s
$\langle s, s' \rangle = \sum_Z s(n)s'(n)$	Scalar product of signals s and s'
$l^2(Z^2)$	Set of images of finite energy
$\sum_Z \sum_Z s(n, m)^2$	Energy of the image s
$\langle s, s' \rangle = \sum_Z \sum_Z s(n, m)s'(n, m)$	Scalar product of images s and s'

Wavelet Notation	Interpretation
A_j	j-level approximation or approximation at level j
D_j	j-level detail or detail at level j
f	Scale or scaling function
y	Wavelet
$\frac{1}{\sqrt{a}}\psi\left(\frac{x-b}{a}\right)$	Family associated with the one-dimensional wavelet, $a > 0$ and $b \in R$.
$\frac{1}{\sqrt{a_1 a_2}}\psi\left(\frac{x_1-b_1}{a_1}, \frac{x_2-b_2}{a_2}\right), x = (x_1, x_2) \in R^2$	Family associated with the two-dimensional wavelet, $a_1 > 0, a_2 > 0, b_1 \in R, b_2 \in R$.
$\phi_{j,k}(x) = 2^{-j/2}\phi(2^{-j}x - k), j \in Z, k \in Z$	Family associated with the one-dimensional scale function for dyadic scales $a = 2^j$, $b = ka$. It should be noted that $\phi = \phi_{0,0}$.

Wavelet Notation	Interpretation
$\psi_{j,k}(x) = 2^{-j/2}\psi(2^{-j}x - k), j \in Z, k \in Z$	Family associated with the one-dimensional ψ for dyadic scales $a = 2^j$, $b = ka$. It should be noted that $\psi = \psi_{0,0}$.
$(h_k), k \in Z$	Scale or scaling filter associated with a discrete wavelet
$(g_k), k \in Z$	Discrete wavelet

General Concepts

This section presents a brief overview of wavelet concepts.

Wavelets: A New Tool for Signal Analysis

Wavelet analysis consists of decomposing a signal or an image into a hierarchical set of approximations and details. The levels in the hierarchy often correspond to those in a dyadic scale.

From the signal analyst's point of view, wavelet analysis is a decomposition of the signal on a family of analyzing signals, which is an "orthogonal function method." From an algorithmic point of view, wavelet analysis offers a harmonious compromise between decomposition and smoothing techniques.

Wavelet Decomposition: A Hierarchical Organization

Unlike conventional techniques, wavelet decomposition produces a family of hierarchically organized decompositions The selection of a suitable level for the hierarchy will depend on the signal and experience. Often the level is chosen based on a desired low-pass cutoff frequency.

At each level *j*, we build the *j*-level approximation, A_j, or approximation at level *j*, and a deviation signal called the *j*-level detail, D_j, or detail at level *j*. The original signal we could consider as the approximation at level 0, denoted by A_0. The words "approximation" and "detail" are justified by the fact that A_1 is an approximation of A_0 taking into account the "low frequencies" of A_0, whereas the detail D_1 corresponds to the "high frequency" correction. The figure on Page 1-23 graphically represents this hierarchical decomposition.

As outlined in Chapter 1, one way of understanding this decomposition consists of using an optical comparison. Successive images A_1, A_2, A_3 of a given object are built. We use the same type of photographic devices, but with increasingly poor resolution. The images are successive approximations; one detail is the discrepancy between two successive images. Image A_2 is therefore the sum of image A_4 and intermediate details D_4, D_3:

$$A_2 = A_3 + D_3 = A_4 + D_4 + D_3$$

Finer and Coarser Resolutions

The organizing parameter, the scale *a*, is related to level *j*, by $a = 2^j$. If we define resolution as 1/*a*, then the resolution increases as the scale decreases. The greater the resolution, the smaller and finer are the details that can be accessed.

j	10	9	...	2	1	0	-1	-2
Scale	1024	512	...	4	2	1	1/2	1/4
Resolution	$1/2^{10}$	$1/2^9$	...	1/4	1/2	1	2	4

From a technical point of view, the size of the revealed details for any *j* is proportional to the size of the domain in which the wavelet or analyzing function of the variable *x*, $\psi\left(\frac{x}{a}\right)$, is not too close to 0. The proportionality coefficient depends on the wavelet.

Wavelet Shapes

One-dimensional analysis is based on one scaling function ϕ and one wavelet ψ. Two-dimensional analysis (on a square or rectangular grid) is based on one scaling function $\phi(x_1, x_2)$ and three wavelets.

Figure 6-1 shows ϕ and ψ for each wavelet, except the Morlet wavelet and the Mexican hat for which ϕ does not exist. All the functions decay quickly to zero. The Haar wavelet is the only noncontinuous function with three points of discontinuity (0, 0.5, 1). The ψ functions oscillate more than associated ϕ functions. `coif2` exhibits some angular points, `db6` and `sym6` are quite smooth. The Morlet and Mexican hat wavelets are symmetrical.

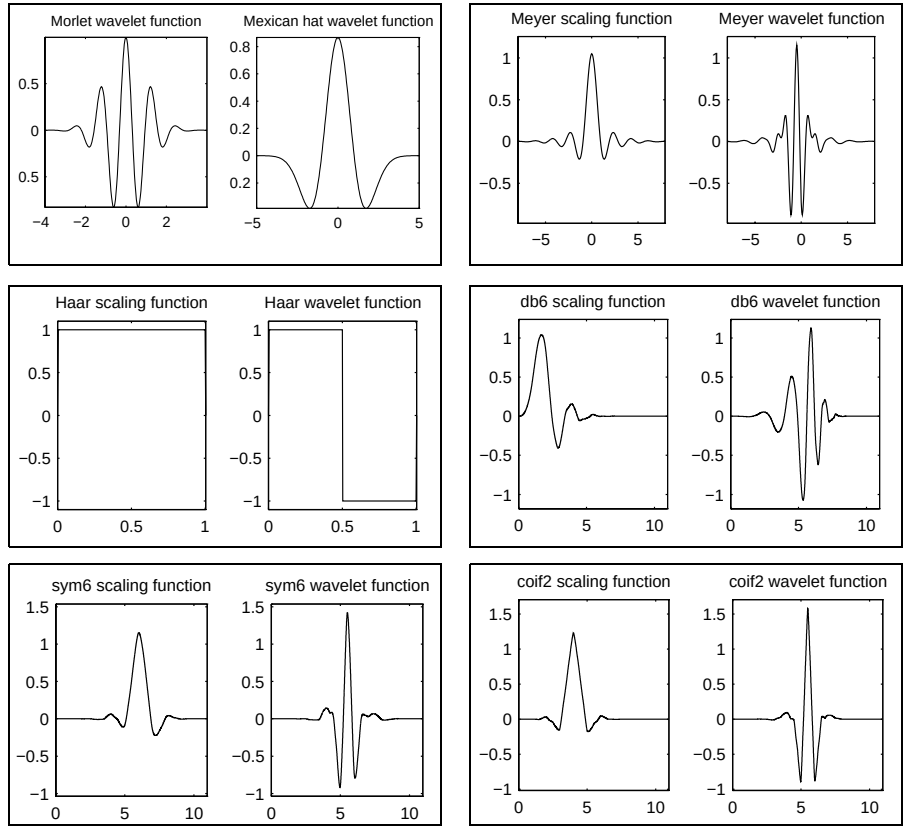


Figure 6-1: Various One-Dimensional Wavelets

Wavelets and Associated Families

In the one-dimensional context, we distinguish the wavelet ψ from the associated function ϕ, called the scaling function. Some properties of the ψ and ϕ are:

- The integral of ψ is zero, $\left(\int \psi(x)dx = 0\right)$, and ψ is used to define the details.
- The integral of ϕ is 1, $\left(\int \phi(x)dx = 1\right)$, and ϕ is used to define the approximations.

The usual two-dimensional wavelets are defined as tensor products of one-dimensional wavelets: $\phi(x,y) = \phi(x)\phi(y)$ is the scaling function and $\psi_1(x,y) = \phi(x)\psi(y)$, $\psi_2(x,y) = \psi(x)\phi(y)$, $\psi_3(x,y) = \psi(x)\psi(y)$ are the three wavelets.

Figure 6-2 shows the four functions associated with the two-dimensional `coif2` wavelet.

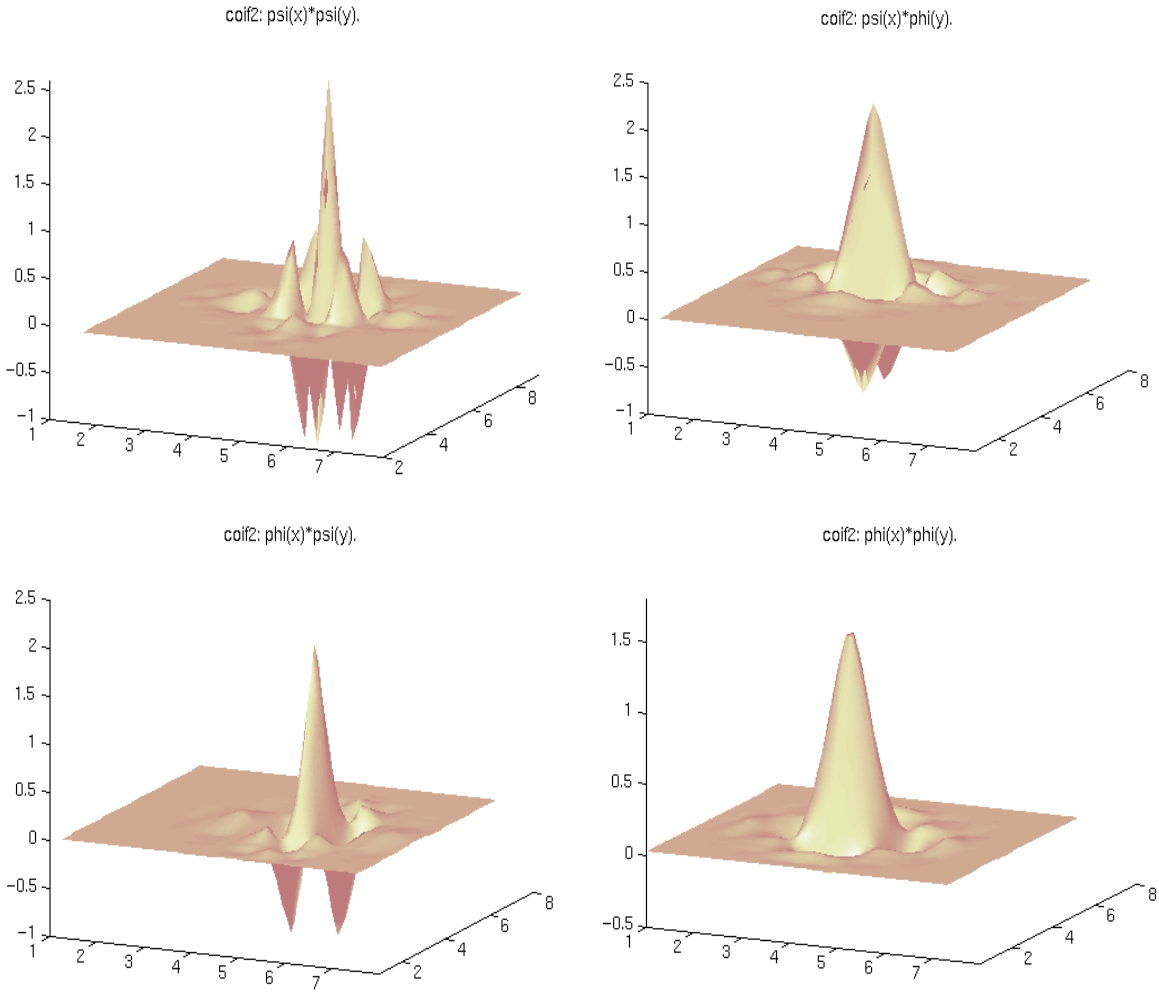


Figure 6-2: Two-Dimensional coif2 Wavelet

To each of these functions, we associate its doubly indexed family, which is used to:

- Move the shape, translating it to position *b* (see Figure 6-3).
- Keep the shape while changing the one-dimensional time scale *a* ($a > 0$), see Figure 6-4.

So a wavelet has to be thought as a function located at a position *b*, and having a scale *a*.

In one-dimensional situations, the family of translated and scaled wavelets associated with ψ is expressed as:

Translation	Change of scale	Translation and change of scale
$\psi(x\text{-}b)$	$\frac{1}{\sqrt{a}}\psi\left(\frac{x}{a}\right)$	$\frac{1}{\sqrt{a}}\psi\left(\frac{x-b}{a}\right)$

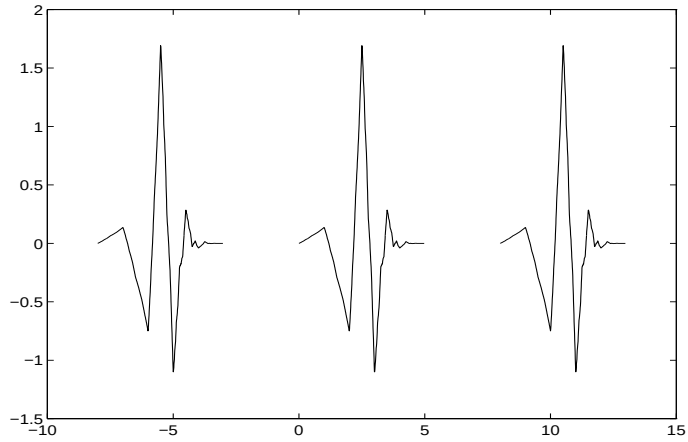


Figure 6-3: Translated Wavelets

Wavelet *db*3(*x*) is in the middle, *db*3(*x* + 8) on the left, *db*3(*x*-8) on the right.

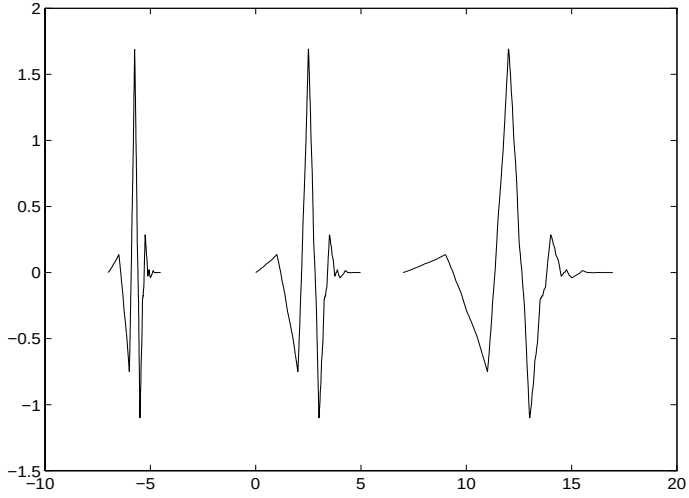


Figure 6-4: Time scaled one-dimensional wavelet

Wavelet *db*3(*x*) is in the middle, *db*3(2*x* + 7) on the left, *db*3(*x*/2 - 7) on the right.

In a two-dimensional context, we have the translation by vector $[b_1,b_2]'$ and a change of scale of parameter $[a_1,a_2]'$.

Translation and change of scale become: $\frac{1}{\sqrt{a_1 a_2}}\psi\left(\frac{x_1 - b_1}{a_1}, \frac{x_2 - b_2}{a_2}\right)$ where $x = (x_1,x_2) \in R^2$.

In most cases, we will limit our choice of *a* and *b* values by using only the following discrete set (coming back to the one-dimensional context):

$$(j,k) \in Z^2 : a = 2^j, \qquad b = k2^j = ka.$$

What is more, let us define:

$$(j,k) \in Z^2 : \psi_{j,k} = 2^{-j/2}\psi(2^{-j}x - k),\ \phi_{j,k} = 2^{-j/2}\phi(2^{-j}x - k).$$

We now have a hierarchical organization similar to the organization of a decomposition, which is represented in the example of the Figure 6-5. Let $k = 0$ and leave the translations aside for the moment. The functions (expressed as $\phi_{j,0}$) associated with $j = 0, 1, 2, 3$ for ϕ and with $j = 1, 2, 3$ for ψ (expressed as $\psi_{j,0}$) are displayed in Figure 6-5 for the `db3` wavelet.

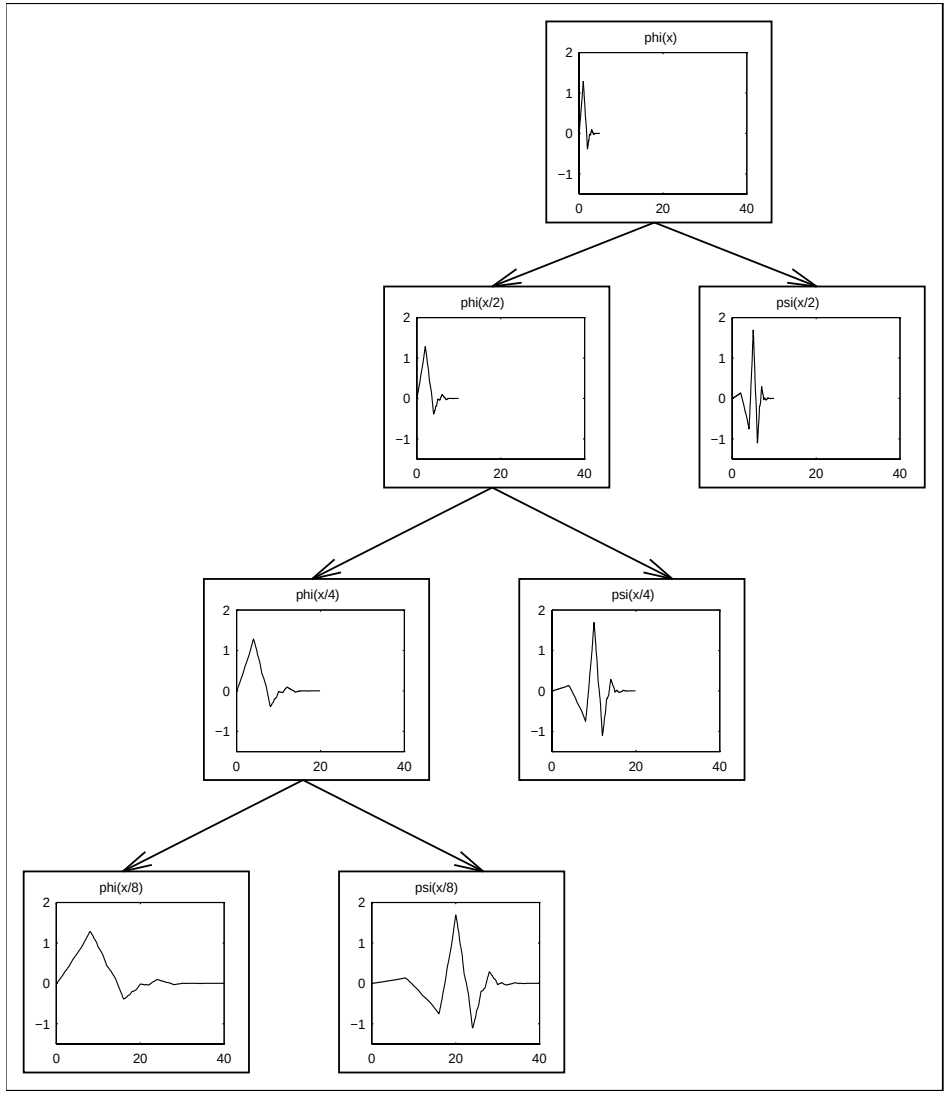


Figure 6-5: Wavelets Organization

In Figure 6-5, the four level decomposition is shown, progressing from the top to the bottom, we find $\phi_{0,0}$, then $2^{1/2}\phi_{1,0}$, $2^{1/2}\psi_{1,0}$ then $2\phi_{2,0}$, $2\psi_{2,0}$ then $2^{3/2}\phi_{3,0}$, $2^{3/2}\psi_{3,0}$. The wavelet is *db*3.

Wavelets on a Regular Discrete Grid

To complement the wavelets introduced previously, we use wavelets defined on grids, when the signal is sampled on a regular grid. The simplest of these wavelets is deduced from wavelets capable of analyzing the signals that are recorded continuously. Figure 6-6 shows some of these wavelets (non-normalized).

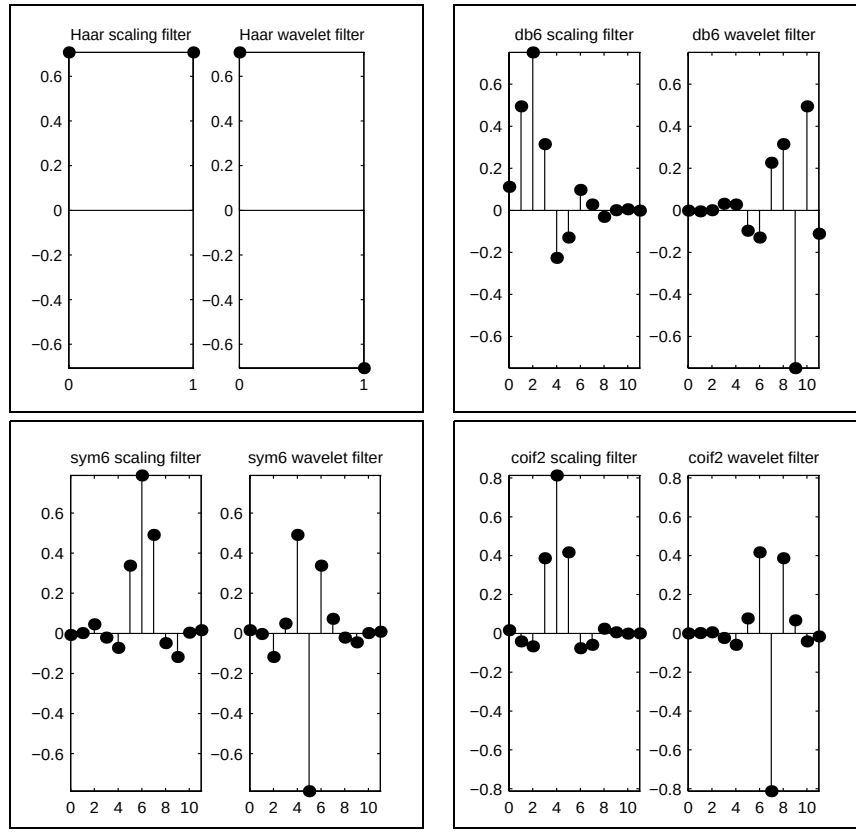


Figure 6-6: Discrete wavelets

The wavelet filter *g*, plays the role of ψ. The scaling filter *h* plays the role of ϕ. They are defined on a regular grid ΔZ, where Δ is the sampling period. Let us set $\Delta = 1$. Like the previous wavelets, functions *g* and *h* are subjected to scalings and translations.

Using a function *g* defined on *Z* and a scale equal to 2^j, for $j \in N$ and $k \in N$, we define the function $g_{j,k}$ by:

$$n \in Z, \qquad g_{j,k}(n) = 2^{-j/2} g(2^{-j} n - k)$$

Wavelet Transforms: Continuous and Discrete

The wavelet transform of a signal *s* is the family *C*(*a*,*b*), which depends on two indices *a* and *b*. The set to which *a* and *b* belong is given below in the table. The studies focus on three kinds of signals:

- Continuous time signal, recorded continuously
- Sampled signal
- Discrete time signal recorded in discrete time

and two transforms:

- Continuous transform
- Discrete transform.

From an intuitive point of view, the wavelet decomposition consists of calculating a “resemblance index” between the signal and the wavelet. If the index is large, the resemblance is strong, otherwise it is slight. The indexes *C*(*a*,*b*) are called coefficients.

We define the coefficients in the following tables. We have three types of analysis at our disposal.

Continuous time "continuous" analysis	**Continuous time "discrete" analysis**	**Discrete time (Δ = 1) "discrete" analysis**
$C(a,b) = \int_R s(t)\frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)dt$	$C(a,b) = \int_R s(t)\frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)dt$	$C(a,b) = C(j,k) = \sum_{n \in Z} s(n) g_{j,k}(n)$
$a \in R^+ - \{0\}, b \in R$	$a = \Delta 2^j, b = \Delta k 2^j, (j,k) \in Z^2$	$a = 2^j, b = k2^j, j \in N, k \in Z$

Let us illustrate the differences between the two transforms, for the analysis of a fractal signal (see Figure 6-7).

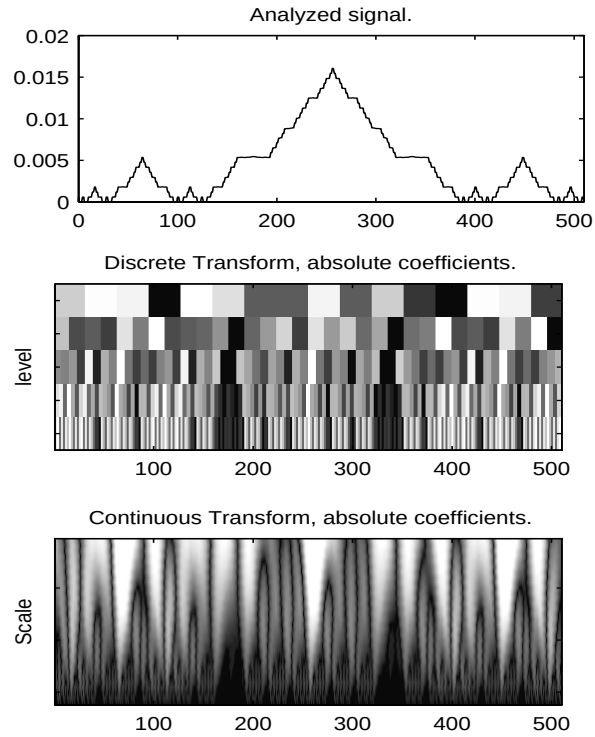


Figure 6-7: Continuous versus discrete transform

Using a redundant representation close to the so-called continuous analysis, instead of a non-redundant discrete time-scale representation, can be useful for analysis purposes. The non-redundant representation is associated with an orthonormal basis, whereas the redundant representation uses much more scale and position parameters than a basis. For a classical fractal signal, the redundant methods are quite accurate.

- **Graphic representation of continuous analysis**: time is on the abscissa and on the ordinate the scale varies almost continuously between 2^1 and 2^5 by step 1 (down to up). Keep in mind that when a scale is small, only small details are analyzed, as in a geographical map.
- **Graphic representation of discrete analysis**: (in the middle of the figure) time is on the abscissa and on the ordinate the scale a is dyadic: 2^1, 2^2, 2^3, 2^4 and 2^5 (down to up). Each coefficient of level k is repeated 2^k times.

Local and Global Analysis

A small scale value permits us to perform a local analysis; a large scale value is used for a global analysis. Combining local and global is a useful feature of the method. Let us be a bit more precise about the local part and glance at the frequency domain counterpart.

Imagine that the analyzing function ϕ or ψ is zero outside of a domain U, which is contained in a disk of radius ρ: $\psi(u) = 0, \forall u \notin U$. The wavelet is localized. The signal *s* and the function ψ are then compared in the disk, taking into account only the *x* values in the disk. The signal values, which are located outside of this domain, do not influence the value of the coefficient $\int_R s(t)\psi(t)dt$. The same argument holds when ψ is translated to position *b* and the corresponding coefficient analyzes *s* around *b*. So this analysis is local.

The wavelets having a compact support are used in local analysis. This is the case for Haar and Daubechies wavelets, for example. The wavelets whose values are considered as very small outside a domain U can be used with caution, as if they were in fact actually zero outside U. Not every wavelet has a compact support. This is the case, for instance, of the Meyer wavelet.

The previous localization is temporal, and is useful in analyzing a temporal signal (or spatial signal if analyzing an image). A result (linked to the Heisenberg uncertainty principle) links the signal dispersion *f* and the dispersion of its Fourier transform $\hat{f}$, and therefore of the dispersion of ψ and $\hat{\psi}$. The product of these dispersions is always greater than a constant *c* (which does not depend on the signal, but only on the dimension of the space). So, it is impossible to reduce arbitrarily both time and frequency localization.

In the Fourier and spectral analysis, the basic function is $f(x) = \exp(i\omega x)$. This function is not a localized function. The support is R. Its Fourier transform $\hat{f}$ is a distribution concentrated at point ω. The function *f* is very poorly localized in time, but $\hat{f}$ is perfectly localized in frequency. The wavelets generate an interesting "compromise" on the supports, and this compromise differs from that of complex exponentials, sine, or cosine.

Synthesis: An Inverse Transform

In order to be efficient and useful, a method designed for analysis also has to be able to perform synthesis. The wavelet method achieves this.

The analysis starts from *s* and results in the coefficients *C*(*a*,*b*). The synthesis starts from the coefficients *C*(*a*,*b*) and reconstructs *s*. Synthesis is the reciprocal operation of analysis.

For signals of finite energy, there are two formulas to perform the inverse wavelet transform:

- Continuous synthesis:

$$s(t) = \frac{1}{K_\psi}\int_{R^+}\int_R C(a,b)\frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)\frac{da\ db}{a^2}$$

  where K_ψ is a constant depending on ψ.

- Discrete synthesis:

$$s(t) = \sum_{j\in Z}\sum_{k\in Z} C(j,k)\psi_{j,k}(t).$$

Details and Approximations

The equations for continuous and discrete synthesis are of considerable interest and can be read in order to define the detail at level *j*:

1 Let us fix *j* and sum on *k*. A detail D_j is nothing more than the function

$$D_j(t) = \sum_{k \in Z} C(j,k)\psi_{j,k}(t).$$

2 Now let us sum on *j*. The signal is the sum of all the details: $s = \sum_{j \in Z} D_j$.

The details have just been defined. Take a reference level called *J*. There are two sorts of details. Those associated with indices $j \le J$ correspond to the scales $a = 2^j \le 2^J$ which are the fine details. The others, which correspond to $j > J$, are the coarser details. We group these latter details into

$$A_J = \sum_{j > J} D_j$$

which defines what is called an approximation of the signal *s*. We have just created the details and an approximation. They are connected. The equality

$$s = A_J + \sum_{j \le J} D_j$$

signifies that *s* is the sum of its approximation A_J and of its fine details. From the previous formula, it is obvious that the approximations are related to one another by:

$$A_{J-1} = A_J + D_J.$$

The calculation of the approximation coefficients will be discussed later.

For an orthogonal analysis, in which the $\psi_{j,k}$ is an orthonormal family,

- A_J is orthogonal to D_J, D_{J-1}, D_{J-2}, ...,
- s is the sum of the two orthogonal signals: A_J and $\sum_{j \le J} D_j$,
- $D_j \perp D_k$ for $j \neq k$.
- the quality (in energy) of the approximation of s by A_J is $qual_J = \frac{\|A_J\|^2}{\|s\|^2}$,
- $qual_{J-1} = qual_J + \frac{\|D_J\|^2}{\|s\|^2}$.

The following table contains definitions of details and approximations.

Definition of the detail at level j	$D_j(t) = \sum_{k \in Z} C(j,k)\psi_{j,k}(t)$
The signal is the sum of its details	$s = \sum_{j \in Z} D_j$
The approximation at level J	$A_J = \sum_{j > J} D_j$
Link between A_{J-1} and A_J	$A_{J-1} = A_J + D_J$
Several decompositions	$s = A_J + \sum_{j \le J} D_j$

From a graphical point of view, when analyzing a signal, it is always valuable to represent the different signals and coefficients.

Let us consider the Figure 6-8. The different signals that are presented exist in the same time grid. We can consider that the t index of detail $D_4(t)$, for example, that of an approximation $A_5(t)$ and that of the signal $s(t)$, identify the same temporal instant. This identity is of considerable practical interest in understanding the composition of the signal, even if the wavelet sometimes introduces dephasing.

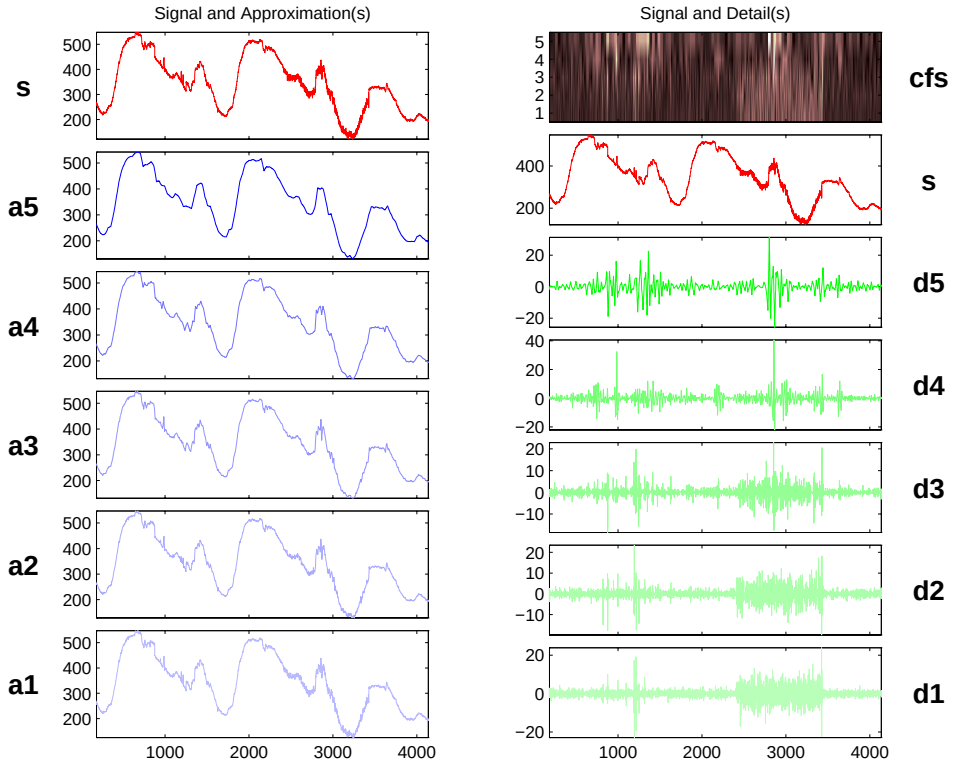


Figure 6-8: Approximations, details and coefficients

The Fast Wavelet Transform (FWT) Algorithm

In 1988, Mallat produced a fast wavelet decomposition and reconstruction algorithm [Mal89]. The Mallat algorithm for discrete wavelet transform (DWT) is, in fact, a classical scheme in the signal processing community, known as a two channel subband coder using conjugate quadrature filters or quadrature mirror filters (QMF).

- The decomposition algorithm starts with signal *s*, then calculates the coordinates of A_1 and D_1, then those of A_2 and D_2 and so on.
- The reconstruction algorithm called the inverse discrete wavelet transform (IDWT), starts from the coordinates of A_J and D_J then calculates the coordinates of A_{J-1}, then from the coordinates of A_{J-1} and D_{J-1} calculates those of A_{J-2} and so on.

Filters Used to Calculate the DWT and IDWT

For an orthogonal wavelet, in the multiresolution framework (see [Dau92] chap. 5), we start with the scaling function ϕ and the wavelet function ψ. One of the fundamental relations is the twin-scale relation (dilation equation or refinement equation):

$$\frac{1}{2}\phi\left(\frac{x}{2}\right) = \sum_{n \in Z} w_n \phi(x-n).$$

All the filters used in DWT and IDWT are intimately related to the sequence $(w_n)_{n \in Z}$. Clearly if ϕ is compactly supported, the sequence (w_n) is finite and can be viewed as a filter. The filter *W*, which is called the scaling filter (non-normalized), is:

- Finite Impulse Response (FIR)
- of length $2N$
- of sum 1
- of norm $\frac{1}{\sqrt{2}}$
- a low-pass filter

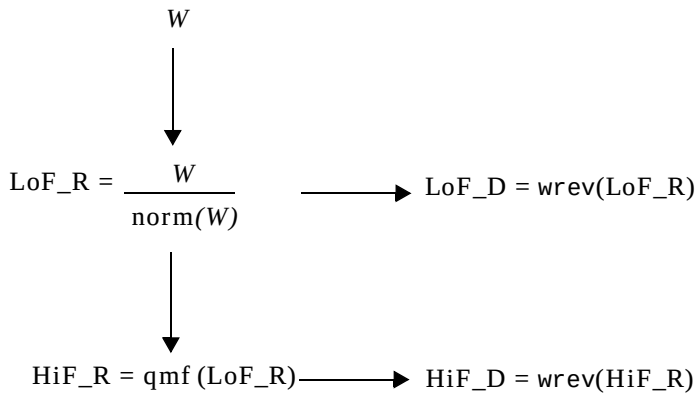
For example, for the db3 scaling filter:

```
load db3
db3

    db3 =

    0.2352   0.5706   0.3252  -0.0955  -0.0604   0.0249

sum(db3)

    ans =
         1.0000

norm(db3)
    ans =
         0.7071
```

From filter *W*, we define four FIR filters, of length 2*N* and of norm 1, organized as follows:

Filters	Low-pass	High-pass
Decomposition	LoF_D	HiF_D
Reconstruction	LoF_R	HiF_R

The four filters are computed using the following scheme:



where qmf is such that HiF_R and LoF_R are quadrature mirror filters (i.e., HiF_R(k) = (-1)^{k}LoF_R($2N$ - 1 - k)). Note that wrev flips the filter coefficients. So HiF_D an LoF_D are also quadrature mirror filters. The computation of these filters is performed using orthfilt. Let us illustrate these properties with the db6 wavelet. The plots associated with the following M-file are shown in the Figure 6-9.

```
% Load scaling filter.
load db6; w = db6;
subplot(421); stem(w); title('Original scaling filter');

% Compute the four filters.
[LoF_D,HiF_D,LoF_R,HiF_R] = orthfilt(w);
subplot(423); stem(LoF_D);
title('Decomposition low-pass filter');
subplot(424); stem(HiF_D);
title('Decomposition high-pass filter');
subplot(425); stem(LoF_R);
title('Reconstruction low-pass filter');
subplot(426); stem(HiF_R);
title('Reconstruction high-pass filter');

% High and low frequency illustration.
fftld = fft(LoF_D); ffthd = fft(HiF_D);
freq = [1:length(LoF_D)]/length(LoF_D);
subplot(427); plot(freq,abs(fftld));
title('Transfer modulus: low-pass')
subplot(428); plot(freq,abs(ffthd));
title('Transfer modulus: high-pass')
```

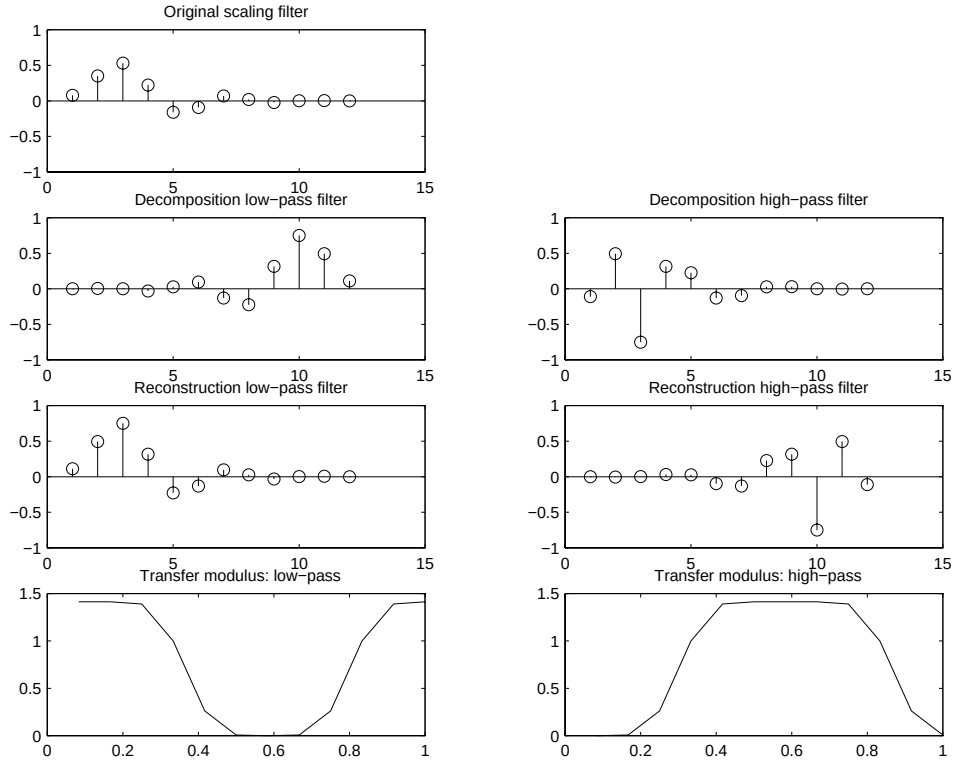
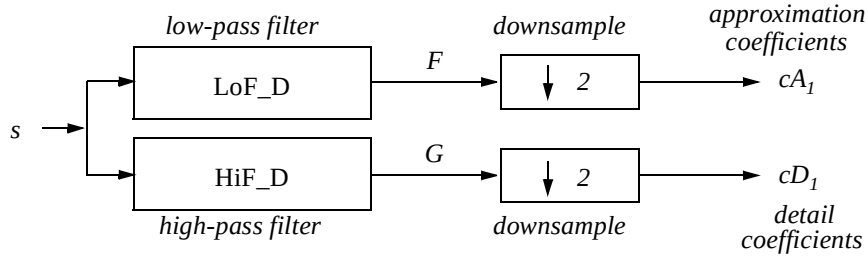


Figure 6-9: The four wavelet filters

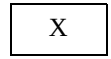
Algorithms

- Given a signal s of length N, the DWT consists of $\log_2 N$ stages at most. The first step produces, starting from s, two sets of coefficients: approximation coefficients cA_1 and detail coefficients cD_1. These vectors are obtained by convolving s with the low-pass filter LoF_D for approximation, and with the high-pass filter HiF_D for detail, followed by dyadic decimation.

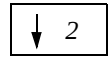
More precisely, the first step is:



where:



Convolve with filter X.



Keep the even indexed elements (see `dyaddown`).

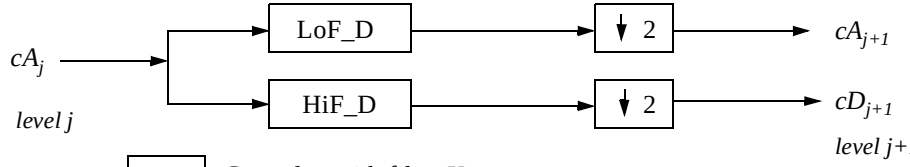
The length of each filter is equal to $2N$. If n = length(s), the signals F and G, are of length $n + 2N - 1$ and then the coefficients cA_1 and cD_1 are of length

$$\text{floor}\left(\frac{n-1}{2}\right) + N \quad .$$

The next step splits the approximation coefficients cA_1 in two parts using the same scheme, replacing s by cA_1, and producing cA_2 and cD_2, and so on.

One-Dimensional DWT

Decomposition step



where

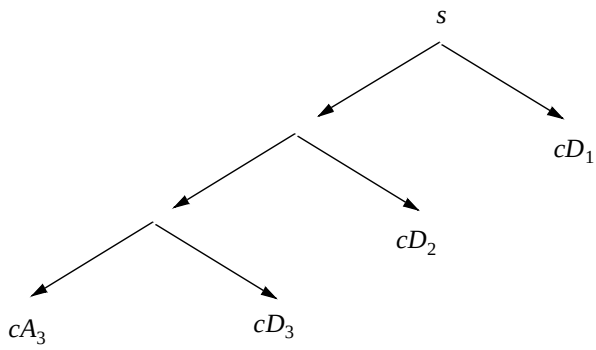
Convolve with filter X.

Downsample.

Initialization $cA_0 = s$.

So the wavelet decomposition of the signal s analyzed at level j has the following structure: $[cA_j, cD_j, \ldots, cD_1]$.

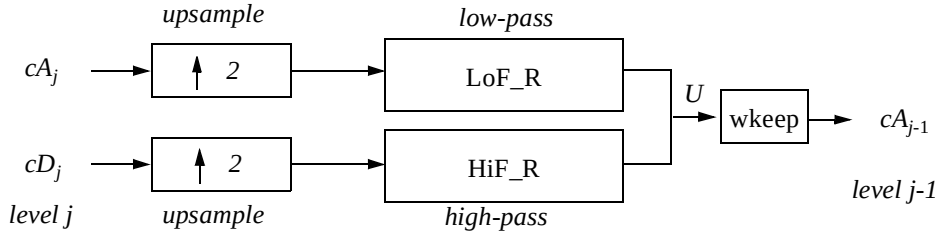
This structure contains for J=3, the terminal nodes of the following tree:



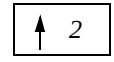
- Conversely, starting from cA_j and cD_j, the IDWT reconstructs cA_{j-1}, inverting the decomposition step by inserting zeros and convolving the results with the reconstruction filters.

One-Dimensional IDWT

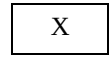
Reconstruction step



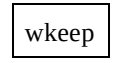
where:



Insert zeros at odd-indexed elements.



Convolve with filter X.



Take the central part of U with the convenient length.

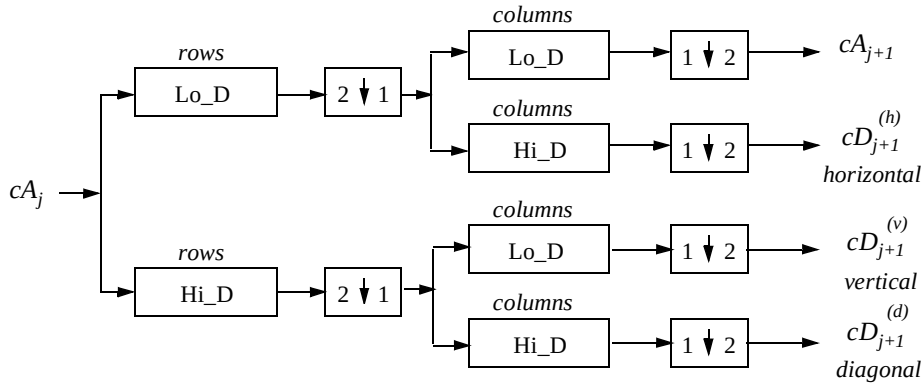
- For images, a similar algorithm is possible for two-dimensional wavelets and scaling functions obtained from one-dimensional wavelets by tensorial product.

  This kind of two-dimensional DWT leads to a decomposition of approximation coefficients at level j in four components: the approximation at level $j + 1$ and the details in three orientations (horizontal, vertical, and diagonal).

  The following charts describe the basic decomposition and reconstruction steps for images:

Two-Dimensional DWT

Decomposition step



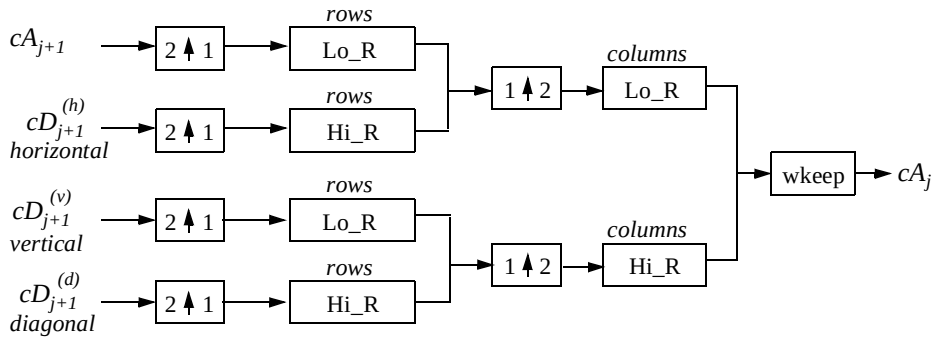
Where:

2 ↓ 1	Downsample columns: keep the even indexed columns.
1 ↓ 2	Downsample rows: keep the even indexed rows.
rows: X	Convolve with filter X the rows of the entry.
columns: X	Convolve with filter X the columns of the entry.

Initialization $CA_0 = s$ for the decomposition initialization.

Two-Dimensional IDWT

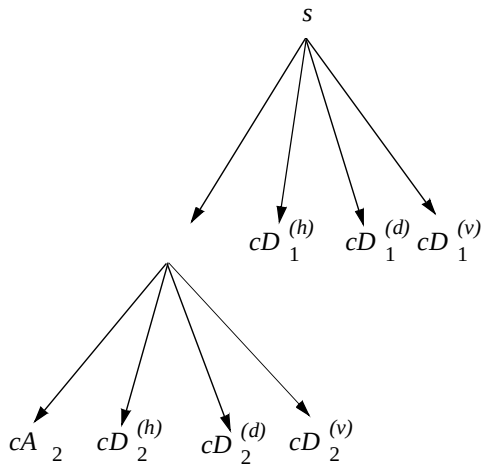
Reconstruction step



Where:

2 ↑ 1	Upsample columns: insert zeros at odd-indexed columns.
1 ↑ 2	Upsample rows: insert zeros at odd-indexed rows.
rows X	Convolve with filter X the rows of the entry.
columns X	Convolve with filter X the columns of the entry.

So, for $J = 2$, the two-dimensional wavelet tree has the form:



Finally, let us mention that, for biorthogonal wavelets, the same algorithms hold but the decomposition filters on one hand and the reconstruction filters on the other hand are obtained from two distinct scaling functions associated with two multiresolution analyses in duality.

In this case, the filters for decomposition and reconstruction are, in general, of different odd lengths. This situation occurs, for example, for "splines" biorthogonal wavelets used in the toolbox. By zero-padding, the four filters can be extended in such a way that they will have the same even length.

Why Does Such an Algorithm Exist?

Let us denote h = LoF_R and g = HiF_R and focus on the one-dimensional case.

We first justify how to go from level j to level j+1, for the approximation vector. This is the main step of the decomposition algorithm for the computation of the approximations. The details are calculated in the same way using the filter g instead of filter h.

Let $(A_k^{(j)})_{k \in Z}$ be the coordinates of the vector A_j:

$$A_j = \sum_k A_k^{(j)} \phi_{j,k}$$

and $A_k^{(j+1)}$ the coordinates of the vector A_{j+1}:

$$A_{j+1} = \sum_k A_k^{(j+1)} \phi_{j+1,k}$$

$A_k^{(j+1)}$ is calculated using the formula:

$$A_k^{(j+1)} = \sum_n h_{n-2k} A_n^{(j)}$$

This formula resembles a convolution formula.

The computation is very simple. Let us define $\tilde{h}(k) = h(-k)$, and

$$F_k^{(j+1)} = \sum_n \tilde{h}_{k-n} A_n^{(j)}$$

We obtain:

$$A_k^{(j+1)} = F_{2k}^{(j+1)}$$

We have to take the even index values of F. This is downsampling.

The initialization is carried out using $A_k^{(0)} = s(k)$ where $s(k)$ is the signal value at time k.

There are several reasons for this surprising result, all of which are linked to the multiresolution situation and to a few of the properties of the functions $\phi_{j,k}$ and $\psi_{j,k}$.

Let us now describe some of them.

1. The family $(\phi_{0,k}, k \in Z)$ is formed of orthonormal functions. As a consequence for any j, the family $(\phi_{j,k}, k \in Z)$ is orthonormal.
2. The double indexed family $(\psi_{j,k}, j \in Z, k \in Z)$ is orthonormal.
3. For any j, the $(\phi_{j,k}, k \in Z)$ are orthogonal to $(\psi_{j',k}, j' \le j, k \in Z)$
4. Between two successive scales, we have a fundamental relation, called the "twin-scale relation":

Twin-scale relation for ϕ	
$\phi_{1,0} = \sum_{k \in Z} h_k \phi_{0,k}$	$\phi_{j+1,0} = \sum_{k \in Z} h_k \phi_{j,k}$

This relation introduces the algorithm's *h* filter ($h_n = \sqrt{2}w_n$), see the section "Filters Used to Calculate the DWT and IDWT" on page 6-21).

5 We check that:

a. the coordinate of $\phi_{j+1,0}$ on $\phi_{j,k}$ is h_k and does not depend on *j*,

b. the coordinate of $\phi_{j+1,n}$ on $\phi_{j,k}$ is equal to: $\langle \phi_{j+1,n}, \phi_{j,k} \rangle = h_{k-2n}$.

6 These relations supply the ingredients for the algorithm.

7 Up to now we used the filter *h*. The high-pass filter *g* is used in the twin scales relation linking the ψ and ϕ functions. Between two successive scales, we have the following twin-scale fundamental relation.

Twin-scale relation between ψ and ϕ	
$\psi_{1,0} = \sum_{k \in Z} g_k \phi_{0,k}$	$\psi_{j+1,0} = \sum_{k \in Z} g_k \phi_{j,k}$

8 We justify now the reconstruction algorithm by building it. Let us simplify the notation, starting from A_1 and D_1, let us study $A_0 = A_1 + D_1$. The procedure is the same to calculate $A_j = A_{j+1} + D_{j+1}$.

Let us define α_n, δ_n, α_k^0 by:

$$A_1 = \sum_n \alpha_n \phi_{1,n}, \qquad D_1 = \sum_n \delta_n \psi_{1,n}, \qquad A_0 = \sum_k \alpha_k^0 \phi_{0,k},$$

Let us assess the α_k^0 coordinates as:

$$\begin{aligned} \alpha_k^0 = \langle A_0, \phi_{0,k} \rangle &= \langle A_1 + D_1, \phi_{0,k} \rangle = \langle A_1, \phi_{0,k} \rangle + \langle D_1, \phi_{0,k} \rangle \\ &= \sum_n \alpha_n \langle \phi_{1,n}, \phi_{0,k} \rangle + \sum_n \delta_n \langle \psi_{1,n}, \phi_{0,k} \rangle \\ &= \sum_n \alpha_n h_{k-2n} + \sum_n \delta_n g_{k-2n} \end{aligned}$$

We will focus our study on the first sum $\sum_n \alpha_n h_{k-2n}$; the second sum

$\sum_n \delta_n g_{k-2n}$ is handled in a similar manner. The calculations are easily organized if we note that (taking $k = 0$ in the previous formulas, makes things simpler):

$$\sum_n \alpha_n h_{-2n} = \ldots + \alpha_{-1} h_2 + \alpha_0 h_0 + \alpha_1 h_{-2} + \alpha_2 h_{-4} + \ldots$$
$$= \ldots + \alpha_{-1} h_2 + 0 h_1 + \alpha_0 h_0 + 0 h_{-1} + \alpha_1 h_{-2} + 0 h_{-3} + \alpha_2 h_{-4} + \ldots$$

If we transform the (α_n) sequence into a new sequence $(\tilde{\alpha}_n)$ defined by ..., α_{-1}, 0, α_0, 0, α_1, 0, α_2, 0, ... or

$$\tilde{\alpha}_{2n} = \tilde{\alpha}_n, \tilde{\alpha}_{2n+1} = 0$$

Then:

$$\sum_n \alpha_n h_{-2n} = \sum_n \tilde{\alpha}_n h_{-n}$$

and by extension:

$$\sum_n \alpha_n h_{k-2n} = \sum_n \tilde{\alpha}_n h_{k-n}$$

Since $\alpha_k^0 = \sum_n \tilde{\alpha}_n h_{k-n} + \sum_n \tilde{\delta}_n g_{k-n}$ the procedure thus becomes:

- Replace the α and δ sequences by upsampled versions $\tilde{\alpha}$ and $\tilde{\delta}$ inserting zeros
- Filter by h and g respectively
- Sum the obtained sequences

These are exactly the reconstruction steps.

One-Dimensional Wavelet Capabilities

The basic one-dimensional objects are:

	Objects	Description
Signal in original time	s	Original signal
	A_k, $0 \le k \le j$	Approximation at level k
	D_k, $1 \le k \le j$	Detail at level k
Coefficients in scale-related time	cA_k, $1 \le k \le j$	Approximation coefficients at level k
	cD_k, $1 \le k \le j$	Detail coefficients at level k
	$[cA_j, cD_j, ..., cD_1]$	Wavelet decomposition at level j, $j \ge 1$

The analysis-decomposition capabilities are:

Purpose	Input	Output	M-file
Single-level decomposition	s	cA_1, cD_1	`dwt`
Single-level decomposition	cA_j	cA_{j+1}, cD_{j+1}	`dwt`
Decomposition	s	$[cA_j, cD_j, ..., cD_1]$	`wavedec`

The synthesis-reconstruction capabilities are:

Purpose	Input	Output	M-file
Single-level reconstruction	cA_1, cD_1	s or A_0	`idwt`
Single-level reconstruction	cA_{j+1}, cD_{j+1}	cA_j	`idwt`
Full reconstruction	$[cA_j, cD_j, ..., cD_1]$	s or A_0	`waverec`
Selective reconstruction	$[cA_j, cD_j, ..., cD_1]$	A_l, D_m	`wrcoef`
Single reconstruction	cA_j (or cD_j)	A_j (or D_j)	`upcoef`

The decomposition structure utilities are:

Purpose	Input	Output	M-file
Extraction of detail coefficients	$[cA_j, cD_j, ..., cD_1]$	cD_k, $1 \le k \le j$	`detcoef`
Extraction of approximation coefficients	$[cA_j, cD_j, ..., cD_1]$	cA_k, $0 \le k \le j$	`appcoef`
Recomposition of the decomposition structure	$[cA_j, cD_j, ..., cD_1]$	$[cA_k, cD_k, ..., cD_1]$ $1 \le k \le j$	`upwlev`

Let us illustrate the command line mode for one-dimensional capabilities:

```
% Load original 1D signal.

load leleccum; s = leleccum(1:3920);
ls = length(s);

% Perform one step decomposition
% of s using db1.

[ca1,cd1] = dwt(s,'db1');
```

Results are displayed in Figure 6-10

```
% Perform one step reconstruction of
% ca1 and cd1.

a1 = upcoef('a',ca1,'db1',1,ls);
d1 = upcoef('d',cd1,'db1',1,ls);
```

Results are displayed in Figure 6-11.

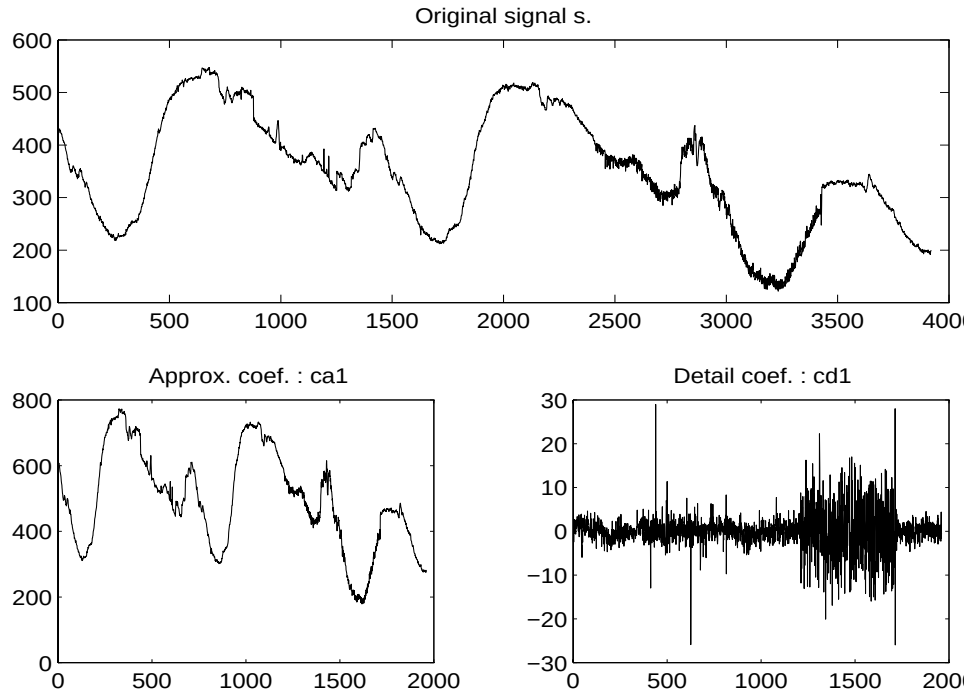


Figure 6-10: Coefficients at level 1

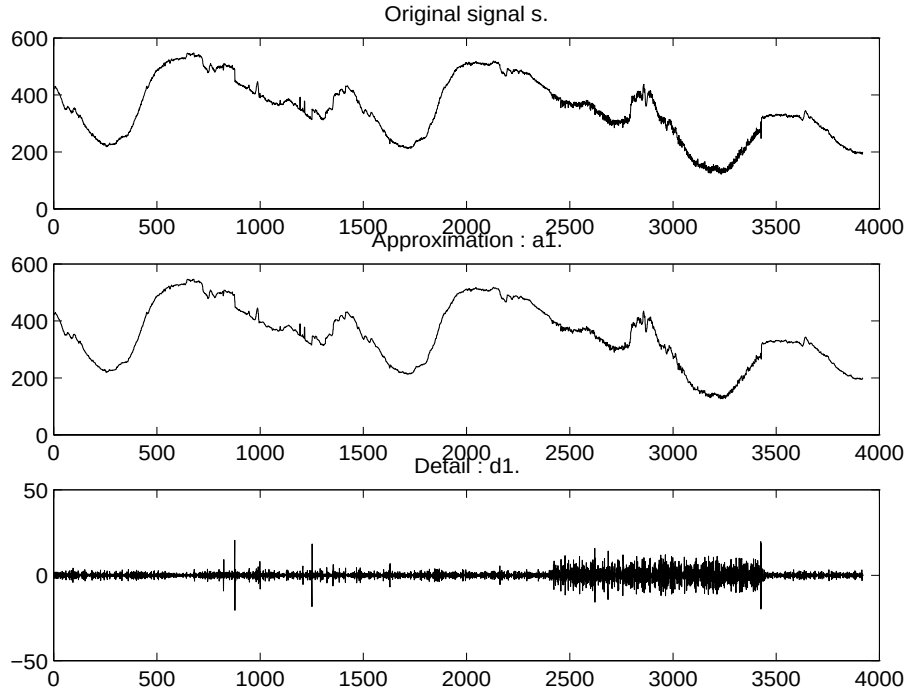


Figure 6-11: Signals at level 1

```
% Invert direct decomposition of s
% using coefficients.

a0 = idwt(ca1,cd1,'db1',ls);

% Perform decomposition at level 3
% of s using db1.

[c,l] = wavedec(s,3,'db1');

% Extract the approximation coefficients
% at level 3, from the wavelet decomposition
% structure [c,l].
```

```
ca3 = appcoef(c,l,'db1',3);

% Extract the detail coefficients at levels
% 1, 2 and 3, from the wavelet decomposition
% structure [c,l].

cd3 = detcoef(c,l,3);
cd2 = detcoef(c,l,2);
cd1 = detcoef(c,l,1);
```

Results are displayed in Figure 6-12, the signal s, `ca3`, `cd3`, `cd2` and `cd1` from the top to the bottom.

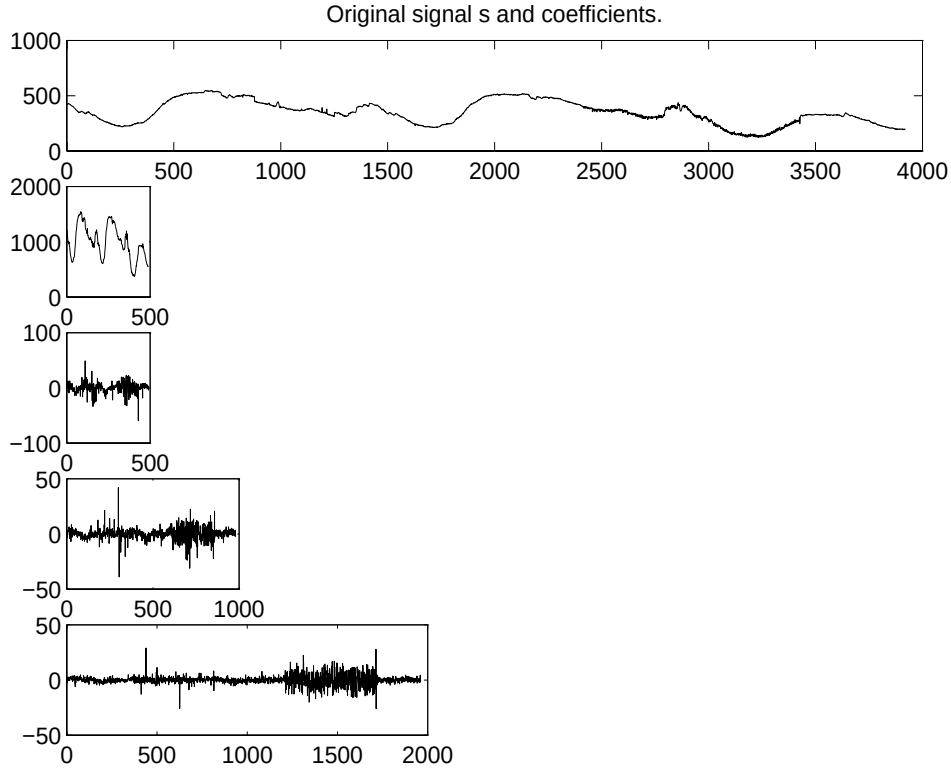


Figure 6-12: Coefficients at levels 1 to 3

```
% Reconstruct the approximation at level 3,
% from the wavelet decomposition
% structure [c,l].

a3 = wrcoef('a',c,l,'db1',3);

% Reconstruct the detail at
% level 2, from the wavelet
% decomposition structure [c,l].

d2 = wrcoef('d',c,l,'db1',2);

% Reconstruct s from the wavelet
% decomposition structure [c,l], s = a0.

a0 = waverec(c,l,'db1');
```

Two-Dimensional Wavelet Capabilities

The basic two-dimensional objects are:

	Objects	Description
Image in original resolution	s	Original image
	A_0	Approximation at level 0
	A_k, $1 \le k \le j$	Approximation at level k
	D_k, $1 \le k \le j$	Details at level k
Coefficients in scale-related resolution	cA_k, $1 \le k \le j$	Approximation coefficients at level k
	cD_k, $1 \le k \le j$	Detail coefficients at level k
	$[cA_j, cD_j, ..., cD_1]$	Wavelet decomposition at level j

D_k stands for $[D_k^{(h)}, D_k^{(v)}, D_k^{(d)}]$, the horizontal, vertical and diagonal details at level k.

The same holds for cD_k which stands for: $[cD_k^{(h)}, cD_k^{(v)}, cD_k^{(d)}]$.

The two-dimensional M-files are exactly the same as in one-dimensional case, appending a `2` on the end of the command. For example, `idwt` becomes `idwt2`.

Let us illustrate the command line mode for two-dimensional capabilities:

```
% Load original image.
load woman;
sX = size(X);
% X contains the loaded image and
% map contains the loaded colormap.
row = sX(1); col = sX(2);

% Image coding.
nbcol = size(map,1);
cod_X = wcodemat(X,nbcol);

% Perform one step decomposition
% of X using db1.
[ca1,chd1,cvd1,cdd1] = dwt2(X,'db1');

% Images coding.
cod_ca1 = wcodemat(ca1,nbcol);
cod_chd1 = wcodemat(chd1,nbcol);
cod_cvd1 = wcodemat(cvd1,nbcol);
cod_cdd1 = wcodemat(cdd1,nbcol);
dec2d = [...
        cod_ca1, cod_chd1; ...
        cod_cvd1, cod_cdd1 ...
        ];

% Visualize the coefficients of the decomposition
% at level 1.
```

Results are displayed in Figure 6-13.

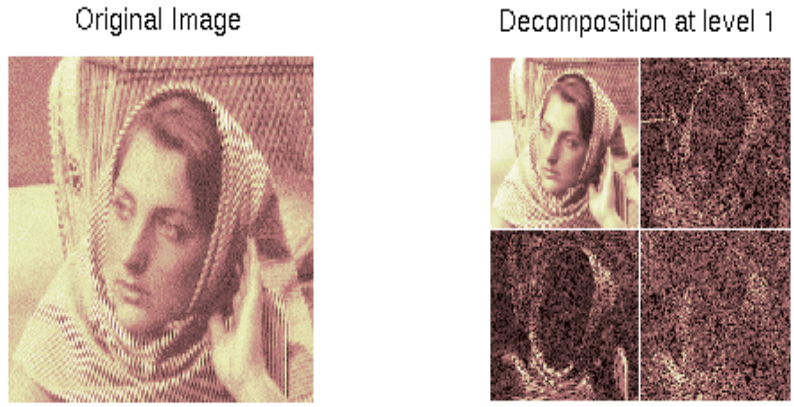


Figure 6-13: Decomposition at level 1

```
% Perform second step decomposition:
% decompose approx. cfs of level 1.
[ca2,chd2,cvd2,cdd2] = dwt2(ca1,'db1');

% Invert directly decomposition of X
% using coefficients at level 1.
a0 = idwt2(ca1,chd1,cvd1,cdd1,'db1',sX);

% Perform decomposition at level 2
% of X using db1.
[c,s] = wavedec2(X,2,'db1');
```

Results are displayed in Figure 6-14.

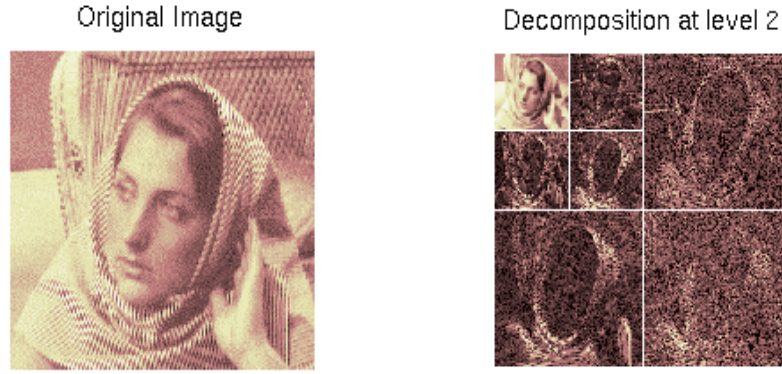


Figure 6-14: Decomposition at level 2

```
% Extract approximation coefficients
% at level 2, from wavelet decomposition
% structure [c,s].
ca2 = appcoef2(c,s,'db1',2);

% Extract details coefficients at level 2
% from wavelet decomposition
% structure [c,s].
chd2 = detcoef2('h',c,s,2);
cvd2 = detcoef2('v',c,s,2);
cdd2 = detcoef2('d',c,s,2);

% Extract approximation and details coefficients
% at level 1, from wavelet decomposition
% structure [c,s].
ca1 = appcoef2(c,s,'db1',1);
chd1 = detcoef2('h',c,s,1);
cvd1 = detcoef2('v',c,s,1);
cdd1 = detcoef2('d',c,s,1);
```

```
% Reconstruct approximation at level 2,
% from the wavelet decomposition
% structure [c,s].
a2 = wrcoef2('a',c,s,'db1',2);

% Reconstruct details at level 2,
% from the wavelet decomposition
% structure [c,s].
hd2 = wrcoef2('h',c,s,'db1',2);
vd2 = wrcoef2('v',c,s,'db1',2);
dd2 = wrcoef2('d',c,s,'db1',2);

% One step reconstruction of wavelet
% decomposition structure [c,s].
sc = size(c)
    sc =
       1 65536

val_s = s

    val_s =
         64    64
         64    64
        128   128
        256   256

[c,s] = upwlev2(c,s,'db1'); sc = size(c)

    sc =
      1 65536

val_s = s
    val_s =
       128 128
       128 128
       256 256
```

```
% Reconstruct approximation and details
% at level 1, from coefficients.
%
% step 1: extract coefficients
% decomposition structure [c,s].
%
% step 2: reconstruct.

siz = s(size(s,1),:);
ca1 = appcoef2(c,s,'db1',1);
a1 = upcoef2('a',ca1,'db1',1,siz);
clear ca1

chd1 = detcoef2('h',c,s,1);
hd1 = upcoef2('h',chd1,'db1',1,siz);
clear chd1

cvd1 = detcoef2('v',c,s,1);
vd1 = upcoef2('v',cvd1,'db1',1,siz);
clear cvd1

cdd1 = detcoef2('d',c,s,1);
dd1 = upcoef2('d',cdd1,'db1',1,siz);
clear cdd1

% Reconstruct X from the wavelet
% decomposition structure [c,s].
a0 = waverec2(c,s,'db1');
```

Dealing with Border Distortion

Classically the DWT is defined for sequences with length of some power of two, and different ways of extending samples of other sizes are needed. Methods for extending the signal include: zero-padding, periodic extension, and boundary value replication (symmetrization).

The basic algorithm for the DWT is not limited to dyadic length and is based on a simple scheme; convolution and downsampling. As usual, when a convolution is performed on finite-length signals, border distortions arise.

Signal Extensions: Zero-Padding, Symmetrization, and Smooth Padding

In order to deal with border distortions, the border should be treated differently. Various methods are available to deal with this problem, referred as "wavelets on the interval" (see [CohDJV93]). These interesting constructions are effective in theory but are not entirely satisfactory from a practical viewpoint.

Often it is preferable to use simple schemes based on signal extension on the boundaries. This involves the computation of a few extra coefficients at each stage of the decomposition process in order to get a perfect reconstruction. Details about the rationale of these schemes can be found in Chapter 8 of the book *Wavelets and Filter Banks*, by Strang and Nguyen.

The available signal extension modes are: (see `dwtmode`)

- **zero-padding**: This method is used in the version of the DWT given in the previous sections and assumes that the signal is zero outside the original support. It is the default mode of the wavelet transform in the toolbox.

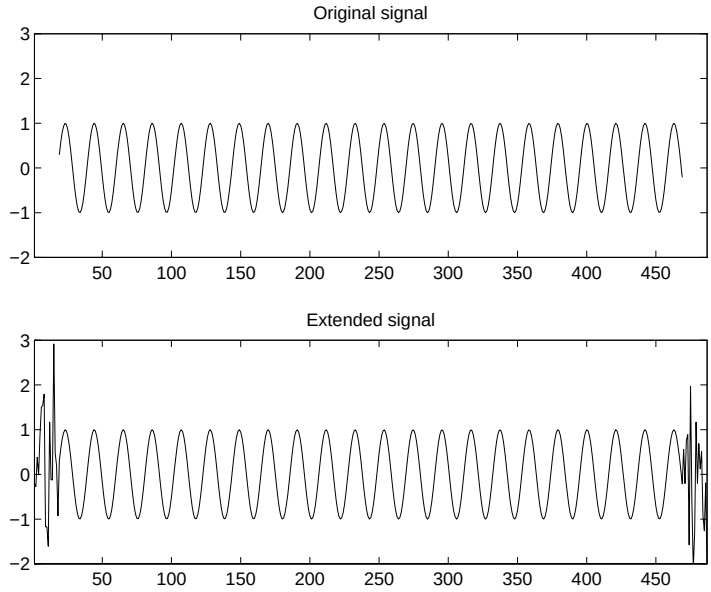
  The disadvantage of zero-padding is that discontinuities are artificially created at the border.
- **symmetrization**: This method assumes that signals or images can be recovered outside their original support by symmetric boundary value replication.

  Symmetrization has the disadvantage of artificially creating discontinuities of the first derivative at the border, but this method works well in general for images.
- **smooth padding**: This method assumes that signals or images can be recovered outside their original support by a simple first order derivative extrapolation. Smooth padding works well in general for smooth signals.

Before looking at an illustrative example, note that the decomposition step with any of these three extension modes has the same inverse reconstruction step. So all the capabilities described in the previous paragraphs are available without any reference to the extension mode.

It is interesting to notice that if arbitrary extension is done, and decomposition performed using the convolution-downsampling scheme, perfect reconstruction is recovered using `idwt` or `idwt2`. This point is illustrated by the following example.

```
% Set initial signal and get filters.
    x = sin(0.3*[1:451]);
    w = 'db9';
    [LoF_D,HiF_D,LoF_R,HiF_R] = wfilters(w);
% In fact using a slightly redundant scheme, any signal
% extension strategy works well.
% For example use random padding.
```




```
    lx = length(x); lf = length(LoF_D);
    randn('seed',654);
    ex = [randn(1,lf) x randn(1,lf)];
    axis([1 lx+2*lf -2 3])
    subplot(211), plot(lf+1:lf+lx,x), title('Original signal')
    axis([1 lx+2*lf -2 3])
    subplot(212), plot(ex), title('Extended signal')
    axis([1 lx+2*lf -2 3])
% Decomposition.
    la = floor((lx+lf-1)/2);
    ar = wkeep(dyaddown(conv(ex,LoF_D)),la);
    dr = wkeep(dyaddown(conv(ex,HiF_D)),la);
% Reconstruction.
    xr = idwt(ar,dr,w,lx);
% Check perfect reconstruction.
    err0 = max(abs(x-xr))

err0 =

    3.0464e-11
```

Now let us illustrate the differences between the three methods both for 1-D and 2-D signals.

Zero-Padding.

Using the GUI we will examine the effects of zero-padding.

1. From the MATLAB prompt, type

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

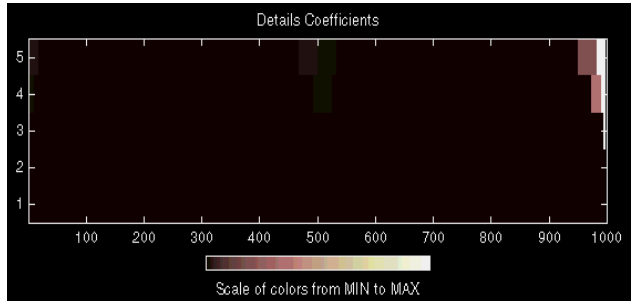
2. From the MATLAB prompt, type `wavemenu`.The Wavelet Toolbox Main Menu appears.

3. Click the **Wavelet 1-D** menu item.The discrete wavelet analysis tool for one-dimensional signal data appears.

4. From the **File** menu, choose the **Demo Analysis** option and select with db2 at level 5 --> two nearby discontinuities.

5. Select **Display Mode**: **Show and Scroll**

The detail coefficients clearly show the signal end effects.



Symmetric Extension.

6. From the MATLAB prompt, type

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

7. From the MATLAB prompt, type `wavemenu`.

   The Wavelet Toolbox Main Menu appears.

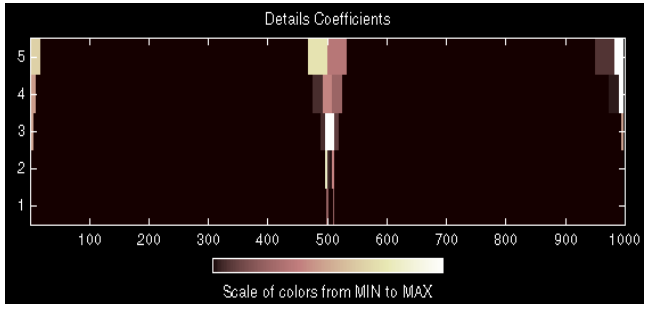
8 Click the **Wavelet 1-D** menu item.

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

9 From the **File** menu, choose the **Demo Analysis** option and select **with db2 at level 5 --> two nearby discontinuities**.

10 Select **Display Mode: Show and Scroll**

The detail coefficients show the signal end effects are present, but the discontinuities are well detected.



Smooth Padding.

11 From the MATLAB prompt, type

```
dwtmode('spd')
```

12 From the MATLAB prompt, type `wavemenu`.

The Wavelet Toolbox Main Menu appears.

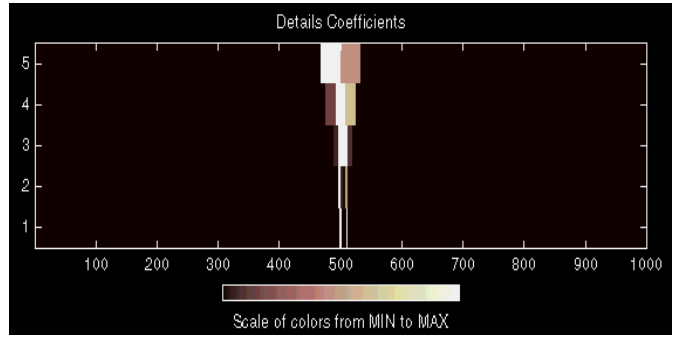
13 Click the **Wavelet 1-D** menu item.

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

14 From the **File** menu, choose the **Demo Analysis** option and select **with db2 at level 5 --> two nearby discontinuities**.

15 Select **Display Mode: Show and Scroll**

The detail coefficients show the signal end effects are not present, and the discontinuities are well detected.

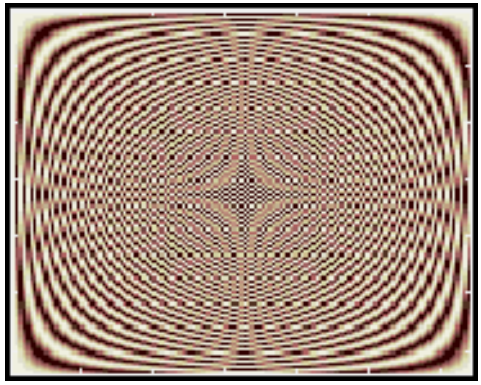


Let us now consider an image example.

Original Image.

1 From the MATLAB prompt, type

```
load geometry; sX = size(X);
% X contains the loaded image and
% map contains the loaded colormap.
row = sX(1); col = sX(2);
nbcol = size(map,1);
colormap(pink(nbcol));
image(wcodemat(X,nbcol));
```

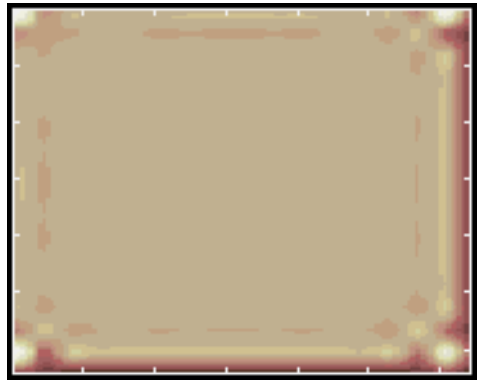


Zero-Padding.

Now we set the extension mode to zero-padding and perform a decomposition of the image to level 3 using the sym4 wavelet, and then reconstruct the approximation of level 3.

2 From the MATLAB prompt, type

```
dwtmode('zpd')
lev = 3;
[c,s] = wavedec2(X,lev,'sym4');
a = wrcoef2('a',c,s,'sym4',lev);
image(wcodemat(a,nbcol));
```

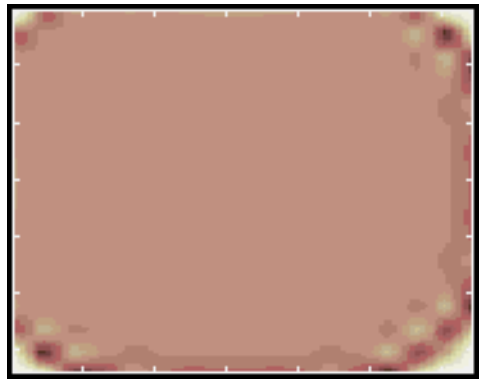


Symmetric Extension.

Now we set the extension mode to symmetric extension and perform a decomposition of the image again to level 3 using the sym4 wavelet, and then reconstruct the approximation of level 3.

3 From the MATLAB prompt, type

```
dwtmode('sym')
[c,s] = wavedec2(X,lev,'sym4');
a = wrcoef2('a',c,s,'sym4',lev);
image(wcodemat(a,nbcol));
```

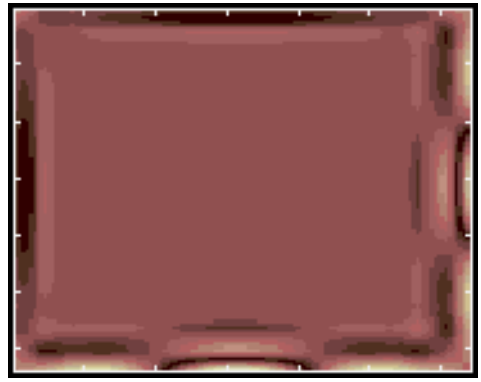


Smooth Padding.

Finally we set the extension mode to smooth padding and perform a decomposition of the image again to level 3 using the sym4 wavelet, and then reconstruct the approximation of level 3.

4 From the MATLAB prompt, type

```
dwtmode('spd')
[c,s] = wavedec2(X,lev,'sym4');
a = wrcoef2('a',c,s,'sym4',lev);
image(wcodemat(a,nbcol));
```



Periodized Wavelet Transform

Another method is the periodized wavelet transform. This method supposes that signals or images are periodic. It is clear that in general it is far from a reasonable assumption. The main advantage of this transform is that it does not require extra coefficients.

In the toolbox, the periodized wavelet transform is handled separately (see `dwtper`, `dwtper2`, `idwtper`, `idwtper2`). For the periodized wavelet transform, the full command line capabilities described previously are not defined and the GUI tools do not support the periodized wavelet transform.

Frequently Asked Questions

Continuous or Discrete Analysis?

When is continuous analysis more appropriate than discrete analysis? To answer this, consider the related questions: Do you need to know all values of a continuous decomposition to reconstruct the signal *s* exactly? Can you perform non-redundant analysis?

When the energy of the signal is finite, not all values of a decomposition are needed to exactly reconstruct the original signal, provided that you are using a wavelet that satisfies some admissibility condition (see [Dau92] p. 7, 24, 27). Usual wavelets satisfy this condition. In that case, a continuous-time signal *s* is entirely characterized by the knowledge of the discrete transform $C(j, k), (j,k) \in Z^2$. In such cases, discrete analysis is sufficient and continuous analysis is redundant. When the signal is recorded in continuous time or on a very fine time grid, both types of analysis are possible. Which should be used? The answer is: each has its own advantages.

- Discrete analysis ensures space-saving coding and is sufficient for the synthesis.
- Continuous analysis is often easier to interpret, since its redundancy tends to reinforce the traits and makes all information more visible. This is especially true of very subtle information. The analysis gains in "readability" and in ease of interpretation what it loses in terms of space saving.

Why Are Wavelets Useful for Space-Saving Coding?

The family of functions $(\phi_{0,k};\psi_{j,l})\ j \le 0,\ k, l \in Z$, used for the analysis is an orthogonal basis, therefore leading to non-redundancy: $\phi_{0,k} \perp \psi_{j',k'}$ as soon as $j' \le 0$, and $\psi_{j,k} \perp \psi_{j',k'}$ as soon as $(j,k) \ne (j',k')$. Let us remember that $u \perp v$ stands for $\int_R u(x)v(x)dx = 0$, for one dimensional signals.
For biorthogonal wavelets, the idea is similar.

Why Do All Wavelets Have Zero Average and Sometimes Several Vanishing Moments?

When the wavelet's $k + 1$ moments are equal to zero ($\int_R t^j \psi(t)dt = 0$ for $j = 0, \ldots, k$) all the polynomial signals $s(t) = \sum_{0 \le j \le k} a_j t^j$ have zero wavelet coefficients, the details are also zero. This property ensures the suppression of signals that are polynomials.

What About the Regularity of a Wavelet ψ?

The notion of regularity has been assuming increasing importance in theoretical and practical studies. Wavelets are tools used to study regularity and to conduct local studies. Deterministic fractal signals or Brownian motion trajectories are locally very irregular; for example, the latter are continuous signals, but their first derivative exists almost nowhere.

The definition of the concept of regularity is somewhat technical. To make things simple, let us say that a signal *f, defined on R,* has a regularity of *s.*

When *s* is an integer, the regularity in x_0 is defined as usual, *s* is the order of differentiability. When *s* is not an integer, let *m be the integer such that* $m < s < m + 1$, then *f* has a regularity of *s* in x_0 if its derivative $f^{(m)}$ of order *m* resembles $|x - x_0|^{s-m}$ locally around x_0.

The regularity of *f* in a domain is that of its least regular point.

The greater *s,* the more regular the signal.

The regularity of certain wavelets is known. The following table gives some indications for Daubechies wavelets.

ψ	***db*1 = Haar**	***db*2**	***db*3**	***db*4**	***db*5**	***db*7**	***db*10**
Regularity	0	0.5	0.91	1.27	1.59	2.15	2.90

We have an asymptotic relation linking the size of the support of the Daubechies wavelets *dbN* and their regularity: when $N \to \infty$,

length(support) = 2*N*, regularity $s \approx \frac{N}{5}$.

The functions are more regular at certain points than at others (see Figure 6-15).

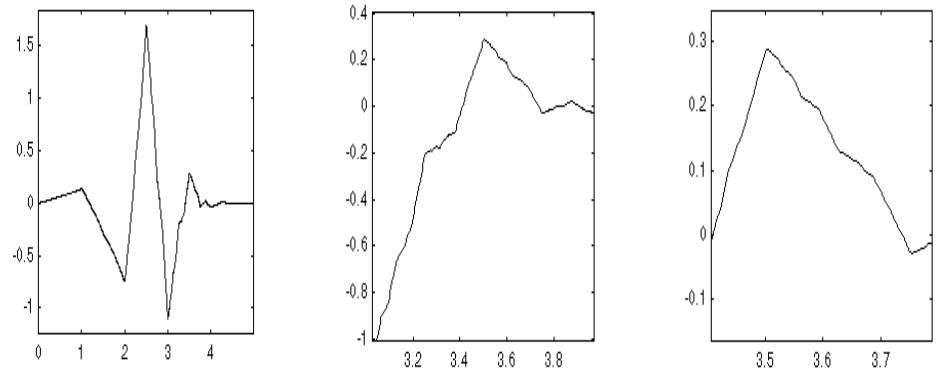


Figure 6-15: Zooming in on *db*3 wavelet

Selecting a regularity and a wavelet for this regularity is useful in estimations of the local properties of functions or signals. This can be used, for example, to make sure that a signal has a constant regularity at all points. Work on function estimation and nonlinear regression is currently underway, notably by Donoho, Johnstone, Kerkyacharian and Picard, in order to adapt the statistical estimators to unknown regularity. See also the remarks by I. Daubechies (see [Dau92] p. 301).

From a practical point of view, these questions arise in the world of finance in dealing with monetary and stock markets for fine studies of very fast transactions.

Are Wavelets Useful in Fields Other Than Signal or Image Processing?

- From a theoretical point of view, wavelets can be used to characterize large sets of mathematical functions and are used in the study of operators linked to partial differential equations.
- From a practical point of view, wavelets are used in several fields of numerical analysis, making certain complex calculations easier to handle or more precise.

What Functions Are Candidates to Be a Wavelet?

If a function f is continuous, has null moments, decreases quickly towards 0 when x tends towards infinity, or is null outside a segment of R, it is a likely candidate to become a wavelet. The family of shifts and dilations of f allows all finite energy signals to be reconstructed using the details in all scales. Such a function will be called ψ. This allows only continuous analysis.

In the toolbox, the ψ wavelet is usually associated with a scaling function ϕ. There are, however, some ψ wavelets for which we do not know how to associate a ϕ. In some cases we know how to prove that ϕ does not exist, for example, the Morlet wavelet.

Is It Easy to Build a New Wavelet?

Not at the present time. More precisely, for a minimal requirement on the wavelet properties, it is easy but without interest. But if more interesting properties (like the existence of ϕ for example) are needed, then it is difficult.

Very few wavelets have an explicit analytical expression. Notable exceptions are wavelets that are piecewise polynomials (Haar, Battle-Lemarie, see [Dau92] p. 146), Morlet, or Mexican hat.

Wavelets, even `db2`, `db3` ..., are defined by functional equations. The solution for constructive equations is numerical, and is accomplished using a fairly simple algorithm.

The basic property is the existence of a linear relation between the two functions $\phi(x/2)$ and $\phi(x)$. Another relation of the same type links $\psi(x/2)$ to $\phi(x)$. These are the relations of the two scales, the twin-scale relation.

Indeed there are two sequences h and g of coefficients such that:

$h \in l^2(Z), g \in l^2(Z)$ and

$$\frac{1}{2}\phi\left(\frac{x}{2}\right) = \frac{1}{\sqrt{2}}\sum_{n \in Z} h_n \phi(x-n)$$

$$\frac{1}{2}\psi\left(\frac{x}{2}\right) = \frac{1}{\sqrt{2}}\sum_{n \in Z} g_n \phi(x-n).$$

By rewriting these formulas using Fourier transforms (expressed using a hat) we obtain:

$$\hat{\phi}(2\omega) = \frac{1}{\sqrt{2}}\hat{h}(\omega)\hat{\phi}(\omega) \qquad \hat{\psi}(2\omega) = \frac{1}{\sqrt{2}}\hat{g}(\omega)\hat{\phi}(\omega)$$

There are ϕ functions for which the h has a finite impulse response (FIR): there is only a finite number of nonzero h_n coefficients. The associated wavelets were built by I. Daubechies (see [Dau92] in Chapter 6) and are used extensively in the toolbox. The reader can refer to p. 164 and Chapter 10 of the book *Wavelets and Filter Banks*, by Strang and Nguyen.

What Is the Link Between Wavelet and Fourier Analysis?

Wavelet analysis complements the Fourier analysis for which there are several MATLAB functions: `fft`, `spa`, `etfe`, `spectrum`.

Fourier analysis uses the basic functions $\sin(\omega t)$, $\cos(\omega t)$, and $\exp(i\omega t)$, with ω being the frequency.

- In the frequency domain, these functions are perfectly localized, since their spectrum loads only two points $-\omega/2$, and $\omega/2$. The functions are suited to the analysis and synthesis of signals with a simple spectrum, which is very well localized in frequency, for example $\sin(\omega_1 t) + 0.5\sin(\omega_2 t) - \cos(\omega_3 t)$.
- In the time domain, these functions are not localized. It is difficult for them to analyze or synthesize complex signals presenting fast local variations such as transients or abrupt changes: the Fourier coefficients for a frequency ω will depend on all values in the signal. To limit the difficulties involved, it is possible to "window" the signal using a regular function, which is zero or nearly zero outside a time segment $[-m, m]$. We then build "a well localized slice" as I. Daubechies (see [Dau92] p. 2) calls it. The windowed-Fourier analysis coefficients are:

  $$\hat{s}(\omega,t) = \int_R s(u)g(t-u)e^{-i\omega u}du$$

  The analogy of this formula with that of the wavelet coefficients is obvious:

  $$C(a,t) = \int_R s(u)\left(\frac{1}{\sqrt{a}}\right)\psi\left(\frac{(t-u)}{a}\right)du$$

  The large values of a correspond to small values of ω.

The Fourier coefficient $\hat{s}(\omega,t)$ depends on the values of the signal *s* on the segment with a constant width [*t* - *m*, *t* + *m*]. If ψ, like *g*, is zero outside of [-*m*, *m*], the *C*(*a*,*t*) coefficients will depend on the values of the signal *s* on the segment of width 2*am*, which varies as a function of [*t* - *am*, *t* + *am*]. This slight difference solves several difficulties, allowing a kind of time-windowed analysis at various scales *a*.

The wavelets stay however competitive, even in contexts considered favorable for the Fourier technique. I. Daubechies (see [Dau92] p. 3-7) gives an example of "Windowed Fourier" processing and complex Morlet wavelet processing $\psi(t) = Ce^{-t^2/\alpha^2}(e^{i\pi t} - e^{-\pi^2\lambda^2/4})$, of a signal composed mainly of the sum of two sines. The wavelet analysis gives good results.

Wavelet Families: Additional Discussion

There are different types of wavelet families whose qualities vary according to several criteria. The main criteria are:

- The support of ψ, $\hat{\psi}$ and ϕ, $\hat{\phi}$: the speed of convergence at infinity to 0 of these functions when the time or the frequency goes to infinity, which quantifies both time and frequency localizations.
- The symmetry, which is useful in avoiding dephasing in image processing.
- The number of vanishing moments for ψ or for ϕ (if it exists), which is useful for compression purpose.
- The regularity, which is useful for getting nice features, like smoothness of the reconstructed signal or image.

These are associated with two properties that allow fast algorithm and space-saving coding:

- The existence of a scaling function ϕ.
- The orthogonality or the biorthogonality of the resulting analysis,

and perhaps less important ones:

- The existence of an explicit expression.
- The ease of tabulating.
- The familiarity with use.

Typing `waveinfo` in command line mode displays a survey of the main properties of all wavelet families available in the toolbox.

Let us mention that the ϕ and ψ functions can be computed using `wavefun`; the filters are generated using `wfilters`. We provide definition equations for several wavelets. Some are given explicitly by their time definition, others by their frequency definition, and still others by their filter.

The table below outlines the wavelet families included in the toolbox.

Wavelets in the toolbox	
`morl`	Morlet
`mexh`	Mexican hat
`meyr`	Meyer
`haar`	Haar
`dbN`	Daubechies
`symN`	Symlets
`coifN`	Coiflets
`biorNr.Nd`	Splines biorthogonal wavelets

Daubechies Wavelets: *dbN*

In *dbN*, *N* is the order. Some authors use 2*N* instead of *N*. More about this family can be found in [Dau92] p. 115, 132, 194, 242. By typing `waveinfo('db')` at the MATLAB command prompt, you can obtain a survey of the main properties of this family.

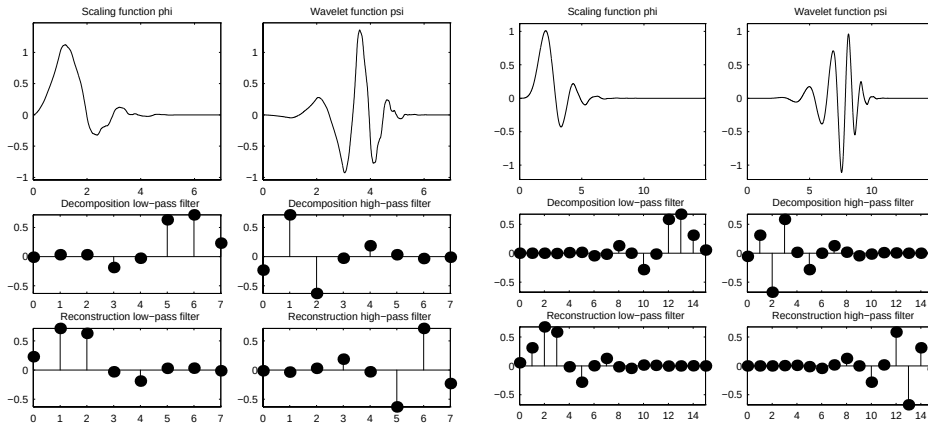


Figure 6-16: Daubechies wavelets db4 and db8

This family includes the Haar wavelet, written *db*1, the simplest wavelet imaginable and certainly the earliest. Using `waveinfo('haar')`, you can obtain a survey of the main properties of this wavelet.

Haar

$\psi(x) = 1,$	if	$0 \le x < \frac{1}{2}$
$\psi(x) = -1,$	if	$\frac{1}{2} \le x < 1$
$\psi(x) = 0,$	if	$x \notin [0, 1]$
$\phi(x) = 1$	if	$x \in [0, 1]$
$\phi(x) = 0$	if	$x \notin [0, 1]$

dbN

These wavelets have no explicit expression except for *db*1, which is the *Haar* wavelet. However, the square modulus of the transfer function of *h* is explicit and fairly simple.

- Let $P(y) = \sum_{k=0}^{N-1} C_k^{N-1+k} y^k$, where C_k^{N-1+k} denotes the binomial coefficients. Then:

$$|m_0(\omega)|^2 = \left(\cos^2\frac{\omega}{2}\right)^N P\left(\sin^2\left(\frac{\omega}{2}\right)\right)$$

  where:

$$m_0(\omega) = \frac{1}{\sqrt{2}} \sum_{k=0}^{2N-1} h_k e^{-ik\omega}$$

- The support length of ψ and ϕ is $2N - 1$. The number of vanishing moments of ψ is N.
- Most *dbN* are not symmetrical. For some, the asymmetry is very pronounced.
- The regularity increases with the order. When N becomes very large, ψ and ϕ belong to $C^{\mu N}$, since μ is approximately equal to 0.2. For sure, this asymptotic value is too pessimistic for small order N. Note that the functions are more regular at certain points than at others.
- The analysis is orthogonal.

Symlet Wavelets: *symN*

In *symN*, *N* is the order. Some authors use 2*N* instead of *N*. Symlets are only near symmetric; consequently some authors do not call them symlets. More about symlets can be found in [Dau92], p. 194, 254-257. By typing `waveinfo('sym')` at the MATLAB command prompt, you can obtain a survey of the main properties of this family.

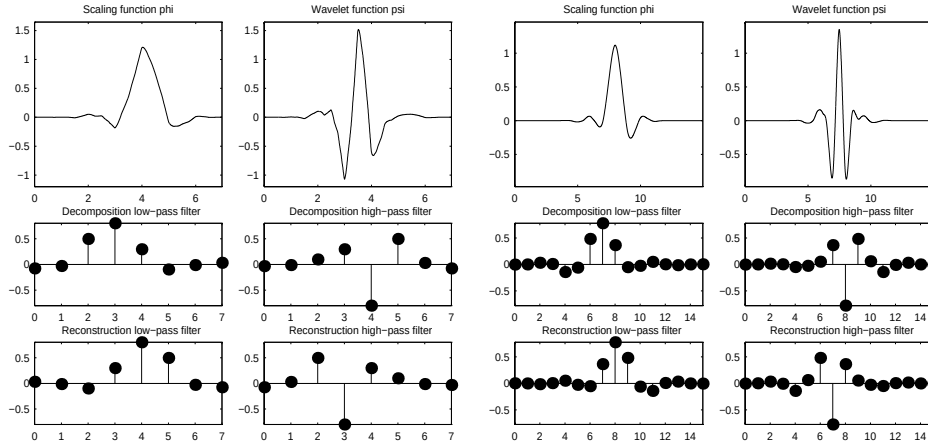


Figure 6-17: Symlets sym4 and sym8

Daubechies proposes modifications of her wavelets such that their symmetry can be increased while retaining great simplicity.

The idea consists of reusing the function m_0 introduced in the *dbN*, considering the $|m_0(\omega)|^2$ as a function W of $z = e^{i\omega}$. Then we can factor W in several different ways in the form of $W(z) = U(z)\overline{U\left(\frac{1}{\bar{z}}\right)}$. The roots of W with modulus not equal to 1 go in pairs. If one is z_1, $\frac{1}{z_1}$ is also a root.

- By selecting U such that the modulus of all its roots is strictly less than 1, we build Daubechies wavelets *dbN*. The U filter is a "minimum phase filter."
- By making another choice, we obtain more symmetrical filters; these are symlets.

The symlets have other properties similar to those of the *dbN*s.

Coiflet Wavelets: *coifN*

In *coifN*, *N* is the order. Some authors use 2*N* instead of *N*. For the coiflet construction, see [Dau92] p. 258-259. By typing `waveinfo('coif')` at the MATLAB command prompt, you can obtain a survey of the main properties of this family.

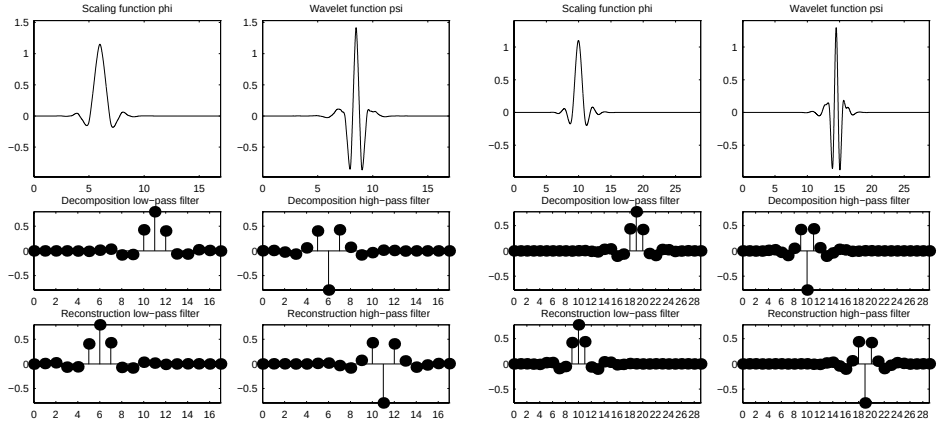


Figure 6-18: Coiflets coif3 and coif5.

Built by Daubechies at the request of Coifman, the function ψ has 2*N* moments equal to 0 and, what is more original, the function ϕ has 2*N*-1 moments equal to 0. The two functions have a support of length 6*N*-1.

The *coifN* ψ and ϕ are much more symmetrical than the *dbN*s. With respect to the support length, *coifN* has to be compared to *db*3*N* or *sym3N*; with respect to the number of vanishing moments of ψ, *coifN* has to be compared to *db*2*N* or *sym2N*.

If *s* is a smooth continuous time signal, for large *j*: the coefficient $\langle s, \phi_{j,k} \rangle \approx 2^{-j/2} s(2^j k)$ (see the "Mathematical Conventions" section at the beginning of this chapter). If *s* is a polynomial of degree *d*, $d \leq N$ -1 the approximation becomes an equality. This property is used, connected with sampling problems, when calculating the difference between an expansion over the $\phi_{j,k}$ of a given signal and its sampled version.

Biorthogonal Wavelet Pairs: *biorNr.Nd*

More about biorthogonal wavelets can be found in [Dau92] p. 259, 269-285 and [Coh92]. By typing `waveinfo('bior')` at the MATLAB command prompt, you can obtain a survey of the main properties of this family, as well as information about Nr and Nd orders and associated filter lengths.

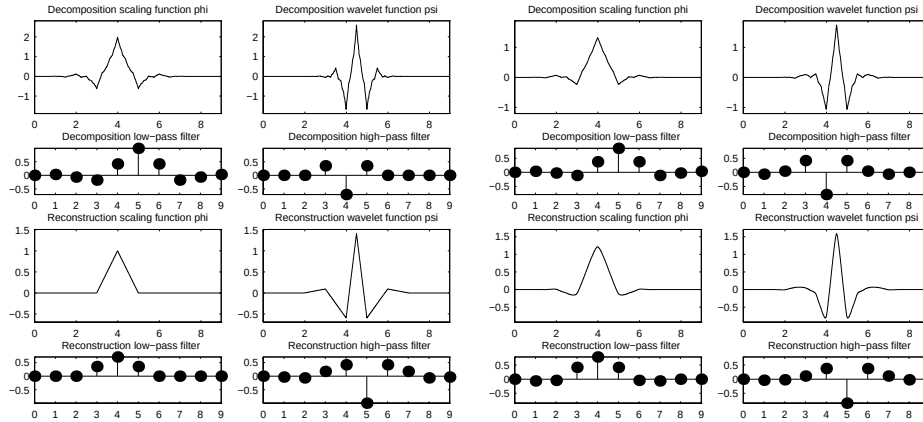


Figure 6-19: Biorthogonal wavelets bior2.4 and bior4.4

The new family extends the wavelet family. It is well known in the subband filtering community that symmetry and exact reconstruction are incompatible, if the same FIR filters are used for reconstruction and decomposition. Two wavelets, instead of just one, are introduced:

- One, $\tilde{\psi}$, is used in the analysis, and the coefficients of a signal s are

  $$\tilde{c}_{j,k} = \int s(x)\tilde{\psi}_{j,k}(x)dx$$

- The other, ψ, is used in the synthesis $s = \sum_{j,k} \tilde{c}_{j,k}\psi_{j,k}$.

In addition, the wavelets are related by duality in the following sense:

$\int \tilde{\psi}_{j,k}(x)\psi_{j',k'}(x)dx = 0$ as soon as $j \neq j'$ or $k \neq k'$ and even

$\int \tilde{\phi}_{0,k}(x)\phi_{0,k'}(x)dx = 0$ as soon as $k \neq k'$.

It becomes apparent, as Cohen pointed out in his thesis (see [Coh92] p. 110), that “the useful properties for analysis (e.g., oscillations, zero moments) can be concentrated on the $\tilde{\psi}$ function whereas the interesting properties for synthesis (regularity) are assigned to the ψ function. The separation of these two tasks proves very useful.”

$\tilde{\psi}$, ψ can have very different regularity properties, ψ being more regular than $\tilde{\psi}$ (see [Dau92] p. 269).

The $\tilde{\psi}$, ψ, $\tilde{\phi}$ and ϕ functions are zero outside of a segment.

The calculation algorithms are maintained and thus very simple.

The filters associated to m_0 and $\tilde{m}_0$ can be symmetrical. The functions used in the calculations are easier to build numerically than those used in the usual wavelets.

Meyer Wavelet: *meyr*

Both ψ and ϕ are defined in the frequency domain, starting with an auxiliary function ν (see [Dau92] p.117, 119, 137, 152). By typing `waveinfo('meyr')` at the MATLAB command prompt, you can obtain a survey of the main properties of this wavelet.

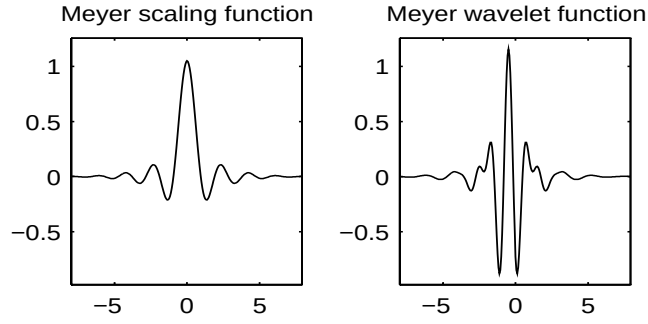


Figure 6-20: The Meyer wavelet

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

- $\hat{\psi}(\omega) = (2\pi)^{-1/2} e^{i\omega/2} \sin\left(\frac{\pi}{2}\nu\left(\frac{3}{2\pi}|\omega| - 1\right)\right)$ *if* $\frac{2\pi}{3} \le |\omega| \le \frac{4\pi}{3}$

  $\hat{\psi}(\omega) = (2\pi)^{-1/2} e^{i\omega/2} \cos\left(\frac{\pi}{2}\nu\left(\frac{3}{4\pi}|\omega| - 1\right)\right)$ *if* $\frac{4\pi}{3} \le |\omega| \le \frac{8\pi}{3}$

  and $\hat{\psi}(\omega) = 0$ *if* $|\omega| \notin [\frac{2\pi}{3}, \frac{8\pi}{3}]$

  where $\nu(a) = a^4(35 - 84a + 70a^2 - 20a^3)$, $a \in [0,1]$

- $\hat{\phi}(\omega) = (2\pi)^{-1/2}$ *if* $|\omega| \le \frac{2\pi}{3}$

  $\hat{\phi}(\omega) = (2\pi)^{-1/2} \cos\left(\frac{\pi}{2}\nu\left(\frac{3}{2\pi}|\omega| - 1\right)\right)$ *if* $\frac{2\pi}{3} \le |\omega| \le \frac{4\pi}{3}$

  $\hat{\phi}(\omega) = 0$ *if* $|\omega| > \frac{4\pi}{3}$

By changing the auxiliary function, one gets a family of different wavelets. For the required properties of the auxiliary function ν, see the list of references. This wavelet ensures orthogonal analysis.

The function ψ does not have finite support, but ψ decreases to 0 when $x \to \infty$, faster than any "inverse polynomial":

$\forall n \in \mathrm{N}, \exists C_n$ such that $|\psi(x)| \le C_n(1 + |x|^2)^{-n}$.

This property holds also for the derivatives:

$$\forall k \in N, \forall n \in N, \exists C_{k,n}, \text{ such that } \left|\psi^{(k)}x\right| \le C_{k,n}(1 + |x|^2)^{-n}.$$

The wavelet is infinitely differentiable.

Battle-Lemarie Wavelets

See [Dau92] p. 146-148, 151.

These wavelets are not included in the toolbox, but we use the spline functions in the biorthogonal family.

There are two forms of the wavelet; one does not ensure the analysis to be an orthogonal one, while the other does. For *N*=1, the scaling functions are linear splines. For *N*=2, the scaling functions are quadratic B-spline with finite support. More generally, for an *N*-degree B-splines:

$$\hat{\phi}(\omega) = (2\pi)^{-1/2} e^{-i\kappa\omega/2}\left[\frac{\sin(\omega/2)}{\omega/2}\right]^{N+1}$$

with $k = 0$ if N is odd, $k = 1$ if N is even. This formula can be used to build the filters.

The twin scale relation is:

$$\phi(x) = 2^{-2M}\sum_{j=0}^{2M+1} C_j^{2M+1}\phi(2x - M - 1 + j) \quad \text{if} \quad N = 2M$$

$$\phi(x) = 2^{-2M-1}\sum_{j=0}^{2M+2} C_j^{2M+2}\phi(2x - M - 1 + j) \text{ if } N = 2M + 1$$

- For an even N, ϕ is symmetrical around, $x = 1/2$; ψ is anti-symmetrical around $x = 1/2$. For an odd N, ϕ is symmetrical around $x = 0$; ψ is symmetrical around $x = 1/2$.
- The analysis becomes orthogonal if we transform the functions ψ and ϕ somewhat. For N=1, for instance, let:

$$\hat{\phi}^{\perp}(\omega) = 3^{1/2}(2\pi)^{-1/2}\frac{4\sin^2(\omega/2)}{\omega^2[1 + 2\cos^2(\omega/2)]^{1/2}}$$

- The supports of ψ and $\phi^{\perp}$ are not finite, but the decrease of the functions ψ and $\phi^{\perp}$ to 0 is exponential. See [Dau92] p. 151.
- The ψ functions have derivatives up to order *N-1*.

Mexican Hat Wavelet: *mexh*

See [Dau92] p. 75.

By typing `waveinfo('mexh')` at the MATLAB command prompt, you can obtain a survey of the main properties of this wavelet.

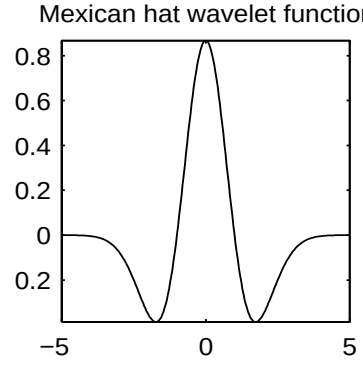


Figure 6-21: The Mexican hat

$$\psi(x) = \left(\frac{2}{\sqrt{3}}\pi^{-1/4}\right)(1-x^2)e^{-x^2/2}$$

This function is proportional to the second derivative function of the Gaussian probability density function.

As the ϕ function does not exist, the analysis is not orthogonal.

Morlet Wavelet: *morl*

See [Dau92] p. 76.

By typing `waveinfo('morl')` at the MATLAB command prompt you can obtain a survey of the main properties of this wavelet.

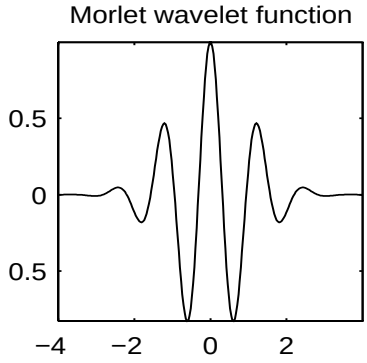


Figure 6-22: The Morlet wavelet

$$\psi(x) = Ce^{-x^2/2}\cos(5x)$$

The *C* constant is used for normalization in view of reconstruction.

As the ϕ function does not exist, the analysis is not orthogonal.

Summary of Wavelet Families and Associated Properties

	morl	**mexh**	**meyr**	**haar**	***db*N**	**symN**	**coifN**	**biorNr.Nd**
"Crude"	•	•						
Infinitely regular	•	•	•					
Compactly supported orthogonal				•	•	•	•	
Compactly supported biothogonal								•
Symmetry	•	•	•	•				•
Asymmetry					•			
Near symmetry						•	•	
Arbitrary number of vanishing moments					•	•	•	•
Vanishing moments for ϕ							•	
Arbitrary regularity					•	•	•	•
Existence of ϕ			•	•	•	•	•	•
Orthogonal analysis			•	•	•	•	•	
Biorthogonal analysis			•	•	•	•	•	•
Exact reconstruction			•	•	•	•	•	•
FIR filters				•	•	•	•	•
Continuous transform	•	•	•	•	•	•	•	•
Discrete transform			•	•	•	•	•	•
Fast algorithm				•	•	•	•	•
Explicit expression	•	•		•				for splines

Wavelet Applications: More Detail

Chapters 3 and 4 illustrate wavelet applications with examples and case studies. This section re-examines some of the applications with additional theory and more detail.

Suppressing Signals

As shown in Chapter 3, "Suppressing Signals," by suppressing a part of a signal the remainder may be highlighted.

Let ψ be a wavelet with at least k+1 vanishing moments (for j = 0, ..., k, $\int_R x^j \psi(x)dx = 0$).

If the signal s is a polynomial of degree k, then the coefficients $C(a,b) = 0$ for all a and all b. Such wavelets automatically suppress the polynomials. The degree of s can vary with time, provided that it remains less than k.

If s is now a polynomial of degree k on segment $[\alpha,\beta]$, then $C(a,b) = 0$ as long as the support of the function $\frac{1}{\sqrt{a}}\psi\left(\frac{x-b}{a}\right)$ is included in $[\alpha,\beta]$. The suppression is local. Effects will appear on the edges of the segment.

Likewise, let us suppose that, on $[\alpha,\beta]$ to which 0 belongs, we have the expansion $s(x) = s(0) + xs'(0) + x^2 s^{(2)}(0) + \ldots + x^k s^{(k)}(0) + g(x)$. The s and g signals then have the same wavelet coefficients. This is the technical meaning of the phrase: "the wavelet suppresses a polynomial part of signal s". The signal g is the "irregular" part of the signal s. The ψ wavelet systematically suppresses the regular part and analyzes the irregular part. This effect is easily seen in Figure 4-2 up to detail D_4; the wavelet suppresses the slow sine wave which is locally assimilated to a polynomial.

Another way of suppressing a component of the signal consists of forcing certain coefficients $C(a,b)$ to be equal to 0. Having selected a set E of indices, we stipulate that $\forall(a,b) \in E$, $C(a,b) = 0$. We then synthesize the signal using the modified coefficients.

Let us illustrate with the following M-file, some features of wavelet processing using coefficients (resulting plots can be found in Figure 6-23).

```
% Load original 1-D signal.
    load sumsin; s = sumsin;

% Set the wavelet name and perform the decomposition
% of s at level 4, using coif3.
    w = 'coif3'; maxlev = 4;
    [c,l] = wavedec(s,maxlev,w);
    newc = c;

% Force to zero the detail coefficients at levels 3 and 4.
    newc = wthcoef('d',c,l,[3,4]);

% Force the detail coefficients at level 1 to zero on
% original time interval [400:600] and shrink otherwise.
% determine first and last index of
% level 1 coefficients.
    k = maxlev+1;
    first = sum(l(1:k-1))+1; last = first+l(k)-1;
    indd1 = first:last;

% shrink by dividing by 3.
    newc(indd1) = c(indd1)/3;

% find at level 1 indices of coefficients
% in the interval [400:600],
% note that time t in original grid corresponds to time
% t/2^k on the grid at level k. Here k=1.
    indd1 = first+400/2:first+600/2;

% force it to zero.
    newc(indd1) = zeros(size(indd1));

% Set to 4 a coefficient at level 2 corresponding roughly
% to original time t = 500.
    k = maxlev; first = sum(l(1:k-1))+1;
    newc(first+500/2^2) = 4;
% Synthesize modified decomposition structure.
    synth = waverec(newc,l,w);
```

A simple procedure to select *E*, called the thresholding procedure, is carried out using the `wthresh` function. The interface mode includes such procedures, which are also used for de-noising and compression.

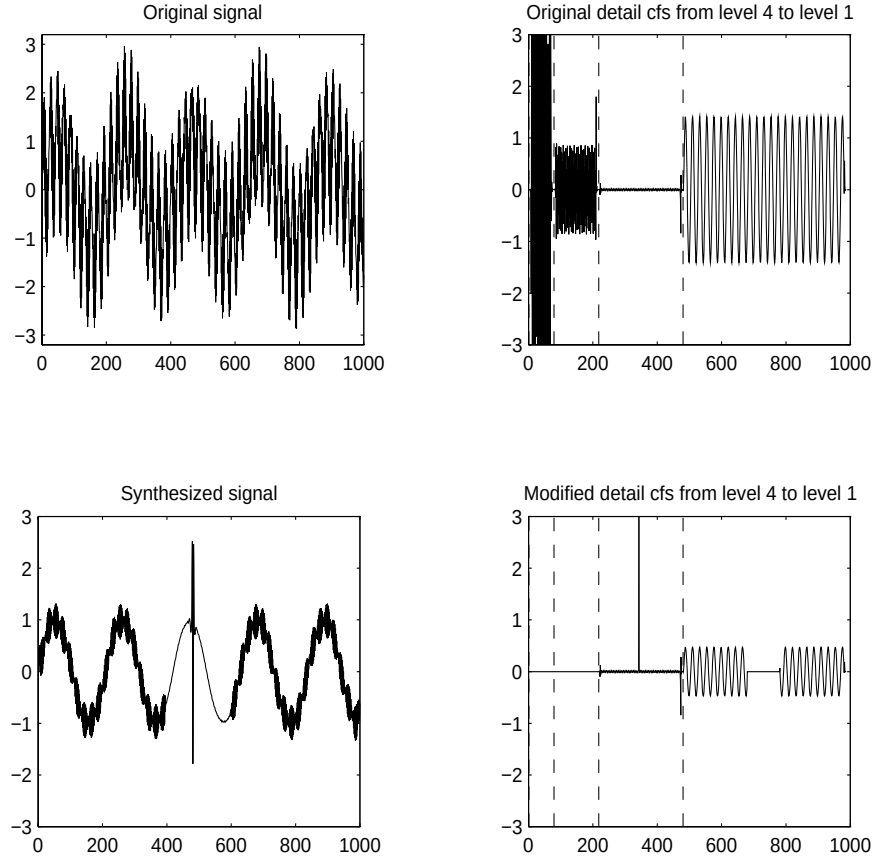


Figure 6-23: Suppress or modify signal components, acting on coefficients

Splitting Signal Components

Wavelet analysis is a linear technique: the wavelet coefficients of the linear combination of two signals $\alpha s^{(1)} + \beta s^{(2)}$ are equal to the linear combination of their wavelet coefficients $\alpha C_{j,k}^{(1)} + \beta C_{j,k}^{(2)}$. The same holds true for the corresponding approximations and details, for example $\alpha A_j^{(1)} + \beta A_j^{(2)}$ and $\alpha D_j^{(1)} + \beta D_j^{(2)}$.

Noise Processing

Let us first analyze noise as an ordinary signal. Then the probability characteristics: correlation function, spectrum, and distribution need to be studied.

In general, for a one-dimensional discrete-time signal, the high frequencies influence the details of the first levels, while the low frequencies influence the deepest levels and the associated approximations.

If a signal comprised only of white noise is analyzed, (see for example, Figure 4-3, the details at the various levels decrease in amplitude as the level increases. The variance of the details also decreases as the level increases. The details and approximations are not white noise anymore, as color is introduced by the filters.

On the coefficients $C(j,k)$, where j stands for the scale and k for the time, we can add often-satisfied properties for discrete time signals:

- If the analyzed signal s is stationary, zero mean, white noise, the coefficients are uncorrelated.
- If furthermore s is Gaussian, the coefficients are independent and Gaussian.
- If s is a colored, stationary, zero mean Gaussian sequence, the coefficients remain Gaussian. For each scale level j, the sequence of coefficients is a colored stationary sequence. It could be interesting to know how to choose the wavelet that would de-correlate the coefficients. This problem has not yet been resolved. What is more, the wavelet (if indeed it exists) most probably depends on the color of the signal. In order for the wavelet to be calculated, the color must be known. In most instances, this is beyond our reach.

- If *s* is a zero mean ARMA model stationary for each scale *j*, then $C(j,k), k \in Z$ is also a stationary, zero mean ARMA process whose characteristics depend on *j*.
- If *s* is a noise whose:
  - correlation function ρ is known, we know how to calculate the correlations of $C(j,k)$ and $C(j,k')$.
  - spectrum $\hat{\rho}$ is known, we know how to calculate the spectrum of $C(j,k)$, $k \in Z$ and the cross spectrum of two different levels *j* and *j'*.

These results are easily established, since they can be deduced from the fact that the $C(a,b)$ coefficients are calculated primarily by convolving ψ and *s*, and using conventional formulas. The quantity that comes into play is the self-reproduction function $U(a,b)$, which is obtained by analyzing the ψ wavelet as if it was a signal:

$$U(a,b) = \int_R \frac{1}{\sqrt{a}} \psi\left(\frac{x-b}{a}\right) \psi(x)\, dx \ .$$

From the results for coefficients we deduce the properties of the details (and of the approximations), by using the formula:

$$D_j(n) = \sum_{k \in Z} C(j,k) \psi_{j,k}(n) \ ,$$

where the $C(j,k)$ coefficients are random variables and the functions $\psi_{j,k}$ are not. If the support of ψ is finite, only a finite number of terms will be summed.

De-Noising

This section discusses the problem of signal recovery from noisy data. This problem is easy to understand looking at the following simple example, where a slow sine is corrupted by a white noise.

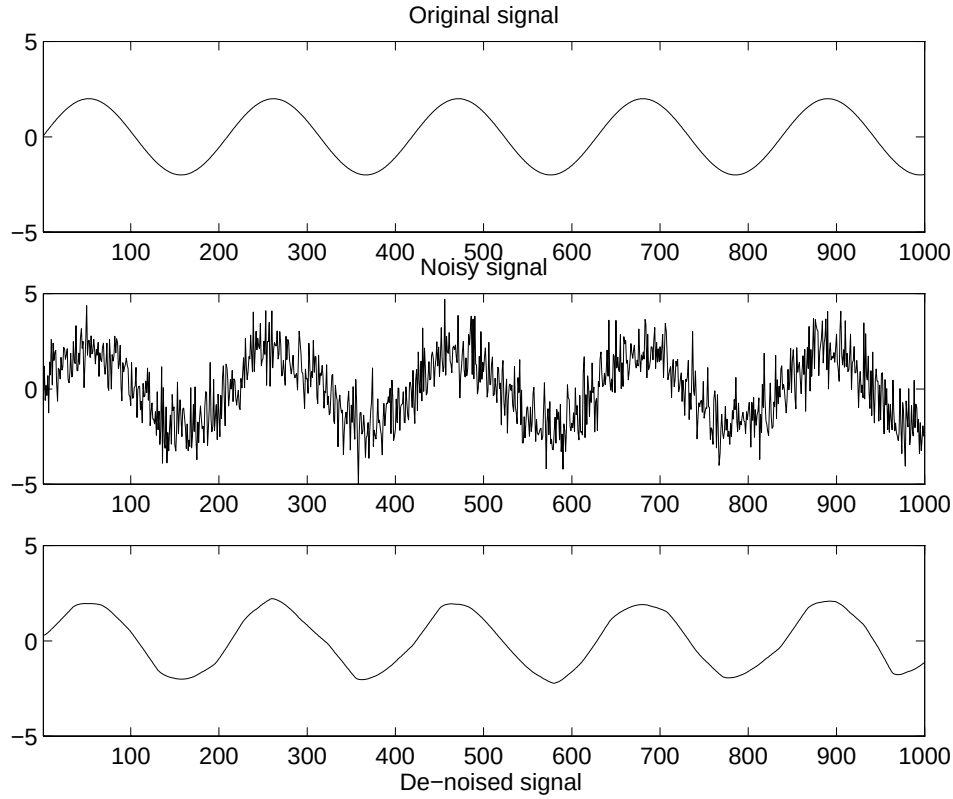


Figure 6-24: What is de-noising?

The Basic One-Dimensional Model

The underlying model for the noisy signal is basically of the following form:

$$s(n) = f(n) + \sigma e(n)$$

where time n is equally spaced.

In the simplest model we suppose that $e(n)$ is a Gaussian white noise $N(0,1)$ and the noise level s is supposed to be equal to 1.

The de-noising objective is to suppress the noise part of the signal s and to recover f. From a statistical viewpoint, the model is a regression model over time and the method can be viewed as a nonparametric estimation of the function f using orthogonal basis.

De-Noising Procedure Principles

The de-noising procedure proceeds in three steps:

1. Decompose

   Choose a wavelet, choose a level N. Compute the wavelet decomposition of the signal s at level N.

2. Threshold detail coefficients

   For each level from 1 to N, select a threshold and apply soft thresholding to the detail coefficients.

3. Reconstruct

   Compute wavelet reconstruction based on the original approximation coefficients of level N and the modified detail coefficients of levels from 1 to N.

Two points must be addressed: how to choose the threshold and how to perform the thresholding.

Soft or Hard Thresholding?

Thresholding can be done using the function:

```
yt = wthresh(y,sorh,thr)
```

which returns soft or hard thresholding of input `y`, depending on the `sorh` option. Hard thresholding is the simplest method. Soft thresholding has nice mathematical properties and the corresponding theoretical results are available.

Let us give a simple example.

```
y = linspace(-1,1,100);
thr = 0.28;
ythard = wthresh(y,'h',thr);
ytsoft = wthresh(y,'s',thr);
```

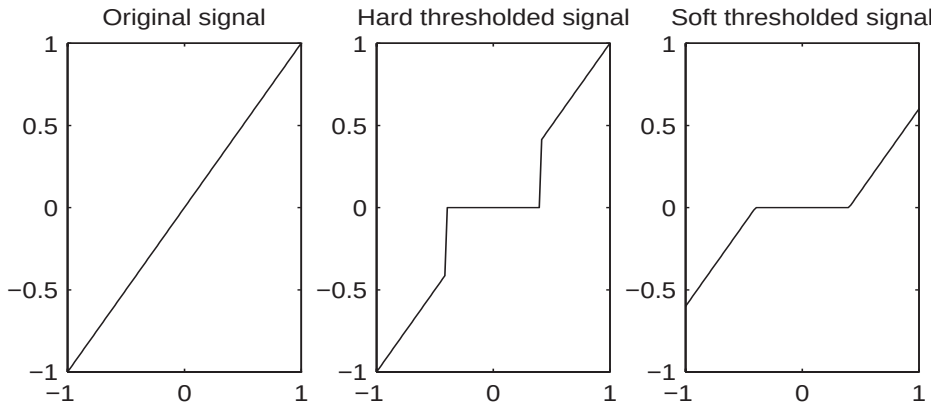


Figure 6-25: Hard and soft thresholding of the signal *s = x*

Comment: Let t denote the threshold. The hard threshold signal is x if $|x| > t$, and is 0 if $|x| \leq t$. The soft threshold signal is sign(x)($|x|$ - t) if $|x| > t$ and is 0 if $|x| \leq t$.

Hard thresholding can be described as the usual process of setting to zero the elements whose absolute values are lower than the threshold. Soft thresholding is an extension of hard thresholding, first setting to zero the elements whose absolute values are lower than the threshold, then shrinking

the nonzero coefficients towards 0 (see Figure 6-25). As can be seen in the comment of Figure 6-25, the hard procedure creates discontinuities at $x = \pm t$, while the soft procedure does not.

Threshold Selection Rules

According to the basic noise model, four threshold selection rules are implemented in the M-file `thselect`. Each rule corresponds to a `tptr` option in the command:

```
thr = thselect(y,tptr)
```

which returns the threshold value.

Option	Threshold selection rule
`'rigrsure'`	Selection using principle of Stein's Unbiased Risk Estimate (SURE)
`'sqtwolog'`	Fixed form threshold equal to *sqrt*(2∗*log*(*length*(*s*)))
`'heursure'`	Selection using a mixture of the first two options
`'minimaxi'`	Selection using minimax principle

- Option `tptr = 'rigrsure'` uses for the soft threshold estimator a threshold selection rule based on Stein's Unbiased Estimate of Risk (quadratic loss function). You get an estimate of the risk for a particular threshold value *t*. Minimizing the risks in *t* gives a selection of the threshold value.
- Option `tptr = 'sqtwolog'` uses a fixed form threshold yielding minimax performance multiplied by a small factor proportional to *log*(*length*(*s*)).
- Option `tptr = 'heursure'` is a mixture of the two previous options. As a result, if the signal-to-noise ratio is very small, the SURE estimate is very noisy. So if such a situation is detected, the fixed form threshold is used.
- Option `tptr = 'minimaxi'` uses a fixed threshold chosen to yield minimax performance for mean square error against an ideal procedure. The minimax principle is used in statistics in order to design estimators. Since the de-noised signal can be assimilated to the estimator of the unknown regression function, the minimax estimator is the option that realizes the minimum of the maximum mean square error obtained for the worst function in a given set.

Typically it is interesting to show how `thselect` works if `y` is a Gaussian white noise $N(0,1)$ signal.

```
y = randn(1,1000); thr = thselect(y,'rigrsure')
thr =
    2.0735

thr = thselect(y,'sqtwolog')
thr =
    3.7169

thr = thselect(y,'heursure')
thr =
    3.7169

thr = thselect(y,'minimaxi')
thr =
    2.2163
```

Because `y` is a standard Gaussian white noise, we expect that each method kills roughly all the coefficients. For Stein's Unbiased Risk Estimate and minimax thresholds, roughly 3% of coefficients are saved. For other selection rules, all the coefficients are set to 0.

We know that the detail coefficients vector is the superposition of the coefficients of *f* and the coefficients of *e*, and that the decomposition of *e* leads to detail coefficients, which are standard Gaussian white noises.

So minimax and SURE threshold selection rules are more conservative and would be more convenient when small details of function *f* lie in the noise range. The two other rules remove the noise more efficiently. The option `'heursure'` is a compromise. In this example, the fixed form threshold wins.

Recalling step 2 of the de-noise procedure, the function `thselect` performs a threshold selection and then each level is thresholded. This second step can be done using `wthcoef`, directly handling the wavelet decomposition structure of the original signal *s*.

Dealing with Unscaled Noise and Non-White Noise

It is clear that in practice the basic model cannot be used directly. We examine here the options available in the main de-noising function `wden`, in order to deal with model deviations.

The simplest use of `wden` is:

```
sd = wden(s,tptr,sorh,scal,n,wav)
```

which returns the de-noised version `sd` of the original signal `s` obtained using the `tptr` threshold selection rule. Other parameters needed are `sorh`, thresholding of details coefficients of the decomposition at level `n` of `s` by the wavelet called `wav`. The remaining parameter `scal` is to be specified. It corresponds to threshold's rescaling methods.

Option	Corresponding model
`'one'`	Basic model
`'sln'`	Basic model with unscaled noise
`'mln'`	Basic model with non-white noise

- Option `scal = 'one'` corresponds to the basic model.
- In general you can ignore the noise level that must be estimated. The detail coefficients cD_1 (the finest scale) are essentially noise coefficients with standard deviation equal to σ. The median absolute deviation of the coefficients is a robust estimate of σ. The use of a robust estimate is crucial for two reasons. The first one is that if level 1 coefficients contain *f* details, these details are concentrated in few coefficients if the function f is sufficiently regular. The second reason is to avoid signal end effects, which are pure artifacts due to computations on the edges.

  Option `scal = 'sln'` handles threshold rescaling using a single estimation of level noise based on the first level coefficients.
- When you suspect a nonwhite noise *e*, thresholds must be rescaled by a level-dependent estimation of the level noise. The same kind of strategy is used by estimating σ_{lev} level by level. This estimation is implemented in M-file `wnoisest`, directly handling the wavelet decomposition structure of the original signal *s*.

  Option `scal = 'mln'` handles threshold rescaling using a level-dependent estimation of the level noise.

A more general procedure wdencmp performs wavelet coefficients thresholding for both de-noising and compression purposes directly handling one-dimensional and two-dimensional signals. It allows you to define your own thresholding strategy selecting in:

```
xd = wdencmp(opt,x,wav,n,thr,sorh,keepapp);
```

- `opt = 'gbl'` and `thr` is a positive real number for uniform threshold `opt = 'lvd'` and `thr` is a vector for level dependent threshold.
- `keepapp = 1` to keep approximation coefficients, as previously and `keepapp = 0` to allow approximation coefficients thresholding.
- `x` is the signal to be de-noised and `wav`, `n`, `sorh` are the same as above.

De-Noising in Action

We begin with examples of one-dimensional de-noising methods with the first example credited to Donoho and Johnstone. For the first test function available using wnoise, use the following M-file.

```
% Set signal to noise ratio and set rand seed.
snr = 4; init = 2055615866;

% Generate original signal xref and a noisy version x adding
% a standard Gaussian white noise.
[xref,x] = wnoise(1,11,snr,init);

% De-noise noisy signal using soft heuristic SURE thresholding
% and scaled noise option, on detail coefficients obtained
% from the decomposition of x, at level 3 by sym8 wavelet.
xd = wden(x,'heursure','s','one',3,'sym8');
```

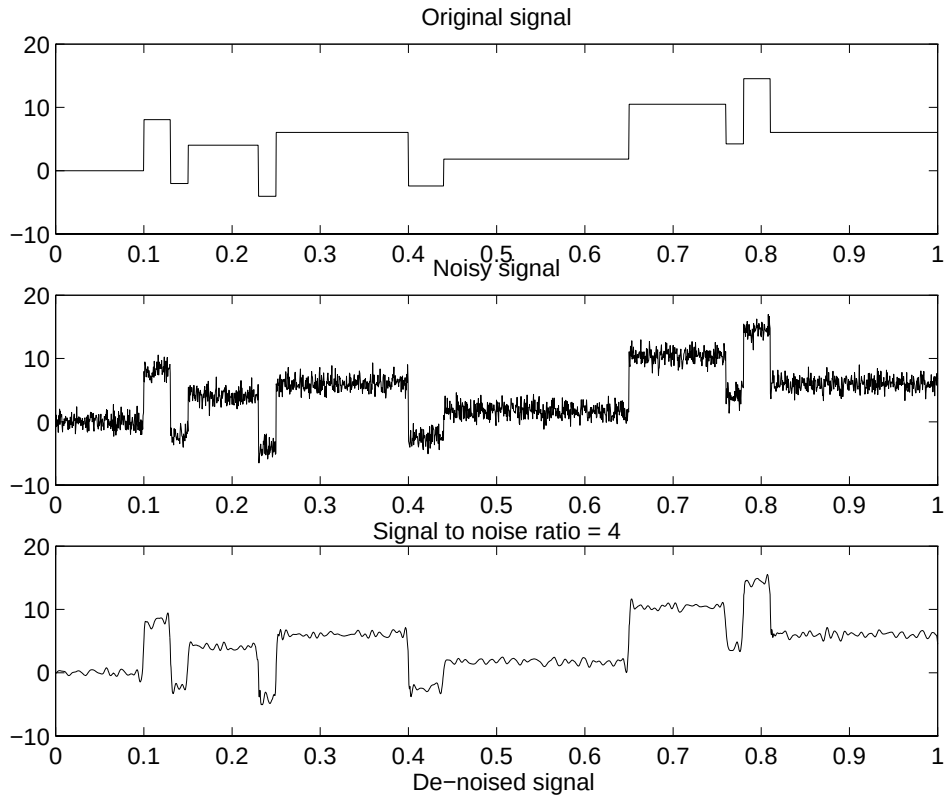


Figure 6-26: Blocks de-noising

So despite the fact that only a small number of large coefficients characterize the original signal, the method performs very well (see Figure 6-26). If you want to see more about how the thresholding works, use the GUI.

As a second example, let us try the method on the highly perturbed part of the electrical signal studied above.

According to this previous analysis, let us use db3 wavelet and decompose at level 3. In order to deal with the composite noise nature, let us try a level-dependent noise size estimation.

```
%Load electrical signal and select part of it.
load leleccum; indx = 2000:3450;
x = leleccum(indx);

% Find first value in order to avoid edge effects.
deb = x(1);

% De-noise signal using soft fixed form thresholding
% and unknown noise option.
xd = wden(x-deb,'sqtwolog','s','mln',3,'db3')+deb;
```

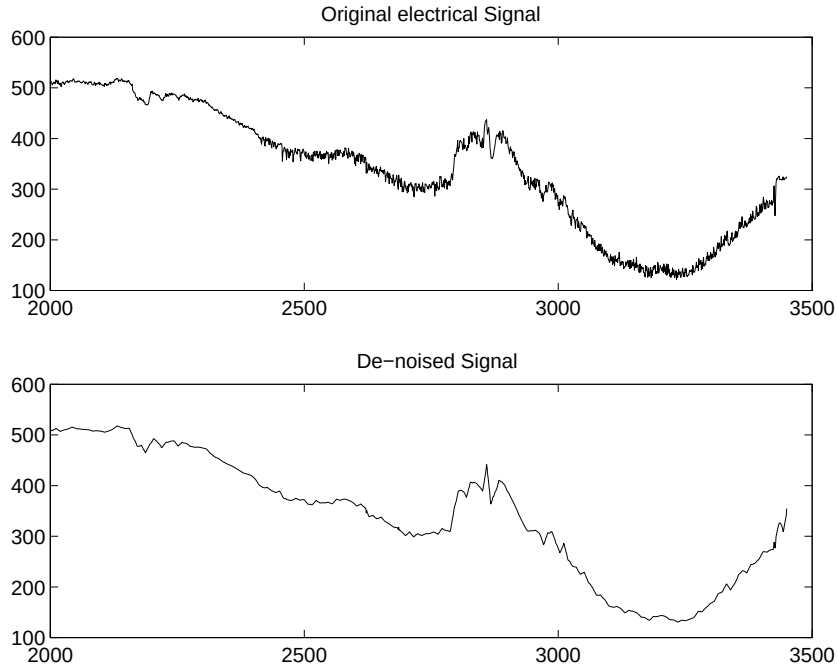


Figure 6-27: Electrical signal de-noising

The result is quite good in spite of the time heterogeneity of the nature of the noise after and before the beginning of the sensor failure around time 2450.

Extension to Image De-Noising

The de-noising method described for the one-dimensional case applies also to images and applies well to geometrical images. A direct translation of the one-dimensional model is:

$$s(i,j) = f(i,j) + \sigma e(i,j),\ i,j = 0, \ldots, m\text{-}1$$

where e is a white Gaussian noise with unit variance.

The two-dimensional de-noising procedure has the same three steps and uses two-dimensional wavelet tools instead of one-dimensional ones. For the threshold selection m^2 is used instead of n if the fixed form threshold is used.

Note that except for the "automatic" one-dimensional de-noising case, de-noising and compression are performed using `wdencmp`. As an example, you can use the following M-file illustrating the de-noising of a synthetic image.

```
%Load original image.
load sinsin

% Generate noisy image.
init=2055615866; randn('seed',init);
x = X + 18*randn(size(X));

% Find default values using ddencmp.
% In this case fixed form threshold is used
% with estimation of level noise, thresholding
% mode is soft and the approximation coefficients
% are kept.
[thr,sorh,keepapp] = ddencmp('den','wv',x);

% thr is roughly equal to 18*sqrt(log(prod(size(x))))
thr
thr =
80.6881

% De-noise image using global thresholding option.
xd = wdencmp('gbl',x,'sym4',2,thr,sorh,keepapp);
```

The result shown below is acceptable.

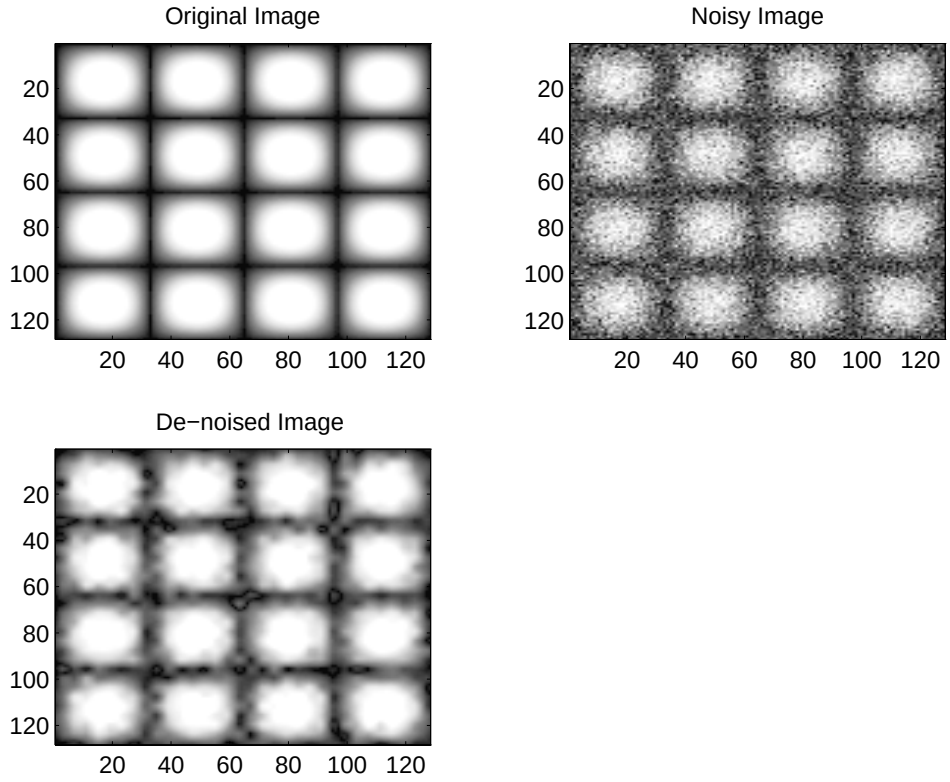


Figure 6-28: Image de-noising

More About De-Noising

Recently, new de-noising methods based on wavelet decomposition appear mainly initiated by Donoho and Johnstone in the USA, and Kerkyacharian and Picard in France. Meyer considers that this topic is one of the most significant applications of wavelets (cf. [Mey93] p. 173). This chapter and the corresponding M-files follow the work of the above mentioned researchers. More details can be found in the bibliography by Donoho.

Data Compression

The compression features of a given wavelet basis are primarily linked to the relative scarceness of the wavelet domain representation for the signal. The notion behind compression is based on the concept that the regular signal component can be accurately approximated using the following elements: a small number of approximation coefficients (at a suitably chosen level) and some of the detail coefficients.

Like de-noising, the compression procedure contains three steps:

1. Decompose
2. Threshold detail coefficients

   For each level from 1 to *N*, a threshold is selected and hard thresholding is applied to the detail coefficients.
3. Reconstruct

The difference with the de-noising procedure is found in step 2. There are two compression approaches available. The first consists of taking the wavelet expansion of the signal and keeping the largest absolute value coefficients. In this case you can set a global threshold, a compression performance, or a relative square norm recovery performance. Thus only a single parameter needs to be selected. The second approach consists of applying visually determined level-dependent thresholds.

Let us examine two real-life examples of compression using global thresholding, for a given and unoptimized wavelet choice, to produce a nearly complete square norm recovery for a signal (see Figure 6-29) and for an image (see Figure 6-30).

```
% Load electrical signal and select a part.
load leleccum; indx = 2600:3100;
x = leleccum(indx);

% Perform wavelet decomposition of the signal.
n = 3; w = 'db3';
[c,l] = wavedec(x,n,w);

% Compress using a fixed threshold.
thr = 35;
[xd,cxd,lxd,perf0,perfl2] = wdencmp('gbl',c,l,w,n,thr,'h',1);
```

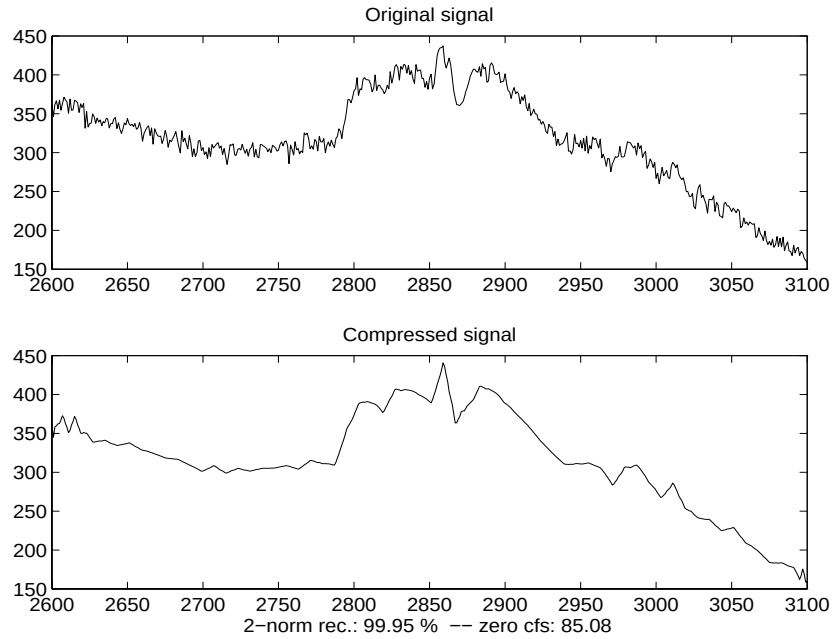


Figure 6-29: Signal compression

```
% Load original image.
load woman; x = X(100:200,100:200);
nbc = size(map,1);

% Wavelet decomposition of x.
n = 5; w = 'sym2'; [c,l] = wavedec2(x,n,w);

% Wavelet coefficients thresholding.
thr = 20;
[xd,cxd,lxd,perf0,perfl2] = wdencmp('gbl',c,l,w,n,thr,'h',1);
```

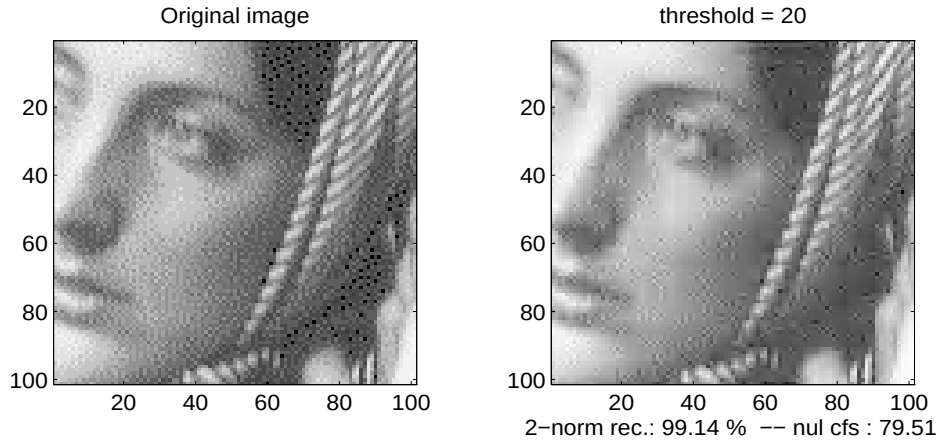


Figure 6-30: Image compression

If the wavelet representation is too dense, similar strategies can be used in the wavelet packet framework in order to obtain a sparser representation. You can then determine the best decomposition with respect to a suitably selected entropy-like criterion, which corresponds to the selected purpose (de-noising or compression).

Default Values for De-Noising and Compression

De-noising.

Automatic mode.

Wavelet 1-D or 2-D:

The global threshold is derived from Donoho-Johnstone fixed form threshold strategy for an unscaled white noise.

Manual mode.

Wavelet 1-D:

The level-dependent thresholds are derived from Birge-Massart strategy with $\alpha = 3$.

Wavelet 2-D:

The global threshold is derived from Donoho-Johnstone fixed form threshold strategy for an unscaled white noise.

Compression.

Automatic mode.

Wavelet 1-D or 2-D and Wavelet Packet 1-D:

The global threshold is derived from an equal balance between the percentages of retained energy and number of zeros.

Wavelet Packet 2-D:

The global threshold is the square root of the threshold value derived from an equal balance between the percentages of retained energy and number of zeros.

Manual mode.

Wavelet 1-D:

The level-dependent thresholds are derived from Birge-Massart strategy with $\alpha = 1.5$.

Wavelet 2-D:

The global threshold is based on the analysis of the level-one detail coefficients `cd1` and is equal to `t = median(abs(cd1))` or `0.005*max(abs(cd1))` if `t` is zero.

About the Birge-Massart Strategy

The Birge-Massart strategy is based on results on adaptive functional estimation in regression or density contexts (more details can be found in the reference [BirM95] at the end of this Chapter).

Fortunately, this sophisticated estimate can be implemented in a very simple way, like the previously described procedures for de-noising or compression.

It uses level-dependent thresholds obtained by the following wavelet coefficients selection rule.

Let j_0 be the decomposition level, m be the length of coarsest approximation coefficients over 2 and α be a real greater than 1.

The numbers j_0, m and α define the strategy:

- at level j_0+1 (and coarser levels), everything is kept.
- for level j from 1 to j_0, the kj larger coefficients in absolute value, are kept with:

$$kj = m/(j_0 + 1 - j)^{\alpha}$$

Typically the parameter α is equal to 1.5 for compression and α is equal to 3 for de-noising.

Wavelet Packets

The wavelet packet method is a generalization of wavelet decomposition that offers a richer signal analysis.

Wavelet packet atoms are waveforms indexed by three naturally interpreted parameters: position and scale (as in wavelet decomposition and frequency).

For a given orthogonal wavelet function, we generate a library of wavelet packet bases. Each of these bases offers a particular way of coding signals, preserving global energy and reconstructing exact features. The wavelet packets can then be used for numerous expansions of a given signal. We then select the most suitable decomposition of a given signal with respect to an entropy-based criterion.

There exist simple and efficient algorithms for both wavelet packet decomposition and optimal decomposition selection. We can then produce adaptive filtering algorithms with direct applications in optimal signal coding and data compression.

From Wavelets to Wavelet Packets: Decomposing the Details

In the orthogonal wavelet decomposition procedure, the generic step splits the approximation coefficients into two parts. After splitting we obtain a vector of approximation coefficients and a vector of detail coefficients, both at a coarser scale. The information lost between two successive approximations is captured in the detail coefficients. Then next step consists of splitting the new approximation coefficient vector; successive details are never re-analyzed.

In the corresponding wavelet packet situation, each detail coefficient vector is also decomposed into two parts using the same approach as in approximation vector splitting. This offers the richest analysis: the complete binary tree is produced as shown in Figure 6-31.

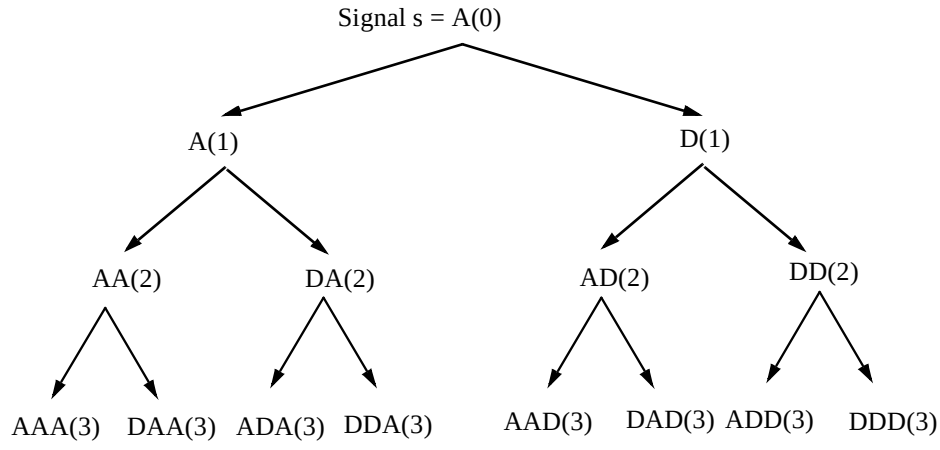


Figure 6-31: Wavelet packet decomposition tree at level 3

The idea of this decomposition is to start from a scale-oriented decomposition and then to analyze the obtained signals on frequency subbands.

Wavelet Packets in Action: An Introduction

The following simple examples illustrate certain differences between wavelet analysis and wavelet packet analysis.

Example 1: Analyzing a Sine Function

The signal to be analyzed is a sampled sine function of period 8. In order to simplify the presentation, the length is 8192 and the `haar` wavelet is used. Only a portion of the signal is displayed. Figure 6-32 contains the "time-frequency" plot (x-axis is time and y-axis is frequency, high to low from the top to the bottom) for the wavelet decomposition (on the left) and for the wavelet packet decomposition (on the right), both corresponding to a decomposition at level 6.

Wavelet decomposition localizes the period of the sine within the interval [8,16]. Wavelet packets provide a more precise estimation of the actual period.

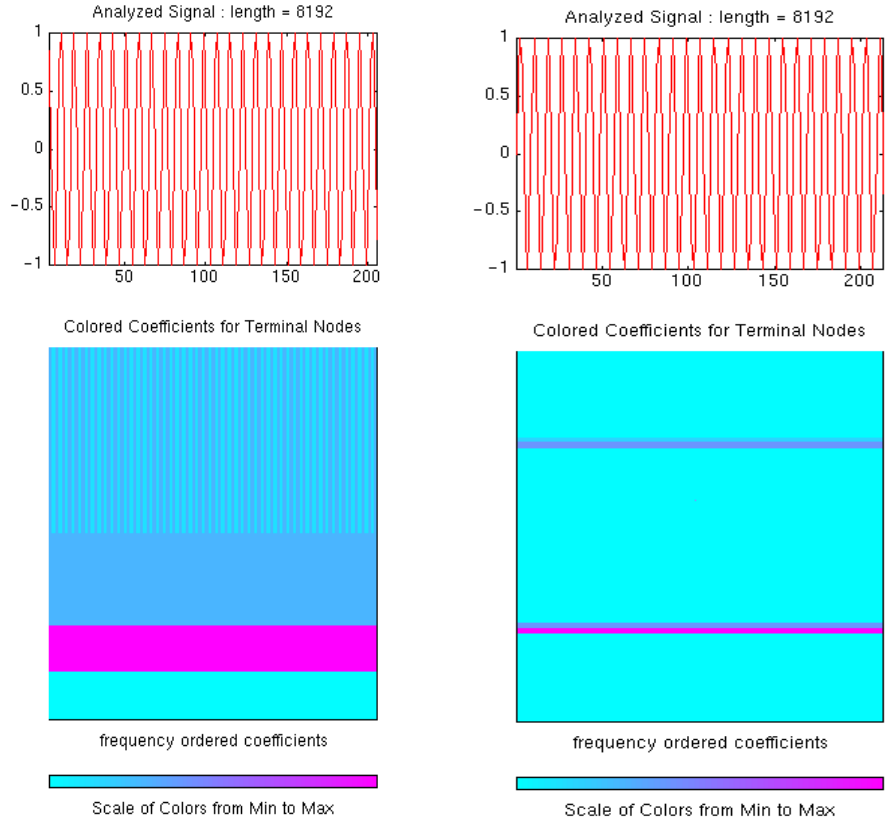


Figure 6-32: Wavelets (left) versus wavelet packets (right): a sine function

Example 2: Analyzing a Chirp Signal

The signal to be analyzed is a chirp: an oscillatory signal with increasing modulation $\sin(250\pi t^2)$ sampled 512 times on [0, 1]. For this "linear" chirp, the derivative of the phase is linear. On the left of Figure 6-33, a wavelet analysis does not easily detect this time-frequency property of the signal. But on the right of Figure 6-33, the linear slope for the greatest wavelet packet coefficients in absolute value is obvious. The same experiment can be done with a "quadratic" chirp of the form $\sin(k\pi t^3)$ in which the greatest wavelet packet coefficients exhibit a quadratic time frequency pattern.

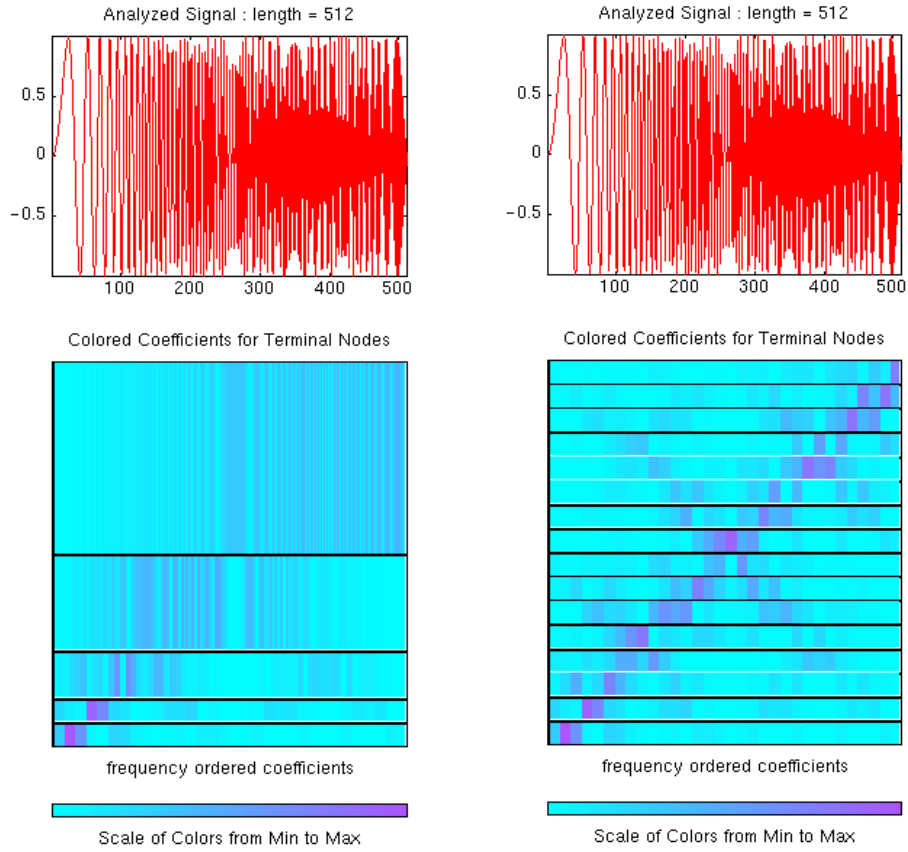


Figure 6-33: Wavelets (left) versus wavelet packets (right): damped oscillations.

Building Wavelet Packets

The computation scheme for wavelet packets generation, is easy when using an orthogonal wavelet. We start with the two filters of length $2N$, denoted $h(n)$ and $g(n)$, corresponding to the wavelet. They are, respectively, the reversed versions of the low-pass decomposition filter and the high-pass decomposition filter divided by $\sqrt{2}$.

Now by induction let us define the following sequence of functions $(W_n(x), n = 0, 1, 2, ...)$ by:

$$W_{2n}(x) = 2\sum_{k=0}^{2N-1} h(k)W_n(2x-k)$$

$$W_{2n+1}(x) = 2\sum_{k=0}^{2N-1} g(k)W_n(2x-k)$$

where $W_0(x) = \phi(x)$ is the scaling function and $W_1(x) = \psi(x)$ is the wavelet function.

For example for the Haar wavelet we have:

$N = 1, h(0) = h(1) = 1/2$ and $g(0) = -g(1) = 1/2$.

The equations become:

$W_{2n}(x) = W_n(2x) + W_n(2x-1)$ and $W_{2n+1}(x) = W_n(2x) - W_n(2x-1)$.

$W_0(x) = \phi(x)$ is the Haar scaling function and $W_1(x) = \psi(x)$ is the Haar wavelet, both supported in [0, 1]. Then we can obtain W_{2n} by adding two 1/2-scaled versions of W_n with distinct supports [0,1/2] and [1/2,1] and obtain W_{2n+1} by subtracting the same versions of W_n.

For n = 0 to 7, we have the W-functions shown below.

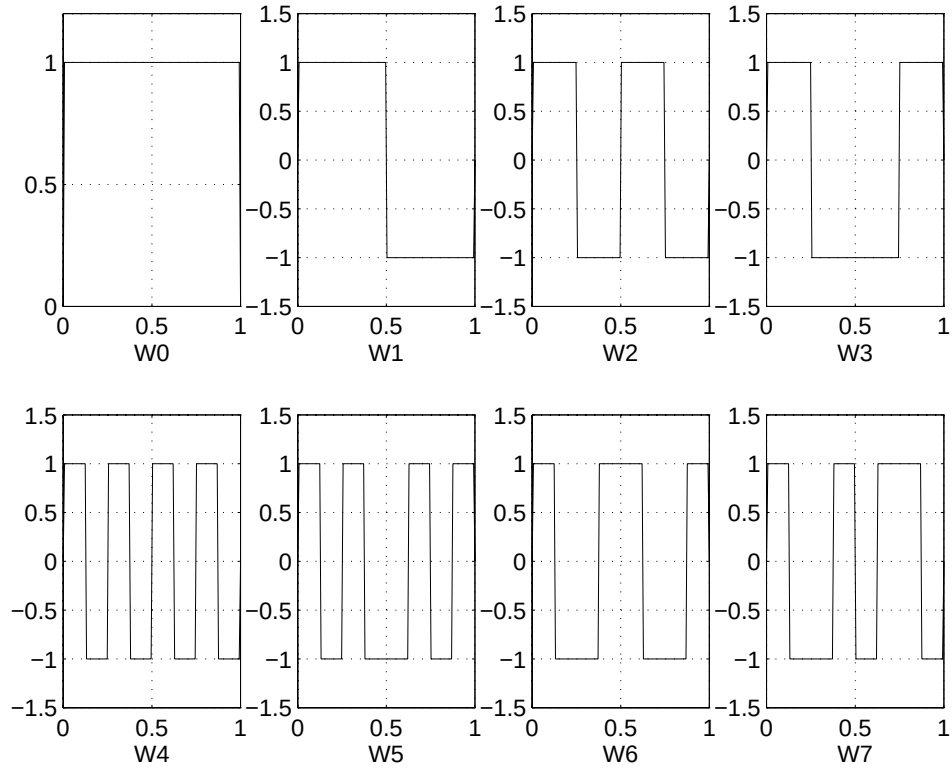


Figure 6-34: The Haar wavelet packets

This can be obtained using the following command:

```
[wfun,xgrid] = wpfun('db1',7,5);
```

which returns in `wfun` the approximate values of W_n for n = 0 to 7, computed on a $1/2^5$ grid of the support `xgrid`.

Starting from more regular original wavelets and using a similar construction, we obtain smoothed versions of this system of *W*-functions, all with support in the interval [0, 2*N*-1]. Figure 6-35 presents the system of *W*-functions for the original db2 wavelet.

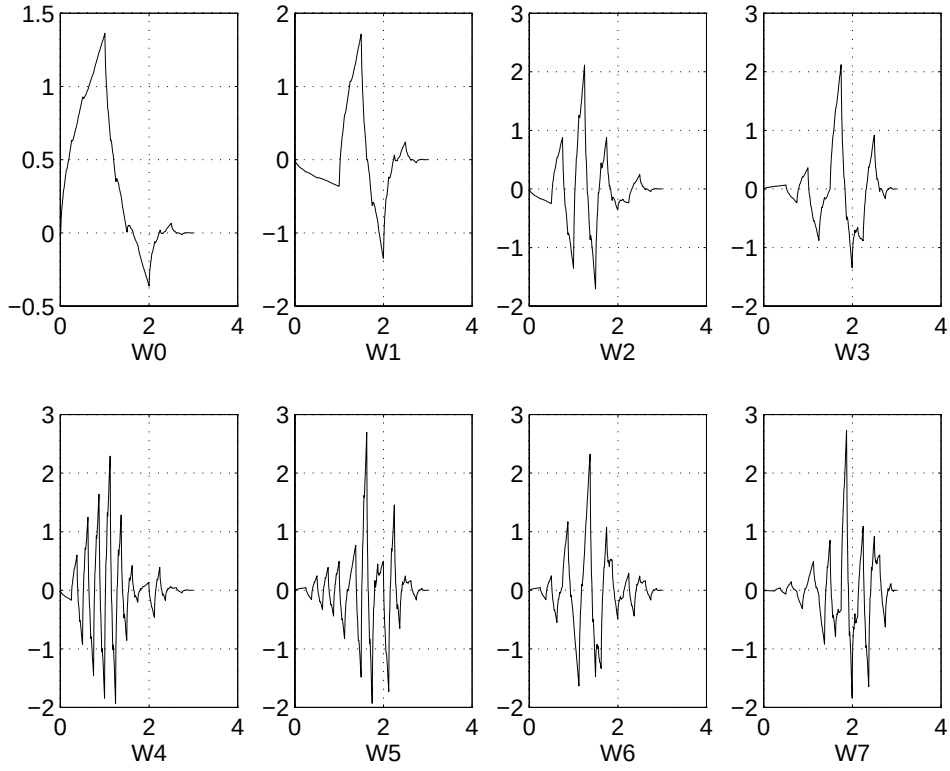


Figure 6-35: The *db*2 wavelet packets

Wavelet Packet Atoms

Starting from the functions $(W_n(x), n \in N)$ and following the same line leading to orthogonal wavelets, we consider the three-indexed family of analyzing functions (the waveforms):

$W_{j,n,k}(x) = 2^{-j/2} W_n(2^{-j}x - k)$ where $n \in N$ and $(j,k) \in Z^2$.

As in the wavelet framework, k can be interpreted as a time-localization parameter and j as a scale parameter. So what is the interpretation of n?

As can be seen in the previous figures, $W_n(x)$ "oscillates" approximately n times. So for fixed values of j and k, $W_{j,n,k}$ analyzes the fluctuations of the signal roughly around the position $2^j \cdot k$, at the scale 2^{-j} and at various frequencies for the different admissible values of the last parameter n.

In fact examining carefully the wavelet packets displayed in Figure 6-34 and Figure 6-35, the naturally ordered W_n for $n = 0, 1, ..., 7, ...$ does not match exactly the property that W_m oscillates more than $W_{m'}$ if $m > m'$. More precisely, counting the number of zero-crossing for the `db1` wavelet packets, we have:

Natural order n	0	1	2	3	4	5	6	7
Number of zero-crossing for $db1$ W_n	2	3	5	4	9	8	6	7

So in order to restore the property that the main frequency increases monotonically with the order, it is convenient to define the "frequency" order obtained from the natural one recursively.

Natural order n	0	1	2	3	4	5	6	7
"Frequency" order $\sigma(n)$	0	1	3	2	6	7	5	4

To analyze a signal (the chirp of example 2 for instance), it is better to plot the wavelet packet coefficients following the "frequency" order (on the right of the Figure 6-36) from the low frequencies at the bottom to the high frequencies at the top, rather than naturally ordered coefficients (on the left of Figure 6-36).

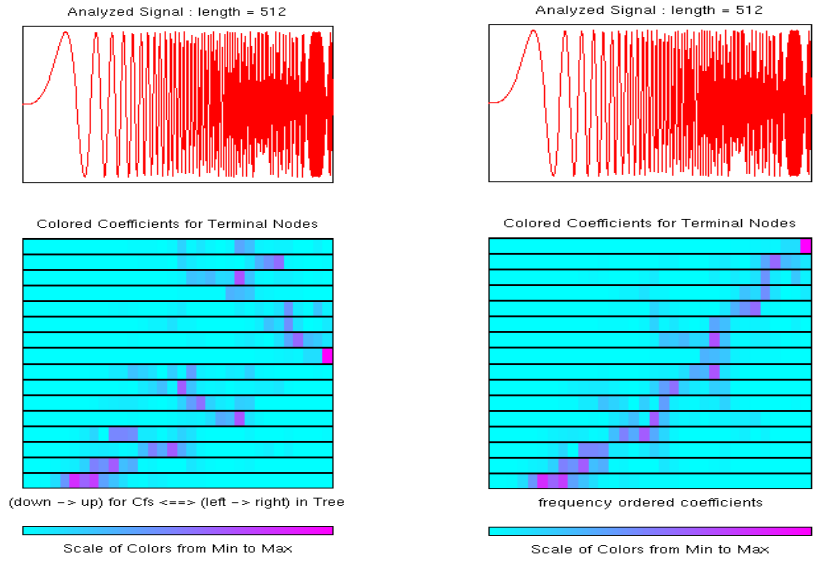


Figure 6-36: Natural and frequency ordered wavelet packets coefficients

The two options are available when the GUI tools are used, since the packets are organized following the natural order (see below) in order to preserve consistency.

Organizing the Wavelet Packets

The set of functions: $W_{j,n} = (W_{j,n,k}(x),\ k \in Z)$ is the (j,n) wavelet packet. For positive values of integers j and n, wavelet packets are organized in trees. The tree in Figure 6-37 is created in order to give a maximum level decomposition equal to 3. For each scale j, the possible values of parameter n are: 0, 1, ..., $2j$ -1.

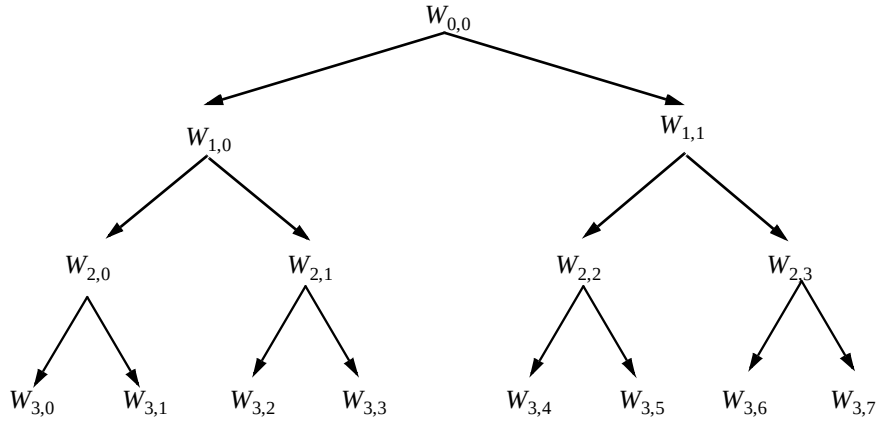


Figure 6-37: Wavelet packets organized in a tree, scale j defines depth and frequency n defines position in the tree

The notation $W_{j,n}$, where j denotes scale parameter and n the frequency parameter, is consistent with the usual depth-position tree labeling.

We have $W_{0,0} = (\phi(x-k), k \in Z)$, and $W_{1,1} = (\psi(x-k), k \in Z)$.

It turns out that the library of wavelet packet bases contains the wavelet basis. More precisely if V_0 denotes the space (spanned by the family $W_{0,0}$) in which the signal to be analyzed lies then $(W_{d,1};\ d \geq 1)$ is an orthogonal basis of V_0.

For every strictly positive integer D, $(W_{D,0}, (W_{d,1};\ 1 \leq d \leq D))$ is an orthogonal basis of V_0.

We also know that $\{(W_{j+1,2n}),(W_{j+1,2n+1})\}$ is an orthogonal basis of the space spanned by $W_{j,n}$.

This last property gives a precise interpretation of splitting in the wavelet packet organization tree, because all the developed nodes are of the form shown in the figure below.

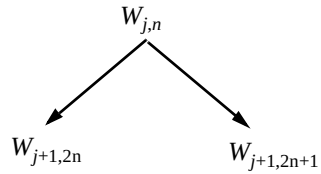


Figure 6-38: Wavelet packet tree: split and merge

It follows that the leaves of every connected binary subtree of the wavelet packet tree correspond to an orthogonal basis of the initial space. For a finite energy signal, any wavelet packet basis will provide exact reconstruction and offer a specific way of coding the signal, using information allocation in frequency scale subbands.

Choosing the Optimal Decomposition

Based on the organization of the wavelet packet library, it is natural to count the decompositions issued from a given orthogonal wavelet. As a result, a signal of length $N = 2^L$ can be expanded in at most 2^N different ways, the number of binary subtrees of a complete binary subtree of depth L. As this number may be very large, and since explicit enumeration is generally unmanageable, it is interesting to find an optimal decomposition with respect to a convenient criterion, computable by an efficient algorithm. We are looking for a minimum of the criterion.

Functionals verifying an additivity-type property are well suited for efficient searching of binary-tree structures and the fundamental splitting. Classical entropy-based criteria match these conditions and describe information-related properties for an accurate representation of a given signal. Entropy is a common concept in many fields, mainly in signal processing. Let us list four different entropy criteria (see [CoiW92]), many others are available and can be easily integrated (type `help wentropy`). In the following expressions s is the signal and $(s_i)_i$ the coefficients of s in an orthonormal basis.

The entropy E must be an additive cost function such that $E(0) = 0$ and

$$E(s) = \sum_i E(s_i)$$

.

- The (non-normalized) Shannon entropy.
  $E1(s_i) = -s_i^2 \log(s_i^2)$ so $E1(s) = -\sum_i s_i^2 \log(s_i^2)$
  with the convention $0\log(0) = 0$.
- The concentration in l^p norm with $1 \le p < 2$.
  $E2(s_i) = |s_i|^p$ so $E2(s) = \sum_i |s_i|^p = \|s\|_p^p$.
- The logarithm of the "energy" entropy.
  $E3(s_i) = \log(s_i^2)$ so $E3(s) = \sum_i \log(s_i^2)$
  with the convention $\log(0) = 0$.
- The threshold entropy.

  $E4(s_i) = 1$ if $|s_i| > \varepsilon$ and 0 elsewhere so $E4(s) = \#\{\text{such that } |s_i| > \varepsilon\}$ is the number of time instants when the signal is greater than a threshold ε.

These entropy functions are available using the `wentropy` M-file.

Example 1: Compute Various Entropies.

1 Generate a signal of energy equal to 1.

```
s = ones(1,16)*0.25;
```

2 Compute Shannon entropy of *s*.

```
e1 = wentropy(s,'shannon')
    e1 = 2.7726
```

3 Compute $l^{1.5}$ entropy of *s*, equivalent to `norm(s,1.5)`$^{1.5}$.

```
e2 = wentropy(s,'norm',1.5)
    e2 = 2
```

4 Compute the "log energy" entropy of *s*.

```
e3 = wentropy(s,'log energy')
    e3 = -44.3614
```

5 Compute threshold entropy of *s*, using a threshold value of 0.24.

```
e4 = wentropy(s,'threshold', 0.24)
    e4 = 16
```

Example 2: Minimum-Entropy Decomposition.

This simple example illustrates the use of entropy to determine whether a new splitting is of interest in order to obtain a minimum-entropy decomposition.

1. We start with a constant original signal. Two pieces of information are sufficient to define and to recover the signal (i.e., length and constant value).

```
w00 = ones(1,16)*0.25;
```

2. Compute entropy of original signal.

```
e00 = wentropy(w00,'shannon')
    e00 = 2.7726
```

3. Then split `w00` using the haar wavelet.

```
[w10,w11] = dwt(w00,'db1');
```

4. Compute entropy of approximation at level 1

```
e10 = wentropy(w10,'shannon')
    e10 = 2.0794
```

The detail of level 1, `w11`, is zero; the entropy `e11` is zero. Due to the additivity property the entropy of decomposition is given by `e10+e11=2.0794`. This has to be compared to the initial entropy `e00=2.7726`. We have `e10 + e11 < e00`, so the splitting is interesting.

5. Now split `w10` and not `w11` simply because the splitting of a null vector is without interest, the entropy being zero.

```
[w20,w21] = dwt(w10,'db1');
```

6. We have `w20=0.5*ones(1,4)` and `w21` is zero. The entropy of approximation level 2 is:

```
e20 = wentropy(w20,'shannon')
    e20 = 1.3863
```

   Again we have `e20 + 0 < e10`, so splitting makes the entropy decrease.

7 Then:

```
[w30,w31] = dwt(w20,'db1');
e30 = wentropy(w30,'shannon')
    e30 = 0.6931
[w40,w41] = dwt(w30,'db1')
    w40 = 1.0000
    w41 = 0
e40 = wentropy(w40,'shannon')
    e40 = 0
```

In the last splitting operation we find that only one piece of information is needed to reconstruct the original signal. The wavelet basis at level 4 is a best basis according to shannon entropy (with null optimal entropy since `e40+e41+e31+e21+e11 = 0`).

8 All this work can be performed simply using:

```
s = ones(1,16)*0.25;
```

9 Perform wavelet packets decomposition.

```
[t,d] = wpdec(s,4,'haar','shannon');
```

The wavelet packet tree below shows the nodes labeled with original entropy numbers.

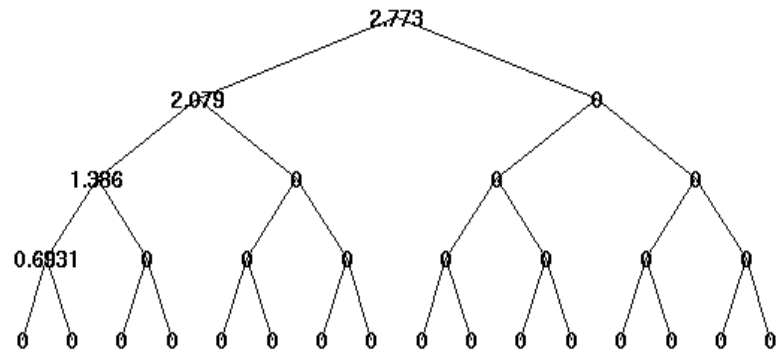


Figure 6-39: Entropy values

10 Now compute the best tree.

```
[bt,bd] = besttree(t,d);
```

The best tree is displayed in the figure below. In this case, the best tree corresponds to the wavelet tree. The nodes are labeled with optimal entropy.

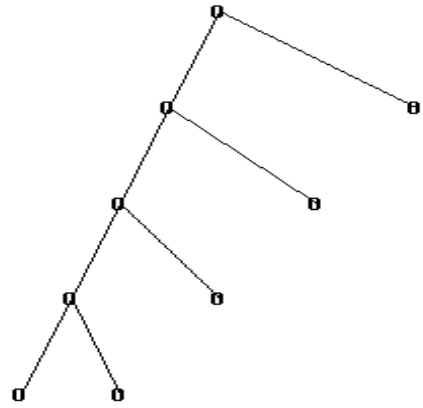


Figure 6-40: Optimal entropy values

Wavelet Packets 1-D Decomposition Structure

Using wavelet packets requires tree-related actions and labeling. The implementation of the user interface is built around this consideration. See the Reference Section for more information on the technical details.

The complete binary tree of depth *D* corresponding to a wavelet packet decomposition tree (WPT) developed at level *D*, *is* shown below:

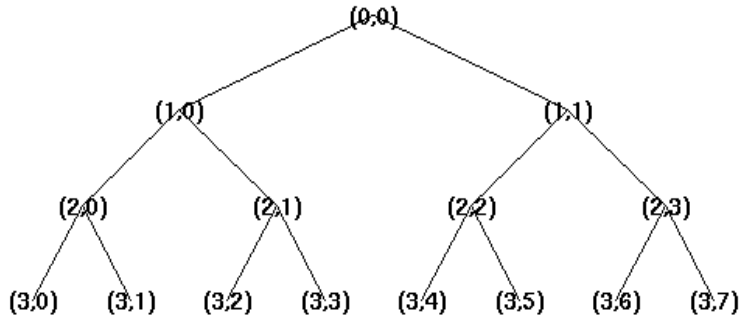


Figure 6-41: Binary tree of depth 3

We have the following relationships:

Decomposition tree	Subtree such that the set of leaves is a basis
Wavelet packets decomposition tree	Complete binary tree: WPT of depth *D*
Wavelet packets optimal decomposition tree	Binary subtree of WPT
Wavelet packets best-level tree	Complete binary subtree of WPT
Wavelet decomposition tree	Left unilateral binary subtree of WPT of depth *D*
Wavelet best-basis tree	Left unilateral binary subtree of WPT

We deduce the following definitions of optimal decompositions, with respect to an entropy criterion *E*.

Decompositions	Optimal decomposition	Best-level decomposition
Wavelet packet decompositions	Search among 2^D trees	Search among D trees
Wavelet decompositions	Search among D trees	Search among D trees

For any nonterminal node in a complete binary tree of depth *D* corresponding to a wavelet packet decomposition tree, we use the following basic step in order to find the optimal subtree with respect to a given entropy criterion *E* (where *Eopt* denotes the optimal entropy value):

Entropy condition	Action on tree and on entropy labelling
$E(node) \leq \sum_{\text{c child of node}} Eopt(c)$	If $(node \neq root)$, merge and set $Eopt(node) = E(node)$
$E(node) > \sum_{\text{c child of node}} Eopt(c)$	Split and set $Eopt(node) = \sum_{\text{c child of node}} Eopt(c)$

with the natural initial condition on the reference tree, $Eopt(t) = E(t)$ for each terminal node *t*.

Wavelet Packets 2-D Decomposition Structure

Exactly as in the wavelet decomposition case, the preceding one-dimensional framework can be extended to image analysis. Minor direct modifications lead to quaternary tree related definitions. An example is shown below for depth 2.

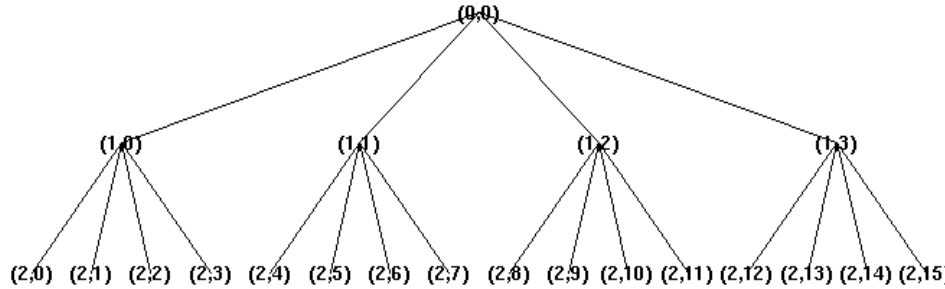


Figure 6-42: Quaternary tree of depth 2

Wavelet Packets for Compression and De-Noising

In the wavelet packet framework, compression and de-noising ideas are identical as those developed in the wavelet framework. The only new feature is a more complex analysis that provides increased flexibility. A single decomposition using wavelet packets generates a large number of bases. You can then look for the best representation with respect to a design objective, using the function `besttree` with an entropy function. See Chapter 5 for detail.

References

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

[BirM95] Birgé, L., P. Massart, Preprint Univ. Paris Sud, France 95.41 p. 1-32(1995), To appear in Festscheift in honor of Le Cam (D. Pollard, E. Torgersen,G. Young Eds., Springer Verlag.

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

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

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

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

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

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

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

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

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

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

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

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

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

[DonJKP95a] Donoho, D.L., I.M. Johnstone, G. Kerkyacharian, D. Picard (1995), "Wavelet shrinkage: asymptopia", *Jour. Roy. Stat. Soc.*, series B, vol. 57 no. 2, pp. 301-369.

[DonJKP95b] Donoho, D.L., I.M. Johnstone, G. Kerkyacharian, D. Picard (1995), "Density estimation by wavelet thesholding", submitted for publication to the *Annals of Stat.*

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

[Kai94] Kaiser, G. (1994),"A friendly guide to wavelets", Birkhauser.

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

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

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

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

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

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

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

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

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

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

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

7

Adding Your Own Wavelets

The Wavelet Toolbox contains a lot of wavelet families, but by using the `wavemngr` function, you can add new wavelets to the existing ones in order to implement your favorite or try out a wavelet of your own design. The toolbox allows you to define new wavelets for use with both the command line functions and the graphical tools.

Caution: This capability must be used carefully, because the toolbox does not check that your wavelet meets all the mathematical requisites.

The `wavemngr` function affords extensive wavelet management. However, this chapter focuses only on the addition of a wavelet family. For more complete information, see the `wavemngr` reference entry in Chapter 8.

This chapter discusses:

- Preparing to Add a New Wavelet Family
- How to Add a New Wavelet Family
- After Adding a New Wavelet Family

Preparing to Add a New Wavelet Family

The `wavemngr` command permits you to add new wavelets and wavelet families to the predefined ones. However, before you can use the `wavemngr` command to add a new wavelet, you must:

1. Choose the full name of the wavelet family (`fn`).
2. Choose the short name of the wavelet family (`fsn`).
3. Determine the wavelet type (`wt`).
4. Define the orders of wavelets within the given family (`nums`).
5. Build a MAT-file or a M-file (`file`).
6. *For wavelets without FIR filters*: Define the effective support.

The remainder of this section describes each of these steps.

Choose the Wavelet Family Full Name

The full name of the wavelet family, `fn`, must be a string. Predefined wavelet family names are: `Haar`, `Daubechies`, `BiorSplines`, `Coiflets`, `Symlets`, `Morlet`, `Mexican_hat`, and `Meyer`.

Choose the Wavelet Family Short Name

The short name of the wavelet family, `fsn`, must be a string of four characters or less. Predefined wavelet family short names are: `haar`, `db`, `bior`, `coif`, `sym`, `morl`, `mexh`, and `meyr`.

Determine the Wavelet Type

We distinguish four types of wavelets:

- Orthogonal wavelets with FIR filters

  These wavelets can be defined through the scaling filter `w`. Predefined families of such wavelets include: `Haar`, `Daubechies`, `Coiflets`, and `Symlets`.

- Biorthogonal wavelets with FIR filters

  These wavelets can be defined through the two scaling filters `wr` and `wd`, for reconstruction and decomposition respectively. The `BiorSplines` wavelet family is a predefined family of this type.

- Orthogonal wavelets without FIR filter but with scale function

  These wavelets can be defined through the definition of the wavelet function and the scaling function. The `Meyer` wavelet family is a predefined family of this type.

- Wavelets without FIR filter and without scale function

  These wavelets can be defined through the definition of the wavelet function. Predefined families of such wavelets include: `Morlet`, and `Mexican_hat`.

Define the Orders of Wavelets Within the Given Family

If a family contains many wavelets, the short name and the order are appended in order to form the wavelet name. Argument `nums` is a string containing the orders separated with blanks. This argument is not used for wavelets of type 3 or 4, nor is it used for a family that only has a single wavelet.

For example, for the first Daubechies wavelets,

```
fsn = 'db'
nums = '1 2 3'
```

yield the three wavelets `db1`, `db2` and `db3`.

For the first BiorSplines wavelets,

```
fsn = 'bior'
nums = '1.1 1.3 1.5 2.2'
```

yield the four wavelets `bior1.1`, `bior1.3`, `bior1.5`, and `bior2.2`.

Build a MAT-File or M-File

The `wavemngr` command requires a `file` argument, which is a string containing a MAT-file or M-file name.

If a family contains many wavelets, a M-file must be defined and must be of a specific form that depends on the wavelet type. The specific M-file formats are described in the remainder of this section.

If a family contains a single wavelet, then a MAT-file can be defined for wavelets of type 1. It must have the wavelet family short name (`fsn`) argument as its name and must contain a single variable whose name is `fsn` and whose value is the scaling filter. An M-file can also be defined as discussed below.

Type 1 (Orthogonal with FIR Filter)

The syntax of the first line in the M-file must be:

```
function w = file(wname)
```

where the input argument `wname` is a string containing the wavelet name, and the output argument `w` is the corresponding scaling filter.

The filter `w` must be of even length otherwise it is zero-padded by the toolbox.

For predefined wavelets, the scaling filter is of sum 1. For a new wavelet, the normalization is free (except 0 of course) since the toolbox uses a suitably normalized version of this filter.

Examples of such M-files for predefined wavelets are: `dbwavf.m` for Daubechies, `coifwavf.m` for Coiflets, and `symwavf.m` for Symlets.

Type 2 (Biorthogonal with FIR Filter)

The syntax of the first line in the M-file must be:

```
function [wr,wd] = file(wname)
```

where the input argument `wname` is a string containing the wavelet name and the output arguments `wr` and `wd` are the corresponding reconstruction and decomposition scaling filters, respectively.

The filters `wr` and `wd` must be of the same even length. In general, initial biorthogonal filters do not meet these requirements, so they are zero-padded by the toolbox.

For predefined wavelets, the scaling filters are of sum 1. For a new wavelet, the normalization is free (except 0 of course) since the toolbox uses a suitably normalized version of these filters.

The M-file `biorwavf.m` (for `BiorSplines`) is an example of an M-file for a type-2 predefined wavelet family.

Type 3 (Orthogonal with Scale Function)

The syntax of the first line in the M-file must be:

```
function [phi,psi,t] = file(lb,ub,n)
```

which returns values of the scaling function `phi` and of the wavelet function `psi` on a regular `n`-point grid with intervals of length `t` and bounded by `[lb ub]`.

The M-file `meyer.m` is an example of an M-file for a type-3 predefined wavelet family.

Type 4 (No FIR Filter; No Scale Function)

The syntax of the first line in the M-file must be:

```
function [psi,t] = file(lb,ub,n)
```

which returns values of the wavelet function `psi` on a regular `n`-point grid with intervals of length `t` and bounded by `[lb ub]`.

Examples of type-4 M-files for predefined wavelet families are `mexihat.m` (for `Mexican_hat`) and `morlet.m` (for `Morlet`).

Define the Effective Support

This definition is required only for wavelets of type 3 or 4, since they are not compactly supported.

Defining the effective support means specifying an upper and lower bound. For predefined wavelet families, we have:

Family	**Lower Bound (`lb`)**	**Upper Bound (`ub`)**
`Meyer`	–8	8
`Mexican_hat`	–5	5
`Morlet`	–4	4

How to Add a New Wavelet Family

To add a new wavelet, use the wavemngr command in one of two forms:

```
wavemngr('add',fn,fsn,wt,nums,file)
```

or

```
wavemngr('add',fn,fsn,wt,nums,file,b).
```

Here are a few examples to illustrate how you would use wavemngr to add some of the predefined wavelet families:

Type	Syntax
1	`wavemngr('add','Ndaubechies','ndb',1,'1 2 3 4 5','dbwavf');`
1	`wavemngr('add','Ndaubechies','ndb',1,'1 2 3 4 5 **','dbwavf');`
2	`wavemngr('add','Nbiorwavf','nbio',2,'1.1 1.3','biorwavf');`
3	`wavemngr('add','Nmeyer','nmey',3,'','meyer',[-8,8]);`
4	`wavemngr('add','Nmorlet','nmor',4,'','morlet',[-4,4]).`

Example 1

Let us take the example of Binlets proposed by Strang and Nguyen in the book *Wavelets and Filter Banks* (See pp. 216-217).

Note: The M-files used in this example can be found in the wavedemo directory.

The full family name is: Binlets.

The short name of the wavelet family is: binl.

The wavelet type is: 2 (Biorthogonal with FIR filters).

The order of the wavelet within the family is: `7.9` (we just use one in this example).

The M-file used to generate the filters is `binlwavf.m`

Then to add the new wavelet, type:

```
% Add new family of biorthogonal wavelets.
     wavemngr('add','Binlets','binl',2,'7.9','binlwavf')

% List wavelets families.
     wavemngr('read')

ans =


==================================
Haar         haar
Daubechies   db
BiorSplines  bior
Coiflets     coif
Symlets      sym
Morlet       morl
Mexican_hat  mexh
Meyer        meyr
Binlets      binl
==================================
```

If you want to get online information on this new family, you can build an associated help file which would look like:

```
function binlinfo
%BINLINFO Information on biorthogonal wavelets (binlets).
%
%       Biorthogonal Wavelets (Binlets)
%
%       Family                  Binlets
%       Short name              binl
%       Order Nr,Nd             Nr = 7 , Nd = 9
%
%       Orthogonal              no
%       Biorthogonal            yes
%       Compact support         yes
%       DWT                     possible
%       CWT                     possible
%
%       binl Nr.Nd              ld                      lr
%                         effective length       effective length
%                             of LoF_D                of HiF_D
%       binl 7.9                 7                       9
```

The associated M-file to generate the filters (`binlwavf.m`) is:

```
function [Rf,Df] = binlwavf(wname)
%BINLWAVF Biorthogonal wavelet filters (Binlets).
%       [RF,DF] = BINLWAVF(W) returns two scaling filters
%       associated with the biorthogonal wavelet specified
%       by the string W.
%       W = 'binlNr.Nd' where possible values for Nr and Nd are:
              Nr = 7  Nd = 9
%       The output arguments are filters:
%               RF is the reconstruction filter
%               DF is the decomposition filter

% Check arguments.
if errargn('binlwavf',nargin,[0 1],nargout,[0:2]), error('*');
end
```

```
% suppress the following line for extension
Nr = 7; Nd = 9;

% for possible extension
% more wavelets in 'Binlets' family
%---------------------------------
if nargin==0
        Nr = 7; Nd = 9;
elseif  isempty(wname)
        Nr = 7; Nd = 9;
else
        if isstr(wname)
                lw = length(wname);
                ab = abs(wname);
                ind = find(ab==46 | 47<ab | ab<58);
                li = length(ind);
                err = 0;
                if      li==0
                        err = 1;
                elseif  ind(1)~=ind(li)-li+1
                        err = 1;
                end
                if err==0 ,
                        wname = str2num(wname(ind));
                        if isempty(wname) , err = 1; end
                end
        end
        if err==0
                Nr = fix(wname); Nd = 10*(wname-Nr);
        else
                Nr = 0; Nd = 0;
        end
end
```

```
% suppress the following lines for extension
% and add a test for errors.
%----------------------------------------
if Nr~=7 , Nr = 7; end
if Nd~=9 , Nd = 9; end

if Nr == 7
   if Nd == 9
      Rf = [-1 0 9 16 9 0 -1]/32;
      Df = [ 1 0 -8 16 46 16 -8 0 1]/64;
   end
end
```

Example 2

In the following example, new compactly supported orthogonal wavelets are added to the toolbox. These wavelets, which are a slight generalization of the Daubechies wavelets, are based on the use of Bernstein polynomials and are due to Kateb and Lemarié in an unpublished work.

Note: The M-files used in this example can be found in the wavedemo directory.

```
% List initial wavelets families.
    wavemngr('read')

ans =
===================================
Haar         haar
Daubechies   db
BiorSplines  bior
Coiflets     coif
Symlets      sym
Morlet       morl
Mexican_hat  mexh
Meyer        meyr
===================================
```

```
% List all wavelets.
    wavemngr('read',1)

ans =
==================================
Haar        haar
==================================
Daubechies  db
----------------------------
db1     db2     db3     db4
db5     db6     db7     db8
db9     db10    dbxx
==================================
BiorSplines bior
----------------------------
bior1.1 bior1.3 bior1.5 bior2.2
bior2.4 bior2.6 bior2.8 bior3.1
bior3.3 bior3.5 bior3.7 bior3.9
bior4.4 bior5.5 bior6.8
==================================
Coiflets    coif
----------------------------
coif1   coif2   coif3   coif4
coif5
==================================
Symlets     sym
----------------------------
sym2    sym3    sym4    sym5
sym6    sym7    sym8
==================================
Morlet      morl
==================================
Mexican_hat mexh
==================================
Meyer       meyr
==================================
```

```
% Add new family of orthogonal wavelets.
% You must define:
%
%     Family Name:          Lemarie
%     Family Short Name:    lem
%     Type of wavelet:      1 (orth)
%     Wavelets numbers:     1 2 3 4 5
%     File driver:          lemwavf
%
% Add new family of orthogonal wavelets.
% You must define:
%
%     Family Name:          Lemarie
%     Family Short Name:    lem
%     Type of wavelet:      1 (orth)
%     Wavelets numbers:     1 2 3 4 5
%     File driver:          lemwavf
%
%     and the function lemwavf.m must be as follow:
%     function w = lemwavf(wname)
%     where the input argument wname is a string:
%     wname = 'lem1' or 'lem2' ... i.e.
%     wname = sh.name + number
%     and w the corresponding scaling filter
% then addition is obtained using:

    wavemngr('add','Lemarie','lem',1,'1 2 3 4 5','lemwavf');

% The ascii file 'wavelets.asc' is saved as
% 'wavelets.prv' then it is modified and
% the mat file 'wavelets.inf' is generated.
```

```
% List wavelets families.
    wavemngr('read')


ans =
==================================
Haar        haar
Daubechies  db
BiorSplines bior
Coiflets    coif
Symlets     sym
Morlet      morl
Mexican_hat mexh
Meyer       meyr
Lemarie     lem
==================================
```

After Adding a New Wavelet Family

When you use the `wavemngr` command to add a new wavelet, the toolbox creates three wavelet extension files in the current directory: the two ASCII files `wavelets.asc` and `wavelets.prv`, and the MAT-file `wavelets.inf`.

If you want to use your own extended wavelet families with the Wavelet Toolbox, you should:

1. Create a new directory specifically to hold the wavelet extension files.
2. Move the previously mentioned files into this new directory.
3. Prepend this directory to the MATLAB's directory search path (see the reference entry for the `path` command).
4. Use this same directory for subsequent modifications. Allowing many wavelet extension files to proliferate in different directories may lead to unpredictable results.
5. Define an M-file called "<*fsn*>`info.m`" (For example, see `dbinfo.m` or `morlinfo.m`).

   This file will be associated automatically with the **Wavelet Family** button in the Wavelet Display option of the graphical tools.

8

Reference

Commands Grouped by Function

Graphical User Interface Tools	
wavemenu	Start graphical user interface tools.

Wavelets: General	
biorfilt	Biorthogonal wavelet filter set.
dyaddown	Dyadic downsampling.
dyadup	Dyadic upsampling.
intwave	Integrate wavelet function psi.
orthfilt	Orthogonal wavelet filter set.
qmf	Quadrature mirror filter.
wavefun	Wavelet and scaling functions.
wfilters	Wavelet filters.
wavemngr	Wavelet manager.
wmaxlev	Maximum wavelet decomposition level.

Wavelet Families	
biorwavf	Biorthogonal spline wavelet filters.
coifwavf	Coiflets wavelet filters.
dbaux	Daubechies wavelet filters computation.
dbwavf	Daubechies wavelet filters.
mexihat	Mexican hat wavelet.
meyer	Meyer wavelet.
meyeraux	Meyer wavelet auxiliary function.
morlet	Morlet wavelet.
symwavf	Symlets wavelet filters.

Continuous Wavelet: One-Dimensional	
`cwt`	Continuous wavelet coefficients 1-D.

Discrete Wavelets: One-Dimensional	
`appcoef`	Extract 1-D approximation coefficients.
`detcoef`	Extract 1-D detail coefficients.
`dwt`	Single-level discrete 1-D wavelet transform.
`dwtper`	Single-level discrete 1-D wavelet transform (periodized).
`dwtmode`	Discrete wavelet transform extension mode.
`idwt`	Single-level inverse discrete 1-D wavelet transform.
`idwtper`	Single-level inverse discrete 1-D wavelet transform (periodized).
`upcoef`	Direct reconstruction from 1-D wavelet coefficients.
`upwlev`	Single-level reconstruction of wavelet decomposition 1-D.
`wavedec`	Multi-level wavelet decomposition 1-D.
`waverec`	Multi-level wavelet reconstruction 1-D.
`wrcoef`	Reconstruct single branch from 1-D wavelet coefficients.

Discrete Wavelets: Two-Dimensional	
appcoef2	Extract 2-D approximation coefficients.
detcoef2	Extract 2-D detail coefficients.
dwt2	Single-level discrete 2-D wavelet transform.
dwtper2	Single-level discrete 2-D wavelet transform (periodized).
dwtmode	Discrete wavelet transform extension mode.
idwt2	Single-level inverse discrete 2-D wavelet transform.
idwtper2	Single-level inverse discrete 2-D wavelet transform (periodized).
upcoef2	Direct reconstruction from 2-D wavelet coefficients.
upwlev2	Single-level reconstruction of wavelet decomposition 2-D.
wavedec2	Multi-level wavelet decomposition 2-D.
waverec2	Multi-level wavelet reconstruction 2-D.
wrcoef2	Reconstruct single branch from 2-D wavelet coefficients.

Wavelet Packet Algorithms	
besttree	Best tree (wavelet packet).
bestlevt	Best level tree (wavelet packet).
entrupd	Entropy update (wavelet packet).
wentropy	Entropy (wavelet packet).
wp2wtree	Extract wavelet tree from wavelet packet tree.
wpcoef	Wavelet packet coefficients.
wpcutree	Cut wavelet packets tree.
wpdec	Wavelet packet decomposition 1-D.
wpdec2	Wavelet packet decomposition 2-D.
wpfun	Wavelet packet functions.
wpjoin	Recompose wavelet packet.
wprcoef	Reconstruct wavelet packet coefficients.
wprec	Wavelet packet reconstruction 1-D
wprec2	Wavelet packet reconstruction 2-D.
wpsplt	Split (decompose) wavelet packet.

De-Noising and Compression for Signals and Images	
ddencmp	Default values for de-noising or compression.
thselect	Threshold selection for de-noising.
wden	Automatic 1-D de-noising using wavelets.
wdencmp	De-noising or compression using wavelets.
wnoise	Generate noisy wavelet test data.
wnoisest	Estimate noise of wavelet coefficients 1-D.
wpdencmp	De-noising or compression using wavelet packets.
wpthcoef	Wavelet packet coefficients thresholding.
wthcoef	Wavelet coefficients thresholding 1-D.
wthcoef2	Wavelet coefficients thresholding 2-D.
wthresh	Perform soft or hard thresholding.

Tree Management Utilities	
allnodes	Tree nodes.
depo2ind	Node depth-position to node index.
ind2depo	Node index to node depth-position.
isnode	True for existing node.
istnode	True for terminal nodes.
maketree	Make tree.
nodeasc	Node ascendants.
nodedesc	Node descendants.
nodejoin	Recompose node.
nodepar	Node parent.
nodesplt	Split (decompose) node.
ntnode	Number of terminal nodes.
plottree	Plot tree.
tnodes	Terminal nodes.
treedpth	Tree depth.
treeord	Tree order.
wdatamgr	Manager for data structure.
wtreemgr	Manager for tree structure.

General Utilities	
deblankl	Convert string to lowercase without blanks.
errargn	Check function arguments number.
errargt	Check function arguments type.
num2mstr	Convert number to string in maximum precision.
wcodemat	Extended pseudocolor matrix scaling.
wcommon	Find common elements.
wkeep	Keep part of a vector or a matrix.
wrev	Flip vector.

Other	
`instdfft`	Inverse nonstandard 1-D fast Fourier transform.
`nstdfft`	Nonstandard 1-D fast Fourier transform.

Wavelets Information	
`waveinfo`	Information on wavelets.

Demos	
`wavedemo`	Wavelet toolbox demos.

allnodes

Purpose

Tree nodes.

Syntax

```
N = allnodes(T)
N = allnodes(T,'deppos')
```

Description

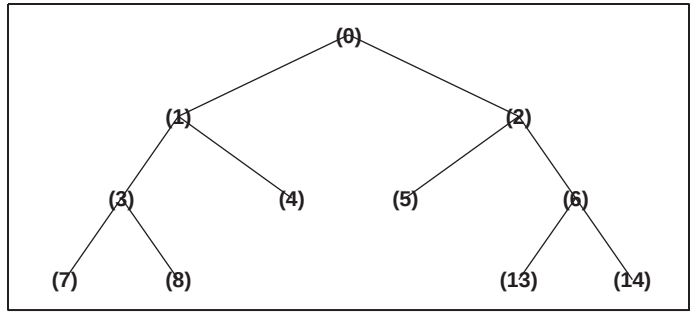
`allnodes` is a tree management utility that returns one of two node descriptions: either indices, or depths and positions. Tree nodes are numbered from left to right and from top to bottom. The root index is 0.

`N = allnodes(T)` returns in column vector `N` the indices of all the nodes of the tree structure `T`.

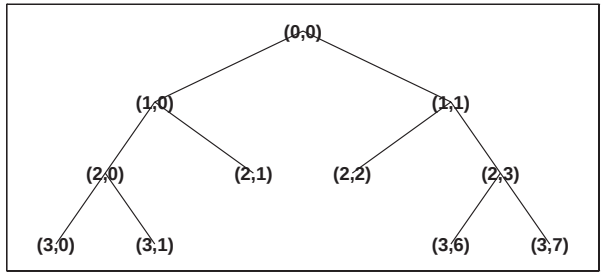
`N = allnodes(T,'deppos')` returns in matrix `N` the depths and positions of all the nodes. `N(i,1)` is the depth and `N(i,2)` the position of the node `i`.

Examples

```
% Create initial tree.
    ord = 2;
    t = maketree(ord,3);    % Binary tree of depth 3.
    tt = nodejoin(t,5);
    tt = nodejoin(tt,4);
    plottree(tt)
```



Node indices



Node depth and position

```
% List tt nodes (index).
    aln_ind = allnodes(tt)
aln_ind =
     0
     1
     2
     3
     4
     5
     6
     7
     8
    13
    14
% List tt nodes (depth-position).
    aln_depo = allnodes(tt,'deppos')
aln_depo =
     0   0
     1   0
     1   1
     2   0
     2   1
     2   2
     2   3
     3   0
     3   1
     3   6
     3   7
```

See Also

maketree

appcoef

Purpose

Extract 1-D approximation coefficients.

Syntax

```
A = appcoef(C,L,'wname',N)
A = appcoef(C,L,'wname')
A = appcoef(C,L,Lo_R,Hi_R)
A = appcoef(C,L,Lo_R,Hi_R,N)
```

Description

`appcoef` is a one-dimensional wavelet analysis function.

`appcoef` computes the approximation coefficients of a one-dimensional signal.

`A = appcoef(C,L,'wname',N)` computes the approximation coefficients at level `N` using the wavelet decomposition structure `[C,L]` (see `wavedec`).

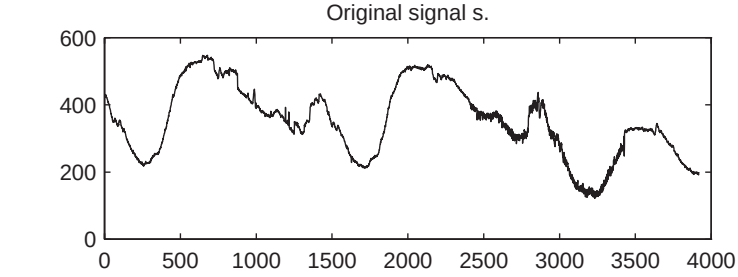
'*wname*' is a string containing the wavelet name. Level `N` must be an integer such that `0 <= N <= length(L)-2`.

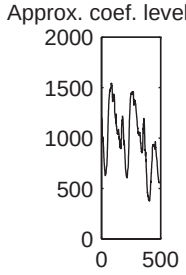
`A = appcoef(C,L,'wname')` extracts the approximation coefficients at the last level `length(L)-2`.

Instead of giving the wavelet name, you can give the filters. For `A = appcoef(C,L,Lo_R,Hi_R)` or `A = appcoef(C,L,Lo_R,Hi_R,N)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

Examples

```
% Load original one-dimensional signal.
    load leleccum; s = leleccum(1:3920); ls = length(s);

% Perform decomposition at level 3 of s using db1.
    [c,l] = wavedec(s,3,'db1');

% Extract approximation coefficients at level 3, from the
% wavelet decomposition structure [c,l].
    ca3 = appcoef(c,l,'db1',3);
```





Algorithm

The input vectors `C` and `L` contain all the information about the signal decomposition.

Let `NMAX = length(L)-2`, then `C = [A(NMAX) D(NMAX) ... D(1)]`, where `A` and the `D` are vectors.

If `N = NMAX` a simple extraction is done, otherwise `appcoef` computes iteratively the approximation coefficients using the inverse wavelet transform.

See Also

`detcoef`, `wavedec`

appcoef2

Purpose

Extract 2-D approximation coefficients.

Syntax

```
A = appcoef2(C,S,'wname',N)
A = appcoef2(C,S,'wname')
A = appcoef2(C,S,Lo_R,Hi_R)
A = appcoef2(C,S,Lo_R,Hi_R,N)
```

Description

`appcoef2` is a two-dimensional wavelet analysis function.

`appcoef2` computes the approximation coefficients of a two-dimensional signal.

`A = appcoef2(C,S,'wname',N)` computes the approximation coefficients at level `N` using the wavelet decomposition structure `[C,S]` (see `wavedec2`).

'*wname*' is a string containing the wavelet name. Level `N` must be an integer such that `0 <= N <= size(S,1)-2`.

`A = appcoef2(C,S,'wname')` extracts the approximation coefficients at the last level `size(S,1)-2`.

Instead of giving the wavelet name, you can give the filters. For `A = appcoef2(C,S,Lo_R,Hi_R)` or `A = appcoef2(C,S,Lo_R,Hi_R,N)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

Examples

```
% Load original image.
    load woman;
    % X contains the loaded image.

% Perform decomposition at level 2
% of X using db1.
    [c,s] = wavedec2(X,2,'db1');
    sizex = size(X)
```

```
sizex =
    256 256
    sizec = size(c)

sizec =
     1     65536
    val_s = s

val_s =
    64     64
    64     64
   128    128
   256    256

% Extract approximation coefficients
% at level 2.
    ca2 = appcoef2(c,s,'db1',2);
    sizeca2 = size(ca2)

sizeca2 =
    64     64

% Compute approximation coefficients
% at level 1.
    ca1 = appcoef2(c,s,'db1',1);
    sizeca1 = size(ca1)

sizeca1 =
   128    128
```

Algorithm

The algorithm is built on the same principle as appcoef.

See Also

detcoef2, wavedec2

bestlevt

Purpose

Best level tree (wavelet packet).

Syntax

```
[T,D] = bestlevt(T,D)
[T,D,E] = bestlevt(T,D)
```

Description

`bestlevt` is a one- or two-dimensional wavelet packet analysis function.

`bestlevt` computes the optimal complete sub-tree of an initial tree with respect to an entropy type criterion. The resulting complete tree may be of smaller depth than the initial one.

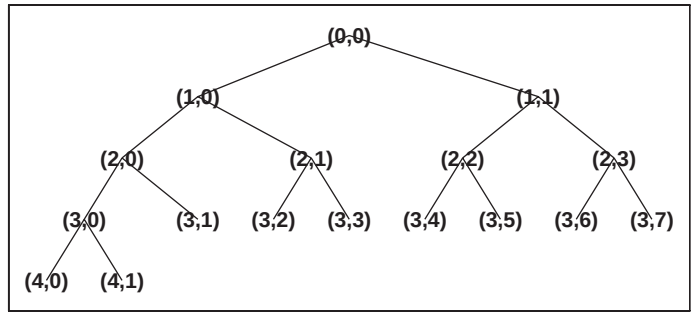
`[T,D] = bestlevt(T,D)` computes the modified tree structure `T` and data structure `D`, corresponding to the best level tree decomposition.

`[T,D,E] = bestlevt(T,D)` returns the best tree `T`, data structure `D`, and in addition, the best entropy value `E`.

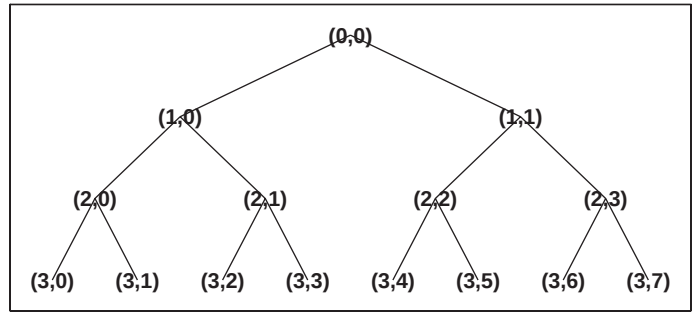
Examples

```
% Load signal.
    load noisdopp;
    x = noisdopp;

% Decompose x at depth 3 with db1 wavelet packets, using
% default entropy (shannon) and decompose the packet [3 0].
    [wpt,wpd] = wpdec(x,3,'db1');
    [wpt,wpd] = wpsplt(wpt,wpd,[3 0]);
```

```
% Plot wavelet packet tree structure wpt.
    plottree(wpt)
```




```
% Compute best level tree.
    [blt,bld] = bestlevt(wpt,wpd);

% Plot best level tree structure blt.
    plottree(blt)
```



Algorithm

See `besttree` algorithm section. The only difference is that the optimal tree is searched among the complete sub-trees of the initial tree.

See Also

`besttree`, `maketree`, `wentropy`, `wpdec`, `wpdec2`

besttree

Purpose

Best tree (wavelet packet).

Syntax

```
[T,D] = besttree(T,D)
[T,D,E] = besttree(T,D)
[T,D,E,N] = besttree(T,D)
```

Description

`besttree` is a one- or two-dimensional wavelet packet analysis function that computes the optimal sub-tree of an initial tree with respect to an entropy type criterion. The resulting tree may be much smaller than the initial one.

Following the organization of the wavelet packets library, it is natural to count the decompositions issued from a given orthogonal wavelet. As a result, a signal of length $N = 2^L$ can be expanded in at most 2^N different ways, the number of binary sub-trees of a complete binary sub-tree of depth L. As this number may be very large, and since explicit enumeration is generally intractable, it is interesting to find an optimal decomposition with respect to a convenient criterion, computable by an efficient algorithm. We are looking for a minimum of the criterion.

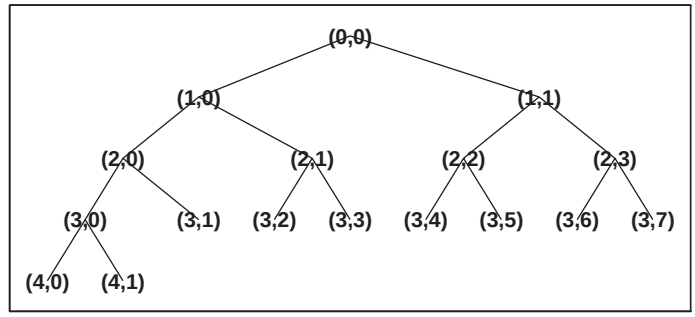
`[T,D] = besttree(T,D)` computes the modified tree structure `T` and data structure `D` (see `maketree`), corresponding to the best entropy value.

`[T,D,E] = besttree(T,D)` returns the best tree `T`, the data structure `D`, and in addition, the best entropy value `E`.

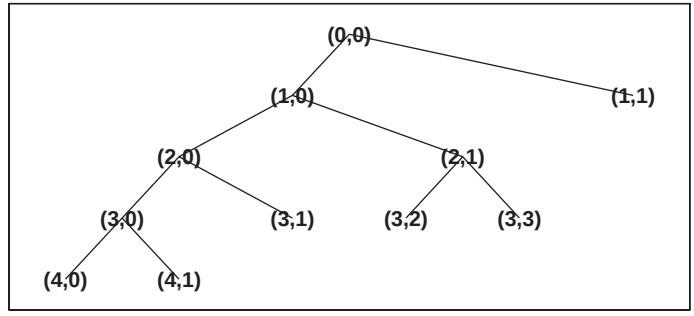
`[T,D,E,N] = besttree(T,D)` returns the best tree `T`, the data structure `D`, the best entropy value `E`, and in addition, the vector `N` containing the indices of the merged nodes.

Examples

```
% Load signal.
    load noisdopp; x = noisdopp;
% Decompose x at depth 3 with db1 wavelet packets, using
% default entropy (shannon) and decompose the packet [3 0].
    [wpt,wpd] = wpdec(x,3,'db1');
    [wpt,wpd] = wpsplt(wpt,wpd,[3 0]);

% Plot wavelet packet tree structure wpt.
    plottree(wpt)
```




```
% Compute best tree.
    [bt,bd] = besttree(wpt,wpd)
% Plot best tree structure bt.
    plottree(bt)
```



Algorithm

Consider the one-dimensional case. Starting with the root node, the best tree is calculated using the following scheme. A node N is split into two nodes N1 and N2 if and only if the sum of the entropy of N1 and N2 is lower than the entropy of N. This is a local criterion based only on the information available at the node N.

Several entropy type criteria can be used (see `wentropy`). If the entropy function is an additive function along the wavelet packet coefficients, this Algorithm leads to the best tree.

Starting from an initial tree T and using the merging side of this algorithm, we obtain the best tree among all the binary sub-trees of T.

See Also

bestlevt, maketree, wentropy, wpdec, `wpdec2`

References

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

Purpose

Biorthogonal wavelet filter set.

Syntax

```
[LO_D,HI_D,LO_R,HI_R] = biorfilt(DF,RF)
[LO_D1,HI_D1,LO_R1,HI_R1,LO_D2,HI_D2,LO_R2,HI_R2] =
   biorfilt(DF,RF,'8')
```

Description

The `biorfilt` command returns either four or eight filters associated with biorthogonal wavelets.

`[LO_D,HI_D,LO_R,HI_R] = biorfilt(DF,RF)` computes four filters associated with the biorthogonal wavelet specified by decomposition filter `DF` and reconstruction filter `RF`. These filters are:

`LO_D`	Decomposition low-pass filter
`HI_D`	Decomposition high-pass filter
`LO_R`	Reconstruction low-pass filter
`HI_R`	Reconstruction high-pass filter

`[LO_D1,HI_D1,LO_R1,HI_R1,LO_D2,HI_D2,LO_R2,HI_R2] = biorfilt(DF,RF,'8')` returns eight filters, the first four associated with the decomposition wavelet, and the last four associated with the reconstruction wavelet.

It is well known in the sub-band filtering community that if the same FIR filters are used for reconstruction and decomposition, then symmetry and exact reconstruction are incompatible (except with the Haar wavelet). Therefore, with biorthogonal filters, two wavelets are introduced instead of just one:

- One wavelet, $\tilde{\psi}$, is used in the analysis, and the coefficients of a signal s are $\tilde{c}_{j,k} = \int s(x)\tilde{\psi}_{j,k}(x)dx$,
- The other wavelet, ψ, is used in the synthesis $s = \sum_{j,k} \tilde{c}_{j,k}\psi_{j,k}$

Further, the two wavelets are related by duality in the following sense: $\int \tilde{\psi}_{j,k}(x)\psi_{j',k'}(x)dx = 0$ as soon as $j \neq j'$ or $k \neq k'$ and

$\int \tilde{\phi}_{0,k}(x)\phi_{0,k'}(x)dx = 0$ as soon as $k \neq k'$.

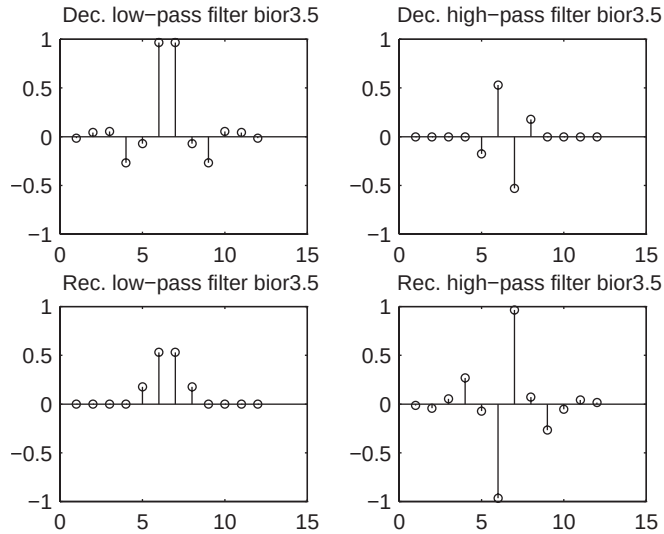
It becomes apparent, as A. Cohen pointed out in his thesis (p. 110), that "the useful properties for analysis (e.g., oscillations, null moments) can be concentrated in the $\tilde{\psi}$ function whereas the interesting properties for synthesis (regularity) are assigned to the ψ function. The separation of these two tasks proves very useful."

$\tilde{\psi}$ and ψ can have very different regularity properties, ψ being more regular than $\tilde{\psi}$ (see Daubechies p. 269).

The $\tilde{\psi}$, ψ, $\tilde{\phi}$ and ϕ functions are zero outside a segment.

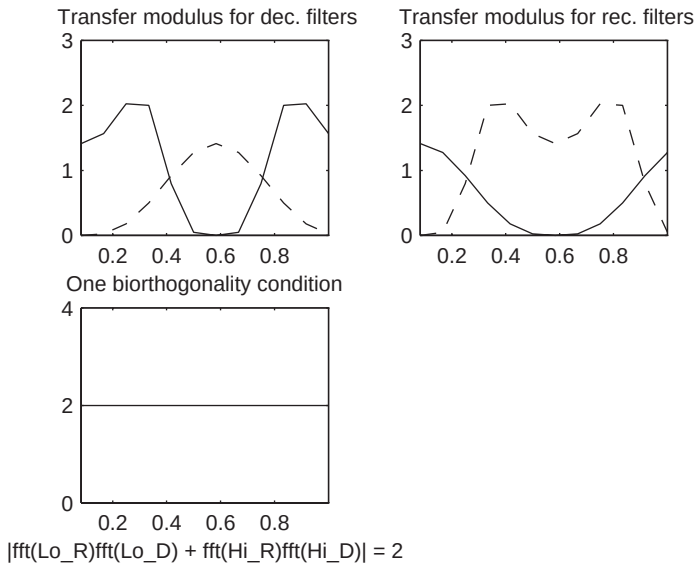
Examples

```
% Compute the four filters associated with spline biorthogonal
% wavelet 3.5: bior3.5.

% Find the two scaling filters associated with bior3.5.
[Rf,Df] = biorwavf('bior3.5');
% Compute the four filters needed.
[Lo_D,Hi_D,Lo_R,Hi_R] = biorfilt(Df,Rf);
subplot(221); stem(Lo_D);
title('Dec. low-pass filter bior3.5');
subplot(222); stem(Hi_D);
title('Dec. high-pass filter bior3.5');
subplot(223); stem(Lo_R);
title('Rec. low-pass filter bior3.5');
subplot(224); stem(Hi_R);
title('Rec. high-pass filter bior3.5');
```




```
% Orthogonality by dyadic translation is lost.
    nzer = [Lo_D 0 0]*[0 0 Lo_D]'
nzer =
    -0.6881
    nzer = [Hi_D 0 0]*[0 0 Hi_D]'
nzer =
    0.1875

% But using duality we have:
    zer = [Lo_D 0 0]*[0 0 Lo_R]'
zer =
    -2.7756e-17
    zer = [Hi_D 0 0]*[0 0 Hi_R]'

zer =
    2.7756e-17

% But perfect reconstruction via DWT is preserved.
    x = randn(1,500);
    [a,d] = dwt(x,Lo_D,Hi_D);
    xrec = idwt(a,d,Lo_R,Hi_R);
    err = norm(x-xrec)
err =
    5.0218e-15
```

```
% High and low frequency illustration.
    fftld = fft(Lo_D); ffthd = fft(Hi_D);
    freq = [1:length(Lo_D)]/length(Lo_D);
    subplot(221); plot(freq,abs(fftld),freq,abs(ffthd));
    title('Transfer modulus for dec. filters')
    fftlr = fft(Lo_R); ffthr = fft(Hi_R);
    freq = [1:length(Lo_R)]/length(Lo_R);
    subplot(222); plot(freq,abs(fftlr),freq,abs(ffthr));
    title('Transfer modulus for rec. filters')
    subplot(223); plot(freq, abs(fftlr.*fftld + ffthd.*ffthr));
    title('One biorthogonality condition')
    xlabel('|fft(Lo_R)fft(Lo_D) + fft(Hi_R)fft(Hi_D)| = 2')
```



Note: For biorthogonal wavelets, the filters for decomposition and reconstruction are in general of different odd lengths. This situation occurs, for example, for “splines” biorthogonal wavelets used in the toolbox, where the four filters are zero-padded to have the same even length.

See Also

`biorwavf`, `orthfilt`

References

A. Cohen (1992) “Ondelettes, analyses multirésolution et traitement numérique du signal,” *Ph. D. Thesis*, University of Paris IX, DAUPHINE.

I. Daubechies (1992), “Ten lectures on wavelets,” CBMS-NSF conference series in applied mathematics. SIAM Ed.

biorwavf

Purpose

Biorthogonal spline wavelet filters.

Syntax

```
[RF,DF] = biorwavf(W)
```

Description

`[RF,DF] = biorwavf(W)` returns two scaling filters associated with biorthogonal wavelet specified by the string `W`.

`W = 'biorNr.Nd'` where possible values for `Nr` and `Nd` are:

`Nr` = 1	`Nd` = 1 , 3 or 5
`Nr` = 2	`Nd` = 2 , 4 , 6 or 8
`Nr` = 3	`Nd` = 1 , 3 , 5 , 7 or 9
`Nr` = 4	`Nd` = 4
`Nr` = 5	`Nd` = 5
`Nr` = 6	`Nd` = 8

The output arguments are filters:

- `RF` is the reconstruction filter.
- `DF` is the decomposition filter.

Examples

```
% Set spline biorthogonal wavelet name.
    wname = 'bior2.2';

% Compute the two corresponding scaling filters,
% rf is the reconstruction scaling filter and
% df is the decomposition scaling filter.
    [rf,rd] = biorwavf(wname)

rf =
         0   0.2500   0.5000   0.2500       0   0

df =
   -0.1250   0.2500   0.7500   0.2500 -0.1250   0
```

See Also

`biorfilt`, `waveinfo`

Purpose

Coiflets wavelets filters.

Syntax

```
F = coifwavf(W)
```

Description

`F = coifwavf(W)` returns the scaling filter associated with coiflet wavelet specified by the string `W`, where `W = 'coifN'`. Possible values for `N` are: 1, 2, 3, 4 or 5.

Examples

```
% Set coiflet wavelet name.
    wname = 'coif2';

% Compute the corresponding scaling filter.
    f = coifwavf(wname)

f =
Columns 1 through 7
0.0116  -0.0293  -0.0476 0.2730 0.5747  0.2949  -0.0541

Columns 8 through 12
-0.0420  0.0167  0.0040 -0.0013 -0.0005
```

See Also

waveinfo

cwt

Purpose

Continuous 1-D wavelet coefficients.

Syntax

```
coefs = cwt(s,scales,'wname')
coefs = cwt(s,scales,'wname','plot')
```

Description

`cwt` is a one-dimensional wavelet analysis function.

`coefs = cwt(s,scales,'wname')` computes the continuous wavelet coefficients of the vector `s` at real, positive `scales`, using the wavelet whose name is '*wname*' (see `waveinfo`).

`coefs = cwt(s,scales,'wname','plot')` computes, and in addition plots, the continuous wavelet transform coefficients.

Let s be the signal and ψ the wavelet. Then the wavelet coefficient of s at scale a and position b is defined by:

$$C_{a,b} = \int_R s(t)\frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)dt$$

since $s(t)$ is a discrete signal, we use a piecewise constant interpolation of the $s(k)$ values, $k = 1$ to `length(s)`.

Then for any strictly positive scale a, we compute $C_{a,b}$ for $b = 1$ to `length(s)`.

Output argument `coefs` contains the wavelet coefficients for the scales within the vector `scales` in the same order, stored rowwise.

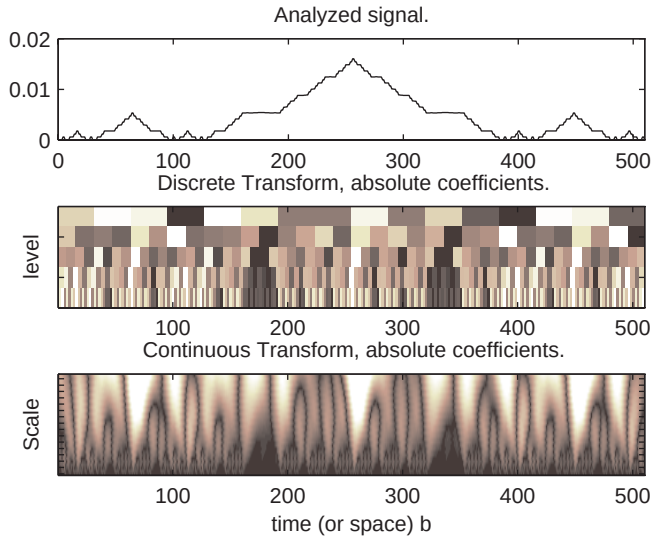
Examples of valid uses are:

```
c = cwt(s,1:32,'meyr')
c = cwt(s,[64 32 16:-2:2],'morl')
c = cwt(s,[3 18 12.9 7 1.5],'db2')
```

Examples

This example demonstrates the difference between discrete and continuous wavelet transforms.

```
% Load original fractal signal.
    load vonkoch
    vonkoch=vonkoch(1:510);
    lv = length(vonkoch);

    subplot(311), plot(vonkoch);title('Analyzed signal.');
    set(gca,'Xlim',[0 510])

% Perform discrete wavelet transform at level 5 by sym2.
% Levels 1 to 5 correspond to scales 2, 4, 8, 16 and 32.
    [c,l] = wavedec(vonkoch,5,'sym2');

% Expand discrete wavelet coefficients for plot.
% Levels 1 to 5 correspond to scales 2, 4, 8, 16 and 32.
    cfd = zeros(5,lv);
    for k = 1:5
        d = detcoef(c,l,k);
        d = d(ones(1,2^k),:);
        cfd(k,:) = wkeep(d(:)',lv);
    end
    cfd = cfd(:);
    I = find(abs(cfd)<sqrt(eps));
    cfd(I)=zeros(size(I));
    cfd = reshape(cfd,5,lv);

% Plot discrete coefficients.
    subplot(312), colormap(pink(64));
    img = image(flipud(wcodemat(cfd,64,'row')));
    set(get(img,'parent'),'YtickLabels',[]);
    title('Discrete Transform, absolute coefficients.')
    ylabel('level')

% Perform continuous wavelet transform by sym2 at all integer
% scales from 1 to 32.
    subplot(313)
    ccfs = cwt(vonkoch,1:32,'sym2','plot');
    title('Continuous Transform, absolute coefficients.')
    colormap(pink(64));
    ylabel('Scale')
```



Algorithm

$$C_{a,b} = \int_R s(t)\frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)dt$$

$$C_{a,b} = \sum_k \int_k^{k+1} s(t)\frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)dt$$

since $s(t) = s(k)$, if $t \in [k, k+1]$ then

$$C_{a,b} = \frac{1}{\sqrt{a}}\sum_k s(k)\int_k^{k+1}\psi\left(\frac{t-b}{a}\right)dt$$

$$C_{a,b} = \frac{1}{\sqrt{a}}\sum_k s(k)\left(\int_{-\infty}^{k+1}\psi\left(\frac{t-b}{a}\right)dt - \int_{-\infty}^{k}\psi\left(\frac{t-b}{a}\right)dt\right)$$

so at any scale a, the wavelet coefficients $C_{a,b}$ for $b = 1$ to `length`(s) can be obtained by convolving the signal s and a dilated and translated version of the integrals of the form $\int_{-\infty}^{k} \psi(t)dt$ (given by `intwave`), and taking finite difference using `diff`.

See Also

`wavedec`, `wavefun`, `waveinfo`, `wcodemat`

dbaux

Purpose

Daubechies wavelets filters computation.

Syntax

```
W = dbaux(N,SUMW)
W = dbaux(N)
```

Description

`W = dbaux(N,SUMW)` is the order `N` Daubechies scaling filter such that `sum(W) = SUMW`. Possible values for `N` are: 1, 2, 3, ...

Note: Instability may occur when `N` is too large.

`W = dbaux(N)` is equivalent to `W = dbaux(N,1)`.

`W = dbaux(N,0)` is equivalent to `W = dbaux(N,1)`.

Examples

```
% P the “Lagrange a trous” filter for N=2 is explicit
% and given by:
    P = [ -1/16 0 9/16 1 9/16 0 -1/16]

P =
   -0.0625        0  0.5625  1.0000  0.5625        0 -0.0625

% The db2 Daubechies scaling filter w, is a
% solution of the equation: P = conv(wrev(w),w) * 2.
%
% This filter P is symmetric, easy to generate, and w is
% a minimum phase solution of the previous equation,
% based on the roots of P.
    rP = roots(P);

% Retaining only the root inside the unit circle (here it
% is the sixth value of rP), and two roots located at -1,
% we obtain the Daubechies wavelet of order 2:
    ww = poly([rP(6) -1 -1]);    % filter construction
    ww = ww / sum(ww)        % normalize sum
```

```
ww =
    0.3415  0.5915  0.1585 -0.0915

% Check that ww is correct and equal to
% the db2 Daubechies scaling filter w.
    w = dbaux(2)

w =
    0.3415  0.5915  0.1585 -0.0915
```

Algorithm

The algorithm used is based on a result obtained by Shensa, showing a correspondence between the "Lagrange a trous" filters and the convolutional squares of the Daubechies wavelet filters.

The computation of the order *N* Daubechies scaling filter *w* proceeds in two steps: compute a "Lagrange a trous" filter *P* and extract a square root. More precisely:

- P the associated "Lagrange a trous" filter is a symmetric filter of length 4N-1. P is defined by:

  $P = [a(N) \quad 0 \quad a(N\text{-}1) \quad 0 \ldots 0 \quad a(1) \quad 1 \quad a(1) \quad 0 \quad a(2) \quad 0 \ldots 0 \quad a(N)]$

  $$\text{where } a(k) = \frac{\prod_{\substack{i=-N+1 \\ i \neq k}}^{N} \left(\frac{1}{2} - i\right)}{\prod_{\substack{i=-N+1 \\ i \neq k}}^{N} (k - i)} \text{ for } k = 1,\ldots,N$$

- Then, if *w* denotes db*N* Daubechies scaling filter of sum $\sqrt{2}$, *w* is a square root of *P*. More precisely *P* = conv(wrev(w),w), and *w* is a filter of length 2*N*. The corresponding polynomial has *N* zeros located at -1 and *N-1* zeros less than 1 in modulus.

Note that other methods can be used; see various solutions of the spectral factorization problem in Strang-Nguyen p. 157.

Limitations

The computation of the db*N* Daubechies scaling filter requires the extraction of the roots of a polynomial of order `4N`. Instability may occur when `N` is too large.

See Also

`dbwavf`, `wfilters`

References

I. Daubechies (1992), "Ten lectures on wavelets," CBMS-NSF conference series in applied mathematics. SIAM Ed.

M.J. Shensa (1992), "The discrete wavelet transform: wedding the a trous and Mallat Algorithms," *IEEE Trans. on Signal Processing*, vol. 40, 10, pp 2464-2482.

G. Strang, T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press.

Purpose

Daubechies wavelets filters.

Syntax

```
F = dbwavf(W)
```

Description

`F = dbwavf(W)` returns the scaling filter associated with Daubechies wavelet specified by the string `W`, where `W = 'dbN'`. Possible values for `N` are: 1, 2, 3, ..., 50.

Examples

```
% Set Daubechies wavelet name.
    wname = 'db4';

% Compute the corresponding scaling filter.
    f = dbwavf(wname)

f =
Columns 1 through 7
0.1629  0.5055  0.4461 -0.0198 -0.1323  0.0218  0.0233
Column 8
-0.0075
```

See Also

`dbaux`, `waveinfo`, `wfilters`

ddencmp

Purpose

Default values for de-noising or compression.

Syntax

```
[THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,IN2,X)
[THR,SORH,KEEPAPP] = ddencmp(IN1,'wv',X)
[THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,'wp',X)
```

Description

`ddencmp` is a de-noising and compression oriented function.

`ddencmp` gives default values for all the general procedures related to de-noising and compression of one- or two-dimensional signals, using wavelets or wavelet packets.

`[THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,IN2,X)` returns default values for de-noising or compression, using wavelets or wavelet packets, of an input vector or matrix `X`, which can be a one- or two-dimensional signal. `THR` is the threshold, `SORH` is for soft or hard thresholding, `KEEPAPP` allows you to keep approximation coefficients, and `CRIT` (used only for wavelet packets) is the entropy name (see `wentropy`).

`IN1` is `'den'` or `'cmp'`.

`IN2` is `'wv'` or `'wp'`.

For wavelets (three output arguments):

`[THR,SORH,KEEPAPP] = ddencmp(IN1,'wv',X)` returns default values for de-noising (if `IN1 = 'den'`) or compression (if `IN1 = 'cmp'`) of `X`. These values can be used for `wdencmp`.

For wavelet packets (four output arguments):

`[THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,'wp',X)` returns default values de-noising (if `IN1 = 'den'`) or compression (if `IN1 = 'cmp'`) of `X`. These values can be used for `wpdencmp`.

Examples

```
% Generate Gaussian white noise.
    init = 2055415866; randn('seed',init);
    x = randn(1,1000);

% Find default values for wavelets (3 output arguments).
% These values can be used for wdencmp with option 'gbl'.
    % default for de-noising:
    % soft thresholding and appr. cfs. kept
    % thr = sqrt(2*log(n)) * s
    % where s is an estimate of level noise.
    [thr,sorh,keepapp] = ddencmp('den','wv',x)

thr =
    3.8593
sorh =
s
keepapp =
    1

    % default for compression:
    % hard thresholding and appr. cfs. kept
    % thr = median(abs(detail at level 1)) if nonzero
    % else thr = 0.05 * max(abs(detail at level 1)).
    [thr,sorh,keepapp] = ddencmp('cmp','wv',x)

thr =
    0.7003
sorh =
h
keepapp =
    1

% Find default values for wavelet packets (4 output arguments).
% These values can be used for wpdencmp.
    % default for de-noising:
    % soft thresholding and appr. cfs. kept
    % thr = sqrt(2*log(n*log(n)/log(2)))
    % the noise level is supposed to be equal to 1;
    % default entropy is 'sure' criterion.
    [thr,sorh,keepapp,crit] = ddencmp('den','wp',x)
```

```
thr =
    4.2911

sorh =
h
keepapp =
    1
crit =
sure
    % default for compression.
    % hard thresholding and appr. cfs. kept
    % thr = median(abs(detail at level 1))
    % default entropy is 'threshold' criterion.
    [thr,sorh,keepapp,crit] = ddencmp('cmp','wp',x)

thr =
    0.7003
sorh =
h
keepapp =
    1
crit =
threshold
```

See Also

wdencmp, wentropy, wpdencmp

References

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

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

D.L. Donoho, I.M. Johnstone, "Ideal de-noising in an orthonormal basis chosen from a library of bases," *C.R.A.S. Paris*, t. 319, Ser. I, pp. 1317–1322.

Purpose

Convert string to lowercase without blanks.

Syntax

```
S = deblankl(X)
```

Description

`deblankl` is a general utility.

This function gives flexibility when using strings.

`S = deblankl(X)` is the string `X` converted to lowercase without blanks.

Examples

```
x = 'AB1 C %9'

x =
     AB1 C %9

y = deblankl(x)

y =
     ab1c%9
```

depo2ind

Purpose

Node depth-position to node index.

Syntax

```
N = depo2ind(O,[D P])
```

Description

depo2ind is a tree management utility.

For a tree of order O, `N = depo2ind(O,[D P])` computes the indices N of the nodes whose depths and positions are encoded within [D,P].

D, P and N are column vectors. The values of D, P and N are constrained by:

D = depths, $0 \le D \le dmax$

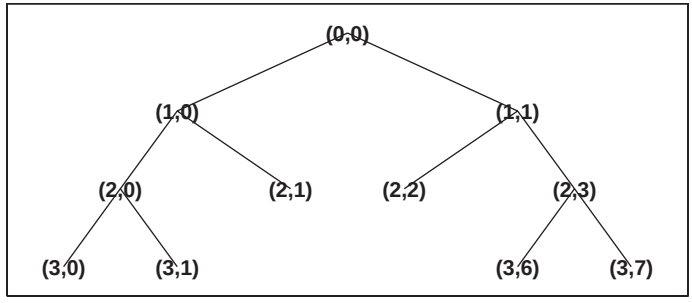
P = positions at depth D, $0 \le P \le order^{D-1}$

N = indices, $0 \le N < (order^{(dmax+1)}-1)/(order-1)$

Note that for a column vector X, we have `depo2ind(O,X) = X`.

Examples

```
% Create initial tree.
    ord = 2;
    t = maketree(ord,3);     % binary tree of depth 3.
    tt = nodejoin(t,5);
    tt = nodejoin(tt,4);
    plottree(tt)
```




```
% List tt nodes (depth-position).
    aln_depo = allnodes(tt,'deppos')
```

```
aln_depo =
    0     0
    1     0
    1     1
    2     0
    2     1
    2     2
    2     3
    3     0
    3     1
    3     6
    3     7

% Switch from depth-position to index.
    aln_ind = depo2ind(ord,aln_depo)

aln_ind =
    0
    1
    2
    3
    4
    5
    6
    7
    8
   13
   14
```

See Also

ind2depo, maketree, wtreemgr

detcoef

Purpose

Extract 1-D detail coefficients.

Syntax

```
D = detcoef(C,L,N)
D = detcoef(C,L)
```

Description

`detcoef` is a one-dimensional wavelet analysis function.

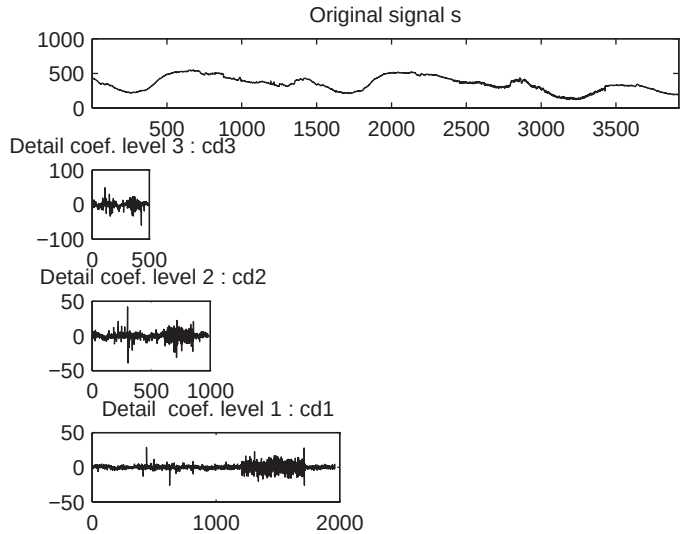
`D = detcoef(C,L,N)` extracts the detail coefficients at level `N` from the wavelet decomposition structure `[C,L]` (see `wavedec`). Level `N` must be an integer such that `1 <= N <= length(L)-2`.

`D = detcoef(C,L)` extracts the detail coefficients at last level `n=length(L)-2`.

Examples

```
% Load original one-dimensional signal.
    load leleccum;
    s = leleccum(1:3920);
    ls = length(s);

% Perform decomposition at level 3 of s using db1.
    [c,l] = wavedec(s,3,'db1');
```

```
% Extract detail coefficients at levels
% 1, 2 and 3, from wavelet decomposition
% structure [c,l].
    cd3 = detcoef(c,l,3);
    cd2 = detcoef(c,l,2);
    cd1 = detcoef(c,l,1);
```



See Also `appcoef`, `wavedec`

detcoef2

Purpose

Extract 2-D detail coefficients.

Syntax

```
D = detcoef2(O,C,S,N)
```

Description

`detcoef2` is a two-dimensional wavelet analysis function.

`D = detcoef2(O,C,S,N)` extracts from the wavelet decomposition structure `[C,S]` (see `wavedec2`), the horizontal, vertical, or diagonal detail coefficients for `O = 'h'`(or `'v'` or `'d'`, respectively), at level `N`.

Level `N` must be an integer such that `1 <= N <= size(S,1)-2`.

Examples

```
% Load original image.
    load woman;
    % X contains the loaded image.

% Perform decomposition at level 2
% of X using db1.
    [c,s] = wavedec2(X,2,'db1');

    sizex = size(X)
sizex =
    256    256

    sizec = size(c)
sizec =
    1    65536

    val_s = s
val_s =
     64     64
     64     64
    128    128
    256    256
```

```
% Extract details coefficients at level 2
% in each orientation, from wavelet decomposition
% structure [c,s].
    chd2 = detcoef2('h',c,s,2);
    cvd2 = detcoef2('v',c,s,2);
    cdd2 = detcoef2('d',c,s,2);

    sizecd2 = size(chd2)

sizecd2 =
     64     64

% Extract details coefficients at level 1
% in each orientation, from wavelet decomposition
% structure [c,s].
    chd1 = detcoef2('h',c,s,1);
    cvd1 = detcoef2('v',c,s,1);
    cdd1 = detcoef2('d',c,s,1);

    sizecd1 = size(chd1)

sizecd1 =
    128    128
```

See Also

appcoef2, wavedec2

dwt

Purpose

Single-level discrete 1-D wavelet transform.

Syntax

```
[cA,cD] = dwt(X,'wname')
[cA,cD] = dwt(X,Lo_D,Hi_D)
```

Description

The `dwt` command performs a single-level one-dimensional wavelet decomposition with respect to either a particular wavelet (*'wname'*, see `wfilters`) or particular wavelet decomposition filters (`Lo_D` and `Hi_D`) you specify.

`[cA,cD] = dwt(X,'wname')` computes the approximation coefficients vector `cA` and detail coefficients vector `cD`, obtained by wavelet decomposition of the vector `X`.

`[cA,cD] = dwt(X,Lo_D,Hi_D)` computes the wavelet decomposition as above, given these filters as input:

- `Lo_D` is the decomposition low-pass filter and
- `Hi_D` is the decomposition high-pass filter.

`Lo_D` and `Hi_D` must be the same length.

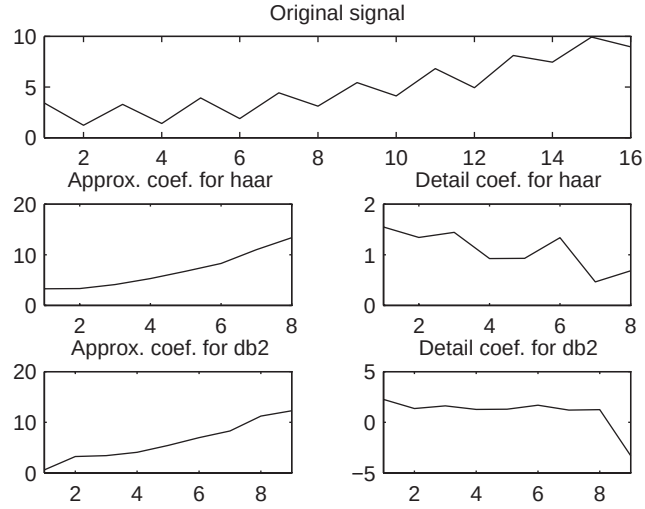
If `lx` is the length of `X` and `lf` is the length of the filters `Lo_D` and `Hi_D`, then `length(cA) = length(cD) = floor((lx+lf-1)/2)`.

For the different signal extension modes, see `dwtmode`.

Examples

```
% Construct elementary original one-dimensional signal.
    randn('seed',531316785)
    s = 2 + kron(ones(1,8),[1 -1]) + ...
        ((1:16).^2)/32 + 0.2*randn(1,16);

% Perform single-level discrete wavelet transform of s by haar.
    [ca1,cd1] = dwt(s,'haar');
    subplot(311); plot(s); title('Original signal');
    subplot(323); plot(ca1); title('Approx. coef. for haar');
    subplot(324); plot(cd1); title('Detail coef. for haar');
```

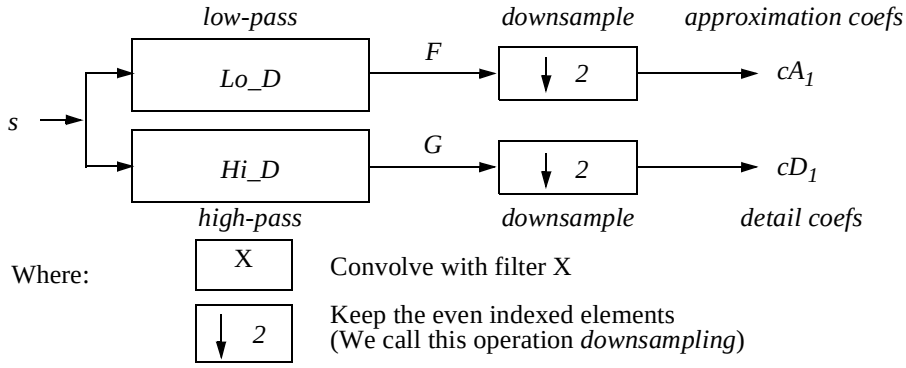
```
% For a given wavelet, compute the two associated decomposition
% filters and compute approximation and detail coefficients
% using directly the filters.
    [Lo_D,Hi_D] = wfilters('haar','d');
    [ca1,cd1] = dwt(s,Lo_D,Hi_D);

% Perform single-level discrete wavelet transform of s by db2
% and observe edge effects for last coefficients.
% These extra coefficients are only used to ensure exact
% global reconstruction.
    [ca2,cd2] = dwt(s,'db2');
    subplot(325); plot(ca2); title('Approx. coef. for db2');
    subplot(326); plot(cd2); title('Detail coef. for db2');
```



Algorithm

Starting from a signal *s*, two sets of coefficients are computed: approximation coefficients CA_1 and detail coefficients CD_1. These vectors are obtained by convolving *s* with the low-pass filter Lo_D for approximation, and with the high-pass filter Hi_D for detail, followed by dyadic decimation.

More precisely, the first step is:



The length of each filter is equal to $2N$. If n = `length`(s), the signals F and G are of length $n + 2N - 1$ and then the coefficients CA_1 and CD_1 are of length $\text{floor}\left(\frac{n-1}{2}\right) + N$.

Note: In order to deal with signal-end effects involved by convolution based algorithm, a global variable managed by `dwtmode` is used. The possible options are: zero-padding (used in the previous example, this mode is the default), symmetric extension, and smooth extension. It should be noted that `dwt` has the same single inverse function `idwt` for the three extension modes.

Limitations

Periodized wavelet transform is handled separately (see `dwtper` and `idwtper`).

See Also

`dwtmode`, `dwtper`, `idwt`, `wavedec`, `waveinfo`

References

I. Daubechies (1992), "Ten lectures on wavelets," CBMS-NSF conference series in applied mathematics. SIAM Ed.

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

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

Purpose

Single-level discrete 2-D wavelet transform.

Syntax

```
[cA,cH,cV,cD] = dwt2(X,'wname')
[cA,cH,cV,cD] = dwt2(X,Lo_D,Hi_D)
```

Description

The `dwt2` command performs a single-level two-dimensional wavelet decomposition with respect to either a particular wavelet (*'wname'*, see `wfilters`) or particular wavelet decomposition filters (`Lo_D` and `Hi_D`) you specify.

`[cA,cH,cV,cD] = dwt2(X,'wname')` computes the approximation coefficients matrix `cA` and the details coefficients matrices `cH`, `cV`, and `cD` (horizontal, vertical, and diagonal), obtained by wavelet decomposition of the input matrix `X`.

`[cA,cH,cV,cD] = dwt2(X,Lo_D,Hi_D)` computes the two-dimensional wavelet decomposition as above, based on wavelet decomposition filters you specify:

- `Lo_D` is the decomposition low-pass filter and
- `Hi_D` is the decomposition high-pass filter.

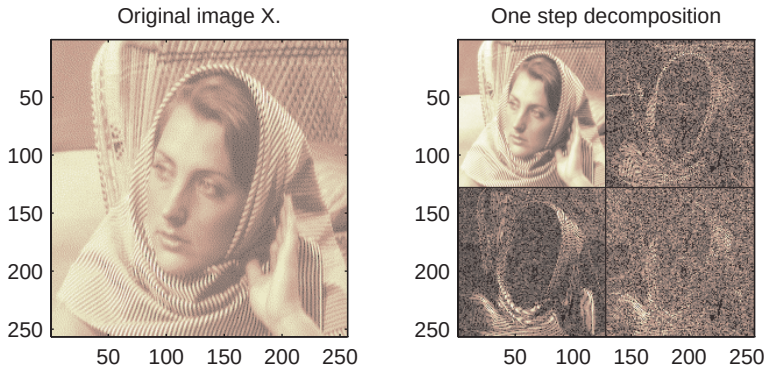
`Lo_D` and `Hi_D` must be the same length. If `sx = size(X)` and `lf` is the length of filters, then `size(cA) = size(cH) = size(cV) = size(cD) = floor((sx+lf-1)/2)`.

For information about the different discrete wavelet transform extension modes, see `dwtmode`.

Examples

```
% Load original image.
    load woman;
    % X contains the loaded image.
    % map contains the loaded colormap.
    nbcol = size(map,1);

% Perform single-level decomposition
% of X using db1.
    [cA1,cH1,cV1,cD1] = dwt2(X,'db1');
```

```
% Images coding.
    cod_X = wcodemat(X,nbcol);
    cod_cA1 = wcodemat(cA1,nbcol);
    cod_cH1 = wcodemat(cH1,nbcol);
    cod_cV1 = wcodemat(cV1,nbcol);
    cod_cD1 = wcodemat(cD1,nbcol);
    dec2d = [...
            cod_cA1,     cod_cH1;     ...
            cod_cV1,     cod_cD1      ...
            ];
```



Algorithm

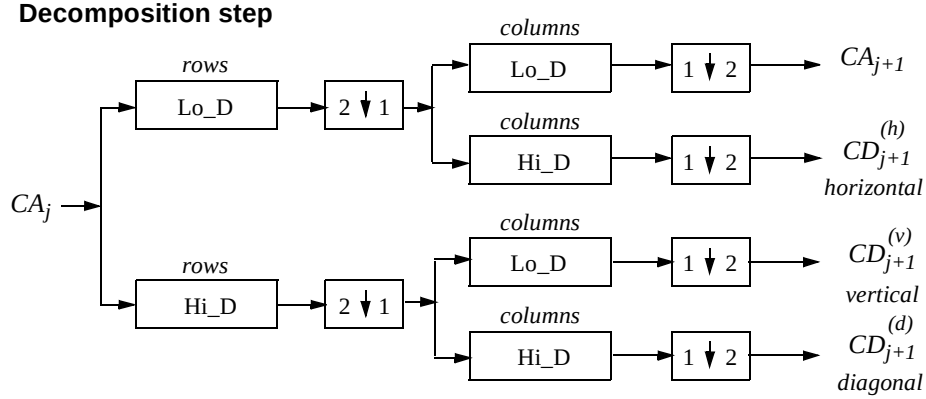
For images, an algorithm similar to the one-dimensional case is possible for two-dimensional wavelets and scaling functions obtained from one-dimensional ones by tensorial product.

This kind of two-dimensional DWT leads to a decomposition of approximation coefficients at level j in four components: the approximation at level $j + 1$ and the details in three orientations (horizontal, vertical, and diagonal).

The following chart describes the basic decomposition steps for images:

Two-Dimensional DWT

Decomposition step



Where:

2 ↓ 1	Downsample columns: keep the even indexed columns
1 ↓ 2	Downsample rows: keep the even indexed rows
rows X	Convolve with filter X the rows of the entry
columns X	Convolve with filter X the columns of the entry

Initialization $CA_0 = s$ for the decomposition initialization

---

Note: In order to deal with signal-end effects involved by convolution based algorithm, a global variable managed by `dwtmode` is used. The possible options are: zero-padding (used in the previous example, this mode is the default), symmetric extension, and smooth extension. It should be noted that `dwt2` has the same single inverse function `idwt2` for the three extension modes.

Limitations

Periodized wavelet transform is handled separately (see `dwtper2` and `idwtper2`).

See Also

`dwtmode`, `dwtper2`, `idwt2`, `wavedec2`, `waveinfo`

References

I. Daubechies (1992), “Ten lectures on wavelets,” CBMS-NSF conference series in applied mathematics. SIAM Ed.

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

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

Purpose

Discrete wavelet transform extension mode.

Syntax

```
dwtmode
dwtmode('mode')  Where 'mode' can be 'zpd', 'sym', or 'spd'
```

Description

The `dwtmode` command sets the signal or image extension mode for discrete wavelet and wavelet packet transforms. The extension modes represent different ways of handling the problem of border distortion in signal and image analysis. For more information, see "Dealing with Border Distortion" in Chapter 6.

`dwtmode` or `dwtmode('status')` displays the current mode.

`dwtmode('mode')` sets the DWT extension mode according to the value of *'mode'*:

'mode'	**DWT Extension Mode**
'zpd'	Zero-padding (default)
'sym'	Symmetrization (boundary value replication)
'spd'	Smooth padding (first derivative interpolation at the edges)

If `dwtmode` is called with two input arguments, the second one is dummy and no text (status or warning) is displayed in the MATLAB command window.

The `dwtmode` function updates a global variable allowing three ways of signal extension. Only `dwt` and `dwt2` use the global variable.

Examples

```
% If the DWT extension mode global variable does not
% exist, default is zero-padding.
clear global
dwtmode

************************************
** DWT Extension Mode: Zero-padding **
************************************
```

```
% Display current DWT signal extension mode.
dwtmode

************************************
** DWT Extension Mode: Zero-Padding **
************************************

% Change to symmetrization extension mode.
dwtmode('sym')

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! WARNING: Change DWT Extension Mode !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

**************************************
** DWT Extension Mode: Symmetrization **
**************************************

% Display current DWT signal extension mode.
dwtmode

**************************************
** DWT Extension Mode: Symmetrization **
**************************************
```

Note: You should change the extension mode only by using `dwtmode`; avoid changing the global variable directly.

Limitations

Periodized wavelet transform is handled separately (see `dwtper`, `dwtper2`, `idwtper`, `idwtper2`).

See Also

`dwt`, `dwt2`

Purpose

Single-level discrete 1-D wavelet transform (periodized).

Syntax

```
[cA,cD] = dwtper(X,'wname')
[cA,cD] = dwtper(X,Lo_D,Hi_D)
```

Description

`dwtper` is a one-dimensional wavelet analysis function.

`[cA,cD] = dwtper(X,'wname')` computes the approximation coefficients vector `cA` and detail coefficients vector `cD`, obtained by periodized wavelet decomposition of the vector `X`.

'*wname*' is a string containing the wavelet name (see `wfilters`).

Instead of giving the wavelet name, you can give the filters. When used with three arguments: `[cA,cD] = dwtper(X,Lo_D,Hi_D)`, `Lo_D` is the decomposition low-pass filter and `Hi_D` is the decomposition high-pass filter.

If `lx = length(X)` then `length(cA) = length(cD) = ceil(lx/2)`.

Examples

```
% Set initial signal and get filters.
x = sin(0.3*[1:300]); lx = length(x)
lx =
    300

w = 'db9';
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters(w);

% DWT with zero-padding signal extension.
[cazp,cdzp] = dwt(x,w);

% The transform uses some extra coefficients,
% at most 2 if lx is odd.
lxtzp = 2*length(cazp)
lxtzp =
    316
```

```
% Reconstruction.
xzp = idwt(cazp,cdzp,w,lx);

% Error with zero-padding.
errzp = max(abs(x-xzp))
errzp =
    7.3231e-12

% Periodized DWT.
[cap,cdp] = dwtper(x,w);

% The transform uses a minimum of extra coefficients.
lxtp = 2*length(cap)
lxtp =
    300

% Reconstruction.
xp = idwtper(cap,cdp,w,lx);

% Error with periodized DWT.
errp = max(abs(x-xp))
errp =
 1.4588e-11
```

Algorithm

The algorithm is the same as in `dwt` but the signal `X` is extended assuming periodicity. More precisely, if `lx = length(X)` is even, the extended signal is `extX = [X(lx-lf+1:lx) X X(1:lf)]` where `lf` is the length of the filter. Then, usual convolution and downsampling operations are done, followed by keeping the central part of length `lx/2`.

See Also

`dwt`, `idwtper`

Purpose

Single-level discrete 2-D wavelet transform (periodized).

Syntax

```
[cA,cH,cV,cD] = dwtper2(X,'wname')
[cA,cH,cV,cD] = dwtper2(X,Lo_D,Hi_D)
```

Description

`dwtper2` is a two-dimensional wavelet analysis function.

`[cA,cH,cV,cD] = dwtper2(X,'wname')` computes the approximation coefficients matrix `cA` and details coefficients matrices `cH`, `cV`, and `cD`, obtained by periodized wavelet decomposition of the input matrix `X`.

'*wname*' is a string containing the wavelet name (see `wfilters`).

Instead of giving the wavelet name, you can give the filters. When used with three arguments: `[cA,cH,cV,cD] = dwtper2(X,Lo_D,Hi_D)`, `Lo_D` is the decomposition low-pass filter and `Hi_D` is the decomposition high-pass filter.

If `sx = size(X)` then `size(cA) = size(cH) = size(cV) = size(cD) = ceil(sx/2)`.

Examples

```
% Set initial signal and get filters.
    load tire
    % X contains the loaded image.
    sx = size(X)

sx =
    205 232

w = 'db9';
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters(w);

% DWT with zero-padding image extension.
[ca0,ch0,cv0,cd0] = dwt2(X,w);

% The transform uses some extra coefficients.
sxtzp = 2*size(ca0)
sxtzp =
    222 248
```

```
% Reconstruction
x0 = idwt2(ca0,ch0,cv0,cd0,w,sx);

% Error with zero-padding.
err0 = max(max(abs(X-x0)))
err0 =
    6.3292e-09

% Periodized DWT
[cap,chp,cvp,cdp] = dwtper2(X,w);

% The transform uses a minimum of extra coefficients.
lxtp = 2*size(cap)
lxtp =
    206 232

% Reconstruction.
xp = idwtper2(cap,chp,cvp,cdp,w,sx);

% Error with periodized DWT.
errp = max(max(abs(X-xp)))
errp =
    6.7353e-09
```

Algorithm See the dwtper algorithm section.

See Also dwt2, idwtper2

Purpose

Dyadic downsampling.

Syntax

```
Y = dyaddown(X,evenodd)
Y = dyaddown(X)
Y = dyaddown(X,evenodd,'type')
Y = dyaddown(X,'type',evenodd)
```

Description

`Y = dyaddown(X,evenodd)`, where `X` is a *vector*, returns a version of `X` that has been downsampled by 2. Whether `Y` contains the even- or odd-indexed samples of `X` depends on the value of positive integer `evenodd`:

- If `evenodd` is even, then `Y(k) = X(2k)`.
- If `evenodd` is odd, then `Y(k) = X(2k+1)`.

If you omit the `evenodd` argument, `dyaddown(X)` defaults to `evenodd = 0` (even-indexed samples).

`Y = dyaddown(X,evenodd,'type')` or `Y = dyaddown(X,'type',evenodd)`, where `X` is a *matrix*, return a version of `X` obtained by suppressing:

Columns of `X`	If `'type' = 'c'`
Rows of `X`	If `'type' = 'r'`
Rows and columns of `X`)	If `'type' = 'm'`

If you omit the `evenodd` or '*type*' arguments, `dyaddown` defaults to `evenodd = 0` (even-indexed samples) and `'type' = 'c'` (columns).

Examples

```
% For a vector.
s = 1:10
s =
   1   2   3   4   5   6   7   8   9  10

dse = dyaddown(s)   % Downsample elements with even indices.
dse =
   2   4   6   8  10
```

```
% or equivalently
dse = dyaddown(s,0)
dse =
   2   4   6   8  10

dso = dyaddown(s,1) % Downsample elements with odd indices.
dso =
   1   3   5   7   9

% For a matrix.
s = (1:3)'*[1:4]
s =
   1   2   3   4
   2   4   6   8
   3   6   9  12

dec = dyaddown(s,0,'c') % Downsample columns with even indices.
dec =
   2   4
   4   8
   6  12

der = dyaddown(s,1,'r') % Downsample rows with odd indices.
der =
 1   2   3   4
 3   6   9  12
```

See Also

dyadup

References

G. Strang, T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press.

Purpose

Dyadic upsampling.

Syntax

```
Y = dyadup(X,evenodd)
Y = dyadup(X)
Y = dyadup(X,evenodd,'type')
Y = dyadup(X,'type',evenodd)
```

Description

dyadup implements a simple zero-padding scheme very useful in the wavelet reconstruction algorithm.

Y = dyadup(X,evenodd), where X is a *vector*, returns an extended copy of vector X obtained by inserting zeros. Whether the zeros are inserted as even- or odd-indexed elements of Y depends on the value of positive integer evenodd:

- If evenodd is even, then Y(2k–1) = X(k), Y(2k) = 0.
- If evenodd is odd, then Y(2k–1) = 0, Y(2k) = X(k).

If you omit the evenodd argument, dyadup(X) defaults to evenodd = 1 (zeros in odd-indexed positions).

Y = dyadup(X,evenodd,'*type*') or Y = dyadup(X,'*type*',evenodd), where X is a *matrix*, return extended copies of X obtained by inserting:

Columns in X	If '*type*' = 'c'
Rows in X	If '*type*' = 'r'
Rows and columns in X	If '*type*' = 'm'

If you omit the evenodd or '*type*' arguments, dyadup defaults to evenodd = 1 (zeros in odd-indexed positions) and '*type*' = 'c' (insert columns).

Examples

```
% For a vector.
    s = 1:5

s =
    1 2 3 4 5

    dse = dyadup(s) % Upsample elements at odd indices.
dse =
    0 1 0 2 0 3 0 4 0 5 0
    % or equivalently
    dse = dyadup(s,1)

dse =
    0 1 0 2 0 3 0 4 0 5 0
    dso = dyadup(s,0) % Upsample elements at even indices.

dso =
    1 0 2 0 3 0 4 0 5

% For a matrix.
    s = (1:2)'*[1:3]

s =
    1 2 3
    2 4 6
    der = dyadup(s,1,'r') % Upsample rows at even indices.

der =
    0 0 0
    1 2 3
    0 0 0
    2 4 6
    0 0 0
    doc = dyadup(s,0,'c') % Upsample columns at odd indices.

doc =
    1 0 2 0 3
    2 0 4 0 6
```

```
% Using default values for dyadup and dyaddown, we have:
% dyaddown(dyadup(s)) = s.
s = 1:5

s =
    1 2 3 4 5
    uds = dyaddown(dyadup(s))

uds =
    1 2 3 4 5
    % In general reversed identity is false.
```

See Also

dyaddown

References

G. Strang, T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press

entrupd

Purpose

Entropy update (wavelet packet).

Syntax

```
NDATA = entrupd(TREE,DATA,ENT)
NDATA = entrupd(TREE,DATA,ENT,PAR)
```

Description

`entrupd` is a one- or two-dimensional wavelet packets utility.

`NDATA = entrupd(TREE,DATA,ENT)` or
`NDATA = entrupd(TREE,DATA,ENT,PAR)` returns for a given wavelet packet decomposition structure `[TREE,DATA]` (see `maketree`), the updated data structure `NDATA` corresponding to entropy function `ENT` with optional parameter `PAR` (see `wentropy`).

`[TREE,NDATA]` is the resulting decomposition structure.

Examples

```
% Load signal.
    load noisdopp; x = noisdopp;

% Decompose x at depth 2 with db1 wavelet packets
% using shannon entropy.
    [t,d] = wpdec(x,2,'db1','shannon');

% Read entropy of all the nodes.
    nodes = allnodes(t);
    ent = wdatamgr('read_ent',d,nodes)

ent =
    1.0e+04 *
    -5.8615 -6.8204 -0.0350 -7.7901 -0.0497 -0.0205 -0.0138

% Update nodes entropy without changing tree
% and data structures.
    d = entrupd(t,d,'threshold',0.5);
    nent = wdatamgr('read_ent',d,nodes)


nent =
    937 488 320 241 175 170 163
```

See Also

`wentropy`, `wpdec`, `wpdec2`

Purpose

Check function arguments number.

Syntax

```
err = errargn('function',numargin,argin,numargout,argout)
```

Description

`errargn` is a general utility.

`err = errargn('function',numargin,argin,numargout,argout)` is equal to 1 if either the number of input (`argin`) or output (`argout`) arguments of the specified *function* does not belong to the vector of allowed values (`numargin` and
`numargout`, respectively). Otherwise `err` = 0.

If `err` = 1, `errargn` displays an error message in the command window. The header of this error message contains the string '*function*'.

Examples

In this example, `errargn` reports an improper call to function `line`:

```
» err = errargn('line',4,[2 3],0,[0 1]);

**************************************
ERROR ...
--------------------------------------
 line ---> invalid number of arguments
**************************************
```

Here, `surf` is passed the proper number of arguments, so no error message results:

```
» err = errargn('surf',4,[3 4],0,[0 1]);
```

See Also

`errargt`

errargt

Purpose

Check function arguments type.

Syntax

```
ERR = errargt(NDFCT,VAR,TYPE)
ERR = errargt(NDFCT,VAR,'msg')
```

Description

`errargn` is a general utility.

`ERR = errargt(NDFCT,VAR,`*`TYPE`*`)` is equal to 1 if any element of input vector or matrix `VAR` (depending on *TYPE* choice listed below) is not of type prescribed by input string *TYPE*. Otherwise `ERR` = 0.

If `ERR` = 1, an error message is displayed in the command window. In the header message, the string `NDFCT` is displayed. This string contains the name of a function.

Available options for *TYPE* are:

`'int'` Strictly positive integers (excluding zero)

`'in0'` Positive integers (including zero)

`'rel'` Integers

`'rep'` Strictly positive reals (excluding zero)

`'re0'` Positive reals (including zero)

`'str'` String

`'vec'` Vector

`'row'` Row vector

`'col'` Column vector

`'dat'` Dates *AAAAMMJJHHMNSS* with:

$0 \le AAAA \le 9999$
$1 \le MM \le 12$
$1 \le JJ \le 31$
$0 \le HH \le 23$
$0 \le MN \le 59$
$0 \le SS \le 59$

`'mon'` Months *MM* with:

$1 \le MM \le 12$

A special use of `errargt` is:

`ERR = errargt(NDFCT,VAR,'msg')` for which `ERR` = 1 and the string `VAR` is the error message.

See Also

`errargn`

idwt

Purpose

Single-level inverse discrete 1-D wavelet transform.

Syntax

```
X = idwt(cA,cD,'wname')
X = idwt(cA,cD,Lo_R,Hi_R)
X = idwt(cA,cD,'wname',L)
X = idwt(cA,cD,Lo_R,Hi_R,L)
```

Description

The `idwt` command performs a single-level one-dimensional wavelet reconstruction with respect to either a particular wavelet (*'wname'*, see `wfilters`) or particular wavelet reconstruction filters (`Lo_R` and `Hi_R`) you specify.

`X = idwt(cA,cD,'wname')` returns the single-level reconstructed approximation coefficients vector `X` based on approximation and detail coefficients vectors `cA` and `cD`, and using the wavelet *'wname'*.

`X = idwt(cA,cD,Lo_R,Hi_R)` reconstructs as above using filters you specify:

- `Lo_R` is the reconstruction low-pass filter
- `Hi_R` is the reconstruction high-pass filter

`Lo_R` and `Hi_R` must be the same length. If `la` is the length of `cA` (which also equals the length of `cD`) and `lf` is the length of the filters `Lo_R` and `Hi_R`, then `length(X) = 2*la-lf+2`.

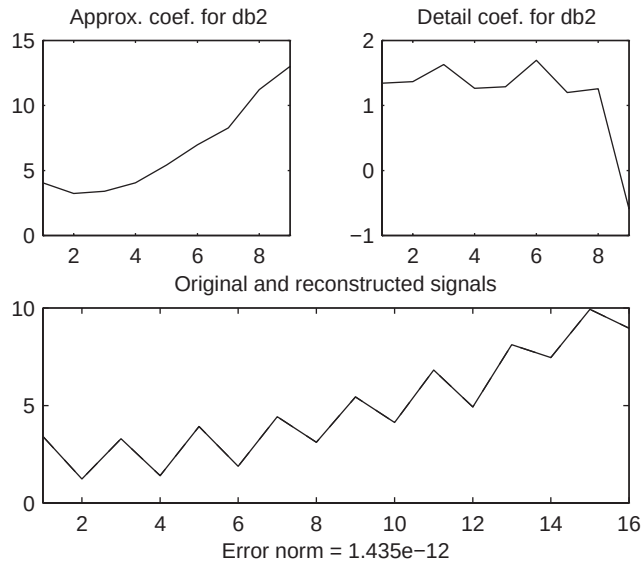
`X = idwt(cA,cD,'wname',L)` or `X = idwt(cA,cD,Lo_R,Hi_R,L)`, returns the length-`L` central portion of the result obtained using `idwt(cA,cD,'wname')`. `L` must be less than `2*la-lf+2`.

Examples

`idwt` is the inverse function of `dwt` in the sense that the abstract statement `idwt(dwt(X,'wname'),'wname')` gives back `X`. Consider this example.

```
% Construct elementary one-dimensional signal s.
    randn('seed',531316785)
    s = 2 + kron(ones(1,8),[1 -1]) + ...
        ((1:16).^2)/32 + 0.2*randn(1,16);
```

```
% Perform single-level dwt of s using db2.
    [ca1,cd1] = dwt(s,'db2');
    subplot(221); plot(ca1);
    title('Approx. coef. for db2');
    subplot(222); plot(cd1);
    title('Detail coef. for db2');

% Perform single-level inverse discrete wavelet transform,
% illustrating that idwt is the inverse function of dwt.
    ss = idwt(ca1,cd1,'db2');
    err = norm(s-ss);     % Check reconstruction.
    subplot(212); plot([s;ss]');
    title('Original and reconstructed signals');
    xlabel(['Error norm = ',num2str(err)])

% For a given wavelet, compute the two associated
% reconstruction filters and inverse transform using
% the filters directly.
    [Lo_R,Hi_R] = wfilters('db2','r');
    ss = idwt(ca1,cd1,Lo_R,Hi_R);
```

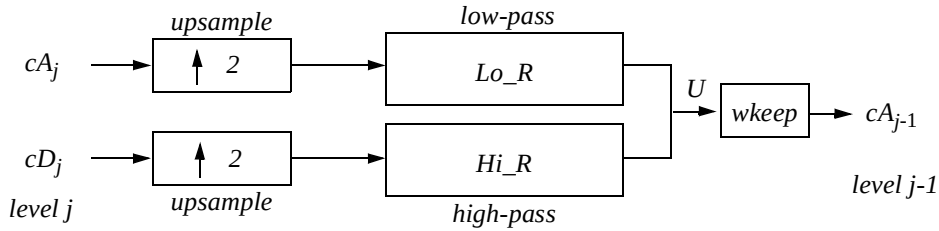


Algorithm

Starting from the approximation and detail coefficients at level j, cA_j and cD_j, the inverse discrete wavelet transform reconstructs cA_{j-1}, inverting the decomposition step by inserting zeros and convolving the results with the reconstruction filters.

One-Dimensional IDWT

Reconstruction step



Where:

↑ 2	Insert zeros at odd-indexed elements
X	Convolve with filter X
wkeep	Take the central part of `U` with the convenient length

See Also

`dwt`, `idwtper`, `upwlev`

References

I. Daubechies (1992), "Ten lectures on wavelets," CBMS-NSF conference series in applied mathematics. SIAM Ed.

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

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

Purpose

Single-level inverse discrete 2-D wavelet transform.

Syntax

```
X = idwt2(cA,cH,cV,cD,'wname')
X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R)
X = idwt2(cA,cH,cV,cD,'wname',S)
X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R,S)
```

Description

The `idwt2` command performs a single-level two-dimensional wavelet reconstruction with respect to either a particular wavelet (*'wname'*, see `wfilters`) or particular wavelet reconstruction filters (`Lo_R` and `Hi_R`) you specify.

`X = idwt2(cA,cH,cV,cD,'wname')` uses the wavelet *'wname'* to compute the single-level reconstructed approximation coefficients vector `X` based on approximation vector `cA` and (horizontal, vertical, and diagonal) detail vectors `cH`, `cV` and `cD`.

`X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R)` reconstructs as above, using filters you specify:

- `Lo_R` is the reconstruction low-pass filter
- `Hi_R` is the reconstruction high-pass filter

`Lo_R` and `Hi_R` must be the same length.

If `sa = size(cA) = size(cH) = size(cV) = size(cD)` and `lf` is the length of the filters, then `size(X) = 2*size(cA)-lf+2`.

`X = idwt2(cA,cH,cV,cD,'wname',S)` and `X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R,S)` return the size `S` central portion of the result obtained using the syntax `idwt2(cA,cH,cV,cD,'wname')`. `S` must be less than `2*size(cA)-lf+2`.

Examples

`idwt2` is the inverse function of `dwt2` in the sense that the abstract statement `idwt2(dwt2(X,'wname'),'wname')` gives back `X`. Consider this example.

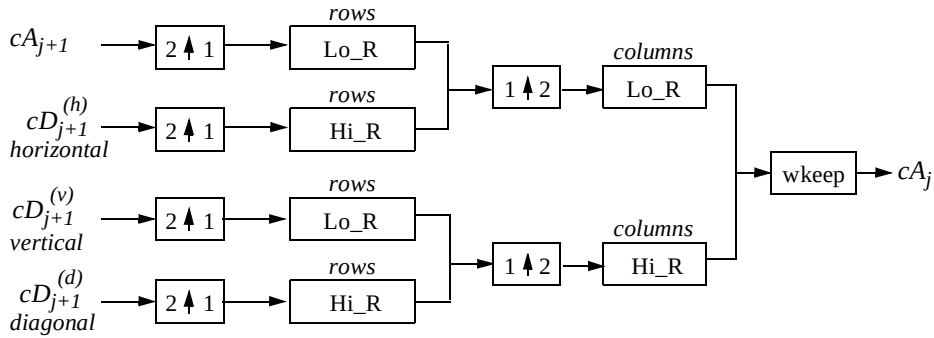
```
% Load original image.
    load woman;
    % X contains the loaded image.
    sX = size(X);
```

```
% Perform single-level decomposition
% of X using db4.
    [cA1,cH1,cV1,cD1] = dwt2(X,'db4');

% Invert directly decomposition of X
% using coefficients at level 1.
    A0 = idwt2(cA1,cH1,cV1,cD1,'db4',sX);

% Check for perfect reconstruction.
    max(max(X-A0))
ans =
    3.3032e-10
```

Algorithm

Two-Dimensional IDWT

Reconstruction step



Where:

2 ↑ 1	Upsample columns: insert zeros at odd-indexed columns
1 ↑ 2	Upsample rows: insert zeros at odd-indexed rows
rows X	Convolve with filter X the rows of the entry
columns X	Convolve with filter X the columns of the entry

See Also

`dwt2`, `idwtper2`, `upwlev2`

Purpose

Single-level inverse discrete 1-D wavelet transform (periodized).

Syntax

```
X = idwtper(cA,cD,'wname')
X = idwtper(cA,cD,Lo_R,Hi_R)
X = idwtper(cA,cD,'wname',L)
X = idwtper(cA,cD,Lo_R,Hi_R,L)
```

Description

`idwtper` is a one-dimensional wavelet analysis function.

`X = idwtper(cA,cD,'wname')` returns the single-level reconstructed approximation coefficients vector `X` based on approximation and detail vectors `cA` and `cD` at a given level, using the periodized inverse wavelet transform. '*wname*' is a string containing the wavelet name (see `wfilters`).

Instead of giving the wavelet name, you can give the filters.

For `X = idwtper(cA,cD,Lo_R,Hi_R)`:

`Lo_R` is the reconstruction low-pass filter.

`Hi_R` is the reconstruction high-pass filter.

If `la = length(cA) = length(cD)` then `length(X) = 2*la`.

For `X = idwtper(cA,cD,'wname',L)` or `X = idwtper(cA,cD,Lo_R,Hi_R,L)`, `L` is the length of the result.

`idwtper` is the inverse function of `dwtper` in the sense that the abstract statement `idwtper(dwtper(X,'wname'),'wname')` gets back to `X`.

Examples

```
% Set initial signal and get filters.
x = sin(0.3*[1:300]); lx = length(x)
lx =
    300

w = 'db9';
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters(w);

% DWT with zero-padding signal extension.
[cazp,cdzp] = dwt(x,w);
```

```
% The transform uses some extra coefficients,
% at most 2 if lx is odd.
lxtzp = 2*length(cazp)
lxtzp =
    316

% Reconstruction.
xzp = idwt(cazp,cdzp,w,lx);

% Error with zero-padding.
errzp = max(abs(x-xzp))
errzp =
    7.3231e-12

% Periodized DWT.
[cap,cdp] = dwtper(x,w);

% The transform uses a minimum of extra coefficients.
lxtp = 2*length(cap)
lxtp =
    300

% Reconstruction.
xp = idwtper(cap,cdp,w,lx);

% Error with periodized DWT.
errp = max(abs(x-xp))
errp =
    1.4588e-11
```

Note: In general, the following abstract statement is not true: `idwtper(dwt(X,'wname'),'wname') = X`.

See Also

`dwtper`

Purpose

Single-level inverse discrete 2-D wavelet transform (periodized).

Syntax

```
X = idwtper2(cA,cH,cV,cD,'wname')
X = idwtper2(cA,cH,cV,cD,Lo_R,Hi_R)
X = idwtper2(cA,cH,cV,cD,'wname',S)
X = idwtper2(cA,cH,cV,cD,Lo_R,Hi_R,S)
```

Description

`idwtper2` is a two-dimensional wavelet analysis function.

`X = idwtper2(cA,cH,cV,cD,'wname')` returns the single-level reconstructed approximation coefficients vector `X` based on approximation and details vectors `cA`, `cH`, `cV`, and `cD` at a given level, using the periodized inverse wavelet transform. *'wname'* is a string containing the wavelet name (see `wfilters`).

Instead of giving the wavelet name, you can give the filters.

For `X = idwtper2(cA,cH,cV,cD,Lo_R,Hi_R)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

If `sa = size(cA) = size(cH) = size(cV) = size(cD)`, then `size(X) = 2*sa`.

For `X = idwtper2(cA,cH,cV,cD,'wname',S)` or
`X = idwtper2(cA,cH,cV,cD,Lo_R,Hi_R,S)`, `S` is the size of the result.

`idwtper2` is the inverse function of `dwtper2` in the sense the abstract statement `idwtper2(dwtper2(X,'wname'),'wname')` gets back to `X`.

Examples

```
% Set initial signal and get filters.
load tire
    % X contains the loaded image.
    sx = size(X)
sx =
    205    232

w = 'db9';
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters(w);

% DWT with zero-padding image extension.
[ca0,ch0,cv0,cd0] = dwt2(X,w);
```

```
% The transform uses some extra coefficients.
sxtzp = 2*size(ca0)
sxtzp =
    222    248

% Reconstruction.
x0 = idwt2(ca0,ch0,cv0,cd0,w,sx);

% Error with zero-padding.
err0 = max(max(abs(X-x0)))
err0 =
    6.3292e-09

% Periodized DWT.
[cap,chp,cvp,cdp] = dwtper2(X,w);

% The transform uses a minimum of extra coefficients.
lxtp = 2*size(cap)
lxtp =
    206    232

% Reconstruction.
xp = idwtper2(cap,chp,cvp,cdp,w,sx);

% Error with periodized DWT.
errp = max(max(abs(X-xp)))
errp =
    6.7353e-09
```

Note: In general, the following abstract statement is not true: `idwtper2(dwt2(X,'wname'),'wname') = X`.

See Also

`dwtper2`

Purpose

Node index to node depth-position.

Syntax

```
[D,P] = ind2depo(ORD,N)
```

Description

`ind2depo` is a tree management utility.

For a tree of order `ORD`, `[D,P] = ind2depo(ORD,N)` computes the depths `D` and the positions `P` (at this depths `D`) for the nodes with indices `N`.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

`D`, `P`, `N` are column vectors. The values of `D`, `P`, `N` are constrained by:

`D` = depths, $0 \le D \le dmax$

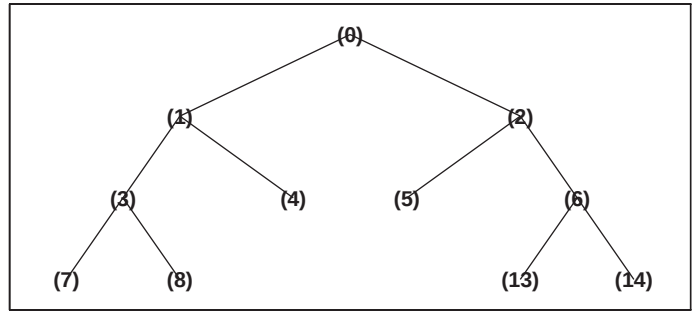
`P` = positions at depth D, $0 \le P \le order^{D-1}$

`N` = indices, $0 \le N < (order^{(dmax+1)}-1)/(order-1)$

Note that `[D,P] = ind2depo(ORD,[D P])`.

Examples

```
% Create initial tree.
    ord = 2; t = maketree(ord,3);    % Binary tree of depth 3.
    tt = nodejoin(t,5);
    tt = nodejoin(tt,4);
    plottree(tt)
```



```
% List tt nodes (index).
    aln_ind = allnodes(tt)
aln_ind =
    0
    1
    2
    3
    4
    5
    6
    7
    8
   13
   14

% Switch from index to depth-position.
[depth,pos] = ind2depo(ord,aln_ind);
aln_depo = [depth,pos]
aln_depo =
    0     0
    1     0
    1     1
    2     0
    2     1
    2     2
    2     3
    3     0
    3     1
    3     6
    3     7
```

See Also depo2ind, maketree, wtreemgr

Purpose

Inverse nonstandard 1-D fast Fourier transform.

Syntax

```
[X,T] = instdfft(XHAT,LOWB,UPPB)
```

Description

`instdfft` is a general mathematical utility.

`[X,T] = instdfft(XHAT,LOWB,UPPB)` returns the inverse nonstandard FFT of `XHAT`, on a power of 2 regular grid (not necessarily integers) on the interval `[LOWB,UPPB]`.

Output arguments are `X` the recovered signal computed on the time interval `T` given by `T = LOWB + [0:n-1]*(UPPB-LOWB)/n`, where `n` is the length of `XHAT`.

Outputs are vectors of length `n`.

Algorithm

See `nstdfft` algorithm section.

Limitations

The length of `XHAT` must be a power of two.

See Also

`fft`, `ifft`, `nstdfft`

intwave

Purpose

Integrate wavelet function psi.

Syntax

```
[INTEG,XVAL] = intwave('wname',PREC)
[INTEG,XVAL] = intwave('wname',PREC,PFLAG)
[INTEG,XVAL] = intwave('wname')
```

Description

`[INTEG,XVAL] = intwave('wname',PREC)` returns values of the wavelet function ψ integrals `INTEG` (from $-\infty$ to `XVAL` values): $\int_{-\infty}^{x} \psi(y)dy$ for x in `XVAL`.

The function ψ is approximated on the 2`PREC` points grid `XVAL`, where `PREC` is a positive integer. '*wname*' is a string containing the name of the wavelet ψ (see `wfilters`).

When used with three arguments, the third one is a dummy argument.

`[INTEG,XVAL] = intwave('wname',PREC,PFLAG)` in addition plots `INTEG` on `XVAL` grid if `PFLAG` is nonzero.

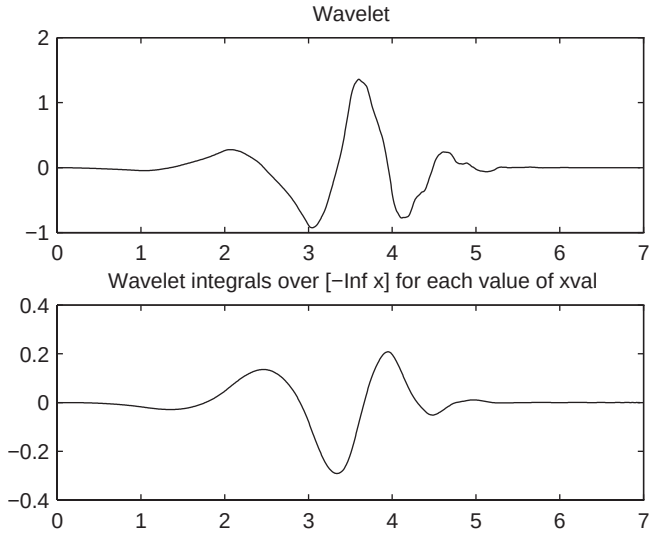
`[INTEG,XVAL] = intwave('wname',PREC)` is equivalent to
`[INTEG,XVAL] = intwave('wname',PREC,0)`.

`[INTEG,XVAL] = intwave('wname')` is equivalent to
`[INTEG,XVAL] = intwave('wname',8)`.

`intwave` is used only for continuous analysis (see `cwt`).

Examples

```
% Set wavelet name.
    wname = 'db4';

% Plot wavelet function.
    [phi,psi,xval] = wavefun(wname,7);
    subplot(211); plot(xval,psi); title('Wavelet');

% Compute and plot wavelet integrals approximations
% on a dyadic grid.
    [integ,xval] = intwave(wname,7);
    subplot(212); plot(xval,integ);
    title(['Wavelet integrals over [-Inf x] ' ...
        'for each value of xval']);
```



Algorithm

First, the wavelet function is approximated on a grid of 2^{PREC} points using `wavefun`. A piecewise constant interpolation is used in order to compute the integrals using `cumsum`.

See Also

`wavefun`

isnode

Purpose

True for existing node.

Syntax

```
R = isnode(T,N)
```

Description

`isnode` is a tree management utility.

`R = isnode(T,N)` returns 1's for nodes `N`, which exist in the tree structure `T`, and 0's for others. `N` can be a column vector containing the indices of nodes or a matrix, that contains the depths and positions of nodes.

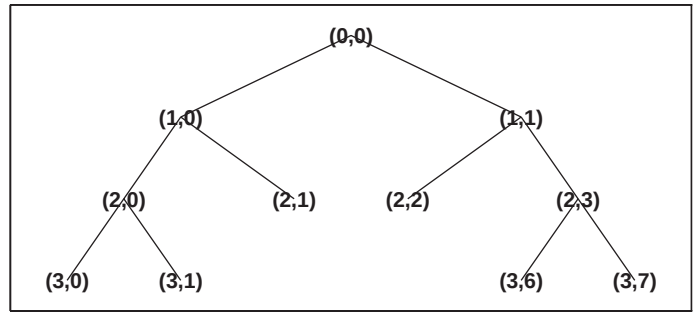
In the last case, `N(i,1)` is the depth of `i`-th node and `N(i,2)` is the position of `i`-th node.

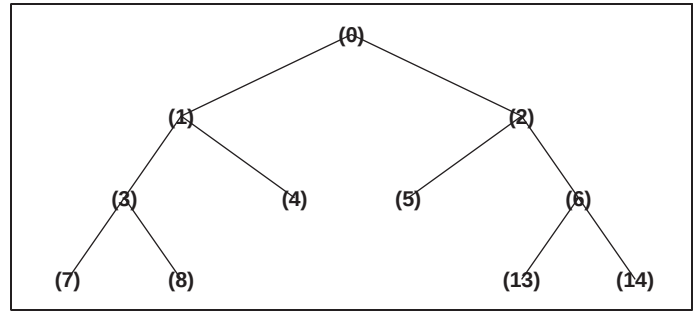
The nodes are numbered from left to right and from top to bottom. The root index is 0.

Examples

```
% Create initial tree.
    ord = 2;
    t = maketree(ord,3);    % binary tree of depth 3.
    tt = nodejoin(t,5);
```

```
    tt = nodejoin(tt,4);
    plottree(tt)
```






```
% Check node index.
    isnode(tt,[1;3;25])
ans =
    1
    1
    0

% Check node depth-position.
    isnode(tt,[1 0;3 1;4 5])
ans =
    1
    1
    0
```

See Also istnode, maketree, wtreemgr

istnode

Purpose

Check if nodes are terminal nodes.

Syntax

```
R = istnode(T,N)
```

Description

`istnode` is a tree management utility.

`R = istnode(T,N)` returns ranks (in left to right terminal nodes ordering) for terminal nodes `N` belonging to the tree structure `T` and 0's for others.

`N` can be a column vector containing the indices of nodes or a matrix that contains the depths and positions of nodes.

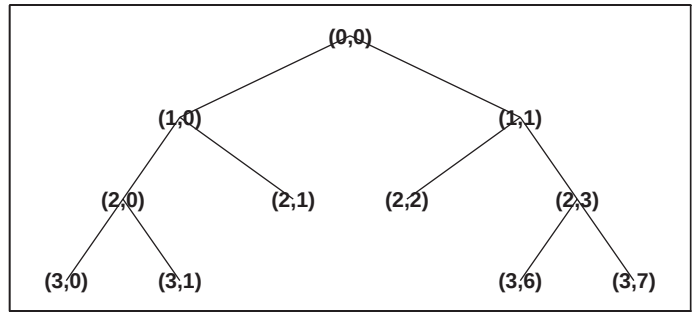
In the last case, `N(i,1)` is the depth of `i`-th node and `N(i,2)` is the position of `i`-th node.

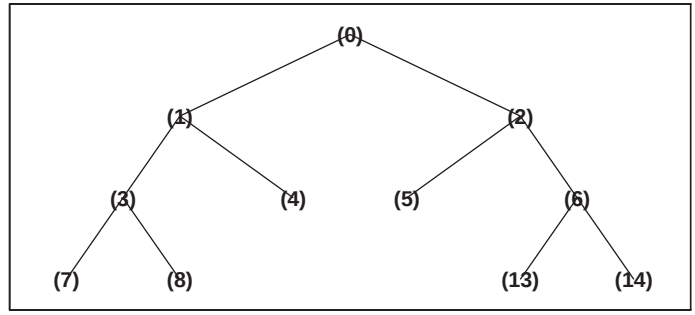
The nodes are numbered from left to right and from top to bottom. The root index is 0.

Examples

% Create initial tree.

```
ord = 2;
t = maketree(ord,3); % binary tree of depth 3.
tt = nodejoin(t,5);
tt = nodejoin(tt,4);
plottree(tt)
```






```
% Find terminal nodes and return indices for terminal
% nodes in the tree structure.
    istnode(tt,[14])
ans =
    6

    istnode(tt,[15])
ans =
    0

    istnode(tt,[1;7;14;25])
ans =
    0
    1
    6
    0
```

```
    istnode(tt,[1 0;3 1;4 5])
ans =
    0
    2
    0
```

See Also isnode, maketree, wtreemgr

Purpose

Make tree.

Syntax

```
[T,NB] = maketree(ORD,D)
[T,NB] = maketree(ORD,D,NBI)
[T,NB] = maketree(ORD)
```

Description

`maketree` is a tree management utility.

`maketree` creates a tree of order `ORD` and depth `D`. `ORD` is an integer equal to the number of children of a generic node. Each nonterminal node has `ORD` children.

For wavelet packet decomposition, a convenient structure is a binary tree for the one-dimensional case (`ORD = 2`) and a quaternary tree for the two-dimensional case (`ORD = 4`). The depth `D` is the number of levels of the tree.

`[T,NB] = maketree(ORD,D)` creates a tree structure of order `ORD` with depth `D`. Output argument `NB` is the number of terminal nodes (`NB = ORD^D`). Output vector `T` is organized as:

`[T(1) ... T(NB+1)]` where `T(i), i = 1, ..., NB` are the indices of the terminal nodes and `T(NB+1) = -ORD`.

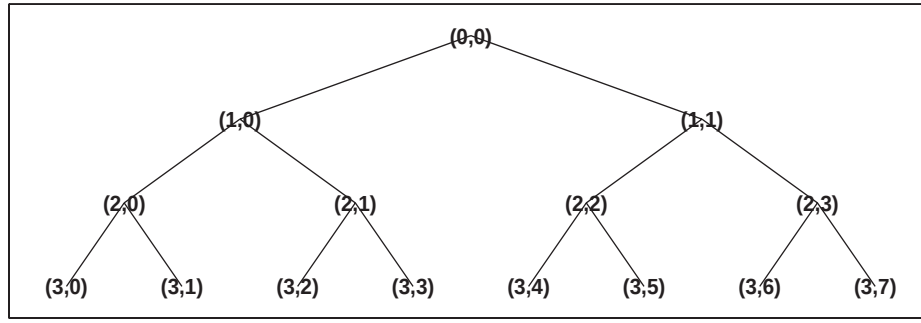
The nodes are numbered from left to right and from top to bottom. The root index is 0.

When used with three input arguments, `[T,NB] = maketree(ORD,D,NBI)` computes `T` as a `(1+NBI)`-by-`(NB+1)` matrix with `T(1,:)` as above and in the range `T(2:NBI+1,:)` the user is free to add their own material.

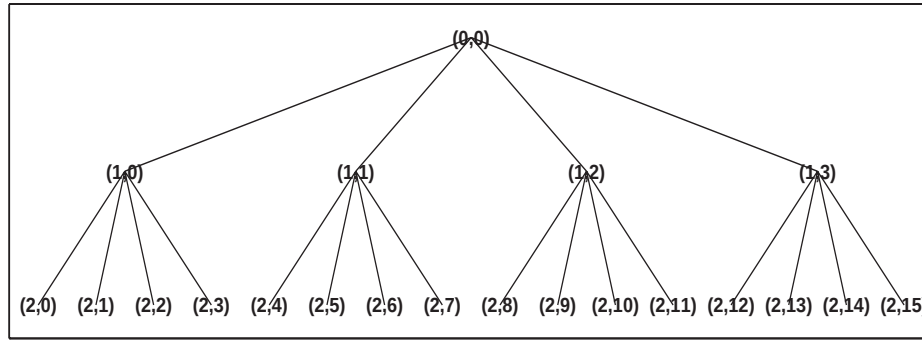
`[T,NB] = maketree(ORD)` is equivalent to `[T,NB] = maketree(ORD,0,0)`.
`[T,NB] = maketree(ORD,D)` is equivalent to `[T,NB] = maketree(ORD,D,0)`.

Examples

```
% Create binary tree (tree of order 2) of depth 3.
    t2 = maketree(2,3);

% Plot tree structure t2.
    plottree(t2)
```




```
% Create binary tree (tree of order 4) of depth 2.
    t4 = maketree(4,2);

% Plot tree structure t4.
    plottree(t4)
```



See Also

plottree, wtreemgr

Purpose

Mexican hat wavelet.

Syntax

```
[PSI,X] = mexihat(LB,UB,N)
```

Description

`[PSI,X] = mexihat(LB,UB,N)` returns values of the Mexican hat wavelet on an `N` point regular grid, `X`, on the interval `[LB,UB]`.

Output arguments are the wavelet function `psi` computed on the grid `X`, and the grid `X`.

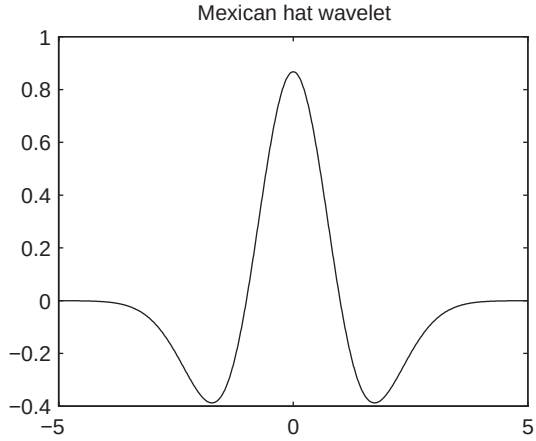
This wavelet has [-5 5] as effective support.

$$\psi(x) = \left(\frac{2}{\sqrt{3}}\pi^{-1/4}\right)(1-x^2)e^{-x^2/2}$$

This function is proportional to the second derivative function of the Gaussian probability density function.

Examples

```
% Set effective support and grid parameters.
    lb = -5; ub = 5; n = 1000;

% Compute and plot Mexican hat wavelet.
    [psi,x] = mexihat(lb,ub,n);
    plot(x,psi), title('Mexican hat wavelet')
```



See Also

`waveinfo`

meyer

Purpose

Meyer wavelet.

Syntax

```
[PHI,PSI,T] = meyer(LOWB,UPPB,N)
[PHI,T] = meyer(LOWB,UPPB,N,'phi')
[PSI,T] = meyer(LOWB,UPPB,N,'psi')
```

Description

`[PHI,PSI,T] = meyer(LOWB,UPPB,N)` returns Meyer wavelet and scaling functions evaluated on an `N` point regular grid on the interval `[LOWB,UPPB]`.

`N` must be a power of two.

Output arguments are the scaling function `PHI` and the wavelet function `PSI` computed on the grid `T`. These functions have [-8 8] as effective support.

A fourth argument is allowed if only one function is required:

```
[PHI,T] = meyer(LOWB,UPPB,N,'phi')
[PSI,T] = meyer(LOWB,UPPB,N,'psi')
```

when the fourth argument is used but not equal to `'phi'` or `'psi'`. Outputs are the same as in the main option.

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

- $\hat{\psi}(\omega) = (2\pi)^{-1/2} e^{i\omega/2} \sin\left(\frac{\pi}{2}\nu\left(\frac{3}{2\pi}|\omega| - 1\right)\right)$ *if* $\frac{2\pi}{3} \le |\omega| \le \frac{4\pi}{3}$
- $\hat{\psi}(\omega) = 0$ *if* $|\omega| \notin [\frac{2\pi}{3}, \frac{8\pi}{3}]$
- $\hat{\psi}(\omega) = (2\pi)^{-1/2} e^{i\omega/2} \cos\left(\frac{\pi}{2}\nu\left(\frac{3}{4\pi}|\omega| - 1\right)\right)$ *if* $\frac{4\pi}{3} \le |\omega| \le \frac{8\pi}{3}$

where $\nu(a) = a^4(35 - 84a + 70a^2 - 20a^3), \qquad a \in [0,1]$

- $\hat{\phi}(\omega) = (2\pi)^{-1/2}\cos\left(\frac{\pi}{2}\nu\left(\frac{3}{2\pi}|\omega| - 1\right)\right) \qquad if \qquad \frac{2\pi}{3} \le |\omega| \le \frac{4\pi}{3}$

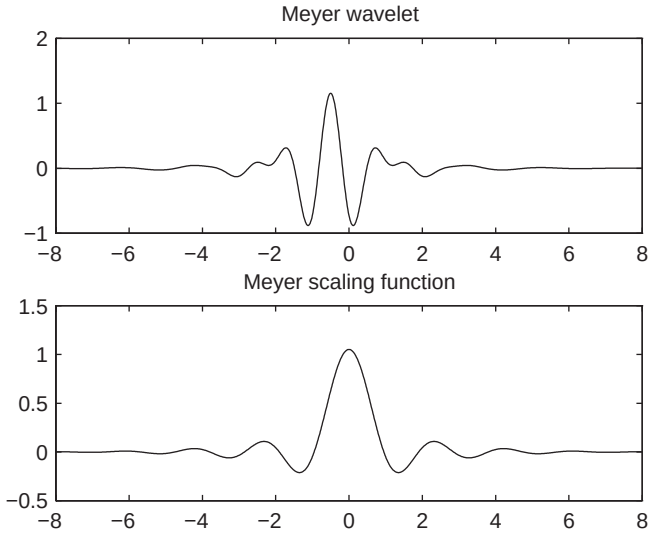
- $\hat{\phi}(\omega) = (2\pi)^{-1/2} \qquad if \qquad |\omega| \le \frac{2\pi}{3},$

- $\hat{\phi}(\omega) = 0 \qquad if \qquad |\omega| > \frac{4\pi}{3}$

By changing the auxiliary function (see `meyeraux`), you get a family of different wavelets. For the required properties of the auxiliary function ν, see References in Chapter 6.

Examples

```
% Set effective support and grid parameters.
    lowb = -8; uppb = 8; n = 1024;

% Compute and plot Meyer wavelet and scaling function.
    [phi,psi,x] = meyer(lowb,uppb,n);
    subplot(211), plot(x,psi)
    title('Meyer wavelet')
    subplot(212), plot(x,phi)
    title('Meyer scaling function')
```



Algorithm

Starting from an explicit form of the Fourier transform $\hat{\phi}$ of ϕ, `meyer` computes the values of $\hat{\phi}$ on a regular grid and then the values of ϕ are computed using `instdfft`, the inverse nonstandard discrete FFT.

The procedure for ψ is along the same lines.

See Also

`meyeraux`, `wavefun`, `waveinfo`

References

I. Daubechies (1992), "Ten lectures on wavelets," CBMS-NSF conference series in applied mathematics. SIAM Ed. pp 117-119, 137, 152.

Purpose

Meyer wavelet auxiliary function.

Syntax

```
Y = meyeraux(X)
```

Description

`Y = meyeraux(X)` returns values of the auxiliary function used for Meyer wavelet generation evaluated at the elements of the vector or matrix `X`.

The function is

$$35x^4 - 84x^5 + 70x^6 - 20x^7$$

See Also

`meyer`

morlet

Purpose

Morlet wavelet.

Syntax

```
[PSI,X] = morlet(LB,UB,N)
```

Description

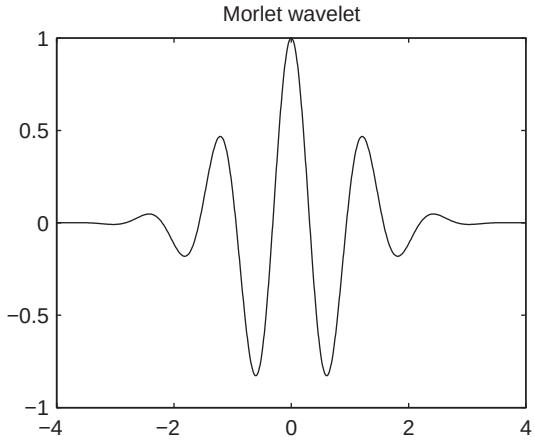
`[PSI,X] = morlet(LB,UB,N)` returns values of the Morlet wavelet on an `N` point regular grid, `X`, on the interval `[LB,UB]`.

Output arguments are the wavelet function `PSI` computed on the grid `X`, and the grid `X`. This wavelet has [-4 4] as effective support.

$$\psi(x) = e^{-x^2/2}\cos(5x)$$

Examples

```
% Set effective support and grid parameters.
    lb = -4; ub = 4; n = 1000;
% Compute and plot Morlet wavelet.
    [psi,x] = morlet(lb,ub,n);
    plot(x,psi), title('Morlet wavelet')
```



See Also

`waveinfo`

Purpose

Node ascendants.

Syntax

```
A = nodeasc(T,N)
A = nodeasc(T,N,'deppos')
```

Description

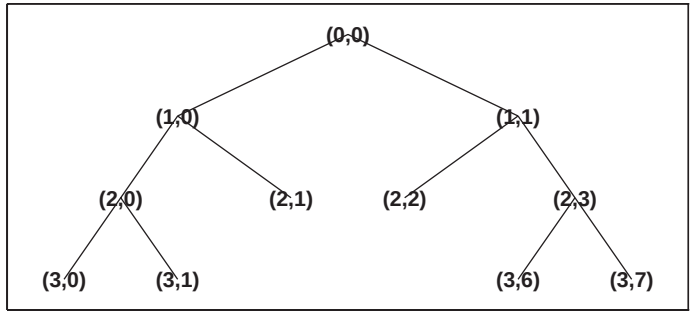
nodeasc is a tree management utility.

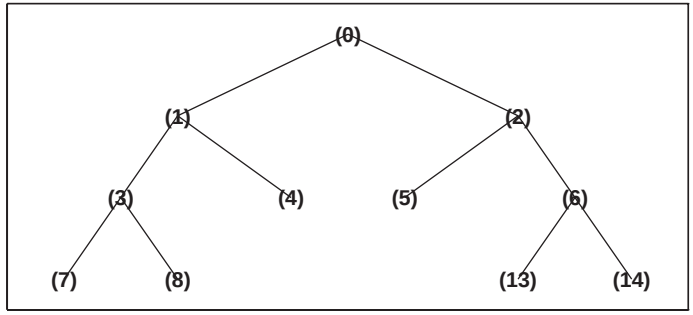
`A = nodeasc(T,N)` returns the indices of all the ascendants of the node `N` in the tree structure `T`. `N` can be the index node or the depth and position of node. `A` is a column vector with `A(1)` = index of node `N`.

`A = nodeasc(T,N,'deppos')` is a matrix that contains the depths and positions of all ascendants. `A(i,1)` is the depth of `i`-th ascendant and `A(i,2)` is the position of `i`-th ascendant.

Examples

```
% Create binary tree of depth 3.
    t = maketree(2,3);
    tt = nodejoin(t,5);
    tt = nodejoin(tt,4);
    plottree(tt)
```





```
% Node descendants.
    nodedesc(tt,2)
ans =
    7
    3
    1
    0

    nodedesc(tt,2,'deppos')
ans =

    nodeasc(tt,[2 2])
ans =
    5
    2
    0

    nodeasc(tt,[2 2],'deppos')
ans =
    2     2
    1     1
    0     0
```

See Also maketree, nodedesc, nodepar, wtreemgr

nodedesc

Purpose

Node descendants.

Syntax

```
D = nodedesc(T,N)
D = nodedesc(T,N,'deppos')
```

Description

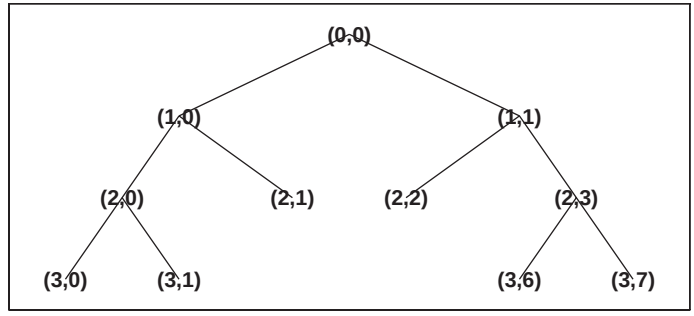
`nodedesc` is a tree management utility.

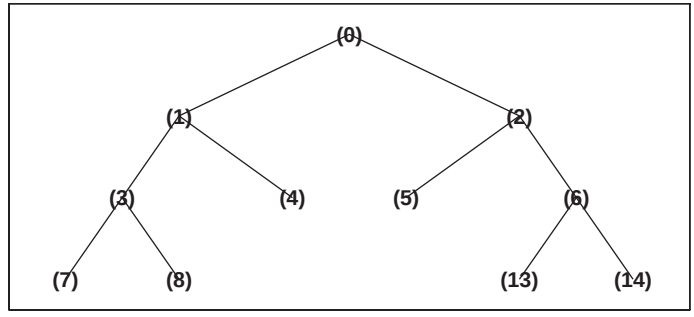
`D = nodedesc(T,N)` returns the indices of all the descendants of the node `N` in the tree structure `T`. `N` can be the index node or the depth and position of node. `D` is a column vector with `D(1)` = index of node `N`.

`D = nodedesc(T,N,'deppos')` is a matrix that contains the depths and positions of all descendants. `D(i,1)` is the depth of `i`-th descendant and `D(i,2)` is the position of `i`-th descendant.

Examples

```
% Create binary tree of depth 3.
    t = maketree(2,3);
    tt = nodejoin(t,5);
    tt = nodejoin(tt,4);
    plottree(tt)
```





```
% Node descendants.
    nodedesc(tt,2)
ans =
    2
    5
    6
   13
   14

    nodedesc(tt,2,'deppos')
ans =
    1     1
    2     2
    2     3
    3     6
    3     7

    nodedesc(tt,[2 2],'deppos')
ans =
    2     2

    nodedesc(tt,[1 1],'deppos')
ans =
    1     1
    2     2
    2     3
    3     6
    3     7

    nodedesc(tt,[1 1])
ans =
    2
    5
    6
   13
   14
```

See Also maketree, nodeasc, nodepar, wtreemgr

Purpose

Recompose node.

Syntax

```
T = nodejoin(T,N)
T = nodejoin(T)
```

Description

`nodejoin` is a tree management utility.

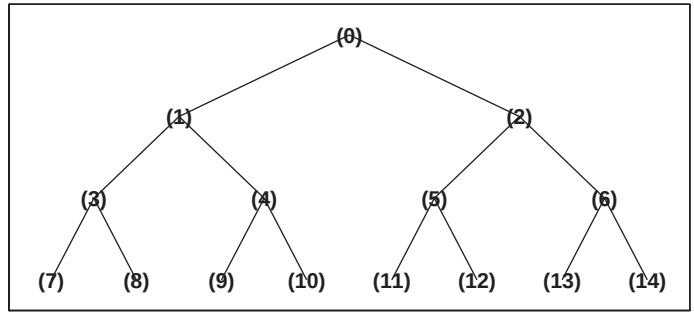
`T = nodejoin(T,N)` returns the modified tree structure `T` corresponding to a recomposition of the node `N`.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

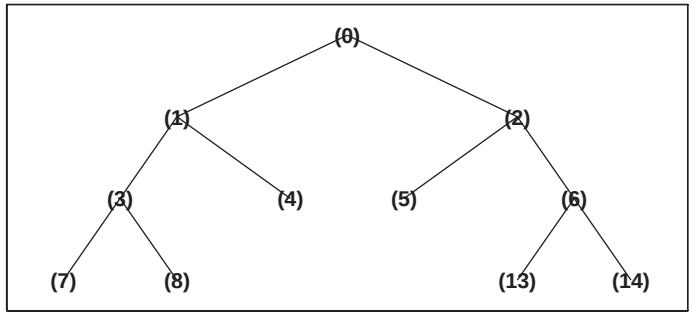
`T = nodejoin(T)` is equivalent to `T = nodejoin(T,0)`.

Examples

```
% Create binary tree of depth 3.
    t = maketree(2,3);
% Plot tree structure t.
    plottree(t)
```



```
% Merge nodes of indices 4 and 5.
    tt = nodejoin(t,5);
    tt = nodejoin(tt,4);
% Plot new tree structure tt.
    plottree(tt)
```



See Also maketree, nodesplt, wtreemgr

Purpose

Node parent.

Syntax

```
F = nodepar(T,N)
F = nodepar(T,N,'deppos')
```

Description

`nodepar` is a tree management utility.

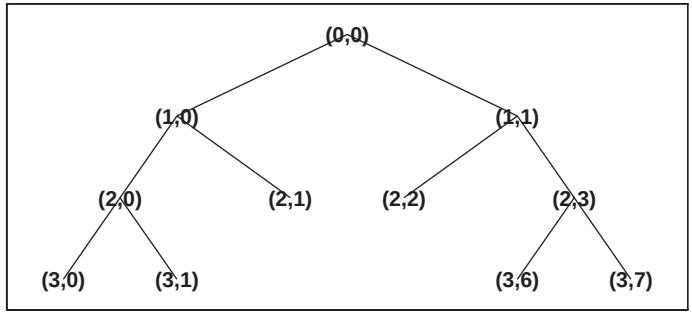
`F = nodepar(T,N)` returns the indices of the "parent(s)" of the nodes `N` in the tree structure `T`. `N` can be a column vector containing the indices of nodes or a matrix that contains the depths and positions of nodes. In the last case, `N(i,1)` is the depth of `i`-th node and `N(i,2)` is the position of `i`-th node.

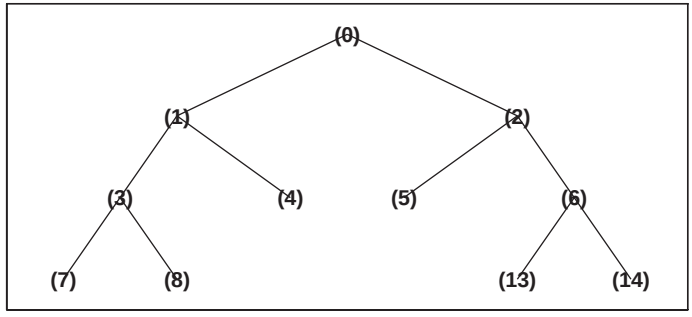
`F = nodepar(T,N,'deppos')` is a matrix that contains the depths and positions of returned nodes. `F(i,1)` is the depth of `i`-th node and `F(i,2)` is the position of `i`-th node.

`nodepar(T,0)` or `nodepar(T,[0,0])` returns `-1`.
`nodepar(T,0,'deppos')` or `nodepar(T,[0,0],'deppos')` returns `[-1,0]`.

Examples

```
% Create binary tree of depth 3.
    t = maketree(2,3);
    tt = nodejoin(t,5);
    tt = nodejoin(tt,4);
    plottree(tt)
```






```
% Nodes parent.
    nodepar(tt,[2 2],'deppos')
ans =
    1     1

    nodepar(tt,[1;7;14])
ans =
    0
    3
    6
```

See Also

maketree, nodeasc, nodedesc, wtreemgr

Purpose

Split (decompose) node.

Syntax

```
T = nodesplt(T,N)
```

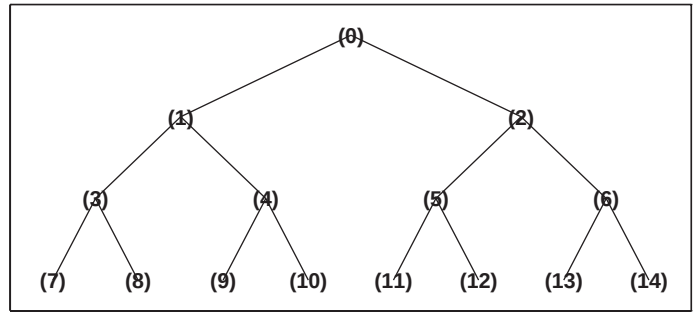
Description

`nodesplt` is a tree management utility.

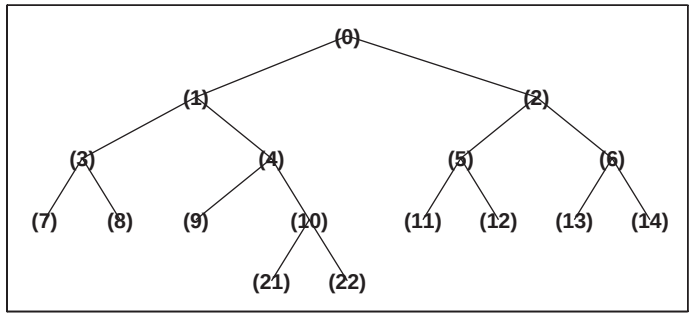
`T = nodesplt(T,N)` returns the modified tree structure `T` corresponding to the decomposition of the node `N`.

Examples

```
% Create binary tree (tree of order 2) of depth 3.
    t = maketree(2,3);

% Plot tree structure t.
    plottree(t)
```



```
Split node of index 10.
    tt = nodesplt(t,10);

% Plot new tree structure tt.
    plottree(tt)
```



See Also maketree, nodejoin, wtreemgr

Purpose

Nonstandard 1-D fast Fourier transform.

Syntax

```
[XHAT,OMEGA] = nstdfft(X,LOWB,UPPB)
```

Description

`nstdfft` is a general mathematical utility.

`[XHAT,OMEGA] = nstdfft(X,LOWB,UPPB)` returns a non-standard FFT of signal `X` sampled on a power of 2 regular grid (not necessarily integers) on the interval `[LOWB,UPPB]`.

Output arguments are `XHAT`, the shifted `FFT` of `X` computed on the interval `OMEGA` given by `OMEGA = [-n:2:n-2] / (2*(UPPB - LOWB))`, where `n` is the length of `X`. Outputs are vectors of length `n`.

Length of `X` must be a power of two.

Algorithm

Given $N = 2^q$ observations between two bounds l and u: $x_1, x_2, \ldots, x_N$, which are regularly sampled from a continuous signal f:

$x_k = f(l + (k - 1)\delta)$ for $k = 1$ to N where $\delta = (u - 1)/N$

`nstdfft` computes approximations of the continuous Fourier transform coefficients: $\hat{f}(\omega) = \int_l^u f(t)e^{-2i\pi\omega t}dt$ for $\omega = \frac{-1}{2\delta} : \frac{1}{N\delta} : \frac{1}{2\delta} - \frac{1}{N\delta}$ using the standard discrete fast fourier transform `fft`.

For a given frequency ω: $\hat{f}(\omega) = \int_l^u f(t)e^{-2i\pi\omega t}dt$ can be rewritten as $\hat{f}(\omega) = N\delta e^{-2i\pi\omega l}\int_0^1 f(l + sN\delta)e^{-2i\pi\omega tsN\delta}ds$ using $t = sN\delta + l$.

The integral term can be approximated by the finite sum:

$$\frac{1}{N}\sum_{k=1}^{N} fl + (k-1)\delta)e^{-2i\pi\omega(k-1)\delta}.$$

Since $\omega = \frac{-1}{2\delta} : \frac{1}{N\delta} : \frac{1}{2\delta} - \frac{1}{N\delta}$

then $j = \omega\delta N = \frac{-N}{2} : 1 : \frac{N}{2} - 1$, which are the usual frequencies of the discrete Fourier transform.

It turns out that $\hat{f}(\omega)$ can be approximated by:

$$\delta e^{-2i\pi\omega l} \sum_{k=1}^{N} x_k \omega^{(k-1)j} \text{ where } \omega_N = e^{-2i\pi/N}$$

which can be computed using standard `fft` and a normalization. The function `instdfft` inverts this transform in three steps: normalization, use of `ifft`, and translation in time.

The length of `X` must be a power of two.

See Also

`fft`, `fftshift`, `instdfft`

Purpose

Number of terminal nodes.

Syntax

```
NB = ntnode(T)
```

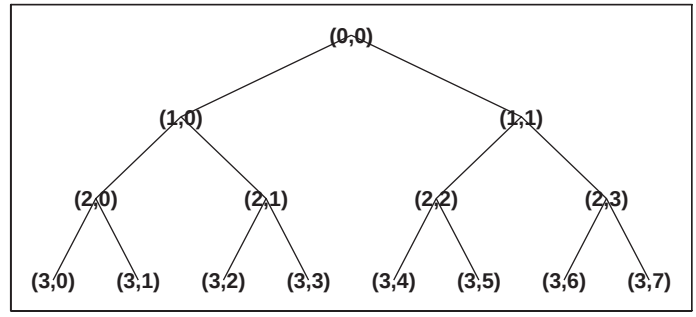
Description

`ntnode` is a tree management utility.

`NB = ntnode(T)` returns the number of terminal nodes in the tree structure `T`.

Examples

```
% Create binary tree (tree of order 2) of depth 3.
    t = maketree(2,3);

% Plot tree structure t.
    plottree(t)
```




```
% Number of terminal nodes.
    ntnode(t)
ans =
    8
```

See Also

`maketree`, `wtreemgr`

num2mstr

Purpose

Convert number to string in maximum precision.

Syntax

```
S = num2mstr(N)
```

Description

`num2mstr` is a general utility.

`S = num2mstr(N)` converts real numbers of input matrix `N` to string output vector `S`, in maximum precision.

See Also

`num2str`

Purpose

Orthogonal wavelet filter set.

Syntax

```
[Lo_D,Hi_D,Lo_R,Hi_R] = orthfilt(W)
```

Description

`[Lo_D,Hi_D,Lo_R,Hi_R] = orthfilt(W)` computes the four filters associated with the scaling filter `W` corresponding to a wavelet:

`Lo_D`	Decomposition low-pass filter
`Hi_D`	Decomposition high-pass filter
`Lo_R`	Reconstruction low-pass filter
`Hi_R`	Reconstruction high-pass filter

For an orthogonal wavelet, in the multiresolution framework, we start with the scaling function ϕ and the wavelet function ψ. One of the fundamental relations is the twin-scale relation:

$$\frac{1}{2}\phi\left(\frac{x}{2}\right) = \sum_{n \in Z} w_n \phi(x - n).$$

All the filters used in `DWT` and `IDWT` are intimately related to the sequence $(w_n)_{n \in Z}$. Clearly if ϕ is compactly supported, the sequence (w_n) is finite and can be viewed as a FIR filter. The scaling filter `W` is:

- A low-pass FIR filter
- Of length $2N$
- Of sum 1
- Of norm $\frac{1}{\sqrt{2}}$

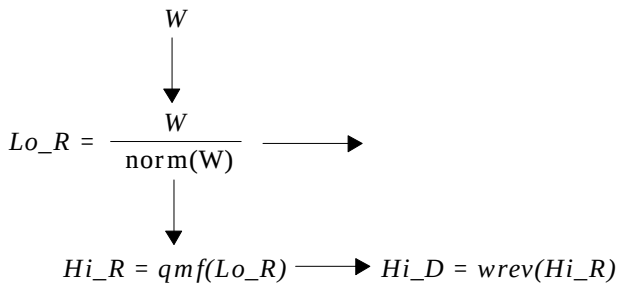
For example, for the db3 scaling filter:

```
load db3
db3
db3 =
    0.2352 0.5706 0.3252 -0.0955 -0.0604 0.0249

sum(db3)
ans =
    1.000

norm(db3)
ans =
    0.7071
```

From filter W, we define four FIR filters, of length 2N and norm 1, organized as follows:

Filters	Low-pass	High-pass
Decomposition	Lo_D	Hi_D
Reconstruction	Lo_R	Hi_R

The four filters are computed using the following scheme:

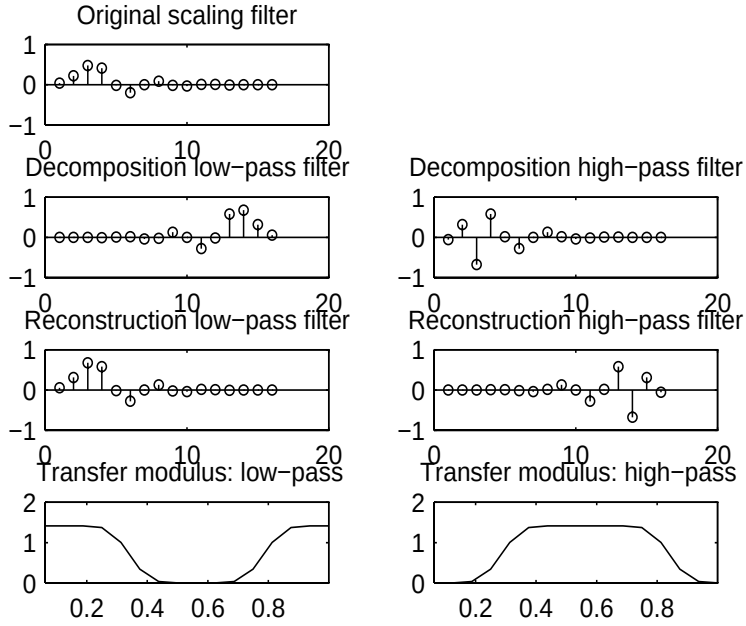


where qmf is such that Hi_R and Lo_R are quadrature mirror filters (i.e. $Hi\_R(k) = (-1)^k Lo\_R(2N - 1 - k)$), and where wrev flips the filter coefficients. So Hi_D and Lo_D are also quadrature mirror filters. The computation of these filters is performed using orthfilt.

Examples

```
% Load scaling filter.
    load db8; w = db8;
    subplot(421); stem(w);
    title('Original scaling filter');

% Compute the four filters.
    [Lo_D,Hi_D,Lo_R,Hi_R] = orthfilt(w);
    subplot(423); stem(Lo_D);
    title('Decomposition low-pass filter');
    subplot(424); stem(Hi_D);
    title('Decomposition high-pass filter');
    subplot(425); stem(Lo_R);
    title('Reconstruction low-pass filter');
    subplot(426); stem(Hi_R);
    title('Reconstruction high-pass filter');

% Check for orthonormality.
    df = [Lo_D;Hi_D];
    rf = [Lo_R;Hi_R];
    id = df*df'

id =
    1.0000 -0.0000
    -0.0000 1.0000
    id = rf*rf'

id =
    1.0000 0.0000
    0.0000 1.0000

% Check for orthogonality by dyadic translation, for example:
    df = [Lo_D 0 0;Hi_D 0 0];
    dft = [0 0 Lo_D; 0 0 Hi_D];
    zer = df*dft'

zer =

    1.0e-12 *
    -0.1883 0.0000
    -0.0000 -0.1883
```

```
% High and low frequency illustration.
    fftld = fft(Lo_D); ffthd = fft(Hi_D);
    freq = [1:length(Lo_D)]/length(Lo_D);
    subplot(427); plot(freq,abs(fftld));
    title('Transfer modulus: low-pass')
    subplot(428); plot(freq,abs(ffthd));
    title('Transfer modulus: high-pass')
```



See Also `biorfilt`, `qmf`, `wfilters`

References

I. Daubechies (1992), “Ten lectures on wavelets,” CBMS-NSF conference series in applied mathematics. SIAM Ed. pp 117-119, 137, 152.

Purpose

Plot tree.

Syntax

```
plottree(T)
```

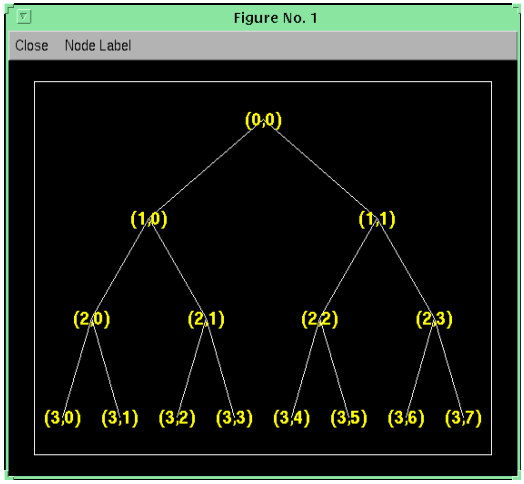
Description

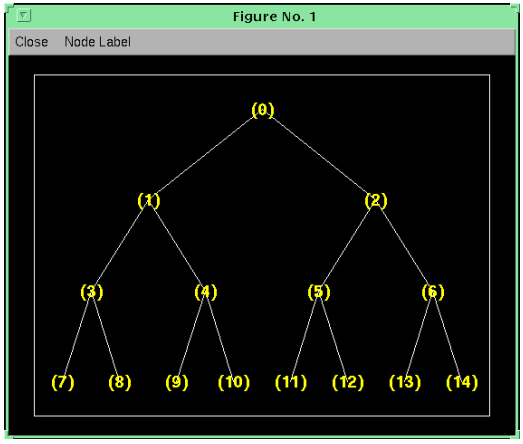
`plottree` is a graphical tree management utility.

`plottree(T)` plots the tree structure `T` (see `maketree`).

Examples

```
% Create binary tree of depth 3.
    t = maketree(2,3);

% Plot tree structure t.
    plottree(t)

% Creates a figure containing the tree
% and a simple menu bar allowing:
%     - to close the window
%     - to choose node labeling mode between
%     index and depth-position.
```





See Also `maketree`, `wpdec`, `wpdec2`

Purpose

Quadrature mirror filter.

Syntax

```
Y = qmf(X,P)
Y = qmf(X)
```

Description

`Y = qmf(X,P)` changes the signs of the even index entries of the reversed vector filter coefficients `X` if `P` is even. If `P` is odd the same holds for odd index entries. `Y = qmf(X)` is equivalent to `Y = qmf(X,0)`.

Let `x` be a finite energy signal. Two filters F_0 and F_1 are quadrature mirror filters (QMF) if, for any x:

$$\|\gamma_0\|^2 + \|\gamma_1\|^2 = \|x\|^2$$

where y_0 is a decimated version of the signal x filtered with F_0 so y_0 is defined by $x_0 = F_0(x)$ and $y_0(n) = x_0(2n)$, and similarly, y_1 is defined by $x_1 = F_1(x)$ and $y_1(n) = x_1(2n)$. This property ensures a perfect reconstruction of the associated two-channel filter banks scheme (See Strang-Nguyen p. 103).

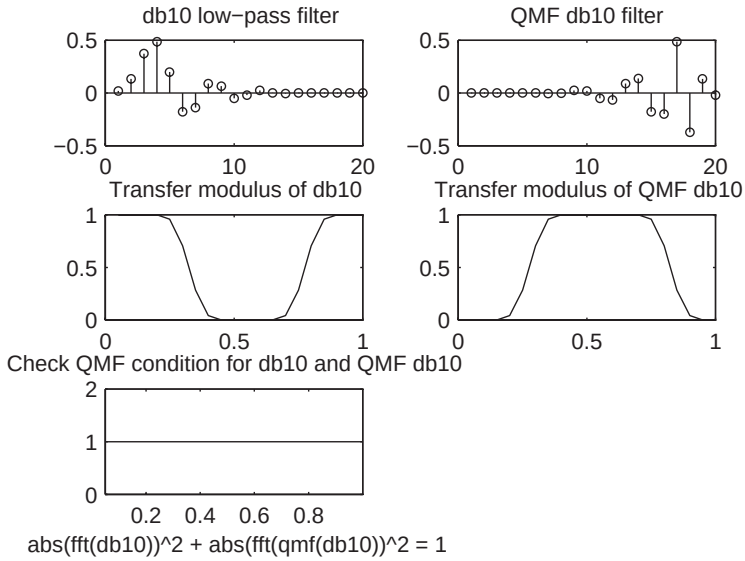
For example, if F_0 is a Daubechies scaling filter and F_1 = `qmf`(F_0) then the transfer functions $F_0(z)$ and $F_1(z)$ of the filters F_0 and F_1 satisfy the condition (see the example for *db*10):

$$|F_0(z)|^2 + |F_1(z)|^2 = 1$$

Examples

```
% Load scaling filter associated with an orthogonal wavelet.
load db10;
subplot(321); stem(db10); title('db10 low-pass filter');

% Compute the quadrature mirror filter.
qmfdb10 = qmf(db10);
subplot(322); stem(qmfdb10); title('QMF db10 filter');
% Check for frequency condition (necessary for orthogonality): %
abs(fft(filter))^2 + abs(fft(qmf(filter))^2 = 1 at each
% frequency.
m = fft(db10);
mt = fft(qmfdb10);
```

```
freq = [1:length(db10)]/length(db10);
subplot(323); plot(freq,abs(m));
title('Transfer modulus of db10')
subplot(324); plot(freq,abs(mt));
title('Transfer modulus of QMF db10')

subplot(325); plot(freq,abs(m).^2 + abs(mt).^2);
title('Check QMF condition for db10 and QMF db10')
xlabel(' abs(fft(db10))^2 + abs(fft(qmf(db10))^2 = 1')

% Check for orthonormality.
df = [db10;qmfdb10]*sqrt(2);
id = df*df'

id =
    1.0000 0.0000
    0.0000 1.0000
```



References

G. Strang, T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press

Purpose

Symlets wavelet filters.

Syntax

```
F = symwavf('symname')
```

Description

`F = symwavf('symname')` returns the scaling filter associated with the symlet wavelet specified by '*symname*'. Possible values for `N` are: 2, 3, 4, 5, 6, 7 or 8.

Examples

```
% Compute the scaling filter corresponding to wavelet sym4.
    w = symwavf('sym4')

w =
 Columns 1 through 7
    0.0228 -0.0089 -0.0702 0.2106 0.5683 0.3519 -0.0210
 Column 8
    -0.0536
```

See Also

`waveinfo`

thselect

Purpose

Threshold selection for de-noising.

Syntax

```
THR = thselect(X,TPTR)
```

Description

`thselect` is a one-dimensional de-noising oriented function.

`THR = thselect(X,TPTR)` returns threshold `X`-adapted value using selection rule defined by string `TPTR`.

Available selection rules are:

`TPTR` = `'rigrsure'`, adaptive threshold selection using principle of Stein's Unbiased Risk Estimate.

`TPTR` = `'heursure'`, heuristic variant of the first option.

`TPTR` = `'sqtwolog'`, threshold is sqrt(2*log(length(X))).

`TPTR` = `'minimaxi'`, minimax thresholding.

Threshold selection rules are based on the underlying model $y = f(t) + e$ where e is a white noise $N(0,1)$. Dealing with unscaled or nonwhite noise can be handled using rescaling output threshold `THR` (see `SCAL` parameter in `wden`).

Available options are:

- `tptr` = `'rigrsure'` uses for the soft threshold estimator a threshold selection rule based on Stein's Unbiased Estimate of Risk (quadratic loss function). One gets an estimate of the risk for a particular threshold value t. Minimizing the risks in `t` gives a selection of the threshold value.
- `tptr` = `'sqtwolog'` uses a fixed form threshold yielding minimax performance multiplied by a small factor proportional to log(length(s)).
- `tptr` = `'heursure'` is a mixture of the two previous options. As a result, if the signal to noise ratio is very small, the SURE estimate is very noisy. If such a situation is detected, the fixed form threshold is used.
- `tptr` = `'minimaxi'` uses a fixed threshold chosen to yield minimax performance for mean square error against an ideal procedure. The minimax principle is used in statistics in order to design estimators. Since the de-noised signal can be assimilated to the estimator of the unknown regression function, the minimax estimator is the one that realizes the

minimum of the maximum mean square error obtained for the worst function in a given set.

Examples

```
% Generate Gaussian white noise.
    init = 2055415866; randn('seed',init);
    x = randn(1,1000);

% Find threshold for each selection rule.
    % adaptive threshold using SURE.
    thr = thselect(x,'rigrsure')

thr =
    1.8065

% Fixed form threshold.
    thr = thselect(x,'sqtwolog')

thr =
    3.7169

% Heuristic variant of the first options.
    thr = thselect(x,'heursure')

thr =
    3.7169

% Minimax threshold.
    thr = thselect(x,'minimaxi')

thr =
    2.2163
```

See Also

wden

References

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

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

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

tnodes

Purpose

Terminal nodes.

Syntax

```
N = tnodes(T)
N = tnodes(T,'deppos')
[N,K] = tnodes(T)
[N,K] = tnodes(T,'deppos')
```

Description

`tnodes` is a tree management utility.

`N = tnodes(T)` returns the indices of terminal nodes of the tree structure `T` (see `maketree`). `N` is a column vector.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

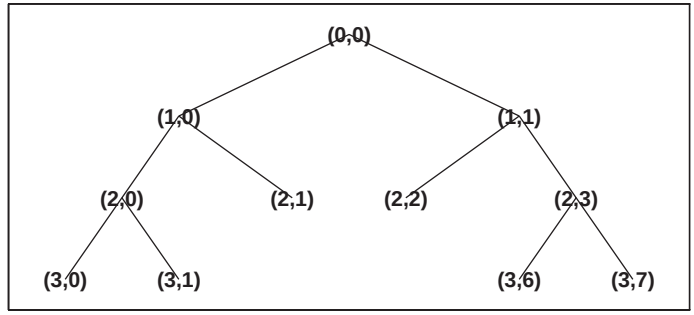
`N = tnodes(T,'deppos')` returns a matrix `N` which contains the depths and positions of terminal nodes.

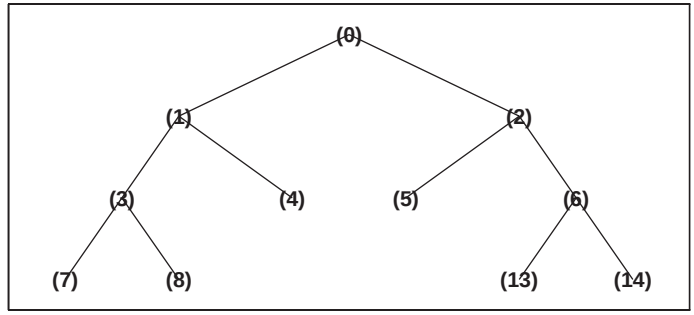
`N(i,1)` is the depth of `i`-th terminal node. `N(i,2)` is the position of `i`-th terminal node.

For `[N,K] = tnodes(T)` or `[N,K] = tnodes(T,'deppos')`, `M = N(K)` are the indices reordered in tree `T`, from left to right.

Examples

```
% Create initial tree.
    ord = 2; t = maketree(ord,3); % Binary tree of depth 3.
    tt = nodejoin(t,5);
    tt = nodejoin(tt,4);
    plottree(tt)
```






```
% List terminal nodes (index).
    tnodes(tt)

ans =
    4
    5
    7
    8
    13
    14
```

```
% List terminal nodes (depth-position).
    tnodes(tt,'deppos')

ans =
    2   1
    2   2
    3   0
    3   1
    3   6
    3   7
```

See Also maketree, wtreemgr

Purpose

Tree depth.

Syntax

```
D = treedpth(T)
```

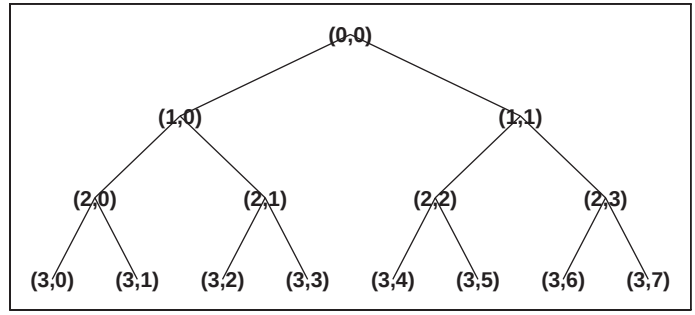
Description

`treedpth` is a tree management utility.

`D = treedpth(T)` returns the depth `D` of the tree `T`.

Examples

```
% Create binary tree (tree of order 2) of depth 3.
    t = maketree(2,3);

% Plot tree structure t.
    plottree(t)
```




```
% Tree depth.
    treedpth(t)

ans =
    3
```

See Also

`maketree`, `wtreemgr`

treeord

Purpose

Tree order.

Syntax

```
ORD = treeord(T)
```

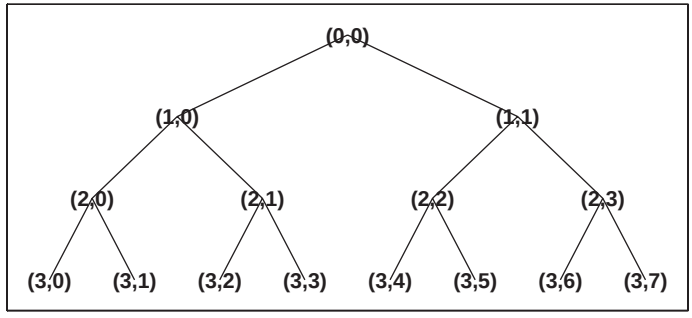
Description

`treeord` is a tree management utility.

`ORD = treeord(T)` returns the order `ORD` of the tree `T`.

Examples

```
% Create binary tree (tree of order 2) of depth 3.
    t = maketree(2,3);

% Plot tree structure t.
    plottree(t)
```




```
% Tree order.
    treeord(t)

ans =
    2
```

See Also

`maketree`, `wtreemgr`

Purpose

Direct reconstruction from 1-D wavelet coefficients.

Syntax

```
Y = upcoef(O,X,'wname',N)
Y = upcoef(O,X,'wname',N,L)
Y = upcoef(O,X,Lo_R,Hi_R,N)
Y = upcoef(O,X,Lo_R,Hi_R,N,L)
Y = upcoef(O,X,'wname')
Y = upcoef(O,X,Lo_R,Hi_R)
```

Description

`upcoef` is a one-dimensional wavelet analysis function.

`Y = upcoef(O,X,'wname',N)` computes the `N` steps reconstructed coefficients of vector `X`.

'*wname*' is a string containing the wavelet name.

`N` must be a strictly positive integer.

If `O = 'a'`, approximation coefficients are reconstructed.

If `O = 'd'`, detail coefficients are reconstructed.

`Y = upcoef(O,X,'wname',N,L)` computes the `N` steps reconstructed coefficients of vector `X` and takes the length-`L` central portion of the result. Instead of giving the wavelet name, you can give the filters.

For `Y = upcoef(O,X,Lo_R,Hi_R,N)` or `Y = upcoef(O,X,Lo_R,Hi_R,N,L)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

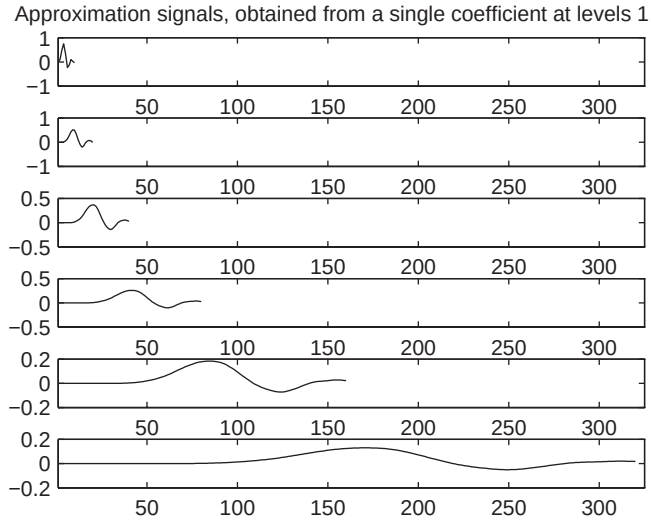
`Y = upcoef(O,X,'wname')` is equivalent to `Y = upcoef(O,X,'wname',1)`.

`Y = upcoef(O,X,Lo_R,Hi_R)` is equivalent to `Y = upcoef(O,X,Lo_R,Hi_R,1)`.

Examples

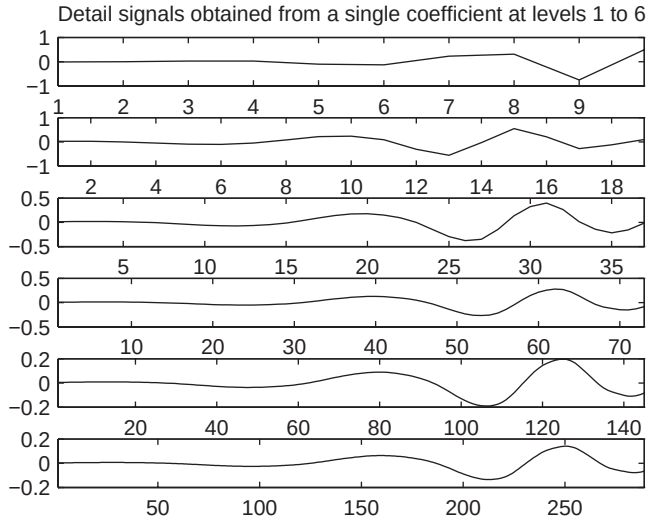
```
% Approximation signals, obtained from a single coefficient
% at levels 1 to 6.
cfs = [1];  % Decomposition reduced a single coefficient.
essup = 10; % Essential support of the scaling filter db6.
figure(1)
for i=1:6
```

```
    % Reconstruct at the top level an approximation
    % which is equal to zero except at level i where only
    % one coefficient is equal to 1.
    rec = upcoef('a',cfs,'db6',i);

    % essup is the essential support of the
    % reconstructed signal.
    subplot(6,1,i),h = plot(rec(1:essup));
    set(get(h,'parent'),'xlim',[1 325]);
    essup = essup*2;

end
subplot(611)
title(['Approximation signals, obtained from a single ' ...
     'coefficient at levels 1 to 6'])
```




```
% The same can be done for details.
% Details signals, obtained from a single coefficient
% at levels 1 to 6.

cfs = [1];
mi = 12; ma = 30;   % Essential support of
                    % the wavelet filter db6.
```

```
rec = upcoef('d',cfs,'db6',1);
figure(2)
subplot(611), plot(rec(3:12))
for i=2:6
    % Reconstruct at top level a single detail
    % coefficient at level i.
    rec = upcoef('d',cfs,'db6',i);
    subplot(6,1,i), plot(rec(mi*2^(i-2):ma*2^(i-2)))
end
subplot(611)
title(['Detail signals obtained from a single ' ...
    'coefficient at levels 1 to 6'])
```



Algorithm

`upcoef` is equivalent to the `N` times repeated use of the inverse wavelet transform.

See Also

`idwt`

upcoef2

Purpose

Direct reconstruction from 2-D wavelet coefficients.

Syntax

```
Y = upcoef2(O,X,'wname',N,S)
Y = upcoef2(O,X,Lo_R,Hi_R,N,S)
Y = upcoef2(O,X,'wname',N)
Y = upcoef2(O,X,Lo_R,Hi_R,N)
Y = upcoef2(O,X,'wname')
Y = upcoef2(O,X,Lo_R,Hi_R)
```

Description

`upcoef2` is a two-dimensional wavelet analysis function.

`Y = upcoef2(O,X,'wname',N,S)` computes the `N` steps reconstructed coefficients of matrix `X` and takes the central part of size `S`. '*wname*' is a string containing the name of the wavelet.

If `O` = 'a', approximation coefficients are reconstructed; otherwise if `O = 'h'` (`'v'` or `'d'` respectively), horizontal (vertical or diagonal respectively) detail coefficients are reconstructed. `N` must be a strictly positive integer.

Instead of giving the wavelet name, you can give the filters.

For `Y = upcoef2(O,X,Lo_R,Hi_R,N,S)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

`Y = upcoef2(O,X,'wname',N)` or `Y = upcoef2(O,X,Lo_R,Hi_R,N)` return the computed result without any truncation.

`Y = upcoef2(O,X,'wname')` is equivalent to `Y = upcoef2(O,X,'wname',1)`.

`Y = upcoef2(O,X,Lo_R,Hi_R)` is equivalent to
`Y = upcoef2(O,X,Lo_R,Hi_R,1)`.

Examples

```
% Load original image.
    load woman;
    % X contains the loaded image.

% Perform decomposition at level 2
% of X using db4.
    [c,s] = wavedec2(X,2,'db4');

% Reconstruct approximation and details
% at level 1, from coefficients.
% This can be done using wrcoef2, or
% equivalently using:
%
% Step 1: Extract coefficients from the
% decomposition structure [c,s].
%
% Step 2: Reconstruct using upcoef2.

    siz = s(size(s,1),:);

    ca1 = appcoef2(c,s,'db4',1);
    a1 = upcoef2('a',ca1,'db4',1,siz);

    chd1 = detcoef2('h',c,s,1);
    hd1 = upcoef2('h',chd1,'db4',1,siz);

    cvd1 = detcoef2('v',c,s,1);
    vd1 = upcoef2('v',cvd1,'db4',1,siz);

    cdd1 = detcoef2('d',c,s,1);
    dd1 = upcoef2('d',cdd1,'db4',1,siz);
```

Algorithm

See `upcoef`.

See Also

`idwt2`

upwlev

Purpose

Single-level reconstruction of 1-D wavelet decomposition.

Syntax

```
[NC,NL,cA] = upwlev(C,L,'wname')
[NC,NL,cA] = upwlev(C,L,Lo_R,Hi_R)
```

Description

`upwlev` is a one-dimensional wavelet analysis function.

`[NC,NL,cA] = upwlev(C,L,'wname')` performs the single-level reconstruction of the wavelet decomposition structure `[C,L]` giving the new one `[NC,NL]`, and extracts the last approximation coefficients vector `cA`.

`[C,L]` is a decomposition at level `n = length(L)-2`, so `[NC,NL]` is the same decomposition at level `n-1` and `cA` is the approximation coefficients vector at level `n`.

'*wname*' is a string containing the wavelet name, `C` is the original wavelet decomposition vector, and `L` the corresponding bookkeeping vector (for detailed storage information, see `wavedec`).

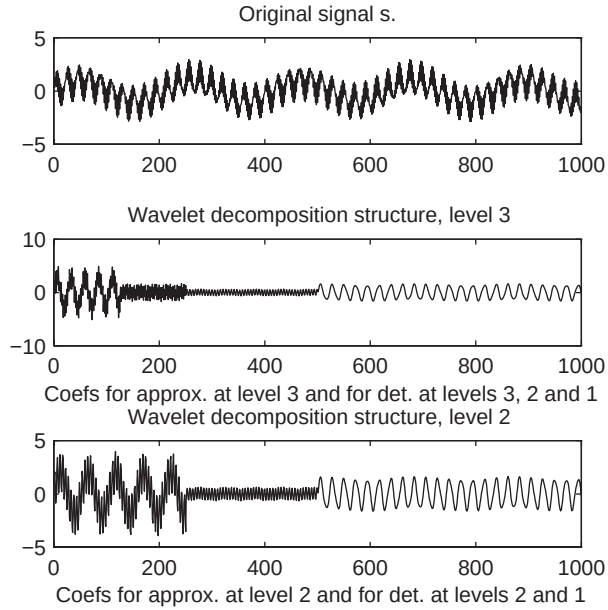
Instead of giving the wavelet name, you can give the filters.

For `[NC,NL,cA] = upwlev(C,L,Lo_R,Hi_R)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

Examples

```
% Load original one-dimensional signal.
    load sumsin; s = sumsin;

% Perform decomposition at level 3 of s using db1.
    [c,l] = wavedec(s,3,'db1');
    subplot(311); plot(s);
    title('Original signal s.');
    subplot(312); plot(c);
    title('Wavelet decomposition structure, level 3')
    xlabel(['Coefs for approx. at level 3 ' ...
      'and for det. at levels 3, 2 and 1'])
```

```
% One step reconstruction of the wavelet decomposition
% structure at level 3 [c,l], so the new structure [nc,nl]
% is the wavelet decomposition structure at level 2.
    [nc,nl] = upwlev(c,l,'db1');
    subplot(313); plot(nc);
    title('Wavelet decomposition structure, level 2')
    xlabel(['Coefs for approx. at level 2 ' ...
      'and for det. at levels 2 and 1'])
```



See Also `idwt`, `upcoef`, `wavedec`

upwlev2

Purpose

Single-level reconstruction of 2-D wavelet decomposition.

Syntax

```
[NC,NS,cA] = upwlev2(C,S,'wname')
[NC,NS,cA] = upwlev2(C,S,Lo_R,Hi_R)
```

Description

`upwlev2` is a two-dimensional wavelet analysis function.

`[NC,NS,cA] = upwlev2(C,S,'wname')` performs the single-level reconstruction of wavelet decomposition structure `[C,S]` giving the new one `[NC,NS]`, and extracts the last approximation coefficients matrix `cA`.

`[C,S]` is a decomposition at level `n = size(S,1)-2`, so `[NC,NS]` is the same decomposition at level `n-1` and `cA` is the approximation matrix at level `n`.

'*wname*' is a string containing the wavelet name, `C` is the original wavelet decomposition vector, and `S` the corresponding bookkeeping matrix (for detailed storage information, see `wavedec2`).

For `[NC,NS,cA] = upwlev2(C,S,Lo_R,Hi_R)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

Examples

```
% Load original image.
    load woman;
    % X contains the loaded image.

% Perform decomposition at level 2
% of X using db1.
    [c,s] = wavedec2(X,2,'db1');
    sc = size(c)

sc =
    1   65536
    val_s = s

val_s =
    64   64
    64   64
    128  128
    256  256

% One step reconstruction of wavelet
% decomposition structure [c,s].
    [nc,ns] = upwlev2(c,s,'db1');
    snc = size(nc)

snc =
    1   65536
val_ns = ns
val_ns =
    128   128
    128   128
    256   256
```

See Also

idwt2, upcoef2, wavedec2

wavedec

Purpose

Multi-level 1-D wavelet decomposition.

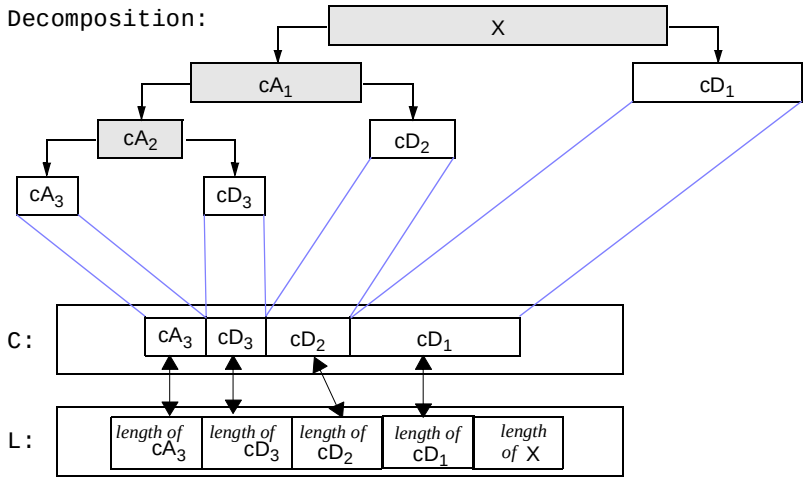
Syntax

```
[C,L] = wavedec(X,N,'wname')
[C,L] = wavedec(X,N,Lo_D,Hi_D)
```

Description

wavedec performs a multi-level one-dimensional wavelet analysis using either a specific wavelet ('*wname*', see `wfilters`) or specific wavelet decomposition filters (`Lo_D` and `Hi_D`).

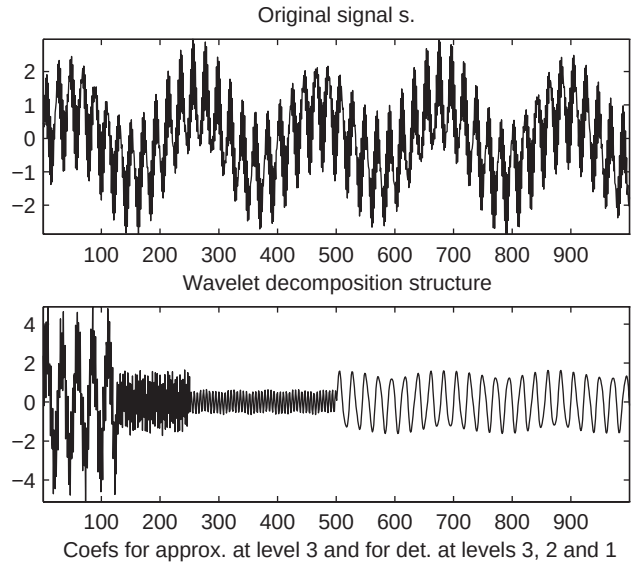
`[C,L] = wavedec(X,N,'wname')` returns the wavelet decomposition of the signal `X` at level `N`, using '*wname*'. `N` must be a strictly positive integer (see `wmaxlev`). The output decomposition structure contains the wavelet decomposition vector `C` and bookkeeping vector `L`. The structure is organized as in this level-3 decomposition example:



`[C,L] = wavedec(X,N,Lo_D,Hi_D)` returns the decomposition structure as above, given the low- and high-pass decomposition filters you specify.

Examples

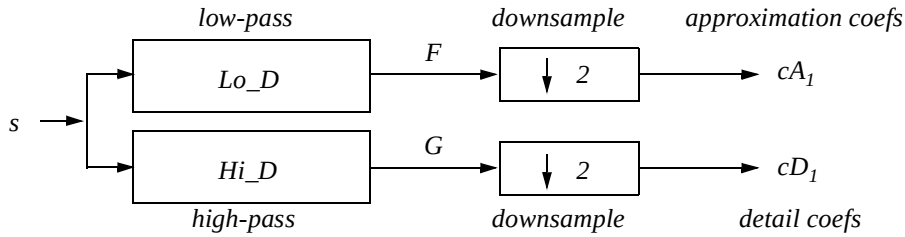
```
% Load original one-dimensional signal.
    load sumsin; s = sumsin;
% Perform decomposition at level 3 of s using db1.
    [c,l] = wavedec(s,3,'db1');
```



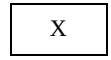
Algorithm

Given a signal s of length N, the DWT consists of $\log_2 N$ stages at most. The first step produces, starting from s, two sets of coefficients: approximation coefficients CA_1 and detail coefficients CD_1. These vectors are obtained by convolving s with the low-pass filter `Lo_D` for approximation, and with the high-pass filter `Hi_D` for detail, followed by dyadic decimation (downsampling).

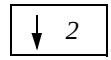
More precisely, the first step is:



where:



Convolve with filter X



Keep the even indexed elements
(We call this operation *downsampling*.)

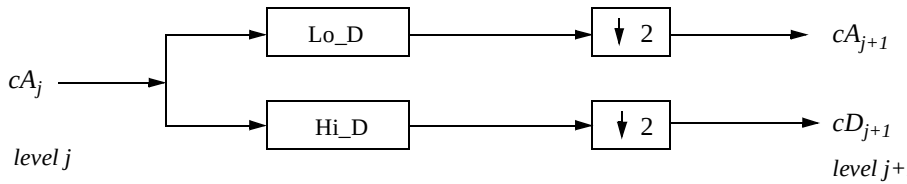
The length of each filter is equal to $2N$. If n = length(s), the signals F and G are of length $n + 2N - 1$ and the coefficients cA_1 and cD_1 are of length

$$\text{floor}\left(\frac{n-1}{2}\right) + N \quad .$$

The next step splits the approximation coefficients cA_1 in two parts using the same scheme, replacing s by cA_1, and producing cA_2 and cD_2, and so on.

One-Dimensional DWT

Decomposition step



where:

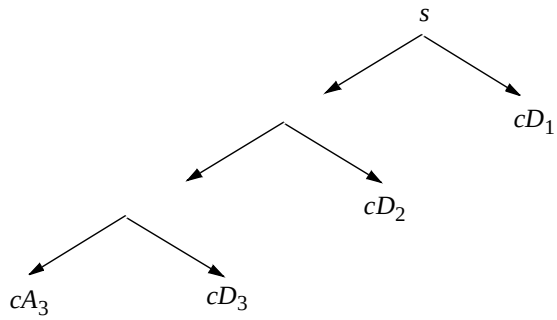
Convolve with filter X

Downsample

Initialization $cA_0 = s$

The wavelet decomposition of the signal s analyzed at level *j* has the following structure: [cA_j, cD_j, ..., cD_1].

This structure contains, for $J = 3$, the terminal nodes of the following tree:



See Also

`dwt`, `waveinfo`, `wfilters`, `wmaxlev`

References

I. Daubechies (1992), "Ten lectures on wavelets," CBMS-NSF conference series in applied mathematics. SIAM Ed.

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

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

wavedec2

Purpose

Multi-level 2-D wavelet decomposition.

Syntax

```
[C,S] = wavedec2(X,N,'wname')
[C,S] = wavedec2(X,N,Lo_D,Hi_D)
```

Description

`wavedec2` is a two-dimensional wavelet analysis function.

`[C,S] = wavedec2(X,N,'wname')` returns the wavelet decomposition of the matrix `X` at level `N`, using the wavelet named in string '*wname*' (see `wfilters`).

Outputs are the decomposition vector `C` and the corresponding bookkeeping matrix `S`.

`N` must be a strictly positive integer (see `wmaxlev`).

Instead of giving the wavelet name, you can give the filters.

For `[C,S] = wavedec2(X,N,Lo_D,Hi_D)`, `Lo_D` is the decomposition low-pass filter and `Hi_D` is the decomposition high-pass filter. The output wavelet two-dimensional decomposition structure `[C,S]` contains the wavelet decomposition vector `C` and the corresponding bookkeeping matrix `S`.

Vector `C` is organized as:

`C = [ A(N) | H(N) | V(N) | D(N) | ...`

`H(N-1) | V(N-1) | D(N-1) | ... | H(1) | V(1) | D(1) ].`

where `A`, `H`, `V`, `D`, are row vectors such that:

`A` = approximation coefficients

`H` = horizontal detail coefficients

`V` = vertical detail coefficients

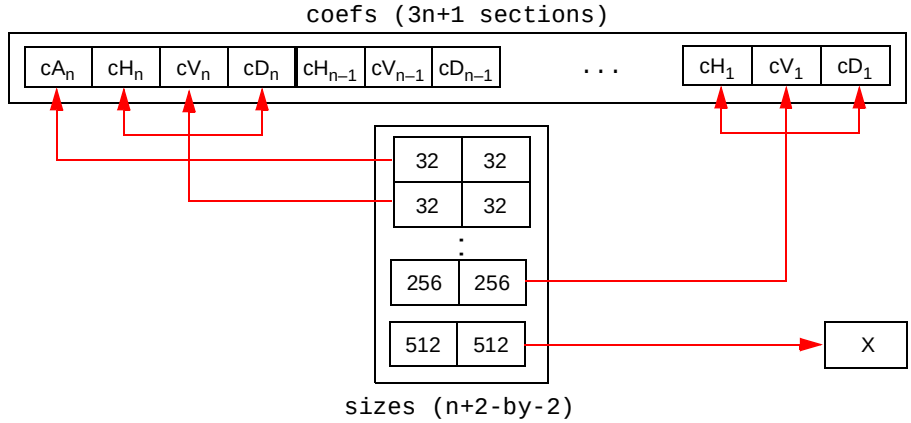
`D` = diagonal detail coefficients

each vector is the vector columnwise storage of a matrix.

Matrix `S` is such that:

`S(1,:)` = size of approximation coefficients(`N`)

`S(i,:)` = size of detail coefficients(`N-i+2`) for `i` = 2, ...`N+1` and
`S(N+2,:) = size(X)`.



Examples

```
% Load original image.
    load woman;
    % X contains the loaded image.

% Perform decomposition at level 2
% of X using db1.
    [c,s] = wavedec2(X,2,'db1');

% Decomposition structure organization.
    sizex = size(X)

sizex =
    256  256
    sizec = size(c)

sizec =
    1  65536
    val_s = s

val_s =
    64  64
    64  64
    128  128
    256  256
```

Algorithm

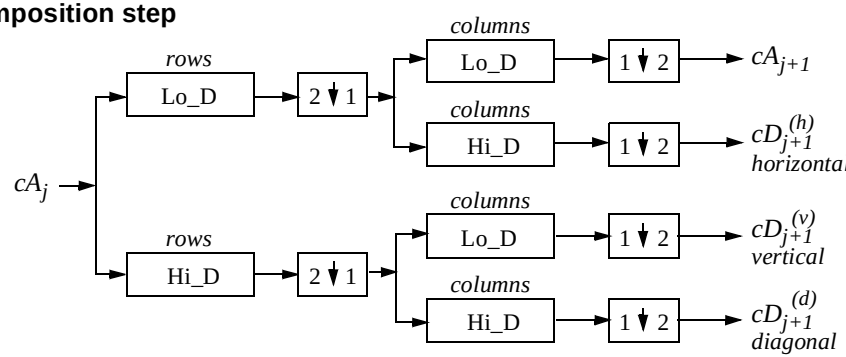
For images, an algorithm similar to the one-dimensional case is possible for two-dimensional wavelets and scaling functions obtained from one-dimensional ones by tensor product.

This kind of two-dimensional DWT leads to a decomposition of approximation coefficients at level j in four components: the approximation at level $j+1$ and the details in three orientations (horizontal, vertical, and diagonal).

The following chart describes the basic decomposition step for images:

Two-Dimensional DWT

Decomposition step

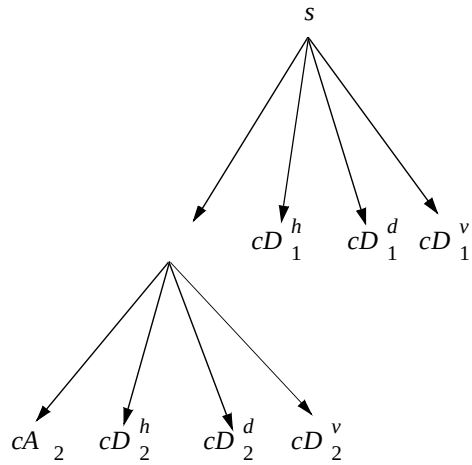


where

2 ↓ 1	Downsample columns: keep the even indexed columns
1 ↓ 2	Downsample rows: keep the even indexed rows
rows X	Convolve with filter X the rows of the entry
columns X	Convolve with filter X the columns of the entry

Initialization $cA_0 = s$ for the decomposition initialization

So, for J=2, the two-dimensional wavelet tree has the form:



See Also

`dwt2`, `waveinfo`, `wfilters`, `wmaxlev`

References

I. Daubechies (1992), "Ten lectures on wavelets," CBMS-NSF conference series in applied mathematics. SIAM Ed.

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

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

wavedemo

Purpose Wavelet toolbox demos.

Syntax `wavedemo`

Description `wavedemo` brings up a GUI that allows you to choose between several Wavelet Toolbox demos.

Purpose

Wavelet and scaling functions.

Syntax

```
[phi,psi,XVAL] = wavefun('wname',iter)
[phi1,psi1,phi2,psi2,XVAL] = wavefun('wname',iter)
[psi,XVAL] = wavefun('wname',iter)
wavefun ('wname',a,b)
```

Description

The function `wavefun` returns approximations of the wavelet function '*wname*' and the associated scaling function, if it exists. Positive integer `iter` determines the number of iterations computed, and thus the refinement of the approximations.

For an orthogonal wavelet:
`[phi,psi,XVAL] = wavefun('wname',iter)` returns the scaling and wavelet functions on the 2^{iter} points grid `XVAL`.

For a biorthogonal wavelet:
`[phi1,psi1,phi2,psi2,XVAL] = wavefun('wname',iter)` returns the scaling and wavelet functions both for decomposition `(phi1,psi1)` and for reconstruction `(phi2,psi2)`.

For a Meyer wavelet:
`[phi,psi,XVAL] = wavefun('wname',iter)`

For a Morlet or Mexican Hat wavelet:
`[psi,XVAL] = wavefun('wname',iter)`

`wavefun('wname',a,b)`, where `a` and `b` are positive integers, is equivalent to `wavefun('wname',max(a,b))`, and draws plots of the wavelet and scale approximations.

When `a` is set equal to the special value 0,

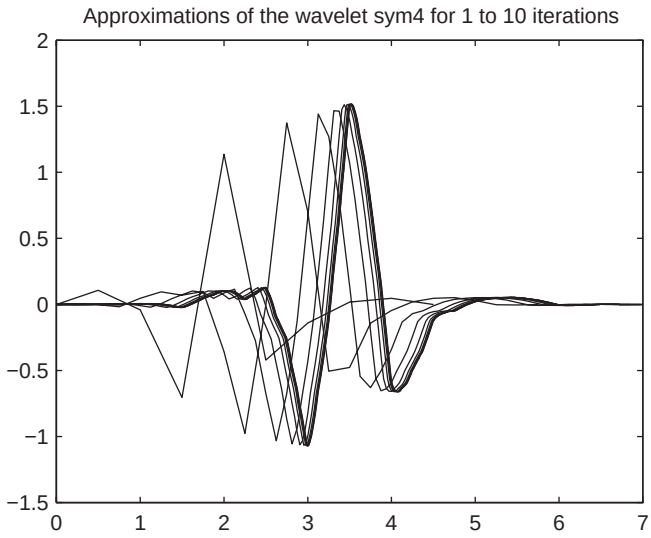
`wavefun('wname',0)` is equivalent to `wavefun('wname',8,b)`.

`wavefun('wname')` is equivalent to `wavefun('wname',8)`.

Examples

On the following graph, 10 piecewise linear approximations of the sym4 wavelet obtained after each iteration of the cascade algorithm are shown.

```
% Set number of iterations and wavelet name.
    iter = 10;
    wav = 'sym4';

% Compute approximations of the wavelet function using the
% cascade algorithm.
    for i = 1:iter
        [phi,psi,xval] = wavefun(wav,i);
        plot(xval,psi);
        hold on
    end
    title(['Approximations of the wavelet ',wav, ...
        ' for 1 to ',num2str(iter),' iterations']);
    hold off
```



Algorithm

For compactly supported wavelets defined by filters, in general no closed form analytic formula exists.

The algorithm used is the cascade algorithm. It uses the single-level inverse wavelet transform repeatedly.

Let us begin with the scaling function ϕ.

Since ϕ is also equal to $\phi_{0,0}$, (according to the notation used in Chapter 6), this function is characterized by the following coefficients in the orthogonal framework:

$\langle \phi, \phi_{0,n} \rangle = 1$ only if $n = 0$ and equal to 0 otherwise

$\langle \phi, \psi_{-j,k} \rangle = 0$ for positive j, and all k.

This expansion can be viewed as a wavelet decomposition structure. Detail coefficients are all zeros and approximation coefficients are all zeros except one equal to 1.

Then we use the reconstruction algorithm in order to approximate the function ϕ over a dyadic grid, according to the following result:

For any dyadic rational of the form $x = n2^{-j}$ in which the function is continuous and where j is sufficiently large, we have pointwise convergence and:

$$\left| \phi(x) - 2^{\frac{j}{2}} \langle \phi, \phi_{-j,\, n2^{j-J}} \rangle \right| \leq C.2^{-j\alpha}$$

where C is a constant, and α is a positive constant depending on the wavelet regularity.

Then using a good approximation of ϕ on dyadic rationals, we can use piecewise constant or piecewise linear interpolations η on dyadic intervals, for which uniform convergence occurs with similar exponential rate:

$$\| \phi - \eta \|_{\infty} \leq C.2^{-j\alpha}$$

So using a J-steps reconstruction scheme, we obtain an approximation that converges exponentially towards ϕ when J goes to infinity.

Approximations are computed over a grid of dyadic rationals covering the support of the function to be approximated.

Since a scaled version of the wavelet function ψ can also be expanded on the $(\phi_{-1,n})_n$, the same scheme can be used, after a single-level reconstruction starting with the appropriate wavelet decomposition structure. Approximation coefficients are all zeros and detail coefficients are all zeros except one equal to 1.

For biorthogonal wavelets, the same ideas can be applied on each of the two multiresolution schemes in duality.

Note: This algorithm may diverge if the function to be approximated is not continuous on dyadic rationals.

See Also

`intwave`, `waveinfo`, `wfilters`

References

I. Daubechies, “Ten lectures on wavelets,” CBMS, SIAM, 1992, p. 202-213.

G. Strang, T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press.

Purpose

Information on wavelets.

Syntax

```
waveinfo
waveinfo('wname')
```

Description

`waveinfo` gives information on all wavelets.

`waveinfo('wname')` gives information on the wavelet family whose short name is specified by the string '*wname*'. Available family short names are:

`'haar'`	:	Haar wavelet.
`'db'`	:	Daubechies wavelets.
`'sym'`	:	Symlets.
`'coif'`	:	Coiflets.
`'bior'`	:	Biorthogonal wavelets.
`'meyr'`	:	Meyer wavelet.
`'mexh'`	:	Mexican hat wavelet.
`'morl'`	:	Morlet wavelet.

or user-defined short names for their own wavelet families (see `wavemngr`).

`waveinfo('wsys')` gives information on wavelet packets.

Examples

```
waveinfo('db')

DBINFO Information on Daubechies wavelets.
Daubechies Wavelets
General characteristics: Compactly supported
wavelets with extremal phase and highest
number of vanishing moments for a given
support width. Associated scaling filters are
minimum-phase filters.

Family          Daubechies
Short name      db
Order N         N strictly positive integer
Examples        db1 or haar, db4, db15

Orthogonal      yes
Biorthogonal    yes
Compact support yes
DWT             possible
CWT             possible

Support width   2N-1
Filters length  2N
Regularity      about 0.2 N for large N
Symmetry        far from
Number of vanishing moments for psi     N

Reference:   I. Daubechies,
Ten lectures on wavelets CBMS, SIAM, 61, 1994, 194-202.
```

See Also

wavemngr

Purpose

Start graphical user interface tools.

Syntax

```
wavemenu
```

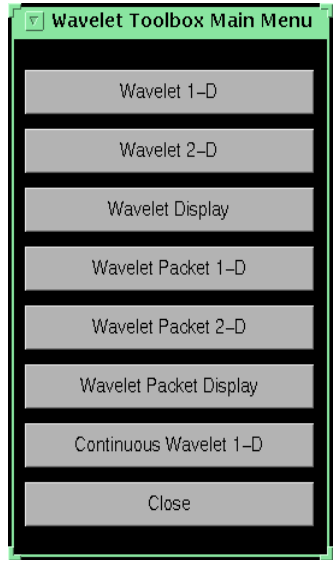
Description

`wavemenu` brings up a menu for accessing the various graphical tools provided in the Wavelet Toolbox. For instructions on using these tools see:

Continuous Wavelet 1-D	*Chapter 2*
Wavelet 1-D and **Wavelet 2-D**	*Chapter 2*
Wavelet Packet 1-D and **Wavelet Packet 2-D**	*Chapter 5*
Wavelet Display and **Wavelet Packet Display**	*Chapter 1*

Examples

```
wavemenu
```



wavemngr

Purpose

Wavelet manager.

Syntax

```
wavemngr('create')
wavemngr('add',FN,FSN,WT,NUMS,FILE)
wavemngr('add',FN,FSN,WT,NUMS,FILE,B)
wavemngr('del',N)
wavemngr('restore')
wavemngr('restore',IN2)
OUT1 = wavemngr('read')
OUT1 = wavemngr('read',IN2)
OUT1 = wavemngr('read_asc')
```

Description

`wavemngr` is a type of wavelets manager. It allows you to create, add, delete, restore, or read wavelets.

`wavemngr('create')` creates the `wavelets.inf` MAT-file using the `wavelets.asc` ASCII-file.

`wavemngr('add',FN,FSN,WT,NUMS,FILE)` or `wavemngr('add',FN,FSN,WT,NUMS,FILE,B)`, adds a new wavelet family to the toolbox.

`FN` = Family Name (string)

`FSN` = Family Short Name (string of length less than four characters)

`WT` = Wavelet type

- `WT` = 1, orthogonal wavelets
- `WT` = 2, biorthogonal wavelets
- `WT` = 3, wavelet with scaling function
- `WT` = 4, wavelet without scaling function

`NUMS` = String of numbers

`FILE` = MAT-file or M-file name (string). See the example for usage.

`B` = `[lb ub]` lower and upper bounds of effective support for wavelets of type = 3 or 4.

This option is fully documented in Chapter Chapter 7, “Adding Your Own Wavelets .”

`wavemngr('del',N)`, deletes a wavelet family. `FSN` = Family Short Name or Wavelet Name (in the family).

`wavemngr('restore')` or `wavemngr('restore',IN2)`, restores previous or initial wavelets. If `nargin = 1`, the previous `wavelets.asc` file is restored; otherwise the initial `wavelets.asc` file is restored. Here `IN2` is a dummy argument.

`OUT1 = wavemngr('read')` `OUT1` gives all wavelets families.

`OUT1 = wavemngr('read',IN2)` returns all wavelets, `IN2` is a dummy argument.

`OUT1 = wavemngr('read_asc')` reads `wavelets.asc` ASCII-file and `OUT1` gives all wavelets information.

Examples

```
% List initial wavelets families.
    wavemngr('read')

ans =
==================================
Haar        haar
Daubechies  db
BiorSplines bior
Coiflets    coif
Symlets     sym
Morlet      morl
Mexican_hat mexh
Meyer       meyr
==================================
% List all wavelets.
    wavemngr('read',1)

ans =
==================================
Haar        haar
==================================
Daubechies  db
------------------------------
db1     db2     db3     db4
db5     db6     db7     db8
db9     db10    dbxx
==================================
BiorSplines bior
------------------------------
bior1.1 bior1.3 bior1.5 bior2.2
bior2.4 bior2.6 bior2.8 bior3.1
bior3.3 bior3.5 bior3.7 bior3.9
bior4.4 bior5.5 bior6.8
==================================
Coiflets    coif
------------------------------
coif1   coif2   coif3   coif4
coif5
==================================
```

```
Symlets     sym
-----------------------------
sym2    sym3    sym4    sym5
sym6    sym7    sym8
==================================
Morlet      morl
==================================
Mexican_hat mexh
==================================
Meyer       meyr
==================================
```

In the following example, new compactly supported orthogonal wavelets are added to the toolbox. These wavelets, which are a slight generalization of the Daubechies wavelets, are based on the use of Bernstein polynomials and are due to Kateb and Lemarié in an unpublished work.

Note: The M-files used in this example can be found in the `wavedemo` directory.

```
% Add new family of orthogonal wavelets.
% You must define:
%
%     Family Name:          Lemarie
%     Family Short Name:    lem
%     Type of wavelet:      1 (orth)
%     Wavelets numbers:     1 2 3 4 5
%     File driver:          lemwavf
%
%     The function lemwavf.m must be as follow:
%     function w = lemwavf(wname)
%     where the input argument wname is a string:
%     wname = 'lem1' or 'lem2' ... i.e.,
%     wname = sh.name + number
%     and w the corresponding scaling filter.
% Ten addition is obtained using:
```

```
wavemngr('add','Lemarie','lem',1,'1 2 3 4 5','lemwavf');

% The ascii file 'wavelets.asc' is saved as
% 'wavelets.prv', then it is modified and
% the MAT file 'wavelets.inf' is generated.

% List wavelets families.
    wavemngr('read')

ans =
===================================
Haar        haar
Daubechies  db
BiorSplines bior
Coiflets    coif
Symlets     sym
Morlet      morl
Mexican_hat mexh
Meyer       meyr
Lemarie     lem
===================================
% Remove the added family.
    wavemngr('del','Lemarie');

% List wavelets families.
    wavemngr('read')

ans =
===================================
Haar        haar
Daubechies  db
BiorSplines bior
Coiflets    coif
Symlets     sym
Morlet      morl
Mexican_hat mexh
Meyer       meyr
===================================
```

```
% Restore the previous ascii file
% 'wavelets.prv', then build
% the MAT-file 'wavelets.inf'.
    wavemngr('restore');

% List restored wavelets.
    wavemngr('read',1)

ans =
===================================
Haar        haar
===================================
Daubechies  db
-----------------------------
db1     db2     db3     db4
db5     db6     db7     db8
db9     db10    dbxx
===================================
BiorSplines bior
-----------------------------
bior1.1 bior1.3 bior1.5 bior2.2
bior2.4 bior2.6 bior2.8 bior3.1
bior3.3 bior3.5 bior3.7 bior3.9
bior4.4 bior5.5 bior6.8
===================================
Coiflets    coif
-----------------------------
coif1   coif2   coif3   coif4   coif5
===================================
Symlets     sym
-----------------------------
sym2    sym3    sym4    sym5
sym6    sym7    sym8
===================================
Morlet      morl
===================================
Mexican_hat mexh
===================================
Meyer       meyr
===================================
```

```
Lemarie     lem
-----------------------------
lem1    lem2    lem3    lem4    lem5
==================================
% Restore initial wavelets.
%
% Restore the initial ascii file
% 'wavelets.ini' and initial
% MAT-file 'wavelets.bin'.
    wavemngr('restore',0);

% List wavelets families.
    wavemngr('read')

ans =
==================================
Haar        haar
Daubechies  db
BiorSplines bior
Coiflets    coif
Symlets     sym
Morlet      morl
Mexican_hat mexh
Meyer       meyr
==================================
% Add new family of orthogonal wavelets.
    wavemngr('add','Lemarie','lem',1,'1 2 3','lemwavf');

% All command line capabilities are available for
% the new wavelets.
%
% Example 1: compute the four associated filters.
    [Lo_D,Hi_D,Lo_R,Hi_R] = wfilters('lem3');

% Example 2: compute scale and wavelet functions.
    [phi,psi,xval] = wavefun('lem3');
```

```
% Add a new family of orthogonal wavelets: special form
% for the GUI mode.
%
% The M-file lemwavf allows you to compute the filter for
% any order. If you want to get a popup of the form
% 1 2 3 **, associated with the family, then wavelets are
% appended for GUI mode using:

    wavemngr('restore',0);
    wavemngr('add','Lemarie','lem',1,'1 2 3 **','lemwavf');

% After this sequence, all GUI capabilities are available for
% the new wavelets.
% Note that the last command allows a short cut in the
% order definition only if possible orders are integers.
```

Caution: `wavemngr` works on the current directory. If you add a new wavelet family, it is available in this directory only. Refer to Chapter 7, “Adding Your Own Wavelets .”

Limitations

`wavemngr` allows you to add a new wavelet. You must verify that it is truly a wavelet. No check is performed either about this point or about the type of the new wavelet.

waverec

Purpose

Multi-level 1-D wavelet reconstruction.

Syntax

```
X = waverec(C,L,'wname')
X = waverec(C,L,Lo_R,Hi_R)
```

Description

waverec performs a multi-level one-dimensional wavelet reconstruction using either a specific wavelet ('*wname*', see wfilters) or specific reconstruction filters (Lo_R and Hi_R). waverec is the inverse function of wavedec in the sense that the abstract statement waverec(wavedec(X,N,'*wname*'),'*wname*') returns X.

X = waverec(C,L,'*wname*') reconstructs the signal X based on the multi-level wavelet decomposition structure [C,L] and wavelet '*wname*'. (For information about the decomposition structure, see wavedec.)

X = waverec(C,L,Lo_R,Hi_R) reconstructs the signal X as above, using the reconstruction filters you specify.

Remarks

Note that X = waverec(C,L,'*wname*') is equivalent to X = appcoef(C,L,'*wname*',0).

Examples

```
% Load original one-dimensional signal.
    load leleccum; s = leleccum(1:3920); ls = length(s);

% Perform decomposition of signal at level 3 using db5.
    [c,l] = wavedec(s,3,'db5');

% Reconstruct s from the wavelet decomposition structure [c,l].
    a0 = waverec(c,l,'db5');

% Check for perfect reconstruction.
    err = norm(s-a0)
err =
    3.2079e-09
```

See Also

appcoef, idwt, wavedec

Purpose

Multi-level 2-D wavelet reconstruction.

Syntax

```
X = waverec2(C,S,'wname')
X = waverec2(C,S,Lo_R,Hi_R)
```

Description

`waverec2` is a two-dimensional wavelet analysis function.

`X = waverec2(C,S,'wname')` performs a multi-level wavelet reconstruction of two-dimensional signal `X` based on the wavelet decomposition structure `[C,S]` (for detailed storage information, see `wavedec2`). '*wname*' is a string containing the name of wavelet (see `wfilters`).

Instead of giving the wavelet name, you can give the filters. For `X = waverec2(C,S,Lo_R,Hi_R)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

`waverec2` is the inverse function of `wavedec2` in the sense that the abstract statement `waverec2(wavedec2(X,N,'wname'),'wname')` gets back to `X`.

Remarks

Note that `X = waverec2(C,S,'wname')` is equivalent to `X = appcoef2(C,S,'wname',0)`.

Examples

```
% Load original image.
    load woman;
    % X contains the loaded image.

% Perform decomposition at level 2
% of X using sym4.
    [c,s] = wavedec2(X,2,'sym4');

% Reconstruct X from the wavelet
% decomposition structure [c,s].
    a0 = waverec2(c,s,'sym4');

% Check for perfect reconstruction.
    max(max(X-a0))

ans =
    1.9463e-10
```

See Also

`appcoef2`, `idwt2`, `wavedec2`

wcodemat

Purpose

Extended pseudocolor matrix scaling.

Syntax

```
Y = wcodemat(X,NB,OPT,ABSOL)
Y = wcodemat(X,NB,OPT)
Y = wcodemat(X,NB)
Y = wcodemat(X)
```

Description

`wcodemat` is a general utility.

`Y = wcodemat(X,NB,OPT,ABSOL)` returns a coded version of input matrix `X` if `ABSOL = 0`, or `ABS(X)` if `ABSOL` is nonzero, using the first `NB` integers. Coding can be done rowwise (`OPT = 'row'`), columnwise (`OPT = 'col'`) or globally (`OPT = 'mat'`). Coding uses a regular grid between the minimum and the maximum values of each row (column or matrix, respectively).

`Y = wcodemat(X,NB,OPT)` is equivalent to `Y = wcodemat(X,NB,OPT,1)`.

`Y = wcodemat(X,NB)` is equivalent to `Y = wcodemat(X,NB, 'mat',1)`.

`Y = wcodemat(X)` is equivalent to `Y = wcodemat(X,16,'mat',1)`.

Purpose

Find common elements.

Syntax

```
[XI,YI] = wcommon(X,Y)
```

Description

`wcommon` is a general utility.

For two vectors `X` and `Y` with integer components, `[XI,YI] = wcommon(X,Y)` returns two vectors with 0 and 1 components such that:

`XI(k) = 1` if `X(k)` belongs to `Y`; otherwise `XI(k) = 0` and
`YI(j) = 1` if `Y(j)` belongs to `X`; otherwise `YI(j) = 0`.

Examples

```
% Define two vectors.
    x = [ 10 20 30 40 50 ];
    y = [ 60 50 70 30 20 12 31 ];

% Find common elements.
    [xi,yi] = wcommon(x,y)

xi =
    0 1 1 0 1

yi =
    0 1 0 1 1 0 0

% List common elements.
    comelem = x(find(xi))

comelem =
    20 30 50
```

wdatamgr

Purpose

Manager for data structure.

Syntax

```
[OUT1,OUT2] = wdatamgr(O,D,IN3,IN4,IN5)
```

Description

`wdatamgr` is a tree management utility.

`[OUT1,OUT2] = wdatamgr(O,D,IN3,IN4,IN5)` where `D` is the data structure and `O` is a string option. The possible options are:

`'write_cfs'`: writes coefficients for a terminal node

```
data = wdatamgr('write_cfs',data,tree,node,coefs);
```

`'read_cfs'`: reads coefficients for a terminal node

```
coefs = wdatamgr('read_cfs',data,tree,node);
```

`'read_ent'`: reads the entropy vector

```
ent = wdatamgr('read_ent',data,nodes);
```

`'read_ento'`: reads the optimal entropy vector

```
ento = wdatamgr('read_ento',data,nodes);
```

`'read_tp_ent'`: reads the type and the parameter for entropy

```
[type_ent,param] = wdatamgr('read_tp_ent',data);
```

`'read_wave'`: reads the name of the wavelet

```
wave = wdatamgr('read_wave',data);
```

Examples

```
% Load signal.
    load noisdopp; x = noisdopp;

% Decompose x at depth 3 with db1 wavelet packets
% using Shannon entropy.
    [t,d] = wpdec(x,3,'db1','shannon');

% Read entropy name.
    ent_name = wdatamgr('read_tp_ent',d)
ent_name =
shannon

% Read wavelet name.
    wav_name = wdatamgr('read_wave',d)
wav_name =
db1
```

```
% Read packet (3,2) coefficients.
    cfs = wdatamgr('read_cfs',d,t,[3 2]);

% Read packet (3,2) entropy and optimal entropy.
    ind_node = depo2ind(2,[3 2]);
    ent = wdatamgr('read_ent',d,ind_node)
ent =
-318.4298

% Optimal entropy is NaN because no optimization has been done.
    ento = wdatamgr('read_ento',d,ind_node)
ento =
NaN

% Modify packet (3,2) coefficients.
    ncfs = cos(cfs);     % or any other modification !

% Update packet (3,2) coefficients.
    d = wdatamgr('write_cfs',d,t,[3 2],ncfs);

% Update nodes entropy.
    d = entrupd(t,d,'shannon');
    nent = wdatamgr('read_ent',d,ind_node)
nent =
22.2830
```

See Also wpdec, wpdec2, wtreemgr

Purpose

Automatic 1-D de-noising using wavelets.

Syntax

```
[XD,CXD,LXD] = wden(X,TPTR,SORH,SCAL,N,'wname')
[XD,CXD,LXD] = wden(C,L,TPTR,SORH,SCAL,N,'wname')
```

Description

`wden` is a one-dimensional de-noising oriented function.

`wden` performs an automatic de-noising process of a one-dimensional signal using wavelets.

`[XD,CXD,LXD] = wden(X,TPTR,SORH,SCAL,N,'wname')` returns a de-noised version `XD` of input signal `X` obtained by thresholding the wavelet coefficients.

Additional output arguments `[CXD,LXD]` are the wavelet decomposition structure (see `wavedec`) of the de-noised signal `XD`.

`TPTR` string contains threshold selection rules:

`'rigrsure'` use the principle of Stein's Unbiased Risk.

`'heursure'` is an heuristic variant of the first option.

`'sqtwolog'` for universal threshold $\sqrt{2\log(.)}$.

`'minimaxi'` for minimax thresholding (see `thselect` for more details).

`SORH` (`'s'` or `'h'`) is for soft or hard thresholding (see `wthresh` for more details).

`SCAL` defines multiplicative threshold rescaling:

`'one'` for no rescaling.

`'sln'` for rescaling using a single estimation of level noise based on first level coefficients.

`'mln'` for rescaling done using level-dependent estimation of level noise.

Wavelet decomposition is performed at level `N` and '*wname*' is a string containing the name of the desired orthogonal wavelet (see `wmaxlev` and `wfilters`).

`[XD,CXD,LXD] = wden(C,L,TPTR,SORH,SCAL,N,'wname')` returns the same output arguments, using the same options as above, but obtained directly from the input wavelet decomposition structure `[C,L]` of the signal to be de-noised, at level `N` and using '*wname*' orthogonal wavelet.

The underlying model for the noisy signal is basically of the following form:

$$s(n) = f(n) + \sigma e(n)$$

where time n is equally spaced.

In the simplest model, suppose that $e(n)$ is a Gaussian white noise $N(0,1)$ and the noise level σ a is supposed to be equal to 1.

The de-noising objective is to suppress the noise part of the signal s and to recover f.

The de-noising procedure proceeds in three steps:

1. Decomposition. Choose a wavelet, and choose a level `N`. Compute the wavelet decomposition of the signal s at level `N`.
2. Detail coefficients thresholding. For each level from 1 to `N`, select a threshold and apply soft thresholding to the detail coefficients.
3. Reconstruction. Compute wavelet reconstruction based on the original approximation coefficients of level `N` and the modified detail coefficients of levels from 1 to `N`.

More details about threshold selection rules can be found in Chapter 6 and in the help for `thselect`. Let us point out that:

- The detail coefficients vector is the superposition of the coefficients of f and the coefficients of e, and that the decomposition of e leads to detail coefficients that are standard Gaussian white noises.
- Minimax and SURE threshold selection rules are more conservative and are more convenient when small details of function f lie in the noise range. The two other rules remove the noise more efficiently. The option `'heursure'` is a compromise.

In practice the basic model cannot be used directly. This section examines the options available, in order to deal with model deviations. The remaining parameter `scal` has to be specified. It corresponds to threshold rescaling methods.

- Option `scal = 'one'` corresponds to the basic model.

- In general you can ignore the noise level that must be estimated. The detail coefficients CD_1 (the finest scale) are essentially noise coefficients with standard deviation equal to σ. The median absolute deviation of the coefficients is a robust estimate of σ. The use of a robust estimate is crucial for two reasons. The first is that if level 1 coefficients contain *f* details, these details are concentrated in few coefficients. The second reason is to avoid signal end effects, which are pure artifacts due to computations on the edges.

  Option `scal = 'sln'` handles threshold rescaling using a single estimation of level noise based on the first level coefficients.

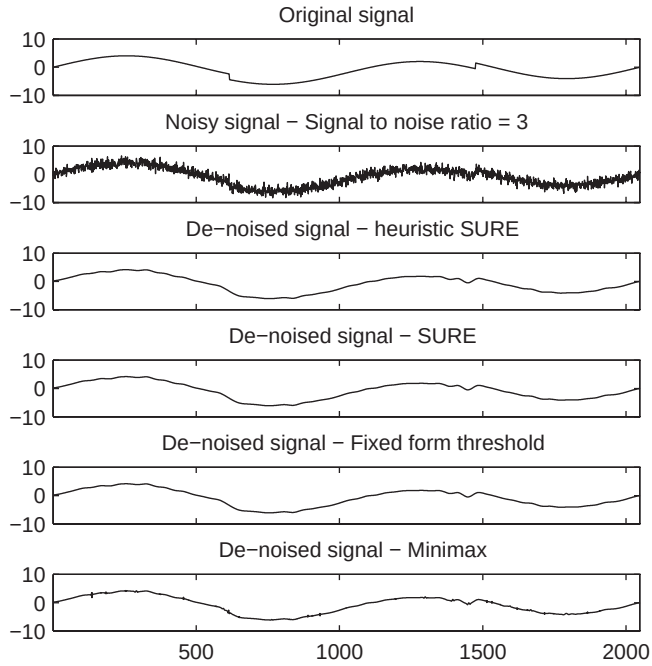
- When you suspect a nonwhite noise *e*, thresholds must be rescaled by a level dependent estimation of the level noise. The same kind of strategy is used by estimating σ_{lev} level by level. This estimation is implemented in M-file `wnoisest`, which handles the wavelet decomposition structure of the original signal *s* directly.

  Option `scal = 'mln'` handles threshold rescaling using a level-dependent estimation of the level noise.

Examples

```
% Set signal to noise ratio and set rand seed.
    snr = 3; init = 2055615866;

% Generate original signal and a noisy version adding
% a standard Gaussian white noise.
    [xref,x] = wnoise(3,11,snr,init);

% De-noise noisy signal using soft heuristic SURE thresholding
% and scaled noise option, on detail coefficients obtained
% from the decomposition of x, at level 5 by sym8 wavelet.
    lev = 5;
    xd = wden(x,'heursure','s','one',lev,'sym8');
```

```
% Plot signals.
    subplot(611), plot(xref), axis([1 2048 -10 10]);
    title('Original signal');
    subplot(612), plot(x), axis([1 2048 -10 10]);
    title(['Noisy signal - Signal to noise ratio = ',...
    num2str(fix(snr))]);
    subplot(613), plot(xd), axis([1 2048 -10 10]);
    title('De-noised signal - heuristic SURE');

% De-noise noisy signal using soft SURE thresholding
    xd = wden(x,'heursure','s','one',lev,'sym8');

% Plot signal.
    subplot(614), plot(xd), axis([1 2048 -10 10]);
    title('De-noised signal - SURE');

% De-noise noisy signal using fixed form threshold with
% a single level estimation of noise standard deviation.
    xd = wden(x,'sqtwolog','s','sln',lev,'sym8');

% Plot signal.
    subplot(615), plot(xd), axis([1 2048 -10 10]);
    title('De-noised signal - Fixed form threshold');

% De-noise noisy signal using minimax threshold with
% a multiple level estimation of noise standard deviation.
    xd = wden(x,'minimaxi','s','sln',lev,'sym8');

% Plot signal.
    subplot(616), plot(xd), axis([1 2048 -10 10]);
    title('De-noised signal - Minimax');

% If many trials are necessary, it is better to perform
% decomposition once and threshold it many times:
    % decomposition.
    [c,l] = wavedec(x,lev,'sym8');
    % threshold the decomposition structure [c,l].
   xd = wden(c,l,'minimaxi','s','sln',lev,'sym8');
```



See Also

`thselect`, `wavedec`, `wdencmp`, `wfilters`, `wthresh`

References

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

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

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

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

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

wdencmp

Purpose

De-noising or compression using wavelets.

Syntax

```
[XC,CXC,LXC,PERF0,PERFL2] =
   wdencmp('gbl',X,'wname',N,THR,SORH,KEEPAPP)
[XC,CXC,LXC,PERF0,PERFL2] = wdencmp('lvd',X,'wname',N,THR,SORH)
[XC,CXC,LXC,PERF0,PERFL2] = wdencmp('lvd',C,L,'wname',N,THR,SORH)
```

Description

`wdencmp` is a one- or two-dimensional de-noising and compression oriented function.

`wdencmp` performs a de-noising or compression process of a signal or an image, using wavelets.

`[XC,CXC,LXC,PERF0,PERFL2] = wdencmp('gbl',X,'wname',N,THR,SORH, KEEPAPP)` returns a de-noised or compressed version `XC` of input signal `X` (one- or two-dimensional) obtained by wavelet coefficients thresholding using global positive threshold `THR`.

Additional output arguments `[CXC,LXC]` are the wavelet decomposition structure of `XC` (see `wavedec` or `wavedec2`). `PERF0` and `PERFL2` are L^2-norm recovery and compression score in percentage.

`PERFL2` = 100 ∗ (vector-norm of `CXC` / vector-norm of `C`)^2 if `[C,L]` denotes the wavelet decomposition structure of `X`.

If `X` is a one-dimensional signal and '*wname*' an orthogonal wavelet, `PERFL2` is reduced to $\frac{100\|XC\|^2}{\|X\|^2}$.

Wavelet decomposition is performed at level `N` and '*wname*' is a string containing wavelet name (see `wmaxlev` and `wfilters`). `SORH` (`'s'` or `'h'`) is for soft or hard thresholding (see `wthresh` for more details). If `KEEPAPP` = 1, approximation coefficients cannot be thresholded, otherwise it is possible.

`wdencmp('gbl',C,L,'wname',N,THR,SORH,KEEPAPP)` has the same output arguments, using the same options as above, but obtained directly from the input wavelet decomposition structure `[C,L]` of the signal to be de-noised or compressed, at level `N` and using '*wname*' wavelet.

For the one-dimensional case and 'lvd' option:

```
[XC,CXC,LXC,PERF0,PERFL2] = wdencmp('lvd',X,'wname',N,THR,SORH)
```

or

```
[XC,CXC,LXC,PERF0,PERFL2] =
wdencmp('lvd',C,L,'wname',N,THR,SORH)
```

has the same output arguments, using the same options as above, but allowing level-dependent thresholds contained in vector THR (THR must be of length N). In addition, the approximation is kept. Note that, with respect to wden (automatic de-noising), wdencmp allows more flexibility and you can implement your own de-noising strategy.

For the two-dimensional case and 'lvd' option:

```
[XC,CXC,LXC,PERF0,PERFL2] = wdencmp('lvd',X,'wname',N,THR,SORH)
```

or

```
[XC,CXC,LXC,PERF0,PERFL2] =
wdencmp('lvd',C,L,'wname',N,THR,SORH)
```

THR must be a matrix 3 by N containing the level-dependent thresholds in the three orientations; horizontal, diagonal, and vertical.

Ideas for de-noising can be found in Chapter 2, “Using Wavelets ,” and in the Description section of the wden reference entry.

The compression features of a given wavelet basis are primarily linked to the relative scarceness of the wavelet domain representation for the signal. The notion behind compression is based on the concept that the regular signal component can be accurately approximated using a small number of approximation coefficients (at a suitably selected level) and some of the detail coefficients.

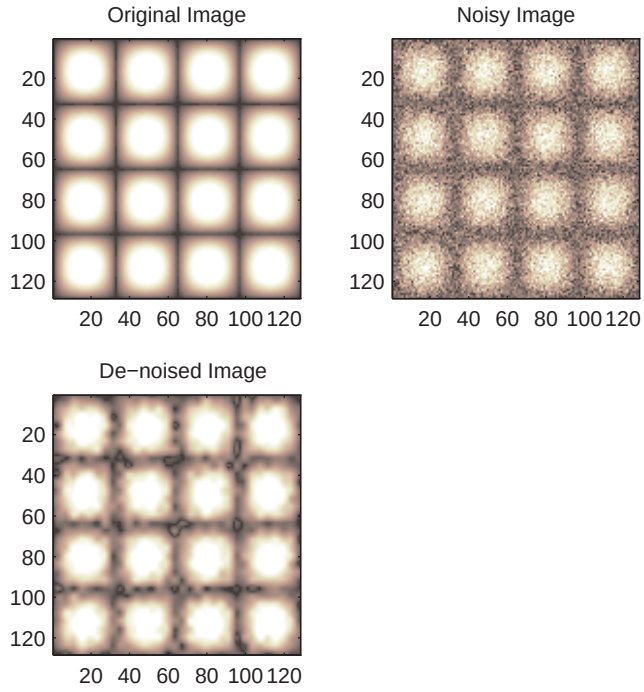
Like de-noising, the compression procedure contains three steps:

1. Decomposition.
2. Detail coefficient thresholding. For each level from 1 to N, a threshold is selected and hard thresholding is applied to the detail coefficients.
3. Reconstruction.

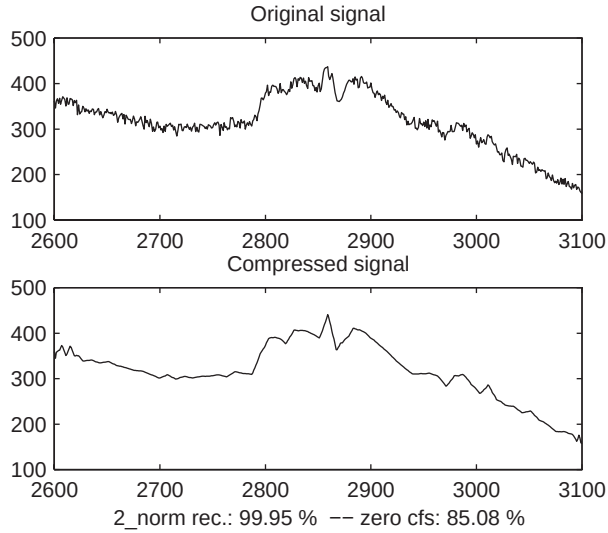
The difference with the de-noising procedure is found in step 2.

Examples

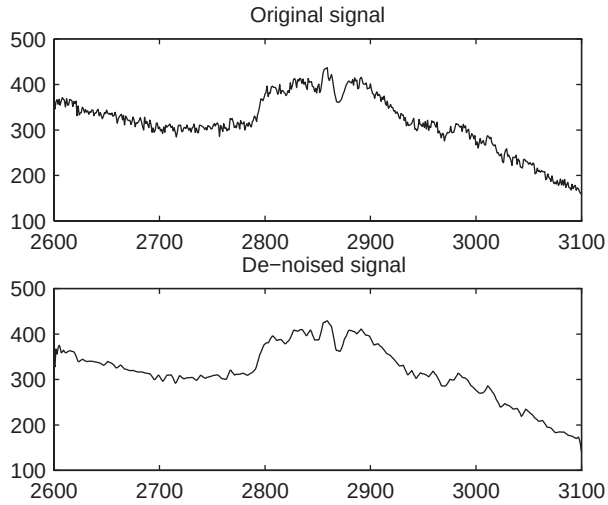
```
% Load original image.
    load sinsin
% X contains the loaded image.

% Generate noisy image.
    init=2055615866; randn('seed',init);
    x = X + 18*randn(size(X));

% Use wdencmp for image de-noising.
    % find default values (see ddencmp).
    [thr,sorh,keepapp] = ddencmp('den','wv',x);
    % de-noise image using global thresholding option.
    xd = wdencmp('gbl',x,'sym4',2,thr,sorh,keepapp);
```



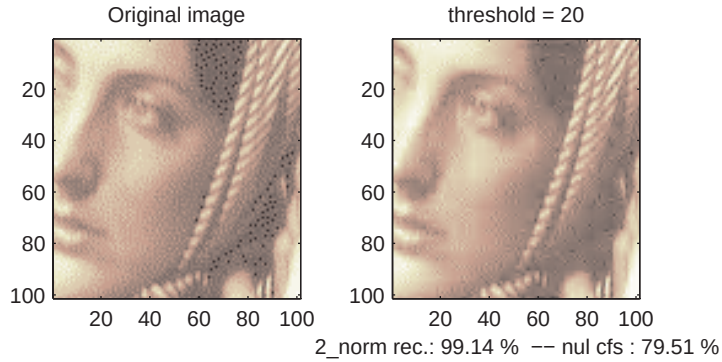
```
% Load electrical signal and select a part.
    load leleccum; indx = 2600:3100;
    x = leleccum(indx);

% Use wdencmp for signal compression.
    % compress using a fixed threshold.
    thr=35;
    [xd,cxd,lxd,perf0,perfl2] = ...
        wdencmp('gbl',x,'db3',3,thr,'h',1);
```




```
% Use wdencmp for signal de-noising.
    % Find default values (see ddencmp).
    [thr,sorh,keepapp] = ddencmp('den','wv',x);

% De-noise signal using global thresholding option.
    xd = wdencmp('gbl',x,'db3',2,thr,sorh,keepapp);
```




```
% Load original image.
    load woman;
    % X contains the loaded image.

    x=X(100:200,100:200);
    nbc = size(map,1);

% Use wdencmp for image compression.
    % Wavelet decomposition of x.
    n = 5; w = 'sym2';
    [c,l] = wavedec2(x,n,w);

% Wavelet coefficients thresholding.
thr=20;
[xd,cxd,lxd,perf0,perfl2] = ...
    wdencmp('gbl',c,l,w,n,thr,'h',1);
```




```
% In addition the first option allows level and orientation-
% dependent thresholds. In this case the approximation is kept.
% The level-dependent thresholds in the three orientations
% horizontal, diagonal and vertical are as follows:
    thr_h = [17 18];    % Horizontal thresholds.
    thr_d = [19 20];    % Diagonal thresholds.
    thr_v = [21 22];    % Vertical thresholds.

    thr = [thr_h ; thr_d ; thr_v]
thr =
    17 18
    19 20
    21 22
    [xd,cxd,lxd,perf0,perfl2] = ...
    wdencmp('lvd',x,'sym8',2,thr,'h');
```

See Also

ddencmp, wavedec, wavedec2, wden, wpdencmp, wthresh

References

R.A. DeVore, B. Jawerth, B.J. Lucier (1992), "Image compression through wavelet transform coding," *IEEE Trans. on Inf. Theory*, vol. 38, No 2, pp. 719-746.

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

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

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

D.L. Donoho, I.M. Johnstone, “Ideal de-noising in an orthonormal basis chosen from a library of bases,” C.R.A.S. Paris, t. 319, Ser. I, pp. 1317–1322.

D.L. Donoho (1995), “De-noising by soft-thresholding,” *IEEE Trans. on Inf. Theory*, 41, 3, pp. 613–627.

wentropy

Purpose

Entropy (wavelet packet).

Syntax

```
E = wentropy(X,T,P)
E = wentropy(X,T)
```

Description

`E = wentropy(X,T,P)` returns the entropy `E` of the vector or matrix input `X`. In both cases, output `E` is a real number. `T` is a string containing the type of entropy:

```
T = 'shannon', 'threshold', 'norm', 'log energy', 'sure', 'user'
```

`P` is an optional parameter depending on `T` value:

If `T = 'shannon'` or `'log energy'`, `P` is not used.

If `T = 'threshold'` or `'sure'`, `P` is the threshold and must be a positive number.

If `T = 'norm'`, `P` is the power and must be such that `1 <= P < 2`.

If `T = 'user'`, `P` is a string containing the M-file name of your own entropy function, with a single input `X`.

`E = wentropy(X,T)` is equivalent to `E = wentropy(X,T,0)`.

Functionals verifying an additive-type property are well suited for efficient searching of binary-tree structures and the fundamental splitting property of the wavelet packets decomposition. Classical entropy-based criteria match these conditions and describe information-related properties for an accurate representation of a given signal. Entropy is a common concept in many fields, mainly in signal processing. The following example lists different entropy criteria, many others are available and can be easily integrated. In the following expressions s is the signal and $(s_i)_i$ the coefficients of s in an orthonormal basis.

The entropy E must be an additive cost function such that $E(0) = 0$ and $E(s) = \sum_i E(s_i)$.

- The (non-normalized) Shannon entropy.

  $E1(s_i) = s_i^2 \log(s_i^2)$ so $E1(s) = -\sum_i s_i^2 \log(s_i^2)$

  with the convention $0 log(0) = 0$.
- The concentration in l^p norm with $1 \le p < 2$.
- $E2(s_i) = |s_i|^p$ so $E2(s) = \sum_i |s_i|^p = \|s\|_p^p$
- The "log energy" entropy.
- $E3(s_i) = \log(s_i^2)$ so $E3(s) = \sum_i \log(s_i^2)$
- with the convention $log(0) = 0$.
- The threshold entropy.
- $E4(s_i) = 1$ if $|s_i| > p$ and 0 elsewhere so $E4(s) = \#\{i$ such that $|s_i| > p\}$ is the number of time instants when the signal is greater than a threshold p.
- The "SURE" entropy.

  E5(s) = n-#{i such that $|s_i| \le p$ } + $\sum_i min(s_i^2, p^2)$

  See the section entitled "Using wavelet packets for compression and de-noising" in Chapter 6 for more information.

Examples

```
% Generate initial signal.
    x = randn(1,200);

% Compute Shannon entropy of x.
    e = wentropy(x,'shannon')

e =
    -142.7607

% Compute log energy entropy of x.
    e = wentropy(x,'log energy')
e =
    -281.8975

% Compute threshold entropy of x
% with threshold equal to 0.2.
    e = wentropy(x,'threshold',0.2)
e =
    162

% Compute Sure entropy of x
% with threshold equal to 3.
    e = wentropy(x,'sure',3)
e =
 -0.6575

% Compute norm entropy of x with power equal to 1.1.
    e = wentropy(x,'norm',1.1)
e =
 160.1583
```

```
% Compute user entropy of x with a user defined
% function: userent for example.
% this function must be an M-file, with first line
% of the following form:
%
%     function e = userent(x)
%
% where x is a vector and e is a real number.
% Then a new entropy is defined and can be used typing:
%
% e = wentropy(x,'user','userent')
```

References

R.R. Coifman, M.V. Wickerhauser, (1992), “Entropy-based Algorithms for best basis selection,” *IEEE Trans. on Inf. Theory*, vol. 38, 2, pp. 713–718.

D.L. Donoho, I.M. Johnstone, “Ideal de-noising in an orthonormal basis chosen from a library of bases,” C.R.A.S. Paris, t. 319, Ser. I, pp. 1317–1322.

Purpose

Wavelet filters.

Syntax

```
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters('wname')
[F1,F2] = wfilters('wname','type')
```

Description

`[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters('wname')` computes four filters associated with the orthogonal or biorthogonal wavelet named in the string '*wname*'.

The four output filters are:

- `Lo_D`, the decomposition low-pass filter
- `Hi_D`, the decomposition high-pass filter
- `Lo_R`, the reconstruction low-pass filter
- `Hi_R`, the reconstruction high-pass filter

Available orthogonal or biorthogonal wavelet names '*wname*' are:

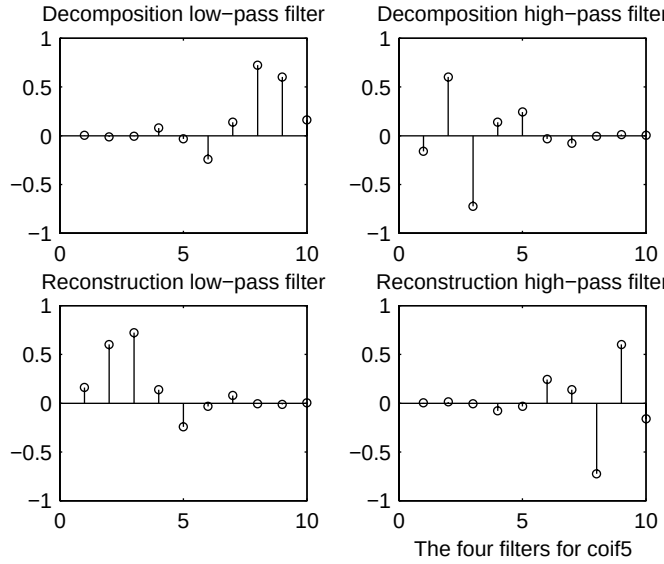
```
Daubechies    : 'db1' or 'haar', 'db2', ... ,'db10', ... ,'db50'
Coiflets      : 'coif1', ... , 'coif5'
Symlets       : 'sym2', ... , 'sym8'
Biorthogonal  : 'bior1.1', 'bior1.3', 'bior1.5'
                'bior2.2', 'bior2.4', 'bior2.6', 'bior2.8'
                'bior3.1', 'bior3.3', 'bior3.5', 'bior3.7'
                'bior3.9', 'bior4.4', 'bior5.5', 'bior6.8'
```

`[F1,F2] = wfilters('wname','type')` returns the following filters:

`Lo_D` and `Hi_D`	(Decomposition filters)	If '*type*' = `'d'`
`Lo_R` and `Hi_R`	(Reconstruction filters)	If '*type*' = `'r'`
`Lo_D` and `Lo_R`	(Low-pass filters)	If '*type*' = `'l'`
`Hi_D` and `Hi_R`	(High-pass filters)	If '*type*' = `'h'`

Examples

```
% Set wavelet name.
    wname = 'db5';
% Compute the four filters associated with wavelet name given
% by the input string wname.
    [Lo_D,Hi_D,Lo_R,Hi_R] = wfilters(wname);
    subplot(221); stem(Lo_D);
    title('Decomposition low-pass filter');
    subplot(222); stem(Hi_D);
    title('Decomposition high-pass filter');
    subplot(223); stem(Lo_R);
    title('Reconstruction low-pass filter');
    subplot(224); stem(Hi_R);
    title('Reconstruction high-pass filter');
    xlabel('The four filters for db5')
```



```
% When used with two input arguments, depending on second
% argument, outputs are one row or one column of the previous
% figure.
    % Decomposition filters (first row).
    [Lo_D,Hi_D] = wfilters(wname,'d');
    % Reconstruction filters (second row).
    [Lo_R,Hi_R] = wfilters(wname,'r');
    % Low-pass filters (first column).
    [Lo_D,Lo_R] = wfilters(wname,'l');
    % High-pass filters (second column).
    [Hi_D,Hi_R] = wfilters(wname,'h');
```

See Also

biorfilt, orthfilt, waveinfo

References

I. Daubechies (1992), “Ten lectures on wavelets,” CBMS-NSF conference series in applied mathematics. SIAM Ed.

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

wkeep

Purpose

Keep part of a vector or a matrix.

Syntax

```
Y = wkeep(X,L,O)
Y = wkeep(X,L)
```

Description

`wkeep` is a general utility.

For a vector, `Y = wkeep(X,L,O)` extracts the vector `Y` from the vector `X`. `L` is the length of result `Y`. If `O = 'c'` (`'l'`, `'r'` respectively), `Y` is the central (left, right respectively) part of `X`.

`Y = wkeep(X,L)` is equivalent to `Y = wkeep(X,L,'c')`.

For a matrix, `Y = wkeep(X,S)` extracts the central part of the matrix `X`. `S` is the size of `Y`.

Examples

```
% For a vector.
    x = 1:10;
    y = wkeep(x,6,'c')
y =
    3     4     5     6     7     8

    y = wkeep(x,6)
y =
    3     4     5     6     7     8

    y = wkeep(x,7,'c')
y =
    2     3     4     5     6     7     8

    y = wkeep(x,6,'l')
y =
    1     2     3     4     5     6

    y = wkeep(x,6,'r')
y =
    5     6     7     8     9     10
```

```
% For a matrix.
    x = magic(5)
x =
    17     24      1      8     15
    23      5      7     14     16
     4      6     13     20     22
    10     12     19     21      3
    11     18     25      2      9

    y = wkeep(x,[3 2])
y =
    5     7
    6    13
   12    19
```

wmaxlev

Purpose

Maximum wavelet decomposition level.

Syntax

```
L = wmaxlev(S,'wname')
```

Description

`wmaxlev` is a one- or two-dimensional wavelet or wavelet packets oriented function.

`wmaxlev` can help you avoid unreasonable maximum level values.
`L = wmaxlev(S,'wname')` returns the maximum level decomposition of signal or image of size `S` using the wavelet named in the string '*wname*' (see `wfilters`).

`wmaxlev` gives the maximum allowed level decomposition, but in general, a smaller value is taken.

Usual values are 5 for the one-dimensional case and 3 for the two-dimensional case.

Examples

```
% For a 1_D signal.
    s = 2^10;
    w = 'db1';

% Compute maximum level decomposition.
% The rule is the last level for which at least
% one coefficient is correct.
    l = wmaxlev(s,w)

l =
    10

% Change wavelet.
    w = 'db7';

% Compute maximum level decomposition.
    l = wmaxlev(s,w)


l =
    6
```

```
% For a 2_D signal.
    s = [2^9 2^7];
    w = 'db1';

% Compute maximum level decomposition.
    l = wmaxlev(s,w)

l =
    7

% which is the same as:
    l = wmaxlev(min(s),w)

l =
    7

% Change wavelet.
    w = 'db7';

% Compute maximum level decomposition.
    l = wmaxlev(s,w)

l =
    3
```

See Also wavedec, wavedec2, wpdec, wpdec2

wnoise

Purpose

Generate noisy wavelet test data.

Syntax

```
X = wnoise(NUM,N)
[X,XN] = wnoise(NUM,N,SNRAT)
[X,XN] = wnoise(NUM,N,SNRAT,INIT)
```

Description

`X = wnoise(NUM,N)` returns values of test function number `NUM`, on a 2^N sample of [0,1].

`[X,XN] = wnoise(NUM,N,SNRAT)` returns a test vector `X` as above, rescaled such that `std(x) = SNRAT`. The returned vector `XN` contains the same test vector corrupted by additive Gaussian white noise $N(0,1)$. `XN` has a signal-to-noise ratio of `SNRAT`.

`[X,XN] = wnoise(NUM,N,SNRAT,INIT)` returns previous vector `X` and `XN`, but the generator seed is set to `INIT` value.

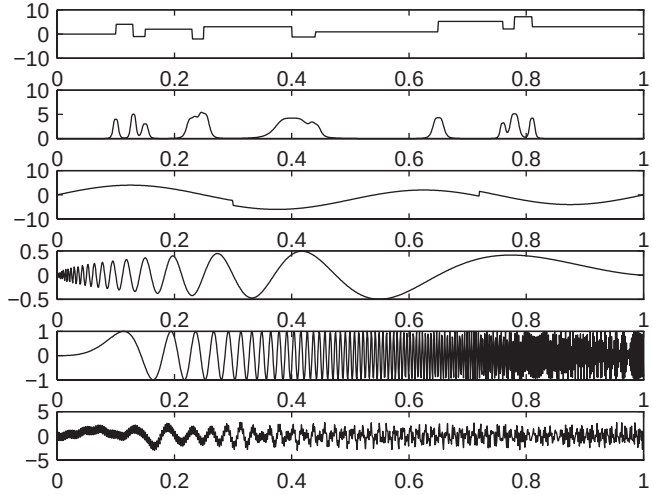
The six functions are due to Donoho and Johnstone (See Reference):

`NUM` = 1	Blocks
`NUM` = 2	Bumps
`NUM` = 3	Heavy sine
`NUM` = 4	Doppler
`NUM` = 5	Quadchirp
`NUM` = 6	Mishmash

Examples

```
% Generate 2^10 samples of 'Heavy sine' (item 3).
    x = wnoise(3,10);

% Generate 2^10 samples of 'Doppler' (item 4) and of
% noisy 'Doppler' with a signal-to-noise ratio of 7.
    [x,noisyx] = wnoise(4,10,7);
```

```
% To introduce your own rand seed, a fourth
% argument is allowed:
    init = 2055415866;
    [x,noisyx] = wnoise(4,10,7,init);

% Plot all the test functions.
    ind = linspace(0,1,2^10);
    for i = 1:6
        x = wnoise(i,10);
        subplot(6,1,i), plot(ind,x)
    end
```



See Also

wden

References

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

wnoisest

Purpose

Estimate noise of 1-D wavelet coefficients.

Syntax

```
STDC = wnoisest(C,L,S)
```

Description

`STDC = wnoisest(C,L,S)` returns estimates of detail coefficients standard deviation for levels contained in input vector `S`. `[C,L]` is the input wavelet decomposition structure (see `wavedec`).

The estimator used is Maximum Absolute Deviation / 0.6745, well suited for zero mean Gaussian white noise in de-noising one-dimensional models (see `thselect`).

Examples

```
% Generate Gaussian white noise.
    init = 2055415866; randn('seed',init);
    x = randn(1,1000);

% Decompose x at level 2 using db3 wavelet.
    [c,l] = wavedec(x,2,'db3');

% Estimate standard deviation of coefficients
% at each level 1 and 2.
% Since x is a Gaussian white noise with unit
% variance, estimates must be close to 1.
    wnoisest(c,l,1:2)

ans =
    1.0111 1.0763

% Now suppose that x contains 10 outliers.
    ind = 50:50:500;
    x(ind) = 100 * ones(size(ind));

% Decompose x at level 1 using db3 wavelet.
    [ca,cd] = dwt(x,'db3');
```

```
% Ordinary estimate of cd standard deviation
% overestimates noise level.
    std(cd)

ans =
    8.0206

% Robust estimate of cd standard deviation
% remains close to 1 the noise level.
    median(abs(cd))/0.6745

ans =
    1.0540
```

Limitations

This procedure is well suited for Gaussian white noise.

See Also

`thselect`, `wavedec`, `wden`

References

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

wp2wtree

Purpose

Extract wavelet tree from wavelet packet tree.

Syntax

```
[T,D] = wp2wtree(T,D)
```

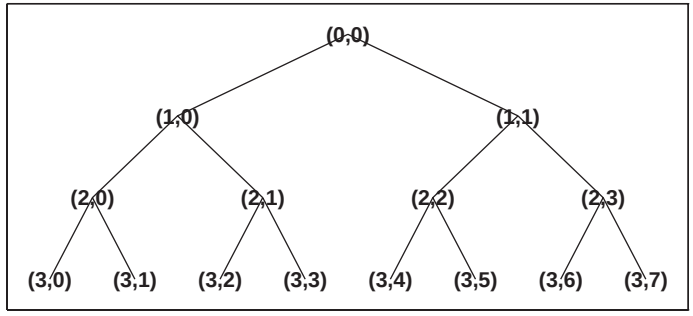
Description

`wp2wtree` is a one- or two-dimensional wavelet packet analysis function.

`[T,D] = wp2wtree(T,D)` computes the modified tree structure `T` and data structure `D` (see `maketree`), corresponding to the wavelet decomposition tree.

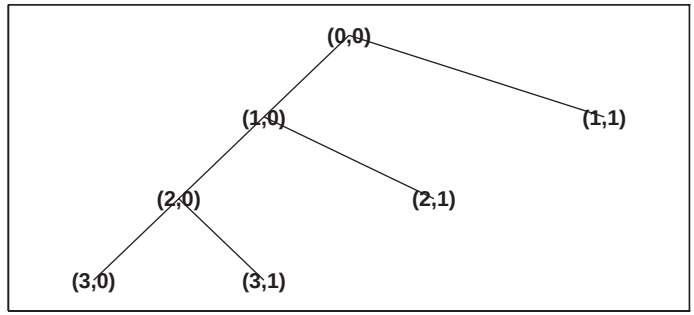
Examples

```
% Load signal.
    load noisdopp; x = noisdopp;

% Decompose x at depth 3 with db1 wavelet packets.
    [wpt,wpd] = wpdec(x,3,'db1');

% Plot wavelet packet tree structure wpt.
    plottree(wpt)
```




```
% Compute wavelet tree.
    [wt,wd] = wp2wtree(wpt,wpd);
```

```
% Plot wavelet tree structure wt.
    plottree(wt)
```



See Also maketree, wpdec, wpdec2

wpcoef

Purpose

Wavelet packet coefficients.

Syntax

```
X = wpcoef(S,D,N)
X = wpcoef(S,D)
```

Description

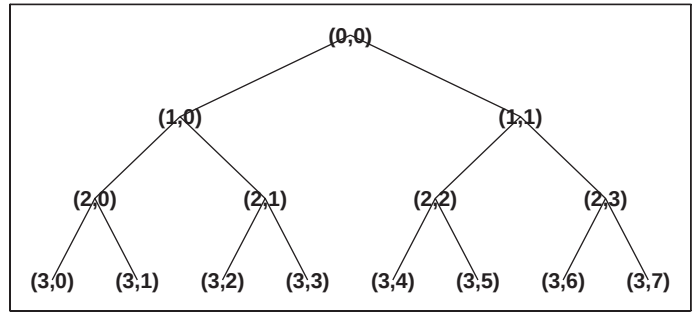
`wpcoef` is a one- or two-dimensional wavelet packet analysis function.

`X = wpcoef(S,D,N)` returns the coefficients associated with the node `N`. `S` is the tree structure and `D` the data structure (see `maketree`). If `N` doesn't exist, `X = [ ];`

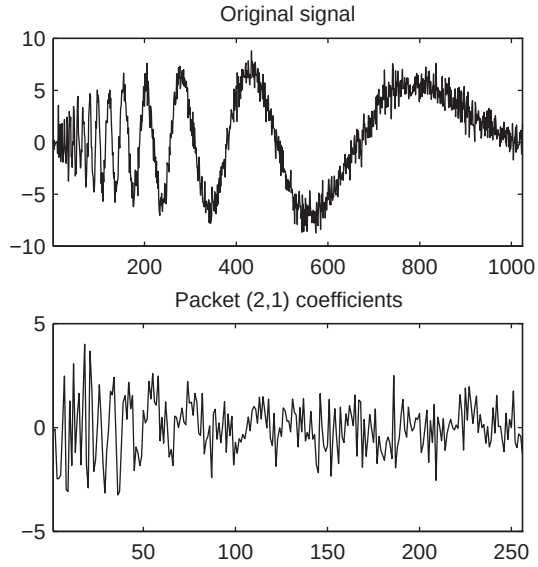
`X = wpcoef(S,D)` is equivalent to `X = wpcoef(S,D,0)`.

Examples

```
% Load signal.
    load noisdopp; x = noisdopp;

    figure(1); subplot(211);
    plot(x); title('Original signal');

% Decompose x at depth 3 with db1 wavelet packets
% using Shannon entropy.
    [t,d] = wpdec(x,3,'db1','shannon');

% Plot tree structure.
    plottree(t)
```



```
% Read packet (2,1) coefficients.
    cfs = wpcoef(t,d,[2 1]);

    figure(1); subplot(212);
    plot(cfs); title('Packet (2,1) coefficients');
```



See Also `maketree`, `wpdec`, `wpdec2`

wpcutree

Purpose

Cut wavelet packet tree.

Syntax

```
[T,D] = wpcutree(T,D,L)
[T,D,RN] = wpcutree(T,D,L)
```

Description

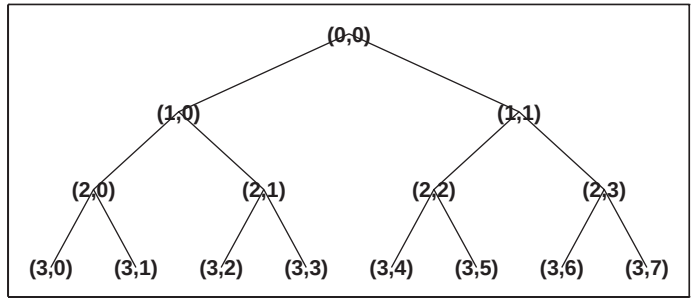
`wpcutree` is a one- or two-dimensional wavelet packet analysis function.

`[T,D] = wpcutree(T,D,L)` cuts the tree `T` at level `L` and computes the corresponding data structure `D` (see `maketree`).

`[T,D,RN] = wpcutree(T,D,L)` returns the same arguments as above and in addition, the vector `RN` contains the indices of the reconstructed nodes.

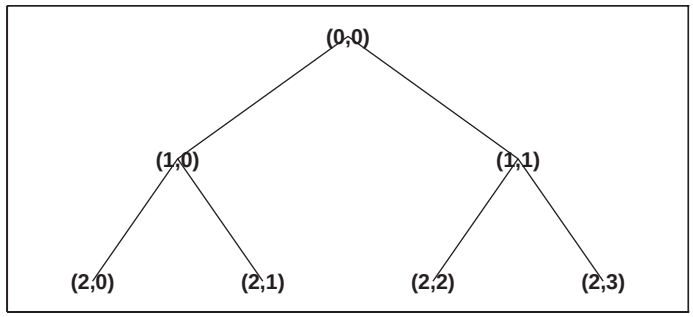
Examples

```
% Load signal.
    load noisdopp; x = noisdopp;

% Decompose x at depth 3 with db1 wavelet packets.
    [wpt,wpd] = wpdec(x,3,'db1');

% Plot wavelet packet tree structure wpt.
    plottree(wpt)
```




```
% Cut wavelet packet tree at level 2.
    [nwpt,nwpd] = wpcutree(wpt,wpd,2);
```

```
% Plot new wavelet packet tree structure wpt.
    plottree(nwpt)
```



See Also maketree, wpdec, wpdec2

wpdec

Purpose

Wavelet packet decomposition 1-D.

Syntax

```
[T,D] = wpdec(X,N,'wname',E,P)
[T,D] = wpdec(X,N,'wname')
```

Description

`wpdec` is a one-dimensional wavelet packet analysis function.

`[T,D] = wpdec(X,N,'wname',E,P)` returns a tree structure `T` and a data structure `D` (see `maketree`), corresponding to a wavelet packet decomposition of the vector `X`, at level `N`, with a particular wavelet (*'wname'*, see `wfilters`).

`E` is a string containing the type of entropy (see `wentropy`):

`E` = `'shannon'`, `'threshold'`, `'norm'`, `'log energy'`, `'sure'`, `'user'`

`P` is an optional parameter:

`'shannon'` or `'log energy'`: `P` is not used

`'threshold'` or `'sure'`: `P` is the threshold ($0 \leq$ `P`)

`'norm'`: `P` is a power ($1 \leq$ `P` $<$ `2`)

`'user'`: `P` is a string containing a name of an user-defined function

`[T,D] = wpdec(X,N,'wname')` is equivalent to
`[T,D] = wpdec(X,N,'wname','shannon')`.

The wavelet packets method is a generalization of wavelet decomposition that offers a richer signal analysis. Wavelet packets atoms are waveforms indexed by three naturally interpreted parameters: position and scale as in wavelet decomposition, and frequency.

For a given orthogonal wavelet function, a library of wavelet packets bases is generated. Each of these bases offers a particular way of coding signals, preserving global energy and reconstructing exact features. The wavelet packets can then be used for numerous expansions of a given signal. The most suitable decomposition of a given signal with respect to an entropy-based criterion is then selected.

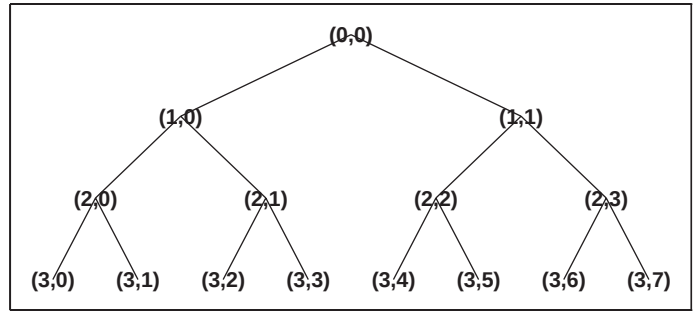
Simple and efficient algorithms exist for both wavelet packets decomposition and optimal decomposition selection. Adaptive filtering algorithms with direct applications in optimal signal coding and data compression can then be produced.

In the orthogonal wavelet decomposition procedure, the generic step splits the approximation coefficients into two parts. After splitting we obtain a vector of approximation coefficients and a vector of detail coefficients, both at a coarser scale. The information lost between two successive approximations is captured in the detail coefficients. The next step consists in splitting the new approximation coefficient vector; successive details are never re-analyzed.

In the corresponding wavelet packets situation, each detail coefficient vector is also decomposed into two parts using the same approach as in approximation vector splitting. This offers the richest analysis: the complete binary tree is produced in the one-dimensional case or a quaternary tree in the two-dimensional case.

Examples

```
% Load signal.
    load noisdopp; x = noisdopp;

% Decompose x at depth 3 with db1 wavelet packets
% using Shannon entropy.
    [t,d] = wpdec(x,3,'db1','shannon');

% The result is the wavelet packets decomposition structure
% which consists of a tree structure t and the associate
% data structure d.

% Plot tree structure (binary tree, or tree of order 2).
    plottree(t)

% Operations on the structure are defined in M-file
% wdatamgr and you are not supposed to handle
% this internal structure directly.
```



Algorithm

The algorithm used for the wavelet packets decomposition follows the same line as the wavelet decomposition process (see `dwt`, `wavedec`).

See Also

`maketree`, `waveinfo`, `wdatamgr`, `wentropy`, `wpdec2`, `wtreemgr`

References

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

Y. Meyer (1993), "Les ondelettes. Algorithmes et applications," Colin Ed., Paris, 2nd edition. (English translation: "Wavelets: Algorithms and Applications," SIAM).

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

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

Purpose

Wavelet packet decomposition 2-D.

Syntax

```
[T,D] = wpdec2(X,N,'wname',E,P)
[T,D] = wpdec2(X,N,'wname')
```

Description

`wpdec2` is a two-dimensional wavelet packet analysis function.

`[T,D] = wpdec2(X,N,'wname',E,P)` returns a tree structure `T` and a data structure `D` (see `maketree`), corresponding to a wavelet packet decomposition of the matrix `X`, at level `N`, with a particular wavelet (*'wname'*, see `wfilters`).

`E` is a string containing the type of entropy (see `wentropy`):

`E` = `'shannon'`, `'threshold'`, `'norm'`, `'log energy'`, `'sure'`, `'user'`

`P` is an optional parameter:

`'shannon'` or `'log energy'`: `P` is not used

`'threshold'` or `'sure'`: `P` is the threshold ($0 \le$ `P`)

`'norm'`: `P` is a power ($1 \le$ `P` $<$ `2`)

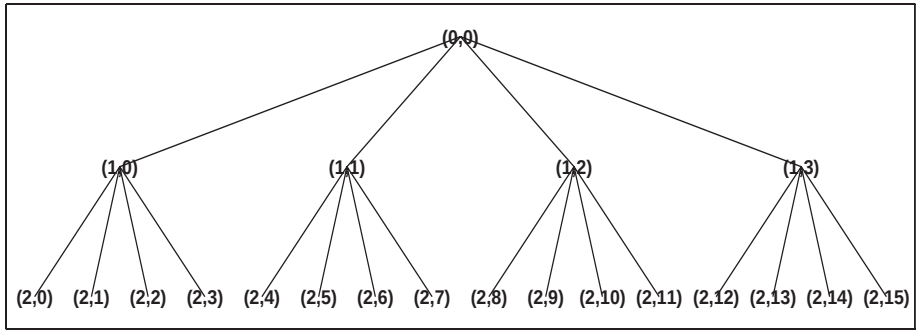
`'user'`: `P` is a string containing a name of an user-defined function

`[T,D] = wpdec2(X,N,'wname')` is equivalent to
`[T,D] = wpdec2(X,N,'wname','shannon')`.

See `wpdec` for a more complete description.

Examples

```
% Load image.
    load tire
    % X contains the loaded image.

% For an image the decomposition is performed using:
    [t,d] = wpdec2(X,2,'db1');
    % default entropy is the shannon one.

% Plot tree structure (quarternary tree, or tree of order 4).
    plottree(t)
```



Algorithm

The algorithm used for the wavelet packets decomposition follows the same line as the wavelet decomposition process (see `dwt2`, `wavedec2`).

See Also

`maketree`, `waveinfo`, `wdatamgr`, `wentropy`, `wpdec`, `wtreemgr`

References

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

Y. Meyer (1993), "Les ondelettes. Algorithmes et applications," Colin Ed., Paris, 2nd edition. (English translation: "Wavelets: Algorithms and Applications," SIAM).

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

M.V. Wickerhauser, (1994) "*Adapted wavelet analysis from theory to software Algorithms*," A.K. Peters.

Purpose

De-noising or compression using wavelet packet.

Syntax

```
[XD,TREED,DATAD,PERF0,PERFL2] =
   wpdencmp(X,SORH,N,'wname',CRIT,PAR,KEEPAPP)
[XD,TREED,DATAD,PERF0,PERFL2] =
   wpdencmp(TREE,DATA,SORH,CRIT,PAR,KEEPAPP)
```

Description

`wpdencmp` is a one- or two-dimensional de-noising and compression oriented function.

`wpdencmp` performs a de-noising or compression process of a signal or an image, using wavelet packet. The ideas and the procedures for de-noising and compression using wavelet packet are the same as those used in the wavelets framework (see `wden` and `wdencmp`).

`[XD,TREED,DATAD,PERF0,PERFL2] = wpdencmp(X,SORH,N,'wname',CRIT,PAR,KEEPAPP)` returns a de-noised or compressed version `XD` of input signal `X` (one- or two-dimensional) obtained by wavelet packet coefficients thresholding.

Additional output arguments `[TREED,DATAD]` are the wavelet packet best decomposition structure (see `besttree`) of `XD`. `PERFL2` and `PERF0` are L^2 recovery and compression scores in percentages.

`PERFL2` = 100 * (vector-norm of WP-cfs of `XD` / vector-norm of WP-cfs of `X`)^2.

If `X` is a one-dimensional signal and '*wname*' an orthogonal wavelet, `PERFL2` is reduced to $\frac{100\|XD\|^2}{\|X\|^2}$.

`SORH` (`'s'` or `'h'`) is for soft or hard thresholding (see `wthresh` for more details).

Wavelet packet decomposition is performed at level `N` and '*wname*' is a string containing wavelet name. Best decomposition is performed using entropy criterion defined by string `CRIT` and parameter `PAR` (see `wentropy` for details). Threshold parameter is also `PAR`. If `KEEPAPP` = 1, approximation coefficients cannot be thresholded, otherwise it is possible.

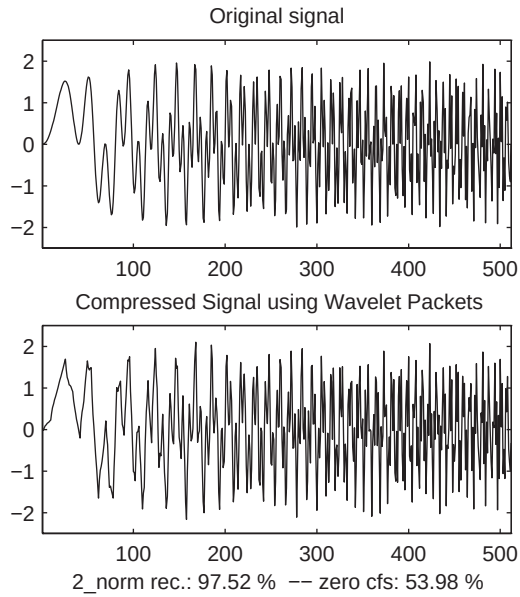
`[XD,TREED,DATAD,PERF0,PERFL2] = wpdencmp(TREE,DATA,SORH,CRIT,PAR,KEEPAPP)` has the same output arguments, using the same options as above, but obtained directly from the input wavelet packet decomposition structure `[TREE,DATA]` (see `maketree` and `wpdec`) of the signal to be de-noised or compressed.

In addition if `CRIT = 'nobest'` no optimization is done and the current decomposition is thresholded.

Examples

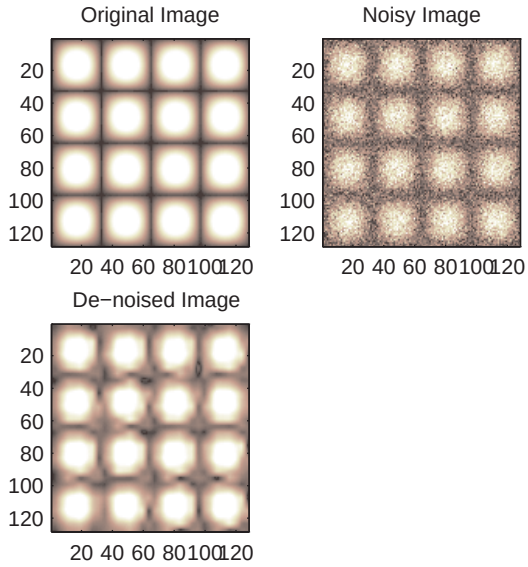
```
% Load original signal.
    load sumlichr; x = sumlichr;

% Use wpdencmp for signal compression.
    % find default values (see ddencmp).
    [thr,sorh,keepapp,crit] = ddencmp('cmp','wp',x)

thr =
    0.5193

sorh =
h

keepapp =
    1

crit =
    threshold

    % Denoise signal using global thresholding with
    % threshold best basis.
    [xc,treed,datad,perf0,perfl2] = ...
```

```
wpdencmp(x,sorh,3,'db2',crit,thr,keepapp);
```

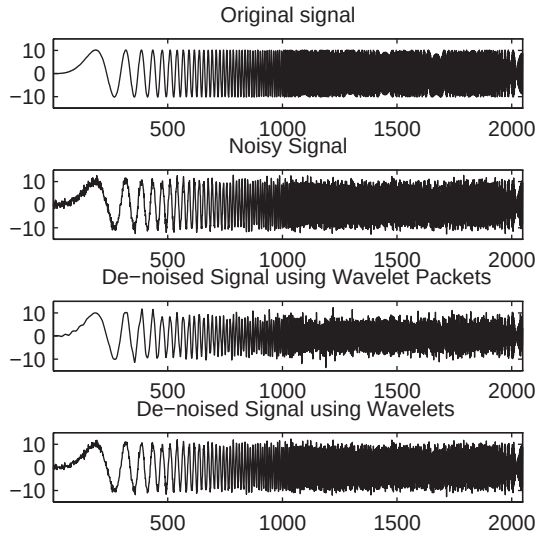



```
% Load original image.
    load sinsin

% Generate noisy image.
    init = 2055615866; randn('seed',init);
    x = X/18 + randn(size(X));

% Use wpdencmp for image de-noising.
    % find default values (see ddencmp).
    [thr,sorh,keepapp,crit] = ddencmp('den','wp',x)

thr =
    4.9685

sorh =
h
```

```
keepapp =
    1

crit =
sure
% Denoise image using global thresholding with
% SURE best basis.
xd = wpdencmp(x,sorh,3,'sym4',crit,thr,keepapp);
```




```
% Generate heavy sine and a noisy version of it.
    [xref,x] = wnoise(5,11,7,init);

% Use wpdencmp for signal de-noising.
    n = length(x);
    thr = sqrt(2*log(n*log(n)/log(2)));
    xwpd = wpdencmp(x,'s',4,'sym4','sure',thr,1);

% Compare with wavelet-based de-noising result.
    xwd = wden(x,'rigrsure','s','one',4,'sym4');
```



See Also

`ddencmp`, `wdencmp`, `wentropy`, `wpdec`, `wpdec2`

References

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

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

R.A. DeVore, B. Jawerth, B.J. Lucier (1992), "Image compression through wavelet transform coding," *IEEE Trans. on Inf. Theory*, vol. 38, No 2, pp. 719–746.

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

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

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

wpfun

Purpose

Wavelet packet functions.

Syntax

```
[WPWS,X] = wpfun('wname',NUM,PREC)
[WPWS,X] = wpfun('wname',NUM)
```

Description

`wpfun` is a wavelet packet analysis function.

`[WPWS,X] = wpfun('wname',NUM,PREC)` computes the wavelet packets for a wavelet '*wname*' (see `wfilters`), on dyadic intervals of length $2^{-\mathrm{PREC}}$.

`PREC` must be a positive integer. Output matrix `WPWS` contains the *W* functions of index from 0 to `NUM`, stored rowwise as $[W_0; W_1;...; W_{NUM}]$. Output vector `X` is the corresponding common `X`-grid vector.

`[WPWS,X] = wpfun('wname',NUM)` is equivalent to
`[WPWS,X] = wpfun('wname',NUM,7)`.

The computation scheme for wavelet packets generation is easy when using an orthogonal wavelet. We start with the two filters of length $2N$, denoted $h(n)$ and $g(n)$, corresponding to the wavelet. They are the reversed versions of the low-pass decomposition filter and the high-pass decomposition filter divided by $\sqrt{2}$ respectively.

Now by induction let us define the following sequence of functions $(W_n(x)\ ,\ n = 0,1,2,...)$ by:

$$W_{2n}(x) = 2 \sum_{k = 0, \ldots, 2N-1} h(k) W_n(2x-k)$$

$$W_{2n+1}(x) = 2 \sum_{k = 0, \ldots, 2N-1} g(k) W_n(2x-k)$$

where $W_0(x) = \phi(x)$ is the scaling function and $W_1(x) = \psi(x)$ is the wavelet function.

For example for the Haar wavelet we have:

$N = 1$, $h(0) = h(1) = 1/2$ and $g(0) = - g(1) = 1/2$.

The equations become:

$$W_{2n}(x) = W_n(2x) + W_n(2x - 1)$$

$$W_{2n+1}(x) = W_n(2x) - W_n(2x - 1)$$

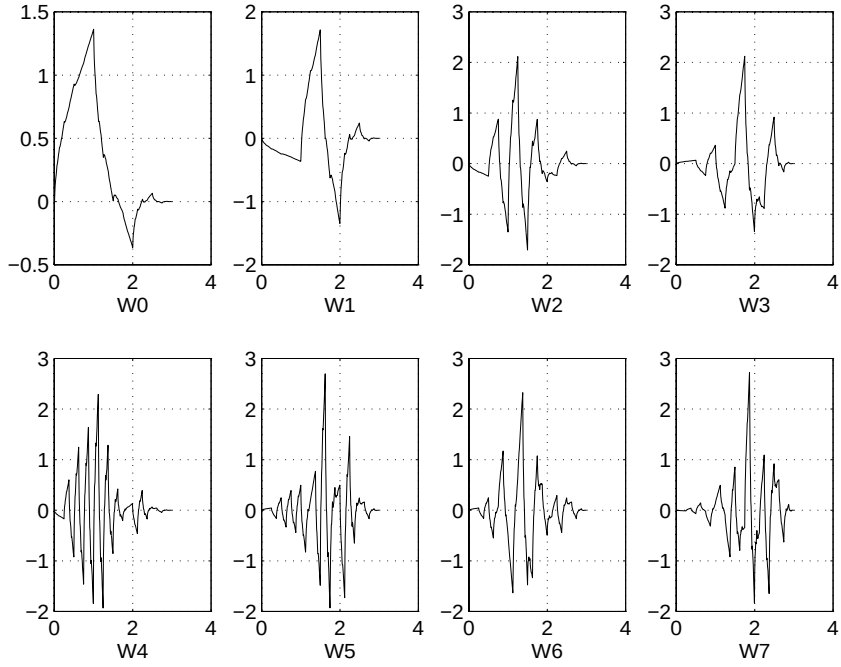
$W_0(x) = \phi(x)$ is the haar scaling function and $W_1(x) = \psi(x)$ is the Haar wavelet, both supported in [0,1].

Then we can obtain W_{2n} by adding two 1/2-scaled versions of W_n with distinct supports [0,1/2] and [1/2,1] and obtain W_{2n+1} by subtracting the same versions of W_n.

Starting from more regular original wavelets, using a similar construction, we obtain smoothed versions of this system of *W*-functions, all with support in the interval [0, 2*N*-1].

Examples

```
% Compute the db2 Wn functions for n = 0 to 7, generating
% the db2 wavelet packets.
[wp,x] = wpfun('db2',7);
```



See Also

`wavefun`, `waveinfo`

References

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

Y. Meyer (1993), "Les ondelettes. Algorithmes et applications," Colin Ed., Paris, 2nd edition. (English translation: "Wavelets: Algorithms and applications," SIAM).

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

M.V. Wickerhauser, (1994) "Adapted wavelet analysis from theory to software algorithms," A.K. Peters.

Purpose

Recompose wavelet packet.

Syntax

```
[T,D] = wpjoin(T,D,N)
[T,D,X] = wpjoin(T,D,N)
[T,D] = wpjoin(T,D)
[T,D,X] = wpjoin(T,D)
```

Description

`wpjoin` is a one- or two-dimensional wavelet packet analysis function. `wpjoin` updates the tree and data structures after the recomposition of a node.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

`[T,D] = wpjoin(T,D,N)` returns the modified tree structure `T` and the modified data structure `D` (see `maketree`), corresponding to a recomposition of the node `N`.

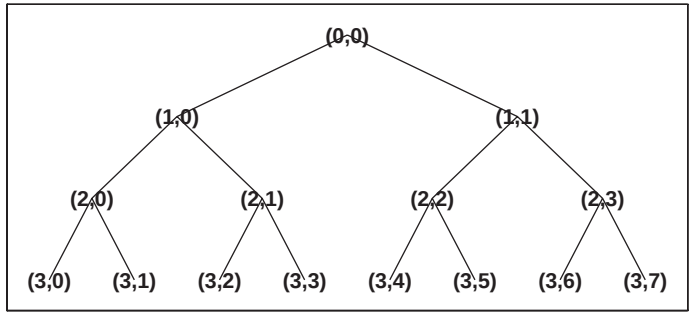
`[T,D,X] = wpjoin(T,D,N)` also returns the coefficients of the node.

`[T,D] = wpjoin(T,D)` is equivalent to `[T,D] = wpjoin(T,D,0)`.

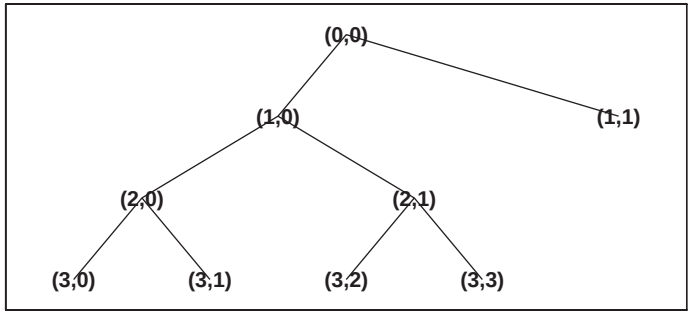
`[T,D,X] = wpjoin(T,D)` is equivalent to `[T,D,X] = wpjoin(T,D,0)`.

Examples

```
% Load signal.
    load noisdopp; x = noisdopp;

% Decompose x at depth 3 with db1 wavelet packets.
    [wpt,wpd] = wpdec(x,3,'db1');

% Plot wavelet packet tree structure wpt.
    plottree(wpt)
```




```
% Recompose packet (1,1) or 2
    [wpt,wpd] = wpjoin(wpt,wpd,[1 1]);

% Plot wavelet packet tree structure wpt.
    plottree(wpt)
```



See Also

maketree, wpdec, wpdec2, wpsplt

Purpose

Reconstruct wavelet packet coefficients.

Syntax

```
X = wprcoef(T,D,N)
```

Description

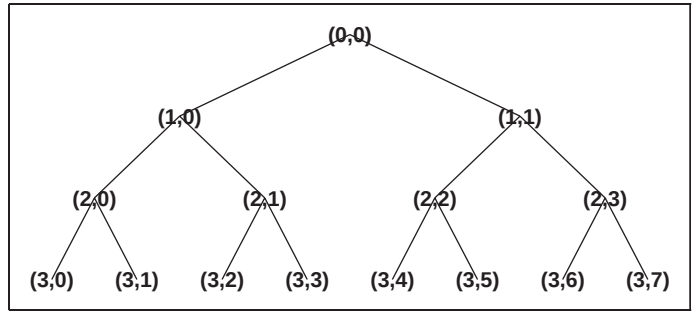
`wprcoef` is a one- or two-dimensional wavelet packet analysis function.

`X = wprcoef(T,D,N)` computes reconstructed coefficients of the node `N`. `T` is the tree structure and `D` the data structure (see `maketree`).

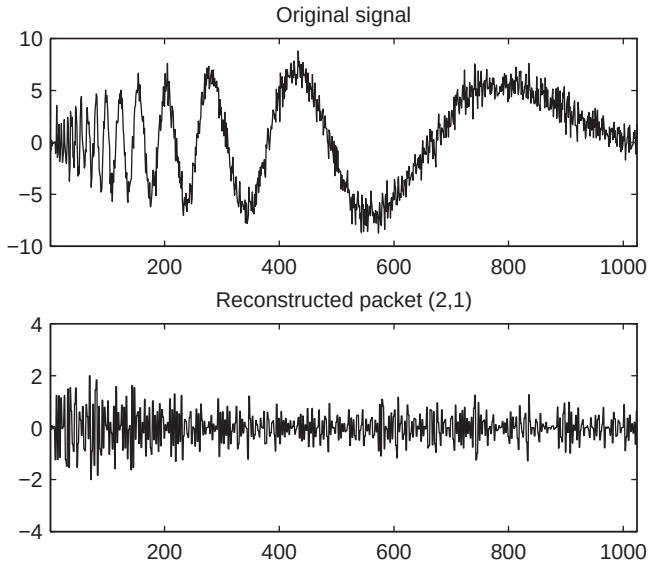
`X = wprcoef(T,D)` is equivalent to `X = wprcoef(T,D,0)`.

Examples

```
% Load signal.
    load noisdopp; x = noisdopp;

    figure(1); subplot(211);
    plot(x); title('Original signal');

% Decompose x at depth 3 with db1 wavelet packets
% using Shannon entropy.
    [t,d] = wpdec(x,3,'db1','shannon');

% Plot tree structure.
    plottree(t)
```



```
% Reconstruct packet (2,1).
    rcfs = wprcoef(t,d,[2 1]);

    figure(1); subplot(212);
    plot(rcfs); title('Reconstructed packet (2,1)');
```



See Also `maketree`, `wpdec`, `wpdec2`, `wprec`, `wprec2`

Purpose

Wavelet packet reconstruction 1-D.

Syntax

```
X = wprec(T,D)
```

Description

`wprec` is a one-dimensional wavelet packet analysis function.

`X = wprec(T,D)` returns the reconstructed vector `X` corresponding to a wavelet packet decomposition structure `[T,D]`. `T` is the tree structure and `D` the data structure (see `maketree`).

`wprec` is the inverse function of `wpdec` in the sense that the abstract statement `wprec(wpdec(X,'wname'))` gets back to `X`.

See Also

`maketree`, `wpdec`, `wpdec2`, `wpjoin`, `wprec2`, `wpsplt`

wprec2

Purpose

Wavelet packet reconstruction 2-D.

Syntax

```
X = wprec2(T,D)
```

Description

`wprec2` is a two-dimensional wavelet packet analysis function.

`X = wprec2(T,D)` returns the reconstructed matrix `X` corresponding to a wavelet packet decomposition structure `[T,D]`. `T` is the tree structure and `D` the data structure (see `maketree`).

`wprec2` is the inverse function of `wpdec2` in the sense that the abstract statement `wprec2(wpdec2(X,'`*`wname`*`'))` gets back to `X`.

See Also

`maketree`, `wpdec`, `wpdec2`, `wpjoin`, `wprec`, `wpsplt`

Purpose

Split (decompose) wavelet packet.

Syntax

```
[T,D] = wpsplt(T,D,N)
[T,D,cA,cD] = wpsplt(T,D,N)
[T,D,cA,cH,cV,cD] = wpsplt(T,D,N)
```

Description

`wpsplt` is a one- or two-dimensional wavelet packet analysis function.

`wpsplt` updates the tree and data structures after the decomposition of a node.

`[T,D] = wpsplt(T,D,N)` returns the modified tree structure `T` and the modified data structure `D`, corresponding to the decomposition of the node `N`.

For a one-dimensional decomposition:

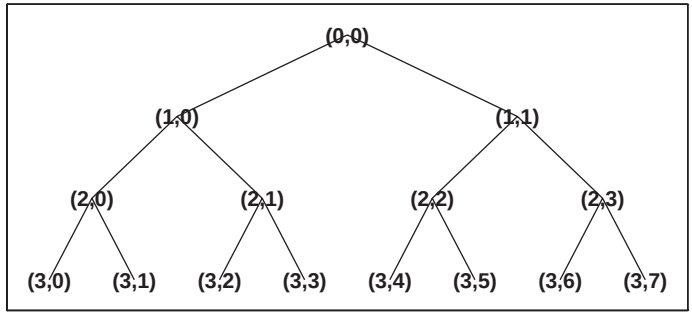
`[T,D,cA,cD] = wpsplt(T,D,N)` with `cA` = approximation and `cD` = detail of node `N`.

For a two-dimensional decomposition:

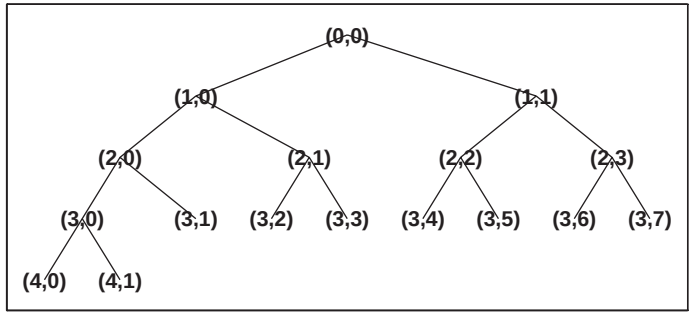
`[T,D,cA,cH,cV,cD] = wpsplt(T,D,N)` with `cA` = approximation and `cH`, `cV`, `cD` = details of node `N`.

Examples

```
% Load signal.
    load noisdopp;
    x = noisdopp;

% Decompose x at depth 3 with db1 wavelet packets.
    [wpt,wpd] = wpdec(x,3,'db1');

% Plot wavelet packet tree structure wpt.
    plottree(wpt)
```




```
% Decompose packet (3,0).
    [wpt,wpd] = wpsplt(wpt,wpd,[3 0]);
    % or equivalently wpsplt(wpt,wpd,7).

% Plot wavelet packet tree structure wpt.
    plottree(wpt)
```



See Also

maketree, wavedec, wavedec2, wpdec, wpdec2, wpjoin

Purpose

Wavelet packet coefficients thresholding.

Syntax

```
NDATA = wpthcoef(DATA,TREE,KEEPAPP,SORH,THR)
```

Description

`wpthcoef` is a one- or two-dimensional de-noising and compression utility.

`NDATA = wpthcoef(DATA,TREE,KEEPAPP,SORH,THR)` returns a new data structure obtained from the wavelet packet decomposition structure `[DATA,TREE]` (see `maketree`) by coefficients thresholding.

If `KEEPAPP` = 1, approximation coefficients are not thresholded, otherwise it is possible.

If `SORH` = `'s'`, soft thresholding is applied, if `SORH` = `'h'`, hard thresholding is applied (see `wthresh`).

`THR` is the threshold value.

See Also

`maketree`, `wpdec`, `wpdec2`, `wpdencmp`, `wthresh`

wrcoef

Purpose

Reconstruct single branch from 1-D wavelet coefficients.

Syntax

```
X = wrcoef('type',C,L,'wname',N)
X = wrcoef('type',C,L,Lo_R,Hi_R,N)
X = wrcoef('type',C,L,'wname')
X = wrcoef('type',C,L,Lo_R,Hi_R)
```

Description

`wrcoef` reconstructs the coefficients of a one-dimensional signal, given a wavelet decomposition structure (`C` and `L`) and either a specified wavelet (*'wname'*, see `wfilters`) or specified reconstruction filters (`Lo_R` and `Hi_R`).

`X = wrcoef('type',C,L,'wname',N)` computes the vector of reconstructed coefficients, based on the wavelet decomposition structure `[C,L]` (see `wavedec`), at level `N`.

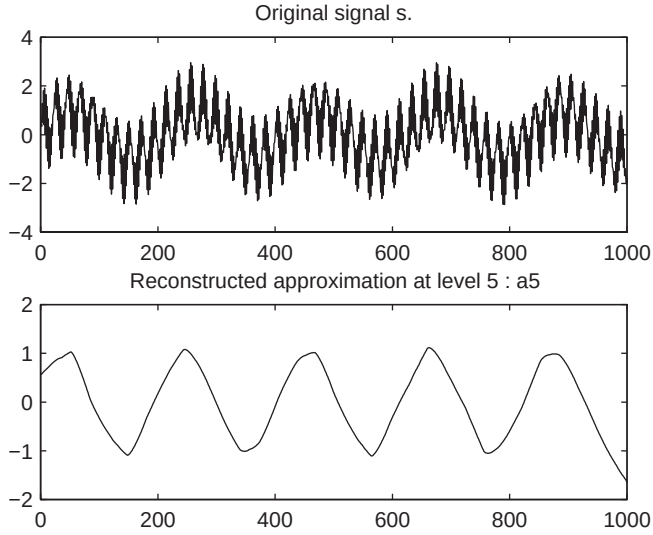
Argument *'type'* determines whether approximation (*'type'* = `'a'`) or detail (*'type'* = `'d'`) coefficients are reconstructed. When *'type'* = `'a'`, `N` is allowed to be 0, otherwise strictly positive `N` is required. Level `N` must be an integer such that `N <= length(L)-2`.

`X = wrcoef('type',C,L,Lo_R,Hi_R,N)` computes coefficients as above, given the reconstruction filters you specify.

`X = wrcoef('type',C,L,'wname')` and `X = wrcoef('type',C,L,Lo_R,Hi_R)` reconstruct coefficients of maximum level `N = length(L)-2`.

Examples

```
% Load original one-dimensional signal.
    load sumsin; s = sumsin;

% Perform decomposition at level 5 of s using sym4.
    [c,l] = wavedec(s,5,'sym4');

% Reconstruct approximation at level 5,
% from the wavelet decomposition structure [c,l].
    a5 = wrcoef('a',c,l,'sym4',5);
```



See Also

`appcoef`, `detcoef`, `wavedec`

wrcoef2

Purpose

Reconstruct single branch from 2-D wavelet coefficients.

Syntax

```
X = wrcoef2('type',C,S,'wname',N)
X = wrcoef2('type',C,S,Lo_R,Hi_R,N)
X = wrcoef2('type',C,S,'wname')
X = wrcoef2('type',C,S,Lo_R,Hi_R)
```

Description

`wrcoef2` is a two-dimensional wavelet analysis function. `wrcoef2` reconstructs the coefficients of an image.

`X = wrcoef2('type',C,S,'wname',N)` computes the matrix of reconstructed coefficients of level `N`, based on the wavelet decomposition structure `[C,S]` (see `wavedec2`).

'*wname*' is a string containing the name of the wavelet. If '*type*' = `'a'`, approximation coefficients are reconstructed; otherwise if '*type*' = `'h'` (`'v'` or `'d'` respectively), horizontal (vertical or diagonal respectively) detail coefficients are reconstructed.

Level `N` must be an integer such that: `0 <= N <= size(S,1)-2` if '*type*' = `'a'` and such that `1 <= N <= size(S,1)-2` if '*type*' = `'h'`, `'v'` or 'd'.

Instead of giving the wavelet name, you can give the filters.

For `X = wrcoef2('type',C,S,Lo_R,Hi_R,N)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

`X = wrcoef2('type',C,S,'wname')` or `X = wrcoef2('type',C,S,Lo_R,Hi_R)` reconstructs coefficients of maximum level `N = size(S,1)-2`.

Examples

```
% Load original image.
    load woman;
    % X contains the loaded image.

% Perform decomposition at level 2
% of X using sym5.
    [c,s] = wavedec2(X,2,'sym5');

% Reconstruct approximations at
% levels 1 and 2, from the wavelet
% decomposition structure [c,s].
    a1 = wrcoef2('a',c,s,'sym5',1);
    a2 = wrcoef2('a',c,s,'sym5',2);

% Reconstruct details at level 2,
% from the wavelet decomposition
% structure [c,s].
% 'h' is for horizontal,
% 'v' is for vertical,
% 'd' is for diagonal.
    hd2 = wrcoef2('h',c,s,'sym5',2);
    vd2 = wrcoef2('v',c,s,'sym5',2);
    dd2 = wrcoef2('d',c,s,'sym5',2);

% All these images are of same size sX.
    sX = size(X)
sX =
 256 256
    sa1 = size(a1)

sa1 =
 256 256
    shd2 = size(hd2)

shd2 =
 256 256
```

See Also

appcoef2, detcoef2, wavedec2

wrev

Purpose

Flip vector.

Syntax

```
Y = wrev(X)
```

Description

`wrev` is a general utility.

`Y = wrev(X)` reverses the vector `X`.

Examples

```
% Set simple vector.
    v = [1 2 3];

% Reverse v.
    wrev(v)

ans =
    3     2    1

% Reverse v transpose.
    wrev(v')

ans =
    3
    2
    1
```

See Also

`fliplr`, `flipud`

Purpose

Wavelet coefficients thresholding 1-D.

Syntax

```
NC = wthcoef('d',C,L,N,P)
NC = wthcoef('d',C,L,N)
NC = wthcoef('a',C,L)
NC = wthcoef('t',C,L,N,T,SORH)
```

Description

`wthcoef` is a one-dimensional de-noising and compression oriented function.

`NC = wthcoef('d',C,L,N,P)` returns coefficients obtained from the wavelet decomposition structure `[C,L]` (see `wavedec`), by rate compression defined in vectors `N` and `P`. `N` contains the detail levels to be compressed and `P` the corresponding percentages of lower coefficients to be set to zero. `N` and `P` must be of same length. Vector `N` must be such that `1 <= N(i) <= length(L)-2`.

`NC = wthcoef('d',C,L,N)` returns coefficients obtained from `[C,L]` by setting to zero all the coefficients of detail levels defined in `N`.

`NC = wthcoef('a',C,L)` returns coefficients obtained by setting approximation coefficients to zero.

`NC = wthcoef('t',C,L,N,T,SORH)` returns coefficients obtained from the wavelet decomposition structure `[C,L]` by soft (if `SORH='s'`) or hard (if `SORH='h'`) thresholding (see `wthresh`) defined in vectors `N` and `T`. `N` contains the detail levels to be thresholded and `T` the corresponding thresholds. `N` and `T` must be of the same length.

`[NC,L]` is the resulting wavelet decomposition structure.

See Also

`wavedec`, `wthresh`

wthcoef2

Purpose

Wavelet coefficients thresholding 2-D.

Syntax

```
NC = wthcoef2('type',C,S,N,T,SORH)
NC = wthcoef2('type',C,S,N)
NC = wthcoef2('a',C,S)
NC = wthcoef2('t',C,S,N,T,SORH)
```

Description

`wthcoef2` is a two-dimensional de-noising and compression oriented function.

For '*type*' = `'h'` (`'v'` or `'d'`), `NC = wthcoef2('type',C,S,N,T,SORH)` returns the horizontal (vertical or diagonal respectively) coefficients obtained from the wavelet decomposition structure `[C,S]` (see `wavedec2`), by soft (if `SORH='s'`) or hard (if `SORH='h'`) thresholding defined in vectors `N` and `T`. `N` contains the detail levels to be compressed and `T` the corresponding thresholds. `N` and `T` must be of the same length. The vector `N` must be such that `1 <= N(i) <= size(S,1)-2`.

For '*type*' = `'h'` (`'v'` or `'d'` respectively), `NC = wthcoef2('type',C,S,N)` returns the coefficients of '*type*' orientation obtained from `[C,S]` by setting to zero all the coefficients of detail levels defined in `N`.

`NC = wthcoef2('a',C,S)` returns the coefficients obtained by setting approximation coefficients to zero.

`NC = wthcoef2('t',C,S,N,T,SORH)` returns the detail coefficients obtained from the wavelet decomposition structure `[C,S]` by soft (if `SORH='s'`) or hard (if `SORH='h'`) thresholding (see `wthresh`) defined in vectors `N` and `T`. `N` contains the detail levels to be thresholded and `T` the corresponding thresholds which are applied in the three detail orientations. `N` and `T` must be of the same length.

`[NC,S]` is the resulting wavelet decomposition structure.

See Also

`wavedec2`, `wthresh`

Purpose

Perform soft or hard thresholding.

Syntax

```
Y = wthresh(X,SORH,T)
```

Description

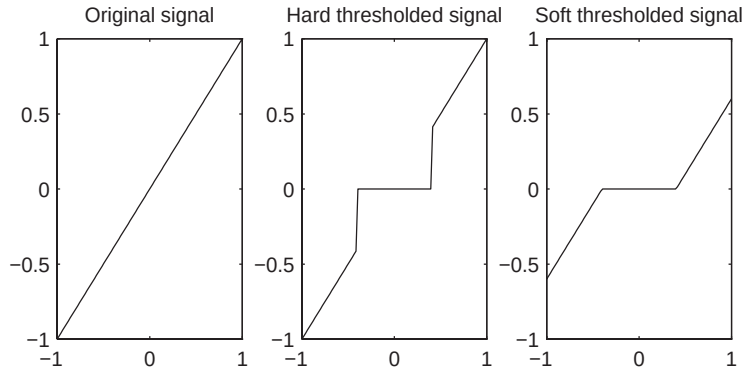
`Y = wthresh(X,SORH,T)` returns the soft (if `SORH = 's'`) or hard (if `SORH = 'h'`) `T`-thresholding of the input vector or matrix `X`. `T` is the threshold value.

`Y = wthresh(X,'s',T)` returns $Y = SIGN(X) \cdot (|X| - T)_+$, soft thresholding is wavelet shrinkage.

`Y = wthresh(X,'h',T)` returns $Y = X.1_{(|X| > T)}$, hard thresholding is more crude.

Examples

```
% Generate signal and set threshold.
    y = linspace(-1,1,100);
    thr = 0.4;

% Perform hard thresholding.
    ythard = wthresh(y,'h',thr);

% Perform soft thresholding.
    ytsoft = wthresh(y,'s',thr);
```



See Also

`wden`, `wdencmp`, `wpdencmp`

wtreemgr

Purpose

Manager for tree structure.

Syntax

```
[OUT1,OUT2,OUT3,OUT4] = wtreemgr(OPT,STRUCTURE,IN3,IN4,IN5)
```

Description

`wtreemgr` is a tree management utility.

Allowed values for `OPT` and associated uses are described in the functions listed in the See Also section:

`'allnodes'`	:	All nodes
`'isnode'`	:	Check if node
`'istnode'`	:	Check if terminal node
`'create'`	:	Create a tree
`'nodeasc'`	:	Node ascendants
`'nodedesc'`	:	Node descendants
`'nodepar'`	:	Node parent
`'ntnode'`	:	Number of terminal nodes
`'tnodes'`	:	Terminal nodes
`'order'`	:	Order of tree
`'depth'`	:	Depth of tree

For tree structure implementation see `maketree`.

See Also

`allnodes`, `isnode`, `istnode`, `maketree`, `nodeasc`, `nodedesc`, `nodepar`, `ntnode`, `tnodes`, `treedpth`, `treeord`

B

GUI Reference

This appendix explains some of the features of the Wavelet Toolbox graphical user interface (GUI) that have not been described in the previous chapters. Topics include:

- General Features
- Continuous Wavelet Tool Features
- Wavelet 1-D Tool Features
- Wavelet 2-D Tool Features
- Wavelet Packet Tool Features (1-D and 2-D)
- Wavelet Display Tool
- Wavelet Packet Display Tool

General Features

Some features of the Wavelet Toolbox's graphical user interface apply to all or several of the tools in the toolbox. These include:

- Color coding
- "Connectedness" of plots
- Using the mouse
- Controlling the colormap
- Controlling the number of colors
- Controlling the coloration mode
- Customizing graphical objects
- Customizing print settings
- Using menus

Color Coding

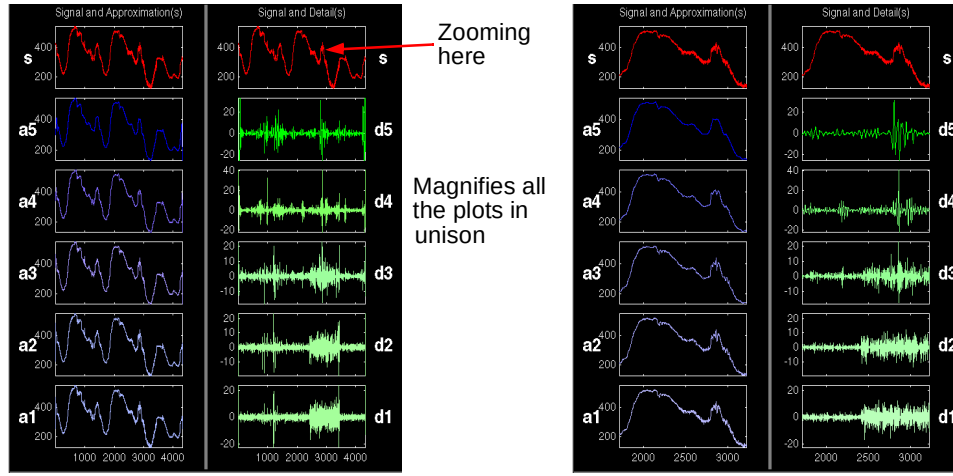
In all the graphical tools, the various signals and analysis components are color coded in this way:

Signal	Shown in
Original	Red
Reconstructed or synthesized	Yellow
Approximations	Variegated shades of blue (high level = darker)
Details	Variegated shades of green (high level = darker)

Connectedness of Plots

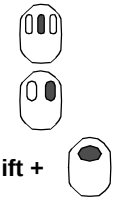
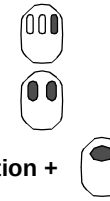
Plots that contain related information and are graphed on the same abscissa are "connected" in the sense that manipulations performed on one plot affect all the others in the same way.

For example, the approximations and details shown in the separate mode view of a decomposition all respond together when any of the plots is magnified or "zoomed":



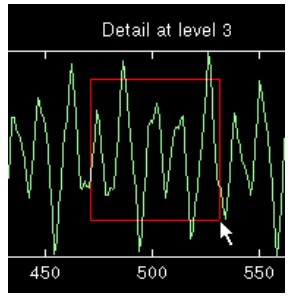
Using the Mouse

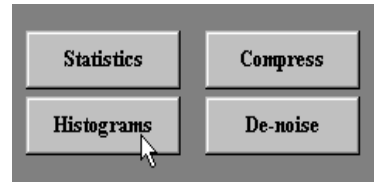
The Wavelet Toolbox uses three distinct types of mouse control:

Left Mouse Button	Middle Mouse Button	Right Mouse Button
Make selections, activate controls.	Display a cross-hair to show position-dependent information	Translate plots up and down, and left and right
	 Shift +	 Option +

Making Selections and Activating Controls

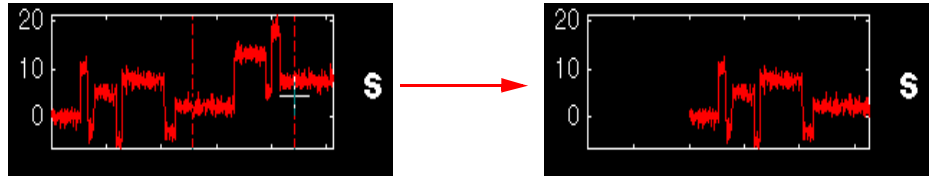
Most of the work you do with the Wavelet Toolbox graphical tools involves making selections and activating controls. You do this using the left (or only) mouse button.





Translating Plots

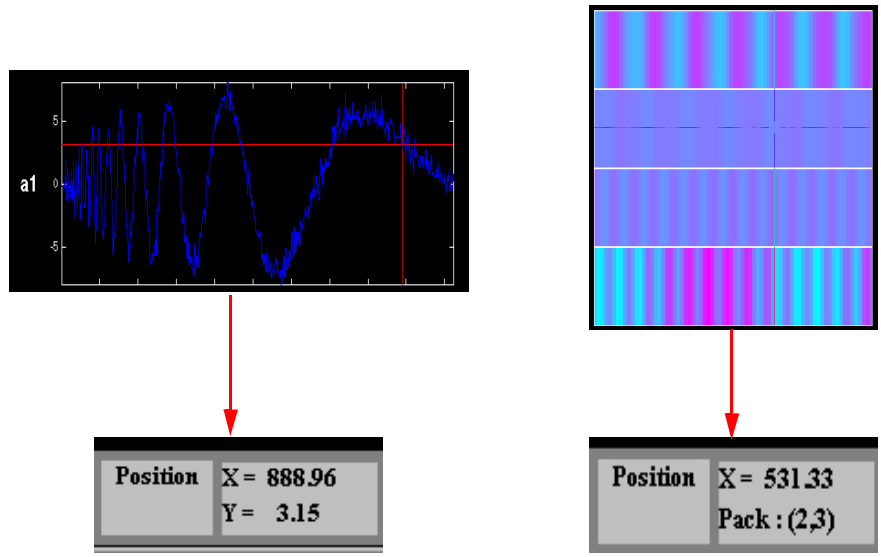
By holding down the right mouse button (or its equivalent on a one- or two-button mouse), you can move the mouse to draw a rectangle in either a horizontal or vertical orientation. Releasing the middle mouse button then causes the plot to shift horizontally or vertically by an amount proportional to the size of the rectangle.



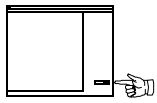
Displaying Position-Dependent Information

When you hold down the middle mouse button (or its equivalent on a one- or two-button mouse), a cross-hair cursor appears over the graph or plot. Position-dependent information also appears in the **Position** box located at the bottom center of the tool.

The type of information that appears depends on what tool you are using and on what plot your cursor is in.



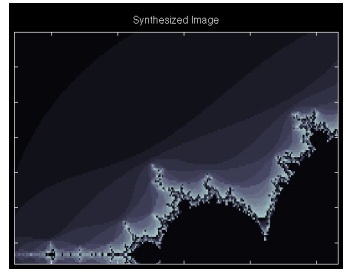
Controlling the Colormap



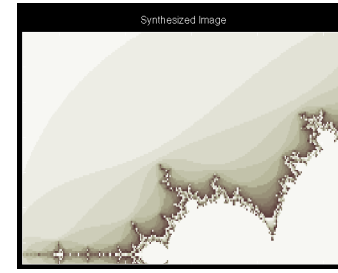
Colormap

The **Colormap** selection box, located at the bottom right of the window, allows you to adjust the colormap that is used to plot images or wavelet coefficients. This is more than an esthetic adjustment: you are likely to see different features depending on your colormap selection.

Consider these images of the Mandelbrot set generated in the **Wavelet Packet 2-D** tool, here using the `bone` and `1-bone` colormaps:

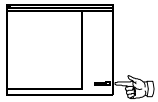


bone



1–bone

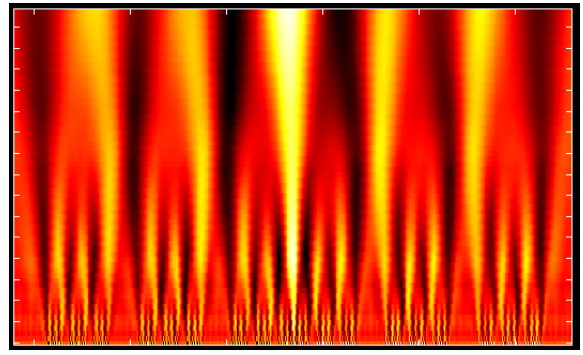
Controlling the Number of Colors



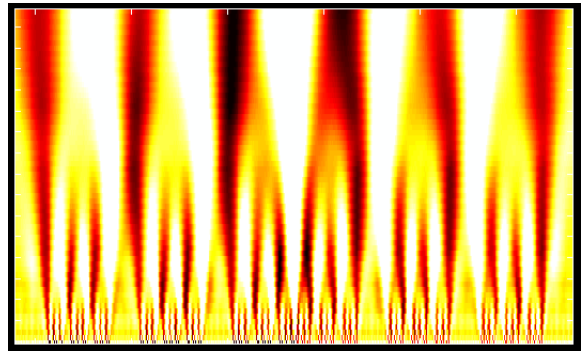
Nb. Colors

The **Nb. Colors** slider, also located at the bottom right of the window, allows you to adjust how many colors the tool uses to plot images or wavelet packet coefficients (you can also use the edit control). At first glance, this might not seem to be particularly important. However, adjusting the number of colors can highlight different features of the plot.

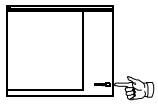
Consider the coefficients plot of the Koch curve generated in the **Continuous Wavelet** tool, here using 129 colors:



and here using 68 colors:

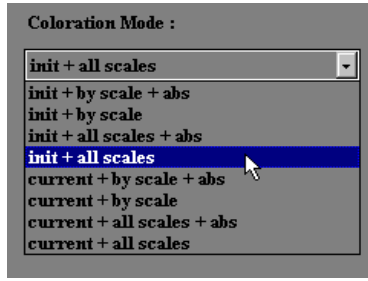


Controlling the Coloration Mode



Coloration mode

In **Wavelet 1-D** tool and **Continuous Wavelet** tool the coloration of coefficients can be done in several different ways.



Three parameters are used to do coefficients coloration:

init or **current**:

When **init** is selected, coloration is performed with all the coefficients values. When **current** is selected, only a portion of the coefficients is used to make the coloration. This portion is taken from the current axis limits of the displayed coefficients.

by level or **all levels**:

When **by level** is selected, the coloration is done separately for each detail level. Otherwise the wavelet coefficients at **all levels** are used to scale the coloration.

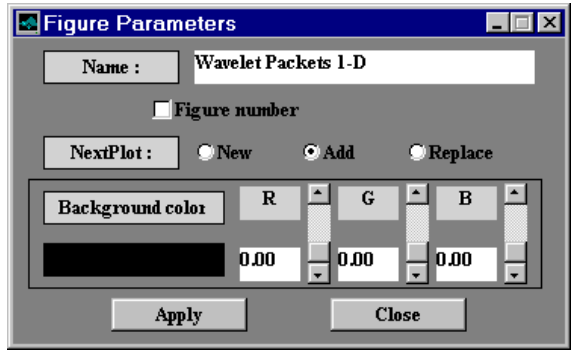
abs (or not):

When **abs** is selected, the absolute values of the coefficients are used.

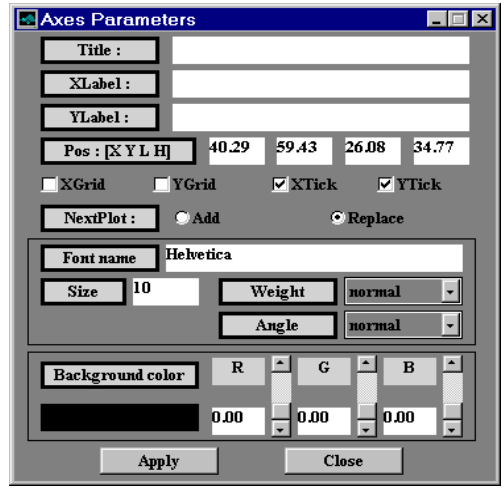
Customizing Graphical Objects

In order to customize your graphics settings, you can select in all the windows the **Options⇒Handles Graphics Settings** menu option. When the **Axes Settings** sub-menu option is selected, you are asked to disable or not the Dynamical Visualization Tool (DVT), located at the bottom of the window. So, after the desired customization is performed, you must disable the **Axes Settings** sub-menu option in order to reactive the DVT.

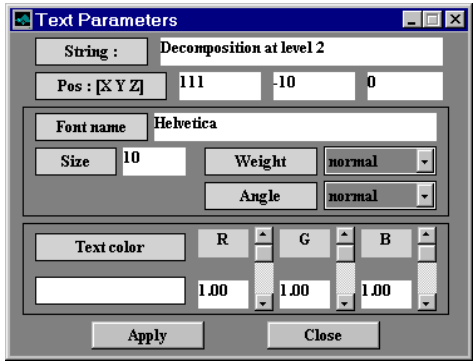
When the sub-menu **Window Settings** is enabled, you can edit the current figure parameters.



When the sub-menu **Axes Settings** is enabled, clicking with the mouse on an axis will select it for editing.

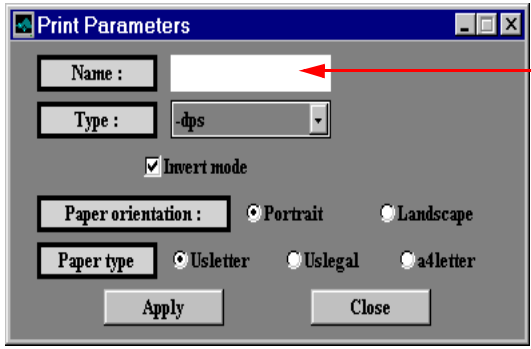


When the sub-menu **Texts Settings** is enabled, double-clicking with the mouse on an axis will select it for editing.



Customizing Print Settings

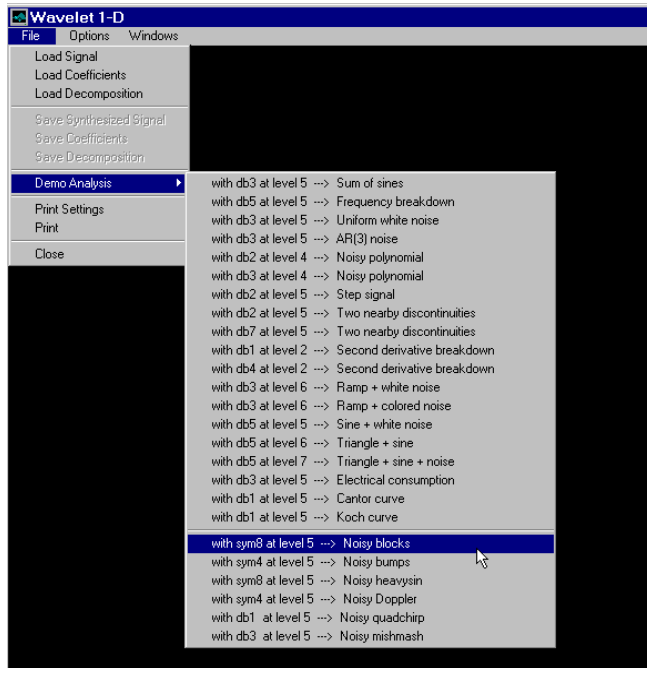
Using the menu option **File⇒Print Settings** you can access to the **Print Parameters** window.



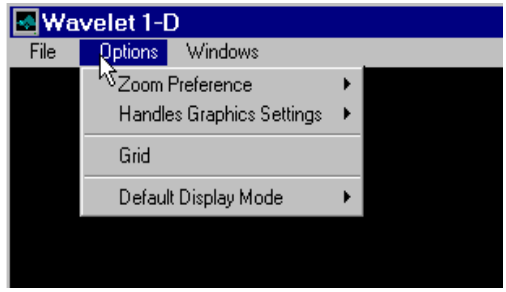
If you want to print to a file, enter a name here.

Using Menus

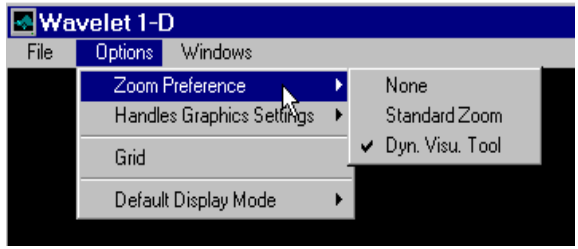
Almost all the windows provide a similar structure at the top of the window. The main analysis windows have a **File⇒Demo Analysis** menu option which allows you to select an example of analysis with pre-defined parameters. Here is an example of the **Wavelet 1-D** tool.



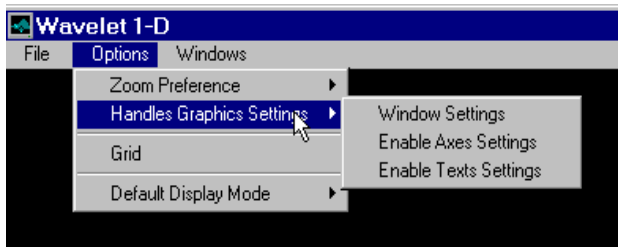
The **Options** menu allows you to change the current settings in your windows.



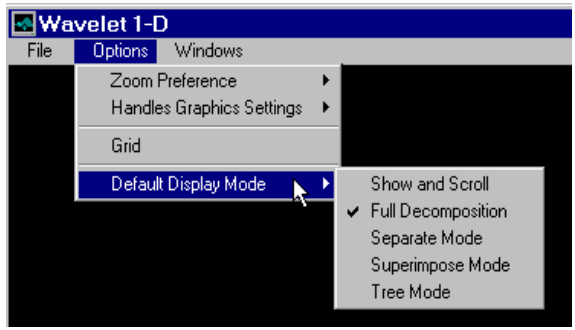
Choosing Zoom Preferences:



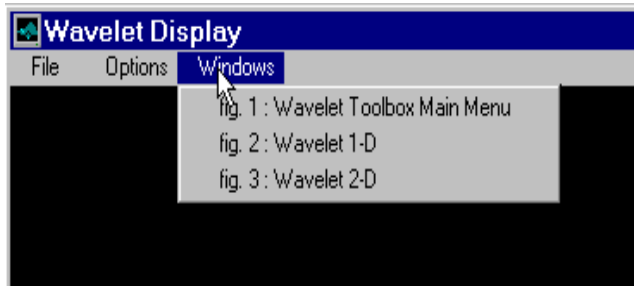
Enabling or disabling the graphical objects settings modification:



Choosing the Default Display Mode you want to be used in the first display of the **Wavelet 1-D** tool:

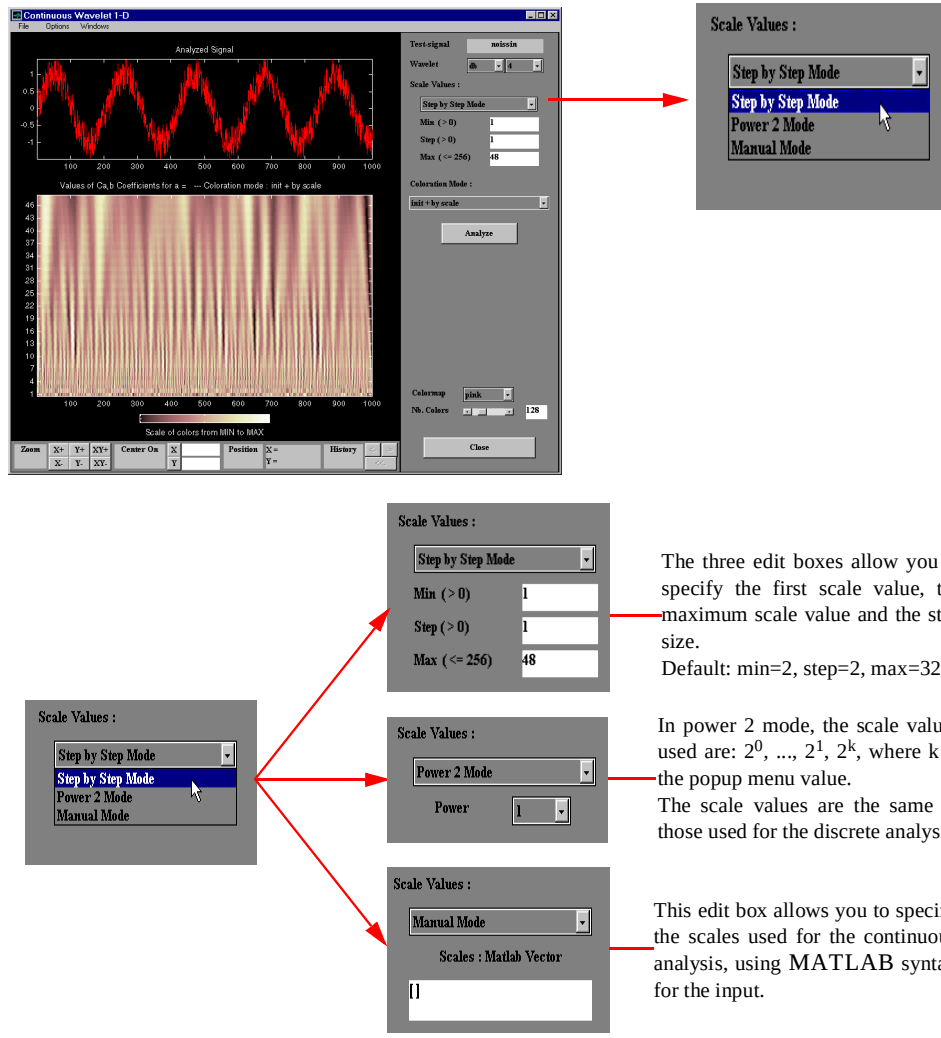


The **Windows** menu allows to jump directly from a window to another.



Continuous Wavelet Tool Features

The **Continuous Wavelet Tool** has been almost completely described in the section "Continuous Analysis Using the Graphical Interface" on page 2-7. Here is an example of an option, previously not described, that allows you to perform analysis using different scale modes.



The three edit boxes allow you to specify the first scale value, the maximum scale value and the step size.
Default: min=2, step=2, max=32

In power 2 mode, the scale values used are: 2^0, ..., 2^1, 2^k, where k is the popup menu value.
The scale values are the same as those used for the discrete analysis.

This edit box allows you to specify the scales used for the continuous analysis, using MATLAB syntax for the input.

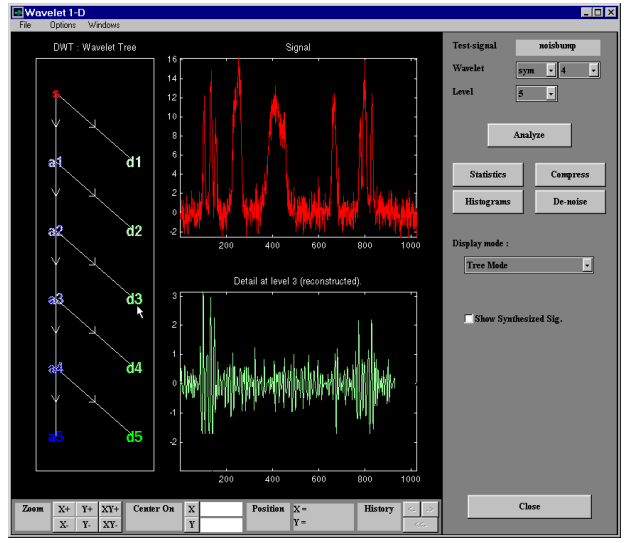
Wavelet 1-D Tool Features

The **Wavelet 1-D Tool** has been almost completely described in the section “One-Dimensional Analysis Using the Graphical Interface” on page 2- 22. Here are two examples of options not covered previously.

Tree Mode

This is one of the display options in which by selecting a node in the tree you can view the corresponding signal.

Here on the left, the node **d3** is selected and the corresponding detail is displayed under the original signal.

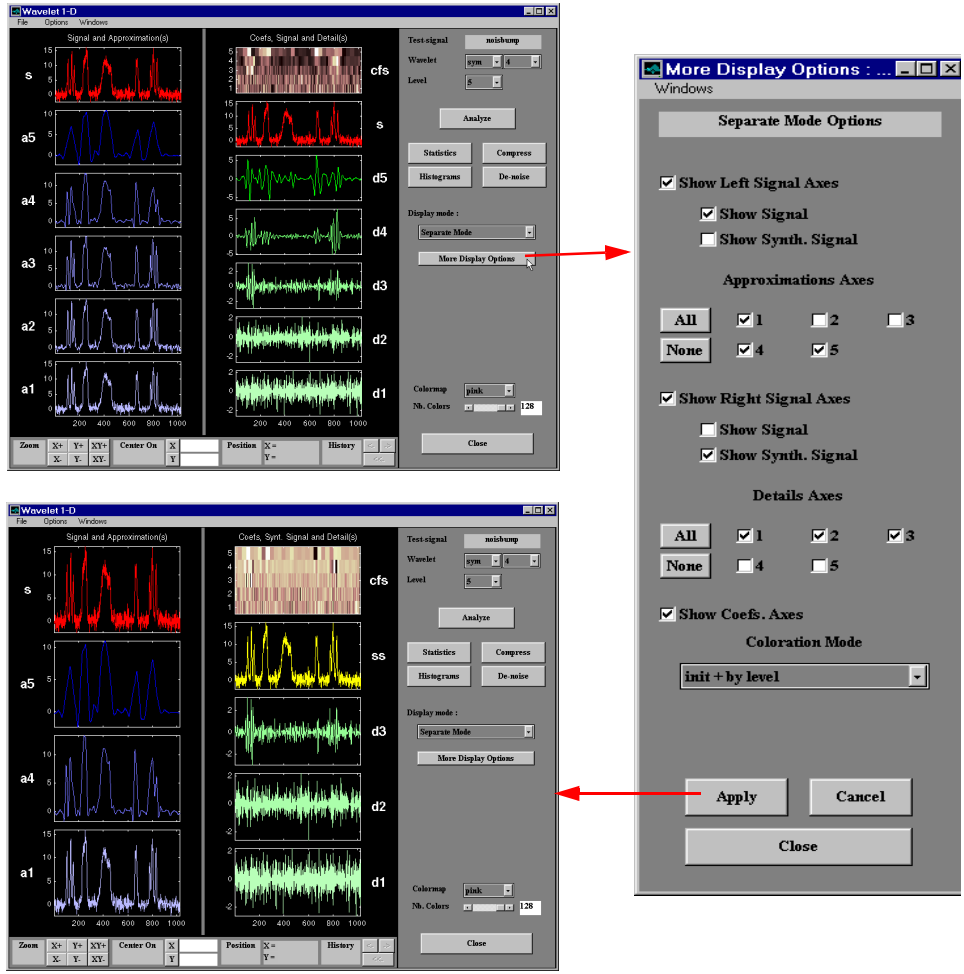


More Display Options

This option allows you to customize what is displayed and is dependent on the current visualization mode.

In this example for the **Separate Mode**, we have chosen not to display the coefficients of approximation for levels2 and 3, as well as the coefficients of detail for levels 4 and 5. The coefficients’ coloration mode has been changed,

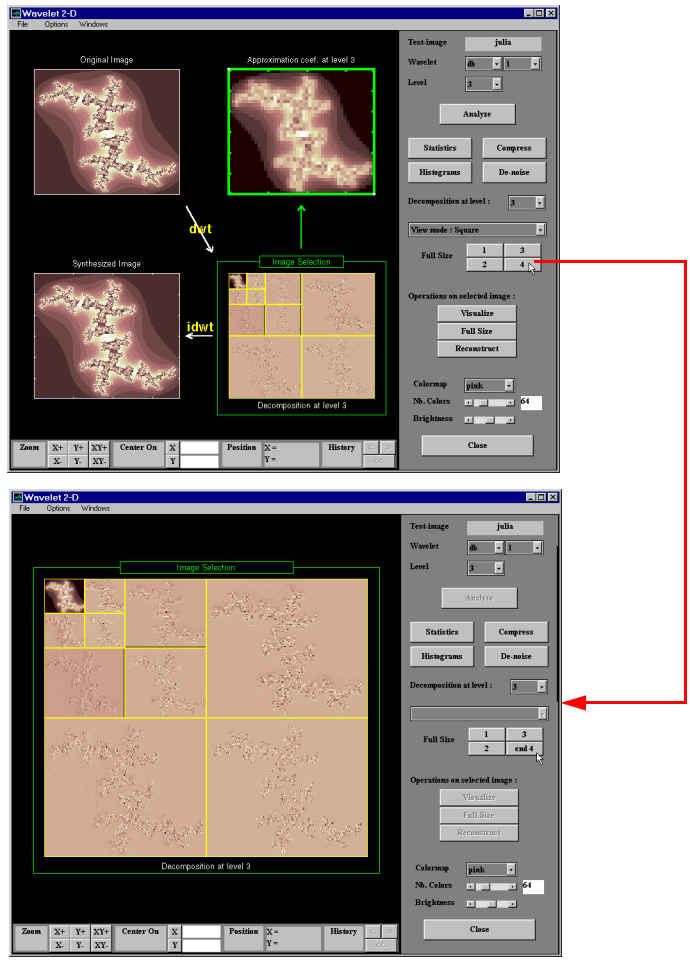
and the synthesized signal is displayed in the right hand column, rather than the original signal.



Wavelet 2-D Tool Features

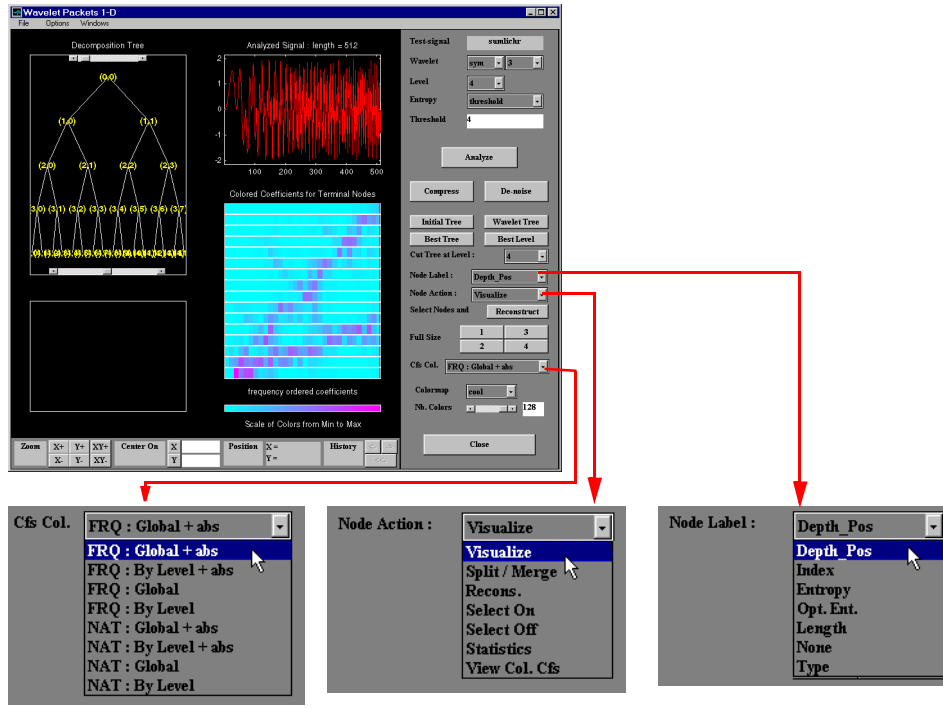
The **Wavelet 2-D Tool** has been almost completely described in the section “Two-Dimensional Analysis Using the Graphical Interface” on page 2-52.

Here is an example of an option that allows you to view a selected part of the window at a full window resolution.



Wavelet Packet Tool Features (1-D and 2-D)

The **Wavelet Packet 1-D Tool** and **Wavelet Packet2-D Tool** have been described in Chapter 5. They are almost identical in their layout and function. The only difference involves the extra coloration modes available in the **Wavelet Packet 1-D** tool, as well as the ability of the tools to work with signals or images as appropriate. Let us focus on the 1-D capabilities.



***Coefficients coloration*:**
NAT or **FRQ** is for Natural or Frequency order (see Wavelet Packet Atoms on page 101 of Chapter 6).
By level or **Global** is for a coloration made level by level or taking all detail levels.
abs is used to take the absolute values of coefficients.

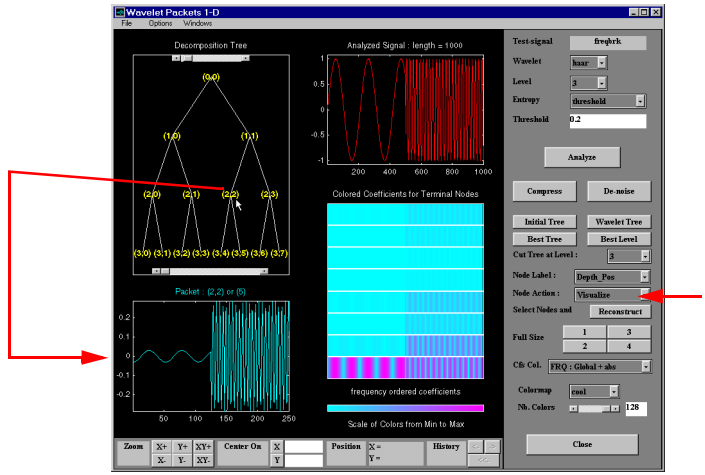
Node Action:
When you select a node in the tree, the selected option is performed. A complete description of options is provided on the next page.

Node Label:
The node labels may be changed using the pop-up menu. For example, the **Type** option labels the with (a) for approximation and (d) for detail.

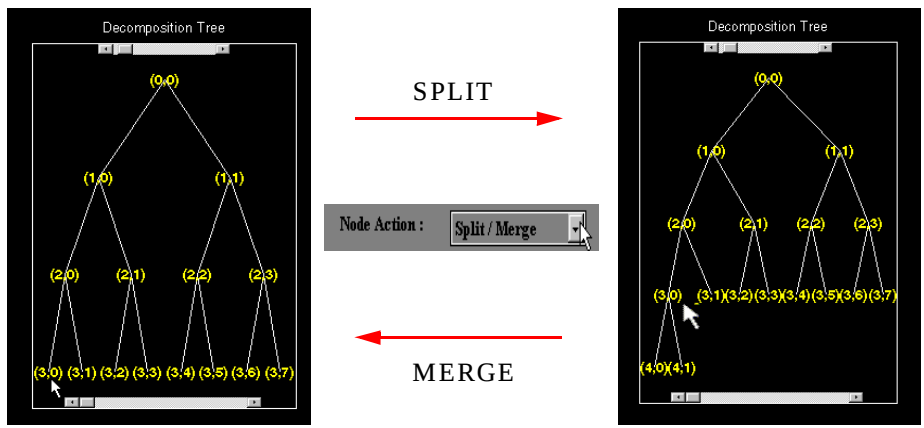
Node Action Functionality

The available options in the **Node Action** pop-up menu are:

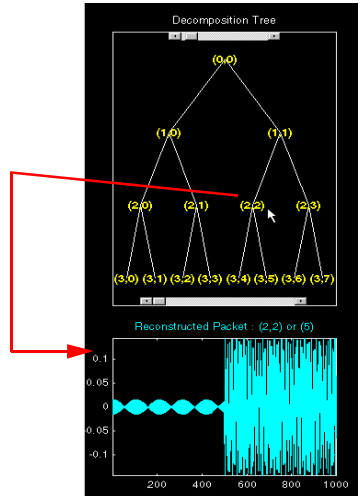
- **Visualize**: When you select a node in the wavelet packet tree the corresponding signal is displayed.



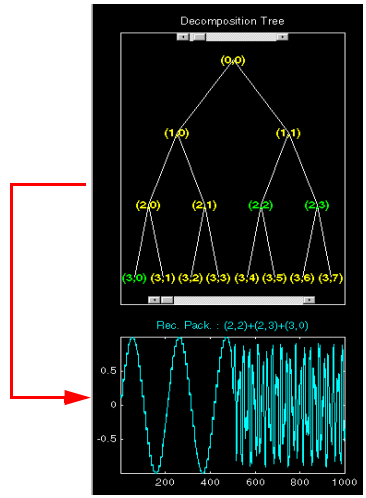
- **Split/Merge:** If a terminal node is selected it is split, growing the wavelet packet tree. Selecting other nodes, has the behavior of merging all the nodes below it in the wavelet packet tree.

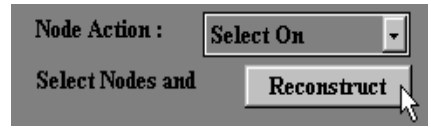


- **Recons.**: When you select a node in the wavelet packet tree, the corresponding reconstructed signal is displayed.

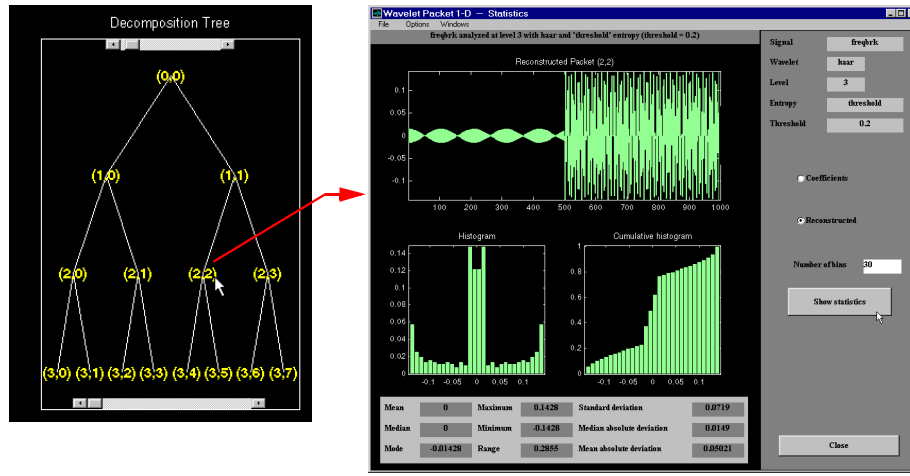


- **Select On/Off**: When **On**, you can select many nodes in the wavelet packet tree and then you can reconstruct a synthesized signal from the selected nodes using the **Reconstruct** push-button in the main window. The **Off** selection is used to unselect all the previous selected nodes.

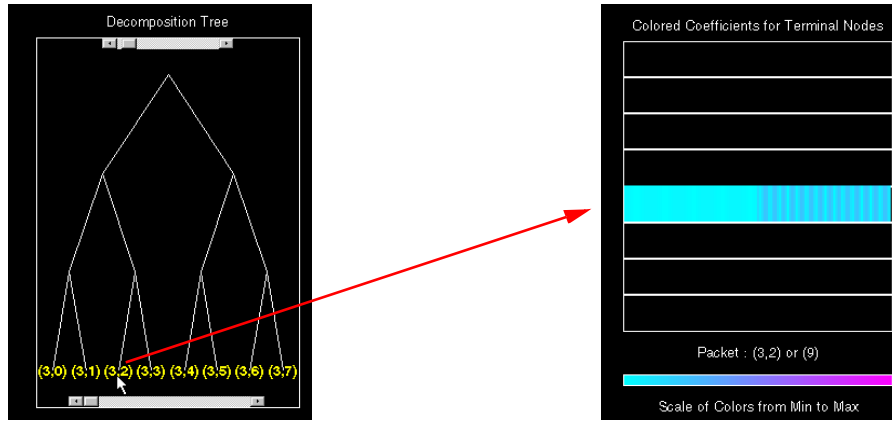




- **Statistics**: When you select a node in the wavelet packet tree, the Statistics Tool is displayed using the signal corresponding to the selected node.



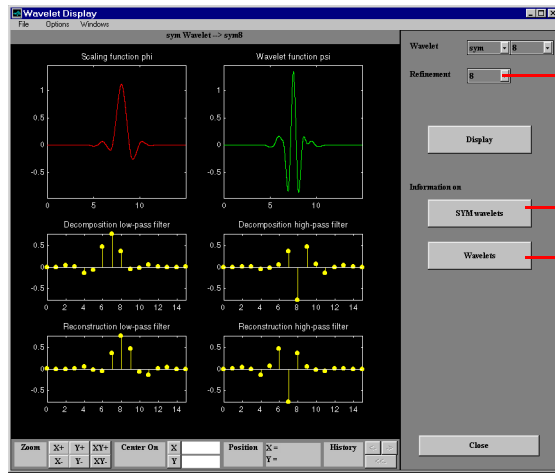
- **View Col. Cfs.**: When active, this option removes all the colored coefficients displayed and lets you redraw only the corresponding coefficients, by selecting a node in the wavelet packet tree.



Wavelet Display Tool

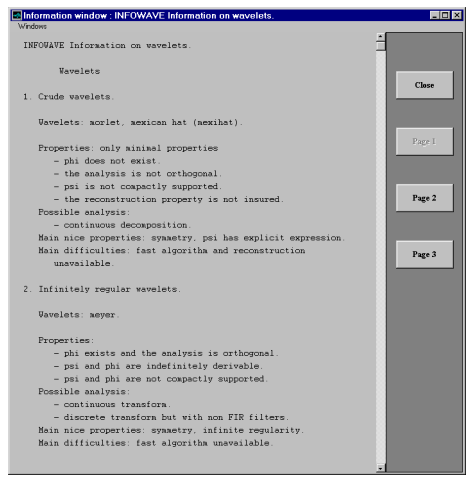
The **Wavelet Display Tool** is mentioned in the section “An Introduction to the Wavelet Families” on page 30 of Chapter 1.

Here, the main window and the associated information windows are displayed with some additional comments.

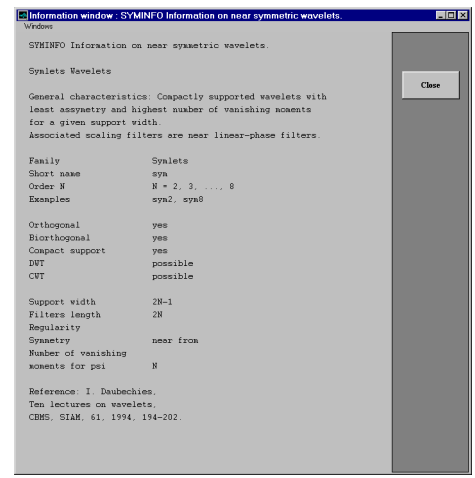


This parameter decides the precision used for the wavelet computation. Here, functions are computed over 2^8 points.

Information on all the wavelets



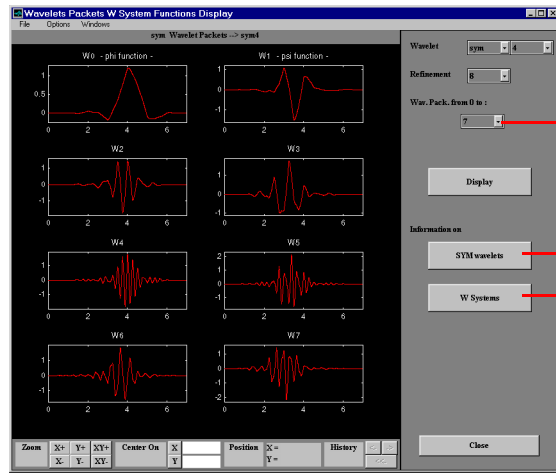
Information on the selected wavelet



Wavelet Packet Display Tool

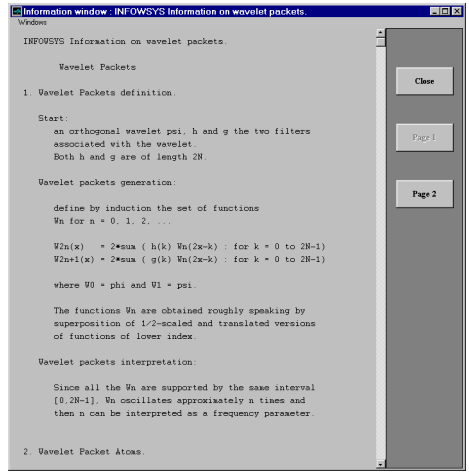
The **Wavelet Packet Display Tool** is very similar to the **Wavelet Display Tool**.

Here, the main window and the associated information windows are displayed with some additional comments.

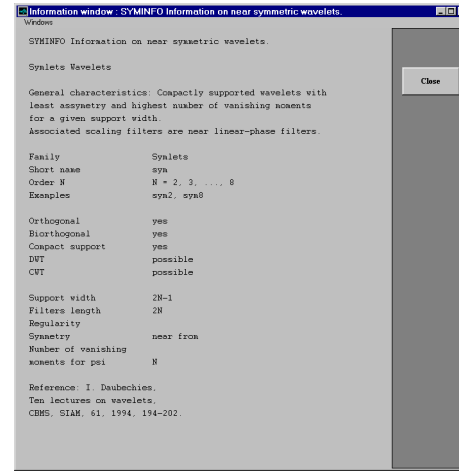


This parameter decides the precision used for the wavelet computation. Here, functions are computed over 2^8 points.

Information on wavelet packets



Information on the selected wavelet



A

B

C

D

E

F

T

U

V

W

Z