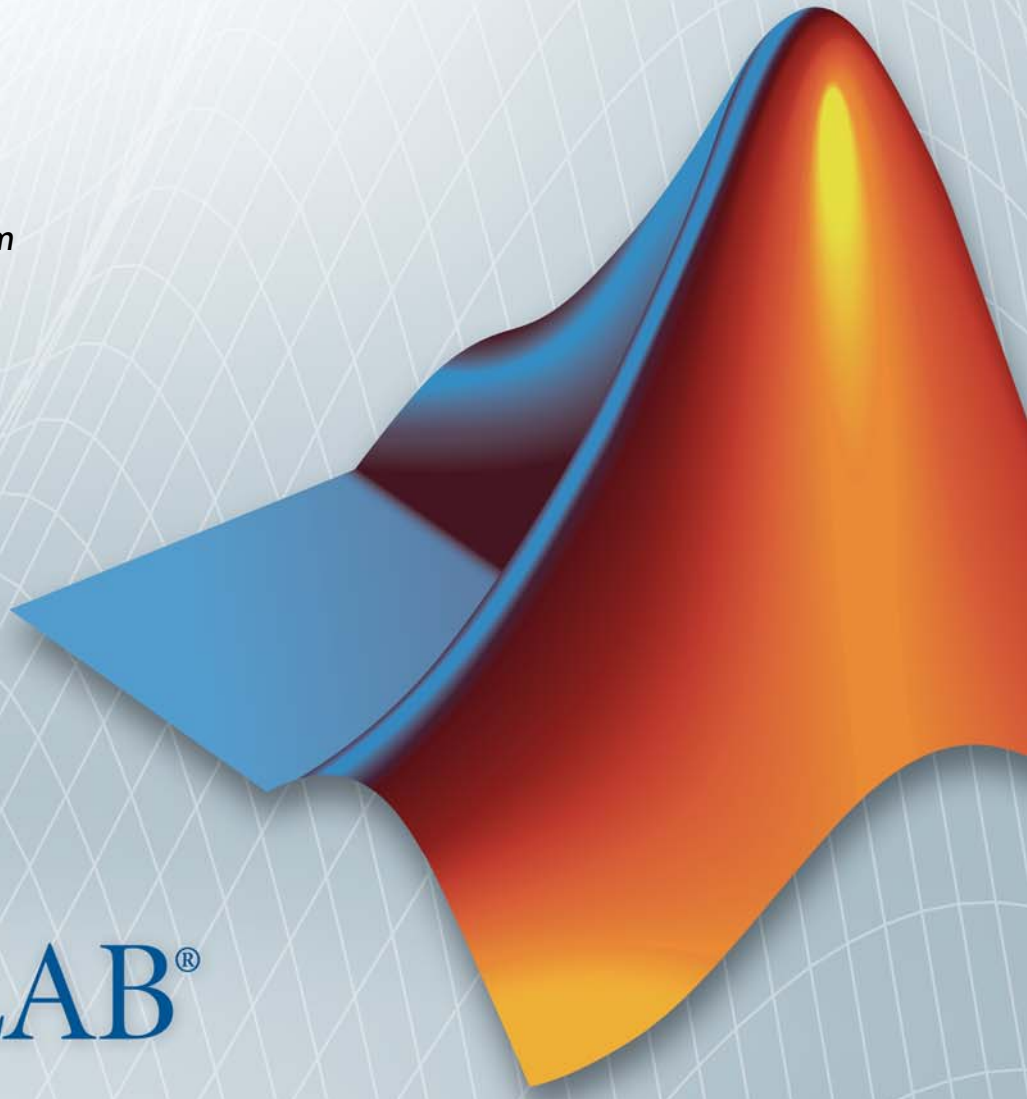
Wavelet Toolbox™

User's Guide

R2011b

Michel Misiti
Yves Misiti
Georges Oppenheim
Jean-Michel Poggi



MATLAB®

MathWorks®

How to Contact MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Wavelet Toolbox™ User's Guide

Trademarks

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

Patents

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

Revision History

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

Contents

2 Wavelets in Action: Examples and Case Studies

3 Using Wavelet Packets

4 1-D Continuous Wavelet Analysis Using Discrete Fourier Transforms (DFT)

Generating MATLAB Code from Wavelet Toolbox GUI

5

Advanced Concepts

6

7 Adding Your Own Wavelets

A GUI Reference

Object-Oriented Programming

B

Index

Acknowledgments

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

Specifically

- For the wavelet questions to Pierre-Gilles Lemarié-Rieusset (Evry) and Yves Meyer (ENS Cachan)
- For the statistical questions to Lucien Birgé (Paris 6), Pascal Massart (Paris 11) and Marc Lavielle (Paris 5)
- To David Donoho (Stanford) and to Anestis Antoniadis (Grenoble), who give generously so many valuable ideas

Colleagues and friends who have helped us steadily are Patrice Abry (ENS Lyon), 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 (Grenoble), Gérard Kerkyacharian (Paris 10), 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, and 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, Bruce Sesnovich, Jad Succari, Jane Carmody, and Paul Costa.

And finally, apologies to those we may have omitted.

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 5 University. Yves Misiti is a research engineer specializing in Computer Sciences at Paris 11 University.

The authors are members of the "Laboratoire de Mathématique" at Orsay-Paris 11 University France. Their fields of interest are statistical signal processing, stochastic processes, adaptive control, and wavelets. The authors' group, established more than 15 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

Notes by Yves Meyer

The history of wavelets is not very old, at most 10 to 15 years. The field experienced a fast and impressive start, characterized by a close-knit international community of researchers who freely circulated scientific information and were driven by the researchers' youthful enthusiasm. Even as the commercial rewards promised to be significant, the ideas were shared, the trials were pooled together, and the successes were shared by the community.

There are lots of successes for the community to share. Why? Probably because the time is ripe. Fourier techniques were liberated by the appearance of windowed Fourier methods that operate locally on a time-frequency approach. In another direction, Burt-Adelson's pyramidal algorithms, the quadrature mirror filters, and filter banks and subband coding are available.

The mathematics underlying those algorithms existed earlier, but new computing techniques enabled researchers to try out new ideas rapidly. The numerical image and signal processing areas are blooming.

The wavelets bring their own strong benefits to that environment: a local outlook, a multiscaled outlook, cooperation between scales, and a time-scale analysis. They demonstrate that sines and cosines are not the only useful functions and that other bases made of weird functions serve to look at new foreign signals, as strange as most fractals or some transient signals.

Recently, wavelets were determined to be the best way to compress a huge library of fingerprints. This is not only a milestone that highlights the practical value of wavelets, but it has also proven to be an instructive process for the researchers involved in the project. Our initial intuition generally was that the proper way to tackle this problem of interweaving lines and textures was to use wavelet packets, a flexible technique endowed with quite a subtle sharpness of analysis and a substantial compression capability. However, it was a biorthogonal wavelet that emerged victorious and at this time represents the best method in terms of cost as well as speed. Our intuitions led one way, but implementing the methods settled the issue by pointing us in the right direction.

For wavelets, the period of growth and intuition is becoming a time of consolidation and implementation. In this context, a toolbox is not only possible, but valuable. It provides a working environment that permits experimentation and enables implementation.

Since the field still grows, it has to be vast and open. The Wavelet Toolbox product addresses this need, offering an array of tools that can be organized according to several criteria:

- Synthesis and analysis tools
- Wavelet and wavelet packets approaches
- Signal and image processing
- Discrete and continuous analyses
- Orthogonal and redundant approaches
- Coding, de-noising and compression approaches

What can we anticipate for the future, at least in the short term? It is difficult to make an accurate forecast. Nonetheless, it is reasonable to think that the pace of development and experimentation will carry on in many different fields. Numerical analysis constantly uses new bases of functions to encode its operators or to simplify its calculations to solve partial differential equations. The analysis and synthesis of complex transient signals touches musical instruments by studying the striking up, when the bow meets the cello string. The analysis and synthesis of multifractal signals, whose regularity (or rather irregularity) varies with time, localizes information of interest at its geographic location. Compression is a booming field, and coding and de-noising are promising.

For each of these areas, the Wavelet Toolbox software provides a way to introduce, learn, and apply the methods, regardless of the user's experience. It includes a command-line mode and a graphical user interface mode, each very capable and complementing to the other. The user interfaces help the novice to get started and the expert to implement trials. The command line provides an open environment for experimentation and addition to the graphical interface.

In the journey to the heart of a signal's meaning, the toolbox gives the traveler both guidance and freedom: going from one point to the other, wandering from a tree structure to a superimposed mode, jumping from low to high scale, and skipping a breakdown point to spot a quadratic chirp. The time-scale graphs of continuous analysis are often breathtaking and more often than not enlightening as to the structure of the signal.

Here are the tools, waiting to be used.

Yves Meyer
Professor, Ecole Normale Supérieure de Cachan and Institut de France

Notes by Ingrid Daubechies

Wavelet transforms, in their different guises, have come to be accepted as a set of tools useful for various applications. Wavelet transforms are good to have at one's fingertips, along with many other mostly more traditional tools.

Wavelet Toolbox software is a great way to work with wavelets. The toolbox, together with the power of MATLAB® software, really allows one to write

complex and powerful applications, in a very short amount of time. The Graphic User Interface is both user-friendly and intuitive. It provides an excellent interface to explore the various aspects and applications of wavelets; it takes away the tedium of typing and remembering the various function calls.

Ingrid C. Daubechies
Professor, Princeton University, Department of Mathematics and Program in Applied and Computational Mathematics

1

Wavelet Applications

This chapter explores various applications of wavelets by presenting a series of sample analyses.

Introduction to Wavelet Analysis

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 another tool as appropriate).

3. Load the sample analysis by selecting the appropriate submenu item from **File > Example Analysis**.

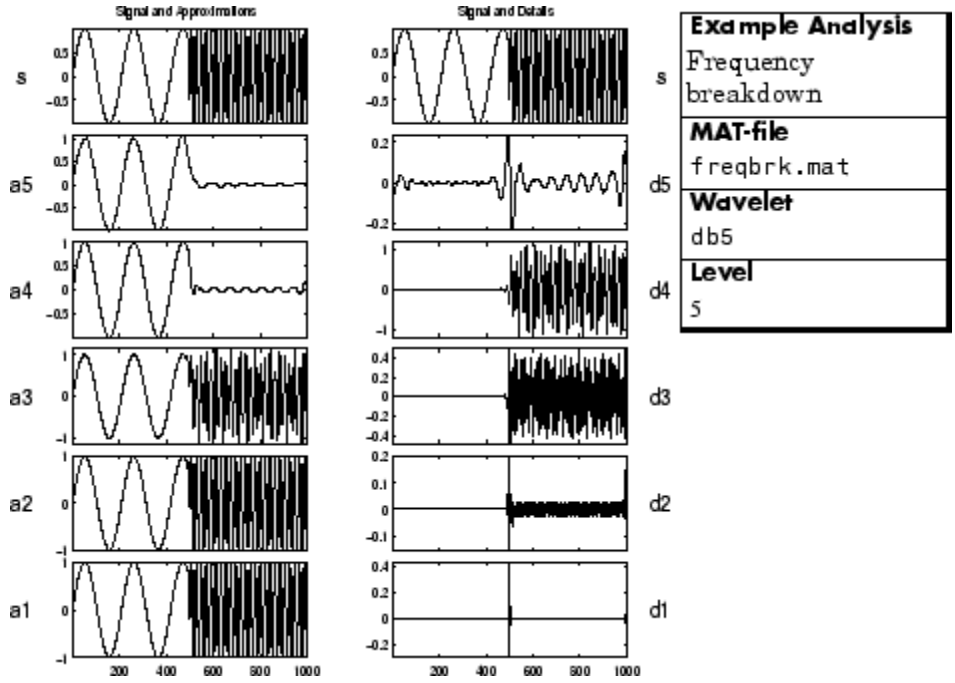
Feel free to explore on your own — use the different options provided in the graphical interface to look at different components of the signal, to compress or de-noise the signal, to examine signal statistics, or to zoom in and out on different signal features.

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

There are also other signals in the `wavedemo` folder 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.



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

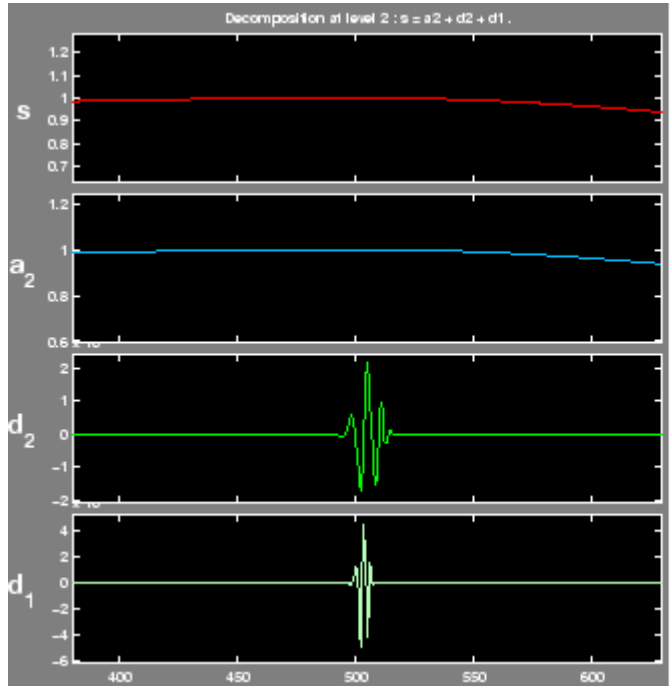
- A signal discontinuity, use the `haar` wavelet
- A rupture in the *j*-th derivative, select a sufficiently regular wavelet with at least *j* vanishing moments. (See “Detecting Discontinuities and Breakdown Points II” on page 1-6.)

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 > Example Analysis > Basic Signals > 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.



Example 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

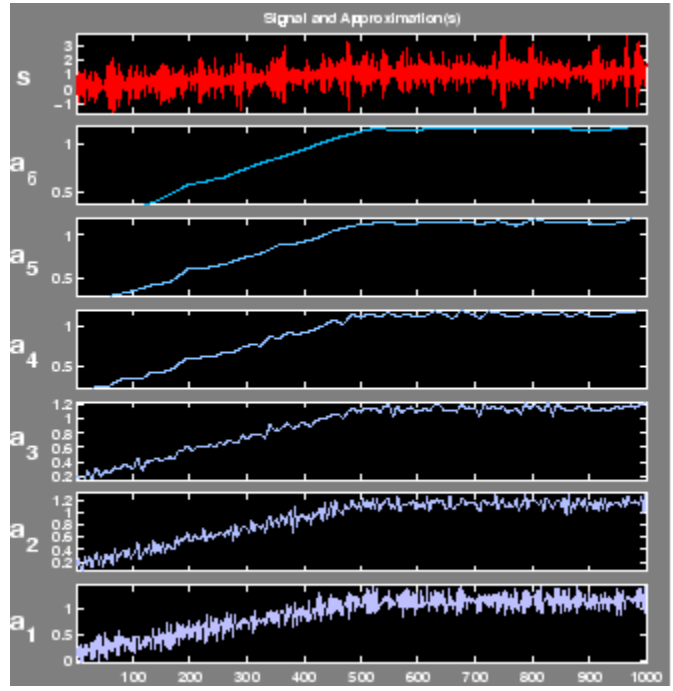
Regularity can be an important criterion in selecting a wavelet. We have chosen to use `db4`, which is sufficiently regular for this analysis. Had we chosen the `haar` wavelet, the discontinuity would not have been detected. If you try repeating this analysis using `haar` at level two, you'll notice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

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

See "Frequently Asked Questions" on page 6-62 and "Wavelet Families: Additional Discussion" on page 6-73 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.)



There is so much noise in the original signal, `s`, that its overall shape is not apparent upon visual inspection. In this level-6 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.

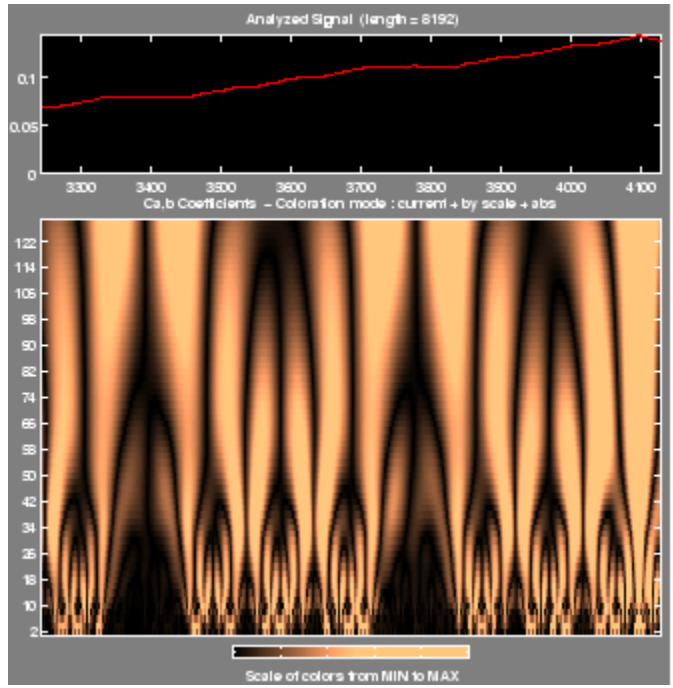
Discussion

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 > Example Analysis > Basic Signals > Step signal** (MAT-file `wstep.mat`). It is instructive to analyze this signal using the **Wavelet 1-D** tool and 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.



Example 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.

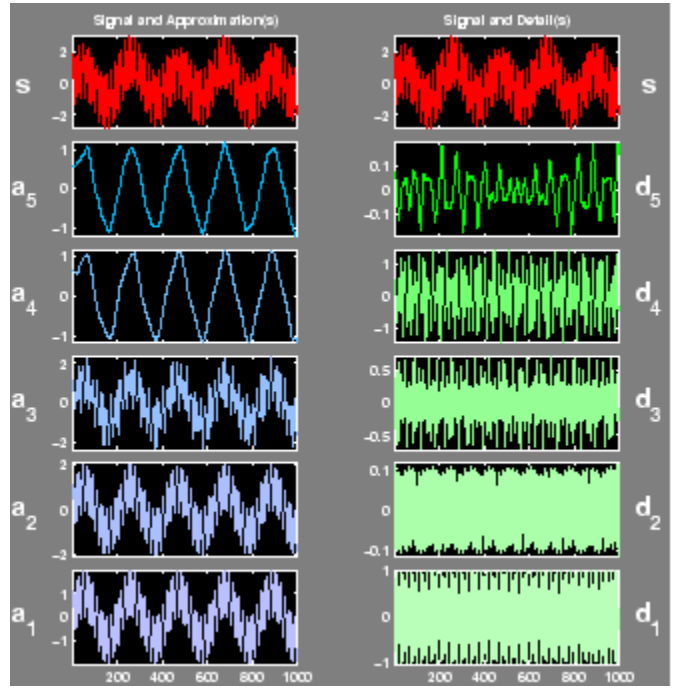
Discussion

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 called 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.



Example Analysis
Sum of sines

MAT-file
`sumsin.mat`

Wavelet
`db3`

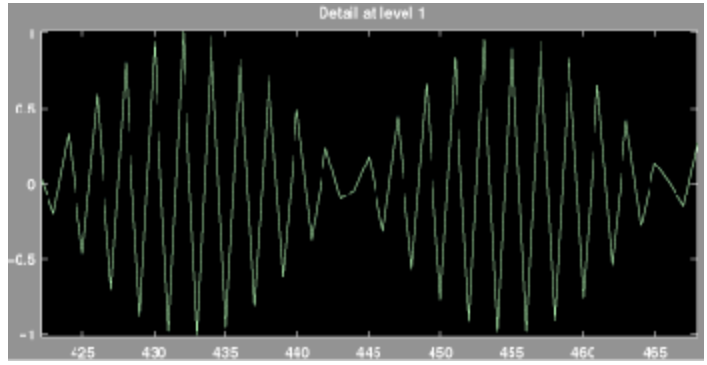
Level
5

Discussion

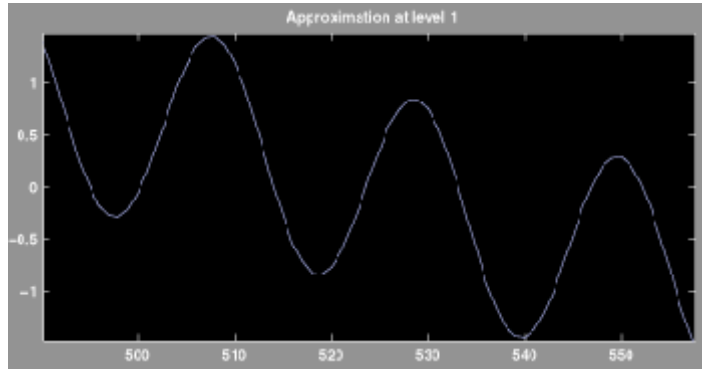
The signal is a sum of three sines: slow, medium, and rapid, which have periods (relative to the sampling period of 1) of 200, 20, and 2, respectively.

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 that 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), the 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 in "Identifying Pure Frequencies" on page 1-12) shows that the distance between two successive maximums is 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 moves 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

This also can be obtained automatically using the scal2frq function, which associates pseudo-frequencies to scales for a given wavelet.

```
lev = [1:5]; a = 2.^lev;     % scales.
wname ='db3';
delta = 1;
f = scal2frq(a,wname,delta); % corresponding pseudo-frequencies.
per  = 1./f;                 % corresponding pseudo-periods.
```

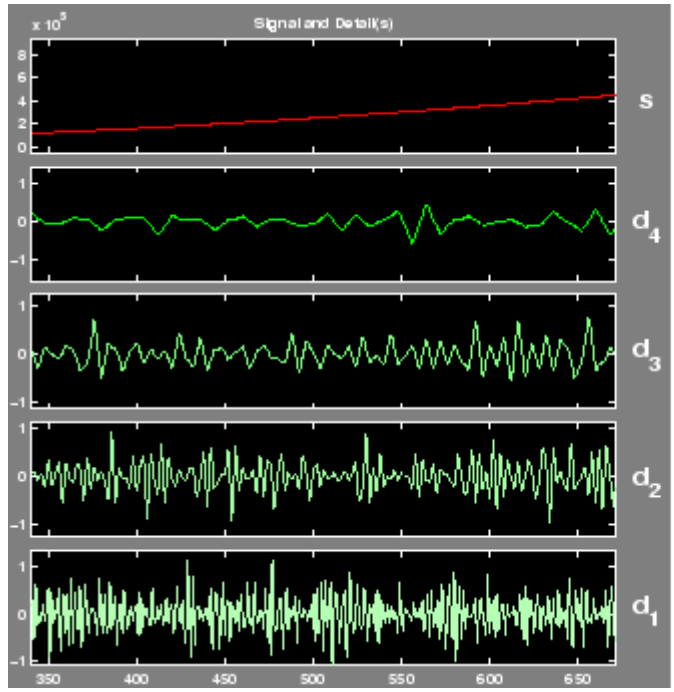
Leading to

Level	Scale	Pseudo-Period	Pseudo-Frequency
1	2	2.5	0.4
2	4	5	0.2
3	8	10	0.1
4	16	20	0.05
5	32	40	0.025

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 different scales, and each scale is taken into account 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.

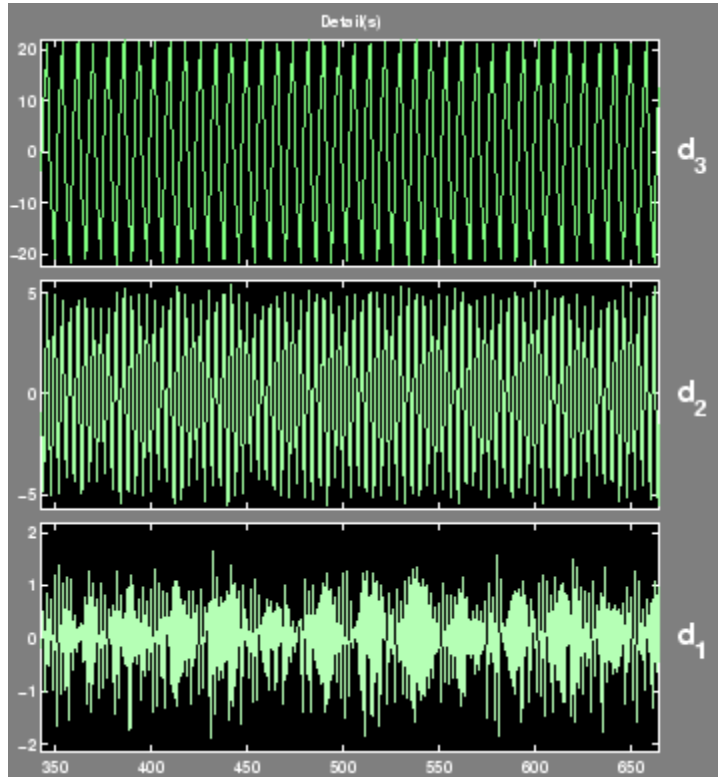


Example Analysis
Noisy polynomial
MAT-file
`noispol.mat`
Wavelet
db3
Level
4

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 the section “Daubechies Wavelets: dbN” on page 6-74.

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.

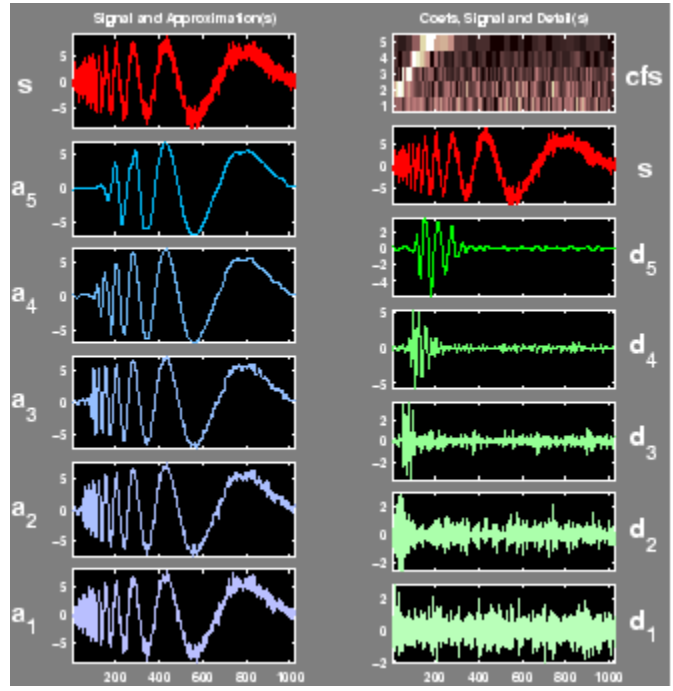
Vanishing Moments

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 the sections "Frequently Asked Questions" on page 6-62 and "Wavelet Families: Additional Discussion" on page 6-73. For the present discussion, it suffices to think of "moment" as an extension of "average." Since a wavelet's average value is zero, 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

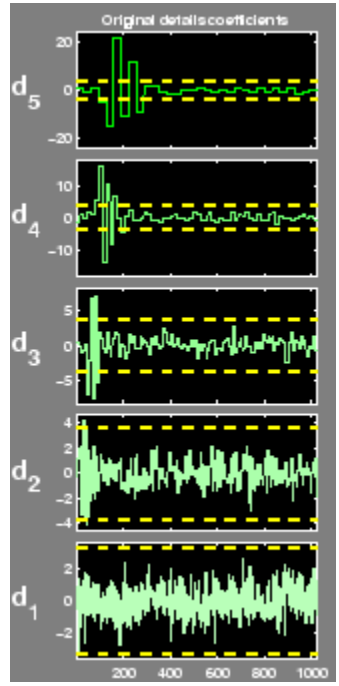
The purpose of this example is to show how to de-noise a signal using wavelet analysis. This example also gives us an opportunity to demonstrate the automatic thresholding feature of the **Wavelet 1-D** graphical interface tool. The signal to be analyzed is a Doppler-shifted sinusoid with some added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



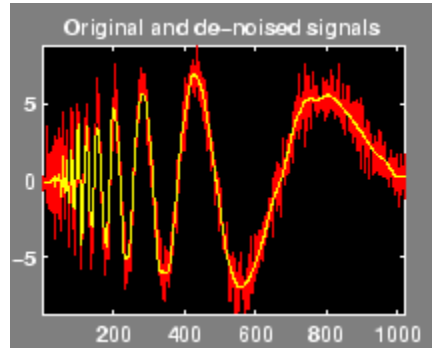
Discussion

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.



Click the **De-noise** button. On the screen, the original and de-noised signals appear superimposed in red and yellow, respectively.



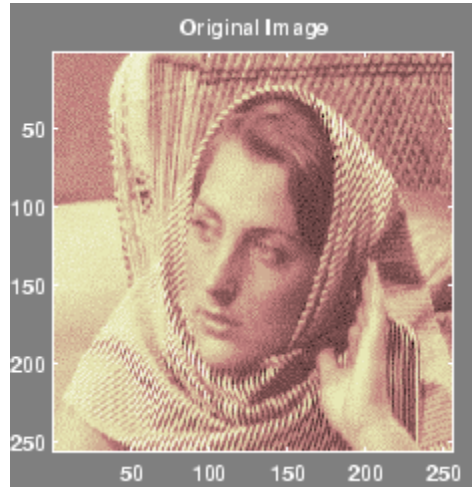
Note that the de-noised signal is flat initially. Some of the highest-frequency signal information was lost during the de-noising process, although less was lost here than in the higher level approximations A4 and A5.

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 > Example Analysis > noisdopp**.

De-Noising Images

The purpose of this example is to show how to de-noise an image using both a two-dimensional wavelet analysis and a two-dimensional stationary wavelet analysis. De-noising is one of the most important applications of wavelets.

The image to be de-noised is a noisy version of a piece of the following image.



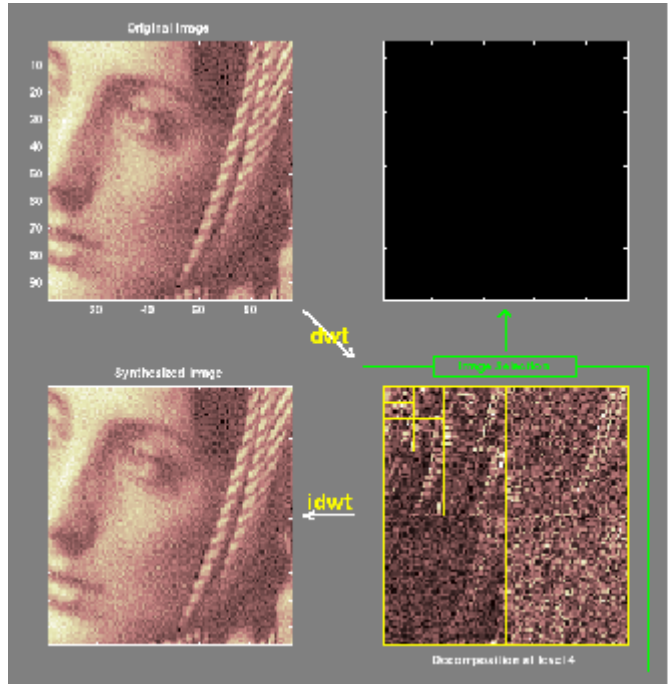
For this example, switch the extension mode to symmetric padding using the command

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

Open the **Wavelet 2-D** tool, select from the **File** menu the **Load Image** option, and select the MAT-file `noiswom.mat`, which should reside in the MATLAB folder `toolbox/wavelet/wavedemo`.

The image is loaded into the **Wavelet 2-D** tool. Select the `haar` wavelet and select **4** from the level menu, and then click the **Analyze** button.

The analysis appears in the **Wavelet 2-D** window.

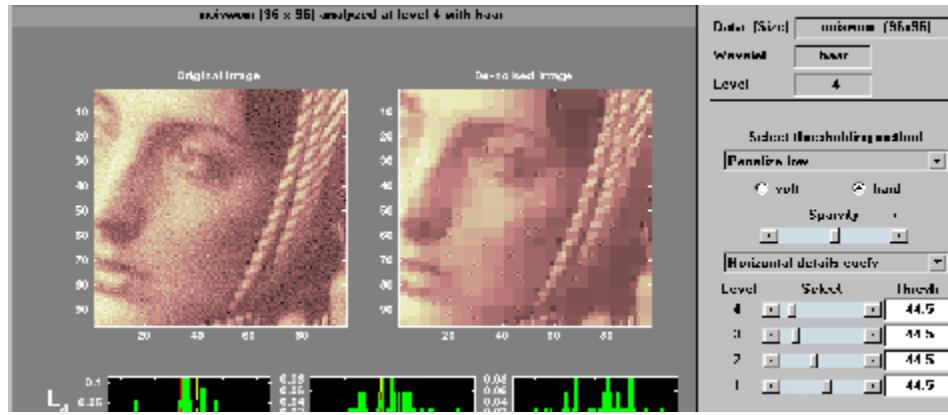


Example Analysis
Noisy Woman
MAT-file
noiswom.mat
Wavelet
haar
Level
4

Click the **De-noise** button (located at the middle right) to bring up the **Wavelet 2-D -- De-noising** window.

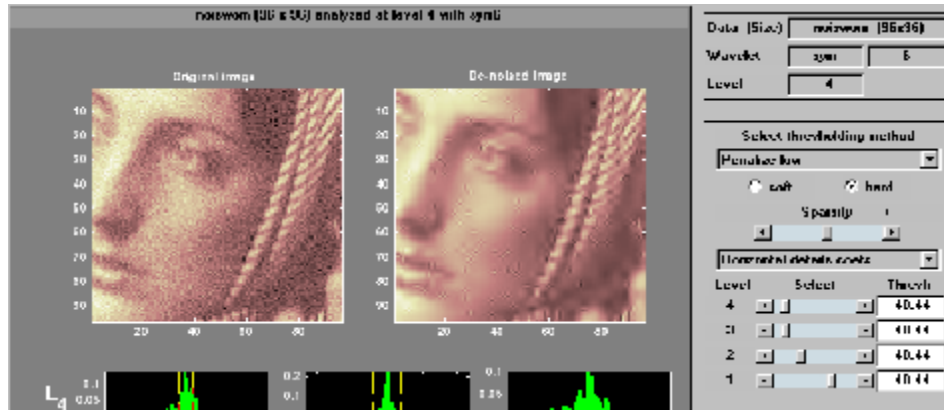
Discussion

The graphical tool provides automatically generated thresholds. From the **Select thresholding method** menu, select the item **Penalize low** and click the **De-noise** button.



The de-noised image exhibits some blocking effects. Let's try another wavelet. Click the **Close** button to go back to the **Wavelet 2-D** window. Select the `sym6` wavelet, and then click the **Analyze** button. Click the **De-noise** button to bring up the **Wavelet 2-D -- De-noising** window again.

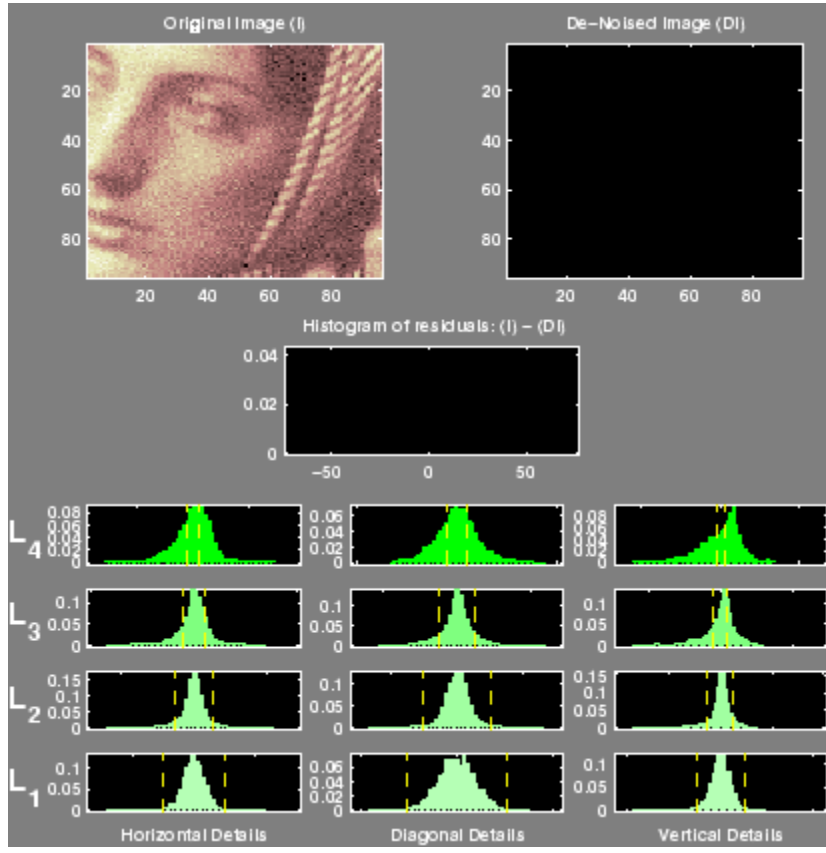
From the **Select thresholding method** menu, select the item **Penalize low**, and click the **De-noise** button.



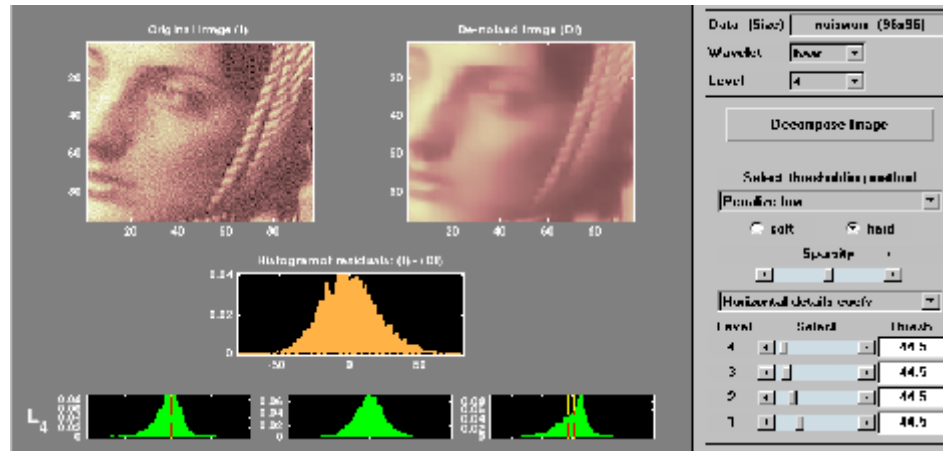
The de-noised image exhibits some ringing effects. Let's try another strategy based on the two-dimensional stationary wavelet analysis to de-noise images. The basic idea is to average many slightly different discrete wavelet analyses.

For more information, see the section “Discrete Stationary Wavelet Transform (SWT)” on page 6-45.

Click the **Close** button to go back to the **Wavelet 2-D** window and click the **Close** button again. Open the **SWT De-noising 2-D** tool, select from the **File** menu the **Load Image** option and select the MAT-file `noiswom.mat`. Select the `haar` wavelet and select **4** from the level menu, and then click the **Decompose Image** button.

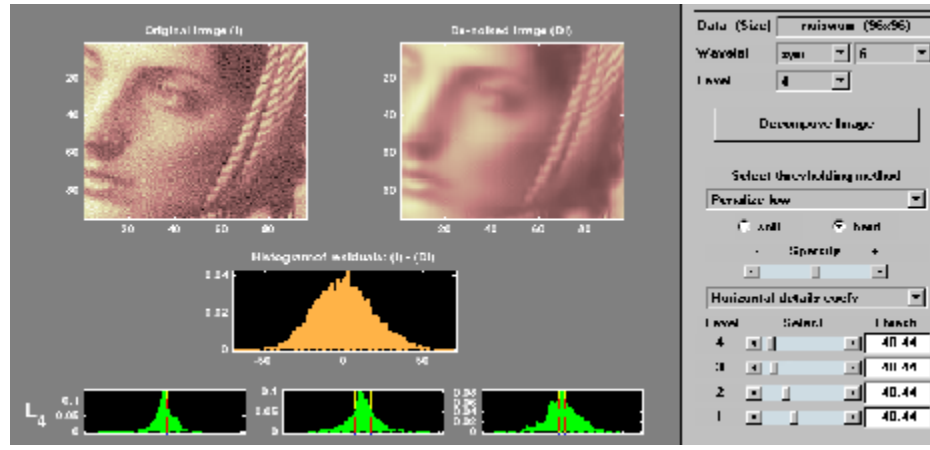


The selected thresholding method is **Penalize low**. Use the **Sparsity** slider to adjust the threshold value close to 44.5 (the same as before to facilitate the comparison with the first trial), and then click the **De-noise** button.



The result is more satisfactory. It's possible to improve it slightly.

Select the `sym6` wavelet and click the **Decompose Image** button. Use the **Sparsity** slider to adjust the threshold value close to 40.44 (the same as before to facilitate the comparison with the second trial), and then click the **De-noise** button.



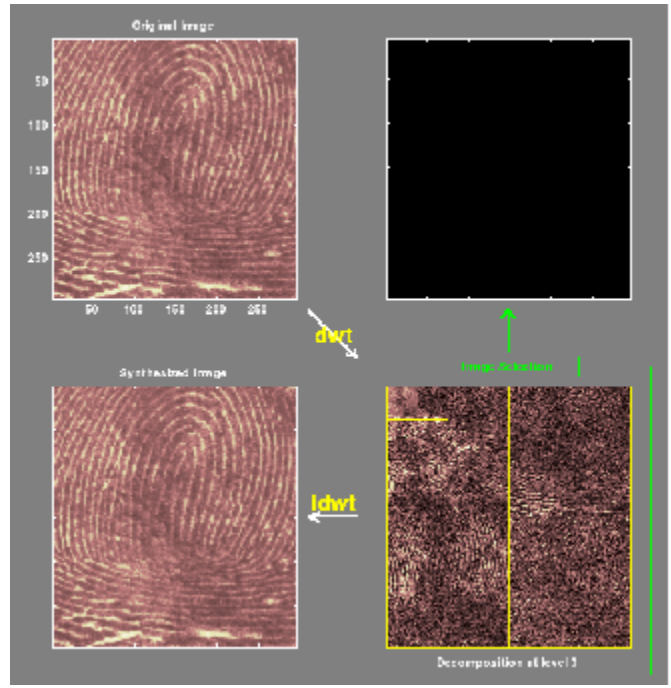
At the end of this example, turn back the extension mode to zero-padding using the command

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

Compressing Images

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 > Example Analysis > at level 3, with haar > finger**.



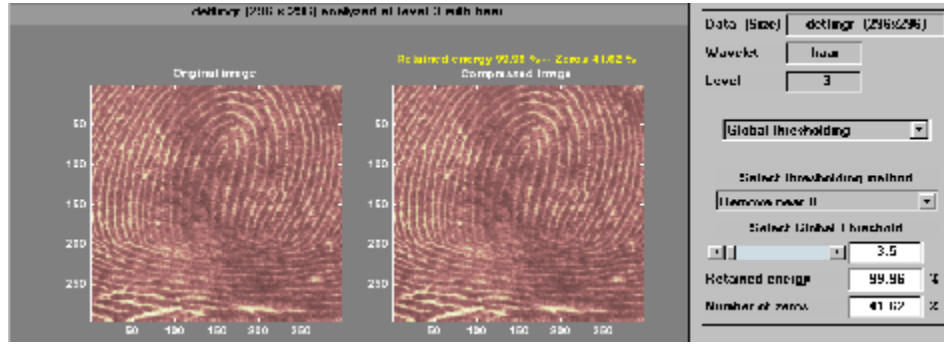
Example Analysis
Finger
MAT-file
detfingr.mat
Wavelet
haar
Level
3

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

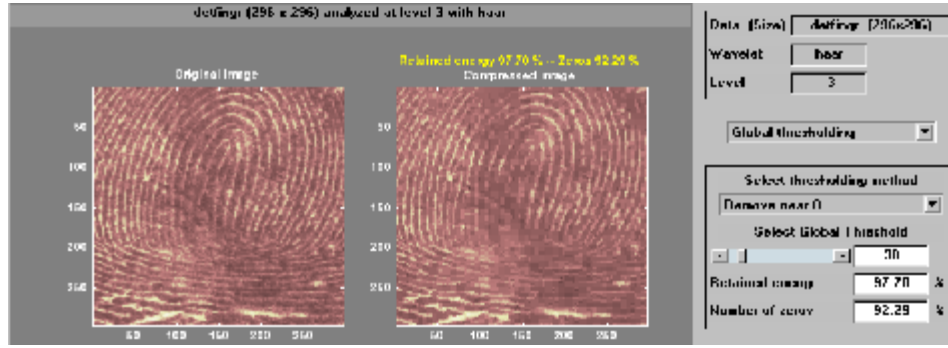
The graphical tool provides an automatically generated threshold. From the **Select thresholding method** menu, select **Remove near 0**, setting

the threshold to 3.5. Then, click the **Compress** button. Values under the threshold are forced to zero, achieving about 42% zeros while retaining almost all (99.96%) the energy of the original image.



The automatic thresholds usually achieve reasonable and various balances between the number of zeros and retained image energy. Depending on your data and your analysis criteria, you may find setting more or less aggressive thresholds achieves better results.

Here we've set the global threshold to around 30. This results in a compressed image consisting of about 92% zeros with 97.7% retained energy.



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 will be 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 the error will be large (for example, if all the coefficients have the same order of magnitude). This is the case in the second example below where m is two-dimensional Gaussian white noise. The figure in Example 1 compares for $n = 512$, the number of floating point operations (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)';
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters(wav);

% ordinary matrix multiplication by a vector.
p = m * v;

% The number of floating point operations used is 524,288

% Compute matrix approximation at level 5.
sm = m;
for i = 1:lev
    sm = dyaddown(conv2(sm,Lo_D),'c');
    sm = dyaddown(conv2(sm,Lo_D'),'r');
end

% The number of floating point operations used is 2,095,154

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

sv = v;
for i = 1:lev, sv = dyaddown(conv(sv,Lo_D)); end
sp = sm * sv;
for i = 1:lev, sp = conv(dyadup(sp),Lo_R); end
sp = wkeep(sp,length(v));

% Now, the number of floating point operations used is 9058

% 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, but applied to a new matrix *m*.

```
% Wavelet based matrix multiplication by a vector:
% a "bad" example with a randn matrix.
% Change the matrix m of example1 using:
randn('state',0);
m = randn(n,n);
```

Then, you obtain

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

rnrm =
        99.2137
```

2

Wavelets in Action: Examples and Case Studies

This chapter presents examples of wavelet decomposition. 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 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:

Example	Equation	Signal Name	MAT-File
"Example 1: A Sum of Sines" on page 2-6	A sum of sines: $s_1(t) = \sin(3t) + \sin(0.3t) + \sin(0.03t)$	`s1(t)`	`sumsin`
"Example 2: A Frequency Breakdown (Discontinuity)" on page 2-10	A frequency breakdown: $1 \le t \le 500, \quad s_2(t) = \sin(0.03t)$ $501 \le t \le 1000, \quad s_2(t) = \sin(0.3t)$	`s2(t)`	`freqbrk`
"Example 3: Uniform White Noise" on page 2-15	A uniform white noise: on the interval [–0.5 0.5]	`b1(t)`	`whitnois`
"Example 4: Colored AR(3) Noise" on page 2-17	A colored AR(3) noise: $b_2(t) = -1.5b_2(t-1) - 0.75b_2(t-2)$ $-0.125b_2(t-3) + b_1(t) + 0.5$	`b2(t)`	`warma`
"Example 5: Polynomial + White Noise" on page 2-19	A polynomial + a white noise: on the interval [1 1000] $s_3(t) = t^2 - t + 1 + b_1(t)$	`s3(t)`	`noispol`
"Example 6: A Step Signal" on page 2-21	A step signal: $1 \le t \le 500, \quad s_4(t) = 0$ $501 \le t \le 1000, \quad s_4(t) = 20$	`s4(t)`	`wstep`

Example	Equation	Signal Name	MAT-File
"Example 7: Two Proximal Discontinuities" on page 2-23	Two proximal discontinuities: $1 \le t \le 499, \quad s_5(t) = 3t$ $500 \le t \le 510, \quad s_5(t) = 1500$ $511 \le t, \quad s_5(t) = 3t - 30$	`s5(t)`	`nearbrk`
"Example 8: A Second-Derivative Discontinuity" on page 2-25	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}	`s6(t)`	`scddvbrk`
"Example 9: A Ramp + White Noise" on page 2-27	A ramp + a white noise: $1 \le t \le 499, \quad s_7(t) = \frac{3t}{500} + b_1(t)$ $500 \le t \le 1000, \quad s_7(t) = 3 + b_1(t)$	`s7(t)`	`wnoislop`
"Example 10: A Ramp + Colored Noise" on page 2-29	A ramp + a colored noise: $1 \le t \le 499, \quad s_8(t) = \frac{t}{500} + b_2(t)$ $500 \le t \le 1000, \quad s_8(t) = 1 + b_2(t)$	`s8(t)`	`cnoislop`
"Example 11: A Sine + White Noise" on page 2-31	A sine + a white noise: $s_9(t) = \sin(0.03t) + b_1(t)$	`s9(t)`	`noissin`
"Example 12: A Triangle + A Sine" on page 2-33	A triangle + a sine: $1 \le t \le 500, \quad s_{10}(t) = \frac{t-1}{500} + \sin(0.3t)$ $501 \le t \le 1000, \quad s_{10}(t) = \frac{1000-t}{500} + \sin(0.3t)$	`s10(t)`	`trsin`

Example	Equation	Signal Name	MAT-File
"Example 13: A Triangle + A Sine + Noise" on page 2-35	A triangle + a sine + a noise: $501 \le t \le 1000,$ $s_{11}(t) = \frac{1000 - t}{500} + \sin(0.3t) + b_1(t)$ $1 \le t \le 500, s_{11}(t) = \frac{t - 1}{500} + \sin(0.3t) + b_1(t)$	$s_{11}(t)$	wntrsin
"Example 14: A Real Electricity Consumption Signal" on page 2-37	A real electricity consumption signal	—	leleccum

Please note that

- All the decompositions use Daubechies wavelets.
- The examples use `wavedec` to obtain the scaling (approximation) and wavelet (detail) coefficients at various levels. For convenience, the examples use these coefficients to reconstruct the signal projections onto the appropriate approximation and detail subspaces using `wrcoef`. These projections are referred to using the shorthand notation A5 for approximation space at level 5 and D1 for detail space at level 1. In other places in the documentation, the notation A5 and D1 may refer to scaling and wavelet coefficients at level 5 and 1 respectively. The context makes clear which interpretation is correct.

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.

Complete code is provided to follow along using the command line. If you prefer to use a GUI:

1. Enter `wavemenu` at the command line. This opens the **Wavelet Toolbox Main Menu**.

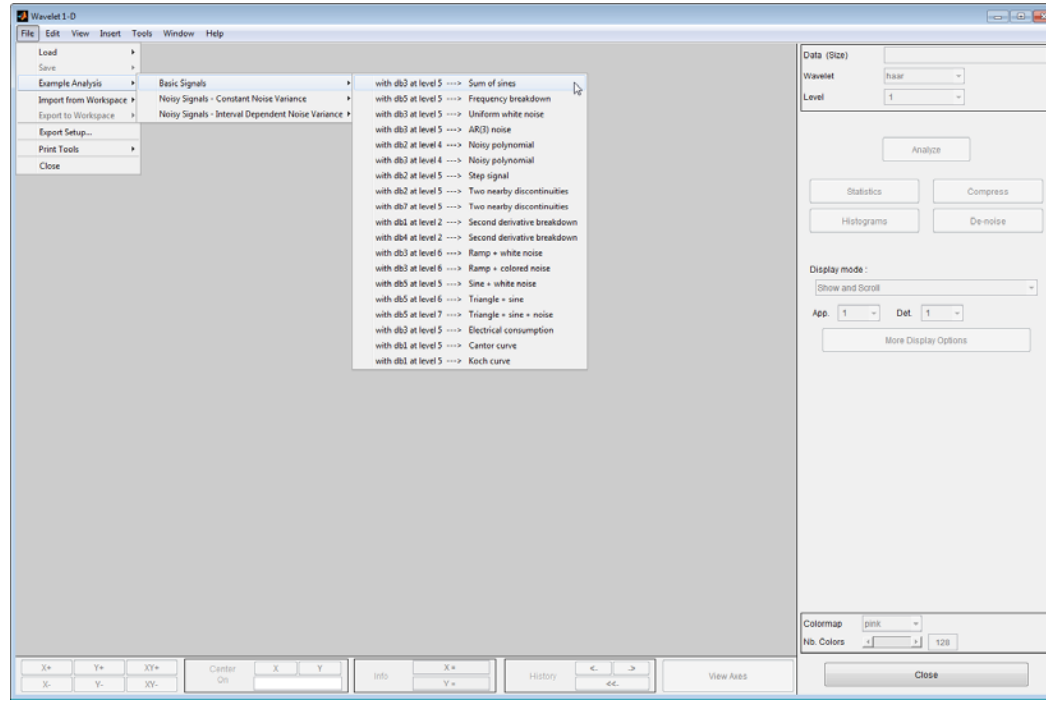
2. Select the **Wavelet 1-D** menu option to open the **Wavelet 1-D** tool.

3. From the **Wavelet 1-D** tool, choose the `File > Example Analysis` menu option and select the appropriate signal from the menu.

4. Selecting the signal performs the wavelet decomposition and displays the projections onto the approximation and detail subspaces.

   The GUI operation does not reproduce all the operations illustrated in the text using command line functionality.

The following figure illustrates the signal selection operation within the **Wavelet 1-D** tool.



Example 1: A Sum of Sines

Analyzing wavelet: *db3*

Decomposition levels: *5*

The signal in `sumsin.mat` is the sum of three sine waves with frequencies of 0.03 radians/sample, 0.3 radians/sample, and 3 radians/sample. If you assume the sampling interval is 1 second, the three frequencies correspond to 0.0048, 0.0477, and 0.4775 cycles/second (Hz). The corresponding periods are approximately 200, 20, and 2 samples respectively.

Let Δt denote the sampling interval. The wavelet coefficients at scale J approximate a bandpass filtering of the input data in the interval:

$$[\frac{1}{2^{J+1}\Delta t}, \frac{1}{2^{J}\Delta t}]$$

The scaling, or approximation, coefficients at scale J approximate a bandpass filtering of the input data in the interval:

$$[0, \frac{1}{2^{J+1}\Delta t}]$$

The frequency localization depends on the analyzing wavelet and should be viewed as an approximation. Based on this bandpass approximation, you should find the highest frequency sine wave localized in the finest scale (D1) wavelet coefficients, the 0.0477 Hz component in the level 4 details (D4), and the 0.0048 Hz component in the level 5 approximation coefficients (A5).

To obtain the wavelet decomposition of the data in `sumsin.mat` down to level using the `db3` wavelet use `wavedec`.

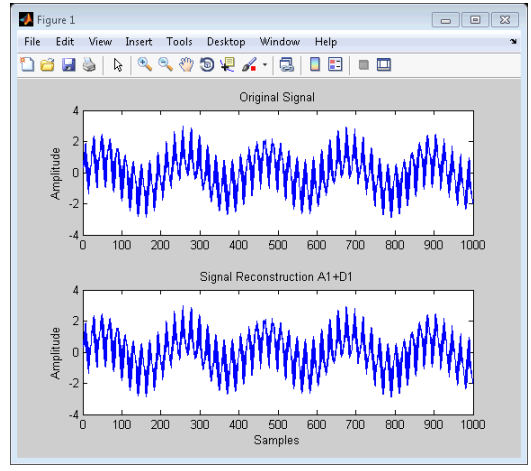
```
load sumsin
[Coeffs,L] = wavedec(sumsin,5,'db3');
```

The following loop extracts the detail and approximation coefficients at levels 1 to 5 and uses `wrcoef` to compute orthogonal projections of the signal onto the corresponding subspaces.

```
Dproject = zeros(length(sumsin),5);
Aproject = zeros(length(sumsin),5);

for J = 1:5
Dproject(:,J) = wrcoef('d',Coeffs,L,'db3',J);
Aproject(:,J) = wrcoef('a',Coeffs,L,'db3',J);
end
```

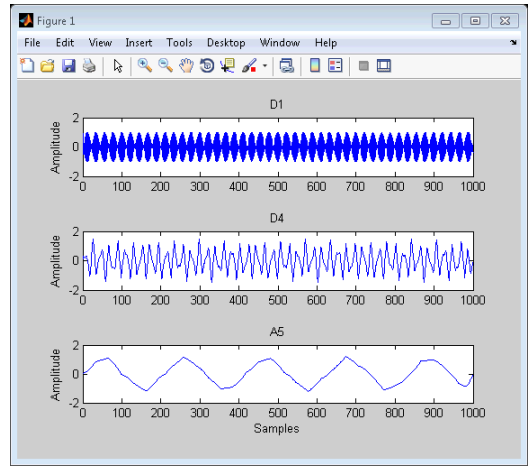
At each level, the orthogonal projections onto the approximation (scaling) and detail (wavelet) spaces constitute a decomposition of the signal lowpass approximation into a direct sum of orthogonal subspaces. This means that the sum of the first columns of `Aproject` and `Dproject` constitute an orthogonal decomposition of the original signal, S=A1+D1.

```
sigdecomp = Aproject(:,1)+Dproject(:,1);
subplot(211)
plot(sumsin);
title('Original Signal'); ylabel('Amplitude');
subplot(212);
plot(sigdecomp);
title('Signal Reconstruction A1+D1'); ylabel('Amplitude');
xlabel('Samples');
% l2 norm of the difference between the signal and
% A1+D1
norm(sumsin'-sigdecomp,2)
```



Plot the projections onto the D1, D4, and A5 subspaces. These projections approximately localize the oscillations with periods of 2, 20, and 200 samples respectively.

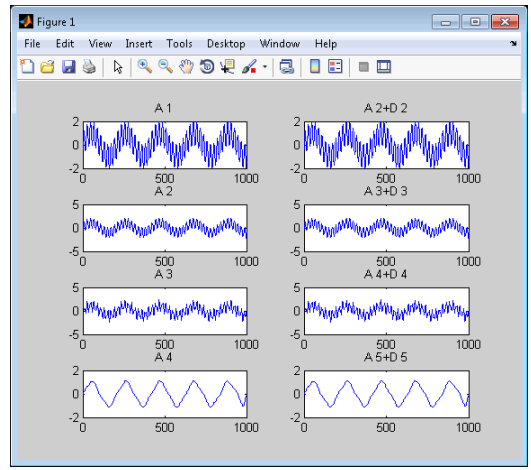
```
subplot(311)
plot(Dproject(:,1)); title('D1');
ylabel('Amplitude');
subplot(312);
plot(Dproject(:,4)); title('D4');
ylabel('Amplitude');
subplot(313);
plot(Aproject(:,5)); title('A5');
xlabel('Samples'); ylabel('Amplitude');
```



Finally, the following code produces a plot to demonstrate that the lowpass approximation at one level higher in resolution is the direct sum of the lowpass (scaling) and highpass (wavelet) projections from one level lower in resolution.

```
N = 1;
for J = 2:5
    subplot(4,2,N)
    plot(Aproject(:,J-1));
    title(['A ', num2str(J-1)]);
    subplot(4,2,N+1)
    plot(Aproject(:,J)+Dproject(:,J));
```

```
    title(['A ', num2str(J) '+D ', num2str(J)]);
    N = N+2;
end
```



The left column of the preceding plot shows the projection onto the approximation space at level J, while the right column shows the sum of the projections onto the approximation and detail spaces at level J-1.

Example 1: A Sum of Sines	
Addressed topics	• Identifying pure frequencies • The effect of a wavelet on a sine wave • Projections onto approximation (scaling) and detail (wavelet) subspaces • Bandpass nature of wavelet and scaling filters
Further exploration	• Compare with a Fourier analysis. • Change the frequencies. Analyze other linear combinations.

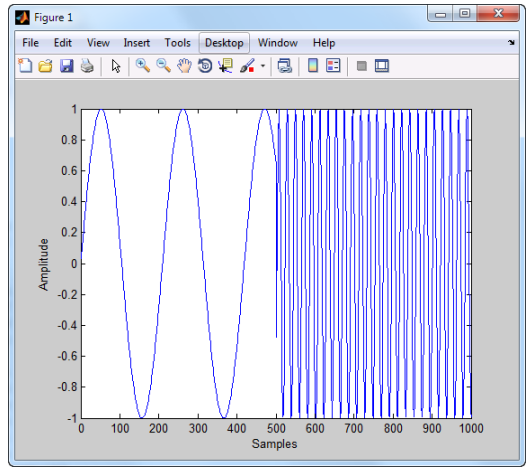
Example 2: A Frequency Breakdown (Discontinuity)

Analyzing wavelet: *db5*

Decomposition levels: *5*

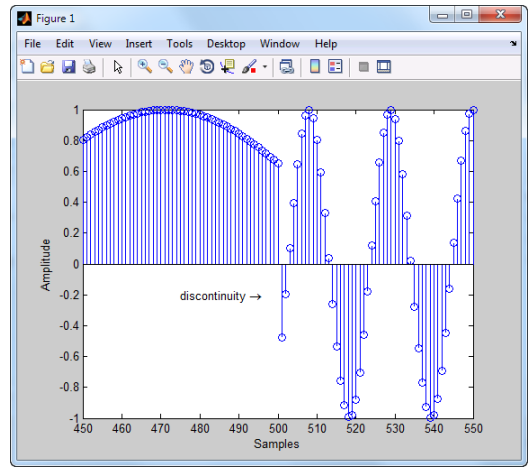
The signal consists of two sine waves defined on disjoint intervals. The signal has a length of 1000 samples. The first 500 samples contain a sine wave with a frequency of 0.03 radians/sample. Samples 501–1000 contain a sine wave with a frequency of 0.3 radians/sample. If you assume a sampling interval, Δt, equal to 1, these frequencies are 0.0048 and 0.0477 Hz.

```
load freqbrk
plot(freqbrk)
xlabel('Samples'); ylabel('Amplitude');
```



The signal is not continuous at the frequency transition because the left-hand and right-hand limits are not equal. You can see this by zooming in on the frequency transition and using a stem plot.

```
stem(450:550,freqbrk(450:550)); xlabel('Samples');
ylabel('Amplitude');
text(475,-0.2,'discontinuity \rightarrow');
```



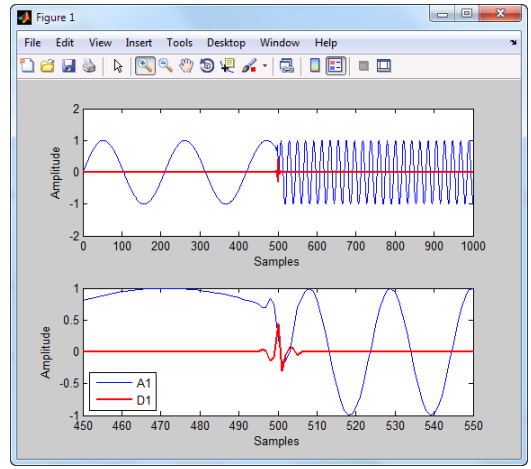
A general strength of wavelets is the detection of abrupt changes in the signal. These transitions are best localized at fine scales where the length of the wavelet filters is the smallest. To illustrate this, decompose the signal down to level 5 with the `db5` wavelet using `wavedec`. Compute the projections of the signal onto the approximation and detail spaces using `wrcoef`.

```
[Coeffs,L] = wavedec(freqbrk,5,'db5');
Dproject = zeros(length(freqbrk),5);
Aproject = zeros(length(freqbrk),5);

for J = 1:5
Dproject(:,J) = wrcoef('d',Coeffs,L,'db5',J);
Aproject(:,J) = wrcoef('a',Coeffs,L,'db5',J);
end
```

Plot the projection onto the level-one approximation space and overlay the projection onto the level-one detail space to visualize how the wavelet coefficients isolate the discontinuity.

```
subplot(211)
plot(Aproject(:,1)); xlabel('Samples'); ylabel('Amplitude');
hold on;
plot(Dproject(:,1),'r','linewidth',2);
subplot(212);
plot(Aproject(:,1)); xlabel('Samples'); ylabel('Amplitude');
```

```
hold on;
plot(Dproject(:,1),'r','linewidth',2);
axis([450 550 -1 1]);
legend('A1','D1','Location','SouthWest');
```



The signal projection onto D1 is zero except in the neighborhood of the discontinuity. If you are only interested in localizing a discontinuity as accurately as possible, the length of the wavelet filter is a critical consideration. The shorter the wavelet filter, the more precise the localization of a discontinuity in the time or space domain, but the poorer the frequency localization properties. Accordingly, the `db1` wavelet outperforms the `db5` wavelet in the localization of the discontinuity, but the `db5` wavelet outperforms the `db1` wavelet in segregation of the sinusoidal components.

To localize the sinusoidal components, let Δt denote the sampling interval. The wavelet coefficients at scale J approximate a bandpass filtering of the input data in the interval:

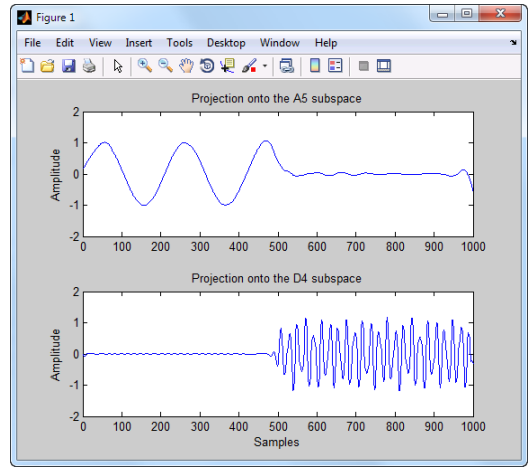
$$[\frac{1}{2^{J+1}\Delta t},\frac{1}{2^{J}\Delta t}]$$

The scaling, or approximation, coefficients at scale J approximate a bandpass filtering of the input data in the interval:

$$[0, \frac{1}{2^{J+1}\Delta t}]$$

Based on the preceding result, you expect D4 to isolate the sine wave at 0.0477 Hz and A5 to contain the sine wave at 0.0048 Hz.

```
subplot(211)
plot(Aproject(:,5)); ylabel('Amplitude');
title('Projection onto the A5 subspace');
subplot(212);
plot(Dproject(:,4)); xlabel('Samples');
ylabel('Amplitude');
title('Projection onto the D4 subspace');
```



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, select a deeper level of decomposition in such a way that the slow sinusoid appears into the details. • Compare with a Fourier analysis. • Compare with a windowed Fourier analysis.

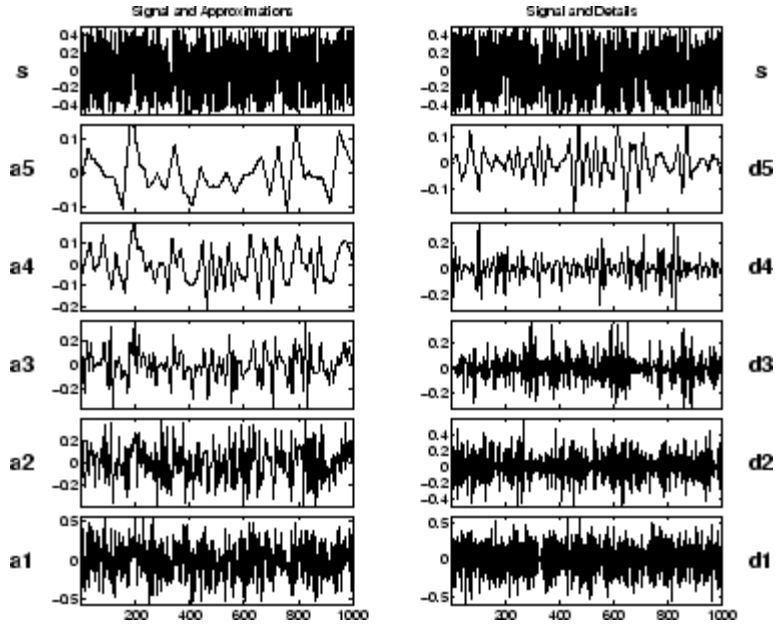
Example 3: Uniform White Noise

Analyzing wavelet: *db3*

Decomposition levels: *5*

At all levels we encounter noise-type signals that are clearly irregular. This is because 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 noise, and that these signals are increasingly interdependent as the resolution decreases. On the other hand, the wavelet coefficients are random, noncorrelated variables. This property is not evident on the reconstructed signals shown here, but it can be guessed at from the representation of the coefficients.



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
Further exploration	• Compare the frequencies included in the details with those in the approximations. • 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.

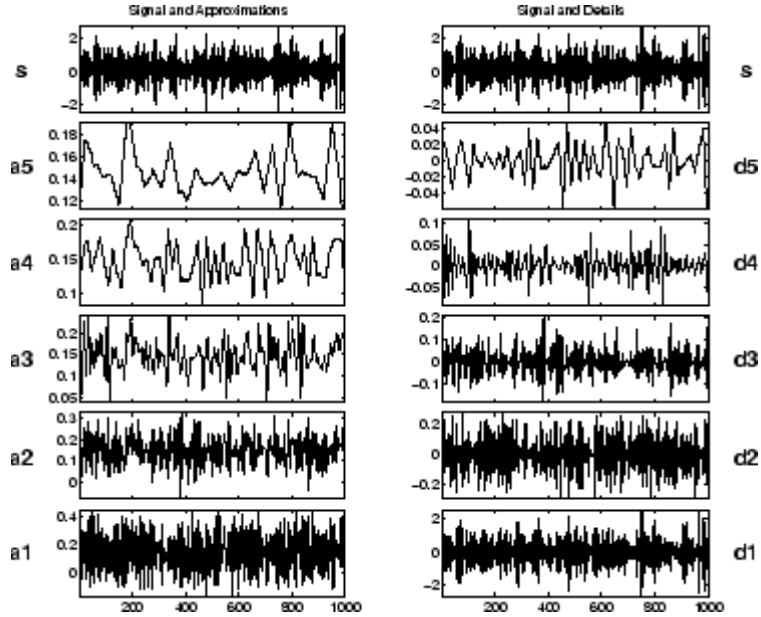
Example 4: Colored AR(3) Noise

Analyzing wavelet: *db3*

Decomposition levels: *5*

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

This figure can be examined in view of "Example 3: Uniform White Noise" on page 2-15, since we are confronted here with a nonwhite noise whose spectrum is mainly at the higher frequencies. Therefore, it is 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 in "Example 3: Uniform White Noise" on page 2-15), the noise from which b_2 is built using linear filtering. (b_1 and b_2 are defined explicitly in "Illustrated Examples" on page 2-2, Examples 3 and 4.)



Example 4: Colored AR(3) Noise	
Addressed topics	• Processing noise • The relative importance of different details • The relative importance of D_1 and A_1
Further exploration	• Compare the detail frequencies with those in the approximations. • Compare approximations A_3, A_4, and A_5 with those shown in "Example 3: Uniform White Noise" on page 2-15. • Replace AR(3) with an ARMA (AutoRegressive Moving Average) model noise. For instance, $$b_3(t) = -15b_3(t-1) - 0.75b_3(t-2) - 0.125b_3(t-3) + b_1(t) - 0.7b_1(t-1)$$ • 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

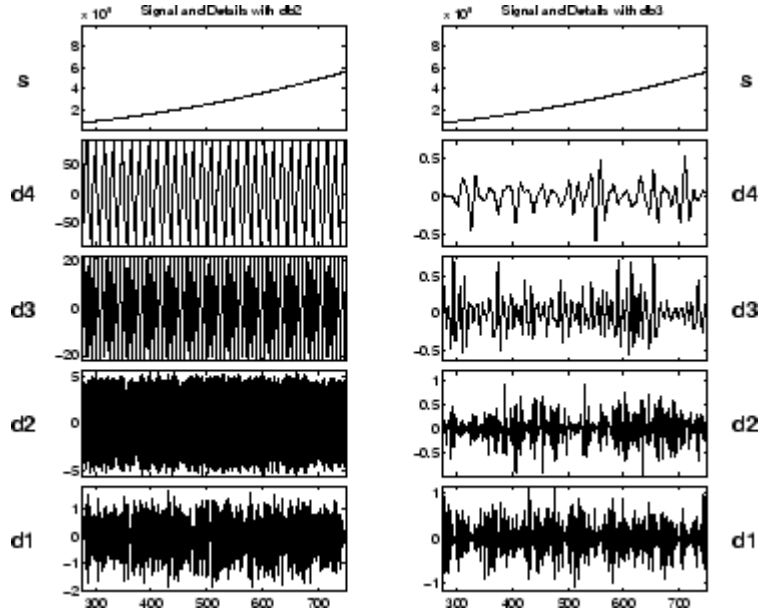
Analyzing wavelets: *db2* and *db3*

Decomposition levels: *4*

The purpose of this analysis is to illustrate the property that 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 because the detail for level `j` takes into account that the fluctuations of the function around its mean value on dyadic intervals 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.



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`.
Further exploration	• Increase noise intensity and repeat the analysis.

Example 6: A Step Signal

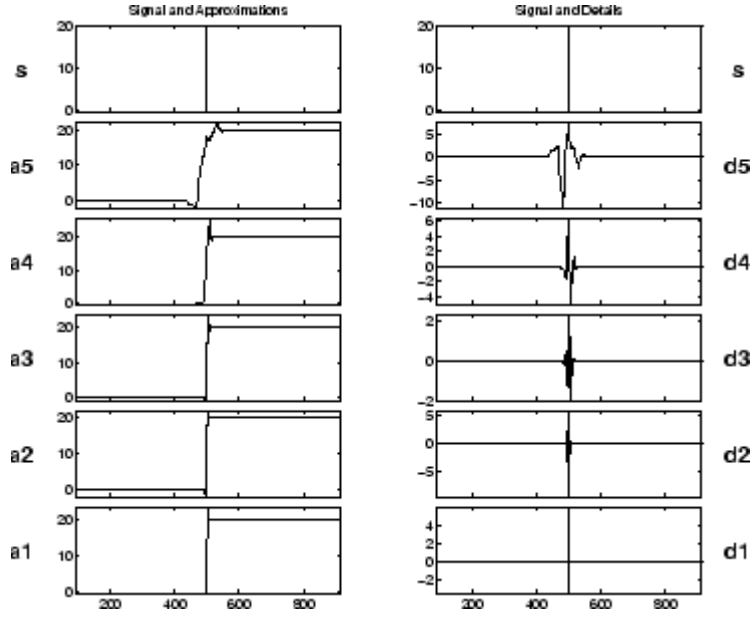
Analyzing wavelet: *db2*

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.

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



Example 6: A Step Signal	
Addressed topics	• Detecting breakdown points • Suppressing signals • Detecting long-term evolution • Identifying 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

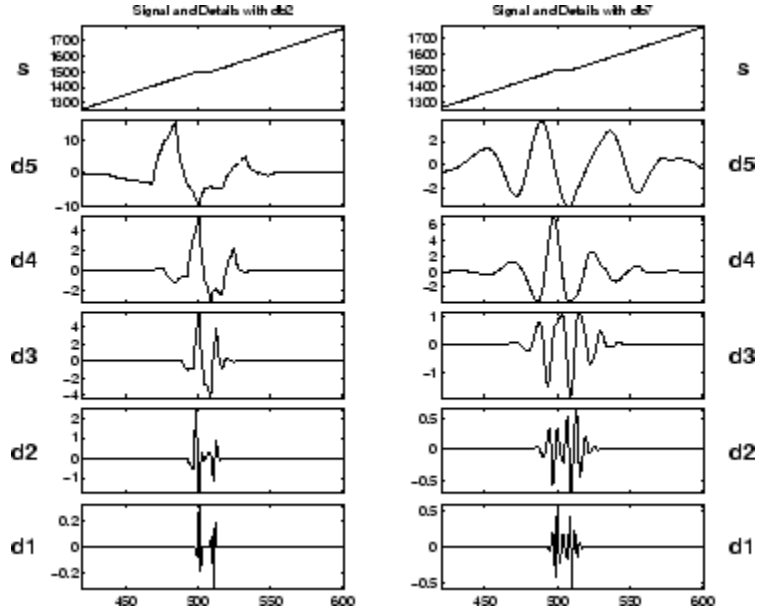
Analyzing wavelet: *db2* and *db7*

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`, shown on the left side of the figure); and the other on a wavelet having a longer filter (`db7`, shown on the right side of the figure).

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 be 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.



Example 7: Two Proximal Discontinuities	
Addressed topics	• Detecting breakdown points
Further exploration	• Move the discontinuities closer together and further apart. • 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 wavelets: *db1* and *db4*

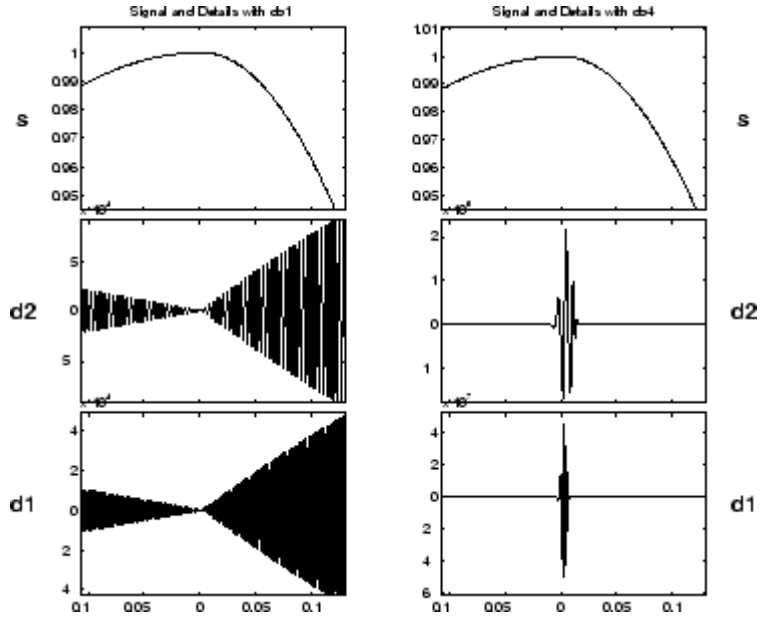
Decomposition levels: *2*

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: `db1`, which is insufficiently regular (shown on the left side of the figure); and `db4`, which is sufficiently regular (shown on the right side of the figure).

Looking at the figure on the left, 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 `db1` and 10^{-6} 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` (right side of the figure), 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, notice that the selected wavelet must be sufficiently regular, which implies a longer filter impulse response to detect the singularity.

Note To produce the figure below you can use the One-Dimensional Wavelet GUI Tool. Type `wavemenu` at the MATLAB prompt and click **Wavelet 1-D**. Then, select **File > Example Analysis > Basic Signals > with db1 at level 2 > Second Derivative Breakdown** (and **... with db4 ...**). Detail values are very small, so to get the same shapes you must zoom the *y*-axis many times (close to 10^{-3} for `db1` and 10^{-6} for `db4`).



Example 8: A Second-Derivative Discontinuity	
Addressed topics	• Detecting breakdown points • Suppressing signals • Identifying a difficult discontinuity • Carefully selecting a wavelet to reveal an effect
Further exploration	• Calculate the detail values for the Haar wavelet. • Be aware 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*

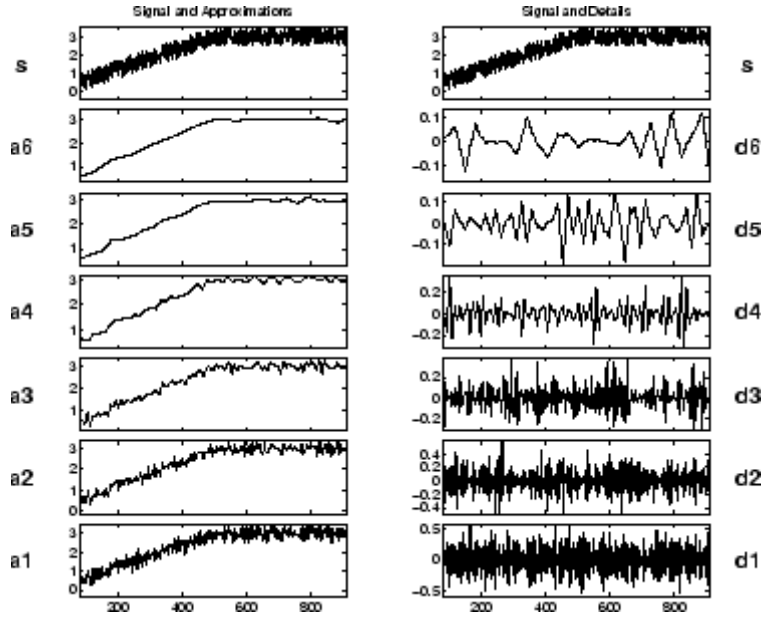
Decomposition levels: *6*

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).

Looking at the figure, in the chart 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.



Example 9: A Ramp + White Noise	
Addressed topics	• Detecting breakdown points • Processing noise • Detecting long-term evolution • Splitting signal components • Identifying noises and approximations
Further exploration	• Compare with the white noise $b_1(t)$ shown in "Example 3: Uniform White Noise" on page 2-15. • Identify the number of levels needed to suppress the noise almost entirely. • Change the noise.

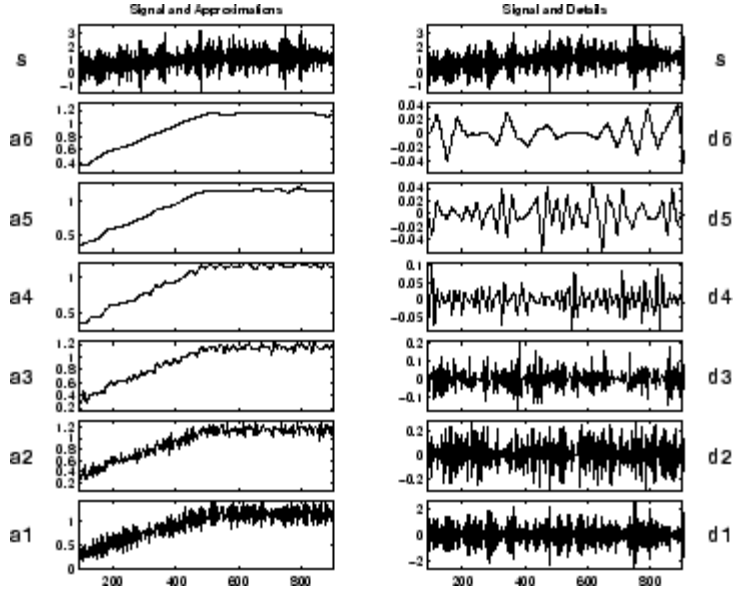
Example 10: A Ramp + Colored Noise

Analyzing wavelet: *db3*

Decomposition levels: *6*

The signal is built in the same manner as in "Example 9: A Ramp + White Noise" on page 2-27, 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 analyzed 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.

Looking at the figure, 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.



Example 10: A Ramp + Colored Noise	
Addressed topics	• Detecting breakdown points • Processing noise • Detecting long-term evolution • Splitting signal components
Further exploration	• Compare with the $s_7(t)$ signal shown in "Example 9: A Ramp + White Noise" on page 2-27. • Identify the number of levels needed to suppress the noise almost entirely. • Identify the noise characteristics. Use the coefficients and the command line mode.

Example 11: A Sine + White Noise

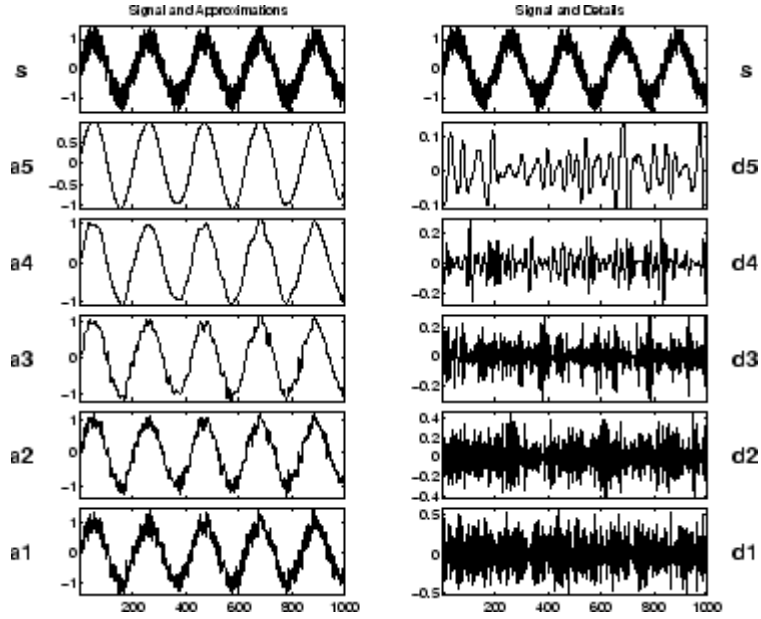
Analyzing wavelet: *db5*

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.



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”.

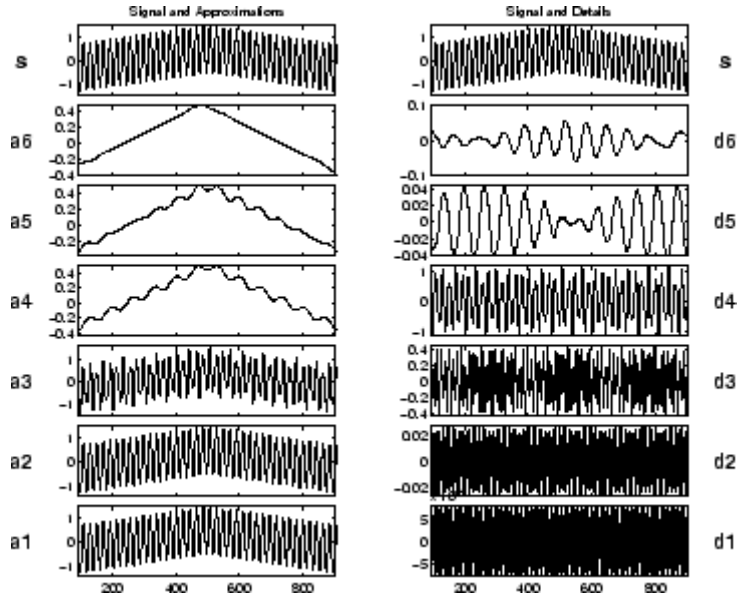
D_1 and D_2 are very small. This suggests that the signal contains no components with periods that are short in relation to the sampling period.

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.



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*

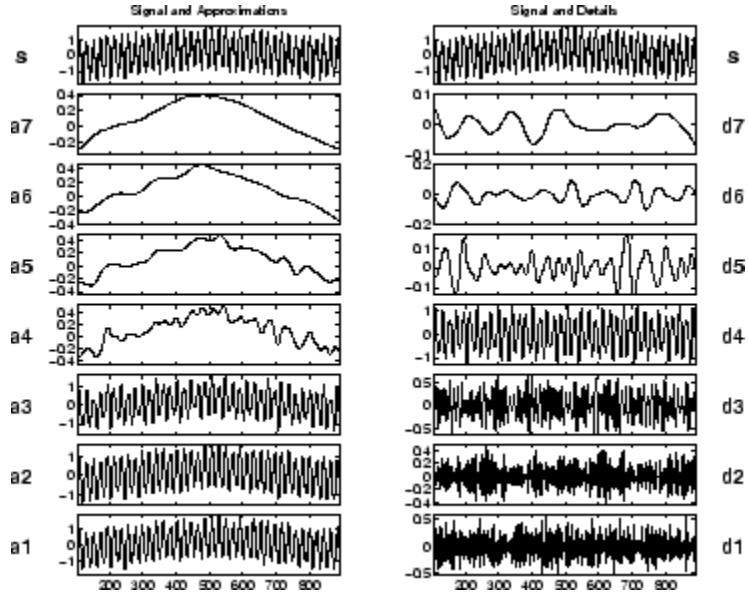
The signal examined here is the same as the previous signal plus a uniform white noise divided by 3. The analysis can, therefore, be compared to the previous analysis. All differences are due to the presence of the noise.

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.



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

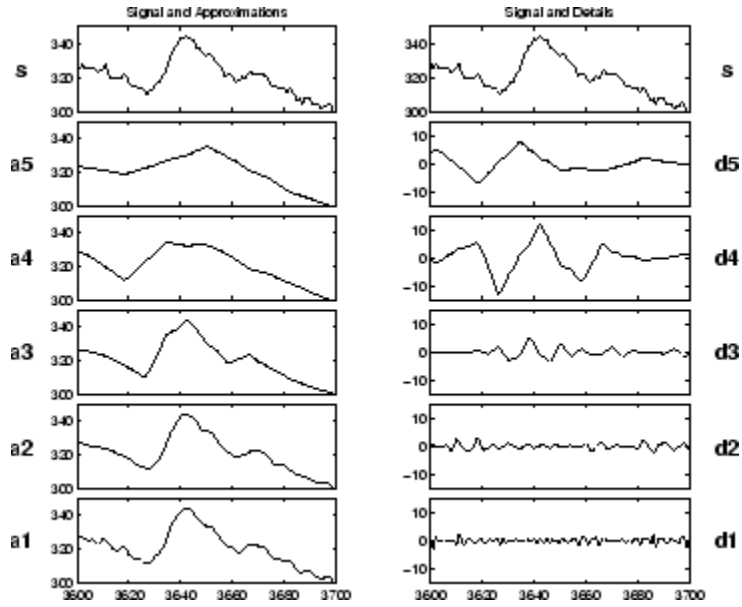
Analyzing wavelet: *db3*

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 multiscale 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 and 16 time-units.



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 “Case Study: An Electrical Signal” on page 2-39.

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 30 minutes to 2 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 second plot in the “Analysis of the End of the Night Period” on page 2-42.

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, 50% of the consumption is accounted for by industry, and the rest by individual consumers. 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 million 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 1-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: sensor errors and the state noise.

We shall not report here the complete analysis, which is included in the paper [MisMOP94] (see "References" on page 6-168). 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, it is much simpler. The midday period has a complicated structure because the

intensity of the electricity consumer activity is high and it presents very large changes. At the end of the night, the activity is low and it changes slowly.

For the local analysis, the decomposition is taken up to the level j = 5, because $2^5 = 32$ is very close to 30 minutes. We are then able to study the components of the signal for which the period is less than 30 minutes.

The analyzing wavelet used here is `db3`.

The results are described similarly for the two periods.

Analysis of the Midday Period

This signal (see "Example 14: A Real Electricity Consumption Signal" on page 2-37) is also analyzed more crudely in "Example 14: A Real Electricity Consumption Signal" on page 2-37.

The shape is a middle mode between 12:30 p.m. and 1:00 p.m., preceded and followed by a hollow off-peak, and next a second smoother mode at 1:15 p.m. 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.

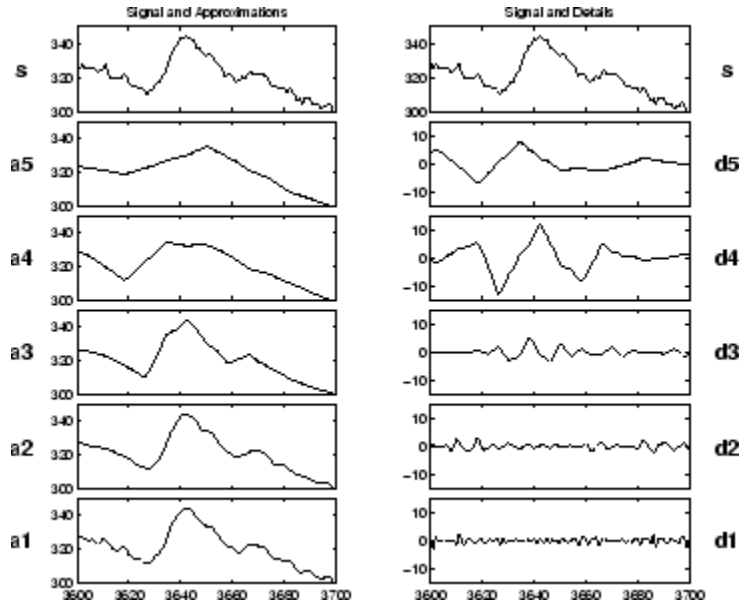
The details D_1 and D_2 have small values and may be considered as local short-period discrepancies caused by the high-frequency components of sensor and state noises. In this bandpass, these noises are essentially due to measurement errors and fast variations of the signal induced by millions of state 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 a period of 8 to 16 minutes contain the crucial dynamics.



Analysis of the End of the Night Period

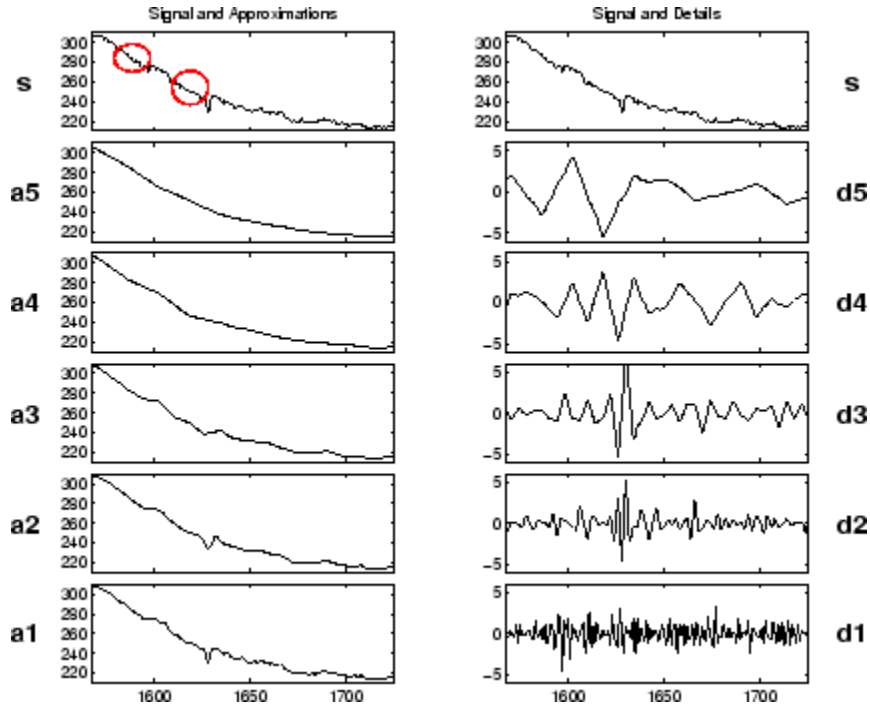
The shape of the curve during the end of the night is a slow descent, globally smooth, but locally highly irregular. One can hardly distinguish two successive local extrema in the vicinity of time $t = 1600$ and $t = 1625$. The approximation A_5 is quite good except at these two modes.

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 electrical 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 you 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.

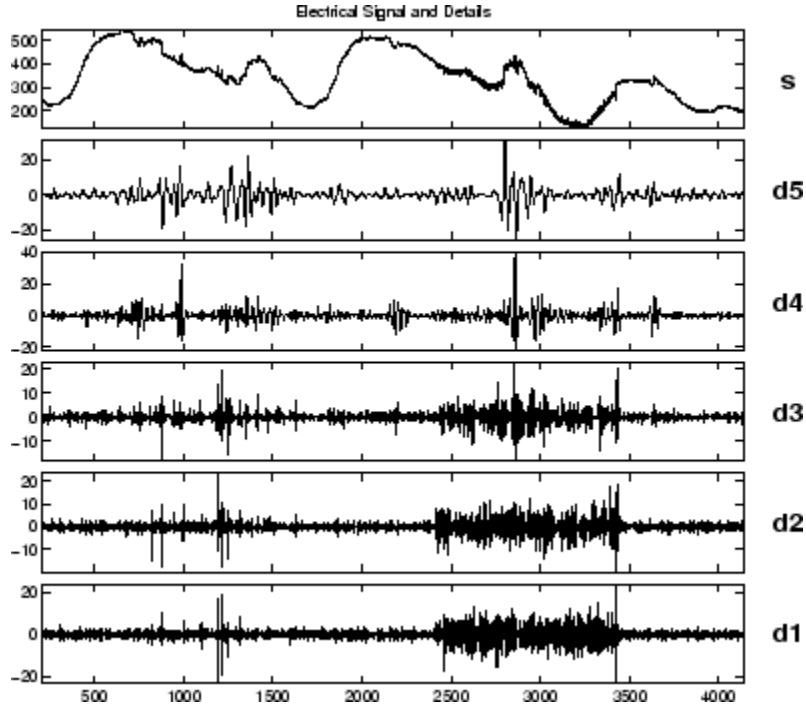


All the figures in this paragraph are generated using the graphical user interface tools, but the user can also process the analysis using the command line mode. The following example corresponds to a command line equivalent for producing the figure below.

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

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

% Reconstruct the details using the decomposition structure.
for i = 1:5
   D(i,:) = wrcoef('d',c,l,w,i);
end
```

Note This loop replaces five separate `wrcoef` statements defining the details. The variable `D` contains the five details.

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



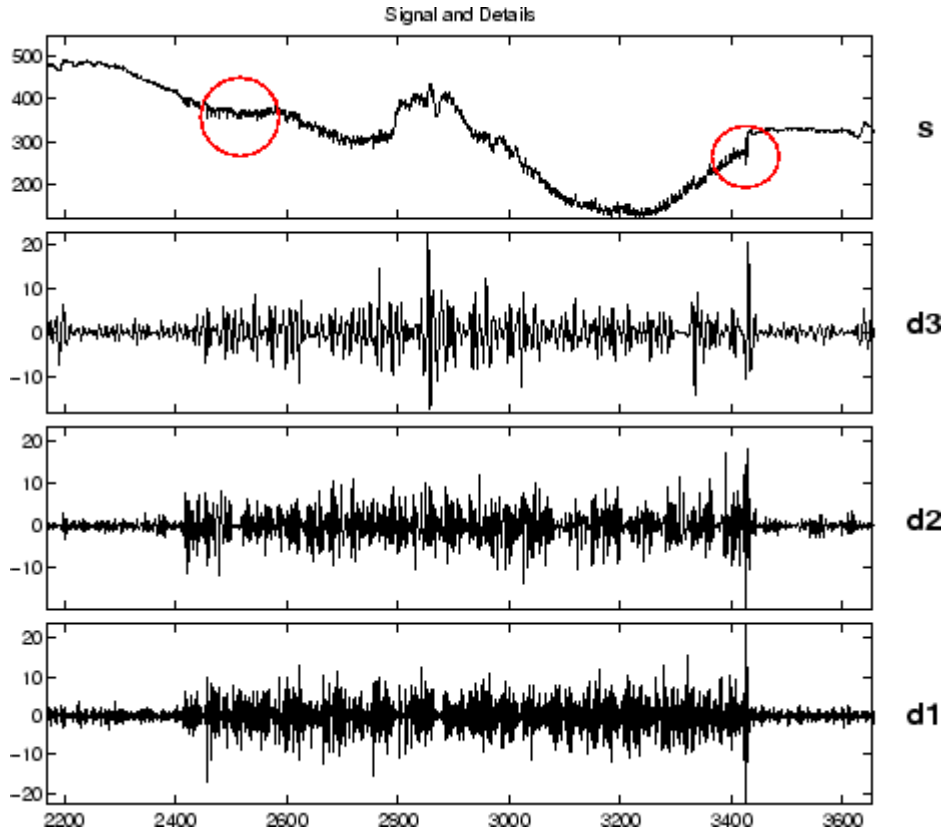
Suggestions for Further Analysis

Let us now make some suggestions for possible further analysis starting from the details of the decomposition at level 5 of 3 days.

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.

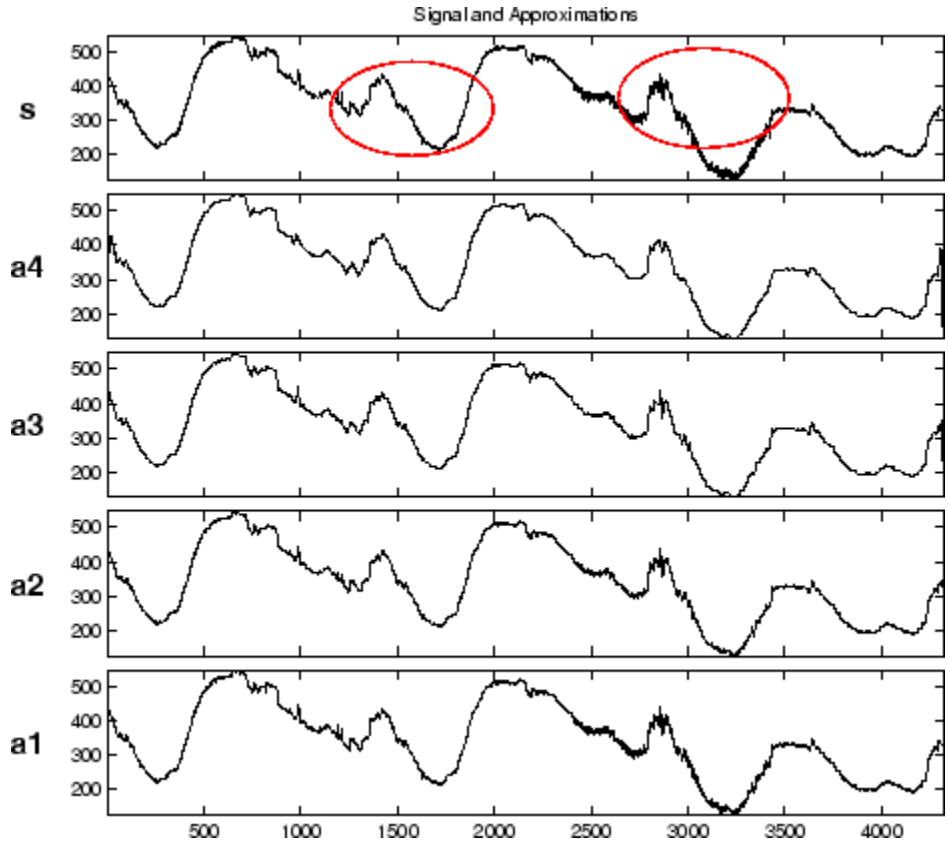
Indication: see figure below.



Suppress the Noise

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

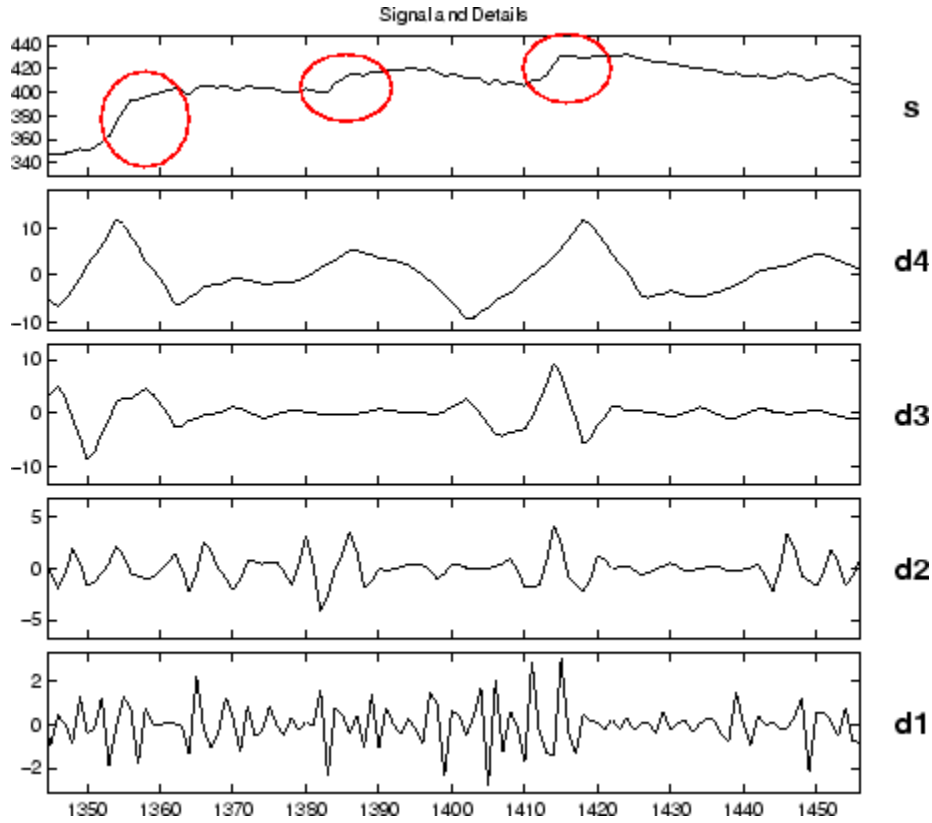
Indication: study the approximations and compare two successive days, the first without sensor failure and the second corrupted by failure (see figure below).



Identify Patterns in the Details

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

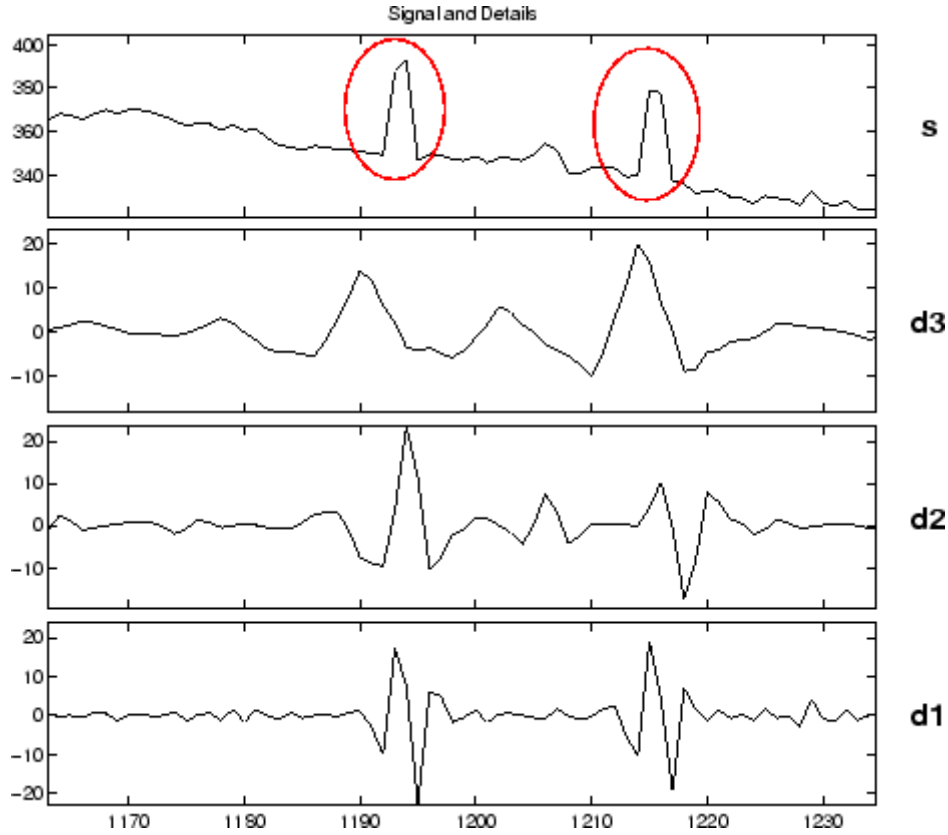
Indication: the figure below gives an example of such a period. Focus on details D_2, D_3, and D_4 around abscissa 1350, 1383, and 1415 to detect abrupt changes of the signal induced by automatic switches.



Locate and Suppress Outlying Values

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

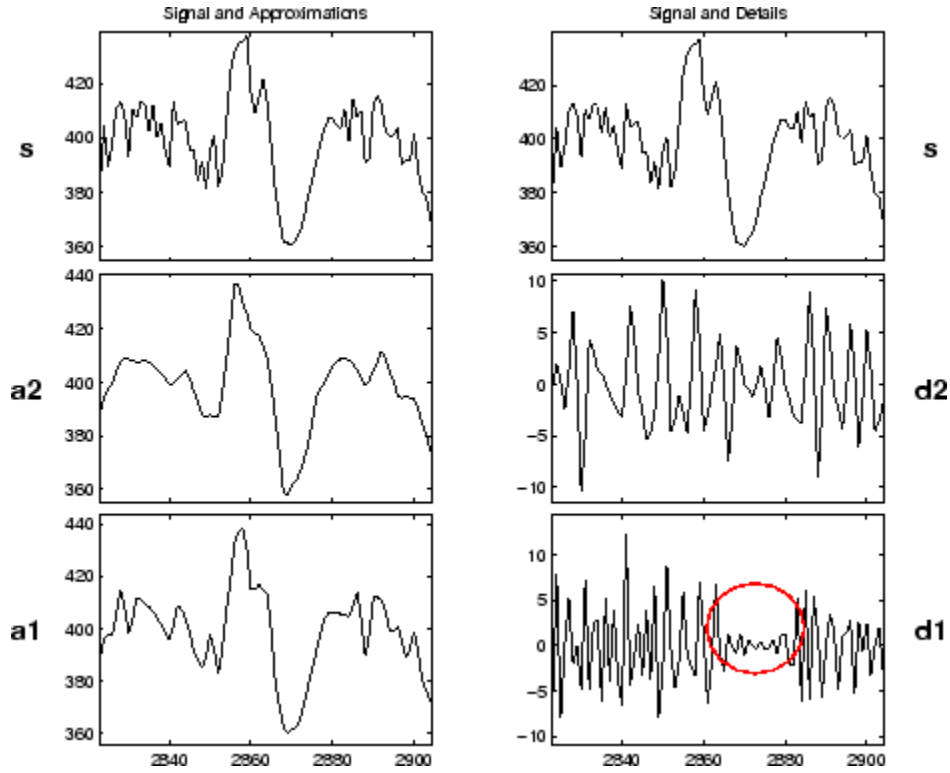
Indication: The figure below gives two examples of outliers around $t = 1193$ and $t = 1215$. The effect produced on the details is clear when focusing on the low levels. As far as outliers are concerned, D_1 and D_2 are synchronized with s, while D_3 shows a delayed effect.



Study Missing Data

Missing data have been crudely substituted (around observation 2870) by the estimation of 30 minutes of sampled data and spline smoothing for the intermediate time points. You can improve the interpolation by using an approximation and portions of the details taken elsewhere, thus implementing a sort of "graft."

Indication: see the figure below focusing around time 2870, and use the small variations part of D_1 to detect the missing data.


Signal and Approximations
Signal and Details
s
a2
a1
d2
d1
420
400
380
360
440
10
5
0
−5
−10
2840
2860
2880
2900

3

Using Wavelet Packets

About Wavelet Packet Analysis

Wavelet Toolbox software 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 data
- 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.

Note All the graphical user interface tools described in this Chapter let you import information from and export information to either disk or workspace. For more information see "File Menu Options" on page A-10.

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.

Note For more background on the wavelet packets, you can see the section "Wavelet Packets" on page 6-143.

Some object-oriented programming features are used for wavelet packet tree structures. For more detail, refer to Appendix B, "Object-Oriented Programming".

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

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

The toolbox provides these functions for wavelet packet analysis. For more information, see the reference pages. The reference entries for these functions include examples showing how to perform wavelet packet analysis via the command line.

Some more advanced examples mixing command line and GUI functions can be found in the section "Simple Use of Objects Through Four Examples" on page B-5.

Analysis-Decomposition Functions

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

Synthesis-Reconstruction Functions

Function Name	Purpose
`wprcoef`	Reconstruct coefficients
`wprec` and `wprec2`	Full reconstruction
`wpjoin`	Recompose packet

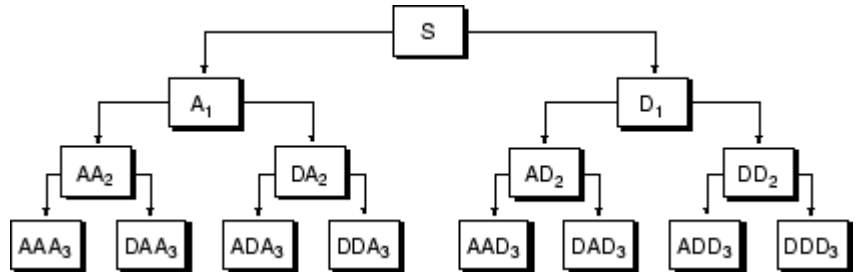
Decomposition Structure Utilities

Function Name	Purpose
`besttree`	Find best tree
`bestlevt`	Find best level tree
`entrupd`	Update wavelet packets entropy
`get`	Get WPTREE object fields contents
`read`	Read values in WPTREE object fields
`wenergy`	Entropy
`wp2wtree`	Extract wavelet tree from wavelet packet tree
`wpcutree`	Cut wavelet packet tree

De-Noising and Compression

Function Name	Purpose
`ddencmp`	Default values for de-noising and compression
`wpbmpen`	Penalized threshold for wavelet packet de-noising
`wpdencmp`	De-noising and compression using wavelet packets
`wpthcoef`	Wavelet packets coefficients thresholding
`wthrmngr`	Threshold settings manager

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 tradeoff 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.

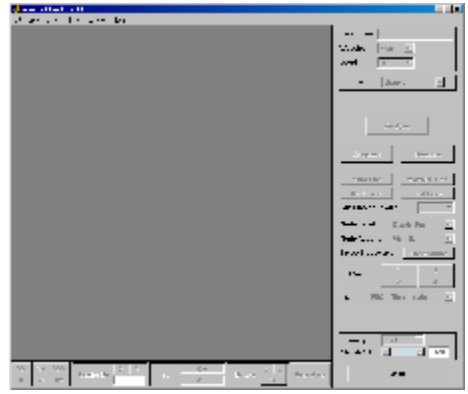
Starting the Wavelet Packet 1-D Tool.

1 From the MATLAB prompt, type `wavemenu`.

The **Wavelet Toolbox Main Menu** appears.

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

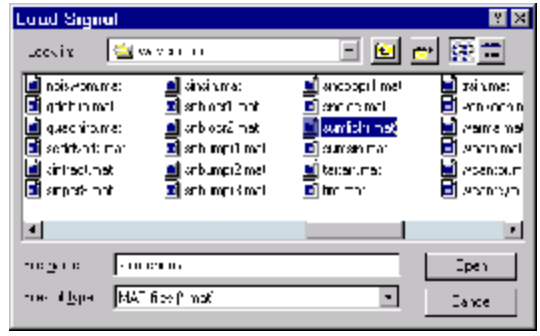
The tool appears on the desktop.



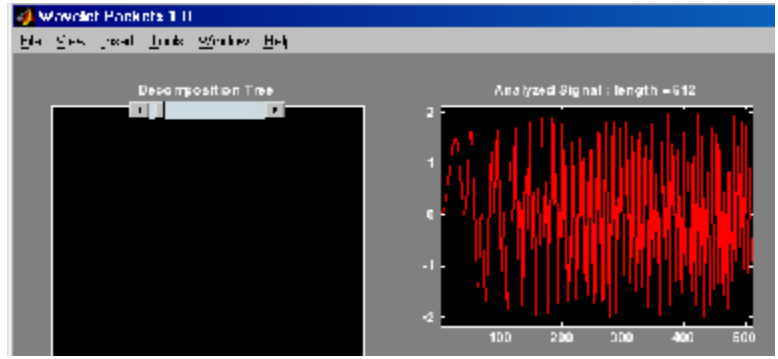
Loading a Signal.

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

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

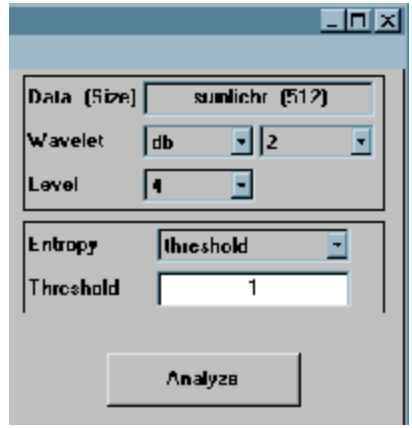


The `sumlichr` signal is loaded into the **Wavelet Packet 1-D** tool.



Analyzing a Signal.

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



The available entropy types are listed below.

Type	Description
Shannon	Nonnormalized 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 $1 \le p$.
Log Energy	The logarithm of "energy," defined as the sum over all samples: $\sum \log(s_i^2)$.
SURE (Stein's Unbiased Risk Estimate)	A threshold-based method in which the threshold equals $\sqrt{2\log_e\left(n\log_2(n)\right)}$ where n is the number of samples in the signal.
User	An entropy type criterion you define in a file.

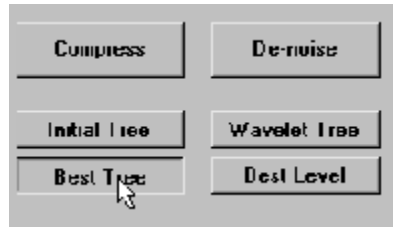
For more information about the available entropy types, user-defined entropy, and threshold parameters, see the `wentropy` reference page and "Choosing the Optimal Decomposition" on page 6-158.

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. For a more complete description, see Appendix A, "Wavelet Packet Tool Features (1-D and 2-D)" on page A-21.

Computing the Best Tree.

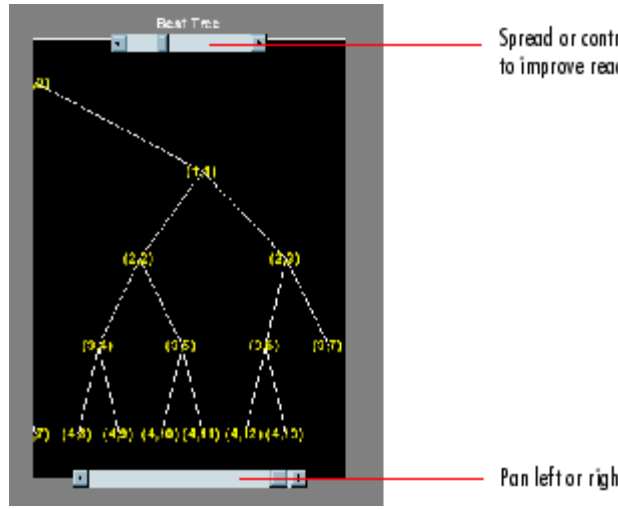
Because there are so many ways to reconstruct the original signal from the wavelet packet decomposition tree, we select the best tree before attempting to compress the signal.

1 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.

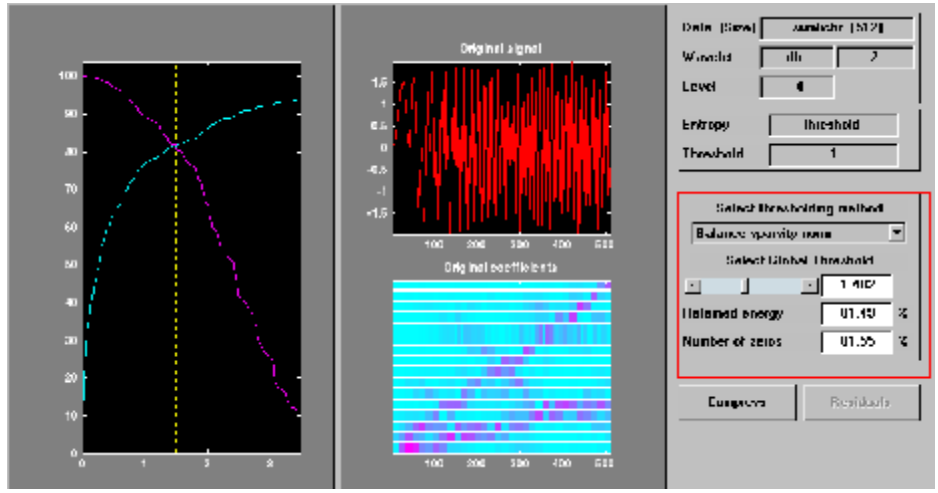


Compressing a Signal Using Wavelet Packets

Selecting a Threshold for Compression.

1 Click the **Compress** button.

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



The leftmost 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).

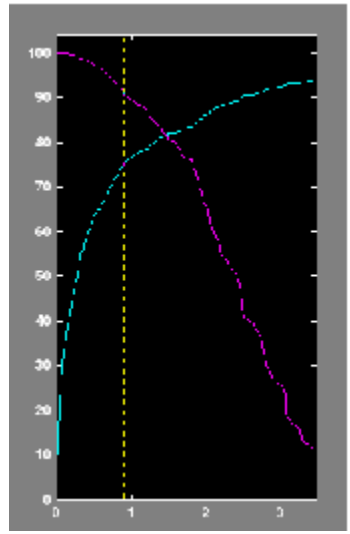
This threshold means that any signal element whose value is less than 1.482 will be set to zero when we perform the compression.

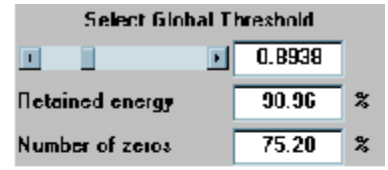
Threshold controls are located to the right (see the red box in the figure above). Note that the automatic threshold of 1.482 results in a retained energy of only 81.49%. This may cause unacceptable amounts of distortion, especially in the peak values of the oscillating signal. Depending on your design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

2 Adjust the threshold by typing `0.8938` in the text field opposite the threshold slider, and 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.





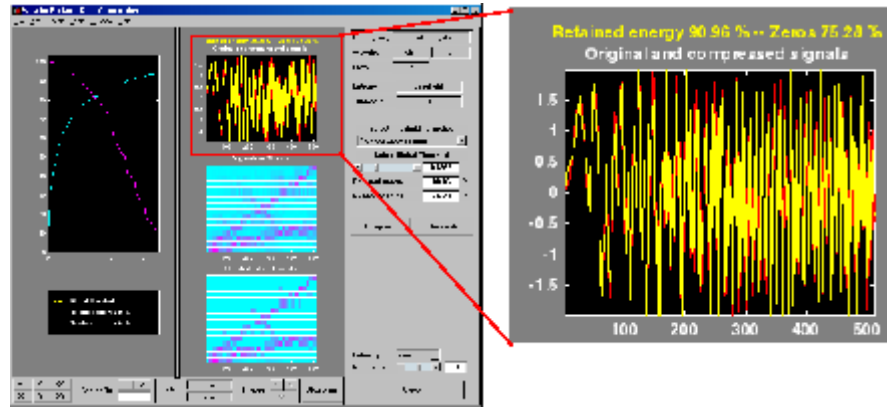
Note that, as we have *reduced* the threshold from 1.482 to 0.8938,

- 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.

1 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 can 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 Packets

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

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

where n is the length of the signal.

A more thorough discussion of the SURE criterion appears in "Choosing the Optimal Decomposition" on page 6-158. 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 ahead to step 3.*

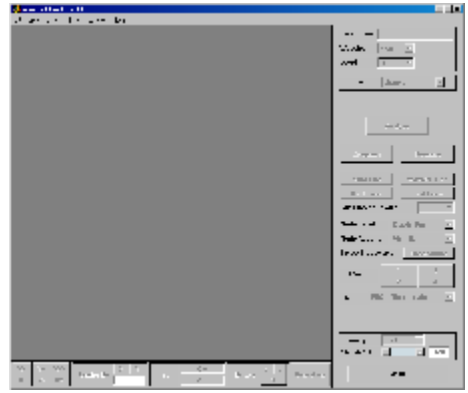
Starting the Wavelet Packet 1-D Tool.

1 From the MATLAB prompt, type `wavemenu`.

The **Wavelet Toolbox Main Menu** appears.

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

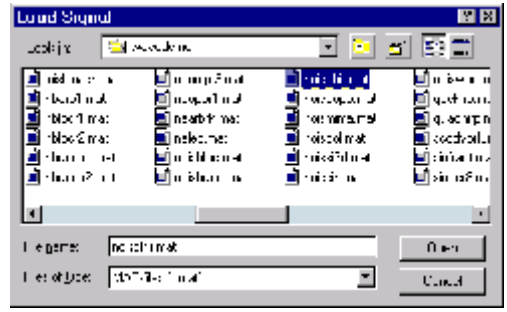
The tool appears on the desktop.



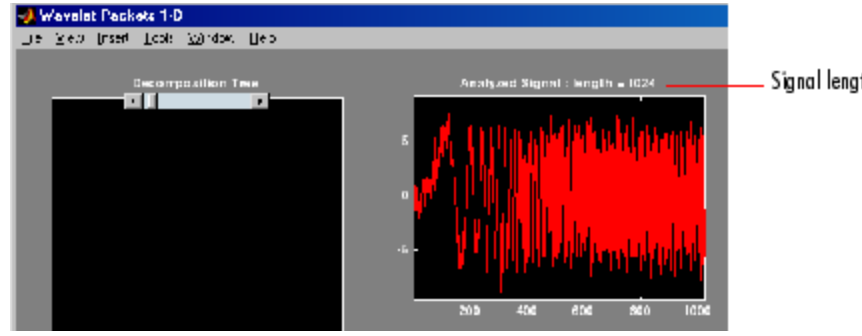
Loading a Signal

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

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

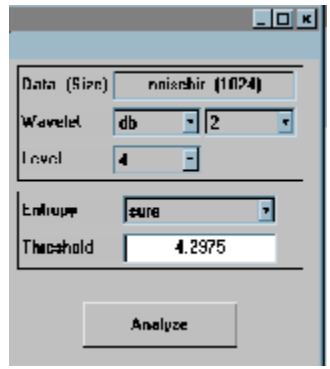


The `noischir` signal is loaded into the **Wavelet Packet 1-D** tool. Notice that the signal's length is 1024. This means we should set the SURE criterion threshold equal to `sqrt(2.*log(1024.*log2(1024)))`, or 4.2975.



Analyzing a Signal

4 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.



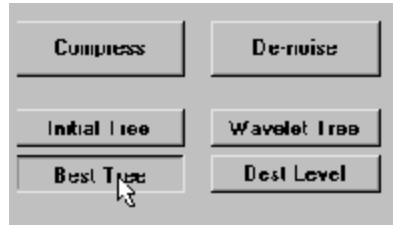
There is a pause while the wavelet packet analysis is computed.

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. For a more complete description, see "Wavelet Packet Tool Features (1-D and 2-D)" on page A-21.

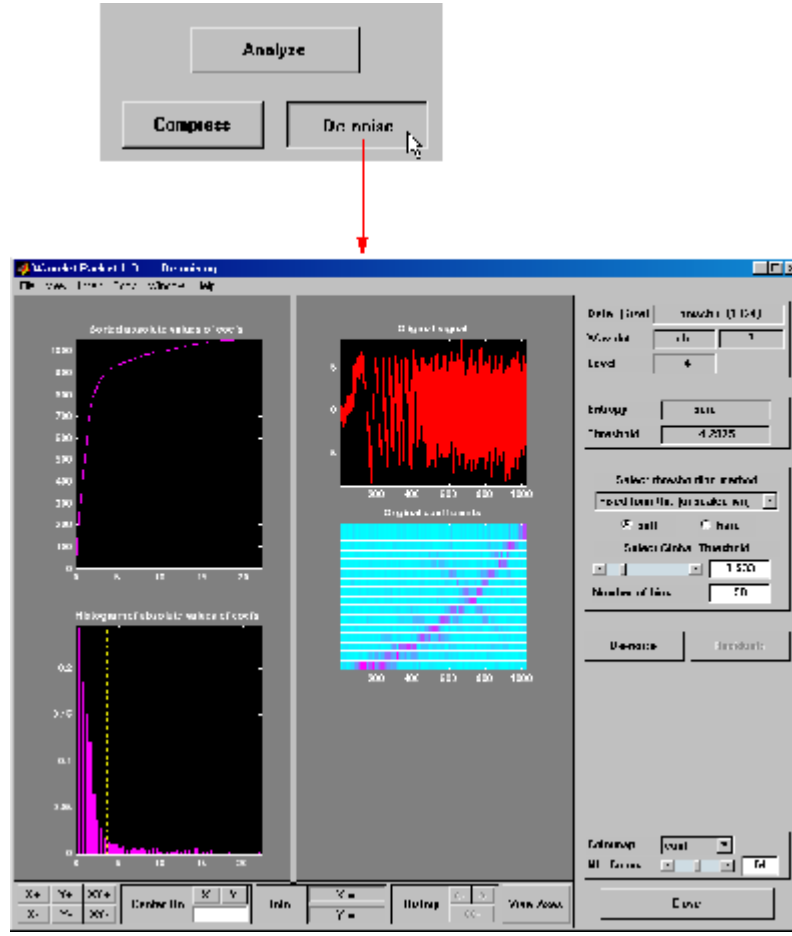
Computing the Best Tree and Performing De-Noising

5 Click the **Best Tree** button.

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

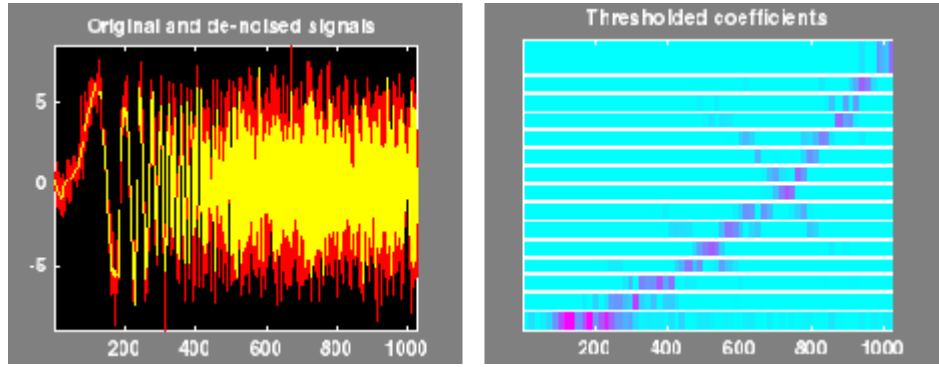


6 Click the **De-noise** button. This brings up the **Wavelet Packet 1-D De-Noising** window.



7 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 `wpdencmp` function 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.

"The FBI uses eight bits per pixel to define the shade of gray and stores 500 pixels per inch, which works out to about 700,000 pixels and 0.7 megabytes per finger to store finger prints in electronic form." (Wickerhauser, see the reference [Wic94] p. 387, listed in "References" on page 6-168).

"The technique involves a two-dimensional DWT, uniform scalar quantization (a process that truncates, or quantizes, the precision of the floating-point DWT output) and Huffman entropy coding (i.e., encoding the quantized DWT output with a minimal number of bits)." (Brislawn, see the reference [Bris95] p. 1278, listed in "References" on page 6-168).

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.

Note that the international standard JPEG 2000 will include the wavelets as a part of the compression and quantization process. This points out the present strength of the wavelets.

Starting the Wavelet Packet 2-D Tool.

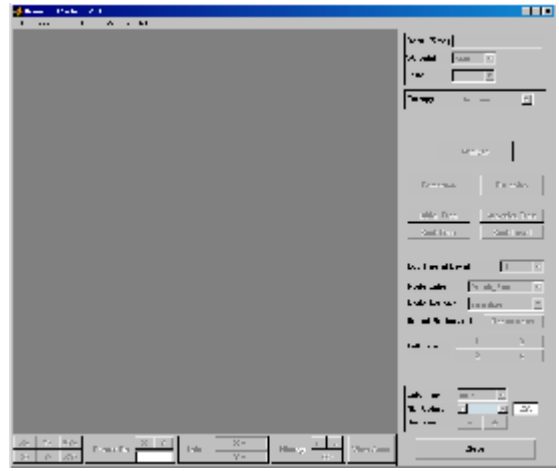
1 From the MATLAB prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.

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

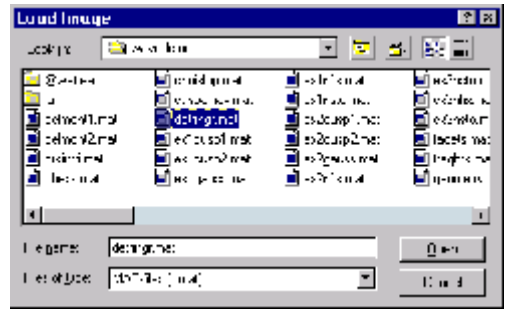
The tool appears on the desktop.



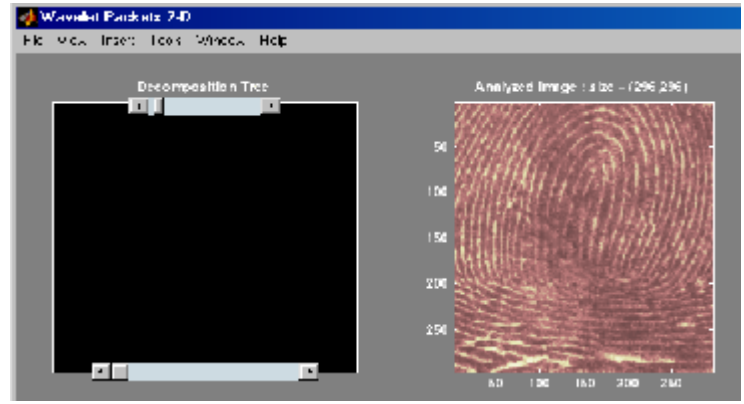
Loading an Image

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

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



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



Analyzing an Image

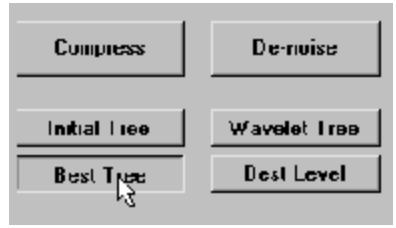
3 Make the appropriate settings for the analysis. Select the `haar` wavelet, level `3`, and entropy type `shannon`. Click the **Analyze** button.

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. For a more complete description, see "Wavelet Packet Tool Features (1-D and 2-D)" on page A-21.

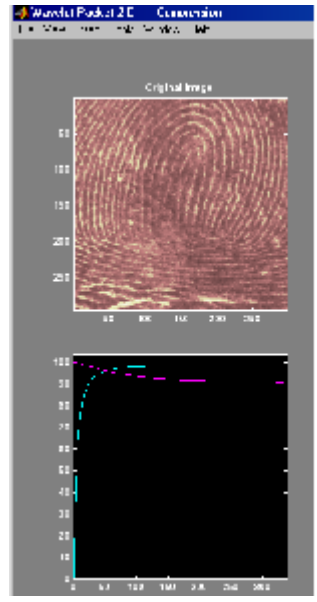
---

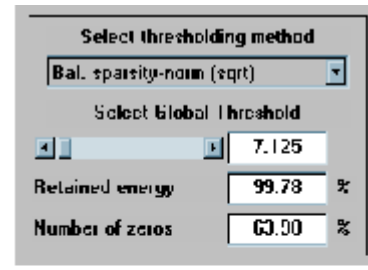
4 Click the **Best Tree** button to compute the best tree before compressing the image.



Compressing an Image Using Wavelet Packets

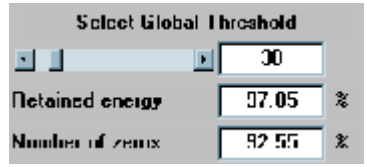
1 Click the **Compress** button to bring up the **Wavelet Packet 2-D Compression** window. Select the **Bal. sparsity-norm (sqrt)** option from the **Select thresholding method** menu.





Notice that the default threshold (7.125) 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 precompression step in a broader compression system.

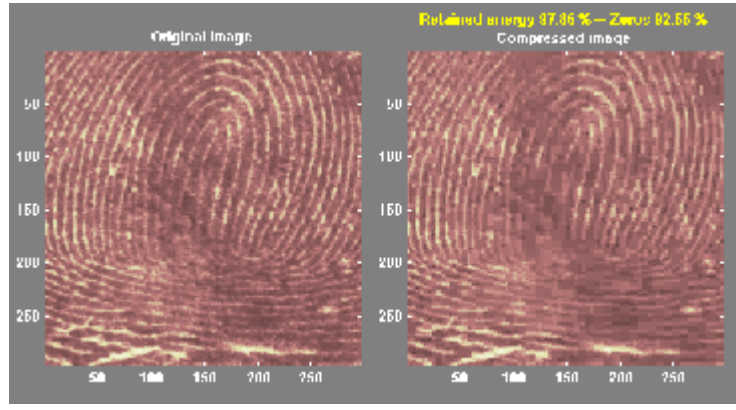
2 Alter the threshold: type the number 30 in the text field opposite the threshold slider located on the right side of the **Wavelet Packet 2-D Compression** window. Then press the **Enter** key.



Setting all wavelet packet coefficients whose value falls below 30 to zero yields much better results. Note that the new threshold achieves around 92% of zeros, while still retaining nearly 98% of the image energy. Compare this wavelet packet analysis to the wavelet analysis of the same image in "Compressing Images" on page 1-27.

3 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.

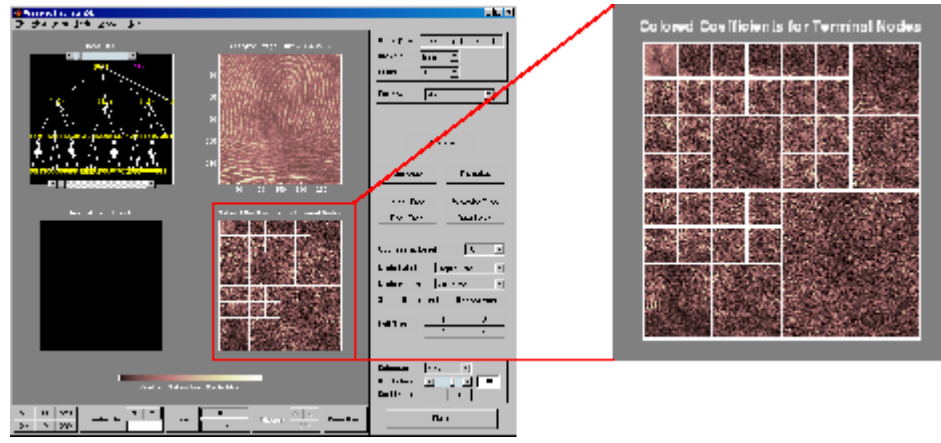


4 Click the **Close** button located at the bottom of the **Wavelet Packet 2-D Compression** window. Update the synthesized image by clicking **Yes** when the dialog box appears.

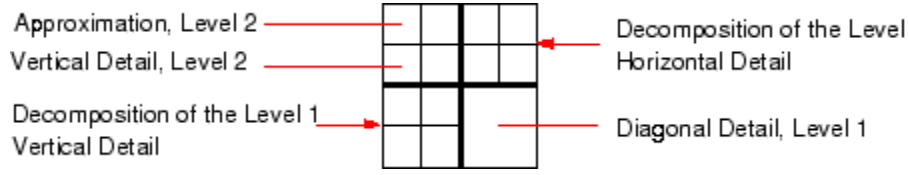
Take this opportunity to try out your own compression strategy. Adjust the threshold value, the entropy function, and the wavelet, and see if you can obtain better results.

Hint The `bior6.8` wavelet is better suited to this analysis than is `haar`, and can lead to a better compression ratio. When a biorthogonal wavelet is used, then instead of "Retained energy" the information displayed is "Energy ratio." For more information, see "Compression Scores" on page 6-118.

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 to disk
- Load signals, images, and one- and two-dimensional decompositions from disk into the **Wavelet Packet 1-D** and **Wavelet Packet 2-D** graphical tools

Saving Information to Disk

Using specific file formats, the graphical tools let you save synthesized signals or images, as well as one- or two-dimensional wavelet packet decomposition structures. 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 example analysis:

File > Example Analysis > db1 – depth: 2 – ent: shannon > sumsin

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 folder and filename for the MAT-file. For this example, choose the name `synthsig`.

To load the signal into your workspace, simply type

```
load synthsig
whos
```

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

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

```
valTHR

valTHR =
    1.9961
```

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 (with extension `mat` or other).

For example, load the example analysis:

File > Example 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 folder 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	Bytes	Class
X	256x256	524288	double array
map	255x3	6120	double array
valTHR	1x1	8	double array
wname	1x3	6	char array

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

Saving 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 folder with a name you choose, followed by the extension `wp1` (wavelet packet 1-D).

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

File > Example 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 folder and file name for storing the decomposition data. Type the name `wpdecex1d`.

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

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

Name	Size	Bytes	Class
data_name	1x6	12	char array
tree_struct	1x1	11176	wptree object
valTHR	0x0	0	double array

The variable `tree_struct` contains the wavelet packet tree structure. The variable `data_name` contains the data name and `valTHR` contains the global threshold, which is currently empty since the synthesized signal does not exist.

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	Bytes	Class
data_name	1x5	10	char array
map	255x3	6120	double array
tree_struct	1x1	527400	wptree object
valTHR	1x1	8	double array

Save options are also available when performing de-noising or compression inside the **Wavelet Packet 1-D** and **Wavelet Packet 2-D** tools.

In the Wavelet Packet De-Noising windows, you can save the de-noised signal or image and the decomposition. The same holds true for the Wavelet Packet Compression windows.

This way, you can save directly many different trials from inside the De-Noising and Compression windows without going back to the main Wavelet Packet windows during a fine-tuning process.

Note When saving a synthesized signal (1-D), a synthesized image (2-D) or a decomposition to a MAT-file, the extension of this file is free. The `mat` extension is not necessary.

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 (with extension `mat` or other).

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

```
save warma warma
```

The workspace variable `warma` must be a vector.

```
sizwarma = size(warma)

sizwarma =
           1        1000
```

To load this signal into the **Wavelet 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.

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

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 (with extension `mat` or other).

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

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

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.

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

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. 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 file contains the following variables:

Variable	Status	Description
`tree_struct`	Required	Object specifying the tree structure
`data_name`	Optional	String specifying the name of the decomposition
`valTHR`	Optional	Global threshold (can be empty if neither compression nor de-noising has been done)

These variables must be saved in a MAT-file (with extension `wp1` or other).

Two-dimensional data file contains the following variables:

Variable	Status	Description
`tree_struct`	Required	Object specifying the tree structure
`data_name`	Optional	String specifying the name of the decomposition
`map`	Optional	Image map
`valTHR`	Optional	Global threshold (can be empty if neither compression nor de-noising has been done)

These variables must be saved in a MAT-file (with extension `wp2` or other).

To load the properly formatted data, use the menu option **File > Load Decomposition Structure** from the appropriate tool, and 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.

Note When loading a signal (1-D), an image (2-D), or a decomposition (1-D or 2-D) from a MAT-file, the extension of this file is free. The `mat` extension is not necessary.

4

1-D Continuous Wavelet Analysis Using Discrete Fourier Transforms (DFT)

DFT-Based Continuous Wavelet Analysis Using Command Line

In this section...

To implement a DFT-based continuous wavelet analysis in the MATLAB command window, use `cwtft` and `icwtft`.

For the mathematical basis of the DFT-based continuous wavelet analysis and synthesis see:

- "Continuous Wavelet Transform via the Inverse Discrete Fourier Transform"
- "Inverse Continuous Wavelet Transform"

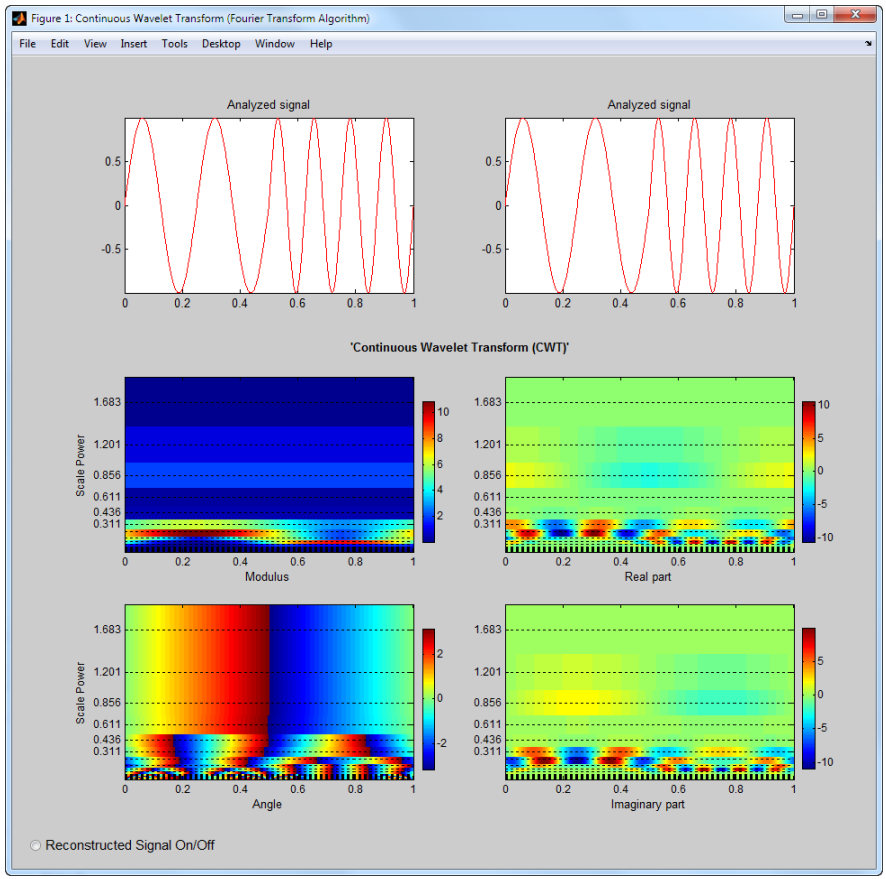
CWT of Sum of Disjoint Sinusoids

The signal is a sum of two disjoint sinusoids. The sampling frequency is 1023 Hz. The total signal duration is 1 second. The frequencies of the two sine waves are 4 and 8 Hz. The 4-Hz sine wave has support on the initial 1/2 second of the 1-second interval. The 8-Hz sine wave has support on the final 1/2 second.

```
N = 1024;
t = linspace(0,1,N);
dt =1/(N-1);
Y = sin(8*pi*t).*(t<=0.5) + sin(16*pi*t).*(t>0.5);
```

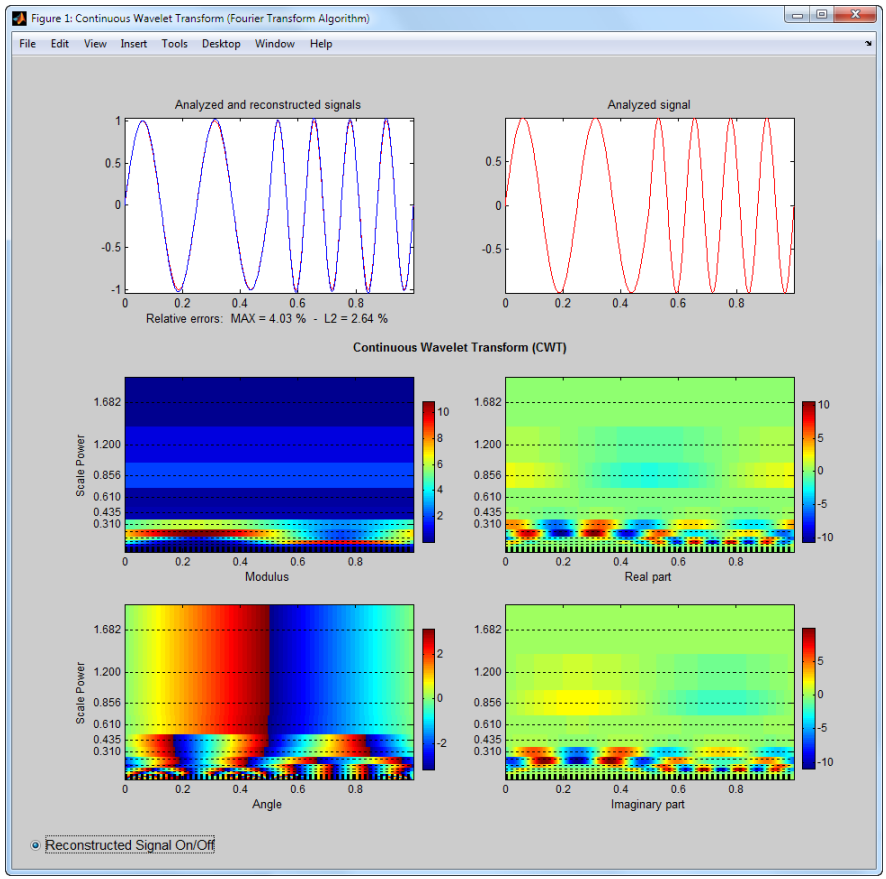
Obtain the continuous wavelet transform (CWT) using the default analytic Morlet wavelet, and plot the results.

```
sig = {Y,dt};
cwtS1 = cwtft(sig,'plot');
```



The figure shows a plot of the original signal. The CWT moduli, real and imaginary parts of the CWT coefficients, and the CWT coefficient arguments (phase angles) also appear as plots.

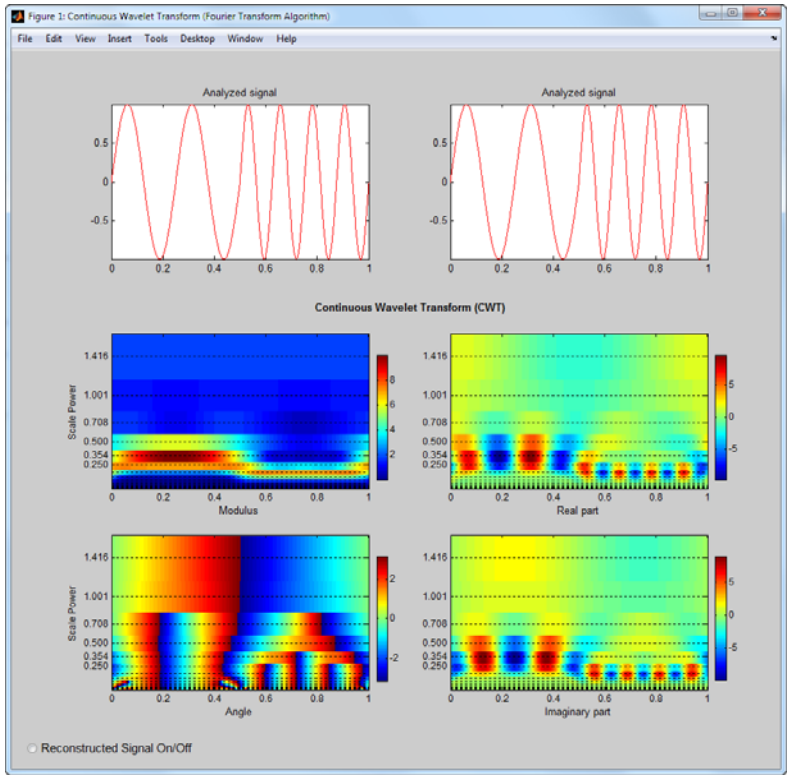
You can display the reconstructed signal by enabling the radio button at the bottom-left corner of the plot. Enabling the radio button superimposes the reconstructed signal on the original signal in the top-left corner of the figure. The relative maximum and quadratic (L2 norm) errors appear under the plot.



You can customize your continuous wavelet analysis by providing additional inputs to `cwtft`. In the following example, specify the analyzing wavelet as the Paul wavelet of order 8. Specify the initial scale, the spacing between scales, and the number of scales. By default, the scale vector is logarithmic to the base 2.

```
% smallest scale, spacing between scales, number of scales
dt = 1/1023;
s0  = 2*dt;  ds = 0.5;  NbSc = 20;
% scale vector is
% scales = s0*2.^((0:NbSc-1)*ds);
wname = 'paul';
```

```
SIG = {Y,dt};
% Create SCA input as cell array
SCA = {s0,ds,NbSc};
% Specify wavelet and parameters
WAV = {wname,8};

% Compute and plot the CWT
cwtS2 = cwtft(SIG,'scales',SCA,'wavelet',WAV,'plot');
```



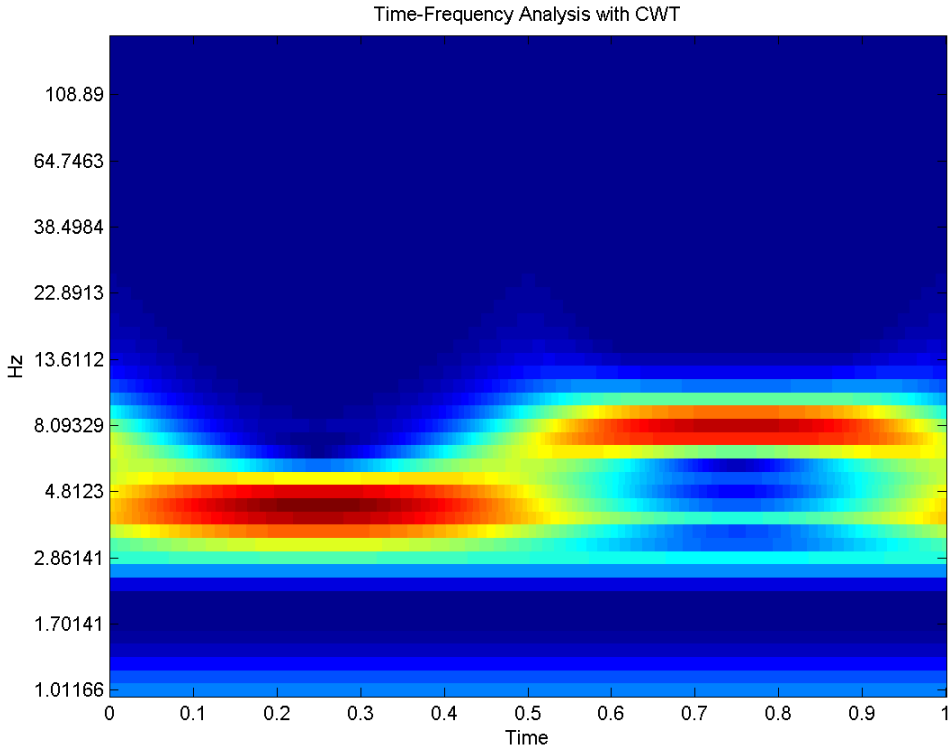
The oscillations at 4 and 8 Hz are clearly visible as alternating positive and negative real and imaginary parts. The 4-Hz oscillation occurs at a longer scale than the 8-Hz oscillation. In the plot of the CWT moduli, you see the transition from the 4-Hz (longer scale) sinusoid to the 8-Hz sinusoid (shorter scale) around 0.5 seconds.

Approximate Scale-Frequency Conversions

There is not a direct correspondence between Fourier wavelength and scale. However, you can find conversion factors for select wavelets that yield an approximate scale-frequency correspondence. You can find these factors for wavelets supported by `cwtft` listed on the reference page.

This example shows you how to change the scale axis to an approximate frequency axis for analysis. Use the sum of disjoint sinusoids as the input signal. Set the initial scale to `6*dt`, the scale increment to `0.15`, and the number of scales to `50`. Use the Fourier factor for the Morlet wavelet to convert the scale vector to an approximate frequency vector in hertz. Plot the result.

```
figure;
s0  = 6*dt;  ds = 0.15;  NbSc = 50;
wname = 'morl';
SCA = {s0,ds,NbSc};
cwtsig = cwtft({Y,dt},'scales',SCA,'wavelet',wname);
MorletFourierFactor = 4*pi/(6+sqrt(2+6^2));
Scales = cwtsig.scales.*MorletFourierFactor;
Freq = 1./Scales;
imagesc(t,[],abs(cwtsig.cfs));
indices = get(gca,'ytick');
set(gca,'yticklabel',Freq(indices));
xlabel('Time'); ylabel('Hz');
title('Time-Frequency Analysis with CWT');
```

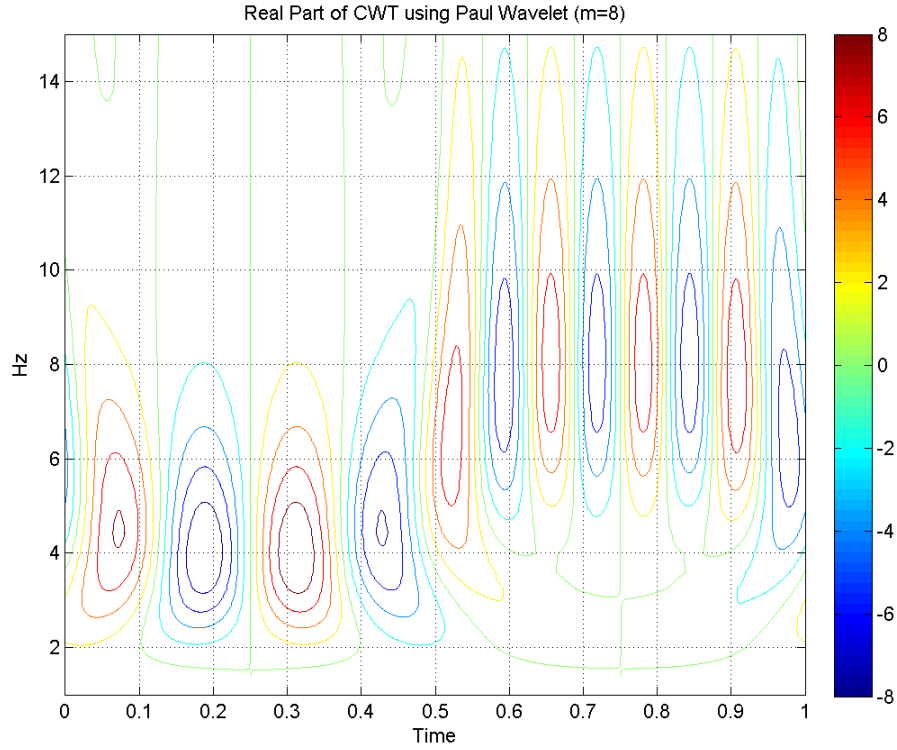


You can see the signal contains significant energy at approximately 4 Hz over the first 1/2 second. In the final 1/2 second interval, the predominant signal energy transitions higher in frequency to approximately 8 Hz.

Repeat the above example using the Paul analyzing wavelet with order, *m*, equal to 8. Use a contour plot of the real part of the CWT to visualize the sine waves at 4 and 8-Hz. The real part exhibits oscillations in the sign of the wavelet coefficients at those frequencies.

```
s0  = 6*dt;  ds = 0.15;  NbSc = 50;
m = 8;
% scale vector is
% scales = s0*2.^((0:NbSc-1)*ds);
wname = 'paul';
```

```
SIG = {Y,dt};
% Create SCA input as cell array
SCA = {s0,ds,NbSc};
% Specify wavelet and parameters
WAV = {wname,m};
cwtPaul = cwtft(SIG,'scales',SCA,'wavelet',WAV);
scales = cwtPaul.scales;
PaulFourierFactor = 4*pi/(2*m+1);
Freq = 1./(PaulFourierFactor.*scales);
contour(t,Freq,real(cwtPaul.cfs));
xlabel('Time'); ylabel('Hz'); colorbar;
title('Real Part of CWT using Paul Wavelet (m=8)');
axis([0 1 1 15]); grid on;
```



Signal Reconstruction from CWT Coefficients

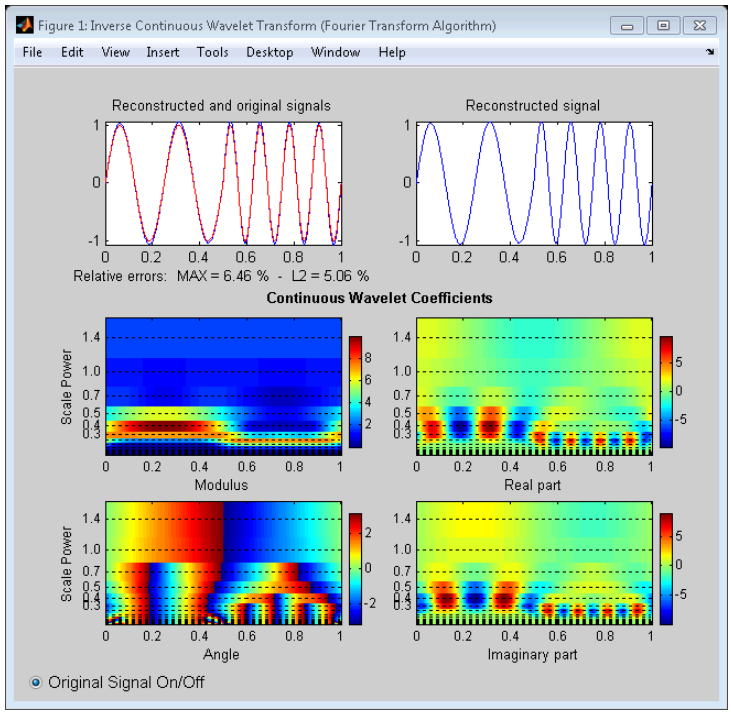
You can use the critically sampled (decimated) and oversampled (nondecimated) discrete wavelet transforms (DWT) to achieve perfect reconstruction of the input signal from the wavelet coefficients. To obtain a time and scale-dependent approximation to a signal, you can use a possibly-modified subset of the decimated or undecimated DWT coefficients.

The inversion of the CWT is not as straightforward. The simplest CWT inversion utilizes the single integral formula due to Morlet, which employs a Dirac delta function as the synthesizing wavelet. See "Inverse Continuous Wavelet Transform" for a brief mathematical motivation. `icwtft` and `icwtlin` both implement the single integral formula. Because of necessary approximations in the implementation of the single integral inverse CWT, you cannot expect to obtain perfect reconstruction. However, you can use the inverse CWT to obtain useful position and scale-dependent approximations to the input signal.

Implement the inverse CWT with logarithmically-spaced scales.

```
N = 1024;
t = linspace(0,1,N);
dt =1/(N-1);
Y = sin(8*pi*t).*(t<=0.5) + sin(16*pi*t).*(t>0.5);
dt = 1/1023;
s0  = 2*dt;  ds = 0.5;  NbSc = 20;
% scale vector is
% scales = s0*2.^((0:NbSc-1)*ds);
wname = 'paul';
SIG = {Y,dt};
% Create SCA input as cell array
SCA = {s0,ds,NbSc};
% Specify wavelet and parameters
WAV = {wname,8};
cwtS2 = cwtft(SIG,'scales',SCA,'wavelet',WAV);
YR1 = icwtft(cwtS2,'plot','signal',SIG);
norm(Y-YR1,2)
```

Enable the radio button in the left corner of the figure to plot the reconstructed signal.

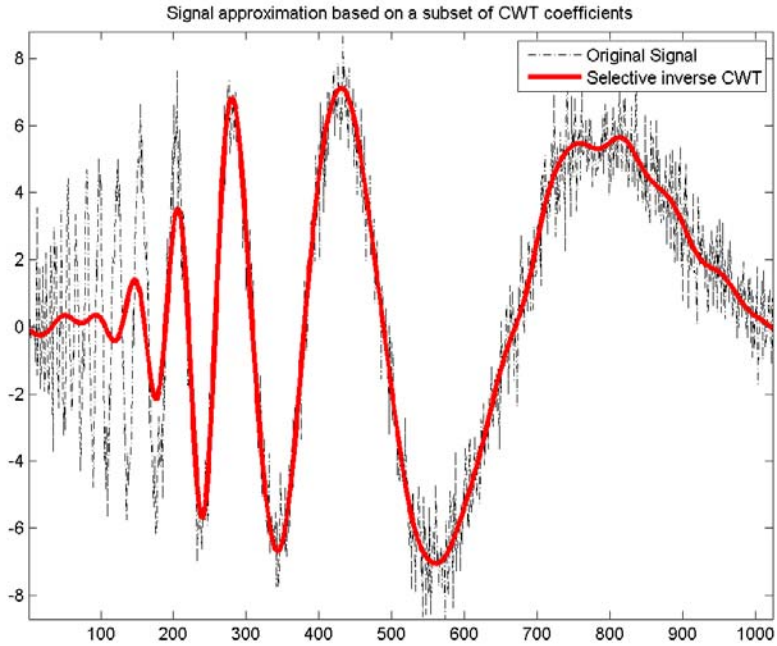


Signal Approximation with Modified CWT Coefficients

Obtain the CWT of a noisy Doppler (frequency-modulated) signal using the analytic Morlet wavelet. Reconstruct an approximation by selecting a subset of the CWT coefficients. By eliminating the smallest scales, you obtain a lowpass approximation to the signal. The lowpass approximation produces a smooth approximation to the lower-frequency features of the noisy Doppler signal. The high-frequency (small scale) features at the beginning of the signal are lost.

```
% Define the signal
load noisdopp; Y = noisdopp;
N = length(Y);

% Define parameters before analysis
dt  = 1;
```

```
s0  = 2*dt;  ds = 0.4875;  NbSc = 20;
wname = 'morl';
SIG = {Y,dt};
SCA = {s0,ds,NbSc};
WAV = {wname,[]};

% Compute CWT analysis
cwtS4 = cwtft(SIG,'scales',SCA,'wavelet',WAV);

% Thresholding step building the new structure
cwtS5 = cwtS4;
newCFS = zeros(size(cwtS4.cfs));
newCFS(11:end,:) = cwtS4.cfs(11:end,:);
cwtS5.cfs = newCFS;

% Reconstruction from the modified structure
YRDen = icwtft(cwtS5,'signal',SIG);
plot(Y,'k-.');
hold on;
plot(YRDen,'r','linewidth',3); axis tight;
legend('Original Signal', 'Selective inverse CWT');
title('Signal approximation based on a subset of CWT coefficients');
```


Signal approximation based on a subset of CWT coefficients
Original Signal
Selective inverse CWT
8
6
4
2
0
-2
-4
-6
-8
100
200
300
400
500
600
700
800
900
1000

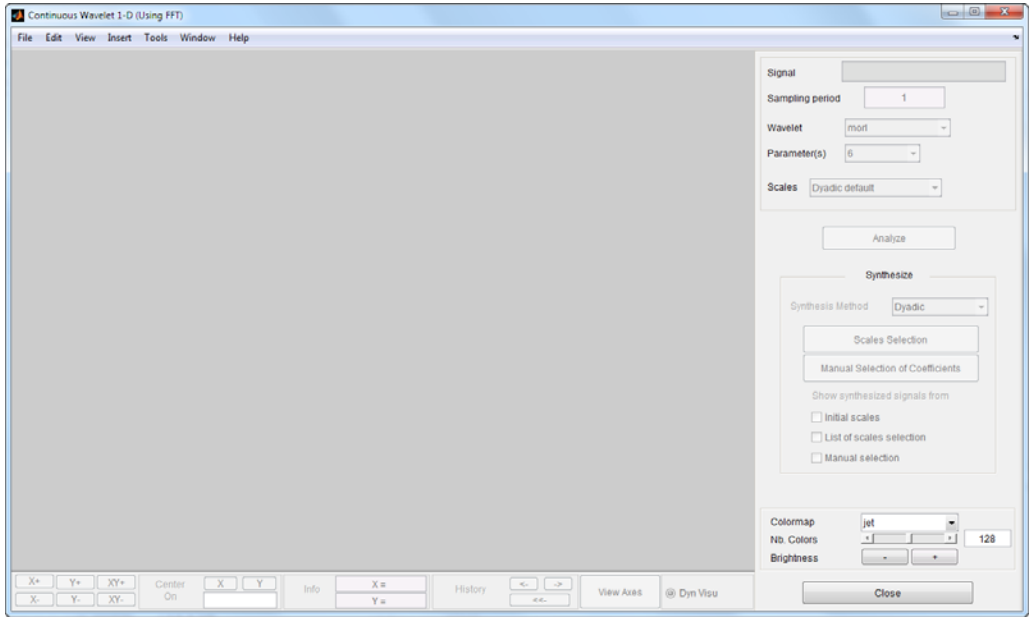
DFT-Based Continuous Wavelet Analysis Using Graphical User Interface

You can use the **Continuous Wavelet 1-D (Using FFT)** tool to analyze the same signals examined in "DFT-Based Continuous Wavelet Analysis Using Command Line" on page 4-2.

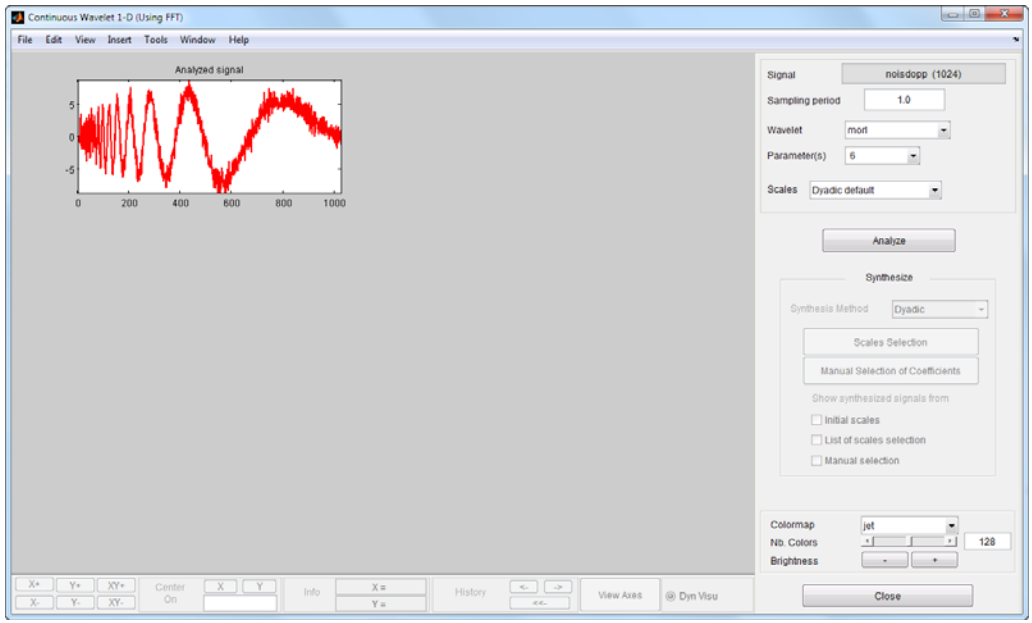
1. At the MATLAB command prompt, enter

   ```
   wavemenu
   ```

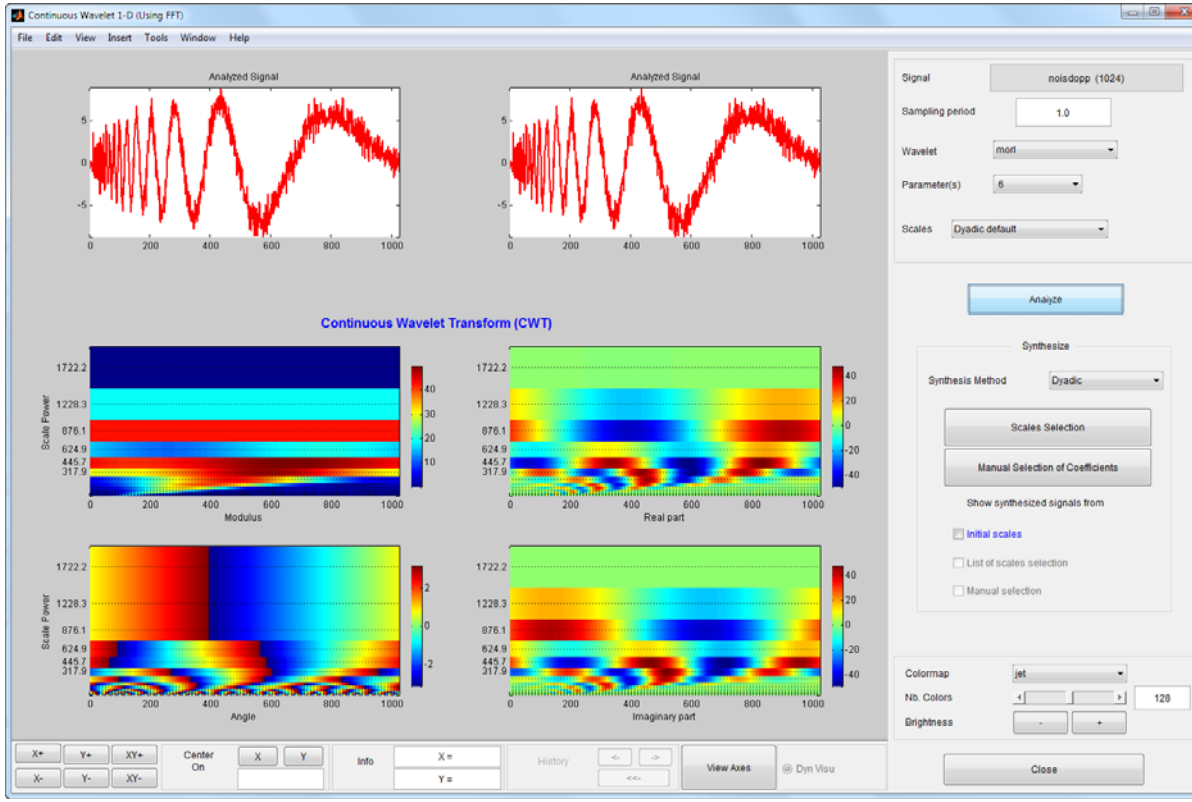
2. Click the **Continuous Wavelet 1-D (Using FFT)** menu item.



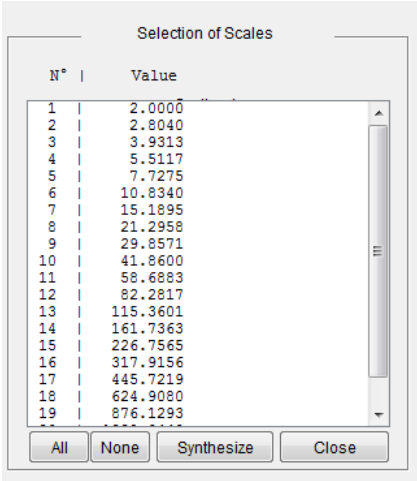
3. Choose the **File > Load Data** option. When the **Pick a file** dialog appears, select `noisdopp.mat` from the `toolbox/wavelet/wavedemo` folder.



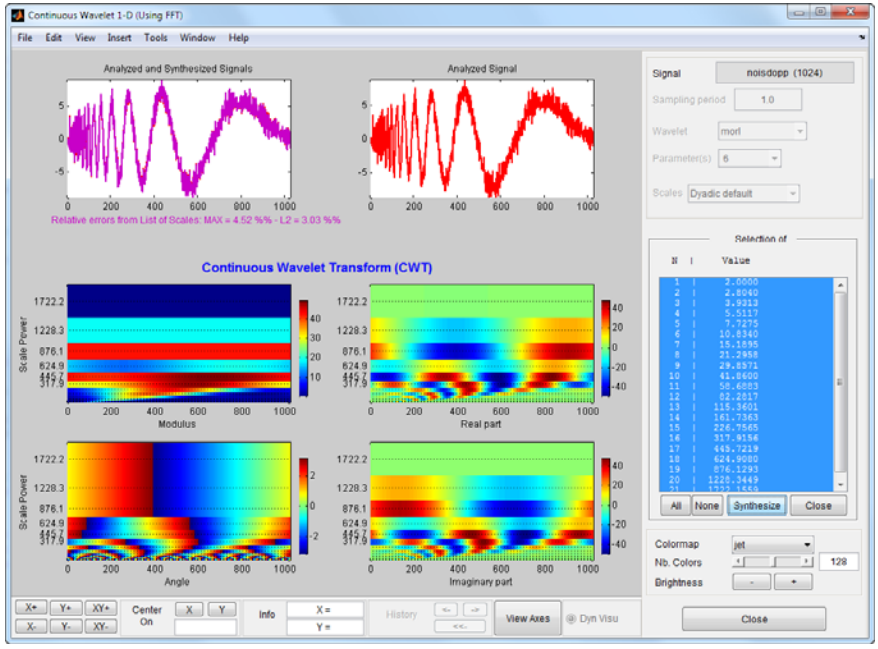
4 Using the menu default parameters, click **Analyze**.



5 Reconstruct the signal based on all the default dyadic scales. Click **Scales Selection**.



Select all scales by clicking **All**. Click **Synthesize**.



In the top left, the synthesized signal plot is superimposed on the original signal. The relative maximum and L2 errors are displayed under the plot.

The single integral CWT inversion does not produce perfect reconstruction, but the relative errors using the default logarithmically–spaced scales are small.

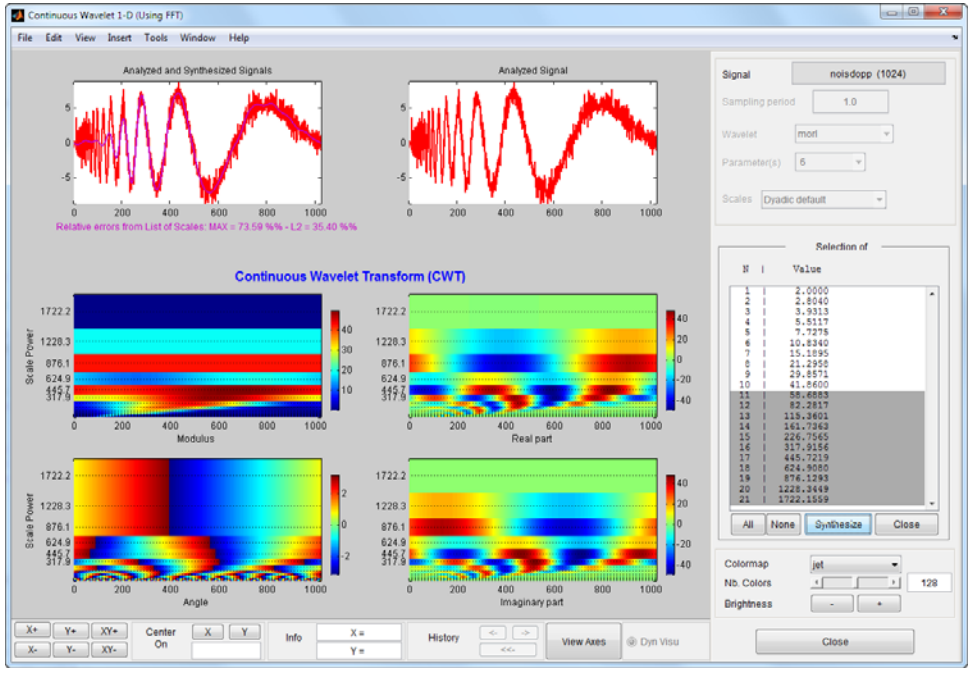
6 Obtain a signal approximation from selected scales.

Click **None** in the **Selection of Scales** panel to undo the scale selection. Then, select only scale indices greater than 10 and reconstruct an approximation to the original signal. Hold the `Ctrl` key while selecting scale indices 11–21. The scale indices correspond to the following physical scales.

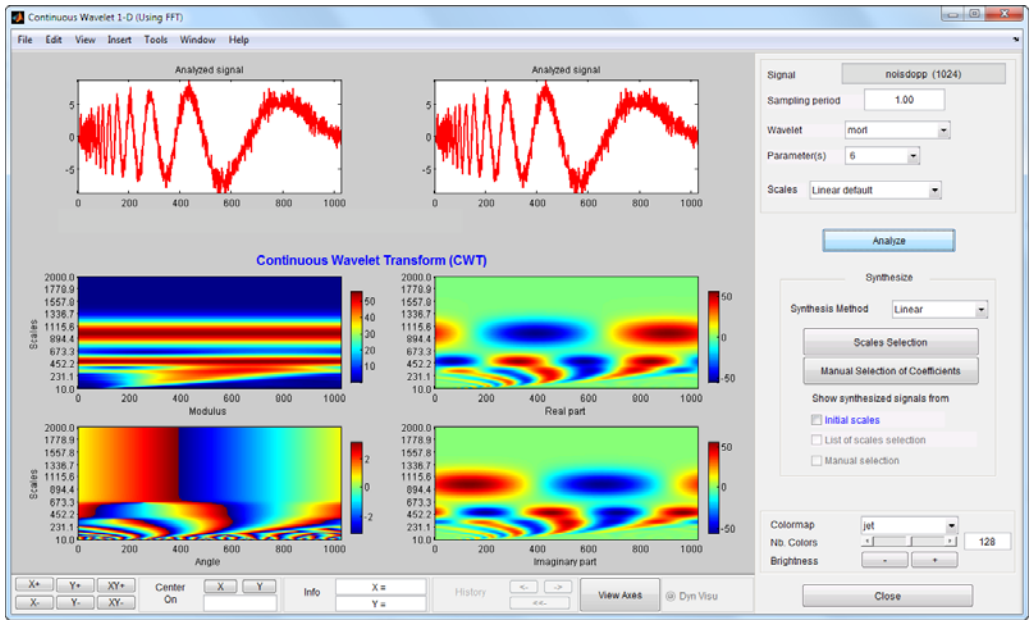
```
dt = 1;
s0 = 2*dt;
ds = 0.4875;
nb = 21;
physical_scales = s0*pow.^((0:nb-1)*ds);
```

7 Click **Synthesize**.

The reconstructed signal from scale indices 11–21 is a lowpass approximation to the noisy Doppler signal.



8 Analyze using linear scales. In the **Scales** drop-down menu in the upper right, select `Linear default` and click **Analyze**.

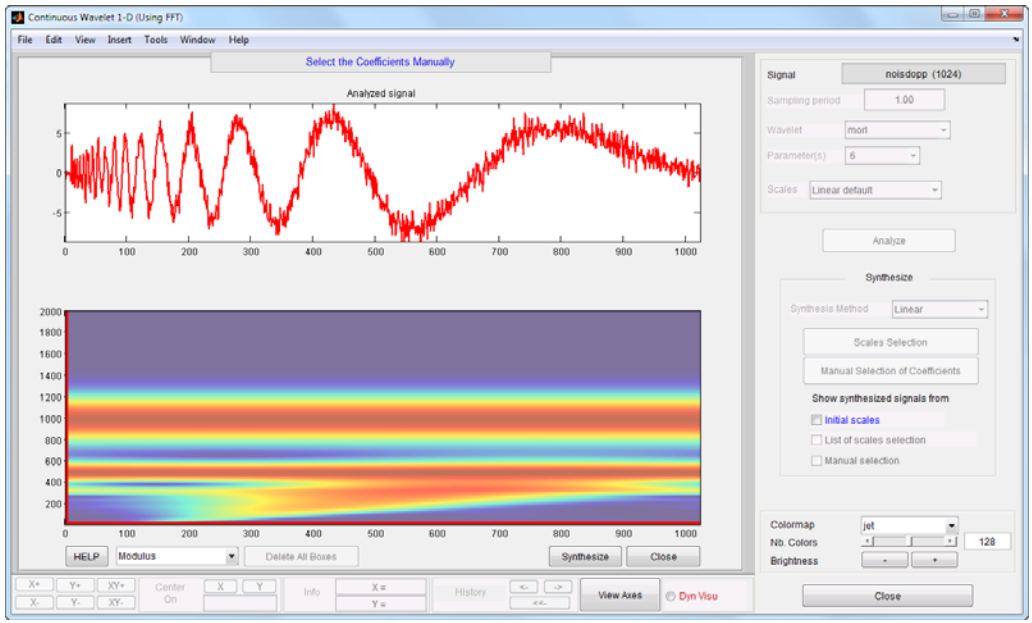


Note The other options under **Scales** include `Dyadic default` and `Manual`.

If you select `Manual`, a **Define Scales** button appears. Click **Define Scales** to set the parameters for your scale vector.

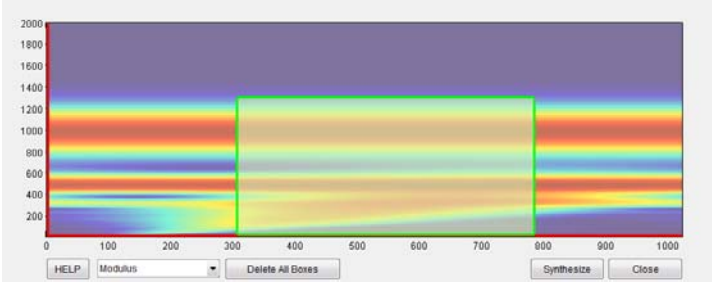
Manual Selection of CWT Coefficients

Select coefficients manually by graphically selecting the CWT coefficients. Reconstruct the signal from the selected coefficients. Click **Manual Selection of Coefficients**. The **Select the Coefficients Manually** panel appears with a single box containing all the CWT coefficient moduli.

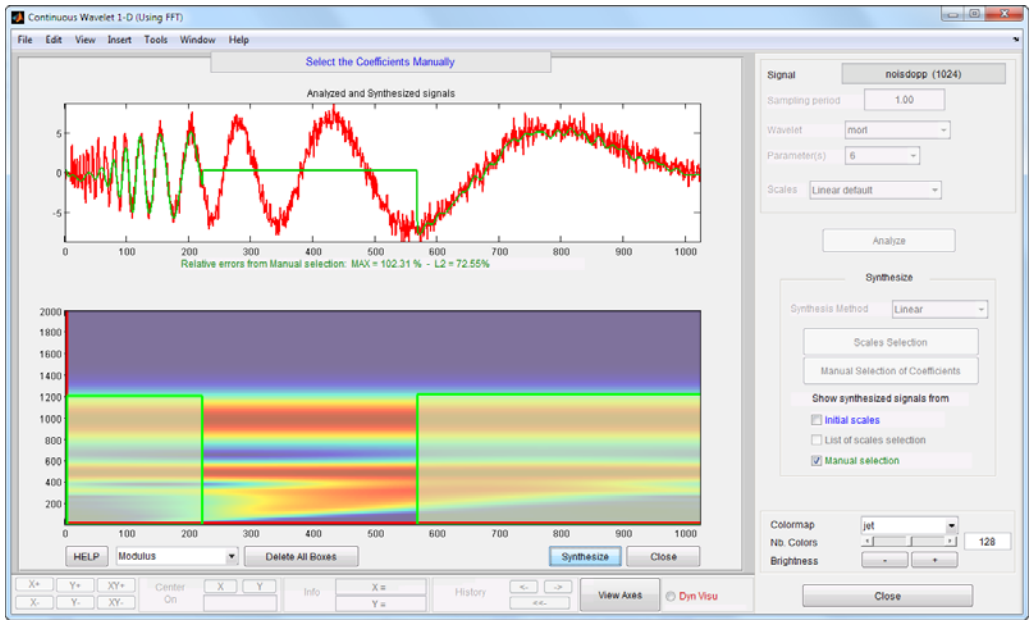


You can change the CWT coefficient view to `Angle`, `Real`, or `Imaginary`.

To select a subset of coefficients, draw a box by left-clicking and dragging the mouse. When you release the mouse button, a semi-transparent box with a green border is superimposed on the plot.



You can place multiple boxes on the same plot. To synthesize a signal based on the selected coefficients, click **Synthesize**.

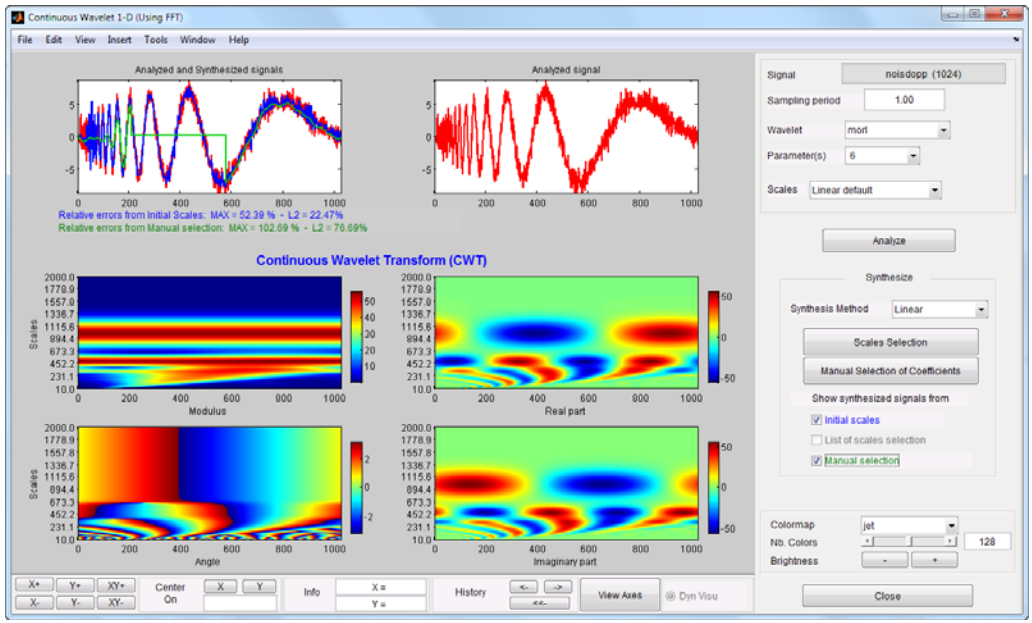


To select, unselect, or delete a box, right-click in the box. A context menu appears that allows you to **select**, **unselect**, or **delete** the box. After you select the coefficients within the box, the border of the box displays in green. When the coefficients within the box are not selected, the border of the box displays in red.

You can move a box by clicking the left mouse button inside the box while simultaneously pressing the `Shift` key. The border of the box changes to yellow, and you can drag the box to the desired location. You must keep the `Shift` key pressed while you are moving the box.

Quit the manual selection mode by clicking the **Close** button.

In the **Show synthesized signals from** panel on the right, you can turn the plot of your synthesized signal on and off by checking and unchecking **Manual selection**.



Using the **File > Save > Synthesized signal** menu, you can save the available synthesized signals.

Using the **File > Save > Decomposition** menu, you can save the wavelet analysis as a `MAT` file.

5

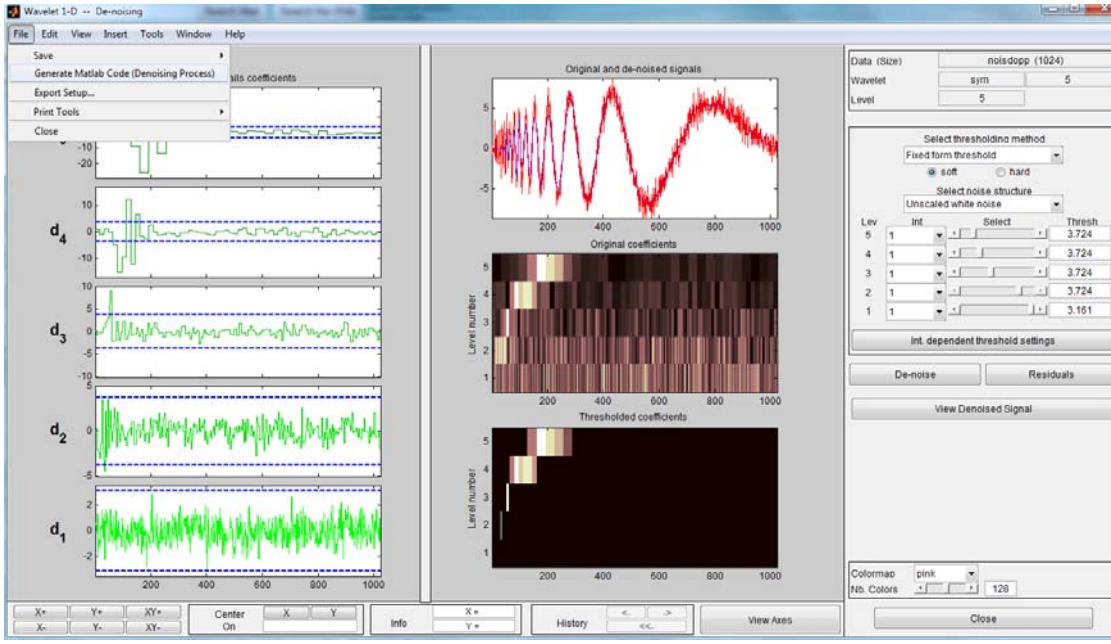
Generating MATLAB Code from Wavelet Toolbox GUI

Generating MATLAB Code for 1-D Decimated Wavelet Denoising and Compression

Wavelet 1-D Denoising

You can generate MATLAB code to reproduce GUI-based 1-D wavelet denoising at the command line. You must perform this operation in the **Wavelet 1-D - - De-noising** tool. You must first denoise your signal before you can enable the **File > Generate Matlab Code (Denoising Process)** operation.

The generated MATLAB code does not include the calculation of the thresholds using `thselect` or `wbmpen`.

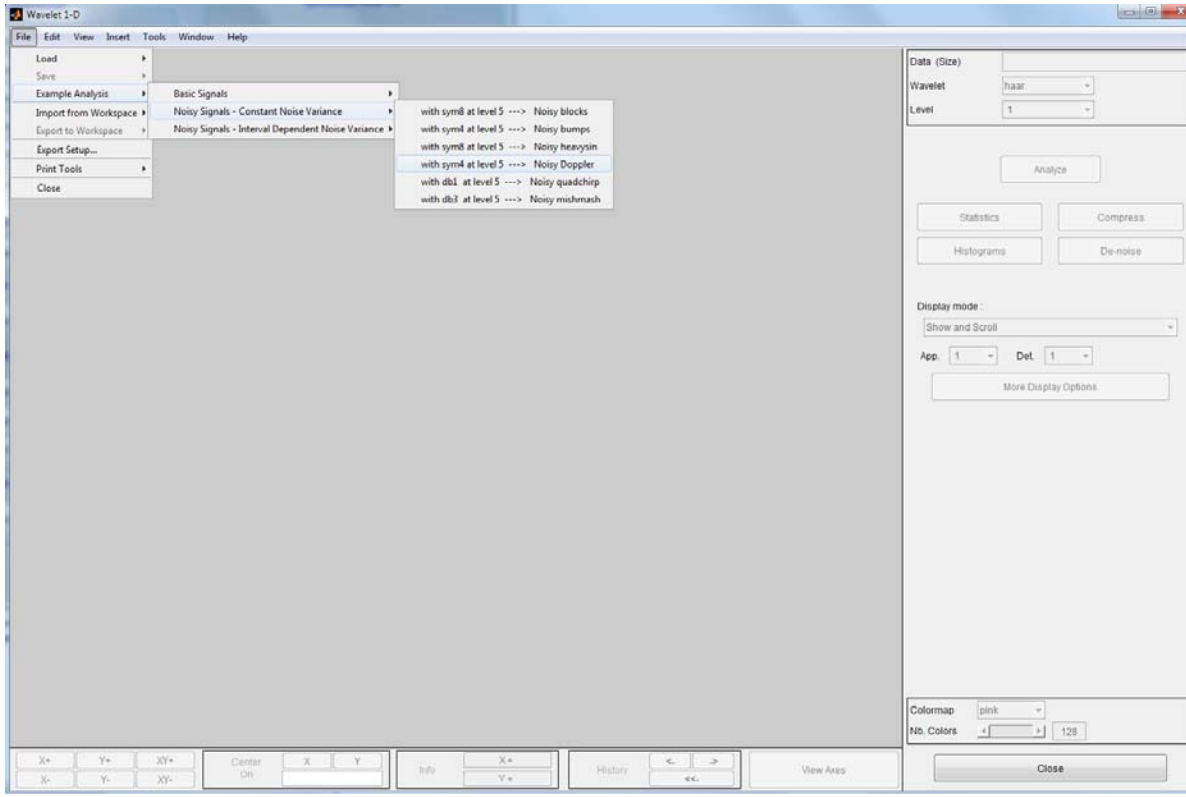


Denoise Doppler Signal

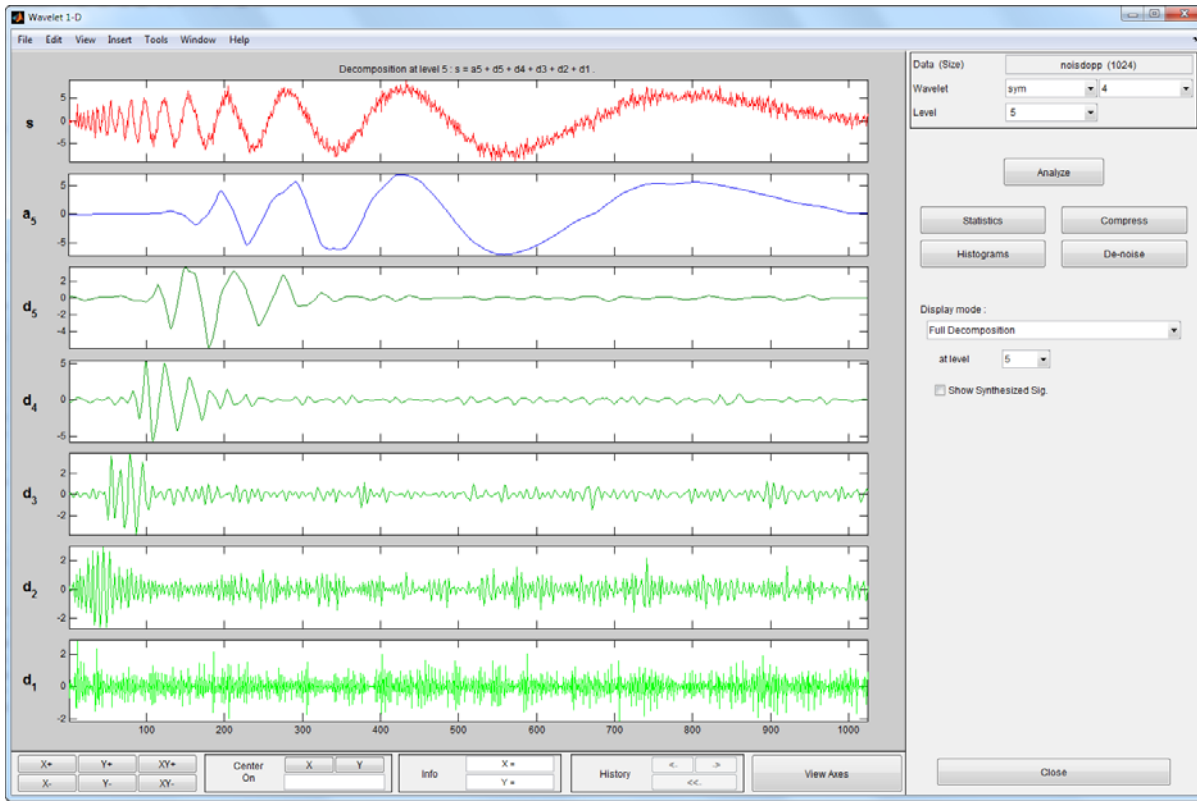
1 Enter `wavemenu` at the MATLAB command prompt.

2 Select **Wavelet 1-D** in the **Wavelet Toolbox Main Menu**.

3 Load the noisy Doppler example analysis. Select **File > Example Analysis > Noisy Signals - Constant Noise Variance > with sym4 at level 5 - -> Noisy Doppler**.

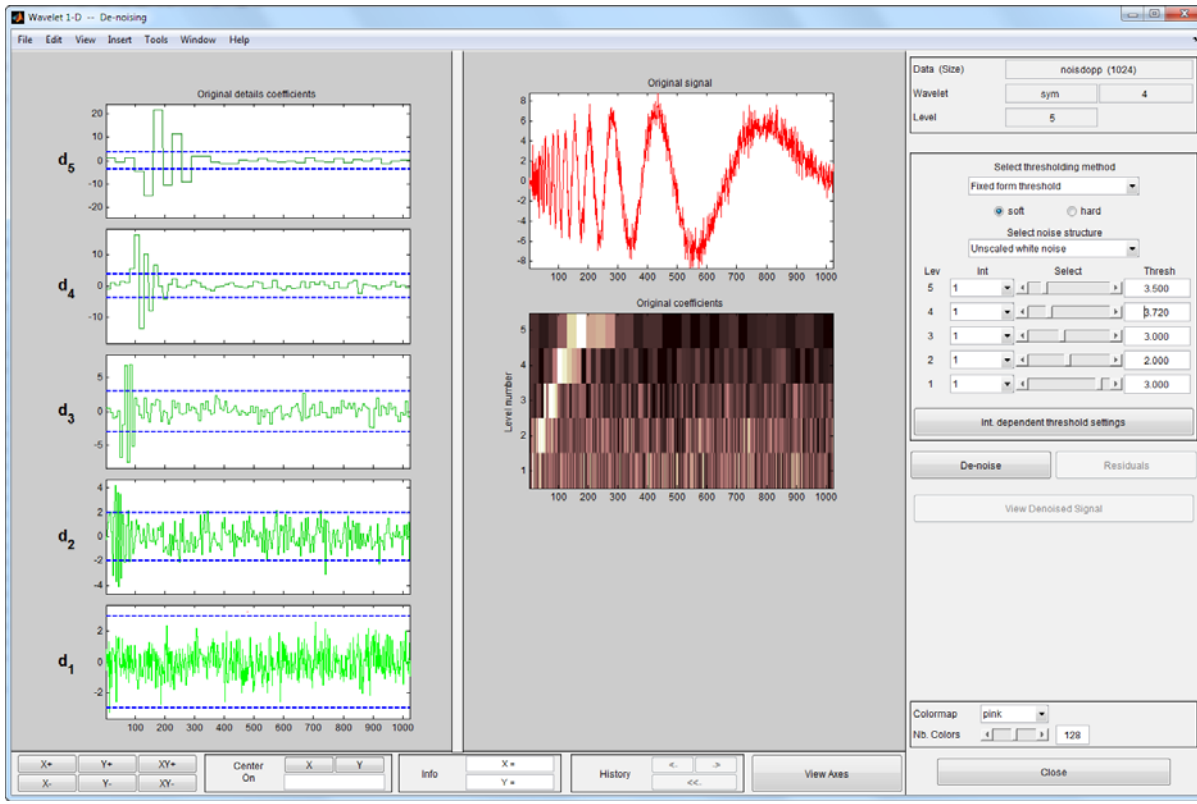


4 Click **De-noise**.



5 In the `Select thresholding method` drop-down menu, select the default `Fixed form threshold`. Use the default **soft** option. Set the thresholds by level as follows:

- level 5 — 3.5
- level 4 — 3.72
- level 3 — 3.0
- level 2 — 2.0
- level 1 — 3.0



Click **De-noise**.

6 Generate the MATLAB code by selecting **File > Generate Matlab Code (Denoising Process)**.

The operation generates the following MATLAB code.

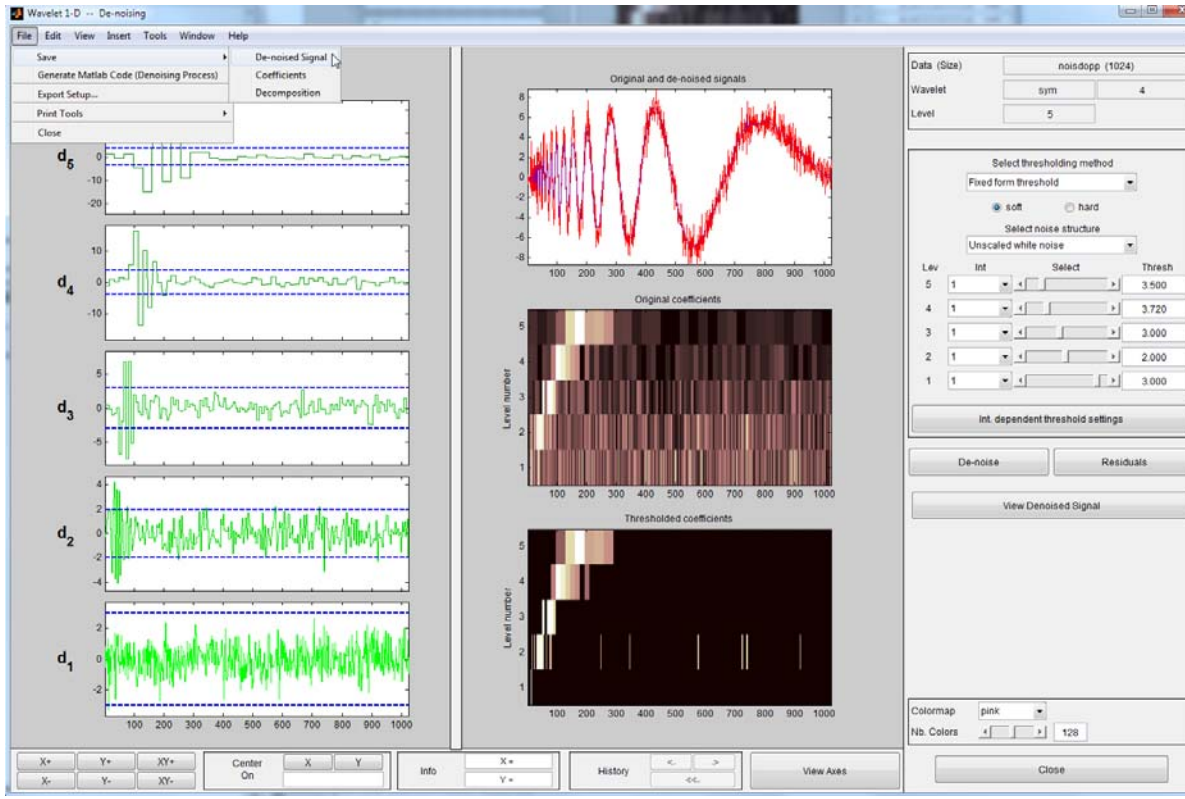
```
function sigDEN = func_denoise_dw1d(SIG)
% FUNC_DENOISE_DW1-D Saved Denoising Process.
%   SIG: vector of data
%   -------------------
%   sigDEN: vector of denoised data


% Analysis parameters.
```

```
%---------------------
wname = 'sym4';
level = 5;

% Denoising parameters.
%----------------------
% meth = 'sqtwolog';
% scal_or_alfa = one;
sorh = 's';    % Specified soft or hard thresholding
thrParams =  [...
    3.00000000 ; ...
    2.00000000 ; ...
    3.00000000 ; ...
    3.72000000 ; ...
    3.50000000   ...
    ];

% Denoise using CMDDENOISE.
%--------------------------
sigDEN = cmddenoise(SIG,wname,level,sorh,NaN,thrParams);
```

7 Save `func_denoise_dw1d.m` in a folder on the MATLAB search path. Execute the following code.

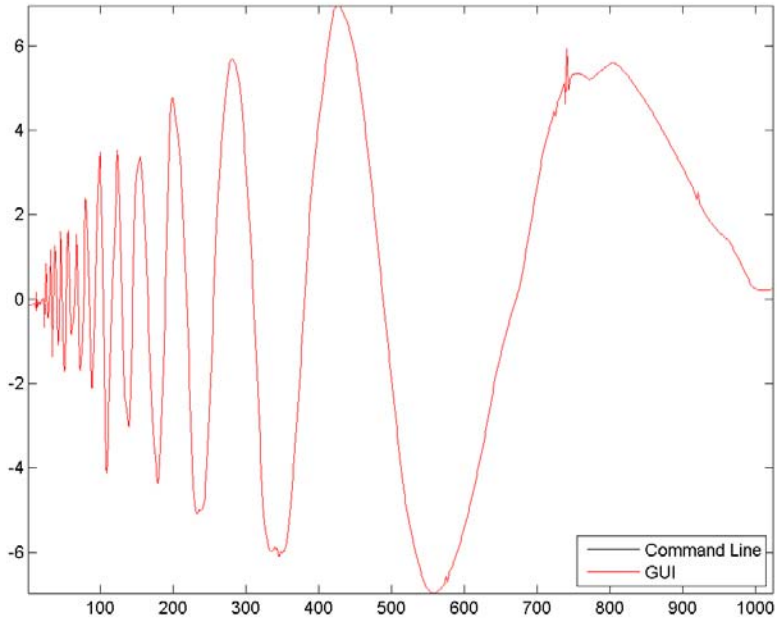
```
load noisdopp;
SIG = noisdopp;
% func_denoise_dw1d.m is generated code
sigDEN = func_denoise_dw1d(SIG);
```

8 Export the denoised signal from the GUI by selecting **File > Save > De-noised Signal**.



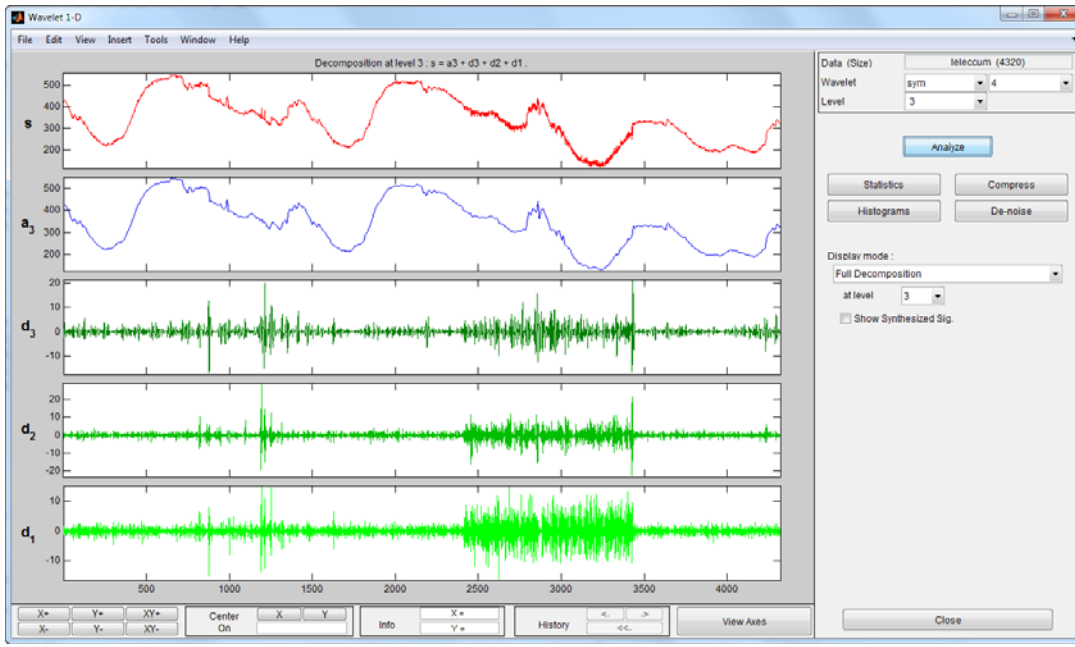
Save the denoised signal as `denoiseddoppler.mat` in a folder on the MATLAB search path. Load `denoiseddoppler.mat` in the MATLAB workspace. Compare `denoiseddoppler` with your command line result.

```
load denoiseddoppler;
plot(sigDEN,'k'); axis tight;
hold on;
plot(denoiseddoppler,'r');
legend('Command Line','GUI','Location','SouthEast');
```



Interval Dependent 1-D Wavelet Denoising

1. Enter `wavemenu` at the MATLAB command prompt.
2. Select **Wavelet 1-D**.
3. Select **File > Load > Signal**, and load `leleccum.mat` from the `matlab/toolbox/wavelet/wavedemo` folder.
4. Select the `sym4` wavelet, and set `Level` equal to 3. Click **Analyze**.



When you inspect the original signal and the finest-scale wavelet coefficients, you see that the noise variance is not constant. In this situation, interval-dependent thresholding is useful. To implement interval-dependent denoising:

1. Click **De-noise**.
2. Under **Select thresholding method**, select `Rigorous SURE`.
3. Select **Int. dependent threshold settings**.
4. In the **Interval Dependent Threshold Settings for Wavelet 1-D** tool, choose **Generate Default Intervals**. Three intervals are created. Click **Propagate** to propagate the intervals to all levels.
5. Click **Close**, and answer **Yes** to `Update Thresholds?`.
6. Select **De-noise**.

7 Generate the MATLAB code by selecting **File > Generate Matlab Code (Denoising Process)**.

The operation generates the following MATLAB code.

```
function sigDEN = func_denoise_dw1d(SIG)
% FUNC_DENOISE_DW1D Saved Denoising Process.
%   SIG: vector of data
%   -------------------
%   sigDEN: vector of denoised data


% Analysis parameters.
%---------------------
wname = 'sym4';
level = 3;

% Denoising parameters.
%----------------------
% meth = 'rigrsure';
% scal_or_alfa = one;
sorh = 's';    % Specified soft or hard thresholding
thrSettings =  {...
    [...
    1.000000000000000   2410.000000000000000      5.659608351110114; ...
    2410.000000000000000   3425.000000000000000   19.721391195242880; ...
    3425.000000000000000   4320.000000000000000    4.907947952868359; ...
    ]; ...
    [...
    1.000000000000000   2410.000000000000000      5.659608351110114; ...
    2410.000000000000000   3425.000000000000000   5.659608351110114; ...
    3425.000000000000000   4320.000000000000000   5.659608351110114; ...
    ]; ...
    [...
    1.000000000000000   2410.000000000000000      5.659608351110114; ...
    2410.000000000000000   3425.000000000000000   5.659608351110114; ...
    3425.000000000000000   4320.000000000000000   5.659608351110114; ...
    ]; ...
    };
```

```
% Denoise using CMDDENOISE.
%-------------------------
sigDEN = cmddenoise(SIG,wname,level,sorh,NaN,thrSettings);
```

8 To avoid confusion with the MATLAB code generated in "Denoise Doppler Signal " on page 5-2, change the function definition line. Change the function definition to:

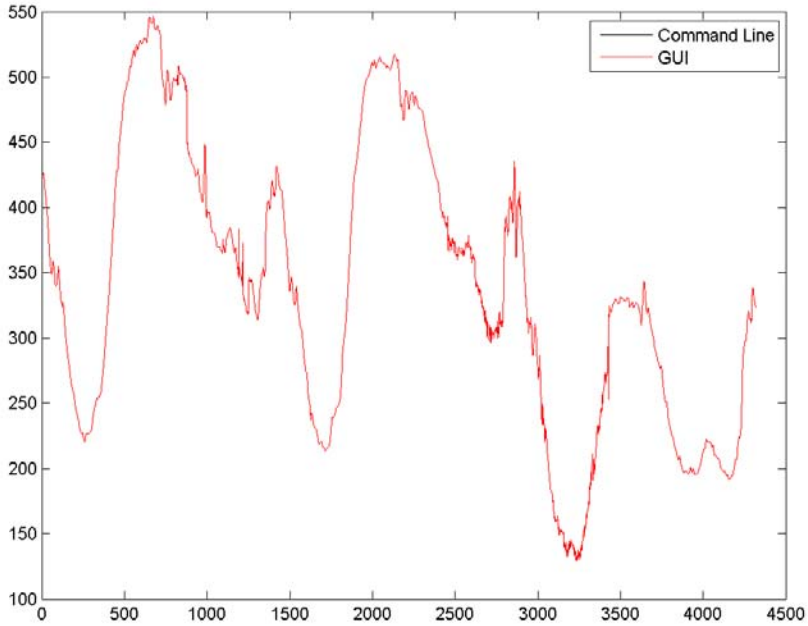
```
function sigDEN = func_IDdenoise_dw1d(SIG)
```

Save the MATLAB program as `func_IDdenoise_dw1d.m` in a folder on the MATLAB search path.

9 Save the denoised signal as `denoisedleleccum.mat` with **File > Save > De-noised Signal** in a folder on the MATLAB search path.

Execute the following code.

```
load leleccum;
load denoisedleleccum;
sigDEN = func_IDdenoise_dw1d(leleccum);
plot(sigDEN,'k');
hold on;
plot(denoisedleleccum,'r');
legend('Command Line','GUI');
norm(sigDEN-denoisedleleccum,2)
```


Command Line
GUI
550
500
450
400
350
300
250
200
150
100
0
500
1000
1500
2000
2500
3000
3500
4000
4500

Generating MATLAB Code for 2-D Decimated Wavelet Denoising and Compression

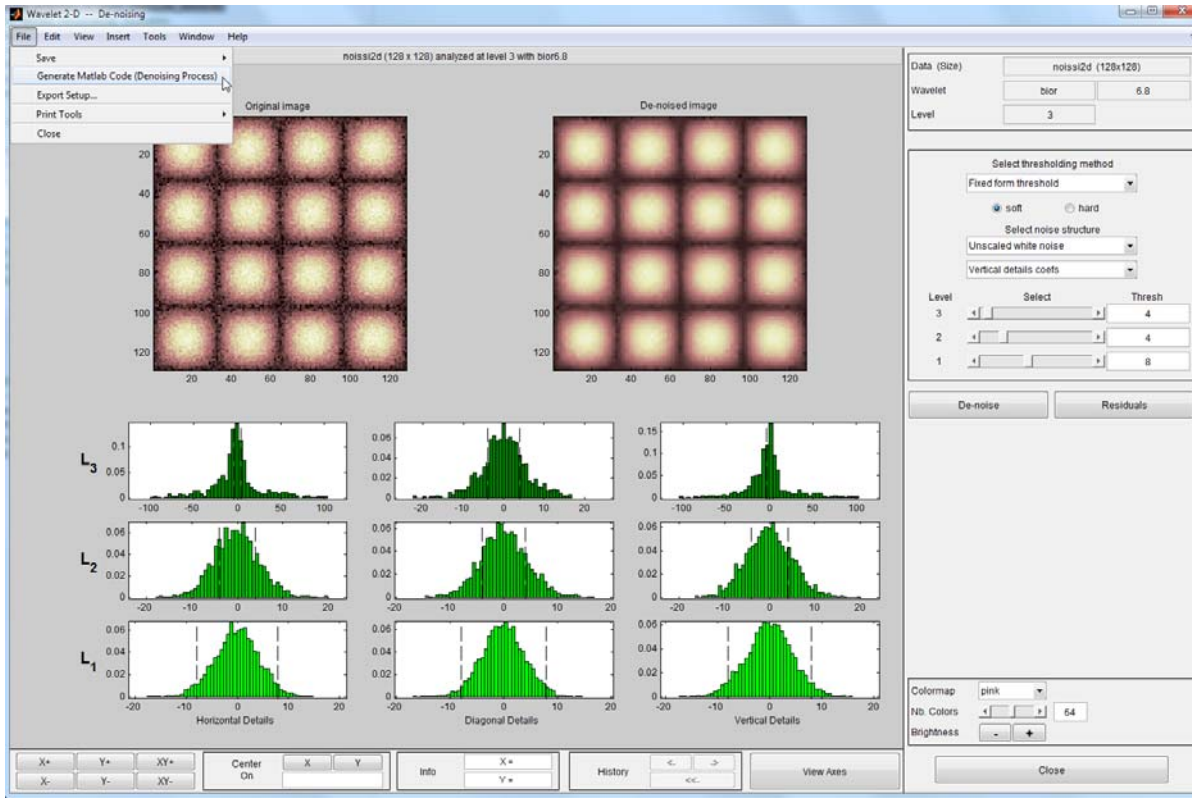
In this section...

2-D Decimated Discrete Wavelet Transform Denoising

You can generate MATLAB code to reproduce GUI-based 2-D decimated wavelet denoising at the command line. You must perform this operation in the **Wavelet 2-D – –De-noising** tool. You must first denoise your image before you can enable the **File > Generate Matlab Code (Denoising Process)** operation.

1. Enter `wavemenu` at the MATLAB command prompt.
2. Select **Wavelet 2-D**.
3. Load the `Noisy SinSin` example indexed image. Using the default biorthogonal wavelet and level 3 decomposition, click **De-noise**.
4. In the `Select thresholding method` drop-down menu, select the default `Fixed form threshold` and **soft** options. Use the default `Unscaled white noise`. Set the thresholds by level for the horizontal, diagonal, and vertical coefficients as follows:
   - Level 3 — 4
   - Level 2 — 4
   - Level 1 — 8

   Enter these thresholds for the horizontal, diagonal, and vertical coefficients.
5. Select **De-noise**.
6. Generate the MATLAB code with **File > Generate Matlab Code (Denoising Process)**.



The operation generates the following MATLAB code.

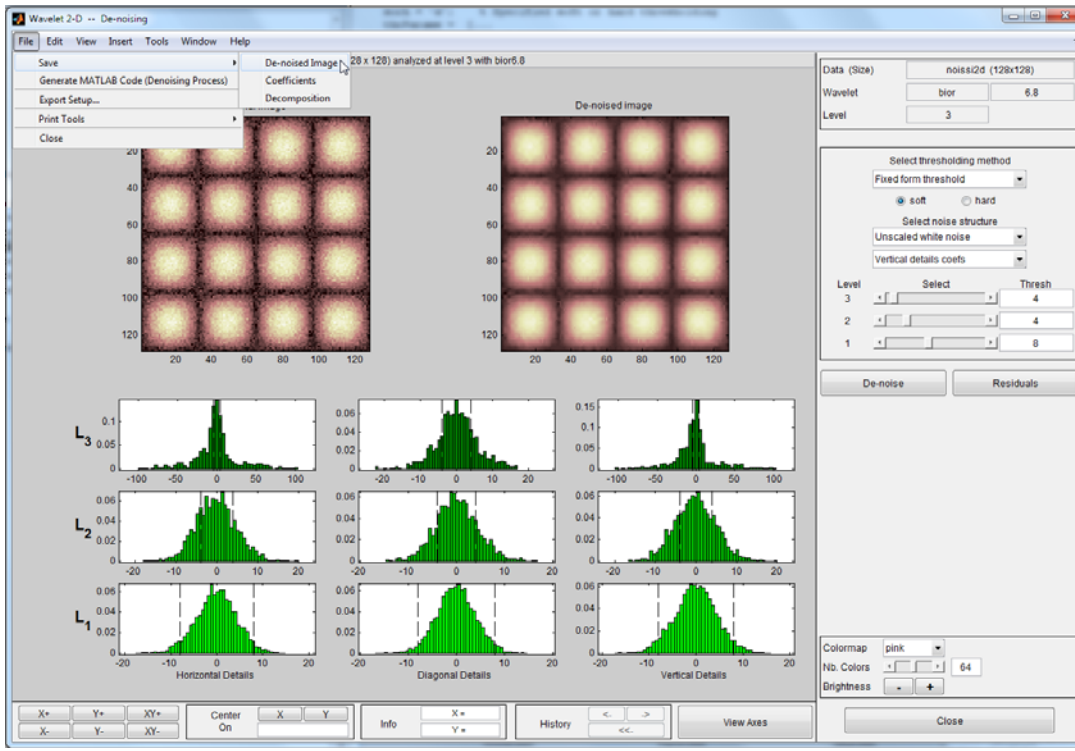
```
function [XDEN,cfsDEN,dimCFS] = func_denoise_dw2d(X)
% FUNC_DENOISE_DW2-D Saved Denoising Process.
%   X: matrix of data
%   -----------------
%   XDEN: matrix of denoised data
%   cfsDEN: decomposition vector (see WAVEDEC2)
%   dimCFS: corresponding bookkeeping matrix


% Analysis parameters.
%---------------------
wname = 'bior6.8';
```

```
level = 3;

% Denoising parameters.
%----------------------
% meth = 'sqtwolog';
% scal_OR_alfa = one;
sorh = 's';    % Specified soft or hard thresholding
thrParams =  [...
    8.00000000     4.00000000     4.00000000 ; ...
    8.00000000     4.00000000     4.00000000 ; ...
    8.00000000     4.00000000     4.00000000   ...
    ];
roundFLAG = true;

% Denoise using CMDDENOISE.
%--------------------------
[coefs,sizes] = wavedec2(X,level,wname);
[XDEN,cfsDEN,dimCFS] = wdencmp('lvd',coefs,sizes, ...
    wname,level,thrParams,sorh);

if roundFLAG , XDEN = round(XDEN); end
if isequal(class(X),'uint8') , XDEN = uint8(XDEN); end
```

7 Save `func_denoise_dw2d.m` in a folder on the MATLAB search path, and execute the following code.

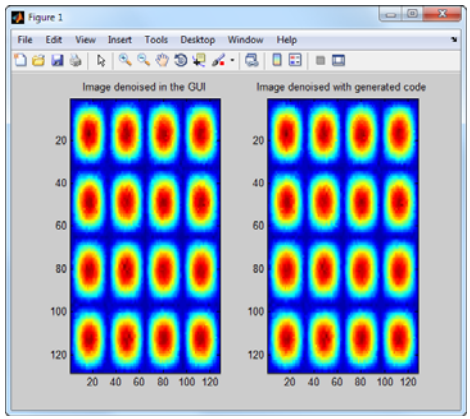
```
load noissi2d.mat;
noissi2d = X;
[XDEN,cfsDEN,dimCFS] = func_denoise_dw2d(noissi2d);
```

8 Save your denoised image in a folder on the MATLAB search path as `denoisedsin.mat`.



Load the denoised image in the MATLAB workspace. Compare the result with your generated code.

```
load denoisedsin.mat;
% denoised image loaded in variable X
subplot(121);
imagesc(X); title('Image denoised in the GUI');
subplot(122);
imagesc(XDEN); title('Image denoised with generated code');
% Norm of the difference is zero
norm(XDEN-X,2)
```



2-D Decimated Discrete Wavelet Transform Compression

You can generate MATLAB code to reproduce GUI-based 2-D decimated wavelet compression at the command line. You must perform this operation in the **Wavelet 2-D --Compression** tool. You must first compress your image before you can enable the **File > Generate Matlab Code (Compression Process)** operation.

1. Enter `wavemenu` at the MATLAB command prompt.
2. Select **Wavelet 2-D**.
3. Select **File > Load > Image** and load the `detfingr.mat` indexed image from the `matlab/toolbox/wavelet/wavedemo` folder. When the **Loading an Image** dialog appears, select **No** to load the grayscale image.
4. Select the `bior3.5` wavelet, and set `Level` to 3.
5. Click **Analyze**, then click **Compress**.
6. Using the default `Global thresholding`, set `Select thresholding method` to `Bal.sparsity-norm (sqrt)`.
7. Click **Compress**.

8 **File > Generate Code (Compression Process)** generates the following code.

```
function [XCMP,cfsCMP,dimCFS] = func_compress_dw2d(X)
% FUNC_COMPRESS_DW2D Saved Compression Process.
%   X: matrix of data
%   -----------------
%   XCMP: matrix of compressed data
%   cfsCMP: decomposition vector (see WAVEDEC2)
%   dimCFS: corresponding bookkeeping matrix

% Analysis parameters.
%---------------------
wname = 'bior3.5';
level = 3;

% Compression parameters.
%------------------------
% meth = 'sqrtbal_sn';
sorh = 'h';    % Specified soft or hard thresholding
thrSettings = 10.064453124999996;
roundFLAG = true;

% Compression using WDENCMP.
%---------------------------
[coefs,sizes] = wavedec2(X,level,wname);
[XCMP,cfsCMP,dimCFS] = wdencmp('gbl',coefs,sizes, ...
    wname,level,thrSettings,sorh,1);
if roundFLAG , XCMP = round(XCMP); end
if isequal(class(X),'uint8') , XCMP = uint8(XCMP); end
```

9 Save the MATLAB program, `func_compress_dw2d.m`, in a folder on the MATLAB search path. Execute the following code at the command line.

```
load detfingr.mat;
% Image data is in X
[XCMP,cfsCMP,dimCFS] = func_compress_dw2d(X);
```

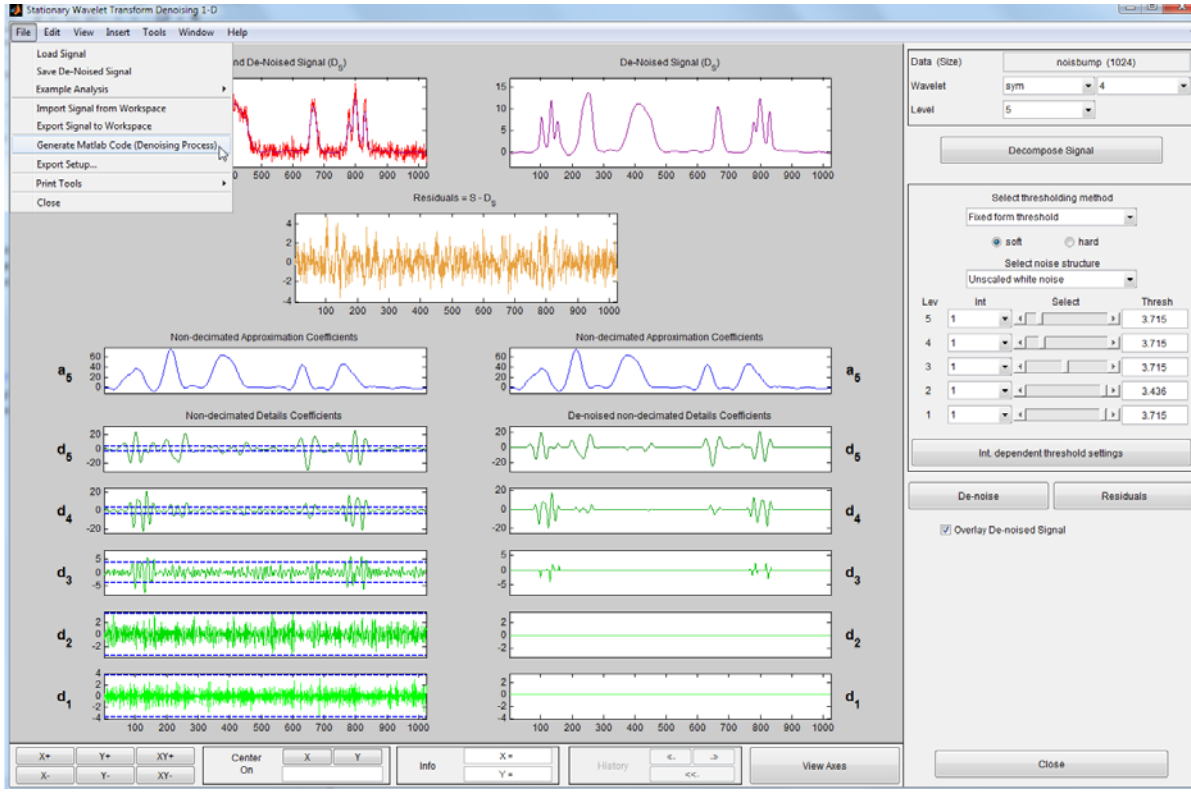
10 Save the compressed image from the **Wavelet 2-D - - Compression** tool in a folder on the MATLAB search path. Use **File > Save > Compressed**

Image, and name the file `compressed_fingerprint.mat`. Execute the following code.

```
load compressed_fingerprint.mat;
% Image data is in X
norm(XCMP-X,2)
```

Generating MATLAB Code for 1-D Stationary Wavelet Denoising

You can generate MATLAB code to reproduce GUI-based 1-D nondecimated (stationary) wavelet denoising at the command line. You must perform this operation in the **Stationary Wavelet Transform Denoising 1-D** tool. You must first denoise your signal before you can enable the **File > Generate Matlab Code (Denoising Process)** operation.

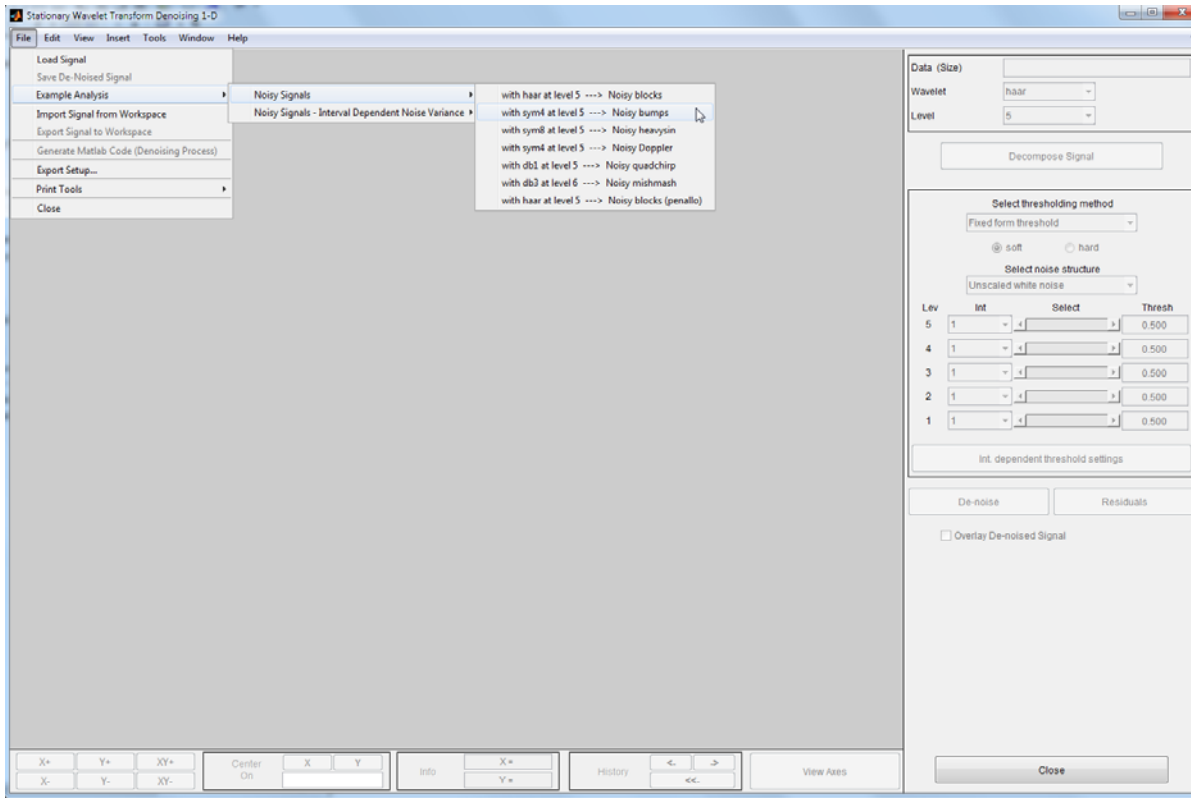


1-D Stationary Wavelet Transform Denoising

1 Enter `wavemenu` at the MATLAB command prompt.

2 Select **SWT Denoising 1-D**.

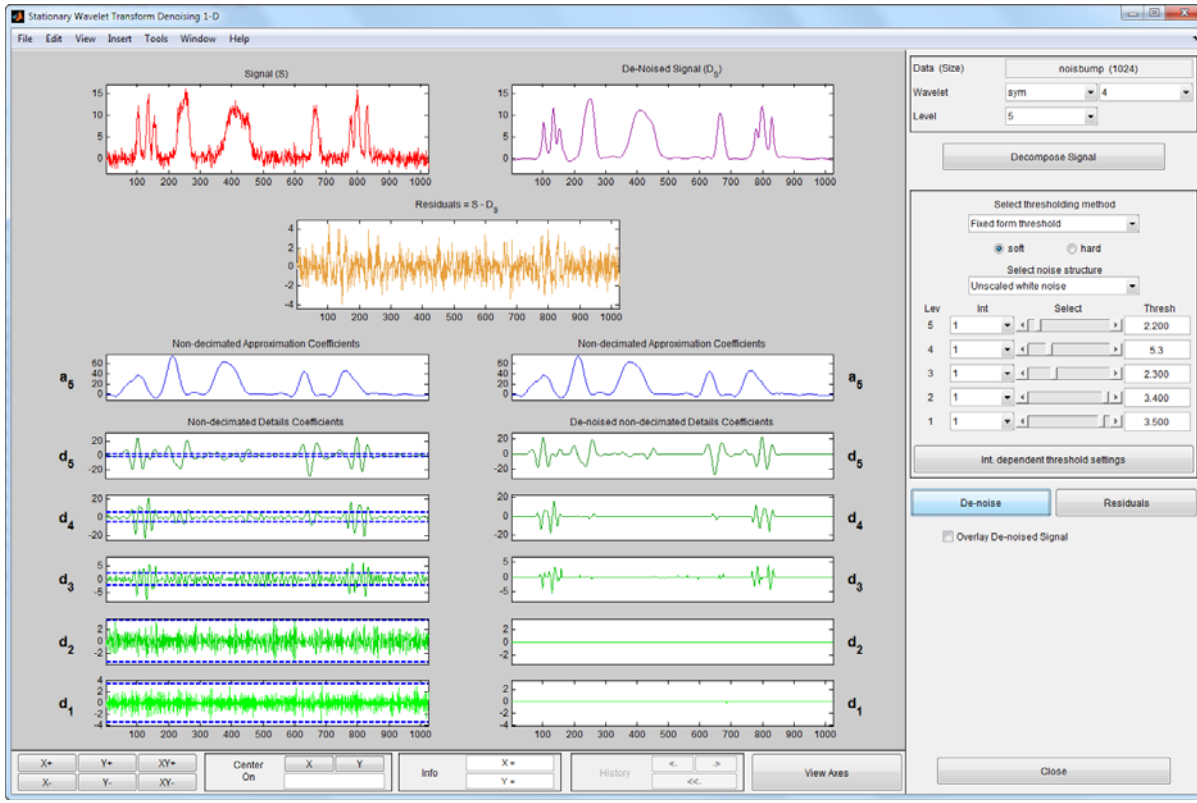
3 Load the `Noisy bumps` example. Select **File > Example Analysis > Noisy Signals > with sym4 at level 5 - - -> Noisy bumps**



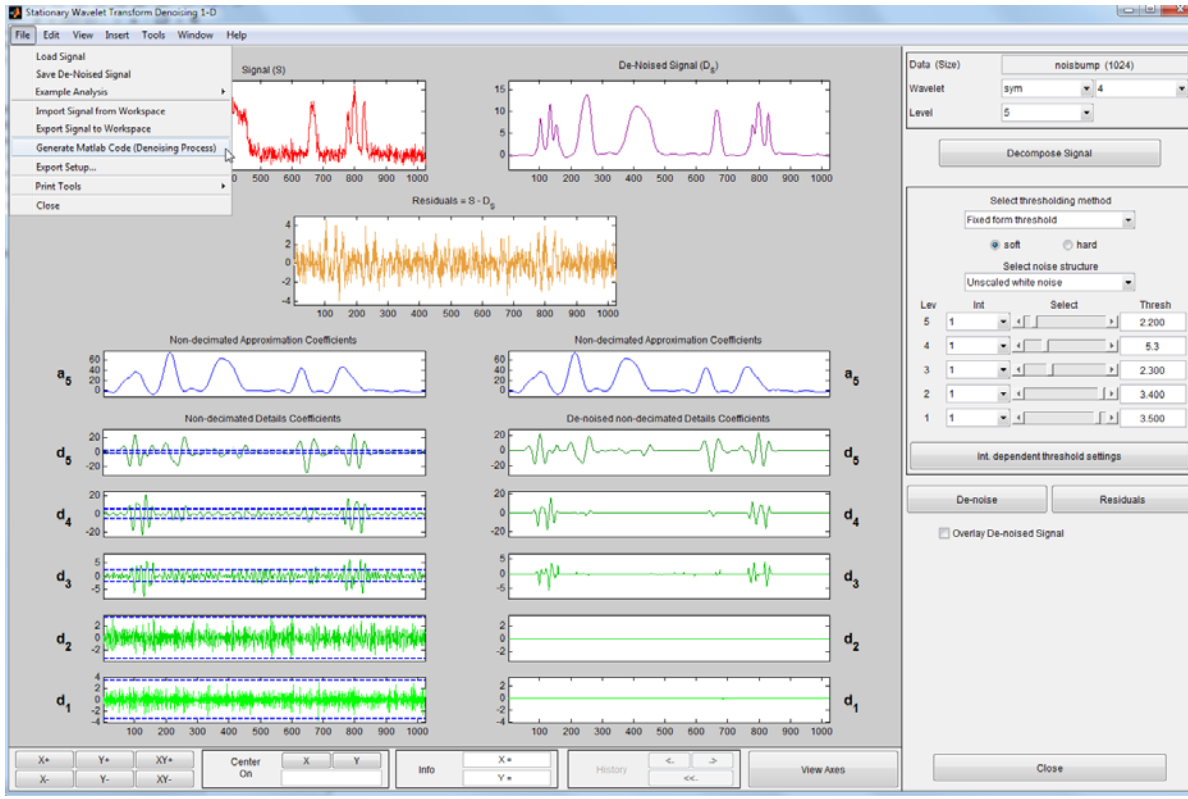
4 Set the thresholds as follows:

- Level 1 — 3.5
- Level 2 — 3.4
- Level 3 — 2.3
- Level 4 — 5.3
- Level 5 — 2.2

Click **De-noise**.



5 Generate the MATLAB code with **File > Generate Matlab Code (Denoising Process)**.



The operation generates the following MATLAB code.

```
function [sigDEN,wDEC] = func_denoise_sw1d(SIG)
% FUNC_DENOISE_SW1-D Saved Denoising Process.
%   SIG: vector of data
%   -------------------
%   sigDEN: vector of denoised data
%   wDEC: stationary wavelet decomposition

% Analysis parameters.
%---------------------
wname = 'sym4';
level = 5;
```

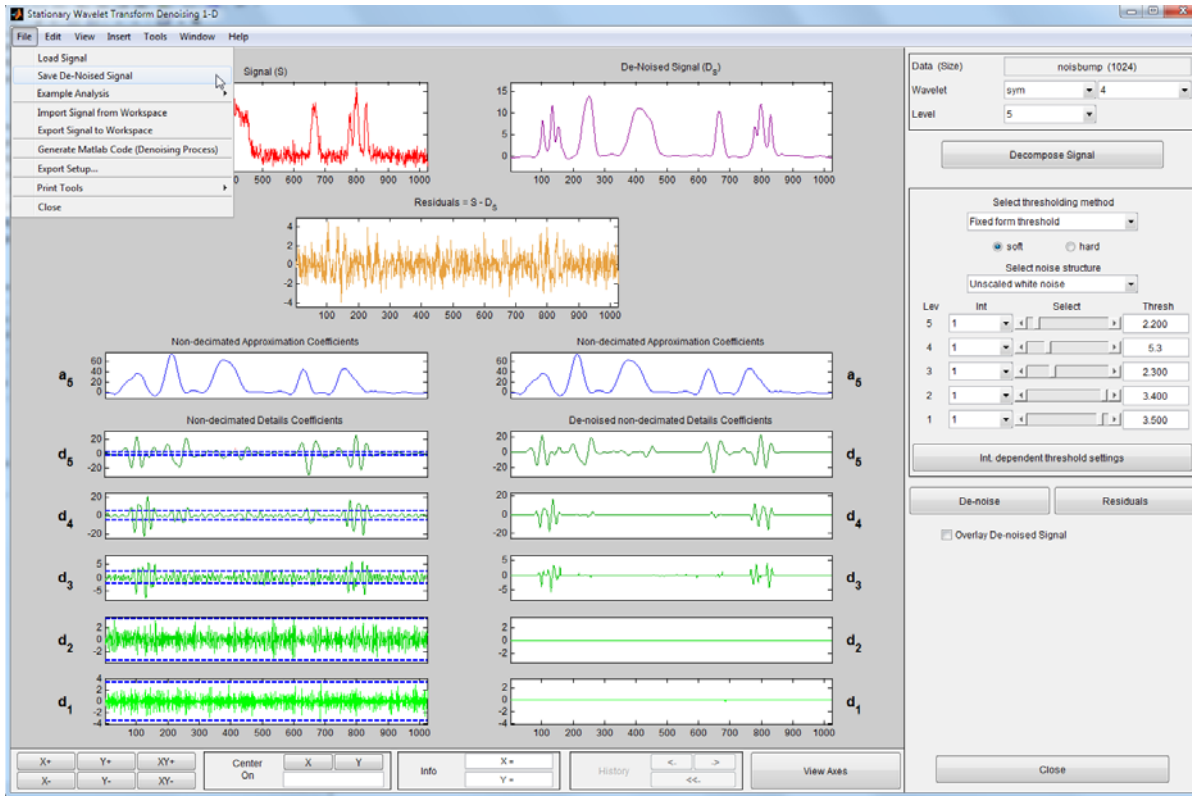
```
% Denoising parameters.
%----------------------
% meth = 'sqtwolog';
% scal_OR_alfa = one;
sorh = 's';    % Specified soft or hard thresholding
thrParams =  {...
    [...
    1.00000000  1024.00000000     3.50000000; ...
    ]; ...
    [...
    1.00000000  1024.00000000     3.40000000; ...
    ]; ...
    [...
    1.00000000  1024.00000000     2.30000000; ...
    ]; ...
    [...
    1.00000000  1024.00000000     5.29965570; ...
    ]; ...
    [...
    1.00000000  1024.00000000     2.20000000; ...
    ]; ...
    };

% Decompose using SWT.
%----------------------
wDEC = swt(SIG,level,wname);

% Denoise.
%---------
len = length(SIG);
for k = 1:level
    thr_par = thrParams{k};
    if ~isempty(thr_par)
        NB_int = size(thr_par,1);
        x      = [thr_par(:,1) ; thr_par(NB_int,2)];
        x      = round(x);
        x(x<1) = 1;
        x(x>len) = len;
        thr = thr_par(:,3);
        for j = 1:NB_int
```

```
            if j==1 , d_beg = 0; else d_beg = 1; end
            j_beg = x(j)+d_beg;
            j_end = x(j+1);
            j_ind = (j_beg:j_end);
            wDEC(k,j_ind) = wthresh(wDEC(k,j_ind),sorh,thr(j));
        end
    end
end

% Reconstruct the denoise signal using ISWT.
%-------------------------------------------
sigDEN = iswt(wDEC,wname);
```

6 Save `func_denoise_sw1d.m` in a folder on the MATLAB search path. Execute the following code.

```
load noisbump.mat;
[sigDEN,wDEC] = func_denoise_sw1d(noisbump);
```

7 Select **File > Save De-noised Signal**, and save the denoised signal as `denoisedbumps.mat` in a folder on the MATLAB search path.

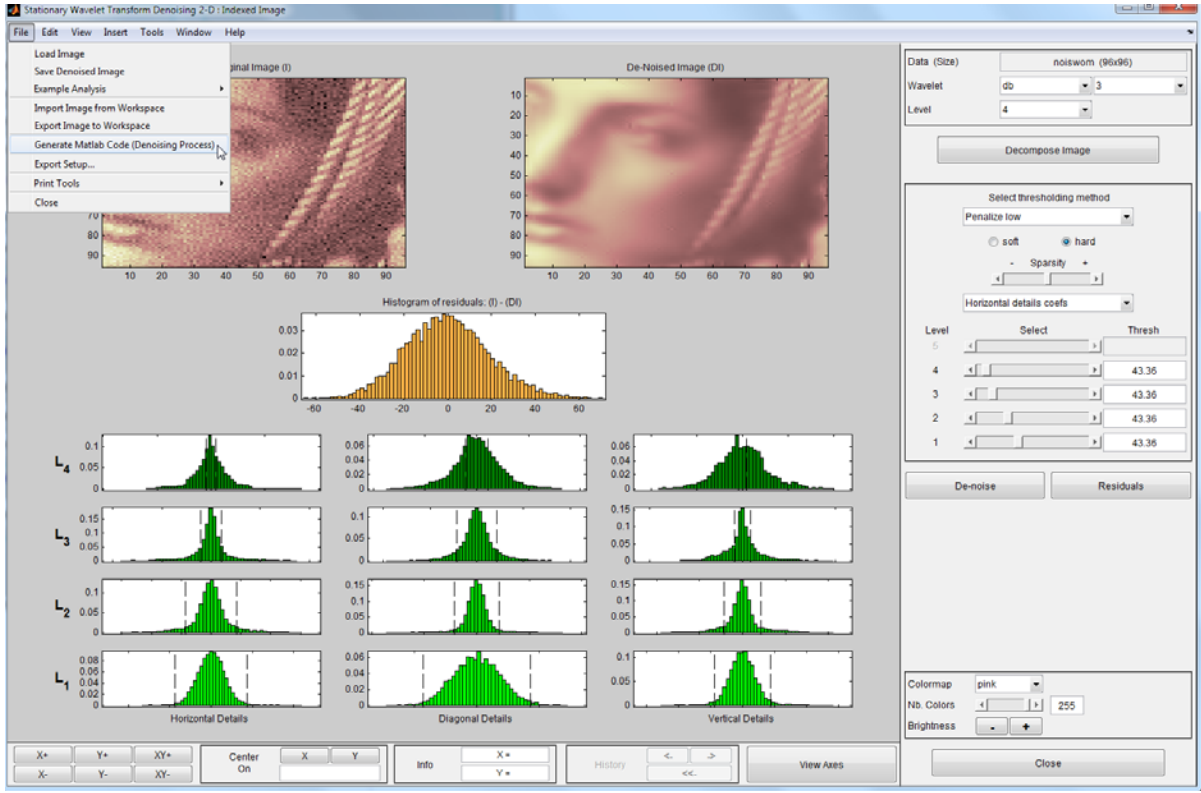


Execute the following code.

```
load denoisedbump.mat;
plot(sigDEN,'k'); axis tight;
hold on;
plot(denoisedbump,'r');
% norm of the difference
norm(sigDEN-denoisedbump,2)
```

Generating MATLAB Code for 2-D Stationary Wavelet Denoising

You can generate MATLAB code to reproduce GUI-based 2-D stationary wavelet denoising at the command line. You can generate code to denoise both indexed and truecolor images. You must perform this operation in the **SWT Denoising 2-D** tool. You must first denoise your image before you can enable the **File > Generate Matlab Code (Denoising Process)** operation.



2-D Stationary Wavelet Transform Denoising

1 Enter `wavemenu` at the MATLAB command prompt.

2. Select **SWT Denoising 2-D**.

3. Select **File > Load Image**, and load `noiswom.mat` from the `matlab/toolbox/wavelet/wavedemo` folder.

   Choose `No` when prompted to use the grayscale image.

4. Select the `db4` wavelet, and set the **Level** to `5`.

5. Click **Decompose Image**.

6. Use the default `soft` thresholding method with `Fixed form threshold` and `Unscaled white noise` for **Select noise structure**.

7. Set the following thresholds for the horizontal, diagonal, and vertical details. Ensure that you set the thresholds for the three detail coefficient types.

   - Level 1 — 5
   - Level 2 — 4
   - Level 3 — 3
   - Level 4 — 2
   - Level 5 — 1

8. Click **De-noise**.

9. Select **File > Generate Matlab Code (Denoising Process)**.

   The operation generates the following MATLAB code.

```
function [XDEN,wDEC] = func_denoise_sw2d(X)
% FUNC_DENOISE_SW2D Saved Denoising Process.
%   X: matrix of data
%   -----------------
%   XDEN: matrix of denoised data
%   wDEC: stationary wavelet decomposition

% Analysis parameters.
%---------------------
wname = 'db2';
```

```
level = 5;

% Denoising parameters.
%----------------------
% meth = 'sqtwolog';
% scal_OR_alfa = one;
sorh = 's';    % Specified soft or hard thresholding
thrSettings =  [...
    1.0000      2.0000      3.0000      4.0000      5.0000 ; ...
    1.0000      2.0000      3.0000      4.0000      5.0000 ; ...
    1.0000      2.0000      3.0000      4.0000      5.0000   ...
    ];
roundFLAG = false;

% Decompose using SWT2.
%----------------------
wDEC = swt2(X,level,wname);

% Denoise.
%---------
permDir = [1 3 2];
for j = 1:level
    for kk=1:3
        ind = (permDir(kk)-1)*level+j;
        thr = thrSettings(kk,j);
        wDEC(:,:,ind) = wthresh(wDEC(:,:,ind),sorh,thr);
    end
end

% Reconstruct the denoise signal using ISWT2.
%--------------------------------------------
XDEN = iswt2(wDEC,wname);
if roundFLAG , XDEN = round(XDEN); end
```

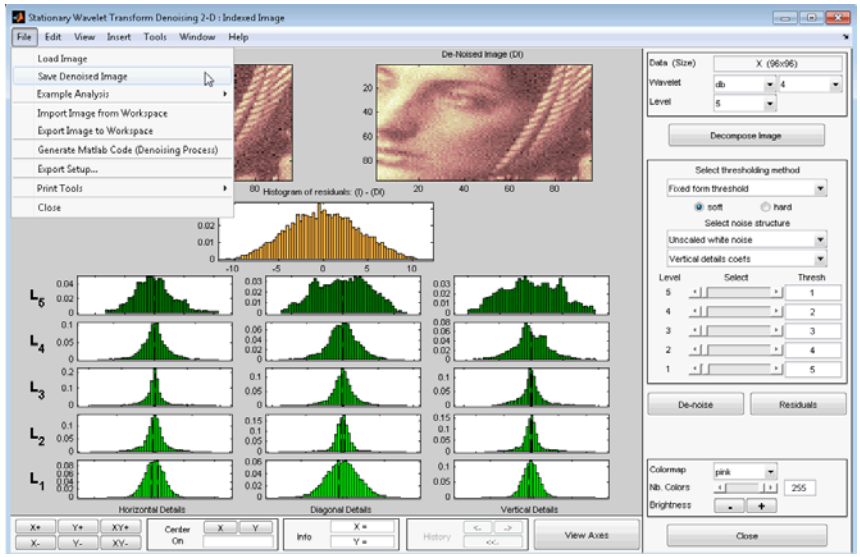
10 Save this MATLAB program as `func_denoise_sw2d.m` in a folder on the MATLAB search path.

Execute the following code.

```
load noiswom
```

```
[XDEN,wDEC] = func_denoise_sw2d(X);
```

11 Save the denoised image as `denoisedwom.mat` in a folder on the MATLAB search path.



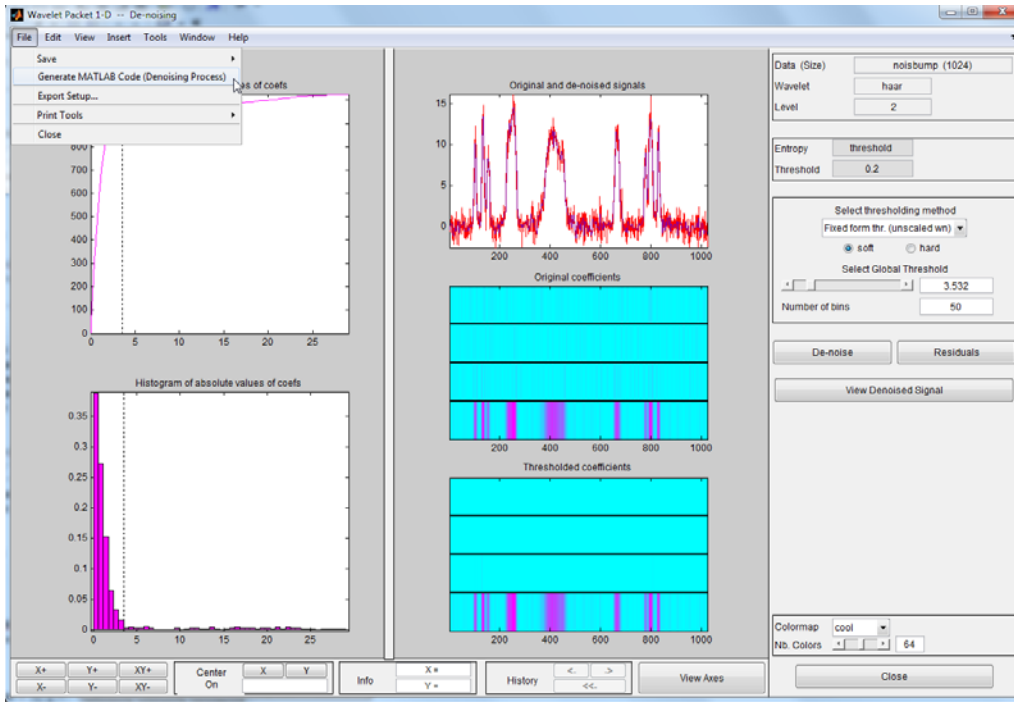
12 Execute the following code.

```
load denoisedwom
% Compare the GUI and command line results
imagesc(X); title('GUI Operation'); colormap(gray);
figure;
imagesc(XDEN); title('Command Line Operation');
colormap(gray);
norm(XDEN-X,2)
```

Generating MATLAB Code for 1-D Wavelet Packet Denoising and Compression

1-D Wavelet Packet Denoising

You can generate MATLAB code to reproduce GUI–based 1-D wavelet packet denoising at the command line. You must perform this operation in the **Wavelet Packet 1-D - - De-noising** tool. You must first denoise your signal before you can enable the **File > Generate Matlab Code (Denoising Process)** operation.



1 Enter `wavemenu` at the MATLAB command prompt.

2 Select **Wavelet Packet 1-D**.

3 Select **File > Load Signal** and load `noisbump.mat` from the `matlab/toolbox/wavelet/wavedemo` folder.

4. Select the `db4` wavelet, and set the **Level** to `4`. Accept the default value `Shannon` for **Entropy**.

5. Click **Analyze**.

6. Click **De-noise**.

7. Under **Select thresholding method**, accept the default `Fixed form thr. (unscaled wn)` with the **soft** radio button enabled.

   Set **Select Global Threshold** to 2.75.

8. Click **De-noise**.

9. Select **File > Generate Matlab Code (Denoising Process)**

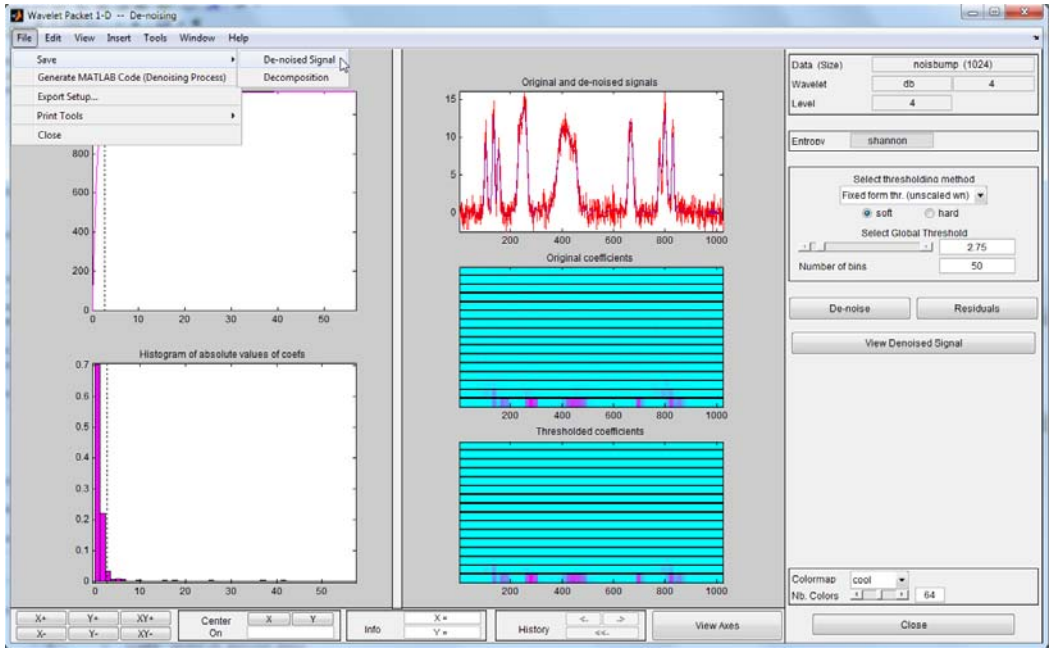
The operation generates the following MATLAB code.

```
function [sigDEN,wptDEN] = func_denoise_wp1d(SIG)
% FUNC_DENOISE_WP1D Saved Denoising Process.
%   SIG: vector of data
%   -------------------
%   sigDEN: vector of denoised data
%   wptDEN: wavelet packet decomposition (wptree object)


% Analysis parameters.
%---------------------
Wav_Nam = 'db4';
Lev_Anal = 4;
Ent_Nam = 'shannon';
Ent_Par = 0;

% Denoising parameters.
%----------------------
% meth = 'sqtwologuwn';
sorh = 's';    % Specified soft or hard thresholding
thrSettings = {sorh,'nobest',2.750000000000000,1};

% Decompose using WPDEC.
%----------------------
```

```
wpt = wpdec(SIG,Lev_Anal,Wav_Nam,Ent_Nam,Ent_Par);

% Nodes to merge.
%-----------------
n2m = [];
for j = 1:length(n2m)
    wpt = wpjoin(wpt,n2m(j));
end

% Denoise using WPDENCMP.
%------------------------
[sigDEN,wptDEN] = wpdencmp(wpt,thrSettings{:});
```

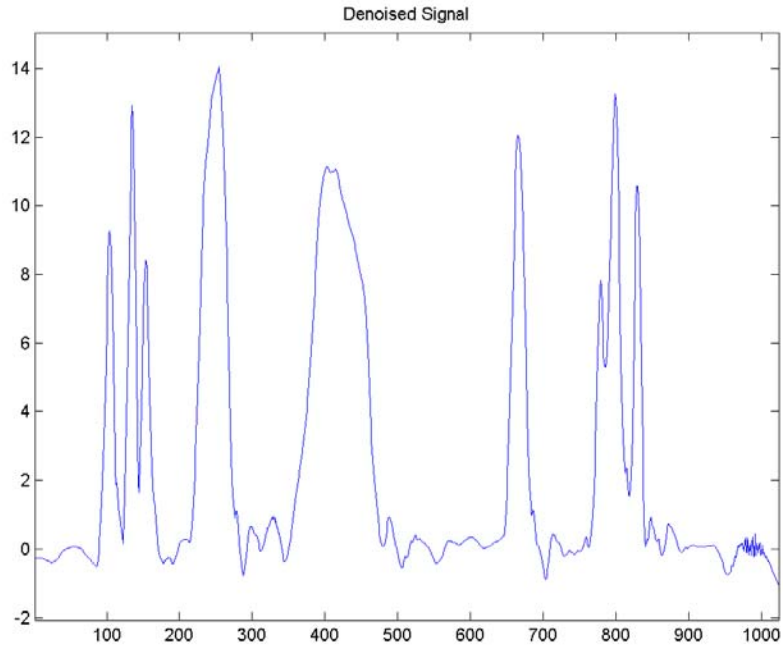
Save `func_denoise_wp1d.m` in a folder on the MATLAB search path.

Save the denoised signal from the **Wavelet Packet 1-D - - De-noising** tool as `wp_denoisedbump.mat` in a folder on the MATLAB search path.



Execute the following code.

```
load noisbump;
[sigDEN,wptDEN] = func_denoise_wp1d(noisbump);
load wp_denoisedbump;
plot(sigDEN); title('Denoised Signal');
axis([1 1024 min(sigDEN)-1 max(sigDEN+1)]);
norm(sigDEN-wp_denoisedbump,2)
```

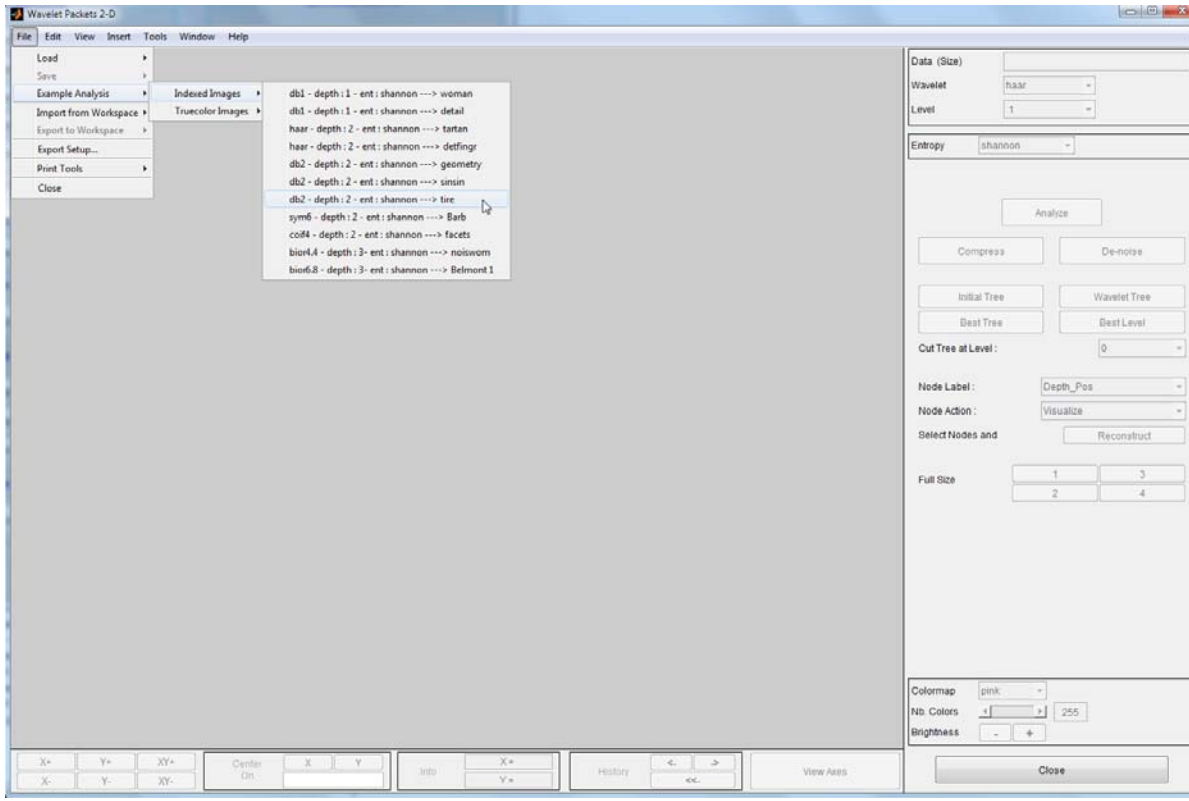


Generating MATLAB Code for 2-D Wavelet Packet Denoising and Compression

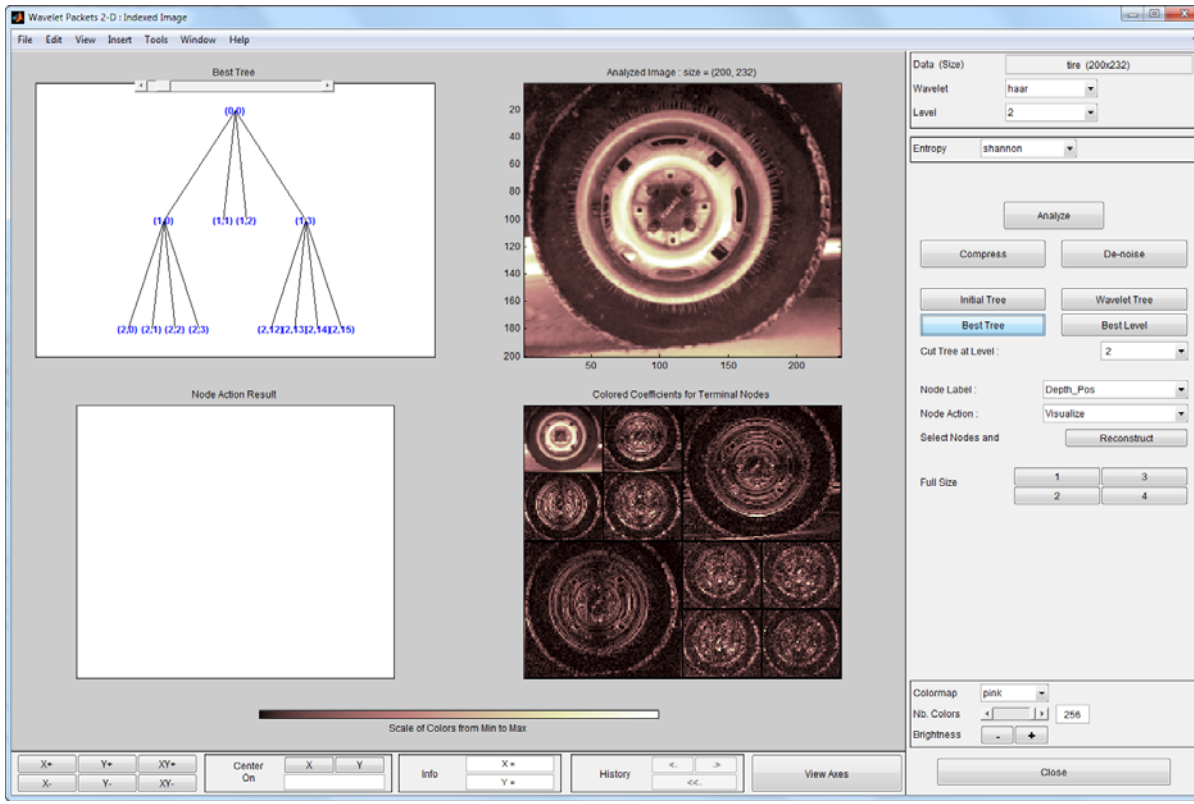
2-D Wavelet Packet Compression

You can generate MATLAB code to reproduce GUI–based 2-D wavelet packet compression at the command line. You must perform this operation in the **Wavelet 2-D - - Compression** tool. You must first compress your image before you can enable the **File > Generate Matlab Code (Compression Process)** operation.

1. Enter `wavemenu` at the MATLAB command prompt.
2. Select **Wavelet Packet 2-D**.
3. Select **File > Load > Example Analysis > Indexed Images**, and load the `tire`.



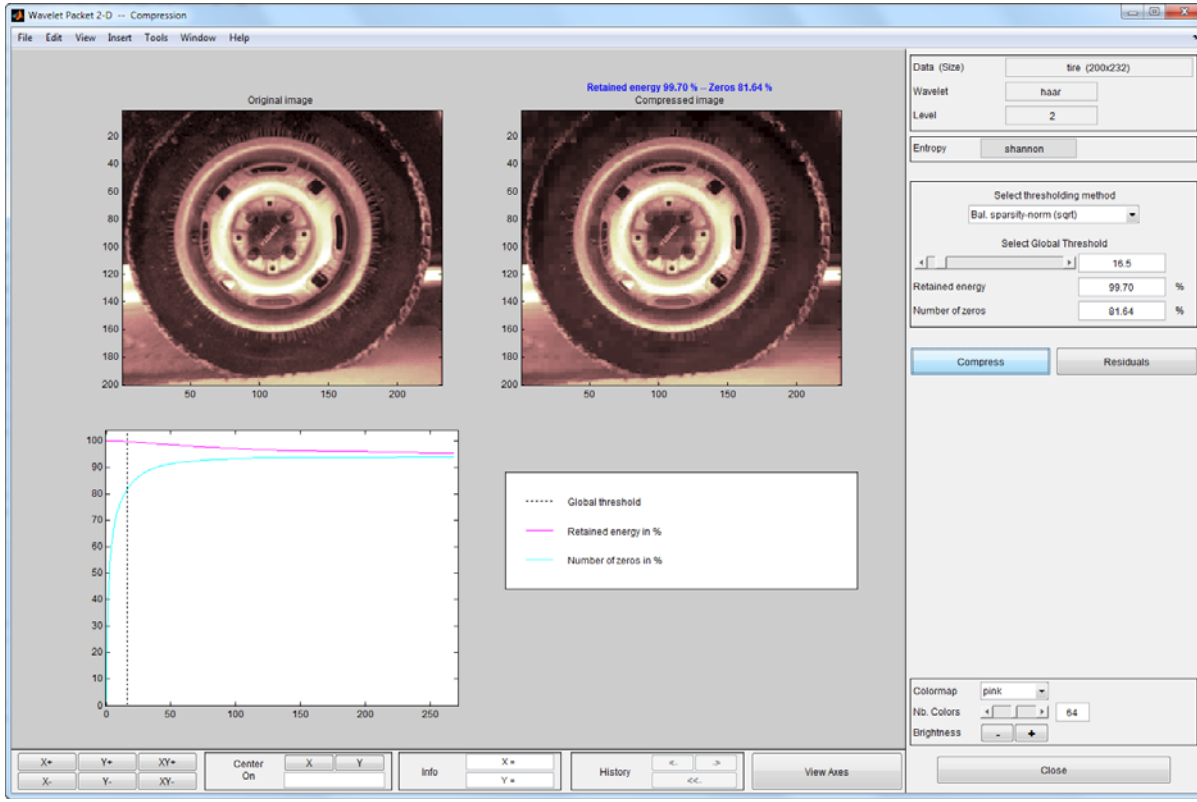
4 Using the default parameter settings, click **Best Tree**.



5 Click **Compress**.

6 Set `Select thresholding method` to `Bal.sparsity-norm (sqrt)`.

7 Click **Compress**.



8 **File > Generate Code (Compression Process)** generates the following code.

```
function [XCMP,wptCMP] = func_compress_wp2d(X)
% FUNC_COMPRESS_WP2D Saved Compression Process.
%   X: matrix of data
%   -----------------
%   XCMP: matrix of compressed data
%   wptCMP: wavelet packet decomposition (wptree object)

% Analysis parameters.
%---------------------
Wav_Nam = 'haar';
Lev_Anal = 2;
```

```
Ent_Nam = 'shannon';
Ent_Par = 0;

% Compression parameters.
%------------------------
% meth = 'sqrtbal_sn';
sorh = 'h';    % Specified soft or hard thresholding
thrSettings = {sorh,'nobest',16.499999999999886,1};
roundFLAG = true;

% Decompose using WPDEC2.
%------------------------
wpt = wpdec2(X,Lev_Anal,Wav_Nam,Ent_Nam,Ent_Par);

% Nodes to merge.
%-----------------
n2m = [2  3];
for j = 1:length(n2m)
    wpt = wpjoin(wpt,n2m(j));
end

% Compression using WPDENCMP.
%----------------------------
[XCMP,wptCMP] = wpdencmp(wpt,thrSettings{:});
if roundFLAG , XCMP = round(XCMP); end
if isequal(class(X),'uint8') , XCMP = uint8(XCMP); end
```

9 Save the generated MATLAB code as `func_compress_wp2d.m` in a folder on the MATLAB search path, and execute the following code.

```
load tire;
[XCMP,wptCMP] = func_compress_wp2d(X);
```

10 Save the compressed image from the **Wavelet 2-D -- Compression** tool as `compressed_tire.mat` in a folder on the MATLAB search path. Use **File > Save > Compressed Image** to save the compressed image.

11 Execute the following code to compare the command line and GUI result.

```
load compressed_tire.mat;
norm(XCMP-X,2)
```

6

Advanced Concepts

This chapter presents a more advanced treatment of wavelet methods, and focuses on real wavelets, except in the two sections dedicated to wavelet families.

Mathematical Conventions

This chapter and the reference pages use certain mathematical conventions.

General Notation	Interpretation
$a = 2^j, j \in Z$	Dyadic scale. j is the level, 1/α 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.
$\hat{f}$	Fourier transform of the function f or the sequence f
Continuous Time	
$L^2(R)$	Set of signals of finite energy
$\int_R s^2(x)dx$	Energy of the signal s
$\langle s, s' \rangle = \int_R s(x)s'(x)dx$	Scalar product of signals s and s'
$L^2(R^2)$	Set of images of finite energy

General Notation	Interpretation
$\int_R \int_R s^2(x,y)dxdy$	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'
Discrete Time	
$l^2(Z)$	Set of signals of finite energy
$\sum_{n \in Z} s^2(n)$	Energy of the signal s
$\langle s, s' \rangle = \sum_{n \in Z} s(n)s'(n)$	Scalar product of signals s and s'
$l^2(Z^2)$	Set of images of finite energy
$\sum_{n \in Z} \sum_{m \in Z} s^2(n,m)$	Energy of the image s
$\langle s, s' \rangle = \sum_{n \in Z} \sum_{m \in Z} s(n,m)s'(n,m)$	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

Wavelet Notation	Interpretation
φ	Scaling function
Ψ	Wavelet
$\frac{1}{\sqrt{a}}\psi\left(\frac{x-b}{a}\right)$	Family associated with the one-dimensional wavelet, indexed by $a > 0$ 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, indexed by $a_1 > 0$, $a_2 > 0$, $b_1 \epsilon R$, $b_2 \epsilon R$
$\varphi_{j,k}(x) = 2^{-j/2}\varphi(2^{-j}x-k),\ j\epsilon Z, k\epsilon Z$	Family associated with the one-dimensional scaling function for dyadic scales $a = 2^j, b = ka$; it should be noted that $\varphi = \varphi_{0,0}$.
$\Psi_{j,k}(x) = 2^{-j/2}\Psi(2^{-j}x-k),\ j\epsilon Z, k\epsilon Z$	Family associated with the one-dimensional Ψ for dyadic scales $a = 2^j, b = ka$; it should be noted that $\Psi = \Psi_{0,0}$.
$(h_k),\ k\epsilon Z$	Scaling filter associated with a wavelet
$(g_k),\ k\epsilon Z$	Wavelet filter associated with a wavelet

General Concepts

This section presents a brief overview of wavelet concepts, focusing mainly on the orthogonal wavelet case. It includes the following sections:

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 usually 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. We can consider the original signal 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. Among the figures presented in "Reconstructing Approximations and Details" in the *Wavelet Toolbox Getting Started Guide*, one of them graphically represents this hierarchical decomposition.

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.

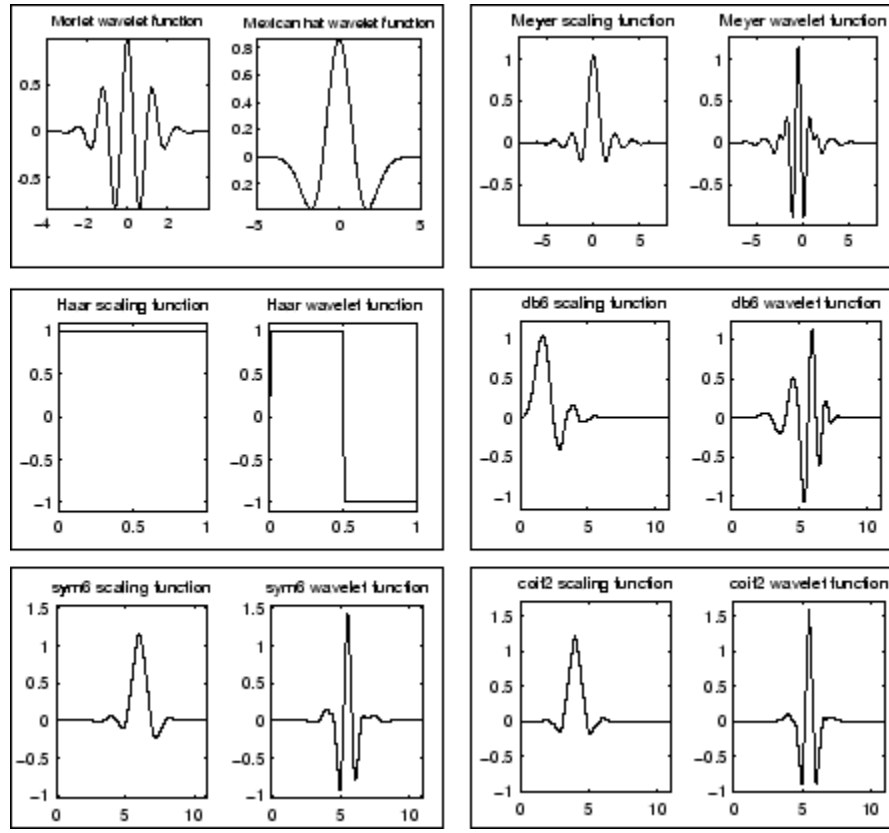
j	10	9	...	2	1	0	–1	–2
Scale	1024	512	...	4	2	1	1/2	1/4
Resolution	$1/2^{10}$	$1/2^9$	...	1/4	1/2	1	2	4

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.

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 $\varphi(x_1, x_2)$ and three wavelets.

The following figure 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.



Various One-Dimensional Wavelets

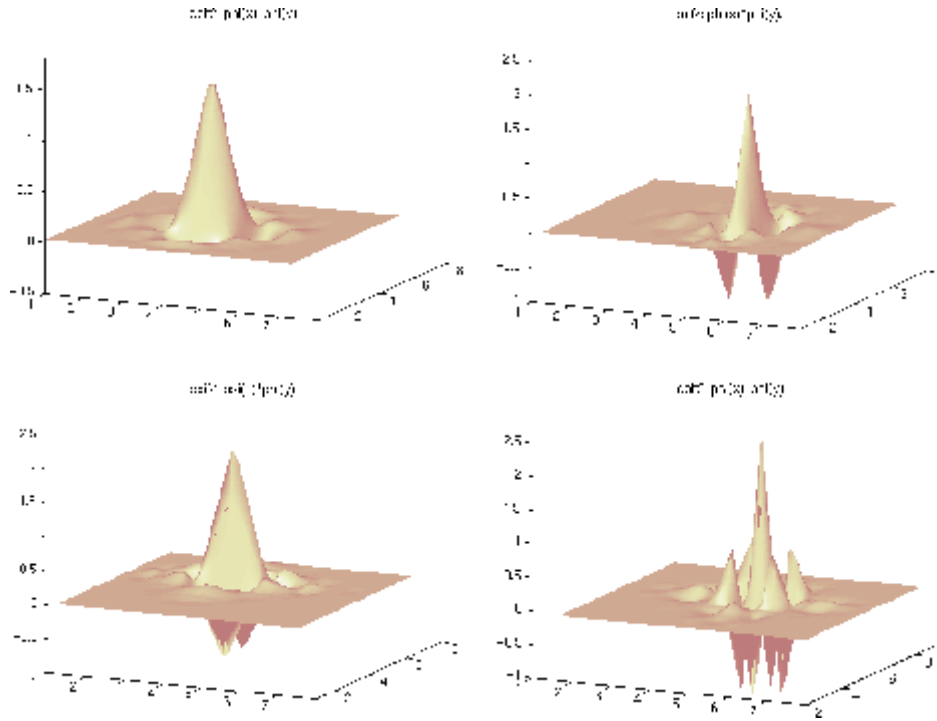
Wavelets and Associated Families

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

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

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

The following figure shows the four functions associated with the 2-D `coif2` wavelet.



Two-Dimensional coif2 Wavelet

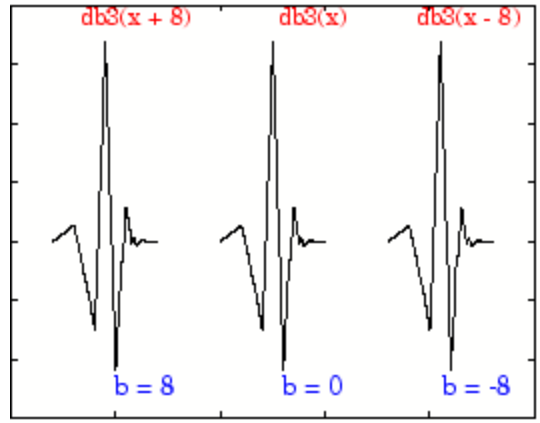
To each of these functions, we associate its doubly indexed family, which is used to:

- Move the basic shape from one side to the other, translating it to position b (see the following figure).
- Keep the shape while changing the one-dimensional time scale α (α > 0) (see Time Scaled One-Dimensional Wavelet on page 6-10).

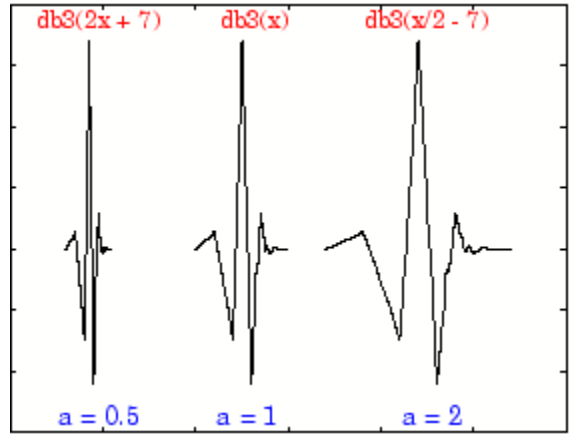
So a wavelet family member has to be thought of as a function located at a position b, and having a scale a.

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

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



Translated Wavelets



Time Scaled One-Dimensional Wavelet

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:

$$\frac{1}{\sqrt{a_1 a_2}}\psi\left(\frac{x_1 - b_1}{a_1},\frac{x_2 - b_2}{a_2}\right) \text{ where } (x = x_1, x_2) \in R^2$$

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):

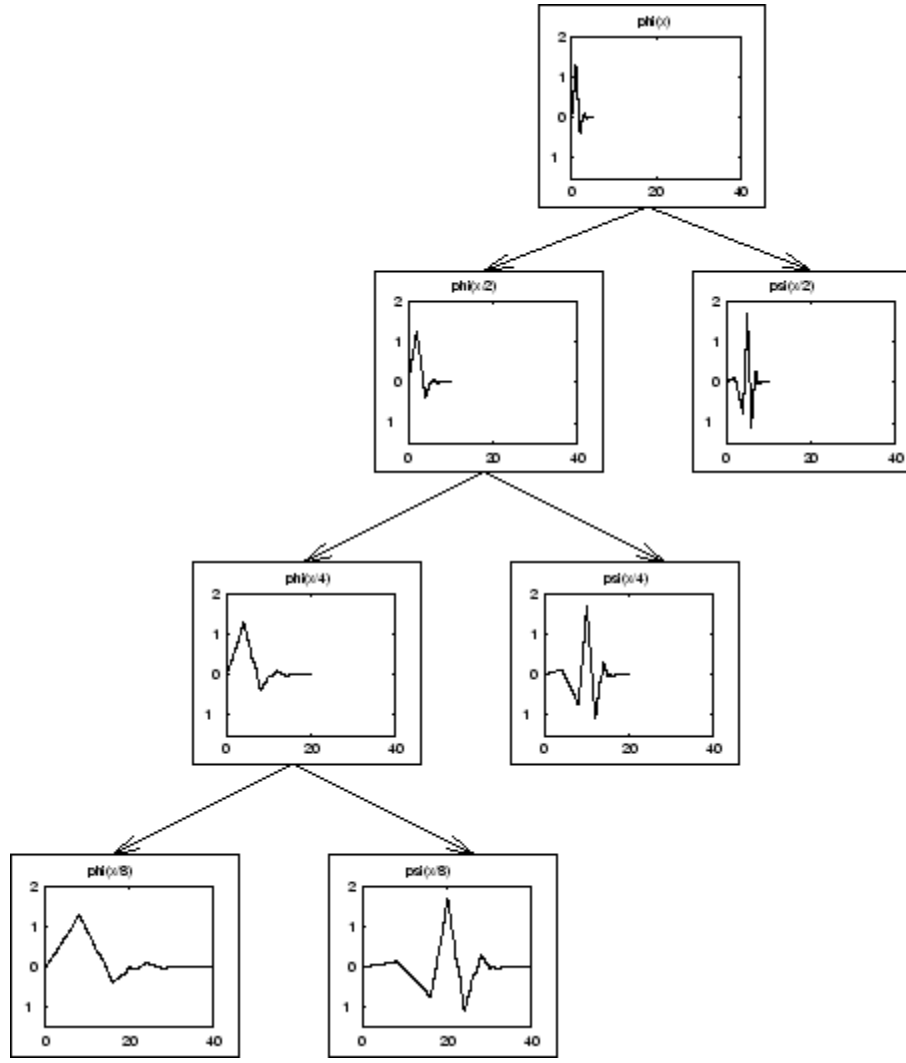
$$(j,k) \in Z^2 : a = 2^j, \quad b = k2^j = ka$$

Let us define:

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

We now have a hierarchical organization similar to the organization of a decomposition; this is represented in the example of Wavelets Organization on page 6-11. Let $k = 0$ and leave the translations aside for the moment. The functions associated with $j = 0, 1, 2, 3$ for φ (expressed as $\varphi_{j,0}$) and with $j =$

1, 2, 3 for ψ (expressed as $\psi_{j,0}$) are displayed in the following figure for the `db3` wavelet.



Wavelets Organization

In the preceding figure, the four-level decomposition is shown, progressing from the top to the bottom. We find $\varphi_{0,0}$; then $2^{1/2}\varphi_{1,0}$, $2^{1/2}\Psi_{1,0}$; then $2\varphi_{2,0}$, $2\Psi_{2,0}$; then $2^{3/2}\varphi_{3,0}$, $2^{3/2}\Psi_{3,0}$. The wavelet is `db3`.

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 two transforms:

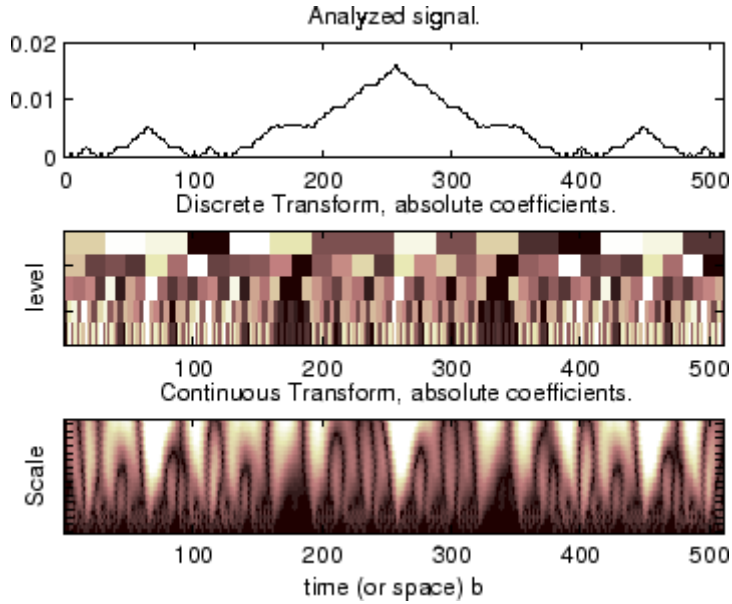
- Continuous transform
- Discrete transform

From an intuitive point of view, the wavelet decomposition consists of calculating a "resemblance index" between the signal and the wavelet located at position b and of scale a. 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 table. We have two types of analysis at our disposal.

Continuous Time Signal Continuous Analysis	Continuous Time Signal Discrete Analysis
$C(a,b) = \int_R s(t)\frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)dt$	$C(a,b) = \int_R s(t)\frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)dt$
$a \in R^+ - \{0\}, b \in R$	$a = 2^j, b = k2^j, (j,k) \in Z^2$

Next we will illustrate the differences between the two transforms, for the analysis of a fractal signal (see the figure Continuous Versus Discrete Transform on page 6-13).



Continuous Versus Discrete Transform

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

- **Graphic representation of discrete analysis**: (in the middle of the figure Continuous Versus Discrete Transform on page 6-13) 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 (from the bottom to the top), levels are 1, 2, 3, 4, and 5. Each coefficient of level k is repeated 2^k times.
- **Graphic representation of continuous analysis**: (at the bottom of the figure Continuous Versus Discrete Transform on page 6-13) time is on the abscissa and on the ordinate the scale varies almost continuously between 2^1 and 2^5 by step 1 (from the bottom to the top). Keep in mind that when a scale is small, only small details are analyzed, as in a geographical map.

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 ψ is localized. The signal *s* and the function ψ are then compared in the disk, taking into account only the *t* values in the disk. The signal values, which are located outside of the domain U, do not influence the value of the coefficient

$$\int_R s(t)\psi(t)dt \text{ and we get } \int_R s(t)\psi(t)dt = \int_U s(t)\psi(t)dt$$

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). The good spectral domain localization is a second type of a useful property. A result (linked to the Heisenberg uncertainty principle) links the dispersion of the signal *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 time localized function. The support is R. Its Fourier transform $\hat{f}$ is a generalized function 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:

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

Of course, the previous formulas need some hypotheses on the ψ function. More precisely, see "What Functions Are Candidates to Be a Wavelet?" on page 6-65 for the continuous synthesis formula and "Why Does Such an Algorithm Exist?" on page 6-28 for the discrete one.

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 $\alpha = 2j \le 2^J$ 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

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

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 $j \neq k$
- A_J is an approximation of s. The quality (in energy) of the approximation of s by A_J is

$$qual_J = \frac{\|A_J\|^2}{\|s\|^2}$$

-

$$qual_{J-1} = qual_J + \frac{\|D_J\|^2}{\|s\|^2} \geq qual_J$$

The following table contains definitions of details and approximations.

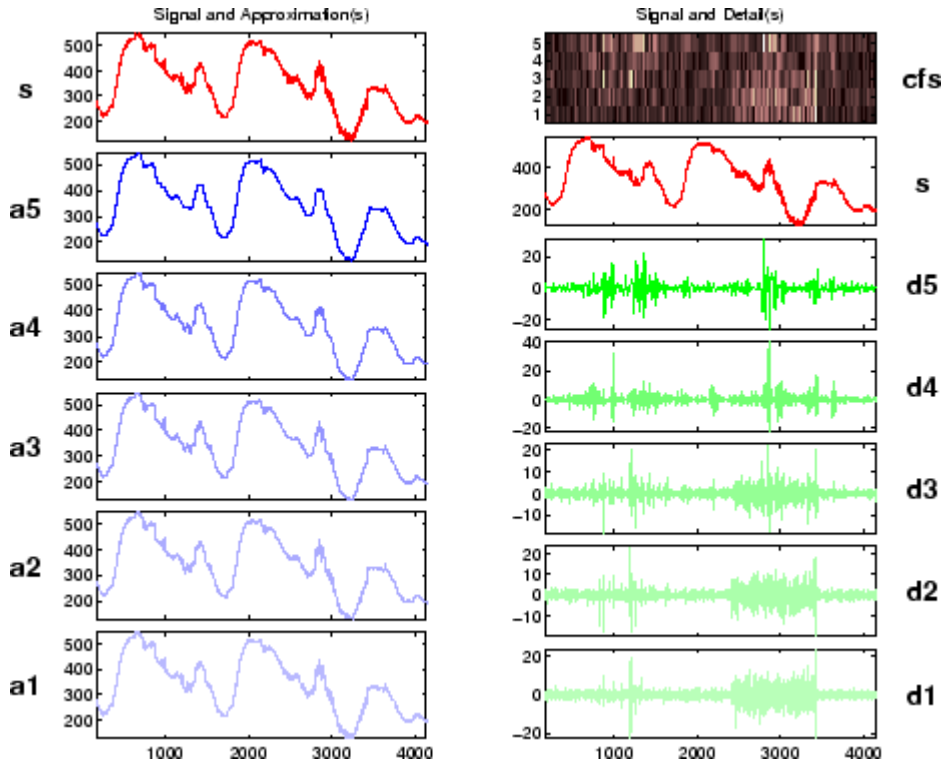
Definition of the detail at level j	$D_j(t) = \sum_{k \in Z} C(j,k)\psi_{j,k}(t)$
The signal is the sum of its details	$s = \sum_{j \in Z} D_J$
The approximation at level J	$s = A_J + \sum_{j>J} D_j$
Link between A_{J-1} and A_J	$A_{J-1} = A_J + D_J$
Several decompositions	$s = A_J + \sum_{j>J} D_j$

From a graphical point of view, when analyzing a signal, it is always valuable to represent the different signals (s, A_j, D_j) and coefficients ($C(j,k)$).

Let us consider Approximations, Details, and Coefficients on page 6-18. On the left side, **s** is the signal; **a5**, **a4**, **a3**, **a2**, and **a1** are the approximations at levels 5, 4, 3, 2, and 1. The best approximation is **a1**; the next one is **a2**, and so on. Noise oscillations are exhibited in **a1**, whereas **a5** is smoother.

On the right side, **cfs** represents the coefficients (for more information, see "Wavelet Transforms: Continuous and Discrete" on page 6-12), **s** is the signal, and **d5**, **d4**, **d3**, **d2**, and **d1** are the details at levels 5, 4, 3, 2, and 1.

The different signals that are presented exist in the same time grid. We can consider that the t index of detail $D_4(t)$ identifies the same temporal instant as that of the approximation $A_5(t)$ and that of the signal $s(t)$. This identity is of considerable practical interest in understanding the composition of the signal, even if the wavelet sometimes introduces dephasing.



Approximations, Details, and Coefficients

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 (QMFs).

- The decomposition algorithm starts with signal s, next calculates the coordinates of A_1 and D_1, and 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, next calculates the coordinates of A_{J-1}, and then using the coordinates of A_{J-1} and D_{J-1} calculates those of A_{J-2}, and so on.

This section addresses the following topics:

Filters Used to Calculate the DWT and IDWT

For an orthogonal wavelet, in the multiresolution framework (see [Dau92] in Chapter 3, "Using Wavelet Packets"), 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 (nonnormalized), is

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

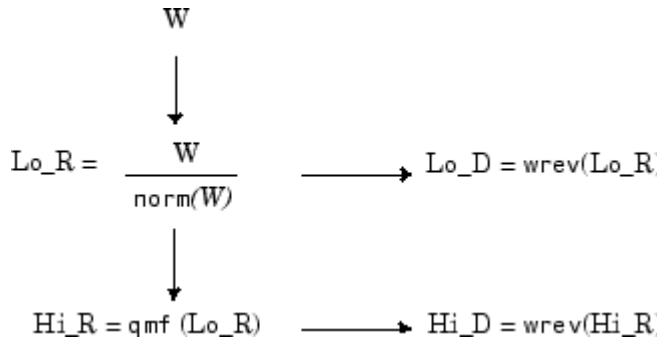
For example, for the `db3` scaling filter,

```
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 $2N$ and of 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.

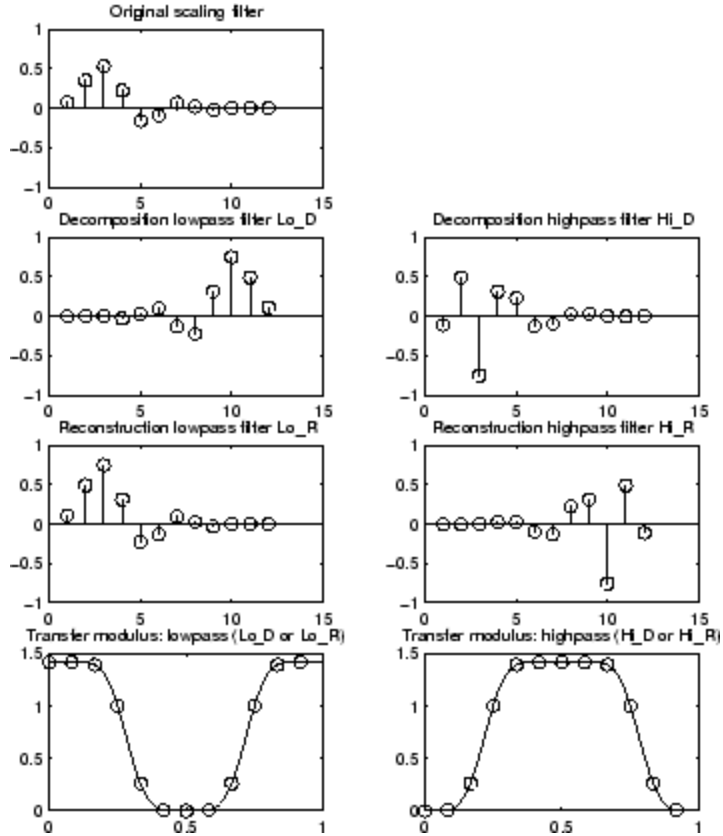


where `qmf` is such that Hi_R and Lo_R are quadrature mirror filters (i.e., $\mathrm{Hi\_R}(k) = (-1)^k \mathrm{Lo\_R}(2N + 1 - k)$) for $k = 1, 2, ..., 2N$.

Note that `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`. Next, we illustrate these properties with the `db6` wavelet. The plots associated with the following commands are shown in Four Wavelet Filters for db6 on page 6-22.

```
% Load scaling filter.
load db6; w = db6;
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 Lo{\_}D');
subplot(424); stem(Hi_D);
title('Decomposition high-pass filter Hi{\_}D');
subplot(425); stem(Lo_R);
title('Reconstruction low-pass filter Lo{\_}R');
subplot(426); stem(Hi_R);
title('Reconstruction high-pass filter Hi{\_}R');

% High and low frequency illustration.
n = length(Hi_D);
freqfft = (0:n-1)/n;
nn = 1:n;
N = 10*n;
```

```
for k=1:N
    lambda(k) = (k-1)/N;
    XLo_D(k) = exp(-2*pi*j*lambda(k)*(nn-1))*Lo_D';
    XHi_D(k) = exp(-2*pi*j*lambda(k)*(nn-1))*Hi_D';
end
fftld = fft(Lo_D);
ffthd = fft(Hi_D);
subplot(427); plot(lambda,abs(XLo_D),freqfft,abs(fftld),'o');
title('Transfer modulus: lowpass (Lo{\_}D or Lo{\_}R')
subplot(428); plot(lambda,abs(XHi_D),freqfft,abs(ffthd),'o');
title('Transfer modulus: highpass (Hi{\_}D or Hi{\_}R')
```

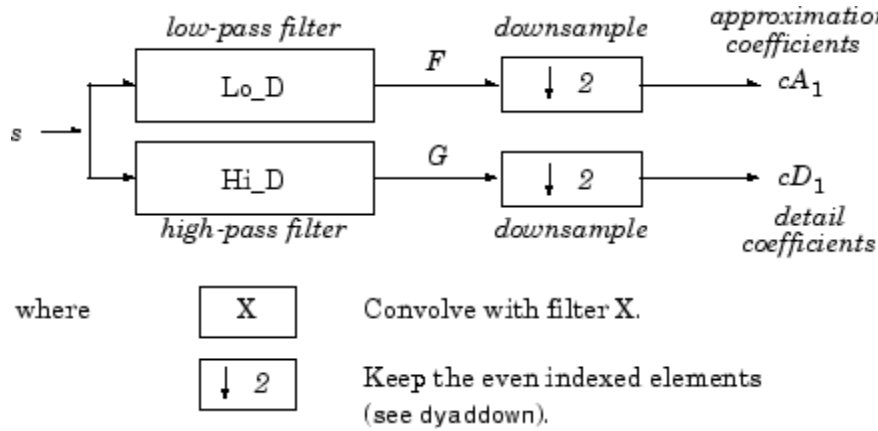


Four Wavelet Filters for db6

Algorithms

Given a signal s of length N, the DWT consists of $\log_2 N$ stages at most. Starting from s, the first step produces 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.

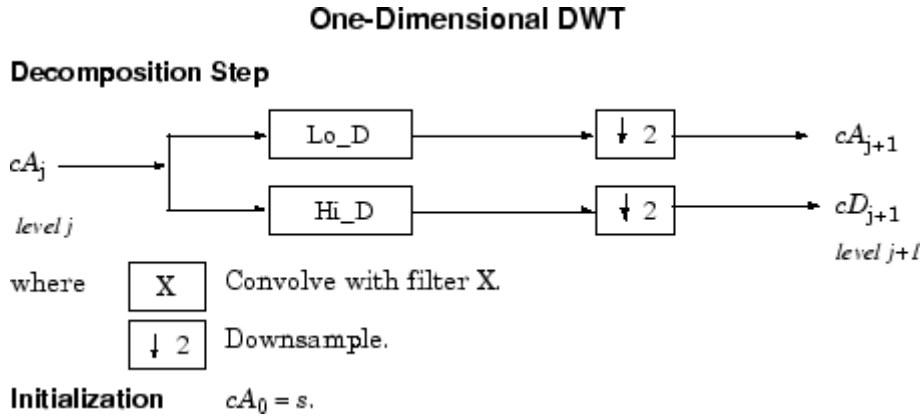
More precisely, the first step is



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

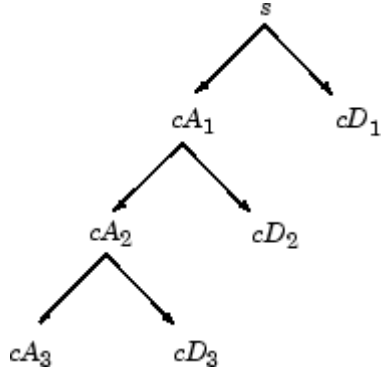
$$\text{floor}\left(\frac{(N-1)}{2} + n\right)$$

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.

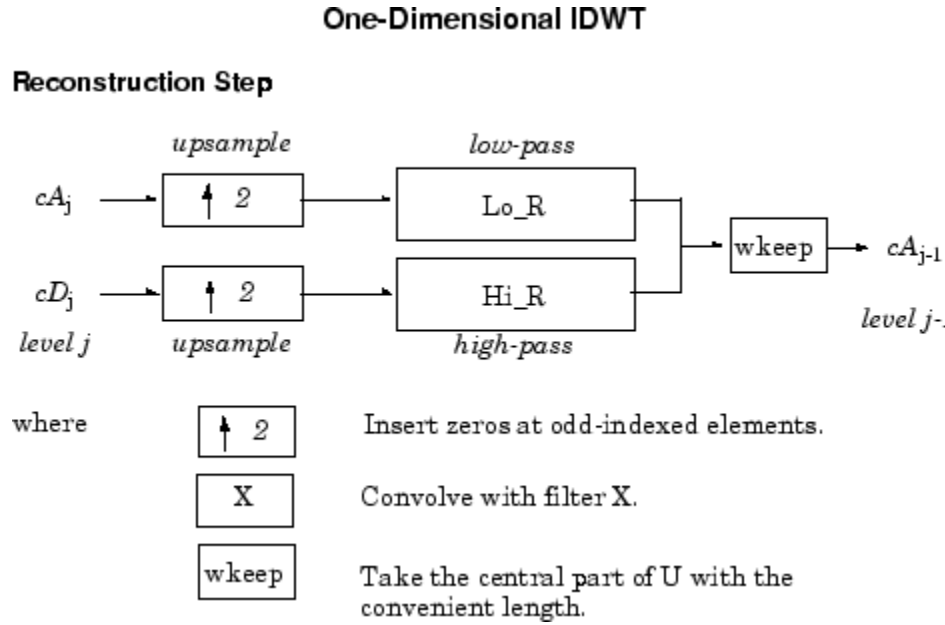


So the wavelet decomposition of the signal s analyzed at level j has the following structure: $[cA_j, cD_j, ..., cD_1]$.

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



- 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.



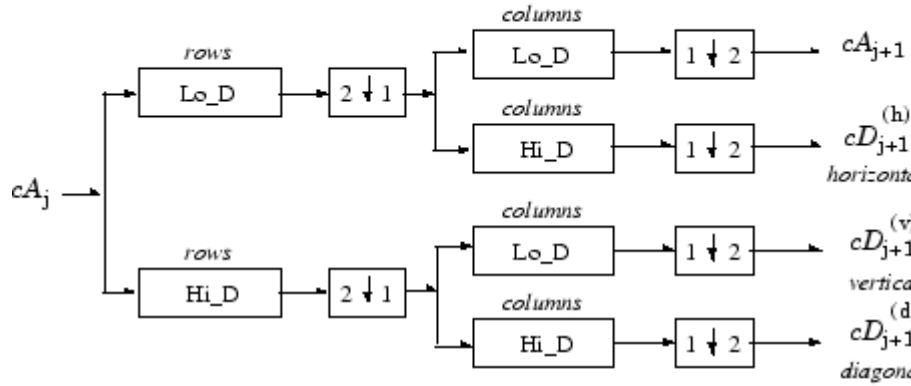
- For images, a similar algorithm is possible for two-dimensional wavelets and scaling functions obtained from one-dimensional wavelets by tensorial product.

  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

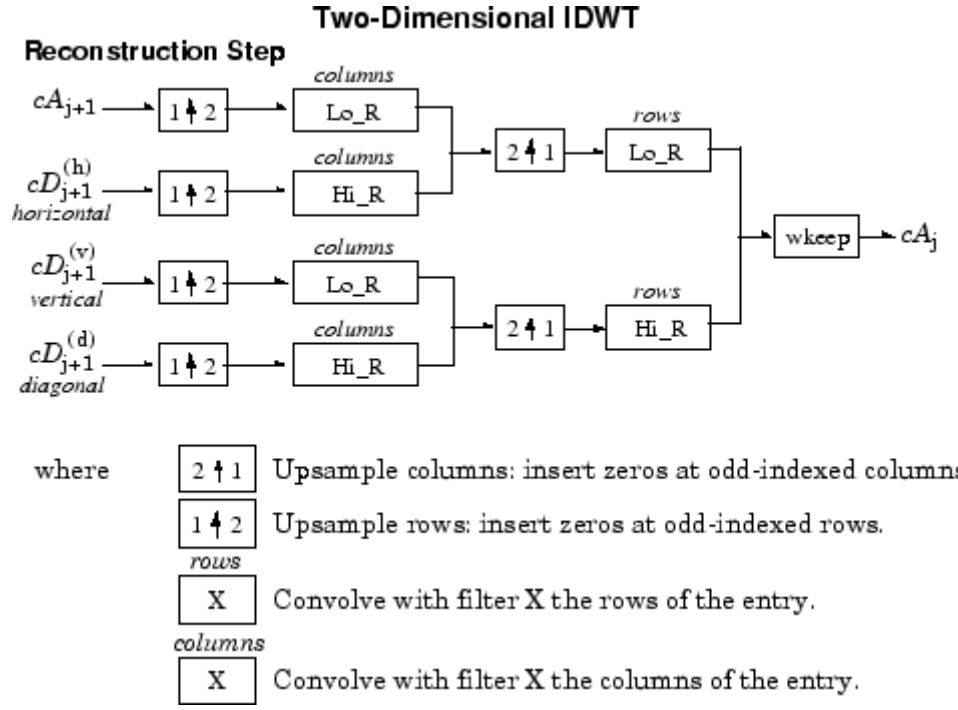
Decomposition Step



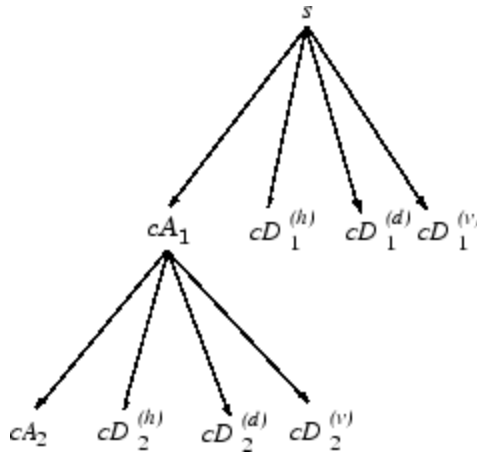
where

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.



So, for $J = 2$, the two-dimensional wavelet tree has the following 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?

The previous paragraph describes algorithms designed for finite-length signals or images. To understand the rationale, we must consider infinite-length signals. The methods for the extension of a given finite-length signal are described in "Dealing with Border Distortion" on page 6-35.

Let us denote h = Lo_R and g = Hi_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 $(A_k^{(j)})k \epsilon Z$ 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

$$\tilde{h}(k) = h(-k), \text{ and } F_k^{(j+1)} = \sum_n \tilde{h}k - n\, An^{(j)}$$

The sequence $F^{(j+1)}$ is the filtered output of the sequence $A^{(j)}$ by the filter $\tilde{h}$.

We obtain

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

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

The sequence $A^{(j+1)}$ is the downsampled version of the sequence $F^{(j+1)}$.

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 $\varphi_{j,k}$ 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 ($h_n = \sqrt{2}\omega_n$). For more information, see "Filters Used to Calculate the DWT and IDWT" on page 6-19.

5 We check that:

a The coordinate of $\varphi_{j+1,0}$ on $\varphi_{j,k}$ is h_k and does not depend on j.

b The coordinate of $\varphi_{j+1,n}$ on $\varphi_{j,k}$ 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 After the decomposition step, we justify now the reconstruction algorithm by building it. Let us simplify the notation. Starting from A_1 and D_1, let us study $A_0 = A_1 + D_{j1}$. The procedure is the same to calculate $A = A_{j+1} + D_{j+1}$.

Let us define α_n, δ_n, α_k^0 by

$$A_1 = \sum_n a_n \phi_{1,n} \qquad D_1 = \sum_n \delta_n \psi_{1,n} \qquad A_0 = \sum_k a_k^0 \phi_{0,k}$$

Let us assess the α_k^0 coordinates as

$$\begin{aligned} a_k^0 &= \langle A_0, \phi_{0,k} \rangle = \langle A_1 + D_1, \phi_{0,k} \rangle = \langle A_1, \phi_{0,k} \rangle + \langle D_1, \phi_{0,k} \rangle \\ &= \sum_n a_n \langle \phi_{1,n}, \phi_{0,k} \rangle + \sum_n \delta_n \langle \psi_{1,n}, \phi_{0,k} \rangle \\ &= \sum_n a_n h_{k-2n} + \sum_n \delta_n g_{k-2n} \end{aligned}$$

We will focus our study on the first sum $\sum_n a_n h_{k-2n}$; 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)

$$\begin{aligned} &\sum_n a_n h_{-2n} = \ldots + a_{-1} h_2 + a_0 h_0 + a_1 h_{-2} + a_2 h_{-4} + \ldots \\ &= \ldots + a_{-1} h_2 + 0 h_1 + a_0 h_0 + 0 h_{-1} + a_1 h_{-2} + 0 h_{-3} + a_2 h_{-4} + \ldots \end{aligned}$$

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, ... that is precisely

$$\tilde{a}_{2n} = a_n, \tilde{a}_{2n+1} = 0$$

Then

$$\sum_n a_n h_{-2n} = \sum_n \tilde{a}_n h_{-n}$$

and by extension

$$\sum_n a_n h_{k-2n} = \sum_n \tilde{a}_n h_{k-n}$$

Since

$$a_k^0 = \sum_n \tilde{a}_n h_{k-n} + \sum_n \tilde{\delta}_n g_{k-n}$$

the reconstruction steps are:

1. Replace the α and δ sequences by upsampled versions α˜ and $\tilde{\delta}$ inserting zeros.
2. Filter by h and g respectively.
3. Sum the obtained sequences.

One-Dimensional Wavelet Capabilities

Basic One-Dimensional Objects.

	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, $j \ge 1$

Analysis-Decomposition Capabilities.

Purpose	Input	Output	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`

Synthesis-Reconstruction Capabilities.

Purpose	Input	Output	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`

Decomposition Structure Utilities.

Purpose	Input	Output	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`

To illustrate command-line mode for one-dimensional capabilities, see "One-Dimensional Analysis Using the Command Line" in the *Wavelet Toolbox Getting Started Guide*.

Two-Dimensional Wavelet Capabilities

Basic Two-Dimensional Objects.

	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 $\left[D_k^{(h)}, D_k^{(v)}, D_k^{(d)}\right]$, the horizontal, vertical, and diagonal details at level k.

The same holds for cD_k, which stands for $\left[cD_k^{(h)}, cD_k^{(v)}, cD_k^{(d)}\right]$.

The two-dimensional files are the same as those for the one-dimensional case, but with a `2` appended on the end of the command.

For example, `idwt` becomes `idwt2`. For more information, see "One-Dimensional Wavelet Capabilities" on page 6-32.

To illustrate command-line mode for two-dimensional capabilities, see "Two-Dimensional Analysis Using the Command Line" in the *Wavelet Toolbox Getting Started Guide*.

Dealing with Border Distortion

Classically, the DWT is defined for sequences with length of some power of 2, and different ways of extending samples of other sizes are needed. Methods for extending the signal include zero-padding, smooth 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

To deal with border distortions, the border should be treated differently from the other parts of the signal.

Various methods are available to deal with this problem, referred to as "wavelets on the interval" (see [CohDJV93] in "References" on page 6-168). 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 to get a perfect reconstruction. It should be noted that extension is needed at each stage of the decomposition process.

Details on the rationale of these schemes are in Chapter 8 of the book *Wavelets and Filter Banks*, by Strang and Nguyen (see [StrN96] in "References" on page 6-168).

The available signal extension modes are as follows (see `dwtmode`):

- **Zero-padding** (`'zpd'`): 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.

  The disadvantage of zero-padding is that discontinuities are artificially created at the border.

- **Symmetrization** (`'sym'`): This method assumes that signals or images can be recovered outside their original support by symmetric boundary value replication.

  It is the default mode of the wavelet transform in the toolbox.

  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 of order 1** (`'spd'` or `'sp1'`): This method assumes that signals or images can be recovered outside their original support by a simple first-order derivative extrapolation: padding using a linear extension fit to the first two and last two values.

  Smooth padding works well in general for smooth signals.

- **Smooth padding of order 0** (`'sp0'`): This method assumes that signals or images can be recovered outside their original support by a simple constant extrapolation. For a signal extension this is the repetition of the first value on the left and last value on the right.

- **Periodic-padding (1)** (`'ppd'`): This method assumes that signals or images can be recovered outside their original support by periodic extension.

  The disadvantage of periodic padding is that discontinuities are artificially created at the border.

The DWT associated with these five modes is slightly redundant. But IDWT ensures a perfect reconstruction for any of the five previous modes whatever the extension mode used for DWT.

- **Periodic-padding (2)** (`'per'`): If the signal length is odd, the signal is first extended by adding an extra-sample equal to the last value on the right. Then a minimal periodic extension is performed on each side. The same kind of rule exists for images. This extension mode is used for SWT (1-D & 2-D).

This last mode produces the smallest length wavelet decomposition. But the extension mode used for IDWT must be the same to ensure a perfect reconstruction.

Before looking at an illustrative example, let us compare some properties of the theoretical Discrete Wavelet Transform versus the actual DWT.

The theoretical DWT is applied to signals that are defined on an infinite length time interval (Z). For an orthogonal wavelet, this transform has the following desirable properties:

1 Norm preservation

Let *cA* and *cD* be the approximation and detail of the DWT coefficients of an infinite length signal *X*. Then the l^2–*norm* is preserved:

$$||X||^2 = ||cA||^2 + ||cD||^2$$

2 Orthogonality

Let *A* and *D* be the reconstructed approximation and detail. Then, *A* and *D* are orthogonal and

$$||X||^2 = ||A||^2 + ||D||^2$$

3 Perfect reconstruction

$$X = A + D$$

Since the DWT is applied to signals that are defined on a finite-length time interval, extension is needed for the decomposition, and truncation is necessary for reconstruction.

To ensure the crucial property **3** (perfect reconstruction) for arbitrary choices of

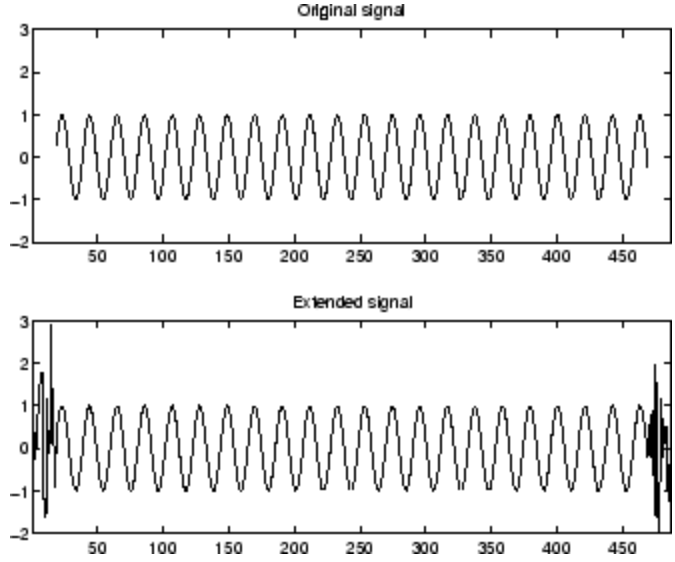
- The signal length
- The wavelet
- The extension mode

the properties **1** and **2** can be lost. These properties hold true for an extended signal of length usually larger than the length of the original signal. So only the perfect reconstruction property is always preserved. Nevertheless if the

DWT is performed using the periodic extension mode ('per') and if the length of the signal is divisible by 2^J, where J is the maximum level decomposition, the properties **1**, **2**, and **3** remain true.

It is interesting to notice that if arbitrary extension is used, and decomposition performed using the convolution-downsampling scheme, perfect reconstruction is recovered using `idwt` or `idwt2`. This point is illustrated below.

```
% Set initial signal and get filters.
x = sin(0.3*[1:451]); w = 'db9';
[Lo_D,Hi_D,Lo_R,Hi_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(Lo_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,Lo_D)),la);
dr = wkeep(dyaddown(conv(ex,Hi_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 first 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`.

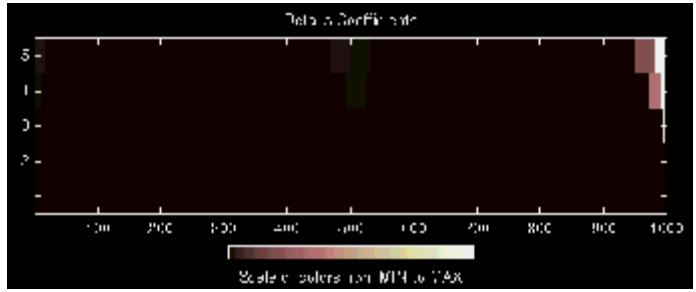
   The **Wavelet Toolbox Main Menu** appears.

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 **Example Analysis** option and select **Basic Signals > 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')
```

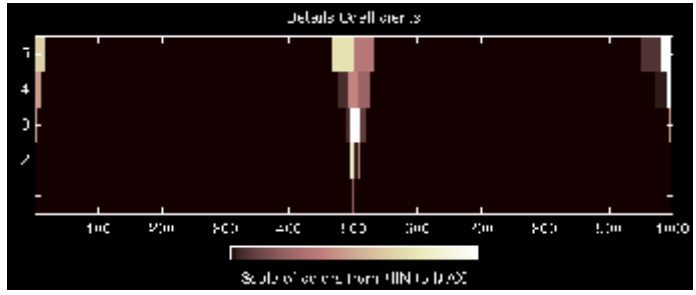
7 Click the **Wavelet 1-D** menu item.

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

8 From the **File** menu, choose the **Example Analysis** option and select **Basic Signals > with db2 at level 5 > Two nearby discontinuities**.

9 Select **Display Mode: Show and Scroll**.

The detail coefficients show the signal end effects are present, but the discontinuities are well detected.



Smooth Padding

10 From the MATLAB prompt, type

```
dwtmode('spd')
```

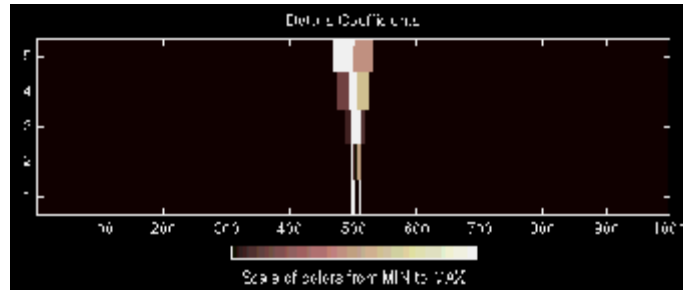
11 Click the **Wavelet 1-D** menu item.

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

12 From the **File** menu, choose the **Example Analysis** option and select **Basic Signals > with db2 at level 5 > Two nearby discontinuities**.

13 Select **Display Mode: Show and Scroll**.

The detail coefficients show the signal end effects are not present, and the discontinuities are well detected.

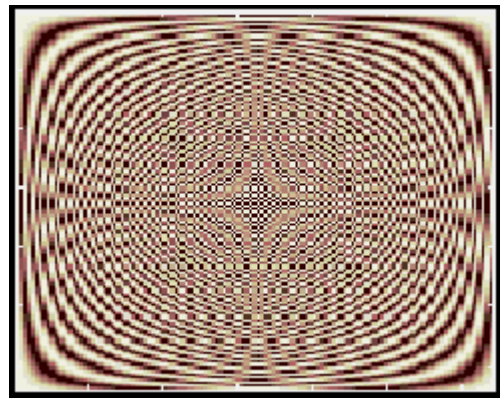


Let us now consider an image example.

Original Image

1 From the MATLAB prompt, type

```
load geometry;
% X contains the loaded image and
% map contains the loaded colormap.
nbcol = size(map,1);
colormap(pink(nbcol));
image(wcodemat(X,nbcol));
```

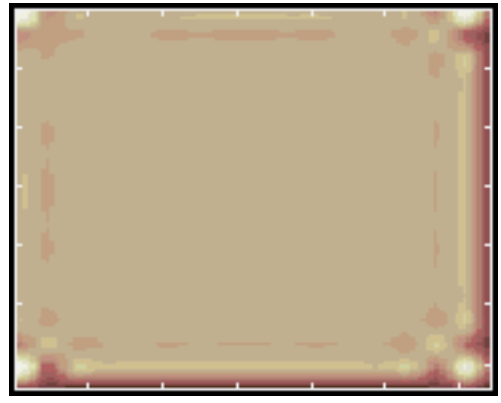


Zero-Padding

Now we set the extension mode to zero-padding and perform a decomposition of the image to level 3 using the `sym4` wavelet. Then we reconstruct the approximation of level 3.

2 From the MATLAB prompt, type

```
lev = 3; wname = 'sym4';
dwtmode('zpd')
[c,s] = wavedec2(X,lev,wname);
a = wrcoef2('a',c,s,wname,lev);
image(wcodemat(a,nbcol));
```

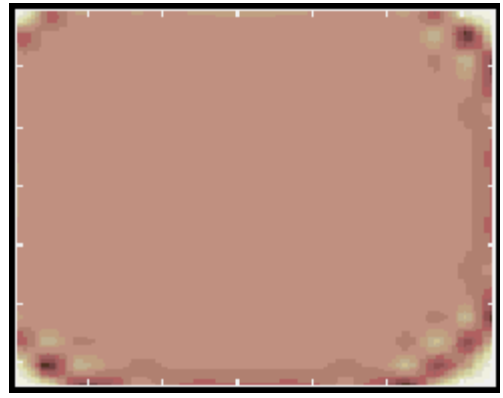


Symmetric Extension

Now we set the extension mode to symmetric extension and perform a decomposition of the image again to level 3 using the `sym4` wavelet. Then we reconstruct the approximation of level 3.

3 From the MATLAB prompt, type

```
dwtmode('sym')
[c,s] = wavedec2(X,lev,wname);
a = wrcoef2('a',c,s,wname,lev);
image(wcodemat(a,nbcol));
```

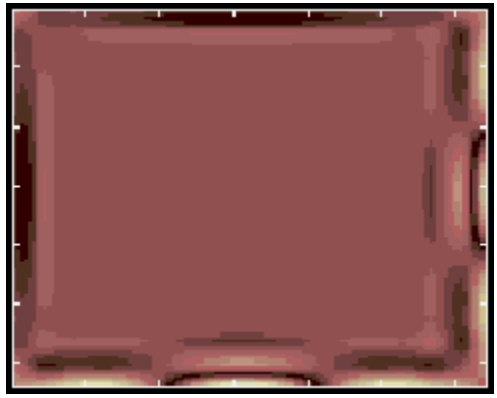


Smooth Padding

Now set the extension mode to smooth padding and perform a decomposition of the image again to level 3 using the `sym4` wavelet. Then reconstruct the approximation of level 3.

4 From the MATLAB prompt, type

```
dwtmode('spd')
[c,s] = wavedec2(X,lev,wname);
a = wrcoef2('a',c,s,wname,lev);
image(wcodemat(a,nbcol));
```



Discrete Stationary Wavelet Transform (SWT)

We know that the classical DWT suffers a drawback: the DWT is not a time-invariant transform. This means that, even with periodic signal extension, the DWT of a translated version of a signal X is not, in general, the translated version of the DWT of X.

How to restore the translation invariance, which is a desirable property lost by the classical DWT? The idea is to average some slightly different DWT, called ε-decimated DWT, to define the stationary wavelet transform (SWT). This property is useful for several applications such as breakdown points detection.

The main application of the SWT is de-noising. For more information on the rationale, see [CoiD95] in "References" on page 6-168. For examples, see "One-Dimensional Discrete Stationary Wavelet Analysis" and "Two-Dimensional Discrete Stationary Wavelet Analysis" in the *Wavelet Toolbox Getting Started Guide*.

The principle is to average several de-noised signals. Each of them is obtained using the usual de-noising scheme (see "De-Noising" on page 6-101), but applied to the coefficients of an ε-decimated DWT.

Note We define the SWT only for signals of length divisible by 2^J, where J is the maximum decomposition level, and we use the DWT with periodic (`per`) extension.

ε-Decimated DWT

What is an ε-decimated DWT?

There exist a lot of slightly different ways to handle the discrete wavelet transform. Let us recall that the DWT basic computational step is a convolution followed by a decimation. The decimation retains even indexed elements.

But the decimation could be carried out by choosing odd indexed elements instead of even indexed elements. This choice concerns every step of the decomposition process, so at every level we chose odd or even.

If we perform all the different possible decompositions of the original signal, we have 2^J different decompositions, for a given maximum level J.

Let us denote by $\varepsilon_j = 1$ or 0 the choice of odd or even indexed elements at step j. Every decomposition is labeled by a sequence of 0s and 1s: $\varepsilon = \varepsilon_1 ..., \varepsilon_J$. This transform is called the ε-decimated DWT.

You can obtain the basis vectors of the ε-decimated DWT from those of the standard DWT by applying a shift and corresponds to a special choice of the origin of the basis functions.

How to Calculate the ε-Decimated DWT: SWT

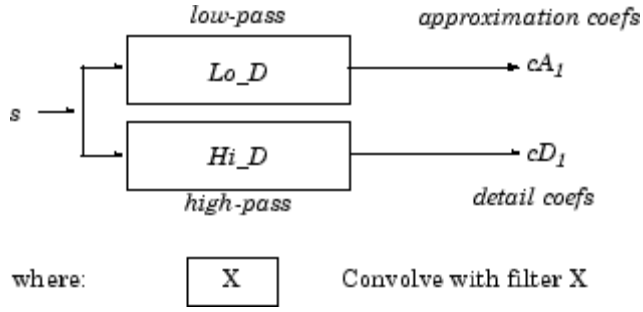
It is possible to calculate all the ε-decimated DWT for a given signal of length N, by computing the approximation and detail coefficients for every possible sequence ε. Do this using iteratively, a slightly modified version of the basic step of the DWT of the form:

```
[A,D] = dwt(X,wname,'mode','per','shift',e);
```

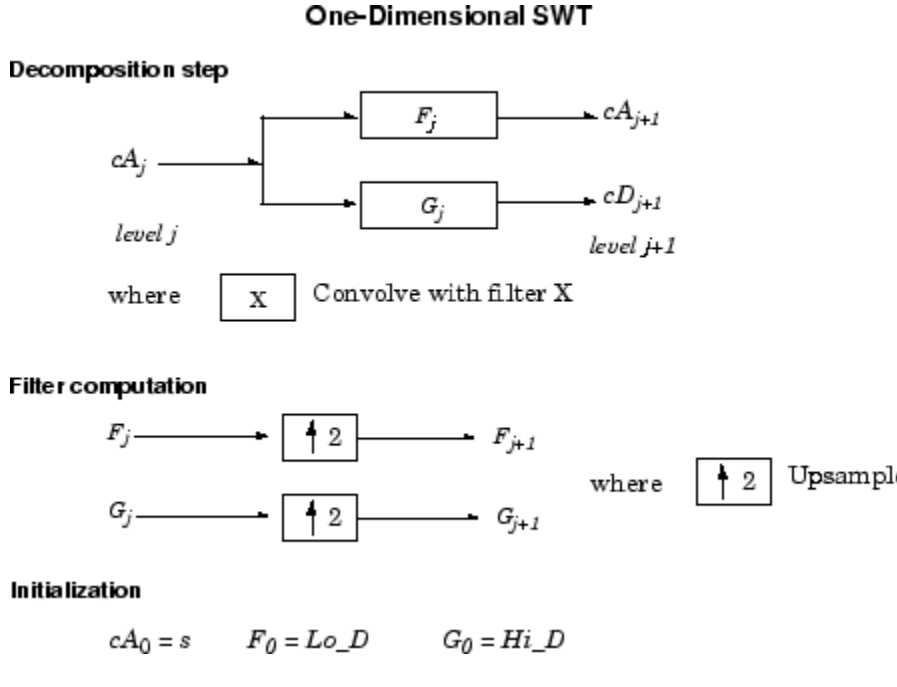
The last two arguments specify the way to perform the decimation step. This is the classical one for `e = 0`, but for `e = 1` the odd indexed elements are retained by the decimation.

Of course, this is not a good way to calculate all the ε-decimated DWT, because many computations are performed many times. We shall now describe another way, which is the stationary wavelet transform (SWT).

The SWT algorithm is very simple and is close to the DWT one. More precisely, for level 1, all the ε-decimated DWT (only two at this level) for a given signal can be obtained by convolving the signal with the appropriate filters as in the DWT case but without downsampling. Then the approximation and detail coefficients at level 1 are both of size N, which is the signal length. This can be visualized in the following figure.



The general step j convolves the approximation coefficients at level j–1, with upsampled versions of the appropriate original filters, to produce the approximation and detail coefficients at level j. This can be visualized in the following figure.



Next, we illustrate how to extract a given ε-decimated DWT from the approximation and detail coefficients structure of the SWT.

We decompose a sequence of height numbers with the SWT, at level $J = 3$, using an orthogonal wavelet.

The function `swt` calculates successively the following arrays, where `A(`$j,\varepsilon_1,...,\varepsilon_j$`)` or `D(`$j,\varepsilon_1,...,\varepsilon_j$`)` denotes an approximation or a detail coefficient at level j obtained for the ε-decimated DWT characterized by $\varepsilon = [\varepsilon_1,...,\varepsilon j]$.

Step 0 (Original Data).

A(0)	A(0)	A(0)	A(0)	A(0)	A(0)	A(0)	A(0)

Step 1.

D(1,0)	**D(1,1)**	D(1,0)	**D(1,1)**	D(1,0)	**D(1,1)**	D(1,0)	**D(1,1)**
A(1,0)	**A(1,1)**	A(1,0)	**A(1,1)**	A(1,0)	**A(1,1)**	A(1,0)	**A(1,1)**

Step 2.

D(1,0)	**D(1,1)**	D(1,0)	**D(1,1)**	D(1,0)	**D(1,1)**	D(1,0)	**D(1,1)**
D(2,0,0)	**D(2,1,0)**	D(2,0,1)	D(2,1,1)	D(2,0,0)	**D(2,1,0)**	D(2,0,1)	D(2,1,1)
A(2,0,0)	**A(2,1,0)**	A(2,0,1)	A(2,1,1)	A(2,0,0)	**A(2,1,0)**	A(2,0,1)	A(2,1,1)

Step 3.

D(1,0)	**D(1,1)**	D(1,0)	**D(1,1)**	D(1,0)	**D(1,1)**	D(1,0)	**D(1,1)**
D(2,0,0)	**D(2,1,0)**	D(2,0,1)	D(2,1,1)	D(2,0,0)	**D(2,1,0)**	D(2,0,1)	D(2,1,1)
D(3,0,0,0)	D(3,1,0,0)	D(3,0,1,0)	D(3,1,1,0)	D(3,0,0,1)	**D(3,1,0,1)**	D(3,0,1,1)	D(3,1,1,1)
A(3,0,0,0)	A(3,1,0,0)	A(3,0,1,0)	A(3,1,1,0)	A(3,0,0,1)	**A(3,1,0,1)**	A(3,0,1,1)	A(3,1,1,1)

Let j denote the current level, where j is also the current step of the algorithm. Then we have the following abstract relations with $\varepsilon_i = 0$ or 1:

```
[tmpAPP,tmpDET] =
dwt(A(j, ε1, , εj),wname,'mode','per','shift', εj+1);
A(j+1, ε1, , εj, εj+1) = wshift('1D',tmpAPP, εj+1);
D(j+1, ε1, , εj, εj+1) = wshift('1D',tmpDET, εj+1);
```

where `wshift` performs a ε-circular shift of the input vector. Therefore, if $\varepsilon_{j+1} = 0$, the `wshift` instruction is ineffective and can be suppressed.

Let $\varepsilon = [\varepsilon_1,...,\varepsilon_J]$ with $\varepsilon_i = 0$ or 1. We have $2^J = 2^3$ = eight different ε-decimated DWTs at level 3. Choosing ε, we can retrieve the corresponding ε-decimated DWT from the SWT array.

Now, consider the last step, $J = 3$, and let [Cε,Lε] denote the wavelet decomposition structure of an ε-decimated DWT for a given ε. Then, it can be retrieved from the SWT decomposition structure by selecting the appropriate coefficients as follows:

Cε =

A(3, ε_1, ε_2, ε_3)	D(3, ε_1, ε_2, ε_3)	D(2, ε_1, ε_2)	D(2, ε_1, ε_2)	D(1, ε_1)	D(1, ε_1)	D(1, ε_1)	D(1, ε_1)

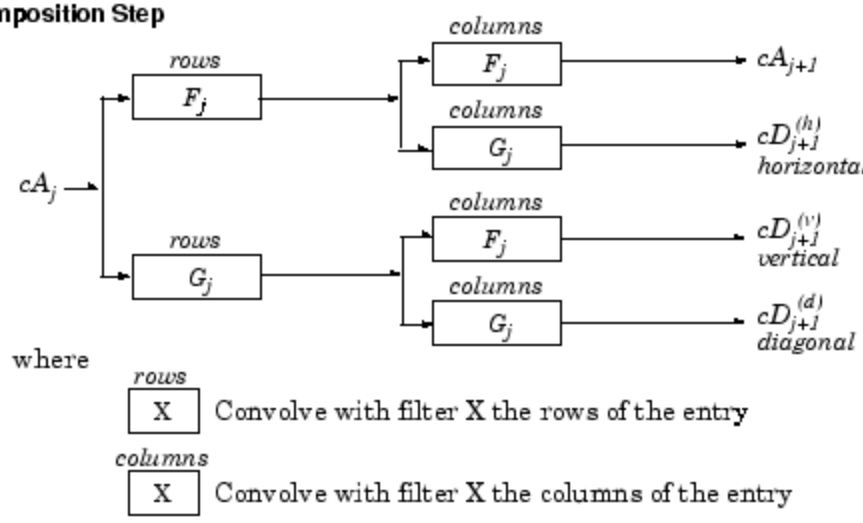
Lε = [1,1,2,4,8]

For example, the ε-decimated DWT corresponding to $\varepsilon = [\varepsilon_1, \varepsilon_2, \varepsilon_3] = [1,0,1]$ is shown in bold in the sequence of arrays of the previous example.

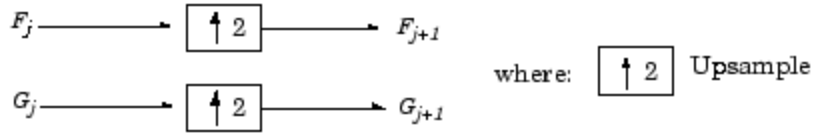
This can be extended to the 2-D case. The algorithm for the stationary wavelet transform for images is visualized in the following figure.

Two-Dimensional SWT

Decomposition Step



Filter Computation



Initialization

$cA_0 = s$ for the decomposition initialization

$F_0 = Lo\_D$

$G_0 = Hi\_D$

Note $size(cA_j) = size(cD_j^{(h)}) = size(cD_j^{(v)}) = size(cD_j^{(d)}) = s$

Where s = *size of the analyzed image*

Inverse Discrete Stationary Wavelet Transform (ISWT)

Each ε-decimated DWT corresponding to a given ε can be inverted.

To reconstruct the original signal using a given ε-decimated DWT characterized by $[\varepsilon_1,...,\varepsilon_J]$, we can use the abstract algorithm

```
FOR j = J:-1:1
  A(j-1, 1, , j-1) = ...
  idwt(A(j, 1, , j),D(S, 1, , j)],wname,'mode','per','shift', j);
END
```

For each choice of $\varepsilon = (\varepsilon_1,...,\varepsilon_J)$, we obtain the original signal A(0), starting from slightly different decompositions, and capturing in different ways the main features of the analyzed signal.

The idea of the inverse discrete stationary wavelet transform is to average the inverses obtained for every ε-decimated DWT. This can be done recursively, starting from level *J* down to level 1.

The ISWT is obtained with the following abstract algorithm:

```
FOR j = J:-1:1
    X0 = idwt(A(j, 1, , j),D(j, 1, , j)],wname, ...
            'mode','per','shift',0);
    X1 = idwt(A(j, 1, , j),D(j, 1, , j)],wname, ...
            'mode','per','shift',1);
    X1 = wshift('1D',X1,1);
    A(j-1, 1, , j-1) = (X0+X1)/2;
END
```

Along the same lines, this can be extended to the 2-D case.

More About SWT

Some useful references for the Stationary Wavelet Transform (SWT) are [CoiD95], [NasS95], and [PesKC96] in "References" on page 6-168.

Lifting Method for Constructing Wavelets

For some applications, you may not be able to find a suitable wavelet among the usual ones widely available. In this case, you can design a new wavelet adapted to the problem to be solved or the task to be processed.

For example, you can adapt a wavelet for the continuous wavelet transform (CWT) to a given pattern so that the resulting wavelet allows accurate pattern detection (see "New Wavelet for CWT" in the *Wavelet Toolbox Getting Started Guide*).

Designing new wavelets that are well suited for the discrete wavelet transform (DWT) is more delicate and, until recently, was exclusively a topic for wavelet specialists. The lifting method proposed by Sweldens (see [Swe98] in "References" on page 6-168) facilitates this kind of construction. It allows you to generate an infinite number of discrete biorthogonal wavelets starting from an initial one.

This section introduces the theory behind lifting methods, then presents the lifting functions of Wavelet Toolbox software and gives two short examples:

- "Lifting Background" on page 6-52
- "Lifting Functions" on page 6-55

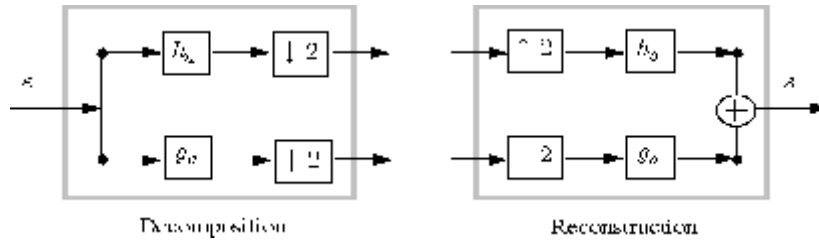
For more information on lifting, see [Swe98], [Mal98], [StrN96], and [MisMOP03] in "References" on page 6-168.

Lifting Background

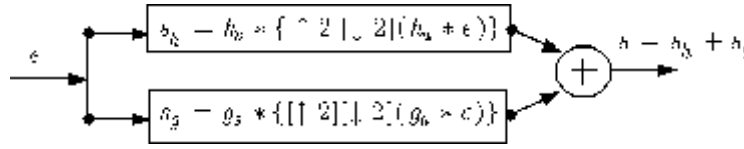
The DWT is defined by four filters as described in "Fast Wavelet Transform (FWT) Algorithm" on page 6-19. Two main properties of interest are

- The perfect reconstruction property
- The link with "true" wavelets (how to generate, starting from the filters, orthogonal or biorthogonal bases of the space of the functions of finite energy)

To illustrate the perfect reconstruction property, the following filter bank contains two decomposition filters, h_a, g_a and two reconstruction filters h_s, g_s.



The *perfect reconstruction property* can be expressed by the equality $s = e$ (up to an eventual shift or delay) where the two signals s and e are defined in the following figure:



This leads to the following two conditions referred to as perfect reconstruction (PR):

$$H_s(z)\, H_a(z^{-1}) + G_s(z)\, G_a(z^{-1}) = 2\, z^{-d}$$

$$H_s(z)\, H_a(-z^{-1}) + G_s(z)\, G_a(-z^{-1}) = 0$$

where $H_s(z)$, $G_s(z)$ are the z-transforms of the filters h_s, g_s respectively, and $H_a(-z^{-1})$ and $G_a(-z^{-1})$ are the z-transforms of h_a, g_a respectively.

The first condition is usually (incorrectly) called the perfect reconstruction condition and the second is the anti-aliasing condition.

Below we refer to the four filters (or equivalently four z-transforms) verifying the (PR) conditions as *biorthogonal quadruplets*.

The principle of lifting is to generate from a given biorthogonal quadruplet a new one by applying a finite sequence of primal or dual elementary lifting steps (ELS).

A *primal ELS* generates from the biorthogonal quadruplet (H_a,G_a,H_s,G_s), a new one (H_a^N,G_a,H_s,G_s^N) by

$$H_a^N(z) = H_a(z) - G_a(z)\, S(z^{-2})$$

$$G_s^N(z) = G_s(z) + H_s(z)\, S(z^2)$$

where S is any Laurent polynomial.

Let us recall that C is a Laurent polynomial if

$$C(z) = c_1\, z^{pmax} + c_2\, z^{pmax-1} + \ldots + c_{end}\, z^{pmin}$$

involving positive and negative integer powers of z. The degree of C is defined as (pmax–pmin).

Similarly, a *dual ELS* generates from the same initial biorthogonal quadruplet, a new one (H_a, G_a^N, H_s^H, G_s) by

$$H_s^N(z) = H_s(z) + G_s(z)\, T(z^2)$$

$$G_a^N(z) = G_a(z) - H_a(z)\, T(z^{-2})$$

where T is any Laurent polynomial.

These new quadruplets verify the perfect reconstruction conditions (PR). Note that even if the initial biorthogonal quadruplet is associated with "true" wavelets, the new ones are not automatically associated with "true" wavelets but remain useful for discrete wavelet transform of sequences instead of functions.

The previous results are sufficient to generate lifted quadruplets. Nevertheless, by introducing the polyphase matrix, interesting theoretical and algorithmic results can be derived. The synthesis polyphase matrix P associated with the biorthogonal quadruplet (H_a, G_a, H_s, G_s) is the 2-by-2 matrix defined (using the MATLAB conventions) by

$$P(z) = [\ \mathrm{even}(H_s)(z)\ \ \mathrm{even}(G_s)(z)\ ;\ \mathrm{odd}(H_s)(z)\ \ \mathrm{odd}(G_s)(z)\ ]$$

where

$$\mathrm{even}(C)(z^2) = (C(z) + C(-z)) / 2$$

$$odd(C)(z^2) = (C(z) - C(-z)) / 2z^{-1}$$

Then after a primal lifting the new polyphase matrix P^N is obtained simply from P the initial one by

$$P^N(z) = P(z) * [1\ S(z)\ ;\ 0\ 1]$$

and after a dual lifting by

$$P^N(z) = P(z) * [1\ 0\ ;\ T(z)\ 1]$$

P itself can be decomposed, up to a normalization, as a product of matrices of the form [1 S(z) ; 0 1] or [1 0 ; T(z) 1] as soon as P is associated with a biorthogonal quadruplet. This form leads to the efficient polyphase algorithm (see [StrN96] in "References" on page 6-168) because the inverses of such elementary matrices are explicit.

Another useful consequence is that any biorthogonal quadruplet can be obtained by a sequence of ELS, up to a normalization, starting from a particular seed called the "lazy" wavelet (which is not a "true" wavelet and which simply separates odd and even samples of the filter bank input signal).

So, in the Wavelet Toolbox software, the key structure to perform what we commonly call the lifting wavelet transform (LWT) is a lifting scheme, which is simply a sequence of ELS and normalization steps.

Lifting Functions

The lifting functions of the toolbox are organized into five groups:

Lifting Schemes

Function Name	Description
`lsinfo`	Information about lifting schemes
`displs`	Display a lifting scheme
`addlift`	Add primal or dual elementary lifting steps to a lifting scheme

Biorthogonal Quadruplets of Filters and Lifting Schemes

These functions connect lifting schemes to biorthogonal quadruplets of filters and associated scaling and wavelet function pairs.

Function Name	Description
`liftfilt`	Apply elementary lifting steps on quadruplet of filters
`filt2ls`	Transform a quadruplet of filters to a lifting scheme
`ls2filt`	Transform a lifting scheme to a quadruplet of filters
`bswfun`	Compute and plot biorthogonal “scaling and wavelet” functions

Usual Biorthogonal Quadruplets

These functions provide some basic lifting schemes associated with some usual orthogonal or biorthogonal (“true”) wavelets and the “lazy” one. These schemes can be used to initialize a lifting procedure.

Function Name	Description
`wavenames`	Provides usual wavelet names available for LWT
`liftwave`	Provides lifting scheme associated with a usual wavelet
`wave2lp`	Provides Laurent polynomials associated with a usual wavelet

Lifting Wavelet Transform (LWT)

These functions contain the direct and inverse lifting wavelet transform (LWT) files for both 1-D and 2-D signals. LWT reduces to the polyphase version of the DWT algorithm with zero-padding extension mode and without extra-coefficients.

Function Name	Description
`lwt`	1-D lifting wavelet transform
`ilwt`	Inverse 1-D lifting wavelet transform
`lwtcoef`	Extract or reconstruct 1-D LWT wavelet coefficients
`lwt2`	2-D lifting wavelet transform
`ilwt2`	Inverse 2-D lifting wavelet transform
`lwtcoef2`	Extract or reconstruct 2-D LWT wavelet coefficients

Laurent Polynomials and Matrices

These functions permit an entry to representation and calculus of Laurent polynomials and matrices.

Function Name	Description
`laurpoly`	Constructor for the class of Laurent polynomials
`laurmat`	Constructor for the class of Laurent matrices

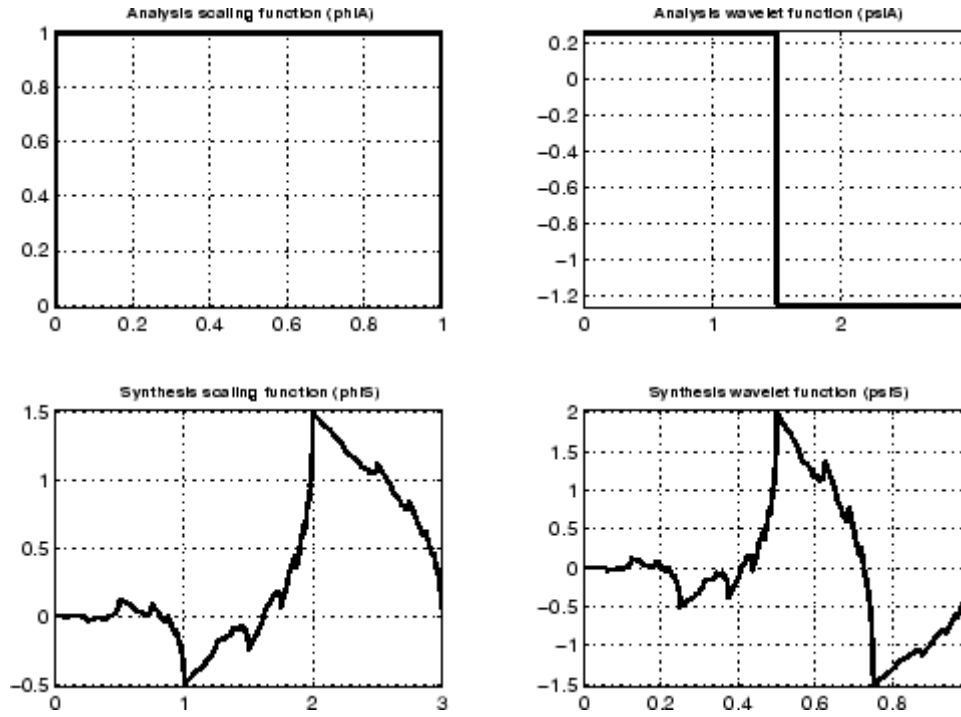
The lifting folder and the two object folders `@laurpoly` and `@laurmat` contain many other files.

Examples of Lifting Methods

These two simple examples illustrate the basic lifting capabilities of Wavelet Toolbox software. For more examples, see Chapter 2, "Wavelets in Action: Examples and Case Studies" and the demos provided with the toolbox.

Example 1. A primal lifting starting from Haar wavelet

```
% Start from the Haar wavelet and get the corresponding
% lifting scheme.
lshaar = liftwave('haar');

% Visualize the obtained lifting scheme.
displs(lshaar);

lshaar = {...
'd'             [ -1.00000000]  [0]
'p'             [  0.50000000]  [0]
[  1.41421356]  [  0.70710678]  []
};

% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);
displs(lsnew);

lsnew = {...
'd'             [ -1.00000000]              [0]
'p'             [  0.50000000]              [0]
'p'             [ -0.12500000  0.12500000]  [0]
[  1.41421356]  [  0.70710678]              []
};

% Transform the lifting scheme to biorthogonal
% filters quadruplet.
[LoD,HiD,LoR,HiR] = ls2filt(lsnew);

% Visualize the two pairs of scaling and wavelet
% functions.
bswfun(LoD,HiD,LoR,HiR,'plot');
```



Illustrating LWT and integer LWT

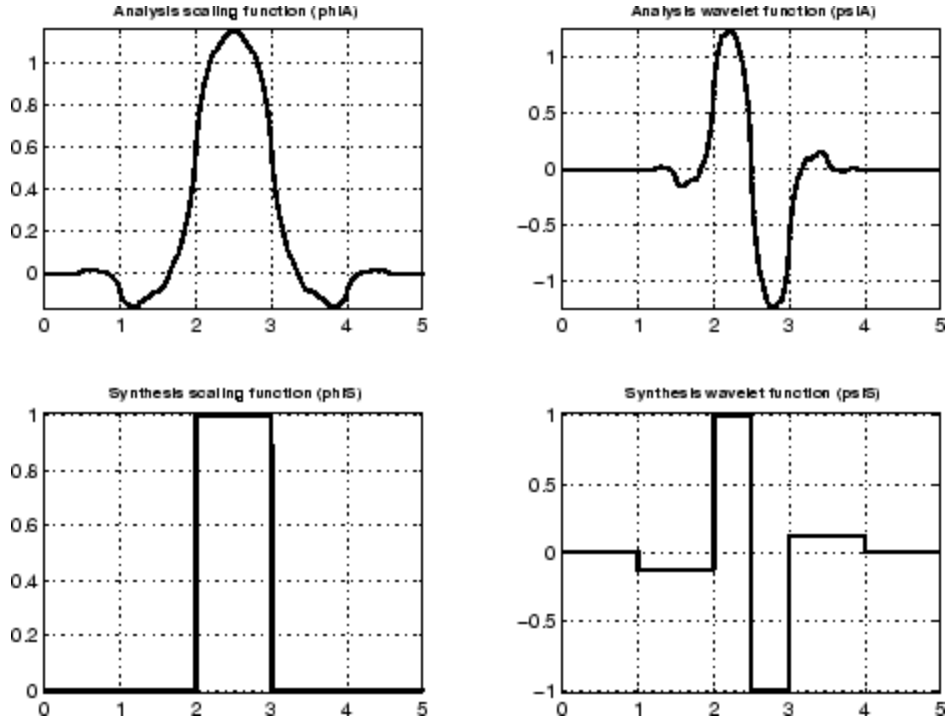
```
% Perform LWT at level 1 of a simple signal.
x = 1:8;
[cA,cD] = lwt(x,lsnew)

cA =

    1.9445    4.9497    7.7782   10.6066

cD =

    0.7071    0.7071    0.7071    0.7071

% Perform
% integer to integer LWT of the same signal.
lshaarInt = liftwave('haar','int2int');
```

```
lsnewInt = addlift(lshaarInt,els);
[cAint,cDint] = lwt(x,lsnewInt)

cAint =

     1     3     5     7

cDint =

     1     1     1     1

% Invert the two transforms.
err = max(max(abs(x-ilwt(cA,cD,lsnew))))

err =

  4.4409e-016

errInt = max(max(abs(x-ilwt(cAint,cDint,lsnewInt))))

errInt =

     0
```

Example 2. Two primal liftings starting from the Haar wavelet

```
% Get Haar filters.
[LoD,HiD,LoR,HiR] = wfilters('haar');

% Lift the Haar filters.
twoels(1) = struct('type','p','value',...
laurpoly([0.125 -0.125],0));
twoels(2) = struct('type','p','value',...
laurpoly([0.125 -0.125],1));
[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,HiD,LoR,HiR,twoels);

% The biorthogonal wavelet bior1.3 is obtained up to
% an unsignificant sign.
[LoDB,HiDB,LoRB,HiRB] = wfilters('bior1.3');
samewavelet = ...
```

```
isequal([LoDB,HiDB,LoRB,HiRB],[LoDN,-HiDN,LoRN,HiRN])

samewavelet =

     1

% Visualize the two times two pairs of scaling and wavelet
% functions.
bswfun(LoDN,HiDN,LoRN,HiRN,'plot');
```



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 nonredundant 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] pages 7, 24, and 27). Usual wavelets satisfy this condition. In which case, a continuous-time signal s is characterized by the knowledge of the discrete transform $C(j,k),(j,k)\epsilon Z^2$.

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 analyses are possible. Which should be used? It depends; each one has its own advantages:

- Discrete analysis ensures space-saving coding and is sufficient for exact reconstruction.
- 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. Thus, the analysis gains in "readability" and in ease of interpretation what it loses in terms of saving space.

Why Are Wavelets Useful for Space-Saving Coding?

The family of functions $(\varphi_{0,k};\psi_{j,l})\ j \le 0,\ (k,l)\epsilon Z^2$ used for the analysis is an orthogonal basis, therefore leading to nonredundancy. The orthogonality properties are $\varphi_{0,k} \perp \psi_{j',k'}$ 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 for a one-dimensional signal, $u \perp v$ stands for

$$\int_R u(x)v(x)dx = 0$$

For biorthogonal wavelets, the idea is similar.

What Is the Advantage Having 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.

As a consequence, the details are also zero. This property ensures the suppression of signals that are polynomials of a degree lower or equal to k.

What About the Regularity of a Wavelet Ψ?

In theoretical and practical studies, the notion of regularity has been increasing in importance. 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, we will define the regularity s of a signal f.

If the signal is s-time continuously differentiable at x_0 and s is an integer (≥ 0), then the regularity is s.

If the derivative of f of order m resembles $|x - x_0|^r$ locally around x_0, then $s = m + r$ with $0 < r < 1$.

The regularity of f in a domain is that of its least regular point.

The greater s, the more regular the signal.

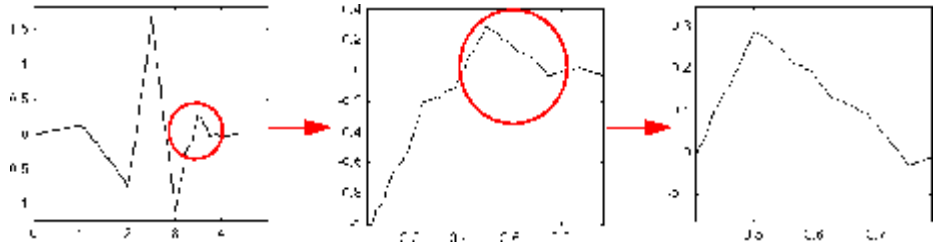
The regularity of certain wavelets is known. The following table gives some indications for Daubechies wavelets.

ψ	**db1 = Haar**	**db2**	**db3**	**db4**	**db5**	**db7**	**db10**
Regularity	Discontinuous	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 $N \to \infty$,

$$\text{length(support)} = 2N, \text{ regularity } | s = \frac{N}{5}$$

The functions are more regular at certain points than at others (see Zooming in on a *db*3 Wavelet on page 6-64).



Zooming in on a *db*3 Wavelet

Selecting a regularity and a wavelet for the 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 by Donoho, Johnstone, Kerkyacharian, and Picard on function estimation and nonlinear regression is currently under way to adapt the statistical estimators to unknown regularity. See also the remarks by I. Daubechies (see [Dau92] in "References" on page 6-168).

From a practical viewpoint, these questions arise in the world of finance in dealing with monetary and stock markets where detailed studies of very fast transactions are required.

Are Wavelets Useful in Fields Other Than Signal or Image Processing?

- From a theoretical viewpoint, wavelets are used to characterize large sets of mathematical functions and are used in the study of operators linked to partial differential equations.
- From a practical viewpoint, 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.

More precisely, the admissibility condition for $\psi \in L^1(R) \cap L^2(R)$ is

$$\int_{R^-} \frac{|\hat{\psi}(s)|^2}{|s|} ds = \int_{R^+} \frac{|\hat{\psi}(s)|^2}{|s|} ds = K_\psi < +\infty$$

The family of shifts and dilations of ψ allows all finite energy signals to be reconstructed using the details in all scales. This allows only continuous analysis.

A wavelet satisfying only the admissibility condition is said to be crude.

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 Mexican hat wavelet.

Is It Easy to Build a New Wavelet?

For a minimal requirement on the wavelet properties, it is easy to build a new wavelet but not very interesting unless the new wavelet is adapted

to a specific task. For example the paragraph "New Wavelet for CWT" in the *Wavelet Toolbox Getting Started Guide* explains how to obtain wavelets adapted to a given pattern, which can then be used for an accurate pattern detection. If more interesting properties (like the existence of φ for example) are needed, then building the wavelet is more difficult. Let us mention that an interesting approach is the lifting method (see "Lifting Method for Constructing Wavelets" on page 6-52).

Very few wavelets have an explicit analytical expression. Notable exceptions are wavelets that are piecewise polynomials (Haar, Battle-Lemarié; see [Dau92] in "References" on page 6-168), Morlet, or Mexican hat.

Wavelets, even `db2`, `db3`, ..., are defined by functional equations. The solution is numerical, and is accomplished using a fairly simple algorithm.

The basic property is the existence of a linear relation between the two functions $\varphi(x/2)$ and $\varphi(x)$. Another relation of the same type links $\psi(x/2)$ to $\varphi(x)$. These are the relations of the two scales, the twin-scale relations.

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 "References" on page 6-168) and are used extensively in the toolbox. The reader can refer to page 164 and Chapter 10 of the book *Wavelets and Filter Banks*, by Strang and Nguyen (see [StrN96] in "References" on page 6-168).

What Is the Link Between Wavelet and Fourier Analysis?

Wavelet analysis complements the Fourier analysis for which there are several functions: `fft` in MATLAB software and `spectrum` and `sptool` in Signal Processing Toolbox™ software.

Fourier analysis uses the basic functions sin(ω*t*), cos(ω*t*), and exp(*i*ω*t*).

- In the frequency domain, these functions are perfectly localized. 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 calls it (see page 2 of [Dau92] in "References" on page 6-168). The windowed-Fourier analysis coefficients are the doubly indexed coefficients:

$$G_s(\omega,t) = \int_R s(u) g(t-u) e^{-i\omega u} du$$

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 $Gs(\omega,t)$ depends on the values of the signal s on the segment $[t - m, t + m]$ with a constant width. 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 $[t - am, t + am]$ of width $2am$, which varies as a function of a. This slight difference solves several difficulties, allowing a kind of time-windowed analysis, different at the various scales a.

The wavelets stay competitive, however, even in contexts considered favorable for the Fourier technique. I. Daubechies (see [Dau92] pages 3 to 6) gives an example of windowed-Fourier processing and complex Morlet wavelet processing, $\psi(t) = Ce^{-t^2/a^2}(e^{i\pi t} - e^{-\pi^2 a^2/4})$ with $a = 4$, of a signal composed mainly of the sum of two sines. This wavelet analysis gives good results.

How to Connect Scale to Frequency?

A common question is, what is the relationship between scale and frequency?

The answer can only be given in a broad sense, and it's better to speak about the pseudo-frequency corresponding to a scale.

A way to do it is to compute the center frequency F_c of the wavelet and to use the following relationship (see [Abr97] in "References" on page 6-168).

$$F_a = \frac{F_c}{a \cdot \Delta}$$

where

- a is a scale.
- Δ is the sampling period.
- F_c is the center frequency of a wavelet in Hz.
- F_a is the pseudo-frequency corresponding to the scale a, in Hz.

The idea is to associate with a given wavelet a purely periodic signal of frequency F_c. The frequency maximizing the `fft` of the wavelet modulus is

F_c. The function `centfrq` can be used to compute the center frequency and it allows the plotting of the wavelet with the associated approximation based on the center frequency. Center Frequencies for Real and Complex Wavelets on page 6-70 shows some examples generated using the `centfrq` function.

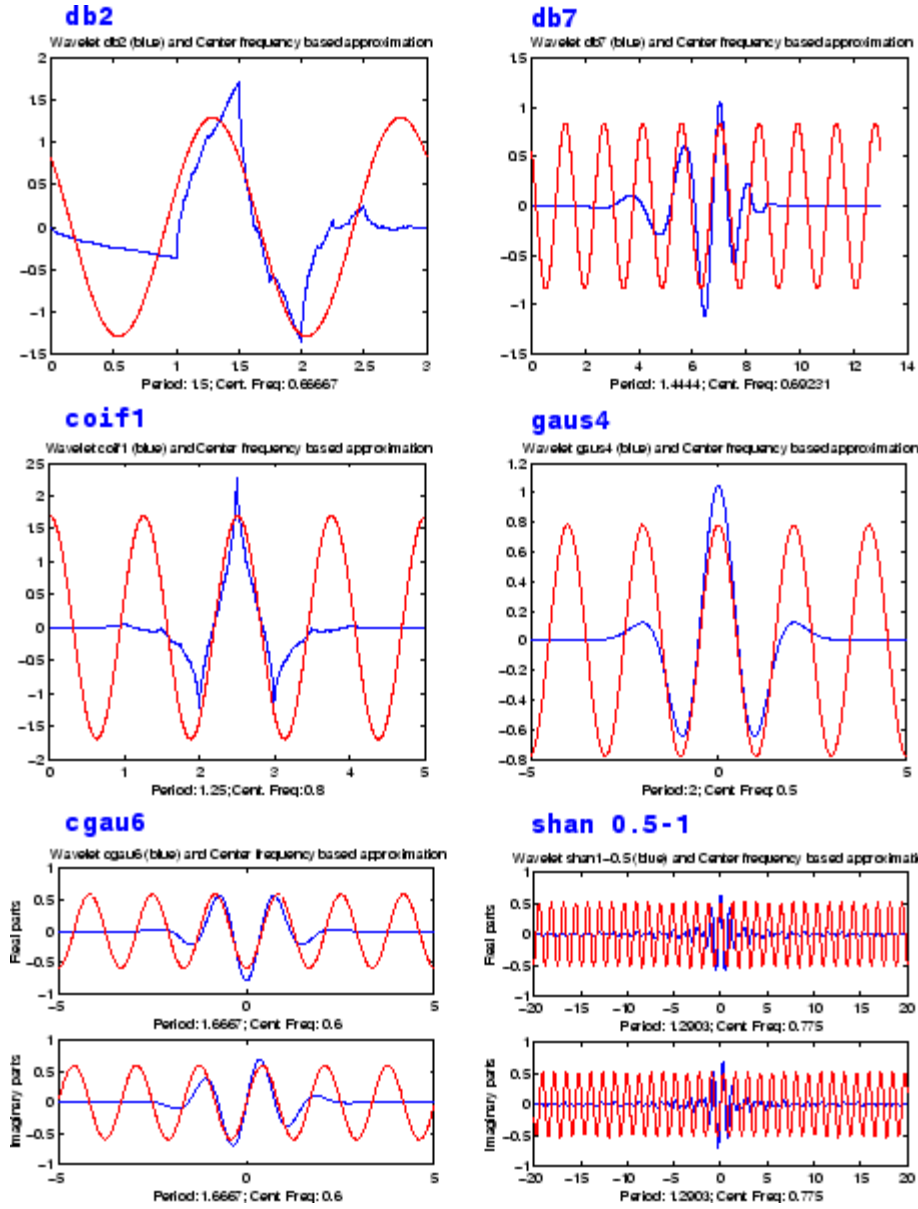
- Four real wavelets: Daubechies wavelets of order 2 and 7, coiflet of order 1, and the Gaussian derivative of order 4.
- Two complex wavelets: the complex Gaussian derivative of order 6 and a Shannon complex wavelet.

As you can see, the center frequency-based approximation captures the main wavelet oscillations. So the center frequency is a convenient and simple characterization of the leading dominant frequency of the wavelet.

If we accept to associate the frequency F_c to the wavelet function, then when the wavelet is dilated by a factor a, this center frequency becomes F_c / a. Lastly, if the underlying sampling period is Δ, it is natural to associate to the scale a the frequency

$$F_a = \frac{F_c}{a \cdot \Delta}$$

The function `scal2frq` computes this correspondence.



Center Frequencies for Real and Complex Wavelets

To illustrate the behavior of this procedure, consider the following simple test. We generate sine functions of sensible frequencies F_0. For each function, we shall try to detect this frequency by a wavelet decomposition followed by a translation of scale to frequency. More precisely, after a discrete wavelet decomposition, we identify the scale a^* corresponding to the maximum value of the energy of the coefficients. The translated frequency F^* is then given by

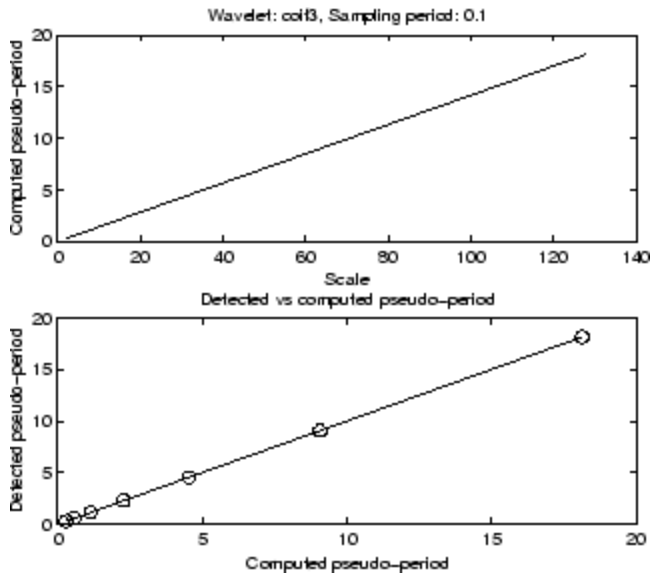
```
scal2frq(a_star,'wname',sampling_period)
```

The F^* values are close to the chosen F_0. The plots at the end of the example present the periods instead of the frequencies. If we change the F_0 values slightly, the results remain satisfactory.

For example:

```
% Set sampling period and wavelet name.
delta = 0.1; wname = 'coif3';

% Set scales.
amax = 7;
a = 2.^[1:amax];

% Compute associated pseudo-frequencies.
f = scal2frq(a,wname,delta);

% Compute associated pseudo-periods.
per = 1./f;

% Plot pseudo-periods versus scales.
subplot(211), plot(a,per)
title(['Wavelet: ',wname, ', Sampling period: ',num2str(delta)])
xlabel('Scale')
ylabel('Computed pseudo-period')

% For each scale 2^i:
% - generate a sine function of period per(i);
% - perform a wavelet decomposition;
% - identify the highest energy level;
% - compute the detected pseudo-period.
for i = 1:amax
```

```
    % Generate sine function of period
    % per(i) at sampling period delta.
    t = 0:delta:100;
    x = sin((t.*2*pi)/per(i));

    % Decompose x at level 9.
    [c,l] = wavedec(x,9,wname);

    % Estimate standard deviation of detail coefficients.
    stdc = wnoisest(c,l,[1:amax]);
    % Compute identified period.
    [y,jmax] = max(stdc);
    idper(i) = per(jmax);
end

% Compare the detected and computed pseudo-periods.
subplot(212), plot(per,idper,'o',per,per)
title('Detected vs computed pseudo-period')
xlabel('Computed pseudo-period')
ylabel('Detected pseudo-period')
```



Detected Versus Computed Pseudo-Periods

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,ψ (and ϕ,ϕ): the speed of convergence to 0 of these functions $(\psi(t)$ or $\psi(\omega))$ when the time t or the frequency ω 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 purposes
- The regularity, which is useful for getting nice features, like smoothness of the reconstructed signal or image, and for the estimated function in nonlinear regression analysis

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

They may also be associated with these less important properties:

- 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.

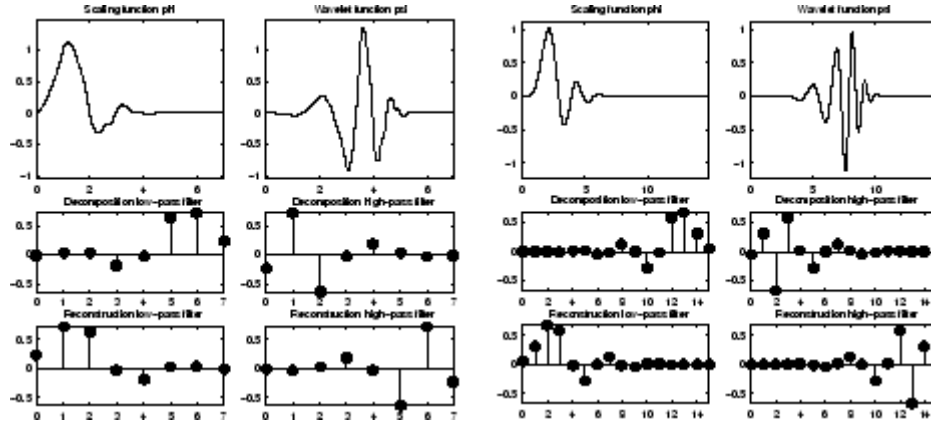
Note that the φ and ψ functions can be computed using `wavefun`; the filters are generated using `wfilters`. We provide definition equations for several wavelets. Some are given explicitly by their time definitions, others by their frequency definitions, and still others by their filters.

The following table outlines the wavelet families included in the toolbox.

Wavelet Family Short Name	Wavelet Family Name
'haar'	Haar wavelet
'db'	Daubechies wavelets
'sym'	Symlets
'coif'	Coiflets
'bior'	Biorthogonal wavelets
'rbio'	Reverse biorthogonal wavelets
'meyr'	Meyer wavelet
'dmey'	Discrete approximation of Meyer wavelet
'gaus'	Gaussian wavelets
'mexh'	Mexican hat wavelet
'morl'	Morlet wavelet
'cgau'	Complex Gaussian wavelets
'shan'	Shannon wavelets
'fbsp'	Frequency B-Spline wavelets
'cmor'	Complex Morlet 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] pages 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.



Daubechies Wavelets db4 on the Left and db8 on the Right

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	$x \in [0, 0.5)$
$\psi(x) = -1,$	if	$x \in [0.5, 1)$
$\psi(x) = 0,$	if	$x \notin [0, 1)$
$\varphi(x) = 1,$	if	$x \in [0, 1)$
$\varphi(x) = 0,$	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

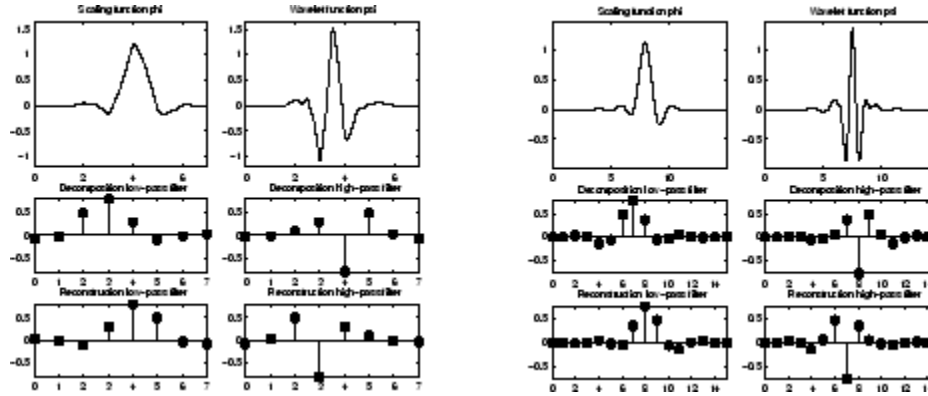
$$|m_0(\omega)|^2 = \left(\cos^2\left(\frac{\omega}{2}\right)\right)^N P\left(\sin^2\left(\frac{\omega}{2}\right)\right)$$

$$\text{where } m_0(\omega) = \frac{1}{\sqrt{2}}\sum_{k=0}^{2N-1} h_k e^{-ik\omega}$$

- 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 $C^{\mu N}$ where μ is approximately equal to 0.2. Certainly, 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 $2N$ instead of N. Symlets are only near symmetric; consequently some authors do not call them symlets. More about symlets can be found in [Dau92], pages 194, 254-257. By typing `waveinfo('sym')` at the MATLAB command prompt, you can obtain a survey of the main properties of this family.



Symlets sym4 on the Left and sym8 on the Right

Daubechies proposes modifications of her wavelets that increase 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

$W(z) = U(z)\overline{U\left(\frac{1}{Z}\right)}$ because the roots of W with modulus not equal to 1 go in pairs. If one of the roots is z_1, then $\frac{1}{\overline{z_1}}$ is also a root.

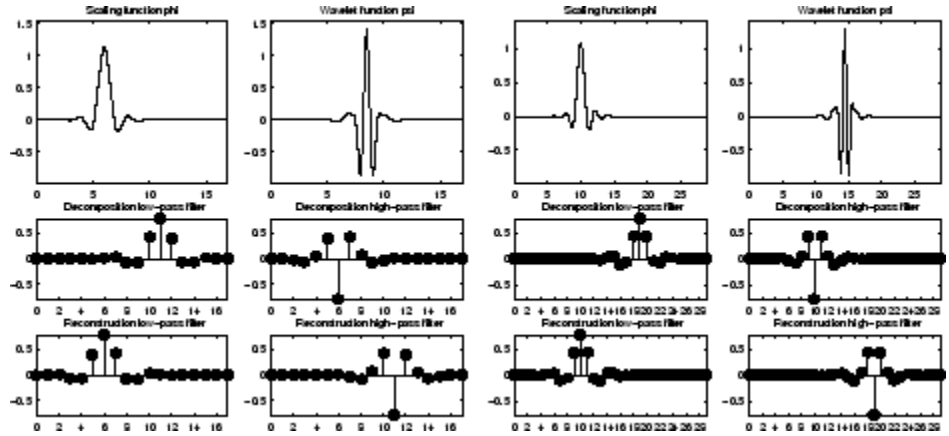
- By selecting U such that the modulus of all its roots is strictly less than 1, we build Daubechies wavelets *dbN*. The U filter is a "minimum phase filter."
- By making another choice, we obtain more symmetrical filters; these are symlets.

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] pages 258–259. By typing `waveinfo('coif')` at the

MATLAB command prompt, you can obtain a survey of the main properties of this family.



Coiflets coif3 on the Left and coif5 on the Right

Built by Daubechies at the request of Coifman, the function ψ has $2N$ moments equal to 0 and, what is more unusual, the function φ has $2N$–1 moments equal to 0. The two functions have a support of length $6N$–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 sufficiently regular continuous time signal, for large j the coefficient

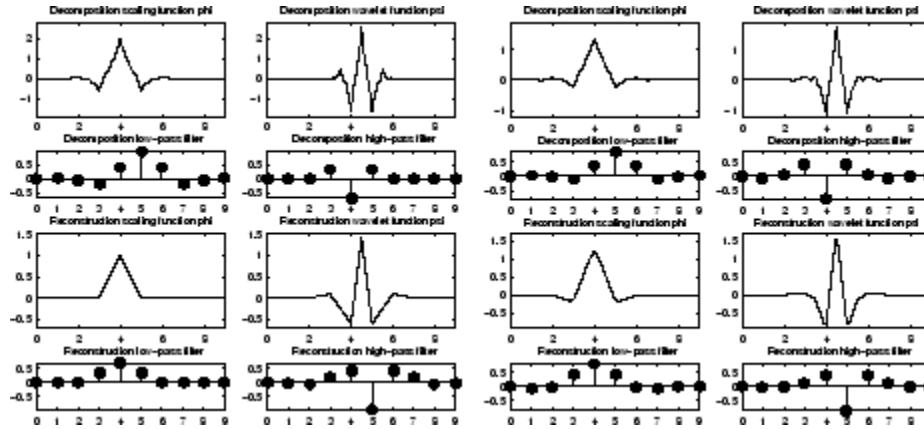
$\left\langle s, \phi_{-j,k} \right\rangle$ is approximated by $2^{-j/2} s(2^{-j}k)$.

If s is a polynomial of degree d, $d \leq N-1$, then the approximation becomes an equality. This property is used, connected with sampling problems, when calculating the difference between an expansion over the $\varphi_{j,l}$ of a given signal and its sampled version.

Biorthogonal Wavelet Pairs: biorNr.Nd

More about biorthogonal wavelets can be found in [Dau92] pages 259, 262–85 and in [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.



Biorthogonal Wavelets bior2.4 on the Left and bior4.4 on the Right

The new family extends the wavelet family. It is well known in the subband filtering community that symmetry and exact reconstruction are incompatible (except for the Haar wavelet) 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 ψ and $\tilde{\psi}$ 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, that "the useful properties for analysis (e.g., oscillations, zero moments) can be concentrated on the ψ function whereas the interesting properties for synthesis (regularity) are assigned to the ψ function. The separation of these two tasks proves very useful" (see [Coh92] page 110).

$\tilde{\psi},\psi$ can have very different regularity properties (see [Dau92] page 269).

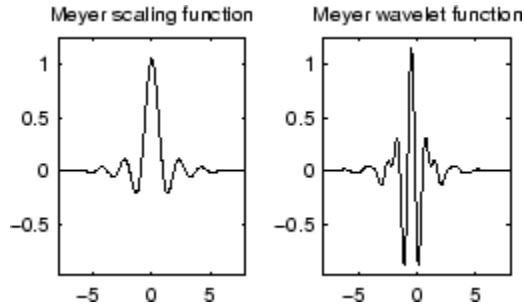
The $\tilde{\psi},\psi,\tilde{\phi}$, and ϕ functions are zero outside of a segment.

The calculation algorithms are maintained, and thus very simple.

The filters associated with 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] pages 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.



Meyer Wavelet

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

- Wavelet function

$$\hat{\psi}(\omega) = (2\pi)^{-1/2} e^{i\omega/2} \sin\left(\frac{\pi}{2}\nu\left(\frac{3}{2\pi}|\omega| - 1\right)\right) \quad if \quad \frac{2\pi}{3} \le |\omega| \le \frac{4\pi}{3}$$

$$\hat{\psi}(\omega) = (2\pi)^{-1/2} e^{i\omega/2} \cos\left(\frac{\pi}{2}\nu\left(\frac{3}{4\pi}|\omega| - 1\right)\right) \quad if \quad \frac{4\pi}{3} \le |\omega| \le \frac{8\pi}{3}$$

$$\text{and } \hat{\psi}(\omega) = 0 \quad if \quad |\omega| \notin \left[\frac{2\pi}{3}, \frac{8\pi}{3}\right]$$

$$\text{where } \nu(a) = a^4\left(35 - 84a + 70a^2 - 20a^3\right) \qquad a \in [0,1]$$

- Scaling function

$$\hat{\phi}(\omega) = (2\pi)^{-1/2} \qquad if \quad |\omega| \le \frac{2\pi}{3}$$

$$\hat{\phi}(\omega) = (2\pi)^{-1/2} \cos\left(\frac{\pi}{2}\nu\left(\frac{3}{2\pi}|\omega| - 1\right)\right) \qquad if \quad \frac{2\pi}{3} \le |\omega| \le \frac{4\pi}{3}$$

$$\hat{\phi}(\omega) = 0 \qquad if \quad |\omega| > \frac{4\pi}{3}$$

By changing the auxiliary function, you get a family of different wavelets. For the required properties of the auxiliary function ν (see "References" on page 6-168 for more information). 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

$$\forall n \in \mathrm{N}, \exists C_n \text{ such that } |\psi(x)| \le C_n\left(1 + |x|^2\right)^{-n}$$

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.

Note Although the Meyer wavelet is not compactly supported, there exists a good approximation leading to FIR filters, and then allowing DWT. By typing `waveinfo('dmey')` at the MATLAB command prompt, you can obtain a survey of the main properties of this pseudo-wavelet.

Battle-Lemarie Wavelets

See [Dau92] pages 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 orthogonal, 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-spline,

$$\hat{\phi}(\omega) = (2\pi)^{-1/2} e^{-i\kappa\omega/2} \left[\frac{\sin(\omega/2)}{\omega/2} \right]^{-N+1}$$

with $\kappa = 0$ if N is odd, $\kappa = 1$ if N is even.

This formula can be used to build the filters. The twin scale relation is

$$\phi(x) = 2^{-2M} \sum_{j=0}^{2M+1} C_j^{2M+1} \phi(2x - M - 1 + j) \text{ if } N = 2M$$

$$\phi(x) = 2^{-2M-1} \sum_{j=0}^{2M+2} C_j^{2M+2} \phi(2x - M - 1 + j) \text{ if } N = 2M + 1$$

- For an even N, φ is symmetrical around, $x = 1/2$; ψ is antisymmetrical 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

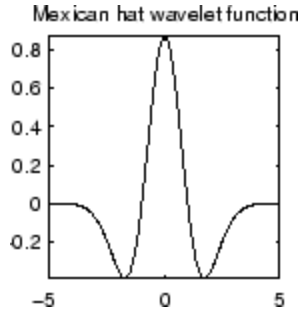
$$\hat{\phi}^{\perp}(\omega) = 3^{1/2}(2\pi)^{-1/2}\frac{4\sin^2(\omega/2)}{\omega^2\left[1+2\cos^2(\omega/2)\right]^{1/2}}$$

- The supports of ψ and $\phi^{\perp}$ are not finite, but the decrease of the functions ψ and $\phi^{\perp}$ to 0 is exponential. The support of φ is compact. See [Dau92] p. 151.
- The ψ functions have derivatives up to order *N*–1.

Mexican Hat Wavelet: mexh

See [Dau92] page 75.

By typing `waveinfo('mexh')` at the MATLAB command prompt, you can obtain a survey of the main properties of this wavelet.



Mexican Hat

$$\psi(x) = \left(\frac{2}{\sqrt{3}}\pi^{-1/4}\right)(1-x^2)e^{-x^2/2}$$

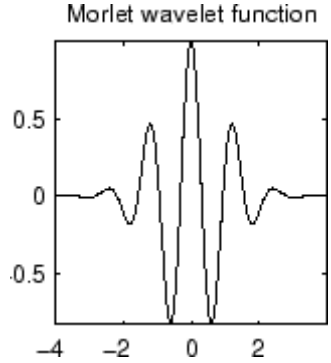
This function is proportional to the second derivative function of the Gaussian probability density function.

As the φ function does not exist, the analysis is not orthogonal.

Morlet Wavelet: morl

See [Dau92] page 76.

By typing `waveinfo('morl')` at the MATLAB command prompt you can obtain a survey of the main properties of this wavelet.



Morlet Wavelet

$$\psi(x) = Ce^{-x^2}\cos(5x)$$

The constant C is used for normalization in view of reconstruction.

The Morlet wavelet does not satisfy exactly the admissibility condition discussed in “What Functions Are Candidates to Be a Wavelet?” on page 6-65.

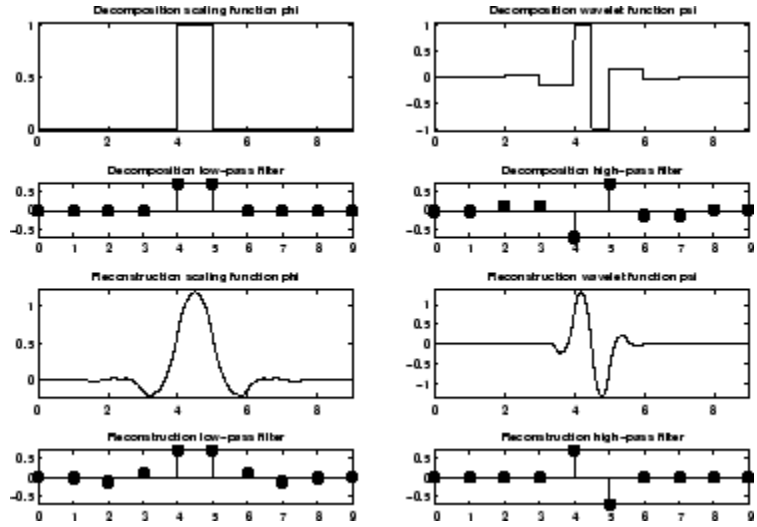
Additional Real Wavelets

Some other real wavelets are available in the toolbox.

Reverse Biorthogonal Wavelet Pairs: rbioNr.Nd

This family is obtained from the biorthogonal wavelet pairs previously described.

You can obtain a survey of the main properties of this family by typing `waveinfo('rbio')` from the MATLAB command line.



Reverse Biorthogonal Wavelet rbio1.5

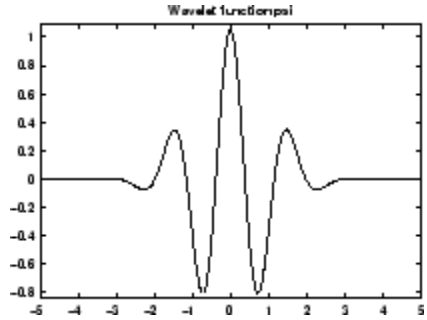
Gaussian Derivatives Family: gaus

This family is built starting from the Gaussian function $f(x) = C_p e^{-x^2}$ by taking the p^{th} derivative of f.

The integer p is the parameter of this family and in the previous formula, C_p is such that

$\left\| f^{(p)} \right\|^2 = 1$ where $f^{(p)}$ is the p^{th} derivative of f.

You can obtain a survey of the main properties of this family by typing `waveinfo('gaus')` from the MATLAB command line.



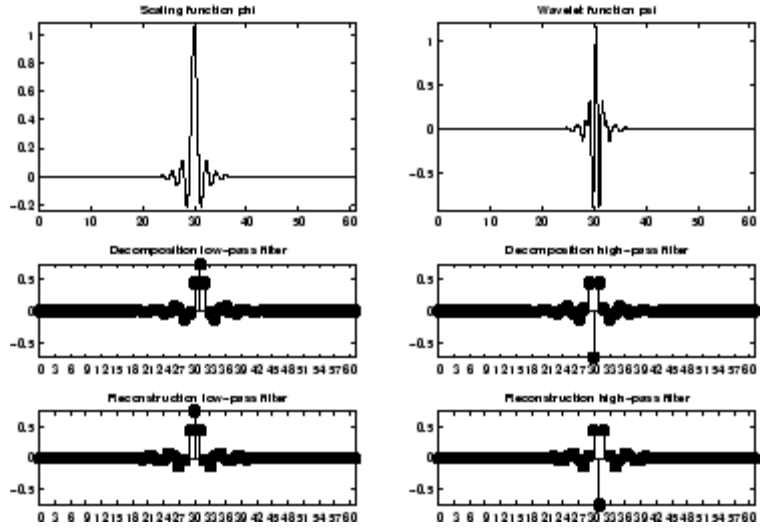
Gaussian Derivative Wavelet gaus8

FIR Based Approximation of the Meyer Wavelet: dmey

See [Abr97] page 268.

This wavelet is a FIR based approximation of the Meyer wavelet, allowing fast wavelet coefficients calculation using DWT.

You can obtain a survey of the main properties of this wavelet by typing `waveinfo('dmey')` from the MATLAB command line.



FIR Based Approximation of the Meyer Wavelet

Complex Wavelets

Some complex wavelet families are available in the toolbox.

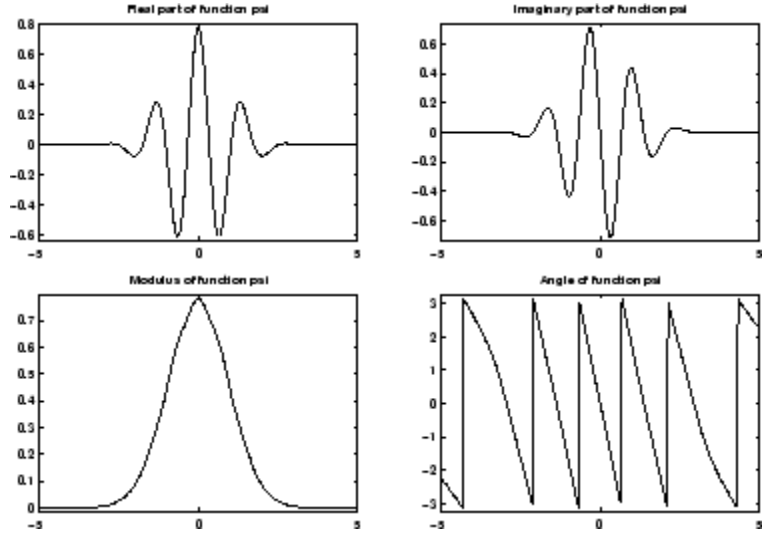
Complex Gaussian Wavelets: cgau

This family is built starting from the complex Gaussian function

$f(x) = C_p e^{-ix} e^{-x^2}$ by taking the p^{th} derivative of f. The integer p is the parameter of this family and in the previous formula, C_p is such that

$\left\| f^{(p)} \right\|^2 = 1$ where $f^{(p)}$ is the p^{th} derivative of f.

You can obtain a survey of the main properties of this family by typing `waveinfo('cgau')` from the MATLAB command line.



Complex Gaussian Wavelet cgau8

Complex Morlet Wavelets: cmor

See [Teo98] pages 62–65.

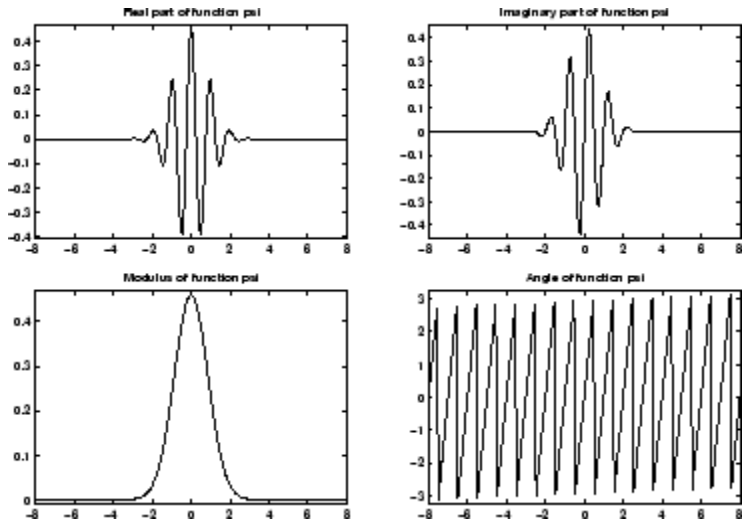
A complex Morlet wavelet is defined by

$$\psi(x) = \frac{1}{\sqrt{\pi f_b}} e^{2i\pi f_c x} e^{\frac{x^2}{f^b}}$$

depending on two parameters:

- f_b is a bandwidth parameter.
- f_c is a wavelet center frequency.

You can obtain a survey of the main properties of this family by typing `waveinfo('cmor')` from the MATLAB command line.



Complex Morlet Wavelet morl 1.5-1

Complex Frequency B-Spline Wavelets: fbsp

See [Teo98] pages 62–65.

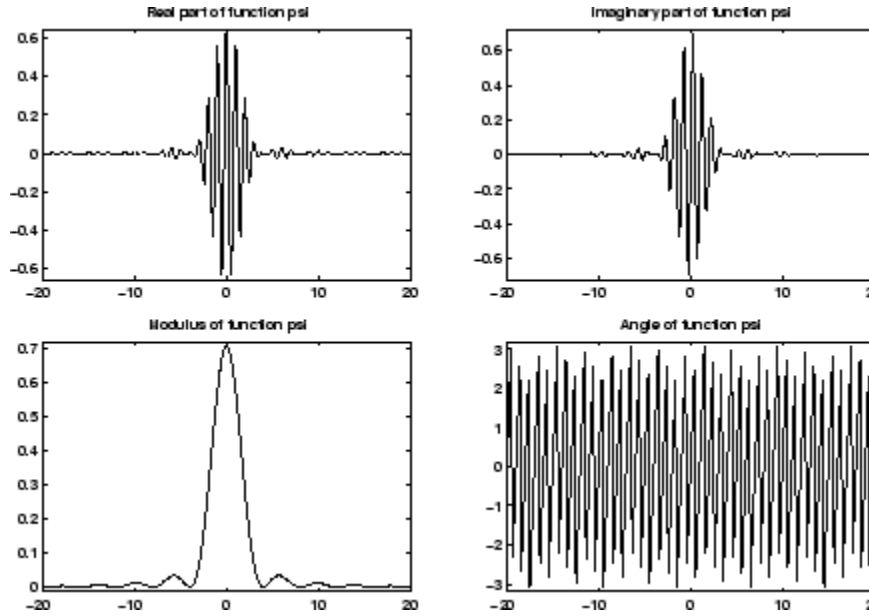
A complex frequency B-spline wavelet is defined by

$$\psi(x) = \sqrt{f_b}\left(\operatorname{sinc}\left(\frac{f_b x}{m}\right)\right)^m e^{2i\pi f_c x}$$

depending on three parameters:

- m is an integer order parameter ($m \geq 1$).
- f_b is a bandwidth parameter.
- f_c is a wavelet center frequency.

You can obtain a survey of the main properties of this family by typing `waveinfo('fbsp')` from the MATLAB command line.



Complex Frequency B-Spline Wavelet fbsp 2-0.5-1

Complex Shannon Wavelets: shan

See [Teo98] pages 62–65.

This family is obtained from the frequency B-spline wavelets by setting m to 1.

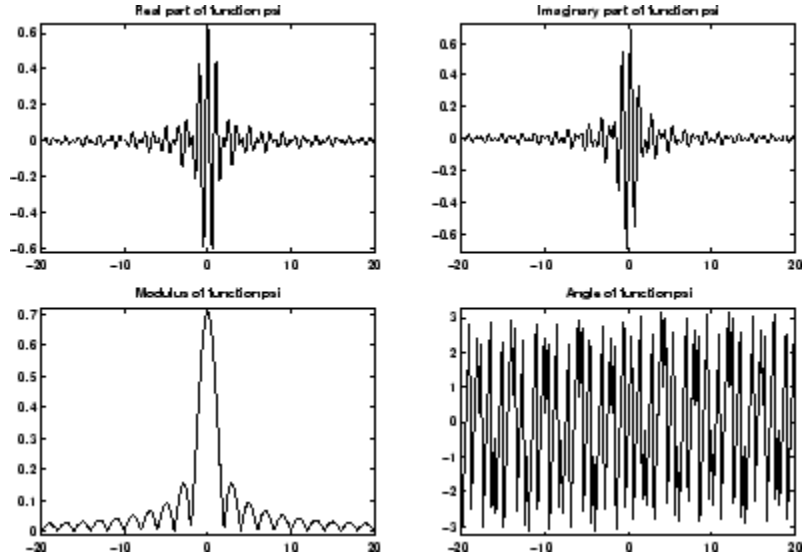
A complex Shannon wavelet is defined by

$$\psi(x) = \sqrt{f_b}\,\mathrm{sinc}\left(f_b x\right)e^{2i\pi f_c x}$$

depending on two parameters:

- f_b is a bandwidth parameter.
- f_c is a wavelet center frequency.

You can obtain a survey of the main properties of this family by typing `waveinfo('shan')` from the MATLAB command line.



Complex Shannon Wavelet shan 0.5-1

Summary of Wavelet Families and Associated Properties (Part 1)

Property	morl	mexh	meyr	haar	*db*N	symN	coifN	biorNr.Nd
Crude	•	•						
Infinitely regular	•	•	•					
Arbitrary regularity					•	•	•	•
Compactly supported orthogonal				•	•	•	•	
Compactly supported biothogonal								•
Symmetry	•	•	•	•				•
Asymmetry					•			
Near symmetry						•	•	
Arbitrary number of vanishing moments					•	•	•	•
Vanishing moments for φ							•	

Property	morl	mexh	meyr	haar	*db*N	symN	coifN	biorNr.Nd
Existence of φ			•	•	•	•	•	•
Orthogonal analysis			•	•	•	•	•	
Biorthogonal analysis			•	•	•	•	•	•
Exact reconstruction	≈	•	•	•	•	•	•	•
FIR filters				•	•	•	•	•
Continuous transform	•	•	•	•	•	•	•	•
Discrete transform				•	•	•	•	•
Fast algorithm				•	•	•	•	•
Explicit expression	•	•		•				For splines

Crude wavelet — A wavelet is said to be crude when satisfying only the admissibility condition. See "What Functions Are Candidates to Be a Wavelet?" on page 6-65.

Regularity — See "What About the Regularity of a Wavelet Ψ?" on page 6-63.

Orthogonal — See "Details and Approximations" on page 6-15.

Biorthogonal — See "Biorthogonal Wavelet Pairs: biorNr.Nd" on page 6-78.

Vanishing moments — See "Suppressing Signals" on page 6-97.

Exact reconstruction — See "Reconstruction Filters" in the *Wavelet Toolbox Getting Started Guide*.

Continuous — See "Continuous Wavelet Transform" in the *Wavelet Toolbox Getting Started Guide*.

Discrete — See "Discrete Wavelet Transform" in the *Wavelet Toolbox Getting Started Guide*.

FIR filters — See "Filters Used to Calculate the DWT and IDWT" on page 6-19.

Summary of Wavelet Families and Associated Properties (Part 2)

Property	rbioNr.Nd	gaus	dmey	cgau	cmor	fbsp	shan
Crude		•		•	•	•	•
Infinitely regular		•		•	•	•	•
Arbitrary regularity	•						
Compactly supported orthogonal							
Compactly supported biothogonal	•						

Property	rbioNr.Nd	gaus	dmey	cgau	cmor	fbsp	shan
Symmetry	•	•	•	•	•	•	•
Asymmetry							
Near symmetry							
Arbitrary number of vanishing moments	•						
Vanishing moments for φ							
Existence of φ	•						
Orthogonal analysis							
Biorthogonal analysis	•						
Exact reconstruction	•	•	≈	•	•	•	•
FIR filters	•		•				
Continuous transform	•	•					
Discrete transform	•		•				
Fast algorithm	•		•				

Property	rbioNr.Nd	gaus	dmey	cgau	cmor	fbsp	shan
Explicit expression	For splines	•		•	•	•	•
Complex valued				•	•	•	•
Complex continuous transform				•	•	•	•
FIR-based approximation			•				

Crude wavelet — A wavelet is said to be crude when satisfying only the admissibility condition. See "What Functions Are Candidates to Be a Wavelet?" on page 6-65.

Regularity — See "What About the Regularity of a Wavelet Ψ?" on page 6-63.

Orthogonal — See "Details and Approximations" on page 6-15.

Biorthogonal — See "Biorthogonal Wavelet Pairs: biorNr.Nd" on page 6-78.

Vanishing moments — See "Suppressing Signals" on page 6-97.

Exact reconstruction — See "Reconstruction Filters" in the *Wavelet Toolbox Getting Started Guide*.

Continuous — See "Continuous Wavelet Transform" in the *Wavelet Toolbox Getting Started Guide*.

Discrete — See "Discrete Wavelet Transform" in the *Wavelet Toolbox Getting Started Guide*.

FIR filters — See “Filters Used to Calculate the DWT and IDWT” on page 6-19.

Wavelet Applications: More Detail

Chapter 1, "Wavelet Applications" and Chapter 2, "Wavelets in Action: Examples and Case Studies" illustrate wavelet applications with examples and case studies. This section reexamines some of the applications with additional theory and more detail.

Suppressing Signals

As shown in "Suppressing Signals" on page 1-15, 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 $j = 0, \ldots, k$, $\int_R x^j \psi(x) dx = 0$

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 x, provided that it remains less than k.

If s is now a polynomial of degree k on segment [α,β], 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 [α,β]. The suppression is local. Effects will appear on the edges of the segment.

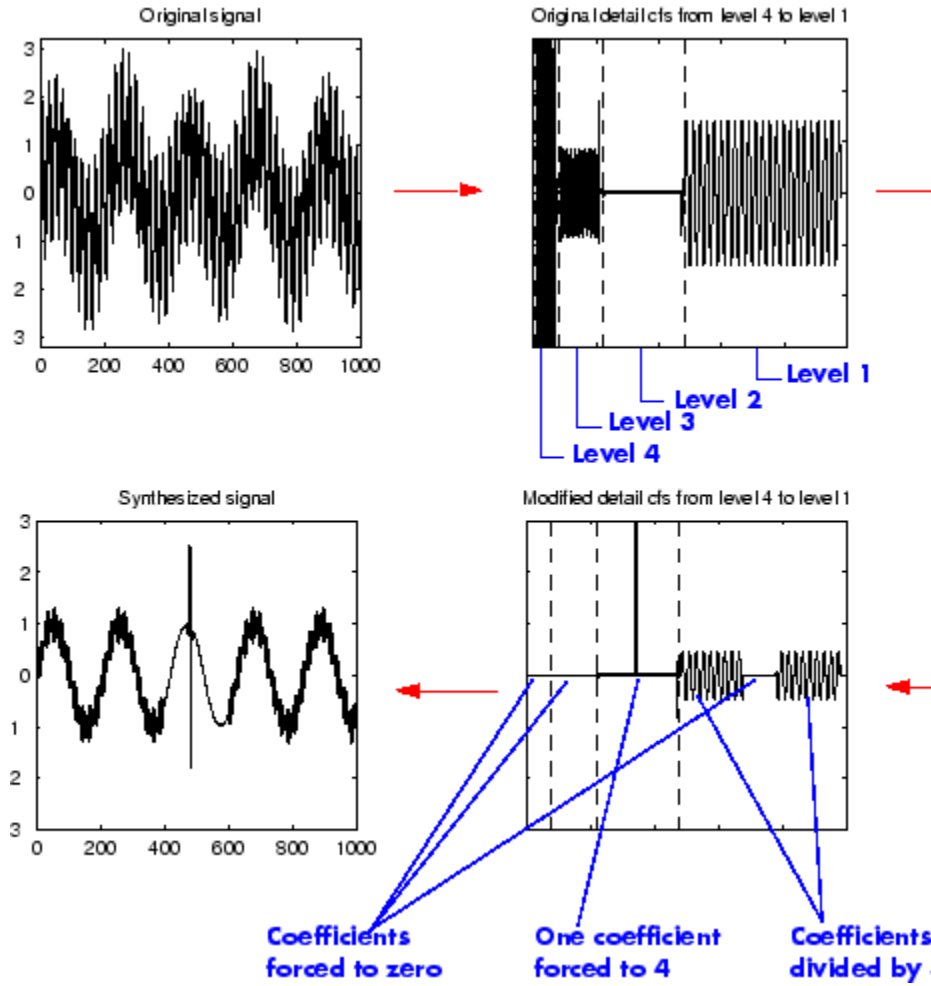
Likewise, let us suppose that, on [α,β] to which 0 belongs, we have the expansion $s(x) = [s(0) + xs'(0) = x^2 s^{(2)}(0) + \ldots + x^k s^{(k)}(0)] + g(x)$. The s and g signals then have the same wavelet coefficients. This is the technical meaning of the phrase "The wavelet suppresses a polynomial part 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 details D_1 through D_4 in "Example 2: A Frequency Breakdown (Discontinuity)" on page 2-10 (see the curves **d1**, **d2**, **d3**, and **d4**). The wavelet suppresses the slow sine wave, which is locally assimilated to a polynomial.

Another way of suppressing a component of the signal is to modify and force 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 file, some features of wavelet processing using coefficients (resulting plots can be found in Suppress or Modify Signal Components, Acting on Coefficients on page 6-99).

```
% 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);
```



Suppress or Modify Signal Components, Acting on Coefficients

Simple procedures to select the set of indices E are used for de-noising and compression purposes (see “De-Noising” on page 6-101 and “Data Compression” on page 6-115).

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 (the small values of *j*), while the low frequencies influence the deepest levels (the large values of *j*) and the associated approximations.

If a signal comprising only white noise is analyzed (for example, see "Example 3: Uniform White Noise" on page 2-15), 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, and a 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, then 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. Furthermore, the wavelet (if indeed it exists) most probably depends on the color of the signal. 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 $C(j,k),k\epsilon Z$ 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 $C(j,k)$, $k \epsilon Z$ 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:

$$U(a,b) = \int_R \frac{1}{\sqrt{a}} \psi\left(\frac{x-b}{a}\right) \psi(x) dx$$

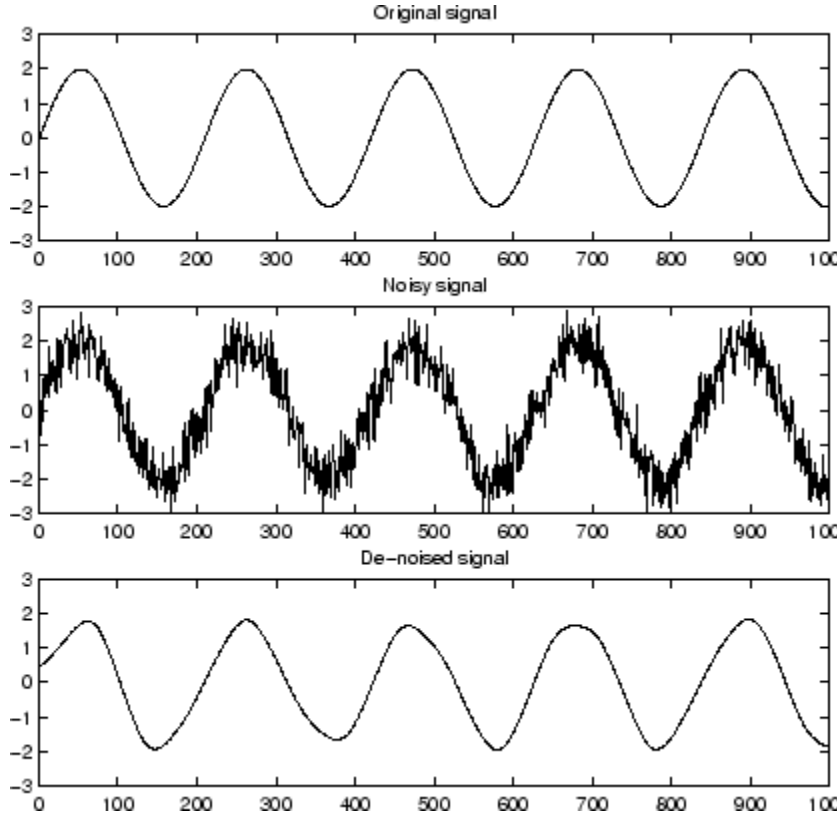
From the results for coefficients we deduce the properties of the details (and of the approximations), by using the formula

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

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.



Simple De-Noising Example

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 σ 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 method is efficient for families of functions *f* that have only a few nonzero wavelet coefficients. These functions have a sparse wavelet representation. For example, a smooth function almost everywhere, with only a few abrupt changes, has such a property.

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 general de-noising procedure involves three steps. The basic version of the procedure follows these 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 using 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

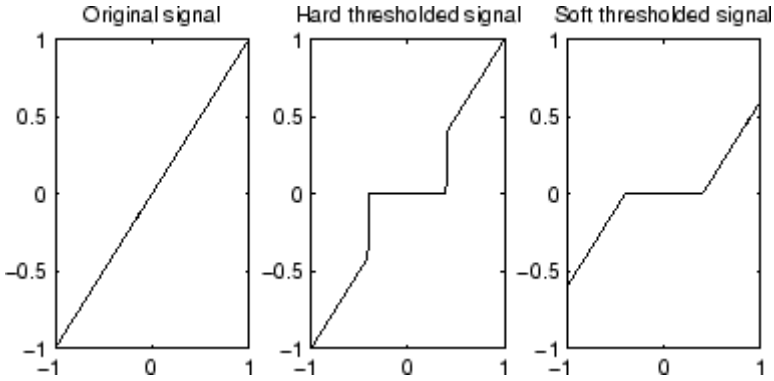
```
yt = wthresh(y,sorh,thr)
```

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 (For instance, see [Don95] in "References" on page 6-168).

Let us give a simple example.

```
y = linspace(-1,1,100);
thr = 0.4;
ythard = wthresh(y,'h',thr);
ytsoft = wthresh(y,'s',thr);
```



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, and then shrinking the nonzero coefficients toward 0 (see Hard and Soft Thresholding of the Signal $s = x$ on page 6-104).

As can be seen in the comment of Hard and Soft Thresholding of the Signal $s = x$ on page 6-104), 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 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 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, over a given set of functions, of the maximum mean square error.

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 and returns the result $f(x) = 0$. 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 near 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 Nonwhite Noise

Usually in practice the basic model cannot be used directly. We examine here the options available to deal with model deviations in the main de-noising function `wden`.

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`, `scal`, `n`, and `wav`. The parameter `sorh` specifies the 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 nonwhite noise

- Option `scal = 'one'` corresponds to the basic model.
- In general, you can ignore the noise level and it 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, then these details are concentrated in a 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 as in the previous option is used by estimating σ_{lev} level by level.

This estimation is implemented in the 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.

For a more general procedure, the `wdencmp` function performs wavelet coefficients thresholding for both de-noising and compression purposes, while directly handling one-dimensional and two-dimensional data. It allows you to define your own thresholding strategy selecting in

```
xd = wdencmp(opt,x,wav,n,thr,sorh,keepapp);
```

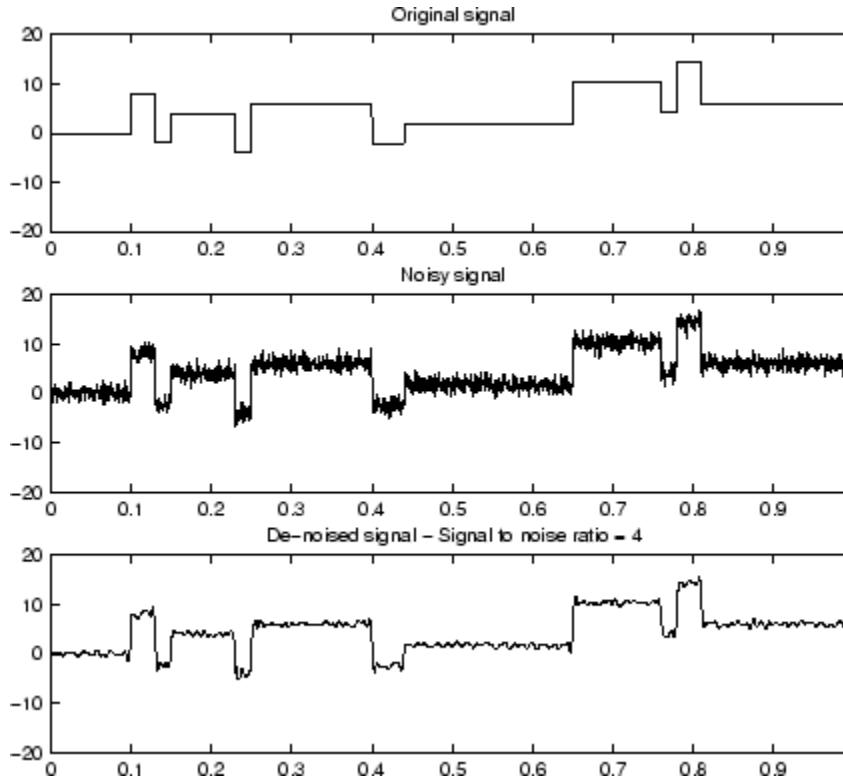
where

- `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. You can use the following file to get the first test function using `wnoise`.

```
% Set signal to noise ratio and set rand seed.
sqrt_snr = 4; init = 2055615866;

% Generate original signal xref and a noisy version x adding
% a standard Gaussian white noise.
[xref,x] = wnoise(1,11,sqrt_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');
```



Blocks Signal De-Noising

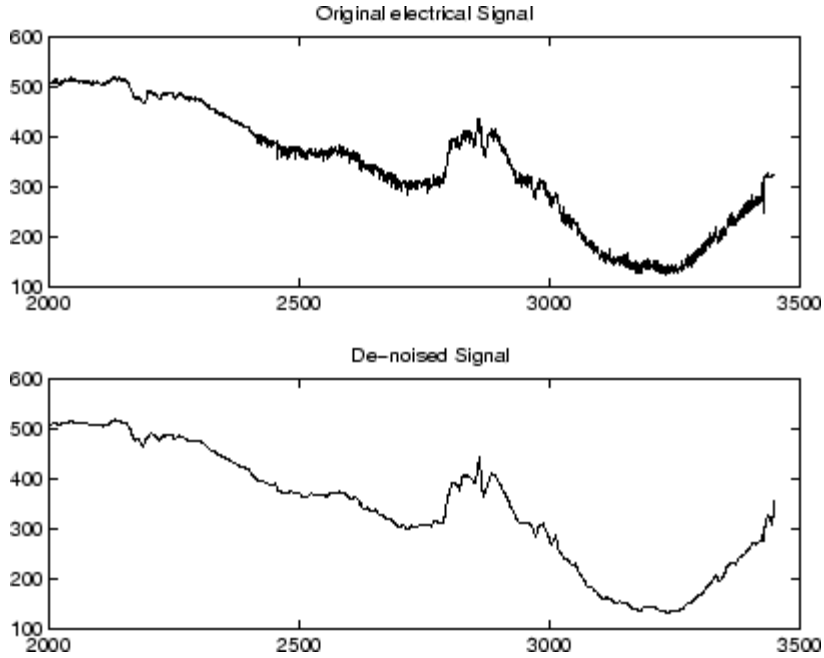
Since only a small number of large coefficients characterize the original signal, the method performs very well (see the figure Blocks Signal De-Noising on page 6-109). If you want to see more about how the thresholding works, use the GUI (see "De-Noising Signals" on page 1-18).

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.

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;
```



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

$$s(i,j) = f(i,j) + \sigma e(i,j)$$

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, `prod(size(s))` is used instead of `length(s)` 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 file illustrating the de-noising of a real image.

```
% Load original image.
load  woman

% Generate noisy image.
init = 2055615866; randn('seed',init);
x = X + 15*randn(size(X));

% Find default values. 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 equal to estimated_sigma*sqrt(log(prod(size(X))))
thr

thr =

  107.6428

% De-noise image using global thresholding option.
xd = wdencmp('gbl',x,'sym4',2,thr,sorh,keepapp);
```

```
% Plots.
colormap(pink(255)), sm = size(map,1);
subplot(221), image(wcodemat(X,sm)), title('Original Image')
subplot(222), image(wcodemat(x,sm)), title('Noisy Image')
subplot(223), image(wcodemat(xd,sm)), title('De-Noised Image')
```

The result shown below is acceptable.

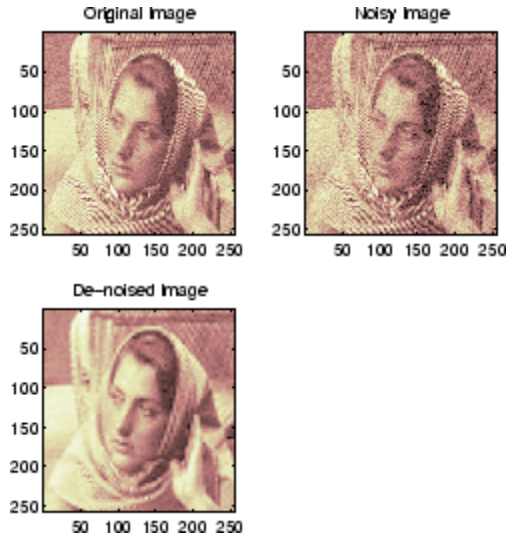


Image De-Noising

One-Dimensional Variance Adaptive Thresholding of Wavelet Coeffiients

Local thresholding of wavelet coefficients, for one- or two-dimensional data, is a capability available from a lot of graphical interface tools throughout Wavelet Toolbox software (see “Using Wavelets”) in the *Wavelet Toolbox Getting Started Guide*.

The idea is to define level by level time-dependent thresholds, and then increase the capability of the de-noising strategies to handle nonstationary variance noise models.

More precisely, the model assumes (as previously) that the observation is equal to the interesting signal superimposed on a noise (see "De-Noising" on page 6-101).

$$s(n) = f(n) + \sigma e(n)$$

But the noise variance can vary with time. There are several different variance values on several time intervals. The values as well as the intervals are unknown.

Let us focus on the problem of estimating the change points or equivalently the intervals. The algorithm used is based on an original work of Marc Lavielle about detection of change points using dynamic programming (see [Lav99] in "References" on page 6-168).

Let us generate a signal from a fixed-design regression model with two noise variance change points located at positions 200 and 600.

```
% Generate blocks test signal.
x = wnoise(1,10);

% Generate noisy blocks with change points.
init = 2055615866; randn('seed',init);
bb = randn(1,length(x));
cp1 = 200; cp2 = 600;
x = x + [bb(1:cp1),bb(cp1+1:cp2)/3,bb(cp2+1:end)];
```

The aim of this example is to recover the two change points from the signal `x`. In addition, this example illustrates how the GUI tools (see "Using Wavelets") locate the change points for interval dependent thresholding.

Step 1. Recover a noisy signal by suppressing an approximation.

```
% Perform a single-level wavelet decomposition
% of the signal using db3.
wname = 'db3'; lev = 1;
[c,l] = wavedec(x,lev,wname);

% Reconstruct detail at level 1.
det = wrcoef('d',c,l,wname,1);
```

The reconstructed detail at level 1 recovered at this stage is almost signal free. It captures the main features of the noise from a change points detection viewpoint if the interesting part of the signal has a sparse wavelet representation. To remove almost all the signal, we replace the biggest values by the mean.

Step 2. To remove almost all the signal, replace 2% of biggest values by the mean.

```
x = sort(abs(det));
v2p100 = x(fix(length(x)*0.98));
ind = find(abs(det)>v2p100);
det(ind) = mean(det);
```

Step 3. Use the `wvarchg` function to estimate the change points with the following parameters:

- The minimum delay between two change points is d = 10.
- The maximum number of change points is 5.

```
[cp_est,kopt,t_est] = wvarchg(det,5)
cp_est =
    199   601

kopt =
    2

t_est =
      1024         0         0         0         0         0
       601      1024         0         0         0         0
       199       601      1024         0         0         0
       199       261       601      1024         0         0
       207       235       261       601      1024         0
       207       235       261       393       601      1024
```

Two change points and three intervals are proposed. Since the three interval variances for the noise are very different the optimization program detects easily the correct structure.

The estimated change points are close to the true change points: 200 and 600.

Step 4. (Optional) Replace the estimated change points.

For $2 \le i \le 6$, `t_est(i,1:i-1)` contains the `i-1` instants of the variance change points, and since `kopt` is the proposed number of change points; then

```
cp_est = t_est(kopt+1,1:kopt);
```

You can replace the estimated change points by computing

```
% cp_New = t_est(knew+1,1:knew); % where 1   knew   5
```

More About De-Noising

The 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] page 173). This chapter and the corresponding files follow the work of the above mentioned researchers. More details can be found in Donoho's references in "References" on page 6-168 and in "More About the Thresholding Strategies" on page 6-132.

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

   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*, a threshold is selected and hard thresholding is applied to the detail coefficients.

3 Reconstruct

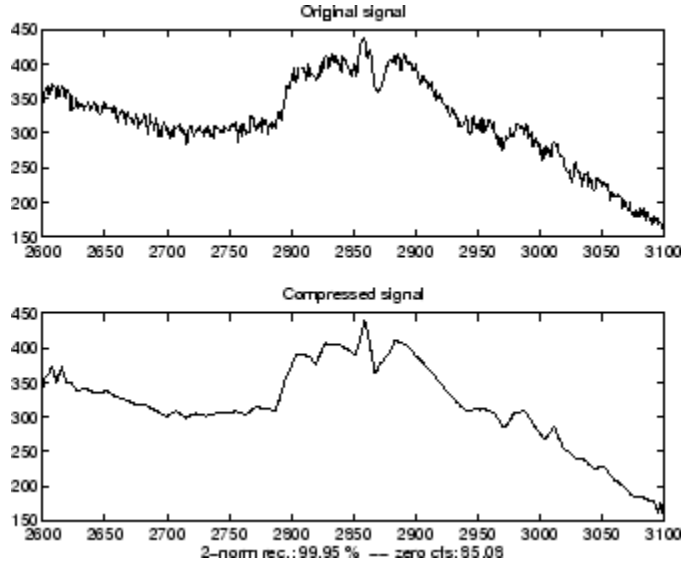
Compute wavelet reconstruction using the original approximation coefficients of level *N* and the modified detail coefficients of levels from 1 to *N*.

The difference of 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 Signal Compression on page 6-117) and for an image (see Image Compression on page 6-118).

```
% 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;
keepapp = 1;
[xd,cxd,lxd,perf0,perfl2] = ...
   wdencmp('gbl',c,l,w,n,thr,'h',keepapp);
```



Signal Compression

The result is quite satisfactory, not only because of the norm recovery criterion, but also on a visual perception point of view. The reconstruction uses only 15% of the coefficients.

```
% 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;
keepapp = 1;
[xd,cxd,lxd,perf0,perfl2] = ...
                 wdencmp('gbl',c,l,w,n,thr,'h',keepapp);
```

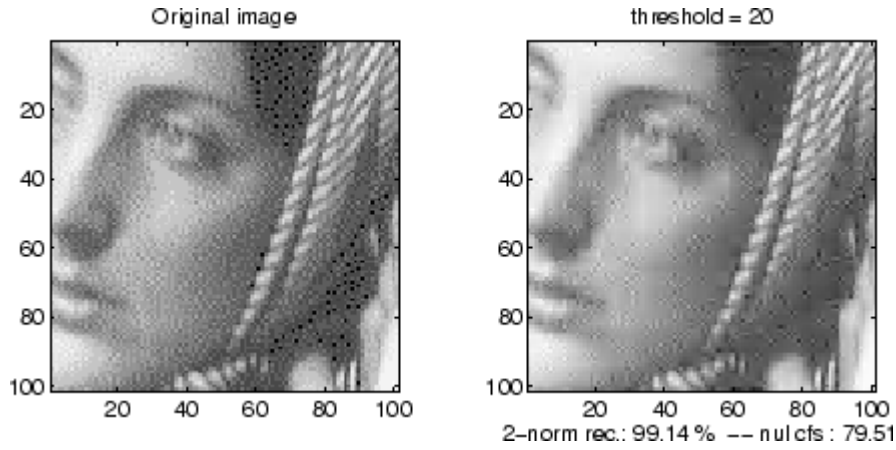


Image Compression

If the wavelet representation is too dense, similar strategies can be used in the wavelet packet framework 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).

Compression Scores

When compressing using orthogonal wavelets, the *Retained energy* in percentage is defined by

$$\frac{100*(\text{vector-norm}(\text{coeffs of the current decomposition},2))^2}{(\text{vector-norm}(\text{original signal},2))^2}$$

When compressing using biorthogonal wavelets, the previous definition is not convenient. We use instead the *Energy ratio* in percentage defined by

$$\frac{100*(\text{vector-norm}(\text{compressed signal},2))^2}{(\text{vector-norm}(\text{original signal},2))^2}$$

and as a tuning parameter the *Norm cfs recovery* defined by

$$\frac{100 * (\text{vector-norm(coeffs of the current decomposition}, 2))^2}{(\text{vector-norm(coeffs of the original decomposition}, 2))^2}$$

The *Number of zeros* in percentage is defined by

$$\frac{100 * (\text{number of zeros of the current decomposition})}{(\text{number of coefficients})}$$

Function Estimation: Density and Regression

In this section we present two problems of functional estimation:

- Density estimation
- Regression estimation

Note According to the classical statistical notations, in this section, $\hat{g}$ denotes the estimator of the function g instead of the Fourier transform of g.

Density Estimation

The data are values (`X(i), 1 ≤ i ≤ n`) sampled from a distribution whose density is unknown. We are looking for an estimate of this density.

What Is Density.

The well known histogram creates the information on the density distribution of a set of measures. At the very beginning of the 19th century, Laplace, a French scientist, repeating sets of observations of the same quantity, was able to fit a simple function to the density distribution of the measures. This function is called now the Laplace-Gauss distribution.

Density Applications.

Density estimation is a core part of reliability studies. It permits the evaluation of the life-time probability distribution of a TV set produced by a factory, the computation of the instantaneous availability, and of such other

useful characteristics as the mean time to failure. A very similar situation occurs in survival analysis, when studying the residual lifetime of a medical treatment.

Density Estimators.

As in the regression context, the wavelets are useful in a nonparametric context, when very little information is available concerning the shape of the unknown density, or when you don't want to tell the statistical estimator what you know about the shape.

Several alternative competitors exist. The orthogonal basis estimators are based on the same ideas as the wavelets. Other estimators rely on statistical window techniques such as kernel smoothing methods.

We have theorems proving that the wavelet-based estimators behave at least as well as the others, and sometimes better. When the density $h(x)$ has irregularities, such as a breakdown point or a breakdown point of the derivative $h'(x)$, the wavelet estimator is a good solution.

How to Perform Wavelet-Based Density Estimation.

The key idea is to reduce the density estimation problem to a fixed-design regression model. More precisely the main steps are as follows:

1. Transform the sample X into (Xb, Yb) data where the Xb are equally spaced, using a binning procedure. For each bin i, $Yb(i)$ = number of $X(j)$ within bin i.

2. Perform a wavelet decomposition of Yb viewed as a signal, using fast algorithm. Thus, the underlying Xb data is 1, 2, ..., nb where nb is the number of bins.

3. Threshold the wavelet coefficients according to one of the methods described for de-noising (see "De-Noising" on page 6-101).

4. Reconstruct an estimate $h1$ of the density function h from the thresholded wavelet coefficients using fast algorithm (see "Fast Wavelet Transform (FWT) Algorithm" on page 6-19).

5 Postprocess the resulting function *h1*. Rescale the resulting function transforming 1, 2, ..., *nb* into *Xb* and interpolate *h1* for each bin to calculate *hest*(*X*).

Steps **2** to **4** are standard wavelet-based steps. But the first step of this estimation scheme depends on *nb* (the number of bins), which can be viewed as a bandwidth parameter. In density estimation, *nb* is generally small with respect to the number of observations (equal to the length of *X*), since the binning step is a presmoother. A typical default value is *nb* = length(*X*) / 4.

For more information, you can refer for example to [AntP98], [HarKPT98], and [Ogd97] in "References" on page 6-168.

A More Technical Viewpoint.

Let us be a little more formal.

Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed random variables, with a common density function $h = h(x)$.

This density h is unknown and we want to estimate it. We have very little information on h.

For technical reasons we suppose that $\int h(x)^2 dx$ is finite. This allows us to express h in the wavelet basis.

We know that in the basis of functions φ and ψ with usual notations, J being an integer,

$$h(x) = \sum_k a_{J,k}\phi_{J,k} + \sum_{j=-\infty}^{J}\sum_k d_{j,k}\psi_{j,k} = A_J + \sum_{j=-\infty}^{J} D.$$

The estimator $\hat{h} = \hat{h}(x)$ will use some wavelet coefficients. The rationale for the estimator is the following.

To estimate h, it is sufficient to estimate the coordinates $a_{J,k}$ and the $d_{j,k}$.

We shall do it now.

We know the definition of the coefficients:

$$a_{J,k} = \int \phi_{J,k}(x) h(x) dx$$

and similarly

$$d_{j,k} = \int \psi_{j,k}(x) h(x) dx$$

The expression of the $a_{J,k}$ has a very funny interpretation. Because h is a density $\int \phi_{J,k}(x) h(x) dx$ is $E\left(\phi_{J,k}(X_i)\right)$, the mean value of the random variable $\phi_{J,k}(X_i)$.

Usually such an expectation is estimated very simply by the mean value:

$$\hat{a}_{J,k} = \frac{1}{n} \sum_{i=1}^{n} \phi_{J,k}(X_i)$$

Of course the same kind of formula holds true for the $d_{j,k}$:

$$\hat{d}_{j,k} = \frac{1}{n} \sum_{i=1}^{n} \psi_{j,k}(X_i)$$

With a finite set of n observations, it is possible to estimate only a finite set of coefficients, those belonging to the levels from $J–j_0$ up to J, and to some positions k.

Besides, several values of the $d_{j,k}$ are not significant and are to be set to 0.

The values $d_{j,k}$, lower than a threshold t, are set to 0 in a very similar manner as the de-noising process and for almost the same reasons.

Inserting these expressions into the definition of h, we get an estimator:

$$\hat{h} = \sum_k \hat{a}_{J,k}\phi_{J,k} + \sum_{j=J-j_0}^{J} \sum_k \hat{d}_{j,k} 1_{\{|\hat{d}_{j,k}|>t\}} \psi_{j,k}$$

This kind of estimator avoids the oscillations that would occur if all the detail coefficients would have been kept.

From the computational viewpoint, it is difficult to use a quick algorithm because the X_i values are not equally spaced.

Note that this problem can be overcome.

Let's introduce the normalized histogram $\hat{H}$ of the values of *X*, having *nb* classes, where the centers of the bins are collected in a vector *Xb*, the frequencies of *Xi* within the bins are collected in a vector *Yb* and then

$$\hat{H}(x) = \frac{Yb(r)}{n} \text{ on the } r\text{-th bin}$$

We can write, using $\hat{H}$,

$$\hat{d}_{j,k} = \frac{1}{n}\sum_{i=1}^{n} \psi_{j,k}(X_i) = \frac{1}{n}\sum_{r=1}^{nb} Yb(r)\psi_{j,k}(Xb(r)) = c\int \psi_{j,k}(x)\hat{H}(x)dx$$

where $\frac{1}{c}$ is the length of each bin.

The signs ≈ occur because we lose some information when using histogram instead of the values X_i and when approximating the integral.

The last = sign is very interesting. It means that $\hat{d}_{j,k}$ is, up to the constant *c*, the wavelet coefficient of the function $\hat{H}$ associated with the level *j* and the position *k*. The same result holds true for the $\hat{a}_{J,k}$.

So, the last = sign of the previous equation shows that the coefficients $\hat{d}_{j,k}$ appear also to be (up to an approximation) wavelet coefficients — those of

the decomposition of the sequence $\hat{H}$. If some of the coefficients at level J are known or computed, the Mallat algorithm computes the others quickly and simply.

And now we are able to finish computing $\hat{h}$ when the $\hat{d}_{j,k}$ and the $\hat{a}_{J,k}$ have been computed.

The trick is the transformation of irregularly spaced X values into equally spaced values by a process similar to the histogram computation, and that is called *binning*.

You can see the different steps of the procedure using the Density Estimation Graphical User Interface, by typing

```
wavemenu
```

and clicking the **Density Estimation 1-D** option.

Regression Estimation

What Is Regression.

The regression problem belongs to the family of the most common practical questions. The goal is to get a model of the relationship between one variable Y and one or more variables X. The model gives the part of the variability of Y taken in account or explained by the variation of X. A function f represents the central part of the knowledge. The remaining part is dedicated to the residuals, which are similar to a noise. The model is $Y = f(X) + e$.

Regression Models.

The simplest case is the linear regression $Y = aX + b + e$ where the function f is affine. A case a little more complicated occurs when the function belongs to a family of parametrized functions as $f(X) = cos(wX)$, the value of w being unknown. Statistics Toolbox™ software provides tools for the study of such models. When f is totally unknown, the problem of the nonlinear regression is said to be a nonparametric problem and can be solved either by using usual statistical window techniques or by wavelet based methods.

Regression Applications.

These regression questions occur in many domains. For example:

- Metallurgy, where you can try to explain the tensile strength by the carbon content
- Marketing, where the house price evolution is connected to an economical index
- Air-pollution studies, where you can explain the daily maximum of the ozone concentration by the daily maximum of the temperature

Two designs are distinguished: the fixed design and the stochastic design. The difference concerns the status of X.

Fixed-Design Regression.

When the X values are chosen by the designer using a predefined scheme, as the days of the week, the age of the product, or the degree of humidity, the design is a fixed design. Usually in this case, the resulting X values are equally spaced. When X represents time, the regression problem can be viewed as a de-noising problem.

Stochastic Design Regression.

When the X values result from a measurement process or are randomly chosen, the design is stochastic. The values are often not regularly spaced. This framework is more general since it includes the analysis of the relationship between a variable Y and a general variable X, as well as the analysis of the evolution of Y as a function of time X when X is randomized.

How to Perform Wavelet-Based Regression Estimation.

The key idea is to reduce a general problem of regression to a fixed-design regression model. More precisely the main steps are as follows:

1. Transform (X,Y) data into (Xb,Yb) data where the Xb are equally spaced, using a binning procedure. For each bin i,

$$(Yb(i)) = \frac{sum\{Y(j) \text{ such that } X(j) \text{ lies in bin } i\}}{number\{Y(j) \text{ such that } X(j) \text{ lies in bin } i\}},$$

with the convention $\frac{0}{0} = 0$.

2 Perform a wavelet decomposition of *Yb* viewed as a signal using fast algorithm. This last sentence means that the underlying *Xb* data is 1, 2, ..., *nb* where *nb* is the number of bins.

3 Threshold the wavelet coefficients according to one of the methods described for de-noising.

4 Reconstruct an estimate *f1* of the function *f* from the thresholded wavelet coefficients using fast algorithm.

5 Post-process the resulting function *f1*. Rescale the resulting function *f1* transforming 1, 2, ..., *nb* onto *Xb* and interpolate *f1* for each bin in order to calculate $fest(x)$.

Steps **2** to **4** are standard wavelet-based steps. But the first step of this estimation scheme depends on the number of bins, which can be viewed as a bandwidth parameter. Generally, the value of `nb` is not chosen too small with respect to the number of observations, since the binning step is a presmoother.

For more information, you can refer for example to [AntP98], [HarKPT98], and [Ogd97]. See “References” on page 6-168.

A More Technical Viewpoint.

The regression problem goes along the same lines as the density estimation. The main differences, of course, concern the model.

There is another difference with the density step: we have here two variables *X* and *Y* instead of one in the density scheme.

The regression model is $Y_i = f(X_i) + \in_i$ where $(\in_i)_{1 \le i \le n}$ is a sequence of independent and identically distributed (i.i.d.) random variables and where the (X_i) are randomly generated according to an unknown density *h*.

Also, let us assume that (X_1,Y_1), ..., (X_n,Y_n) is a sequence of i.i.d. random variables.

The function f is unknown and we look for an estimator $\hat{f}$.

We introduce the function $g = f \cdot h$. So $f = \frac{g}{h}$ with the convention $0 = \frac{0}{0}$.

We could estimate g by a certain $\hat{g}$ and, from the density part, an $\hat{h}$, and

then use $\hat{f} = \frac{\hat{g}}{\hat{h}}$. We choose to use the estimate of h given by the histogram suitably normalized.

Let us bin the X-values into nb bins. The l-th bin-center is called $Xb(l)$, the number of X-values belonging to this bin is $n(l)$. Then, we define $Yb(l)$ by the sum of the Y-values within the bin divided by $n(l)$.

Let's turn to the f estimator. We shall apply the technique used for the density function. The coefficients of f, are estimated by

$$\hat{d}_{j,k} = \frac{1}{n}\sum_{i=1}^{n} Y_i \psi_{j,k}(X_i)$$

$$\hat{a}_{J,k} = \frac{1}{n}\sum_{i=1}^{n} Y_i \phi_{J,k}(X_i)$$

We get approximations of the coefficients by the following formula that can be written in a form proving that the approximated coefficients are also the wavelet decomposition coefficients of the sequence Yb:

$$\hat{d}_{j,k} = \frac{1}{n}\sum_{l=1}^{nb} Yb(l) \psi_{j,k}(Xb(l))$$

$$\hat{a}_{J,k} = \frac{1}{n}\sum_{l=1}^{nb} Yb(l) \phi_{J,k}(Xb(l))$$

The usual simple algorithms can be used.

You can see the different steps of the procedure using the Regression Estimation Graphical User Interface by typing `wavemenu`, and clicking the **Regression Estimation 1-D** option.

Available Methods for De-Noising, Estimation, and Compression Using GUI Tools

This section presents the predefined strategies available using the de-noising, estimation, and compression GUI tools.

One-Dimensional DWT and SWT De-Noising

Level-dependent or interval-dependent thresholding methods are available. Predefined thresholding strategies:

- Hard or soft (default) thresholding
- Scaled white noise, unscaled white noise (default) or nonwhite noise
- Thresholds values are
  - Donoho-Johnstone methods: Fixed-form (default), Heursure, Rigsure, Minimax
  - Birgé-Massart method: Penalized high, Penalized medium, Penalized low

    The last three choices include a sparsity parameter a $(a > 1)$.

    Using this strategy the defaults are $a = 6.25$, 2, and 1.5, respectively, and the thresholding mode is hard. Only scaled and unscaled white noise options are supported.

One-Dimensional DWT Compression

1. Level-dependent or interval-dependent hard thresholding methods are available. Predefined thresholding strategies are:
   - Birgé-Massart method: Scarce high (default), Scarce medium, Scarce low

This method includes a sparsity parameter a $(1 < a < 5)$. Using this strategy the default is $a = 1.5$.

- Empirical methods

  - Equal balance sparsity-norm

  - Remove near 0

2 Global hard thresholding methods with GUI-driven choice are available. Predefined thresholding strategies are:

- Empirical methods

  - Balance sparsity-norm (default = equal)

  - Remove near 0

Two-Dimensional DWT and SWT De-Noising

Level-dependent and orientation-dependent (horizontal, vertical, and diagonal) thresholding methods are available. Predefined thresholding strategies are:

- Hard or soft (default) thresholding
- Scaled white noise, unscaled white noise (default) or nonwhite noise
- Thresholds values are:
  - Donoho-Johnstone method: Fixed form (default)
  - Birgé-Massart method: Penalized high, Penalized medium, Penalized low

    The last three choices include a sparsity parameter a ($a > 1$). See "One-Dimensional DWT and SWT De-Noising" on page 6-128.
  - Empirical method: Balance sparsity-norm, default = sqrt

Two-Dimensional DWT Compression

Level-dependent and orientation-dependent (horizontal, vertical, and diagonal) thresholding methods are available.

1 Level-dependent or interval-dependent hard thresholding methods are available. Predefined thresholding strategies are:

- Birgé-Massart method: Scarce high (default); Scarce medium, Scarce low

  This method includes a sparsity parameter a ($1 < a < 5$), the default is $a = 1.5$.

- Empirical methods

  - Equal balance sparsity-norm

  - Square root of the threshold associated with Equal balance sparsity-norm

  - Remove near 0

2 Global hard thresholding methods with GUI-driven choice are available. Predefined thresholding strategies are:

- Empirical methods

  - Balance sparsity-norm (default = equal); Balance sparsity-norm (sqrt)

  - Remove near 0

One-Dimensional Wavelet Packet De-Noising

Global thresholding methods with GUI-driven choice are available. Predefined thresholding strategies are:

- Hard or soft (default) thresholding
- Thresholds values:
  - Donoho-Johnstone methods: Fixed form (unscaled noise) (default); Fixed form (scaled noise)
  - Birgé-Massart method: Penalized high, Penalized medium, Penalized low

    This method includes a sparsity parameter a ($a > 1$). See "One-Dimensional DWT and SWT De-Noising" on page 6-128.

One-Dimensional Wavelet Packet Compression

Global hard thresholding methods with GUI-driven choice are available. Predefined thresholding strategies are:

- Empirical methods

- Balance sparsity-norm (default = equal)
- Remove near 0

Two-Dimensional Wavelet Packet De-Noising

Global thresholding methods with GUI-driven choice are available. Predefined thresholding strategies are:

- Hard or soft (default) thresholding
- Thresholds values:
  - Donoho-Johnstone methods: Fixed form (unscaled noise) (default); Fixed form (scaled noise)
  - Birgé-Massart method: Penalized high, Penalized medium, Penalized low

    The last three choices include a sparsity parameter a ($a > 1$). See "One-Dimensional DWT and SWT De-Noising" on page 6-128.
  - Empirical method: Balance sparsity-norm (sqrt)

Two-Dimensional Wavelet Packet Compression

Global thresholding methods with GUI-driven choice are available. Predefined thresholding strategies are:

- Empirical methods
  - Balance sparsity-norm (default = equal), Balance sparsity-norm (sqrt)
  - Remove near 0

One-Dimensional Regression Estimation

A preliminary histogram estimator (binning) is used, and then the predefined thresholding strategies described in "One-Dimensional DWT and SWT De-Noising" on page 6-128 are available.

Density Estimation

A preliminary histogram estimator (binning) is used, and then the predefined thresholding strategies are as follows:

- Global threshold
- By level threshold 1, By level threshold 2, By level threshold 3

The last choice includes a sparsity parameter a ($a < 1$); the default is 0.6.

More About the Thresholding Strategies

A lot of references are available for this topic of de-noising, estimation, and compression.

For example, [Ant94], [AntP98], [HalPKP97], [AntG99], [Ogd97], [HarKPT98], [DonJ94a&b], [DonJKP95], and [DonJKP96] (see "References" on page 6-168). A short description of the available methods previously mentioned follows.

Scarce High, Medium, and Low.

These strategies are based on an approximation result from Birgé and Massart (for more information, see [BirM97]) and are well suited for compression.

Three parameters characterize the strategy:

- J, the level of the decomposition
- M, a positive constant
- a, a sparsity parameter ($a > 1$)

The strategy is such that:

- At level J the approximation is kept
- For level j from 1 to J, the n_j largest coefficients are kept with

$$n_j = \frac{M}{(J + 2 - j)^a}$$

So the strategy leads to select the highest coefficients in absolute value at each level, the numbers of kept coefficients grow scarcely with $J - j$.

Typically, $a = 1.5$ for compression and $a = 3$ for de-noising.

A natural default value for M is the length of the coarsest approximation coefficients, since the previous formula for $j = J + 1$, leads to $M = n_{J+1}$.

Let L denote the length of the coarsest approximation coefficients in the 1-D case and S the size of the coarsest approximation coefficients in the 2-D case.

Three different choices for M are proposed:

- Scarce high:
  - $M = L$ in the 1-D case
  - M = 4*prod(S) in the 2-D case
- Scarce medium:
  - $M = 1.5*L$ in the 1-D case
  - M = 4*4*prod(S) / 3 in the 2-D case
- Scarce low:
  - $M = 2*L$ in the 1-D case
  - M = 4*8*prod(S) / 3 in the 2-D case

The related files are `wdcbm`, `wdcbm2`, and `wthrmngr` (for more information, see the corresponding reference pages).

Penalized High, Medium, and Low.

These strategies are based on a recent de-noising result by Birgé and Massart, and can be viewed as a variant of the fixed form strategy (see “De-Noising” on page 6-101) of the wavelet shrinkage.

The threshold T applied to the detail coefficients for the wavelet case or the wavelet packet coefficients for a given fixed WP tree, is defined by

$$T = |c(t^*)|$$

with

$$t^* = \arg\min\left[-sum\{c^2(k), k < t\} + 2vt\left(a + \log\left(\frac{n}{t}\right)\right); t = 1, \ldots, n\right]$$

where

- The sparsity parameter $a > 1$
- The coefficients $c(k)$ are sorted in decreasing order of their absolute value
- v is the noise variance

Three different intervals of choices for the sparsity parameter a are proposed:

- Penalized high, $2.5 \le a < 10$
- Penalized medium, $1.5 < a < 2.5$
- Penalized low, $1 < a < 2$

The related files are `wbmpen`, `wpbmpen`, and `wthrmngr` (for more information, see the corresponding reference pages).

Remove Near 0.

Let c denote the detail coefficients at level 1 obtained from the decomposition of the signal or the image to be compressed, using `db1`. The threshold value is set to `median(abs(c))` or to `0.05*max(abs(c))` if `median(abs(c)) = 0`.

The related files are `ddencmp` and `wthrmngr` (for more information, see the corresponding reference pages).

Balance Sparsity-Norm.

Let c denote all the detail coefficients; two curves are built associating, for each possible threshold value t, two percentages:

- The 2-norm recovery in percentage
- The relative sparsity in percentage, obtained from the compressed signal by setting to 0 the coefficients less than t in absolute value

A default is provided for the 1-D case taking t such that the two percentages are equal. Another one is obtained for the 2-D case by taking the square root of the previous t.

The related file is `wthrmngr` (for more information, see the corresponding reference page).

Fixed Form.

This thresholding strategy comes from Donoho-Johnstone (see "References" on page 6-168 and the `'sqtwolog'` option of the `wden` function in "De-Noising" on page 6-101). The universal threshold is of the following form:

- DWT or SWT 1-D, $t = s\sqrt{2\log(n)}$ where n is the signal length and s is the noise standard deviation
- DWT or SWT 2-D, $t = s\sqrt{2\log(nm)}$ where $[n,m]$ is the image size
- WP 1-D, $t = s\sqrt{2\log(n\log(n)/(\log(2)))}$
- WP 2-D, $t = s\sqrt{2\log(nm\log(nm)/(\log(2)))}$

The related files are `ddencmp`, `thselect`, `wden`, `wdencmp`, and `wthrmngr` (for more information, see the corresponding reference pages).

Heursure, Rigsure, and Minimax. These methods are available for 1-D de-noising tools and come from Donoho-Johnstone (see "References" on page 6-168).

The related files are `thselect`, `wden`, `wdencmp`, and `wthrmngr` (for more information, see the corresponding reference pages).

Global, and By Level 1, 2, 3. These options are dedicated to the density estimation problem.

See [HalPKP97], [AntG99], [Ogd97], and [HarKPT98] in "References" on page 6-168 for more details.

Note that:

- c is all the detail coefficients of the binned data.
- $d(j)$ is the detail coefficients at level j.
- n is the number of bins chosen for the preliminary estimator (binning).

Then, these options are defined as follows:

1 *Global*:

Threshold value is set to

$$max(|c|) \times \frac{\log(n)}{\sqrt{n}}.$$

2 *By level 1*:

Level dependent thresholds $T(j)$ are defined by $0.4 \times max\left(|d(j)|\right)$.

3 *By level 2*:

Level dependent thresholds $T(j)$ are defined by $0.8 \times max\left(|d(j)|\right)$.

4 *By level 3*:

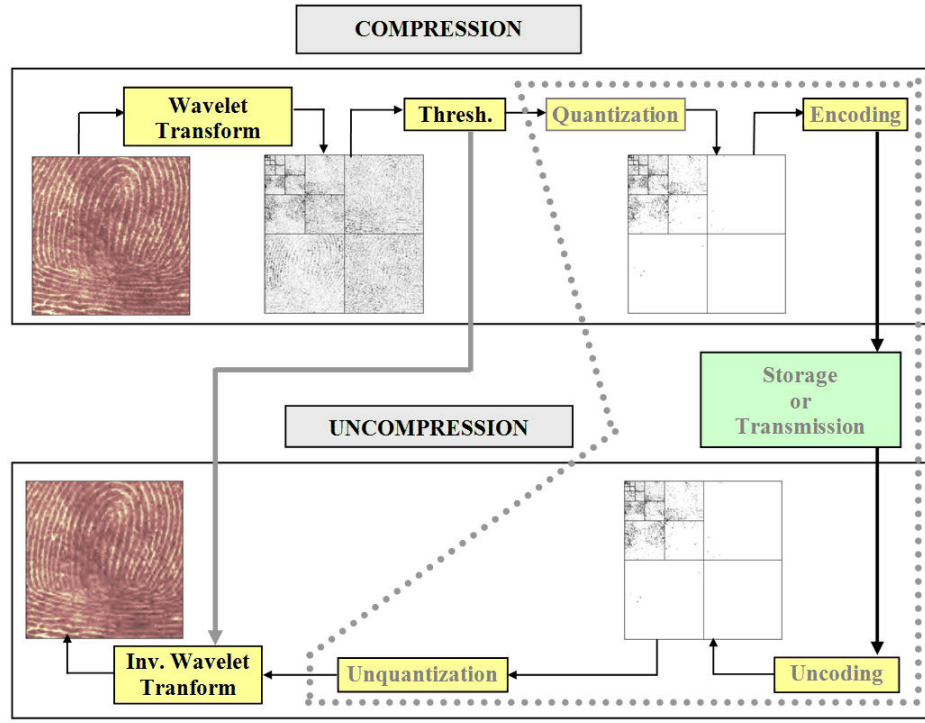
Level dependent thresholds $T(j)$ are defined by $a \times max\left(|d(j)|\right)$,

where a is a sparsity parameter ($0.2 < a \le 1, a = 0.6$ is the default).

True Compression for Images

In "Data Compression" on page 6-115, we addressed the aspects specifically related to compression using wavelets. However, in addition to the algorithms related to wavelets like DWT and IDWT, it is necessary to use other ingredients concerning the quantization mode and the coding type in order to deal with true compression.

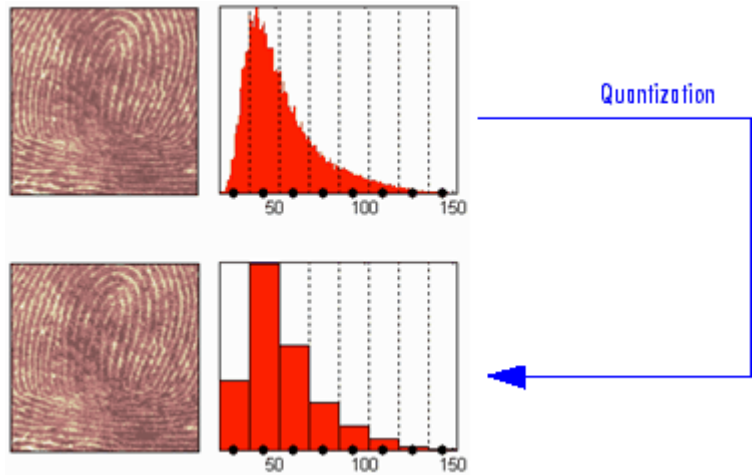
This more complex process can be represented by the following figure.



Effects of Quantization

Let us show the effects of quantization on the visualization of the fingerprint image. This indexed image corresponds to a matrix of integers ranging between 0 and 255. Through quantization we can decrease the number of colors which is here equal to 256.

The next figure illustrates how to decrease from 256 to 16 colors by working on the *values* of the original image.



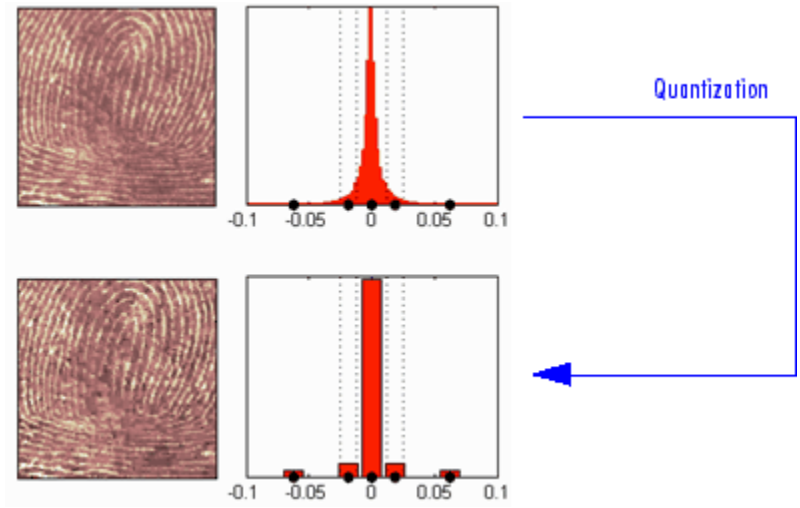
We can see on this figure:

- At the top
  - On the left: the original image
  - On the right: the corresponding histogram of *values*
- At the bottom
  - On the left: the reconstructed image
  - On the right: the corresponding histogram of *quantized values*

This quantization leads to a compression of the image. Indeed, with a fixed length binary code, 8 bits per pixel are needed to code 256 colors and 4 bits per pixel to code 16 colors. We notice that the image obtained after quantization is of good quality. However, within the framework of true compression, quantization is not used on the original image, but on its wavelet decomposition.

Let us decompose the fingerprint image at level 4 with the Haar wavelet. The histogram of wavelet coefficients and the quantized histogram are normalized so that the values vary between –1 and +1. The 15 intervals of quantization do not have the same length.

The next figure illustrates how to decrease information by binning on the wavelet *coefficient values* of the original image.



We can see on this figure:

- At the top
  - On left: the original image
  - On the right: the corresponding histogram (central part) of *coefficient values*
- At the bottom
  - On the left: the reconstructed image
  - On the right: the corresponding histogram (central part) of *quantized coefficient values*

The key point is that the histogram of the quantized coefficients is massively concentrated in the class centered in 0. Let us note that yet again the image obtained is of good quality.

True Compression Methods

The basic ideas presented above are used by three methods which cascade in a single step, coefficient thresholding (global or by level), and encoding by quantization. Fixed or Huffman coding can be used for the quantization depending on the method.

The following table summarizes these methods, often called Coefficients Thresholding Methods (CTM), and gives the MATLAB name used by the true compression tools for each of them.

MATLAB Name	Compression Method Name
`'gbl_mmc_f'`	Global thresholding of coefficients and fixed encoding
`'gbl_mmc_h'`	Global thresholding of coefficients and Huffman encoding
`'lvl_mmc'`	Subband thresholding of coefficients and Huffman encoding

More sophisticated methods are available which combine wavelet decomposition and quantization. This is the basic principle of progressive methods.

On one hand, progressivity makes it possible during decoding to obtain an image whose resolution increases gradually. In addition, it is possible to obtain a set of compression ratios based on the length of the preserved code. This compression usually involves a loss of information, but this kind of algorithm enables also lossless compression.

Such methods are based on three ideas. The two first, already mentioned, are the use of wavelet decomposition to ensure sparsity (a large number of zero coefficients) and classical encoding methods. The third idea, decisive for the use of wavelets in image compression, is to exploit fundamentally the tree structure of the wavelet decomposition. Certain codes developed from 1993 to 2000 use this idea, in particular, the EZW coding algorithm introduced by Shapiro. See [Sha93] in “References” on page 6-168.

EZW combines stepwise thresholding and progressive quantization, focusing on the more efficient way to encode the image coefficients, in order to minimize

the compression ratio. Two variants SPIHT and STW (see the following table) are refined versions of the seminal EZW algorithm.

Following a slightly different objective, WDR (and the refinement ASWDR) focuses on the fact that in general some portions of a given image require more refined coding leading to a better perceptual result even if there is generally a small price to pay in terms of compression ratio.

A complete review of these progressive methods is in the Walker reference [Wal99] in "References" on page 6-168.

The following table summarizes these methods, often called Progressive Coefficients Significance Methods (PCSM), and gives the MATLAB coded name used by the true compression tools for each of them.

MATLAB Name	Compression Method Name
`'ezw'`	Embedded Zerotree Wavelet
`'spiht'`	Set Partitioning In Hierarchical Trees
`'stw'`	Spatial-orientation Tree Wavelet
`'wdr'`	Wavelet Difference Reduction
`'aswdr'`	Adaptively Scanned Wavelet Difference Reduction
`'spiht_3d'`	Set Partitioning In Hierarchical Trees 3D for truecolor images

Quantitative and Perceptual Quality Measures

Let us close this section by defining two quantitative measures of the compression performance as well as two measures of the perceptual quality.

Compression Performance.

Two quantitative measures giving equivalent information are commonly used as a performance indicator for the compression:

- The compression ratio *CR,* which means that the compressed image is stored using only *CR*% of the initial storage size.

- The Bit-Per-Pixel ratio BPP, which gives the number of bits required to store one pixel of the image.

Perceptual Quality.

Two measures are commonly used to evaluate the perceptual quality:

- The Mean Square Error (*MSE*). It represents the mean squared error between the compressed and the original image and is given by:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m=1} \sum_{j=0}^{n=1} \left| X(i,j) - X_c(i,j) \right|^2$$

  The lower the value of *MSE*, the lower the error.

- The Peak Signal to Noise Ratio (*PSNR*). It represents a measure of the peak error and is expressed in decibels. It is defined by:

$$PSNR = 10 \cdot \log_{10} \left(\frac{255^2}{MSE} \right)$$

  The higher the *PSNR*, the better the quality of the compressed or reconstructed image. Typical values for lossy compression of an image are between 30 and 50 dB and when the *PSNR* is greater than 40 dB, then the two images are indistinguishable.

More Information on the True Compression

Various examples illustrating either the command-line mode or GUI tools for true compression using wavelets are in "Two-Dimensional True Compression" in the *Wavelet Toolbox Getting Started Guide*. More details on how to use the main command-line function are in the Reference document (see `wcompress`).

More information on the true compression for images and more precisely on the compression methods is in [Wal99], [Sha93], [Sai96], [StrN96], and **[Chr06]**. See "References" on page 6-168.

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, scale (as in wavelet decomposition), and frequency.

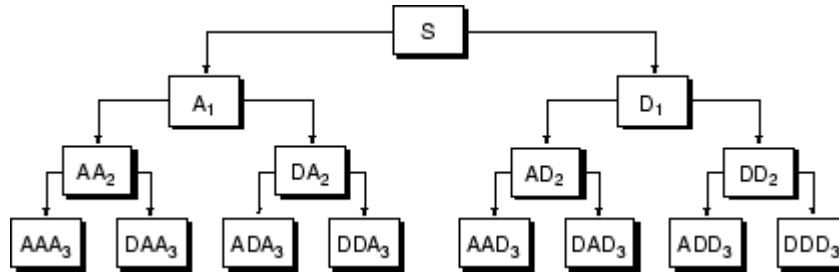
For a given orthogonal wavelet function, we generate a library of bases called *wavelet packet bases*. Each of these bases offers a particular way of coding signals, preserving global energy, and reconstructing exact features. The wavelet packets can be used for numerous expansions of a given signal. 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 the next step consists of splitting the new approximation coefficient vector; successive details are never reanalyzed.

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 the following figure.



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, called `sinper8`, is a 256-length sampled sine function of period 8. The Haar wavelet is used to decompose the signal at level 7.

The following figure 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).

Wavelet decomposition localizes the period of the sine within the interval [8,16]. Wavelet packets provide a more precise estimation of the actual period.

How to Obtain and Explain These Graphs.

You can reproduce these graphs by typing at the MATLAB prompt

```
wavemenu
```

Then click the **Wavelet Packet 1-D** option and select the **Example Analysis** using the `sinper8` demo signal. For more information on using this GUI tool, see "One-Dimensional Wavelet Packet Analysis" on page 3-7.

The length of the WP tree leaves is 2; there are 128 leaves, labeled from (7,0) to (7,127) and indexed from 127 to 254.

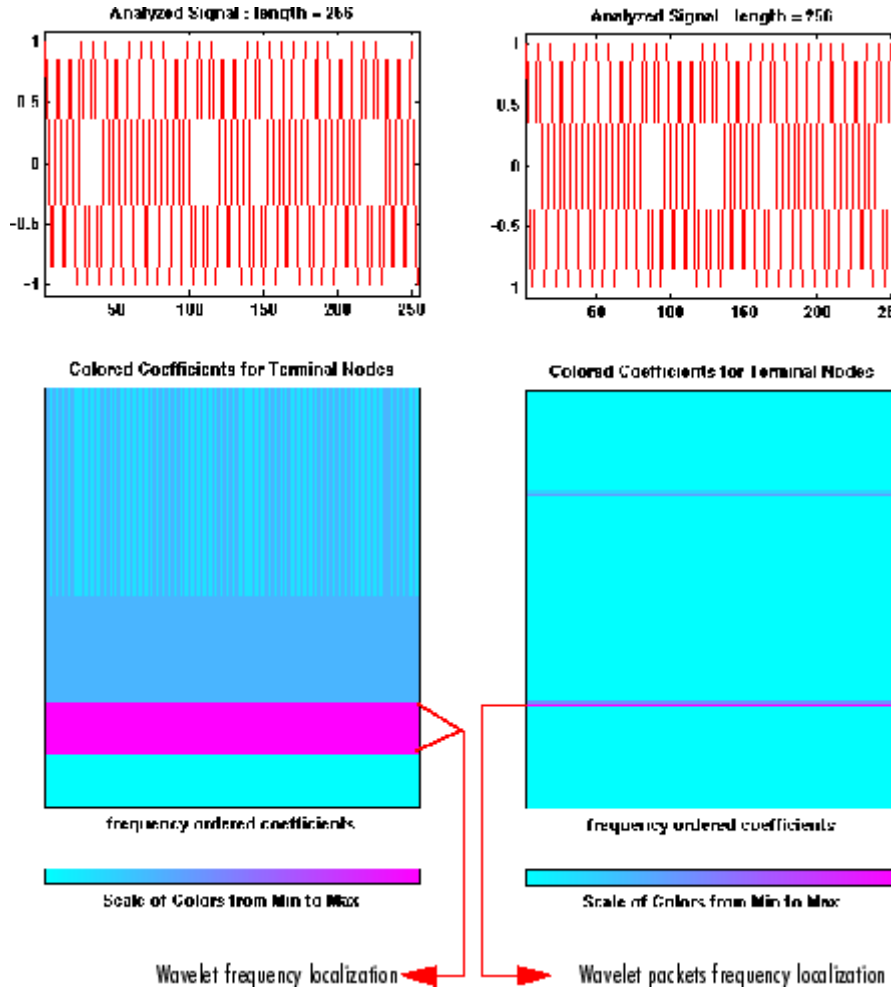
The associated wavelet tree (click the **Wavelet Tree** button) is obviously simpler than the wavelet packet tree. There are eight leaves labeled (7,0), (7,1), (6,1), . . . (2,1), (1,1).

The **Colored Coefficients for Terminal Nodes** graph deserves explanation. In principle the graphic displays eight stripes. When using **Global + abs**, only four seem to be present. In fact, the eight are drawn. As the values of several coefficients are close to 0, the stripes are merged and only four can be seen. The eight stripes are recovered when using the option **By level + abs**.

Getting back to the **Colored Coefficients for Terminal Nodes** graph of the initial tree, with cool colormap, two stripes are present. By zooming in, we determine their WP index or position:

- Stripe 1: index 175 or position (7,48) and index 143 or position (7,16)
- Stripe 2: index 207 or position (7,80) and index 239 or position (7,112)

Using the two sliders of the **Decomposition Tree** graphic, we can visualize the coefficients or the reconstructed signals corresponding to these four leaves.

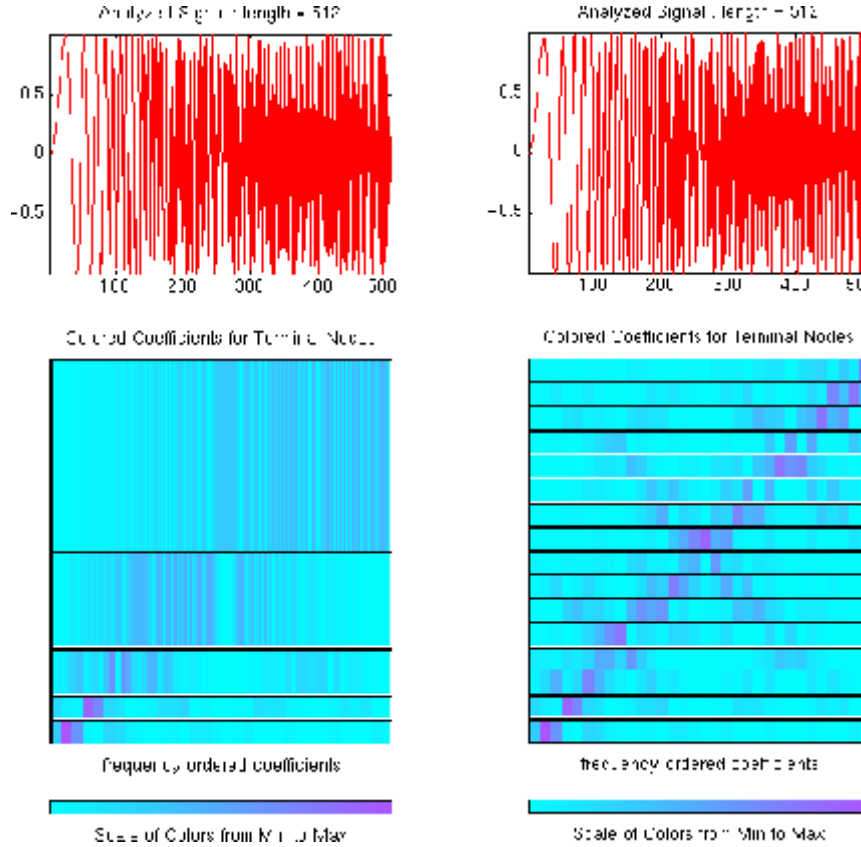


Wavelets (Left) Versus Wavelet Packets (Right): A Sine Function

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 Wavelets (Left) Versus Wavelet Packets (Right): Damped Oscillations on page 6-147, a wavelet analysis does not easily detect this time-frequency property of the signal. But

on the right of this figure, 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.



Wavelets (Left) Versus Wavelet Packets (Right): Damped Oscillations

Wavelet Packet Spectrum

The spectral analysis of wide-sense stationary signals using the Fourier transform is well-established. For nonstationary signals, there exist local Fourier methods such as the short-time Fourier transform (STFT). See "Short-Time Fourier Transform" for a brief description.

Because wavelets are localized in time and frequency, it is possible to use wavelet-based counterparts to the STFT for the time-frequency analysis of nonstationary signals. For example, it is possible to construct the scalogram (`wscalogram`) based on the continuous wavelet transform (CWT). However, a potential drawback of using the CWT is that it is computationally expensive.

The discrete wavelet transform (DWT) permits a time-frequency decomposition of the input signal, but the degree of frequency resolution in the DWT is typically considered too coarse for practical time-frequency analysis.

As a compromise between the DWT- and CWT-based techniques, wavelet packets provide a computationally-efficient alternative with sufficient frequency resolution. You can use `wpspectrum` to perform a time-frequency analysis of your signal using wavelet packets.

The following examples illustrate the use of wavelet packets to perform a local spectral analysis. The following examples also use `spectrogram` from the Signal Processing Toolbox software as a benchmark to compare against the wavelet packet spectrum. If you do not have the Signal Processing Toolbox software, you can simply run the wavelet packet spectrum examples.

Wavelet packet spectrum of a sine wave.

```
fs = 1000; % sampling rate
t = 0:1/fs:2; % 2 secs at 1kHz sample rate
y = sin(256*pi*t); % sine of period 128
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');
```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```
figure;
windowsize = 128;
window = hanning(windowsize);
nfft = windowsize;
noverlap = windowsize-1;
[S,F,T] = spectrogram(y,window,noverlap,nfft,fs);
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
```

```
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')
```

Sum of two sine waves with frequencies of 64 and 128 hertz.

```
fs = 1000;
t = 0:1/fs:2;
y = sin(128*pi*t) + sin(256*pi*t); % sine of periods 64 and 128.
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');
```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```
figure;
windowsize = 128;
window = hanning(windowsize);
nfft = windowsize;
noverlap = windowsize-1;
[S,F,T] = spectrogram(y,window,noverlap,nfft,fs);
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')
```

Signal with an abrupt change in frequency from 16 to 64 hertz at two seconds.

```
fs = 500;
t = 0:1/fs:4;
y = sin(32*pi*t).*(t<2) + sin(128*pi*t).*(t>=2);
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');
```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```
figure;
windowsize = 128;
window = hanning(windowsize);
nfft = windowsize;
noverlap = windowsize-1;
[S,F,T] = spectrogram(y,window,noverlap,nfft,fs);
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')
```

Wavelet packet spectrum of a linear chirp.

```
fs = 1000;
t = 0:1/fs:2;
y = sin(256*pi*t.^2);
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');
```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```
figure;
windowsize = 128;
window = hanning(windowsize);
nfft = windowsize;
noverlap = windowsize-1;
[S,F,T] = spectrogram(y,window,noverlap,nfft,fs);
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')
```

Wavelet packet spectrum of quadratic chirp.

```
y = wnoise('quadchirp',10);
len = length(y);
t = linspace(0,5,len);
fs = 1/t(2);
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');
```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```
windowsize = 128;
window = hanning(windowsize);
nfft = windowsize;
noverlap = windowsize-1;
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')
```

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 $2N$, where $h(n)$ and $g(n)$, corresponding to the wavelet.

Now by induction let us define the following sequence of functions:

$(W_n(x),\ n = 0,\ 1,\ 2,\ ...)$

by

$$W_{2n}(x) = \sqrt{2}\sum_{k=0}^{2N-1} h(k)W_n(2x-k)$$

$$W_{2n+1}(x) = \sqrt{2}\sum_{k=0}^{2N-1} g(k)W_n(2x-k)$$

where $W_0(x) = \varphi(x)$ is the scaling function and $W_1(x) = \psi(x)$ is the wavelet function.

For example for the Haar wavelet we have

$$N = 1, h(0) = h(1) = \frac{1}{\sqrt{2}}$$

and

$$g(0) = -g(1) = \frac{1}{\sqrt{2}}$$

The equations become

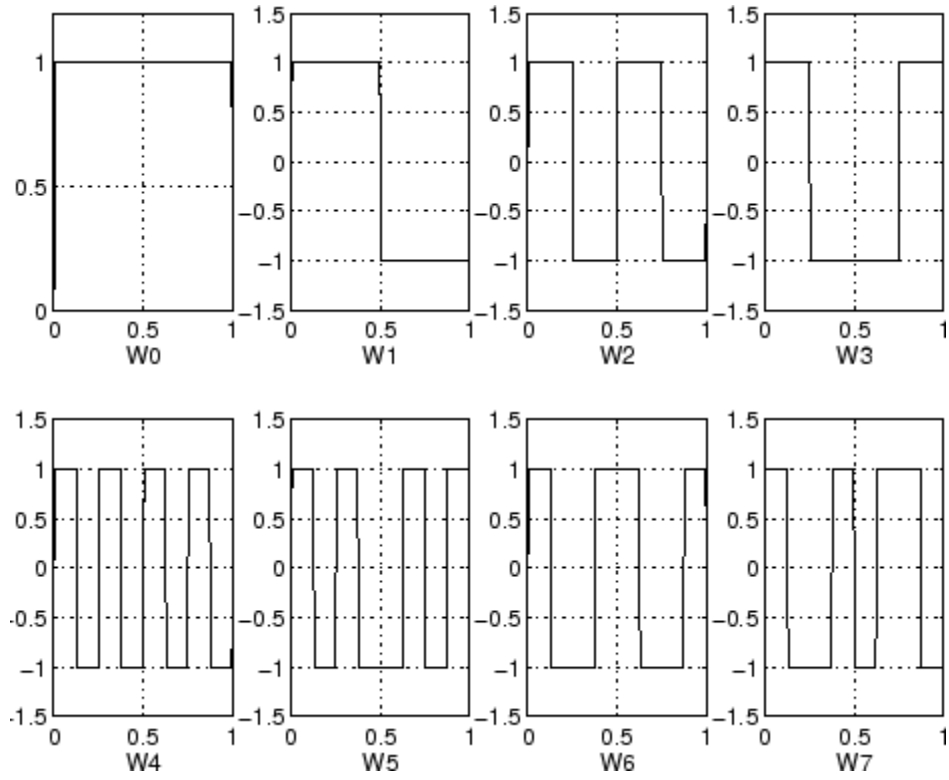
$$W_{2n}(x) = W_n(2x) + W_n(2x-1)$$

and

$$W_{2n+1}(x) = W_n(2x) - W_n(2x-1)$$

$W_0(x) = \varphi(x)$ 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 in the figure Haar Wavelet Packets on page 6-153.



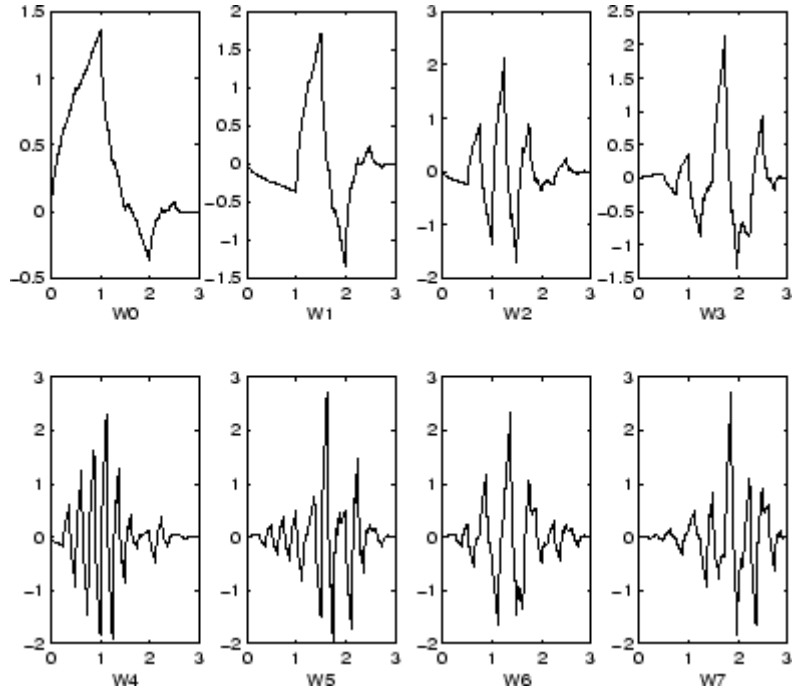
Haar Wavelet Packets

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]. The figure *db*2 Wavelet Packets on page 6-154 presents the system of *W*-functions for the original `db2` wavelet.



***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 $n \epsilon N$ and $(j,k) \epsilon Z^2$.

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?

The basic idea of the wavelet packets is that 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 Haar Wavelet Packets on page 6-153 and *db*2 Wavelet Packets on page 6-154, the naturally ordered W_n for $n = 0, 1, ..., 7$, does not match exactly the order defined by the number of oscillations. More precisely, counting the number of zero crossings (up-crossings and down-crossings) for the `db1` wavelet packets, we have the following.

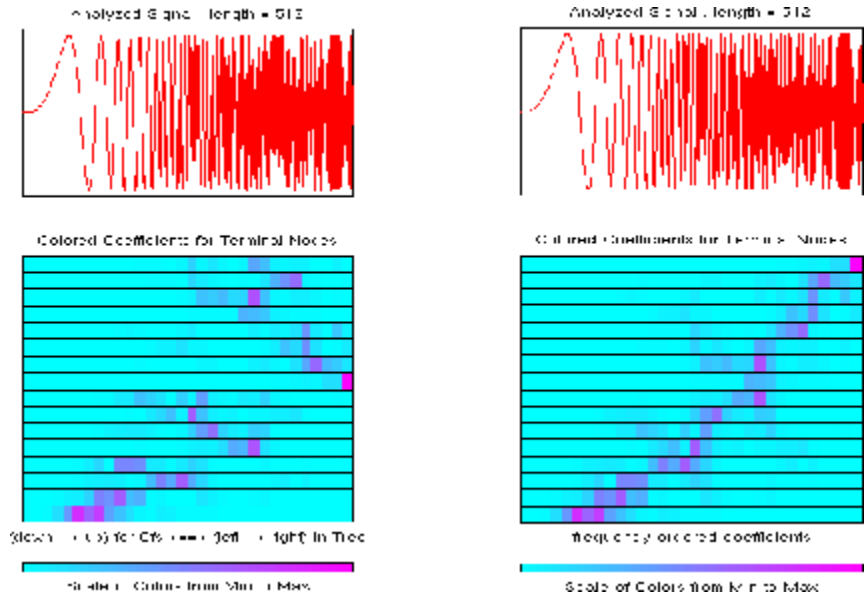
Natural order n	0	1	2	3	4	5	6	7
Number of zero crossings for $db1$ W_n	2	3	5	4	9	8	6	7

So, 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 r(n)	0	1	3	2	6	7	5	4

As can be seen in the previous figures, $W_{r(n)}(x)$ "oscillates" approximately n times.

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 Natural and Frequency Ordered Wavelet Packets Coefficients on page 6-156) from the low frequencies at the bottom to the high frequencies at the top, rather than naturally ordered coefficients (on the left of this same figure).



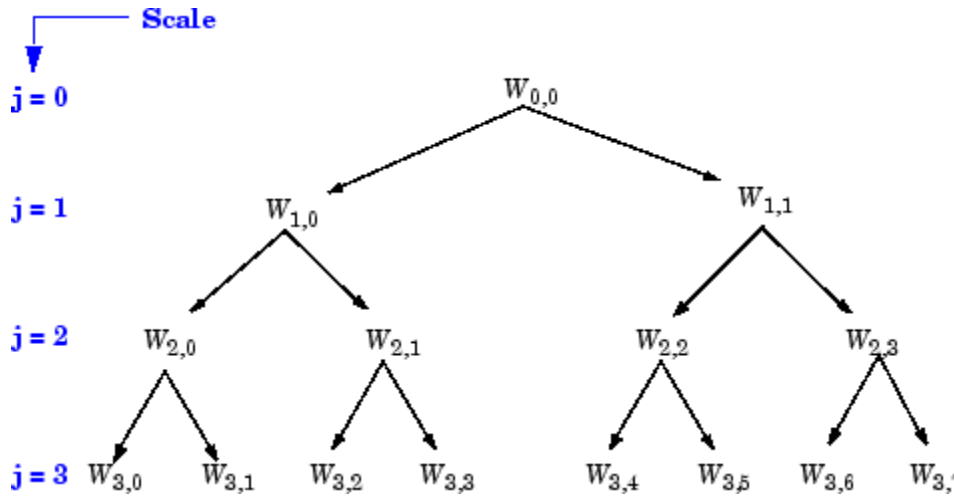
Natural and Frequency Ordered Wavelet Packets Coefficients

When plotting the coefficients, the various options related to the “Frequency” or “Natural” order choice are available using the GUI tools.

These options are also available from command-line mode when using the `wpviewcf` function.

Organizing the Wavelet Packets

The set of functions $W_{j,n} = (W_{j,n,k}(x), k \epsilon Z)$ is the (j,n) wavelet packet. For positive values of integers j and n, wavelet packets are organized in trees. The tree in the figure Wavelet Packets Organized in a Tree; Scale j Defines Depth and Frequency n Defines Position in the Tree on page 6-157 is created to give a maximum level decomposition equal to 3. For each scale j, the possible values of parameter n are 0, 1, ..., 2^j–1.



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 $W_{1,1} = (\psi(\frac{x}{2}-k), k \in Z)$.

It turns out that the library of wavelet packet bases contains the wavelet basis and also several other bases. Let us have a look at some of those bases. More precisely, let V_0 denote 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 the family of functions $\{(W_{j+1,2n}), (W_{j+1,2n+1})\}$ is an orthogonal basis of the space spanned by $W_{j,n}$, which is split into two subspaces: $W_{j+1,2n}$ spans the first subspace, and $W_{j+1,2n+1}$ the second one.

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 Wavelet Packet Tree: Split and Merge on page 6-158.

Wavelet Packet Tree: Split and Merge

It follows that the leaves of every connected binary subtree of the complete tree correspond to an orthogonal basis of the initial space.

For a finite energy signal belonging to V_0, 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.

A signal of length $N = 2^L$ can be expanded in α different ways, where α is the number of binary subtrees of a complete binary tree of depth L. As a result, $\alpha \geq 2^{N/2}$ (see [Mal98] page 323).

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.

Functions 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) are 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 (nonnormalized) 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 0log(0) = 0.
- The concentration in l^p norm with $1 \Re \leq p$

  $$E2(s_i) = |s_i|^p$$

  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 $E4(s)$ = # {i 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` file.

Example 1: Compute Various Entropies.

1 Generate a signal of energy equal to 1.

```
s = ones(1,16)*0.25;
```

2 Compute the Shannon entropy of *s*.

```
e1 = wentropy(s,'shannon')
    e1 = 2.7726
```

3 Compute the $l^{1.5}$ 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 the 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 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 (not w11 because the splitting of a null vector is without interest since the entropy is zero).

```
[w20,w21] = dwt(w10,'db1');
```

6 We have w20=0.5*ones(1,4) and w21 is zero. The entropy of the 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

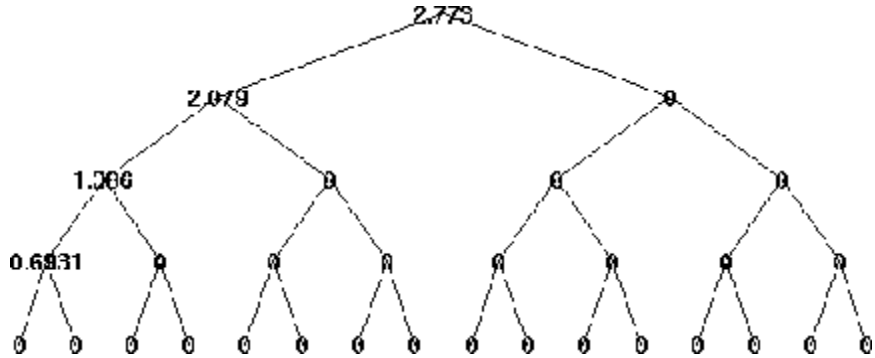
```
[w30,w31] = dwt(w20,'db1');
e30 = wentropy(w30,'shannon')
    e30 = 0.6931

[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 Perform wavelet packets decomposition of the signal *s* defined in example 1.

```
t = wpdec(s,4,'haar','shannon');
```

The wavelet packet tree in Entropy Values on page 6-162 shows the nodes labeled with original entropy numbers.

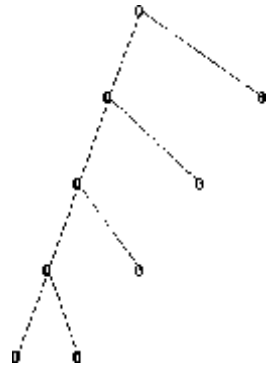


Entropy Values

9 Compute the best tree.

```
bt = besttree(t);
```

The best tree is shown in the following figure. In this case, the best tree corresponds to the wavelet tree. The nodes are labeled with optimal entropy.



Optimal Entropy Values

Some Interesting Subtrees

Using wavelet packets requires tree-related actions and labeling. The implementation of the user interface is built around this consideration. For more information on the technical details, see the reference pages.

The complete binary tree of depth D corresponding to a wavelet packet decomposition tree developed at level D is denoted by WPT.

We have the following interesting subtrees.

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, we use the following basic step 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 Labeling
$E(node) \leq \sum_{c \text{ child of node}} Eopt(c)$	If ($node \neq root$), merge and set $Eopt(node) = E(node)$
$E(node) > \sum_{c \text{ child of node}} Eopt(c)$	Split and set $Eopt(node) = \sum_{c \text{ child of node}} Eopt(c)$

with the natural initial condition on the reference tree, $Eopt(t) = E(t)$ for each terminal node t.

Reconstructing a Signal Approximation from a Node

You can use the function `wprcoef` to reconstruct an approximation to your signal from any node in the wavelet packet tree. This is true irrespective of whether you are working with a full wavelet packet tree, or a subtree determined by an optimality criterion. Use `wpcoef` if you want to extract the wavelet packet coefficients from a node without reconstructing an approximation to the signal.

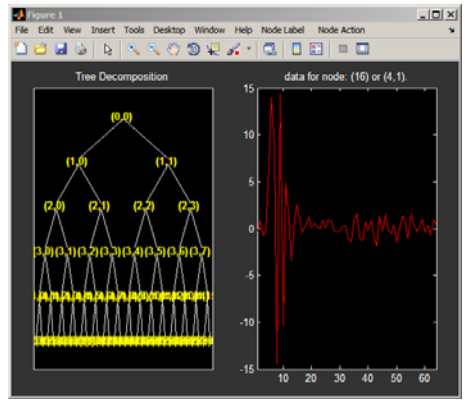
Load the noisy Doppler signal.

```
load noisdopp
```

Compute the wavelet packet decomposition down to level 5 using the `sym4` wavelet. Use the periodization mode.

```
dwtmode('per');
T = wpdec(noisdopp,5,'sym4');
```

Plot the binary wavelet packet tree and click on the (4,1) doublet (node 16).

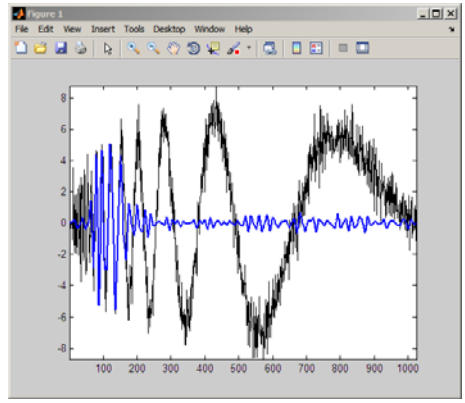


Extract the wavelet packet coefficients from node 16.

```
wpc = wpcoef(T,16);
% wpc is length 64
```

Obtain an approximation to the signal from node 16.

```
rwpc = wprcoef(T,16);
% rwpc is length 1024
plot(noisdopp,'k'); hold on;
plot(rwpc,'b','linewidth',2);
axis tight;
```

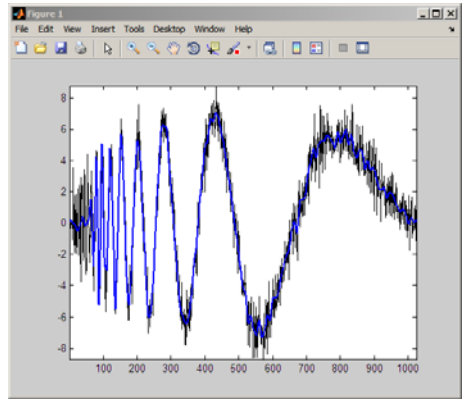


Determine the optimum binary wavelet packet tree.

```
Topt = besttree(T);
% plot the best tree
plot(Topt)
```

Reconstruct an approximation to the signal from the (3,0) doublet (node 7).

```
rsig = wprcoef(Topt,7);
% rsig is length 1024
plot(noisdopp,'k'); hold on;
plot(rsig,'b','linewidth',2);
axis tight;
```



If you know which doublet in the binary wavelet packet tree you want to extract, you can determine the node corresponding to that doublet with `depo2ind`.

For example, to determine the node corresponding to the doublet (3,0), enter:

```
Node = depo2ind(2,[3 0]);
```

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 the following figure for depth 2.

Quaternary Tree of Depth 2

Wavelet Packets for Compression and De-Noising

In the wavelet packet framework, compression and de-noising ideas are identical to those developed in the wavelet framework. The only new feature is a more complete 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. For more details, see Chapter 3, “Using Wavelet Packets”.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7

Adding Your Own Wavelets

This chapter discusses how to add your own wavelet families to the toolbox.

Preparing to Add a New Wavelet Family

Wavelet Toolbox software contains a lot of wavelet families, but by using the `wavemngr` function, you can add new wavelets to the existing ones to implement your favorite wavelet or try out one of your own design. The toolbox allows you to define new wavelets for use with both the command line functions and the graphical interface 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` entry in the Reference Guide.

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 MATLAB file (`file`).
6. *For wavelets without FIR filters*: Define the effective support.

These steps are described below.

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`, `Symlets`, `Coiflets`, `BiorSplines`,

`ReverseBior`, `Meyer`, `DMeyer`, `Gaussian`, `Mexican_hat`, `Morlet`, `Complex Gaussian`, `Shannon`, `Frequency B-Spline`, and `Complex Morlet`.

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`, `sym`, `coif`, `bior`, `rbio`, `meyr`, `dmey`, `gaus`, `mexh`, `morl`, `cgau`, `fbsp`, and `cmor`.

Determine the Wavelet Type

We distinguish five types of wavelets:

1. 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`.

2. 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.

3. 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.

4. 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`.

5. Complex 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 `Complex Gaussian` and `Shannon`.

Define the Orders of Wavelets Within the Given Family

If a family contains many wavelets, the short name and the order are appended to form the wavelet name. Argument `nums` is a string containing the orders separated with blanks. This argument is not used for wavelet families that only have a single wavelet (`Haar`, `Meyer`, and `Morlet` for example).

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 Code File

The `wavemngr` command requires a `file` argument, which is a string containing a MATLAB code file or MAT-file name.

If a family contains many wavelets, a MATLAB code file (with a `.m` extension) must be defined and must be of a specific form that depends on the wavelet type. The specific 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 code file can also be defined as discussed below.

Note If no file extension is specified, a `.m` extension is used as default.

Type 1 (Orthogonal with FIR Filter)

The syntax of the first line in the code 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 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 code 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 file `biorwavf.m` (for `BiorSplines`) is an example of a file for a type 2 predefined wavelet family.

Type 3 (Orthogonal with Scale Function)

The syntax of the first line in the code file must be

```
function [phi,psi,t] = file(lb,ub,n,wname)
```

which returns values of the scaling function `phi` and of the wavelet function `psi` on t, a regular `n`-point grid of the interval `[lb ub]`.

The argument `wname` is optional (see **Note** below).

The file `meyer.m` is an example of a file for a type 3 predefined wavelet family.

Type 4 or Type 5 (No FIR Filter; No Scale Function)

The syntax of the first line in the code file must be

```
function [psi,t] = file(lb,ub,n,wname)
```

or

```
function [psi,t] = file(lb,ub,n,wname, "additional arguments")
```

which returns values of the wavelet function `psi` on t, a regular `n`-point grid of the interval `[lb ub]`.

The argument `wname` is optional (see **Note** below).

Examples of type 4 files for predefined wavelet families are `mexihat.m` (for `Mexican_hat`) and `morlet.m` (for `Morlet`).

Examples of type 5 files for predefined wavelet families are `shanwavf.m` (for `Shannon`) and `cmorwavf.m` (for `Complex Morlet`).

Note For the types 3, 4, and 5, the `wname` argument can be optional. It is only required if the new wavelet family contains more than one wavelet and if you plan to use this new family in the GUI mode. For the types 4 and 5, a complete example of using the *"additional arguments"* can be found looking at the reference page for the `fbspwavf` function.

Define the Effective Support

This definition is required only for wavelets of types 3, 4, and 5, since they are not compactly supported.

Defining the effective support means specifying an upper and lower bound. For example, for some predefined wavelet families, we have the following.

Family	Lower Bound (`lb`)	Upper Bound (`ub`)
`Meyer`	–8	8
`Mexican_hat`	–5	5
`Morlet`	–4	4

Note For wavelets of type 3, 4, and 5, [–4 4] are the correct effective support theoretical values, but a wider effective support, [–8 8], is used in computation to provide more accurate results.

Adding 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 pages 216-217 of the book *Wavelets and Filter Banks* (see [StrN96] in "References" on page 6-168).

Note The files used in this example can be found in the `wavedemo` folder.

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 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
Symlets             sym
Coiflets            coif
BiorSplines         bior
ReverseBior         rbio
Meyer               meyr
DMeyer              dmey
Gaussian            gaus
Mexican_hat         mexh
Morlet              morl
Complex Gaussian    cgau
Shannon             shan
Frequency B-Spline  fbsp
Complex Morlet      cmor
Binlets             binl
==================================
```

If you want to get online information on this new family, you can build an associated help file which would look like the following:

```
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 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 ischar(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 files used in this example can be found in the `wavedemo` folder.

```
% List initial wavelets families.
    wavemngr('read')
ans =
==================================
Haar                haar
Daubechies          db
Symlets             sym
Coiflets            coif
BiorSplines         bior
ReverseBior         rbio
Meyer               meyr
DMeyer              dmey
Gaussian            gaus
Mexican_hat         mexh
Morlet              morl
Complex Gaussian    cgau
Shannon             shan
Frequency B-Spline  fbsp
Complex Morlet      cmor
==================================
% List all wavelets.
    wavemngr('read',1)

ans =

==================================
Haar                haar
```

```
==================================
Daubechies          db
------------------------------
db1  db2   db3  db4
db5  db6   db7  db8
db9  db10  db**
==================================
Symlets             sym
------------------------------
sym2  sym3  sym4  sym5
sym6  sym7  sym8  sym**
==================================
Coiflets            coif
------------------------------
coif1  coif2  coif3  coif4
coif5
==================================
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
==================================
ReverseBior         rbio
------------------------------
rbio1.1  rbio1.3  rbio1.5  rbio2.2
rbio2.4  rbio2.6  rbio2.8  rbio3.1
rbio3.3  rbio3.5  rbio3.7  rbio3.9
rbio4.4  rbio5.5  rbio6.8
==================================
Meyer               meyr
==================================
DMeyer              dmey
==================================
Gaussian            gaus
------------------------------
gaus1  gaus2  gaus3  gaus4
gaus5  gaus6  gaus7  gaus8
gaus**
```

```
==================================
Mexican_hat         mexh
==================================
Morlet              morl
==================================
Complex Gaussian    cgau
------------------------------
cgau1  cgau2  cgau3  cgau4
cgau5  cgau**
==================================
Shannon             shan
------------------------------
shan1-1.5  shan1-1  shan1-0.5  shan1-0.1
shan2-3  shan**
==================================
Frequency B-Spline  fbsp
------------------------------
fbsp1-1-1.5  fbsp1-1-1    fbsp1-1-0.5  fbsp2-1-1
fbsp2-1-0.5  fbsp2-1-0.1  fbsp**
==================================
Complex Morlet      cmor
------------------------------
cmor1-1.5  cmor1-1    cmor1-0.5  cmor1-1
cmor1-0.5  cmor1-0.1  cmor**
==================================
% 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.
```

```
%     The 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
Symlets             sym
Coiflets            coif
BiorSplines         bior
ReverseBior         rbio
Meyer               meyr
DMeyer              dmey
Gaussian            gaus
Mexican_hat         mexh
Morlet              morl
Complex Gaussian    cgau
Shannon             shan
Frequency B-Spline  fbsp
Complex Morlet      cmor
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 folder: 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 software, you should

1. Create a new folder specifically to hold the wavelet extension files.
2. Move the previously mentioned files into this new folder.
3. Prepend this folder to the MATLAB folder search path (see the reference entry for the `path` command).
4. Use this same folder for subsequent modifications. Allowing many wavelet extension files to proliferate in different folders may lead to unpredictable results.
5. Define a 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.

A

GUI Reference

This appendix explains some of the features of the Wavelet Toolbox graphical user interface (GUI).

General Features

Some features of the Wavelet Toolbox graphical user interface are

- Color coding
- Connectedness of plots
- Using the mouse
- Controlling the colormap
- Controlling the number of colors
- Controlling the coloration mode
- Customizing graphical objects
- Using menus
- Using **View Axes** button
- Using **Interval Dependent Threshold Settings** tool

Note In this appendix, *axis* (or *axes*) refers to the MATLAB graphic object.

Color Coding

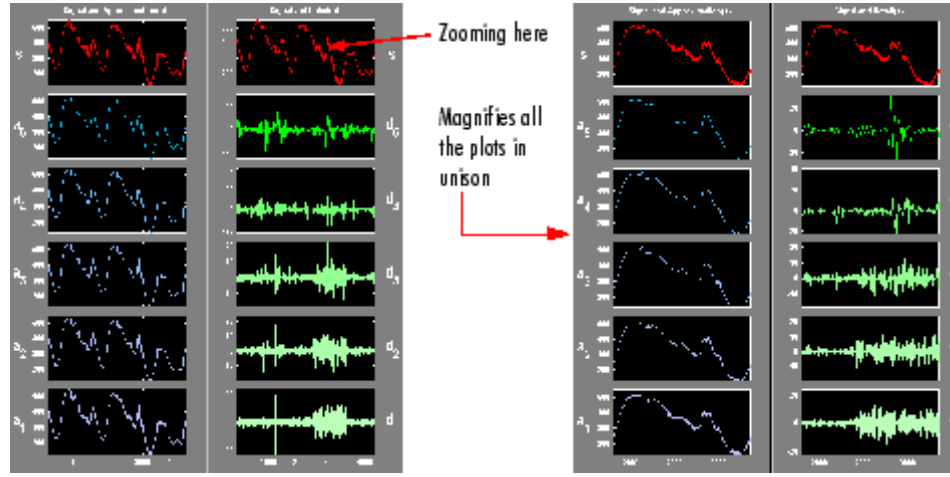
In all the graphical tools, signals and analysis components are color coded as follows.

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)

Connection of Plots

Plots containing related information and graphed on the same abscissa are connected in the sense that manipulations performed on one plot affect all others in the same way. For images, the connection holds in both abscissa and ordinate. You can manipulate all plots along an individual axis (X or Y) or you can manipulate all plots along both axes at the same time (XY).

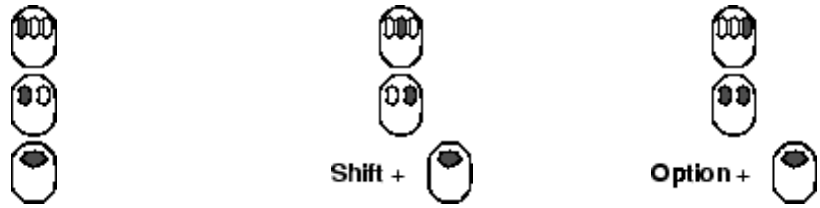
For example, the approximations and details shown in the separate mode view of a decomposition all respond together when any of the plots is magnified or zoomed.



Using the Mouse

Wavelet Toolbox software uses three types of mouse control.

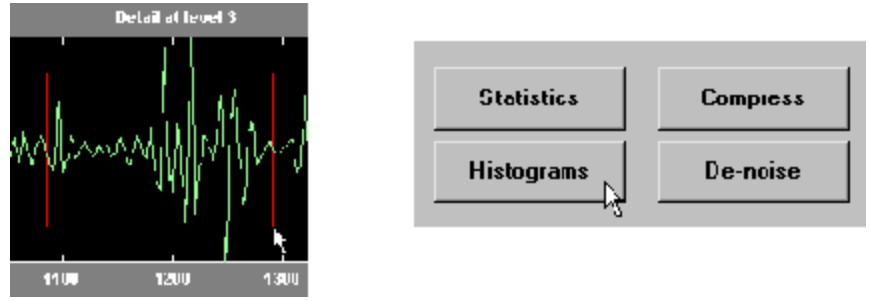
Left Mouse Button	Middle Mouse Button	Right Mouse Button
Make selections. Activate controls.	Display cross-hairs to show position-dependent information.	Translate plots up and down, and left and right.



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

Making Selections and Activating Controls

Most of your work with Wavelet Toolbox graphical tools involves making selections and activating controls. You do this using the left (or only) mouse button.



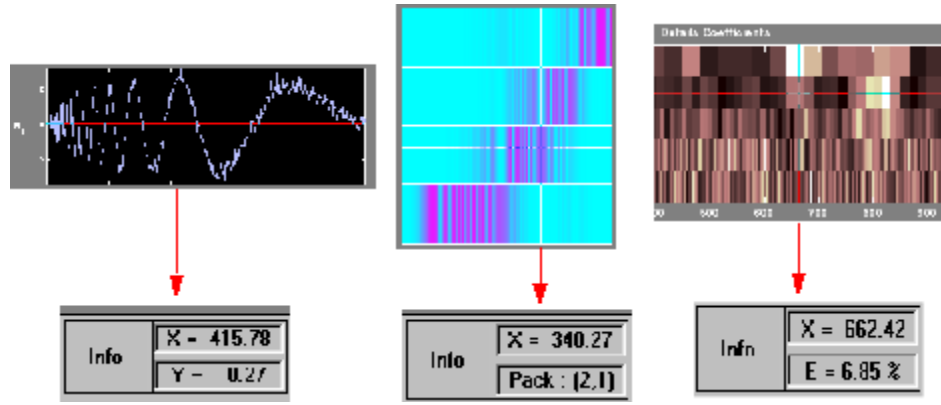
Translating Plots

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 width (or height) 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 **Info** box located at the bottom center of the tool. The type of information that appears depends on what tool you are using and the plot in which your cursor is located. For example, the figure on the left shows the position in X and Y for a signal, the figure on the center shows the X position and the packet number for a discrete wavelet packet analysis, and the figure on the right shows the X position and the percentage of energy present in the detail level for a one-dimensional discrete wavelet analysis.

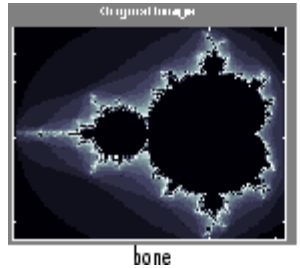


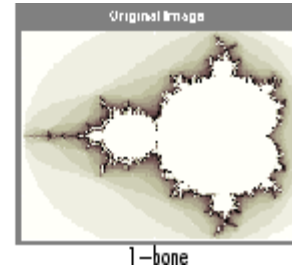
Controlling the Colormap

The **Colormap** selection box, located at the lower right of the window, allows you to adjust the colormap that is used to plot images or coefficients (wavelet or wavelet packet).

This is more than an aesthetic adjustment because 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, shown here using the `bone` and `1 bone` colormaps.

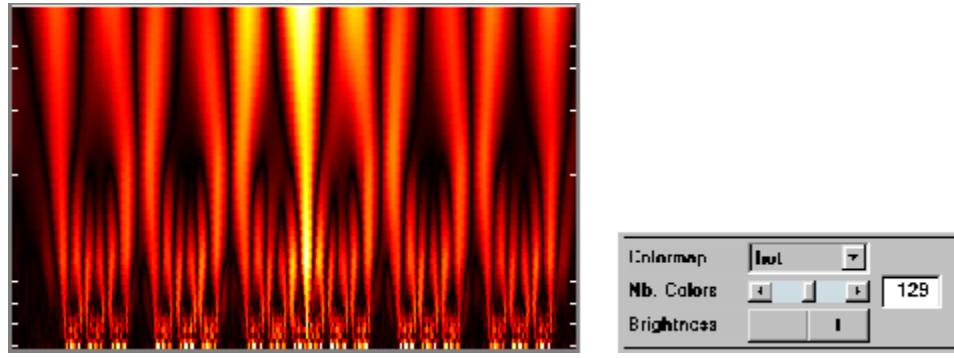




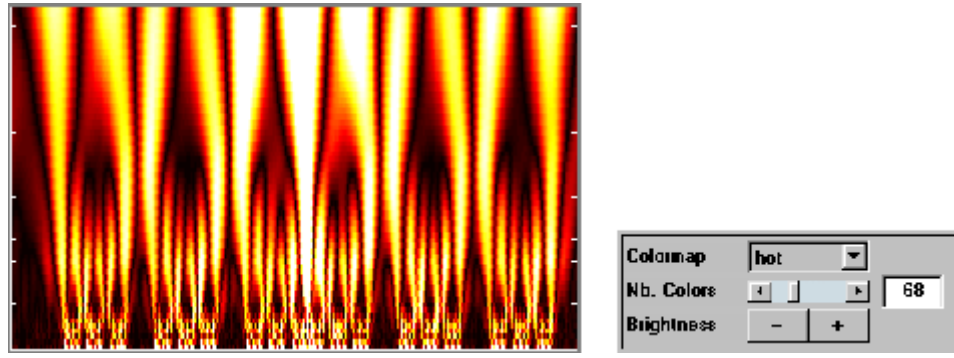
Controlling the Number of Colors

The **Nb. Colors** slider, located at the bottom right of the window, allows you to adjust how many colors the tool uses to plot images or coefficients (wavelet or wavelet packet). You can also use the edit control to adjust the number of colors. 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, shown here using 129 colors.

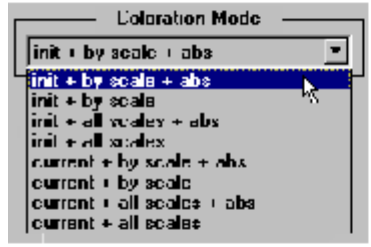


and here using 68 colors.



Controlling the Coloration Mode

In the **Continuous Wavelet** tools, 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 coefficient 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 scale or **all scales**:

When **by scale** is selected, the coloration is done separately for each scale. Otherwise the wavelet coefficients at **all scales** are used to scale the coloration.

abs (or not):

When **abs** is not selected, the values of the coefficients are used (this is called a *Normal Mode*). Otherwise, the absolute values are used (this is called an *Absolute Mode*).

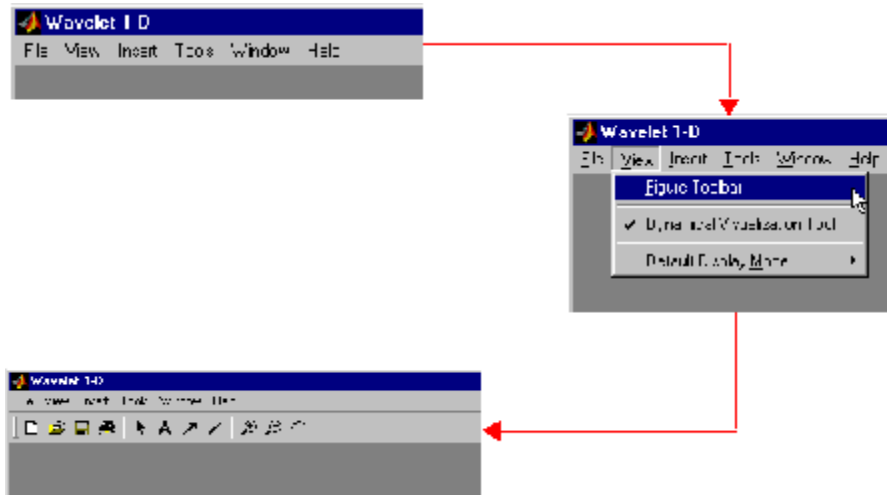
In the **Wavelet 1-D** tool, you access coefficients coloration with the **More Display Options** button, and then select the desired **Coloration Mode** option.

The **More Display Options** button appears only when the **Display mode** is one of the following — Show and Scroll, Show and Scroll (Stem Cfs), Superimposed, and Separate). In this case, **scales** are replaced by **levels** in all options of the **Coloration Mode** menu.

Using Menus

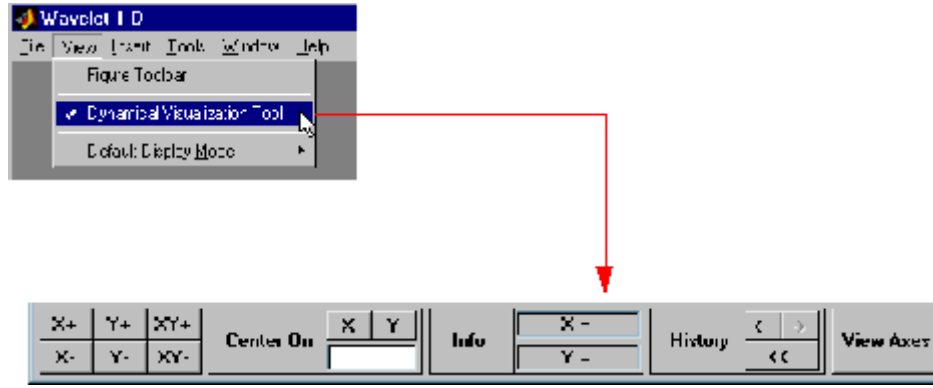
General Menu Bar

At the top of most windows you find the same kind of structure. The menu bar of each figure in Wavelet Toolbox software is very similar to the menu bar of the default MATLAB figures. You can use many of the tools that are offered in the menus and associated toolbar of the standard MATLAB figures.



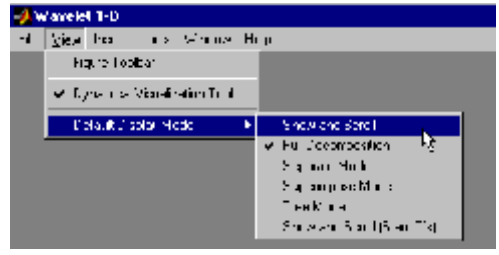
One of the main differences is the **View** menu, which depends on the current tool used.

View Dynamical Visualization Tool Option. The **View > Dynamical Visualization Tool** option lets you enable or disable the **Dynamical Visualization Tool** located at the bottom of each window.



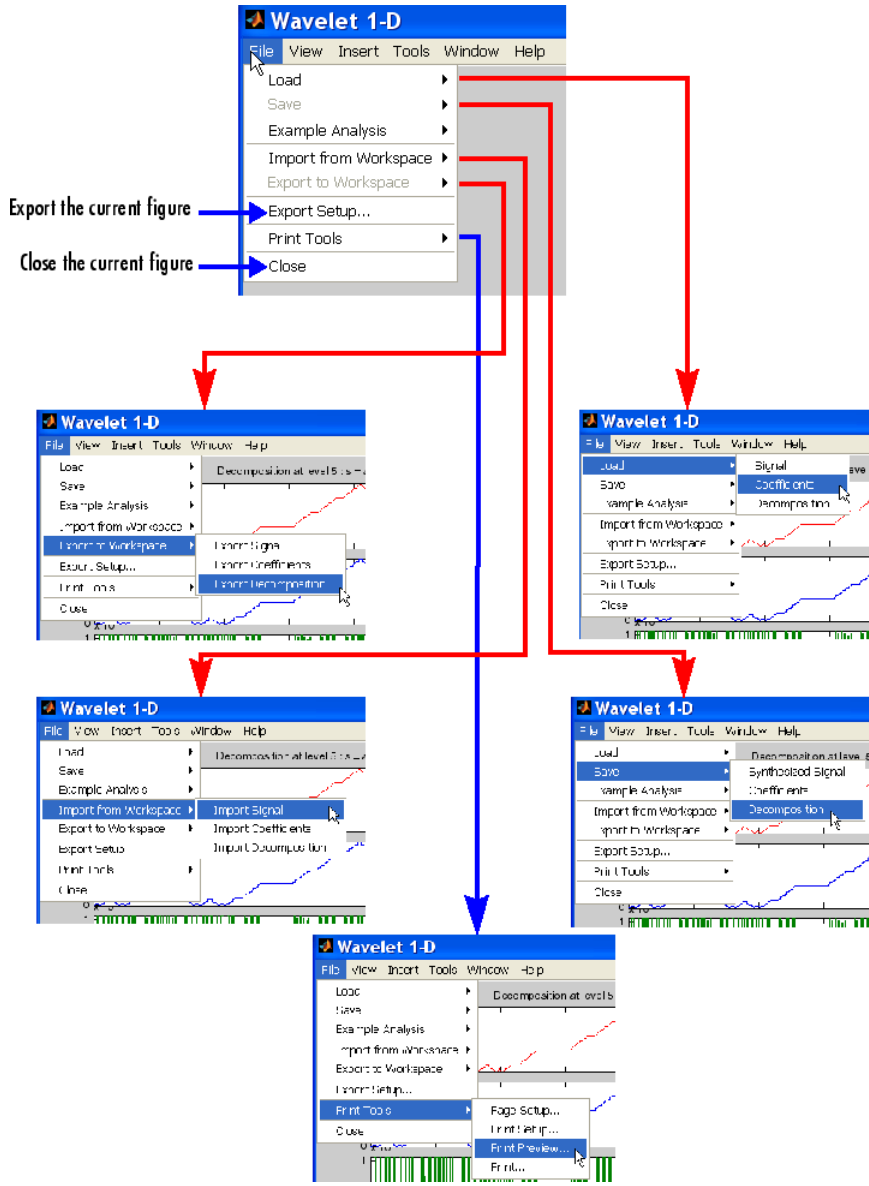
Before using **Zoom In**, **Zoom Out**, or **Rotate 3D** options (or the equivalent icons from the toolbar), you must disable the **Dynamical Visualization Tool** to avoid possible conflicts.

Default Display Mode Option. The **Default Display Mode** option is specific to the **Wavelet 1-D** tool and lets you set a default **Display Mode** for all the different analyses you perform inside the same tool.



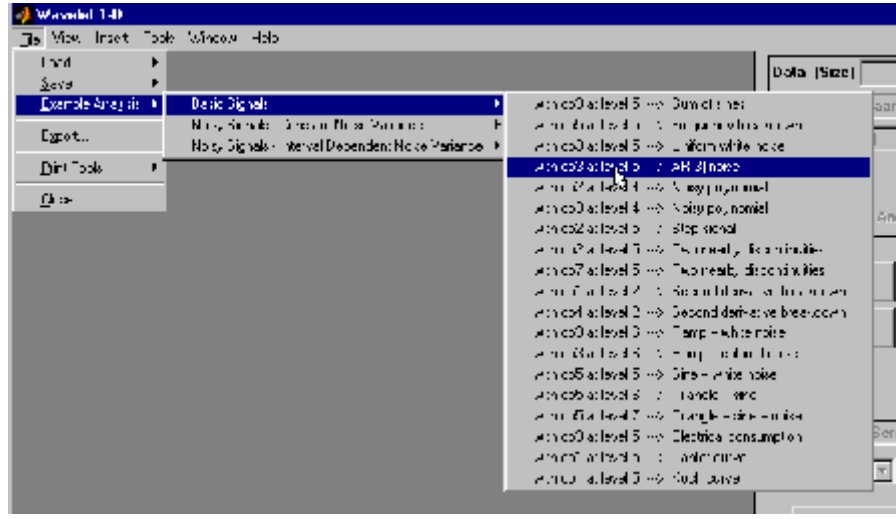
File Menu Options

Depending on the tool you are using, the **File** menu contains customized options. For example, for the **Wavelet 1-D** tool, the following options are added:



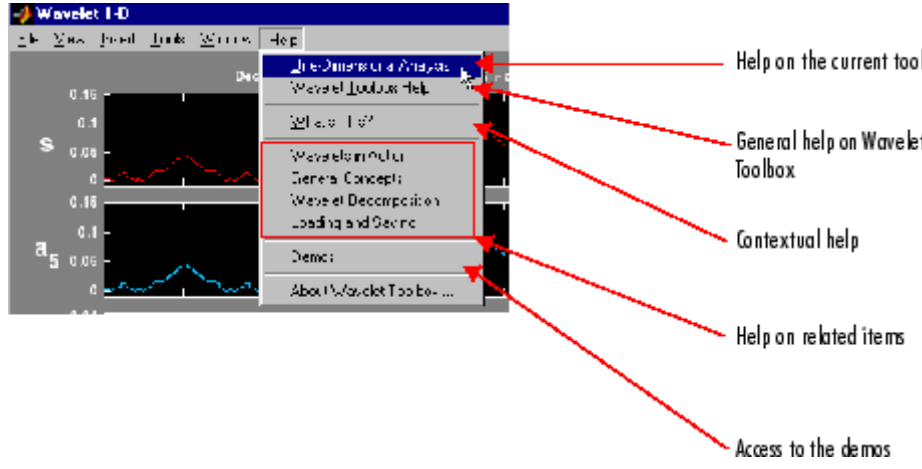
Many windows have a **File⇒Example Analysis** menu option, which allows you to select an analysis example using predefined parameters.

Here is an example of the **Wavelet 1-D** tool.



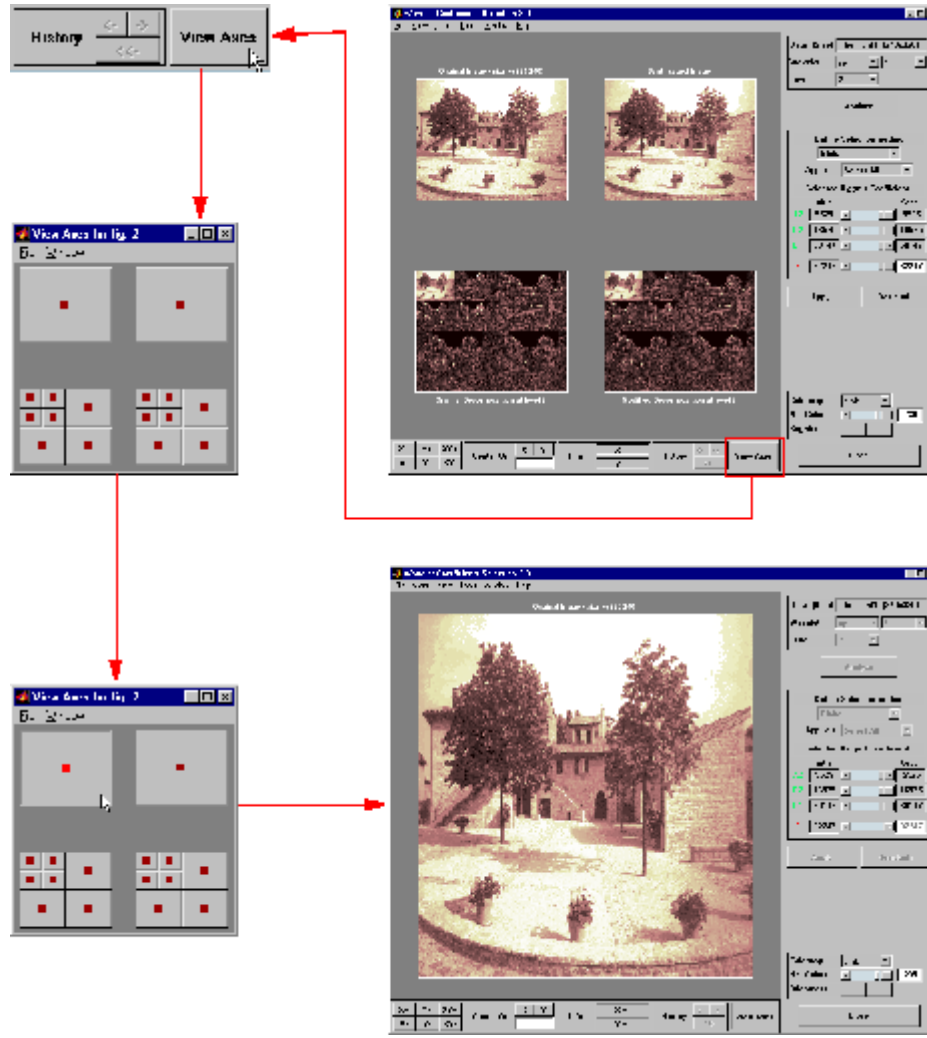
Help Menu Options

The help menu structure is very similar in all the figures, but many options are specific to the current tool in use.

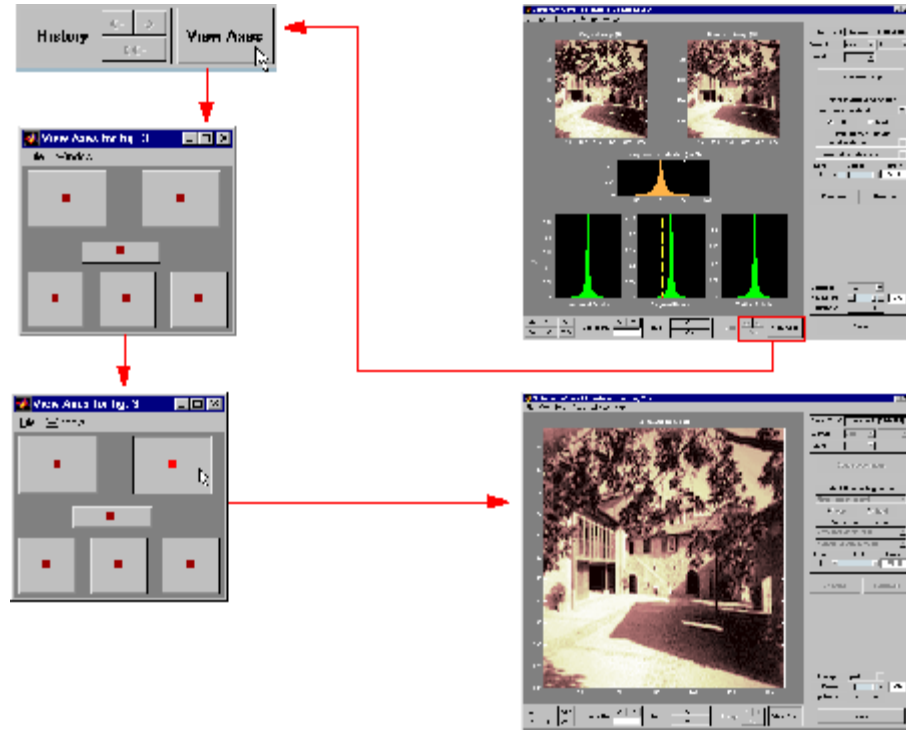


Using the View Axes Button

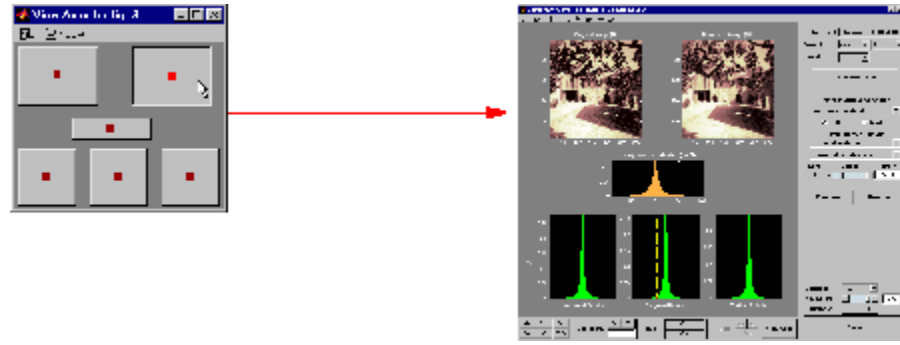
The **Dynamical Visualization Tool** is located at the bottom of most of the windows in the Wavelet Toolbox software. In this tool, the **View Axes** toggle button lets you magnify the axis that you choose.



The toggle buttons in the **View Axes** figure are positioned so that you can understand which axis is correlated with a button.



When you click the same toggle button again, you restore the original view.



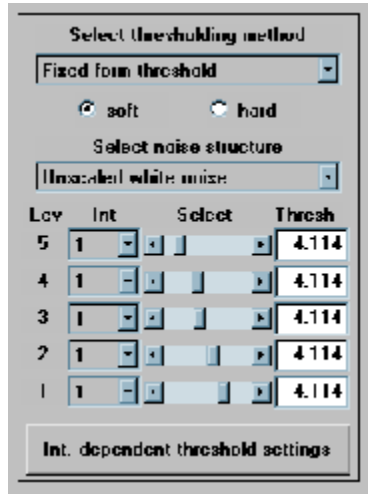
Clicking the **View Axes** toggle button again closes the **View Axes** figure.

Using the Interval-Dependent Threshold Settings Tool

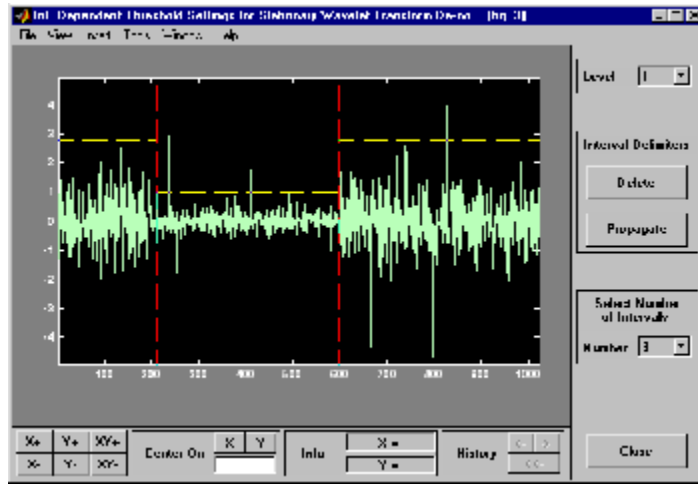
The following tools in the Wavelet Toolbox software let you define interval-dependent thresholds:

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

To the right of the main window for these tools, you see a command frame similar to the following



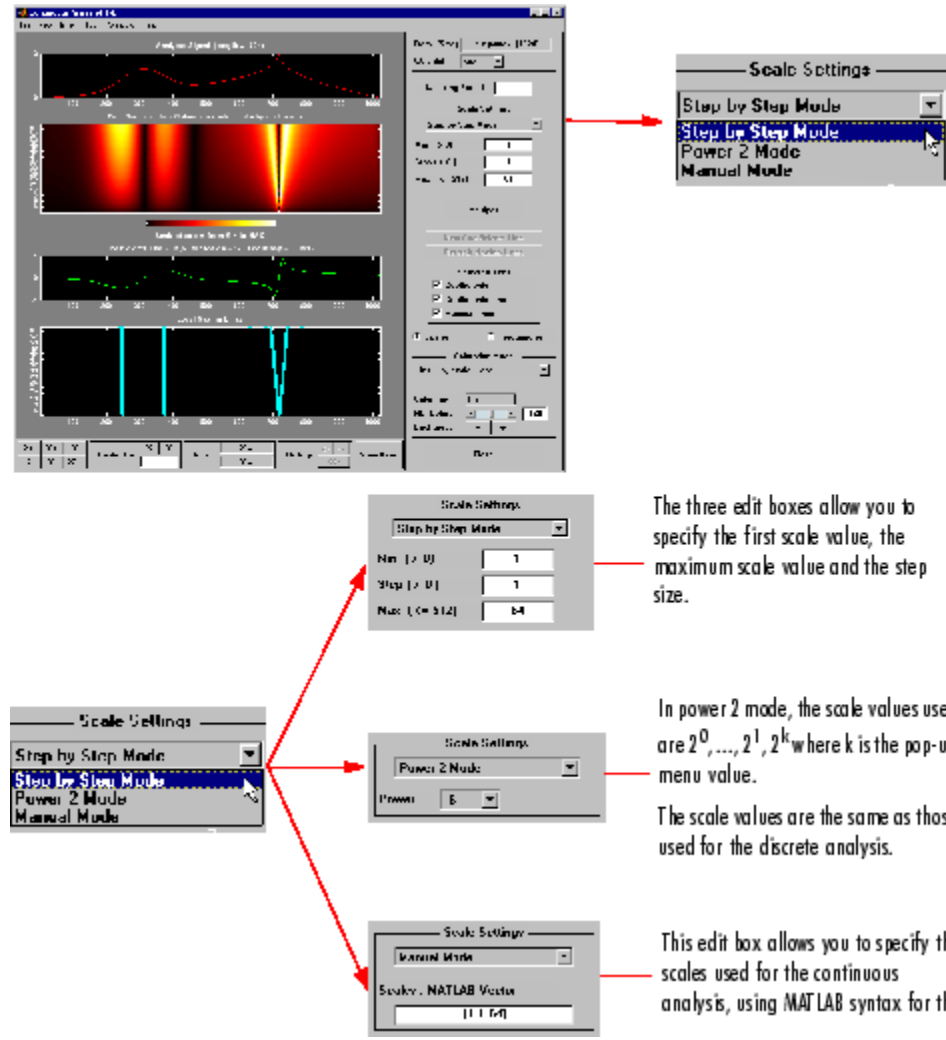
Clicking the **Int. dependent threshold settings** toggle button displays the **Int. dependent Threshold Settings for ...** window. After some computation is performed, the following figure appears.



For more information on how to change interval limits and threshold values, see the section “One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients” in the *Wavelet Toolbox Getting Started Guide*.

Continuous Wavelet Tool Features

Here is an example of an option that allows you to perform analysis using different scale modes.



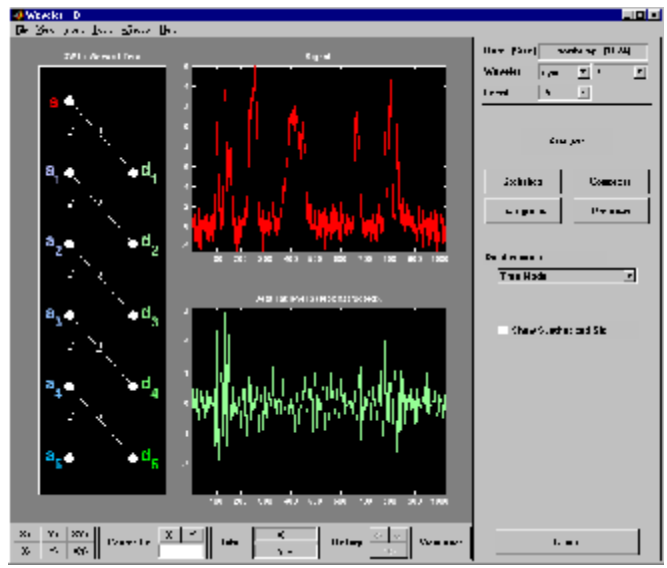
Wavelet 1-D Tool Features

The **Wavelet 1-D** tool is described in the section "One-Dimensional Analysis Using the Graphical Interface" in the *Wavelet Toolbox Getting Started Guide*. Here are two examples of options not covered there.

Tree Mode

This is one of the display options in which you can view the corresponding signal by selecting a node in the tree.

Here, on the left, the node $\mathbf{d}_3$ is selected and the corresponding detail is displayed under the original signal.

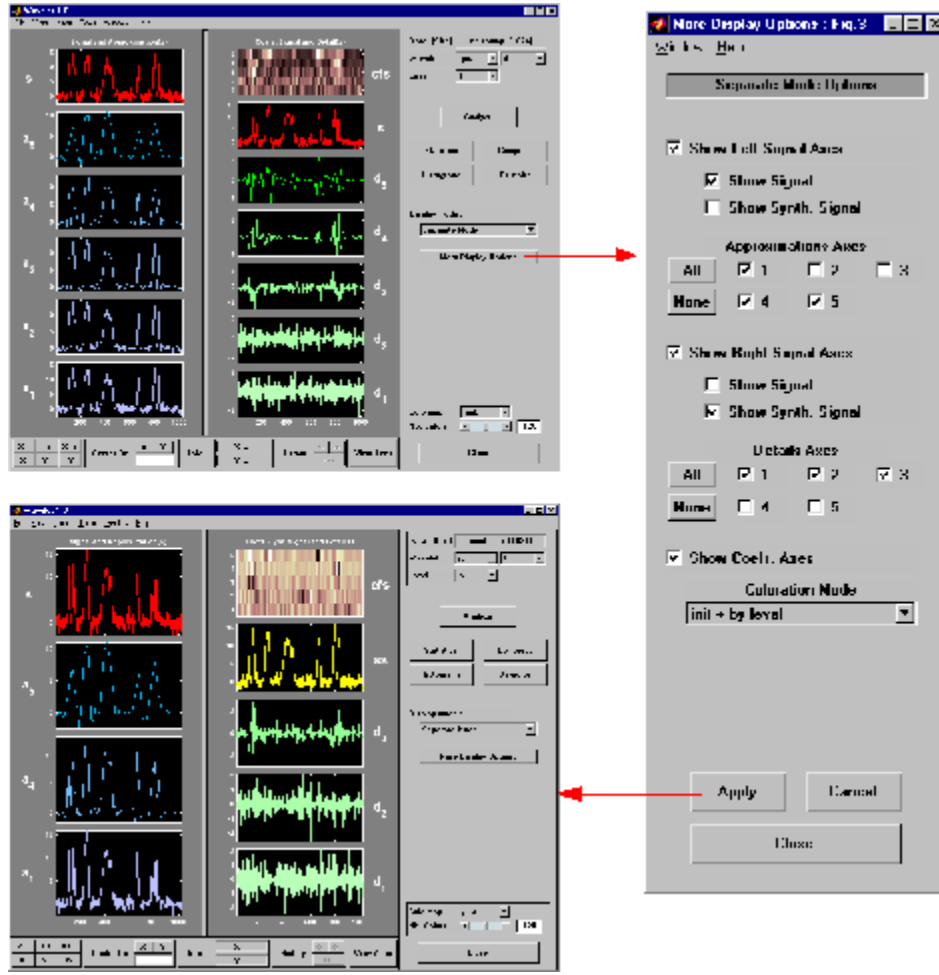


More Display Options

This option allows you to customize what is displayed and is dependent on the current visualization mode.

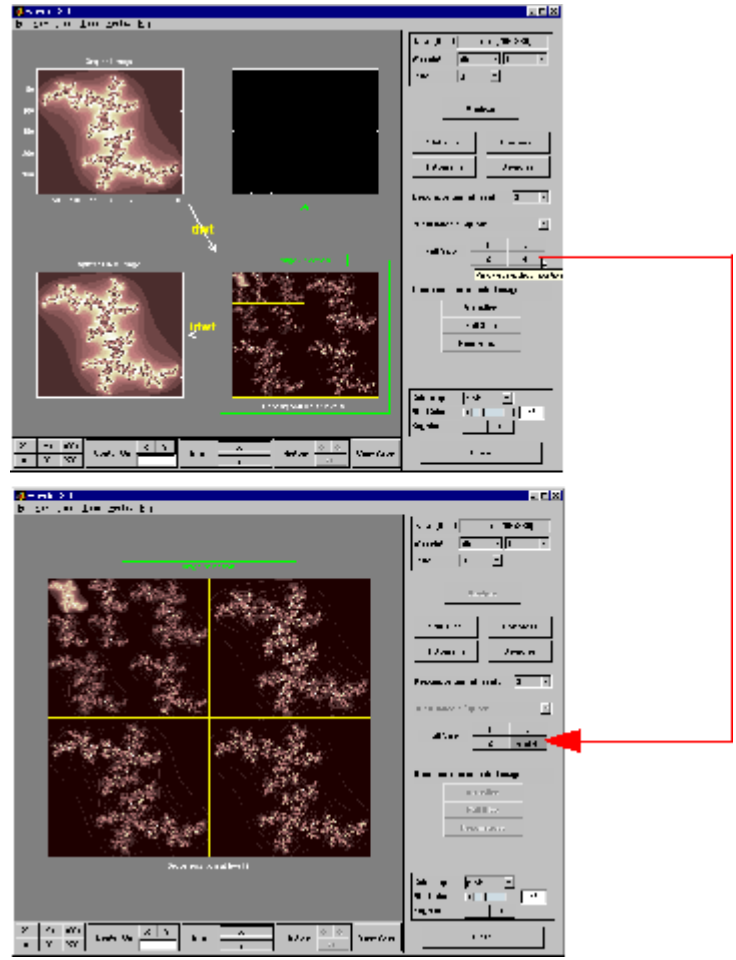
In this example for the **Separate Mode**, we have chosen not to display the coefficients of approximation for levels 2 and 3, nor the coefficients of detail

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.



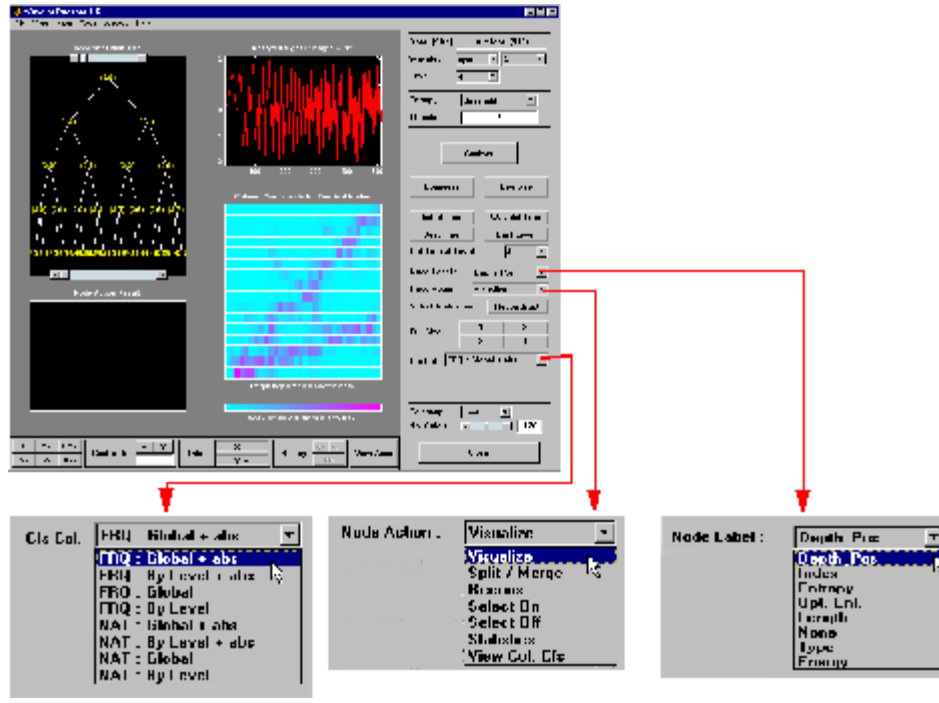
Wavelet 2-D Tool Features

The **Wavelet 2-D** tool is described in the section "Two-Dimensional Analysis Using the Graphical Interface" in the *Wavelet Toolbox Getting Started Guide*. 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)

For descriptions of the **Wavelet Packet 1-D** and **Wavelet Packet 2-D** tools, refer to Chapter 3, "Using Wavelet Packets". These tools 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 6-154).

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 in the following sections.

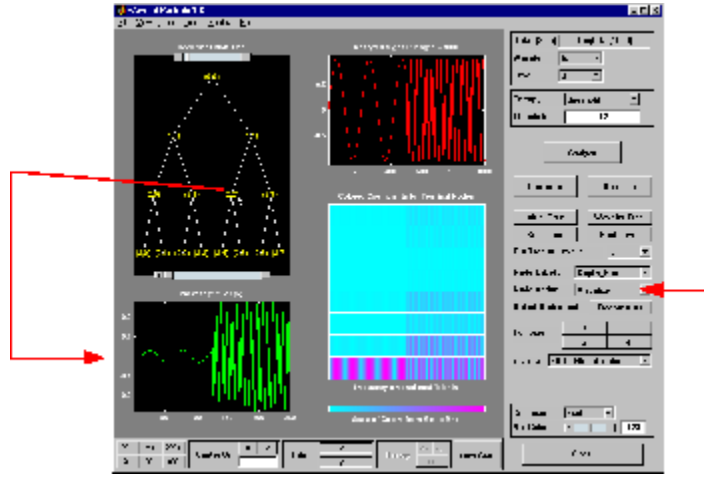
Node Label

The node labels can be changed using the pop-up menu. For example, the **Type** option labels the nodes with **(a)** for approximation and **(d)** for detail.

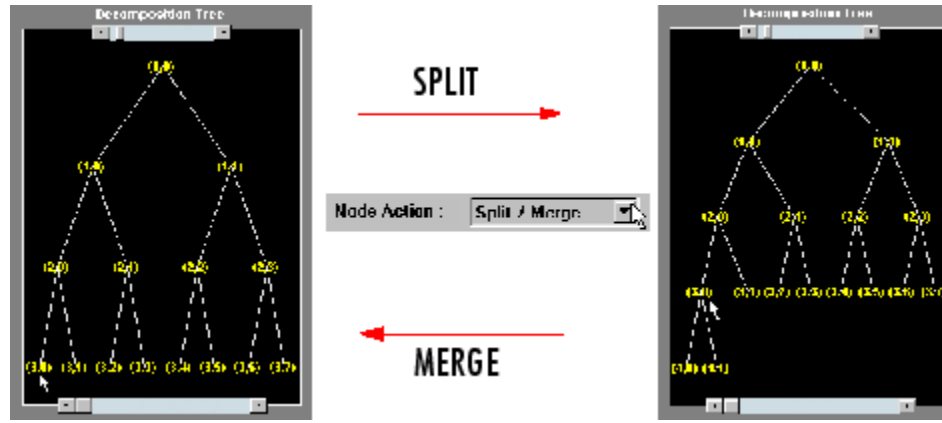
Node Action Functionality

The available options in the **Node Action** menu are

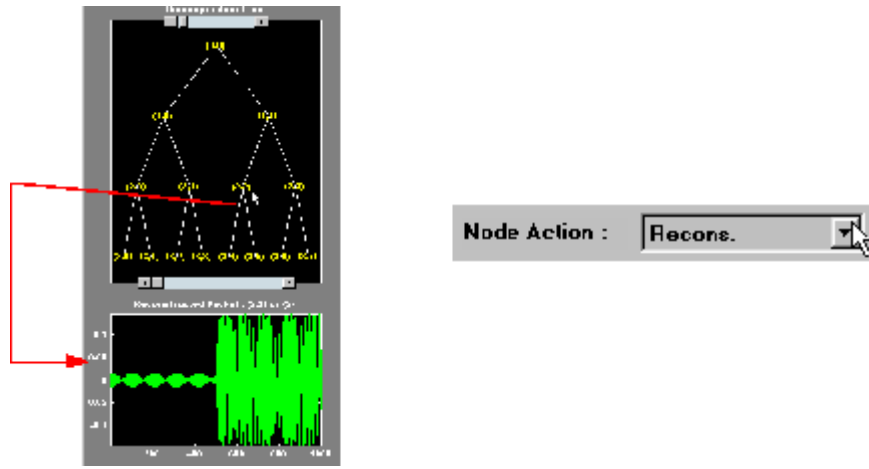
- **Visualize:** When you select a node in the wavelet packet tree the corresponding signal appears.



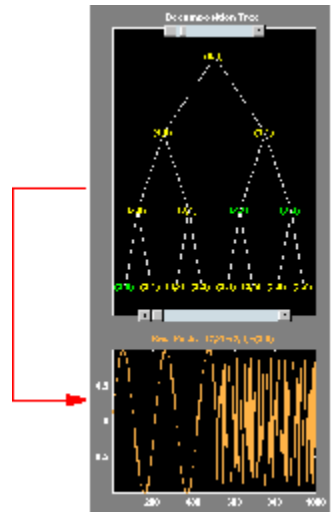
- **Split/Merge:** If a terminal node is selected, it is split, growing the wavelet packet tree. Selecting other nodes has the behavior of merging all the nodes below it in the wavelet packet tree.

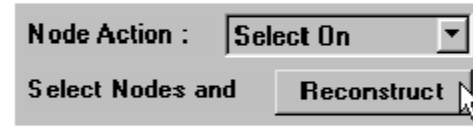


- **Recons.:** When you select a node in the wavelet packet tree, the corresponding reconstructed signal appears.

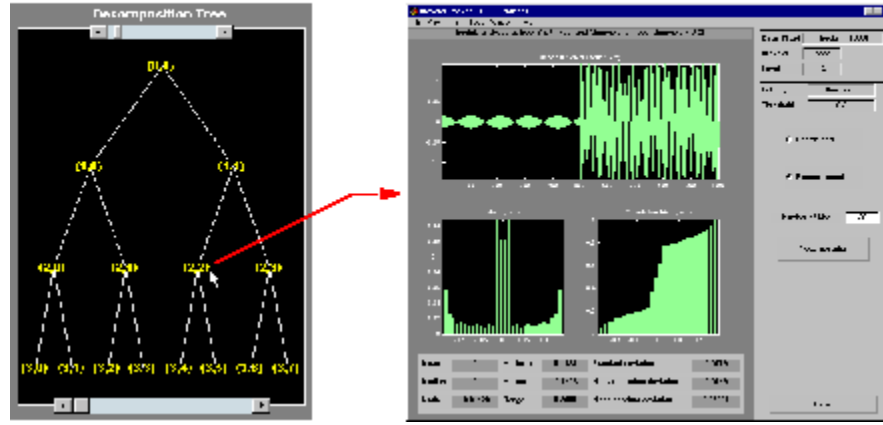


- **Select On/Off:** When **On**, you can select many nodes in the wavelet packet tree. Then you can reconstruct a synthesized signal from the selected nodes using the **Reconstruct** button on the main window. Use the **Off** selection to deselect all the previous selected nodes.

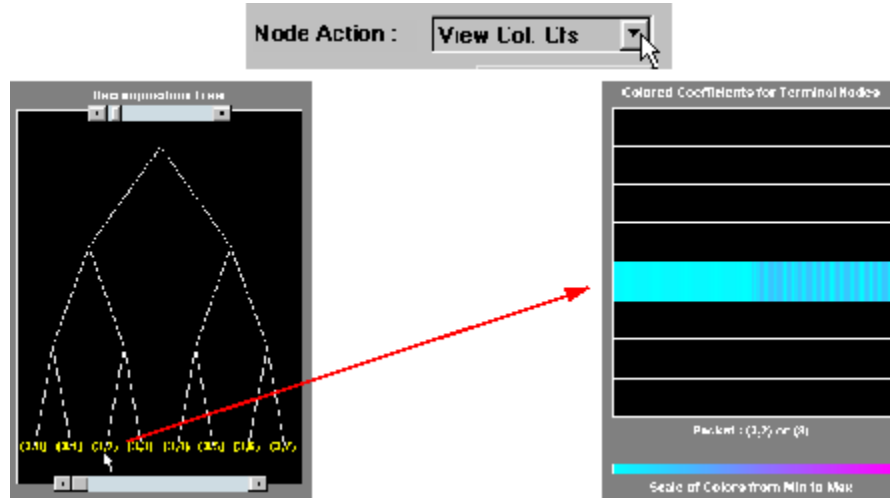




- **Statistics:** When you select a node in the wavelet packet tree, the **Statistics** tool appears using the signal corresponding to the selected node.



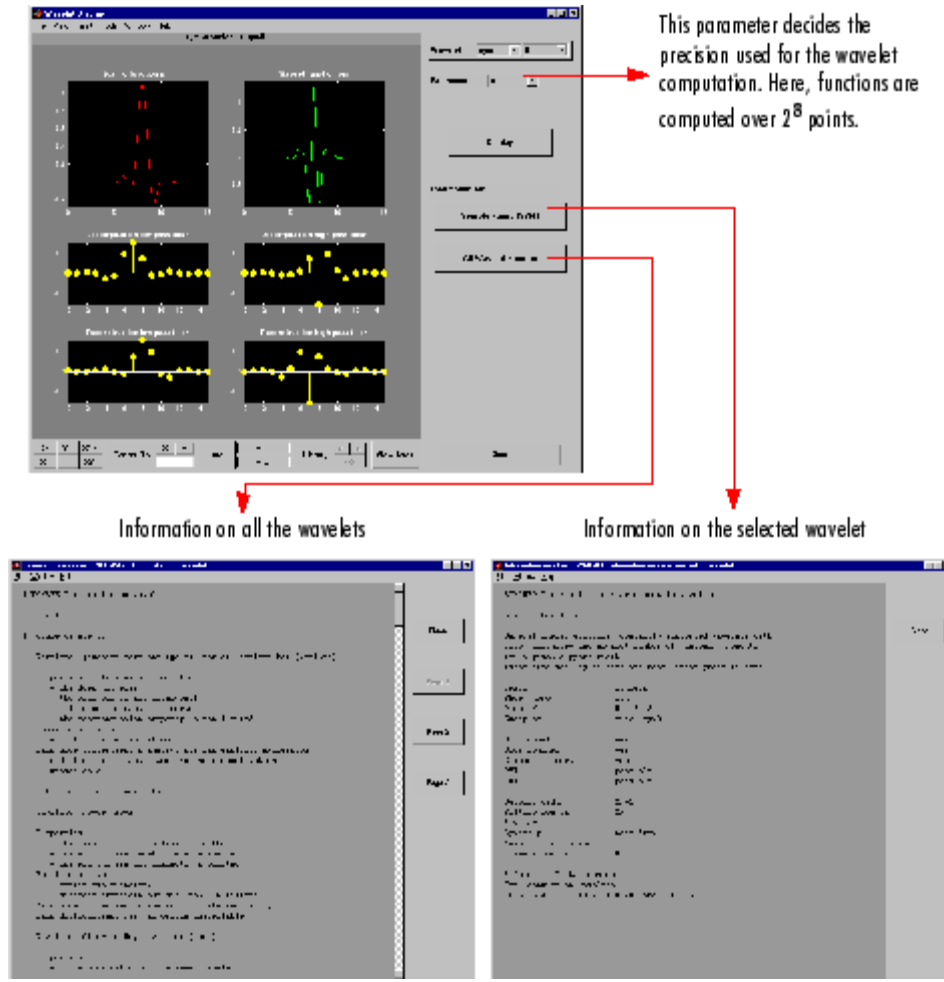
- **View Col. Cfs.:** When active, this option removes all the colored coefficients displayed, and lets you redraw only the corresponding coefficients by selecting a node in the wavelet packet tree.


Node Action :
View Col. Cfs
Colored Coefficients for Terminal Nodes
Scale of Colors from Min to Max

Wavelet Display Tool

The **Wavelet Display** tool is mentioned in the section "Introduction to the Wavelet Families" in the *Wavelet Toolbox Getting Started Guide*.

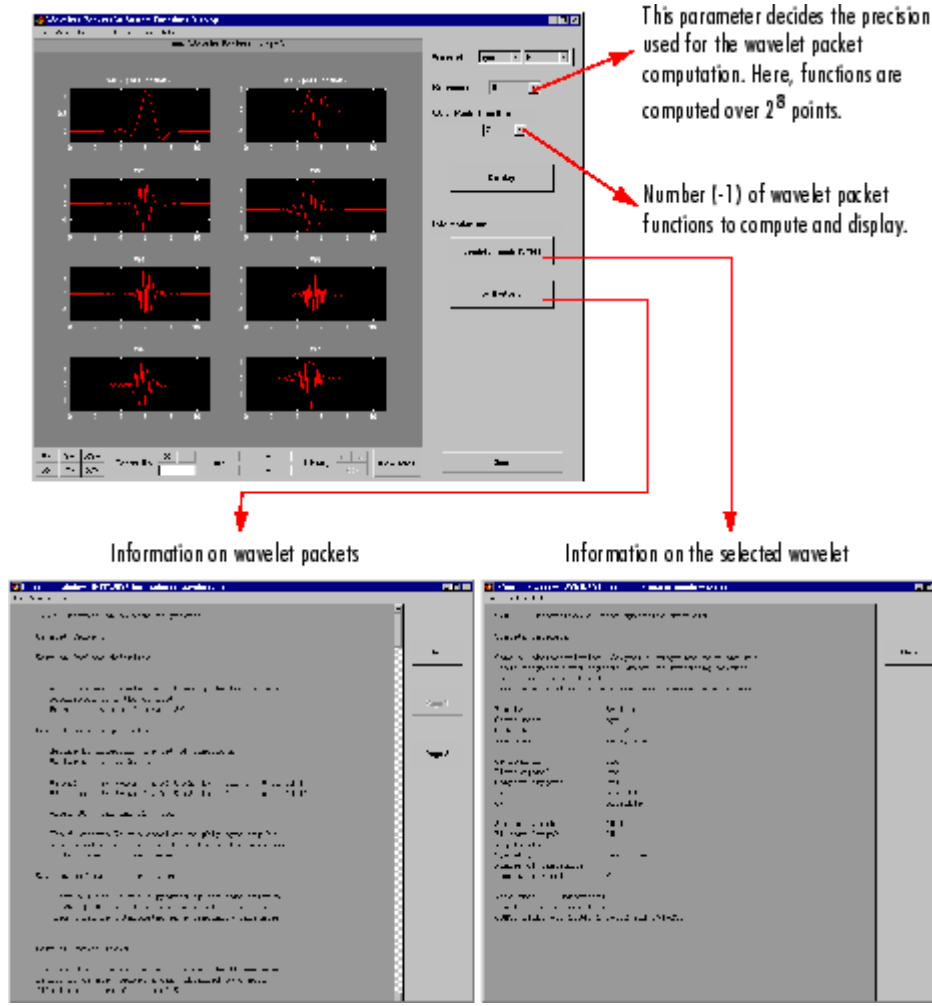
Here, we show the main window and the associated information windows with some additional comments.



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.



B

Object-Oriented Programming

This appendix explains concepts of object-oriented programming as they apply to Wavelet Toolbox software.

Introduction to Object-Oriented Features

In the Wavelet Toolbox software, some object-oriented programming features are used for wavelet packet tree structures.

You may want to skip this appendix, if you prefer to use the command line functions and graphical user interface (GUI) without knowing about the underlying objects and classes. But, it is useful for **Save** and **Load** actions where objects are involved.

These aspects, related to a minimal use, are described in Chapter 3, “Using Wavelet Packets”, and in the reference pages for the corresponding functions.

This appendix lets you understand the objects used in the toolbox, use some functions that are not fully documented in the reference pages, and extend the toolbox functionality using the predefined tree structures and some object programming features.

It is helpful to be familiar with the basic MATLAB object-oriented language and terminology.

Short Description of Objects in the Wavelet Toolbox Software

Four classes of objects are defined in the Wavelet Toolbox software.

The hierarchical organization of these objects is described in the following scheme:

Only the Wavelet Packet tools (1-D and 2-D) use the previous objects. More precisely, **WPTREE** objects are used to build wavelet packets.

A short description of this hierarchy of objects follows. For a more detailed description see “Short Description of Objects in the Wavelet Toolbox Software” on page B-3.

The **WTBO** class is an abstract class. Any object in the toolbox is parented by a **WTBO** object and would inherit the methods and fields of the **WTBO** class.

The **NTREE** class is dedicated to tree manipulation (node labels, node splitting, node merging, ...), and it is also an abstract class. The main methods are

- `nodejoin`, which recomposes nodes
- `nodesplt`, which decomposes nodes
- `wtreemgr`, which lets you access most of tree and node information (order, depth, terminal nodes, ascendants of a node, ...)

In fact, the `wtreemgr` method is not used directly, but you can use the functions `treeord`, `treedpth`, `leaves`, `nodeasc`, ..., and the method `get`.

The **DTREE** class is dedicated to trees with associated data: vectors or matrices.

This class is also an abstract class and some methods have to be overloaded.

The aim of the **WPTREE** class is to manage wavelet packets 1-D and 2-D.

Some methods of the **DTREE** class have been overloaded, for example: `split`, `merge`, and `recons`.

Most of the methods are specific to the class **WPTREE**; for example: `bestlevt`, `besttree`, and `wp2wtree`.

By typing `help wavelet` you can see the available methods in the **Tree Management Utilities** and **Wavelets Packets Algorithms** sections.

Simple Use of Objects Through Four Examples

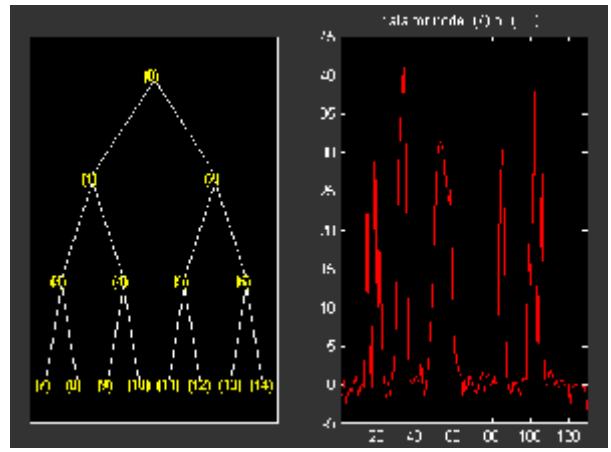
You can use command line functions, GUI functions, or you can mix both of them to work with wavelet packet trees (**WPTREE** objects). The most useful commands are

- `plot`, `drawtree`, and `readtree`, which let you plot and get a wavelet packet tree
- `wpjoin` and `wpsplt`, which let you change a wavelet packet tree structure
- `get`, `read`, and `write`, which let you read and write coefficients or information in a wavelet packet tree

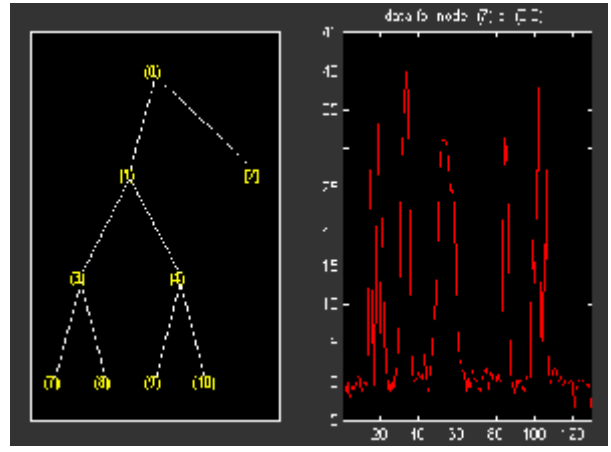
We can see some of these features in the following examples.

Example 1: plot and wpviewcf

```
load noisbump
x = noisbump;
t = wpdec(x,3,'db2');
fig = plot(t);

% Change Node Label from Depth_position to Index and
% click the node (7). You get the following figure.
```

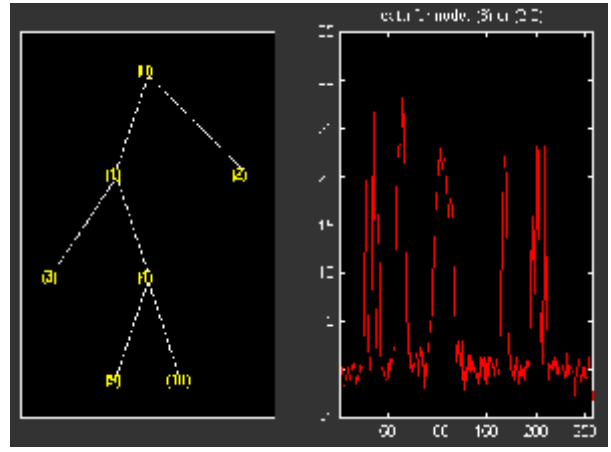


```
% Change Node Action from Visualize to Split-Merge and
% merge the node 2. You get the following figure.
```

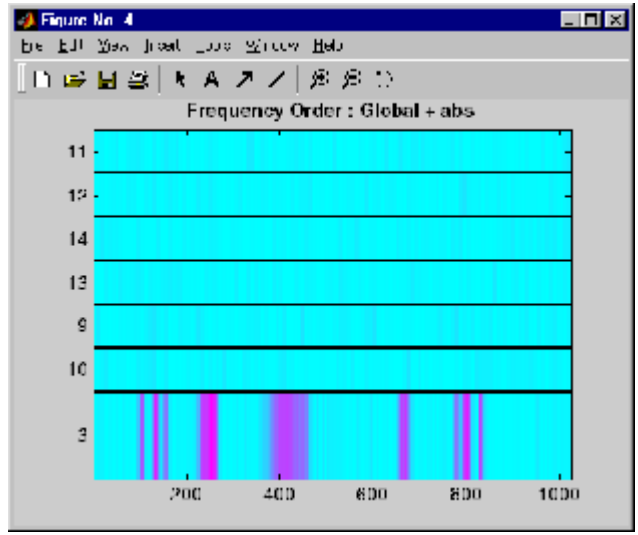


```
% From the command line, you can get the new tree.
newt = plot(t,'read',fig);

% The first argument of the plot function in the last command
% is dummy. Then the general syntax is:
%    newt = plot(DUMMY,'read',fig);
% where DUMMY is any object parented by an NTREE object.
```

```
% DUMMY can be any object constructor name, which returns
% an object parented by an NTREE object. For example:
%    newt = plot(ntree,'read',fig);
%    newt = plot(dtree,'read',fig);
%    newt = plot(wptree,'read',fig);

% From the command line you can modify the new tree,
% then plot it.
newt = wpjoin(newt,3);
fig2 = plot(newt);

% Change Node Label from Depth_position to Index and
% click the node (3). You get the following figure.
```

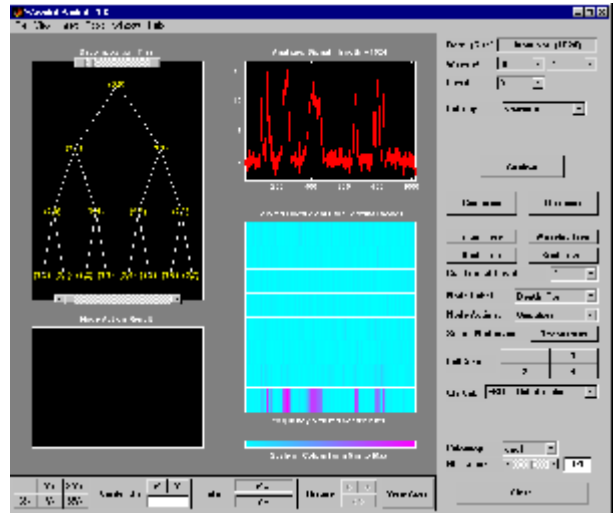


```
% Using plot(newt,fig), the plot is done in the figure fig,
% which already contains a tree object.

% You can see the colored wavelet packets coefficients using
% from the command line, the wpviewcf function (type help
% wpviewcf for more information).
wpviewcf(newt,1)

% You get the following plot, which contains the terminal nodes
% colored coefficients.
```

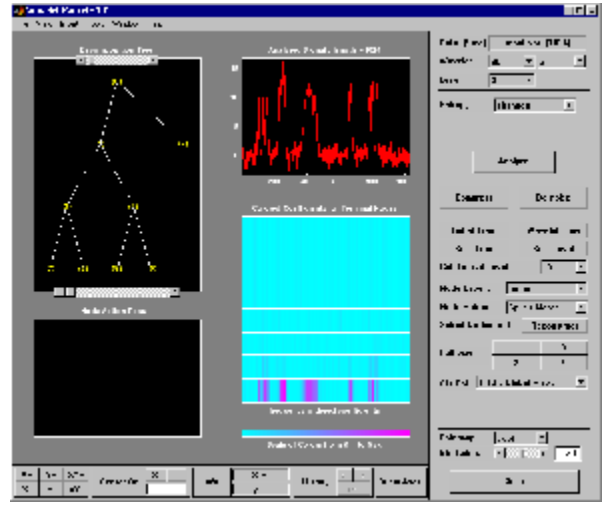


Example 2: drawtree and readtree

```
load noisbump
x = noisbump;
t = wpdec(x,3,'db2');
fig = drawtree(t);

% The last command creates a GUI.
% The same GUI can be obtained using the main menu and:
% - clicking the Wavelet Packet 1-D button,
% - loading the signal noisbump,
% - choosing the level and the wavelet
% - clicking the decomposition button.
% You get the following figure.
```

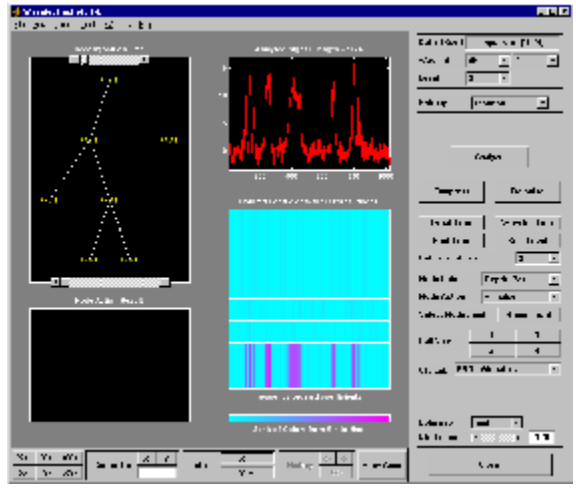


```
% From the GUI, you can modify the tree.
% For example, change Node label from Depth_Position to Index,
% change Node Action from Visualize to Split_Merge and
% merge the node 2.
% You get the following figure.
```



```
% From the command line, you can get the new tree.
```

```
newt = readtree(fig);

% From the command line you can modify the new tree;
% then plot it in the same figure.
newt = wpjoin(newt,3);
drawtree(newt,fig);
```



You can mix previous commands. The GUI associated with the `plot` command is simpler and quicker, but more actions and information are available using the full GUI tools related to wavelet packets.

The methods associated with WPTREE objects let you do more complicated actions.

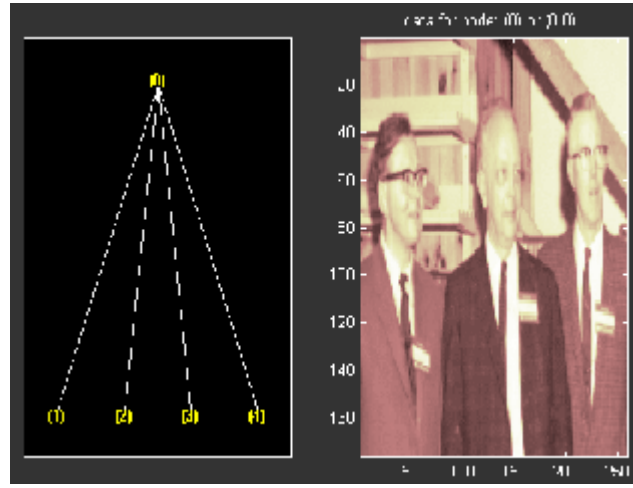
Namely, using `read` and `write` methods, you can change terminal node coefficients.

Let's illustrate this point with the following "funny" example.

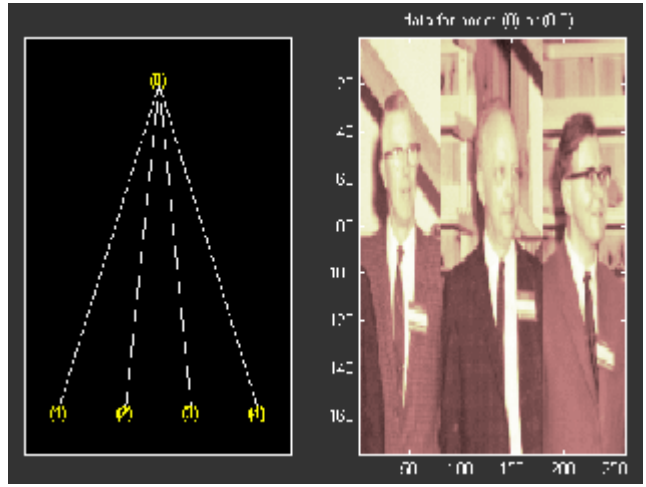
Example 3: A Funny One

```
load gatlin2
t = wpdec2(X,1,'haar');
plot(t);
```

```
% Change Node Label from Depth_position to Index and
% click the node (0). You get the following figure.
```



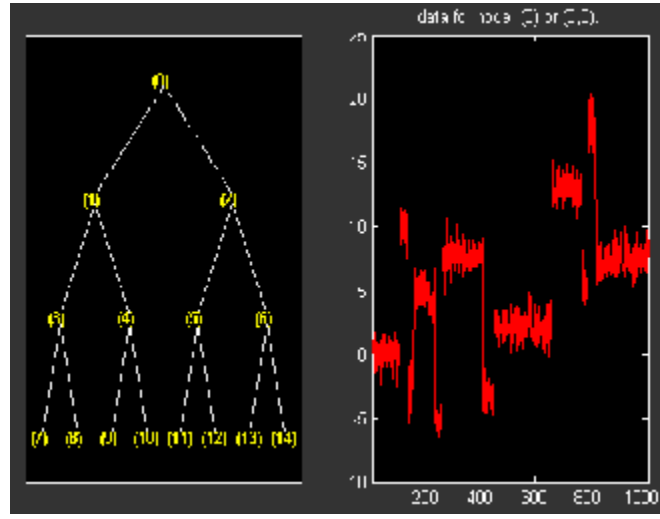
```
% Now modify the coefficients of the four terminal nodes.
newt = t;
NBcols = 40;

for node = 1:4
  cfs = read(t,'data',node);
  tmp = cfs(1:end,1:NBcols);
  cfs(1:end,1:NBcols) = cfs(1:end,end-NBcols+1:end);
  cfs(1:end,end-NBcols+1:end) = tmp;
  newt = write(newt,'data',node,cfs);
end
plot(newt)

% Change Node Label from Depth_position to Index and
% click on the node (0). You get the following figure.
```



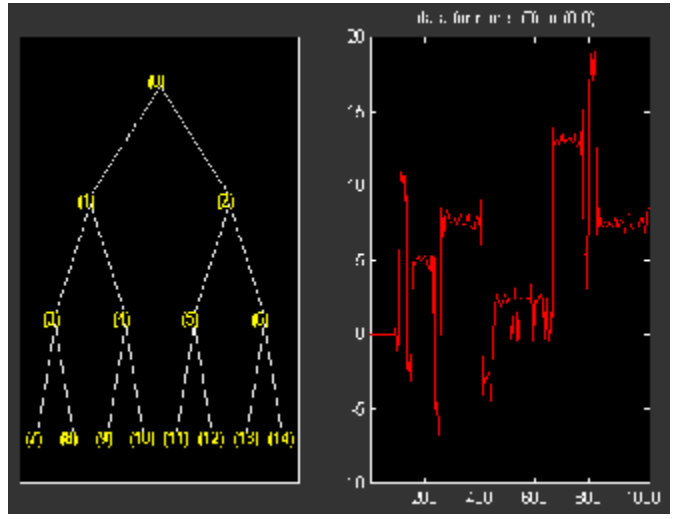
You can use this method for a more useful purpose. Let's see a de-noising example.

Example 4: Thresholding Wavelet Packets

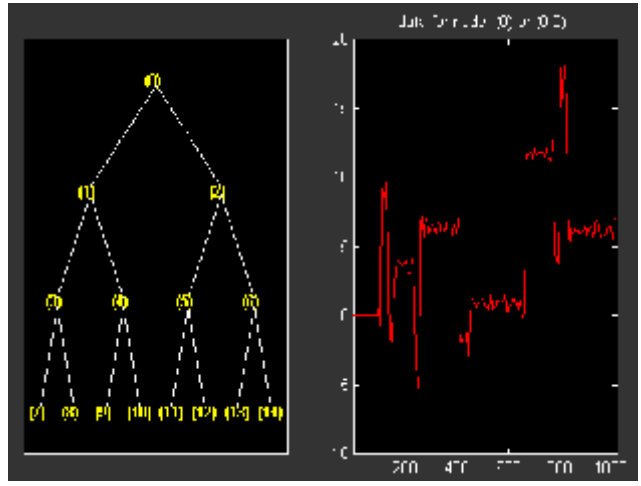
```
load noisbloc
x = noisbloc;
t = wpdec(x,3,'sym4');
plot(t);
% Change Node Label from Depth_position to Index and
% click the node (0). You get the following plot.
```



```
% Global thresholding.
t1 = t;
sorh = 'h';
thr = wthrmngr('wp1ddenoGBL','penalhi',t);
cfs = read(t,'data');
cfs = wthresh(cfs,sorh,thr);
t1  = write(t1,'data',cfs);
plot(t1)

% Change Node Label from Depth_position to Index and
% click the node (0). You get the following plot.
```



```
% Node by node thresholding.
t2 = t;
sorh = 's';
thr(1) = wthrmngr('wp1ddenoGBL','penalhi',t);
thr(2) = wthrmngr('wp1ddenoGBL','sqtwologswn',t);
tn  = leaves(t);
for k=1:length(tn)
  node = tn(k);
  cfs = read(t,'data',node);
  numthr = rem(node,2)+1;
  cfs = wthresh(cfs,sorh,thr(numthr));
  t2 = write(t2,'data',node,cfs);
end
plot(t2)

% Change Node Label from Depth_position to Index and
% click the node (0). You get the following plot.
```



Detailed Description of Objects in the Wavelet Toolbox Software

The following sections describe the objects in the Wavelet Toolbox software:

WTBO Object

Class **WTBO** (Wavelet Toolbox Object) -- Parent class: none

Fields

`wtboInfo`	Object information (Not used)
`ud`	Userdata field

Methods

`wtbo`	Constructor for the class WTBO.
`get`	Get WTBO object field contents.
`set`	Set WTBO object field contents.

Comments

Since any object in the toolbox is parented by a WTBO object, you can associate your own data to an object using the `'ud'` field, and then access it.

If `Obj` is an object (parented by a WTBO object), use

```
Obj = set(Obj,'ud',MyData)
```

to define the data.

To retrieve the data, use

```
MyData = get(O,'ud')
```

NTREE Object

Class **NTREE** (New Tree) -- Parent class: **WTBO**

Fields

`wtbo`	Parent object
`order`	Tree order
`depth`	Tree depth
`spsch`	Split scheme for nodes
`tn`	Column vector with terminal nodes indices

Methods

`ntree`	Constructor for the class NTREE.
`findactn`	Find active nodes.
`get`	Get NTREE object field contents.
`nodejoin`	Recompose node(s).
`nodesplt`	Split (decompose) node(s).
`plot`	Plot NTREE object.
`set`	Set NTREE object field contents.
`tlabels`	Labels for the nodes of a tree.
`wtreemgr`	Manager for NTREE object.

Private

`locnumcn`	Local number for a child node
`tabofasc`	Table of ascendants of nodes

DTREE Object

Class **DTREE** (Data Tree) -- Parent class: **NTREE**

Fields

`ntree`	Parent object
`allNI`	All Nodes Information
`terNI`	Terminal Nodes Information

Fields Description

`allNI` is a NBnodes-by-3 array such that

```
allNI(N,:) = [ind,size(1,1),size(1,2)]
```

- `ind` = index of the node `N`
- `size` = size of data associated with the node `N`

`terNI` is a 1-by-2 cell array such that

- `terNI{1}` is an `NB_TerminalNodes`-by-2 array such that
  - `terNI{1}(N,:)` is the size of coefficients associated with the `N`-th terminal node. The nodes are numbered from left to right and from top to bottom. The root index is 0.
- `terNI{2}` is a row vector containing the previous coefficients stored row-wise in the above specified order.

Methods

`dtree`	Constructor for the class DTREE.

`expand`	Expand data tree.
`fmdtree`	Field manager for DTREE object.
`nodejoin`	Recompose node.

`nodesplt`	Split (decompose) node.
`rnodcoef`	Reconstruct node coefficients.

`defaninf`	Define node information (all nodes).
`get`	Get DTREE object field contents.
`plot`	Plot DTREE object.
`read`	Read values in DTREE object fields.
`set`	Set DTREE object field contents.
`write`	Write values in DTREE object fields.

`merge`	Merge (recompose) the data of a node.
`recons`	Reconstruct node coefficients.
`split`	Split (decompose) the data of a terminal node.

Comments

- After the constructor, the first set of methods (between line separators) might not be overloaded (or only with great care). The second set of methods can be overloaded. The third set of methods must be overloaded to recompose, reconstruct, or decompose nodes data.
- The method `nodejoin` calls the method `merge`, the method `nodesplt` calls the method `split`, and the method `rnodcoef` calls the method `recons`.
- To define nodes information, you must overload the method `defaninf`. For each node `N`, the basic information is given by

  ```
  allNI(N,1:3): [index,size(1,1),size(1,2)];
  ```

  You can add other information by adding columns to `allNI`.

  See the WPTREE object method for an example.
- If the method `get` is not overloaded, using the DTREE `get` method you can get some object field contents (but not all).

For example, if `T` is parented by a DTREE object of order 2 and if `'Tfield'` is a field of `T`, whose content is `Tval`, `[a,b] = get(t,'order','Tfield')` returns `a = 2` and `b = 'errorWTBX'`. Nevertheless, using a nondocumented method you can get the right values. Namely: `[a,b] = getwtbo(t,'order','Tfield')` returns `a = 2` and `b=Tval`.

WPTREE Object

Class **WPTREE** (Wavelet Packet Tree) -- Parent class: **DTREE**

Fields

`dtree`	Parent object
`wavInfo`	Structure (wavelet information)
`entInfo`	Structure (entropy information)

Fields Description

`wavInfo`

`wavName`	Wavelet Name
`Lo_D`	Low Decomposition filter
`Hi_D`	High Decomposition filter
`Lo_R`	Low Reconstruction filter
`Hi_R`	High Reconstruction filter

`entInfo`

`entName`	Entropy Name
`entPar`	Entropy Parameter

`allNI` Array(nbnode,5) `(field of the dtree parent object)`

`[ind,size,ent,ento]`

`ind`	Index
`size`	Size of data
`ent`	Entropy
`ento`	Optimal Entropy

Methods

Constructor.

Method	Description
`wptree`	Constructor for the class **WPTREE**

Methods That Overload Those of DTREE Class.

Method	Description
`defaninf`	Define node information (all nodes).
`get`	Get WPTREE object field contents.
`merge`	Merge (recompose) the data of a node.
`read`	Read values in WPTREE object fields.
`recons`	Reconstruct wavelet packet coefficients.
`set`	Set WPTREE object field contents.
`split`	Split (decompose) the data of a terminal node.
`tlabels`	Labels for the nodes of a wavelet packet tree.
`write`	Write values in WPTREE object fields.

Proper Methods of WPTREE Class.

Method	Description
`bestlevt`	Best level of a wavelet packet tree.
`besttree`	Best wavelet packet tree.

Method	Description
`entrupd`	Entropy update (wavelet packet tree).
`wp2wtree`	Extract wavelet tree from wavelet packet tree.
`wpcoef`	Wavelet packet coefficients.
`wpcutree`	Cut wavelet packet tree.
`wpjoin`	Recompose wavelet packet.
`wpplotcf`	Plot wavelet packets colored coefficients.
`wprcoef`	Reconstruct wavelet packet coefficients.
`wprec`	Wavelet packet reconstruction 1-D.
`wprec2`	Wavelet packet reconstruction 2-D.
`wpsplt`	Split (decompose) wavelet packet.
`wpthcoef`	Wavelet packet coefficients thresholding.
`wpviewcf`	Plot wavelet packets colored coefficients.

Advanced Use of Objects

The following sections explain how to extend the toolbox with new objects through four examples.

Example 1: Building a Wavelet Tree Object (WTREE)

This example creates a new class of objects: **WTREE**.

Starting from the class DTREE and overloading the methods `split` and `merge`, we define a wavelet tree class.

To plot a WTREE, the DTREE `plot` method is used.

You can have a look at a one-dimensional example in the `ex1_wt` file and at a two-dimensional example in the `ex2_wt` file located in the `toolbox/wavelet/wavedemo` folder. These examples can be used directly, but they are also useful to learn how to build new object-oriented programming functions.

The definition of the new class is described below.

Class **WTREE** (parent class: **DTREE**)

Fields

`dtree`	Parent object
`dwtMode`	DWT extension mode
`wavInfo`	Structure (wavelet information)

wavInfo Structure information

`wavName`	Wavelet Name
`Lo_D`	Low Decomposition filter
`Hi_D`	High Decomposition filter
`Lo_R`	Low Reconstruction filter
`Hi_R`	High Reconstruction filter

Methods

`wtree`	Constructor for the class WTREE.
`merge`	Merge (recompose) the data of a node.
`split`	Split (decompose) the data of a terminal node.

Example 2: Building a Right Wavelet Tree Object (RWVTREE)

This example creates a new class of objects: **RWVTREE**.

We define a right wavelet tree class starting from the class WTREE and overloading the methods `split`, `merge`, and `plot` (inherited from DTREE).

The `plot` method shows how to add **Node Labels**.

You can have a look at a one-dimensional example in the `ex1_rwvt` file and at a two-dimensional example in the `ex2_rwvt` file located in the `toolbox/wavelet/wavedemo` folder. These programs can be used directly, but they are also useful to learn how to build new object-oriented programming functions.

The definition of the new class is described below.

Class **RWVTREE** (parent class: **WTREE**)

Fields

`dummy`	Not used
`wtree`	Parent object

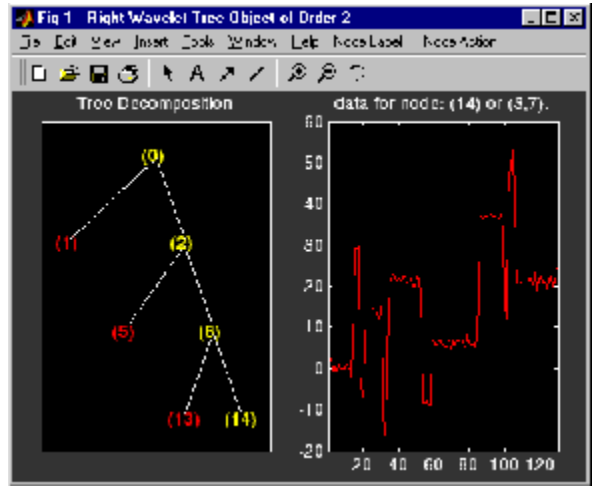
Methods

`rwvtree`	Constructor for the class RWVTREE.
`merge`	Merge (recompose) the data of a node.
`plot`	Plot RWVTREE object.
`split`	Split (decompose) the data of a terminal node.

Running This Example

The following figure is obtained using the example `ex1_rwvt` and clicking the node 14.

The approximations are labeled in yellow and the details are labeled in red. The last nodes cannot be split.



Example 3: Building a Wavelet Tree Object (WVTREE)

This example creates a new class of objects: **WVTREE**.

We define a wavelet tree class starting from the class WTREE and overloading the methods `get`, `plot`, and `recons` (all inherited from DTREE).

The `split` and `merge` methods of the class WTREE are used.

The `plot` method shows how to add **Node Labels** and **Node Actions**.

You can have a look at a one-dimensional example in the `ex1_wvt` file and at a two-dimensional example in the `ex2_wvt` file located in the `toolbox/wavelet/wavedemo` folder. These programs can be used directly, but they are also useful to learn how to build new object-oriented programming functions.

The definition of the new class is described below.

Class **WVTREE** (parent class: **WTREE**)

Fields

`dummy`	Not used
`wtree`	Parent object

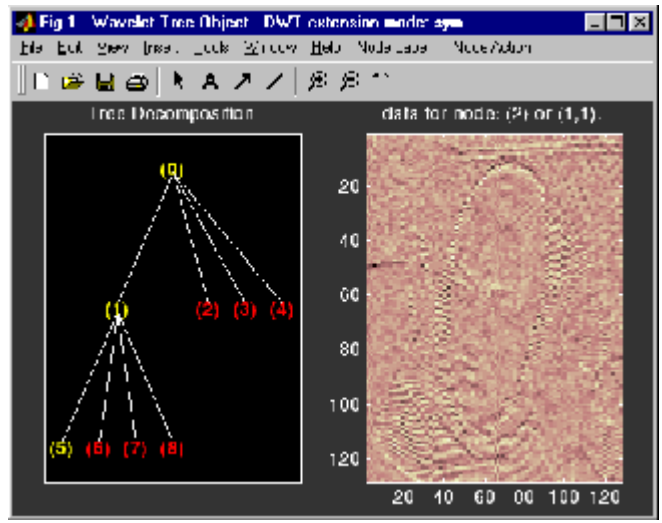
Methods

`wvtree`	Constructor for the class WVTREE.
`get`	Get WVTREE object field contents.
`plot`	Plot WVTREE object.
`recons`	Reconstruct node coefficients.

Running This Example

The following figure is obtained using the example `ex2_wvt` and clicking the node 2.

The approximations are labeled in yellow and the details are labeled in red. The last nodes cannot be split. The title of the figure contains the DWT extension mode used (`'sym'` in the present example).



Example 4: Building a Wavelet Tree Object (EDWTTREE)

This example creates a new class of objects: **EDWTTREE**.

We define an ε-DWT tree class starting from the class DTREE and overloading the methods `merge`, `plot`, `recons`, and `split`.

For more information on the ε-DWT, see the section "ε -Decimated DWT" on page 6-45.

The `plot` method shows how to add **Node Labels**, **Node Actions**, and **Tree Actions**.

You can have a look at the example in the `ex1_edwt` file located in the `toolbox/wavelet/wavedemo` folder. This program can be used directly, but it is also useful to learn how to build new object-oriented programming functions.

The definition of the new class is described below.

Class **EDWTTREE** (parent class: **DTREE**)

Fields

`dtree`	Parent object
`dwtMode`	DWT extension mode
`wavInfo`	Structure (wavelet information)

Fields Description

`wavInfo`

`wavName`	Wavelet Name
`Lo_D`	Low Decomposition filter
`Hi_D`	High Decomposition filter
`Lo_R`	Low Reconstruction filter
`Hi_R`	High Reconstruction filter

Methods

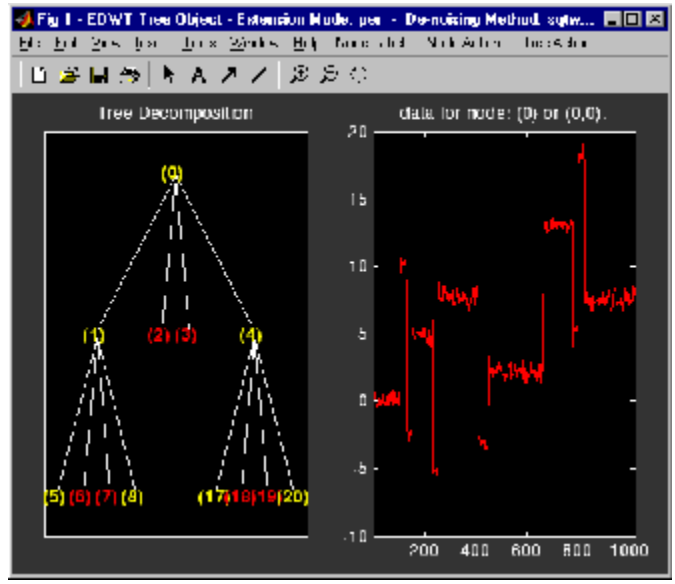
`edwttree`	Constructor for the class *EDWTTREE*.
`merge`	Merge (recompose) the data of a node.
`plot`	Plot *EDWTTREE* object.
`recons`	Reconstruct node coefficients.
`split`	Split (decompose) the data of a terminal node.

Running This Example

The following figure is obtained using the example `ex1_edwt`, selecting the **De-noise** option in the **Tree Action** menu and clicking the node 0.

The approximations are labeled in yellow and the details are labeled in red. The last nodes cannot be split.

The title of the figure contains the DWT extension mode used (`'sym'` in the present example) and the name of the de-noising method.



Index

D

E

F

G

Q

R

S

T

Z