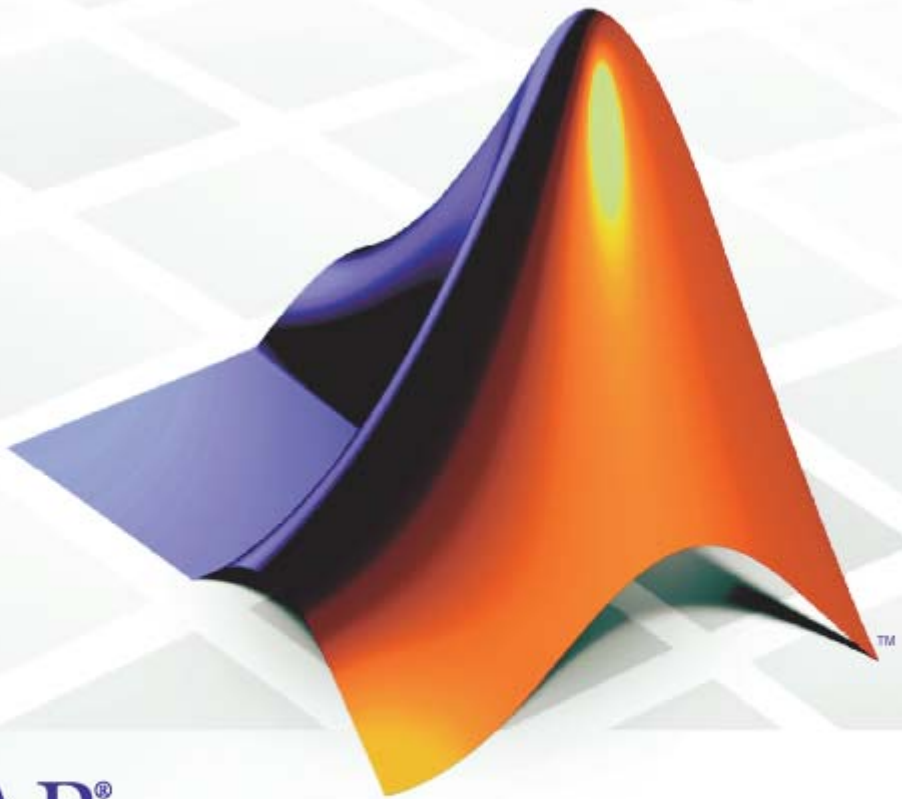


# Wavelet Toolbox™ 4

## User's Guide

*Michel Misiti  
Yves Misiti  
Georges Oppenheim  
Jean-Michel Poggi*



**MATLAB®**

## How to Contact The MathWorks:



[www.mathworks.com](http://www.mathworks.com)  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab)  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html)

Web  
Newsgroup  
Technical support



[suggest@mathworks.com](mailto:suggest@mathworks.com)  
[bugs@mathworks.com](mailto:bugs@mathworks.com)  
[doc@mathworks.com](mailto:doc@mathworks.com)  
[service@mathworks.com](mailto:service@mathworks.com)  
[info@mathworks.com](mailto:info@mathworks.com)

Product enhancement suggestions  
Bug reports  
Documentation error reports  
Order status, license renewals, passcodes  
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*

© COPYRIGHT 1997–2009 by The MathWorks, Inc.

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

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

### **Trademarks**

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://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**

The MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://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)



---

## 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 Kerkycharian (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<sup>®</sup> 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



## Wavelets: A New Tool for Signal Analysis

1

<b>Product Overview</b> .....	1-2
<b>Background Reading</b> .....	1-4
<b>Installing Wavelet Toolbox™ Software</b> .....	1-5
System Recommendations .....	1-5
Platform-Specific Details .....	1-5
<b>Wavelet Applications</b> .....	1-7
Scale Aspects .....	1-7
Time Aspects .....	1-7
Wavelet Decomposition as a Whole .....	1-8
<b>Fourier Analysis</b> .....	1-9
<b>Short-Time Fourier Analysis</b> .....	1-10
<b>Wavelet Analysis</b> .....	1-11
What Can Wavelet Analysis Do? .....	1-12
<b>What Is Wavelet Analysis?</b> .....	1-14
Number of Dimensions .....	1-14
<b>Continuous Wavelet Transform</b> .....	1-15
Scaling .....	1-16
Shifting .....	1-17
Five Easy Steps to a Continuous Wavelet Transform .....	1-17
Scale and Frequency .....	1-20
Scale of Nature .....	1-21
What's Continuous About the Continuous Wavelet Transform? .....	1-22

<b>Discrete Wavelet Transform</b> .....	<b>1-24</b>
One-Stage Filtering: Approximations and Details .....	<b>1-24</b>
Multiple-Level Decomposition .....	<b>1-27</b>
<b>Wavelet Reconstruction</b> .....	<b>1-29</b>
Reconstruction Filters .....	<b>1-30</b>
Reconstructing Approximations and Details .....	<b>1-30</b>
Relationship of Filters to Wavelet Shapes .....	<b>1-32</b>
Multistep Decomposition and Reconstruction .....	<b>1-35</b>
<b>Wavelet Packet Analysis</b> .....	<b>1-36</b>
<b>History of Wavelets</b> .....	<b>1-38</b>
<b>Introduction to the Wavelet Families</b> .....	<b>1-39</b>
Haar .....	<b>1-41</b>
Daubechies .....	<b>1-42</b>
Biorthogonal .....	<b>1-43</b>
Coiflets .....	<b>1-45</b>
Symlets .....	<b>1-45</b>
Morlet .....	<b>1-46</b>
Mexican Hat .....	<b>1-46</b>
Meyer .....	<b>1-47</b>
Other Real Wavelets .....	<b>1-47</b>
Complex Wavelets .....	<b>1-47</b>

## Using Wavelets

# 2

<b>Introduction to Wavelet Toolbox™ GUIs and Functions</b> ..	<b>2-3</b>
<b>One-Dimensional Continuous Wavelet Analysis</b> .....	<b>2-4</b>
Continuous Analysis Using the Command Line .....	<b>2-5</b>
Continuous Analysis Using the Graphical Interface .....	<b>2-8</b>
Importing and Exporting Information from the Graphical Interface .....	<b>2-17</b>

<b>One-Dimensional Complex Continuous Wavelet Analysis</b>	<b>2-20</b>
Complex Continuous Analysis Using the Command Line . . . .	<b>2-21</b>
Complex Continuous Analysis Using the Graphical Interface	<b>2-23</b>
Importing and Exporting Information from the Graphical Interface . . . . .	<b>2-28</b>
<b>One-Dimensional Discrete Wavelet Analysis</b> . . . . .	<b>2-29</b>
One-Dimensional Analysis Using the Command Line . . . . .	<b>2-31</b>
One-Dimensional Analysis Using the Graphical Interface . . .	<b>2-40</b>
Importing and Exporting Information from the Graphical Interface . . . . .	<b>2-58</b>
<b>Two-Dimensional Discrete Wavelet Analysis</b> . . . . .	<b>2-66</b>
Two-Dimensional Analysis Using the Command Line . . . . .	<b>2-69</b>
Two-Dimensional Analysis Using the Graphical Interface . . .	<b>2-76</b>
Importing and Exporting Information from the Graphical Interface . . . . .	<b>2-88</b>
<b>Wavelets: Working with Images</b> . . . . .	<b>2-97</b>
Understanding Images in the MATLAB® Environment . . . . .	<b>2-97</b>
Indexed Images . . . . .	<b>2-97</b>
Wavelet Decomposition of Indexed Images . . . . .	<b>2-98</b>
RGB (Truecolor) Images . . . . .	<b>2-99</b>
Wavelet Decomposition of Truecolor Images . . . . .	<b>2-100</b>
Other Images . . . . .	<b>2-100</b>
Image Conversion . . . . .	<b>2-100</b>
<b>One-Dimensional Discrete Stationary Wavelet Analysis</b>	<b>2-104</b>
One-Dimensional Analysis Using the Command Line . . . . .	<b>2-105</b>
One-Dimensional Analysis for De-Noising Using the Graphical Interface . . . . .	<b>2-114</b>
Importing and Exporting from the GUI . . . . .	<b>2-120</b>
<b>Two-Dimensional Discrete Stationary Wavelet Analysis</b>	<b>2-122</b>
Two-Dimensional Analysis Using the Command Line . . . . .	<b>2-123</b>
Two-Dimensional Analysis for De-Noising Using the Graphical Interface . . . . .	<b>2-132</b>
Importing and Exporting Information from the Graphical Interface . . . . .	<b>2-138</b>

<b>One-Dimensional Wavelet Regression Estimation</b> . . . . .	<b>2-140</b>
One-Dimensional Estimation Using the GUI for Equally Spaced Observations (Fixed Design) . . . . .	<b>2-140</b>
One-Dimensional Estimation Using the GUI for Randomly Spaced Observations (Stochastic Design) . . . . .	<b>2-146</b>
Importing and Exporting Information from the Graphical Interface . . . . .	<b>2-148</b>
 <b>One-Dimensional Wavelet Density Estimation</b> . . . . .	<b>2-150</b>
One-Dimensional Estimation Using the Graphical Interface	<b>2-150</b>
Importing and Exporting Information from the Graphical Interface . . . . .	<b>2-156</b>
 <b>One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients</b> . . . . .	<b>2-158</b>
One-Dimensional Local Thresholding for De-Noising Using the Graphical Interface . . . . .	<b>2-159</b>
Importing and Exporting Information from the Graphical Interface . . . . .	<b>2-166</b>
 <b>One-Dimensional Selection of Wavelet Coefficients Using the Graphical Interface</b> . . . . .	<b>2-168</b>
 <b>Two-Dimensional Selection of Wavelet Coefficients Using the Graphical Interface</b> . . . . .	<b>2-178</b>
 <b>One-Dimensional Extension</b> . . . . .	<b>2-186</b>
One-Dimensional Extension Using the Command Line . . . . .	<b>2-186</b>
One-Dimensional Extension Using the Graphical Interface .	<b>2-186</b>
Importing and Exporting Information from the Graphical Interface . . . . .	<b>2-194</b>
 <b>Two-Dimensional Extension</b> . . . . .	<b>2-195</b>
Two-Dimensional Extension Using the Command Line . . . . .	<b>2-195</b>
Two-Dimensional Extension Using the Graphical Interface .	<b>2-195</b>
Importing and Exporting Information from the Graphical Interface . . . . .	<b>2-198</b>

<b>Image Fusion</b> .....	<b>2-199</b>
Image Fusion Using the Command Line .....	<b>2-199</b>
Image Fusion Using the Graphical Interface .....	<b>2-202</b>
<b>One-Dimensional Fractional Brownian Motion Synthesis</b>	<b>2-208</b>
Fractional Brownian Motion Synthesis Using the Command Line .....	<b>2-208</b>
Fractional Brownian Motion Synthesis Using the Graphical Interface .....	<b>2-210</b>
Saving the Synthesized Signal .....	<b>2-214</b>
<b>New Wavelet for CWT</b> .....	<b>2-216</b>
New Wavelet for CWT Using the Command Line .....	<b>2-216</b>
New Wavelet for CWT Using the Graphical Interface .....	<b>2-218</b>
Saving the New Wavelet .....	<b>2-227</b>
<b>Multivariate Wavelet De-Noising</b> .....	<b>2-228</b>
Multivariate Wavelet De-Noising Using the Command Line	<b>2-228</b>
Multivariate Wavelet De-Noising Using the Graphical Interface .....	<b>2-234</b>
Importing and Exporting from the GUI .....	<b>2-244</b>
<b>Multiscale Principal Components Analysis</b> .....	<b>2-247</b>
Multiscale Principal Components Analysis Using the Command Line .....	<b>2-247</b>
Multiscale Principal Components Analysis Using the Graphical Interface .....	<b>2-251</b>
Importing and Exporting from the GUI .....	<b>2-259</b>
<b>One-Dimensional Multisignal Analysis</b> .....	<b>2-261</b>
One-Dimensional Multisignal Analysis Using the Command Line .....	<b>2-262</b>
One-Dimensional Multisignal Analysis Using the Graphical Interface .....	<b>2-271</b>
Importing and Exporting Information from the Graphical Interface .....	<b>2-308</b>

<b>Two-Dimensional True Compression</b> .....	<b>2-317</b>
Two-Dimensional True Compression Using the Command Line .....	<b>2-317</b>
Two-Dimensional True Compression Using the Graphical Interface .....	<b>2-325</b>
Importing and Exporting from the GUI .....	<b>2-338</b>

## Wavelet Applications

---

# 3

<b>Introduction to Wavelet Analysis</b> .....	<b>3-2</b>
<b>Detecting Discontinuities and Breakdown Points I</b> .....	<b>3-3</b>
Discussion .....	<b>3-4</b>
<b>Detecting Discontinuities and Breakdown Points II</b> .....	<b>3-6</b>
Discussion .....	<b>3-7</b>
<b>Detecting Long-Term Evolution</b> .....	<b>3-8</b>
Discussion .....	<b>3-9</b>
<b>Detecting Self-Similarity</b> .....	<b>3-10</b>
Wavelet Coefficients and Self-Similarity .....	<b>3-10</b>
Discussion .....	<b>3-11</b>
<b>Identifying Pure Frequencies</b> .....	<b>3-12</b>
Discussion .....	<b>3-12</b>
<b>Suppressing Signals</b> .....	<b>3-15</b>
Discussion .....	<b>3-16</b>
<b>De-Noising Signals</b> .....	<b>3-18</b>
Discussion .....	<b>3-18</b>
<b>De-Noising Images</b> .....	<b>3-21</b>
Discussion .....	<b>3-22</b>

<b>Compressing Images</b> .....	<b>3-26</b>
Discussion .....	<b>3-27</b>
<b>Fast Multiplication of Large Matrices</b> .....	<b>3-28</b>

## Wavelets in Action: Examples and Case Studies

# 4

<b>Illustrated Examples</b> .....	<b>4-2</b>
Advice to the Reader .....	<b>4-5</b>
Example 1: A Sum of Sines .....	<b>4-8</b>
Example 2: A Frequency Breakdown .....	<b>4-10</b>
Example 3: Uniform White Noise .....	<b>4-12</b>
Example 4: Colored AR(3) Noise .....	<b>4-14</b>
Example 5: Polynomial + White Noise .....	<b>4-16</b>
Example 6: A Step Signal .....	<b>4-18</b>
Example 7: Two Proximal Discontinuities .....	<b>4-20</b>
Example 8: A Second-Derivative Discontinuity .....	<b>4-22</b>
Example 9: A Ramp + White Noise .....	<b>4-24</b>
Example 10: A Ramp + Colored Noise .....	<b>4-26</b>
Example 11: A Sine + White Noise .....	<b>4-28</b>
Example 12: A Triangle + A Sine .....	<b>4-30</b>
Example 13: A Triangle + A Sine + Noise .....	<b>4-32</b>
Example 14: A Real Electricity Consumption Signal .....	<b>4-34</b>
<b>Case Study: An Electrical Signal</b> .....	<b>4-36</b>
Data and the External Information .....	<b>4-36</b>
Analysis of the Midday Period .....	<b>4-38</b>
Analysis of the End of the Night Period .....	<b>4-39</b>
Suggestions for Further Analysis .....	<b>4-42</b>

## Using Wavelet Packets

---

### 5

<b>About Wavelet Packet Analysis</b> .....	<b>5-2</b>
<b>One-Dimensional Wavelet Packet Analysis</b> .....	<b>5-7</b>
Compressing a Signal Using Wavelet Packets .....	<b>5-13</b>
De-Noising a Signal Using Wavelet Packets .....	<b>5-16</b>
<b>Two-Dimensional Wavelet Packet Analysis</b> .....	<b>5-23</b>
Compressing an Image Using Wavelet Packets .....	<b>5-28</b>
<b>Importing and Exporting from Graphical Tools</b> .....	<b>5-32</b>
Saving Information to Disk .....	<b>5-32</b>
Loading Information into the Graphical Tools .....	<b>5-36</b>

## Advanced Concepts

---

### 6

<b>Mathematical Conventions</b> .....	<b>6-2</b>
<b>General Concepts</b> .....	<b>6-5</b>
Wavelets: A New Tool for Signal Analysis .....	<b>6-5</b>
Wavelet Decomposition: A Hierarchical Organization .....	<b>6-5</b>
Finer and Coarser Resolutions .....	<b>6-6</b>
Wavelet Shapes .....	<b>6-6</b>
Wavelets and Associated Families .....	<b>6-8</b>
Wavelet Transforms: Continuous and Discrete .....	<b>6-12</b>
Local and Global Analysis .....	<b>6-14</b>
Synthesis: An Inverse Transform .....	<b>6-15</b>
Details and Approximations .....	<b>6-15</b>



<b>Fast Wavelet Transform (FWT) Algorithm</b> .....	<b>6-19</b>
Filters Used to Calculate the DWT and IDWT .....	<b>6-19</b>
Algorithms .....	<b>6-23</b>
Why Does Such an Algorithm Exist? .....	<b>6-28</b>
One-Dimensional Wavelet Capabilities .....	<b>6-32</b>
Two-Dimensional Wavelet Capabilities .....	<b>6-33</b>
<b>Dealing with Border Distortion</b> .....	<b>6-35</b>
Signal Extensions: Zero-Padding, Symmetrization, and Smooth Padding .....	<b>6-35</b>
<b>Discrete Stationary Wavelet Transform (SWT)</b> .....	<b>6-44</b>
e-Decimated DWT .....	<b>6-44</b>
How to Calculate the e-Decimated DWT: SWT .....	<b>6-45</b>
Inverse Discrete Stationary Wavelet Transform (ISWT) ....	<b>6-50</b>
More About SWT .....	<b>6-50</b>
<b>Lifting Method for Constructing Wavelets</b> .....	<b>6-51</b>
Lifting Background .....	<b>6-51</b>
Lifting Functions .....	<b>6-54</b>
<b>Frequently Asked Questions</b> .....	<b>6-61</b>
Continuous or Discrete Analysis? .....	<b>6-61</b>
Why Are Wavelets Useful for Space-Saving Coding? .....	<b>6-61</b>
<b>Wavelet Families: Additional Discussion</b> .....	<b>6-71</b>
Daubechies Wavelets: dbN .....	<b>6-72</b>
Symlet Wavelets: symN .....	<b>6-74</b>
Coiflet Wavelets: coifN .....	<b>6-75</b>
Biorthogonal Wavelet Pairs: biorNr.Nd .....	<b>6-76</b>
Meyer Wavelet: meyr .....	<b>6-78</b>
Battle-Lemarie Wavelets .....	<b>6-79</b>
Mexican Hat Wavelet: mexh .....	<b>6-80</b>
Morlet Wavelet: morl .....	<b>6-81</b>
Additional Real Wavelets .....	<b>6-82</b>
Complex Wavelets .....	<b>6-84</b>

Summary of Wavelet Families and Associated Properties (Part 1) .....	6-88
Summary of Wavelet Families and Associated Properties (Part 2) .....	6-90
<b>Wavelet Applications: More Detail</b> .....	<b>6-92</b>
Suppressing Signals .....	6-92
Splitting Signal Components .....	6-95
Noise Processing .....	6-95
De-Noising .....	6-97
Data Compression .....	6-110
Function Estimation: Density and Regression .....	6-114
Available Methods for De-Noising, Estimation, and Compression Using GUI Tools .....	6-122
True Compression for Images .....	6-130
<b>Wavelet Packets</b> .....	<b>6-136</b>
From Wavelets to Wavelet Packets: Decomposing the Details	6-136
Wavelet Packets in Action: An Introduction .....	6-137
Building Wavelet Packets .....	6-141
Wavelet Packet Atoms .....	6-143
Organizing the Wavelet Packets .....	6-146
Choosing the Optimal Decomposition .....	6-147
Some Interesting Subtrees .....	6-152
Wavelet Packets 2-D Decomposition Structure .....	6-153
Wavelet Packets for Compression and De-Noising .....	6-154
<b>References</b> .....	<b>6-155</b>

<b>Preparing to Add a New Wavelet Family</b> .....	<b>7-2</b>
Choose the Wavelet Family Full Name .....	<b>7-2</b>
Choose the Wavelet Family Short Name .....	<b>7-3</b>
Determine the Wavelet Type .....	<b>7-3</b>
Define the Orders of Wavelets Within the Given Family .....	<b>7-4</b>
Build a MAT-File or M-File .....	<b>7-4</b>
Define the Effective Support .....	<b>7-6</b>
<b>Adding a New Wavelet Family</b> .....	<b>7-8</b>
Example 1 .....	<b>7-8</b>
Example 2 .....	<b>7-12</b>
<b>After Adding a New Wavelet Family</b> .....	<b>7-16</b>

---

<b>General Wavelet Functions</b> .....	<b>18</b>
<b>Wavelet Families</b> .....	<b>19</b>
<b>1-D Continuous Wavelets</b> .....	<b>20</b>
<b>1-D Discrete Wavelets</b> .....	<b>21</b>
<b>2-D Discrete Wavelets</b> .....	<b>22</b>
<b>Wavelet Packet Algorithms</b> .....	<b>23</b>
<b>Discrete Stationary Wavelet Transform Algorithms</b> .....	<b>24</b>
<b>Lifting Wavelet Transforms</b> .....	<b>25</b>
<b>De-Noising and Compression</b> .....	<b>26</b>
<b>1-D Multisignal Wavelet Analysis</b> .....	<b>27</b>
<b>True Compression for Images</b> .....	<b>28</b>
<b>Other Wavelet Applications</b> .....	<b>29</b>
<b>Tree Management Utilities</b> .....	<b>30</b>
<b>General Utilities and Demos</b> .....	<b>32</b>

GUI Reference

A

<b>General Features</b> .....	<b>A-2</b>
Color Coding .....	<b>A-2</b>
Connection of Plots .....	<b>A-3</b>
Using the Mouse .....	<b>A-4</b>
Controlling the Colormap .....	<b>A-6</b>
Using Menus .....	<b>A-9</b>
Using the View Axes Button .....	<b>A-13</b>
Using the Interval-Dependent Threshold Settings Tool .....	<b>A-16</b>
<b>Continuous Wavelet Tool Features</b> .....	<b>A-18</b>
<b>Wavelet 1-D Tool Features</b> .....	<b>A-19</b>
Tree Mode .....	<b>A-19</b>
More Display Options .....	<b>A-19</b>
<b>Wavelet 2-D Tool Features</b> .....	<b>A-21</b>
<b>Wavelet Packet Tool Features (1-D and 2-D)</b> .....	<b>A-22</b>
Node Action Functionality .....	<b>A-23</b>
<b>Wavelet Display Tool</b> .....	<b>A-27</b>
<b>Wavelet Packet Display Tool</b> .....	<b>A-28</b>

<b>Introduction to Object-Oriented Features</b> .....	<b>B-2</b>
<b>Short Description of Objects in the Wavelet Toolbox™</b>	
<b>Software</b> .....	<b>B-3</b>
<b>Simple Use of Objects Through Four Examples</b> .....	<b>B-5</b>
Example 1: plot and wvviewcf .....	<b>B-5</b>
Example 2: drawtree and readtree .....	<b>B-8</b>
Example 3: A Funny One .....	<b>B-11</b>
Example 4: Thresholding Wavelet Packets .....	<b>B-12</b>
<b>Detailed Description of Objects in the Wavelet Toolbox™</b>	
<b>Software</b> .....	<b>B-16</b>
WTBO Object .....	<b>B-16</b>
NTREE Object .....	<b>B-17</b>
DTREE Object .....	<b>B-18</b>
WPTREE Object .....	<b>B-20</b>
<b>Advanced Use of Objects</b> .....	<b>B-23</b>
Example 1: Building a Wavelet Tree Object (WTREE) .....	<b>B-23</b>
Example 2: Building a Right Wavelet Tree Object (RWVTREE) .....	<b>B-24</b>
Example 3: Building a Wavelet Tree Object (WVTREE) .....	<b>B-26</b>
Example 4: Building a Wavelet Tree Object (EDWTTREE) ..	<b>B-27</b>



# Wavelets: A New Tool for Signal Analysis

---

- Product Overview (p. 1-2)
- Background Reading (p. 1-4)
- Installing Wavelet Toolbox™ Software (p. 1-5)
- Wavelet Applications (p. 1-7)
- Fourier Analysis (p. 1-9)
- Short-Time Fourier Analysis (p. 1-10)
- Wavelet Analysis (p. 1-11)
- What Is Wavelet Analysis? (p. 1-14)
- Continuous Wavelet Transform (p. 1-15)
- Discrete Wavelet Transform (p. 1-24)
- Wavelet Reconstruction (p. 1-29)
- Wavelet Packet Analysis (p. 1-36)
- History of Wavelets (p. 1-38)
- Introduction to the Wavelet Families (p. 1-39)



## Product Overview

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

Wavelet Toolbox™ software is a collection of functions built on the MATLAB® technical computing environment. It provides tools for the analysis and synthesis of signals and images, and tools for statistical applications, using wavelets and wavelet packets within the framework of MATLAB.

The MathWorks™ provides several products that are relevant to the kinds of tasks you can perform with the toolbox. For more information about any of these products, see the products section of The MathWorks Web site.

Wavelet Toolbox software provides two categories of tools:

- Command-line functions
- Graphical interactive tools

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

```
type function_name
```

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

```
help function_name
```

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

```
help wavelet
```

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

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

```
wavemenu
```

---

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

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

---

## Background Reading

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

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

Excellent supplementary texts provide complementary treatments of wavelet theory and practice (see “References” on page 6-155). For instance:

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

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

## Installing Wavelet Toolbox™ Software

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

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

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

### System Recommendations

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

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

### Platform-Specific Details

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

#### Windows Fonts

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

### Fonts for Non-Windows Platforms

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

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

### Mouse Compatibility

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

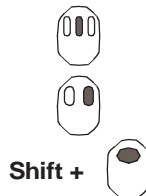
#### Left Mouse Button

Make selections.  
Activate controls.



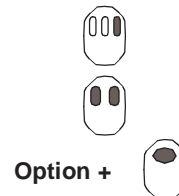
#### Middle Mouse Button

Display cross-hairs to  
show position-dependent  
information.



#### Right Mouse Button

Translate plots up and  
down, and left and  
right.



---

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

---

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

## Wavelet Applications

Wavelets have scale aspects and time aspects, consequently every application has scale and time aspects. To clarify them we try to untangle the aspects somewhat arbitrarily.

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

### Scale Aspects

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

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

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

Several domains use the wavelet techniques of regularity study:

- Biology for cell membrane recognition, to distinguish the normal from the pathological membranes
- Metallurgy for the characterization of rough surfaces
- Finance (which is more surprising), for detecting the properties of quick variation of values
- In Internet traffic description, for designing the services size

### Time Aspects

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

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

As domain applications, we get:

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

## Wavelet Decomposition as a Whole

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

We find many published papers in oceanography and earth studies.

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

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

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

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

## Fourier Analysis

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



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

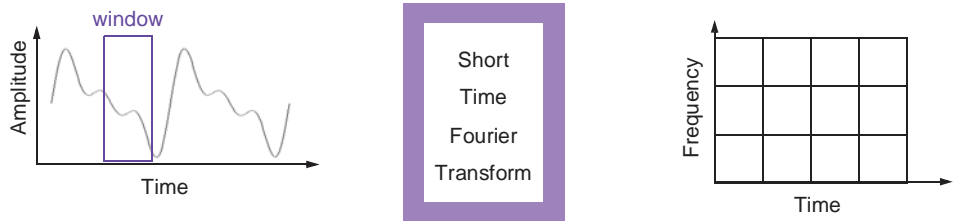
Fourier analysis has a serious drawback. In transforming to the frequency domain, time information is lost. When looking at a Fourier transform of a signal, it is impossible to tell *when* a particular event took place.

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



## Short-Time Fourier Analysis

In an effort to correct this deficiency, Dennis Gabor (1946) adapted the Fourier transform to analyze only a small section of the signal at a time — a technique called *windowing* the signal. Gabor’s adaptation, called the *Short-Time Fourier Transform* (STFT), maps a signal into a two-dimensional function of time and frequency.



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

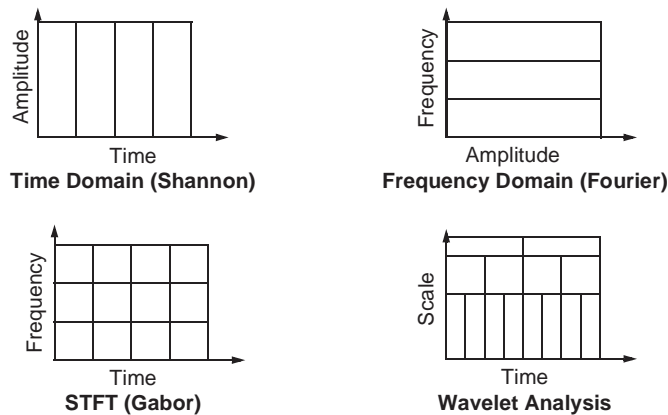
While the STFT compromise between time and frequency information can be useful, the drawback is that once you choose a particular size for the time window, that window is the same for all frequencies. Many signals require a more flexible approach — one where we can vary the window size to determine more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

## Wavelet Analysis

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



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

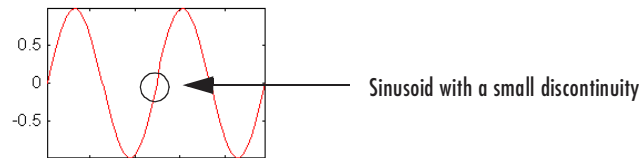


You may have noticed that wavelet analysis does not use a time-frequency region, but rather a time-*scale* region. For more information about the concept of scale and the link between scale and frequency, see “How to Connect Scale to Frequency?” on page 6-66.

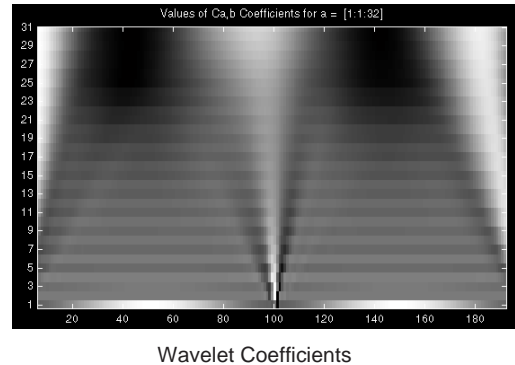
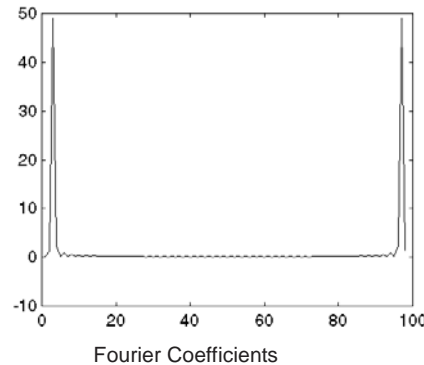
## What Can Wavelet Analysis Do?

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

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



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



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

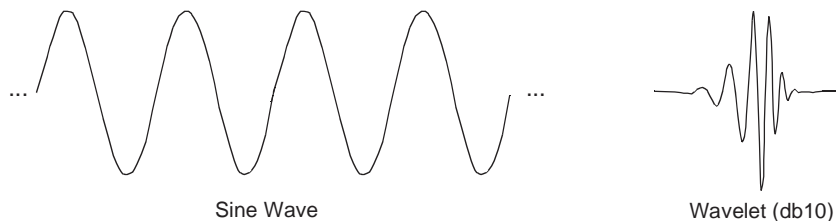
Indeed, in their brief history within the signal processing field, wavelets have already proven themselves to be an indispensable addition to the analyst's collection of tools and continue to enjoy a burgeoning popularity today.

## What Is Wavelet Analysis?

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

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

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



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

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

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

## Number of Dimensions

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

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

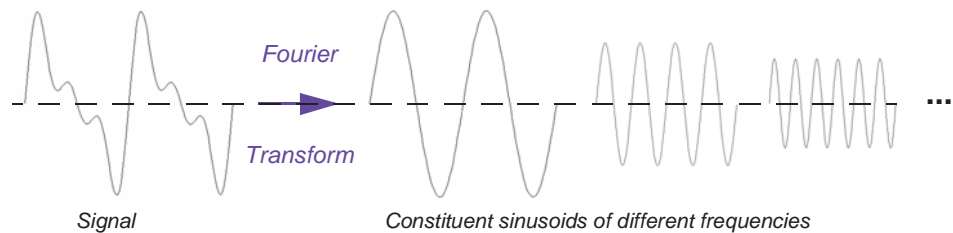
## Continuous Wavelet Transform

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

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

which is the sum over all time of the signal  $f(t)$  multiplied by a complex exponential. (Recall that a complex exponential can be broken down into real and imaginary sinusoidal components.)

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

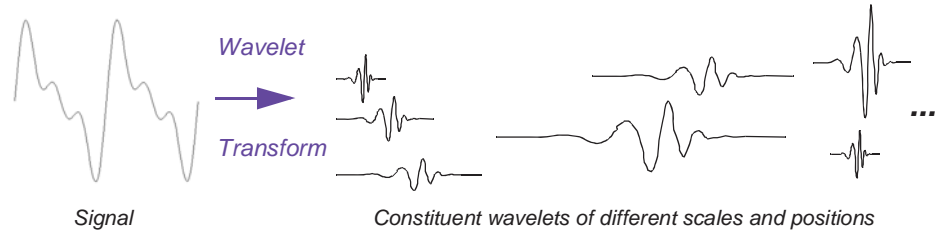


Similarly, the *continuous wavelet transform* (CWT) is defined as the sum over all time of the signal multiplied by scaled, shifted versions of the wavelet function  $\psi$ :

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

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

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

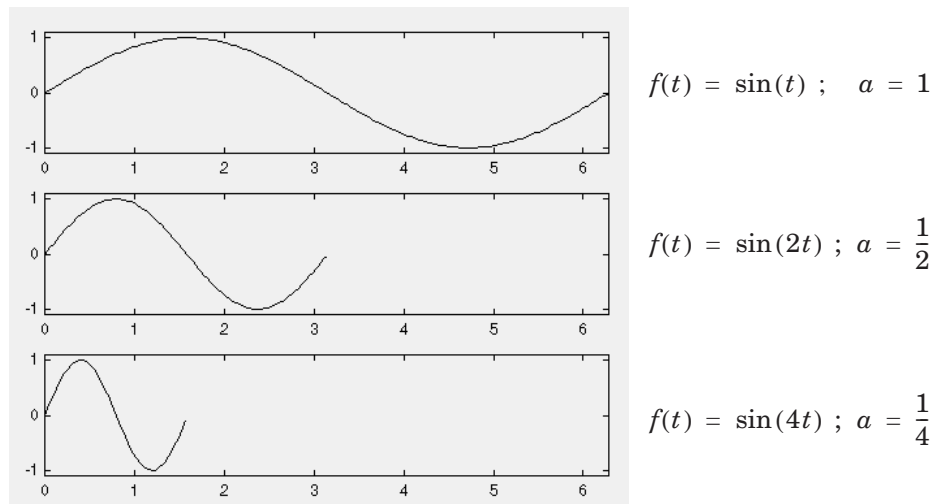


## Scaling

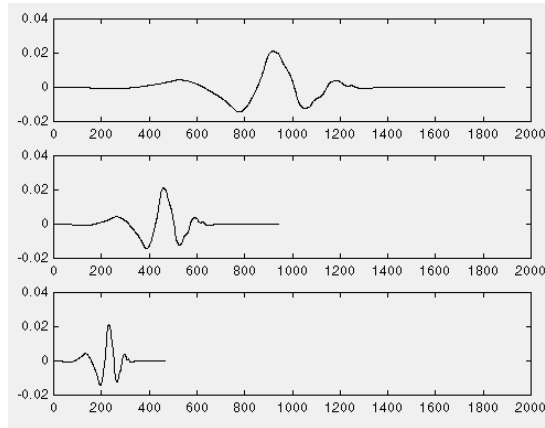
We've already alluded to the fact that wavelet analysis produces a time-scale view of a signal, and now we're talking about scaling and shifting wavelets. What exactly do we mean by *scale* in this context?

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

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



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



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

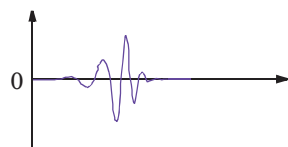
$$f(t) = \psi(2t) \quad ; \quad a = \frac{1}{2}$$

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

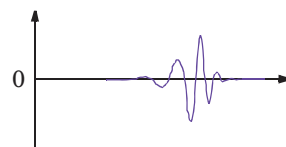
It is clear from the diagrams that, for a sinusoid  $\sin(\omega t)$ , the scale factor  $a$  is related (inversely) to the radian frequency  $\omega$ . Similarly, with wavelet analysis, the scale is related to the frequency of the signal. We’ll return to this topic later.

### Shifting

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



Wavelet function  
 $\psi(t)$



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

### Five Easy Steps to a Continuous Wavelet Transform

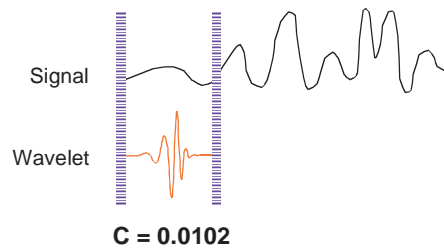
The continuous wavelet transform is the sum over all time of the signal multiplied by scaled, shifted versions of the wavelet. This process produces wavelet coefficients that are a function of scale and position.



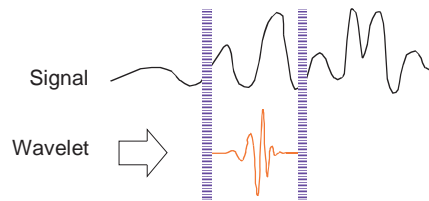
It's really a very simple process. In fact, here are the five steps of an easy recipe for creating a CWT:

- 1 Take a wavelet and compare it to a section at the start of the original signal.
- 2 Calculate a number,  $C$ , that represents how closely correlated the wavelet is with this section of the signal. The higher  $C$  is, the more the similarity. More precisely, if the signal energy and the wavelet energy are equal to one,  $C$  may be interpreted as a correlation coefficient.

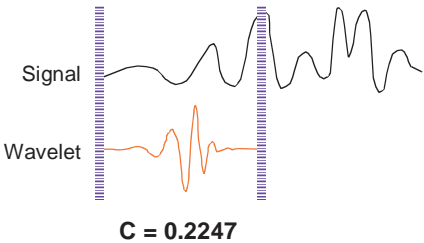
Note that the results will depend on the shape of the wavelet you choose.



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



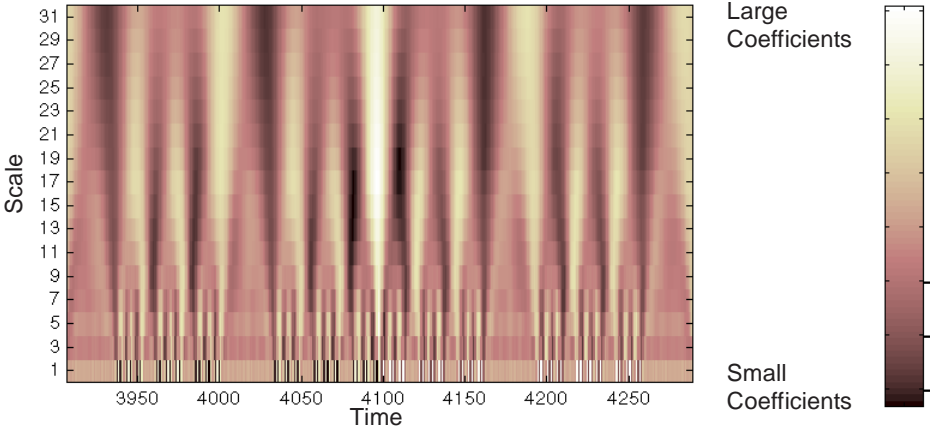
- 4 Scale (stretch) the wavelet and repeat steps 1 through 3.



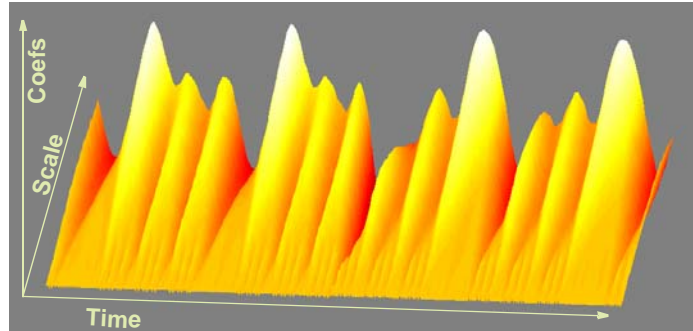
5 Repeat steps 1 through 4 for all scales.

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

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



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



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

## Scale and Frequency

Notice that the scales in the coefficients plot (shown as  $y$ -axis labels) run from 1 to 31. Recall that the higher scales correspond to the most “stretched” wavelets. The more stretched the wavelet, the longer the portion of the signal with which it is being compared, and thus the coarser the signal features being measured by the wavelet coefficients.



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

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

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

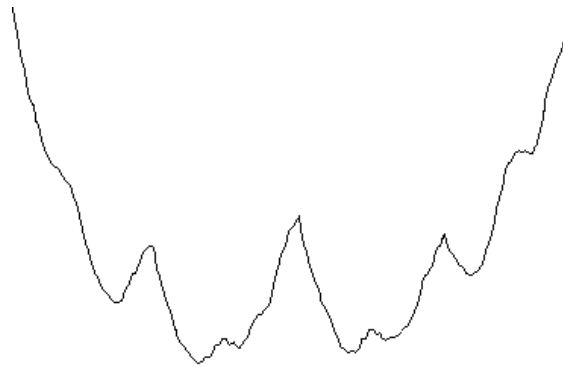
## Scale of Nature

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

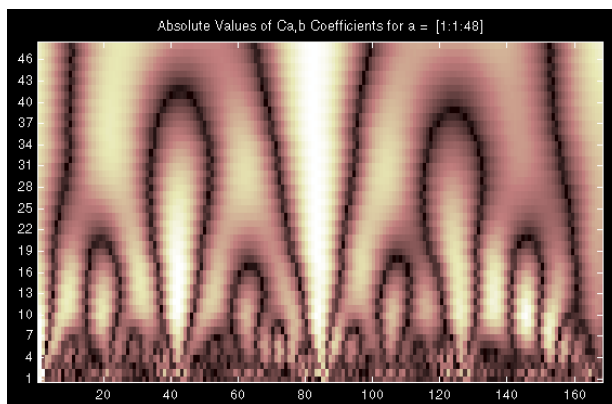
Not only is time-scale a different way to view data, it is a very natural way to view data deriving from a great number of natural phenomena.

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

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



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



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

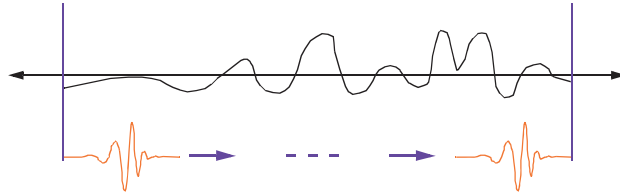
## What's Continuous About the Continuous Wavelet Transform?

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

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

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

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



## Discrete Wavelet Transform

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

It turns out, rather remarkably, that if we choose scales and positions based on powers of two — so-called *dyadic* scales and positions — then our analysis will be much more efficient and just as accurate. We obtain such an analysis from the *discrete wavelet transform* (DWT). For more information on DWT, see “Algorithms” on page 6-23.

An efficient way to implement this scheme using filters was developed in 1988 by Mallat (see [Mal89] in “References” on page 6-155). The Mallat algorithm is in fact a classical scheme known in the signal processing community as a *two-channel subband coder* (see page 1 of the book *Wavelets and Filter Banks*, by Strang and Nguyen [StrN96]).

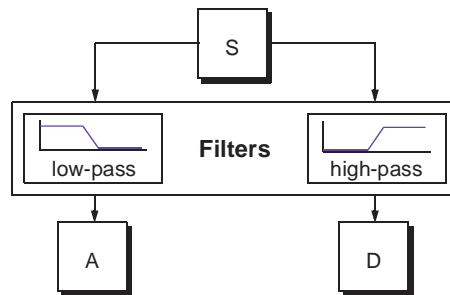
This very practical filtering algorithm yields a *fast wavelet transform* — a box into which a signal passes, and out of which wavelet coefficients quickly emerge. Let’s examine this in more depth.

### One-Stage Filtering: Approximations and Details

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

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

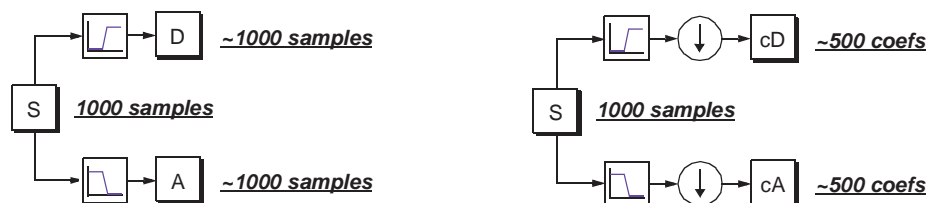
The filtering process, at its most basic level, looks like this.



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

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

These signals  $A$  and  $D$  are interesting, but we get 2000 values instead of the 1000 we had. There exists a more subtle way to perform the decomposition using wavelets. By looking carefully at the computation, we may keep only one point out of two in each of the two 2000-length samples to get the complete information. This is the notion of *downsampling*. We produce two sequences called  $cA$  and  $cD$ .

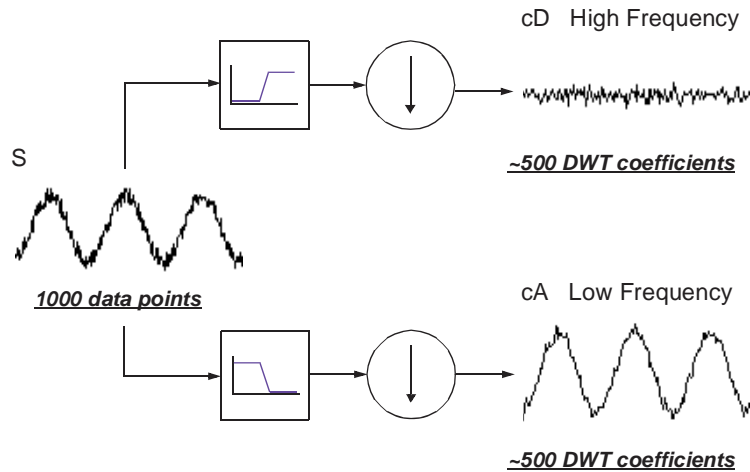


The process on the right, which includes downsampling, produces DWT coefficients.

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



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



The MATLAB<sup>®</sup> code needed to generate  $s$ ,  $cD$ , and  $cA$  is

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

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

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

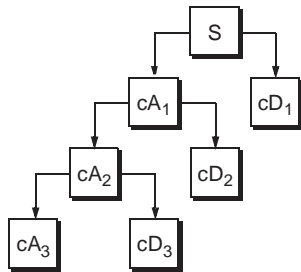
```
[length(cA) length(cD)]
```

```
ans =
    501    501
```

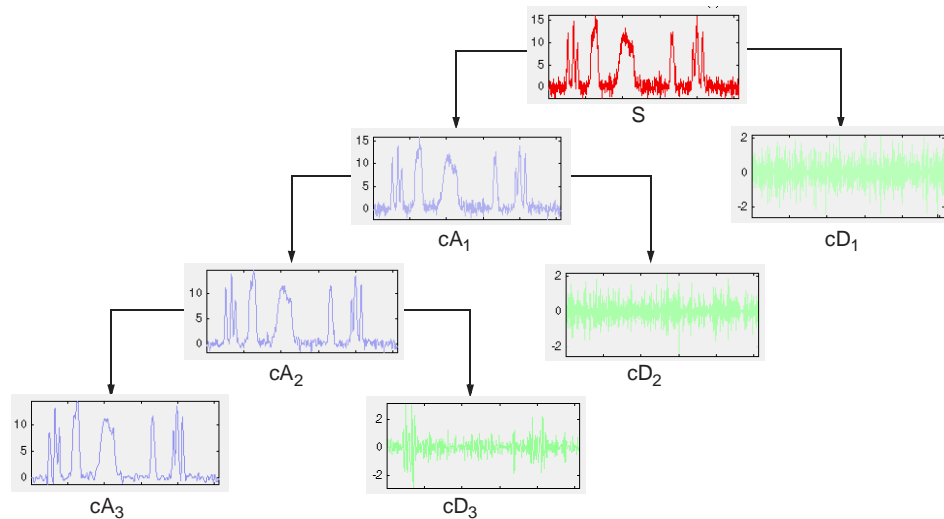
You may observe that the actual lengths of the detail and approximation coefficient vectors are slightly *more* than half the length of the original signal. This has to do with the filtering process, which is implemented by convolving the signal with a filter. The convolution “smears” the signal, introducing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

## Multiple-Level Decomposition

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



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



### Number of Levels

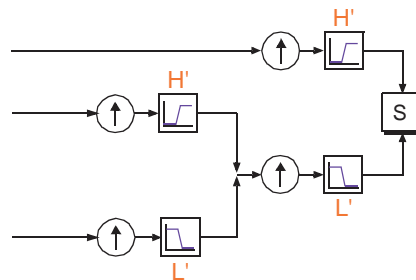
Since the analysis process is iterative, in theory it can be continued indefinitely. In reality, the decomposition can proceed only until the individual

details consist of a single sample or pixel. In practice, you'll select a suitable number of levels based on the nature of the signal, or on a suitable criterion such as *entropy* (see "Choosing the Optimal Decomposition" on page 6-147).

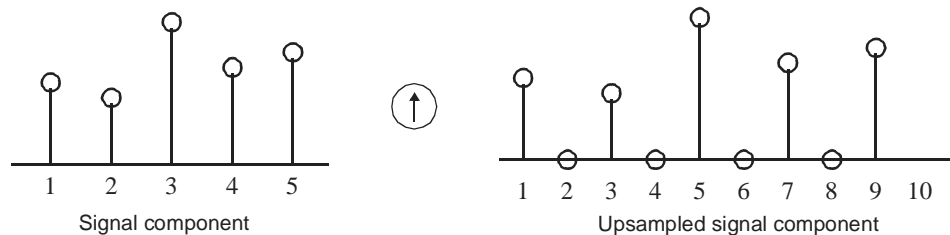
## Wavelet Reconstruction

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

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



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



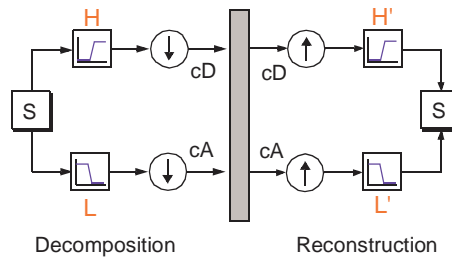
The toolbox includes commands, like `idwt` and `waverec`, that perform single-level or multilevel reconstruction, respectively, on the components of one-dimensional signals. These commands have their two-dimensional analogs, `idwt2` and `waverec2`.

## Reconstruction Filters

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

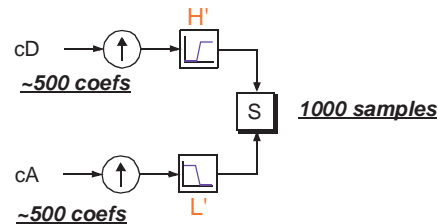
The downsampling of the signal components performed during the decomposition phase introduces a distortion called aliasing. It turns out that by carefully choosing filters for the decomposition and reconstruction phases that are closely related (but not identical), we can “cancel out” the effects of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

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



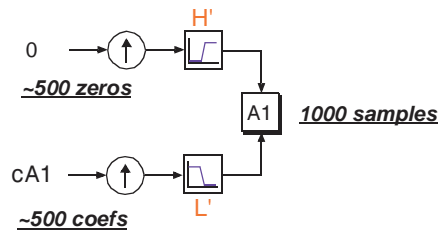
## Reconstructing Approximations and Details

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



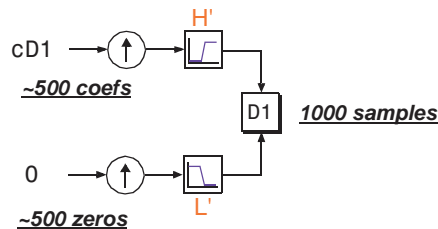
It is also possible to reconstruct the approximations and details themselves from their coefficient vectors. As an example, let's consider how we would reconstruct the first-level approximation  $A_1$  from the coefficient vector  $cA_1$ .

We pass the coefficient vector  $cA_1$  through the same process we used to reconstruct the original signal. However, instead of combining it with the level-one detail  $cD_1$ , we feed in a vector of zeros in place of the detail coefficients vector:



The process yields a reconstructed *approximation*  $A_1$ , which has the same length as the original signal  $S$  and which is a real approximation of it.

Similarly, we can reconstruct the first-level detail  $D_1$ , using the analogous process:

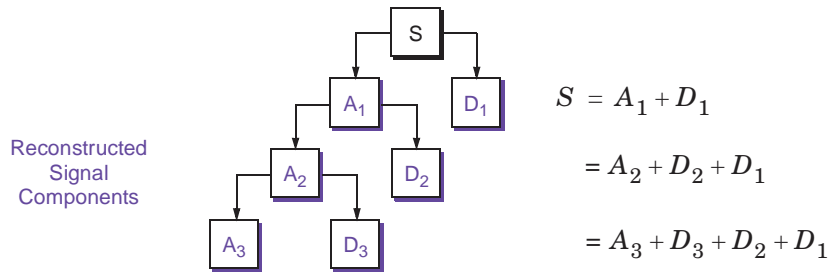


The reconstructed details and approximations are true constituents of the original signal. In fact, we find when we combine them that

$$A_1 + D_1 = S$$

Note that the coefficient vectors  $cA_1$  and  $cD_1$  — because they were produced by downsampling and are only half the length of the original signal — cannot directly be combined to reproduce the signal. *It is necessary to reconstruct the approximations and details before combining them.*

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

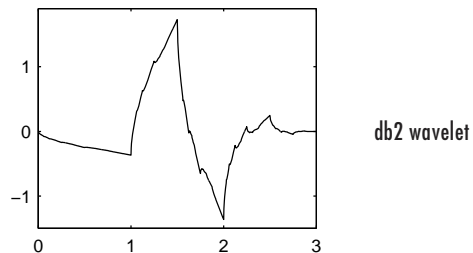


## Relationship of Filters to Wavelet Shapes

In the section “Reconstruction Filters” on page 1-30, we spoke of the importance of choosing the right filters. In fact, the choice of filters not only determines whether perfect reconstruction is possible, it also determines the shape of the wavelet we use to perform the analysis.

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

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



The filter coefficients can be obtained from the dbaux command:

```
Lprime = dbaux(2)
```

```
Lprime =
    0.3415    0.5915    0.1585    0.0915
```

If we reverse the order of this vector (see `wrev`), and then multiply every even sample by  $-1$ , we obtain the high-pass filter  $H'$ :

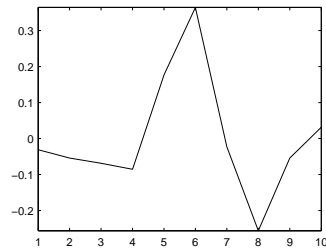
```
Hprime =
    0.0915    0.1585    0.5915    0.3415
```

Next, upsample  $H_{prime}$  by two (see `dyadup`), inserting zeros in alternate positions:

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

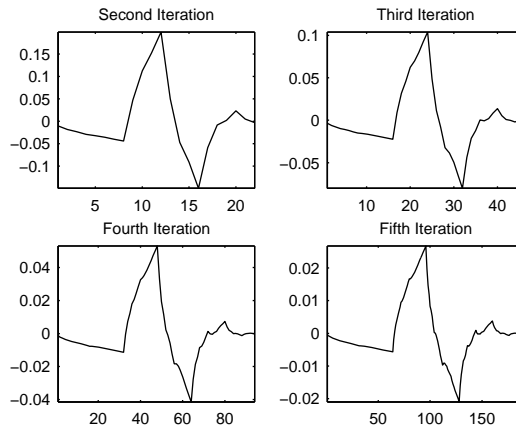
Finally, convolve the upsampled vector with the original low-pass filter:

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



If we iterate this process several more times, repeatedly upsampling and convolving the resultant vector with the four-element filter vector  $L_{prime}$ , a pattern begins to emerge.





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

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

### Scaling Function

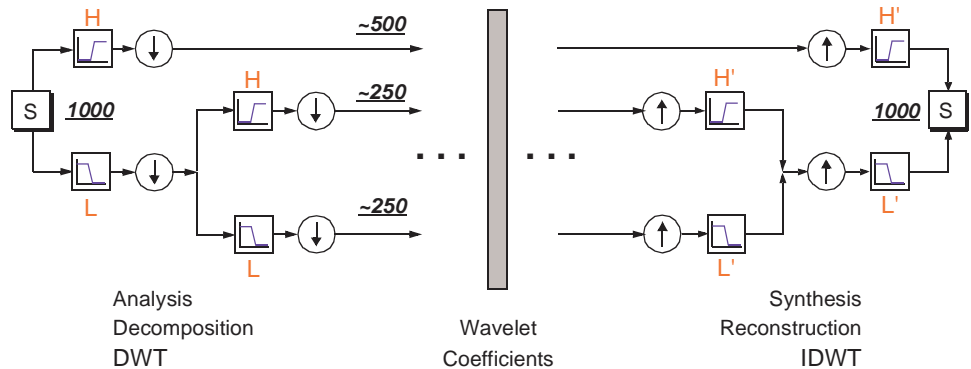
We’ve seen the interrelation of wavelets and quadrature mirror filters. The wavelet function  $\psi$  is determined by the high-pass filter, which also produces the details of the wavelet decomposition.

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

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

## Multistep Decomposition and Reconstruction

A multistep analysis-synthesis process can be represented as



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

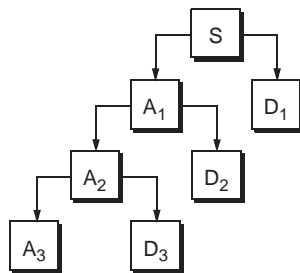
We've already discussed decomposition and reconstruction at some length. Of course, there is no point breaking up a signal merely to have the satisfaction of immediately reconstructing it. We may modify the wavelet coefficients before performing the reconstruction step. We perform wavelet analysis because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

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

## Wavelet Packet Analysis

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

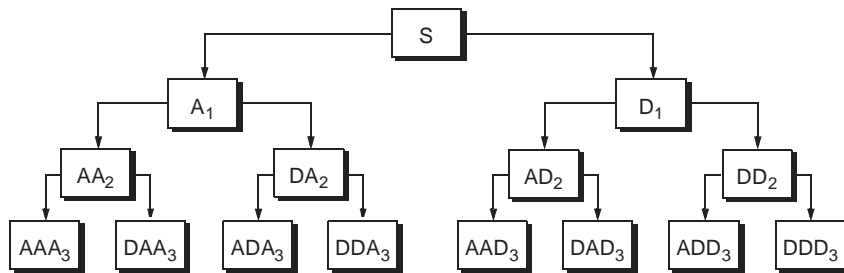
In wavelet analysis, a signal is split into an approximation and a detail. The approximation is then itself split into a second-level approximation and detail, and the process is repeated. For an  $n$ -level decomposition, there are  $n+1$  possible ways to decompose or encode the signal.



$$\begin{aligned}
 S &= A_1 + D_1 \\
 &= A_2 + D_2 + D_1 \\
 &= A_3 + D_3 + D_2 + D_1
 \end{aligned}$$

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

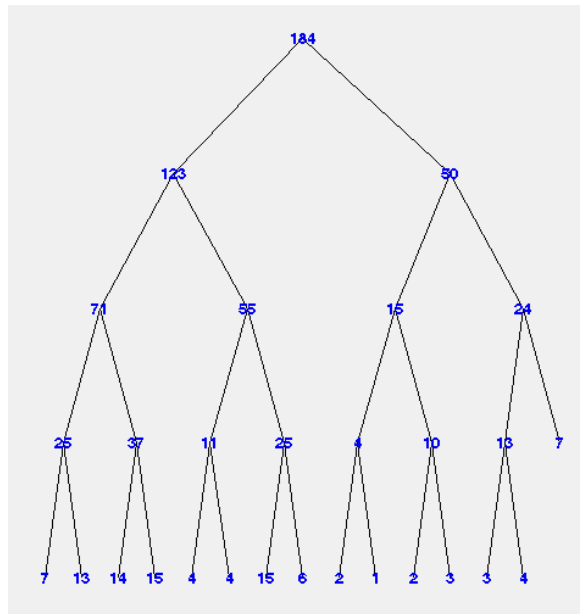
This yields more than  $2^{2^{n-1}}$  different ways to encode the signal. This is the *wavelet packet decomposition tree*.



The wavelet decomposition tree is a part of this complete binary tree.

For instance, wavelet packet analysis allows the signal  $S$  to be represented as  $A_1 + AAD_3 + DAD_3 + DD_2$ . This is an example of a representation that is not possible with ordinary wavelet analysis.

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



Simple and efficient algorithms exist for both wavelet packet decomposition and optimal decomposition selection. This toolbox uses an adaptive filtering algorithm, based on work by Coifman and Wickerhauser (see [CoiW92] in “References” on page 6-155), with direct applications in optimal signal coding and data compression.

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

## History of Wavelets

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

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

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

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

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

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

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

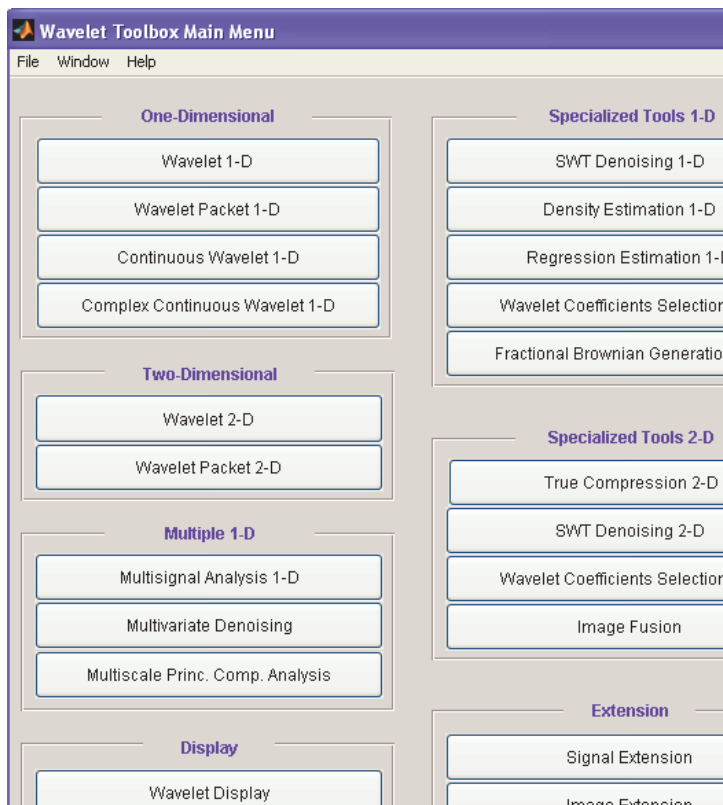
## Introduction to the Wavelet Families

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

- “Haar” on page 1-41
- “Daubechies” on page 1-42
- “Biorthogonal” on page 1-43
- “Coiflets” on page 1-45
- “Symlets” on page 1-45
- “Morlet” on page 1-46
- “Mexican Hat” on page 1-46
- “Meyer” on page 1-47
- “Other Real Wavelets” on page 1-47
- “Complex Wavelets” on page 1-47

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

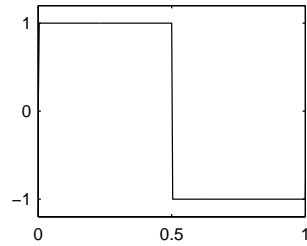
- 1 Type `wavemenu` at the MATLAB® command line. The **Wavelet Toolbox Main Menu** appears.



- 2 Click the **Wavelet Display** menu item. The **Wavelet Display** tool appears.
- 3 Select a family from the **Wavelet** menu at the top right of the tool.
- 4 Click the **Display** button. Pictures of the wavelets and their associated filters appear.
- 5 Obtain more information by clicking the information buttons located at the right.

## Haar

Any discussion of wavelets begins with Haar wavelet, the first and simplest. Haar wavelet is discontinuous, and resembles a step function. It represents the same wavelet as Daubechies db1. See “Haar” on page 6-73 for more detail.



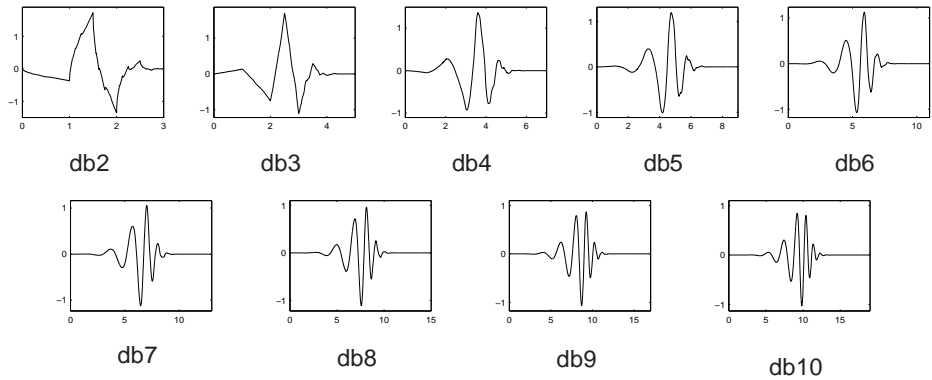
Wavelet function  $\psi$



## Daubechies

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

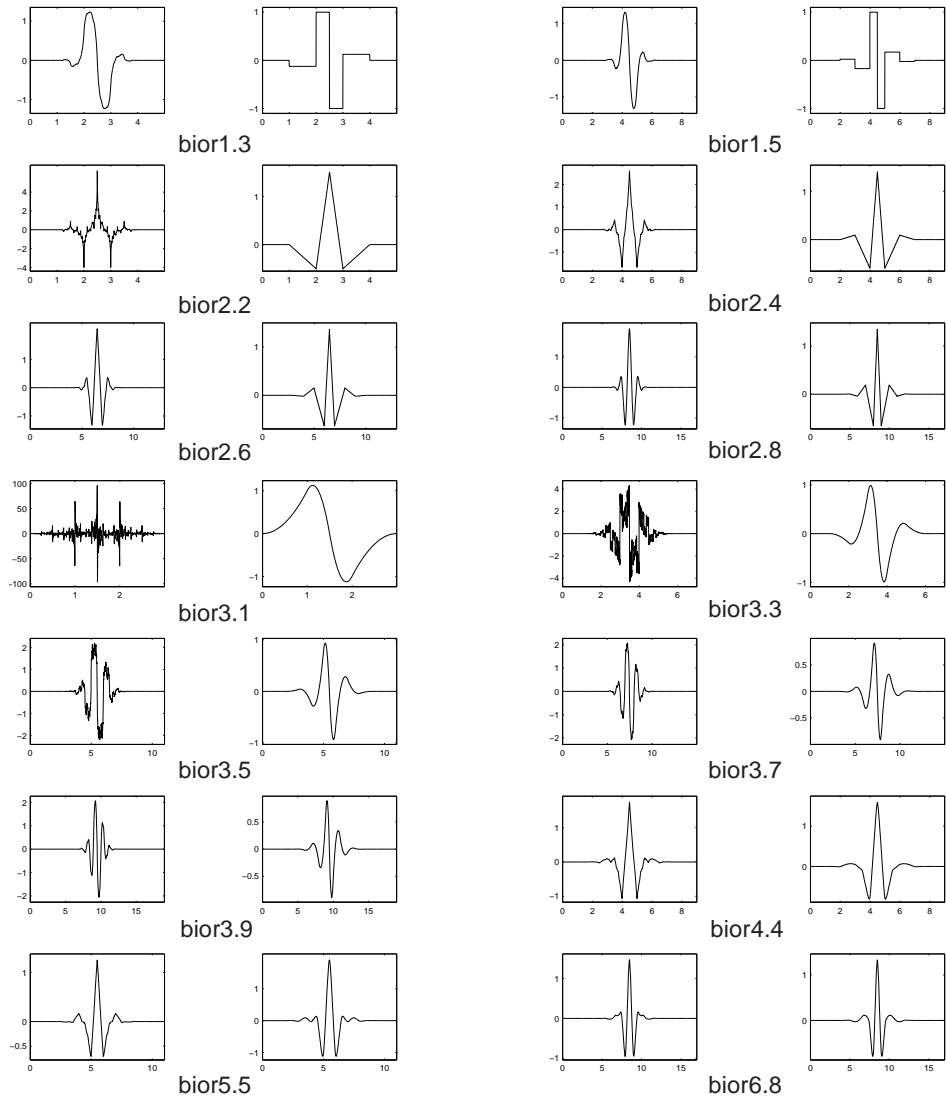
The names of the Daubechies family wavelets are written dbN, where N is the order, and db the “surname” of the wavelet. The db1 wavelet, as mentioned above, is the same as Haar wavelet. Here are the wavelet functions  $\psi$  of the next nine members of the family:



You can obtain a survey of the main properties of this family by typing `waveinfo('db')` from the MATLAB command line. See “Daubechies Wavelets: dbN” on page 6-72 for more detail.

## **Biorthogonal**

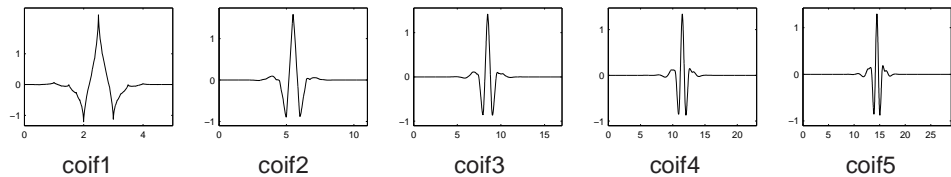
This family of wavelets exhibits the property of linear phase, which is needed for signal and image reconstruction. By using two wavelets, one for decomposition (on the left side) and the other for reconstruction (on the right side) instead of the same single one, interesting properties are derived.



You can obtain a survey of the main properties of this family by typing `waveinfo('bior')` from the MATLAB command line. See “Biorthogonal Wavelet Pairs: biorNr.Nd” on page 6-76 for more detail.

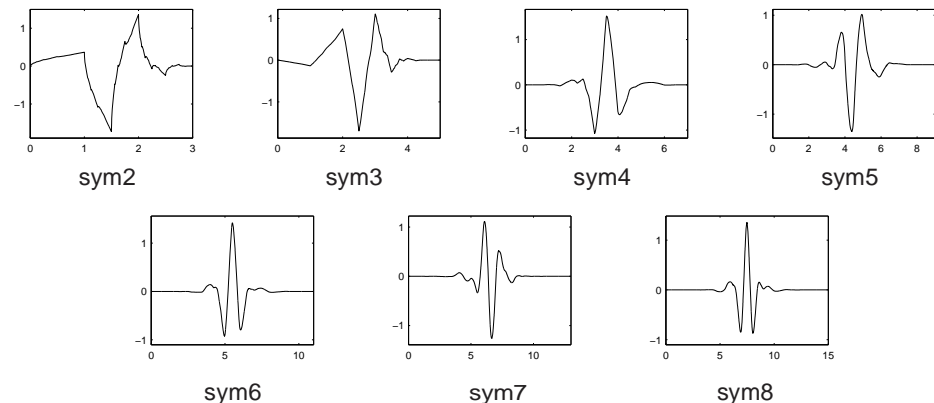
## Coiflets

Built by I. Daubechies at the request of R. Coifman. The wavelet function has  $2N$  moments equal to 0 and the scaling function has  $2N-1$  moments equal to 0. The two functions have a support of length  $6N-1$ . You can obtain a survey of the main properties of this family by typing `waveinfo('coif')` from the MATLAB command line. See “Coiflet Wavelets: `coifN`” on page 6-75 for more detail.



## Symlets

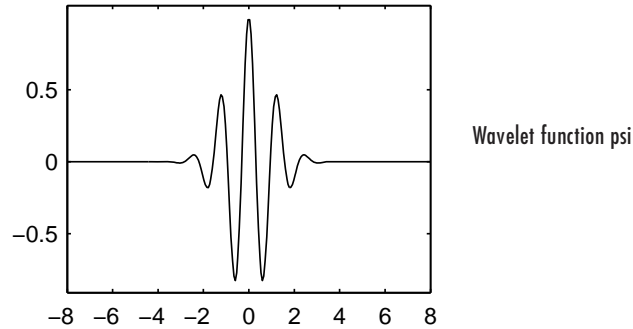
The symlets are nearly symmetrical wavelets proposed by Daubechies as modifications to the db family. The properties of the two wavelet families are similar. Here are the wavelet functions  $\psi$ .



You can obtain a survey of the main properties of this family by typing `waveinfo('sym')` from the MATLAB command line. See “Symlet Wavelets: `symN`” on page 6-74 for more detail.

## Morlet

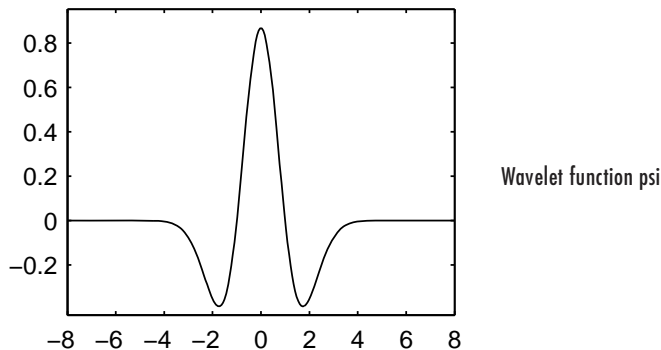
This wavelet has no scaling function, but is explicit.



You can obtain a survey of the main properties of this family by typing `waveinfo('mor1')` from the MATLAB command line. See “Morlet Wavelet: mor1” on page 6-81 for more detail.

## Mexican Hat

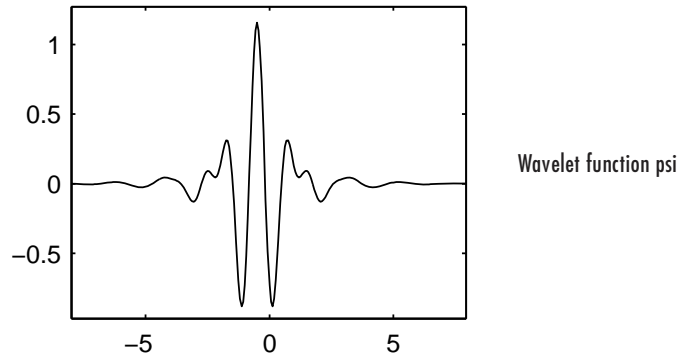
This wavelet has no scaling function and is derived from a function that is proportional to the second derivative function of the Gaussian probability density function.



You can obtain a survey of the main properties of this family by typing `waveinfo('mexh')` from the MATLAB command line. See “Mexican Hat Wavelet: mexh” on page 6-80 for more information.

## Meyer

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



You can obtain a survey of the main properties of this family by typing `waveinfo('meyer')` from the MATLAB command line. See “Meyer Wavelet: meyr” on page 6-78 for more detail.

## Other Real Wavelets

Some other real wavelets are available in the toolbox:

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

See “Additional Real Wavelets” on page 6-82 for more information.

## Complex Wavelets

Some complex wavelet families are available in the toolbox:

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

See “Complex Wavelets” on page 6-84 for more information.



# Using Wavelets

---

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

- Introduction to Wavelet Toolbox™ GUIs and Functions (p. 2-3)
- One-Dimensional Continuous Wavelet Analysis (p. 2-4)
- One-Dimensional Complex Continuous Wavelet Analysis (p. 2-20)
- One-Dimensional Discrete Wavelet Analysis (p. 2-29)
- Two-Dimensional Discrete Wavelet Analysis (p. 2-66)
- Wavelets: Working with Images (p. 2-97)
- One-Dimensional Discrete Stationary Wavelet Analysis (p. 2-104)
- Two-Dimensional Discrete Stationary Wavelet Analysis (p. 2-122)
- One-Dimensional Wavelet Regression Estimation (p. 2-140)
- One-Dimensional Wavelet Density Estimation (p. 2-150)
- One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients (p. 2-158)
- One-Dimensional Selection of Wavelet Coefficients Using the Graphical Interface (p. 2-168)
- Two-Dimensional Selection of Wavelet Coefficients Using the Graphical Interface (p. 2-178)
- One-Dimensional Extension (p. 2-186)
- Two-Dimensional Extension (p. 2-195)
- Image Fusion (p. 2-199)
- One-Dimensional Fractional Brownian Motion Synthesis (p. 2-208)
- New Wavelet for CWT (p. 2-216)
- Multivariate Wavelet De-Noising (p. 2-228)



- Multiscale Principal Components Analysis (p. 2-247)
- One-Dimensional Multisignal Analysis (p. 2-261)
- Two-Dimensional True Compression (p. 2-317)

## Introduction to Wavelet Toolbox™ GUIs and Functions

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

- Examine and explore properties of individual wavelets and wavelet packets
- Examine statistics of signals and signal components
- Perform a continuous wavelet transform of a one-dimensional signal
- Perform discrete analysis and synthesis of one- and two-dimensional signals
- Perform wavelet packet analysis of one- and two-dimensional signals (see “Using Wavelet Packets” on page 5-1)
- Compress and remove noise from signals and images

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

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

---

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

---

# One-Dimensional Continuous Wavelet Analysis

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

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

In this section, you'll learn how to

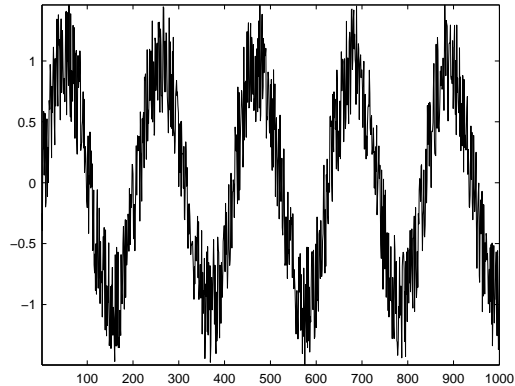
- Load a signal
- Perform a continuous wavelet transform of a signal
- Produce a plot of the coefficients
- Produce a plot of coefficients at a given scale
- Produce a plot of local maxima of coefficients across scales
- Select the displayed plots
- Switch from scale to pseudo-frequency information
-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

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

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

## Continuous Analysis Using the Command Line

This example involves a noisy sinusoidal signal.



### 1 Load a signal.

From the MATLAB® prompt, type

```
load noissin;
```

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

```
whos
```

Name	Size	Bytes	Class
<code>noissin</code>	1x1000	8000	double array

### 2 Perform a Continuous Wavelet Transform.

Use the `cwt` command. Type

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

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

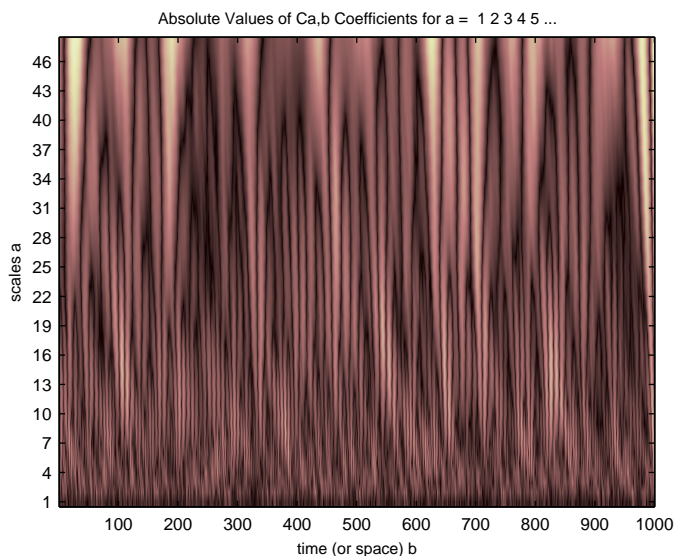
### 3 Plot the coefficients.

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

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

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

A plot appears.



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

#### 4 Choose scales for the analysis.

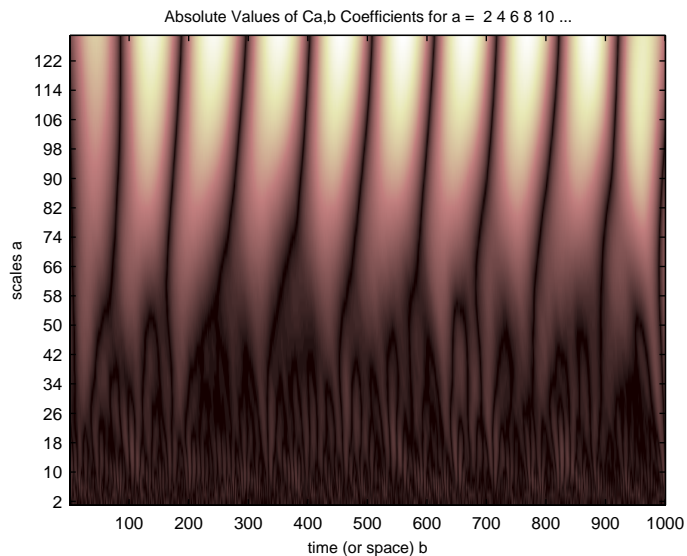
The second argument to `cwt` gives you fine control over the scale levels on which the continuous analysis is performed. In the previous example, we used all scales from 1 to 48, but you can construct any scale vector subject to these constraints:

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

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

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

A new plot appears:



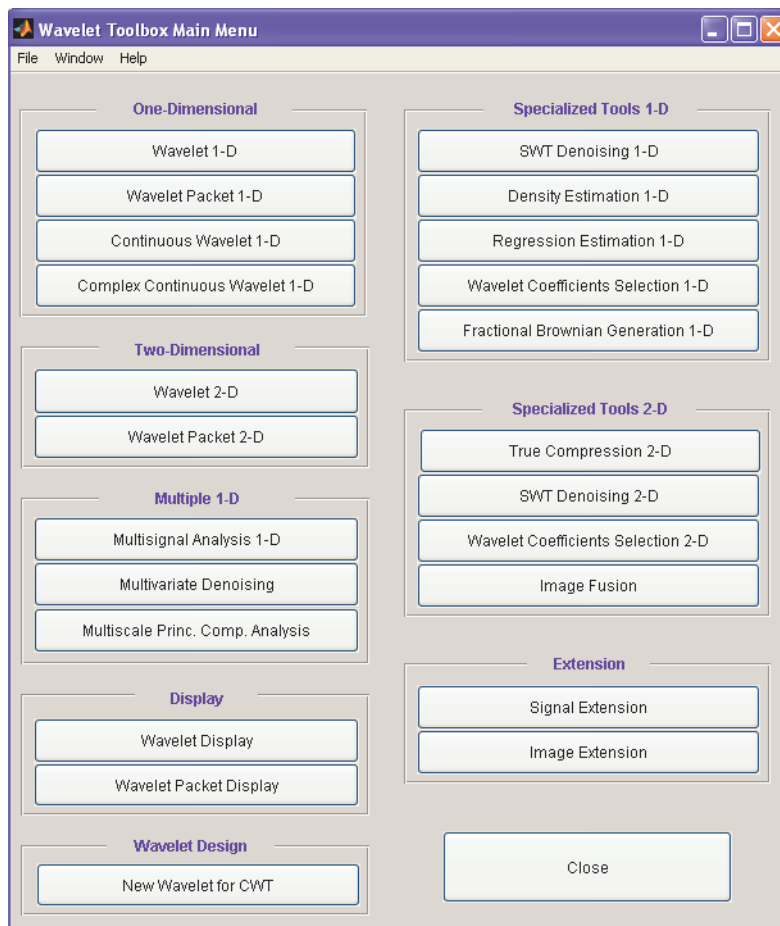
This plot gives a clearer picture of what's happening with the signal, highlighting the periodicity.

### Continuous Analysis Using the Graphical Interface

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

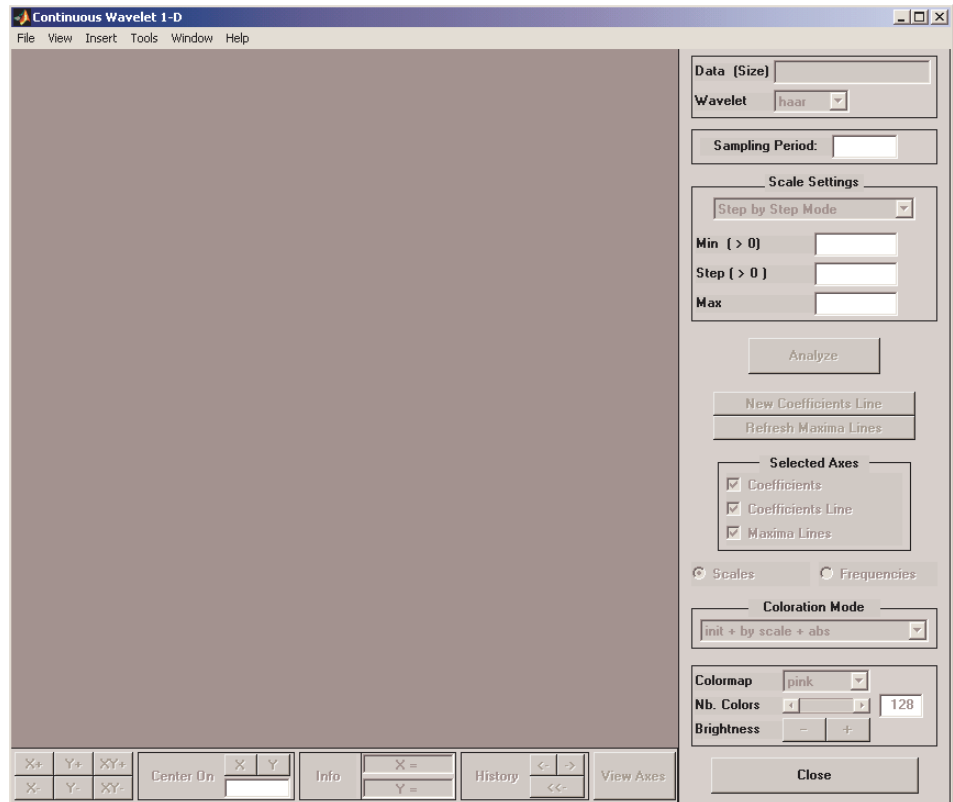
- 1 Start the Continuous Wavelet 1-D Tool. From the MATLAB prompt, type `wavemenu`

The **Wavelet Toolbox Main Menu** appears.



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

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





### 2 Load a signal.

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

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

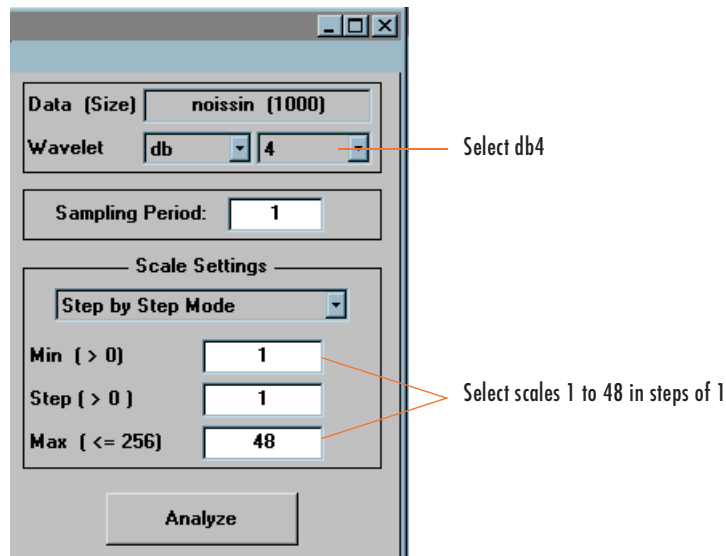
The noisy sinusoidal signal is loaded into the **Continuous Wavelet 1-D** tool.

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

### 3 Perform a Continuous Wavelet Transform.

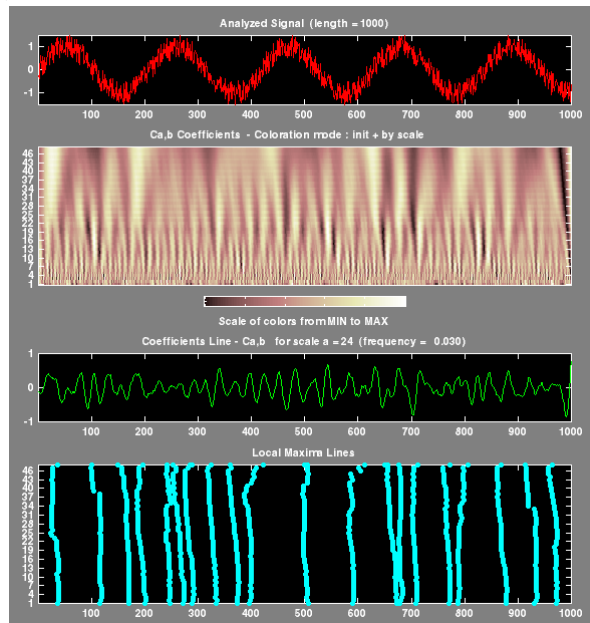
To start our analysis, let's perform an analysis using the db4 wavelet at scales 1 through 48, just as we did using command line functions in the previous section.

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



#### 4 Click the **Analyze** button.

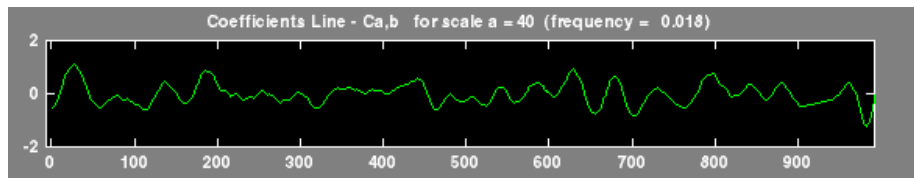
After a pause for computation, the tool displays the coefficients plot, the coefficients line plot corresponding to the scale  $a = 24$ , and the local maxima plot, which displays the chaining across scales (from  $a = 48$  down to  $a = 1$ ) of the coefficients local maxima.



#### 5 View Wavelet Coefficients Line.

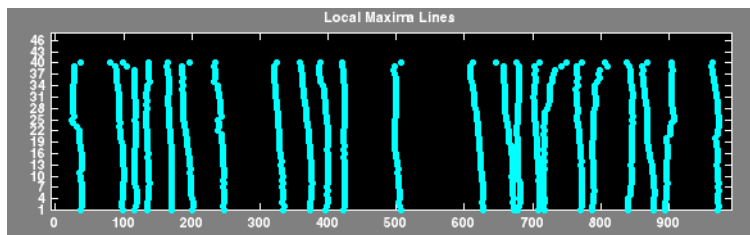
Select another scale  $a = 40$  by clicking in the coefficients plot with the right mouse button. See step 9 to know, more precisely, how to select the desired scale.

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

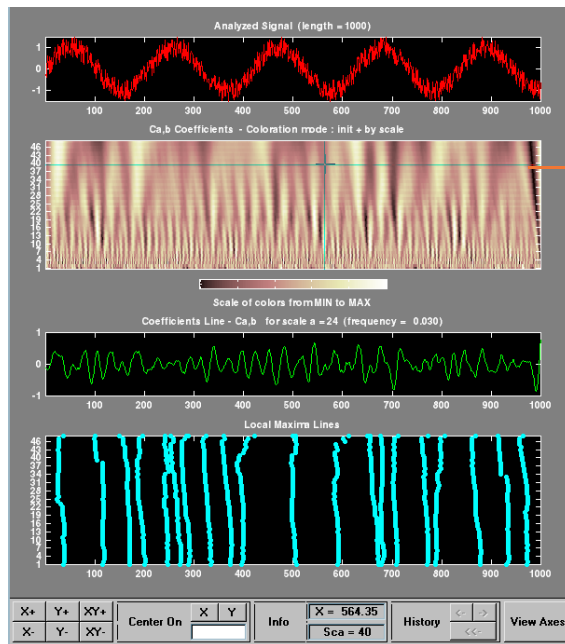


### 6 View Maxima Line.

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

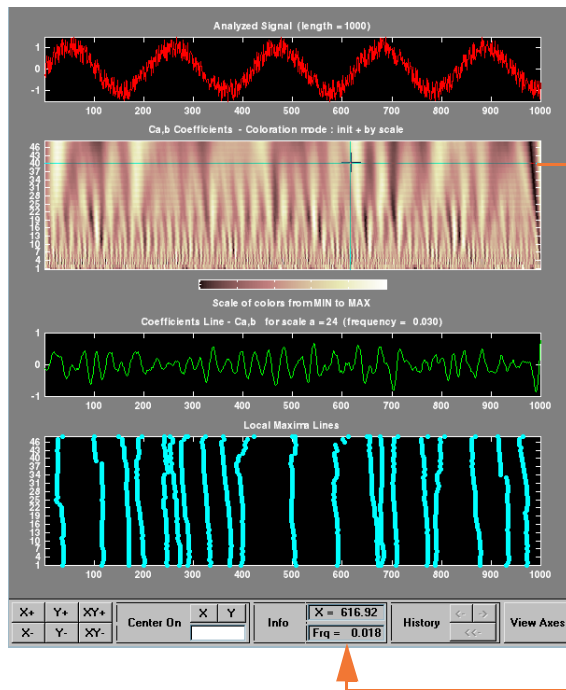


Hold down the right mouse button over the coefficients plot. The position of the mouse is given by the **Info** frame (located at the bottom of the screen) in terms of location (**X**) and scale (**Sca**).



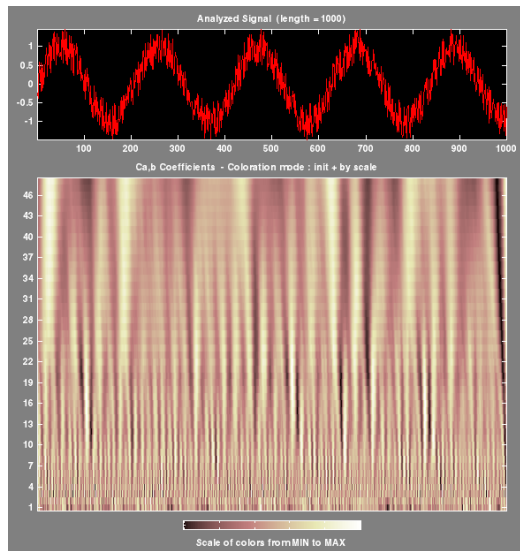
## 7 Switch from scale to Pseudo-Frequency Information.

Using the option button on the right part of the screen, select **Frequencies** instead of **Scales**. Again hold down the right mouse button over the coefficients plot, the position of the mouse is given in terms of location (**X**) and frequency (**Frq**) in Hertz.



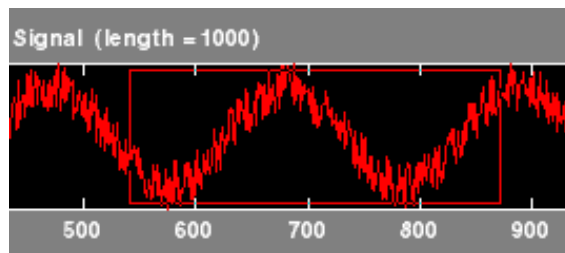
This facility allows you to interpret scale in terms of an associated pseudo-frequency, which depends on the wavelet and the sampling period. For more information on the connection between scale and frequency, see “How to Connect Scale to Frequency?” on page 6-66.

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



**9** Zoom in on detail.

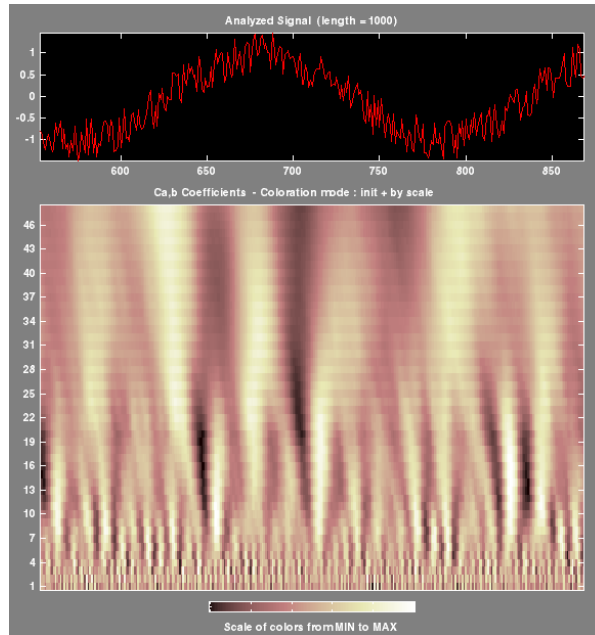
Drag a rubber band box (by holding down the left mouse button) over the portion of the signal you want to magnify.



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

X+	Y+	XY+	Center On	X	Y	Info	X =	History	<-	>	View Axes
X-	Y-	XY-					Y =		<<-		

The **Continuous Wavelet 1-D** tool enlarges the displayed signal and coefficients plot (for more information on zooming, see “Connection of Plots” on page A-3).



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

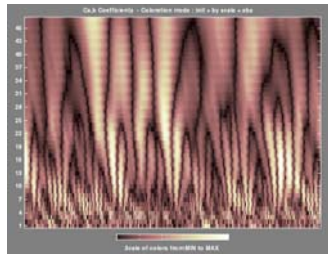
### 11 View normal or absolute coefficients.

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

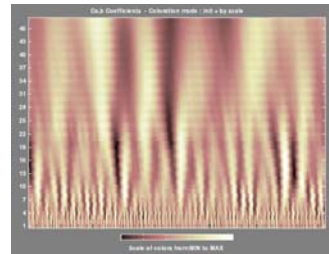
More generally, the coefficients coloration can be done in several different ways. For more details on the Coloration Mode, see “Controlling the Coloration Mode” on page A-7.

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

The coefficients plot is redisplayed in the mode you select.



Absolute Mode



Normal Mode

## Importing and Exporting Information from the Graphical Interface

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

You can

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

### Loading Signals into the Continuous Wavelet 1-D Tool

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

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

```
save warma warma
```



The workspace variable `warma` must be a vector.

```
sizwarma = size(warma)

sizwarma =
         1         1000
```

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

---

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

---

### Saving Wavelet Coefficients

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

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

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

Consider the example analysis:

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

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

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

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

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

## One-Dimensional Complex Continuous Wavelet Analysis

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

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

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

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

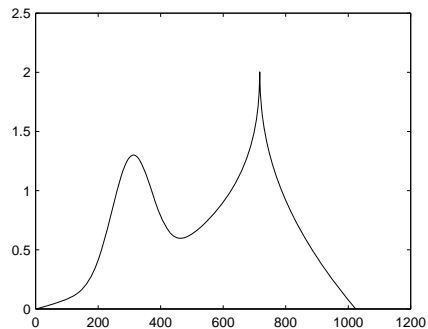
In this section, you’ll learn how to

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

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

## Complex Continuous Analysis Using the Command Line

This example involves a cusp signal.



### 1 Load a signal.

From the MATLAB® prompt, type

```
load cuspamax;
```

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

```
whos
```

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

```
caption
```

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

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

### 2 Perform a Continuous Wavelet Transform.

Use the `cwt` command. Type

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

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

### 3 Plot the coefficients.

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

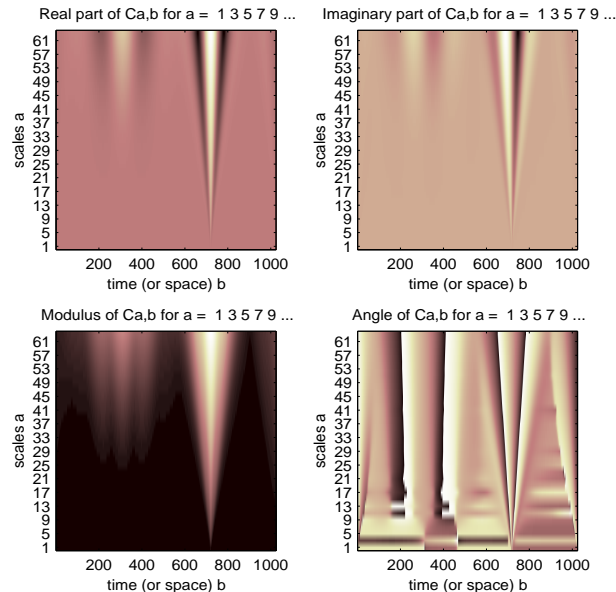
- Real and imaginary parts
- Modulus and angle

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

Type

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

A plot appears:



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

## Complex Continuous Analysis Using the Graphical Interface

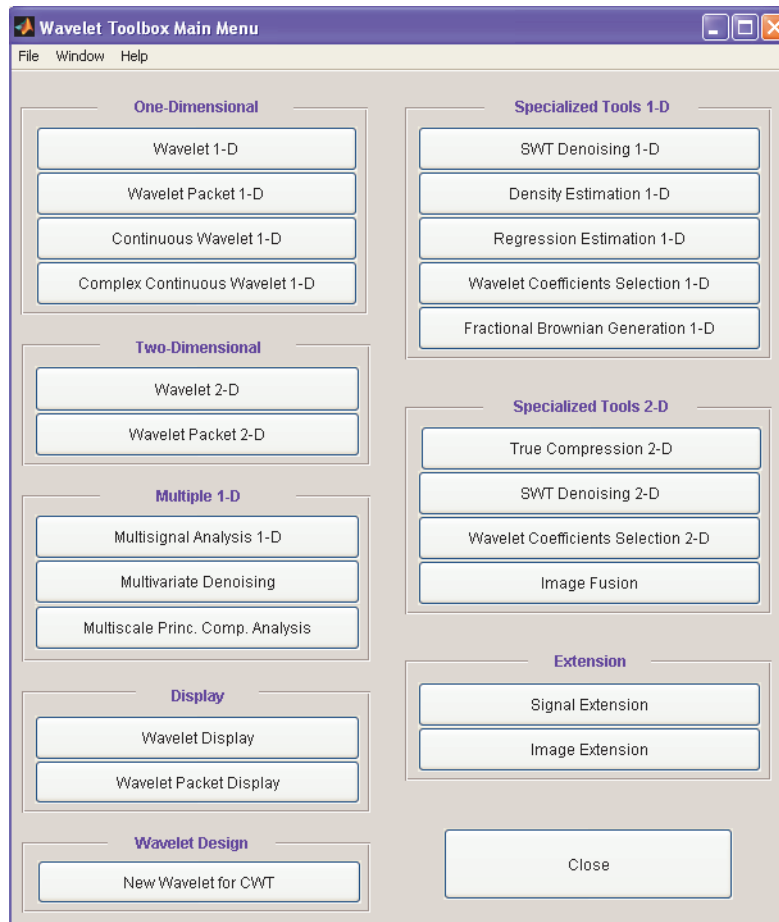
We now use the **Complex Continuous Wavelet 1-D** tool to analyze the same cusp signal we examined using the command line interface in the previous section.

- 1 Start the Complex Continuous Wavelet 1-D Tool.

From the MATLAB prompt, type

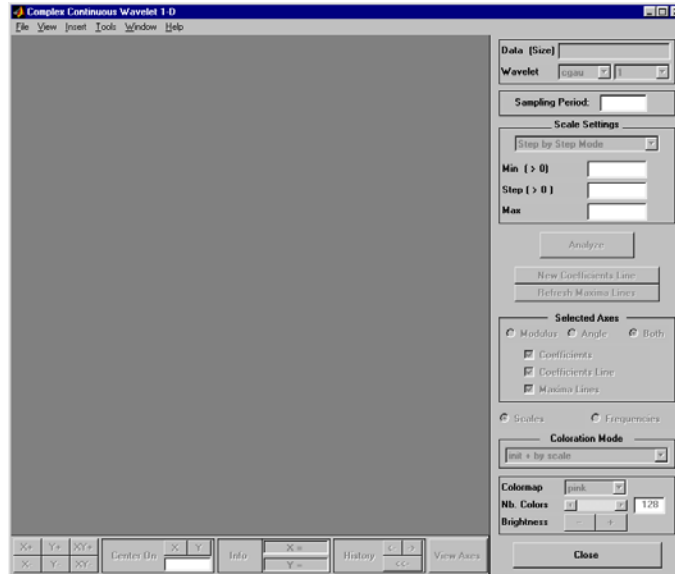
```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.



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

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



## 2 Load a signal.

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

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

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

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

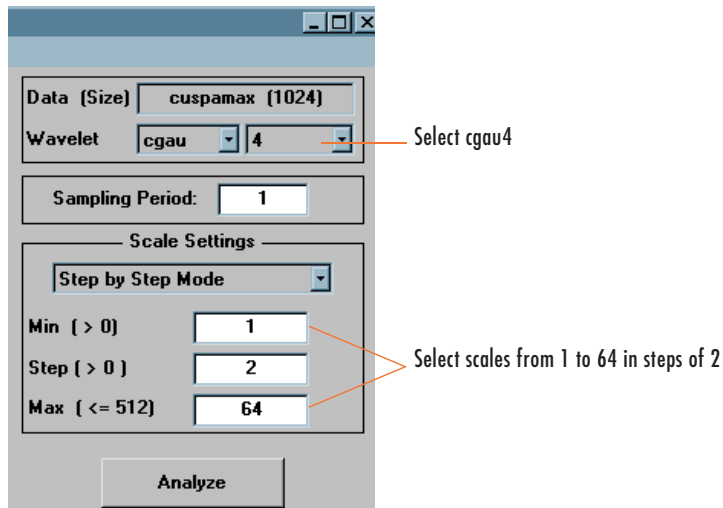
## 3 Perform a Complex Continuous Wavelet Transform

To start our analysis, let's perform an analysis using the `cgau4` wavelet at scales 1 through 64 in steps of 2, just as we did using command-line



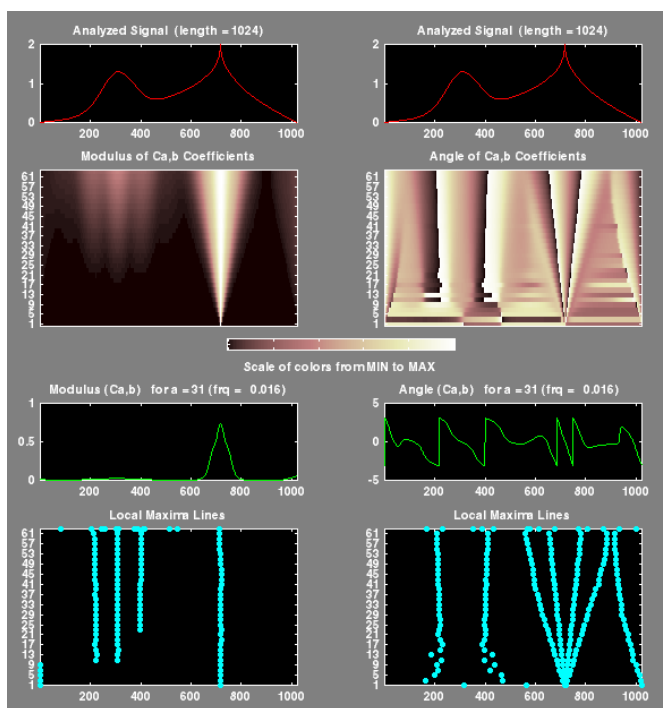
functions in “Complex Continuous Analysis Using the Command Line” on page 2-21.

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



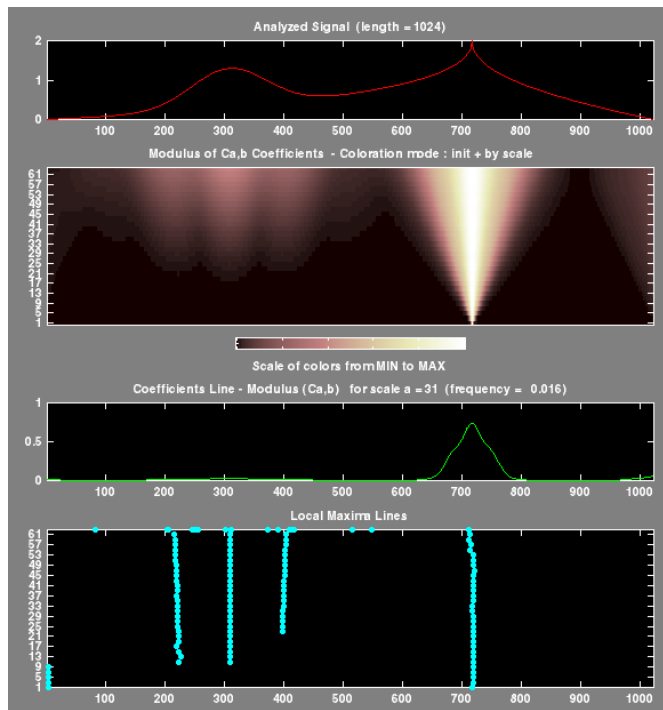
Click the **Analyze** button.

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



Each side has exactly the same representation that we found in “Continuous Analysis Using the Graphical Interface” on page 2-8.

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



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

### Importing and Exporting Information from the Graphical Interface

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

The only difference is that the variable `coefs` is a complex matrix (see “Saving Wavelet Coefficients” on page 2-18).

## One-Dimensional Discrete Wavelet Analysis

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

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

### Analysis-Decomposition Functions

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

### Synthesis-Reconstruction Functions

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

### Decomposition Structure Utilities

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

## De-noising and Compression

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

In this section, you'll learn how to

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

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

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

## One-Dimensional Analysis Using the Command Line

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



### 1 Load a signal.

From the MATLAB<sup>®</sup> prompt, type

```
load leleccum;
```

Set the variables. Type

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

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

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

Type

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

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

**3** Construct approximations and details from the coefficients.

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

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

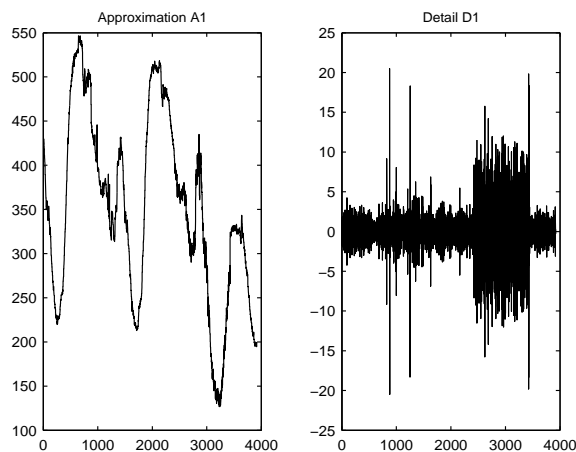
or

```
A1 = idwt(cA1,[],'db1',1_s);  
D1 = idwt([],cD1,'db1',1_s);
```

**4** Display the approximation and detail.

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

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

**5** Regenerate a signal by using the Inverse Wavelet Transform.

To find the inverse transform, type

```
A0 = idwt(cA1,cD1,'db1',1_s);  
err = max(abs(s-A0))
```

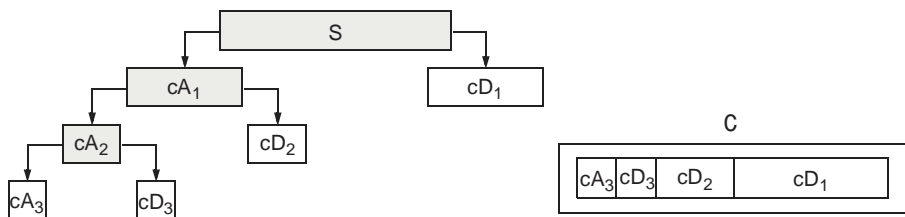
```
err =
    2.2737e-013
```

### 6 Perform a multilevel wavelet decomposition of a signal.

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

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

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



### 7 Extract approximation and detail coefficients.

To extract the level 3 approximation coefficients from C, type

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

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

```
cD3 = detcoef(C,L,3);
```

```
cD2 = detcoef(C,L,2);
```

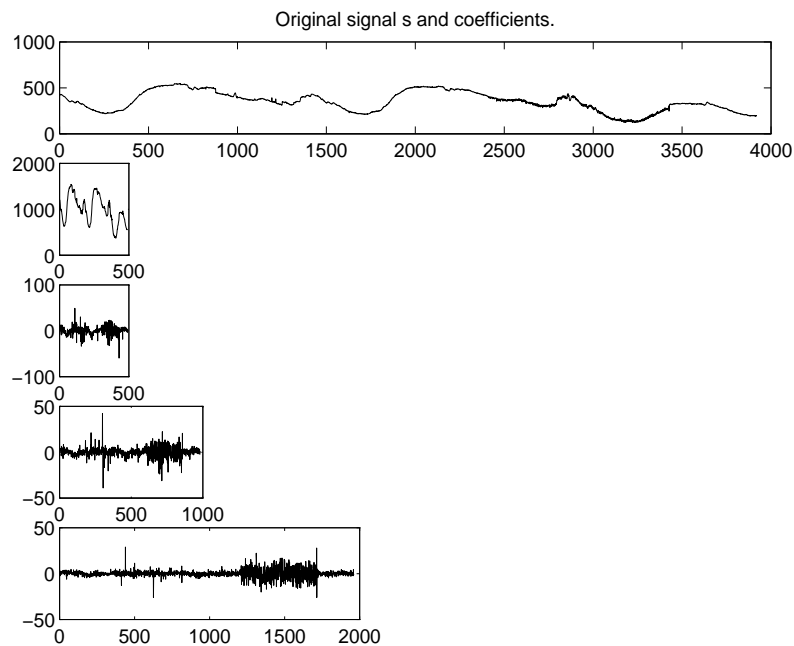
```
cD1 = detcoef(C,L,1);
```

or

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



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



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

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

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

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

```
D1 = wrcoef('d',C,L,'db1',1);
```

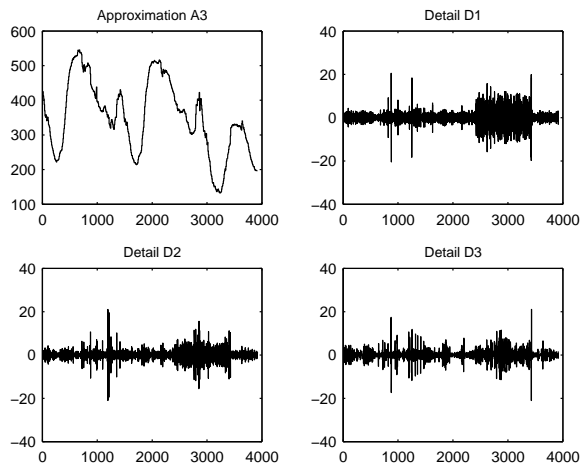
```
D2 = wrcoef('d',C,L,'db1',2);
```

```
D3 = wrcoef('d',C,L,'db1',3);
```

**9** Display the results of a multilevel decomposition.

To display the results of the level 3 decomposition, type

```
subplot(2,2,1); plot(A3);
title('Approximation A3')
subplot(2,2,2); plot(D1);
title('Detail D1')
subplot(2,2,3); plot(D2);
title('Detail D2')
subplot(2,2,4); plot(D3);
title('Detail D3')
```

**10** Reconstruct the original signal from the Level 3 decomposition.

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

```
A0 = waverec(C,L,'db1');
err = max(abs(s-A0))
err =
    4.5475e-013
```

### 11 Crude de-noising of a signal.

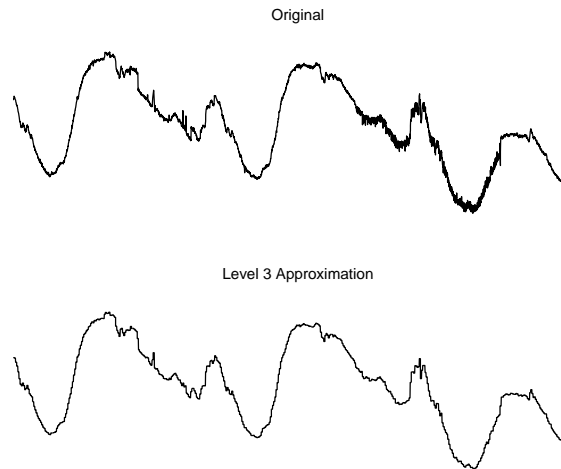
Using wavelets to remove noise from a signal requires identifying which component or components contain the noise, and then reconstructing the signal without those components.

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

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

To compare the approximation to the original signal, type

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



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

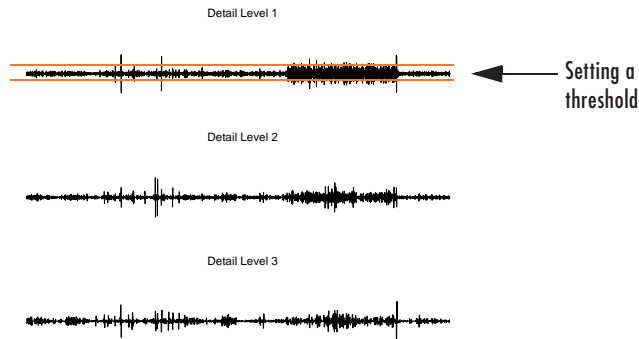
Optimal de-noising requires a more subtle approach called *thresholding*. This involves discarding only the portion of the details that exceeds a certain limit.

## 12 Remove noise by thresholding.

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

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

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



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

Note that `cd1`, `cd2`, and `cd3` are just MATLAB vectors, so we could directly manipulate each vector, setting each element to some fraction of the vectors' peak or average value. Then we could reconstruct new detail signals `D1`, `D2`, and `D3` from the thresholded coefficients.

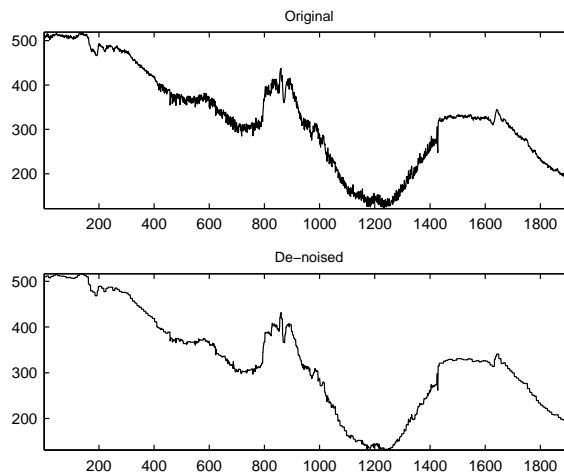
To denoise the signal, use the `ddencmp` command to calculate the default parameters and the `wdencomp` command to perform the actual de-noising, type

```
[thr,sorh,keepapp] = ddencmp('den','wv',s);  
clean = wdencomp('gb1',C,L,'db1',3,thr,sorh,keepapp);
```

Note that `wdencomp` uses the results of the decomposition (C and L) that we calculated in step 6 on page 2-33. We also specify that we used the `db1` wavelet to perform the original analysis, and we specify the global thresholding option `'gb1'`. See `ddencomp` and `wdencomp` in the reference pages for more information about the use of these commands.

To display both the original and denoised signals, type

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



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

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

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

- “Remove noise from a signal.” on page 2-49
- “De-Noising” on page 6-97
- “One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients” on page 2-158
- “One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients” on page 6-107

### **One-Dimensional Analysis Using the Graphical Interface**

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

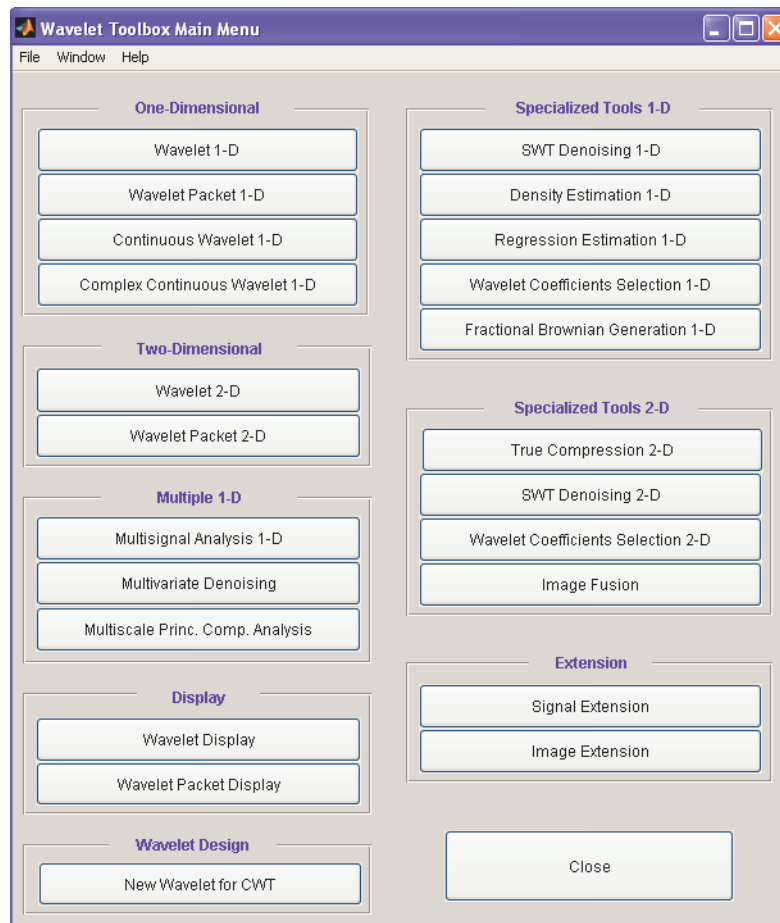


- 1 Start the 1-D Wavelet Analysis Tool.

From the MATLAB prompt, type

```
wavemenu
```

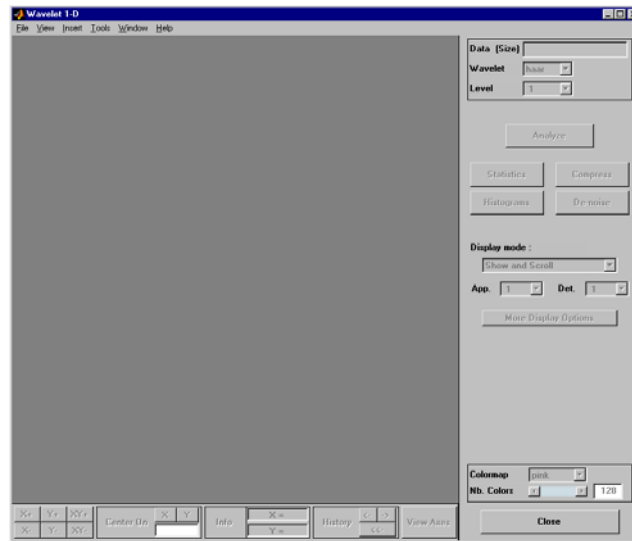
The **Wavelet Toolbox Main Menu** appears.



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

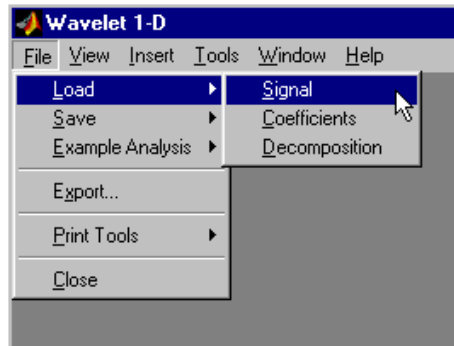
The discrete wavelet analysis tool for one-dimensional signal data appears.



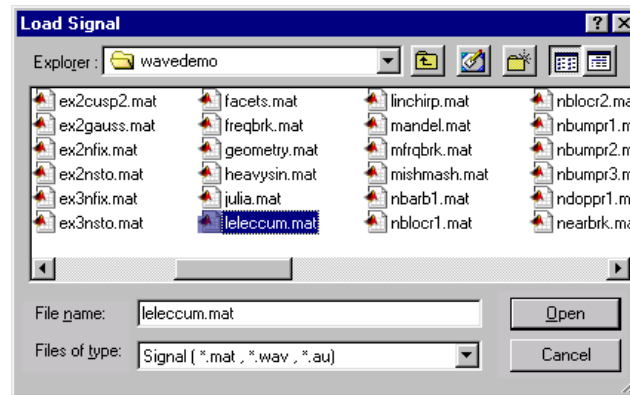


### 3 Load a signal.

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



When the **Load Signal** dialog box appears, select the demo MAT-file `1e1eccum.mat`, which should reside in the MATLAB directory `toolbox/wavelet/wavedemo`. Click the **OK** button.

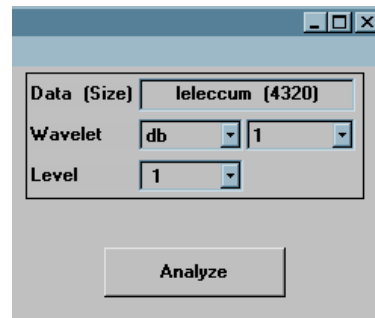


The electrical consumption signal is loaded into the **Wavelet 1-D** tool.

#### 4 Perform a single-level wavelet decomposition.

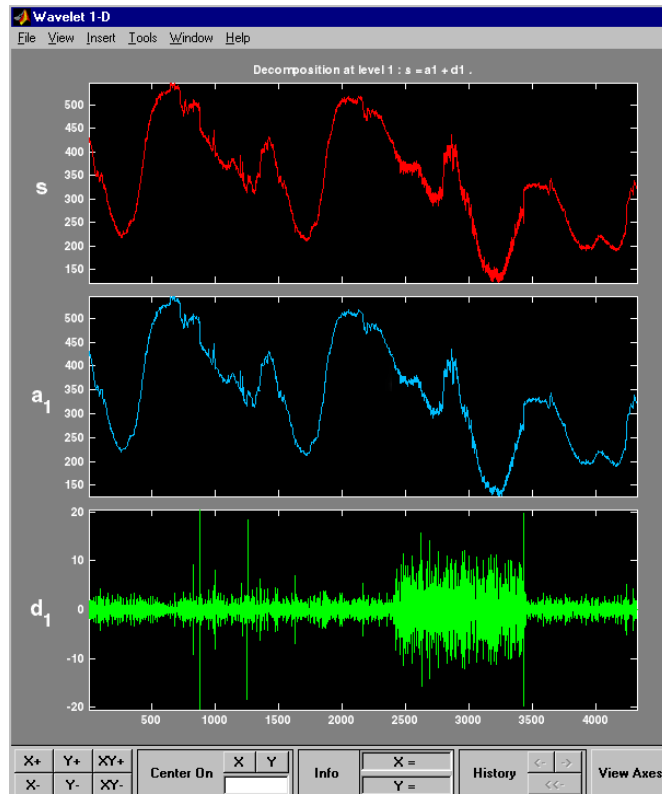
To start our analysis, let's perform a single-level decomposition using the db1 wavelet, just as we did using the command-line functions in "One-Dimensional Analysis Using the Command Line" on page 2-31.

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



Click the **Analyze** button.

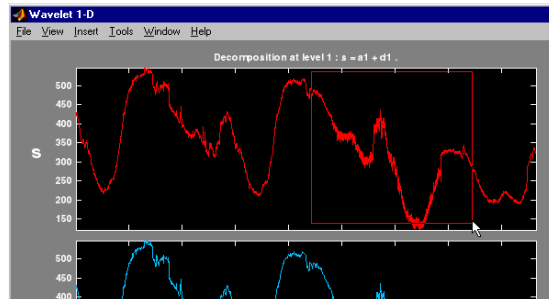
After a pause for computation, the tool displays the decomposition.



### 5 Zoom in on relevant detail.

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

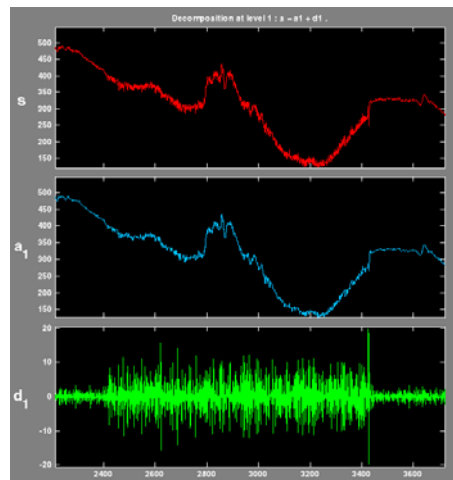
Drag a rubber band box (by holding down the left mouse button) over the portion of the signal you want to magnify. Here, we've selected the noisy part of the original signal.



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



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

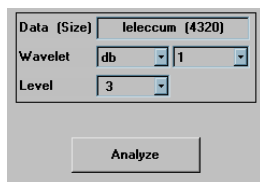


The other zoom controls do more or less what you'd expect them to. The **X-** button, for example, zooms out horizontally. The history function keeps track of all your views of the signal. Return to a previous zoom level by clicking the left arrow button.

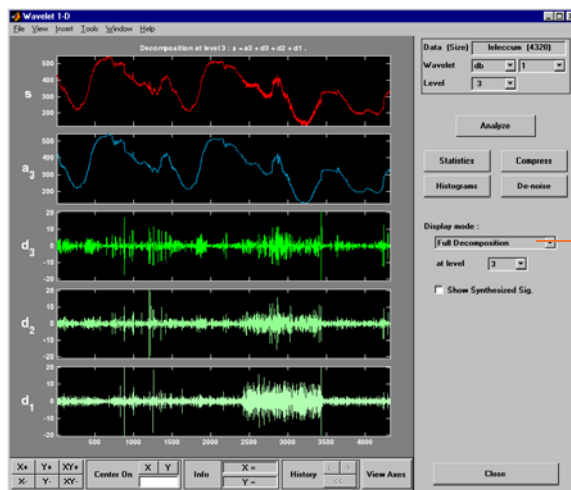
### 6 Perform a multilevel decomposition.

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

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



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



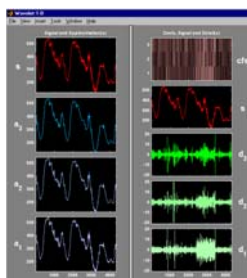
Select a view

## Selecting Different Views of the Decomposition

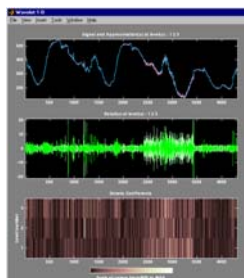
The **Display mode** menu (middle right) lets you choose different views of the wavelet decomposition.

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

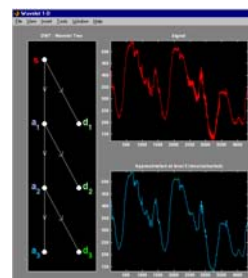
- “Separate Mode,” which shows the details and the approximations in separate columns.
- “Superimpose Mode,” which shows the details on a single plot superimposed in different colors. The approximations are plotted similarly.
- “Tree Mode,” which shows the decomposition tree, the original signal, and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



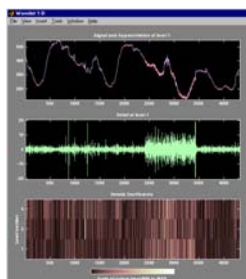
Separate Mode



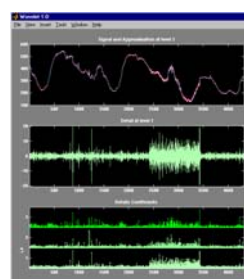
Superimpose Mode



Tree Mode



Show & Scroll Mode



Show & Scroll Mode (Stem Cfs)

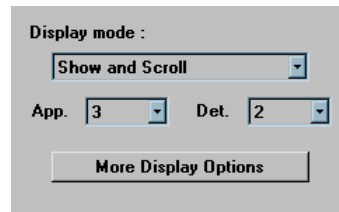
You can change the default display mode on a per-session basis. Select the desired mode from the **View > Default Display Mode** submenu.

---

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

---

Depending on which display mode you select, you may have access to additional display options through the **More Display Options** button (for more information, see “More Display Options” on page A-19).

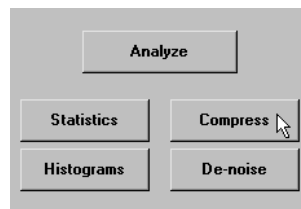


These options include the ability to suppress the display of various components, and to choose whether or not to display the original signal along with the details and approximations.

#### 7 Remove noise from a signal.

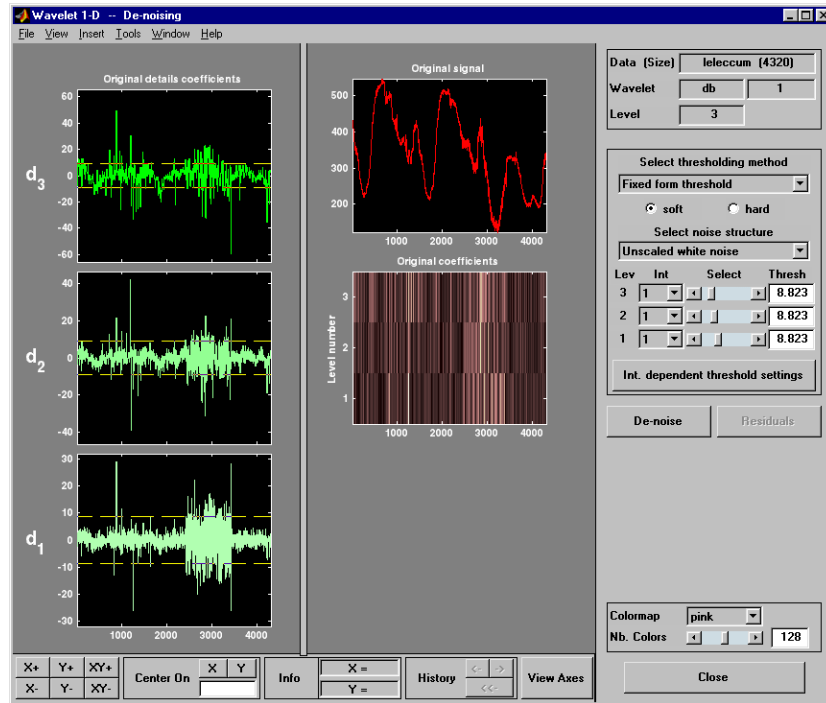
The graphical interface tools feature a de-noising option with a predefined thresholding strategy. This makes it very easy to remove noise from a signal.

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





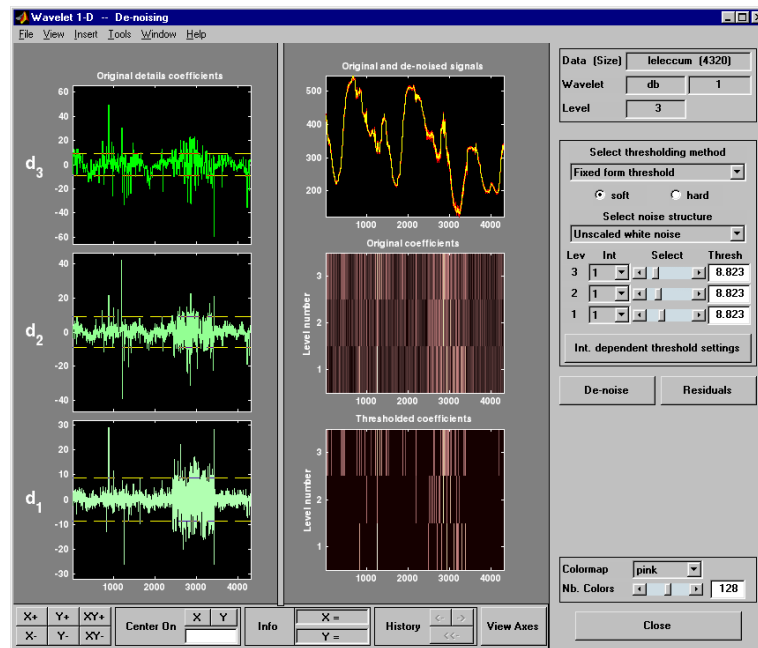
The **Wavelet 1-D De-noising** window appears.



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

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

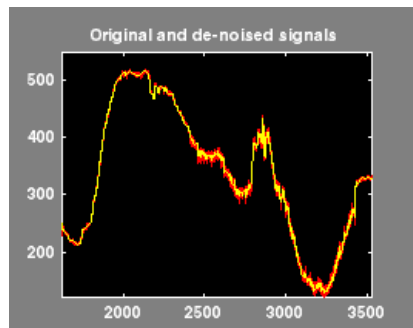
The de-noised signal appears superimposed on the original. The tool also plots the wavelet coefficients of both signals.



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

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

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



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

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

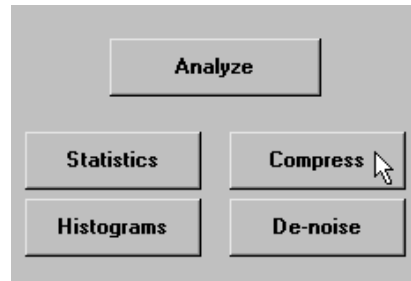
### 8 Refine the analysis.

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

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

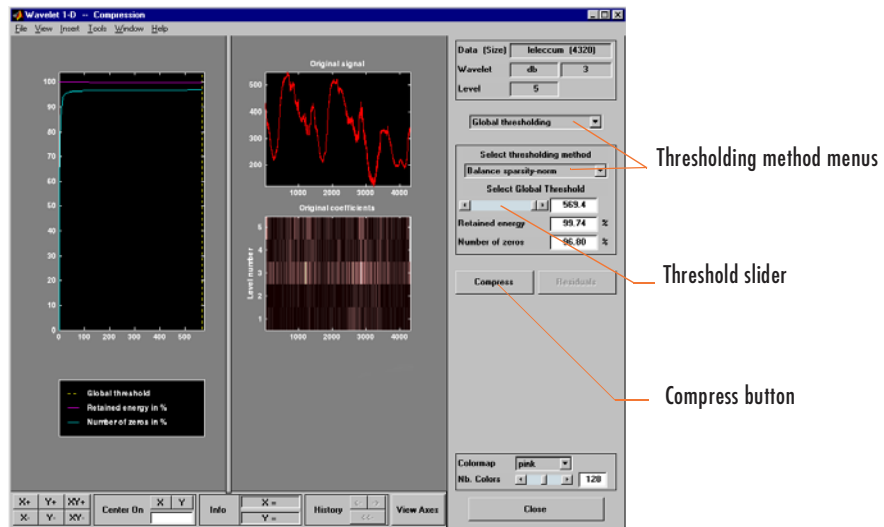
### 9 Compress the signal.

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



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

The **Compression** window appears.



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

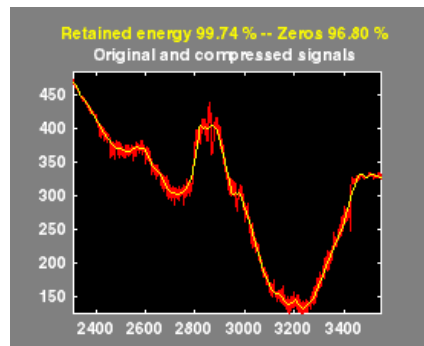
---

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

---

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

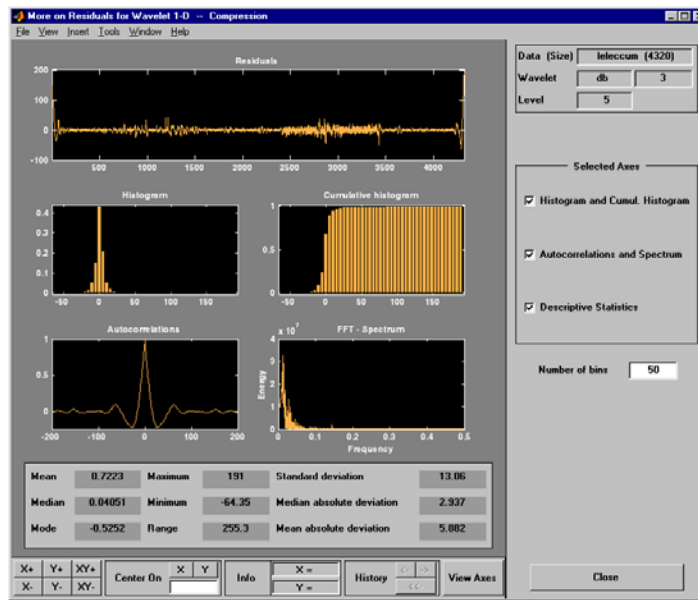
After a pause for computation, the electrical consumption signal is redisplayed in red with the compressed version superimposed in yellow. Below, we've zoomed in to get a closer look at the noisy part of the signal.



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

**10** Show the residuals.

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



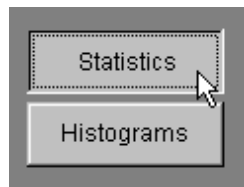
Displayed statistics include measures of tendency (mean, mode, median) and dispersion (range, standard deviation). In addition, the tool provides frequency-distribution diagrams (histograms and cumulative histograms), as well as time-series diagrams: autocorrelation function and spectrum. The same feature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

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

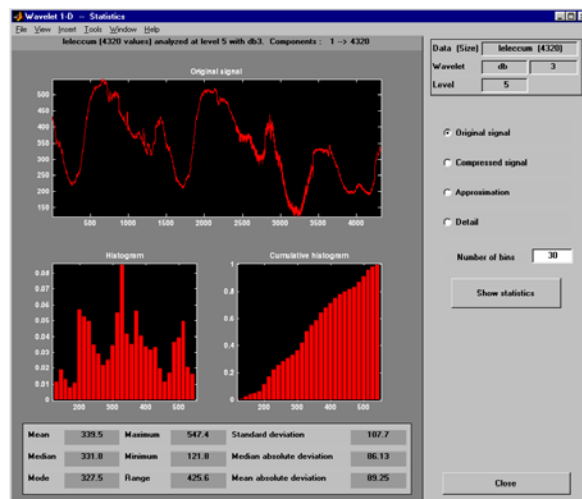
## 11 Show statistics.

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

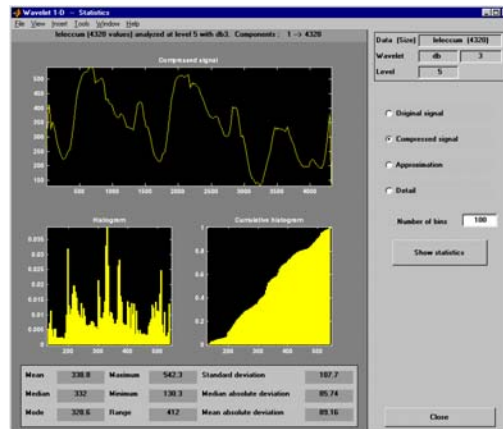
From the **Wavelet 1-D** tool, click the **Statistics** button.



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



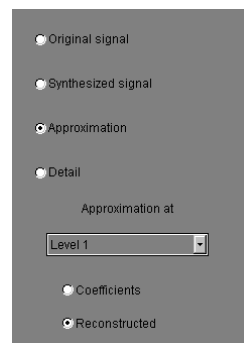
Select the synthesized signal or signal component whose statistics you want to examine. Click the appropriate option button, and then click the **Show Statistics** button. Here, we've chosen to examine the compressed signal using more 100 bins instead of 30, which is the default:



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

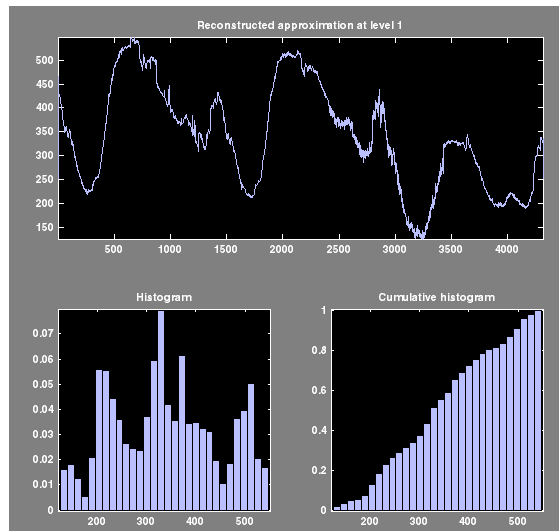
In addition, the tool provides frequency-distribution diagrams (histograms and cumulative histograms). You can plot these histograms separately using the **Histograms** button from the **Wavelets 1-D** window.

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



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



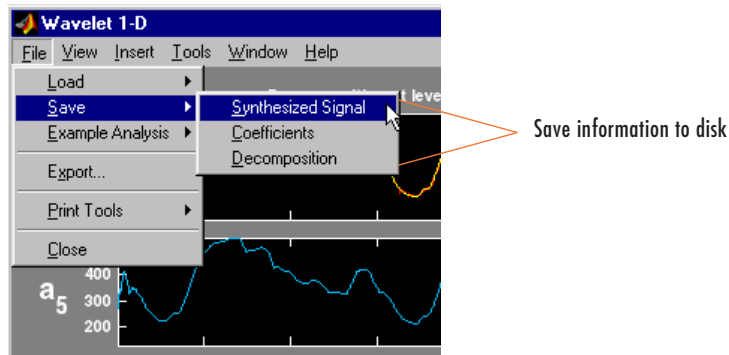


### Importing and Exporting Information from the Graphical Interface

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

#### Saving Information to Disk

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



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

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

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

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

To load the signal into your workspace, simply type

```
load synthsig
```

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

```
whos
```

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

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

For  $i$  from 1 to 5, `thrParams{i}` contains the lower and upper bounds of the thresholding interval and the threshold value (since interval dependent thresholds are allowed, see “One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients” on page 2-158).

For example, for level 1,

```
thrParams{1}

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

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

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

where the variable `valTHR` contains the global threshold:

```
valTHR

valTHR =
    1.2922
```

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

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

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

Consider the example analysis:

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

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

```
load cantor
whos
```

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

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

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

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

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

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

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

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

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

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

---

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

---

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

---

---

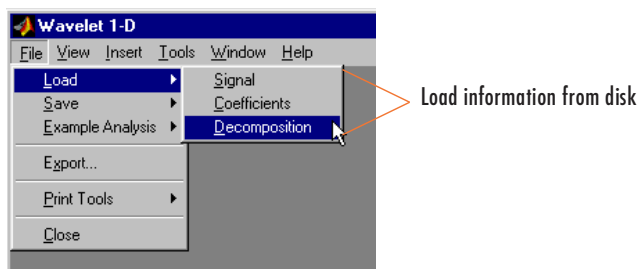
**Note** When saving a synthesized signal, a decomposition or coefficients to a MAT-file, the extension of this file is free. The mat extension is not necessary.

---

### Loading Information into the Wavelet 1-D Tool

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

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



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

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

```
save warma warma
```

The workspace variable `warma` must be a vector.

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

```
sizwarma =
         1         1000
```

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

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

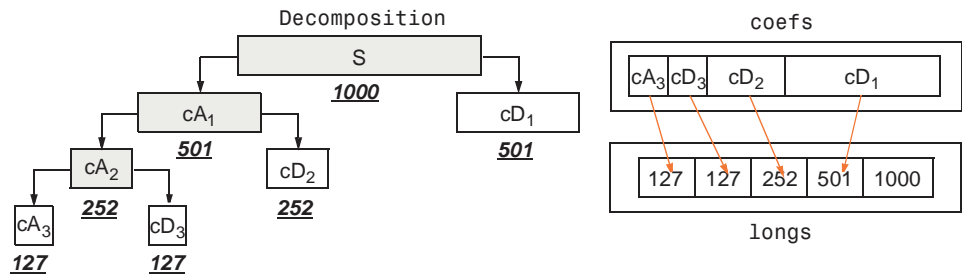
---

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

---

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

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



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

```
save myfile coefs longs
```

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

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

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

The MAT-file contains the following variables.

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

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

```
save myfile coefs longs wave_name
```

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

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

---

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

---



# Two-Dimensional Discrete Wavelet Analysis

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

---

**Note** In this section the presentation and examples use two-dimensional arrays corresponding to indexed image representations. However, the functions described are also available when using truecolor images, which are represented by m-by-n-by-3 arrays of `uint8`. For more information on image formats, see “Wavelets: Working with Images” on page 2-97.

---

### Analysis-Decomposition Functions

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

### Synthesis-Reconstruction Functions

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

### Decomposition Structure Utilities

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

## De-Noising and Compression

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

In this section, you'll learn

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

## Two-Dimensional Analysis Using the Command Line

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

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

### 1 Load an image.

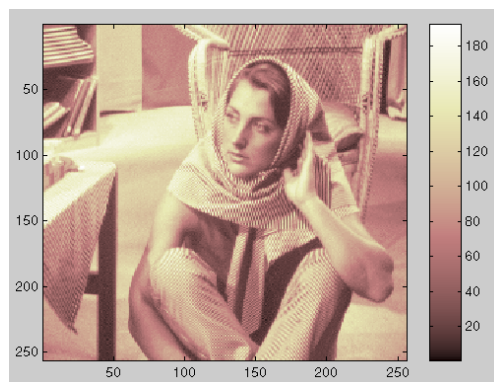
From the MATLAB<sup>®</sup> prompt, type

```
load wbarb;
whos
```

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

### 2 Display the image. Type

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



**3** Convert an indexed image to a grayscale image.

If the colormap is smooth, the wavelet transform can be directly applied to the indexed image; otherwise the indexed image should be converted to grayscale format. For more information, see “Wavelets: Working with Images” on page 2-97.

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

**4** Perform a single-level wavelet decomposition.

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

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

This generates the coefficient matrices of the level-one approximation ( $cA_1$ ) and horizontal, vertical and diagonal details ( $cH_1$ ,  $cV_1$ ,  $cD_1$ , respectively).

**5** Construct and display approximations and details from the coefficients.

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

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

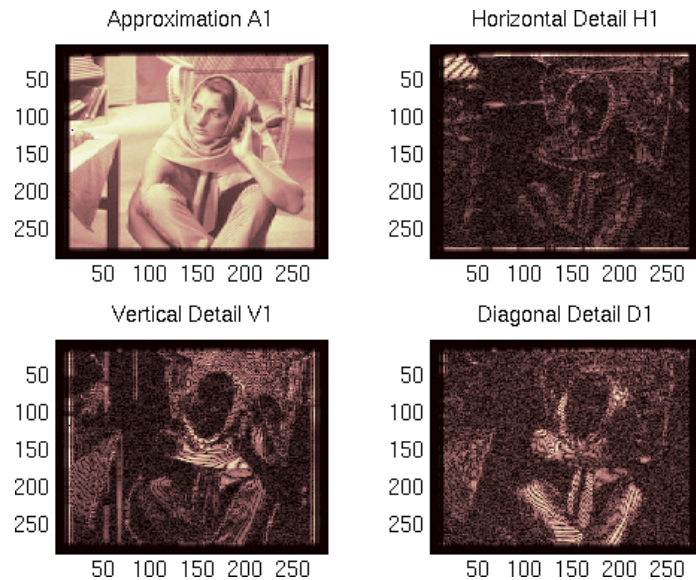
or

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

To display the results of the level 1 decomposition, type

```
colormap(map);  
subplot(2,2,1); image(wcodemat(A1,192));  
title('Approximation A1')
```

```
subplot(2,2,2); image(wcodemat(H1,192));
title('Horizontal Detail H1')
subplot(2,2,3); image(wcodemat(V1,192));
title('Vertical Detail V1')
subplot(2,2,4); image(wcodemat(D1,192));
title('Diagonal Detail D1')
```



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

To find the inverse transform, type

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

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

**7** Perform a multilevel wavelet decomposition.

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

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

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

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

### 8 Extract approximation and detail coefficients.

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

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

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

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

or

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

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

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

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

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

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

```
H1 = wrcoef2('h',C,S,'bior3.7',1);  
V1 = wrcoef2('v',C,S,'bior3.7',1);
```

```

D1 = wrcoef2('d',C,S,'bior3.7',1);
H2 = wrcoef2('h',C,S,'bior3.7',2);
V2 = wrcoef2('v',C,S,'bior3.7',2);
D2 = wrcoef2('d',C,S,'bior3.7',2);

```

**10** Display the results of a multilevel decomposition.

---

**Note** With all the details involved in a multilevel image decomposition, it makes sense to import the decomposition into the **Wavelet 2-D** graphical tool in order to more easily display it. For information on how to do this, see “Loading Decompositions” on page 2-95.

---

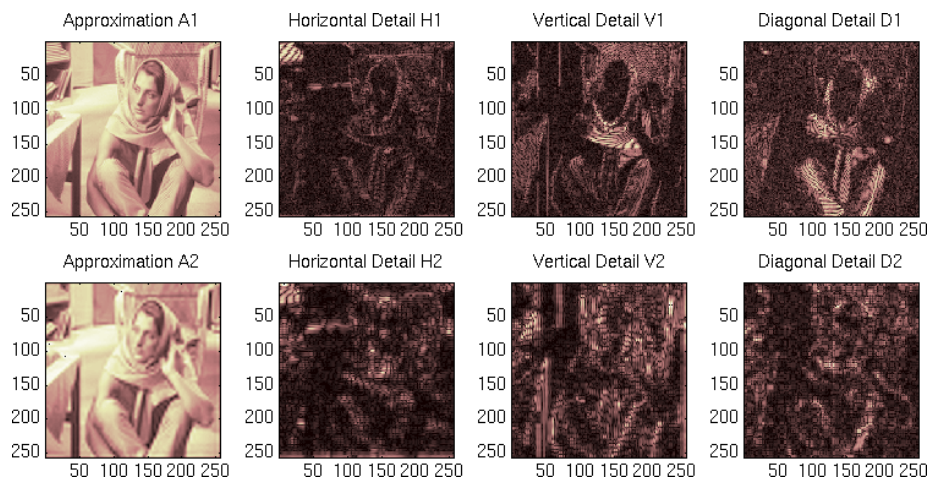
To display the results of the level 2 decomposition, type

```

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

```





### 11 Reconstruct the original image from the multilevel decomposition.

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

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

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

### 12 Compress the image and display it.

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

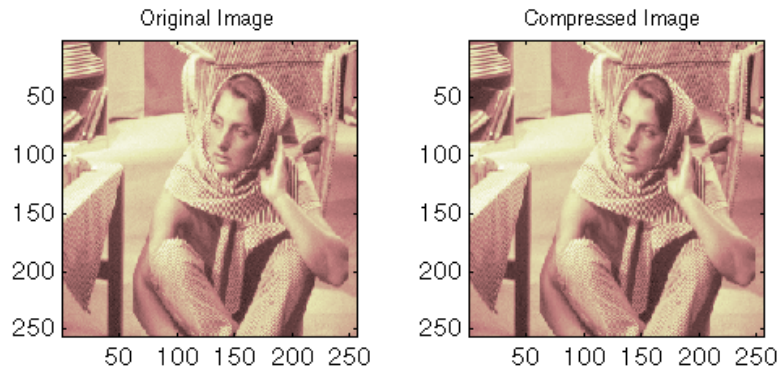
```
[thr,sorh,keepapp] = ddencmp('cmp','wv',X);
[Xcomp,CXC,LXC,PERFO,PERFL2] =
wdencmp('gb1',C,S,'bior3.7',2,thr,sorh,keepapp);
```

Note that we pass in to `wdencmp` the results of the decomposition (*C* and *S*) we calculated in step 7 on page 2-71. We also specify the `bior3.7` wavelets, because we used this wavelet to perform the original analysis. Finally, we

specify the global thresholding option 'gb1'. See `ddencmp` and `wdecmp` reference pages for more information about the use of these commands.

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

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



```
PERF0
  PERF0 =
    49.8076
```

```
PERFL2
  PERFL2 =
    99.9817
```

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

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

### **Two-Dimensional Analysis Using the Graphical Interface**

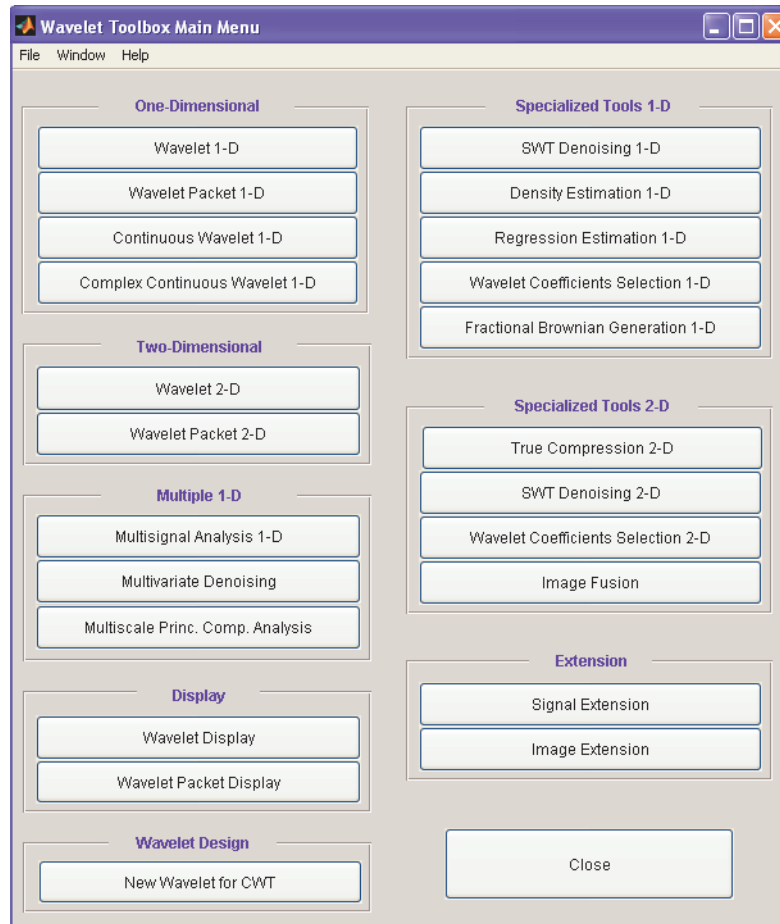
In this section we explore the same image as in the previous section, but we use the graphical interface tools to analyze the image.

- 1 Start the 2-D Wavelet Analysis Tool.

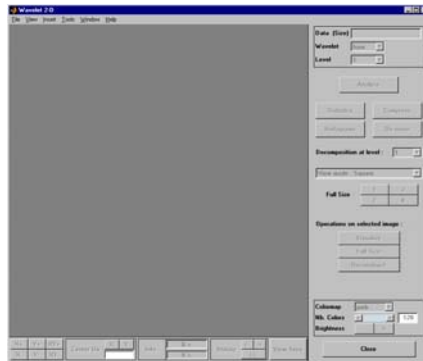
From the MATLAB prompt, type

```
wavemenu
```

The **Wavelet Tool Main Menu** appears.

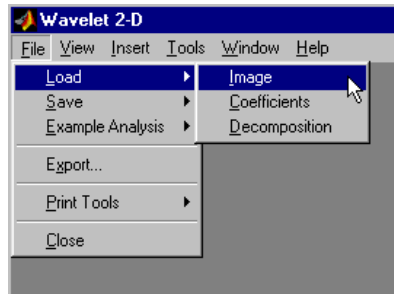


Click the **Wavelet 2-D** menu item. The discrete wavelet analysis tool for two-dimensional image data appears.

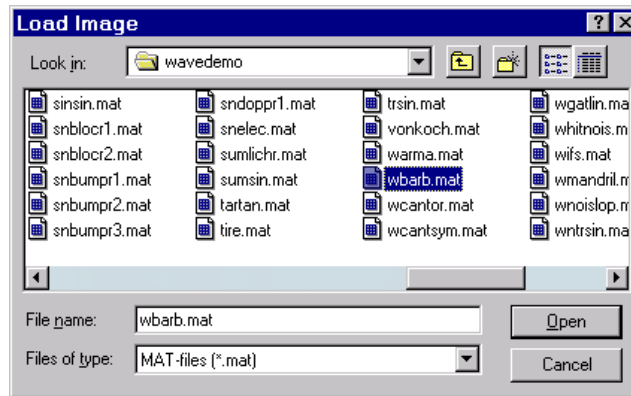


### 2 Load an image.

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



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

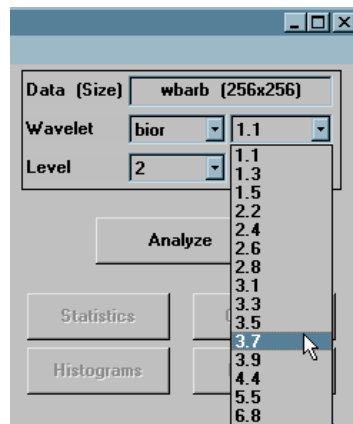


The image is loaded into the **Wavelet 2-D** tool.

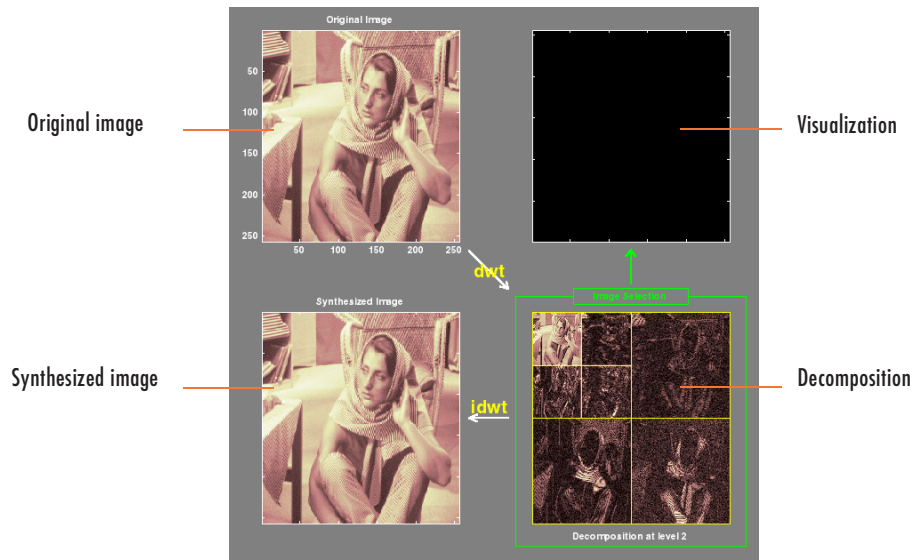
### 3 Analyze the image.

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

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



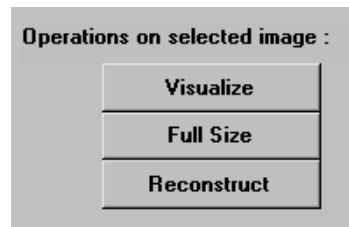
Click the **Analyze** button. After a pause for computation, the **Wavelet 2-D** tool displays its analysis.



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

Click on any decomposition component in the lower right window.

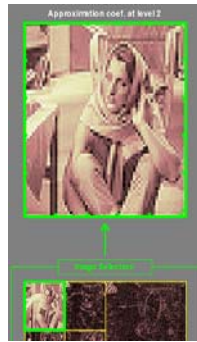
A green border highlights the selected component. At the lower right of the **Wavelet 2-D** window, there is a set of three buttons labeled “Operations on selected image.” Note that if you click again on the same component, you’ll deselect it and the green border disappear.



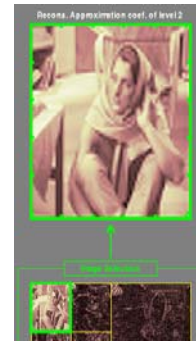
Click the **Visualize** button.

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



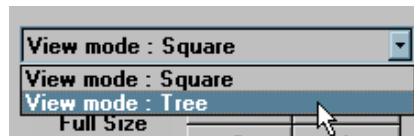


Approximation coefficients cA2

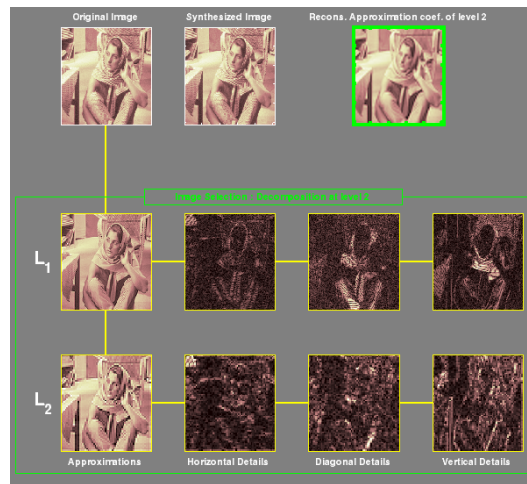


Reconstructed Approximation A2

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

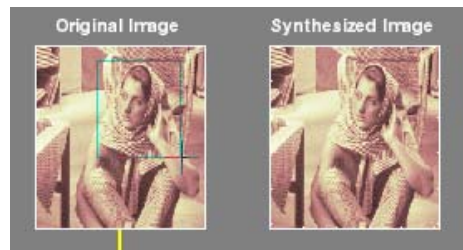


Your display changes to reveal the following.



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

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



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



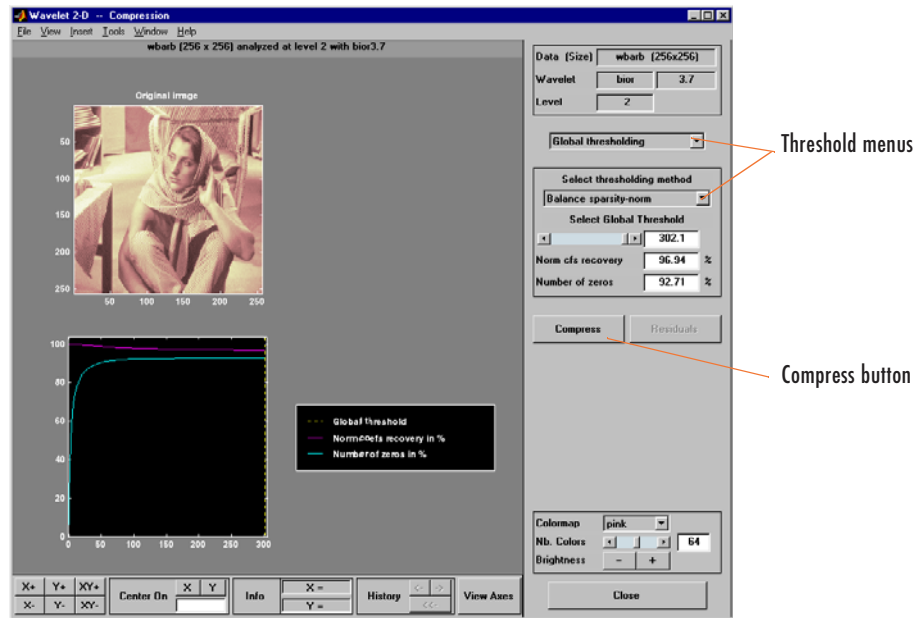
The **Wavelet 2-D** tool enlarges the displayed images.



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

#### 4 Compress the image

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

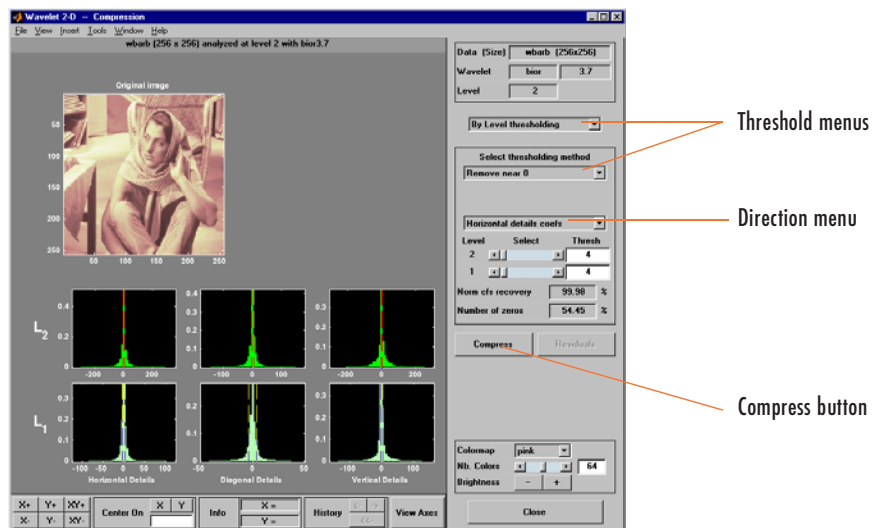


The tool automatically selects thresholding levels to provide a good initial balance between retaining the image's energy while minimizing the number of coefficients needed to represent the image.

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

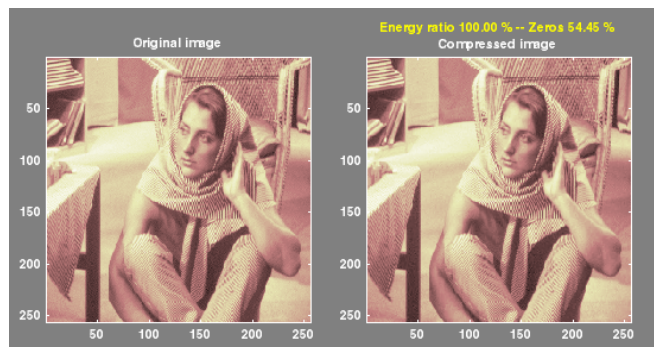
For this example, select the **By Level thresholding** option and select the **Remove near 0** method from the **Select thresholding method** menu.

The following window is displayed.



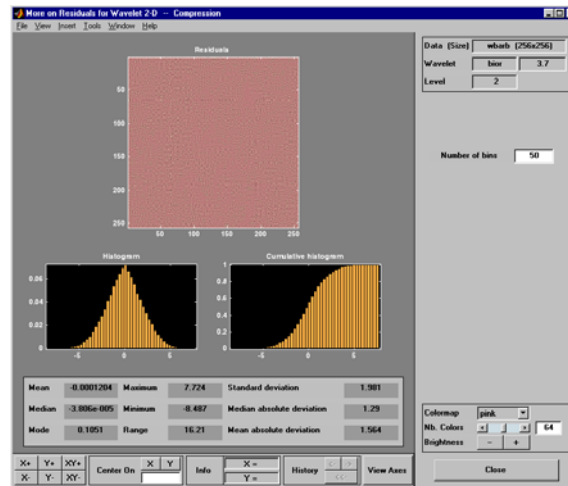
Select from the direction menu whether you want to adjust thresholds for horizontal, diagonal or vertical details. To make the actual adjustments for each level, use the sliders or use the left mouse button to directly drag the yellow vertical lines.

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



5 Show the residuals.

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



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

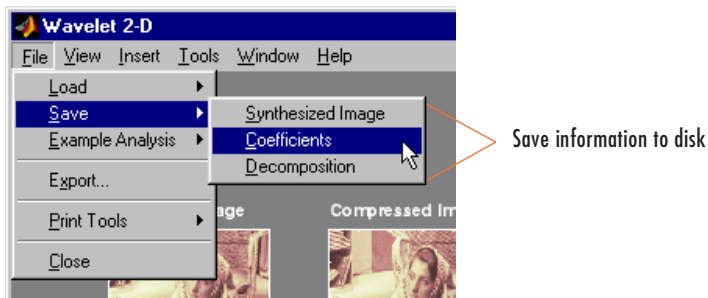
**Note** The statistics displayed in the above figure are related to the displayed image but not to the original one. Usually this information is the same, but in some cases, edge effects may cause the original image to be cropped slightly. To see the exact statistics, use the command line functions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

## Importing and Exporting Information from the Graphical Interface

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

### Saving Information to Disk

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



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

For example, load the example analysis:

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

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

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

To load the image into your workspace, type

```
load symage
whos
```

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

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

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

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

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

Consider the example analysis:

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

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

```
load cfsdurer
whos
```

Name	Size	Bytes	Class
coefs	1x142299	1138392	double array
map	64x3	1536	double array
sizes	5x2	80	double array



Name	Size	Bytes	Class
valTHR	0x0	0	double array
wname	1x4	8	char array

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

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

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

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

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

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

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

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

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

Name	Size	Bytes	Class
coefs	1x142299	1138392	double array
data_name	1x6	12	char array
map	64x3	1536	double array

---

<b>Name</b>	<b>Size</b>	<b>Bytes</b>	<b>Class</b>
sizes	5x2	80	double array
valTHR	0x0	0	double array
wave_name	1x4	8	char array

---

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

---

**Note** Save options are also available when performing de-noising or compression inside the **Wavelet 2-D** tool. In the **Wavelet 2-D De-noising** window, you can save de-noised image and decomposition. The same holds true for the **Wavelet 2-D Compression** window. This way, you can save many different trials from inside the De-noising and Compression windows without going back to the main **Wavelet 2-D** window during a fine-tuning process.

---

---

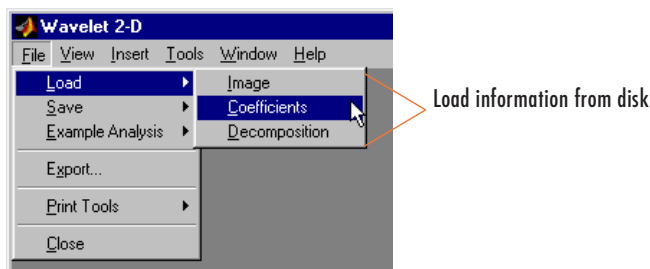
**Note** When saving a synthesized image, a decomposition, or coefficients to a MAT-file, the extension of this file is free. The `mat` extension is not necessary.

---

### Loading Information into the Wavelet 2-D Tool

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

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



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

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

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

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

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

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

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

---

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

---

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

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

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

---

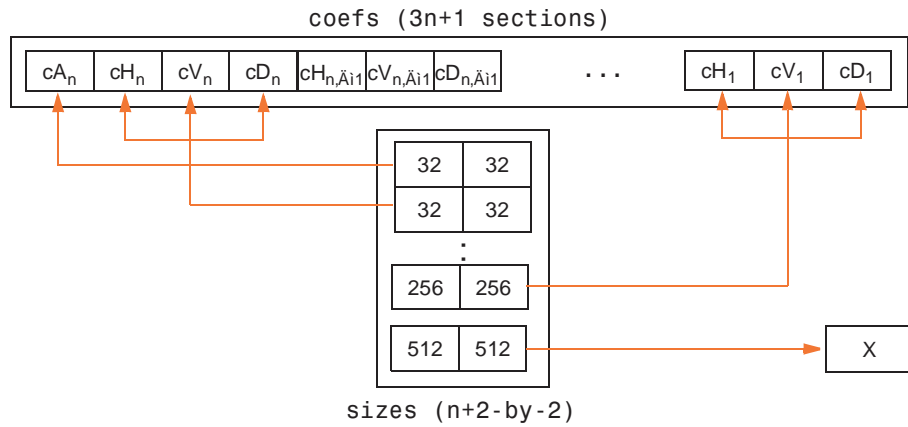
**Note** The first two-dimensional variable encountered in the file (except the variable map, which is reserved for the colormap) is considered the image. Variables are inspected in alphabetical order.

---

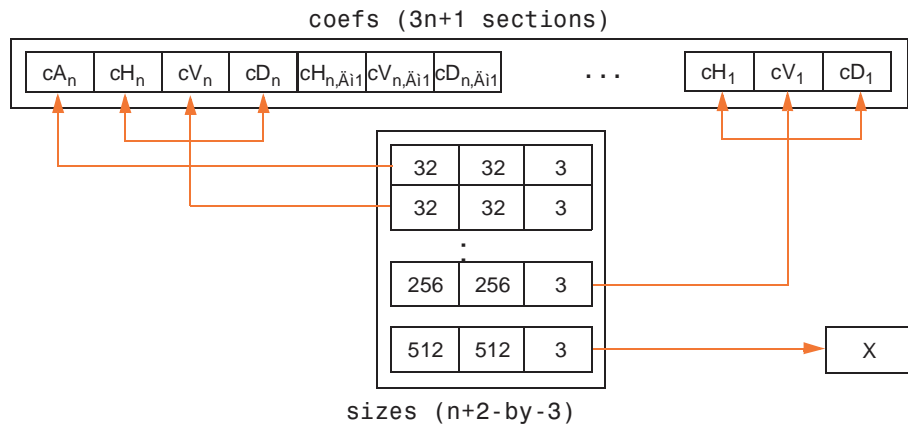
**Loading Discrete Wavelet Transform Coefficients.** To load discrete wavelet transform (DWT) coefficients into the **Wavelet 2-D** tool, first save the appropriate data in a MAT-file, which must contain at least the two variables:

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

For an indexed image the matrix sizes is a  $(n+2\text{-by-}2)$  array:



For a truecolor image, the matrix sizes is a  $(n+2\text{-by-}3)$ :



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

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

```
save myfile coefs sizes
```

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

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

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

The MAT-file contains these variables.

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

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

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

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

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

---

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

---

## Wavelets: Working with Images

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

### Understanding Images in the MATLAB® Environment

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

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

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

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

### Indexed Images

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

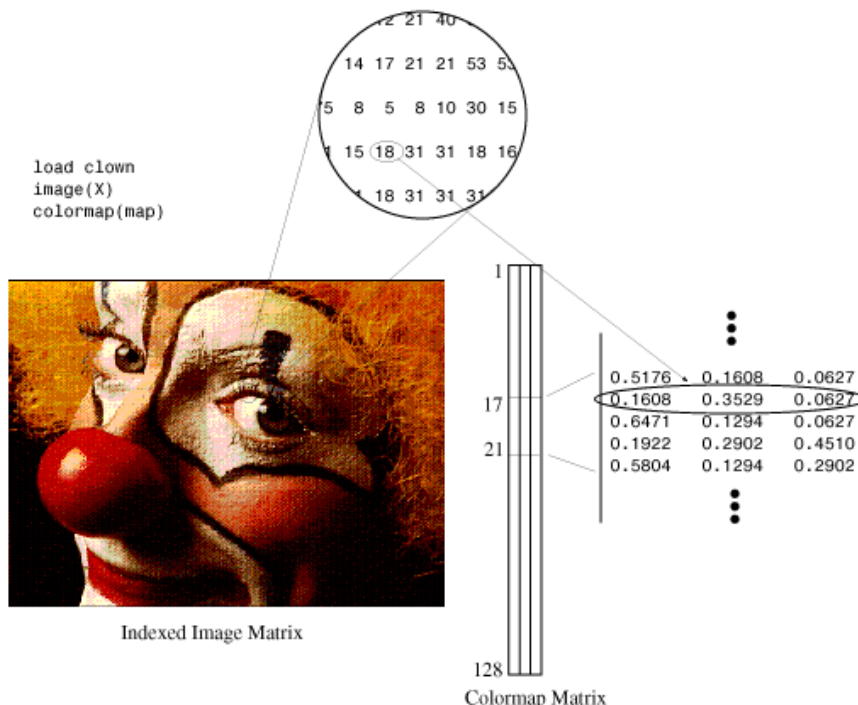
The size of the colormap matrix is n-by-3 for an image containing n colors. Each row of the colormap matrix is a 1-by-3 red, green, blue (RGB) color vector

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

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



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



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

## Wavelet Decomposition of Indexed Images

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

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

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

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

### How Decompositions Are Displayed

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

To display these images in a suitable way, the graphical user interface tools follow these rules:

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

### RGB (Truecolor) Images

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

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

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

## Wavelet Decomposition of Truecolor Images

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

## Other Images

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

These tools are:

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

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

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

## Image Conversion

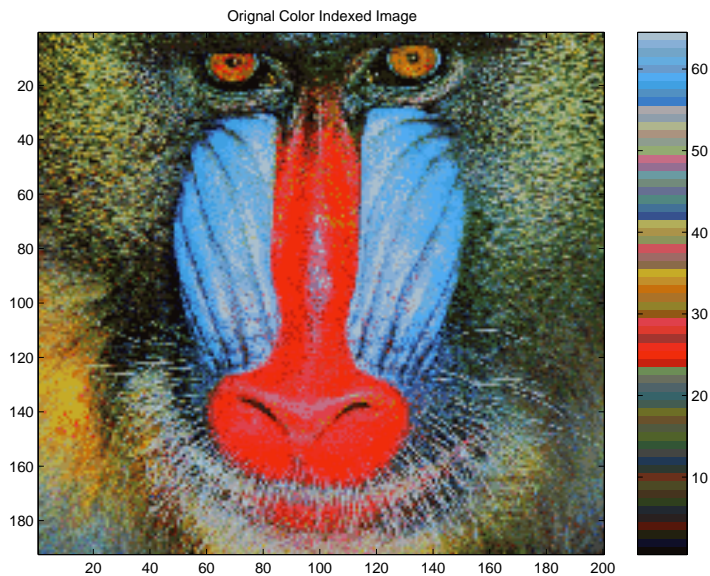
Image Processing Toolbox™ software provides a comprehensive set of functions that let you easily convert between image types. If you do not have Image Processing Toolbox software, the examples below demonstrate how this conversion may be performed using basic MATLAB commands.

### Example 1: Converting Color Indexed Images

```
load xpmndr11
whos
```

Name	Size	Bytes	Class
X2	192x200	307200	double array
map	64x3	1536	double array

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



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

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

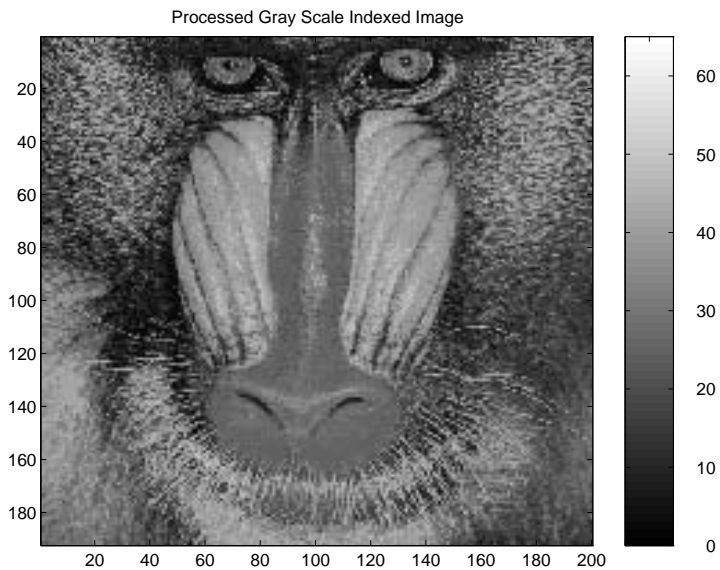
```
R = map(X2,1); R = reshape(R,size(X2));
G = map(X2,2); G = reshape(G,size(X2));
B = map(X2,3); B = reshape(B,size(X2));
```

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

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

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

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



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

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

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

### Example 2: Converting an RGB TIF Image

Suppose the file `myImage.tif` contains an RGB image (noncompressed) of size  $S_1 \times S_2$ . Use the following commands to convert this image:

```
A = imread('myImage.tif');
```

```
% A is an S1xS2x3 array of uint8.  
  
A = double(A);  
Xrgb = 0.2990*A(:,:,1) + 0.5870*A(:,:,2) + 0.1140*A(:,:,3);  
NbColors = 255;  
X = wcodemat(Xrgb,NbColors);  
map = pink(NbColors);
```

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

## One-Dimensional Discrete Stationary Wavelet Analysis

This section takes you through the features of one-dimensional discrete stationary wavelet analysis using the Wavelet Toolbox™ software. For more information see “Discrete Stationary Wavelet Transform (SWT)” on page 6-44.

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

### Analysis-Decomposition Functions

Function Name	Purpose
swt	Decomposition

### Synthesis-Reconstruction Functions

Function Name	Purpose
iswt	Reconstruction

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

In this section, you'll learn to

- Load a signal
- Perform a stationary wavelet decomposition of a signal
- Construct approximations and details from the coefficients
- Display the approximation and detail at level 1
- Regenerate a signal by using inverse stationary wavelet transform
- Perform a multilevel stationary wavelet decomposition of a signal
- Reconstruct the level 3 approximation
- Reconstruct the level 1, 2, and 3 details
- Reconstruct the level 1 and 2 approximations

- Display the results of a decomposition
- Reconstruct the original signal from the level 3 decomposition
- Remove noise from a signal

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

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

## One-Dimensional Analysis Using the Command Line

This example involves a noisy Doppler test signal.

- 1 Load a signal.

From the MATLAB<sup>®</sup> prompt, type

```
load noisdopp
```

- 2 Set the variables. Type

```
s = noisdopp;
```

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

- 3 Perform a single-level Stationary Wavelet Decomposition.

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

Type

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



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

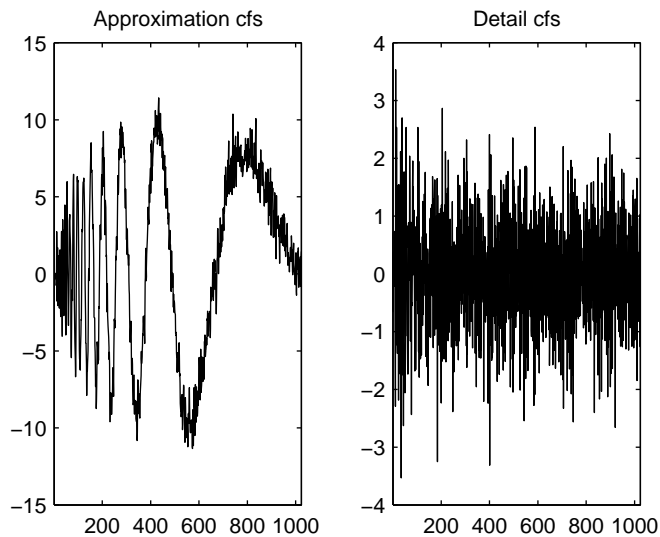
```
whos
```

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

#### 4 Display the coefficients of approximation and detail.

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

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



**5** Regenerate the signal by Inverse Stationary Wavelet Transform.

To find the inverse transform, type

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

To check the perfect reconstruction, type

```
err = norm(s-A0)
```

```
err =
```

```
2.1450e-14
```

**6** Construct and display approximation and detail from the coefficients.

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

```
nulcfs = zeros(size(swa));
```

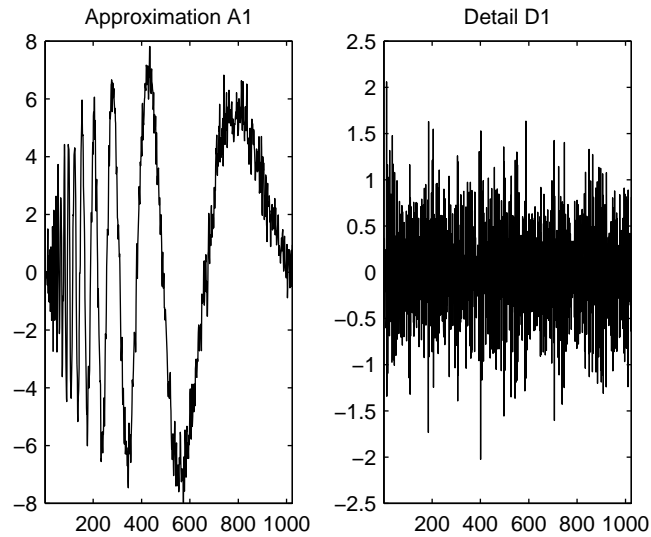
```
A1 = iswt(swa,nulcfs,'db1');
```

```
D1 = iswt(nulcfs,swd,'db1');
```

To display the approximation and detail at level 1, type

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

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



### 7 Perform a multilevel Stationary Wavelet Decomposition.

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

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

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

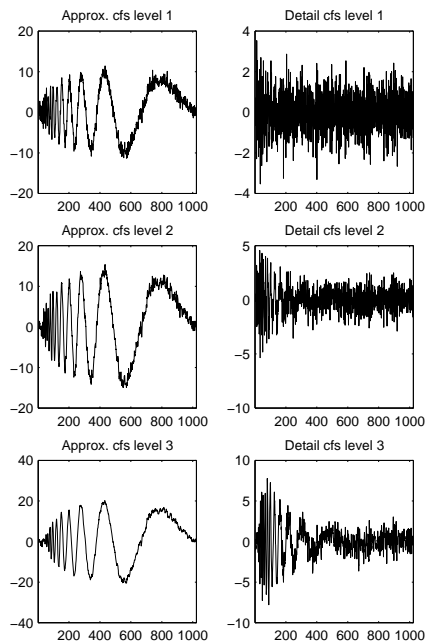
```
clear A0 A1 D1 err nulcfs  
whos
```

<b>Name</b>	<b>Size</b>	<b>Bytes</b>	<b>Class</b>
noisdopp	1x1024	8192	double array
s	1x1024	8192	double array
swa	3x1024	24576	double array
swd	3x1024	24576	double array

### 8 Display the coefficients of approximations and details.

To display the coefficients of approximations and details, type

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



### 9 Reconstruct approximation at Level 3 From coefficients.

To reconstruct the approximation at level 3, type

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

### 10 Reconstruct details from coefficients.

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

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

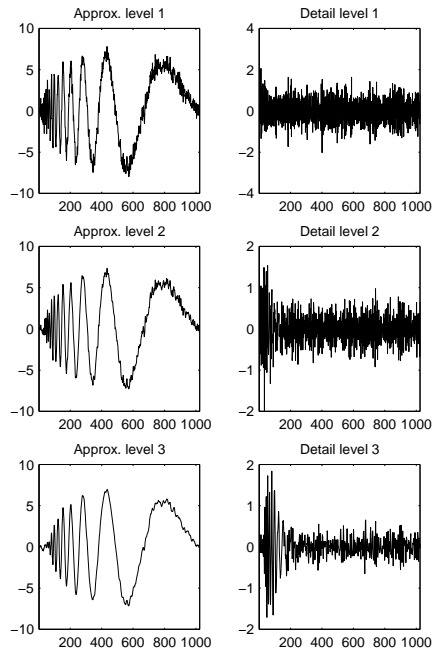
- 11** Reconstruct and display approximations at Levels 1 and 2 from approximation at Level 3 and details at Levels 2 and 3.

To reconstruct the approximations at levels 2 and 3, type

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

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

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



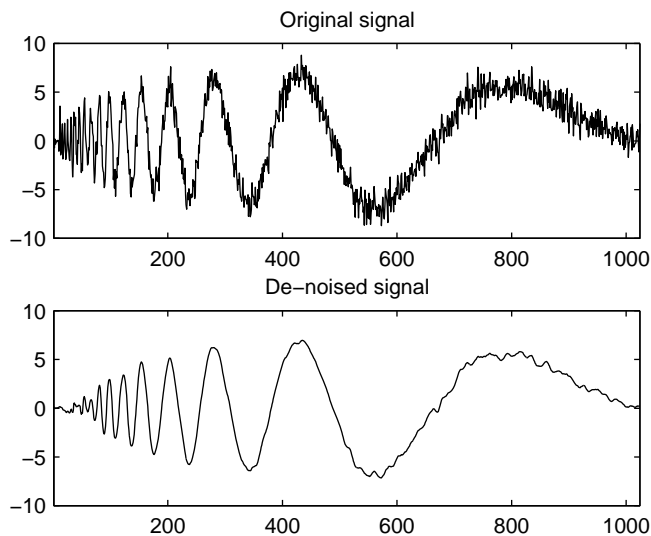
**12** Remove noise by thresholding.

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

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

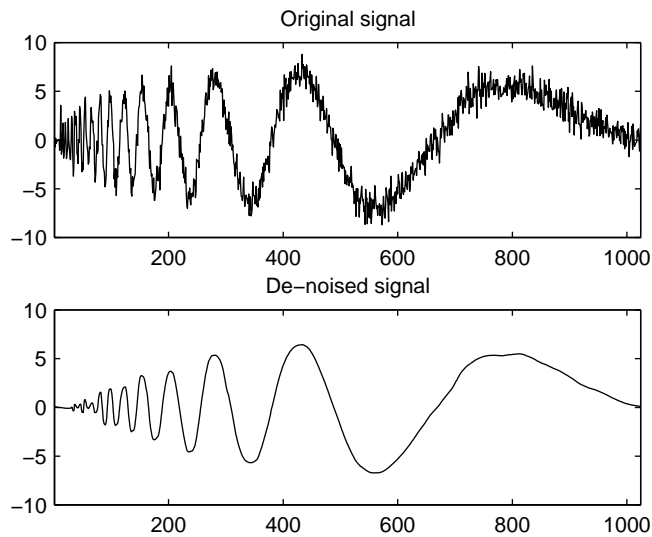
To display both the original and de-noised signals, type

```
subplot(2,1,1), plot(s);  
title('Original signal')  
subplot(2,1,2), plot(clean);  
title('De-noised signal')
```



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

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



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

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

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



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

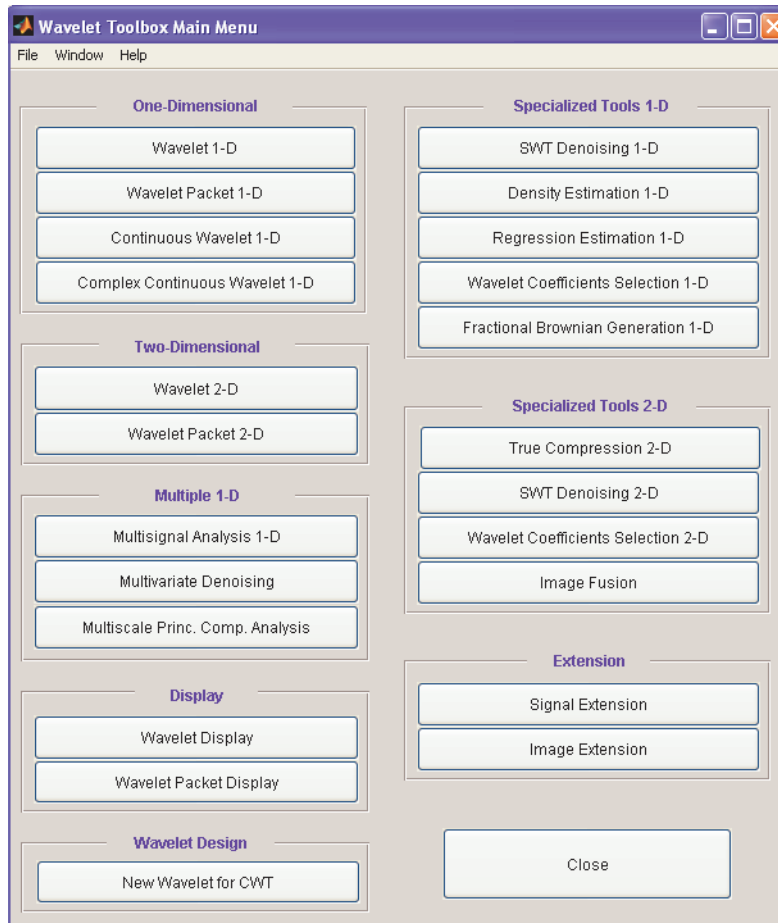
Now we explore a strategy to de-noise signals, based on the one-dimensional stationary wavelet analysis using the graphical interface tools. The basic idea is to average many slightly different discrete wavelet analyses.

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

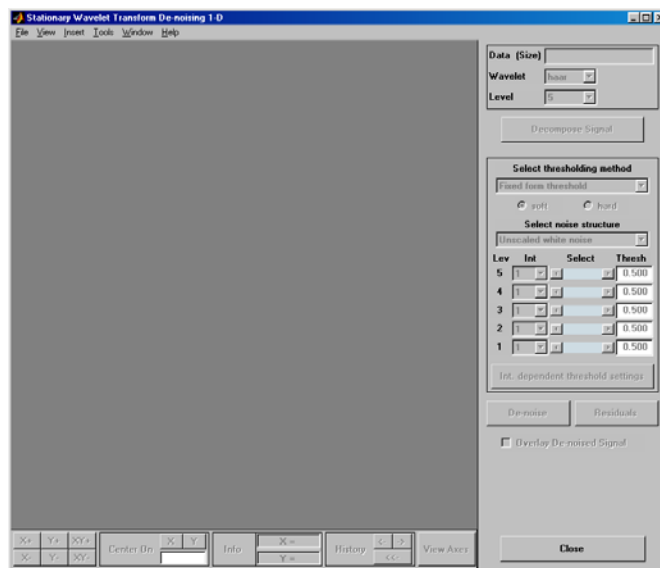
From the MATLAB prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.



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

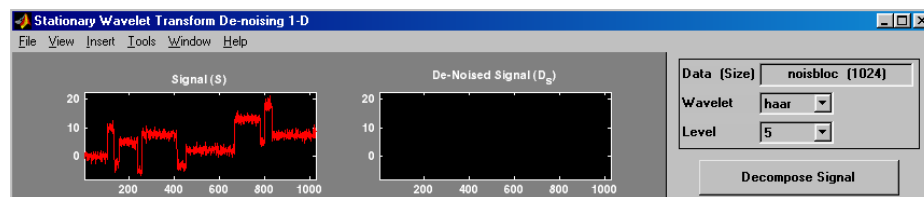


### 2 Load data.

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

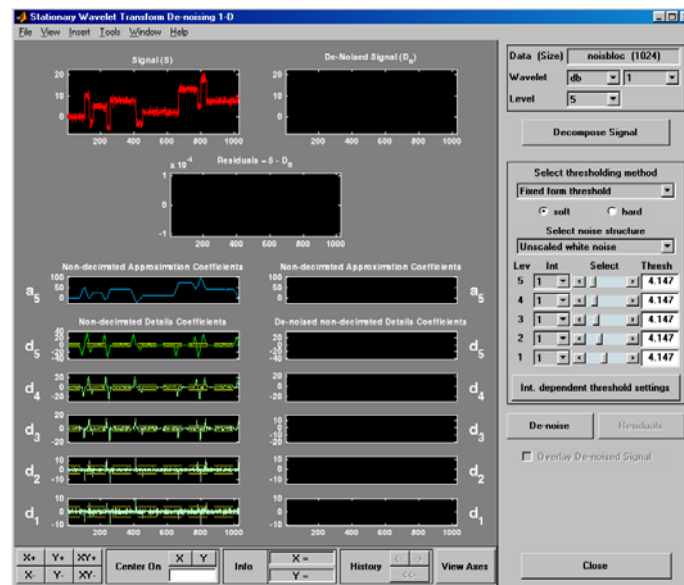
When the **Load Signal** dialog box appears, select the MAT-file `noisbloc.mat`, which should reside in the MATLAB directory `toolbox/wavelet/wavedemo`.

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



### 3 Perform a Stationary Wavelet Decomposition.

Select the db1 wavelet from the **Wavelet** menu and select **5** from the **Level** menu, and then click the **Decompose Signal** button. After a pause for computation, the tool displays the stationary wavelet approximation and detail coefficients of the decomposition. These are also called nondecimated coefficients since they are obtained using the same scheme as for the DWT, but omitting the decimation step (see “Fast Wavelet Transform (FWT) Algorithm” on page 6-19).



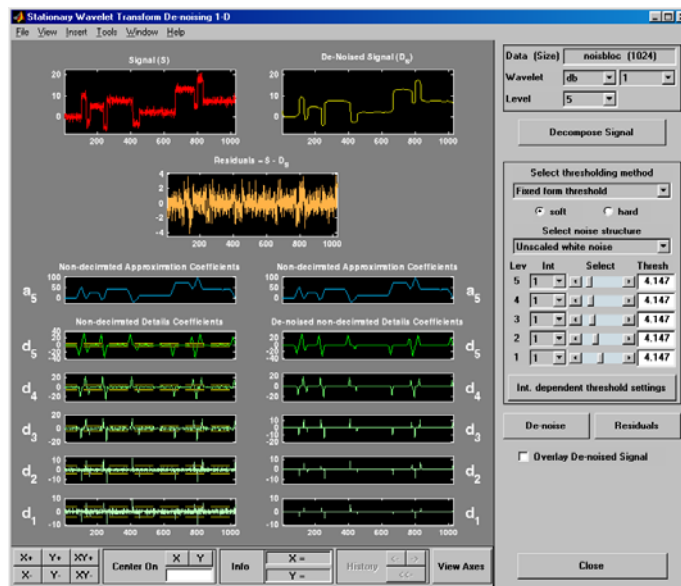
### 4 De-noise the signal using the Stationary Wavelet Transform.

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

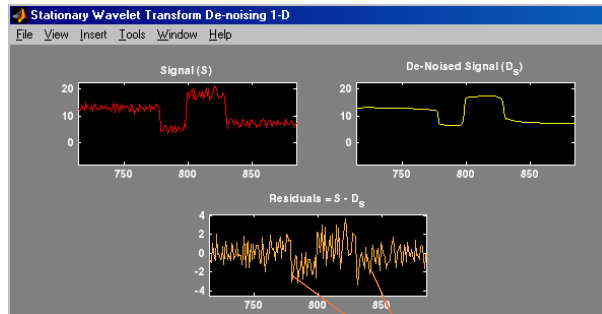
of the window. The yellow dotted lines can also be dragged directly using the left mouse button over the graphs.

Note that the approximation coefficients are not thresholded.

Click the **De-noise** button.

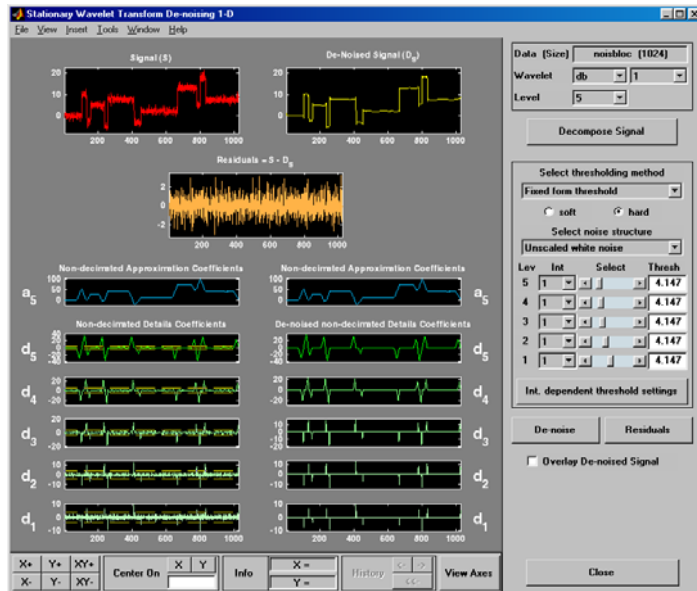


The result is quite satisfactory, but seems to be oversmoothed around the discontinuities of the signal. This can be seen by looking at the residuals, and zooming on a breakdown point, for example around position 800.



The residuals clearly contain some signal information.

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



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

### Importing and Exporting from the GUI

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

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

```
load dnoibloc
whos
```

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

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

```
wname
  
wname =
    db1
```

and the level dependent thresholds are encoded in `thrParams`, which is a cell array of length 5 (the level of the decomposition). For  $i$  from 1 to 5, `thrParams{i}` contains the lower and upper bounds of the interval of thresholding and the threshold value (since interval dependent thresholds are allowed). For more information, see “One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients” on page 2-158.

For example, for level 1,

```
thrParams{1}
```

```
ans =
```

```
1.0e+03 *
```

```
0.0010 1.0240 0.0041
```

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



## Two-Dimensional Discrete Stationary Wavelet Analysis

This section takes you through the features of two-dimensional discrete stationary wavelet analysis using the Wavelet Toolbox™ software. For more information, see “Available Methods for De-Noising, Estimation, and Compression Using GUI Tools” on page 6-122.

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

### Analysis-Decomposition Function

Function Name	Purpose
swt2	Decomposition

### Synthesis-Reconstruction Function

Function Name	Purpose
iswt2	Reconstruction

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

In this section, you'll learn to

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

## Two-Dimensional Analysis Using the Command Line

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

---

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

---

This example involves a image containing noise.

### 1 Load an image.

From the MATLAB<sup>®</sup> prompt, type

```
load noiswom
whos
```

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

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

### 2 Perform a single-level Stationary Wavelet Decomposition.

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

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

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

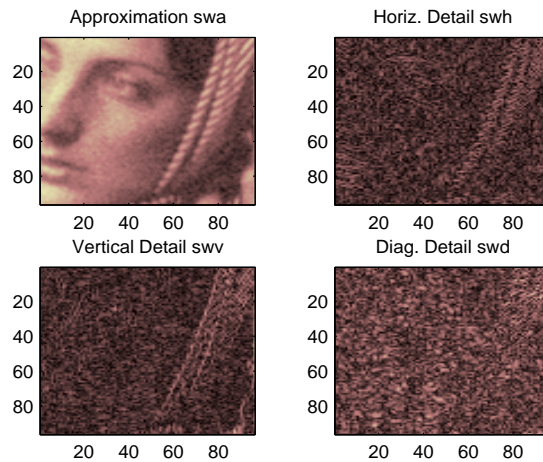
```
whos
```

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

### 3 Display the coefficients of approximation and details.

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

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



**4** Regenerate the image by Inverse Stationary Wavelet Transform.

To find the inverse transform, type

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

To check the perfect reconstruction, type

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

```
err =  
1.1369e-13
```

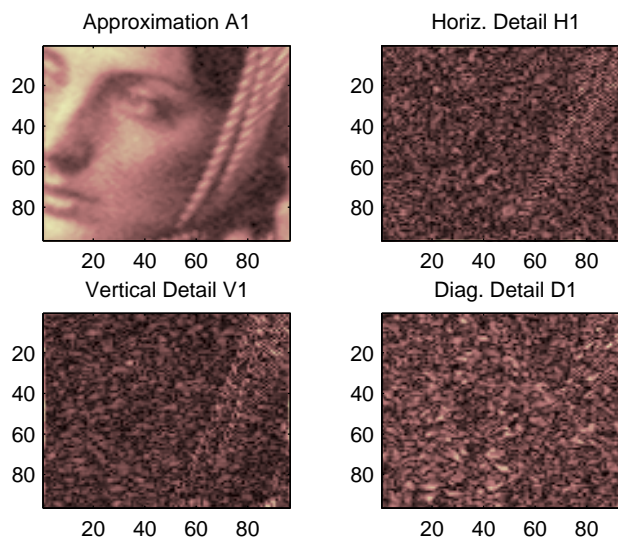
**5** Construct and display approximation and details from the coefficients.

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

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

To display the approximation and details at level 1, type

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



### 6 Perform a multilevel Stationary Wavelet Decomposition.

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

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

This generates the coefficients of the approximations at levels 1, 2, and 3 (swa) and the coefficients of the details (swh, swv and swd). Observe that the

matrices `swa(:,:,i)`, `swh(:,:,i)`, `swv(:,:,i)`, and `swd(:,:,i)` for a given level `i` are of size-the-image size. Type

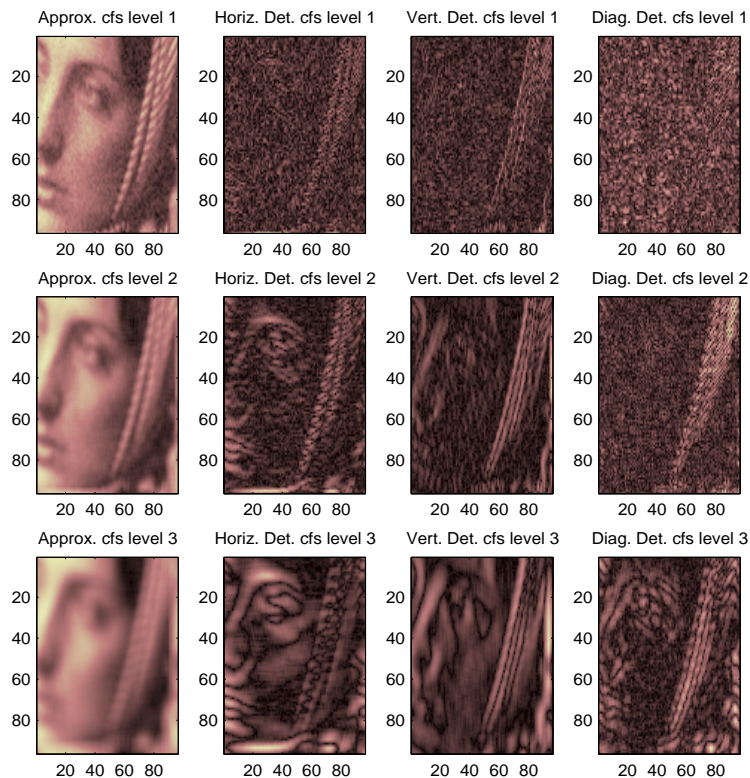
```
clear A0 A1 D1 H1 V1 err nulcfs
whos
```

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

## 7 Display the coefficients of approximations and details.

To display the coefficients of approximations and details, type

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



### 8 Reconstruct approximation at Level 3 and details from coefficients.

To reconstruct the approximation at level 3, type

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

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

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

```

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

```

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

To reconstruct the approximations at levels 2 and 3, type

```

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

```

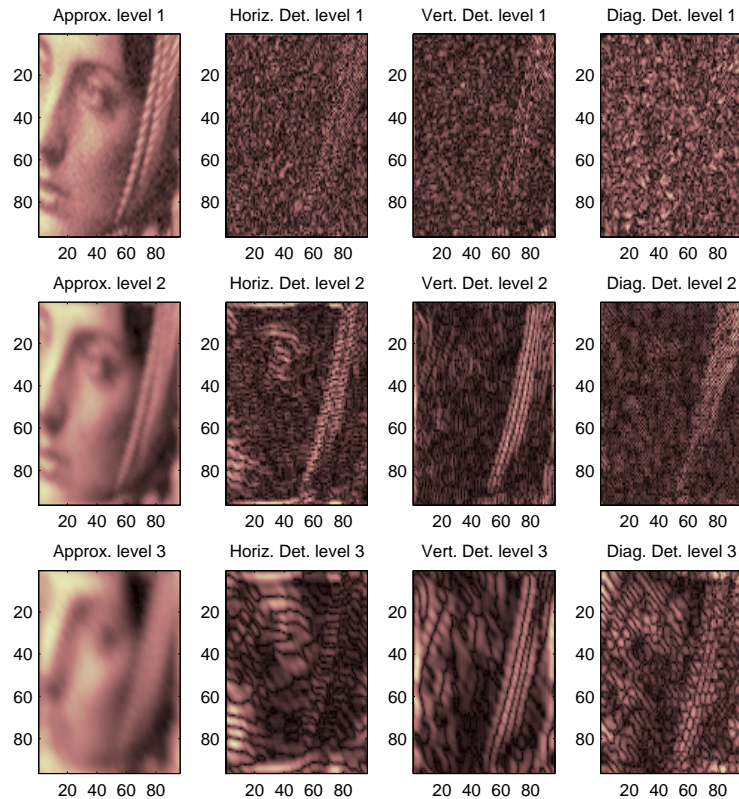
To display the approximations and details at levels 1, 2, and 3, type

```

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

```





### 10 Remove noise by thresholding.

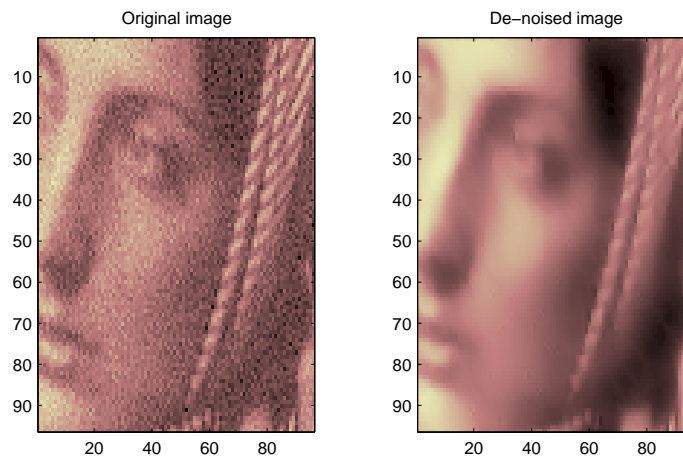
To de-noise an image, use the threshold value we find using the GUI tool (see the next section), use the `wthresh` command to perform the actual thresholding of the detail coefficients, and then use the `iswt2` command to obtain the de-noised image.

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

```
clean = iswt2(swa,dswh,dswv,dswd,'db1');
```

To display both the original and de-noised images, type

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



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

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

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

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

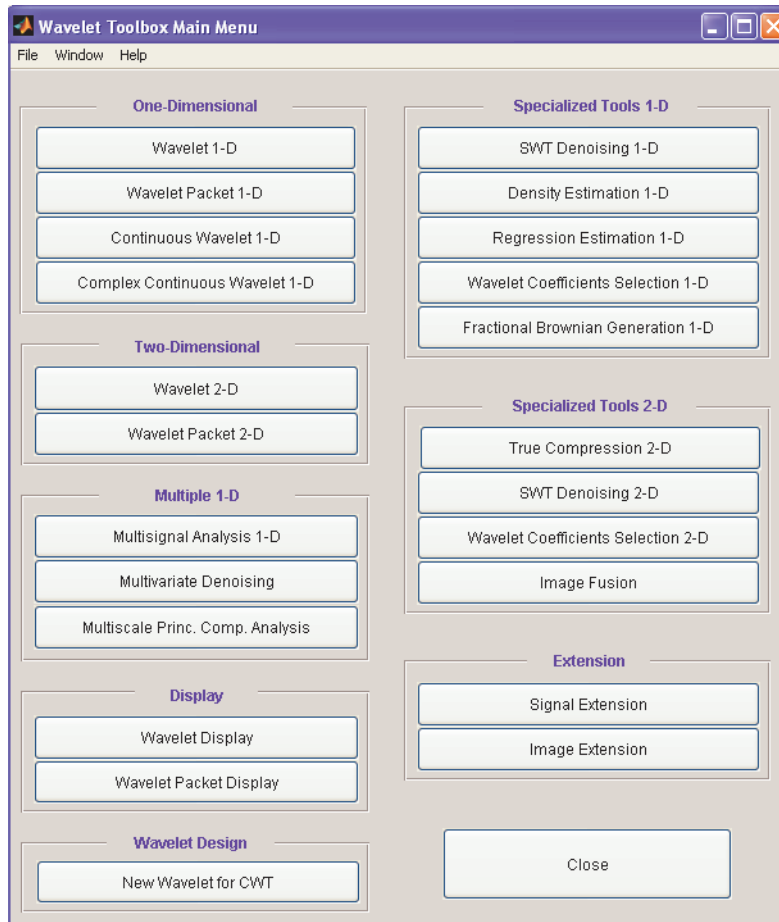
In this section, we explore a strategy for de-noising images based on the two-dimensional stationary wavelet analysis using the graphical interface tools. The basic idea is to average many slightly different discrete wavelet analyses.

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

From the MATLAB prompt, type

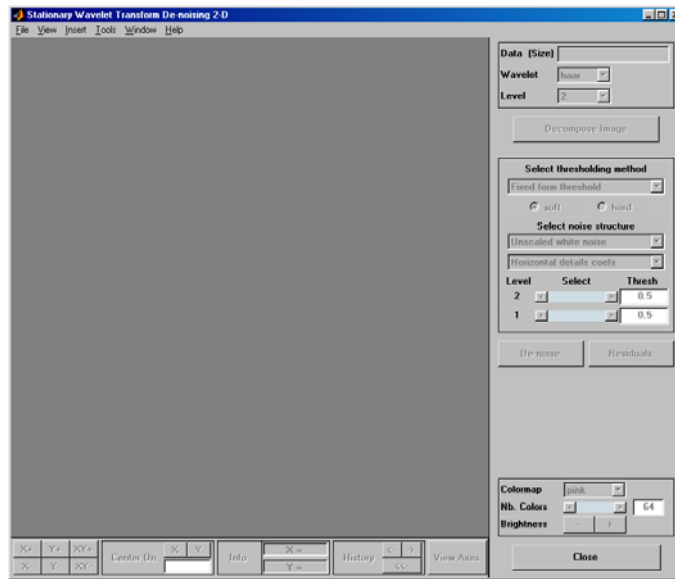
```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears:



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

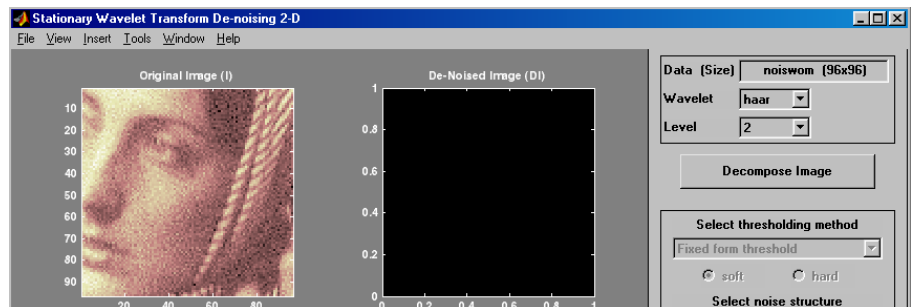
The discrete stationary wavelet transform de-noising tool for images appears.



### 2 Load data.

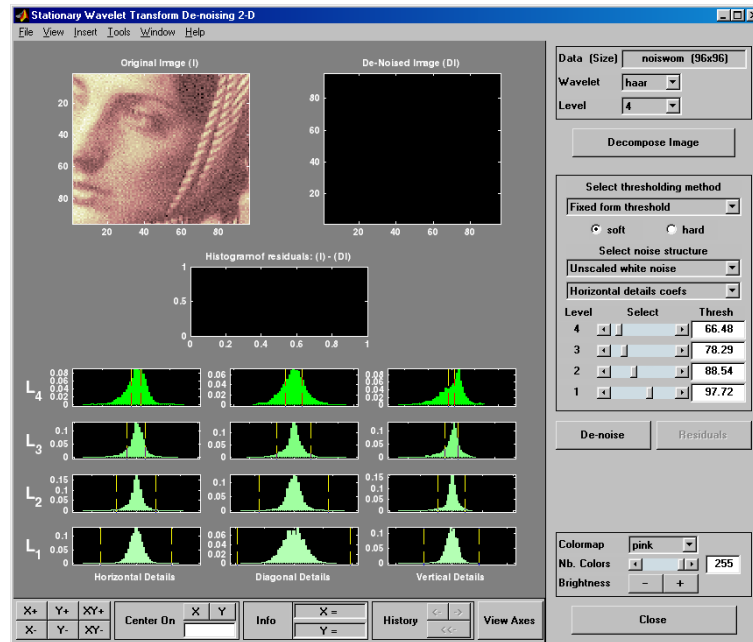
From the **File** menu, choose the **Load Image** option.

When the **Load Image** dialog box appears, select the MAT-file `noiswom.mat`, which should reside in the MATLAB directory `toolbox/wavelet/wavedemo`. Click the **OK** button. The noisy woman image is loaded into the **SWT De-noising 2-D** tool.



### 3 Perform a Stationary Wavelet Decomposition.

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



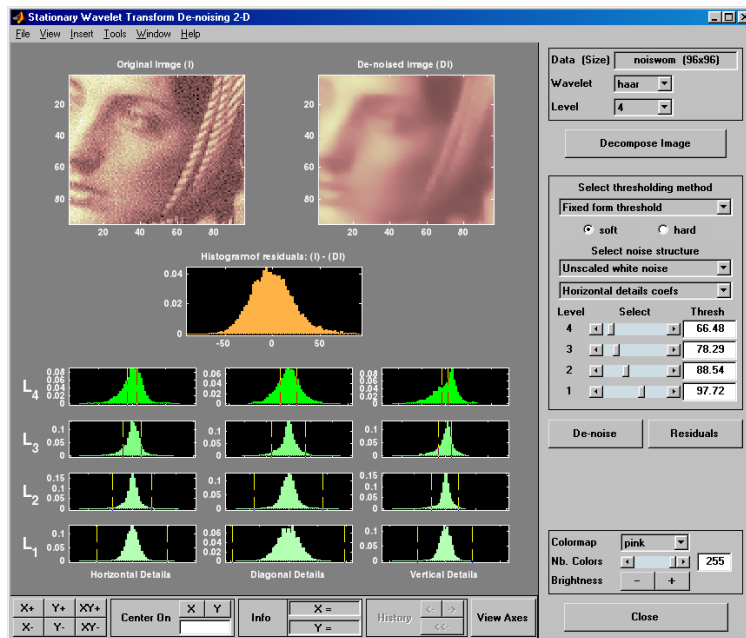
The tool displays the histograms of the stationary wavelet detail coefficients of the image on the left of the window. These histograms are organized as follows:

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

### 4 De-noise the image using the Stationary Wavelet Transform.

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

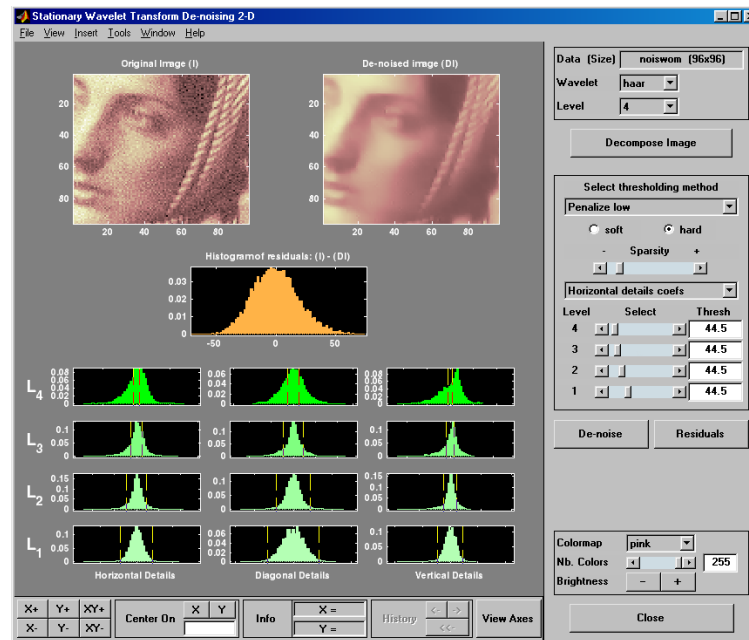
vertically through the histograms of the coefficients on the left of the window. Click the **De-noise** button.



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

### 5 Selecting a thresholding method.

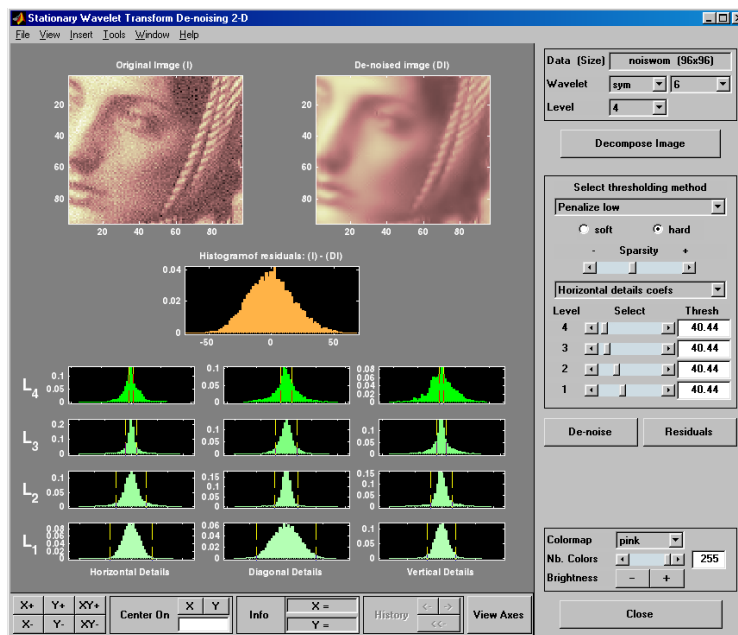
From the **Select thresholding method** menu, choose the **Penalize low** item. The associated default for the thresholding mode is automatically set to **hard**; accept it. Use the **Sparsity** slider to adjust the threshold value close to 44.5, and then click the **De-noise** button.



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

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





### Importing and Exporting Information from the Graphical Interface

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

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

```
load dnoiswom
whos
```

<b>Name</b>	<b>Size</b>	<b>Bytes</b>	<b>Class</b>
X	96x96	73728	double array
map	255x3	6120	double array
valTHR	3x4	96	double array
wname	1x4	8	char array

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

## One-Dimensional Wavelet Regression Estimation

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

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

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

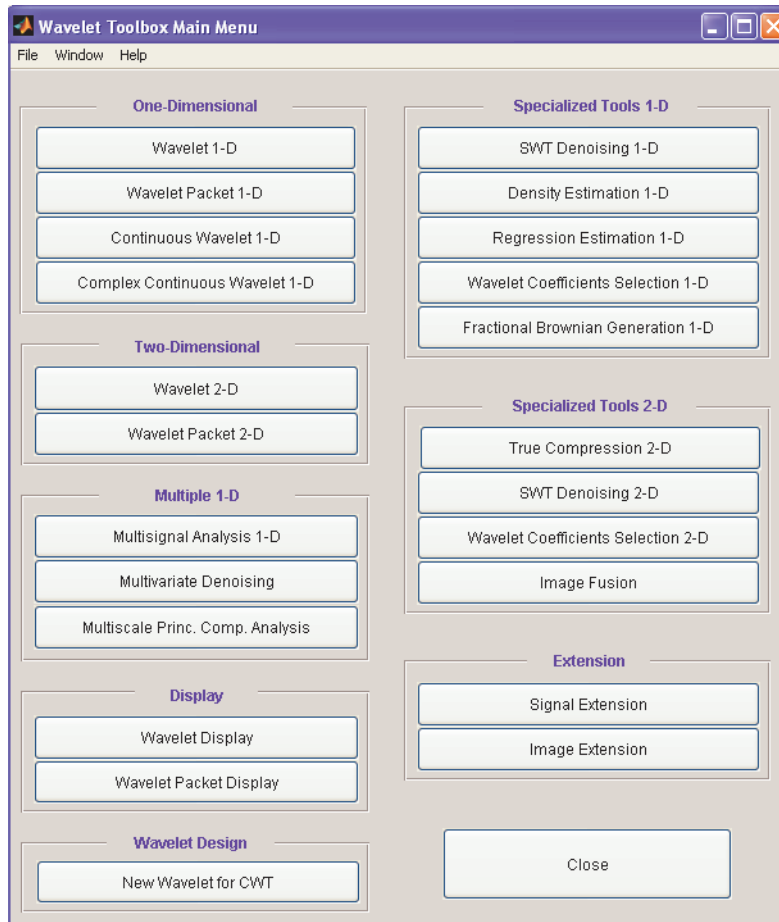
### One-Dimensional Estimation Using the GUI for Equally Spaced Observations (Fixed Design)

- 1 Start the Regression Estimation 1-D Tool.

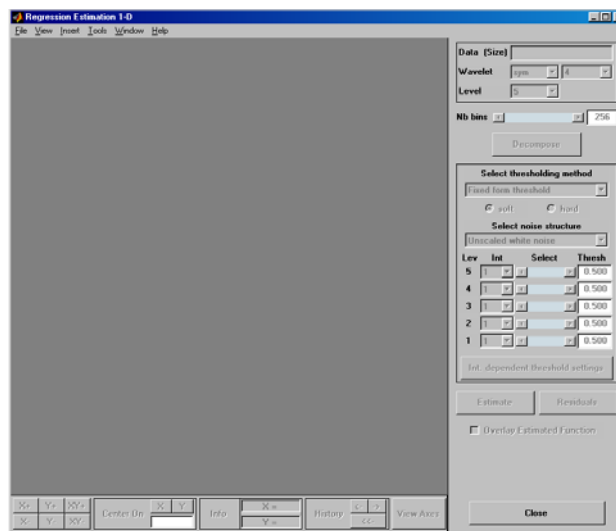
From the MATLAB® prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.

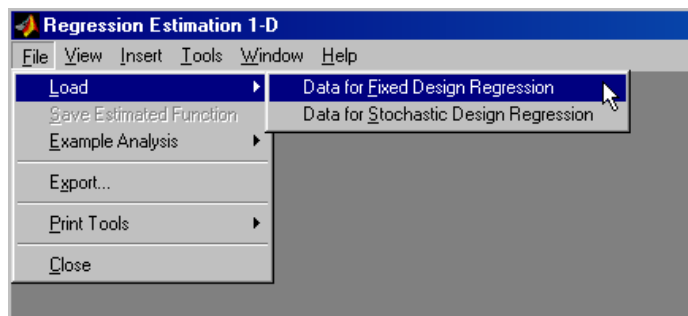


Click the **Regression Estimation 1-D** menu item. The discrete wavelet analysis tool for one-dimensional regression estimation appears.



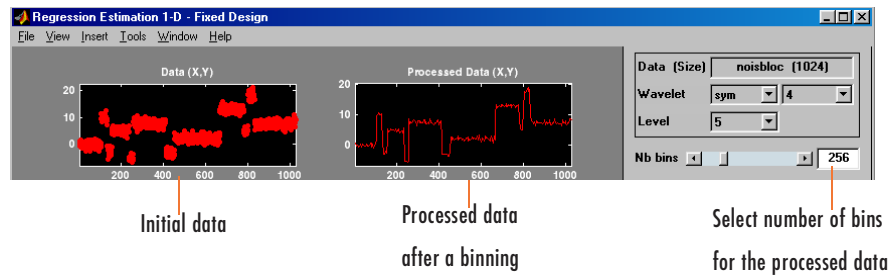
### 2 Load data.

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



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

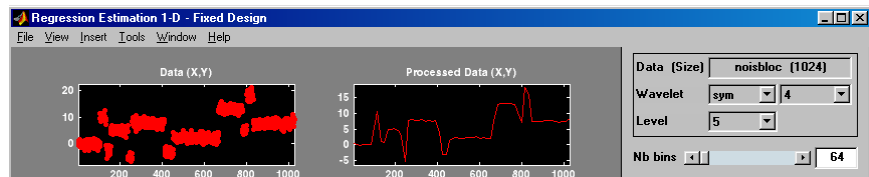
Click the **OK** button. The noisy blocks data is loaded into the **Regression Estimation 1-D - Fixed Design** tool.



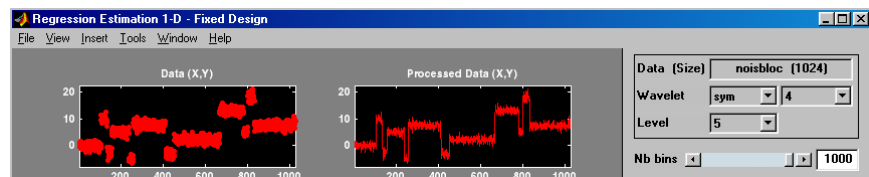
The loaded data denoted  $(X,Y)$  and the processed data obtained after a binning, are displayed.

### 3 Choose the processed data.

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



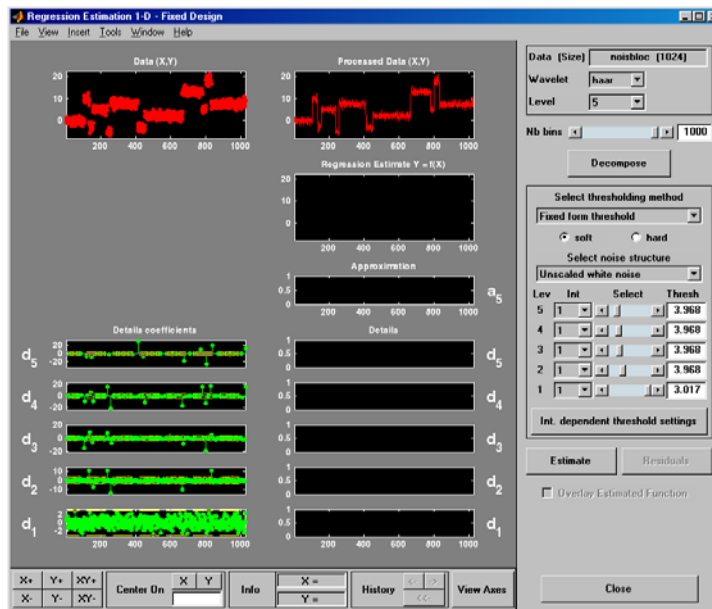
The binned data appears to be very smoothed. Select 1000 from the **Nb bins** edit and press **Enter** or use the slider. The new data to be processed appears.



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

4 Perform a Wavelet Decomposition of the processed data.

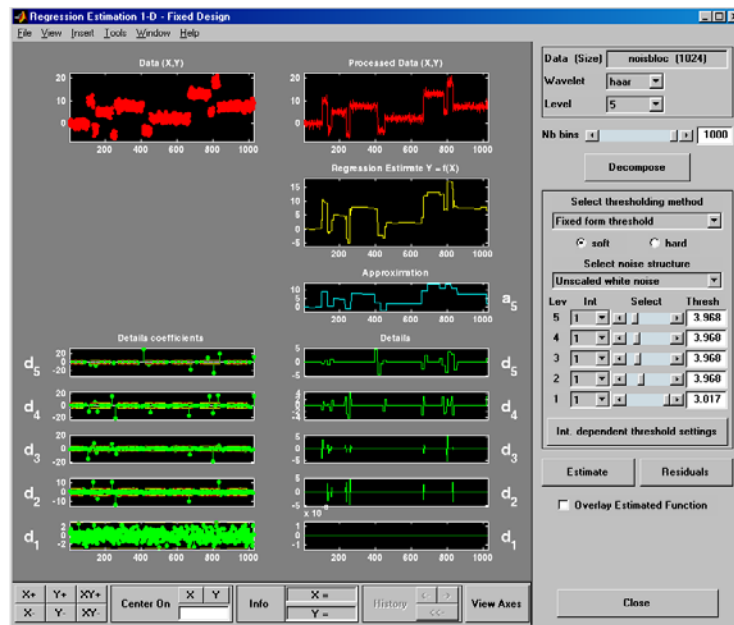
Select the haar wavelet from the **Wavelet** menu and select **5** from the **Level** menu, and then click the **Decompose** button. After a pause for computation, the tool displays the detail coefficients of the decomposition.



5 Perform a regression estimation.

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

Continue by clicking the **Estimate** button.



You can see that the process removed the noise and that the blocks are well reconstructed. The regression estimate (in yellow) is the sum of the signals located below it: the approximation  $a_5$  and the reconstructed details after coefficient thresholding.

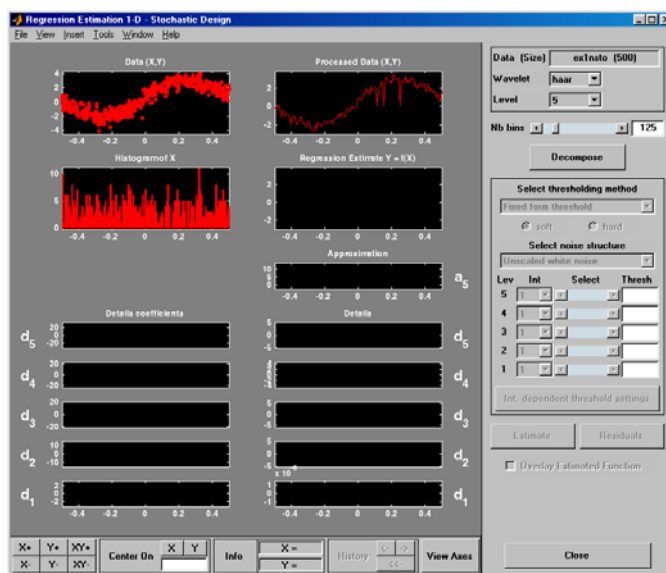
You can experiment with the various predefined thresholding strategies by selecting the appropriate options from the menu located on the right part of the window or directly by dragging the yellow horizontal lines with the left mouse button.

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



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

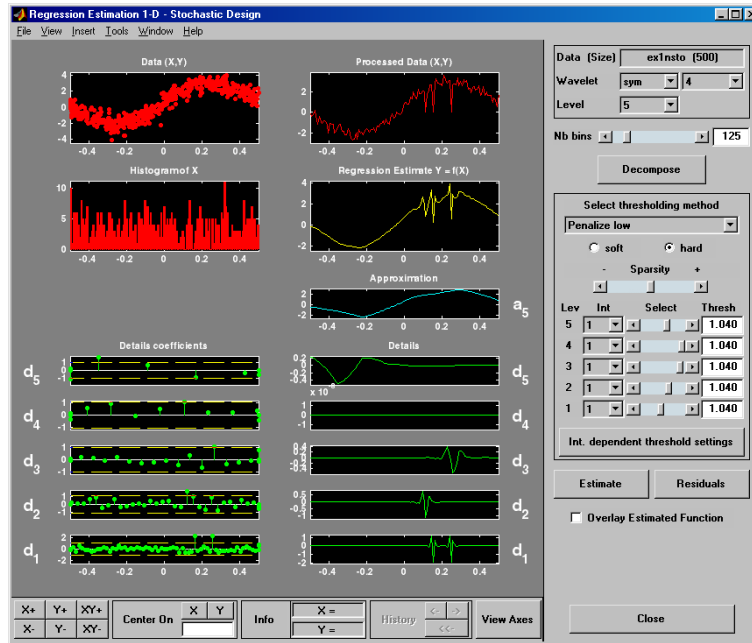
- 1 From the **File** menu, choose the **Load > Data for Stochastic Design Regression** option. When the **Load data for Stochastic Design Regression** dialog box appears, select the demo MAT-file `ex1nsto.mat`, which should reside in the MATLAB directory `toolbox/wavelet/wavedemo`. Click the **OK** button. This short set of data (of size 500) is loaded into the **Regression Estimation 1-D -- Stochastic Design** tool.



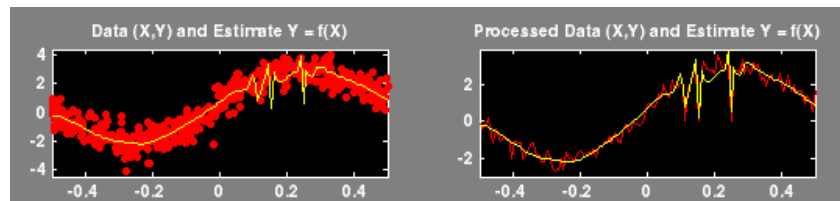
The loaded data denoted  $(X,Y)$ , the histogram of  $X$ , and the processed data obtained after a binning are displayed. The histogram is interesting, because the values of  $X$  are randomly distributed. The binning step is essential since it transforms a problem of regression estimation for irregularly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

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

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



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



## Importing and Exporting Information from the Graphical Interface

### Saving Function

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

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

```
load fex1nsto
whos
```

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

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

```
wname

wname =
    sym4
```

and the level dependent thresholds contained in `thrParams`, which is a cell array of length 5 (the level of the decomposition). For `i` from 1 to 5, `thrParams{i}` contains the lower and upper bounds of the interval of thresholding and the threshold value (since interval dependent thresholds are allowed). For more information, see “One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients” on page 2-158.

For example, for level 1,

```
thrParams{1}

ans =
    -0.4987  0.4997  1.0395
```

### Loading Data

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

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

These vectors must be the same length.

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

```
clear
load ex1nsto
whos
```

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

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

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

## One-Dimensional Wavelet Density Estimation

This section takes you through the features of one-dimensional wavelet density estimation using one of the Wavelet Toolbox™ specialized tools. For more information, see “Function Estimation: Density and Regression” on page 6-114.

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

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

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

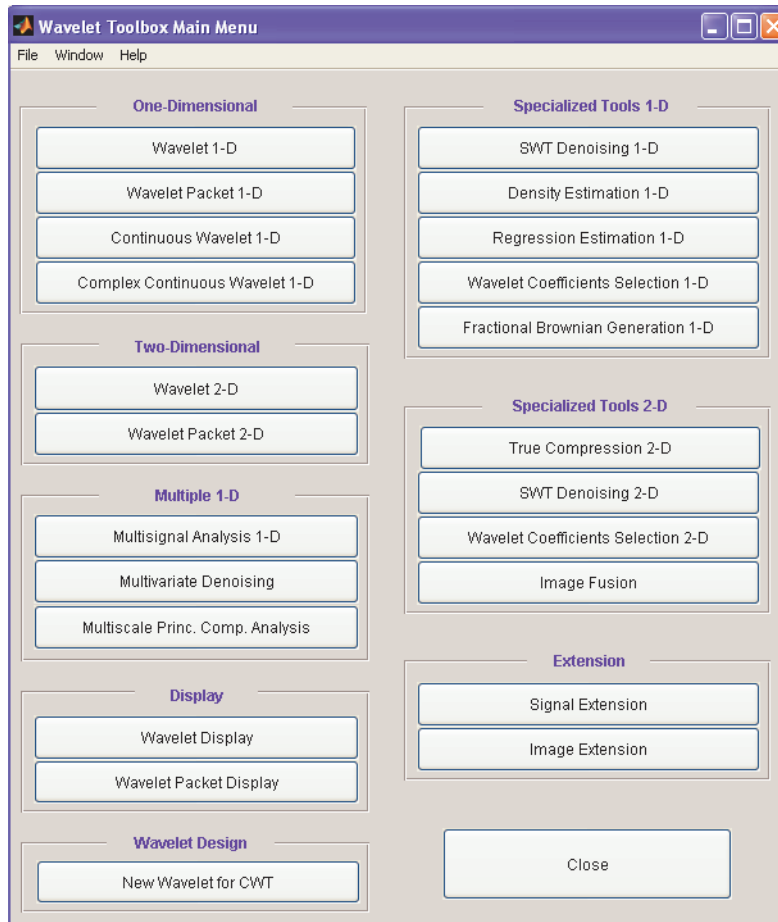
### One-Dimensional Estimation Using the Graphical Interface

- 1 Start the Density Estimation 1-D Tool.

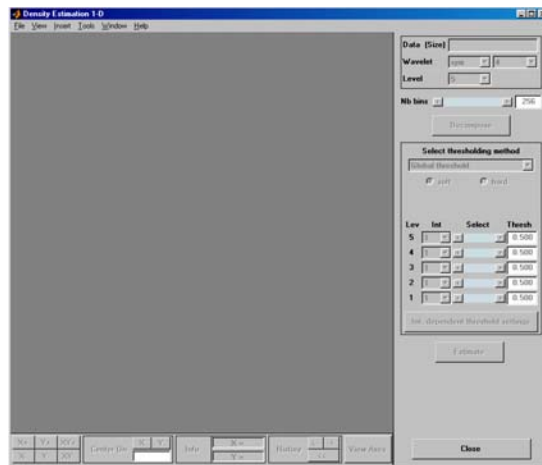
From the MATLAB prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.

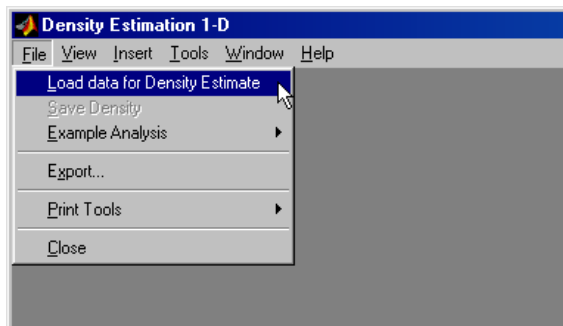


Click the **Density Estimation 1-D** menu item. The discrete wavelet analysis tool for one-dimensional density estimation appears.

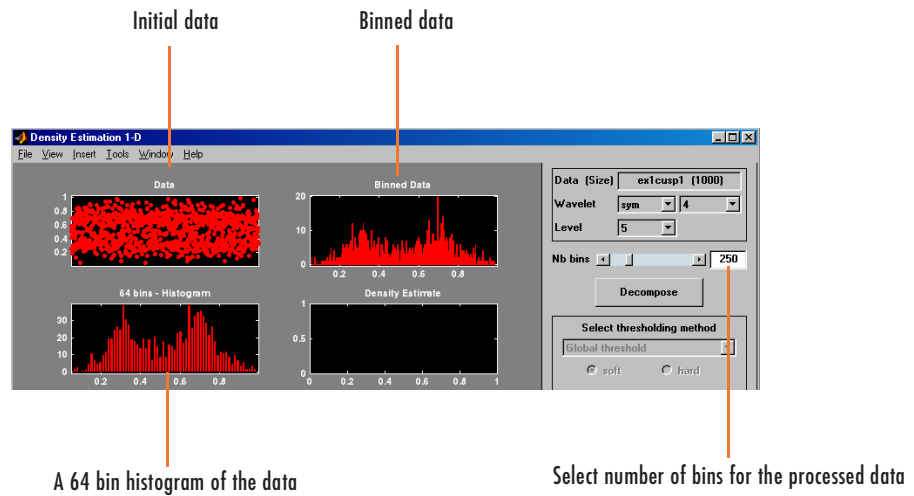


### 2 Load data.

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



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

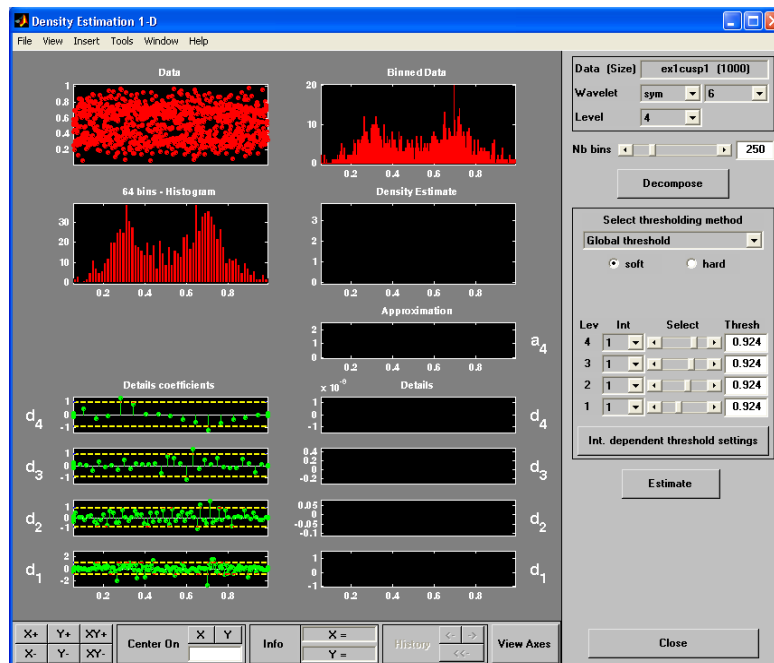


The sample, a 64-bin histogram, and the processed data obtained after a binning are displayed. In this example, we'll accept the default value for the number of bins (250). The binned data, suitably normalized, will be processed by wavelet decomposition.

### 3 Perform a Wavelet Decomposition of the binned data.

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

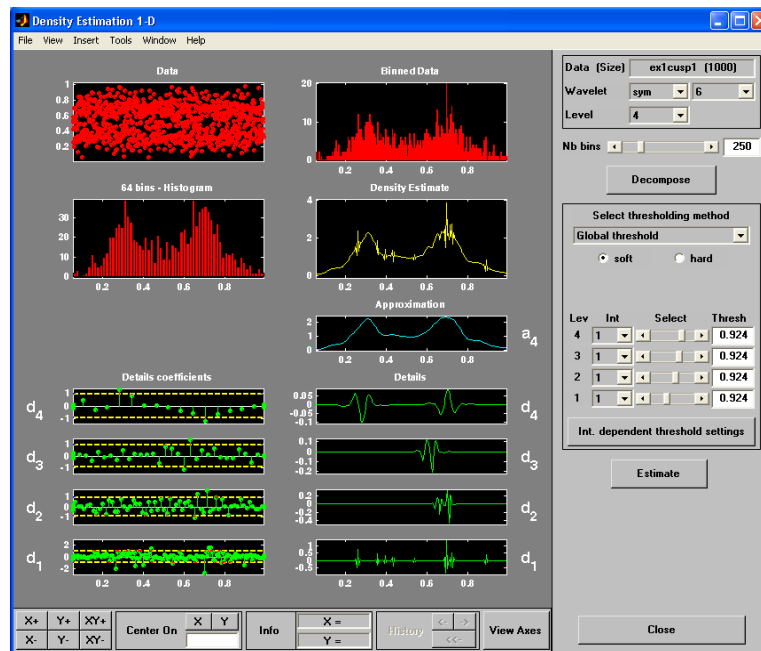




#### 4 Perform a density estimation.

We'll accept the defaults of global soft thresholding. The sliders located on the right of the window control the level dependent thresholds, indicated by yellow dotted lines running horizontally through the graphs on the left of the window.

Continue by clicking the **Estimate** button.

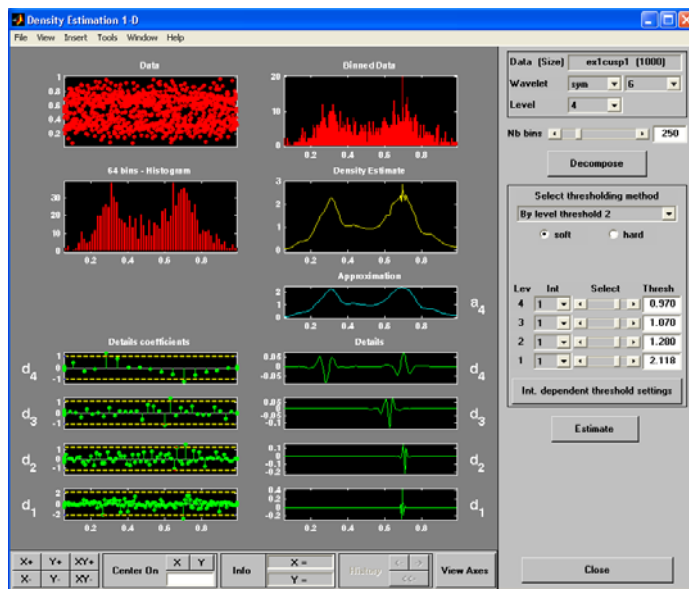


You can see that the estimation process delivers a very irregular resulting density. The density estimate (in yellow) is the normalized sum of the signals located below it: the approximation  $a_4$  and the reconstructed details after coefficient thresholding.

## 5 Perform thresholding.

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

From the menu **Select thresholding method**, select the item **By level threshold 2**. For more information about these methods, see “Function Estimation: Density and Regression” on page 6-114. Next, click the **Estimate** button.



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

## Importing and Exporting Information from the Graphical Interface

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

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

```
load dex1cusp
whos
```

<b>Name</b>	<b>Size</b>	<b>Bytes</b>	<b>Class</b>
thrParams	1x4	464	cell array
wname	1x4	8	char array
xdata	1x250	2000	double array
ydata	1x250	2000	double array

The estimated density is given by `xdata` and `ydata`. The length of these vectors is of the same as the number of bins you choose in step 4. In addition, the parameters of the estimation process are given by the wavelet name in `wname`.

```
wname
```

```
wname =  
    sym6
```

and the level dependent thresholds contained in `thrParams`, which is a cell array of length 4 (the level of the decomposition). For `i` from 1 to 4, `thrParams{i}` contains the lower and upper bounds of the interval of thresholding and the threshold value (since interval dependent thresholds are allowed). For more information, see “One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients” on page 2-158. For example, for level 1,

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

---

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

---

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

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

# One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients

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

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

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

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

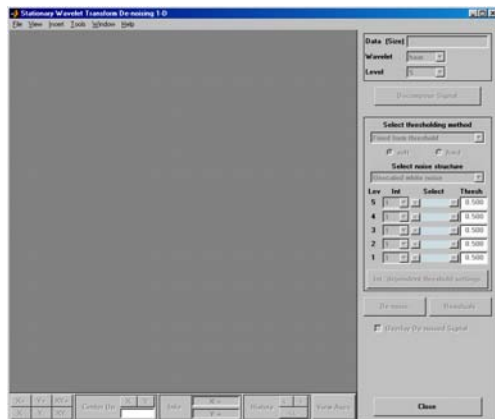
- 1 From the MATLAB<sup>®</sup> prompt, type  
wavemenu

The **Wavelet Toolbox Main Menu** appears.



Click the **SWT De-noising 1-D** menu item.

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



### 2 Load data.

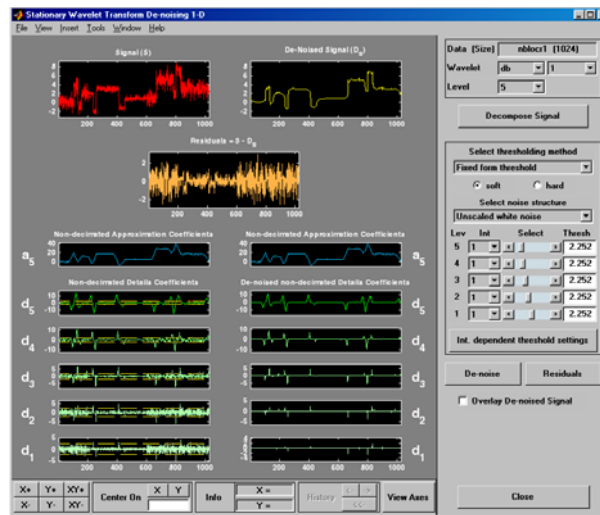
From the **File** menu, choose the **Load Signal** option.

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

### 3 Perform signal decomposition.

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

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



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

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

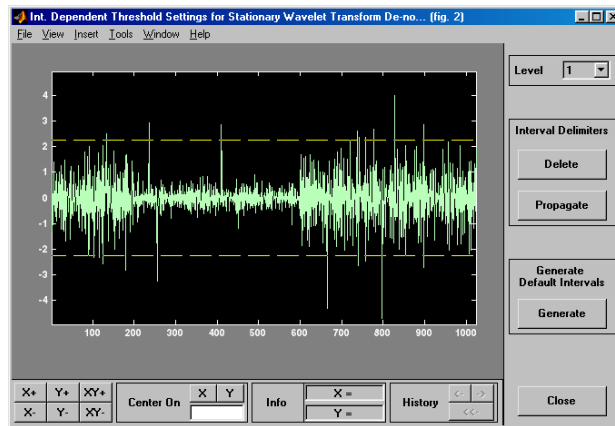




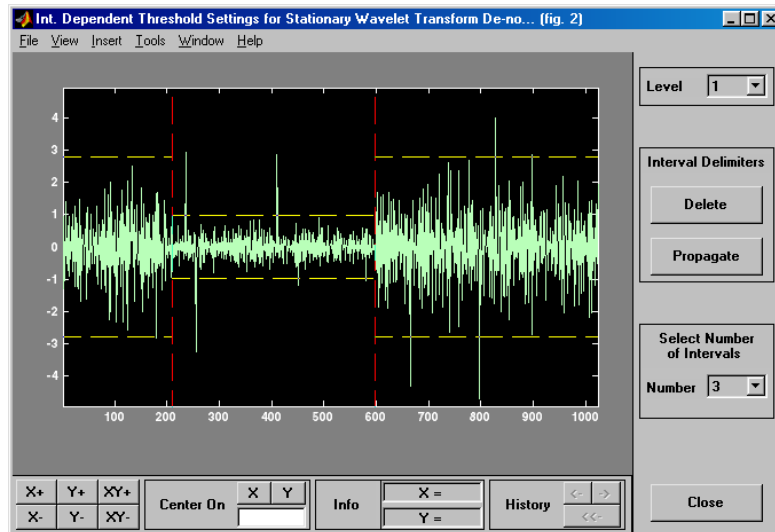
The result is not satisfactory. The de-noised signal remains noisy before position 200 and after position 700. This illustrates the limits of the classical de-noising strategies. In addition, the residuals obtained during the last trials clearly suggest to try a local thresholding strategy.

#### 4 Generate interval-dependent thresholds.

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

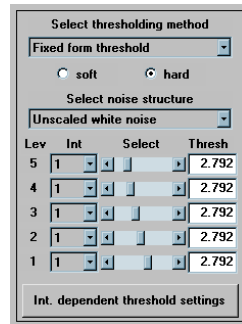


Click the **Generate** button. After a pause for computation, the tool displays the default intervals associated with adapted thresholds.



Three intervals are proposed. Since the variances for the three intervals are very different, the optimization program easily detects the correct structure. Nevertheless, you can visualize the intervals proposed for a number of intervals from 1 to 6 using the **Select Number of Intervals** menu (which





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

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

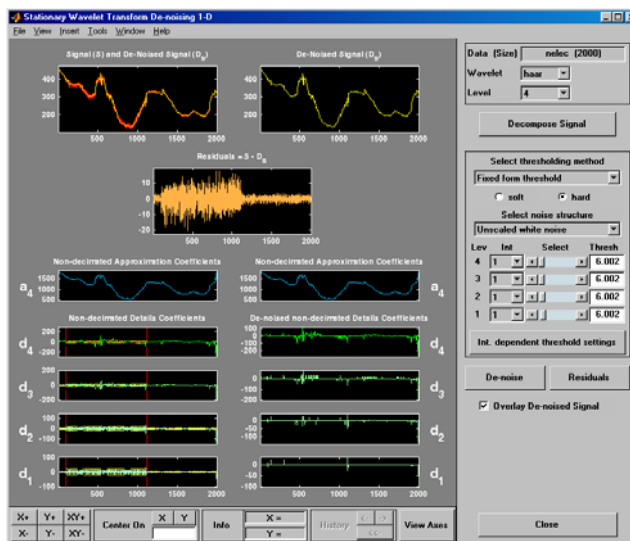
Note that

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

### Some Examples of De-noising with Interval Dependent Thresholds.

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

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



### Importing and Exporting Information from the Graphical Interface

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

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

```
load dnelec
```

```
whos
```

<b>Name</b>	<b>Size</b>	<b>Bytes</b>	<b>Class</b>
dnelec	1x2000	16000	double array
thrParams	1x4	656	cell array
wname	1x4	8	char array

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

```
wname
```

```
wname =
    haar
```

and the level dependent thresholds contained in thrParams, which is a cell array of length 4 (the level of the decomposition). For  $i$  from 1 to 4, thrParams{ $i$ } is an array nbintx3 (where nbint is the number of intervals, here 3), and each row contains the lower and upper bounds of the interval of thresholding and the threshold value. For example, for level 1,

```
thrParams{1}
ans =
    1.0e+03 *
    0.0010 0.0980 0.0060
    0.0980 1.1240 0.0204
    1.1240 2.0000 0.0049
```

## One-Dimensional Selection of Wavelet Coefficients Using the Graphical Interface

This section takes you through the features of one-dimensional selection of wavelet coefficients using one of the Wavelet Toolbox™ specialized tools. The toolbox provides a graphical interface tool to explore some reconstruction schemes based on various wavelet coefficients selection strategies:

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

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

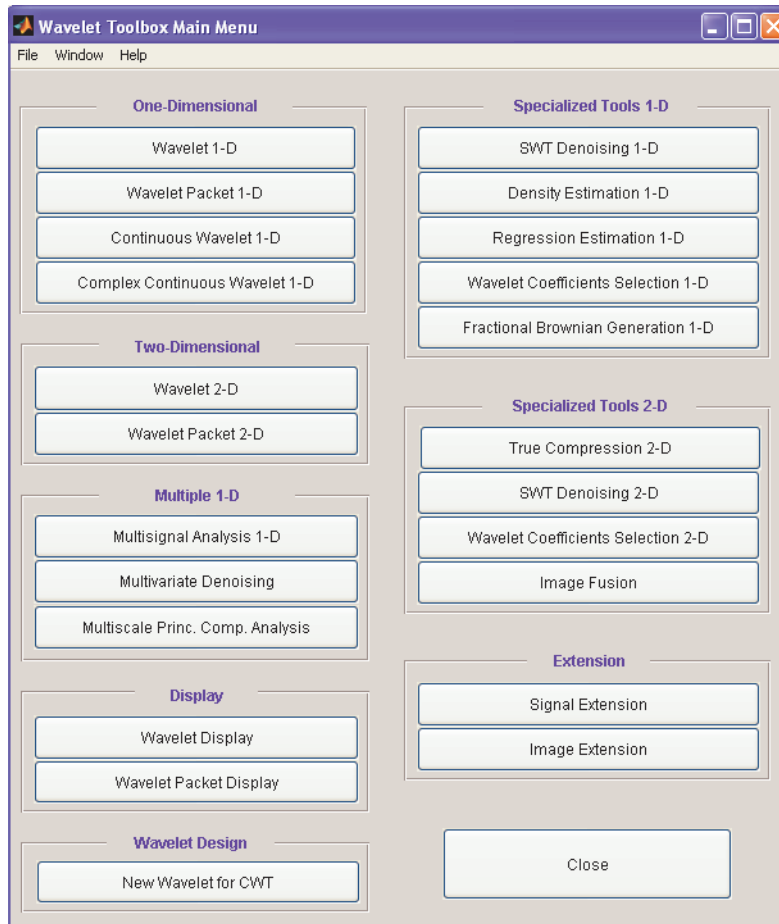
```
dwtmode('sym')
```

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

From the MATLAB® prompt, type

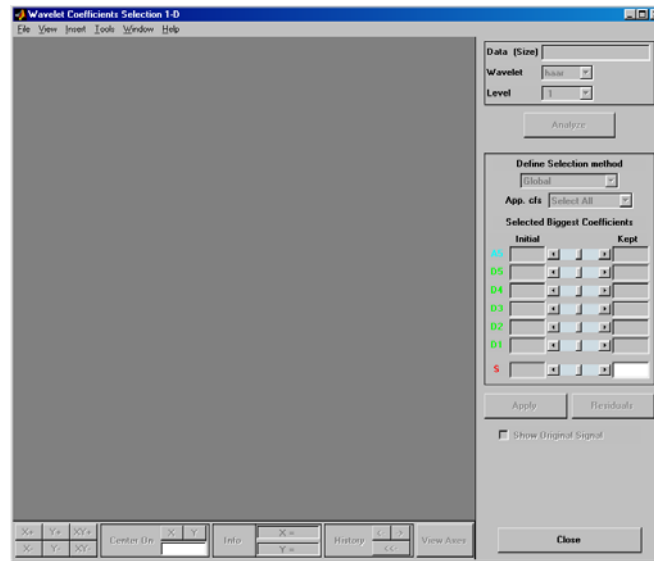
```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.



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





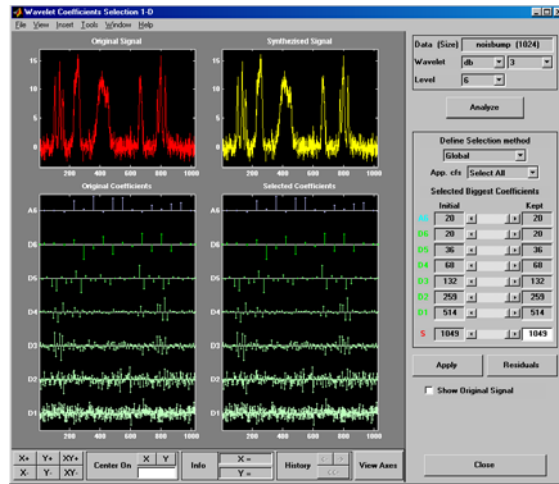
### 2 Load data.

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

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

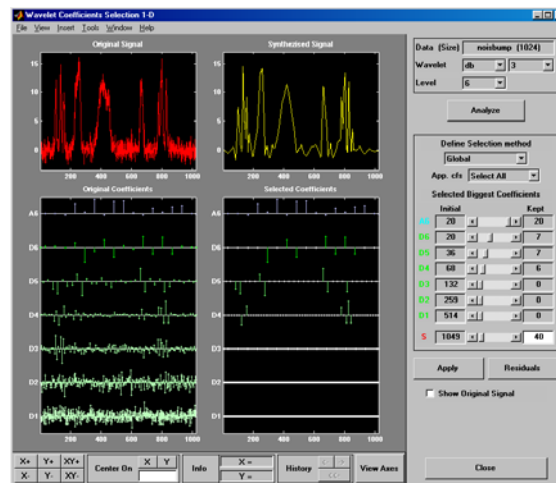
### 3 Perform a Wavelet Decomposition.

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

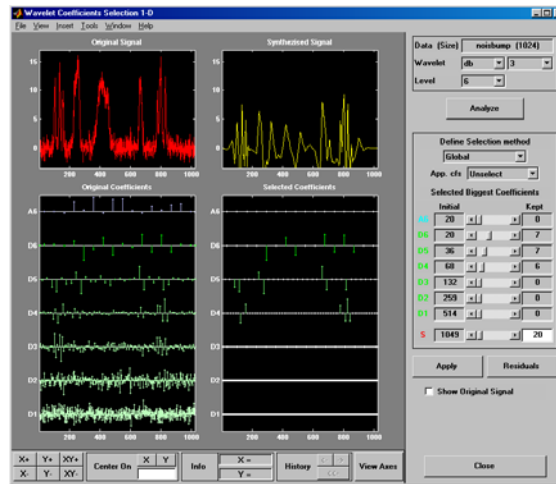


The tool displays below the original signal (on the left) its wavelet decomposition: the approximation coefficients A6 and detail coefficients from D6 at the top to D1 at the bottom. In the middle of the window, below the synthesized signal (which at this step is the same, since all the wavelet coefficients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

**Selecting Biggest Coefficients Globally.** On the right of the window, find a column labeled **Kept**. The last line shows the total number of coefficients: 1049. This is a little bit more than the number of observations, which is 1024. You can choose the number of selected biggest coefficients by typing a number instead of 1049 or by using the slider. Type 40 and press **Enter**. The numbers of selected biggest coefficients level by level are updated (but cannot be modified since **Global** is the current selection method). Then click the **Apply** button. The resulting coefficients are now displayed.



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



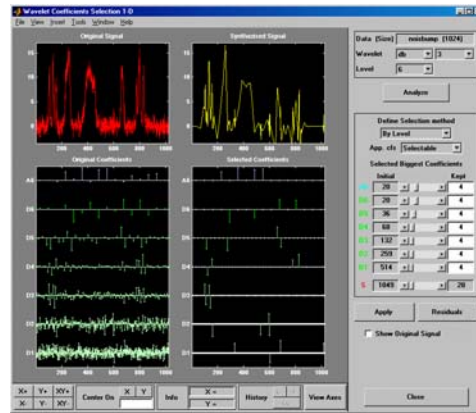
None of the approximation coefficients are kept.

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

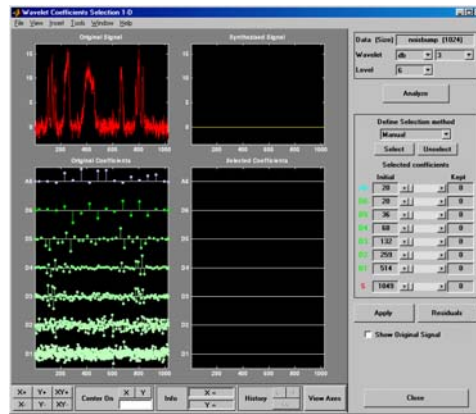


Some of the approximation coefficients (15) have been kept.

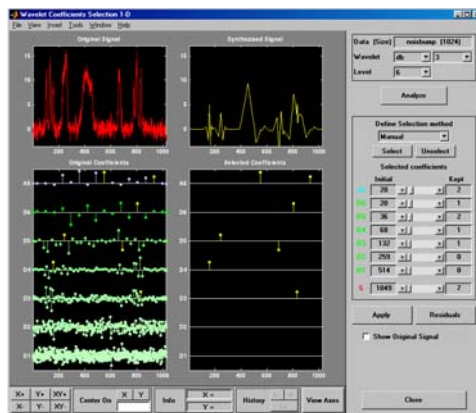
**Selecting Biggest Coefficients by Level.** From the **Define Selection method** menu, select the **By Level** option. You can choose the number of selected biggest coefficients by level or select it using the sliders. Type 4 for the approximation and each detail, and then click the **Apply** button.



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



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

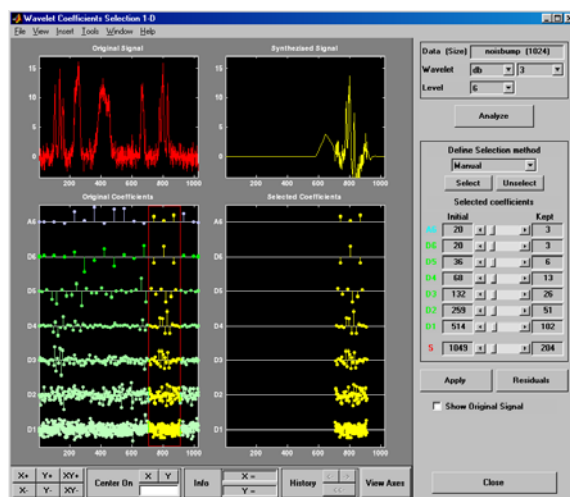


You can deselect the currently selected coefficients by double clicking each of them. Another way to select or deselect a set of coefficients is to use the selection box. Drag a rubber band box (hold down the left mouse button) over a portion of the coefficient axes (original or selected) containing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

right of the window. Click the **Apply** button. The tool displays the null signal again.

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

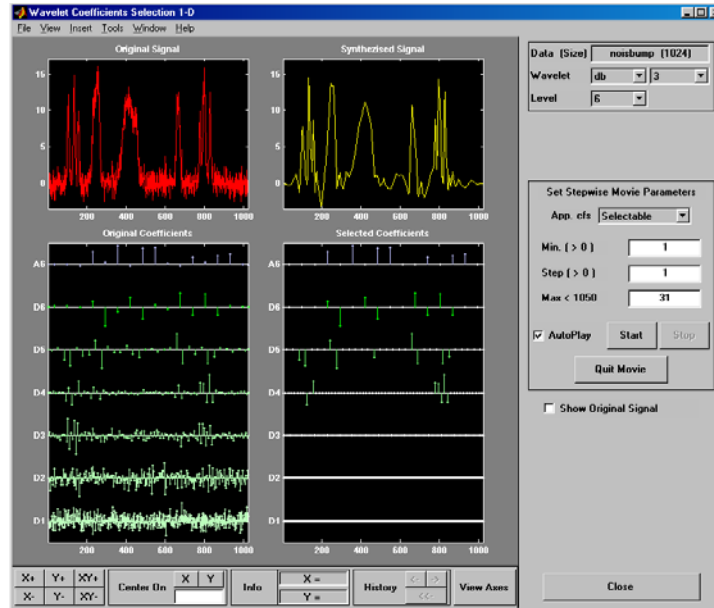
Drag a rubber band box over the portion of the coefficient axes around the position 800 and containing all scales and click the **Select** button. Click the **Apply** button.



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

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

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



#### 4 Save the synthesized signal.

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

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

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

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



## Two-Dimensional Selection of Wavelet Coefficients Using the Graphical Interface

This section takes you through the features of two-dimensional selection of wavelet coefficients using one of the Wavelet Toolbox™ specialized tools. The toolbox provides a graphical interface tool to explore some reconstruction schemes based on various wavelet coefficient selection strategies:

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

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

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

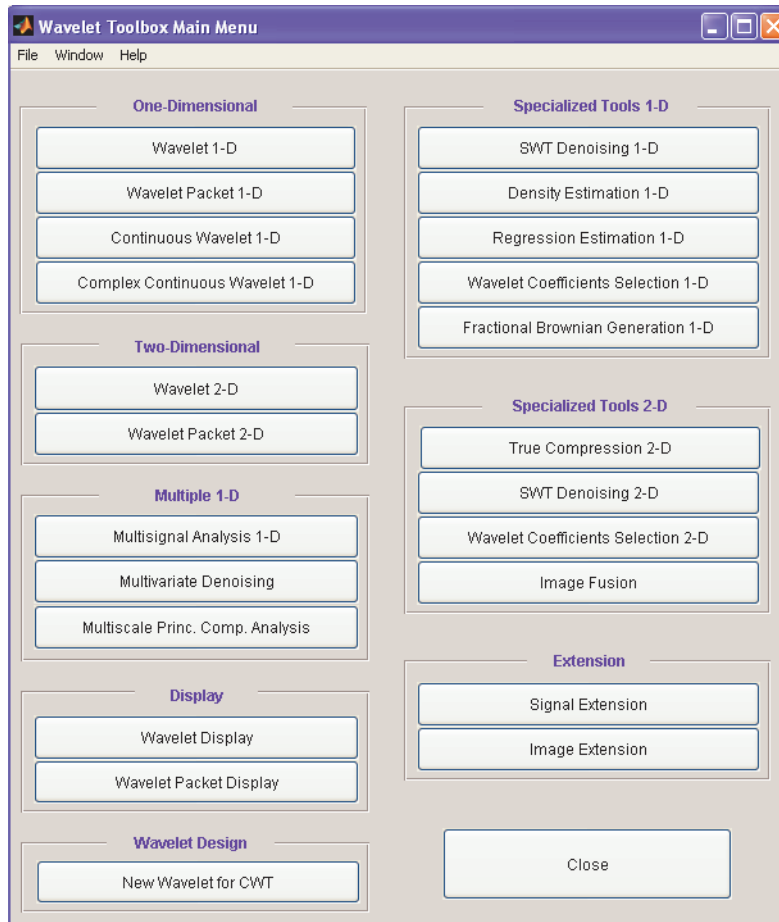
```
dwtmode('sym')
```

- 1 Start the Wavelet Coefficients Selection 2-D Tool.

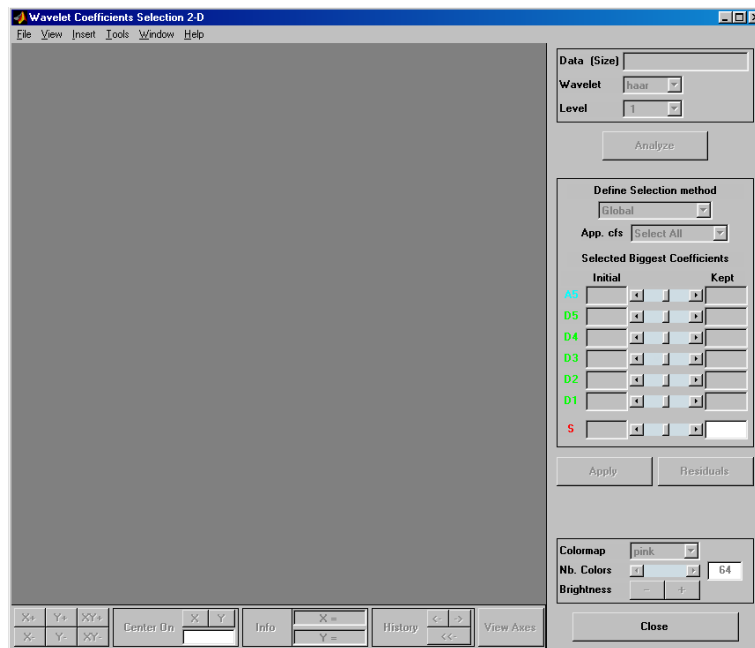
From the MATLAB® prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.



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



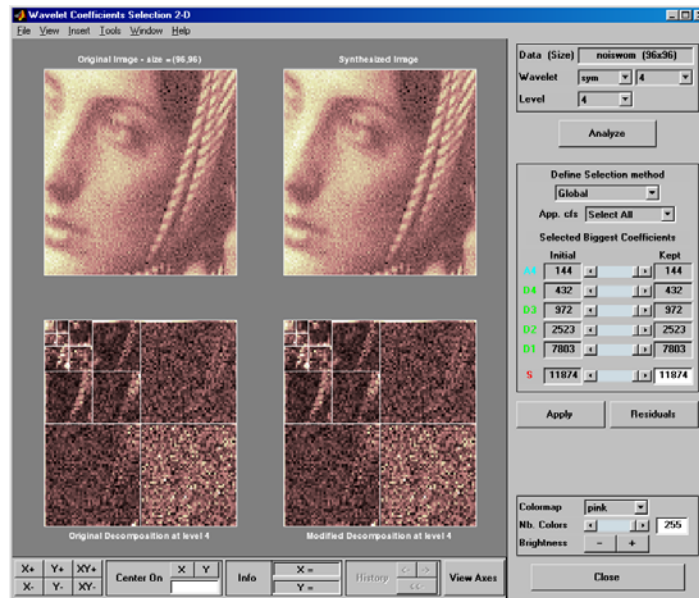
### 2 Load data.

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

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

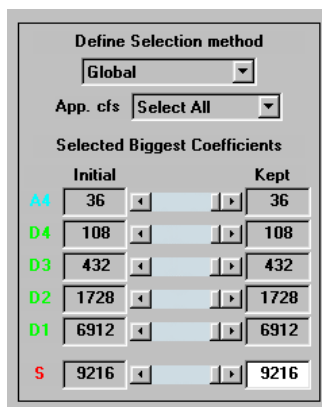
### 3 Perform a Wavelet Decomposition.

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



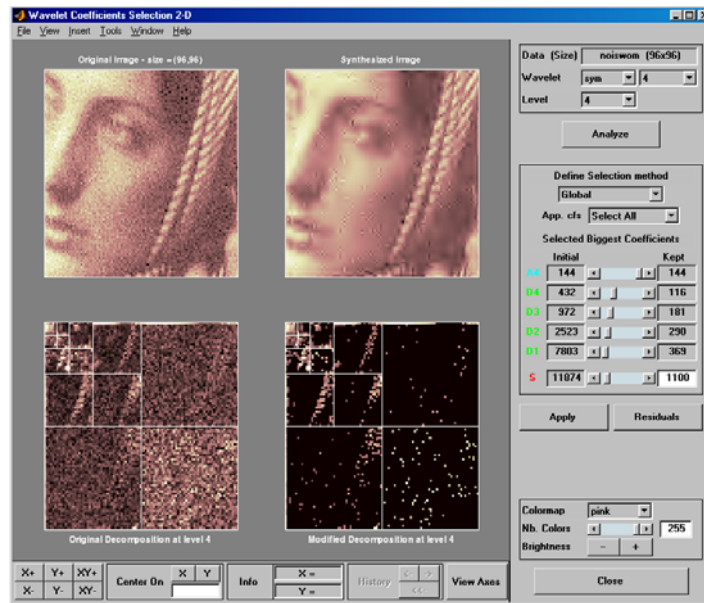
The tool displays its wavelet decomposition below the original image (on the left). The selected coefficients are displayed in the middle of the window, below the synthesized image (which, at this step, is the same since all the wavelet coefficients are kept). There are 11874 coefficients, a little bit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

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



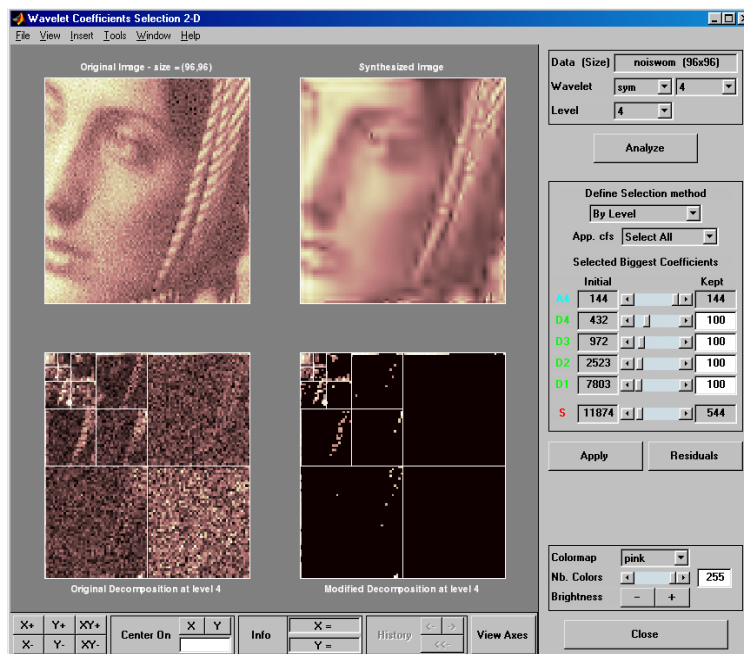
**Selecting Biggest Coefficients Globally.** On the right of the window, find a column labeled **Kept**. The last line shows the total number of coefficients: 11874. This is a little bit more than the original image number of pixels. You can choose the number of selected biggest coefficients by typing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

Then click the **Apply** button.

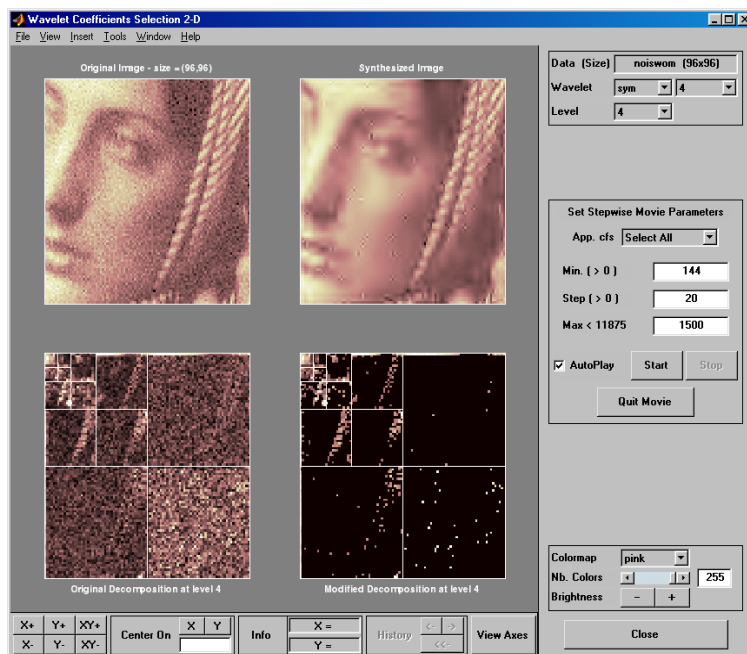


In the previous trial, the approximation coefficients were all kept. It is possible to relax this constraint by selecting another option from the **App. cfs** menu (see “One-Dimensional Selection of Wavelet Coefficients Using the Graphical Interface” on page 2-168).

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



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



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

#### 4 Save the synthesized image.

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

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

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

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



# One-Dimensional Extension

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

## One-Dimensional Extension Using the Command Line

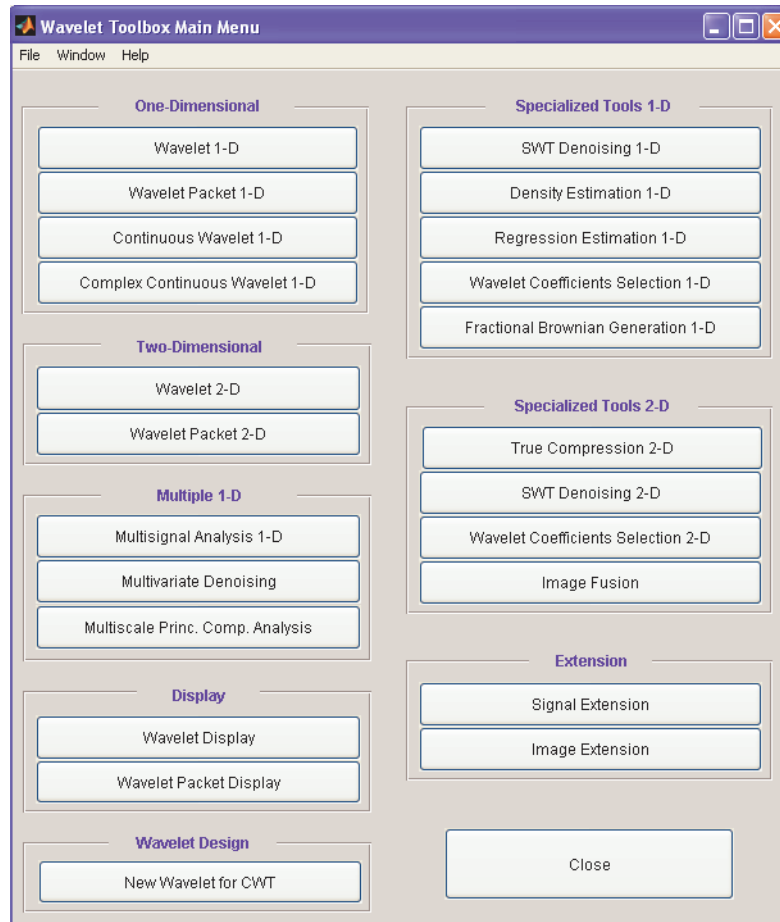
The function `wextend` performs signal extension. For more information, see its reference page.

## One-Dimensional Extension Using the Graphical Interface

- 1 Start the Signal Extension Tool.

From the MATLAB® prompt, type  
`wavemenu`

The **Wavelet Toolbox Main Menu** appears.



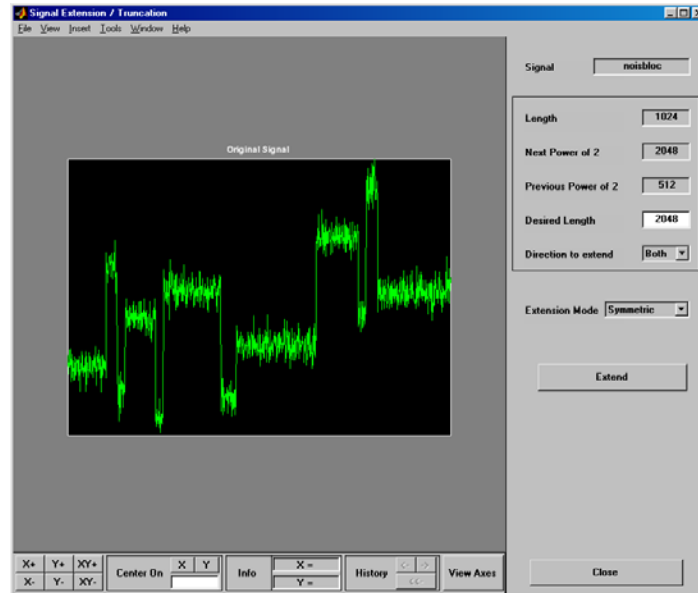
Click the **Signal Extension** menu item.

## 2 Load data.

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

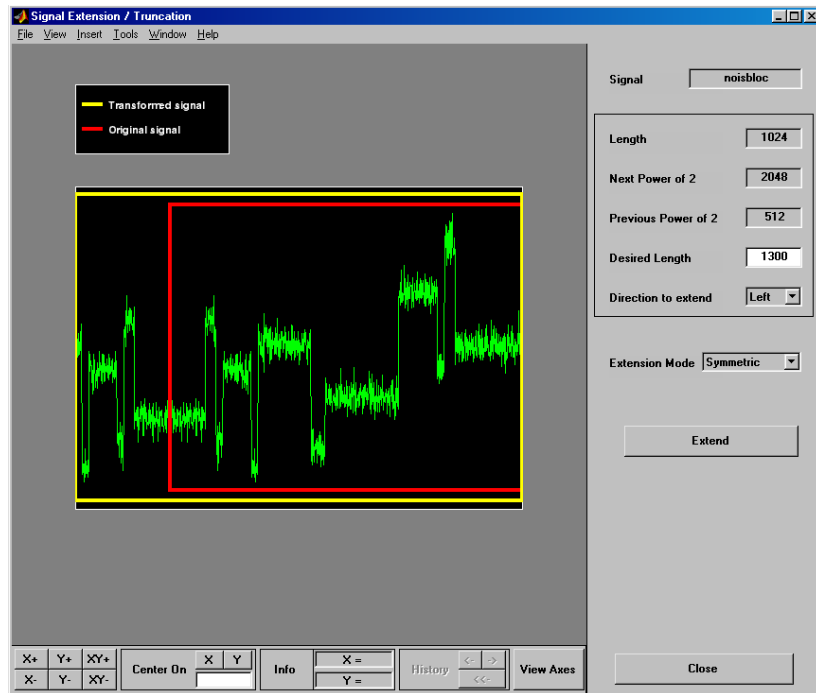
When the **Load Signal** dialog box appears, select the demo MAT-file `noisbloc.mat`, which should reside in the MATLAB directory

toolbox/wavelet/wavedemo. Click the **OK** button. The noisy blocks data is loaded into the **Signal Extension** tool.



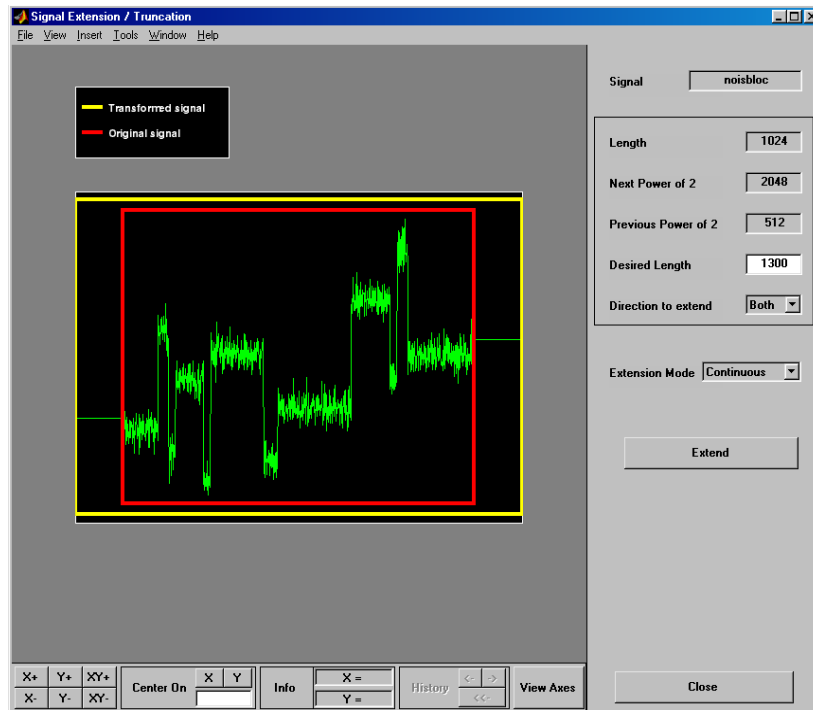
### 3 Extend the signal.

Enter 1300 in the **Desired Length** box of the extended signal, and select the **Left** option from the **Direction to extend** menu. Then accept the default **Symmetric** for the **Extension mode**, and click the **Extend** button.



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

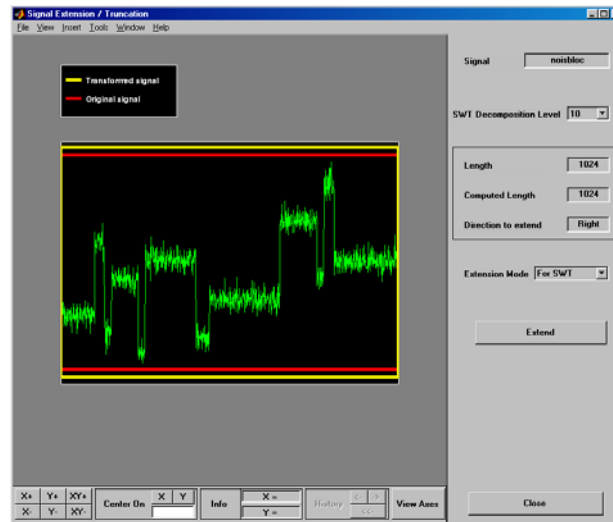
Select the **Both** option from the **Direction to extend** menu and select the **Continuous** option from the **Extension mode** menu. Click the **Extend** button.



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

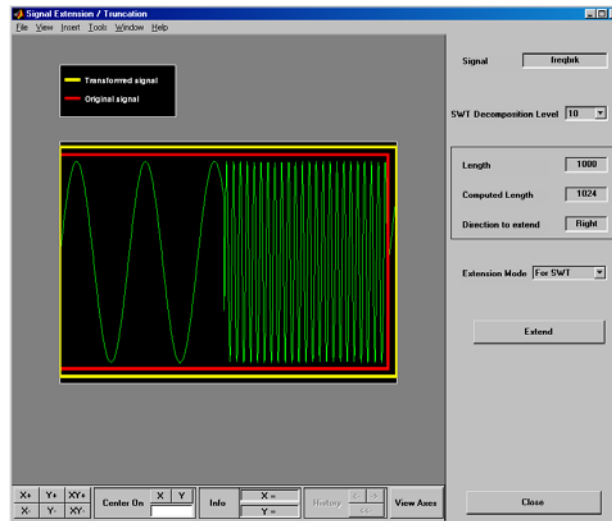
**Extending Signal for SWT.** Since the decomposition at level  $k$  of a signal using SWT requires that  $2^k$  divides evenly into the length of the signal, the tool provides a special option dedicated to this kind of extension.

Select the **For SWT** option from the **Extension mode** menu. Click the **Extend** button.



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

From the **File** menu, choose the **Example Extension** option and select the last item of the list.



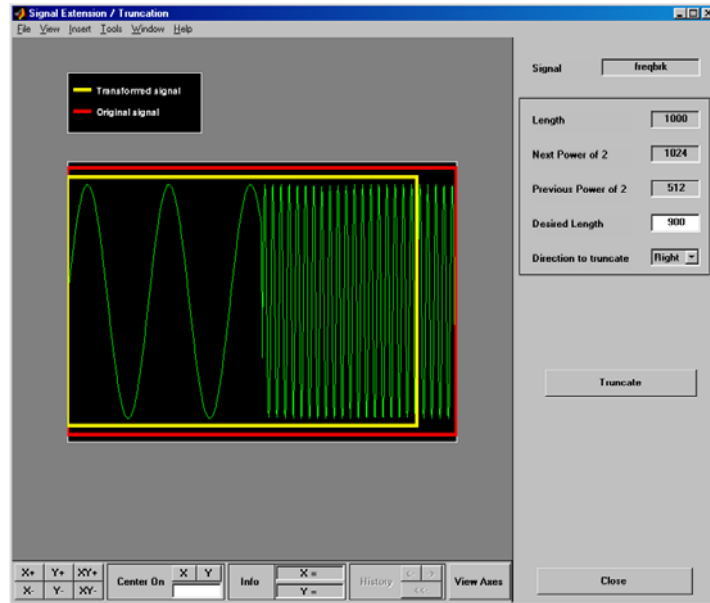
Since the signal is of length 1000 and the decomposition level needed for SWT is 10, the tool performs a minimal right periodic extension. The extended signal is of length 1024.

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

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

**Truncating Signal.** The same tool allows you to truncate a signal.

Since truncation is not allowed for the special mode **For SWT**, select the **Periodic** option from the **Extension mode** menu. Type 900 for the desired length and press **Enter**. Click the **Truncate** button.



The tool displays the original signal delimited by a red box and the truncated signal delimited by a yellow box. The signal has been truncated by deleting 100 values on the right side.



### Importing and Exporting Information from the Graphical Interface

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

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

```
load tfrqbrk
whos
```

Name	Size	Bytes	Class
tfrqbrk	1x900	7200	double array

## Two-Dimensional Extension

This section takes you through the features of two-dimensional extension or truncation using one of the Wavelet Toolbox™ utilities. This section is short since it is very similar to “One-Dimensional Extension” on page 2-186.

### Two-Dimensional Extension Using the Command Line

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

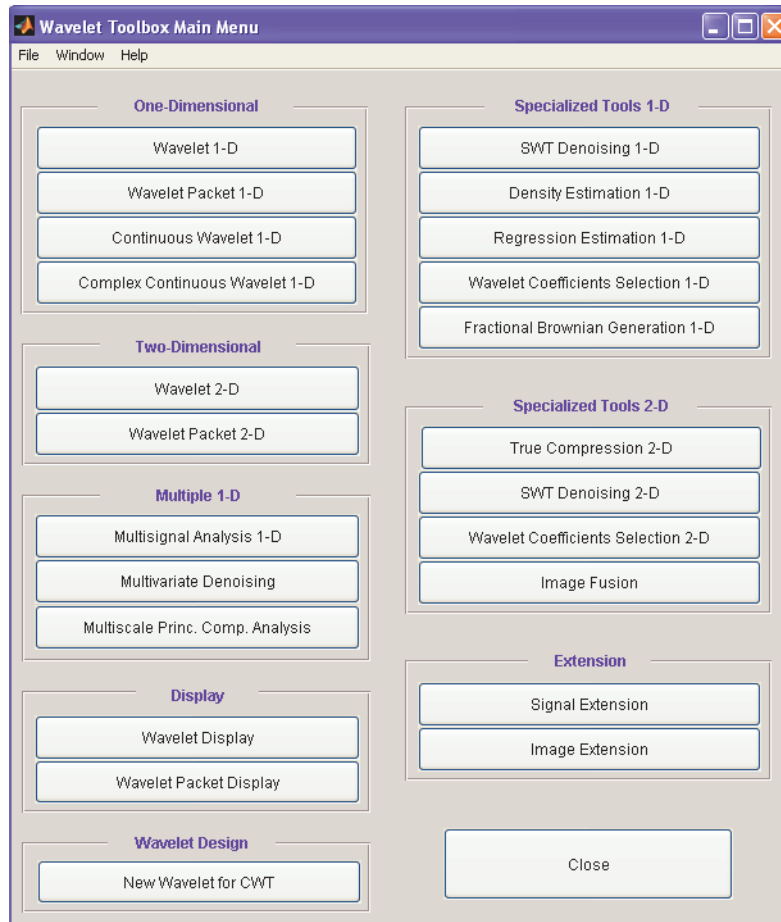
### Two-Dimensional Extension Using the Graphical Interface

- 1 Start the Image Extension Tool.

From the MATLAB® prompt, type

```
wavemenu
```

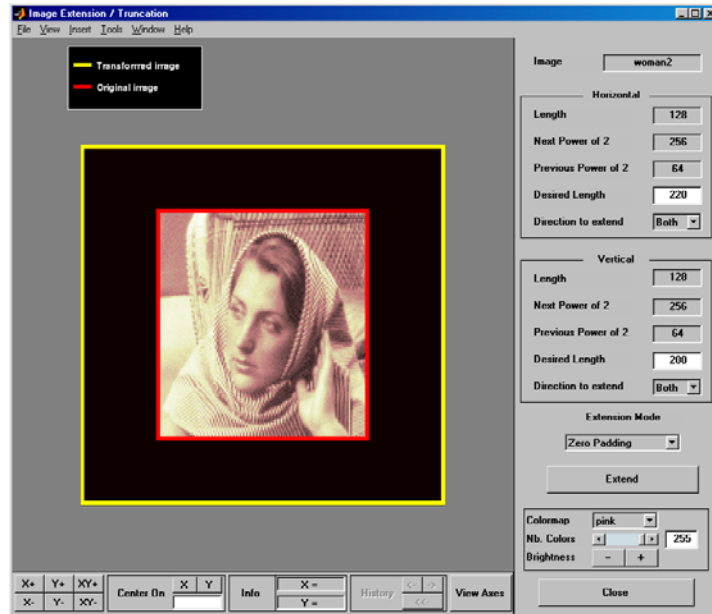
The **Wavelet Toolbox Main Menu** appears.



Click the **Image Extension** menu item.

## 2 Extend (or truncate) the image.

From the **File** menu, choose the **Example Extension** option and select the first item of the list.



The tool displays the original image delimited by a red box and the transformed image delimited by a yellow box. The image has been extended by zero padding. The right part of the window allows you to control the parameters of the extension/truncation process for the vertical and horizontal directions, respectively. The possibilities are similar to the one-dimensional ones described in “One-Dimensional Extension” on page 2-186.

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

## Importing and Exporting Information from the Graphical Interface

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

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

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

```
load woman2
whos
```

<b>Name</b>	<b>Size</b>	<b>Bytes</b>	<b>Class</b>
woman2	200x220	352000	double array
map	253x3	6120	double array

The transformed image is stored together with its colormap.

## Image Fusion

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

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

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

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

In this section, you'll learn how to

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

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

The principle of image fusion using wavelets is to merge the wavelet decompositions of the two original images using fusion methods applied to approximations coefficients and details coefficients (see [MisMOP03] and [Zee98] in “References” on page 6-155).

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

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

### Image Fusion Using the Command Line

#### Example 1: Fusion of Two Different Images

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

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

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

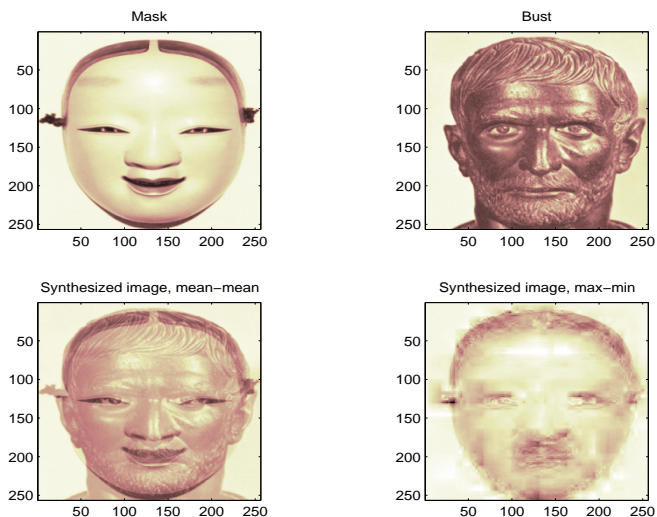
```
XFUSmean = wfusing(X1,X2,'db2',5,'mean','mean');
```

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

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

- 3** Plot original and synthesized images.

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



## Example 2: Restoration by Fusion from Fuzzy Images

- 1 Load two fuzzy versions of an original image.

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

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

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

- 3 Plot original and synthesized images.

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



```
subplot(223), image(XFUS), axis square,  
title('Synthesized image')
```



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

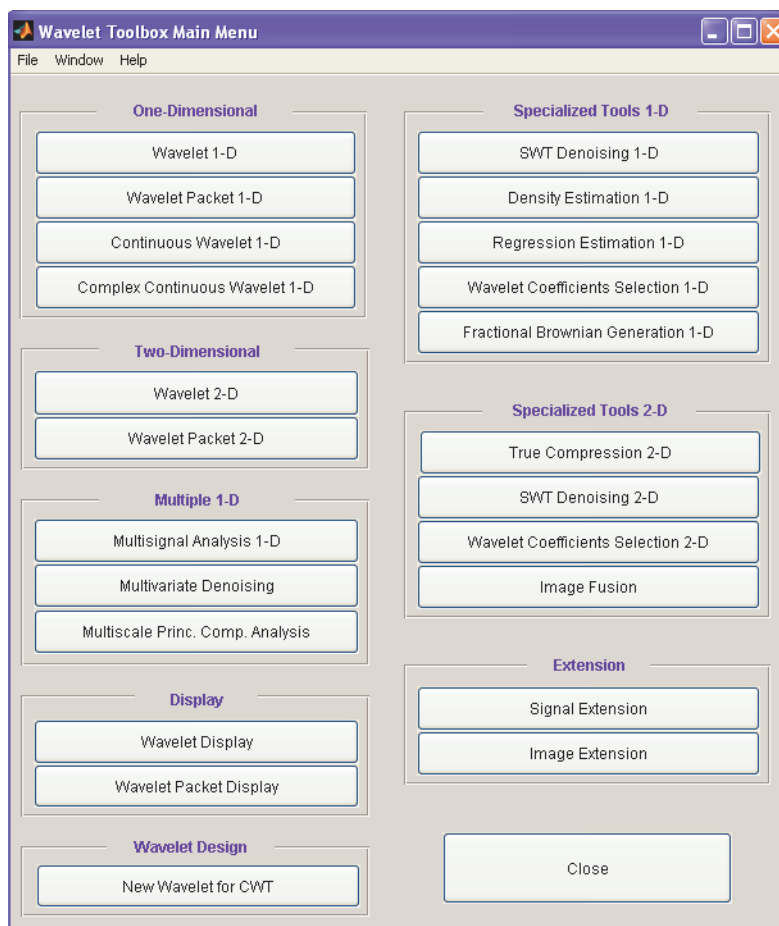
## Image Fusion Using the Graphical Interface

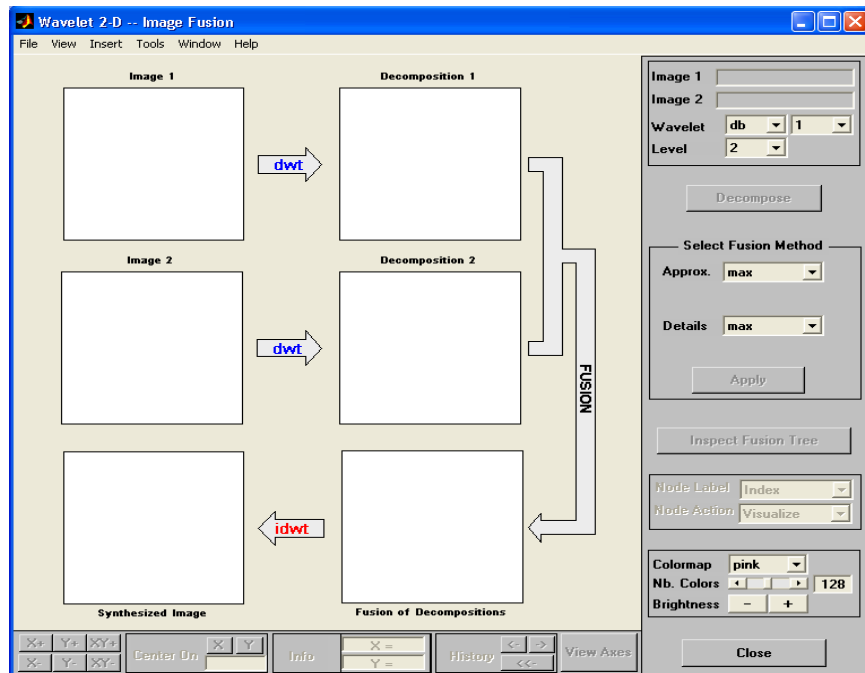
- 1 Start the Image Fusion Tool.

From the MATLAB<sup>®</sup> prompt, type

```
wavemenu
```

to display the **Wavelet Toolbox Main Menu** and then click the **Image Fusion** menu item to display the Image Fusion Tool.





### 2 Load original images.

From the **File** menu, choose the **Load Image 1** option.

When the **Load Image 1** dialog box appears, select the demo MAT-file mask.mat, which should reside in the MATLAB directory toolbox/wavelet/wavedemo. Click the **OK** button.

Perform the same sequence choosing the **Load Image 2** option and selecting the demo MAT-file bust.mat.

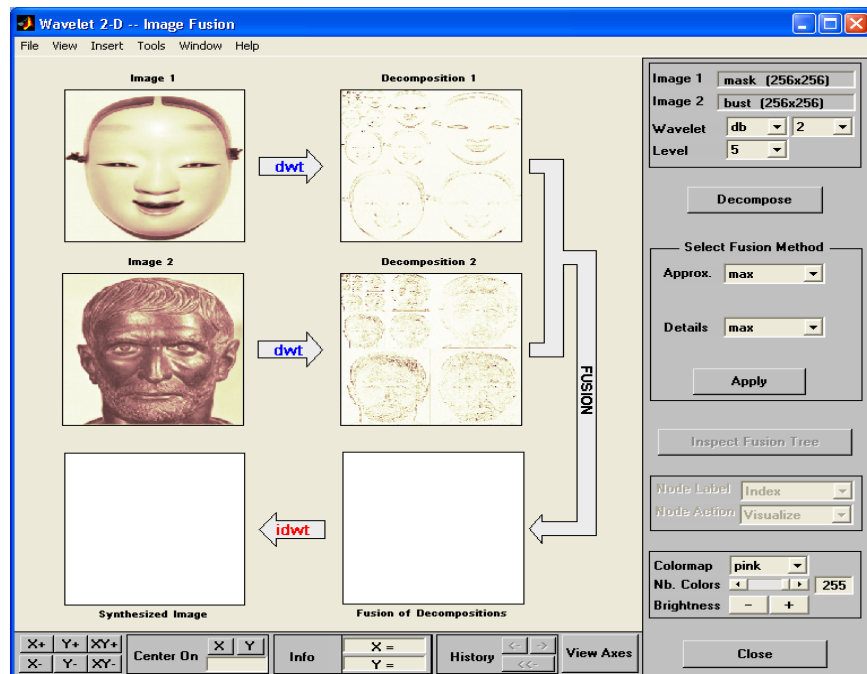
### 3 Perform wavelet decompositions.

Using the **Wavelet** and **Level** menus located to the upper right, determine the wavelet family, the wavelet type, and the number of levels to be used for the analysis.

For this analysis, select the db2 wavelet at level 5.

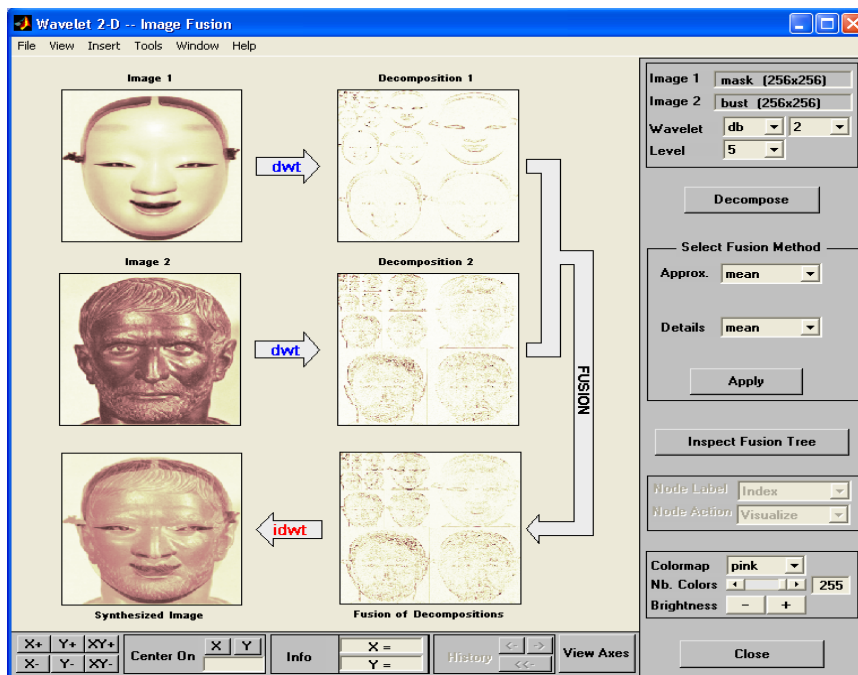
Click the **Decompose** button.

After a pause for computation, the tool displays the two analyses.



### 4 Merge two images from their decompositions.

From **Select Fusion Method** frame, select the item mean for both **Approx.** and **Details**. Next, click the **Apply** button.



The synthesized image and its decomposition (which is equal to the fusion of the two decompositions) appear. The new image produced by fusion clearly exhibits features from the two original ones.

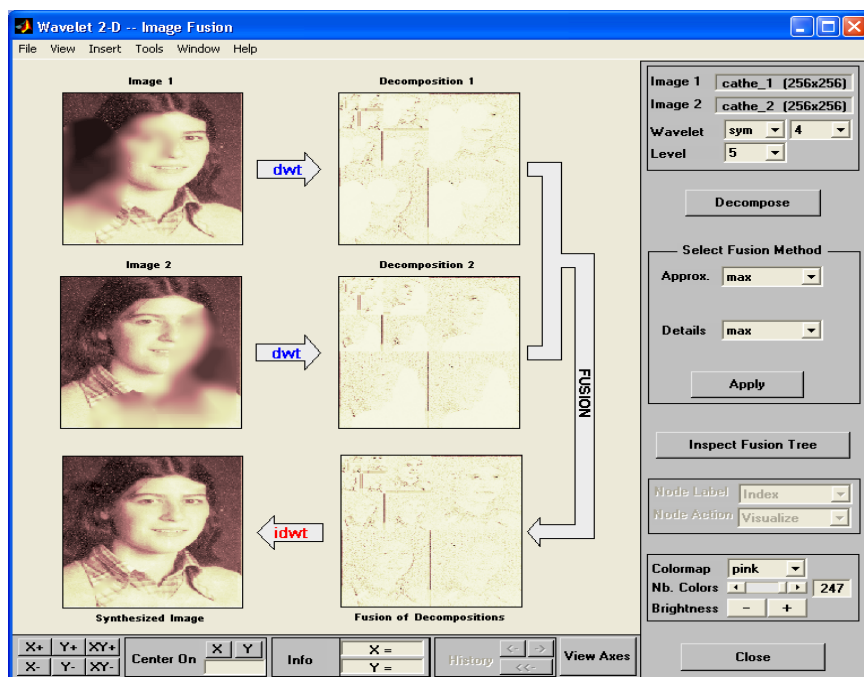
Let us now examine another example illustrating restoration using image fusion.

**5** Restore the image using image fusion.

From the **File** menu, load Image 1 by selecting the demo MAT-file `cathe_1.mat`, and Image 2 by selecting the demo MAT-file `cathe_2.mat`.

**6** Using the **Wavelet** and **Level** menus, select the `sym4` wavelet at level 5. Click the **Decompose** button.

**7** From **Select Fusion Method** frame, select the item `max` for both **Approx.** and **Details**. Next, click the **Apply** button.



The synthesized image is a restored version of good quality of the common underlying original image.

### Saving the Synthesized Image

The Image Fusion Tool lets you save the synthesized image to disk. The toolbox creates a MAT-file in the current directory with a name you choose.

To save the synthesized image from the present selection, use the menu option **File > Save Synthesized Image**.

A dialog box appears that lets you specify a directory and filename for storing the image. After you save the image data to the file `rescathe.mat`, the synthesized image is given by `X` and the colormap by `map`.

## One-Dimensional Fractional Brownian Motion Synthesis

This section takes you through the features of One-Dimensional Fractional Brownian Motion Synthesis using one of the Wavelet Toolbox™ specialized tools.

For the examples in this section, switch the extension mode to symmetric padding, using the command

```
dwtmode('sym')
```

The toolbox requires only one function to generate a fractional Brownian motion signal: `wfbm`. You'll find full information about this function in its reference page.

In this section, you'll learn how to

- Generate a fractional Brownian motion signal
- Look at its main properties
- Save the synthesized signal

Since you can perform the generation either from the command line or using the graphical interface tools, this section has subsections covering each method.

A fractional Brownian motion (fBm) is a continuous-time Gaussian process depending on the Hurst parameter  $0 < H < 1$ . It generalizes the ordinary Brownian motion corresponding to  $H = 0.5$  and whose derivative is the white noise. The fBm is self-similar in distribution and the variance of the increments is given by

$$\text{Var}(fBm(t) - fBm(s)) = v |t - s|^{2H}$$

where  $v$  is a positive constant.

### Fractional Brownian Motion Synthesis Using the Command Line

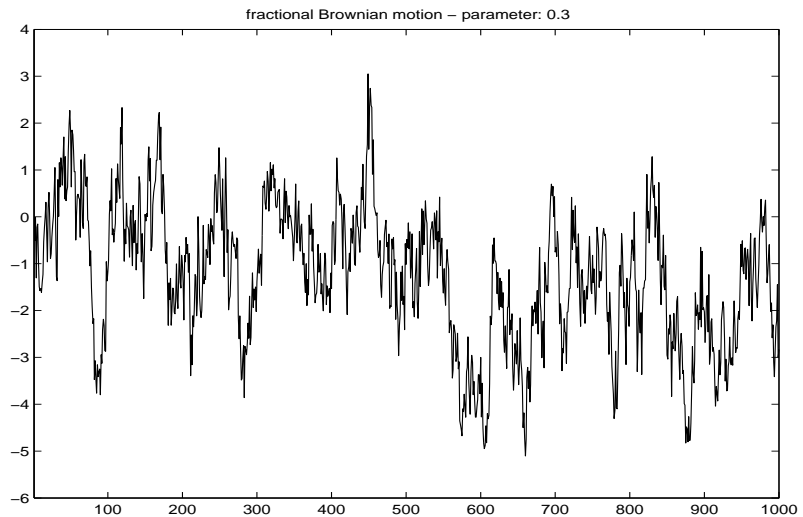
According to the value of  $H$ , the fBm exhibits for  $H > 0.5$ , long-range dependence and for  $H < 0.5$ , short or intermediate dependence.

Let us give an example of each situation using the `wfbm` M-file, which generates a sample path of this process.

```
% Generate fBm for H = 0.3 and H = 0.7

% Initialize the randn generator
randn('state',1)

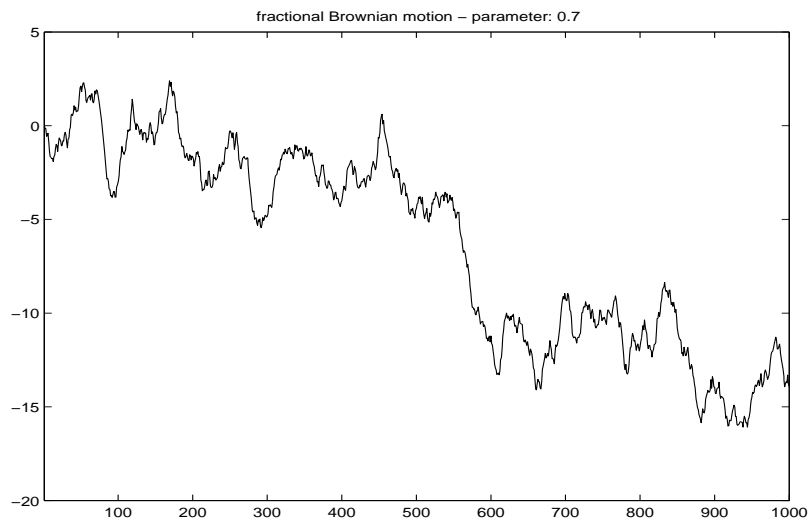
% Set the parameter H and the sample length
H = 0.3; lg = 1000;
% Generate and plot wavelet-based fBm for H = 0.3
fBm03 = wfbm(H,lg,'plot');
```



```
% Reset randn generator and parameter H
randn('state',1); H = 0.7;
% Generate and plot wavelet-based fBm for H = 0.7
fBm07 = wfbm(H,lg,'plot');

% The last step is equivalent to
% Define wavelet and level of decomposition
% w = 'db10'; ns = 6;
% Generate
% fBm07 = wfbm(H,lg,'plot',w,ns);
```





It appears that fBm07 clearly exhibits a stronger low-frequency component and has, locally, a less irregular behavior.

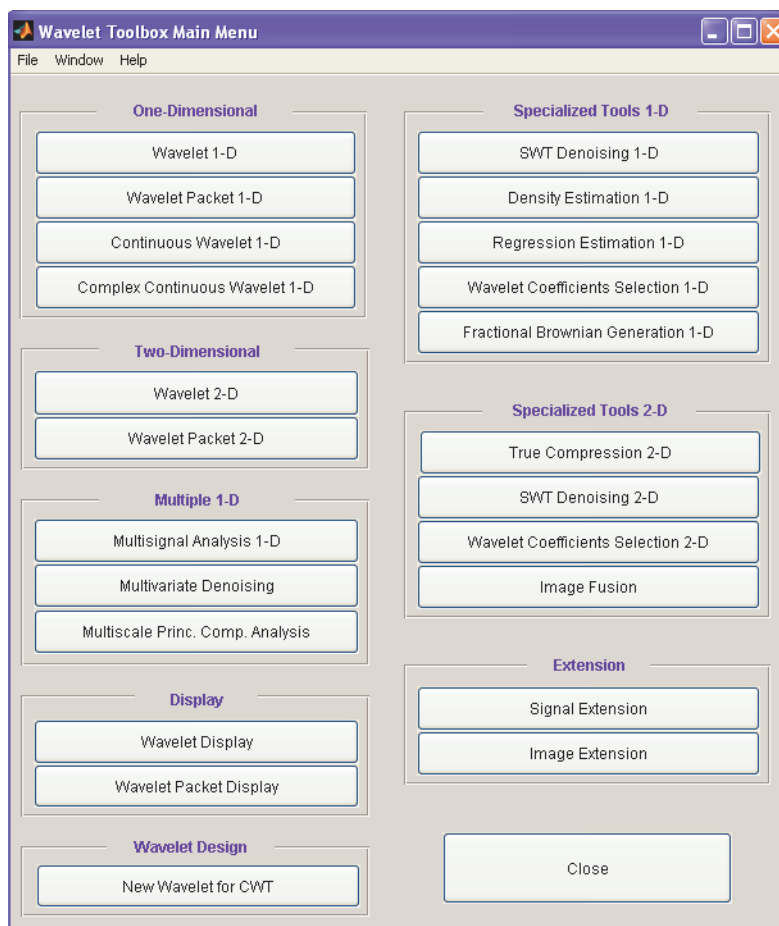
### Fractional Brownian Motion Synthesis Using the Graphical Interface

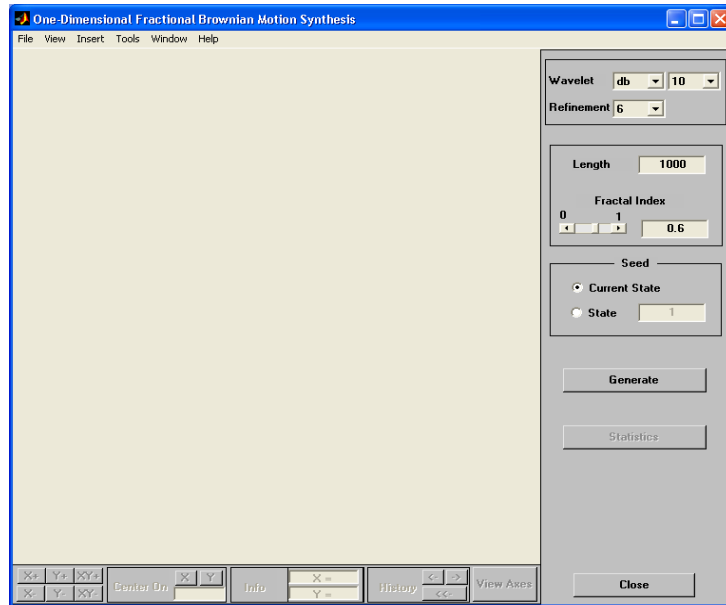
- 1 Start the Fractional Brownian Motion Synthesis Tool.

From the MATLAB<sup>®</sup> prompt, type

```
wavemenu
```

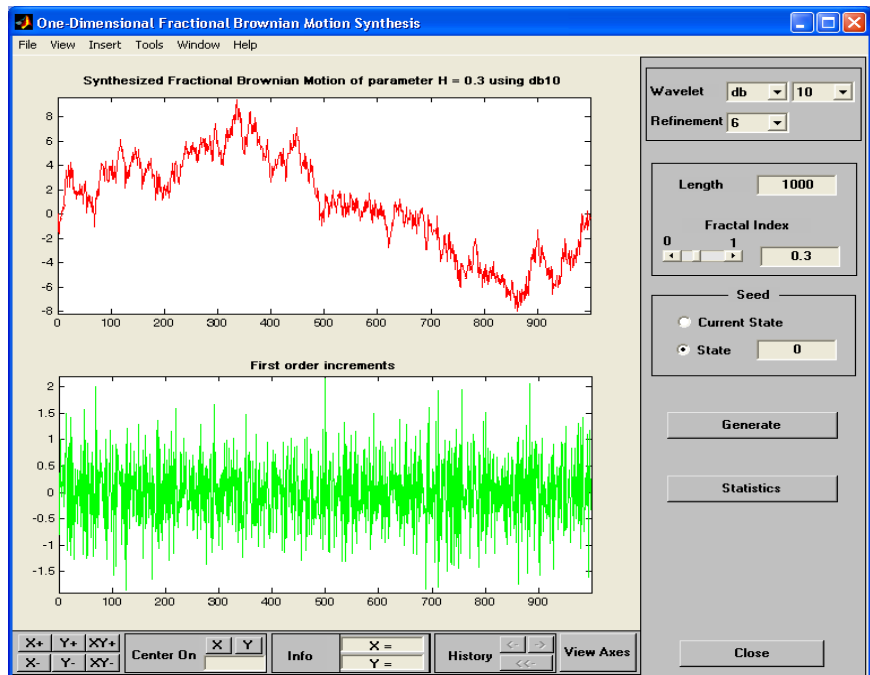
The **Wavelet Toolbox Main Menu** appears. Click **Fractional Brownian Generation 1-D** to display the One-Dimensional Fractional Brownian Motion Synthesis Tool.





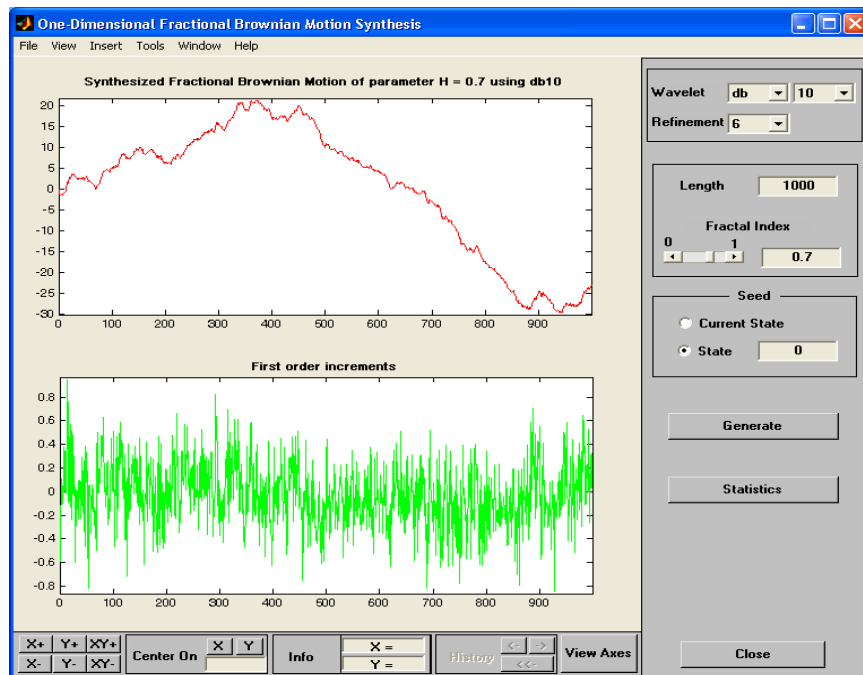
### 2 Generate fBm.

From the **Fractal Index** edit button, type 0.3 and from the **Seed** frame, select the item **State** and set the value to 0. Next, click the **Generate** button.



The synthesized signal exhibits a locally highly irregular behavior.

- 3 Now let us try another value for the fractal index. From the **Fractal Index** edit button, type 0.7 and from the **Seed** frame, select the item **State** and set the value to 0. Next, click the **Generate** button.



The synthesized signal clearly exhibits a stronger low-frequency component and has locally a less irregular behavior. These properties can be investigated by clicking the **Statistics** button.

## Saving the Synthesized Signal

The Fractional Brownian Motion Synthesis Tool lets you save the synthesized signal to disk. The toolbox creates a MAT-file in the current directory with a name you choose.

To save the synthesized signal from the present selection, use the option **File > Save Synthesized Signal**. A dialog box appears that lets you specify a directory and filename for storing the signal. After saving the signal data to the file `fbm07.mat`, load the variables into workspace.

```
load fbm07
```

```
whos
```

<b>Name</b>	<b>Size</b>	<b>Bytes</b>	<b>Class</b>
FBM_PARAMS	1x1	668	struct array
fbm07	1x1000	8000	double array

```
FBM_PARAMS
```

```
FBM_PARAMS =
```

```
    SEED: [2x1 double]  
    Wav: 'db10'  
    Length: 1000  
    H: 0.7000  
    Refinement: 6
```

The synthesized signal is given by `fbm07`. In addition, the parameters of the generation are given by `FBM_PARAMS`, which is a cell array of length 5.

## New Wavelet for CWT

This section takes you through the features of New Wavelet for CWT, one of the Wavelet Toolbox™ specialized tools.

The toolbox requires only one function to design a new wavelet adapted to a given pattern for CWT: `pat2cwav`. You'll find full information about this function in its reference page.

In this section, you'll learn how to

- Load a pattern
- Synthesize a new wavelet adapted to the given pattern
- Detect patterns by CWT using the adapted wavelet
- Compare the detection using both the adapted wavelet and well-known wavelets
- Save the synthesized wavelet

Since you can perform the design of the new wavelet for CWT either from the command line or using the graphical interface tools, this section has subsections covering each method.

The principle for designing a new wavelet for CWT is to approximate a given pattern using least squares optimization under constraints leading to an admissible wavelet well suited for the pattern detection using the continuous wavelet transform (see [MisMOP03] in “References” on page 6-155).

### New Wavelet for CWT Using the Command Line

The following example illustrates how to generate a new wavelet starting from a pattern.

```
% Load original pattern: a pseudo sine one.  
load ptpsin1;  
  
% Variables X and Y contain the pattern.  
whos
```

Name	Size	Bytes	Class
IntVAL	1x1	8	double array
X	1x256	2048	double array
Y	1x256	2048	double array
caption	1x35	70	char array

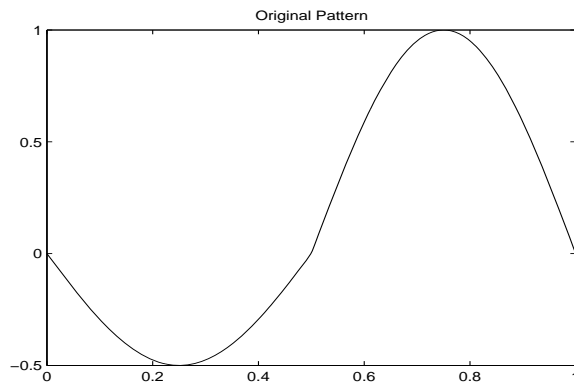
```
% This example is a demo-example, so we have the value of the
% integral of the pattern as well as the details about its
% construction in the caption variable.
```

```
IntVAL
```

```
IntVAL =
```

```
0.1592
```

```
% The pattern defined on the interval [0,1] is of integral 0.1592.
% So it is not a wavelet but it is a good candidate since it
% oscillates like a wavelet.
plot(X,Y), title('Original Pattern')
```



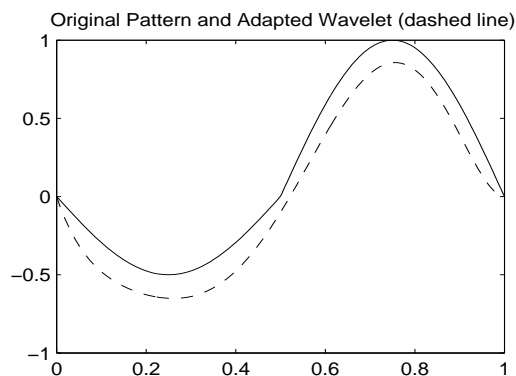


```
% To synthesize a new wavelet adapted to the given pattern, let
% us use a least squares polynomial approximation of degree 6 with
% constraints of continuity at the beginning and the end of the
% pattern.
```

```
[psi,xval,nc] = pat2cwav(Y, 'polynomial',6, 'continuous') ;
```

```
% The new wavelet is given by xval and nc*psi.
```

```
plot(X,Y,'-',xval,nc*psi,'--'),
title('Original Pattern and Adapted Wavelet (dashed line)')
```



```
% Let us notice that the version of the wavelet correctly
% defined in order to be used in the CWT algorithm must be of
% square norm equal to 1. It is simply given by xval and psi.
```

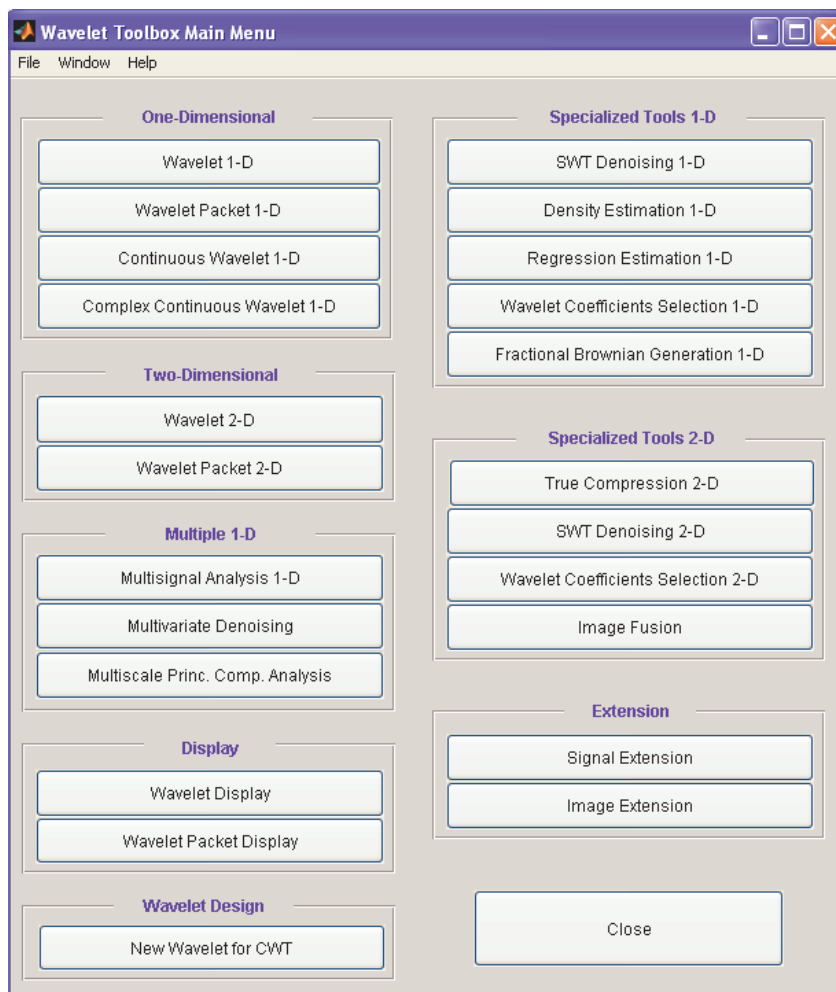
## New Wavelet for CWT Using the Graphical Interface

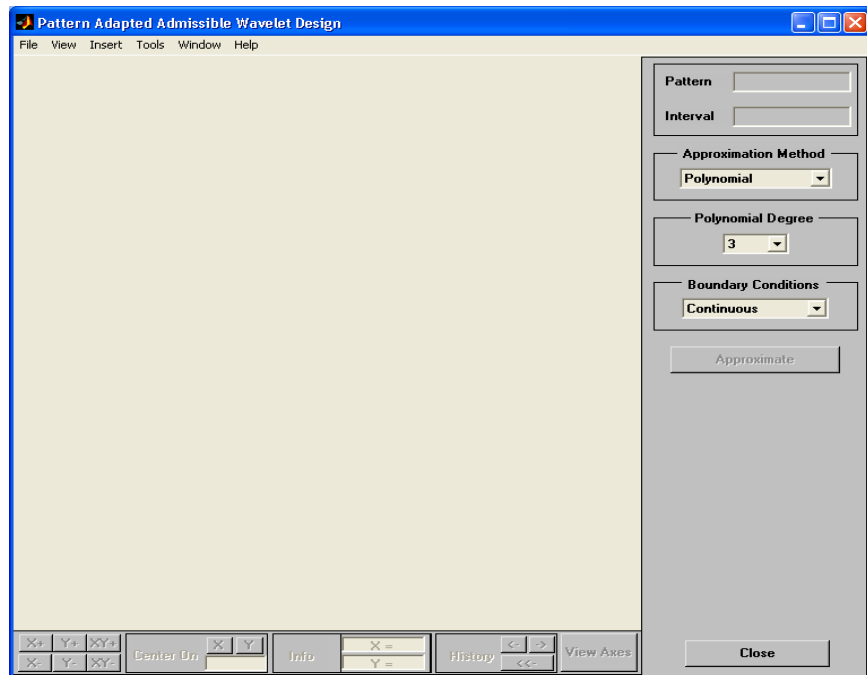
- 1 Start the New Wavelet for CWT Tool.

From the MATLAB<sup>®</sup> prompt, type

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears. Click the **New Wavelet for CWT** menu item to display the Pattern Adapted Admissible Wavelet Design Tool.



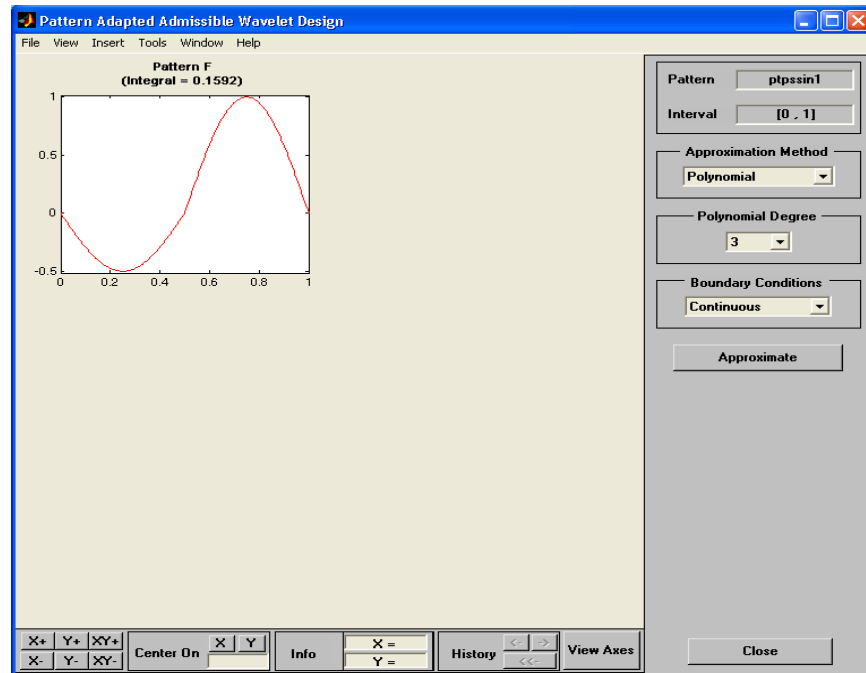


### 2 Load the original pattern.

The MAT-file defining the pattern can contain more than one variable. In that case, the variable  $Y$  is considered if it exists; otherwise, the first variable is considered.

### 3 From the **File** menu, choose the **Load Pattern** option.

When the **Load Pattern** dialog box appears, select the demo MAT-file `ptpssin1.mat`, which should reside in the MATLAB directory `toolbox/wavelet/wavedemo`. Click the **OK** button.

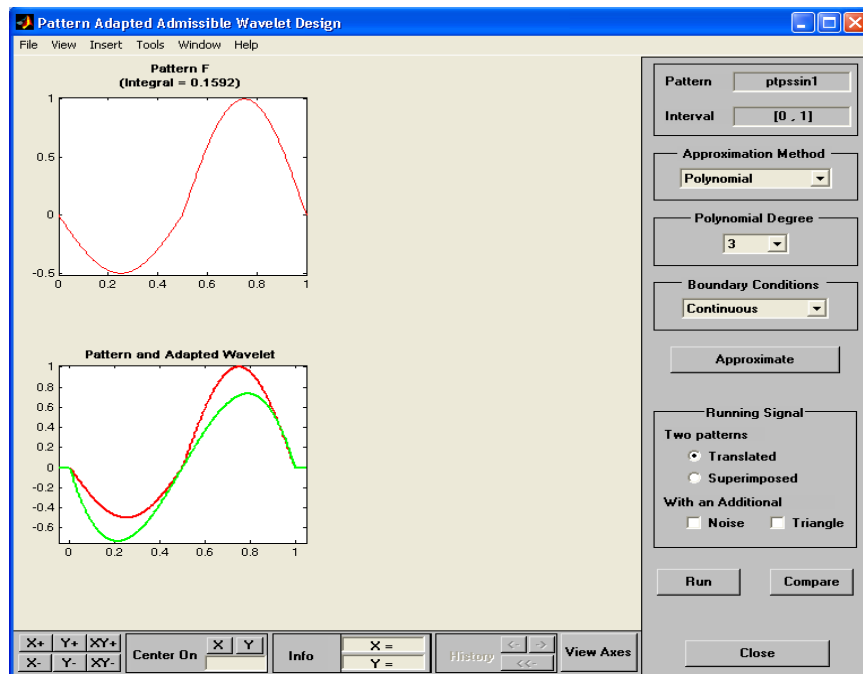


The selected pattern denoted by  $F$  is defined on the interval  $[0, 1]$  and is of integral 0.1592. It is not a wavelet, but it is a good candidate because it oscillates like a wavelet.

#### 4 Perform pattern approximation.

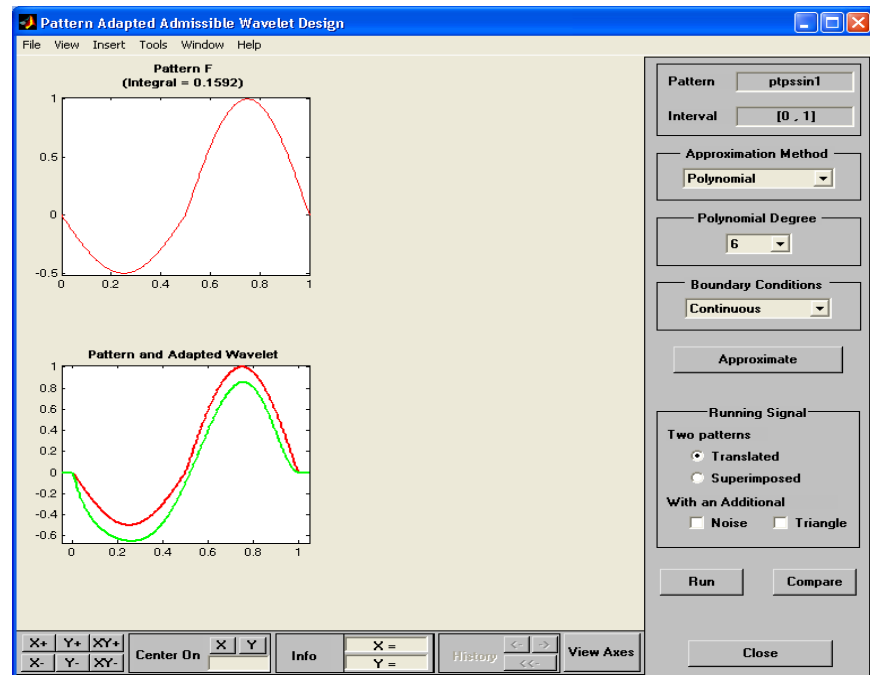
Accept the default parameters leading to use a polynomial of degree 3 with constraints of continuity at the borders 0 and 1, to approximate the pattern  $F$ . Click the **Approximate** button.

After a pause for computation, the tool displays the new wavelet in green superimposed with the original pattern in red.



The result is not really satisfactory. A solution is to increase the polynomial degree to fit better the pattern.

- 5 Using the **Polynomial Degree** menu, increase the degree by selecting 6. Then click the **Approximate** button again.

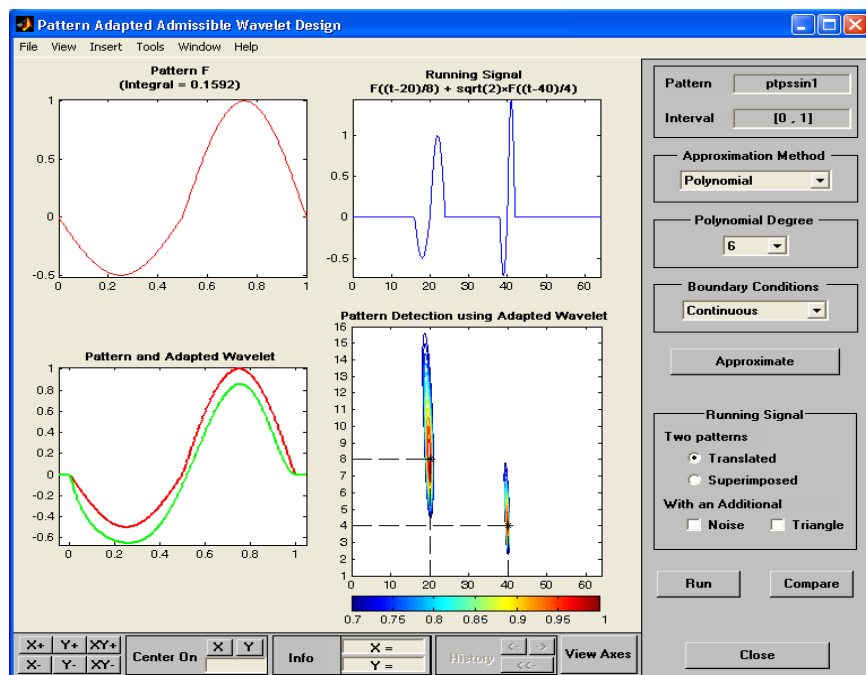


The result is now of good quality and can be used for pattern detection.

## 6 Pattern detection using the new wavelet.

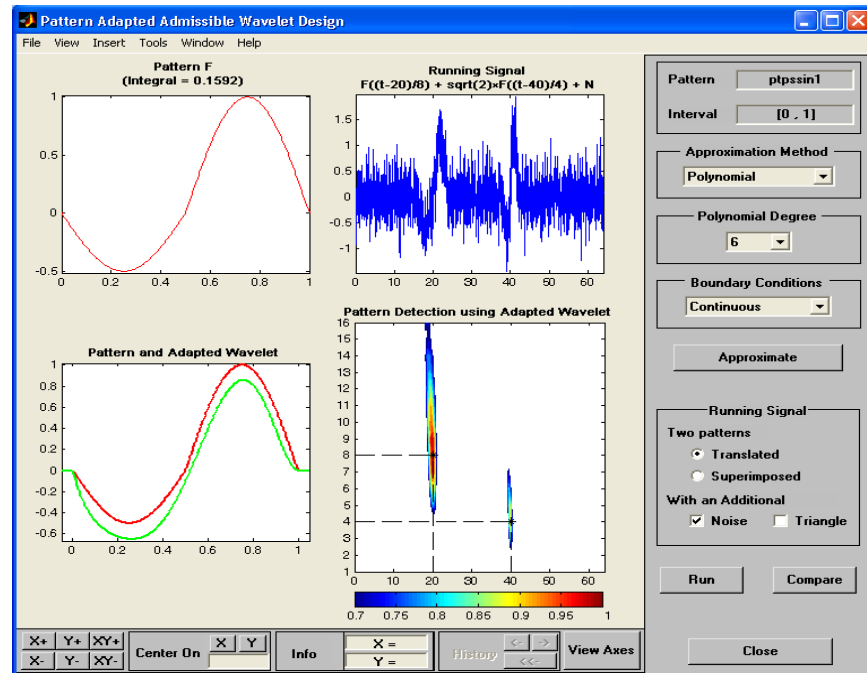
Click the **Run** button.

After a pause for computation, the tool displays the running signal and the pattern detection by CWT using the adapted wavelet.



The running signal is the superimposition of two dilated and translated versions of the pattern  $F$ , namely  $F((t-20)/8)$  and  $F((t-40)/4)$ . The two pairs (position, scale) to be detected are given by  $(20, 8)$  and  $(40, 4)$  and are materialized by dashed lines in the lower right graph of the contour plot of the CWT. The detection is perfect because the two local maxima of the absolute values of the continuous wavelet coefficients fit perfectly.

- 7 Using the **Running signal** frame, select the **Noise** check box to add an additive noise to the previous signal. Click the **Run** button again.

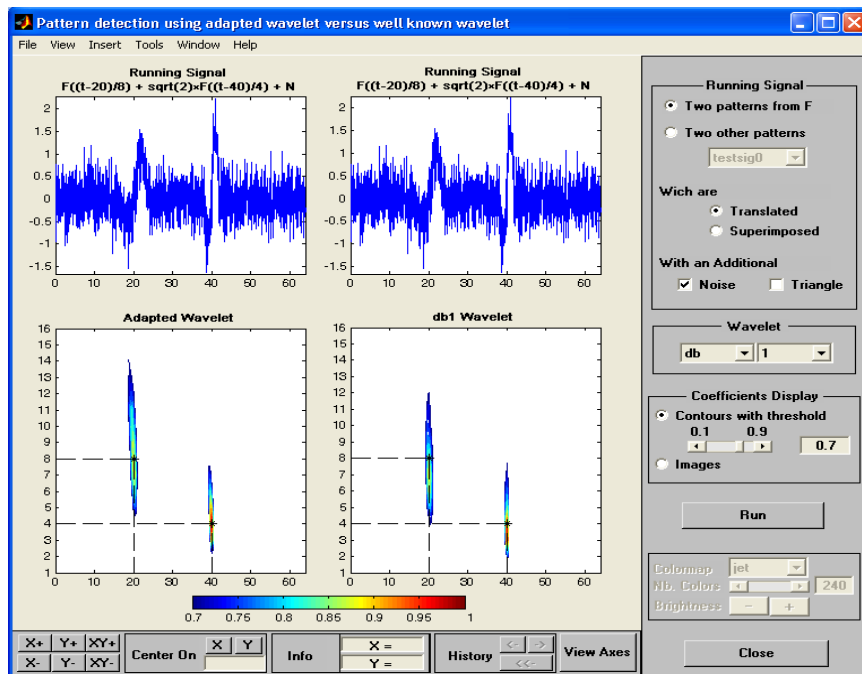


The quality of the detection is not altered at all.

### 8 Compare the adapted wavelet and well-known wavelets.

Let us now compare the performance for pattern detection of the adapted wavelet versus well-known wavelets. Click the **Compare** button. A new window appears.





This tool displays the pattern detection performed with the adapted wavelet on the left and db1 wavelet (default) on the right. The two positions are perfectly detected in both cases but scales are slightly underestimated by the db1 wavelet.

The tool allows you to generate various running signals and choose the wavelet to be compared with the adapted one.

Click the **Close** button to get back to the main window.

## Saving the New Wavelet

The New Wavelet for CWT Tool lets you save the synthesized wavelet. The toolbox creates a MAT-file in the current directory with a name you choose.

To save the new wavelet from the present selection, use the option **File > Save Adapted Wavelet**. A dialog box appears that lets you specify a directory and filename for storing the data. After you save the wavelet data to the file `newwave1.mat`, the adapted wavelet is given by  $X$  and  $Y$ .

Note that the version of the saved wavelet is correctly defined to be used in the CWT algorithm and is such that its square norm is equal to 1.

## Multivariate Wavelet De-Noising

This section demonstrates the features of multivariate de-noising provided in the Wavelet Toolbox™ software. The toolbox includes the `wmulden` function and a graphical user interface (GUI) tool available from `wavemenu`. This section also describes the command-line and GUI methods and includes information about transferring signal and parameter information between the disk and the GUI.

This multivariate wavelet de-noising problem deals with models of the form  $X(t) = F(t) + e(t)$ , where the observation  $X$  is  $p$ -dimensional,  $F$  is the deterministic signal to be recovered, and  $e$  is a spatially correlated noise signal. This kind of model is well suited for situations for which such additive, spatially correlated noise is realistic.

### Multivariate Wavelet De-Noising Using the Command Line

This example uses noisy test signals. In this section, you will

- Load a multivariate signal.
- Display the original and observed signals.
- Remove noise by a simple multivariate thresholding after a change of basis.
- Display the original and de-noised signals.
- Improve the obtained result by retaining less principal components.
- Display the number of retained principal components.
- Display the estimated noise covariance matrix.

**1** Load a multivariate signal by typing the following at the MATLAB® prompt:

```
load ex4mwden
```

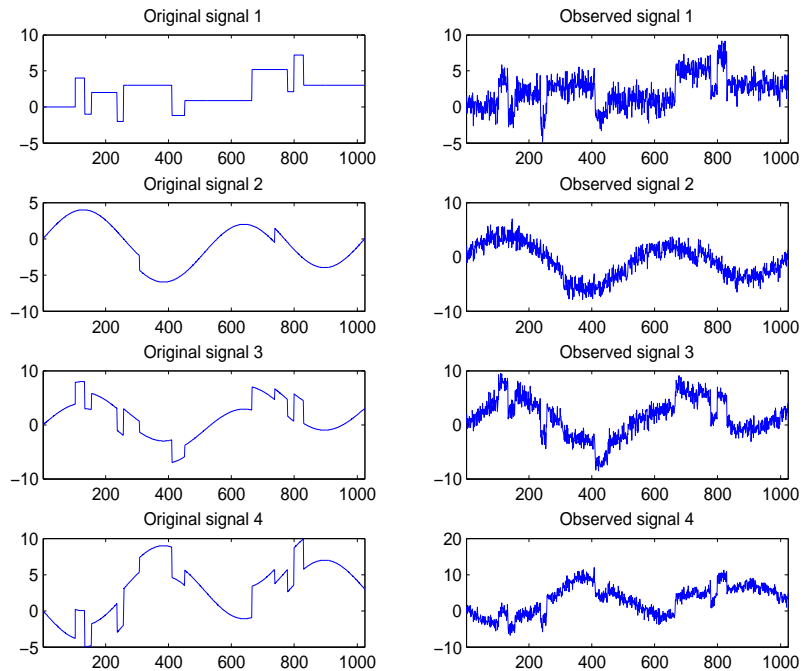
```
whos
```

Name	Size	Bytes	Class
covar	4x4	128	double array
x	1024x4	32768	double array
x_orig	1024x4	32768	double array

Usually, only the matrix of data  $x$  is available. Here, we also have the true noise covariance matrix ( $covar$ ) and the original signals ( $x\_orig$ ). These signals are noisy versions of simple combinations of the two original signals. The first one is “Blocks” which is irregular, and the second is “HeavySine,” which is regular except around time 750. The other two signals are the sum and the difference of the two original signals. Multivariate Gaussian white noise exhibiting strong spatial correlation is added to the resulting four signals, which leads to the observed data stored in  $x$ .

## 2 Display the original and observed signals by typing

```
kp = 0;
for i = 1:4
    subplot(4,2,kp+1), plot(x_orig(:,i));
    title(['Original signal ',num2str(i)])
    subplot(4,2,kp+2), plot(x(:,i));
    title(['Observed signal ',num2str(i)])
    kp = kp + 2;
end
```



The true noise covariance matrix is given by

`covar`

`covar =`

1.0000	0.8000	0.6000	0.7000
0.8000	1.0000	0.5000	0.6000
0.6000	0.5000	1.0000	0.7000
0.7000	0.6000	0.7000	1.0000

### 3 Remove noise by simple multivariate thresholding.

The de-noising strategy combines univariate wavelet de-noising in the basis where the estimated noise covariance matrix is diagonal with noncentered

Principal Component Analysis (PCA) on approximations in the wavelet domain or with final PCA.

First, perform univariate de-noising by typing the following to set the de-noising parameters:

```
level = 5;
wname = 'sym4';
tptr = 'sqtwolog';
sorh = 's';
```

Then, set the PCA parameters by retaining all the principal components:

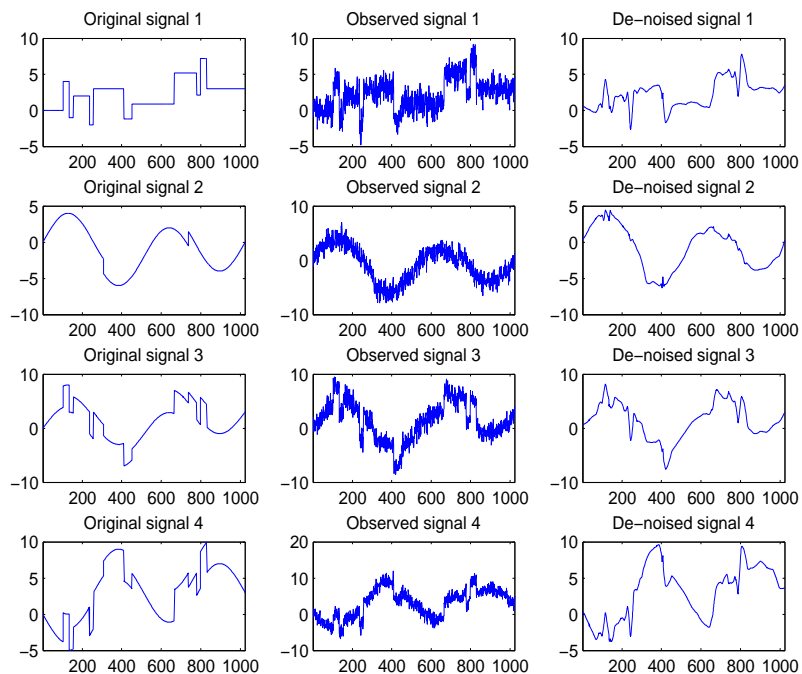
```
npc_app = 4;
npc_fin = 4;
```

Finally, perform multivariate de-noising by typing

```
x_den = wmulden(x, level, wname, npc_app, npc_fin, tptr, sorh);
```

#### 4 Display the original and de-noised signals by typing

```
kp = 0;
for i = 1:4
    subplot(4,3,kp+1), plot(x_orig(:,i));
    title(['Original signal ',num2str(i)])
    subplot(4,3,kp+2), plot(x(:,i));
    title(['Observed signal ',num2str(i)])
    subplot(4,3,kp+3), plot(x_den(:,i));
    title(['De-noised signal ',num2str(i)])
    kp = kp + 3;
end
```



**5** Improve the first result by retaining fewer principal components.

The results are satisfactory. Focusing on the two first signals, note that they are correctly recovered, but the result can be improved by taking advantage of the relationships between the signals, leading to an additional de-noising effect.

To automatically select the numbers of retained principal components by Kaiser's rule (which keeps the components associated with eigenvalues exceeding the mean of all eigenvalues), type

```
npc_app = 'kais';
npc_fin = 'kais';
```

Perform multivariate de-noising again by typing

```
[x_den, npc, nestco] = wmulden(x, level, wname, npc_app, ...
    npc_fin, tptr, sorh);
```

**6** Display the number of retained principal components.

The second output argument gives the numbers of retained principal components for PCA for approximations and for final PCA.

```
npc
npc =
     2     2
```

As expected, since the signals are combinations of two initial ones, Kaiser's rule automatically detects that only two principal components are of interest.

**7** Display the estimated noise covariance matrix.

The third output argument contains the estimated noise covariance matrix:

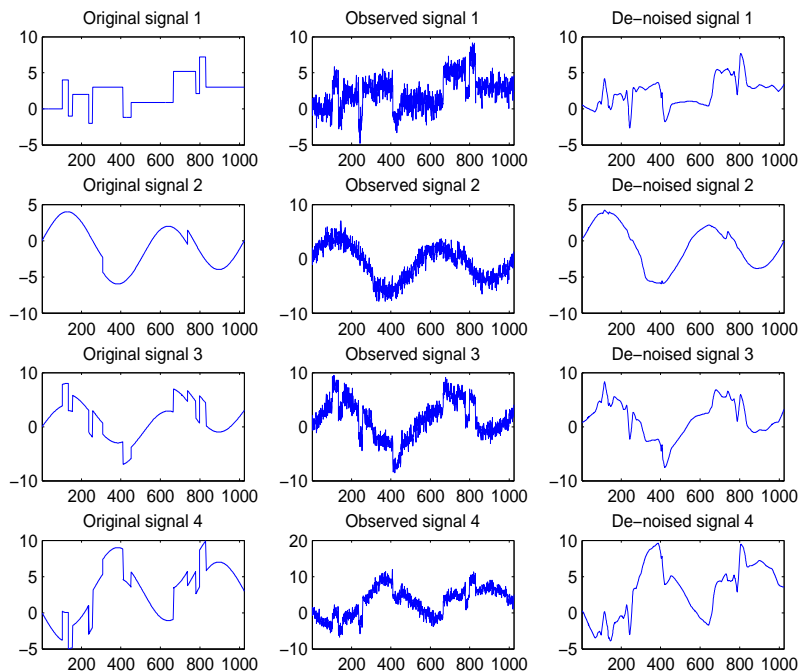
```
nestco
nestco =
    1.0784    0.8333    0.6878    0.8141
    0.8333    1.0025    0.5275    0.6814
    0.6878    0.5275    1.0501    0.7734
    0.8141    0.6814    0.7734    1.0967
```

As you can see by comparing with the true matrix covar given previously, the estimation is satisfactory.

**8** Display the original and final de-noised signals by typing

```
kp = 0;
for i = 1:4
    subplot(4,3,kp+1), plot(x_orig(:,i));
    title(['Original signal ',num2str(i)])
    subplot(4,3,kp+2), plot(x(:,i));
    title(['Observed signal ',num2str(i)])
    subplot(4,3,kp+3), plot(x_den(:,i));
    title(['De-noised signal ',num2str(i)])
    kp = kp + 3;
end
```





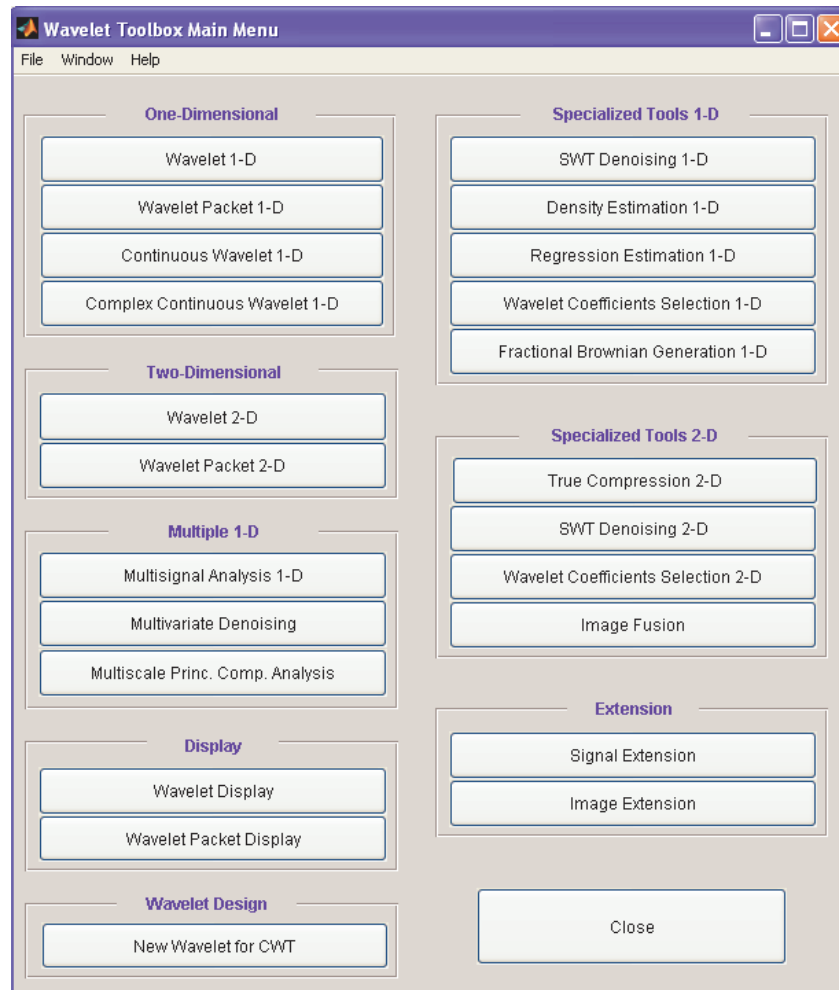
The results are better than those previously obtained. The first signal, which is irregular, is still correctly recovered, while the second signal, which is more regular, is de-noised better after this second stage of PCA.

### Multivariate Wavelet De-Noiseing Using the Graphical Interface

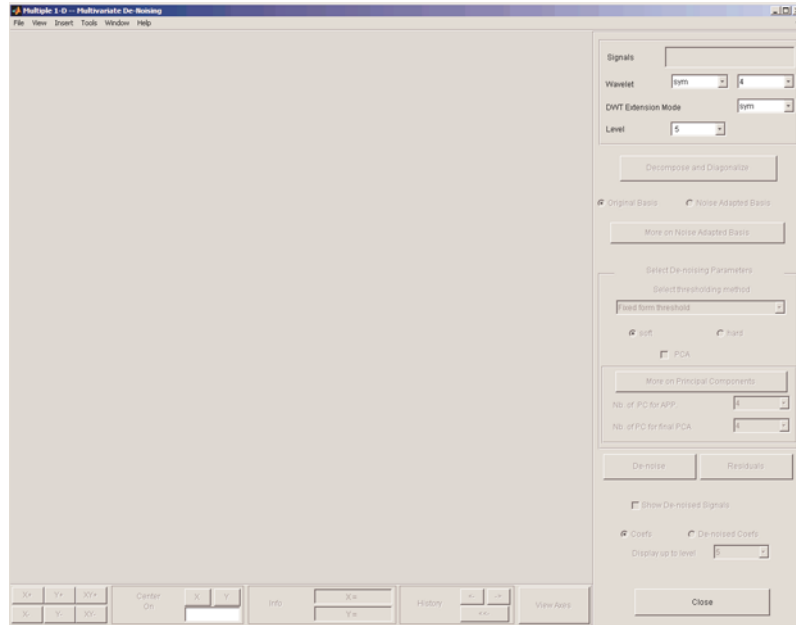
This section explores a de-noising strategy for multivariate signals using the graphical interface tools.

- 1 Start the Multivariate De-noising Tool by first opening the **Wavelet Toolbox Main Menu**.

```
wavemenu
```



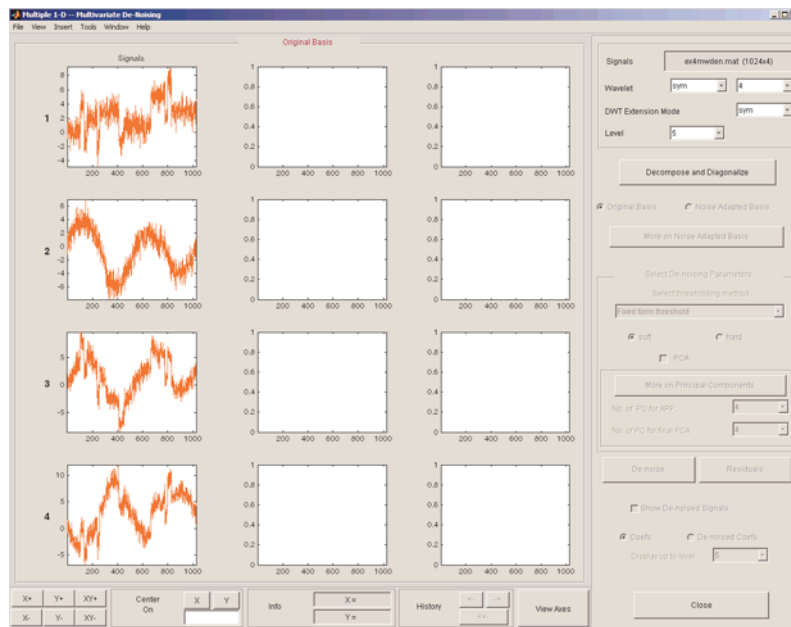
2 Click **Multivariate Denoising** to open the **Multivariate De-Noising GUI**.



### 3 Load data.

Select **File > Load Signals**. In the **Select** dialog box, select the MAT-file `ex4mwden.mat` from the MATLAB directory `toolbox/wavelet/wmultsig1d`.

Click **Open** to load the noisy multivariate signal into the GUI. The signal is a matrix containing four columns, where each column is a signal to be de-noised.



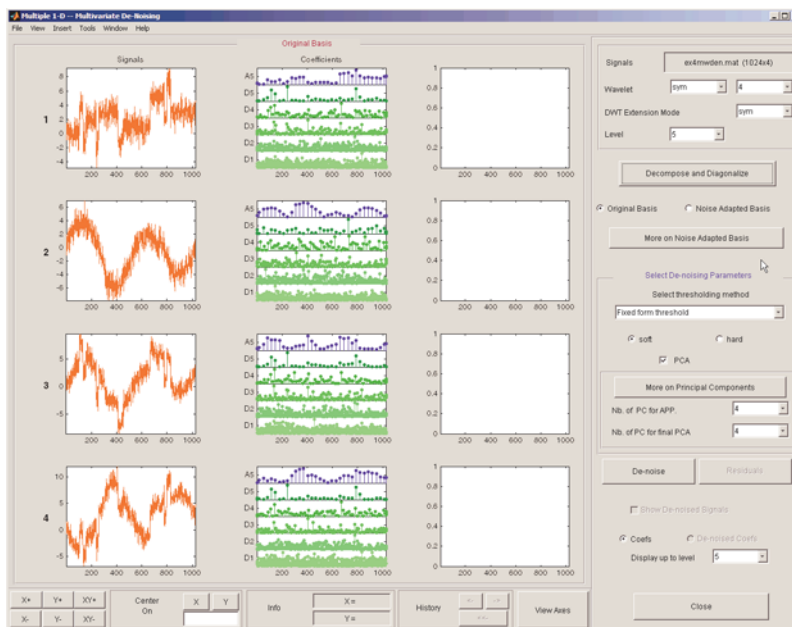
These signals are noisy versions from simple combinations of the two original signals. The first one is “Blocks” which is irregular and the second is “HeavySine” which is regular except around time 750. The other two signals are the sum and the difference between the original signals. Multivariate Gaussian white noise exhibiting strong spatial correlation is added to the resulting four signals.

The following example illustrates the two different aspects of the proposed de-noising method. First, perform a convenient change of basis to cope with spatial correlation and de-noise in the new basis. Then, use PCA to take

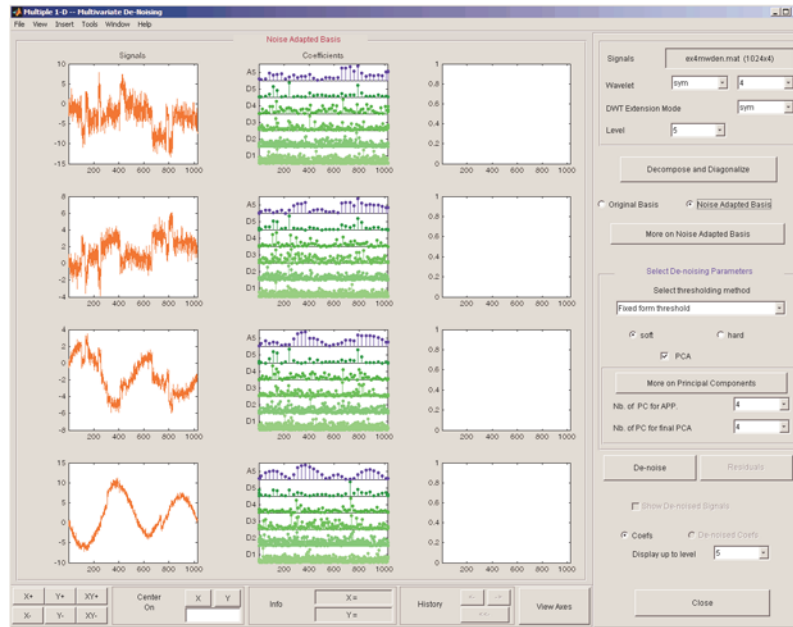
advantage of the relationships between the signals, leading to an additional de-noising effect.

- 4 Perform a wavelet decomposition and diagonalize the noise covariance matrix.

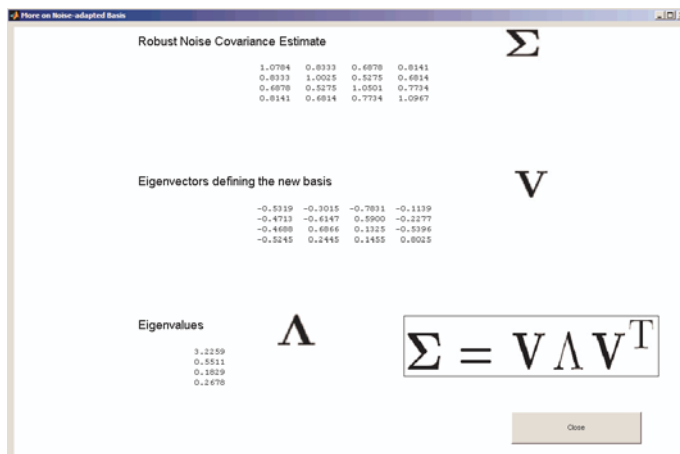
Use the displayed default values for the **Wavelet**, the **DWT Extension Mode**, and the decomposition **Level**, and then click **Decompose and Diagonalize**. The tool displays the wavelet approximation and detail coefficients of the decomposition of each signal in the original basis.



Select **Noise Adapted Basis** to display the signals and their coefficients in the noise-adapted basis.



To see more information about this new basis, click **More on Noise Adapted Basis**. A new figure displays the robust noise covariance estimate matrix and the corresponding eigenvectors and eigenvalues.



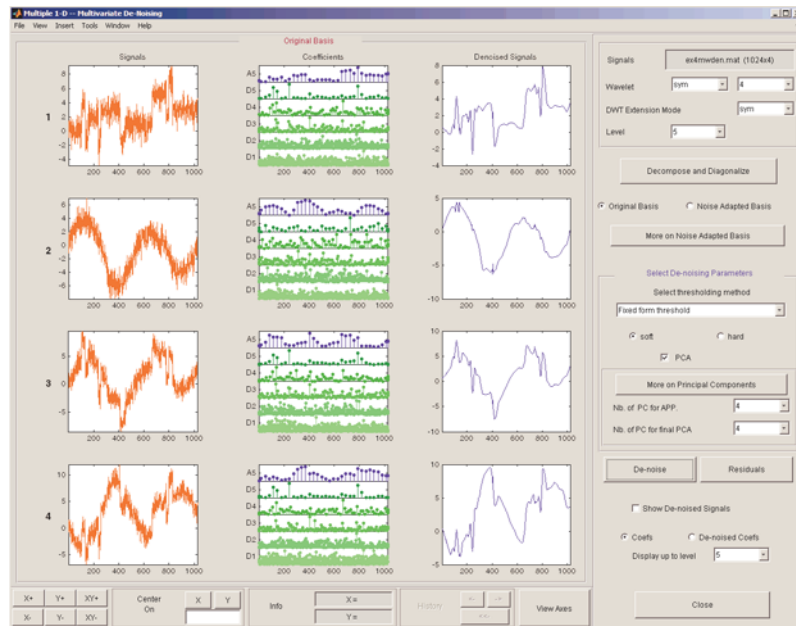
Eigenvectors define the change of basis, and eigenvalues are the variances of uncorrelated noises in the new basis.

The multivariate de-noising method proposed below is interesting if the noise covariance matrix is far from diagonal exhibiting spatial correlation, which, in this example, is the case.

### 5 De-noise the multivariate signal.

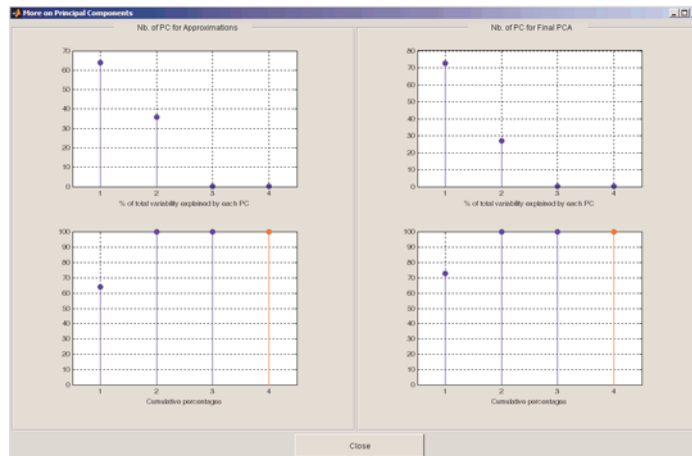
A number of options are available for fine-tuning the de-noising algorithm. However, we will use the defaults: fixed form soft thresholding, scaled white noise model, and the proposed numbers of retained principal components. In this case, the default values for PCA lead to retaining all the components.

Select **Original Basis** to return to the original basis and then click **De-noise**.



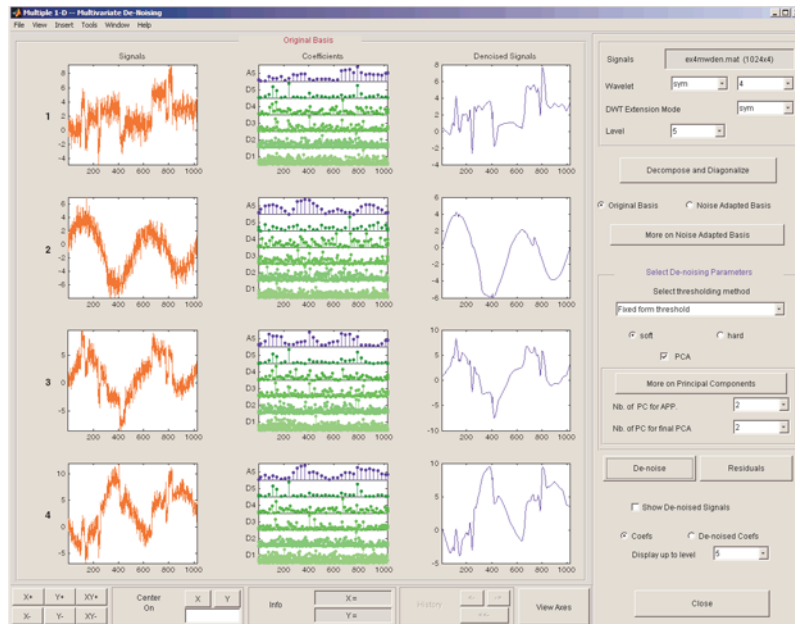
The results are satisfactory. Both of the two first signals are correctly recovered, but they can be improved by getting more information about the principal components. Click **More on Principal Components**.



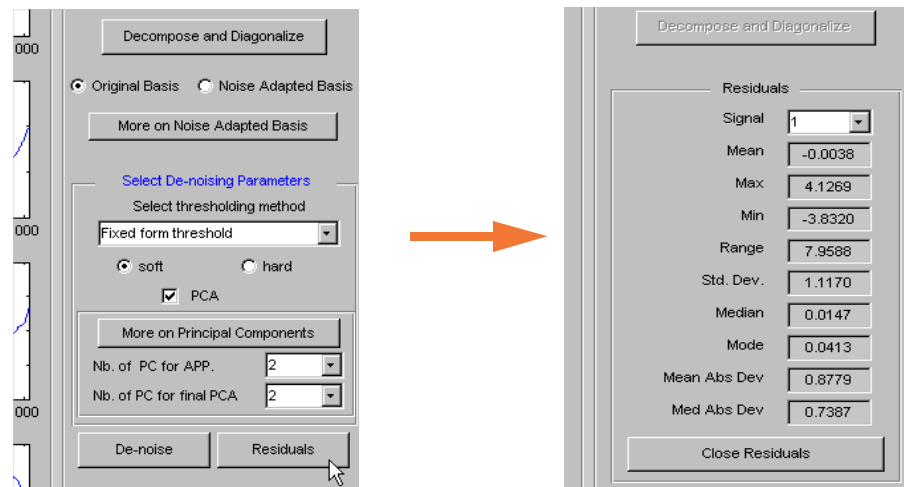


A new figure displays information to select the numbers of components to keep for the PCA of approximations and for the final PCA after getting back to the original basis. You can see the percentages of variability explained by each principal component and the corresponding cumulative plot. Here, it is clear that only two principal components are of interest.

Close the **More on Principal Components** window. Select 2 as the **Nb. of PC for APP**. Select 2 as the **Nb. of PC for final PCA**, and then click **De-noise**.



The results are better than those previously obtained. The first signal, which is irregular, is still correctly recovered. The second signal, which is more regular, is de-noised better after this second stage of PCA. You can get more information by clicking **Residuals**.



### Importing and Exporting from the GUI

The tool lets you save de-noised signals to disk by creating a MAT-file in the current directory with a name of your choice.

To save the signal de-noised in the previous section,

- 1 Select **File > Save De-Noised Signals**.
- 2 Select **Save De-Noised Signals and Parameters**. A dialog box appears that lets you specify a directory and filename for storing the signal.
- 3 Type the name `s_ex4mwden` and click **OK** to save the data.
- 4 Load the variables into your workspace:  

```
load s_ex4mwdent
```

```
whos
```

Name	Size	Bytes	Class
DEN_Params	1x1	430	struct array
PCA_Params	1x1	1536	struct array
x	1024x4	32768	double array

The de-noised signals are in matrix `x`. The parameters (`PCA_Params` and `DEN_Params`) of the two-stage de-noising process are also available.

- `PCA_Params` are the change of basis and PCA parameters:

```
PCA_Params
```

```
PCA_Params =
    NEST: {[4x4 double] [4x1 double] [4x4 double]}
    APP:  {[4x4 double] [4x1 double] [2]}
    FIN:  {[4x4 double] [4x1 double] [2]}
```

`PCA_Params.NEST{1}` contains the change of basis matrix.

`PCA_Params.NEST{2}` contains the eigenvalues, and `PCA_Params.NEST{3}` is the estimated noise covariance matrix.

`PCA_Params.APP{1}` contains the change of basis matrix, `PCA_Params.APP{2}` contains the eigenvalues, and `PCA_Params.APP{3}` is the number of retained principal components for approximations.

The same structure is used for `PCA_Params.FIN` for the final PCA.

- `DEN_Params` are the de-noising parameters in the diagonal basis:

```
DEN_Params
```

```
DEN_Params =
    thrVAL: [4.8445 2.0024 1.1536 1.3957 0]
    thrMETH: 'sqrtwolog'
    thrTYPE: 's'
```

The thresholds are encoded in `thrVAL`. For `j` from 1 to 5, `thrVAL(j)` contains the value used to threshold the detail coefficients at level `j`. The thresholding method is given by `thrMETH` and the thresholding mode is given by `thrTYPE`.

## Multiscale Principal Components Analysis

This section demonstrates the features of multiscale principal components analysis provided in the Wavelet Toolbox™ software. The toolbox includes the `wmspca` function and a graphical user interface (GUI) available from `wavemenu`. This section describes the command-line and GUI methods, and information about transferring signal and parameter information between the disk and the GUI.

The aim of multiscale PCA is to reconstruct, starting from a multivariate signal and using a simple representation at each resolution level, a simplified multivariate signal. The multiscale principal components generalizes the normal PCA of a multivariate signal represented as a matrix by performing a PCA on the matrices of details of different levels simultaneously. A PCA is also performed on the coarser approximation coefficients matrix in the wavelet domain as well as on the final reconstructed matrix. By selecting the numbers of retained principal components, interesting simplified signals can be reconstructed.

For a more detailed introduction and use of multiscale PCA for Statistical Process Control see the paper written by B. Bakshi (see [Bak95] in “References” on page 6-155).

Since you can perform multiscale PCA either from the command line or using the GUI, this section has subsections covering each method.

### Multiscale Principal Components Analysis Using the Command Line

This example uses noisy test signals. In this section, you will:

- Load a multivariate signal.
- Perform a simple multiscale PCA.
- Display the original and simplified signals.
- Improve the obtained result by retaining less principal components.

1 Load a multivariate signal by typing at the MATLAB® prompt:

```
load ex4mwden
```

whos

<b>Name</b>	<b>Size</b>	<b>Bytes</b>	<b>Class</b>
covar	4x4	128	double array
x	1024x4	32768	double array
x_orig	1024x4	32768	double array

The data stored in matrix `x` comes from two test signals, `Blocks` and `HeavySine`, and from their sum and difference, to which multivariate Gaussian white noise has been added.

## 2 Perform a simple multiscale PCA.

The multiscale PCA combines noncentered PCA on approximations and details in the wavelet domain and a final PCA. At each level, the most significant principal components are selected.

First, set the wavelet parameters:

```
level = 5;  
wname = 'sym4';
```

Then, automatically select the number of retained principal components using Kaiser's rule by typing

```
npc = 'kais';
```

Finally, perform multiscale PCA:

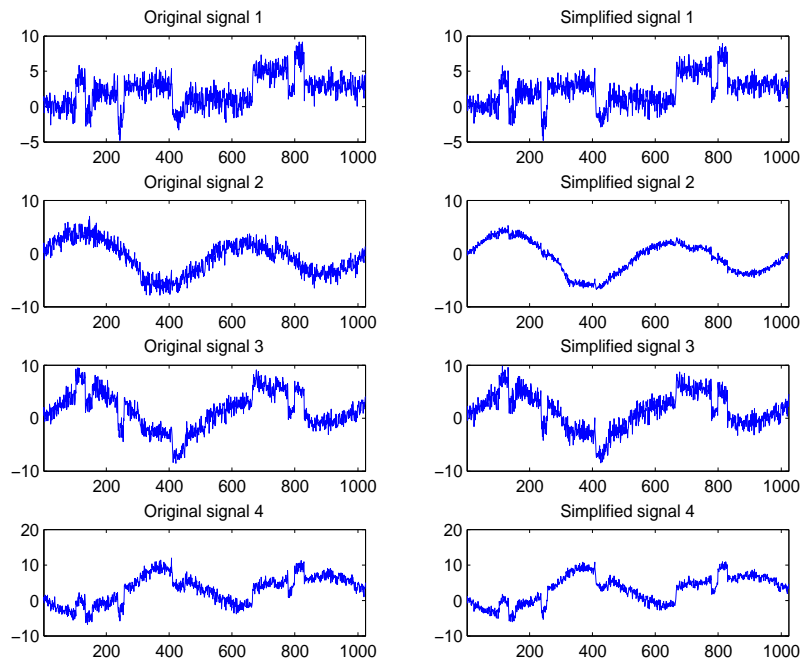
```
[x_sim, qual, npc] = wmspca(x ,level, wname, npc);
```

**3** Display the original and simplified signals:

```

kp = 0;
for i = 1:4
    subplot(4,2,kp+1), plot(x (:,i));
    title(['Original signal ',num2str(i)])
    subplot(4,2,kp+2), plot(x_sim(:,i));
    title(['Simplified signal ',num2str(i)])
    kp = kp + 2;
end

```



The results from a compression perspective are good. The percentages reflecting the quality of column reconstructions given by the relative mean square errors are close to 100%.

```
qual
```

```
qual =
```

```
98.0545    93.2807    97.1172    98.8603
```



#### 4 Improve the first result by retaining fewer principal components.

The results can be improved by suppressing noise, because the details at levels 1 to 3 are composed essentially of noise with small contributions from the signal. Removing the noise leads to a crude, but large, denoising effect.

The output argument `npc` contains the numbers of retained principal components selected by Kaiser's rule:

```
npc
npc =
     1     1     1     1     1     2     2
```

For  $d$  from 1 to 5, `npc(d)` is the number of retained noncentered principal components (PCs) for details at level  $d$ . The number of retained noncentered PCs for approximations at level 5 is `npc(6)`, and `npc(7)` is the number of retained PCs for final PCA after wavelet reconstruction. As expected, the rule keeps two principal components, both for the PCA approximations and the final PCA, but one principal component is kept for details at each level.

To suppress the details at levels 1 to 3, update the `npc` argument as follows:

```
npc(1:3) = zeros(1,3);
npc
npc =
     0     0     0     1     1     2     2
```

Then, perform multiscale PCA again:

```
[x_sim, qual, npc] = wmspca(x, level, wname, npc);
```

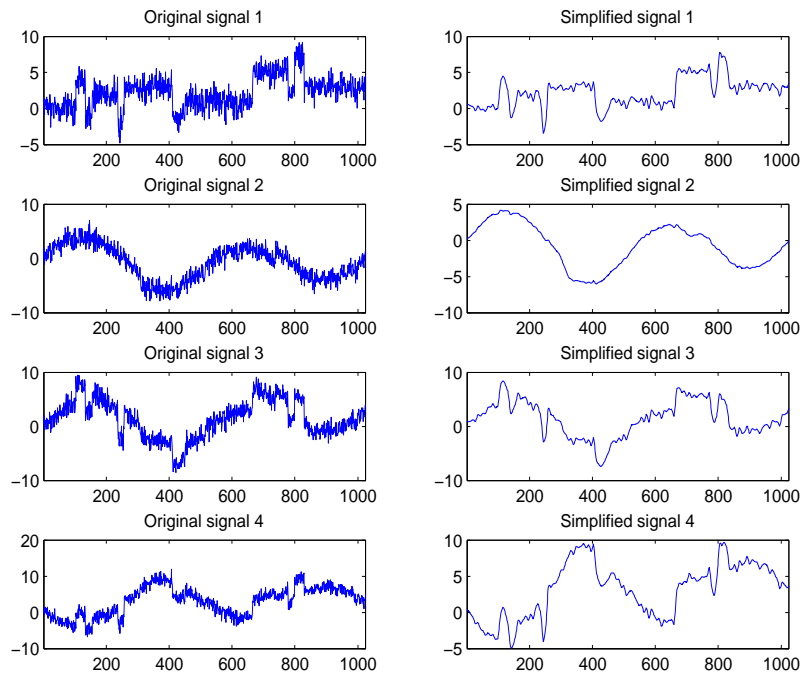
#### 5 Display the original and final simplified signals:

```
kp = 0;
for i = 1:4
    subplot(4,2,kp+1), plot(x(:,i));
    title(['Original signal ', num2str(i)])
    subplot(4,2,kp+2), plot(x_sim(:,i));
    title(['Simplified signal ', num2str(i)])
end
```

```

kp = kp + 2;
end

```



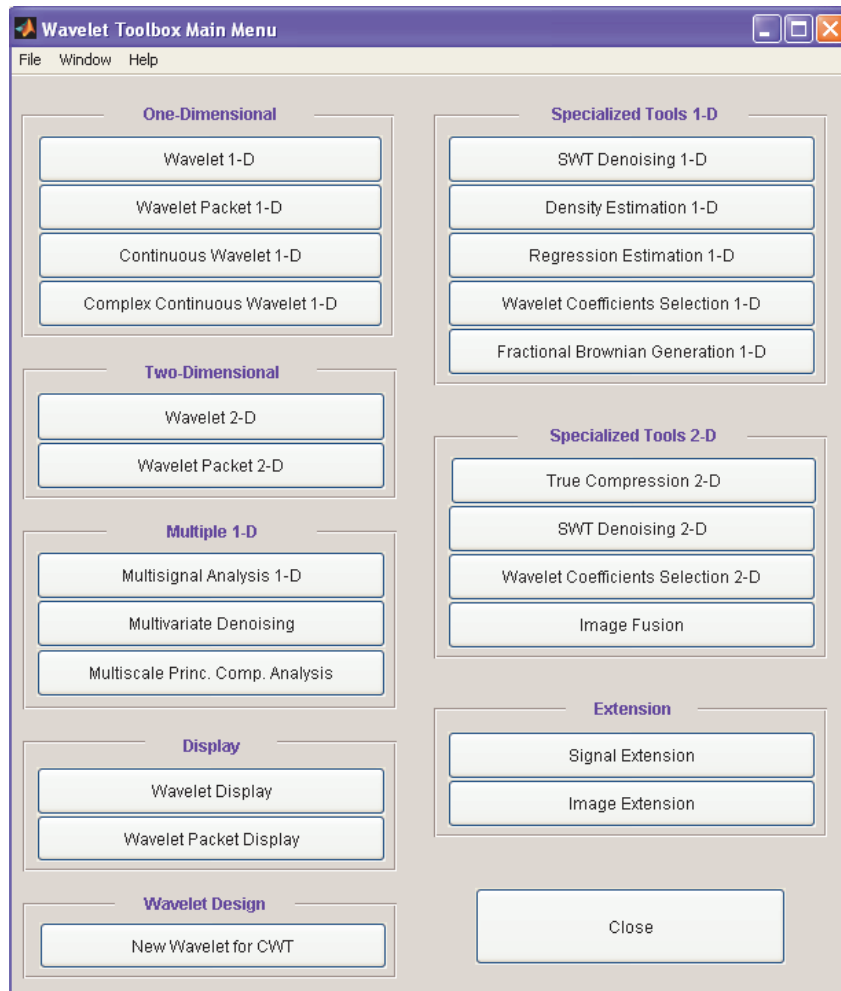
As shown, the results are improved.

## Multiscale Principal Components Analysis Using the Graphical Interface

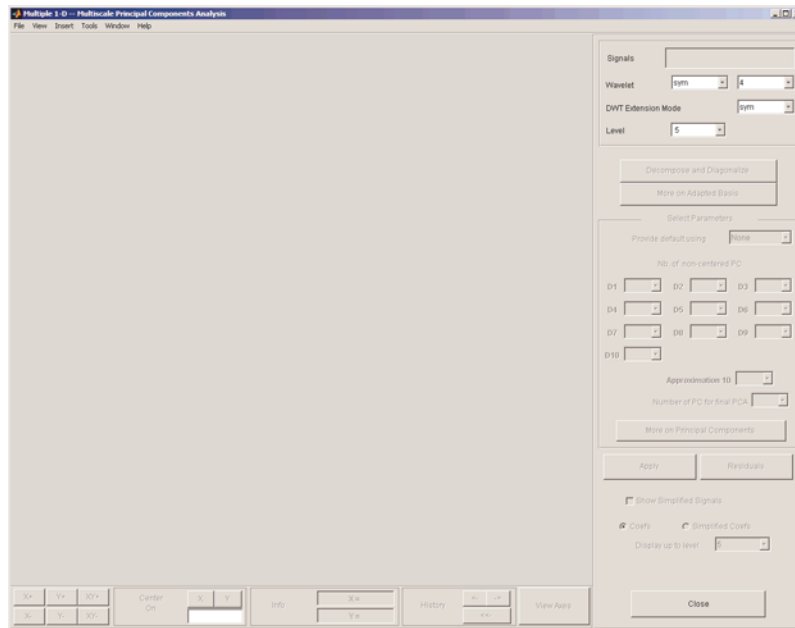
This section explores multiscale PCA using the GUIs.

- 1 Start the Multiscale Princ. Comp. Analysis tool by first opening the Wavelet Toolbox Main Menu:

```
wavemenu
```



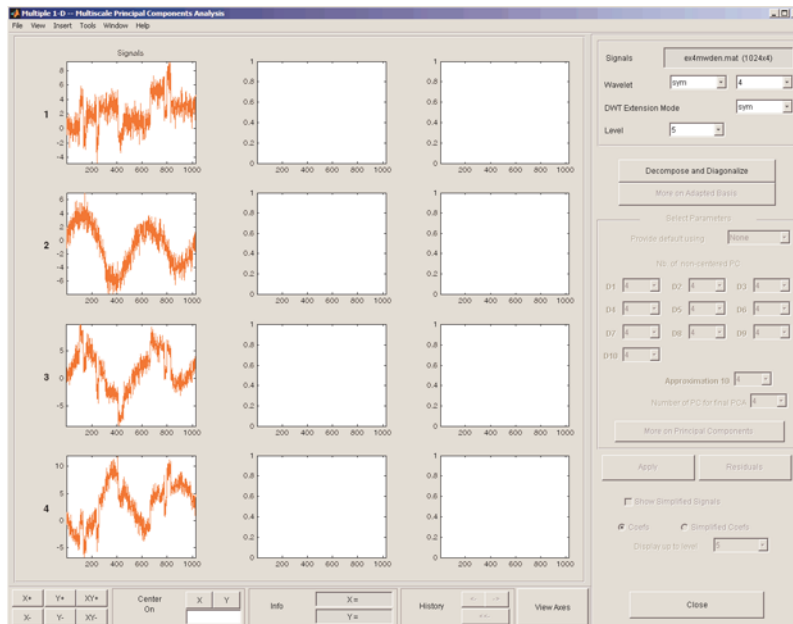
**2 Click Multiscale Princ. Comp. Analysis to open the Multiscale Principal Components Analysis GUI.**



**3 Load data.**

Select **File > Load Signals**. In the **Select** dialog box, select the MAT-file `ex4mwden.mat` from the MATLAB directory `toolbox/wavelet/wmultsig1d`.

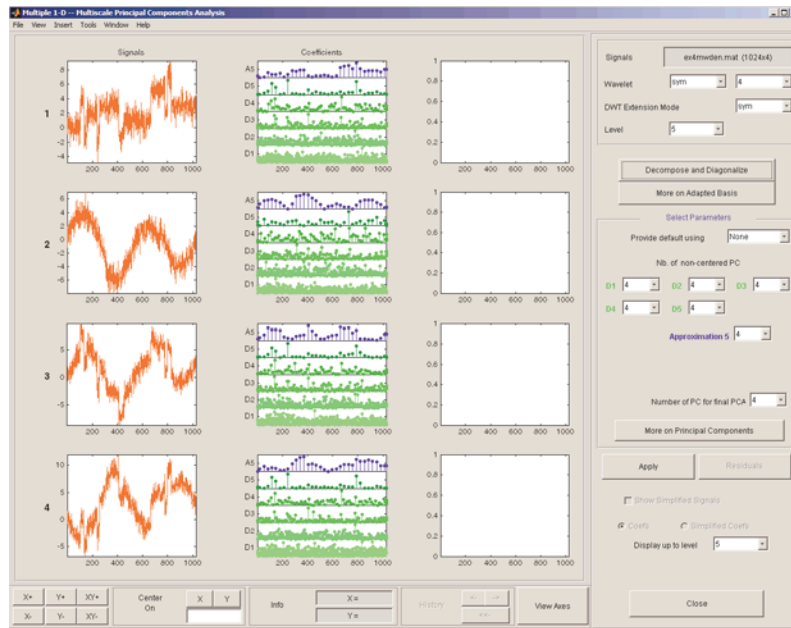
Click **Open** to load the multivariate signal into the GUI. The signal is a matrix containing four columns, where each column is a signal to be simplified.



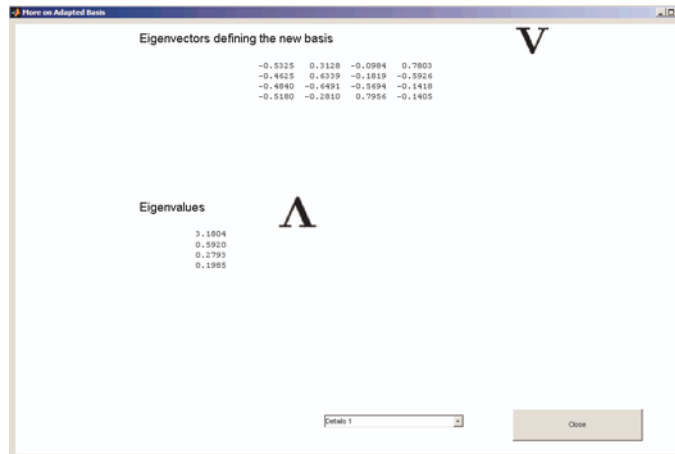
These signals are noisy versions from simple combinations of the two original signals, Blocks and HeavySine and their sum and difference, each with added multivariate Gaussian white noise.

#### 4 Perform a wavelet decomposition and diagonalize each coefficients matrix.

Use the default values for the **Wavelet**, the **DWT Extension Mode**, and the decomposition **Level**, and then click **Decompose and Diagonalize**. The tool displays the wavelet approximation and detail coefficients of the decomposition of each signal in the original basis.



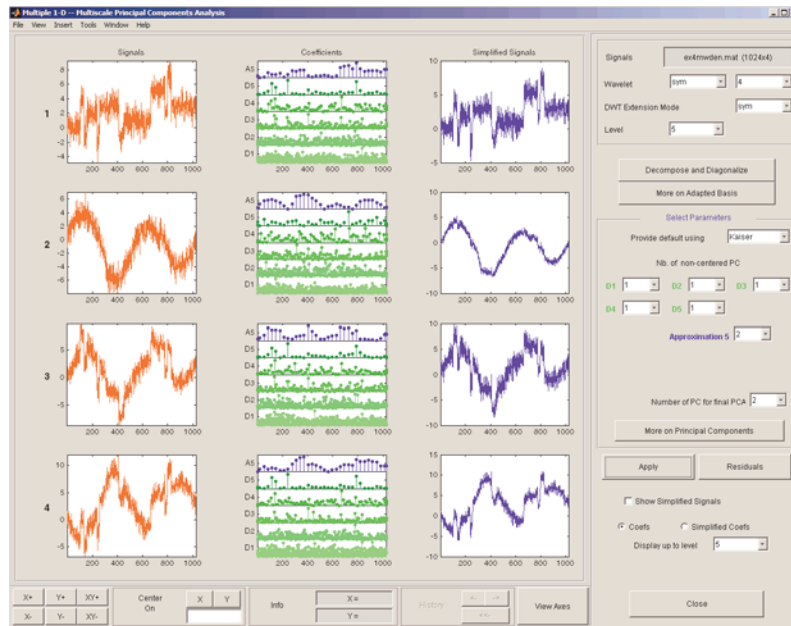
To get more information about the new bases allowed for performing a PCA for each scale, click **More on Adapted Basis**. A new figure displays the corresponding eigenvectors and eigenvalues for the matrix of the detail coefficients at level 1.



You can change the level or select the coarser approximations or the reconstructed matrix to investigate the different bases. When you finish, click **Close**.

### 5 Perform a simple multiscale PCA.

The initial values for PCA lead to retaining all the components. Select Kaiser from the **Provide default using** drop-down list, and click **Apply**.



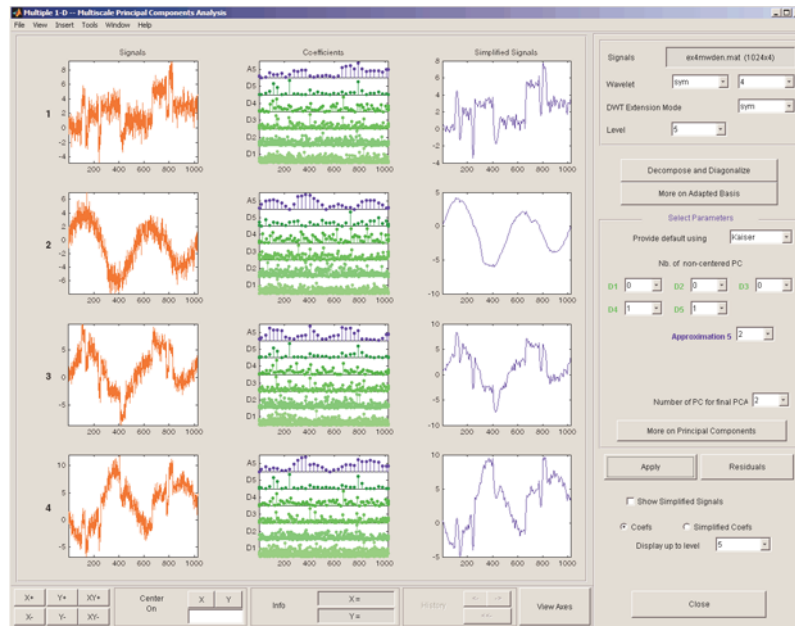
The results are good from a compression perspective.

## 6 Improve the obtained result by retaining fewer principal components.

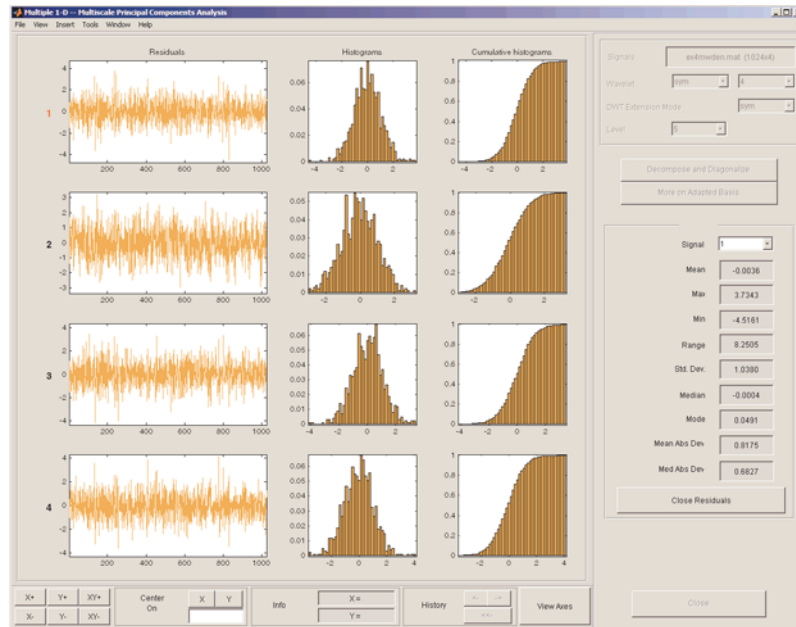
The results can be improved by suppressing the noise, because the details at levels 1 to 3 are composed essentially of noise with small contributions from the signal, as you can see by careful inspection of the detail coefficients. Removing the noise leads to a crude, but large, de-noising effect.

For **D1**, **D2** and **D3**, select 0 as the **Nb. of non-centered PC** and click **Apply**.





The results are better than those previously obtained. The first signal, which is irregular, is still correctly recovered, while the second signal, which is more regular, is de-noised better after this second stage of PCA. You can get more information by clicking **Residuals**.



## Importing and Exporting from the GUI

The Multiscale Principal Components Analysis tool lets you save the simplified signals to disk. The toolbox creates a MAT-file in the current directory with a name of your choice.

To save the simplified signals from the previous section:

- 1 Select **File > Save Simplified Signals**.
- 2 Select **Save Simplified Signals and Parameters**. A dialog box appears that lets you specify a directory and file name for storing the signal.
- 3 Type the name `s_ex4mwden` and click **OK** to save the data.
- 4 Load the variables into your workspace:
 

```
load s_ex4mwden
```

```
whos
```

<b>Name</b>	<b>Size</b>	<b>Bytes</b>	<b>Class</b>
PCA_Params	1x7	2628	struct array
x	1024x4	32768	double array

The simplified signals are in matrix x. The parameters of multiscale PCA are available in PCA\_Params:

```
PCA_Params
```

```
PCA_Params =  
1x7 struct array with fields:  
    pc  
    variances  
    npc
```

PCA\_Params is a structure array of length d+2 (here, the maximum decomposition level d=5) such that PCA\_Params(d).pc is the matrix of principal components. The columns are stored in descending order of the variances. PCA\_Params(d).variances is the principal component variances vector, and PCA\_Params(d).npc is the vector of selected numbers of retained principal components.

## One-Dimensional Multisignal Analysis

This section takes you through the features of one-dimensional multisignal wavelet analysis, compression and denoising using the Wavelet Toolbox™ software. The rationale for each topic is the same as in the 1-D single signal case (for details, see Chapter 6, “Advanced Concepts”).

The toolbox provides the following functions for multisignal analysis.

### Analysis-Decomposition and Synthesis-Reconstruction Functions

Function Name	Purpose
mdwtdec	Multisignal wavelet decomposition
mdwtrec	Multisignal wavelet reconstruction and extraction of approximation and detail coefficients

### Decomposition Structure Utilities

Function Name	Purpose
chgwdeccfs	Change multisignal 1-D decomposition coefficients
wdcenergy	Multisignal 1-D decomposition energy repartition

## Compression and Denoising Functions

Function Name	Purpose
mswcmp	Multisignal 1-D compression using wavelets
mswcmpscr	Multisignal 1-D wavelet compression scores
mswcmptp	Multisignal 1-D compression thresholds and performance
mswden	Multisignal 1-D denoising using wavelets
mswthresh	Perform multisignal 1-D thresholding

You can perform analyses from the MATLAB<sup>®</sup> command line or by using the graphical interface tools. This section describes each method. The last section discusses how to exchange signal and coefficient information between the disk and the graphical tools.

## One-Dimensional Multisignal Analysis Using the Command Line

**1** Load a file, from the MATLAB prompt, by typing

```
load thinker
```

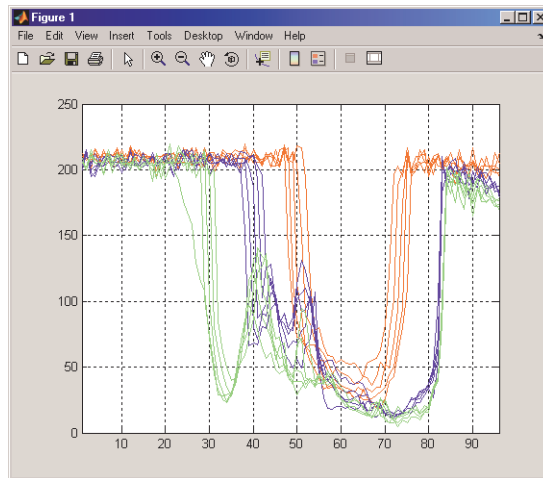
The file `thinker.mat` contains a single variable `X`. Use `whos` to show information about `X`.

```
whos
```

Name	Size	Bytes	Class
X	192x96	147456	double array

**2** Plot some signals.

```
figure;
plot(X(1:5,:),'r'); hold on
plot(X(21:25,:),'b'); plot(X(31:35,:),'g')
set(gca,'Xlim',[1,96])
grid
```

**3** Perform a wavelet decomposition of signals at level 2 of row signals using the db2 wavelet.

```
dec = mdwtdec('r',X,2,'db2')
```

This generates the decomposition structure dec:

```
dec =
```

```
dirDec: 'r'
level: 2
wname: 'db2'
dwtFilters: [1x1 struct]
dwtEXTM: 'sym'
dwtShift: 0
dataSize: [192 96]
ca: [192x26 double]
cd: {[192x49 double] [192x26 double]}
```

### 4 Change wavelet coefficients.

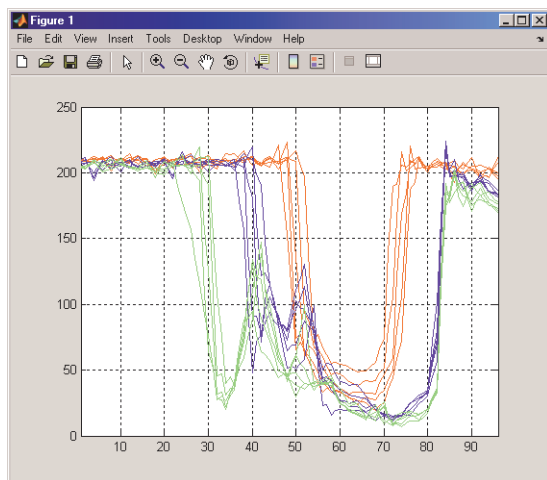
For each signal change the wavelet coefficients by setting all the coefficients of the detail of level 1 to zero.

```
decBIS = chgwdeccfs(dec, 'cd', 0, 1);
```

This generates a new decomposition structure decBIS.

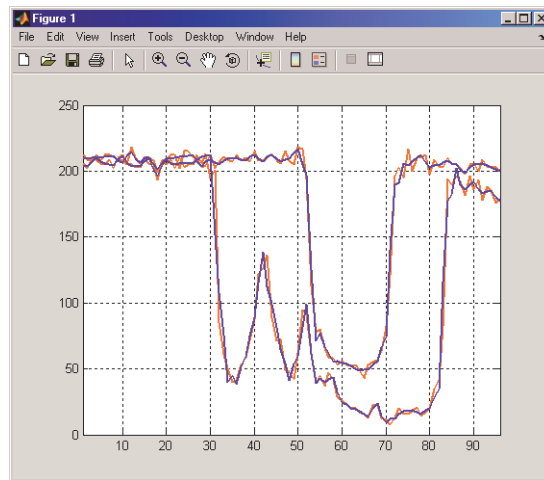
### 5 Perform a wavelet reconstruction of signals and plot some of the new signals.

```
Xbis = mdwtrec(decBIS);
figure;
plot(Xbis(1:5,:),'r'); hold on
plot(Xbis(21:25,:),'b'); plot(Xbis(31:35,:),'g')
grid; set(gca,'Xlim',[1,96])
```



Compare old and new signals by plotting them together.

```
figure; idxSIG = [1 31];
plot(X(idxsIG,:),'r','linewidth',2); hold on
plot(Xbis(idxsIG,:),'b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
```



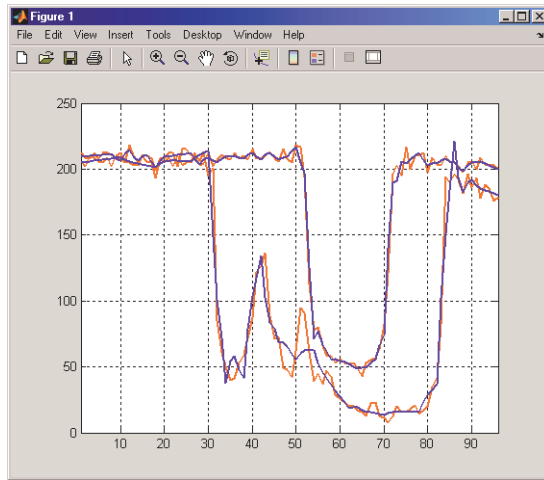
- 6 Set the wavelet coefficients at level 1 and 2 for signals 31 to 35 to the value zero, perform a wavelet reconstruction of signal 31, and compare some of the old and new signals.

```

decter = chgwdccfs(dec, 'cd', 0, 1:2, 31:35);
Y = mdwtrec(decter, 'a', 0, 31);
figure;
plot(X([1 31],:),'r','linewidth',2); hold on
plot([Xbis(1,:); Y]','b','linewidth',2);
grid; set(gca, 'Xlim', [1, 96])

```





- 7** Compute the energy of signals and the percentage of energy for wavelet components.

```
[E,PEC,PECFS] = wdecenergy(dec);
```

Energy of signals 1 and 31:

```
Ener_1_31 = E([1 31])
```

```
Ener_1_31 =
```

```
1.0e+006 *
```

```
3.7534
```

```
2.2411
```

- 8** Compute the percentage of energy for wavelet components of signals 1 and 31.

```
PEC_1_31 = PEC([1 31],:)
```

```
PEC_1_31 =
```

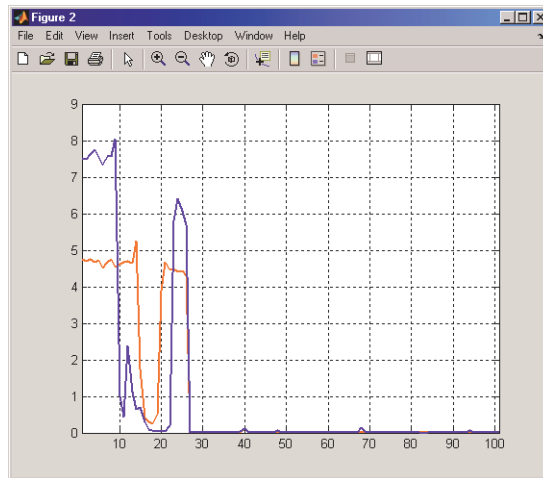
```
99.7760    0.1718    0.0522
```

```
99.3850    0.2926    0.3225
```

The first column shows the percentage of energy for approximations at level 2. Columns 2 and 3 show the percentage of energy for details at level 2 and 1, respectively.

- 9 Display the percentage of energy for wavelet coefficients of signals 1 and 31. As we can see in the dec structure, there are 26 coefficients for the approximation and the detail at level 2, and 49 coefficients for the detail at level 1.

```
PECFS_1 = PECFS(1,:); PECFS_31 = PECFS(31,:);
figure;
plot(PECFS_1,'r','linewidth',2); hold on
plot(PECFS_31,'b','linewidth',2);
grid; set(gca,'Xlim',[1,size(PECFS,2)])
```

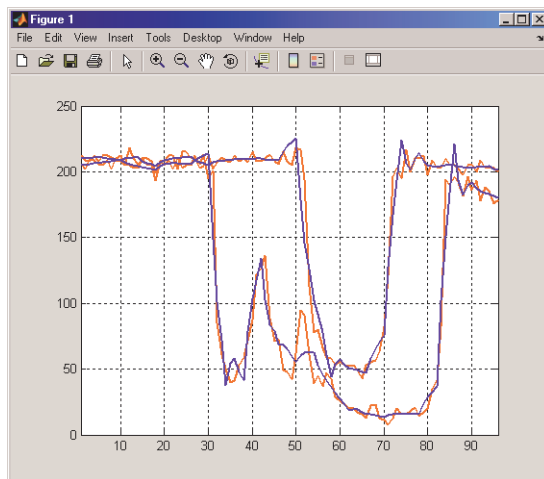


- 10 Compress the signals to obtain a percentage of zeros near 95% for the wavelet coefficients.

```
[XC,decCMP,THRESH] = mswcmp('cmp',dec,'NO_perf',95);
[Ecmp,PECcmp,PECFScmp] = wdecenergy(decCMP);
```

Plot the original signals 1 and 31, and the corresponding compressed signals.

```
figure;
plot(X([1 31],:),'r','linewidth',2); hold on
plot(XC([1 31],:),'b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
```



Compute thresholds, percentage of energy preserved and percentage of zeros associated with the L2\_perf method preserving at least 95% of energy.

```
[THR_VAL,L2_Perf,N0_Perf] = mswcmptp(dec,'L2_perf',95);
idxSIG = [1,31];
```

```
Thr = THR_VAL(idxSIG)
```

```
Thr =
    256.1914
    158.6085
```

```
L2per = L2_Perf(idxSIG)
```

```
L2per =
    96.5488
    94.7197
```

```

N0per = N0_Perf(idxSIG)
N0per =
    79.2079
    86.1386

```

Compress the signals to obtain a percentage of zeros near 60% for the wavelet coefficients.

```
[XC,decCMP,THRESH] = mswcmp('cmp',dec,'N0_perf',60);
```

XC signals are the compressed versions of the original signals in the row direction.

Compress the XC signals in the column direction

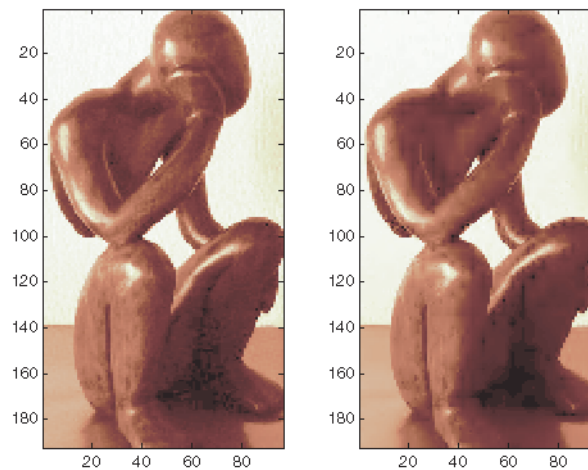
```
XX = mswcmp('cmpsig','c',XC,'db2',2,'N0_perf',60);
```

Plot original signals X and the compressed signals XX as images.

```

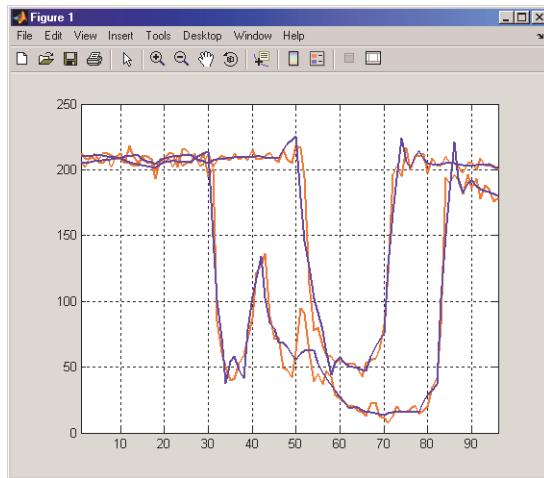
figure;
subplot(1,2,1); image(X)
subplot(1,2,2); image(XX)
colormap(pink(222))

```



### 11 Denoise the signals using the universal threshold:

```
[XD,decDEN,THRESH] = mswwden('den',dec,'sqrtwolog','sln');
figure;
plot(X([1 31],:),'r','linewidth',2); hold on
plot(XD([1 31],:),'b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
```



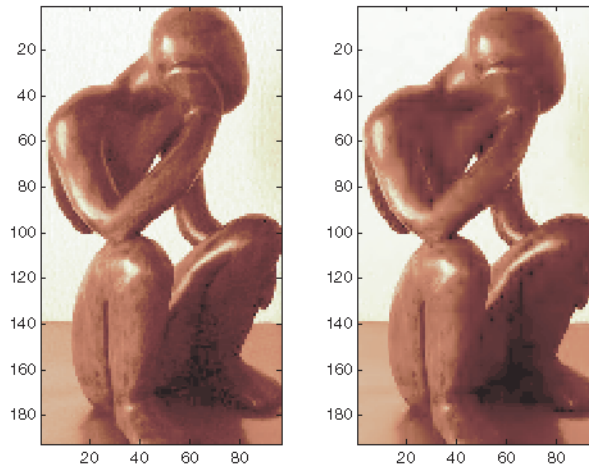
XD signals are the denoised versions of the original signals in the row direction.

Denoise the XD signals in column direction

```
XX = mswwden('densig','c',XD,'db2',2,'sqrtwolog','sln');
```

Plot original signals X and the denoised signals XX as images.

```
figure;
subplot(1,2,1); image(X)
subplot(1,2,2); image(XX)
colormap(pink(222))
```



## One-Dimensional Multisignal Analysis Using the Graphical Interface

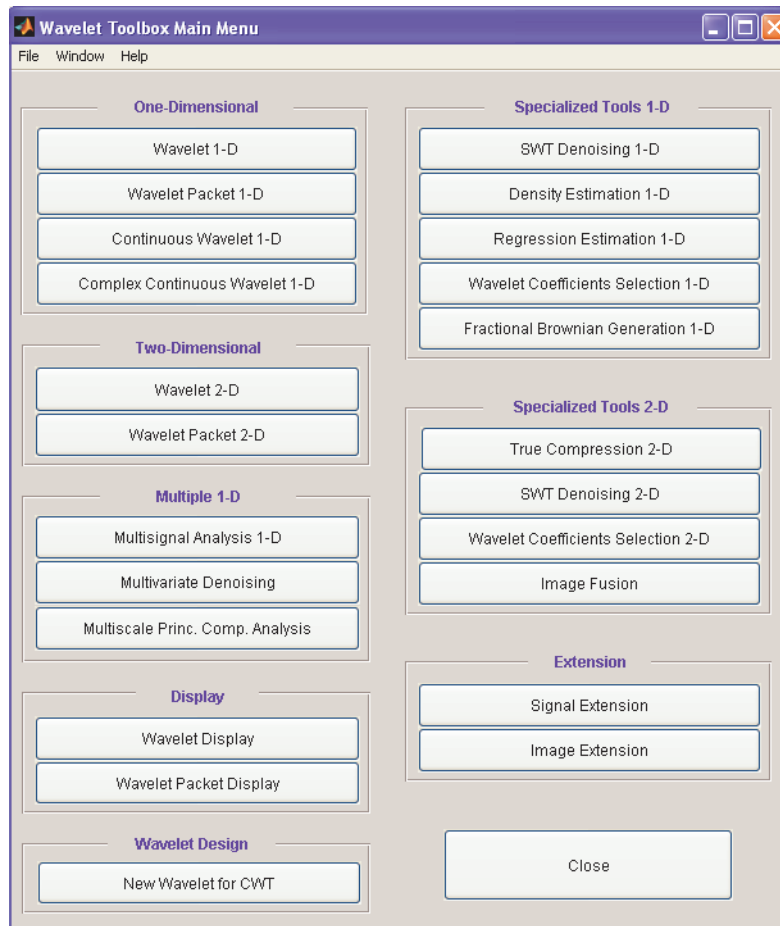
In this section, we explore the same signal as in the previous section, but use the graphical interface tools to analyze it.

- 1 Start the Wavelet 1-D Multisignal Analysis Tool.

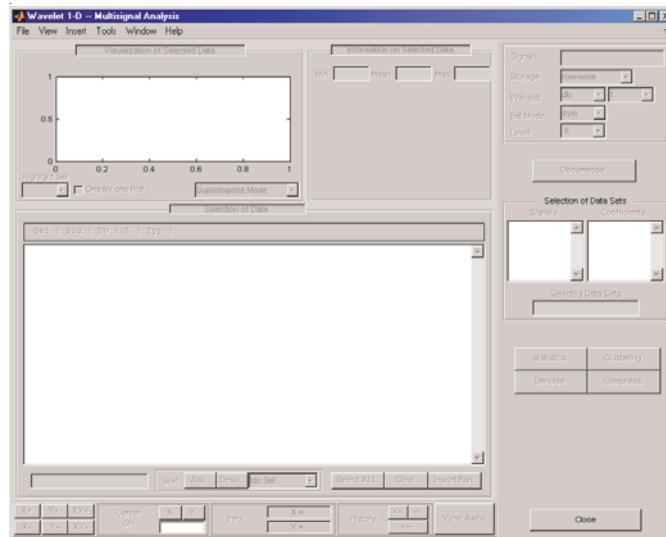
From the MATLAB prompt, type:

```
wavemenu
```

The **Wavelet Toolbox Main Menu** appears.



Click **Multisignal Analysis 1-D** to open the Wavelet 1-D Multisignal Analysis tool.



The tool is divided into five panes. Two of them are the same as in all Wavelet Toolbox GUIs: the Command Frame on the right side of the figure and the Dynamic Visualization tool at the bottom. The Command Frame contains a special component found in all multisignal tools: the **Selection of Data Sets** pane which is used to manage two lists.

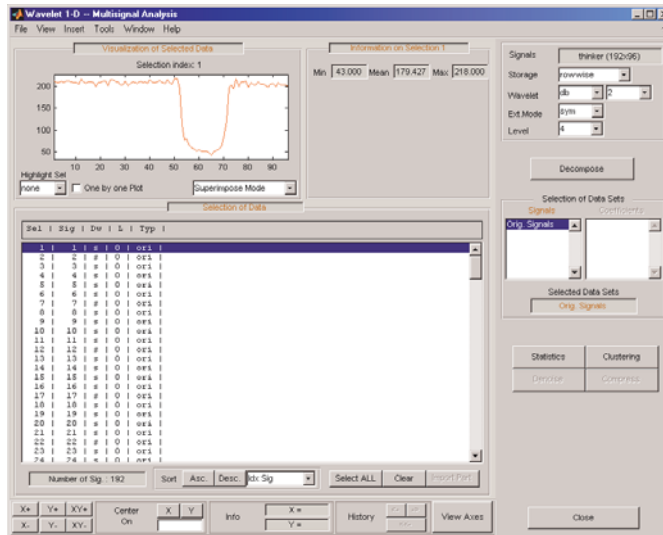
The three new panes are the **Visualization of Selected Data** pane, the **Information on Selected Data** pane, and the **Selection of Data** pane.

## 2 Load the signals.

From the **File** menu, select **Load > Signals**. When the Load Signal dialog box appears, select the demo MAT-file `thinner.mat` from the MATLAB directory `toolbox/wavelet/wmultisig1d` and click **OK**.

The data matrix loads in the Wavelet 1-D Multisignal Analysis tool, and the first signal appears.





The **Selection of Data** pane contains a list of selectable signals. At the beginning, only the originally loaded signals are available. You can generate and add new signals to the list by decomposing, compressing, or denoising original signals.

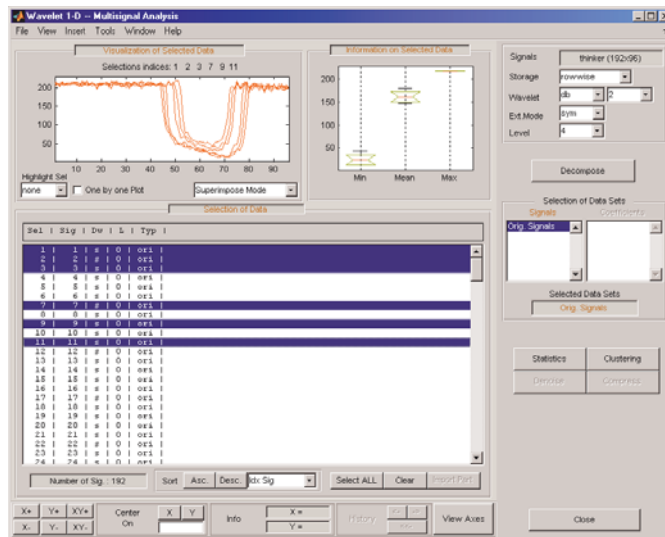
Each row of the list displays the index of selectable signal (Idx Sel), the index of original signal (Idx Sig) and three wavelet transform attributes describing the process used to obtain the selectable signal from the original one.

### 3 View the signals and signal information.

With signal 1 highlighted, Shift-click the mouse on signal 3 to select signals 1, 2, and 3.

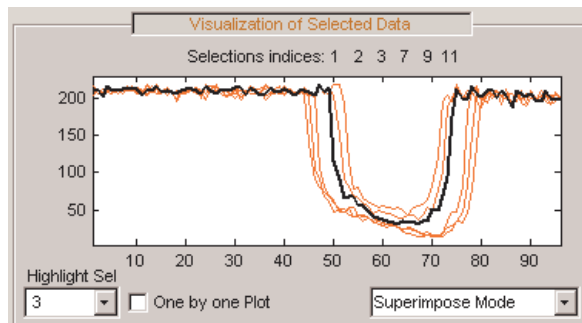
Ctrl-click the mouse on signals 7, 9, and 11. (The **Select ALL** button at the bottom of the **Selection of Data** pane selects all signals and the **Clear** button deselects all signals.)

The selected signals (1, 2, 3, 7, 9 and 11) appear in the **Visualization of Selected Data** pane. The **Information on Selected Data** pane contains the box plots of the minimums, the means, and the maximums of these signals.



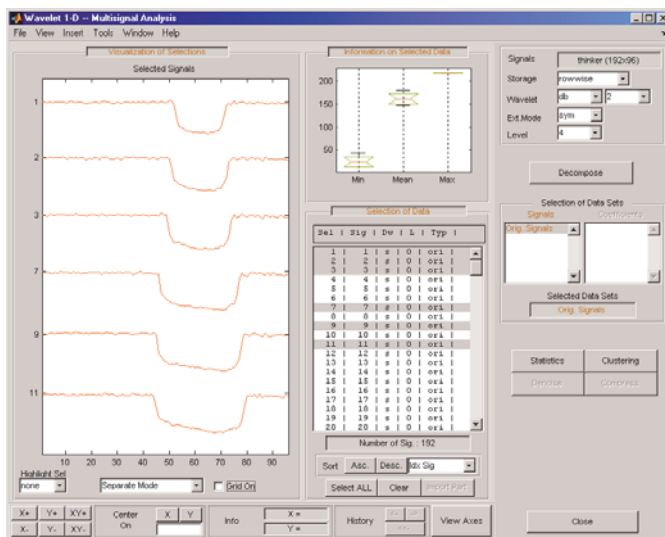
#### 4 Highlight a signal.

Using the **Highlight Sel** button in the lower-left corner of the **Visualization of Selected Data** pane, select signal 3.



#### 5 Select Different Views.

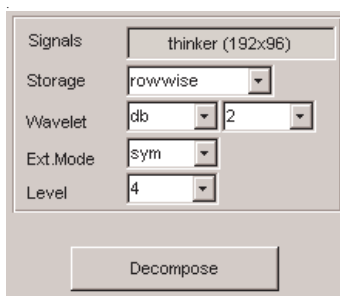
In the **Visualization of Selected Data** pane, change the view mode using the pop-up in the lower-right corner. Choose **Separate Mode**. The selected signals appear.



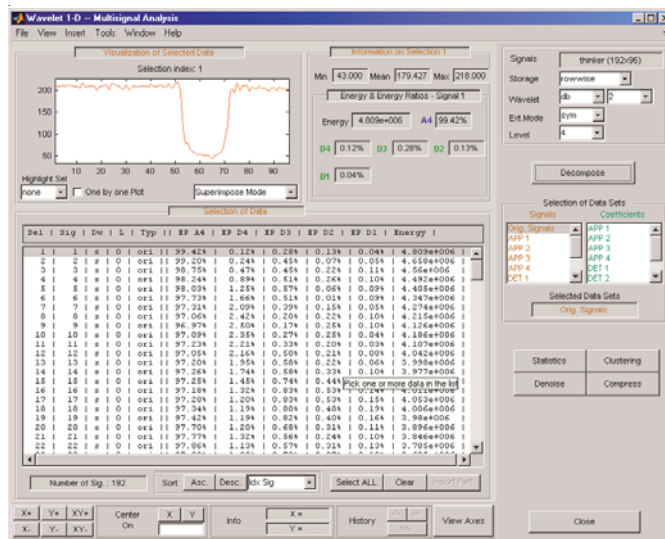
### 6 Decompose a multisignal.

Perform an analysis at level 4 using the db2 wavelet and the same file used in the command line section: `thinker.mat`.

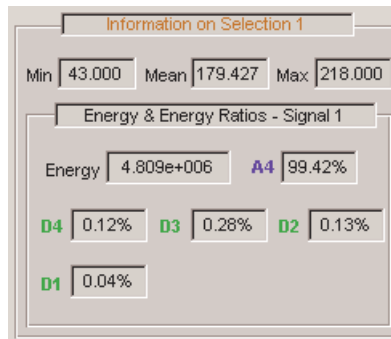
In the upper right portion of the Wavelet 1-D Multisignal Analysis tool, select db2 and level 4 in the Wavelet fields.



Click **Decompose**. After a pause for computation, all the original signals are decomposed and signal 1 is automatically selected

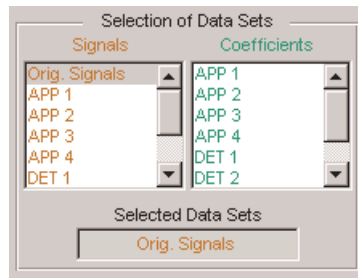


In the **Selection of Data** pane, new information is added for each original signal — the percentage of energy of the wavelet components (**D1**, ..., **D4** and **A4**) and the total energy. The **Information on Selected Data** pane contains information on the single selected signal: **Min**, **Mean**, **Max** and the energy distribution of the signal.



Since the original signals are decomposed, new objects appear and the **Selection of Data Sets** pane in the Command Frame updates.

The **Selection of Data Sets** pane defines the available signals that are now selectable from the **Selection of Data** pane.

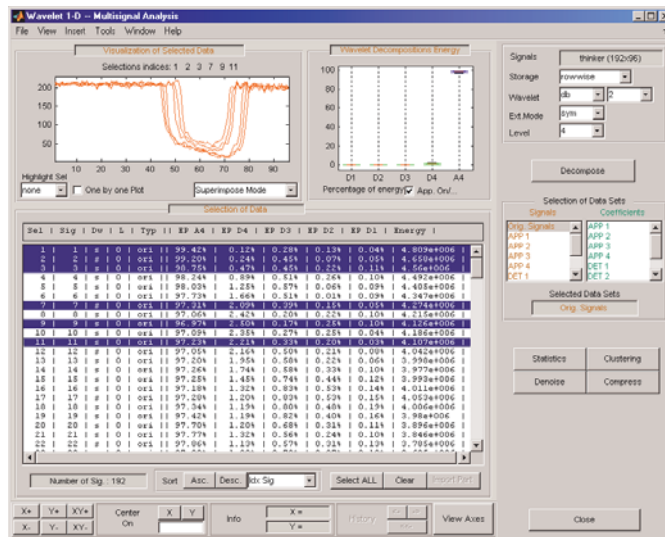


The list on the left allows you to select sets of signals and the right list allows you to select sets of corresponding coefficients: original signals (Orig. Signals), approximations (APP 1,...) and details from levels 1 to 4 (DET 1,...).

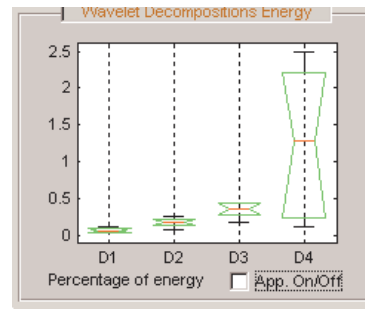
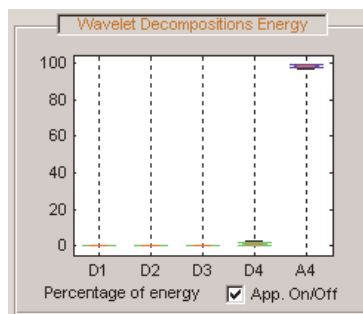
In the list on the right, the coefficients vectors can be of different lengths, but only components of the same length can be selected together.

After a decomposition the original signals (Orig. Signals) data set appears automatically selected.

Select signals 1, 2, 3, 7, 9 and 11.

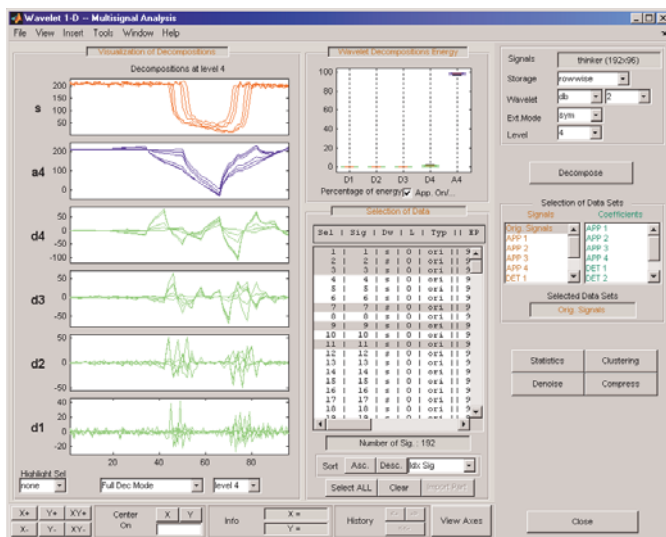


The energy of selected signals is primarily concentrated in the approximation A4, so the box plot is crushed (see figure below on the left). Deselect **App. On/Off** to see a better representation of details energy (see figure below on the right).



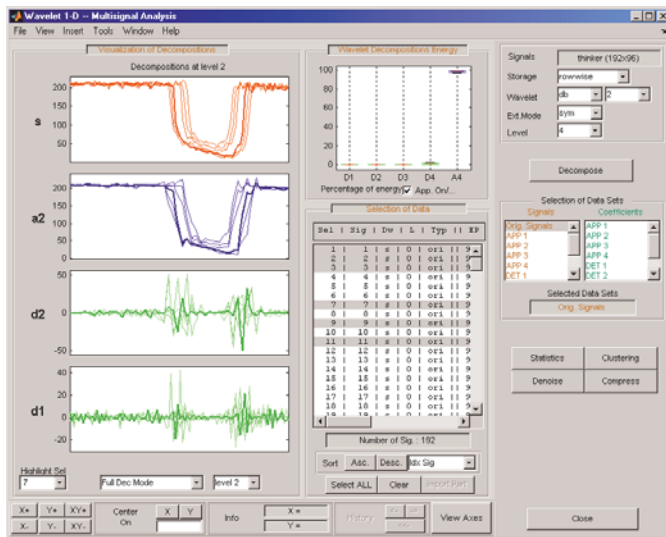
**7** Display multisignal decompositions.

In the **Visualization of Selected Data** pane, change the view mode using the pop-up below the plots and select Full Dec Mode. The decompositions of the selected signals display.



Change the **Level** to 2.

Select the signal 7 in **Highlight Sel**.

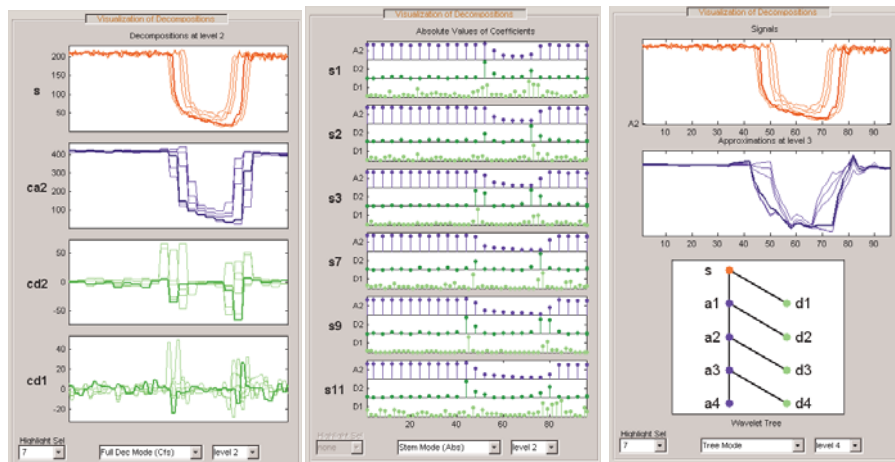


## 8 Change the visualization modes.

Using the second pop-up from the left at the bottom of the pane, select **Full Dec Mode (Cfs)**. The coefficients of the decompositions of the selected signals display. At level  $k$ , coefficients are duplicated  $2^k$  times.

Change the view mode to **Stem Mode (Abs)**, and then, change to **Tree Mode**. The wavelet tree corresponding to the decompositions of the selected signals displays.

Select the level 4 and click the node **a3**. Then highlight signal 7.

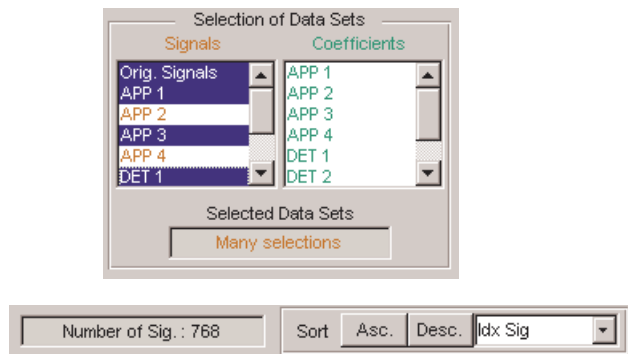


## 9 Select Different Wavelet Components.

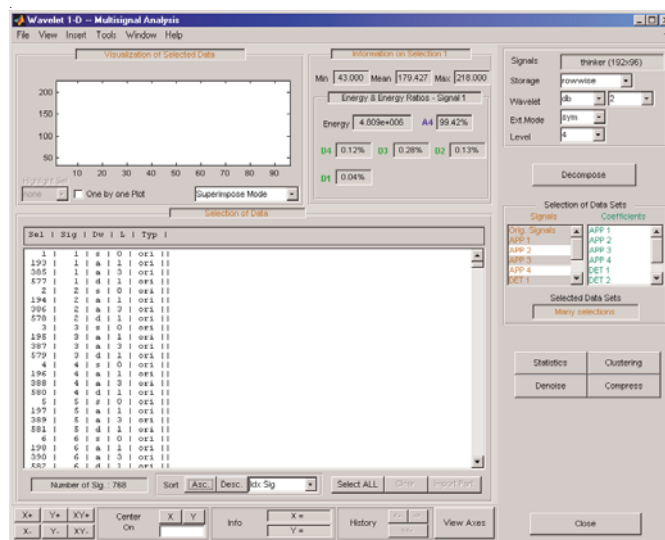
Ctrl-click **Orig. Signals**, **APP 1**, **APP 3** and **DET 1** to select these four sets of signals from the list on the left in the **Selection of Data Sets** pane.

The total number of selected data (Number of Sig.) appears in the **Selection of Data Sets** pane: four sets of 192 signals each is a total of 768 signals.



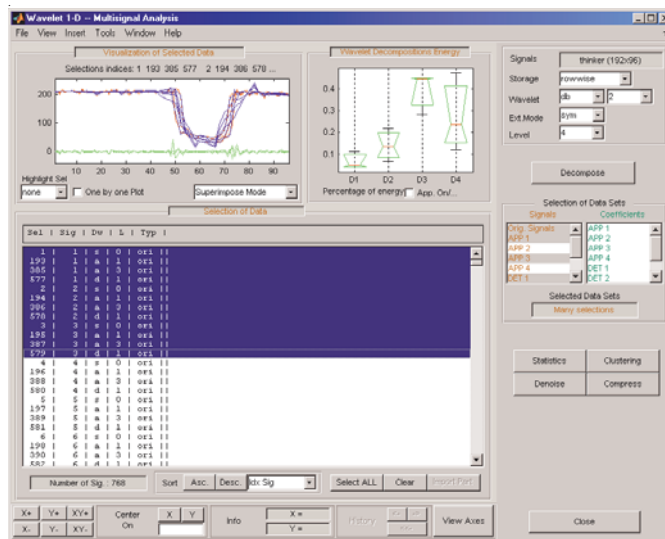


Click the **Asc.** button in the **Sort** pane. The selected data are sorted in ascending order with respect to the **Idx Sig** parameter

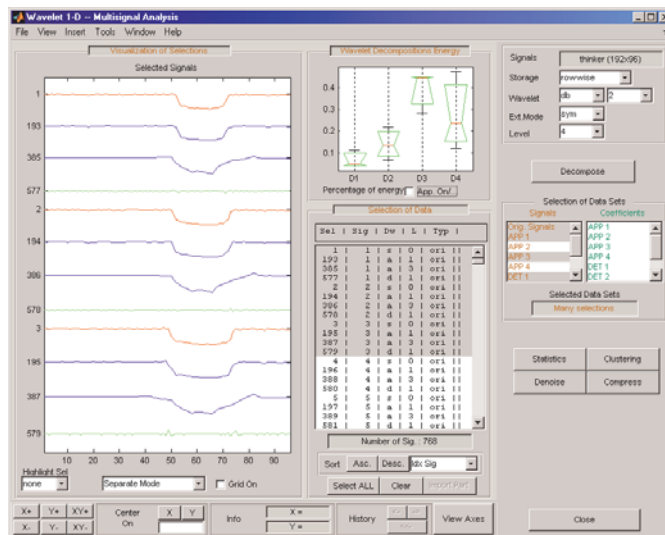


Note that DWT attributes of each selectable signal have been updated where **a** stands for approximation, **d** for detail and **s** for signal.

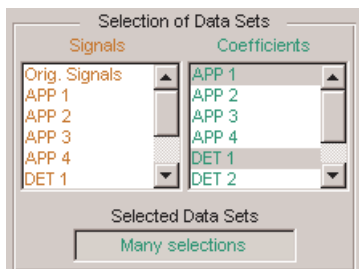
Click the **Idx Se1 1** signal and then shift-click the **Idx Se1 579** signal.



Choose Separate Mode.



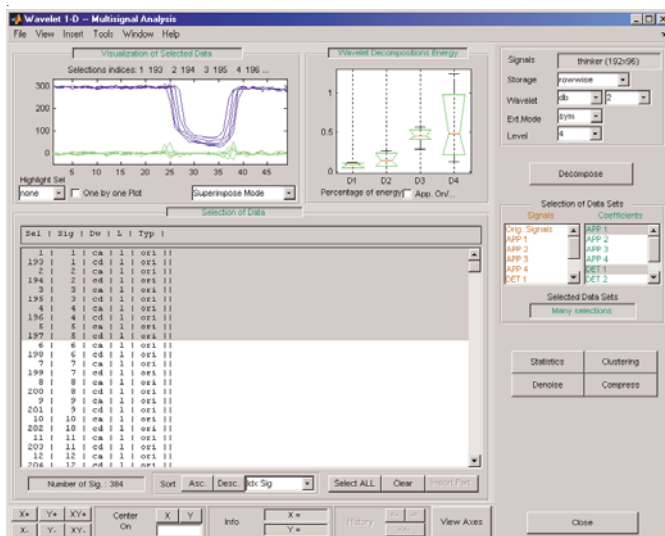
Ctrl-click to select two sets of signals from the right-most list of the **Selection of Data Sets** pane: APP 1 and DET 1.



Note that in this list of coefficients sets, the selected vectors must be of same length, which means that you must select components of the same level.

Click the **Asc.** button in the Sort pane. The selected data are sorted in ascending order with respect to Idx Sig parameter.

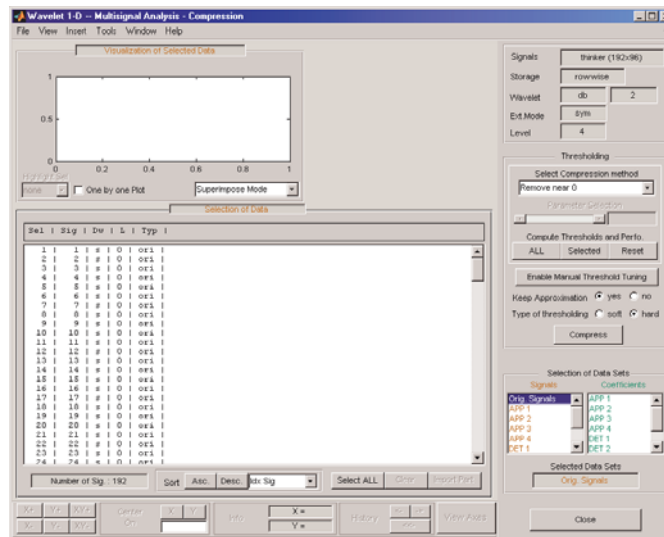
Select the ten first signals.



## 10 Compress a multisignal.

The graphical interface tools feature a compression option with automatic or manual thresholding.

Click **Compress**, located in the lower-right side of the window. This displays the Compression window.



**Note** The tool always compresses all the original signals when you click the **Compress** button.

Before compressing, choose the particular strategy for computing the thresholds. Select the adapted parameters in the **Select Compression Method** frame. Then, apply this strategy to compute the thresholds according to the current method, either to the current selected signals by clicking the **Selected** button, or to all signals by clicking the **ALL** button. For this example, accept the defaults and click the **ALL** button.

Selection of Data

Sig	ThrD1	ThrD2	ThrD3	ThrD4	En. Rat.	NbZ Rat.
1	3.871	3.871	3.871	3.871	100.00%	50.48%
2	4.631	4.631	4.631	4.631	99.99%	50.48%
3	3.831	3.831	3.831	3.831	100.00%	50.48%
4	4.166	4.166	4.166	4.166	99.99%	50.48%
5	4.455	4.455	4.455	4.455	99.99%	50.48%
6	3.793	3.793	3.793	3.793	100.00%	50.48%
7	3.630	3.630	3.630	3.630	100.00%	50.48%
8	4.407	4.407	4.407	4.407	99.99%	50.48%
9	3.536	3.536	3.536	3.536	100.00%	50.48%
10	3.523	3.523	3.523	3.523	100.00%	50.48%
11	3.829	3.829	3.829	3.829	99.99%	50.48%
12	3.251	3.251	3.251	3.251	100.00%	50.48%
13	3.820	3.820	3.820	3.820	99.99%	50.48%
14	4.028	4.028	4.028	4.028	99.99%	50.48%
15	3.635	3.635	3.635	3.635	99.99%	50.48%
16	4.406	4.406	4.406	4.406	99.99%	50.48%
17	4.108	4.108	4.108	4.108	100.00%	50.48%
18	4.518	4.518	4.518	4.518	99.99%	50.48%
19	4.287	4.287	4.287	4.287	99.99%	50.48%
20	3.730	3.730	3.730	3.730	99.99%	50.48%
21	4.770	4.770	4.770	4.770	99.99%	50.48%
22	4.513	4.513	4.513	4.513	99.99%	50.48%
23	4.915	4.915	4.915	4.915	99.99%	50.48%
24	4.311	4.311	4.311	4.311	99.99%	50.48%

Number of Sig. : 192    Sort    Asc.    Desc.    Idx Sig    Select ALL    Clear    Import Part

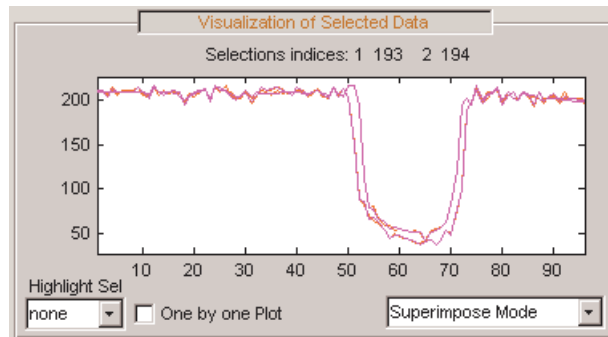
The thresholds for each level (ThrD1, ThrD2, ThrD3, ThrD4), the energy ratio (En. Rat.) and the sparsity ratio (NbZ Rat.) are displayed in the **Selection of Data** pane.

Click the **Compress** button at the bottom of the **Thresholding** pane. Now you can select new data sets: compressed Signals, the corresponding approximations, details and coefficients.

Press the **Ctrl** key and click the **Compressed** item in the left list of the **Selection of Data Sets** pane. The original signals and their compressed versions are selected (2 x 192 = 384 signals).

Click the **Asc.** button at the bottom of the **Selection of Data** pane to sort the signals using Idx Sig number.

With the mouse, select the first four signals. They correspond to the original signals 1, 2 and the corresponding compressed signals 193, 194.

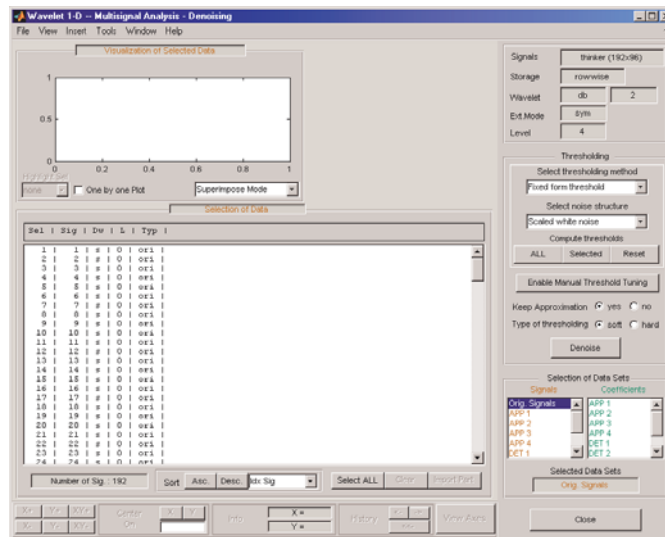


Click the **Close** button to close the Compression window.

### 11 Denoise a multisignal.

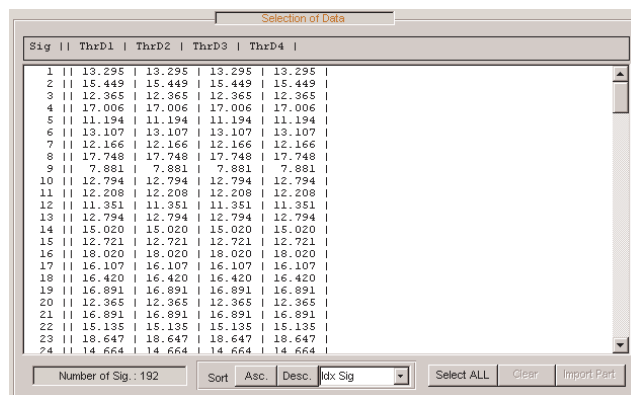
The graphical user interface offers a denoising option with either a predefined thresholding strategy or a manual thresholding method. Using this tool makes very easy to remove noise from many signals in one step.

Display the Denoising window by clicking the **Denoise** button located in the bottom part of the Command Frame on the right of the window.



A number of options are available for fine-tuning the denoising algorithm. For this example, accept the defaults: soft type of thresholding, Fixed form threshold method, and Scaled white noise as noise structure.

Click the **ALL** button in the Thresholding pane. The threshold for each level (ThrD1, ..., ThrD4) computes and displays in the **Selection of Data** pane.

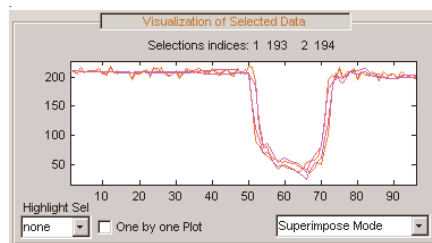


Then click the **Denoise** button at the bottom of the **Thresholding** pane.

Ctrl-click the **Denoised** item in the list on the left of the **Selection of Data Sets** pane. The original signals and the corresponding denoised ones are selected ( $2 \times 192 = 384$  signals).

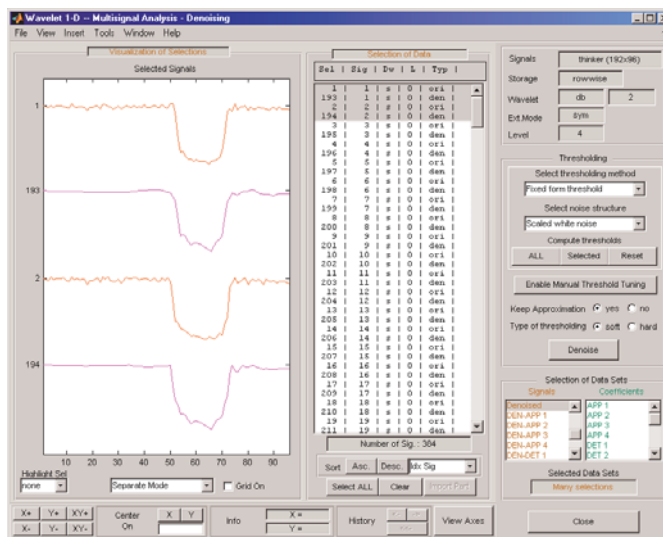
Click the **Asc.** button at the bottom of the **Selection of Data** pane to sort the signals according to the Idx Sig parameter.

With the mouse, select the first four signals. They correspond to the original signals 1, 2 and the corresponding denoised signals 193, 194



Choose Separate Mode.



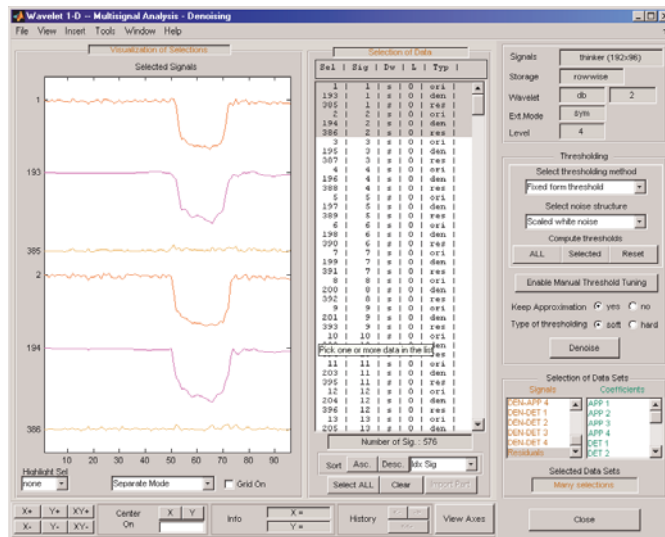


**12** To view residuals, Ctrl-click the **Orig.** Signal, the **Denoised** and the **Residuals** items in the list on the left of the **Selection of Data Sets** pane. Original, denoised and residual signals are selected (3 x 192 = 576 signals).

Click the **Asc.** button at the bottom of the **Selection of Data** pane to sort the signals using the **Idx Sig** parameter.

With the mouse, select the first six signals. They correspond to the original signals 1, 2, the corresponding denoised signals 193, 194 and the residuals 385, 386.

Then, choose **Separate Mode**.

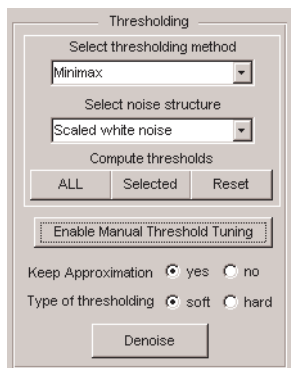


- 13 Click **Close** to close the denoising tool. Then, click the **Yes** button to update the synthesized signals.

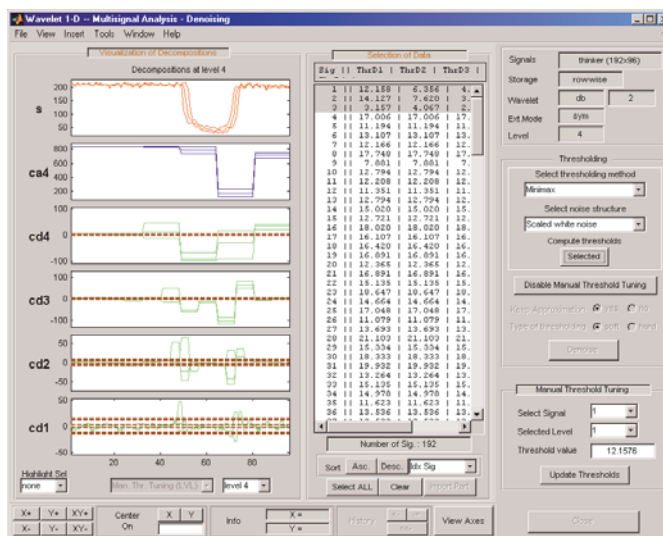


## Manual Threshold Tuning

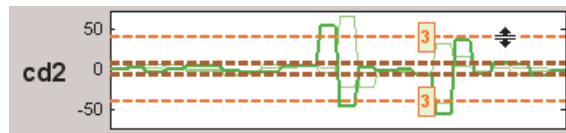
- 1 Choose a method, select one or several signals in the **Selection of Data** pane using the mouse and keys. Then click the **Selected** button. You can select another group of signals using the same method. Press the **Denoise** button to denoise the selected signal(s).



You can also use manual threshold tuning. Click the **Enable Manual Thresholding Tuning** button.



The horizontal lines in the wavelet coefficient axes (cd1, ..., cd4) can be dragged using the mouse. This may be done individually, by group or all together depending on the values in the **Select Signal** and **Selected Level** fields in the **Manual Threshold Tuning** pane.



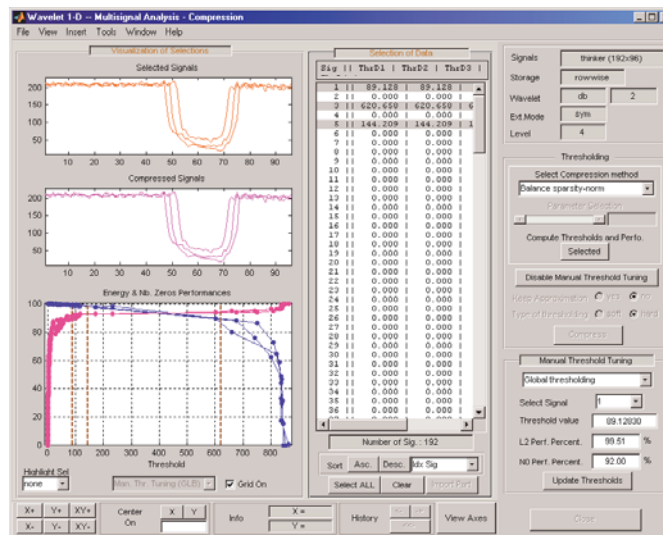
**Manual Threshold Tuning**

Select Signal 3

Selected Level 2

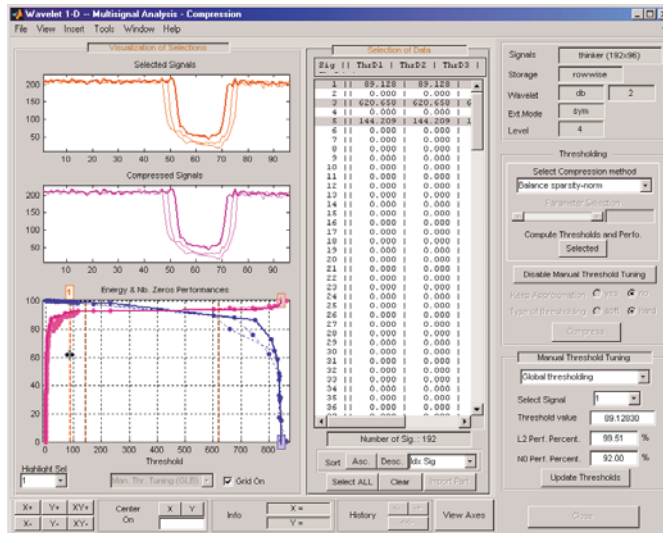
Threshold value 42.51011

- 2 In the Wavelet 1-D Multisignal Analysis Compression tool, you can use two methods for threshold tuning: the **By level thresholding** method which is used in the Wavelet 1-D Multisignal Analysis Denoising tool, and the **Global thresholding** method.



You can drag the vertical lines in the **Energy and Nb. Zeros Performances** axes using the mouse. This can be done individually or all together.

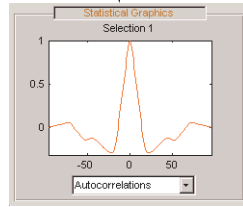
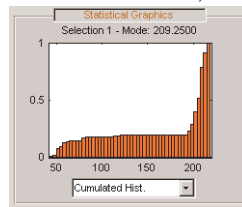
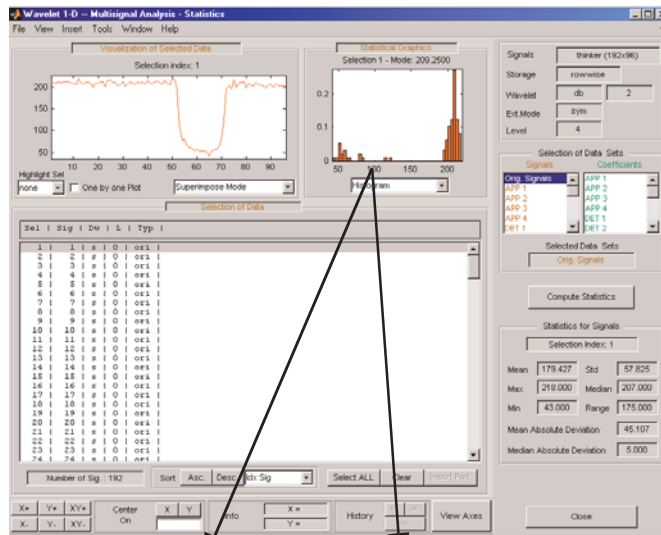
depending on the values of **Select Signal** in the **Manual Threshold Tuning** pane.



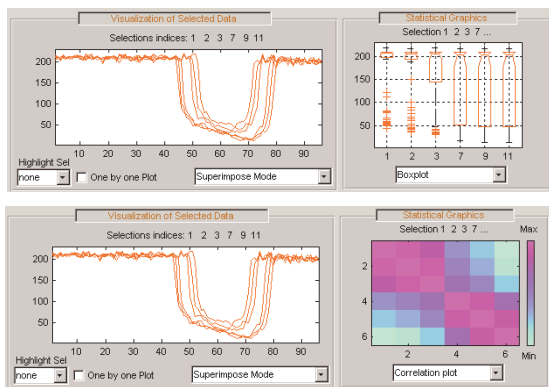
The threshold value, L2 performance, and number of zeros performance are updated in the corresponding edit buttons in the **Manual Threshold Tuning** pane.

### Statistics on Signals

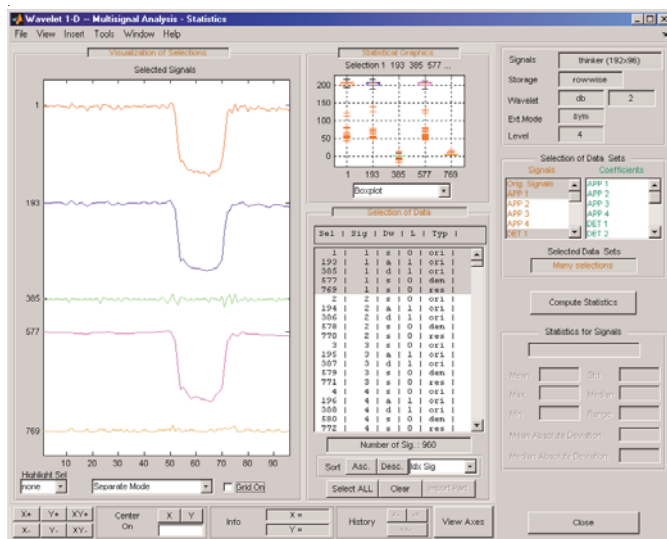
- 1 You can display various statistical parameters related to the signals and their components. From the Wavelet 1-D Multisignal Analysis tool, click the **Statistics** button. Then select the signal 1 in the **Selection of Data Sets** pane.



Select the signals 1, 2, 3, 7, 9 and 11 in the **Selection of Data** pane, and display the corresponding boxplots and correlation plots.



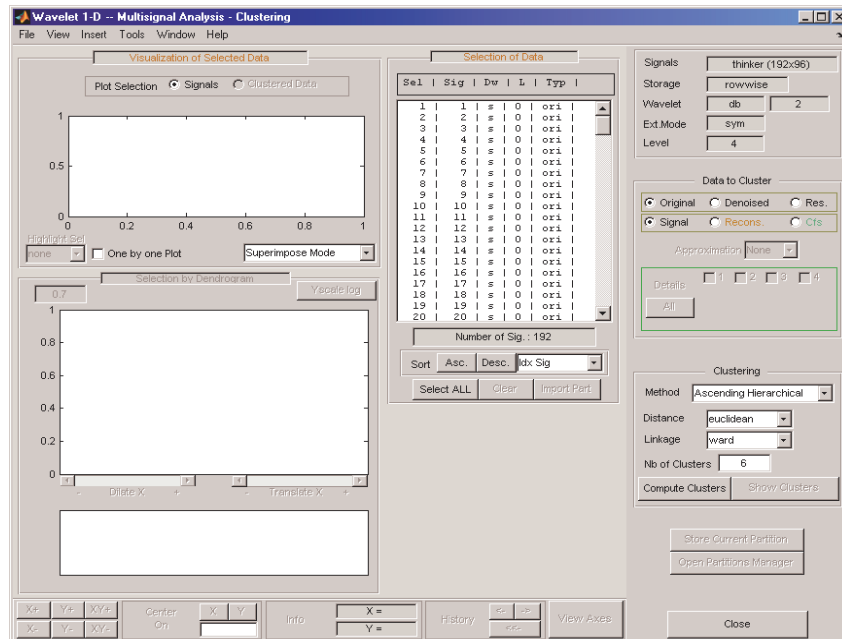
2 To display statistics on many wavelet components, in the **Selection Data Sets** pane, in the left column, select **Orig. Signals, APP 1, DET 1, Denoised** and **Residuals** signals. Then choose **Separate Mode**, and click the **Asc.** button in the **Sort** pane. The selected data are sorted in ascending order with respect to **Idx Sig** parameter. In the **Selection of Data** pane, select data related to signal 1.



## Clustering Signals

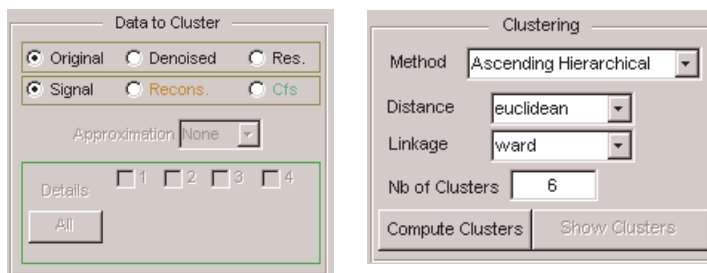
**Note** To use clustering, you must have Statistics Toolbox™ software installed.

- 1 Click the **Clustering** button located in the Command Frame, which is in the lower right of the Wavelet 1-D Multisignal Analysis window to open the Clustering tool.



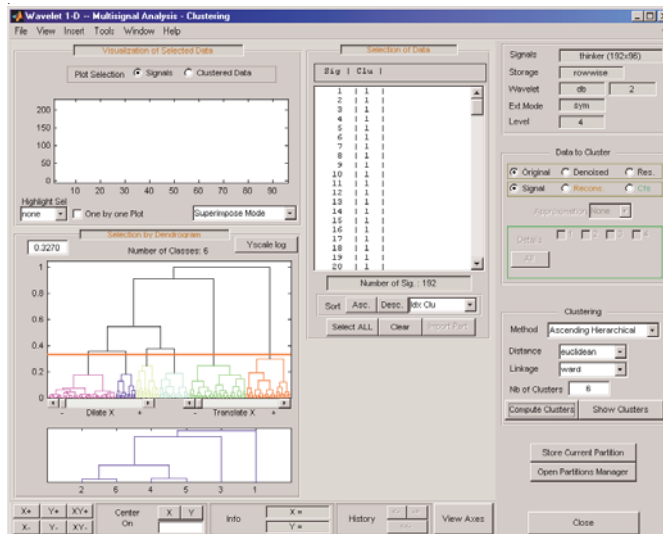
You can cluster various type of signals and wavelet components: original, denoised or compressed, residuals, and approximations or details (reconstructed or coefficients). Similarly, there are several methods for constructing partitions of data.





Use the default parameters (Original and Signal in Data to Cluster, and in Ascending Hierarchical, euclidean, ward, and 6 in Clustering) and click the **Compute Clusters** button.

A full dendrogram and a restricted dendrogram display in the **Selection by Dendrogram** pane. For each signal, the cluster number displays in the **Selection of Data** pane.

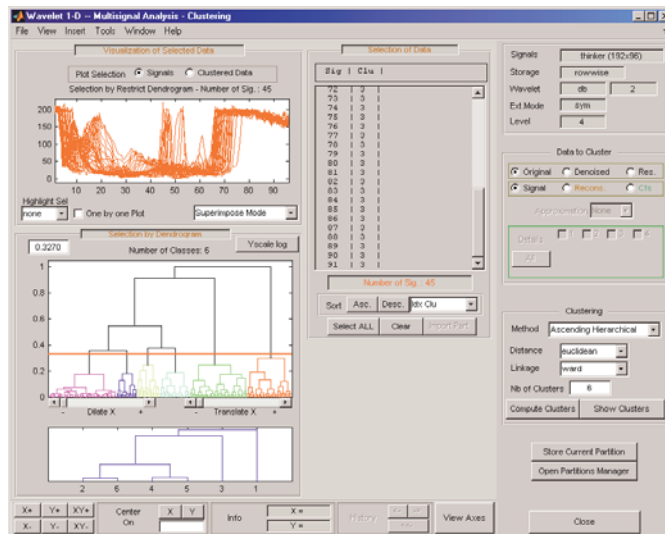


**2** Select one cluster, several clusters, or a part of a cluster.

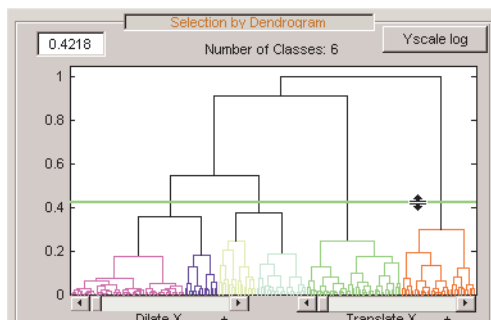
Click the `xticklabel 3` at the bottom of the restricted dendrogram. The links of the third cluster blink in the full dendrogram and the 24 signals of

this class display in the **Visualization of Selected Data** pane. You can see their numbers in the **Selection of Data** pane.

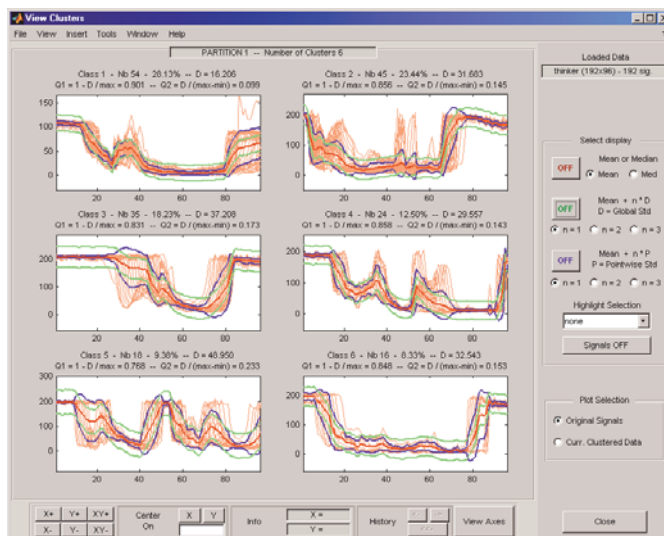
Clicking the line in the restricted or in the full dendrogram lets you select one cluster, several linked clusters, or a part of a cluster. For a more accurate selection, use the **Dilate X** and the **Translate X** sliders under the full dendrogram. You can also use the **Yscale** button located above the full dendrogram. The corresponding signals display in the **Visualization of Selected Data** pane and in the list of the **Selection of Data** pane.



You can use the horizontal line in the full dendrogram to change the number of clusters. Use the left mouse button to drag the line up or down.

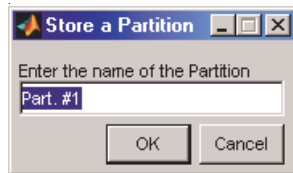


- 3 Use the **Show Clusters** button to examine the clusters of the current partition. You can display the mean (or the median) of each cluster, the global standard deviation and the pointwise standard deviation distance around the mean (or the median). The number of the cluster, the number of elements, the percentage of signals, and two indices of quality display for each cluster.



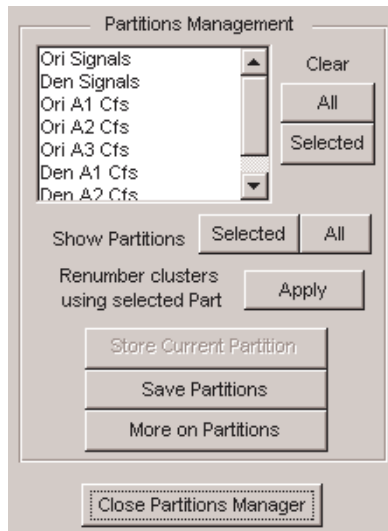
- 4 Click the **Store Current Partition** button below the **Clustering** pane to store the current partition for further comparisons. A default name is

suggested. Note that the 1-D Wavelet Multisignal Analysis tool stores the partitions and they are not saved on the disk.



## Partitions

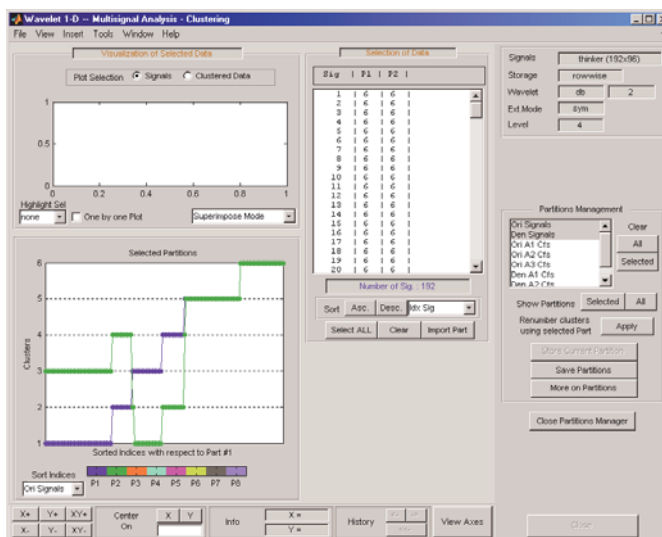
- 1 Build and store several partitions (for example, partitions with signals, denoised signals approximations at level 1, 2 and 3, and denoised signals). Then, click the **Open Partition Manager** button below the **Store Current Partition** button. The **Partitions Management** pane appears. The names of all stored partitions are listed.



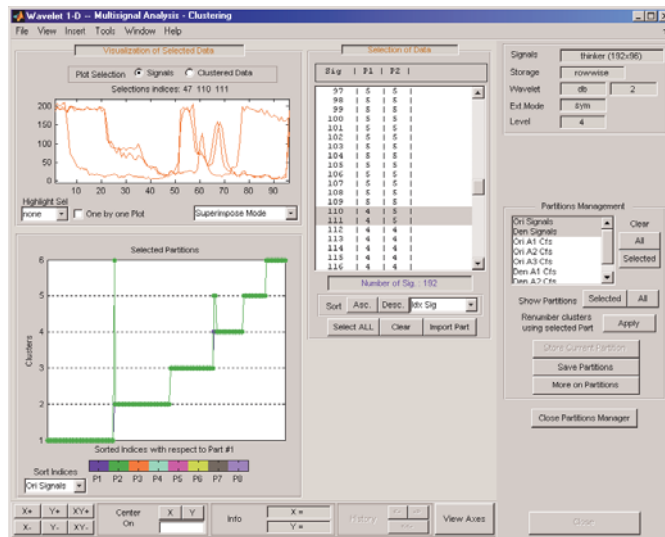
Now, you can show, clear, or save the partitions (individually, selected ones, or all together).

- To display partitions, select the Ori Signals and the Den Signals partitions, and click the **Selected** button next to the **Show Partitions** label.

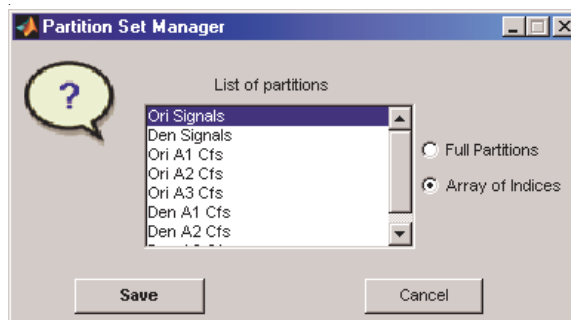
The clusters are almost the same, but it is difficult to see this on the **Selected Partitions** axis, due to the scaling difference. Press the **Apply** button to renumber the clusters (starting from the selected partition as basic numbering) to compare the two partitions.



Only three signals are not classified in the same cluster for the two considered partitions.

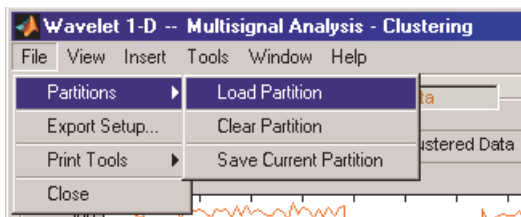
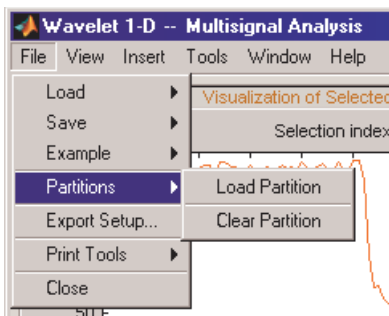


- 3 Select the partitions you want to save and click the **Save Partitions** button below the **Store Current Partition** button in the **Partitions Management** pane.

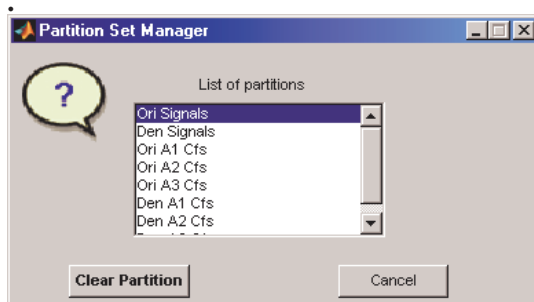


Partitions are saved as an array of integers, where each column corresponds to one partition and contains the indices of clusters. When you choose the Full Partitions option, an array object (wpartobj) is saved.

- 4 To load or clear stored partitions use **File > Partitions** in the Wavelet 1-D Multisignal Analysis tool. (**File > Partitions** is also available in the Wavelet 1-D Multisignal Analysis Clustering tool and you can also save the current partition.)



To clear one or more stored partitions, select **File > Partitions > Clear Partition**.



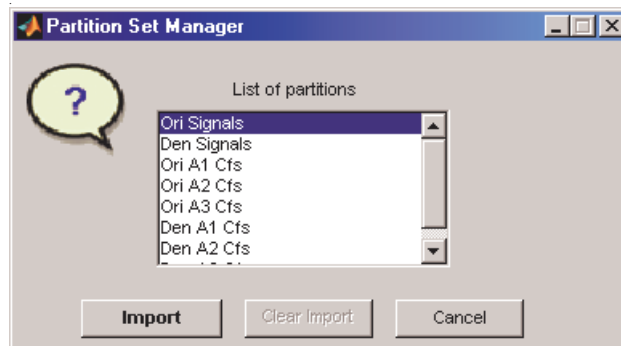
Select **File > Partitions > Load Partition** to load one or several partitions from the disk. The loaded partitions are stored in Wavelet 1-D Multisignal Analysis tool with any previously stored partitions. A partition can also be a manually created column vector.

---

**Note** The number of signals in loaded partitions must be equal to the number of signals in the Wavelet 1-D Multisignal Analysis tool. A warning appears if this condition is not true.

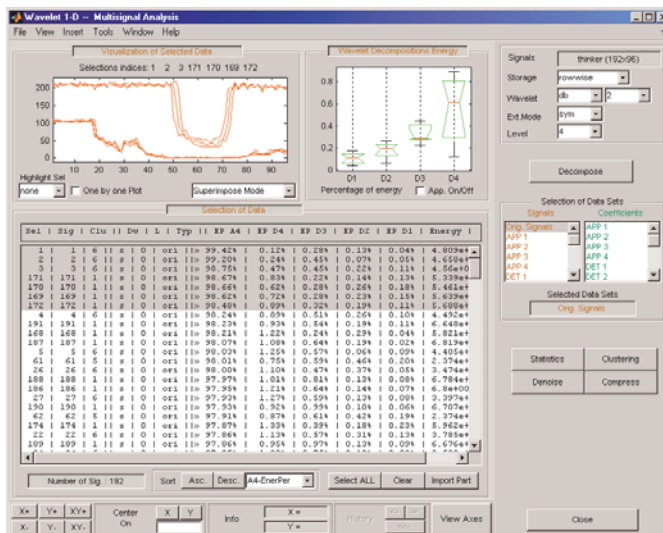
---

- 5 In each subcomponent of the Wavelet 1-D Multisignal Analysis tool (main, statistics, denoising, compression, clustering), you can import a stored partition from the list in the **Selection of Data** pane. Click the **Import Part** button at the bottom of the **Selection of Data** pane, the Partition Set Manager window appears. Select one partition and click the **Import** button.

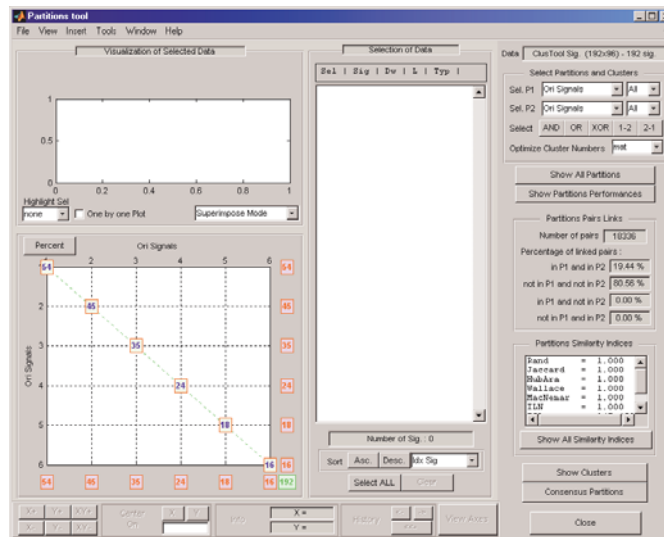




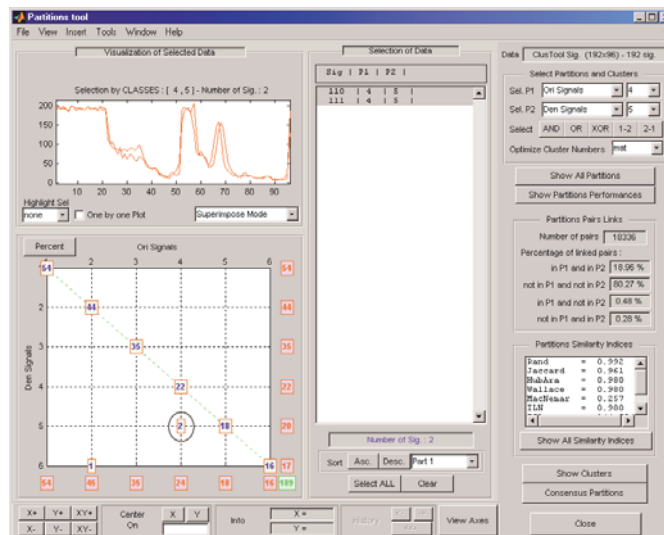
For this example, go back to the main window, import the Ori Signals partition and sort the signals in descending order with respect to **A4** energy percentage.



6 Click the **More on Partitions** button at the bottom of the Partitions Management pane to display the Partition tool.



- 7 Select the Den Signals in **Sel P2** in the upper-right corner of the window. Then, in the lower left axis, click the yellow text containing the value 2 (the coordinates of the corresponding point are (4,5)). The corresponding signals are displayed together with all related information.

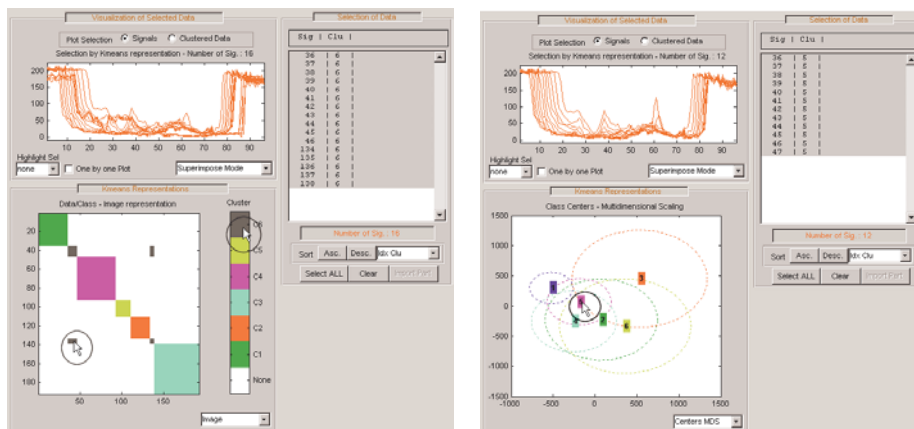


### More on Clustering

Instead of the Ascending Hierarchical Tree clustering method, you can use the K-means method. For this case, the partition cannot be represented by a dendrogram and other representations should be used.

In the image representation (see figure below on the left), you can select a cluster by clicking on the corresponding color on the colorbar. You can also select a cluster or part of a cluster by clicking on the image.

In the center representation (see figure below on the right) you can select a cluster by clicking on the corresponding colored center.

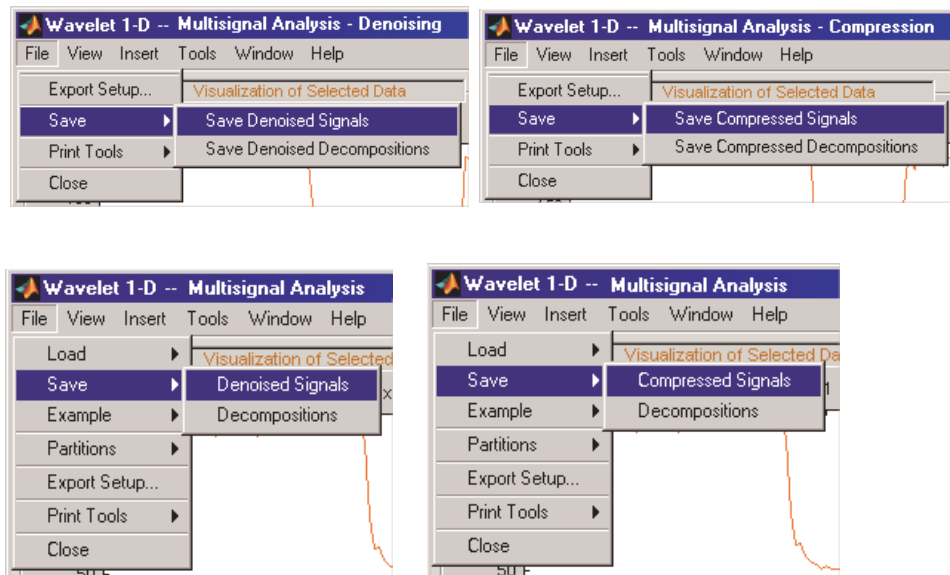


### Importing and Exporting Information from the Graphical Interface

The Wavelet 1-D Multisignal Analysis tool lets you move data to and from disk.

#### Saving Information to Disk

You can save decompositions and denoised or compressed signals (including the corresponding decompositions from Wavelet 1-D Multisignal Analysis tools) to disk. You then can manipulate the data and later import it again into the graphical tools.



## Saving Decompositions

The Wavelet 1-D Multisignal Analysis main tool lets you save the entire set of data from a wavelet analysis to disk. The toolbox creates a MAT-file in the current directory with a name you choose.

- 1 Open the Wavelet 1-D Multisignal Analysis main tool and load the example analysis by selecting **File > Example > Ex 21: Thinker (rows)**.
- 2 Save the data from this analysis, using the menu option **File > Save Decompositions**.

A dialog box appears that lets you specify a directory and filename for storing the decomposition data. For this example, use the name `decORI.mat`.

- 3 Type the name `decORI`.
- 4 After saving the decomposition data to the file `decORI.mat`, load the variables into your workspace:
 

```
load decORI
```

```
whos
```

Name	Size	Bytes	Class
dec	1x1	163306	struct

```
dec
dec =
    dirDec: 'r'
    level: 4
    wname: 'db2'
    dwtFilters: [1x1 struct]
    dwtEXTM: 'sym'
    dwtShift: 0
    dataSize: [192 96]
    ca: [192x8 double]
    cd: {1x4 cell}
```

The field `ca` of the structure `dec` gives the coefficients of approximation at level 4, the field `cd` is a cell array which contains the coefficients of details.

```
size(dec.cd{1})
ans =
    192    49
size(dec.cd{2})
ans =
    192    26
size(dec.cd{3})
ans =
    192    14
size(dec.cd{4})
ans =
    192     8
```

You can change the coefficients using the `chgwdeccfs` function.

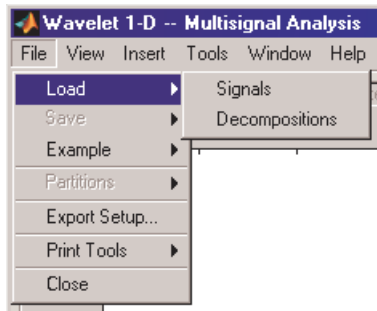
---

**Note** For a complete description of the `dec` structure, see “Loading Decompositions” on page 2-312.

---

## Loading Information into the Wavelet 1-D Multisignal Analysis Tool

You can load signals or decompositions into the graphical interface. The information you load may be previously exported from the graphical interface, and then manipulated in the workspace; or it may be information you initially generated from the command line. In either case, you must observe the strict file formats and data structures used by the Wavelet 1-D Multisignal Analysis tools or errors will occur when you try to load information.



**Loading Signals.** To load a signal you constructed in your MATLAB workspace into the Wavelet 1-D Multisignal Analysis tool, save the signal in a MAT-file (with extension `.mat`).

For example, if you design a signal called `magic128` and want to analyze it in the Wavelet 1-D Multisignal Analysis tool, type

```
save magic128 magic128
```

---

**Note** The workspace variable `magic128` must be a matrix and the number of rows and columns must be greater than 1.

---

```
sizmag = size(magic128)
```

```
sizmag =
    128    128
```

To load this signal into the Wavelet 1-D Multisignal Analysis tool, use the **File > Load Signal** menu item. A dialog box appears in which you select the appropriate MAT-file to be loaded.

---

**Note** When you load a matrix of signals from the disk, the name of 2-D variables are inspected in the following order: `x`, `X`, `sigDATA`, and `signals`. Then, the 2-D variables encountered in the file are inspected in alphabetical order.

---

**Loading Decompositions.** To load decompositions that you constructed in the MATLAB workspace into the Wavelet 1-D Multisignal Analysis tool, save the signal in a MAT-file (with extension `mat`).

For instance, if you design a signal called `magic128` and want to analyze it in the Wavelet 1-D Multisignal Analysis tool, the structure `dec` must have the following fields:

<code>'dirDec'</code>	Direction indicator with <code>'r'</code> for row or <code>'c'</code> for column
<code>'level'</code>	Level of DWT decomposition
<code>'wname'</code>	Wavelet name
<code>'dwtFilters'</code>	Structure with four fields: <code>LoD</code> , <code>HiD</code> , <code>LoR</code> , <code>HiR</code>
<code>'dwtEXTM'</code>	DWT extension mode (see <code>dwtmode</code> )
<code>'dwtShift'</code>	DWT shift parameter (0 or 1)
<code>'dataSize'</code>	Size of original matrix <code>X</code>
<code>'ca'</code>	Approximation coefficients at level <code>dec.level</code>
<code>'cd'</code>	Cell array of detail coefficients, from 1 to <code>dec.level</code>

The coefficients `cA` and `cD{k}`, for ( $k = 1$  to `dec.level`), are matrices and are stored rowwise if `dec.dirDec` is equal to `'r'` or columnwise if `dec.dirDec` is equal to `'c'`.

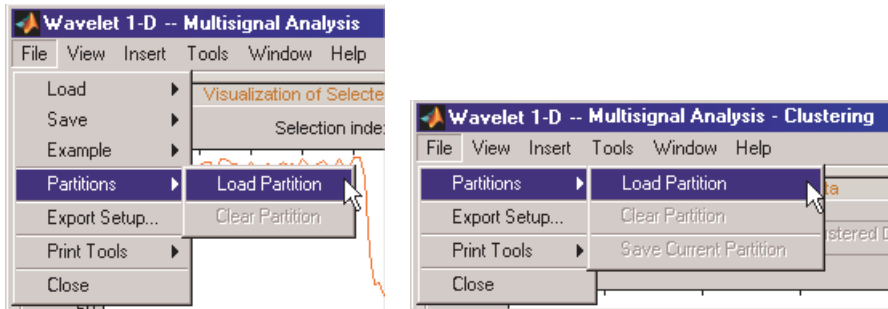
---

**Note** The fields 'wname' and 'dwtFilters' have to be compatible (see the `wfilters` function). The sizes of `cA` and `cD{k}`, (for  $k = 1$  to `dec.level`) must be compatible with the direction, the level of the decomposition, and the extension mode.

---

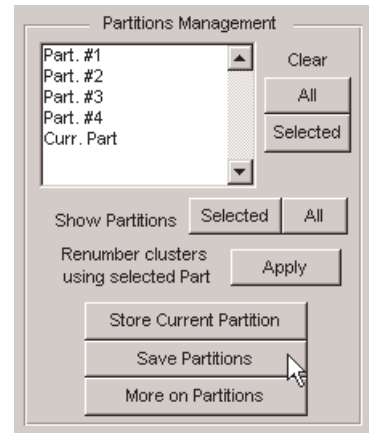
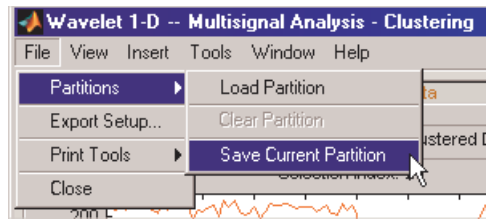
### Loading and Saving Partitions.

**Loading.** The Wavelet 1-D Multisignal Analysis main tool and clustering tool let you load a set of partitions from disk.



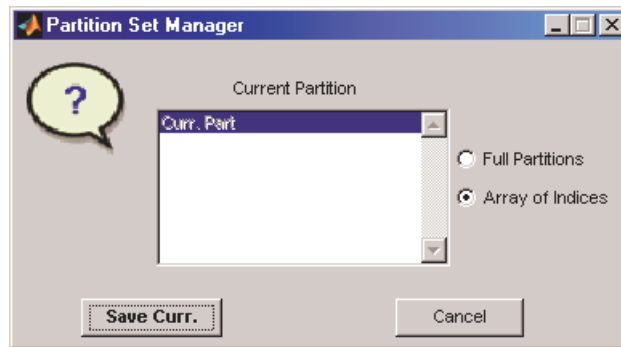


**Saving Partitions.** The Wavelet 1-D Multisignal Analysis clustering tool lets you save a set of partitions to disk.



For example:

- 1 Open the Wavelet 1-D Multisignal Analysis main tool and load the example analysis using **File > Example > Ex 21: Thinker (rows)**.
- 2 Click the **Clustering** button. The Wavelet 1-D Multisignal Analysis Clustering window appears.
- 3 Click the **Compute Clusters** button, and then save the current partition using menu option **File > Partitions > Save Current Partition**. A dialog box appears that lets you specify the type of data to save.
- 4 Click the **Save Curr.** button.



- 5 Another dialog box appears that lets you specify a directory and filename for storing the partition data. Type the name curPART.
- 6 After saving the partition data to the file curPART.mat, load the variables into your workspace:

```
load curPART
whos
```

Name	Size	Bytes	Class
tab_IdxCLU	192x1	1536	double

- 7 You can modify the array tab\_IdxCLU in the workspace, and save the partition data in a file. For example to create two new partitions with four and two clusters, type the following lines:

```
tab_IdxCLU(:,2) = rem((1:192)',4) + 1;
tab_IdxCLU(:,3) = double((1:192) '>96') + 1;
save newpart tab_IdxCLU
```

Now you can use three partitions for the example Ex 21: Thinker (rows). Then, you can load the partitions stored in the file newPART.mat in the Wavelet 1-D Multisignal Analysis main tool and clustering tool.

---

**Note** A partition is a column vector of integers. The values must vary from 1 to `NbClusters` (`NbClusters > 1`), and each cluster must contain at least one element. The number of rows must be equal to the number of signals.

---

## Two-Dimensional True Compression

This section takes you through the features of two-dimensional true compression using the Wavelet Toolbox™ software.

For more information on the compression methods see “True Compression for Images” on page 6-130.

For more information on the main function available when using command-line mode, see the `wcompress` reference pages.

Starting from a given image, the goal of the true compression is to minimize the length of the sequence of bits needed to represent it, while preserving information of acceptable quality. Wavelets contribute to effective solutions for this problem.

The complete chain of compression includes phases of quantization, coding and decoding in addition of the wavelet processing itself.

The purpose of this section is to show how to compress and uncompress a grayscale or truecolor image using various compression methods.

In this section, you'll learn to

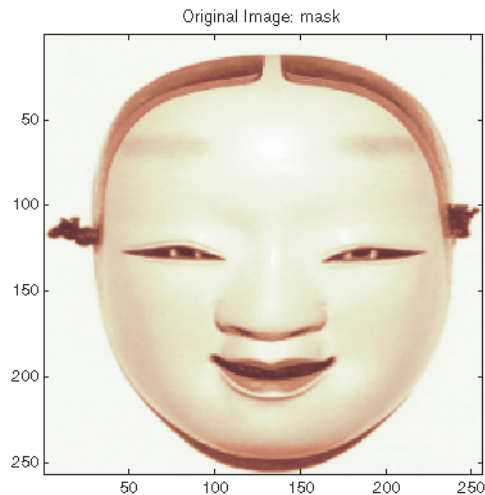
- Compress using global thresholding and Huffman encoding
- Uncompress
- Compress using progressive methods
- Handle truecolor images

### Two-Dimensional True Compression Using the Command Line

#### Compression by Global Thresholding and Huffman Encoding

First load and display the grayscale image mask.

```
load mask;  
image(X)  
axis square;  
colormap(pink(255))  
title('Original Image: mask')
```



A synthetic performance of the compression is given by the compression ratio and the Bit-Per-Pixel ratio which are equivalent.

The compression ratio CR means that the compressed image is stored using only CR% of the initial storage size.

The Bit-Per-Pixel ratio BPP gives the number of bits used to store one pixel of the image.

For a grayscale image, the initial BPP is 8 while for a truecolor image the initial BPP is 24 because 8 bits are used to encode each of the three colors (RGB color space).

The challenge of compression methods is to find the best compromise between a weak compression ratio and a good perceptual result.

Let us begin with a simple method cascading global coefficients thresholding and Huffman encoding. We use the default wavelet *bior4.4* and the default level which is the maximum possible level (see the `wmaxlev` function) divided by 2.

The desired Bit-Per-Pixel ratio BPP is set to 0.5 and the compressed image will be stored in the file named 'mask.wtc'.

```

meth = 'gbl_mmc_h'; % Method name
option = 'c'; % 'c' stands for compression
[CR,BPP] = wcompress(option,X,'mask.wtc',meth,'bpp',0.5)

CR =

    6.6925

BPP =

    0.5354

```

The achieved Bit-Per-Pixel ratio is actually about 0.53 (closed to the desired one) for a compression ratio of 6.7%.

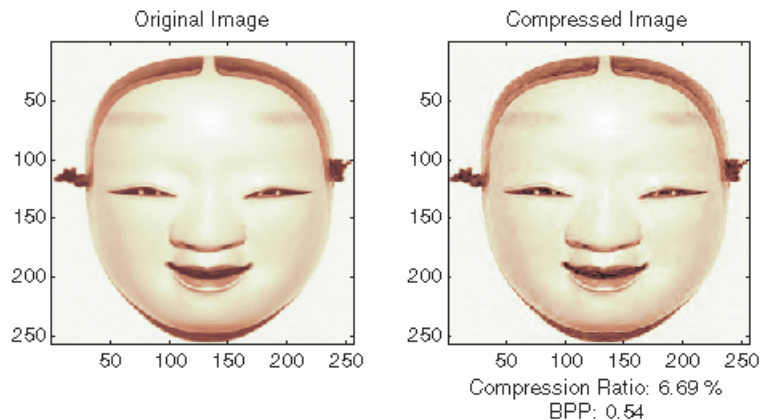
### Uncompression

Let us uncompress the image retrieved from the file 'mask.wtc' and compare it to the original image.

```

option = 'u'; % 'u' stands for uncompression
Xc = wcompress(option,'mask.wtc');
colormap(pink(255))
subplot(1,2,1); image(X);
axis square;
title('Original Image')
subplot(1,2,2); image(Xc);
axis square;
title('Compressed Image')
xlabel({'Compression Ratio: ' num2str(CR,'%1.2f %%'), ...
      ['BPP: ' num2str(BPP,'%3.2f')]})

```



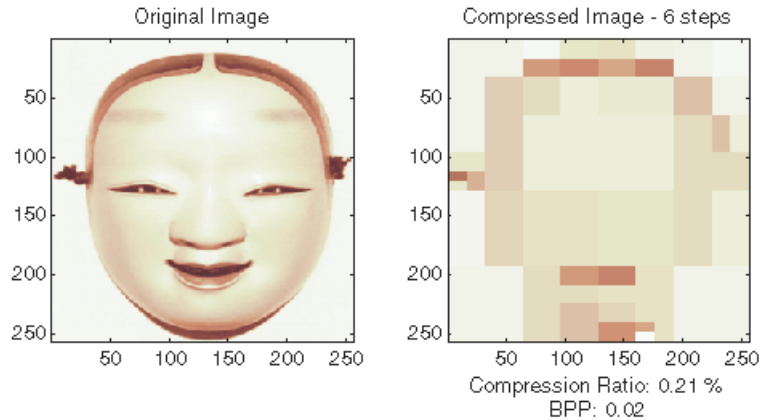
### Compression by Progressive Methods

Let us now illustrate the use of progressive methods starting with the well known EZW algorithm using the Haar wavelet. The key parameter is the number of loops. Increasing it, leads to better recovery but worse compression ratio.

```

meth = 'ezw'; % Method name
wname = 'haar'; % Wavelet name
nbloop = 6; % Number of loops
[CR,BPP] = wcompress('c',X,'mask.wtc',meth,'maxloop', nbloop, ...
                    'wname',wname);
Xc = wcompress('u','mask.wtc');
colormap(pink(255))
subplot(1,2,1); image(X);
axis square;
title('Original Image')
subplot(1,2,2); image(Xc);
axis square;
title('Compressed Image - 6 steps')
xlabel(['Compression Ratio: ' num2str(CR,'%1.2f %%'), ...
       ['BPP: ' num2str(BPP,'%3.2f')])

```



A too small number of steps (here 6) produces a very coarse compressed image. So let us examine a little better result for 9 steps and a satisfactory result for 12 steps.

```
[CR,BPP] = wcompress('c',X,'mask.wtc',meth,'maxloop',9, ...
                    'wname','haar');
```

```
Xc = wcompress('u','mask.wtc');
```

```
colormap(pink(255))
```

```
subplot(1,2,1); image(Xc);
```

```
axis square;
```

```
title('Compressed Image - 9 steps')
```

```
xlabel({'Compression Ratio: ' num2str(CR,'%1.2f %%')},...
       ['BPP: ' num2str(BPP,'%3.2f')])
```

```
[CR,BPP] = wcompress('c',X,'mask.wtc',meth,'maxloop',12, ...
                    'wname','haar');
```

```
Xc = wcompress('u','mask.wtc');
```

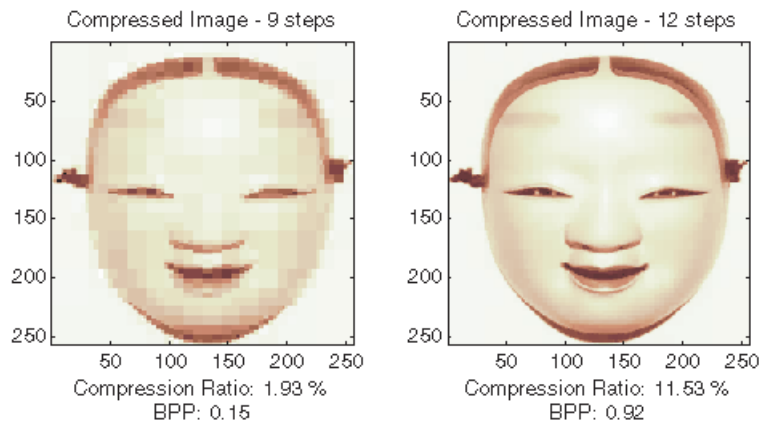
```
subplot(1,2,2); image(Xc);
```

```
axis square;
```

```
title('Compressed Image - 12 steps')
```

```
xlabel({'Compression Ratio: ' num2str(CR,'%1.2f %%')}, ...
       ['BPP: ' num2str(BPP,'%3.2f')])
```



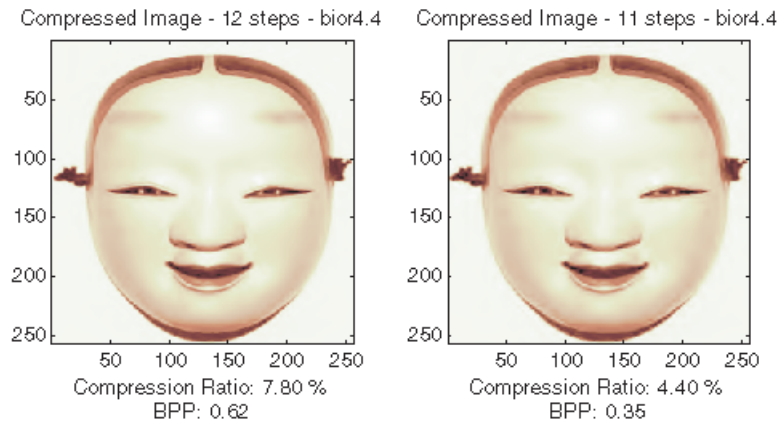


As can be seen, the reached BPP ratio is about 0.92 when using 12 steps.

Let us try to improve it by using the wavelet *bior4.4* instead of *haar* and looking at obtained results for steps 12 and 11.

```
[CR,BPP] = wcompress('c',X,'mask.wtc','ezw','maxloop',12, ...
                    'wname','bior4.4');
Xc = wcompress('u','mask.wtc');
colormap(pink(255))
subplot(1,2,1); image(Xc);
axis square;
title('Compressed Image - 12 steps - bior4.4')
xlabel({'Compression Ratio: ' num2str(CR,'%1.2f %')', ...
       ['BPP: ' num2str(BPP,'%3.2f')']})

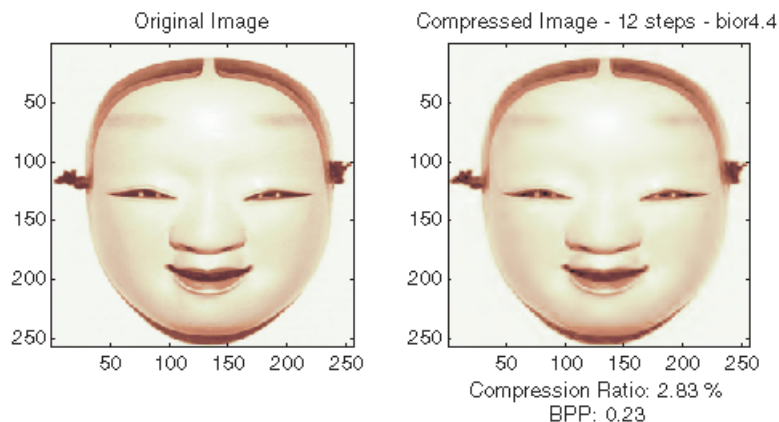
[CR,BPP] = wcompress('c',X,'mask.wtc','ezw','maxloop',11, ...
                    'wname','bior4.4');
Xc = wcompress('u','mask.wtc');
subplot(1,2,2); image(Xc);
axis square;
title('Compressed Image - 11 steps - bior4.4')
xlabel({'Compression Ratio: ' num2str(CR,'%1.2f %')', ...
       ['BPP: ' num2str(BPP,'%3.2f')']})
```



Starting from the eleventh loop, the result can be considered satisfactory. The reached BPP ratio is now about 0.35. It can even be slightly improved by using a more recent method: SPIHT (Set Partitioning In Hierarchical Trees).

```
[CR,BPP] = wcompress('c',X,'mask.wtc','spiht','maxloop',12, ...
                    'wname','bior4.4');
Xc = wcompress('u','mask.wtc');
colormap(pink(255))
subplot(1,2,1); image(X);
axis square;
title('Original Image')
subplot(1,2,2); image(Xc);
axis square;
title('Compressed Image - 12 steps - bior4.4')
xlabel({'Compression Ratio: ' num2str(CR,'%1.2f %')', ...
      ['BPP: ' num2str(BPP,'%3.2f')']})
[psnr,mse] = psnr_mse_maxerr(X,Xc)

delete('mask.wtc')
```



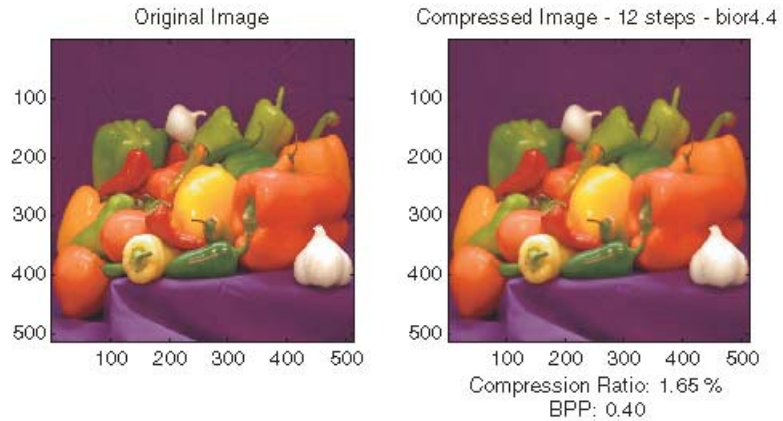
The final compression ratio (2.8%) and the Bit-Per-Pixel ratio (0.23) are very satisfactory. Let us recall that the first ratio means that the compressed image is stored using only 2.8% of the initial storage size.

### Handling Truecolor Images

Finally, let us illustrate how to compress the `wpeppers.jpg` truecolor image. Truecolor images can be compressed along the same scheme as the grayscale images by applying the same strategies to each of the three color components.

The progressive compression method used is SPIHT (Set Partitioning In Hierarchical Trees) and the number of encoding loops is set to 12.

```
X = imread('wpeppers.jpg');
[CR,BPP] = wcompress('c',X,'wpeppers.wtc','spiht','maxloop',12);
Xc = wcompress('u','wpeppers.wtc');
subplot(1,2,1); image(X);
axis square;
title('Original Image')
subplot(1,2,2); image(Xc);
axis square;
title('Compressed Image - 12 steps - bior4.4')
xlabel({'Compression Ratio: ' num2str(CR,'%1.2f %')'}, ...
       ['BPP: ' num2str(BPP,'%3.2f')'])
delete('wpeppers.wtc')
```



The compression ratio (1.65%) and the Bit-Per-Pixel ratio (0.4) are very satisfactory while maintaining a good visual perception.

## Two-Dimensional True Compression Using the Graphical Interface

In this section, we explore the different methods for two-dimensional true compression, using the graphical interface tools.

- 1 Start the True Compression 2-D Tool.

From the MATLAB® prompt, type

```
wavemenu
```

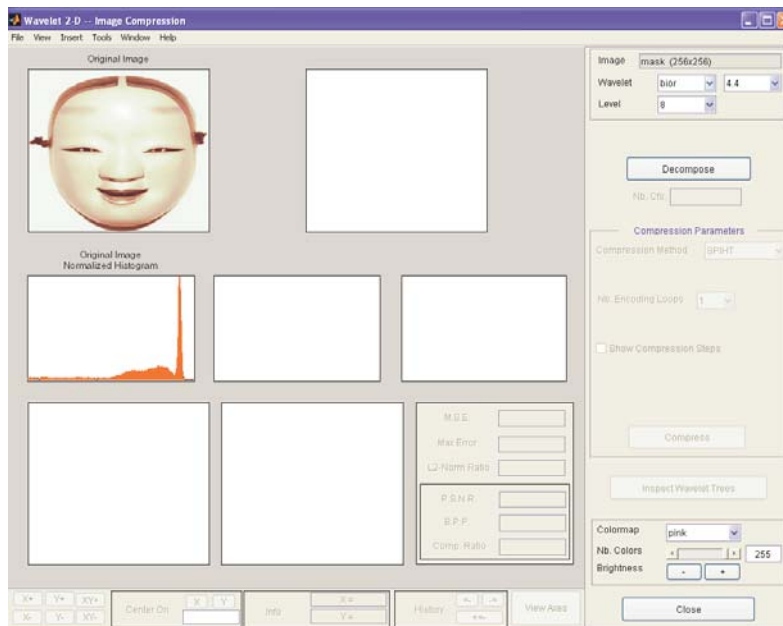
The **Wavelet Toolbox Main Menu** appears. Click the **True Compression 2-D menu** item. The true compression tool for images appears.

2 Load the image.

From the **File** menu, choose the **Load Image** option and select the **Matlab Supported Formats** item.

When the **Load Image** dialog box appears, select the **MAT-file** mask.mat, which should reside in the **MATLAB** directory toolbox/wavelet/wavedemo.

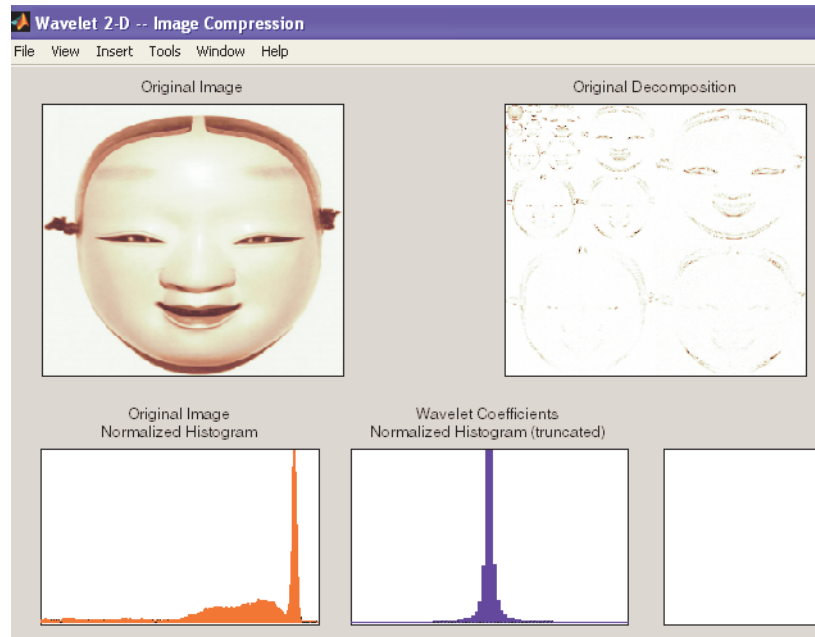
Click the **OK** button. A window appears asking you if you want to consider the loaded image as a truecolor one. Click the **No** button since it is a grayscale image. The mask image is loaded into the **True Compression 2-D** tool. It appears at the top left of the window together with the gray level histogram just below.



3 Perform a Wavelet Decomposition.

Accept the default wavelet **bior4.4** and select **4** from the **Level** menu which is the maximum possible level divided by 2 and then click the **Decompose** button. After a pause for computation, the tool displays the wavelet

approximation and details coefficients of the decomposition for the three directions, together with the histogram of the original coefficients.



The peak of the bin containing zero illustrates the capability of wavelets to concentrate the image energy into a few nonzero coefficients.

#### 4 Try a simple method.

Begin with a simple method cascading global coefficients thresholding and Huffman encoding.

Choose the **GBL\_MMC\_H** option from the menu **Compression method** located at the top right of the **Compression Parameters** frame. For more information on the compression methods, see “True Compression for Images” on page 6-130.

Set the desired Bit-Per-Pixel ratio to **0.5**.

The screenshot shows a software interface for wavelet processing. It is divided into two main sections: decomposition and compression.

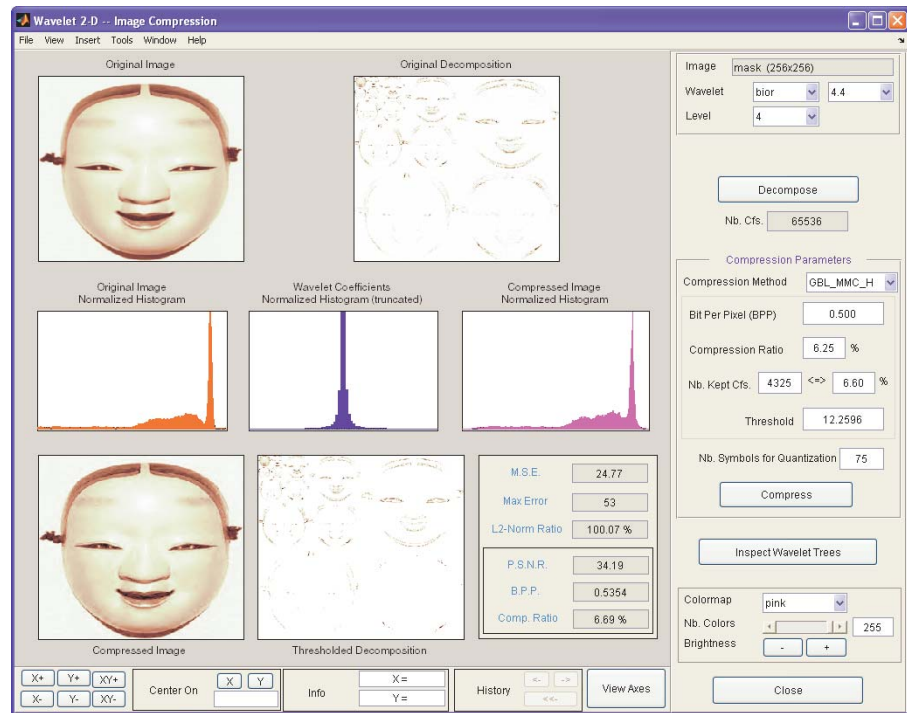
**Decomposition Section:**

- Image: mask (256x256)
- Wavelet: bior (dropdown), 4.4 (dropdown)
- Level: 4 (dropdown)
- Decompose button
- Nb. Cfs.: 65536

**Compression Parameters Section:**

- Compression Method: GBL\_MMC\_H (dropdown)
- Bit Per Pixel (BPP): 0.500
- Compression Ratio: 6.25 %
- Nb. Kept Cfs.: 4325 <=> 6.60 %
- Threshold: 12.2596
- Nb. Symbols for Quantization: 75
- Compress button

Values of the other parameters are automatically updated. Note that these values are only approximations of the true performances since the quantization step cannot be exactly predicted without performing it. Click the **Compress** button.



Synthetic performance is given by the compression ratio and the computed Bit-Per-Pixel (BPP). This last one is actually about 0.53 (close to the desired one 0.5) for a compression ratio of 6.7%.

The compressed image, at the bottom left, can be compared with the original image.

The result is satisfactory but a better compromise between compression ratio and visual quality can be obtained using more sophisticated true compression which combines the thresholding and quantization steps.

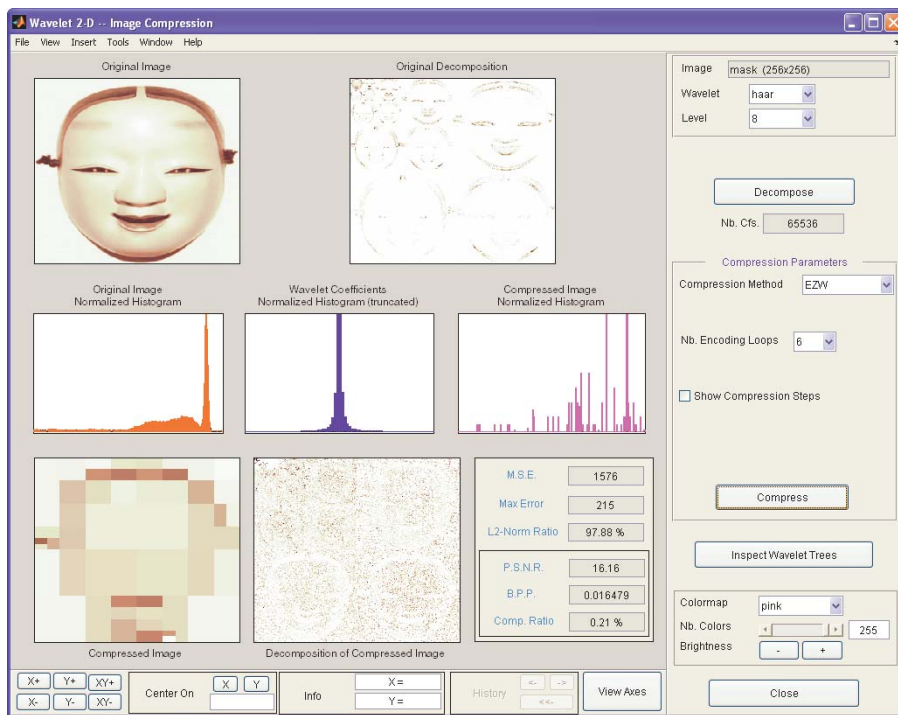
##### 5 Compress using a first progressive method: EZW.

Let us now illustrate the use of progressive methods starting with the well known EZW algorithm. Let us start by selecting the wavelet **haar** from the



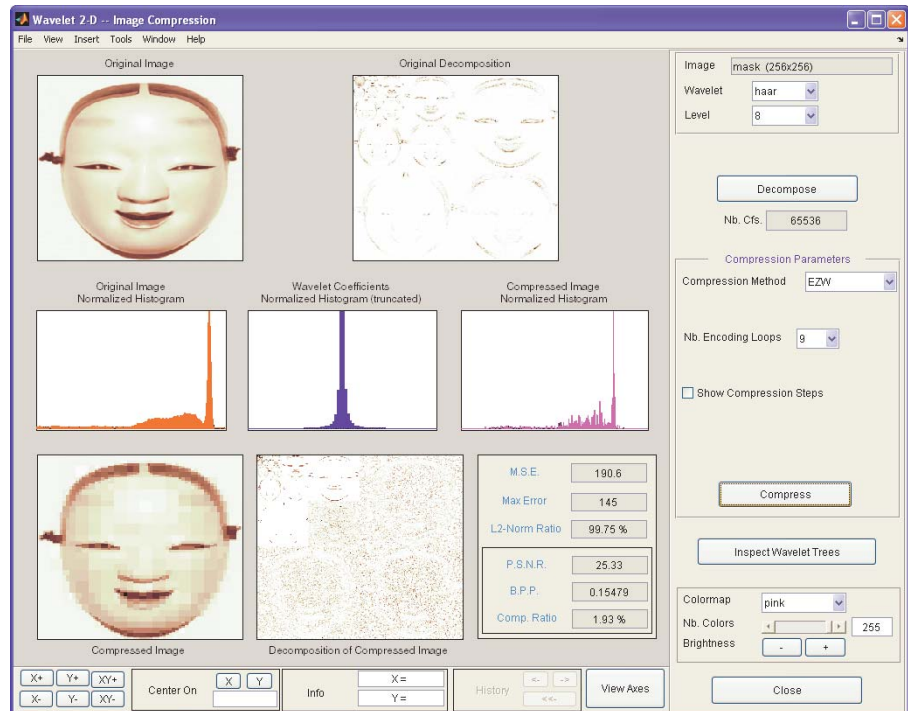
**Wavelet** menu and select **8** from the **Level** menu. Then click the **Decompose** button.

Choose the **EZW** option from the menu **Compression method**. The key parameter is the number of loops: increasing it leads to a better recovery but worse compression ratio. From the **Nb. Encoding Loops** menu, set the number of encoding loops to **6**, which is a small value. Click the **Compress** button.



6 Refine the result by increasing the number of loops.

Too few steps produce a very coarse compressed image. So let us examine a little better result for **9** steps. Set the number of encoding loops to **9** and click the **Compress** button.



As can be seen, the result is better but not satisfactory, both by visual inspection and by calculating the Peak Signal to Noise Ratio (PSNR) which is less than 30.

M.S.E.	190.6
Max Error	145
L2-Norm Ratio	99.75 %
P.S.N.R.	25.33
B.P.P.	0.15479
Comp. Ratio	1.93 %

Now try a large enough number of steps to get a satisfactory result. Set the number of encoding loops to **12** and click the **Compress** button.

The screenshot shows the 'Wavelet 2-D - Image Compression' software window. The interface is divided into several sections:

- Top Left:** 'Original Image' showing a face.
- Top Right:** 'Original Decomposition' showing the wavelet coefficients of the original image.
- Middle Row:** Three histograms: 'Original Image Normalized Histogram', 'Wavelet Coefficients Normalized Histogram (truncated)', and 'Compressed Image Normalized Histogram'.
- Bottom Left:** 'Compressed Image' showing the reconstructed face.
- Bottom Middle:** 'Decomposition of Compressed Image' showing the wavelet coefficients of the compressed image.
- Bottom Right (Metrics):**

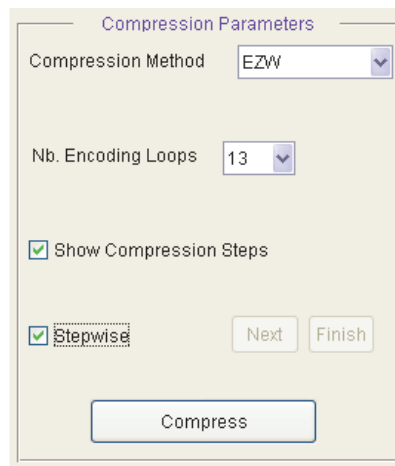
M.S.E.	10.85
Max Error	28
L2-Norm Ratio	99.99 %
P.S.N.R.	37.78
B.P.P.	0.92273
Comp. Ratio	11.53 %
- Right Panel (Parameters):**
  - Image: mask (256x256)
  - Wavelet: haar
  - Level: 8
  - Decompose button
  - Nb. Cfs: 65536
  - Compression Method: EZW
  - Nb. Encoding Loops: 12
  - Show Compression Steps:
  - Compress button
  - Inspect Wavelet Trees button
  - Colormap: pink
  - Nb. Colors: 255
  - Brightness: - +
  - Close button
- Bottom Bar:** Navigation and utility buttons including X+, Y+, XY+, X-, Y-, XY-, Center On, Info, X=, Y=, History, and View Axes.

The result is now acceptable. But for 12 steps, we attain a Bit-Per-Pixel ratio about 0.92.

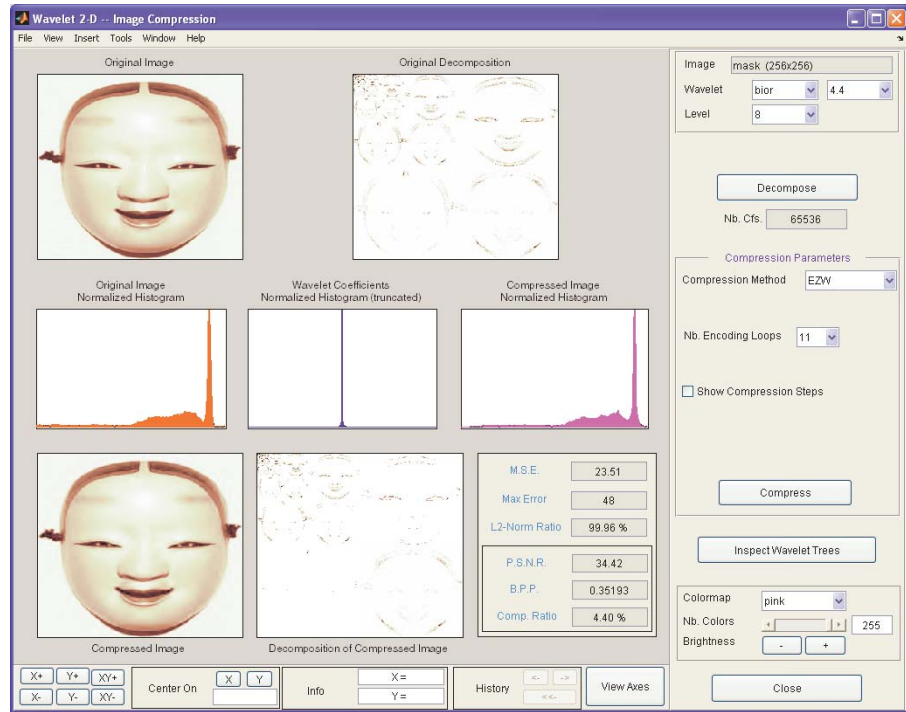
- 7 Get better compression performance by changing the wavelet and selecting the best adapted number of loops.

Let us try to improve the compression performance by changing the wavelet: select **bior4.4** instead of **haar** and then click the **Decompose** button.

In order to select the number of loops, the GUI tool allows you to examine the successive results obtained by this kind of stepwise procedure. Set the number of encoding loops at a large value, for example **13**, and click the **Show Compression Steps** button. Moreover you could execute the procedure stepwise by clicking the **Stepwise** button.



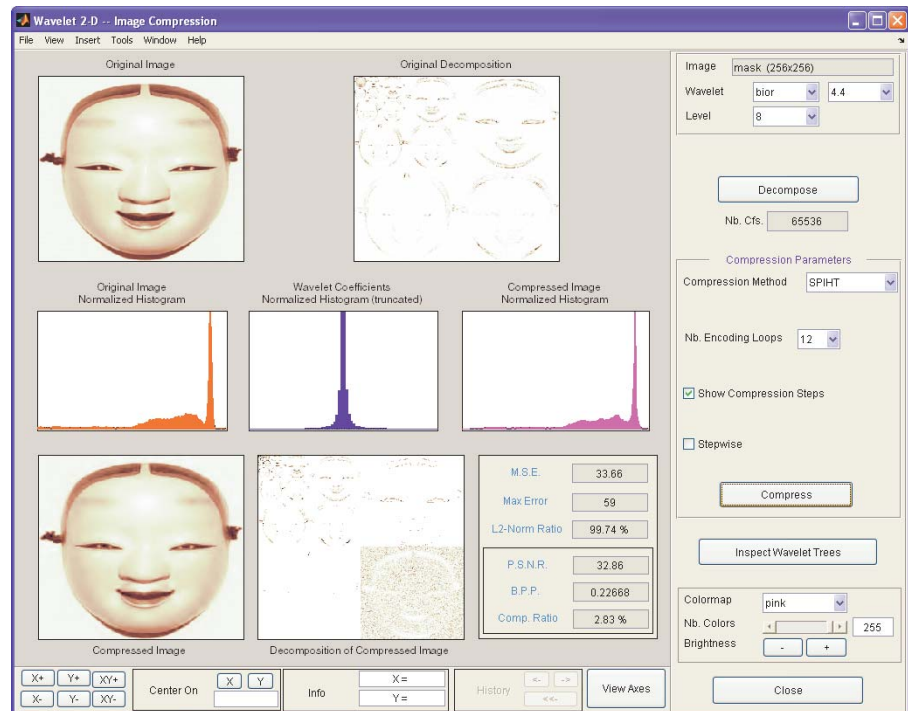
Then, click the **Compress** button. Thirteen progressively more finely compressed images appear, and you can then select visually a convenient value for the number of loops. A satisfactory result seems to be obtained after 11 loops. So, you can set the number of encoding loops to **11** and click the **Compress** button.



The reached BPP ratio is now about 0.35 which is commonly considered a very satisfactory result. Nevertheless, it can be slightly improved by using a more recent method SPIHT (Set Partitioning In Hierarchical Trees).

8 Obtain a final compressed image by using a third method.

Choose the **SPIHT** option from the menu **Compression method**, set the number of encoding loops to **12**, and click the **Compress** button.



The final compression ratio and the Bit-Per-Pixel ratio are very satisfactory: 2.8% and 0.22. Let us recall that the first ratio means that the compressed image is stored using only 2.8% of the initial storage size.

### 9 Handle truecolor images.

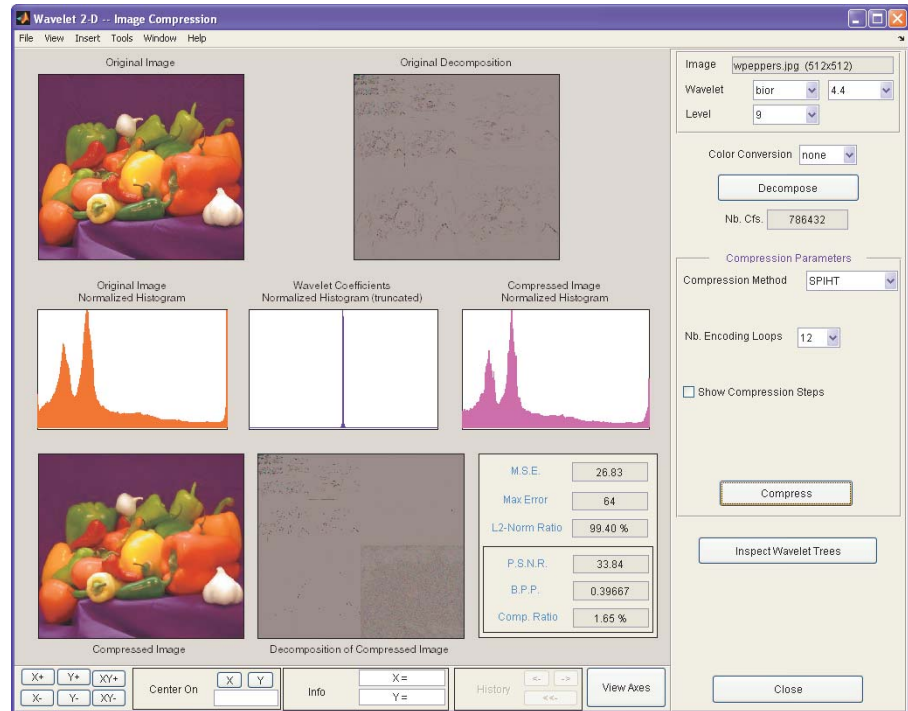
Finally, let us illustrate how to compress truecolor images. The truecolor images can be compressed along the same scheme by applying the same strategies to each of the three-color components.

From the **File** menu, choose the **Load Image** option and select the **Matlab Supported Formats** item.

When the **Load Image** dialog box appears, select the MAT-file `wpeppers.jpg` which should reside in the MATLAB directory `toolbox/wavelet/wavedemo`.

Click the **OK** button. A window appears asking you if you want to consider the loaded image as a truecolor one. Click the **Yes** button. Accept the defaults for wavelet and decomposition level menus and then click the **Decompose** button.

Then, accept the default compression method **SPIHT** and set the number of encoding loops to **12**. Click the **Compress** button.

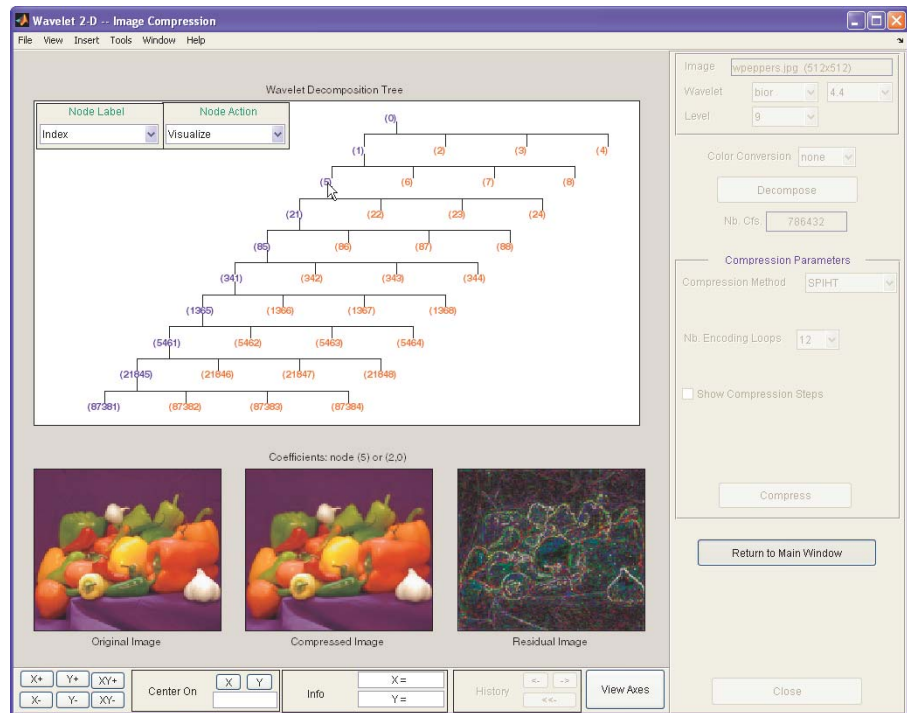


The compression ratio and Bit-Per-Pixel (BPP) ratio are very satisfactory: 1.65% and 0.4 together with a very good perceptual result.

#### 10 Inspect the wavelet tree.

For both grayscale and truecolor images, more insight on the multiresolution structure of the compressed image can be retrieved by clicking the **Inspect Wavelet Trees** button and then on the various active menus available from the displayed tree.





### Importing and Exporting from the GUI

You can save the compressed image to disk in the current directory with a name of your choice.

The Wavelet Toolbox compression tools can create a file using either one of the Matlab Supported Format types or a specific format which can be recognized by the extension of the file: **wtc** (Wavelet Toolbox Compressed).

To save the above compressed image, use the menu option **File > Save Compressed Image > Wavelet Toolbox Compressed Image**. A dialog box appears that lets you specify a directory and filename for storing the image. Of course, the use of the **wtc** format requires you to uncompress the stored image using the Wavelet Toolbox true compression tools.



# Wavelet Applications

---

This chapter explores various applications of wavelets by presenting a series of sample analyses.

- Introduction to Wavelet Analysis (p. 3-2)
- Detecting Discontinuities and Breakdown Points I (p. 3-3)
- Detecting Discontinuities and Breakdown Points II (p. 3-6)
- Detecting Long-Term Evolution (p. 3-8)
- Detecting Self-Similarity (p. 3-10)
- Identifying Pure Frequencies (p. 3-12)
- Suppressing Signals (p. 3-15)
- De-Noising Signals (p. 3-18)
- De-Noising Images (p. 3-21)
- Compressing Images (p. 3-26)
- Fast Multiplication of Large Matrices (p. 3-28)

# 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**.

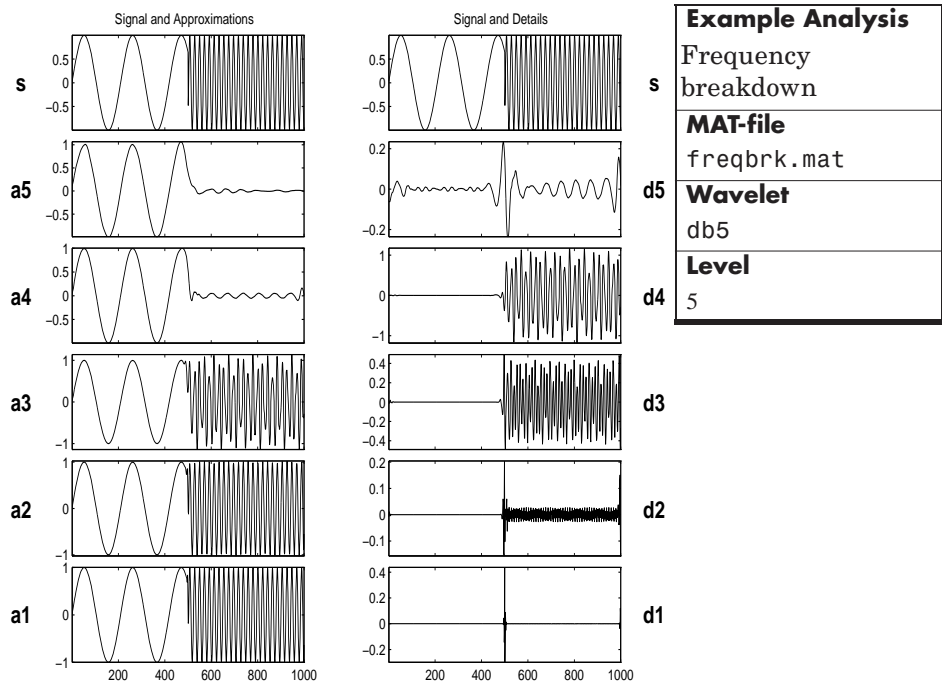
Feel free to explore on your own — use the different options provided in the graphical interface to look at different components of the signal, to compress or de-noise the signal, to examine signal statistics, or to zoom in and out on different signal features.

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 directory `toolbox/wavelet/wavedemo`.

There are also other signals in the `wavedemo` directory that you can analyze on your own.

## Detecting Discontinuities and Breakdown Points I

The purpose of this example is to show how analysis by wavelets can detect the exact instant when a signal changes. The discontinuous signal consists of a slow sine wave abruptly followed by a medium sine wave.



The first- and second-level details ( $D_1$  and  $D_2$ ) show the discontinuity most clearly, because the rupture contains the high-frequency part. Note that if we were *only* interested in identifying the discontinuity, db1 would be a more useful wavelet to use for the analysis than db5.

The discontinuity is localized very precisely: only a small domain around time = 500 contains any large first- or second-level details.

Here is a noteworthy example of an important advantage of wavelet analysis over Fourier. If the same signal had been analyzed by the Fourier transform, we would not have been able to detect the instant when the signal's frequency changed, whereas it is clearly observable here.

Details  $D_3$  and  $D_4$  contain the medium sine wave. The slow sine is clearly isolated in approximation  $A_5$ , from which the higher-frequency information has been filtered.

## Discussion

The deterministic part of the signal may undergo abrupt changes such as a jump, or a sharp change in the first or second derivative. In image processing, one of the major problems is edge detection, which also involves detecting abrupt changes. Also in this category, we find signals with very rapid evolutions such as transient signals in dynamic systems.

The main characteristic of these phenomena is that the change is localized in time or in space.

The purpose of the analysis is to determine

- The site of the change (e.g., time or position)
- The type of change (a rupture of the signal, or an abrupt change in its first or second derivative)
- The amplitude of the change

The local aspects of wavelet analysis are well adapted for processing this type of event, as the processing scales are linked to the speed of the change.

## Guidelines for Detecting Discontinuities

Short wavelets are often more effective than long ones in detecting a signal rupture. In the initial analysis scales, the support is small enough to allow fine analysis. The shapes of discontinuities that can be identified by the smallest wavelets are simpler than those that can be identified by the longest wavelets. Therefore, to identify

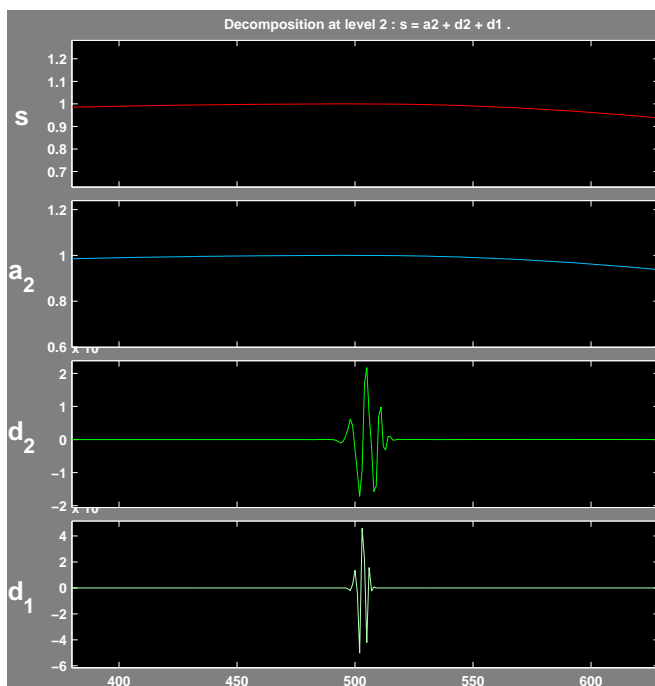
- A signal discontinuity, use the haar wavelet
- A rupture in the  $j$ -th derivative, select a sufficiently regular wavelet with at least  $j$  vanishing moments. (See "Detecting Discontinuities and Breakdown Points II" on page 3-6.)

The presence of noise, which is after all a fairly common situation in signal processing, makes identification of discontinuities more complicated. If the first levels of the decomposition can be used to eliminate a large part of the noise, the rupture is sometimes visible at deeper levels in the decomposition.

Check, for example, the sample analysis **File**⇒**Example Analysis**⇒**Basic Signals**⇒**ramp + white noise** (MAT-file `wnois1op`). 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

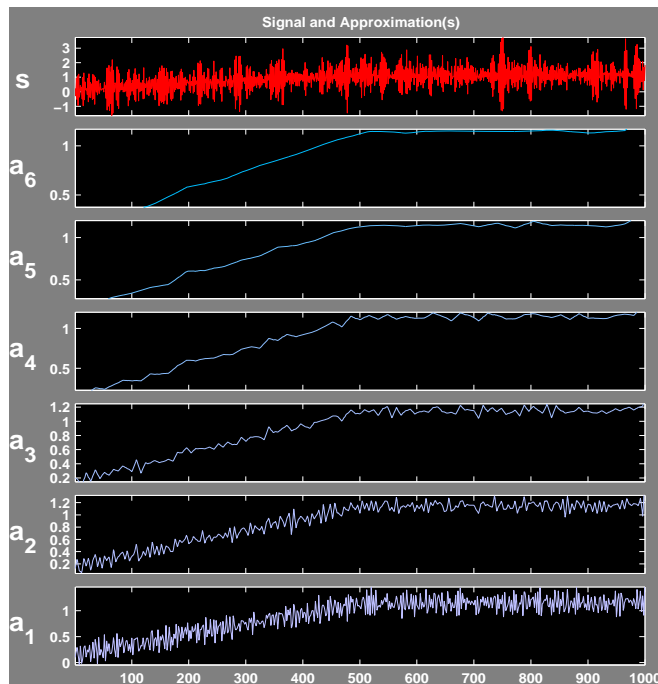
Regularity can be an important criterion in selecting a wavelet. We have chosen to use db4, which is sufficiently regular for this analysis. Had we chosen the haar wavelet, the discontinuity would not have been detected. If you try repeating this analysis using haar at level two, you'll notice that the 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 6-61 and "Wavelet Families: Additional Discussion" on page 6-71 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.)



### Example Analysis

Ramp + colored noise

#### MAT-file

cnoislop.mat

#### Wavelet

db3

#### Level

6

There is so much noise in the original signal,  $s$ , that its overall shape is not apparent upon visual inspection. In this level-6 analysis, we note that the trend becomes more and more clear with each approximation,  $A_1$  to  $A_6$ . 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.

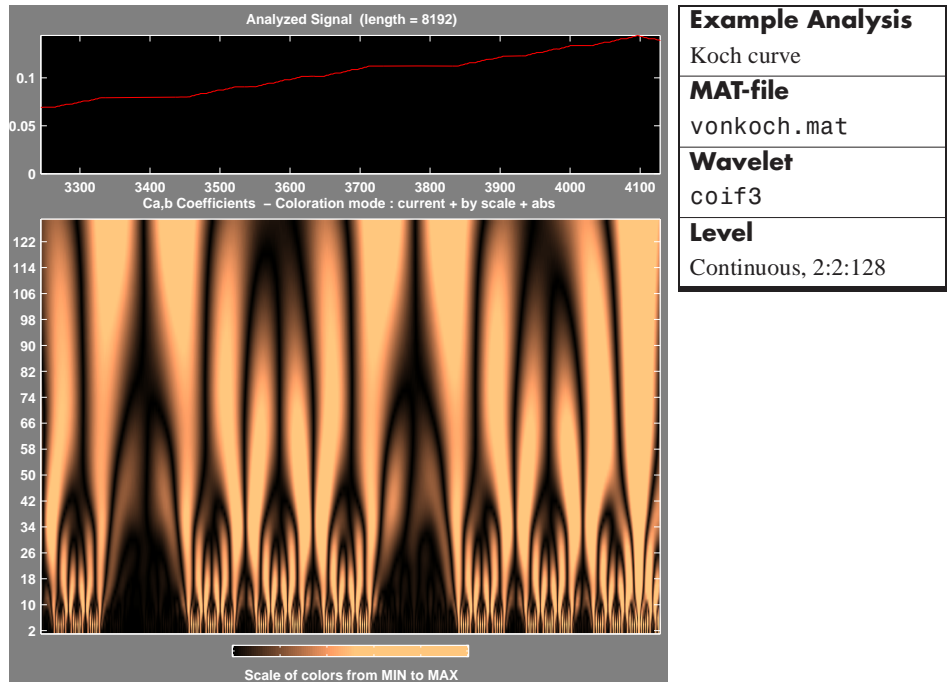
## Discussion

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.



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.

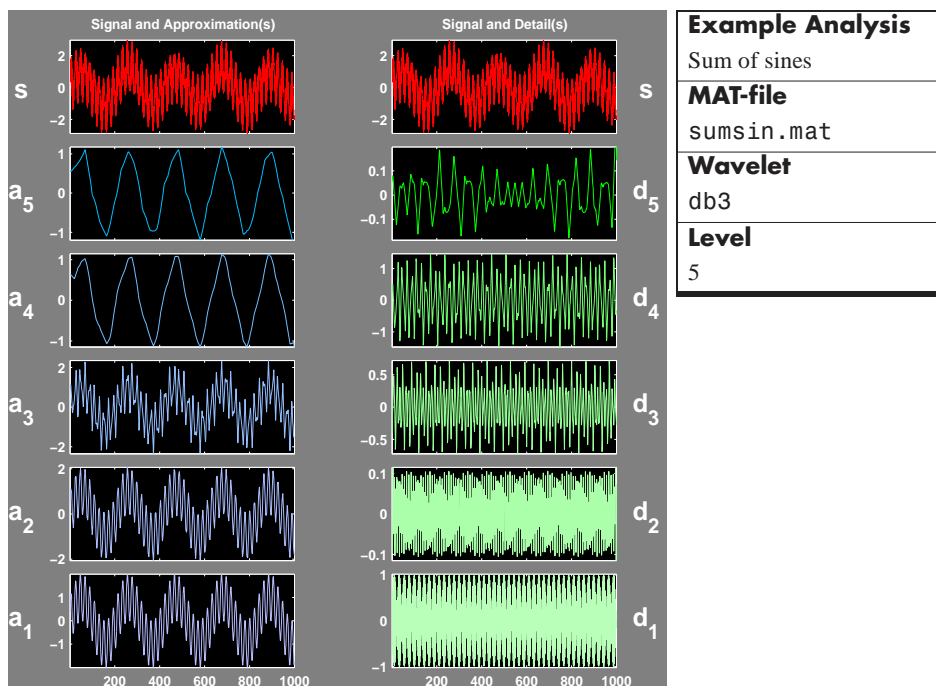
## **Discussion**

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.



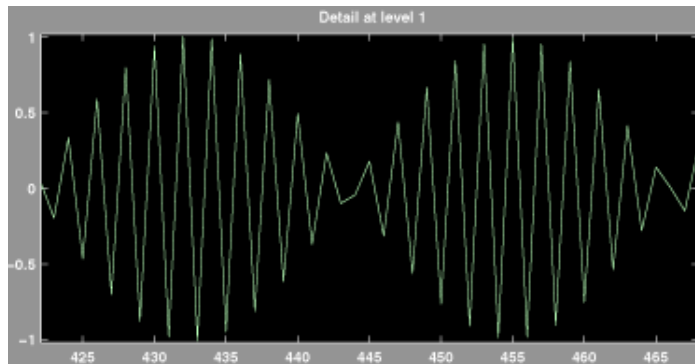
### Discussion

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.

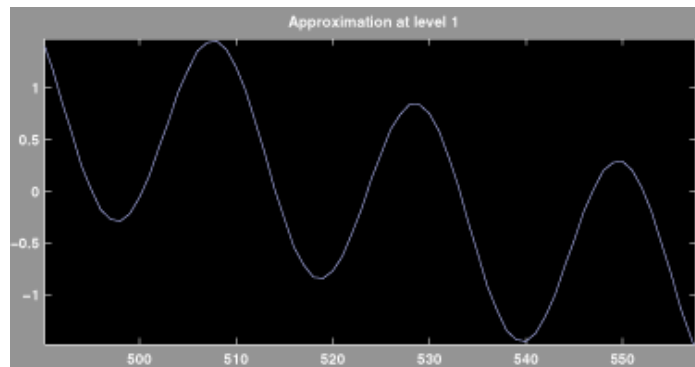
The slow, medium, and rapid sinusoids appear most clearly in approximation  $A_4$ , detail  $D_4$ , and detail  $D_1$ , respectively. The slight differences that can be observed on the decompositions can be attributed to the sampling period.

Detail  $D_1$  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 3-12) 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

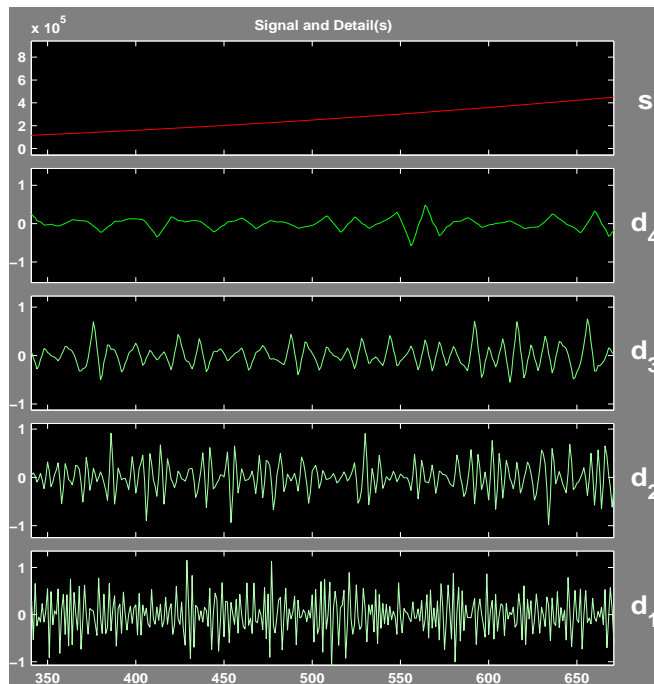
Level	Scale	Pseudo-Period	Pseudo-Frequency
1	2	2.5	0.4
2	4	5	0.2
3	8	10	0.1
4	16	20	0.05
5	32	40	0.025

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

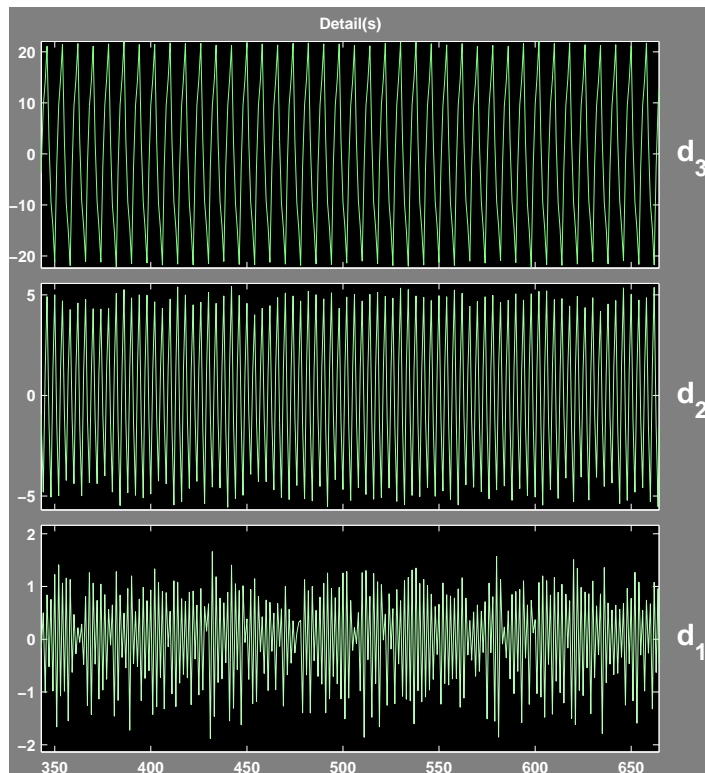
#### Level

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 6-72.

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  $D_1$ ,  $D_2$ , and  $D_3$  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.

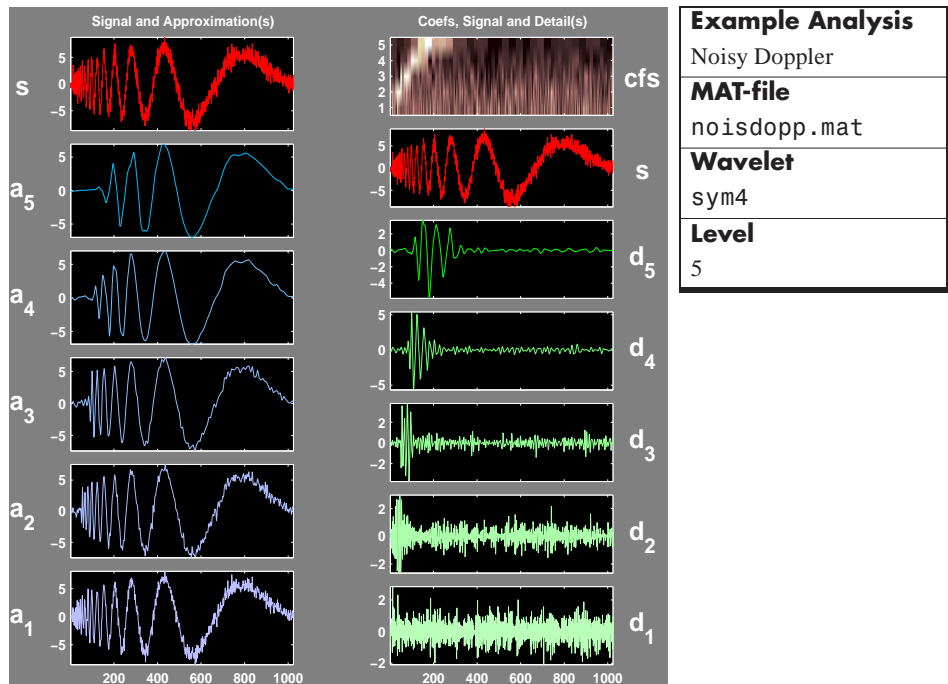
### **Vanishing Moments**

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 6-61 and “Wavelet Families: Additional Discussion” on page 6-71. 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  $k = 0, \dots, n$ , then the wavelet has  $n + 1$  vanishing moments and polynomials of degree  $n$  are suppressed by this wavelet.

## De-Noising Signals

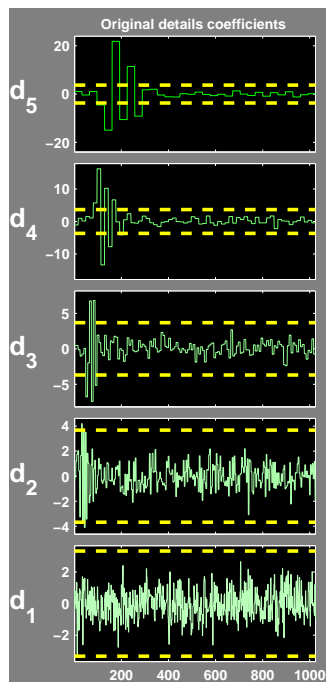
The purpose of this example is to show how to de-noise a signal using wavelet analysis. This example also gives us an opportunity to demonstrate the automatic thresholding feature of the **Wavelet 1-D** graphical interface tool. The signal to be analyzed is a Doppler-shifted sinusoid with some added noise.



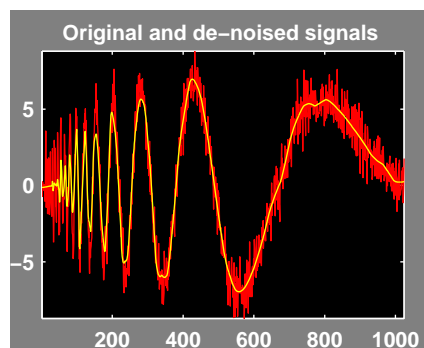
### Discussion

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  $A_5$ , 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.



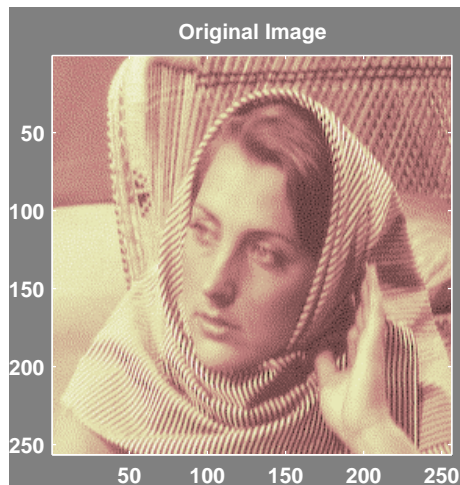
Note that the de-noised signal is flat initially. Some of the highest-frequency signal information was lost during the de-noising process, although less was lost here than in the higher level approximations A4 and A5.

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.



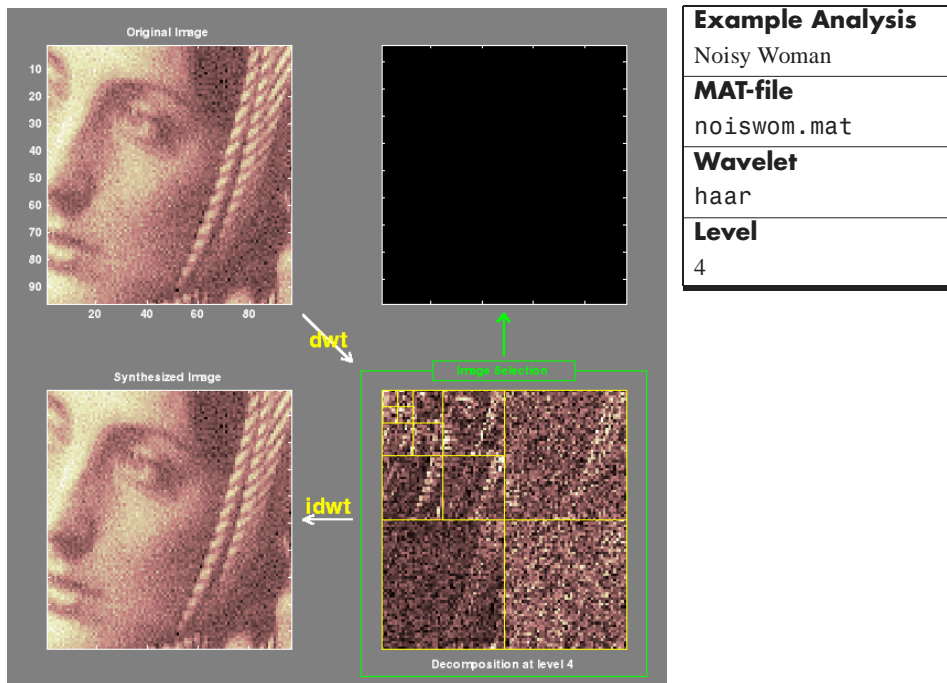
For this example, switch the extension mode to symmetric padding using the command

```
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<sup>®</sup> directory `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.

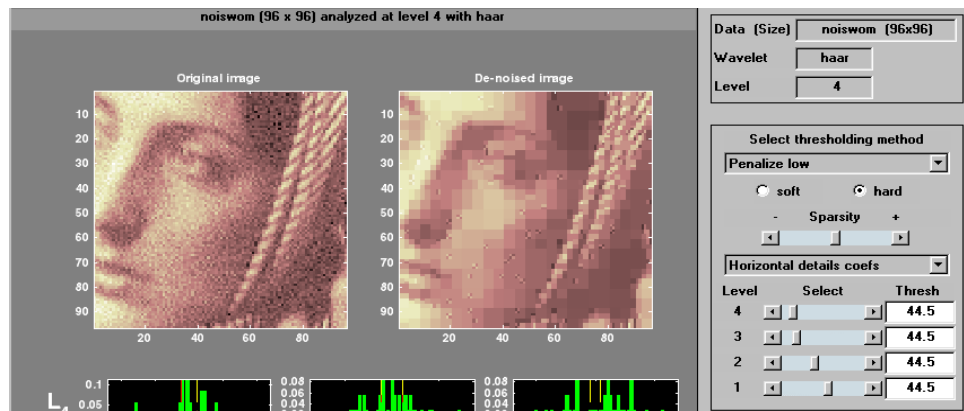


Click the **De-noise** button (located at the middle right) to bring up the **Wavelet 2-D -- De-noising** window.

### Discussion

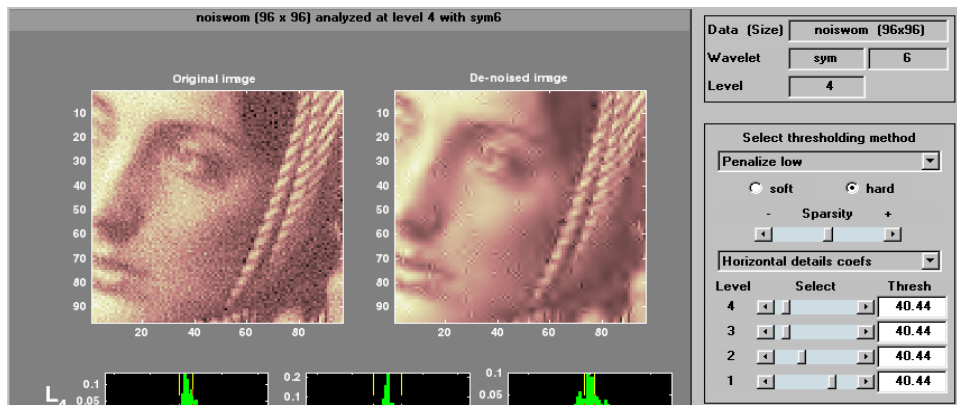
The graphical tool provides automatically generated thresholds. From the **Select thresholding method** menu, select the item **Penalize low** and click the **De-noise** button.





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.

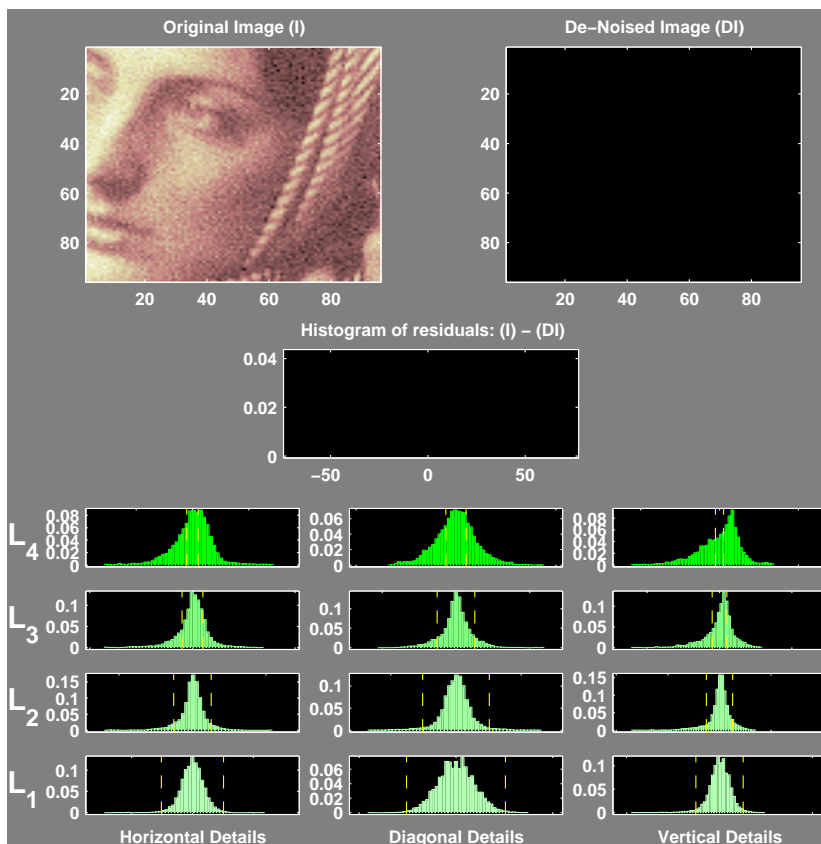
From the **Select thresholding method** menu, select the item **Penalize low**, and click the **De-noise** button.



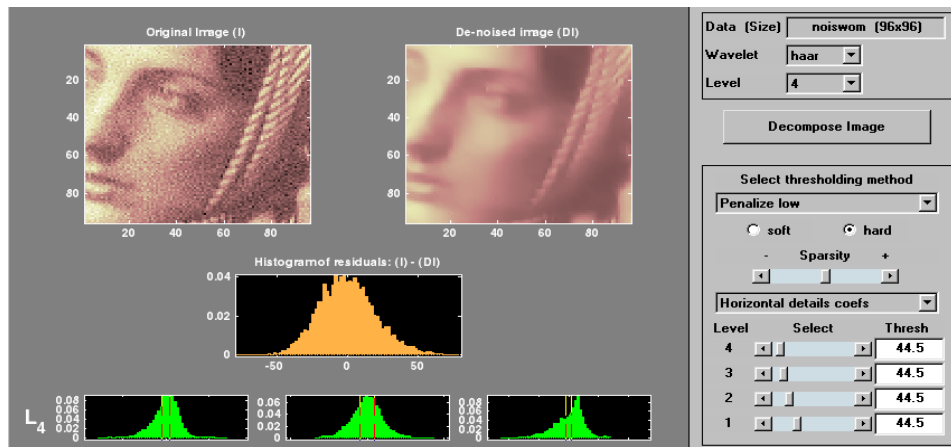
The de-noised image exhibits some ringing effects. Let's try another strategy based on the two-dimensional stationary wavelet analysis to de-noise images. The basic idea is to average many slightly different discrete wavelet analyses.

For more information, see the section “Discrete Stationary Wavelet Transform (SWT)” on page 6-44.

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.

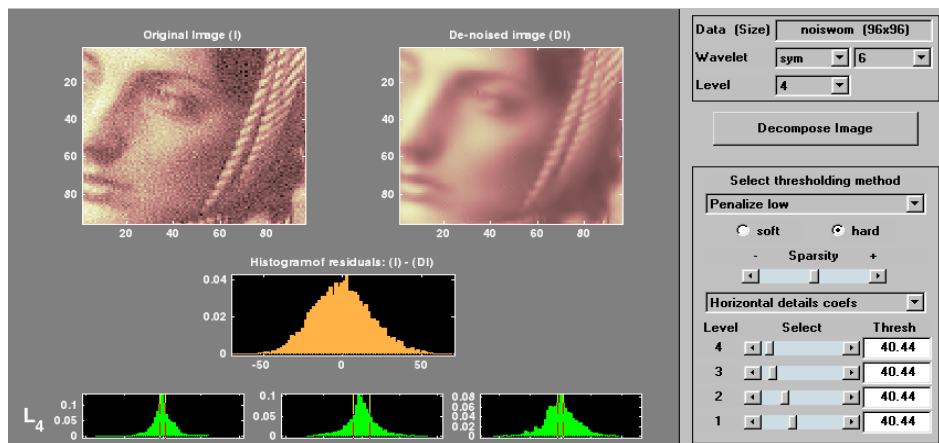


The selected thresholding method is **Penalize low**. Use the **Sparsity** slider to adjust the threshold value close to 44.5 (the same as before to facilitate the comparison with the first trial), and then click the **De-noise** button.



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.



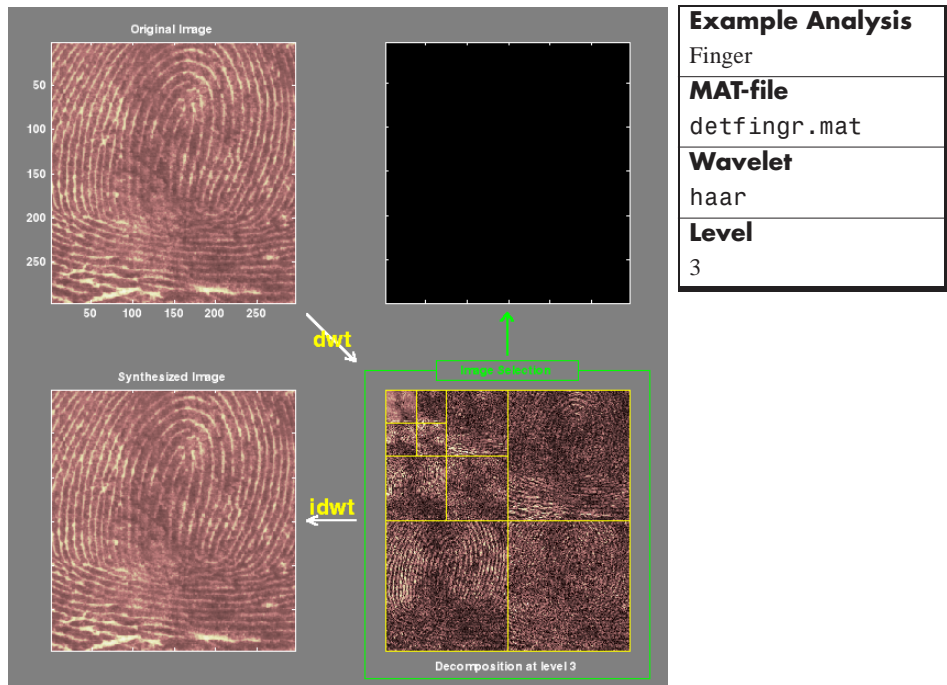
At the end of this example, turn back the extension mode to zero-padding using the command

```
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.

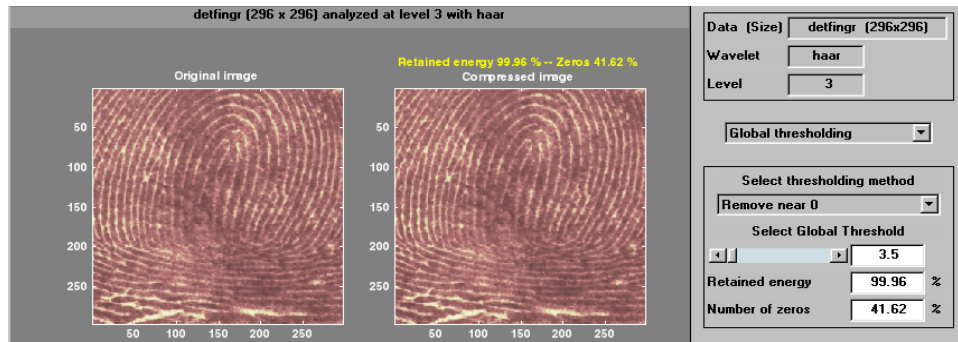
For this example, open the **Wavelet 2-D** tool and select the menu item **File⇒Example Analysis⇒at level 3, with haar** → **finger**.



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.

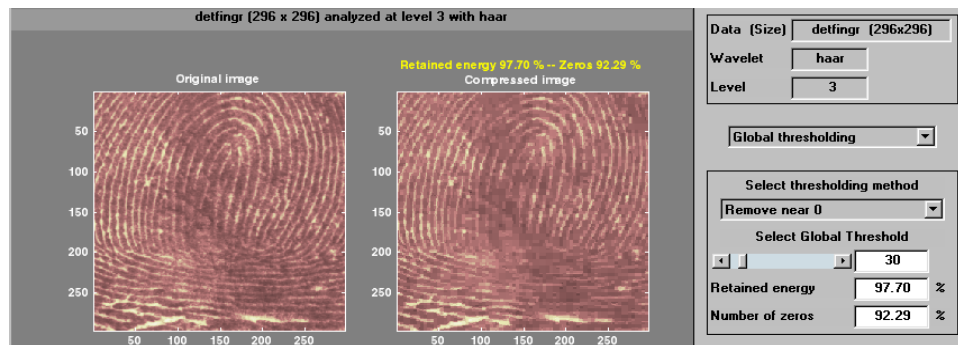
## Discussion

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:
    - a Compute vector approximation.
    - b Compute multiplication in wavelet domain.
    - c 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.  
rnorm = 100 * (norm(p-sp)/norm(p))
```

```
rnorm =  
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  
rnorm = 100 * (norm(p-sp)/norm(p))
```

```
rnorm =  
99.2137
```



# 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 “Advanced Concepts” on page 6-1.

- Illustrated Examples (p. 4-2)
- Case Study: An Electrical Signal (p. 4-36)

## 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 4-8	A sum of sines: $s_1(t) = \sin(3t) + \sin(0.3t) + \sin(0.03t)$	$s_1(t)$	sumsin
Example 2: A Frequency Breakdown on page 4-10	A frequency breakdown: $1 \leq t \leq 500, \quad s_2(t) = \sin(0.03t)$ $501 \leq t \leq 1000, \quad s_2(t) = \sin(0.3t)$	$s_2(t)$	freqbrk
Example 3: Uniform White Noise on page 4-12	A uniform white noise: on the interval $[-0.5 \ 0.5]$	$b_1(t)$	whitnois
Example 4: Colored AR(3) Noise on page 4-14	A colored AR(3) noise: $b_2(t) = -1.5b_2(t-1) - 0.75b_2(t-2) - 0.125b_2(t-3) + b_1(t) + 0.5$	$b_2(t)$	warma
Example 5: Polynomial + White Noise on page 4-16	A polynomial + a white noise: on the interval $[1 \ 1000]$ $s_3(t) = t^2 - t + 1 + b_1(t)$	$s_3(t)$	noispol
Example 6: A Step Signal on page 4-18	A step signal: $1 \leq t \leq 500, \quad s_4(t) = 0$ $501 \leq t \leq 1000, \quad s_4(t) = 20$	$s_4(t)$	wstep

Example	Equation	Signal Name	MAT-File
Example 7: Two Proximal Discontinuities on page 4-20	Two proximal discontinuities: $1 \leq t \leq 499, \quad s_5(t) = 3t$ $500 \leq t \leq 510, \quad s_5(t) = 1500$ $511 \leq t, \quad s_5(t) = 3t - 30$	$s_5(t)$	nearbrk
Example 8: A Second-Derivative Discontinuity on page 4-22	A second-derivative discontinuity: $t \in [-0.5 \ 0.5] \subset R;$ $t < 0, f_3(t) = \exp(-4t^2)$ $t \geq 0, f_3(t) = \exp(-t^2)$ $s_6$ is $f_3$ sampled at $10^{-3}$	$s_6(t)$	scddvbrk
Example 9: A Ramp + White Noise on page 4-24	A ramp + a white noise: $1 \leq t \leq 499, \quad s_7(t) = \frac{3t}{500} + b_1(t)$ $500 \leq t \leq 1000, \quad s_7(t) = 3 + b_1(t)$	$s_7(t)$	wnoislop
Example 10: A Ramp + Colored Noise on page 4-26	A ramp + a colored noise: $1 \leq t \leq 499, \quad s_8(t) = \frac{t}{500} + b_2(t)$ $500 \leq t \leq 1000, \quad s_8(t) = 1 + b_2(t)$	$s_8(t)$	cnoislop
Example 11: A Sine + White Noise on page 4-28	A sine + a white noise: $s_9(t) = \sin(0.03t) + b_1(t)$	$s_9(t)$	noissin

Example	Equation	Signal Name	MAT-File
Example 12: A Triangle + A Sine on page 4-30	<p>A triangle + a sine:</p> $1 \leq t \leq 500, \quad s_{10}(t) = \frac{t-1}{500} + \sin(0.3t)$ $501 \leq t \leq 1000, \quad s_{10}(t) = \frac{1000-t}{500} + \sin(0.3t)$	$s_{10}(t)$	trsin
Example 13: A Triangle + A Sine + Noise on page 4-32	<p>A triangle + a sine + a noise:</p> $501 \leq t \leq 1000,$ $s_{11}(t) = \frac{1000-t}{500} + \sin(0.3t) + b_1(t)$ $1 \leq t \leq 500, \quad s_{11}(t) = \frac{t-1}{500} + \sin(0.3t) + b_1(t)$	$s_{11}(t)$	wntrsin
Example 14: A Real Electricity Consumption Signal on page 4-34	A real electricity consumption signal	—	leleccum

Please note that

- All the decompositions use Daubechies wavelets.
- The examples show the signal, the approximations, and the details.

The examples include specific comments and feature distinct domains — for instance, if the level of decomposition is 5,

- The left column contains the signal and the approximations  $A_5$  to  $A_1$ .
- The right column contains the signal and the details  $D_5$  to  $D_1$ .
- The approximation  $A_1$  is located under  $A_2$ ,  $A_2$  under  $A_3$ , and so on; the same is true for the details.
- The abscissa axis represents the time; the unit for the ordinate axis for approximations and details is the same as that of the signal.

- When the approximations do not provide enough information, they are replaced by details obtained by changing wavelets.
- The examples include questions for you to think about:
  - What can be seen on the figure?
  - What additional questions can be studied?

## Advice to the Reader

You should follow along and process these examples on your own, using either the graphical interface or the command line functions.

*Use the graphical interface* for immediate signal processing. To execute the analyses included in the figures,

- 1** To open the **Wavelet Toolbox Main Menu**, type  
wavemenu
- 2** Select the **Wavelet 1-D** menu option to open the **Wavelet 1-D** tool.
- 3** From the **Wavelet 1-D** tool, choose the **File⇒Example Analysis** menu option.
- 4** From the dialog box, select the sample analysis in question.

This triggers the execution of the examples.

*When using the command line*, follow the process illustrated in this M-file to conduct calculations:

```
% Load original 1-D signal.
load sumsin; s = sumsin;

% Perform the decomposition of s at level 5, using coif3.
w = 'coif3'
[c,l] = wavedec(s,5,w);

% Reconstruct the approximation signals and detail signals at
% levels 1 to 5, using the wavelet decomposition structure [c,l].
for i = 1:5
    A(i,:) = wrcoef('a',c,l,w,i);
```

```
        D(i,:) = wrcoef('d',c,l,w,i);  
    end
```

---

**Note** This loop replaces 10 separate `wrcoef` statements defining approximations and details. The variable `A` contains the five approximations and the variable `D` contains the five details.

---

```
% Plots.  
t = 100:900;  
subplot(6,2,1); plot(t,s(t),'r');  
title('Orig. signal and approx. 1 to 5.');
```

```
subplot(6,2,2); plot(t,s(t),'r');  
title('Orig. signal and details 1 to 5.');
```

```
for i = 1:5,  
    subplot(6,2,2*i+1); plot(t,A(5-i+1,t),'b');  
    subplot(6,2,2*i+2); plot(t,D(5-i+1,t),'g');  
end
```

## About Further Exploration

**Tip 1.** On all figures, visually check that for  $j = 0, 1, \dots, A_j = A_{j+1} + D_{j+1}$ .

**Tip 2.** Don't forget to change wavelets. Test the shortest ones first.

**Tip 3.** Identify edge effects. They will create problems for a correct analysis. At present, there is no easy way to avoid them perfectly. You can use tools described in the section "Dealing with Border Distortion" on page 6-35 and see also the `dwtmode` reference page. They should eliminate or greatly reduce these effects.

**Tip 4.** As much as possible, conduct calculations manually to cross-check results with the values in the graphic representations. Manual calculations are possible with the `db1` wavelet.

For the sake of simplicity in the following examples, we use only the `haar` and `db` family wavelets, which are the most frequently used wavelets.

### Example 1: A Sum of Sines

Analyzing wavelet: *db3*

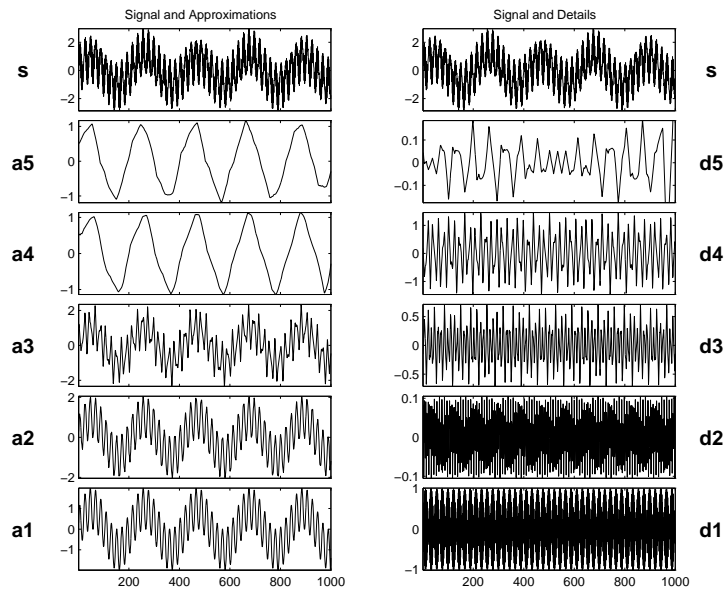
Decomposition levels: 5

The signal is composed of the sum of three sines: slow, medium, and rapid. With regard to the sampling period equal to 1, the periods are approximately 200, 20, and 2 respectively. We should, therefore, see this later period in  $D_1$ , the medium sine in  $D_4$ , and the slow sine in  $A_4$ . The slight differences that can be observed on the decompositions can be attributed to the sampling period. The scale of the approximation charts is 2, 4, or 10 times larger than that of the details.  $D_1$  contains primarily the components whose period is situated between 1 and 2 (i.e., the rapid sine), but this period is not visible at the scale that is used for the graph. Zooming in on  $D_1$  reveals that each “belly” is composed of 10 oscillations, and can be used to estimate the period. We find that the period is close to 2.  $D_2$  is very small. This is also seen in the approximations: the first two resemble one another, since  $A_1 = A_2 + D_2$ .

The detail  $D_3$  and, to an even greater extent, the detail  $D_4$  contain the medium sine. We notice that there is a breakdown between approximations 3 and 4.

Approximations  $A_1$  to  $A_3$  can be used to estimate the period of the medium sine. Now, only the slow sine, which appears in  $A_4$ , remains to be determined. The distance between two successive maximums is equal to 200, which is the period of the slow sine. This latter sine is still visible in  $A_5$ , but will disappear from the approximation and move into the details at level 8.





### Example 1: A Sum of Sines

Addressed topics	<ul style="list-style-type: none"> <li>• Detecting breakdown points</li> <li>• Detecting long-term evolution</li> <li>• Identifying pure frequencies</li> <li>• The effect of a wavelet on a sine</li> <li>• Details and approximations: a signal moves from an approximation to a detail</li> <li>• The level at which characteristics appear</li> </ul>
Further exploration	<ul style="list-style-type: none"> <li>• Compare with a Fourier analysis.</li> <li>• Change the frequencies. Analyze other linear combinations.</li> </ul>

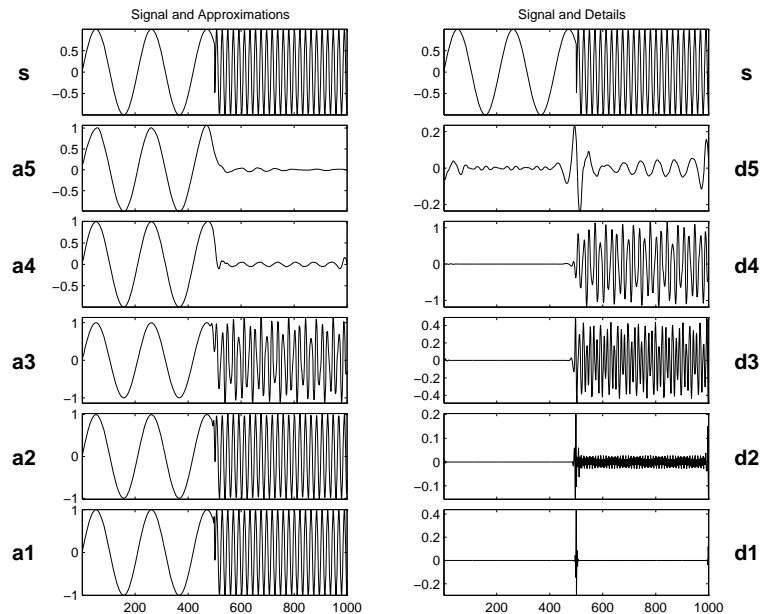
## Example 2: A Frequency Breakdown

Analyzing wavelet: *db5*

Decomposition levels: 5

The signal is formed of a slow sine and a medium sine, on either side of 500. These two sines are not connected in a continuous manner:  $D_1$  and  $D_2$  can be used to detect this discontinuity. It is localized very precisely: only a small domain around 500 contains large details. This is because the rupture contains the high-frequency part; the frequencies in the rest of the signal are not as high. It should be noted that if we are interested only in identifying the discontinuity, *db1* is more useful than *db5*.

$D_3$  and  $D_4$  contain the medium sine as in the previous analysis. The slow sine appears clearly alone in  $A_5$ . It is more regular than in the  $s_1$  analysis, since *db5* is more regular than *db3*. If the same signal had been analyzed by the Fourier transform, we would not have been able to detect the instant corresponding to the signal's frequency change, whereas it is clearly observable here.



---

**Example 2: A Frequency Breakdown**

---

Addressed topics	<ul style="list-style-type: none"><li>• Suppressing signals</li><li>• Detecting long-term evolution</li></ul>
Further exploration	<ul style="list-style-type: none"><li>• Compare to the signal <math>s_1</math>.</li><li>• On a longer signal, select a deeper level of decomposition in such a way that the slow sinusoid appears into the details.</li><li>• Compare with a Fourier analysis.</li><li>• Compare with a windowed Fourier analysis.</li></ul>

---

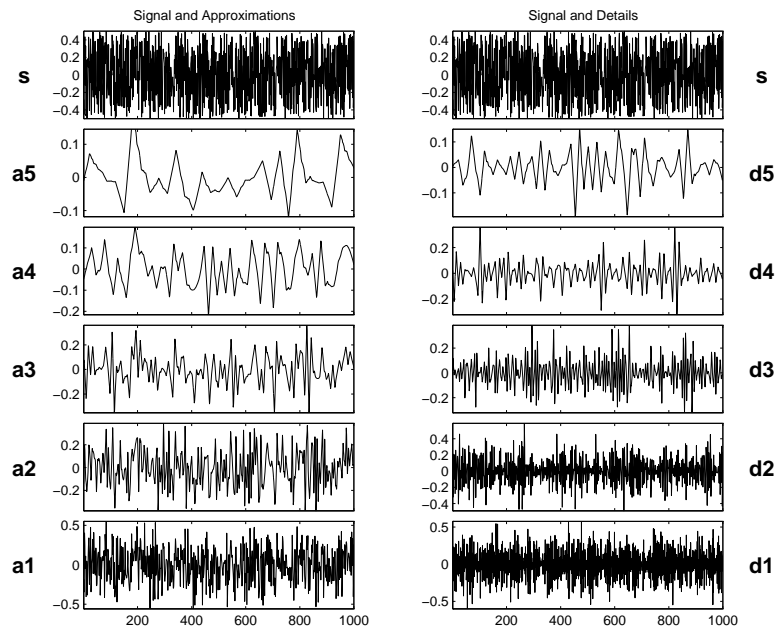
### Example 3: Uniform White Noise

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.,  $\text{variance}(D_j) = \text{variance}(D_{j-1}) / 2$ . 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	<ul style="list-style-type: none"><li>• Processing noise</li><li>• The shapes of the decomposition values</li><li>• The evolution of these shapes according to level; the correlation increases, the variance decreases</li></ul>
Further exploration	<ul style="list-style-type: none"><li>• Compare the frequencies included in the details with those in the approximations.</li><li>• Study the values of the coefficients and their distribution.</li><li>• On the continuous analysis, identify the chaotic aspect of the colors.</li><li>• Replace the uniform white noise by a Gaussian white noise or other noise.</li></ul>

---

## Example 4: Colored AR(3) Noise

Analyzing wavelet: *db3*

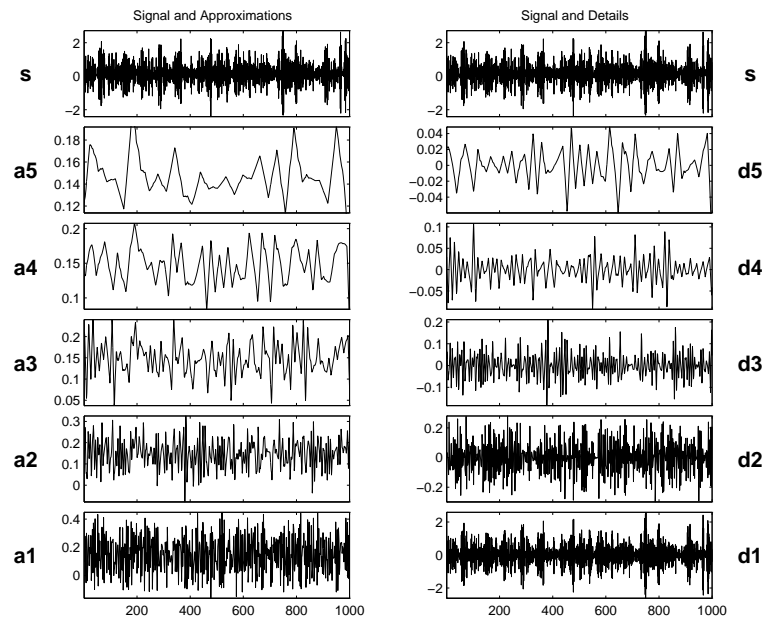
Decomposition levels: 5

---

**Note** AR(3) means AutoRegressive model of order 3.

---

This figure can be examined in view of Example 3: Uniform White Noise on page 4-12, 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 4-12), the noise from which  $b_2$  is built using linear filtering. ( $b_1$  and  $b_2$  are defined explicitly in “Illustrated Examples” on page 4-2, 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 4-12.

- Replace AR(3) with an ARMA (AutoRegressive Moving Average) model noise. For instance,

$$b_3(t) = -1.5b_3(t-1) - 0.75b_3(t-2) - 0.125b_3(t-3) + b_1(t) - 0.7b_1(t-1)$$

- Study an ARIMA (Integrated ARMA) model noise. For instance,

$$b_4(t) = b_4(t-1) + b_3(t)$$

- Check that each detail can be modeled by an ARMA process.
-

### Example 5: Polynomial + White Noise

Analyzing wavelets: *db2* and *db3*

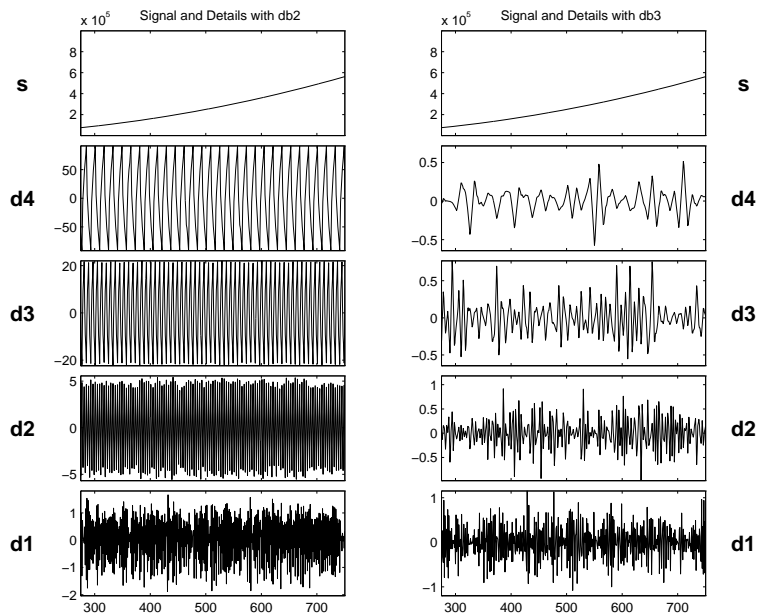
Decomposition levels: 4

The purpose of this analysis is to illustrate the property 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  $t^2 + b_I(t)$ , 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	<ul style="list-style-type: none"> <li>• Suppressing signals</li> <li>• Compare the results of the processing for the following wavelets: the short db2 and the longer db3.</li> <li>• Explain the regularity that is visible in <math>D_3</math> and <math>D_4</math> in the analysis by db2.</li> </ul>
Further exploration	<ul style="list-style-type: none"> <li>• Increase noise intensity and repeat the analysis.</li> </ul>

## Example 6: A Step Signal

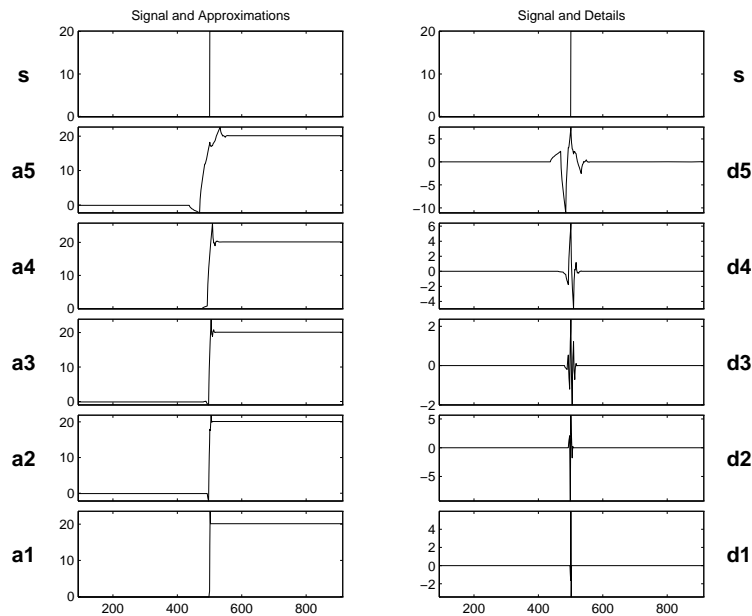
Analyzing wavelet: *db2*

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	<ul style="list-style-type: none"><li>• Detecting breakdown points</li><li>• Suppressing signals</li><li>• Detecting long-term evolution</li><li>• Identifying the range width of the variations of details and approximations</li></ul>
Further exploration	<ul style="list-style-type: none"><li>• Use the coefficients of the FIR filter associated with the wavelet to check the values of <math>D_1</math>.</li><li>• Replace the step by an impulse.</li><li>• Add noise to the signal and repeat the analysis.</li></ul>

---

### Example 7: Two Proximal Discontinuities

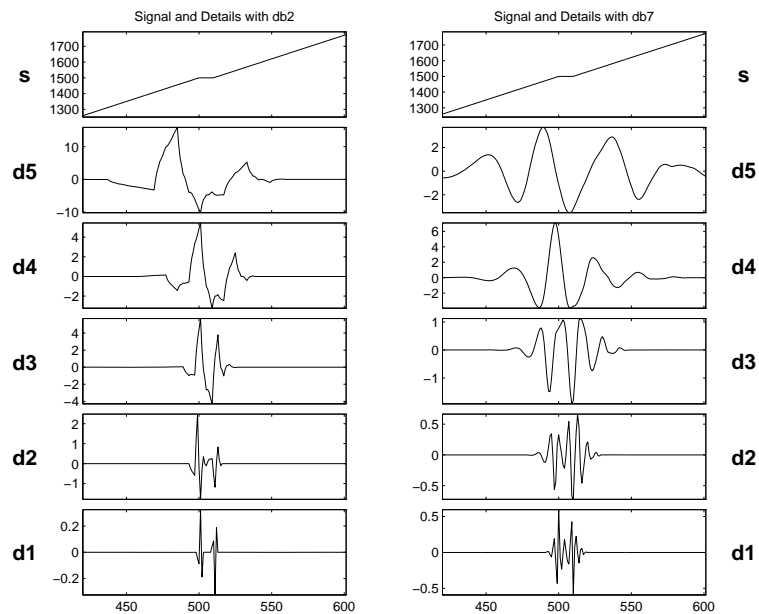
Analyzing wavelet: *db2* and *db7*

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	<ul style="list-style-type: none"> <li>• Detecting breakdown points</li> </ul>
Further exploration	<ul style="list-style-type: none"> <li>• Move the discontinuities closer together and further apart.</li> <li>• Add noise to the signal until the rupture is no longer visible.</li> <li>• Try using other wavelets, haar for instance.</li> </ul>

### 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).

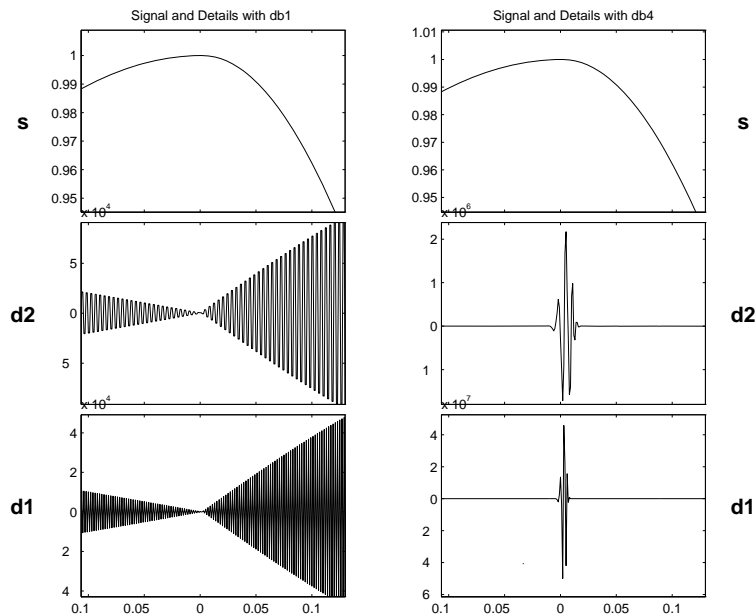
Looking at the figure on the left, notice that the singularity has not been detected in the extent that the details are equal to 0 at 0. The black areas correspond to very rapid oscillations of the details. These values are equal to the difference between the function and an approximation using a constant 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<sup>®</sup> 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	<ul style="list-style-type: none"> <li>• Detecting breakdown points</li> <li>• Suppressing signals</li> <li>• Identifying a difficult discontinuity</li> <li>• Carefully selecting a wavelet to reveal an effect</li> </ul>
Further exploration	<ul style="list-style-type: none"> <li>• Calculate the detail values for the Haar wavelet.</li> <li>• Be aware of parasitic effects: rapid detail fluctuations may be artifacts.</li> <li>• Add noise to the signal until the rupture is no longer visible.</li> </ul>

### Example 9: A Ramp + White Noise

Analyzing wavelet: *db3*

Decomposition levels: 6

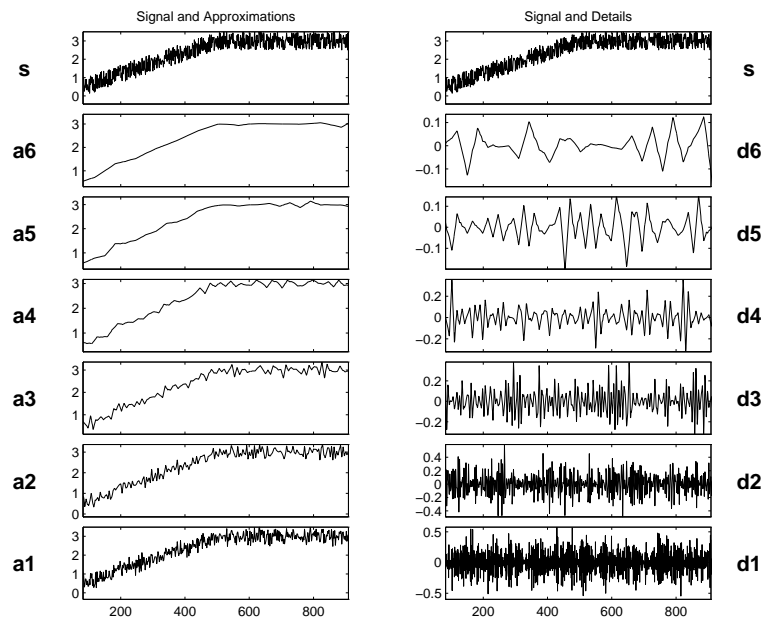
The signal is built from a trend plus noise. The trend is a slow linear rise from 0 to 3, up to  $t=500$ , and becoming constant afterwards. The noise is a uniform zero-mean white noise, varying between -0.5 and 0.5 (see the analyzed signal  $b_1$ ).

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 4-12.
- Identify the number of levels needed to suppress the noise almost entirely.
- Change the noise.

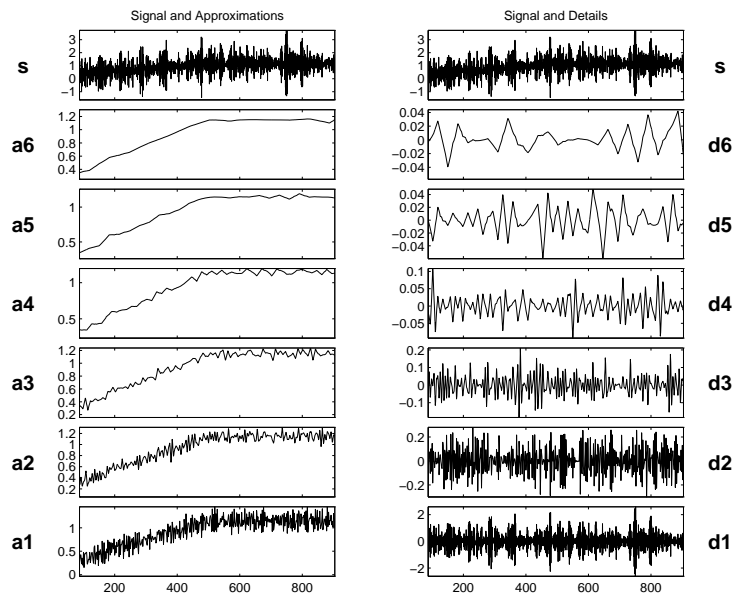
## Example 10: A Ramp + Colored Noise

Analyzing wavelet: *db3*

Decomposition levels: 6

The signal is built in the same manner as in “Example 9: A Ramp + White Noise” on page 4-24, 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	<ul style="list-style-type: none"><li>• Detecting breakdown points</li><li>• Processing noise</li><li>• Detecting long-term evolution</li><li>• Splitting signal components</li></ul>
Further exploration	<ul style="list-style-type: none"><li>• Compare with the <math>s_7(t)</math> signal shown in Example 9: A Ramp + White Noise on page 4-24.</li><li>• Identify the number of levels needed to suppress the noise almost entirely.</li><li>• Identify the noise characteristics. Use the coefficients and the command line mode.</li></ul>

---

## Example 11: A Sine + White Noise

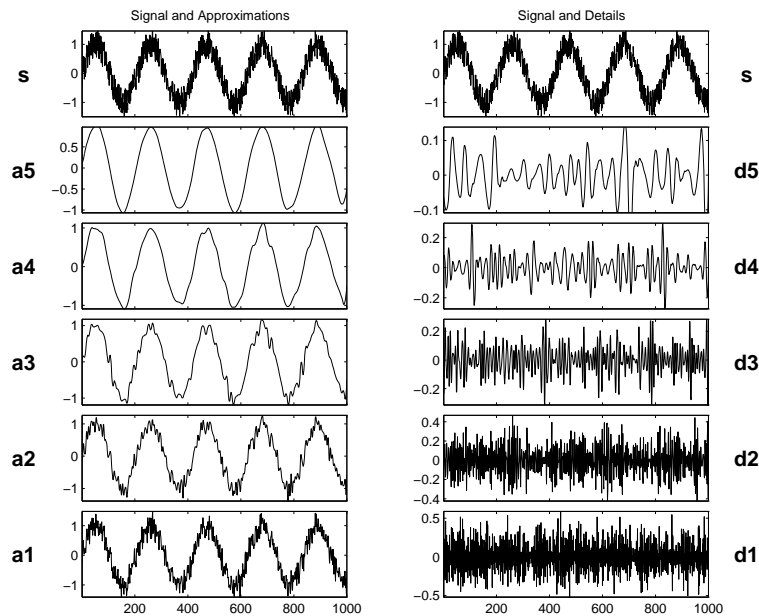
Analyzing wavelet: *db5*

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	<ul style="list-style-type: none"><li>• Processing noise</li><li>• Detecting long-term evolution</li><li>• Splitting signal components</li><li>• Identifying the frequency of a sine</li></ul>
Further exploration	<ul style="list-style-type: none"><li>• Identify the noise characteristics. Use the coefficients and the command line mode.</li></ul>

---

## 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”.

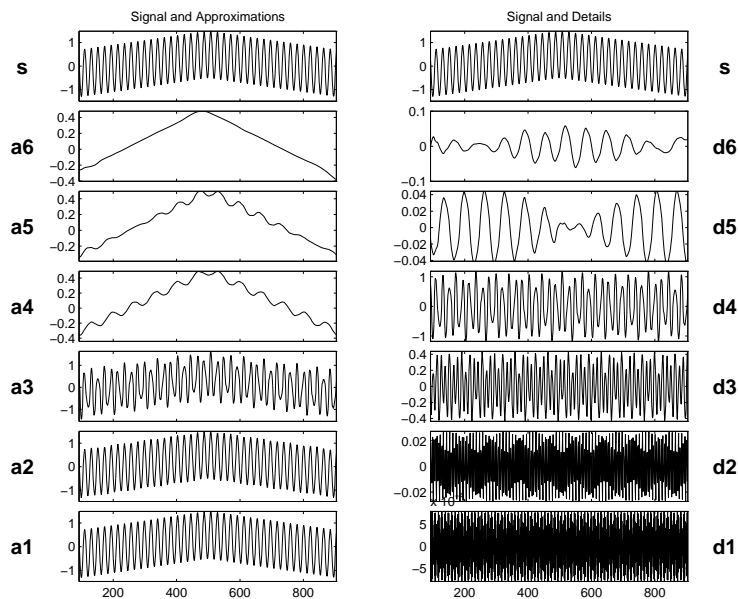
$D_1$  and  $D_2$  are very small. This suggests that the signal contains no components with periods that are short in relation to the sampling period.

$D_3$  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	<ul style="list-style-type: none"><li>• Detecting long-term evolution</li><li>• Splitting signal components</li><li>• Identifying the frequency of a sine</li></ul>
Further exploration	<ul style="list-style-type: none"><li>• Try using sinusoids whose period is a power of 2.</li></ul>

---

### Example 13: A Triangle + A Sine + Noise

Noise Analyzing wavelet: *db5*

Decomposition levels: 7

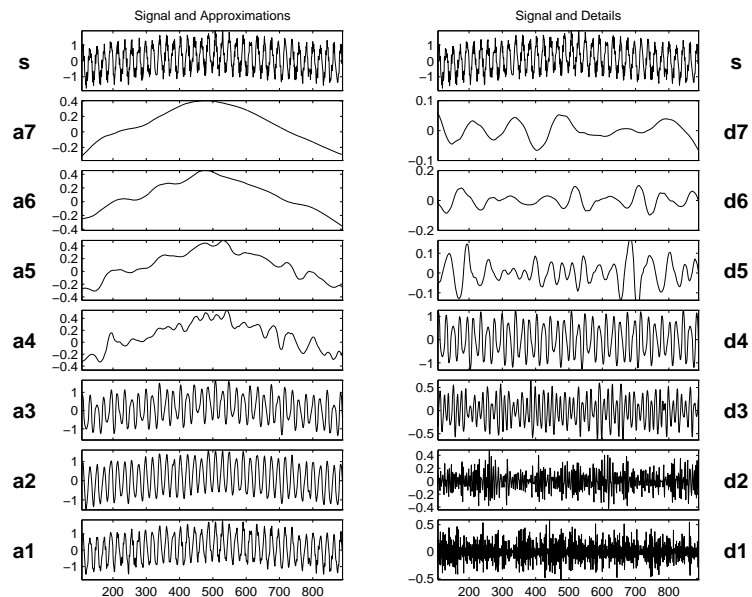
The signal examined here is the same as the previous signal plus a uniform white noise divided by 3. The analysis can, therefore, be compared to the previous analysis. All differences are due to the presence of the noise.

$D_1$  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	<ul style="list-style-type: none"><li>• Detecting long-term evolution</li><li>• Splitting signal components</li></ul>
Further exploration	<ul style="list-style-type: none"><li>• Increase the amplitude of the noise.</li><li>• Replace the triangle by a polynomial.</li><li>• Replace the white noise by an ARMA noise.</li></ul>

---

### **Example 14: A Real Electricity Consumption Signal**

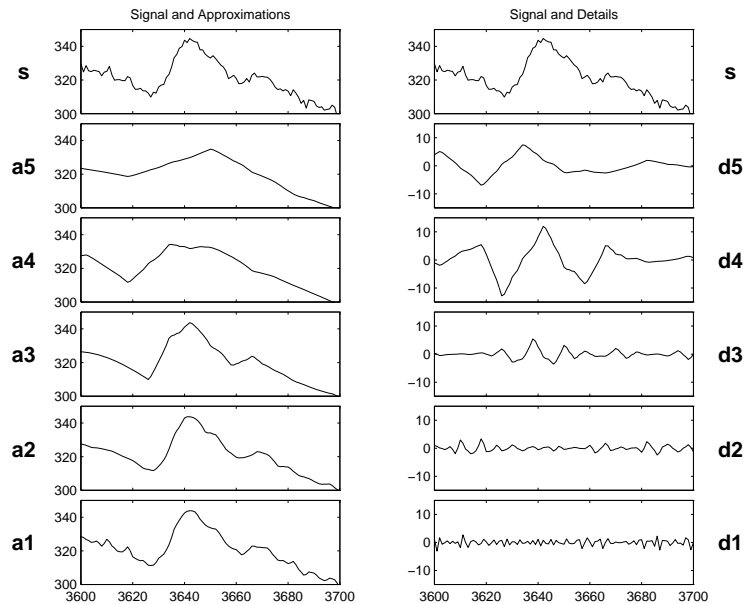
Analyzing wavelet: *db3*

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	<ul style="list-style-type: none"> <li>• Detecting long-term evolution</li> <li>• Splitting signal components</li> <li>• Detecting breakdown points</li> <li>• Multiscale analysis</li> </ul>
Further exploration	<ul style="list-style-type: none"> <li>• Try the same analysis on various sections of the signal. Focus on a range other than the [3600:3700] shown here.</li> </ul>

This signal is explored in much greater detail in “Case Study: An Electrical Signal” on page 4-36.

### 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 4-39.

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 6-155). 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 4-34) is also analyzed more crudely in “Example 14: A Real Electricity Consumption Signal” on page 4-34.

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).

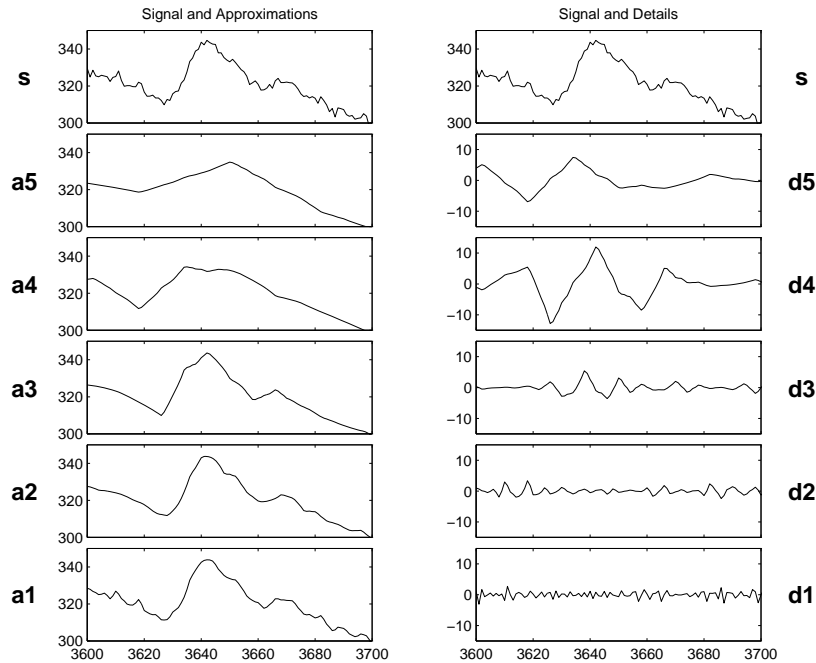
Let us examine the corresponding details.

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

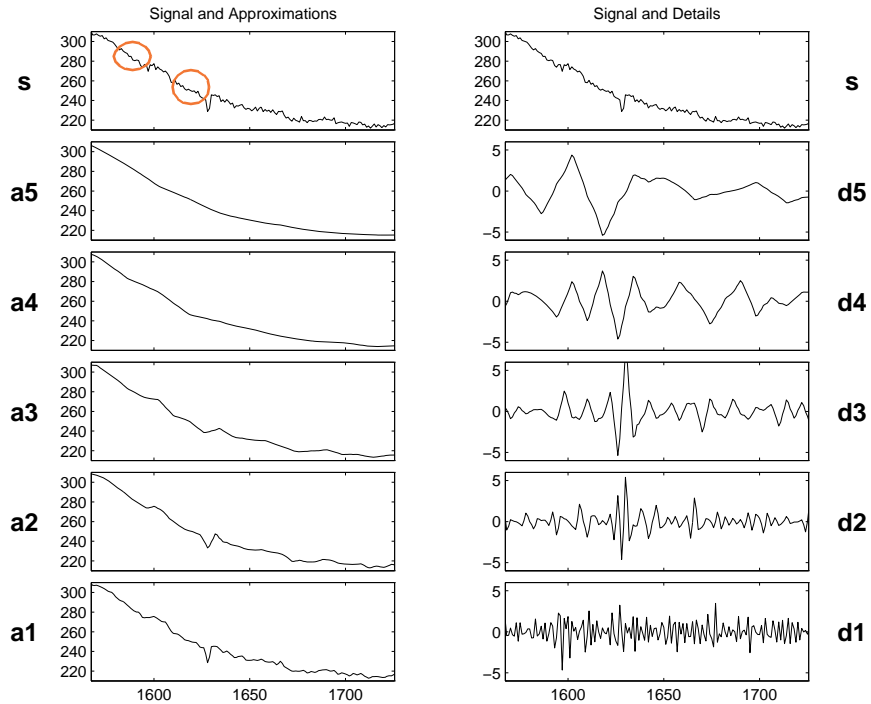
The shape of the curve during the end of the night is a slow descent, globally smooth, but locally highly irregular. One can hardly distinguish two successive local extrema in the vicinity of time  $t = 1600$  and  $t = 1625$ . The approximation  $A_5$  is quite good except at these two modes.

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 in
% original time 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

```

---

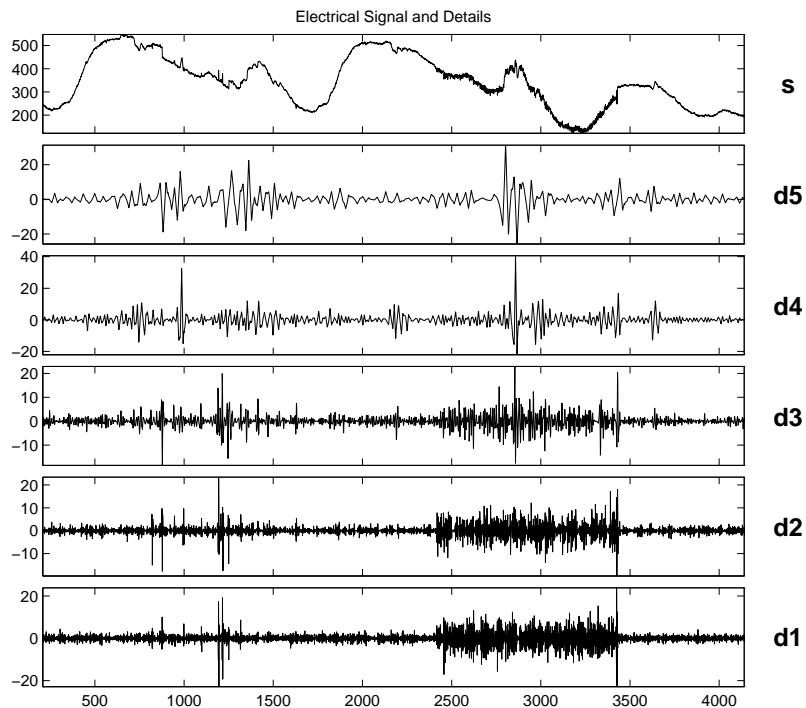
**Note** This loop replaces five separate `wrcoef` statements defining the details. The variable `D` contains the five details.

---

```

% 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

```



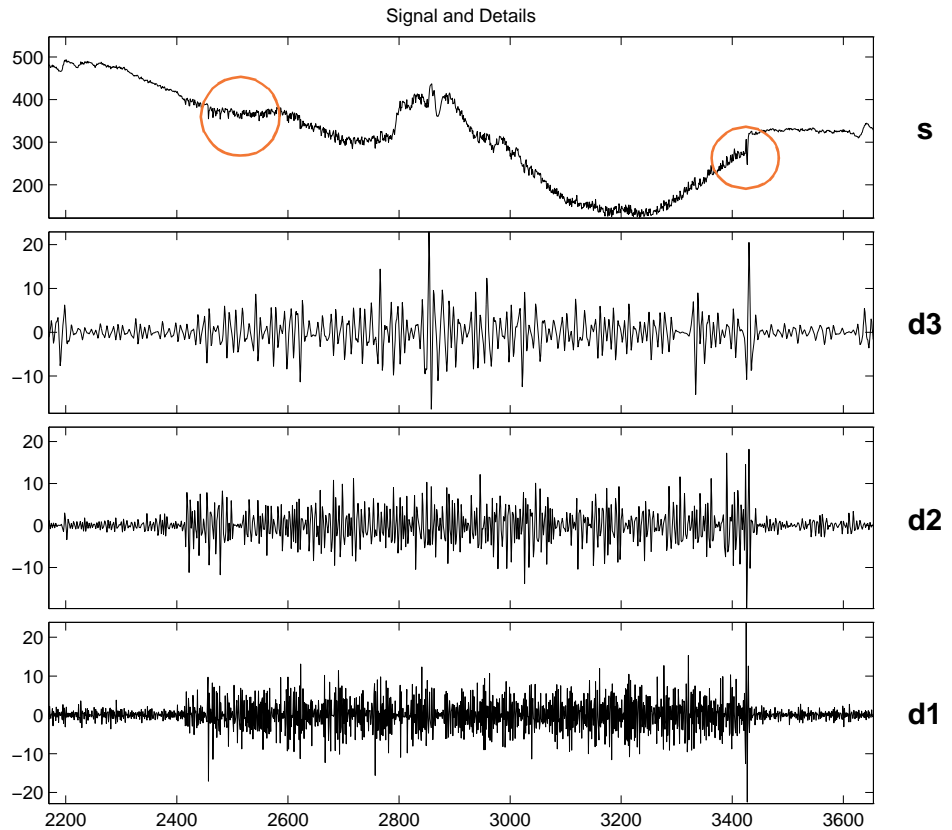
## Suggestions for Further Analysis

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.

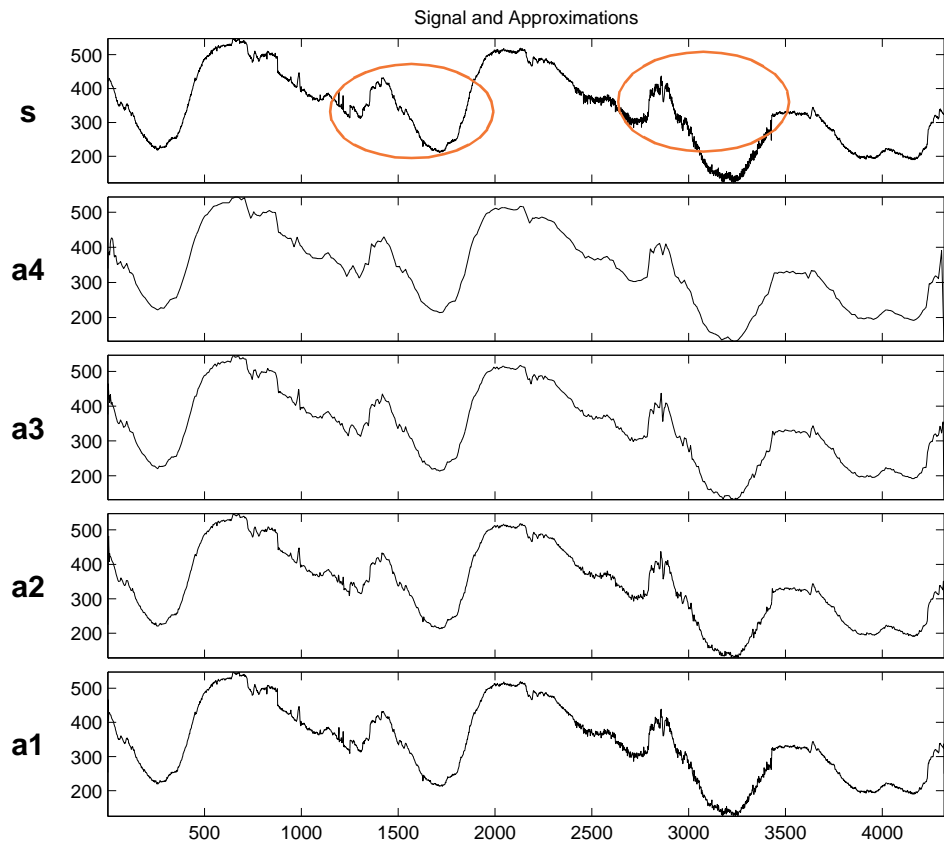
Indication: see figure below.



## Suppress the Noise

Suppress measurement noise. Try by yourself and afterwards use the de-noising tools.

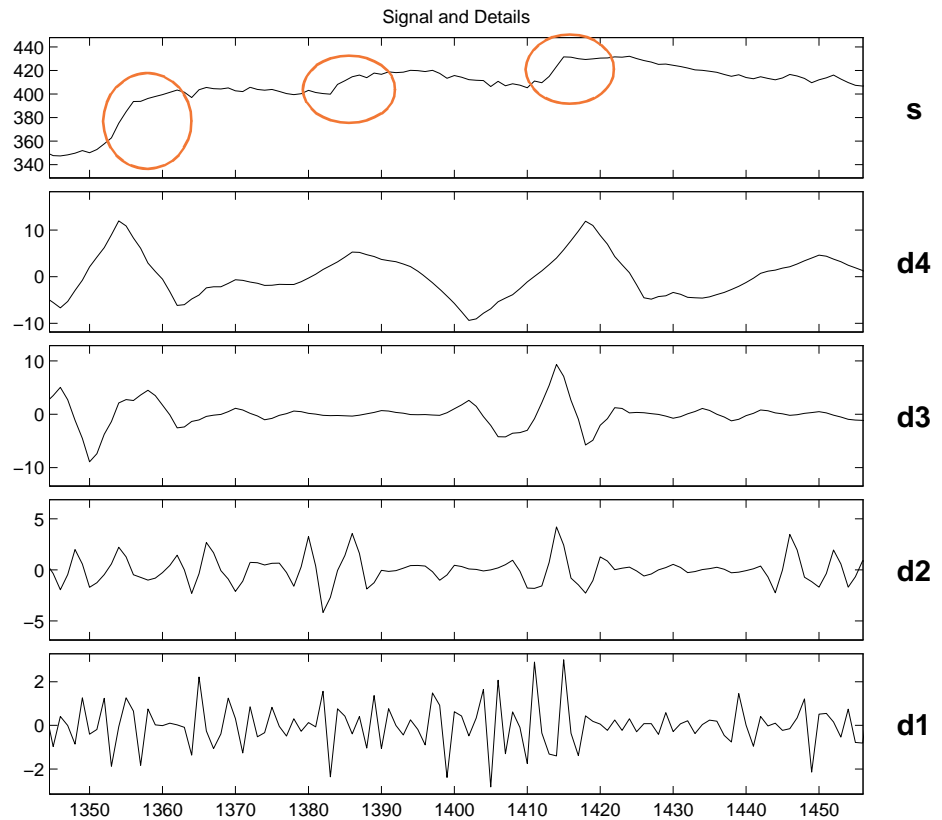
Indication: study the approximations and compare two successive days, the first without sensor failure and the second corrupted by failure (see figure below).



### Identify Patterns in the Details

The idea here is to identify a pattern in the details typical of relay-switched water heaters.

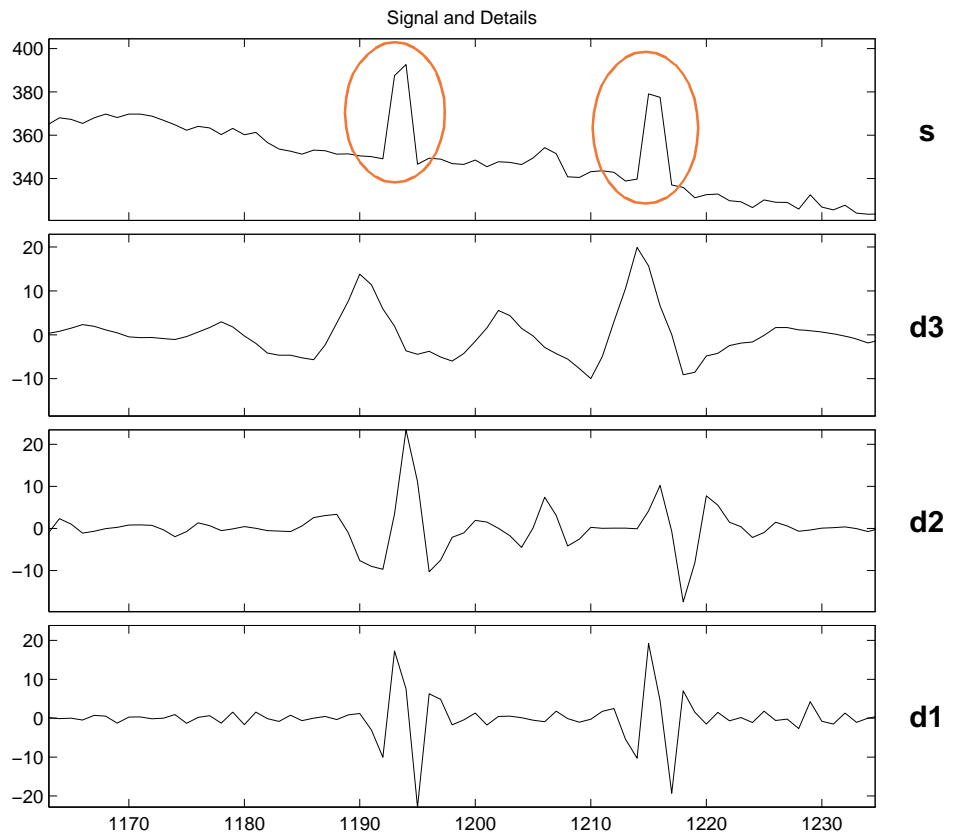
Indication: the figure below gives an example of such a period. Focus on details  $D_2$ ,  $D_3$ , and  $D_4$  around abscissa 1350, 1383, and 1415 to detect abrupt changes of the signal induced by automatic switches.



## Locate and Suppress Outlying Values

Suppress the outliers by setting the corresponding values of the details to 0.

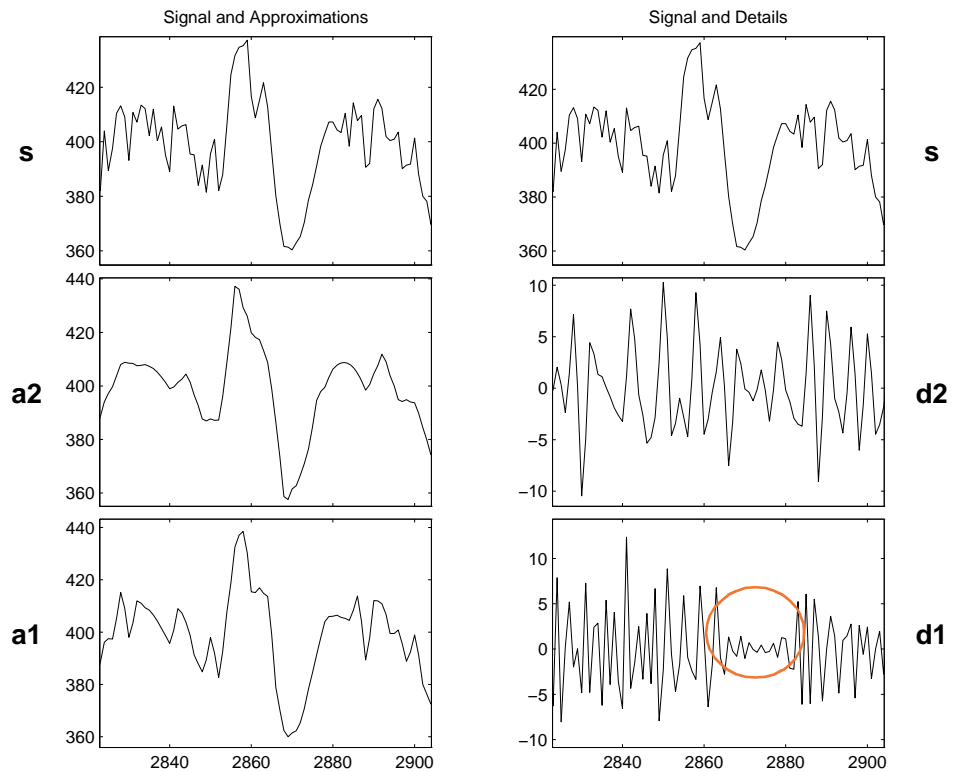
Indication: The figure below gives two examples of outliers around  $t = 1193$  and  $t = 1215$ . The effect produced on the details is clear when focusing on the low levels. As far as outliers are concerned,  $D_1$  and  $D_2$  are synchronized with  $s$ , while  $D_3$  shows a delayed effect.



### 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.



# Using Wavelet Packets

---

- About Wavelet Packet Analysis (p. 5-2)
- One-Dimensional Wavelet Packet Analysis (p. 5-7)
- Two-Dimensional Wavelet Packet Analysis (p. 5-23)
- Importing and Exporting from Graphical Tools (p. 5-32)

# 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 6-136.

---

Some object-oriented programming features are used for wavelet packet tree structures. For more detail, refer to “Object-Oriented Programming” on page B-1.

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
wpccoef	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

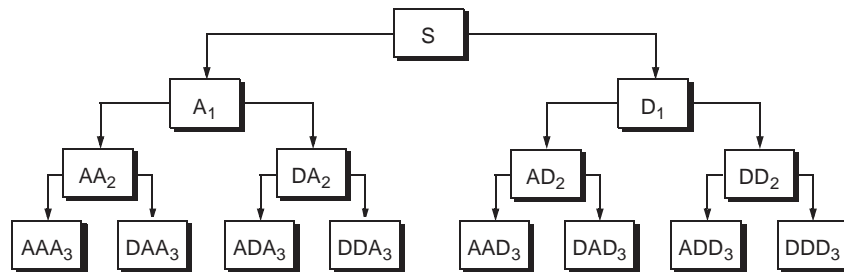
**Decomposition Structure Utilities.**

<b>Function Name</b>	<b>Purpose</b>
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.**

<b>Function Name</b>	<b>Purpose</b>
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.

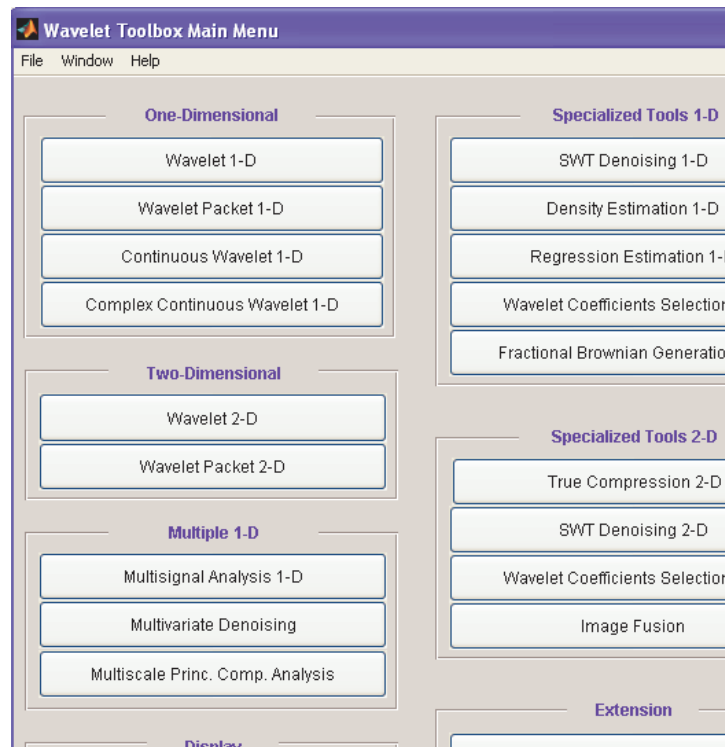
# One-Dimensional Wavelet Packet Analysis

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.

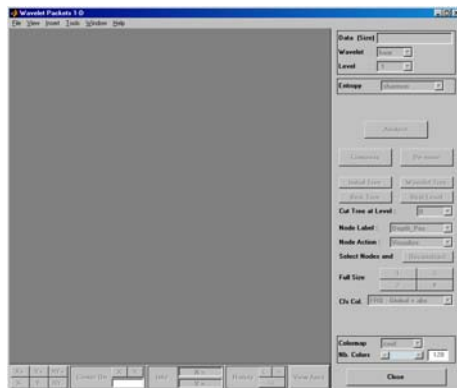
- 1 From the MATLAB<sup>®</sup> prompt, type `wavemenu`.

The **Wavelet Toolbox Main Menu** appears.



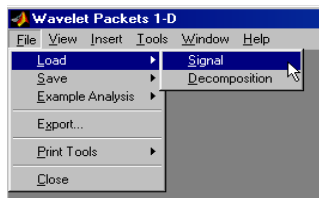
Click the **Wavelet Packet 1-D** menu item.

The tool appears on the desktop.

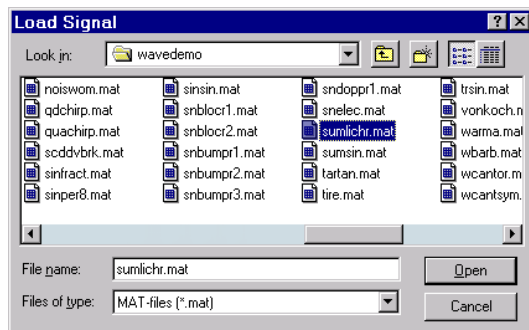


**Loading a Signal.**

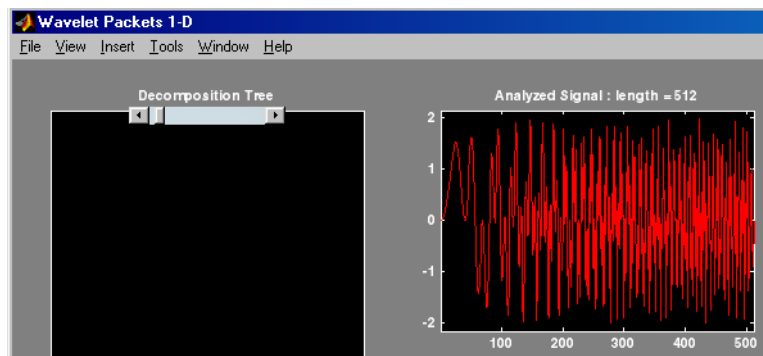
**2** From the **File** menu, choose the **Load Signal** option.



**3** When the **Load Signal** dialog box appears, select the demo MAT-file `sumlichr.mat`, which should reside in the MATLAB directory `toolbox/wavelet/wavedemo`. Click the **OK** button.

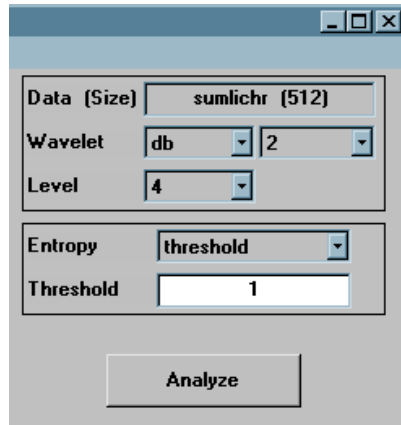


The `sum1ichr` signal is loaded into the **Wavelet Packet 1-D** tool.



**Analyzing a Signal.**

- 4 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 $\epsilon$ .
Norm	The concentration in $l^p$ norm with $1 \leq p$ .
Log Energy	The logarithm of “energy,” defined as the sum over all samples: $\sum \log(s_i^2)$ .



Type	Description
SURE (Stein's Unbiased Risk Estimate)	A threshold-based method in which the threshold equals $\sqrt{2\log_e(n\log_2(n))}$ where $n$ is the number of samples in the signal.
User	An entropy type criterion you define in an M-file.

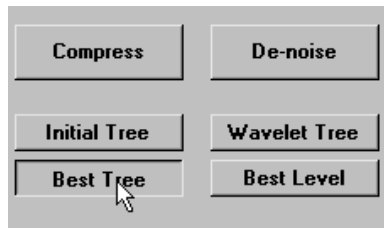
For more information about the available entropy types, user-defined entropy, and threshold parameters, see the `wentropy` reference page and “Choosing the Optimal Decomposition” on page 6-147.

**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-22.

### Computing the Best Tree.

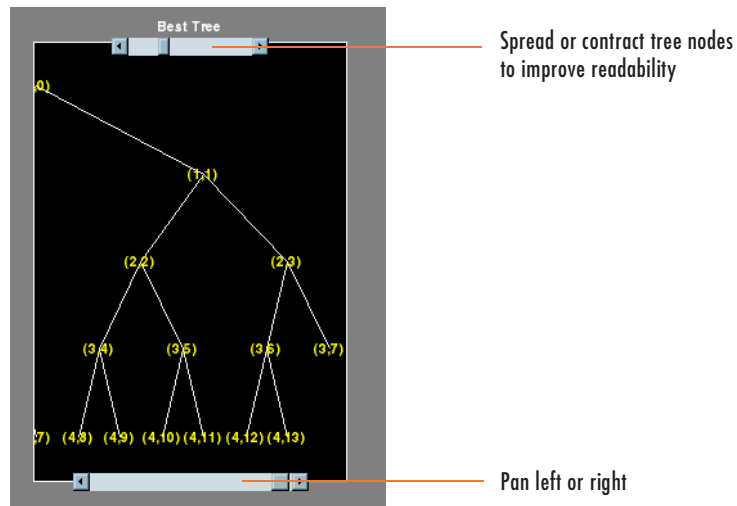
Because there are so many ways to reconstruct the original signal from the wavelet packet decomposition tree, we select the best tree before attempting to compress the signal.

**5** 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.

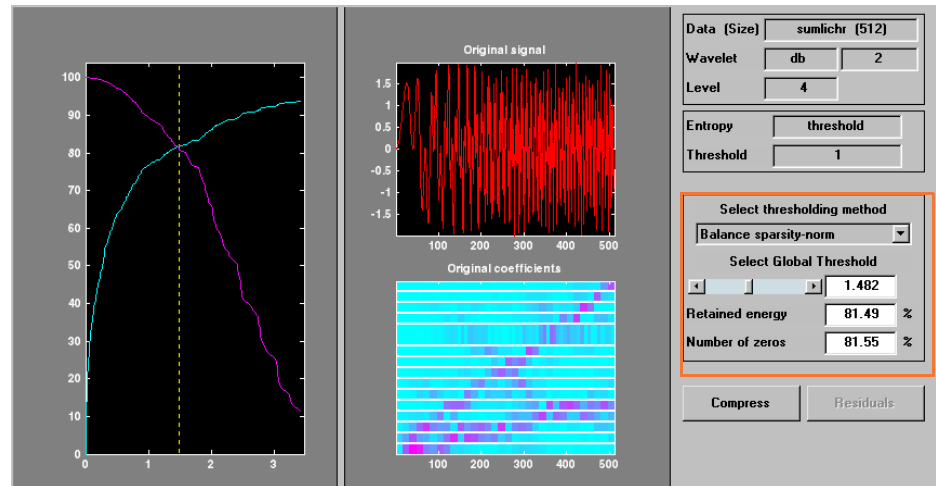


## Compressing a Signal Using Wavelet Packets

### 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

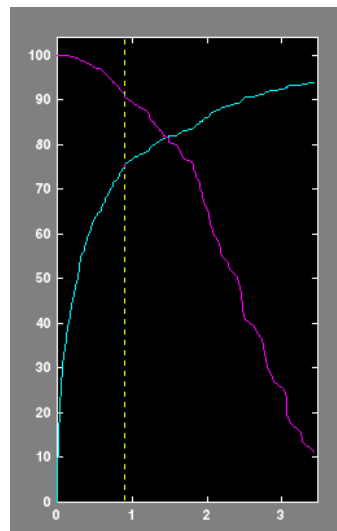
design criteria, you may want to choose a threshold that retains more of the original signal's energy.

- 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.



Select Global Threshold		
	0.8938	
Retained energy	90.96	%
Number of zeros	75.28	%

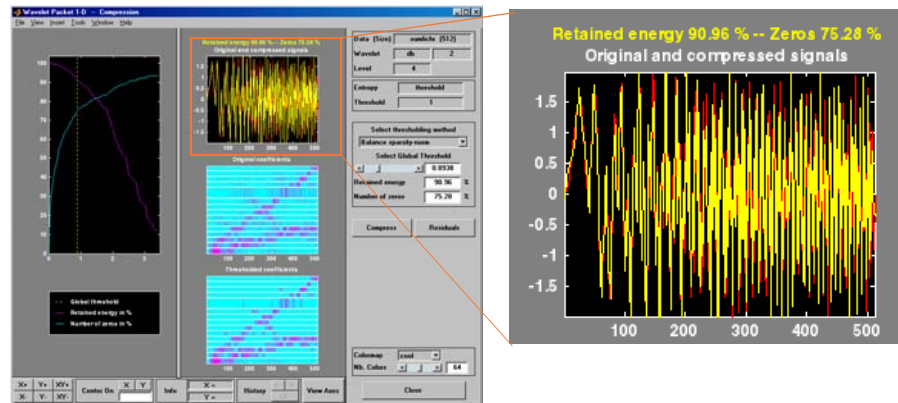
Note that, as we have *reduced* the threshold from 1.482 to 0.8938,

- 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.

#### 3 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

$$T = \sqrt{2 \log_e(n \log_2(n))}$$

where  $n$  is the length of the signal.

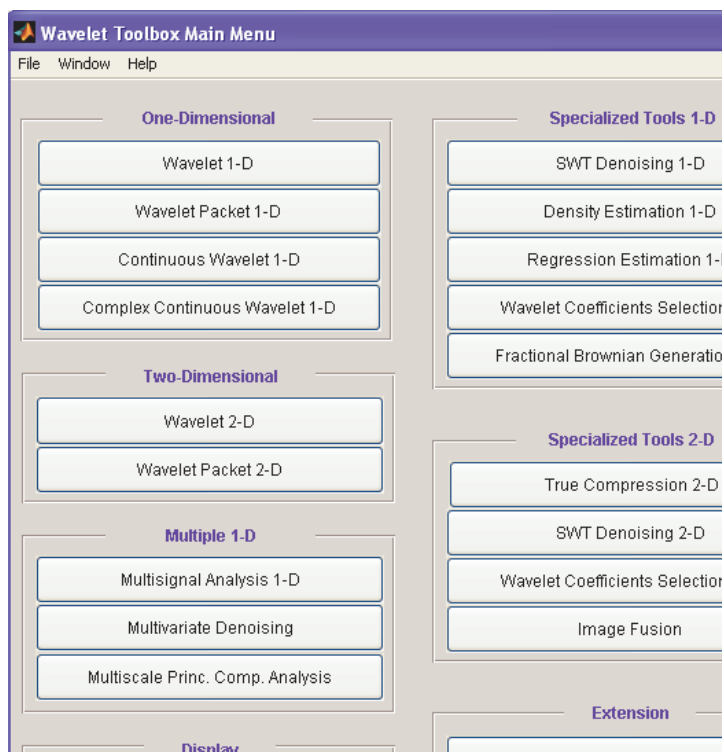
A more thorough discussion of the SURE criterion appears in "Choosing the Optimal Decomposition" on page 6-147. For now, suffice it to say that this method works well if your signal is normalized in such a way that the data fit the model  $x(t) = f(t) + e(t)$ , where  $e(t)$  is a Gaussian white noise 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.

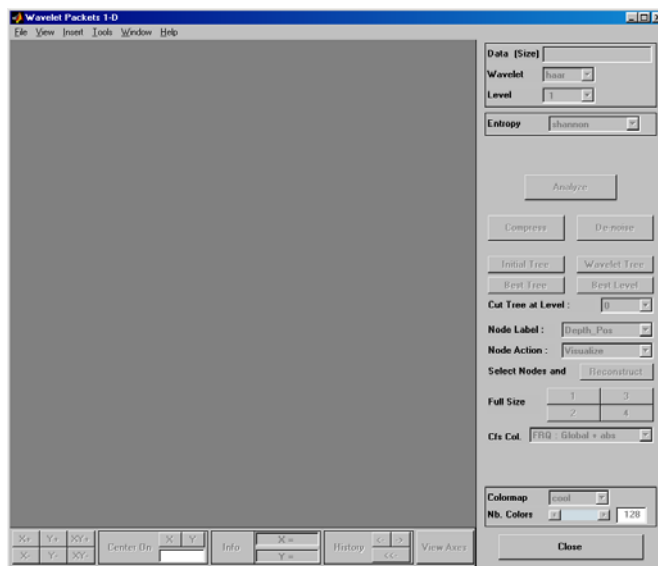
- 1 From the MATLAB prompt, type `wavemenu`.

The **Wavelet Toolbox Main Menu** appears.



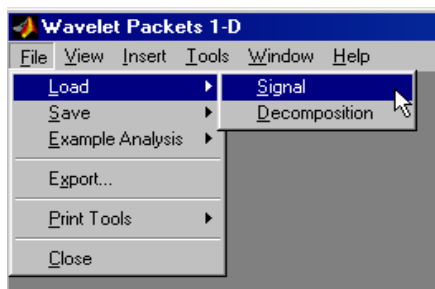
Click the **Wavelet Packet 1-D** menu item.

The tool appears on the desktop.



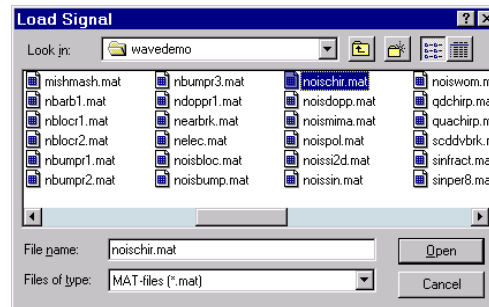
## Loading a Signal.

- 2 From the **File** menu, choose the **Load Signal** option.

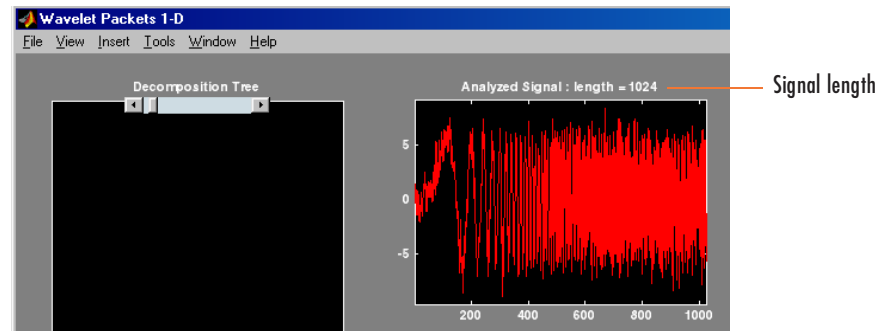


- 3 When the **Load Signal** dialog box appears, select the demo MAT-file `noisichir.mat`, which should reside in the MATLAB directory `toolbox/wavelet/wavedemo`. Click the **OK** button.



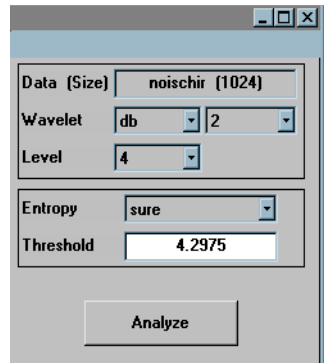


The noischir signal is loaded into the **Wavelet Packet 1-D** tool. Notice that the signal's length is 1024. This means we should set the SURE criterion threshold equal to  $\sqrt{2 \cdot \log(1024 \cdot \log_2(1024))}$ , or 4.2975.



#### 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.

---

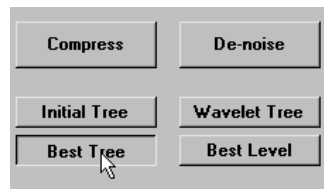
**Note** Many capabilities are available using the command area on the right of the **Wavelet Packet 1-D** window. Some of them are used in the sequel. For a more complete description, see “Wavelet Packet Tool Features (1-D and 2-D)” on page A-22.

---

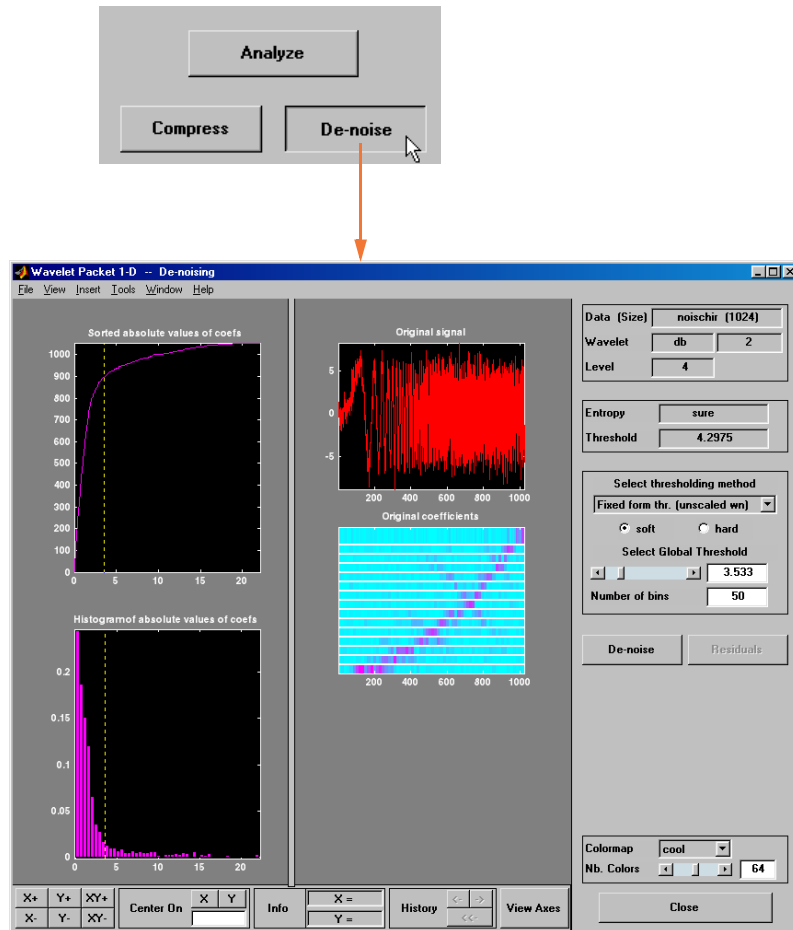
### Computing the Best Tree and Performing De-Noising.

5 Click the **Best Tree** button.

Computing the best tree makes the de-noising calculations more efficient.

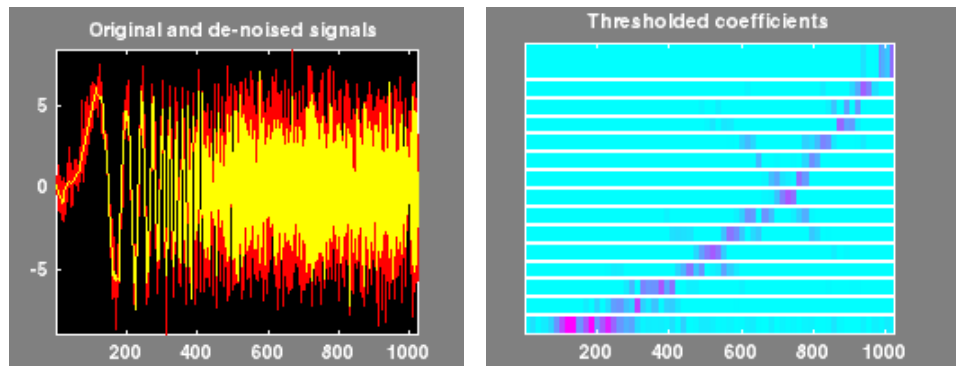


- 6 Click the **De-noise** button. This brings up the **Wavelet Packet 1-D De-Noising** window.



- 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 de-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 6-155).

“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 6-155).

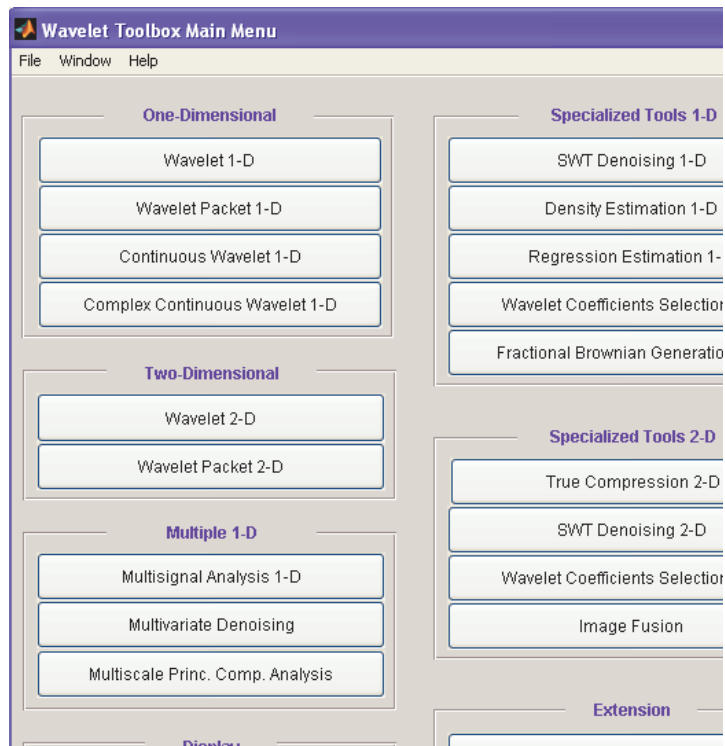
By turning to wavelets, the FBI has achieved a 15:1 compression ratio. In this application, wavelet compression is better than the more traditional JPEG compression, as it avoids small square artifacts and is particularly well suited to detect discontinuities (lines) in the fingerprint.

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.

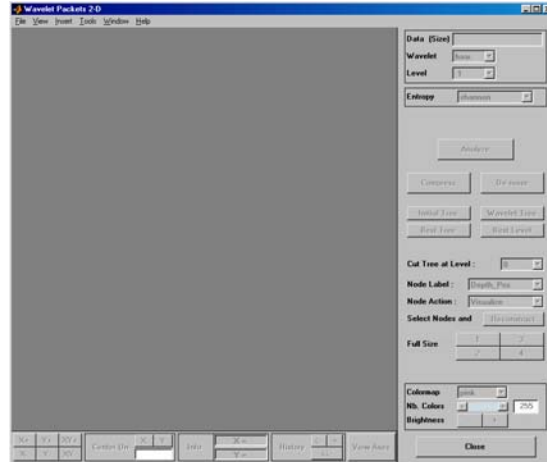
- 1 From the MATLAB® prompt, type  
wavemenu

The **Wavelet Toolbox Main Menu** appears.



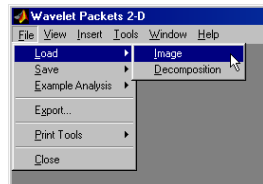
Click the **Wavelet Packet 2-D** menu item.

The tool appears on the desktop.

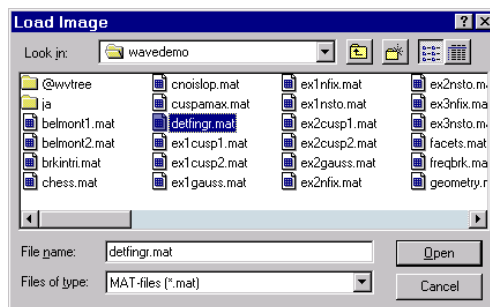


### Loading an Image.

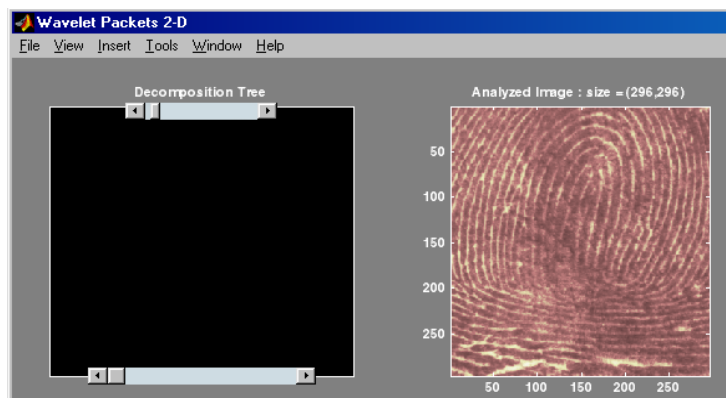
From the **File** menu, choose the **Load Image** option.



- 2 When the **Load Image** dialog box appears, select the demo MAT-file `detfingr.mat`, which should reside in the MATLAB directory `toolbox/wavelet/wavedemo`. Click the **OK** button.



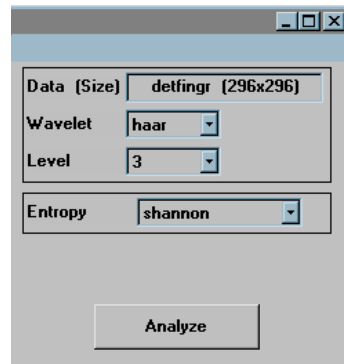
The fingerprint image is loaded into the **Wavelet Packet 2-D** tool.



### 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.





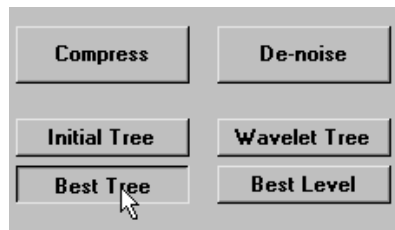
There is a pause while the wavelet packet analysis is computed.

---

**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-22.

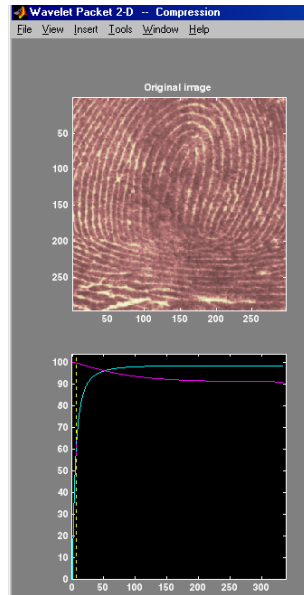
---

- 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.



The dialog box shows the 'Select thresholding method' dropdown menu set to 'Bal. sparsity-norm (sqrt)'. Below it, the 'Select Global Threshold' slider is set to 7.125. The 'Retained energy' is 99.78% and the 'Number of zeros' is 63.90%.

Select thresholding method	
Bal. sparsity-norm (sqrt)	
Select Global Threshold	
7.125	
Retained energy	99.78 %
Number of zeros	63.90 %

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.

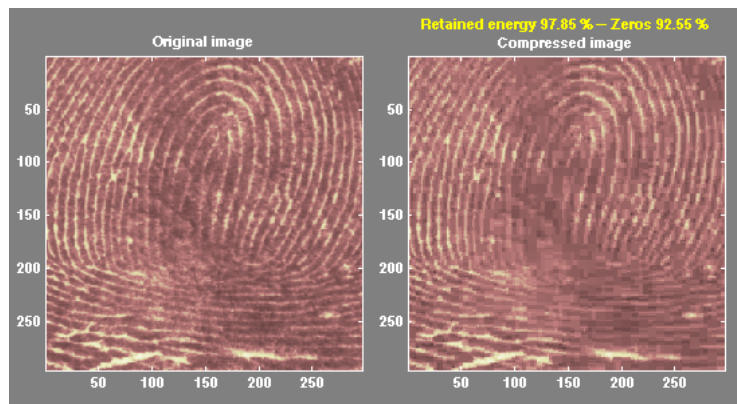
- 2 Alter the threshold: type the number 30 in the text field opposite the threshold slider located on the right side of the **Wavelet Packet 2-D Compression** window. Then press the **Enter** key.

Select Global Threshold		
	30	
Retained energy	97.85	%
Number of zeros	92.55	%

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 3-26.

- 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.

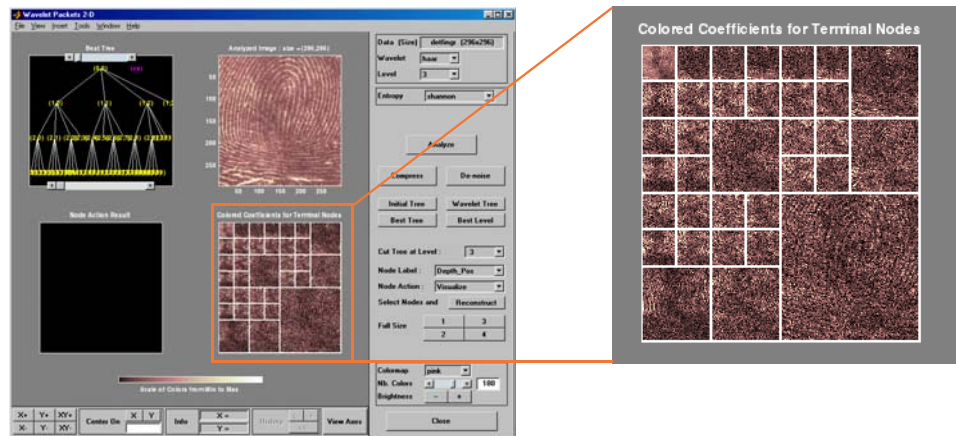


- 4 Click the **Close** button located at the bottom of the **Wavelet Packet 2-D Compression** window. Update the synthesized image by clicking **Yes** when the dialog box appears.

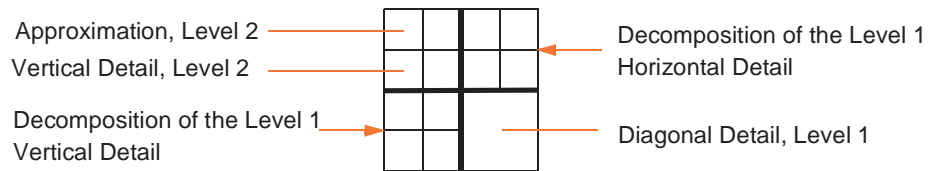
Take this opportunity to try out your own compression strategy. Adjust the threshold value, the entropy function, and the wavelet, and see if you can obtain better results.

**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 6-113.

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 directory and filename for the MAT-file. For this example, choose the name `synthsig`.

To load the signal into your workspace, simply type

```
load synthsig
whos
```

Name	Size	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 directory and filename for the MAT-file. For this example, choose the name `wpsymage`.

To load the image into your workspace, simply type

```
load wpsymage
whos
```

Name	Size	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 directory with a name you choose, followed by the extension wp1 (wavelet packet 1-D).

Open the **Wavelet Packet 1-D** tool and load the 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 directory 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
```



<b>Name</b>	<b>Size</b>	<b>Bytes</b>	<b>Class</b>
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`:

<b>Name</b>	<b>Size</b>	<b>Bytes</b>	<b>Class</b>
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<sup>®</sup> 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:

<b>Variable</b>	<b>Status</b>	<b>Description</b>
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.

---



# 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 (p. 6-2)
- General Concepts (p. 6-5)
- Fast Wavelet Transform (FWT) Algorithm (p. 6-19)
- Dealing with Border Distortion (p. 6-35)
- Discrete Stationary Wavelet Transform (SWT) (p. 6-44)
- Lifting Method for Constructing Wavelets (p. 6-51)
- Frequently Asked Questions (p. 6-61)
- Wavelet Families: Additional Discussion (p. 6-71)
- Wavelet Applications: More Detail (p. 6-92)
- Wavelet Packets (p. 6-136)
- References (p. 6-155)

## 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, $1/a$ or $2^{-j}$ is the resolution.
$b = ka, k \in Z$	Dyadic translation
$t$	Continuous time
$k$ or $n$	Discrete time
$(i, j)$	Pixel
$s$	Signal or image. The signal is a function defined on $R$ or $Z$ ; the image is defined on $R^2$ or $Z^2$ .
$\hat{f}$	Fourier transform of the function $f$ or the sequence $f$
<b>Continuous Time</b>	
$L^2(R)$	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
$\int_R \int_R s^2(x, y) dx dy$	Energy of the image $s$
$\langle s, s' \rangle = \int_R \int_R s(x, y)s'(x, y) dx dy$	Scalar product of images $s$ and $s'$



General Notation	Interpretation (Continued)
<b>Discrete Time</b>	
$l^2(\mathbb{Z})$	Set of signals of finite energy
$\sum_{n \in \mathbb{Z}} s^2(n)$	Energy of the signal $s$
$\langle s, s' \rangle = \sum_{n \in \mathbb{Z}} s(n)s'(n)$	Scalar product of signals $s$ and $s'$
$l^2(\mathbb{Z}^2)$	Set of images of finite energy
$\sum_{n \in \mathbb{Z}} \sum_{m \in \mathbb{Z}} s^2(n, m)$	Energy of the image $s$
$\langle s, s' \rangle = \sum_{n \in \mathbb{Z}} \sum_{m \in \mathbb{Z}} s(n, m)s'(n, m)$	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 $j$
$\phi$	Scaling function
$\psi$	Wavelet
$\frac{1}{\sqrt{a}} \psi\left(\frac{x-b}{a}\right)$	Family associated with the one-dimensional wavelet, indexed by $a > 0$ and $b \in \mathbb{R}$
$\frac{1}{\sqrt{a_1 a_2}} \psi\left(\frac{x_1 - b_1}{a_1}, \frac{x_2 - b_2}{a_2}\right), x = (x_1, x_2) \in \mathbb{R}^2$	Family associated with the two-dimensional wavelet, indexed by $a_1 > 0, a_2 > 0, b_1 \in \mathbb{R}, b_2 \in \mathbb{R}$

Wavelet Notation	Interpretation (Continued)
$\phi_{j,k}(x) = 2^{-j/2} \phi(2^{-j}x - k), j \in \mathbb{Z}, k \in \mathbb{Z}$	Family associated with the one-dimensional scaling function for dyadic scales $a = 2^j, b = ka$ ; it should be noted that $\phi = \phi_{0,0}$ .
$\psi_{j,k}(x) = 2^{-j/2} \psi(2^{-j}x - k), j \in \mathbb{Z}, k \in \mathbb{Z}$	Family associated with the one-dimensional $\psi$ for dyadic scales $a = 2^j, b = ka$ ; it should be noted that $\psi = \psi_{0,0}$ .
$(h_k), k \in \mathbb{Z}$	Scaling filter associated with a wavelet
$(g_k), k \in \mathbb{Z}$	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” on page 6-5
- “Wavelet Decomposition: A Hierarchical Organization” on page 6-5
- “Finer and Coarser Resolutions” on page 6-6
- “Wavelet Shapes” on page 6-6
- “Wavelets and Associated Families” on page 6-8
- “Wavelet Transforms: Continuous and Discrete” on page 6-12
- “Local and Global Analysis” on page 6-14
- “Synthesis: An Inverse Transform” on page 6-15
- “Details and Approximations” on page 6-15

### 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” on page 1-30, 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  $A_1$  and intermediate details  $D_1, D_2$ :

$$A_2 = A_1 + D_2 = A_1 + D_1 + D_2$$

### 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.

<b><math>j</math></b>	10	9	...	2	1	0	-1	-2
<b>Scale</b>	1024	512	...	4	2	1	1/2	1/4
<b>Resolution</b>	$1/2^{10}$	$1/2^9$	...	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

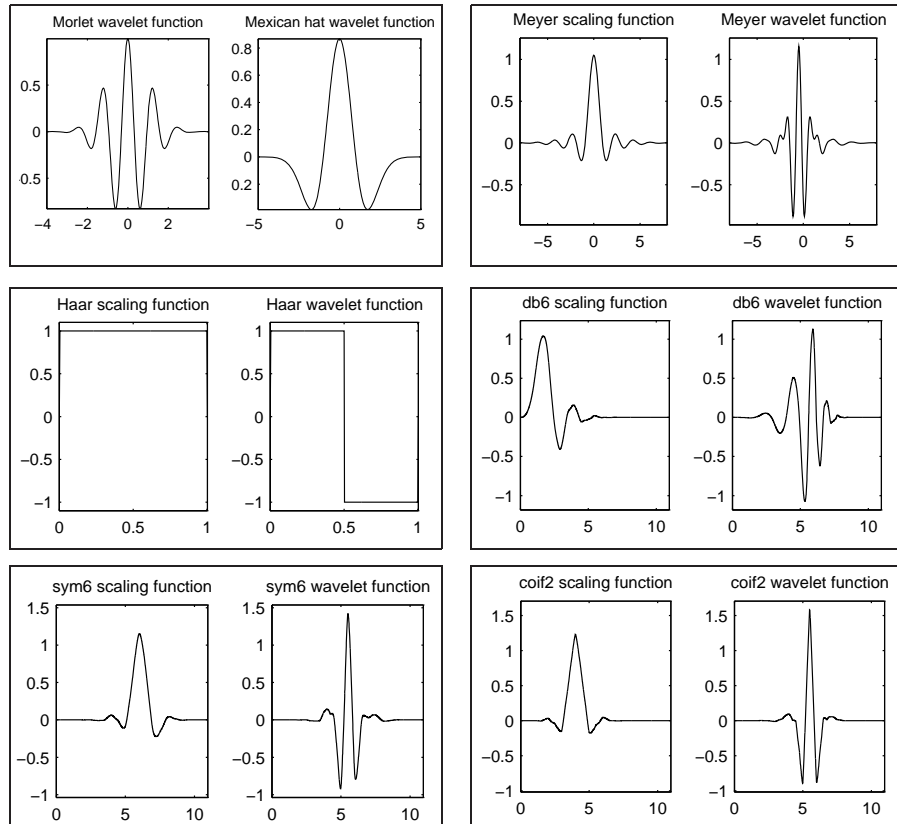
function of the variable  $x$ ,  $\psi\left(\frac{x}{a}\right)$  is not too close to 0.

### Wavelet Shapes

One-dimensional analysis is based on one scaling function  $\phi$  and one wavelet  $\psi$ . Two-dimensional analysis (on a square or rectangular grid) is based on one scaling function  $\phi(x_1, x_2)$  and three wavelets.

Figure 6-1 shows  $\phi$  and  $\psi$  for each wavelet, except the Morlet wavelet and the Mexican hat, for which  $\phi$  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  $\psi$  functions oscillate more than associated  $\phi$

functions. `coif2` exhibits some angular points; `db6` and `sym6` are quite smooth. The Morlet and Mexican hat wavelets are symmetrical.



**Figure 6-1: Various One-Dimensional Wavelets**

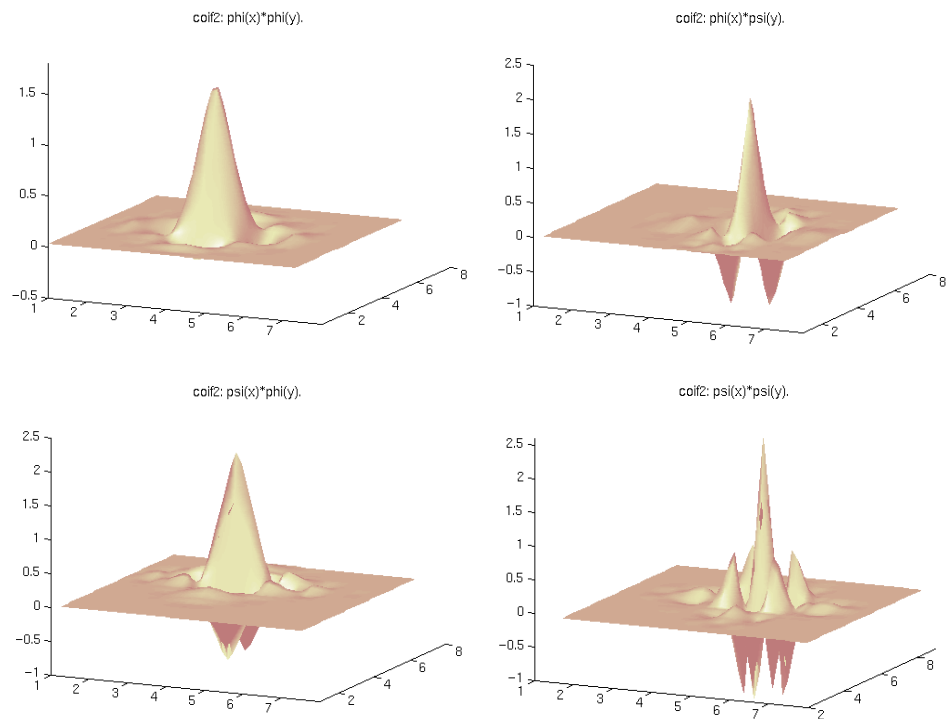
## Wavelets and Associated Families

In the one-dimensional context, we distinguish the wavelet  $\psi$  from the associated function  $\phi$ , called the scaling function. Some properties of  $\psi$  and  $\phi$  are

- The integral of  $\psi$  is zero, ( $\int \psi(x)dx = 0$ ), and  $\psi$  is used to define the details.
- The integral of  $\phi$  is 1, ( $\int \phi(x)dx = 1$ ), and  $\phi$  is used to define approximations.

The usual two-dimensional wavelets are defined as tensor products of one-dimensional wavelets:  $\phi(x, y) = \phi(x)\phi(y)$  is the scaling function and  $\psi_1(x, y) = \phi(x)\psi(y)$ ,  $\psi_2(x, y) = \psi(x)\phi(y)$ ,  $\psi_3(x, y) = \psi(x)\psi(y)$  are the three wavelets.

Figure 6-2 shows the four functions associated with the 2-D `coif2` wavelet.



**Figure 6-2: Two-Dimensional `coif2` Wavelet**

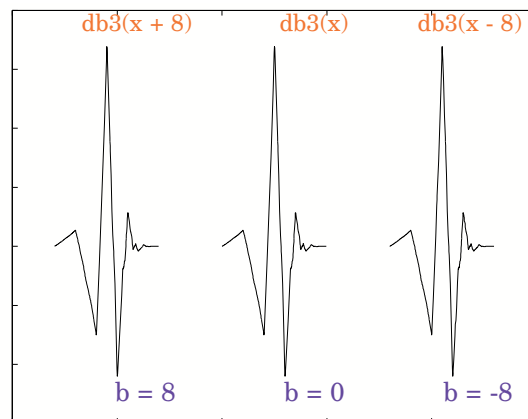
To each of these functions, we associate its doubly indexed family, which is used to:

- Move the 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  $a$  ( $a > 0$ ) (see Figure 6-4 on page 6-10).

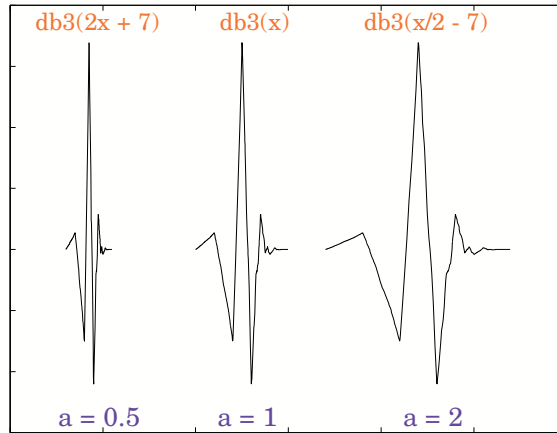
So a wavelet family member has to be thought of as a function located at a position  $b$ , and having a scale  $a$ .

In one-dimensional situations, the family of translated and scaled wavelets associated with  $\psi$  is expressed as follows.

Translation	Change of Scale	Translation and Change of Scale
$\psi(x-b)$	$\frac{1}{\sqrt{a}}\psi\left(\frac{x}{a}\right)$	$\frac{1}{\sqrt{a}}\psi\left(\frac{x-b}{a}\right)$



**Figure 6-3: Translated Wavelets**



**Figure 6-4: 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:

$$\frac{1}{\sqrt{a_1 a_2}} \psi\left(\frac{x_1 - b_1}{a_1}, \frac{x_2 - b_2}{a_2}\right) \text{ where } (x = (x_1, x_2) \in \mathbb{R}^2)$$

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):

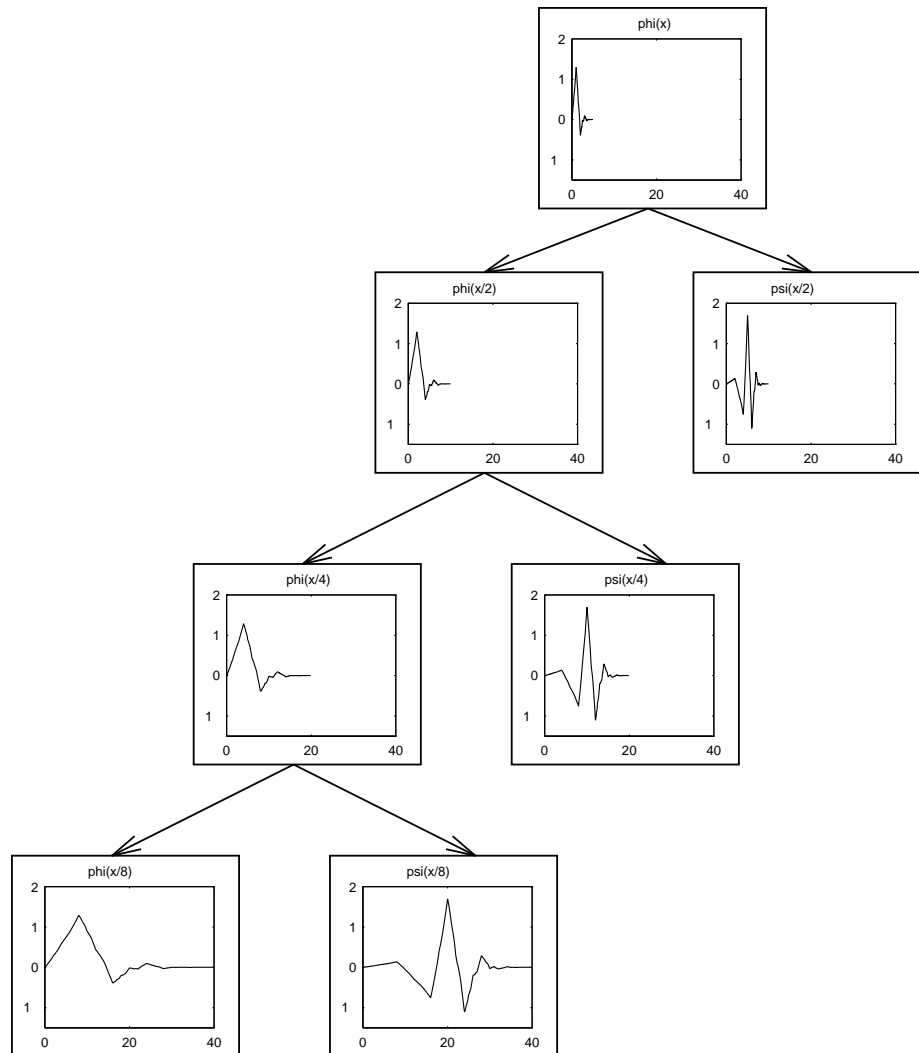
$$(j, k) \in \mathbb{Z}^2 : a = 2^j, \quad b = k2^j = ka$$

Let us define:

$$(j, k) \in \mathbb{Z}^2 : \psi_{j, k} = 2^{-j/2} \psi(2^{-j}x - k), \quad \phi_{j, k} = 2^{-j/2} \phi(2^{-j}x - k)$$

We now have a hierarchical organization similar to the organization of a decomposition; this is represented in the example of Figure 6-5, Wavelets Organization. Let  $k = 0$  and leave the translations aside for the moment. The functions associated with  $j = 0, 1, 2, 3$  for  $\phi$  (expressed as  $\phi_{j,0}$ ) and with  $j = 1, 2, 3$  for  $\psi$  (expressed as  $\psi_{j,0}$ ) are displayed in the following figure for the db3 wavelet.





**Figure 6-5: Wavelets Organization**

In Figure 6-5, Wavelets Organization, the four-level decomposition is shown, progressing from the top to the bottom. We find  $\phi_{0,0}$ ; then  $2^{1/2}\phi_{1,0}$ ,  $2^{1/2}\psi_{1,0}$ ; then  $2\phi_{2,0}$ ,  $2\psi_{2,0}$ ; then  $2^{3/2}\phi_{3,0}$ ,  $2^{3/2}\psi_{3,0}$ . The wavelet is 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:

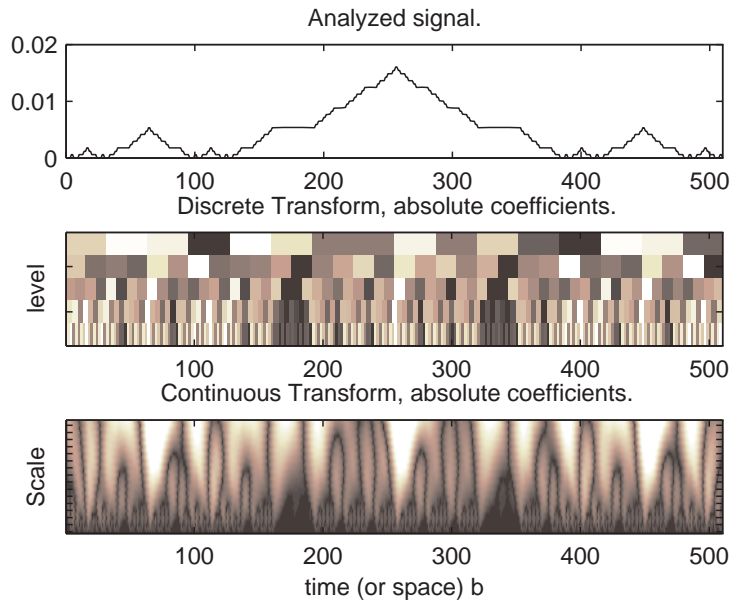
- Continuous transform
- Discrete transform

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 Figure 6-6).



**Figure 6-6: 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 Figure 6-6, Continuous Versus Discrete Transform) 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 Figure 6-6, Continuous Versus Discrete Transform) 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  $\phi$  or  $\psi$  is zero outside of a domain  $U$ , which is contained in a disk of radius  $\rho$ :  $\psi(u) = 0, \forall u \notin U$ . The wavelet  $\psi$  is localized. The signal  $s$  and the function  $\psi$  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

$$\int_R s(t)\psi(t)dt \quad \text{and we get} \quad \int_R s(t)\psi(t)dt = \int_U s(t)\psi(t)dt$$

The same argument holds when  $\psi$  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  $\psi$  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  $\mathbb{R}$ . Its Fourier transform  $\hat{f}$  is a generalized function concentrated at point  $\omega$ .

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  $K_\psi$  is a constant depending on  $\psi$ .

- Discrete synthesis:

$$s(t) = \sum_{j \in \mathbb{Z}} \sum_{k \in \mathbb{Z}} C(j,k) \psi_{j,k}(t).$$

Of course, the previous formulas need some hypotheses on the  $\psi$  function. More precisely, see “What Functions Are Candidates to Be a Wavelet?” on page 6-63 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

$$D_j(t) = \sum_{k \in \mathbb{Z}} C(j,k) \psi_{j,k}(t).$$

- 2 Now, let us sum on  $j$ . The signal is the sum of all the details:

$$s = \sum_{j \in \mathbb{Z}} D_j$$

The details have just been defined. Take a reference level called  $J$ . There are two sorts of details. Those associated with indices  $j \leq J$  correspond to the scales  $\alpha = 2^j \leq 2^J$  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

$$s = A_J + \sum_{j \leq J} D_j$$

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  $D_J, D_{J-1}, D_{J-2}, \dots$
- $s$  is the sum of the two orthogonal signals:  $A_J$  and  $\sum_{j \leq J} D_j$
- $D_j \perp D_k$  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 $J$	$A_J = \sum_{j > J} D_j$
Link between $A_{J-1}$ and $A_J$	$A_{J-1} = A_J + D_J$
Several decompositions	$s = A_J + \sum_{j \leq J} D_j$

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 Figure 6-7. 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.

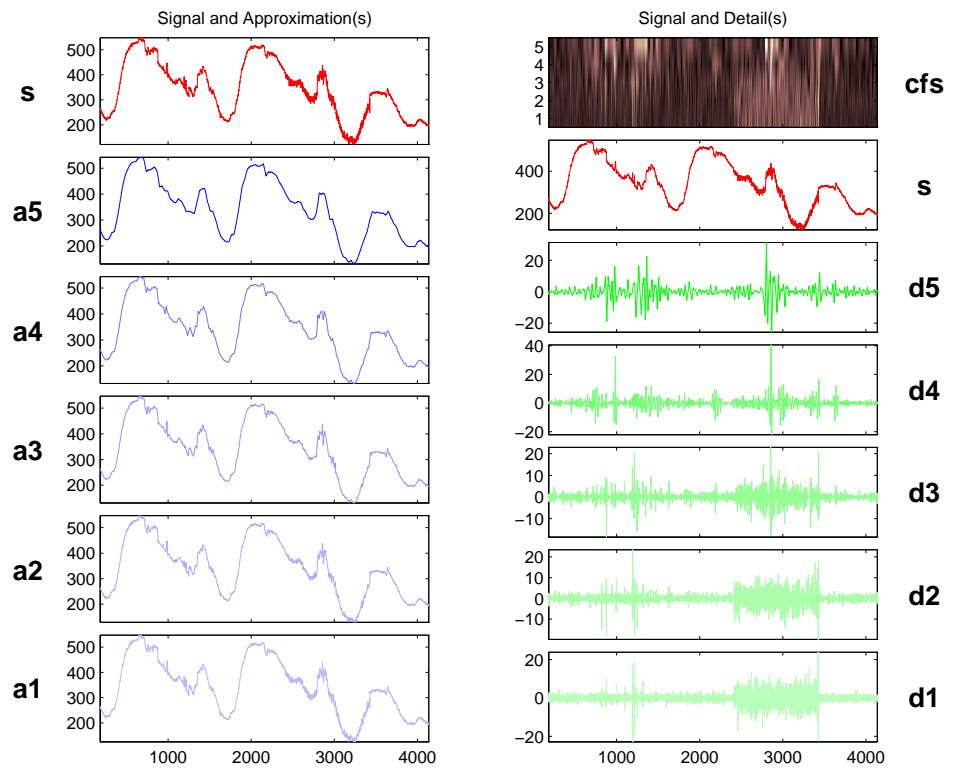


Figure 6-7: Approximations, Details, and Coefficients



## 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”
- “Algorithms” on page 6-23
- “Why Does Such an Algorithm Exist?” on page 6-28
- “One-Dimensional Wavelet Capabilities” on page 6-32
- “Two-Dimensional Wavelet Capabilities” on page 6-33

### Filters Used to Calculate the DWT and IDWT

For an orthogonal wavelet, in the multiresolution framework (see [Dau92] in Chapter 5, “Using Wavelet Packets”), we start with the scaling function  $\phi$  and the wavelet function  $\psi$ . One of the fundamental relations is the twin-scale relation (dilation equation or refinement equation):

$$\frac{1}{2}\phi\left(\frac{x}{2}\right) = \sum_{n \in \mathbb{Z}} w_n \phi(x - n)$$

All the filters used in DWT and IDWT are intimately related to the sequence

$$(w_n)_{n \in \mathbb{Z}}$$

Clearly if  $\phi$  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

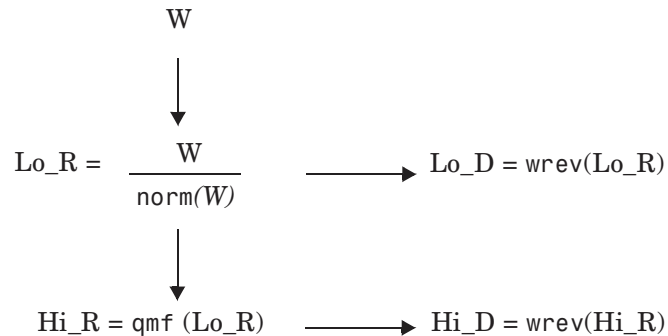
sum(db3)
ans =
    1.0000

norm(db3)
ans =
    0.7071
```

From filter  $W$ , we define four FIR filters, of length  $2N$  and of norm 1, organized as follows.

<b>Filters</b>	<b>Low-Pass</b>	<b>High-Pass</b>
Decomposition	Lo_D	Hi_D
Reconstruction	Lo_R	Hi_R

The four filters are computed using the following scheme.



where  $\text{qmf}$  is such that  $\text{Hi\_R}$  and  $\text{Lo\_R}$  are quadrature mirror filters (i.e.,  $\text{Hi\_R}(k) = (-1)^k \text{Lo\_R}(2N + 1 - k)$ ) for  $k = 1, 2, \dots, 2N$ .

Note that  $\text{wrev}$  flips the filter coefficients. So  $\text{Hi\_D}$  and  $\text{Lo\_D}$  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 Figure 6-8 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)');

```

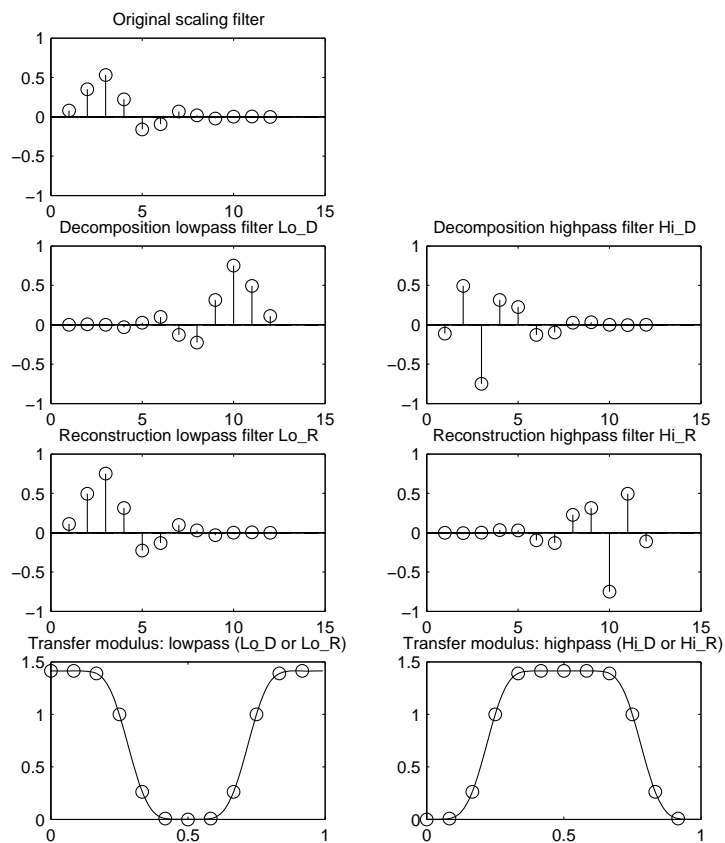
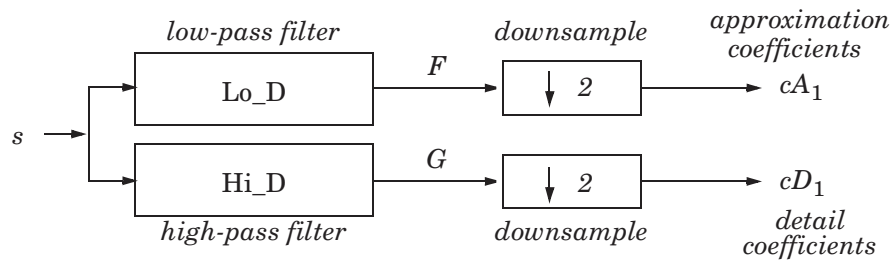


Figure 6-8: Four Wavelet Filters for db6

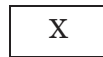
## Algorithms

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.

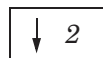
More precisely, the first step is



where



Convolve with filter X.



Keep the even indexed elements  
(see dyaddown).

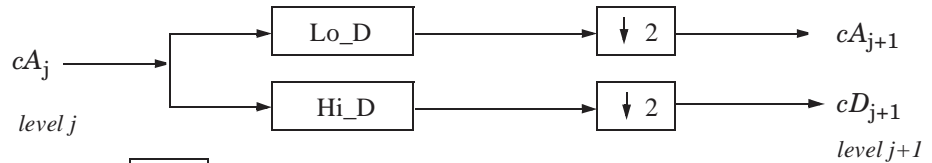
The length of each filter is equal to  $2n$ . If  $N = \text{length}(s)$ , the signals  $F$  and  $G$  are of length  $N + 2n - 1$ , and then the coefficients  $cA_1$  and  $cD_1$  are of length

$$\text{floor}\left(\frac{(N-1)}{2} + n\right)$$

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.

## One-Dimensional DWT

### Decomposition Step

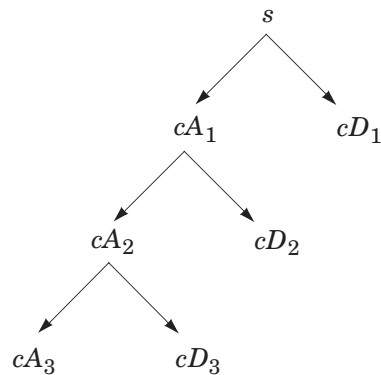


where  $\boxed{X}$  Convolve with filter X.  
 $\boxed{\downarrow 2}$  Downsample.

**Initialization**  $cA_0 = s$ .

So the wavelet decomposition of the signal  $s$  analyzed at level  $j$  has the following structure:  $[cA_j, cD_j, \dots, cD_1]$ .

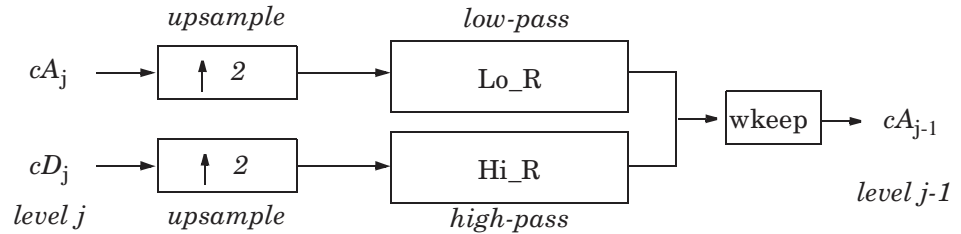
This structure contains for  $J = 3$  the terminal nodes of the following tree.



- Conversely, starting from  $cA_j$  and  $cD_j$ , the IDWT reconstructs  $cA_{j-1}$ , inverting the decomposition step by inserting zeros and convolving the results with the reconstruction filters.

## One-Dimensional IDWT

### Reconstruction Step



where

↑ 2	Insert zeros at odd-indexed elements.
X	Convolve with filter X.
wkeep	Take the central part of U with the convenient length.

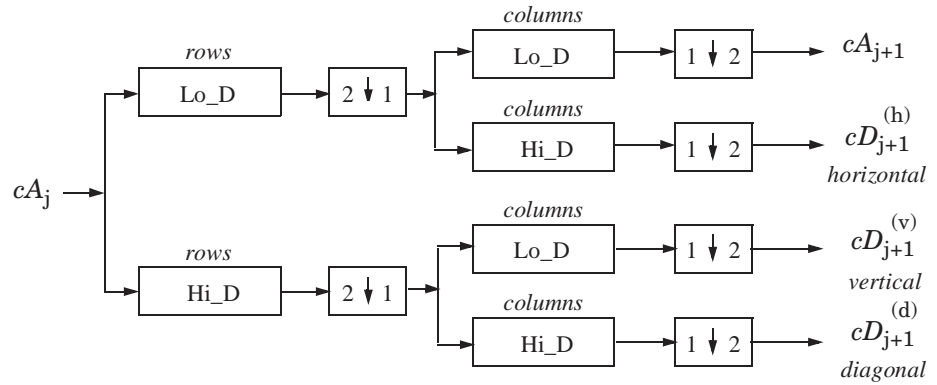
- For images, a similar algorithm is possible for two-dimensional wavelets and scaling functions obtained from one-dimensional wavelets by tensorial product.

This kind of two-dimensional DWT leads to a decomposition of approximation coefficients at level  $j$  in four components: the approximation at level  $j + 1$  and the details in three orientations (horizontal, vertical, and diagonal).

The following charts describe the basic decomposition and reconstruction steps for images.

## Two-Dimensional DWT

### Decomposition Step



where  $\begin{bmatrix} 2 \\ \downarrow \\ 1 \end{bmatrix}$  Downsample columns: keep the even indexed columns.

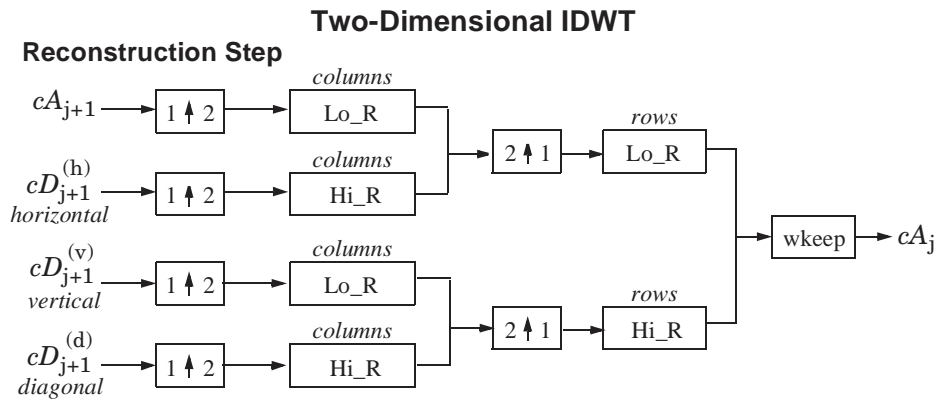
$\begin{bmatrix} 1 \\ \downarrow \\ 2 \end{bmatrix}$  Downsample rows: keep the even indexed rows.

$\begin{matrix} \text{rows} \\ \boxed{X} \end{matrix}$  Convolve with filter X the rows of the entry.

$\begin{matrix} \text{columns} \\ \boxed{X} \end{matrix}$  Convolve with filter X the columns of the entry.

**Initialization**  $CA_0 = s$  for the decomposition initialization.





where

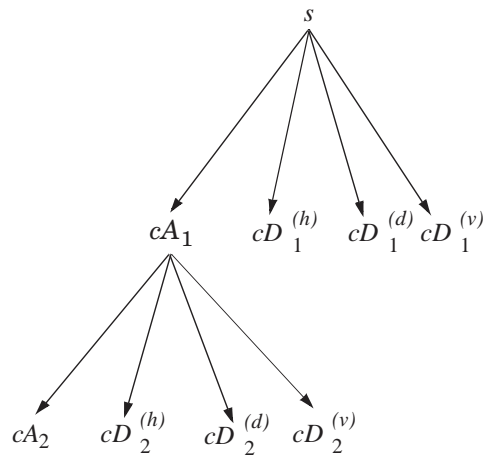
$\begin{matrix} 2 \uparrow 1 \end{matrix}$  Upsample columns: insert zeros at odd-indexed columns.

$\begin{matrix} 1 \uparrow 2 \end{matrix}$  Upsample rows: insert zeros at odd-indexed rows.

$\begin{matrix} \text{rows} \\ \text{X} \end{matrix}$  Convolve with filter X the rows of the entry.

$\begin{matrix} \text{columns} \\ \text{X} \end{matrix}$  Convolve with filter X the columns of the entry.

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  $h = \text{Lo\_R}$  and  $g = \text{Hi\_R}$  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  $(A_k^{(j)})_{k \in Z}$  be the coordinates of the vector  $A_j$ :

$$A_j = \sum_k A_k^{(j)} \phi_{j,k}$$

and  $A_k^{(j+1)}$  the coordinates of the vector  $A_{j+1}$ :

$$A_{j+1} = \sum_k A_k^{(j+1)} \phi_{j+1,k}$$

$A_k^{(j+1)}$  is calculated using the formula

$$A_k^{(j+1)} = \sum_n h_{n-2k} A_n^{(j)}$$

This formula resembles a convolution formula.

The computation is very simple.

Let us define

$$\tilde{h}(k) = h(-k), \text{ and } F_k^{(j+1)} = \sum_n \tilde{h}_{k-n} A_n^{(j)}$$

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  $\phi_{j,k}$  and  $\psi_{j,k}$ .

Let us now describe some of them.

- 1 The family  $(\phi_{0,k}, k \in Z)$  is formed of orthonormal functions. As a consequence for any  $j$ , the family  $(\phi_{j,k}, k \in Z)$  is orthonormal.
- 2 The double indexed family  $(\psi_{j,k}, j \in Z, k \in Z)$  is orthonormal.
- 3 For any  $j$ , the  $(\phi_{j,k}, k \in Z)$  are orthogonal to  $(\psi_{j',k}, j' \leq j, k \in Z)$ .
- 4 Between two successive scales, we have a fundamental relation, called the *twin-scale relation*.

---

#### Twin-Scale Relation for $\phi$

---

$$\phi_{1,0} = \sum_{k \in Z} h_k \phi_{0,k}$$

$$\phi_{j+1,0} = \sum_{k \in Z} h_k \phi_{j,k}$$


---

This relation introduces the algorithm's  $h$  filter ( $h_n = \sqrt{2}w_n$ ). For more information, see "Filters Used to Calculate the DWT and IDWT" on page 6-19.

5 We check that:

- The coordinate of  $\phi_{j+1,0}$  on  $\phi_{j,k}$  is  $h_k$  and does not depend on  $j$ .

**b** The coordinate of  $\phi_{j+1, n}$  on  $\phi_{j, k}$  is equal to  $\langle \phi_{j+1, n}, \phi_{j, k} \rangle = h_{k-2n}$ .

**6** These relations supply the ingredients for the algorithm.

**7** Up to now we used the filter  $h$ . The high-pass filter  $g$  is used in the twin scales relation linking the  $\psi$  and  $\phi$  functions. Between two successive scales, we have the following twin-scale fundamental relation.

---

**Twin-Scale Relation Between  $\psi$  and  $\phi$**

---

$$\Psi_{1, 0} = \sum_{k \in Z} g_k \phi_{0, k}$$

$$\Psi_{j+1, 0} = \sum_{k \in Z} g_k \phi_{j, k}$$


---

**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  $A_0 = A_1 + D_1$ . The procedure is the same to calculate  $A_j = A_{j+1} + D_{j+1}$ .

Let us define  $\alpha_n, \delta_n, \alpha_k^0$  by

$$A_1 = \sum_n \alpha_n \phi_{1, n} \quad D_1 = \sum_n \delta_n \psi_{1, n} \quad A_0 = \sum_k \alpha_k^0 \phi_{0, k}$$

Let us assess the  $\alpha_k^0$  coordinates as

$$\begin{aligned} \alpha_k^0 &= \langle A_0, \phi_{0, k} \rangle = \langle A_1 + D_1, \phi_{0, k} \rangle = \langle A_1, \phi_{0, k} \rangle + \langle D_1, \phi_{0, k} \rangle \\ &= \sum_n \alpha_n \langle \phi_{1, n}, \phi_{0, k} \rangle + \sum_n \delta_n \langle \psi_{1, n}, \phi_{0, k} \rangle \\ &= \sum_n \alpha_n h_{k-2n} + \sum_n \delta_n g_{k-2n} \end{aligned}$$

We will focus our study on the first sum  $\sum_n \alpha_n h_{k-2n}$ ; the second sum

$\sum_n \delta_n g_{k-2n}$  is handled in a similar manner.

The calculations are easily organized if we note that (taking  $k = 0$  in the previous formulas, makes things simpler)

$$\begin{aligned}\sum_n \alpha_n h_{-2n} &= \dots + \alpha_{-1} h_2 + \alpha_0 h_0 + \alpha_1 h_{-2} + \alpha_2 h_{-4} + \dots \\ &= \dots + \alpha_{-1} h_2 + 0h_1 + \alpha_0 h_0 + 0h_{-1} + \alpha_1 h_{-2} + 0h_{-3} + \alpha_2 h_{-4} + \dots\end{aligned}$$

If we transform the  $(\alpha_n)$  sequence into a new sequence  $(\tilde{\alpha}_n)$  defined by

$\dots, \alpha_{-1}, 0, \alpha_0, 0, \alpha_1, 0, \alpha_2, 0, \dots$  that is precisely

$$\alpha_{2n} = \alpha_n, \alpha_{2n+1} = 0$$

Then

$$\sum_n \alpha_n h_{-2n} = \sum_n \tilde{\alpha}_n h_{-n}$$

and by extension

$$\sum_n \alpha_n h_{k-2n} = \sum_n \tilde{\alpha}_n h_{k-n}$$

Since

$$\alpha_k^0 = \sum_n \tilde{\alpha}_n h_{k-n} + \sum_n \tilde{\delta}_n g_{k-n}$$

the reconstruction steps are:

- 1** Replace the  $\alpha$  and  $\delta$  sequences by upsampled versions  $\tilde{\alpha}$  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
	$A_k, 0 \leq k \leq j$	Approximation at level $k$
	$D_k, 1 \leq k \leq j$	Detail at level $k$
Coefficients in scale-related time	$cA_k, 1 \leq k \leq j$	Approximation coefficients at level $k$
	$cD_k, 1 \leq k \leq j$	Detail coefficients at level $k$
	$[cA_j, cD_j, \dots, cD_1]$	Wavelet decomposition at level $j, j \geq 1$

### Analysis-Decomposition Capabilities.

Purpose	Input	Output	M-File
Single-level decomposition	$s$	$cA_1, cD_1$	dwt
Single-level decomposition	$cA_j$	$cA_{j+1}, cD_{j+1}$	dwt
Decomposition	$s$	$[cA_j, cD_j, \dots, cD_1]$	wavedec

### Synthesis-Reconstruction Capabilities.

Purpose	Input	Output	M-File
Single-level reconstruction	$cA_1, cD_1$	$s$ or $A_0$	idwt
Single-level reconstruction	$cA_{j+1}, cD_{j+1}$	$cA_j$	idwt
Full reconstruction	$[cA_j, cD_j, \dots, cD_1]$	$s$ or $A_0$	waverec
Selective reconstruction	$[cA_j, cD_j, \dots, cD_1]$	$A_l, D_m$	wrcoef

**Decomposition Structure Utilities.**

<b>Purpose</b>	<b>Input</b>	<b>Output</b>	<b>M-File</b>
Extraction of detail coefficients	$[cA_j, cD_j, \dots, cD_1]$	$cD_k,$ $1 \leq k \leq j$	detcoef
Extraction of approximation coefficients	$[cA_j, cD_j, \dots, cD_1]$	$cA_k,$ $0 \leq k \leq j$	appcoef
Recomposition of the decomposition structure	$[cA_j, cD_j, \dots, cD_1]$	$[cA_k, cD_k, \dots,$ $cD_1]$ $1 \leq k \leq j$	upwlev

To illustrate command-line mode for one-dimensional capabilities, see “One-Dimensional Analysis Using the Command Line” on page 2-31.

**Two-Dimensional Wavelet Capabilities****Basic Two-Dimensional Objects.**

	<b>Objects</b>	<b>Description</b>
Image in original resolution	$s$	Original image
	$A_0$	Approximation at level 0
	$A_k, 1 \leq k \leq j$	Approximation at level $k$
	$D_k, 1 \leq k \leq j$	Details at level $k$
Coefficients in scale-related resolution	$cA_k, 1 \leq k \leq j$	Approximation coefficients at level $k$
	$cD_k, 1 \leq k \leq j$	Detail coefficients at level $k$
	$[cA_j, cD_j, \dots, cD_1]$	Wavelet decomposition at level $j$

$D_k$  stands for  $[D_k^{(h)}, D_k^{(v)}, D_k^{(d)}]$ , the horizontal, vertical, and diagonal details at level  $k$ .

The same holds for  $cD_k$ , which stands for  $[cD_k^{(h)}, cD_k^{(v)}, cD_k^{(d)}]$ .

The two-dimensional M-files are 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” on page 2-69.



## 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 6-155). 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 6-155).

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

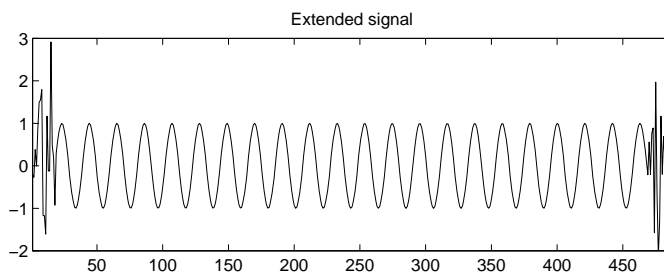
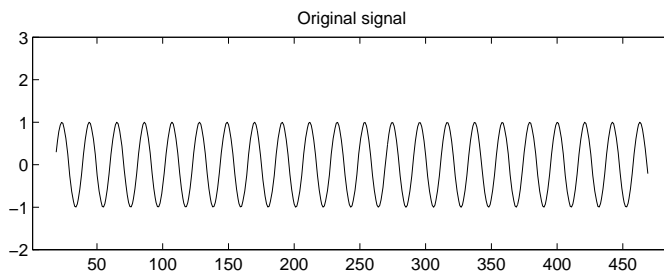
the properties **1** and **2** can be lost. These properties hold true for an extended signal of length usually larger than the length of the original signal. So only the perfect reconstruction property is always preserved. Nevertheless if the DWT is performed using the periodic extension mode ('per') 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

```

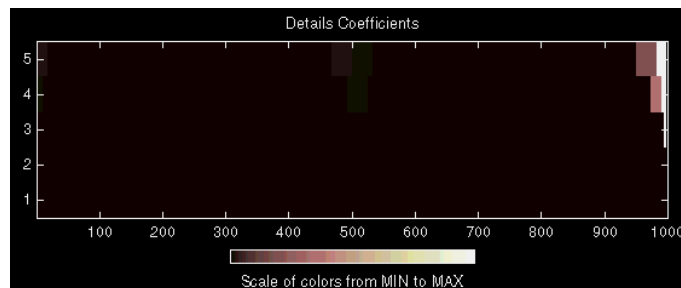
Now let us illustrate the differences between the first three methods both for 1-D and 2-D signals.

### Zero-Padding.

Using the GUI we will examine the effects of zero-padding.

- 1 From the MATLAB<sup>®</sup> 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')
```

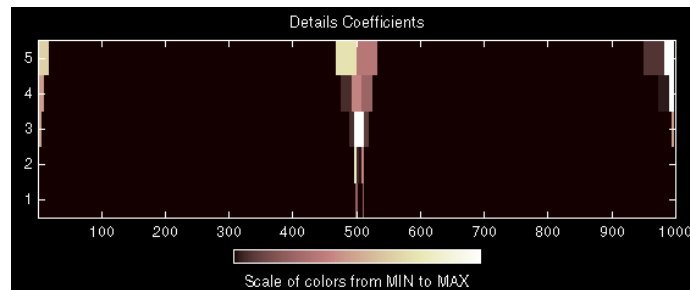
**7** Click the **Wavelet 1-D** menu item.

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')
```

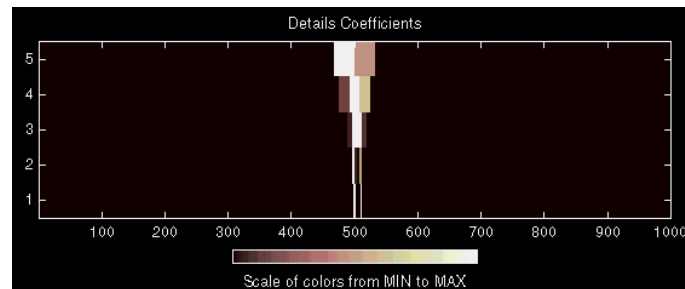
**11** Click the **Wavelet 1-D** menu item.

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.

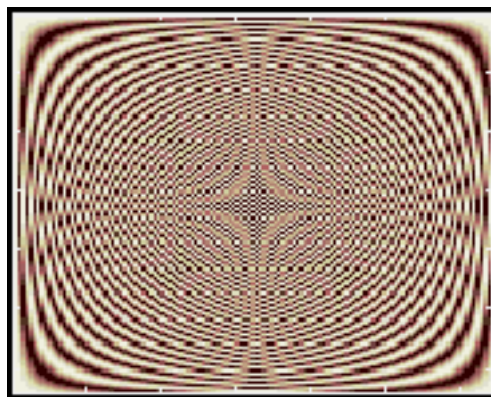


Let us now consider an image example.

#### Original Image.

1 From the MATLAB prompt, type

```
load geometry;
% X contains the loaded image and
% map contains the loaded colormap.
nbc = size(map,1);
colormap(pink(nbc));
image(wcodemat(X,nbc));
```



**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));
```

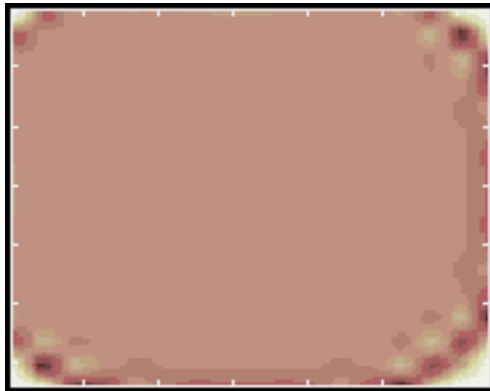
**Symmetric Extension.**

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));
```



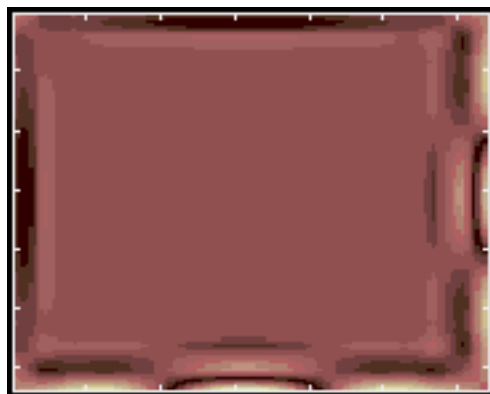


#### Smooth Padding.

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  $\varepsilon$ -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 6-155. For examples, see “One-Dimensional Discrete Stationary Wavelet Analysis” on page 2-104 and “Two-Dimensional Discrete Stationary Wavelet Analysis” on page 2-122.

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 6-97), but applied to the coefficients of an  $\varepsilon$ -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.

---

### $\varepsilon$ -Decimated DWT

What is an  $\varepsilon$ -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:  $\varepsilon = \varepsilon_1, \dots, \varepsilon_J$ . This transform is called the  $\varepsilon$ -decimated DWT.

You can obtain the basis vectors of the  $\varepsilon$ -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 $\varepsilon$ -Decimated DWT: SWT

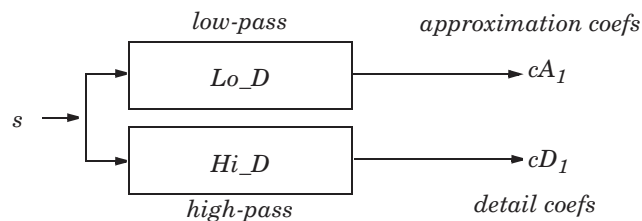
It is possible to calculate all the  $\varepsilon$ -decimated DWT for a given signal of length  $N$ , by computing the approximation and detail coefficients for every possible sequence  $\varepsilon$ . Do this using iteratively, a slightly modified version of the basic step of the DWT of the form:

```
[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  $\varepsilon$ -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  $\varepsilon$ -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.

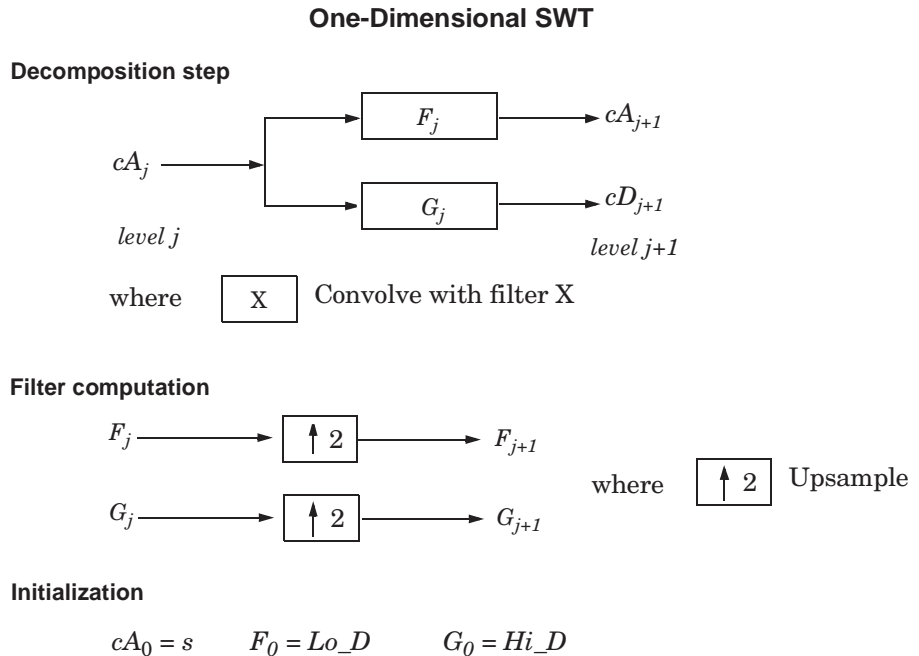


where:



Convolve with filter X

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  $\varepsilon$ -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  $A(j, \varepsilon_1, \dots, \varepsilon_j)$  or  $D(j, \varepsilon_1, \dots, \varepsilon_j)$  denotes an approximation or a detail coefficient at level  $j$  obtained for the  $\varepsilon$ -decimated DWT characterized by  $\varepsilon = [\varepsilon_1, \dots, \varepsilon_j]$ .

**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)	<b>D(1,1)</b>	D(1,0)	<b>D(1,1)</b>	D(1,0)	<b>D(1,1)</b>	D(1,0)	<b>D(1,1)</b>
A(1,0)	<b>A(1,1)</b>	A(1,0)	<b>A(1,1)</b>	A(1,0)	<b>A(1,1)</b>	A(1,0)	<b>A(1,1)</b>

**Step 2.**

D(1,0)	<b>D(1,1)</b>	D(1,0)	<b>D(1,1)</b>	D(1,0)	<b>D(1,1)</b>	D(1,0)	<b>D(1,1)</b>
D(2,0,0)	<b>D(2,1,0)</b>	D(2,0,1)	D(2,1,1)	D(2,0,0)	<b>D(2,1,0)</b>	D(2,0,1)	D(2,1,1)
A(2,0,0)	<b>A(2,1,0)</b>	A(2,0,1)	A(2,1,1)	A(2,0,0)	<b>A(2,1,0)</b>	A(2,0,1)	A(2,1,1)

**Step 3.**

D(1,0)	<b>D(1,1)</b>	D(1,0)	<b>D(1,1)</b>	D(1,0)	<b>D(1,1)</b>	D(1,0)	<b>D(1,1)</b>
D(2,0,0)	<b>D(2,1,0)</b>	D(2,0,1)	D(2,1,1)	D(2,0,0)	<b>D(2,1,0)</b>	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)	<b>D(3,1,0,1)</b>	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)	<b>A(3,1,0,1)</b>	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  $\varepsilon_j = 0$  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  $\varepsilon$ -circular shift of the input vector. Therefore, if  $\varepsilon_{j+1} = 0$ , the `wshift` instruction is ineffective and can be suppressed.

Let  $\varepsilon = [\varepsilon_1, \dots, \varepsilon_J]$  with  $\varepsilon_i = 0$  or  $1$ . We have  $2^J = 2^3 = 8$  different  $\varepsilon$ -decimated DWTs at level 3. Choosing  $\varepsilon$ , we can retrieve the corresponding  $\varepsilon$ -decimated DWT from the SWT array.

Now, consider the last step,  $J = 3$ , and let  $[C_\varepsilon, L_\varepsilon]$  denote the wavelet decomposition structure of an  $\varepsilon$ -decimated DWT for a given  $\varepsilon$ . Then, it can be retrieved from the SWT decomposition structure by selecting the appropriate coefficients as follows:

$C_\varepsilon =$

<b>A(3, <math>\varepsilon_1, \varepsilon_2, \varepsilon_3</math>)</b>	<b>D(3, <math>\varepsilon_1, \varepsilon_2, \varepsilon_3</math>)</b>	<b>D(2, <math>\varepsilon_1, \varepsilon_2</math>)</b>	<b>D(2, <math>\varepsilon_1, \varepsilon_2</math>)</b>	<b>D(1, <math>\varepsilon_1</math>)</b>	<b>D(1, <math>\varepsilon_1</math>)</b>	<b>D(1, <math>\varepsilon_1</math>)</b>	<b>D(1, <math>\varepsilon_1</math>)</b>
---	---	--	--	---	---	---	---

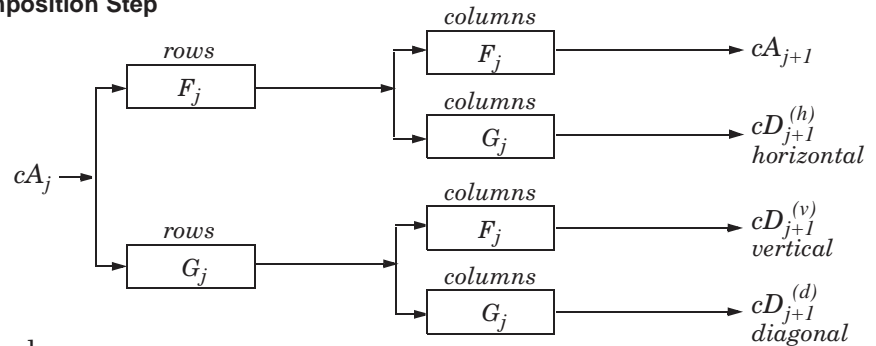
$L_\varepsilon = [1, 1, 2, 4, 8]$

For example, the  $\varepsilon$ -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

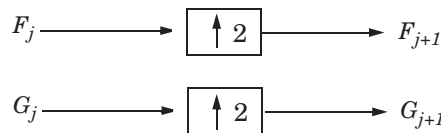


where

$\begin{matrix} \text{rows} \\ \boxed{X} \end{matrix}$  Convolve with filter X the rows of the entry

$\begin{matrix} \text{columns} \\ \boxed{X} \end{matrix}$  Convolve with filter X the columns of the entry

#### Filter Computation



where:  $\begin{matrix} \uparrow \\ \boxed{2} \end{matrix}$  Upsample

#### 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  $\varepsilon$ -decimated DWT corresponding to a given  $\varepsilon$  can be inverted.

To reconstruct the original signal using a given  $\varepsilon$ -decimated DWT characterized by  $[\varepsilon_1, \dots, \varepsilon_J]$ , we can use the abstract algorithm

```
FOR j = J:-1:1
  A(j-1,  $\varepsilon_1, \dots, \varepsilon_{j-1}$ ) = ...
  idwt(A(j,  $\varepsilon_1, \dots, \varepsilon_j$ ), D(S,  $\varepsilon_1, \dots, \varepsilon_j$ ), wname, 'mode', 'per', 'shift',  $\varepsilon_j$ );
END
```

For each choice of  $\varepsilon = (\varepsilon_1, \dots, \varepsilon_J)$ , 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  $\varepsilon$ -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,  $\varepsilon_1, \dots, \varepsilon_j$ ), D(j,  $\varepsilon_1, \dots, \varepsilon_j$ ), wname, ...
    'mode', 'per', 'shift', 0);
  X1 = idwt(A(j,  $\varepsilon_1, \dots, \varepsilon_j$ ), D(j,  $\varepsilon_1, \dots, \varepsilon_j$ ), wname, ...
    'mode', 'per', 'shift', 1);
  X1 = wshift('1D', X1, 1);
  A(j-1,  $\varepsilon_1, \dots, \varepsilon_{j-1}$ ) = (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 6-155.



## 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” on page 2-216).

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 6-155) 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 6-51
- “Lifting Functions” on page 6-54

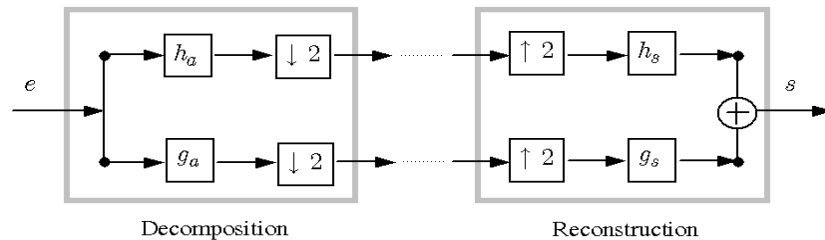
For more information on lifting, see [Swe98], [Mal98], [StrN96], and [MisMOP03] in “References” on page 6-155.

### Lifting Background

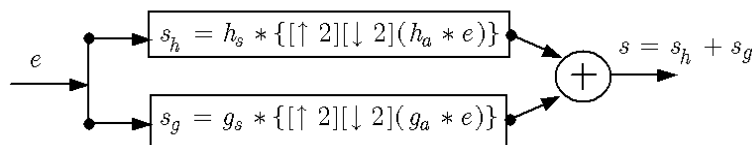
The DWT is defined by four filters as described in “Fast Wavelet Transform (FWT) Algorithm” on page 6-19. Two main properties of interest are

- 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

$$C(z) = c_1 z^{\text{pmax}} + c_2 z^{\text{pmax}-1} + \dots + c_{\text{end}} z^{\text{pmin}}$$

involving positive and negative integer powers of  $z$ . The degree of  $C$  is defined as  $(\text{pmax}-\text{pmin})$ .

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

$$H_s^H(z) = H_s(z) + G_s(z) T(z^2)$$

$$G_a^N(z) = G_a(z) - H_a(z) T(z^{-2})$$

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<sup>®</sup> conventions) by

$$P(z) = [ \text{even}(H_s)(z) \quad \text{even}(G_s)(z) ; \text{odd}(H_s)(z) \quad \text{odd}(G_s)(z) ]$$

where

$$\text{even}(C)(z^2) = (C(z) + C(-z)) / 2$$

$$\text{odd}(C)(z^2) = (C(z) - C(-z)) / 2z^{-1}$$

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 6-155) 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” on page 6-55
- “Biorthogonal Quadruplets of Filters and Lifting Schemes” on page 6-55
- “Usual Biorthogonal Quadruplets” on page 6-55
- “Lifting Wavelet Transform (LWT)” on page 6-56
- “Laurent Polynomials and Matrices” on page 6-56

## 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) M-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 directory and the two object directories @laurpoly and @laurmat contain many other M-files.

## Examples of Lifting Methods

These two simple examples illustrate the basic lifting capabilities of Wavelet Toolbox software. For more examples, see “Wavelets in Action: Examples and Case Studies” on page 4-1 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] []
};

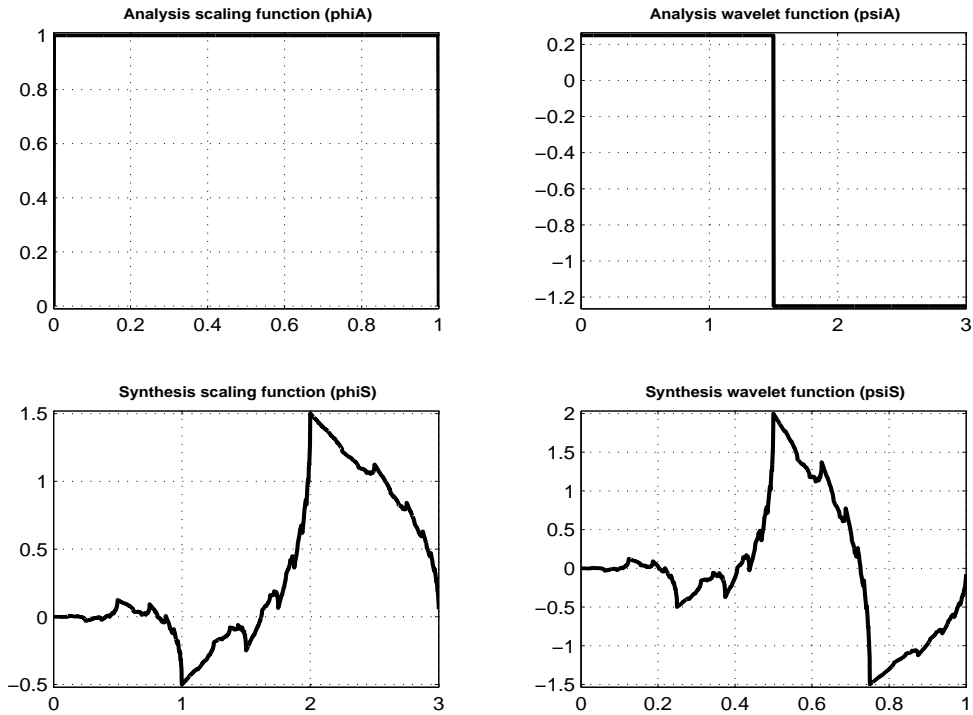
% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);
displs(lsnew);

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

```

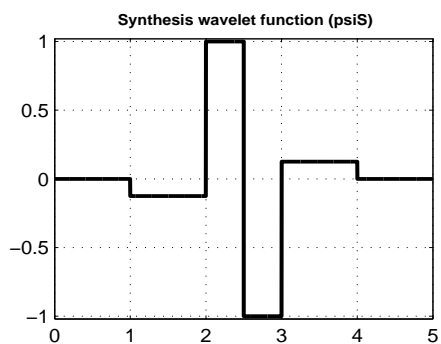
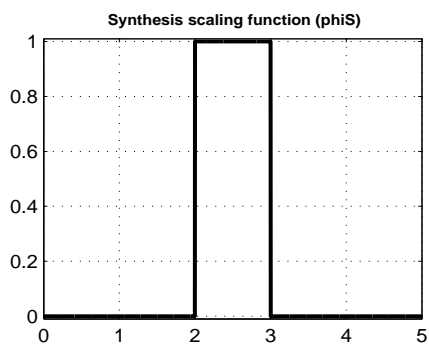
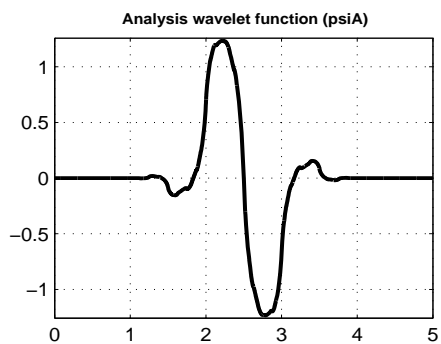
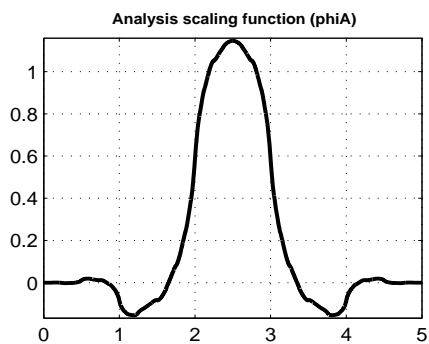
% Get Haar filters.
[LoD,HiD,LoR,HiR] = wfilters('haar');

% 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 insignificant 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  $C(j, k), (j, k) \in \mathbb{Z}^2$ .

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  $(\phi_{0,k}; \psi_{j,l})_{j \leq 0, (k, l) \in \mathbb{Z}^2}$  used for the analysis is an orthogonal basis, therefore leading to nonredundancy. The orthogonality properties are  $\phi_{0,k} \perp \psi_{j',k'}$  as soon as  $j' \leq 0$ , and  $\psi_{j,k} \perp \psi_{j',k'}$  as soon as  $(j,k) \neq (j',k')$ .

Let us remember that for a one-dimensional signal,  $u \perp v$  stands for

$$\int_{\mathbb{R}} u(x)v(x)dx = 0$$

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 ( $\int_{\mathbb{R}} t^j \psi(t) dt = 0$  for

$j = 0, \dots, k$ ) all the polynomial signals  $s(t) = \sum_{0 \leq j \leq k} a_j t^j$  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 $\psi$ ?

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 ( $\geq 0$ ), 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.

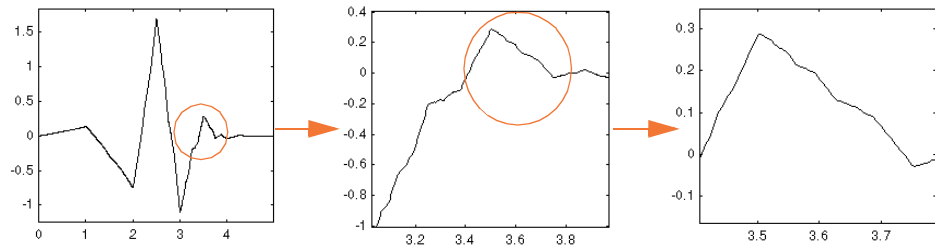
The regularity of certain wavelets is known. The following table gives some indications for Daubechies wavelets.

$\psi$	<b>db1 = Haar</b>	<b>db2</b>	<b>db3</b>	<b>db4</b>	<b>db5</b>	<b>db7</b>	<b>db10</b>
<b>Regularity</b>	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  $N \rightarrow \infty$ ,

$$\text{length}(\text{support}) = 2N, \text{ regularity } s \approx \frac{N}{5}.$$

The functions are more regular at certain points than at others (see Figure 6-9 on page 6-63).



**Figure 6-9: Zooming in on a *db3* 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] page 301).

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  $\psi \in L^1(\mathbb{R}) \cap L^2(\mathbb{R})$  is

$$\int_{\mathbb{R}^+} \frac{|\hat{\psi}(s)|^2}{|s|} ds = \int_{\mathbb{R}^+} \frac{|\hat{\psi}(s)|^2}{|s|} ds = K_{\psi} < +\infty$$

The family of shifts and dilations of  $\psi$  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  $\psi$  wavelet is usually associated with a scaling function  $\phi$ . There are, however, some  $\psi$  wavelets for which we do not know how to associate a  $\phi$ . In some cases we know how to prove that  $\phi$  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” on page 2-216 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  $\phi$  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 6-51).

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 6-155), 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  $\phi(x/2)$  and  $\phi(x)$ . Another relation of the same type links  $\psi(x/2)$  to  $\phi(x)$ . These are the relations of the two scales, the twin-scale relations.

Indeed there are two sequences  $h$  and  $g$  of coefficients such that

$$h \in l^2(\mathbb{Z}), g \in l^2(\mathbb{Z})$$

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

$$\hat{\phi}(2\omega) = \frac{1}{\sqrt{2}} \hat{h}(\omega) \hat{\phi}(\omega) \quad \hat{\psi}(2\omega) = \frac{1}{\sqrt{2}} \hat{g}(\omega) \hat{\phi}(\omega)$$

There are  $\phi$  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 6-155) 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 6-155).

### What Is the Link Between Wavelet and Fourier Analysis?

Wavelet analysis complements the Fourier analysis for which there are several functions: `fft` in MATLAB<sup>®</sup> software and `spectrum` and `sptool` in Signal Processing Toolbox<sup>™</sup> software.

Fourier analysis uses the basic functions  $\sin(\omega t)$ ,  $\cos(\omega t)$ , and  $\exp(i\omega t)$ .

- 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  $\omega$  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 6-155). 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  $\omega$ .

The Fourier coefficient  $G_s(\omega, t)$  depends on the values of the signal  $s$  on the segment  $[t - m, t + m]$  with a constant width. If  $\psi$ , 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,  $\psi(t) = Ce^{-t^2/\alpha^2}(e^{i\pi t} - e^{-\pi^2\alpha^2/4})$  with  $\alpha = 4$ , 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 6-155).

$$F_a = \frac{F_c}{a \cdot \Delta}$$



where

- $a$  is a scale.
- $\Delta$  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. Figure 6-10 on page 6-68 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  $\Delta$ , 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.

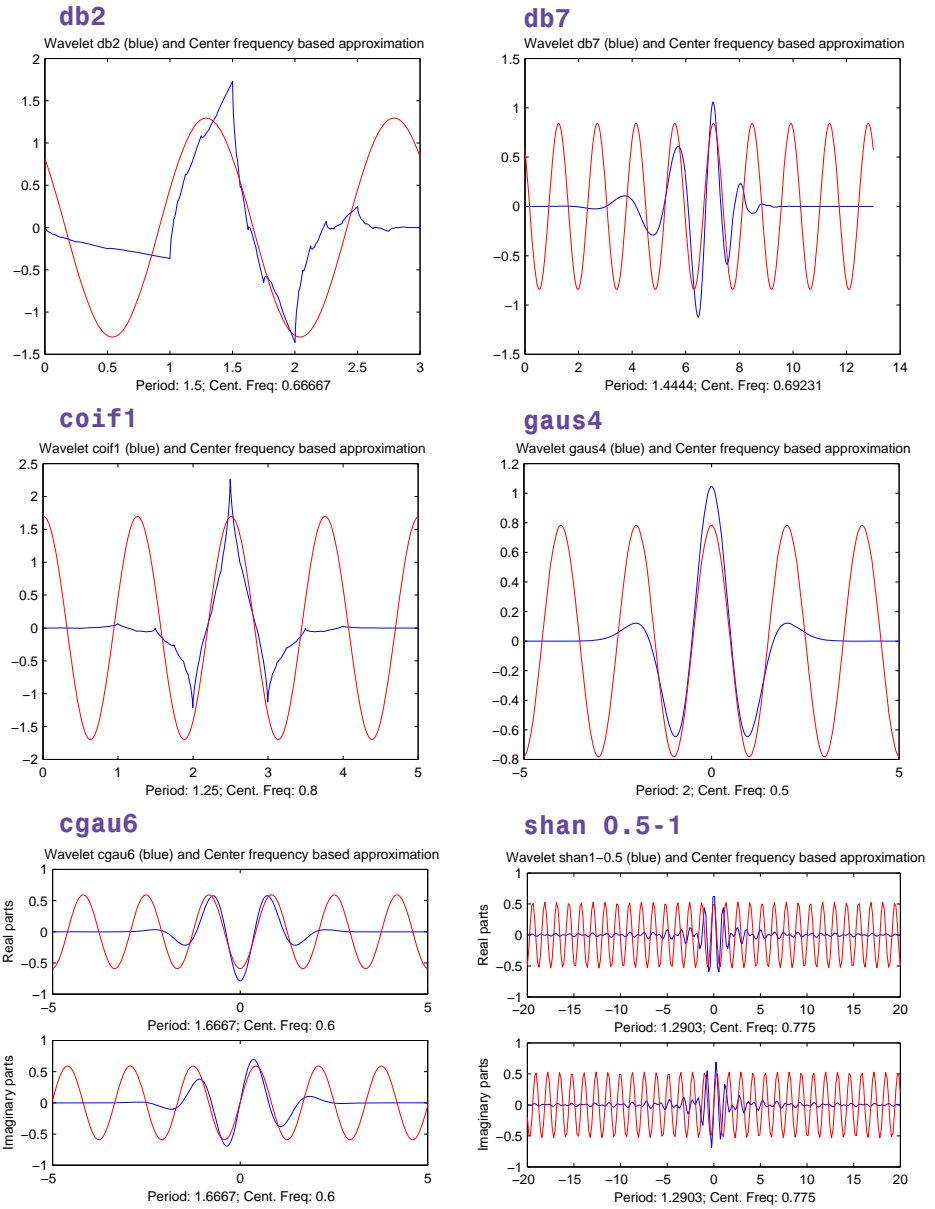


Figure 6-10: Center Frequencies for Real and Complex Wavelets

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.
```

```

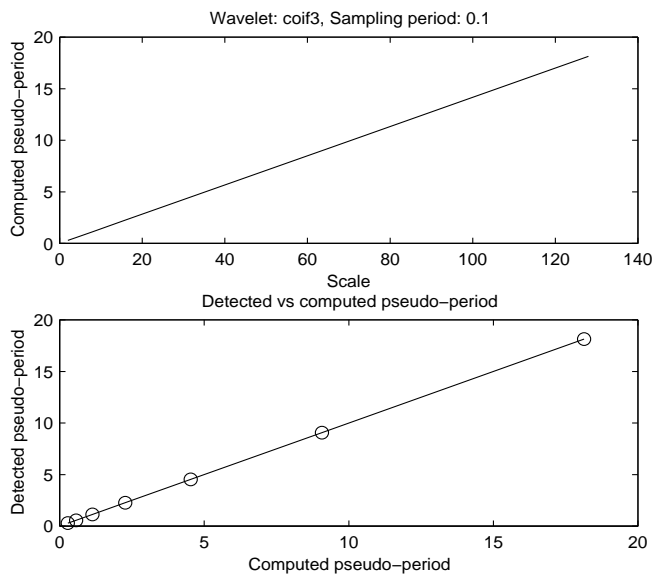
t = 0:delta:100;
x = sin((t.*2*pi)/per(i));

% 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')

```



**Figure 6-11: 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  $\psi$ ,  $\hat{\psi}$  (and  $\phi$ ,  $\hat{\phi}$ ): the speed of convergence to 0 of these functions ( $\psi(t)$  or  $\psi(\omega)$ ) when the time  $t$  or the frequency  $\omega$  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  $\psi$  or for  $\phi$  (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  $\phi$
- 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.

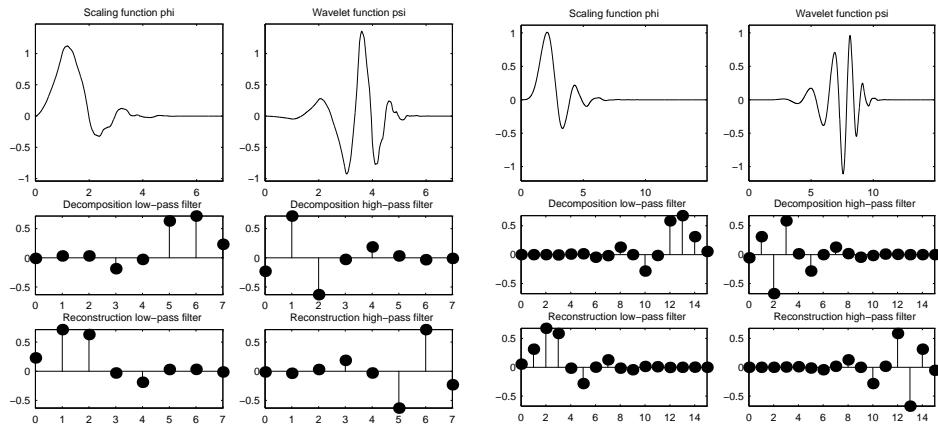
Note that the  $\phi$  and  $\psi$  functions can be computed using `wavefun`; the filters are generated using `wfilters`. We provide definition equations for several wavelets. Some are given explicitly by their time definitions, others by their frequency definitions, and still others by their filters.

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
'mor1'	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  $2N$  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.



**Figure 6-12: Daubechies Wavelets db4 on the Left and db8 on the Right**

This family includes the Haar wavelet, written *db1*, the simplest wavelet imaginable and certainly the earliest. Using `waveinfo('haar')`, you can obtain a survey of the main properties of this wavelet.

### Haar

$$\psi(x) = 1, \quad \text{if} \quad x \in [0, 0.5[$$

$$\psi(x) = -1, \quad \text{if} \quad x \in [0.5, 1[$$

$$\psi(x) = 0, \quad \text{if} \quad x \notin [0, 1[$$

$$\phi(x) = 1, \quad \text{if} \quad x \in [0, 1]$$

$$\phi(x) = 0, \quad \text{if} \quad x \notin [0, 1]$$

### dbN

These wavelets have no explicit expression except for *db1*, 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  $\psi$  and  $\phi$  is  $2N - 1$ . The number of vanishing moments of  $\psi$  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,  $\psi$  and  $\phi$  belong to  $C^{\mu N}$  where  $\mu$  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.

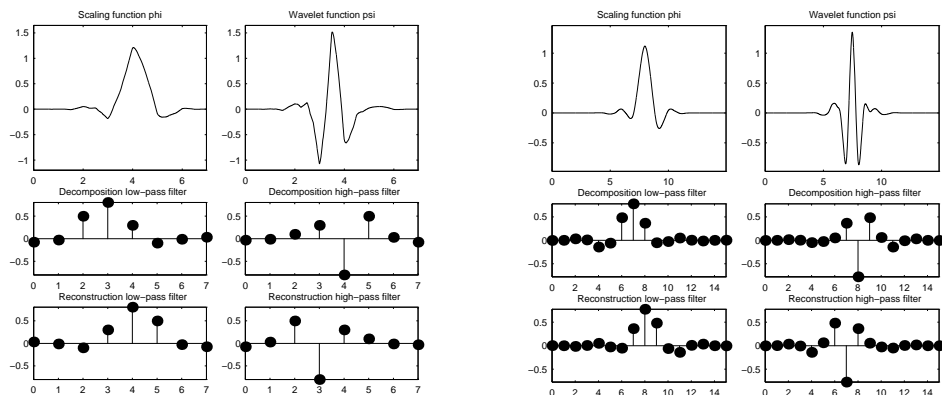


Figure 6-13: Symlets *sym4* on the Left and *sym8* on the Right



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  $W(z) = U(z)\overline{U(\frac{1}{z})}$  because the roots of  $W$  with modulus not equal to 1 go in pairs. If one of the roots is  $z_1$ , then  $\frac{1}{z_1}$  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  $dbNs$ .

## Coiflet Wavelets: *coifN*

In *coifN*,  $N$  is the order. Some authors use  $2N$  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.

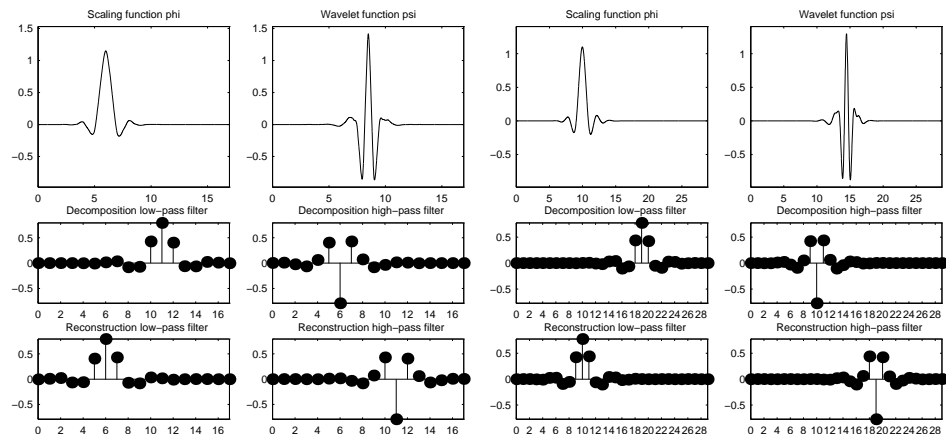


Figure 6-14: Coiflets *coif3* on the Left and *coif5* on the Right

Built by Daubechies at the request of Coifman, the function  $\psi$  has  $2N$  moments equal to 0 and, what is more unusual, the function  $\phi$  has  $2N-1$  moments equal to 0. The two functions have a support of length  $6N-1$ .

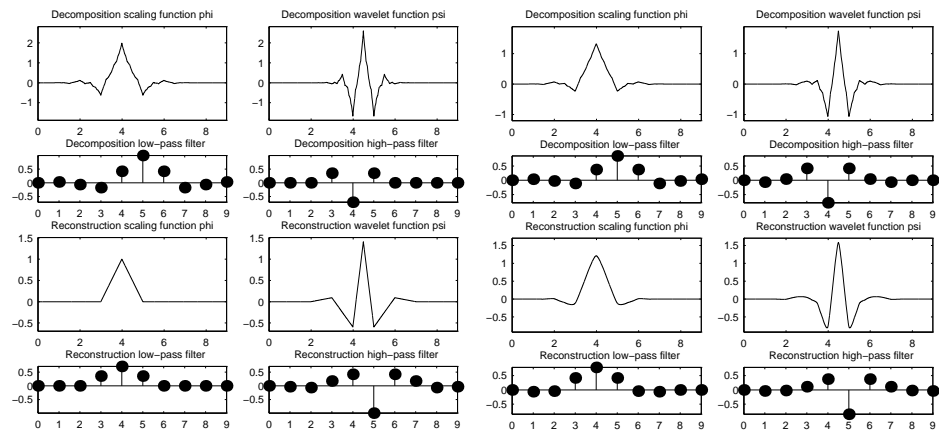
The *coifN*  $\psi$  and  $\phi$  are much more symmetrical than the *dbNs*. With respect to the support length, *coifN* has to be compared to *db3N* or *sym3N*. With respect to the number of vanishing moments of  $\psi$ , *coifN* has to be compared to *db2N* or *sym2N*.

If  $s$  is a sufficiently regular continuous time signal, for large  $j$  the coefficient  $\langle s, \phi_{-j, k} \rangle$  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  $\phi_{j, k}$  of a given signal and its sampled version.

## Biorthogonal Wavelet Pairs: *biorNr.Nd*

More about biorthogonal wavelets can be found in [Dau92] 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.



**Figure 6-15: 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,  $\psi$ , is used in the synthesis

$$s = \sum_{j,k} \tilde{c}_{j,k} \psi_{j,k}$$

In addition, the wavelets  $\tilde{\psi}$  and  $\psi$  are related by duality in the following sense:

$$\int \tilde{\psi}_{j,k}(x) \psi_{j',k'}(x) dx = 0 \text{ as soon as } j \neq j' \text{ or } k \neq k' \text{ and even}$$

$$\int \tilde{\phi}_{0,k}(x) \phi_{0,k'}(x) dx = 0 \text{ as soon as } k \neq k'$$

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  $\tilde{\psi}$  function whereas the interesting properties for synthesis (regularity) are assigned to the  $\psi$  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).

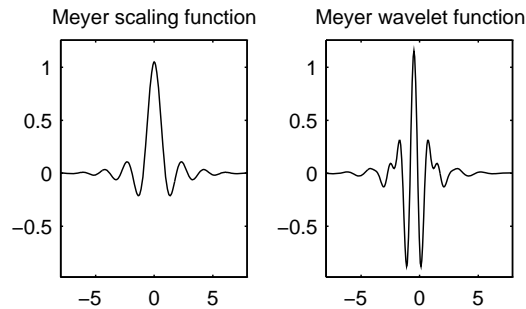
The  $\tilde{\psi}$ ,  $\psi$ ,  $\tilde{\phi}$ , and  $\phi$  functions are zero outside of a segment.

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  $\psi$  and  $\phi$  are defined in the frequency domain, starting with an auxiliary function  $v$  (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.



**Figure 6-16: Meyer Wavelet**

The Meyer wavelet and scaling function are defined in the frequency domain:

- Wavelet function

$$\hat{\psi}(\omega) = (2\pi)^{-1/2} e^{i\omega/2} \sin\left(\frac{\pi}{2} v\left(\frac{3}{2\pi}|\omega| - 1\right)\right) \quad \text{if} \quad \frac{2\pi}{3} \leq |\omega| \leq \frac{4\pi}{3}$$

$$\hat{\psi}(\omega) = (2\pi)^{-1/2} e^{i\omega/2} \cos\left(\frac{\pi}{2} v\left(\frac{3}{4\pi}|\omega| - 1\right)\right) \quad \text{if} \quad \frac{4\pi}{3} \leq |\omega| \leq \frac{8\pi}{3}$$

$$\text{and } \hat{\psi}(\omega) = 0 \quad \text{if} \quad |\omega| \notin \left[\frac{2\pi}{3}, \frac{8\pi}{3}\right]$$

$$\text{where } v(a) = a^4(35 - 84a + 70a^2 - 20a^3) \quad a \in [0,1]$$

- Scaling function

$$\hat{\phi}(\omega) = (2\pi)^{-1/2} \quad \text{if} \quad |\omega| \leq \frac{2\pi}{3}$$

$$\hat{\phi}(\omega) = (2\pi)^{-1/2} \cos\left(\frac{\pi}{2} v\left(\frac{3}{2\pi}|\omega| - 1\right)\right) \quad \text{if} \quad \frac{2\pi}{3} \leq |\omega| \leq \frac{4\pi}{3}$$

$$\hat{\phi}(\omega) = 0 \quad \text{if} \quad |\omega| > \frac{4\pi}{3}$$

By changing the auxiliary function, you get a family of different wavelets. For the required properties of the auxiliary function  $v$  (see “References” on page 6-155 for more information). This wavelet ensures orthogonal analysis.

The function  $\psi$  does not have finite support, but  $\psi$  decreases to 0 when  $x \rightarrow \infty$ , faster than any inverse polynomial

$$\forall n \in \mathbb{N}, \exists C_n \text{ such that } |\psi(x)| \leq C_n (1 + |x|^2)^{-n}$$

This property holds also for the derivatives

$$\forall k \in \mathbb{N}, \forall n \in \mathbb{N}, \exists C_{k,n}, \text{ such that } |\psi^{(k)}(x)| \leq C_{k,n} (1 + |x|^2)^{-n}$$

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

$$\phi(x) = 2^{-2M} \sum_{j=0}^{2M+1} C_j^{2M+1} \phi(2x - M - 1 + j) \quad \text{if } N = 2M$$

$$\phi(x) = 2^{-2M-1} \sum_{j=0}^{2M+2} C_j^{2M+2} \phi(2x - M - 1 + j) \quad \text{if } N = 2M + 1$$

- For an even  $N$ ,  $\phi$  is symmetrical around,  $x = 1/2$ ;  $\psi$  is antisymmetrical around  $x = 1/2$ . For an odd  $N$ ,  $\phi$  is symmetrical around  $x = 0$ ;  $\psi$  is symmetrical around  $x = 1/2$ .
- The analysis becomes orthogonal if we transform the functions  $\psi$  and  $\phi$  somewhat. For  $N=1$ , for instance, let

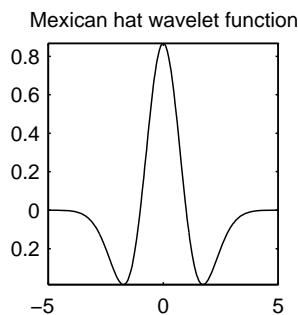
$$\widehat{\phi^\perp}(\omega) = 3^{1/2} (2\pi)^{-1/2} \frac{4 \sin^2(\omega/2)}{\omega^2 [1 + 2 \cos^2(\omega/2)]^{1/2}}$$

- The supports of  $\psi$  and  $\phi^\perp$  are not finite, but the decrease of the functions  $\psi$  and  $\phi^\perp$  to 0 is exponential. The support of  $\phi$  is compact. See [Dau92] p. 151.
- The  $\psi$  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.



**Figure 6-17: Mexican Hat**

$$\psi(x) = \left(\frac{2}{\sqrt{3}}\pi^{-1/4}\right)(1-x^2)e^{-x^2/2}$$

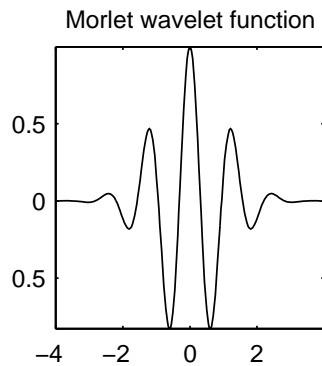
This function is proportional to the second derivative function of the Gaussian probability density function.

As the  $\phi$  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.



**Figure 6-18: Morlet Wavelet**

$$\psi(x) = Ce^{-x^2/2}\cos(5x)$$

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 6-63.

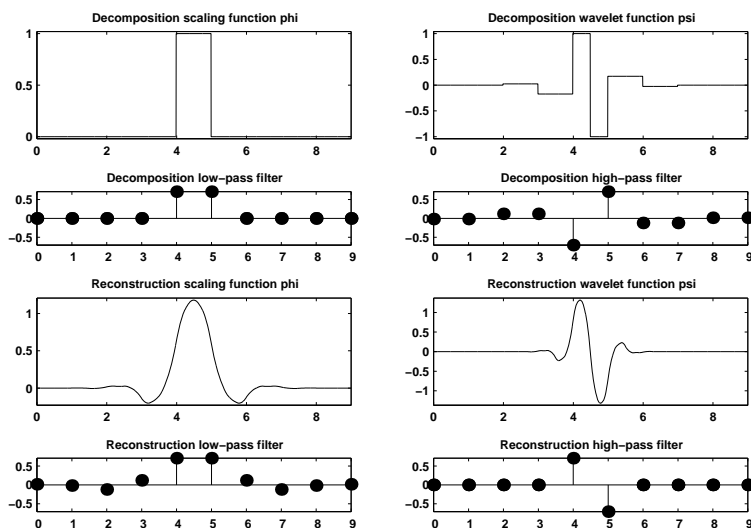
## Additional Real Wavelets

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.



**Figure 6-19: Reverse Biorthogonal Wavelet *rbio1.5***

### Gaussian Derivatives Family: *gaus*

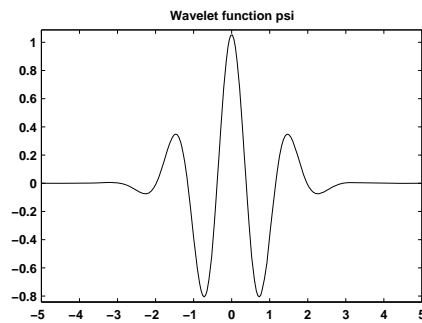
This family is built starting from the Gaussian function  $f(x) = C_p e^{-x^2}$  by taking the  $p^{\text{th}}$  derivative of  $f$ .

The integer  $p$  is the parameter of this family and in the previous formula,  $C_p$  is such that

$$\|f^{(p)}\|^2 = 1 \text{ where } f^{(p)} \text{ is the } p^{\text{th}} \text{ derivative of } f.$$

You can obtain a survey of the main properties of this family by typing `waveinfo('gaus')` from the MATLAB command line.





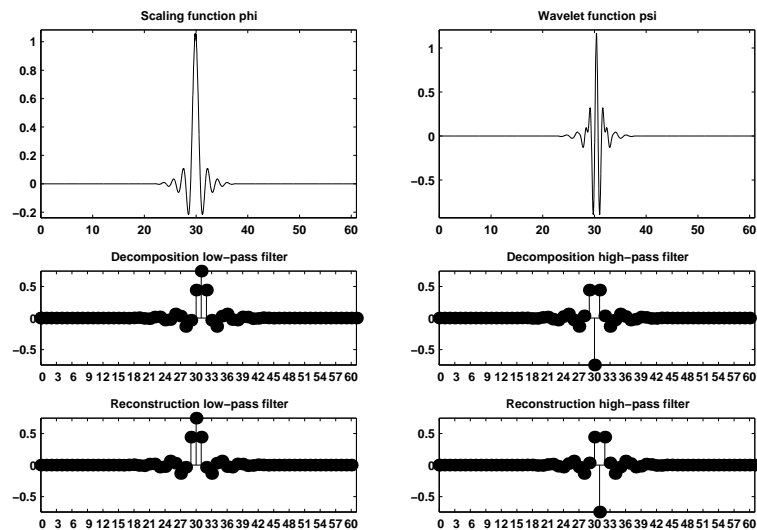
**Figure 6-20: Gaussian Derivative Wavelet gauss8**

### 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.



**Figure 6-21: FIR Based Approximation of the Meyer Wavelet**

## Complex Wavelets

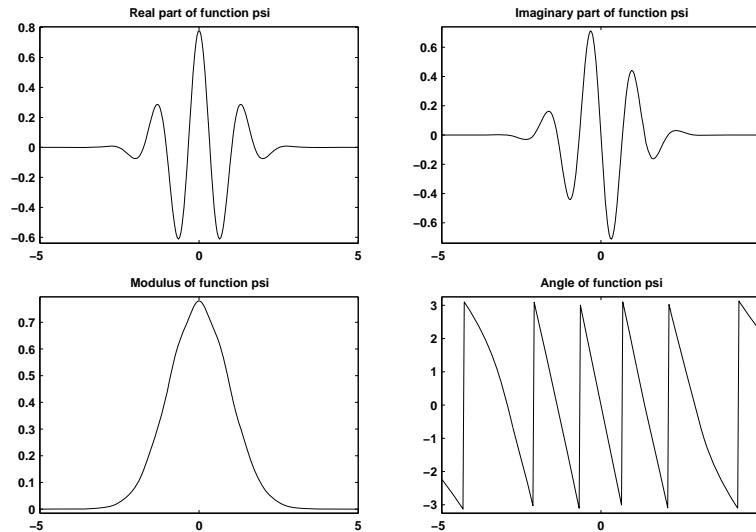
Some complex wavelet families are available in the toolbox.

### 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  $p^{\text{th}}$  derivative of  $f$ . The integer  $p$  is the parameter of this family and in the previous formula,  $C_p$  is such that  $\|f^{(p)}\|^2 = 1$  where  $f^{(p)}$  is the  $p^{\text{th}}$  derivative of  $f$ .

You can obtain a survey of the main properties of this family by typing `waveinfo('cgau')` from the MATLAB command line.



**Figure 6-22: Complex Gaussian Wavelet *cgau8***

### Complex Morlet Wavelets: *cmor*

See [Teo98] pages 62–65.

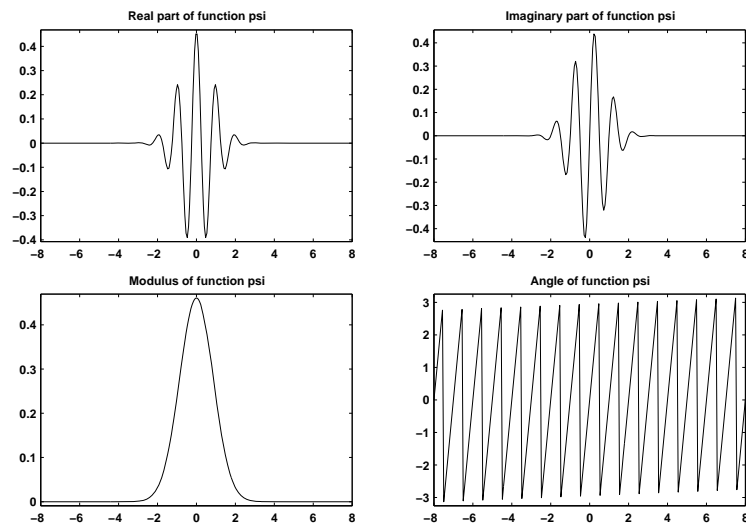
A complex Morlet wavelet is defined by

$$\psi(x) = \frac{1}{\sqrt{\pi f_b}} e^{2i\pi f_c x} e^{-\frac{x^2}{f_b}}$$

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.



**Figure 6-23: Complex Morlet Wavelet morl 1.5-1**

### Complex Frequency B-Spline Wavelets: *fbsp*

See [Teo98] pages 62–65.

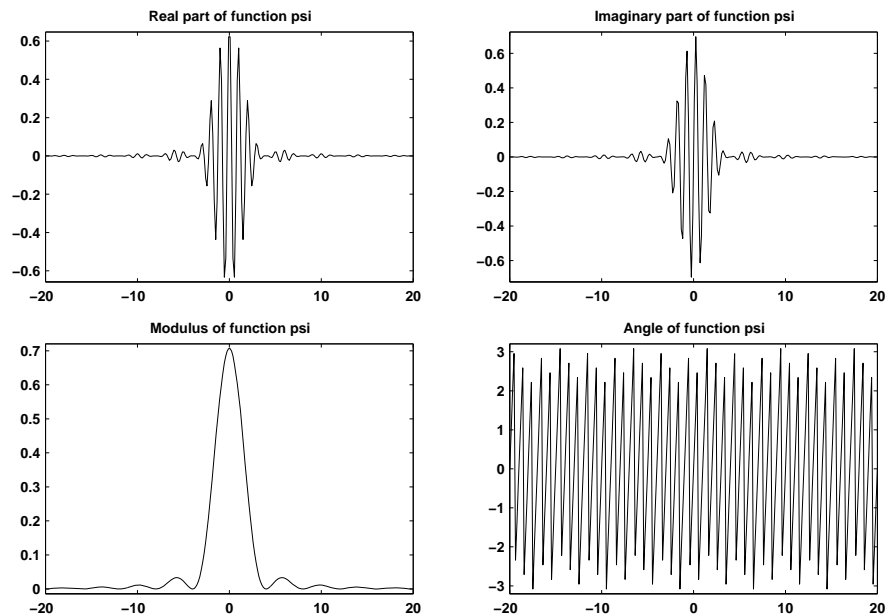
A complex frequency B-spline wavelet is defined by

$$\psi(x) = \sqrt{f_b} \left( \text{sinc}\left(\frac{f_b x}{m}\right) \right)^m e^{2i\pi f_c x}$$

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.



**Figure 6-24: 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.

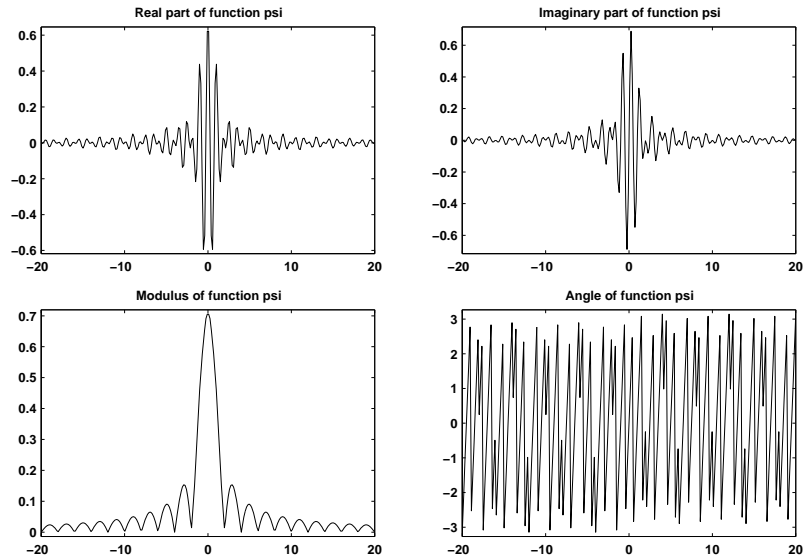
A complex Shannon wavelet is defined by

$$\psi(x) = \sqrt{f_b} \operatorname{sinc}(f_b x) e^{2i\pi f_c x}$$

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.



**Figure 6-25: Complex Shannon Wavelet shan 0.5-1**

## Summary of Wavelet Families and Associated Properties (Part 1)

Property	morl	mexh	meyr	haar	dbN	symN	coifN	biorNr.Nd
Crude	•	•						
Infinitely regular	•	•	•					
Arbitrary regularity					•	•	•	•
Compactly supported orthogonal				•	•	•	•	
Compactly supported biorthogonal								•
Symmetry	•	•	•	•				•
Asymmetry					•			
Near symmetry						•	•	
Arbitrary number of vanishing moments					•	•	•	•
Vanishing moments for $\phi$							•	
Existence of $\phi$			•	•	•	•	•	•
Orthogonal analysis			•	•	•	•	•	
Biorthogonal analysis			•	•	•	•	•	•
Exact reconstruction	≈	•	•	•	•	•	•	•
FIR filters				•	•	•	•	•
Continuous transform	•	•	•	•	•	•	•	•
Discrete transform				•	•	•	•	•

Property	morl	mexh	meyr	haar	dbN	symN	coifN	biorNr.Nd
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 6-63.

**Regularity** — See “What About the Regularity of a Wavelet  $\psi$ ?” on page 6-62.

**Orthogonal** — See “Details and Approximations” on page 6-15.

**Biorthogonal** — See “Biorthogonal Wavelet Pairs: biorNr.Nd” on page 6-76.

**Vanishing moments** — See “Suppressing Signals” on page 6-92.

**Exact reconstruction** — See “Reconstruction Filters” on page 1-30.

**Continuous** — See “Continuous Wavelet Transform” on page 1-15.

**Discrete** — See “Discrete Wavelet Transform” on page 1-24.

**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 biorthogonal	•						
Symmetry	•	•	•	•	•	•	•
Asymmetry							
Near symmetry							
Arbitrary number of vanishing moments	•						
Vanishing moments for $\phi$							
Existence of $\phi$	•						
Orthogonal analysis							
Biorthogonal analysis	•						
Exact reconstruction	•	•	$\approx$	•	•	•	•
FIR filters	•		•				
Continuous transform	•	•					
Discrete transform	•		•				
Fast algorithm	•		•				
Explicit expression	For splines	•		•	•	•	•



Property	rbioNr.Nd	gaus	dmey	cgau	cmor	fbsp	shan
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 6-63.

**Regularity** — See “What About the Regularity of a Wavelet  $\psi$ ?” on page 6-62.

**Orthogonal** — See “Details and Approximations” on page 6-15.

**Biorthogonal** — See “Biorthogonal Wavelet Pairs: biorNr.Nd” on page 6-76.

**Vanishing moments** — See “Suppressing Signals” on page 6-92.

**Exact reconstruction** — See “Reconstruction Filters” on page 1-30.

**Continuous** — See “Continuous Wavelet Transform” on page 1-15.

**Discrete** — See “Discrete Wavelet Transform” on page 1-24.

**FIR filters** — See “Filters Used to Calculate the DWT and IDWT” on page 6-19.

## Wavelet Applications: More Detail

Chapter 3, “Wavelet Applications,” and Chapter 4, “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 3-15, by suppressing a part of a signal the remainder may be highlighted.

Let  $\psi$  be a wavelet with at least  $k+1$  vanishing moments:

$$\text{for } j = 0, \dots, k, \int_{\mathbb{R}} x^j \psi(x) dx = 0$$

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  $[\alpha, \beta]$ , then  $C(a,b) = 0$  as long as the support of the function  $\frac{1}{\sqrt{a}} \psi\left(\frac{x-b}{a}\right)$  is included in  $[\alpha, \beta]$ . The suppression is local. Effects will appear on the edges of the segment.

Likewise, let us suppose that, on  $[\alpha, \beta]$  to which 0 belongs, we have the

expansion  $s(x) = [s(0) + xs'(0) + x^2 s^{(2)}(0) + \dots + x^k s^{(k)}(0)] + g(x)$ . 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  $\psi$  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” in Chapter 4 (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 M-file, some features of wavelet processing using coefficients (resulting plots can be found in Figure 6-26 on page 6-94).

```

% 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;

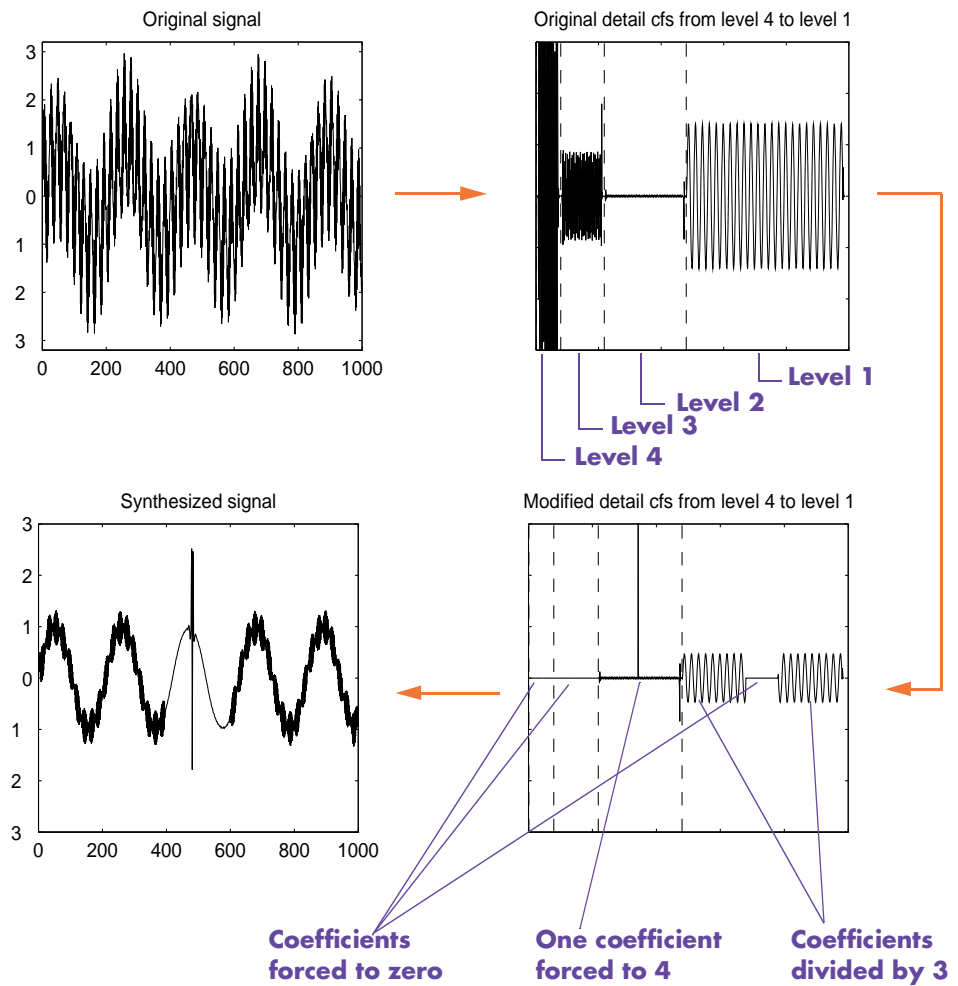
% shrink by dividing by 3.
newc(indd1) = c(indd1)/3;

% 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);

```



**Figure 6-26: 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 6-97 and “Data Compression” on page 6-110).

## 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” in Chapter 4), 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  $C(j,k)$ ,  $k \in Z$  is also a stationary, zero mean ARMA process whose characteristics depend on  $j$ .
- If  $s$  is a noise whose
  - Correlation function  $\rho$  is known, we know how to calculate the correlations of  $C(j,k)$  and  $C(j,k')$ .
  - Spectrum  $\hat{\rho}$  is known, we know how to calculate the spectrum of  $C(j,k)$ ,  $k \in Z$  and the cross spectrum of two different levels  $j$  and  $j'$ .

These results are easily established, since they can be deduced from the fact that the  $C(a,b)$  coefficients are calculated primarily by convolving  $\psi$  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  $\psi$  wavelet as if it was a signal:

$$U(a,b) = \int_R \frac{1}{\sqrt{a}} \psi\left(\frac{x-b}{a}\right) \psi(x) dx$$

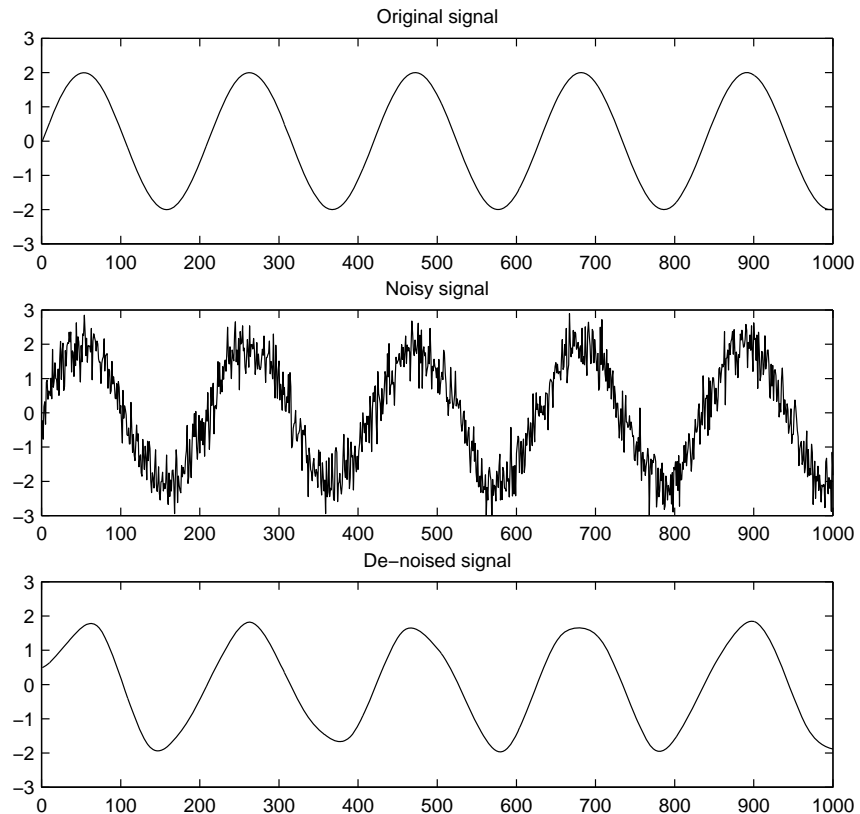
From the results for coefficients we deduce the properties of the details (and of the approximations), by using the formula

$$D_j(n) = \sum_{k \in Z} C(j,k) \psi_{j,k}(n)$$

where the  $C(j,k)$  coefficients are random variables and the functions  $\psi_{j,k}$  are not. If the support of  $\psi$  is finite, only a finite number of terms will be summed.

## De-Noising

This section discusses the problem of signal recovery from noisy data. This problem is easy to understand looking at the following simple example, where a slow sine is corrupted by a white noise.



**Figure 6-27: 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  $\sigma$  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?

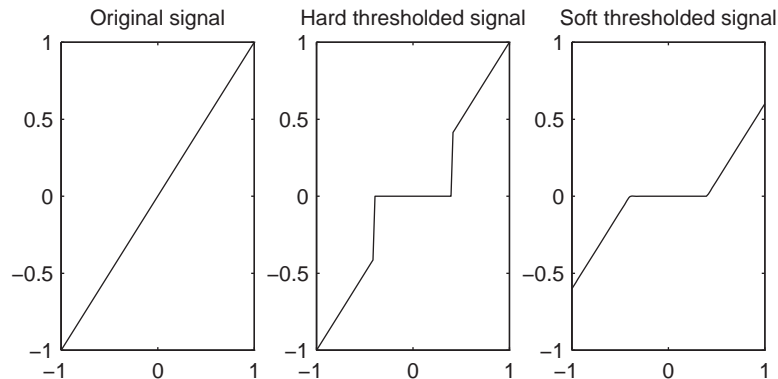
Thresholding can be done using the function

```
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 6-155).

Let us give a simple example.

```
y = linspace(-1,1,100);
thr = 0.4;
ythard = wthresh(y,'h',thr);
ytsoft = wthresh(y,'s',thr);
```



**Figure 6-28: 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  $\text{sign}(x)(|x| - t)$  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 Figure 6-28).

As can be seen in the comment of Figure 6-28 on page 6-99, the hard procedure creates discontinuities at  $x = \pm t$ , while the soft procedure does not.

### Threshold Selection Rules

According to the basic noise model, four threshold selection rules are implemented in the M-file `thselect`. Each rule corresponds to a `tptr` option in the command

```
thr = thselect(y,tptr)
```

which returns the threshold value.

Option	Threshold Selection Rule
'rigrsure'	Selection using principle of Stein's Unbiased Risk Estimate (SURE)
'sqtwo log'	Fixed form threshold equal to $\text{sqrt}(2 * \log(\text{length}(s)))$
'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 = 'sqtwo log'` uses a fixed form threshold yielding minimax performance multiplied by a small factor proportional to  $\log(\text{length}(s))$ .
- 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
's1n'	Basic model with unscaled noise
'm1n'	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  $\sigma$ . The median absolute deviation of the coefficients is a robust estimate of  $\sigma$ . 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 = 's1n'` 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  $\sigma_{lev}$  level by level.

This estimation is implemented in M-file `wnoisest`, directly handling the wavelet decomposition structure of the original signal  $s$ .

Option `scal = 'mln'` handles threshold rescaling using a level-dependent estimation of the level noise.

For a more general procedure, the `wdencomp` 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 = wdencomp(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.

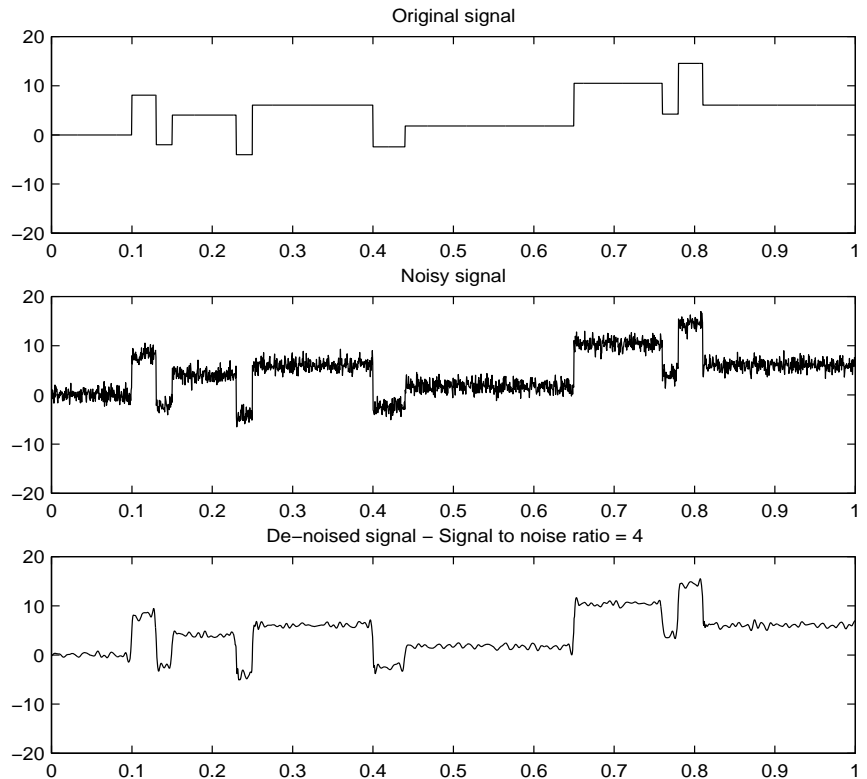
### De-Noising in Action

We begin with examples of one-dimensional de-noising methods with the first example credited to Donoho and Johnstone. You can use the following M-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');
```



**Figure 6-29: Blocks Signal De-Noising**

Since only a small number of large coefficients characterize the original signal, the method performs very well (see Figure 6-29). If you want to see more about how the thresholding works, use the GUI (see “De-Noising Signals” on page 3-18).

As a second example, let us try the method on the highly perturbed part of the electrical signal studied above.

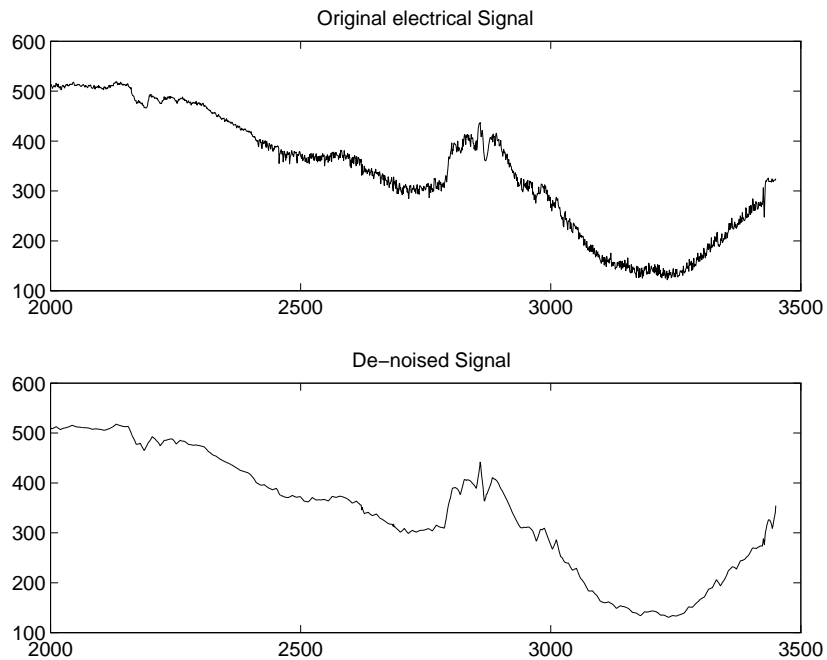
According to this previous analysis, let us use db3 wavelet and decompose at level 3.

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;
```



**Figure 6-30: Electrical Signal De-Noiseing**

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 `wdencomp`. As an example, you can use the following M-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] = ddencomp('den','wv',x);

% thr is equal to estimated_sigma*sqrt(log(prod(size(X))))
thr

thr =

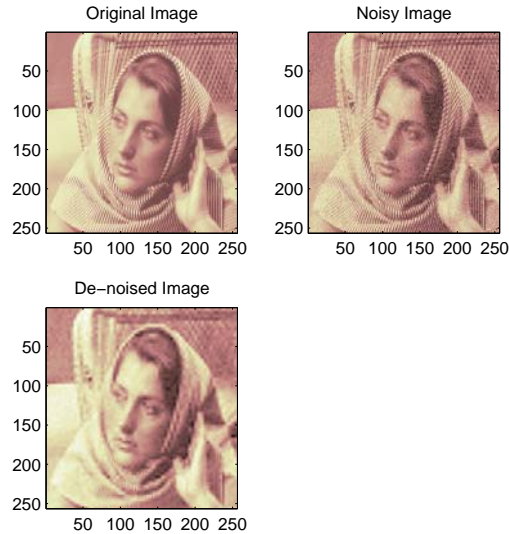
    107.6428

% De-noise image using global thresholding option.
xd = wdencomp('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-Noise Image')
```

The result shown below is acceptable.



**Figure 6-31: Image De-Noising**

### One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients

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” on page 2-1).

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 6-97).

$$s(n) = f(n) + \sigma e(n)$$

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 6-155).

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” on page 2-1) 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  $2 \leq i \leq 6$ , `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 M-files follow the work of the above mentioned researchers. More details can be found in Donoho's references in "References" on page 6-155 and in "More About the Thresholding Strategies" on page 6-125.

### **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.

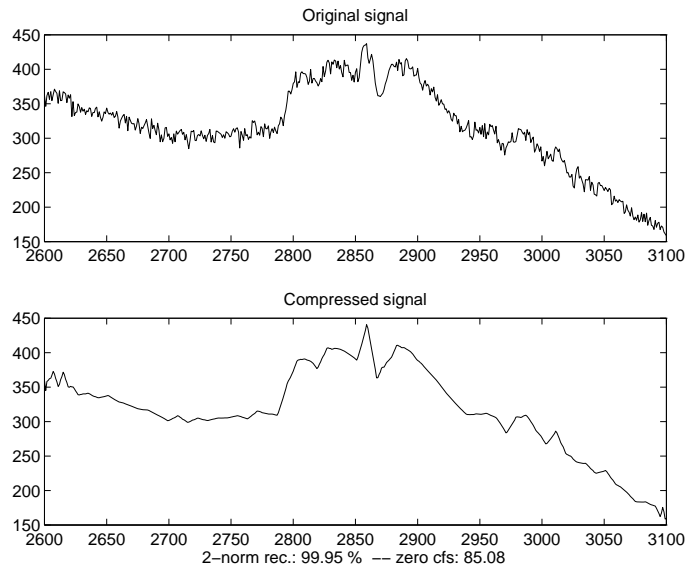
Thus, only a single parameter needs to be selected. The second approach consists of applying visually determined level-dependent thresholds.

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 Figure 6-32 on page 6-112) and for an image (see Figure 6-33 on page 6-113).

```
% 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,perf12] =
    wdencomp('gbl',c,l,w,n,thr,'h',keepapp);
```



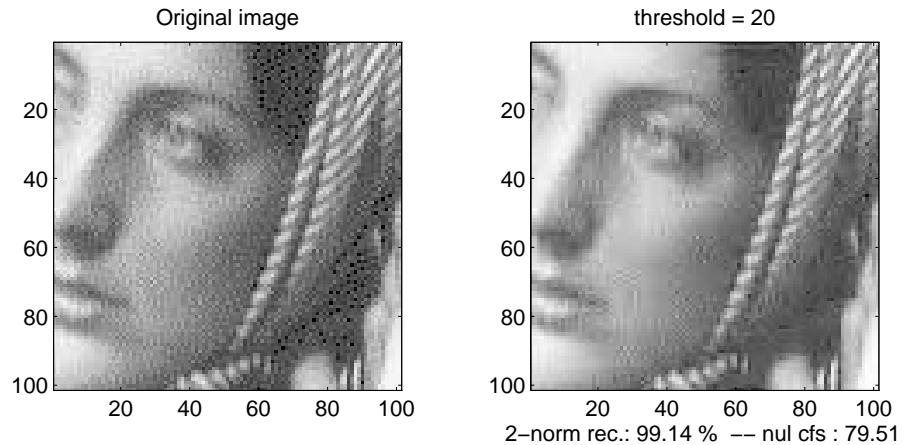
**Figure 6-32: Signal Compression**

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,perf12] =
    wdencomp('gbl',c,l,w,n,thr,'h',keepapp);
```



**Figure 6-33: 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

$$\frac{100 \cdot (\text{vector-norm}(\text{coeffs of the current decomposition}, 2))^2}{(\text{vector-norm}(\text{coeffs of the original decomposition}, 2))^2}$$

The *Number of zeros* in percentage is defined by

$$\frac{100 \cdot (\text{number of zeros of the current decomposition})}{(\text{number of coefficients})}$$

## 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  $(X(i), 1 \leq i \leq n)$  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  $1, 2, \dots, nb$  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 6-97).
- 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  $1, 2, \dots, nb$  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  $nb = \text{length}(X) / 4$ .

For more information, you can refer for example to [AntP98], [HarKPT98], and [Ogd97] in “References” on page 6-155.

### A More Technical Viewpoint.

Let us be a little more formal.

Let  $X_1, X_2, \dots, X_n$  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  $\phi$  and  $\psi$  with usual notations,  $J$  being an integer,

$$h = \sum_k a_{J,k} \phi_{J,k} + \sum_{j=-\infty}^J \sum_k d_{j,k} \psi_{j,k} = A_J + \sum_{j=-\infty}^J D_j$$

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  $E(\phi_{J,k}(X_i))$ , 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:

$$\hat{h} = \sum_k \hat{a}_{J,k} \phi_{J,k} + \sum_{j=J-j_0}^J \sum_k \hat{d}_{j,k} \mathbf{1}_{\{|\hat{d}_{j,k}| > t\}} \psi_{j,k}$$

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  $X_i$  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  $\hat{H}$ ,

$$\hat{d}_{j,k} = \frac{1}{n} \sum_{i=1}^n \psi_{j,k}(X_i) \approx \frac{1}{n} \sum_{r=1}^{nb} Yb(r) \psi_{j,k}(Xb(r)) \approx c \int \psi_{j,k}(x) \hat{H}(x) dx$$

where  $\frac{1}{c}$  is the length of each bin.

The signs  $\approx$  occur because we lose some information when using histogram instead of the values  $X_i$  and when approximating the integral.

The last  $\approx$  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  $\approx$  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  $H$ . 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  $i$ ,

$$Yb(i) = \frac{\text{sum}\{Y(j) \text{ such that } X(j) \text{ lies in bin } i\}}{\text{number}\{Y(j) \text{ such that } X(j) \text{ lies in bin } i\}}$$

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  $1, 2, \dots, nb$  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  $1, 2, \dots, nb$  onto  $Xb$  and interpolate  $f1$  for each bin in order to calculate  $f_{est}(x)$ .

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 6-155.

#### 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  $Y_i = f(X_i) + \varepsilon_i$  where  $(\varepsilon_i)_{1 \leq i \leq n}$  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  $(X_1, Y_1), \dots, (X_n, Y_n)$  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  $\frac{0}{0} = 0$ .

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  $Yb$ :

$$\hat{d}_{j,k} \approx \frac{1}{n} \sum_{l=1}^{nb} Yb(l) \psi_{j,k}(Xb(l))$$

$$\hat{a}_{J,k} \approx \frac{1}{n} \sum_{l=1}^{nb} Yb(l)\phi_{J,k}(Xb(l))$$

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 lowThe 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 lowThis 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 6-122.
  - 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 6-122.

### **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 6-122.
  - 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 6-122 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 6-155). 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
  - $M = 4 \cdot \text{prod}(S)$  in the 2-D case
- Scarce medium:
  - $M = 1.5 \cdot L$  in the 1-D case
  - $M = 4 \cdot 4 \cdot \text{prod}(S) / 3$  in the 2-D case
- Scarce low:
  - $M = 2 \cdot L$  in the 1-D case
  - $M = 4 \cdot 8 \cdot \text{prod}(S) / 3$  in the 2-D case

The related M-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 6-97) 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

$$t^* = \operatorname{argmin} \left[ -\operatorname{sum} \{c^2(k), k < t\} + 2vt \left( a + \log \left( \frac{n}{t} \right) \right); t = 1, \dots, n \right]$$

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,  $2.5 \leq a < 10$
- Penalized medium,  $1.5 < a < 2.5$
- Penalized low,  $1 < a < 2$

The related M-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 M-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 M-file is `wthrmngr` (for more information, see the corresponding reference page).

**Fixed Form.**

This thresholding strategy comes from Donoho-Johnstone (see “References” on page 6-155 and the 'sqrtwo1og' option of the `wden` function in “De-Noising” on page 6-97). 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 M-files are `ddencomp`, `thselect`, `wden`, `wdencomp`, 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 6-155).

The related M-files are `thselect`, `wden`, `wdencomp`, 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 6-155 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  $0.4 \times \max(|d(j)|)$ .

**3 By level 2:**

Level dependent thresholds  $T(j)$  are defined by  $0.8 \times \max(|d(j)|)$ .

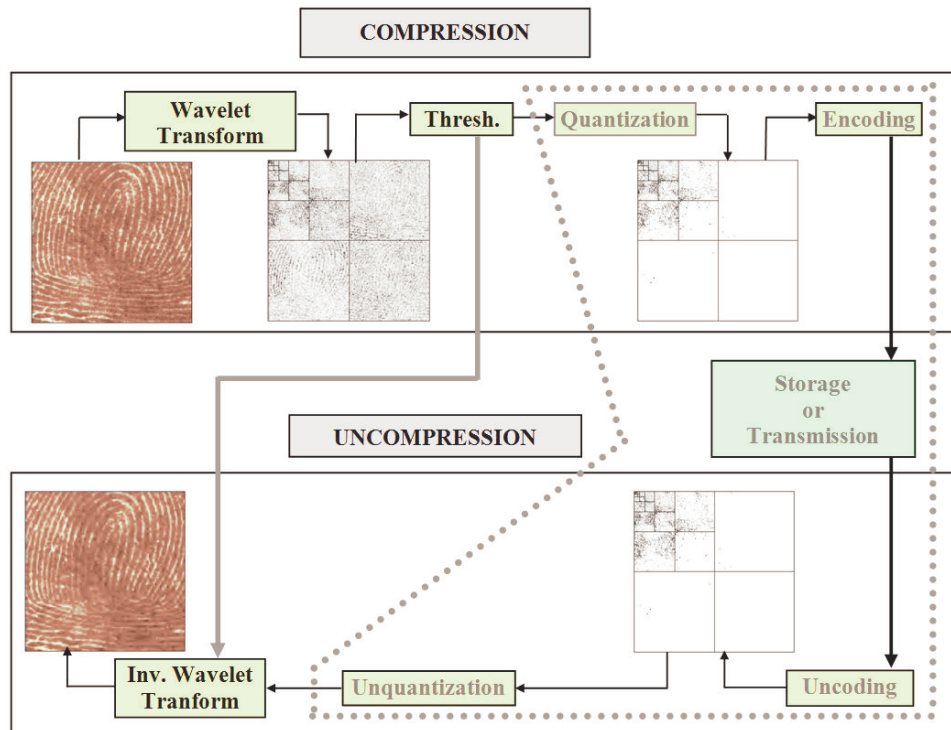
**4 By level 3:**

Level dependent thresholds  $T(j)$  are defined by  $a \times \max(|d(j)|)$ , where  $a$  is a sparsity parameter ( $0.2 < a \leq 1$ ,  $a = 0.6$  is the default).

## True Compression for Images

In “Data Compression” on page 6-110, 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.

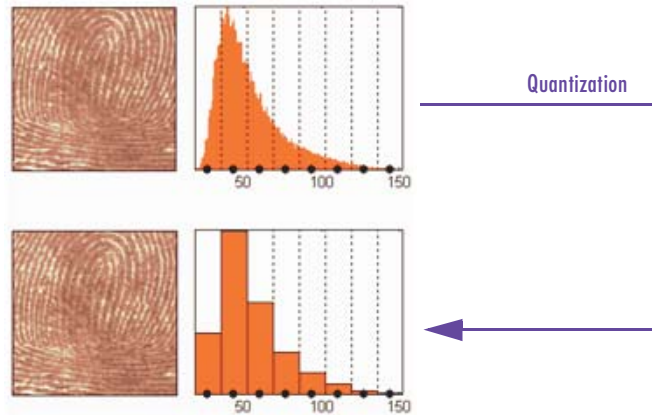


### Effects of Quantization

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.



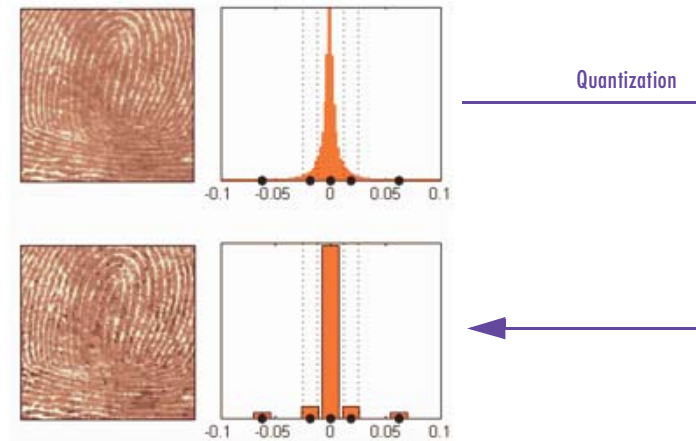
We can see on this figure:

- 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.

<b>MATLAB Name</b>	<b>Compression Method Name</b>
'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 6-155.

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 6-155.

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.

<b>MATLAB Name</b>	<b>Compression Method Name</b>
'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:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |X(i,j) - X_c(i,j)|^2$$

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” on page 2-317. 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 6-155.

## 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.

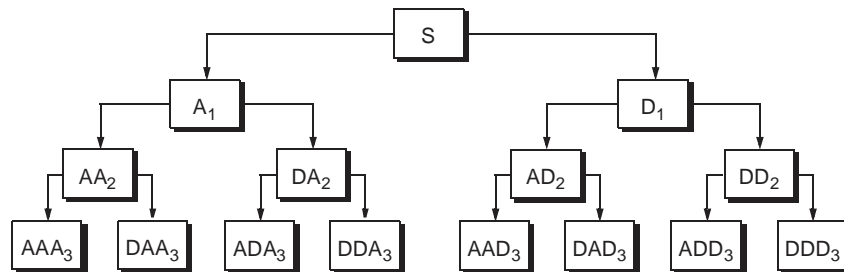
For a given orthogonal wavelet function, we generate a library of bases called *wavelet packet bases*. Each of these bases offers a particular way of coding signals, preserving global energy, and reconstructing exact features. The wavelet packets can be used for numerous expansions of a given signal. 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.



**Figure 6-34: 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 5-7.

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).

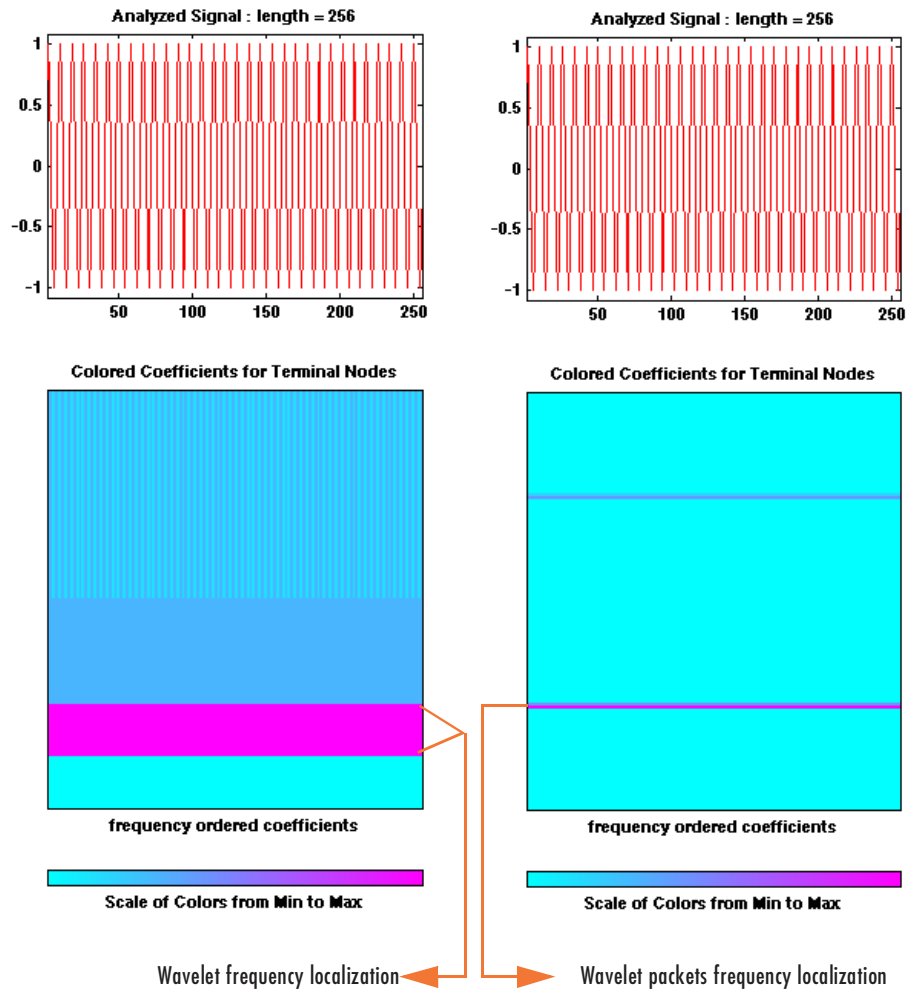
The **Colored Coefficients for Terminal Nodes** graph deserves explanation. In principle the graphic displays eight stripes. When using **Global + abs**, only four seem to be present. In fact, the eight are drawn. As the values of several coefficients are close to 0, the stripes are merged and only four 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.

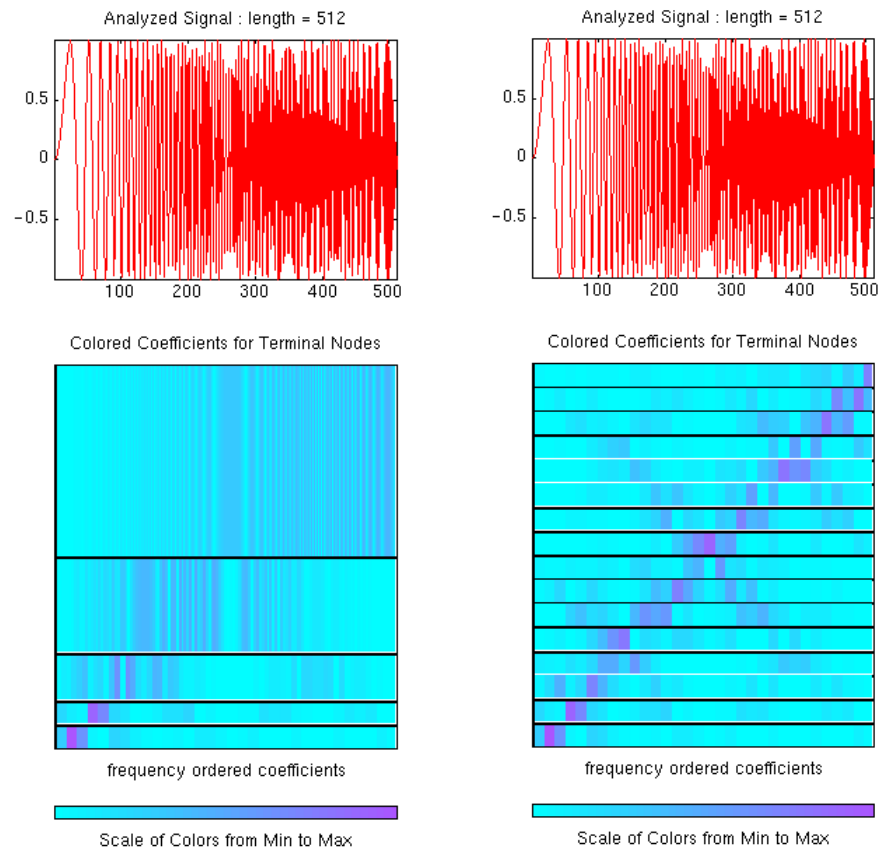




**Figure 6-35: Wavelets (Left) Versus Wavelet Packets (Right): A Sine Function**

### Example 2: Analyzing a Chirp Signal

The signal to be analyzed is a chirp: an oscillatory signal with increasing modulation  $\sin(250\pi t^2)$  sampled 512 times on  $[0, 1]$ . For this “linear” chirp, the derivative of the phase is linear. On the left of Figure 6-36, a wavelet analysis does not easily detect this time-frequency property of the signal. But on the right of Figure 6-36, the linear slope for the greatest wavelet packet coefficients in absolute value is obvious. The same experiment can be done with a “quadratic” chirp of the form  $\sin(k\pi t^3)$  in which the greatest wavelet packet coefficients exhibit a quadratic time frequency pattern.



**Figure 6-36: Wavelets (Left) Versus Wavelet Packets (Right): Damped Oscillations**

## Building Wavelet Packets

The computation scheme for wavelet packets generation is easy when using an orthogonal wavelet. We start with the two filters of length  $2N$ , where  $h(n)$  and  $g(n)$ , corresponding to the wavelet.

Now by induction let us define the following sequence of functions:

$$(W_n(x), n = 0, 1, 2, \dots)$$

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  $W_0(x) = \phi(x)$  is the scaling function and  $W_1(x) = \psi(x)$  is the wavelet function.

For example for the Haar wavelet we have

$$N = 1, h(0) = h(1) = \frac{1}{\sqrt{2}}$$

and

$$g(0) = -g(1) = \frac{1}{\sqrt{2}}$$

The equations become

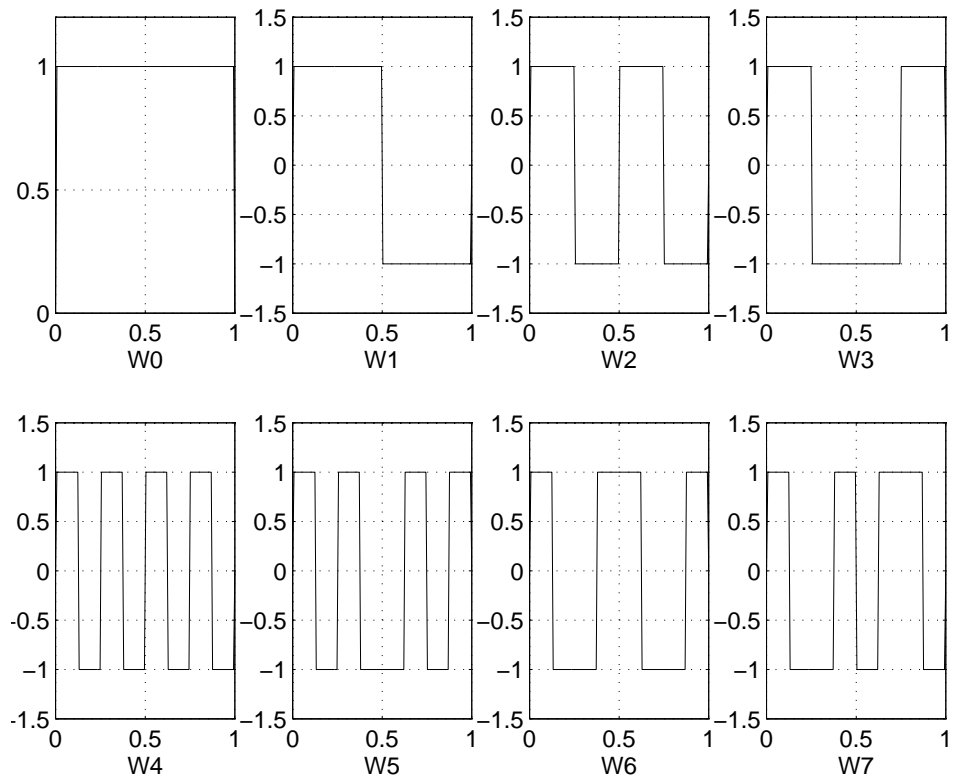
$$W_{2n}(x) = W_n(2x) + W_n(2x-1)$$

and

$$W_{2n+1}(x) = W_n(2x) - W_n(2x-1)$$

$W_0(x) = \phi(x)$  is the Haar scaling function and  $W_1(x) = \psi(x)$  is the Haar wavelet, both supported in  $[0, 1]$ . Then we can obtain  $W_{2n}$  by adding two  $1/2$ -scaled versions of  $W_n$  with distinct supports  $[0, 1/2]$  and  $[1/2, 1]$  and obtain  $W_{2n+1}$  by subtracting the same versions of  $W_n$ .

For  $n = 0$  to 7, we have the  $W$ -functions shown in Figure 6-37.



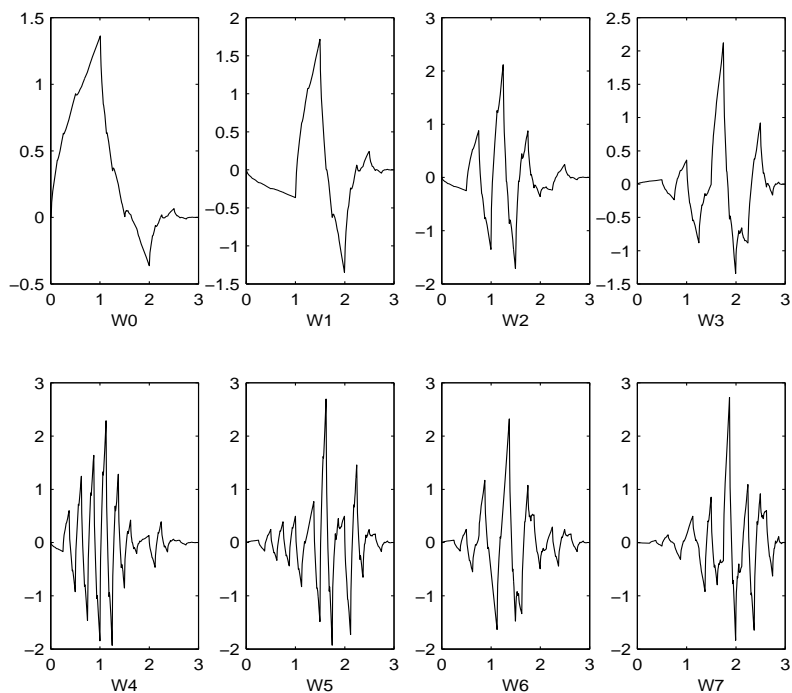
**Figure 6-37: Haar Wavelet Packets**

This can be obtained using the following command:

```
[wfun,xgrid] = wfun('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  $[0, 2N-1]$ . Figure 6-38 presents the system of  $W$ -functions for the original  $db2$  wavelet.



**Figure 6-38: db2 Wavelet Packets**

## Wavelet Packet Atoms

Starting from the functions  $(W_n(x), n \in N)$  and following the same line leading to orthogonal wavelets, we consider the three-indexed family of analyzing functions (the waveforms):

$$W_{j,n,k}(x) = 2^{-j/2} W_n(2^{-j}x - k)$$

where  $n \in N$  and  $(j,k) \in \mathbb{Z}^2$ .

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 Figure 6-37 on page 6-142 and Figure 6-38 on page 6-143, the naturally ordered  $W_n$  for  $n = 0, 1, \dots, 7$ , 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.

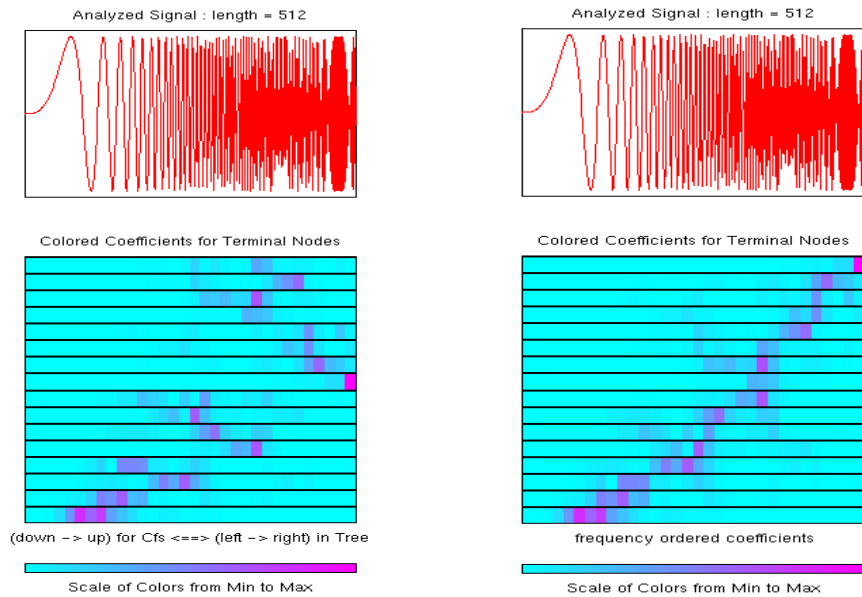
Natural order $n$	0	1	2	3	4	5	6	7
Number of zero crossings for $db1 W_n$	2	3	5	4	9	8	6	7

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 Figure 6-39) from the low frequencies at the bottom to the high frequencies at the top, rather than naturally ordered coefficients (on the left of Figure 6-39).



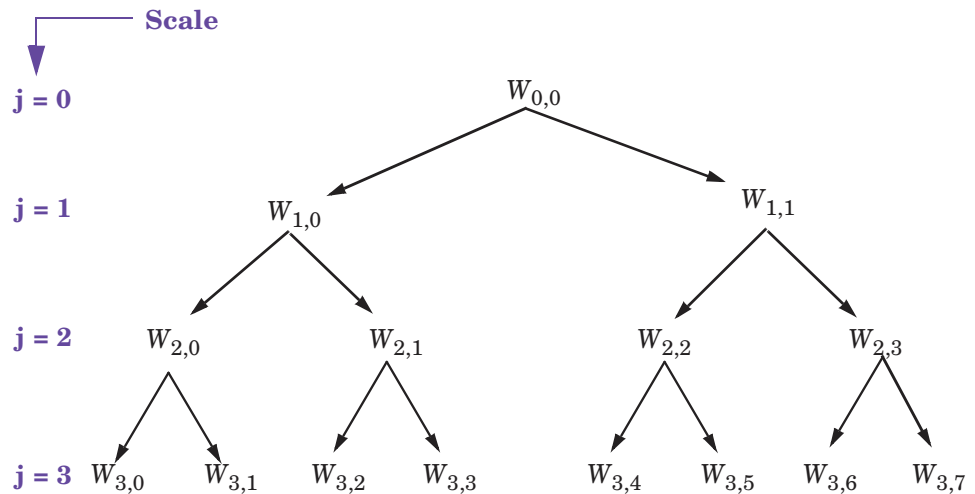
**Figure 6-39: Natural and Frequency Ordered Wavelet Packets Coefficients**

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  $W_{j,n} = (W_{j,n,k}(x), k \in \mathbb{Z})$  is the  $(j,n)$  wavelet packet. For positive values of integers  $j$  and  $n$ , wavelet packets are organized in trees. The tree in Figure 6-40 is created to give a maximum level decomposition equal to 3. For each scale  $j$ , the possible values of parameter  $n$  are  $0, 1, \dots, 2^j - 1$ .



**Figure 6-40: 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  $W_{0,0} = (\phi(x-k), k \in \mathbb{Z})$ , and  $W_{1,1} = \left(\psi\left(\frac{x}{2} - k\right), k \in \mathbb{Z}\right)$ .

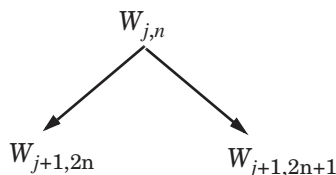
It turns out that the library of wavelet packet bases contains the wavelet basis and also several other bases. Let us have a look at some of those bases. More precisely, let  $V_0$  denote the space (spanned by the family  $W_{0,0}$ ) in which the signal to be analyzed lies; then  $(W_{d,1}; d \in \mathbb{N}, d \geq 1)$  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 Figure 6-41.



**Figure 6-41: 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  $\alpha$  different ways, where  $\alpha$  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  $0 \log(0) = 0$ .

- The concentration in  $l^p$  norm with  $1 \leq p$

$$E2(s_i) = |s_i|^p$$

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  $E4(s) = \# \{i \text{ such that } |s_i| > \varepsilon\}$  is the number of time instants when the signal is greater than a threshold  $\varepsilon$ .

These entropy functions are available using the wentropy M-file.

**Example 1: Compute Various Entropies.**

- 1 Generate a signal of energy equal to 1.

```
s = ones(1,16)*0.25;
```

- 2 Compute the Shannon entropy of  $s$ .

```
e1 = wentropy(s, 'shannon')
e1 = 2.7726
```

- 3 Compute the  $l^{1.5}$  entropy of  $s$ , equivalent to  $\text{norm}(s, 1.5)^{1.5}$ .

```
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,  $w_{11}$ , is zero; the entropy  $e_{11}$  is zero. Due to the additivity property the entropy of decomposition is given by  $e_{10}+e_{11}=2.0794$ . This has to be compared to the initial entropy  $e_{00}=2.7726$ . We have  $e_{10} + e_{11} < e_{00}$ , so the splitting is interesting.

- 5** Now split  $w_{10}$  (not  $w_{11}$  because the splitting of a null vector is without interest since the entropy is zero).

```
[w20,w21] = dwt(w10, 'db1');
```

- 6** We have  $w_{20}=0.5*\text{ones}(1,4)$  and  $w_{21}$  is zero. The entropy of the approximation level 2 is

```
e20 = wentropy(w20, 'shannon')
e20 = 1.3863
```

Again we have  $e_{20} + 0 < e_{10}$ , so splitting makes the entropy decrease.

- 7** Then

```
[w30,w31] = dwt(w20, 'db1');
e30 = wentropy(w30, 'shannon')
e30 = 0.6931
```

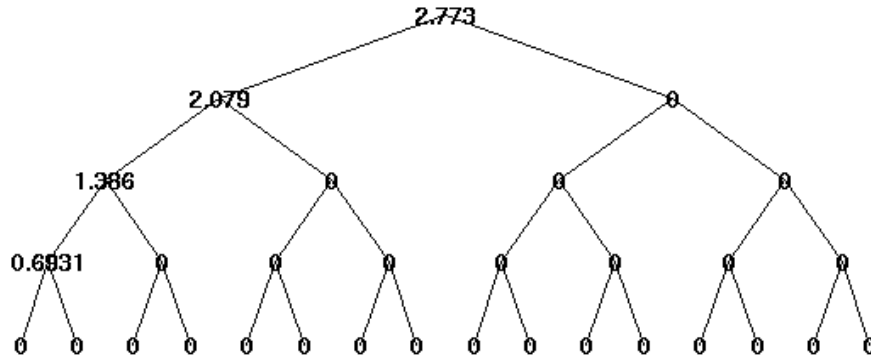
```
[w40,w41] = dwt(w30, 'db1')
w40 = 1.0000
w41 = 0
```

```
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  $e_{40}+e_{41}+e_{31}+e_{21}+e_{11} = 0$ ).

- 8 Perform wavelet packets decomposition of the signal  $s$  defined in example 1.  
`t = wpdec(s,4,'haar','shannon');`

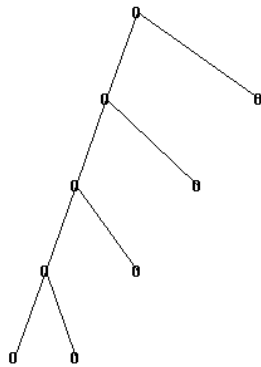
The wavelet packet tree in Figure 6-42 shows the nodes labeled with original entropy numbers.



**Figure 6-42: Entropy Values**

- 9 Compute the best tree.  
`bt = besttree(t);`

The best tree is shown in Figure 6-43. In this case, the best tree corresponds to the wavelet tree. The nodes are labeled with optimal entropy.



**Figure 6-43: 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.

<b>Decomposition Tree</b>	<b>Subtree Such That the Set of Leaves Is a Basis</b>
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  $E$ .

<b>Decompositions</b>	<b>Optimal Decomposition</b>	<b>Best-Level Decomposition</b>
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  $E_{opt}$  denotes the optimal entropy value).

Entropy Condition	Action on Tree and on Entropy Labeling
$E(\text{node}) \leq \sum_{c \text{ child of node}} E_{opt}(c)$	If ( $\text{node} \neq \text{root}$ ), merge and set $E_{opt}(\text{node}) = E(\text{node})$
$E(\text{node}) > \sum_{c \text{ child of node}} E_{opt}(c)$	Split and set $E_{opt}(\text{node}) = \sum_{c \text{ child of node}} E_{opt}(c)$

with the natural initial condition on the reference tree,  $E_{opt}(t) = E(t)$  for each terminal node  $t$ .

## Wavelet Packets 2-D Decomposition Structure

Exactly as in the wavelet decomposition case, the preceding one-dimensional framework can be extended to image analysis. Minor direct modifications lead to quaternary tree-related definitions. An example is shown Figure 6-44 for depth 2.

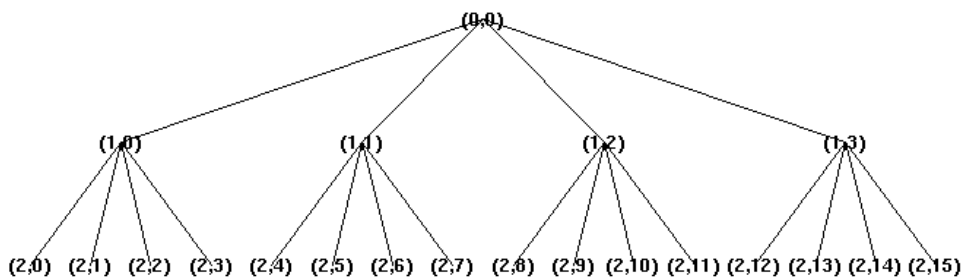


Figure 6-44: Quaternary Tree of Depth 2

### **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 `best tree` with an entropy function. For more details, see “Using Wavelet Packets” on page 5-1.



## 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.
- [**Ant095**] Antoniadis, A., G. Oppenheim, Eds. (1995), *Wavelets and statistics*, Lecture Notes in Statistics 103, Springer Verlag.
- [**AntP98**] Antoniadis A., D.T. Pham (1998), "Wavelet regression for random or irregular design," *Comp. Stat. and Data Analysis*, 28, pp. 353–369.
- [**AntG99**] Antoniadis, A., G. Gregoire (1999), "Density and Hazard rate estimation for right-censored data using wavelet methods," *J. R. Statist. Soc. B*, 61, 1, pp. 63–84.
- [**ArnABEM95**] Arneodo, A., F. Argoul, E. Bacry, J. Elezgaray, J.F. Muzy (1995), *Ondelettes, multifractales et turbulence*, Diderot Editeur, Paris.
- [**Bak95**] Bakshi, B. (1998), "Multiscale PCA with application to MSPC monitoring," *AIChE J.* 44, pp. 1596–1610.
- [**BarJM03**] Bardet, J.-M., G. Lang, G. Oppenheim, A. Philippe, S. Stoev, M.S. Taqqu (2003), "Generators of long-range dependence processes: a survey" *Theory and applications of long-range dependence*, Birkhäuser Boston, pp. 579–623.
- [**BirM97**] Birgé, L., P. Massart (1997), "From model selection to adaptive estimation," in D. Pollard (ed.), *Festschrift for L. Le Cam*, Springer, pp. 55–88.
- [**Bri95**] Brislawn, C.M. (1995), "Fingerprints to digital," *Notices of the AMS*. Vol. 42, pp. 1278–1283.
- [**Bur96**] Burke Hubbard, B. (1996), *The world according to wavelets*, AK Peters, Wellesley. The French original version is titled *Ondes et Ondelettes. La saga d'un outil mathématique*, Pour la Science, (1995).
- [**Chr06**] Christophe, E., C. Mailhes, P. Duhamel (2006), "Adaptation of zero-trees 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 Appl. Math.*, vol. 45, pp. 485–560.

[**CohDJV93**] Cohen, A., I. Daubechies, B. Jawerth, P. Vial (1993), “Multiresolution analysis, wavelets and fast wavelet transform on an interval,” *CRAS Paris, Ser. A*, t. 316, pp. 417–421.

[**CoID95**] Coifman, R.R., D.L. Donoho (1995), “Translation invariant de-noising,” *Lecture Notes in Statistics*, 103, pp. 125–150.

[**CoimW92**] Coifman, R.R., Y. Meyer, M.V. Wickerhauser (1992), “Wavelet analysis and signal processing,” in *Wavelets and their applications*, M.B. Ruskai et al. (Eds.), pp. 153–178, Jones and Bartlett.

[**CoIW92**] Coifman, R.R., M.V. Wickerhauser (1992), “Entropy-based algorithms for best basis selection,” *IEEE Trans. on Inf. Theory*, vol. 38, 2, pp. 713–718.

[**Dau92**] Daubechies, I. (1992), *Ten lectures on wavelets*, SIAM.

[**DevJL92**] DeVore, R.A., B. Jawerth, B.J. Lucier (1992), “Image compression through wavelet transform coding,” *IEEE Trans. on Inf. Theory*, vol. 38, 2, pp. 719–746.

[**Don93**] Donoho, D.L. (1993), “Progress in wavelet analysis and WVD: a ten minute tour,” in *Progress in wavelet analysis and applications*, Y. Meyer, S. Roques, pp. 109–128. Frontières Ed.

- [Don95] Donoho, D.L. (1995), “De-Noising by soft-thresholding,” *IEEE Trans. on Inf. Theory*, vol. 41, 3, pp. 613–627.
- [DonJ94a] Donoho, D.L., I.M. Johnstone (1994), “Ideal spatial adaptation by wavelet shrinkage,” *Biometrika*, vol. 81, pp. 425–455.
- [DonJ94b] Donoho, D.L., I.M. Johnstone (1994), “Ideal de-noising in an orthonormal basis chosen from a library of bases,” *CRAS Paris, Ser I*, t. 319, pp. 1317–1322.
- [DonJKP95] Donoho, D.L., I.M. Johnstone, G. Kerkyacharian, D. Picard (1995), “Wavelet shrinkage: asymptopia,” *Jour. Roy. Stat. Soc., series B*, vol. 57, no. 2, pp. 301–369.
- [DonJKP96] Donoho, D.L., I.M. Johnstone, G. Kerkyacharian, D. Picard (1996), “Density estimation by wavelet thresholding,” *Annals of Stat.*, 24, pp. 508–539.
- [Fla92] Flandrin, P. (1992), “Wavelet analysis and synthesis of fractional Brownian motion,” *IEEE Trans. on Inf. Th.*, 38, pp. 910–917.
- [HalPKP97] Hall, P., S. Penev, G. Kerkyacharian, D. Picard (1997), “Numerical performance of block thresholded wavelet estimators,” *Stat. and Computing*, 7, pp. 115–124.
- [HarKPT98] Hardle, W., G. Kerkyacharian, D. Picard, A. Tsybakov (1998), *Wavelets, approximation and statistical applications*, Lecture Notes in Statistics, 129, Springer Verlag.
- [Ist94] Istas, J., G. Lang (1994), “Quadratic variations and estimation of the local Hölder index of a Gaussian process,” *Ann. Inst. Poincaré*, 33, pp. 407–436.
- [KahL95] Kahane, J.P., P.G. Lemarié (1995), *Fourier series and wavelets*, Gordon and Research Publishers, Studies in the Development of Modern Mathematics, vol 3.
- [Kai94] Kaiser, G. (1994), *A friendly guide to wavelets*, Birkhäuser.
- [Lav99] Lavielle, M. (1999), “Detection of multiple changes in a sequence of dependent variables,” *Stoch. Proc. and their Applications*, 83, 2, pp. 79–102.
- [Lem90] Lemarié, P.G., Ed., (1990), *Les ondelettes en 1989, Lecture Notes in Mathematics*, Springer Verlag.

- [**Mal89**] Mallat, S. (1989), “A theory for multiresolution signal decomposition: the wavelet representation,” *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, pp. 674–693.
- [**Mal98**] Mallat, S. (1998), *A wavelet tour of signal processing*, Academic Press.
- [**Mey90**] Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*), Cambridge Univ. Press, 1993.
- [**Mey93**] Meyer, Y. (1993), *Les ondelettes. Algorithmes et applications*, Colin Ed., Paris, 2nd edition. (English translation: *Wavelets: algorithms and applications*, SIAM).
- [**MeyR93**] Meyer, Y., S. Roques, Eds. (1993), *Progress in wavelet analysis and applications*, Frontières Ed.
- [**MisMOP93a**] Misiti, M., Y. Misiti, G. Oppenheim, J.M. Poggi (1993a), “Analyse de signaux classiques par décomposition en ondelettes,” *Revue de Statistique Appliquée*, vol. XLI, no. 4, pp. 5–32.
- [**MisMOP93b**] Misiti, M., Y. Misiti, G. Oppenheim, J.M. Poggi (1993b), “Ondelettes en statistique et traitement du signal,” *Revue de Statistique Appliquée*, vol. XLI, no. 4, pp. 33–43.
- [**MisMOP94**] Misiti, M., Y. Misiti, G. Oppenheim, J.M. Poggi (1994), “Décomposition en ondelettes et méthodes comparatives: étude d’une courbe de charge électrique,” *Revue de Statistique Appliquée*, vol. XLII, no. 2, pp. 57–77.
- [**MisMOP03**] Misiti, M., Y. Misiti, G. Oppenheim, J.-M. Poggi (2003), “Les ondelettes et leurs applications,” Hermes.
- [**MisMOP07**] Misiti, M., Y. Misiti, G. Oppenheim, J.-M. Poggi (2007), *Wavelets and their applications*, ISTE DSP Series.
- [**NasS95**] Nason, G.P., B.W. Silverman (1995), “The stationary wavelet transform and some statistical applications,” *Lecture Notes in Statistics*, 103, pp. 281–299.
- [**Ogd97**] Ogden, R.T. (1997), *Essential wavelets for statistical applications and data analysis*, Birkhäuser.
- [**PesKC96**] Pesquet, J.C., H. Krim, H. Carfatan (1996), “Time-invariant orthonormal wavelet representations,” *IEEE Trans. Sign. Proc.*, vol. 44, 8, pp. 1964–1970.

- [**Sai96**] Said A., W.A. Pearlman (1996), “A new, fast, and efficient image codec based on set partitioning in hierarchical trees,” *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 6, No. 3, pp. 243–250.
- [**Sha93**] Shapiro J.M. (1993), “Embedded image coding using zerotrees of wavelet coefficients,” *IEEE Trans. Signal Proc.*, Vol. 41, No. 12, pp. 3445–3462.
- [**StrN96**] Strang, G., T. Nguyen (1996), *Wavelets and filter banks*, Wellesley-Cambridge Press.
- [**Swe98**] Sweldens, W. (1998), “The Lifting Scheme: a Construction of Second Generation of Wavelets,” *SIAM J. Math. Anal.*, 29 (2), pp. 511–546.
- [**Teo98**] Teolis, A. (1998), *Computational signal processing with wavelets*, Birkhäuser.
- [**VetK95**] Vetterli, M., J. Kovacevic (1995), *Wavelets and subband coding*, Prentice Hall.
- [**Wal99**] Walker, J.S. (1999), “Wavelet-Based Image Compression,” University of Wisconsin, Eau Claire, Wisconsin, USA, , Sub-chapter of CRC Press book: *Transform and Data Compression. A Primer on Wavelets and Their Scientific Applications*. A second edition is published in 2008.
- [**Wic91**] Wickerhauser, M.V. (1991), “INRIA lectures on wavelet packet algorithms,” *Proceedings ondelettes et paquets d’ondes*, 17–21 June, Rocquencourt France, pp. 31–99.
- [**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/>



# Adding Your Own Wavelets

---

This chapter discusses how to add your own wavelet families to the toolbox.

- [Preparing to Add a New Wavelet Family \(p. 7-2\)](#)
- [Adding a New Wavelet Family \(p. 7-8\)](#)
- [After Adding a New Wavelet Family \(p. 7-16\)](#)

## 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 M-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  $w_r$  and  $w_d$ , 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 M-File

The `wavemngr` command requires a `file` argument, which is a string containing a MAT-file or M-file name.

*If a family contains many wavelets*, an M-file must be defined and must be of a specific form that depends on the wavelet type. The specific M-file formats are described in the remainder of this section.

*If a family contains a single wavelet*, then a MAT-file can be defined for wavelets of type 1. It must have the wavelet family short name (`fsn`) argument as its name and must contain a single variable whose name is `fsn` and whose value is the scaling filter. An M-file can also be defined as discussed below.

### Type 1 (Orthogonal with FIR Filter)

The syntax of the first line in the M-file must be

```
function w = file(wname)
```

where the input argument `wname` is a string containing the wavelet name, and the output argument `w` is the corresponding scaling filter.

The filter `w` must be of even length; otherwise, it is zero-padded by the toolbox.

For predefined wavelets, the scaling filter is of sum 1. For a new wavelet, the normalization is free (except 0 of course) since the toolbox uses a suitably normalized version of this filter.

Examples of such M-files for predefined wavelets are `dbwavf.m` for Daubechies, `coifwavf.m` for coiflets, and `symwavf.m` for symlets.

### Type 2 (Biorthogonal with FIR Filter)

The syntax of the first line in the M-file must be

```
function [wr,wd] = file(wname)
```

where the input argument `wname` is a string containing the wavelet name and the output arguments `wr` and `wd` are the corresponding reconstruction and decomposition scaling filters, respectively.

The filters `wr` and `wd` must be of the same even length. In general, initial biorthogonal filters do not meet these requirements, so they are zero-padded by the toolbox.

For predefined wavelets, the scaling filters are of sum 1. For a new wavelet, the normalization is free (except 0 of course) since the toolbox uses a suitably normalized version of these filters.

The M-file `biorwavf.m` (for `BiorSplines`) is an example of an M-file for a type 2 predefined wavelet family.

### Type 3 (Orthogonal with Scale Function)

The syntax of the first line in the M-file must be

```
function [phi,psi,t] = file(lb,ub,n,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 M-file `meyer.m` is an example of an M-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 M-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 M-files for predefined wavelet families are `mexihat.m` (for Mexican\_hat) and `morlet.m` (for Morlet).

Examples of type 5 M-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 `fbpwavf` 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	<code>wavemngr('add','Ndaubechies','ndb',1,'1 2 3 4 5','dbwavf');</code>
1	<code>wavemngr('add','Ndaubechies','ndb',1,'1 2 3 4 5 **','dbwavf');</code>
2	<code>wavemngr('add','Nbiorwavf','nbio',2,'1.1 1.3','biorwavf');</code>
3	<code>wavemngr('add','Nmeyer','nmey',3,'','meyer',[-8,8]);</code>
4	<code>wavemngr('add','Nmorlet','nmor',4,'','morlet',[-4,4]).</code>

### 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 6-155).

---

**Note** The M-files used in this example can be found in the `wavedemo` directory.

---

The full family name is `Binlets`.

The short name of the wavelet family is `binl`.

The wavelet type is 2 (Biorthogonal with FIR filters).

The order of the wavelet within the family is 7.9 (we just use one in this example).

The M-file used to generate the filters is `binlwavf.m`

Then to add the new wavelet, type

```
% Add new family of biorthogonal wavelets.
    wavemngr('add','Binlets','binl',2,'7.9','binlwavf')

% List wavelets families.
    wavemngr('read')
```

```
ans =
```

```
=====
Haar                haar
Daubechies          db
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 M-file to generate the filters (binlwavf.m) is

```
function [Rf,Df] = binlwavf(wname)
%BINLWAVF Biorthogonal wavelet filters (Binlets).
%   [RF,DF] = BINLWAVF(W) returns two scaling filters
%   associated with the biorthogonal wavelet specified
%   by the string W.
%   W = 'binlNr.Nd' where possible values for Nr and Nd are:
%           Nr = 7  Nd = 9
%   The output arguments are filters:
%           RF is the reconstruction filter
%           DF is the decomposition filter
%
% Check arguments.
if errargin('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 M-files used in this example can be found in the wavedemo directory.

---

```
% 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                 mor1
=====
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 directory: the two ASCII files `wavelets.asc` and `wavelets.prv`, and the MAT-file `wavelets.inf`.

If you want to use your own extended wavelet families with the Wavelet Toolbox™ software, you should

- 1** Create a new directory specifically to hold the wavelet extension files.
- 2** Move the previously mentioned files into this new directory.
- 3** Prepend this directory to the MATLAB® directory search path (see the reference entry for the `path` command).
- 4** Use this same directory for subsequent modifications. Allowing many wavelet extension files to proliferate in different directories may lead to unpredictable results.
- 5** Define an M-file called `<fsn>info.m` (for example, see `dbinfo.m` or `morlinfo.m`).

This file will be associated automatically with the **Wavelet Family** button in the Wavelet Display option of the graphical tools.

# Function Reference

---

- General Wavelet Functions (p. 8-18)
- Wavelet Families (p. 8-19)
- 1-D Continuous Wavelets (p. 8-20)
- 1-D Discrete Wavelets (p. 8-21)
- 2-D Discrete Wavelets (p. 8-22)
- Wavelet Packet Algorithms (p. 8-23)
- Discrete Stationary Wavelet Transform Algorithms (p. 8-24)
- Lifting Wavelet Transforms (p. 8-25)
- De-Noising and Compression (p. 8-26)
- 1-D Multisignal Wavelet Analysis (p. 8-27)
- True Compression for Images (p. 8-28)
- Other Wavelet Applications (p. 8-29)
- Tree Management Utilities (p. 8-30)
- General Utilities and Demos (p. 8-32)

## General Wavelet Functions

<code>biorfilt</code>	Biorthogonal wavelet filter set
<code>centfrq</code>	Wavelet center frequency
<code>dyaddown</code>	Dyadic downsampling
<code>dyadup</code>	Dyadic upsampling
<code>intwave</code>	Integrate wavelet function $\psi$
<code>orthfilt</code>	Orthogonal wavelet filter set
<code>qmf</code>	Quadrature mirror filter
<code>scal2frq</code>	Scale to frequency
<code>wavefun</code>	Wavelet and scaling functions
<code>wavefun2</code>	Wavelet and scaling functions 2-D
<code>wavemngr</code>	Wavelet manager
<code>wfilters</code>	Wavelet filters
<code>wmaxlev</code>	Maximum wavelet decomposition level



## Wavelet Families

<code>biorwavf</code>	Biorthogonal spline wavelet filters
<code>cgauwavf</code>	Complex Gaussian wavelet
<code>cmorwavf</code>	Complex Morlet wavelet
<code>coifwavf</code>	Coiflet wavelet filter
<code>dbaux</code>	Daubechies wavelet filter computation
<code>dbwavf</code>	Daubechies wavelet filter
<code>fbspwavf</code>	Complex frequency B-spline wavelet
<code>gauswavf</code>	Gaussian wavelet
<code>mexihat</code>	Mexican hat wavelet
<code>meyer</code>	Meyer wavelet
<code>meyeraux</code>	Meyer wavelet auxiliary function
<code>morlet</code>	Morlet wavelet
<code>rbiowavf</code>	Reverse biorthogonal spline wavelet filters
<code>shanwavf</code>	Complex Shannon wavelet
<code>symaux</code>	Symlet wavelet filter computation
<code>symwavf</code>	Symlet wavelet filter

## **1-D Continuous Wavelets**

<code>cwt</code>	Real or complex continuous 1-D wavelet coefficients
<code>cwtext</code>	Real or complex continuous 1-D wavelet coefficients using extension parameter
<code>pat2cwav</code>	Build wavelet from pattern
<code>wscalogram</code>	Scalogram for continuous wavelet transform

## 1-D Discrete Wavelets

appcoef	1-D approximation coefficients
detcoef	1-D detail coefficients
dwt	Single-level discrete 1-D wavelet transform
dwtmode	Discrete wavelet transform extension mode
idwt	Single-level inverse discrete 1-D wavelet transform
upcoef	Direct reconstruction from 1-D wavelet coefficients
upwlev	Single-level reconstruction of 1-D wavelet decomposition
wavedec	Multilevel 1-D wavelet decomposition
waverec	Multilevel 1-D wavelet reconstruction
wenergy	Energy for 1-D wavelet decomposition
wrcoef	Reconstruct single branch from 1-D wavelet coefficients

## 2-D Discrete Wavelets

appcoef2	2-D approximation coefficients
detcoef2	2-D detail coefficients
dwt2	Single-level discrete 2-D wavelet transform
dwtmode	Discrete wavelet transform extension mode
idwt2	Single-level inverse discrete 2-D wavelet transform
upcoef2	Direct reconstruction from 2-D wavelet coefficients
upwlev2	Single-level reconstruction of 2-D wavelet decomposition
wavedec2	Multilevel 2-D wavelet decomposition
waverec2	Multilevel 2-D wavelet reconstruction
wenergy2	Energy for 2-D wavelet decomposition
wrcoef2	Reconstruct single branch from 2-D wavelet coefficients

## Wavelet Packet Algorithms

bestlevt	Best level tree wavelet packet analysis
besttree	Best tree wavelet packet analysis
entrupd	Entropy update (wavelet packet)
wenergy	Energy for wavelet packet decomposition
wentropy	Entropy (wavelet packet)
wp2wtree	Extract wavelet tree from wavelet packet tree
wpcoef	Wavelet packet coefficients
wpcutree	Cut wavelet packet tree
wpdec	Wavelet packet decomposition 1-D
wpdec2	Wavelet packet decomposition 2-D
wpfun	Wavelet packet functions
wpjoin	Recompose wavelet packet
wprcoef	Reconstruct wavelet packet coefficients
wprec	Wavelet packet reconstruction 1-D
wprec2	Wavelet packet reconstruction 2-D
wpsplt	Split (decompose) wavelet packet

## **Discrete Stationary Wavelet Transform Algorithms**

iswt	Inverse discrete stationary wavelet transform 1-D
iswt2	Inverse discrete stationary wavelet transform 2-D
swt	Discrete stationary wavelet transform 1-D
swt2	Discrete stationary wavelet transform 2-D

## Lifting Wavelet Transforms

<code>addlift</code>	Add lifting steps to lifting scheme
<code>bswfun</code>	Biorthogonal scaling and wavelet functions
<code>displs</code>	Display lifting scheme
<code>filt2ls</code>	Transform quadruplet of filters to lifting scheme
<code>ilwt</code>	Inverse 1-D lifting wavelet transform
<code>ilwt2</code>	Inverse 2-D lifting wavelet transform
<code>liftfilt</code>	Apply elementary lifting steps on quadruplet of filters
<code>liftwave</code>	Lifting schemes
<code>laurmat</code>	Laurent matrices constructor
<code>laurpoly</code>	Laurent polynomials constructor
<code>ls2filt</code>	Transform lifting scheme to quadruplet of filters
<code>lsinfo</code>	Lifting schemes information
<code>lwt</code>	1-D lifting wavelet transform
<code>lwt2</code>	2-D lifting wavelet transform
<code>lwtcoef</code>	Extract or reconstruct 1-D LWT wavelet coefficients
<code>lwtcoef2</code>	Extract or reconstruct 2-D LWT wavelet coefficients
<code>wave2lp</code>	Laurent polynomials associated with wavelet
<code>wavenames</code>	Wavelet names for LWT

## De-Noising and Compression

<code>ddencmp</code>	Default values for de-noising or compression
<code>thselect</code>	Threshold selection for de-noising
<code>wbmpen</code>	Penalized threshold for wavelet 1-D or 2-D de-noising
<code>wdcbm</code>	Thresholds for wavelet 1-D using Birge-Massart strategy
<code>wdcbm2</code>	Thresholds for wavelet 2-D using Birge-Massart strategy
<code>wden</code>	Automatic 1-D de-noising
<code>wdencmp</code>	De-noising or compression
<code>wnoise</code>	Noisy wavelet test data
<code>wnoisest</code>	Estimate noise of 1-D wavelet coefficients
<code>wpbmpen</code>	Penalized threshold for wavelet packet de-noising
<code>wpdencmp</code>	De-noising or compression using wavelet packets
<code>wpthcoef</code>	Wavelet packet coefficients thresholding
<code>wthcoef</code>	Wavelet coefficient thresholding 1-D
<code>wthcoef2</code>	Wavelet coefficient thresholding 2-D
<code>wthresh</code>	Soft or hard thresholding
<code>wthrmngr</code>	Threshold settings manager



## 1-D Multisignal Wavelet Analysis

wmspca	Multiscale Principal Components Analysis
wmulden	Wavelet multivariate de-noising
chgwdceccfs	Change multisignal 1-D decomposition coefficients
mdwtcluster	Multisignal 1-D clustering
mdwtdec	Multisignal 1-D wavelet decomposition
mdwtrec	Multisignal 1-D wavelet reconstruction
mswcmp	Multisignal 1-D compression using wavelets
mswcmpscr	Multisignal 1-D wavelet compression scores
mswcmptp	Multisignal 1-D compression thresholds and performances
mswden	Multisignal 1-D denoising using wavelets
mswthresh	Perform multisignal 1-D thresholding
wdcenergy	Multisignal 1-D decomposition energy distribution

## **True Compression for Image<sub>s</sub>**

wcompress      True compression of images using wavelets

## Other Wavelet Applications

wfbm	Fractional Brownian motion synthesis
wfbmesti	Parameter estimation of fractional Brownian motion
wfusing	Fusion of two images
wfusmat	Fusion of two matrices or arrays

## Tree Management Utilities

<code>allnodes</code>	Tree nodes
<code>cfs2wpt</code>	Wavelet packet tree construction from coefficients
<code>depo2ind</code>	Node depth-position to node index
<code>disp</code>	WPTREE information
<code>drawtree</code>	Draw wavelet packet decomposition tree (GUI)
<code>dtree</code>	DTREE constructor
<code>get</code>	WPTREE contents
<code>ind2depo</code>	Node index to node depth-position
<code>isnode</code>	Existing node test
<code>istnode</code>	Terminal nodes indices test
<code>leaves</code>	Determine terminal nodes
<code>nodeasc</code>	Node ascendants
<code>nodedesc</code>	Node descendants
<code>nodejoin</code>	Recompose node
<code>nodepar</code>	Node parent
<code>nodesplt</code>	Split (decompose) node
<code>noleaves</code>	Determine nonterminal nodes
<code>ntnode</code>	Number of terminal nodes
<code>ntree</code>	NTREE constructor
<code>plot</code>	Plot tree object
<code>read</code>	Read values of WPTREE
<code>readtree</code>	Read wavelet packet decomposition tree from figure
<code>set</code>	WPTREE field contents
<code>tnodes</code>	Determine terminal nodes
<code>treedpth</code>	Tree depth

<code>treeord</code>	Tree order
<code>wptree</code>	Constructor for the class <code>WPTREE</code>
<code>wpviewcf</code>	Plot wavelet packets colored coefficients
<code>write</code>	Write values in <code>WPTREE</code> fields
<code>wtbo</code>	<code>WTBO</code> constructor
<code>wtreemgr</code>	<code>NTREE</code> manager

## General Utilities and Demos

wcodemat	Extended pseudocolor matrix scaling
localmax	Compute local maxima positions
wextend	Extend vector or matrix
wkeep	Keep part of vector or matrix
wrev	Flip vector
wtbxmng	Wavelet Toolbox manager
wavemenu	Wavelet GUI tools
wvarchg	Find variance change points
waveinfo	Wavelet information
waveletfamilies	Wavelet families and family members
wavedemo	Wavelet Toolbox demos

## **Functions — Alphabetical List**

This section contains function reference pages listed alphabetically.





**Purpose** Add lifting steps to lifting scheme

**Syntax**

```
LSN = addlift(LS,ELS)
LSN = addlift(LS,ELS,'begin')
LSN = addlift(LS,ELS,'end')
```

**Description**

LSN = addlift(LS,ELS) returns the new lifting scheme LSN obtained by appending the elementary lifting step ELS to the lifting scheme LS.

LSN = addlift(LS,ELS,'begin') prepends the specified elementary lifting step.

ELS is either a cell array (see lsinfo)

```
{TYPEVAL, COEFS, MAX_DEG}
```

or a structure (see liftfilt)

```
struct('type',TYPEVAL,'value',LPVAL)
```

with

```
LPVAL = laurpoly(COEFS, MAX_DEG)
```

LSN = addlift(LS,ELS,'end') is equivalent to addfilt(LS,ELS).

If ELS is a sequence of elementary lifting steps, stored in a cell array or an array of structures, then each of the elementary lifting steps is added to LS.

For more information about lifting schemes, see lsinfo.

**Examples**

```
% 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] []
};
```

# addlift

---

```
% 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]          []
};
```

## See Also

liftfilt

**Purpose**

Tree nodes

**Syntax**

```
N = allnodes(T)
N = allnodes(T, 'deppos')
```

**Description**

allnodes is a tree management utility that returns one of two node descriptions: either indices, or depths and positions.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

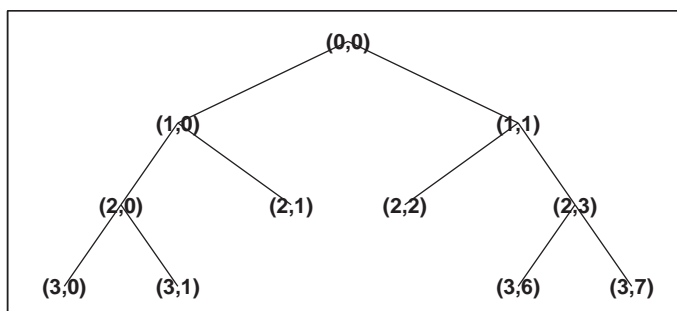
`N = allnodes(T)` returns the indices of all the nodes of the tree `T` in column vector `N`.

`N = allnodes(T, 'deppos')` returns the depths and positions of all the nodes in matrix `N`.

`N(i,1)` is the depth and `N(i,2)` the position of the node `i`.

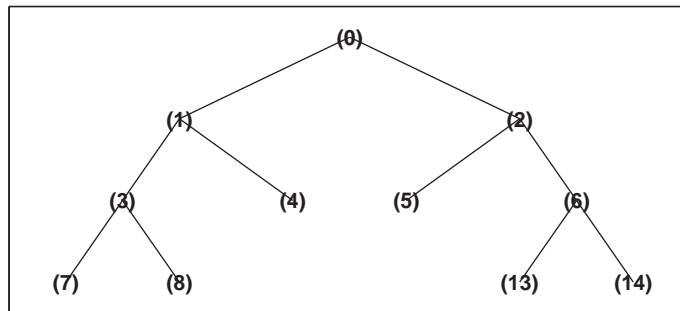
**Examples**

```
% Create initial tree.
ord = 2;
t = ntree(ord,3);    % Binary tree of depth 3.
t = nodejoin(t,5);
t = nodejoin(t,4);
plot(t)
```



```
% Change Node Label from Depth_Position to Index
% (see the plot function).
```

# allnodes



```
% List t nodes (index).
aln_ind = allnodes(t)
aln_ind =
    0
    1
    2
    3
    4
    5
    6
    7
    8
   13
   14

% List t nodes (Depth_Position).
aln_depo = allnodes(t,'depos')
aln_depo =
    0  0
    1  0
    1  1
    2  0
    2  1
    2  2
    2  3
    3  0
    3  1
    3  6
    3  7
```

**Purpose** 1-D approximation coefficients

**Syntax**

```
A = appcoef(C,L,'wname',N)
A = appcoef(C,L,'wname')
A = appcoef(C,L,Lo_R,Hi_R)
A = appcoef(C,L,Lo_R,Hi_R,N)
```

**Description** `appcoef` is a one-dimensional wavelet analysis function. `appcoef` computes the approximation coefficients of a one-dimensional signal. `A = appcoef(C,L,'wname',N)` computes the approximation coefficients at level `N` using the wavelet decomposition structure `[C,L]` (see `wavedec` for more information).

`'wname'` is a string containing the wavelet name. Level `N` must be an integer such that  $0 \leq N \leq \text{length}(L) - 2$ .

`A = appcoef(C,L,'wname')` extracts the approximation coefficients at the last level:  $\text{length}(L) - 2$ .

Instead of giving the wavelet name, you can give the filters.

For `A = appcoef(C,L,Lo_R,Hi_R)` or `A = appcoef(C,L,Lo_R,Hi_R,N)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter (see `wfilters` for more information).

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

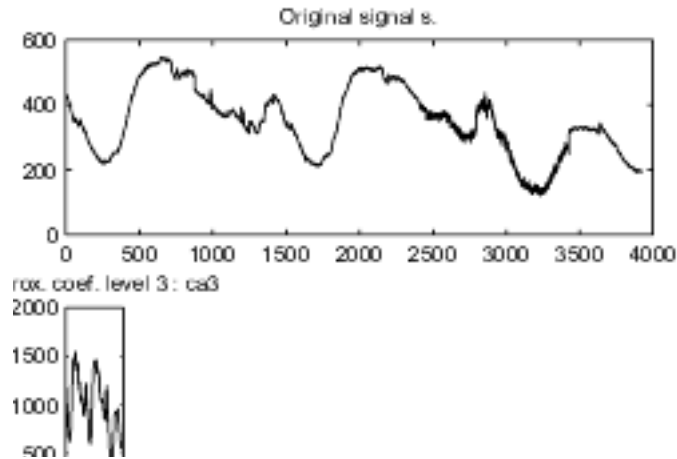
% Load a one-dimensional signal.
load leleccum; s = leleccum(1:3920);

% Perform decomposition at level 3 of s using db1.
[c,l] = wavedec(s,3,'db1');

% Extract approximation coefficients at level 3, from the
% wavelet decomposition structure [c,l].
ca3 = appcoef(c,l,'db1',3);
```

# appcoef

```
% Using some plotting commands,  
% the following figure is generated.
```



## Algorithm

The input vectors  $C$  and  $L$  contain all the information about the signal decomposition.

Let  $NMAX = \text{length}(L) - 2$ ; then  $C = [A(NMAX) \ D(NMAX) \ \dots \ D(1)]$  where  $A$  and the  $D$  are vectors.

If  $N = NMAX$ , then a simple extraction is done; otherwise, `appcoef` computes iteratively the approximation coefficients using the inverse wavelet transform.

## See Also

`detcoef`, `wavedec`

<b>Purpose</b>	2-D approximation coefficients
<b>Syntax</b>	<pre>A = appcoef2(C,S,'wname',N) A = appcoef2(C,S,'wname') A = appcoef2(C,S,Lo_R,Hi_R) A = appcoef2(C,S,Lo_R,Hi_R,N)</pre>
<b>Description</b>	<p>appcoef2 is a two-dimensional wavelet analysis function. It computes the approximation coefficients of a two-dimensional signal. The syntaxes allow you to give the wavelet name or the filters as inputs.</p> <p>A = appcoef2(C,S,'wname',N) computes the approximation coefficients at level N using the wavelet decomposition structure [C,S] (see wavedec2 for more information).</p> <p>'wname' is a string containing the wavelet name. Level N must be an integer such that <math>0 \leq N \leq \text{size}(S,1) - 2</math>.</p> <p>A = appcoef2(C,S,'wname') extracts the approximation coefficients at the last level: <math>\text{size}(S,1) - 2</math>.</p> <p>A = appcoef2(C,S,Lo_R,Hi_R) or A = appcoef2(C,S,Lo_R,Hi_R,N), Lo_R is the reconstruction low-pass filter and Hi_R is the reconstruction high-pass filter (see wfilters for more information).</p>
<b>Remarks</b>	<p>If C and S are obtained from an indexed image analysis or a truecolor image analysis, A is an m-by-n matrix or an m-by-n-by-3 array, respectively.</p> <p>For more information on image formats, see the image and imfinfo reference pages.</p>
<b>Examples</b>	<pre>% The current extension mode is zero-padding (see dwtmode).  % Load original image. load woman;  % X contains the loaded image.  % Perform decomposition at level 2 % of X using db1. [c,s] = wavedec2(X,2,'db1');</pre>

```
sizeX = size(X)
sizeX =
    256 256

sizec = size(c)

sizec =
     1 65536
val_s = s

val_s =
    64 64
    64 64
   128 128
   256 256

% Extract approximation coefficients
% at level 2.
ca2 = appcoef2(c,s,'db1',2);
sizeca2 = size(ca2)

sizeca2 =
    64 64

% Compute approximation coefficients
% at level 1.
ca1 = appcoef2(c,s,'db1',1);
sizeca1 = size(ca1)

sizeca1 =
   128 128
```

## Algorithm

The algorithm is built on the same principle as `appcoef`.

## See Also

`detcoef2`, `wavedec2`

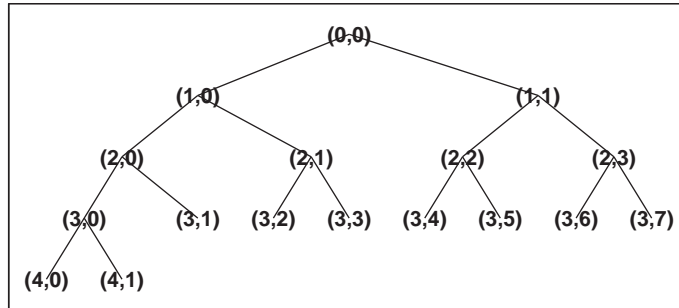


---

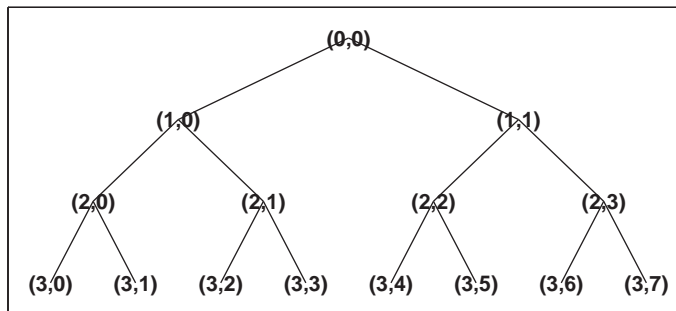
<b>Purpose</b>	Best level tree wavelet packet analysis
<b>Syntax</b>	<code>T = bestlevt(T)</code> <code>[T,E] = bestlevt(T)</code>
<b>Description</b>	<p><code>bestlevt</code> is a one- or two-dimensional wavelet packet analysis function.</p> <p><code>bestlevt</code> computes the optimal complete subtree of an initial tree with respect to an entropy type criterion. The resulting complete tree may be of smaller depth than the initial one.</p> <p><code>T = bestlevt(T)</code> computes the modified wavelet packet tree <code>T</code> corresponding to the best level tree decomposition.</p> <p><code>[T,E] = bestlevt(T)</code> computes the best level tree <code>T</code>, and in addition, the best entropy value <code>E</code>.</p> <p>The optimal entropy of the node, whose index is <code>j - 1</code>, is <code>E(j)</code>.</p>
<b>Examples</b>	<pre>% The current extension mode is zero-padding (see dwtmode).  % Load signal. load noisdopp; x = noisdopp;  % Decompose x at depth 3 with db1 wavelet, using default % entropy (shannon). wpt = wpdec(x,3,'db1');  % Decompose the packet [3 0]. wpt = wpsplt(wpt,[3 0]);</pre>

# bestlevt

```
% Plot wavelet packet tree wpt.  
plot(wpt)
```



```
% Compute best level tree.  
blt = bestlevt(wpt);  
  
% Plot best level tree blt.  
plot(blt)
```



## Algorithm

See `besttree` algorithm section. The only difference is that the optimal tree is searched among the complete subtrees of the initial tree, instead of among all the binary subtrees.

## See Also

`besttree`, `wenergy`, `wpdec`, `wpdec2`

**Purpose** Best tree wavelet packet analysis

**Syntax**  
`T = besttree(T)`  
`[T,E] = besttree(T)`  
`[T,E,N] = besttree(T)`

**Description** `besttree` is a one- or two-dimensional wavelet packet analysis function that computes the optimal subtree of an initial tree with respect to an entropy type criterion. The resulting tree may be much smaller than the initial one.

Following the organization of the wavelet packets library, it is natural to count the decompositions issued from a given orthogonal wavelet.

A signal of length  $N = 2^L$  can be expanded in  $\alpha$  different ways, where  $\alpha$  is the number of binary subtrees of a complete binary tree of depth  $L$ .

As a result, we can conclude that  $\alpha \geq 2^{N/2}$  (for more information, see the Mallat's book given in References at page 323).

This number may be very large, and since explicit enumeration is generally intractable, it is interesting to find an optimal decomposition with respect to a convenient criterion, computable by an efficient algorithm. We are looking for a minimum of the criterion.

`T = besttree(T)` computes the best tree  $T$  corresponding to the best entropy value.

`[T,E] = besttree(T)` computes the best tree  $T$  and, in addition, the best entropy value  $E$ .

The optimal entropy of the node, whose index is  $j - 1$ , is  $E(j)$ .

`[T,E,N] = besttree(T)` computes the best tree  $T$ , the best entropy value  $E$  and, in addition, the vector  $N$  containing the indices of the merged nodes.

## Examples

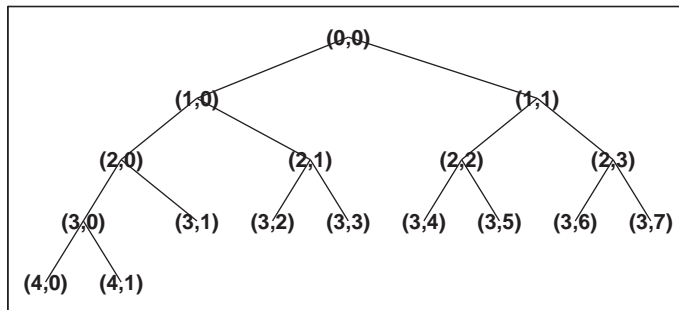
```
% The current extension mode is zero-padding (see dwtmode).

% Load signal.
load noisdopp; x = noisdopp;

% Decompose x at depth 3 with db1 wavelet, using default
% entropy (shannon).
wpt = wpdec(x,3,'db1');

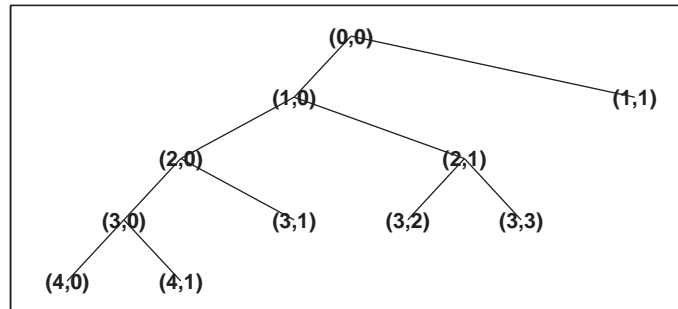
% Decompose the packet [3 0].
wpt = wpsplt(wpt,[3 0]);

% Plot wavelet packet tree wpt.
plot(wpt)
```



```
% Compute best tree.
bt = besttree(wpt);

% Plot best tree bt.
plot(bt)
```



## Algorithm

Consider the one-dimensional case. Starting with the root node, the best tree is calculated using the following scheme. A node  $N$  is split into two nodes  $N1$  and  $N2$  if and only if the sum of the entropy of  $N1$  and  $N2$  is lower than the entropy of  $N$ . This is a local criterion based only on the information available at the node  $N$ .

Several entropy type criteria can be used (see `wenergy` for more information). If the entropy function is an additive function along the wavelet packet coefficients, this algorithm leads to the best tree.

Starting from an initial tree  $T$  and using the merging side of this algorithm, we obtain the best tree among all the binary subtrees of  $T$ .

## See Also

`bestlevt`, `wenergy`, `wpdec`, `wpdec2`

## References

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.

Mallat, S. (1998), *A wavelet tour of signal processing*, Academic Press.

# biorfilt

---

**Purpose** Biorthogonal wavelet filter set

**Syntax**  
`[Lo_D,Hi_D,Lo_R,Hi_R] = biorfilt(DF,RF)`  
`[Lo_D1,Hi_D1,Lo_R1,Hi_R1,Lo_D2,Hi_D2,Lo_R2,Hi_R2] =`  
`biorfilt(DF,RF,'8')`

**Description** The `biorfilt` command returns either four or eight filters associated with biorthogonal wavelets.

`[Lo_D,Hi_D,Lo_R,Hi_R] = biorfilt(DF,RF)` computes four filters associated with the biorthogonal wavelet specified by decomposition filter `DF` and reconstruction filter `RF`. These filters are

`Lo_D`        Decomposition low-pass filter  
`Hi_D`        Decomposition high-pass filter  
`Lo_R`        Reconstruction low-pass filter  
`Hi_R`        Reconstruction high-pass filter

`[Lo_D1,Hi_D1,Lo_R1,Hi_R1,Lo_D2,Hi_D2,Lo_R2,Hi_R2] =`  
`biorfilt(DF,RF,'8')` returns eight filters, the first four associated with the decomposition wavelet, and the last four associated with the reconstruction wavelet.

It is well known in the subband filtering community that if the same FIR filters are used for reconstruction and decomposition, then symmetry and exact reconstruction are incompatible (except with the Haar wavelet). Therefore, with biorthogonal filters, two wavelets are introduced instead of just one:

One wavelet,  $\tilde{\psi}$ , is used in the analysis, and the coefficients of a signal  $s$  are

$$\tilde{c}_{j,k} = \int s(x) \tilde{\Psi}_{j,k}(x) dx$$

The other wavelet,  $\psi$ , is used in the synthesis:

$$s = \sum_{j,k} \tilde{c}_{j,k} \Psi_{j,k}$$

Furthermore, the two wavelets are related by duality in the following sense:

$$\int \tilde{\psi}_{j,k}(x) \psi_{j',k'}(x) dx = 0 \text{ as soon as } j \neq j' \text{ or } k \neq k' \text{ and}$$

$$\int \tilde{\phi}_{0,k}(x) \phi_{0,k'}(x) dx = 0 \text{ as soon as } k \neq k'.$$

It becomes apparent, as A. Cohen pointed out in his thesis (p. 110), that “the useful properties for analysis (e.g., oscillations, null moments) can be concentrated in the  $\tilde{\psi}$  function; whereas, the interesting properties for synthesis (regularity) are assigned to the  $\psi$  function. The separation of these two tasks proves very useful.”

$\tilde{\psi}$  and  $\psi$  can have very different regularity properties,  $\psi$  being more regular than  $\tilde{\psi}$ .

The  $\tilde{\psi}$ ,  $\psi$ ,  $\tilde{\phi}$  and  $\phi$  functions are zero outside a segment.

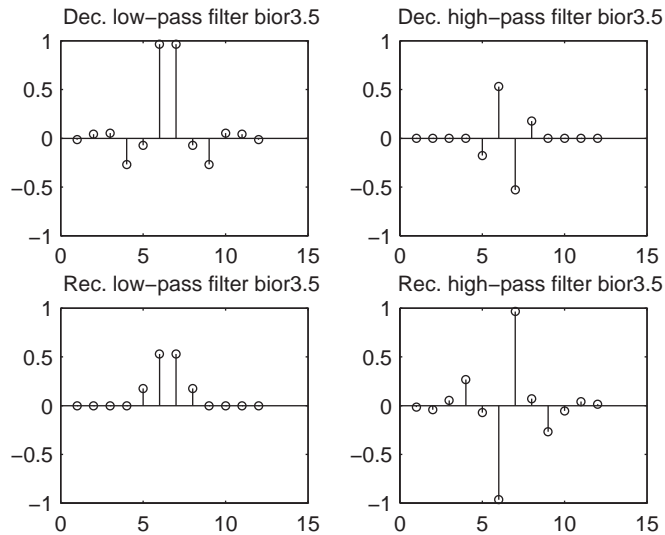
## Examples

```
% Compute the four filters associated with spline biorthogonal
% wavelet 3.5: bior3.5.
```

```
% Find the two scaling filters associated with bior3.5.
[Rf,Df] = biorwavf('bior3.5');
```

```
% Compute the four filters needed.
[Lo_D,Hi_D,Lo_R,Hi_R] = biorfilt(Df,Rf);
subplot(221); stem(Lo_D);
title('Dec. low-pass filter bior3.5');
subplot(222); stem(Hi_D);
title('Dec. high-pass filter bior3.5');
subplot(223); stem(Lo_R);
title('Rec. low-pass filter bior3.5');
subplot(224); stem(Hi_R);
title('Rec. high-pass filter bior3.5');
```

```
% Editing some graphical properties,
% the following figure is generated.
```



```
% Orthogonality by dyadic translation is lost.
```

```
nzer = [Lo_D 0 0]*[0 0 Lo_D]'
nzer =
    -0.6881
nzer = [Hi_D 0 0]*[0 0 Hi_D]'
nzer =
    0.1875
```

```
% But using duality we have:
```

```
zer = [Lo_D 0 0]*[0 0 Lo_R]'
zer =
   -2.7756e-17
zer = [Hi_D 0 0]*[0 0 Hi_R]'
zer =
    2.7756e-17
```

```
% But, perfect reconstruction via DWT is preserved.
```

```
x = randn(1,500);
[a,d] = dwt(x,Lo_D,Hi_D);
xrec = idwt(a,d,Lo_R,Hi_R);
err = norm(x-xrec)
```



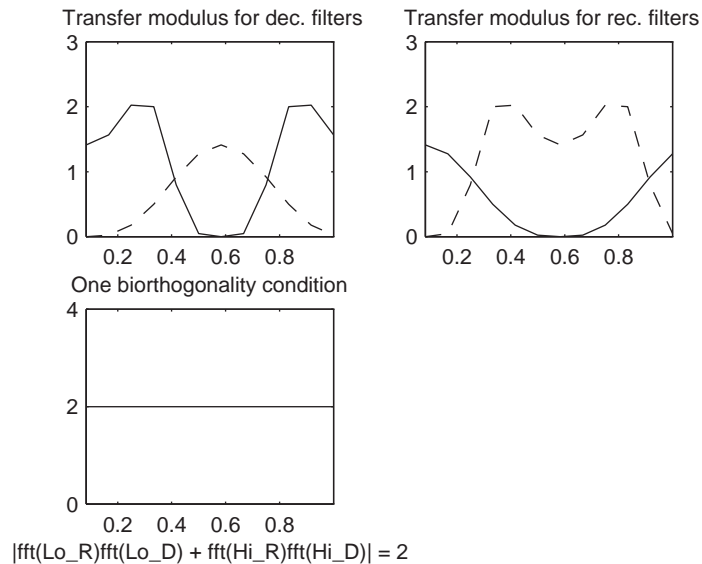
```

err =
    5.0218e-15

% High and low frequency illustration.
fftlD = fft(Lo_D); ffthD = fft(Hi_D);
freq = [1:length(Lo_D)]/length(Lo_D);
subplot(221); plot(freq,abs(fftlD),freq,abs(ffthD));
title('Transfer modulus for dec. filters')
fftlR = fft(Lo_R); ffthR = fft(Hi_R);
freq = [1:length(Lo_R)]/length(Lo_R);
subplot(222); plot(freq,abs(fftlR),freq,abs(ffthR));
title('Transfer modulus for rec. filters')
subplot(223); plot(freq, abs(fftlR.*fftlD + ffthD.*ffthR));
title('One biorthogonality condition')
xlabel('|fft(Lo_R)fft(Lo_D) + fft(Hi_R)fft(Hi_D)| = 2')

% Editing some graphical properties,
% the following figure is generated.

```



---

**Note** For biorthogonal wavelets, the filters for decomposition and reconstruction are generally of different odd lengths. This situation occurs, for example, for “splines” biorthogonal wavelets used in the toolbox where the four filters are zero-padded to have the same even length.

---

## See Also

`biorwavf`, `orthfilt`

## References

Cohen, A. (1992), “Ondelettes, analyses multirésolution et traitement numérique du signal,” *Ph. D. Thesis*, University of Paris IX, DAUPHINE.

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

**Purpose** Biorthogonal spline wavelet filters

**Syntax** [RF,DF] = biorwavf(W)

**Description** [RF,DF] = biorwavf(W) returns two scaling filters associated with the biorthogonal wavelet specified by the string W.

W = 'biorNr.Nd' where possible values for Nr and Nd are

Nr = 1	Nd = 1 , 3 or 5
Nr = 2	Nd = 2 , 4 , 6 or 8
Nr = 3	Nd = 1 , 3 , 5 , 7 or 9
Nr = 4	Nd = 4
Nr = 5	Nd = 5
Nr = 6	Nd = 8

The output arguments are filters.

- RF is the reconstruction filter.
- DF is the decomposition filter.

### Examples

```
% Set spline biorthogonal wavelet name.
wname = 'bior2.2';

% Compute the two corresponding scaling filters.
% rf is the reconstruction scaling filter.
% df is the decomposition scaling filter.
[rf,rd] = biorwavf(wname)

rf =
    0.2500    0.5000    0.2500

df =
   -0.1250    0.2500    0.7500    0.2500   -0.1250
```

**See Also** biorfilt, waveinfo

# bswfun

---

**Purpose** Biorthogonal scaling and wavelet functions

**Syntax**

```
[PHIS,PSIS,PHIA,PSIA,XVAL] = bswfun(LoD,HiD,LoR,HiR)
[PHIS,PSIS,PHIA,PSIA,XVAL] = bswfun(LoD,HiD,LoR,HiR,ITER)
[PHIS,PSIS,PHIA,PSIA,XVAL] = bswfun(LoD,HiD,LoR,HiR,'plot')
[PHIS,PSIS,PHIA,PSIA,XVAL] = bswfun(LoD,HiD,LoR,HiR,ITER,'plot')
```

**Description** [PHIS,PSIS,PHIA,PSIA,XVAL] = bswfun(LoD,HiD,LoR,HiR) returns approximations on the grid XVAL of the two pairs of scaling function and wavelet (PHIA,PSIA), (PHIS,PSIS) associated with the two pairs of filters (LoD,HiD), (LoR,HiR).

bswfun(LoD,HiD,LoR,HiR,ITER) computes the two pairs of scaling and wavelet functions using ITER iterations.

bswfun(LoD,HiD,LoR,HiR,'plot') or bswfun(LoD,HiD,LoR,HiR,ITER,'plot') or bswfun(LoD,HiD,LoR,HiR,'plot',ITER) computes and plots the functions.

**Examples**

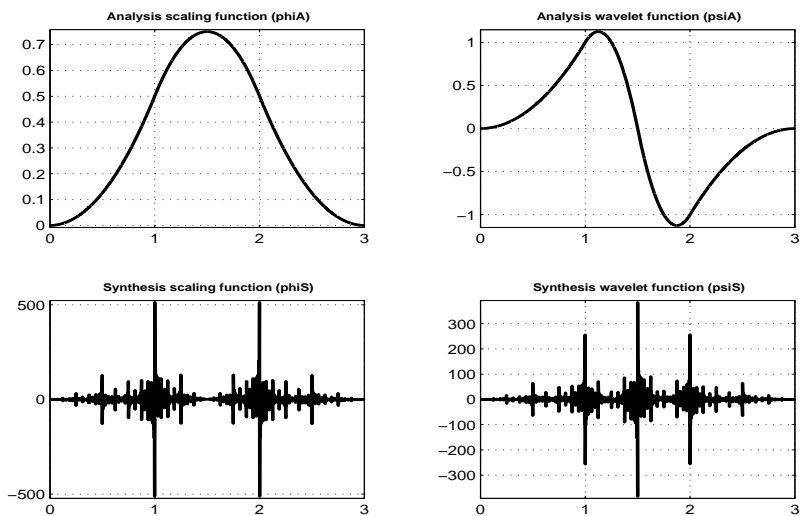
```
% Start from the Cohen-Daubechies-Feauveau wavelet
% and get the corresponding lifting scheme.
lscdf = liftwave('cdf3.1');
```

```
% Visualize the obtained lifting scheme.
displs(lscdf);
```

```
lscdf = {...
'p'          [ -0.33333333]          [-1]
'd'          [ -0.37500000 -1.12500000] [1]
'p'          [ 0.44444444]          [0]
[ 2.12132034] [ 0.47140452]          []
};
```

```
% Transform the lifting scheme to biorthogonal
% filters quadruplet.
[LoD,HiD,LoR,HiR] = ls2filt(lscdf);
```

```
% Visualize the two pairs of scaling and wavelet
% functions.
bswfun(LoD,HiD,LoR,HiR,'plot');
```

**Algorithm**

This function uses the cascade algorithm.

**See Also**

wavefun

# centfrq

---

**Purpose** Wavelet center frequency

**Syntax**

```
FREQ = centfrq('wname')  
FREQ = centfrq('wname',ITER)  
[FREQ,XVAL,RECFREQ] = centfrq('wname',ITER,'plot')
```

**Description** FREQ = centfrq('wname') returns the center frequency in herz of the wavelet function, 'wname' (see wavefun for more information).

For FREQ = centfrq('wname',ITER), ITER is the number of iterations performed by the function wavefun, which is used to compute the wavelet.

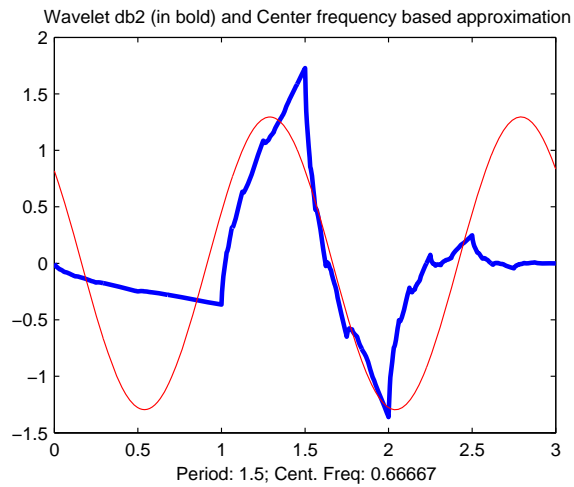
[FREQ,XVAL,RECFREQ] = centfrq('wname',ITER,'plot') returns, in addition, the associated center frequency based approximation RECFREQ on the  $2^{\text{ITER}}$  points grid XVAL and plots the wavelet function and RECFREQ.

**Examples**

```
% Example 1: a real wavelet  
wname = 'db2';
```

```
% Compute the center frequency and display  
% the wavelet function and the associated  
% center frequency based approximation.  
iter = 8;  
cfreq = centfrq(wname,8,'plot')
```

```
cfreq =  
    0.6667
```



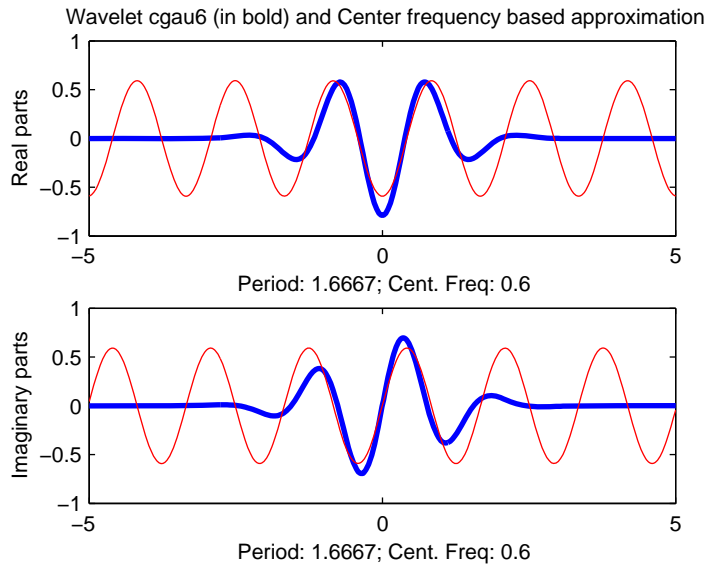
```
% Example 2: a complex wavelet  
wname = 'cgau6';
```

```
% Compute the center frequency and display  
% the wavelet function and the associated  
% center frequency based approximation.  
cfreq = centfrq(wname,8,'plot')
```

```
cfreq =
```

```
0.6000
```

# centfrq



## See Also

`sca12frq`, `wavefun`



<b>Purpose</b>	Wavelet packet tree construction from coefficients
<b>Syntax</b>	<code>T = cfs2wpt(WNAME,SIZE_OF_DATA,TN_OF_TREE,ORDER,CFS)</code>
<b>Description</b>	<p>CFS2WPT builds a wavelet packet tree (T) and the related analyzed signal or image (X) using the following input information:</p> <p>WNAME: name of the wavelet used for the analysis</p> <p>SIZE_OF_DATA: size of the analyzed signal or image</p> <p>TN_OF_TREE: vector containing the terminal node indices of the tree</p> <p>ORDER: 2 for a signal or 4 for an image</p> <p>CFS: coefficients used to reconstruct the original signal or image. CFS is optional. When CFS2WPT is used without the CFS input parameter, the wavelet packet tree structure (T) is generated, but all the tree coefficients are null (including X).</p>

**Examples**

```

% Example 1: Using cfs2wpt with the CFS argument

% Loading an image
load detail

% Building the wavelet packet tree decomposition
t = wpdec2(X,2,'sym4');

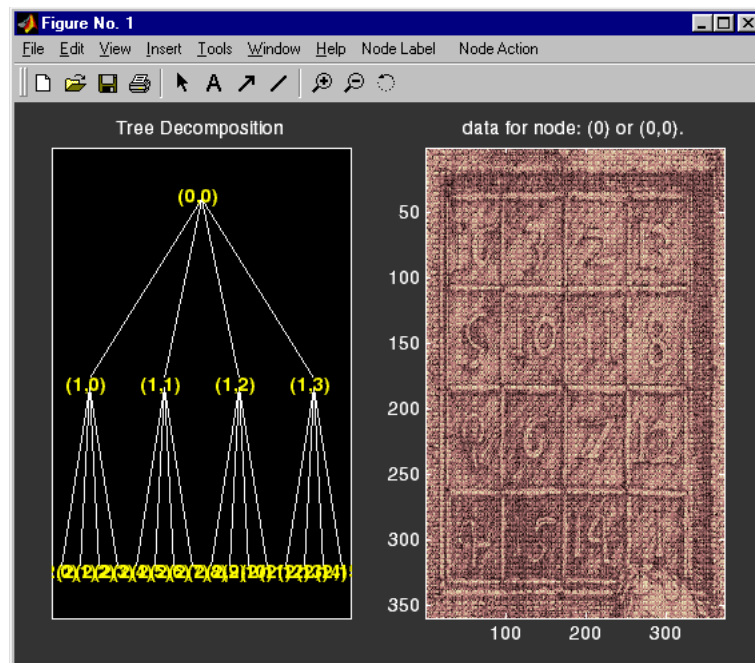
% Reading the coefficient values from the tree
cfs = read(t,'allcfs');

% Adding noise to the coefficients
noisyCfs = cfs + 40*rand(size(cfs));

% Building the wavelet packet tree object and the reconstructed
% noisy image from the noisyCfs using cfs2wpt
noisyT = cfs2wpt('sym4',size(X),tnodes(t),4,noisyCfs);

% Plotting the new tree and clicking the node (0) or (0,0)
plot(noisyT)

```



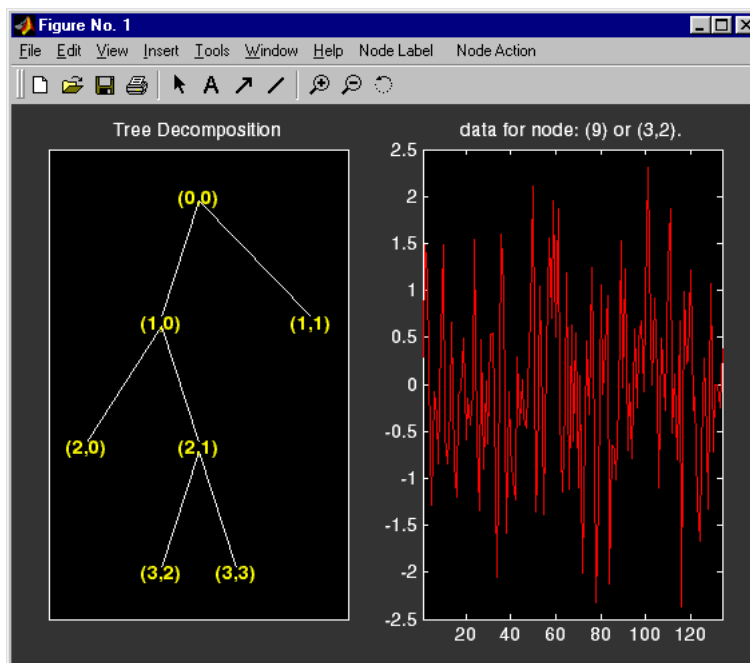
% Example 2: Using cfs2wpt without the CFS argument

```
% Building an empty wavelet packet tree object
t = cfs2wpt('sym4',[1 1024],[3 9 10 2]',2);
```

```
% Getting the terminal node sizes
sN = read(t,'sizes',[3,9]);
sN3 = sN(1,:); sN9 = sN(2,:);
```

```
% Building coefficient values vectors and writing them in the tree
cfsN3 = ones(sN3);
cfsN9 = randn(sN9);
t = write(t,'cfs',3,cfsN3,'cfs',9,cfsN9);
```

```
% Plotting the updated tree and clicking the node (9) or (3,2)
plot(t)
```



# cgauwavf

---

**Purpose** Complex Gaussian wavelet

**Syntax** [PSI,X] = cgauwavf(LB,UB,N,P)

**Description** [PSI,X] = cgauwavf(LB,UB,N,P) returns values of the P-th derivative of the complex Gaussian function on an N point regular grid for the interval [LB,UB]. Cp is such that the 2-norm of the P-th derivative of  $F$  is equal to 1.

For  $P > 8$ , Symbolic Math Toolbox™ software is required.

Output arguments are the wavelet function PSI computed on the grid X.

[PSI,X] = cgauwavf(LB,UB,N) is equivalent to

[PSI,X] = cgauwavf(LB,UB,N,1)

These wavelets have an effective support of [-5 5].

## Examples

```
% Set effective support and grid parameters.
```

```
lb = -5; ub = 5; n = 1000;
```

```
% Compute complex Gaussian wavelet of order 4.
```

```
[psi,x] = cgauwavf(lb,ub,n,4);
```

```
% Plot complex Gaussian wavelet of order 4.
```

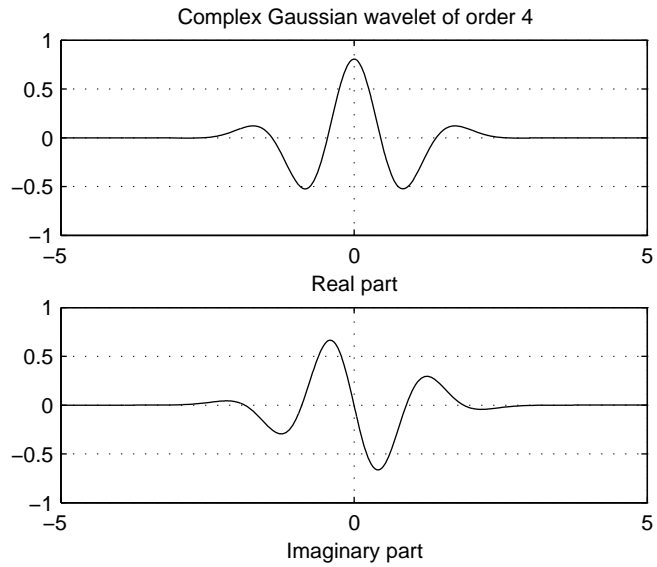
```
subplot(211)
```

```
plot(x,real(psi)),
```

```
title('Complex Gaussian wavelet of order 4')
```

```
xlabel('Real part'), grid
```

```
subplot(212)  
plot(x,imag(psi))  
xlabel('Imaginary part'), grid
```

**See Also**

waveinfo

# chgwdeccfs

---

**Purpose** Change multisignal 1-D decomposition coefficients

**Syntax**

```
DEC = chgwdeccfs(DEC, 'ca', COEFS)
DEC = chgwdeccfs(DEC, 'cd', COEFS, LEV)
DEC = chgwdeccfs(DEC, 'all', CA, CD)
DEC = chgwdeccfs(DEC, 'all', V)
DEC = chgwdeccfs(..., IDXSIG)
```

**Description** `DEC = chgwdeccfs(DEC, 'ca', COEFS)` replaces the approximation coefficients at level `DEC.level1` with those contained in the matrix `COEFS`. If `COEFS` is a single value `V`, all coefficients are replaced by `V`.

`DEC = chgwdeccfs(DEC, 'cd', COEFS, LEV)` replaces the detail coefficients at level `LEV` with those contained in the matrix `COEFS`. If `COEFS` is a single value `V`, then `LEV` can be a vector of levels and all the coefficients that belong to these levels are replaced by `V`. `LEV` must be such that  $1 \leq LEV \leq DEC.level1$

`DEC = chgwdeccfs(DEC, 'all', CA, CD)` replaces all the approximation and detail coefficients. `CA` must be a matrix and `CD` must be a cell array of length `DEC.level1`.

If `COEFS` (or `CA` or `CD`) is a single number, then it replaces all the related coefficients. Otherwise, `COEFS` (or `CA`, or `CD`) must be a matrix of appropriate size.

For a real value `V`, `DEC = chgwdeccfs(DEC, 'all', V)` replaces all the coefficients by `V`.

`DEC = chgwdeccfs(..., IDXSIG)` replaces the coefficients for the signals whose indices are given by the vector `IDXSIG`. If the initial data are stored row-wise or column-wise in a matrix `X`, then `IDXSIG` contains the row or column indices, respectively, of the data.

**Examples**

```
% Load original 1D-multisignal
load thinker
```

```
% Perform a decomposition at level 2 using wavelet db2
dec = mdwtdec('r', X, 2, 'db2');
```

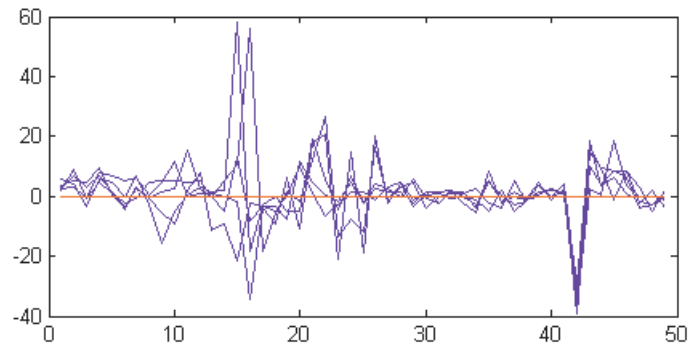
```

% Change the coefficients of details at level 1.
% Replace all values by 0.
decBIS = chgwdeccfs(dec, 'cd', 0, 1);

% Change the coefficients of details at level 1 and
% level 2 for signals 31 to 35. Replace all values by 0.
decTER = chgwdeccfs(dec, 'cd', 0, 1:2, 31:35);

% Compare original and new coefficients for details
% at level 1 for signals 31 to 35.
plot(dec.cd{1}(31:35, :)', 'b'); hold on;
plot(decTER.cd{1}(31:35, :)', 'r')

```



## See Also

mdwtdec, mdwtrec

# cmorwavf

---

**Purpose** Complex Morlet wavelet

**Syntax** [PSI,X] = cmorwavf(LB,UB,N,FB,FC)

**Description** [PSI,X] = cmorwavf(LB,UB,N,FB,FC) returns values of the complex Morlet wavelet defined by a positive bandwidth parameter FB, a wavelet center frequency FC, and the expression

$$\text{PSI}(X) = ((\pi \cdot \text{FB})^{-0.5}) \cdot \exp(2 \cdot i \cdot \pi \cdot \text{FC} \cdot X) \cdot \exp(-X^2 / \text{FB})$$

on an N point regular grid for the interval [LB,UB].

Output arguments are the wavelet function PSI computed on the grid X.

**Examples**

```
% Set bandwidth and center frequency parameters.
fb = 1.5; fc = 1;

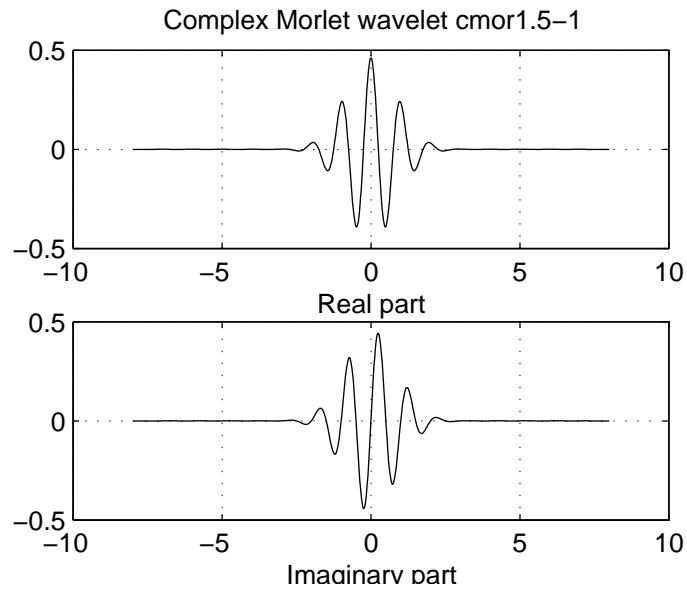
% Set effective support and grid parameters.
lb = -8; ub = 8; n = 1000;

% Compute complex Morlet wavelet cmor1.5-1.
[psi,x] = cmorwavf(lb,ub,n,fb,fc);

% Plot complex Morlet wavelet.
subplot(211)
plot(x,real(psi)),
title('Complex Morlet wavelet cmor1.5-1')
xlabel('Real part'), grid
```



```
subplot(212)
plot(x,imag(psi))
xlabel('Imaginary part'), grid
```

**See Also**

waveinfo

**References**

Teolis, A. (1998), *Computational signal processing with wavelets*, Birkhauser, p. 65.

# coifwavf

---

**Purpose** Coiflet wavelet filter

**Syntax** `F = coifwavf(W)`

**Description** `F = coifwavf(W)` returns the scaling filter associated with the Coiflet wavelet specified by the string `W` where `W = 'coifN'`. Possible values for `N` are 1, 2, 3, 4, or 5.

**Examples**

```
% Set coiflet wavelet name.
wname = 'coif2';

% Compute the corresponding scaling filter.
f = coifwavf(wname)

f =
Columns 1 through 7
0.0116 -0.0293 -0.0476 0.2730 0.5747 0.2949 -0.0541

Columns 8 through 12
-0.0420 0.0167 0.0040 -0.0013 -0.0005
```

**See Also** `waveinfo`

**Purpose** Real or complex continuous 1-D wavelet coefficient

**Syntax**

```
COEFS = cwt(S, SCALES, 'wname')
COEFS = cwt(S, SCALES, 'wname', 'plot')
COEFS = cwt(S, SCALES, 'wname', PLOTMODE)
COEFS = cwt(S, SCALES, 'wname', PLOTMODE, XLIM)
```

**Description** `cwt` is a one-dimensional wavelet analysis function.

`COEFS = cwt(S, SCALES, 'wname')` computes the continuous wavelet coefficients of the vector `S` at real, positive `SCALES`, using the wavelet whose name is `'wname'` (see `waveinfo` for more information).

The signal `S` is real, the wavelet can be real or complex.

`COEFS = cwt(S, SCALES, 'wname', 'plot')` computes and, in addition, plots the continuous wavelet transform coefficients.

`COEFS = cwt(S, SCALES, 'wname', PLOTMODE)` computes and plots the continuous wavelet transform coefficients.

Coefficients are colored using `PLOTMODE`. Valid values for the string `PLOTMODE` are listed in the table below.

PLOTMODE	Description
'lv1'	Coloration made scale-by-scale
'glb'	Coloration made considering all scales
'abslv1' or 'lvlabs'	Coloration made scale-by-scale using the absolute values of the coefficients
'absglb' or 'glbabs'	Coloration made considering all scales using the absolute values of the coefficients

`COEFS = cwt(..., 'plot')` is equivalent to `COEFS = cwt(..., 'absglb')`.

---

**Note** You can get 3-D plots (surfaces) using the same keywords listed above for the PLOTMODE parameter, preceded by '3D'. For example:  
 COEFS = cwt(...,'3Dplot') or COEFS = cwt(...,'3D1v1') ...

---

COEFS = cwt(S, SCALES, 'wname', PLOTMODE, XLIM) computes and plots the continuous wavelet transform coefficients. Coefficients are colored using PLOTMODE and XLIM, where XLIM is a vector, [x1 x2], with  $1 \leq x1 < x2 \leq \text{length}(S)$

Let  $s$  be the signal and  $\psi$  the wavelet. The wavelet coefficient of  $s$  at scale  $a$  and position  $b$  is defined by

$$C_{a,b} = \int_R s(t) \frac{1}{\sqrt{a}} \overline{\psi\left(\frac{t-b}{a}\right)} dt$$

Since  $s(t)$  is a discrete signal, we use a piecewise constant interpolation of the  $s(k)$  values,  $k = 1$  to  $\text{length}(s)$ .

COEFS = cwt(...,'scal')  
 [COEFS,SC] = cwt(...,'scal')  
 COEFS = cwt(...,'scalCNT')  
 [COEFS,SC] = cwt(...,'scalCNT') computes the continuous wavelet transform coefficients and the corresponding scalogram which represents the percentage of energy for each coefficient. When PLOTMODE is equal to 'scal', a scaled image of the scalogram is displayed, or when PLOTMODE is equal to 'scalCNT', a contour representation of the scalogram is displayed.

For each given scale  $a$  within the vector SCALES, the wavelet coefficients  $C_{a,b}$  are computed for  $b = 1$  to  $ls = \text{length}(s)$ , and are stored in COEFS(i,:) if  $a = \text{SCALES}(i)$ .

Output argument COEFS is an  $la$ -by- $ls$  matrix, where  $la$  is the length of SCALES. COEFS is a real or complex matrix, depending on the wavelet type.

Examples of valid uses are as follows:

```
t = linspace(-1,1,512);
s = 1-abs(t);
c = cwt(s,1:32,'cgau4');
```

```

c = cwt(s,[64 32 16:-2:2],'morl');
c = cwt(s,[3 18 12.9 7 1.5],'db2');
c = cwt(s,1:64,'sym4','abslvl',[100 400]);
[c,sc] = cwt(s,1:64,'sym4','scal');
[c,sc] = cwt(s,1:64,'sym4','scalCNT');

```

## Examples

This example demonstrates the difference between discrete and continuous wavelet transforms.

```

% Load a fractal signal.
load vonkoch
vonkoch=vonkoch(1:510);
lv = length(vonkoch);

subplot(311), plot(vonkoch);title('Analyzed signal. ');
set(gca,'Xlim',[0 510])
% Perform discrete wavelet transform at level 5 by sym2.
% Levels 1 to 5 correspond to scales 2, 4, 8, 16 and 32.
[c,l] = wavedec(vonkoch,5,'sym2');

% Expand discrete wavelet coefficients for plot.
% Levels 1 to 5 correspond to scales 2, 4, 8, 16 and 32.
cfd = zeros(5,lv);
for k = 1:5
    d = detcoef(c,l,k);
    d = d(ones(1,2^k),:);
    cfd(k,:) = wkeep(d(:)',lv);
end

cfd = cfd(:);
I = find(abs(cfd)<sqrt(eps));
cfd(I)=zeros(size(I));
cfd = reshape(cfd,5,lv);

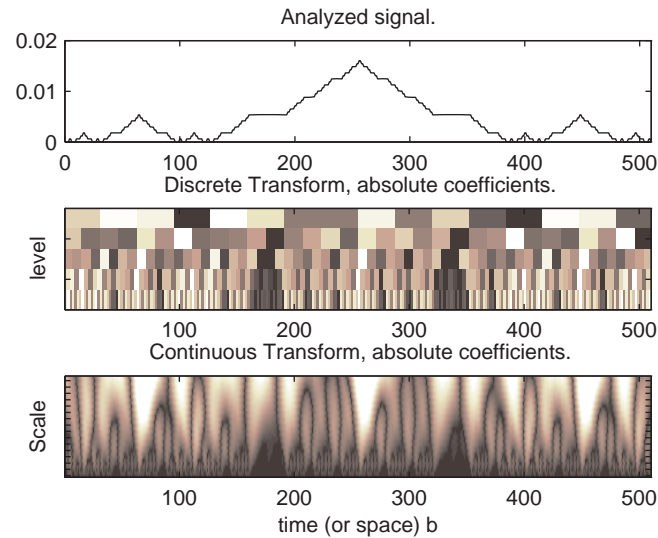
% Plot discrete coefficients.
subplot(312), colormap(pink(64));
img = image(flipud(wcodemat(cfd,64,'row')));
set(get(img,'parent'),'YtickLabel',[]);
title('Discrete Transform, absolute coefficients.')
ylabel('level')

% Perform continuous wavelet transform by sym2 at all integer

```

```
% scales from 1 to 32.  
subplot(313)  
ccfs = cwt(vonkoch,1:32,'sym2','plot');  
title('Continuous Transform, absolute coefficients.')  
colormap(pink(64));  
ylabel('Scale')
```

% Editing some graphical properties,  
% the following figure is generated.



## Algorithm

$$C_{a,b} = \int_R s(t) \frac{1}{\sqrt{a}} \overline{\psi\left(\frac{t-b}{a}\right)} dt$$

$$C_{a,b} = \sum_k \int_k^{k+1} s(t) \frac{1}{\sqrt{a}} \overline{\psi\left(\frac{t-b}{a}\right)} dt$$

if  $s(t) = s(k)$ , for  $t \in [k, k+1]$  then

$$C_{a,b} = \frac{1}{\sqrt{a}} \sum_k s(k) \int_k^{k+1} \overline{\psi\left(\frac{t-b}{a}\right)} dt$$

$$C_{a,b} = \frac{1}{\sqrt{a}} \sum_k s(k) \left( \int_{-\infty}^{k+1} \overline{\psi\left(\frac{t-b}{a}\right)} dt - \int_{-\infty}^k \overline{\psi\left(\frac{t-b}{a}\right)} dt \right)$$

So at any scale  $a$ , the wavelet coefficients  $C_{a,b}$  for  $b = 1$  to `length(s)` can be obtained by convolving the signal  $s$  and a dilated and translated version of the integrals of the form  $\int_{-\infty}^k \psi(t)dt$  (given by `intwave`), and taking the finite difference using `diff`.

### See Also

`cwtext`, `wavedec`, `wavefun`, `waveinfo`, `wcodemat`



**Purpose** Real or complex continuous 1-D wavelet coefficients using extension parameters

**Syntax**  
 COEFS = cwttext(S, SCALES, 'wname')  
 COEFS = cwttext(S, SCALES, 'wname', PropName1, ProVal1, ...)

**Description**  
 COEFS = cwttext(S, SCALES, 'wname') computes the continuous wavelet coefficients of the vector S at real, positive SCALES, using a wavelet named 'wname'. The signal S is real; the wavelet can be real or complex.  
 COEFS = cwttext(S, SCALES, 'wname', PropName1, ProVal1, ...) computes and plots the continuous wavelet transform coefficients using extra parameters. Valid values for PropName are:

- 'ExtMode'
- 'ExtSide'
- 'ExtLen'
- 'PlotMode'
- 'xlim'

The continuous wavelet transform coefficients are computed using the extension parameters:

- 'ExtMode'
- 'ExtSide'
- 'ExtLen'

Valid values for *ExtMode* are:

- 'zpd' (zero padding)
- 'sp0' (smooth extension of order 0)
- 'sp1' (smooth extension of order 1)

etc.

Valid values for *ExtSide* are:

- ExtSide = 'l' (or 'u') for left (or up) extension
- ExtSide = 'r' (or 'd') for right (or down) extension

- ExtSide = 'b' for extension on both sides
- ExtSide = 'n' null extension

For the complete list of valid values for ExtMode and ExtSide, see wextend.

ExtLen is the length of extension.

Default values for extension parameters are 'zpd' and 'b'. ExtLen is computed using the maximum of SCALES.

Instead of three parameters, use the following syntaxes:

```
EXTMODE = struct('Mode',ModeVAL,'Side',SideVAL,'Len',LenVAL);  
EXTMODE = {ModeVAL,SideVAL,LenVAL};
```

COEFS = cwtext(...,'PlotMode',PLOTMODE) computes and plots the continuous wavelet transform coefficients. Coefficients are colored using PLOTMODE:

- PLOTMODE = 'lv1' (By scale)
- PLOTMODE = 'glb' (All scales)
- PLOTMODE = 'abslv1' or 'lv1abs' (Absolute value and By scale)
- PLOTMODE = 'absglb' or 'glbabs' (Absolute value and All scales)

You get 3-D plots (surfaces) using the same keywords listed above for the PLOTMODE parameter, preceded by '3D', for example, PLOTMODE = '3Dlv1'.

When PLOTMODE = 'scal' or 'scalCNT' the continuous wavelet transform coefficients and the corresponding scalogram (percentage of energy for each coefficient) are computed.

When PLOTMODE is 'scal', a scaled image of scalogram is displayed. When PLOTMODE is 'scalCNT', a contour representation of scalogram is displayed.

If the XLIM parameter is given, the continuous wavelet transform coefficients are colored using PLOTMODE and XLIM.

XLIM = [x1 x2] with  $1 \leq x1 < x2 \leq \text{length}(S)$ .

For each given scale  $a$  within the vector `SCALES`, the wavelet coefficients  $C(a, b)$  are computed for  $b = 1$  to  $1s = \text{length}(S)$ , and are stored in `COEFS(i, :)` if  $a = \text{SCALES}(i)$ .

Output argument `COEFS` is a  $1a$ -by- $1s$  matrix where  $1a$  is the length of `SCALES`. `COEFS` is a real or complex matrix depending on the wavelet type.

Examples of valid use are as follows:

```
t = linspace(-1,1,512);
s = 1-abs(t);
c = cwtext(s,1:32,'cgau4');
c = cwtext(s,[64 32 16:-2:2],'morl');
c = cwtext(s,[3 18 12.9 7 1.5],'db2');
c = cwtext(s,1:32,'sym2','plotMode','lvl');
c = cwtext(s,1:64,'sym4','plotMode','abslvl','xlim',[100 400]);

[c,Sc] = cwtext(s,1:64,'sym4','plotMode','scal');
[c,Sc] = cwtext(s,1:64,'sym4','plotMode','scalCNT');
[c,Sc] = cwtext(s,1:64,'sym4','plotMode','scalCNT', ...
               'extMode','sp1');

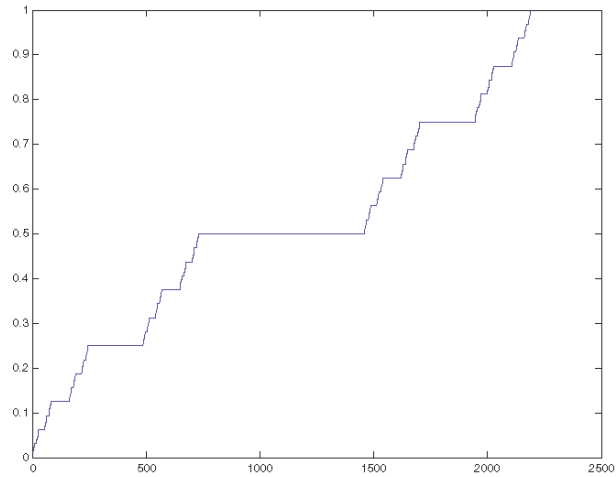
c = cwtext(s,1:64,'sym4','plotMode','lvl','extMode','sp0');
c = cwtext(s,1:64,'sym4','plotMode','lvl','extMode','sp1');
c = cwtext(s,1:64,'sym4','plotMode','lvl', ...
           'extMode',{'sp1','b',300});

ext = struct('Mode','sp1','Side','b','Len',300);
c = cwtext(s,1:64,'sym4','plotMode','lvl','extMode',ext);
```

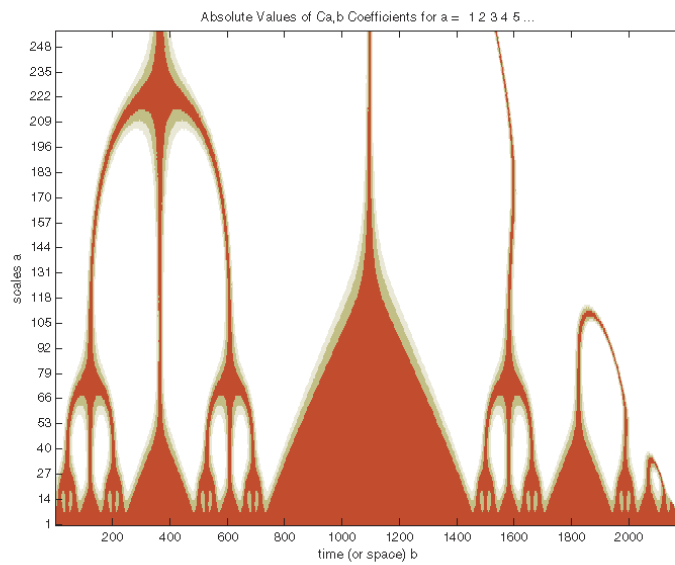
## Examples

This example demonstrates the difference between a continuous wavelet transform which deals with signal extension and one which does not.

```
% Load and plot the signal
load wcantor
plot(wcantor)
```

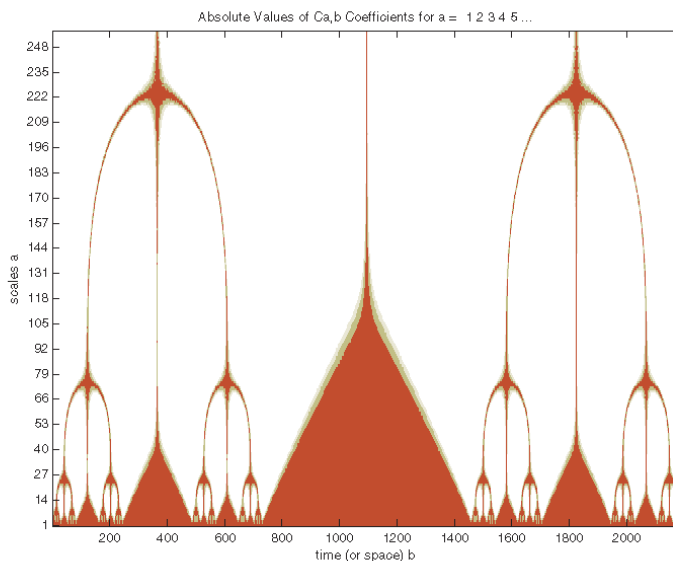


```
% Compute and plot the coefficients  
cwt(wcantor,(1:256),'mexh','absglb');  
colormap(pink(4))
```



In this figure above, which is produced by the `cwt` function, the values of coefficients are tremendously affected by the boundary effect due to the discontinuity of the signal on the right. The default (zero-padding) extension mode on the right explains this important discontinuity because the last value is 1. On the left there is no effect because the first value is 0.

```
% Compute and plot the coefficients with adapted extension mode
figure;
cwttext(wcantor,(1:256),'mexh','extmode','sp0','extLen',2000, ...
        'plotMode','absglb');
colormap(pink(4))
```



In this figure, produced by the `cwtext` function, the suitable extension mode of the signal is very efficient, giving as it can be seen, a good result.

**See Also**

`cwt`, `wavedec`, `wavefun`, `waveinfo`, `wcodemat`

# dbaux

---

**Purpose** Daubechies wavelet filter computation

**Syntax**  $W = \text{dbaux}(N, \text{SUMW})$   
 $W = \text{dbaux}(N)$

**Description**  $W = \text{dbaux}(N, \text{SUMW})$  is the order  $N$  Daubechies scaling filter such that  $\text{sum}(W) = \text{SUMW}$ . Possible values for  $N$  are 1, 2, 3, ...

---

**Note** Instability may occur when  $N$  is too large.

---

$W = \text{dbaux}(N)$  is equivalent to  $W = \text{dbaux}(N, 1)$

$W = \text{dbaux}(N, 0)$  is equivalent to  $W = \text{dbaux}(N, 1)$

## Examples

```
% P the Lagrange trous filter for N=2 is explicit
% and given by:
P = [ -1/16 0 9/16 1 9/16 0 -1/16]

P =
   -0.0625         0  0.5625  1.0000  0.5625         0 -0.0625

% The db2 Daubechies scaling filter w, is a
% solution of the equation: P = conv(wrev(w),w) * 2.
%
% This filter P is symmetric, easy to generate, and w is
% a minimum phase solution of the previous equation,
% based on the roots of P.
rP = roots(P);

% Retaining only the root inside the unit circle (here it
% is the sixth value of rP), and two roots located at -1,
% we obtain the Daubechies wavelet of order 2:
ww = poly([rP(6) -1 -1]); % filter construction
ww = ww / sum(ww) % normalize sum

ww =
    0.3415  0.5915  0.1585 -0.0915
```

```

% Check that ww is correct and equal to
% the db2 Daubechies scaling filter w.
w = dbaux(2)

w =
    0.3415    0.5915    0.1585   -0.0915

```

## Algorithm

The algorithm used is based on a result obtained by Shensa (see “References”), showing a correspondence between the “Lagrange à trous” filters and the convolutional squares of the Daubechies wavelet filters.

The computation of the order  $N$  Daubechies scaling filter  $w$  proceeds in two steps: compute a “Lagrange à trous” filter  $P$ , and extract a square root. More precisely:

- $P$  the associated “Lagrange à trous” filter is a symmetric filter of length  $4N-1$ .  $P$  is defined by

$$P = [\alpha(N) \ 0 \ \alpha(N-1) \ 0 \dots 0 \ \alpha(1) \ 1 \ \alpha(1) \ 0 \ \alpha(2) \ 0 \dots 0 \ \alpha(N)]$$

where

$$\alpha(k) = \frac{\prod_{\substack{i = -N+1 \\ i \neq k}}^N \left(\frac{1}{2} - i\right)}{N \prod_{\substack{i = -N+1 \\ i \neq k}} (k - i)} \quad \text{for } k = 1, \dots, N$$

- Then, if  $w$  denotes db $N$  Daubechies scaling filter of sum  $\sqrt{2}$ ,  $w$  is a square root of  $P$ :

$$P = \text{conv}(\text{wrev}(w), w) \text{ where } w \text{ is a filter of length } 2N.$$

The corresponding polynomial has  $N$  zeros located at  $-1$  and  $N-1$  zeros less than  $1$  in modulus.

Note that other methods can be used; see various solutions of the spectral factorization problem in Strang-Nguyen (p. 157).

**Limitations**

The computation of the dbN Daubechies scaling filter requires the extraction of the roots of a polynomial of order  $4N$ . Instability may occur when  $N$  is too large.

**See Also**

dbwavf, wfilters

**References**

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics, SIAM Ed.

Shensa, M.J. (1992), "The discrete wavelet transform: wedding the a trous and Mallat Algorithms," *IEEE Trans. on Signal Processing*, vol. 40, 10, pp. 2464-2482.

Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press.



---

<b>Purpose</b>	Daubechies wavelet filter
<b>Syntax</b>	<code>F = dbwavf(W)</code>
<b>Description</b>	<code>F = dbwavf(W)</code> returns the scaling filter associated with Daubechies wavelet specified by the string <code>W</code> where <code>W = 'dbN'</code> . Possible values for <code>N</code> are 1, 2, 3, ..., 45.
<b>Examples</b>	<pre>% Set Daubechies wavelet name. wname = 'db4';  % Compute the corresponding scaling filter. f = dbwavf(wname)  f = Columns 1 through 7 0.1629  0.5055  0.4461 -0.0198 -0.1323  0.0218  0.0233 Column 8 -0.0075</pre>
<b>See Also</b>	<code>dbaux</code> , <code>waveinfo</code> , <code>wfilters</code>

# ddencmp

---

**Purpose** Default values for de-noising or compression

**Syntax**

```
[THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,IN2,X)
[THR,SORH,KEEPAPP] = ddencmp(IN1,'wv',X)
[THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,'wp',X)
```

**Description** ddencmp is a de-noising and compression-oriented function.

ddencmp gives default values for all the general procedures related to de-noising and compression of one- or two-dimensional signals, using wavelets or wavelet packets.

[THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,IN2,X) returns default values for de-noising or compression, using wavelets or wavelet packets, of an input vector or matrix X, which can be a one- or two-dimensional signal. THR is the threshold, SORH is for soft or hard thresholding, KEEPAPP allows you to keep approximation coefficients, and CRIT (used only for wavelet packets) is the entropy name (see wentropy for more information).

IN1 is 'den' for de-noising or 'cmp' for compression.

IN2 is 'wv' for wavelet or 'wp' for wavelet packet.

*For wavelets (three output arguments):*

[THR,SORH,KEEPAPP] = ddencmp(IN1,'wv',X) returns default values for de-noising (if IN1 = 'den') or compression (if IN1 = 'cmp') of X. These values can be used for wdencomp.

*For wavelet packets (four output arguments):*

[THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,'wp',X) returns default values for de-noising (if IN1 = 'den') or compression (if IN1 = 'cmp') of X. These values can be used for wpdencomp.

**Examples** % The current extension mode is zero-padding (see dwtmode).

```
% Generate Gaussian white noise.
init = 2055415866; randn('seed',init);
x = randn(1,1000);
```

```

% Find default values for wavelets (3 output arguments).
% These values can be used for wdencmp with option 'gbl'.
% default for de-noising:
% soft thresholding and approximation coefficients kept
% thr = sqrt(2*log(n)) * s
% where s is an estimate of level noise
% and n is equal to prod(size(x)).
[thr,sorh,keepapp] = ddencmp('den','wv',x)

thr =
    3.8593
sorh =
    s
keepapp =
    1

% default for compression:
% hard thresholding and approximation coefficients kept
% thr = median(abs(detail at level 1)) if nonzero
% else thr = 0.05 * max(abs(detail at level 1)).
[thr,sorh,keepapp] = ddencmp('cmp','wv',x)

thr =
    0.7003
sorh =
    h
keepapp =
    1

% Find default values for wavelet packets (4 output arguments).
% These values can be used for wpdencmp.

% default for de-noising:
% soft thresholding and appr. cfs. kept
% thr = sqrt(2*log(n*log(n)/log(2)))
% the noise level is supposed to be equal to 1;
% default entropy is 'sure' criterion.
[thr,sorh,keepapp,crit] = ddencmp('den','wp',x)

```

# ddencmp

---

```
thr =
    4.2911
sorh =
    h
keepapp =
    1
crit =
    sure

% default for compression.
% hard thresholding and approximation coefficients kept
% thr = median(abs(detail at level 1))
% default entropy is 'threshold' criterion.
[thr,sorh,keepapp,crit] = ddencmp('cmp','wp',x)

thr =
    0.7003
sorh =
    h
keepapp =
    1
crit =
    threshold
```

## See Also

wdencmp, wenergy, wpdencmp

## References

Donoho, D.L. (1995), “De-noising by soft-thresholding,” *IEEE, Trans. on Inf. Theory*, 41, 3, pp. 613–627.

Donoho, D.L.; I.M. Johnstone (1994), “Ideal spatial adaptation by wavelet shrinkage,” *Biometrika*, vol 81, pp. 425–455.

Donoho, D.L.; I.M. Johnstone (1994), “Ideal de-noising in an orthonormal basis chosen from a library of bases,” *C.R.A.S. Paris, Ser. I*, t. 319, pp. 1317–1322.

**Purpose** Node depth-position to node index

**Syntax**  $N = \text{depo2ind}(\text{ORD}, [D \ P])$

**Description** `depo2ind` is a tree-management utility.

For a tree of order  $\text{ORD}$ ,  $N = \text{depo2ind}(\text{ORD}, [D \ P])$  computes the indices  $N$  of the nodes whose depths and positions are encoded within  $[D, P]$ .

The nodes are numbered from left to right and from top to bottom. The root index is 0.

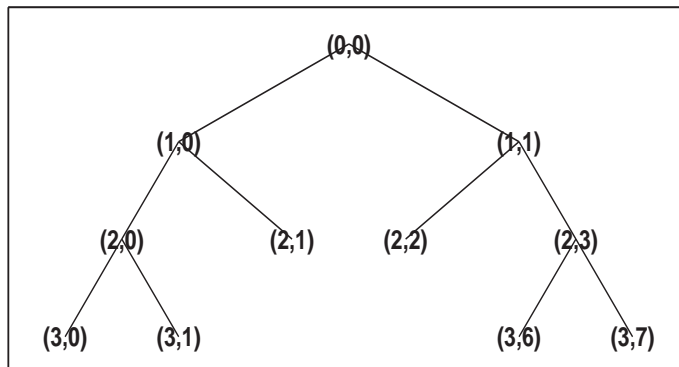
$D$  and  $P$  are column vectors. The values of depths  $D$  and positions  $P$  must be such that  $D \geq 0$  and  $0 \leq P \leq \text{ORD}^{D-1}$ .

Output indices  $N$  are such that  $0 \leq N < (\text{ORD}^{\max(D)} - 1) / (\text{ORD} - 1)$ .

Note that for a column vector  $X$ , we have  $\text{depo2ind}(0, X) = X$ .

### Examples

```
% Create initial tree.
ord = 2;
t = ntree(ord,3);      % binary tree of depth 3.
t = nodejoin(t,5);
t = nodejoin(t,4);
plot(t)
```



# depo2ind

---

```
% List t nodes (Depth_Position).
aln_depo = allnodes(t,'depos')
aln_depo =
    0     0
    1     0
    1     1
    2     0
    2     1
    2     2
    2     3
    3     0
    3     1
    3     6
    3     7

% Switch from Depth_Position to index.
aln_ind = depo2ind(ord,aln_depo)
aln_ind =
    0
    1
    2
    3
    4
    5
    6
    7
    8
   13
   14
```

**See Also**      ind2depo

**Purpose** 1-D detail coefficients

**Syntax** `D = detcoef(C,L,N)`  
`D = detcoef(C,L)`

**Description** `detcoef` is a one-dimensional wavelet analysis function.

`D = detcoef(C,L,N)` extracts the detail coefficients at level `N` from the wavelet decomposition structure `[C,L]`. See `wavedec` for more information on `C` and `L`.

Level `N` must be an integer such that  $1 \leq N \leq NMAX$   
 where  $NMAX = \text{length}(L) - 2$ .

`D = detcoef(C,L)` extracts the detail coefficients at last level `NMAX`.

If `N` is a vector of integers such that  $1 \leq N(j) \leq NMAX$ :

- `DCELL = detcoef(C,L,N,'cells')` returns a cell array where `DCELL{j}` contains the coefficients of detail `N(j)`.
- If  $\text{length}(N) > 1$ , `DCELL = detcoef(C,L,N)` is equivalent to `DCELL = detcoef(C,L,N,'cells')`.
- `DCELL = detcoef(C,L,'cells')` is equivalent to `DCELL = detcoef(C,L,[1:NMAX])`.
- `[D1, ..., Dp] = detcoef(C,L,[N(1), ..., N(p)])` extracts the details coefficients at levels `[N(1), ..., N(p)]`.

**Examples** `% The current extension mode is zero-padding (see dwtmode).`

```
% Load original one-dimensional signal.
```

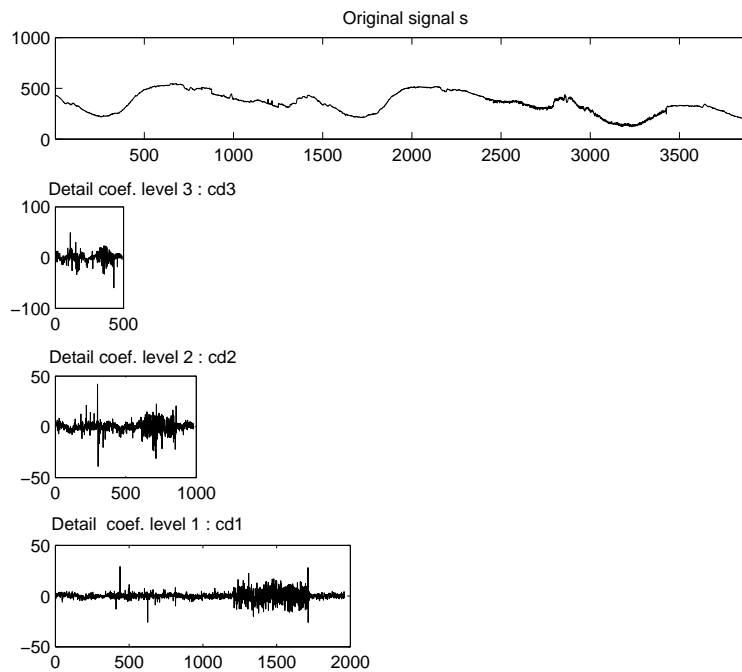
```
load leleccum;
s = leleccum(1:3920);
```

```
% Perform decomposition at level 3 of s using db1.
```

```
[c,l] = wavedec(s,3,'db1');
```

# detcoef

```
% Extract detail coefficients at levels  
% 1, 2 and 3, from wavelet decomposition  
% structure [c,l].  
[cd1,cd2,cd3] = detcoef(c,l,[1 2 3]);  
  
% Using some plotting commands,  
% the following figure is generated.
```



## See Also

appcoef, wavedec



**Purpose** 2-D detail coefficients

**Syntax** `D = detcoef2(0,C,S,N)`

**Description** `detcoef2` is a two-dimensional wavelet analysis function.

`D = detcoef2(0,C,S,N)` extracts from the wavelet decomposition structure `[C,S]` the horizontal, vertical, or diagonal detail coefficients for `0 = 'h'` (or `'v'` or `'d'`, respectively), at level `N`, where `N` must be an integer such that  $1 \leq N \leq \text{size}(S,1) - 2$ . See `wavedec2` for more information on `C` and `S`.

`[H,V,D] = detcoef2('all',C,S,N)` returns the horizontal `H`, vertical `V`, and diagonal `D` detail coefficients at level `N`.

`D = detcoef2('compact',C,S,N)` returns the detail coefficients at level `N`, stored row-wise.

`detcoef2('a',C,S,N)` is equivalent to `detcoef2('all',C,S,N)`.

`detcoef2('c',C,S,N)` is equivalent to `detcoef2('compact',C,S,N)`.

**Remarks** If `C` and `S` are obtained from an indexed image analysis or a truecolor image analysis, `D` is an `m`-by-`n` matrix or an `m`-by-`n`-by-3 array, respectively.

For more information on image formats, see the `image` and `imfinfo` reference pages.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

% Load original image.
load woman;

% X contains the loaded image.

% Perform decomposition at level 2
% of X using db1.
[c,s] = wavedec2(X,2,'db1');
sizeX = size(X)
sizeX =
    256    256
```

## detcoef2

---

```
sizec = size(c)
sizec =
     1    65536

val_s = s
val_s =
     64     64
     64     64
    128    128
    256    256

% Extract details coefficients at level 2
% in each orientation, from wavelet decomposition
% structure [c,s].
[chd2,cvd2,cdd2] = detcoef2('all',c,s,2);
sizecd2 = size(chd2)
sizecd2 =
     64     64

% Extract details coefficients at level 1
% in each orientation, from wavelet decomposition
% structure [c,s].
[chd1,cvd1,cdd1] = detcoef2('all',c,s,1);
sizecd1 = size(chd1)
sizecd1 =
    128    128
```

### See Also

appcoef2, wavedec2

**Purpose** WPTREE information

**Syntax** disp(T)

**Description** disp(T) displays the content of the WPTREE object T.

**Examples**

```
% Compute a wavelet packets tree
x = rand(1,1000);
t = wpdec(x,2,'db2');
disp(t)
```

```
Wavelet Packet Object Structure
=====
Size of initial data      : [1 1000]
Order                    : 2
Depth                    : 2
Terminal nodes           : [3 4 5 6]
-----
Wavelet Name              : db2
Low Decomposition filter  : [-0.1294  0.2241  0.8365  0.483]
High Decomposition filter : [ -0.483  0.8365 -0.2241 -0.1294]
Low Reconstruction filter : [ 0.483  0.8365  0.2241 -0.1294]
High Reconstruction filter: [-0.1294 -0.2241  0.8365 -0.483]
-----
Entropy Name              : shannon
Entropy Parameter        : 0
-----
```

**See Also** get, read, set, write

# displs

---

**Purpose** Display lifting scheme

**Syntax** `S = displs(LS,FRM)`

**Description** `S = displs(LS,FRM)` returns a string describing the lifting scheme `LS`. The format string `FRM` (see `sprintf`) builds `S`.

`displs(LS)` is equivalent to `DISPLS(LS, '%12.8f')`

For more information about lifting schemes, see `lsinfo`.

## Examples

```
% 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] []
};
```

**See Also** `lsinfo`

**Purpose** Draw wavelet packet decomposition tree (GUI)

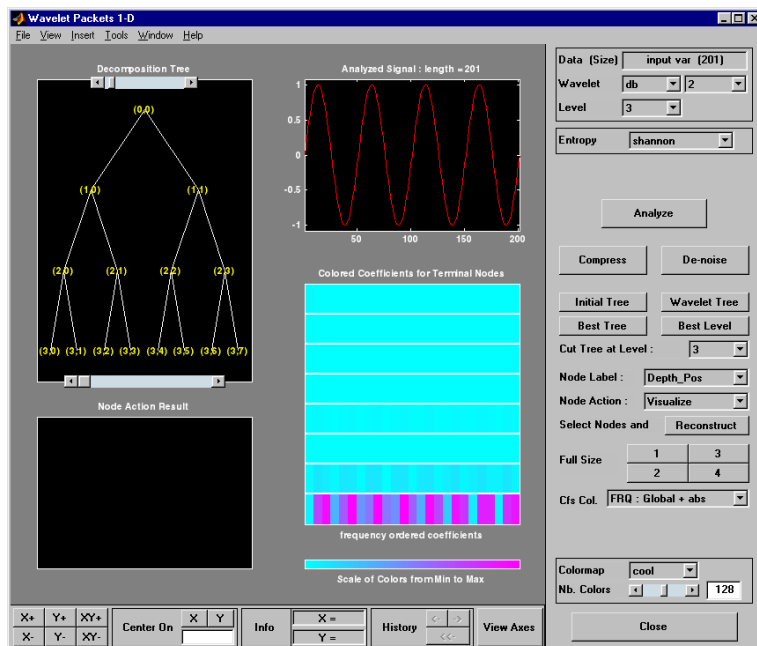
**Syntax**  
`drawtree(T)`  
`drawtree(T,F)`  
`F = drawtree(T)`

**Description** `drawtree(T)` draws the wavelet packet tree `T`, and `F = drawtree(T)` also returns the figure's handle.

For an existing figure `F` produced by a previous call to the `drawtree` function, `drawtree(T,F)` draws the wavelet packet tree `T` in the figure whose handle is `F`. For more information see Chapter 5 of the User's Guide, "Using Wavelet Packets" and Appendix B, "Wavelet Toolbox™ Software and Object Programming".

**Examples**

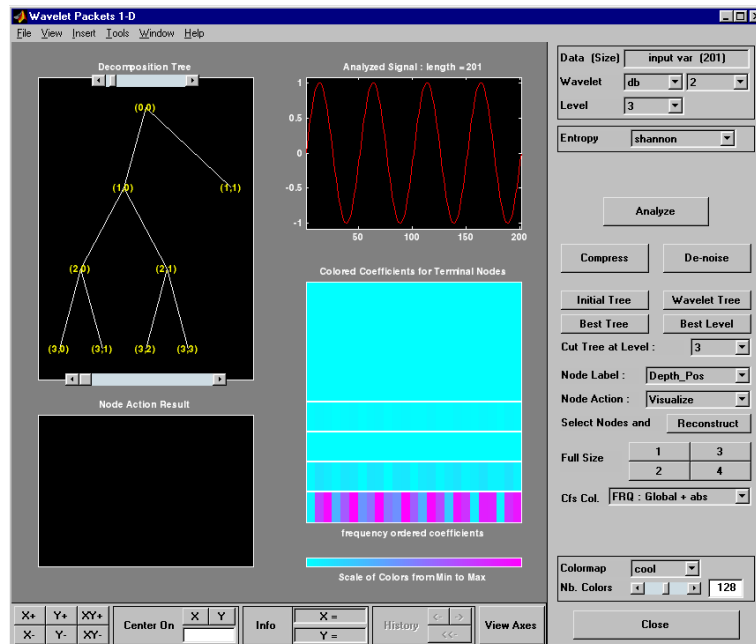
```
x = sin(8*pi*[0:0.005:1]);  
t = wpdec(x,3,'db2');  
fig = drawtree(t);
```



```

%-----
% Use command line function to modify t.
%-----
t = wpjoin(t,2);
drawtree(t,fig);

```



**See Also** readtree

# dtree

---

## Purpose

DTREE constructor

## Syntax

```
T = dtree(ORD,D,X)
T = dtree(ORD,D,X,U)
[T,NB] = dtree(...)
T = dtree('PropName1',PropValue1,'PropName2',PropValue2, ...)
```

## Description

T = dtree(ORD,D,X) returns a complete data tree (DTREE) object of order ORD and depth D. The data associated with the tree T is X.

With T = dtree(ORD,D,X,U) you can set a user data field.

[T,NB] = dtree(...) returns also the number of terminal nodes (leaves) of T.

[T,NB] = dtree('PropName1',PropValue1,'PropName2',PropValue2,...) is the most general syntax to construct a DTREE object.

The valid choices for 'PropName' are

- 'order' Order of the tree
- 'depth' Depth of the tree
- 'data' Data associated to the tree
- 'spsch' Split scheme for nodes
- 'ud' User data field

The split scheme field is an order ORD by 1 logical array. The root of the tree can be split and it has ORD children. If spsch(j) = 1, you can split the j-th child. Each node that you can split has the same property as the root node.

For more information on object fields, type help dtree/get.

Class DTREE (Parent class: NTREE)

## Fields

- dtree Parent object
- allNI All nodes information
- terNI Terminal nodes information



## Examples

```
% Create a data tree.  
x = [1:10];  
t = dtree(3,2,x);  
t = nodejoin(t,2);
```

## See Also

ntree, wtbo

# dwt

---

**Purpose** Single-level discrete 1-D wavelet transform

**Syntax**

```
[cA,cD] = dwt(X,'wname')  
[cA,cD] = dwt(X,'wname','mode',MODE)  
[cA,cD] = dwt(X,Lo_D,Hi_D)  
[cA,cD] = dwt(X,Lo_D,Hi_D,'mode',MODE)
```

**Description** The `dwt` command performs a single-level one-dimensional wavelet decomposition with respect to either a particular wavelet (`'wname'`, see `wfilters` for more information) or particular wavelet decomposition filters (`Lo_D` and `Hi_D`) that you specify.

`[cA,cD] = dwt(X,'wname')` computes the approximation coefficients vector `cA` and detail coefficients vector `cD`, obtained by a wavelet decomposition of the vector `X`. The string `'wname'` contains the wavelet name.

`[cA,cD] = dwt(X,Lo_D,Hi_D)` computes the wavelet decomposition as above, given these filters as input:

- `Lo_D` is the decomposition low-pass filter.
- `Hi_D` is the decomposition high-pass filter.

`Lo_D` and `Hi_D` must be the same length.

Let  $l_x$  = the length of `X` and  $l_f$  = the length of the filters `Lo_D` and `Hi_D`; then  $\text{length}(cA) = \text{length}(cD) = l_a$  where  $l_a = \text{ceil}(l_x/2)$ , if the DWT extension mode is set to periodization. For the other extension modes,  $l_a = \text{floor}(l_x+l_f-1)/2$ .

For more information about the different Discrete Wavelet Transform extension modes, see `dwtmode`.

`[cA,cD] = dwt(...,'mode',MODE)` computes the wavelet decomposition with the extension mode `MODE` that you specify. `MODE` is a string containing the desired extension mode.

Example:

```
[cA,cD] = dwt(x,'db1','mode','sym');
```

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

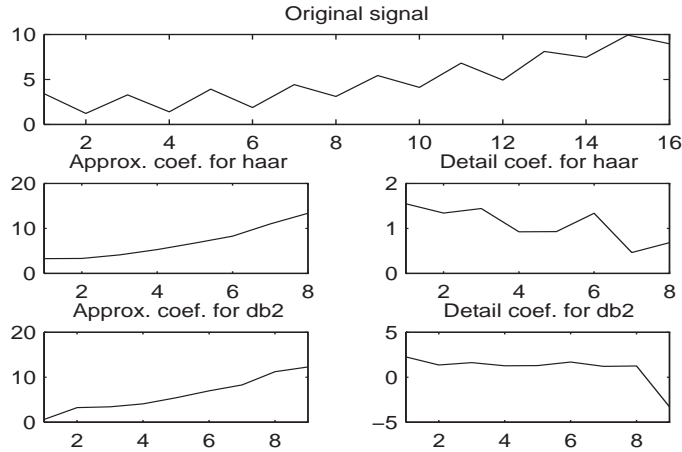
% Construct elementary original one-dimensional signal.
randn('seed',531316785)
s = 2 + kron(ones(1,8),[1 -1]) + ...
    ((1:16).^2)/32 + 0.2*randn(1,16);

% Perform single-level discrete wavelet transform of s by haar.
[ca1,cd1] = dwt(s,'haar');
subplot(311); plot(s); title('Original signal');
subplot(323); plot(ca1); title('Approx. coef. for haar');
subplot(324); plot(cd1); title('Detail coef. for haar');

% For a given wavelet, compute the two associated decomposition
% filters and compute approximation and detail coefficients
% using directly the filters.
[Lo_D,Hi_D] = wfilters('haar','d');
[ca1,cd1] = dwt(s,Lo_D,Hi_D);

% Perform single-level discrete wavelet transform of s by db2
% and observe edge effects for last coefficients.
% These extra coefficients are only used to ensure exact
% global reconstruction.
[ca2,cd2] = dwt(s,'db2');
subplot(325); plot(ca2); title('Approx. coef. for db2');
subplot(326); plot(cd2); title('Detail coef. for db2');
```

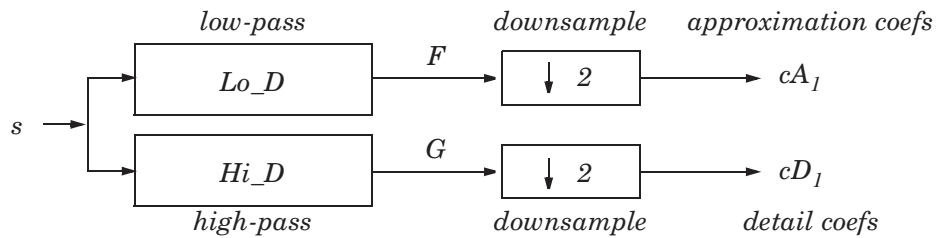
```
% Editing some graphical properties,
% the following figure is generated.
```



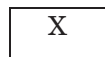
## Algorithm

Starting from a signal  $s$ , two sets of coefficients are computed: approximation coefficients  $CA_1$ , and detail coefficients  $CD_1$ . These vectors are obtained by convolving  $s$  with the low-pass filter  $Lo\_D$  for approximation and with the high-pass filter  $Hi\_D$  for detail, followed by dyadic decimation.

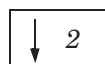
More precisely, the first step is



Where



Convolve with filter  $X$



Keep the even indexed elements  
(We call this operation *downsampling*)

The length of each filter is equal to  $2N$ . If  $n = \text{length}(s)$ , the signals  $F$  and  $G$  are of length  $n + 2N - 1$ , and then the coefficients  $CA_1$  and  $CD_1$  are of length

$$\text{floor}\left(\frac{n-1}{2}\right) + N$$

To deal with signal-end effects involved by a convolution-based algorithm, a global variable managed by `dwtmode` is used. This variable defines the kind of signal extension mode used. The possible options include zero-padding (used in the previous example) and symmetric extension, which is the default mode.

---

**Note** For the same input, this `dwt` function and the DWT block in the Signal Processing Blockset do not produce the same results. The blockset is designed for real-time implementation while Wavelet Toolbox™ software is designed for analysis, so they produce handle boundary conditions and filter states differently.

To make the `dwt` function output match the DWT block output, set the function boundary condition to zero-padding by typing `dwtmode('zpd')` at the MATLAB® command prompt. To match the latency of the DWT block, which is implemented using FIR filters, add zeros to the input of the `dwt` function. The number of zeros you add must be equal to half the filter length.

---

## See Also

`dwtmode`, `idwt`, `wavedec`, `waveinfo`

## References

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

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.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

# dwt2

---

**Purpose** Single-level discrete 2-D wavelet transform

**Syntax** `[cA,cH,cV,cD] = dwt2(X,'wname')`  
`[cA,cH,cV,cD] = dwt2(X,Lo_D,Hi_D)`

**Description** The `dwt2` command performs a single-level two-dimensional wavelet decomposition with respect to either a particular wavelet (`'wname'`, see `wfilters` for more information) or particular wavelet decomposition filters (`Lo_D` and `Hi_D`) you specify.

`[cA,cH,cV,cD] = dwt2(X,'wname')` computes the approximation coefficients matrix `cA` and details coefficients matrices `cH`, `cV`, and `cD` (horizontal, vertical, and diagonal, respectively), obtained by wavelet decomposition of the input matrix `X`. The `'wname'` string contains the wavelet name.

`[cA,cH,cV,cD] = dwt2(X,Lo_D,Hi_D)` computes the two-dimensional wavelet decomposition as above, based on wavelet decomposition filters that you specify.

- `Lo_D` is the decomposition low-pass filter.
- `Hi_D` is the decomposition high-pass filter.

`Lo_D` and `Hi_D` must be the same length.

Let `sx = size(X)` and `lf = the length of filters`; then `size(cA) = size(cH) = size(cV) = size(cD) = sa` where `sa = ceil(sx/2)`, if the DWT extension mode is set to symmetric padding. For the other extension modes, `sa = floor((sx+lf-1)/2)`.

For information about the different Discrete Wavelet Transform extension modes, see `dwtmode`.

`[cA,cH,cV,cD] = dwt2(...,'mode',MODE)` computes the wavelet decomposition with the extension mode `MODE` that you specify.

`MODE` is a string containing the desired extension mode.

An example of valid use is

```
[cA,cH,cV,cD] = dwt2(x,'db1','mode','sym');
```

**Remarks**

When  $X$  represents an indexed image, then  $X$ , as well as the output arrays  $cA$ ,  $cH$ ,  $cV$ ,  $cD$  are  $m$ -by- $n$  matrices. When  $X$  represents a truecolor image, it is an  $m$ -by- $n$ -by-3 array, where each  $m$ -by- $n$  matrix represents a red, green, or blue color plane concatenated along the third dimension.

For more information on image formats, see the `image` and `imfinfo` reference pages.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

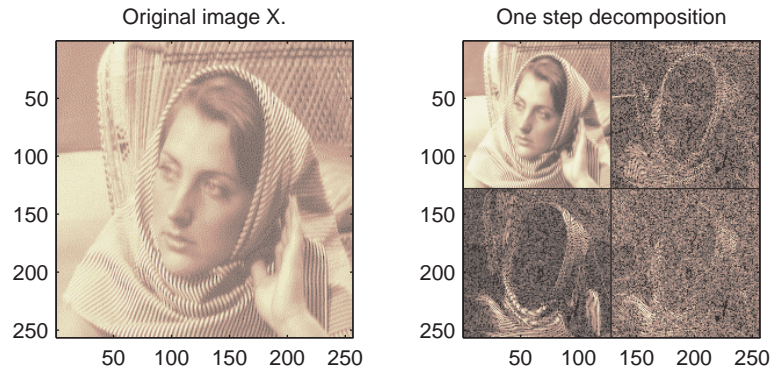
% Load original image.
load woman;

% X contains the loaded image.
% map contains the loaded colormap.
nbcoll = size(map,1);

% Perform single-level decomposition
% of X using db1.
[cA1,cH1,cV1,cD1] = dwt2(X,'db1');

% Images coding.
cod_X = wcodemat(X,nbcoll);
cod_cA1 = wcodemat(cA1,nbcoll);
cod_cH1 = wcodemat(cH1,nbcoll);
cod_cV1 = wcodemat(cV1,nbcoll);
cod_cD1 = wcodemat(cD1,nbcoll);
dec2d = [...
          cod_cA1,    cod_cH1;    ...
          cod_cV1,    cod_cD1    ...
        ];

% Using some plotting commands,
% the following figure is generated.
```



## Algorithm

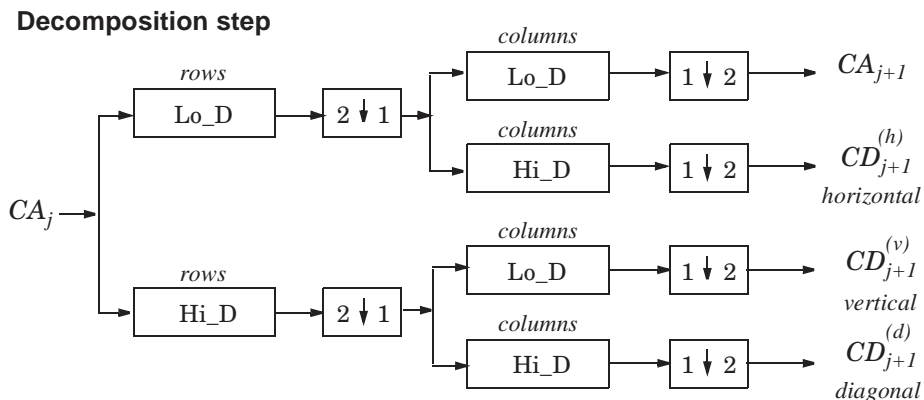
For images, there exist an algorithm similar to the one-dimensional case for two-dimensional wavelets and scaling functions obtained from one-dimensional ones by tensorial product.

This kind of two-dimensional DWT leads to a decomposition of approximation coefficients at level  $j$  in four components: the approximation at level  $j + 1$ , and the details in three orientations (horizontal, vertical, and diagonal).

The following chart describes the basic decomposition steps for images:



## Two-Dimensional DWT



Where

$2 \downarrow 1$	Downsample columns: keep the even indexed columns
$1 \downarrow 2$	Downsample rows: keep the even indexed rows
$\begin{matrix} \text{rows} \\ \text{X} \end{matrix}$	Convolve with filter X the rows of the entry
$\begin{matrix} \text{columns} \\ \text{X} \end{matrix}$	Convolve with filter X the columns of the entry

**Initialization**  $CA_0 = s$  for the decomposition initialization

---

**Note** To deal with signal-end effects involved by a convolution-based algorithm, a global variable managed by `dwtmode` is used. This variable defines the kind of signal extension mode used. The possible options include zero-padding (used in the previous example) and symmetric extension, which is the default mode.

---

**See Also**

`dwtmode`, `idwt2`, `wavedec2`, `waveinfo`

## References

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

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.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

**Purpose** Discrete wavelet transform extension mode

**Syntax** ST = dwtmode  
dwtmode('mode')

**Description** The dwtmode command sets the signal or image extension mode for discrete wavelet and wavelet packet transforms. The extension modes represent different ways of handling the problem of border distortion in signal and image analysis. For more information, see “Dealing with Border Distortion” in Chapter 6, “Advanced Concepts”, of the User’s Guide.

dwtmode or dwtmode('status') display the current mode.

ST = dwtmode or ST = dwtmode('status') display and returns in ST the current mode.

ST = dwtmode('status','nodisp') returns in ST the current mode and no text (status or warning) is displayed in the MATLAB® Command Window.

dwtmode('mode') sets the DWT extension mode according to the value of 'mode':

'mode'	DWT Extension Mode
'sym' or 'symh'	Symmetric-padding (half-point): boundary value symmetric replication — default mode
'symw'	Symmetric-padding (whole-point): boundary value symmetric replication
'asym' or 'asymh'	Antisymmetric-padding (half-point): boundary value antisymmetric replication
'asymw'	Antisymmetric-padding (whole-point): boundary value antisymmetric replication
'zpd'	Zero-padding
'spd' or 'sp1'	Smooth-padding of order 1 (first derivative interpolation at the edges)

<i>'mode'</i>	<b>DWT Extension Mode</b>
<i>'sp0'</i>	Smooth-padding of order 0 (constant extension at the edges)
<i>'ppd'</i>	Periodic-padding (periodic extension at the edges)

For more information on symmetric extension modes see “References”.

The DWT associated with these five modes is slightly redundant. But, the IDWT ensures a perfect reconstruction for any of the five previous modes whatever is the extension mode used for DWT.

`dwtmode('per')` sets the DWT mode to periodization.

This mode produces the smallest length wavelet decomposition. But, the extension mode used for IDWT must be the same to ensure a perfect reconstruction.

Using this mode, `dwt` and `dwt2` produce the same results as the obsolete functions `dwtper` and `dwtper2`, respectively.

All functions and GUI tools involving the DWT (1-D & 2-D) or Wavelet Packet transform (1-D & 2-D) use the specified DWT extension mode.

`dwtmode` updates a global variable allowing the use of these six signal extensions. The extension mode should only be changed using this function. Avoid changing the global variable directly.

The default mode is loaded from the file `DWTMODE.DEF` (in the current path) if it exists. If not, the file `DWTMODE.CFG` (in the `toolbox/wavelet/wavelet` directory) is used.

`dwtmode('save',MODE)` saves `MODE` as the new default mode in the file `DWTMODE.DEF` (in the current directory). If a file with the same name already exists in the current directory, it is deleted before saving.

`dwtmode('save')` is equivalent to `dwtmode('save',CURRENTMODE)`.

In these last two cases, the new default mode saved in the file `DWTMODE.DEF` will be active as default mode in the next MATLAB session.

**Examples**

```

% If the DWT extension mode global variable does not
% exist, default is Symmetrization.
clear global
dwtmode

*****
**  DWT Extension Mode: Symmetrization  **
*****

% Display current DWT signal extension mode.
dwtmode

*****
**  DWT Extension Mode: Symmetrization  **
*****

% Change to Periodization extension mode.
dwtmode('per')

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  WARNING: Change DWT Extension Mode  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

*****
**  DWT Extension Mode: Periodization  **
*****

% Display current DWT signal extension mode.
dwtmode

*****
**  DWT Extension Mode: Periodization  **
*****

```

---

**Note** You should change the extension mode only by using `dwtmode`. Avoid changing the global variable directly.

---

**See Also**

`idwt`, `idwt2`, `dwt`, `dwt2`, `wextend`

## References

Strang, G.; T. Nguyen (1996), *Wavelets and filter banks*, Wellesley- Cambridge Press.

**Purpose** Dyadic downsampling

**Syntax**

```
Y = dyaddown(X,EVENODD)
Y = dyaddown(X)
Y = dyaddown(X,EVENODD,'type')
Y = dyaddown(X,'type',EVENODD)
```

**Description**  $Y = \text{dyaddown}(X, \text{EVENODD})$  where  $X$  is a *vector*, returns a version of  $X$  that has been downsampled by 2. Whether  $Y$  contains the even- or odd-indexed samples of  $X$  depends on the value of positive integer  $\text{EVENODD}$ :

- If  $\text{EVENODD}$  is even, then  $Y(k) = X(2k)$ .
- If  $\text{EVENODD}$  is odd, then  $Y(k) = X(2k+1)$ .

$Y = \text{dyaddown}(X)$  is equivalent to  $Y = \text{dyaddown}(X, 0)$  (even-indexed samples).

$Y = \text{dyaddown}(X, \text{EVENODD}, 'type')$  or  $Y = \text{dyaddown}(X, 'type', \text{EVENODD})$ , where  $X$  is a *matrix*, returns a version of  $X$  obtained by suppressing one out of two:

Columns of $X$	If $'type' = 'c'$
Rows of $X$	If $'type' = 'r'$
Rows and columns of $X$	If $'type' = 'm'$

according to the parameter  $\text{EVENODD}$ , which is as above.

If you omit the  $\text{EVENODD}$  or  $'type'$  arguments,  $\text{dyaddown}$  defaults to  $\text{EVENODD} = 0$  (even-indexed samples) and  $'type' = 'c'$  (columns).

$Y = \text{dyaddown}(X)$  is equivalent to  $Y = \text{dyaddown}(X, 0, 'c')$ .

$Y = \text{dyaddown}(X, 'type')$  is equivalent to  $Y = \text{dyaddown}(X, 0, 'type')$ .

$Y = \text{dyaddown}(X, \text{EVENODD})$  is equivalent to  $Y = \text{dyaddown}(X, \text{EVENODD}, 'c')$ .

## Examples

```
% For a vector.
s = 1:10
s =
     1     2     3     4     5     6     7     8     9    10

dse = dyaddown(s) % Downsample elements with even indices.
```

# dyaddown

---

```
dse =  
    2    4    6    8   10  
% or equivalently  
dse = dyaddown(s,0)  
dse =  
    2    4    6    8   10  
  
dso = dyaddown(s,1) % Downsample elements with odd indices.  
dso =  
    1    3    5    7    9  
  
% For a matrix.  
s = (1:3)'*(1:4)  
s =  
    1    2    3    4  
    2    4    6    8  
    3    6    9   12  
  
dec = dyaddown(s,0,'c') % Downsample columns with even indices.  
dec =  
    2    4  
    4    8  
    6   12  
  
der = dyaddown(s,1,'r') % Downsample rows with odd indices.  
der =  
    1    2    3    4  
    3    6    9   12  
  
dem = dyaddown(s,1,'m') % Downsample rows and columns  
% with odd indices.  
dem =  
    1    3  
    3    9
```

## See Also

dyadup

## References

Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press.



**Purpose** Dyadic upsampling

**Syntax**

```
Y = dyadup(X,EVENODD)
Y = dyadup(X)
Y = dyadup(X,EVENODD,'type')
Y = dyadup(X,'type',EVENODD)
```

**Description** dyadup implements a simple zero-padding scheme very useful in the wavelet reconstruction algorithm.

$Y = \text{dyadup}(X, \text{EVENODD})$ , where  $X$  is a *vector*, returns an extended copy of vector  $X$  obtained by inserting zeros. Whether the zeros are inserted as even- or odd-indexed elements of  $Y$  depends on the value of positive integer  $\text{EVENODD}$ :

- If  $\text{EVENODD}$  is even, then  $Y(2k-1) = X(k)$ ,  $Y(2k) = 0$ .
- If  $\text{EVENODD}$  is odd, then  $Y(2k-1) = 0$ ,  $Y(2k) = X(k)$ .

$Y = \text{dyadup}(X)$  is equivalent to  $Y = \text{dyadup}(X, 1)$  (odd-indexed samples).

$Y = \text{dyadup}(X, \text{EVENODD}, 'type')$  or  $Y = \text{dyadup}(X, 'type', \text{EVENODD})$ , where  $X$  is a *matrix*, returns extended copies of  $X$  obtained by inserting

Columns in $X$	If $'type' = 'c'$
Rows in $X$	If $'type' = 'r'$
Rows and columns in $X$	If $'type' = 'm'$

according to the parameter  $\text{EVENODD}$ , which is as above.

If you omit the  $\text{EVENODD}$  or  $'type'$  arguments,  $\text{dyadup}$  defaults to  $\text{EVENODD} = 1$  (zeros in odd-indexed positions) and  $'type' = 'c'$  (insert columns).

$Y = \text{dyadup}(X)$  is equivalent to  $Y = \text{dyadown}(X, 1, 'c')$ .

$Y = \text{dyadup}(X, 'type')$  is equivalent to  $Y = \text{dyadup}(X, 1, 'type')$ .

$Y = \text{dyadup}(X, \text{EVENODD})$  is equivalent to  $Y = \text{dyadup}(X, \text{EVENODD}, 'c')$ .

# dyadup

---

## Examples

```
% For a vector.
s = 1:5
s =
    1 2 3 4 5

dse = dyadup(s) % Upsample elements at odd indices.
dse =
    0 1 0 2 0 3 0 4 0 5 0

% or equivalently
dse = dyadup(s,1)
dse =
    0 1 0 2 0 3 0 4 0 5 0

dso = dyadup(s,0) % Upsample elements at even indices.
dso =
    1 0 2 0 3 0 4 0 5

% For a matrix.
s = (1:2)'*(1:3)
s =
    1 2 3
    2 4 6

der = dyadup(s,1,'r') % Upsample rows at even indices.
der =
    0 0 0
    1 2 3
    0 0 0
    2 4 6
    0 0 0

doc = dyadup(s,0,'c') % Upsample columns at odd indices.
doc =
    1 0 2 0 3
    2 0 4 0 6
```

```
dem = dyadup(s,1,'m') % Upsample rows and columns
                        % at even indices.
dem =
    0     0     0     0     0     0     0
    0     1     0     2     0     3     0
    0     0     0     0     0     0     0
    0     2     0     4     0     6     0
    0     0     0     0     0     0     0

% Using default values for dyadup and dyaddown, we have:
% dyaddown(dyadup(s)) = s.
s = 1:5
s =
    1  2  3  4  5

uds = dyaddown(dyadup(s))
uds =
    1  2  3  4  5

% In general reversed identity is false.
```

**See Also**

dyaddown

**References**

Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press.

# entrupd

---

**Purpose** Entropy update (wavelet packet)

**Syntax** T = entrupd(T,ENT)  
T = entrupd(T,ENT,PAR)

**Description** entrupd is a one- or two-dimensional wavelet packet utility.  
T = entrupd(T,ENT) or T = entrupd(T,ENT,PAR) returns for a given wavelet packet tree T, the updated tree using the entropy function ENT with the optional parameter PAR (see wenergy for more information).

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).  
  
% Load signal.  
load noisdopp; x = noisdopp;  
  
% Decompose x at depth 2 with db1 wavelet packets  
% using shannon entropy.  
t = wpdec(x,2,'db1','shannon');  
  
% Read entropy of all the nodes.  
nodes = allnodes(t);  
ent = read(t,'ent',nodes);  
ent'  
ent =  
    1.0e+04 *  
    -5.8615 -6.8204 -0.0350 -7.7901 -0.0497 -0.0205 -0.0138  
  
% Update nodes entropy.  
t = entrupd(t,'threshold',0.5);  
nent = read(t,'ent');  
nent'  
nent =  
    937 488 320 241 175 170 163
```

**See Also** wenergy, wpdec, wpdec2

**Purpose** Complex frequency B-spline wavelet

**Syntax** [PSI,X] = fbspwavf(LB,UB,N,M,FB,FC)

**Description** [PSI,X] = fbspwavf(LB,UB,N,M,FB,FC) returns values of the complex frequency B-Spline wavelet defined by the order parameter M (M is an integer such that  $1 \leq M$ ), a bandwidth parameter FB, and a wavelet center frequency FC.

The function PSI is computed using the explicit expression

$$\text{PSI}(X) = (\text{FB}^{0.5}) * ((\text{sinc}(\text{FB}*X/\text{M}).^{\text{M}}) .* \exp(2*i*\pi*\text{FC}*X))$$

on an N point regular grid in the interval [LB,UB].

FB and FC must be such that  $\text{FC} > 0$  and  $\text{FB} > 0$ .

Output arguments are the wavelet function PSI computed on the grid X.

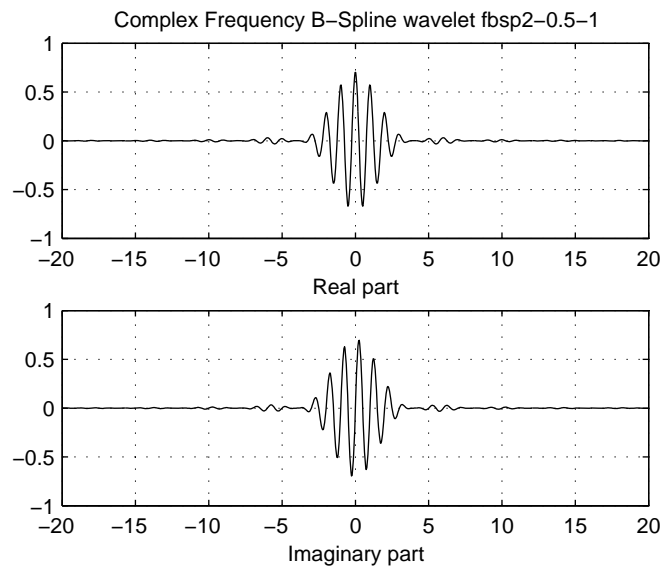
**Examples**

```
% Set order, bandwidth and center frequency parameters.
m = 2; fb = 1; fc = 0.5;

% Set effective support and grid parameters.
lb = -20; ub = 20; n = 1000;

% Compute complex Frequency B-Spline wavelet fbsp2-0.5-1.
[psi,x] = fbspwavf(lb,ub,n,m,fb,fc);
```

```
% Plot complex Frequency B-Spline wavelet.  
subplot(211)  
plot(x,real(psi))  
title('Complex Frequency B-Spline wavelet fbsp2-0.5-1')  
xlabel('Real part'), grid  
subplot(212)  
plot(x,imag(psi))  
xlabel('Imaginary part'), grid
```



## See Also

`waveinfo`

## References

Teolis, A. (1998), *Computational signal processing with wavelets*, Birkhauser, p. 63.

**Purpose** Transform quadruplet of filters to lifting scheme

**Syntax** `LS = filt2ls(LoD,HiD,LoR,HiR)`

**Description** `LS = filt2lsv(LoD,HiD,LoR,HiR)` returns the lifting scheme LS associated with the four input filters LoD, HiD, LoR, and HiR that verify the perfect reconstruction condition.

**Examples**

```
[LoD,HiD,LoR,HiR] = wfilters('db2')

LoD =

    -0.1294    0.2241    0.8365    0.4830

HiD =

    -0.4830    0.8365   -0.2241   -0.1294

LoR =

    0.4830    0.8365    0.2241   -0.1294

HiR =

    -0.1294   -0.2241    0.8365   -0.4830

LS = filt2ls(LoD,HiD,LoR,HiR);
displs(LS);

LS = {...
'd'          [ -1.73205081]          [0]
'p'          [ -0.06698730  0.43301270] [1]
'd'          [  1.00000000]          [-1]
[ 1.93185165] [  0.51763809]          []
};

LSref = liftwave('db2');
displs(LSref);
```

# filt2ls

---

```
LSref = {...  
'd'          [ -1.73205081]          [0]  
'p'          [ -0.06698730  0.43301270] [1]  
'd'          [  1.00000000]          [-1]  
[  1.93185165] [  0.51763809]          []  
};
```

## See Also

ls2filt, lsinfo



**Purpose** Gaussian wavelet

**Syntax** `[PSI,X] = gauswavf(LB,UB,N,P)`

**Description** `[PSI,X] = gauswavf(LB,UB,N,P)` returns values of the  $P$ -th derivative of the Gaussian function on an  $N$  point regular grid for the interval  $[LB,UB]$ .  $C_p$  is such that the 2-norm of the  $P$ -th derivative of  $F$  is equal to 1.

For  $P > 8$ , Symbolic Math Toolbox™ software is required.

Output arguments are the wavelet function  $PSI$  computed on the grid  $X$ .

`[PSI,X] = gauswavf(LB,UB,N)` is equivalent to

`[PSI,X] = gauswavf(LB,UB,N,1)`.

These wavelets have an effective support of  $[-5\ 5]$ .

**Examples**

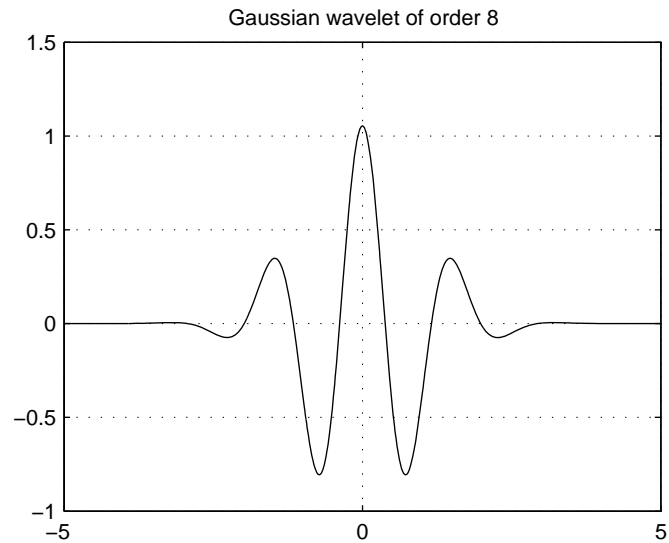
```
% Set effective support and grid parameters.
lb = -5; ub = 5; n = 1000;

% Compute Gaussian wavelet of order 8.
[psi,x] = gauswavf(lb,ub,n,8);
```

# gauswavf

---

```
% Plot Gaussian wavelet of order 8.  
plot(x,psi),  
title('Gaussian wavelet of order 8'), grid
```



## See Also

`waveinfo`

<b>Purpose</b>	WPTREE contents
<b>Syntax</b>	[FieldValue1,FieldValue2, ] = get(T,'FieldName1','FieldName2', ) [FieldValue1,FieldValue2, ] = get(T)
<b>Description</b>	<p>[FieldValue1,FieldValue2, ] = get(T,'FieldName1','FieldName2', ) returns the content of the specified fields for the WPTREE object T.</p> <p>For the fields that are objects or structures, you can get the subfield contents, giving the name of these subfields as '<i>FieldName</i>' values. (See “Examples” below.)</p> <p>[FieldValue1,FieldValue2, ] = get(T) returns all the field contents of the tree T.</p> <p>The valid choices for '<i>FieldName</i>' are</p> <ul style="list-style-type: none"> <li>'dtree' DTREE parent object</li> <li>'wavInfo' Structure (wavelet information)</li> </ul> <p>The fields of the wavelet information structure, 'wavInfo', are also valid for '<i>FieldName</i>':</p> <ul style="list-style-type: none"> <li>'wavName' Wavelet name</li> <li>'Lo_D' Low Decomposition filter</li> <li>'Hi_D' High Decomposition filter</li> <li>'Lo_R' Low Reconstruction filter</li> <li>'Hi_R' High Reconstruction filter</li> </ul> <p>'entInfo' Structure (entropy information)</p> <p>The fields of the entropy information structure, 'entInfo', are also valid for '<i>FieldName</i>':</p> <ul style="list-style-type: none"> <li>'entName' Entropy name</li> <li>'entPar' Entropy parameter</li> </ul>

Or fields of DTREE parent object:

'ntree'     NTREE parent object  
'allNI'     All nodes information  
'terNI'     Terminal nodes information

Or fields of NTREE parent object:

'wtbo'     WTBO parent object  
'order'     Order of the tree  
'depth'     Depth of the tree  
'spsch'     Split scheme for nodes  
'tn'        Array of terminal nodes of the tree

Or fields of WTBO parent object:

'wtboInfo'   Object information  
'ud'         Userdata field

## Examples

```
% Compute a wavelet packets tree
x = rand(1,1000);
t = wpdec(x,2,'db2');
o = get(t,'order');
[o,tn] = get(t,'order','tn');
[o,allNI,tn] = get(t,'order','allNI','tn');
[o,wavInfo,allNI,tn] = get(t,'order','wavInfo','allNI','tn');
[o,tn,Lo_D,EntName] = get(t,'order','tn','Lo_D','EntName');
[wo,nt,dt] = get(t,'wtbo','ntree','dtree');
```

## See Also

disp, read, set, write

**Purpose** Single-level inverse discrete 1-D wavelet transform

**Syntax**

```
X = idwt(cA,cD,'wname')
X = idwt(cA,cD,Lo_R,Hi_R)
X = idwt(cA,cD,'wname',L)
X = idwt(cA,cD,Lo_R,Hi_R,L)
X = idwt(...,'mode',MODE)
```

**Description** The `idwt` command performs a single-level one-dimensional wavelet reconstruction with respect to either a particular wavelet ('*wname*', see `wfilters` for more information) or particular wavelet reconstruction filters (`Lo_R` and `Hi_R`) that you specify.

`X = idwt(cA,cD,'wname')` returns the single-level reconstructed approximation coefficients vector `X` based on approximation and detail coefficients vectors `cA` and `cD`, and using the wavelet '*wname*'.

`X = idwt(cA,cD,Lo_R,Hi_R)` reconstructs as above using filters that you specify.

- `Lo_R` is the reconstruction low-pass filter.
- `Hi_R` is the reconstruction high-pass filter.

`Lo_R` and `Hi_R` must be the same length.

Let `1a` be the length of `cA` (which also equals the length of `cD`) and `1f` the length of the filters `Lo_R` and `Hi_R`; then `length(X) = LX` where  $LX = 2 \cdot 1a$  if the DWT extension mode is set to periodization. For the other extension modes  $LX = 2 \cdot 1a - 1f + 2$ .

For more information about the different Discrete Wavelet Transform extension modes, see `dwtmode`.

`X = idwt(cA,cD,'wname',L)` or `X = idwt(cA,cD,Lo_R,Hi_R,L)` returns the length-`L` central portion of the result obtained using `idwt(cA,cD,'wname')`. `L` must be less than `LX`.

`X = idwt(...,'mode',MODE)` computes the wavelet reconstruction using the specified extension mode `MODE`.

`X = idwt(cA,[],...)` returns the single-level reconstructed approximation coefficients vector `X` based on approximation coefficients vector `cA`.

`X = idwt([],cD,...)` returns the single-level reconstructed detail coefficients vector `X` based on detail coefficients vector `cD`.

`idwt` is the inverse function of `dwt` in the sense that the abstract statement `idwt(dwt(X,'wname'),'wname')` would give back `X`.

## Examples

```
% The current extension mode is zero-padding (see dwtmode).

% Construct elementary one-dimensional signal s.

randn('seed',531316785)
s = 2 + kron(ones(1,8),[1 -1]) + ...
    ((1:16).^2)/32 + 0.2*randn(1,16);

% Perform single-level dwt of s using db2.

[ca1,cd1] = dwt(s,'db2');
subplot(221); plot(ca1);
title('Approx. coef. for db2');
subplot(222); plot(cd1);
title('Detail coef. for db2');

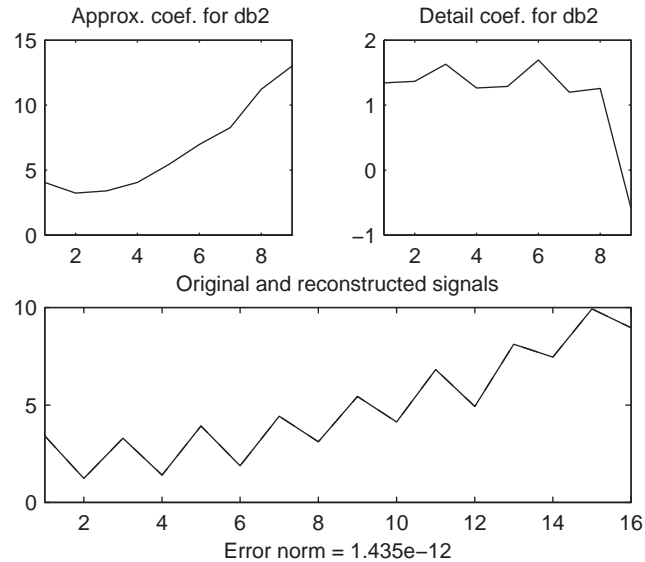
% Perform single-level inverse discrete wavelet transform,
% illustrating that idwt is the inverse function of dwt.

ss = idwt(ca1,cd1,'db2');
err = norm(s-ss); % Check reconstruction.
subplot(212); plot([s;ss]');
title('Original and reconstructed signals');
xlabel(['Error norm = ',num2str(err)])

% For a given wavelet, compute the two associated
% reconstruction filters and inverse transform using
% the filters directly.

[Lo_R,Hi_R] = wfilters('db2','r');
ss = idwt(ca1,cd1,Lo_R,Hi_R);
```

```
% Using some plotting commands,
% the following figure is generated.
```

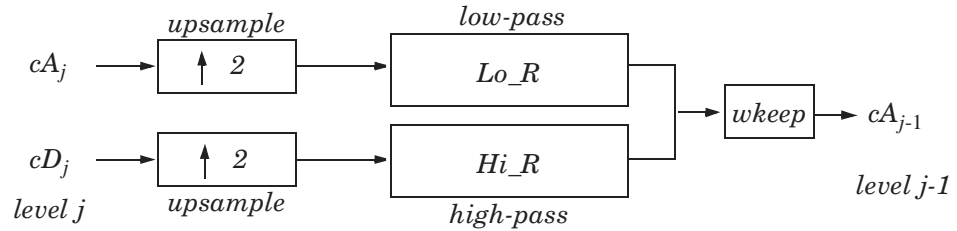





## Algorithm

Starting from the approximation and detail coefficients at level  $j$ ,  $cA_j$  and  $cD_j$ , the inverse discrete wavelet transform reconstructs  $cA_{j-1}$ , inverting the decomposition step by inserting zeros and convolving the results with the reconstruction filters.

One-Dimensional IDWT

Reconstruction step



- Where
-  Insert zeros at odd-indexed elements
  -  Convolve with filter X
  -  Take the central part of U with the convenient length

See Also

dwt, dwtmode, upwlev

References

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

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.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)



**Purpose** Single-level inverse discrete 2-D wavelet transform

**Syntax**

```
X = idwt2(cA,cH,cV,cD,'wname')
X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R)
X = idwt2(cA,cH,cV,cD,'wname',S)
X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R,S)
X = idwt2(...,'mode',MODE)
```

**Description** The `idwt2` command performs a single-level two-dimensional wavelet reconstruction with respect to either a particular wavelet (`'wname'`, see `wfilters` for more information) or particular wavelet reconstruction filters (`Lo_R` and `Hi_R`) that you specify.

`X = idwt2(cA,cH,cV,cD,'wname')` uses the wavelet `'wname'` to compute the single-level reconstructed approximation coefficients matrix `X`, based on approximation matrix `cA` and details matrices `cH`, `cV`, and `cD` (horizontal, vertical, and diagonal, respectively).

`X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R)` reconstructs as above, using filters that you specify.

- `Lo_R` is the reconstruction low-pass filter.
- `Hi_R` is the reconstruction high-pass filter.

`Lo_R` and `Hi_R` must be the same length.

Let `sa = size(cA) = size(cH) = size(cV) = size(cD)` and `lf` the length of the filters; then `size(X) = SX`, where `SX = 2* SA`, if the DWT extension mode is set to periodization. For the other extension modes, `SX = 2*size(cA) - lf + 2`.

For more information about the different Discrete Wavelet Transform extension modes, see `dwtmode`.

`X = idwt2(cA,cH,cV,cD,'wname',S)` and `X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R,S)` return the size-`S` central portion of the result obtained using the syntax `idwt2(cA,cH,cV,cD,'wname')`. `S` must be less than `SX`.

`X = idwt2(...,'mode',MODE)` computes the wavelet reconstruction using the extension mode `MODE` that you specify.

`X = idwt2(cA,[],[],[],...)` returns the single-level reconstructed approximation coefficients matrix `X` based on approximation coefficients matrix `cA`.

`X = idwt2([],cH,[],[],...)` returns the single-level reconstructed detail coefficients matrix `X` based on horizontal detail coefficients matrix `cH`.

The same result holds for `X = idwt2([],[],cV,[],...)` and `X = idwt2([],[],[],cD,...)`, based on vertical and diagonal details.

More generally, `X = idwt2(AA,HH,VV,DD,...)` returns the single-level reconstructed matrix `X`, where `AA` can be `cA` or `[]`, and so on.

`idwt2` is the inverse function of `dwt2` in the sense that the abstract statement `idwt2(dwt2(X,'wname'),'wname')` would give back `X`.

## Remarks

If `cA`, `cH`, `cV`, `cD` are obtained from an indexed image analysis or a truecolor image analysis, they are `m`-by-`n` matrices or `m`-by-`n`-by-3 arrays, respectively.

For more information on image formats, see the `image` and `imfinfo` reference pages.

## Examples

```
% The current extension mode is zero-padding (see dwtmode).

% Load original image.
load woman;

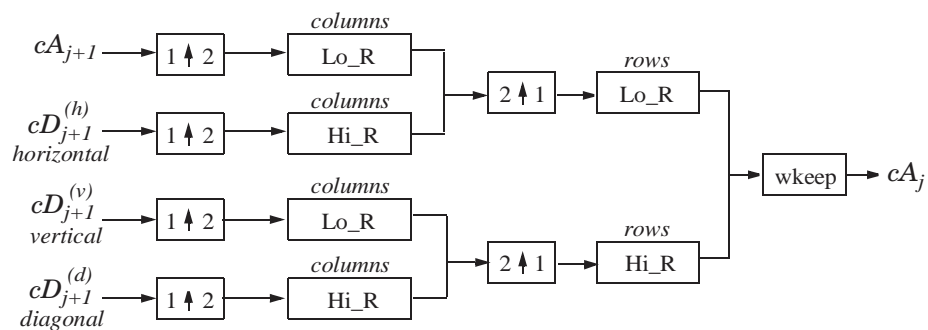
% X contains the loaded image.
sX = size(X);

% Perform single-level decomposition
% of X using db4.
[cA1,cH1,cV1,cD1] = dwt2(X,'db4');

% Invert directly decomposition of X
% using coefficients at level 1.
A0 = idwt2(cA1,cH1,cV1,cD1,'db4',sX);

% Check for perfect reconstruction.
max(max(abs(X-A0)))
ans =
```

3.4176e-10

**Algorithm****Two-Dimensional IDWT****Reconstruction step**

Where

$2 \uparrow 1$	Upsample columns: insert zeros at odd-indexed columns.
$1 \uparrow 2$	Upsample rows: insert zeros at odd-indexed rows.
$\begin{matrix} \text{rows} \\ \text{X} \end{matrix}$	Convolve with filter X the rows of the entry.
$\begin{matrix} \text{columns} \\ \text{X} \end{matrix}$	Convolve with filter X the columns of the entry.

**See Also**

dwt2, dwtmode, upw1ev2

**Purpose** Inverse 1-D lifting wavelet transform

**Syntax**

```
X = ilwt(AD_In_Place,W)
X = ilwt(CA,CD,W)
X = ilwt(AD_In_Place,W,LEVEL)
X = ilwt(CA,CD,W,LEVEL)
X = ilwt(AD_In_Place,W,LEVEL,'typeDEC',typeDEC)
X = ilwt(CA,CD,W,LEVEL,'typeDEC',typeDEC)
```

**Description** `ilwt` performs a 1-D lifting wavelet reconstruction with respect to a particular lifted wavelet that you specify.

`X = ilwt(AD_In_Place,W)` computes the reconstructed vector `X` using the approximation and detail coefficients vector `AD_In_Place` obtained by a lifting wavelet reconstruction. `W` is a lifted wavelet name (see `liftwave`).

`X = ilwt(CA,CD,W)` computes the reconstructed vector `X` using the approximation coefficients vector `CA` and detail coefficients vector `CD` obtained by a lifting wavelet reconstruction.

`X = ilwt(AD_In_Place,W,LEVEL)` or `X = ILWT(CA,CD,W,LEVEL)` computes the lifting wavelet reconstruction, at level `LEVEL`.

`X = ilwt(AD_In_Place,W,LEVEL,'typeDEC',typeDEC)` or `X = ilwt(CA,CD,W,LEVEL,'typeDEC',typeDEC)` with `typeDEC = 'w'` or `'wp'` computes the wavelet or the wavelet packet decomposition using lifting, at level `LEVEL`.

Instead of a lifted wavelet name, you may use the associated lifting scheme `LS`:  
`X = ilwt(...,LS,...)` instead of `X = ILWT(...,W,...)`.

For more information about lifting schemes, see `lsinfo`.

**Examples**

```
% Start from the Haar wavelet and get the
% corresponding lifting scheme.
lshaar = liftwave('haar');

% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);
```

```
% Perform LWT at level 1 of a simple signal.
x = 1:8;
[cA,cD] = lwt(x,lsnew);

% Perform integer LWT of the same signal.
lshaarInt = liftwave('haar','int2int');
lsnewInt = addlift(lshaarInt,els);
[cAint,cDint] = lwt(x,lsnewInt);

% Invert the two transforms.
xRec = ilwt(cA,cD,lsnew);
err = max(max(abs(x-xRec)))

err =

    4.4409e-016

xRecInt = ilwt(cAint,cDint,lsnewInt);
errInt = max(max(abs(x-xRecInt)))

errInt =

    0
```

**See Also**

lwt

# ilwt2

---

**Purpose** Inverse 2-D lifting wavelet transform

**Syntax**

```
X = ilwt2(AD_In_Place,W)
X = ilwt2(CA,CH,CV,CD,W)
X = ilwt2(AD_In_Place,W,LEVEL)
X = ilwt2(CA,CH,CV,CD,W,LEVEL)
X = ilwt2(AD_In_Place,W,LEVEL,'typeDEC',typeDEC)
X = ilwt2(CA,CH,CV,CD,W,LEVEL,'typeDEC',typeDEC)
```

**Description** `ilwt2` performs a 2-D lifting wavelet reconstruction with respect to a particular lifted wavelet that you specify.

`X = ilwt2(AD_In_Place,W)` computes the reconstructed matrix `X` using the approximation and detail coefficients matrix `AD_In_Place`, obtained by a lifting wavelet decomposition. `W` is a lifted wavelet name (see `liftwave`).

`X = ilwt2(CA,CH,CV,CD,W)` computes the reconstructed matrix `X` using the approximation coefficients vector `CA` and detail coefficients vectors `CH`, `CV`, and `CD` obtained by a lifting wavelet decomposition.

`X = ilwt2(AD_In_Place,W,LEVEL)` or `X = ILWT2(CA,CH,CV,CD,W,LEVEL)` computes the lifting wavelet reconstruction, at level `LEVEL`.

`X = ilwt2(AD_In_Place,W,LEVEL,'typeDEC',typeDEC)` or  
`X = ilwt2(CA,CH,CV,CD,W,LEVEL,'typeDEC',typeDEC)` with `typeDEC = 'w'`  
or `'wp'` computes the wavelet or the wavelet packet decomposition using lifting, at level `LEVEL`.

Instead of a lifted wavelet name, you may use the associated lifting scheme `LS`:  
`X = ilwt2(...,LS,...)` instead of `X = ilwt2(...,W,...)`.

For more information about lifting schemes, see `lsinfo`.

**Remarks** If `AD_In_Place` or `cA`, `cH`, `cV`, `cD` are obtained from an indexed image analysis or a truecolor image analysis, they are `m`-by-`n` matrices or `m`-by-`n`-by-3 arrays, respectively.

For more information on image formats, see the `image` and `imfinfo` reference pages.

**Examples** `% Start from the Haar wavelet and get the`

```
% corresponding lifting scheme.
lshaar = liftwave('haar');

% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);

% Perform LWT at level 1 of a simple image.
x = reshape(1:16,4,4);
[cA,cH,cV,cD] = lwt2(x,lsnew);

% Perform integer LWT of the same image.
lshaarInt = liftwave('haar','int2int');
lsnewInt = addlift(lshaarInt,els);
[cAint,cHint,cVint,cDint] = lwt2(x,lsnewInt);

% Invert the two transforms.
xRec = ilwt2(cA,cH,cV,cD,lsnew);
err = max(max(abs(x-xRec)))

err =

    0

xRecInt = ilwt2(cAint,cHint,cVint,cDint,lsnewInt);
errInt = max(max(abs(x-xRecInt)))

errInt =

    0
```

**See Also**

lwt2

# ind2depo

---

**Purpose** Node index to node depth-position

**Syntax** `[D,P] = ind2depo(ORD,N)`

**Description** `ind2depo` is a tree-management utility.

For a tree of order `ORD`, `[D,P] = ind2depo(ORD,N)` computes the depths `D` and the positions `P` (at these depths `D`) for the nodes with indices `N`.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

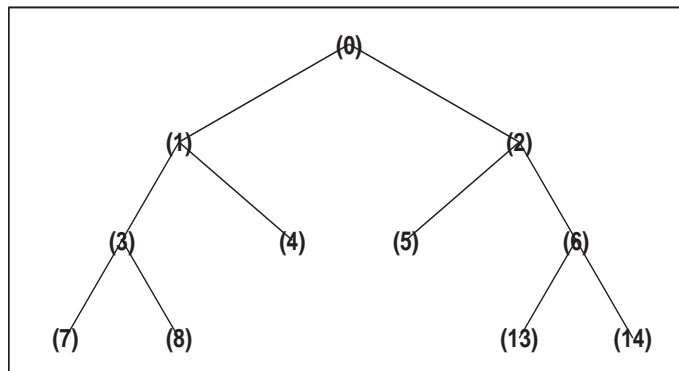
`N` must be a column vector of integers (`N >= 0`).

Note that `[D,P] = ind2depo(ORD,[D P])`.

## Examples

```
% Create initial tree.
ord = 2; t = ntree(ord,3);    % Binary tree of depth 3.
t = nodejoin(t,5);
t = nodejoin(t,4);
plot(t)

% Change Node Label from Depth_Position to Index
% (see the plot function).
```





```
% List t nodes (index).
aln_ind = allnodes(t)

aln_ind =
    0
    1
    2
    3
    4
    5
    6
    7
    8
   13
   14

% Switch from index to Depth_Position.
[depth,pos] = ind2depo(ord,aln_ind);
aln_depo = [depth,pos]

aln_depo =
    0     0
    1     0
    1     1
    2     0
    2     1
    2     2
    2     3
    3     0
    3     1
    3     6
    3     7
```

**See Also** `depo2ind`

# intwave

---

## Purpose

Integrate wavelet function  $\psi$

## Syntax

```
[INTEG,XVAL] = intwave('wname',PREC)
[INTEG,XVAL] = intwave('wname',PREC,PFLAG)
[INTEG,XVAL] = intwave('wname')
```

## Description

[INTEG,XVAL] = intwave('wname',PREC) computes the integral, INTEG, of the wavelet function  $\psi$  (from  $-\infty$  to XVAL values):  $\int_{-\infty}^x \psi(y)dy$  for  $x$  in XVAL.

The function  $\psi$  is approximated on the  $2^{\text{PREC}}$  points grid XVAL where PREC is a positive integer. 'wname' is a string containing the name of the wavelet  $\psi$  (see wfilters for more information).

Output argument INTEG is a real or complex vector depending on the wavelet type.

For biorthogonal wavelets,

[INTDEC,XVAL,INTREC] = intwave('wname',PREC) computes the integrals, INTDEC and INTREC, of the wavelet decomposition function  $\psi_{\text{dec}}$  and the wavelet reconstruction function  $\psi_{\text{rec}}$ .

```
[INTEG,XVAL] = intwave('wname',PREC) is equivalent to
[INTEG,XVAL] = intwave('wname',PREC,0).
```

```
[INTEG,XVAL] = intwave('wname') is equivalent to
[INTEG,XVAL] = intwave('wname',8).
```

When used with three arguments intwave('wname',IN2,IN3), PREC = max(IN2,IN3) and plots are given.

When IN2 is equal to the special value 0, intwave('wname',0) is equivalent to intwave('wname',8,IN3).

intwave('wname') is equivalent to intwave('wname',8).

intwave is used only for continuous analysis (see cwt for more information).

**Examples**

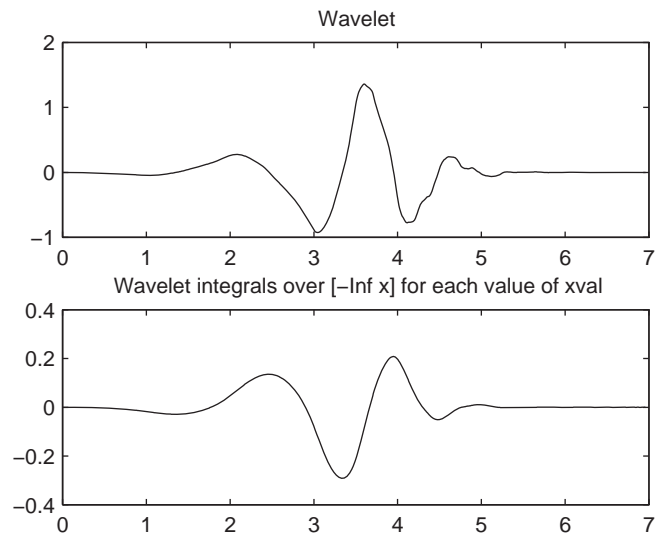
```

% Set wavelet name.
wname = 'db4';

% Plot wavelet function.
[phi,psi,xval] = wavefun(wname,7);
subplot(211); plot(xval,psi); title('Wavelet');

% Compute and plot wavelet integrals approximations
% on a dyadic grid.
[integ,xval] = intwave(wname,7);
subplot(212); plot(xval,integ);
title(['Wavelet integrals over [-Inf x] ' ...
      'for each value of xval']);

```

**Algorithm**

First, the wavelet function is approximated on a grid of  $2^{\text{PREC}}$  points using `wavefun`. A piecewise constant interpolation is used to compute the integrals using `cumsum`.

**See Also**

`wavefun`

# isnode

**Purpose** Existing node test

**Syntax** `R = isnode(T,N)`

**Description** `isnode` is a tree-management utility.

`R = isnode(T,N)` returns 1's for nodes `N`, which exist in the tree `T`, and 0's for others.

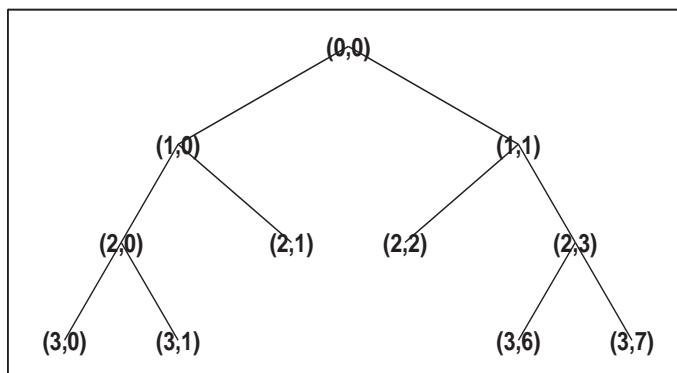
`N` can be a column vector containing the indices of nodes or a matrix, that contains the depths and positions of nodes.

In the last case, `N(i,1)` is the depth of the *i*-th node and `N(i,2)` is the position of the *i*-th node.

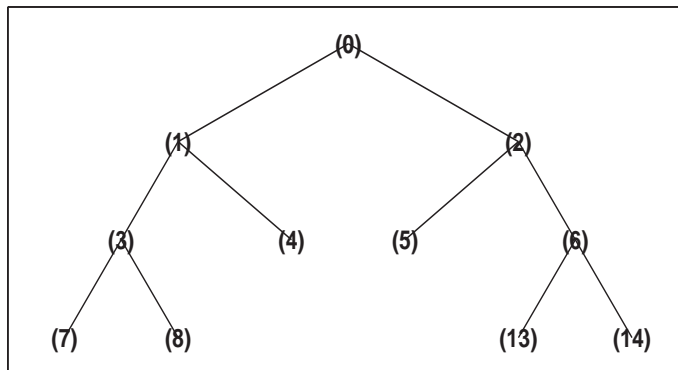
The nodes are numbered from left to right and from top to bottom. The root index is 0.

## Examples

```
% Create initial tree.  
ord = 2;  
t = ntree(ord,3);    % binary tree of depth 3.  
t = nodejoin(t,5);  
t = nodejoin(t,4);  
plot(t)
```



```
% Change Node Label from Depth_Position to Index
% (see the plot function).
```



```
% Check node index.
isnode(t,[1;3;25])
```

```
ans =
     1
     1
     0
```

```
% Check node Depth_Position.
isnode(t,[1 0;3 1;4 5])
```

```
ans =
     1
     1
     0
```

## See Also

istnode, wtreemgr

# istnode

**Purpose** Terminal nodes indices test

**Syntax** `R = istnode(T,N)`

**Description** `istnode` is a tree-management utility.

`R = istnode(T,N)` returns ranks (in left to right terminal nodes ordering) for terminal nodes `N` belonging to the tree `T`, and 0's for others.

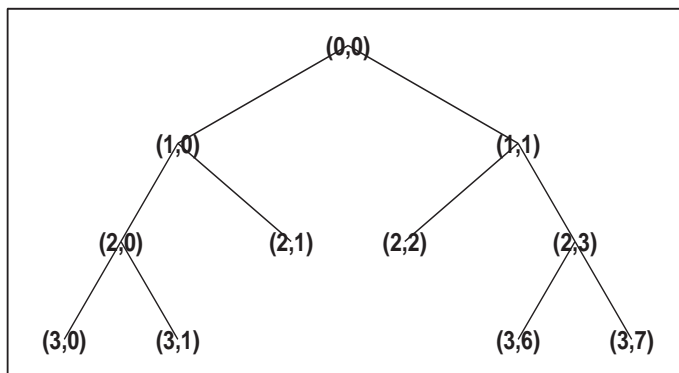
`N` can be a column vector containing the indices of nodes or a matrix that contains the depths and positions of nodes.

In the last case, `N(i,1)` is the depth of the *i*-th node and `N(i,2)` is the position of the *i*-th node.

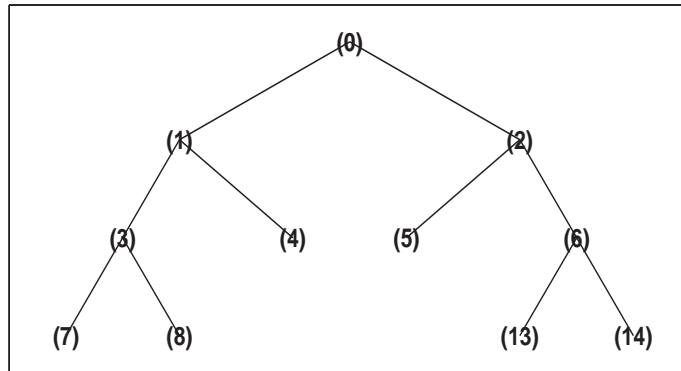
The nodes are numbered from left to right and from top to bottom. The root index is 0.

## Examples

```
% Create initial tree.
ord = 2;
t = ntree(ord,3); % binary tree of depth 3.
t = nodejoin(t,5);
t = nodejoin(t,4);
plot(t)
```



```
% Change Node Label from Depth_Position to Index
% (see the plot function)x.
```



```
% Find terminal nodes and return indices for terminal
% nodes in the tree.
```

```
istnode(t,[14])
```

```
ans =
     6
```

```
istnode(t,[15])
```

```
ans =
     0
```

```
istnode(t,[1;7;14;25])
```

```
ans =
     0
     1
     6
     0
```

```
istnode(t,[1 0;3 1;4 5])
```

```
ans =
     0
     2
     0
```

## See Also

isnode, wtreemgr

# iswt

---

**Purpose** Inverse discrete stationary wavelet transform 1-D

**Syntax**

```
X = iswt(SWC, 'wname')
X = iswt(SWA, SWD, 'wname')
X = iswt(SWC, Lo_R, Hi_R)
X = iswt(SWA, SWD, Lo_R, Hi_R)
```

**Description** iswt performs a multilevel 1-D stationary wavelet reconstruction using either a specific orthogonal wavelet ('wname', see `wfilters` for more information) or specific reconstruction filters (Lo\_R and Hi\_R).

```
X = iswt(SWC, 'wname') or X = iswt(SWA, SWD, 'wname') or
X = iswt(SWA(end, :), SWD, 'wname')
```

 reconstructs the signal X based on the multilevel stationary wavelet decomposition structure SWC or [SWA, SWD] (see `swt` for more information).

```
X = iswt(SWC, Lo_R, Hi_R) or X = iswt(SWA, SWD, Lo_R, Hi_R) or
X = iswt(SWA(end, :), SWD, Lo_R, Hi_R)
```

 reconstruct as above, using filters that you specify.

- Lo\_R is the reconstruction low-pass filter.
- Hi\_R is the reconstruction high-pass filter.

Lo\_R and Hi\_R must be the same length.

**Examples**

```
% Load original 1D signal.
load noisbloc; s = noisbloc;

% Perform SWT decomposition at level 3 of s using db1.
swc = swt(s,3,'db1');
% Second usage.
[swa,swd] = swt(s,3,'db1');

% Reconstruct s from the stationary wavelet
% decomposition structure swc.
a0 = iswt(swc,'db1');
% Second usage.
a0bis = iswt(swa,swd,'db1');
```



```
% Check for perfect reconstruction.
err = norm(s-a0)
err =
    9.6566e-014

errbis = norm(s-a0bis)
errbis =
    9.6566e-014
```

**Algorithm** See the section “Stationary Wavelet Transform” in Chapter 6, “Advanced Concepts”, of the User’s Guide.

**See Also** idwt, swt, waverec

**References** Nason, G.P.; B.W. Silverman (1995), “The stationary wavelet transform and some statistical applications,” *Lecture Notes in Statistics*, 103, pp. 281–299.

Coifman, R.R.; Donoho D.L. (1995), “Translation invariant de-noising,” *Lecture Notes in Statistics*, 103, pp 125–150.

Pesquet, J.C.; H. Krim, H. Carfatan (1996), “Time-invariant orthonormal wavelet representations,” *IEEE Trans. Sign. Proc.*, vol. 44, 8, pp. 1964–1970.

# iswt2

---

**Purpose** Inverse discrete stationary wavelet transform 2-D

**Syntax**

```
X = iswt2(SWC, 'wname')
X = iswt2(A,H,V,D, 'wname')
X = iswt2(SWC,Lo_R,Hi_R)
X = iswt2(A,H,V,D,Lo_R,Hi_R)
```

**Description** `iswt2` performs a multilevel 2-D stationary wavelet reconstruction using either a specific orthogonal wavelet ('wname' see `wfilters` for more information) or specific reconstruction filters (`Lo_R` and `Hi_R`).

```
X = iswt2(SWC, 'wname') or X = iswt2(A,H,V,D, 'wname') or
X = iswt2(A(:, :, end), H, V, D, 'wname')
```

reconstructs the signal `X`, based on the multilevel stationary wavelet decomposition structure `SWC` or `[A,H,V,D]` (see `swt2`).

```
X = iswt2(SWC,Lo_R,Hi_R) or X = iswt2(A,H,V,D,Lo_R,Hi_R) or
X = iswt2(A(:, :, end), H, V, D, Lo_R, Hi_R)
```

reconstructs as in the previous syntax, using filters that you specify:

- `Lo_R` is the reconstruction low-pass filter.
- `Hi_R` is the reconstruction high-pass filter.

`Lo_R` and `Hi_R` must be the same length.

**Remarks** If `SWC` or (`cA`, `cH`, `cV`, `cD`) are obtained from an indexed image analysis or a truecolor image analysis, then `X` is an `m`-by-`n` matrix or an `m`-by-`n`-by-3 array, respectively.

For more information on image formats, see the `image` and `imfinfo` reference pages.

**Examples**

```
% Load original image.
load nbarb1;

% Perform SWT decomposition
% of X at level 3 using sym4.
swc = swt2(X,3,'sym4');
% Second usage.
[ca,chd,cvd,cdd] = swt2(X,3,'sym4');
```

```
% Reconstruct s from the stationary wavelet
% decomposition structure swc.
a0 = iswt2(swc,'sym4');
% Second usage.
a0 = iswt2(ca,chd,cvd,cdd,'sym4');
```

```
% Check for perfect reconstruction.
err = max(max(abs(X-a0)))
ans =
    2.3482e-010

errbis = max(max(abs(X-a0bis)))
ans =
    2.3482e-010
```

## Algorithm

See the section “Stationary Wavelet Transform” in Chapter 6, “Advanced Concepts”, of the User’s Guide.

## See Also

idwt2, swt2, waverec2

## References

- Nason, G.P.; B.W. Silverman (1995), “The stationary wavelet transform and some statistical applications,” *Lecture Notes in Statistics*, 103, pp. 281–299.
- Coifman, R.R.; Donoho D.L. (1995), “Translation invariant de-noising,” *Lecture Notes in Statistics*, 103, pp. 125–150.
- Pesquet, J.C.; H. Krim, H. Carfatan (1996), “Time-invariant orthonormal wavelet representations,” *IEEE Trans. Sign. Proc.*, vol. 44, 8, pp. 1964–1970.

**Purpose** Laurent matrices constructor

**Syntax** `M = laurmat(V)`

**Description** `M = laurmat(V)` returns the Laurent matrix object `M` associated with `V` which can be a cell array (at most two dimensional) of Laurent polynomials (see `laurpoly`) or an ordinary matrix.

**Examples**

```
% Define Laurent matrices.
M1 = laurmat(eye(2,2))
```

$$M1 = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$$

```
Z = laurpoly(1,1);
M2 = laurmat({1 Z;0 1})
```

$$M2 = \begin{vmatrix} 1 & z^{(+1)} \\ 0 & 1 \end{vmatrix}$$

```
% Calculus on Laurent polynomials.
P = M1 * M2
```

$$P = \begin{vmatrix} 1 & z^{(+1)} \\ 0 & 1 \end{vmatrix}$$

```
d = det(P)
```

```
d(z) = 1
```

## References

Strang, G.; T. Nguyen (1996), *Wavelets and filter banks*, Wellesley-Cambridge Press.

Sweldens, W. (1998), "The Lifting Scheme: a Construction of Second Generation of Wavelets," *SIAM J. Math. Anal.*, 29 (2), pp. 511–546.

## See Also

laurpoly

**Purpose** Laurent polynomials constructor

**Syntax**

```
P = laurpoly(C,d)
P = laurpoly(C,'dmin',d)
P = laurpoly(C,'dmax',d)
```

**Description** `P = laurpoly(C,d)` returns a Laurent polynomial object. `C` is a vector whose elements are the coefficients of the polynomial `P` and `d` is the highest degree of the monomials of `P`.

If `m` is the length of the vector `C`, `P` represents the following Laurent polynomial:

$$P(z) = C(1)*z^d + C(2)*z^{(d-1)} + \dots + C(m)*z^{(d-m+1)}$$

`P = laurpoly(C,'dmin',d)` specifies the lowest degree instead of the highest degree of monomials of `P`. The corresponding output `P` represents the following Laurent polynomial:

$$P(z) = C(1)*z^{(d+m-1)} + \dots + C(m-1)*z^{(d+1)} + C(m)*z^d$$

`P = laurpoly(C,'dmax',d)` is equivalent to `P = laurpoly(C,d)`.

## Examples

```
% Define Laurent polynomials.
P = laurpoly([1:3],2);
P = laurpoly([1:3],'dmax',2)

P(z) = + z^(+2) + 2*z^(+1) + 3

P = laurpoly([1:3],'dmin',2)

P(z) = + z^(+4) + 2*z^(+3) + 3*z^(+2)

% Calculus on Laurent polynomials.
Z = laurpoly(1,1)

Z(z) = z^(+1)

Q = Z*P

Q(z) = + z^(+5) + 2*z^(+4) + 3*z^(+3)
```

$$R = Z^1 - Z^{-1}$$

$$R(z) = + z^{(+1)} - z^{(-1)}$$

## References

Strang, G.; T. Nguyen (1996), *Wavelets and filter banks*, Wellesley-Cambridge Press.

Sweldens, W. (1998), "The Lifting Scheme: a Construction of Second Generation of Wavelets," *SIAM J. Math. Anal.*, 29 (2), pp. 511–546.

## See Also

laurmat



**Purpose** Determine terminal nodes

**Syntax**

```
N = leaves(T)
N = leaves(T, 'dp')
[N,K] = leaves(T, 'sort')
[N,K] = leaves(T, 'sortdp')
```

**Description** `N = leaves(T)` returns the indices of terminal nodes of the tree `T` where `N` is a column vector.

The nodes are ordered from left to right as in tree `T`.

`[N,K] = leaves(T, 's')` or `[N,K] = leaves(T, 'sort')` returns sorted indices. `M = N(K)` are the indices reordered as in tree `T`, from left to right.

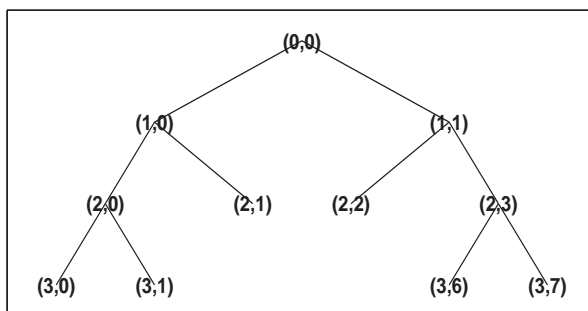
`N = leaves(T, 'dp')` returns a matrix `N`, which contains the depths and positions of terminal nodes.

`N(i,1)` is the depth of the `i`-th terminal node, and `N(i,2)` is the position of the `i`-th terminal node.

`[N,K] = leaves(T, 'sortdp')` or `[N,K] = leaves(T, 'sdp')` returns sorted nodes.

## Examples

```
% Create initial tree.
ord = 2;
t = ntree(ord,3);           % binary tree of depth 3.
t=nodejoin(t,5);
t=nodejoin(t,4);
plot(t)
```



# leaves

---

```
% List terminal nodes (index).
tnodes_ind = leaves(t)
tnodes_ind =
     7
     8
     4
     5
    13
    14

% List terminal nodes (sorted on index).
[tnodes_ind,Ind] = leaves(t,'sort')
tnodes_ind =
     4
     5
     7
     8
    13
    14

Ind =
     3
     4
     1
     2
     5
     6

% List terminal nodes (Depth_Position).
tnodes_depo = leaves(t,'dp')
tnodes_depo =
     3     0
     3     1
     2     1
     2     2
     3     6
     3     7

% List terminal nodes (sorted on Depth_Position).
[tnodes_depo,Ind] = leaves(t,'sortdp')
```

```
tnodes_depo =
```

```
  2    1
```

```
  2    2
```

```
  3    0
```

```
  3    1
```

```
  3    6
```

```
  3    7
```

```
Ind =
```

```
  3
```

```
  4
```

```
  1
```

```
  2
```

```
  5
```

```
  6
```

## See Also

tnodes, noleaves

# liftfilt

---

**Purpose** Apply elementary lifting steps on quadruplet of filters

**Syntax**  
`[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,HiD,LoR,HiR,ELS)`  
`[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,HiD,LoR,HiR,ELS,TYPE,VALUE)`

**Description** `[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,HiD,LoR,HiR,ELS)` returns the four filters LoDN, HiDN, LoRN, and HiRN obtained by an elementary lifting step (ELS) starting from the four filters LoD, HiD, LoR, and HiR. The four input filters verify the perfect reconstruction condition.

ELS is a structure such that

- `TYPE = ELS.type` contains the type of the elementary lifting step. The valid values for TYPE are 'p' (primal) or 'd' (dual).
- `VALUE = ELS.value` contains the Laurent polynomial T associated with the elementary lifting step (see `laurpoly`). If VALUE is a vector, the associated Laurent polynomial T is equal to `laurpoly(VALUE,0)`.

In addition, ELS may be a scaling step. In that case, TYPE is equal to 's' (scaling) and VALUE is a scalar different from zero.

`liftfilt(LoD,HiD,LoR,HiR,ELS,TYPE,VALUE)` gives the same outputs.

---

**Note** If TYPE = 'p', HiD and LoR are unchanged.

If TYPE = 'd', LoD and HiR are unchanged.

If TYPE = 's', the four filters are changed.

If ELS is an array of elementary lifting steps, `liftfilt(...,ELS)` performs each step successively.

---

`liftfilt(...,FLAGPLOT)` plots the successive biorthogonal pairs—scaling function and wavelet.

## Examples

```
% 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 insignificant sign.
[LoDB,HiDB,LoRB,HiRB] = wfilters('bior1.3');
samewavelet =
isequal([LoDB,HiDB,LoRB,HiRB],[LoDN,-HiDN,LoRN,HiRN])

samewavelet =

    1
```

**See Also**

laurpoly

# liftwave

**Purpose** Lifting schemes

**Syntax**  
LS = liftwave(WNAME)  
LS = liftwave(WNAME, 'Int2Int')

**Description** LS = liftwave(WNAME) returns the lifting scheme associated with the wavelet specified by WNAME. LS is a structure, not an integer, and used by lwt, ilwt, lwt2, etc.

LS = liftwave(WNAME, 'Int2Int') performs an integer to integer wavelet transform. Using 'Int2Int' produces an LS such that when you use [CA,CD] = lwt(X,LS) or Y = lwt(X,LS) and X is a vector of integers, the resulting CA, CD, and Y are vectors of integers. If you omit 'Int2Int' then lwt produces vectors of real numbers.

The valid values for WNAME are

WNAME Values	Comments
'lazy'	A “lazy” wavelet is a second-generation wavelet and is not a true mathematical wavelet.
'haar'	Same as 'db1', 'bior1.1', and 'cdf1.1'
'db1', 'db2', 'db3', 'db4', 'db5', 'db6', 'db7', 'db8'	'db2' same as 'sym2', 'db3', and 'sym4'
'sym2', 'sym3', 'sym4', 'sym5', 'sym6', 'sym7', 'sym8'	

WNAME Values	Comments
Cohen-Daubechies-Feauveau wavelets 'cdf1.1', 'cdf1.3', 'cdf1.5' 'cdf3.1', 'cdf3.3', 'cdf3.5' 'cdf5.1', 'cdf5.3', 'cdf5.5' 'cdf2.2', 'cdf2.4', 'cdf2.6' 'cdf4.2', 'cdf4.4', 'cdf4.6' 'cdf6.2', 'cdf6.4', 'cdf6.6'	'cdfX.Y' same as 'biorX.Y' except for bior4.4 and bior5.5.
'biorX.Y'	See waveinfo
'rbioX.Y'	Reverse of 'biorX.Y'. See waveinfo
'bs3'	Same as 'cdf4.2'
'rbs3'	Reverse of 'bs3'
'9.7'	Same as 'bior4.4'
'r9.7'	Reverse of '9.7'

For more information about lifting schemes, see `lsinfo`.

### Examples

```
% Start from the db2 wavelet and get the
% corresponding lifting scheme.
lsdb2 = liftwave('db2');

% Visualize the obtained lifting scheme.
displs(lsdb2);

lsdb2 = {...
'd'          [ -1.73205081]          [0]
'p'          [ -0.06698730  0.43301270] [1]
'd'          [  1.00000000]          [-1]
[  1.93185165] [  0.51763809]          []
};
```

### See Also

`laurpoly`

# localmax

---

**Purpose** Compute local maxima positions

**Syntax**  
`[Y,I] = localmax(X,ROWINIT,REGFLAG)`  
`[Y,I] = localmax(X,ROWINIT)`  
`[Y,I] = localmax(X)`

**Description** For a matrix  $X$ , `localmax` computes and chains the local maxima along the rows.

The default values are `ROWINIT = size(X,1)` and `REGFLAG = true`.

First, `localmax` computes the local maxima positions on each row of  $X$ . Then, starting from the row  $(\text{ROWINIT}-1)$ , `localmax` chains the maxima positions along the columns. If  $p_0$  is a local maxima position on the row  $R_0$ , then  $p_0$  is linked to the nearest maxima position on the row  $R_0+1$ .

$Y$  is a matrix of the same size of  $X$  such that:

- When  $R = \text{ROWINIT}$ , then, if  $X(\text{ROWINIT}, j)$  is a local maximum,  $Y(\text{ROWINIT}, j) = j$  and otherwise,  $Y(\text{ROWINIT}, j) = 0$ .
- When  $R < \text{ROWINIT}$ , if  $X(R, j)$  is not a local maximum then  $Y(R, j) = 0$ .
- Otherwise if  $X(R, j)$  is a local maximum, then  $Y(R, j) = k$ , where  $k$  is such that  $X(R+1, k)$  is a local maximum and  $k$  is the nearest position of  $j$ .

$I$  contains the indices of nonzero values of  $Y$ .

If `REGFLAG = true`,  $S = X(\text{ROWINIT}, :)$  is first regularized using the wavelet 'sym4'. Instead of  $S$ , the approximation of level 5 is used to start the algorithm.



**Purpose** Transform lifting scheme to quadruplet of filters

**Syntax** [LoD,HiD,LoR,HiR] = ls2filt(LS)

**Description** [LoD,HiD,LoR,HiR] = ls2filt(LS) returns the four filters LoD, HiD, LoR, and HiR associated with the lifting scheme LS.

**Examples**

```
% Start from the db2 wavelet and get the
% corresponding lifting scheme.
LS = liftwave('db2')

LS =

    'd'          [ -1.7321]    [ 0]
    'p'          [1x2 double]  [ 1]
    'd'          [          1]  [-1]
    [1.9319]     [ 0.5176]    []

% Visualize the obtained lifting scheme.

displs(LS);

LS = {...
    'd'          [ -1.73205081]          [0]
    'p'          [ -0.06698730  0.43301270] [1]
    'd'          [ 1.00000000]          [-1]
    [ 1.93185165] [ 0.51763809]          []
};

% Get the filters from the lifting scheme.

[LoD,HiD,LoR,HiR] = ls2filt(LS)

LoD =

    -0.1294    0.2241    0.8365    0.4830

HiD =
```

# ls2filt

---

```
-0.4830    0.8365   -0.2241   -0.1294
LoR =
    0.4830    0.8365    0.2241   -0.1294
HiR =
   -0.1294   -0.2241    0.8365   -0.4830
% Get the db2 filters using wfilters.
% You can check the equality.
[LoDref,HiDref,LoRref,HiRref] = wfilters('db2')
LoDref =
   -0.1294    0.2241    0.8365    0.4830
HiDref =
   -0.4830    0.8365   -0.2241   -0.1294
LoRref =
    0.4830    0.8365    0.2241   -0.1294
HiRref =
   -0.1294   -0.2241    0.8365   -0.4830
```

## See Also

`filt2ls`, `lsinfo`

**Purpose** Lifting schemes information

**Syntax** lsinfo

**Description** lsinfo displays the following information about lifting schemes. A lifting scheme LS is a  $N \times 3$  cell array. The  $N-1$  first rows of the array are elementary lifting steps (ELS). The last row gives the normalization of LS.

Each ELS has this format:

```
{type, coefficients, max_degree}
```

where `type` is 'p' (primal) or 'd' (dual), `coefficients` is a vector  $C$  of real numbers defining the coefficients of a Laurent polynomial  $P$  described below, and `max_degree` is the highest degree  $d$  of the monomials of  $P$ .

The Laurent polynomial  $P$  is of the form

$$P(z) = C(1)*z^d + C(2)*z^{(d-1)} + \dots + C(m)*z^{(d-m+1)}$$

The lifting scheme LS is such that for

$k = 1:N-1$ ,  $LS\{k, : \}$  is an ELS, where

$LS\{k, 1\}$  is the lifting type 'p' (primal) or 'd' (dual).

$LS\{k, 2\}$  is the corresponding lifting filter.

$LS\{k, 3\}$  is the highest degree of the Laurent polynomial corresponding to the filter  $LS\{k, 2\}$ .

$LS\{N, 1\}$  is the primal normalization (real number).

$LS\{N, 2\}$  is the dual normalization (real number).

$LS\{N, 3\}$  is not used.

Usually, the normalizations are such that  $LS\{N, 1\} * LS\{N, 2\} = 1$ .

For example, the lifting scheme associated with the wavelet db1 is

```
LS = { ...
      'd'          [ -1]   [0]
      'p'          [0.5000] [0]
      [1.4142]     [0.7071] []
      }
```

# lsinfo

---

## See Also

displs, laurpoly

<b>Purpose</b>	1-D lifting wavelet transform
<b>Syntax</b>	<pre>[CA,CD] = lwt(X,W) X_InPlace = lwt(X,W) lwt(X,W,LEVEL) X_InPlace = lwt(X,W,LEVEL,'typeDEC',typeDEC) [CA,CD] = lwt(X,W,LEVEL,'typeDEC',typeDEC)</pre>
<b>Description</b>	<p>lwt performs a 1-D lifting wavelet decomposition with respect to a particular lifted wavelet that you specify.</p> <p>[CA,CD] = lwt(X,W) computes the approximation coefficients vector CA and detail coefficients vector CD, obtained by a lifting wavelet decomposition, of the vector X. W is a lifted wavelet name (see liftwave).</p> <p>X_InPlace = lwt(X,W) computes the approximation and detail coefficients. These coefficients are stored in place:</p> <p>CA = X_InPlace(1:2:end) and CD = X_InPlace(2:2:end)</p> <p>lwt(X,W,LEVEL) computes the lifting wavelet decomposition at level LEVEL.</p> <p>X_InPlace = lwt(X,W,LEVEL,'typeDEC',typeDEC) or  [CA,CD] = lwt(X,W,LEVEL,'typeDEC',typeDEC) with typeDEC = 'w' or 'wp' computes the wavelet or the wavelet packet decomposition using lifting, at level LEVEL.</p> <p>Instead of a lifted wavelet name, you may use the associated lifting scheme LS: lwt(X,LS,...) instead of lwt(X,W,...).</p> <p>For more information about lifting schemes, see lsinfo.</p>
<b>Examples</b>	<pre>% Start from the Haar wavelet and get the % corresponding lifting scheme. lshaar = liftwave('haar');  % Add a primal ELS to the lifting scheme. els = {'p',[-0.125 0.125],0}; lsnew = addlift(lshaar,els);  % Perform LWT at level 1 of a simple signal. x = 1:8;</pre>

```
[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 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
```

## Algorithm

This function uses the polyphase algorithm.

`lwt` reduces to `dwt` with zero-padding extension mode and without extra-coefficients.

## See Also

`ilwt`

## References

Strang, G.; T. Nguyen (1996), *Wavelets and filter banks*, Wellesley-Cambridge Press.

Sweldens, W. (1998), "The Lifting Scheme: a Construction of Second Generation of Wavelets," *SIAM J. Math. Anal.*, 29 (2), pp. 511–546.

<b>Purpose</b>	2-D lifting wavelet transform
<b>Syntax</b>	<pre>[ CA,CH,CV,CD ] = lwt2(X,W) X_InPlace = lwt2(X,LS) lwt2(X,W,LEVEL) X_InPlace = lwt2(X,W,LEVEL, 'typeDEC', typeDEC) [ CA,CD ] = lwt2(X,W,LEVEL, 'typeDEC', typeDEC)</pre>
<b>Description</b>	<p><code>lwt2</code> performs a 2-D lifting wavelet decomposition with respect to a particular lifted wavelet that you specify.</p> <p><code>[ CA,CH,CV,CD ] = lwt2(X,W)</code> computes the approximation coefficients matrix <code>CA</code> and detail coefficients matrices <code>CH</code>, <code>CV</code>, and <code>CD</code>, obtained by a lifting wavelet decomposition, of the matrix <code>X</code>. <code>W</code> is a lifted wavelet name (see <code>liftwave</code>).</p> <p><code>X_InPlace = lwt2(X,LS)</code> computes the approximation and detail coefficients. These coefficients are stored in place:</p> <ul style="list-style-type: none"> <li>• <code>CA = X_InPlace(1:2:end, 1:2:end)</code></li> <li>• <code>CH = X_InPlace(2:2:end, 1:2:end)</code></li> <li>• <code>CV = X_InPlace(1:2:end, 2:2:end)</code></li> <li>• <code>CD = X_InPlace(2:2:end, 2:2:end)</code></li> </ul> <p><code>lwt2(X,W,LEVEL)</code> computes the lifting wavelet decomposition at level <code>LEVEL</code>.</p> <p><code>X_InPlace = lwt2(X,W,LEVEL, 'typeDEC', typeDEC)</code> or <code>[ CA,CH,CV,CD ] = LWT2(X,W,LEVEL, 'typeDEC', typeDEC)</code> with <code>typeDEC = 'w'</code> or <code>'wp'</code> computes the wavelet or the wavelet packet decomposition using lifting, at level <code>LEVEL</code>.</p> <p>Instead of a lifted wavelet name, you may use the associated lifting scheme <code>LS</code>: <code>lwt2(X,LS,...)</code> instead of <code>LWT2(X,W,...)</code>.</p> <p>For more information about lifting schemes, see <code>lsinfo</code>.</p>
<b>Remarks</b>	<p>When <code>X</code> represents an indexed image, <code>X</code>, as well as the output arrays <code>cA</code>, <code>cH</code>, <code>cV</code>, <code>cD</code>, or <code>X_InPlace</code> are <code>m</code>-by-<code>n</code> matrices. When <code>X</code> represents a truecolor image, it is an <code>m</code>-by-<code>n</code>-by-3 array, where each <code>m</code>-by-<code>n</code> matrix represents a red, green, or blue color plane concatenated along the third dimension.</p> <p>For more information on image formats, see the <code>image</code> and <code>imfinfo</code> reference pages .</p>

## Examples

```
% Start from the Haar wavelet and get the
% corresponding lifting scheme.
lshaar = liftwave('haar');

% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);

% Perform LWT at level 1 of a simple image.
x = reshape(1:16,4,4);
[cA,cH,cV,cD] = lwt2(x,lsnew)

cA =

    5.7500    22.7500
   10.0000    27.0000

cH =

    1.0000    1.0000
    1.0000    1.0000

cV =

    4.0000    4.0000
    4.0000    4.0000

cD =

    0     0
    0     0

% Perform integer LWT of the same image.
lshaarInt = liftwave('haar','int2int');
lsnewInt = addlift(lshaarInt,els);
[cAint,cHint,cVint,cDint] = lwt2(x,lsnewInt)
```



```
cAint =  
      3   11  
      5   13
```

```
cHint =  
      1   1  
      1   1
```

```
cVint =  
      4   4  
      4   4
```

```
cDint =  
      0   0  
      0   0
```

**Algorithm**

This function implements the polyphase algorithm.

lwt reduces to dwt with zero-padding extension mode and without extra-coefficients.

**See Also**

ilwt2

**References**

Strang, G.; T. Nguyen (1996), *Wavelets and filter banks*, Wellesley-Cambridge Press.

Sweldens, W. (1998), "The Lifting Scheme: a Construction of Second Generation of Wavelets," *SIAM J. Math. Anal.*, 29 (2), pp. 511–546.

# lwtcoef

**Purpose** Extract or reconstruct 1-D LWT wavelet coefficients

**Syntax**  
`Y = lwtcoef(TYPE,XDEC,LS,LEVEL,LEVEXT)`  
`Y = lwtcoef(TYPE,XDEC,W,LEVEL,LEVEXT)`

**Description** `Y = lwtcoef(TYPE,XDEC,LS,LEVEL,LEVEXT)` returns the coefficients or the reconstructed coefficients of level `LEVEXT`, extracted from `XDEC`, the LWT decomposition at level `LEVEL` obtained with the lifting scheme `LS`.

The valid values for `TYPE` are

TYPE Values	Description
'a'	Approximations
'd'	Details
'ca'	Coefficients of approximations
'cd'	Coefficients of details

`Y = lwtcoef(TYPE,XDEC,W,LEVEL,LEVEXT)` returns the same output using `W`, which is the name of a lifted wavelet.

## Examples

```
% Start from the Haar wavelet and get the
% corresponding lifting scheme.
lshaar = liftwave('haar');

% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);

% Perform LWT at level 2 of a simple signal.
x = 1:8;
xDec = lwt(x,lsnew,2)

xDec =

    4.3438    0.7071    2.1250    0.7071   13.0313    0.7071
    2.0000    0.7071
```

```
% Extract approximation coefficients of level 1.
ca1 = lwtcoef('ca',xDec,lsnew,2,1)

ca1 =

    1.9445    4.9497    7.7782   10.6066

% Reconstruct approximations and details.
a1 = lwtcoef('a',xDec,lsnew,2,1)

a1 =

    1.3750    1.3750    3.5000    3.5000    5.5000    5.5000
    7.5000    7.5000

a2 = lwtcoef('a',xDec,lsnew,2,2)

a2 =

    2.1719    2.1719    2.1719    2.1719    6.5156    6.5156
    6.5156    6.5156

d1 = lwtcoef('d',xDec,lsnew,2,1)

d1 =

   -0.3750    0.6250   -0.5000    0.5000   -0.5000    0.5000
   -0.5000    0.5000

d2 = lwtcoef('d',xDec,lsnew,2,2)

d2 =

   -0.7969   -0.7969    1.3281    1.3281   -1.0156   -1.0156
    0.9844    0.9844

% Check perfect reconstruction.
err = max(abs(x-a2-d2-d1))
```

# lwtcoef

---

err =

9.9920e-016

## See Also

ilwt, lwt

**Purpose** Extract or reconstruct 2-D LWT wavelet coefficients

**Syntax**  
`Y = lwtcoef2(TYPE,XDEC,LS,LEVEL,LEVEXT)`  
`Y = lwtcoef2(TYPE,XDEC,W,LEVEL,LEVEXT)`

**Description** `Y = lwtcoef2(TYPE,XDEC,LS,LEVEL,LEVEXT)` returns the coefficients or the reconstructed coefficients of level `LEVEXT`, extracted from `XDEC`, the LWT decomposition at level `LEVEL` obtained with the lifting scheme `LS`.

The valid values for `TYPE` are listed in this table.

<b>TYPE Values</b>	<b>Description</b>
'a'	Approximations
'h'	Horizontal details
'v'	Vertical details
'd'	Diagonal details
'ca'	Coefficients of approximations
'ch'	Coefficients of horizontal details
'cv'	Coefficients of vertical details
'cd'	Coefficients of diagonal details

`Y = lwtcoef2(TYPE,XDEC,W,LEVEL,LEVEXT)` returns the same output using `W`, which is the name of a lifted wavelet.

**Remarks** If `XDEC` is obtained from an indexed image analysis or a truecolor image analysis, it is an `m`-by-`n` matrix or an `m`-by-`n`-by-3 array, respectively.

For more information on image formats, see the `image` and `imfinfo` reference pages.

**Examples**

```
% Start from the Haar wavelet and get the
% corresponding lifting scheme.
lshaar = liftwave('haar');
```

# lwtcoef2

---

```
% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);

% Perform LWT at level 2 of a simple image.
x = reshape(1:16,4,4);
xDec = lwt2(x,lsnew,2)

xDec =

    27.4375    4.0000   17.0000    4.0000
     1.0000         0     1.0000         0
     4.2500    4.0000    0.0000    4.0000
     1.0000         0     1.0000         0

% Extract approximation coefficients of level 1.
ca1 = lwtcoef2('ca',xDec,lsnew,2,1)

ca1 =

     5.7500    22.7500
    10.0000    27.0000

% Reconstruct approximations and details.
a1 = lwtcoef2('a',xDec,lsnew,2,1)

a1 =

     2.8750     2.8750    11.3750    11.3750
     2.8750     2.8750    11.3750    11.3750
     5.0000     5.0000    13.5000    13.5000
     5.0000     5.0000    13.5000    13.5000

a2 = lwtcoef2('a',xDec,lsnew,2,2)

a2 =

     6.8594     6.8594     6.8594     6.8594
     6.8594     6.8594     6.8594     6.8594
```

```

        6.8594    6.8594    6.8594    6.8594
        6.8594    6.8594    6.8594    6.8594

```

```
h1 = lwtcoef2('h',xDec,lsnew,2,1)
```

```
h1 =
```

```

    -0.3750   -0.3750   -0.3750   -0.3750
     0.6250    0.6250    0.6250    0.6250
    -0.5000   -0.5000   -0.5000   -0.5000
     0.5000    0.5000    0.5000    0.5000

```

```
v1 = lwtcoef2('v',xDec,lsnew,2,1)
```

```
v1 =
```

```

    -1.5000    2.5000   -2.0000    2.0000
    -1.5000    2.5000   -2.0000    2.0000
    -1.5000    2.5000   -2.0000    2.0000
    -1.5000    2.5000   -2.0000    2.0000

```

```
d1 = lwtcoef2('d',xDec,lsnew,2,1)
```

```
d1 =
```

```

     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0

```

```
h2 = lwtcoef2('h',xDec,lsnew,2,2)
```

```
h2 =
```

```

    -0.7969   -0.7969   -0.7969   -0.7969
    -0.7969   -0.7969   -0.7969   -0.7969
     1.3281    1.3281    1.3281    1.3281
     1.3281    1.3281    1.3281    1.3281

```

```
v2 = lwtcoef2('v',xDec,lsnew,2,2)
```

# lwtcoef2

---

```
v2 =  
  
    -3.1875    -3.1875     5.3125     5.3125  
    -3.1875    -3.1875     5.3125     5.3125  
    -3.1875    -3.1875     5.3125     5.3125  
    -3.1875    -3.1875     5.3125     5.3125  
  
d2 = lwtcoef2('d',xDec,lsnew,2,2)  
  
d2 =  
  
    1.0e-015 *  
  
    0.2498     0.2498    -0.4163    -0.4163  
    0.2498     0.2498    -0.4163    -0.4163  
   -0.4163    -0.4163     0.6939     0.6939  
   -0.4163    -0.4163     0.6939     0.6939  
  
% Check perfect reconstruction.  
err = max(max(abs(x-a2-h2-v2-d2-h1-v1-d1)))  
  
err =  
  
    3.5527e-015
```

## See Also

ilwt2, lwt2



**Purpose** Multisignals 1-D clustering

**Syntax**  
`S = mdwtcluster(X)`  
`S = mdwtcluster(X, 'PropName1', PropVal1, 'PropName2', PropVal2, ...)`

**Description** `S = mdwtcluster(X)` constructs clusters from a hierarchical cluster tree. The input matrix `X` is decomposed in row direction using the DWT function with the haar wavelet and the maximum allowed level.

`S = mdwtcluster(X, 'PropName1', PropVal1, 'PropName2', PropVal2, ...)` allows you to modify some properties. The valid choices for `PropName` are:

`'dirDec'` 'r' (row) or 'c' (column). Default value is 'r'.

`'level'` Level of the DWT decomposition. Default value is:

`level=fix(log2(size(X,d)))`

where `d=1` or `d=2`, depending on the `dirDec` value.

`'wname'` Wavelet name used for DWT. Default value is 'haar'.

`'dwtEXTM'` DWT extension mode (see `dwtmode`).

`'pdist'` See Statistics Toolbox™ `pdist` function. Default value is 'euclidean'.

`'linkage'` See Statistics Toolbox™ `linkage` function. Default value is 'ward'.

# mdwtcluster

---

'maxclust'	Number of clusters. Default value is 6. The input variable can be a vector.
'lst2clu'	Cell array that contains the list of data to classify. If $N$ is the level of decomposition, the allowed name values for the cells are: <ul style="list-style-type: none"><li>• 's' — Signal</li><li>• 'aj' — Approximation at level <math>j</math></li><li>• 'dj' — Detail at level <math>j</math></li><li>• 'caj' — Coefficients of approximation at level <math>j</math></li><li>• 'cdj' — Coefficients of detail at level <math>j</math></li></ul> Default value is {'s'; 'ca1'; ...; 'caN'}.

The output structure  $S$  is such that for each partition  $j$ :

$S.Idx(:, j)$	Contains the cluster numbers obtained from the hierarchical cluster tree (see <code>cluster</code> in the Statistics Toolbox™ software).
$S.Incons(:, j)$	Contains the inconsistent values of each non-leaf node in the hierarchical cluster tree (see Statistics Toolbox™ software function <code>inconsistent</code> ).
$S.Corr(j)$	Contains the cophenetic correlation coefficients of the partition (see Statistics Toolbox™ software function <code>cophenet</code> ).

---

**Note** If `maxclustVal` is a vector, then `IdxCLU` is a multidimensional array such that `IdxCLU(:, j, k)` contains the cluster numbers obtained from the hierarchical cluster tree for  $k$  clusters.

---

## Examples

```
load elecsig10
lst2clu = {'s', 'ca1', 'ca3', 'ca6'};
```

```
% Compute the structure resulting from multisignal clustering
S = mdwtcluster(signals,'maxclust',4,'1st2clu',1st2clu)
```

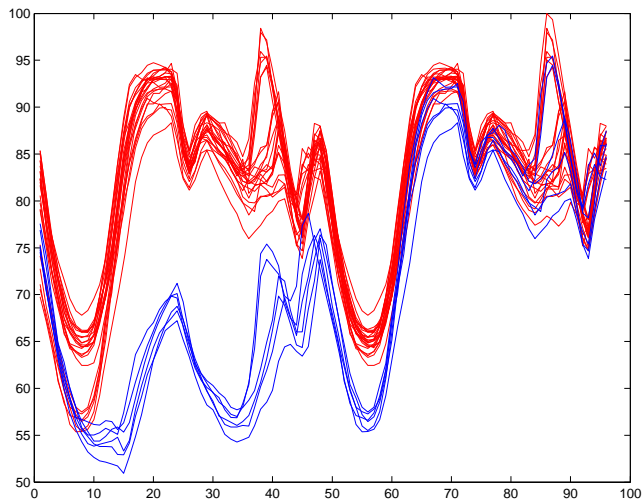
```
S =
```

```
IdxCLU: [70x4 double]
Incons: [69x4 double]
Corr: [0.7920 0.7926 0.7947 0.7631]
```

```
% Retrieve indices of clusters
IdxCLU = S.IdxCLU;
```

```
% Plot the first cluster
plot(signals(IdxCLU(:,1)==1,:), 'r');
hold on;
```

```
% Plot the third clustering
plot(signals(IdxCLU(:,1)==3,:), 'b')
```



```
% Check the equality of partitions
equalPART = isequal(IdxCLU(:,1),IdxCLU(:,3))
```

# mdwtcluster

---

```
equalPART =
```

```
1
```

```
% So we can see that we obtain the same partitions using  
% coefficients of approximation at level 3 instead of original  
% signals. Much less information is then used.
```

## See Also

mdwtdec, wavedec

**Purpose** Multisignal 1-D wavelet decomposition

**Syntax**

```
DEC = mdwtdec(DIRDEC,X,LEV,WNAME)
DEC = mdwtdec(DIRDEC,X,LEV,LoD,HiD,LoR,HiR)
DEC = mdwtdec(...,'mode',EXTMODE)
```

**Description** `DEC = mdwtdec(DIRDEC,X,LEV,WNAME)` returns the wavelet decomposition at level `LEV` of each row (if `DIRDEC = 'r'`) or each column (if `DIRDEC = 'c'`) of matrix `X`, using the wavelet `WNAME`.

The output `DEC` is a structure with the following fields:

<code>'dirDec'</code>	Direction indicator: <code>'r'</code> (row) or <code>'c'</code> (column)
<code>'level'</code>	Level of the DWT decomposition
<code>'wname'</code>	Wavelet name
<code>'dwtFilters'</code>	Structure with four fields <code>LoD</code> , <code>HiD</code> , <code>LoR</code> , and <code>HiR</code>
<code>'dwtEXTM'</code>	DWT extension mode (see <code>dwtmode</code> )
<code>'dwtShift'</code>	DWT shift parameter (0 or 1)
<code>'dataSize'</code>	Size of <code>X</code>
<code>'ca'</code>	Approximation coefficients at level <code>LEV</code>
<code>'cd'</code>	Cell array of detail coefficients, from level 1 to level <code>LEV</code>

Coefficients `ca` and `cd{k}` (for `k = 1` to `LEV`) are matrices and are stored in rows if `DIRDEC = 'r'` or in columns if `DIRDEC = 'c'`.

`DEC = mdwtdec(DIRDEC,X,LEV,LoD,HiD,LoR,HiR)` uses the four filters instead of the wavelet name.

`DEC = mdwtdec(...,'mode',EXTMODE)` computes the wavelet decomposition with the `EXTMODE` extension mode that you specify (see `dwtmode` for the valid extension modes).

**Examples**

```
% Load original 1D-multisignal.
load thinker
```

```
% Perform a decomposition at level 2 using wavelet db2.
dec = mdwtdec('r',X,2,'db2')
```

```
dec =
    dirDec: 'r'
    level: 2
    wname: 'db2'
    dwtFilters: [1x1 struct]
    dwtEXTM: 'sym'
    dwtShift: 0
    dataSize: [192 96]
    ca: [192x26 double]
    cd: {[192x49 double] [192x26 double]}

% Compute the associated filters of db2 wavelet.
[LoD,HiD,LoR,HiR] = wfilters('db2');

% Perform a decomposition at level 2 using filters.
decBIS = mdwtdec('r',X,2,LoD,HiD,LoR,HiR)

decBIS =
    dirDec: 'r'
    level: 2
    wname: ''
    dwtFilters: [1x1 struct]
    dwtEXTM: 'sym'
    dwtShift: 0
    dataSize: [192 96]
    ca: [192x26 double]
    cd: {[192x49 double] [192x26 double]}
```

## References

Daubechies, I. , *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed., 1992.

Mallat, S., “A theory for multiresolution signal decomposition: the wavelet representation,” *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, 1989, pp. 674–693.

Meyer, Y. , *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

## See Also

mdwtdec, wavedec

**Purpose** Multisignal 1-D wavelet reconstruction

**Syntax**

```
X = mdwtrec(DEC)
X = mdwtrec(DEC,IDXSIG)
Y = mdwtrec(DEC,TYPE,LEV)
A = mdwtrec(DEC,'a')
A = mdwtrec(DEC,'a',LEVDEC)
D = mdwtrec(DEC,'d')
CA = mdwtrec(DEC,'ca')
CA = mdwtrec(DEC,'ca',LEVDEC)
CD = mdwtrec(DEC,'cd',MODE)
CFS = mdwtrec(DEC,'cfs',MODE)
Y = mdwtrec(...,IDXSIG)
```

**Description** `X = mdwtrec(DEC)` returns the original matrix of signals, starting from the wavelet decomposition structure `DEC` (see `mdwtdec`).

`X = mdwtrec(DEC,IDXSIG)` reconstructs the signals whose indices are given by the vector `IDXSIG`.

`Y = mdwtrec(DEC,TYPE,LEV)` extracts or reconstructs the detail or approximation coefficients at level `LEV` depending on the `TYPE` value. The maximum value for `LEV` is `LEVDEC = DEC.level`.

When `TYPE` is equal to:

- `'cd'` or `'ca'`, coefficients of level `LEV` are extracted.
- `'d'` or `'a'`, coefficients of level `LEV` are reconstructed.
- `'a'` or `'ca'`, `LEV` must be such that  $0 \leq \text{LEV} \leq \text{LEVDEC}$ .
- `'d'` or `'cd'`, `LEV` must be such that  $1 \leq \text{LEV} \leq \text{LEVDEC}$ .

`A = mdwtrec(DEC,'a')` is equivalent to `A = mdwtrec(DEC,'a',LEVDEC)`.

`D = mdwtrec(DEC,'d')` returns a matrix containing the sum of all the details, so that  $X = A + D$ .

`CA = mdwtrec(DEC,'ca')` is equivalent to `CA = mdwtrec(DEC,'ca',LEVDEC)`.

`CD = mdwtrec(DEC, 'cd', MODE)` returns a matrix containing all the detail coefficients.

`CFS = mdwtrec(DEC, 'cfs', MODE)` returns a matrix containing all the coefficients.

For `MODE = 'ascend'` the coefficients are concatenated from level `LEVDEC` to level 1 and `MODE = 'descend'` concatenates from level 1 to level `LEVDEC`. The default is `MODE = 'descend'`. The concatenation is made row-wise if `DEC.dirDEC = 'r'` or column-wise if `DEC.dirDEC = 'c'`.

`Y = mdwtrec(..., IDXSIG)` extracts or reconstructs the detail or the approximation coefficients for the signals whose indices are given by the vector `IDXSIG`.

## Examples

```
% Load original 1D-multisignal.
load thinker

% Perform a decomposition at level 2 using wavelet db2.
dec = mdwtdec('r', X, 2, 'db2');

% Reconstruct the original matrix of signals, starting from
% the wavelet decomposition structure dec.
XR = mdwtrec(dec);

% Compute the reconstruction error.
errREC = max(max(abs(X-XR)))

errREC =
    2.1026e-010

% Reconstruct the original signal 31, the corresponding
% approximation at level 2, details at levels 1 and 2.
Y = mdwtrec(dec, 31);
A2 = mdwtrec(dec, 'a', 2, 31);
D2 = mdwtrec(dec, 'd', 2, 31);
D1 = mdwtrec(dec, 'd', 1, 31);

% Compute the reconstruction error for signal 31.
errREC = max(abs(Y-A2-D2-D1))
```



errREC =  
6.8390e-014

**References**

Daubechies, I., *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed., 1992.

Mallat, S., "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, 1989, pp. 674–693.

Meyer, Y., *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

**See Also**

mdwtdec, waverec

# mexihat

---

**Purpose** Mexican hat wavelet

**Syntax** [PSI,X] = mexihat(LB,UB,N)

**Description** [PSI,X] = mexihat(LB,UB,N) returns values of the Mexican hat wavelet on an N point regular grid, X, in the interval [LB,UB].

Output arguments are the wavelet function PSI computed on the grid X.

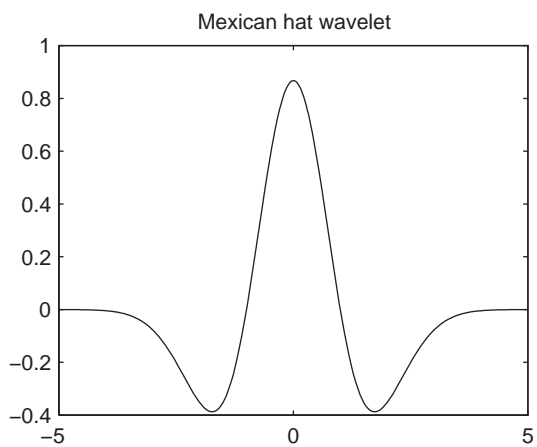
This wavelet has [-5 5] as effective support. Although [-5 5] is the correct theoretical effective support, a wider effective support, [-8 8], is used in the computation to provide more accurate results.

$$\psi(x) = \left(\frac{2}{\sqrt{3}}\pi^{-1/4}\right)(1-x^2)e^{-x^2/2}$$

This function is proportional to the second derivative function of the Gaussian probability density function.

**Examples**

```
% Set effective support and grid parameters.  
lb = -5; ub = 5; n = 1000;  
  
% Compute and plot Mexican hat wavelet.  
[psi,x] = mexihat(lb,ub,n);  
plot(x,psi), title('Mexican hat wavelet')
```



**See Also**

`waveinfo`

**Purpose**

Meyer wavelet

**Syntax**

```
[PHI,PSI,T] = meyer(LB,UB,N)
[PHI,T] = meyer(LB,UB,N,'phi')
[PSI,T] = meyer(LB,UB,N,'psi')
```

**Description**

[PHI,PSI,T] = meyer(LB,UB,N) returns Meyer scaling and wavelet functions evaluated on an N point regular grid in the interval [LB,UB].

N must be a power of two.

Output arguments are the scaling function PHI and the wavelet function PSI computed on the grid T. These functions have [-8 8] as effective support.

If only one function is required, a fourth argument is allowed:

```
[PHI,T] = meyer(LB,UB,N,'phi')
[PSI,T] = meyer(LB,UB,N,'psi')
```

When the fourth argument is used, but not equal to 'phi' or 'psi', outputs are the same as in the main option.

The Meyer wavelet and scaling function are defined in the frequency domain:

- Wavelet function

$$\hat{\psi}(\omega) = (2\pi)^{-1/2} e^{i\omega/2} \sin\left(\frac{\pi}{2} v\left(\frac{3}{2\pi}|\omega| - 1\right)\right) \quad \text{if} \quad \frac{2\pi}{3} \leq |\omega| \leq \frac{4\pi}{3}$$

$$\hat{\psi}(\omega) = (2\pi)^{-1/2} e^{i\omega/2} \cos\left(\frac{\pi}{2} v\left(\frac{3}{4\pi}|\omega| - 1\right)\right) \quad \text{if} \quad \frac{4\pi}{3} \leq |\omega| \leq \frac{8\pi}{3}$$

$$\text{and } \hat{\psi}(\omega) = 0 \quad \text{if} \quad |\omega| \notin \left[\frac{2\pi}{3}, \frac{8\pi}{3}\right]$$

$$\text{where } v(a) = a^4(35 - 84a + 70a^2 - 20a^3), \quad a \in [0,1]$$

- Scaling function

$$\hat{\phi}(\omega) = (2\pi)^{-1/2} \quad \text{if} \quad |\omega| \leq \frac{2\pi}{3}$$

$$\hat{\phi}(\omega) = (2\pi)^{-1/2} \cos\left(\frac{\pi}{2} v\left(\frac{3}{2\pi}|\omega| - 1\right)\right) \quad \text{if} \quad \frac{2\pi}{3} \leq |\omega| \leq \frac{4\pi}{3}$$

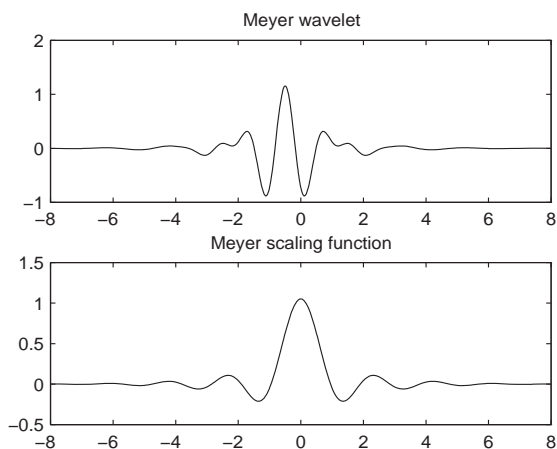
$$\hat{\phi}(\omega) = 0 \quad \text{if} \quad |\omega| > \frac{4\pi}{3}$$

By changing the auxiliary function (see `meyeraux` for more information), you get a family of different wavelets. For the required properties of the auxiliary function `v`, see “References” in Chapter 6, “Advanced Concepts”, of the User’s Guide.

## Examples

```
% Set effective support and grid parameters.
lb = -8; ub = 8; n = 1024;

% Compute and plot Meyer wavelet and scaling functions.
[phi,psi,x] = meyer(lb,ub,n);
subplot(211), plot(x,psi)
title('Meyer wavelet')
subplot(212), plot(x,phi)
title('Meyer scaling function')
```



# meyer

---

## Algorithm

Starting from an explicit form of the Fourier transform  $\hat{\phi}$  of  $\phi$ , `meyer` computes the values of  $\hat{\phi}$  on a regular grid, and then the values of  $\phi$  are computed using `instdfft`, the inverse nonstandard discrete FFT.

The procedure for  $\psi$  is along the same lines.

## References

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics, SIAM Ed., pp. 117–119, 137, 152.

## See Also

`meyeraux`, `wavefun`, `waveinfo`

**Purpose** Meyer wavelet auxiliary function

**Syntax** `Y = meyeraux(X)`

**Description** `Y = meyeraux(X)` returns values of the auxiliary function used for Meyer wavelet generation evaluated at the elements of the vector or matrix `X`.

The function is

$$35x^4 - 84x^5 + 70x^6 - 20x^7$$

**See Also** `meyer`

# morlet

---

**Purpose** Morlet wavelet

**Syntax** [PSI,X] = morlet(LB,UB,N)

**Description** [PSI,X] = morlet(LB,UB,N) returns values of the Morlet wavelet on an N point regular grid in the interval [LB,UB].

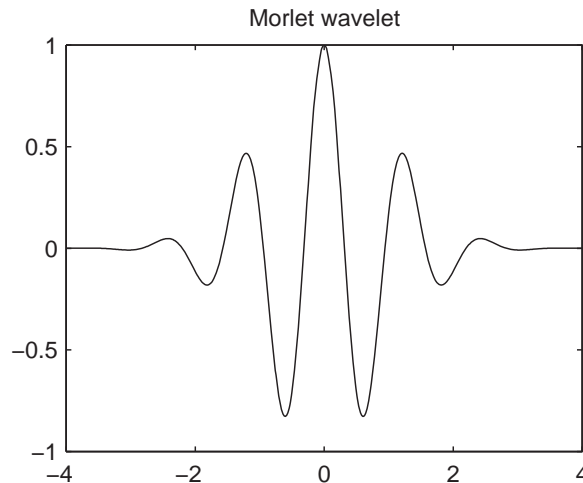
Output arguments are the wavelet function PSI computed on the grid X, and the grid X.

This wavelet has [-4 4] as effective support. Although [-4 4] is the correct theoretical effective support, a wider effective support, [-8 8], is used in the computation to provide more accurate results.

$$\psi(x) = e^{-x^2/2} \cos(5x)$$

## Examples

```
% Set effective support and grid parameters.  
lb = -4; ub = 4; n = 1000;  
% Compute and plot Morlet wavelet.  
[psi,x] = morlet(lb,ub,n);  
plot(x,psi), title('Morlet wavelet')
```



**See Also** waveinfo



**Purpose** Multisignal 1-D compression using wavelets

**Syntax**

```
[XC,DECCMP,THRESH] = mswcmp('cmp',DEC,METH)
[XC,DECCMP,THRESH] = mswcmp('cmp',DEC,METH,PARAM)
[XC,THRESH] = mswcmp('cmpsig',...)
[DECCMP,THRESH] = mswcmp('cmpdec',...)
THRESH = mswcmp('thr',...)
[...] = mswcmp(OPTION,DIRDEC,X,WNAME,LEV,METH)
[...] = mswcmp(OPTION,DIRDEC,X,WNAME,LEV,METH,PARAM)
[...] = mswcmp(...,S_OR_H)
[...] = mswcmp(...,S_OR_H,KEEPAPP)
[...] = mswcmp(...,S_OR_H,KEEPAPP,IDXSIG)
```

**Description** mswcmp computes thresholds and, depending on the selected option, performs compression of 1-D signals using wavelets.

[XC,DECCMP,THRESH] = mswcmp('cmp',DEC,METH) or [XC,DECCMP,THRESH] = mswcmp('cmp',DEC,METH,PARAM) returns a compressed (indicated by 'cmp' input) version XC of the original multisignal matrix X, whose wavelet decomposition structure is DEC. The output XC is obtained by thresholding the wavelet coefficients: DECCMP, which is the wavelet decomposition associated with XC (see mdwtdec), and THRESH is the matrix of threshold values. The input METH is the name of the compression method and PARAM is the associated parameter, if required.

Valid compression methods METH are shown in the following tables. For methods that use an associated parameter, the range of allowable PARAM values is also shown.

'rem_n0'	Remove near 0
'bal_sn'	Balance sparsity-norm
'sqrtbal_sn'	Balance sparsity-norm (sqrt)
'scarce'	Scarce, PARAM (any number)
'scarcehi'	Scarce high, $2.5 \leq \text{PARAM} \leq 10$
'scarceme'	Scarce medium, $1.5 \leq \text{PARAM} \leq 2.5$
'scarcelo'	Scarce low, $1 \leq \text{PARAM} \leq 2$

PARAM is a sparsity parameter, and it should be such that:  $1 \leq \text{PARAM} \leq 10$ . For scarce method no control is done.

'L2_perf'	Energy ratio
'NO_perf'	Zero coefficients ratio

PARAM is a real number which represents the required performance:

$$0 \leq \text{PARAM} \leq 100.$$

'g1b_thr'	Global threshold
-----------	------------------

PARAM is a real positive number.

'man_thr'	Manual method
-----------	---------------

PARAM is an NbSIG-by-NbLEV matrix or NbSIG-by-(NbLEV+1) matrix such that:

- -  $\text{PARAM}(i, j)$  is the threshold for the detail coefficients of level  $j$  for the  $i$ th signal ( $1 \leq j \leq \text{NbLEV}$ ).
- -  $\text{PARAM}(i, \text{NbLEV}+1)$  is the threshold for the approximation coefficients for the  $i$ th signal (if KEEPAPP is 0).

Where NbSIG is the number of signals and NbLEV the number of levels of decomposition.

`[XC, THRESH] = mswcmp('cmpsig', ...)` or  
`[DECCMP, THRESH] = mswcmp('cmpdec', ...)` or  
`THRESH = mswcmp('thr', ...)` Instead of the 'cmp' input OPTION, you can use 'cmpsig', 'cmpdec' or 'thr' to select other output arguments. 'thr' returns the computed thresholds, but compression is not performed.

`[...] = mswcmp(OPTION, DIRDEC, X, WNAME, LEV, METH)`  
`[...] = mswcmp(OPTION, DIRDEC, X, WNAME, LEV, METH, PARAM)` The decomposition structure input argument DEC can be replaced by four arguments: DIRDEC, X, WNAME, and LEV. Before performing a compression or computing thresholds, the multisignal matrix X is decomposed at level LEV using the wavelet WNAME, in the direction DIRDEC.

```
[...] = mswcmp(...,S_OR_H)
[...] = mswcmp(...,S_OR_H,KEEPAPP)
[...] = mswcmp(...,S_OR_H,KEEPAPP,IDXSIG) Three more optional inputs
may be used:
```

- S\_OR\_H ('s' or 'h') stands for soft or hard thresholding (see mswthresh for more details). Default is 'h'.
- KEEPAPP (true or false) indicates whether to keep approximation coefficients (true) or not (false). Default is false.
- IDXSIG is a vector which contains the indices of the initial signals, or the string 'all'. Default is 'all'.

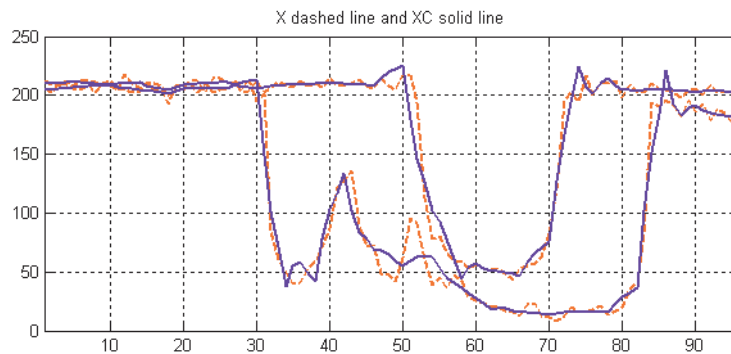
## Examples

```
% Load original 1D-multisignal.
load thinker

% Perform a decomposition at level 2 using wavelet db2.
dec = mdwtdec('r',X,2,'db2');

% Compress the signals to obtain a percentage of zeros
% near 95% for the wavelet coefficients.
[XC,decCMP,THRESH] = mswcmp('cmp',dec,'NO_perf',95);
[Ecmp,PECcmp,PECFScmp] = wdecenergy(decCMP);

% Plot the original signals 1 and 31, and
% the corresponding compressed signals.
figure;
plot(X([1 31],:),'r--','linewidth',2); hold on
plot(XC([1 31],:),'b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
title('X dashed line and XC solid line')
```



## References

- Birgé L.; P. Massart (1997), “From Model Selection to Adaptive Estimation,” in D. Pollard (ed), *Festschrift for L. Le Cam*, Springer, pp. 55–88.
- DeVore, R.A.; B. Jawerth, B.J. Lucier (1992), “Image Compression Through Wavelet Transform Coding,” *IEEE Trans. on Inf. Theory*, vol. 38, No 2, pp. 719–746.
- 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.
- Donoho, D.L.; I.M. Johnstone (1994), “Ideal Spatial Adaptation by Wavelet Shrinkage,” *Biometrika*, vol. 81, pp. 425–455.
- Donoho, D.L.; I.M. Johnstone, G. Kerkycharian, D. Picard (1995), “Wavelet Shrinkage: Asymptopia,” *Jour. Roy. Stat. Soc., series B*, vol. 57 no. 2, pp. 301–369.
- Donoho, D.L.; I.M. Johnstone, “Ideal De-noising in an Orthonormal Basis Chosen from a Library of Bases,” *C.R.A.S. Paris*, t. 319, Ser. I, pp. 1317–1322.
- Donoho, D.L. (1995), “De-noising by Soft-thresholding,” *IEEE Trans. on Inf. Theory*, 41, 3, pp. 613–627.

## See Also

mdwtdec, mdwtrec, mswthresh, wthresh

**Purpose** Multisignal 1-D wavelet compression scores

**Syntax** [THR, L2SCR, NOSCR, IDXSORT] = mswcmpscr(DEC)  
 [THR, L2SCR, NOSCR, IDXSORT] = mswcmpscr(DEC, S\_OR\_H, KEEPAPP, IDXSIG)

**Description** [THR, L2SCR, NOSCR, IDXSORT] = mswcmpscr(DEC) computes four matrices: thresholds THR, compression scores L2SCR and NOSCR, and indices IDXSORT. The decomposition DEC corresponds to a matrix of wavelet coefficients CFS obtained by concatenation of detail and (optionally) approximation coefficients, where

CFS = [cd{DEC.level}, ... , cd{1}] or  
 CFS = [ca, cd{DEC.level}, ... , cd{1}]

The concatenation is made rowwise if DEC.dirDec is equal to 'r' or columnwise if DEC.dirDec is equal to 'c' .

If NbSIG is the number of original signals and NbCFS the number of coefficients for each signal (all or only the detail coefficients), then CFS is an NbSIG-by-NbCFS matrix. Therefore,

- THR, L2SCR, NOSCR are NbSIG-by-(NbCFS+1) matrices
- IDXSORT is an NbSIG-by-NbCFS matrix
- THR(:, 2:end) is equal to CFS sorted by row in ascending order with respect to the absolute value.
- For each row, IDXSORT contains the order of coefficients and THR(:, 1)=0.

For the *i*th signal:

- L2SCR(*i*, *j*) is the percentage of preserved energy (L2-norm), corresponding to a threshold equal to CFS(*i*, *j*-1) ( $2 \leq j \leq \text{NbCFS}$ ), and L2SCR(:, 1)=100.
- NOSCR(*i*, *j*) is the percentage of zeros corresponding to a threshold equal to CFS(*i*, *j*-1) ( $2 \leq j \leq \text{NbCFS}$ ), and NOSCR(:, 1)=0.

Three more optional inputs may be used:

```
[...] = mswcmpscr(...,S_OR_H,KEEPAPP,IDXSIG)
```

- S\_OR\_H ('s' or 'h') stands for soft or hard thresholding (see mswthresh for more details).
- KEEPAPP (true or false) indicates whether to keep approximation coefficients (true) or not (false).
- IDXSIG is a vector that contains the indices of the initial signals, or the string 'all'.

The defaults are, respectively, 'h', false and 'all'.

## Examples

```
% Load original 1D-multisignal.
load thinker

% Perform a decomposition at level 2 using wavelet db2.
dec = mdwtdec('r',X,2,'db2');

% Compute compression performances for soft an hard thresholding.
[THR_S,L2SCR_S,NOSCR_S] = mswcmpscr(dec,'s');
[THR_H,L2SCR_H,NOSCR_H] = mswcmpscr(dec,'h');
```

## References

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

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.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

## See Also

mdwtdec, mdwtrec, ddencmp, wdencomp

**Purpose**

Multisignal 1-D compression thresholds and performances

**Syntax**

```
[THR_VAL,L2_Perf,NO_Perf] = mswcmptp(DEC,METH)
[THR_VAL,L2_Perf,NO_Perf] = mswcmptp(DEC,METH,PARAM)
[THR_VAL,L2_Perf,NO_Perf] = mswcmptp(...,S_OR_H)
[THR_VAL,L2_Perf,NO_Perf] = mswcmptp(...,S_OR_H,KEEPAPP)
[THR_VAL,L2_Perf,NO_Perf] = mswcmptp(...,S_OR_H,KEEPAPP,IDXSIG)
```

**Description**

[THR\_VAL,L2\_Perf,NO\_Perf] = mswcmptp(DEC,METH) or  
 [THR\_VAL,L2\_Perf,NO\_Perf] = mswcmptp(DEC,METH,PARAM) computes the  
 vectors THR\_VAL, L2\_Perf and NO\_Perf obtained after a compression using the  
 METH method and, if required, the PARAM parameter (see mswcmp for more  
 information on METH and PARAM).

For the  $i$ th signal:

- THR\_VAL( $i$ ) is the threshold applied to the wavelet coefficients. For a level dependent method, THR\_VAL( $i, j$ ) is the threshold applied to the detail coefficients at level  $j$ .
- L2\_Perf( $i$ ) is the percentage of energy (L2\_norm) preserved after compression.
- NO\_Perf( $i$ ) is the percentage of zeros obtained after compression.

You can use three more optional inputs:

```
[...] = mswcmptp(...,S_OR_H,KEEPAPP,IDXSIG)
```

- S\_OR\_H ('s' or 'h') stands for soft or hard thresholding (see mswthresh for more details).
- KEEPAPP (true or false) indicates whether to keep approximation coefficients (true) or not (false)
- IDXSIG is a vector which contains the indices of the initial signals, or the string 'all'.

The defaults are, respectively, 'h', false and 'all'.

**Examples**

```
% Load original 1D-multisignal.
load thinker

% Perform a decomposition at level 2 using wavelet db2.
```

```
dec = mdwtdec('r',X,2,'db2');

% Compute compression thresholds and exact performances
% obtained after a compression using the method 'NO_perf' and
% requiring a percentage of zeros near 95% for the wavelet
% coefficients.
[THR_VAL,L2_Perf,NO_Perf] = mswcmptp(dec,'NO_perf',95);
```

## References

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

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.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

## See Also

mdwtdec, mdwtrec, ddencmp, wdencomp



**Purpose** Multisignal 1-D denoising using wavelets

**Syntax**

```
[XD,DEC DEN,THRESH] = mswden('den',DEC,METH)
[XD,DEC DEN,THRESH] = mswden('den',DEC,METH,PARAM)
[XD,THRESH] = mswden('densig',...)
[DEC DEN,THRESH] = mswden('dendec',...)
THRESH = mswden('thr',...)
[...] = mswden(OPTION,DIRDEC,X,WNAME,LEV,METH,PARAM)
[...] = mswden(...,S_OR_H)
[...] = mswden(...,S_OR_H,KEEPAPP)
[...] = mswden(...,S_OR_H,KEEPAPP,IDX SIG)
```

**Description** mswden computes thresholds and, depending on the selected option, performs denoising of 1-D signals using wavelets.

[XD,DEC DEN,THRESH] = mswden('den',...) returns a denoised version XD of the original multisignal matrix X, whose wavelet decomposition structure is DEC. The output XD is obtained by thresholding the wavelet coefficients, DEC DEN is the wavelet decomposition associated to XD (see mdwtdec), and THRESH is the matrix of threshold values. The input METH is the name of the denoising method and PARAM is the associated parameter, if required.

Valid denoising methods METH and associated parameters PARAM are:

'rigrsure'	Principle of Stein's Unbiased Risk
'heursure'	Heuristic variant of the first option
'sqtwolog'	Universal threshold $\sqrt{2 \cdot \log(\cdot)}$
'minimaxi'	Minimax thresholding (see thselect)

For these methods PARAM defines the multiplicative threshold rescaling:

'one'	No rescaling
'sln'	Rescaling using a single estimation of level noise based on first level coefficients
'mln'	Rescaling using a level dependent estimation of level noise

## Penalization methods

'penal'	Penal
'penalhi'	Penal high, $2.5 \leq \text{PARAM} \leq 10$
'penalme'	Penal medium, $1.5 \leq \text{PARAM} \leq 2.5$
'penallo'	Penal low, $1 \leq \text{PARAM} \leq 2$

PARAM is a sparsity parameter, and it should be such that:  $1 \leq \text{PARAM} \leq 10$ . For penal method, no control is done.

## Manual method

'man_thr'	Manual method
-----------	---------------

PARAM is an NbSIG-by-NbLEV matrix or NbSIG-by-(NbLEV+1) matrix such that:

- $\text{PARAM}(i, j)$  is the threshold for the detail coefficients of level  $j$  for the  $i$ th signal ( $1 \leq j \leq \text{NbLEV}$ ).
- $\text{PARAM}(i, \text{NbLEV}+1)$  is the threshold for the approximation coefficients for the  $i$ th signal (if KEEPAPP is 0).

where NbSIG is the number of signals and NbLEV the number of levels of decomposition.

Instead of the 'den' input OPTION, you can use 'densig', 'dendec' or 'thr' OPTION to select output arguments:

```
[XD,THRESH] = mswden('densig',...) or
[DECDEN,THRESH] = mswden('dendec',...)
```

THRESH = mswden('thr',...) returns the computed thresholds, but denoising is not performed.

The decomposition structure input argument DEC can be replaced by four arguments: DIRDEC, X, WNAME and LEV.

[...] = mswden(OPTION,DIRDEC,X,WNAME,LEV,METH,PARAM) before performing a denoising or computing thresholds, the multisignal matrix X is decomposed at level LEV using the wavelet WNAME, in the direction DIRDEC.

You can use three more optional inputs:

```
[...] = mswden(...,S_OR_H) or
[...] = mswden(...,S_OR_H,KEEPAPP) or
[...] = mswden(...,S_OR_H,KEEPAPP,IDXSIG)
```

- S\_OR\_H ('s' or 'h') stands for soft or hard thresholding (see mswthresh for more details).
- KEEPAPP (true or false) indicates whether to keep approximation coefficients (true) or not (false).
- IDXSIG is a vector that contains the indices of the initial signals, or the string 'all'.

The defaults are, respectively, 'h', false and 'all'.

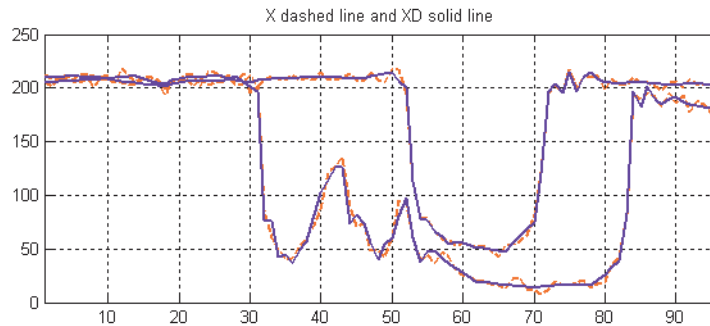
## Examples

```
% Load original 1D-multisignal.
load thinker

% Perform a decomposition at level 2 using the wavelet db2.
dec = mdwtdec('r',X,2,'db2');

% Denoise signals using the universal method
% of thresholding (sqrtwolog) and the 'sln'
% threshold rescaling (with a single estimation
% of level noise, based on first level coefficients).
[XD,decDEN,THRESH] = mswden('den',dec,'sqrtwolog','sln');

% Plot the original signals 1 and 31, and the
% corresponding denoised signals.
figure;
plot(X([1 31],:),'r--','linewidth',2); hold on
plot(XD([1 31],:),'b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
title('X dashed line and XD solid line')
```



## References

Birgé, L.; P. Massart (1997), “From model selection to adaptive estimation,” in D. Pollard (ed), *Festschrift for L. Le Cam*, Springer, pp. 55–88.

DeVore, R.A.; B. Jawerth, B.J. Lucier (1992), “Image compression through wavelet transform coding,” *IEEE Trans. on Inf. Theory*, vol. 38, No 2, pp. 719–746.

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.

Donoho, D.L.; I.M. Johnstone (1994), “Ideal spatial adaptation by wavelet shrinkage,” *Biometrika*, vol. 81, pp. 425–455.

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.

Donoho, D.L.; I.M. Johnstone, “Ideal de-noising in an orthonormal basis chosen from a library of bases,” *C.R.A.S. Paris*, t. 319, Ser. I, pp. 1317–1322.

Donoho, D.L. (1995), “De-noising by soft-thresholding,” *IEEE Trans. on Inf. Theory*, 41, 3, pp. 613–627.

## See Also

mdwtdec, mdwtrec, mswthresh, wthresh

**Purpose** Perform multisignal 1-D thresholding

**Syntax**

```
Y = msthresh(X,SORH,T)
Y = msthresh(X,SORH,T,'c')
Y = msthresh(X,'s',T)
Y = msthresh(X,'h',T)
```

**Description** `Y = msthresh(X,SORH,T)` returns soft (if `SORH='s'`) or hard (if `SORH='h'`) `T`-thresholding of the input matrix `X`. `T` can be a single value, a matrix of the same size as `X` or a vector. In this last case, thresholding is performed rowwise and `LT = length(T)` must be such that `size(X,1) ≤ LT`.

`Y = msthresh(X,SORH,T,'c')` performs a columnwise thresholding and `size(X,2) ≤ LT`.

`Y = msthresh(X,'s',T)` returns  $Y = \text{SIGN}(X) \cdot (|X| - T)_+$ , soft thresholding is shrinkage.

`Y = msthresh(X,'h',T)` returns  $Y = X \cdot 1_{(|X| > T)}$ , hard thresholding is cruder.

**See Also** `mstden`, `mstdcmp`, `wthresh`, `wden`, `wdencomp`, `wpdencomp`

# nodeasc

**Purpose** Node ascendants

**Syntax**  
`A = nodeasc(T,N)`  
`A = nodeasc(T,N, 'deppos')`

**Description** nodeasc is a tree-management utility.

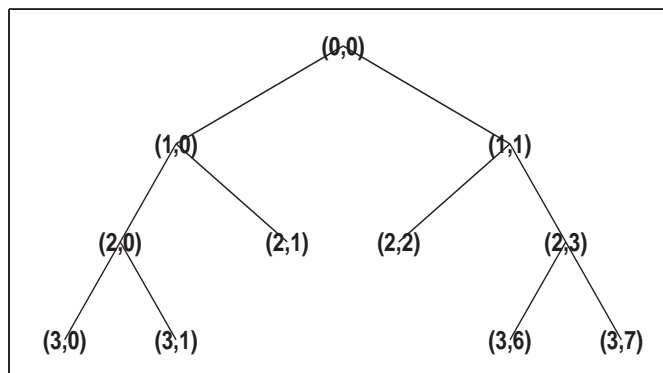
`A = nodeasc(T,N)` returns the indices of all the ascendants of the node `N` in the tree `T` where `N` can be the index node or the depth and position of the node. `A` is a column vector with `A(1) = index of node N`.

`A = nodeasc(T,N, 'deppos')` is a matrix, which contains the depths and positions of all ascendants. `A(i,1)` is the depth of the `i`-th ascendant and `A(i,2)` is the position of the `i`-th ascendant.

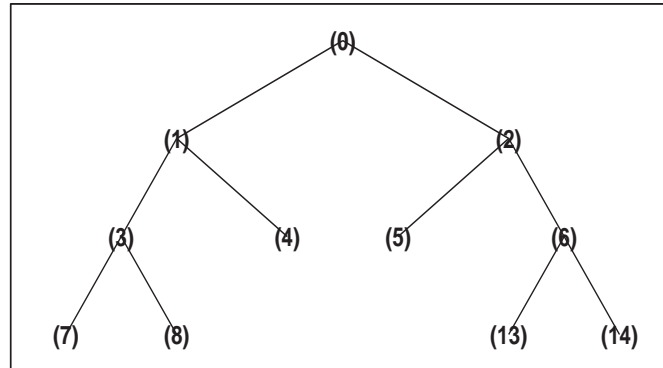
The nodes are numbered from left to right and from top to bottom. The root index is 0.

**Examples**

```
% Create binary tree of depth 3.  
t = ntree(2,3);  
t = nodejoin(t,5);  
t = nodejoin(t,4);  
plot(t)
```



```
% Change Node Label from Depth_Position to Index
% (see the plot function).
```



```
nodeasc(t,[2 2])
```

```
ans =
```

```
5
```

```
2
```

```
0
```

```
nodeasc(t,[2 2], 'deppos')
```

```
ans =
```

```
2    2
```

```
1    1
```

```
0    0
```

## See Also

nodedesc, nodepar, wtreesmgr

# nodedesc

---

**Purpose** Node descendants

**Syntax**  
`D = nodedesc(T,N)`  
`D = nodedesc(T,N, 'deppos')`

**Description** `nodedesc` is a tree-management utility.

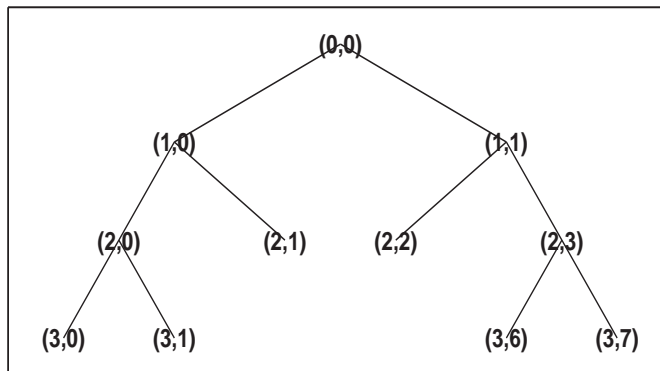
`D = nodedesc(T,N)` returns the indices of all the descendants of the node `N` in the tree `T` where `N` can be the index node or the depth and position of node. `D` is a column vector with `D(1) = index of node N`.

`D = nodedesc(T,N, 'deppos')` is a matrix that contains the depths and positions of all descendants. `D(i,1)` is the depth of the *i*-th descendant and `D(i,2)` is the position of the *i*-th descendant.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

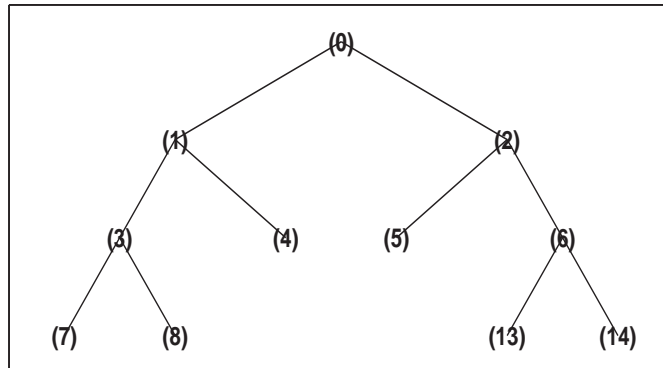
**Examples**

```
% Create binary tree of depth 3.  
t = ntree(2,3);  
t = nodejoin(t,5);  
t = nodejoin(t,4);  
plot(t)
```





```
% Change Node Label from Depth_Position to Index
% (see the plot function).
```



```
% Node descendants.
```

```
nodedesc(t,2)
```

```
ans =
     2
     5
     6
    13
    14
```

```
nodedesc(t,2,'deppos')
```

```
ans =
     1     1
     2     2
     2     3
     3     6
     3     7
```

```
nodedesc(t,[1 1],'deppos')
```

```
ans =
     1     1
     2     2
     2     3
     3     6
     3     7
```

# nodedesc

---

```
nodedesc(t,[1 1])
ans =
     2
     5
     6
    13
    14
```

**See Also** [nodeasc](#), [nodepar](#), [wtreemgr](#)

**Purpose** Recompose node

**Syntax** `T = nodejoin(T,N)`  
`T = nodejoin(T)`

**Description** `nodejoin` is a tree-management utility.

`T = nodejoin(T,N)` returns the modified tree `T` corresponding to a recomposition of the node `N`.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

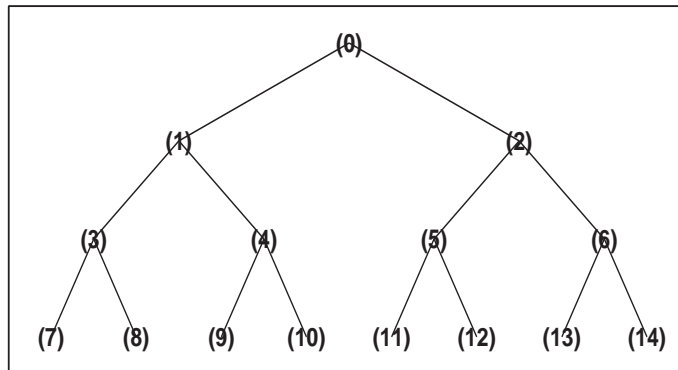
`T = nodejoin(T)` is equivalent to `T = nodejoin(T,0)`.

### Examples

```
% Create binary tree of depth 3.
t = ntree(2,3);

% Plot tree t.
plot(t)

% Change Node Label from Depth_Position to Index
% (see the plot function).
```



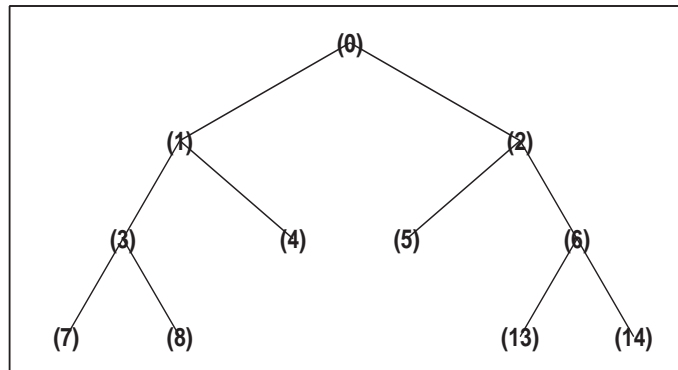
```
% Merge nodes of indices 4 and 5.
t = nodejoin(t,5);
t = nodejoin(t,4);
```

# nodejoin

---

```
% Plot new tree t.  
plot(t)
```

```
% Change Node Label from Depth_Position to Index  
% (see the plot function).
```



## See Also

`nodesplit`

**Purpose** Node parent

**Syntax**  
`F = nodepar(T,N)`  
`F = nodepar(T,N, 'deppos')`

**Description** nodepar is a tree-management utility.

`F = nodepar(T,N)` returns the indices of the 'parent(s)' of the nodes `N` in the tree `T` where `N` can be a column vector containing the indices of nodes or a matrix that contains the depths and positions of nodes. In the last case, `N(i,1)` is the depth of the `i`-th node and `N(i,2)` is the position of the `i`-th node.

`F = nodepar(T,N, 'deppos')` is a matrix that contains the depths and positions of returned nodes. `F(i,1)` is the depth of the `i`-th node and `F(i,2)` is the position of the `i`-th node.

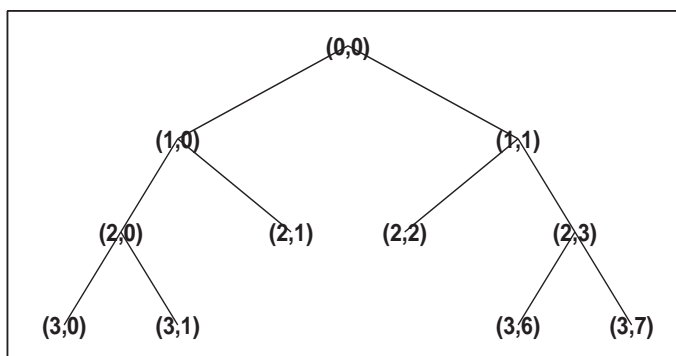
`nodepar(T,0)` or `nodepar(T,[0,0])` returns `-1`.

`nodepar(T,0, 'deppos')` or `nodepar(T,[0,0], 'deppos')` returns `[-1,0]`.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

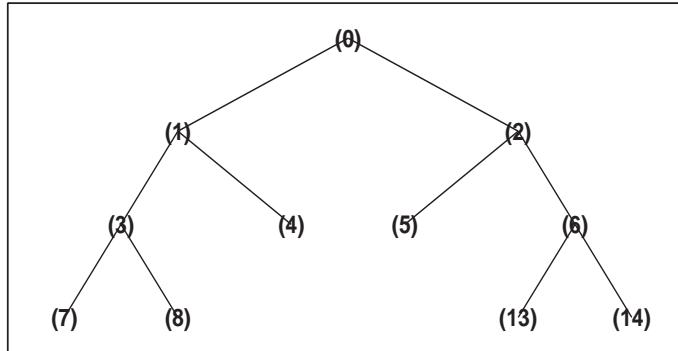
## Examples

```
% Create binary tree of depth 3.
t = ntree(2,3);
t = nodejoin(t,5);
t = nodejoin(t,4);
plot(t)
```



# nodepar

```
% Change Node Label from Depth_Position to Index  
% (see the plot function).
```



```
% Nodes parent.  
nodepar(t,[2 2], 'depos')
```

```
ans =  
    1    1
```

```
nodepar(t,[1;7;14])
```

```
ans =  
    0  
    3  
    6
```

## See Also

nodeasc, nodedesc, wtremgr

**Purpose** Split (decompose) node

**Syntax** `T = nodesplt(T,N)`

**Description** `nodesplt` is a tree-management utility.

`T = nodesplt(T,N)` returns the modified tree `T` corresponding to the decomposition of the node `N`.

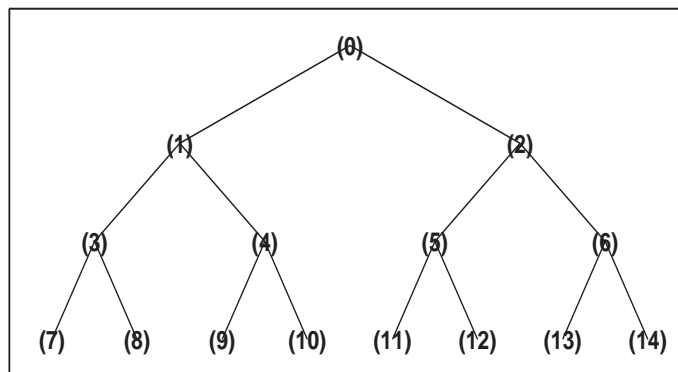
The nodes are numbered from left to right and from top to bottom. The root index is 0.

## Examples

```
% Create binary tree (tree of order 2) of depth 3.
t = ntree(2,3);
```

```
% Plot tree t.
plot(t)
```

```
% Change Node Label from Depth_Position to Index
% (see the plot function).
```

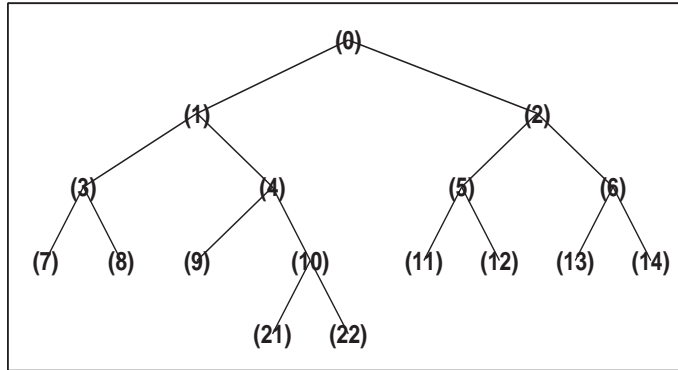


```
% Split node of index 10.
t = nodesplt(t,10);
```

# nodesplt

---

```
% Plot new tree t.  
plot(t)  
  
% Change Node Label from Depth_Position to Index  
% (see the plot function).
```



## See Also

nodejoin



**Purpose** Determine nonterminal nodes

**Syntax**  
`N = noleaves(T)`  
`N = noleaves(T, 'dp')`

**Description** `N = noleaves(T)` returns the indices of nonterminal nodes of the tree `T` (i.e., nodes that are not leaves). `N` is a column vector.

The nodes are ordered from left to right as in tree `T`.

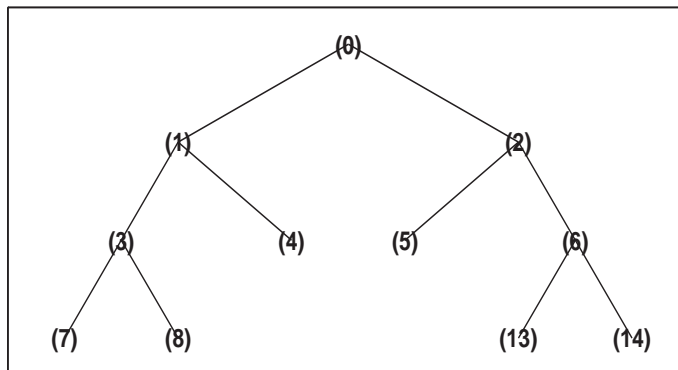
`N = noleaves(T, 'dp')` returns a matrix `N`, which contains the depths and positions of nonterminal nodes.

`N(i,1)` is the depth of the `i`-th nonterminal node and  
`N(i,2)` is the position of the `i`-th nonterminal node.

**Examples**

```
% Create initial tree.
ord = 2;
t = ntree(ord,3);           % binary tree of depth 3.
t=nodejoin(t,5);
t=nodejoin(t,4);
plot(t)

% Change Node Label from Depth_Position to Index
% (see the plot function).
```



# noleaves

---

```
% List nonterminal nodes (index).
ntnodes_ind = noleaves(t)

ntnodes_ind =
    0
    1
    2
    3
    6

% List nonterminal nodes (Depth_Position).
ntnodes_depo = noleaves(t,'dp')

ntnodes_depo =
    0     0
    1     0
    1     1
    2     0
    2     3
```

## See Also

leaves

**Purpose** Number of terminal nodes

**Syntax** NB = nnode(T)

**Description** nnode is a tree-management utility.

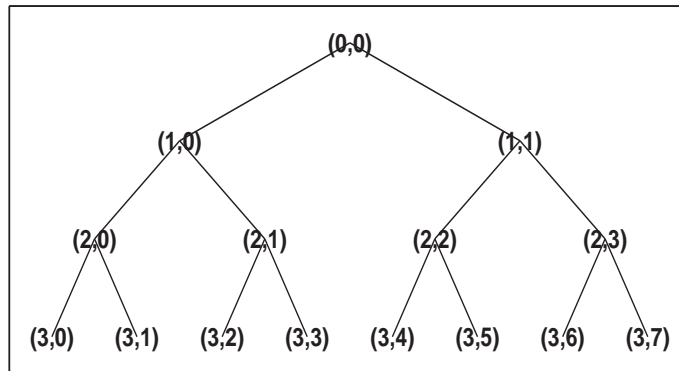
NB = nnode(T) returns the number of terminal nodes in the tree T.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

**Examples**

```
% Create binary tree (tree of order 2) of depth 3.
t = ntree(2,3);

% Plot tree t.
plot(t)
```



```
% Number of terminal nodes.
nnode(t)
```

```
ans =
     8
```

**See Also**

wtreemgr

# ntree

---

## Purpose

NTREE constructor

## Syntax

```
T = ntree(ORD,D)
T = ntree
T = ntree(ORD)
T = ntree(ORD,D,S,U)
T = ntree('PropName1',PropValue1,'PropName2',PropValue2, ...)
```

## Description

`T = ntree(ORD,D)` returns an NTREE object, which is a complete tree of order ORD and depth D.

`T = ntree` is equivalent to `T = ntree(2,0)`.

`T = ntree(ORD)` is equivalent to `T = ntree(ORD,0)`.

With `T = ntree(ORD,D,S)` you can set a “split scheme” for nodes. The split scheme field `S` is a logical array of size ORD by 1.

The root of the tree can be split and it has ORD children. You can split the *j*-th child if `S(j) = 1`.

Each node that you can split has the same property as the root node.

With `T = ntree(ORD,D,S,U)` you can, in addition, set a userdata field.

Inputs can be given in another way:

`T = ntree('order',ORD,'depth',D,'spsch',S,'ud',U)`. For “missing” inputs the defaults are `ORD = 2`, `D = 0`, `S = ones([1:ORD])`, `U = {}`.

`[T,NB] = ntree( ... )` returns also the number of terminal nodes (leaves) of `T`.

For more information on object fields, type `help ntree/get`.

Class NTREE (Parent class: WTBO)

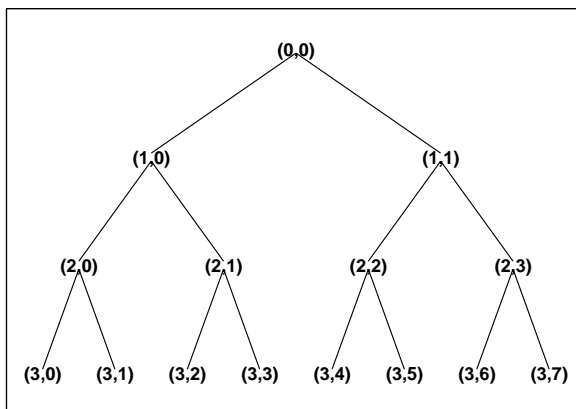
## Fields

wtbo	Parent object
order	Tree order
depth	Tree depth
spsch	Split scheme for nodes
tn	Column vector with terminal node indices

## Examples

```
% Create binary tree (tree of order 2) of depth 3.
t2 = ntree(2,3);
```

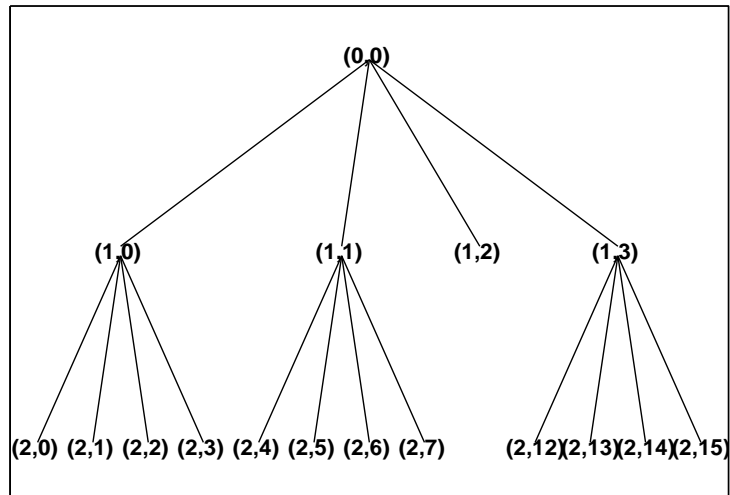
```
% Plot tree t2.
plot(t2)
```



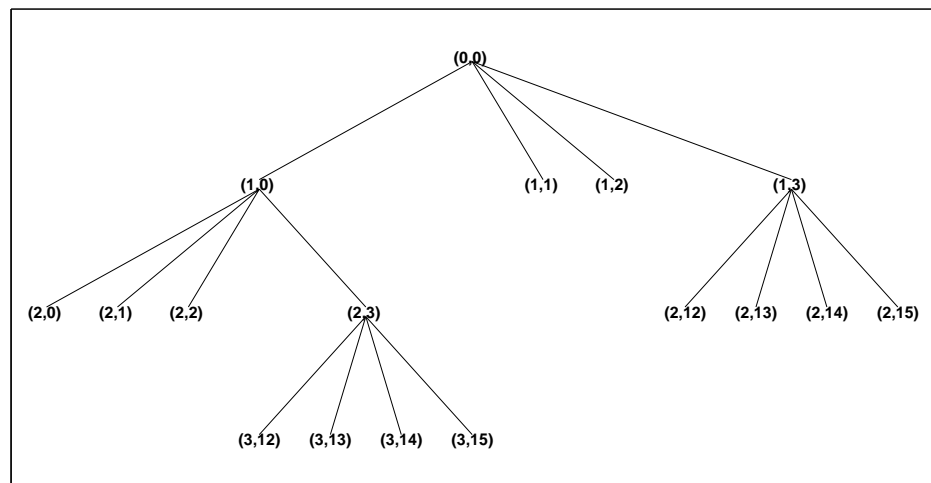
```
% Create a quadtree (tree of order 4) of depth 2.
t4 = ntree(4,2,[1 1 0 1]);
```

```
% Plot tree t4.
plot(t4)
```

# ntree



```
% Split and merge some nodes using the gui  
% generated by plot (see the plot function).  
% The figure becomes:
```



**See Also**

wtbo

**Purpose** Orthogonal wavelet filter set

**Syntax** [Lo\_D,Hi\_D,Lo\_R,Hi\_R] = orthfilt(W)

**Description** [Lo\_D,Hi\_D,Lo\_R,Hi\_R] = orthfilt(W) computes the four filters associated with the scaling filter W corresponding to a wavelet:

Lo\_D      Decomposition low-pass filter  
 Hi\_D      Decomposition high-pass filter  
 Lo\_R      Reconstruction low-pass filter  
 Hi\_R      Reconstruction high-pass filter

For an orthogonal wavelet, in the multiresolution framework, we start with the scaling function  $\phi$  and the wavelet function  $\psi$ . One of the fundamental relations is the twin-scale relation:

$$\frac{1}{2}\phi\left(\frac{x}{2}\right) = \sum_{n \in \mathbb{Z}} w_n \phi(x-n)$$

All the filters used in `dwt` and `idwt` are intimately related to the sequence  $(w_n)_{n \in \mathbb{Z}}$ . Clearly if  $\phi$  is compactly supported, the sequence  $(w_n)$  is finite and can be viewed as a FIR filter. The scaling filter W is

- A low-pass FIR filter
- Of length  $2N$
- Of sum 1
- Of norm  $\frac{1}{\sqrt{2}}$

For example, for the db3 scaling filter,

```
load db3
db3
db3 =
    0.2352  0.5706  0.3252 -0.0955 -0.0604  0.0249

sum(db3)
ans =
```

```

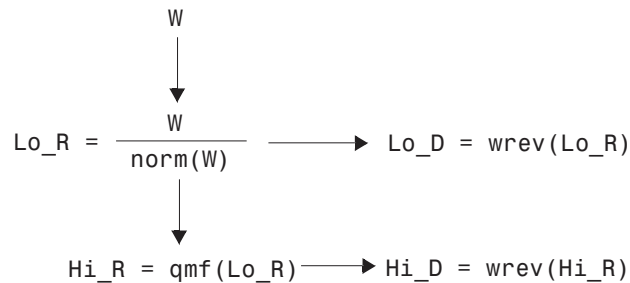
1.000
norm(db3)
ans =
0.7071

```

From filter  $W$ , we define four FIR filters, of length  $2N$  and norm 1, organized as follows:

Filters	Low-Pass	High-Pass
Decomposition	Lo_D	Hi_D
Reconstruction	Lo_R	Hi_R

The four filters are computed using the following scheme:



where  $\text{qmf}$  is such that  $\text{Hi}_R$  and  $\text{Lo}_R$  are quadrature mirror filters (i.e.,  $\text{Hi}_R(k) = (-1)^k \text{Lo}_R(2N + 1 - k)$ , for  $k = 1, 2, \dots, 2N$ ), and where  $\text{wrev}$  flips the filter coefficients. So  $\text{Hi}_D$  and  $\text{Lo}_D$  are also quadrature mirror filters. The computation of these filters is performed using `orthfilt`.

## Examples

```

% Load scaling filter.
load db8; w = db8;
subplot(421); stem(w);
title('Original scaling filter');

% Compute the four filters.
[Lo_D,Hi_D,Lo_R,Hi_R] = orthfilt(w);
subplot(423); stem(Lo_D);
title('Decomposition low-pass filter');

```



```
subplot(424); stem(Hi_D);
title('Decomposition high-pass filter');
subplot(425); stem(Lo_R);
title('Reconstruction low-pass filter');
subplot(426); stem(Hi_R);
title('Reconstruction high-pass filter');

% Check for orthonormality.
df = [Lo_D;Hi_D];
rf = [Lo_R;Hi_R];
id = df*df'

id =
    1.0000    0
         0    1.0000

id = rf*rf'

id =
    1.0000    0
         0    1.0000

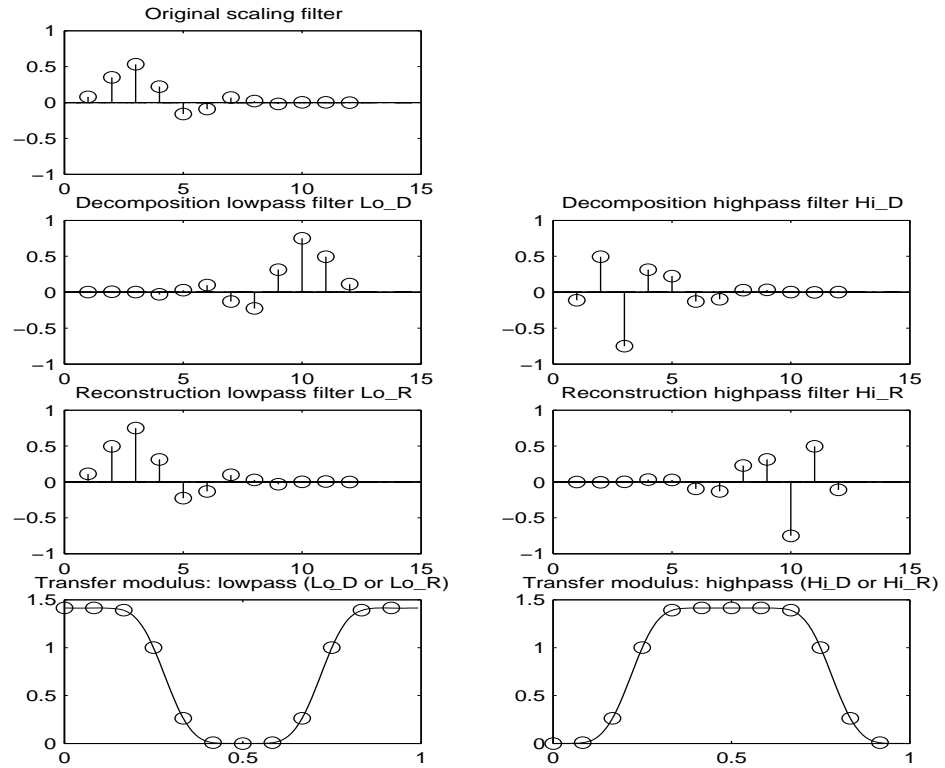
% Check for orthogonality by dyadic translation, for example:
df = [Lo_D 0 0;Hi_D 0 0];
dft = [0 0 Lo_D; 0 0 Hi_D];
zer = df*dft'

zer =

    1.0e-12 *
    -0.1883  0.0000
    -0.0000 -0.1883

% High- and low-frequency illustration.
fftlD = fft(Lo_D); ffthD = fft(Hi_D);
freq = [1:length(Lo_D)]/length(Lo_D);
subplot(427); plot(freq,abs(fftlD));
title('Transfer modulus: low-pass');
subplot(428); plot(freq,abs(ffthD));
title('Transfer modulus: high-pass')
```

```
% Editing some graphical properties,  
% the following figure is generated.
```



## References

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics, SIAM Ed. pp. 117–119, 137, 152.

## See Also

biorfilt, qmf, wfilters

- Purpose** Build wavelet from pattern
- Syntax** [PSI,XVAL,NC] = pat2cwav(YPAT,METHOD,POLDEGREE,REGULARITY)
- Description** [PSI,XVAL,NC] = pat2cwav(YPAT,METHOD,POLDEGREE,REGULARITY) computes an admissible wavelet for CWT (given by XVAL and PSI) adapted to the pattern defined by the vector YPAT, and of norm equal to 1.

The underlying x-values pattern is set to

```
xpat = linspace(0,1,length(YPAT))
```

The constant NC is such that NC\*PSI approximates YPAT on the interval [0,1] by least squares fitting using

- a polynomial of degree POLDEGREE when METHOD is equal to 'polynomial'
- a projection on the space of functions orthogonal to constants when METHOD is equal to 'othconst'

The REGULARITY parameter defines the boundary constraints at the points 0 and 1. Allowable values are 'continuous', 'differentiable', and 'none'.

When METHOD is equal to 'polynomial'

- if REGULARITY is equal to 'continuous', POLDEGREE must be  $\mathfrak{N} \geq 3$ .
- if REGULARITY is equal to 'differentiable', POLDEGREE must be  $\mathfrak{N} \geq 5$ .

- Examples** The principle for designing a new wavelet for CWT is to approximate a given pattern using least squares optimization under constraints leading to an admissible wavelet well suited for the pattern detection using the continuous wavelet transform (see Misiti et al.).

```
% Example: Generate a new wavelet starting from a pattern.
```

```
% Load original pattern: a pseudo sine one.
load ptpssin1;
```

```
% Variables X and Y contain the pattern.
whos
```

Name	Size	Bytes	Class
------	------	-------	-------

```
IntVAL      1x1          8 double array
X           1x256       2048 double array
Y           1x256       2048 double array
caption     1x35        70 char array
```

Grand total is 548 elements using 4174 bytes

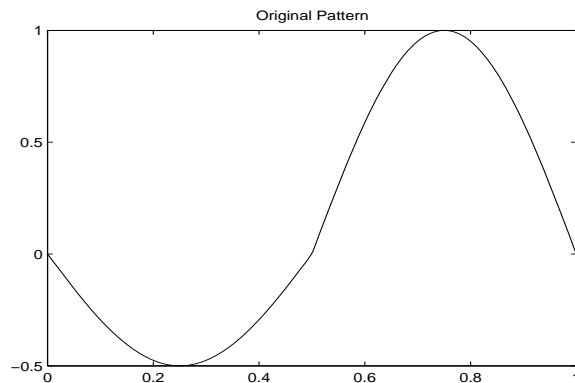
```
% This example is a demo-example, so we have the value of the
% integral of the pattern as well as the details about its
% construction in the caption variable.
```

```
IntVAL
```

```
IntVAL =
```

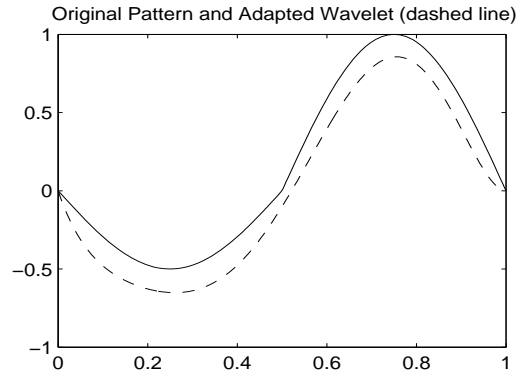
```
0.1592
```

```
% The pattern defined on the interval [0,1] is of integral 0.1592.
% So it is not a wavelet but it is a good candidate since it
% oscillates like a wavelet.
plot(X,Y), title('Original Pattern')
```



```
% To synthesize a new wavelet adapted to the given pattern, let
% us use a least squares polynomial approximation of degree 6 with
% constraints of continuity at the beginning and the end of the
% pattern.
```

```
[psi,xval,nc] = pat2cwav(Y, 'polynomial',6, 'continuous') ;  
  
% The new wavelet is given by xval and nc*psi.  
plot(X,Y,'-',xval,nc*psi,'--'),  
title('Original Pattern and Adapted Wavelet (dashed line)')
```



```
% Note that the version of the wavelet is correctly  
% defined in order to be used in the CWT algorithm must be of  
% square norm equal to 1. It is simply given by xval and psi.
```

## References

Misiti, M.; Y. Misiti, G. Oppenheim, J.-M. Poggi (2003), "Les ondelettes et leurs applications," Hermes.

# plot

---

**Purpose** Plot tree GUI

**Syntax**

```
plot(T)
plot(T,FIG)
FIG = plot(T)
NEWT = plot(T,'read',FIG)
NEWT = plot(DUMMY,'read',FIG)
```

**Description** plot is a graphical tree-management utility.

plot(T) plots the tree T.

The figure that contains the tree is a GUI tool. It lets you change the **Node Label** to **Depth\_Position** or **Index**, and **Node Action** to **Split-Merge** or **Visualize**.

The default values are **Depth\_Position** and **Visualize**.

You can click the nodes to execute the current **Node Action**.

plot(T,FIG) plots the tree T in the figure whose handle is FIG. This figure was already used to plot a tree, for example using the command

```
FIG = plot(T)
```

After some split or merge actions, you can get the new tree using its parent figure handle. The following syntax lets you perform this functionality:

```
NEWT = plot(T,'read',FIG)
```

In fact, the first argument is dummy. The most general syntax is

```
NEWT = plot(DUMMY,'read',FIG)
```

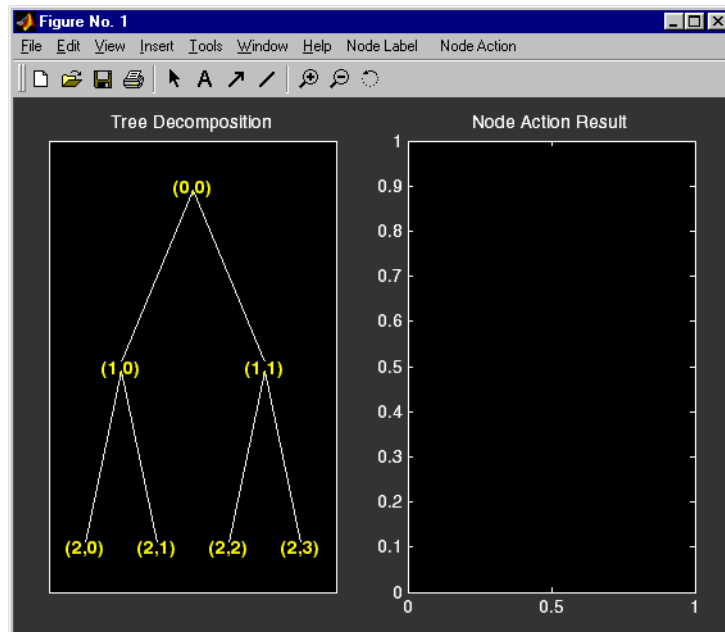
where DUMMY is any object parented by an NTREE object. More generally, DUMMY can be any object constructor name returning an NTREE parented object. For example:

```
NEWT = plot(ntree,'read',FIG)
NEWT = plot(dtree,'read',FIG)
NEWT = plot(wptree,'read',FIG)
```

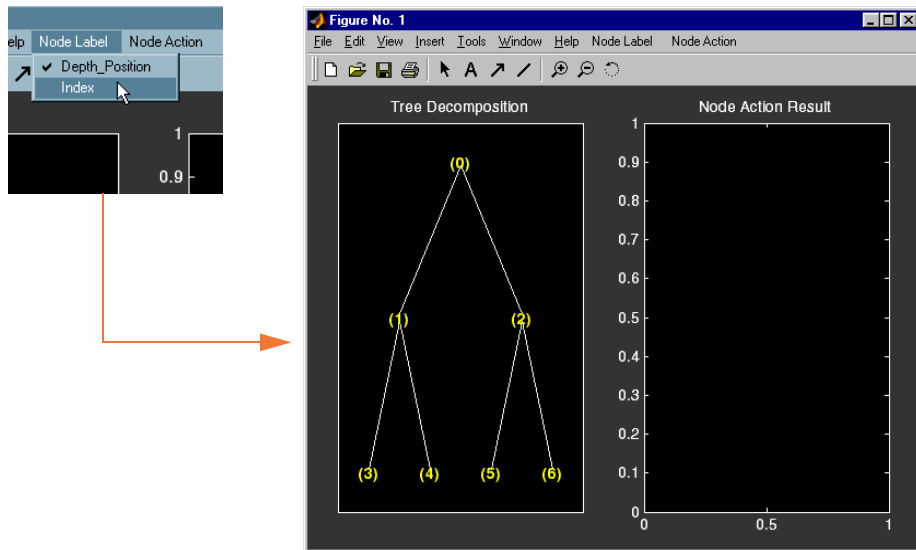
## Examples

```
% Create a wavelet packets tree (1-D)
load noisbloc
x = noisbloc;
t = wpdec(x,2,'db2');

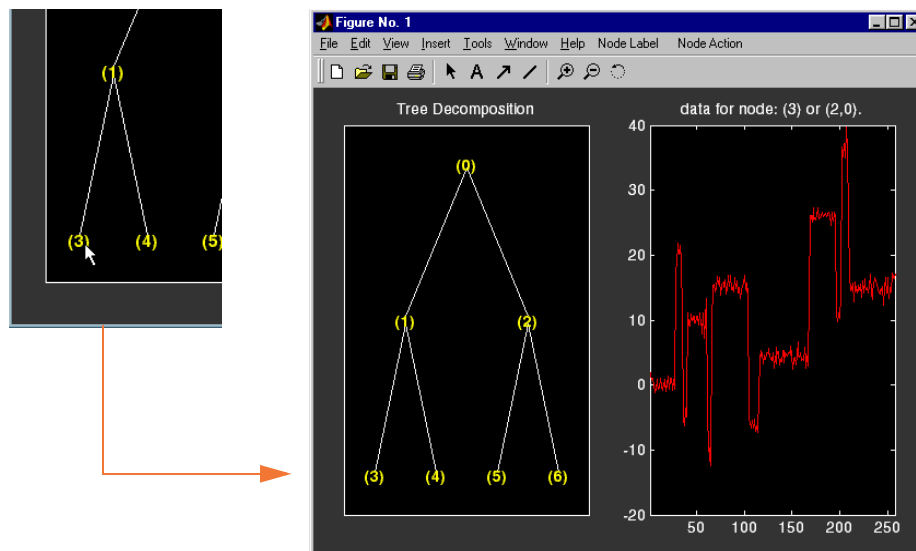
% Plot tree t.
plot(t)
```



```
% Change Node Label from Depth_Position to Index.
```

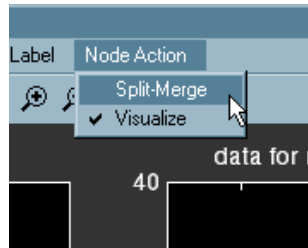


% Click the node (3). You get the following figure.

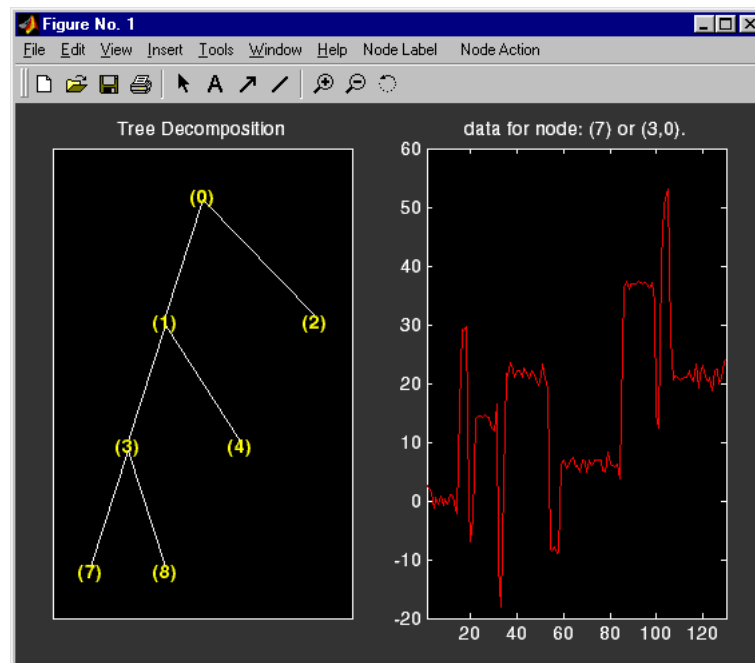


% Change Node Action from Visualize to Split\_Merge.





```
% Merge the node (2) and split the node (3).
% Change Node Action from Split_Merge to Visualize.
% Click the node (7). You obtain the following figure,
% which represents the wavelet decomposition at level 3.
```

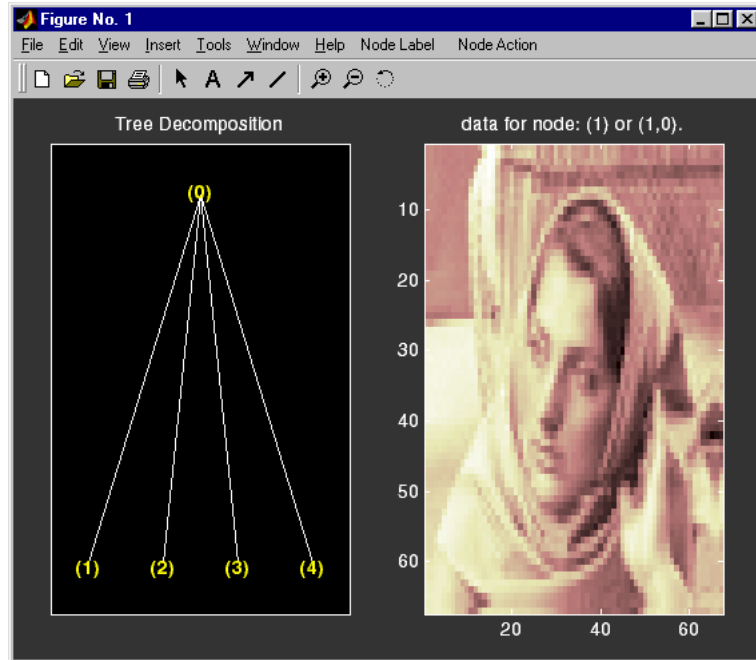


```
% Create a wavelet packets tree (2-D)
load woman2
t = wpdec2(X,1,'sym4');
```

# plot

```
% Plot tree t.  
plot(t)
```

```
% Change Node Label from Depth_Position to Index.  
% Click the node (1). You get the following figure.
```



**Purpose** Quadrature mirror filter

**Syntax**  
 $Y = \text{qmf}(X, P)$   
 $Y = \text{qmf}(X)$

**Description**  $Y = \text{qmf}(X, P)$  changes the signs of the even index entries of the reversed vector filter coefficients  $X$  if  $P$  is even. If  $P$  is odd the same holds for odd index entries.  $Y = \text{qmf}(X)$  is equivalent to  $Y = \text{qmf}(X, 0)$ .

Let  $x$  be a finite energy signal. Two filters  $F_0$  and  $F_1$  are quadrature mirror filters (QMF) if, for any  $x$ ,

$$\|y_0\|^2 + \|y_1\|^2 = \|x\|^2$$

where  $y_0$  is a decimated version of the signal  $x$  filtered with  $F_0$  so  $y_0$  is defined by  $x_0 = F_0(x)$  and  $y_0(n) = x_0(2n)$ , and similarly,  $y_1$  is defined by  $x_1 = F_1(x)$  and  $y_1(n) = x_1(2n)$ . This property ensures a perfect reconstruction of the associated two-channel filter banks scheme (see Strang-Nguyen p. 103).

For example, if  $F_0$  is a Daubechies scaling filter and  $F_1 = \text{qmf}(F_0)$ , then the transfer functions  $F_0(z)$  and  $F_1(z)$  of the filters  $F_0$  and  $F_1$  satisfy the condition (see the example for *db10*):

$$|F_0(z)|^2 + |F_1(z)|^2 = 1$$

**Examples**

```
% Load scaling filter associated with an orthogonal wavelet.
load db10;
subplot(321); stem(db10); title('db10 low-pass filter');

% Compute the quadrature mirror filter.
qmfdb10 = qmf(db10);
subplot(322); stem(qmfdb10); title('QMF db10 filter');

% Check for frequency condition (necessary for orthogonality):
% abs(fft(filter))^2 + abs(fft(qmf(filter)))^2 = 1 at each
% frequency.
m = fft(db10);
mt = fft(qmfdb10);
freq = [1:length(db10)]/length(db10);
```

```

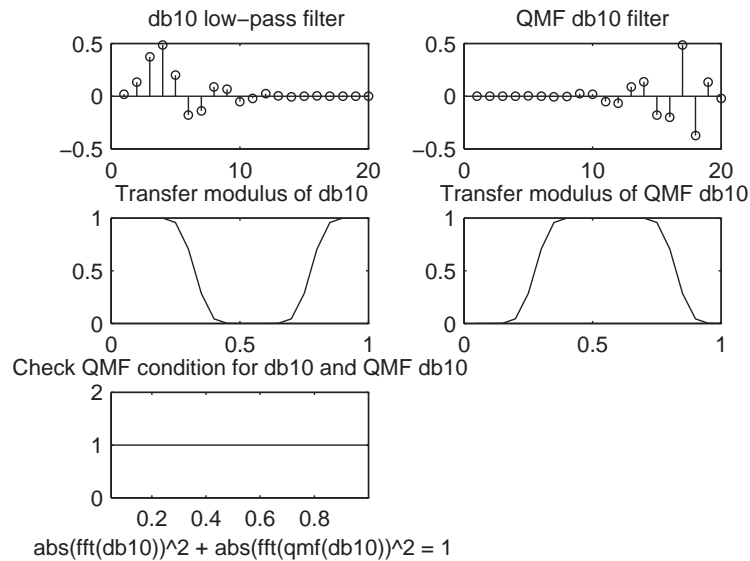
subplot(323); plot(freq,abs(m));
title('Transfer modulus of db10')
subplot(324); plot(freq,abs(mt));
title('Transfer modulus of QMF db10')
subplot(325); plot(freq,abs(m).^2 + abs(mt).^2);
title('Check QMF condition for db10 and QMF db10')
xlabel(' abs(fft(db10))^2 + abs(fft(qmf(db10)))^2 = 1')

```

```

% Editing some graphical properties,
% the following figure is generated.

```



```

% Check for orthonormality.
df = [db10;qmfdb10]*sqrt(2);
id = df*df'

id =
    1.0000    0.0000
    0.0000    1.0000

```

## References

Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press.

**Purpose** Reverse biorthogonal spline wavelet filters

**Syntax** [RF,DF]= rbiowavf(W)

**Description** [RF,DF] = rbiowavf(W) returns two scaling filters associated with the biorthogonal wavelet specified by the string W.

W = 'rbioNr.Nd' where possible values for Nr and Nd are

Nr = 1            Nd = 1 , 3 or 5

Nr = 2            Nd = 2 , 4 , 6 or 8

Nr = 3            Nd = 1 , 3 , 5 , 7 or 9

Nr = 4            Nd = 4

Nr = 5            Nd = 5

Nr = 6            Nd = 8

The output arguments are filters.

- RF is the reconstruction filter.
- DF is the decomposition filter.

## Examples

```
% Set reverse biorthogonal spline wavelet name.
wname = 'rbio2.2';

% Compute the two corresponding scaling filters,
% rf is the reconstruction scaling filter and
% df is the decomposition scaling filter.
[rf,df] = rbiowavf(wname)

rf =
    -0.1250    0.2500    0.7500    0.2500   -0.1250

df =
    0.2500    0.5000    0.2500
```

**See Also** biorfilt, waveinfo

# read

**Purpose** Read values of WPTREE

**Syntax** VARARGOUT = read(T,VARARGIN)

**Description** VARARGOUT = read(T,VARARGIN) is the most general syntax to read one or more property values from the fields of a WPTREE object .

The different ways to call the read function are

PropValue = read(T, 'PropName' ) or

PropValue = read(T, 'PropName' , 'PropParam' )

or any combination of the previous syntaxes:

[PropValue1,PropValue2, ] =  
read(T, 'PropName1' , 'PropParam1' , 'PropName2' , 'PropParam2' , )

where 'PropParam' is optional.

The valid choices for 'PropName' and 'PropParam' are listed in this table.

<i>PropName</i>	<i>PropParam</i>
'ent' , 'ento' or 'sizes' (see wptree)	Without 'PropParam' or with 'PropParam' = Vector of node indices, PropValue contains the entropy (or optimal entropy, or size) of the tree nodes in ascending node index order.
'cfs'	With 'PropParam' = One terminal node index. cfs = read(T, 'cfs' ,NODE) is equivalent to cfs = read(T, 'data' ,NODE) and returns the coefficients of the terminal node NODE .
'entName' , 'entPar' 'wavName' (see wptree) or 'allcfs'	Without 'PropParam'. cfs = read(T, 'allcfs' ) is equivalent to cfs = read(T, 'data' ). PropValue contains the desired information in ascending node index order of the tree nodes.

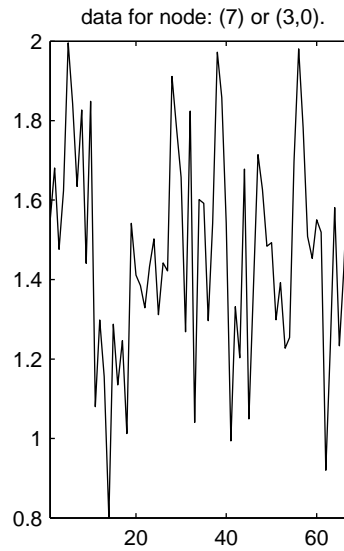
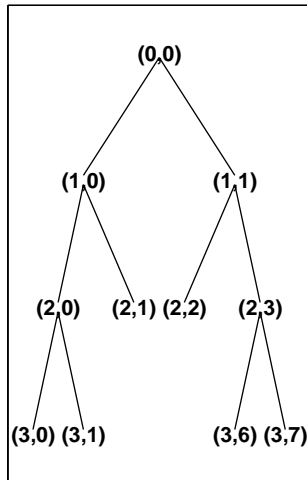
<i>PropName</i>	<i>PropParam</i>
'wfilters' (see wfilters)	Without ' <i>PropParam</i> ' or with ' <i>PropParam</i> ' = 'd','r','l','h'.
'data'	Without ' <i>PropParam</i> ' or with ' <i>PropParam</i> ' = One terminal node index or ' <i>PropParam</i> ' = Column vector of terminal node indices. In this last case, <i>PropValue</i> is a cell array. Without ' <i>PropParam</i> ', <i>PropValue</i> contains the coefficients of the tree nodes in ascending node index order.

## Examples

```
% Create a wavelet packet tree.
x = rand(1,512);
t = wpdec(x,3,'db3');
t = wpjoin(t,[4;5]);
plot(t);

% Click the node (3,0), (see the plot function).
```

```
% Read values.
```



```
sAll = read(t,'sizes');  
sNod = read(t,'sizes',[0,4,5]);  
eAll = read(t,'ent');  
eNod = read(t,'ent',[0,4,5]);  
dAll = read(t,'data');  
dNod = read(t,'data',[4;5]);  
[lo_D,hi_D,lo_R,hi_R] = read(t,'wfilters');  
[lo_D,lo_R,hi_D,hi_R] = read(t,'wfilters','l','wfilters','h');  
[ent,ento,cfs4,cfs5] = read(t,'ent','ento','cfs',4,'cfs',5);
```

## See Also

disp, get, set, wptree, write



**Purpose** Read wavelet packet decomposition tree from figure

**Syntax** `T = readtree(F)`

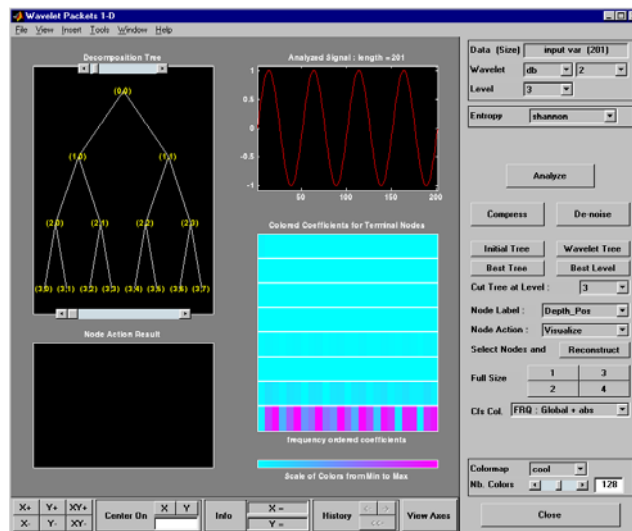
**Description** `T = readtree(F)` reads the wavelet packet decomposition tree from the figure whose handle is `F`.

For more information, see Chapter 5, “Using Wavelet Packets”, and Appendix B, “Object-Oriented Programming”.

**Examples**

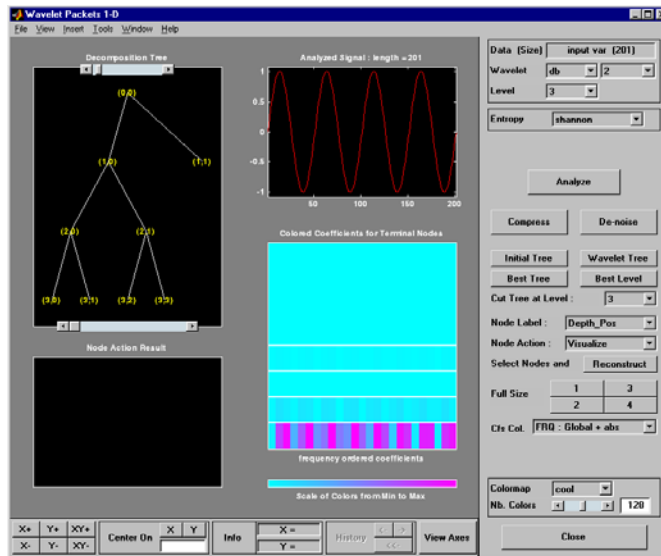
```
% Create a wavelet packet tree.
x = sin(8*pi*[0:0.005:1]);
t = wpdec(x,3,'db2');

% Display the generated tree in a Wavelet Packet 1-D GUI window.
fig = drawtree(t);
```



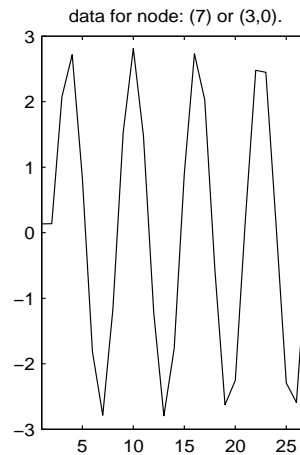
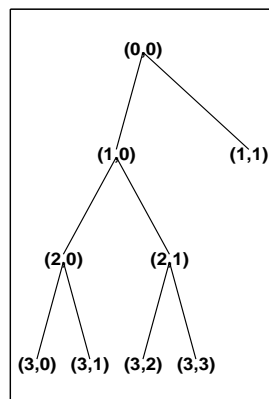
```
% -----
% Use the GUI to split or merge Nodes.
% -----
```

# readtree



```
t = readtree(fig);  
plot(t)
```

```
% Click the node (3,0), (see the plot function).
```



**See Also**

drawtree

**Purpose** Scale to frequency

**Syntax** `F = scal2frq(A, 'wname', DELTA)`

**Description** `F = scal2frq(A, 'wname', DELTA)` returns the pseudo-frequencies corresponding to the scales given by `A`, the wavelet function `'wname'` (see `wavefun` for more information) and the sampling period `DELTA`.

`scal2frq(A, 'wname')` is equivalent to `scal2frq(A, 'wname', 1)`.

One of the most frequently asked questions is “How does one map a scale, for a given wavelet and a sampling period, to a kind of 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.

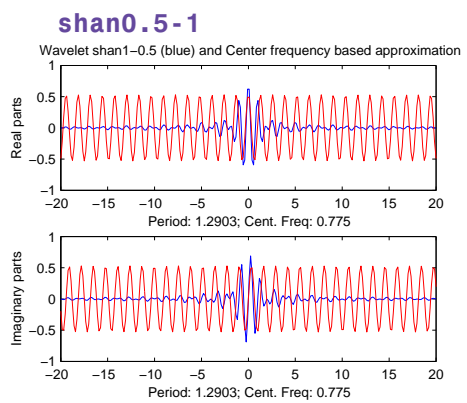
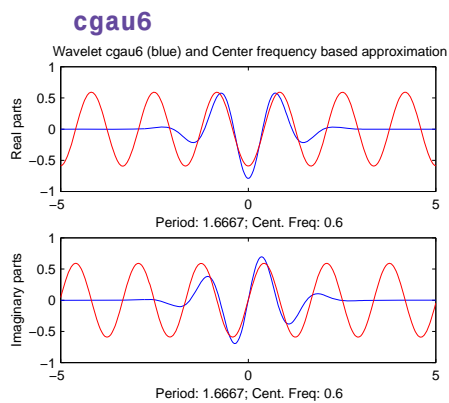
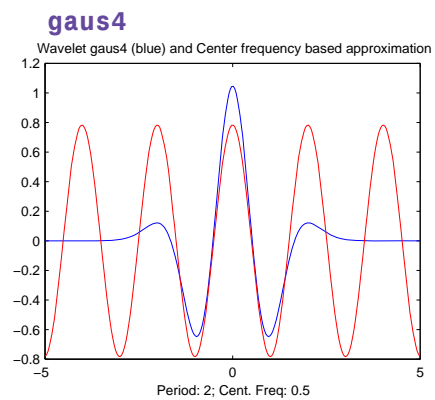
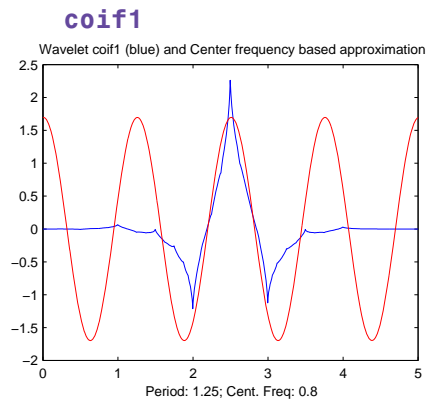
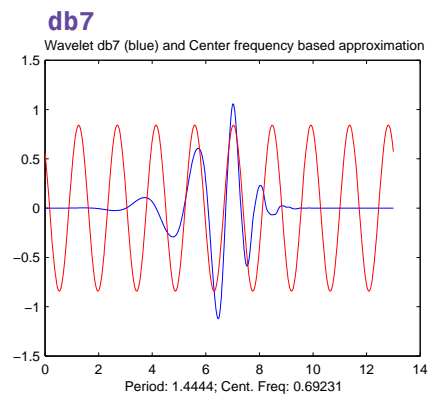
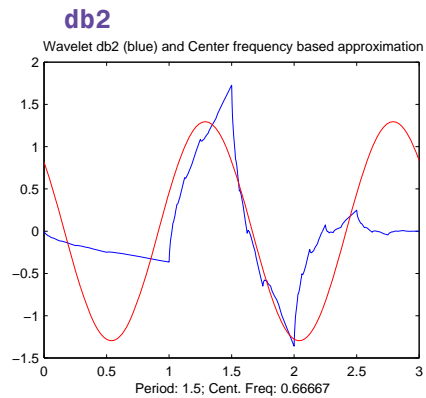
$$F_a = \frac{F_c}{a \cdot \Delta}$$

where

- $a$  is a scale.
- $\Delta$  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. The figure on the next page 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.



## Center Frequencies for Real and Complex Wavelets

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  $\Delta$ , 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.

To illustrate the behavior of this procedure, let us 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 example 2 presents the periods instead of frequencies. If we change slightly the  $F_0$  values, the results remain satisfactory.

## Examples

### Example 1

```
% Set sampling period and wavelet name.
delta = 0.1; wname = 'coif3';

% Define scales.
amax = 7; a = 2.^[1:amax];

% Compute associated pseudo-frequencies.
f = scal2frq(a,wname,delta);

% Compute associated pseudo-periods.
per = 1./f;
```

```
% Display information.
disp('   Scale   Frequency   Period')
disp(['a' f' per'])
```

Scale	Frequency	Period
2.0000	3.5294	0.2833
4.0000	1.7647	0.5667
8.0000	0.8824	1.1333
16.0000	0.4412	2.2667
32.0000	0.2206	4.5333
64.0000	0.1103	9.0667
128.0000	0.0551	18.1333

## Example 2

```
% Set sampling period and wavelet name.
delta = 0.1; wname = 'coif3';

% Define 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.
```

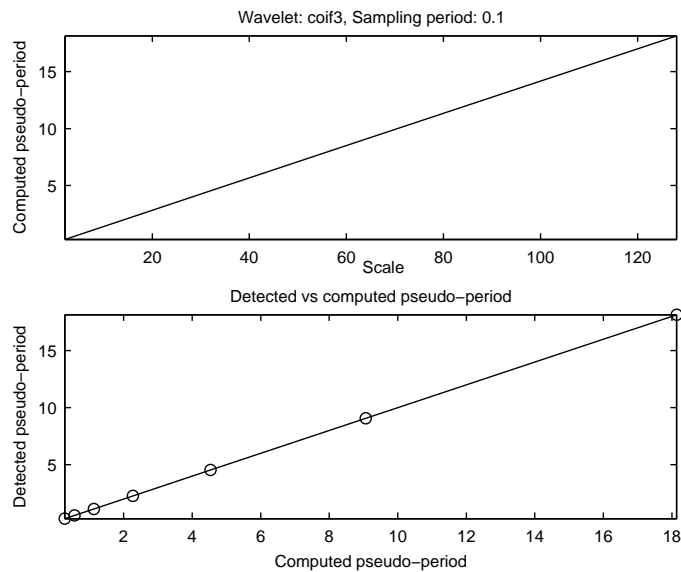
```

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')

```



## Example 3

This example demonstrates that, starting from the periodic function  $x(t) = \cos(5t)$ , the `scal2frq` function translates the scale corresponding to the maximum value of the CWT coefficients to a pseudo-frequency (0.795), which is near to the true frequency ( $5/(2\pi) \approx 0.796$ ).

```
% Set wavelet name, interval and number of samples.
wname = 'db10';
A = -64; B = 64; P = 224;

% Compute the sampling period and the sampled function,
% and the true frequency.
delta = (B-A)/(P-1);
t = linspace(A,B,P);
omega = 5; x = cos(omega*t);
freq = omega/(2*pi);

% Set scales and use scal2frq to compute the array
% of pseudo-frequencies.
scales = [0.25:0.25:3.75];
TAB_PF = scal2frq(scales,wname,delta);

% Compute the nearest pseudo-frequency
% and the corresponding scale.
[dummy,ind] = min(abs(TAB_PF-freq));
freq_APP = TAB_PF(ind);
scale_APP = scales(ind);

% Continuous analysis and plot.
str1 = ['224 samples of x = cos(5t) on [-64,64] - ' ...
       'True frequency = 5/(2*pi) =~ ' num2str(freq,3)];
str2 = ['Array of pseudo-frequencies and scales: '];
str3 = [num2str([TAB_PF,scales'],3)];
str4 = ['Pseudo-frequency = ' num2str(freq_APP,3)];
str5 = ['Corresponding scale = ' num2str(scale_APP,3)];
figure; cwt(x,scales,wname,'plot'); ax = gca; colorbar
axTITL = get(ax,'title');
axXLAB = get(ax,'xlabel');
set(axTITL,'String',str1)
set(axXLAB,'String',[str4,' - ' str5])
```



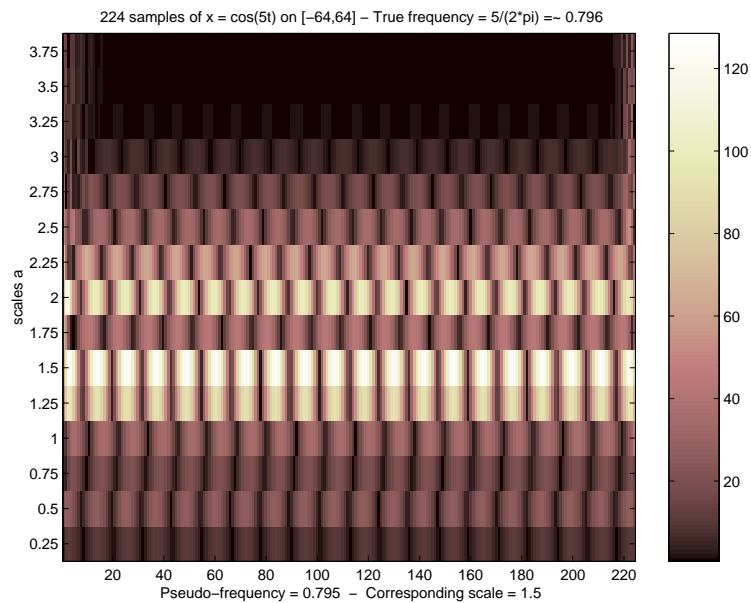
```
clc ; disp(strvcat(' ',str1,' ',str2,str3,' ',str4,str5))
```

224 samples of  $x = \cos(5t)$  on  $[-64,64]$  -  
 True frequency =  $5/(2\pi) \approx 0.796$

Array of pseudo-frequencies and scales:

4.77	0.25
2.38	0.5
1.59	0.75
1.19	1
0.954	1.25
0.795	1.5
0.681	1.75
0.596	2
. . . .	
0.341	3.5
0.318	3.75

Pseudo-frequency = 0.795  
 Corresponding scale = 1.5



## Example 4

This example demonstrates that, starting from the periodic function  $x(t) = 5\sin(5t) + 3\sin(2t) + 2\sin(t)$ , the `scal2frq` function translates the scales corresponding to the maximum values of the CWT coefficients to pseudo-frequencies (`[0.796 0.318 0.159]`), which are near to the true frequencies (`[5 2 1] / (2*pi) ≈ [0.796 0.318 0.159]`).

```
% Set wavelet name, interval and number of samples.
wname = 'morl';
A = 0; B = 64; P = 500;

% Compute the sampling period and the sampled function,
% and the true frequencies.
t = linspace(A,B,P);
delta = (B-A)/(P-1);
tab_OMEGA = [5,2,1];
tab_FREQ = tab_OMEGA/(2*pi);
tab_COEFS = [5,3,2];
x = zeros(1,P);
for k = 1:3;
    x = x+tab_COEFS(k)*sin(tab_OMEGA(k)*t);
end

% Set scales and use scal2frq to compute the array
% of pseudo-frequencies.
scales = [1:1:60];
tab_PF = scal2frq(scales,wname,delta);

% Compute the nearest pseudo-frequencies
% and the corresponding scales.
for k=1:3
    [dummy,ind] = min(abs(tab_PF-tab_FREQ(k)));
    PF_app(k) = tab_PF(ind);
    SC_app(k) = scales(ind);
end

% Continuous analysis and plot.
str1 = strvcats( ...
    '500 samples of x = 5*sin(5t)+3*sin(2t)+2*sin(t) on [0,64]',...
    ['True frequencies (in Hz): [5 2 1]/(2*pi) ≈ [' ...
```

```

        num2str(tab_FREQ,3) ']' ] ...
    );
str2 = ['Array of pseudo-frequencies and scales: '];
str3 = [num2str([tab_PF,scales'],3)];
str4 = ['Pseudo-frequencies = ' num2str(PF_app,3)];
str5 = ['Corresponding scales = ' num2str(SC_app,3)];
figure; cwt(x,scales,wname,'plot'); ax = gca; colorbar
axTITL = get(ax,'title');
axXLAB = get(ax,'xlabel');
set(axTITL,'String',str1)
set(axXLAB,'String',strvcat(str4, str5))
clc; disp(strvcat(' ',str1,' ',str2,str3,' ',str4,str5))

500 samples of  $x = 5\sin(5t)+3\sin(2t)+2\sin(t)$  on [0,64]
True frequencies (in Hz): [5 2 1]/(2*pi) =~ [0.796  0.318  0.159]

Array of pseudo-frequencies and scales:
  6.33      1
  3.17      2
  2.11      3
  1.58      4
  1.27      5
  1.06      6
  0.905     7
  0.792     8
  0.704     9
  0.633     10

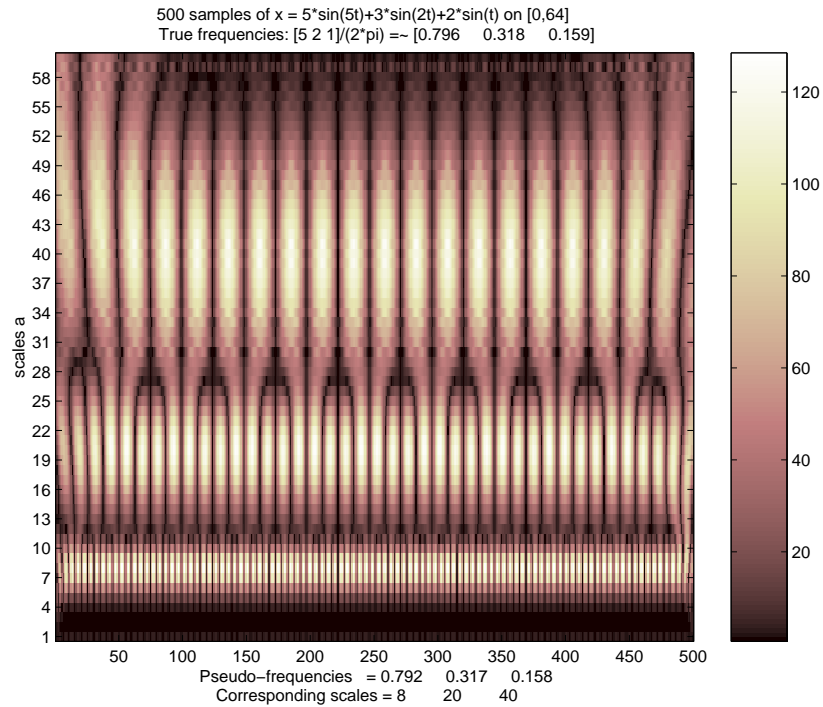
. . . .
. . . .

  0.122     52
  0.12      53
  0.117     54
  0.115     55
  0.113     56
  0.111     57
  0.109     58
  0.107     59
  0.106     60

```

# scal2frq

Pseudo-frequencies = 0.792      0.317      0.158  
Corresponding scales = 8      20      40



## References

Abry, P. (1997), *Ondelettes et turbulence. Multirésolutions, algorithmes de décomposition, invariance d'échelles*, Diderot Editeur, Paris.

## See Also

centfrq

**Purpose**

WPTREE field contents

**Syntax**

```
T = set(T, 'FieldName1', FieldValue1, 'FieldName2', FieldValue2, ...)
```

**Description**

`T = set(T, 'FieldName1', FieldValue1, 'FieldName2', FieldValue2, ...)` sets the content of the specified fields for the WPTREE object `T`.

For the fields that are objects or structures, you can set the subfield contents, giving the name of these subfields as '*FieldName*' values.

The valid choices for '*FieldName*' are

'dtree'     DTREE parent object  
'wavInfo'   Structure (wavelet information)

The fields of the wavelet information structure, 'wavInfo', are also valid for '*FieldName*':

'wavName'   Wavelet name  
'Lo\_D'       Low Decomposition filter  
'Hi\_D'       High Decomposition filter  
'Lo\_R'       Low Reconstruction filter  
'Hi\_R'       High Reconstruction filter

'entInfo'   Structure (entropy information)

The fields of the entropy information structure, 'entInfo', are also valid for '*FieldName*':

'entName'   Entropy name  
'entPar'     Entropy parameter

## set

---

Or fields of DTREE parent object:

'ntree'     NTREE parent object  
'allNI'     All nodes information  
'terNI'     Terminal nodes information

Or fields of NTREE parent object:

'wtbo'     WTBO parent object  
'order'     Order of the tree  
'depth'     Depth of the tree  
'spsch'     Split scheme for nodes  
'tn'        Array of terminal nodes of the tree

Or fields of WTBO parent object:

'wtboInfo'   Object information  
'ud'         Userdata field

---

**Caution** The set function should only be used to set the field '*ud*' .

---

### See Also

disp, get, read, write

**Purpose** Complex Shannon wavelet

**Syntax** [PSI,X] = shanwavf(LB,UB,N,FB,FC)

**Description** [PSI,X] = shanwavf(LB,UB,N,FB,FC) returns values of the complex Shannon wavelet defined by a bandwidth parameter FB, a wavelet center frequency FC, and the expression

$$\text{PSI}(X) = (\text{FB}^{0.5}) * (\text{sinc}(\text{FB} * X) .* \exp(2 * i * \pi * \text{FC} * X))$$

on an N point regular grid in the interval [LB,UB].

FB and FC must be such that  $\text{FC} > 0$  and  $\text{FB} > 0$ .

Output arguments are the wavelet function PSI computed on the grid X.

### Examples

```
% Set bandwidth and center frequency parameters.
fb = 1; fc = 1.5;
```

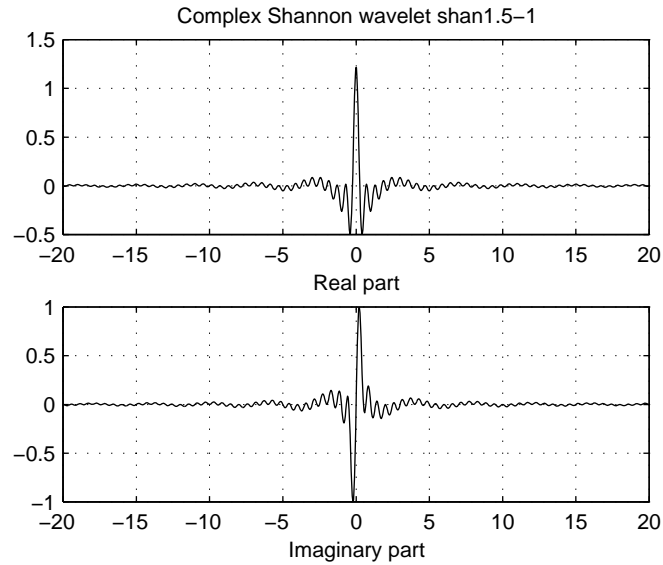
```
% Set effective support and grid parameters.
lb = -20; ub = 20; n = 1000;
```

```
% Compute complex Shannon wavelet shan1.5-1.
[psi,x] = shanwavf(lb,ub,n,fb,fc);
```

```
% Plot complex Shannon wavelet.
subplot(211)
plot(x,real(psi)),
title('Complex Shannon wavelet shan1.5-1')
xlabel('Real part'), grid
```

# shanwavf

```
subplot(212)  
plot(x,imag(psi))  
xlabel('Imaginary part'), grid
```



## References

Teolis, A. (1998), *Computational signal processing with wavelets*, Birkäuser, p. 62.

## See Also

waveinfo



**Purpose**

Discrete stationary wavelet transform 1-D

**Syntax**

```
SWC = swt(X,N,'wname')
SWC = swt(X,N,Lo_D,Hi_D)
[SWA,SWD] = swt(X,N,'wname')
[SWA,SWD] = swt(X,N,Lo_D,Hi_D)
```

**Description**

swt performs a multilevel 1-D stationary wavelet decomposition using either a specific orthogonal wavelet ('wname', see `wfilters` for more information) or specific orthogonal wavelet decomposition filters.

`SWC = swt(X,N,'wname')` computes the stationary wavelet decomposition of the signal `X` at level `N`, using 'wname'.

`N` must be a strictly positive integer (see `wmaxlev` for more information) and `length(X)` must be a multiple of  $2^N$ .

`SWC = swt(X,N,Lo_D,Hi_D)` computes the stationary wavelet decomposition as above, given these filters as input:

- `Lo_D` is the decomposition low-pass filter.
- `Hi_D` is the decomposition high-pass filter.

`Lo_D` and `Hi_D` must be the same length.

The output matrix `SWC` contains the vectors of coefficients stored row-wise:

For  $1 \leq i \leq N$ , the output matrix `SWC(i,:)` contains the detail coefficients of level `i` and `SWC(N+1,:)` contains the approximation coefficients of level `N`.

`[SWA,SWD] = swt( )` computes approximations, `SWA`, and details, `SWD`, stationary wavelet coefficients.

The vectors of coefficients are stored row-wise:

For  $1 \leq i \leq N$ , the output matrix `SWA(i,:)` contains the approximation coefficients of level `i` and the output matrix `SWD(i,:)` contains the detail coefficients of level `i`.

---

**Note** `swt` is defined using `dwt` with periodic extension.

---

## Examples

```
% Load original 1D signal.  
load noisbloc; s = noisbloc;
```

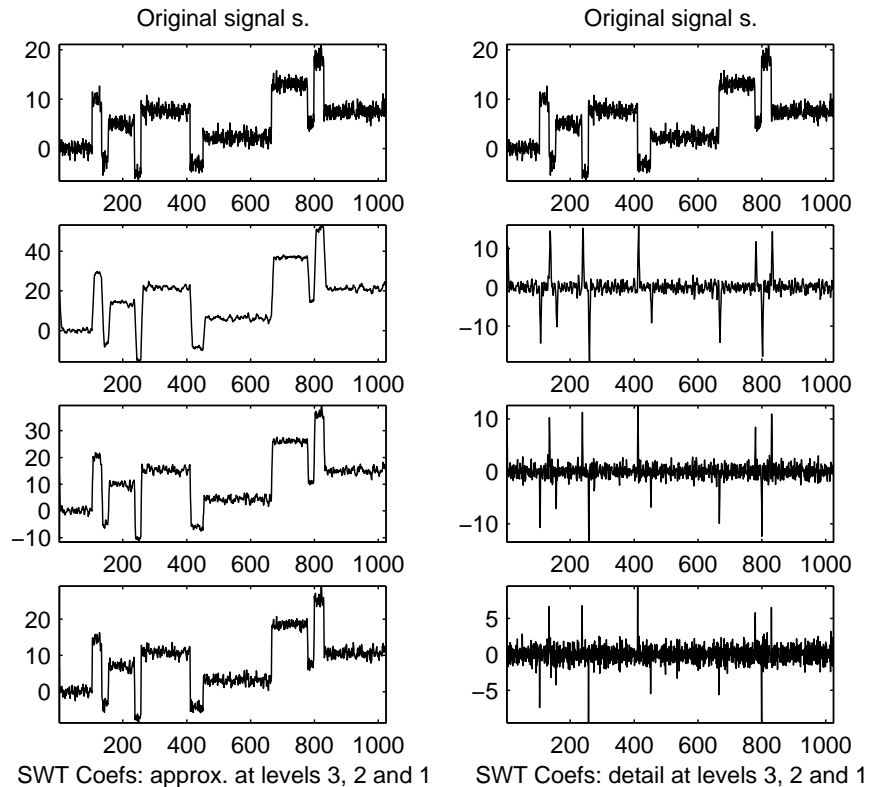
```

% Perform SWT decomposition at level 3 of s using db1.
[swa,swd] = swt(s,3,'db1');

% Plots of SWT coefficients of approximations and details
% at levels 3 to 1.

% Using some plotting commands,
% the following figure is generated.

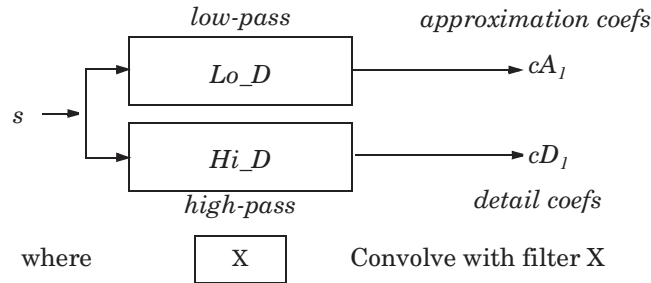
```



## Algorithm

Given a signal  $s$  of length  $N$ , the first step of the SWT produces, starting from  $s$ , two sets of coefficients: approximation coefficients  $cA_1$  and detail coefficients  $cD_1$ . These vectors are obtained by convolving  $s$  with the low-pass filter  $Lo\_D$  for approximation, and with the high-pass filter  $Hi\_D$  for detail.

More precisely, the first step is

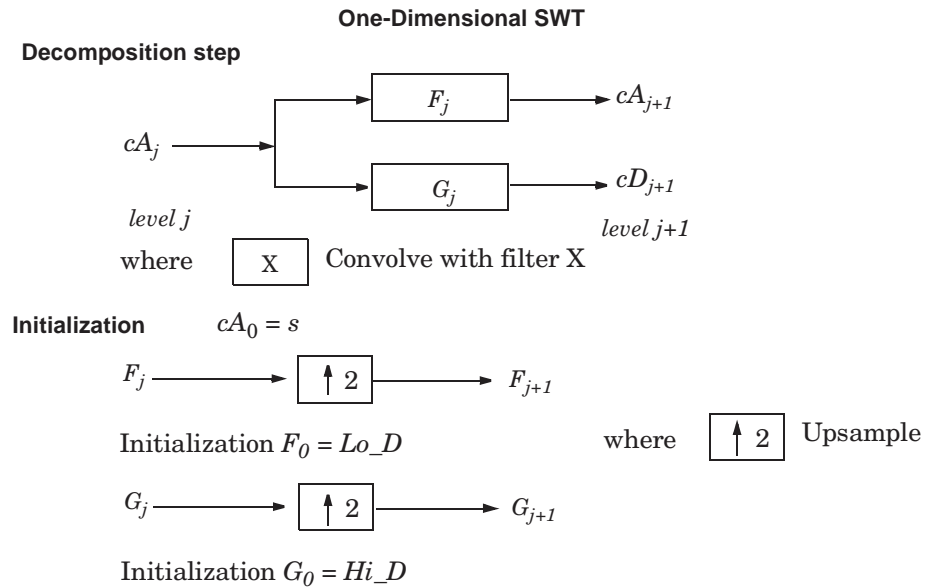



---

**Note**  $cA_1$  and  $cD_1$  are of length  $N$  instead of  $N/2$  as in the DWT case.

---

The next step splits the approximation coefficients  $cA_1$  in two parts using the same scheme, but with modified filters obtained by upsampling the filters used for the previous step and replacing  $s$  by  $cA_1$ . Then, the SWT produces  $cA_2$  and  $cD_2$ . More generally,



**References**

Nason, G.P.; B.W. Silverman (1995), "The stationary wavelet transform and some statistical applications," *Lecture Notes in Statistics*, 103, pp. 281–299.

Coifman, R.R.; Donoho, D.L. (1995), "Translation invariant de-noising," *Lecture Notes in Statistics*, 103, pp. 125–150.

Pesquet, J.C.; H. Krim, H. Carfatan (1996), "Time-invariant orthonormal wavelet representations," *IEEE Trans. Sign. Proc.*, vol. 44, 8, pp. 1964–1970.

**See Also**

dwt, wavedec

# swt2

---

**Purpose** Discrete stationary wavelet transform 2-D

**Syntax**

```
SWC = swt2(X,N,'wname')
[A,H,V,D] = swt2(X,N,'wname')
SWC = swt2(X,N,Lo_D,Hi_D)
[A,H,V,D] = swt2(X,N,Lo_D,Hi_D)
```

**Description** swt2 performs a multilevel 2-D stationary wavelet decomposition using either a specific orthogonal wavelet ('wname' — see `wfilters` for more information) or specific orthogonal wavelet decomposition filters.

`SWC = swt2(X,N,'wname')` or `[A,H,V,D] = swt2(X,N,'wname')` compute the stationary wavelet decomposition of the matrix `X` at level `N`, using 'wname'.

`N` must be a strictly positive integer (see `wmaxlev` for more information), and  $2^N$  must divide `size(X,1)` and `size(X,2)`.

Outputs `[A,H,V,D]` are 3-D arrays, which contain the coefficients:

- For  $1 \leq i \leq N$ , the output matrix `A(:, :, i)` contains the coefficients of approximation of level `i`.
- The output matrices `H(:, :, i)`, `V(:, :, i)` and `D(:, :, i)` contain the coefficients of details of level `i` (horizontal, vertical, and diagonal):

```
SWC = [H(:, :, 1:N) ; V(:, :, 1:N) ; D(:, :, 1:N) ; A(:, :, N)]
```

`SWC = swt2(X,N,Lo_D,Hi_D)` or `[A,H,V,D] = swt2(X,N,Lo_D,Hi_D)`, computes the stationary wavelet decomposition as in the previous syntax, given these filters as input:

- `Lo_D` is the decomposition low-pass filter.
- `Hi_D` is the decomposition high-pass filter.

`Lo_D` and `Hi_D` must be the same length.

---

**Note** swt2 is defined using `dwt` with periodic extension.

---

**Remarks** When `X` represents an indexed image, `X` is an `m`-by-`n` matrix and the output arrays `SWC` or `cA`, `cH`, `cV`, and `cD` are `m`-by-`n`-by-`p` arrays.

When  $X$  represents a truecolor image, it becomes an  $m$ -by- $n$ -by-3 array. This array is an  $m$ -by- $n$ -by-3 array, where each  $m$ -by- $n$  matrix represents a red, green, or blue color plane concatenated along the third dimension. The output arrays  $SWC$  or  $cA$ ,  $cH$ ,  $cV$ , and  $cD$  are  $m$ -by- $n$ -by- $p$ -by-3 arrays.

For more information on image formats, see the `image` and `imfinfo` reference pages.

## Examples

```
% Load original image.
load nbarb1;

% Image coding.
nbc = size(map,1);
cod_X = wcodemat(X,nbc);

% Visualize the original image.
subplot(221)
image(cod_X)
title('Original image');
colormap(map)

% Perform SWT decomposition
% of X at level 3 using sym4.
[ca, chd, cvd, cdd] = swt2(X,3, 'sym4');

% Visualize the decomposition.

for k = 1:3
    % Images coding for level k.
    cod_ca = wcodemat(ca(:,:,k),nbc);
    cod_chd = wcodemat(chd(:,:,k),nbc);
    cod_cvd = wcodemat(cvd(:,:,k),nbc);
    cod_cdd = wcodemat(cdd(:,:,k),nbc);
    decl = [cod_ca,cod_chd;cod_cvd,cod_cdd];

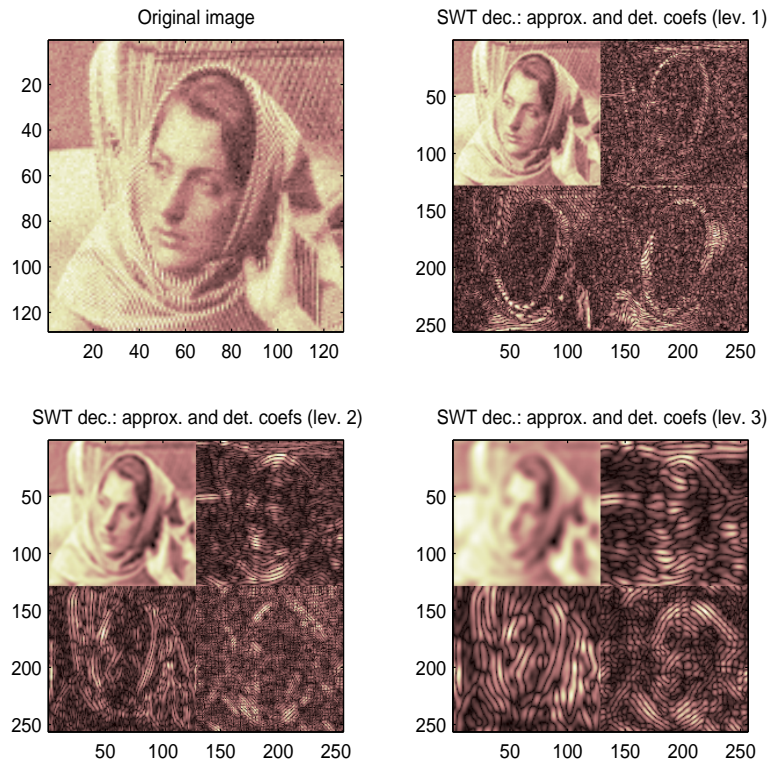
    % Visualize the coefficients of the decomposition
    % at level k.
    subplot(2,2,k+1)
    image(decl)
end
```

```

        title(['SWT dec.: approx. ', ...
              'and det. coefs (lev. ', num2str(k), ')']);
        colormap(map)
    end

% Editing some graphical properties,
% the following figure is generated.

```



## Algorithm

For images, an algorithm similar to the one-dimensional case is possible for two-dimensional wavelets and scaling functions obtained from one-dimensional ones by tensor product.

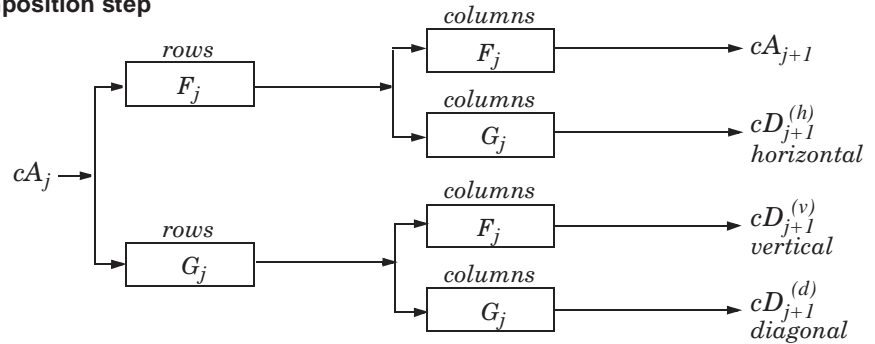
This kind of two-dimensional SWT leads to a decomposition of approximation coefficients at level  $j$  in four components: the approximation at level  $j+1$ , and the details in three orientations (horizontal, vertical, and diagonal).



The following chart describes the basic decomposition step for images:

### Two-Dimensional SWT

Decomposition step



where

$\begin{matrix} \text{rows} \\ \boxed{X} \end{matrix}$  Convolve with filter X the rows of the entry

$\begin{matrix} \text{columns} \\ \boxed{X} \end{matrix}$  Convolve with filter X the columns of the entry

**Initialization**  $cA_0 = s$  for the decomposition initialization

$$F_j \longrightarrow \boxed{\uparrow 2} \longrightarrow F_{j+1}$$

Initialization  $F_0 = Lo\_D$

where  $\boxed{\uparrow 2}$  Upsample

$$G_j \longrightarrow \boxed{\uparrow 2} \longrightarrow G_{j+1}$$

Initialization  $G_0 = Hi\_D$

**Note**  $size(cA_j) = size(cD_j^{(h)}) = size(cD_j^{(v)}) = size(cD_j^{(d)}) = s$

where  $s = size\ of\ the\ analyzed\ image$

## References

- Nason, G.P.; B.W. Silverman (1995), "The stationary wavelet transform and some statistical applications," *Lecture Notes in Statistics*, 103, pp. 281–299.
- Coifman, R.R.; Donoho, D.L. (1995), "Translation invariant de-noising," *Lecture Notes in Statistics*, 103, pp. 125–150.
- Pesquet, J.C.; H. Krim, H. Carfatan (1996), "Time-invariant orthonormal wavelet representations," *IEEE Trans. Sign. Proc.*, vol. 44, 8, pp. 1964–1970.

## See Also

dwt2, wavedec2

**Purpose** Symlet wavelet filter computation

**Syntax** `W = SYMAUX(N, SUMW)`  
`W = SYMAUX(N)`

**Description** Symlets are the "least asymmetric" Daubechies wavelets.  
`W = SYMAUX(N, SUMW)` is the order N Symlet scaling filter such that  $SUM(W) = SUMW$ . Possible values for N are 1, 2, 3, ...

**Note** Instability may occur when N is too large.

`W = SYMAUX(N)` is equivalent to `W = SYMAUX(N, 1)`.

`W = SYMAUX(N, 0)` is equivalent to `W = SYMAUX(N, 1)`.

**Examples**

```
% Generate wdb4 the order 4 Daubechies scaling filter.
wdb4 = dbaux(4)
```

```
wdb4 =
```

```
Columns 1 through 7
```

```
0.1629    0.5055    0.4461   -0.0198   -0.1323    0.0218    0.0233
```

```
Column 8
```

```
-0.0075
```

```
% wdb4 is a solution of the equation: P = conv(wrev(w),w)*2,
% where P is the "Lagrange - trous" filter for N=4.
% wdb4 is a minimum phase solution of the previous equation,
% based on the roots of P (see dbaux).
P = conv(wrev(wdb4),wdb4)*2;
```

```
% Generate wsym4 the order 4 symlet scaling filter.
% The Symlets are the "least asymmetric" Daubechies'
% wavelets obtained from another choice between the roots of P.
wsym4 = symaux(4)

wsym4 =

Columns 1 through 7

    0.0228   -0.0089   -0.0702    0.2106    0.5683    0.3519   -0.0210

Column 8

   -0.0536

% Compute conv(wrev(wsym4),wsym4) * 2 and check that wsym4
% is another solution of the equation P = conv(wrev(w),w)*2.
Psym = conv(wrev(wsym4),wsym4)*2;
err = norm(P-Psym)

err =

    7.4988e-016
```

## See Also

symwavf, wfilters

**Purpose** Symlet wavelet filter

**Syntax** `F = symwavf(W)`

**Description** `F = symwavf(W)` returns the scaling filter associated with the symlet wavelet specified by the string `W` where `W = 'symN'`. Possible values for `N` are 2, 3, ..., 45.

**Examples**

```
% Compute the scaling filter corresponding to wavelet sym4.  
w = symwavf('sym4')  
  
w =  
Columns 1 through 7  
    0.0228 -0.0089 -0.0702 0.2106 0.5683 0.3519 -0.0210  
Column 8  
    -0.0536
```

**See Also** `symaux`, `waveinfo`

# thselect

---

**Purpose** Threshold selection for de-noising

**Syntax** `THR = thselect(X,TPTR)`

**Description** `thselect` is a one-dimensional de-noising oriented function.

`THR = thselect(X,TPTR)` returns threshold  $X$ -adapted value using selection rule defined by string `TPTR`.

Available selection rules are

- `TPTR = 'rigrsure'`, adaptive threshold selection using principle of Stein's Unbiased Risk Estimate.
- `TPTR = 'heursure'`, heuristic variant of the first option.
- `TPTR = 'sqtwolog'`, threshold is  $\sqrt{2 \cdot \log(\text{length}(X))}$ .
- `TPTR = 'minimaxi'`, minimax thresholding.

Threshold selection rules are based on the underlying model  $y = f(t) + e$  where  $e$  is a white noise  $N(0,1)$ . Dealing with unscaled or nonwhite noise can be handled using rescaling output threshold `THR` (see `SCAL` parameter in `wden` for more information).

Available options are

- `tptr = 'rigrsure'` uses for the soft threshold estimator, a threshold selection rule based on Stein's Unbiased Estimate of Risk (quadratic loss function). One gets an estimate of the risk for a particular threshold value  $t$ . Minimizing the risks in  $t$  gives a selection of the threshold value.
- `tptr = 'sqtwolog'` uses a fixed-form threshold yielding minimax performance multiplied by a small factor proportional to  $\log(\text{length}(X))$ .
- `tptr = 'heursure'` is a mixture of the two previous options. As a result, if the signal to noise ratio is very small, the SURE estimate is very noisy. If such a situation is detected, the fixed form threshold is used.
- `tptr = 'minimaxi'` uses a fixed threshold chosen to yield minimax performance for mean square error against an ideal procedure. The minimax principle is used in statistics in order to design estimators. Since the de-noised signal can be assimilated to the estimator of the unknown regression function, the minimax estimator is the one that realizes the

minimum of the maximum mean square error obtained for the worst function in a given set.

## Examples

```
% The current extension mode is zero-padding (see dwtmode).

% Generate Gaussian white noise.
init = 2055415866; randn('seed',init);
x = randn(1,1000);

% Find threshold for each selection rule.
% Adaptive threshold using SURE.
thr = thselect(x,'rigrsure')
thr =
    1.8065

% Fixed form threshold.
thr = thselect(x,'sqtwolog')
thr =
    3.7169

% Heuristic variant of the first option.
thr = thselect(x,'heursure')
thr =
    3.7169

% Minimax threshold.
thr = thselect(x,'minimaxi')
thr =
    2.2163
```

## References

- 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.
- Donoho, D.L., I.M. Johnstone (1994), “Ideal spatial adaptation by wavelet shrinkage,” *Biometrika*, vol 81, pp. 425–455.
- Donoho, D.L. (1995), “De-noising by soft-thresholding,” *IEEE Trans. on Inf. Theory*, 41, 3, pp. 613–627.

## See Also

wden

# tnodes

**Purpose** Determine terminal nodes

**Syntax**

```
N = tnodes(T)
N = tnodes(T, 'deppos')
[N,K] = tnodes(T)
[N,K] = tnodes(T, 'deppos')
```

**Description** tnodes is a tree-management utility.

$N = \text{tnodes}(T)$  returns the indices of terminal nodes of the tree  $T$ .  $N$  is a column vector.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

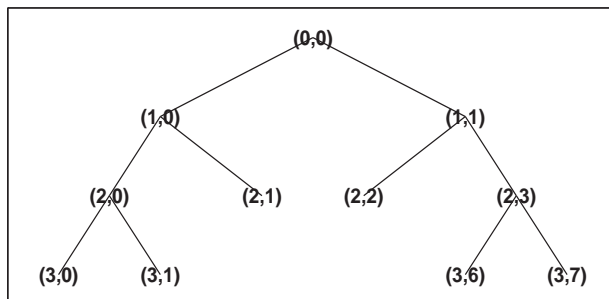
$N = \text{tnodes}(T, 'deppos')$  returns a matrix  $N$ , which contains the depths and positions of terminal nodes.

$N(i, 1)$  is the depth of the  $i$ -th terminal node.  $N(i, 2)$  is the position of the  $i$ -th terminal node.

For  $[N,K] = \text{tnodes}(T)$  or  $[N,K] = \text{tnodes}(T, 'deppos')$ ,  $M = N(K)$  are the indices reordered as in tree  $T$ , from left to right.

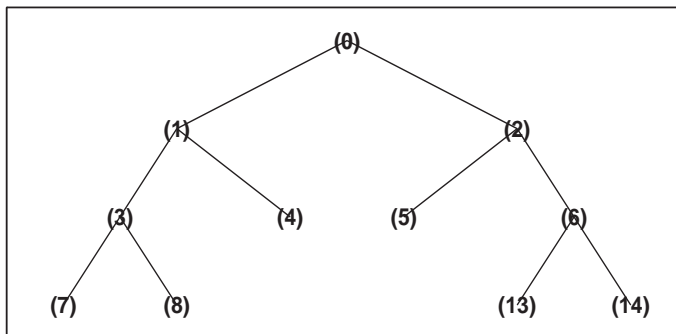
## Examples

```
% Create initial tree.
ord = 2;
t = ntree(ord,3);      % Binary tree of depth 3.
t = nodejoin(t,5);
t = nodejoin(t,4);
plot(t)
```





```
% Change Node Label from Depth_Position to Index
% (see the plot function).
```



```
% List terminal nodes (index).
tnodes(t)
```

```
ans =
     4
     5
     7
     8
    13
    14
```

```
% List terminal nodes (Depth_Position).
tnodes(t,'deppos')
```

```
ans =
     2     1
     2     2
     3     0
     3     1
     3     6
     3     7
```

## See Also

leaves, noleaves, wtreesmgr

# treedpth

---

**Purpose** Tree depth

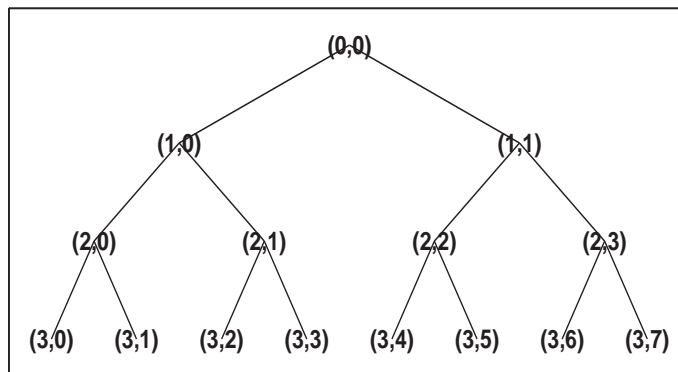
**Syntax** `D = treedpth(T)`

**Description** `treedpth` is a tree-management utility.

`D = treedpth(T)` returns the depth `D` of the tree `T`.

**Examples**

```
% Create binary tree (tree of order 2) of depth 3.  
t = ntree(2,3);  
  
% Plot tree t.  
plot(t)
```



```
% Tree depth.  
treedpth(t)
```

```
ans =  
    3
```

**See Also** `wtreemgr`

**Purpose** Tree order

**Syntax** ORD = treeord(T)

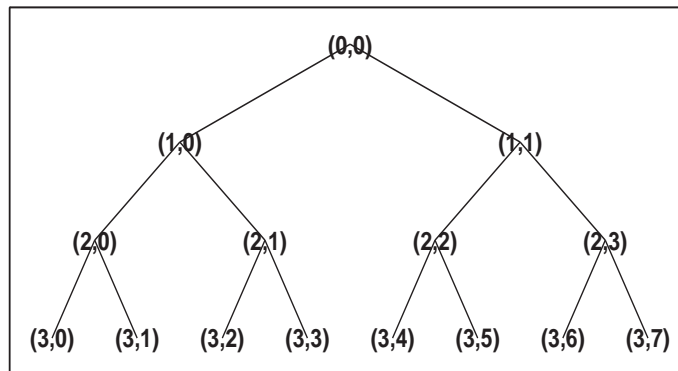
**Description** treeord is a tree-management utility.

ORD = treeord(T) returns the order ORD of the tree T.

**Examples**

```
% Create binary tree (tree of order 2) of depth 3.
t = ntree(2,3);
```

```
% Plot tree t.
plot(t)
```



```
% Tree order.
treeord(t)
```

```
ans =
     2
```

**See Also** wtreemgr

**Purpose** Direct reconstruction from 1-D wavelet coefficients

**Syntax**

```
Y = upcoef(0,X,'wname',N)
Y = upcoef(0,X,'wname',N,L)
Y = upcoef(0,X,Lo_R,Hi_R,N)
Y = upcoef(0,X,Lo_R,Hi_R,N,L)
Y = upcoef(0,X,'wname')
Y = upcoef(0,X,Lo_R,Hi_R)
```

**Description** upcoef is a one-dimensional wavelet analysis function.

`Y = upcoef(0,X,'wname',N)` computes the N-step reconstructed coefficients of vector X.

'wname' is a string containing the wavelet name. See `wfilters` for more information.

N must be a strictly positive integer.

If 0 = 'a', approximation coefficients are reconstructed.

If 0 = 'd', detail coefficients are reconstructed.

`Y = upcoef(0,X,'wname',N,L)` computes the N-step reconstructed coefficients of vector X and takes the length-L central portion of the result.

Instead of giving the wavelet name, you can give the filters.

For `Y = upcoef(0,X,Lo_R,Hi_R,N)` or `Y = upcoef(0,X,Lo_R,Hi_R,N,L)`, Lo\_R is the reconstruction low-pass filter and Hi\_R is the reconstruction high-pass filter.

`Y = upcoef(0,X,'wname')` is equivalent to `Y = upcoef(0,X,'wname',1)`.

`Y = upcoef(0,X,Lo_R,Hi_R)` is equivalent to `Y = upcoef(0,X,Lo_R,Hi_R,1)`.

**Examples**      % The current extension mode is zero-padding (see `dwtmode`).

```
% Approximation signals, obtained from a single coefficient
% at levels 1 to 6.
```

```
cfs = [1]; % Decomposition reduced a single coefficient.
essup = 10; % Essential support of the scaling filter db6.
figure(1)
```

```

for i=1:6
    % Reconstruct at the top level an approximation
    % which is equal to zero except at level i where only
    % one coefficient is equal to 1.
    rec = upcoef('a',cfs,'db6',i);

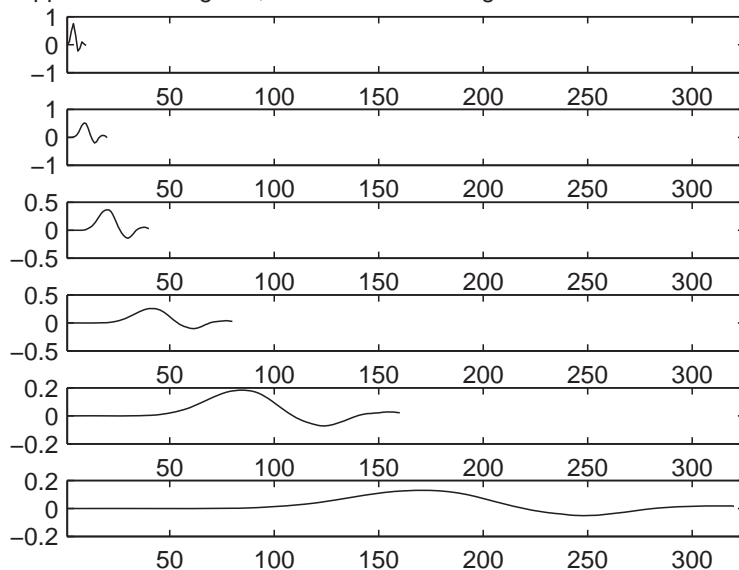
    % essup is the essential support of the
    % reconstructed signal.
    % rec(j) is very small when j is  $\mathfrak{R} \geq \text{essup}$ .
    ax = subplot(6,1,i),h = plot(rec(1:essup));
    set(ax,'xlim',[1 325]);
    essup = essup*2;

end
subplot(6,1,1)
title(['Approximation signals, obtained from a single ' ...
      'coefficient at levels 1 to 6'])

% Editing some graphical properties,
% the following figure is generated.

```

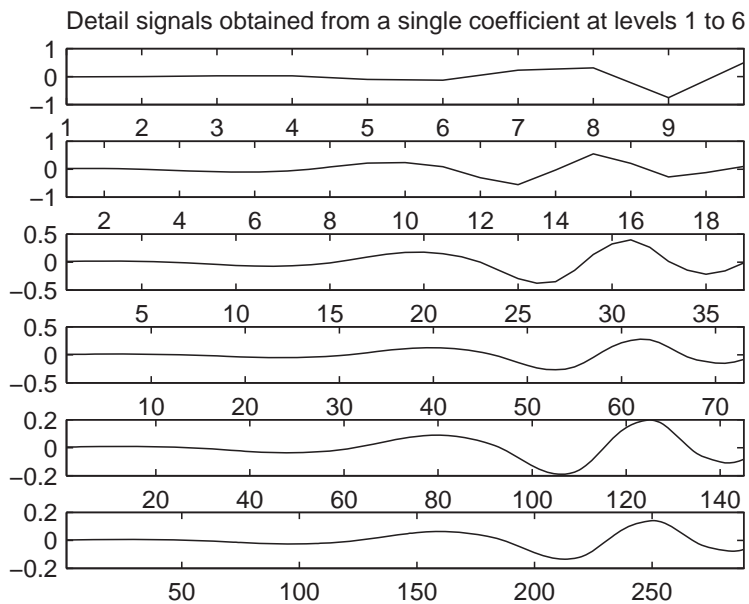
Approximation signals, obtained from a single coefficient at levels 1 to 6



```
% The same can be done for details.
% Details signals, obtained from a single coefficient
% at levels 1 to 6.

cfs = [1];
mi = 12; ma = 30; % Essential support of
                  % the wavelet filter db6.
rec = upcoef('d',cfs,'db6',1);
figure(2)
subplot(611), plot(rec(3:12))
for i=2:6
    % Reconstruct at top level a single detail
    % coefficient at level i.
    rec = upcoef('d',cfs,'db6',i);
    subplot(6,1,i), plot(rec(mi*2^(i-2):ma*2^(i-2)))
end
subplot(611)
title(['Detail signals obtained from a single ' ...
      'coefficient at levels 1 to 6'])
```

```
% Editing some graphical properties,
% the following figure is generated.
```



### Algorithm

upcoef is equivalent to an N time repeated use of the inverse wavelet transform.

### See Also

idwt

# upcoef2

---

**Purpose** Direct reconstruction from 2-D wavelet coefficients

**Syntax**

```
Y = upcoef2(O,X,'wname',N,S)
Y = upcoef2(O,X,Lo_R,Hi_R,N,S)
Y = upcoef2(O,X,'wname',N)
Y = upcoef2(O,X,Lo_R,Hi_R,N)
Y = upcoef2(O,X,'wname')
Y = upcoef2(O,X,Lo_R,Hi_R)
```

**Description** upcoef2 is a two-dimensional wavelet analysis function.

`Y = upcoef2(O,X,'wname',N,S)` computes the N-step reconstructed coefficients of matrix `X` and takes the central part of size `S`. `'wname'` is a string containing the name of the wavelet. See `wfilters` for more information.

If `O = 'a'`, approximation coefficients are reconstructed; otherwise if `O = 'h'` (`'v'` or `'d'`, respectively), horizontal (vertical or diagonal, respectively) detail coefficients are reconstructed. `N` must be a strictly positive integer.

Instead of giving the wavelet name, you can give the filters.

For `Y = upcoef2(O,X,Lo_R,Hi_R,N,S)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

`Y = upcoef2(O,X,'wname',N)` or `Y = upcoef2(O,X,Lo_R,Hi_R,N)` returns the computed result without any truncation.

`Y = upcoef2(O,X,'wname')` is equivalent to `Y = upcoef2(O,X,'wname',1)`.

`Y = upcoef2(O,X,Lo_R,Hi_R)` is equivalent to `Y = upcoef2(O,X,Lo_R,Hi_R,1)`.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).
```

```
% Load original image.
load woman;
% X contains the loaded image.
```

```
% Perform decomposition at level 2
% of X using db4.
[c,s] = wavedec2(X,2,'db4');
```



```
% Reconstruct approximation and details
% at level 1, from coefficients.
% This can be done using wrcoef2, or
% equivalently using:
%
% Step 1: Extract coefficients from the
% decomposition structure [c,s].
%
% Step 2: Reconstruct using upcoef2.

siz = s(size(s,1),:);

ca1 = appcoef2(c,s,'db4',1);
a1 = upcoef2('a',ca1,'db4',1,siz);

chd1 = detcoef2('h',c,s,1);
hd1 = upcoef2('h',chd1,'db4',1,siz);

cvd1 = detcoef2('v',c,s,1);
vd1 = upcoef2('v',cvd1,'db4',1,siz);

cdd1 = detcoef2('d',c,s,1);
dd1 = upcoef2('d',cdd1,'db4',1,siz);
```

**Algorithm**

See upcoef.

**See Also**

idwt2

# upwlev

---

**Purpose** Single-level reconstruction of 1-D wavelet decomposition

**Syntax** `[NC,NL,cA] = upwlev(C,L,'wname')`  
`[NC,NL,cA] = upwlev(C,L,Lo_R,Hi_R)`

**Description** `upwlev` is a one-dimensional wavelet analysis function.

`[NC,NL,cA] = upwlev(C,L,'wname')` performs the single-level reconstruction of the wavelet decomposition structure `[C,L]` giving the new one `[NC,NL]`, and extracts the last approximation coefficients vector `cA`.

`[C,L]` is a decomposition at level `n = length(L) - 2`, so `[NC,NL]` is the same decomposition at level `n-1` and `cA` is the approximation coefficients vector at level `n`.

'`wname`' is a string containing the wavelet name, `C` is the original wavelet decomposition vector, and `L` the corresponding bookkeeping vector (for detailed storage information, see `wavedec`).

Instead of giving the wavelet name, you can give the filters.

For `[NC,NL,cA] = upwlev(C,L,Lo_R,Hi_R)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

**Examples** `% The current extension mode is zero-padding (see dwtmode).`

```
% Load original one-dimensional signal.
load sumsin; s = sumsin;

% Perform decomposition at level 3 of s using db1.
[c,l] = wavedec(s,3,'db1');
subplot(311); plot(s);
title('Original signal s.');
```

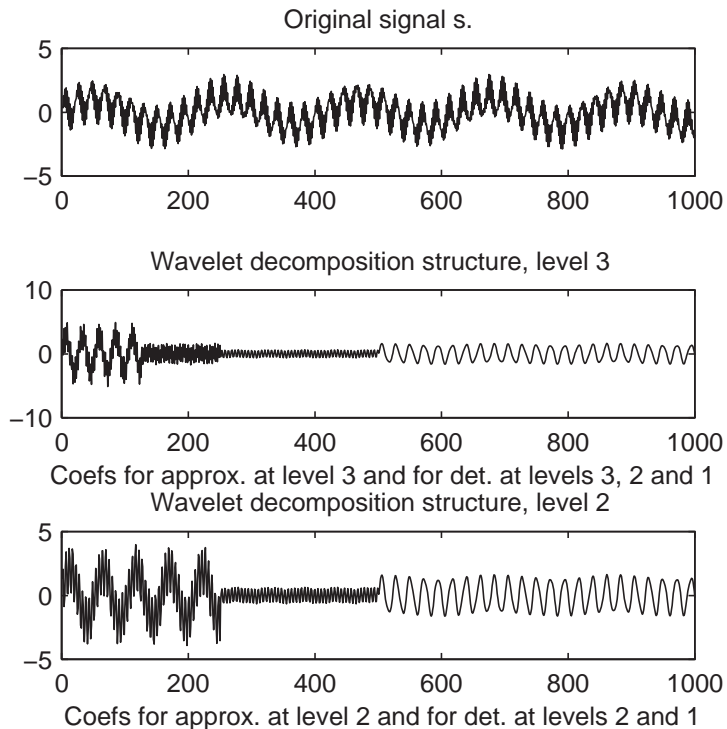
```
subplot(312); plot(c);
title('Wavelet decomposition structure, level 3')
xlabel(['Coefs for approx. at level 3 ' ...
        'and for det. at levels 3, 2 and 1'])
```

```

% One step reconstruction of the wavelet decomposition
% structure at level 3 [c,l], so the new structure [nc,nl]
% is the wavelet decomposition structure at level 2.
[nc,nl] = upwlev(c,l,'db1');
subplot(313); plot(nc);
title('Wavelet decomposition structure, level 2')
xlabel(['Coefs for approx. at level 2 ' ...
        'and for det. at levels 2 and 1'])

% Editing some graphical properties,
% the following figure is generated.

```

**See Also**

idwt, upcoef, wavedec

# upwlev2

---

**Purpose** Single-level reconstruction of 2-D wavelet decomposition

**Syntax** `[NC,NS,cA] = upwlev2(C,S,'wname')`  
`[NC,NS,cA] = upwlev2(C,S,Lo_R,Hi_R)`

**Description** `upwlev2` is a two-dimensional wavelet analysis function.

`[NC,NS,cA] = upwlev2(C,S,'wname')` performs the single-level reconstruction of wavelet decomposition structure `[C,S]` giving the new one `[NC,NS]`, and extracts the last approximation coefficients matrix `cA`.

`[C,S]` is a decomposition at level `n = size(S,1)-2`, so `[NC,NS]` is the same decomposition at level `n-1` and `cA` is the approximation matrix at level `n`.

'*wname*' is a string containing the wavelet name, `C` is the original wavelet decomposition vector, and `S` the corresponding bookkeeping matrix (for detailed storage information, see `wavedec2`).

Instead of giving the wavelet name, you can give the filters.

For `[NC,NS,cA] = upwlev2(C,S,Lo_R,Hi_R)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

% Load original image.
load woman;
% X contains the loaded image.

% Perform decomposition at level 2
% of X using db1.
[c,s] = wavedec2(X,2,'db1');
sc = size(c)

sc =
     1    65536

val_s = s
```

```
val_s =
    64    64
    64    64
   128   128
   256   256

% One step reconstruction of wavelet
% decomposition structure [c,s].
[nc,ns] = upwlev2(c,s,'db1');
snc = size(nc)

snc =
     1   65536

val_ns = ns

val_ns =
    128   128
    128   128
    256   256
```

**See Also**

idwt2, upcoef2, wavedec2

# wave2lp

---

**Purpose** Laurent polynomials associated with wavelet

**Syntax** [Hs,Gs,Ha,Ga] = wave2lp(W)

**Description** [Hs,Gs,Ha,Ga] = wave2lp(W) returns the four Laurent polynomials associated with the wavelet W (see liftwave).

The pairs (Hs,Gs) and (Ha,Ga) are the synthesis and the analysis pair respectively.

The H-polynomials (G-polynomials) are low-pass (high-pass) polynomials.

For an orthogonal wavelet, Hs = Ha and Gs = Ga.

**Examples** % Get Laurent polynomials associated to the "lazy" wavelet.  
[Hs,Gs,Ha,Ga] = wave2lp('lazy')

$$Hs(z) = 1$$

$$Gs(z) = z^{-1}$$

$$Ha(z) = 1$$

$$Ga(z) = z^{-1}$$

% Get Laurent polynomials associated to the db1 wavelet.  
[Hs,Gs,Ha,Ga] = wave2lp('db1')

$$Hs(z) = + 0.7071 + 0.7071*z^{-1}$$

$$Gs(z) = - 0.7071 + 0.7071*z^{-1}$$

$$Ha(z) = + 0.7071 + 0.7071*z^{-1}$$

$$Ga(z) = - 0.7071 + 0.7071*z^{-1}$$

% Get Laurent polynomials associated to the bior1.3 wavelet.  
[Hs,Gs,Ha,Ga] = wave2lp('bior1.3')

$$Hs(z) = + 0.7071 + 0.7071*z^{-1}$$

$$\begin{aligned} G_s(z) = & \dots \\ & + 0.08839*z^{(+2)} + 0.08839*z^{(+1)} - 0.7071 + 0.7071*z^{(-1)} - \\ & 0.08839*z^{(-2)} \dots \\ & - 0.08839*z^{(-3)} \end{aligned}$$

$$\begin{aligned} H_a(z) = & \dots \\ & - 0.08839*z^{(+2)} + 0.08839*z^{(+1)} + 0.7071 + 0.7071*z^{(-1)} + \\ & 0.08839*z^{(-2)} \dots \\ & - 0.08839*z^{(-3)} \end{aligned}$$

$$G_a(z) = - 0.7071 + 0.7071*z^{(-1)}$$

**See Also**

laurpoly

# wavedec

---

**Purpose** Multilevel 1-D wavelet decomposition

**Syntax** `[C,L] = wavedec(X,N,'wname')`  
`[C,L] = wavedec(X,N,Lo_D,Hi_D)`

**Description** `wavedec` performs a multilevel one-dimensional wavelet analysis using either a specific wavelet (`'wname'`) or a specific wavelet decomposition filters (`Lo_D` and `Hi_D`, see `wfilters`).

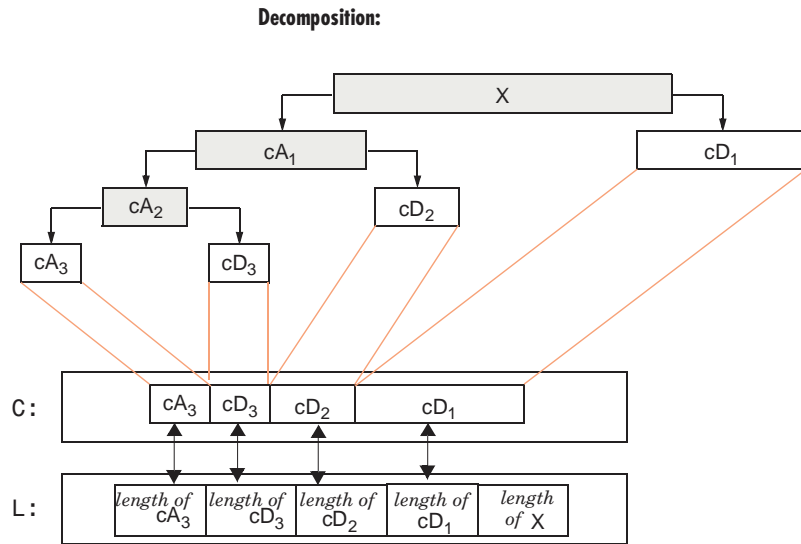
---

**Note** `wavedec` supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets.

---

`[C,L] = wavedec(X,N,'wname')` returns the wavelet decomposition of the signal `X` at level `N`, using `'wname'`. `N` must be a strictly positive integer (see `wmaxlev` for more information). The output decomposition structure contains the wavelet decomposition vector `C` and the bookkeeping vector `L`. The structure is organized as in this level-3 decomposition example.

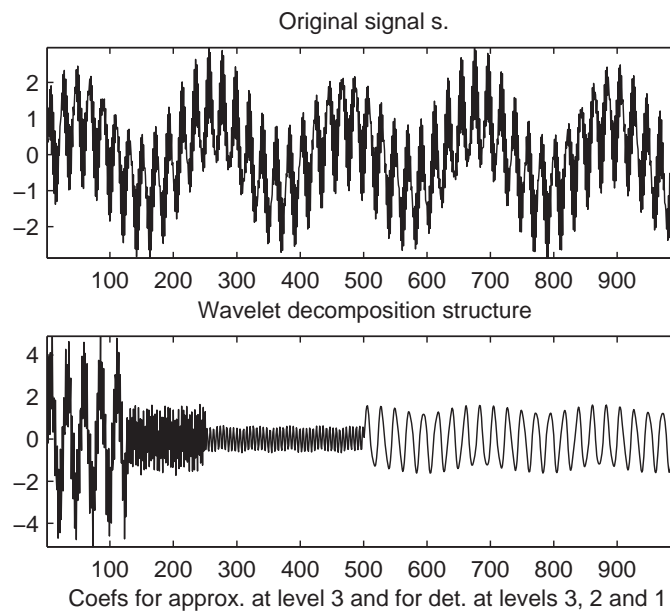




$[C, L] = \text{wavedec}(X, N, \text{Lo}_D, \text{Hi}_D)$  returns the decomposition structure as above, given the low- and high-pass decomposition filters you specify.

## Examples

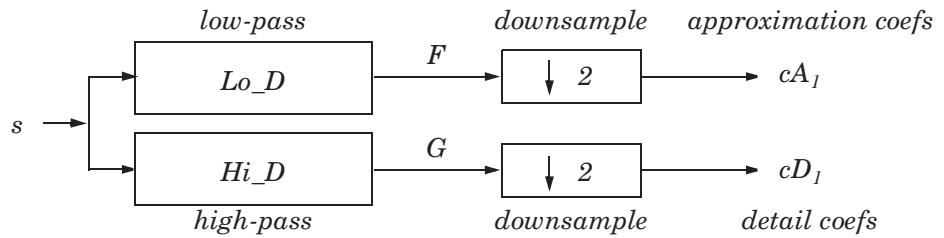
```
% The current extension mode is zero-padding (see dwtmode).  
  
% Load original one-dimensional signal.  
load sumsin; s = sumsin;  
% Perform decomposition at level 3 of s using db1.  
[c,l] = wavedec(s,3,'db1');  
  
% Using some plotting commands,  
% the following figure is generated.
```



## Algorithm

Given a signal  $s$  of length  $N$ , the DWT consists of  $\log_2 N$  stages at most. The first step produces, starting from  $s$ , two sets of coefficients: approximation coefficients  $CA_1$ , and detail coefficients  $CD_1$ . These vectors are obtained by convolving  $s$  with the low-pass filter  $Lo\_D$  for approximation, and with the high-pass filter  $Hi\_D$  for detail, followed by dyadic decimation (downsampling).

More precisely, the first step is



where

X	Convolve with filter X
↓ 2	Keep the even indexed elements (We call this operation <i>downsampling</i> .)

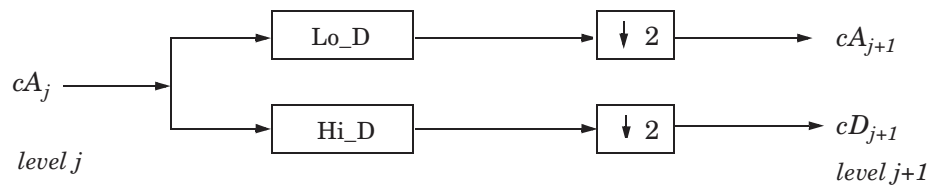
The length of each filter is equal to  $2N$ . If  $n = \text{length}(s)$ , the signals  $F$  and  $G$  are of length  $n + 2N - 1$  and the coefficients  $cA_1$  and  $cD_1$  are of length

$$\text{floor}\left(\frac{n-1}{2}\right) + N$$

The next step splits the approximation coefficients  $cA_1$  in two parts using the same scheme, replacing  $s$  by  $cA_1$ , and producing  $cA_2$  and  $cD_2$ , and so on

### One-Dimensional DWT

#### Decomposition step



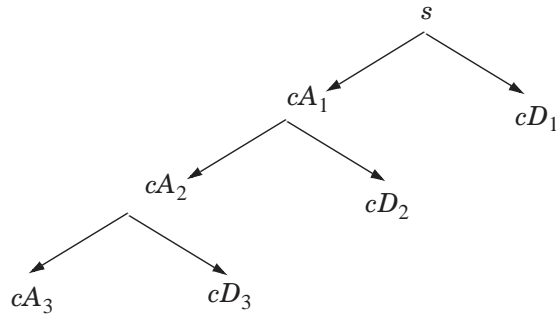
where

X	Convolve with filter X
↓ 2	Downsample

**Initialization**  $cA_0 = s$

The wavelet decomposition of the signal  $s$  analyzed at level  $j$  has the following structure:  $[cA_j, cD_j, \dots, cD_1]$ .

This structure contains, for  $J = 3$ , the terminal nodes of the following tree:



## References

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

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.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

## See Also

dwt, waveinfo, waverec, wfilters, wmaxlev

**Purpose** Multilevel 2-D wavelet decomposition

**Syntax** `[C,S] = wavedec2(X,N,'wname')`  
`[C,S] = wavedec2(X,N,Lo_D,Hi_D)`

**Description** `wavedec2` is a two-dimensional wavelet analysis function.

`[C,S] = wavedec2(X,N,'wname')` returns the wavelet decomposition of the matrix `X` at level `N`, using the wavelet named in string `'wname'` (see `wfilters` for more information).

Outputs are the decomposition vector `C` and the corresponding bookkeeping matrix `S`.

`N` must be a strictly positive integer (see `wmaxlev` for more information).

Instead of giving the wavelet name, you can give the filters.

For `[C,S] = wavedec2(X,N,Lo_D,Hi_D)`, `Lo_D` is the decomposition low-pass filter and `Hi_D` is the decomposition high-pass filter.

Vector `C` is organized as

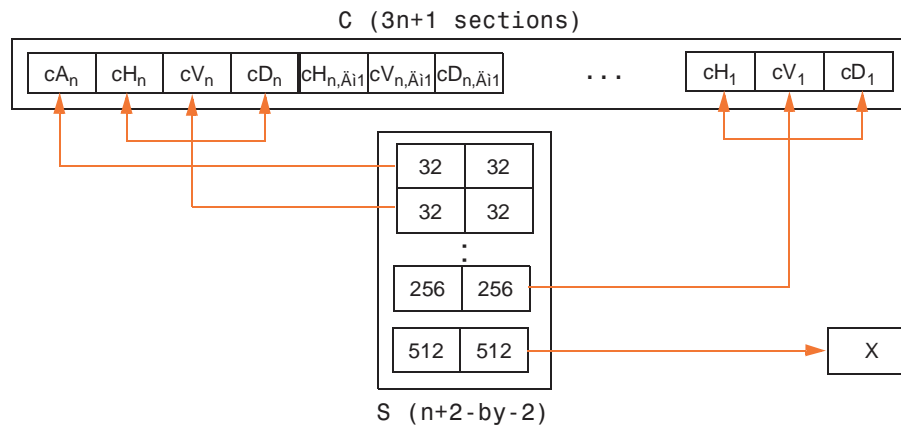
$$C = [ A(N) \mid H(N) \mid V(N) \mid D(N) \mid \dots \\ H(N-1) \mid V(N-1) \mid D(N-1) \mid \dots \mid H(1) \mid V(1) \mid D(1) ].$$

where `A`, `H`, `V`, `D`, are row vectors such that

- `A` = approximation coefficients
- `H` = horizontal detail coefficients
- `V` = vertical detail coefficients
- `D` = diagonal detail coefficients
- Each vector is the vector column-wise storage of a matrix.

Matrix `S` is such that

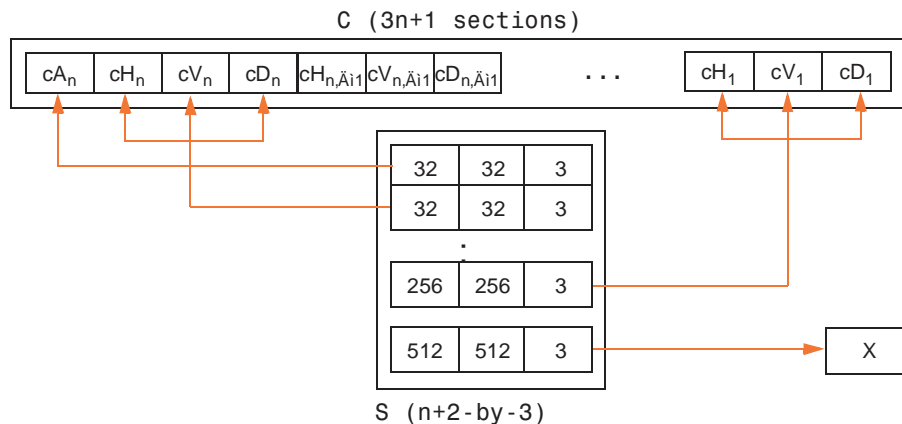
- `S(1,:) = size of approximation coefficients(N)`.
- `S(i,:) = size of detail coefficients(N-i+2)` for `i = 2, ...N+1` and `S(N+2,:) = size(X)`.



## Remarks

When  $X$  represents an indexed image,  $X$ , as well as the output arrays  $cA$ ,  $cH$ ,  $cV$ , and  $cD$  are  $m$ -by- $n$  matrices. When  $X$  represents a truecolor image, it is an  $m$ -by- $n$ -by-3 array, where each  $m$ -by- $n$  matrix represents a red, green, or blue color plane concatenated along the third dimension. The size of vector  $C$  and the size of matrix  $S$  depend on the type of analyzed image.

For a truecolor image, the decomposition vector  $C$  and the corresponding bookkeeping matrix  $S$  can be represented as follows.



For more information on image formats, see the `image` and `imfinfo` reference pages.

## Examples

```
% The current extension mode is zero-padding (see dwtmode).

% Load original image.
load woman;
% X contains the loaded image.

% Perform decomposition at level 2
% of X using db1.
[c,s] = wavedec2(X,2,'db1');

% Decomposition structure organization.
sizeX = size(X)

sizeX =
    256  256
sizeC = size(c)

sizeC =
     1 65536
    val_s = s

val_s =
    64  64
    64  64
   128 128
   256 256
```

## Algorithm

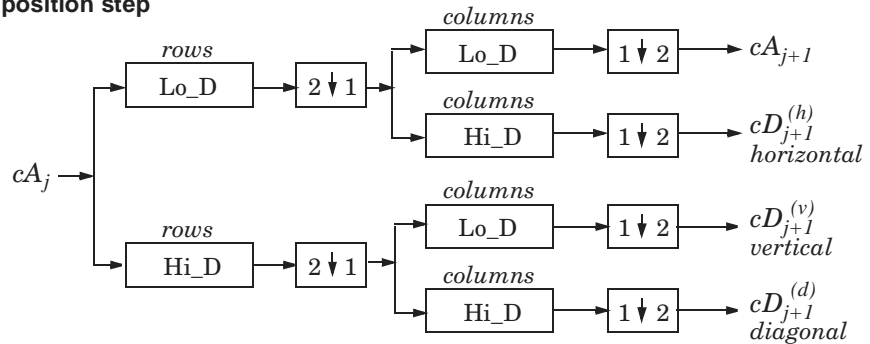
For images, an algorithm similar to the one-dimensional case is possible for two-dimensional wavelets and scaling functions obtained from one-dimensional ones by tensor product.

This kind of two-dimensional DWT leads to a decomposition of approximation coefficients at level  $j$  in four components: the approximation at level  $j+1$ , and the details in three orientations (horizontal, vertical, and diagonal).

The following chart describes the basic decomposition step for images:

## Two-Dimensional DWT

### Decomposition step



where  $\begin{bmatrix} 2 \downarrow 1 \end{bmatrix}$  Downsample columns: keep the even indexed columns

$\begin{bmatrix} 1 \downarrow 2 \end{bmatrix}$  Downsample rows: keep the even indexed rows

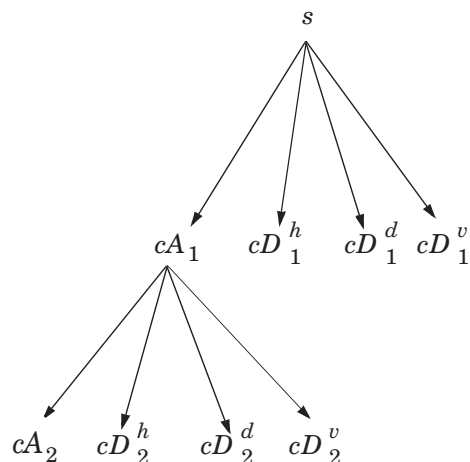
$\begin{matrix} \text{rows} \\ \boxed{X} \end{matrix}$  Convolve with filter X the rows of the entry

$\begin{matrix} \text{columns} \\ \boxed{X} \end{matrix}$  Convolve with filter X the columns of the entry

**Initialization**  $cA_0 = s$  for the decomposition initialization

So, for  $J=2$ , the two-dimensional wavelet tree has the form





## References

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

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.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

## See Also

dwt, waveinfo, waverec2, wfilters, wmaxlev

# wavedemo

---

**Purpose** Wavelet Toolbox™ software demos

**Syntax** wavedemo

**Description** wavedemo opens a GUI that allows you to choose between several Wavelet Toolbox demos.

**Purpose**

Wavelet and scaling functions

**Syntax**

```
[PHI,PSI,XVAL] = wavefun('wname',ITER)
[PHI1,PSI1,PHI2,PSI2,XVAL] = wavefun('wname',ITER)
[PSI,XVAL] = wavefun('wname',ITER)
[...] = wavefun('wname',A,B)
```

**Description**

The function wavefun returns approximations of the wavelet function 'wname' and the associated scaling function, if it exists. The positive integer ITER determines the number of iterations computed; thus, the refinement of the approximations.

*For an orthogonal wavelet:*

[PHI,PSI,XVAL] = wavefun('wname',ITER) returns the scaling and wavelet functions on the points grid XVAL.

*For a biorthogonal wavelet:*

[PHI1,PSI1,PHI2,PSI2,XVAL] = wavefun('wname',ITER) returns the scaling and wavelet functions both for decomposition (PHI1,PSI1) and for reconstruction (PHI2,PSI2).

*For a Meyer wavelet:*

[PHI,PSI,XVAL] = wavefun('wname',ITER)

*For a wavelet without scaling function (e.g., Morlet, Mexican Hat, Gaussian derivatives wavelets or complex wavelets):*

[PSI,XVAL] = wavefun('wname',ITER)

[...] = wavefun('wname',A,B), where A and B are positive integers, is equivalent to [...] = wavefun('wname',max(A,B)), and draws plots.

When A is set equal to the special value 0,

- [...] = wavefun('wname',0) is equivalent to
- [...] = wavefun('wname',8,0).
- [...] = wavefun('wname') is equivalent to
- [...] = wavefun('wname',8).

The output arguments are optional.

# wavefun

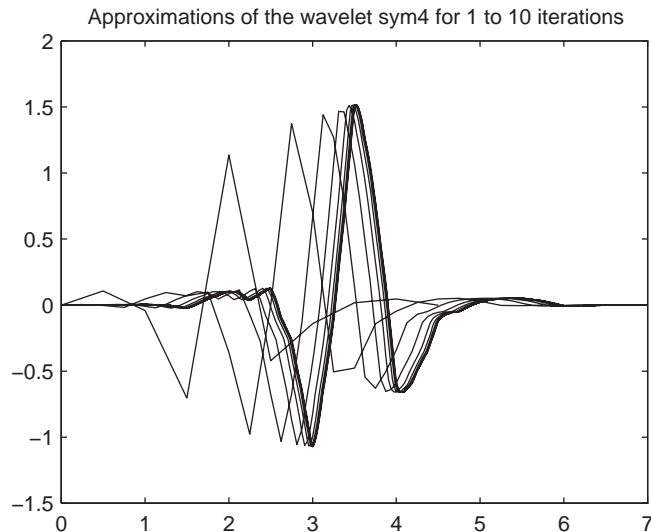
---

## Examples

On the following graph, 10 piecewise linear approximations of the sym4 wavelet obtained after each iteration of the cascade algorithm are shown.

```
% Set number of iterations and wavelet name.
iter = 10;
wav = 'sym4';

% Compute approximations of the wavelet function using the
% cascade algorithm.
for i = 1:iter
    [phi,psi,xval] = wavefun(wav,i);
    plot(xval,psi);
    hold on
end
title(['Approximations of the wavelet ',wav, ...
       ' for 1 to ',num2str(iter),' iterations']);
hold off
```



## Algorithm

For compactly supported wavelets defined by filters, in general no closed form analytic formula exists.

The algorithm used is the cascade algorithm. It uses the single-level inverse wavelet transform repeatedly.

Let us begin with the scaling function  $\phi$ .

Since  $\phi$  is also equal to  $\phi_{0,0}$ , (according to the notation used in Chapter 6, “Advanced Concepts”), this function is characterized by the following coefficients in the orthogonal framework:

- $\langle \phi, \phi_{0,n} \rangle = 1$  only if  $n = 0$  and equal to 0 otherwise
- $\langle \phi, \psi_{-j,k} \rangle = 0$  for positive  $j$ , and all  $k$ .

This expansion can be viewed as a wavelet decomposition structure. Detail coefficients are all zeros and approximation coefficients are all zeros except one equal to 1.

Then we use the reconstruction algorithm to approximate the function  $\phi$  over a dyadic grid, according to the following result:

For any dyadic rational of the form  $x = n2^{-j}$  in which the function is continuous and where  $j$  is sufficiently large, we have pointwise convergence and

$$\left| \phi(x) - 2^{\frac{j}{2}} \langle \phi, \phi_{-j, n2^{-j}} \rangle \right| \leq C \cdot 2^{-j\alpha}$$

where  $C$  is a constant, and  $\alpha$  is a positive constant depending on the wavelet regularity.

Then using a good approximation of  $\phi$  on dyadic rationals, we can use piecewise constant or piecewise linear interpolations  $\eta$  on dyadic intervals, for which uniform convergence occurs with similar exponential rate:

$$\|\phi - \eta\|_{\infty} \leq C \cdot 2^{-j\alpha}$$

So using a  $J$ -step reconstruction scheme, we obtain an approximation that converges exponentially towards  $\phi$  when  $J$  goes to infinity.

Approximations are computed over a grid of dyadic rationals covering the support of the function to be approximated.

Since a scaled version of the wavelet function  $\psi$  can also be expanded on the  $(\phi_{-1,n})_n$ , the same scheme can be used, after a single-level reconstruction

starting with the appropriate wavelet decomposition structure. Approximation coefficients are all zeros and detail coefficients are all zeros except one equal to 1.

For biorthogonal wavelets, the same ideas can be applied on each of the two multiresolution schemes in duality.

---

**Note** This algorithm may diverge if the function to be approximated is not continuous on dyadic rationals.

---

## References

Daubechies, I., *Ten lectures on wavelets*, CBMS, SIAM, 1992, pp. 202,Äì213.  
Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press.

## See Also

`intwave`, `waveinfo`, `wfilters`

**Purpose** Wavelet and scaling functions 2-D

**Syntax**

```
[S,W1,W2,W3,XYVAL] = wavefun2('wname',ITER)
[S,W1,W2,W3,XYVAL] = wavefun2('wname',ITER,'plot')
[S,W1,W2,W3,XYVAL] = wavefun2('wname',A,B)
```

**Description** For an orthogonal wavelet *wname*, wavefun2 returns the scaling function and the three wavelet functions resulting from the tensor products of the one-dimensional scaling and wavelet functions.

If [PHI,PSI,XVAL] = wavefun('wname',ITER), the scaling function S is the tensor product of PHI and PSI.

The wavelet functions W1, W2, and W3 are the tensor products (PHI,PSI), (PSI,PHI), and (PSI,PSI), respectively.

The two-dimensional variable XYVAL is a  $2^{ITER} \times 2^{ITER}$  points grid obtained from the tensor product (XVAL,XVAL).

The positive integer ITER determines the number of iterations computed and thus, the refinement of the approximations.

[S,W1,W2,W3,XYVAL] = wavefun2('wname',ITER,'plot') computes and also plots the functions.

[S,W1,W2,W3,XYVAL] = wavefun2('wname',A,B), where A and B are positive integers, is equivalent to

[S,W1,W2,W3,XYVAL] = wavefun2('wname',max(A,B)). The resulting functions are plotted.

When A is set equal to the special value 0,

- [S,W1,W2,W3,XYVAL] = wavefun2('wname',0) is equivalent to  
[S,W1,W2,W3,XYVAL] = wavefun2('wname',4,0).
- [S,W1,W2,W3,XYVAL] = wavefun2('wname') is equivalent to  
[S,W1,W2,W3,XYVAL] = wavefun2('wname',4).

The output arguments are optional.

---

**Note** The wavefun2 function can only be used with an orthogonal wavelet.

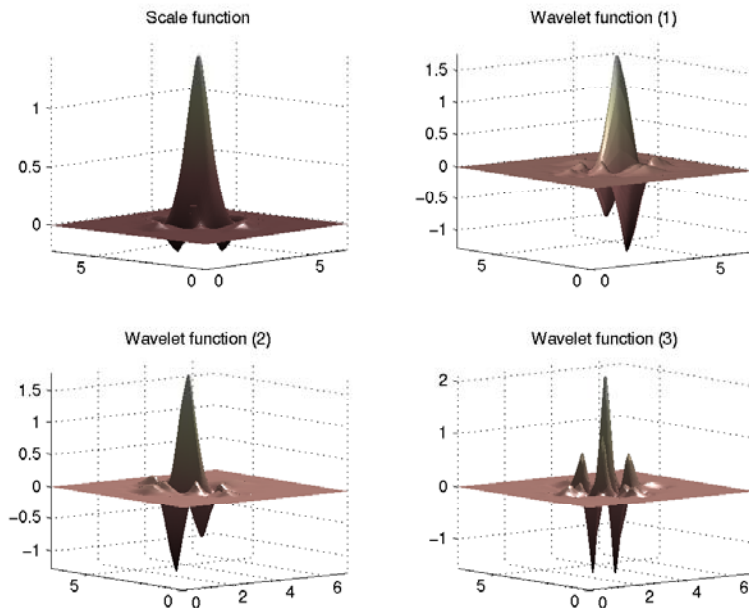
---

# wavefun2

## Examples

On the following graph, a linear approximation of the sym4 wavelet obtained using the cascade algorithm is shown.

```
% Set number of iterations and wavelet name.  
iter = 4;  
wav = 'sym4';  
  
% Compute approximations of the wavelet and scale functions using  
% the cascade algorithm and plot.  
[s,w1,w2,w3,xyval] = wavefun2(wav,iter,0);
```



## Algorithm

See wavefun for more information.

## References

Daubechies, I., *Ten lectures on wavelets*, CBMS, SIAM, 1992, pp. 202,Äi213.  
Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press.

## See Also

intwave, wavefun, waveinfo, wfilters



**Purpose** Wavelets information

**Syntax** `waveinfo`  
`waveinfo('wname')`

**Description** `waveinfo` provides information on all wavelets within the toolbox.

`waveinfo('wname')` provides information on the wavelet family whose short name is specified by the string 'wname'. Available family short names are listed in the table below.

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

# waveinfo

---

The family short names can also be user-defined ones (see `wavemngr` for more information).

`waveinfo('wsys')` provides information on wavelet packets.

## Examples

```
waveinfo('db')
```

```
DBINFO Information on Daubechies wavelets.
```

```
Daubechies Wavelets
```

```
General characteristics: Compactly supported  
wavelets with extremal phase and highest  
number of vanishing moments for a given  
support width. Associated scaling filters are  
minimum-phase filters.
```

```
Family           Daubechies  
Short name       db  
Order N          N strictly positive integer  
Examples         db1 or haar, db4, db15
```

```
Orthogonal       yes  
Biorthogonal     yes  
Compact support  yes  
DWT              possible  
CWT              possible
```

```
Support width    2N-1  
Filters length   2N  
Regularity       about 0.2 N for large N  
Symmetry         far from  
Number of vanishing moments for psi    N
```

```
Reference:  I. Daubechies,  
Ten lectures on wavelets CBMS, SIAM, 61, 1994, 194-202.
```

## See Also

`wavemngr`

**Purpose** Wavelet families and family members

**Syntax**

```
waveletfamilies
waveletfamilies('f')
waveletfamilies('n')
waveletfamilies('a')
```

**Description**

waveletfamilies or waveletfamilies('f') displays the names of all available wavelet families.

waveletfamilies('n') displays the names of all available wavelets in each family.

waveletfamilies('a') displays all available wavelet families with their corresponding properties.

**Examples**

```
waveletfamilies

=====
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-Splinefbsp
Complex Morlet     cmor
=====

waveletfamilies('n')

=====
Haar                haar
```

# waveletfamilies

---

```
=====
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**
=====

```

```

waveletfamilies('a')

```

```

Type of Wavelets
-----

```

```

type = 1  - orthogonal wavelets          (F.I.R.)
type = 2  - biorthogonal wavelets       (F.I.R.)
type = 3  - with scale function
type = 4  - without scale function
type = 5  - complex wavelet.
-----

```

```

-

```

```

-----

```

```

Family Name : Haar

```

# waveletfamilies

---

haar  
1  
no  
no  
dbwavf

-----  
Family Name : Daubechies  
db  
1  
1 2 3 4 5 6 7 8 9 10 \*\*  
integer  
dbwavf

-----  
Family Name : Symlets  
sym  
1  
2 3 4 5 6 7 8 \*\*  
integer  
symwavf

-----  
Family Name : Coiflets  
coif  
1  
1 2 3 4 5  
integer  
coifwavf

-----  
Family Name : BiorSplines  
bior  
2  
1.1 1.3 1.5 2.2 2.4 2.6 2.8 3.1 3.3 3.5 3.7 3.9 4.4 5.5 6.8  
real  
biorwavf

-----  
Family Name : ReverseBior

```
rbio
2
1.1 1.3 1.5 2.2 2.4 2.6 2.8 3.1 3.3 3.5 3.7 3.9 4.4 5.5 6.8
real
rbiowavf
```

```
-----
Family Name : Meyer
```

```
meyr
```

```
3
```

```
no
```

```
no
```

```
meyer
```

```
-8 8
```

```
-----
Family Name : DMeyer
```

```
dmey
```

```
1
```

```
no
```

```
no
```

```
dmey.mat
```

```
-----
Family Name : Gaussian
```

```
gaus
```

```
4
```

```
1 2 3 4 5 6 7 8 **
```

```
integer
```

```
gauswavf
```

```
-5 5
```

```
-----
Family Name : Mexican_hat
```

```
mexh
```

```
4
```

```
no
```

```
no
```

```
mexihat
```

```
-8 8
```

```
-----
Family Name : Morlet
```

# waveletfamilies

---

```
morl
4
no
no
morlet
-8 8
-----
Family Name : Complex Gaussian
cgau
5
1 2 3 4 5 **
integer
cgauwavf
-5 5
-----
Family Name : Shannon
shan
5
1-1.5 1-1 1-0.5 1-0.1 2-3 **
string
shanwavf
-20 20
-----
Family Name : Frequency B-Spline
fbsp
5
1-1-1.5 1-1-1 1-1-0.5 2-1-1 2-1-0.5 2-1-0.1 **
string
fbspwavf
-20 20
-----
Family Name : Complex Morlet
cmor
5
1-1.5 1-1 1-0.5 1-1 1-0.5 1-0.1 **
string
cmorwavf
-8 8
-----
```



## See Also

wavemngr

# wavemenu

---

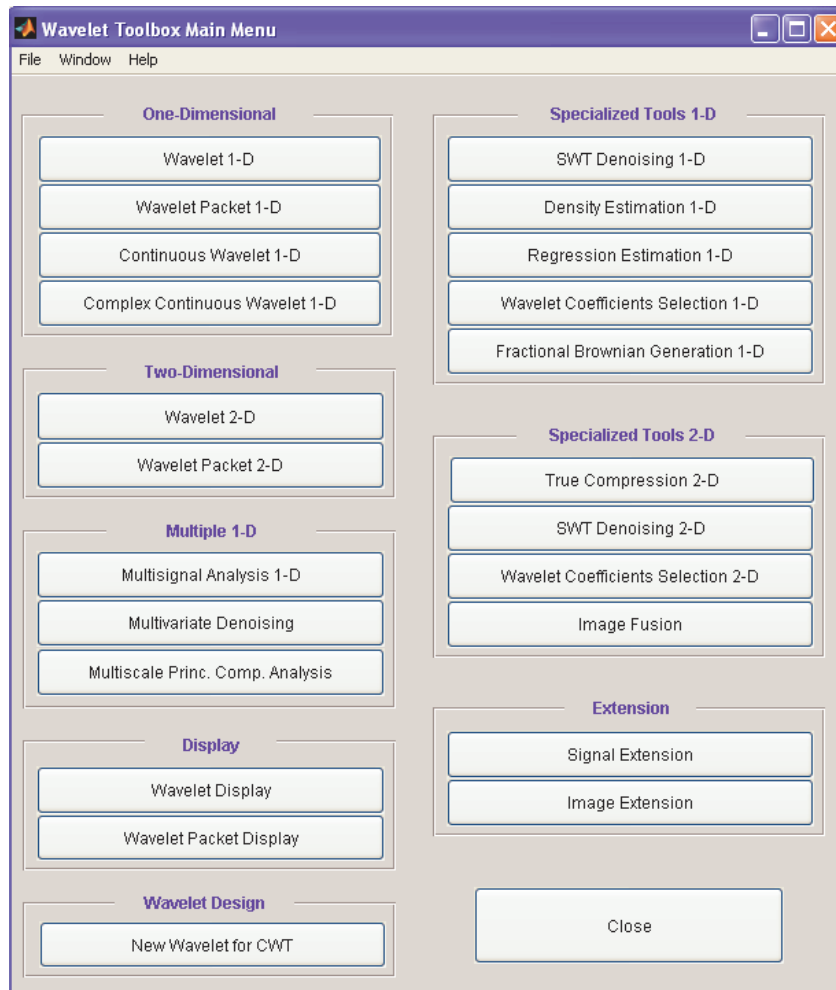
**Purpose** Wavelet Toolbox™ GUI tools

**Syntax** wavemenu

**Description** wavemenu opens a menu for accessing the various graphical tools provided in the Wavelet Toolbox software. For instructions on using these tools see the corresponding chapters in the Wavelet Toolbox User's Guide.

<b>Tools</b>	<b>Chapter</b>
Wavelet 1-D and Wavelet 2-D	Chapter 2, "Using Wavelets"
Wavelet Packet 1-D and Wavelet Packet 2-D	Chapter 5, "Using Wavelet Packets"
Continuous Wavelet 1-D	Chapter 2, "Using Wavelets"
Complex Continuous Wavelet 1-D	Chapter 2, "Using Wavelets"
Wavelet Display and Wavelet Packet Display	Chapter 1, "Wavelets: A New Tool for Signal Analysis"
SWT De-noising 1-D and SWT De-noising 2-D	Chapter 2, "Using Wavelets"
Density Estimation 1-D	Chapter 2, "Using Wavelets"
Regression Estimation 1-D	Chapter 2, "Using Wavelets"
Wavelet Coefficients Selection 1-D	Chapter 2, "Using Wavelets"
Wavelet Coefficients Selection 2-D	Chapter 2, "Using Wavelets"
Signal Extension and Image Extension	Chapter 2, "Using Wavelets"

**Examples** wavemenu



# wavemngr

---

## Purpose

Wavelet manager

## Syntax

```
wavemngr('add',FN,FSN,WT,NUMS,FILE)
wavemngr('add',FN,FSN,WT,NUMS,FILE,B)
wavemngr('del',N)
wavemngr('restore')
wavemngr('restore',IN2)
OUT1 = wavemngr('read')
OUT1 = wavemngr('read',IN2)
OUT1 = wavemngr('read_asc')
```

## Description

wavemngr is a type of wavelets manager. It allows you to add, delete, restore, or read wavelets.

wavemngr('add',FN,FSN,WT,NUMS,FILE) or  
wavemngr('add',FN,FSN,WT,NUMS,FILE,B) or  
wavemngr('add',FN,FSN,WT,{NUMS,TYPNUMS},FILE) or  
wavemngr('add',FN,FSN,WT,{NUMS,TYPNUMS},FILE,B), add a new wavelet family to the toolbox.

FN = Family Name (string)

FSN = Family Short Name (string of length equal or less than four characters)

WT defines the wavelet type:

- WT = 1, for orthogonal wavelets
- WT = 2, for biorthogonal wavelets
- WT = 3, for wavelet with scaling function
- WT = 4, for wavelet without scaling function
- WT = 5, for complex wavelet without scaling function

If the family contains a single wavelet, NUMS = '1'.

Examples:

```
mexh  j
morl
```

If the wavelet is member of a finite family of wavelets, NUMS is a string containing a blank separated list of items representing wavelet parameters.

Example:

```
bior  NUMS = '1.1  1.3  ...  4.4  5.5  6.8'
```

If the wavelet is part of an infinite family of wavelets, NUMS is a string containing a blank separated list of items representing wavelet parameters, terminated by the special sequence `**`.

Examples:

```
db      NUMS = '1  2  3  4  5  6  7  8  9  10  **'
shan   NUMS = '1-1.5  1-1  1-0.5  1-0.1  2-3  **'
```

In these last two cases, TYPNUMS specifies the wavelet parameter input format: 'integer' or 'real' or 'string'; the default value is 'integer'.

Examples:

```
db      TYPNUMS = 'integer'
bior    TYPNUMS = 'real'
shan    TYPNUMS = 'string'
```

FILE = MAT-file or M-file name (string). See usage in the “Examples” section.

B = [lb ub] specifies lower and upper bounds of effective support for wavelets of type = 3, 4, or 5.

This option is fully documented in Chapter 7, “Adding Your Own Wavelets”.

wavemngr('del', N), deletes a wavelet or a wavelet family. N is the Family Short Name or the Wavelet Name (in the family). N is a string.

# wavemngr

---

wavemngr('restore') or wavemngr('restore',IN2) restores previous or initial wavelets. If nargin = 1, the previous wavelets.asc ASCII-file is restored; otherwise the initial wavelets.asc ASCII-file is restored. Here IN2 is a dummy argument.

OUT1 = wavemngr('read') returns all wavelet family names.

OUT1 = wavemngr('read',IN2) returns all wavelet names, IN2 is a dummy argument.

OUT1 = wavemngr('read\_asc') reads wavelets.asc ASCII-file and returns all wavelets information.

## Examples

```
% 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           mor1
=====
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**
=====
```

In the following example, new compactly supported orthogonal wavelets are added to the toolbox. These wavelets, which are a slight generalization of the Daubechies wavelets, are based on the use of Bernstein polynomials and are due to Kateb and Lemarié in an unpublished work.

---

**Note** The M-files used in this example can be found in the `wavedemo` directory.

---



```

% Add new family of orthogonal wavelets.
% You must define:
%
%   Family Name:           Lemarie
%   Family Short Name:    lem
%   Type of wavelet:      1 (orth)
%   Wavelets numbers:    1 2 3 4 5
%   File driver:          lemwavf
%
%   The function lemwavf.m must be as follows:
%   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
=====
% Remove the added family.
wavemngr('del','Lemarie');

% 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
=====
% Restore the previous ascii file
% 'wavelets.prv', then build
% the MAT-file 'wavelets.inf'.
wavemngr('restore');

% List restored wavelets.
wavemngr('read',1)

ans =
=====
Haar             haar
```

```
=====
Daubechies          db
-----
db1  db2  db3  db4
db5  db6  db7  db8
db9  db10 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           mor1
=====
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**
=====
Lemarie         lem
-----
lem1   lem2   lem3   lem4   lem5
=====
% Restore initial wavelets.
%
% Restore the initial ascii file
% 'wavelets.ini' and initial
% MAT-file 'wavelets.bin'.
wavemngr('restore',0);

% List wavelets families.
wavemngr('read')
```

```

ans =
=====
Haar                haar
Daubechies         db
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
=====
% Add new family of orthogonal wavelets.
wavemngr('add','Lemarie','lem',1,'1 2 3','lemwavf');

% All command line capabilities are available for
% the new wavelets.
%
% Example 1: compute the four associated filters.
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters('lem3');

% Example 2: compute scale and wavelet functions.
[phi,psi,xval] = wavefun('lem3');

% Add a new family of orthogonal wavelets: special form
% for the GUI mode.
%
% The M-file lemwavf allows you to compute the filter for
% any order. If you want to get a popup of the form
% 1 2 3 **, associated with the family, then wavelets are
% appended for GUI mode using:

wavemngr('restore',0);
wavemngr('add','Lemarie','lem',1,'1 2 3 **','lemwavf');

```

# wavemngr

---

```
% After this sequence, all GUI capabilities are available for
% the new wavelets.
% Note that the last command allows a short cut in the
% order definition only if possible orders are integers.
```

---

**Caution** wavemngr works on the current directory. If you add a new wavelet family, it is available in this directory only. Refer to Chapter 7, “Adding Your Own Wavelets”.

---

## Limitations

wavemngr allows you to add a new wavelet. You must verify that it is truly a wavelet. No check is performed either about this point or about the type of the new wavelet.

**Purpose** Wavelet names for LWT

**Syntax** `W = wavenames(T)`

**Description** `W = wavenames(T)` returns a cell array that contains the name of all wavelets of type `T`. The valid values for `T` are

- 'all' — all wavelets
- 'lazy' — “lazy” wavelet
- 'orth' — orthogonal wavelets
- 'bior' — biorthogonal wavelets

`W = wavenames` is equivalent to `W = wavenames('all')`.

# waverec

---

**Purpose** Multilevel 1-D wavelet reconstruction

**Syntax**  
`X = waverec(C,L,'wname')`  
`X = waverec(C,L,Lo_R,Hi_R)`

**Description** `waverec` performs a multilevel one-dimensional wavelet reconstruction using either a specific wavelet ('*wname*', see `wfilters`) or specific reconstruction filters (`Lo_R` and `Hi_R`). `waverec` is the inverse function of `wavedec` in the sense that the abstract statement `waverec(wavedec(X,N,'wname'),'wname')` returns `X`.

---

**Note** `waverec` supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets.

---

`X = waverec(C,L,'wname')` reconstructs the signal `X` based on the multilevel wavelet decomposition structure `[C,L]` and wavelet '*wname*'. (For information about the decomposition structure, see `wavedec`.)

`X = waverec(C,L,Lo_R,Hi_R)` reconstructs the signal `X` as above, using the reconstruction filters you specify. `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

Note that `X = waverec(C,L,'wname')` is equivalent to  
`X = appcoef(C,L,'wname',0)`.

## Examples

```
% The current extension mode is zero-padding (see dwtmode).  
  
% Load original one-dimensional signal.  
load leleccum; s = leleccum(1:3920); ls = length(s);  
  
% Perform decomposition of signal at level 3 using db5.  
[c,l] = wavedec(s,3,'db5');  
  
% Reconstruct s from the wavelet decomposition structure [c,l].  
a0 = waverec(c,l,'db5');  
  
% Check for perfect reconstruction.  
err = norm(s-a0)
```



```
err =  
    3.2079e-09
```

**See Also**

appcoef, idwt, wavedec

# waverec2

---

**Purpose** Multilevel 2-D wavelet reconstruction

**Syntax**  
`X = waverec2(C,S,'wname')`  
`X = waverec2(C,S,Lo_R,Hi_R)`

**Description** `waverec2` is a two-dimensional wavelet analysis function.

`X = waverec2(C,S,'wname')` performs a multilevel wavelet reconstruction of the matrix `X` based on the wavelet decomposition structure `[C,S]`. For detailed storage information, see `wavedec2`. `'wname'` is a string containing the name of the wavelet. See `wfilters` for more information.

Instead of specifying the wavelet name, you can specify the filters.

- `X = waverec2(C,S,Lo_R,Hi_R)`, `Lo_R` is the reconstruction low-pass filter
- `Hi_R` is the reconstruction high-pass filter.

`waverec2` is the inverse function of `wavedec2` in the sense that the abstract statement `waverec2(wavedec2(X,N,'wname'),'wname')` returns `X`.

`X = waverec2(C,S,'wname')` is equivalent to `X = appcoef2(C,S,'wname',0)`.

**Remarks** If `C` and `S` are obtained from an indexed image analysis or a truecolor image analysis, `X` is an `m`-by-`n` matrix or an `m`-by-`n`-by-3 array, respectively.

For more information on image formats, see the `image` and `imfinfo` reference pages.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).
% Load original image.
load woman;
% X contains the loaded image.

% Perform decomposition at level 2
% of X using sym4.
[c,s] = wavedec2(X,2,'sym4');

% Reconstruct X from the wavelet
% decomposition structure [c,s].
a0 = waverec2(c,s,'sym4');

% Check for perfect reconstruction.
```

```
max(max(abs(X-a0))  
ans =  
2.5565e-10
```

**See Also**

appcoef2, idwt2, wavedec2

# wbmpen

---

**Purpose** Penalized threshold for wavelet 1-D or 2-D de-noising

**Syntax** THR = wbmpen(C,L,SIGMA,ALPHA)

**Description** THR = wbmpen(C,L,SIGMA,ALPHA) returns global threshold THR for de-noising. THR is obtained by a wavelet coefficients selection rule using a penalization method provided by Birgé-Massart.

[C,L] is the wavelet decomposition structure of the signal or image to be de-noised.

SIGMA is the standard deviation of the zero mean Gaussian white noise in de-noising model (see wnoisest for more information).

ALPHA is a tuning parameter for the penalty term. It must be a real number greater than 1. The sparsity of the wavelet representation of the de-noised signal or image grows with ALPHA. Typically ALPHA = 2.

THR minimizes the penalized criterion given by

let  $t^*$  be the minimizer of

$$\text{crit}(t) = -\sum(c(k)^2, k \leq t) + 2 \cdot \text{SIGMA}^{2 \cdot t} \cdot (\text{ALPHA} + \log(n/t))$$

where  $c(k)$  are the wavelet coefficients sorted in decreasing order of their absolute value and  $n$  is the number of coefficients; then  $\text{THR} = |c(t^*)|$ .

wbmpen(C,L,SIGMA,ALPHA,ARG) computes the global threshold and, in addition, plots three curves:

- $2 \cdot \text{SIGMA}^{2 \cdot t} \cdot (\text{ALPHA} + \log(n/t))$
- $\sum(c(k)^2, k \leq t)$
- $\text{crit}(t)$

## Examples

```
% Example 1: Signal de-noising.
% Load noisy bumps signal.
load noisbump; x = noisbump;

% Perform a wavelet decomposition of the signal
% at level 5 using sym6.
wname = 'sym6'; lev = 5;
[c,l] = wavedec(x,lev,wname);
```

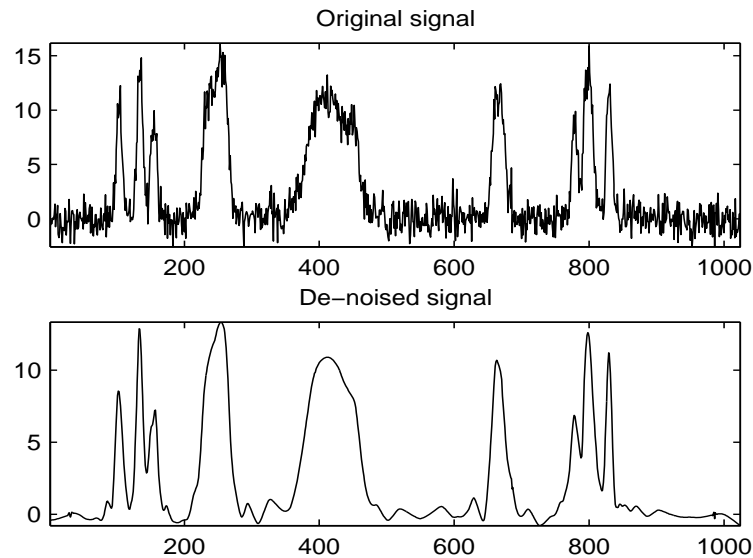
```
% Estimate the noise standard deviation from the
% detail coefficients at level 1, using wnoisest.
sigma = wnoisest(c,l,1);

% Use wbmpen for selecting global threshold
% for signal de-noising, using the tuning parameter.
alpha = 2;
thr = wbmpen(c,l,sigma,alpha)
thr =

    2.7681

% Use wdencomp for de-noising the signal using the above
% threshold with soft thresholding and approximation kept.
keepapp = 1;
xd = wdencomp('gbl',c,l,wname,lev,thr,'s',keepapp);

% Plot original and de-noised signals.
figure(1)
subplot(211), plot(x), title('Original signal')
subplot(212), plot(xd), title('De-noised signal')
```



```
% Example 2: Image de-noising.
% Load original image.
load noiswom;
nbc = size(map,1);

% Perform a wavelet decomposition of the image
% at level 3 using coif2.
wname = 'coif2'; lev = 3;
[c,s] = wavedec2(X,lev,wname);

% Estimate the noise standard deviation from the
% detail coefficients at level 1.
det1 = detcoef2('compact',c,s,1);
sigma = median(abs(det1))/0.6745;

% Use wbmpen for selecting global threshold
% for image de-noising.
alpha = 1.2;
thr = wbmpen(c,l,sigma,alpha)

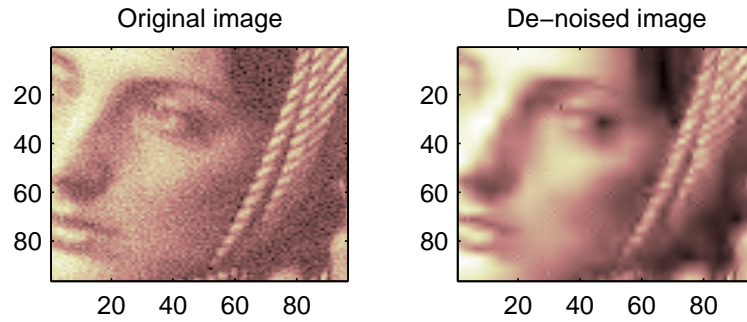
thr =

    36.0621

% Use wdencomp for de-noising the image using the above
% thresholds with soft thresholding and approximation kept.
keepapp = 1;
xd = wdencomp('gbl',c,s,wname,lev,thr,'s',keepapp);

% Plot original and de-noised images.
figure(2)
colormap(pink(nbc));
subplot(221), image(wcodemat(X,nbc))
title('Original image')
```

```
subplot(222), image(wcodemat(xd,nbc))  
title('De-noised image')
```

**See Also**

wden, wdencomp, wbpmpen, wdencomp

# wcodemat

---

**Purpose** Extended pseudocolor matrix scaling

**Syntax**

```
Y = wcodemat(X,NBCODES,OPT,ABSOL)
Y = wcodemat(X,NBCODES,OPT)
Y = wcodemat(X,NBCODES)
Y = wcodemat(X)
```

**Description** wcodemat is a general utility.

`Y = wcodemat(X,NBCODES,OPT,ABSOL)` returns a coded version of input matrix `X` if `ABSOL = 0`, or `ABS(X)` if `ABSOL` is nonzero, using the first `NBCODES` integers. Coding can be done row-wise (`OPT = 'row'` or `'r'`), columnwise (`OPT = 'col'` or `'c'`), or globally (`OPT = 'mat'` or `'m'`).

Coding uses a regular grid between the minimum and the maximum values of each row (column or matrix, respectively).

`Y = wcodemat(X,NBCODES,OPT)` is equivalent to  
`Y = wcodemat(X,NBCODES,OPT,1)`.

`Y = wcodemat(X,NBCODES)` is equivalent to  
`Y = wcodemat(X,NBCODES,'mat',1)`.

`Y = wcodemat(X)` is equivalent to `Y = wcodemat(X,16,'mat',1)`.



**Purpose** True compression of images using wavelets

**Syntax**

```
wcompress('c',X,SAV_FILENAME,COMP_METHOD)
wcompress('c',FILENAME,...)
wcompress('c',I,...)
wcompress(...,'ParName1',ParVal1,'ParName2',ParVal2,...)
[COMPRAT,BPP] = wcompress('c',...)
XC = wcompress('u',SAV_FILENAME)
XC = wcompress('u',SAV_FILENAME,'plot')
XC = wcompress('u',SAV_FILENAME,'step')
```

**Description** The `wcompress` command performs either compression or uncompression of grayscale or truecolor images.

More theoretical information on true compression is in “True Compression for Images” on page 6-130 of the Wavelet Toolbox™ User’s Guide.

### Compression

`wcompress('c',X,SAV_FILENAME,COMP_METHOD)` compresses the image `X` using the compression method `COMP_METHOD`.

The compressed image is saved in the file `SAV_FILENAME`. `X` can be either a 2-D array containing an indexed image or a 3-D array of `uint8` containing a truecolor image.

`wcompress('c',FILENAME,...)` loads the image `X` from the file `FILENAME` which is a MATLAB Supported Format (MSF) file: MAT-file or other image files (see `imread`).

`wcompress('c',I,...)` converts the indexed image `X = I{1}` to a truecolor image `Y` using the colormap `map = I{2}` and then compresses `Y`.

The valid compression methods are divided in three categories.

## 1 Progressive Coefficients Significance Methods (PCSM):

<b>MATLAB Name</b>	<b>Compression Method Name</b>
'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

For more details on these methods, see the references and especially Walker and also Said and Pearlman.

## 2 Coefficients Thresholding Methods (CTM-1):

<b>MATLAB Name</b>	<b>Compression Method Name</b>
'lvl_mmc'	Subband thresholding of coefficients and Huffman encoding

For more details on this method, see the Strang and Nguyen reference.

## 3 Coefficients Thresholding Methods (CTM-2):

<b>MATLAB Name</b>	<b>Compression Method Name</b>
'gbl_mmc_f'	Global thresholding of coefficients and fixed encoding
'gbl_mmc_h'	Global thresholding of coefficients and Huffman encoding

---

**Note** The Discrete Wavelet Transform uses the periodized extension mode. Each of the two dimensions of the image must be a power of 2.

---

All the compression methods use parameters which have default values. You can change these values using the following syntax:

```
wcompress(..., 'ParName1', ParVal1, 'ParName2', ParVal2, ...)
```

Some of the parameters are related to display or to data transform functionalities. The others are linked to the compression process itself.

#### Data transform parameters.

- 'ParName' = 'wname' or 'WNAME' sets the wavelet name.  
ParVal is a string (see `waveletfamilies`). The default for is *bior4.4*
- 'ParName' = 'level' or 'LEVEL' sets the level of decomposition.  
ParVal is an integer such that:  $1 \leq \text{level} \leq \text{levmax}$  which is the maximum possible level (see `wmaxlev`).  
The default level depends on the method:
  - for **PCSM** methods level is equal to `levmax`.
  - for **CTM** methods level is equal to `fix(levmax/2)`.
- 'ParName' = 'it' or 'IT' sets Image type Transform.  
ParVal must be one of the following strings:
  - 'n' : no transformation (default), image type (truecolor or grayscale) is automatically detected.
  - 'g' : grayscale transformation type.
  - 'c' : color transformation type (RGB uint8).
- 'ParName' = 'cc' or 'CC' sets Color Conversion parameter if X is a truecolor image.  
ParVal must be one of the following strings:
  - 'rgb' or 'none' : No conversion (default).
  - 'yuv' : YUV color space transform.
  - 'klt' : Karhunen-Loeve transform.
  - 'yiq' : YIQ color space transform.
  - 'xyz' : CIEXYZ color space transform.

## **Parameter for Progressive Coefficients Significance Methods (PCSM).**

- 'ParName' = 'maxloop' or 'MAXLOOP' sets the maximum number of steps for the compression algorithm.  
ParVal must be a positive integer or Inf (default is 10).

## **Parameters for Coefficients Thresholding Methods (CTM-1).**

Either of the following parameters may be used:

- 'ParName' = 'bpp' or 'BPP' sets the bit-per-pixel ratio.  
ParVal must be such that  $0 \leq \text{ParVal} \leq 8$  (grayscale) or 24 (truecolor).
- 'ParName' = 'comprat' or 'COMPRAT' sets the compression ratio.  
ParVal must be such that  $0 \leq \text{ParVal} \leq 100$ .

## **Parameters for Coefficients Thresholding Methods (CTM-2).**

Two parameters may be used. The first is related to the threshold and the second is the number of classes for quantization.

The first one may be chosen among the five following parameters:

- 'ParName' = 'threshold' or 'THRESHOLD' sets the threshold value for compression.  
ParVal must be a positive (or zero) real number.
- 'ParName' = 'nbcfs' or 'NBCFS' sets the number of preserved coefficients in the wavelet decomposition.  
ParVal must be an integer such that:  $0 \leq \text{ParVal} \leq$  total number of coefficients of wavelet decomposition.
- 'ParName' = 'percfs' or 'PERCFS' sets the percentage of preserved coefficients in the wavelet decomposition.  
ParVal must be a real number such that:  $0 \leq \text{ParVal} \leq 100$ .
- 'ParName' = 'bpp' or 'BPP' sets the bit-per-pixel ratio.  
ParVal must be such that:  $0 \leq \text{ParVal} \leq 8$  (grayscale) or 24 (truecolor)

- 'ParName' = 'comprat' or 'COMPRAT' sets the compression ratio. ParVal must be such that:  $0 \leq \text{ParVal} \leq 100$ .

The second parameter sets the number of classes for quantization:

- 'ParName' = 'nbclas' or 'NBCLAS' sets the number of classes. ParVal must be a real number such that:  $2 \leq \text{ParVal} \leq 200$ .

#### Display parameter.

- 'ParName' = 'plotpar' or 'PLOTPAR' sets the plot parameter. ParVal must be one of the following strings or numbers:  
   'plot' or 0: plots only the compressed image.  
   'step' or 1: displays each step of the encoding process (only for **PCSM** methods).

[COMPRAT,BPP] = wcompress('c',...) returns the compression ratio COMPRAT and the bit\_per\_pixel ratio BPP.

### Uncompression

XC = wcompress('u',SAV\_FILENAME) uncompresses the file SAV\_FILENAME and returns the image XC. Depending on the initial compressed image, XC can be a 2-D array containing either an indexed image or a 3-D array of uint8 containing a truecolor image.

XC = wcompress('u',SAV\_FILENAME,'plot') plots the uncompressed image.

XC = wcompress('u',SAV\_FILENAME,'step') shows the step-by-step uncompression, only for **PCSM** methods.

### Examples

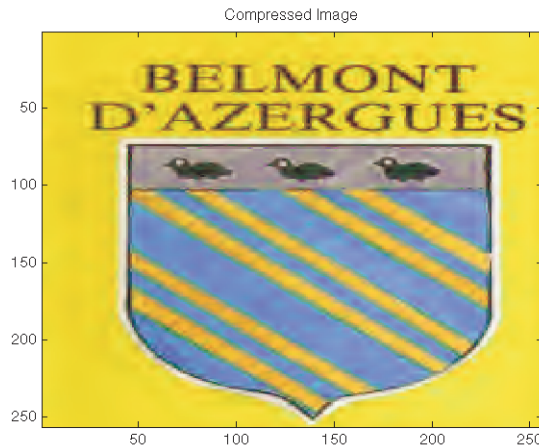
```
% Example 1: Compression and uncompression using
% basic parameters.
%
% This example demonstrates first how to compress the jpeg
% image arms.jpg using the 'stw' compression method and
% save it to the file: 'comp_arms.wtc'.
```

# wcompress

```
wcompress('c','arms.jpg','comp_arms.wtc','stw');
```

```
% Then, it shows how to load the stored image from  
% the file 'comp_arms.wtc' and to display the step by  
% step uncompression leading to the final image below.
```

```
wcompress('u','comp_arms.wtc','step');
```



```
% Example 2: Compression and uncompression using  
% advanced parameters.
```

```
%
```

```
% This example demonstrates how to compress a jpeg  
% image using the 'aswdr' compression method and  
% save it to the file: 'woodstatue.wtc'.
```

```
% During the compression process 3 parameters are used:
```

```
% - Conversion color (cc) set to Karhunen-Loeve transform 'klt'
```

```
% - Maximum number of loops (maxloop) set to 11
```

```
% - Plot type (plotpar) set to step by step display
```

```
% By the way two performance indicators are displayed:
```

```
% the compression ratio (cr) and the bit-per-pixel ratio (bpp).
```

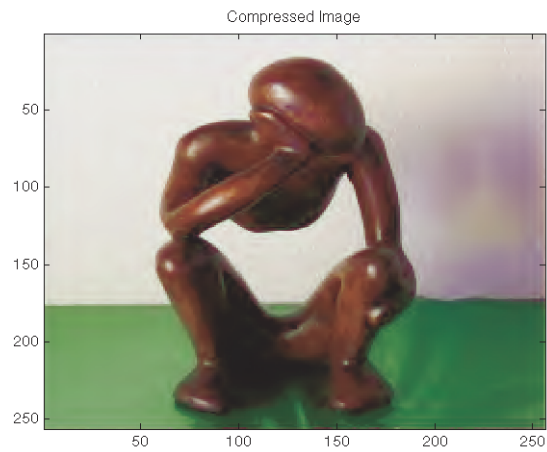
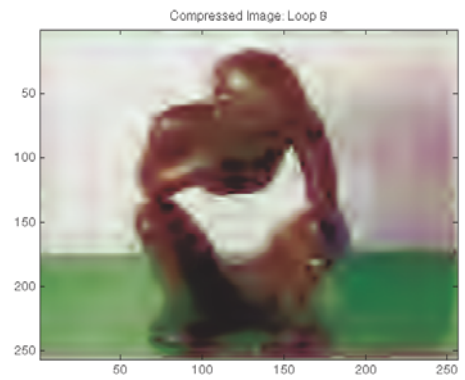
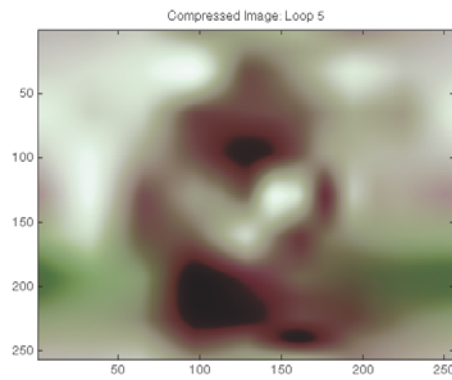
```
[cr,bpp] = wcompress('c','woodstatue.jpg','woodstatue.wtc', ...  
                    'aswdr','cc','klt','maxloop',11,'plotpar','step')
```

```
cr =
```

3.0701

bpp =

0.7368

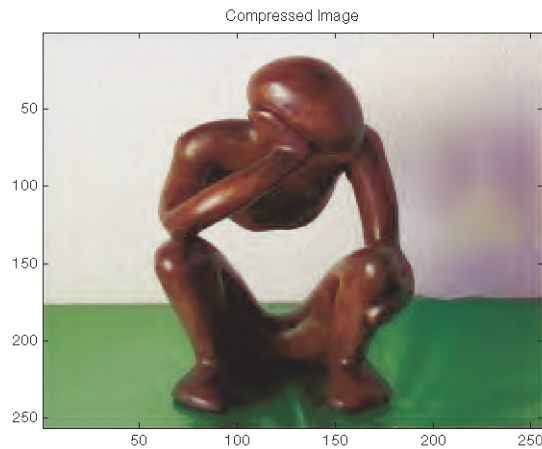


% Then, it shows how to load the stored image from the

## wcompress

```
% file 'woodstatue.wtc' and to display the step by step  
% uncompression process.
```

```
wcompress('u','woodstatue.wtc','step');
```



```
delete('woodstatue.wtc')
```

```
% Example 3: Compression and uncompression of a grayscale image  
% and computed MSE and PSNR error values.
```

```
%
```

```
% Two measures are commonly used to quantify the error between  
% two images: the Mean Square Error(MSE) and the Peak Signal  
% to Noise Ratio (PSNR) which is expressed in decibels.
```

```
%
```

```
% This example demonstrates how to compress the mask image using  
% the 'spiht' compression method and save it to the 'mask.wtc'  
% file.
```

```
load mask;
```

```
[cr,bpp] = wcompress('c',X,'mask.wtc','spiht','maxloop',12)
```

```
cr =  
    2.8336
```

```
bpp =
```

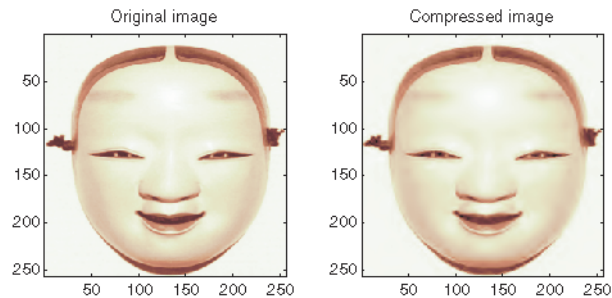


```
0.2267
```

```
% Then, it shows how to load the stored image from the file
% 'mask.wtc', uncompress it and delete the file 'mask.wtc'.
```

```
Xc = wcompress('u','mask.wtc');
delete('mask.wtc')
```

```
% The original and compressed images are displayed.
colormap(pink(255))
subplot(1,2,1); image(X); title('Original image')
axis square
subplot(1,2,2); image(Xc); title('Compressed image')
axis square
```



```
% Finally the MSE and the PSNR are computed.
```

```
D = abs(X-Xc).^2;
mse = sum(D(:))/numel(X)
```

```
mse =
    33.6564
```

```
psnr = 10*log10(255*255/mse)
```

```
psnr =
    32.8601
```

```
% Example 4: Compression and uncompression of a truecolor image
```

# wcompress

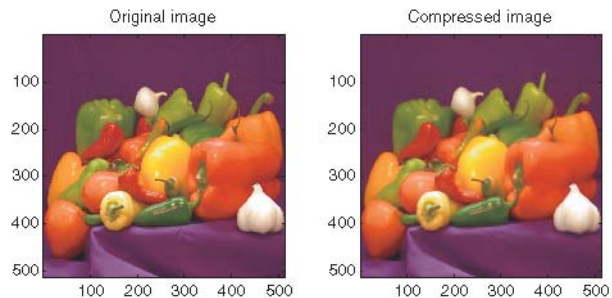
```
% and computed MSE and PSNR error values.  
% Compression parameters are the same as those used for example 3,  
% but using the 'spiht_3d' method give better performance yet.
```

```
X = imread('wpeppers.jpg');  
[cr,bpp] = wcompress('c',X,'wpeppers.wtc','spiht','maxloop',12)
```

```
cr =  
    1.6527
```

```
bpp =  
    0.3966
```

```
Xc = wcompress('u','wpeppers.wtc');  
delete('wpeppers.wtc')  
subplot(1,2,1); image(X); title('Original image'), axis square  
subplot(1,2,2); image(Xc); title('Compressed image'), axis square
```



```
D = abs(double(X)-double(Xc)).^2;  
mse = sum(D(:))/numel(X)
```

```
mse =  
  
    26.7808
```

```
psnr = 10*log10(255*255/mse)
```

```
psnr =  
  
    33.8526
```

**References**

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.

Misiti, M., Y. Misiti, G. Oppenheim, J.-M. Poggi (2007), *Wavelets and their applications*, ISTE DSP Series.

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.

Shapiro J.M. (1993), "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Proc.*, Vol. 41, No. 12, pp. 3445–3462.

Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press.

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*.

**See Also**

fileformats, imread, imwrite, wmaxlev

# wdcbm

---

## Purpose

Thresholds for wavelet 1-D using Birgé-Massart strategy

## Syntax

```
[THR,NKEEP] = wdcbm(C,L,ALPHA)
[THR,NKEEP] = wdcbm(C,L,ALPHA,M)
```

## Description

[THR,NKEEP] = wdcbm(C,L,ALPHA,M) returns level-dependent thresholds THR and numbers of coefficients to be kept NKEEP, for de-noising or compression. THR is obtained using a wavelet coefficients selection rule based on the Birgé-Massart strategy.

[C,L] is the wavelet decomposition structure of the signal to be de-noised or compressed, at level  $j = \text{length}(L) - 2$ . ALPHA and M must be real numbers greater than 1.

THR is a vector of length  $j$ ; THR(i) contains the threshold for level  $i$ .

NKEEP is a vector of length  $j$ ; NKEEP(i) contains the number of coefficients to be kept at level  $i$ .

$j$ , M and ALPHA define the strategy:

- At level  $j+1$  (and coarser levels), everything is kept.
- For level  $i$  from 1 to  $j$ , the  $n_i$  largest coefficients are kept with  $n_i = M (j+2-i)^{\text{ALPHA}}$ .

Typically ALPHA = 1.5 for compression and ALPHA = 3 for de-noising.

A default value for M is  $M = L(1)$ , the number of the coarsest approximation coefficients, since the previous formula leads for  $i = j+1$ , to  $n_{j+1} = M = L(1)$ . Recommended values for M are from  $L(1)$  to  $2^*L(1)$ .

wdcbm(C,L,ALPHA) is equivalent to wdcbm(C,L,ALPHA,L(1)).

## Examples

```
% Load electrical signal and select a part of it.
load leleccum; indx = 2600:3100;
x = leleccum(indx);

% Perform a wavelet decomposition of the signal
% at level 5 using db3.
wname = 'db3'; lev = 5;
[c,l] = wavedec(x,lev,wname);
```

```

% Use wdcbm for selecting level dependent thresholds
% for signal compression using the advised parameters.
alpha = 1.5; m = 1(1);
[thr,nkeep] = wdcbm(c,l,alpha,m)

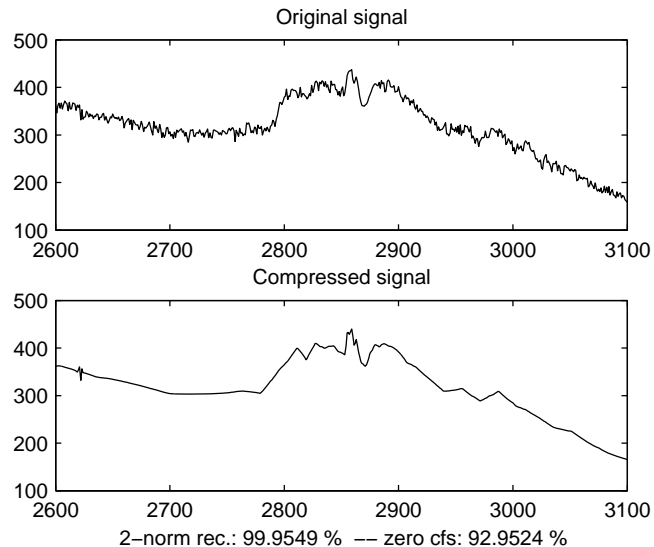
thr =
    19.5569    17.1415    20.2599    42.8959    15.0049

nkeep =
     1     2     3     4     7

% Use wdencomp for compressing the signal using the above
% thresholds with hard thresholding.
[xd,cxd,lxd,perf0,perf12] = ...
    wdencomp('lvd',c,l,wname,lev,thr,'h');

% Plot original and compressed signals.
subplot(211), plot(indx,x), title('Original signal');
subplot(212), plot(indx,xd), title('Compressed signal');
xlabel1 = ['2-norm rec.: ',num2str(perf12)];
xlabel2 = [' % -- zero cfs: ',num2str(perf0), ' %'];
xlabel([xlabel1 xlabel2]);

```



## References

Birgé, L.; P. Massart (1997), “From model selection to adaptive estimation,” in D. Pollard (ed), *Festschrift for L. Le Cam*, Springer, pp. 55–88.

## See Also

wden, wdencmp, wpdencmp

- Purpose** Thresholds for wavelet 2-D using Birgé-Massart strategy
- Syntax** `[ THR, NKEEP ] = wdcbm2(C, S, ALPHA)`  
`[ THR, NKEEP ] = wdcbm2(C, S, ALPHA, M)`
- Description** `[ THR, NKEEP ] = wdcbm2(C, S, ALPHA, M)` returns level-dependent thresholds THR and numbers of coefficients to be kept NKEEP, for de-noising or compression. THR is obtained using a wavelet coefficients selection rule based on the Birgé-Massart strategy.
- `[ C, S ]` is the wavelet decomposition structure of the image to be de-noised or compressed, at level  $j = \text{size}(S, 1) - 2$ .
- ALPHA and M must be real numbers greater than 1.
- THR is a matrix 3 by j; `THR(:, i)` contains the level dependent thresholds in the three orientations: horizontal, diagonal, and vertical, for level i.
- NKEEP is a vector of length j; `NKEEP(i)` contains the number of coefficients to be kept at level i.
- j, M and ALPHA define the strategy:
- At level j+1 (and coarser levels), everything is kept.
  - For level i from 1 to j, the  $n_i$  largest coefficients are kept with  $n_i = M (j+2-i)^{\text{ALPHA}}$ .
- Typically ALPHA = 1.5 for compression and ALPHA = 3 for de-noising.
- A default value for M is  $M = \text{prod}(S(1, :))$ , the length of the coarsest approximation coefficients, since the previous formula leads for  $i = j+1$ , to  $n_{j+1} = M = \text{prod}(S(1, :))$ .
- Recommended values for M are from  $\text{prod}(S(1, :))$  to  $6 * \text{prod}(S(1, :))$ .
- `wdcbm2(C, S, ALPHA)` is equivalent to `wdcbm2(C, S, ALPHA, prod(S(1, :)))`.

**Examples**

```
% Load original image.
load detfingr;
nbc = size(map,1);

% Perform a wavelet decomposition of the image
% at level 3 using sym4.
```

```
wname = 'sym4'; lev = 3;
[c,s] = wavedec2(X,lev,wname);

% Use wdcbm2 for selecting level dependent thresholds
% for image compression using the advised parameters.
alpha = 1.5; m = 2.7*prod(s(1,:));
[thr,nkeep] = wdcbm2(c,s,alpha,m)

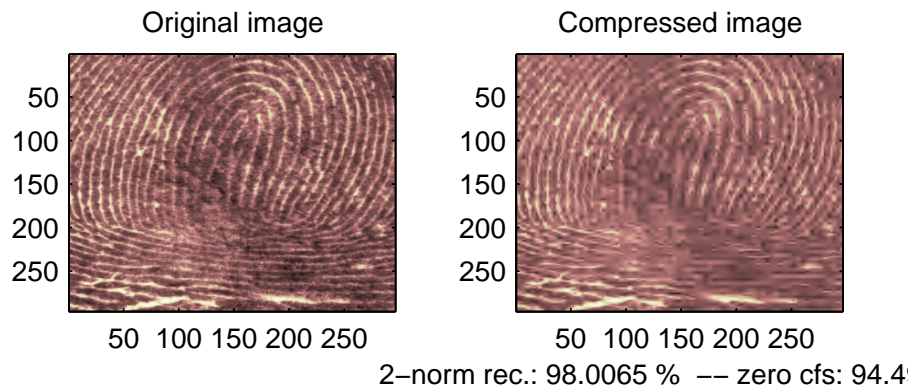
thr =
    21.4814    46.8354    40.7907
    21.4814    46.8354    40.7907
    21.4814    46.8354    40.7907

nkeep =
         624         961        1765

% Use wdencomp for compressing the image using the above
% thresholds with hard thresholding.
[xd,cxd,sxd,perf0,perf12] = ...
    wdencomp('lvd',c,s,wname,lev,thr,'h');

% Plot original and compressed images.
colormap(pink(nbc));
subplot(221), image(wcodemat(X,nbc)),
title('Original image')
subplot(222), image(wcodemat(xd,nbc)),
title('Compressed image')
xlab1 = ['2-norm rec.: ',num2str(perf12)];
xlab2 = [' % -- zero cfs: ',num2str(perf0), ' %'];
xlabel([xlab1 xlab2]);
```



**References**

Birgé, L.; P. Massart (1997). "From model selection to adaptive estimation," in D. Pollard (ed), *Festschrift for L. Le Cam*, Springer, pp. 55–88.

**See Also**

wdencomp, wpcdencomp

**Purpose** Multisignal 1-D decomposition energy distribution

**Syntax**

```
[E,PEC,PECFS] = wdecenergy(DEC)
[E,PEC,PECFS,IDXSORT,LONGS] = wdecenergy(DEC,'sort')
[E,PEC,PECFS] = wdecenergy(DEC,OPTSORT,IDXSIG)
[E,PEC,PECFS,IDXSORT,LONGS] = wdecenergy(DEC,OPTSORT,IDXSIG)
```

**Description** [E,PEC,PECFS] = wdecenergy(DEC) computes the vector E that contains the energy (L2-Norm) of each decomposed signal, the matrix PEC that contains the percentage of energy for each wavelet component (approximation and details) of each signal, and the matrix PECFS that contains the percentage of energy for each coefficient.

- E(i) is the energy (L2-norm) of the ith signal.
- PEC(i,1) is the percentage of energy for the approximation of level MAXLEV = DEC.level of the ith signal.
- PEC(i,j), j=2,...,MAXLEV+1 is the percentage of energy for the detail of level (MAXLEV+1-j) of the ith signal.
- PECFS(i,j), is the percentage of energy for jth coefficients of the ith signal.

[E,PEC,PECFS,IDXSORT,LONGS] = wdecenergy(DEC,'sort') returns PECFS sorted (by row) in ascending order and an index vector IDXSORT.

- Replacing 'sort' by 'ascend' returns the same result.
- Replacing 'sort' by 'descend' returns PECFS sorted in descending order.

LONGS is a vector containing the lengths of each family of coefficients.

[E,PEC,PECFS] = wdecenergy(DEC,OPTSORT,IDXSIG) returns the values for the signals whose indices are given by the IDXSIG vector.

[E,PEC,PECFS,IDXSORT,LONGS] = wdecenergy(DEC,OPTSORT,IDXSIG) returns the values for the signals whose indices are given by the IDXSIG vector, the index vector IDXSORT, and LONGS, which is a vector containing the lengths of each family of coefficients. Valid values for OPSORT are 'none', 'sort', 'ascend', 'descend'.

**Examples**

```
% Load original 1D-multisignal.
load thinker
```

```
% Perform a decomposition at level 2 using wavelet db2.
dec = mdwtdec('r',X,2,'db2');

% Compute the energy distribution.
[E,PEC,PECFS] = wdecenergy(dec);

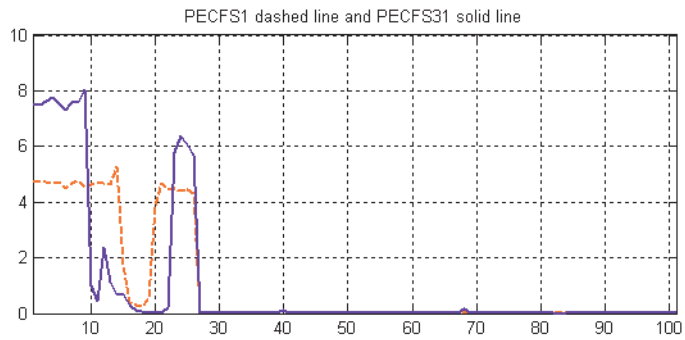
% Display the total energy and the distribution of energy
% for each wavelet component (A2, D2, D1).
E31 = E(31)

E31 =
    2.2411e+006

perA2D2D1 = PEC(31,:)

perA2D2D1 =
    99.3850    0.2926    0.3225

% Compare the coefficient energy distribution
% for signal 1 and signal 31.
PECFS_1 = PECFS(1,:);
PECFS_31 = PECFS(31,:);
figure;
plot(PECFS_1,'r','linewidth',2); hold on
plot(PECFS_31,'b','linewidth',2);
grid; set(gca,'Xlim',[1,size(PECFS,2)])
title('X dashed line and XD solid line')
```



## See Also

`mdwtdec`, `mdwtrec`

**Purpose**

Automatic 1-D de-noising

**Syntax**

```
[XD,CXD,LXD] = wden(X,TPTR,SORH,SCAL,N,'wname')
[XD,CXD,LXD] = wden(C,L,TPTR,SORH,SCAL,N,'wname')
```

**Description**

wden is a one-dimensional de-noising function.

wden performs an automatic de-noising process of a one-dimensional signal using wavelets.

[XD,CXD,LXD] = wden(X,TPTR,SORH,SCAL,N,'wname') returns a de-noised version XD of input signal X obtained by thresholding the wavelet coefficients.

Additional output arguments [CXD,LXD] are the wavelet decomposition structure (see wavedec for more information) of the de-noised signal XD.

TPTR string contains the threshold selection rule:

- 'rigrsure' uses the principle of Stein's Unbiased Risk.
- 'heursure' is an heuristic variant of the first option.
- 'sqtwo log' for universal threshold  $\sqrt{2\log(\cdot)}$
- 'minimaxi' for minimax thresholding (see thselect for more information)

SORH ('s' or 'h') is for soft or hard thresholding (see wthresh for more information).

SCAL defines multiplicative threshold rescaling:

'one' for no rescaling

's1n' for rescaling using a single estimation of level noise based on first-level coefficients

'm1n' for rescaling done using level-dependent estimation of level noise

Wavelet decomposition is performed at level N and 'wname' is a string containing the name of the desired orthogonal wavelet (see wmaxlev and wfilters for more information).

[XD,CXD,LXD] = wden(C,L,TPTR,SORH,SCAL,N,'wname') returns the same output arguments, using the same options as above, but obtained directly from the input wavelet decomposition structure [C,L] of the signal to be de-noised, at level N and using 'wname' orthogonal wavelet.

The underlying model for the noisy signal is basically of the following form:

$$s(n) = f(n) + \sigma e(n)$$

where time  $n$  is equally spaced.

In the simplest model, suppose that  $e(n)$  is a Gaussian white noise  $N(0,1)$  and the noise level  $\sigma$  is supposed to be equal to 1.

The de-noising objective is to suppress the noise part of the signal  $s$  and to recover  $f$ .

The de-noising procedure proceeds in three steps:

- 1** Decomposition. Choose a wavelet, and choose a level  $N$ . Compute the wavelet decomposition of the signal  $s$  at level  $N$ .
- 2** Detail coefficients thresholding. For each level from 1 to  $N$ , select a threshold and apply soft thresholding to the detail coefficients.
- 3** Reconstruction. Compute wavelet reconstruction based on the original approximation coefficients of level  $N$  and the modified detail coefficients of levels from 1 to  $N$ .

More details about threshold selection rules are in “De-noising” of Chapter 6, “Advanced Concepts”, in the User’s Guide, and in the help of the `thselect` function. Let us point out that

- The detail coefficients vector is the superposition of the coefficients of  $f$  and the coefficients of  $e$ , and that the decomposition of  $e$  leads to detail coefficients that are standard Gaussian white noises.
- Minimax and SURE threshold selection rules are more conservative and are more convenient when small details of function  $f$  lie in the noise range. The two other rules remove the noise more efficiently. The option `'heursure'` is a compromise.

In practice, the basic model cannot be used directly. This section examines the options available, to deal with model deviations. The remaining parameter `scal` has to be specified. It corresponds to threshold rescaling methods.

- Option `scal = 'one'` corresponds to the basic model.
- In general, you can ignore the noise level that must be estimated. The detail coefficients  $CD_1$  (the finest scale) are essentially noise coefficients with

standard deviation equal to  $\sigma$ . The median absolute deviation of the coefficients is a robust estimate of  $\sigma$ . The use of a robust estimate is crucial because if level 1 coefficients contain  $f$  details, these details are concentrated in few coefficients to avoid signal end effects, which are pure artifacts due to computations on the edges.

The option `scal = 's1n'` handles threshold rescaling using a single estimation of level noise based on the first-level coefficients.

- When you suspect a nonwhite noise  $e$ , thresholds must be rescaled by a level-dependent estimation of the level noise. The same kind of strategy is used by estimating  $\sigma_{lev}$  level by level. This estimation is implemented in the M-file `wnoisest`, which handles the wavelet decomposition structure of the original signal  $s$  directly.

The option `scal = 'm1n'` handles threshold rescaling using a level-dependent estimation of the level noise.

## Examples

```
% The current extension mode is zero-padding (see dwtmode).

% Set signal to noise ratio and set rand seed.
snr = 3; init = 2055615866;

% Generate original signal and a noisy version adding
% a standard Gaussian white noise.
[xref,x] = wnoise(3,11,snr,init);

% De-noise noisy signal using soft heuristic SURE thresholding
% and scaled noise option, on detail coefficients obtained
% from the decomposition of x, at level 5 by sym8 wavelet.
lev = 5;
xd = wden(x,'heursure','s','one',lev,'sym8');

% Plot signals.
subplot(611), plot(xref), axis([1 2048 -10 10]);
title('Original signal');
subplot(612), plot(x), axis([1 2048 -10 10]);
title(['Noisy signal - Signal to noise ratio = ',...
num2str(fix(snr))]);
subplot(613), plot(xd), axis([1 2048 -10 10]);
title('De-noised signal - heuristic SURE');
```

```
% De-noise noisy signal using soft SURE thresholding
xd = wden(x,'heursure','s','one',lev,'sym8');

% Plot signal.
subplot(614), plot(xd), axis([1 2048 -10 10]);
title('De-noised signal - SURE');

% De-noise noisy signal using fixed form threshold with
% a single level estimation of noise standard deviation.
xd = wden(x,'sqrtwolog','s','sln',lev,'sym8');

% Plot signal.
subplot(615), plot(xd), axis([1 2048 -10 10]);
title('De-noised signal - Fixed form threshold');

% De-noise noisy signal using minimax threshold with
% a multiple level estimation of noise standard deviation.
xd = wden(x,'minimaxi','s','sln',lev,'sym8');

% Plot signal.
subplot(616), plot(xd), axis([1 2048 -10 10]);
title('De-noised signal - Minimax');

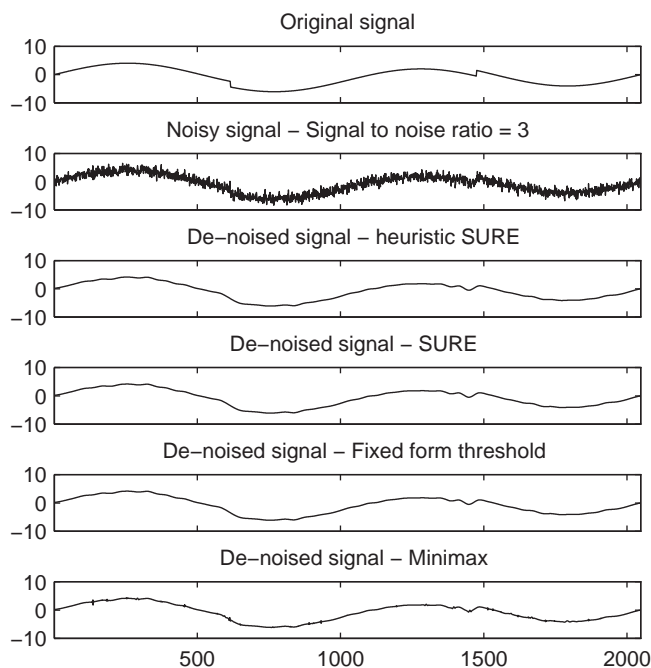
% If many trials are necessary, it is better to perform
% decomposition once and threshold it many times:

% decomposition.
[c,l] = wavedec(x,lev,'sym8');

% threshold the decomposition structure [c,l].
xd = wden(c,l,'minimaxi','s','sln',lev,'sym8');

% Editing some graphical properties,
% the following figure is generated.
```





## References

- Antoniadis, A.; G. Oppenheim, Eds. (1995), *Wavelets and statistics*, 103, Lecture Notes in Statistics, Springer Verlag.
- 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.
- Donoho, D.L.; I.M. Johnstone (1994), "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, Vol. 81, pp. 425–455.
- Donoho, D.L. (1995), "De-noising by soft-thresholding," *IEEE Trans. on Inf. Theory*, 42 3, pp. 613– 627.
- Donoho, D.L.; I.M. Johnstone, G. Kerkyacharian, D. Picard (1995), "Wavelet shrinkage: asymptotia," *Jour. Roy. Stat. Soc., series B*, Vol. 57, No. 2, pp. 301–369.

## See Also

thselect, wavedec, wdencomp wfilters, wthresh

# wdencmp

---

**Purpose** De-noising or compression

**Syntax**

```
[XC,CXC,LXC,PERFO,PERFL2] =  
    wdencmp('gb1',X,'wname',N,THR,SORH,KEEPAPP)  
[XC,CXC,LXC,PERFO,PERFL2] = wdencmp('lvd',X,'wname',N,THR,SORH)  
[XC,CXC,LXC,PERFO,PERFL2] = wdencmp('lvd',C,L,'wname',N,THR,SORH)
```

**Description** wdencmp is a one- or two-dimensional de-noising and compression-oriented function.

wdencmp performs a de-noising or compression process of a signal or an image, using wavelets.

[XC,CXC,LXC,PERFO,PERFL2] = wdencmp('gb1',X,'wname',N,THR,SORH,KEEPAPP) returns a de-noised or compressed version XC of input signal X (one- or two-dimensional) obtained by wavelet coefficients thresholding using global positive threshold THR.

Additional output arguments [CXC,LXC] are the wavelet decomposition structure of XC (see wavedec or wavedec2 for more information). PERFO and PERFL2 are  $L^2$ -norm recovery and compression score in percentage.

$PERFL2 = 100 * (\text{vector-norm of CXC} / \text{vector-norm of C})^2$  if [C,L] denotes the wavelet decomposition structure of X.

If X is a one-dimensional signal and 'wname' an orthogonal wavelet, PERFL2 is reduced to

$$\frac{100 \|XC\|^2}{\|X\|^2}$$

Wavelet decomposition is performed at level N and 'wname' is a string containing wavelet name (see wmaxlev and wfilters for more information). SORH ('s' or 'h') is for soft or hard thresholding (see wthresh for more information). If KEEPAPP = 1, approximation coefficients cannot be thresholded, otherwise it is possible.

wdencmp('gb1',C,L,'wname',N,THR,SORH,KEEPAPP) has the same output arguments, using the same options as above, but obtained directly from the input wavelet decomposition structure [C,L] of the signal to be de-noised or compressed, at level N and using 'wname' wavelet.

For the one-dimensional case and 'lvd' option,

[XC,CXC,LXC,PERFO,PERFL2] = wdencmp('lvd',X,'wname',N,THR,SORH) or

[XC,CXC,LXC,PERFO,PERFL2] = wdencmp('lvd',C,L,'wname',N,THR,SORH)

have the same output arguments, using the same options as above, but allowing level-dependent thresholds contained in vector THR (THR must be of length N). In addition, the approximation is kept. Note that, with respect to wden (automatic de-noising), wdencmp allows more flexibility and you can implement your own de-noising strategy.

For the two-dimensional case and 'lvd' option,

[XC,CXC,LXC,PERFO,PERFL2] = wdencmp('lvd',X,'wname',N,THR,SORH) or

[XC,CXC,LXC,PERFO,PERFL2] = wdencmp('lvd',C,L,'wname',N,THR,SORH).

THR must be a matrix 3 by N containing the level-dependent thresholds in the three orientations, horizontal, diagonal, and vertical.

Ideas for de-noising are in Chapter 2, “Using Wavelets”, of the Wavelet Toolbox User’s Guide, and in the “Description” section of the wden function.

The compression features of a given wavelet basis are primarily linked to the relative scarceness of the wavelet domain representation for the signal. The notion behind compression is based on the concept that the regular signal component can be accurately approximated using a small number of approximation coefficients (at a suitably selected level) and some of the detail coefficients.

Like de-noising, the compression procedure contains three steps:

- 1 Decomposition.
- 2 Detail coefficient thresholding. For each level from 1 to N, a threshold is selected and hard thresholding is applied to the detail coefficients.
- 3 Reconstruction.

The difference with the de-noising procedure is found in step 2.

# wdencmp

## Examples

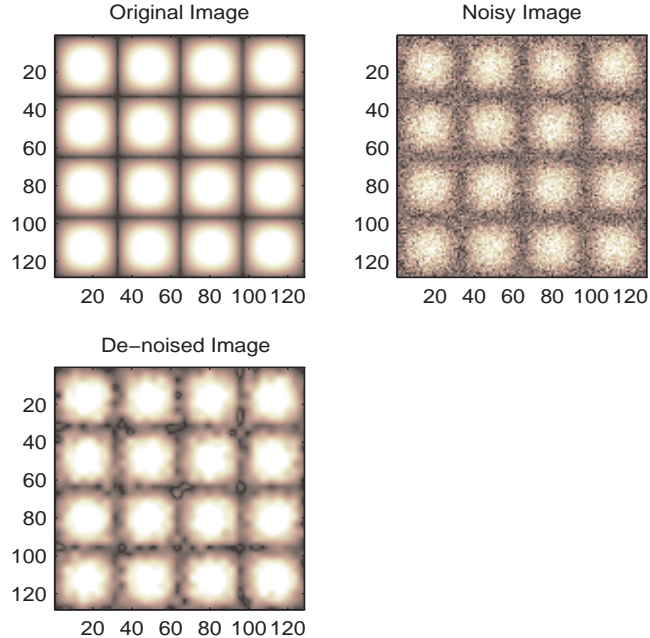
```
% The current extension mode is zero-padding (see dwtmode).

% Example 1: Use wdencmp for image de-noising.
%
% Load original image.
load sinsin
% X contains the loaded image.

% Generate noisy image.
init=2055615866; randn('seed',init);
x = X + 18*randn(size(X));

% find default values (see ddencomp).
[thr,sorh,keepapp] = ddencomp('den','wv',x);
% de-noise image using global thresholding option.
xd = wdencmp('gbl',x,'sym4',2,thr,sorh,keepapp);

% Using some plotting commands,
% the following figure is generated.
```



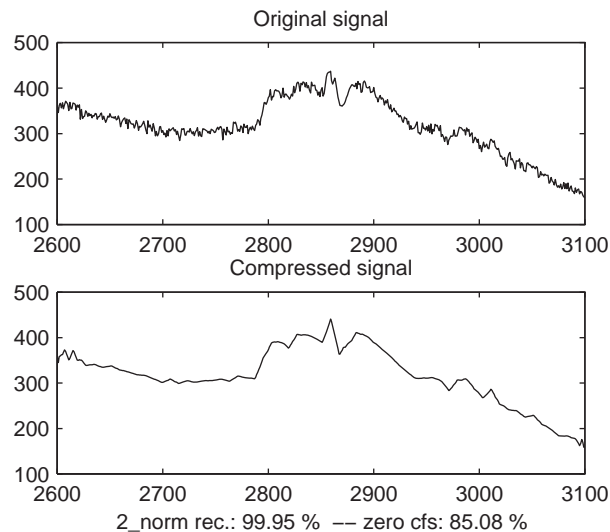
```

% Example 2: Use wdencmp for signal compression.
%
% Load electrical signal and select a part.
load leleccum; indx = 2600:3100;
x = leleccum(indx);

% Compress using a fixed threshold.
thr=35;
[xd,cxd,lxd,perf0,perf12] = ...
wdencmp('gbl',x,'db3',3,thr,'h',1);

% Using some plotting commands,
% the following figure is generated.

```



```

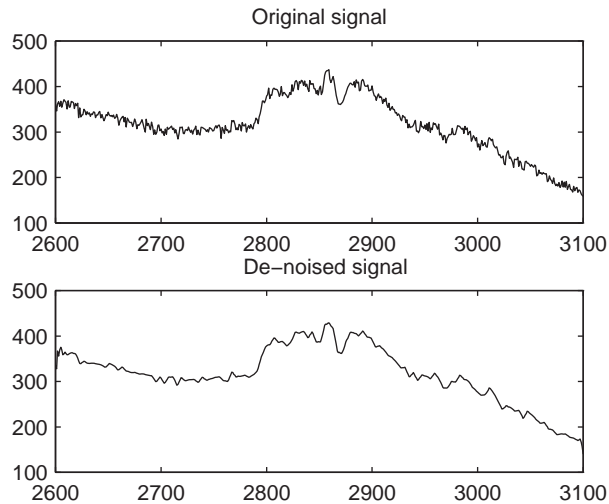
% Example 3: Use wdencmp for signal de-noising.
%
% Find default values (see ddencmp).
[thr,sorh,keepapp] = ddencmp('den','wv',x);

% De-noise signal using global thresholding option.
xd = wdencmp('gbl',x,'db3',2,thr,sorh,keepapp);

```

# wdencmp

```
% Using some plotting commands,  
% the following figure is generated.
```



```
% Example 4: Use wdencmp for image compression.  
%
```

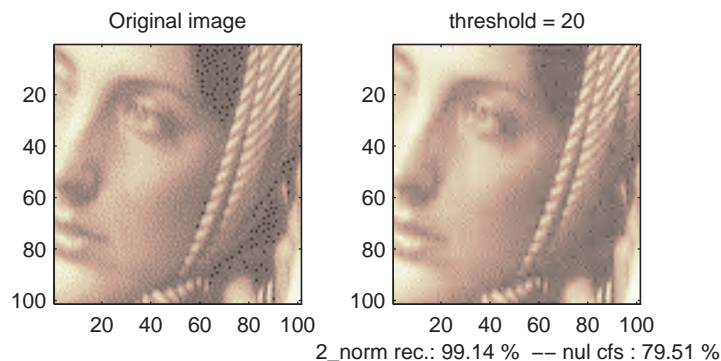
```
% Load original image.  
load woman;  
% X contains the loaded image.
```

```
x=X(100:200,100:200);  
nbc = size(map,1);
```

```
% Wavelet decomposition of x.  
n = 5; w = 'sym2';  
[c,l] = wavedec2(x,n,w);
```

```
% Wavelet coefficients thresholding.  
thr=20;  
[xd,cxd,lxd,perf0,perf12] = ...  
wdencmp('gbl',c,l,w,n,thr,'h',1);
```

```
% Using some plotting commands,  
% the following figure is generated.
```



```
% In addition the first option allows level and orientation-
% dependent thresholds. In this case the approximation is kept.
% The level-dependent thresholds in the three orientations
% horizontal, diagonal and vertical are as follows:
thr_h = [17 18]; % Horizontal thresholds.
thr_d = [19 20]; % Diagonal thresholds.
thr_v = [21 22]; % Vertical thresholds.

thr = [thr_h ; thr_d ; thr_v]

thr =
    17 18
    19 20
    21 22
[xd,cxd,lxd,perf0,perf12] = ...
wdencmp('lvd',x,'sym8',2,thr,'h');
```

**References**

DeVore, R.A.; B. Jawerth, B.J. Lucier (1992), "Image compression through wavelet transform coding," *IEEE Trans. on Inf. Theory*, vol. 38, No 2, pp. 719–746.

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.

Donoho, D.L.; I.M. Johnstone (1994), "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, pp. 425–455.

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.

Donoho, D.L.; I.M. Johnstone, “Ideal de-noising in an orthonormal basis chosen from a library of bases,” *C.R.A.S. Paris*, t. 319, Ser. I, pp. 1317–1322.

Donoho, D.L. (1995), “De-noising by soft-thresholding,” *IEEE Trans. on Inf. Theory*, 41, 3, pp. 613–627.

## See Also

ddencomp, wavedec, wavedec2, wcompress, wdcbm2, wden, wbmpen, wdencomp, wthresh





# wenergy

---

**Purpose** Energy for 1-D wavelet or wavelet packet decomposition

**Syntax** [Ea,Ed] = wenergy(C,L)  
E = wenergy(T)

**Description** For a one-dimensional wavelet decomposition [C,L] (see wavedec for details), [Ea,Ed] = wenergy(C,L) returns Ea, which is the percentage of energy corresponding to the approximation and Ed, which is the vector containing the percentages of energy corresponding to the details.

For a wavelet packet tree T (see wptree, wpdec, wpdec2), E = wenergy(T) returns a vector E, which contains the percentages of energy corresponding to the terminal nodes of the tree T. In this case, wenergy is a method of the wptree object T, which overloads the previous wenergy function.

## Examples

```
% Example 1: 1-D wavelet decomposition
%-----
load noisbump
[C,L] = wavedec(noisbump,4,'sym4');
[Ea,Ed] = wenergy(C,L)
```

```
Ea =
```

```
88.2860
```

```
Ed =
```

```
2.1560    1.2286    1.4664    6.8630
```

```
% Example 2: 1-D wavelet packet decomposition
%-----
load noisbump
T = wpdec(noisbump,3,'sym4');
E = wenergy(T)
```

```
E =
```

```
95.0329    1.4664    0.6100    0.6408    0.5935    0.5445    0.5154
0.5965
```

**Purpose** Energy for 2-D wavelet decomposition

**Syntax** [Ea,Eh,Ev,Ed] = wenergy2(C,S)  
[Ea,EDetail] = wenergy2(C,S)

**Description** For a two-dimensional wavelet decomposition [C,S] (see wavedec2 for details), [Ea,Eh,Ev,Ed] = wenergy2(C,S) returns Ea, which is the percentage of energy corresponding to the approximation, and vectors Eh, Ev, Ed, which contain the percentages of energy corresponding to the horizontal, vertical, and diagonal details, respectively.

[Ea,EDetail] = wenergy2(C,S) returns Ea, and EDetail, which is the sum of vectors Eh, Ev, and Ed.

**Examples**

```
load detail
[C,S] = wavedec2(X,2,'sym4');
[Ea,Eh,Ev,Ed] = wenergy2(C,S)

Ea =
    89.3520

Eh =
    1.8748    2.7360

Ev =
    1.5860    2.6042

Ed =
    0.7539    1.0932

[Ea,EDetails] = wenergy2(C,S)

Ea =
    89.3520

EDetails =
    4.2147    6.4334
```

# wentropy

---

**Purpose** Entropy (wavelet packet)

**Syntax**  $E = \text{wentropy}(X, T, P)$   
 $E = \text{wentropy}(X, T)$

**Description**  $E = \text{wentropy}(X, T, P)$  returns the entropy  $E$  of the vector or matrix input  $X$ . In both cases, output  $E$  is a real number.

$E = \text{wentropy}(X, T)$  is equivalent to  $E = \text{wentropy}(X, T, 0)$ .

$T$  is a string containing the type of entropy and  $P$  is an optional parameter depending on the value of  $T$ .

Entropy Type Name (T)	Parameter (P)	Comments
'shannon'		P is not used.
'log energy'		P is not used.
'threshold'	$0 \leq P$	P is the threshold.
'sure'	$0 \leq P$	P is the threshold.
'norm'	$1 \leq P$	P is the power.
'user'	string	P is a string containing the M-file name of your own entropy function, with a single input $X$ .
FunName	No constraints on P	FunName is any other string except those used for the previous Entropy Type Names listed above. FunName contains the M-file name of your own entropy function, with $X$ as input and $P$ as additional parameter to your entropy function.

**Note** The 'user' option is historical and still kept for compatibility, but it is obsoleted by the last option described in the table above. The FunName option do the same as the 'user' option and in addition gives the possibility to pass a parameter to your own entropy function.

Functionals verifying an additive-type property are well suited for efficient searching of binary-tree structures and the fundamental splitting property of the wavelet packets decomposition. Classical entropy-based criteria match these conditions and describe information-related properties for an accurate representation of a given signal. Entropy is a common concept in many fields, mainly in signal processing. The following example lists different entropy criteria. Many others are available and can be easily integrated. In the following expressions,  $s$  is the signal and  $(s_i)_i$  the coefficients of  $s$  in an orthonormal basis.

The entropy  $E$  must be an additive cost function such that  $E(0) = 0$  and

$$E(s) = \sum_i E(s_i)$$

- The (nonnormalized) Shannon entropy.

$$E1(s_i) = s_i^2 \log(s_i^2) \quad \text{so} \quad E1(s) = -\sum_i s_i^2 \log(s_i^2)$$

with the convention  $0 \log(0) = 0$ .

- The concentration in  $l^p$  norm entropy with  $1 \leq p$ .

$$E2(s_i) = |s_i|^p \quad \text{so} \quad E2(s) = \sum_i |s_i|^p = \|s\|_p^p$$

- The “log energy” entropy.

$$E3(s_i) = \log(s_i^2) \quad \text{so} \quad E3(s) = \sum_i \log(s_i^2)$$

with the convention  $\log(0) = 0$ .

- The threshold entropy.

$E4(s_i) = 1$  if  $|s_i| > p$  and 0 elsewhere so  $E4(s) = \#\{i \text{ such that } |s_i| > p\}$  is the number of time instants when the signal is greater than a threshold  $p$ .

- The “SURE” entropy.

$$E5(s) = n - \#\{i \text{ such that } |s_i| \leq p\} + \sum_i \min(s_i^2, p^2)$$

For more information, see the section “Wavelet Packets for Compression and De-Noising” in Chapter 6 of the User’s Guide.

## Examples

```
% The current extension mode is zero-padding (see dwtmode).

% Generate initial signal.
x = randn(1,200);

% Compute Shannon entropy of x.
e = wentropy(x,'shannon')

e =
    -142.7607

% Compute log energy entropy of x.
e = wentropy(x,'log energy')
e =
    -281.8975

% Compute threshold entropy of x
% with threshold equal to 0.2.
e = wentropy(x,'threshold',0.2)
e =
    162

% Compute Sure entropy of x
% with threshold equal to 3.
e = wentropy(x,'sure',3)
e =
    -0.6575
```

```
% Compute norm entropy of x with power equal to 1.1.
e = wentropy(x,'norm',1.1)
e =
    160.1583

% Compute user entropy of x with a user defined
% function: userent for example.
% This function must be an M-file, with first line
% of the following form:
%
%     function e = userent(x)
%
% where x is a vector and e is a real number.
% Then a new entropy is defined and can be used typing:
%
% e = wentropy(x,'user','userent')
%
% or more directly
%
% e = wentropy(x,'userent')
```

## References

- 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.
- Donoho, D.L.; I.M. Johnstone, "Ideal de-noising in an orthonormal basis chosen from a library of bases," *C.R.A.S. Paris, Ser. I*, t. 319, pp. 1317–1322.

# wextend

---

**Purpose** Extend vector or matrix

**Syntax** Y = wextend(TYPE,MODE,X,L,LOC)  
Y = wextend(TYPE,MODE,X,L)

**Description** The valid extension types (TYPE) are listed in the table below.

TYPE	Description
1, '1', '1d' or '1D'	1-D extension
2, '2', '2d' or '2D'	2-D extension
'ar' or 'addrow'	Add rows
'ac' or 'addcol'	Add columns



The valid extension modes (MODE) are listed in the table below.

MODE	Description
'zpd'	Zero extension
'sp0'	Smooth extension of order 0
'spd' (or 'sp1')	Smooth extension of order 1
'sym' or 'symh'	Symmetric-padding (half-point): boundary value symmetric replication
'symw'	Symmetric-padding (whole-point): boundary value symmetric replication
'asym' or 'asymh'	Antisymmetric-padding (half-point): boundary value antisymmetric replication
'asymw'	Antisymmetric-padding (whole-point): boundary value antisymmetric replication
'ppd'	Periodized extension (1)
'per'	Periodized extension (2): If the signal length is odd, wextend adds an extra sample, equal to the last value, on the right and performs extension using the 'ppd' mode. Otherwise, 'per' reduces to 'ppd'. The same kind of rule stands for images.

With TYPE = {1, 'l', 'ld' or 'lD'}:

- LOC = 'l' (or 'u') for left (or up) extension.
- LOC = 'r' (or 'd') for right (or down) extension.
- LOC = 'b' for extension on both sides.
- LOC = 'n' null extension.
- The default is LOC = 'b'.
- L is the length of the extension.

With TYPE = {'ar', 'addrow'}:

- LOC is a 1D extension location.
- The default is LOC = 'b'.
- L is the number of rows to add.

With TYPE = {'ac', 'addcol'}:

- LOC is a 1D extension location.
- The default is LOC = 'b'.
- L is the number of columns to add.

With TYPE = {2, '2', '2d' or '2D'}:

- LOC = [LOCROW, LOCCOL] where LOCROW and LOCCOL are 1D extension locations or 'n' (none).
- The default is LOC = 'bb'.
- L = [LROW, LCOL] where LROW is the number of rows to add and LCOL is the number of columns to add.

For more information on symmetric extension modes see “References”.

## Examples

```
% Original signal.
x = [1 2 3]

x =

     1     2     3

% 1-D extension length.
l = 2;

% Zero-padding extensions 1-D.
xextzpd1 = wextend('1','zpd',x,l)
xextzpd1 =

     0     0     1     2     3     0     0

xextzpd2 = wextend('1D','zpd',x,l,'b')

xextzpd2 =
```

```

    0    0    1    2    3    0    0

% Symmetric extension 1-D.
xextsym = wextend('1D','sym',x,l)

xextsym =

    2    1    1    2    3    3    2

% Periodic extension 1-D.
xextper = wextend('1D','per',x,l)

xextper =

    3    3    1    2    3    3    1    2

% Original image.
X = [1 2 3;4 5 6]

X =

    1    2    3
    4    5    6

% 2-D extension length.
l = 2;

% Zero-padding extension 2-D.
Xextzpd = wextend(2,'zpd',X,l)

Xextzpd =

    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    1    2    3    0    0
    0    0    4    5    6    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0

% Symmetric extension 2-D.
Xextsym = wextend('2D','sym',X,l)

```

Xextsym =

5	4	4	5	6	6	5
2	1	1	2	3	3	2
2	1	1	2	3	3	2
5	4	4	5	6	6	5
5	4	4	5	6	6	5
2	1	1	2	3	3	2

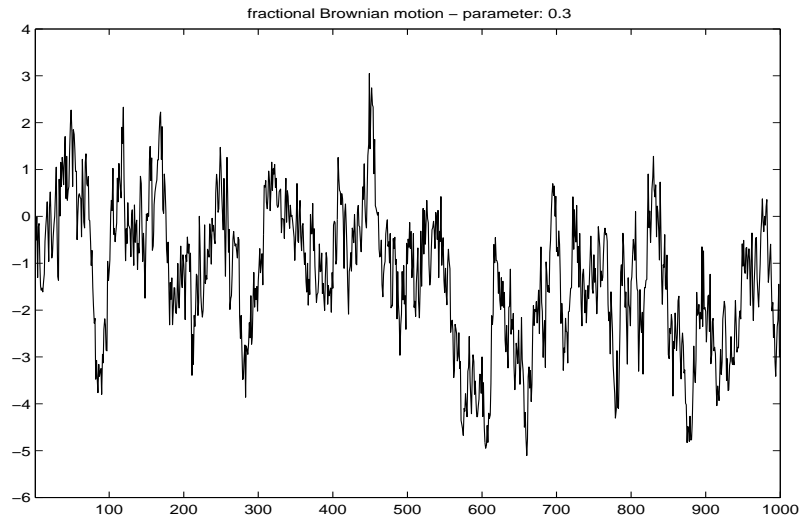
## References

Strang, G.; T. Nguyen (1996), *Wavelets and filter banks*, Wellesley- Cambridge Press.

<b>Purpose</b>	Fractional Brownian motion synthesis
<b>Syntax</b>	<pre>FBM = wfbm(H,L) FBM = wfbm(H,L,'plot') FBM = wfbm(H,L,NS,W), FBM = WFBM(H,L,W,NS) wfbm(H,L,'plot',NS,W), WFBM(H,L,'plot',W,NS)</pre>
<b>Description</b>	<p><code>FBM = wfbm(H,L)</code> returns a fractional Brownian motion signal FBM of the Hurst parameter <math>H</math> (<math>0 &lt; H &lt; 1</math>) and length <math>L</math>, following the algorithm proposed by Abry and Sellan.</p> <p><code>FBM = wfbm(H,L,'plot')</code> generates and plots the FBM signal.</p> <p><code>FBM = wfbm(H,L,NS,W)</code> or <code>FBM = wfbm(H,L,W,NS)</code> returns the FBM using NS reconstruction steps and the sufficiently regular orthogonal wavelet <math>W</math>.</p> <p><code>wfbm(H,L,'plot',NS)</code> or <code>wfbm(H,L,'plot',W)</code> or <code>wfbm(H,L,'plot',NS,W)</code> or <code>wfbm(H,L,'plot',W,NS)</code> generates and plots the FBM signal.</p> <p><code>wfbm(H,L)</code> is equivalent to <code>WFBM(H,L,6,'db10')</code>.</p> <p><code>wfbm(H,L,NS)</code> is equivalent to <code>WFBM(H,L,NS,'db10')</code>.</p> <p><code>wfbm(H,L,W)</code> is equivalent to <code>WFBM(H,L,W,6)</code>.</p> <p>A fractional Brownian motion (fBm) is a continuous-time Gaussian process depending on the Hurst parameter <math>0 &lt; H &lt; 1</math>. It generalizes the ordinary Brownian motion corresponding to <math>H = 0.5</math> and whose derivative is the white noise. The fBm is self-similar in distribution and the variance of the increments is given by</p> $\text{Var}(fBm(t) - fBm(s)) = v  t-s ^{2H}$ <p>where <math>v</math> is a positive constant.</p>
<b>Examples</b>	<p>According to the value of <math>H</math>, the fBm exhibits for <math>H &gt; 0.5</math>, long-range dependence and for <math>H &lt; 0.5</math>, short or intermediate dependence. This example shows each situation using the <code>wfbm</code> M-file, which generates a sample path of this process.</p> <pre>% Generate fBm for H = 0.3 and H = 0.7  % Initialize the randn generator</pre>

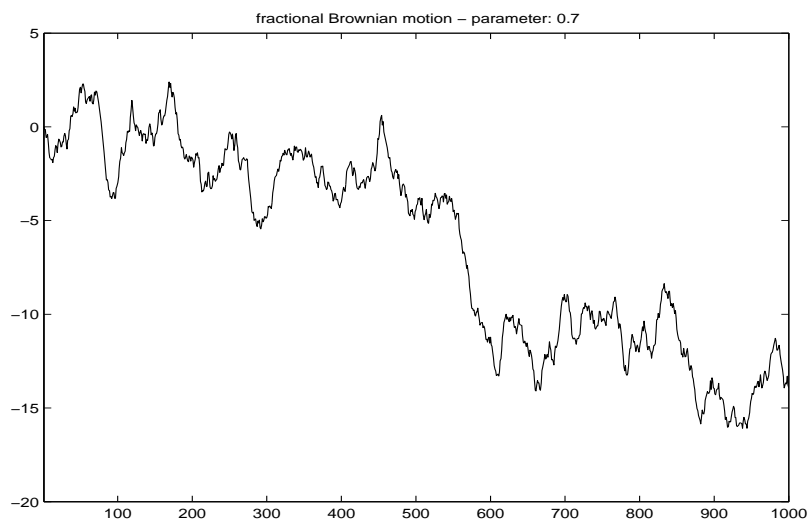
```
randn('state',1)

% Set the parameter H and the sample length
H = 0.3; lg = 1000;
% Generate and plot wavelet-based fBm for H = 0.3
fBm03 = wfbm(H,lg,'plot');
```



```
% Reset randn generator and parameter H
randn('state',1); H = 0.7;
% Generate and plot wavelet-based fBm for H = 0.7
fBm07 = wfbm(H,lg,'plot');

% The last step is equivalent to
% Define wavelet and level of decomposition
% w = 'db10'; ns = 6;
% Generate
% fBm07 = wfbm(H,lg,'plot',w,ns);
```



fBm07 clearly exhibits a stronger low-frequency component and has, locally, less irregular behavior.

## Algorithm

Starting from the expression of the fBm process as a fractional integral of the white noise process, the idea of the algorithm is to build a biorthogonal wavelet depending on a given orthogonal one and adapted to the parameter  $H$ .

Then the generated sample path is obtained by the reconstruction using the new wavelet starting from a wavelet decomposition at a given level designed as follows: details coefficients are independent random Gaussian realizations and approximation coefficients come from a fractional ARIMA process.

This method was first proposed by Meyer and Sellan and implementation issues were examined by Abry and Sellan.

Nevertheless, the samples generated following this original scheme exhibit too many high-frequency components. To circumvent this undesirable behavior Bardet et al. propose downsampling the obtained sample by a factor 10.

Two internal parameters  $\text{delta} = 10$  (the downsampling factor) and a threshold  $\text{prec} = 1E-4$ , to evaluate series by truncated sums, can be modified by the user for extreme values of  $H$ .

A complete overview of long-range dependence process generators is available in Bardet et al.

## References

Abry, P.; F. Sellan (1996), "The wavelet-based synthesis for the fractional Brownian motion proposed by F. Sellan and Y. Meyer: Remarks and fast implementation," *Appl. and Comp. Harmonic Anal.*, 3(4), pp. 377–383.

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, pp. 579–623.

## See Also

wfbmesti



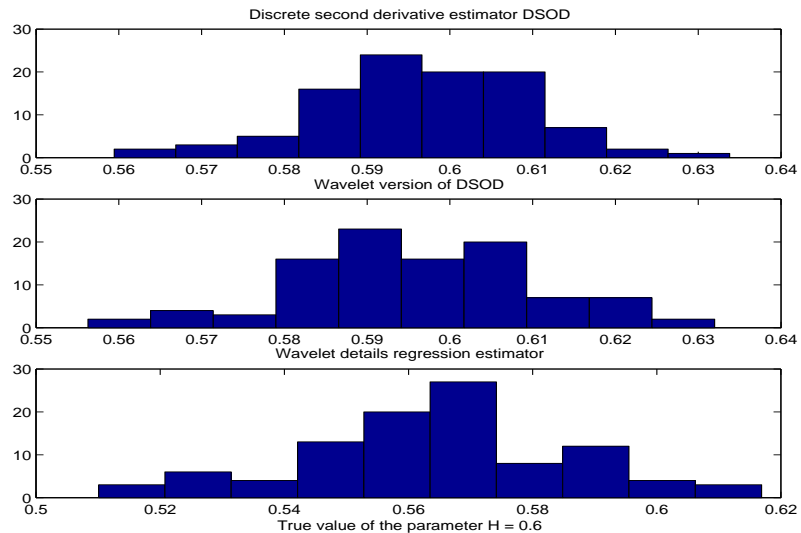
<b>Purpose</b>	Parameter estimation of fractional Brownian motion
<b>Syntax</b>	<code>HEST = wfbmesti(X)</code>
<b>Description</b>	<p><code>HEST = wfbmesti(X)</code> returns a row vector <code>HEST</code> which contains three estimates of the fractal index <math>H</math> of the signal <math>X</math> supposed to come from a fractional Brownian motion of parameter <math>H</math>.</p> <p>The two first estimates are based on second order discrete derivative, the second one is wavelet-based.</p> <p>The third estimate is based on the linear regression in loglog plot, of the variance of detail versus level.</p> <p>A fractional Brownian motion (fBm) is a continuous-time Gaussian process depending on the so-called Hurst parameter <math>0 &lt; H &lt; 1</math>. It generalizes the ordinary Brownian motion corresponding to <math>H = 0.5</math> and whose derivative is the white noise. The fBm is self-similar in distribution and the variance of the increments is given by</p> $\text{Var}(fBm(t) - fBm(s)) = v  t-s ^{(2H)}$ <p>where <math>v</math> is a positive constant.</p> <p>This special form of the variance of the increments suggests various ways to estimate the parameter <math>H</math>. One can find in Bardet et al. a survey of such methods. The <code>wfbmesti</code> M-file provides three different estimates. The first one, due to Istas and Lang, is based on the discrete second-order derivative. The second one is a wavelet-based adaptation and has similar properties. The third one, proposed by Flandrin, estimates <math>H</math> using the slope of the loglog plot of the detail variance versus the level. A more recent extension can be found in Abry et al.</p>
<b>Examples</b>	<p>This example shows a statistical comparison of the three estimators by a short Monte-Carlo study.</p> <pre> % Initialize the randn generator randn('state',1)  % Set parameter H to 0.6 and sample length H = 0.6; lg = 10000; % Generate 100 wavelet-based fBm realizations for H = 0.6 </pre>

```

% and compute the three estimates for each of them
n = 100; Hest = zeros(n,3);
for i = 1:n
    fBm06 = wfbm(H,lg);
    Hest(i,:) = wfbmesti(fBm06);
end

% Compare empirical distributions
subplot(311), hist(Hest(:,1));
title('Discrete second derivative estimator DSOD')
subplot(312), hist(Hest(:,2));
title('Wavelet version of DSOD')
subplot(313), hist(Hest(:,3));
title('Wavelet details regression estimator')
xlabel('True value of the parameter H = 0.6')

```



For these experimental conditions, the two first methods give similar results with smaller dispersion than the third one. The third one is clearly slightly biased and has greater dispersion.

These experimental results depend on  $H$  and on the various experimental conditions. For a complete study, see Bardet et al.

**References**

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.

Bardet, J.-M.; G. Lang, G. Oppenheim, A. Philippe, S. Stoev, M.S. Taqqu (2003), “Semi-parametric estimation of the long-range dependence parameter: a survey,” *Theory and applications of long-range dependence*, Birkhäuser, pp. 557–577.

Flandrin, P. (1992), “Wavelet analysis and synthesis of fractional Brownian motion,” *IEEE Trans. on Inf. Th.*, 38, pp. 910–917.

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.

**See Also**

wfbm

# wfilters

## Purpose

Wavelet filters

## Syntax

```
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters('wname')  
[F1,F2] = wfilters('wname','type')
```

## Description

[Lo\_D,Hi\_D,Lo\_R,Hi\_R] = wfilters('wname') computes four filters associated with the orthogonal or biorthogonal wavelet named in the string 'wname'.

The four output filters are

- Lo\_D, the decomposition low-pass filter
- Hi\_D, the decomposition high-pass filter
- Lo\_R, the reconstruction low-pass filter
- Hi\_R, the reconstruction high-pass filter

Available orthogonal or biorthogonal wavelet names 'wname' are listed in the table below.

Wavelet Families	Wavelets
Daubechies	'db1' or 'haar', 'db2', ... , 'db10', ... , 'db45'
Coiflets	'coif1', ... , 'coif5'
Symlets	'sym2', ... , 'sym8', ... , 'sym45'
Discrete Meyer	'dmey'
Biorthogonal	'bior1.1', 'bior1.3', 'bior1.5' 'bior2.2', 'bior2.4', 'bior2.6', 'bior2.8' 'bior3.1', 'bior3.3', 'bior3.5', 'bior3.7' 'bior3.9', 'bior4.4', 'bior5.5', 'bior6.8'
Reverse Biorthogonal	'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'

[F1,F2] = wfilters('wname','type') returns the following filters:

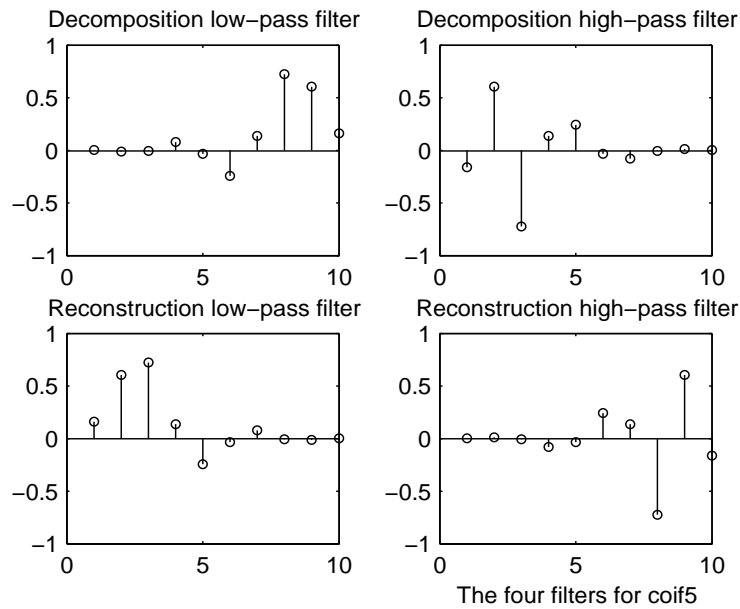
Lo_D and Hi_D	(Decomposition filters)	If 'type' = 'd'
Lo_R and Hi_R	(Reconstruction filters)	If 'type' = 'r'
Lo_D and Lo_R	(Low-pass filters)	If 'type' = 'l'
Hi_D and Hi_R	(High-pass filters)	If 'type' = 'h'

## Examples

```
% Set wavelet name.
wname = 'db5';

% Compute the four filters associated with wavelet name given
% by the input string wname.
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters(wname);
subplot(221); stem(Lo_D);
title('Decomposition low-pass filter');
subplot(222); stem(Hi_D);
title('Decomposition high-pass filter');
subplot(223); stem(Lo_R);
title('Reconstruction low-pass filter');
subplot(224); stem(Hi_R);
title('Reconstruction high-pass filter');
xlabel('The four filters for db5')

% Editing some graphical properties,
% the following figure is generated.
```



## References

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

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.

## See Also

`biorfilt`, `orthfilt`, `waveinfo`

**Purpose** Fusion of two images

**Syntax** `XFUS = wfusing(X1,X2,WNAME,LEVEL,AFUSMETH,DFUSMETH)`  
`[XFUS,TFUS,TX1,TX2] = wfusing(X1,X2,WNAME,LEVEL,AFUSMETH,DFUSMETH)`  
`wfusing(X1,X2,WNAME,LEVEL,AFUSMETH,DFUSMETH,FLAGPLOT)`

**Description** The principle of image fusion using wavelets is to merge the wavelet decompositions of the two original images using fusion methods applied to approximations coefficients and details coefficients (see Zeeuw and Misiti et al.).

`XFUS = wfusing(X1,X2,WNAME,LEVEL,AFUSMETH,DFUSMETH)` returns the fused image `XFUS` obtained by fusion of the two original images `X1` and `X2`. Each fusion method, defined by `AFUSMETH` and `DFUSMETH`, merges in a specific way detailed below, the decompositions of `X1` and `X2`, at level `LEVEL` and using wavelet `WNAME`.

`AFUSMETH` and `DFUSMETH` define the fusion method for approximations and details, respectively.

`[XFUS,TFUS,TX1,TX2] = wfusing(X1,X2,WNAME,LEVEL,AFUSMETH,DFUSMETH)` returns, in addition to matrix `XFUS`, three objects of the class `WDECTREE` associated with `XFUS`, `X1`, and `X2` respectively (see `@WDECTREE`).

`wfusing(X1,X2,WNAME,LEVEL,AFUSMETH,DFUSMETH,FLAGPLOT)` also plots the objects `TFUS`, `TX1`, and `TX2`.

`Fusmeth` denotes `AFUSMETH` or `DFUSMETH`. Available fusion methods are

- Simple — `Fusmeth` can be 'max', 'min', 'mean', 'img1', 'img2' or 'rand', which merges the two approximations or details structures obtained from `X1` and `X2` elementwise by taking the maximum, the minimum, the mean, the first element, the second element, or a randomly chosen element
- Parameter-dependent — `Fusmeth` is of the following form

```
Fusmeth = struct('name',nameMETH,'param',paramMETH)
```

where `nameMETH` can be

```
'linear'
'UD_fusion'      Up-down fusion
'DU_fusion'      Down-up fusion
```

'RL_fusion'	Right-left fusion
'UserDEF'	User-defined fusion

For the description of these options and the paramMETH parameter, see wfusmat.

## Remarks

X1 and X2 must be of same size (see wextend to resize images) and represent indexed images or truecolor images, which are m-by-n matrices or m-by-n-by-3 arrays, respectively.

For more information on image formats, see the image and imfinfo reference pages.

## Examples

The following three examples examine the process of image fusion

- The first example merges two different images leading to a new image
- The second example restores an image from two fuzzy versions of an original image.
- The third example shows how to make an image fusion using a user defined fusion method.

```
% Example 1: Fusion of two different images

% Load two original images: a mask and a bust
load mask; X1 = X;
load bust; X2 = X;

% Merge the two images from wavelet decompositions at level 5
% using db2 by taking two different fusion methods

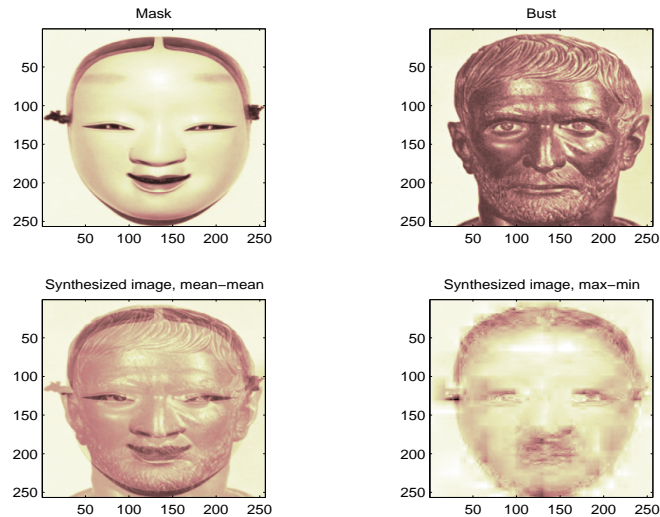
% fusion by taking the mean for both approximations and details
XFUSmean = wfusimg(X1,X2,'db2',5,'mean','mean');

% fusion by taking the maximum for approximations and the
% minimum for the details
XFUSmaxmin = wfusimg(X1,X2,'db2',5,'max','min');

% Plot original and synthesized images
colormap(map);
subplot(221), image(X1), axis square, title('Mask')
```



```
subplot(222), image(X2), axis square, title('Bust')
subplot(223), image(XFUSmean), axis square,
title('Synthesized image, mean-mean')
subplot(224), image(XFUSmaxmin), axis square,
title('Synthesized image, max-min')
```



```
% Example 2: Restoration by fusion of fuzzy images
```

```
% Load two fuzzy versions of an original image
load cathe_1; X1 = X;
load cathe_2; X2 = X;
```

```
% Merge the two images from wavelet decompositions at level 5
% using sym4 by taking the maximum of absolute value of the
% coefficients for both approximations and details
XFUS = wfusing(X1,X2,'sym4',5,'max','max');
```

```
% Plot original and synthesized images
colormap(map);
subplot(221), image(X1), axis square,
title('Catherine 1')
subplot(222), image(X2), axis square,
```

```
title('Catherine 2')  
subplot(223), image(XFUS), axis square,  
title('Synthesized image')
```



```
% The synthesized image is a restored version of good  
% quality of the common underlying original image.
```

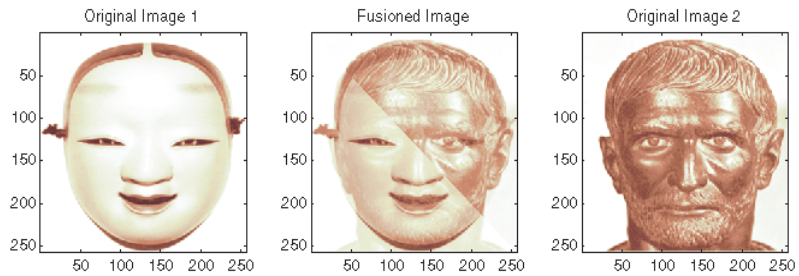
```
% Example 3: Fusion using a user defined fusion method.  
% This example calls a user fusion method defined by the  
% file myfus_FUN.m which is listed below at the end of  
% the example.
```

```
% load two images of the same size.  
load mask; A = X;  
load bust; B = X;
```

```
% Define the fusion method and call the fusion function
```

```
Fus_Method = struct('name','userDEF','param','myfus_FUN');
C = wfusmat(A,B,Fus_Method);
```

```
figure;
colormap(pink(220))
subplot(1,3,1), image(A), title('Original Image 1'), axis square
subplot(1,3,2), image(C), title('Fused Image'), axis square
subplot(1,3,3), image(B), title('Original Image 2'), axis square
```



```
%*****
% User defined fusion method. *
%*****
function C = myfus_FUN(A,B)

D = logical(triu(ones(size(A)))); t = 0.3;
C = A;
C(D) = t*A(D)+(1-t)*B(D);
C(~D) = t*B(~D)+(1-t)*A(~D);
```

**References**

Zeeuw, P.M. (1998), “Wavelet and image fusion,” CWI, Amsterdam, March 1998, <http://www.cwi.nl/~pauldz/>

Misiti, M.; Y. Misiti, G. Oppenheim, J.-M. Poggi (2003), “Les ondelettes et leurs applications,” Hermes.

**See Also**

wfusmat, wextend

# wfusmat

---

**Purpose** Fusion of two matrices or arrays

**Syntax** `C = wfusmat(A,B,METHOD)`  
`[C,D] = wfusmat(A,B,METHOD)`

**Description** `C = wfusmat(A,B,METHOD)` returns the fused matrix `C` obtained from the matrices `A` and `B` using the fusion method defined by `METHOD`.

The matrices `A` and `B` must be of the same size. The output matrix `C` is of the same size as `A` and `B`.

Available fusion methods are

- Simple, where `METHOD` is
  - 'max' :  $D = \text{abs}(A) \geq \text{abs}(B)$  ;  $C = A(D) + B(\sim D)$
  - 'min' :  $D = \text{abs}(A) \leq \text{abs}(B)$  ;  $C = A(D) + B(\sim D)$
  - 'mean' :  $C = (A+B) / 2$  ;  $D = \text{ones}(\text{size}(A))$
  - 'rand' :  $C = A(D) + B(\sim D)$  ;  $D$  is a Boolean random matrix
  - 'img1' :  $C = A$
  - 'img2' :  $C = B$
- Parameter-dependent, where `METHOD` is of the following form:

```
METHOD = struct('name',nameMETH,'param',paramMETH)
```

where `nameMETH` can be

- 'linear' :  $C = A*\text{paramMETH} + B*(1-\text{paramMETH})$ ,  
where  $0 \leq \text{paramMETH} \leq 1$
- 'UD\_fusion': Up-down fusion, with  $\text{paramMETH} \geq 0$   
`x = linspace(0,1,size(A,1));`  
`P = x.^paramMETH;`

Then each row of `C` is computed with

```
C(i,:) = A(i,)*(1-P(i)) + B(i,)*P(i);  
So C(1,:) = A(1,:) and C(end,:) = A(end,:)
```

- 'DU\_fusion': Down-up fusion
- 'LR\_fusion': Left-right fusion (columnwise fusion)
- 'RL\_fusion': Right-left fusion (columnwise fusion)

- 'UserDEF': User-defined fusion, paramMETH is a string 'userFUNCTION' containing a function name such that  $C = \text{userFUNCTION}(A, B)$ .

In addition,  $[C, D] = \text{wfusmat}(A, B, \text{METHOD})$  returns the Boolean matrix  $D$  when defined, or an empty matrix otherwise.

# wkeep

---

**Purpose** Keep part of vector or matrix

**Syntax** `Y = wkeep(X,L,OPT)`  
`Y = wkeep(X,L)`

**Description** `wkeep` is a general utility.

For a vector, `Y = wkeep(X,L,OPT)` extracts the vector `Y` from the vector `X`. The length of `Y` is `L`.

If `OPT = 'c'` ('l', 'r', respectively), `Y` is the central (left, right, respectively) part of `X`.

`Y = wkeep(X,L,FIRST)` returns the vector `X(FIRST:FIRST+L-1)`.

`Y = wkeep(X,L)` is equivalent to `Y = wkeep(X,L,'c')`.

For a matrix, `Y = wkeep(X,S)` extracts the central part of the matrix `X`. The size of `Y` is `S`.

`Y = wkeep(X,S,[FIRSTR FIRSTC])` extracts the submatrix of matrix `X`, of size `S` and starting from `X(FIRSTR,FIRSTC)`.

## Examples

```
% For a vector.  
x = 1:10;  
y = wkeep(x,6,'c')  
y =  
    3    4    5    6    7    8  
  
y = wkeep(x,6)  
y =  
    3    4    5    6    7    8  
  
y = wkeep(x,7,'c')  
y =  
    2    3    4    5    6    7    8  
y = wkeep(x,6,'l')  
y =  
    1    2    3    4    5    6
```

```
y = wkeep(x,6,'r')
y =
     5     6     7     8     9    10

% For a matrix.
x = magic(5)
x =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

y = wkeep(x,[3 2])
y =
     5     7
     6    13
    12    19
```

# wmaxlev

---

**Purpose** Maximum wavelet decomposition level

**Syntax** `L = wmaxlev(S, 'wname')`

**Description** `wmaxlev` is a one- or two-dimensional wavelet or wavelet packets oriented function.

`wmaxlev` can help you avoid unreasonable maximum level values.

`L = wmaxlev(S, 'wname')` returns the maximum level decomposition of signal or image of size `S` using the wavelet named in the string `'wname'` (see `wfilters` for more information).

`wmaxlev` gives the maximum allowed level decomposition, but in general, a smaller value is taken.

Usual values are 5 for the one-dimensional case, and 3 for the two-dimensional case.

## Examples

```
% For a 1-D signal.
s = 2^10;
w = 'db1';

% Compute maximum level decomposition.
% The rule is the last level for which at least
% one coefficient is correct.
l = wmaxlev(s,w)

l =
    10

% Change wavelet.
w = 'db7';

% Compute maximum level decomposition.
l = wmaxlev(s,w)

l =
     6
```



```
% For a 2-D signal.
s = [2^9 2^7];
w = 'db1';

% Compute maximum level decomposition.
l = wmaxlev(s,w)

l =
    7

% which is the same as:
l = wmaxlev(min(s),w)

l =
    7

% Change wavelet.
w = 'db7';

% Compute maximum level decomposition.
l = wmaxlev(s,w)

l =
    3
```

**See Also**

wavedec, wavedec2, wpdec, wpdec2

**Purpose** Multiscale Principal Component Analysis

**Syntax**  
`[X_SIM,QUAL,NPC,DEC_SIM,PCA_Params] = wmspca(X,LEVEL,WNAME,NPC)`  
`[...] = wmspca(X,LEVEL,WNAME,'mode',EXTMODE,NPC)`  
`[...] = wmspca(DEC,NPC)`

**Description** `[X_SIM,QUAL,NPC,DEC_SIM,PCA_Params] = wmspca(X,LEVEL,WNAME,NPC)` or `[...] = wmspca(X,LEVEL,WNAME,'mode',EXTMODE,NPC)` returns a simplified version `X_SIM` of the input matrix `X` obtained from the wavelet-based multiscale Principal Component Analysis (PCA).

The input matrix `X` contains `P` signals of length `N` stored columnwise ( $N > P$ ).

### Wavelet Decomposition Parameters

The wavelet decomposition is performed using the decomposition level `LEVEL` and the wavelet `WNAME`.

`EXTMODE` is the extended mode for the DWT (See `dwtmode`).

If a decomposition `DEC` obtained using `mdwtdec` is available, you can use

`[...] = wmspca(DEC,NPC)` instead of  
`[...] = wmspca(X,LEVEL,WNAME,'mode',EXTMODE,NPC)`.

### Principal Components Parameter: NPC

If `NPC` is a vector, then it must be of length `LEVEL+2`. It contains the number of retained principal components for each PCA performed:

- `NPC(d)` is the number of retained noncentered principal components for details at level `d`, for  $1 \leq d \leq \text{LEVEL}$ .
- `NPC(LEVEL+1)` is the number of retained non-centered principal components for approximations at level `LEVEL`.
- `NPC(LEVEL+2)` is the number of retained principal components for final PCA after wavelet reconstruction.

`NPC` must be such that  $0 \leq \text{NPC}(d) \leq P$  for  $1 \leq d \leq \text{LEVEL}+2$ .

If `NPC = 'kais'` (respectively, `'heur'`), then the number of retained principal components is selected automatically using Kaiser's rule (or the heuristic rule).

- Kaiser's rule keeps the components associated with eigenvalues greater the mean of all eigenvalues.
- The heuristic rule keeps the components associated with eigenvalues greater than 0.05 times the sum of all eigenvalues.

If NPC = 'nodet', then the details are “killed” and all the approximations are retained.

### Output Parameters

X\_SIM is a simplified version of the matrix X.

QUAL is a vector of length P containing the quality of column reconstructions given by the relative mean square errors in percent.

NPC is the vector of selected numbers of retained principal components.

DEC\_SIM is the wavelet decomposition of X\_SIM.

PCA\_Params is a structure array of length LEVEL+2 such that:

- PCA\_Params(d).pc is a P-by-P matrix of principal components.  
The columns are stored in descending order of the variances.
- PCA\_Params(d).variances is the principal component variances vector.
- PCA\_Params(d).npc = NPC

## Examples

```
% Load a multivariate signal x.
load ex4mwden

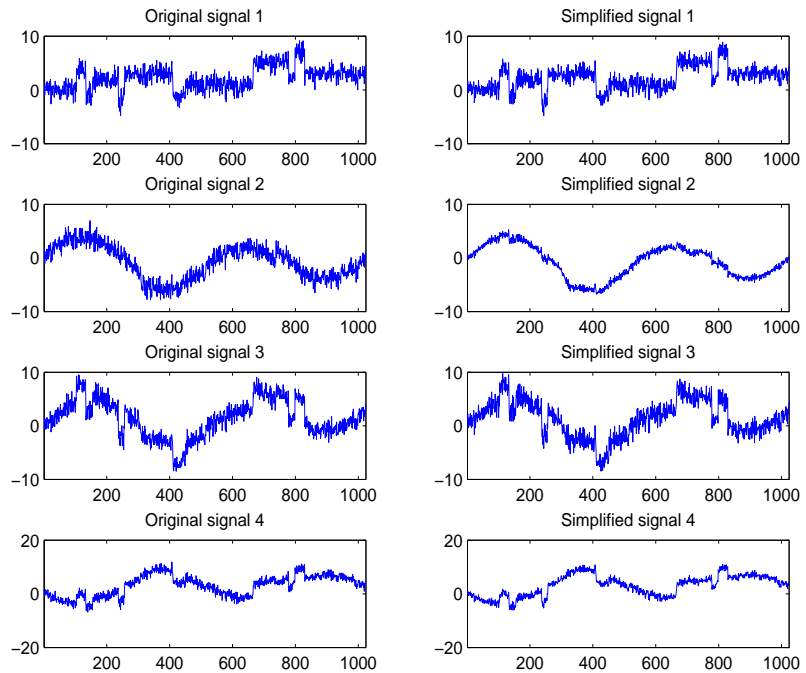
% Set the wavelet parameters.
level = 5;
wname = 'sym4';

% Set the PCA parameters to select the number
% of retained principal components automatically
% by Kaiser's rule.

npc = 'kais';

% Perform a first multiscale PCA.
[x_sim, qual, npc] = wmspca(x,level,wname,npc);

% Display the original and simplified signals.
kp = 0;
for i = 1:4
    subplot(4,2,kp+1), plot(x(:,i));
    title(['Original signal ',num2str(i)])
    subplot(4,2,kp+2), plot(x_sim(:,i));
    title(['Simplified signal ',num2str(i)])
    kp = kp + 2;
end
```



```
% The results from a compression perspective are good: the
% quality of column reconstructions are close to 100%.
```

```
qual
```

```
qual =
```

```
98.0545 93.2807 97.1172 98.8603
```

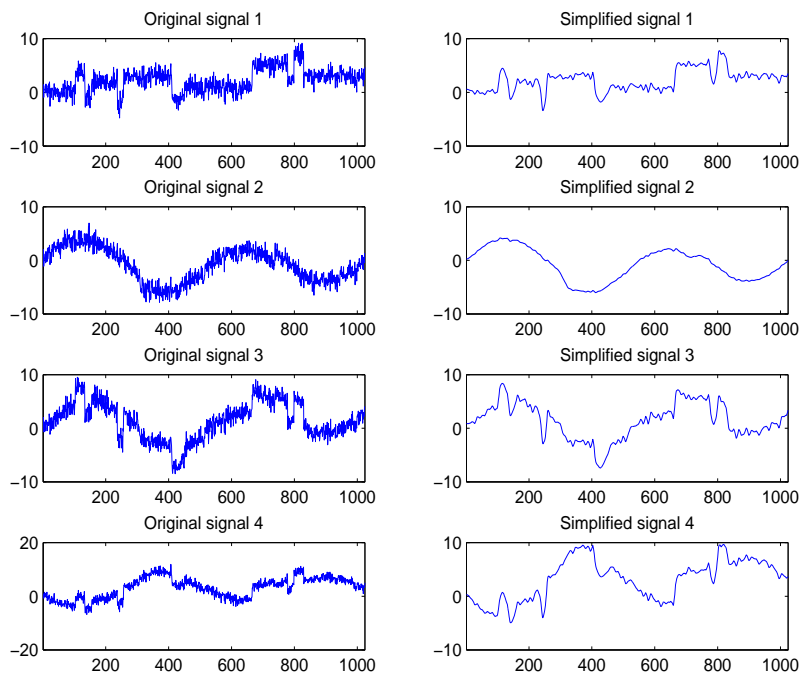
```
% The output argument npc gives the number
% of retained principal components.
```

```
npc =
```

```
npc =
```

```
1 1 1 1 1 2 2
```

```
% The results can be improved by suppressing noise,  
% by killing the details at levels 1 to 3.  
  
npc(1:3) = zeros(1,3);  
  
% Perform again multiscale PCA.  
  
[x_sim, qual, npc] = wmspca(x,level,wname,npc);  
  
% Display the original and final simplified signals.  
% Use these plotting commands to generate the figure.  
  
kp = 0;  
for i = 1:4  
    subplot(4,2,kp+1), plot(x (:,i));  
    title(['Original signal ',num2str(i)])  
    subplot(4,2,kp+2), plot(x_sim(:,i));  
    title(['Simplified signal ',num2str(i)])  
    kp = kp + 2;  
end
```



## Algorithm

The multiscale principal components generalizes the usual PCA of a multivariate signal seen as a matrix by performing simultaneously a PCA on the matrices of details of different levels. In addition, a PCA is performed also on the coarser approximation coefficients matrix in the wavelet domain as well as on the final reconstructed matrix. By selecting conveniently the numbers of retained principal components, interesting simplified signals can be reconstructed.

## References

- Aminghafari, M.; Cheze, N.; Poggi, J-M. (2006), "Multivariate de-noising using wavelets and principal component analysis," *Computational Statistics & Data Analysis*, 50, pp. 2381–2398.
- Bakshi, B. (1998), "Multiscale PCA with application to MSPC monitoring," *AIChE J.*, 44, pp. 1596–1610.

## See Also

wmulden

# wmulden

---

## Purpose

Wavelet multivariate de-noising

## Syntax

```
[X_DEN,NPC,NESTCOV,DEC_DEN,PCA_Params,DEN_Params] = ...  
    wmulden(X,LEVEL,WNAME,NPC_APP,NPC_FIN,TPTR,SORH)  
[...] = wmulden(X,LEVEL,WNAME,'mode',EXTMODE,NPC_APP,...)  
[DEC,PCA_Params] = wmulden('estimate',DEC,NPC_APP,NPC_FIN)  
[X_DEN,NPC,DEC_DEN,PCA_Params] = ...  
    wmulden('execute',DEC,PCA_Params)  
[...] = wmulden('execute',DEC,PCA_Params,TPTR,SORH)
```

## Description

```
[X_DEN,NPC,NESTCOV,DEC_DEN,PCA_Params,DEN_Params] = ...  
    wmulden(X,LEVEL,WNAME,NPC_APP,NPC_FIN,TPTR,SORH) or  
[...] = wmulden(X,LEVEL,WNAME,'mode',EXTMODE,NPC_APP,...) returns a  
de-noised version X_DEN of the input matrix X. The strategy combines  
univariate wavelet de-noising in the basis where the estimated noise  
covariance matrix is diagonal with noncentered Principal Component Analysis  
(PCA) on approximations in the wavelet domain or with final PCA.
```

The input matrix X contains P signals of length N stored columnwise where  $N > P$ .

### Wavelet Decomposition Parameters

The wavelet decomposition is performed using the decomposition level LEVEL and the wavelet WNAME.

EXTMODE is the extended mode for the DWT (See `dwtmode`).

If a decomposition DEC obtained using `mdwtdec` is available, you can use

```
[...] = wmulden(DEC,NPC_APP) instead of  
[...] = wmulden(X,LEVEL,WNAME,'mode',EXTMODE,NPC_APP).
```

### Principal Components Parameters: NPC\_APP and NPC\_FIN

The input selection methods NPC\_APP and NPC\_FIN define the way to select principal components for approximations at level LEVEL in the wavelet domain and for final PCA after wavelet reconstruction, respectively.



If NPC\_APP (or NPC\_FIN) is an integer, it contains the number of retained principal components for approximations at level LEVEL (or for final PCA after wavelet reconstruction).

NPC\_XXX must be such that  $0 \leq \text{NPC\_XXX} \leq P$

NPC\_APP or NPC\_FIN = 'kais' (or 'heur') selects the number of retained principal components using Kaiser's rule (or the heuristic rule) automatically.

- Kaiser's rule keeps the components associated with eigenvalues greater than the mean of all eigenvalues.
- Heuristic rule keeps the components associated with eigenvalues greater than 0.05 times the sum of all eigenvalues.

NPC\_APP or NPC\_FIN = 'none' is equivalent to NPC\_APP or NPC\_FIN = P.

### De-noising Parameters: TPTR and SORH

The default values for the de-noising parameters TPTR and SORH are:

TPTR = 'sqrtwolog' and SORH = 's'

- Valid values for TPTR are  
'rigsure', 'heursure', 'sqrtwolog', 'minimaxi',  
'penalhi', 'penalme', 'penallo'
- Valid values for SORH are:  
's' (soft) or 'h' (hard)

For additional information, ssee wden and wbmpe.

### Output Parameters

X\_DEN is a de-noised version of the input matrix X.

NPC is the vector of selected numbers of retained principal components.

NESTCOV is the estimated noise covariance matrix obtained using the minimum covariance determinant (MCD) estimator.

DEC\_DEN is the wavelet decomposition of X\_DEN.

PCA\_Params is a structure such that:

```
PCA_Params.NEST = {pc_NEST,var_NEST,NESTCOV}  
PCA_Params.APP  = {pc_APP,var_APP,npc_APP}  
PCA_Params.FIN  = {pc_FIN,var_FIN,npc_FIN}
```

where:

- `pc_XXX` is a P-by-P matrix of principal components.  
The columns are stored in descending order of the variances.
- `var_XXX` is the principal component variances vector.
- `NESTCOV` is the covariance matrix estimate for detail at level 1.

`DEN_Params` is a structure such that:

- `DEN_Params.thrVAL` is a vector of length `LEVEL` which contains the threshold values for each level.
- `DEN_Params.thrMETH` is a string containing the name of the de-noising method (TPTR).
- `DEN_Params.thrTYPE` is a character variable containing the type of the thresholding (SORH).

## Special Cases

`[DEC,PCA_Params] = wmulden('estimate',DEC,NPC_APP,NPC_FIN)` returns the wavelet decomposition `DEC` and the Principal Components Estimates `PCA_Params`.

`[X_DEN,NPC,DEC_DEN,PCA_Params] = wmulden('execute',DEC,PCA_Params)` or `[...] = wmulden('execute',DEC,PCA_Params,TPTR,SORH)` uses the Principal Components Estimates `PCA_Params` previously computed.

The input value `DEC` can be replaced by `X`, `LEVEL`, and `WNAME`.

## Examples

```
% Load a multivariate signal x together with  
% the original signals (x_orig) and true noise  
% covariance matrix (covar).  
  
load ex4mwden  
  
% Set the de-noising method parameters.  
level = 5;
```

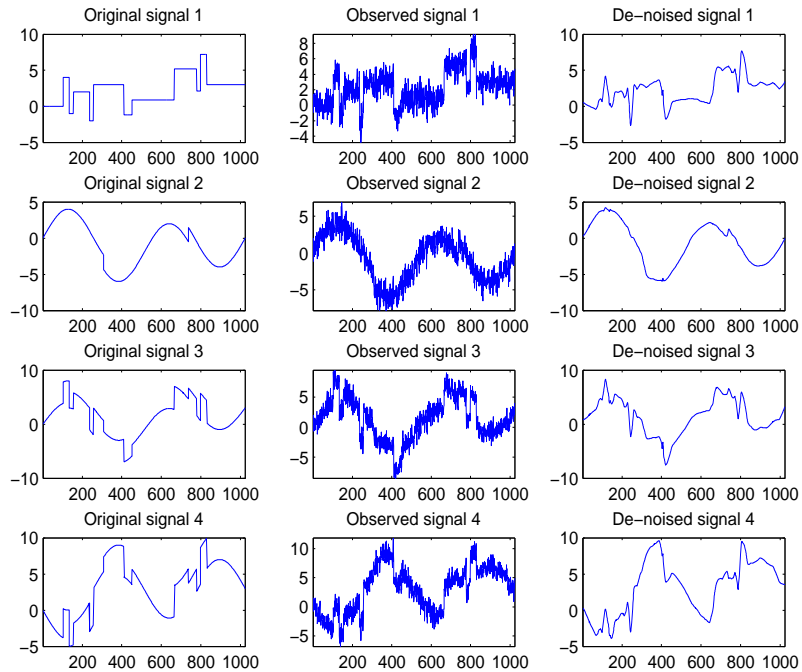
```
wname = 'sym4';
tptr  = 'sqrtwolog';
sorh  = 's';

% Set the PCA parameters to select the number of
% retained principal components automatically by
% Kaiser's rule.

npc_app = 'kais';
npc_fin = 'kais';

% Perform multivariate de-noising.
[x_den, npc, nestco] = wmulden(x, level, wname, npc_app, ...
                             npc_fin, tptr, sorh);

% Display the original and de-noised signals.
kp = 0;
for i = 1:4
    subplot(4,3,kp+1), plot(x_orig(:,i));
    title(['Original signal ',num2str(i)])
    subplot(4,3,kp+2), plot(x(:,i));
    title(['Observed signal ',num2str(i)])
    subplot(4,3,kp+3), plot(x_den(:,i));
    title(['De-noised signal ',num2str(i)])
    kp = kp + 3;
end
```



```
% The results are good: the first function, which is  
% irregular, is correctly recovered while the second  
% function, more regular, is well de-noised.
```

```
% The second output argument gives the numbers  
% of retained principal components for PCA for  
% approximations and for final PCA.
```

```
npc
```

```
npc =
```

```
2 2
```

```
% The third output argument contains the estimated  
% noise covariance matrix using the MCD based
```

```

% on the matrix of finest details.

nestco

nestco =

    1.0784    0.8333    0.6878    0.8141
    0.8333    1.0025    0.5275    0.6814
    0.6878    0.5275    1.0501    0.7734
    0.8141    0.6814    0.7734    1.0967

% The estimation is satisfactory since the values are close
% to the true values given by covar.

covar

covar =

    1.0000    0.8000    0.6000    0.7000
    0.8000    1.0000    0.5000    0.6000
    0.6000    0.5000    1.0000    0.7000
    0.7000    0.6000    0.7000    1.0000

```

## Algorithm

The multivariate de-noising procedure is a generalization of the one-dimensional strategy. It combines univariate wavelet de-noising in the basis where the estimated noise covariance matrix is diagonal and non-centered Principal Component Analysis (PCA) on approximations in the wavelet domain or with final PCA.

The robust estimate of the noise covariance matrix given by the minimum covariance determinant estimator based on the matrix of finest details.

## References

Aminghafari, M.; Cheze, N.; Poggi, J-M. (2006), "Multivariate de-noising using wavelets and principal component analysis," *Computational Statistics & Data Analysis*, 50, pp. 2381–2398.

Rousseeuw, P.; Van Driessen, K. (1999), "A fast algorithm for the minimum covariance determinant estimator," *Technometrics*, 41, pp. 212–223.

## See Also

wmspca

# wnoise

---

**Purpose** Noisy wavelet test data

**Syntax**

```
X = wnoise(FUN,N)
[X,XN] = wnoise(FUN,N,SQRT_SNR)
[X,XN] = wnoise(FUN,N,SQRT_SNR,INIT)
```

**Description** `X = wnoise(FUN,N)` returns values of the test signal given by `FUN`, on a  $2^N$  grid of  $[0,1]$ .

`[X,XN] = wnoise(FUN,N,SQRT_SNR)` returns a test vector `X` as above, rescaled such that  $\text{std}(X) = \text{SQRT\_SNR}$ . The returned vector `XN` contains the same test vector corrupted by additive Gaussian white noise  $N(0,1)$ . Then, `XN` has a signal-to-noise ratio of  $\text{SNR} = (\text{SQRT\_SNR})^2$ .

`[X,XN] = wnoise(FUN,N,SQRT_SNR,INIT)` returns previous vectors `X` and `XN`, but the generator seed is set to `INIT` value.

The six functions below are due to Donoho and Johnstone (See “References”).

```
FUN = 1 or 'blocks'
FUN = 2 or 'bumps'
FUN = 3 or 'heavy sine'
FUN = 4 or 'doppler'
FUN = 5 or 'quadchirp'
FUN = 6 or 'mishmash'
```

**Examples**

```
% Generate 2^10 samples of 'Heavy sine' (item 3).
x = wnoise(3,10);

% Generate 2^10 samples of 'Doppler' (item 4) and of
% noisy 'Doppler' with a square root of signal-to-noise
% ratio equal to 7.
[x,noisyx] = wnoise(4,10,7);

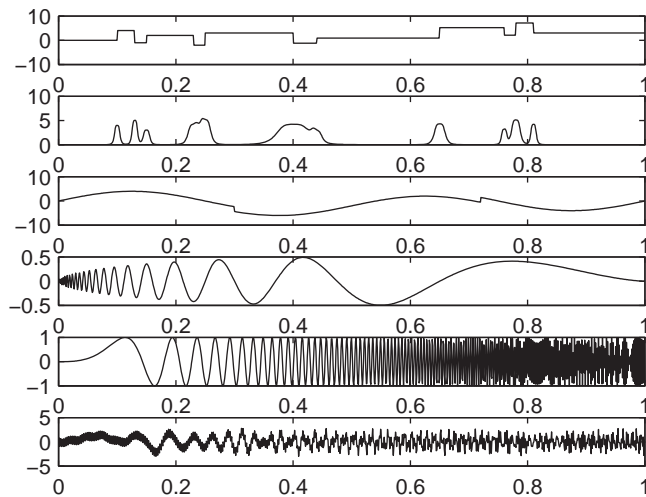
% To introduce your own rand seed, a fourth
% argument is allowed:
init = 2055415866;
[x,noisyx] = wnoise(4,10,7,init);
```

```

% Plot all the test functions.
ind = linspace(0,1,2^10);
for i = 1:6
    x = wnoise(i,10);
    subplot(6,1,i), plot(ind,x)
end

% Editing some graphical properties,
% the following figure is generated.

```



## References

Donoho, D.L.; I.M. Johnstone (1994), "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, pp. 425–455.

Donoho, D.L.; I.M. Johnstone (1995), "Adapting to unknown smoothness via wavelet shrinkage via wavelet shrinkage," *JASA*, vol. 90, 432, pp. 1200–1224.

## See Also

wden

# wnoisest

---

**Purpose** Estimate noise of 1-D wavelet coefficients

**Syntax** STDC = wnoisest(C,L,S)

**Description** STDC = wnoisest(C,L,S) returns estimates of the detail coefficients' standard deviation for levels contained in the input vector S. [C,L] is the input wavelet decomposition structure (see wavedec for more information).

If C is a one dimensional cell array, STDC = wnoisest(C) returns a vector such that STDC(k) is an estimate of the standard deviation of C{k}.

If C is a numeric array, STDC = wnoisest(C) returns a vector such that STDC(k) is an estimate of the standard deviation of C(k, :).

The estimator used is Median Absolute Deviation / 0.6745, well suited for zero mean Gaussian white noise in the de-noising one-dimensional model (see thselect for more information).

## Examples

```
% The current extension mode is zero-padding (see dwtmode).
```

```
% Generate Gaussian white noise.  
init = 2055415866; randn('seed',init);  
x = randn(1,1000);
```

```
% Decompose x at level 2 using db3 wavelet.  
[c,l] = wavedec(x,2,'db3');
```

```
% Estimate standard deviation of coefficients  
% at each level 1 and 2.  
% Since x is a Gaussian white noise with unit  
% variance, estimates must be close to 1.  
wnoisest(c,l,1:2)
```

```
ans =  
    1.0111    1.0763
```



```

% Now suppose that x contains 10 outliers.
ind = 50:50:500;
x(ind) = 100 * ones(size(ind));

% Decompose x at level 1 using db3 wavelet.
[ca,cd] = dwt(x,'db3');

% Ordinary estimate of cd standard deviation
% overestimates noise level.
std(cd)

ans =
    8.0206

% Robust estimate of cd standard deviation
% remains close to 1 the noise level.
median(abs(cd))/0.6745

ans =
    1.0540

```

**Limitations**

This procedure is well suited for Gaussian white noise.

**References**

Donoho, D.L.; I.M. Johnstone (1994), “Ideal spatial adaptation by wavelet shrinkage,” *Biometrika*, vol 81, pp. 425–455.

Donoho, D.L.; I.M. Johnstone (1995), “Adapting to unknown smoothness via wavelet shrinkage via wavelet shrinkage,” *JASA*, vol 90, 432, pp. 1200–1224.

**See Also**

thselect, wavedec, wden

# wp2wtree

**Purpose** Extract wavelet tree from wavelet packet tree

**Syntax** `T = wp2wtree(T)`

**Description** `wp2wtree` is a one- or two-dimensional wavelet packet analysis function.

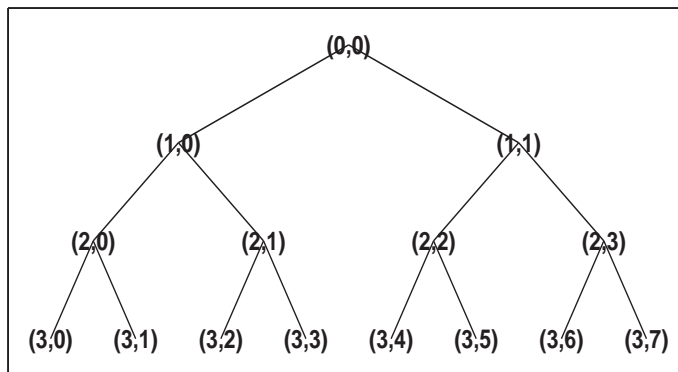
`T = wp2wtree(T)` computes the modified wavelet packet tree `T` corresponding to the wavelet decomposition tree.

**Examples** `% The current extension mode is zero-padding (see dwtmode).`

```
% Load signal.
    load noisdopp; x = noisdopp;

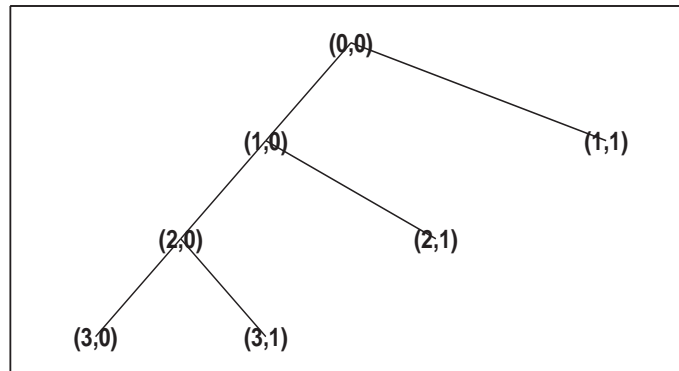
% Decompose x at depth 3 with db1 wavelet packets
% using shannon entropy.
wpt = wpdec(x,3,'db1');

% Plot wavelet packet tree wpt.
plot(wpt)
```



```
% Compute wavelet tree.
wt = wp2wtree(wpt);
```

```
% Plot wavelet tree wt.  
plot(wt)
```

**See Also**

wpdec, wpdec2

# wpbmpen

---

**Purpose** Penalized threshold for wavelet packet de-noising

**Syntax** THR = wpbmpen(T, SIGMA, ALPHA)  
THR = wpbmpen(T, SIGMA, ALPHA, ARG)

**Description** THR = wpbmpen(T, SIGMA, ALPHA) returns a global threshold THR for de-noising. THR is obtained by a wavelet packet coefficients selection rule using a penalization method provided by Birge-Massart.

T is a wavelet packet tree corresponding to the wavelet packet decomposition of the signal or image to be de-noised.

SIGMA is the standard deviation of the zero mean Gaussian white noise in the de-noising model (see wnoisest for more information).

ALPHA is a tuning parameter for the penalty term. It must be a real number greater than 1. The sparsity of the wavelet packet representation of the de-noised signal or image grows with ALPHA. Typically ALPHA = 2.

THR minimizes the penalized criterion given by

let  $t^*$  be the minimizer of

$$\text{crit}(t) = -\sum(c(k)^2, k \leq t) + 2 \cdot \text{SIGMA}^2 \cdot t \cdot (\text{ALPHA} + \log(n/t))$$

where  $c(k)$  are the wavelet packet coefficients sorted in decreasing order of their absolute value and  $n$  is the number of coefficients, then  $\text{THR} = |c(t^*)|$ .

wpbmpen(T, SIGMA, ALPHA, ARG) computes the global threshold and, in addition, plots three curves:

- $2 \cdot \text{SIGMA}^2 \cdot t \cdot (\text{ALPHA} + \log(n/t))$
- $\sum(c(k)^2, k \leq t)$
- $\text{crit}(t)$

**Examples**

```
% Example 1: Signal de-noising.  
% Load noisy chirp signal.  
load noisichir; x = noisichir;
```

```
% Perform a wavelet packet decomposition of the signal
% at level 5 using sym6.
wname = 'sym6'; lev = 5;
tree = wpdec(x,lev,wname);

% Estimate the noise standard deviation from the
% detail coefficients at level 1,
% corresponding to the node index 2.
det1 = wpccoef(tree,2);
sigma = median(abs(det1))/0.6745;

% Use wpbmpen for selecting global threshold
% for signal de-noising, using the recommended parameter.
alpha = 2;
thr = wpbmpen(tree,sigma,alpha)

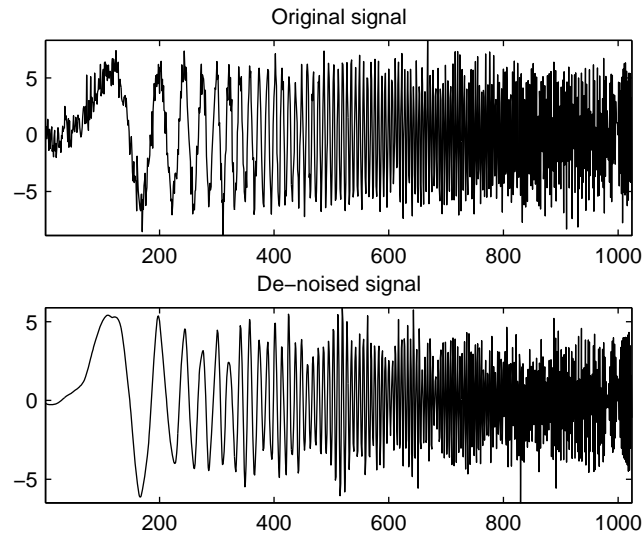
thr =

    4.5740

% Use wpdencmp for de-noising the signal using the above
% threshold with soft thresholding and keeping the approximation.
keepapp = 1;
xd = wpdencmp(tree,'s','nobest',thr,keepapp);

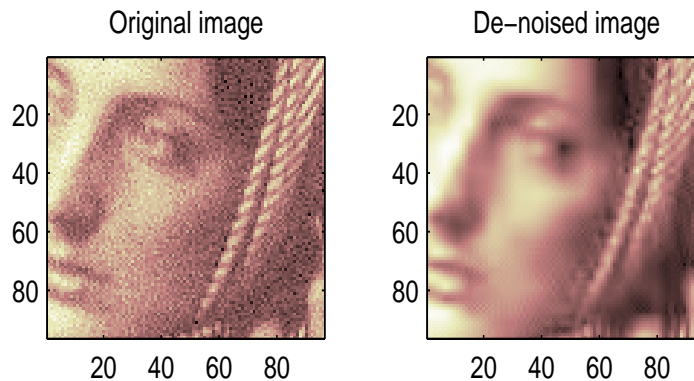
% Plot original and de-noised signals.
figure(1)
subplot(211), plot(x),
title('Original signal')
```

```
subplot(212), plot(xd)  
title('De-noised signal')
```



```
% Example 2: Image de-noising.  
% Load original image.  
load noiswom;  
nbc = size(map,1);  
  
% Perform a wavelet packet decomposition of the image  
% at level 3 using coif2.  
wname = 'coif2'; lev = 3;  
tree = wpdec2(X,lev,wname);  
  
% Estimate the noise standard deviation from the  
% detail coefficients at level 1.  
det1 = [wpcoef(tree,2) wpcoef(tree,3) wpcoef(tree,4)];  
sigma = median(abs(det1(:)))/0.6745;  
  
% Use wpbmpen for selecting global threshold  
% for image de-noising.  
alpha = 1.1;  
thr = wpbmpen(tree,sigma,alpha)
```

```
thr =  
  
    38.5125  
  
% Use wpdncmp for de-noising the image using the above  
% thresholds with soft thresholding and keeping the approximation.  
keepapp = 1;  
xd = wpdncmp(tree,'s','nobest',thr,keepapp);  
  
% Plot original and de-noised images.  
figure(2)  
colormap(pink(nbc));  
subplot(221), image(wcodemat(X,nbc))  
title('Original image')  
subplot(222), image(wcodemat(xd,nbc))  
title('De-noised image')
```

**See Also**

wbmpen, wden, wdencmp, wpdncmp

# wpcoef

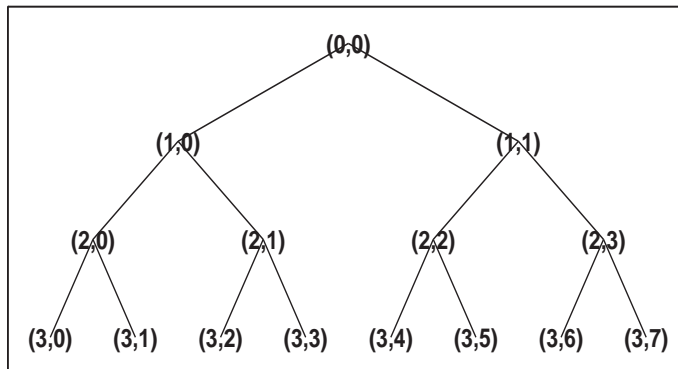
**Purpose** Wavelet packet coefficients

**Syntax**  
`X = wpcoef(T,N)`  
`X = wpcoef(T)`

**Description**  
`wpcoef` is a one- or two-dimensional wavelet packet analysis function.  
`X = wpcoef(T,N)` returns the coefficients associated with the node `N` of the wavelet packet tree `T`. If `N` doesn't exist, `X = []`;  
`X = wpcoef(T)` is equivalent to `X = wpcoef(T,0)`.

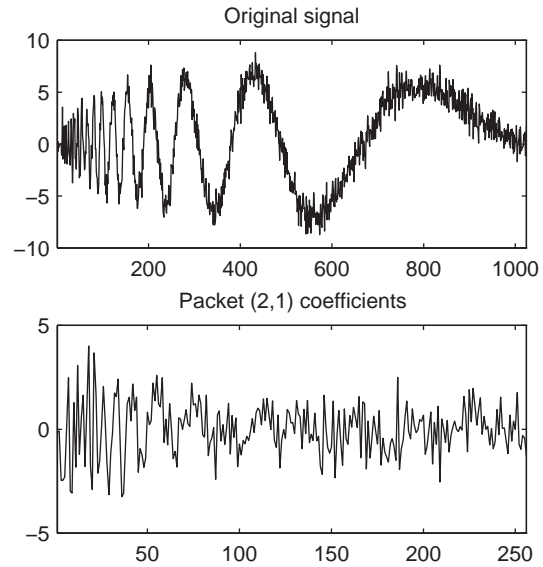
**Examples**

```
% The current extension mode is zero-padding (see dwtmode).  
  
% Load signal.  
load noisdopp; x = noisdopp;  
  
figure(1); subplot(211);  
plot(x); title('Original signal');  
  
% Decompose x at depth 3 with db1 wavelet packets  
% using Shannon entropy.  
wpt = wptdec(x,3,'db1');  
  
% Plot wavelet packet tree wpt.  
plot(wpt)
```





```
% Read packet (2,1) coefficients.  
cfs = wpcoef(wpt,[2 1]);  
  
figure(1); subplot(212);  
plot(cfs); title('Packet (2,1) coefficients');
```

**See Also**

wpdec, wpdec2

# wpcutree

**Purpose** Cut wavelet packet tree

**Syntax**  
`T = wpcutree(T,L)`  
`[T,RN] = wpcutree(T,L)`

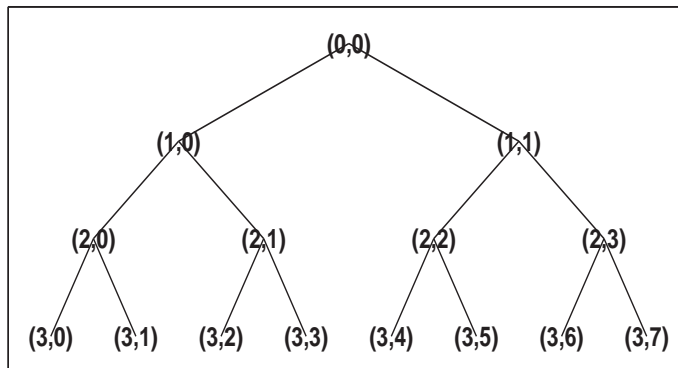
**Description** `wpcutree` is a one- or two-dimensional wavelet packet analysis function.

`T = wpcutree(T,L)` cuts the tree `T` at level `L`.

`[T,RN] = wpcutree(T,L)` returns the same arguments as above and, in addition, the vector `RN` contains the indices of the reconstructed nodes.

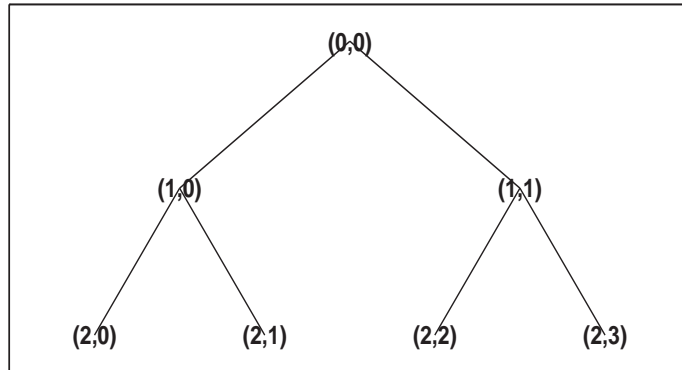
**Examples** `% The current extension mode is zero-padding (see dwtmode).`

```
% Load signal.  
load noisdopp; x = noisdopp;  
  
% Decompose x at depth 3 with db1 wavelet packets  
% using Shannon entropy.  
wpt = wpdec(x,3,'db1');  
  
% Plot wavelet packet tree wpt.  
plot(wpt)
```



```
% Cut wavelet packet tree at level 2.  
nwpt = wpcutree(wpt,2);
```

```
% Plot new wavelet packet tree nwpt.  
plot(nwpt)
```

**See Also**

wpdec, wpdec2

**Purpose** Wavelet packet decomposition 1-D

**Syntax**  
T = wpdec(X,N,'wname',E,P)  
T = wpdec(X,N,'wname')

**Description** wpdec is a one-dimensional wavelet packet analysis function.

T = wpdec(X,N,'wname',E,P) returns a wavelet packet tree T corresponding to the wavelet packet decomposition of the vector X at level N, with a particular wavelet ('wname', see wfilters for more information).

T = wpdec(X,N,'wname') is equivalent to  
T = wpdec(X,N,'wname','shannon').

E is a string containing the type of entropy and P is an optional parameter depending on the value of T (see wentropy for more information).

Entropy Type Name (E)	Parameter (P)	Comments
'shannon'		P is not used.
'log energy'		P is not used.
'threshold'	$0 \leq P$	P is the threshold.
'sure'	$0 \leq P$	P is the threshold.
'norm'	$1 \leq P$	P is the power.
'user'	string	P is a string containing the M-file name of your own entropy function, with a single input X.
FunName	No constraints on P	FunName is any other string except those used for the previous Entropy Type Names listed above. FunName contains the M-file name of your own entropy function, with X as input and P as additional parameter to your entropy function.

---

**Note** The 'user' option is historical and still kept for compatibility, but it is obsoleted by the last option described in the table above. The FunName option do the same as the 'user' option and in addition gives the possibility to pass a parameter to your own entropy function.

---

The wavelet packet method is a generalization of wavelet decomposition that offers a richer signal analysis. Wavelet packet atoms are waveforms indexed by three naturally interpreted parameters: position and scale as in wavelet decomposition, and frequency.

For a given orthogonal wavelet function, a library of wavelet packets bases is generated. Each of these bases offers a particular way of coding signals, preserving global energy and reconstructing exact features. The wavelet packets can then be used for numerous expansions of a given signal.

Simple and efficient algorithms exist for both wavelet packets decomposition and optimal decomposition selection. Adaptive filtering algorithms with direct applications in optimal signal coding and data compression can then be produced.

In the orthogonal wavelet decomposition procedure, the generic step splits the approximation coefficients into two parts. After splitting we obtain a vector of approximation coefficients and a vector of detail coefficients, both at a coarser scale. The information lost between two successive approximations is captured in the detail coefficients. The next step consists in splitting the new approximation coefficient vector; successive details are never re-analyzed.

In the corresponding wavelet packets situation, each detail coefficient vector is also decomposed into two parts using the same approach as in approximation vector splitting. This offers the richest analysis: the complete binary tree is produced in the one-dimensional case or a quaternary tree in the two-dimensional case.

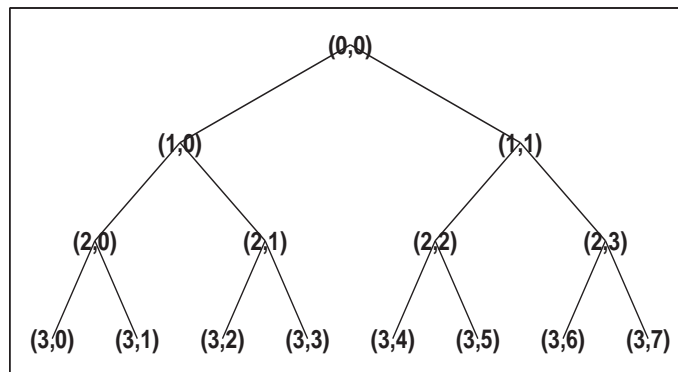
## Examples

```
% The current extension mode is zero-padding (see dwtmode).  
  
% Load signal.  
load noisdopp; x = noisdopp;
```

```
% Decompose x at depth 3 with db1 wavelet packets
% using Shannon entropy.
wpt = wpdec(x,3,'db1','shannon');

% The result is the wavelet packet tree wpt.

% Plot wavelet packet tree (binary tree, or tree of order 2).
plot(wpt)
```



## Algorithm

The algorithm used for the wavelet packets decomposition follows the same line as the wavelet decomposition process (see `dwt` and `wavedec` for more information).

## References

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.

Meyer, Y. (1993), *Les ondelettes. Algorithmes et applications*, Colin Ed., Paris, 2nd edition. (English translation: *Wavelets: Algorithms and Applications*, SIAM).

Wickerhauser, M.V. (1991), "INRIA lectures on wavelet packet algorithms," *Proceedings ondelettes et paquets d'ondes*, 17–21 June, Rocquencourt, France, pp. 31–99.

Wickerhauser, M.V. (1994), *Adapted wavelet analysis from theory to software algorithms*, A.K. Peters.

## See Also

`wavedec`, `waveinfo`, `wenergy`, `wpdec`, `wprec`

**Purpose** Wavelet packet decomposition 2-D

**Syntax**  
`T = wpdec2(X,N, 'wname', E,P)`  
`T = wpdec2(X,N, 'wname')`

**Description** `wpdec2` is a two-dimensional wavelet packet analysis function.

`T = wpdec2(X,N, 'wname', E,P)` returns a wavelet packet tree `T` corresponding to the wavelet packet decomposition of the matrix `X`, at level `N`, with a particular wavelet (`'wname'`, see `wfilters` for more information).

`T = wpdec2(X,N, 'wname')` is equivalent to  
`T = wpdec2(X,N, 'wname', 'shannon')`.

`E` is a string containing the type of entropy and `P` is an optional parameter depending on the value of `T` (see `wentropy` for more information).

Entropy Type Name (E)	Parameter (P)	Comments
'shannon'		P is not used.
'log energy'		P is not used.
'threshold'	$0 \leq P$	P is the threshold.
'sure'	$0 \leq P$	P is the threshold.
'norm'	$1 \leq P$	P is the power.
'user'	string	P is a string containing the M-file name of your own entropy function, with a single input X.
STR	No constraints on P	STR is any other string except those used for the previous Entropy Type Names listed above. STR contains the M-file name of your own entropy function, with X as input and P as additional parameter to your entropy function.

---

**Note** The 'user' option is historical and still kept for compatibility, but it is obsolete by the last option described in the preceding table. The FunName option does the same as the 'user' option and in addition, allows you to pass a parameter to your own entropy function.

---

See `wpdec` for a more complete description of the wavelet packet decomposition.

## Remarks

When  $X$  represents an indexed image,  $X$  is an  $m$ -by- $n$  matrix. When  $X$  represents a truecolor image, it is an  $m$ -by- $n$ -by-3 array, where each  $m$ -by- $n$  matrix represents a red, green, or blue color plane concatenated along the third dimension.

For more information on image formats, see the `image` and `imfinfo` reference pages.

## Examples

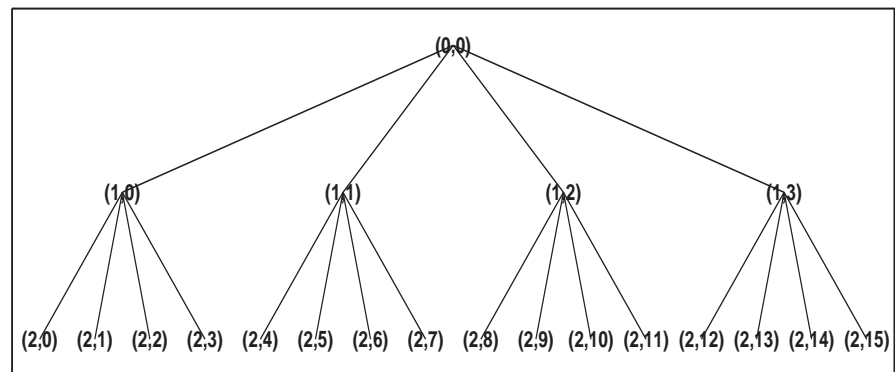
```
% The current extension mode is zero-padding (see dwtmode).

% Load image.
load tire
% X contains the loaded image.

% For an image the decomposition is performed using:
t = wpdec2(X,2,'db1');
% The default entropy is shannon.

% Plot wavelet packet tree
% (quarternary tree, or tree of order 4).
plot(t)
```





## Algorithm

The algorithm used for the wavelet packets decomposition follows the same line as the wavelet decomposition process (see `dwt2` and `wavedec2` for more information).

## References

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.

Meyer, Y. (1993), *Les ondelettes. Algorithmes et applications*, Colin Ed., Paris, 2nd edition. (English translation: *Wavelets: Algorithms and Applications*, SIAM).

Wickerhauser, M.V. (1991), "INRIA lectures on wavelet packet algorithms," *Proceedings ondelettes et paquets d'ondes*, 17–21 June, Rocquencourt, France, pp. 31–99.

Wickerhauser, M.V. (1994), *Adapted wavelet analysis from theory to software Algorithms*, A.K. Peters.

## See Also

`wavedec2`, `waveinfo`, `wenergy`, `wpdec`, `wprec2`

# wpdencmp

---

**Purpose** De-noising or compression using wavelet packets

**Syntax**

```
[XD, TREED, PERFO, PERFL2] =  
    wpdencmp(X, SORH, N, 'wname', CRIT, PAR, KEEPAPP)  
[XD, TREED, PERFO, PERFL2] =  
    wpdencmp(TREE, SORH, CRIT, PAR, KEEPAPP)
```

**Description** wpdencmp is a one- or two-dimensional de-noising and compression oriented function.

wpdencmp performs a de-noising or compression process of a signal or an image, using wavelet packet. The ideas and the procedures for de-noising and compression using wavelet packet decomposition are the same as those used in the wavelets framework (see wden and wdencmp for more information).

```
[XD, TREED, PERFO, PERFL2] =  
wpdencmp(X, SORH, N, 'wname', CRIT, PAR, KEEPAPP) returns a de-noised or  
compressed version XD of input signal X (one- or two-dimensional) obtained by  
wavelet packets coefficients thresholding.
```

The additional output argument TREED is the wavelet packet best tree decomposition (see besttree for more information) of XD. PERFL2 and PERFO are  $L^2$  energy recovery and compression scores in percentages.

$PERFL2 = 100 * (\text{vector-norm of WP-cfs of XD} / \text{vector-norm of WP-cfs of X})^2$ .

If X is a one-dimensional signal and 'wname' an orthogonal wavelet, PERFL2 is reduced to

$$\frac{100 \|XD\|^2}{\|X\|^2}$$

SORH ('s' or 'h') is for soft or hard thresholding (see wthresh for more information).

Wavelet packet decomposition is performed at level N and 'wname' is a string containing the wavelet name. Best decomposition is performed using entropy criterion defined by string CRIT and parameter PAR (see wentropy for more information). Threshold parameter is also PAR. If KEEPAPP = 1, approximation coefficients cannot be thresholded; otherwise, they can be.

[XD, TREED, PERFO, PERFL2] = wpdencmp(TREE, SORH, CRIT, PAR, KEEPAPP) has the same output arguments, using the same options as above, but obtained directly from the input wavelet packet tree decomposition TREE (see wpdec for more information) of the signal to be de-noised or compressed.

In addition if CRIT = 'nobest' no optimization is done and the current decomposition is thresholded.

## Examples

```
% The current extension mode is zero-padding (see dwtmode).

% Load original signal.
load sumlichr; x = sumlichr;

% Use wpdencmp for signal compression.
% Find default values (see ddencmp).
[thr, sorh, keepapp, crit] = ddencmp('cmp', 'wp', x)

thr =
    0.5193

sorh =
    h

keepapp =
    1

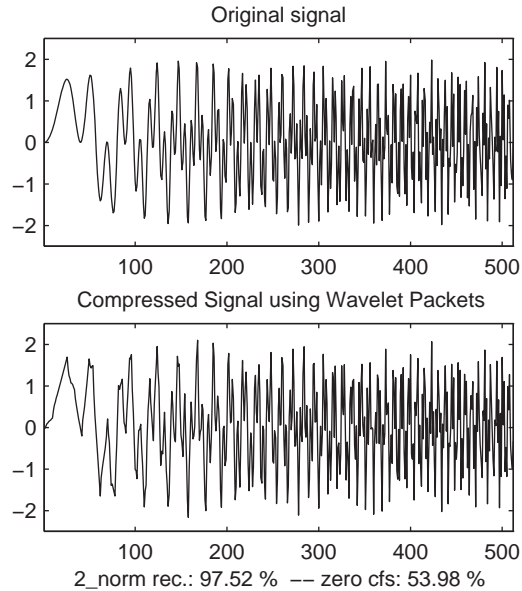
crit =
    threshold

% De-noise signal using global thresholding with
% threshold best basis.
```

# wpdencmp

```
[xc,treed,datad,perf0,perf12] = ...  
wpdencmp(x,sorh,3,'db2',crit,thr,keepapp);
```

```
% Using some plotting commands,  
% the following figure is generated.
```



```
% Load original image.  
load sinsin  
  
% Generate noisy image.  
init = 2055615866; randn('seed',init);  
x = X/18 + randn(size(X));  
  
% Use wpdencmp for image de-noising.  
% Find default values (see dencmp).  
[thr,sorh,keepapp,crit] = dencmp('den','wp',x)  
  
thr =  
    4.9685
```

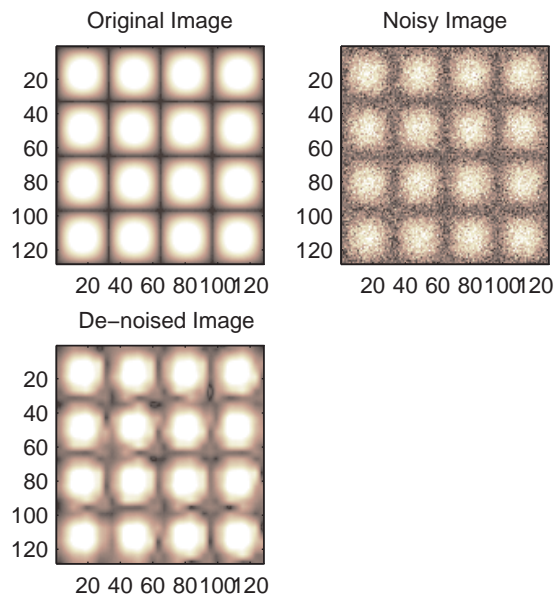
```

sorth =
h
keepapp =
    1

crit =
sure
% De-noise image using global thresholding with
% SURE best basis.
xd = wpdencmp(x,sorth,3,'sym4',crit,thr,keepapp);

% Using some plotting commands,
% the following figure is generated.

```



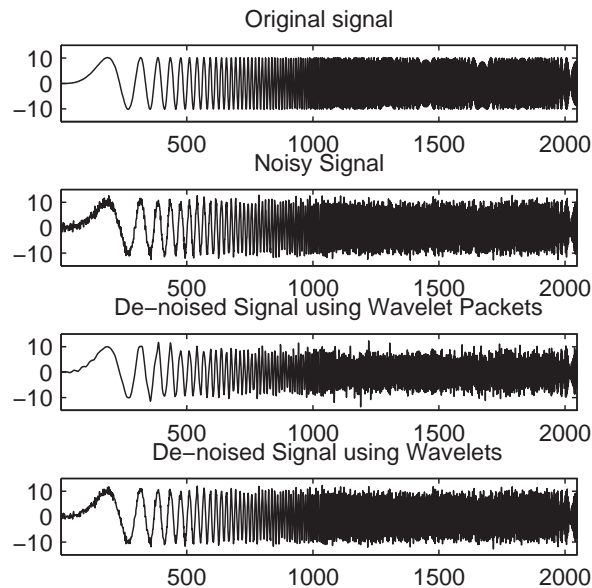
```

% Generate heavy sine and a noisy version of it.
[xref,x] = wnoise(5,11,7,init);

% Use wpdencmp for signal de-noising.
n = length(x);
thr = sqrt(2*log(n*log(n)/log(2)));

```

```
xwpd = wpdencmp(x,'s',4,'sym4','sure',thr,1);  
  
% Compare with wavelet-based de-noising result.  
xwd = wden(x,'rigrsure','s','one',4,'sym4');  
  
% Using some plotting commands,  
% the following figure is generated.
```



## References

- Antoniadis, A.; G. Oppenheim, Eds. (1995), *Wavelets and statistics*, Lecture Notes in Statistics, 103, Springer Verlag.
- 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.
- DeVore, R.A.; B. Jawerth, B.J. Lucier (1992), "Image compression through wavelet transform coding," *IEEE Trans. on Inf. Theory*, vol. 38, No 2, pp. 719–746.
- 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.

Donoho, D.L.; I.M. Johnstone (1994), "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, pp. 425–455.

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.

**See Also**

besttree, ddencmp, wdencmp, wenergy, wpbmpen, wpdec, wpdec2, wthresh

## Purpose

Wavelet packet functions

## Syntax

```
[WPWS,X] = wpfun('wname',NUM,PREC)
[WPWS,X] = wpfun('wname',NUM)
```

## Description

wpfun is a wavelet packet analysis function.

[WPWS,X] = wpfun('wname',NUM,PREC) computes the wavelet packets for a wavelet 'wname' (see wfilters for more information), on dyadic intervals of length  $2^{-\text{PREC}}$ .

PREC must be a positive integer. Output matrix WPWS contains the  $W$  functions of index from 0 to NUM, stored row-wise as  $[W_0; W_1; \dots; W_{\text{NUM}}]$ . Output vector X is the corresponding common X-grid vector.

[WPWS,X] = wpfun('wname',NUM) is equivalent to  
[WPWS,X] = wpfun('wname',NUM,7).

The computation scheme for wavelet packets generation is easy when using an orthogonal wavelet. We start with the two filters of length  $2N$ , denoted  $h(n)$  and  $g(n)$ , corresponding to the wavelet.

Now by induction let us define the following sequence of functions ( $W_n(x)$ ,  $n = 0,1,2,\dots$ ) by

$$W_{2n}(x) = \sqrt{2} \sum_{k=0, \dots, 2N-1} h(k) W_n(2x-k)$$

$$W_{2n}(x) = \sqrt{2} \sum_{k=0, \dots, 2N-1} h(k) W_n(2x-k)$$

where  $W_0(x) = \phi(x)$  is the scaling function and  $W_1(x) = \psi(x)$  is the wavelet function.

For example for the Haar wavelet we have

$$N = 1, h(0) = h(1) = \frac{1}{\sqrt{2}}$$



and

$$g(0) = -g(1) = \frac{1}{\sqrt{2}}$$

The equations become

$$W_{2n}(x) = W_n(2x) + W_n(2x - 1) \quad \text{and} \quad (W_{2n+1}(x) = W_n(2x) - W_n(2x - 1))$$

$W_0(x) = \phi(x)$  is the haar scaling function and  $W_1(x) = \psi(x)$  is the haar wavelet, both supported in  $[0,1]$ .

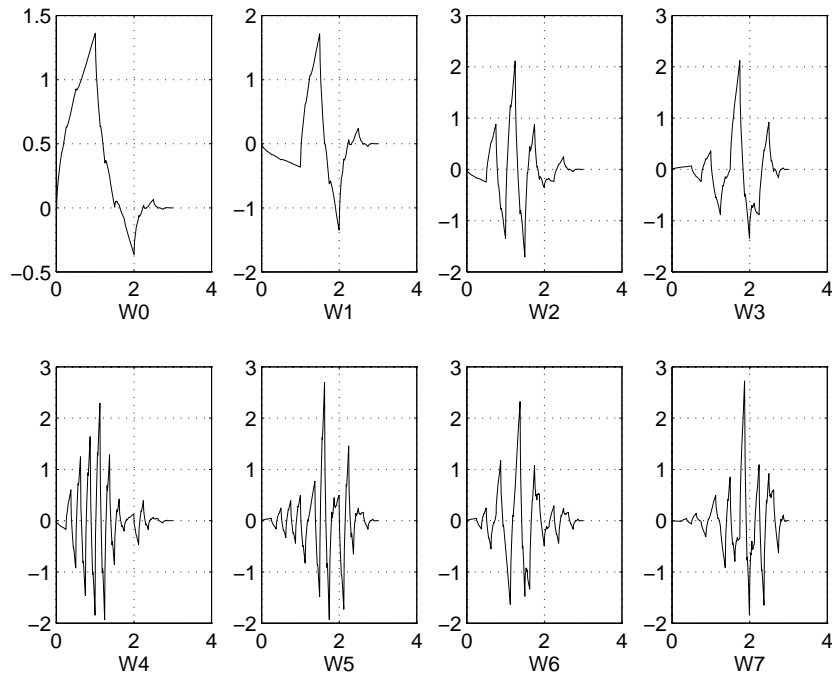
Then we can obtain  $W_{2n}$  by adding two  $1/2$ -scaled versions of  $W_n$  with distinct supports  $[0,1/2]$  and  $[1/2,1]$ , and obtain  $W_{2n+1}$  by subtracting the same versions of  $W_n$ .

Starting from more regular original wavelets, using a similar construction, we obtain smoothed versions of this system of  $W$ -functions, all with support in the interval  $[0, 2N-1]$ .

## Examples

```
% Compute the db2 Wn functions for n = 0 to 7, generating
% the db2 wavelet packets.
[wp,x] = wppfun('db2',7);

% Using some plotting commands,
% the following figure is generated.
```



## References

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.

Meyer, Y. (1993), *Les ondelettes. Algorithmes et applications*, Colin Ed., Paris, 2nd edition. (English translation: *Wavelets: Algorithms and applications*, SIAM).

Wickerhauser, M.V. (1991), "INRIA lectures on wavelet packet algorithms," *Proceedings ondelettes et paquets d'ondes*, 17–21 June, Rocquencourt, France, pp. 31–99.

Wickerhauser, M.V. (1994), *Adapted wavelet analysis from theory to software algorithms*, A.K. Peters.

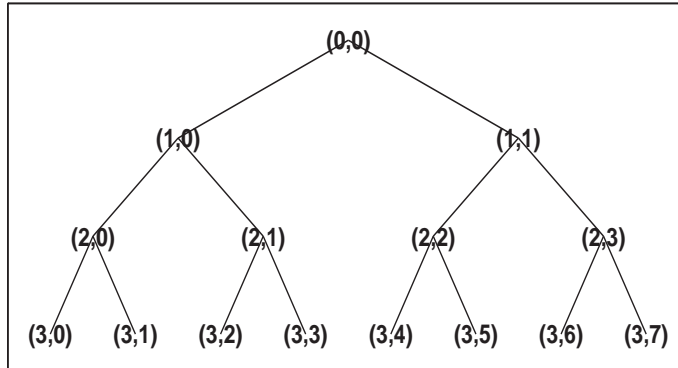
## See Also

wavefun, waveinfo

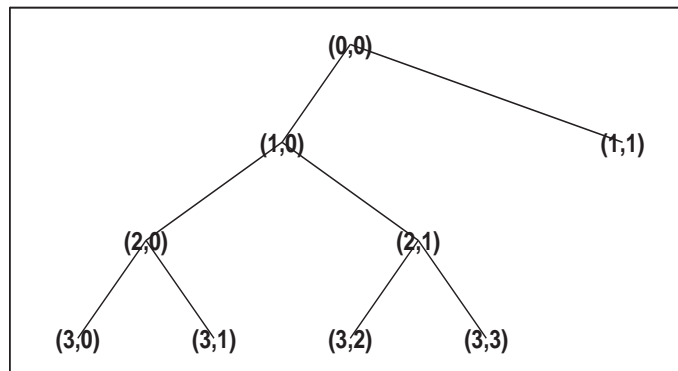
<b>Purpose</b>	Recompose wavelet packet
<b>Syntax</b>	<pre>T = wpjoin(T,N) [T,X] = wpjoin(T,N) T = wpjoin(T) [T,X] = wpjoin(T)</pre>
<b>Description</b>	<p>wpjoin is a one- or two-dimensional wavelet packet analysis function.</p> <p>wpjoin updates the wavelet packet tree after the recomposition of a node.</p> <p>The nodes are numbered from left to right and from top to bottom. The root index is 0.</p> <p>T = wpjoin(T,N) returns the modified wavelet packet tree T corresponding to a recomposition of the node N.</p> <p>[T,X] = wpjoin(T,N) also returns the coefficients of the node.</p> <p>T = wpjoin(T) is equivalent to T = wpjoin(T,0).</p> <p>[T,X] = wpjoin(T) is equivalent to [T,X] = wpjoin(T,0).</p>
<b>Examples</b>	<pre>% The current extension mode is zero-padding (see dwtmode).  % Load signal. load noisdopp; x = noisdopp;  % Decompose x at depth 3 with db1 wavelet packets. wpt = wptdec(x,3,'db1');</pre>

# wpjoin

```
% Plot wavelet packet tree wpt.  
plot(wpt)
```



```
% Recompose packet (1,1) or 2  
wpt = wpjoin(wpt,[1 1]);  
  
% Plot wavelet packet tree wpt.  
plot(wpt)
```



## See Also

wpdec, wpdec2, wpsplt

**Purpose** Reconstruct wavelet packet coefficients

**Syntax** `X = wprcoef(T,N)`

**Description** `wprcoef` is a one- or two-dimensional wavelet packet analysis function.

`X = wprcoef(T,N)` computes reconstructed coefficients of the node `N` of the wavelet packet tree `T`.

`X = wprcoef(T)` is equivalent to `X = wprcoef(T,0)`.

### Examples

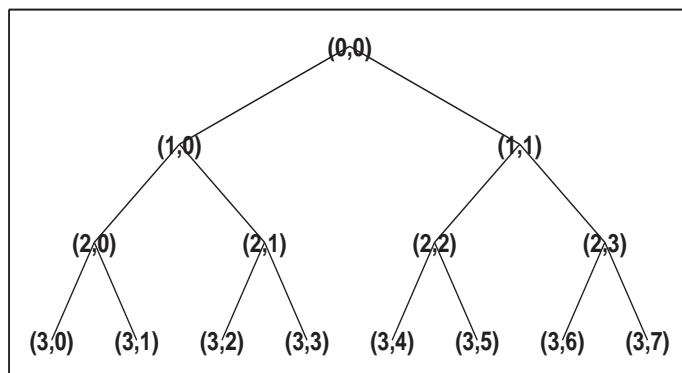
```
% The current extension mode is zero-padding (see dwtmode).
```

```
% Load signal.
load noisdopp; x = noisdopp;
```

```
figure(1); subplot(211);
plot(x); title('Original signal');
```

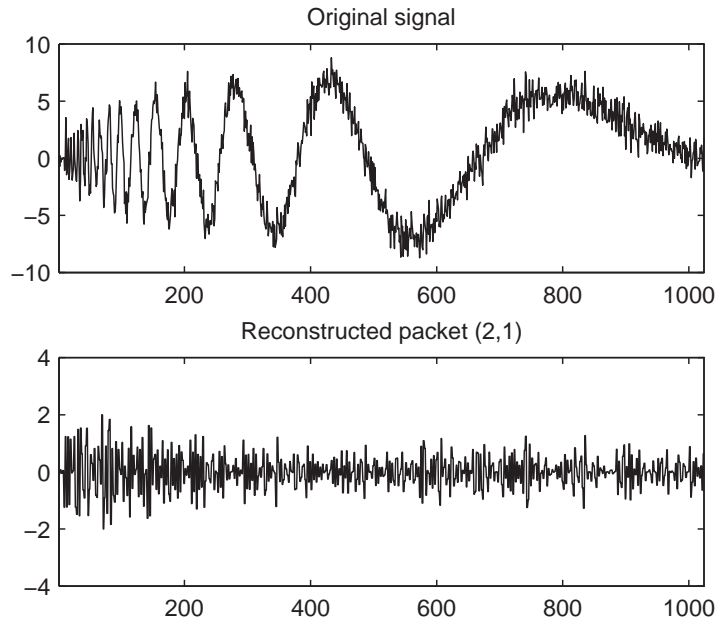
```
% Decompose x at depth 3 with db1 wavelet packets
% using Shannon entropy.
t = wptec(x,3,'db1','shannon');
```

```
% Plot wavelet packet tree.
plot(t)
```



# wprcoef

```
% Reconstruct packet (2,1).  
rcfs = wprcoef(t,[2 1]);  
  
figure(1); subplot(212);  
plot(rcfs); title('Reconstructed packet (2,1)');
```



## See Also

wpdec, wpdec2, wprec, wprec2

**Purpose** Wavelet packet reconstruction 1-D

**Syntax**  $X = \text{wprec}(T)$

**Description** `wprec` is a one-dimensional wavelet packet analysis function.

$X = \text{wprec}(T)$  returns the reconstructed vector  $X$  corresponding to a wavelet packet tree  $T$ .

`wprec` is the inverse function of `wpdec` in the sense that the abstract statement `wprec(wpdec(X, 'wname'))` would give back  $X$ .

**See Also** `wpdec`, `wpdec2`, `wpjoin`, `wprec2`, `wpsplt`

# wprec2

---

**Purpose** Wavelet packet reconstruction 2-D

**Syntax**  $X = \text{wprec2}(T)$

**Description** `wprec2` is a two-dimensional wavelet packet analysis function.  
 $X = \text{wprec2}(T)$  returns the reconstructed matrix  $X$  corresponding to a wavelet packet tree  $T$ .  
`wprec2` is the inverse function of `wpdec2` in the sense that the abstract statement `wprec2(wpdec2(X, 'wname'))` would give back  $X$ .

**Remarks** If  $T$  is obtained from an indexed image analysis or a truecolor image analysis,  $X$  is an  $m$ -by- $n$  matrix or an  $m$ -by- $n$ -by-3 array, respectively.  
For more information on image formats, see the `image` and `imfinfo` reference pages.

**See Also** `wpdec`, `wpdec2`, `wpjoin`, `wprec`, `wpsplt`



**Purpose** Split (decompose) wavelet packet

**Syntax**

```
T = wpsplt(T,N)
[T,cA,cD] = wpsplt(T,N)
[T,cA,cH,cV,cD] = wpsplt(T,N)
```

**Description** `wpsplt` is a one- or two-dimensional wavelet packet analysis function. `wpsplt` updates the wavelet packet tree after the decomposition of a node. `T = wpsplt(T,N)` returns the modified wavelet packet tree `T` corresponding to the decomposition of the node `N`.

For a one-dimensional decomposition,

`[T,cA,cD] = wpsplt(T,N)` with `cA` = approximation and `cD` = detail of node `N`.

For a two-dimensional decomposition,

`[T,cA,cH,cV,cD] = wpsplt(T,N)` with `cA` = approximation and `cH`, `cV`, `c` = horizontal, vertical, and diagonal details of node `N`.

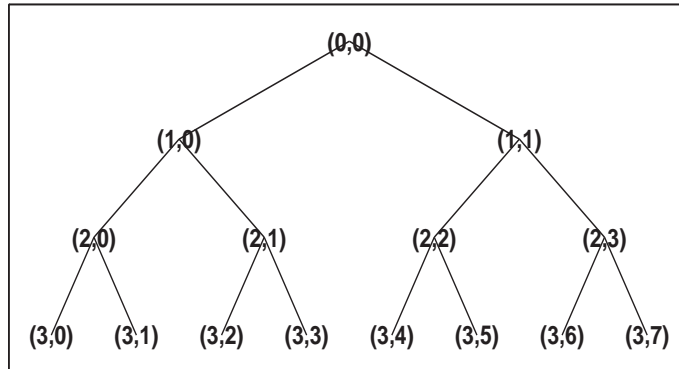
**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

% Load signal.
load noisdopp;
x = noisdopp;

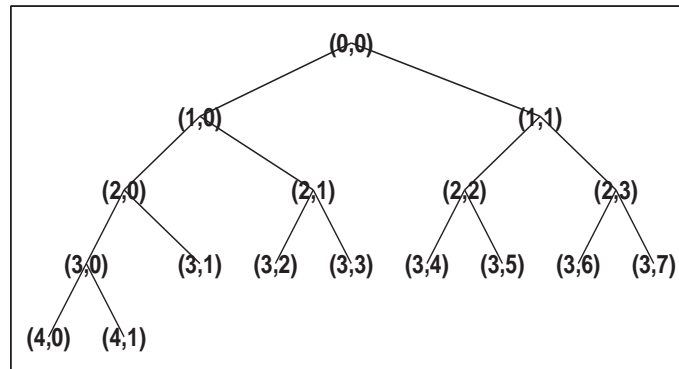
% Decompose x at depth 3 with db1 wavelet packets.
wpt = wptdec(x,3,'db1');
```

```
% Plot wavelet packet tree wpt.  
plot(wpt)
```



```
% Decompose packet (3,0).  
wpt = wpsplt(wpt,[3 0]);  
% or equivalently wpsplt(wpt,7).
```

```
% Plot wavelet packet tree wpt.  
plot(wpt)
```



## See Also

wavedec, wavedec2, wpdec, wpdec2, wpjoin

**Purpose** Wavelet packet coefficients thresholding

**Syntax** `T = wpthcoef(T,KEEPAPP,SORH,THR)`

**Description** `wpthcoef` is a one- or two-dimensional de-noising and compression utility.

`NT = wpthcoef(T,KEEPAPP,SORH,THR)` returns a new wavelet packet tree `NT` obtained from the wavelet packet tree `T` by coefficients thresholding.

If `KEEPAPP = 1`, approximation coefficients are not thresholded; otherwise, they can be thresholded.

If `SORH = 's'`, soft thresholding is applied; if `SORH = 'h'`, hard thresholding is applied (see `wthresh` for more information).

`THR` is the threshold value.

**See Also** `wpdec`, `wpdec2`, `wpdencmp`, `wthresh`

# wptree

---

## Purpose

WPTREE constructor

## Syntax

```
T = wptree(ORDER,DEPTH,X,WNAME,ENT_TYPE,ENT_PAR)
T = wptree(ORDER,DEPTH,X,WNAME)
T = wptree( ... ,USERDATA)
```

## Description

T = wptree(ORDER,DEPTH,X,WNAME,ENT\_TYPE,PARAMETER) returns a complete wavelet packet tree T.

ORDER is an integer representing the order of the tree (the number of “children” of each non terminal node). ORDER must be equal to 2 or 4.

If ORDER = 2, T is a WPTREE object corresponding to a wavelet packet decomposition of the vector (signal) X, at level DEPTH with a particular wavelet WNAME.

If ORDER = 4, T is a WPTREE object corresponding to a wavelet packet decomposition of the matrix (image) X, at level DEPTH with a particular wavelet WNAME.

ENT\_TYPE is a string containing the entropy type and ENT\_PAR is an optional parameter used for entropy computation ( see wentropy, wpdec, or wpdec2 for more information).

```
T = wptree(ORDER,DEPTH,X,WNAME) is equivalent to
T = wptree(ORDER,DEPTH,X,WNAME, 'shannon')
```

With T = wptree(ORDER,DEPTH,X,WNAME,ENT\_TYPE,ENT\_PAR,USERDATA) you may set a userdata field.

The function wptree returns a WPTREE object.

For more information on object fields, see the get function or type

```
help wptree/get
```

Class WPTREE (Parent class: DTREE)

## Fields

```
'dtree'    DTREE parent object
'wavInfo'  Structure (wavelet information)
'entInfo'  Structure (entropy information)
```

The wavelet information structure, 'wavInfo', contains

```
'wavName' Wavelet name
'Lo_D'     Low Decomposition filter
'Hi_D'     High Decomposition filter
'Lo_R'     Low Reconstruction filter
'Hi_R'     High Reconstruction filter
```

The entropy information structure, 'entInfo', contains

```
'entName' Entropy name
'entPar'  Entropy parameter
```

Fields from the DTREE parent object:

```
'allNI'   All nodes information
```

'allNI' is an array of size nbnode by 5, which contains

```
ind      Index
size     Size of data
ent      Entropy
ento     Optimal entropy
```

Each line is built based on the following scheme:

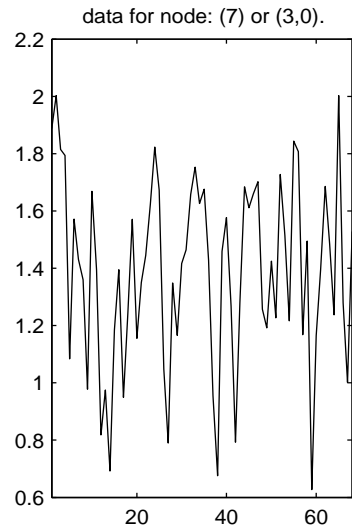
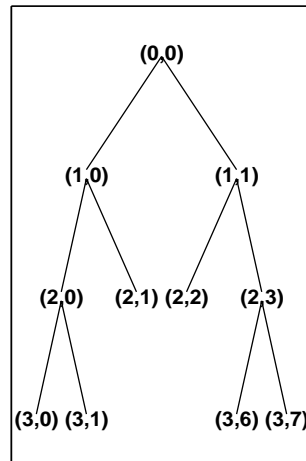


## Examples

```
% Create a wavelet packet tree.
x = rand(1,512);
t = wptree(2,3,x,'db3');
t = wpjoin(t,[4;5]);
```

```
% Plot tree t4.  
plot(t);
```

```
% Click the node (3,0), (see the plot function).
```



## See Also

dtree, ntree

**Purpose** Plot wavelet packets colored coefficients

**Syntax** `wpviewcf(T,CMODE)`  
`wpviewcf(T,CMODE,NBCOL)`

**Description** `wpviewcf(T,CMODE)` plots the colored coefficients for the terminal nodes of the tree `T`.

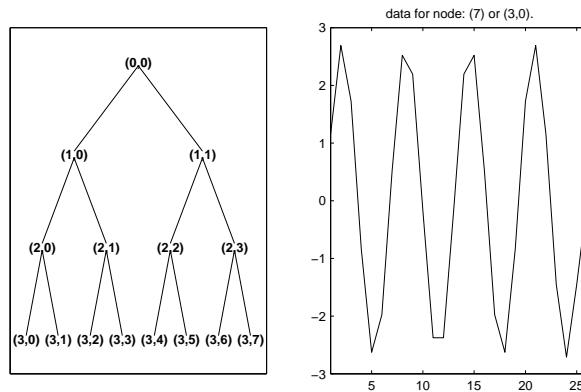
`T` is a wavelet packet tree and `CMODE` is an integer, which represents the color mode. The color modes are listed in the table below.

Color Mode	Description
1	Frequency order – Global coloration – Absolute values
2	Frequency order – By level – Absolute values
3	Frequency order – Global coloration – Values
4	Frequency order – By level coloration – Values
5	Natural order – Global coloration – Absolute values
6	Natural order – By level – Absolute values
7	Natural order – Global coloration – Values
8	Natural order – By level coloration – Values

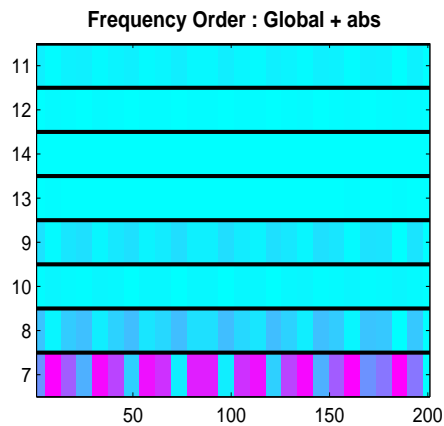
`wpviewcf(T,CMODE,NBCOL)` uses `NBCOL` colors.

## Examples

```
% Create a wavelet packet tree.  
x = sin(8*pi*[0:0.005:1]);  
t = wpdec(x,3,'db1');  
  
% Plot tree t.  
% Click the node (3,0), (see the plot function)  
plot(t);
```



```
% Plot the colored wavelet packet coefficients.  
wpviewcf(t,1);
```



## See Also

[wpdec](#)



**Purpose** Reconstruct single branch from 1-D wavelet coefficients

**Syntax**

```
X = wrcoef('type',C,L,'wname',N)
X = wrcoef('type',C,L,Lo_R,Hi_R,N)
X = wrcoef('type',C,L,'wname')
X = wrcoef('type',C,L,Lo_R,Hi_R)
```

**Description** `wrcoef` reconstructs the coefficients of a one-dimensional signal, given a wavelet decomposition structure (`C` and `L`) and either a specified wavelet (`'wname'`, see `wfilters` for more information) or specified reconstruction filters (`Lo_R` and `Hi_R`).

`X = wrcoef('type',C,L,'wname',N)` computes the vector of reconstructed coefficients, based on the wavelet decomposition structure [`C`, `L`] (see `wavedec` for more information), at level `N`. `'wname'` is a string containing the wavelet name.

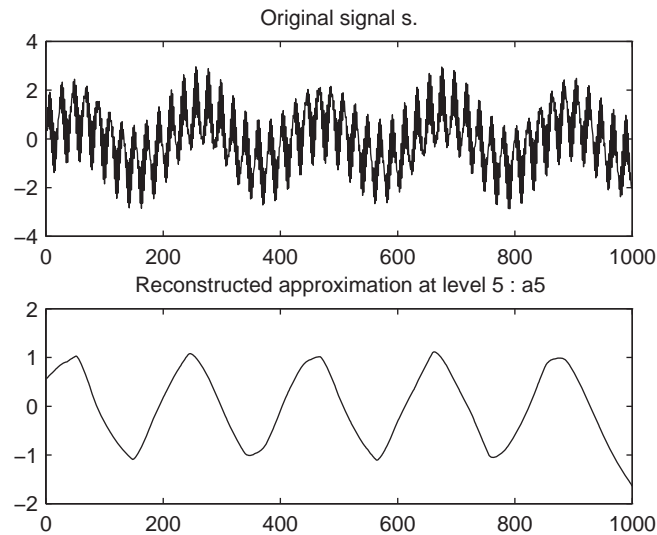
Argument `'type'` determines whether approximation (`'type' = 'a'`) or detail (`'type' = 'd'`) coefficients are reconstructed. When `'type' = 'a'`, `N` is allowed to be 0; otherwise, a strictly positive number `N` is required. Level `N` must be an integer such that  $N \leq \text{length}(L) - 2$ .

`X = wrcoef('type',C,L,Lo_R,Hi_R,N)` computes coefficients as above, given the reconstruction filters you specify.

`X = wrcoef('type',C,L,'wname')` and `X = wrcoef('type',C,L,Lo_R,Hi_R)` reconstruct coefficients of maximum level  $N = \text{length}(L) - 2$ .

## Examples

```
% The current extension mode is zero-padding (see dwtmode).  
  
% Load a one-dimensional signal.  
load sumsin; s = sumsin;  
  
% Perform decomposition at level 5 of s using sym4.  
[c,l] = wavedec(s,5,'sym4');  
  
% Reconstruct approximation at level 5,  
% from the wavelet decomposition structure [c,l].  
a5 = wrcoef('a',c,l,'sym4',5);  
  
% Using some plotting commands,  
% the following figure is generated.
```



## See Also

appcoef, detcoef, wavedec

<b>Purpose</b>	Reconstruct single branch from 2-D wavelet coefficients
<b>Syntax</b>	<pre>X = wrcoef2('type',C,S,'wname',N) X = wrcoef2('type',C,S,Lo_R,Hi_R,N) X = wrcoef2('type',C,S,'wname') X = wrcoef2('type',C,S,Lo_R,Hi_R)</pre>
<b>Description</b>	<p>wrcoef2 is a two-dimensional wavelet analysis function. wrcoef2 reconstructs the coefficients of an image.</p> <p><code>X = wrcoef2('type',C,S,'wname',N)</code> computes the matrix of reconstructed coefficients of level N, based on the wavelet decomposition structure [C,S] (see wavedec2 for more information).</p> <p>'wname' is a string containing the name of the wavelet (see wfilters for more information). If 'type' = 'a', approximation coefficients are reconstructed; otherwise if 'type' = 'h' ('v' or 'd', respectively), horizontal (vertical or diagonal, respectively) detail coefficients are reconstructed.</p> <p>Level N must be an integer such that <math>0 \leq N \leq \text{size}(S,1) - 2</math> if 'type' = 'a' and such that <math>1 \leq N \leq \text{size}(S,1) - 2</math> if 'type' = 'h', 'v', or 'd'.</p> <p>Instead of giving the wavelet name, you can give the filters.</p> <p>For <code>X = wrcoef2('type',C,S,Lo_R,Hi_R,N)</code>, Lo_R is the reconstruction low-pass filter and Hi_R is the reconstruction high-pass filter.</p> <p><code>X = wrcoef2('type',C,S,'wname')</code> or <code>X = wrcoef2('type',C,S,Lo_R,Hi_R)</code> reconstruct coefficients of maximum level <math>N = \text{size}(S,1) - 2</math>.</p>
<b>Remarks</b>	<p>If C and S are obtained from an indexed image analysis (respectively a truecolor image analysis) then X is an m-by-n matrix (respectively an m-by-n-by-3 array).</p> <p>For more information on image formats, see the reference pages of image and imfinfo functions.</p>
<b>Examples</b>	<pre>% The current extension mode is zero-padding (see dwtmode).  % Load an image. load woman; % X contains the loaded image.</pre>

## wrcoef2

---

```
% Perform decomposition at level 2  
% of X using sym5.  
[c,s] = wavedec2(X,2,'sym5');
```

```
% Reconstruct approximations at
% levels 1 and 2, from the wavelet
% decomposition structure [c,s].
a1 = wrcoef2('a',c,s,'sym5',1);
a2 = wrcoef2('a',c,s,'sym5',2);

% Reconstruct details at level 2,
% from the wavelet decomposition
% structure [c,s].
% 'h' is for horizontal,
% 'v' is for vertical,
% 'd' is for diagonal.
hd2 = wrcoef2('h',c,s,'sym5',2);
vd2 = wrcoef2('v',c,s,'sym5',2);
dd2 = wrcoef2('d',c,s,'sym5',2);

% All these images are of same size sX.
sX = size(X)

sX =
    256 256

sa1 = size(a1)

sa1 =
    256 256

shd2 = size(hd2)

shd2 =
    256 256
```

**See Also**

appcoef2, detcoef2, wavedec2

# wrev

---

**Purpose** Flip vector

**Syntax** `Y = wrev(X)`

**Description** `wrev` is a general utility.  
`Y = wrev(X)` reverses the vector `X`.

**Examples**

```
% Set simple vector.  
v = [1 2 3];  
  
% Reverse v.  
wrev(v)  
  
ans =  
    3    2    1  
  
% Reverse v transpose.  
wrev(v')  
  
ans =  
    3  
    2  
    1
```

**See Also** `fliplr`, `flipud`

**Purpose** Write values in WPTREE fields

**Syntax**

```
T = write(T,'cfs',NODE,COEFS)
T = write(T,'cfs',N1,CFS1,'cfs',N2,CFS2, ...)
```

**Description** `T = write(T,'cfs',NODE,COEFS)` writes coefficients for the terminal node NODE.

`T = write(T,'cfs',N1,CFS1,'cfs',N2,CFS2, ...)` writes coefficients CFS1, CFS2, ... for the terminal nodes N1, N2, ...

---

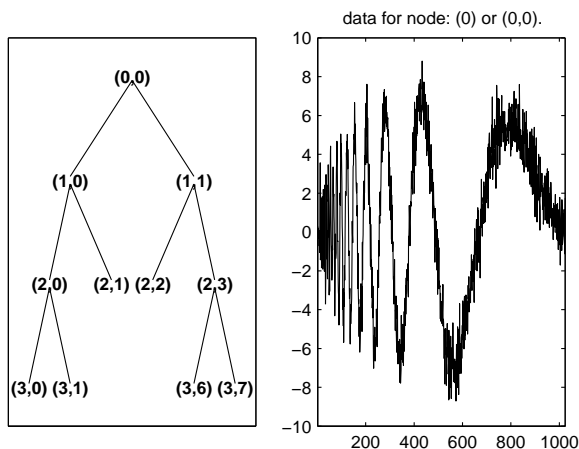
**Caution** The coefficients values must have the suitable size. You can use `S = read(T,'sizes',NODE)` or `S = read(T,'sizes',[N1;N2; ...])` in order to get those sizes.

---

## Examples

```
% Create a wavelet packet tree.
load noisdopp; x = noisdopp;
t = wpdec(x,3,'db3');
t = wpjoin(t,[4;5]);

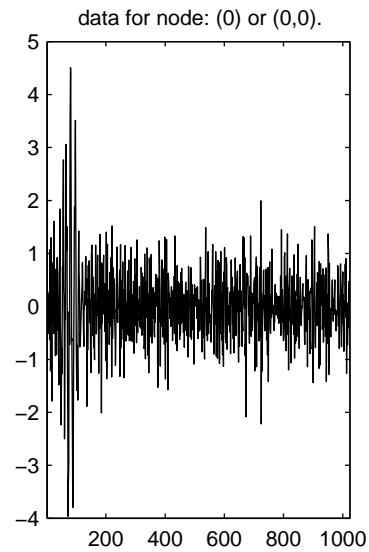
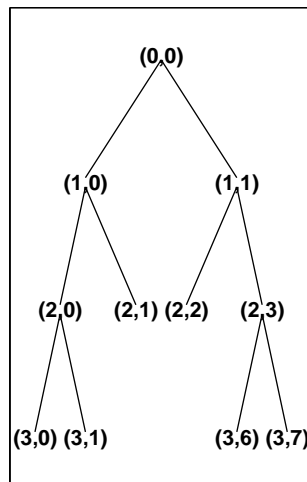
% Plot tree t and click the node (0,0) (see the plot function).
plot(t);
```



# write

```
% Write values.
sNod = read(t,'sizes',[4,5,7]);
cfs4 = zeros(sNod(1,:));
cfs5 = zeros(sNod(2,:));
cfs7 = zeros(sNod(3,:));
t = write(t,'cfs',4,cfs4,'cfs',5,cfs5,'cfs',7,cfs7);

% Plot tree t and click the node (0,0) (see the plot function).
plot(t)
```



## See Also

disp, get, read, set



**Purpose** Scalogram for continuous wavelet transform

**Syntax**  
`SC = wscalogram(TYPEPLOT, COEFS)`  
`SC = wscalogram(TYPEPLOT, COEFS, 'PropName1', PropVal1, ...)`

**Description** `SC = wscalogram(TYPEPLOT, COEFS)` computes the scalogram `SC` which represents the percentage of energy for each coefficient. `COEFS` is the matrix of the continuous wavelet coefficients (see `cwt`).

The scalogram is obtained by computing:

$$S = \text{abs}(\text{coefs}.*\text{coefs}); \quad SC = 100*S./\text{sum}(S(:))$$

When `TYPEPLOT` is equal to `'image'`, a scaled image of scalogram is displayed. When `TYPEPLOT` is equal to `'contour'`, a contour representation of scalogram is displayed. Otherwise, the scalogram is returned without plot representation.

`SC = wscalogram(TYPEPLOT, COEFS, 'PropName1', PropVal1, ...)` allows you to modify some properties. The valid choices for `PropName` are:

<code>'scales'</code>	Scales used for the CWT.
<code>'ydata'</code>	Signal used for the CWT.
<code>'xdata'</code>	$x$ values corresponding to the signal values.
<code>'power'</code>	Positive real value. Default value is zero.

If `power > 0`, coefficients are first normalized

$$\text{coefs}(k, :) = \text{coefs}(k, :)/(\text{scales}(k)^{\text{power}})$$

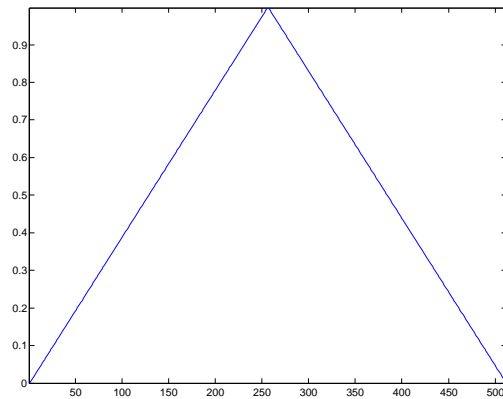
and then the scalogram is computed as explained above.

**Examples**

```
% Compute signal s
t = linspace(-1,1,512);
s = 1-abs(t);

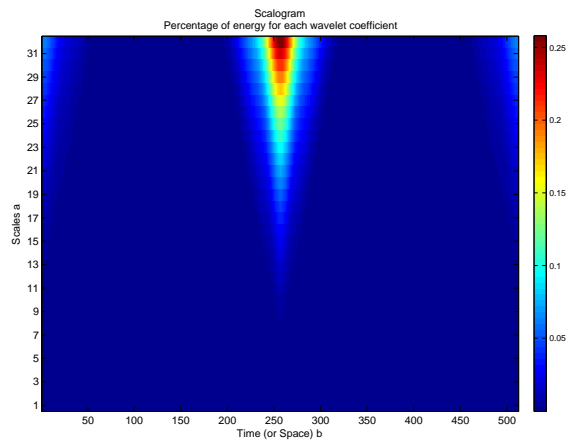
% Plot signal s
figure;
plot(s), axis tight
```

# wscalogram

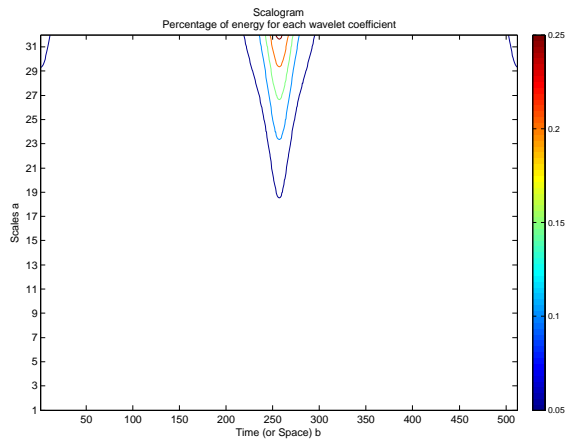


```
% Compute coefficients COEFS using cwt  
COEFS = cwt(s,1:32,'cgau4');
```

```
% Compute and plot the scalogram (image option)  
figure;  
SC = wscalogram('image',COEFS);
```



```
% Compute and plot the scalogram (contour option)  
figure;  
SC = wscalogram('contour',COEFS);
```



**See Also**

cwt

# wtbo

---

**Purpose** WTBO constructor

**Syntax** OBJ = wtbo  
OBJ = wtbo(USERDATA)

**Description** OBJ = wtbo returns a WTBO object. Any object in the Wavelet Toolbox™ software is parented by a WTBO object.

With OBJ = wtbo(USERDATA) you can set a userdata field.

Class WTBO (Parent class: none)

## Fields

wtboInfo Object information (not used in the current version of the toolbox)

ud Userdata field

**Purpose** Wavelet Toolbox manager

**Syntax**  
 wtbxmngr  
 wtbxmngr(OPTION)  
 V = wtbxmngr('version')

**Description** wtbxmngr or wtbxmngr('version') displays the current version of Wavelet Toolbox software.

wtbxmngr(OPTION) sets a toolbox option. Available options are

Option	Description
'LargeFonts'	Sets the size of future-created figures to use large fonts.
'DefaultSize'	Restores the default figure size for future-created figures.
'FigRatio'	Returns the current figure ratio value.
'FigRatio',ratio	Changes the size of future-created figures by multiplying the default size by the specified ratio, where ratio must be between 0.75 and 1.25.

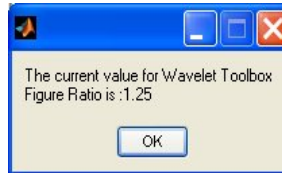
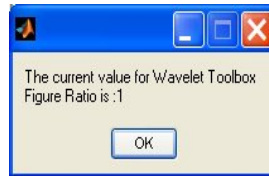
V = wtbxmngr('version') saves the current version of the toolbox to variable V.

## Examples

```
wtbxmngr('version')
```

```
*****
** Wavelet Toolbox Version: V3.1 **
*****
```

```
wtbxmngr('FigRatio')           % Display the current figure ratio
wtbxmngr('FigRatio',1.25)      % Set the figure ratio to 1.25
wtbxmngr('FigRatio')           % Display the current figure ratio
wtbxmngr('DefaultSize')        % Return to the default figure ratio
```



---

<b>Purpose</b>	Wavelet coefficient thresholding 1-D
<b>Syntax</b>	<pre>NC = wthcoef('d',C,L,N,P) NC = wthcoef('d',C,L,N) NC = wthcoef('a',C,L) NC = wthcoef('t',C,L,N,T,SORH)</pre>
<b>Description</b>	<p>wthcoef is a one-dimensional de-noising and compression oriented function.</p> <p>NC = wthcoef('d',C,L,N,P) returns coefficients obtained from the wavelet decomposition structure [C,L] (see wavedec for more information), by rate compression defined in vectors N and P. N contains the detail levels to be compressed and P the corresponding percentages of lower coefficients to be set to zero. N and P must be of same length. Vector N must be such that <math>1 \leq N(i) \leq \text{length}(L) - 2</math>.</p> <p>NC = wthcoef('d',C,L,N) returns coefficients obtained from [C,L] by setting all the coefficients of detail levels defined in N to zero.</p> <p>NC = wthcoef('a',C,L) returns coefficients obtained by setting approximation coefficients to zero.</p> <p>NC = wthcoef('t',C,L,N,T,SORH) returns coefficients obtained from the wavelet decomposition structure [C,L] by soft (if SORH = 's') or hard (if SORH = 'h') thresholding (see wthresh for more information) defined in vectors N and T. N contains the detail levels to be thresholded and T the corresponding thresholds. N and T must be of the same length.</p> <p>[NC,L] is the modified wavelet decomposition structure.</p>
<b>See Also</b>	wavedec, wthresh

# wthcoef2

---

**Purpose** Wavelet coefficient thresholding 2-D

**Syntax**  
NC = wthcoef2('type',C,S,N,T,SORH)  
NC = wthcoef2('type',C,S,N)  
NC = wthcoef2('a',C,S)  
NC = wthcoef2('t',C,S,N,T,SORH)

**Description** wthcoef2 is a two-dimensional de-noising and compression oriented function.

For 'type' = 'h' ('v' or 'd'), NC = wthcoef2('type',C,S,N,T,SORH) returns the horizontal (vertical or diagonal, respectively) coefficients obtained from the wavelet decomposition structure [C,S] (see wavedec2 for more information), by soft (if SORH = 's') or hard (if SORH = 'h') thresholding defined in vectors N and T. N contains the detail levels to be thresholded and T the corresponding thresholds. N and T must be of the same length. The vector N must be such that  $1 \leq N(i) \leq \text{size}(S,1) - 2$ .

For 'type' = 'h' ('v' or 'd'), NC = wthcoef2('type',C,S,N) returns the horizontal (vertical or diagonal, respectively) coefficients obtained from [C,S] by setting all the coefficients of detail levels defined in N to zero.

NC = wthcoef2('a',C,S) returns the coefficients obtained by setting approximation coefficients to zero.

NC = wthcoef2('t',C,S,N,T,SORH) returns the detail coefficients obtained from the wavelet decomposition structure [C,S] by soft (if SORH = 's') or hard (if SORH = 'h') thresholding (see wthresh for more information) defined in vectors N and T. N contains the detail levels to be thresholded and T the corresponding thresholds which are applied in the three detail orientations. N and T must be of the same length.

[NC,S] is the modified wavelet decomposition structure.

**See Also** wavedec2, wthresh



**Purpose** Soft or hard thresholding

**Syntax** `Y = wthresh(X,SORH,T)`

**Description** `Y = wthresh(X,SORH,T)` returns the soft (if `SORH = 's'`) or hard (if `SORH = 'h'`) thresholding of the input vector or matrix `X`. `T` is the threshold value.

`Y = wthresh(X,'s',T)` returns  $Y = \text{sign}(X) \cdot (|X| - T)_+$ , soft thresholding is wavelet shrinkage ( $(x)_+ = 0$  if  $x < 0$ ;  $(x)_+ = x$ , if  $x \geq 0$ ).

`Y = wthresh(X,'h',T)` returns  $Y = X \cdot 1_{(|X| > T)}$ , hard thresholding is cruder.

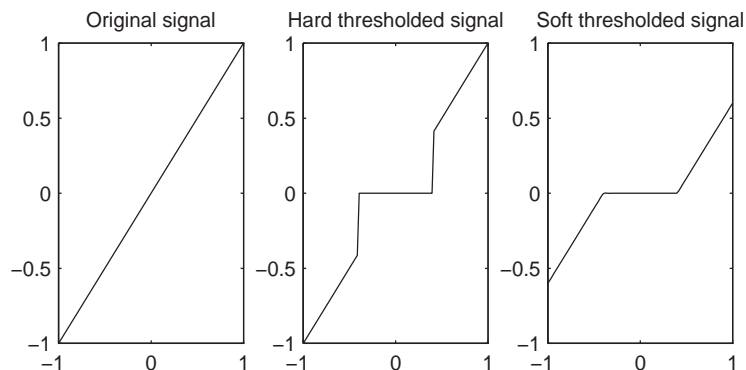
### Examples

```
% Generate signal and set threshold.
y = linspace(-1,1,100);
thr = 0.4;

% Perform hard thresholding.
ythard = wthresh(y,'h',thr);

% Perform soft thresholding.
ytsoft = wthresh(y,'s',thr);

% Using some plotting commands,
% the following figure is generated.
```



### See Also

`wden`, `wdencomp`, `wpdencomp`

# wthrmngr

---

**Purpose** Threshold settings manager

**Syntax** THR = wthrmngr(OPTION,METHOD,VARARGIN)

**Description** THR = wthrmngr(OPTION,METHOD,VARARGIN) returns a global threshold or level dependent thresholds depending on OPTION. The inputs, VARARGIN, depend on the OPTION and METHOD values.

This M-file returns the thresholds used throughout the Wavelet Toolbox™ software for de-noising and compression tools (command line M-files or GUI tools).

Valid options for the METHOD parameter are listed in the table below.

METHOD	Description
'scarcehi'	See wdcbm or wdcbm2 when used with 'high' predefined value of parameter M.
'scarceme'	See wdcbm or wdcbm2 when used with 'medium' predefined value of parameter M.
'scarcelo'	See wdcbm or wdcbm2 when used with 'low' predefined value of parameter M.
'sqtwolog'	See 'sqtwolog' option in thselect, and see also wden.
'sqtwologuwn'	See 'sqtwolog' option in thselect, and see also wden when used with 'sln' option.
'sqtwologswn'	See 'sqtwolog' option in thselect, and see also wden when used with 'mln' option.
'rigsure'	See 'rigsure' option in thselect, and see also wden.
'heursure'	See 'heursure' option in thselect, and see also wden.
'minimaxi'	See 'minimaxi' option in thselect, and see also wden.
'penalhi'	See wbpmpen or wbpmpen when used with 'high' value of parameter ALPHA.

METHOD	Description (Continued)
'penalme'	See wbmopen or wpbmopen when used with 'medium' value of parameter ALPHA.
'penallo'	See wbmopen or wpbmopen when used with 'low' value of parameter ALPHA.
'rem_n0'	This option returns a threshold close to 0. A typical THR value is $\text{median}(\text{abs}(\text{coefficients}))$ .
'bal_sn'	This option returns a threshold such that the percentages of retained energy and number of zeros are the same.
'sqrtbal_sn'	This option returns a threshold equal to the square root of the value such that the percentages of retained energy and number of zeros are the same.

## Discrete Wavelet 1-D Options

**Compression using a global threshold.**  $X$  is the signal to be compressed and  $[C,L]$  is the wavelet decomposition structure of the signal to be compressed.

```
THR = wthrmngr('dw1dcompGBL','rem_n0',X)
```

```
THR = wthrmngr('dw1dcompGBL','bal_sn',X)
```

**Compression using level dependent thresholds.**  $X$  is the signal to be compressed and  $[C,L]$  is the wavelet decomposition structure of the signal to be compressed.

ALFA is a sparsity parameter (see wdcbm for more information).

```
THR = wthrmngr('dw1dcompLVL','scarcehi',C,L,ALFA)
ALFA must be such that  $2.5 < ALFA < 10$ 
```

```
THR = wthrmngr('dw1dcompLVL','scarce',C,L,ALFA)
ALFA must be such that  $1.5 < ALFA < 2.5$ 
```

```
THR = wthrmngr('dw1dcompLVL','scarcelo',C,L,ALFA)
ALFA must be such that  $1 < ALFA < 2$ 
```

**De-noising using level dependent thresholds.** [C,L] is the wavelet decomposition structure of the signal to be de-noised, SCAL defines the multiplicative threshold rescaling (see wden for more information) and ALFA is a sparsity parameter (see wbmpe for more information).

```
THR = wthrmngr('dw1ddenoLVL','sqrtwolog',C,L,SCAL)
```

```
THR = wthrmngr('dw1ddenoLVL','rigrsure',C,L,SCAL)
```

```
THR = wthrmngr('dw1ddenoLVL','heursure',C,L,SCAL)
```

```
THR = wthrmngr('dw1ddenoLVL','minimaxi',C,L,SCAL)
```

```
THR = wthrmngr('dw1ddenoLVL','penalhi',C,L,ALFA)
ALFA must be such that  $2.5 < ALFA < 10$ 
```

```
THR = wthrmngr('dw1ddenoLVL','penalme',C,L,ALFA)
ALFA must be such that  $1.5 < ALFA < 2.5$ 
```

```
THR = wthrmngr('dw1ddenoLVL','penallo',C,L,ALFA)
ALFA must be such that  $1 < ALFA < 2$ 
```

## Discrete Stationary Wavelet 1-D Options

**De-noising using level dependent thresholds.** SWTDEC is the stationary wavelet decomposition structure of the signal to be de-noised, SCAL defines the multiplicative threshold rescaling (see wden for more information) and ALFA is a sparsity parameter (see wbmpe for more information).

```
THR = wthrmngr('sw1ddenoLVL',METHOD,SWTDEC,SCAL)
```

```
THR = wthrmngr('sw1ddenoLVL',METHOD,SWTDEC,ALFA)
```

The options for METHOD are the same as in the 'dw1ddenoLVL' case.

## Discrete Wavelet 2-D Options

**Compression using a global threshold.** X is the image to be compressed and [C,S] is the wavelet decomposition structure of the image to be compressed.

```
THR = wthrmngr('dw2dcompGBL','rem_n0',X)
```

```
THR = wthrmngr('dw2dcompGBL','bal_sn',C,S)
```

```
THR = wthrmngr('dw2dcompGBL','sqrtbal_sn',C,S)
```

**Compression using level dependent thresholds.** X is the image to be compressed and [C,S] is the wavelet decomposition structure of the image to be compressed. ALFA is a sparsity parameter (see wdcbm2 for more information).

```
THR = wthrmngr('dw2dcompLVL','scarcehi',C,L,ALFA)
ALFA must be such that 2.5 < ALFA < 10
```

```
THR = wthrmngr('dw2dcompLVL','scarceme',C,L,ALFA)
ALFA must be such that 1.5 < ALFA < 2.5
```

```
THR = wthrmngr('dw2dcompLVL','scarcelo',C,L,ALFA)
ALFA must be such that 1 < ALFA < 2
```

**De-noising using level dependent thresholds.** [C,S] is the wavelet decomposition structure of the image to be de-noised, SCAL defines the multiplicative threshold rescaling (see wden for more information) and ALFA is a sparsity parameter (see wbmpen for more information).

```
THR = wthrmngr('dw2ddenoLVL','penalhi',C,S,ALFA)
ALFA must be such that 2.5 < ALFA < 10
```

```
THR = wthrmngr('dw2ddenoLVL','penalme',C,L,ALFA)
ALFA must be such that 1.5 < ALFA < 2.5
```

```
THR = wthrmngr('dw2ddenoLVL','penal1o',C,L,ALFA)
ALFA must be such that 1 < ALFA < 2
```

```
THR = wthrmngr('dw2ddenoLVL','sqrtwo1og',C,S,SCAL)
```

```
THR = wthrmngr('dw2ddenoLVL','sqrtbal_sn',C,S)
```

## Discrete Stationary Wavelet 2-D Options

**De-noising using level dependent thresholds.** SWTDEC is the stationary wavelet decomposition structure of the image to be de-noised, SCAL defines the multiplicative threshold rescaling (see wden for more information) and ALFA is a sparsity parameter (see wbmpen for more information).

```
THR = wthrmngr('sw2ddenoLVL',METHOD,SWTDEC,SCAL)
```

```
THR = wthrmngr('sw2ddenoLVL',METHOD,SWTDEC,ALFA)
```

The options for METHOD are the same as in the 'dw2ddenoLVL' case.

## Discrete Wavelet Packet 1-D Options

**Compression using a global threshold.** X is the signal to be compressed and WPT is the wavelet packet decomposition structure of the signal to be compressed.

```
THR = wthrmngr('wp1dcompGBL','bal_sn',WPT)
```

```
THR = wthrmngr('wp1dcompGBL','rem_n0',X)
```

**De-noising using a global threshold.** WPT is the wavelet packet decomposition structure of the signal to be de-noised.

```
THR = wthrmngr('wp1ddenoGBL','sqrtwologuwn',WPT)
```

```
THR = wthrmngr('wp1ddenoGBL','sqrtwologswn',WPT)
```

```
THR = wthrmngr('wp1ddenoGBL','bal_sn',WPT)
```

```
THR = wthrmngr('wp1ddenoGBL','penalhi',WPT)  
see wbmpen with ALFA = 6.25
```

```
THR = wthrmngr('wp1ddenoGBL','penalme',WPT)  
see wbmpen with ALFA = 2
```

```
THR = wthrmngr('wp1ddenoGBL','penallo',WPT)  
see wbmpen with ALFA = 1.5
```

## Discrete Wavelet Packet 2-D Options

**Compression using a global threshold.** X is the image to be compressed and WPT is the wavelet packet decomposition structure of the image to be compressed.

```
THR = wthrmngr('wp2dcompGBL','bal_sn',WPT)
```

```
THR = wthrmngr('wp2dcompGBL','rem_n0',X)
```

```
THR = wthrmngr('wp2dcompGBL','sqrtbal_sn',WPT)
```

**De-noising using a global threshold.** WPT is the wavelet packet decomposition structure of the image to be de-noised.

THR = wthrmngr('wp2ddenoGBL', 'sqtwoologuwn', WPT)

THR = wthrmngr('wp2ddenoGBL', 'sqtwoologswn', WPT)

THR = wthrmngr('wp2ddenoGBL', 'sqrta1\_sn', WPT)

THR = wthrmngr('wp2ddenoGBL', 'penalhi', WPT)  
see wbmopen with ALFA = 6.25

THR = wthrmngr('wp2ddenoGBL', 'penalme', WPT)  
see wbmopen with ALFA = 2

THR = wthrmngr('wp2ddenoGBL', 'penallo', WPT)  
see wbmopen with ALFA = 1.5

# wtreemgr

---

**Purpose** NTREE manager

**Syntax** VARARGOUT = wtreemgr(OPT,T,VARARGIN)

**Description** wtreemgr is a tree management utility.

This function returns information on the tree T depending on the value of the OPT parameter.

Allowed values for OPT are listed in the table below.

'allnodes'	Tree nodes
'isnode'	True for existing node
'istnode'	True for terminal nodes
'nodeasc'	Node ascendants
'nodedesc'	Node descendants
'nodepar'	Node parent
'ntnode'	Number of terminal nodes
'tnodes'	Terminal nodes
'leaves'	Terminal nodes
'noleaves'	Not terminal nodes
'order'	Tree order
'depth'	Tree depth

The functionality associated with the OPT value you specify is described in the functions listed in the “See Also” section.

**See Also** allnodes, isnode, istnode, leaves, nodeasc, nodedesc, nodepar, noleaves, ntnode, tnodes, treedpth, treeord



<b>Purpose</b>	Find variance change points
<b>Syntax</b>	<pre>[PTS_OPT,KOPT,T_EST] = wvarchg(Y,K,D) [PTS_OPT,KOPT,T_EST] = wvarchg(Y,K) [PTS_OPT,KOPT,T_EST] = wvarchg(Y)</pre>
<b>Description</b>	<p>[PTS_OPT,KOPT,T_EST] = wvarchg(Y,K,D) computes the estimated change points of the variance of signal Y for j change points, with <math>j = 0, 1, 2, \dots, K</math>.</p> <p>Integer D is the minimum delay between two change points.</p> <p>Integer KOPT is the proposed number of change points (<math>0 \leq KOPT \leq K</math>). The vector PTS_OPT contains the corresponding change points.</p> <p>For <math>1 \leq k \leq K</math>, T_EST(k+1,1:k) contains the k instants of the variance change points and then, if <math>KOPT &gt; 0</math>, PTS_OPT = T_EST(KOPT+1,1:KOPT) else PTS_OPT = [].</p> <p>K and D must be integers such that <math>1 &lt; K \ll \text{length}(Y)</math> and <math>1 \leq D \ll \text{length}(Y)</math>.</p> <p>The signal Y should be zero mean.</p> <p>wvarchg(Y,K) is equivalent to wvarchg(Y,K,10).</p> <p>wvarchg(Y) is equivalent to wvarchg(Y,6,10).</p>
<b>Examples</b>	<pre>% Generate signal from a fixed design regression model % with two change points in the noise variance located % at positions 200 and 600. % Generate block test function. 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 in signal x. % In addition, this example illustrates how the GUI % tools propose change point locations for interval</pre>

```
% dependent de-noising thresholds.
% 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);

% 2. Replace 2% of the biggest values by the mean
% in order to remove almost all the signal.
x = sort(abs(det));
v2p100 = x(fix(length(x)*0.98));
ind = find(abs(det)>v2p100);
det(ind) = mean(det);

% 3. Use wvarchg for estimating the change points with
% the following parameters.
% - the minimum delay between two change points d = 10.
% - the maximum number of change points is 5.
[pts_Opt,kopt,t_est] = wvarchg(det,5)

pts_Opt =
    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

% Estimated change points are close to the true change
% points [200,600].
```

**References**

Lavielle, M. (1999), "Detection of multiple changes in a sequence of dependent variables," *Stoch. Proc. and their Applications*, 83, 2, pp. 79–102.



# GUI Reference

---

This appendix explains some of the features of the Wavelet Toolbox™ graphical user interface (GUI).

- General Features (p. A-2)
- Continuous Wavelet Tool Features (p. A-18)
- Wavelet 1-D Tool Features (p. A-19)
- Wavelet 2-D Tool Features (p. A-21)
- Wavelet Packet Tool Features (1-D and 2-D) (p. A-22)
- Wavelet Display Tool (p. A-27)
- Wavelet Packet Display Tool (p. A-28)

## 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.

---

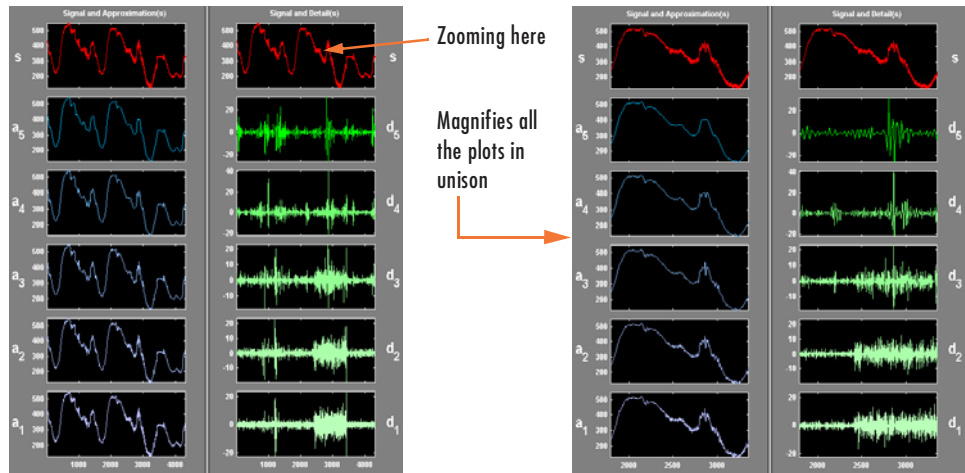
<b>Signal</b>	<b>Shown In</b>
Original	Red
Reconstructed or synthesized	Yellow
Approximations	Variegated shades of blue (high level = darker)
Details	Variegated shades of green (high level = darker)

---

## Connection of Plots


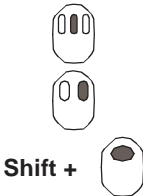
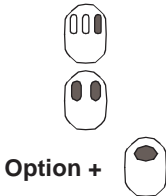
Plots containing related information and graphed on the same abscissa are connected in the sense that manipulations performed on one plot affect all others in the same way. For images, the connection holds in both abscissa and ordinate. You can manipulate all plots along an individual axis (X or Y) or 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

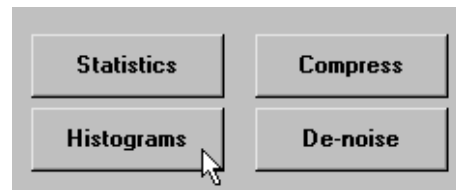
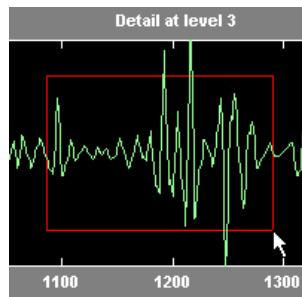
Wavelet Toolbox software uses three types of mouse control.

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

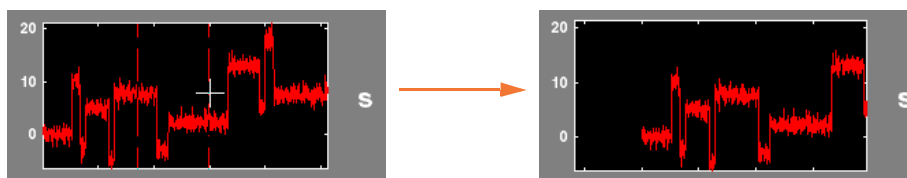
Most of your work with Wavelet Toolbox graphical tools involves making selections and activating controls. You do this using the left (or only) mouse button.





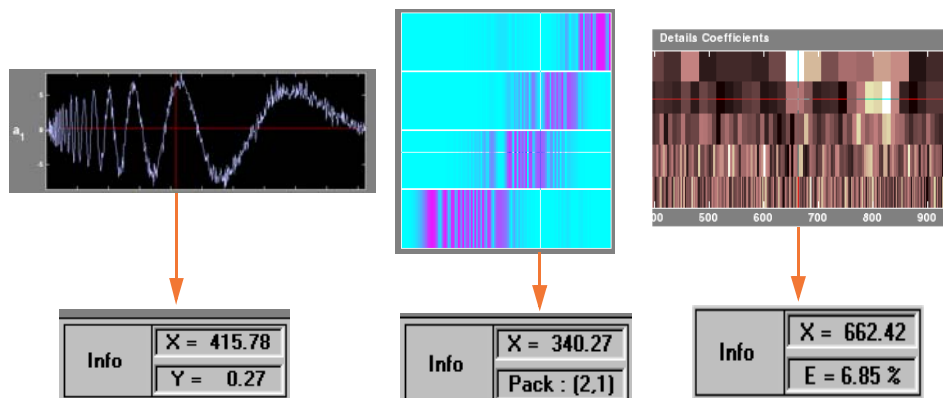
## Translating Plots

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.



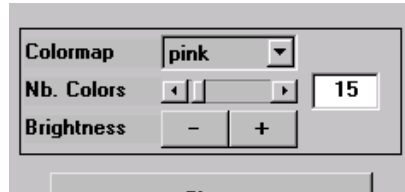
## Displaying Position-Dependent Information

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.



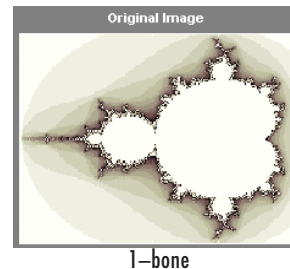
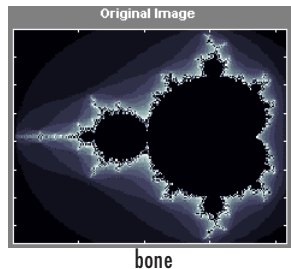
## Controlling the Colormap

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.

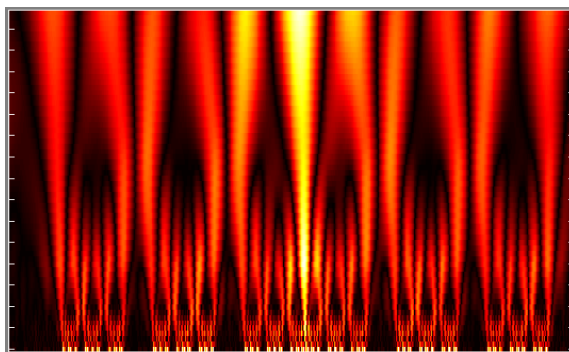
Consider these images of the Mandelbrot set generated in the **Wavelet Packet 2-D** tool, shown here using the **bone** and **1-bone** colormaps.



## Controlling the Number of Colors

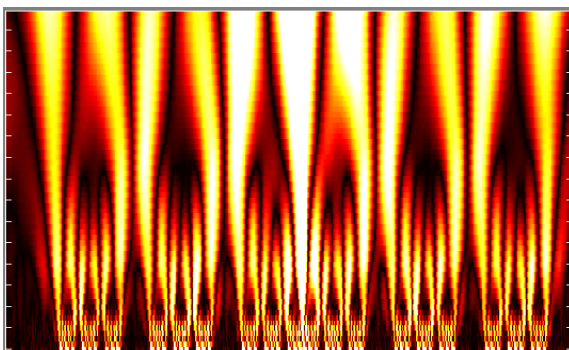
The **Nb. Colors** slider, located at the bottom right of the window, allows you to adjust how many colors the tool uses to plot images or coefficients (wavelet or wavelet packet). You can also use the edit control to adjust the number of colors. Adjusting the number of colors can highlight different features of the plot.

Consider the coefficients plot of the Koch curve generated in the **Continuous Wavelet** tool, shown here using 129 colors.



Colormap	hot	
Nb. Colors	<   >	129
Brightness	-	+

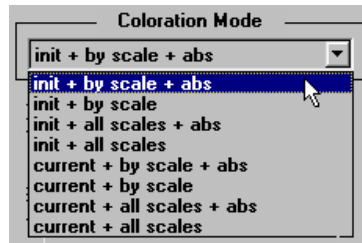
and here using 68 colors.



Colormap	hot	
Nb. Colors	<   >	68
Brightness	-	+

### 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*).

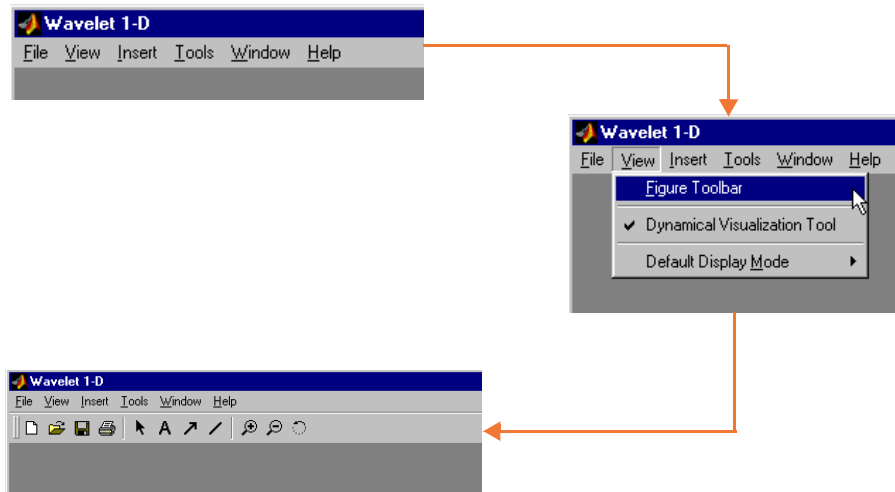
In the **Wavelet 1-D** tool, you access coefficients coloration with the **More Display Options** button, and then select the desired **Coloration Mode** option.

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

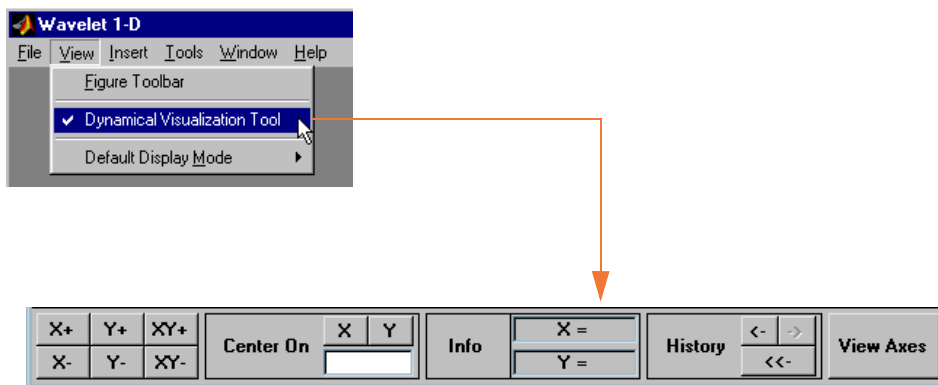
At the top of most windows you find the same kind of structure. The menu bar of each figure in Wavelet Toolbox software is very similar to the menu bar of the default MATLAB figures. You can use many of the tools that are offered in the menus and associated toolbar of the standard MATLAB figures.



One of the main differences is the **View** menu, which depends on the current tool used.

### View Dynamical Visualization Tool Option.

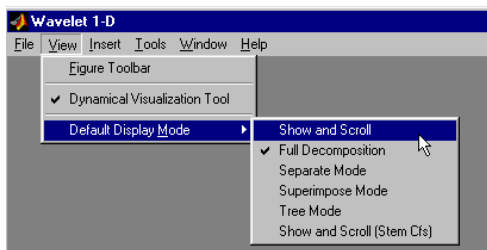
The **View**⇒**Dynamical Visualization Tool** option lets you enable or disable the **Dynamical Visualization Tool** located at the bottom of each window.



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.

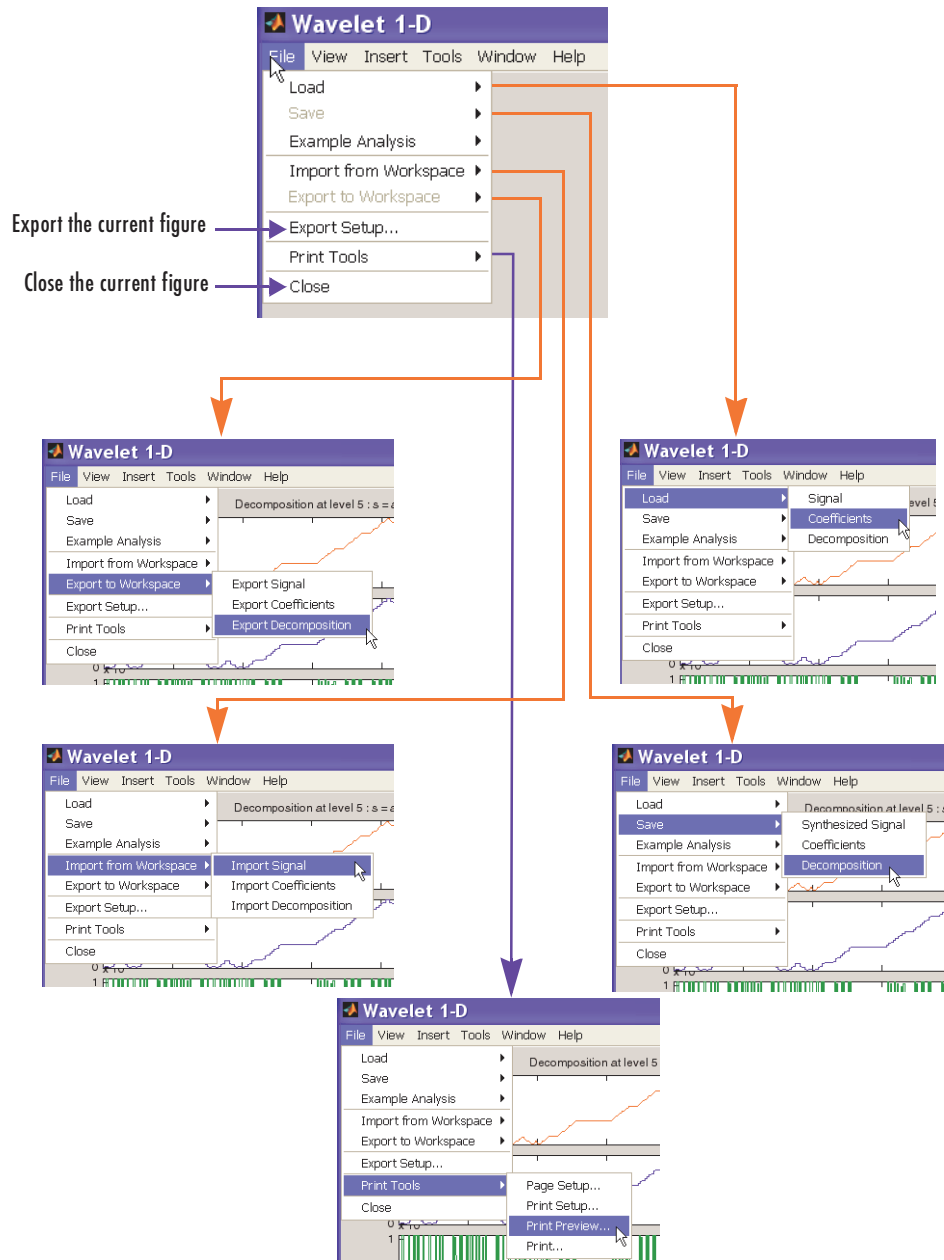
#### Default Display Mode Option.

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.



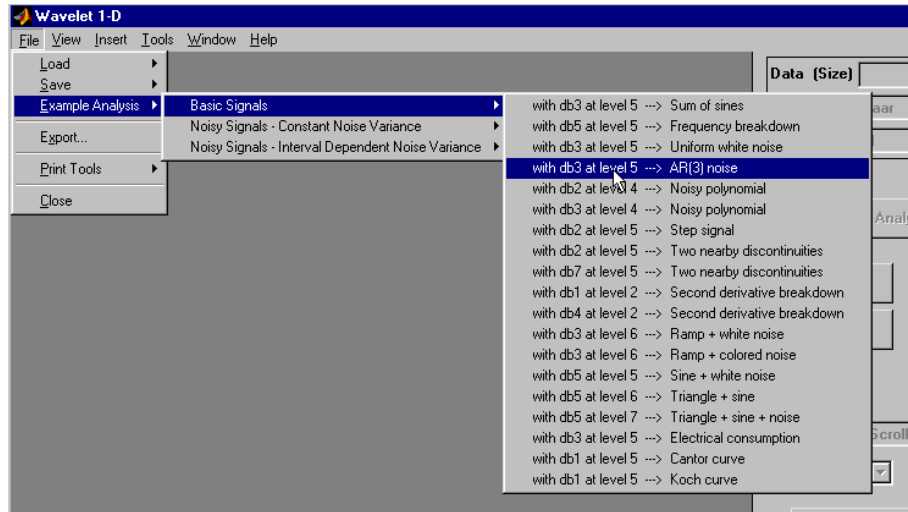
#### File Menu Options

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.

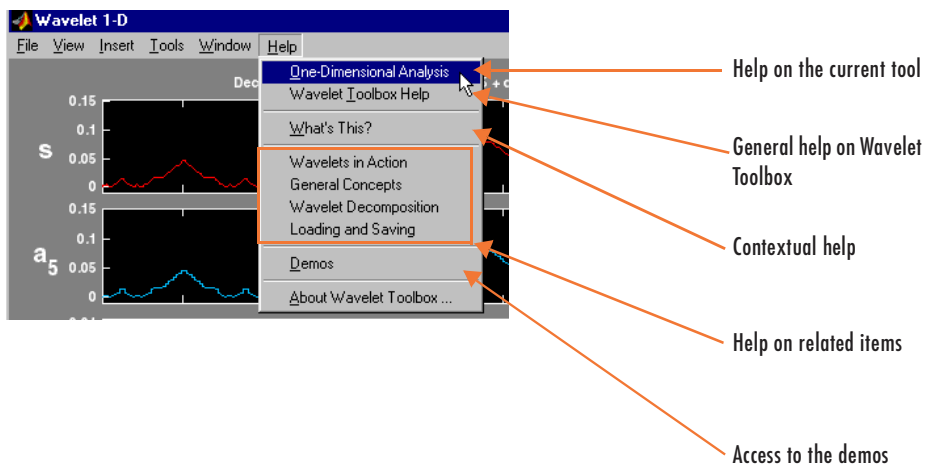
Here is an example of the **Wavelet 1-D** tool.



### Help Menu Options

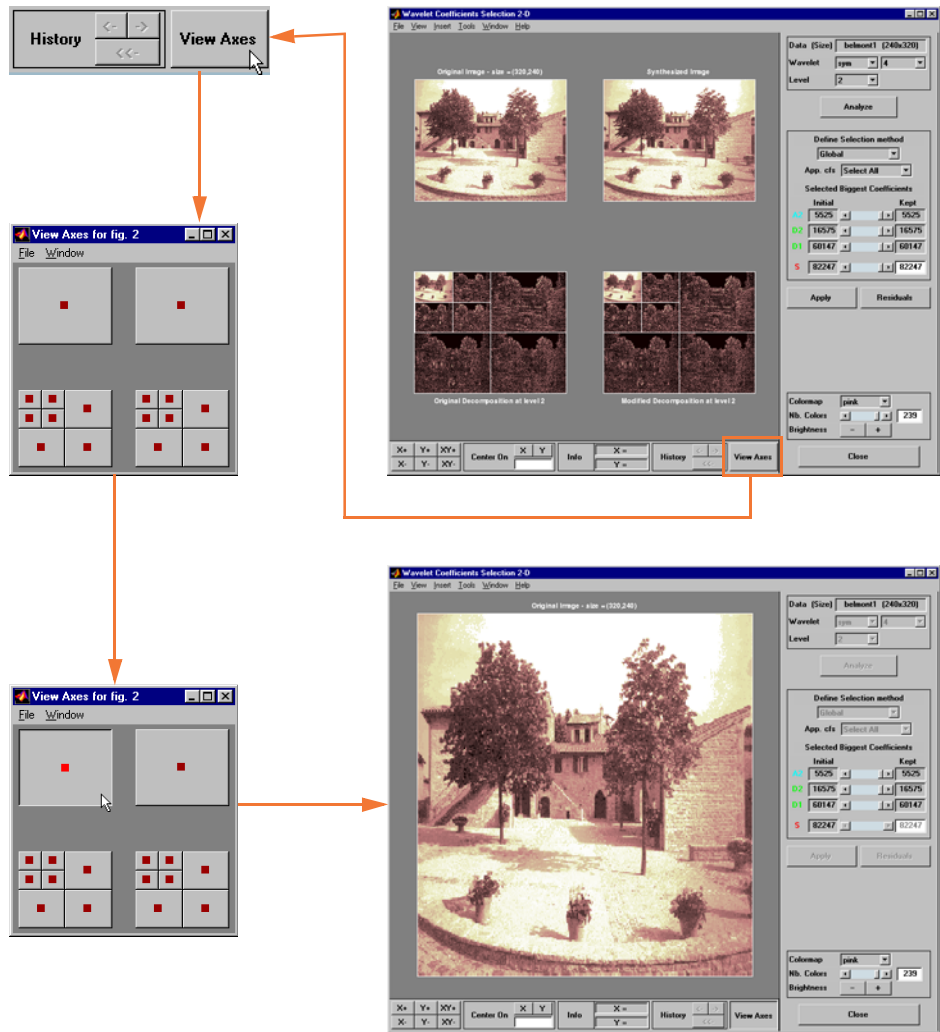
The help menu structure is very similar in all the figures, but many options are specific to the current tool in use.



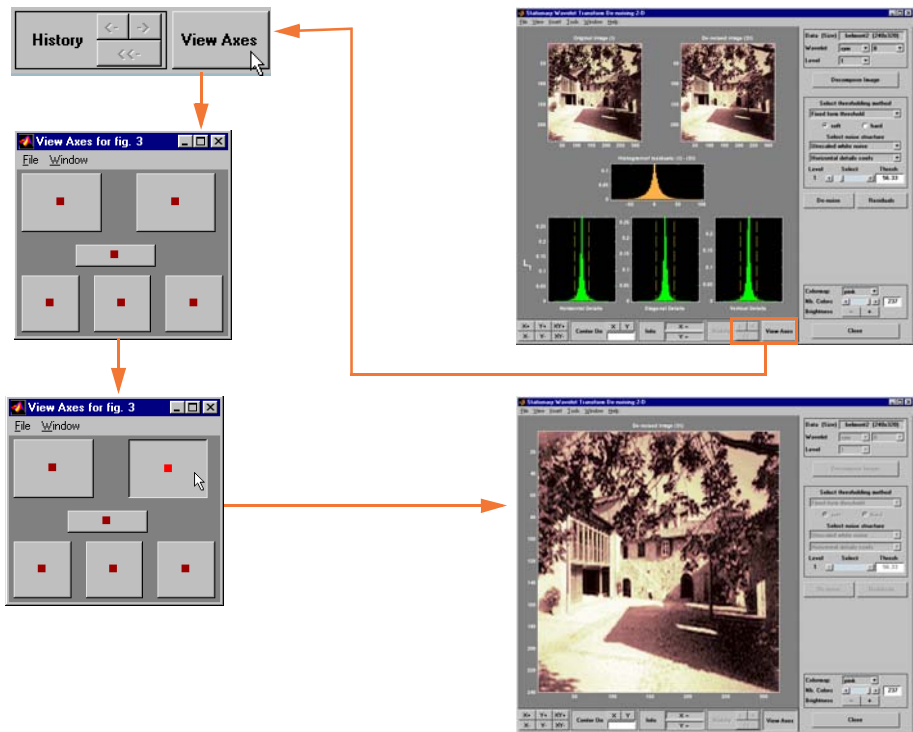


## Using the View Axes Button

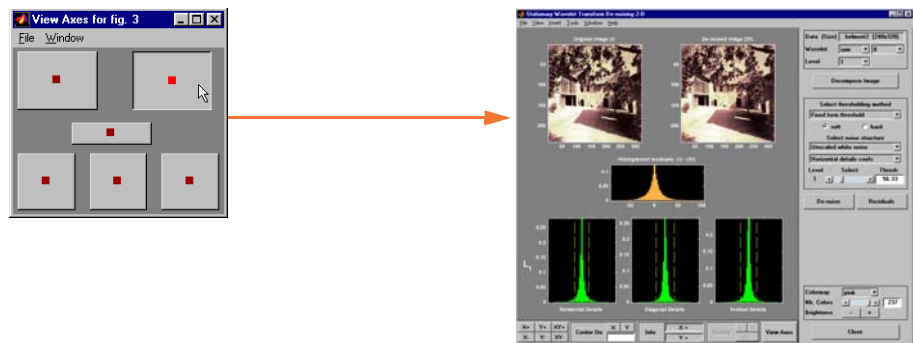
The **Dynamical Visualization Tool** is located at the bottom of most of the windows in the Wavelet Toolbox software. In this tool, the **View Axes** toggle button lets you magnify the axis that you choose.



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

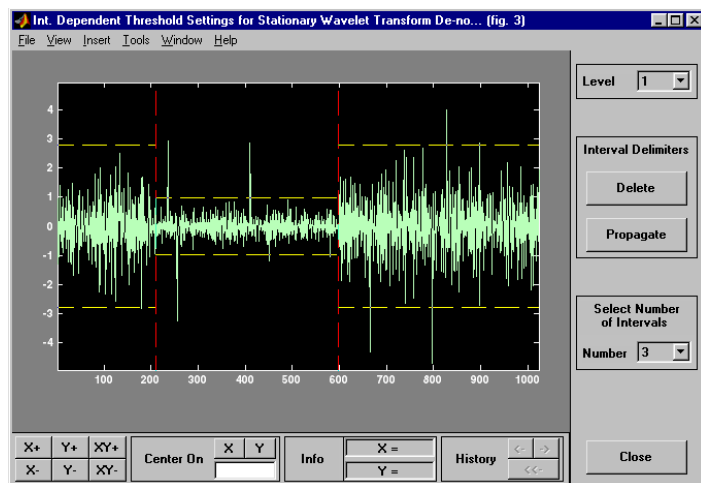
The following tools in the Wavelet Toolbox software let you define interval-dependent thresholds:

- 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

Lev	Int	Select	Thresh
5	1		4.114
4	1		4.114
3	1		4.114
2	1		4.114
1	1		4.114

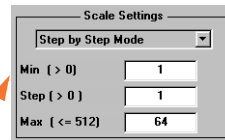
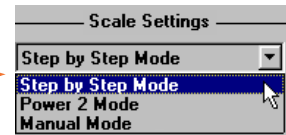
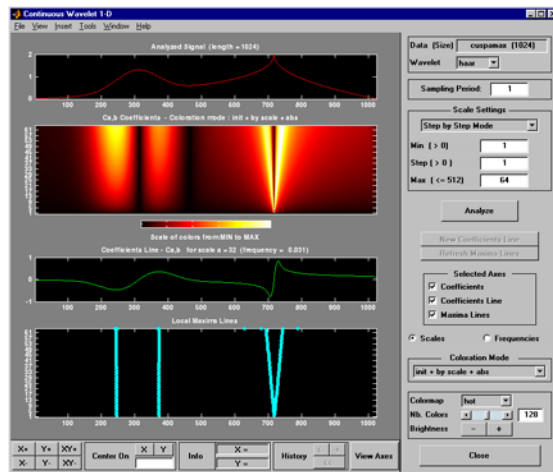
Clicking the **Int. dependent threshold settings** toggle button displays the **Int. dependent Threshold Settings for ...** window. After some computation is performed, the following figure appears.



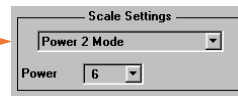
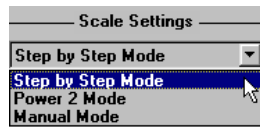
For more information on how to change interval limits and threshold values, see the section “One-Dimensional Variance Adaptive Thresholding of Wavelet Coefficients” on page 2-158.

## Continuous Wavelet Tool Features

Here is an example of an option that allows you to perform analysis using different scale modes.

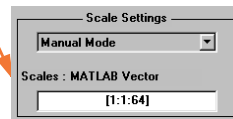


The three edit boxes allow you to specify the first scale value, the maximum scale value and the step size.



In power 2 mode, the scale values used are  $2^0, \dots, 2^k$  where  $k$  is the pop-up menu value.

The scale values are the same as those used for the discrete analysis.



This edit box allows you to specify the scales used for the continuous analysis, using MATLAB syntax for the input.

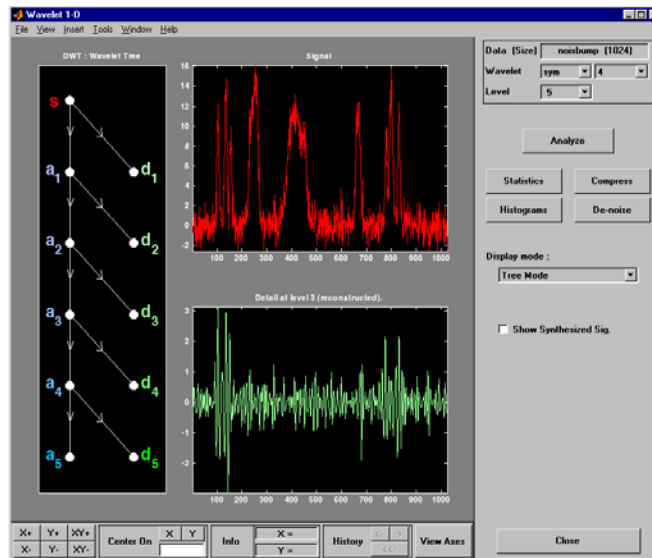
## Wavelet 1-D Tool Features

The **Wavelet 1-D** tool is described in the section “One-Dimensional Analysis Using the Graphical Interface” on page 2-40. 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.

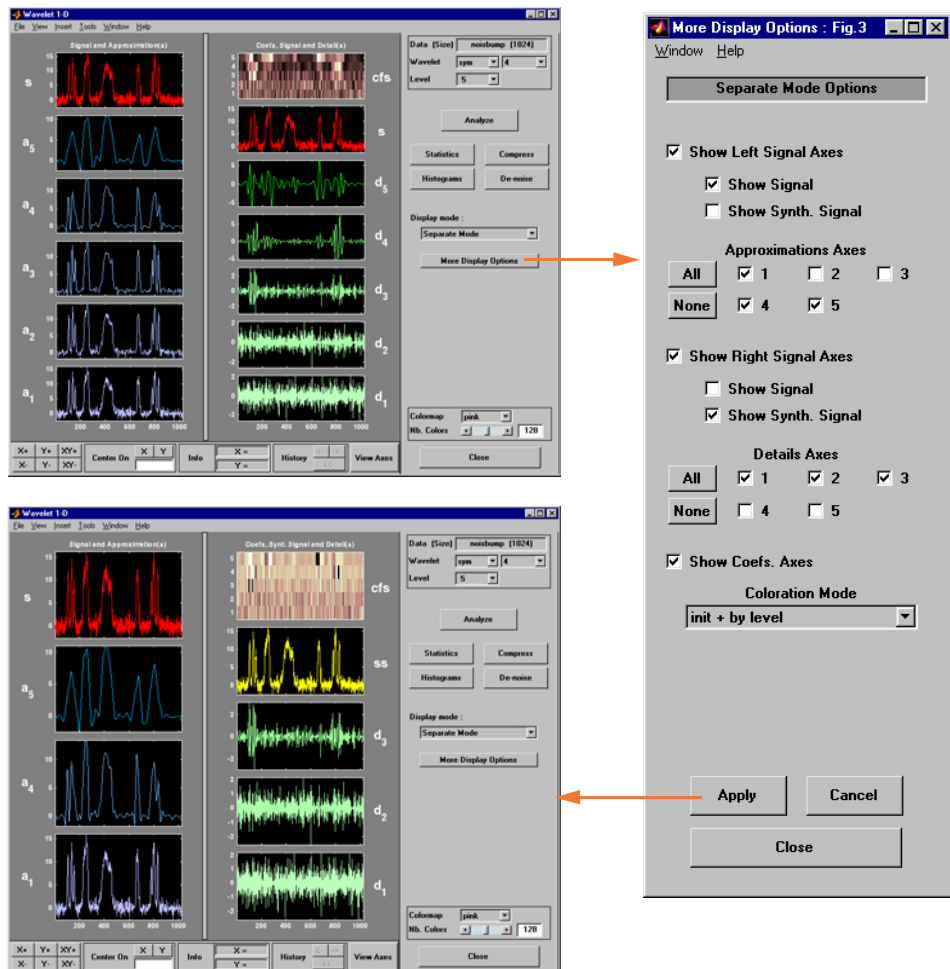
Here, on the left, the node  $d_3$  is selected and the corresponding detail is displayed under the original signal.



### More Display Options

This option allows you to customize what is displayed and is dependent on the current visualization mode.

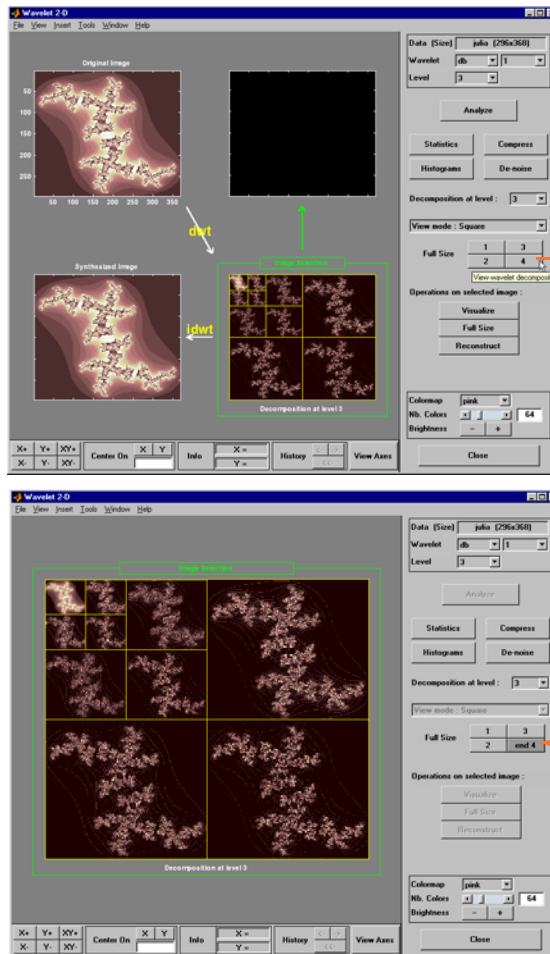
In this example for the **Separate Mode**, we have chosen not to display the coefficients of approximation for levels 2 and 3, nor the coefficients of detail for levels 4 and 5. The coefficients' coloration mode has been changed, and the synthesized signal is displayed in the right-hand column, rather than the original signal.





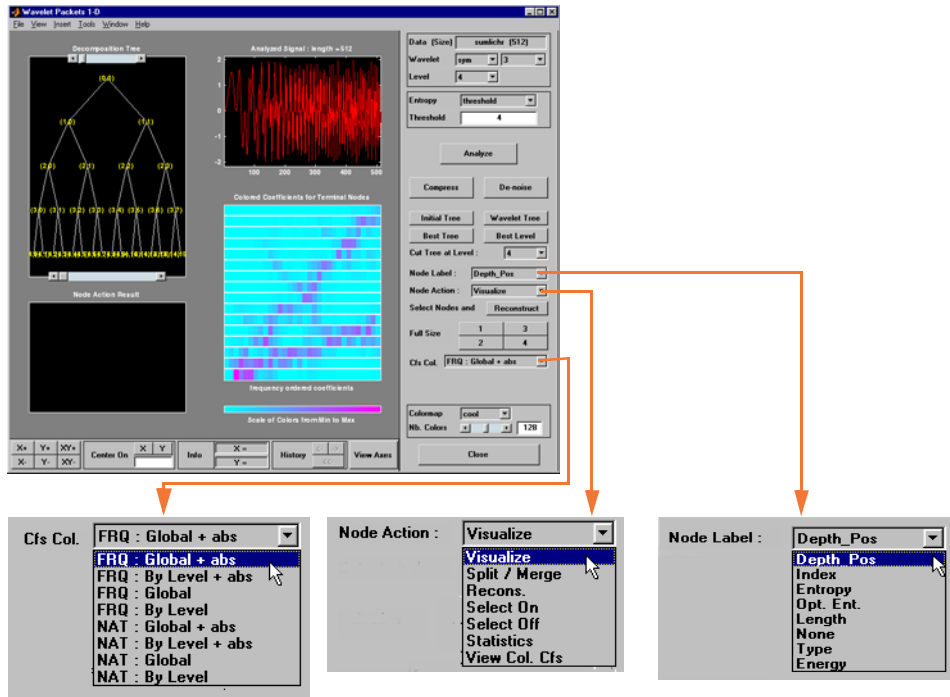
## Wavelet 2-D Tool Features

The **Wavelet 2-D** tool is described in the section “Two-Dimensional Analysis Using the Graphical Interface” on page 2-76. 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 “Using Wavelet Packets” on page 5-1. 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.



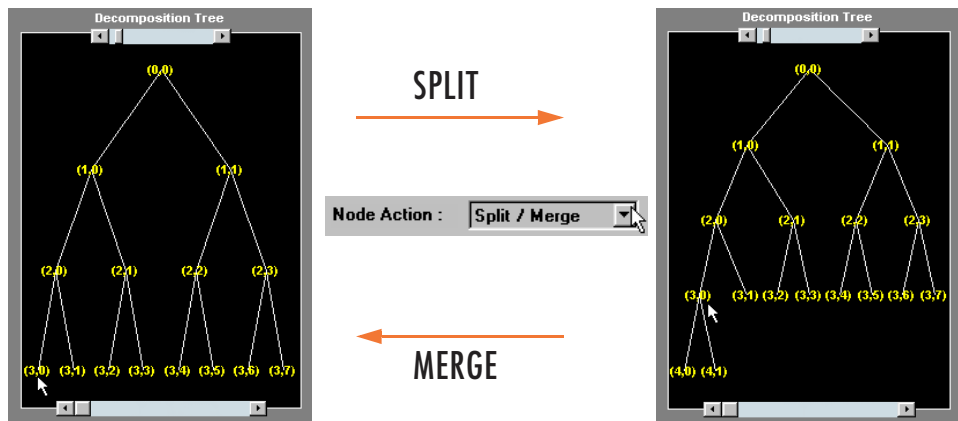
### Coefficients Coloration

**NAT** or **FRQ** is for Natural or Frequency order (see “Wavelet Packet Atoms” on page 6-143).

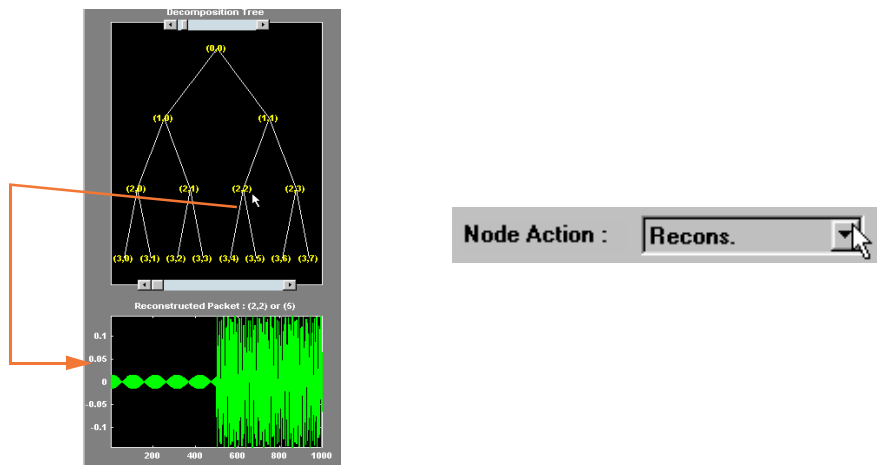
**By level** or **Global** is for a coloration made level by level or taking all detail levels.

**abs** is used to take the absolute values of coefficients.

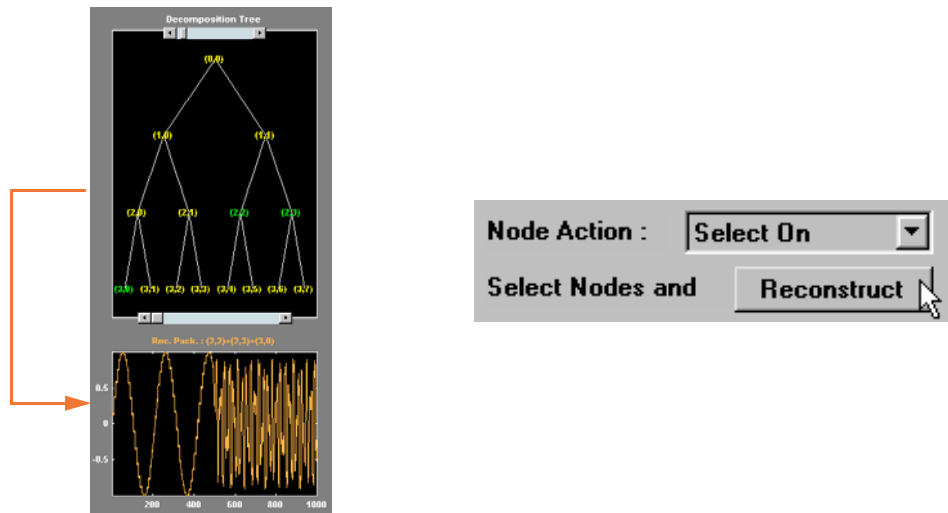




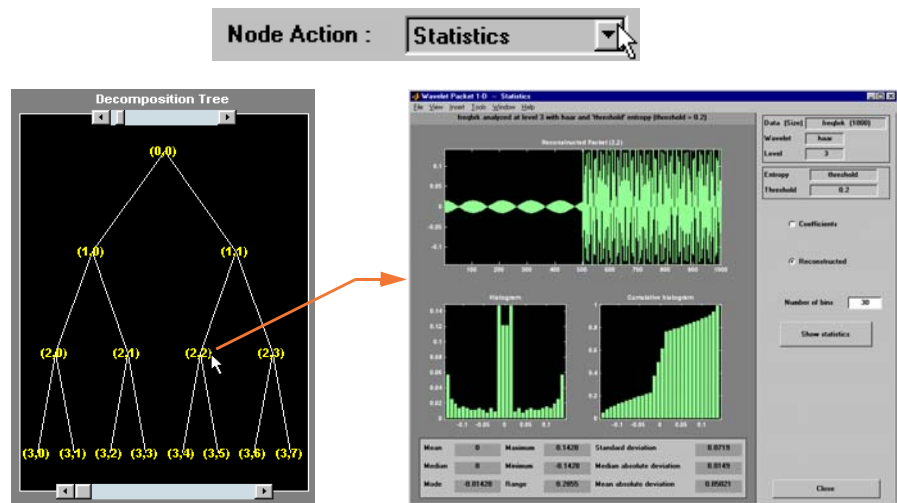
- **Recons.:** When you select a node in the wavelet packet tree, the corresponding reconstructed signal appears.



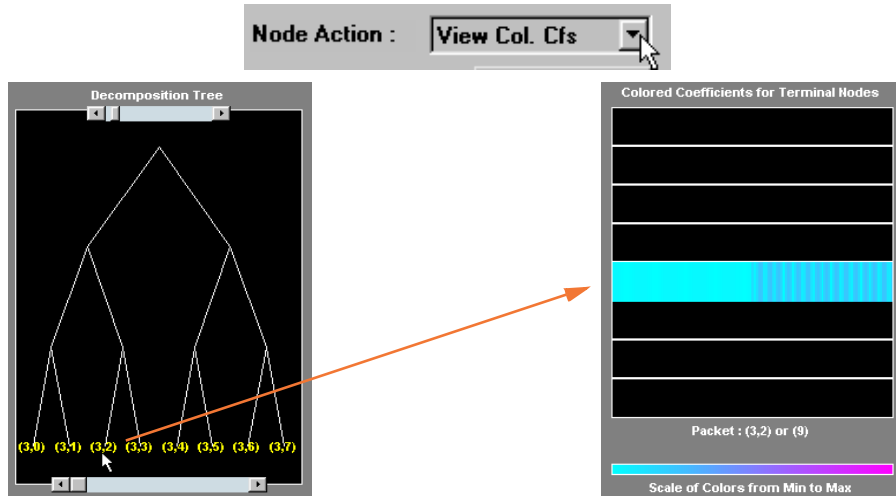
- **Select On/Off:** When **On**, you can select many nodes in the wavelet packet tree. Then you can reconstruct a synthesized signal from the selected nodes using the **Reconstruct** button on the main window. Use the **Off** selection to deselect all the previous selected nodes.



- **Statistics:** When you select a node in the wavelet packet tree, the **Statistics** tool appears using the signal corresponding to the selected node.



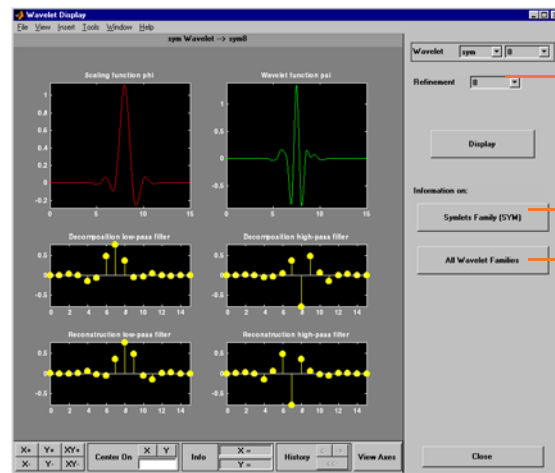
- **View Col. Cfs.:** When active, this option removes all the colored coefficients displayed, and lets you redraw only the corresponding coefficients by selecting a node in the wavelet packet tree.



# Wavelet Display Tool

The **Wavelet Display** tool is mentioned in the section “Introduction to the Wavelet Families” on page 1-39.

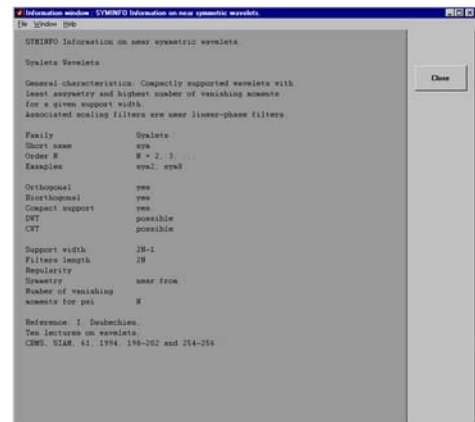
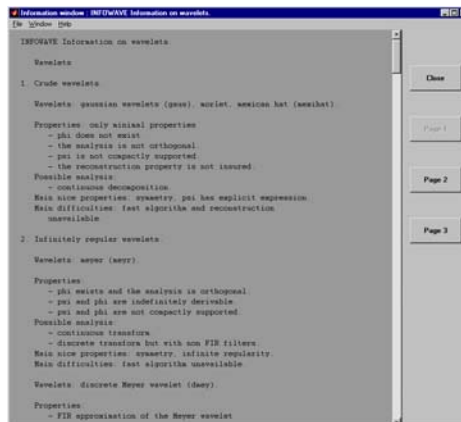
Here, we show the main window and the associated information windows with some additional comments.



This parameter decides the precision used for the wavelet computation. Here, functions are computed over  $2^8$  points.

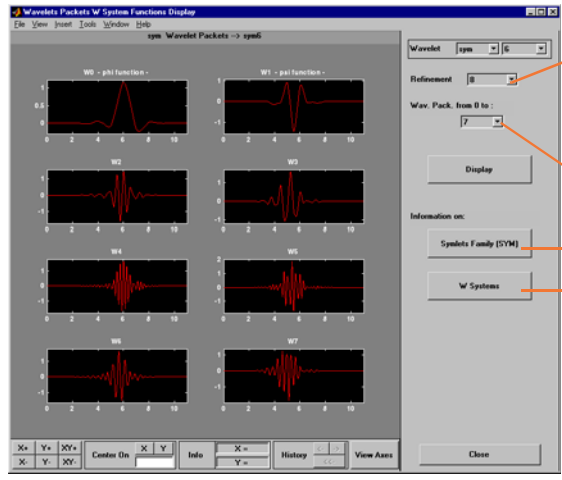
Information on all the wavelets

Information on the selected wavelet



# Wavelet Packet Display Tool

The **Wavelet Packet Display** tool is very similar to the **Wavelet Display** tool. Here, the main window and the associated information windows are displayed with some additional comments.

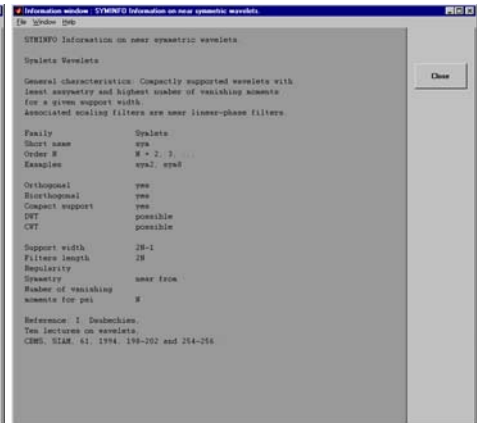
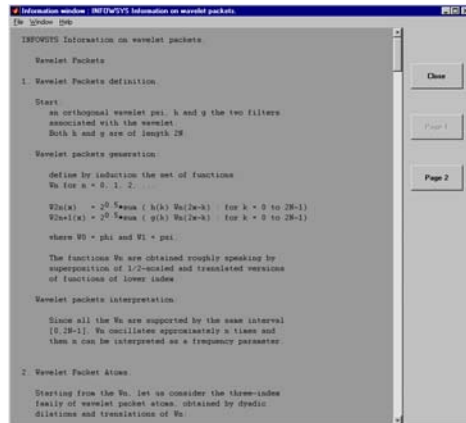


This parameter decides the precision used for the wavelet packet computation. Here, functions are computed over  $2^8$  points.

Number (-1) of wavelet packet functions to compute and display.

Information on wavelet packets

Information on the selected wavelet





# Object-Oriented Programming

---

This appendix explains concepts of object-oriented programming as they apply to Wavelet Toolbox™ software.

- Introduction to Object-Oriented Features (p. B-2)
- Short Description of Objects in the Wavelet Toolbox™ Software (p. B-3)
- Simple Use of Objects Through Four Examples (p. B-5)
- Detailed Description of Objects in the Wavelet Toolbox™ Software (p. B-16)
- Advanced Use of Objects (p. B-23)

## 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 “Using Wavelet Packets” on page 5-1, 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:

***WTBO*** → ***NTREE*** → ***DTREE*** → ***WPTREE***

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 “Detailed Description of Objects in the Wavelet Toolbox™ Software” on page B-16.

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

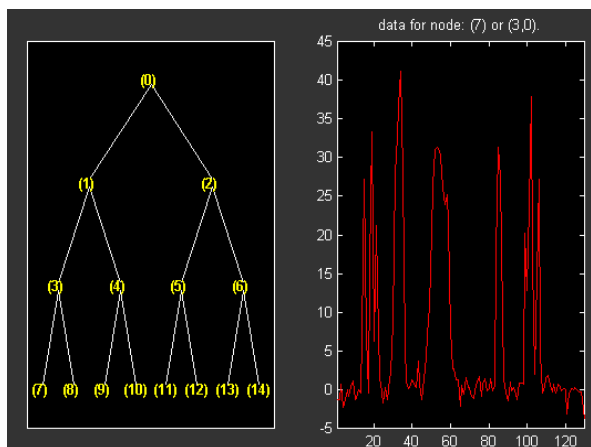
We can see some of these features in the following examples.

- “Example 1: `plot` and `wpviewcf`”
- “Example 2: `drawtree` and `readtree`” on page B-8
- “Example 3: A Funny One” on page B-11
- “Example 4: Thresholding Wavelet Packets” on page B-12

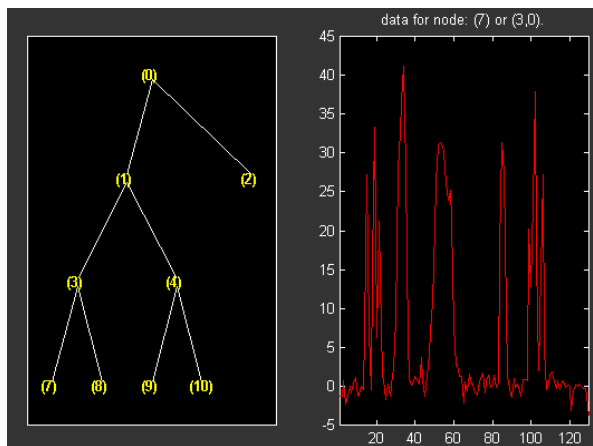
### Example 1: `plot` and `wpviewcf`

```
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.

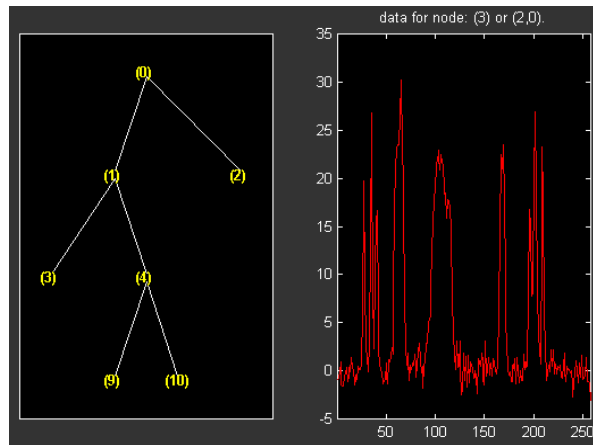
```

% 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).

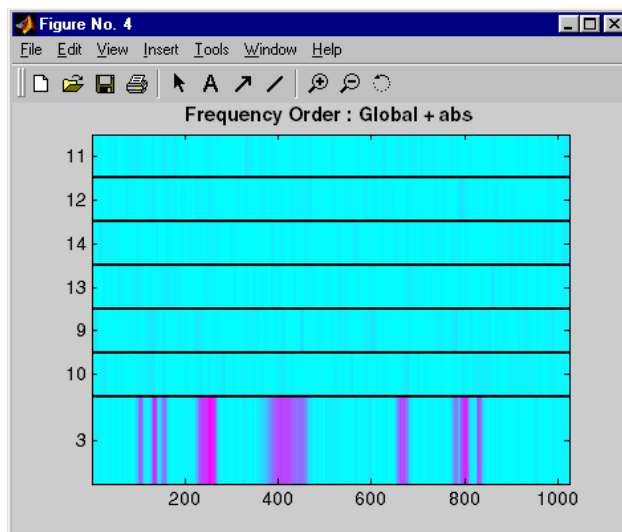
```

```
wpviewcf(newt,1)
```

```

% You get the following plot, which contains the terminal nodes
% colored coefficients.

```

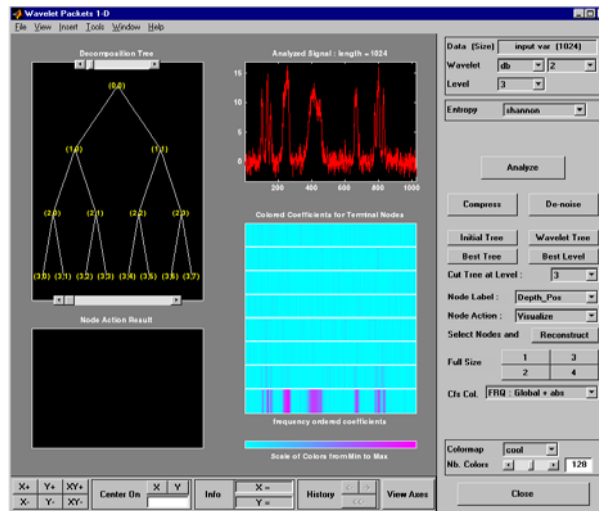


## Example 2: drawtree and readtree

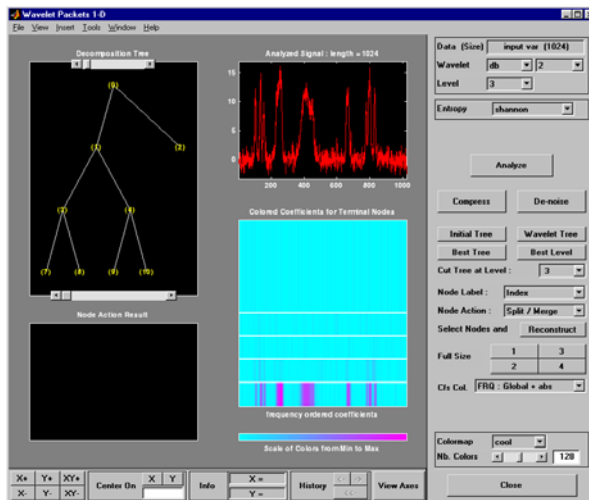
```
load noisbump
x = noisbump;
t = wpdec(x,3,'db2');
fig = drawtree(t);
```

```
% 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.
```



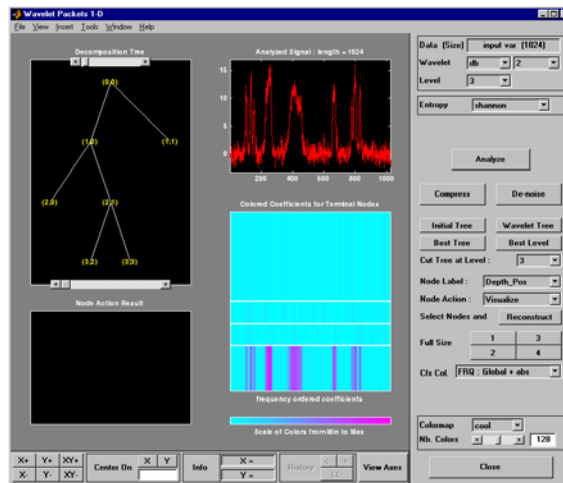


- % From the GUI, you can modify the tree.
- % For example, change Node label from Depth\_Position to Index,
- % change Node Action from Visualize to Split\_Merge and
- % merge the node 2.
- % You get the following figure.



```
% 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.

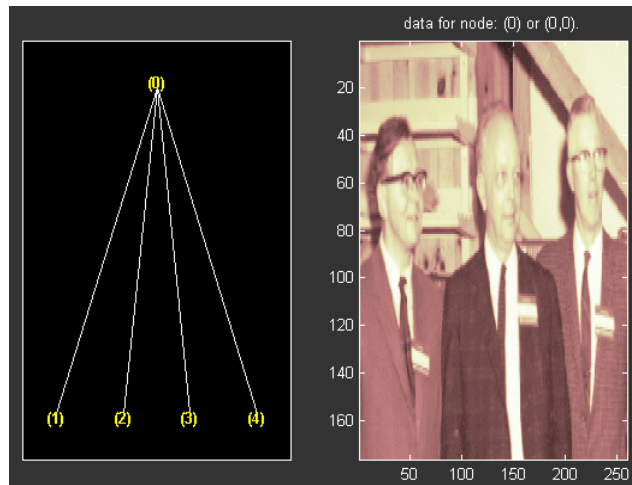
The methods associated with *WPTREE* objects let you do more complicated actions.

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.
```

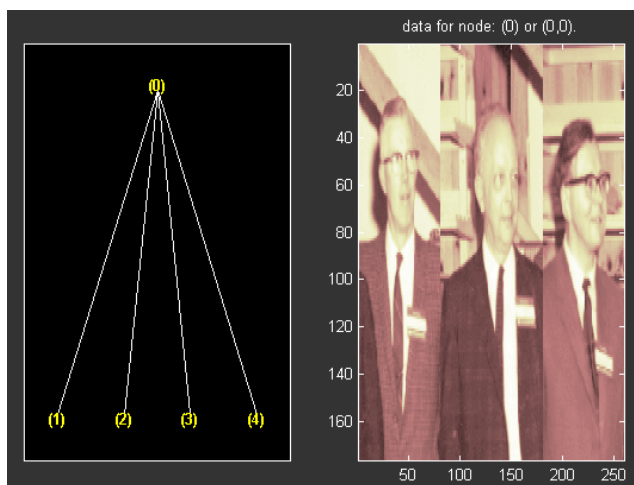


```
% Now modify the coefficients of the four terminal nodes.
```

```
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)
```

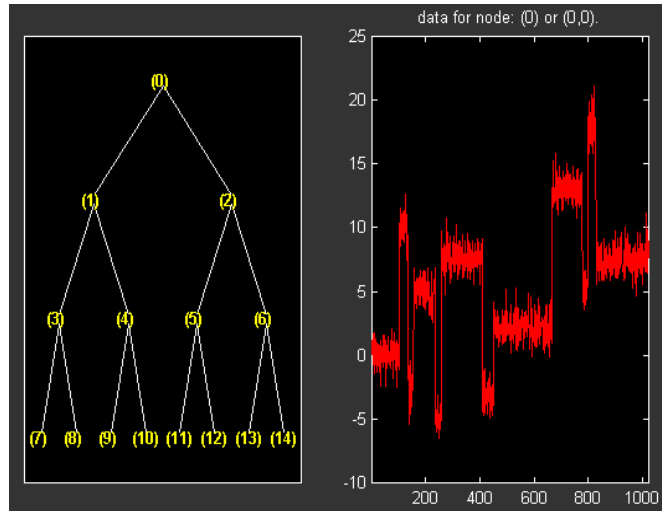
```
% Change Node Label from Depth_position to Index and
% click on the node (0). You get the following figure.
```



You can use this method for a more useful purpose. Let's see a de-noising example.

### Example 4: Thresholding Wavelet Packets

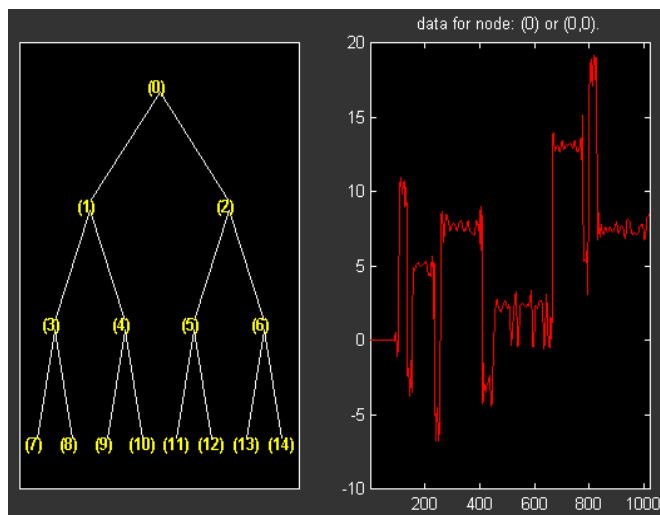
```
load noisbloc
x = noisbloc;
t = wpdec(x,3,'sym4');
plot(t);
% Change Node Label from Depth_position to Index and
% click the node (0). You get the following plot.
```



```
% 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.
```

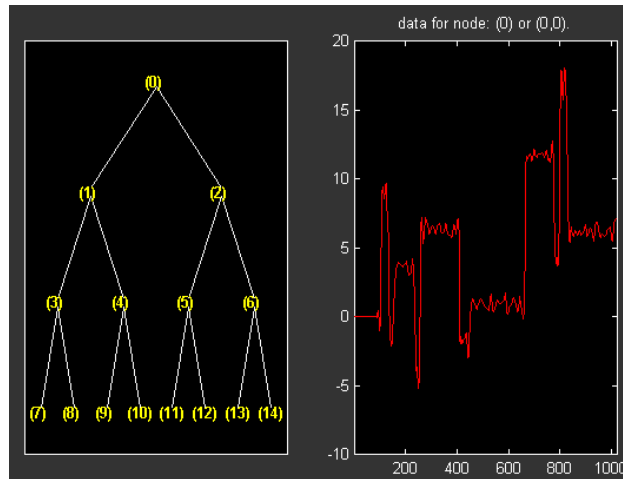


```

% Node by node thresholding.
t2 = t;
sorh = 's';
thr(1) = wthrmngr('wp1ddenoGBL','penalhi',t);
thr(2) = wthrmngr('wp1ddenoGBL','sqrtwologswn',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”
- “NTREE Object” on page B-17
- “DTREE Object” on page B-18
- “WPTREE Object” on page B-20

### **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 <b>WTBO</b> .
get	Get <b>WTBO</b> object field contents.
set	Set <b>WTBO</b> 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(Obj, '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 <i>NTREE</i> .
findactn	Find active nodes.
get	Get <i>NTREE</i> object field contents.
nodejoin	Recompose node(s).
nodesplt	Split (decompose) node(s).
plot	Plot <i>NTREE</i> object.
set	Set <i>NTREE</i> object field contents.
tlabls	Labels for the nodes of a tree.
wtreemgr	Manager for <i>NTREE</i> 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**

<code>ntree</code>	Parent object
<code>allNI</code>	All Nodes Information
<code>terNI</code>	Terminal Nodes Information

### **Fields Description**

`allNI` is a `NBnodes-by-3` array such that

$$\text{allNI}(N, :) = [\text{ind}, \text{size}(1,1), \text{size}(1,2)]$$

- `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

<code>dtree</code>	Constructor for the class <i>DTREE</i> .
<code>expand</code>	Expand data tree.
<code>fmdtree</code>	Field manager for <i>DTREE</i> object.
<code>nodejoin</code>	Recompose node.
<code>nodesplt</code>	Split (decompose) node.
<code>rnodcoef</code>	Reconstruct node coefficients.
<code>defaninf</code>	Define node information (all nodes).
<code>get</code>	Get <i>DTREE</i> object field contents.
<code>plot</code>	Plot <i>DTREE</i> object.
<code>read</code>	Read values in <i>DTREE</i> object fields.
<code>set</code>	Set <i>DTREE</i> object field contents.
<code>write</code>	Write values in <i>DTREE</i> object fields.
<code>merge</code>	Merge (recompose) the data of a node.
<code>recons</code>	Reconstruct node coefficients.
<code>split</code>	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**

<code>dtree</code>	Parent object
<code>wavInfo</code>	Structure (wavelet information)
<code>entInfo</code>	Structure (entropy information)

### **Fields Description**

`wavInfo`

<code>wavName</code>	Wavelet Name
<code>Lo_D</code>	Low Decomposition filter
<code>Hi_D</code>	High Decomposition filter
<code>Lo_R</code>	Low Reconstruction filter
<code>Hi_R</code>	High Reconstruction filter

`entInfo`

<code>entName</code>	Entropy Name
<code>entPar</code>	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 <i>WPTREE</i>

### Methods That Overload Those of *DTREE* Class.

Method	Description
defaninf	Define node information (all nodes).
get	Get <i>WPTREE</i> object field contents.
merge	Merge (recompose) the data of a node.
read	Read values in <i>WPTREE</i> object fields.
recons	Reconstruct wavelet packet coefficients.
set	Set <i>WPTREE</i> 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 <i>WPTREE</i> object fields.

**Proper Methods of *WPTREE* Class.**

<b>Method</b>	<b>Description</b>
bestlevt	Best level of a wavelet packet tree.
besttree	Best wavelet packet tree.
entrupd	Entropy update (wavelet packet tree).
wp2wtree	Extract wavelet tree from wavelet packet tree.
wpccoef	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*)”
- “Example 2: Building a Right Wavelet Tree Object (*RWVTREE*)” on page B-24
- “Example 3: Building a Wavelet Tree Object (*WVTREE*)” on page B-26
- “Example 4: Building a Wavelet Tree Object (*EDWTTREE*)” on page B-27

### 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` M-file and at a two-dimensional example in the `ex2_wt` M-file located in the `toolbox/wavelet/wavedemo` directory. 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

<code>dtree</code>	Parent object
<code>dwtMode</code>	DWT extension mode
<code>wavInfo</code>	Structure (wavelet information)

### wavInfo Structure information

<code>wavName</code>	Wavelet Name
<code>Lo_D</code>	Low Decomposition filter
<code>Hi_D</code>	High Decomposition filter
<code>Lo_R</code>	Low Reconstruction filter
<code>Hi_R</code>	High Reconstruction filter

### Methods

<code>wtree</code>	Constructor for the class <i>WTREE</i> .
<code>merge</code>	Merge (recompose) the data of a node.
<code>split</code>	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` M-file and at a two-dimensional example in the `ex2_rwvt` M-file located in the `toolbox/wavelet/wavedemo` directory. 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

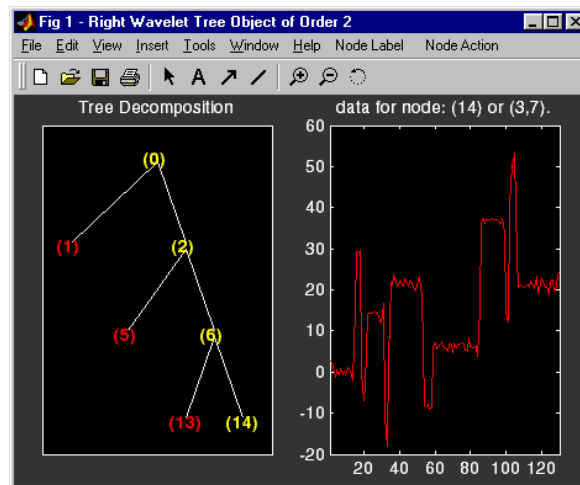
### Methods

rwtree	Constructor for the class <i>RWVTREE</i> .
merge	Merge (recompose) the data of a node.
plot	Plot <i>RWVTREE</i> 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` M-file and at a two-dimensional example in the `ex2_wvt` M-file located in the `toolbox/wavelet/wavedemo` directory. 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**

<code>dummy</code>	Not used
<code>wtree</code>	Parent object

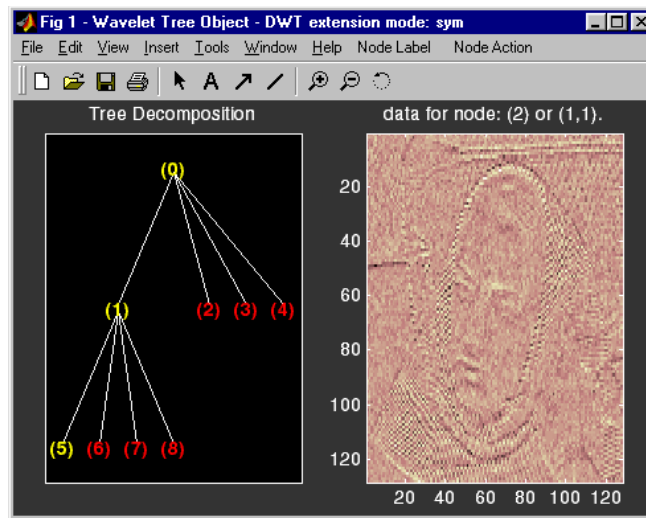
#### **Methods**

<code>wtree</code>	Constructor for the class <i>WVTREE</i> .
<code>get</code>	Get <i>WVTREE</i> object field contents.
<code>plot</code>	Plot <i>WVTREE</i> object.
<code>recons</code>	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  $\varepsilon$ -DWT tree class starting from the class *DTREE* and overloading the methods `merge`, `plot`, `recons`, and `split`.

For more information on the  $\varepsilon$ -DWT, see the section “e-Decimated DWT” on page 6-44.

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` M-file located in the `toolbox/wavelet/wavedemo` directory. 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

<code>dtree</code>	Parent object
<code>dwtMode</code>	DWT extension mode
<code>wavInfo</code>	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.

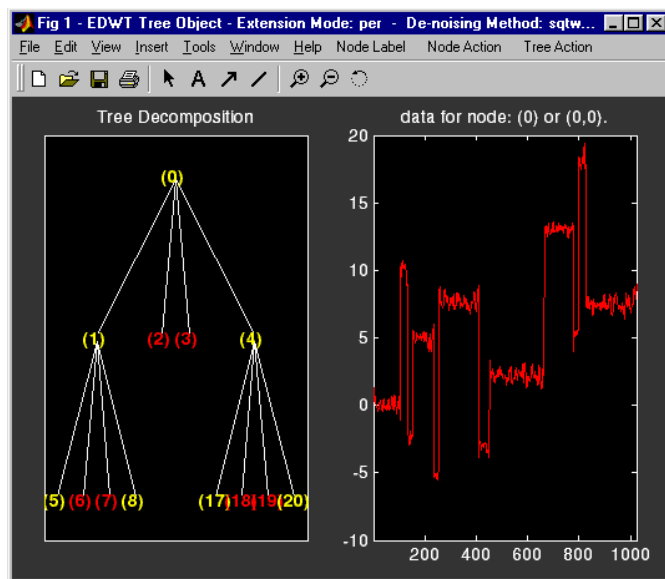
`split` Split (decompose) the data of a terminal node.

## 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.





## A

- adding a new wavelet 7-2
- algorithms
  - Coifman-Wickerhauser 1-37
  - decomposition 6-23
  - discrete wavelet transform (DWT) 6-19
  - fast wavelet transform (FWT) 6-19
  - filters 6-19
  - for biorthogonal 6-28
  - lifting wavelet transform (LWT) 6-54
  - Mallat 6-19
  - polyphase 6-56
  - rationale 6-28
  - reconstruction 6-30
  - stationary wavelet transform (SWT) 6-44
- analysis
  - biorthogonal 6-76
  - case study 4-36
  - continuous
    - coefficients 6-12
    - features 2-4
    - procedure 1-17
  - continuous complex 2-20
  - continuous or discrete 6-61
  - continuous real 2-4
  - discrete
    - coefficients 6-12
    - procedure 1-24
  - illustrated examples 4-2
  - local 1-12
  - local and global 6-14
  - multiscale 4-36
  - one-dimensional discrete wavelet 2-29
  - one-dimensional wavelet packet 5-7
  - orthogonal
    - algorithm 6-28
    - and wavelet families 6-71

- basis 6-61
  - dbN wavelets 6-74
  - filters 6-19
  - redundant 6-13
  - time-scale
    - compared to other views 1-11
    - using redundant representation instead 6-13
  - translation invariant 6-44
  - two-dimensional discrete wavelet 2-66
  - two-dimensional wavelet packet 5-23
  - wavelet 1-11
  - wavelet packet 5-2
  - See also* transforms
- approximations
  - coefficients
    - discrete wavelet transform 6-23
    - extracting one-dimensional 2-33
    - extracting two-dimensional 2-72
  - definition 6-17
  - description 1-24
  - notation 6-3
  - reconstruction
    - example code 2-34
    - filters 1-30
  - wavelet decomposition 6-5
- axes
  - view A-13

## B

- bases
  - See* analysis, wavelet packets
- besttree function 6-154
- binning
  - density estimation 6-115

- processed data 2-146
- regression estimation 6-120
- biorthogonal
  - quadruplets 6-52
- biorthogonal wavelets
  - definition 6-76
  - presentation 1-43
  - See also* analysis
- border distortion
  - boundary value replication 6-35
  - periodic extension 6-35
  - periodic padding 6-36
  - periodized wavelet transform 6-44
  - smooth padding 6-36
  - symmetric extension 6-35
  - symmetrization 6-35
  - zero-padding 6-35
- breakdowns
  - frequency 3-3
  - peak 4-34
  - proximal slopes 4-20
  - rupture 4-18
  - second derivative 4-22
  - signal's derivative 3-6
  - variance 6-107
- C**
- calculating a default global threshold 2-112
- centfrq function 6-67
- chirp signal
  - example analysis 6-140
- coefficients
  - approximation
    - extracting one-dimensional 2-33
    - fast wavelet transform 6-23
  - coloration A-22
- complex continuous wavelet 2-22
- continuous wavelet 2-6
- detail
  - fast wavelet transform 6-23
  - one-dimensional 2-33
  - two-dimensional 2-72
- discrete wavelet
  - Wavelet 1-D tool 2-64
  - Wavelet 2-D tool 2-93
- line 2-11
- load
  - See* importing to the GUI
- save
  - See* exporting from the GUI
- coiflets
  - definition 6-75
  - presentation 1-45
- Coloration Mode
  - color coding A-2
  - controlling A-7
  - controlling the colormap A-6
- Coloration Mode menu 2-17
- colored AR(3) noise example 4-14
- complex frequency B-spline wavelets 6-85
- complex Gaussian wavelets 6-84
- complex Morlet wavelets 6-84
- complex Shannon wavelets 6-86
- compressing images
  - fingerprint example 3-26
  - true compression 6-130
  - using graphical interface 2-84
- compression
  - ddencmp function 5-4
  - difference with de-noising 6-110
  - energy ratio 6-113
  - methods 6-125
  - norm recovery 6-113



- number of zeros 6-114
  - predefined strategies 6-122
  - procedure
    - wavelet packets 5-5
    - wavelets 6-110
  - retained energy 6-113
  - thresholding strategies 6-125
  - true 6-130
  - using wavelet packets 5-28
  - continuous wavelet transform (CWT)
    - creating 1-17
    - definition 1-15
    - difference from discrete wavelet transform 1-22
    - See also* analysis, transforms
  - CWT
    - See* continuous wavelet transform (CWT)
- D**
- Daubechies wavelets
    - definition 6-72
    - presentation 1-42
  - ddencmp command 2-112
  - decimation
    - See* downsampling
  - decomposition
    - best-level 6-152
    - choosing optimal 6-147
    - displaying results of multilevel 2-35
    - entropy-based criteria 6-147
    - hierarchical organization 6-10
    - load
      - See* importing to the GUI
    - multistep 1-35
    - optical comparison 6-6
    - performing multilevel
      - using command line 2-33
      - using graphical interface 2-46
  - save
    - See* exporting from the GUI
  - single-level 2-43
  - structure
    - one-dimensional 2-64
    - two-dimensional 2-93
  - wavelet decomposition tree 1-27
  - de-noising
    - basic model
      - one-dimensional 6-97
      - two-dimensional 6-106
    - default values 5-4
    - fixed form threshold 6-100
    - graphical interface tools option 2-49
    - methods 6-125
    - minimax performance 6-100
    - noise size estimate 6-102
    - nonwhite noise 6-102
    - predefined strategies 6-122
    - procedure
      - wavelet packets 5-5
      - wavelets 6-98
    - SURE estimate 6-100
    - thresholding 2-37
    - using SWT
      - 1-D using command line 2-112
      - 1-D using graphical interface 2-114
      - 2-D analysis example 3-24
      - 2-D using graphical interface 2-132
    - using thresholding 2-130
    - variance adaptive 6-107
    - white noise 6-97
  - de-noising images
    - 2-D wavelet analysis and 2-D stationary wavelet analysis 3-21

- stationary wavelet transform 2-135
- two-dimensional procedure 6-106
- de-noising signals
  - process 2-36
  - wavelet analysis 3-18
- density estimation
  - definition 6-114
  - one-dimensional wavelet 2-150
- details
  - coefficients
    - extracting one-dimensional 2-33
    - extracting two-dimensional 2-72
  - decomposition 6-136
  - definition 1-24
  - mathematical definition 6-17
  - notation 6-3
  - orientation 6-25
  - reconstruction
    - from command line 2-34
    - procedure 1-30
  - wavelet decomposition 6-5
- dilation equation
  - twin-scale relation 6-19
- discontinuities
  - detecting 3-3
  - See also* breakdowns
- discrete Meyer wavelet 6-83
- discrete wavelet transform (DWT)
  - definition 1-24
  - See also* analysis, transforms
- display mode 2-D
  - square 2-81
  - tree 2-82
- downsampling
  - one-dimensional 6-24
  - two-dimensional 6-25
- DWT

- See* discrete wavelet transform, transforms
- dyadic scale 1-24

## E

- edge effects
  - See* border distortion
- elementary lifting steps (ELS) 6-52
- ELS
  - See* lifting
- entropy
  - criterion to select the best decomposition 1-37
  - definitions 6-148
- estimation
  - default values 6-122
  - See also* function estimation
- examples
  - colored AR(3) noise 4-14
  - frequency breakdown 4-10
  - polynomial + white noise 4-16
  - ramp + colored noise 4-26
  - ramp + white noise 4-24
  - real electricity consumption signal 4-34
  - second-derivative discontinuity 4-22
  - sine + white noise 4-28
  - step signal 4-18
  - sum of sines 4-8
  - triangle + a sine 4-30
  - triangle + a sine + noise 4-32
  - two proximal discontinuities 4-20
  - uniform white noise 4-12
- exporting from the GUI
  - complex continuous wavelet 2-28
  - continuous wavelet 2-17
  - discrete stationary wavelet 1-D 2-120
  - discrete stationary wavelet 2-D 2-138
  - discrete wavelet 2-D 2-88

- image extension 2-198
- signal extension 2-194
- variance adaptive thresholding 2-166
- wavelet 1-D 2-58
- wavelet density estimation 1-D 2-156
- wavelet packets 5-32
- wavelet regression estimation 1-D 2-148

extension mode

- See* border distortion

## F

- fast multiplication of large matrices 3-28
- fast wavelet transform (FWT)

- See* transforms

filters

FIR

- biorthogonal case 6-77
- M-file used for construction 7-4

high-pass 6-23

low-pass 6-23

minimum phase 6-75

quadrature mirror

- construction example 6-21

- definition 1-30

reconstruction 1-30

scaling 6-19

- See also* twin-scale relations

fixed design

- See* regression

Fourier analysis

- basic function 6-14

- introduction 1-9

- short-time analysis (STFT) 1-10

- windowed 4-11

Fourier coefficients 1-15

Fourier transform 1-15

fractal

- properties of signals and images 3-11

- redundant methods 6-13

fractional Brownian motion 2-208

frequencies

- identifying pure 3-12

- parameter 6-144

- related to scale 6-66

- wavelets interpretation 1-20

frequency breakdown example 4-10

frequency B-spline wavelets 6-85

Full Decomposition Mode 2-47

function estimation 6-114

fusion of images

- See* image fusion

FWT

- See* transforms

## G

Gaussian wavelets 6-82

GUI

- complex continuous wavelet 2-23

- continuous wavelet 2-8

- density estimation 2-150

- fractional brownian motion 2-210

- full window resolution A-21

- image de-noising using SWT 2-132

- image extension / truncation 2-195

- image fusion 2-202

- local variance adaptive thresholding 2-158

- multiscale principal components analysis

- 2-251

- multisignal analysis 2-271

- multivariate wavelet de-noising 2-234

- new wavelet for CWT 2-218

- regression estimation 2-140

- signal de-noising using SWT 2-114
- signal extension / truncation 2-186
- true compression 2-317
- using menus A-9
- using the mouse A-4
- wavelet coefficients selection 1-D 2-168
- wavelet coefficients selection 2-D 2-178
- wavelet display A-27
- wavelet one-dimensional 2-40
- wavelet packet 5-7
- wavelet packet display A-28
- wavelet two-dimensional 2-76

## H

- Haar wavelet
  - definition 6-73
  - presentation 1-41
- Heisenberg uncertainty principle 6-14

## I

- IDWT
  - See* inverse discrete wavelet transform, transforms
- ILWT
  - See* inverse lifting wavelet transform
- image fusion 2-199
- images
  - indexed 2-92
- importing to the GUI
  - complex continuous wavelet 2-28
  - continuous wavelet 2-17
  - discrete stationary wavelet 1-D 2-120
  - discrete stationary wavelet 2-D 2-138
  - discrete wavelet 2-D 2-88
  - variance adaptive thresholding 2-166

- wavelet 1-D 2-58
- wavelet density estimation 1-D 2-156
- wavelet packets 5-32
- wavelet regression estimation 1-D 2-148
- indexed images
  - image matrix 2-97
  - matrix indices
    - shifting up 2-98
- inverse discrete wavelet transform (IDWT) 1-29
- inverse lifting wavelet transform (ILWT) 6-56
- inverse stationary wavelet transform (ISWT) 6-50
- ISWT
  - See* inverse stationary wavelet, transforms
- iswt command 2-112
- iswt2 command 2-130

## L

- Laurent polynomial 6-53
- lazy wavelet 6-54
- level
  - decomposition 1-27
  - See also* wavelet packet best level
- lifting 6-51
  - dual 6-53
  - elementary step (ELS) 6-52
  - primal 6-53
  - scheme (LS) 6-54
- lifting wavelet transform (LWT) 6-56
- load
  - See* importing in the GUI
- Load data for Density Estimate dialog box 2-152
- Load data for Stochastic Design Regression dialog box 2-146
- Load Signal dialog box
  - wavelet packets 5-8
  - wavelets 2-42

- local analysis
  - See* analysis
- local maxima lines 2-11
- long-term evolution
  - detecting 3-8
- LS
  - See* lifting scheme
- LWT
  - See* lifting wavelet transform
  
- M**
- Mallat algorithm 1-24
- merge
  - See* wavelet packets
- Mexican hat wavelet
  - definition 6-80
  - presentation 1-46
- Meyer wavelet
  - definition 6-78
  - presentation 1-47
- M-files
  - for wavelet families 7-4
- minimax 6-100
- missing data 4-46
- More Display Options button 2-48
- More on Residuals for Wavelet 1-D Compression
  - window 2-54
- Morlet wavelet
  - definition 6-81
  - presentation 1-46
- multiresolution 6-28
- multistep 1-35
  
- N**
- node
  - action A-23
- noise
  - ARMA 4-15
  - colored 4-26
  - Gaussian 6-95
  - processing 6-95
  - removing using GUI option 2-49
  - suppressing 3-15
    - See also* de-noising
  - unscaled 6-102
  - white 6-97
- nondecimated DWT
  - See* transforms (stationary wavelet)
  
- O**
- objects B-3
- orthogonal wavelets 6-5
- outliers
  - suppressing 4-45
  
- P**
- packets
  - See* wavelet packets
- padding
  - See* border distortion
- pattern
  - adapted wavelet 2-216
  - detection 2-216
- perfect reconstruction 6-52
- periodic-padding
  - signal extension 6-36
- periodized wavelet transform
  - See* border distortion
- polynomial + white noise example 4-16
- polyphase matrix 6-53

positions 1-24

predefined wavelet families

  type 1 7-4

  type 2 7-5

  type 3 7-5

  type 4 7-6

  type 5 7-6

## Q

quadrature mirror filters (QMF)

  and scaling function 1-34

  creating the waveform 1-32

  orthfilt function 6-21

  system 1-30

## R

ramp + colored noise example 4-26

ramp + white noise example 4-24

random design

*See* regression estimation

real electricity consumption signal example

  4-34

reconstruction

  approximation 1-30

  definition 1-29

  detail 1-30

  filters 1-30

  M-files 6-32

  multistep 1-35

  one step 6-27

  one-dimensional IDWT 6-24

  two-dimensional IDWT 6-25

redundancy 6-61

regression estimation

  goal 6-118

  one-dimensional wavelet 2-140

regularity

  definition 6-62

  wavelet families 6-88

resemblance index 3-10

residuals display

  1-D discrete wavelet compression 2-54

  1-D stationary wavelet decomposition 2-118

  2-D discrete wavelet compression 2-87

  2-D stationary wavelet decomposition 2-136

reverse biorthogonal wavelets 6-82

RGB images

  colormap matrix 2-97

  converting from 2-101

## S

save

*See* exporting from the GUI

scal2frq function 3-14

scale

  and frequency 1-20

  choosing using command line 2-7

  choosing using graphical interface 2-23

  definition 1-16

  dyadic

    definition 6-4

    for DWT 1-24

  to frequency

    display 2-13

    relationship 6-66

    scal2frq function 3-14

scale factor 1-16

scale mode A-18

scaling filters

  definition 6-19

  notation 6-4

scaling functions  
   definition 1-34  
   notation 6-3  
   shapes 6-6  
 scalogram 8-469  
 second-derivative discontinuity example 4-22  
 self-similarity  
   detecting 3-10  
 Separate Mode 2-47  
 Shannon wavelets 6-86  
 shift 1-17  
   *See also* translation  
 Show and Scroll Mode 2-47  
 Show and Scroll Mode (Stem Cfs) 2-47  
 shrink  
   *See* thresholding  
 signal extensions  
   border distortion 6-35  
 signal-end effects  
   *See* border distortion  
 sine + white noise example 4-28  
 smooth padding  
   signal extension 6-36  
 splines  
   biorthogonal family 6-79  
   filter lengths 6-28  
 split  
   *See* wavelet packets  
 Square Mode 2-81  
 stationary wavelet transform (SWT) 6-44  
 step signal example 4-18  
 STFT  
   *See* Fourier analysis  
 sum of sines example 4-8  
 Superimpose Mode 2-47  
 support  
   *See* wavelet families

SWT  
   *See* stationary wavelet transform (SWT)  
 symlets  
   definition 6-74  
   presentation 1-45  
 symmetrization  
   signal extension 6-36  
 symmetry  
   *See* wavelet families  
 synthesis  
   inverse transform 6-15  
   wavelet reconstruction 1-29

## T

thresholding  
   for optimal de-noising 2-37  
   hard 6-99  
   interval dependent 6-108  
   rules  
     tptr options 6-100  
   soft 6-99  
   strategies 6-125  
   *See also* de-noising, compression  
 thselect M-file 6-100  
 time-scale  
   *See* analysis  
 transforms  
   continuous versus discrete 6-13  
   continuous wavelet (CWT) 1-15  
   discrete wavelet (DWT) 1-24  
   fast wavelet (FWT) 6-19  
   integer to integer 6-59  
   inverse (IDWT)  
     reconstruction 1-29  
     synthesis 6-15  
   inverse lifting wavelet transform (ILWT) 6-56

- inverse stationary wavelet (ISWT) 6-50
- lifting wavelet (LWT) 6-56
- stationary wavelet (SWT) 6-44
- translation invariant 6-44
- transient 1-9
  - See also* breakdowns
- translation 6-9
  - using the mouse A-5
  - See also* shift
- translation invariance 6-44
- Tree Mode
  - definition 2-47
  - features 2-82
- trees
  - best 5-11
  - best-level 6-152
  - decomposition 1-36
  - mode
    - using 2-82
  - objects B-3
  - Tree Mode 2-47
  - wavelet
    - two-dimensional 6-27
  - wavelet decomposition 1-27
  - wavelet packet
    - notation 6-146
    - subtrees 6-152
  - wavelet packet decomposition 1-36
- trend 1-9
  - See also* long-term evolution
- triangle + a sine + noise example 4-32
- triangle + a sine example 4-30
- True 2-317
- true compression for images 6-130
- twin-scale relations
  - definition 6-19
- two proximal discontinuities example 4-20

## U

- uniform white noise example 4-12
- upsampling
  - two-dimensional IDWT 6-27
  - wavelet reconstruction process 1-29
- Using Wavelet Coefficients 2-168

## V

- vanishing moments
  - suppression of signals 6-62
  - wavelet families 6-88
- variance adaptive thresholding 6-107
- view axes A-13

## W

- Wavelet 1-D De-Noising window 3-18
- Wavelet 2-D Compression window 3-26
- Wavelet 2-D tool
  - fingerprint example 3-26
- wavelet analysis
  - advantage over Fourier 3-4
  - as Fourier-type function 3-12
  - de-noising signals 3-18
  - revealing signal trends 3-9
- wavelet coefficients 1-15
- wavelet families
  - adding new 7-2
  - criteria 6-71
  - full name 7-2
  - notation 6-3
  - properties (Part 1) 6-88
  - properties (Part 2) 6-90
  - regularity
    - advantage 6-71
    - definition 6-62



- short name 7-3
- support 6-71
- symmetry 6-71
- vanishing moments 6-71
- wavelet filters
  - notation 6-4
- wavelet notation
  - associated family 6-4
- Wavelet Packet 1-D Compression window 5-13
- Wavelet Packet 1-D menu item 5-7
- Wavelet Packet 1-D tool
  - starting 5-7
- Wavelet Packet 2-D Compression window 5-28
- wavelet packets
  - analysis
    - definition 1-36
  - and wavelet analysis
    - differences 6-137
  - atoms 6-143
  - bases 6-146
  - best level decomposition 6-152
  - best level feature 1-37
  - best tree 5-11
  - best tree feature 1-37
  - besttree function 5-4
  - building 6-141
  - compression 6-154
  - computing the best tree 5-20
  - decomposition 6-153
  - decomposition tree
    - complete binary tree 1-36
    - subtrees 6-152
  - definition 6-136
  - de-noising
    - ideas 6-154
    - using SURE 5-16
  - finding best level 5-4
  - frequency order 6-144
  - from wavelets to 6-136
  - merge 6-147
  - natural order 6-144
  - objects B-3
  - organization 6-146
  - selecting threshold for compression 5-13
  - split 6-147
  - tree
    - notation 6-146
- wavelets
  - "lazy" 6-54
  - adapted to a pattern 2-216
  - adding new 7-2
  - applications 3-1
  - associated family 6-8
  - Battle-Lemarie 6-79
  - biorthogonal
    - definition 6-76
    - presentation 1-43
  - candidates 6-63
  - coiflets
    - definition 6-75
    - presentation 1-45
  - complex frequency B-spline 6-85
  - complex Gaussian 6-84
  - complex Morlet 6-84
  - complex Shannon 6-86
  - Daubechies
    - definition 6-72
    - presentation 1-42
  - decomposition tree 1-27
  - defining order 7-4
  - determining type 7-3
  - discrete Meyer 6-83
  - families 1-39
  - Gaussian 6-82

- Haar
    - definition 6-73
    - presentation 1-41
  - history 1-38
  - lifted 6-54
  - Mexican hat
    - definition 6-80
    - presentation 1-46
  - Meyer
    - definition 6-78
    - presentation 1-47
  - Morlet
    - definition 6-81
    - presentation 1-46
  - notation 6-3
  - one-dimensional analysis 6-6
  - one-dimensional capabilities
    - objects 6-32
    - table 2-29
  - organization 6-11
  - relationship of filters to 1-32
  - reverse biorthogonal 6-82
  - shapes 6-6
  - shifted 1-16
  - symlets
    - definition 6-74
    - presentation 1-45
  - translation 6-9
    - See also* shift
  - tree
    - one-dimensional 1-36
    - two-dimensional 6-27
  - two-dimensional 6-8
  - two-dimensional analysis 6-6
  - two-dimensional capabilities
    - objects 6-33
    - table 2-66
  - vanishing moments
    - number of 6-71
    - suppression of signals 6-62
  - wavelets.asc file 7-16
  - wavelets.inf file 7-16
  - wavelets.prv file 7-16
  - wavemngr command 7-2
  - wscalogram 8-469
  - wthresh command 2-112
- Z**
- zero-padding
    - signal extension 6-35
  - zoom 2-15