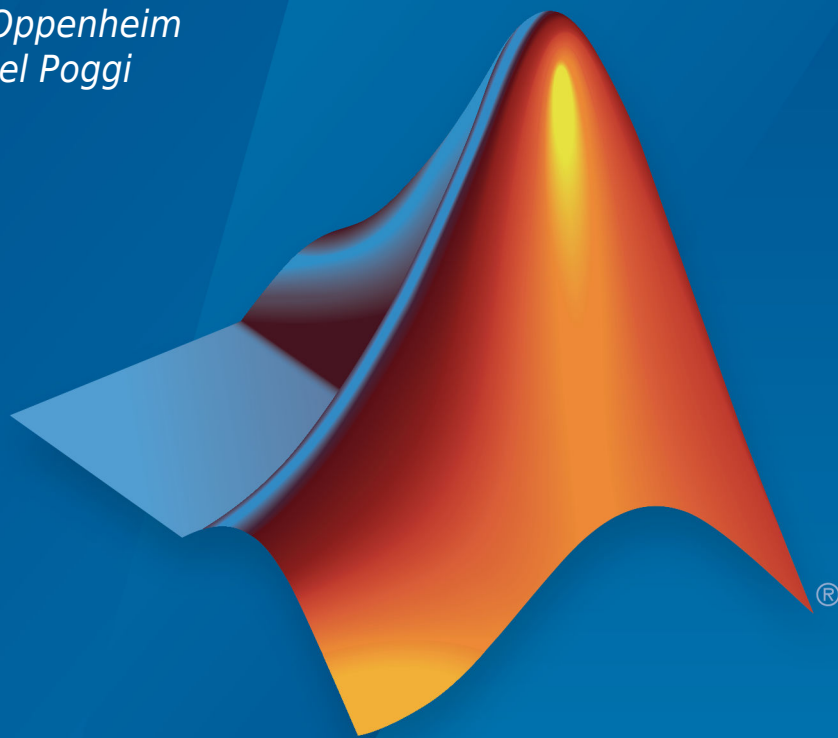


Wavelet Toolbox™

User's Guide

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



MATLAB®

R2019b

 MathWorks®

How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Wavelet Toolbox™ User's Guide

© COPYRIGHT 1997–2019 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 for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

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

Revision History

March 1997	First printing	New for Version 1.0
September 2000	Second printing	Revised for Version 2.0 (Release 12)
June 2001	Online only	Revised for Version 2.1 (Release 12.1)
July 2002	Online only	Revised for Version 2.2 (Release 13)
June 2004	Online only	Revised for Version 3.0 (Release 14)
July 2004	Third printing	Revised for Version 3.0
October 2004	Online only	Revised for Version 3.0.1 (Release 14SP1)
March 2005	Online only	Revised for Version 3.0.2 (Release 14SP2)
June 2005	Fourth printing	Minor revision for Version 3.0.2
September 2005	Online only	Minor revision for Version 3.0.3 (Release R14SP3)
March 2006	Online only	Minor revision for Version 3.0.4 (Release 2006a)
September 2006	Online only	Revised for Version 3.1 (Release 2006b)
March 2007	Online only	Revised for Version 4.0 (Release 2007a)
September 2007	Online only	Revised for Version 4.1 (Release 2007b)
October 2007	Fifth printing	Revised for Version 4.1
March 2008	Online only	Revised for Version 4.2 (Release 2008a)
October 2008	Online only	Revised for Version 4.3 (Release 2008b)
March 2009	Online only	Revised for Version 4.4 (Release 2009a)
September 2009	Online only	Minor revision for Version 4.4.1 (Release 2009b)
March 2010	Online only	Revised for Version 4.5 (Release 2010a)
September 2010	Online only	Revised for Version 4.6 (Release 2010b)
April 2011	Online only	Revised for Version 4.7 (Release 2011a)
September 2011	Online only	Revised for Version 4.8 (Release 2011b)
March 2012	Online only	Revised for Version 4.9 (Release 2012a)
September 2012	Online only	Revised for Version 4.10 (Release 2012b)
March 2013	Online only	Revised for Version 4.11 (Release 2013a)
September 2013	Online only	Revised for Version 4.12 (Release 2013b)
March 2014	Online only	Revised for Version 4.13 (Release 2014a)
October 2014	Online only	Revised for Version 4.14 (Release 2014b)
March 2015	Online only	Revised for Version 4.14.1 (Release 2015a)
September 2015	Online only	Revised for Version 4.15 (Release 2015b)
March 2016	Online only	Revised for Version 4.16 (Release 2016a)
September 2016	Online only	Revised for Version 4.17 (Release 2016b)
March 2017	Online only	Revised for Version 4.18 (Release 2017a)
September 2017	Online only	Revised for Version 4.19 (Release 2017b)
March 2018	Online only	Revised for Version 5.0 (Release 2018a)
September 2018	Online only	Revised for Version 5.1 (Release 2018b)
March 2019	Online only	Revised for Version 5.2 (Release 2019a)
September 2019	Online only	Revised for Version 5.3 (Release 2019b)

Acknowledgments

Acknowledgments	xviii
-----------------------	-------

Wavelets, Scaling Functions, and Conjugate Quadrature Mirror Filters

1

Wavelet Families	1-2
Daubechies Wavelets: dbN	1-6
Symlet Wavelets: symN	1-7
Coiflet Wavelets: coifN	1-8
Biorthogonal Wavelet Pairs: biorNr.Nd	1-9
Meyer Wavelet: meyr	1-11
Gaussian Derivatives Family: gaus	1-13
Mexican Hat Wavelet: mexh	1-13
Morlet Wavelet: morl	1-14
Additional Real Wavelets	1-15
Complex Wavelets	1-15
Wavelet Families and Associated Properties – I	1-20
Wavelet Families and Associated Properties – II	1-22
Lifting Method for Constructing Wavelets	1-24
Lifting Background	1-25
Polyphase Representation	1-27
Split, Predict, and Update	1-28
Haar Wavelet Via Lifting	1-29
Bior2.2 Wavelet Via Lifting	1-30
Lifting Functions	1-31

Primal Lifting from Haar	1-33
Integer-to-Integer Wavelet Transform	1-34
Orthogonal and Biorthogonal Filter Banks	1-36
Scaling Function and Wavelet	1-49
Lifting a Filter Bank	1-53
Add Quadrature Mirror and Biorthogonal Wavelet Filters ..	1-59
Least Asymmetric Wavelet and Phase	1-72

Continuous Wavelet Analysis

2

1-D Continuous Wavelet Analysis	2-2
Continuous Wavelet Analysis of Noisy Sinusoid Using Command Line Functions	2-4
Continuous Wavelet Analysis of Noisy Sinusoid Using the Wavelet Analyzer App	2-8
Importing and Exporting Information from the Wavelet Analyzer App	2-19
Loading Signals	2-19
Saving Wavelet Coefficients	2-19
Morse Wavelets	2-21
What Are Morse Wavelets?	2-21
Morse Wavelet Parameters	2-22
Effect of Parameter Values on Morse Wavelet Shape	2-22
Relationship Between Analytic Morse Wavelet and Analytic Signal	2-24
Comparison of Analytic Wavelet Transform and Analytic Signal Coefficients	2-25
Recommended Morse Wavelet Settings for the CWT	2-30
References	2-31

Boundary Effects and the Cone of Influence	2-32
Time-Frequency Analysis and Continuous Wavelet Transform	2-45
Continuous Wavelet Analysis of Modulated Signals	2-57
Remove Time-Localized Frequency Components	2-61
Time-Varying Coherence	2-67
Continuous Wavelet Analysis of Cusp Signal	2-72
Complex Continuous Analysis Using the Wavelet Analyzer App	2-75
DFT-Based Continuous Wavelet Analysis Using the Wavelet Analyzer App	2-80
Manual Selection of CWT Coefficients	2-85
Two-Dimensional CWT of Noisy Pattern	2-89
2-D Continuous Wavelet Transform App	2-98
2-D Continuous Wavelet Transform	2-98
2-D CWT App Example	2-99

Discrete Wavelet Analysis

3

Critically Sampled and Oversampled Wavelet Filter Banks ...	3-2
Double-Density Wavelet Transform	3-3
Dual-Tree Complex Wavelet Transform	3-6
Dual-Tree Double-Density Wavelet Transforms	3-9
1-D Decimated Wavelet Transforms	3-11
Analysis-Decomposition Functions	3-11
Synthesis-Reconstruction Functions	3-11
Decomposition Structure Utilities	3-11
Denoising and Compression	3-12
1-D Analysis Using the Command Line	3-13

1-D Analysis Using the Wavelet Analyzer App	3-21
Importing and Exporting Information from the Wavelet Analyzer App	3-34
Fast Wavelet Transform (FWT) Algorithm	3-43
Filters Used to Calculate the DWT and IDWT	3-43
Algorithms	3-46
Why Does Such an Algorithm Exist?	3-51
1-D Wavelet Capabilities	3-54
2-D Wavelet Capabilities	3-55
Border Effects	3-57
Signal Extensions: Zero-Padding, Symmetrization, and Smooth Padding	3-57
Nondecimated Discrete Stationary Wavelet Transforms (SWTs)	3-66
ε -Decimated DWT	3-66
How to Calculate the ε -Decimated DWT: SWT	3-67
Inverse Discrete Stationary Wavelet Transform (ISWT)	3-71
More About SWT	3-72
1-D Stationary Wavelet Transform	3-73
Analysis-Decomposition Functions	3-73
Synthesis-Reconstruction Functions	3-73
1-D Analysis Using the Command Line	3-74
Interactive 1-D Stationary Wavelet Transform Denoising	3-83
Importing and Exporting from the Wavelet Analysis App	3-86
Wavelet Changepoint Detection	3-88
Scale-Localized Volatility and Correlation	3-103
R Wave Detection in the ECG	3-114
Wavelet Cross-Correlation for Lead-Lag Analysis	3-125
1-D Multisignal Analysis	3-138
1-D Multisignal Analysis — Command Line	3-138
1-D Multisignal Analysis Using the Wavelet Analyzer App	3-147
Importing and Exporting Information from the Wavelet Analyzer App	3-180

2-D Discrete Wavelet Analysis	3-188
Analysis-Decomposition Functions	3-188
Synthesis-Reconstruction Functions	3-188
Decomposition Structure Utilities	3-188
Denoising and Compression	3-189
2-D Analysis — Command Line	3-189
2-D Wavelet Analysis Using the Wavelet Analyzer App	3-195
Importing and Exporting Information from the Wavelet Analyzer App	3-205
2-D Stationary Wavelet Transform	3-214
Analysis-Decomposition Function	3-214
Synthesis-Reconstruction Function	3-214
2-D Analysis Using the Command Line	3-214
Interactive 2-D Stationary Wavelet Transform Denoising ...	3-222
Importing and Exporting Information from the Wavelet Analyzer App	3-225
Shearlet Systems	3-227
Shearlets	3-227
Transform Type	3-229
References	3-229
3-D Discrete Wavelet Analysis	3-231
Performing 3-D Analysis Using the Command Line	3-231
Performing 3-D Analysis Using the Wavelet Analyzer App ..	3-232
Importing and Exporting Information from the Wavelet Analyzer App	3-239
Dual-Tree Wavelet Transforms	3-243
Analytic Wavelets Using the Dual-Tree Wavelet Transform .	3-282
Multifractal Analysis	3-286
Wavelet Analysis of Financial Data	3-306

4

Time-Frequency Gallery	4-2
Short-Time Fourier Transform (Spectrogram)	4-4
Continuous Wavelet Transform (Scalogram)	4-9
Wigner-Ville Distribution	4-11
Reassignment and Synchrosqueezing	4-13
Constant-Q Gabor Transform	4-22
Empirical Mode Decomposition and Hilbert-Huang Transform	4-24

Wavelet Packets

5

About Wavelet Packet Analysis	5-2
1-D Wavelet Packet Analysis	5-6
Starting the Wavelet Packet 1-D Tool	5-6
Importing a Signal	5-7
Analyzing a Signal	5-7
Computing the Best Tree	5-9
Compressing a Signal Using Wavelet Packets	5-10
De-Noising a Signal Using Wavelet Packets	5-13
2-D Wavelet Packet Analysis	5-19
Starting the Wavelet Packet 2-D Tool	5-19
Compressing an Image Using Wavelet Packets	5-21
Importing and Exporting from Wavelet Analyzer App	5-25
Saving Information to Disk	5-25
Loading Information into the Graphical Tools	5-28
Wavelet Packets	5-32
From Wavelets to Wavelet Packets	5-32
Wavelet Packets in Action: An Introduction	5-33
Building Wavelet Packets	5-36
Wavelet Packet Atoms	5-39
Organizing the Wavelet Packets	5-40

Choosing the Optimal Decomposition	5-42
Some Interesting Subtrees	5-46
Wavelet Packets 2-D Decomposition Structure	5-50
Wavelet Packets for Compression and Denoising	5-50
Introduction to Object-Oriented Features	5-51
Objects in the Wavelet Toolbox Software	5-52
Examples Using Wavelet Packet Tree Objects	5-53
plot and wpviewcf	5-53
drawtree and readtree	5-57
Change Terminal Node Coefficients	5-59
Thresholding Wavelet Packets	5-61
Description of Objects in the Wavelet Toolbox Software	5-65
WTBO Object	5-65
NTREE Object	5-66
Private	5-66
DTREE Object	5-66
WPTREE Object	5-68
Build Wavelet Tree Objects	5-71
Building a Wavelet Tree Object (WTREE)	5-71
Building a Right Wavelet Tree Object (RWVTREE)	5-72
Building a Wavelet Tree Object (WVTREE)	5-73
Building a Wavelet Tree Object (EDWTTREE)	5-75

Denoising, Nonparametric Function Estimation, and Compression

6

Wavelet Denoising and Nonparametric Function Estimation	6-2
Denoising Methods	6-4
Soft or Hard Thresholding	6-6
Dealing with Unscaled Noise and Nonwhite Noise	6-7
Wavelet Denoising in Action	6-8
Extension to Image Denoising	6-14
1-D Wavelet Variance Adaptive Thresholding	6-16

Wavelet Denoising Analysis Measurements	6-20
Wavelet Denoising	6-21
Denoise a Signal with the Wavelet Signal Denoiser	6-29
Translation Invariant Wavelet Denoising with Cycle Spinning	6-43
1-D Cycle Spinning	6-43
1-D Adaptive Thresholding of Wavelet Coefficients	6-48
1-D Local Thresholding Using the Wavelet Analyzer App	6-48
Importing and Exporting Information from the Wavelet Analyzer App	6-56
Multivariate Wavelet Denoising	6-58
Multivariate Wavelet Denoising — Command Line	6-58
Multivariate Wavelet Denoising Using the Wavelet Analyzer App	6-63
Importing and Exporting from the Wavelet Analyzer App ...	6-68
Wavelet Multiscale Principal Components Analysis	6-70
Multiscale Principal Components Analysis — Command Line	6-70
Multiscale Principal Components Analysis Using the Wavelet Analyzer App	6-74
Importing and Exporting from the Wavelet Analyzer App ...	6-78
Wavelet Data Compression	6-80
Compression Scores	6-82
Wavelet Compression for Images	6-84
Effects of Quantization	6-84
True Compression Methods	6-86
Quantitative and Perceptual Quality Measures	6-88
More Information on True Compression	6-89
2-D Wavelet Compression	6-90
2-D Wavelet Compression Commands	6-90
2-D Wavelet Compression using the Wavelet Analyzer App ..	6-97
Importing and Exporting from the Wavelet Analyzer App ...	6-110

Univariate Wavelet Regression	6-111
Regression for Equally-Spaced Observations	6-111
Regression for Randomly-Spaced Observations	6-114
Importing and Exporting Information from the Wavelet Analyzer App	6-115

Matching Pursuit

7

Matching Pursuit Algorithms	7-2
Redundant Dictionaries and Sparsity	7-2
Nonlinear Approximation in Dictionaries	7-3
Basic Matching Pursuit	7-3
Orthogonal Matching Pursuit	7-6
Weak Orthogonal Matching Pursuit	7-6
 Matching Pursuit	 7-8
Matching Pursuit Dictionary Creation and Visualization	7-8
Orthogonal Matching Pursuit on a 1-D Signal	7-10
Electricity Consumption Analysis Using Matching Pursuit ...	7-12
 Matching Pursuit Using Wavelet Analyzer App	 7-23
Matching Pursuit 1-D Interactive Tool	7-23
Interactive Matching Pursuit of Electricity Consumption Data	7-39

Generating MATLAB Code from Wavelet Toolbox Wavelet Analyzer App

8

Generate MATLAB Code for 1-D Decimated Wavelet Denoising and Compression	8-2
Wavelet 1-D Denoising	8-2
 Generate MATLAB Code for 2-D Decimated Wavelet Denoising and Compression	 8-11
2-D Decimated Discrete Wavelet Transform Denoising	8-11

2-D Decimated Discrete Wavelet Transform Compression . . .	8-14
Generate MATLAB Code for 1-D Stationary Wavelet Denoising	
.	8-17
1-D Stationary Wavelet Transform Denoising	8-17
Generate MATLAB Code for 2-D Stationary Wavelet Denoising	
.	8-23
2-D Stationary Wavelet Transform Denoising	8-23
Generate MATLAB Code for 1-D Wavelet Packet Denoising and Compression	
.	8-27
1-D Wavelet Packet Denoising	8-27
Generate MATLAB Code for 2-D Wavelet Packet Denoising and Compression	
.	8-31
2-D Wavelet Packet Compression	8-31
Generate Code to Denoise a Signal	8-35
Code Generation Support, Usage Notes, and Limitations . . .	8-37

Wavelet Analyzer App Features Summary

A

General Features	A-2
Color Coding	A-2
Connection of Plots	A-3
Using the Mouse	A-5
Controlling the Colormap	A-7
Using Menus	A-9
Using the View Axes Button	A-10
Continuous Wavelet Tool Features	A-12
Wavelet 2-D Tool Features	A-13
Wavelet Packet Tool Features (1-D and 2-D)	A-14
Coefficients Coloration	A-14
Node Action	A-14

Node Label	A-14
Node Action Functionality	A-14
Wavelet Display Tool	A-16
Wavelet Packet Display Tool	A-17

Acknowledgments

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

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

Other colleagues and friends who have helped us enormously are Patrice Abry (ENS Lyon), Samir Akkouche (Ecole Centrale de Lyon), Mark Asch (Paris 11), Patrice Assouad (Paris 11), Roger Astier (Paris 11), Jean Coursol (Paris 11), Didier Dacunha-Castelle (Paris 11), Claude Deniau (Marseille), Patrick Flandrin (Ecole Normale de Lyon), Eric Galin (Ecole Centrale de Lyon), Christine Graffigne (Paris 5), Anatoli Juditsky (Grenoble), Gérard Kerkyacharian (Paris 10), Gérard Malgouyres (Paris 11), Olivier Nowak (Ecole Centrale de Lyon), Dominique Picard (Paris 7), and Franck Tarpin-Bernard (Ecole Centrale de Lyon).

One of our first opportunities to apply the ideas of wavelets connected with signal analysis and its modeling occurred in collaboration 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.

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 has published numerous

theoretical papers and carried out applications in close collaboration with industrial teams. For instance:

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

Notes by Yves Meyer

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

There are lots of successes for the community to share. Why? Probably because the time is ripe. Fourier techniques were liberated by the appearance of windowed Fourier methods that operate locally on a time-frequency approach. In another direction, Burt-Adelson's pyramidal algorithms, the quadrature mirror filters, and filter banks and subband coding are available. The mathematics underlying those algorithms existed earlier, but new computing techniques enabled researchers to try out new ideas rapidly. The numerical image and signal processing areas are blooming.

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

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

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

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

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

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

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

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

Here are the tools, waiting to be used.

Yves Meyer

Professor, Ecole Normale Supérieure de Cachan and Institut de France

Notes by Ingrid Daubechies

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

Wavelet Toolbox software is a great way to work with wavelets. The toolbox, together with the power of MATLAB® software, really allows one to write complex and powerful applications, in a very short amount of time. The Graphic User Interface is both user-friendly and intuitive. It provides an excellent interface to explore the various aspects and applications of wavelets; it takes away the tedium of typing and remembering the various function calls.

Ingrid C. Daubechies

Professor, Princeton University, Department of Mathematics and Program in Applied and Computational Mathematics

Wavelets, Scaling Functions, and Conjugate Quadrature Mirror Filters

- “Wavelet Families” on page 1-2
- “Wavelet Families and Associated Properties — I” on page 1-20
- “Wavelet Families and Associated Properties — II” on page 1-22
- “Lifting Method for Constructing Wavelets” on page 1-24
- “Orthogonal and Biorthogonal Filter Banks” on page 1-36
- “Scaling Function and Wavelet” on page 1-49
- “Lifting a Filter Bank” on page 1-53
- “Add Quadrature Mirror and Biorthogonal Wavelet Filters” on page 1-59
- “Least Asymmetric Wavelet and Phase” on page 1-72

Wavelet Families

The Wavelet Toolbox software includes a large number of wavelets that you can use for both continuous and discrete analysis. For discrete analysis, examples include orthogonal wavelets (Daubechies' extremal phase and least asymmetric wavelets) and B-spline biorthogonal wavelets. For continuous analysis, the Wavelet Toolbox software includes Morlet, Meyer, derivative of Gaussian, and Paul wavelets.

The choice of wavelet is dictated by the signal or image characteristics and the nature of the application. If you understand the properties of the analysis and synthesis wavelet, you can choose a wavelet that is optimized for your application.

Wavelet families vary in terms of several important properties. Examples include:

- Support of the wavelet in time and frequency and rate of decay.
- Symmetry or antisymmetry of the wavelet. The accompanying perfect reconstruction filters have linear phase.
- Number of vanishing moments. Wavelets with increasing numbers of vanishing moments result in sparse representations for a large class of signals and images.
- Regularity of the wavelet. Smoother wavelets provide sharper frequency resolution. Additionally, iterative algorithms for wavelet construction converge faster.
- Existence of a scaling function, φ .

For continuous analysis, the Wavelet Toolbox software analytic wavelet-based analysis for select wavelets. See `cwt` and `icwt` for details. "Inverse Continuous Wavelet Transform" for a basic theoretical motivation. "CWT-Based Time-Frequency Analysis" illustrates the use of the continuous wavelet transform for simulated and real-world signals.

Entering `waveinfo` at the command line displays a survey of the main properties of available wavelet families. For a specific wavelet family, use `waveinfo` with the wavelet family short name. You can find the wavelet family short names listed in the following table and on the reference page for `waveinfo`.

Wavelet Family Short Name	Wavelet Family Name
'haar'	Haar wavelet
'db'	Daubechies wavelets
'sym'	Symlets

Wavelet Family Short Name	Wavelet Family Name
'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 (also known as the Ricker wavelet)
'morl'	Morlet wavelet
'cgau'	Complex Gaussian wavelets
'shan'	Shannon wavelets
'fbsp'	Frequency B-Spline wavelets
'cmor'	Complex Morlet wavelets
'fk'	Fejer-Korovkin wavelets

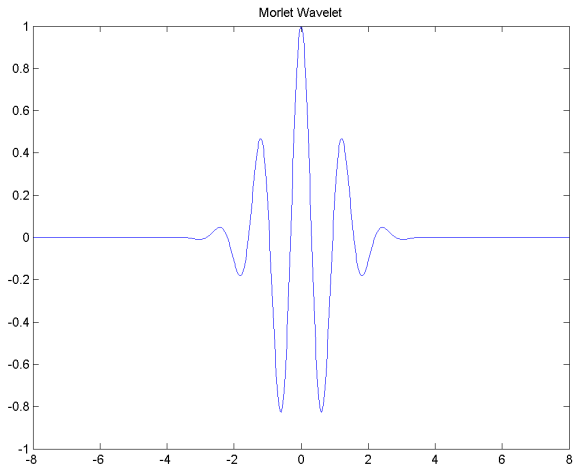
To display detailed information about the Daubechies' least asymmetric orthogonal wavelets, enter:

```
waveinfo('sym')
```

To compute the wavelet and scaling function (if available), use `wavefun`.

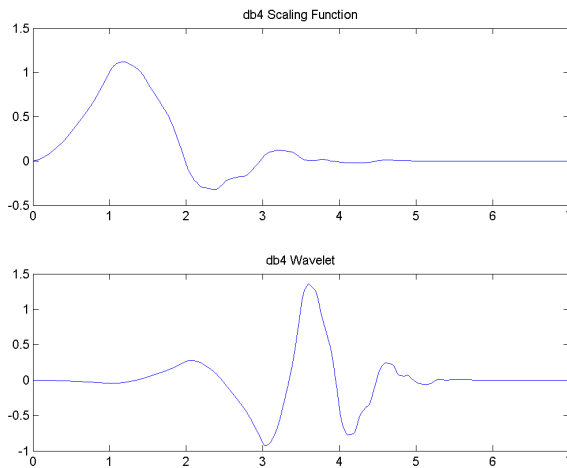
The Morlet wavelet is suitable for continuous analysis. There is no scaling function associated with the Morlet wavelet. To compute the Morlet wavelet, you can enter:

```
[psi,xval] = wavefun('morl',10);
plot(xval,psi); title('Morlet Wavelet');
```



For wavelets associated with a multiresolution analysis, you can compute both the scaling function and wavelet. The following code returns the scaling function and wavelet for the Daubechies' extremal phase wavelet with 4 vanishing moments.

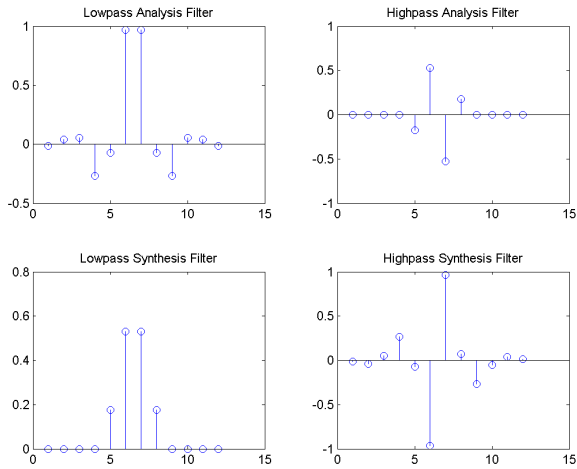
```
[phi,psi,xval] = wavefun('db4',10);  
subplot(211);  
plot(xval,phi);  
title('db4 Scaling Function');  
subplot(212);  
plot(xval,psi);  
title('db4 Wavelet');
```



In discrete wavelet analysis, the analysis and synthesis filters are of more interest than the associated scaling function and wavelet. You can use `wfilters` to obtain the analysis and synthesis filters.

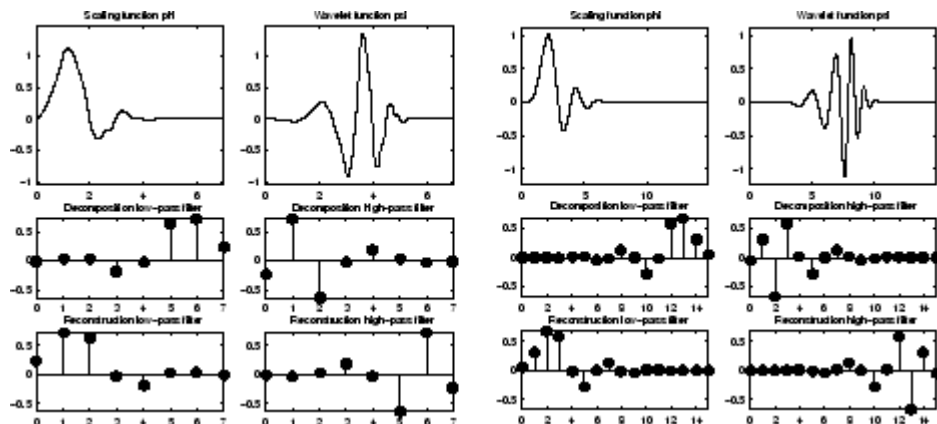
Obtain the decomposition (analysis) and reconstruction (synthesis) filters for the B-spline biorthogonal wavelet. Specify 3 vanishing moments in the synthesis wavelet and 5 vanishing moments in the analysis wavelet. Plot the filters' impulse responses.

```
[LoD,HiD,LoR,HiR] = wfilters('bior3.5');
subplot(221);
stem(LoD);
title('Lowpass Analysis Filter');
subplot(222);
stem(HiD);
title('Highpass Analysis Filter');
subplot(223);
stem(LoR);
title('Lowpass Synthesis Filter');
subplot(224);
stem(HiR);
title('Highpass Synthesis Filter');
```



Daubechies Wavelets: dbN

The dbN wavelets are the Daubechies' extremal phase wavelets. N refers to the number of vanishing moments. These filters are also referred to in the literature by the number of filter taps, which is $2N$. More about this family can be found in [Dau92] page 195. Enter `waveinfo('db')` at the MATLAB command prompt to obtain a survey of the main properties of this family.

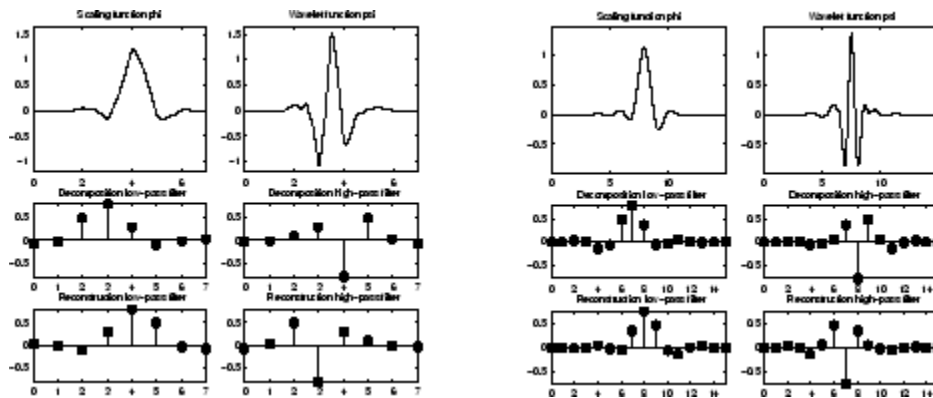


Daubechies Wavelets db4 on the Left and db8 on the Right

The db1 wavelet is also known as the Haar wavelet. The Haar wavelet is the only orthogonal wavelet with linear phase. Using `waveinfo('haar')`, you can obtain a survey of the main properties of this wavelet.

Symlet Wavelets: symN

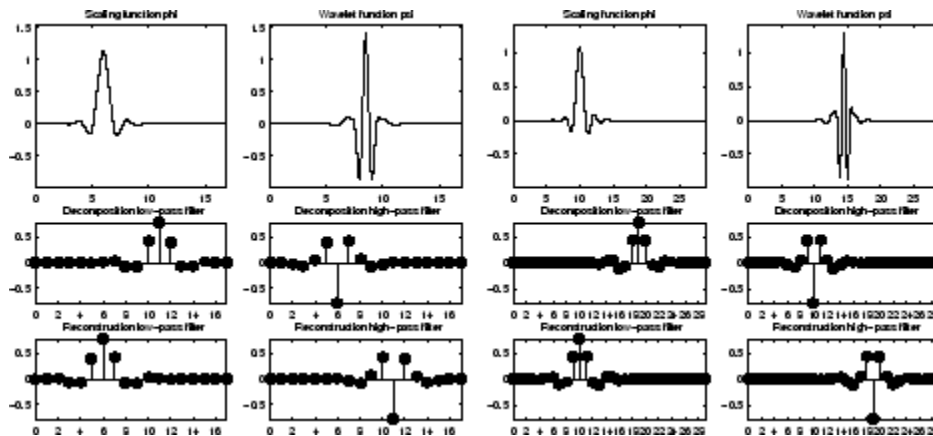
The symN wavelets are also known as Daubechies' least-asymmetric wavelets. The symlets are more symmetric than the extremal phase wavelets. In symN, N is the number of vanishing moments. These filters are also referred to in the literature by the number of filter taps, which is $2N$. More about symlets can be found in [Dau92], pages 198, 254-257. Enter `waveinfo('sym')` at the MATLAB command prompt to obtain a survey of the main properties of this family.



Symlets `sym4` on the Left and `sym8` on the Right

Coiflet Wavelets: `coifN`

Coiflet scaling functions also exhibit vanishing moments. In `coifN`, N is the number of vanishing moments for both the wavelet and scaling functions. These filters are also referred to in the literature by the number of filter coefficients, which is $3N$. For the coiflet construction, see [Dau92] pages 258–259. Enter `waveinfo('coif')` at the MATLAB command prompt to obtain a survey of the main properties of this family.



Coiflets `coif3` on the Left and `coif5` on the Right

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

Biorthogonal Wavelet Pairs: biorNr.Nd

While the Haar wavelet is the only orthogonal wavelet with linear phase, you can design biorthogonal wavelets with linear phase.

Biorthogonal wavelets feature a pair of scaling functions and associated scaling filters — one for analysis and one for synthesis.

There is also a pair of wavelets and associated wavelet filters — one for analysis and one for synthesis.

The analysis and synthesis wavelets can have different numbers of vanishing moments and regularity properties. You can use the wavelet with the greater number of vanishing moments for analysis resulting in a sparse representation, while you use the smoother wavelet for reconstruction.

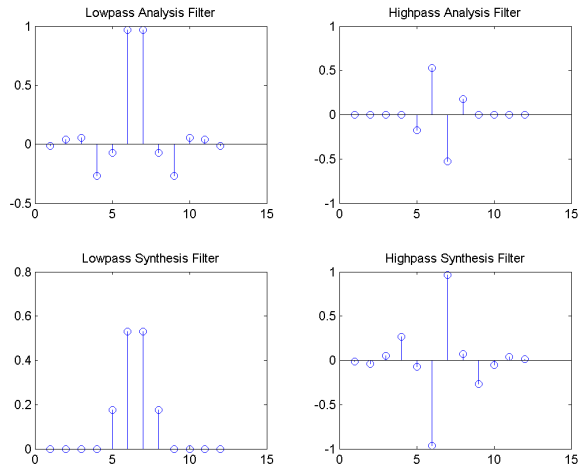
See [Dau92] pages 259, 262–85 and [Coh92] for more details on the construction of biorthogonal wavelet bases. Enter `waveinfo('bior')` at the command line to obtain a survey of the main properties of this family.

The following code returns the B-spline biorthogonal reconstruction and decomposition filters with 3 and 5 vanishing moments and plots the impulse responses.

The impulse responses of the lowpass filters are symmetric with respect to the midpoint. The impulse responses of the highpass filters are antisymmetric with respect to the midpoint.

```
[LoD,HiD,LoR,HiR] = wfilters('bior3.5');
subplot(221);
stem(LoD);
title('Lowpass Analysis Filter');
subplot(222);
stem(HiD);
title('Highpass Analysis Filter');
```

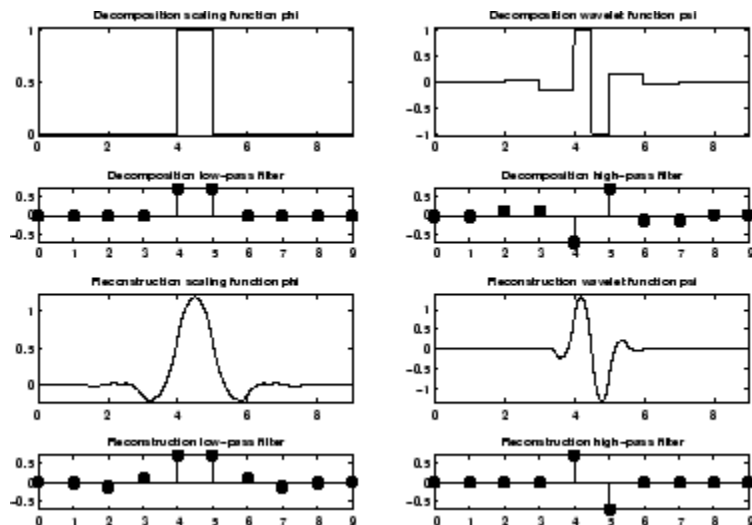
```
subplot(223);  
stem(LoR);  
title('Lowpass Synthesis Filter');  
subplot(224);  
stem(HiR);  
title('Highpass Synthesis Filter');
```



Reverse Biorthogonal Wavelet Pairs: rbioNr.Nd

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

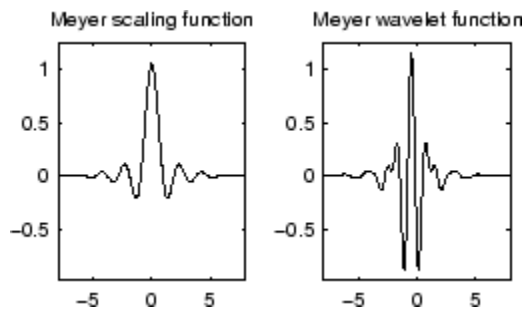
You can obtain a survey of the main properties of this family by typing `waveinfo('rbio')` from the MATLAB command line.



Reverse Biorthogonal Wavelet rbio1.5

Meyer Wavelet: meyr

Both ψ and φ are defined in the frequency domain, starting with an auxiliary function ν (see [Dau92] pages 117, 119, 137, 152). By typing `waveinfo('meyr')` at the MATLAB command prompt, you can obtain a survey of the main properties of this wavelet.



Meyer Wavelet

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

- Wavelet function

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

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

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

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

- Scaling function

$$\widehat{\phi}(\omega) = (2 \quad \text{if } |\omega| \leq \frac{2\pi}{3}$$

$$\widehat{\phi}(\omega) = (2 \cos\left(\frac{\pi}{2} \nu\left(\frac{3}{2\pi}|\omega| - 1\right)\right)) \quad \text{if } \frac{2\pi}{3} \leq |\omega| \leq \frac{4\pi}{3}$$

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

By changing the auxiliary function, you get a family of different wavelets. For the required properties of the auxiliary function ν (see “References” for more information). This wavelet ensures orthogonal analysis.

The function ψ does not have finite support, but ψ decreases to 0 when $x \rightarrow \infty$, faster than any inverse polynomial

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

This property holds also for the derivatives

$$\forall k \in N, \forall n \in N, \exists C_{k,n}, \text{ such that } |\psi^{(k)}(x)| \leq C_{k,n}(1 + |x|^2)^{-n}$$

The wavelet is infinitely differentiable.

Note Although the Meyer wavelet is not compactly supported, there exists a good approximation leading to FIR filters that you can use in the DWT. Enter

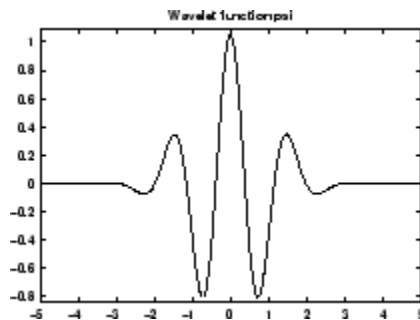
`waveinfo('dmey')` at the MATLAB command prompt to obtain a survey of the main properties of this pseudo-wavelet.

Gaussian Derivatives Family: `gaus`

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

The integer p is the parameter of this family and in the previous formula, C_p is such that $\|f^{(p)}\|^2 = 1$ where $f^{(p)}$ is the p^{th} derivative of f .

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



Gaussian Derivative Wavelet `gaus8`

Mexican Hat Wavelet: `mexh`

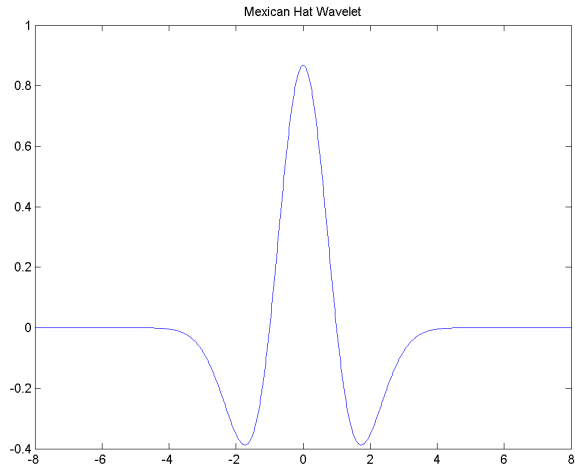
This wavelet is proportional to the second derivative function of the Gaussian probability density function. The wavelet is a special case of a larger family of derivative of Gaussian (DOG) wavelets. It is also known as the Ricker wavelet.

There is no scaling function associated with this wavelet.

Enter `waveinfo('mexh')` at the MATLAB command prompt to obtain a survey of the main properties of this wavelet.

You can compute the wavelet with `wavefun`.

```
[psi,xval] = wavefun('mexh',10);  
plot(xval,psi);  
title('Mexican Hat Wavelet');
```



Morlet Wavelet: morl

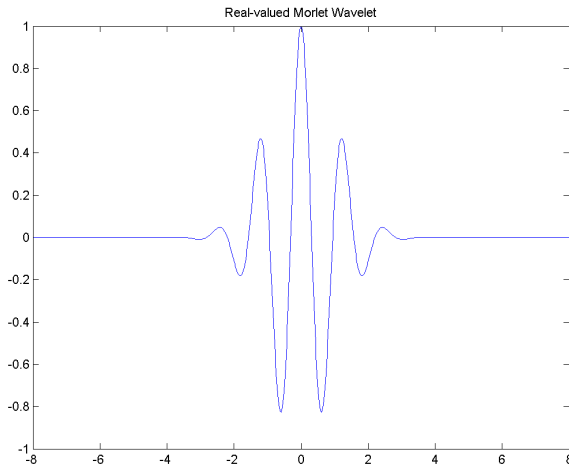
Both real-valued and complex-valued versions of this wavelet exist. Enter `waveinfo('morl')` at the MATLAB command line to obtain the properties of the real-valued Morlet wavelet.

The real-valued Morlet wavelet is defined as:

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

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

```
[psi,xval] = wavefun('morl',10);  
plot(xval,psi);  
title('Real-valued Morlet Wavelet');
```

The Morlet wavelet does not technically satisfy the admissibility condition..

Additional Real Wavelets

Some other real wavelets are available in the toolbox.

Complex Wavelets

The toolbox also provides a number of complex-valued wavelets for continuous wavelet analysis. Complex-valued wavelets provide phase information and are therefore very important in the time-frequency analysis of nonstationary signals.

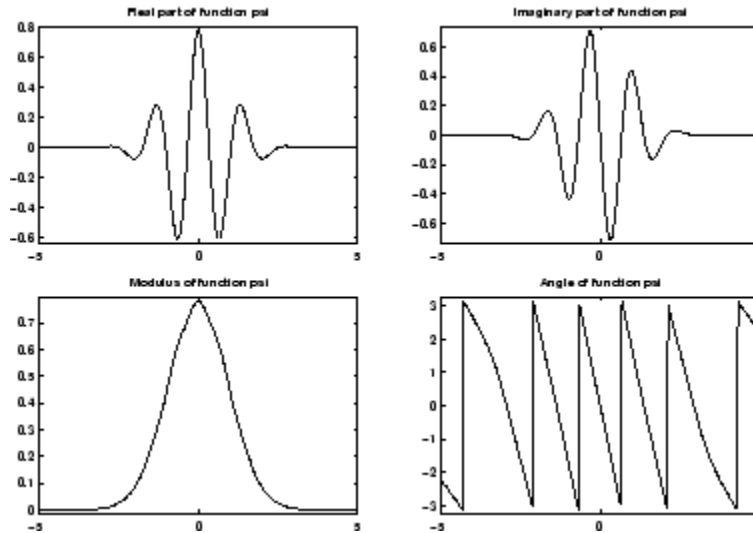
Complex Gaussian Wavelets: cgau

This family is built starting from the complex Gaussian function

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

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

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



Complex Gaussian Wavelet `cgau8`

Complex Morlet Wavelets: `cmor`

See [Teo98] pages 62-65.

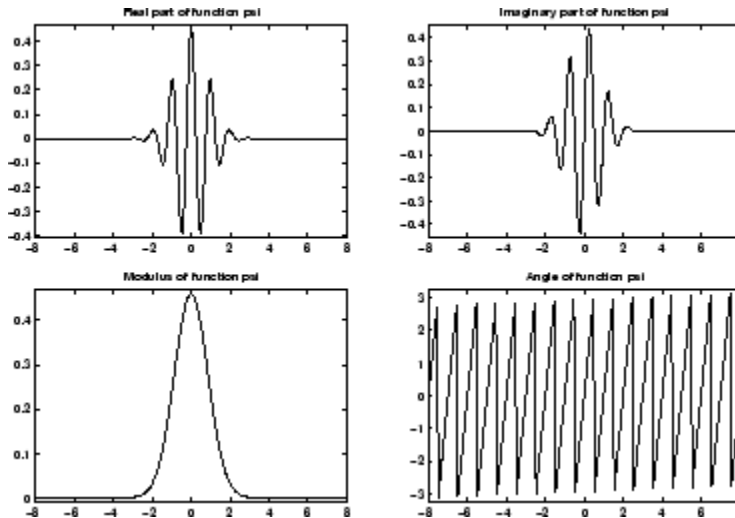
A complex Morlet wavelet is defined by

$$\psi(x) = \frac{1}{\sqrt{\pi f_b}} e^{2i\pi f_c x} e^{-\frac{x^2}{f_b}}$$

depending on two parameters:

- f_b is a bandwidth parameter.
- f_c is a wavelet center frequency.

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



Complex Morlet Wavelet morl 1.5-1

Complex Frequency B-Spline Wavelets: fbasp

See [Teo98] pages 62-65.

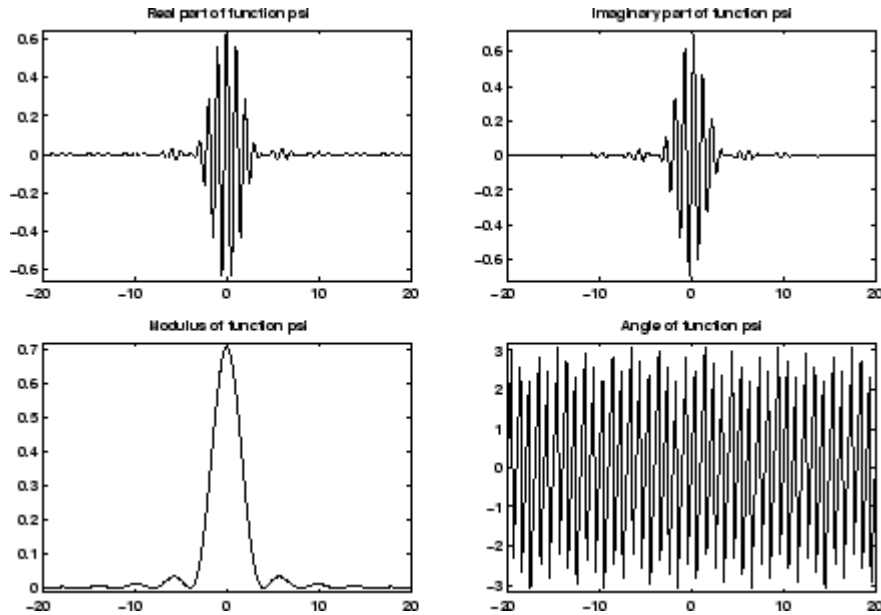
A complex frequency B-spline wavelet is defined by

$$\psi(x) = \sqrt{f_b} \left(\text{sinc} \left(\frac{f_b x}{m} \right) \right)^m e^{2\pi i f_c x}$$

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('fbasp')` from the MATLAB command line.



Complex Frequency B-Spline Wavelet fbsp 2-0.5-1

Complex Shannon Wavelets: shan

See [Teo98] pages 62-65.

This family is obtained from the frequency B-spline wavelets by setting m to 1.

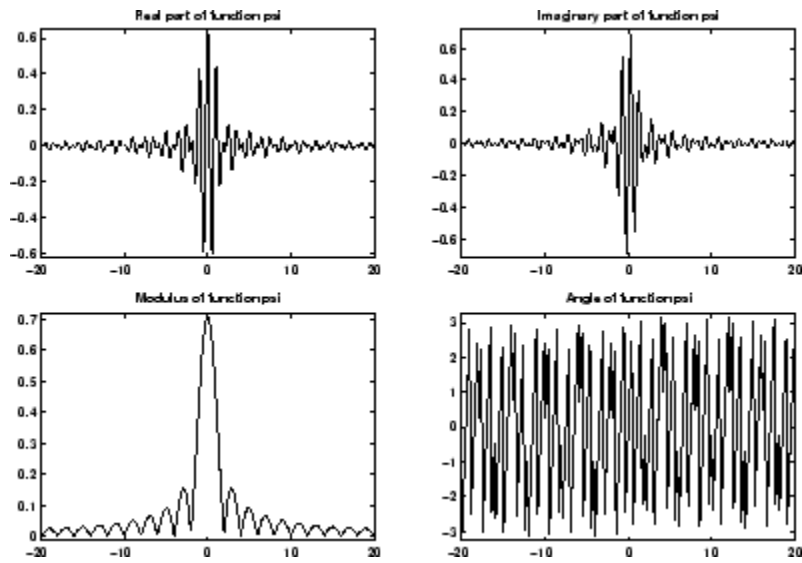
A complex Shannon wavelet is defined by

$$\psi(x) = \sqrt{f_b} \text{sinc}(f_b x) e^{2i\pi f_c x}$$

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.



Complex Shannon Wavelet shan 0.5-1

Wavelet Families and Associated Properties – I

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 φ							■	
Existence of φ			■	■	■	■	■	■
Orthogonal analysis			■	■	■	■	■	
Biorthogonal analysis			■	■	■	■	■	■
Exact reconstruction	≈	■	■	■	■	■	■	■
FIR filters				■	■	■	■	■
Continuous transform	■	■	■	■	■	■	■	■
Discrete transform				■	■	■	■	■
Fast algorithm				■	■	■	■	■
Explicit expression	■	■		■				For splines

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

Regularity — See “General Considerations” section in “Choose a Wavelet”.

Orthogonal — See “General Considerations” section in “Choose a Wavelet”.

Biorthogonal — See “Biorthogonal Wavelet Pairs: biorNr.Nd” on page 1-9.

Vanishing moments — See “General Considerations” section in “Choose a Wavelet”.

Exact reconstruction — See “Reconstruction Filters” in the *Wavelet Toolbox Getting Started Guide*.

Continuous — See “Continuous and Discrete Wavelet Transforms” in the *Wavelet Toolbox Getting Started Guide*.

Discrete — See “Critically-Sampled Discrete Wavelet Transform” in the *Wavelet Toolbox Getting Started Guide*.

Wavelet Families and Associated Properties – II

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 φ							
Existence of φ	■						
Orthogonal analysis							
Biorthogonal analysis	■						
Exact reconstruction	■	■	≈	■	■	■	■
FIR filters	■		■				
Continuous transform	■	■					
Discrete transform	■		■				
Fast algorithm	■		■				
Explicit expression	For splines	■		■	■	■	■
Complex valued				■	■	■	■
Complex continuous transform				■	■	■	■
FIR-based approximation			■				

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

Regularity — See “General Considerations” section in “Choose a Wavelet”.

Orthogonal — See “General Considerations” section in “Choose a Wavelet”.

Biorthogonal — See “Biorthogonal Wavelet Pairs: biorNr.Nd” on page 1-9.

Vanishing moments — See “General Considerations” section in “Choose a Wavelet”.

Exact reconstruction — See “Reconstruction Filters” in the *Wavelet Toolbox Getting Started Guide*.

Continuous — See “Continuous and Discrete Wavelet Transforms” in the *Wavelet Toolbox Getting Started Guide*.

Discrete — See “Continuous and Discrete Wavelet Transforms” in the *Wavelet Toolbox Getting Started Guide*.

FIR filters — See “Filters Used to Calculate the DWT and IDWT” on page 3-43.

Lifting Method for Constructing Wavelets

The so-called *first generation* wavelets and scaling functions are dyadic dilations and translates of a single function. Fourier methods play a key role in the design of these wavelets. However, the requirement that the wavelet basis consist of translates and dilates of a single function imposes some constraints that limit the utility of the multiresolution idea at the core of wavelet analysis.

The utility of wavelet methods is extended by the design of *second generation* wavelets via *lifting*.

Typical settings where translation and dilation of a single function cannot be used include:

- *Designing wavelets on bounded domains* — This includes the construction of wavelets on an interval, or bounded domain in a higher-dimensional Euclidean space.
- *Weighted wavelets* — In certain applications, such as the solution of partial differential equations, wavelets biorthogonal with respect to a weighted inner product are needed.
- *Irregularly-spaced data* — In many real-world applications, the sampling interval between data samples is not equal.

Designing new *first generation* wavelets requires expertise in Fourier analysis. The lifting method proposed by Sweldens (see [Swe98] in “References”) removes the necessity of expertise in Fourier analysis and allows you to generate an infinite number of discrete biorthogonal wavelets starting from an initial one. In addition to generation of *first generation* wavelets with lifting, the lifting method also enables you to design *second generation* wavelets, which cannot be designed using Fourier-based methods. With lifting, you can design wavelets that address the shortcomings of the *first generation* wavelets.

The following section introduces the theory behind lifting, presents the lifting functions of Wavelet Toolbox software and gives two short examples:

- “Lifting Background” on page 1-25
- “Lifting Functions” on page 1-31

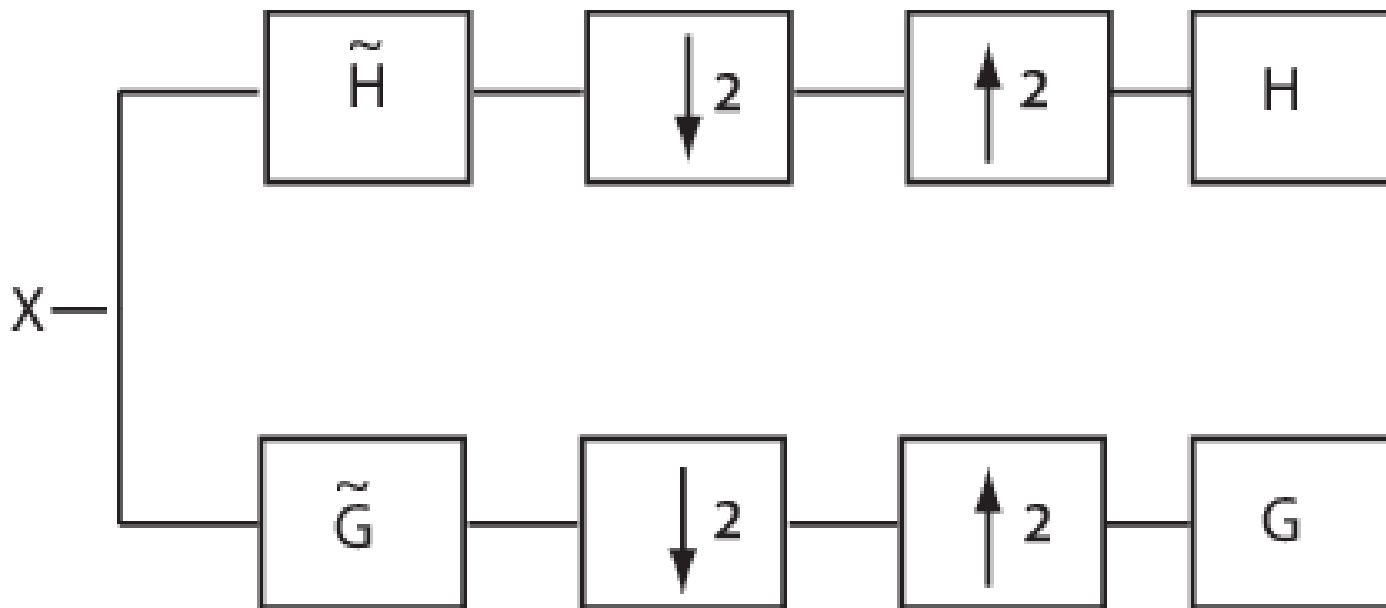
For more information on lifting, see [Swe98], [Mal98], [StrN96], and [MisMOP03] in “References”.

Lifting Background

The DWT implemented by a filter bank is defined by four filters as described in “Fast Wavelet Transform (FWT) Algorithm” on page 3-43. 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 and two synthesis filters. The decomposition and synthesis filters may constitute a pair of biorthogonal bases or an orthogonal basis. The capital letters denote the Z-transforms of the filters..



This leads to the following two conditions for a perfect reconstruction (PR) filter bank:

$$\tilde{H}(z)H(z) + \tilde{G}(z)G(z) = 2z^{-L} + 1$$

and

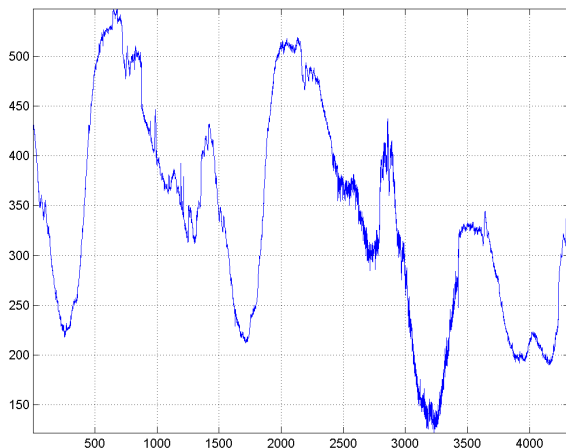
$$\tilde{H}(-z)H(z) + \tilde{G}(-z)G(z) = 0$$

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

The z^{-L+1} term implies that perfect reconstruction is achieved up to a delay of one sample less than the filter length, L . This results if the analysis filters are shifted to be causal.

Lifting designs perfect reconstruction filter banks by beginning from the basic nature of the wavelet transform. Wavelet transforms build sparse representations by exploiting the correlation inherent in most real world data. For example, plot the example of electricity consumption over a 3-day period.

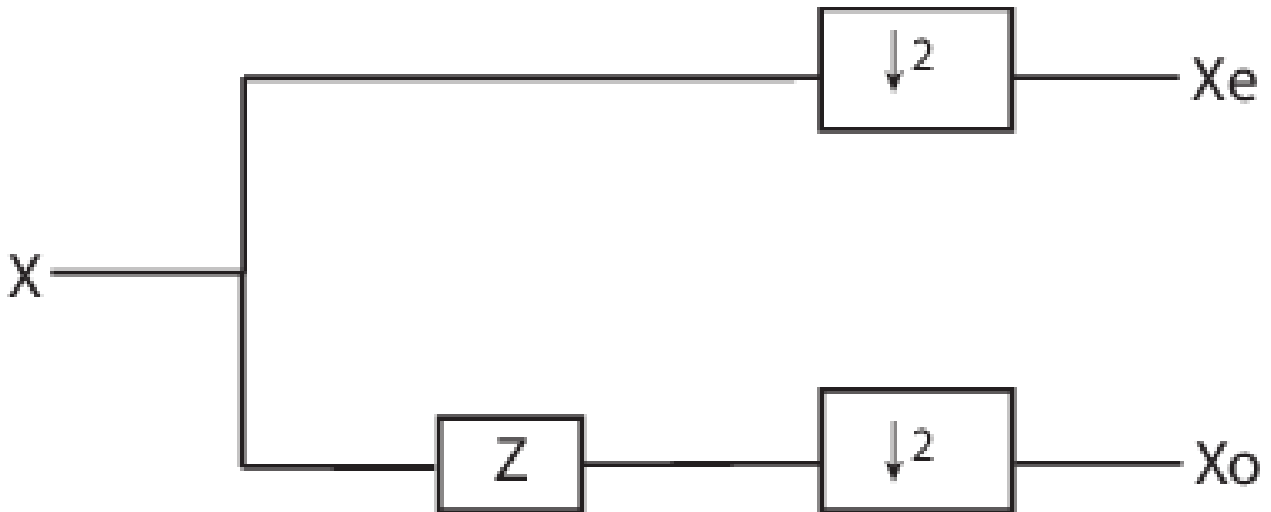
```
load leleccum;  
plot(leleccum)  
grid on; axis tight;
```



The data do not exhibit arbitrary changes from sample to sample. Neighboring samples exhibit correlation. A relatively low (high) value at index (sample) n is associated with a relatively low (high) value at index $n-1$ and $n+1$. This implies that if you have only the odd or even samples from the data, you can predict the even or odd samples. How accurate your prediction is obviously depends on the nature of the correlation between adjacent samples and how closely your predictor approximates that correlation.

Polyphase Representation

The *polyphase* representation of a signal is an important concept in lifting. You can view each signal as consisting of *phases*, which consist of taking every N -th sample beginning with some index. For example, if you index a time series from $n=0$ and take every other sample starting at $n=0$, you extract the even samples. If you take every other sample starting from $n=1$, you extract the odd samples. These are the even and odd polyphase components of the data. Because your increment between samples is 2, there are only two phases. If you increased your increment to 4, you can extract 4 phases. For lifting, it is sufficient to concentrate on the even and odd polyphase components. The following diagram illustrates this operation for an input signal.



where Z denotes the unit advance operator and the downward arrow with the number 2 represents downsampling by two. In the language of lifting, the operation of separating an input signal into even and odd components is known as the *split* operation, or the *lazy wavelet*.

To understand lifting mathematically, it is necessary to understand the z-domain representation of the even and odd polyphase components.

The z-transform of the even polyphase component is

$$X_0(z) = \sum_n x(2n)z^{-n}$$

The z-transform of the odd polyphase component is

$$X_1(z) = \sum_n x(2n + 1)z^{-n}$$

You can write the z-transform of the input signal as the sum of dilated versions of the z-transforms of the polyphase components.

$$X(z) = \sum_n x(2n)z^{-2n} + \sum_n x(2n + 1)z^{-2n + 1} = X_0(z^2) + z^{-1}X_1(z^2)$$

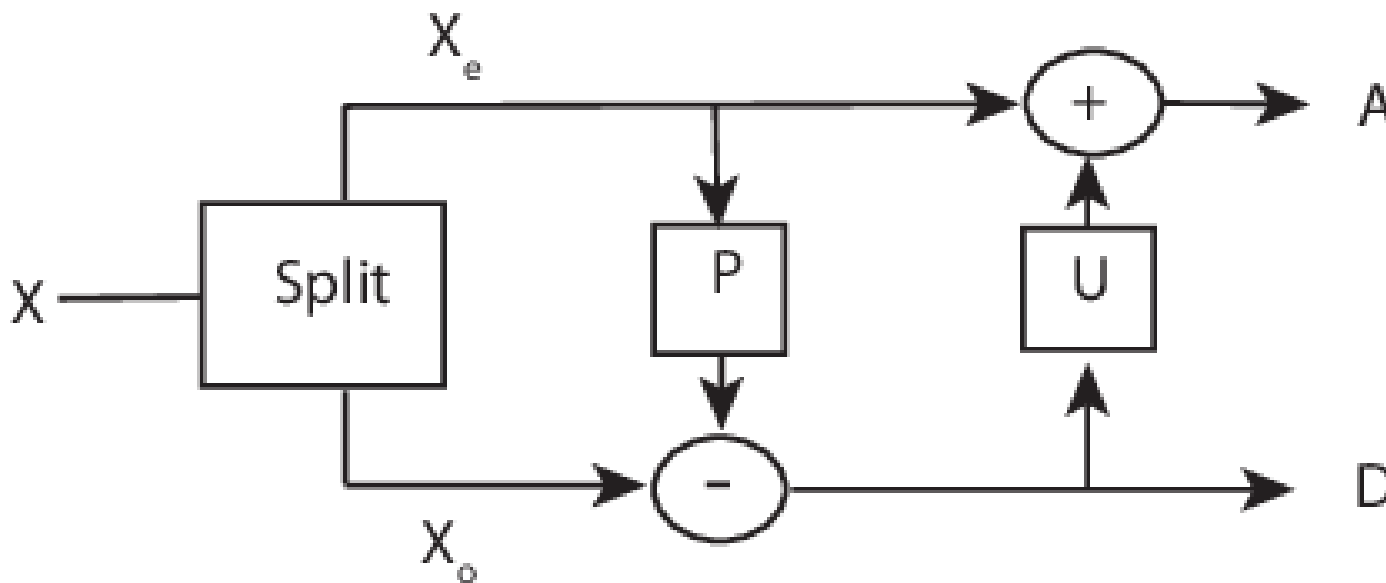
Split, Predict, and Update

A single lifting step can be described by the following three basic operations:

- *Split* — the signal into disjoint components. A common way to do this is to extract the even and odd polyphase components explained in “Polyphase Representation” on page 1-27. This is also known as the *lazy wavelet*.
- *Predict* — the odd polyphase component based on a linear combination of samples of the even polyphase component. The samples of the odd polyphase component are replaced by the difference between the odd polyphase component and the predicted value. The predict operation is also referred to as the *dual lifting step*.
- *Update* — the even polyphase component based on a linear combination of difference samples obtained from the predict step. The update step is also referred to as the *primal lifting step*.

In practice, a normalization is incorporated for both the primal and dual liftings.

The following diagram illustrates one lifting step.



Haar Wavelet Via Lifting

Using the operations defined in “Split, Predict, and Update” on page 1-28, you can implement the Haar wavelet via lifting.

- *Split* — Divide the signal into even and odd polyphase components.
- *Predict* — Replace $x(2n+1)$ with $d(n)=x(2n+1)-x(2n)$. The predict operator is simply $x(2n)$.
- *Update* — Replace $x(2n)$ with $x(2n)+d(n)/2$. This is equal to $(x(2n)+x(2n+1))/2$.

The dual lifting in the z domain can be written in the following matrix form

$$\begin{pmatrix} 1 & 0 \\ -P(z) & 1 \end{pmatrix} \begin{pmatrix} X_0(z) \\ X_1(z) \end{pmatrix}$$

with $P(z)=1$.

The primal lifting can be written in the z domain in the following matrix form

$$\begin{pmatrix} 1 & S(z) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -P(z) & 1 \end{pmatrix} \begin{pmatrix} X_0(z) \\ X_1(z) \end{pmatrix}$$

with $S(z)=1/2$.

Finally, the primal and dual normalization can be incorporated as follows.

$$\begin{pmatrix} \sqrt{2} & 0 \\ 0 & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 & S(z) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -P(z) & 1 \end{pmatrix} \begin{pmatrix} X_0(z) \\ X_1(z) \end{pmatrix}$$

To construct this lifting step in MATLAB, enter:

```
LiftHaar = liftwave('haar');
displs(LiftHaar)
```

The following is displayed in the MATLAB command window.

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

'd' denotes the dual lifting. Note that for convenience, the negative sign is incorporated into the dual lifting step in the Wavelet Toolbox software. 'p' denotes the primal lifting and [1.41421356] [0.70710678] are the primal and dual normalization constants. LiftHaar{1,3} and LiftHaar{2,3} give the highest degree of the Laurent polynomials, which describe the dual and primal liftings. In this case, both are zero because the dual and primal liftings are both described by scalars.

Bior2.2 Wavelet Via Lifting

This example presents the lifting scheme for the bior2.2 biorthogonal scaling and wavelet filters.

In the Haar lifting scheme, the dual lifting (predict operator) differenced the odd and even samples. In this example, define a new predict operator that computes the average of the two neighboring even samples. Subtract the average from the intervening odd sample.

$$d(n) = x(2n + 1) - \frac{1}{2}[x(2n) + x(2n + 2)]$$

In the z-domain you can write the dual lifting step as

$$\begin{pmatrix} 1 & 0 \\ -\frac{1}{2}(1+z) & 1 \end{pmatrix} \begin{pmatrix} X_0(z) \\ X_1(z) \end{pmatrix}$$

To obtain the primal lifting, or update, examine the primal lifting in “Haar Wavelet Via Lifting” on page 1-29. The update is defined in such a way that the sum of the approximation coefficients is proportional to the mean of the input data vector.

$$\sum_n x(n) = \frac{1}{2} \sum_n a(n)$$

To obtain the same result in this lifting step, define the update as

$$\begin{pmatrix} 1 & \frac{1}{4}(z^{-1} + 1) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -\frac{1}{2}(1+z) & 1 \end{pmatrix} \begin{pmatrix} X_0(z) \\ X_1(z) \end{pmatrix}$$

To obtain this lifting scheme at the command line, enter:

```
liftwave('bior2.2')
```

Lifting Functions

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

- “Lifting Schemes” on page 1-32
- “Biorthogonal Quadruplets of Filters and Lifting Schemes” on page 1-32
- “Usual Biorthogonal Quadruplets” on page 1-32
- “Lifting Wavelet Transform (LWT)” on page 1-33
- “Laurent Polynomials and Matrices” on page 1-33

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
wavenames	Wavelets with lifting schemes

Biorthogonal Quadruplets of Filters and Lifting Schemes

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

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

Usual Biorthogonal Quadruplets

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

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

Lifting Wavelet Transform (LWT)

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

Function Name	Description
<code>lwt</code>	1-D lifting wavelet transform
<code>ilwt</code>	Inverse 1-D lifting wavelet transform
<code>lwtcoef</code>	Extract or reconstruct 1-D LWT wavelet coefficients
<code>lwt2</code>	2-D lifting wavelet transform
<code>ilwt2</code>	Inverse 2-D lifting wavelet transform
<code>lwtcoef2</code>	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
<code>laurpoly</code>	Constructor for the class of Laurent polynomials
<code>laurmat</code>	Constructor for the class of Laurent matrices

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

Primal Lifting from Haar

These two simple examples illustrate the basic lifting capabilities of Wavelet Toolbox software.

A primal lifting starting from Haar wavelet.

Start from the Haar wavelet and get the corresponding lifting scheme.

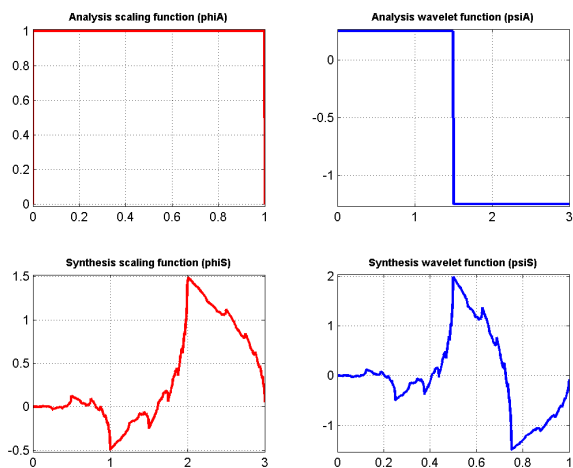
```
lshaar = liftwave('haar');
displs(lshaar);
```

Add a primal ELS to the lifting scheme.

```
els = {'p',[-0.125 0.125],0};  
lsnew = addlift(lshaar,els);  
displs(lsnew);
```

Transform the lifting scheme to biorthogonal filters quadruplet and plot the resulting scaling function and wavelet.

```
[LoD,HiD,LoR,HiR] = ls2filt(lsnew);  
bswfun(LoD,HiD,LoR,HiR,'plot');
```



Integer-to-Integer Wavelet Transform

In several applications it is desirable to have a wavelet transform that maps integer inputs to integer scaling and wavelet coefficients. You can accomplish easily using lifting.

Start with the Haar transform for an integer to integer wavelet transform and apply a primal lifting step.

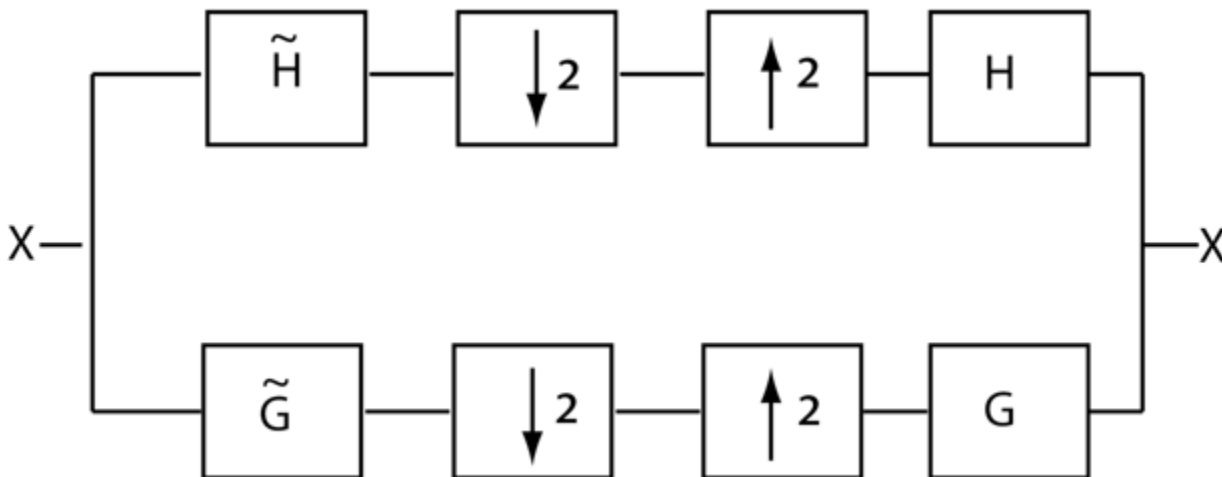
```
lshaar = liftwave('haar','int2int');  
els = {'p',[-0.125 0.125],0};  
lsnewint = addlift(lshaar,els);
```

Obtain the integer-to-integer wavelet transform of a 1-D signal and invert the transform to demonstrate perfect reconstruction.

```
x = 1:8;  
[cA,cD] = lwt(x,lsnewint);  
xnew = ilwt(cA,cD,lsnewint)
```

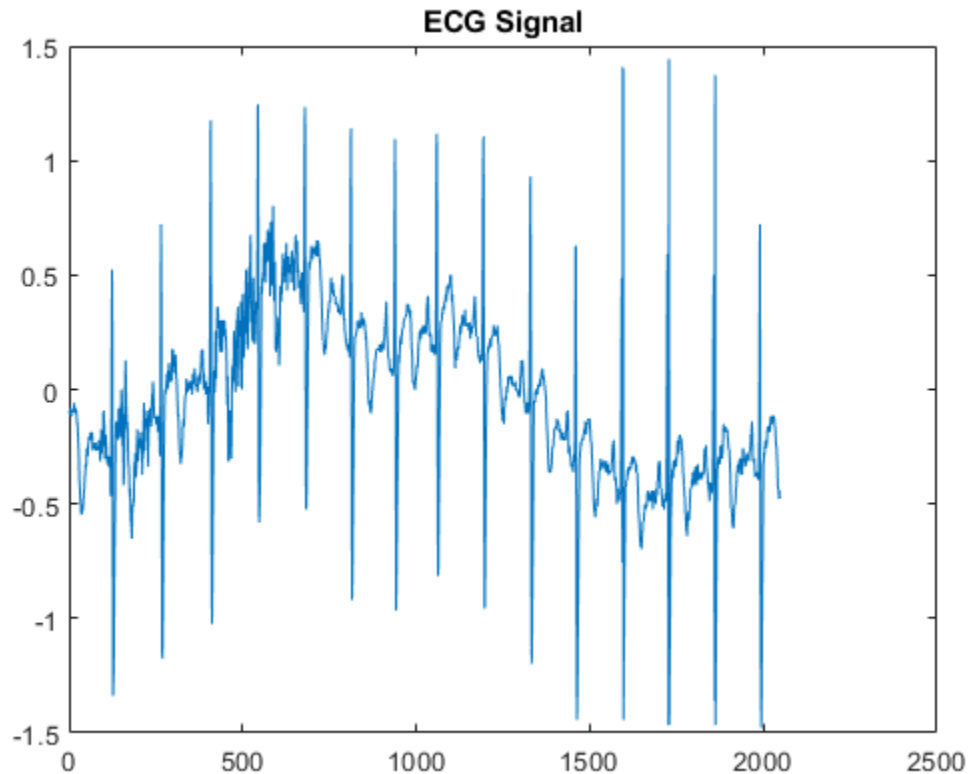
Orthogonal and Biorthogonal Filter Banks

This example shows to construct and use orthogonal and biorthogonal filter banks with the Wavelet Toolbox software. The classic critically sampled two-channel filter bank is shown in the following figure.



Let \tilde{G} and \tilde{H} denote the lowpass and highpass analysis filters and G and H denote the corresponding lowpass and highpass synthesis filters. A two-channel critically sampled filter bank filters the input signal using a lowpass and highpass filter. The subband outputs of the filters are downsampled by two to preserve the overall number of samples. To reconstruct the input, upsample by two and then interpolate the results using the lowpass and highpass synthesis filters. If the filters satisfy certain properties, you can achieve perfect reconstruction of the input. To demonstrate this, filter an ECG signal using Daubechies's extremal phase wavelet with two vanishing moments. The example explains the notion of vanishing moments in a later section.

```
load wecg;
plot(wecg);
title('ECG Signal')
```



Obtain the lowpass (scaling) and highpass (wavelet) analysis and synthesis filters.

```
[gtilde,htilde,g,h] = wfilters('db2');
```

For this example, set the padding mode for the DWT to periodization. This does not extend the signal.

```
origmodestatus = dwtmode('status','nodisplay');  
dwtmode('per','nodisplay');
```

Obtain the level-one discrete wavelet transform (DWT) of the ECG signal. This is equivalent to the analysis branch (with downsampling) of the two-channel filter bank in the figure.

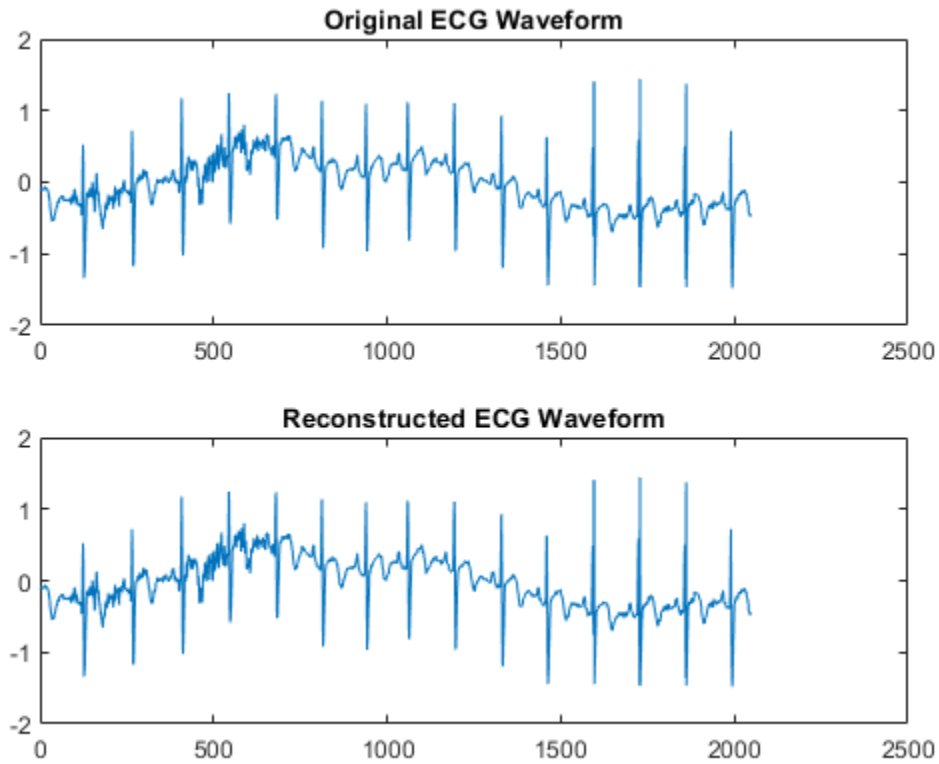
```
[lowpass,highpass] = dwt(wecg,gtilde,htilde);
```

Upsample and interpolate the lowpass (scaling coefficients) and highpass (wavelet coefficients) subbands with the synthesis filters and demonstrate perfect reconstruction.

```
xrec = idwt(lowpass,highpass,g,h);  
max(abs(wecg-xrec))  
subplot(2,1,1)  
plot(wecg); title('Original ECG Waveform')  
subplot(2,1,2)  
plot(xrec); title('Reconstructed ECG Waveform');
```

```
ans =
```

```
1.3658e-12
```

The analysis and synthesis filters for the 'db2' wavelet are just time reverses of each other. You can see this by comparing the following.

```
scalingFilters = [flip(gtilde); g]
waveletFilters = [flip(htilde); h]
```

```
scalingFilters =
```

```
    0.4830    0.8365    0.2241   -0.1294
    0.4830    0.8365    0.2241   -0.1294
```

```
waveletFilters =
```

```
-0.1294    -0.2241     0.8365    -0.4830  
-0.1294    -0.2241     0.8365    -0.4830
```

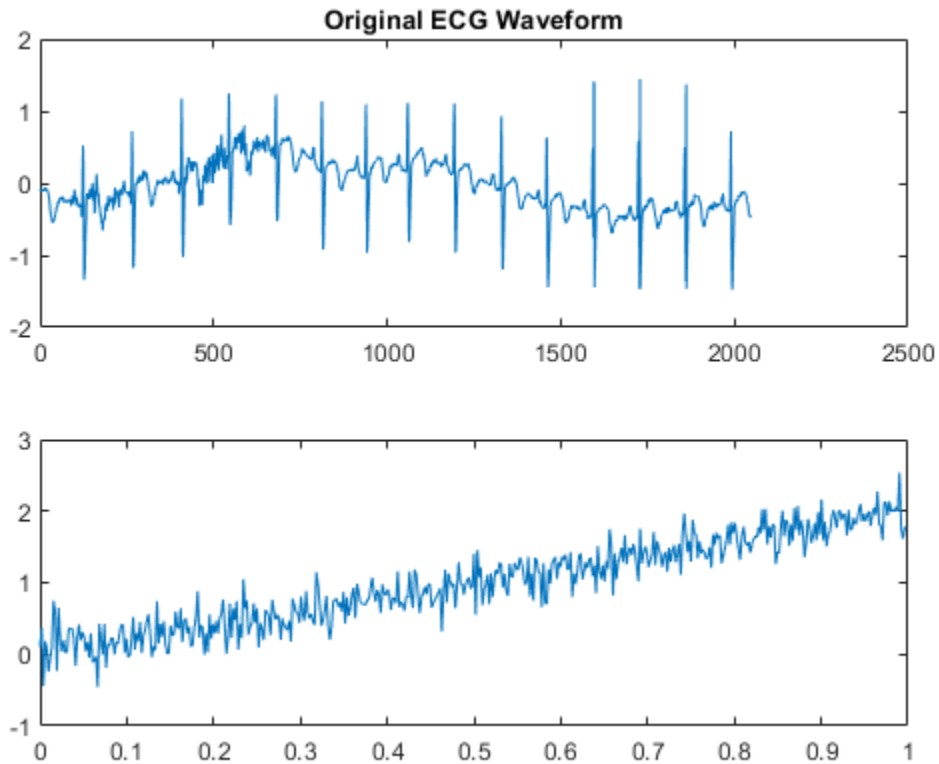
This is the case with all orthogonal wavelet filter banks. The orthogonal wavelet families supported by the Wavelet Toolbox are 'dbN', 'fkN', 'symN', and 'coifN' where N is a valid filter number.

Instead of providing `dwt` with the filters in the previous example, you the string 'db2' instead. Using the wavelet family short name and filter number, you do not have to correctly specify the analysis and synthesis filters.

```
[lowpass,highpass] = dwt(wecg, 'db2');  
xrec = idwt(lowpass,highpass, 'db2');
```

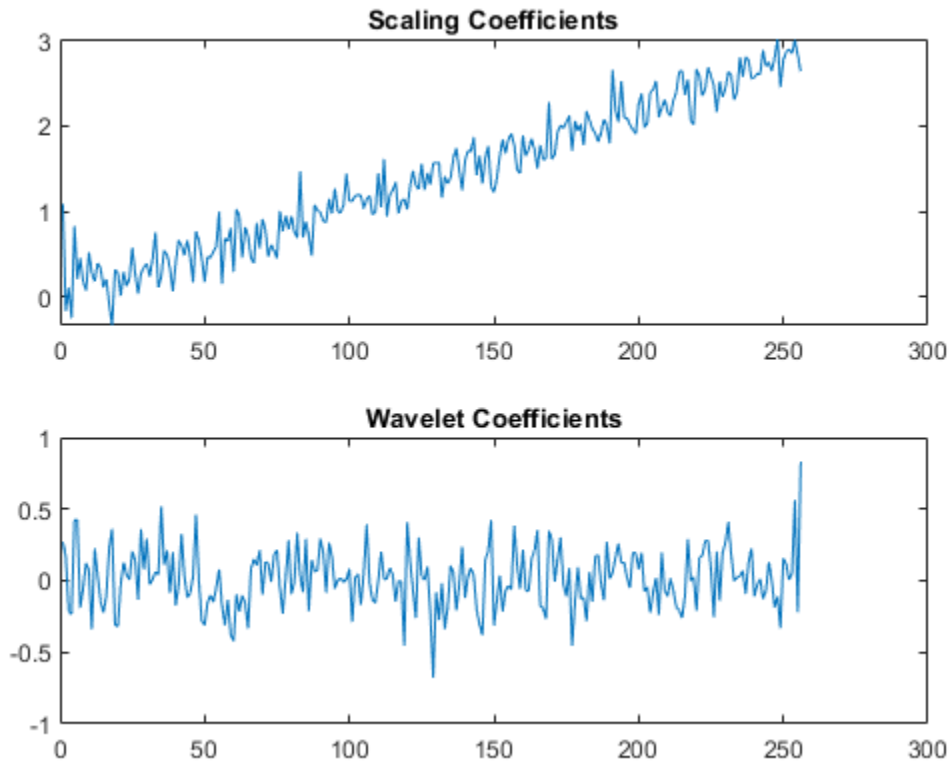
The filter number in the Daubechies's extremal phase and least asymmetric phase wavelets ('db' and 'sym') refers to the number of vanishing moments. Basically, a wavelet with N vanishing moments removes a polynomial of order N-1 in the wavelet coefficients. To illustrate this, construct a signal which consists of a linear trend with additive noise.

```
n = (0:511)/512;  
x = 2*n+0.2*randn(size(n));  
plot(n,x)
```

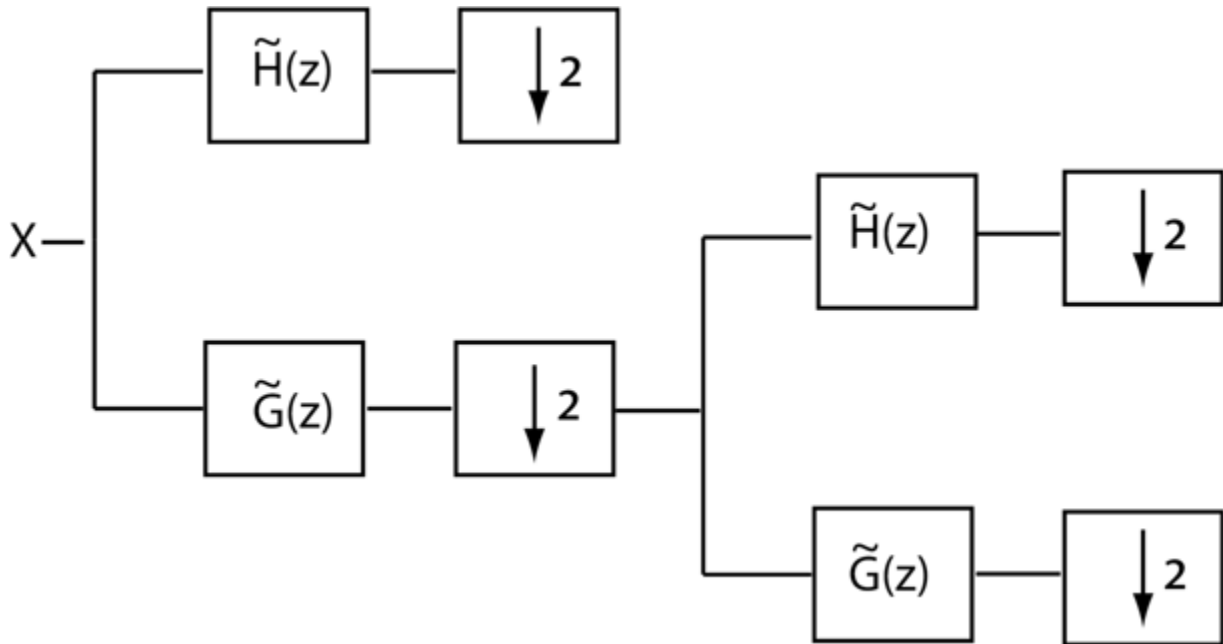


A linear trend is a polynomial of degree 1. Therefore, a wavelet with two vanishing moments removes this polynomial. The linear trend is preserved in the scaling coefficients and the wavelet coefficients can be regarded as consisting of only noise. Obtain the level-one DWT of the signal with the 'db2' wavelet (two vanishing moments) and plot the coefficients.

```
[A,D] = dwt(x,'db2');  
subplot(2,1,1)  
plot(A); title('Scaling Coefficients');  
subplot(2,1,2)  
plot(D); title('Wavelet Coefficients');
```



You can use `dwt` and `idwt` to implement a two-channel orthogonal filter bank, but it is often more convenient to implement a multi-level two-channel filter bank using `wavedec`. The multi-level DWT iterates on the output of the lowpass (scaling) filter. In other words, the input to the second level of the filter bank is the output of the lowpass filter at level 1. A two-level wavelet filter bank is illustrated in the following figure.



At each successive level, the number of scaling and wavelet coefficients is downsampled by two so the total number of coefficients are preserved. Obtain the level three DWT of the ECG signal using the 'sym4' orthogonal filter bank.

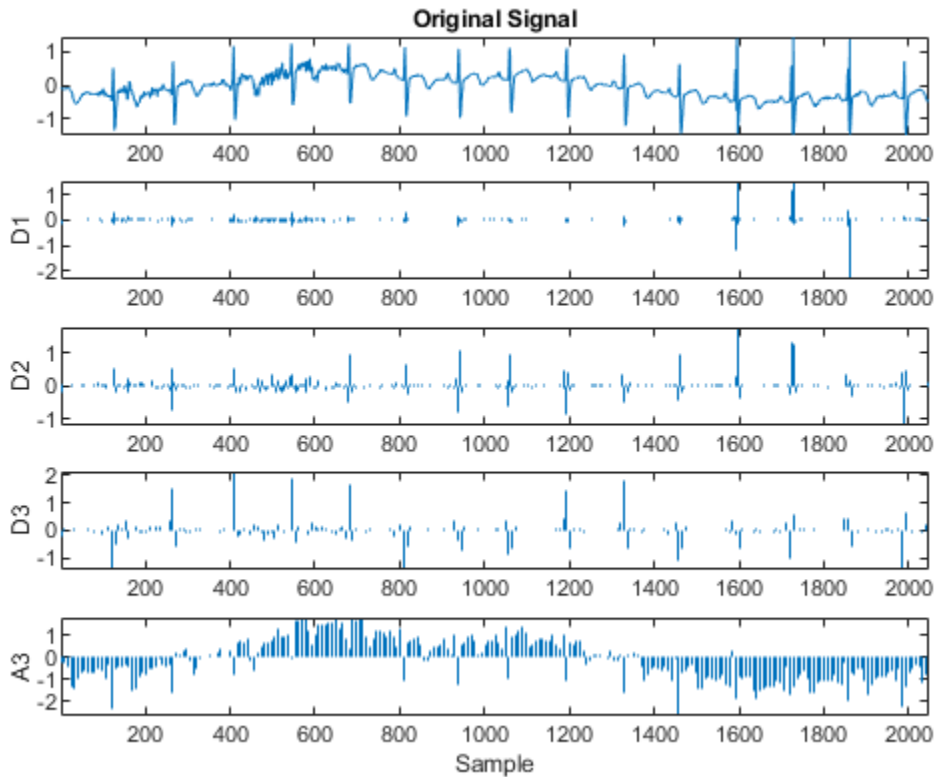
```
[C,L] = wavedec(wecg,3,'sym4');
```

The number of coefficients by level is contained in the vector, L. The first element of L is equal to 256, which represents the number of scaling coefficients at level 3 (the final level). The second element of L is the number of wavelet coefficients at level 3. Subsequent elements give the number of wavelet coefficients at higher levels until you reach the final element of L. The final element of L is equal to the number of samples in the original signal. The scaling and wavelet coefficients are stored in the vector C in the same order. To extract the scaling or wavelet coefficients, use `appcoef` or `detcoef`. Extract all the wavelet coefficients in a cell array and final-level scaling coefficients.

```
wavcoefs = detcoef(C,L,'dcells');
a3 = appcoef(C,L,'sym4');
```

You can plot the wavelet and scaling coefficients at their approximate positions.

```
cfsmatrix = zeros(numel(wecg),4);
cfsmatrix(1:2:end,1) = wavcoefs{1};
cfsmatrix(1:4:end,2) = wavcoefs{2};
cfsmatrix(1:8:end,3) = wavcoefs{3};
cfsmatrix(1:8:end,4) = a3;
subplot(5,1,1)
plot(wecg); title('Original Signal');
axis tight;
for kk = 2:4
    subplot(5,1,kk)
    stem(cfsmatrix(:,kk-1),'marker','none','ShowBaseLine','off');
    ylabel(['D' num2str(kk-1)]);
    axis tight;
end
subplot(5,1,5);
stem(cfsmatrix(:,end),'marker','none','ShowBaseLine','off');
ylabel('A3'); xlabel('Sample');
axis tight;
```



Because the critically sampled wavelet filter bank downsamples the data at each level, the analysis must stop when you have only one coefficient left. In the case of the ECG signal with 2048 samples, this must occur when $L = \log_2 2048$.

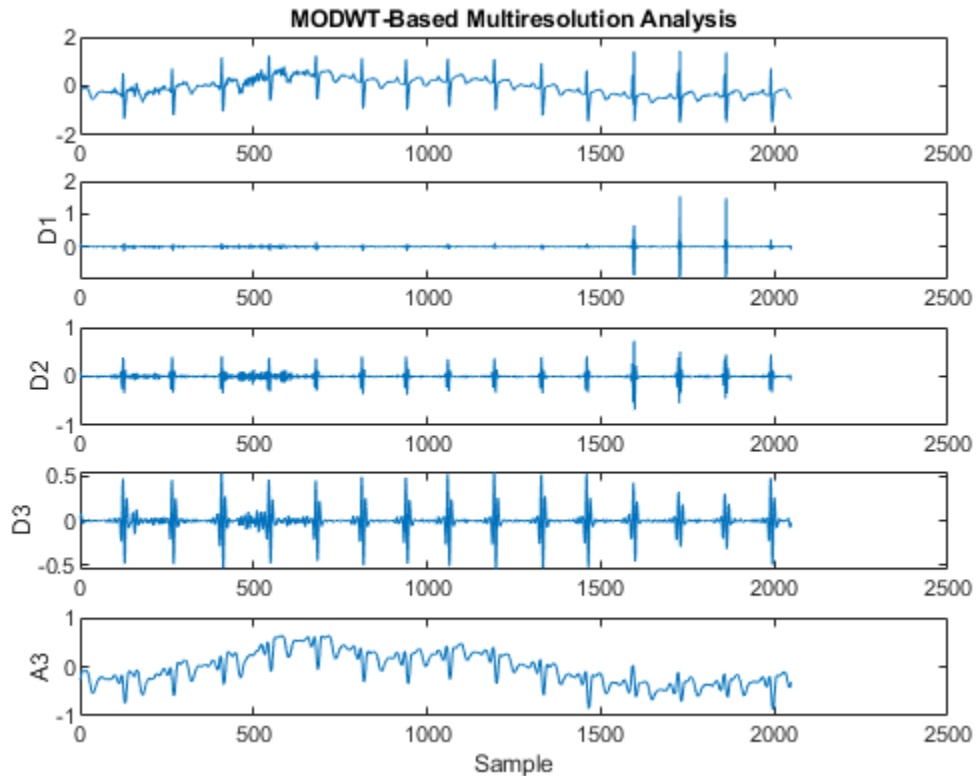
```
[C,L] = wavedec(wecg,log2(numel(wecg)),'sym4');
fprintf('The number of coefficients at the final level is %d. \n',L(1));
```

The number of coefficients at the final level is 1.

If you wish to implement an orthogonal wavelet filter bank without downsampling, you can use `modwt`.

```
ecgmodwt = modwt(wecg,'sym4',3);
ecgmra = modwtmra(ecgmodwt,'sym4');
```

```
subplot(5,1,1);  
plot(wecg); title('Original Signal');  
  
title('MODWT-Based Multiresolution Analysis');  
for kk = 2:4  
    subplot(5,1,kk)  
    plot(ecgmra(kk-1,:));  
    ylabel(['D' num2str(kk-1)]);  
end  
subplot(5,1,5);  
plot(ecgmra(end,:));  
ylabel('A3'); xlabel('Sample');
```



In a biorthogonal filter bank, the synthesis filters are not simply time-reversed versions of the analysis filters. The family of biorthogonal spline wavelet filters are an example of such filter banks.

```
[LoD,HiD,LoR,HiR] = wfilters('bior3.5');
```

If you examine the analysis filters (LoD,HiD) and the synthesis filters (LoR,HiR), you see that they are very different. These filter banks still provide perfect reconstruction.

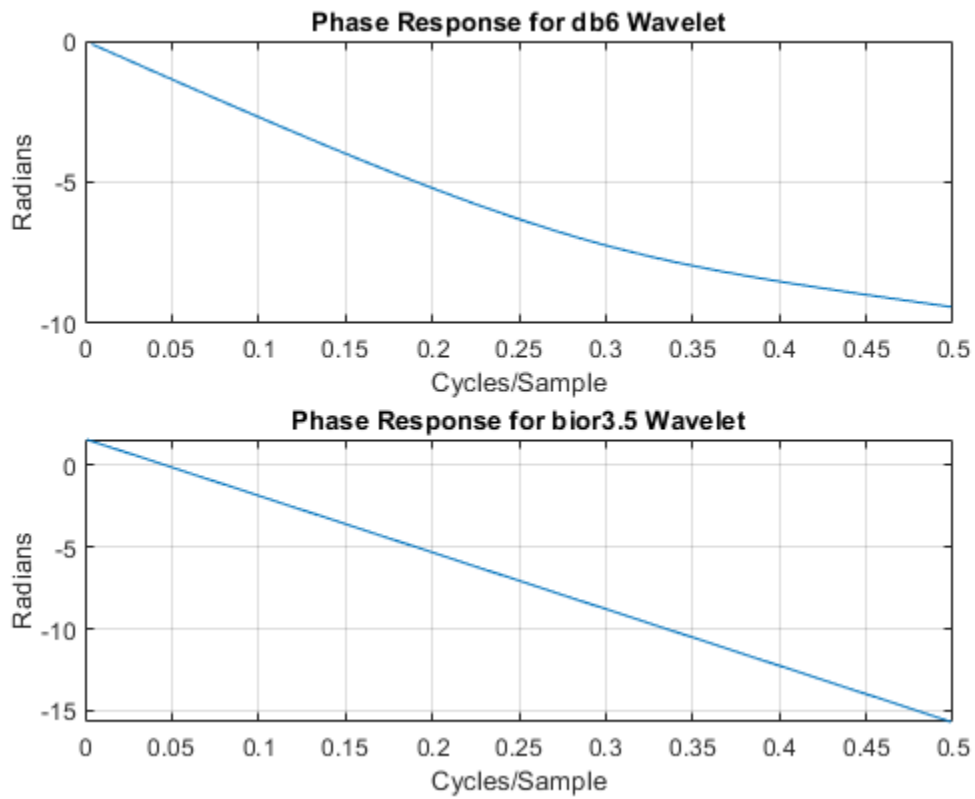
```
[A,D] = dwt(wecg,LoD,HiD);
xrec = idwt(A,D,LoR,HiR);
max(abs(wecg-xrec))
```

```
ans =
```

```
6.6613e-16
```

Biorthogonal filters are useful when linear phase is a requirement for your filter bank. Orthogonal filters cannot have linear phase with the exception of the Haar wavelet filter. If you have the Signal Processing Toolbox software, you can look at the phase responses for an orthogonal and biorthogonal pair of wavelet filters.

```
[Lodb6,Hidb6] = wfilters('db6');
[PHIdb6,W] = phasez(Hidb6,1,512);
PHIbior35 = phasez(HiD,1,512);
figure;
subplot(2,1,1)
plot(W./(2*pi),PHIdb6); title('Phase Response for db6 Wavelet');
grid on;
xlabel('Cycles/Sample'); ylabel('Radians');
subplot(2,1,2)
plot(W./(2*pi),PHIbior35); title('Phase Response for bior3.5 Wavelet');
grid on;
xlabel('Cycles/Sample'); ylabel('Radians');
```



Set the `dwtmode` back to the original setting.

```
dwtmode(origmodestatus, 'nodisplay');
```

Scaling Function and Wavelet

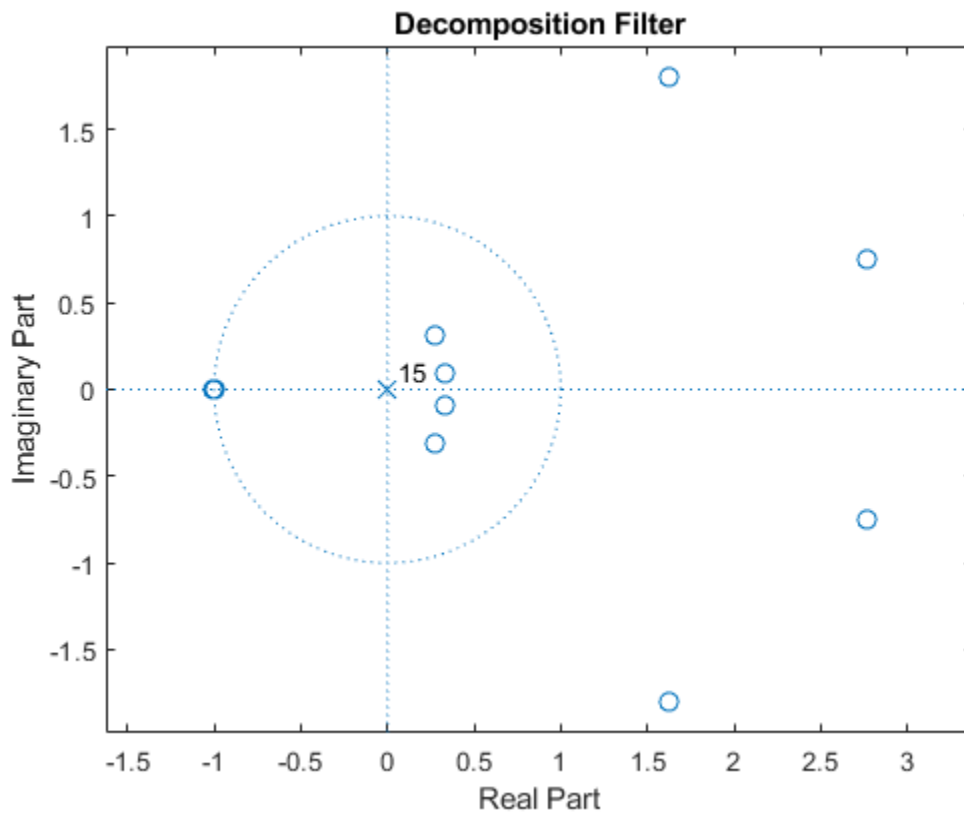
This example uses `wavefun` to demonstrate how the number of vanishing moments in a biorthogonal filter pair affects the smoothness of the corresponding dual scaling function and wavelet. While this example uses `wavefun` for a biorthogonal wavelet, 'bior3.7', you can also use `wavefun` to obtain orthogonal scaling and wavelet functions.

First, obtain the scaling and wavelet filters and look at the number of vanishing moments in the wavelets. This is equivalent to looking at the number of zeros at $-1+i0$ in the dual filter.

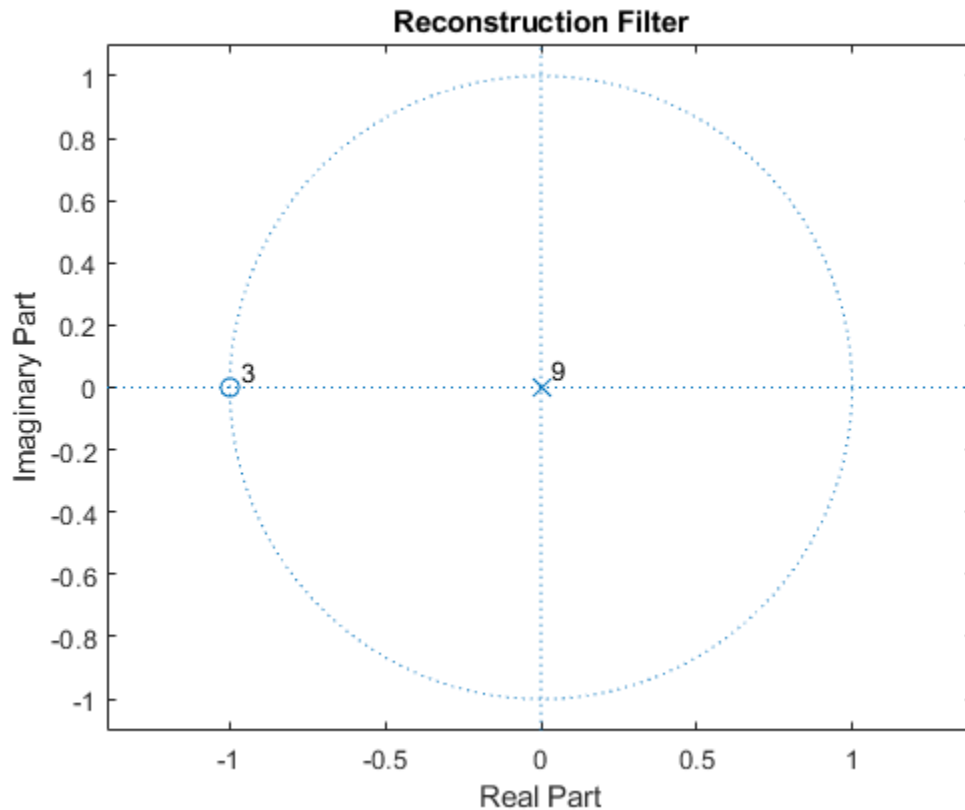
```
[LoD,HiD,LoR,HiR] = wfilters('bior3.7');
```

If you have the Signal Processing Toolbox™, you can use `zplane` to look at the number of zeros at $-1+i0$ for both the decomposition and reconstruction filters.

```
zplane(LoD); title('Decomposition Filter');
```



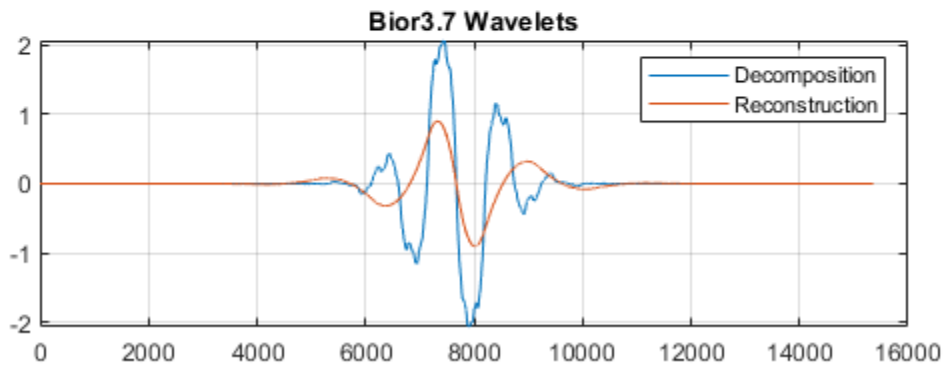
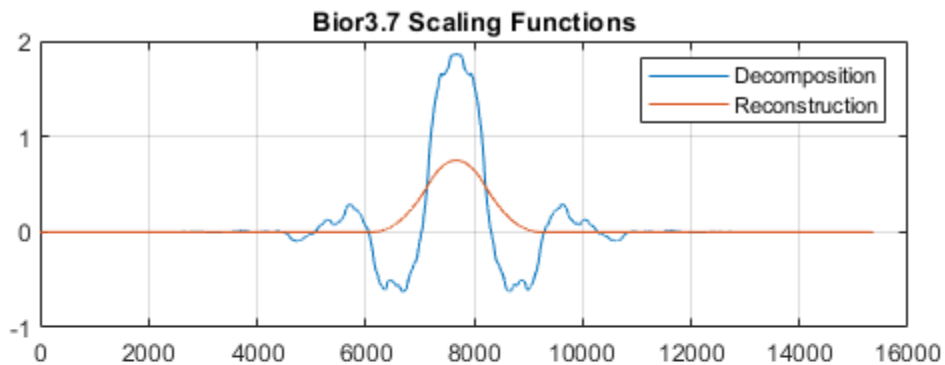
```
figure;  
zplane(LoR); title('Reconstruction Filter');
```



If you zoom in on the region around $-1+i0$, you find there are 7 zeros in the decomposition filter and 3 zeros in the reconstruction filter. This has important consequences for the smoothness of the corresponding scaling functions and wavelets. For biorthogonal wavelets, the more zeros at $-1+i0$ in the lowpass filter, the smoother the **opposite** scaling function and wavelet is. In other words, more zeros in the decomposition filter implies a smoother reconstruction scaling function and wavelet. Conversely, more zeros in the reconstruction filter implies a smoother decomposition scaling function and wavelet.

Use `wavefun` to confirm this. For orthogonal and biorthogonal wavelets, `wavefun` works by reversing the Mallat algorithm. Specifically, the algorithm starts with a single wavelet or scaling coefficient at the coarsest resolution level and reconstructs the wavelet or scaling function to the specified finest resolution level. Generally, 8 to 10 levels is sufficient to get an accurate representation of the scaling function and wavelet.

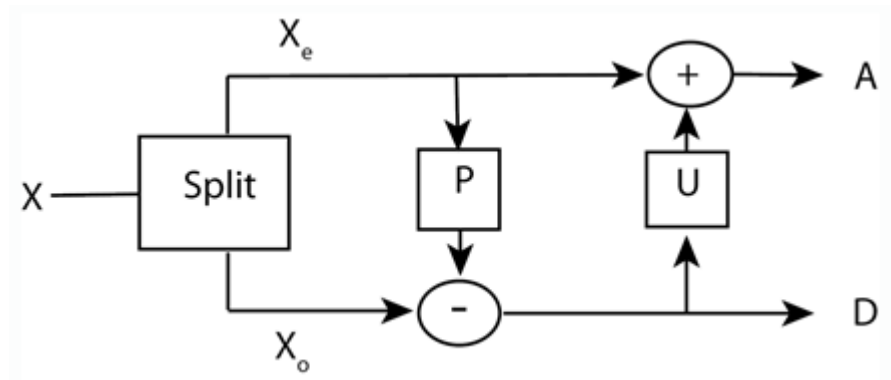
```
[phiD,psiD,phiR,psiR] = wavefun('bior3.7',10);  
subplot(2,1,1)  
plot([phiD' phiR']); grid on;  
title('Bior3.7 Scaling Functions');  
legend('Decomposition','Reconstruction');  
subplot(2,1,2)  
plot([psiD' psiR']); grid on;  
title('Bior3.7 Wavelets');  
legend('Decomposition','Reconstruction');
```



Because there are more than twice the number of zeros at $-1+i0$ for the lowpass decomposition filter, the dual (reconstruction) scaling function and wavelet are much smoother than the analysis (decomposition) scaling function and wavelet.

Lifting a Filter Bank

This example shows how to use lifting to progressively change the properties of a perfect reconstruction filter bank. The following figure shows the three canonical steps in lifting: split, predict, and update.



The first step in lifting is simply to split the signal into its even- and odd-indexed samples. These are called polyphase components and that step in the lifting process is often referred to as the "lazy" lifting step because you really are not doing that much work. You can do this in MATLAB by creating a "lazy" lifting scheme.

```
LS = liftwave('lazy');
```

Apply the lifting scheme to some data.

```
x = randn(8,1);
[ALazy,DLazy] = lwt(x,LS);
```

MATLAB™ indexes from 1 so ALazy contains the odd-indexed samples of x and DLazy contains the even-indexed samples. Most explanations of lifting assume that the signal starts with sample 0, so ALazy would be the even-indexed samples and DLazy the odd-indexed samples. This example follows that latter convention. The "lazy" wavelet transform treats one half of the signal as wavelet coefficients, DLazy, and the other half as scaling coefficients, ALazy. This is perfectly consistent within the context of lifting, but a simple split of the data does really sparsify or capture any relevant detail.

The next step in the lifting scheme is to predict the odd samples based on the even samples. The theoretical basis for this is that most natural signals and images exhibit

correlation among neighboring samples. Accordingly, you can "predict" the odd-indexed samples using the even-indexed samples. The difference between your prediction and the actual value is the "detail" in the data missed by the predictor. That missing detail comprises the wavelet coefficients.

The prediction step is also referred to as a "dual lifting step". In equation form, you can write the prediction step as $d_j(n) = d_{j-1}(n) - P(a_{j-1})$ where $d_{j-1}(n)$ are the wavelet coefficients at the finer scale and a_{j-1} is some number of finer-scale scaling coefficients. $P(\cdot)$ is the prediction operator.

Add a simple (Haar) dual lifting step that subtracts the even (approximation) coefficient from the odd (detail) coefficient. In this case the prediction operator is simply $(-1)a_{j-1}(n)$. In other words, it predicts the odd samples based on the immediately preceding even sample.

```
ElemLiftStep = {'d', -1, 0};
```

The above code says "create an elementary dual (predict) lifting step using a polynomial in z with the highest power z^0 . The coefficient is -1. Update the lazy lifting scheme.

```
LSN = addlift(LS, ElemLiftStep, 'end');
```

Apply the new lifting scheme to the signal.

```
[A, D] = lwt(x, LSN);
```

Note that the elements of A are identical to those in ALazy. This is expected because you did not modify the approximation coefficients. If you look at the elements of D, you see that they are equal to

```
Dnew = DLazy - ALazy;
```

Compare Dnew to D. Imagine an example where the signal was piecewise constant over every two samples.

```
v = [1 -1 1 -1 1 -1];  
u = repelem(v, 2);
```

Apply the new lifting scheme to u.

```
[Au, Du] = lwt(u, LSN);
```


You see that all the D_u are zero. This signal has been compressed because all the information is now contained in 6 samples instead of 12 samples. You can easily reconstruct the original signal

```
urecon = ilwt(Au,Du,LSN);
```

In your prediction (dual lifting) step, you predicted that the adjacent odd sample in your signal had the same value as the immediately preceding even sample. Obviously, this is true only for trivial signals. The wavelet coefficients capture the difference between the prediction and the actual values (at the odd samples). Finally, use the update step to update the even samples based on differences obtained in the prediction step. In this case, update using the following $a_j(n) = a_{j-1}(n) + d_{j-1}(n)/2$. This replaces each even-indexed coefficient by the arithmetic average of the even and odd coefficients. An update step is also referred to as a primal lifting step.

```
elsprimal = {'p',1/2,0};
LSupdated = addlift(LSN,elsprimal,'end');
```

Obtain the wavelet transform of the signal with the updated lifting scheme.

```
[A,D] = lwt(x,LSupdated);
```

If you compare A to the original signal, x , you see that the signal mean is captured in the approximation coefficients.

```
mean(A)
ans = -0.0131
mean(x)
ans = -0.0131
```

In fact, the elements of A are easily obtainable from x by the following.

```
n = 1;
for ii = 1:2:numel(x)
    meanz(n) = mean([x(ii) x(ii+1)]);
    n = n+1;
end
```

Compare `meanz` and A . As always, you can invert the lifting scheme to obtain a perfect reconstruction of the data.

```
xrec = ilwt(A,D,LSupdated);  
max(abs(x-xrec))  
  
ans = 2.2204e-16
```

It is common to add a normalization step at the end so that the energy in the signal (ℓ^2 norm) is preserved as the sum of the energies in the scaling and wavelet coefficients. Without this normalization step, the energy is not preserved.

```
norm(x,2)^2  
  
ans = 11.6150  
  
norm(A,2)^2+norm(D,2)^2  
  
ans = 16.8091
```

Add the necessary normalization step.

```
LSscaled = LSupdated;  
LSscaled(end,1:2) = {sqrt(2), sqrt(2)/2};  
[A,D] = lwt(x,LSscaled);  
norm(A,2)^2+norm(D,2)^2  
  
ans = 11.6150
```

Now the ℓ^2 norm of the signal is equal to the sum of the energies in the scaling and wavelet coefficients. The lifting scheme you developed in this example is the Haar lifting scheme.

The Wavelet Toolbox™ supports many commonly used lifting schemes through `liftwave` with pre-defined dual, primal, and normalization steps. For example, you can obtain the Haar lifting scheme with the following.

```
lshaar = liftwave('haar');
```

If you compare `lshaar` to `LSupdated`, you see that our step-by-step lifting scheme matches the Haar lifting scheme. To see that not all lifting schemes consist of single dual and primal lifting steps, examine the lifting scheme that corresponds to the 'bior3.1' wavelet.

```
lsbior3_1 = liftwave('bior3.1')  
  
lsbior3_1=4x3 cell  
    {'d'      }    {[ 0.3333]}    {[      1]}
```

```

{'p'      }      {1x2 double}      {[      0]}
{'d'      }      {[ -0.4444]}      {[      0]}
{[0.4714]}      {[  2.1213]}      {0x0 double}

```

You can also use `liftfilt` if you want to start with a set of biorthogonal (or orthogonal) scaling and wavelet filters and "lift" them to another set. For example, start with the Haar scaling and lifting filters.

```
[LoD,HiD,LoR,HiR] = wfilters('haar');
```

Lift the Haar filters with two primal lifting steps.

```

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);

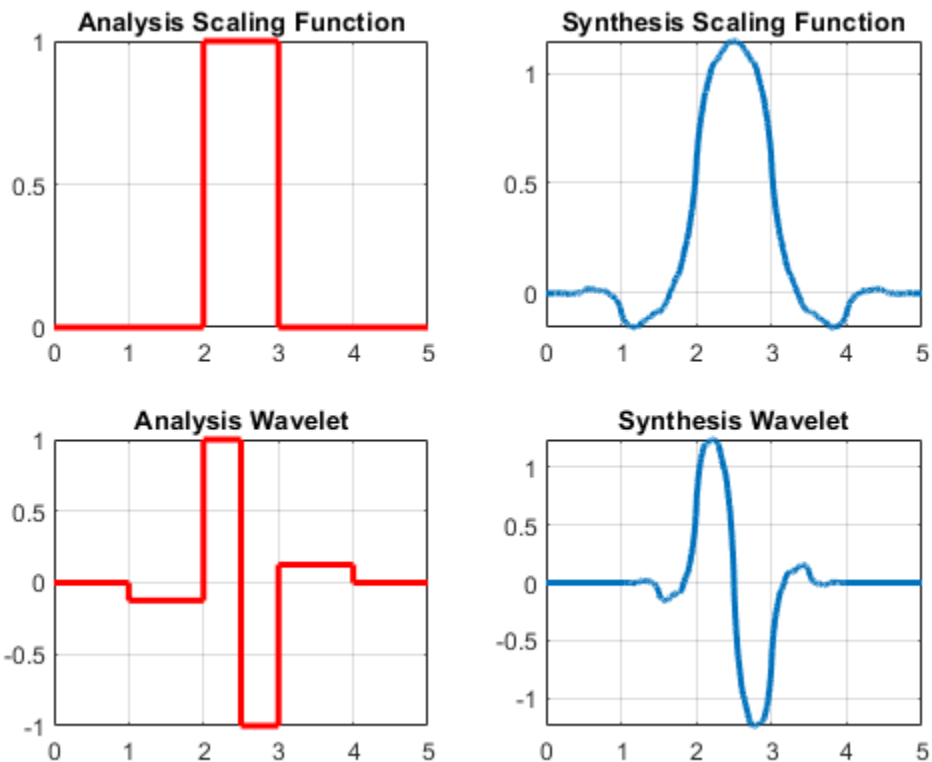
```

Plot the resulting scaling and wavelet functions.

```

[phia,psia,phis,psis,xval] = bswfun(LoDN,HiDN,LoRN,HiRN);
subplot(2,2,1)
plot(xval,phia,'r','linewidth',2);
title('Analysis Scaling Function');
axis tight;
grid on;
subplot(2,2,2)
plot(xval,phis,'linewidth',2);
axis tight;
grid on;
title('Synthesis Scaling Function');
subplot(2,2,3);
plot(xval,psia,'r','linewidth',2);
axis tight;
grid on;
title('Analysis Wavelet');
subplot(2,2,4);
plot(xval,psis,'linewidth',2);
axis tight;
grid on;
title('Synthesis Wavelet');

```



If you plot the analysis and synthesis scaling functions and wavelets for the 'bior1.3' wavelet, you see that lifting the Haar wavelet as in the previous example has essentially provided the 'bior1.3' wavelet to within a change of sign on the synthesis wavelet.

```
[LoD,HiD,LoR,HiR] = wfilters('bior1.3');  
[phia,psia,phis,psis,xval] = bswfun(LoD,HiD,LoR,HiR);
```

Add Quadrature Mirror and Biorthogonal Wavelet Filters

This example shows how to add an orthogonal quadrature mirror filter (QMF) pair and biorthogonal wavelet filter quadruple to Wavelet Toolbox™. While Wavelet Toolbox™ already contains many of the most widely used orthogonal and biorthogonal wavelet families, including the Daubechies' extremal-phase, the Daubechies' least-asymmetric phase, the coiflet, the Fejer-Korovkin filters, and biorthogonal spline wavelets, you can easily add your own filters and use the filter in any of the discrete wavelet or wavelet packet algorithms.

This example adds the Beylkin(18) QMF filter pair to the toolbox and shows how to subsequently use the filter in discrete wavelet analysis. The example then demonstrates how to verify the necessary and sufficient conditions for the QMF pair to constitute a scaling and wavelet filter. After the adding the QMF pair, the example adds the nearly-orthogonal biorthogonal wavelet quadruple based on the Laplacian pyramid scheme of Burt and Adelson (Table 8.4 on page 283 in [1]).

Adding a QMF

First, you must have some way of obtaining the coefficients. In this case, here are the coefficients for the lowpass (scaling) Beylkin(18) filter. You only need a valid scaling filter, `wfilters` creates the corresponding wavelet filter for you.

```
beyl = [9.93057653743539270E-02
4.24215360812961410E-01
6.99825214056600590E-01
4.49718251149468670E-01
-1.10927598348234300E-01
-2.64497231446384820E-01
2.69003088036903200E-02
1.55538731877093800E-01
-1.75207462665296490E-02
-8.85436306229248350E-02
1.96798660443221200E-02
4.29163872741922730E-02
-1.74604086960288290E-02
-1.43658079688526110E-02
1.00404118446319900E-02
1.48423478247234610E-03
-2.73603162625860610E-03
6.40485328521245350E-04];
```

Save the Beylkin(18) filter and add the new filter to the toolbox.

```
save beyl beyl
```

Use `wavemngr` to add the wavelet filter to the toolbox. Define the wavelet family name and the short name used to access the filter. Define the wavelet type to be 1. Type 1 wavelets are orthogonal wavelets in the toolbox. Because you are adding only one wavelet in this family, define the NUMS variable input to `wavemngr` to be an empty string.

```
familyName      = 'beylkin';  
familyShortName = 'beyl';  
familyWaveType  = 1;  
familyNums      = '';  
fileWaveName    = 'beyl.mat';
```

Add the wavelet using `wavemngr`.

```
wavemngr('add',familyName,familyShortName,familyWaveType, ...  
        familyNums,fileWaveName)
```

Verify that the wavelet has been added to the toolbox.

```
wavemngr('read')
```

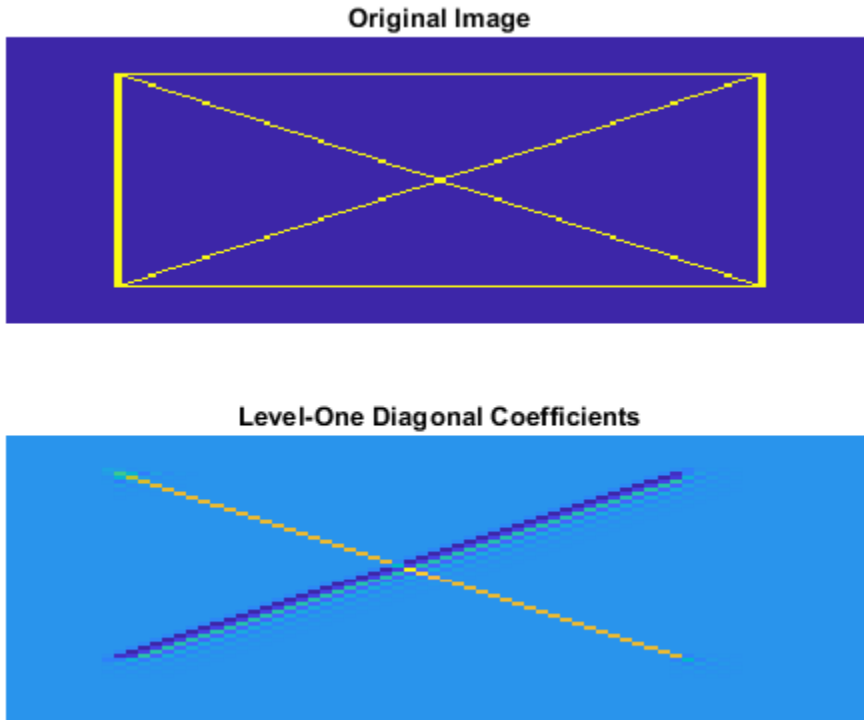
```
ans = 19x35 char array  
'===== '  
'Haar          ->->haar      '  
'Daubechies    ->->db        '  
'Symlets       ->->sym       '  
'Coiflets      ->->coif      '  
'BiorSplines   ->->bior      '  
'ReverseBior   ->->rbio      '  
'Meyer         ->->meyr      '  
'DMeyer        ->->dmey      '  
'Gaussian      ->->gaus      '  
'Mexican_hat   ->->mexh      '  
'Morlet        ->->morl      '  
'Complex Gaussian ->->cgau    '  
'Shannon       ->->shan      '  
'Frequency B-Spline->->fbsp    '  
'Complex Morlet ->->cmor      '  
'Fejer-Korovkin ->->fk        '  
'beylkin       ->->beyl      '  
'===== '
```

You can now use the wavelet to analyze signals or images. For example, load an ECG signal and obtain the MODWT of the signal down to level four using the Beylkin(18) filter.

```
load wecg
wtecg = modwt(wecg, 'beyl', 4);
```

Load a box image, obtain the 2-D DWT using the Beylkin(18) filter. Show the level-one diagonal detail coefficients.

```
load xbox
[C,S] = wavedec2(xbox,1, 'beyl');
[H,V,D] = detcoef2('all',C,S,1);
subplot(2,1,1)
imagesc(xbox)
axis off
title('Original Image')
subplot(2,1,2)
imagesc(D)
axis off
title('Level-One Diagonal Coefficients')
```



Finally, verify that the new filter satisfies the conditions for an orthogonal QMF pair. Obtain the scaling (lowpass) and wavelet (highpass) filters.

```
[Lo,Hi] = wfilters('beyl');
```

Sum the lowpass filter coefficients to verify that the sum equals $\sqrt{2}$. Sum the wavelet filter coefficients and verify that the sum is 0.

```
sum(Lo)
```

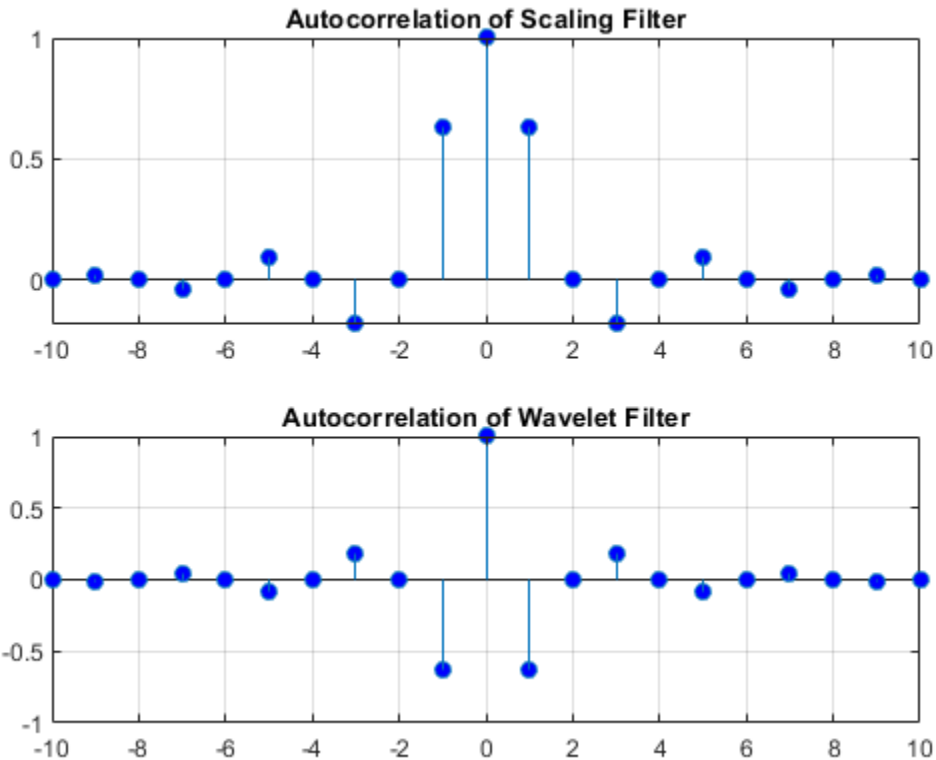
```
ans = 1.4142
```

```
sum(Hi)
```

```
ans = -1.9873e-16
```

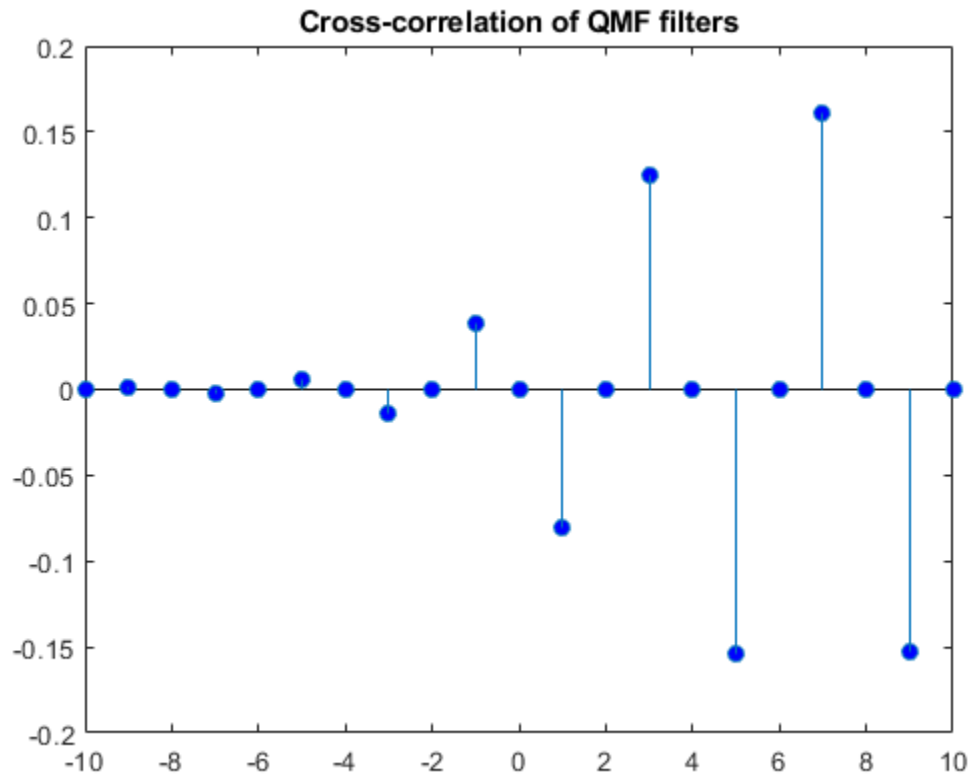

Verify that the autocorrelation of the scaling and wavelet filters at all even nonzero lags is 0. You must have the Signal Processing Toolbox™ to use `xcorr`.

```
[Clow,lags] = xcorr(Lo,Lo,10);  
Chigh = xcorr(Hi,Hi,10);  
subplot(2,1,1)  
stem(lags,Clow,'markerfacecolor',[0 0 1])  
grid on;  
title('Autocorrelation of Scaling Filter');  
subplot(2,1,2)  
stem(lags,Chigh,'markerfacecolor',[0 0 1])  
grid on;  
title('Autocorrelation of Wavelet Filter');
```

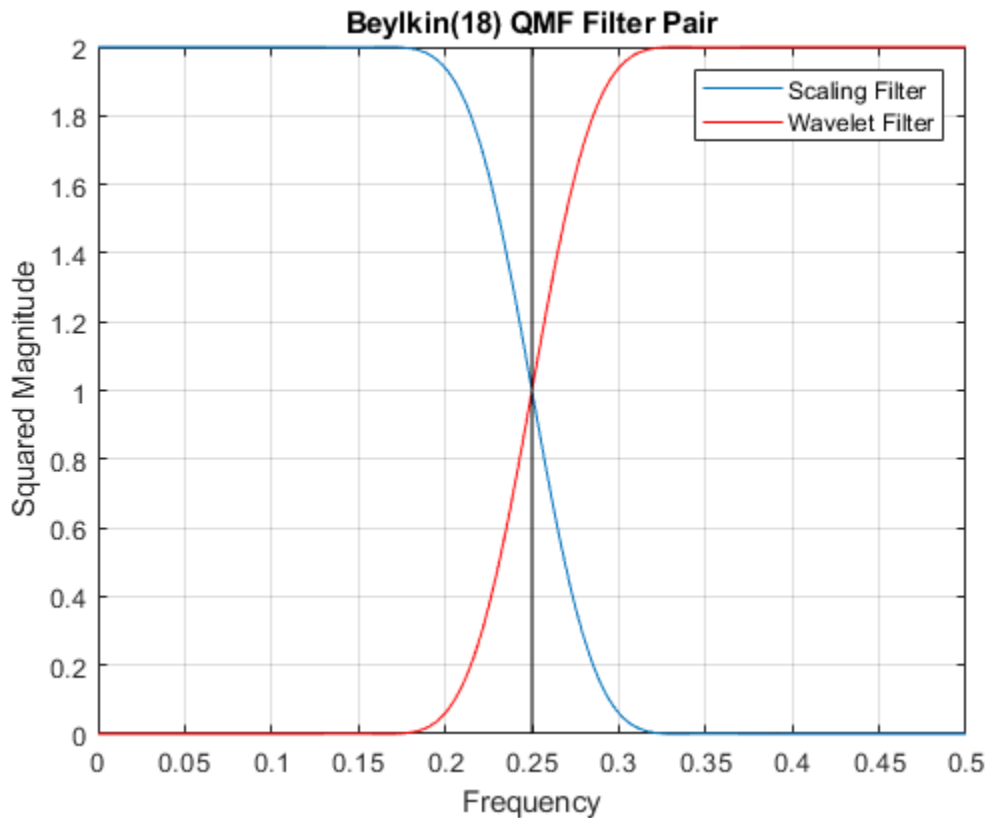


Note that the autocorrelation values in both plots is zero for nonzero even lags. Show that the cross-correlation of the scaling and wavelet filter is zero at all even lags.

```
[xcr,lags] = xcorr(Lo,Hi,10);  
figure  
stem(lags,xcr,'markerfacecolor',[0 0 1]);  
title('Cross-correlation of QMF filters')
```



The final criterion states the sum of squared magnitudes of the Fourier transforms of scaling and wavelet filters at each frequency is equal to 2. In other words, let $G(f)$ be the Fourier transform of the scaling filter and $H(f)$ be the Fourier transform of the wavelet filter. The following holds for all f : $|H(f)|^2 + |G(f)|^2 = 2$. The DFT version of this equality



Note the magnitude responses are symmetric, or mirror images, of each other around the quadrature frequency of $1/4$.

The following code removes the Beylkin(18) wavelet filter.

```
wavemngr('del', familyShortName);  
delete('beyl.mat')
```

Adding a Biorthogonal Wavelet

Adding a biorthogonal wavelet to the toolbox is similar to adding a QMF. You provide valid lowpass (scaling) filters pair used in analysis and synthesis. The `wfilters` function will generate the highpass filters.

To be recognized by `wfilters`, the analysis scaling filter **must** be assigned to the variable `Df`, and the synthesis scaling filter **must** be assigned to the variable `Rf`. The biorthogonal scaling filters do not have to be of even equal length. The output biorthogonal filter pairs created will have even equal lengths. Here are the scaling function pairs of the nearly-orthogonal biorthogonal wavelet quadruple based on the Laplacian pyramid scheme of Burt and Adelson.

```
Df = [-1 5 12 5 -1]/20*sqrt(2);
Rf = [-3 -15 73 170 73 -15 -3]/280*sqrt(2);
```

Save the filters to a `.mat` file.

```
save burt Df Rf
```

Use `wavemngr` to add the biorthogonal wavelet filters to the toolbox. Define the wavelet family name and the short name used to access the filter. Since the wavelets are biorthogonal, set the wavelet type to be 2. Because you are adding only one wavelet in this family, define the `NUMS` variable input to `wavemngr` to be an empty string.

```
familyName      = 'burtAdelson';
familyShortName = 'burt';
familyWaveType  = 2;
familyNums      = '';
fileWaveName    = 'burt.mat';
wavemngr('add', familyName, familyShortName, familyWaveType, ...
         familyNums, fileWaveName)
```

Verify that the biorthogonal wavelet has been added to the toolbox.

```
wavemngr('read')
```

```
ans = 19x35 char array
'=====
'Haar          ->->haar      |
'Daubechies    ->->db        |
'Symlets       ->->sym       |
'Coiflets      ->->coif      |
'BiorSplines   ->->bior      |
'ReverseBior   ->->rbio      |
'Meyer         ->->meyr      |
'DMeyer        ->->dmey      |
'Gaussian      ->->gaus      |
'Mexican_hat   ->->mexh      |
'Morlet        ->->morl      |
```

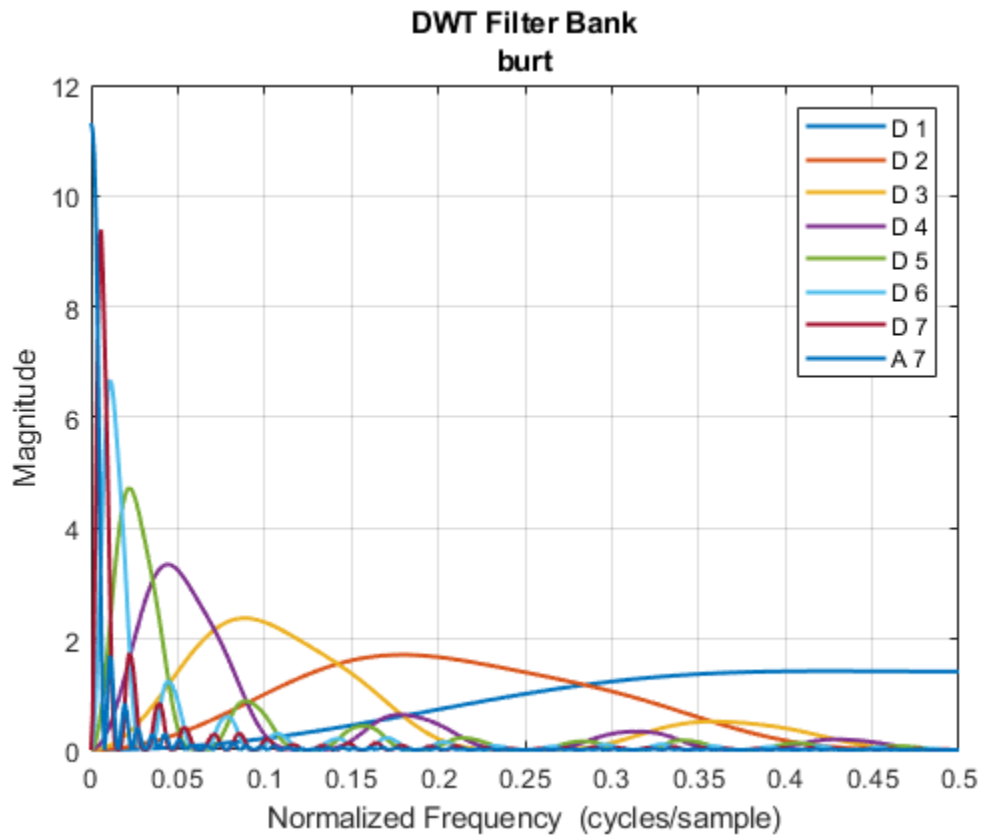
```
'Complex Gaussian ->->cgau      |
'Shannon          ->->shan      |
'Frequency B-Spline->->fbasp   |
'Complex Morlet   ->->cmor     |
'Fejer-Korovkin   ->->fk       |
'burtAdelson      ->->burt     |
'====='
```

You can now use the wavelet within the toolbox. Create an analysis DWT filter bank using the burt wavelet. Confirm the DWT filter bank is biorthogonal. Plot the magnitude frequency responses of the wavelet bandpass filters and coarsest resolution scaling function.

```
fb = dwtfilterbank('Wavelet','burt');
isBiorthogonal(fb)
```

```
ans = logical
     1
```

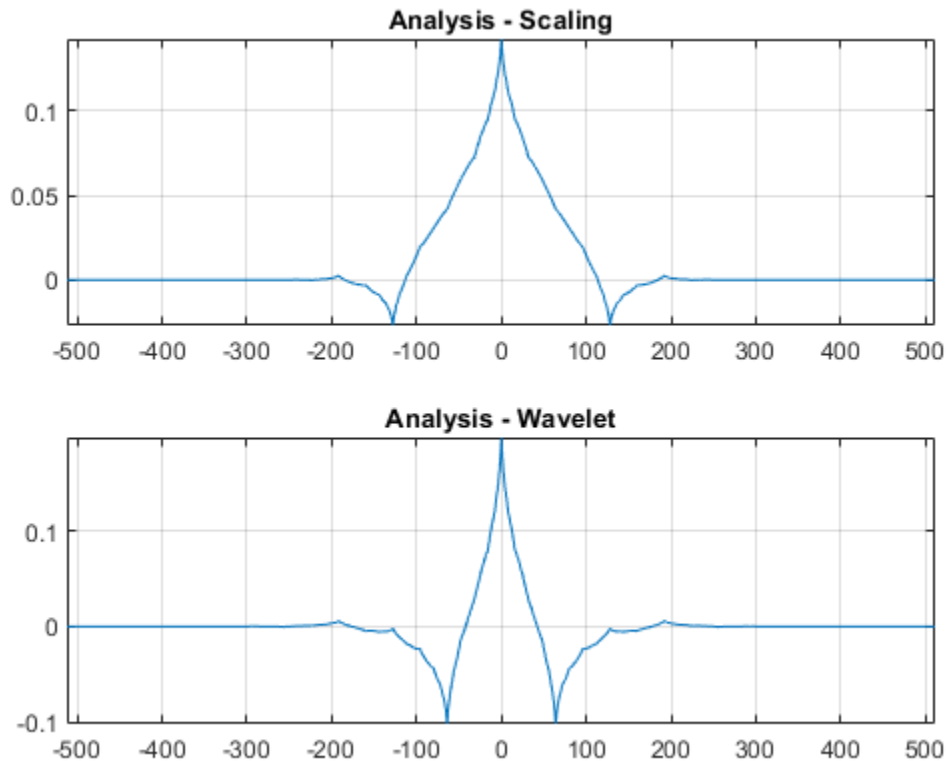
```
freqz(fb)
```



Obtain the wavelet and scaling functions of the filter bank. Plot the wavelet and scaling functions at the coarsest scale.

```
[fb_phi,t] = scalingfunctions(fb);
[fb_psi,~] = wavelets(fb);
subplot(2,1,1)
plot(t,fb_phi(end,:))
axis tight
grid on
title('Analysis - Scaling')
subplot(2,1,2)
plot(t,fb_psi(end,:))
axis tight
```

```
grid on  
title('Analysis - Wavelet')
```



Create a synthesis DWT filter bank using the burt wavelet. Compute the framebounds.

```
fb2 = dwtfilterbank('Wavelet', 'burt', 'FilterType', 'Synthesis', 'Level', 4);  
[synthesisLowerBound, synthesisUpperBound] = framebounds(fb2)
```

```
synthesisLowerBound = 0.9800
```

```
synthesisUpperBound = 1.0509
```

Obtain the lowpass and highpass analysis and synthesis filters associated with burt. Note the output filters are all of equal even length. Confirm the lowpass filter coefficients sum to $\sqrt{2}$ and the highpass filter coefficients sum to 0.


```
[LoD,HiD,LoR,HiR] = wfilters('burt');
[LoD' HiD' LoR' HiR']
```

```
ans = 8×4
```

```

      0      0.0152     -0.0152         0
      0     -0.0758     -0.0758         0
    -0.0707   -0.3687      0.3687    -0.0707
     0.3536     0.8586      0.8586    -0.3536
     0.8485   -0.3687      0.3687     0.8485
     0.3536   -0.0758     -0.0758    -0.3536
    -0.0707     0.0152     -0.0152    -0.0707
      0          0          0          0
```

```
sum([(LoD'/sqrt(2)) HiD' (LoR'/sqrt(2)) HiR'])
```

```
ans = 1×4
```

```

    1.0000   -0.0000    1.0000         0
```

Remove the Burt-Adelson filter from the Toolbox.

```
wavemngr('del',familyShortName);
delete('burt.mat')
```

References

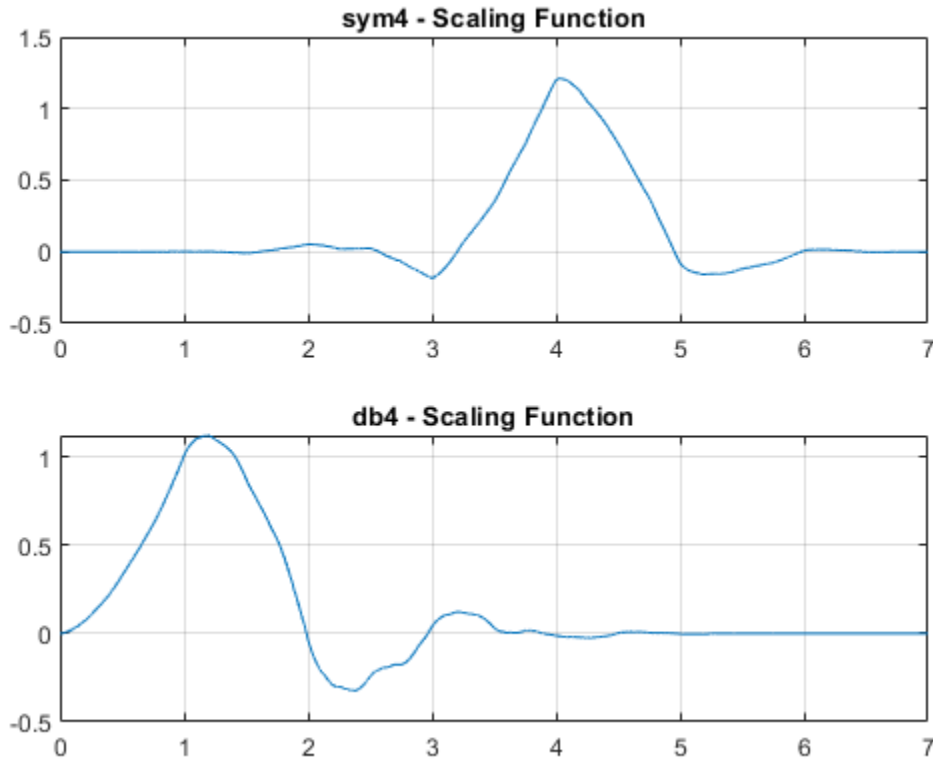
- [1] Daubechies, I. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1992.

Least Asymmetric Wavelet and Phase

For a given support, the orthogonal wavelet with a phase response that most closely resembles a linear phase filter is called least asymmetric. Symlets are examples of least asymmetric wavelets. They are modified versions of the classic Daubechies db wavelets. In this example you will show that the order 4 symlet has a nearly linear phase response, while the order 4 Daubechies wavelet does not.

First plot the order 4 symlet and order 4 Daubechies scaling functions. While neither is perfectly symmetric, note how much more symmetric the symlet is.

```
[phi_sym,~,xval_sym]=wavefun('sym4',10);  
[phi_db,~,xval_db]=wavefun('db4',10);  
subplot(2,1,1)  
plot(xval_sym,phi_sym)  
title('sym4 - Scaling Function')  
grid on  
subplot(2,1,2)  
plot(xval_db,phi_db)  
title('db4 - Scaling Function')  
grid on
```



Generate the filters associated with the order 4 symlet and Daubechies wavelets.

```
scal_sym = symaux(4,sqrt(2));
scal_db = dbaux(4,sqrt(2));
```

Compute the frequency response of the scaling synthesis filters.

```
[h_sym,w_sym] = freqz(scal_sym);
[h_db,w_db] = freqz(scal_db);
```

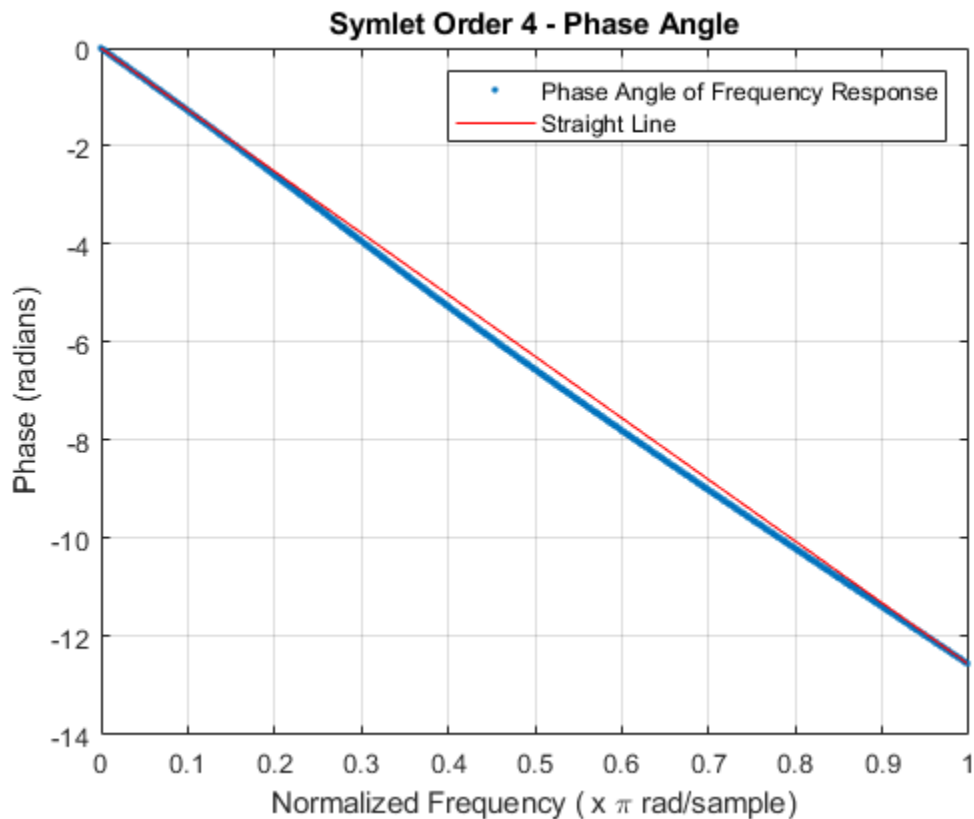
To avoid visual discontinuities, unwrap the phase angles of the frequency responses and plot them. Note how well the phase angle of the symlet filter approximates a straight line.

```
h_sym_u = unwrap(angle(h_sym));
h_db_u = unwrap(angle(h_db));
```

```

figure
plot(w_sym/pi,h_sym_u, '.')
hold on
plot(w_sym([1 end])/pi,h_sym_u([1 end]), 'r')
grid on
xlabel('Normalized Frequency ( x \pi rad/sample)')
ylabel('Phase (radians)')
legend('Phase Angle of Frequency Response', 'Straight Line')
title('Symlet Order 4 - Phase Angle')

```



```

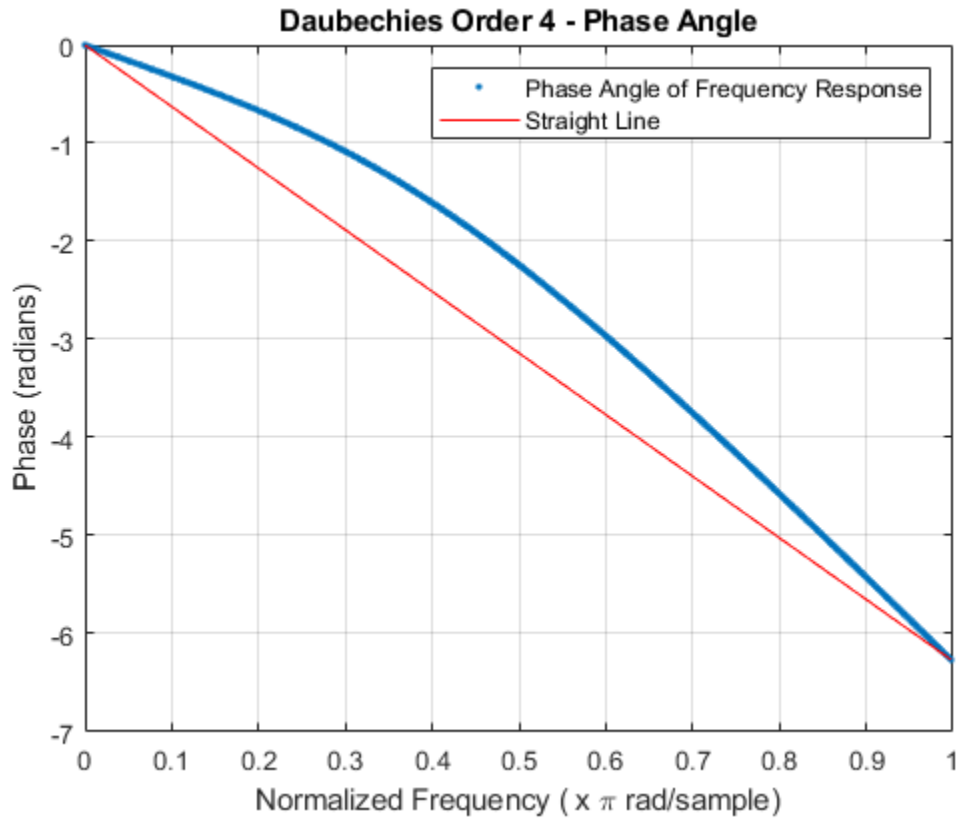
figure
plot(w_db/pi,h_db_u, '.')
hold on
plot(w_db([1 end])/pi,h_db_u([1 end]), 'r')

```

```

grid on
xlabel('Normalized Frequency ( x \pi rad/sample)')
ylabel('Phase (radians)')
legend('Phase Angle of Frequency Response','Straight Line')
title('Daubechies Order 4 - Phase Angle')

```



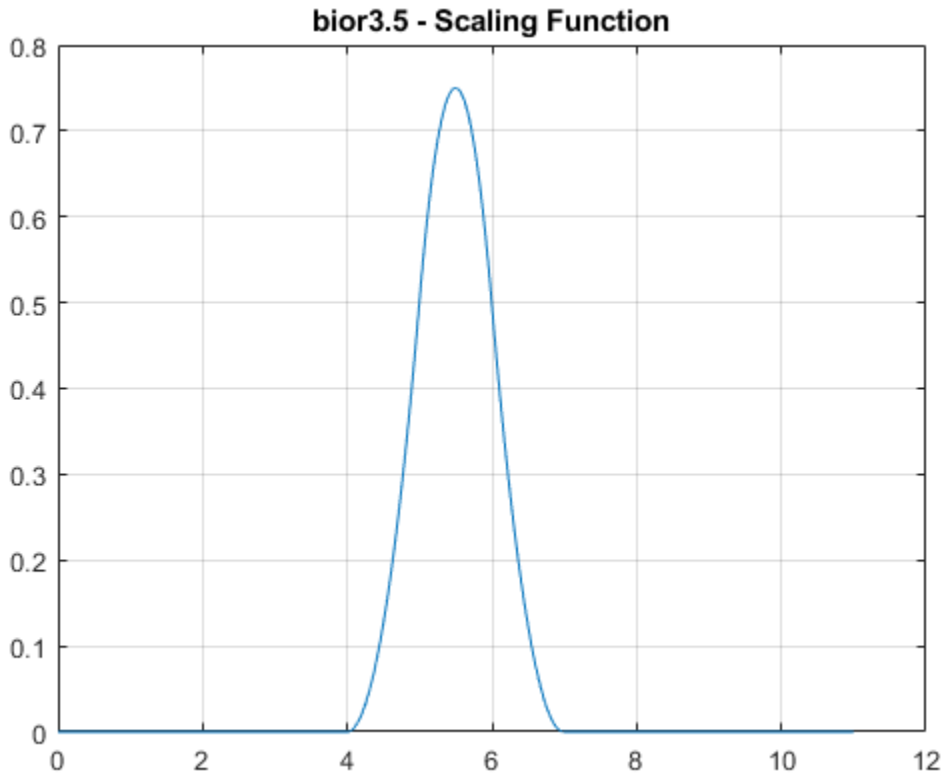
The sym4 and db4 wavelets are not symmetric, but the biorthogonal wavelet is. Plot the scaling function associated with the bior3.5 wavelet. Compute the frequency response of the synthesis scaling filter for the wavelet and verify that it has linear phase.

```

[~,~,phi_bior_r,~,xval_bior]=wavefun('bior3.5',10);
figure
plot(xval_bior,phi_bior_r)

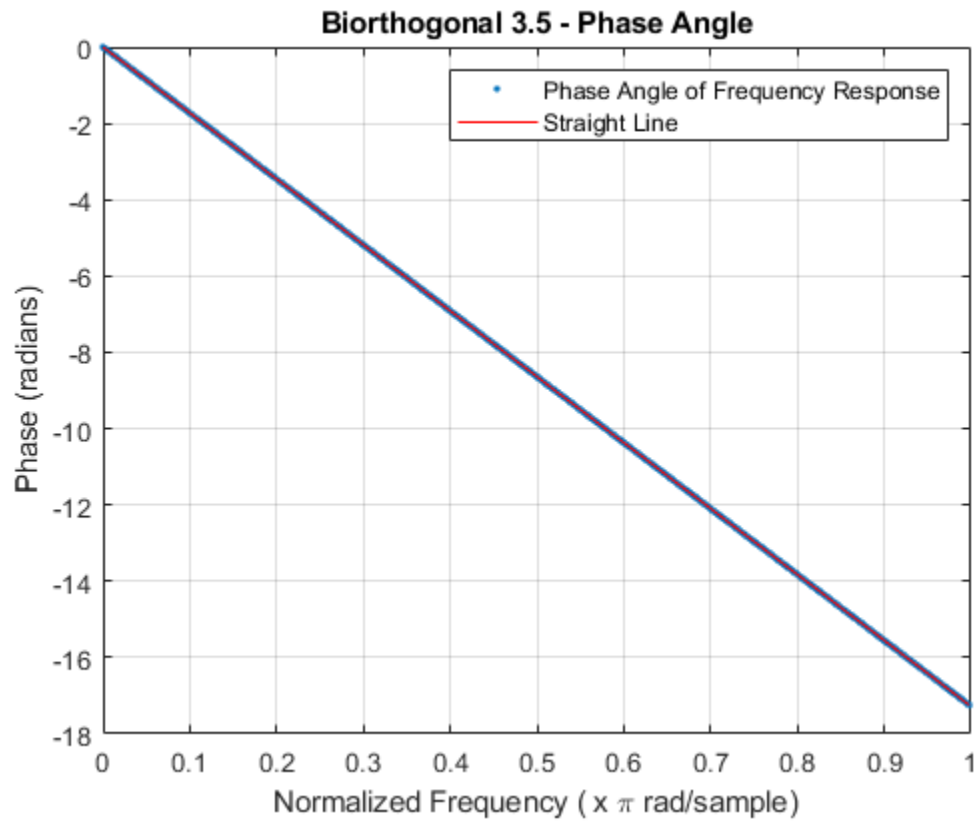
```

```
title('bior3.5 - Scaling Function')  
grid on
```



```
[LoD_bior,HiD_bior,LoR_bior,HiR_bior] = wfilters('bior3.5');  
[h_bior,w_bior] = freqz(LoR_bior);  
h_bior_u = unwrap(angle(h_bior));  
figure  
plot(w_bior/pi,h_bior_u,'.')  
hold on  
plot(w_bior([1 end])/pi,h_bior_u([1 end]),'r')  
grid on  
xlabel('Normalized Frequency ( x \pi rad/sample)')  
ylabel('Phase (radians)')
```

```
legend('Phase Angle of Frequency Response','Straight Line')  
title('Biorthogonal 3.5 - Phase Angle')
```



See Also

dbaux | symaux

Continuous Wavelet Analysis

- “1-D Continuous Wavelet Analysis” on page 2-2
- “Continuous Wavelet Analysis of Noisy Sinusoid Using Command Line Functions” on page 2-4
- “Continuous Wavelet Analysis of Noisy Sinusoid Using the Wavelet Analyzer App” on page 2-8
- “Importing and Exporting Information from the Wavelet Analyzer App” on page 2-19
- “Morse Wavelets” on page 2-21
- “Boundary Effects and the Cone of Influence” on page 2-32
- “Time-Frequency Analysis and Continuous Wavelet Transform” on page 2-45
- “Continuous Wavelet Analysis of Modulated Signals” on page 2-57
- “Remove Time-Localized Frequency Components” on page 2-61
- “Time-Varying Coherence” on page 2-67
- “Continuous Wavelet Analysis of Cusp Signal” on page 2-72
- “Complex Continuous Analysis Using the Wavelet Analyzer App” on page 2-75
- “DFT-Based Continuous Wavelet Analysis Using the Wavelet Analyzer App” on page 2-80
- “Two-Dimensional CWT of Noisy Pattern” on page 2-89
- “2-D Continuous Wavelet Transform App” on page 2-98

1-D Continuous Wavelet Analysis

Note

This page is no longer recommended. See “Continuous and Discrete Wavelet Transforms”.

The Wavelet Toolbox software enables you to perform a continuous wavelet analysis of your univariate or bivariate 1-D input signals. You can perform continuous wavelet analyses at the command line or with the app which you access by typing `waveletAnalyzer` at the command line.

Key features include:

- Continuous wavelet transform (CWT) of a 1-D input signal using real-valued and complex-valued wavelets. The Wavelet Toolbox software features a CWT algorithm, `cwt`, which is based on the correlation of the signal with an analyzing analytic wavelet, .
- Inverse CWT of 1-D input signal. For select analyzing wavelets, you can invert the CWT to reconstruct a time and scale-localized approximation to your input signal. See `icwt` for details.
- Wavelet cross spectrum and coherence. You can use `wcoherence` to compute the wavelet cross spectrum and coherence between two time series. The wavelet cross spectrum and coherence can reveal localized similarities between two time series in time and scale. See “Compare Time-Frequency Content in Signals with Wavelet Coherence” for examples.

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

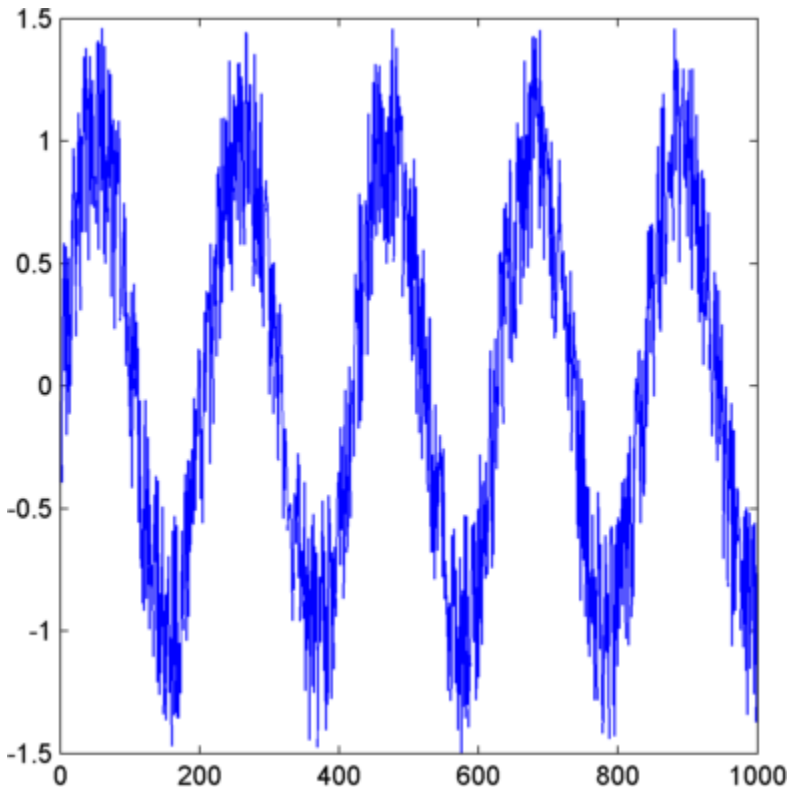
- Zoom in on detail
- Display coefficients in normal or absolute mode
- Choose the scales at which analysis is performed

Since you can perform analyses either from the command line or using the Wavelet Analyzer app, 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 Wavelet Analysis of Noisy Sinusoid Using Command Line Functions

This example involves a noisy sinusoidal signal.



- 1 Load a signal.

From the MATLAB prompt, type

```
load noissin;
```

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

```
whos
```

Name	Size	Bytes	Class
noissin	1x1000	8000	double array

2 Perform a Continuous Wavelet Transform.

Use the `cwt` command. Type

```
c = cwt(noissin);
```

The arguments to `cwt` specify the signal to be analyzed. The returned argument `c` contains the coefficients at various scales. In this case, `c` is a 80-by-1000 matrix with each row corresponding to a single scale.

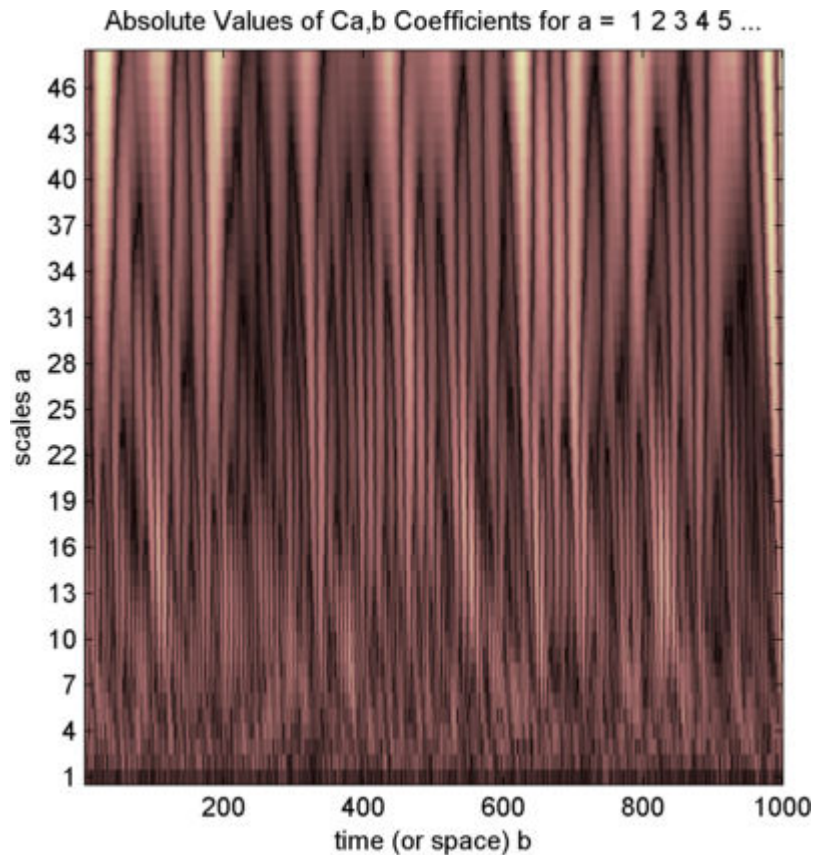
3 Plot the coefficients.

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.

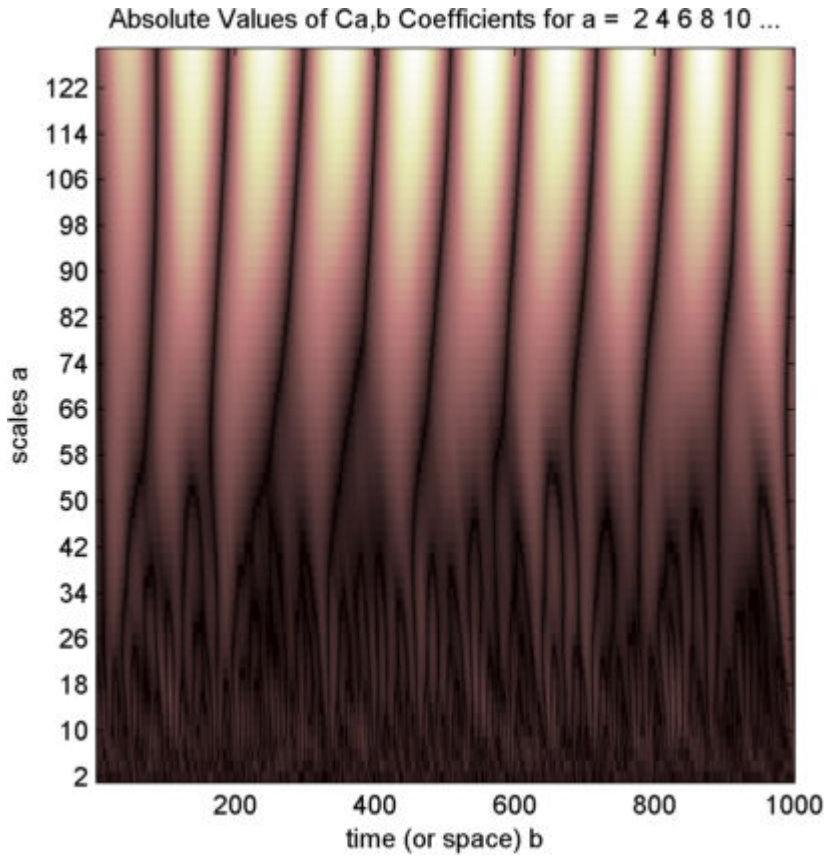
The second argument to `cwt` gives you fine control over the scale levels on which the continuous analysis is performed. In the previous example, we used all scales from 1 to 48, but you can construct any scale vector subject to these constraints:

- 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:



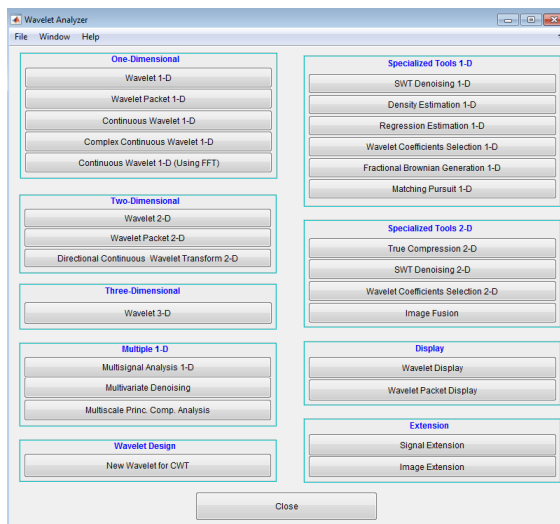
This plot gives a clearer picture of what's happening with the signal, highlighting the periodicity.

Continuous Wavelet Analysis of Noisy Sinusoid Using the Wavelet Analyzer App

This example shows how to use **Continuous Wavelet 1-D** tool to analyze a noisy sinusoidal signal.

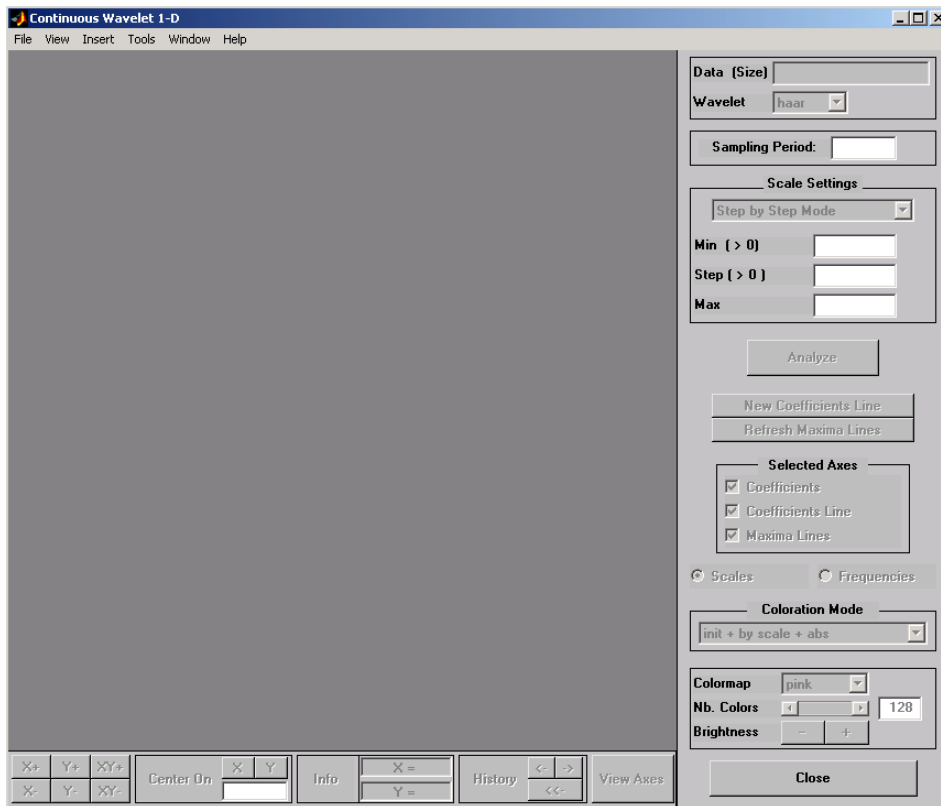
- 1 Start the Continuous Wavelet 1-D Tool. From the MATLAB prompt, type `waveletAnalyzer`

The **Wavelet Analyzer** appears.



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

The continuous wavelet analysis tool for 1-D signal data appears.



2 Load a signal.

At the MATLAB command prompt, type

```
load noissin;
```

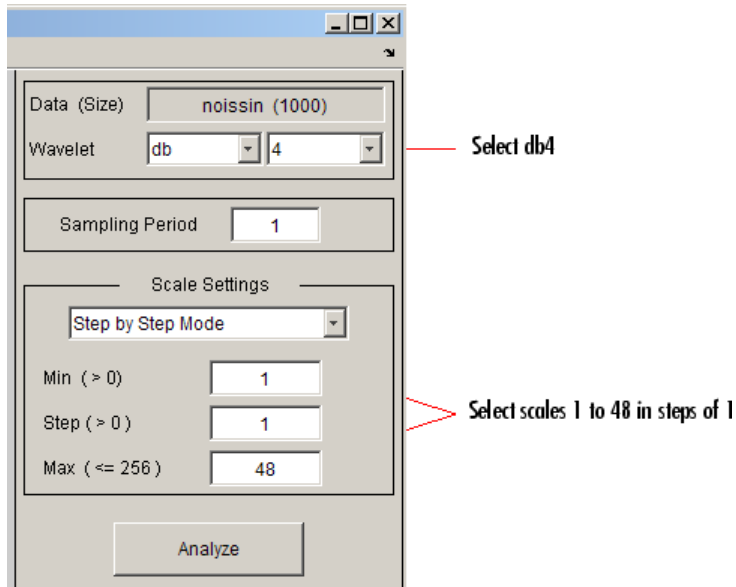
In the **Continuous Wavelet 1-D** tool, select **File > Import from Workspace**. When the **Import from Workspace** dialog box appears, select the `noissin` variable. Click **OK** to import the noisy sinusoid signal.

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

3 Perform a Continuous Wavelet Transform.

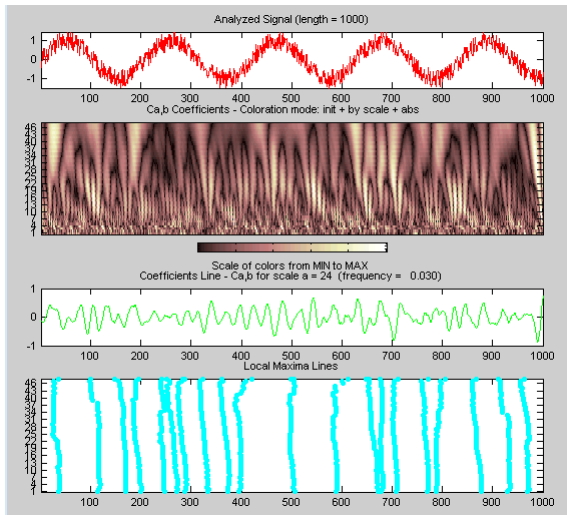
To start our analysis, let's perform an analysis using the `db4` wavelet at scales 1 through 48, just as we did using command line functions in the previous section.

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.

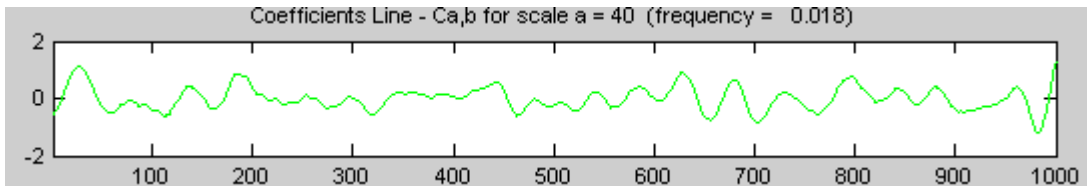
After a pause for computation, the tool displays the coefficients plot, the coefficients line plot corresponding to the scale $a = 24$, and the local maxima plot, which displays the chaining across scales (from $a = 48$ down to $a = 1$) of the coefficients local maxima.



5 View Wavelet Coefficients Line.

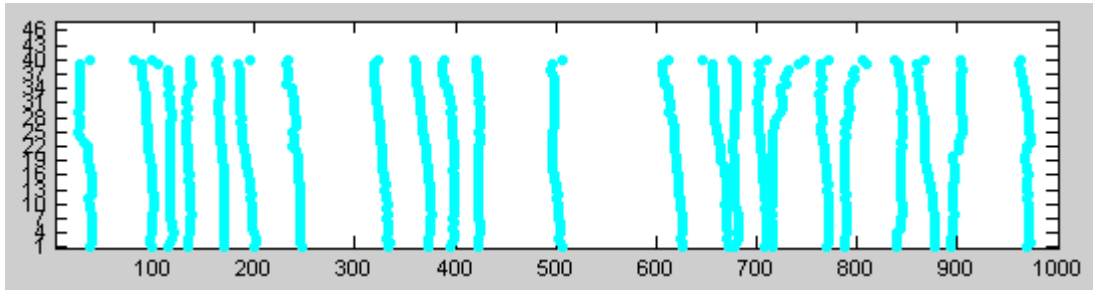
Select another scale $a = 40$ by clicking in the coefficients plot with the right mouse button. See step 9 to know, more precisely, how to select the desired scale.

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

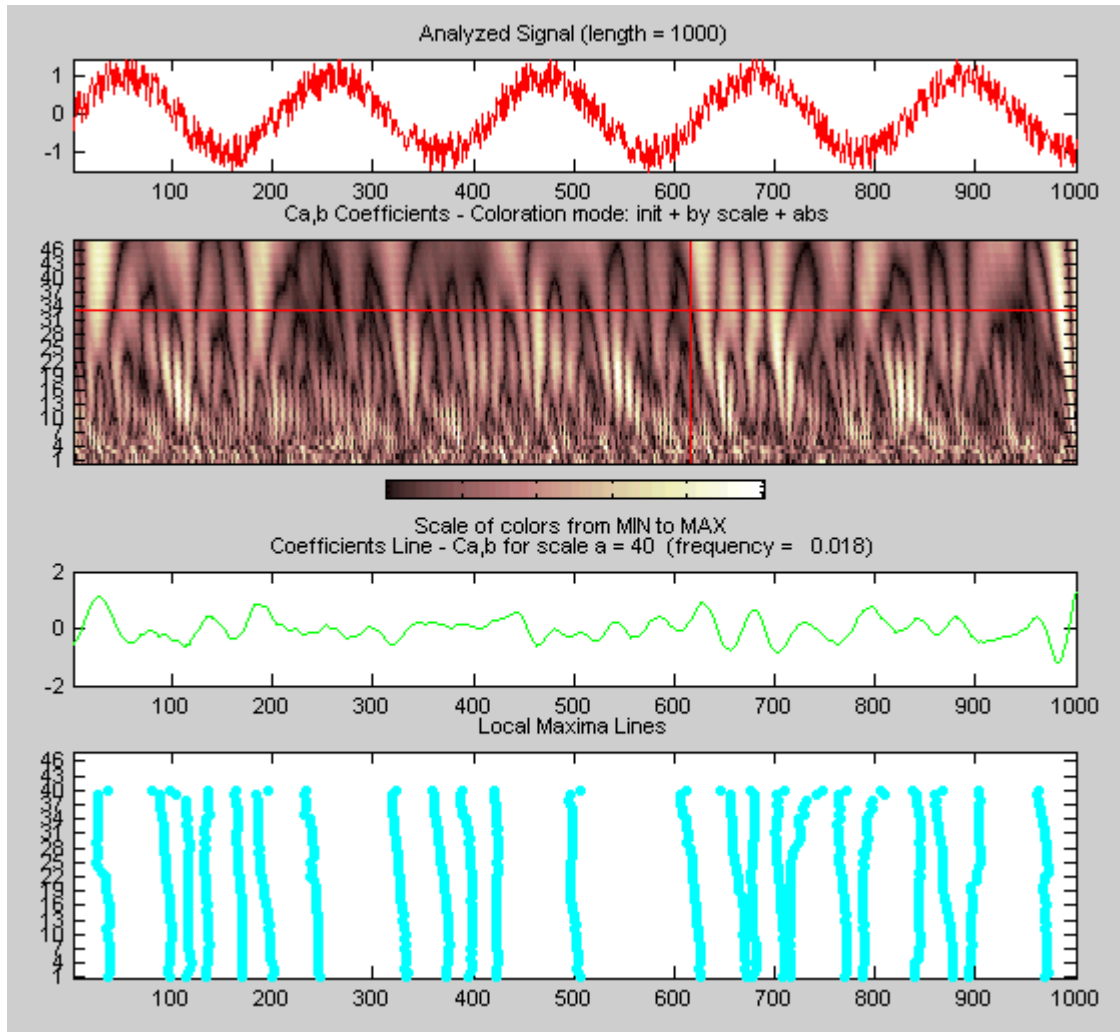


6 View Maxima Line.

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 $a = 1$.

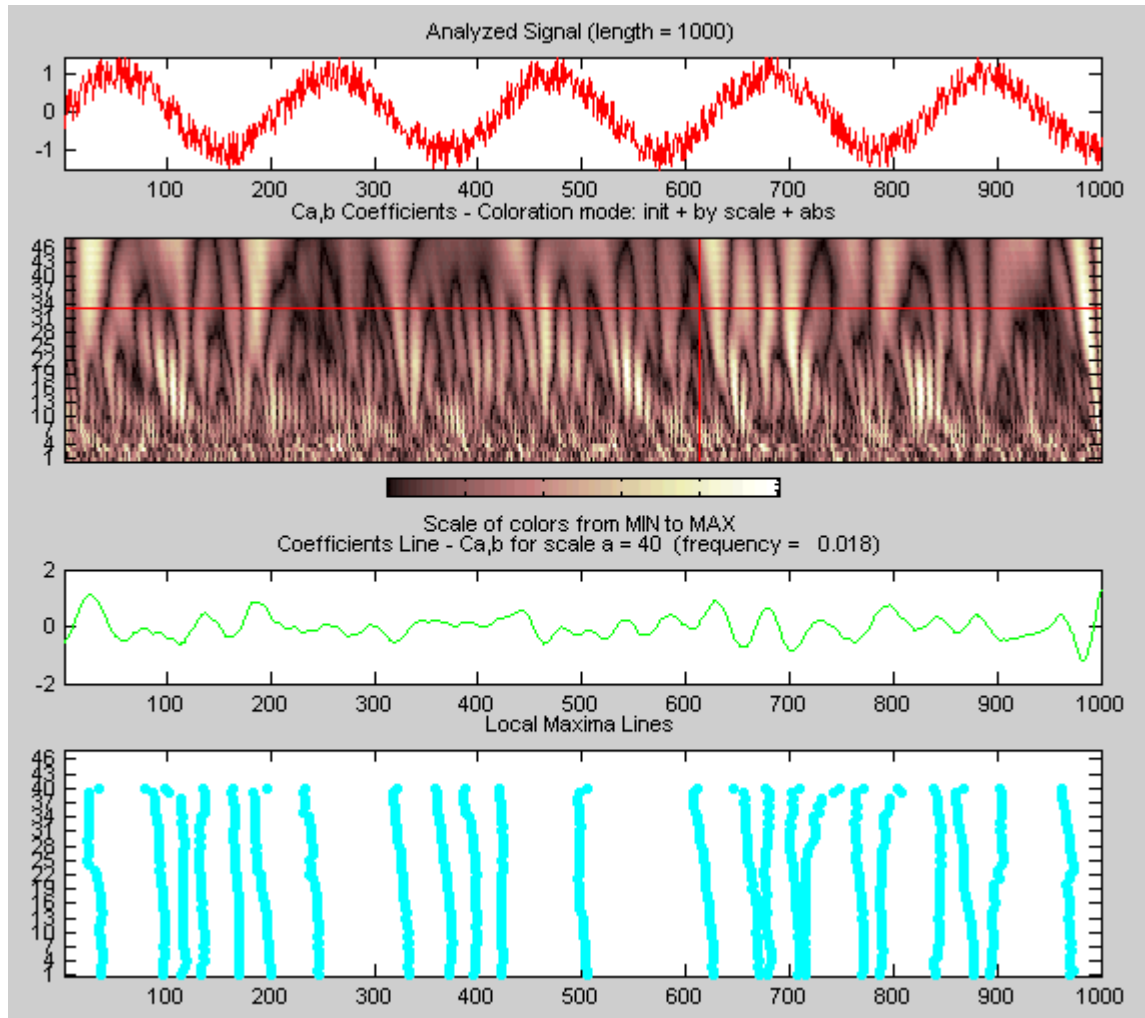


Hold down the right mouse button over the coefficients plot. The position of the mouse is given by the **Info** frame (located at the bottom of the screen) in terms of location (**X**) and scale (**Sca**).



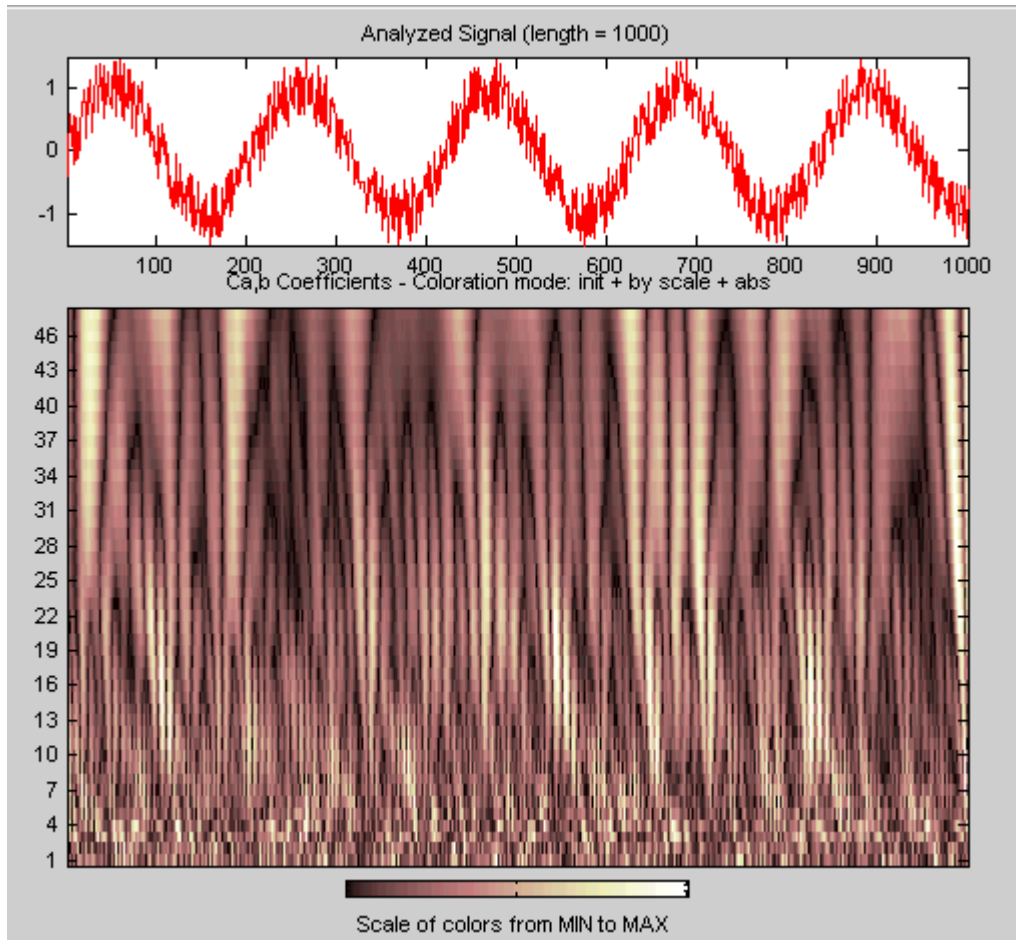
- 7 Switch from scale to Pseudo-Frequency Information.

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.



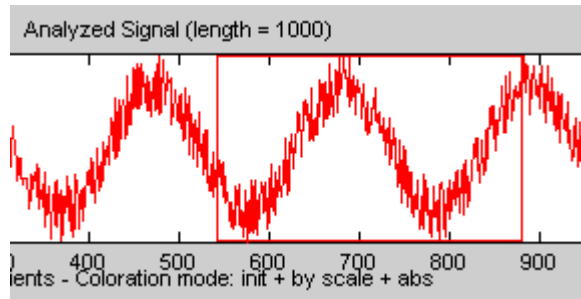
This facility allows you to interpret scale in terms of an associated pseudo-frequency, which depends on the wavelet and the sampling period..

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

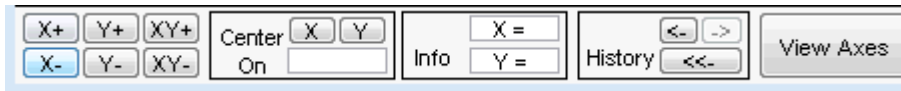


9 Zoom in on detail.

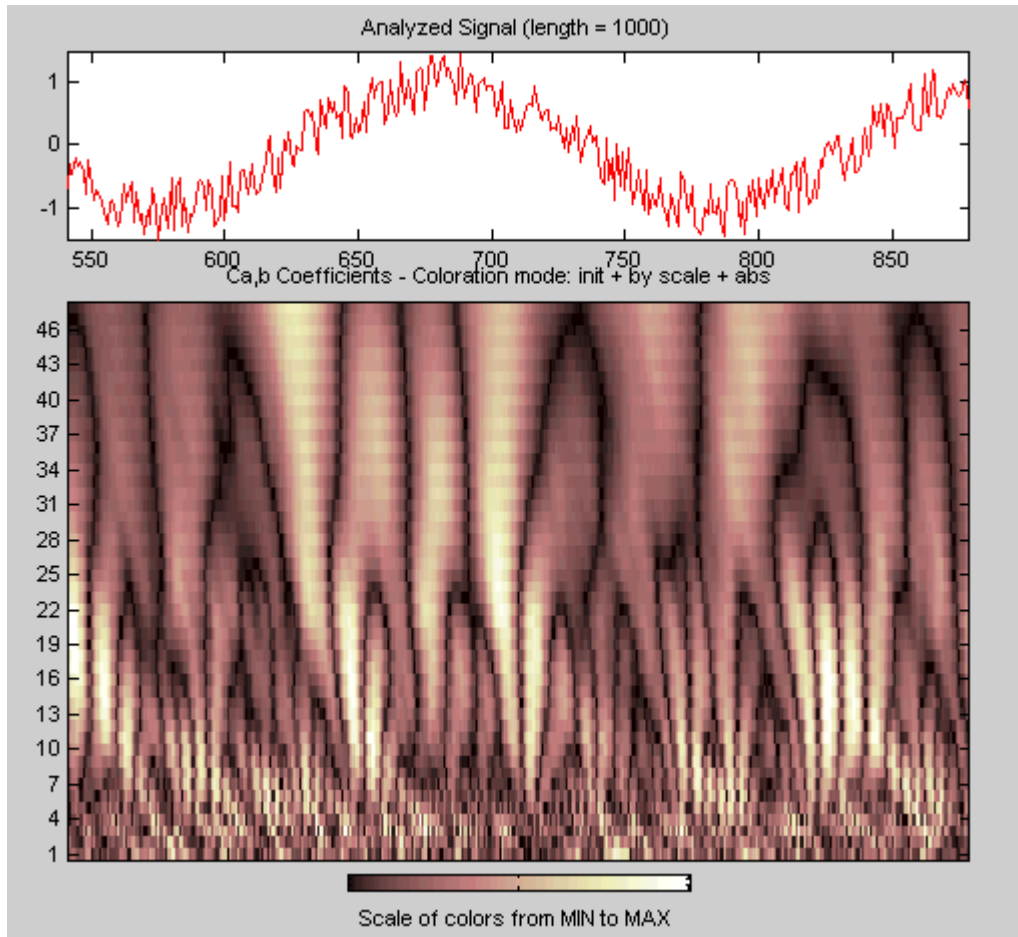
Drag a rubber band box (by holding down the left mouse button) over the portion of the signal you want to magnify.



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



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



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

- 11** View normal or absolute coefficients.

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

More generally, the coefficients coloration can be done in several different ways. For more details on the Coloration Mode, see “Controlling the Coloration Mode” on page A-8.

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.

Importing and Exporting Information from the Wavelet Analyzer App

The Continuous Wavelet 1-D tool in the Wavelet Analyzer app 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

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

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

```
save warma warma
```

The workspace variable `warma` must be a vector.

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

```
sizwarma =  
         1      1000
```

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

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

Saving Wavelet Coefficients

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

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

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

Consider the example analysis:

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

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

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

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

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

Morse Wavelets

In this section...

“What Are Morse Wavelets?” on page 2-21

“Morse Wavelet Parameters” on page 2-22

“Effect of Parameter Values on Morse Wavelet Shape” on page 2-22

“Relationship Between Analytic Morse Wavelet and Analytic Signal” on page 2-24

“Comparison of Analytic Wavelet Transform and Analytic Signal Coefficients” on page 2-25

“Recommended Morse Wavelet Settings for the CWT” on page 2-30

“References” on page 2-31

What Are Morse Wavelets?

Generalized Morse wavelets are a family of exactly analytic wavelets. Analytic wavelets are complex-valued wavelets whose Fourier transforms are supported only on the positive real axis. They are useful for analyzing modulated signals, which are signals with time-varying amplitude and frequency. They are also useful for analyzing localized discontinuities. The seminal paper for generalized Morse wavelets is Olhede and Walden [1]. The theory of Morse wavelets and their applications to the analysis of modulated signals is further developed in a series of papers by Lilly and Olhede [2], [3], and [4]. Efficient algorithms for the computation of Morse wavelets and their properties were developed by Lilly [5].

The Fourier transform of the generalized Morse wavelet is

$$\Psi_{P, \gamma}(\omega) = U(\omega) a_{P, \gamma} \omega^{\frac{P^2}{\gamma}} e^{-\omega^\gamma}$$

where $U(\omega)$ is the unit step, $a_{p, \gamma}$ is a normalizing constant, P^2 is the time-bandwidth product, and γ characterizes the symmetry of the Morse wavelet. Much of the literature about Morse wavelets uses β , which can be viewed as a decay or compactness parameter, rather than the time-bandwidth product, $P^2 = \beta\gamma$. The equation for the Morse wavelet in the Fourier domain parameterized by β and γ is

$$\Psi_{\beta, \gamma}(\omega) = U(\omega) a_{\beta, \gamma} \omega^\beta e^{-\omega^\gamma}$$

For a detailed explanation of the parameterization of Morse wavelets, see [2].

By adjusting the time-bandwidth product and symmetry parameters of a Morse wavelet, you can obtain analytic wavelets with different properties and behavior. A strength of Morse wavelets is that many commonly used analytic wavelets are special cases of a generalized Morse wavelet. For example, Cauchy wavelets have $\gamma = 1$ and Bessel wavelets are approximated by $\beta = 8$ and $\gamma = 0.25$.

Morse Wavelet Parameters

As previously mentioned, Morse wavelets have two parameters, symmetry and time-bandwidth product, which determine the wavelet shape and affect the behavior of the transform. The Morse wavelet gamma parameter, γ , controls the symmetry of the wavelet in time through the demodulate skewness [2]. The square root of the time-bandwidth product, P , is proportional to the wavelet duration in time. For convenience, the Morse wavelets in `cwt` and `cwtfilterbank` are parameterized as the time-bandwidth product and gamma. The duration determines how many oscillations can fit into the time-domain

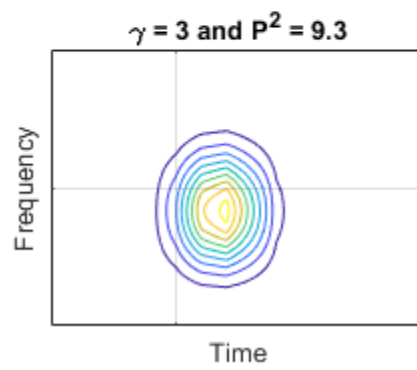
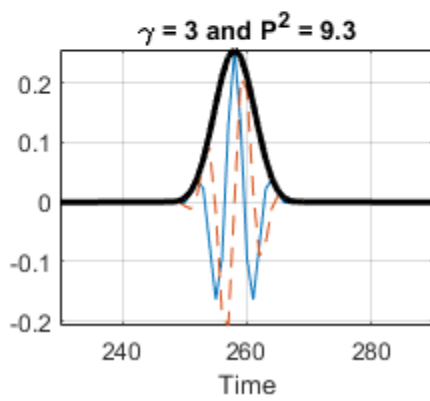
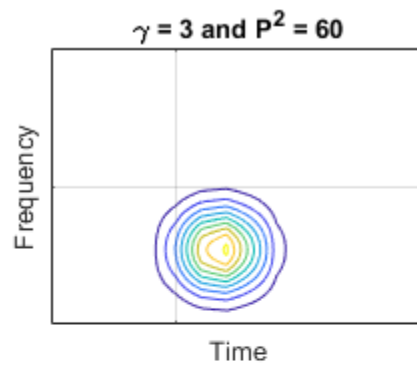
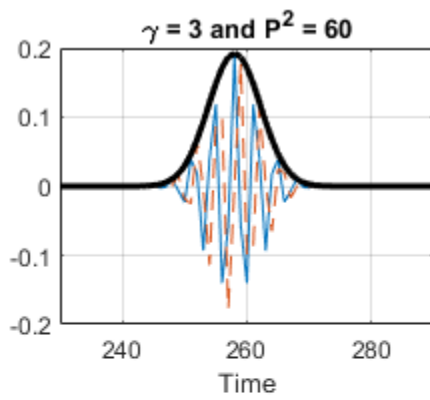
wavelet's center window at its peak frequency. The peak frequency is $\left(\frac{P^2}{\gamma}\right)^{\frac{1}{\gamma}}$.

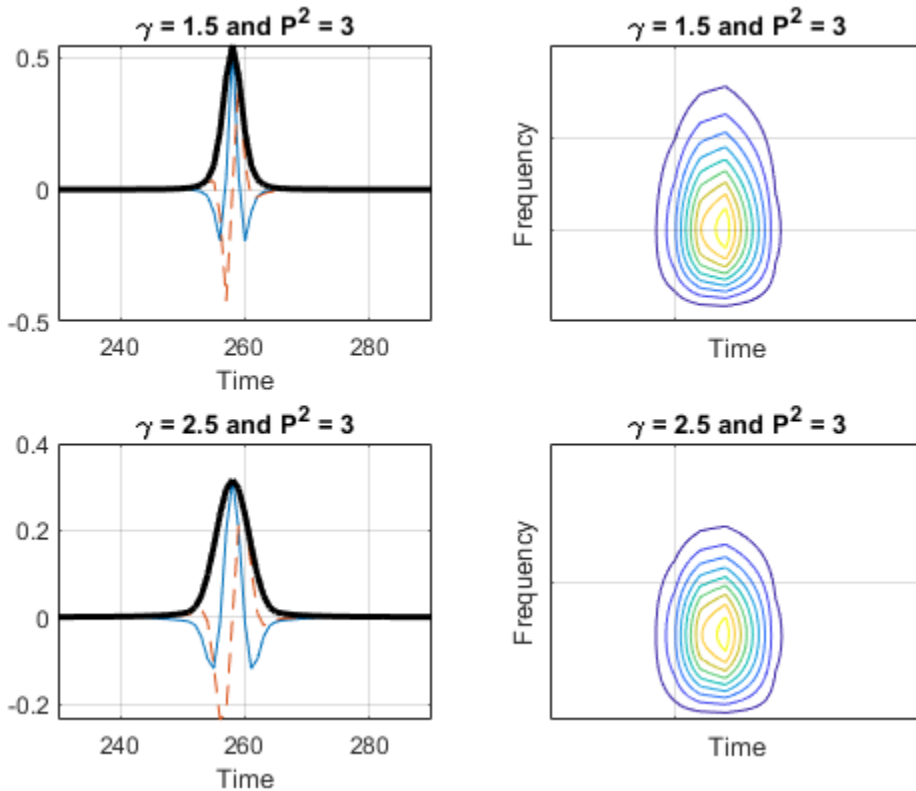
The (demodulate) skewness of the Morse wavelet is equal to 0 when gamma is equal to 3. The Morse wavelets also have the minimum Heisenberg area when gamma is equal to 3. For these reasons, `cwt` and `cwtfilterbank` use this as the default value.

Effect of Parameter Values on Morse Wavelet Shape

These plots show how different values of symmetry and time-bandwidth affect the shape of a Morse wavelet. Longer time-bandwidths broaden the central portion of the wavelet and increase the rate of the long time decay. Increasing the symmetry broadens the wavelet envelope, but does not affect the long time decay. For symmetry values less than or equal to 3, the time decay increases as the time-bandwidth increases. For symmetry greater than or equal to 3, reducing the time-bandwidth makes the wavelet less symmetric. As both symmetry and time-bandwidth increase, the wavelet oscillates more in time and narrows in frequency. Very small time-bandwidth and large symmetry values produce undesired time-domain sidelobes and frequency-domain asymmetry.

In the time-domain plots in the left column, the red line is the real part and the blue line is the imaginary part. The contour plots in the right column show how the parameters affect the spread in time and frequency.





Relationship Between Analytic Morse Wavelet and Analytic Signal

The coefficients from a wavelet transform using an analytic wavelet on a real signal are proportional to the coefficients of the corresponding analytic signal. An analytic signal is defined as the inverse Fourier transform of

$$\widehat{\chi}_{a(\omega)} = \widehat{\chi}(\omega) + \text{sgn}(\omega)\widehat{\chi}(\omega)$$

The value of the analytic signal depends on ω .

- For $\omega > 0$, the Fourier transform of an analytic signal is two times the Fourier transform of the corresponding nonanalytic signal, $\widehat{x}(\omega)$.
- For $\omega = 0$, the Fourier transform of an analytic signal is equal to the Fourier transform of the corresponding nonanalytic signal.
- For $\omega < 0$, the Fourier transform of an analytic signal vanishes.

Let $Wf(u, s)$ denote the wavelet transform of a signal, $f(t)$, at translation u and scale s . If the analyzing wavelet is analytic, you obtain $Wf(u, s) = \frac{1}{2}Wf_a(u, s)$, where $f_a(t)$ is the analytic signal corresponding to $f(t)$. For all wavelets used in cwt, the amplitude of the wavelet bandpass filter at the peak frequency for each scale is set to 2. Additionally, cwt uses L1 normalization. For a real-valued sinusoidal input with radian frequency ω_0 and amplitude A , the wavelet transform using an analytic wavelet yields coefficients that oscillate at the same frequency, ω_0 , with an amplitude equal to $\frac{A}{2}\widehat{\psi}(s\omega_0)$. By isolating the coefficients at the scale, $\frac{\omega_\psi}{\omega_0}$, a peak magnitude of 2 assures that the analyzed oscillatory component has the correct amplitude, A .

Comparison of Analytic Wavelet Transform and Analytic Signal Coefficients

This example shows how the analytic wavelet transform of a real signal approximates the corresponding analytic signal.

This is demonstrated using a sine wave. If you obtain the wavelet transform of a sine wave using an analytic wavelet and extract the wavelet coefficients at a scale corresponding to the frequency of the sine wave, the coefficients approximate the analytic signal. For a sine wave, the analytic signal is a complex exponential of the same frequency.

Create a sinusoid with a frequency of 50 Hz.

```
t = 0:.001:1;
x = cos(2*pi*50*t);
```

Obtain its continuous wavelet transform using an analytic Morse wavelet and the analytic signal. You must have the Signal Processing Toolbox™ to use `hilbert`.

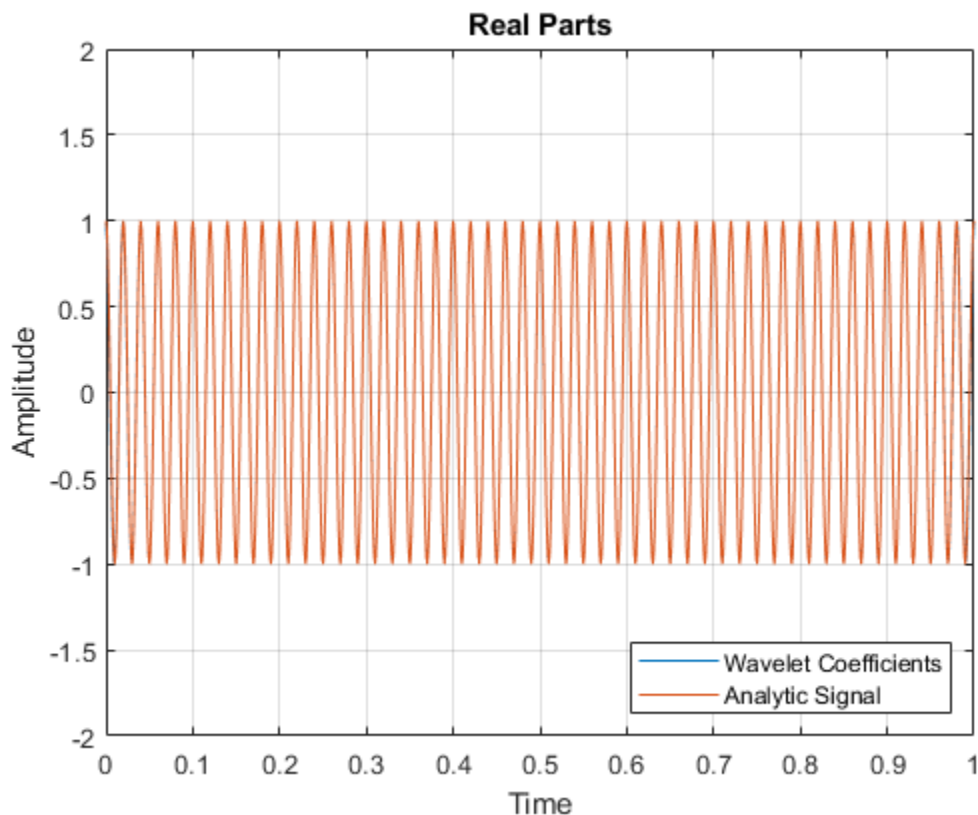
```
[wt,f] = cwt(x,1000,'voices',32,'ExtendSignal',false);
analytsig = hilbert(x);
```

Obtain the wavelet coefficients at the scale closest to the sine wave's frequency of 50 Hz.

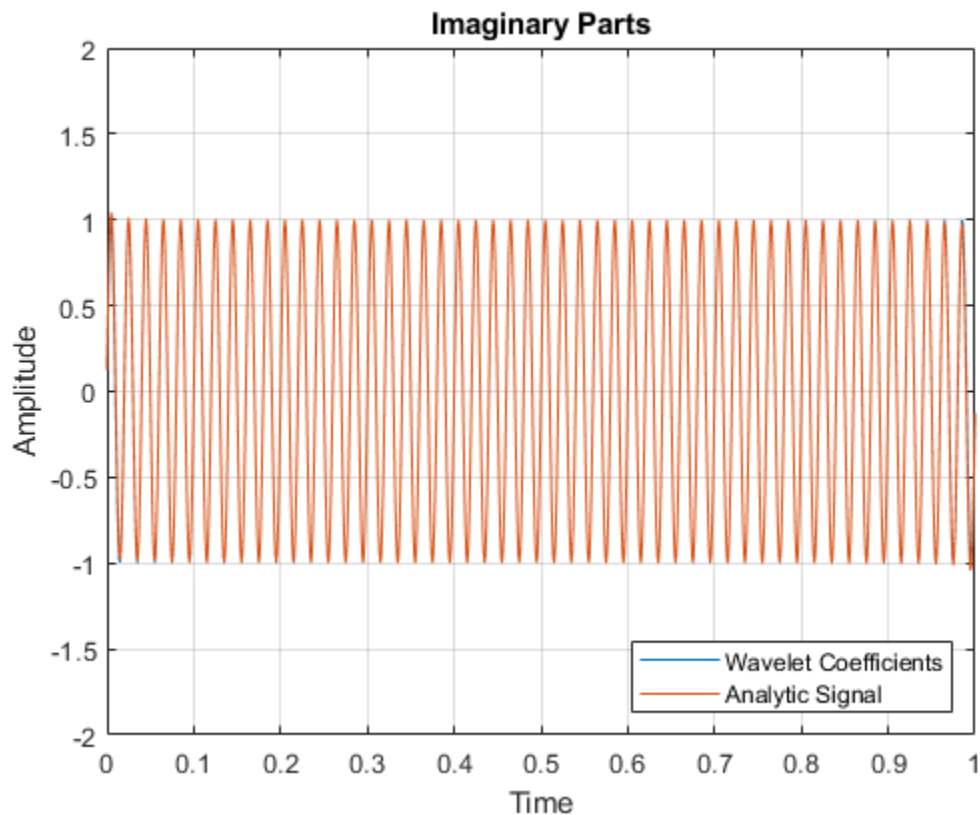
```
[~,idx] = min(abs(f-50));  
morsecoefx = wt(idx,:);
```

Compare the real and imaginary parts of the analytic signal with the wavelet coefficients at the signal frequency.

```
figure;  
plot(t,[real(morsecoefx)' real(analytsig)']);  
title('Real Parts');  
ylim([-2 2]); grid on;  
legend('Wavelet Coefficients','Analytic Signal','Location','SouthEast');  
xlabel('Time'); ylabel('Amplitude');
```



```
figure;  
plot(t,[imag(morsecoefx)' imag(analytsig)']);  
title('Imaginary Parts');  
ylim([-2 2]); grid on;  
legend('Wavelet Coefficients','Analytic Signal','Location','SouthEast');  
xlabel('Time'); ylabel('Amplitude');
```



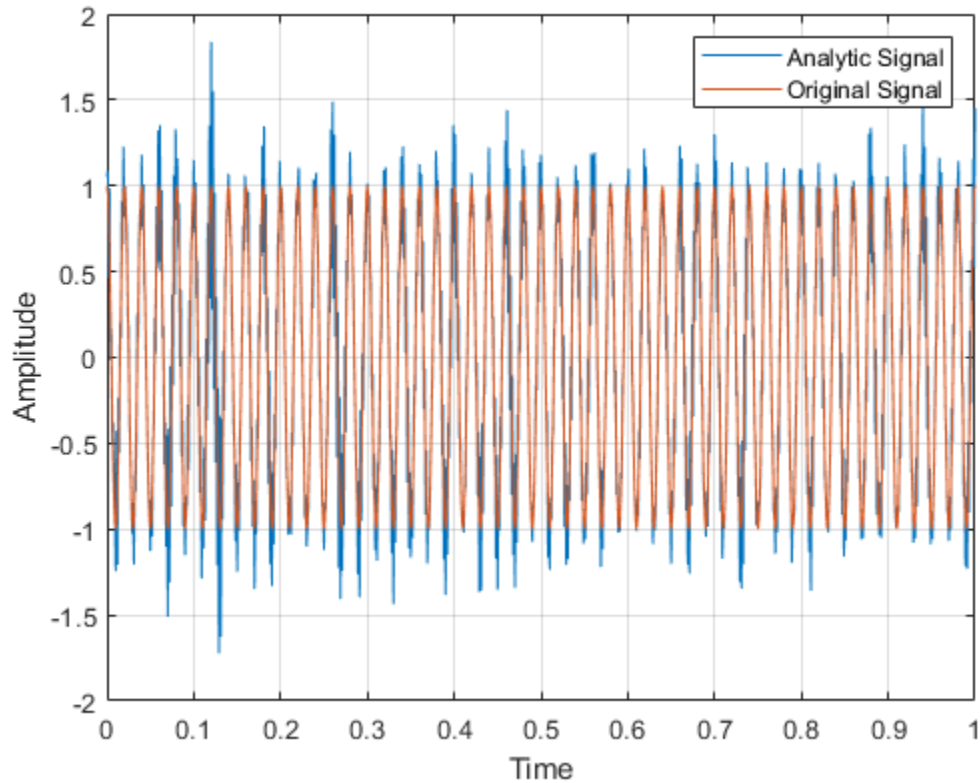
cwt uses L1 normalization and scales the wavelet bandpass filters to have a peak magnitude of 2. The factor of $1/2$ in the above equation is canceled by the peak magnitude value.

The wavelet transform represents a frequency-localized filtering of the signal. Accordingly, the CWT coefficients are less sensitive to noise than are the Hilbert transform coefficients.

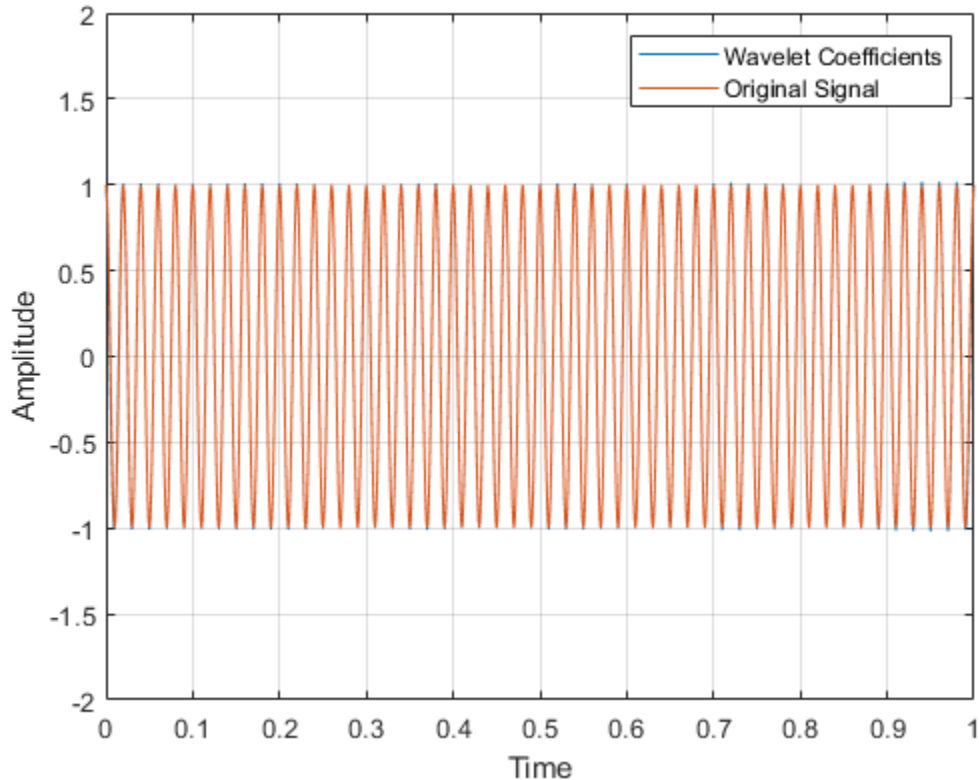
Add highpass noise to the signal and reexamine the wavelet coefficients and the analytic signal.

```
y = x + filter(1,[1 0.9],0.1*randn(size(x)));
analytsig = hilbert(y);
[wt,f] = cwt(y,1000,'voices',32,'ExtendSignal',0);
morsecoefy = wt(idx,:);

figure;
plot(t,[real(analytsig)' x']);
legend('Analytic Signal','Original Signal');
grid on;
xlabel('Time'); ylabel('Amplitude');
ylim([-2 2])
```



```
figure;  
plot(t,[real(morsecoefy)' x']);  
legend('Wavelet Coefficients','Original Signal');  
grid on;  
xlabel('Time'); ylabel('Amplitude');  
ylim([-2 2])
```



Recommended Morse Wavelet Settings for the CWT

For the best results when using the CWT, use a symmetry, γ , of 3, which is the default for `cwt` and `cwtfilterbank`. With gamma fixed, increasing the time-bandwidth product P^2 narrows the wavelet filter in frequency while increasing the width of the central portion of the filter in time. It also increases the number of oscillations of the wavelet under the central portion of the filter.

References

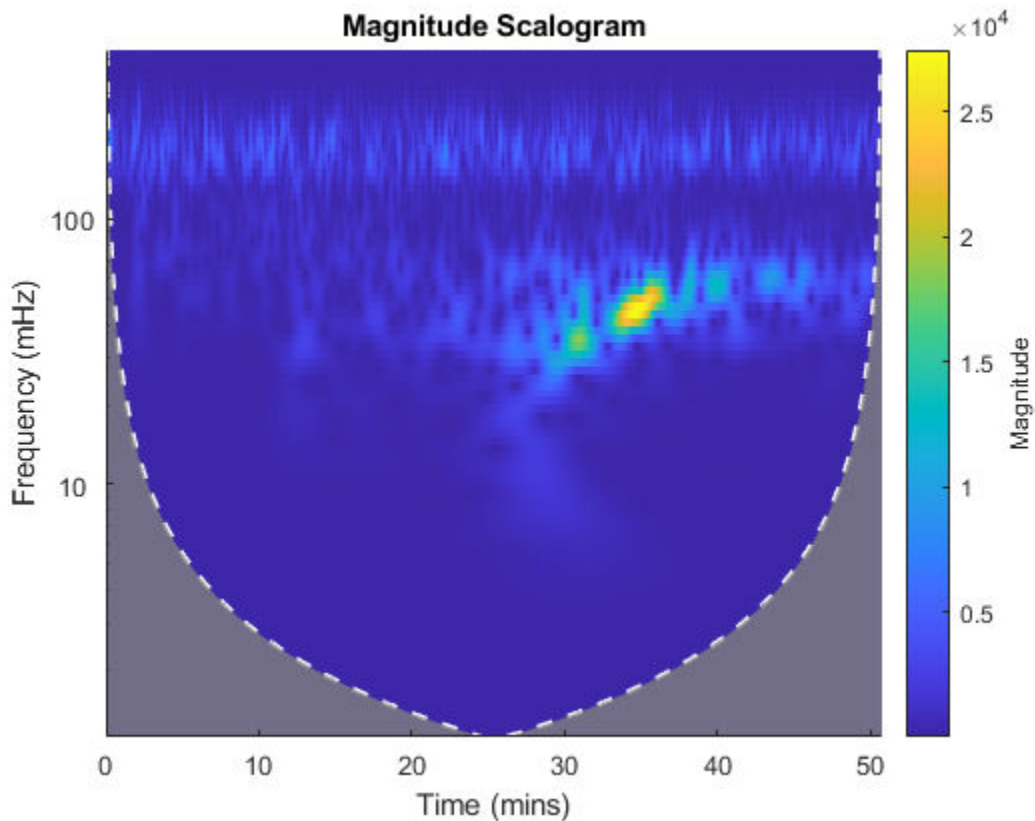
- [1] Olhede, S. C., and A. T. Walden. "Generalized morse wavelets." *IEEE Transactions on Signal Processing*, Vol. 50, No. 11, 2002, pp. 2661-2670.
- [2] Lilly, J. M., and S. C. Olhede. "Higher-order properties of analytic wavelets." *IEEE Transactions on Signal Processing*, Vol. 57, No. 1, 2009, pp. 146-160.
- [3] Lilly, J. M., and S. C. Olhede. "On the analytic wavelet transform." *IEEE Transactions on Information Theory*, Vol. 56, No. 8, 2010, pp. 4135-4156.
- [4] Lilly, J. M., and S. C. Olhede. "Generalized Morse wavelets as a superfamily of analytic wavelets." *IEEE Transactions on Signal Processing* Vol. 60, No. 11, 2012, pp. 6036-6041.
- [5] Lilly, J. M. *jLab: A data analysis package for Matlab*, version 1.6.2., 2016. <http://www.jmlilly.net/jmlsoft.html>.
- [6] Lilly, J. M. "Element analysis: a wavelet-based method for analysing time-localized events in noisy time series." *Proceedings of the Royal Society A*. Volume 473: 20160776, 2017, pp. 1-28. [dx.doi.org/10.1098/rspa.2016.0776](https://doi.org/10.1098/rspa.2016.0776).

Boundary Effects and the Cone of Influence

This topic explains the cone of influence (COI) and the convention Wavelet Toolbox™ uses to compute it. The topic also explains how to interpret the COI in the scalogram plot, and exactly how the COI is computed in `cwtfilterbank` and `cwt`.

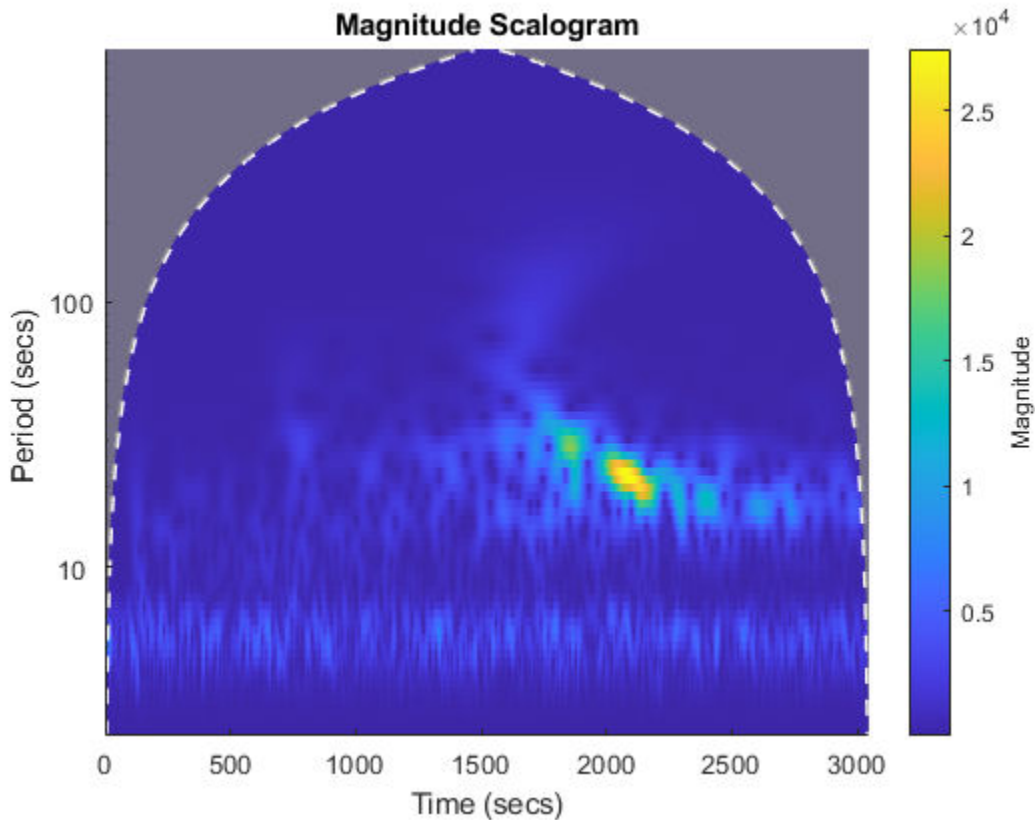
Load the Kobe earthquake seismograph signal. Plot the scalogram of the Kobe earthquake seismograph signal. The data is sampled at 1 hertz.

```
load kobe  
cwt(kobe,1)
```



In addition to the scalogram, the plot also features a dashed white line and shaded gray regions from the edge of the white line to the time and frequency axes. Plot the same data using the sampling interval instead of sampling rate. Now the scalogram is displayed in periods instead of frequency.

```
cwt(kobe, seconds(1))
```



The orientation of the dashed white line has flipped upside down, but the line and the shaded regions are still present.

The white line marks what is known as the *cone of influence*. The cone of influence includes the line and the shaded region from the edge of the line to the frequency (or period) and time axes. The cone of influence shows areas in the scalogram potentially affected by *edge-effect artifacts*. These are effects in the scalogram that arise from areas

where the stretched wavelets extend beyond the edges of the observation interval. Within the unshaded region delineated by the white line, you are sure that the information provided by the scalogram is an accurate time-frequency representation of the data. Outside the white line in the shaded region, information in the scalogram should be treated as suspect due to the potential for edge effects.

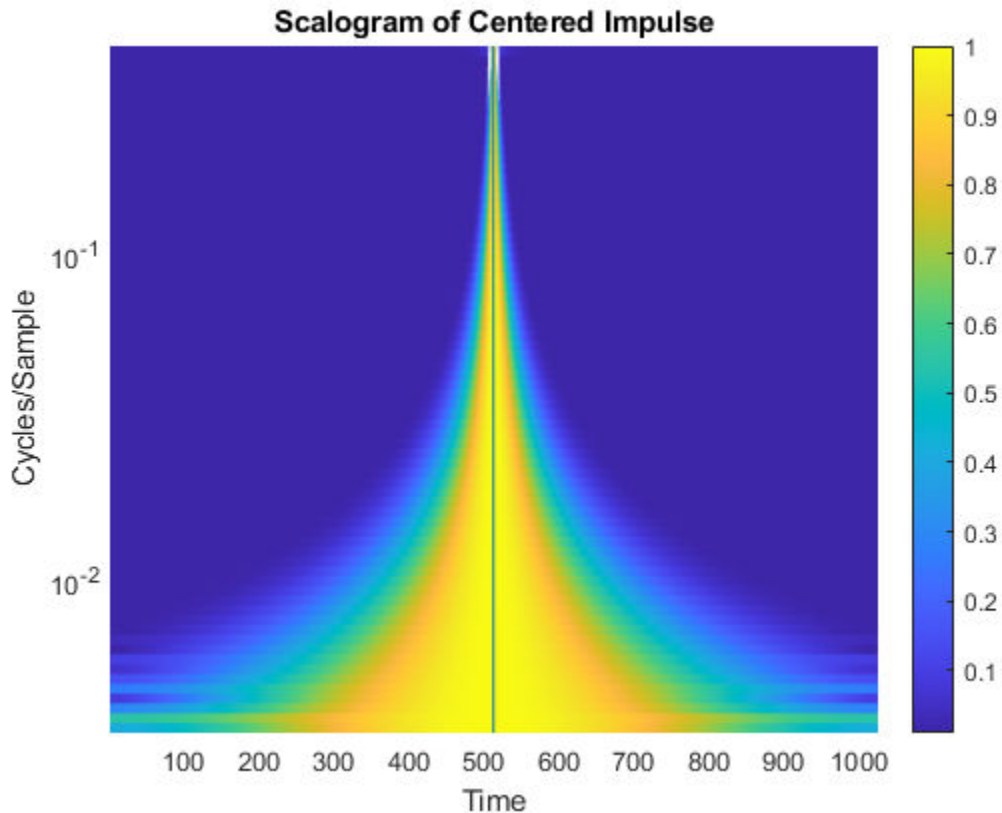
CWT of Centered Impulse

To begin to understand the cone of influence, create a centered impulse signal of length 1024 samples. Create a CWT filter bank using `cwtfilterbank` with default values. Use `wt` to return the CWT coefficients and frequencies of the impulse. For better visualization, normalize the CWT coefficients so that the maximum absolute value at each frequency (for each scale) is equal to 1.

```
x = zeros(1024,1);
x(512) = 1;
fb = cwtfilterbank;
[cfs,f] = wt(fb,x);
cfs = cfs./max(cfs,[],2);
```

Use the helper function `helperPlotScalogram` to the scalogram. The code for `helperPlotFunction` is at the end of this example. Mark the location of the impulse with a line.

```
ax = helperPlotScalogram(f,cfs);
hl = line(ax,[512 512],[min(f) max(f)],...
         [max(abs(cfs(:))) max(abs(cfs(:)))]);
title('Scalogram of Centered Impulse')
```



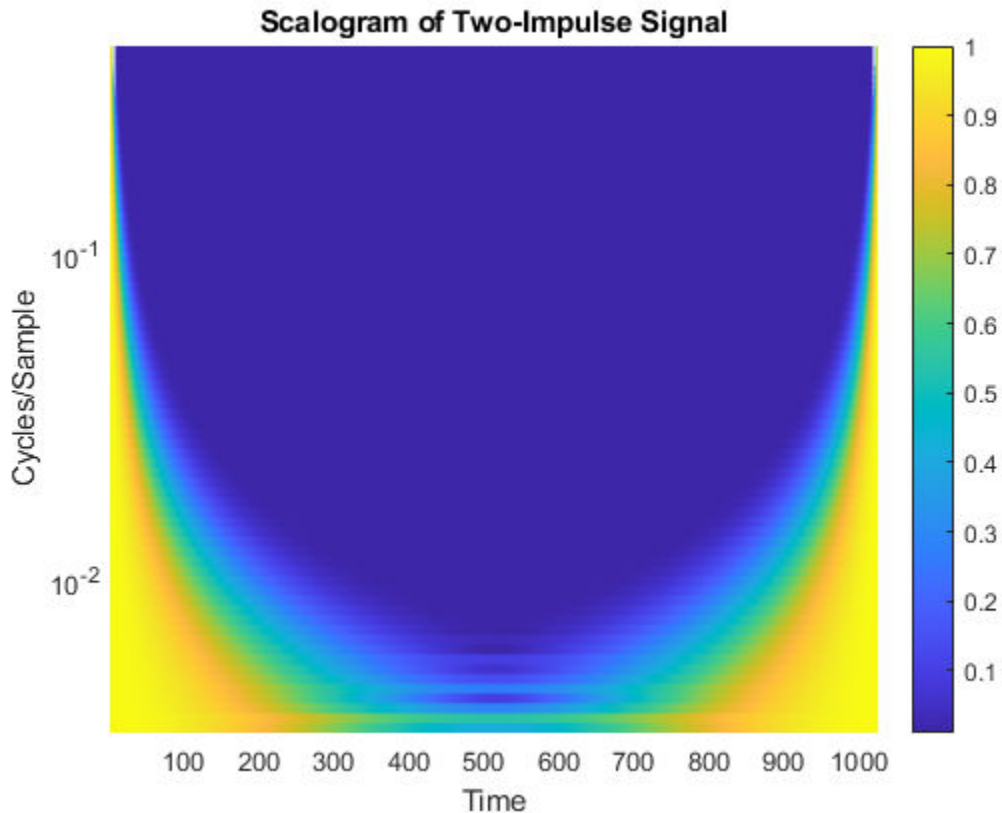
The solid black line shows the location of the impulse in time. Note that as the frequency decreases, the width of the CWT coefficients in time that are nonzero and centered on the impulse increases. Conversely, as the frequency increases, the width of the CWT coefficients that are nonzero decreases and becomes increasingly centered on the impulse. Low frequencies correspond to wavelets of longer scale, while higher frequencies correspond to wavelets of shorter scale. The effect of the impulse persists longer in time with longer wavelets. In other words, the longer the wavelet, the longer the duration of influence of the signal. For a wavelet centered at a certain point in time, stretching or shrinking the wavelet results in the wavelet "seeing" more or less of the signal. This is referred to as the wavelet's *cone of influence*.

Boundary Effects

The previous section illustrates the cone of influence for an impulse in the center of the observation, or data interval. But what happens when the wavelets are located near the beginning or end of the data? In the wavelet transform, we not only dilate the wavelet, but also translate it in time. Wavelets near the beginning or end of the data inevitably "see" data outside the observation interval. Various techniques are used to compensate for the fact that the wavelet coefficients near the beginning and end of the data are affected by the wavelets extending outside the boundary. The `cwtfilterbank` and `cwt` functions offer the option to treat the boundaries by reflecting the signal symmetrically or periodically extending it. However, regardless of which technique is used, you should exercise caution when interpreting wavelet coefficients near the boundaries because the wavelet coefficients are affected by values outside the extent of the signal under consideration. Further, the extent of the wavelet coefficients affected by data outside the observation interval depends on the scale (frequency). The longer the scale, the larger the cone of influence.

Repeat the impulse example, but place two impulses, one at the beginning of the data and one at the end. Also return the cone of influence. For better visualization, normalize the CWT coefficients so that the maximum absolute value at each frequency (for each scale) is equal to 1.

```
dirac = zeros(1024,1);  
dirac([1 1024]) = 1;  
[cfs,f,coi] = wt(fb,dirac);  
cfs = cfs./max(cfs,[],2);  
helperPlotScalogram(f,cfs)  
title('Scalogram of Two-Impulse Signal')
```



Here it is clear that the cone of influence for the extreme boundaries of the observation interval extends into the interval to a degree that depends on the scale of the wavelet. Therefore, wavelet coefficients well inside the observation interval can be affected by what data the wavelet sees at the boundaries of the signal, or even before the signal's actual boundaries if you extend the signal in some way.

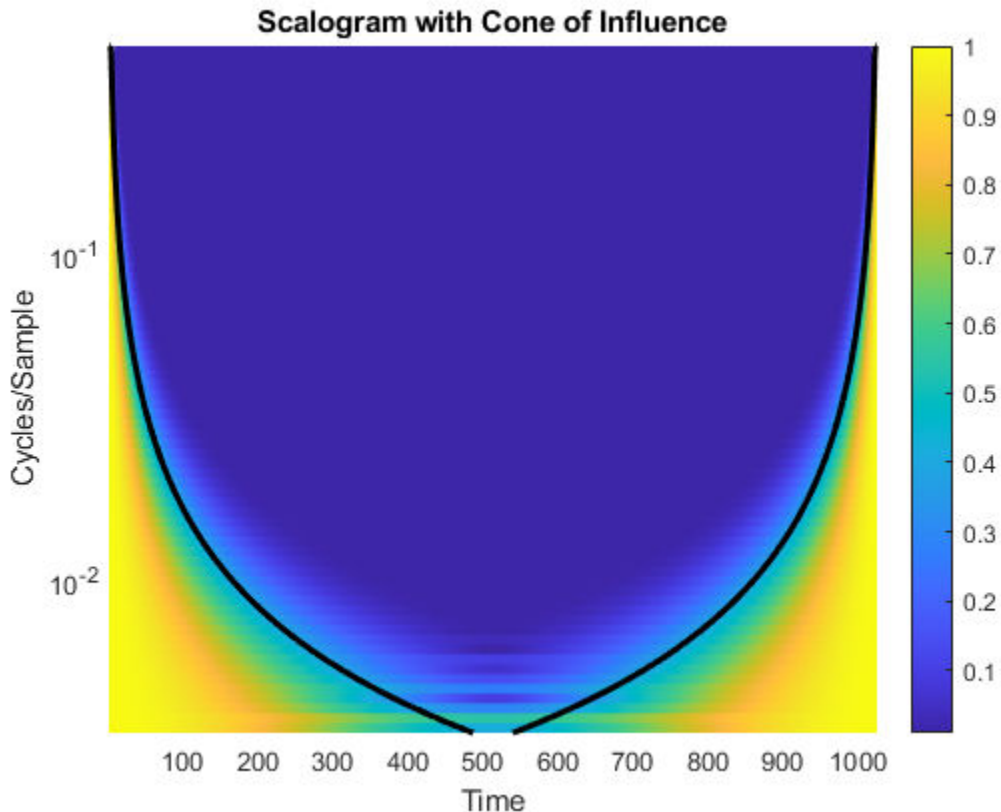
In the previous figure, you should already see a striking similarity between the cone of influence returned by `cwtfilterbank` or plotted by the `cwt` function and areas where the scalogram coefficients for the two-impulse signal are nonzero.

While it is important to understand these boundary effects on the interpretation of wavelet coefficients, there is no mathematically precise rule to determine the extent of the cone of influence at each scale. Nobach et al. [2] define the extent of the cone of

influence at each scale as the point where the wavelet transform magnitude decays to 2% of its peak value. Because many of the wavelets used in continuous wavelet analysis decay exponentially in time, Torrence and Compo [3] use the time constant $1/e$ to delineate the borders of the cone of influence at each scale. For Morse wavelets, Lilly [1] uses the concept of the "wavelet footprint," which is the time interval that encompasses approximately 95% of the wavelet's energy. Lilly delineates the COI by adding $1/2$ the wavelet footprint to the beginning of the observation interval and subtracting $1/2$ the footprint from the end of the interval at each scale.

The `cwtfilterbank` and `cwt` functions use an approximation to the $1/e$ rule to delineate the COI. The approximation involves adding one time-domain standard deviation at each scale to the beginning of the observation interval and subtracting one time-domain standard deviation at each scale from the end of the interval. Before we demonstrate this correspondence, add the computed COI to the previous plot.

```
helperPlotScalogram(f,cfs,coi)
title('Scalogram with Cone of Influence')
```



You see that the computed COI is a good approximation to boundaries of the significant effects of an impulse at the beginning and end of the signal.

To show how `cwtfilterbank` and `cwt` compute this rule explicitly, consider two examples, one for the analytic Morlet wavelet and one for the default Morse wavelet. Begin with the analytic Morlet wavelet, where our one time-domain standard deviation rule agrees exactly with the expression of the folding time used by Torrence and Compo [3].

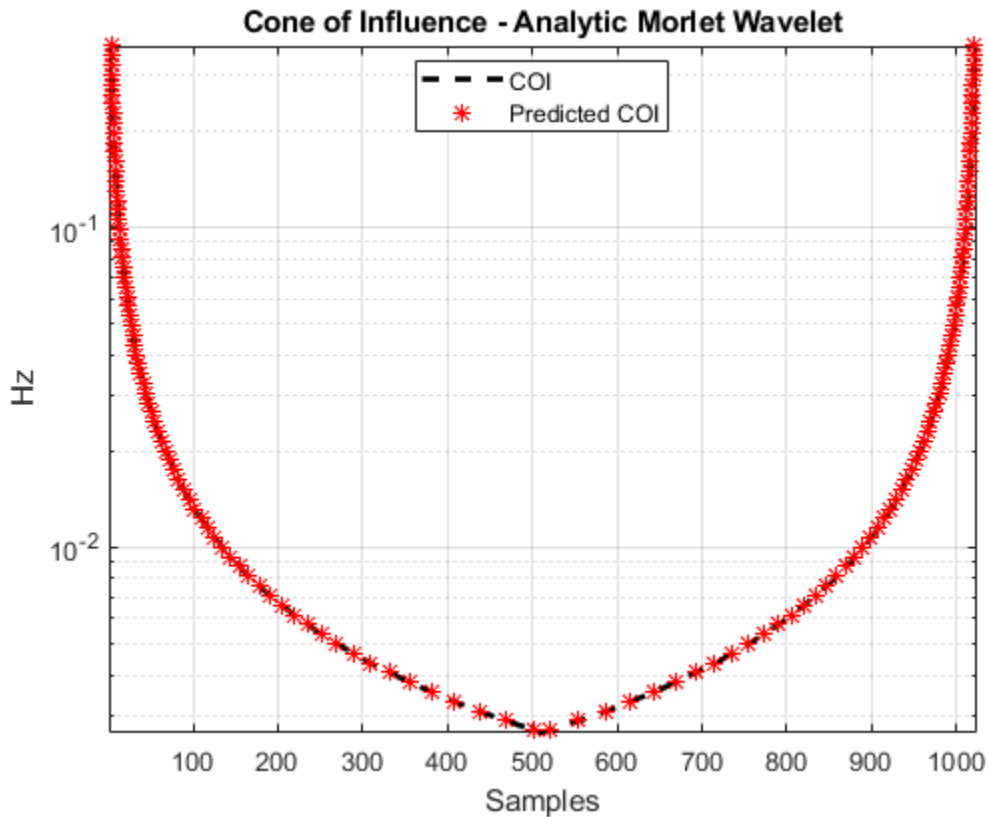
```
fb = cwtfilterbank('Wavelet','amor');
[~,f,coi] = wt(fb,dirac);
```

The expression for the COI in Torrence and Compo is $\sqrt{2}s$ where s is the scale. For the analytic Morlet wavelet in `cwtfilterbank` and `cwt`, this is given by:

```
cf = 6/(2*pi);  
predtimes = sqrt(2)*cf./f;
```

Plot the COI returned by `cwtfilterbank` along with the expression used in Torrence and Compo.

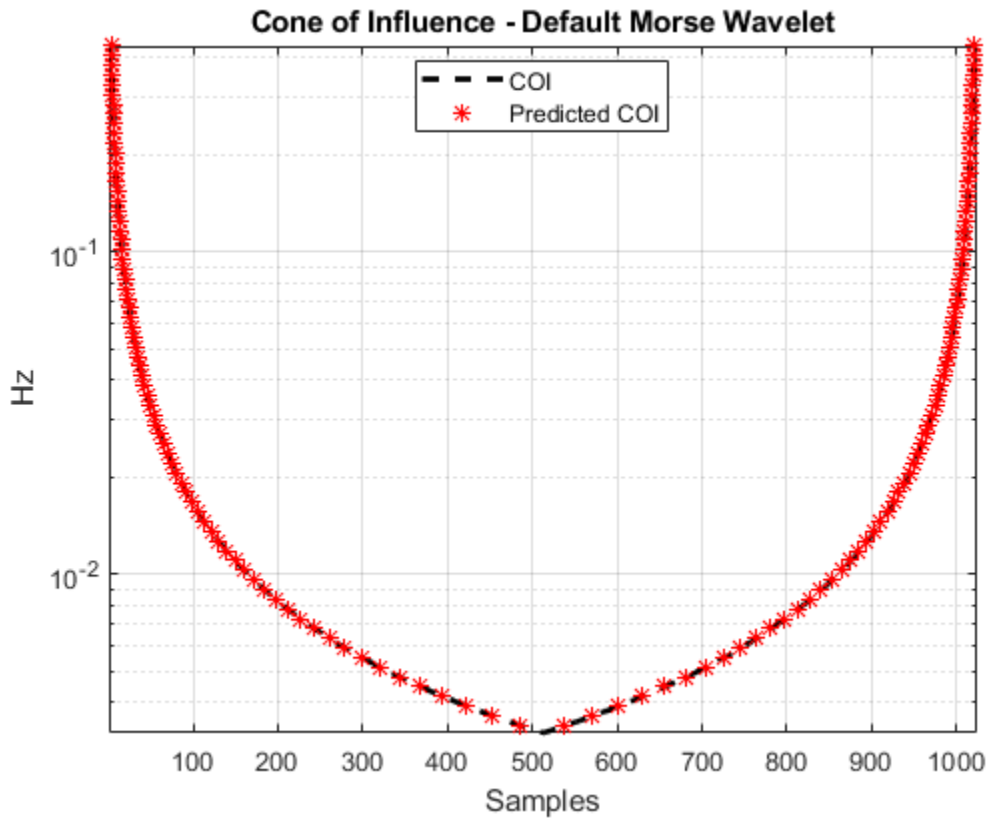
```
plot(1:1024,coi,'k--','linewidth',2)  
hold on  
grid on  
plot(predtimes,f,'r*')  
plot(1024-predtimes,f,'r*')  
set(gca,'yscale','log')  
axis tight  
legend('COI','Predicted COI','Location','best')  
xlabel('Samples')  
ylabel('Hz')  
title('Cone of Influence - Analytic Morlet Wavelet')
```

The last example shows the same correspondence for the default Morse wavelet in `cwtfilterbank` and `cwt`. The time-domain standard deviation of the default Morse wavelet is 5.5008, and the peak frequency is 0.2995 cycles/sample. Use the center frequencies of the wavelet bandpass filters as well as the time-domain standard deviation rule to obtain the predicted COI and compare against the values returned by `cwtfilterbank`.

```
fb = cwtfilterbank;
[~,f,coi] = wt(fb,dirac);
sd = 5.5008;
cf = 0.2995;
predtimes = cf./f*sd;
figure
plot(1:1024,coi,'k--','linewidth',2)
```

```
hold on
grid on
plot(predtimes,f,'r*')
plot(1024-predtimes,f,'r*')
set(gca,'yscale','log')
axis tight
legend('COI','Predicted COI','Location','best')
xlabel('Samples')
ylabel('Hz')
title('Cone of Influence - Default Morse Wavelet')
```



Appendix

The following helper function is used in this example.

helperPlotScalogram

```
function varargout = helperPlotScalogram(f,cfs,coi)
nargoutchk(0,1);
ax = newplot;
surf(ax,1:1024,f,abs(cfs), 'EdgeColor', 'none')
ax.YScale = 'log';
caxis([0.01 1])
colorbar
grid on
ax.YLim = [min(f) max(f)];
ax.XLim = [1 size(cfs,2)];
view(0,90)

xlabel('Time')
ylabel('Cycles/Sample')

if nargin == 3
    hl = line(ax,1:1024,coi,ones(1024,1));
    hl.Color = 'k';
    hl.LineWidth = 2;
end

if nargout > 0
    varargout{1} = ax;
end

end
```

References

- [1] Lilly, J. M. "Element analysis: a wavelet-based method for analysing time-localized events in noisy time series." *Proceedings of the Royal Society A*. Volume 473: 20160776, 2017, pp. 1-28. [dx.doi.org/10.1098/rspa.2016.0776](https://doi.org/10.1098/rspa.2016.0776).
- [2] Nobach, H., Tropea, C., Cordier, L., Bonnet, J. P., Delville, J., Lewalle, J., Farge, M., Schneider, K., and R. J. Adrian. "Review of Some Fundamentals of Data Processing." *Springer Handbook of Experimental Fluid Mechanics* (C. Tropea, A. L. Yarin, and J. F. Foss, eds.). Berlin, Heidelberg: Springer, 2007, pp. 1337-1398.
- [3] Torrence, C., and G. Compo. "A Practical Guide to Wavelet Analysis." *Bulletin of the American Meteorological Society*. Vol. 79, Number 1, 1998, pp. 61-78.

See Also

cwt | cwtfilterbank

More About

- “Morse Wavelets” on page 2-21

Time-Frequency Analysis and Continuous Wavelet Transform

This example shows how the variable time-frequency resolution of the continuous wavelet transform can help you obtain a sharp time-frequency representation.

The continuous wavelet transform (CWT) is a time-frequency transform, which is ideal for analyzing nonstationary signals. A signal being nonstationary means that its frequency-domain representation changes over time. Many signals are nonstationary, such as electrocardiograms, audio signals, earthquake data, and climate data.

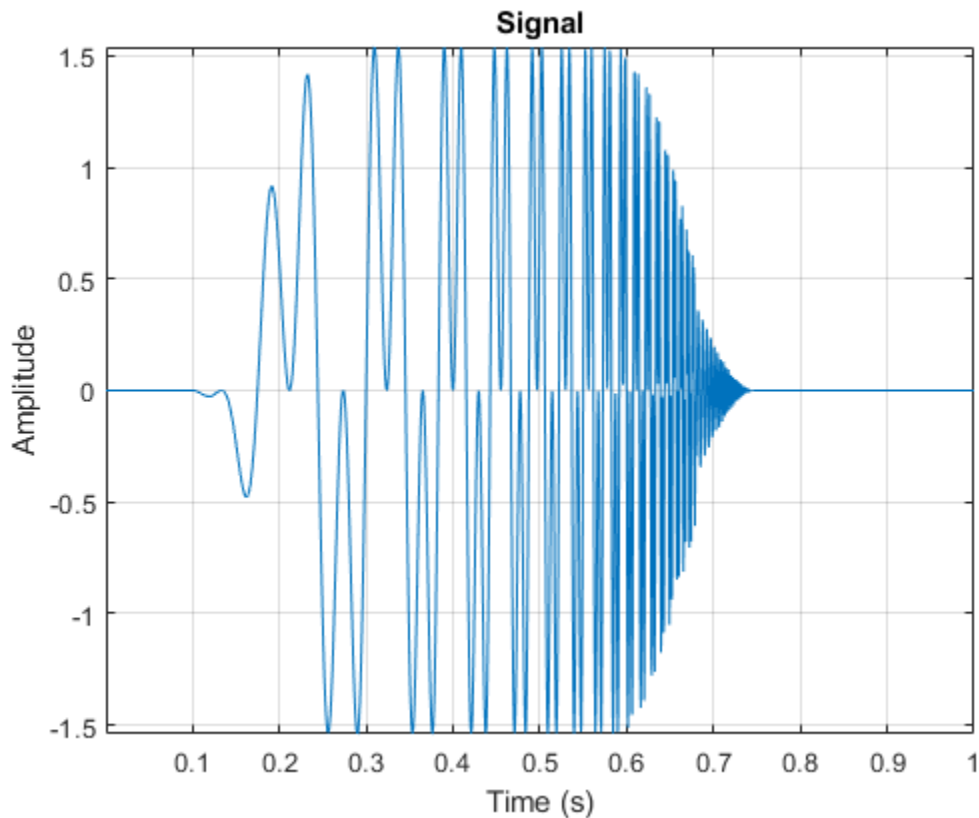
Load Hyperbolic Chirp

Load a signal that has two hyperbolic chirps. The data are sampled at 2048 Hz. The first chirp is active between 0.1 and 0.68 seconds, and the second chirp is active between 0.1 and 0.75 seconds. The instantaneous frequency (in hertz) of the first chirp at time t is

$\frac{15\pi}{(0.8 - t)^2}/2\pi$. The instantaneous frequency of the second chirp at time t is

$\frac{5\pi}{(0.8 - t)^2}/2\pi$. Plot the signal.

```
load hychirp
plot(t,hychirp)
grid on
title('Signal')
axis tight
xlabel('Time (s)')
ylabel('Amplitude')
```



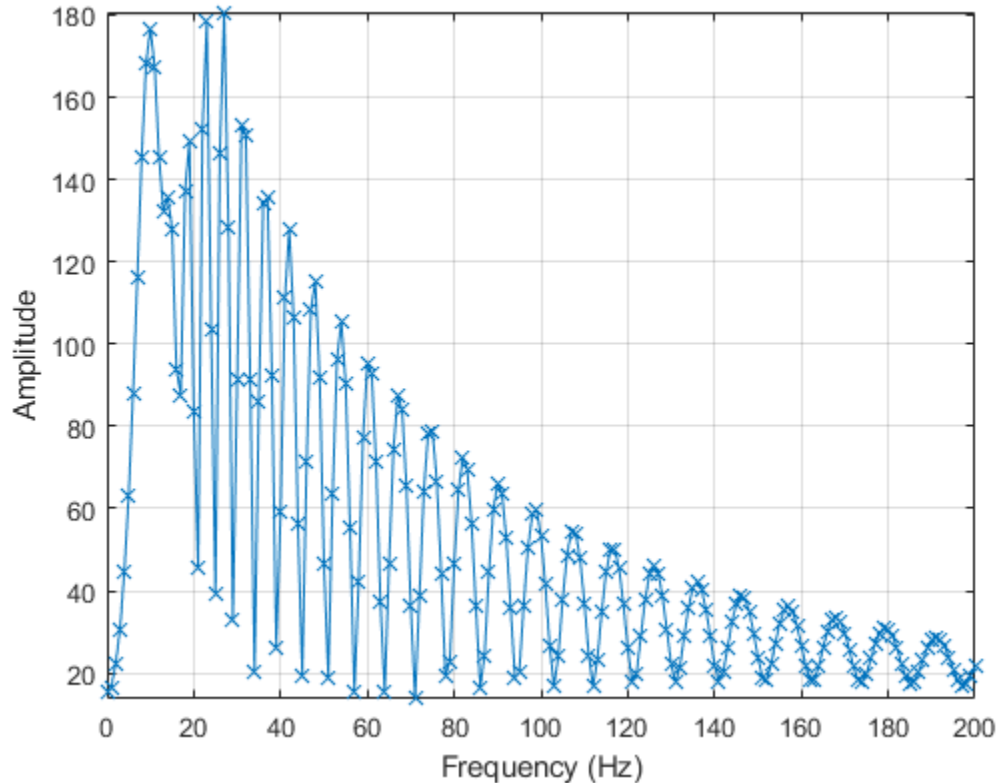
Time-Frequency Analysis: Fourier Transform

The Fourier transform (FT) is very good at identifying frequency components present in a signal. However, the FT does not identify when the frequency components occur.

Plot the magnitude spectrum of the signal. Zoom in on the region between 0 and 200 Hz.

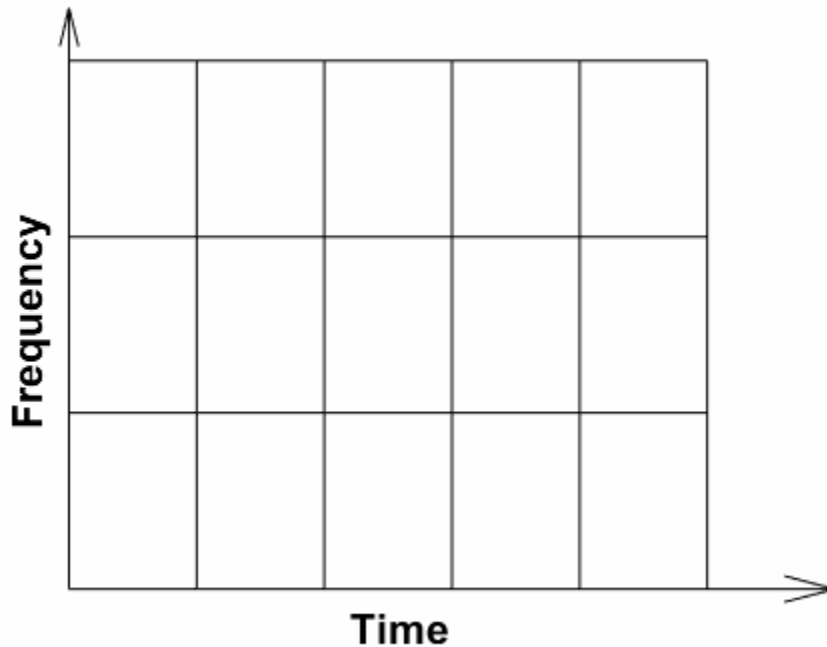
```
sigLen = numel(hychirp);  
fchirp = fft(hychirp);  
fr = Fs*(0:1/Fs:1-1/Fs);  
plot(fr(1:sigLen/2),abs(fchirp(1:sigLen/2)),'x-')  
xlabel('Frequency (Hz)')  
ylabel('Amplitude')  
axis tight
```

```
grid on  
xlim([0 200])
```



Time-Frequency Analysis: Short-Time Fourier Transform

The Fourier transform does not provide time information. To determine when the changes in frequency occur, the short-time Fourier transform (STFT) approach segments the signal into different chunks and performs the FT on each chunk. The STFT tiling in the time-frequency plane is shown here.



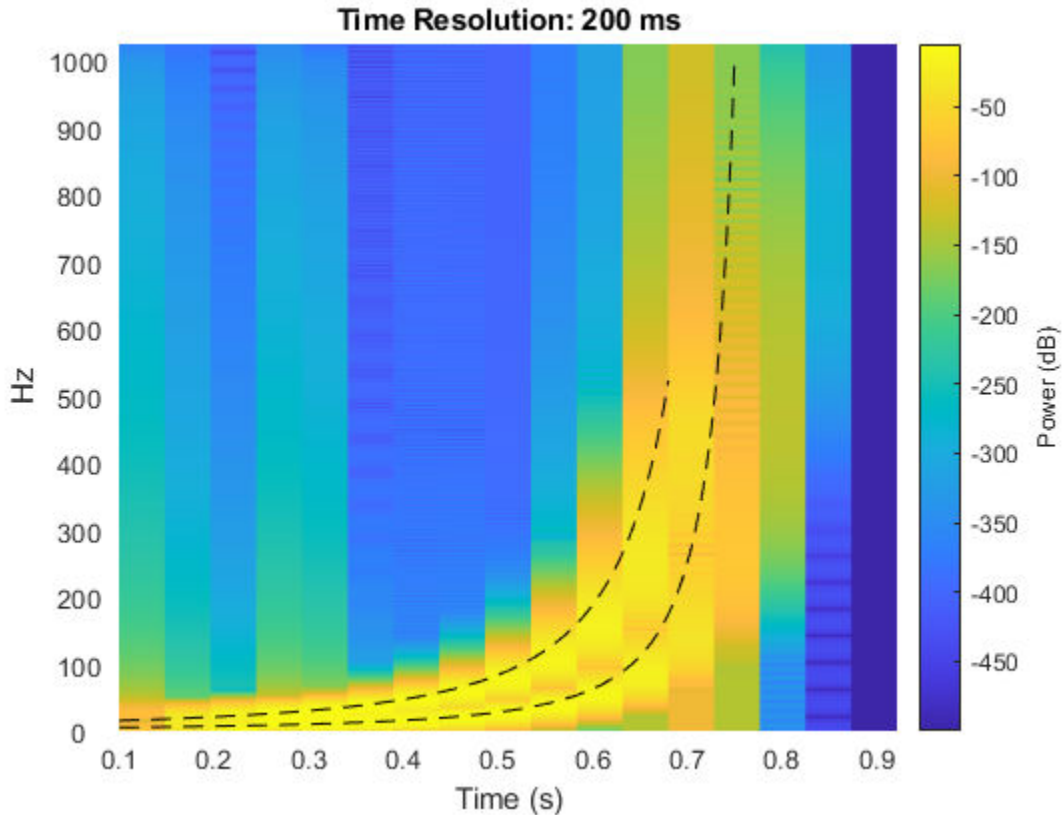
The STFT provides some information on both the timing and the frequencies at which a signal event occurs. However, choosing a window (segment) size is key. For time-frequency analysis using the STFT, choosing a shorter window size helps obtain good time resolution at the expense of frequency resolution. Conversely, choosing a larger window helps obtain good frequency resolution at the expense of time resolution.

Once you pick a window size, it remains fixed for the entire analysis. If you can estimate the frequency components you are expecting in your signal, then you can use that information to pick a window size for the analysis.

The instantaneous frequencies of the two chirps at their initial time points are approximately 5 Hz and 15 Hz. Use the helper function `helperPlotSpectrogram` to plot the spectrogram of the signal with a time window size of 200 milliseconds. The

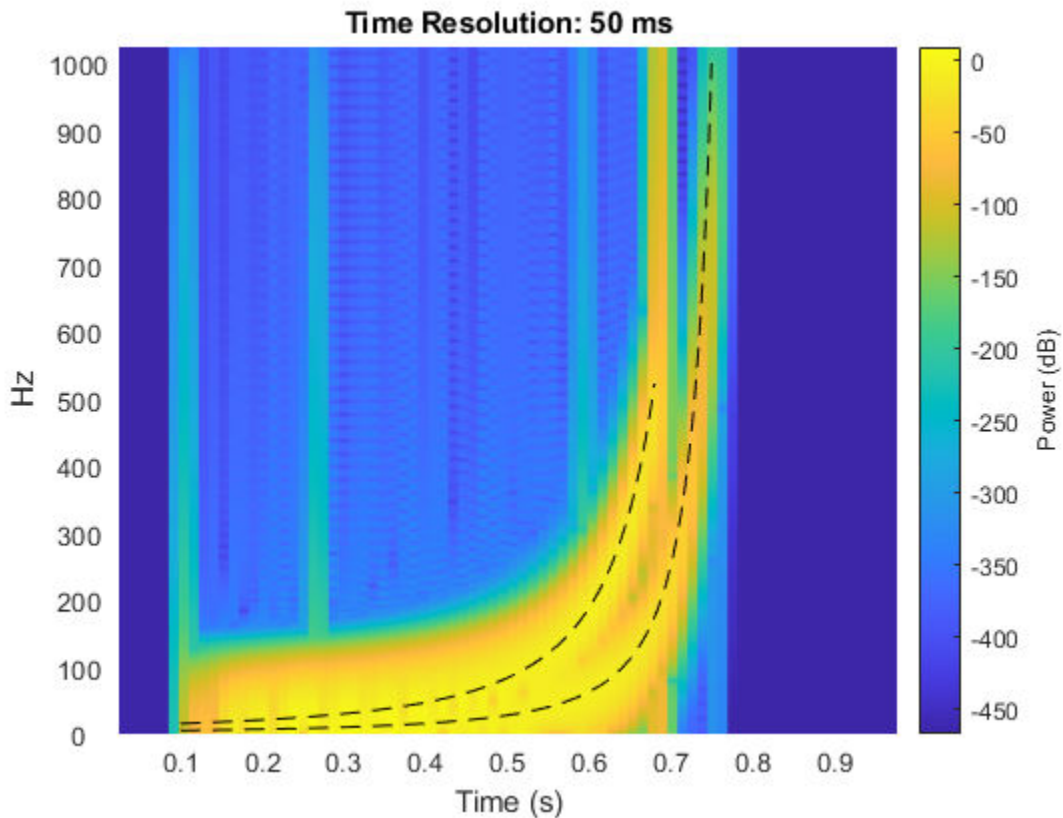
source code for `helperPlotSpectrogram` is listed in the appendix. The helper function plots the instantaneous frequencies over the spectrogram as black dashed-line segments. The instantaneous frequencies are resolved early in the signal, but not as well later.

```
helperPlotSpectrogram(hychirp,t,Fs,200)
```



Now use `helperPlotSpectrogram` to plot the spectrogram with a time window size of 50 milliseconds. The higher frequencies, which occur later in the signal, are now resolved, but the lower frequencies at the beginning of the signal are not.

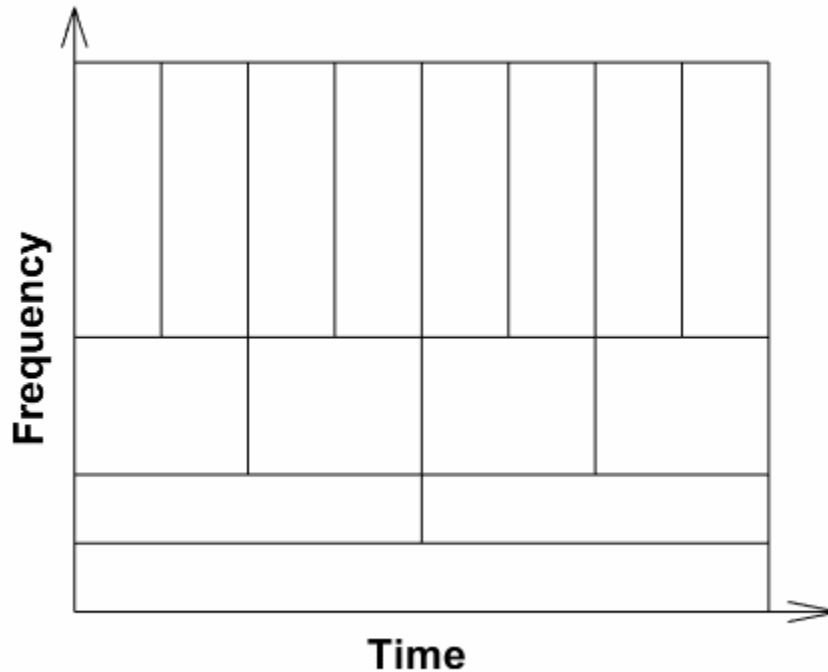
```
helperPlotSpectrogram(hychirp,t,Fs,50)
```



For nonstationary signals like the hyperbolic chirp, using the STFT is problematic. No single window size can resolve the entire frequency content of such signals.

Time-Frequency Analysis: Continuous Wavelet Transform

The continuous wavelet transform (CWT) was created to overcome the resolution issues inherent in the STFT. The CWT tiling on the time-frequency plane is shown here.

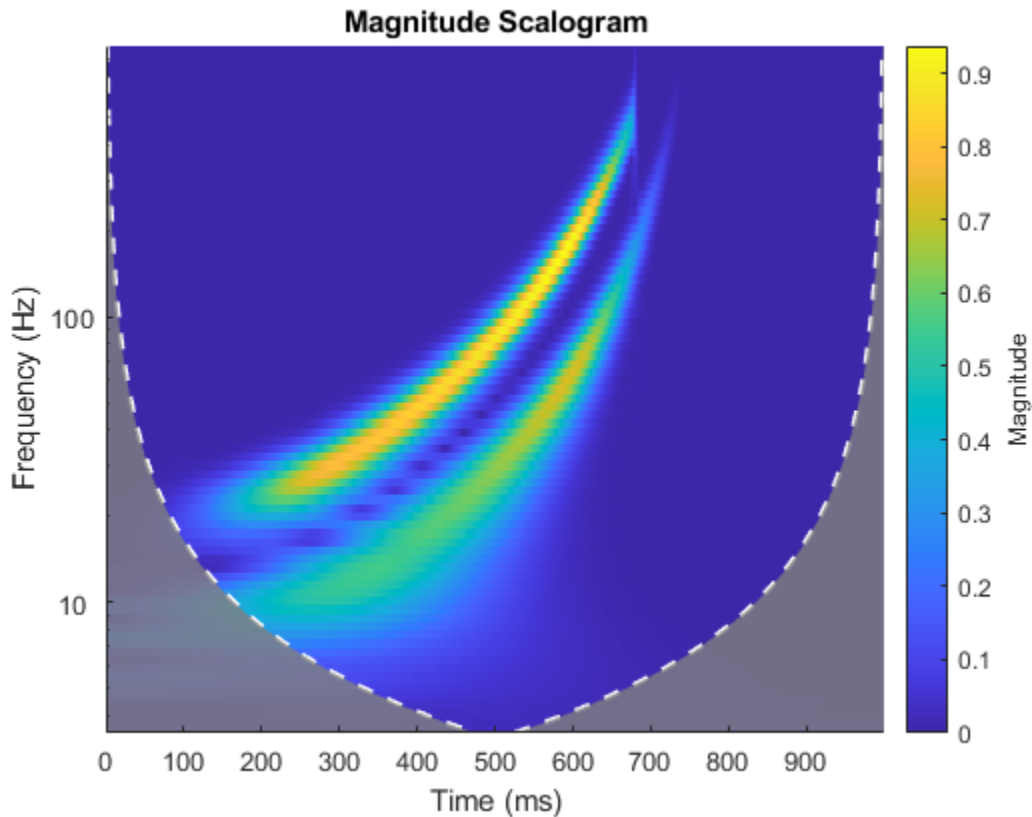


The CWT tiling of the plane is useful because many real-world signals have slowly oscillating content that occurs on long scales, while high frequency events tend to be abrupt or transient. However, if it were natural for high-frequency events to be long in duration, then using the CWT would not be appropriate. You would have poorer frequency resolution without gaining any time resolution. But that is quite often not the case. The human auditory system works this way; we have much better frequency localization at lower frequencies, and better time localization at high frequencies.

Plot the scalogram of the CWT. The scalogram is the absolute value of the CWT plotted as a function of time and frequency. The plot uses a logarithmic frequency axis because frequencies in the CWT are logarithmic. The presence of the two hyperbolic chirps in the signal is clear from the scalogram. With the CWT, you can accurately estimate the

instantaneous frequencies throughout the duration of the signal, without worrying about picking a segment length.

```
cwt(hychirp,Fs)
```

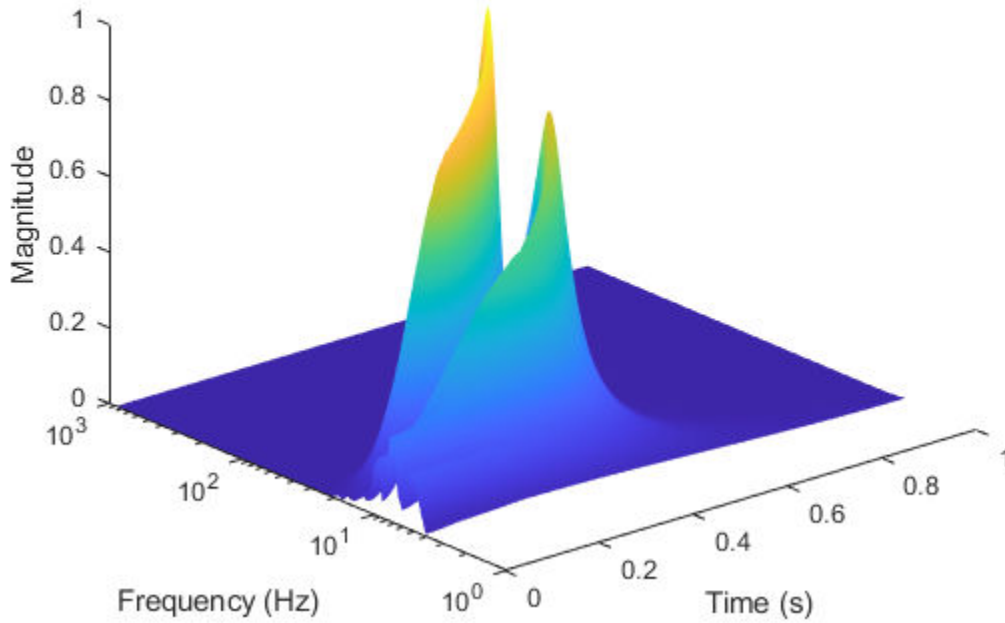


The white dashed line marks what is known as the *cone of influence*. The cone of influence shows areas in the scalogram potentially affected by boundary effects. For more information, see “Boundary Effects and the Cone of Influence” on page 2-32.

To get a sense of how rapidly the magnitude of the wavelet coefficients grows, use the helper function `helperPlotScalogram3d` to plot the scalogram as a 3-D surface. The source code for `helperPlotScalogram3d` is listed in the appendix.

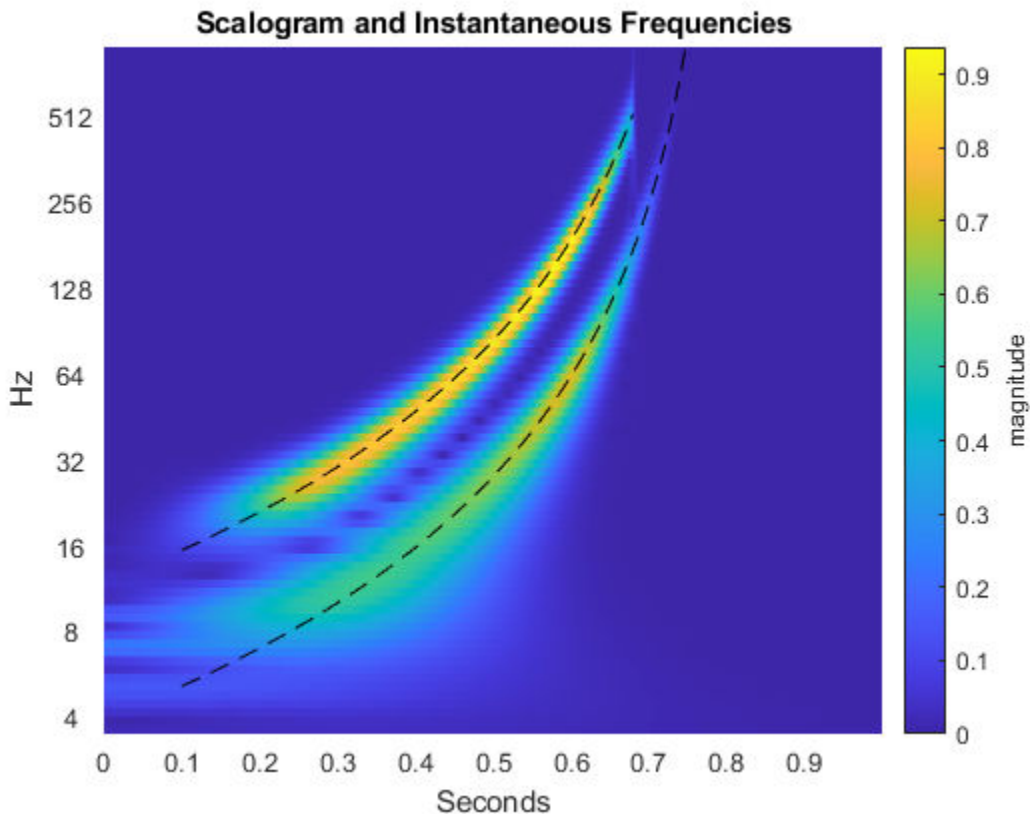
```
helperPlotScalogram3d(hychirp,Fs)
```

Scalogram In 3-D



Use the helper function `helperPlotScalogram` to plot the scalogram of the signal and the instantaneous frequencies. The source code for `helperPlotScalogram` is listed in the appendix. The instantaneous frequencies align well with the scalogram features.

```
helperPlotScalogram(hychirp,Fs)
```



Appendix - Helper Functions

helperPlotSpectrogram

```
function helperPlotSpectrogram(sig,t,Fs,timeres)
% This function is only intended to support this wavelet example.
% It may change or be removed in a future release.

[px,fx,tx] = pspectrum(sig,Fs,'spectrogram','TimeResolution',timeres/1000);
hp = pcolor(tx,fx,20*log10(abs(px)));
hp.EdgeAlpha = 0;
ylims = hp.Parent.YLim;
yticks = hp.Parent.YTick;
cl = colorbar;
```

```

cl.Label.String = 'Power (dB)';
axis tight
hold on
title(['Time Resolution: ',num2str(timeres),' ms'])
xlabel('Time (s)')
ylabel('Hz');
dt = 1/Fs;
idxbegin = round(0.1/dt);
idxend1 = round(0.68/dt);
idxend2 = round(0.75/dt);
instfreq1 = abs((15*pi)./(0.8-t).^2)./(2*pi);
instfreq2 = abs((5*pi)./(0.8-t).^2)./(2*pi);
plot(t(idxbegin:idxend1),(instfreq1(idxbegin:idxend1)),'k--');
hold on;
plot(t(idxbegin:idxend2),(instfreq2(idxbegin:idxend2)),'k--');
ylim(ylims);
hp.Parent.YTick = yticks;
hp.Parent.YTickLabels = yticks;
hold off
end

```

helperPlotScalogram

```

function helperPlotScalogram(sig,Fs)
% This function is only intended to support this wavelet example.
% It may change or be removed in a future release.
[cfs,f] = cwt(sig,Fs);

sigLen = numel(sig);
t = (0:sigLen-1)/Fs;

hp = pcolor(t,log2(f),abs(cfs));
hp.EdgeAlpha = 0;
ylims = hp.Parent.YLim;
yticks = hp.Parent.YTick;
cl = colorbar;
cl.Label.String = 'magnitude';
axis tight
hold on
title('Scalogram and Instantaneous Frequencies')
xlabel('Seconds');
ylabel('Hz');
dt = 1/2048;
idxbegin = round(0.1/dt);
idxend1 = round(0.68/dt);

```

```
idxend2 = round(0.75/dt);
instfreq1 = abs((15*pi)./(0.8-t).^2)./(2*pi);
instfreq2 = abs((5*pi)./(0.8-t).^2)./(2*pi);
plot(t(idxbegin:idxend1),log2(instfreq1(idxbegin:idxend1)),'k--');
hold on;
plot(t(idxbegin:idxend2),log2(instfreq2(idxbegin:idxend2)),'k--');
ylim(ylims);
hp.Parent.YTick = yticks;
hp.Parent.YTickLabels = 2.^yticks;
end
```

helperPlotScalogram3d

```
function helperPlotScalogram3d(sig,Fs)
% This function is only intended to support this wavelet example.
% It may change or be removed in a future release.
figure
[cfs,f] = cwt(sig,Fs);

sigLen = numel(sig);
t = (0:sigLen-1)/Fs;
surface(t,f,abs(cfs));
xlabel('Time (s)')
ylabel('Frequency (Hz)')
zlabel('Magnitude')
title('Scalogram In 3-D')
set(gca,'yscale','log')
shading interp
view([-40 30])
end
```

See Also

[cwt](#) | [cwtfilterbank](#) | [waveletScattering](#) | [waveletScattering2](#)

More About

- “Boundary Effects and the Cone of Influence” on page 2-32
- “Classify Time Series Using Wavelet Analysis and Deep Learning”
- “Wavelet Time Scattering Classification of Phonocardiogram Data”

Continuous Wavelet Analysis of Modulated Signals

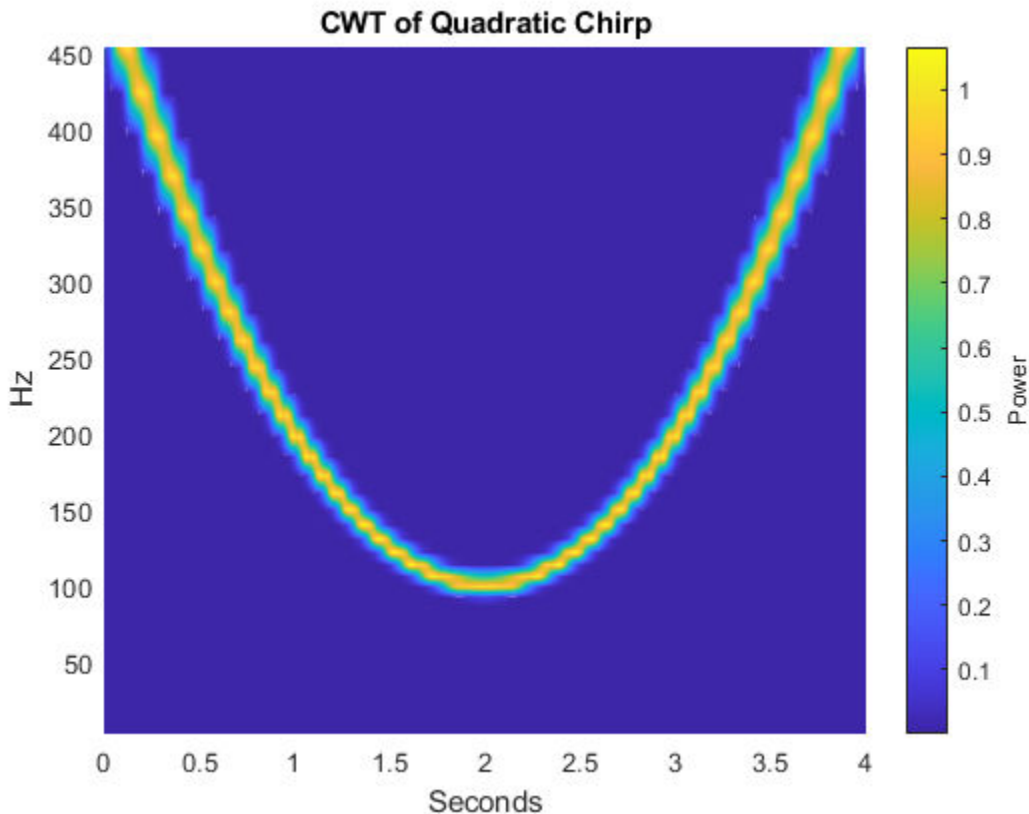
This example shows how to use the continuous wavelet transform (CWT) to analyze modulated signals.

Load a quadratic chirp signal. The signal's frequency begins at approximately 500 Hz at $t = 0$, decreases to 100 Hz at $t=2$, and increases back to 500 Hz at $t=4$. The sampling frequency is 1 kHz.

```
load quadchirp;  
fs = 1000;
```

Obtain a time-frequency plot of this signal using the CWT with a bump wavelet. The bump wavelet is a good choice for the CWT when your signals are oscillatory and you are more interested in time-frequency analysis than localization of transients.

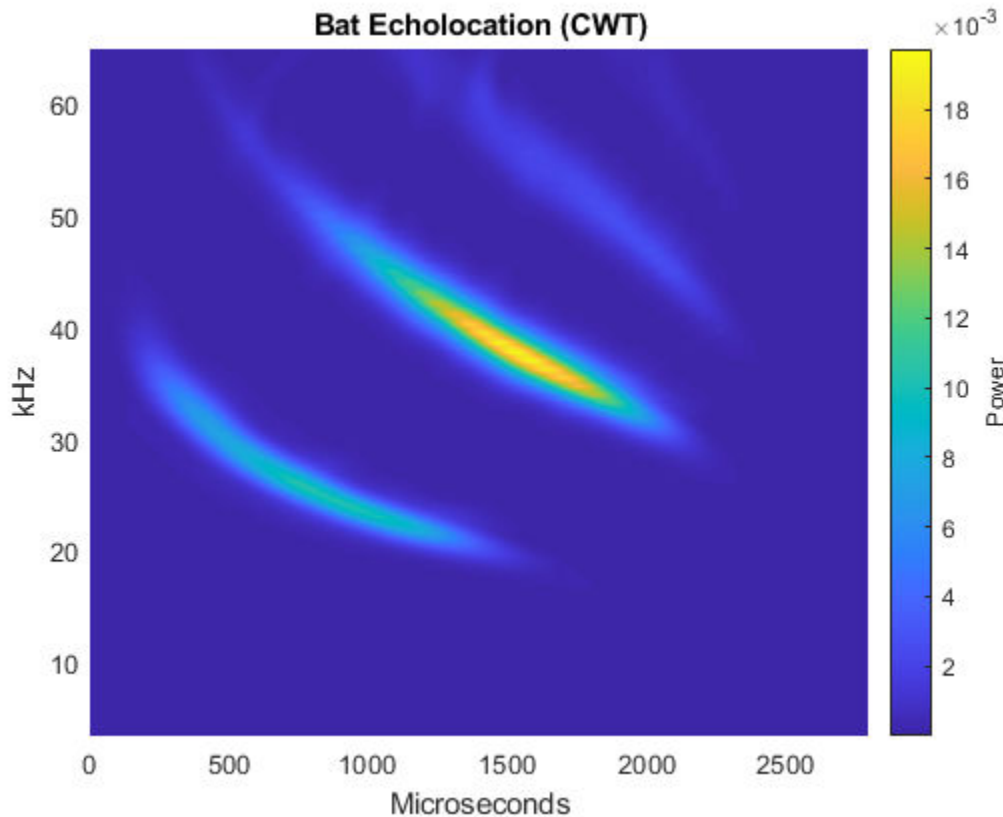
```
[cfs,f] = cwt(quadchirp,'bump',fs);  
helperCWTTimeFreqPlot(cfs,tquad,f,'surf','CWT of Quadratic Chirp','Seconds','Hz')
```



The CWT clearly shows the time evolution of the quadratic chirp's frequency. The quadratic chirp is a frequency-modulated signal. While that signal is synthetic, frequency and amplitude modulation occur frequently in natural signals as well. Use the CWT to obtain a time-frequency analysis of an echolocation pulse emitted by a big brown bat (*Eptesicus Fuscus*). The sampling interval is 7 microseconds. Use the bump wavelet with 32 voices per octave. Thanks to Curtis Condon, Ken White, and Al Feng of the Beckman Center at the University of Illinois for the bat data and permission to use it in this example.

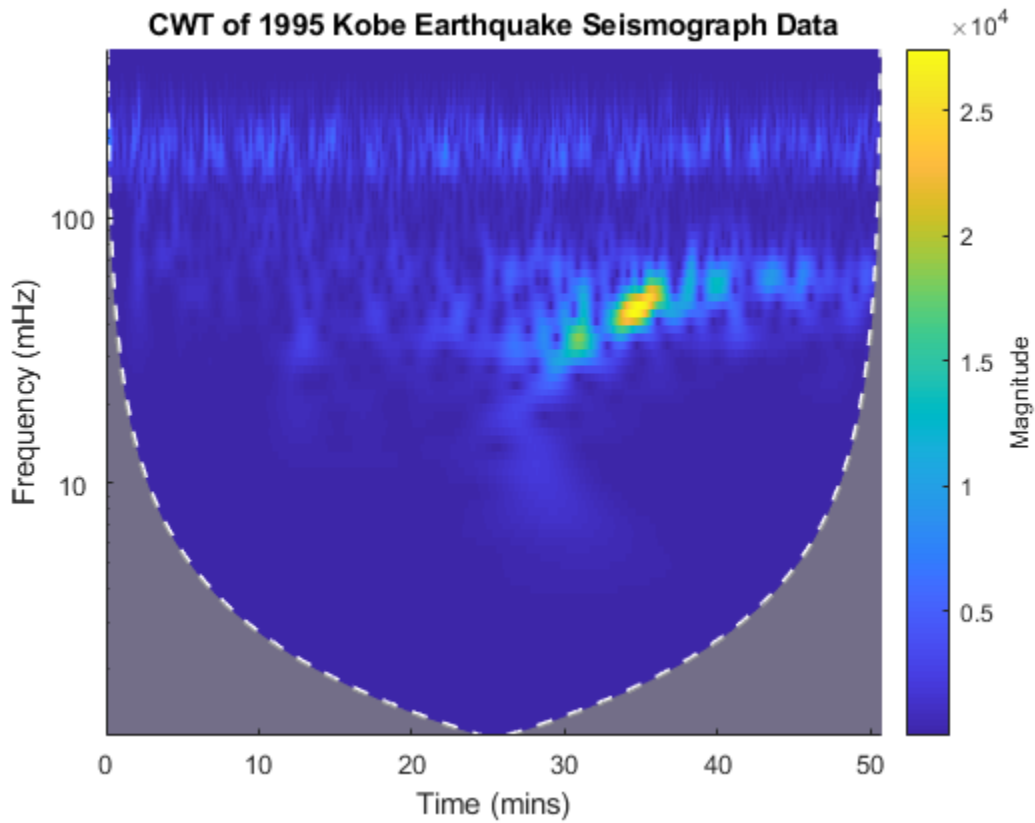
```
load batsignal
t = 0:DT:(numel(batsignal)*DT)-DT;
[cfs,f] = cwt(batsignal,'bump',1/DT,'VoicesPerOctave',32);
```

```
helperCWTTimeFreqPlot(cfs,t.*1e6,f./1e3,'surf','Bat Echolocation (CWT)',...
    'Microseconds','kHz')
```



For the final example, obtain a time-frequency analysis of some seismograph data recorded during the 1995 Kobe earthquake. The data are seismograph (vertical acceleration, nm/sq.sec) measurements recorded at Tasmania University, Hobart, Australia on 16 January 1995 beginning at 20:56:51 (GMT) and continuing for 51 minutes at 1 second intervals. Use the default analytic Morse wavelet.

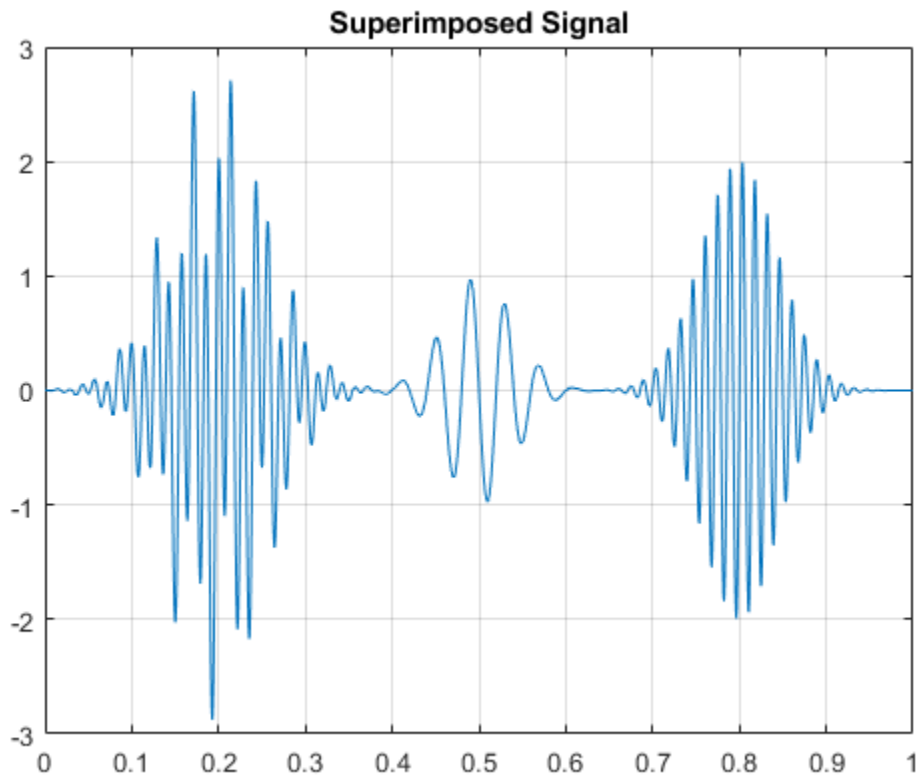
```
load kobe;
dt = 1;
cwt(kobe,1);
title('CWT of 1995 Kobe Earthquake Seismograph Data');
```



Remove Time-Localized Frequency Components

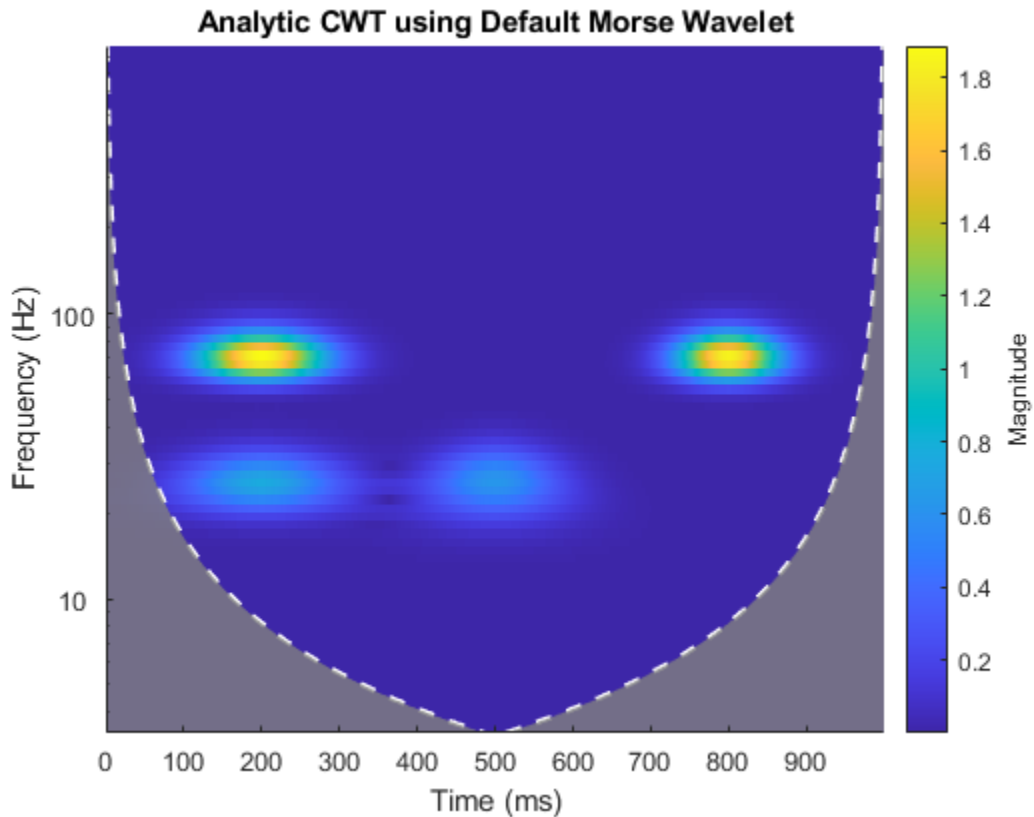
Create a signal consisting of exponentially weighted sine waves. The signal has two 25-Hz components -- one centered at 0.2 seconds and one centered at 0.5 seconds. It also has two 70-Hz components -- one centered at 0.2 and one centered at 0.8 seconds. The first 25-Hz and 70-Hz components co-occur in time.

```
t = 0:1/2000:1-1/2000;
dt = 1/2000;
x1 = sin(50*pi*t).*exp(-50*pi*(t-0.2).^2);
x2 = sin(50*pi*t).*exp(-100*pi*(t-0.5).^2);
x3 = 2*cos(140*pi*t).*exp(-50*pi*(t-0.2).^2);
x4 = 2*sin(140*pi*t).*exp(-80*pi*(t-0.8).^2);
x = x1+x2+x3+x4;
plot(t,x)
grid on;
title('Superimposed Signal')
```



Obtain and display the CWT.

```
cwt(x,2000);  
title('Analytic CWT using Default Morse Wavelet');
```

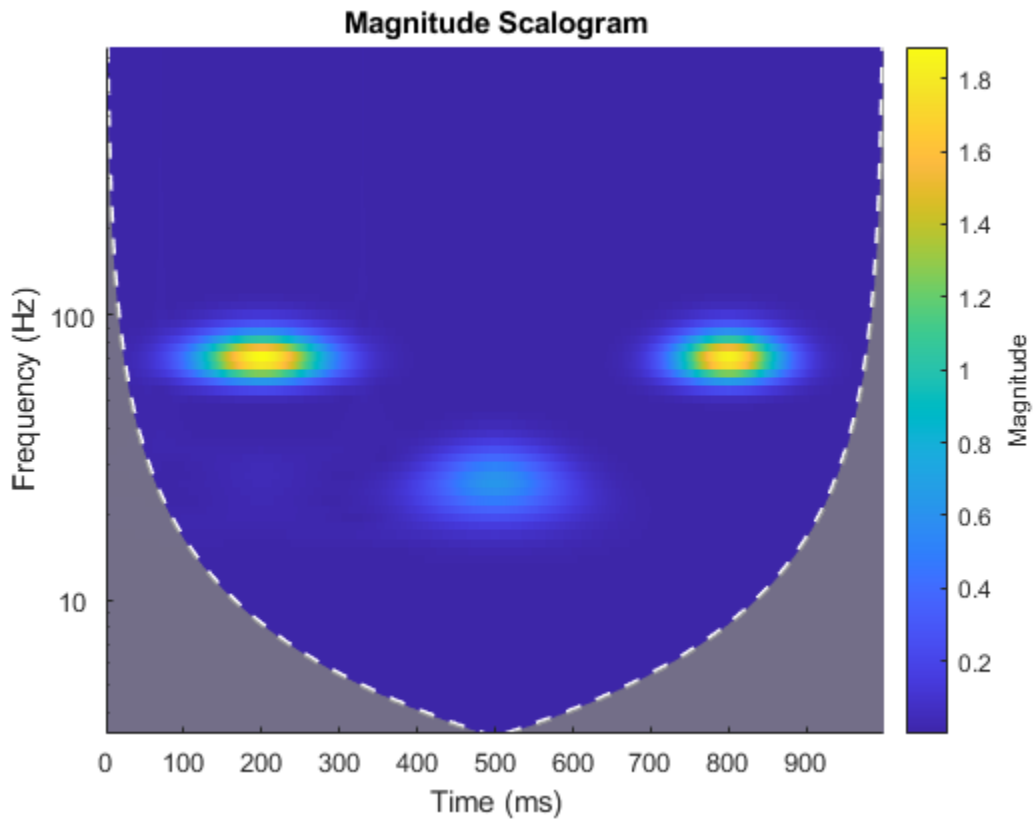


Remove the 25 Hz component which occurs from approximately 0.07 to 0.3 seconds by zeroing out the CWT coefficients. Use the inverse CWT (`icwt`) to reconstruct an approximation to the signal.

```
[cfs,f] = cwt(x,2000);
T1 = .07; T2 = .33;
F1 = 19; F2 = 34;
cfs(f > F1 & f < F2, t > T1 & t < T2) = 0;
xrec = icwt(cfs);
```

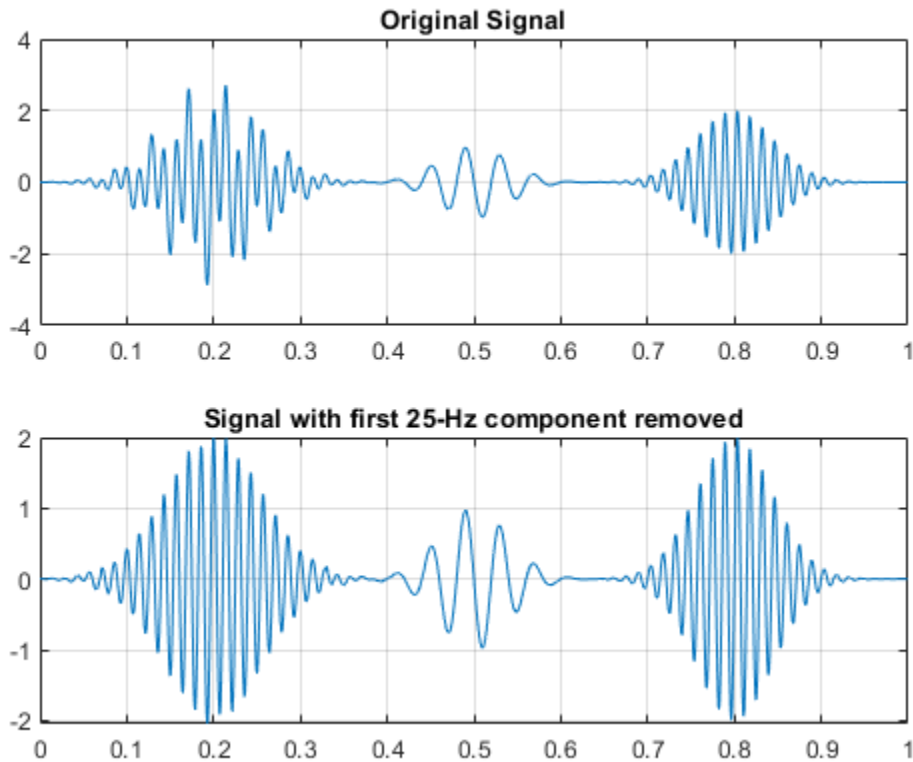
Display the CWT of the reconstructed signal. The initial 25-Hz component is removed.

```
cwt(xrec,2000)
```



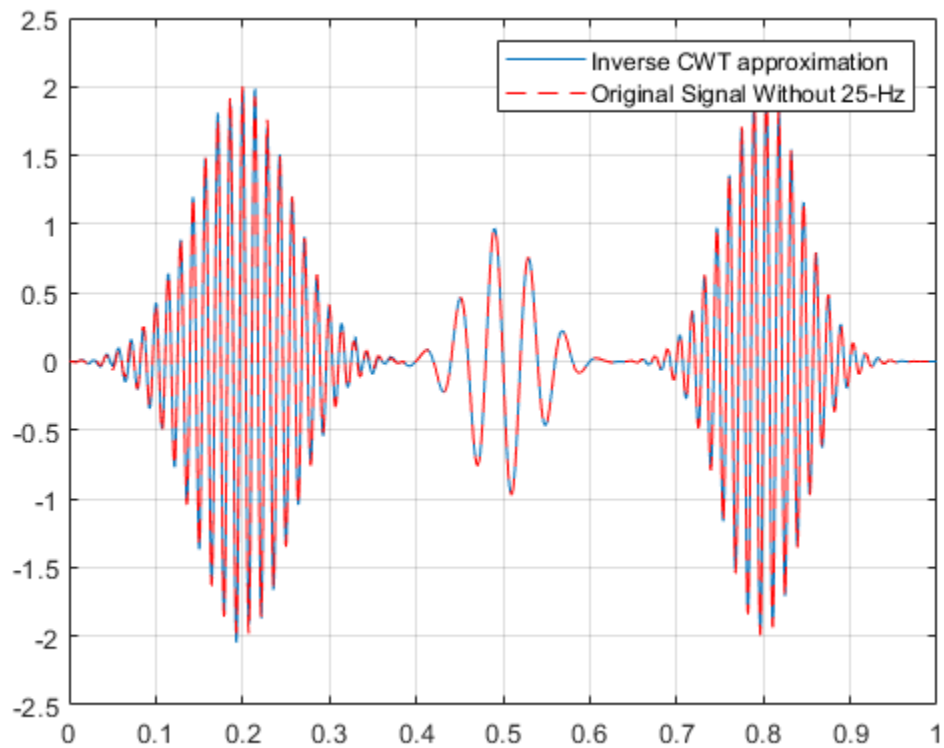
Plot the original signal and the reconstruction.

```
subplot(2,1,1);  
plot(t,x);  
grid on;  
title('Original Signal');  
subplot(2,1,2);  
plot(t,xrec)  
grid on;  
title('Signal with first 25-Hz component removed');
```

Compare the reconstructed signal with the original signal without the 25-Hz component centered at 0.2 seconds.

```
y = x2+x3+x4;  
figure;  
plot(t,xrec)  
hold on  
plot(t,y,'r--')  
grid on;  
legend('Inverse CWT approximation','Original Signal Without 25-Hz');  
hold off
```



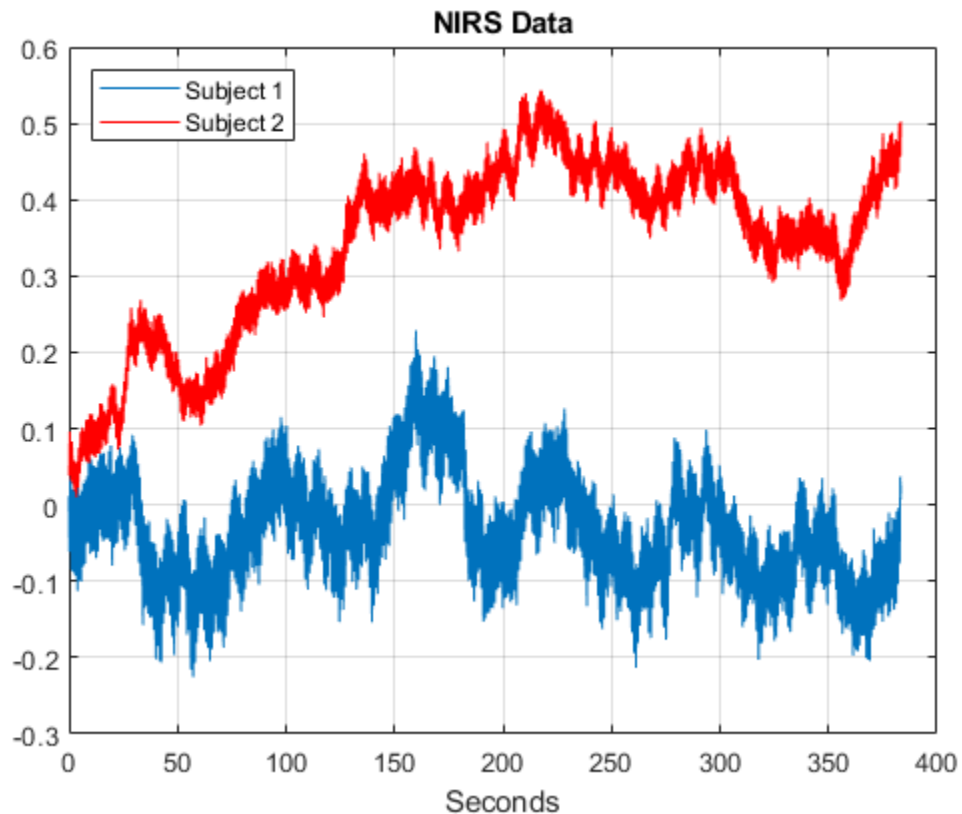
Time-Varying Coherence

Fourier-domain coherence is a well-established technique for measuring the linear correlation between two stationary processes as a function of frequency on a scale from 0 to 1. Because wavelets provide local information about data in time and scale (frequency), wavelet-based coherence allows you to measure time-varying correlation as a function of frequency. In other words, a coherence measure suitable for nonstationary processes.

To illustrate this, examine near-infrared spectroscopy (NIRS) data obtained in two human subjects. NIRS measures brain activity by exploiting the different absorption characteristics of oxygenated and deoxygenated hemoglobin. The recording site was the superior frontal cortex for both subjects and the data was sampled at 10 Hz. The data is taken from Cui, Bryant, & Reiss (2012) and was kindly provided by the authors for this example.

In the experiment, the subjects alternatively cooperated and competed on a task. The period of the task was approximately 7.5 seconds.

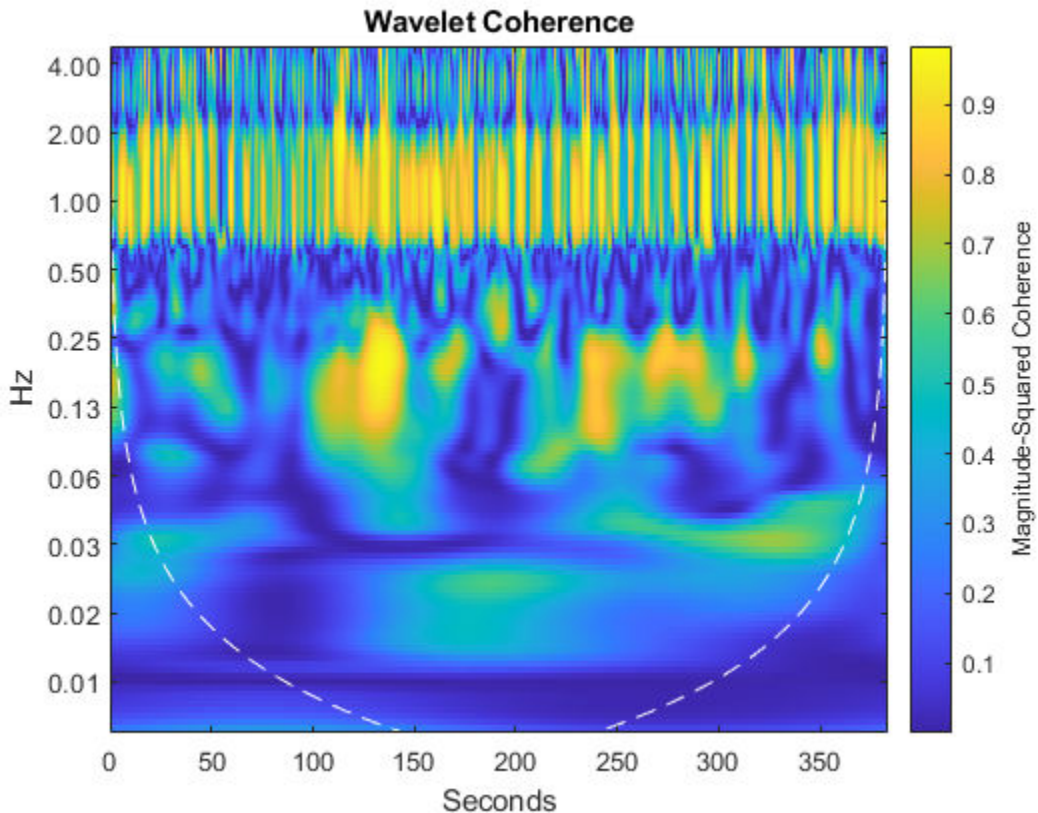
```
load NIRSData;
figure
plot(tm,NIRSData(:,1))
hold on
plot(tm,NIRSData(:,2),'r')
legend('Subject 1','Subject 2','Location','NorthWest')
xlabel('Seconds')
title('NIRS Data')
grid on;
hold off;
```



Examining the time-domain data, it is not clear what oscillations are present in the individual time series, or what oscillations are common to both data sets. Use wavelet analysis to answer both questions.

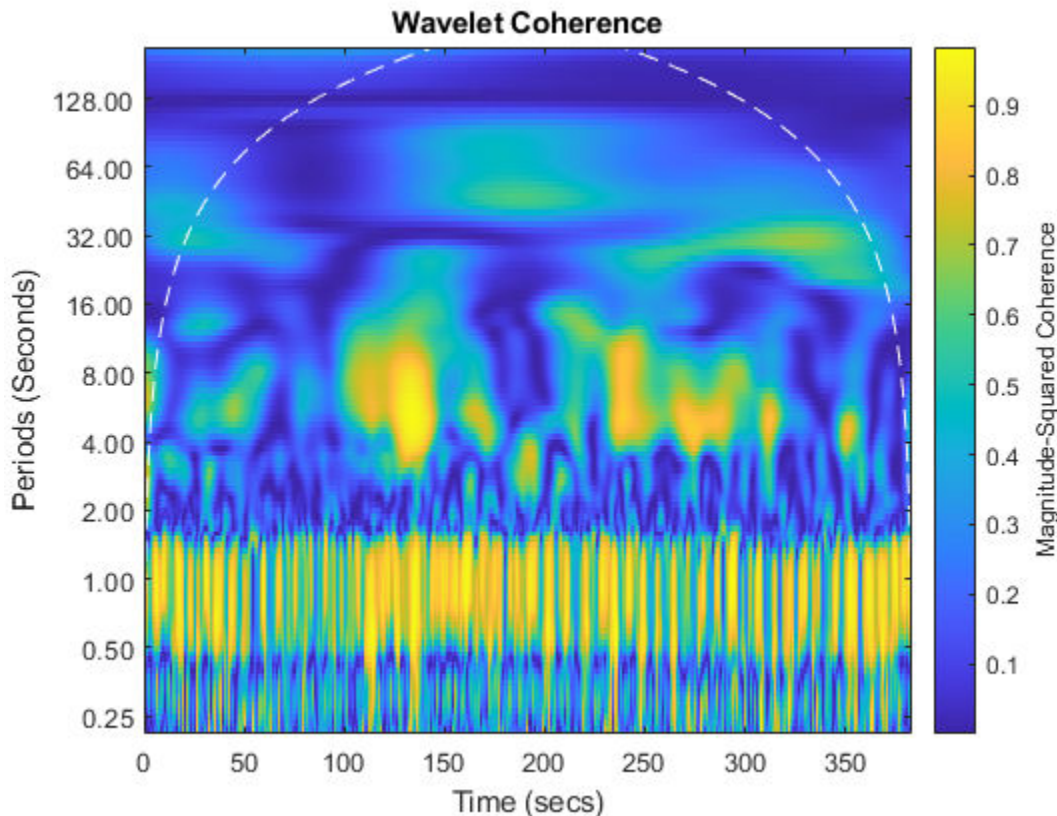
Obtain the wavelet coherence as a function of time and frequency. You can use `wcoherence` to output the wavelet coherence, cross-spectrum, scale-to-frequency, or scale-to-period conversions, as well as the cone of influence. In this example, the helper function `helperPlotCoherence` packages some useful commands for plotting the outputs of `wcoherence`.

```
[wcoh,~,f,coi] = wcoherence(NIRSData(:,1),NIRSData(:,2),10,'numscals',16);
helperPlotCoherence(wcoh,tm,f,coi,'Seconds','Hz');
```



In the plot, you see a region of strong coherence throughout the data collection period around 1 Hz. This results from the cardiac rhythms of the two subjects. Additionally, you see regions of strong coherence around 0.13 Hz. This represents coherent oscillations in the subjects' brains induced by the task. If it is more natural to view the wavelet coherence in terms of periods rather than frequencies, you can input the sampling interval. With the sampling interval, `wcoherence` provides scale-to-period conversions.

```
[wcoh,~,P,coi] = wcoherence(NIRSData(:,1),NIRSData(:,2),seconds(1/10),...
    'numscals',16);
helperPlotCoherence(wcoh,tm,seconds(P),seconds(coi),'Time (secs)','Periods (Seconds)')
```



Again, note the coherent oscillations corresponding to the subjects' cardiac activity occurring throughout the recordings with a period of approximately one second. The task-related activity is also apparent with a period of approximately 8 seconds. Consult Cui, Bryant, & Reiss (2012) for a more detailed wavelet analysis of this data.

In summary, this example showed how to use wavelet coherence to look for time-localized coherent oscillatory behavior in two time series. For nonstationary signals, a measure of coherence that provides simultaneous time and frequency (period) information is often more useful.

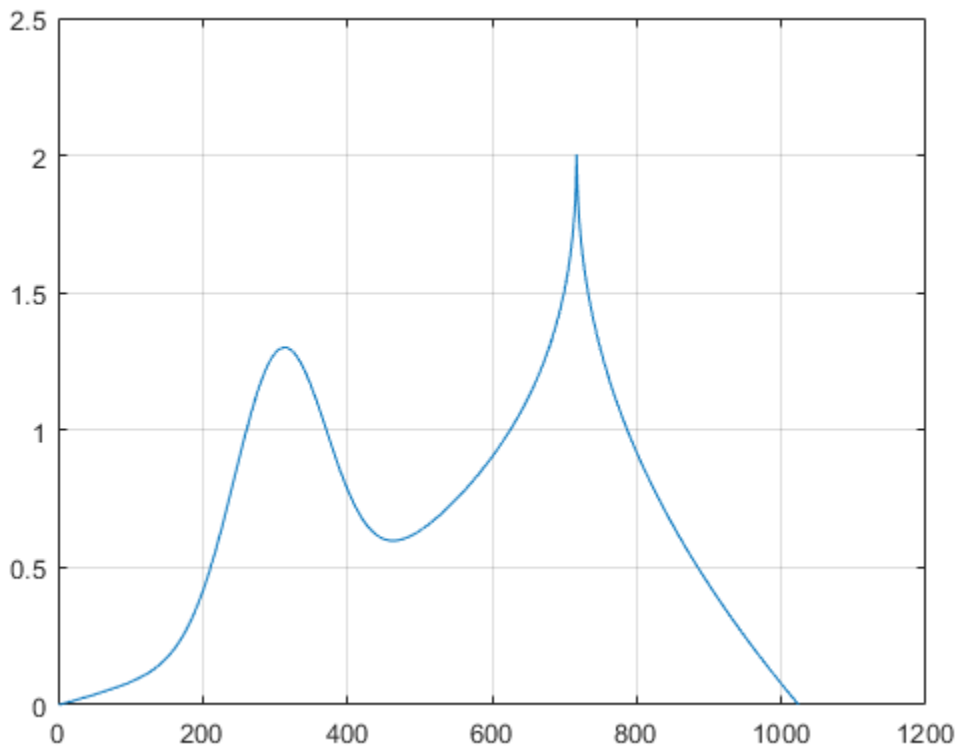
Reference: Cui, X., D. M. Bryant, and A. L. Reiss. "NIRS-Based hyperscanning reveals increased interpersonal coherence in superior frontal cortex during cooperation." *Neuroimage*. Vol. 59, Number 3, 2012, pp. 2430-2437.

Continuous Wavelet Analysis of Cusp Signal

This example shows how to perform continuous wavelet analysis of a cusp signal. You can use `cwt` for analysis using an analytic wavelet and `wtmm` to isolate and characterized singularities.

Load and plot a cusp signal. Display its definition at the command line.

```
load cuspamax;  
plot(cuspamax); grid on;
```

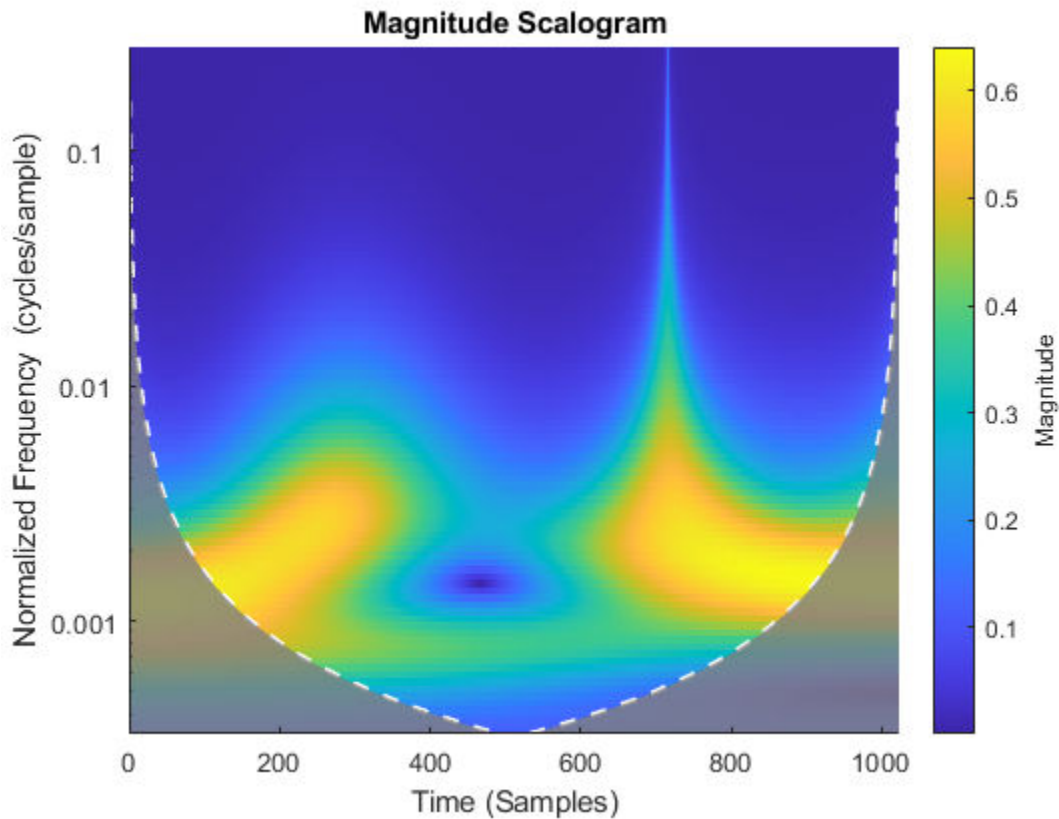


```
disp(caption)
```

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

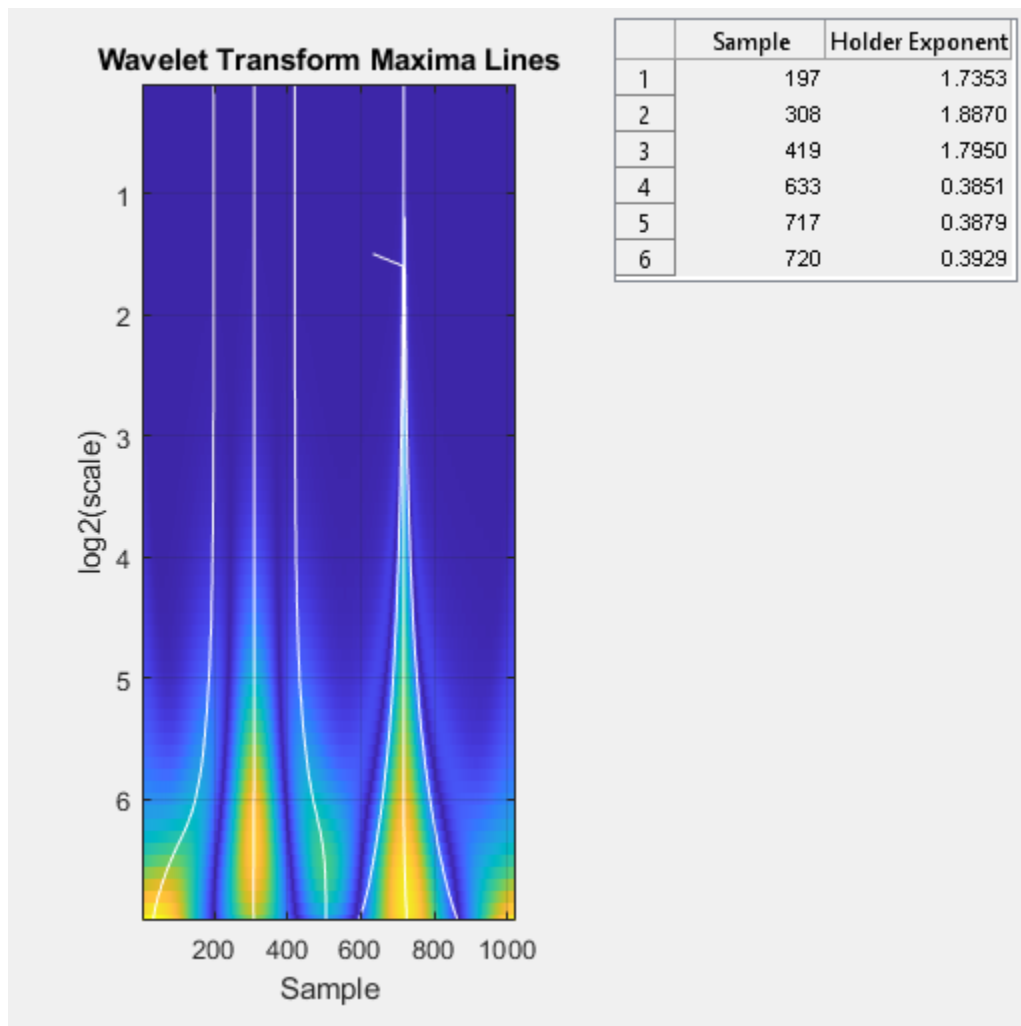

Obtain and view the CWT of the cusp signal. The CWT uses an analytic Morse wavelet with gamma equal to 2 and a time-bandwidth parameter of 2.5. Notice the narrow region in the scalogram converging to the finest scale (highest frequency). This indicates a discontinuity in the signal.

```
cwt(cuspsamax, 'WaveletParameters', [2 2.5]);
```



Obtain a plot of the wavelet maxima lines using wavelet transform modulus maxima. `wtmm` returns estimates of the Holder exponents, which characterize isolated singularities in a signal. Notice that the cusp is shown very clearly using `wtmm`.

```
wtmm(cuspsamax, 'ScalingExponent', 'local');
```

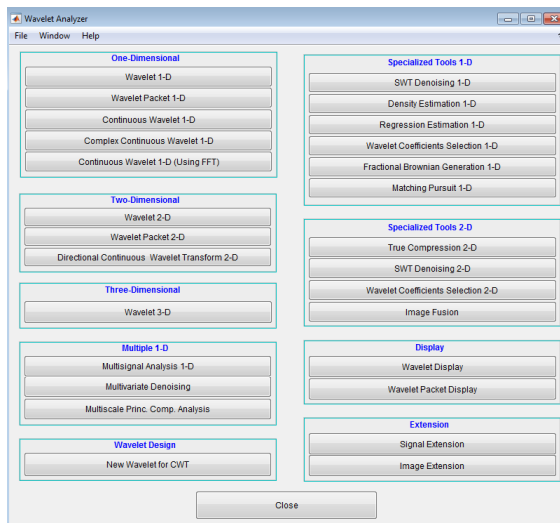


Complex Continuous Analysis Using the Wavelet Analyzer App

This example shows how to use the **Complex Continuous Wavelet 1-D** tool in the Wavelet Analyzer app to analyze a cusp signal.

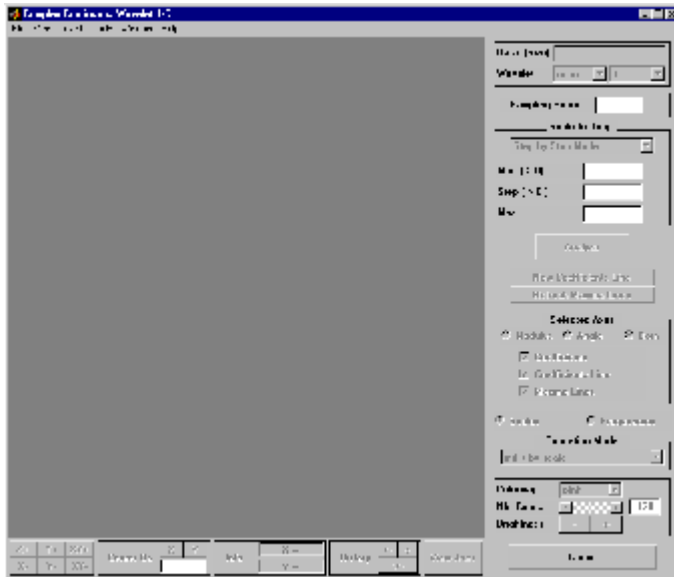
- 1 Start the Complex Continuous Wavelet 1-D Tool.

From the MATLAB prompt, type `waveletAnalyzer`. The **Wavelet Analyzer** appears.



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

The continuous wavelet analysis tool for 1-D signal data appears.



2 Load a signal.

At the MATLAB command prompt, type

```
load cuspamax;
```

In the **Complex Continuous Wavelet 1-D** tool, select **File > Import from Workspace**. When the **Import from Workspace** dialog box appears, select the `cuspamax` variable. Click **OK** to import the `cusp` signal data.

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

3 Perform a Complex Continuous Wavelet Transform

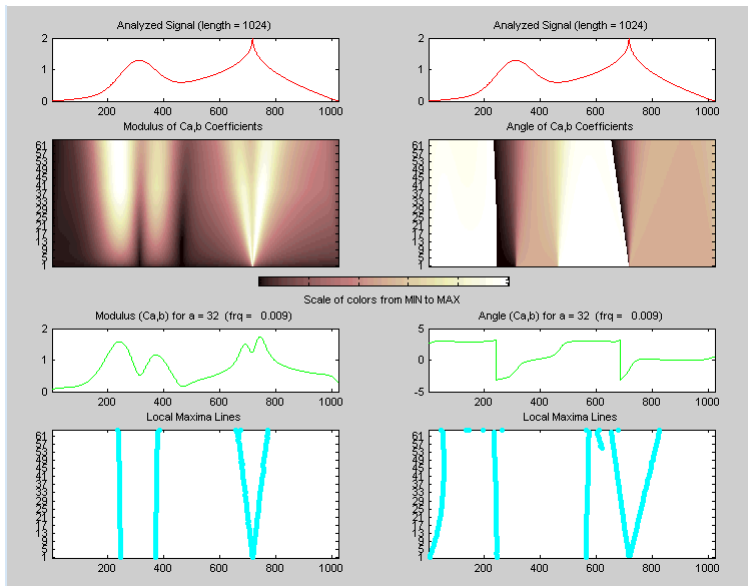
To start our analysis, let's perform an analysis using the `cgau4` wavelet at scales 1 through 64 in steps of 2, just as we did using command-line functions in “Continuous Wavelet Analysis of Cusp Signal” on page 2-72.

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.

Data (Size)	cuspamax (1024)	
Wavelet	cgau	1
Sampling Period		
1		
Scale Settings		
Step by Step Mode		
Min. (> 0)	1	
Step (> 0)	1	
Max. (<= 512)	64	
Analyze		

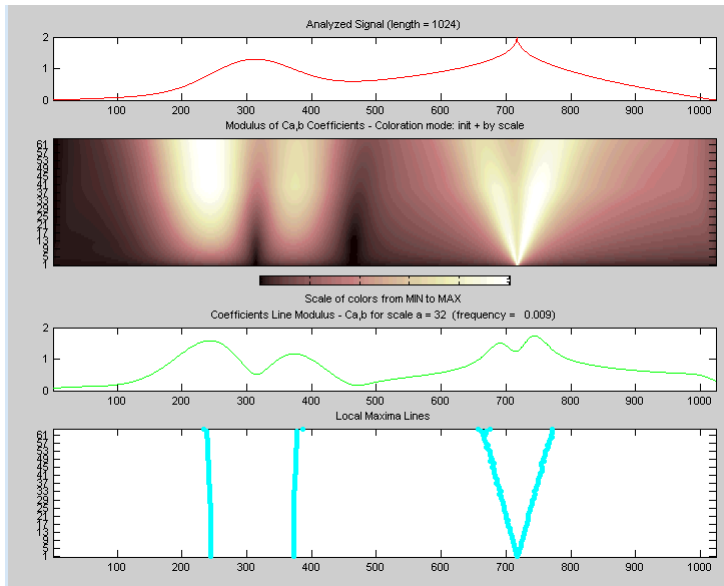
Click the **Analyze** button.

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 Wavelet Analysis of Noisy Sinusoid Using the Wavelet Analyzer App” on page 2-8.

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



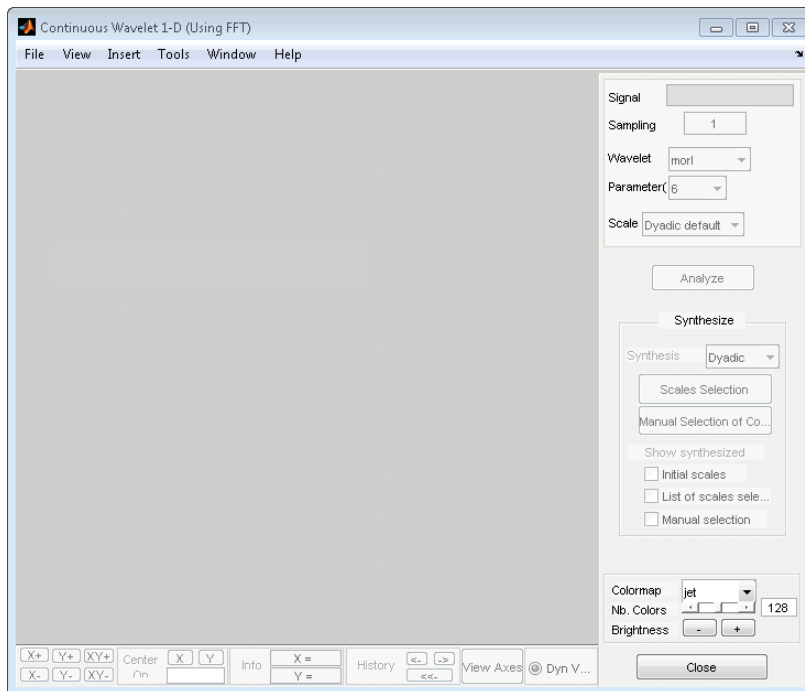
The figure now looks like the one in the real **Continuous Wavelet 1-D** tool.

To import and export information from the Complex Continuous Wavelet tool, see “Importing and Exporting Information from the Wavelet Analyzer App” on page 2-19. The only difference is that the variable `coefs` is a complex matrix (see “Saving Wavelet Coefficients” on page 2-19).

DFT-Based Continuous Wavelet Analysis Using the Wavelet Analyzer App

You can use the **Continuous Wavelet 1-D (Using FFT)** tool to perform continuous wavelet analysis.

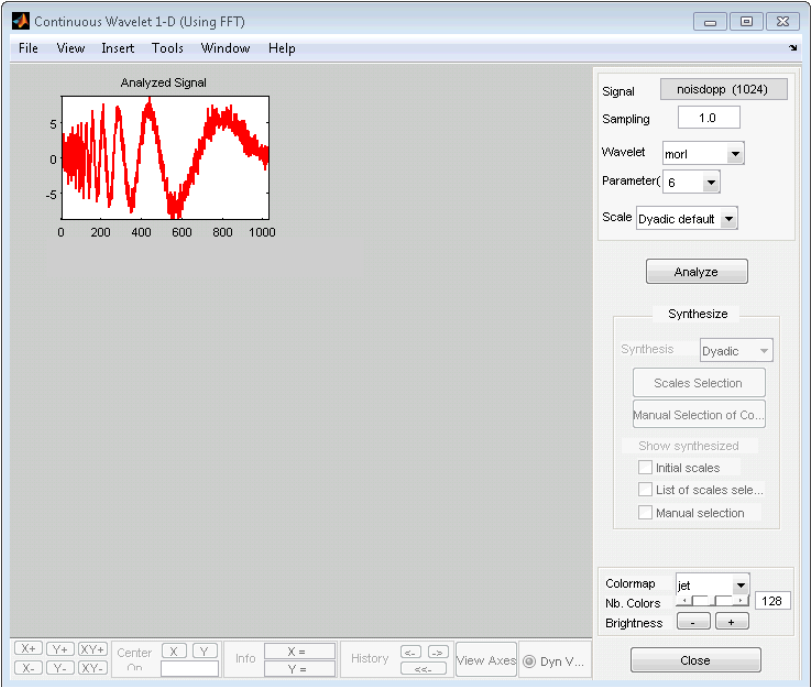
- 1 At the MATLAB command prompt, enter
`waveletAnalyzer`
- 2 Click the **Continuous Wavelet 1-D (Using FFT)** menu item.



- 3 At the MATLAB command prompt, type

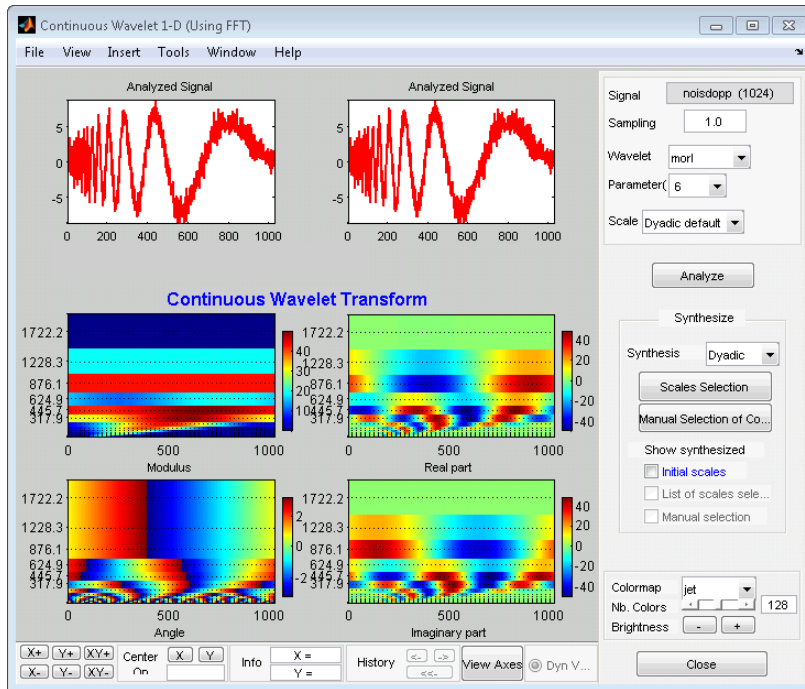
```
load noisdopp;
```

In the **Continuous Wavelet 1-D (Using FFT)** tool, select **File > Import from Workspace**. When the **Import from Workspace** dialog box appears, select the `noisdopp` variable. Click **OK** to import the data.

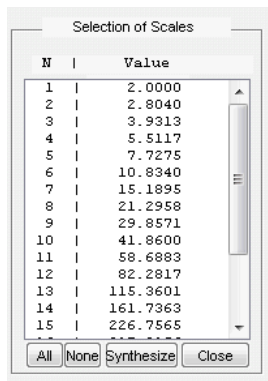


4 Using the menu default parameters, click **Analyze**.

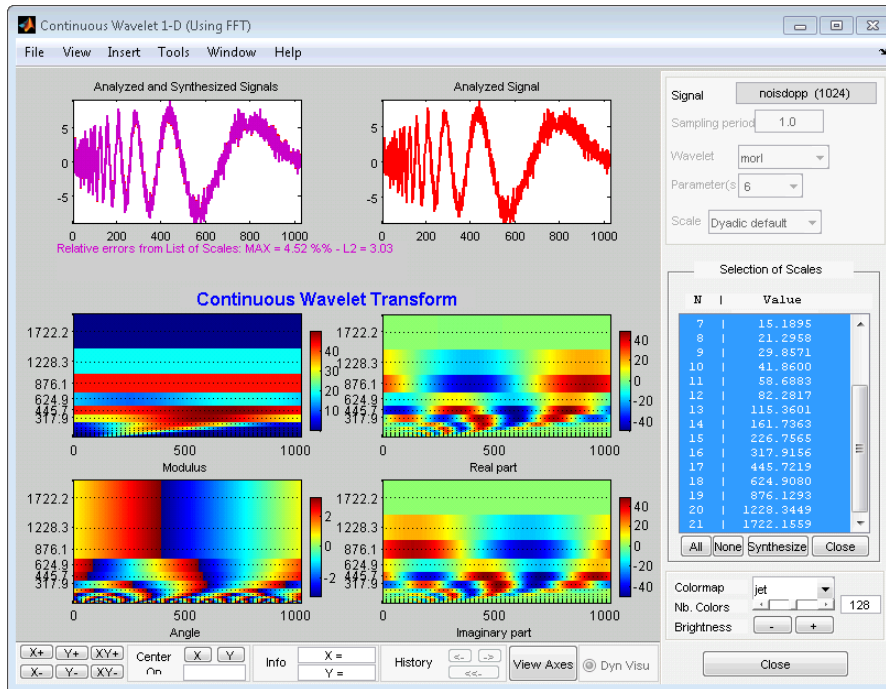
2 Continuous Wavelet Analysis



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



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



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

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

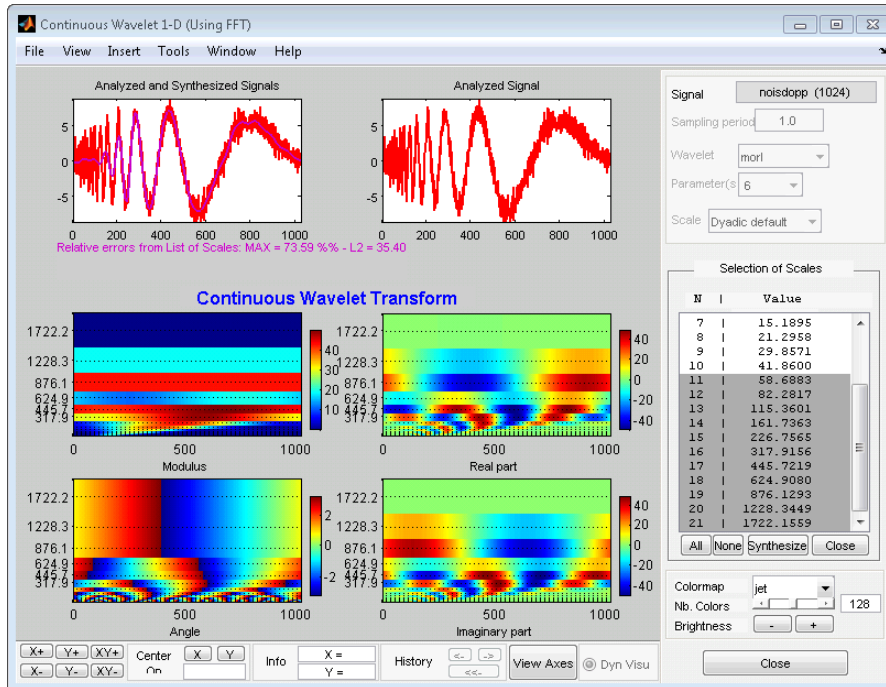
- Obtain a signal approximation from selected scales.

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

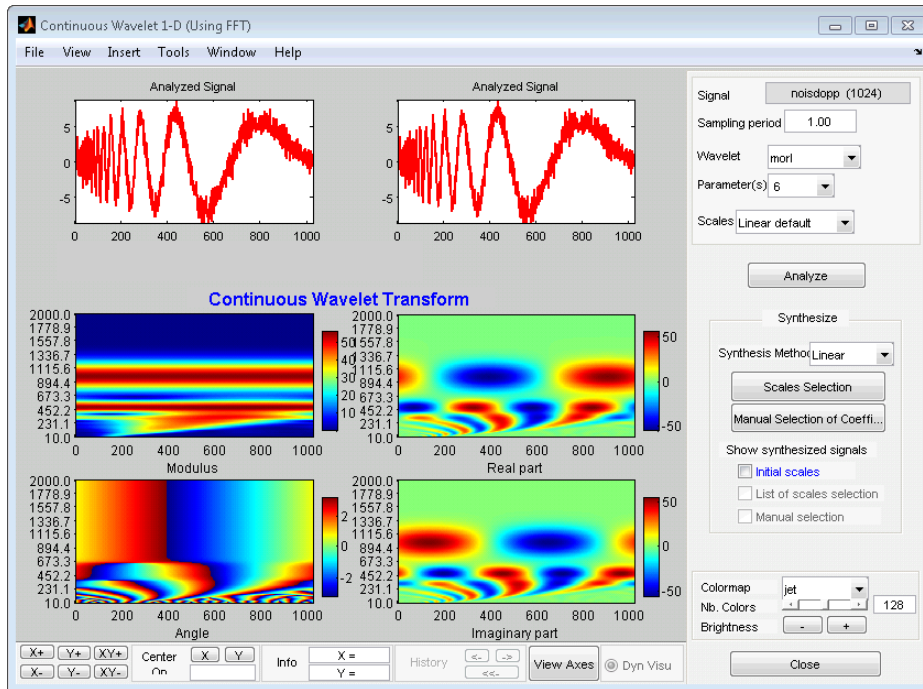
```
dt = 1;
s0 = 2*dt;
ds = 0.4875;
nb = 21;
physical_scales = s0*pow.^(0:nb-1)*ds;
```

- Click **Synthesize**.

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



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

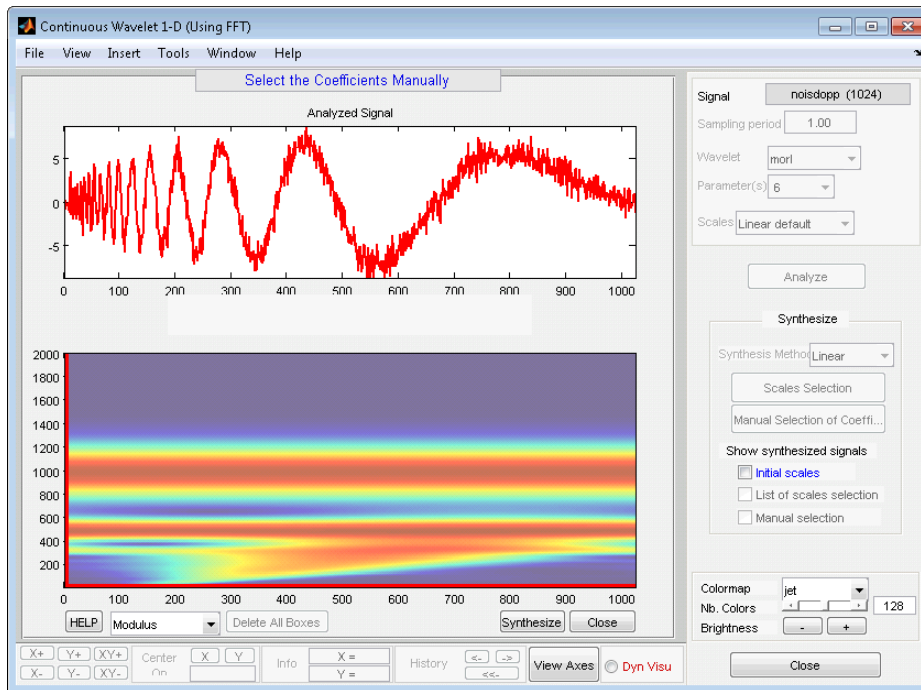


Note The other options under **Scales** include Dyadic default and Manual.

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

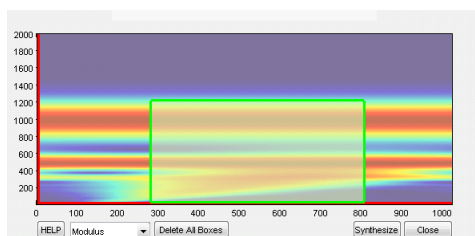
Manual Selection of CWT Coefficients

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

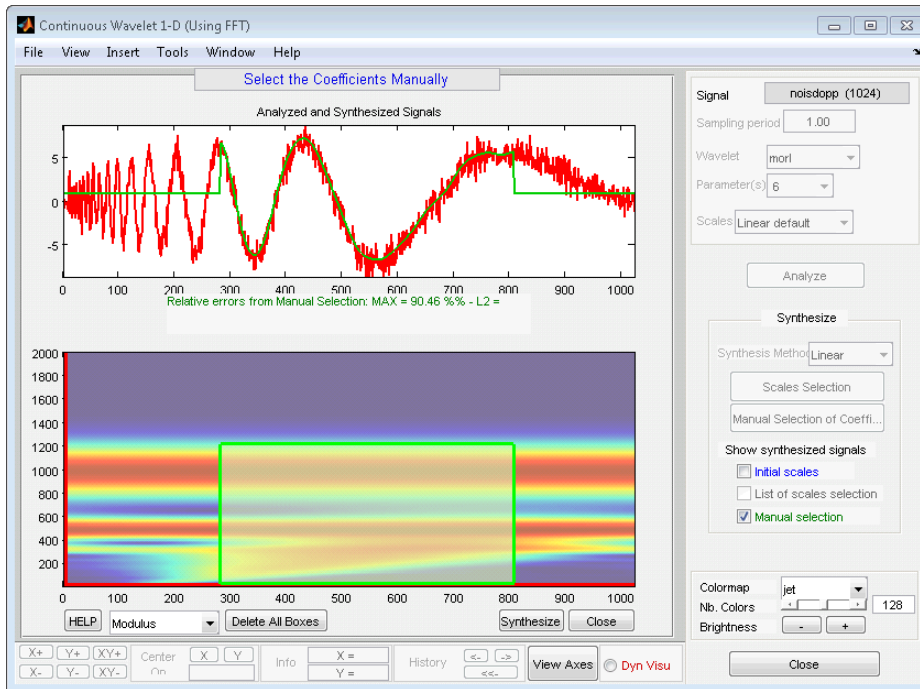


You can change the CWT coefficient view to Angle, Real, or Imaginary.

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



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



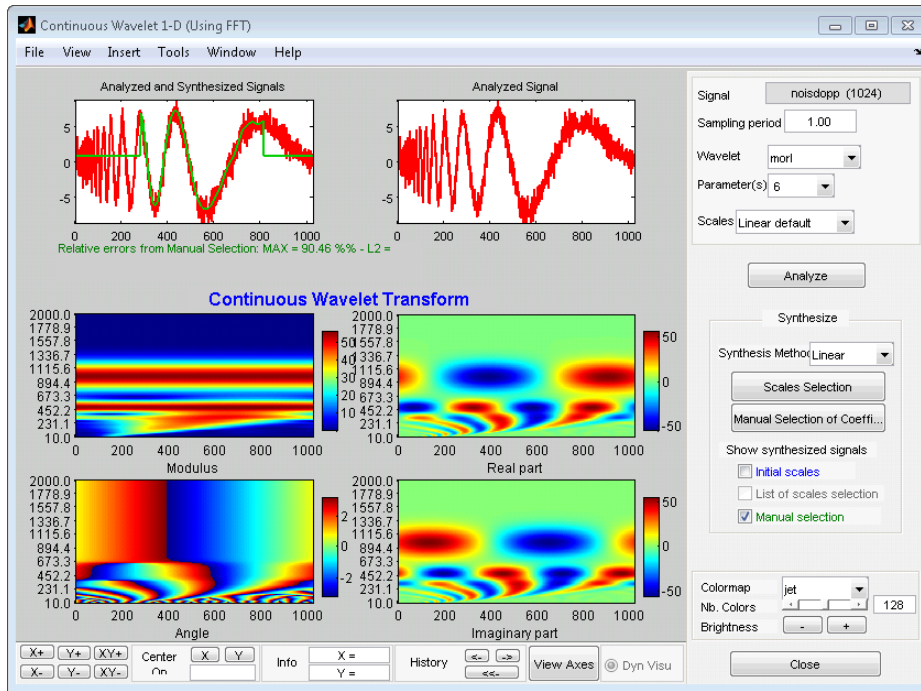
To select, unselect, or delete a box, right-click in the box. A context menu appears that allows you to **select**, **unselect**, or **delete** the box. After you select the coefficients within the box, the border of the box displays in green. When the coefficients within the box are not selected, the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

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

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

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

2 Continuous Wavelet Analysis



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

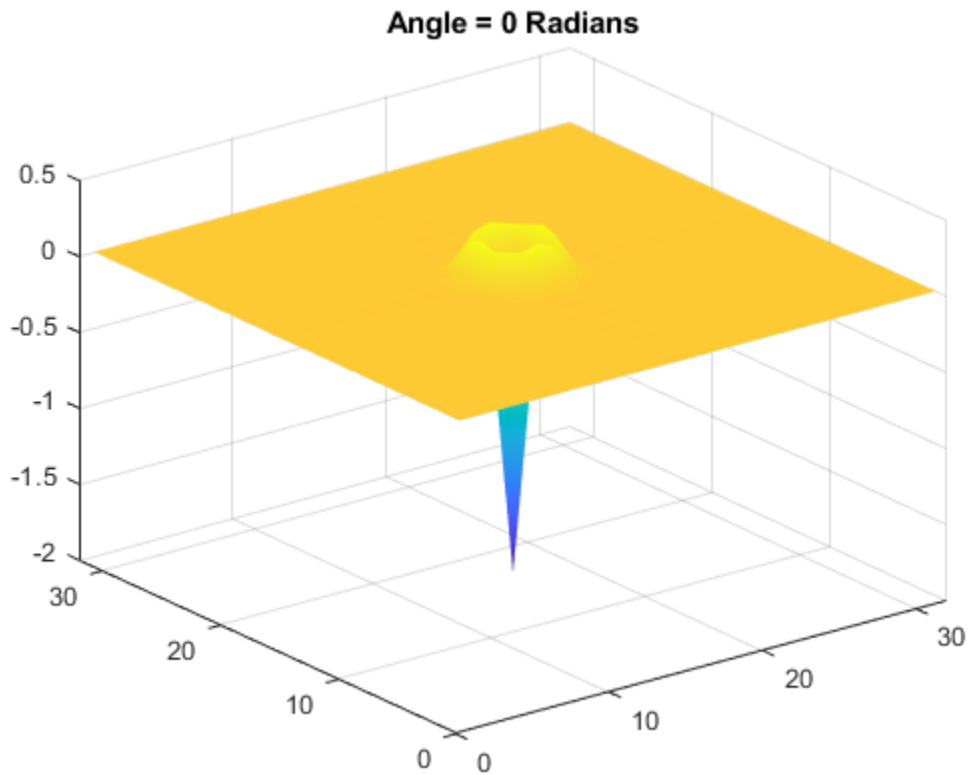
Using the **File > Save > Decomposition** menu, you can save the wavelet analysis as a MAT file.

Two-Dimensional CWT of Noisy Pattern

This example shows how to detect a pattern in a noisy image using the 2-D continuous wavelet transform (CWT). The example uses both isotropic (non-directional) and anisotropic (directional) wavelets. The isotropic wavelet is not sensitive to the orientation of the feature, while the directional wavelet is.

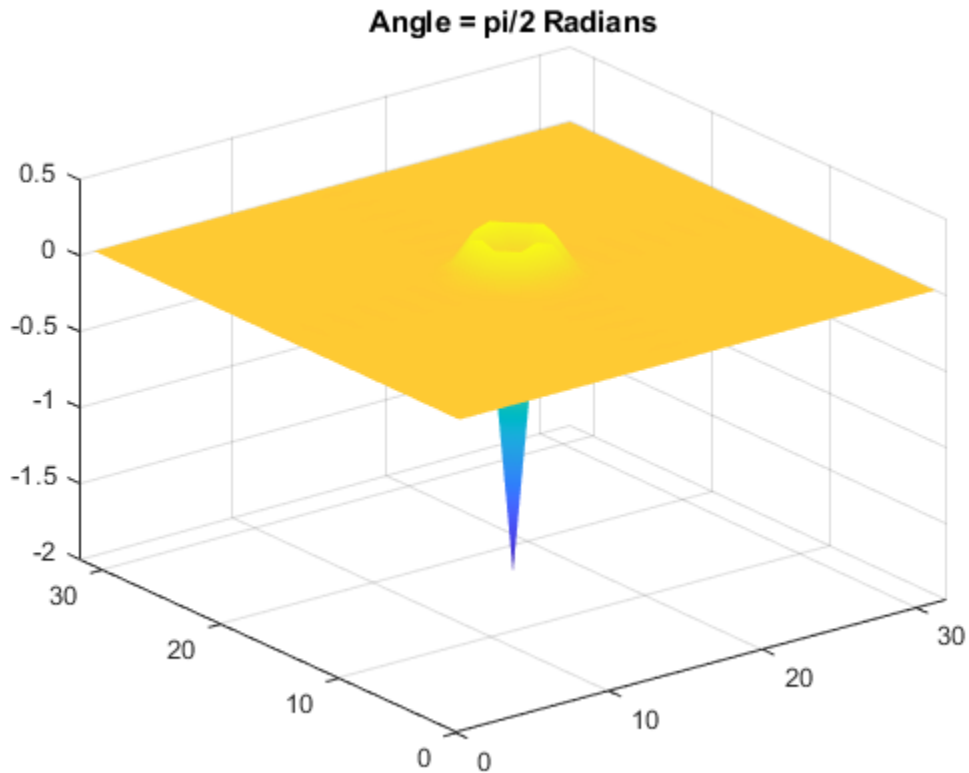
Use the isotropic (non-directional) Mexican hat wavelet, also known as the Ricker wavelet, and the anisotropic (directional) Morlet wavelet. Demonstrate that the real-valued Mexican hat wavelet does not depend on the angle.

```
Y = zeros(32,32);
Y(16,16) = 1;
cwtmexh = cwtft2(Y, 'wavelet', 'mexh', 'scales', 1, ...
    'angles', [0 pi/2]);
surf(real(cwtmexh.cfs(:,:,1,1,1)));
shading interp; title('Angle = 0 Radians');
```



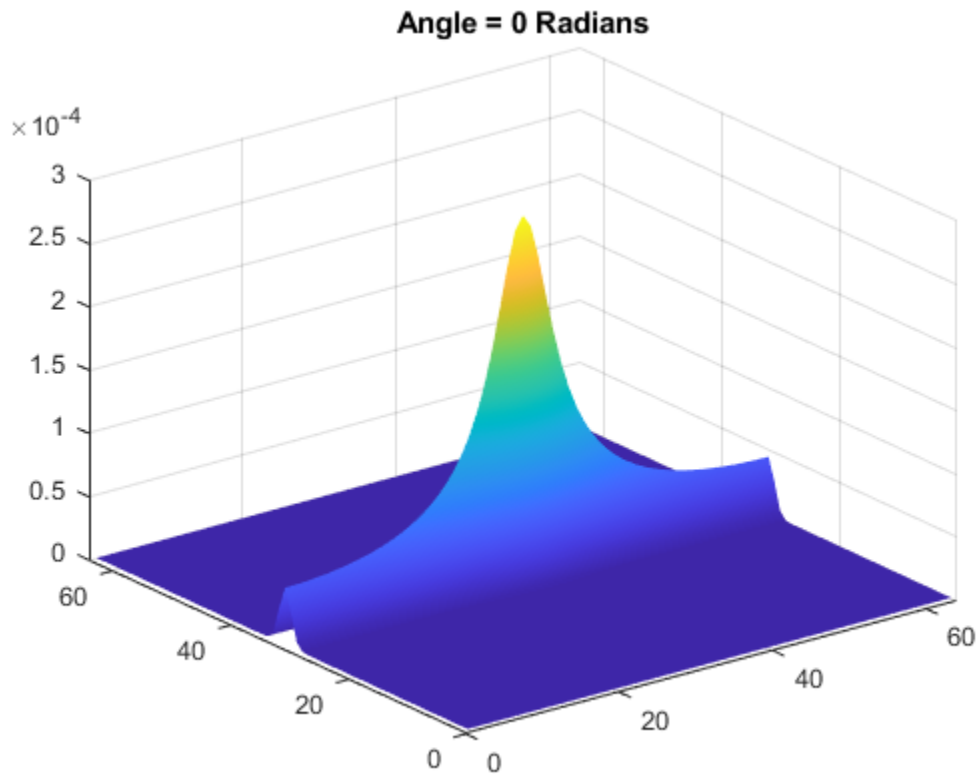
Extract the wavelet corresponding to an angle of $\pi/2$ radians. The wavelet is isotropic and therefore does not differentiate oriented features in data.

```
surf(real(cwtmexh.cfs(:,:,1,1,2)));  
shading interp; title('Angle = pi/2 Radians');
```



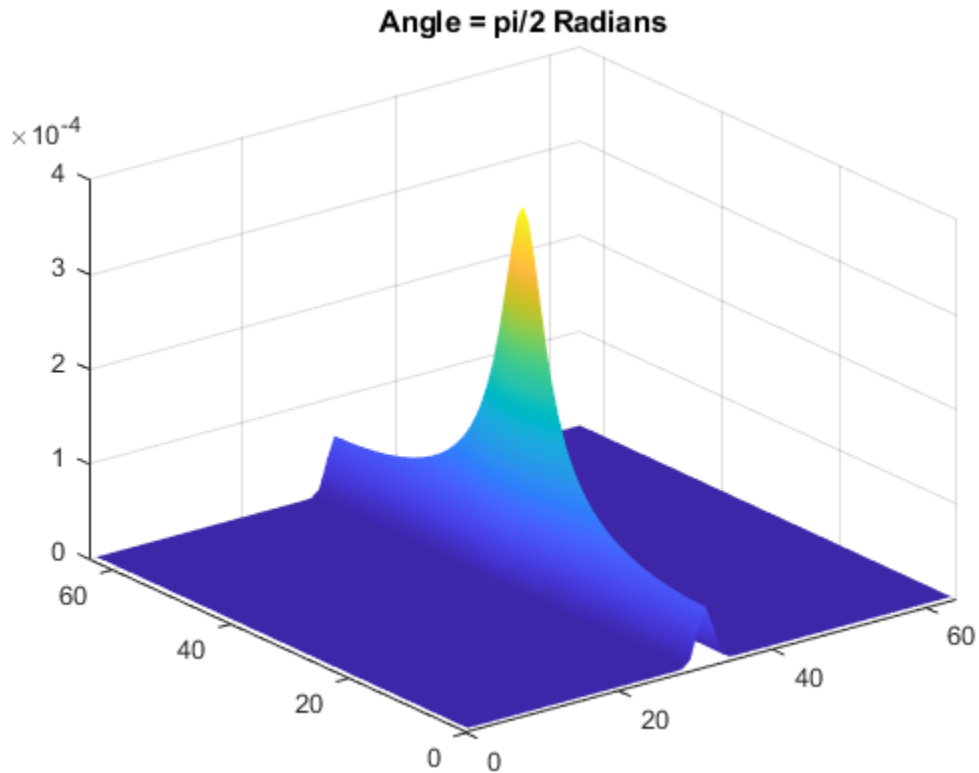
Repeat the preceding steps for the complex-valued Morlet wavelet. The Morlet wavelet has a larger spatial support than the Mexican hat wavelet, therefore this example uses a larger matrix. The wavelet is complex-valued, so the modulus is plotted.

```
Y = zeros(64,64);
Y(32,32) = 1;
cwtmorl = cwtft2(Y,'wavelet','morl','scales',1,...
    'angles',[0 pi/2]);
surf(abs(cwtmorl.cfs(:,:,1,1,1)));
shading interp; title('Angle = 0 Radians');
```



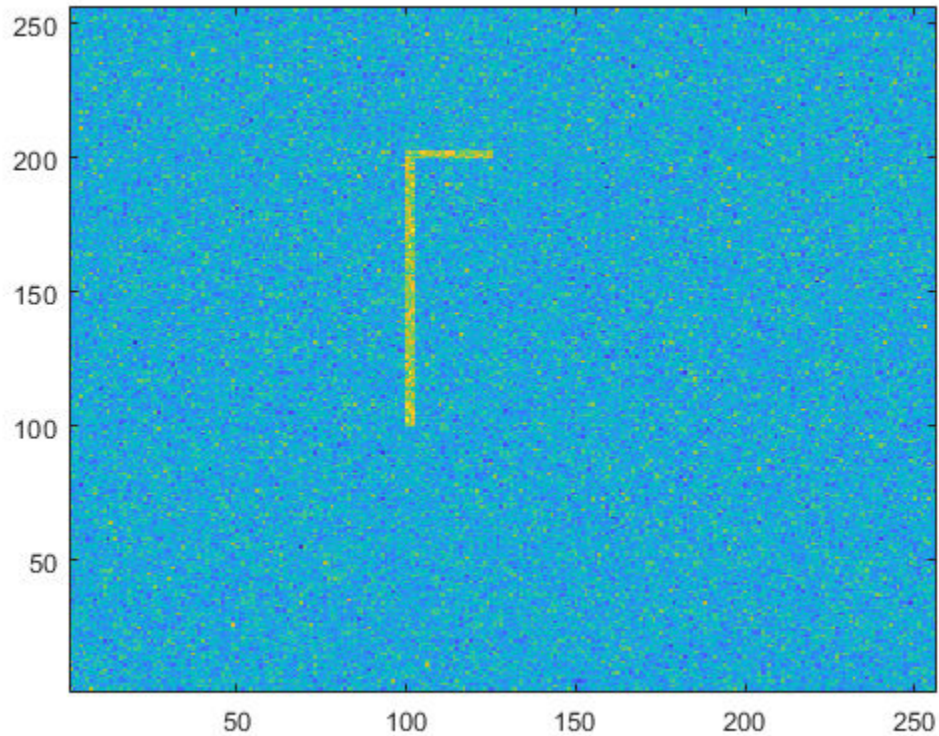
Extract the wavelet corresponding to an angle of $\pi/2$ radians. Unlike the Mexican hat wavelet, the Morlet wavelet is not isotropic and therefore is sensitive to the direction of features in the data.

```
surf(abs(cwtmorl.cfs(:,:,1,1,2)));  
shading interp; title('Angle = pi/2 Radians');
```



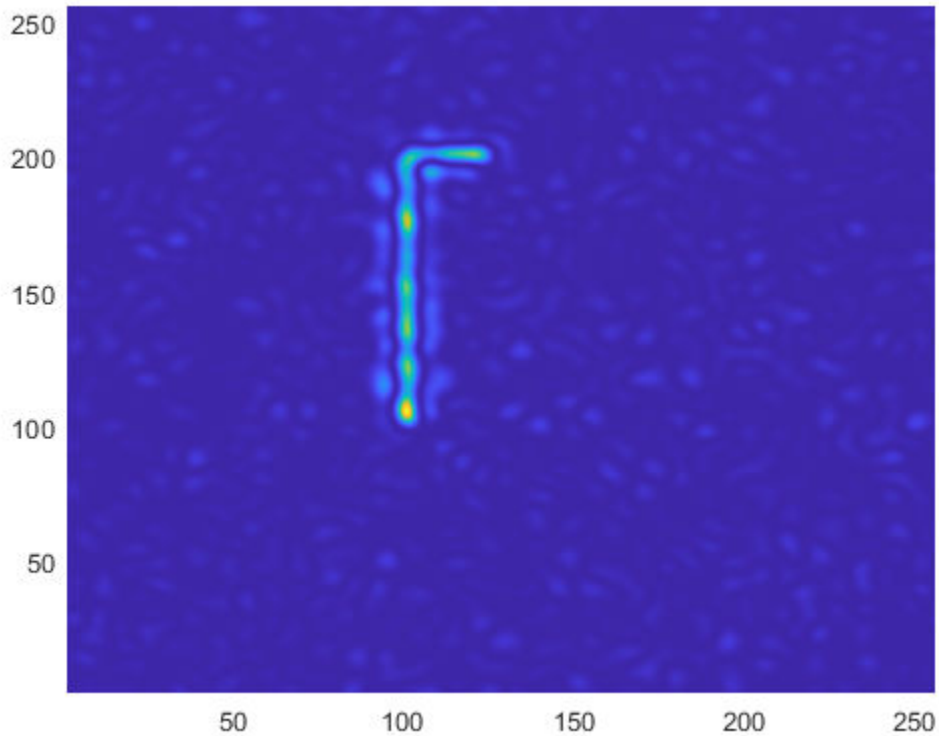
Apply the Mexican hat and Morlet wavelets to the detection of a pattern in noise. Create a pattern consisting of line segments joined at a 90-degree angle. The amplitude of the pattern is 3 and it occurs in additive $N(0,1)$ white Gaussian noise.

```
X = zeros(256,256);
X(100:200,100:102) = 3;
X(200:202,100:125) = 3;
X = X+randn(size(X));
imagesc(X); axis xy;
```



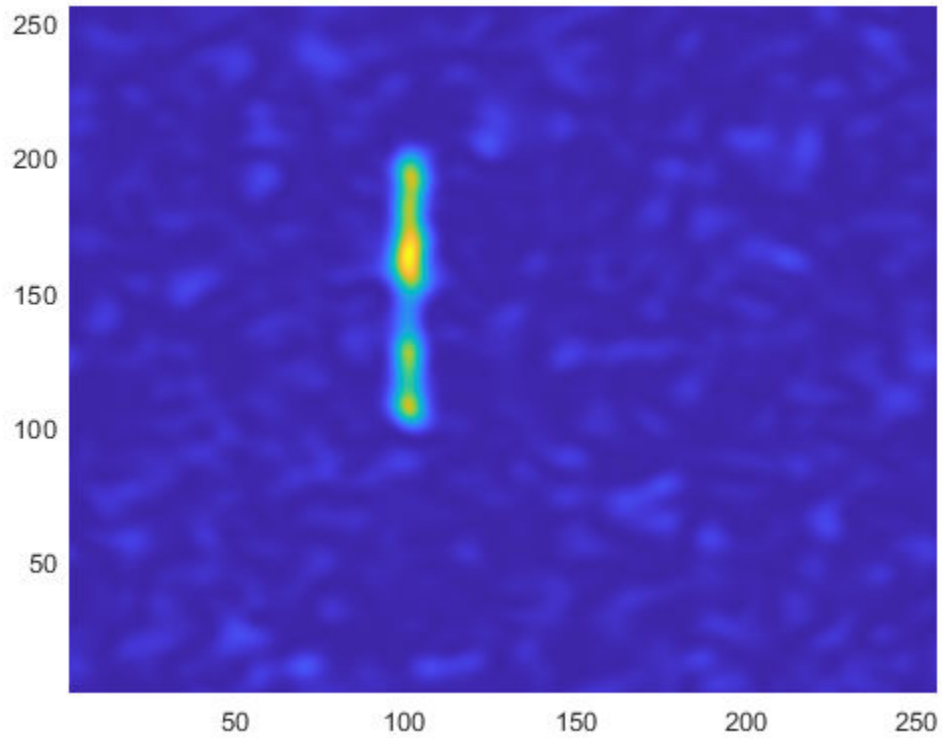
Obtain the 2-D CWT at scales 3 to 8 in 0.5 increments with the Mexican hat wavelet. Visualize the magnitude-squared 2-D wavelet coefficients at scale 3.

```
cwtmexh = cwtft2(X, 'wavelet', 'mexh', 'scales', 3:0.5:8);  
surf(abs(cwtmexh.cfs(:, :, 1, 3, 1)).^2);  
view(0, 90); shading interp; axis tight;
```

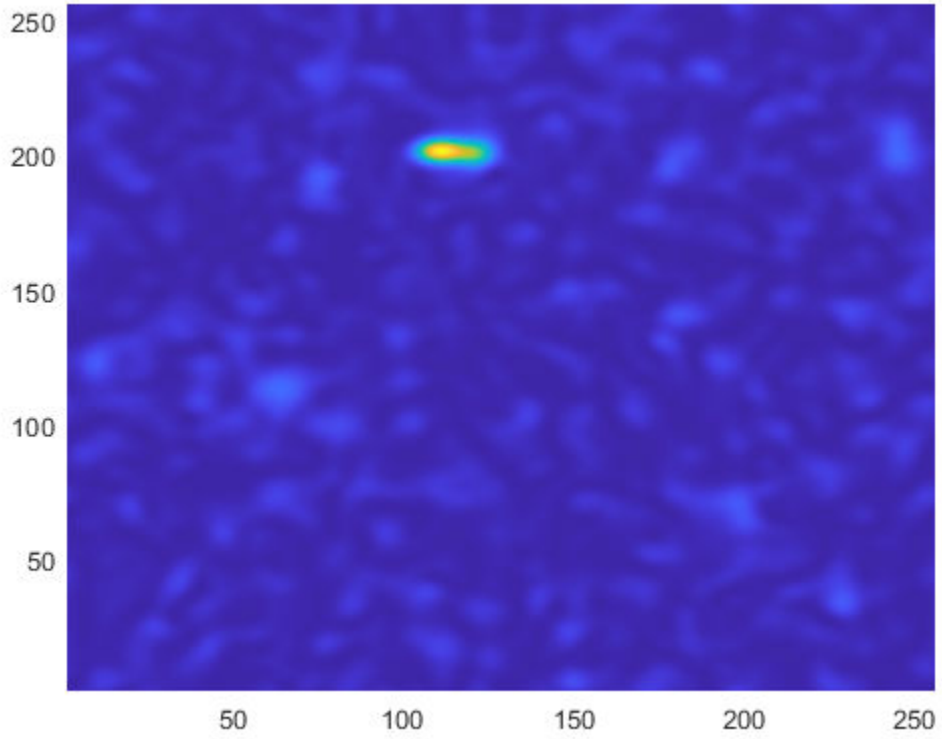


Use a directional Morlet wavelet to extract the vertical and horizontal line segments separately. The vertical line segment is extracted by one angle. The horizontal line segment is extracted by another angle.

```
cwtmorl = cwtft2(X,'wavelet','morl','scales',3:0.5:8,...
    'angles',[0 pi/2]);
surf(abs(cwtmorl.cfs(:,:,1,4,1)).^2);
view(0,90); shading interp; axis tight;
```



```
figure;  
surf(abs(cwtmorl.cfs(:,:,1,4,2)).^2);  
view(0,90); shading interp; axis tight;
```

2-D Continuous Wavelet Transform App

In this section...

“2-D Continuous Wavelet Transform” on page 2-98

“2-D CWT App Example” on page 2-99

The 2-D continuous wavelet transform (CWT) app enables you to analyze your image data and export the results of that analysis to the MATLAB workspace. The app provides all the functionality of the command line functions `cwtft2` and `cwtftinfo2`. Access the 2-D CWT app in the apps gallery by selecting **Wavelet Design & Analysis** in the **Signal Processing and Communications** section or entering

```
cwtfttool2
```

at the MATLAB command prompt.

2-D Continuous Wavelet Transform

The 2-D continuous wavelet transform is a representation of 2-D data (image data) in 4 variables: dilation, rotation, and position. Dilation and rotation are real-valued scalars and position is a 2-D vector with real-valued elements. Let x denote a two-element vector of real-numbers. If

$$f(x) \in L^2(\mathbb{R}^2)$$

is square-integrable on the plane, the 2-D CWT is defined as

$$\text{WT}_f(a, b, \theta) = \int_{\mathbb{R}^2} f(x) \frac{1}{a} \bar{\psi}(r_{-\theta}(\frac{x-b}{a})) dx \quad a \in \mathbb{R}^+, x, b \in \mathbb{R}^2$$

where the bar denotes the complex conjugate and r_{θ} is the 2-D rotation matrix

$$r_{\theta} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \quad \theta \in [0, 2\pi)$$

The 2-D CWT is a space-scale representation of an image. You can view the inverse of the scale and the rotation angle taken together as a spatial-frequency variable, which gives the 2-D CWT an interpretation as a space-frequency representation. For all admissible 2-D wavelets, the 2-D CWT acts as a local filter for an image in scale and position. If the wavelet is isotropic, there is no dependence on angle in the analysis. The Mexican hat

wavelet is an example of an isotropic wavelet. Isotropic wavelets are suitable for pointwise analysis of images. If the wavelet is anisotropic, there is a dependence on angle in the analysis, and the 2-D CWT acts a local filter for an image in scale, position, and angle. The Cauchy wavelet is an example of an anisotropic wavelet. In the Fourier domain, this means that the spatial frequency support of the wavelet is a convex cone with the apex at the origin. Anisotropic wavelets are suitable for detecting directional features in an image. See “Two-Dimensional CWT of Noisy Pattern” on page 2-89 for an illustration of the difference between isotropic and anisotropic wavelets.

2-D CWT App Example

This example shows how to analyze an image using the 2-D CWT app.

Load the triangle image in the MATLAB workspace.

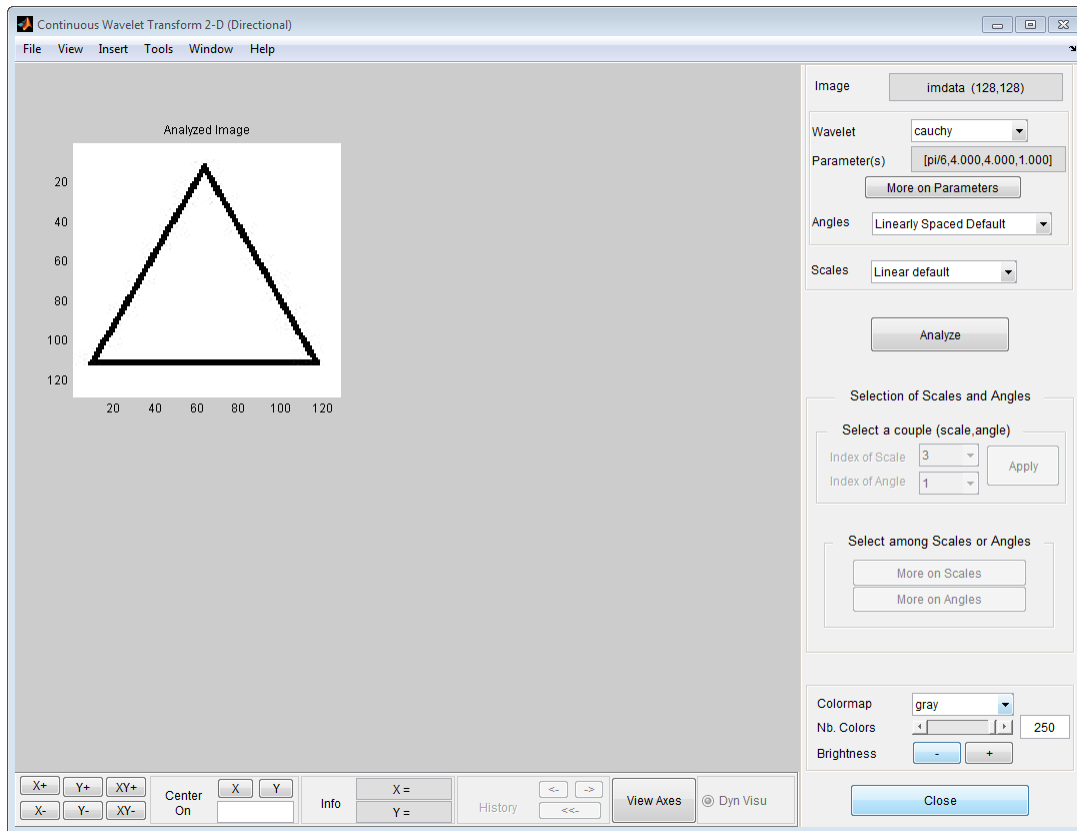
```
imdata = imread('triangle.jpg');
```

Launch the 2-D CWT app by selecting **Wavelet Design & Analysis** in the **Signal Processing and Communications** section of the apps gallery. From the **2-D** section, select **Continuous Wavelet Transform 2-D**. Alternatively, enter

```
cwtffttool2
```

at the MATLAB command prompt.

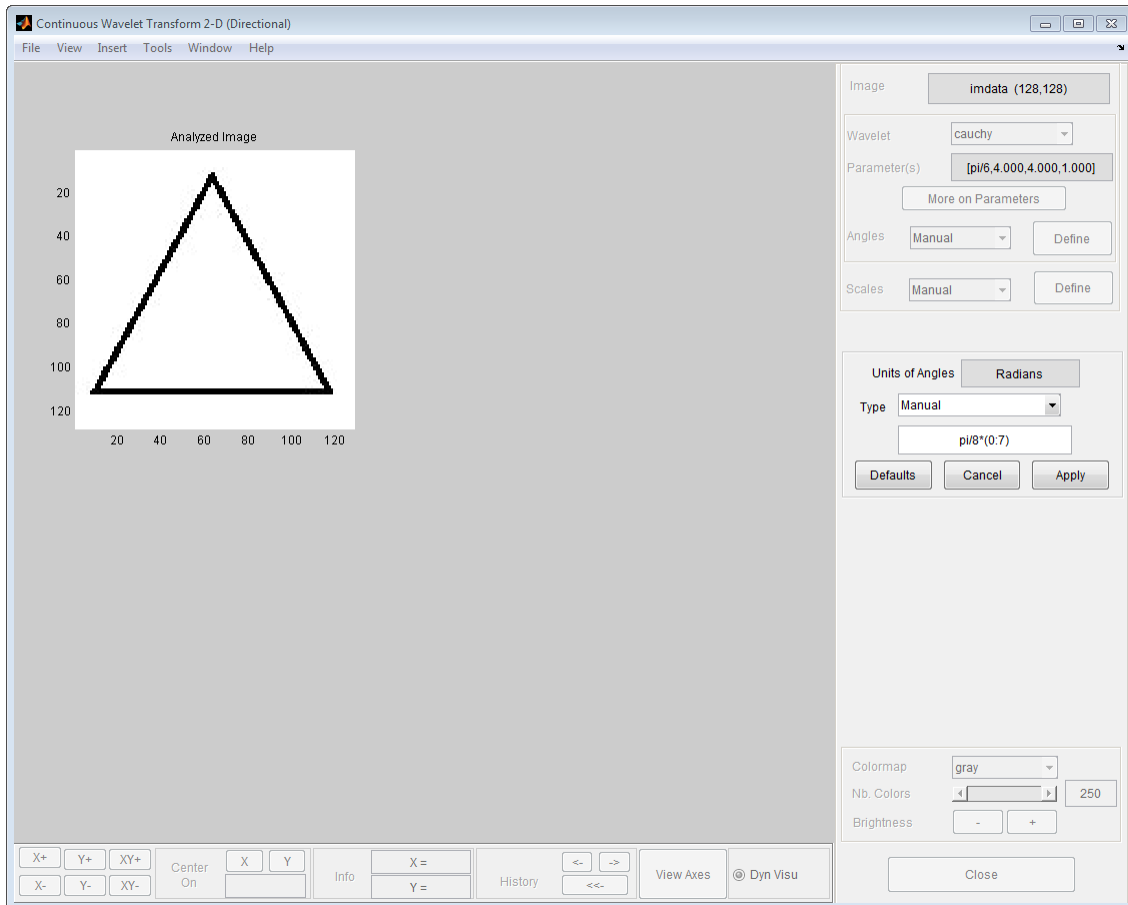
Select **File -> Import Data** to import the `imdata` variable.



From the **Wavelet** drop down menu, select the **cauchy** wavelet.

For the **Angles** and **Scales**, select the **Manual** option.

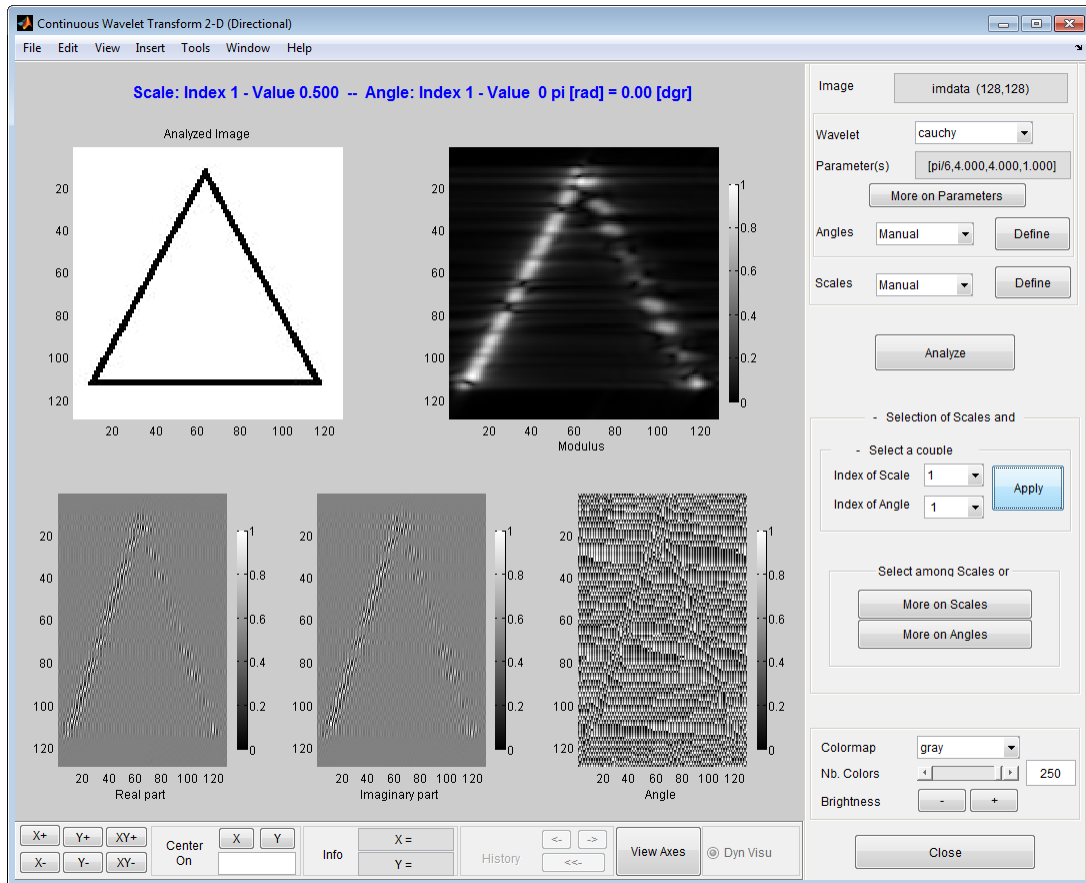
Click **Define** to specify a vector of angles. Select **Manual** from the Type drop-down list and specify a vector of angles from 0 to $7 \cdot \pi / 8$ radians in increments of $\pi / 8$ radians, $0 : \pi / 8 : (7 \cdot \pi) / 8$. Click **Apply** to apply your choice of angles.



Click **Define** to specify a vector of scales from 0.5 to 4 in increments of 0.5. Select **Linear** from the Type drop-down list. Set **First Scale** equal to 0.5, **Gap between two scales** equal to 0.5, and **Number of Scales** equal to 8. Equivalently, you can select **Manual** from the Type drop-down list and specify the vector of scales as $0.5 : 0.5 : 4$. Click **Apply** to apply your choice of scales.

Click **Analyze** to obtain the 2-D CWT.

2 Continuous Wavelet Analysis



Set the Index of Scale to be 1 and click **More on Angles**. Click **Movie** to step through the manually-defined angles for the 2-D CWT coefficients at scale 0.5.

Select **File -> Export Data -> Export CWTF2 Struct to Workspace** to export the analysis to the MATLAB workspace. You can find an explanation of the structure fields in the function reference for `cwtft2`.

Discrete Wavelet Analysis

- “Critically Sampled and Oversampled Wavelet Filter Banks” on page 3-2
- “1-D Decimated Wavelet Transforms” on page 3-11
- “Fast Wavelet Transform (FWT) Algorithm” on page 3-43
- “Border Effects” on page 3-57
- “Nondecimated Discrete Stationary Wavelet Transforms (SWTs)” on page 3-66
- “1-D Stationary Wavelet Transform” on page 3-73
- “Wavelet Changepoint Detection” on page 3-88
- “Scale-Localized Volatility and Correlation” on page 3-103
- “R Wave Detection in the ECG” on page 3-114
- “Wavelet Cross-Correlation for Lead-Lag Analysis” on page 3-125
- “1-D Multisignal Analysis” on page 3-138
- “2-D Discrete Wavelet Analysis” on page 3-188
- “2-D Stationary Wavelet Transform” on page 3-214
- “Shearlet Systems” on page 3-227
- “3-D Discrete Wavelet Analysis” on page 3-231
- “Dual-Tree Wavelet Transforms” on page 3-243
- “Analytic Wavelets Using the Dual-Tree Wavelet Transform” on page 3-282
- “Multifractal Analysis” on page 3-286
- “Wavelet Analysis of Financial Data” on page 3-306

Critically Sampled and Oversampled Wavelet Filter Banks

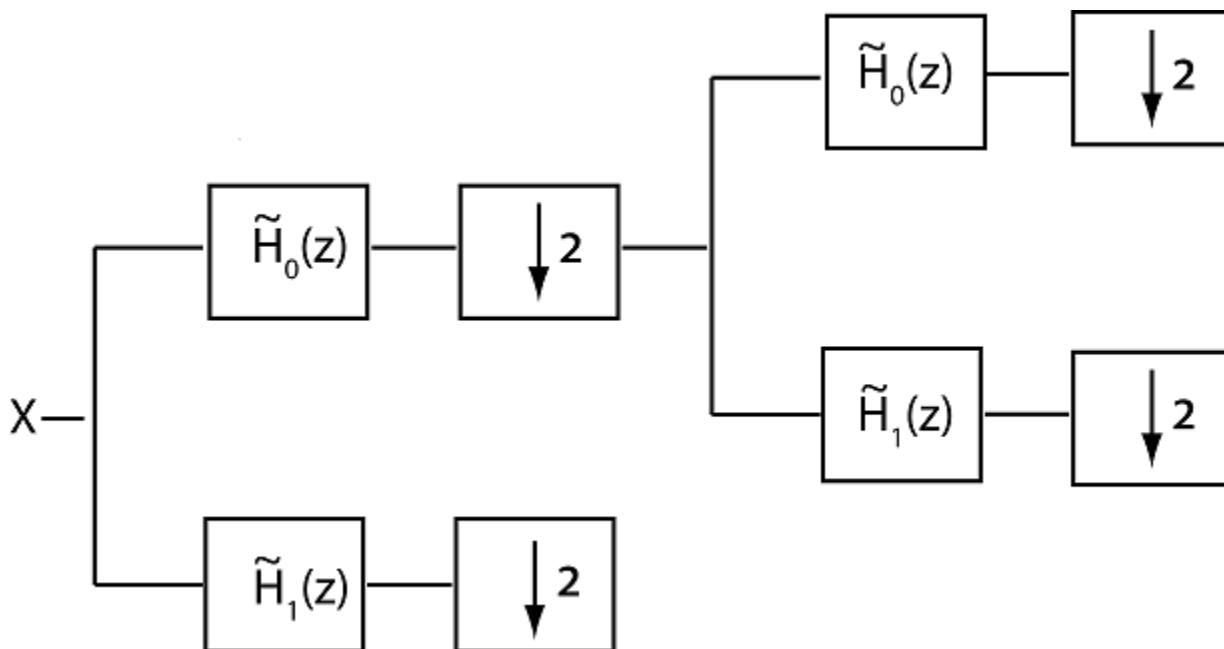
In this section...

“Double-Density Wavelet Transform” on page 3-3

“Dual-Tree Complex Wavelet Transform” on page 3-6

“Dual-Tree Double-Density Wavelet Transforms” on page 3-9

Wavelet filter banks are special cases of multirate filter banks called tree-structured filter banks. In a filter bank, two or more filters are applied to an input signal and the filter outputs are typically downsampled. The following figure illustrates two stages, or levels, of a critically sampled two-channel tree-structured analysis filter bank. The filters are depicted in the z domain.

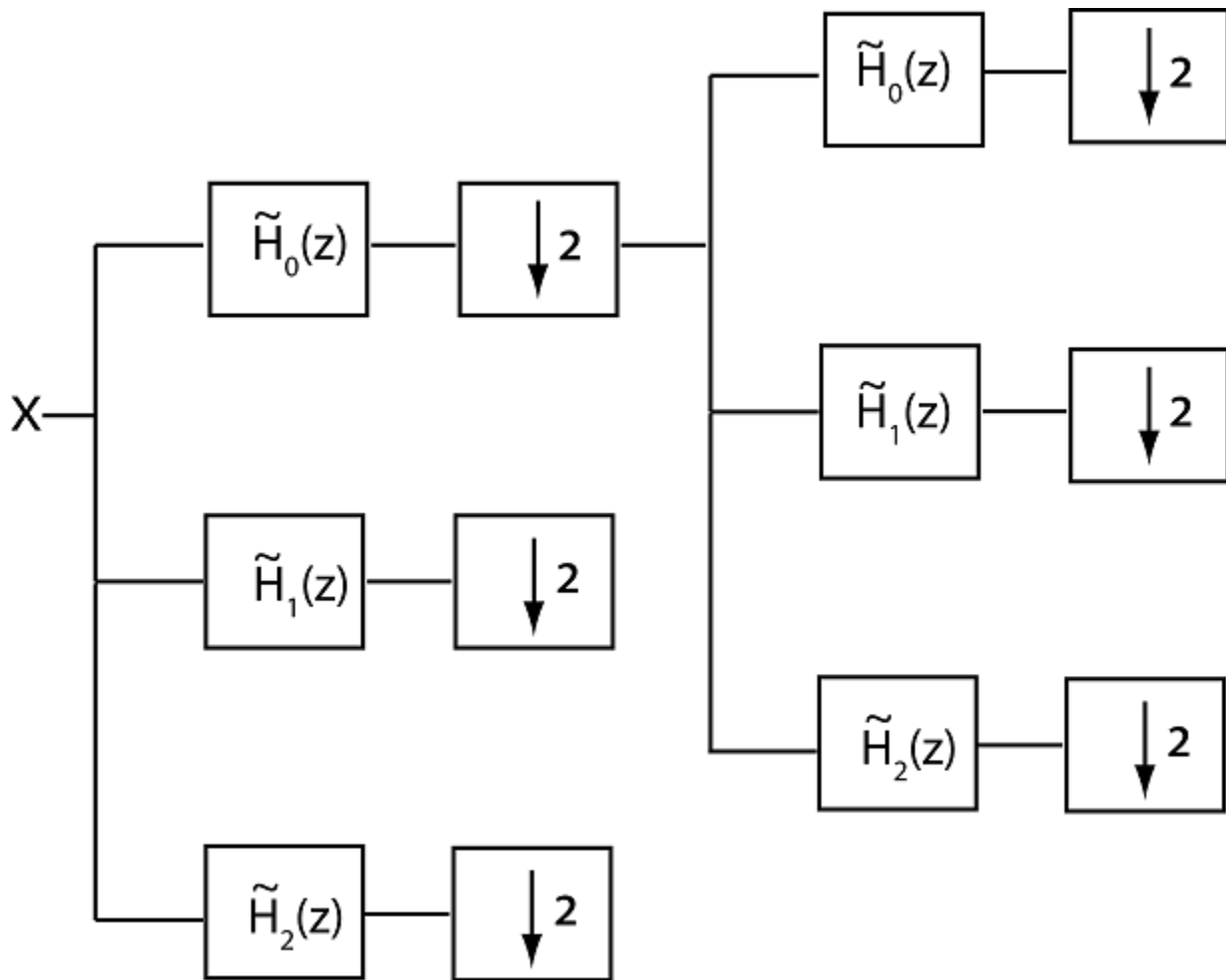


The filter system functions, $\tilde{H}_0(z)$ and $\tilde{H}_1(z)$, are typically designed to approximately partition the input signal, X , into disjoint subbands. In wavelet tree-structured filter

banks, the filter $\tilde{H}_0(z)$ is a lowpass, or scaling, filter, with a non-zero frequency response on the interval $[-\pi/2, \pi/2]$ radians/sample or $[-1/4, 1/4]$ cycles/sample. The filter $\tilde{H}_1(z)$ is a highpass, or wavelet, filter, with a non-zero frequency response on the interval $[-\pi, -\pi/2] \cup [\pi/2, \pi]$ radians/sample or $[-1/2, -1/4] \cup [1/4, 1/2]$ cycles/sample. The filter bank iterates on the output of the lowpass analysis filter to obtain successive levels resulting into an approximate octave-band filtering of the input. The two analysis filters are not ideal, which results in aliasing that must be canceled by appropriately designed synthesis filters for perfect reconstruction. For an orthogonal filter bank, the union of the scaling filter and its even shifts and the wavelet filter and its even shifts forms an orthonormal basis for the space of square-summable sequences, $\ell^2(\mathbb{Z})$. The synthesis filters are the time-reverse and conjugates of the analysis filters. For biorthogonal filter banks, the synthesis filters and their even shifts form the reciprocal, or dual, basis to the analysis filters. With two analysis filters, downsampling the output of each analysis filter by two at each stage ensures that the total number of output samples equals the number of input samples. The case where the number of analysis filters is equal to the downsampling factor is referred to as *critical sampling*. An analysis filter bank where the number of channels is greater than the downsampling factor is an *oversampled* filter bank.

Double-Density Wavelet Transform

The following figure illustrates two levels of an oversampled analysis filter bank with three channels and a downsampling factor of two. The filters are depicted in the z domain.



Assume the filter $\tilde{H}_0(z)$, is a lowpass half-band filter and the filters $\tilde{H}_1(z)$ and $\tilde{H}_2(z)$ are highpass half-band filters.

Assume the three filters together with the corresponding synthesis filters form a perfect reconstruction filter bank. If additionally, $\tilde{H}_1(z)$ and $\tilde{H}_2(z)$ generate wavelets that satisfy the following relation

$$\psi_1(t) = \psi_2(t - 1/2),$$

the filter bank implements the *double-density* wavelet transform. The preceding condition guarantees that the integer translates of one wavelet fall halfway between the integer translates of the second wavelet. In frame-theoretic terms, the double-density wavelet transform implements a tight frame expansion.

The following code illustrates the two wavelets used in the double-density wavelet transform.

```
x = zeros(256,1);
df = dtfilters('filters1');
wt1 = dddtree('ddt',x,5,df,df);
wt2 = dddtree('ddt',x,5,df,df);
wt1.cfs{5}(5,1,1) = 1;
wt2.cfs{5}(5,1,2) = 1;
wav1 = idddtree(wt1);
wav2 = idddtree(wt2);
plot(wav1); hold on;
plot(wav2,'r'); axis tight;
legend('\psi_1(t)', '\psi_2(t)')
```

You cannot choose the two wavelet filters arbitrarily to implement the double-density wavelet transform. The three analysis and synthesis filters must satisfy the perfect reconstruction (PR) conditions. For three real-valued filters, the PR conditions are

$$H_0(z)H_0(1/z) + H_1(z)H_1(1/z) + H_2(z)H_2(1/z) = 2$$

$$H_0(z)H_0(-1/z) + H_1(z)H_1(-1/z) + H_2(z)H_2(-1/z) = 0$$

You can obtain wavelet analysis and synthesis frames for the double-density wavelet transform with 6 and 12 taps using `dtfilters`.

```
[df1,sf1] = dtfilters('filters1');
[df2,sf2] = dtfilters('filters2');
```

`df1` and `df2` are three-column matrices containing the analysis filters. The first column contains the scaling filter and columns two and three contain the wavelet filters. The corresponding synthesis filters are in `sf1` and `sf2`.

See [4] and [5] for details on how to generate wavelet frames for the double-density wavelet transform.

The main advantages of the double-density wavelet transform over the critically sampled discrete wavelet transform are

- Reduced shift sensitivity
- Reduced rectangular artifacts in the 2-D transform
- Smoother wavelets for a given number of vanishing moments

The main disadvantages are

- Increased computational costs
- Non-orthogonal transform

Additionally, while exhibiting less shift sensitivity than the critically sampled DWT, the double-density DWT is not shift-invariant like the complex dual-tree wavelet transform. The double-density wavelet transform also lacks the directional selectivity of the oriented dual-tree wavelet transforms.

Dual-Tree Complex Wavelet Transform

The critically sampled discrete wavelet transform (DWT) suffers from a lack of shift invariance in 1-D and directional sensitivity in N-D. You can mitigate these shortcomings by using approximately analytic wavelets. An analytic wavelet is defined as

$$\psi_c(t) = \psi_r(t) + j\psi_i(t)$$

where j denotes the unit imaginary. The imaginary part of the wavelet, $\psi_i(t)$, is the Hilbert transform of the real part, $\psi_r(t)$. In the frequency domain, the analytic wavelet has support on only one half of the frequency axis. This means that the analytic wavelet $\psi_c(t)$ has only one half the bandwidth of the real-valued wavelet $\psi_r(t)$.

It is not possible to obtain exactly analytic wavelets generated by FIR filters. The Fourier transforms of compactly supported wavelets cannot vanish on any set of nonzero measure. This means that the Fourier transform cannot be zero on the negative frequency axis. Additionally, the efficient two-channel filter bank implementation of the DWT derives from the following perfect reconstruction condition for the scaling filter, $H_0(e^{j\omega})$, of a multiresolution analysis (MRA)

$$|H_0(e^{j\omega})|^2 + |H_0(e^{j(\omega + \pi)})|^2 = 2.$$

If the wavelet associated with an MRA is analytic, the scaling function is also analytic. This implies that

$$H_0(e^{j\omega}) = 0 \quad -\pi \leq \omega < 0,$$

from which it follows that $|H_0(e^{j\omega})|^2 = 2 \quad 0 \leq \omega \leq \pi$. The result is that the scaling filter is allpass.

The preceding results demonstrate that you cannot find a compactly support wavelet determined by FIR filters that is exactly analytic. However, you can obtain wavelets that are approximately analytic by combining two tree-structured filter banks as long as the filters in the dual-tree transform are carefully constructed to satisfy certain conditions [1], [6].

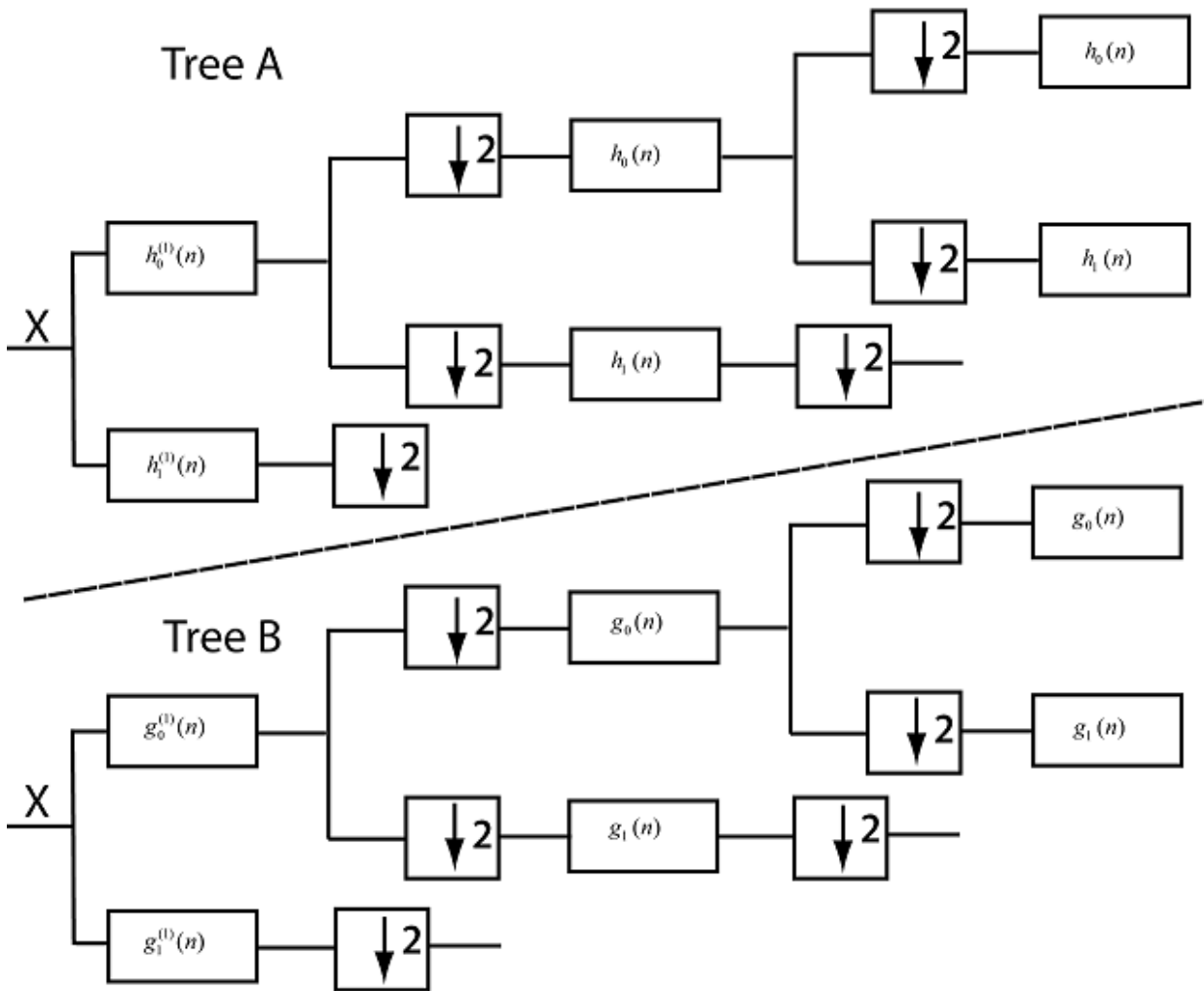
The dual-tree complex wavelet transform is implemented with two separate two-channel FIR filter banks. The output of one filter bank is considered to be the real part, while the output of the other filter bank is the imaginary part. Because the dual-tree complex wavelet transform uses two critically sampled filter banks, the redundancy is 2^d for a d -dimensional signal (image). There are a few critical considerations in implementing the dual-tree complex wavelet transform. For convenience, refer to the two trees as: Tree A and Tree B.

- The analysis filters in the first stage of each filter bank must differ from the filters used at subsequent stages in both trees. It is not important which scaling and wavelet filters you use in the two trees for stage 1. You can use the same first stage scaling and wavelet filters in both trees.
- The scaling filter in Tree B for stages ≥ 2 must approximate a 1/2 sample delay of the scaling filter in Tree A. The one-half sample delay condition is a necessary and sufficient condition for the corresponding Tree B wavelet to be the Hilbert transform of the Tree A wavelet.[3].

The following figure illustrates three stages of the analysis filter bank for the 1-D dual-tree complex wavelet transform. The FIR scaling filters for the two trees are denoted by $\{h_0(n), g_0(n)\}$. The FIR wavelet filters for the two trees are denoted by $\{h_1(n), g_1(n)\}$. The two scaling filters are designed to approximately satisfy the half-sample delay condition

$$g_0(n) = h_0(n - 1/2)$$

The superscript (1) denotes that the first-stage filters must differ from the filters used in subsequent stages. You can use any valid scaling-wavelet filter pair for the first stage. The filters $\{h_0(n), g_0(n)\}$ cannot be arbitrary scaling filters and provide the benefits of using approximately analytic wavelets.



2-D Dual-Tree Wavelet Transforms

The dual-tree wavelet transform with approximately analytic wavelets offers substantial advantages over the separable 2-D DWT for image processing. The traditional separable 2-D DWT suffers from checkerboard artifacts due to symmetric frequency support of real-valued (non-analytic) scaling functions and wavelets. Additionally, the critically sampled separable 2-D DWT lacks shift invariance just as the 1-D critically sampled DWT does. The

Wavelet Toolbox software supports two variants of the dual-tree 2-D wavelet transform, the real oriented dual-tree wavelet transform and the oriented 2-D dual-tree complex wavelet transform. Both are described in detail in [6].

The real oriented dual-tree transform consists of two separable (row and column filtering) wavelet filter banks operating in parallel. The complex oriented 2-D wavelet transform requires four separable wavelet filter banks and is therefore not technically a dual-tree transform. However, it is referred to as a dual-tree transform because it is the natural extension of the 1-D complex dual-tree transform. To implement the real oriented dual-tree wavelet transform, use the 'realdt' option in `dddtree2`. To implement the oriented complex dual-tree transform, use the 'cplxdt' option.

Both the real oriented and oriented complex dual-tree transforms are sensitive to directional features in an image. Only the oriented complex dual-tree transform is approximately shift invariant. Shift invariance is not a feature possessed by the real oriented dual-tree transform.

Dual-Tree Double-Density Wavelet Transforms

The dual-tree double-density wavelet transform combines the properties of the double-density wavelet transform and the dual-tree wavelet transform [2].

In 1-D, the dual-tree double-density wavelet transform consists of two three-channel filter banks. The two wavelets in each tree satisfy the conditions described in “Double-Density Wavelet Transform” on page 3-3. Specifically, the integer translates of one wavelet fall halfway between the integer translates of the second wavelet. Additionally, the wavelets in Tree B are the approximate Hilbert transform of the wavelets in Tree A. To implement the dual-tree double-density wavelet transform for 1-D signals, use the 'cplxdddtt' option in `dddtree`. Similar to the dual-tree wavelet transform, the dual-tree double-density wavelet transform provides both real oriented and complex oriented wavelet transforms in 2-D. To obtain the real oriented dual-tree double-density wavelet transform, use the 'realdddtt' option in `dddtree2`. To obtain the complex oriented dual-tree double-density wavelet transform, use the 'cplxdddtt' option.

References

- [1] Kingsbury, N.G. “Complex Wavelets for Shift Invariant Analysis and Filtering of Signals”. *Journal of Applied and Computational Harmonic Analysis*. Vol 10, Number 3, May 2001, pp. 234-253.

- [2] Selesnick, I. "The Double-Density Dual-Tree Wavelet Transform". *IEEE® Transactions on Signal Processing*. Vol. 52, Number 5, May 2004, pp. 1304-1314.
- [3] Selesnick, I. "The Design of Approximate Hilbert Transform Pairs of Wavelet Bases." *IEEE Transactions on Signal Processing*, Vol. 50, Number 5, pp. 1144-1152.
- [4] Selesnick, I. "The Double Density DWT" *Wavelets in Signal and Image Analysis: From Theory to Practice* (A.A Petrosian, F.G. Meyer, eds.). Norwell, MA: Kluwer Academic Publishers:, 2001.
- [5] Abdelnour, F. "Symmetric Wavelets Dyadic Siblings and Dual Frames" *Signal Processing*, Vol. 92, Number 5, 2012, pp. 1216-1225.
- [6] Selesnick, I., R.G Baraniuk, and N.G. Kingsbury. "The Dual-Tree Complex Wavelet Transform." *IEEE Signal Processing Magazine*. Vol. 22, Number 6, November, 2005, pp. 123-151.
- [7] Vetterli, M. "Wavelets, Approximation, and Compression". *IEEE Signal Processing Magazine*, Vol. 18, Number 5, September, 2001, pp. 59-73.

1-D Decimated Wavelet Transforms

This section takes you through the features of 1-D critically-sampled wavelet analysis using the Wavelet Toolbox software.

The toolbox provides these functions for 1-D 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

Denoising and Compression

Function Name	Purpose
ddencmp	Provide default values for denoising and compression
wbmpen	Penalized threshold for wavelet 1-D or 2-D denoising
wdcbm	Thresholds for wavelet 1-D using Birgé-Massart strategy
wdencmp	Wavelet denoising and compression
wden	Automatic wavelet denoising
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 Wavelet Analyzer app, 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.

1-D 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 the signal and select a portion for wavelet analysis.

```
load leleccum;
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.

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

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

- 3 Construct approximations and details from the coefficients.

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

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

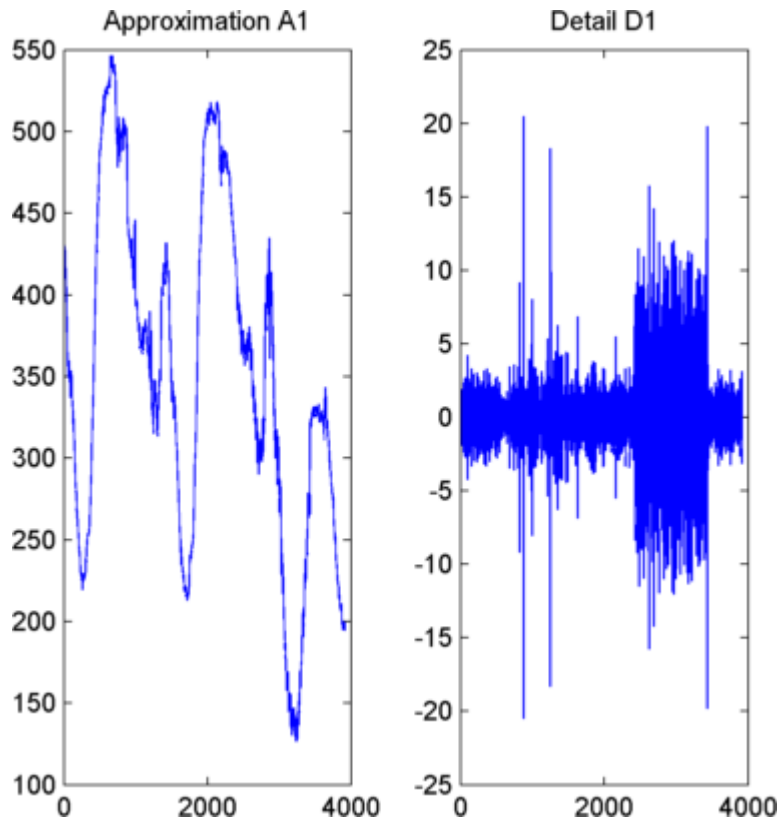
or

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

- 4 Display the approximation and detail.

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

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



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

To find the inverse transform, enter

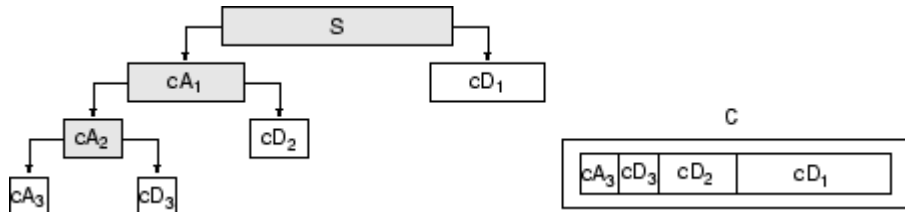
```
A0 = idwt(cA1,cD1,'db1',l_s);  
err = max(abs(s-A0))
```

- 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]);
```

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);
```

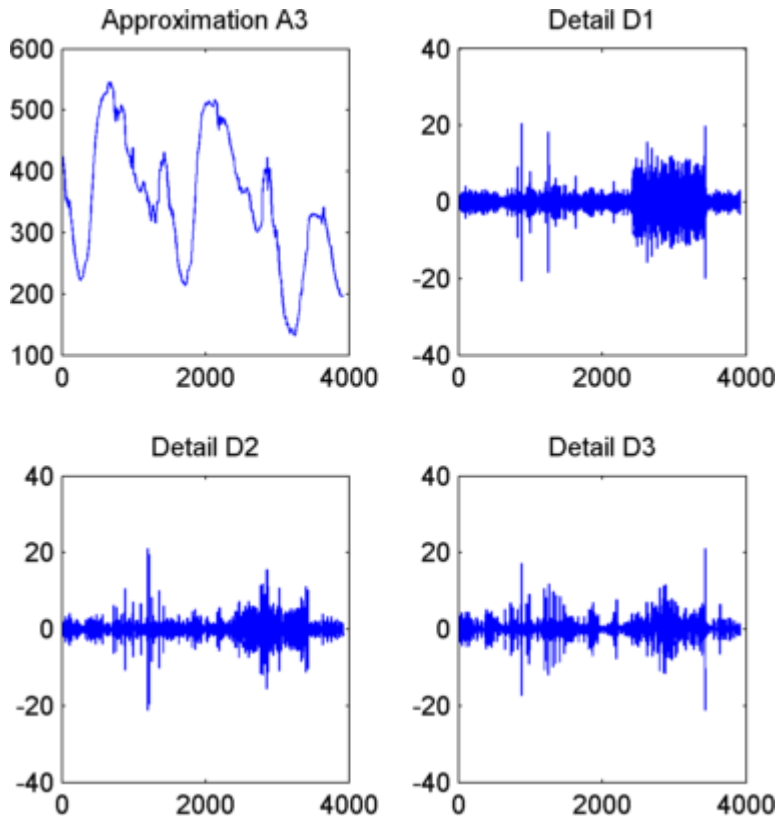
```
D3 = wrcoef('d',C,L,'db1',3);
```

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))
```

- 11** Crude denoising of a signal.

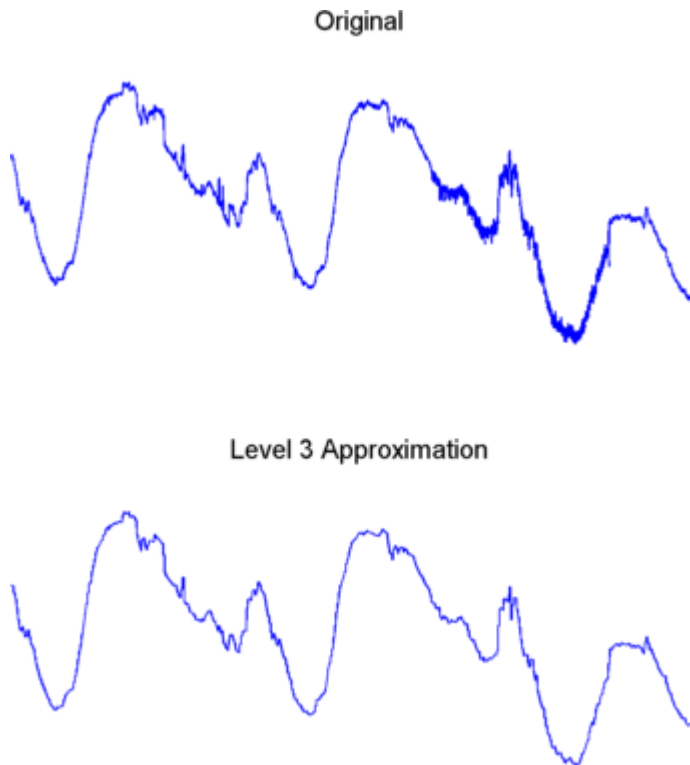
Using wavelets to remove noise from a signal requires identifying which component or components contain the noise, and then reconstructing the signal without those components.

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 denoising 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
```

Detail Level 1



Detail Level 2



Detail Level 3



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.

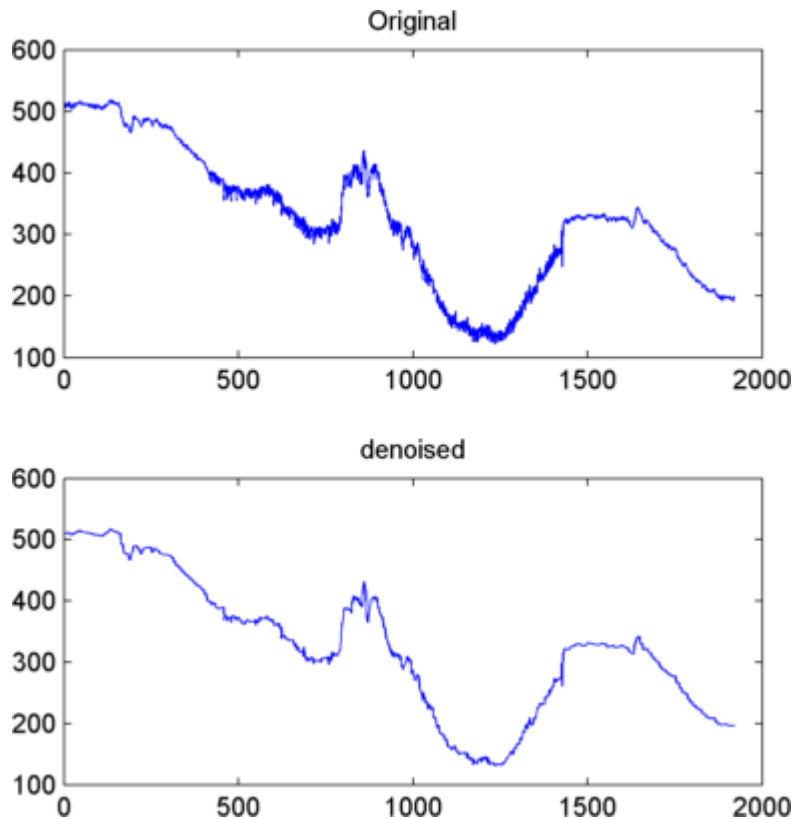
To denoise the signal, use the `ddencmp` command to calculate the default parameters and the `wdencomp` command to perform the actual denoising, type

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

Note that `wdencomp` uses the results of the decomposition (C and L) that we already calculated. We also specify that we used the `db1` wavelet to perform the original analysis, and we specify the global thresholding option `'gbl'`. See `ddencmp` and `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('denoised')
```



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

While using command line functions to remove the noise from a signal can be cumbersome, the software's Wavelet Analyzer app includes an easy-to-use denoising feature that includes automatic thresholding.

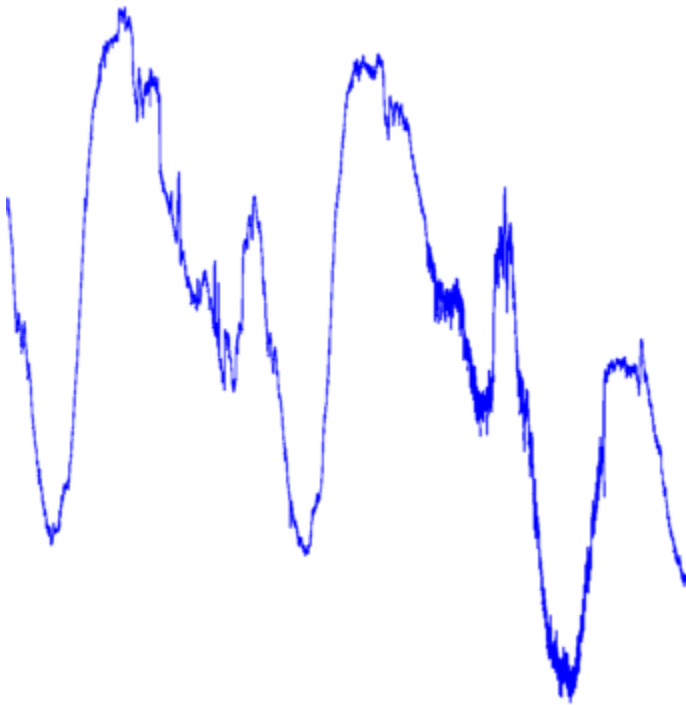
More information on the denoising process can be found in the following sections:

- "1-D Analysis Using the Wavelet Analyzer App" on page 3-21
- "Wavelet Denoising and Nonparametric Function Estimation" on page 6-2 in the *Wavelet Toolbox User's Guide*

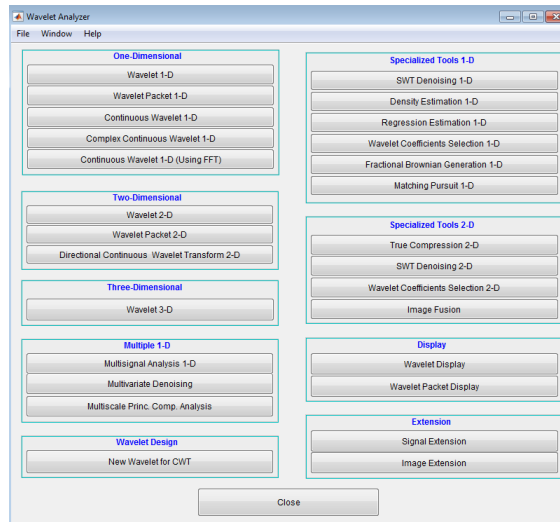
- “1-D Adaptive Thresholding of Wavelet Coefficients” on page 6-48
- “1-D Wavelet Variance Adaptive Thresholding” on page 6-16 in the *Wavelet Toolbox User's Guide*

1-D Analysis Using the Wavelet Analyzer App

In this section, we explore the same electrical consumption signal as in the previous section, but we use the Wavelet Analyzer app to analyze and denoise the signal.



- 1 Start the 1-D Wavelet Analysis Tool.
From the MATLAB prompt, type `waveletAnalyzer`.
The **Wavelet Analyzer** appears.



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

The discrete wavelet analysis tool for 1-D signal data appears.

- 2 Load a signal.

At the MATLAB command prompt, type

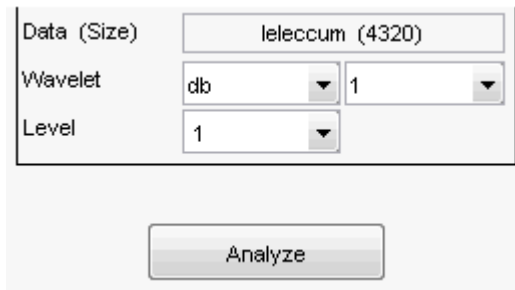
```
load leleccum;
```

In the **Wavelet 1-D** tool, select **File > Import from Workspace**. When the **Import from Workspace** dialog box appears, select the `leleccum` variable. Click **OK** to import the electrical consumption signal.

- 3 Perform a single-level wavelet decomposition.

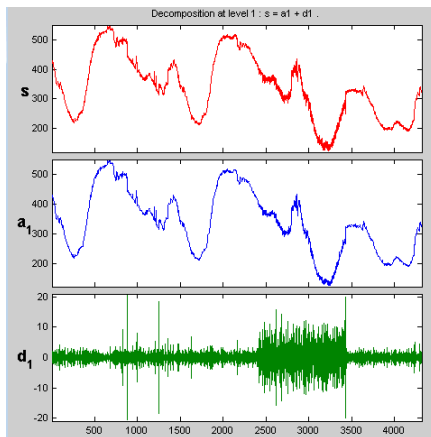
To start our analysis, let's perform a single-level decomposition using the `db1` wavelet, just as we did using the command-line functions in "1-D Analysis Using the Command Line" on page 3-13.

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



Click the **Analyze** button.

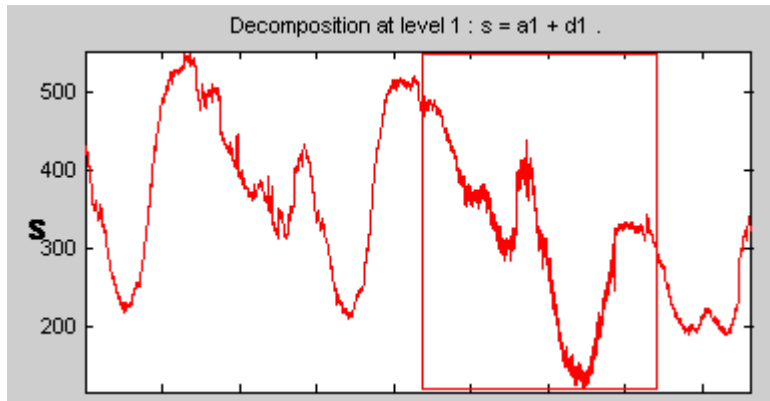
After a pause for computation, the tool displays the decomposition.



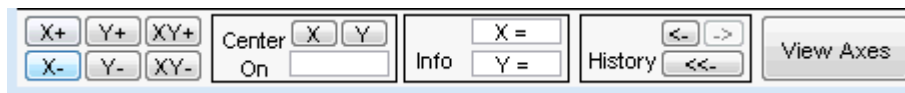
4 Zoom in on relevant detail.

One advantage of using the Wavelet Analyzer app is that you can zoom in easily on any part of the signal and examine it in greater detail.

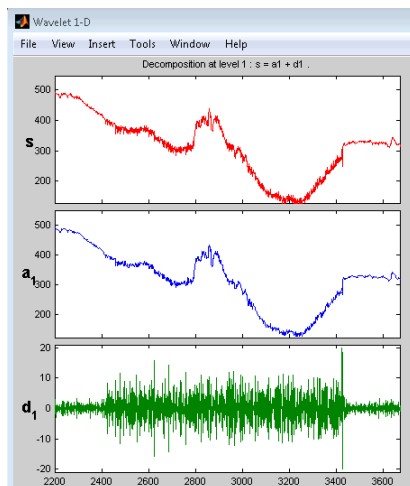
Drag a rubber band box (by holding down the left mouse button) over the portion of the signal you want to magnify. Here, we've selected the noisy part of the original signal.



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



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

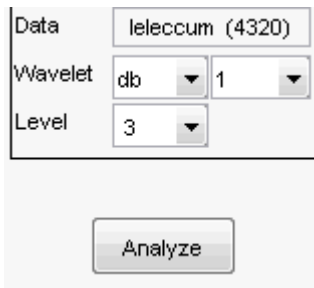


The other zoom controls do more or less what you'd expect them to. The **X-** button, for example, zooms out horizontally. The history function keeps track of all your views of the signal. Return to a previous zoom level by clicking the left arrow button.

5 Perform a multilevel decomposition.

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

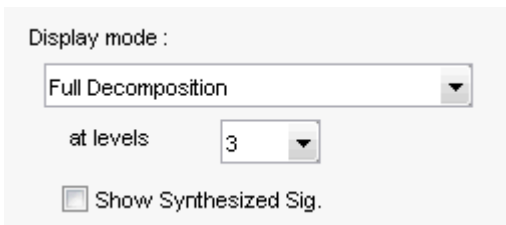
Select **3** from the **Level** menu at the upper right, and then click the **Analyze** button again.



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

Selecting Different Views of the Decomposition

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



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

- “Separate Mode,” which shows the details and the approximations in separate columns.
- “Superimpose Mode,” which shows the details on a single plot superimposed in different colors. The approximations are plotted similarly.

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

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 **Denoising** 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.

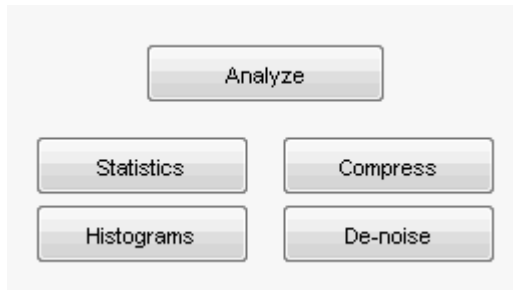


These options include the ability to suppress the display of various components, and to choose whether or not to display the original signal along with the details and approximations.

6 Remove noise from a signal.

The Wavelet Analyzer app features a denoising option with a predefined thresholding strategy. This makes it very easy to remove noise from a signal.

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

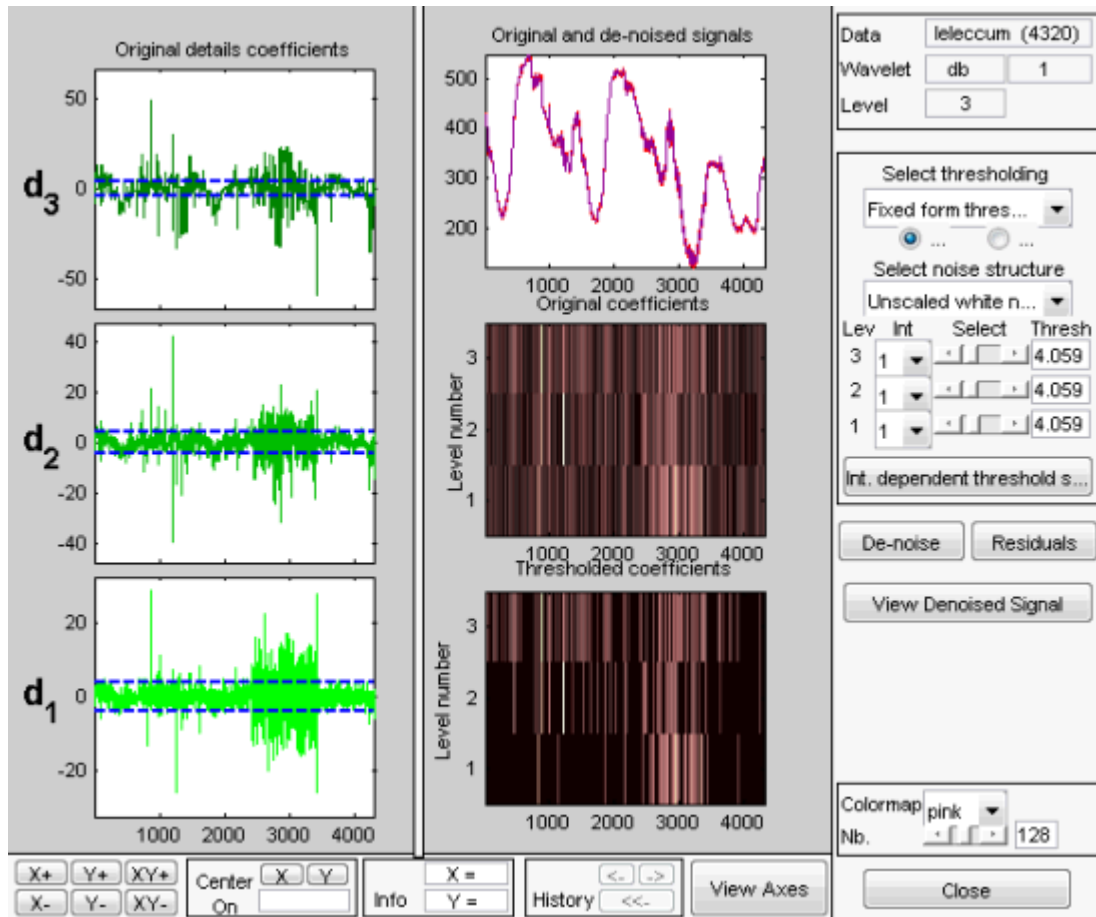


The **Wavelet 1-D Denoising** window appears.

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

Continue by clicking the **denoise** button.

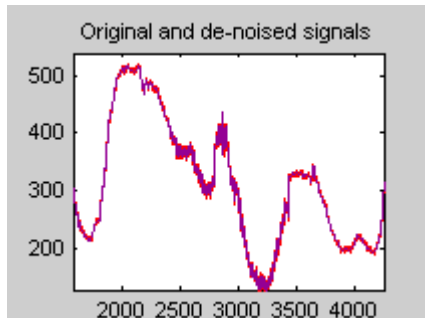
The denoised signal appears superimposed on the original. The tool also plots the wavelet coefficients of both signals.



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

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

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



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

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

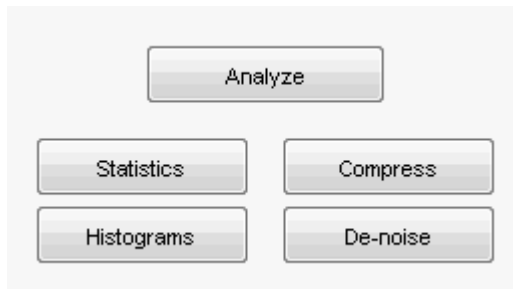
7 Refine the analysis.

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

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

8 Compress the signal.

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



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

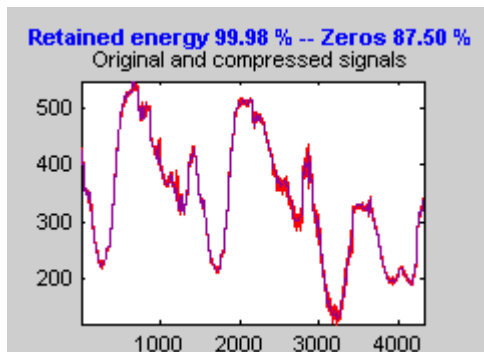
The **Compression** window appears.

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

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

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

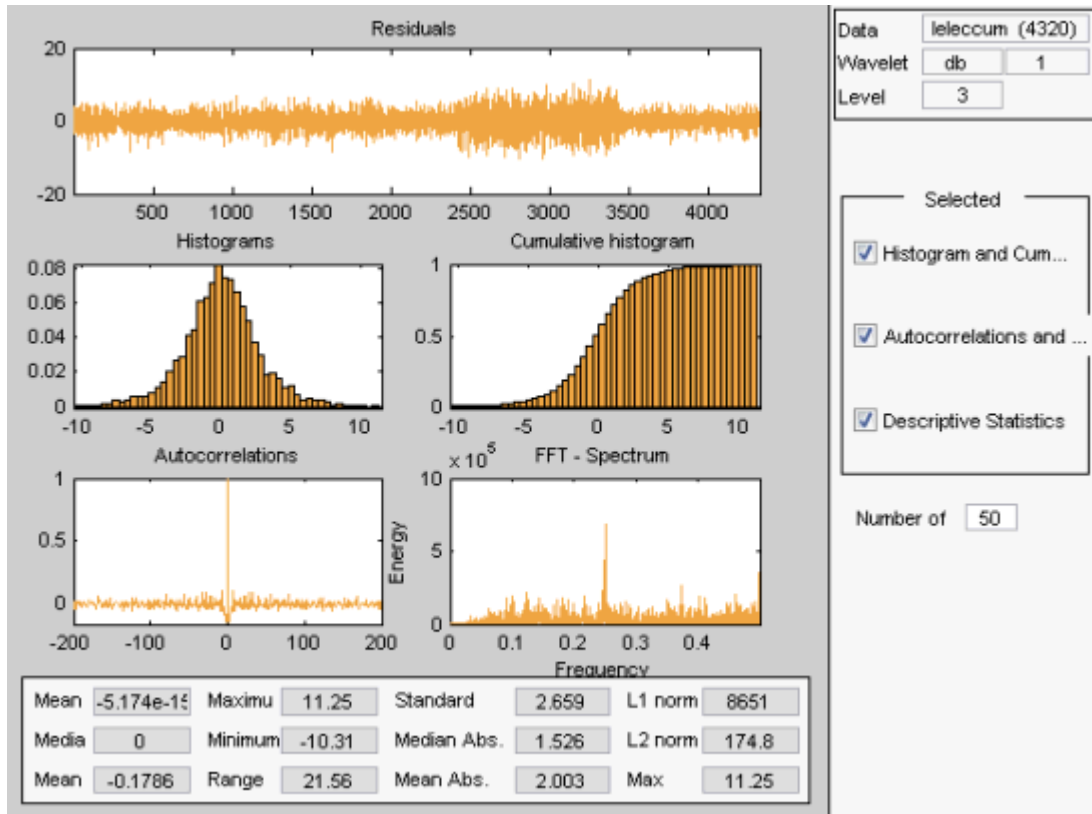
After a pause for computation, the electrical consumption signal is redisplayed in red with the compressed version superimposed in yellow. Below, we've zoomed in to get a closer look at the noisy part of the signal.



You can see that the compression process removed most of the noise, but preserved 99.99% of the energy of the signal.

- 9 Show the residuals.

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



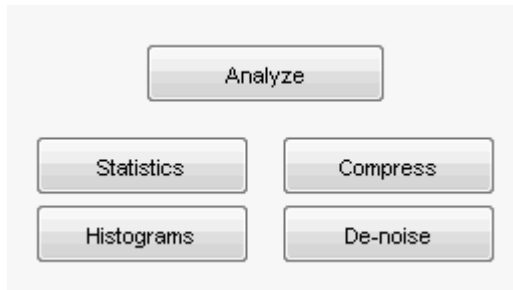
Displayed statistics include measures of tendency (mean, mode, median) and dispersion (range, standard deviation). In addition, the tool provides frequency-distribution diagrams (histograms and cumulative histograms), as well as time-series diagrams: autocorrelation function and spectrum. The same feature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

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

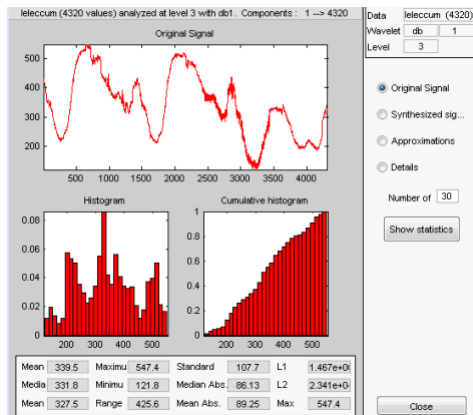
10 Show statistics.

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

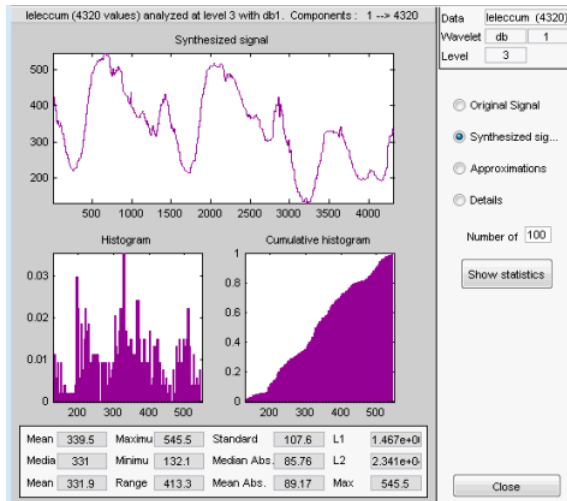
From the **Wavelet 1-D** tool, click the **Statistics** button.



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



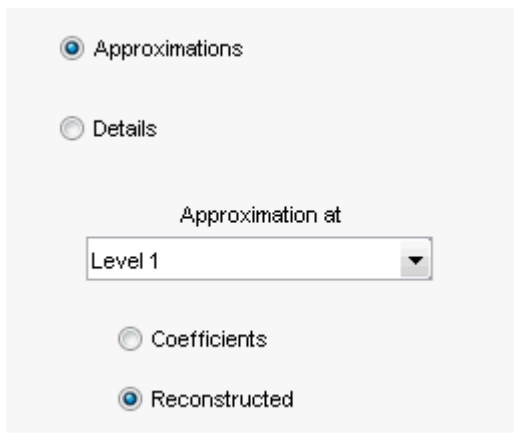
Select the synthesized signal or signal component whose statistics you want to examine. Click the appropriate option button, and then click the **Show Statistics** button. Here, we've chosen to examine the synthesized signal using 100 bins instead of 30, which is the default:



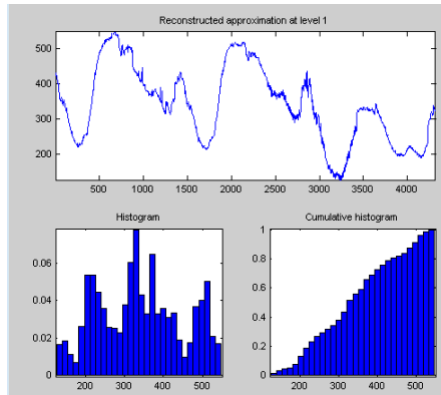
Displayed statistics include measures of tendency (mean, mode, median) and dispersion (range, standard deviation).

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.

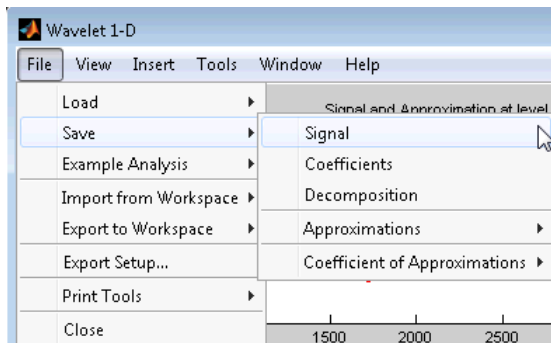


Importing and Exporting Information from the Wavelet Analyzer App

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

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 denoising operation on the original signal. When you close the **Denoising** 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 > Signal** menu option.

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

To load the signal into your workspace, simply type

```
load synthsig;
```

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

```
whos
```

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

The synthesized signal is given by the variable `synthsig`. In addition, the parameters of the denoising 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 “1-D Adaptive Thresholding of Wavelet Coefficients” on page 6-48).

For example, for level 1,

```
thrParams{1}
```

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

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

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

where the variable `valTHR` contains the global threshold:

```
valTHR
valTHR =
    1.2922
```

Saving Discrete Wavelet Transform Coefficients

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

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

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

Consider the example analysis:

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

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

```
load cantor
whos
```

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

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

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

Saving Decompositions

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

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

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

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

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

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

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

Name	Size	Bytes	Class
coefs	1x1023	8184	double array
data_name	1x6	12	char array
longs	1x7	56	double array

Name	Size	Bytes	Class
thrParams	0x0	0	double array
wave_name	1x3	6	char array

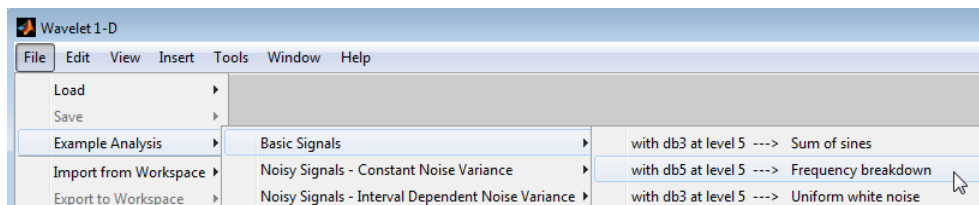
Note Save options are also available when performing denoising or compression inside the **Wavelet 1-D** tool. In the **Wavelet 1-D Denoising** window, you can save denoised signal and decomposition. The same holds true for the **Wavelet 1-D Compression** window. This way, you can save many different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

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

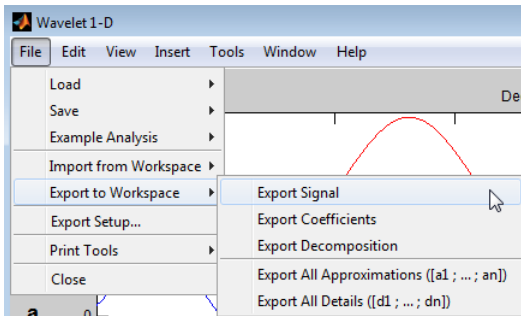
Export to Workspace

The **Wavelet 1-D** tool allows you to export your 1-D wavelet analysis to the MATLAB workspace in a number of formats.

For example, load the example analysis for the freqbrk signal.



After the wavelet 1-D analysis loads, select File → Export to Workspace.



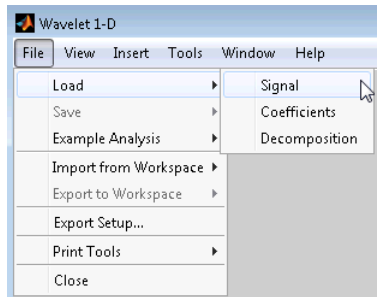
You have the option to

- **Export Signal** — This option exports the synthesized signal vector.
- **Export Coefficients** — This option exports the vector of wavelet and scaling coefficients, the bookkeeping vector, and the analyzing wavelet in a structure array. The wavelet and scaling coefficient and bookkeeping vectors are identical to the output of `wavedec`.
- **Export Decomposition** — This option is identical to **Export Coefficients** except that it also contains the name of the analyzed signal.
- **Export All Approximations** — This option exports a L-by-N matrix where L is the value of **Level** and N is the length of the input signal. Each row of the matrix is the projection onto the approximation space at the corresponding level. For example, the first row of the matrix is the projection onto the approximation space at level 1.
- **Export All Details** — This option exports a L-by-N matrix where L is the value of **Level** and N is the length of the input signal. Each row of the matrix is the projection onto the detail (wavelet) space at the corresponding level. For example, the first row of the matrix is the projection onto the detail space at level 1.

Loading Information into the Wavelet 1-D Tool

You can load signals, coefficients, or decompositions into the Wavelet Analyzer app. The information you load may have been previously exported from the app 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**.

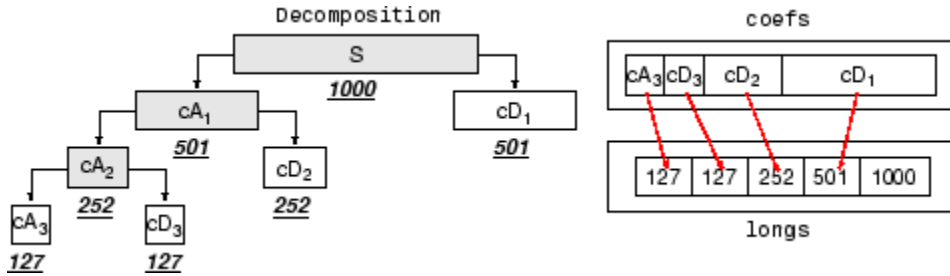
A dialog box appears that lets you select the appropriate MAT-file to be loaded.

Note The first 1-D variable encountered in the file is considered the signal. Variables are inspected in alphabetical order.

Loading Discrete Wavelet Transform Coefficients

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

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



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

```
save myfile coefs longs
```

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

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

Loading Decompositions

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

The MAT-file contains the following variables.

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

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

```
save myfile coefs longs wave_name
```

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

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

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

Fast Wavelet Transform (FWT) Algorithm

In 1988, Mallat produced a fast wavelet decomposition and reconstruction algorithm [1]. 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” on page 3-43
- “Algorithms” on page 3-46
- “Why Does Such an Algorithm Exist?” on page 3-51
- “1-D Wavelet Capabilities” on page 3-54
- “2-D Wavelet Capabilities” on page 3-55

Filters Used to Calculate the DWT and IDWT

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

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

All the filters used in DWT and IDWT are intimately related to the sequence

$$(w_n)_{n \in \mathbb{Z}}$$

Clearly if φ is compactly supported, the sequence (w_n) is finite and can be viewed as a filter. The filter W , which is called the scaling filter (nonnormalized), is

- Finite Impulse Response (FIR)

- Of length $2N$
- Of sum 1
- Of norm $\frac{1}{\sqrt{2}}$
- Of norm 1
- A low-pass filter

For example, for the db3 scaling filter,

```
load db3
db3
    db3 =
    0.2352    0.5706    0.3252   -0.0955   -0.0604    0.0249

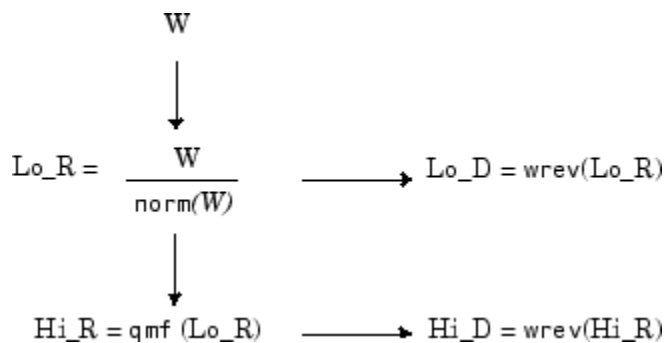
sum(db3)
    ans =
    1.0000

norm(db3)
    ans =
    0.7071
```

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

Filters	Low-Pass	High-Pass
Decomposition	Lo_D	Hi_D
Reconstruction	Lo_R	Hi_R

The four filters are computed using the following scheme.



where qmf is such that Hi_R and Lo_R are quadrature mirror filters (i.e., $Hi_R(k) = (-1)^k Lo_R(2N + 1 - k)$ for $k = 1, 2, \dots, 2N$).

Note that `wrev` flips the filter coefficients. So Hi_D and Lo_D are also quadrature mirror filters. The computation of these filters is performed using `orthfilt`. Next, we illustrate these properties with the `db6` wavelet.

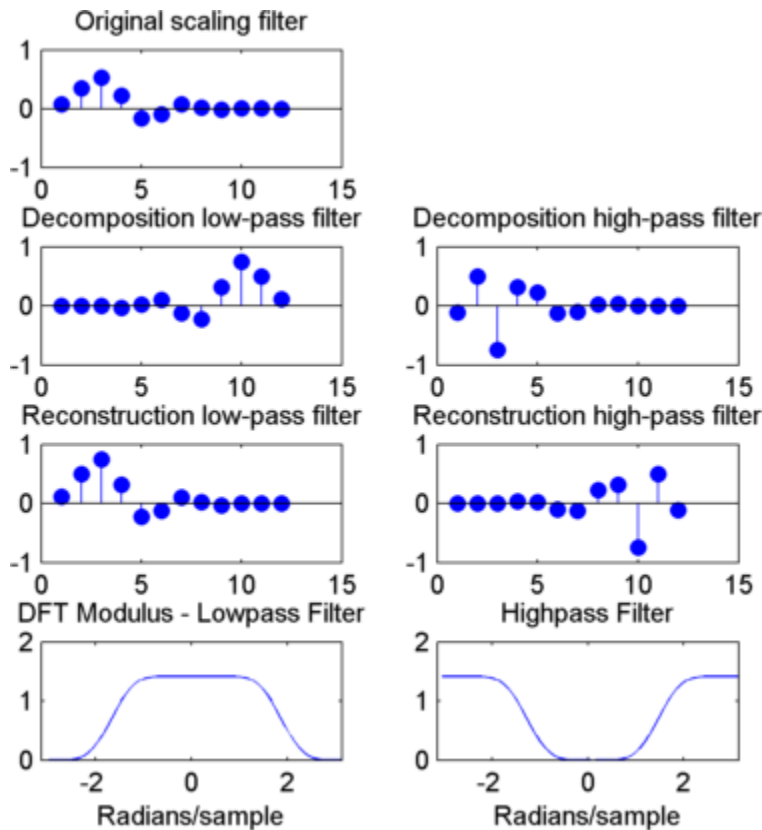
Load the Daubechies' extremal phase scaling filter and plot the coefficients.

```
load db6;
subplot(421); stem(db6, 'markerfacecolor', [0 0 1]);
title('Original scaling filter');
```

Use `orthfilt` to return the analysis (decomposition) and synthesis (reconstruction) filters.

Obtain the discrete Fourier transforms (DFT) of the lowpass and highpass analysis filters. Plot the modulus of the DFT.

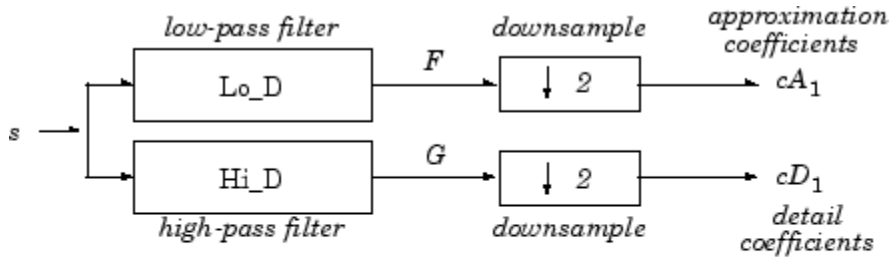
```
LoDFT = fft(Lo_D, 64);
HiDFT = fft(Hi_D, 64);
freq = -pi+(2*pi)/64:(2*pi)/64:pi;
subplot(427); plot(freq, fftshift(abs(LoDFT)));
set(gca, 'xlim', [-pi, pi]); xlabel('Radians/sample');
title('DFT Modulus - Lowpass Filter')
subplot(428); plot(freq, fftshift(abs(HiDFT)));
set(gca, 'xlim', [-pi, pi]); xlabel('Radians/sample');
title('Highpass Filter');
```



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

X	Convolve with filter X.
↓ 2	Keep the even indexed elements (see dyaddown).

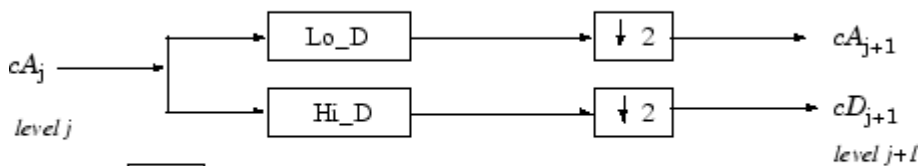
The length of each filter is equal to $2L$. The result of convolving a length N signal with a length $2L$ filter is $N+2L-1$. Therefore, the signals F and G are of length $N + 2L - 1$. After downsampling by 2, the coefficient vectors cA_1 and cD_1 are of length

$$\left\lfloor \frac{N-1}{2} + L \right\rfloor.$$

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



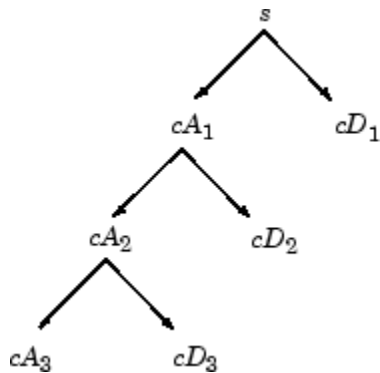
where

X	Convolve with filter X.
↓ 2	Downsample.

Initialization $cA_0 = s$.

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

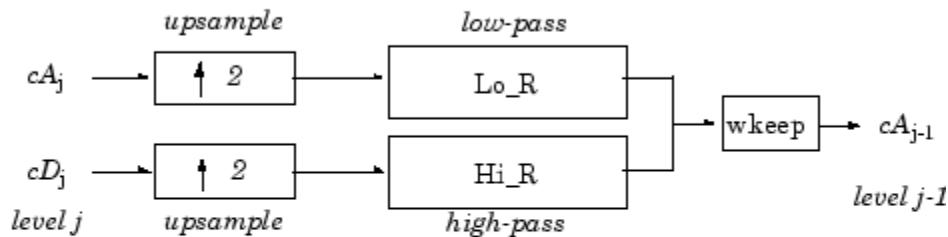
This structure contains for $J = 3$ the terminal nodes of the following tree.



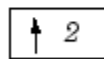
- Conversely, starting from cA_j and cD_j , the IDWT reconstructs cA_{j-1} , inverting the decomposition step by inserting zeros and convolving the results with the reconstruction filters.

One-Dimensional IDWT

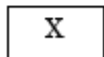
Reconstruction Step



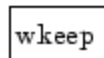
where



Insert zeros at odd-indexed elements.



Convolve with filter X.



Take the central part of U with the convenient length.

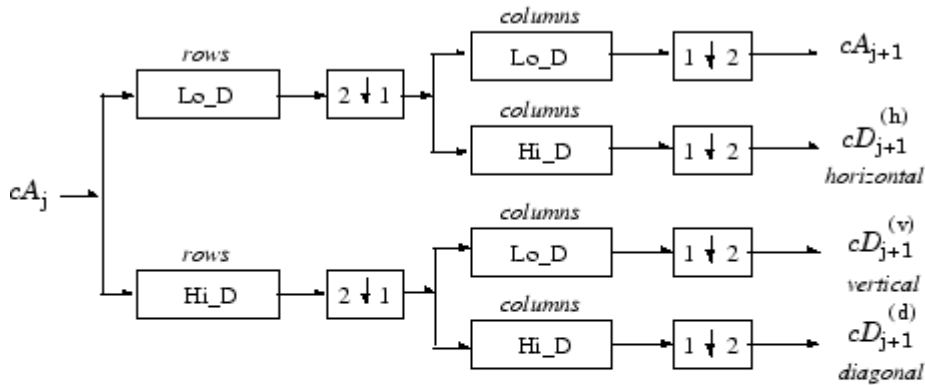
- For images, a similar algorithm is possible for two-dimensional wavelets and scaling functions obtained from 1-D wavelets by tensorial product.

This kind of 2-D 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 $\begin{matrix} \boxed{2 \downarrow 1} \end{matrix}$ Downsample columns: keep the even indexed columns.

$\begin{matrix} \boxed{1 \downarrow 2} \end{matrix}$ Downsample rows: keep the even indexed rows.

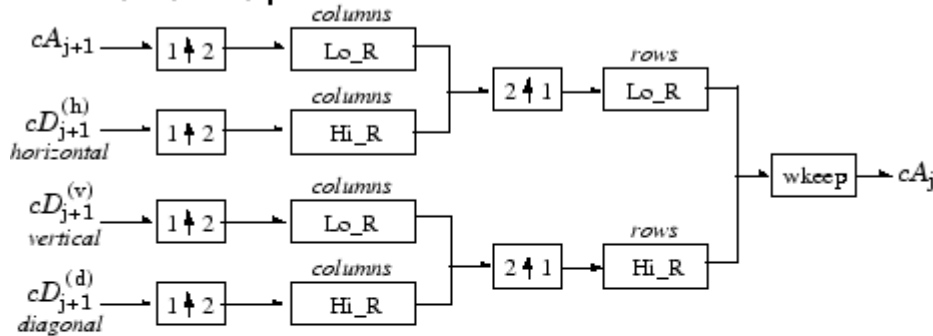
$\begin{matrix} \text{rows} \\ \boxed{X} \end{matrix}$ Convolve with filter X the rows of the entry.

$\begin{matrix} \text{columns} \\ \boxed{X} \end{matrix}$ Convolve with filter X the columns of the entry.

Initialization $CA_0 = s$ for the decomposition initialization.

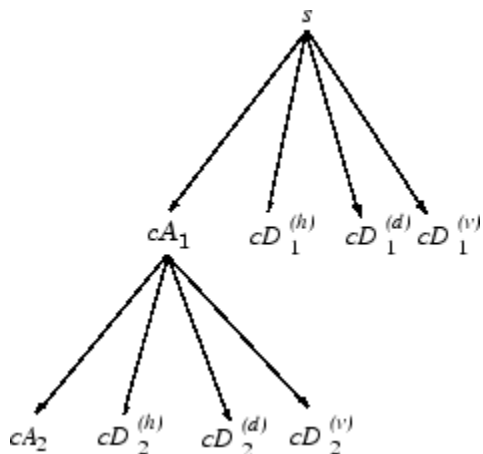
Two-Dimensional IDWT

Reconstruction Step



- where
- $\begin{matrix} \boxed{2 \uparrow 1} \\ \text{columns} \end{matrix}$ Upsample columns: insert zeros at odd-indexed columns.
 - $\begin{matrix} \boxed{1 \uparrow 2} \\ \text{rows} \end{matrix}$ Upsample rows: insert zeros at odd-indexed rows.
 - $\begin{matrix} \boxed{X} \\ \text{rows} \end{matrix}$ Convolve with filter X the rows of the entry.
 - $\begin{matrix} \boxed{X} \\ \text{columns} \end{matrix}$ Convolve with filter X the columns of the entry.

So, for $J = 2$, the 2-D 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 “Border Effects” on page 3-57.

Let us denote $h = \text{Lo_R}$ and $g = \text{Hi_R}$ and focus on the 1-D 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 ϕ	
$\phi_{1,0} = \sum_{k \in Z} h_k \phi_{0,k}$	$\phi_{j+1,0} = \sum_{k \in Z} h_k \phi_{j,k}$

This relation introduces the algorithm's h filter ($h_n = \sqrt{2\omega_n}$). For more information, see "Filters Used to Calculate the DWT and IDWT" on page 3-43.

- 5 We check that:

- a** The coordinate of $\varphi_{j+1,0}$ on $\varphi_{j,k}$ is h_k and does not depend on j .
- b** The coordinate of $\varphi_{j+1,n}$ on $\varphi_{j,k}$ is equal to $\langle \phi_{j+1,n}, \phi_{j,k} \rangle = h_{k-2n}$.
- 6** These relations supply the ingredients for the algorithm.
- 7** Up to now we used the filter h . The high-pass filter g is used in the twin scales relation linking the ψ and φ functions. Between two successive scales, we have the following twin-scale fundamental relation.

Twin-Scale Relation Between ψ and ϕ	
$\psi_{1,0} = \sum_{k \in \mathbb{Z}} g_k \phi_{0,k}$	$\psi_{j+1,0} = \sum_{k \in \mathbb{Z}} g_k \phi_{j,k}$

- 8** After the decomposition step, we justify now the reconstruction algorithm by building it. Let us simplify the notation. Starting from A_1 and D_1 , let us study $A_0 = A_1 + D_{j1}$. The procedure is the same to calculate $A = A_{j+1} + D_{j+1}$.

Let us define $\alpha_n, \delta_n, \alpha_k^0$ by

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

Let us assess the α_k^0 coordinates as

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

We will focus our study on the first sum $\sum_n a_n h_{k-2n}$; the second sum $\sum_n \delta_n g_{k-2n}$ is handled in a similar manner.

The calculations are easily organized if we note that (taking $k = 0$ in the previous formulas, makes things simpler)

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

If we transform the (α_n) sequence into a new sequence $(\tilde{\alpha}_n)$ defined by

..., $\alpha_{-1}, 0, \alpha_0, 0, \alpha_1, 0, \alpha_2, 0, \dots$ that is precisely

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

Then

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

and by extension

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

Since

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

the reconstruction steps are:

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

1-D Wavelet Capabilities

Basic 1-D 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	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	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

Purpose	Input	Output	File
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 1-D capabilities, see “1-D Analysis Using the Command Line” on page 3-13. .

2-D Wavelet Capabilities

Basic 2-D Objects

	Objects	Description
Image in original resolution	s	Original image
	A_0	Approximation at level 0

	Objects	Description
	$A_k, 1 \leq k \leq j$	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 2-D files are the same as those for the 1-D case, but with a 2 appended on the end of the command.

For example, `idwt` becomes `idwt2`. For more information, see “1-D Wavelet Capabilities” on page 3-54.

To illustrate command-line mode for 2-D capabilities, see “2-D Analysis — Command Line” on page 3-189..

References

- [1] Mallat, S. G. “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 11, Issue 7, July 1989, pp. 674-693.

Border Effects

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”). 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”).

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.

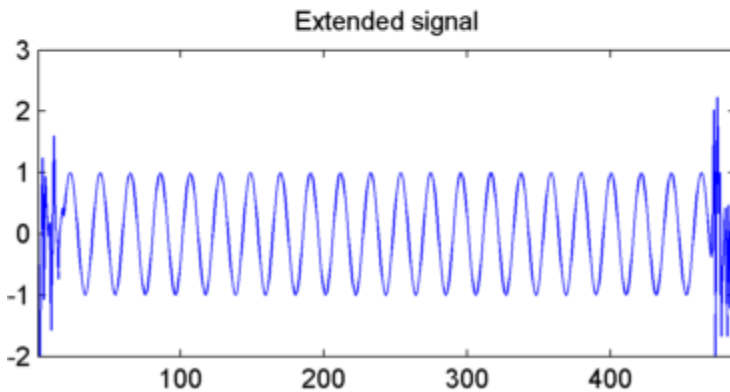
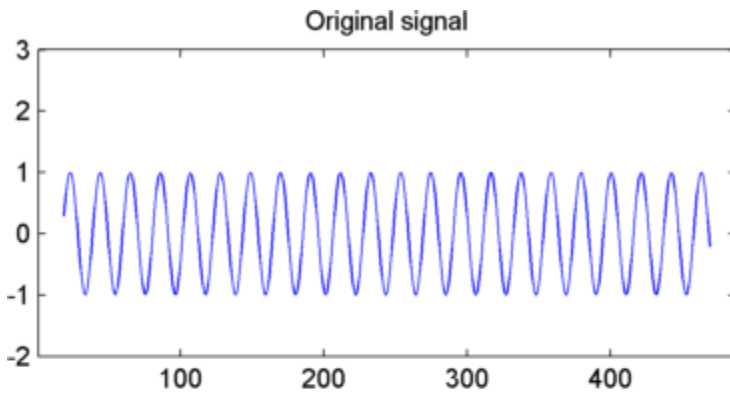
To ensure the crucial property **3** (perfect reconstruction) for arbitrary choices of

- 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);  
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))
```

Now let us illustrate the differences between the first three methods both for 1-D and 2-D signals.

Zero-Padding

Using the **Wavelet Analysis** app we will examine the effects of zero-padding.

- 1 From the MATLAB prompt, type

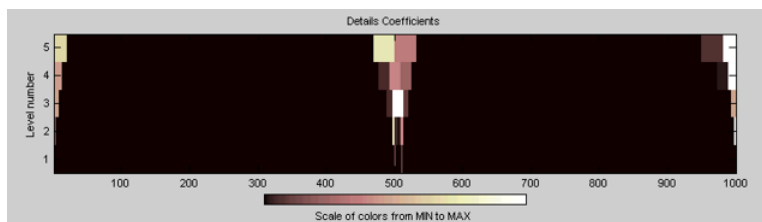
```
dwtmode('zpd')
```

- 2 From the MATLAB prompt, type `waveletAnalyzer`.

The **Wavelet Analyzer** appears.

- 3 Click the **Wavelet 1-D** menu item. The discrete wavelet analysis tool for 1-D 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 1-D 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 From the MATLAB prompt, type

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

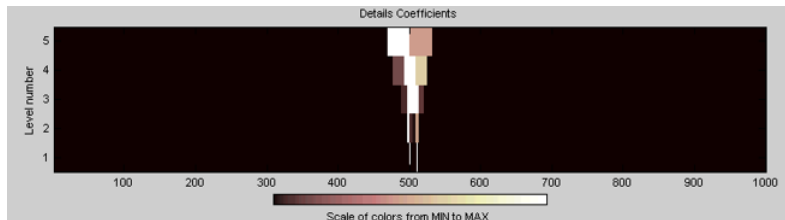
10 Click the **Wavelet 1-D** menu item.

The discrete wavelet analysis tool for 1-D signal data appears.

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

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

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

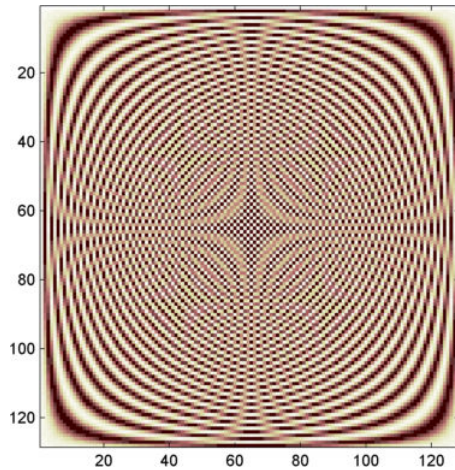


Let us now consider an image example.

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));
```

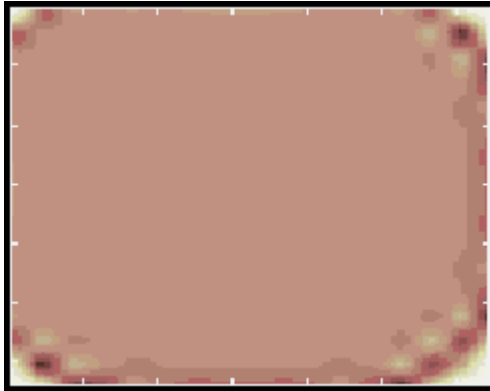


Symmetric Extension

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));
```

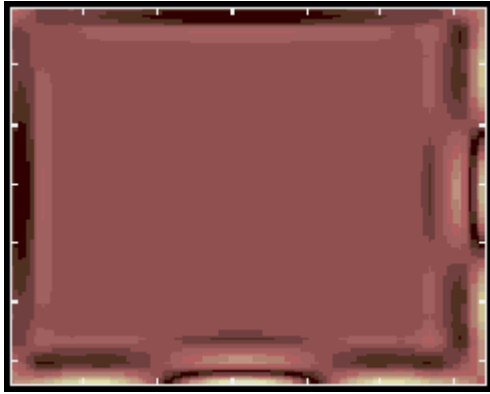


Smooth Padding

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));
```



Nondecimated Discrete Stationary Wavelet Transforms (SWTs)

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

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

The main application of the SWT is denoising. For more information on the rationale, see [CoiD95] in “References”. For examples, see “1-D Stationary Wavelet Transform” on page 3-73 and “2-D Stationary Wavelet Transform” on page 3-214.

The principle is to average several denoised signals. Each of them is obtained using the usual denoising scheme (see “Wavelet Denoising and Nonparametric Function Estimation” on page 6-2), but applied to the coefficients of an ε -decimated DWT.

Note The SWT is defined only for signals of length divisible by 2^J , where J is the maximum decomposition level. The SWT uses periodic (per) extension.

ε -Decimated DWT

What is an ε -decimated DWT?

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

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

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

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

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

How to Calculate the ε -Decimated DWT: SWT

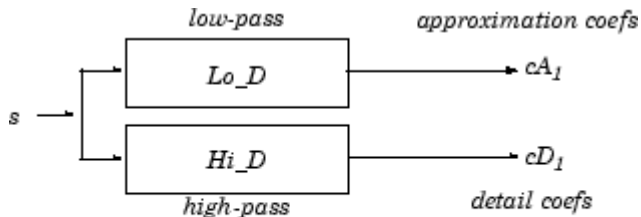
It is possible to calculate all the ε -decimated DWT for a given signal of length N , by computing the approximation and detail coefficients for every possible sequence ε . Do this using iteratively, a slightly modified version of the basic step of the DWT of the form:

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

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

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

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

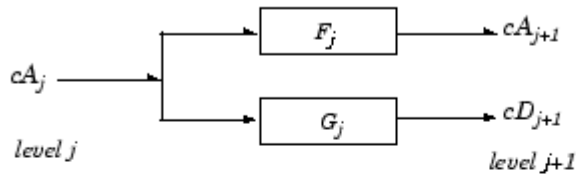


where: X Convolve with filter X

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.

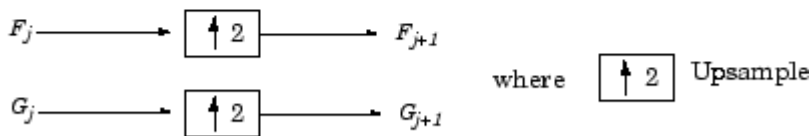
One-Dimensional SWT

Decomposition step



where \boxed{X} Convolve with filter X

Filter computation



where $\boxed{\uparrow 2}$ Upsample

Initialization

$$cA_0 = s \quad F_0 = Lo_D \quad G_0 = Hi_D$$

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

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

The function swt calculates successively the following arrays, where $A(j, \varepsilon_1, \dots, \varepsilon_j)$ or $D(j, \varepsilon_1, \dots, \varepsilon_j)$ denotes an approximation or a detail coefficient at level j obtained for the ε -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)	D(1,1)	D(1,0)	D(1,1)	D(1,0)	D(1,1)	D(1,0)	D(1,1)
A(1,0)	A(1,1)	A(1,0)	A(1,1)	A(1,0)	A(1,1)	A(1,0)	A(1,1)

Step 2

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

Step 3

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

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

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

where `circshift` performs a ε -circular shift of the input vector. Therefore, if $\varepsilon_{j+1} = 0$, the `circshift` 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 ε -decimated DWTs at level 3. Choosing ε , we can retrieve the corresponding ε -decimated DWT from the SWT array.

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

$C_\varepsilon =$

$A(3, \varepsilon_1, \varepsilon_2, \varepsilon_3)$	$D(3, \varepsilon_1, \varepsilon_2, \varepsilon_3)$	$D(2, \varepsilon_1, \varepsilon_2)$	$D(2, \varepsilon_1, \varepsilon_2)$	$D(1, \varepsilon_1)$	$D(1, \varepsilon_1)$	$D(1, \varepsilon_1)$	$D(1, \varepsilon_1)$
---	---	--------------------------------------	--------------------------------------	-----------------------	-----------------------	-----------------------	-----------------------

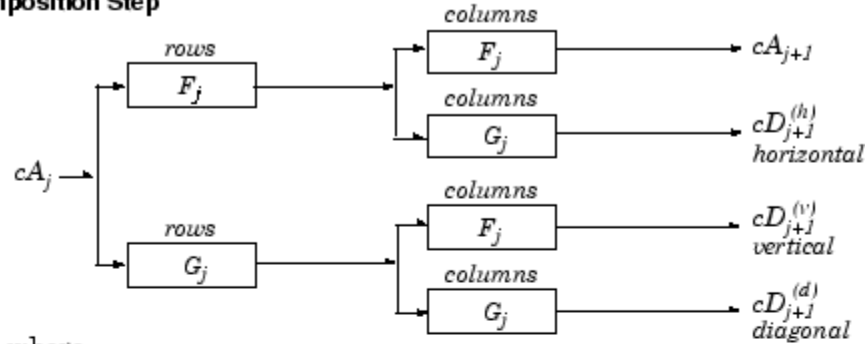
$$L_\varepsilon = [1, 1, 2, 4, 8]$$

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

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

Two-Dimensional SWT

Decomposition Step

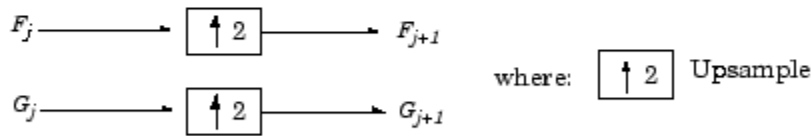


where

$\begin{matrix} \text{rows} \\ \boxed{X} \end{matrix}$ 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



Initialization

$cA_0 = s$ for the decomposition initialization

$F_0 = Lo_D$

$G_0 = Hi_D$

Note $size(cA_j) = size(cD_j^{(h)}) = size(cD_j^{(v)}) = size(cD_j^{(d)}) = s$
 Where $s = size\ of\ the\ analyzed\ image$

Inverse Discrete Stationary Wavelet Transform (ISWT)

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

To reconstruct the original signal using a given ε -decimated DWT characterized by $[\varepsilon_1, \dots, \varepsilon_j]$, 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 ε -decimated DWT. This can be done recursively, starting from level J down to level 1.

The ISWT is obtained with the following abstract algorithm:

```
FOR j = J:-1:1
  X0 = idwt(A(j,  $\varepsilon_1, \dots, \varepsilon_j$ ), 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 = circshift(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”.

1-D Stationary Wavelet Transform

This topic takes you through the features of 1-D discrete stationary wavelet analysis using the Wavelet Toolbox software. For more information see “Nondecimated Discrete Stationary Wavelet Transforms (SWTs)” on page 3-66 in the *Wavelet Toolbox User's Guide*.

The toolbox provides these functions for 1-D 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 Wavelet Analyzer app, 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.

1-D Analysis Using the Command Line

This example involves a noisy Doppler test signal.

- 1 Load a signal.

From the MATLAB prompt, type

```
load noisdopp
```

- 2 Set the variables. Type

```
s = noisdopp;
```

For the SWT, if a decomposition at level k is needed, 2^k must divide evenly into the length of the signal. If your original signal does not have the correct length, you can use the **Signal Extension** tool in the **Wavelet Analysis** app 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
```

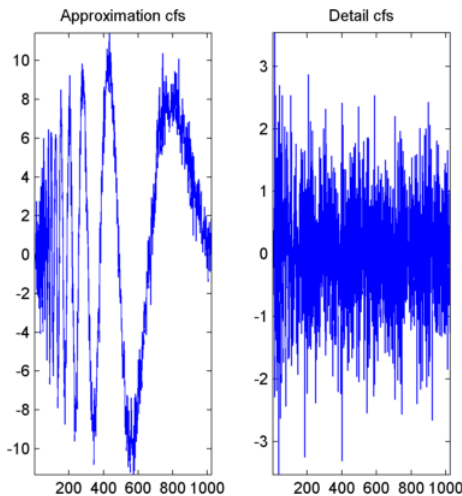
Name	Size	Bytes	Class
noisdopp	1x1024	8192	double array

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

- 4 Display the coefficients of approximation and detail.

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

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



- 5 Regenerate the signal by Inverse Stationary Wavelet Transform.

To find the inverse transform, type

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

To check the perfect reconstruction, type

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

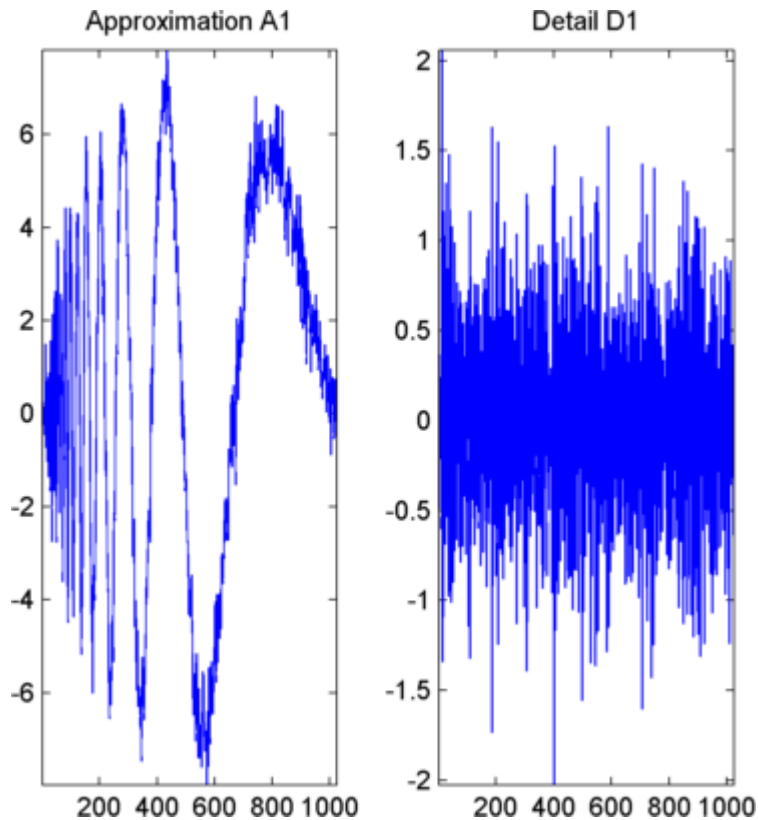
- 6 Construct and display approximation and detail from the coefficients.

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

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

To display the approximation and detail at level 1, type

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



7 Perform a multilevel Stationary Wavelet Decomposition.

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

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

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

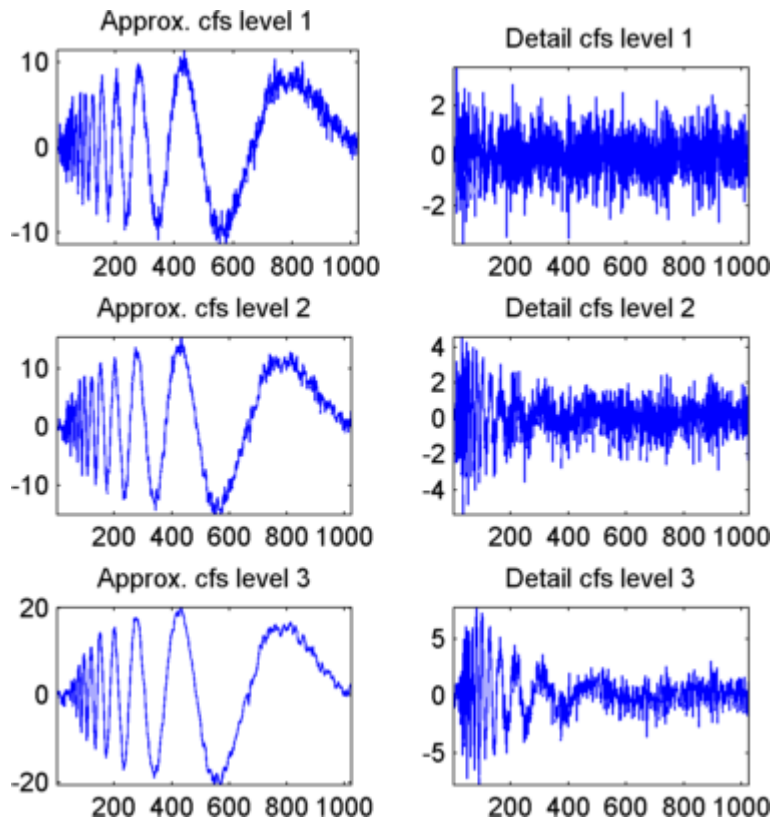
```
clear A0 A1 D1 err nulcfs
whos
```

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

- 8** Display the coefficients of approximations and details.

To display the coefficients of approximations and details, type

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



9 Reconstruct approximation at Level 3 From coefficients.

To reconstruct the approximation at level 3, type

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

10 Reconstruct details from coefficients.

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

```
D = mzero;
for i = 1:3
    swcfs = mzero;
    swcfs(i,:) = swd(i,:);
```

```
    D(i,:) = iswt(mzero,swcfs,'db1');  
end
```

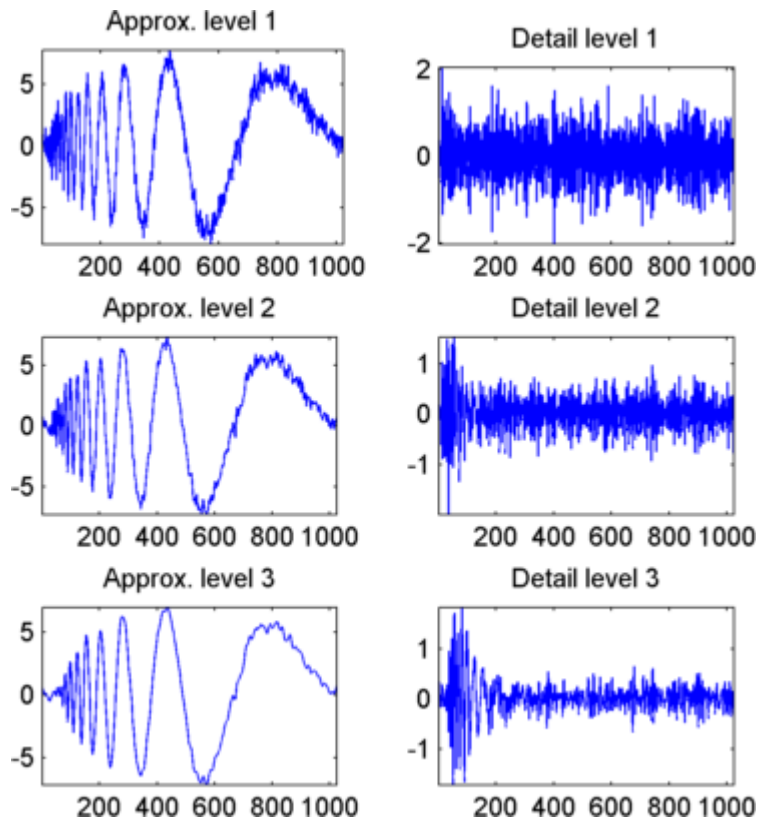
- 11** Reconstruct and display approximations at Levels 1 and 2 from approximation at Level 3 and details at Levels 2 and 3.

To reconstruct the approximations at levels 2 and 3, type

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

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

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

**12** Remove noise by thresholding.

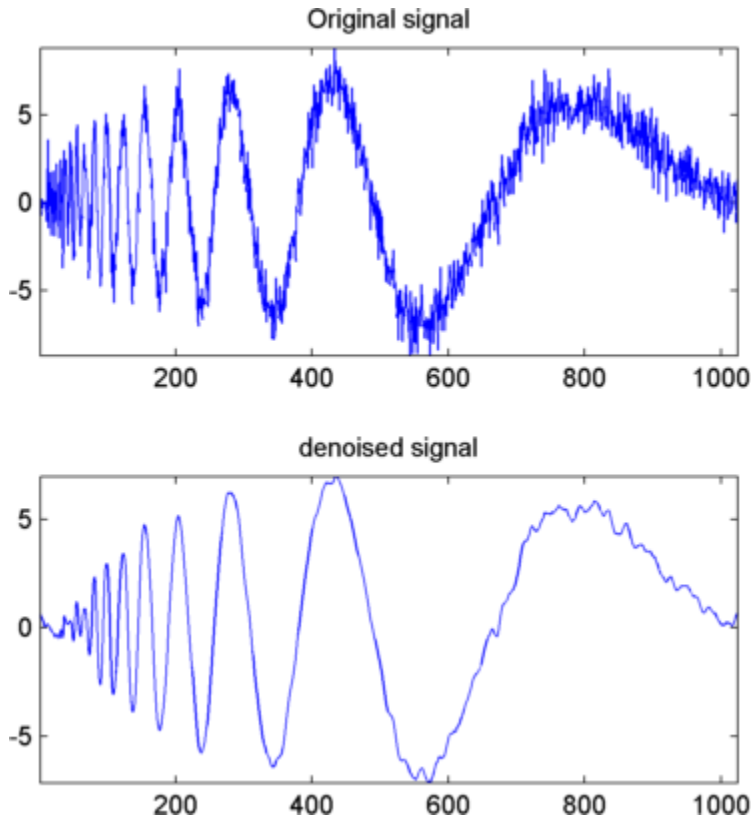
To denoise the signal, use the `ddencmp` command to calculate a default global threshold. Use the `wthresh` command to perform the actual thresholding of the detail coefficients, and then use the `iswt` command to obtain the denoised signal.

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

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

To display both the original and denoised signals, type

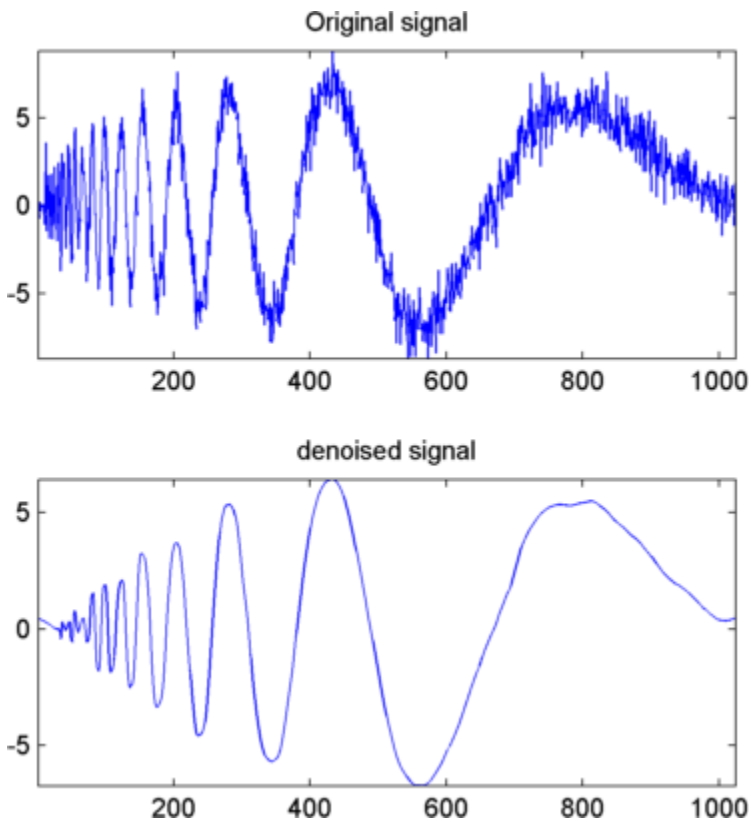
```
subplot(2,1,1), plot(s);
title('Original signal')
subplot(2,1,2), plot(clean);
title('denoised signal')
```



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

```
[swa,swd] = swt(s,5,'db1');
[thr,sorh] = ddenomp('den','wv',s);
dswd = wthresh(swd,sorh,thr);
clean = iswt(swa,dswd,'db1');
```

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



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

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

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

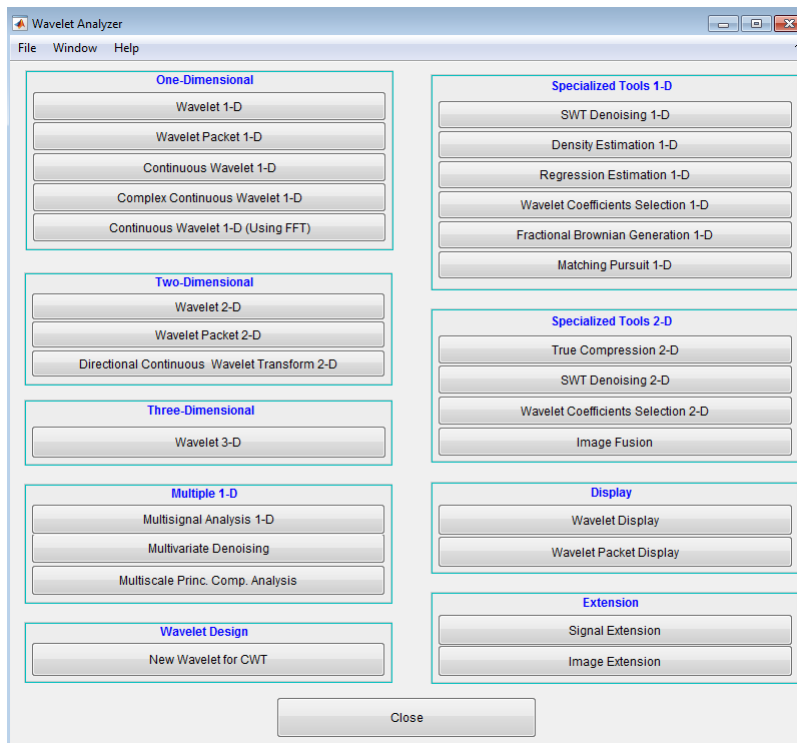
Interactive 1-D Stationary Wavelet Transform Denoising

Now we explore a strategy to denoise signals, based on the 1-D stationary wavelet analysis using the Wavelet Analyzer app. The basic idea is to average many slightly different discrete wavelet analyses.

- 1 Start the Stationary Wavelet Transform Denoising 1-D Tool.

From the MATLAB prompt, type `waveletAnalyzer`.

The **Wavelet Analyzer** appears.



Click the **SWT Denoising 1-D** menu item. The discrete stationary wavelet transform denoising tool for 1-D signals appears.

- 2 Load data.

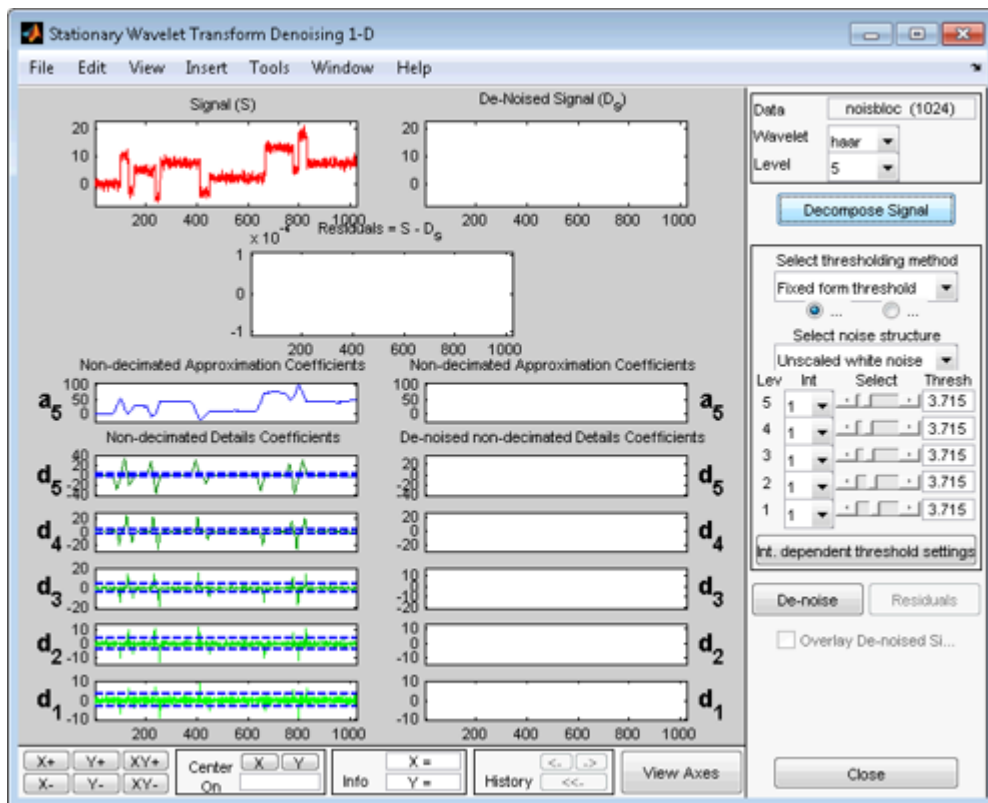
At the MATLAB command prompt, type

```
load noisbloc;
```

In the **SWT Denoising 1-D** tool, select **File > Import Signal from Workspace**. When the **Import from Workspace** dialog box appears, select the noisbloc variable. Click **OK** to import the noisy blocks signal.

- 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 3-43 in the *Wavelet Toolbox User's Guide*).



- 4 denoise the signal using the Stationary Wavelet Transform.

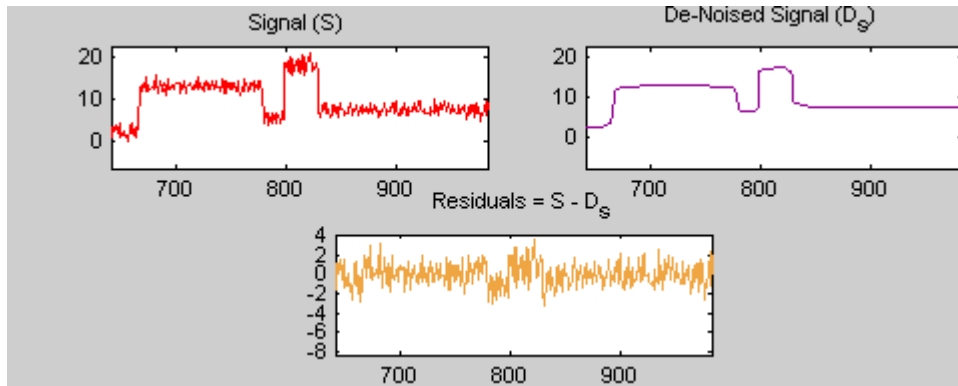
While a number of options are available for fine-tuning the denoising algorithm, we'll accept the defaults of fixed form soft thresholding and unscaled white noise. The sliders located on the right part of the window control the level-dependent thresholds, indicated by yellow dotted lines running horizontally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

Note that the approximation coefficients are not thresholded.

Click the **denoise** button.



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



Selecting a Thresholding Method

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

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

Importing and Exporting from the Wavelet Analysis App

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

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

```
load dnoibloc
whos
```

Name	Size	Bytes	Class
dnoibloc	1x1024	8192	double array

Name	Size	Bytes	Class
thrParams	1x5	580	cell array
wname	1x3	6	char array

The denoised signal is given by `dnoibloc`. In addition, the parameters of the denoising 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 “1-D Adaptive Thresholding of Wavelet Coefficients” on page 6-48.

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.

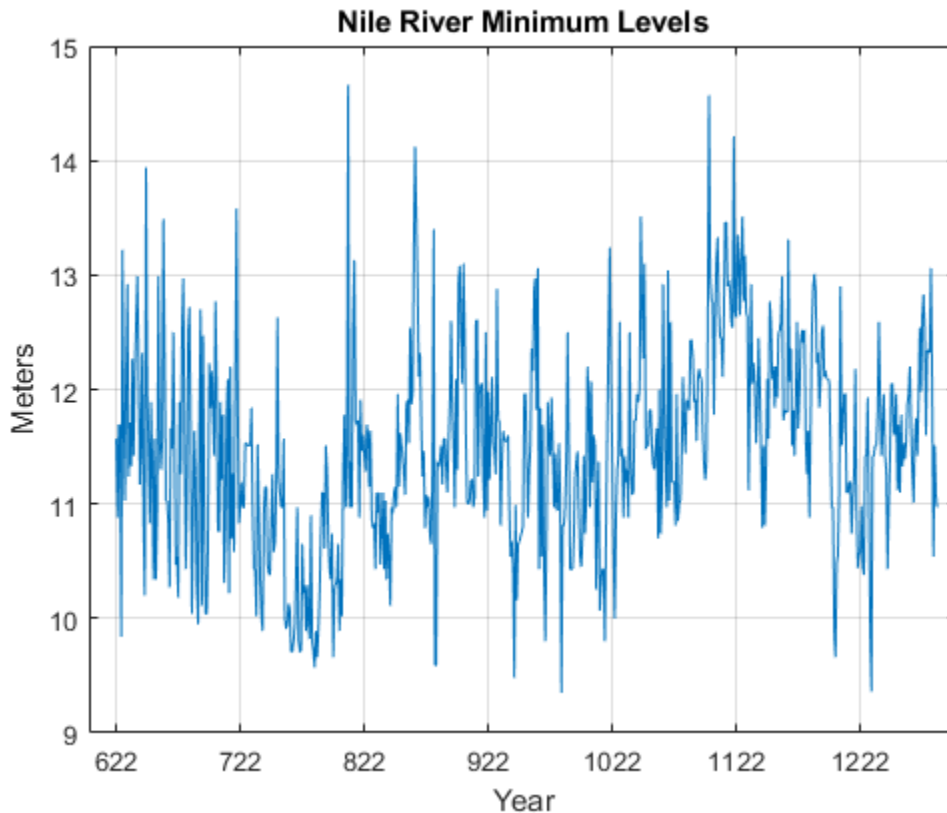
Wavelet Changepoint Detection

This example shows how to use wavelets to detect changes in the variance of a process. Changes in variance are important because they often indicate that something fundamental has changed about the data-generating mechanism.

The first example applies wavelet changepoint detection to a very old time series -- the Nile river minima data for the years 622 to 1281 AD. The river-level minima were measured at the Roda gauge near Cairo. Measurements are in meters.

Load and plot the data.

```
load nileriverminima
years = 622:1284;
figure
plot(years,nileriverminima)
title('Nile River Minimum Levels')
AX = gca;
AX.XTick = 622:100:1222;
grid on
xlabel('Year')
ylabel('Meters')
```



Construction began on a new measuring device around 715 AD. Examining the data prior to and after approximately 722 AD, there appears to be a change in the variability of the data. You can use wavelets to explore the hypothesis that the variability of the measurements has been affected by the introduction of a new measuring device.

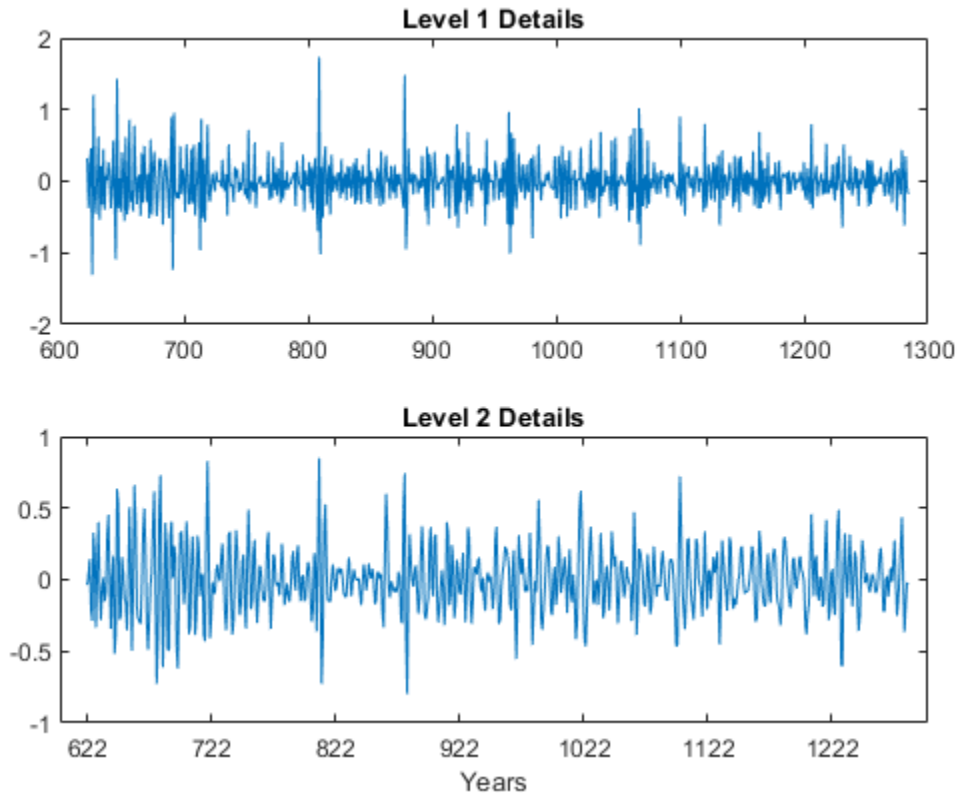
Obtain a multiresolution analysis (MRA) of the data using the Haar wavelet.

```
wt = modwt(nileriverminima, 'haar', 4);  
mra = modwtmra(wt, 'haar');
```

Plot the MRA and focus on the level-one and level-two details.

```
figure  
subplot(2,1,1)
```

```
plot(years,mra(1,:))
title('Level 1 Details')
subplot(2,1,2)
plot(years,mra(2,:))
title('Level 2 Details')
AX = gca;
AX.XTick = 622:100:1222;
xlabel('Years')
```



Apply an overall change of variance test to the wavelet coefficients.

```
for JJ = 1:5
    pts_0pt = wvarchg(wt(JJ,:),2);
    changepoints{JJ} = pts_0pt;
```



```
end
cellfun(@(x) ~isempty(x), changepoints, 'uni', 0)
```

```
ans =
```

```
1x5 cell array
```

```
{[1]} {[0]} {[0]} {[0]} {[0]}
```

Determine the year corresponding to the detected change of variance.

```
years(cell2mat(changepoints))
```

```
ans =
```

```
721
```

Split the data into two segments. The first segment includes the years 622 to 721 when the fine-scale wavelet coefficients indicate a change in variance. The second segment contains the years 722 to 1284. Obtain unbiased estimates of the wavelet variance by scale.

```
tspre = nileriverminima(1:100);
tspost = nileriverminima(101:end);
wpre = modwt(tspre, 'haar', 4);
wpost = modwt(tspost, 'haar', 4);
wvarpre = modwtvar(wpre, 'haar', 0.95, 'table')
wvarpost = modwtvar(wpost, 'haar', 0.95, 'table')
```

```
wvarpre =
```

```
5x4 table
```

	NJ	Lower	Variance	Upper
	—	—	—	—
D1	99	0.25199	0.36053	0.55846
D2	97	0.15367	0.25149	0.48477
D3	93	0.056137	0.11014	0.30622
D4	85	0.018881	0.047427	0.26453

```
S4    85    0.017875    0.0449    0.25044
```

```
wvarpost =
```

```
5x4 table
```

	NJ	Lower	Variance	Upper
D1	562	0.11394	0.13354	0.15869
D2	560	0.085288	0.10639	0.13648
D3	556	0.0693	0.094168	0.13539
D4	548	0.053644	0.081877	0.14024
S4	548	0.24608	0.37558	0.64329

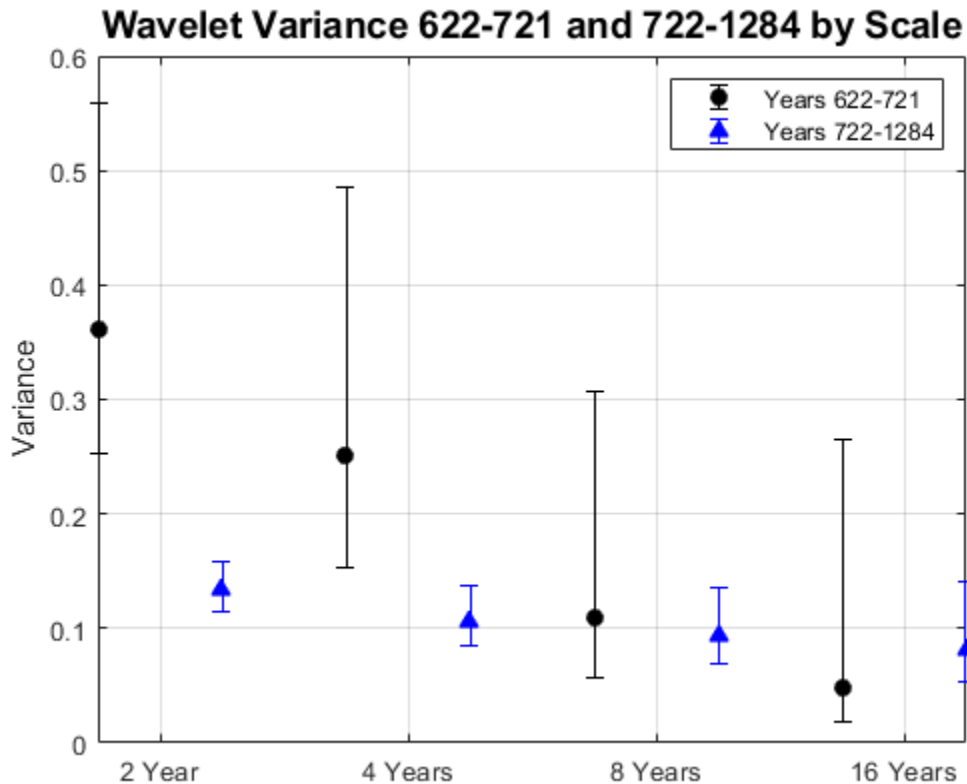
Compare the results.

```
Vpre = table2array(wvarpre);
Vpost = table2array(wvarpost);
Vpre = Vpre(1:end-1,2:end);
Vpost = Vpost(1:end-1,2:end);

Vpre(:,1) = Vpre(:,2)-Vpre(:,1);
Vpre(:,3) = Vpre(:,3)-Vpre(:,2);

Vpost(:,1) = Vpost(:,2)-Vpost(:,1);
Vpost(:,3) = Vpost(:,3)-Vpost(:,2);

figure
errorbar(1:4,Vpre(:,2),Vpre(:,1),Vpre(:,3),'ko',...
         'MarkerFaceColor',[0 0 0])
hold on
errorbar(1.5:4.5,Vpost(:,2),Vpost(:,1),Vpost(:,3),'b^',...
         'MarkerFaceColor',[0 0 1])
set(gca,'xtick',1.25:4.25)
set(gca,'xticklabel',{'2 Year','4 Years','8 Years','16 Years','32 Years'})
grid on
ylabel('Variance')
title('Wavelet Variance 622-721 and 722-1284 by Scale','fontsize',14)
legend('Years 622-721','Years 722-1284','Location','NorthEast')
```



The wavelet variance indicates a significant change in variance between the 622-721 and 722-1284 data over scales of 2 and 4 years.

The above example used the Haar wavelet filter with only two coefficients because of concern over boundary effects with the relatively small time series (100 samples from 622-721). If your data are approximately first or second-order difference stationary, you can substitute the biased estimate using the 'reflection' boundary. This permits you to use a longer wavelet filter without worrying about boundary coefficients. Repeat the analysis using the default 'sym4' wavelet.

```
wpre = modwt(tspre,4,'reflection');
wpost = modwt(tspost,4,'reflection');
wvarpre = modwtvar(wpre,[],[],'EstimatorType','biased',...
    'Boundary','reflection','table');
```

```
wvarpost = modwtvar(wpost,[],[],'EstimatorType','biased',...  
    'Boundary','reflection','table');
```

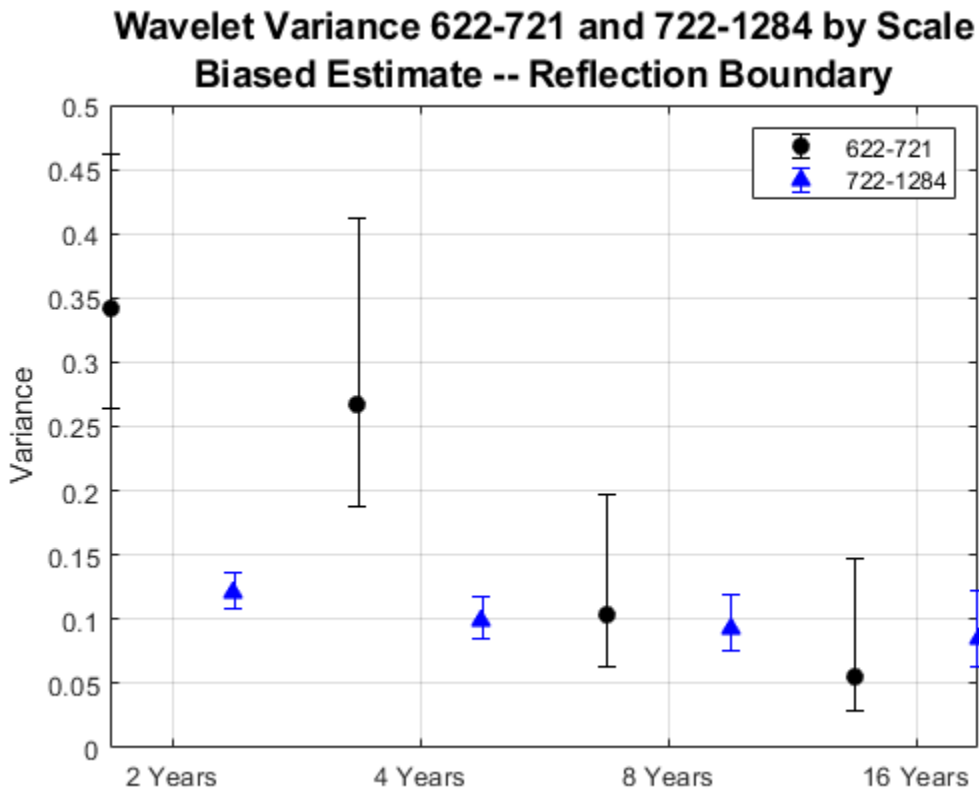
Plot the results.

```
Vpre = table2array(wvarpre);  
Vpost = table2array(wvarpost);  
Vpre = Vpre(1:end-1,2:end);  
Vpost = Vpost(1:end-1,2:end);
```

```
Vpre(:,1) = Vpre(:,2)-Vpre(:,1);  
Vpre(:,3) = Vpre(:,3)-Vpre(:,2);
```

```
Vpost(:,1) = Vpost(:,2)-Vpost(:,1);  
Vpost(:,3) = Vpost(:,3)-Vpost(:,2);
```

```
figure  
errorbar(1:4,Vpre(:,2),Vpre(:,1),Vpre(:,3),'ko','MarkerFaceColor',[0 0 0])  
hold on  
errorbar(1.5:4.5,Vpost(:,2),Vpost(:,1),Vpost(:,3),'b^','MarkerFaceColor',[0 0 1])  
set(gca,'xtick',1.25:4.25)  
set(gca,'xticklabel',{'2 Years','4 Years','8 Years','16 Years','32 Years'})  
grid on  
ylabel('Variance')  
title({'Wavelet Variance 622-721 and 722-1284 by Scale'; ...  
    'Biased Estimate -- Reflection Boundary'},'fontsize',14)  
legend('622-721','722-1284','Location','NorthEast')  
hold off
```



The conclusion is reinforced. There is a significant difference in the variance of the data over scales of 2 and 4 years, but not at longer scales. You can conclude that something has changed about the process variance.

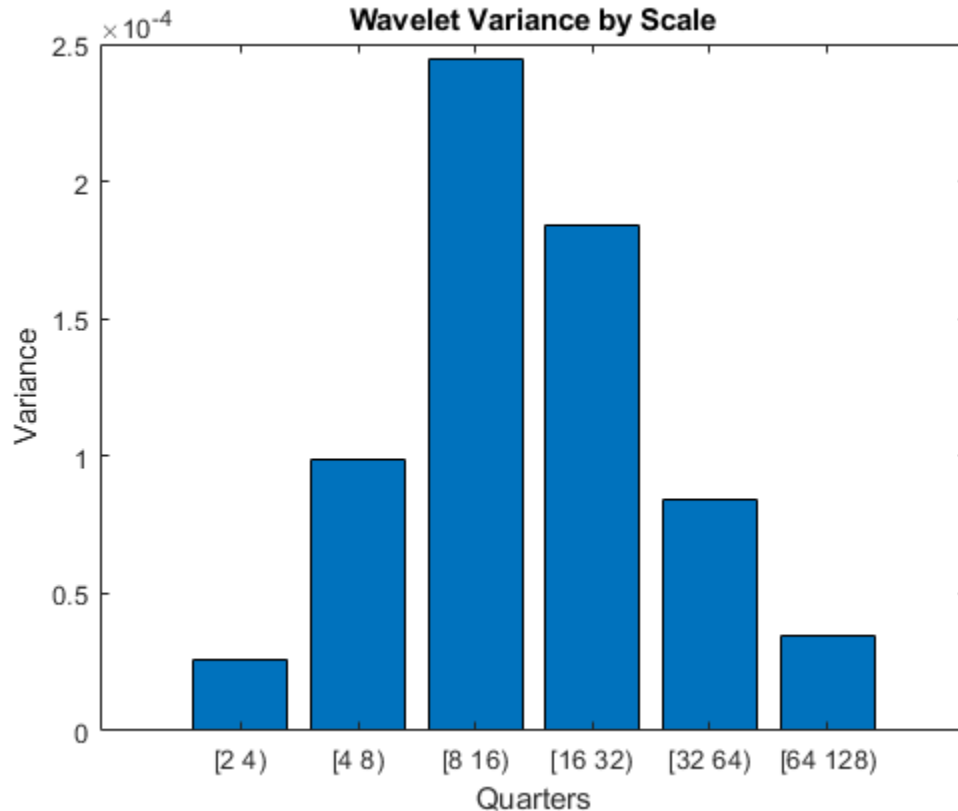
In financial time series, you can use wavelets to detect changes in volatility. To illustrate this, consider the quarterly chain-weighted U.S. real GDP data for 1974Q1 to 2012Q4. The data were transformed by first taking the natural logarithm and then calculating the year-over-year difference. Obtain the wavelet transform (MODWT) of the real GDP data down to level six with the 'db2' wavelet. Examine the variance of the data and compare that to the variances by scale obtained with the MODWT.

```
load GDPcomponents
realgdpwt = modwt(realgdp, 'db2', 6, 'reflection');
```

```
gdpmra = modwtmra(realgdpwt, 'db2', 'reflection');  
vardata = var(realgdp, 1);  
varwt = var(realgdpwt(:, 1: numel(realgdp)), 1, 2);
```

In `vardata` you have the variance for the aggregate GDP time series. In `varwt` you have the variance by scale for the MODWT. There are seven elements in `varwt` because you obtained the MODWT down to level six resulting in six wavelet coefficient variances and one scaling coefficient variance. Sum the variances by scale to see that the variance is preserved. Plot the wavelet variances by scale ignoring the scaling coefficient variance.

```
totalMODWTvar = sum(varwt);  
bar(varwt(1:end-1, :))  
AX = gca;  
AX.XTickLabels = {'[2 4]', '[4 8]', '[8 16]', '[16 32]', '[32 64]', '[64 128]'};  
xlabel('Quarters')  
ylabel('Variance')  
title('Wavelet Variance by Scale')
```

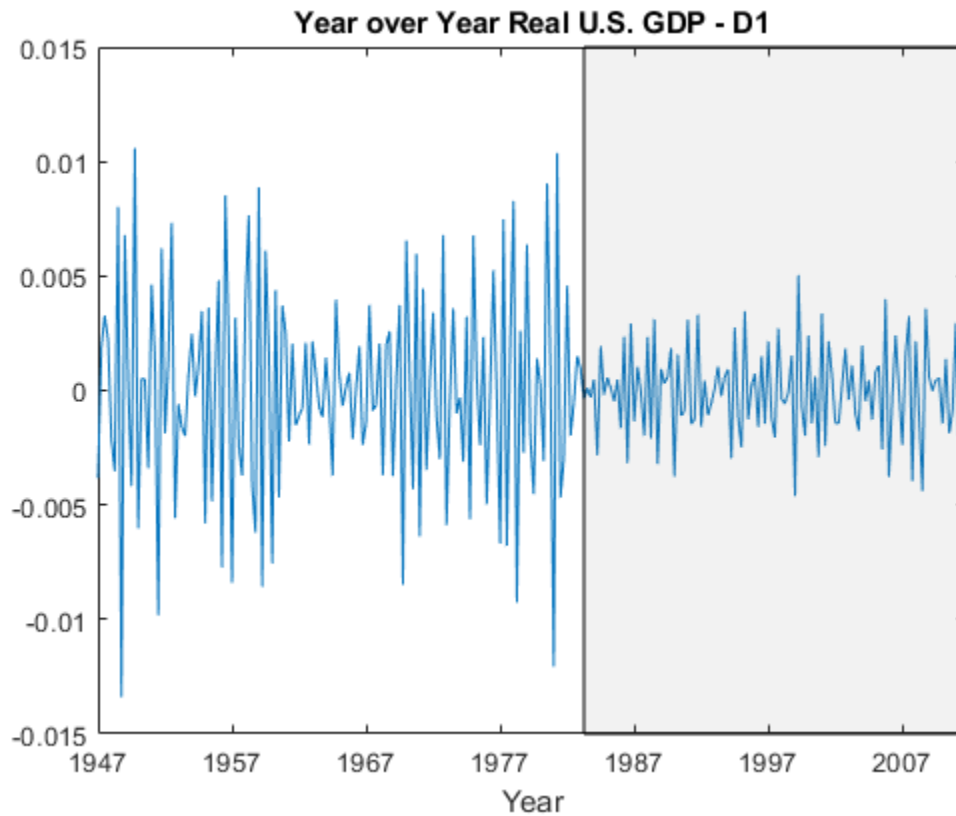


Because this data is quarterly, the first scale captures variations between two and four quarters, the second scale between four and eight, the third between 8 and 16, and so on.

From the MODWT and a simple bar plot, you see that cycles in the data between 8 and 32 quarters account for the largest variance in the GDP data. If you consider the wavelet variances at these scales, they account for 57% of the variability in the GDP data. This means that oscillations in the GDP over a period of 2 to 8 years account for most of the variability seen in the time series.

Plot the level-one details, D1. These details capture oscillations in the data between two and four quarters in duration.

```
helperFinancialDataExample1(gdpmra(1,:), years, ...
    'Year over Year Real U.S. GDP - D1')
```



Examining the level-one details, it appears there is a reduction of variance beginning in the 1980s.

Test the level-one wavelet coefficients for significant variance changepoints.

```
pts_Opt = wvarchg(realgdpwt(1,1: numel(realgdp)),2);  
years(pts_Opt)
```

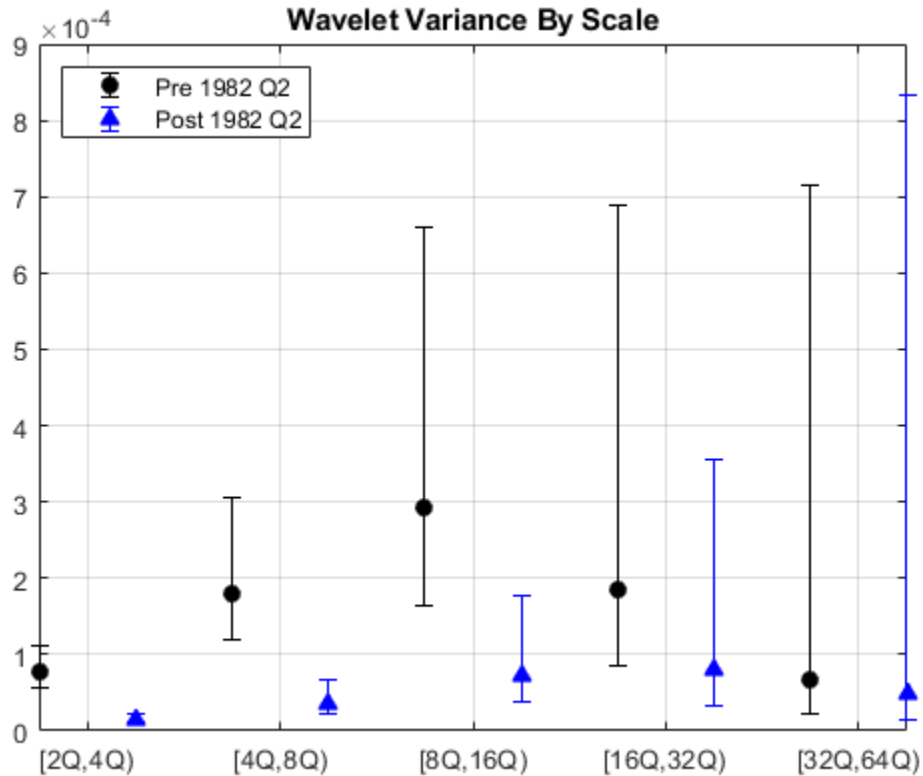
```
ans =
```

```
1982
```


There is a variance changepoint identified in 1982. This example does not correct for the delay introduced by the 'db2' wavelet at level one. However, that delay is only two samples so it does not appreciably affect the results.

To assess changes in the volatility of the GDP data pre and post 1982, split the original data into pre- and post-changepoint series. Obtain the wavelet transforms of the pre and post datasets. In this case, the series are relatively short so use the Haar wavelet to minimize the number of boundary coefficients. Compute unbiased estimates of the wavelet variance by scale and plot the result.

```
tspre = realgdp(1:pts_0pt);
tspost = realgdp(pts_0pt+1:end);
wtpre = modwt(tspre, 'haar', 5);
wtpost = modwt(tspost, 'haar', 5);
prevar = modwtvar(wtpre, 'haar', 'table');
postvar = modwtvar(wtpost, 'haar', 'table');
xlab = {'[2Q,4Q]', '[4Q,8Q]', '[8Q,16Q]', '[16Q,32Q]', '[32Q,64Q]'};
helperFinancialDataExampleVariancePlot(prevar, postvar, 'table', xlab)
title('Wavelet Variance By Scale')
legend('Pre 1982 Q2', 'Post 1982 Q2', 'Location', 'NorthWest')
```



From the preceding plot, it appears there are significant differences between the pre-1982Q2 and post-1982Q2 variances at scales between 2 and 16 quarters.

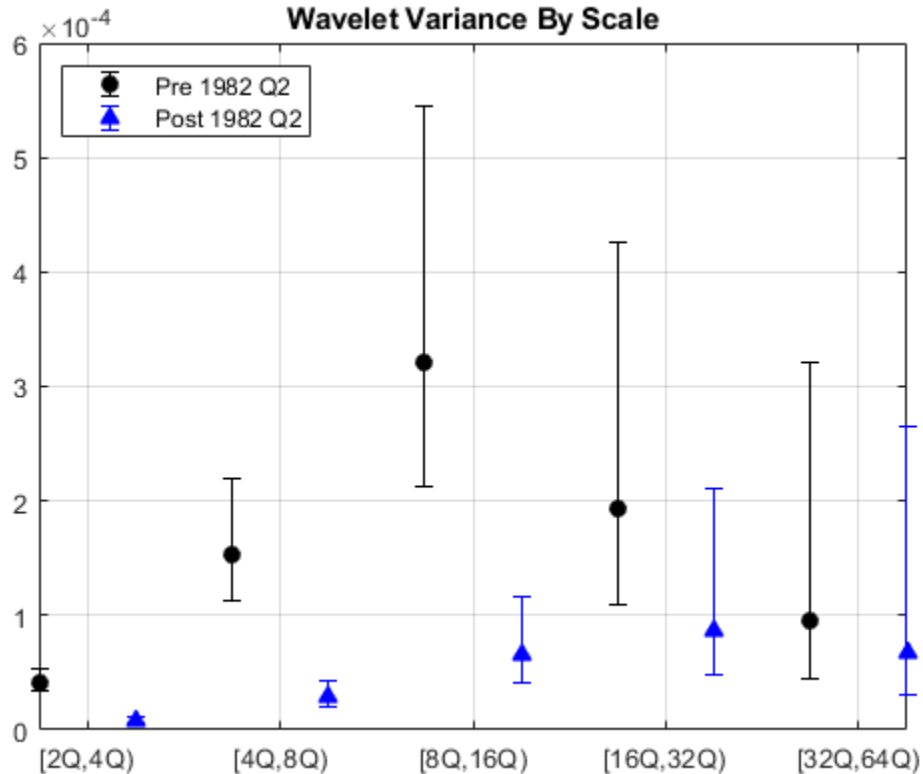
Because the time series are so short in this example, it can be useful to use biased estimates of the variance. Biased estimates do not remove boundary coefficients. Use a 'db2' wavelet filter with four coefficients.

```
wtpre = modwt(tspre,'db2',5,'reflection');
wtpost = modwt(tspost,'db2',5,'reflection');
prevar = modwtvar(wtpre,'db2',0.95,'EstimatorType','biased','table');
postvar = modwtvar(wtpost,'db2',0.95,'EstimatorType','biased','table');
xlab = {'[2Q,4Q)', '[4Q,8Q)', '[8Q,16Q)', '[16Q,32Q)', '[32Q,64Q)'};
figure
helperFinancialDataExampleVariancePlot(prevar,postvar,'table',xlab)
```

```

title('Wavelet Variance By Scale')
legend('Pre 1982 Q2', 'Post 1982 Q2', 'Location', 'NorthWest')

```



The results confirm our original finding that there is a reduction in volatility over scales from 2 to 16 quarters.

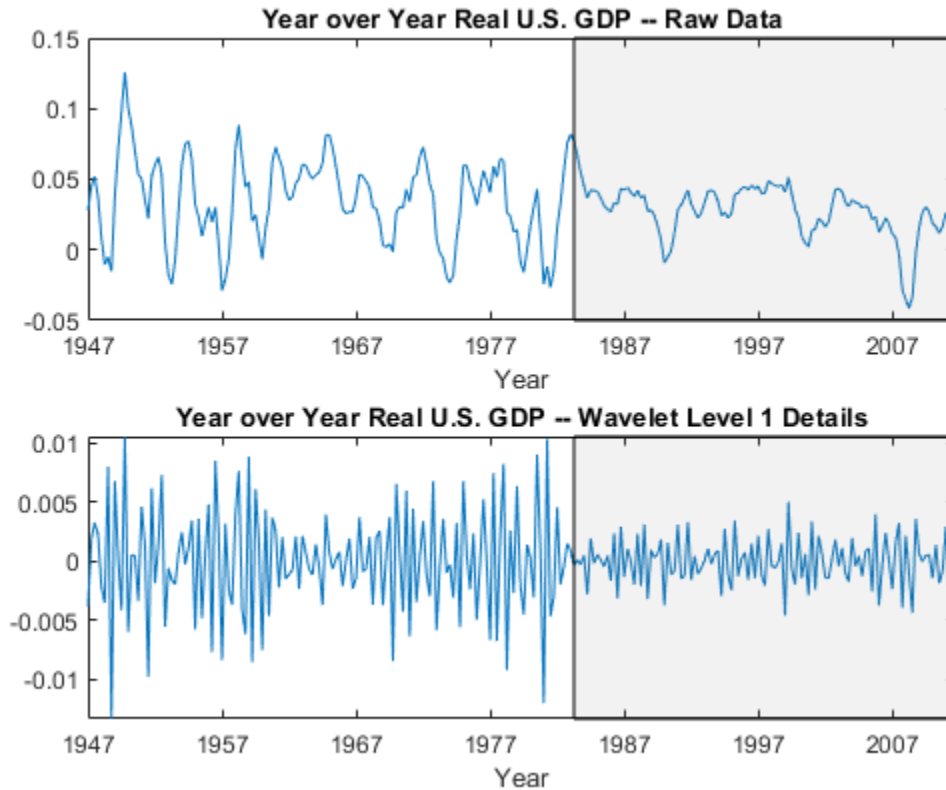
Using the wavelet transform allows you to focus on scales where the change in volatility is localized. To see this, examine a plot of the raw data along with the level-one wavelet details.

```

subplot(2,1,1)
helperFinancialDataExample1(realgdp,years,...
    'Year over Year Real U.S. GDP -- Raw Data')
subplot(2,1,2)

```

```
helperFinancialDataExample1(gdpmra(1,:),years,...  
    'Year over Year Real U.S. GDP -- Wavelet Level 1 Details')
```



The shaded region is referred to as the "Great Moderation" signifying a period of decreased macroeconomic volatility in the U.S. beginning in the mid 1980s.

Examining the aggregate data, it is not clear that there is in fact reduced volatility in this period. However, the wavelet level-one details uncover the change in volatility.

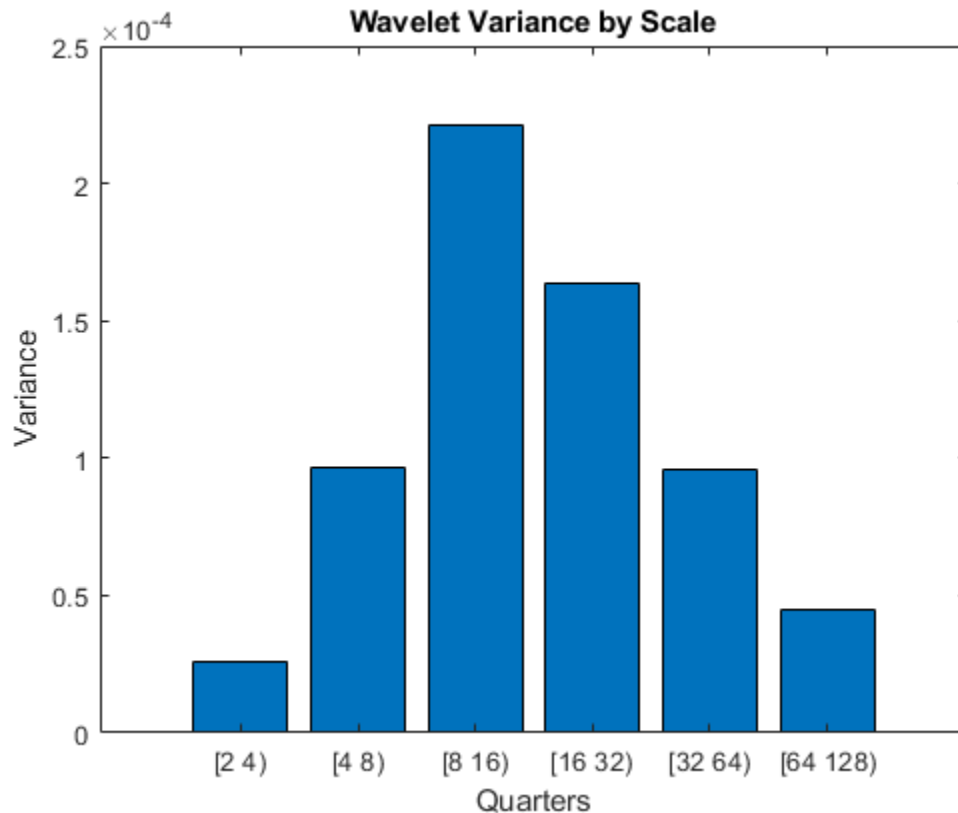
Scale-Localized Volatility and Correlation

There are a number of different variations of the wavelet transform. This example focuses on the maximal overlap discrete wavelet transform (MODWT). The MODWT is an undecimated wavelet transform over dyadic (powers of two) scales, which is frequently used with financial data. One nice feature of the MODWT for time series analysis is that it partitions the data variance by scale. To illustrate this, consider the quarterly chain-weighted U.S. real GDP data for 1974Q1 to 2012Q4. The data were transformed by first taking the natural logarithm and then calculating the year-over-year difference. Obtain the MODWT of the real GDP data down to level six with the 'db2' wavelet. Examine the variance of the data and compare that to the variances by scale obtained with the MODWT.

```
load GDPcomponents
realgdpwt = modwt(realgdp, 'db2', 6);
vardata = var(realgdp, 1);
varwt = var(realgdpwt, 1, 2);
```

In `vardata` you have the variance for the aggregate GDP time series. In `varwt` you have the variance by scale for the MODWT. There are seven elements in `varwt` because you obtained the MODWT down to level six resulting in six wavelet coefficient variances and one scaling coefficient variance. Sum the variances by scale to see that the variance is preserved. Plot the wavelet variances by scale ignoring the scaling coefficient variance.

```
totalMODWTvar = sum(varwt);
bar(varwt(1:end-1, :))
AX = gca;
AX.XTickLabels = {'[2 4]', '[4 8]', '[8 16]', '[16 32]', '[32 64]', '[64 128]'};
xlabel('Quarters')
ylabel('Variance')
title('Wavelet Variance by Scale')
```

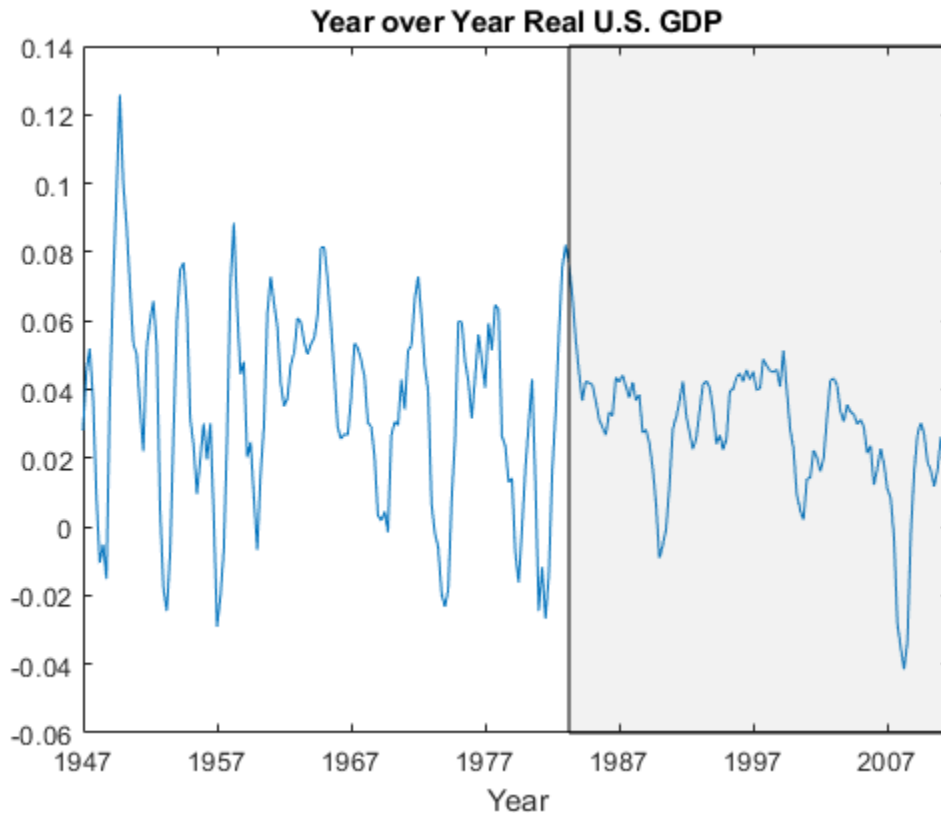


Because this data is quarterly, the first scale captures variations between two and four quarters, the second scale between four and eight, the third between 8 and 16, and so on.

From the MODWT and a simple bar plot, you see that cycles in the data between 8 and 32 quarters account for the largest variance in the GDP data. If you consider the wavelet variances at these scales, they account for 57% of the variability in the GDP data. This means that oscillations in the GDP over a period of 2 to 8 years account for most of the variability seen in the time series.

Wavelet analysis can often reveal changes in volatility not evident in aggregate data. Begin with a plot of the GDP data.

```
helperFinancialDataExample1(realgdp, years, 'Year over Year Real U.S. GDP')
```



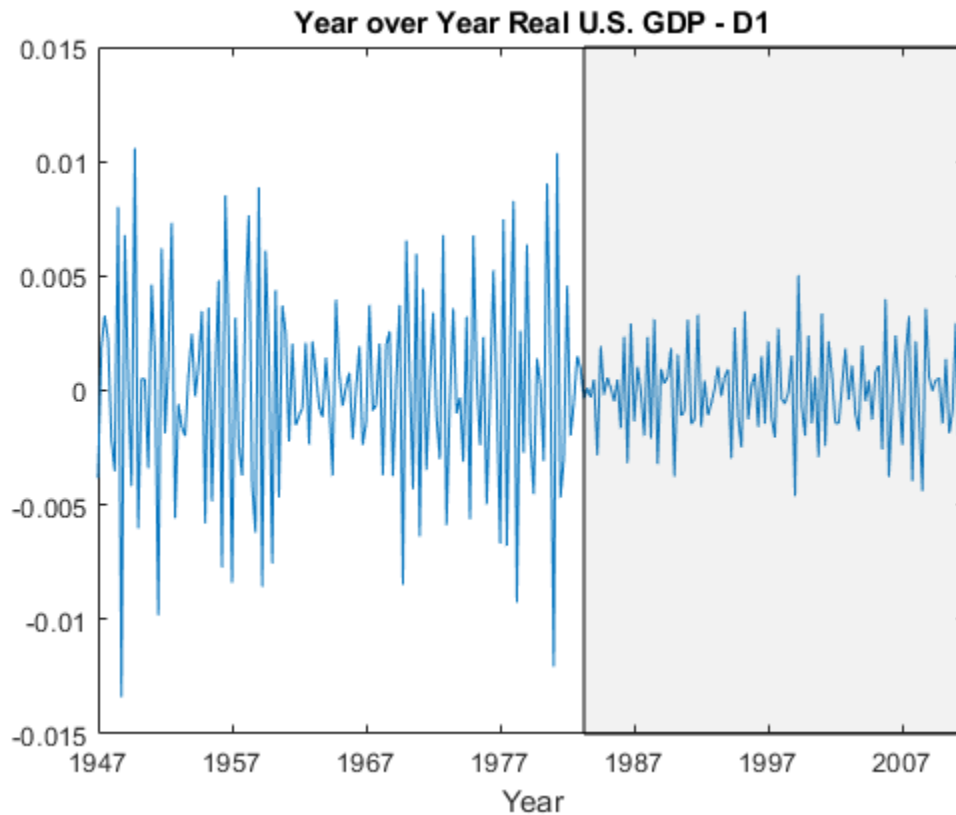
The shaded region is referred to as the "Great Moderation" signifying a period of decreased macroeconomic volatility in the U.S. beginning in the mid 1980s.

Examining the aggregate data, it is not clear that there is in fact reduced volatility in this period. Use wavelets to investigate this by first obtaining a multiresolution analysis of the real GDP data using the 'db2' wavelet down to level 6.

```
realgdpwt = modwt(realgdp, 'db2', 6, 'reflection');
gdpmra = modwtmra(realgdpwt, 'db2', 'reflection');
```

Plot the level-one details, D1. These details capture oscillations in the data between two and four quarters in duration.

```
helperFinancialDataExample1(gdpmra(1,:),years,...  
    'Year over Year Real U.S. GDP - D1')
```



Examining the level-one details, it appears there is a reduction of variance in the period of the Great Moderation.

Test the level-one wavelet coefficients for significant variance changepoints.

```
[pts_0pt,kopt,t_est] = wvarchg(realgdppwt(1,1:numel(realgdp)),2);  
years(pts_0pt)
```

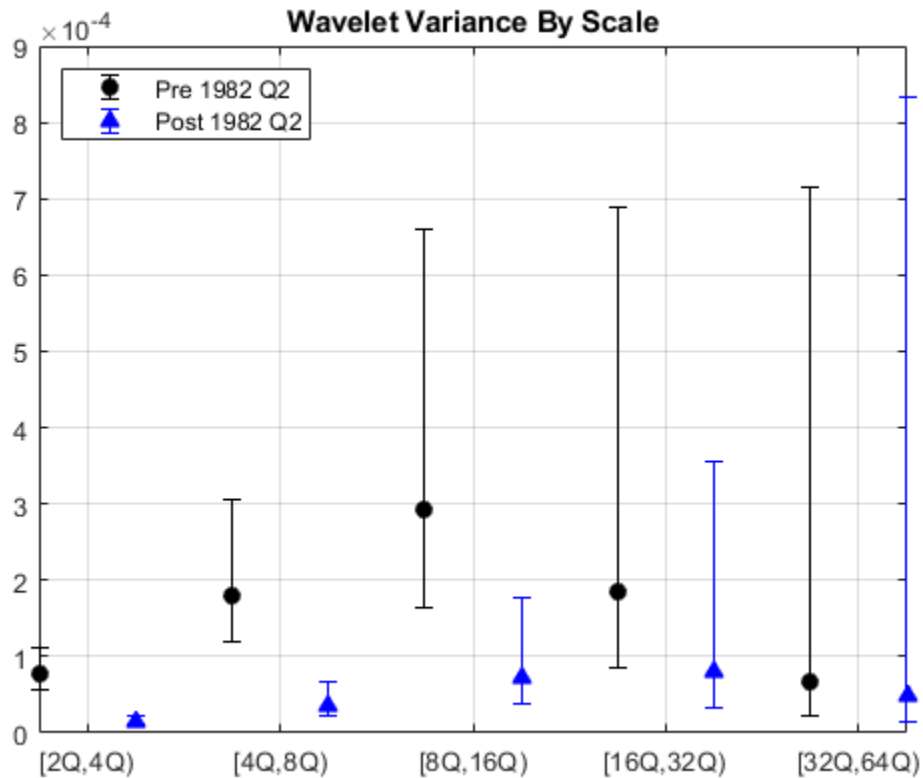
```
ans =
```


1982

There is a variance changepoint identified in 1982. This example does not correct for the delay introduced by the 'db2' wavelet at level one. However, that delay is only two samples so it does not appreciably affect the results.

To assess changes in the volatility of the GDP data pre and post 1982, split the original data into pre- and post-changepoint series. Obtain the wavelet transforms of the pre and post datasets. In this case, the series are relatively short so use the Haar wavelet to minimize the number of boundary coefficients. Compute unbiased estimates of the wavelet variance by scale and plot the result.

```
tspre = realgdp(1:pts_0pt);
tspost = realgdp(pts_0pt+1:end);
wtpre = modwt(tspre, 'haar', 5);
wtpost = modwt(tspost, 'haar', 5);
prevar = modwtvar(wtpre, 'haar', 'table');
postvar = modwtvar(wtpost, 'haar', 'table');
xlab = {'[20,40)', '[40,80)', '[80,160)', '[160,320)', '[320,640)'};
helperFinancialDataExampleVariancePlot(prevar, postvar, 'table', xlab)
title('Wavelet Variance By Scale');
legend('Pre 1982 Q2', 'Post 1982 Q2', 'Location', 'NorthWest');
```

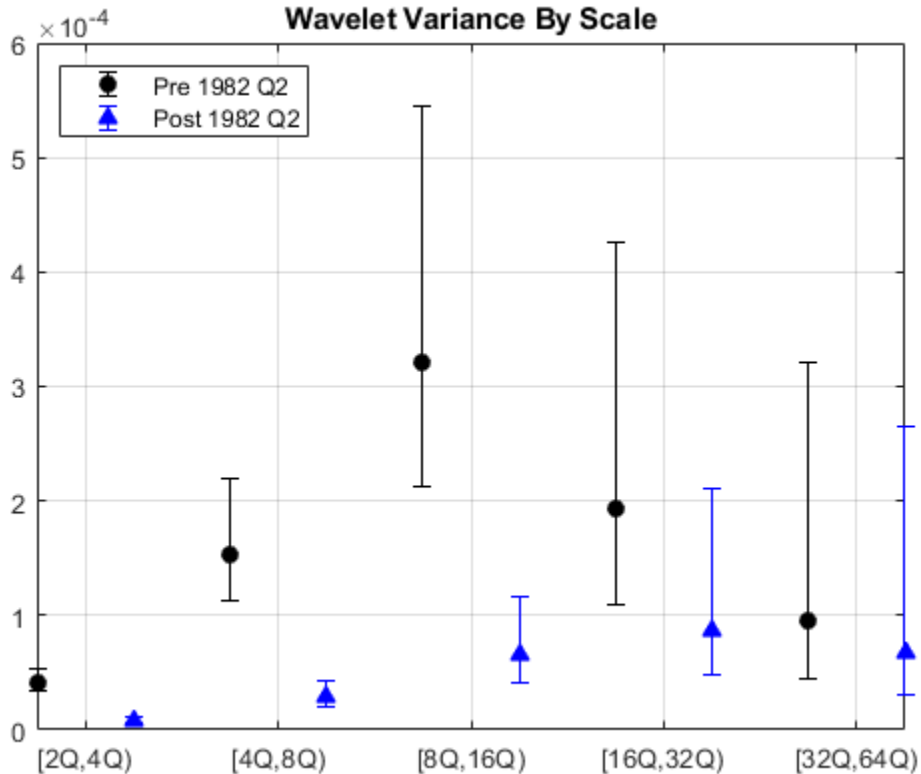


From the preceding plot, it appears there are significant differences between the pre-1982Q2 and post-1982Q2 variances at scales between 2 and 16 quarters.

Because the time series are so short in this example, it can be useful to use biased estimates of the variance. Biased estimates do not remove boundary coefficients. Use a 'db2' wavelet filter with four coefficients.

```
wtpre = modwt(tspre,'db2',5,'reflection');
wtpost = modwt(tspost,'db2',5,'reflection');
prevar = modwtvar(wtpre,'db2',0.95,'EstimatorType','biased','table');
postvar = modwtvar(wtpost,'db2',0.95,'EstimatorType','biased','table');
xlab = {'[2Q,4Q)', '[4Q,8Q)', '[8Q,16Q)', '[16Q,32Q)', '[32Q,64Q)'};
figure;
helperFinancialDataExampleVariancePlot(prevar,postvar,'table',xlab)
```

```
title('Wavelet Variance By Scale');
legend('Pre 1982 Q2','Post 1982 Q2','Location','NorthWest');
```



The results confirm our original finding that the Great Moderation is manifested in volatility reductions over scales from 2 to 16 quarters.

You can also use wavelets to analyze correlation between two datasets by scale. Examine the correlation between the aggregate data on government spending and private investment. The data cover the same period as the real GDP data and are transformed in the exact same way.

```
[rho,pval] = corrcoef(privateinvest,govtexp);
```

Government spending and personal investment demonstrate a weak, but statistically significant, negative correlation of -0.215. Repeat this analysis using the MODWT.

```
wtPI = modwt(privateinvest, 'db2', 5, 'reflection');
wtGE = modwt(govtexp, 'db2', 5, 'reflection');
wcorrtable = modwtcorr(wtPI, wtGE, 'db2', 0.95, 'reflection', 'table');
display(wcorrtable)
```

wcorrtable =

6x6 table

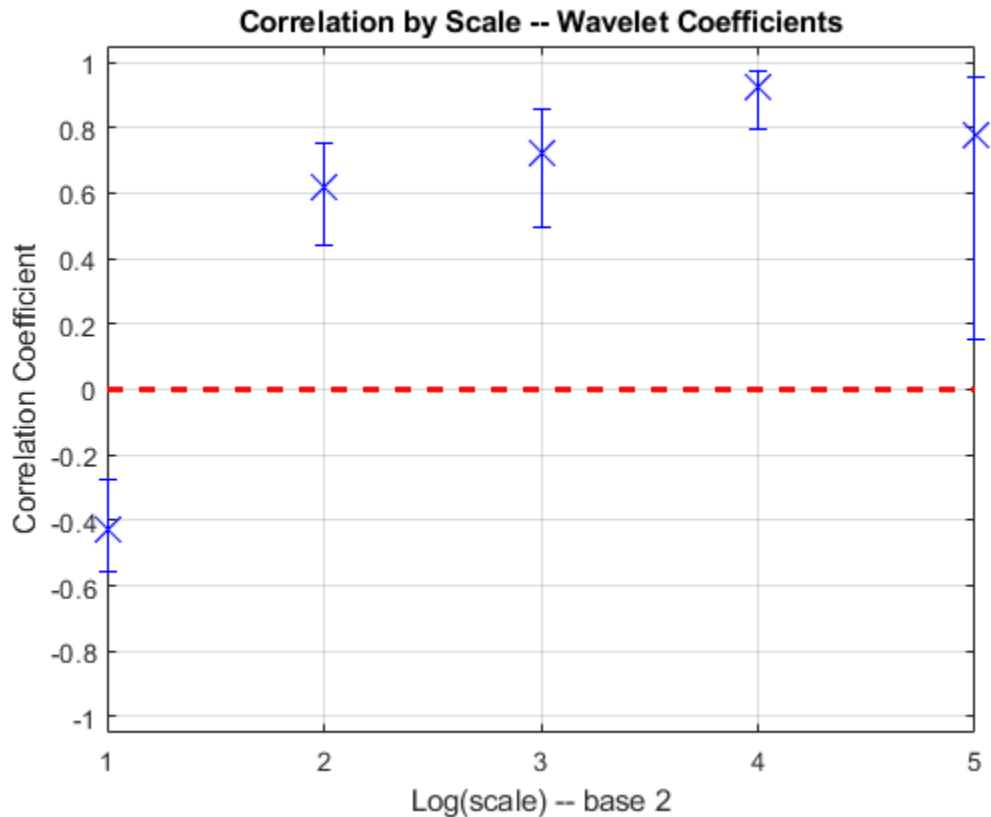
	NJ	Lower	Rho	Upper	Pvalue	AdjustedPvalue
D1	257	-0.29187	-0.12602	0.047192	0.1531	0.7502
D2	251	-0.54836	-0.35147	-0.11766	0.0040933	0.060171
D3	239	-0.62443	-0.35248	-0.0043207	0.047857	0.35175
D4	215	-0.70466	-0.32112	0.20764	0.22523	0.82773
D5	167	-0.63284	0.12965	0.76448	0.75962	1
S5	167	-0.63428	0.12728	0.76347	0.76392	1

The multiscale correlation available with the MODWT shows a significant negative correlation only at scale 2, which corresponds to cycles in the data between 4 and 8 quarters. Even this correlation is only marginally significant when adjusting for multiple comparisons.

The multiscale correlation analysis reveals that the slight negative correlation in the aggregate data is driven by the behavior of the data over scales of four to eight quarters. When you consider the data over different time periods (scales), there is no significant correlation.

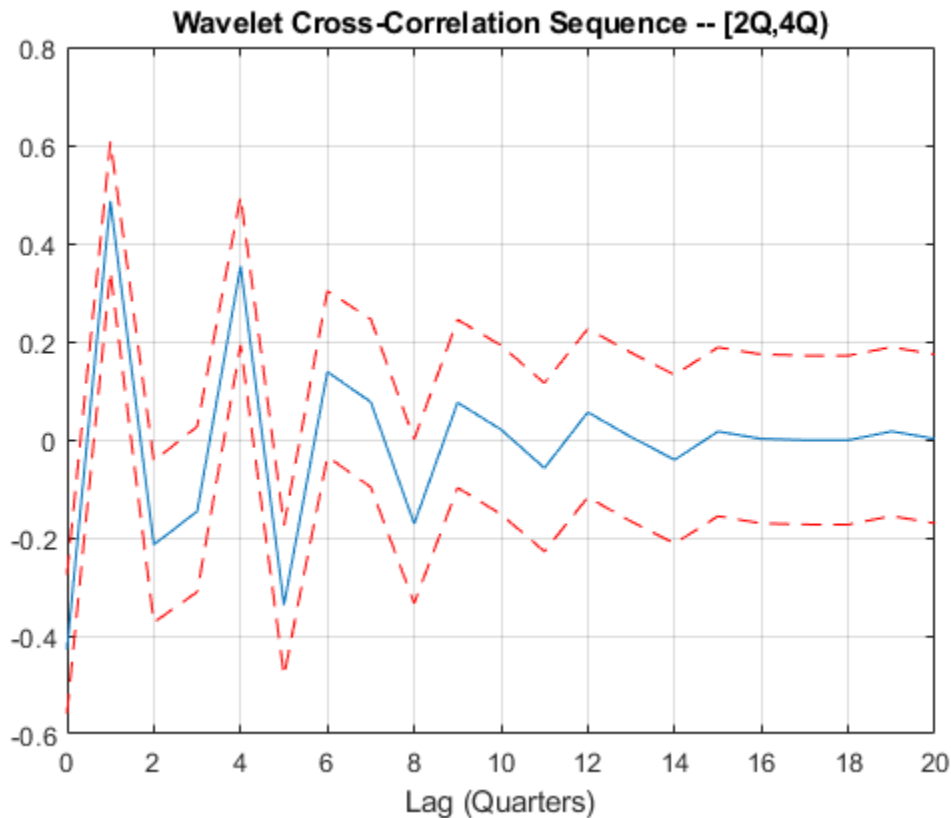
With financial data, there is often a leading or lagging relationship between variables. In those cases, it is useful to examine the cross-correlation sequence to determine if lagging one variable with respect to another maximizes their cross-correlation. To illustrate this, consider the correlation between two components of the GDP -- personal consumption expenditures and gross private domestic investment.

```
piwt = modwt(privateinvest, 'fk8', 5);
pcwt = modwt(pc, 'fk8', 5);
figure;
modwtcorr(piwt, pcwt, 'fk8')
```



Personal expenditure and personal investment are negatively correlated over a period of 2-4 quarters. At longer scales, there is a strong positive correlation between personal expenditure and personal investment. Examine the wavelet cross-correlation sequence at the scale representing 2-4 quarter cycles.

```
[xcseq,xcseqci,lags] = modwtcorr(piwt,pcwt,'fk8');
zerolag = floor(numel(xcseq{1})/2)+1;
plot(lags{1}(zerolag:zerolag+20),xcseq{1}(zerolag:zerolag+20));
hold on;
plot(lags{1}(zerolag:zerolag+20),xcseqci{1}(zerolag:zerolag+20,:), 'r--');
xlabel('Lag (Quarters)');
grid on;
title('Wavelet Cross-Correlation Sequence -- [2Q,4Q]');
```



The finest-scale wavelet cross-correlation sequence shows a peak positive correlation at a lag of one quarter. This indicates that personal investment lags personal expenditures by one quarter.

References:

Aguigar-Conraria, L. Martins. M.F., and Soares, M.J. "The Yield Curve and the Macro-Economy Across Time and Frequencies.", *Journal of Economic Dynamics and Control*, 36, 12, 1950-1970, 2012.

Crowley, P.M. "A Guide to Wavelets for Economists.", *Journal of Economic Surveys*, 21, 2, 207-267, 2007.

Gallegati, M and Semmler, W. (Eds.) "Wavelet Applications in Economics and Finance", Springer, 2014.

Percival, D.B. and Walden, A.T. "Wavelet Methods for Time Series Analysis", Cambridge University Press, 2000.

R Wave Detection in the ECG

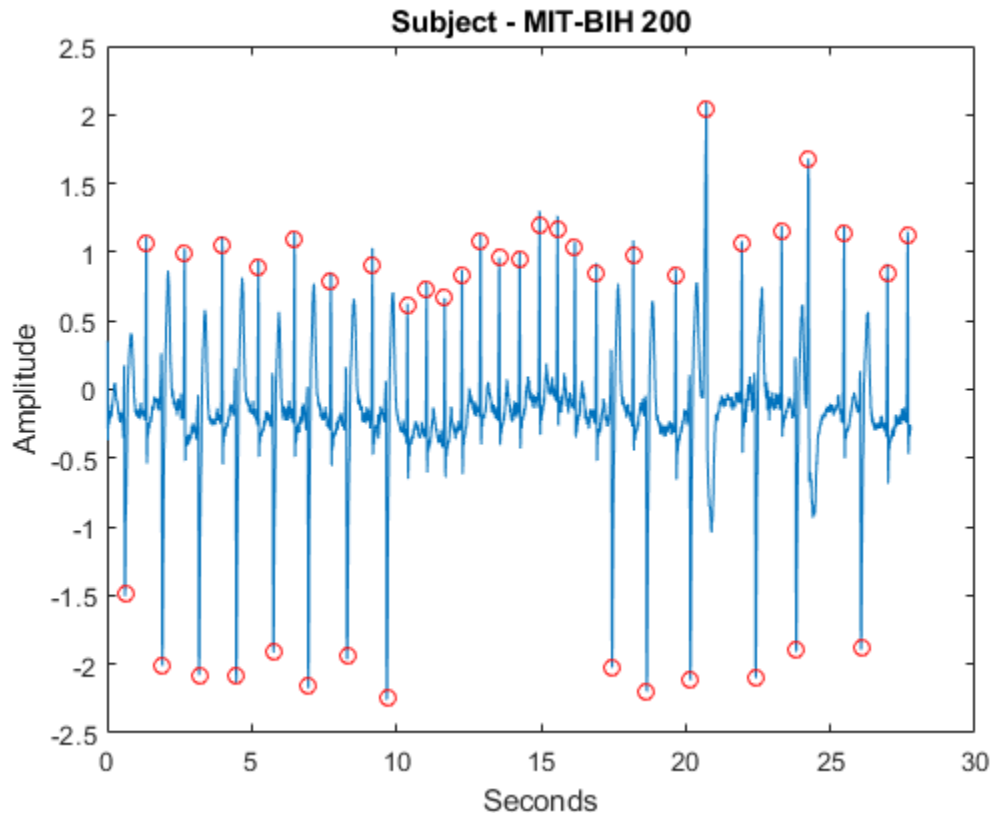
This example shows how to use wavelets to analyze electrocardiogram (ECG) signals. ECG signals are frequently nonstationary meaning that their frequency content changes over time. These changes are the events of interest.

Wavelets decompose signals into time-varying frequency (scale) components. Because signal features are often localized in time and frequency, analysis and estimation are easier when working with sparser (reduced) representations.

The QRS complex consists of three deflections in the ECG waveform. The QRS complex reflects the depolarization of the right and left ventricles and is the most prominent feature of the human ECG.

Load and plot an ECG waveform where the R peaks of the QRS complex have been annotated by two or more cardiologists. The ECG data and annotations are taken from the MIT-BIH Arrhythmia Database. The data are sampled at 360 Hz.

```
load mit200
figure
plot(tm,ecgsig)
hold on
plot(tm(ann),ecgsig(ann),'ro')
xlabel('Seconds')
ylabel('Amplitude')
title('Subject - MIT-BIH 200')
```

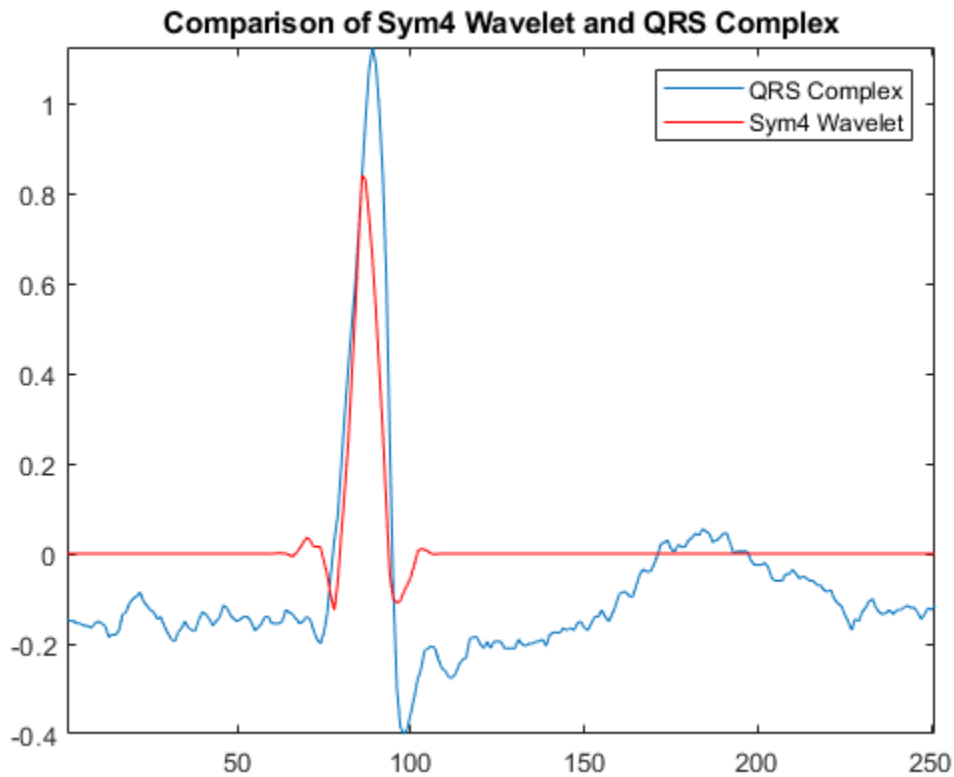
You can use wavelets to build an automatic QRS detector for use in applications like R-R interval estimation.

There are two keys for using wavelets as general feature detectors:

- The wavelet transform separates signal components into different frequency bands enabling a sparser representation of the signal.
- You can often find a wavelet which resembles the feature you are trying to detect.

The 'sym4' wavelet resembles the QRS complex, which makes it a good choice for QRS detection. To illustrate this more clearly, extract a QRS complex and plot the result with a dilated and translated 'sym4' wavelet for comparison.

```
qrsEx = ecgsig(4560:4810);  
[mpdict,~,~,longs] = wmpdictionary(numel(qrsEx),'lstcpt',{ 'sym4',3});  
figure  
plot(qrsEx)  
hold on  
plot(2*circshift(mpdict(:,11),[-2 0]),'r')  
axis tight  
legend('QRS Complex','Sym4 Wavelet')  
title('Comparison of Sym4 Wavelet and QRS Complex')
```



Use the maximal overlap discrete wavelet transform (MODWT) to enhance the R peaks in the ECG waveform. The MODWT is an undecimated wavelet transform, which handles arbitrary sample sizes.

First, decompose the ECG waveform down to level 5 using the default 'sym4' wavelet. Then, reconstruct a frequency-localized version of the ECG waveform using only the wavelet coefficients at scales 4 and 5. The scales correspond to the following approximate frequency bands.

- Scale 4 -- [11.25, 22.5) Hz
- Scale 5 -- [5.625, 11.25) Hz.

This covers the passband shown to maximize QRS energy.

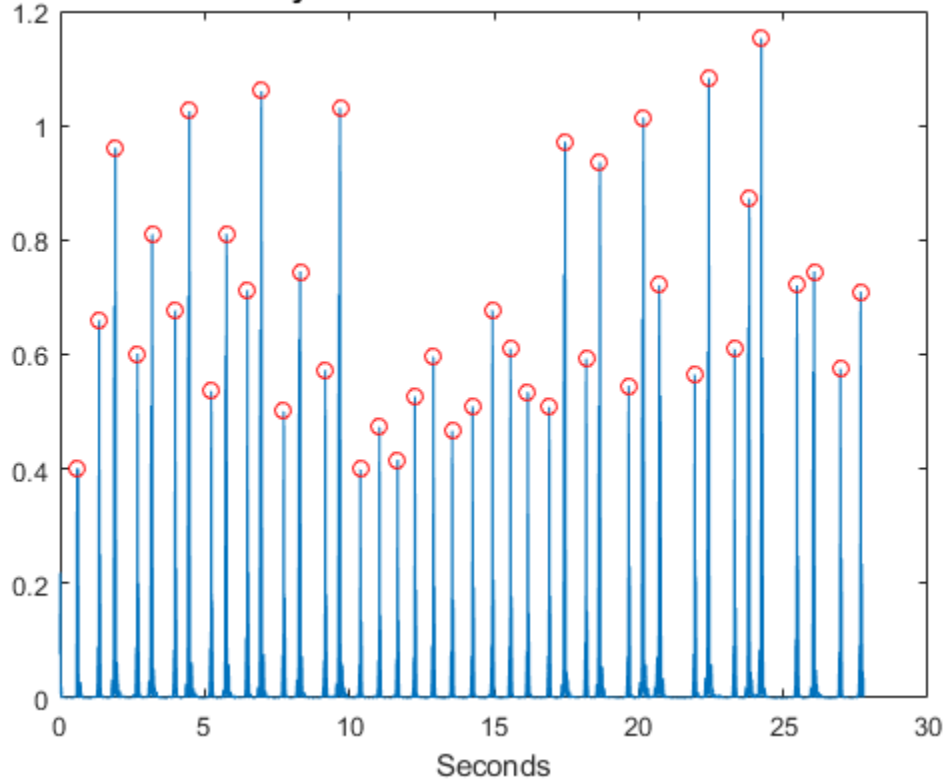
```
wt = modwt(ecgsig,5);  
wtrec = zeros(size(wt));  
wtrec(4:5,:) = wt(4:5,:);  
y = imodwt(wtrec,'sym4');
```

Use the squared absolute values of the signal approximation built from the wavelet coefficients and employ a peak finding algorithm to identify the R peaks.

If you have the Signal Processing Toolbox™, you can use `findpeaks` to locate the peaks. Plot the R-peak waveform obtained with the wavelet transform annotated with the automatically-detected peak locations.

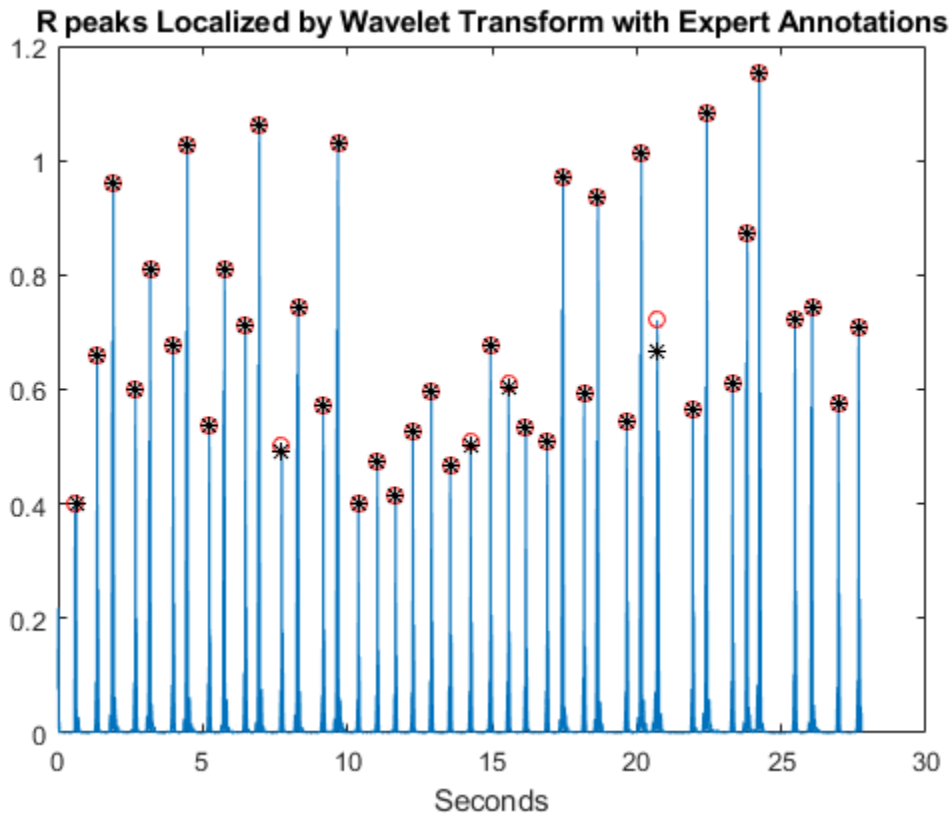
```
y = abs(y).^2;  
[qrspeaks,locs] = findpeaks(y,tm,'MinPeakHeight',0.35,...  
    'MinPeakDistance',0.150);  
figure  
plot(tm,y)  
hold on  
plot(locs,qrspeaks,'ro')  
xlabel('Seconds')  
title('R Peaks Localized by Wavelet Transform with Automatic Annotations')
```

R Peaks Localized by Wavelet Transform with Automatic Annotations



Add the expert annotations to the R-peak waveform. Automatic peak detection times are considered accurate if within 150 msec of the true peak (± 75 msec).

```
plot(tm(ann),y(ann),'k*')
title('R peaks Localized by Wavelet Transform with Expert Annotations')
```



At the command line, you can compare the values of `tm(ann)` and `locs`, which are the expert times and automatic peak detection times respectively. Enhancing the R peaks with the wavelet transform results in a hit rate of 100% and no false positives. The calculated heart rate using the wavelet transform is 88.60 beats/minute compared to 88.72 beats/minute for the annotated waveform.

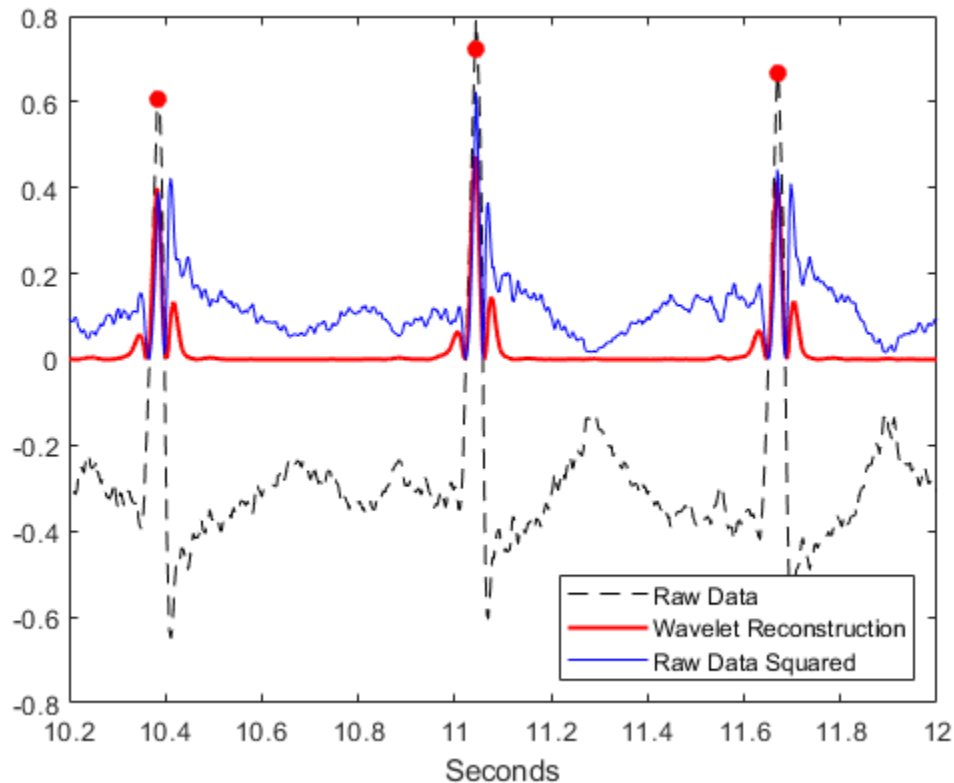
If you try to work on the square magnitudes of the original data, you find the capability of the wavelet transform to isolate the R peaks makes the detection problem much easier. Working on the raw data can cause misidentifications such as when the squared S-wave peak exceeds the R-wave peak around 10.4 seconds.

```
figure
plot(tm,ecgsig,'k--')
```

```

hold on
plot(tm,y,'r','linewidth',1.5)
plot(tm,abs(ecgsig).^2,'b')
plot(tm(ann),ecgsig(ann),'ro','markerfacecolor',[1 0 0])
set(gca,'xlim',[10.2 12])
legend('Raw Data','Wavelet Reconstruction','Raw Data Squared', ...
'Location','SouthEast');
xlabel('Seconds')

```



Using `findpeaks` on the squared magnitudes of the raw data results in twelve false positives.

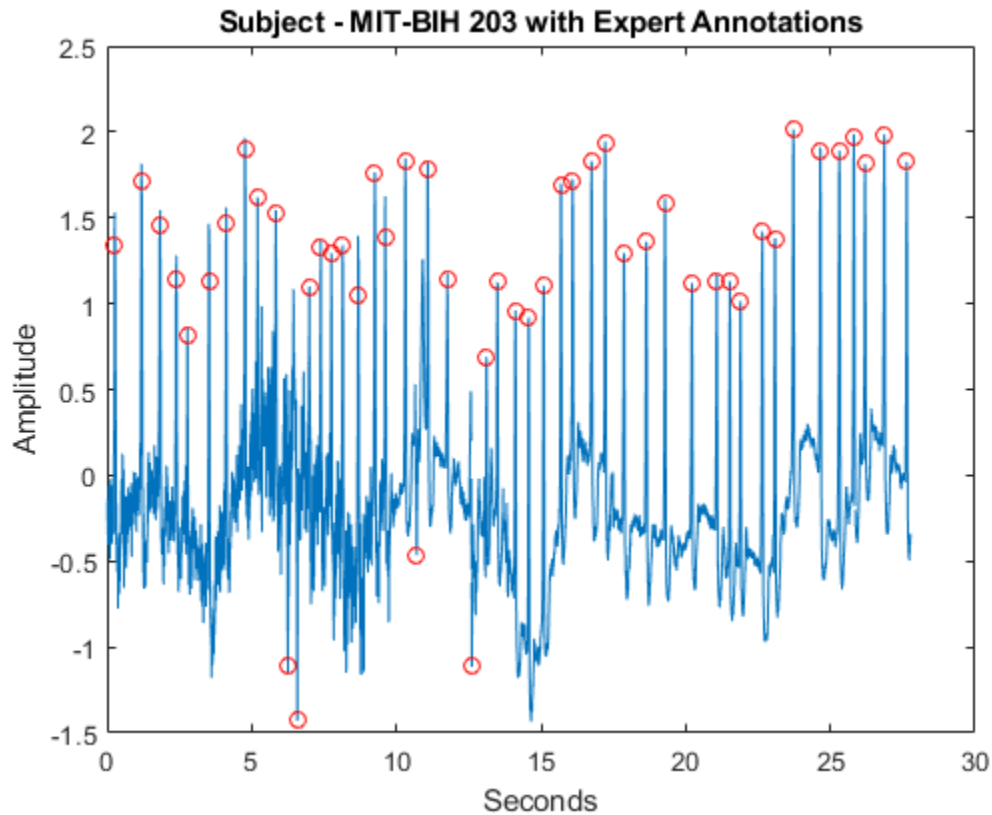
```

[qrspeaks,locs] = findpeaks(ecgsig.^2,tm,'MinPeakHeight',0.35,...
'MinPeakDistance',0.150);

```

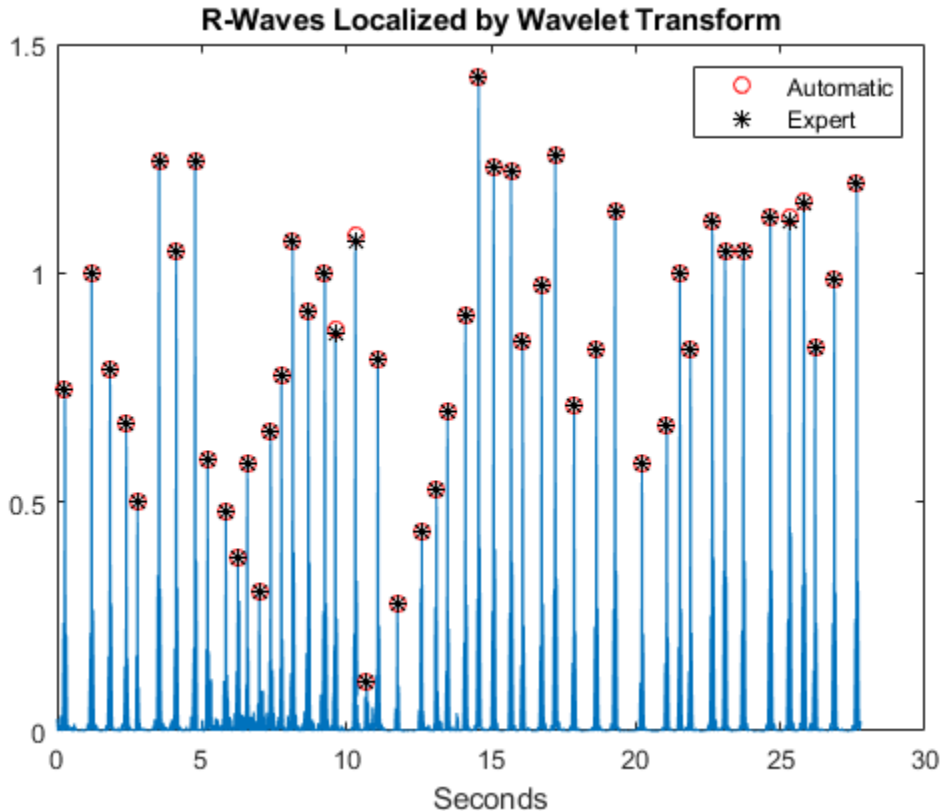
In addition to switches in polarity of the R peaks, the ECG is often corrupted by noise.

```
load mit203
figure
plot(tm,ecgsig)
hold on
plot(tm(ann),ecgsig(ann),'ro')
xlabel('Seconds')
ylabel('Amplitude')
title('Subject - MIT-BIH 203 with Expert Annotations')
```



Use the MODWT to isolate the R peaks. Use `findpeaks` to determine the peak locations. Plot the R-peak waveform along with the expert and automatic annotations.

```
wt = modwt(ecgsig,5);
wtrec = zeros(size(wt));
wtrec(4:5,:) = wt(4:5,:);
y = imodwt(wtrec,'sym4');
y = abs(y).^2;
[qrspeaks,locs] = findpeaks(y,tm,'MinPeakHeight',0.1,...
    'MinPeakDistance',0.150);
figure
plot(tm,y)
title('R-Waves Localized by Wavelet Transform')
hold on
hwav = plot(locs,qrspeaks,'ro');
hexp = plot(tm(ann),y(ann),'k*');
xlabel('Seconds')
legend([hwav hexp],'Automatic','Expert','Location','NorthEast');
```

The hit rate is again 100% with zero false alarms.

The previous examples used a very simple wavelet QRS detector based on a signal approximation constructed from `modwt`. The goal was to demonstrate the ability of the wavelet transform to isolate signal components, not to build the most robust wavelet-transform-based QRS detector. It is possible, for example, to exploit the fact that the wavelet transform provides a multiscale analysis of the signal to enhance peak detection.

References

Goldberger A. L., L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. Ch. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C-K Peng, H. E. Stanley. "PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals." *Circulation*

101. Vol.23, e215-e220, 2000. <http://circ.ahajournals.org/cgi/content/full/101/23/e215>

Moody, G. B. "Evaluating ECG Analyzers". <http://www.physionet.org/physiotools/wfdb/doc/wag-src/eval0.tex>

Moody G. B., R. G. Mark. "The impact of the MIT-BIH Arrhythmia Database." IEEE Eng in Med and Biol. Vol. 20, Number 3, 2001), pp. 45-50 .

Wavelet Cross-Correlation for Lead-Lag Analysis

This example shows how to use wavelet cross-correlation to measure similarity between two signals at different scales.

Wavelet cross-correlation is simply a scale-localized version of the usual cross-correlation between two signals. In cross-correlation, you determine the similarity between two sequences by shifting one relative to the other, multiplying the shifted sequences element by element and summing the result. For deterministic sequences, you can write this as an ordinary inner product: $\langle x_n, y_{n-k} \rangle_n = \sum_n x_n \bar{y}_{n-k}$ where x_n and y_n are sequences

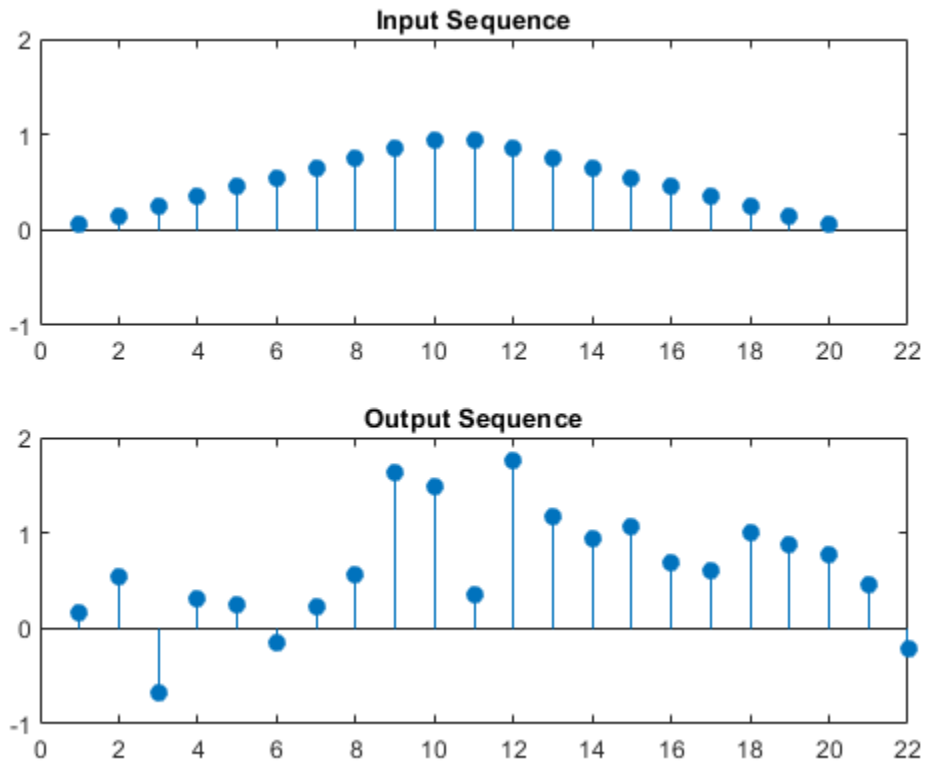
(signals) and the bar denotes complex conjugation. The variable, k , is the lag variable and represents the shift applied to y_n . If both x_n and y_n are real, the complex conjugate is not necessary. Assume that y_n is the same sequence as x_n but delayed by $L > 0$ samples, where L is an integer. For concreteness, assume $y_n = x_{n-10}$. If you express y_n in terms of x_n above, you obtain $\langle x_n, x_{n-10-k} \rangle_n = \sum_n x_n \bar{x}_{n-10-k}$. By the Cauchy-Schwartz inequality, the above is maximized when $k = -10$. This means if you left shift (advance) y_n by 10 samples, you get the maximum cross-correlation sequence value. If x_n is a L -delayed version of y_n , $x_n = y_{n-L}$, then the cross-correlation sequence is maximized at $k = L$.

If you have the Signal Processing Toolbox™, you can show this by using `xcorr`. Create a triangular signal consisting of 20 samples. Create a noisy shifted version of this signal. The shift in the peak of the triangle is 3 samples. Plot the x and y sequences.

```
x = triang(20);
rng default;
y = [zeros(3,1);x]+0.3*randn(length(x)+3,1);

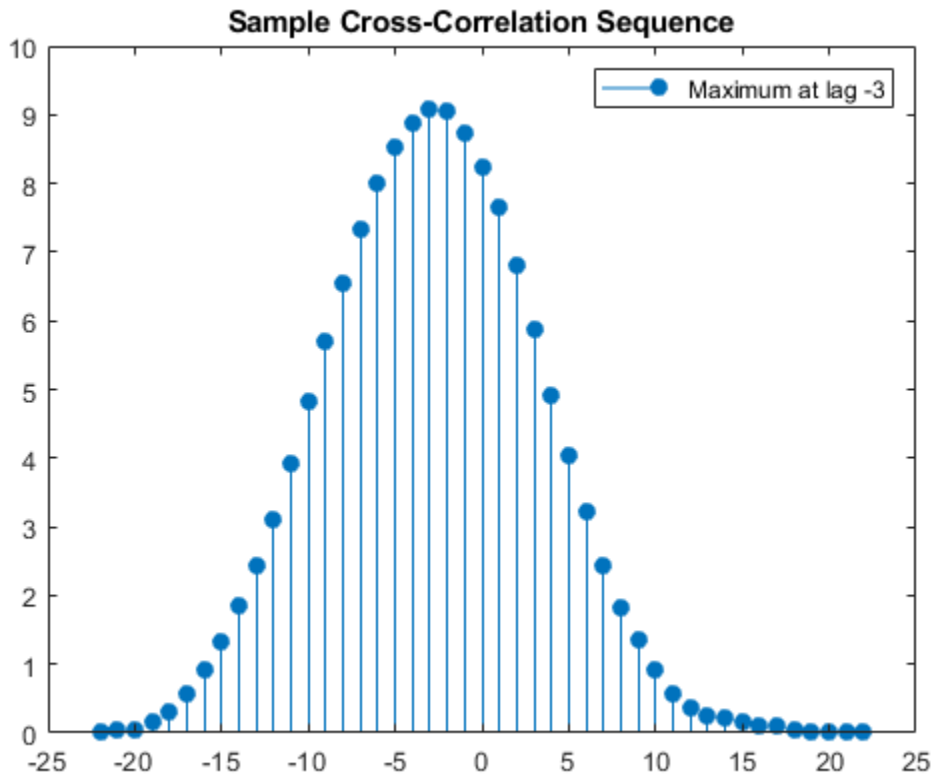
subplot(2,1,1)
stem(x,'filled')
axis([0 22 -1 2])
title('Input Sequence')

subplot(2,1,2)
stem(y,'filled')
axis([0 22 -1 2])
title('Output Sequence')
```



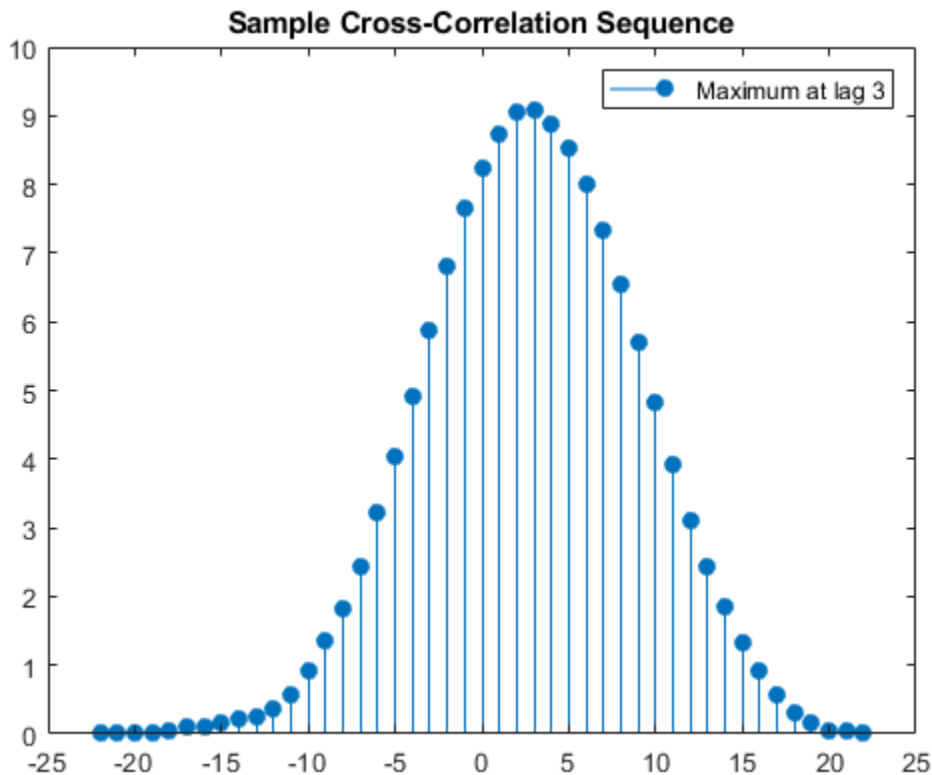
Use `xcorr` to obtain the cross-correlation sequence and determine the lag where the maximum is obtained.

```
[xc,lags] = xcorr(x,y);
[~,I] = max(abs(xc));
figure
stem(lags,xc,'filled')
legend(sprintf('Maximum at lag %d',lags(I)))
title('Sample Cross-Correlation Sequence')
```



The maximum is found at a lag of -3. The signal y is the second input to `xcorr` and it is a delayed version of x . You have to shift y 3 samples to the left (a negative shift) to maximize the cross correlation. If you reverse the roles of x and y as inputs to `xcorr`, the maximum lag now occurs at a positive lag.

```
[xc,lags] = xcorr(y,x);
[~,I] = max(abs(xc));
figure
stem(lags,xc,'filled')
legend(sprintf('Maximum at lag %d',lags(I)))
title('Sample Cross-Correlation Sequence')
```



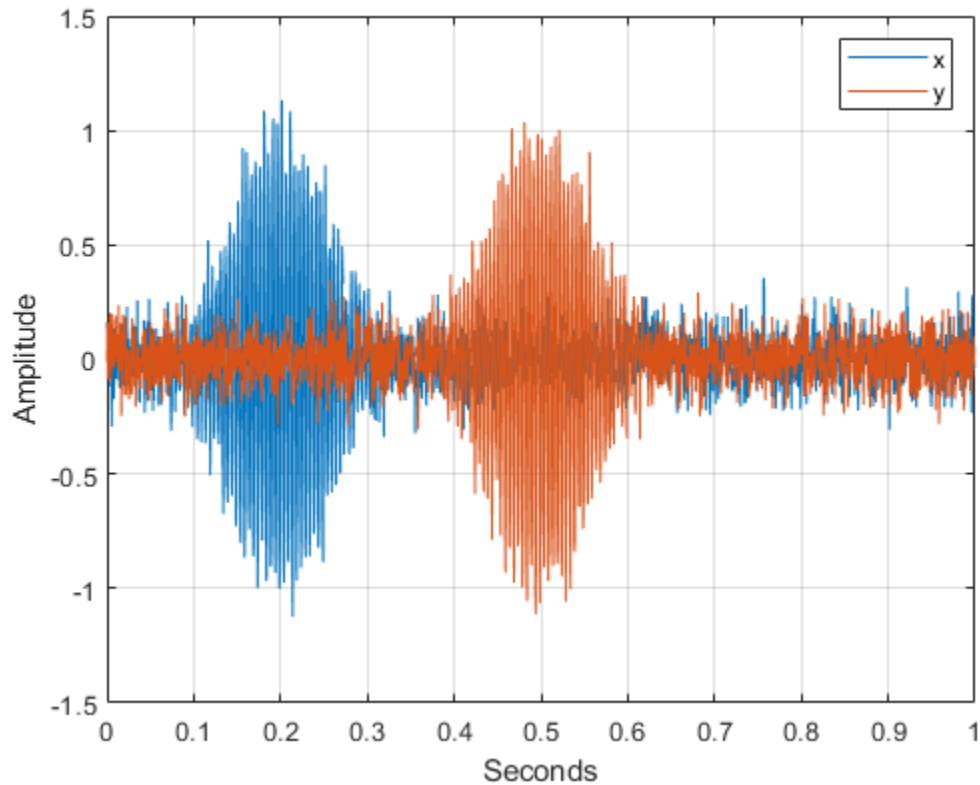
x is an advanced version of y and you delay x by three samples to maximize the cross correlation.

`modwtxcorr` is the scale-based version of `xcorr`. To use `modwtxcorr`, you first obtain the nondecimated wavelet transforms.

Apply wavelet cross-correlation to two signals that are shifted versions of each other. Construct two exponentially-damped 200-Hz sine waves with additive noise. The x signal has its time center at $t = 0.2$ seconds while y is centered at $t = 0.5$ seconds.

```
t = 0:1/2000:1-1/2000;
x = sin(2*pi*200*t).*exp(-50*pi*(t-0.2).^2)+0.1*randn(size(t));
y = sin(2*pi*200*t).*exp(-50*pi*(t-0.5).^2)+0.1*randn(size(t));
figure
```

```
plot(t,x)
hold on
plot(t,y)
xlabel('Seconds')
ylabel('Amplitude')
grid on
legend('x','y')
```



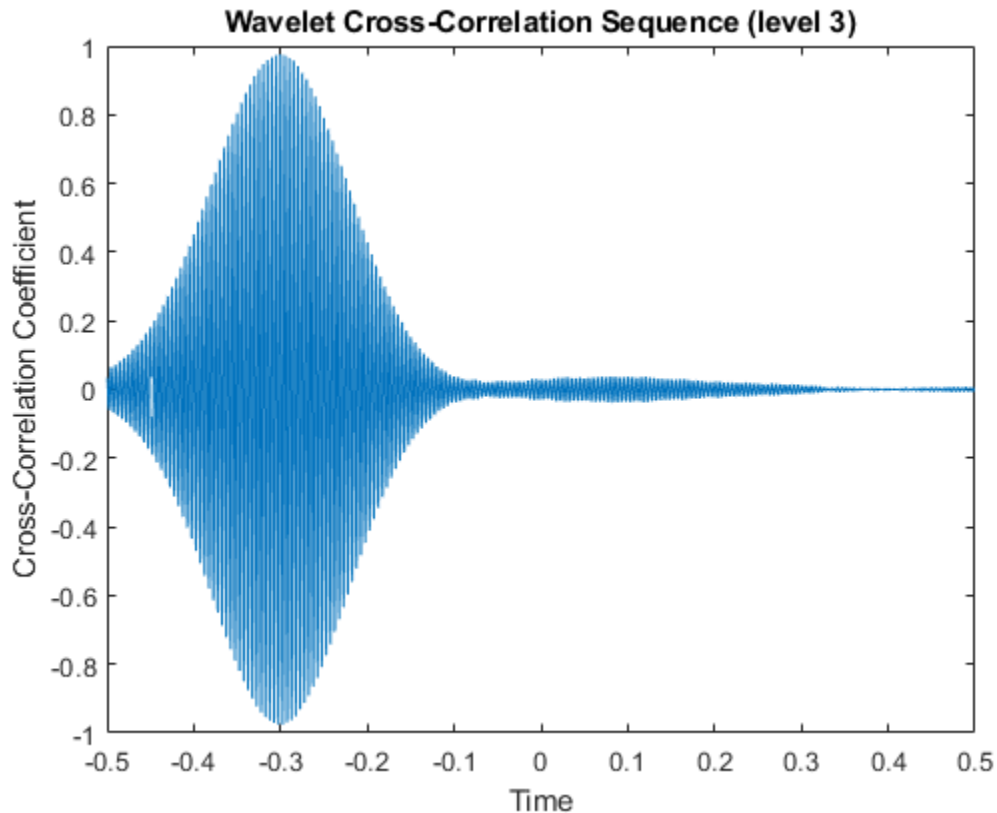
You see that x and y are very similar except that y is delayed by 0.3 seconds. Obtain the nondecimated wavelet transform of x and y down to level 5 using the Fejer-Korovkin (14) wavelet. The wavelet coefficients at level 3 with a sampling frequency of 2 kHz are an approximate $[2000/2^4, 2000/2^3)$ bandpass filtering of the inputs. The frequency

localization of the Fejer-Korovkin filters ensures that this bandpass approximation is quite good.

```
wx = modwt(x, 'fk14',5);  
wy = modwt(y, 'fk14',5);
```

Obtain the wavelet cross-correlation sequences for the wavelet transforms of x and y . Plot the level 3 wavelet cross-correlation sequence for 2000 lags centered at zero lag. Multiply the lags by the sampling period to obtain a meaningful time axis.

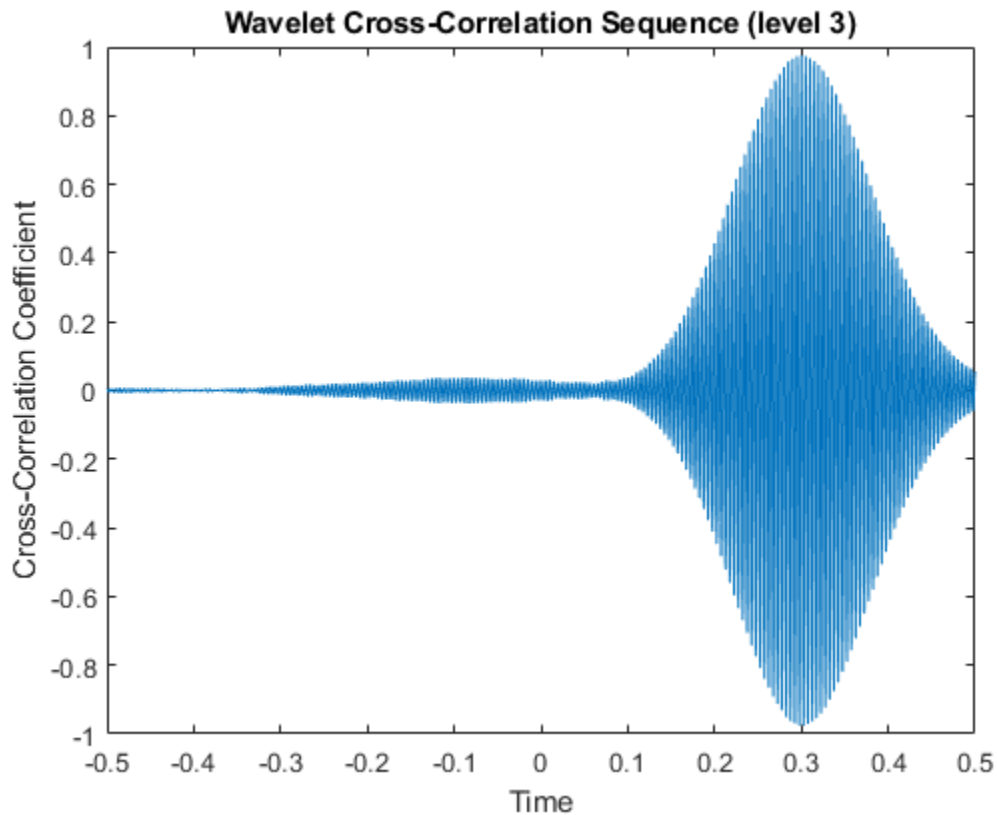
```
[xc,~,lags] = modwtcorr(wx,wy, 'fk14');  
lev = 3;  
zerolag = floor(numel(xc{lev})/2+1);  
tlag = lags{lev}(zerolag-999:zerolag+1000).*(1/2000);  
figure  
plot(tlag,xc{lev}(zerolag-999:zerolag+1000))  
title('Wavelet Cross-Correlation Sequence (level 3)')  
xlabel('Time')  
ylabel('Cross-Correlation Coefficient')
```

The cross-correlation sequence peaks at a delay of -0.3 seconds. The wavelet transform of y is the second input to `modwtcorr`. Because the second input of `modwtcorr` is shifted relative to the first, the peak correlation occurs at a negative delay. You have to left shift (advance) the cross-correlation sequence to align the time series. If you reverse the roles of the inputs to `modwtcorr`, you obtain the peak correlation at a positive lag.

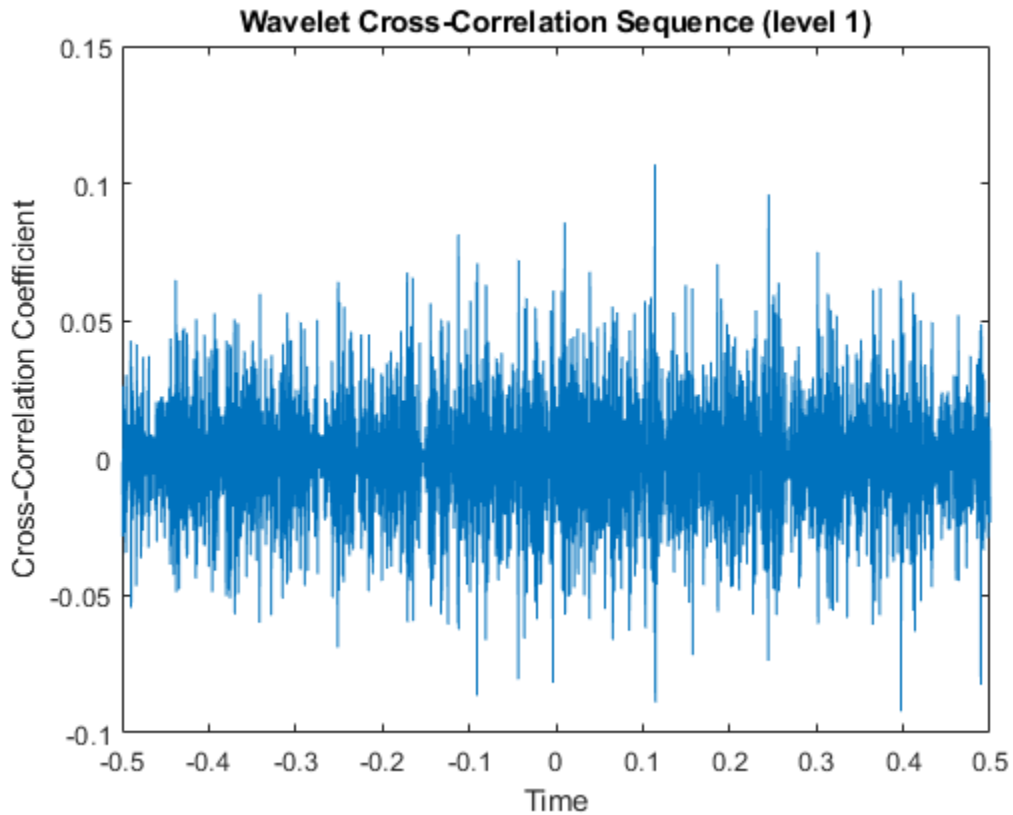
```
[xc,~,lags] = modwtcorr(wy,wx,'fk14');
lev = 3;
zerolag = floor(numel(xc{lev})/2+1);
tlag = lags{lev}(zerolag-999:zerolag+1000).*(1/2000);
figure
plot(tlag,xc{lev}(zerolag-999:zerolag+1000))
title('Wavelet Cross-Correlation Sequence (level 3)')
```

```
xlabel('Time')
ylabel('Cross-Correlation Coefficient')
```

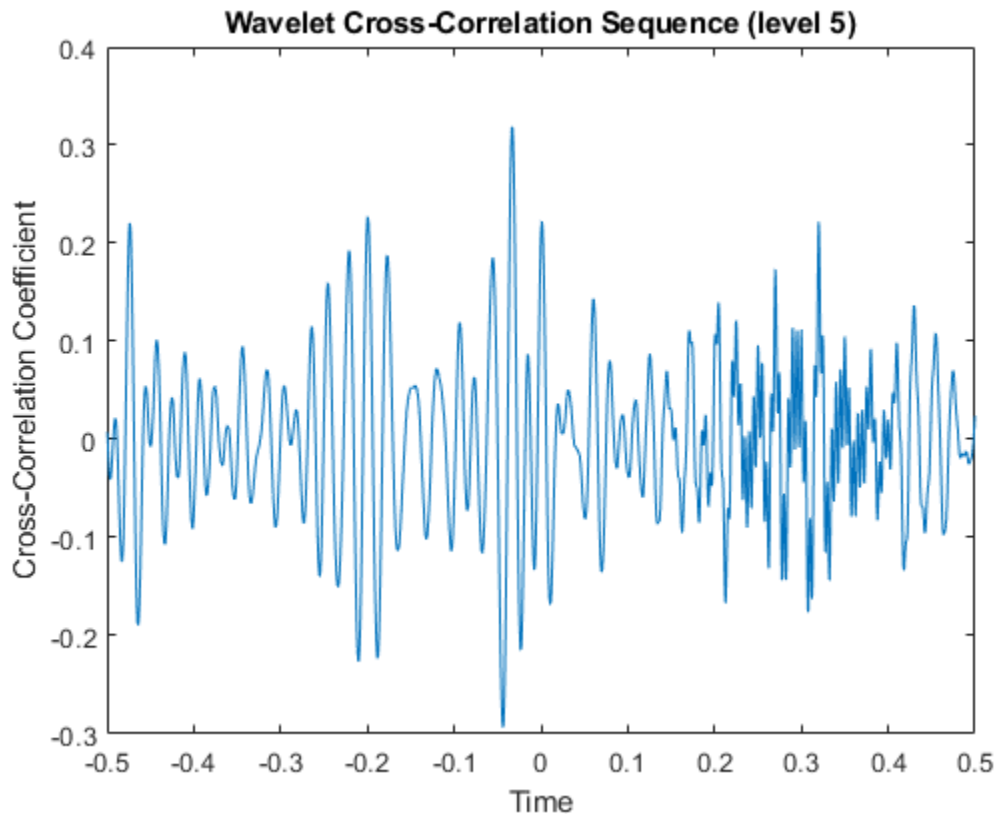


To show that wavelet cross-correlation enables scale(frequency)-localized correlation, plot the cross-correlation sequences at levels 1 and 5.

```
lev = 1;
zerolag = floor(numel(xc{lev})/2+1);
tlag = lags{lev}(zerolag-999:zerolag+1000).*(1/2000);
plot(tlag,xc{lev}(zerolag-999:zerolag+1000))
title('Wavelet Cross-Correlation Sequence (level 1)')
xlabel('Time')
ylabel('Cross-Correlation Coefficient')
```



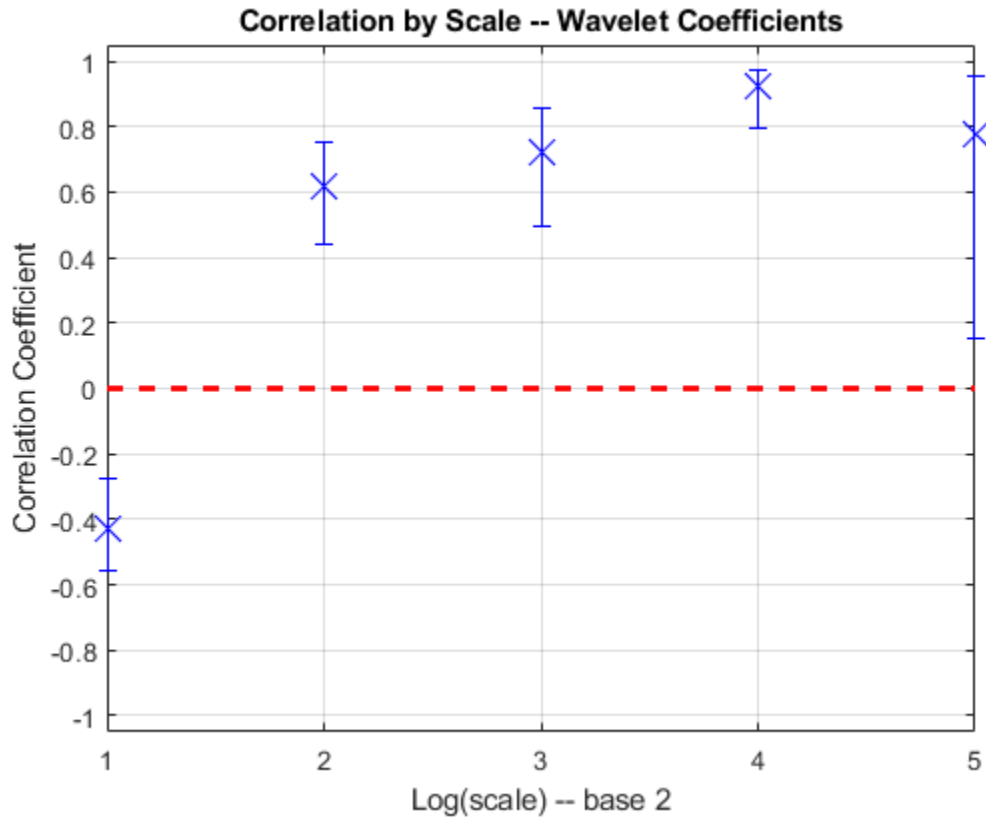
```
figure
lev = 5;
zerolag = floor(numel(xc{lev})/2+1);
tlag = lags{lev}(zerolag-999:zerolag+1000).*(1/2000);
plot(tlag,xc{lev}(zerolag-999:zerolag+1000))
title('Wavelet Cross-Correlation Sequence (level 5)')
xlabel('Time')
ylabel('Cross-Correlation Coefficient')
```



The wavelet cross-correlation sequences at levels 1 and 5 do not show any evidence of the exponentially-weighted sinusoids due to the bandpass nature of the wavelet transform.

With financial data, there is often a leading or lagging relationship between variables. In those cases, it is useful to examine the cross-correlation sequence to determine if lagging one variable with respect to another maximizes their cross-correlation. To illustrate this, consider the correlation between two components of the GDP -- personal consumption expenditures and gross private domestic investment. The data is quarterly chain-weighted U.S. real GDP data for 1974Q1 to 2012Q4. The data were transformed by first taking the natural logarithm and then calculating the year-over-year difference. Look at the correlation between two components of the GDP -- personal consumption expenditures, `pc`, and gross private domestic investment, `privateinvest`.

```
load GDPcomponents
piwt = modwt(privateinvest, 'fk8',5);
pcwt = modwt(pc, 'fk8',5);
figure
modwtcorr(piwt,pcwt, 'fk8')
```



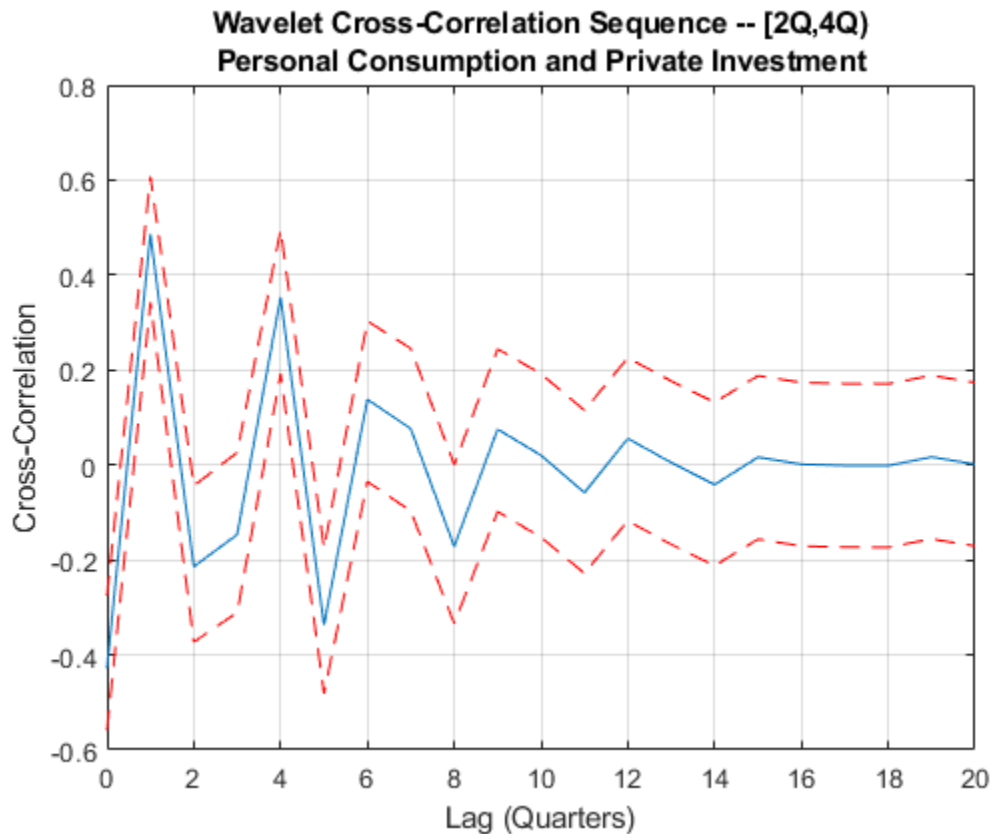
Personal expenditure and personal investment are negatively correlated over a period of 2-4 quarters. At longer scales, there is a strong positive correlation between personal expenditure and personal investment. Examine the wavelet cross-correlation sequence at the scale representing 2-4 quarter cycles. Plot the cross-correlation sequence along with 95% confidence intervals.

```
[xcseq,xcseqci,lags] = modwtcorr(piwt,pcwt, 'fk8');
zerolag = floor(numel(xcseq{1})/2)+1;
```

```

figure
plot(lags{1}(zerolag:zerolag+20),xcseq{1}(zerolag:zerolag+20))
hold on
plot(lags{1}(zerolag:zerolag+20),xcseqci{1}(zerolag:zerolag+20,:), 'r--')
xlabel('Lag (Quarters)')
ylabel('Cross-Correlation')
grid on
title({'Wavelet Cross-Correlation Sequence -- [2Q,4Q]'; ...
      'Personal Consumption and Private Investment'})

```



The finest-scale wavelet cross-correlation sequence shows a peak positive correlation at a lag of one quarter. This indicates that personal investment lags personal expenditures by one quarter. If you take that lagging relationship into account, then there is a positive correlation between the GDP components at all scales.

1-D Multisignal Analysis

This section takes you through the features of 1-D 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.

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
chgwdccfs	Change multisignal 1-D decomposition coefficients
wdecenergy	Multisignal 1-D decomposition energy repartition

Compression and Denoising Functions

Function Name	Purpose
mswcmp	Multisignal 1-D compression using wavelets
mswcmpscr	Multisignal 1-D wavelet compression scores
mswcmptp	Multisignal 1-D compression thresholds and performance
mswden	Multisignal 1-D denoising using wavelets
mswthresh	Perform multisignal 1-D thresholding

You can perform analyses from the MATLAB command line or by using the Wavelet Analyzer app. This section describes each method. The last section discusses how to exchange signal and coefficient information between the disk and the graphical tools.

1-D Multisignal Analysis — Command Line

- 1 Load a file, from the MATLAB prompt, by typing


```
load thinker
```

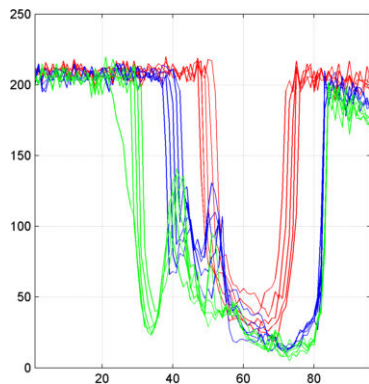
The file `thinker.mat` contains a single variable `X`. Use `whos` to show information about `X`.

```
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.

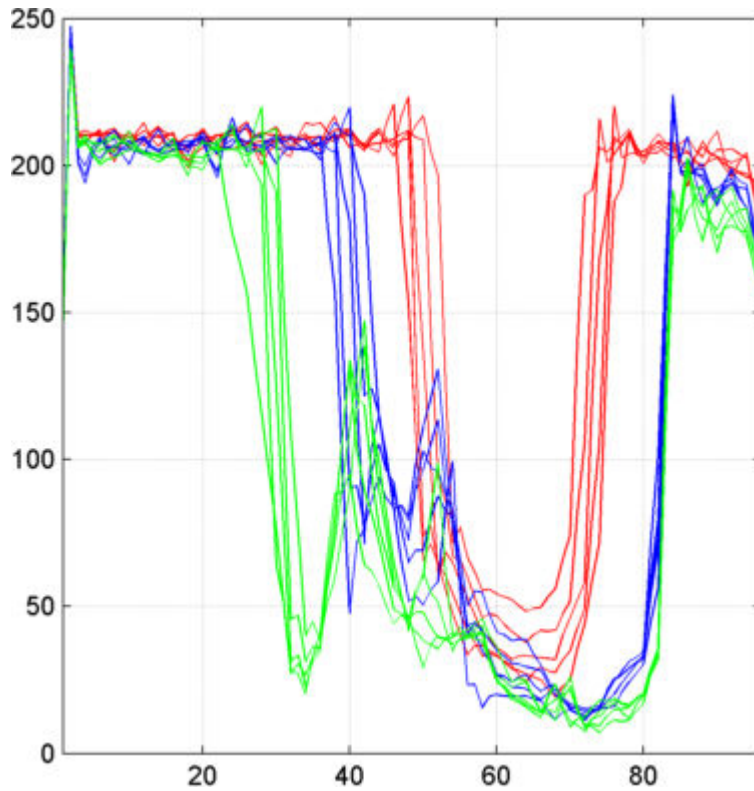
For each signal change the wavelet coefficients by setting all the coefficients of the detail of level 1 to zero.

```
decBIS = chgwdeccfs(dec, 'cd', 0, 1);
```

This generates a new decomposition structure `decBIS`.

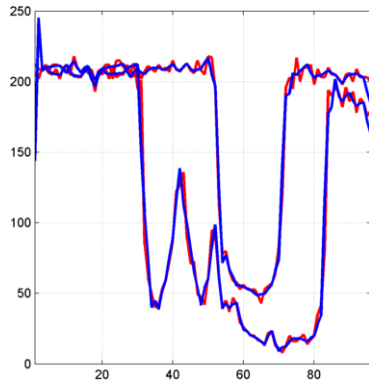
5 Perform a wavelet reconstruction of signals and plot some of the new signals.

```
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])
```



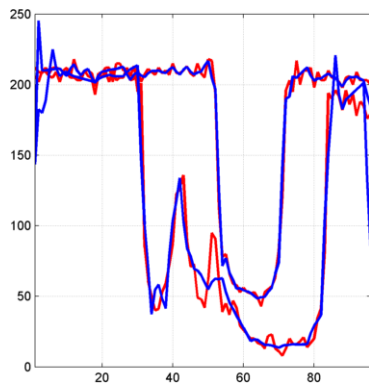
Compare old and new signals by plotting them together.

```
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.

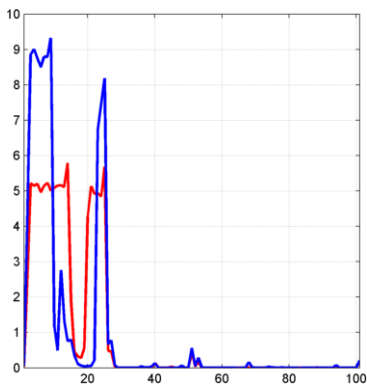
```
PEC_1_31 = PEC([1 31],:)
```

```
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)])
```

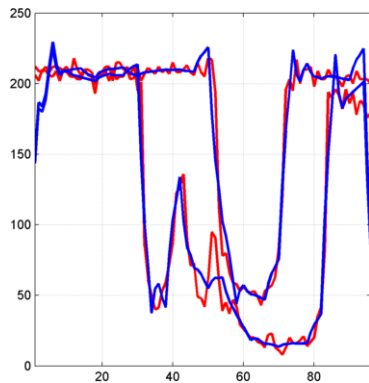


- 10** Compress the signals to obtain a percentage of zeros near 95% for the wavelet coefficients.

```
[XC,decCMP,THRESH] = mswcmp('cmp',dec,'N0_perf',95);
[Ecmp,PECmp,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);
```

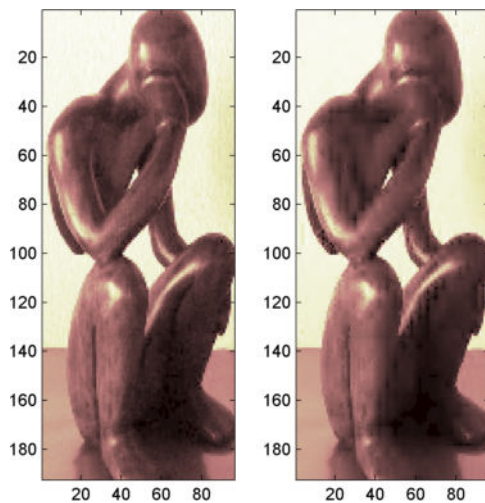
XC signals are the compressed versions of the original signals in the row direction.

Compress the XC signals in the column direction

```
XX = mswcmp('cmpsig','c',XC,'db2',2,'N0_perf',60);
```

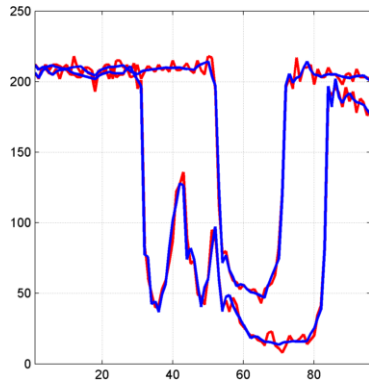
Plot original signals X and the compressed signals XX as images.

```
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] = mswden('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])
```



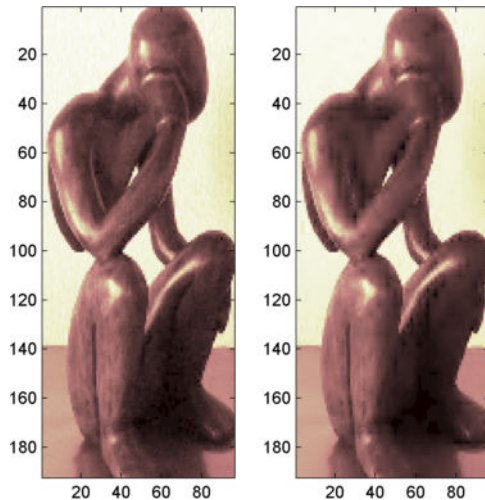
XD signals are the denoised versions of the original signals in the row direction.

Denoise the XD signals in column direction

```
XX = mswden('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))
```



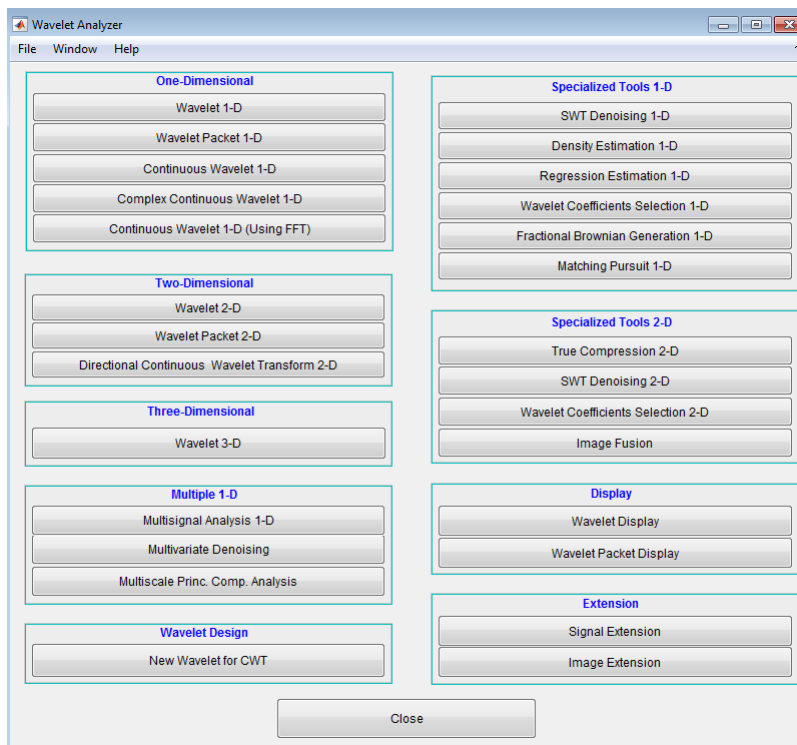
1-D Multisignal Analysis Using the Wavelet Analyzer App

In this section, we explore the same signal as in the previous section, but use the Wavelet Analyzer app to analyze it.

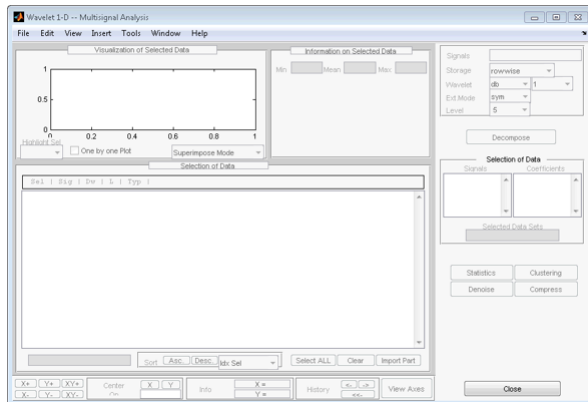
- 1 Start the Wavelet 1-D Multisignal Analysis Tool.

From the MATLAB prompt, type `waveletAnalyzer`.

The **Wavelet Analyzer** 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 app tools, 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.

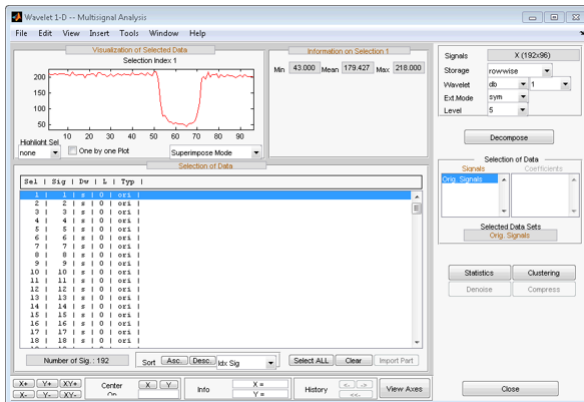
The three new panes are the **Visualization of Selected Data** pane, the **Information on Selected Data** pane, and the **Selection of Data** pane.

2 Load the signals.

At the MATLAB command prompt, type

```
load thinker
```

In the **Multisignal Analysis 1-D** tool, select **File > Import from Workspace > Import Signals**. When the **Import from Workspace** dialog box appears, select the X variable. Click **OK** to import the data matrix and display the first signal.



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.

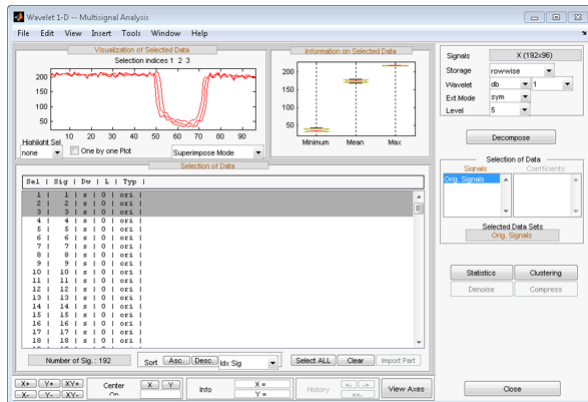
3 View the signals and signal information.

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.)

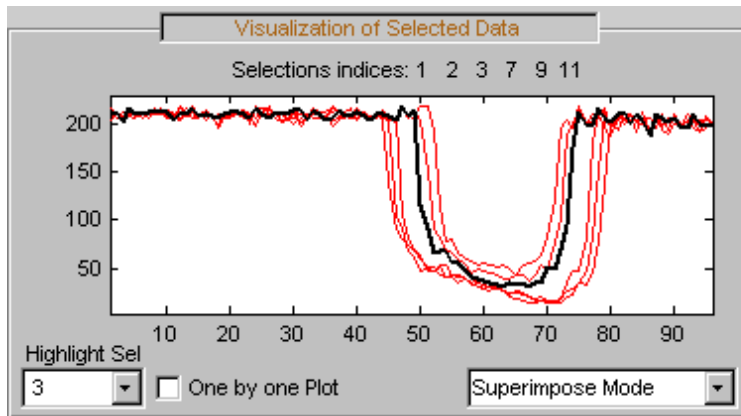
The selected signals (1, 2, 3, 7, 9 and 11) appear in the **Visualization of Selected Data** pane. The **Information on Selected Data** pane contains the box plots of the minimums, the means, and the maximums of these signals.

3 Discrete Wavelet Analysis



4 Highlight a signal.

Using the **Highlight Sel** button in the lower-left corner of the **Visualization of Selected Data** pane, select signal 3.



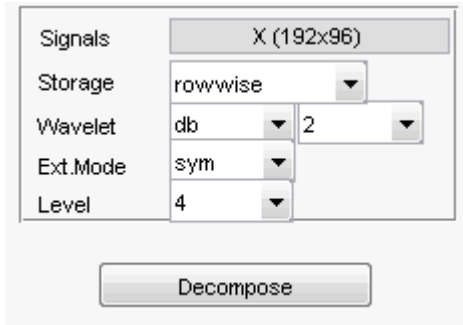
5 Select Different Views.

In the **Visualization of Selected Data** pane, change the view mode using the pop-up in the lower-right corner. Choose **Separate Mode**. The selected signals appear.

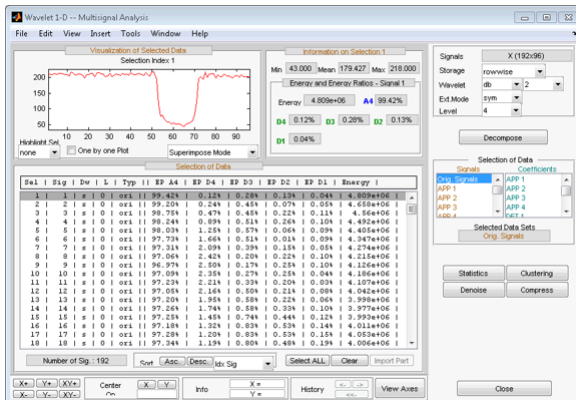
6 Decompose a multisignal.

Perform an analysis at level 4 using the db2 wavelet and the same file used in the command line section: `thinker.mat`.

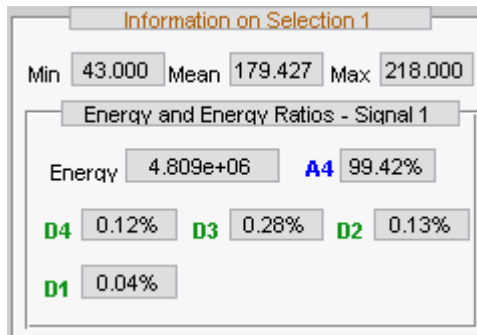
In the upper right portion of the Wavelet 1-D Multisignal Analysis tool, select db2 and level 4 in the Wavelet fields.



Click **Decompose**. After a pause for computation, all the original signals are decomposed and signal 1 is automatically selected

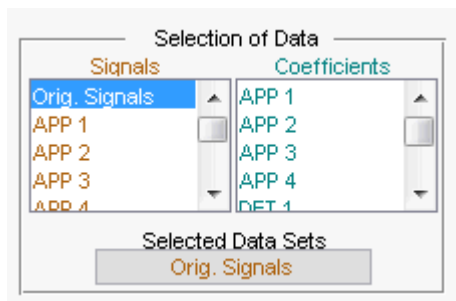


In the **Selection of Data** pane, new information is added for each original signal — the percentage of energy of the wavelet components (**D1**,...,**D4** and **A4**) and the total energy. The **Information on Selected Data** pane contains information on the single selected signal: **Min**, **Mean**, **Max** and the energy distribution of the signal.



Since the original signals are decomposed, new objects appear and the **Selection of Data Sets** pane in the Command Frame updates.

The **Selection of Data Sets** pane defines the available signals that are now selectable from the **Selection of Data** pane.

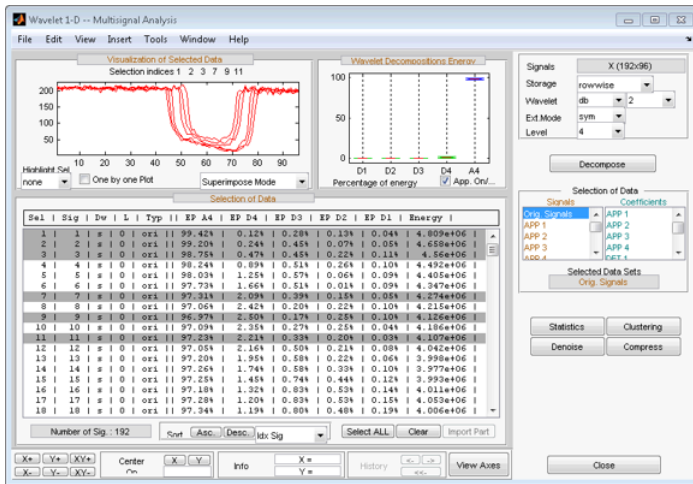


The list on the left allows you to select sets of signals and the right list allows you to select sets of corresponding coefficients: original signals (Orig. Signals), approximations (APP 1,...)and details from levels 1 to 4 (DET 1,...).

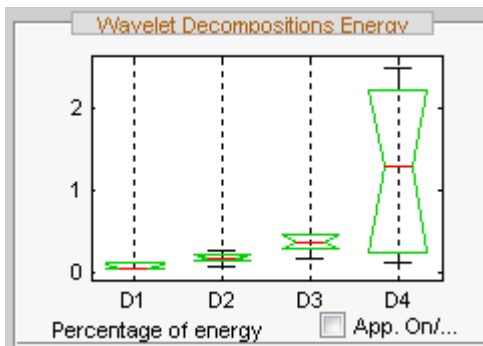
In the list on the right, the coefficients vectors can be of different lengths, but only components of the same length can be selected together.

After a decomposition the original signals (Orig. Signals) data set appears automatically selected.

Select signals 1, 2, 3, 7, 9 and 11.

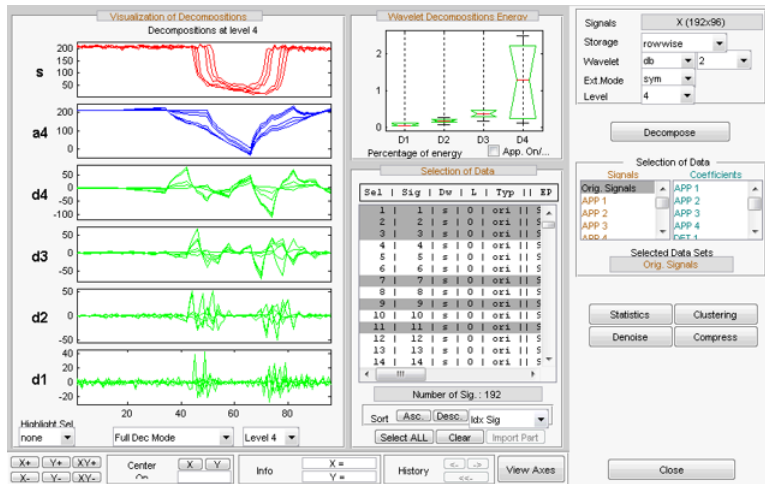


The energy of selected signals is primarily concentrated in the approximation A4, so the box plot is crushed (see following figure on the left). Deselect **App. On/Off** to see a better representation of details energy (see following figure on the right).



7 Display multisignal decompositions.

In the **Visualization of Selected Data** pane, change the view mode using the pop-up below the plots and select **Full Dec Mode**. The decompositions of the selected signals display.



Change the **Level** to 2.

Select the signal 7 in **Highlight Sel**.

- 8 Change the visualization modes.

Using the second pop-up from the left at the bottom of the pane, select **Full Dec Mode (Cfs)**. The coefficients of the decompositions of the selected signals display. At level k , coefficients are duplicated $2k$ times.

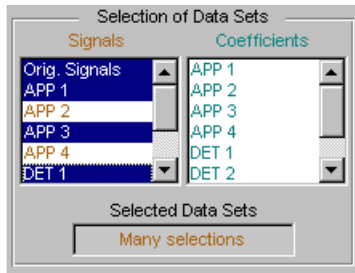
Change the view mode to **Stem Mode (Abs)**, and then change to **Tree Mode**. The wavelet tree corresponding to the decompositions of the selected signals displays.

Select the level 4 and click the node a3. Then highlight signal 7.

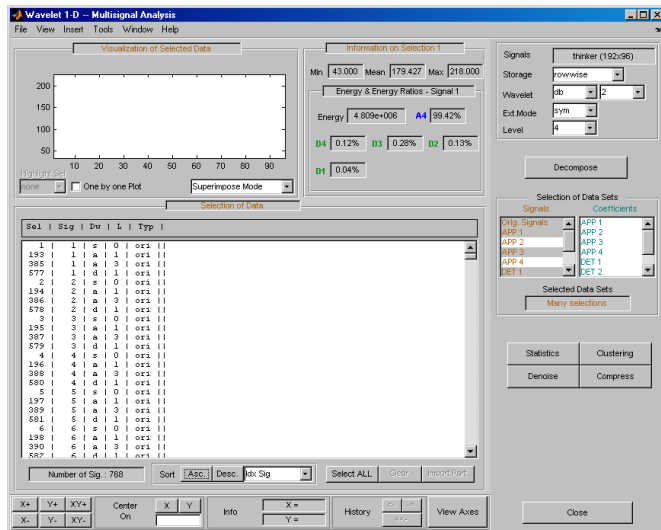
- 9 Select Different Wavelet Components.

Ctrl-click **Orig. Signals**, **APP 1**, **APP 3** and **DET 1** to select these four sets of signals from the list on the left in the **Selection of Data Sets** pane.

The total number of selected data (**Number of Sig.**) appears in the **Selection of Data Sets** pane: four sets of 192 signals each is a total of 768 signals.



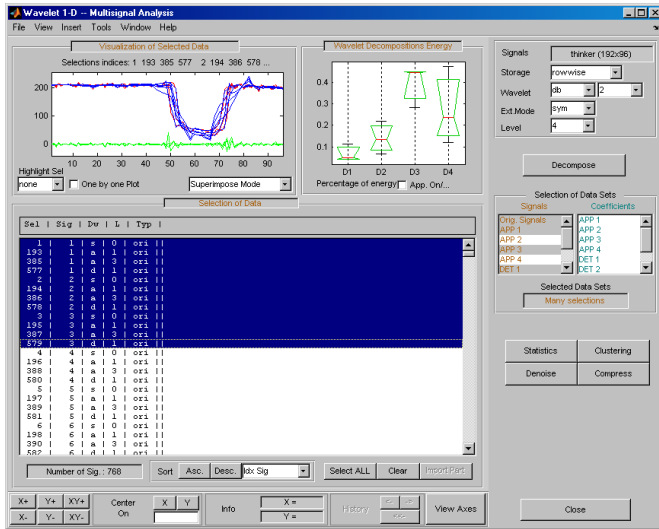
Click the **Asc.** button in the **Sort** pane. The selected data are sorted in ascending order with respect to the **Idx Sig** parameter



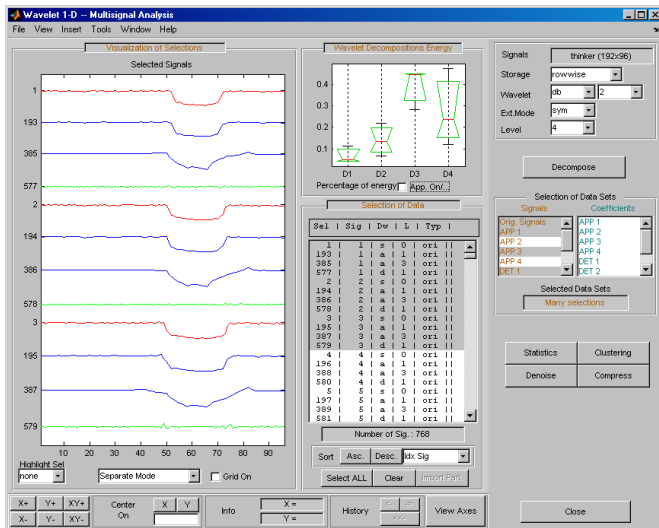
Note that DWT attributes of each selectable signal have been updated where **a** stands for approximation, **d** for detail and **s** for signal.

Click the **Idx Sel 1** signal and then shift-click the **Idx Sel 579** signal.

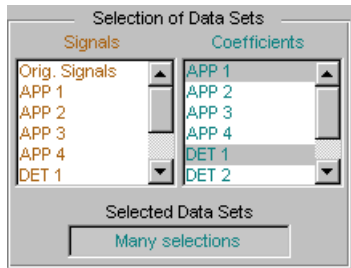
3 Discrete Wavelet Analysis



Choose Separate Mode.



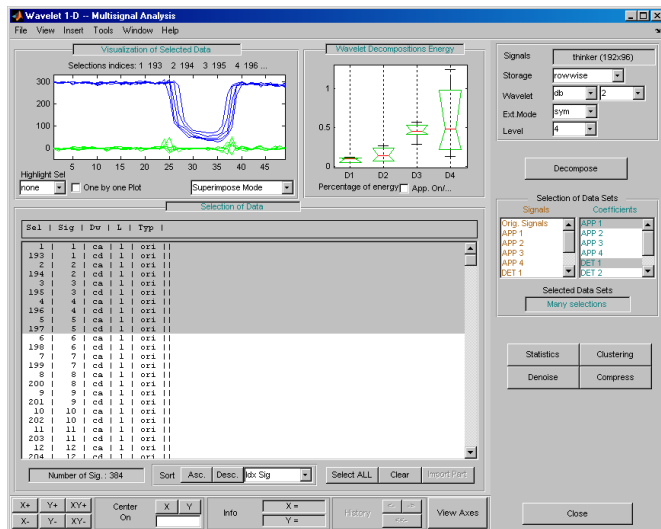
Ctrl-click to select two sets of signals from the right-most list of the **Selection of Data Sets** pane: APP 1 and DET 1.



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.

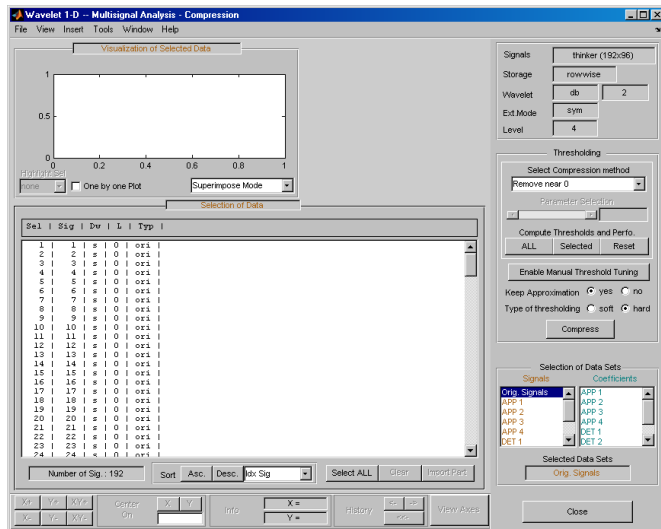
Select the ten first signals.



10 Compress a multisignal.

The Wavelet Analyzer app features 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

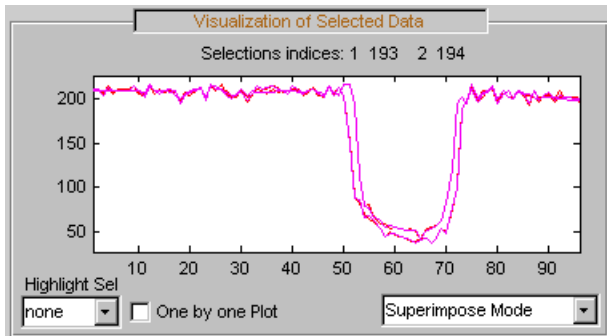
The thresholds for each level (**ThrD1** to **ThrD4**), the energy ratio (**En. Rat.**) and the sparsity ratio (**NbZ Rat.**) are displayed in the **Selection of Data** pane.

Click the **Compress** button at the bottom of the **Thresholding** pane. Now you can select new data sets: compressed Signals, the corresponding approximations, details and coefficients.

Press the **Ctrl** key and click the **Compressed** item in the left list of the **Selection of Data Sets** pane. The original signals and their compressed versions are selected (2 x 192 = 384 signals).

Click the **Asc.** button at the bottom of the **Selection of Data** pane to sort the signals using **Idx Sig** number.

With the mouse, select the first four signals. They correspond to the original signals 1, 2 and the corresponding compressed signals 193, 194.

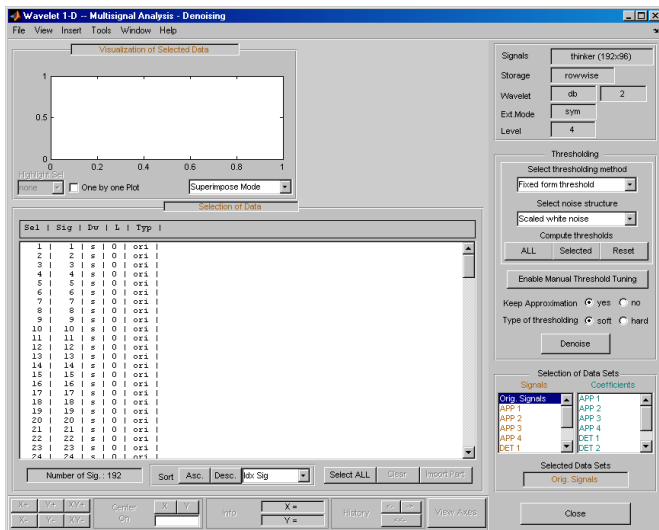


Click the **Close** button to close the Compression window.

11 Denoise a multisignal.

The Wavelet Analyzer app 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.

Sig	ThrD1	ThrD2	ThrD3	ThrD4
1	13.295	13.295	13.295	13.295
2	15.449	15.449	15.449	15.449
3	12.365	12.365	12.365	12.365
4	17.006	17.006	17.006	17.006
5	11.194	11.194	11.194	11.194
6	13.107	13.107	13.107	13.107
7	12.166	12.166	12.166	12.166
8	17.748	17.748	17.748	17.748
9	7.881	7.881	7.881	7.881
10	12.794	12.794	12.794	12.794
11	12.208	12.208	12.208	12.208
12	11.351	11.351	11.351	11.351
13	12.794	12.794	12.794	12.794
14	15.020	15.020	15.020	15.020
15	12.721	12.721	12.721	12.721
16	18.020	18.020	18.020	18.020
17	16.107	16.107	16.107	16.107
18	16.420	16.420	16.420	16.420
19	16.891	16.891	16.891	16.891
20	12.365	12.365	12.365	12.365
21	16.891	16.891	16.891	16.891
22	15.135	15.135	15.135	15.135
23	18.647	18.647	18.647	18.647
24	14.654	14.654	14.654	14.654

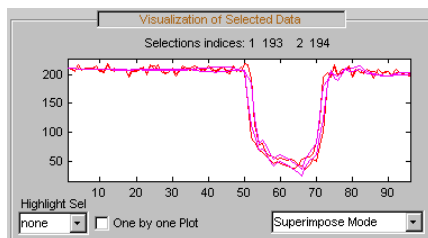
Number of Sig.: 192 Sort Asc. Desc. Idx Sig Select ALL Clear Import Part

Then click the **Denoise** button at the bottom of the **Thresholding** pane.

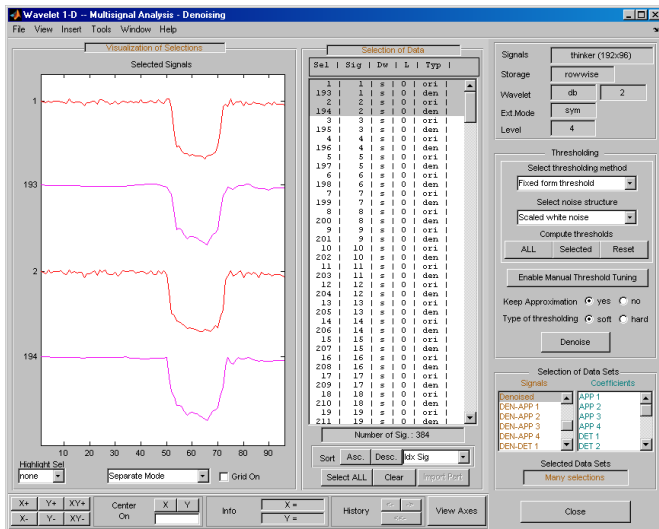
Ctrl-click the **Denoised** item in the list on the left of the **Selection of Data Sets** pane. The original signals and the corresponding denoised ones are selected (2 x 192 = 384 signals).

Click the **Asc.** button at the bottom of the **Selection of Data** pane to sort the signals according to the Idx Sig parameter.

With the mouse, select the first four signals. They correspond to the original signals 1, 2 and the corresponding denoised signals 193, 194



Choose Separate Mode.

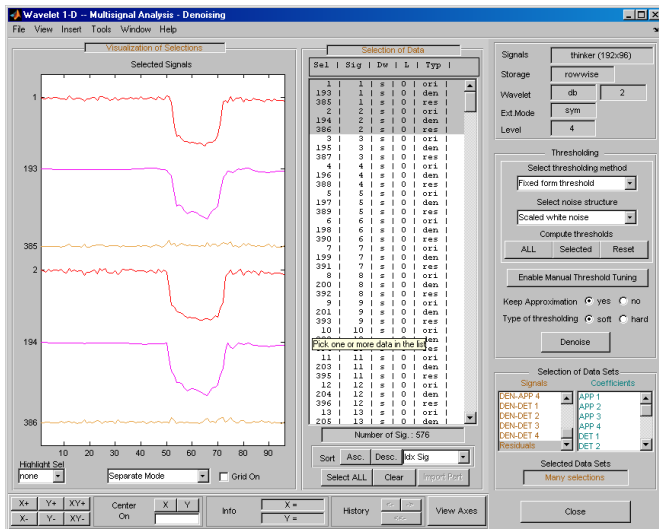


12 To view residuals, Ctrl-click the **Orig.** Signal, the **Denoised** and the **Residuals** items in the list on the left of the **Selection of Data Sets** pane. Original, denoised and residual signals are selected (3 x 192 = 576 signals).

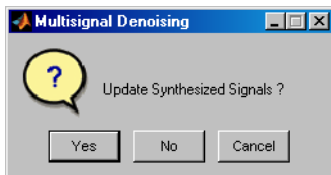
Click the **Asc.** button at the bottom of the **Selection of Data** pane to sort the signals using the **Idx Sig** parameter.

With the mouse, select the first six signals. They correspond to the original signals 1, 2, the corresponding denoised signals 193, 194 and the residuals 385, 386.

Then, choose **Separate Mode**.

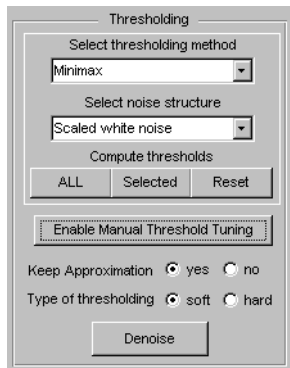


- 13 Click **Close** to close the denoising tool. Then, click the **Yes** button to update the synthesized signals.

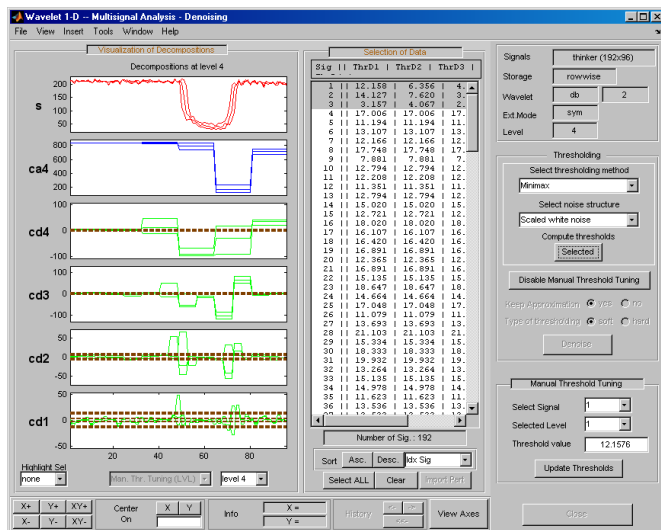


Manual Threshold Tuning

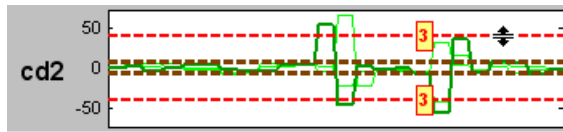
- 1 Choose a method, select one or several signals in the **Selection of Data** pane using the mouse and keys. Then click the **Selected** button. You can select another group of signals using the same method. Press the **Denoise** button to denoise the selected signal(s).



You can also use manual threshold tuning. Click the **Enable Manual Threshold 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

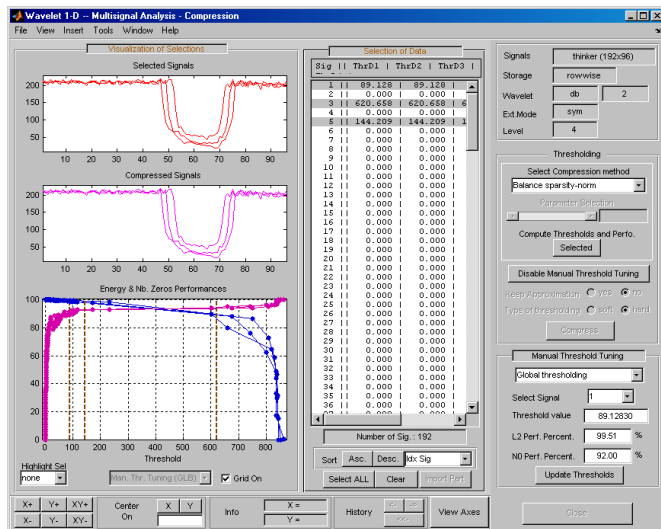
Select Signal: 3

Selected Level: 2

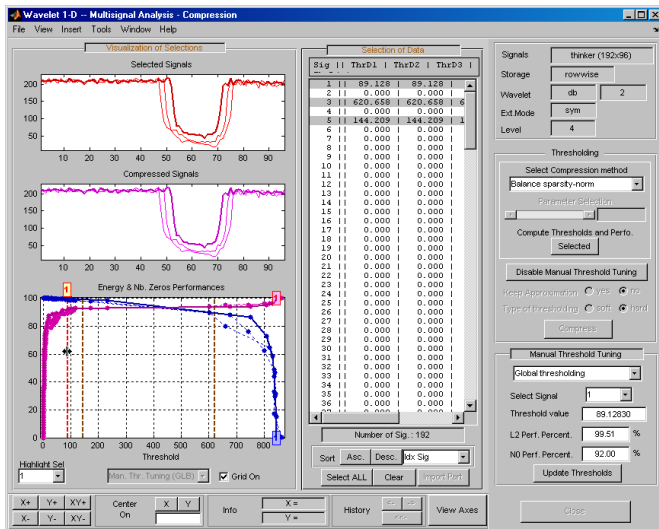
Threshold value: 42.51011

Update Thresholds

- 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**.



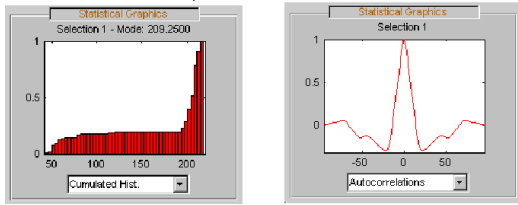
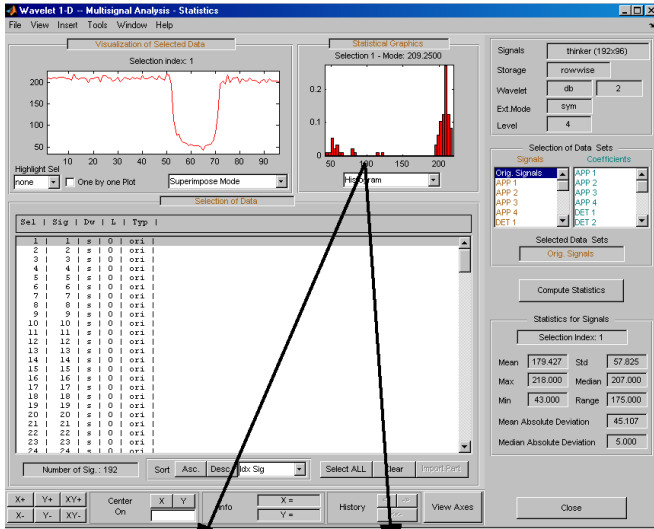
You can drag the vertical lines in the **Energy and Nb. Zeros Performances** axes using the mouse. This can be done individually or all together depending on the values of **Select Signal** in the **Manual Threshold Tuning** pane.



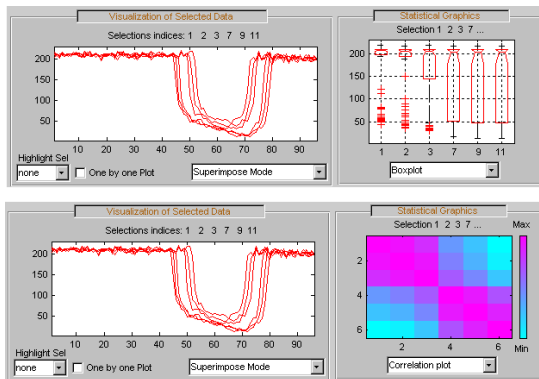
The threshold value, L2 performance, and number of zeros performance are updated in the corresponding edit buttons in the **Manual Threshold Tuning** pane.

Statistics on Signals

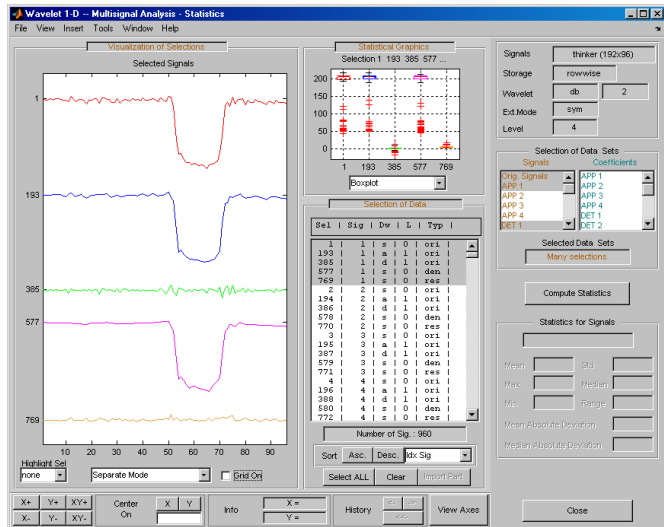
- 1 You can display various statistical parameters related to the signals and their components. From the Wavelet 1-D Multisignal Analysis tool, click the **Statistics** button. Then select the signal 1 in the **Selection of Data Sets** pane.



Select the signals 1, 2, 3, 7, 9 and 11 in the **Selection of Data** pane, and display the corresponding boxplots and correlation plots.



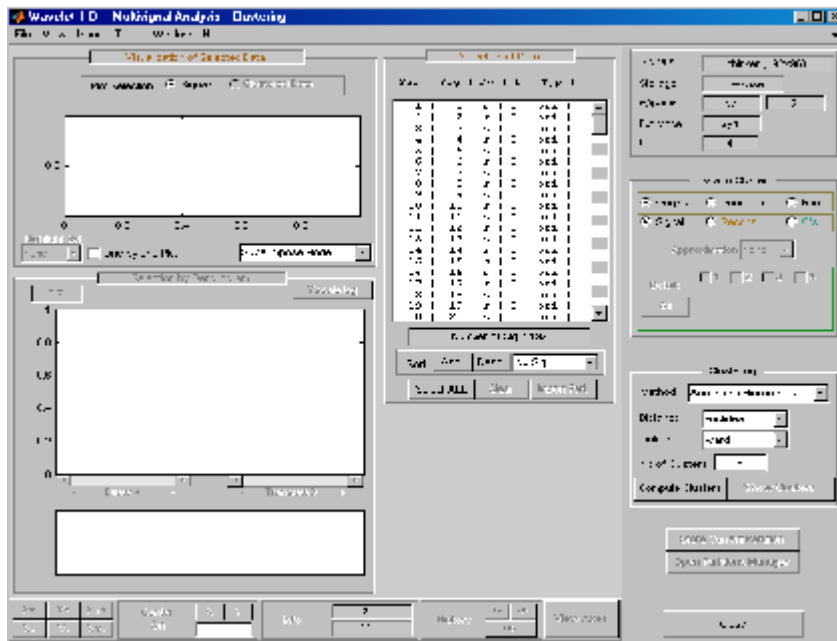
- To display statistics on many wavelet components, in the **Selection Data Sets** pane, in the left column, select **Orig. Signals**, **APP 1**, **DET 1**, **Denosed** and **Residuals** signals. Then choose **Separate Mode**, and click the **Asc.** button in the **Sort** pane. The selected data are sorted in ascending order with respect to **Idx Sig** parameter. In the **Selection of Data** pane, select data related to signal 1.



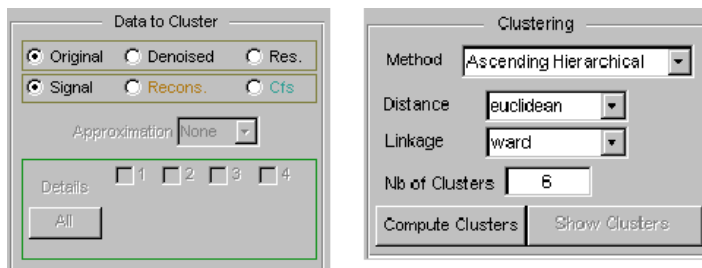
Clustering Signals

Note To use clustering, you must have Statistics and Machine Learning Toolbox™ software installed. For more information about clustering, including measuring distances between objects and determining the proximity of objects to each other, see “Hierarchical Clustering” (Statistics and Machine Learning Toolbox).

- 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.

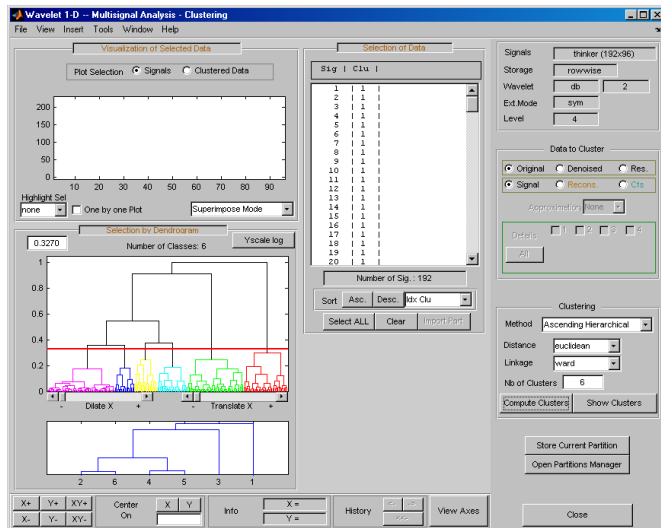


You can cluster various type of signals and wavelet components: original, denoised or compressed, residuals, and approximations or details (reconstructed or coefficients). Similarly, there are several methods for constructing partitions of data.



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.

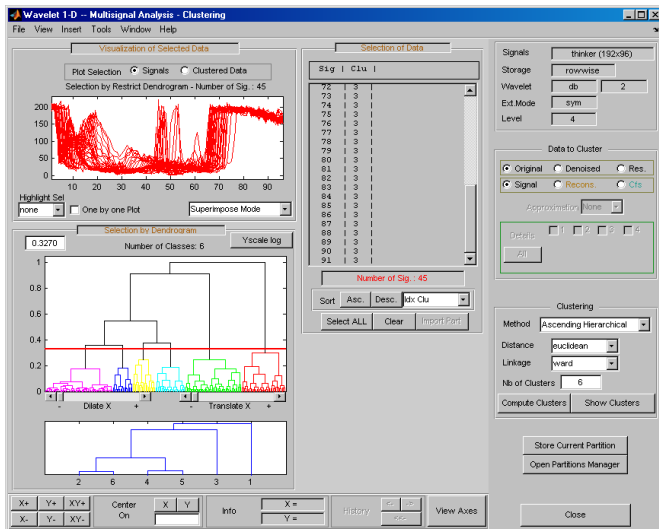
A full dendrogram and a restricted dendrogram display in the **Selection by Dendrogram** pane. For each signal, the cluster number displays in the **Selection of Data** pane.



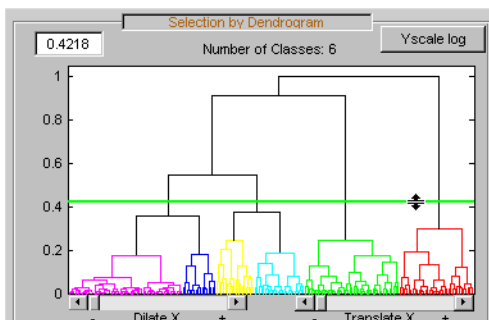
- 2 Select one cluster, several clusters, or a part of a cluster.

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.



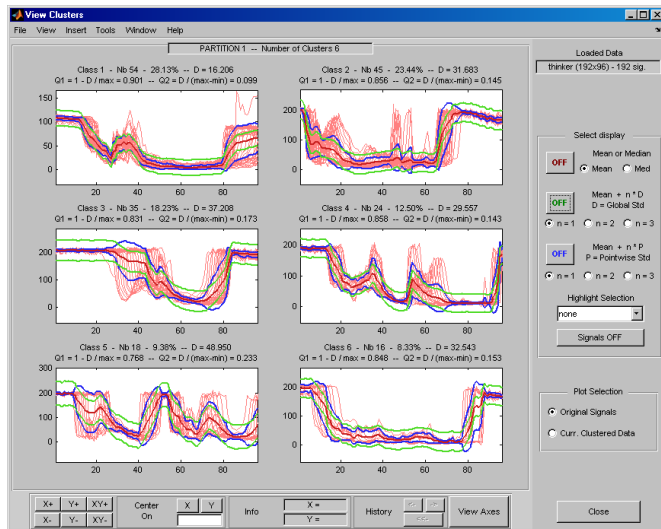
You can use the horizontal line in the full dendrogram to change the number of clusters. Use the left mouse button to drag the line up or down.



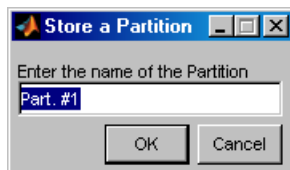
- Use the **Show Clusters** button to examine the clusters of the current partition. You can display the mean (or the median) of each cluster, the global standard deviation and the pointwise standard deviation distance around the mean (or the median). Each plot title contains: the number of signals in the cluster (Nb) and percent of total, the global standard deviation (D) of the cluster, and two indices of quality, Q1 and Q2.

The Q1 and Q2 indices can be interpreted as measures of how well a cluster is concentrated in multidimensional space. The values `min` and `max` are the minimum and maximum values, respectively, of the absolute values of the signal samples in the

cluster. If the global standard deviation D is small, then $Q1$ is close to 1 and $Q2$ is close to 0. In this case, the cluster is considered well-concentrated. If D is large, then $Q1$ is small and $Q2$ is large. In this case, the cluster is considered more disperse. For example, in Cluster 3 the minimum and maximum values of the absolute values of the signal samples are 5 and 220, respectively. Then $Q1 = 1 - 37.208/220 = 0.831$ and $Q2 = 37.208/(220-5) = 0.173$.



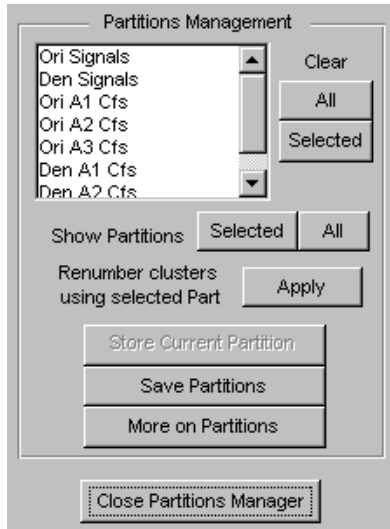
- 4 Click the **Store Current Partition** button below the **Clustering** pane to store the current partition for further comparisons. A default name is suggested. Note that the 1-D Wavelet Multisignal Analysis tool stores the partitions and they are not saved on the disk.



Partitions

- 1 Build and store several partitions (for example, partitions with signals, denoised signals approximations at level 1, 2 and 3, and denoised signals). Then, click the

Open Partition Manager button below the **Store Current Partition** button. The **Partitions Management** pane appears. The names of all stored partitions are listed.

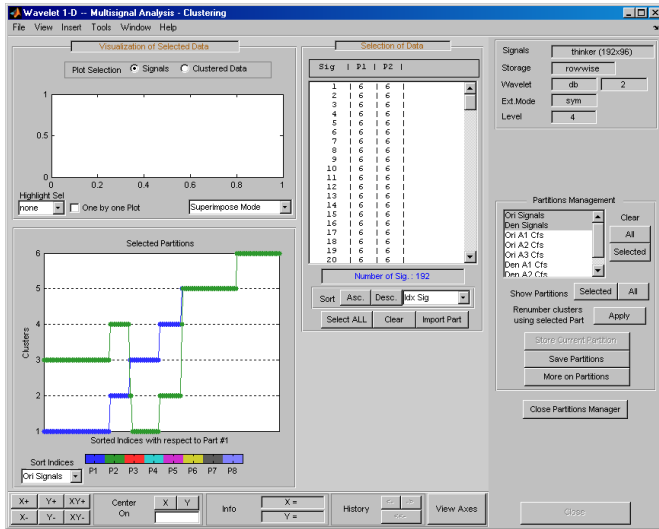


Now, you can show, clear, or save the partitions (individually, selected ones, or all together).

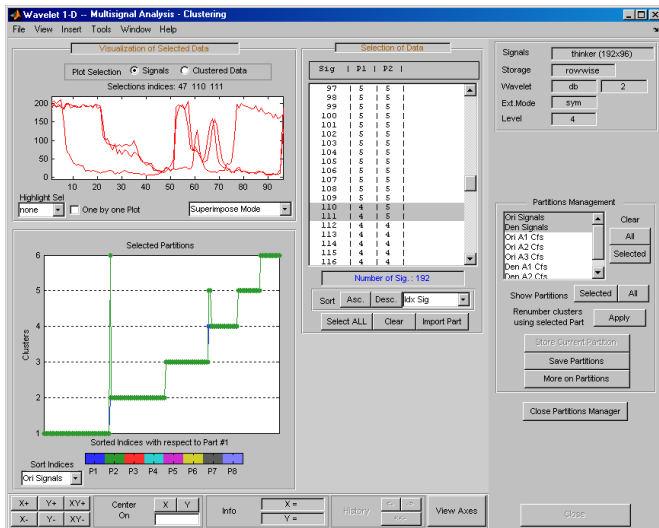
- 2 To display partitions, select the **Ori Signals** and the **Den Signals** partitions, and click the **Selected** button next to the **Show Partitions** label.

The clusters are almost the same, but it is difficult to see this on the **Selected Partitions** axis, due to the scaling difference. Press the **Apply** button to renumber the clusters (starting from the selected partition as basic numbering) to compare the two partitions.

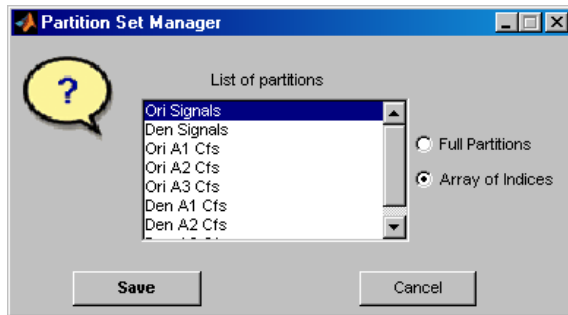
3 Discrete Wavelet Analysis



Only three signals are not classified in the same cluster for the two considered partitions.

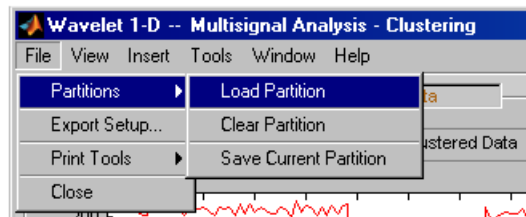
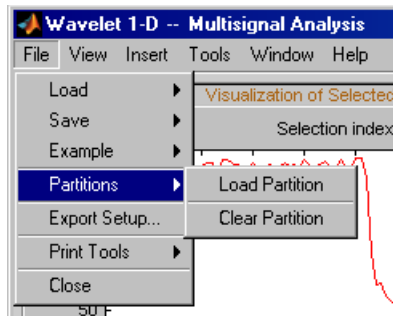


- 3 Select the partitions you want to save and click the **Save Partitions** button below the **Store Current Partition** button in the **Partitions Management** pane.

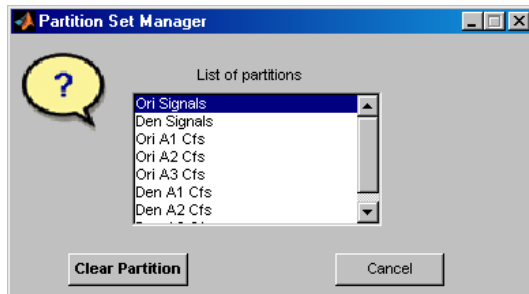


Partitions are saved as an array of integers, where each column corresponds to one partition and contains the indices of clusters. When you choose the Full Partitions option, an array object (wpartobj) is saved.

- 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.)



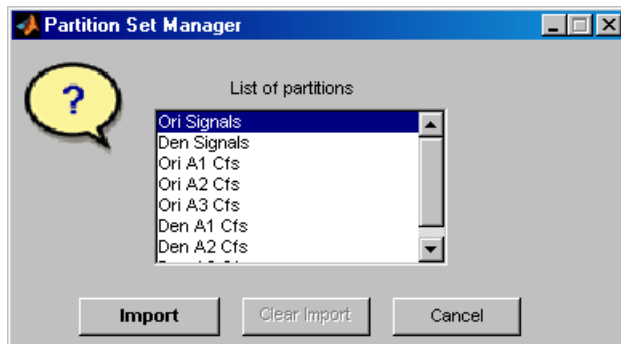
To clear one or more stored partitions, select **File > Partitions > Clear Partition**.



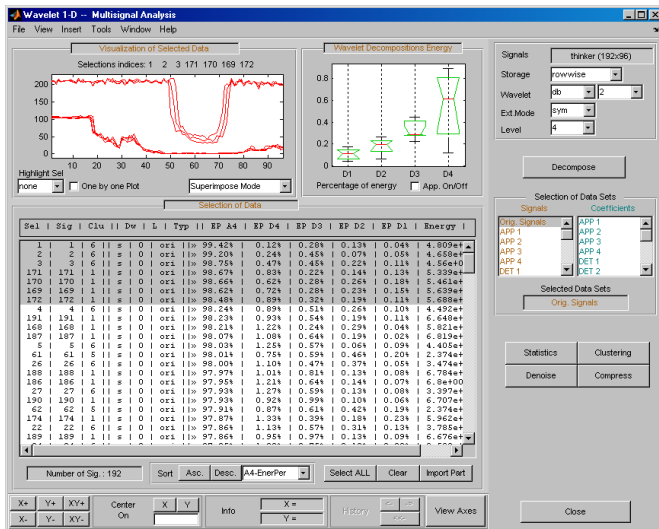
Select **File > Partitions > Load Partition** to load one or several partitions from the disk. The loaded partitions are stored in Wavelet 1-D Multisignal Analysis tool with any previously stored partitions. A partition can also be a manually created column vector.

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.

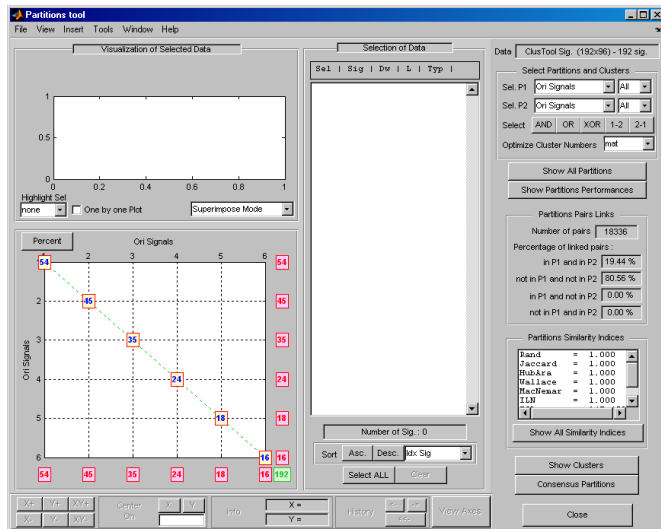
- 5 In each subcomponent of the Wavelet 1-D Multisignal Analysis tool (main, statistics, denoising, compression, clustering), you can import a stored partition from the list in the **Selection of Data** pane. Click the **Import Part** button at the bottom of the **Selection of Data** pane, the Partition 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 You can compare partitions with the Partition tool. To display the Partition tool, click the **More on Partitions** button at the bottom of the Partitions Management pane. By default, when the Partition tool opens, the currently selected partition, in this case Ori Signals, is compared with itself. In the lower left plot, an integer N at (i, j) means there is a group of N signals in the i^{th} cluster of P1 and the j^{th} cluster of P2. Since the Partition tool is comparing a partition with itself, all the numbers are plotted on the main diagonal.



Statistics measuring the similarity of the partitions P1 and P2 are displayed in the panels on the right. The **Partitions Pairs Links** panel counts pairs of signals. Two signals x and y are considered a *pair* if they are in the same cluster. For any two signals x and y , there are four possibilities.

- The signals are paired in P1 and P2. This is called a *true positive*.
- The signals are paired in P1 but not in P2. This is called a *false positive*.
- The signals are not paired in P1 but are in P2. This is called a *false negative*.
- The signals are not paired in either P1 or P2. This is called a *true negative*.

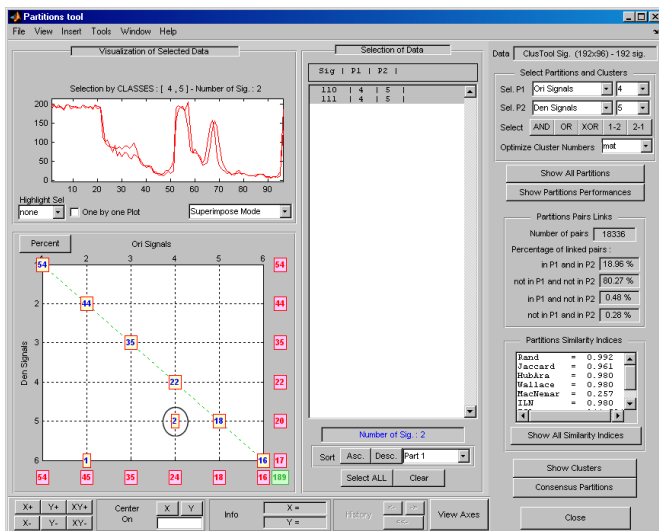
The **Partitions Pairs Links** shows the percentages of each of the four possibilities. In this example, since there are 192 signals, there are total of $192 \cdot 191 / 2 = 18336$ possible pairs. Slightly more than 19% of the linked pairs are considered true positives, and approximately 80% are true negatives. Since the partition is being compared with itself, no pair is considered a false positive or false negative.

You can also measure how similar two partitions are by assigning a distance (or index) between them [1]. The distance is based on the number of true and false positives and negatives. Distance can be defined in different ways. Let R be the number of true positives, S be the number of true negatives, U be the number of false positives, and V be the number of false negatives. The **Partitions Similarity Indices**

panel shows the distance between the P1 and P2 partitions using different definitions of distance:

- Rand Index (Rand): $(R+S)/(R+S+U+V)$.
- Asymmetric Rand Index (RandAsym): $(2 \times (R+S+U)+NS)/NS^2$ where NS is the total number of signals.
- Jaccard Index (Jaccard): $R/(R+U+V)$.
- Corrected Rand Index (HubAra): $(R-ER)/(MR-ER)$ where ER is the expected value of R and MR is the maximum value of the index. If the two partitions are identical, the index is equal to 1. It is possible for this index to be negative.
- Wallace Index (Wallace): $R/\sqrt{(|\pi(P1)| \times |\pi(P2)|)}$ where $|\pi(Pi)|$ is the number of joined pairs in the partition Pi .
- McNemar Index (MacNemar): If $\text{abs}(U+V) = 0$, then the index is equal to 1. Otherwise, the index is equal to $\text{abs}(U-V)/(U+V)$.
- Lerman Index (ICL): $(R-ER)/\sqrt{VR}$ where VR is the variance of R .
- Normalized Lerman Index (ILN): $ICL(P1,P2)/\sqrt{ICL(P1,P1) \times ICL(P2,P2)}$ where $ICL(P,Q)$ denotes the Lerman index of two partitions P and Q .

- 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.

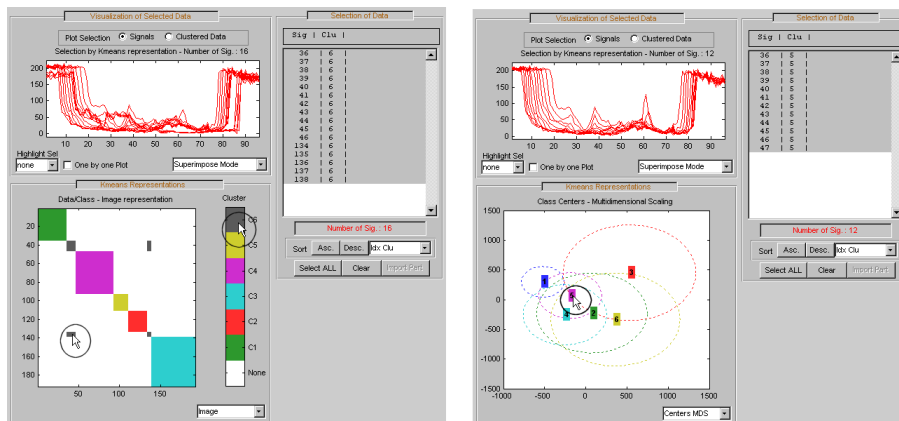


More on Clustering

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.

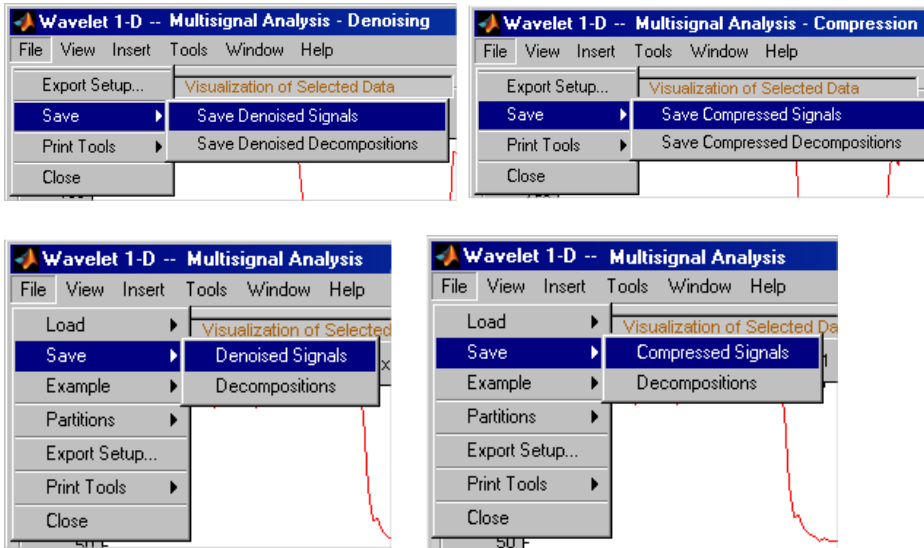


Importing and Exporting Information from the Wavelet Analyzer App

The Wavelet 1-D Multisignal Analysis tool lets you move data to and from disk.

Saving Information to Disk

You can save decompositions and denoised or compressed signals (including the corresponding decompositions from Wavelet 1-D Multisignal Analysis tools) to disk. You then can manipulate the data and later import it again into the graphical tools.



Saving Decompositions

The Wavelet 1-D Multisignal Analysis main tool lets you save the entire set of data from a wavelet analysis to disk. The toolbox creates a MAT-file in the current folder with a name you choose.

- 1 Open the Wavelet 1-D Multisignal Analysis main tool and load the example analysis by selecting **File > Example > Ex 21: Thinker (rows)**.
- 2 Save the data from this analysis, using the menu option **File > Save Decompositions**.

A dialog box appears that lets you specify a folder and filename for storing the decomposition data. For this example, use the name `decORI.mat`.

- 3 Type the name `decORI`.
- 4 After saving the decomposition data to the file `decORI.mat`, load the variables into your workspace:

```
load decORI
whos
```

Name	Size	Bytes	Class
dec	1x1	163306	struct

```
dec
dec =
    dirDec: 'r'
    level: 4
    wname: 'db2'
    dwtFilters: [1x1 struct]
    dwtEXTM: 'sym'
    dwtShift: 0
    dataSize: [192 96]
             ca: [192x8 double]
             cd: {1x4 cell}
```

The field `ca` of the structure `dec` gives the coefficients of approximation at level 4, the field `cd` is a cell array which contains the coefficients of details.

```
size(dec.cd{1})
ans =
    192    49
size(dec.cd{2})
ans =
    192    26
size(dec.cd{3})
ans =
    192    14
size(dec.cd{4})
ans =
    192     8
```

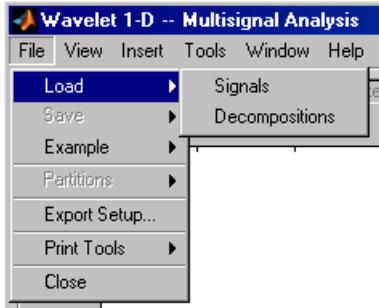
You can change the coefficients using the `chgwdeccfs` function.

Note For a complete description of the `dec` structure, see “Loading Decompositions” on page 3-184.

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 'r' for row or 'c' for column
<code>'level'</code>	Level of DWT decomposition
<code>'wname'</code>	Wavelet name
<code>'dwtFilters'</code>	Structure with four fields: LoD, HiD, LoR, HiR
<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 X
<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>

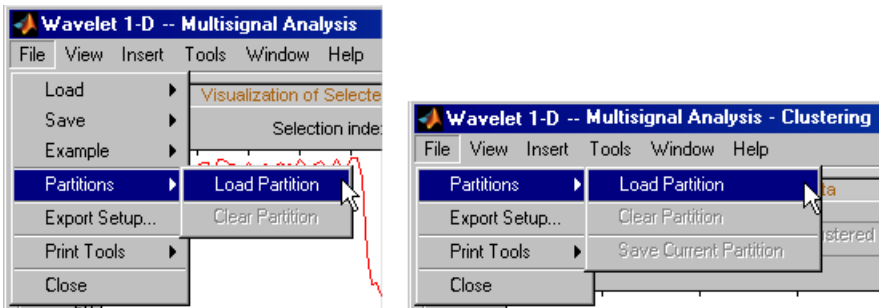
The coefficients `ca` and `cd{k}`, for $k = 1$ to `dec.level`, are matrices and are stored row-wise if `dec.dirDec` is equal to 'r' or column-wise if `dec.dirDec` is equal to 'c'.

Note The fields `'wname'` and `'dwtFilters'` have to be compatible (see the `wfilters` function). The sizes of `ca` and `cd{k}`, (for $k = 1$ to `dec.level`) must be compatible with the direction, the level of the decomposition, and the extension mode.

Loading and Saving Partitions.

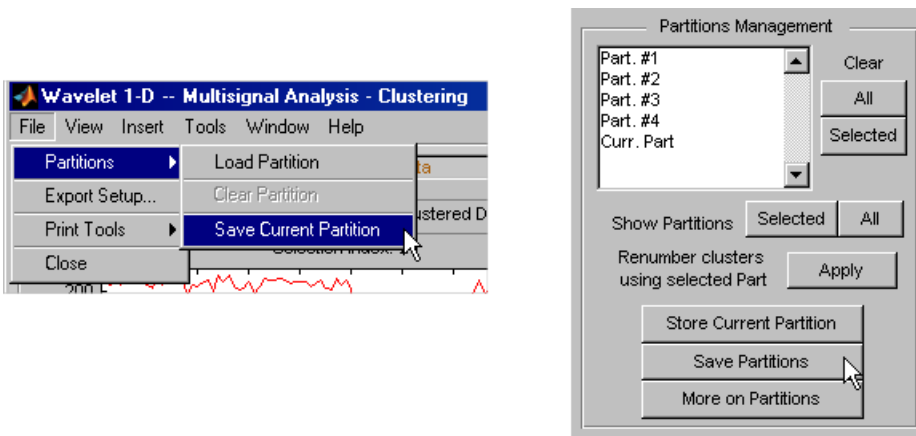
Loading

The Wavelet 1-D Multisignal Analysis main tool and clustering tool let you load a set of partitions from disk.



Saving Partitions

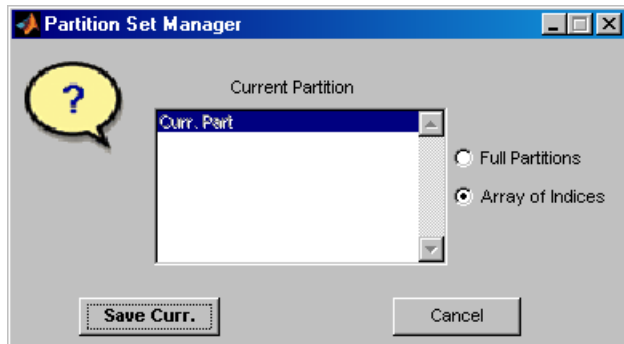
The Wavelet 1-D Multisignal Analysis clustering tool lets you save a set of partitions to disk.



For example:

- 1 Open the Wavelet 1-D Multisignal Analysis main tool and load the example analysis using **File > Example > Ex 21: Thinker (rows)**.
- 2 Click the **Clustering** button. The Wavelet 1-D Multisignal Analysis Clustering window appears.
- 3 Click the **Compute Clusters** button, and then save the current partition using menu option **File > Partitions > Save Current Partition**. A dialog box appears that lets you specify the type of data to save.

- 4 Click the **Save Curr.** button.



- 5 Another dialog box appears that lets you specify a folder and filename for storing the partition data. Type the name `curPART`.
- 6 After saving the partition data to the file `curPART.mat`, load the variables into your workspace:

```
load curPART
whos
```

Name	Size	Bytes	Class
tab_IdxCLU	192x1	1536	double

- 7 You can modify the array `tab_IdxCLU` in the workspace, and save the partition data in a file. For example to create two new partitions with four and two clusters, type the following lines:

```
tab_IdxCLU(:,2) = rem((1:192)',4) + 1;
tab_IdxCLU(:,3) = double((1:192) '>96) + 1;
save newpart tab_IdxCLU
```

Now you can use three partitions for the example Ex 21: Thinker (rows). Then, you can load the partitions stored in the file `newPART.mat` in the Wavelet 1-D Multisignal Analysis main tool and clustering tool.

Note A partition is a column vector of integers. The values must vary from 1 to `NbClusters` (`NbClusters > 1`), and each cluster must contain at least one element. The number of rows must be equal to the number of signals.

References

- [1] Denoeud, L., Garreta, H., and A. Guénoche. "Comparison of Distance Indices Between Partitions." In *International Symposium on Applied Stochastic Models and Data Analysis*, 432-440. Brest, France: École Nationale des Télécommunications de Bretagne, 2005.

2-D Discrete Wavelet Analysis

This section takes you through the features of 2-D 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 2-D 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”.

Analysis-Decomposition Functions

Function Name	Purpose
<code>dwt2</code>	Single-level decomposition
<code>wavedec2</code>	Decomposition
<code>wmaxlev</code>	Maximum wavelet decomposition level

Synthesis-Reconstruction Functions

Function Name	Purpose
<code>idwt2</code>	Single-level reconstruction
<code>waverec2</code>	Full reconstruction
<code>wrcoef2</code>	Selective reconstruction
<code>upcoef2</code>	Single reconstruction

Decomposition Structure Utilities

Function Name	Purpose
<code>detcoef2</code>	Extraction of detail coefficients
<code>appcoef2</code>	Extraction of approximation coefficients
<code>upwlev2</code>	Recomposition of decomposition structure

Denosing and Compression

Function Name	Purpose
ddencmp	Provide default values for denosing and compression
wbmpen	Penalized threshold for wavelet 1-D or 2-D denosing
wdcbm2	Thresholds for wavelet 2-D using Birgé-Massart strategy
wdencmp	Wavelet denosing 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

2-D Analysis – Command Line

In this example we'll show how you can use 2-D wavelet analysis to compress an image efficiently without sacrificing its clarity.

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

- 1 Load an image.

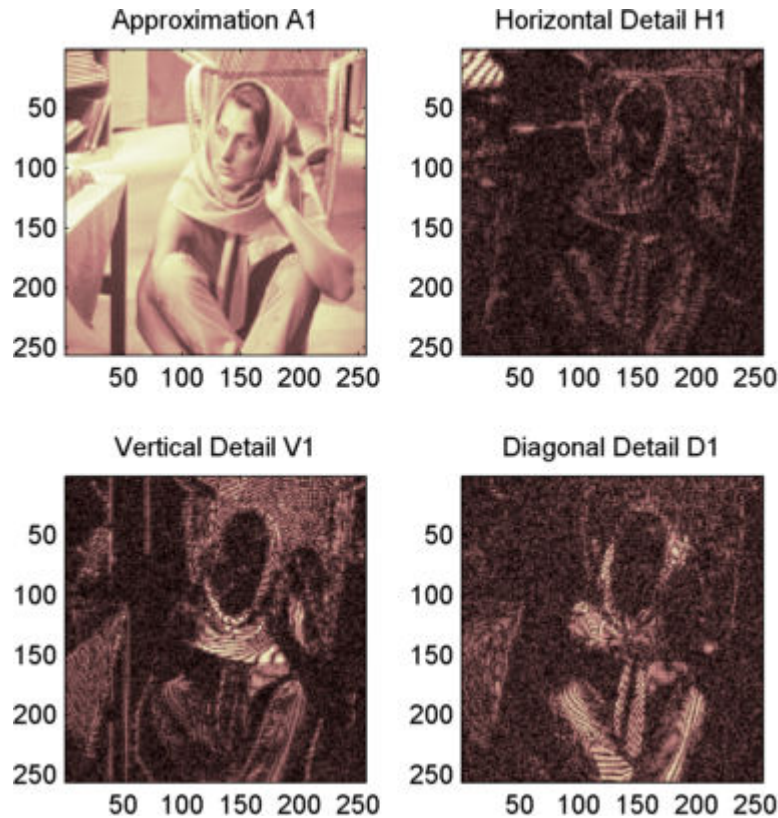
From the MATLAB prompt, type

```
load wbarb;
whos
```

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

2 Display the image. Type

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



3 Convert an indexed image to a grayscale image.

If the colormap is smooth, the wavelet transform can be directly applied to the indexed image; otherwise the indexed image should be converted to grayscale format. For more information, see "Wavelets: Working with Images".

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

4 Perform a single-level wavelet decomposition.

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

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

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

5 Construct and display approximations and details from the coefficients.

To construct the level-one approximation and details (A_1 , H_1 , V_1 , and D_1) from the coefficients `cA1`, `cH1`, `cV1`, and `cD1`, type

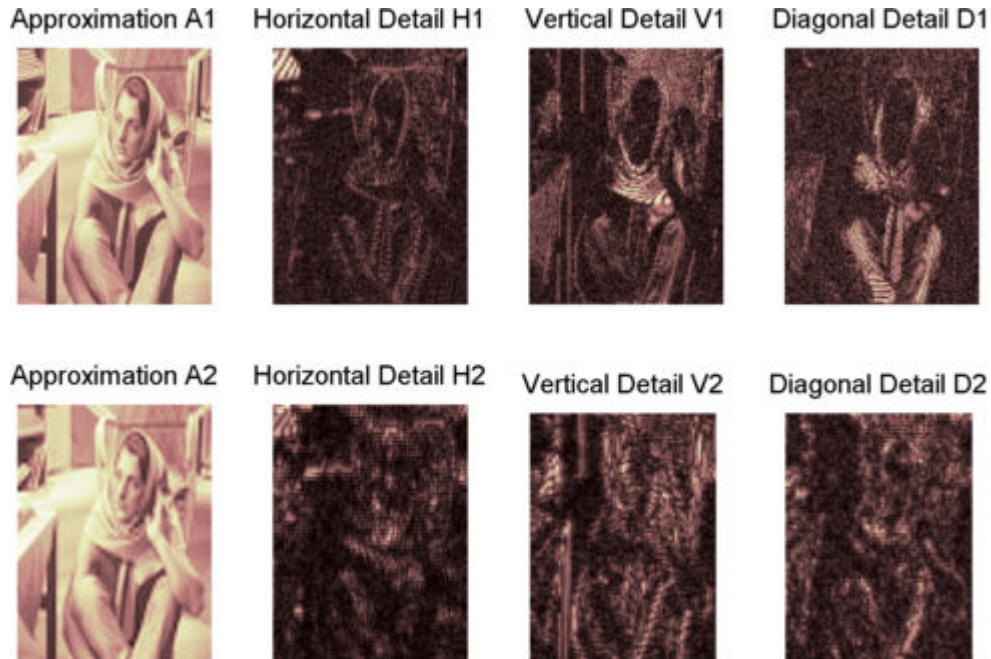
```
A1 = upcoef2('a',cA1,'bior3.7',1);
H1 = upcoef2('h',cH1,'bior3.7',1);
V1 = upcoef2('v',cV1,'bior3.7',1);
D1 = upcoef2('d',cD1,'bior3.7',1);
```

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 3-212.

To display the results of the level 2 decomposition, type

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

- 11** Reconstruct the original image from the multilevel decomposition.

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

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

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

- 12** Compress the image and display it.

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

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

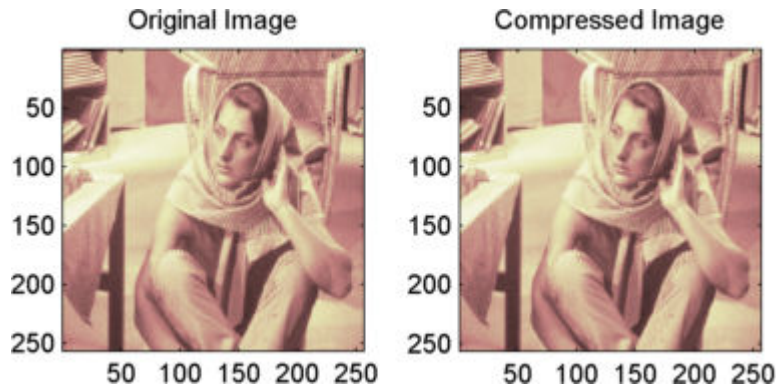
Note that we pass in to `wdencmp` the results of the decomposition (C and S) we calculated in step 7. We also specify the `bior3.7` wavelets, because we used this wavelet to perform the original analysis. Finally, we specify the global thresholding option `'gbl'`. See `ddencmp` and `wdencmp` reference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

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


```

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

```



```

PERF0
  PERF0 =
    49.8076

```

```

PERFL2
  PERFL2 =
    99.9817

```

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

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

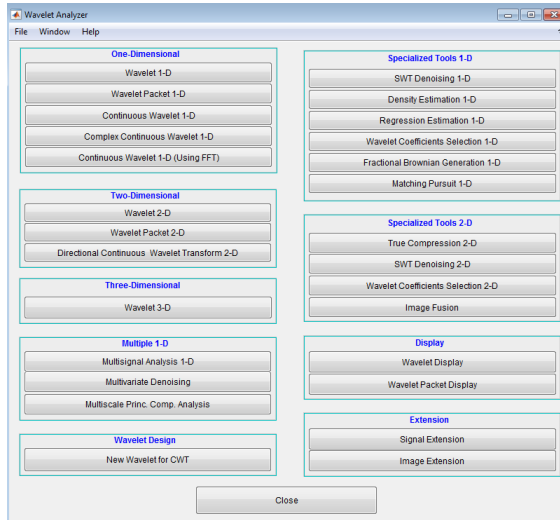
2-D Wavelet Analysis Using the Wavelet Analyzer App

In this section we explore the same image as in the previous section, but we use the Wavelet Analyzer app to analyze the image.

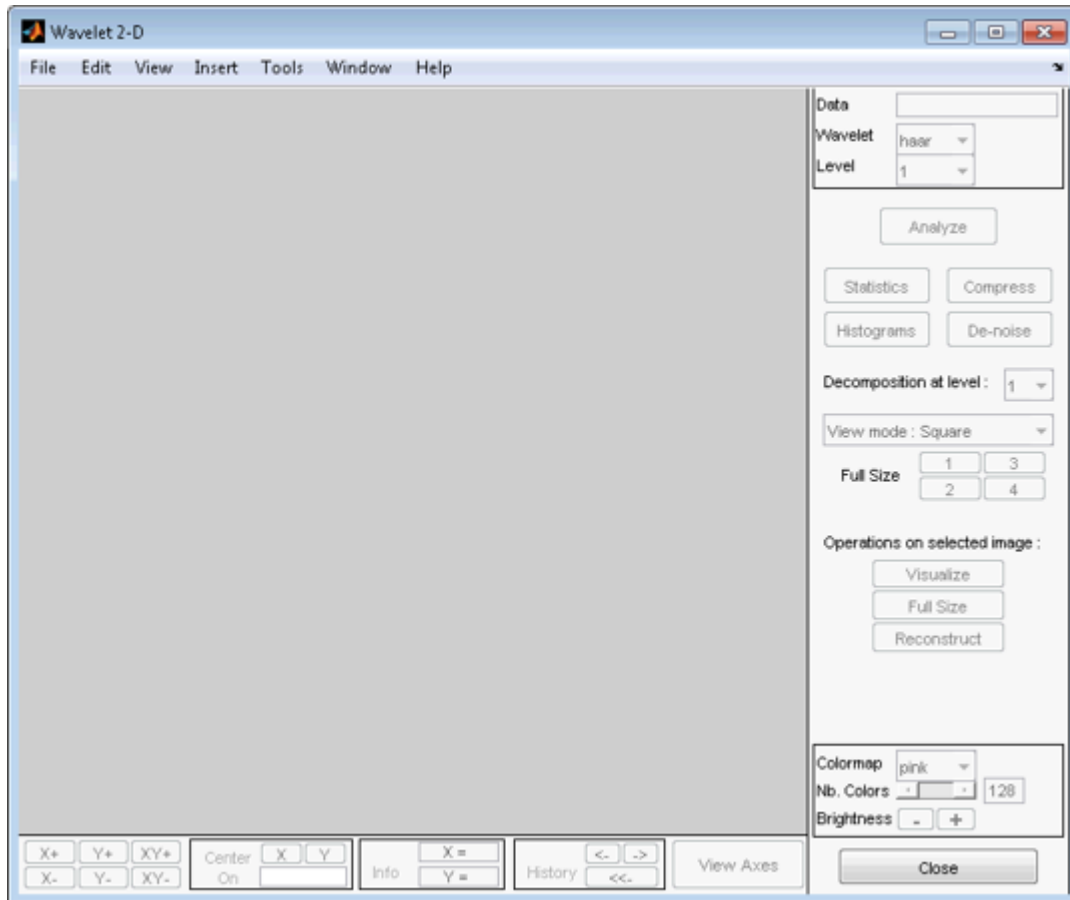
- 1 Start the 2-D Wavelet Analysis Tool.

From the MATLAB prompt, type `waveletAnalyzer`.

The **Wavelet Tool Main Menu** appears.



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



- 2 Load an image.

At the MATLAB command prompt, type

load wbarb

In the **Wavelet 2-D** tool, select **File > Import from Workspace > Import Image**. When the **Import from Workspace** dialog box appears, select the X variable. Click **OK** to import the image.

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

- 3 Analyze the image.

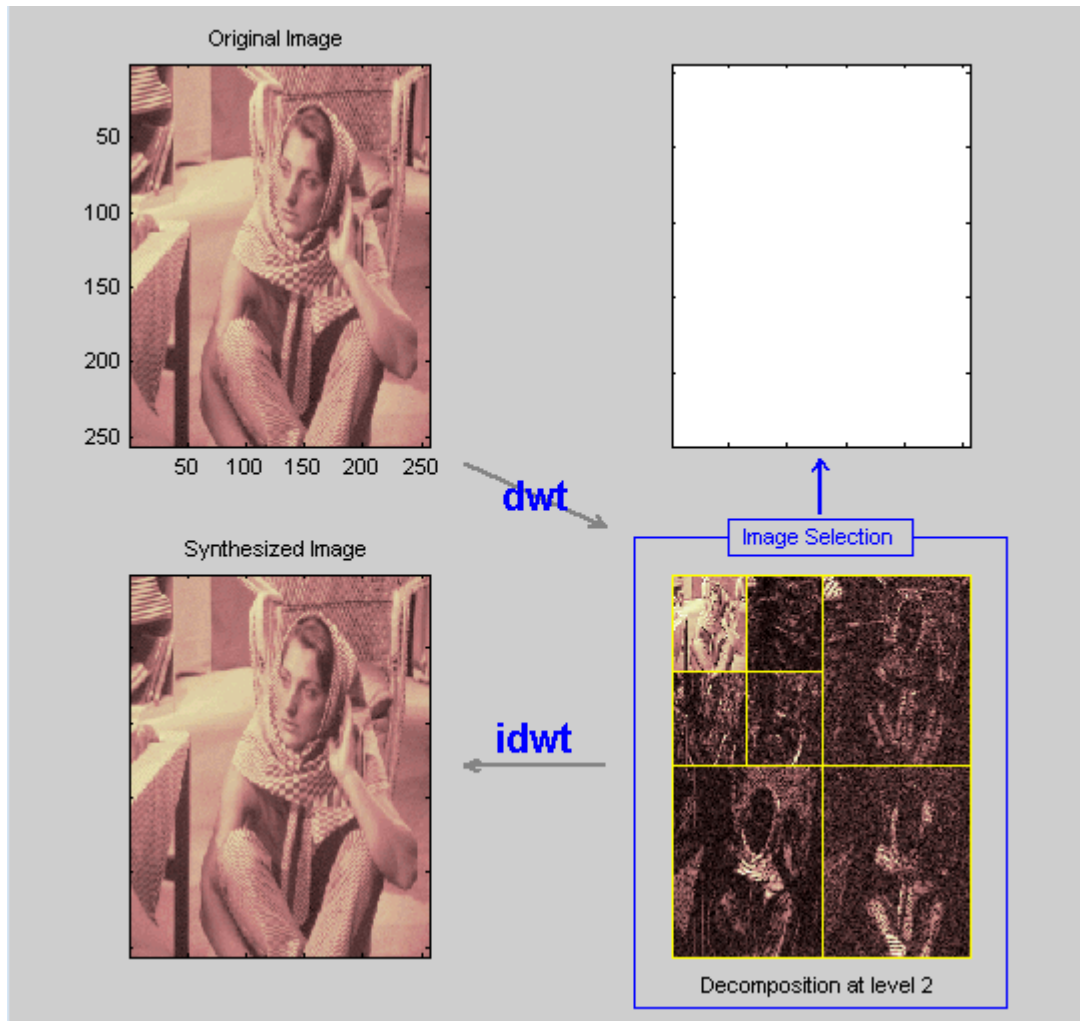
Using the **Wavelet** and **Level** menus located to the upper right, determine the wavelet family, the wavelet type, and the number of levels to be used for the analysis.

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

Wavelet	bior	3.7
Level	2	

Analyze

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 blue 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 blue border disappears.

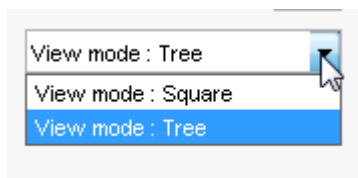


Click the **Visualize** button.

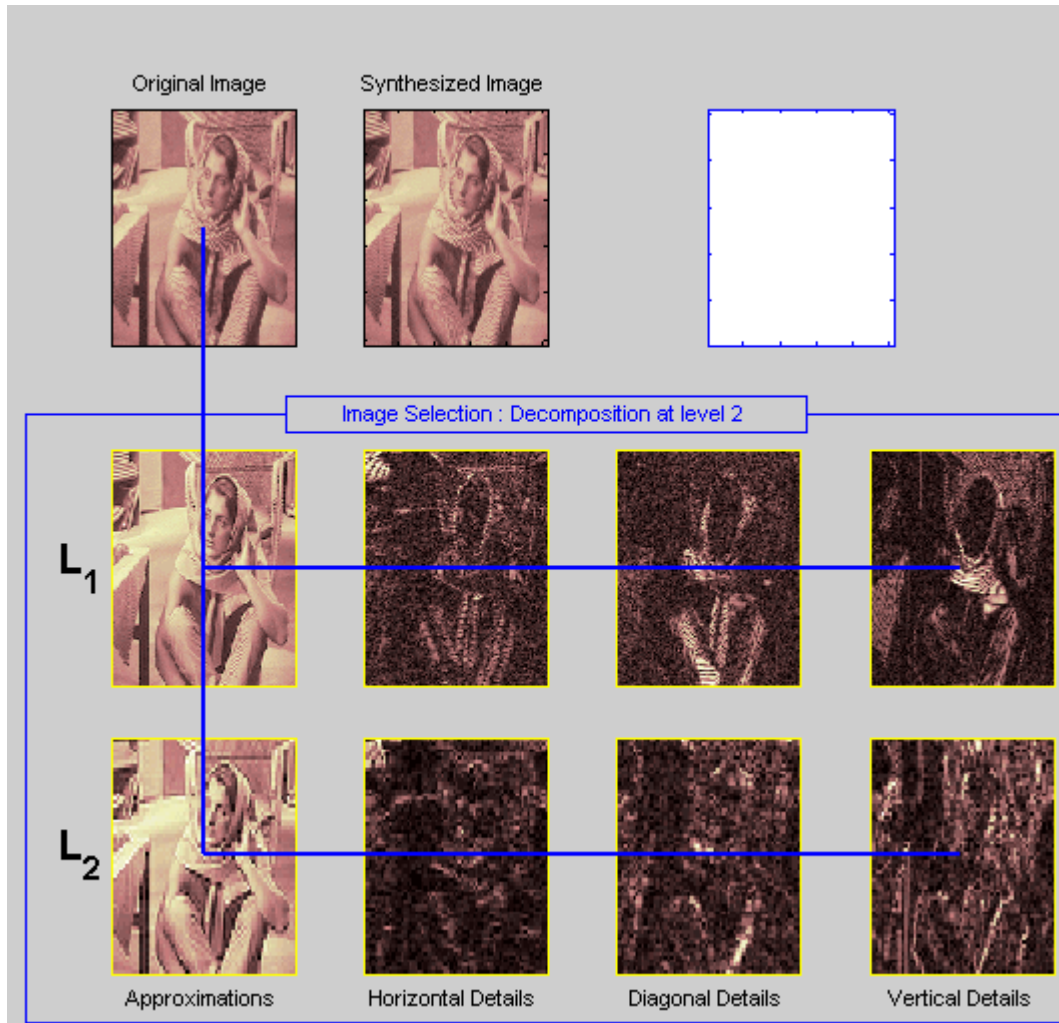
The selected image is displayed in the visualization area. You are seeing the raw, unreconstructed 2-D wavelet coefficients. Using the other buttons, you can display the reconstructed version of the selected image component, or you can view the selected component at full screen resolution.

Using Tree Mode Features

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



Your display changes to reveal the following.

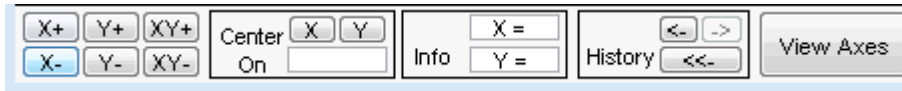


This is the same information shown in square mode, with in addition all the approximation coefficients, but arranged to emphasize the tree structure of the decomposition. The various buttons and menus work just the same as they do in square mode.

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.

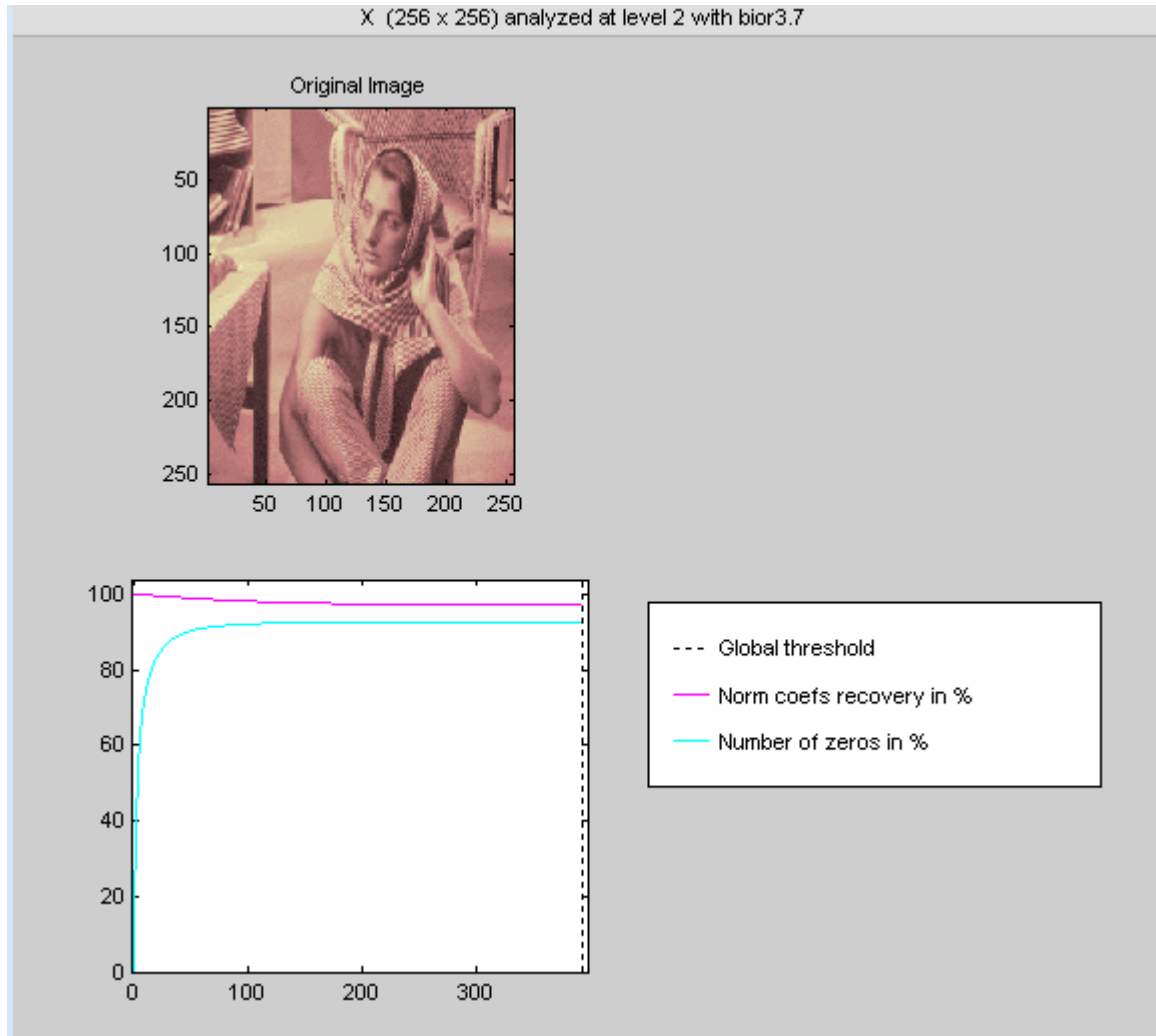


The **Wavelet 2-D** tool enlarges the displayed images.

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.

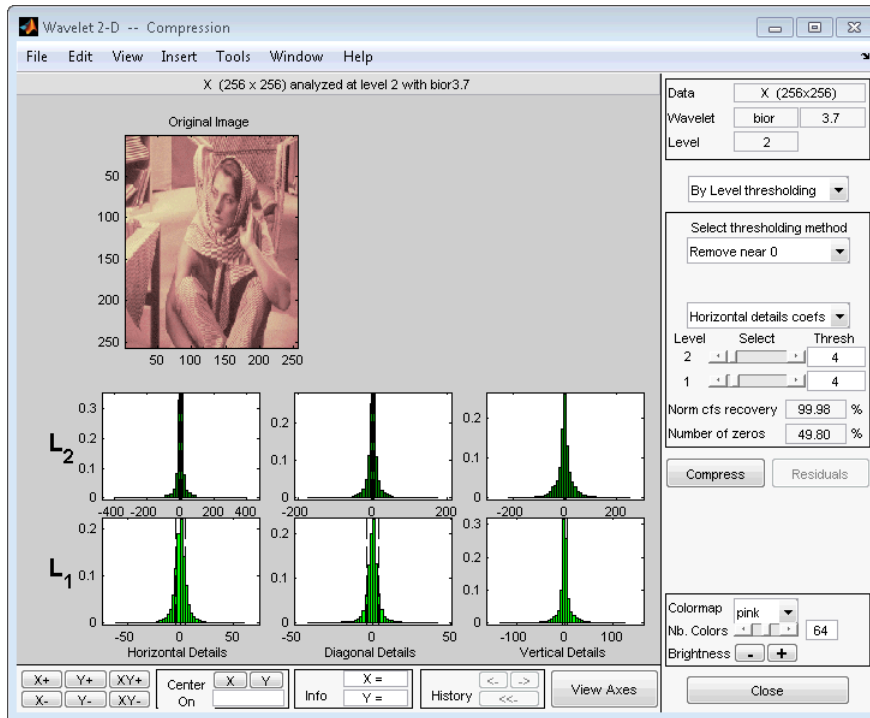


The tool automatically selects thresholding levels to provide a good initial balance between retaining the image's energy while minimizing the number of coefficients needed to represent the image.

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

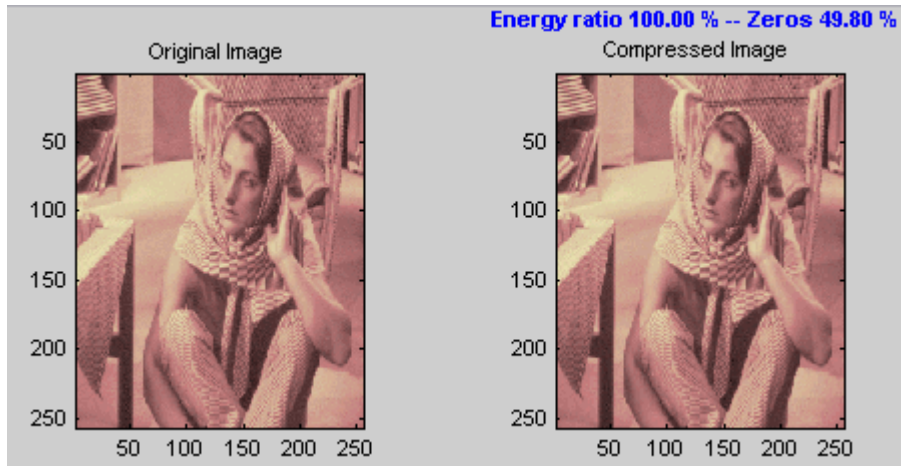
For this example, select the **By Level thresholding** option and select the **Remove near 0** method from the **Select thresholding method** menu.

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 dashed 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 Denoising** tool.

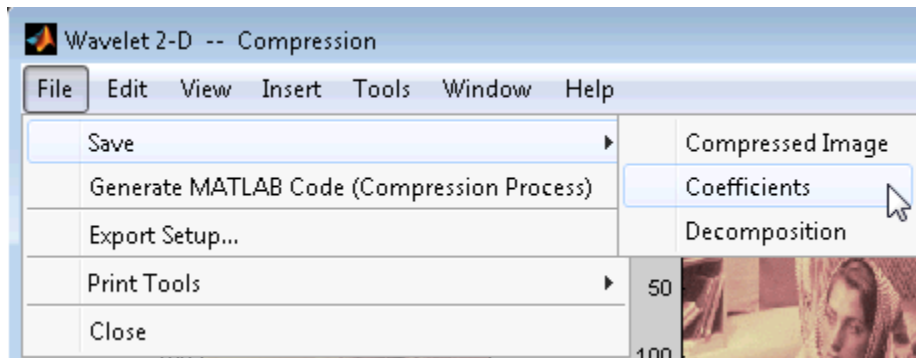
Note The statistics displayed in the above figure are related to the displayed image but not to the original one. Usually this information is the same, but in some cases, edge effects may cause the original image to be cropped slightly. To see the exact statistics, use the command line functions to get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

Importing and Exporting Information from the Wavelet Analyzer App

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 > Indexed Images > at level 3, with sym4 → Detail Durer

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

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

To load the image into your workspace, type

```
load symage
whos
```

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

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

Saving Discrete Wavelet Transform Coefficients

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

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

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

Consider the example analysis:

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

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

```
load cfsdurer
whos
```

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

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

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

Saving Decompositions

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

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

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

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

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

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

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

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

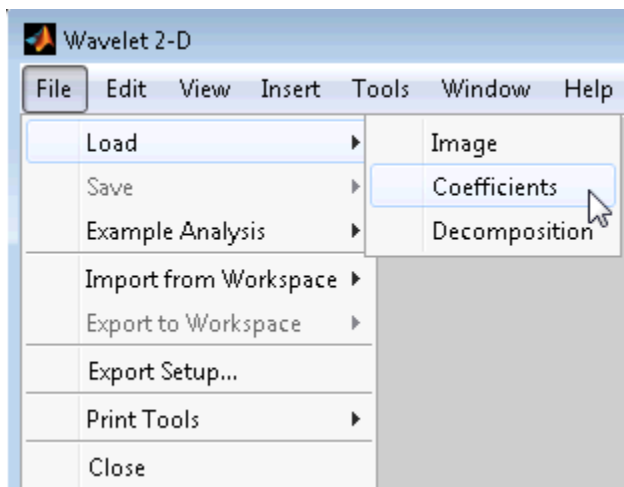
Variables `coefs` and `sizes` contain the wavelet decomposition structure. Other variables contain the wavelet name, the colormap, and the filename containing the data. Variable `valTHR` is empty since the synthesized image is the same as the original one.

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

Loading Information into the Wavelet 2-D Tool

You can load images, coefficients, or decompositions into the Wavelet Analyzer app. The information you load may have been previously exported from the Wavelet Analyzer app, 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 2-D 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.

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

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

```
save myfile coefs sizes
```

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

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

Loading Decompositions

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

The MAT-file contains these variables.

Variable	Status	Description
<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	Character vector 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	Character vector specifying name of decomposition

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

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

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

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

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

2-D Stationary Wavelet Transform

This section takes you through the features of 2-D discrete stationary wavelet analysis using the Wavelet Toolbox software.

Analysis-Decomposition Function

Function Name	Purpose
swt2	Decomposition

Synthesis-Reconstruction Function

Function Name	Purpose
iswt2	Reconstruction

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

In this section, you'll learn to

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

2-D Analysis Using the Command Line

In this example, we'll show how you can use 2-D stationary wavelet analysis to denoise an image.

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

This example involves a image containing noise.

1 Load an image.

From the MATLAB prompt, type

```
load noiswom
whos
```

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

For the SWT, if a decomposition at level k is needed, 2^k must divide evenly into $\text{size}(X, 1)$ and $\text{size}(X, 2)$. If your original image is not of correct size, you can use the **Image Extension Wavelet Analyzer** app 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

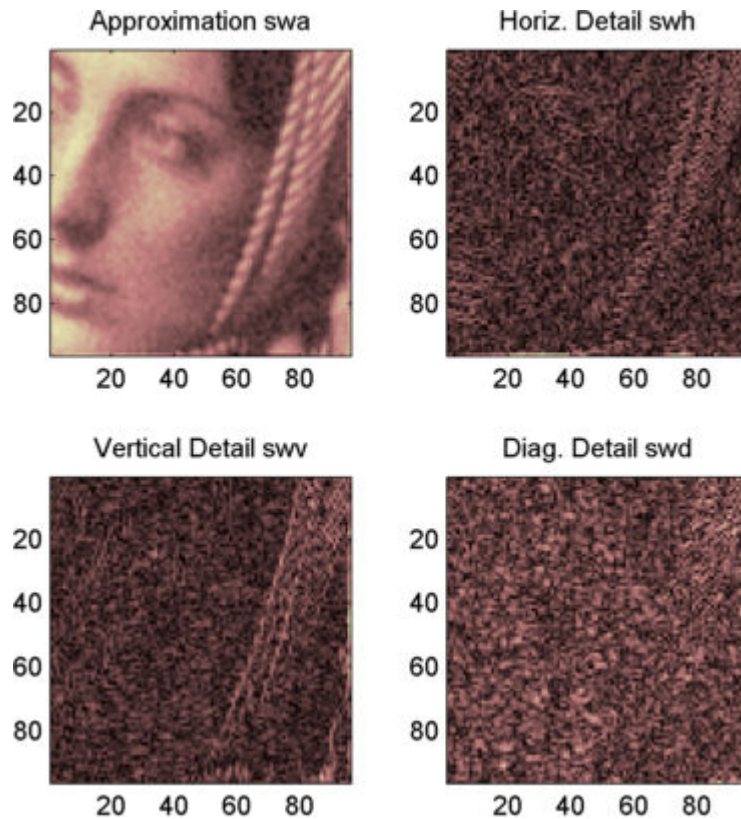
```
whos
```

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(svw,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));
```

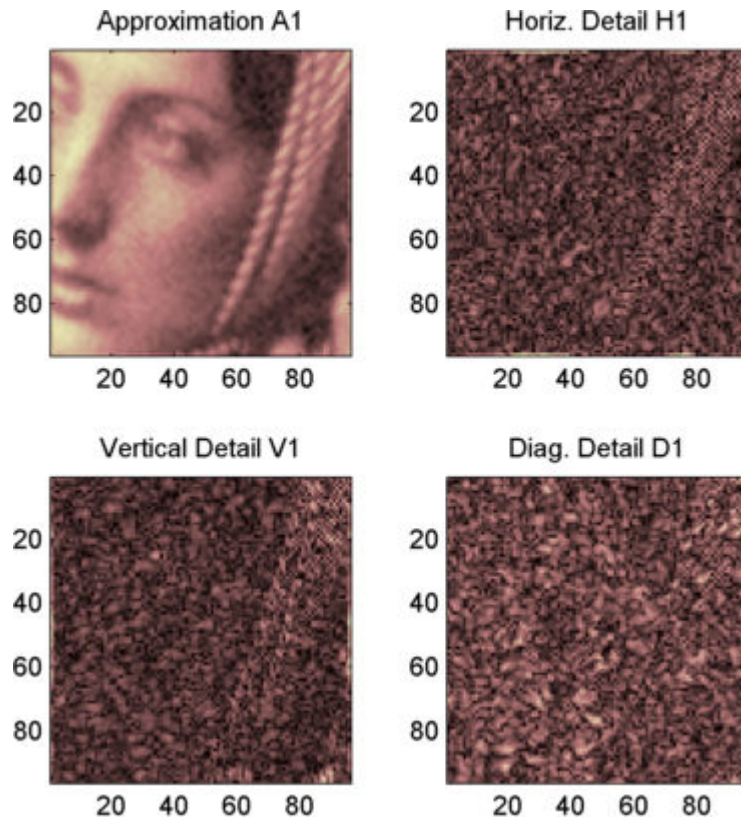
```
title('Horiz. Detail H1')
```

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

```

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

```

To display both the original and denoised images, type

```

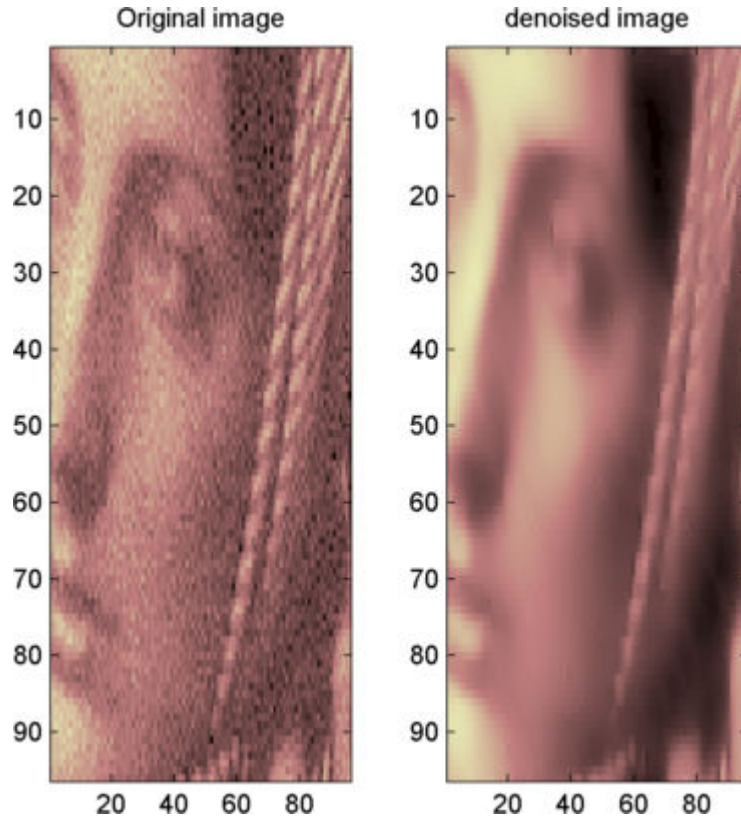
colormap(map)
subplot(1,2,1), image(wcodemat(X,192));

```

```

title('Original image')
subplot(1,2,2), image(wcodemat(clean,192));
title('denoised image')

```



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

```

lev= 4;
swc = swt2(X,lev,'db1');
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.

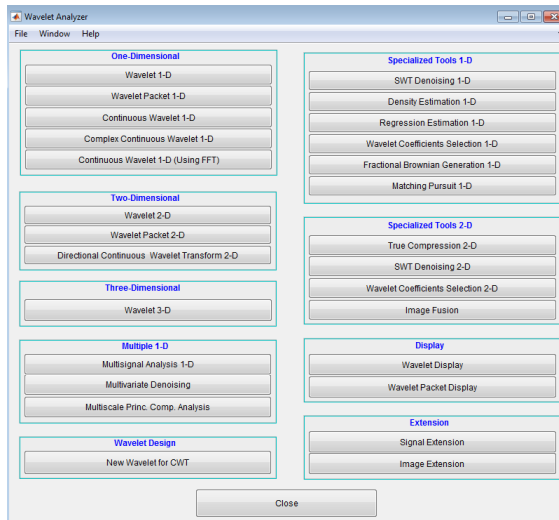
Interactive 2-D Stationary Wavelet Transform Denoising

In this section, we explore a strategy for denoising images based on the 2-D stationary wavelet analysis using the Wavelet Analyzer app. The basic idea is to average many slightly different discrete wavelet analyses.

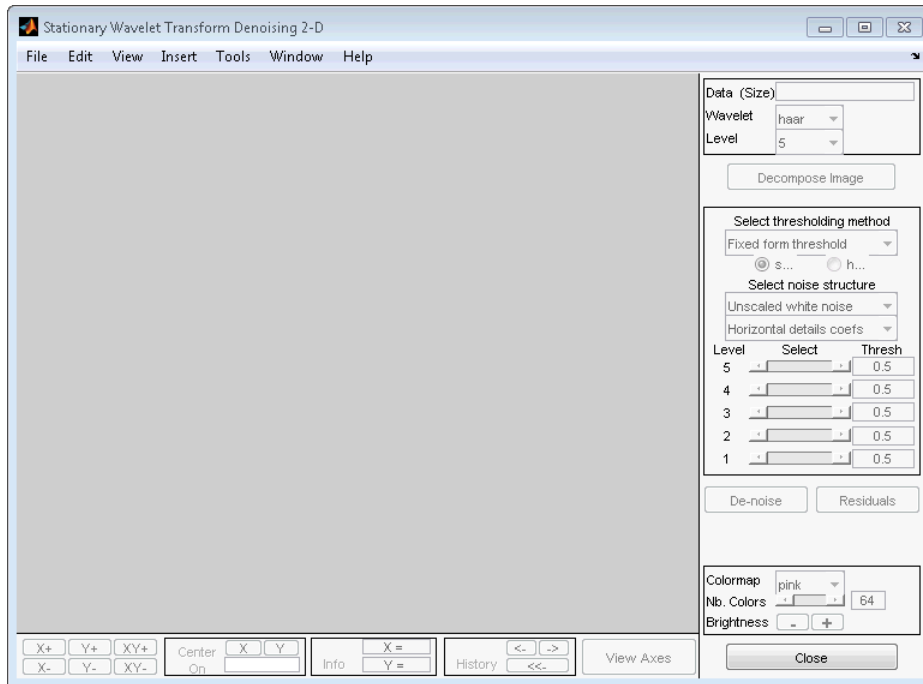
- 1 Start the Stationary Wavelet Transform Denoising 2-D Tool.

From the MATLAB prompt, type `waveletAnalyzer`.

The **Wavelet Analyzer** appears:



Click the **SWT Denoising 2-D** menu item.



2 Load data.

At the MATLAB command prompt, type

```
load noiswom
```

In the **SWT Denoising 2-D** tool, select **File > Import Image from Workspace**. When the **Import from Workspace** dialog box appears, select the X variable. Click **OK** to import the image.

3 Perform a Stationary Wavelet Decomposition.

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

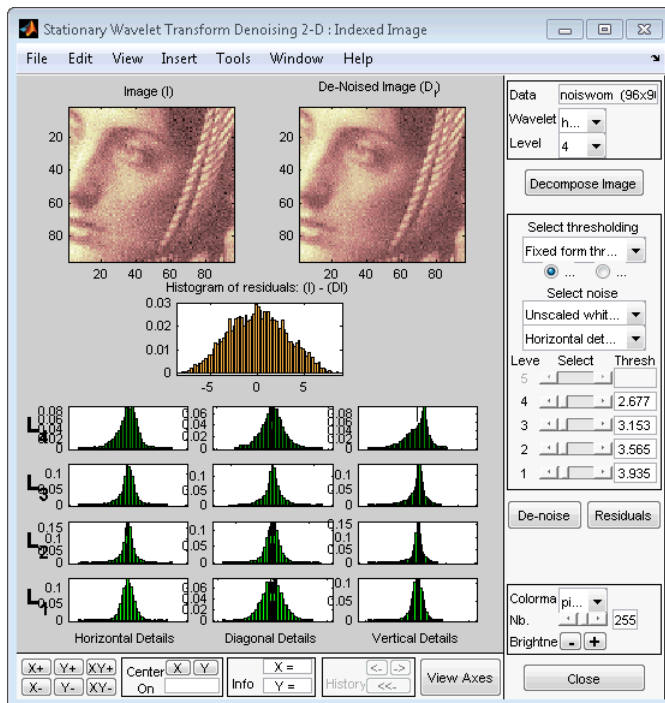
The tool displays the histograms of the stationary wavelet detail coefficients of the image on the left of the window. These histograms are organized as follows:

- From the bottom for level 1 to the top for level 4

- On the left horizontal coefficients, in the middle diagonal coefficients, and on the right vertical coefficients

4 Denoise the image using the Stationary Wavelet Transform.

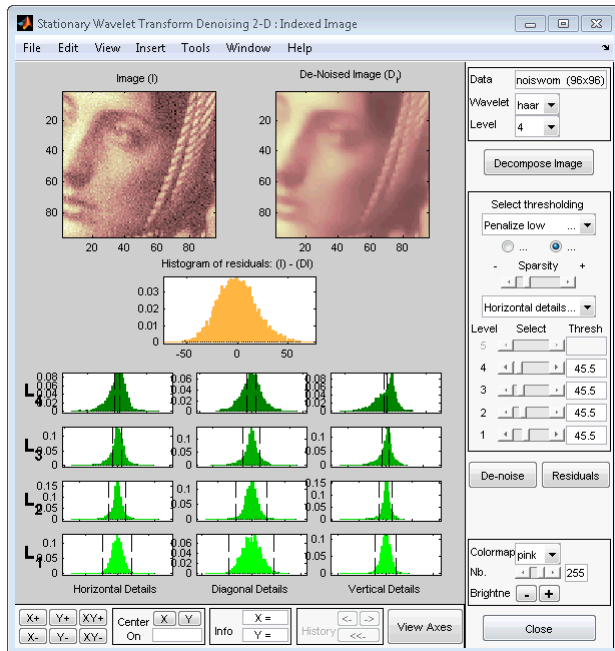
While a number of options are available for fine-tuning the denoising algorithm, we'll accept the defaults of fixed form soft thresholding and unscaled white noise. The sliders located to the right of the window control the level dependent thresholds indicated by the dashed lines running vertically through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



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

5 Selecting a thresholding method.

From the **Select thresholding method** menu, choose the **Penalize low** item. The associated default for the thresholding mode is automatically set to **hard**; accept it. Use the **Sparsity** slider to adjust the threshold value close to 45.5, and then click the **denoise** button.



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

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

Importing and Exporting Information from the Wavelet Analyzer App

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

To save the denoised image from the present denoising process, use the menu **File > Save denoised Image**. A dialog box appears that lets you specify a folder and filename

for storing the image. Type the name `dnoiswom`. After saving the image data to the file `dnoiswom.mat`, load the variables into your workspace:

```
load dnoiswom  
whos
```

Name	Size	Bytes	Class
X	96x96	73728	double array
map	255x3	6120	double array
valTHR	3x4	96	double array
wname	1x4	8	char array

The denoised image is `X` and `map` is the colormap. In addition, the parameters of the denoising 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).

Shearlet Systems

A shearlet system enables you to create directionally sensitive sparse representations of images with anisotropic features. Shearlets are used in image processing applications including denoising, compression, restoration, and feature extraction. Shearlets are also used in statistical learning to address problems of image classification, inverse scattering problems such as tomography, and data separation. You can find additional applications at ShearLab [5].

A strength of wavelet analysis for 1-D signals is its ability to efficiently represent smooth functions that have pointwise discontinuities. However, wavelets do not represent curved singularities, such as the edge of a disk in an image, as sparsely as they do pointwise discontinuities. Geometric multiscale analysis is an attempt to design systems capable of efficiently representing curved singularities in higher dimensional data. In addition to shearlets, other geometric multiscale systems include curvelets, contourlets, and bandlets.

Guo, Kutyniok, and Labate [1] pioneered the development of the theory of shearlets. They also developed efficient algorithms for shearlet transforms [4], as have Häuser and Steidl [6]. ShearLab [5] provides an extensive set of algorithms for processing two- and three-dimensional data using shearlets.

Like wavelets, a comprehensive theory relates the continuous shearlet transform with the discrete transform. Also, a multiresolution analysis framework exists for shearlets. As the name suggests, shearlets have the noteworthy feature of using shears, not rotations, to control directional sensitivity. This characteristic allows you to create a shearlet system from a single or finite set of generating functions. Other reasons for the success of shearlets include:

- Shearlets provide optimally sparse approximations of anisotropic features of multivariate data.
- Both compactly supported and bandlimited shearlets exist.
- Shearlet transforms have efficient algorithmic implementations.

Shearlets

Similar to wavelets, shearlets do not have a unique system. Dilation, shearing, and translation operations generate the shearlets. A dilation can be expressed as a matrix,

$A_a = \begin{pmatrix} a^{1/2} & 0 \\ 0 & a^{-1/2} \end{pmatrix}$ where $a \in \mathbb{R}^+$. A shear can be expressed as $S_s = \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix}$ where $s \in \mathbb{R}$. The variable s parameterizes orientations.

If the function $\psi \in L^2(\mathbb{R}^2)$ satisfies certain (admissibility) conditions, then the set of functions

$$SH\{\psi\} = \{\psi_{a,s,t} = a^{3/4}\psi(S_s A_a(\cdot - t))\}$$

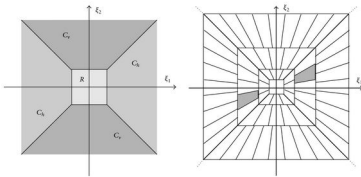
is a *continuous shearlet system* where a and s are defined as noted earlier, and $t \in \mathbb{R}^2$.

If you discretize the dilation, shearing, and translation parameters appropriately, you obtain a *discrete shearlet system*:

$$SH(\psi) = \{\psi_{j,k,m} = 2^{3/4j}\psi(S_k A_{2^j} \cdot - m) : j, k \in \mathbb{Z}, m \in \mathbb{Z}^2\}.$$

The function `shearletSystem` creates a cone-adapted bandlimited shearlet system. The implementation of the `shearletSystem` function follows the approach described in Häuser and Steidl [6]. The shearlet system is an example of a frame, which you can normalize to create a Parseval frame. The discrete shearlet transform of a function $f \in L^2(\mathbb{R}^2)$ is the inner product of f with all the shearlets in the discrete shearlet system $\langle f, \psi_{j,k,m} \rangle$ where $(j, k, m) \in \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}^2$. You use `shearlet2` to take the discrete shearlet transform of an image. For additional information, see the “References” on page 3-229.

The following figure shows how a cone-adapted shearlet system partitions the 2-D frequency plane. The image on the left shows the partition of a cone-adapted real-valued shearlet system with one scale. The R region in the center is the lowpass part of the system. In addition, the image includes a horizontal cone shearlet (symmetric in frequency because it is real valued) and a vertical cone shearlet. The image on the right depicts a system with three scales. The fan-like pattern gives a shearlet system its directional sensitivity. Note that the number of shearing factors increases as the frequency support of the shearlet increases. As the support in the frequency domain increases, the support in the spatial domain decreases.



The spectra of the real-valued shearlets are the same over the positive and negative ξ_1 , ξ_2 supports. Shearlets in complex-valued shearlet systems partition individually, not in pairs.

Transform Type

Shearlets are either real valued or complex valued in the spatial domain. You specify the transform type when you use `shearletSystem` to create the system. Real-valued shearlets have two-sided frequency spectra. The Fourier transforms of the complex-valued shearlets have support on only one half of the 2-D frequency space. The Fourier transforms of both types of shearlets are real valued.

References

- [1] Guo, K., G. Kutyniok, and D. Labate. "Sparse multidimensional representations using anisotropic dilation and shear operators." *Wavelets and Splines: Athens 2005* (G. Chen, and M.-J. Chen, eds.), 189-201. Brentwood, TN: Nashboro Press, 2006.
- [2] Guo, K., and D. Labate. "Optimally Sparse Multidimensional Representation Using Shearlets." *SIAM Journal on Mathematical Analysis*. Vol. 39, Number 1, 2007, pp. 298-318.
- [3] Kutyniok, G., and W-Q Lim. "Compactly supported shearlets are optimally sparse." *Journal of Approximation Theory*. Vol. 163, Number 11, 2011, pp. 1564-1589.
- [4] *Shearlets: Multiscale Analysis for Multivariate Data* (G. Kutyniok, and D. Labate, eds.). New York: Springer, 2012.
- [5] *ShearLab*. <https://www3.math.tu-berlin.de/numerik/www.shearlab.org/>.
- [6] Häuser, S., and G. Steidl. "Fast Finite Shearlet Transform: a tutorial." arXiv preprint arXiv:1202.1773 (2014).

[7] Rezaeilouyeh, H., A. Mollahosseini, and M. Mahoor. "Microscopic medical image classification framework via deep learning and shearlet transform." *Journal of Medical Imaging*. Vol. 3, Number 4, 044501, 2016. doi:10.1117/1.JMI.3.4.044501.

See Also

isheart2 | shearletSystem | sheart2

3-D Discrete Wavelet Analysis

This section demonstrates the features of three-dimensional discrete wavelet analysis using the Wavelet Toolbox software. The toolbox provides these functions for 3-D data analysis. You use the Wavelet 3-D tool in the **Wavelet Analyzer** app to perform all tasks except the first task.

- Getting information on the command line functions
- Loading 3-D data
- Analyzing a 3-D data
- Selecting and displaying slices
- Creating a slice movie
- Creating true 3-D display
- Importing and exporting information

Performing 3-D Analysis Using the Command Line

The example `wavelet3ddemo` and the documentation of the *Analysis-Decomposition* and *Synthesis-Reconstruction* functions show how you can analyze 3-D arrays efficiently using command line functions dedicated to the 3-D wavelet analysis. For more information, see the function reference pages.

Analysis-Decomposition Functions

Function Name	Purpose
<code>dwt3</code>	Single-level decomposition
<code>wavedec3</code>	Decomposition

Synthesis-Reconstruction Functions

Function Name	Purpose
<code>idwt3</code>	Single-level reconstruction
<code>waverec3</code>	Full reconstruction

Performing 3-D Analysis Using the Wavelet Analyzer App

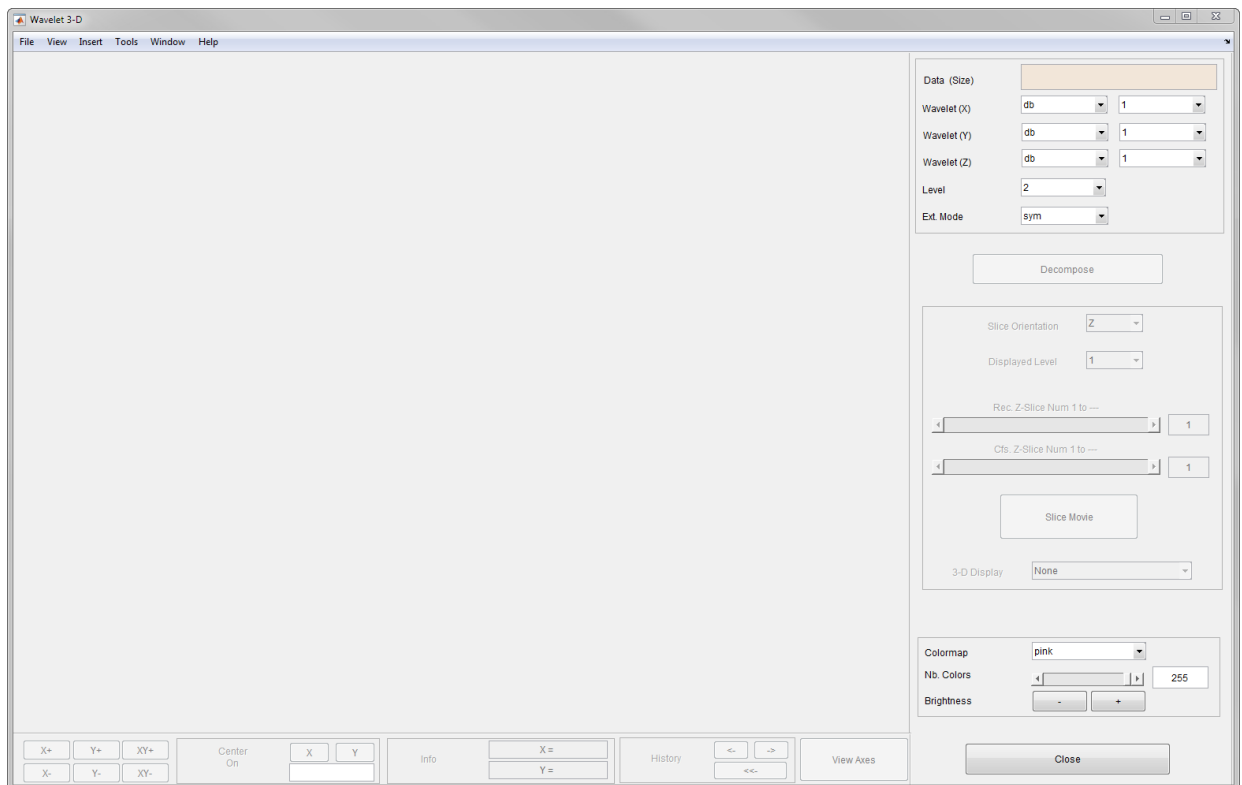
In this section you explore the same 3-D data as in the `wavelet3ddemo` example, but you use the Wavelet Analyzer app.

- 1 Start the 3-D Wavelet Analysis Tool.

From the MATLAB prompt, type `waveletAnalyzer`.

The **Wavelet Analyzer** appears.

Click the **Wavelet 3-D** menu item. The discrete wavelet analysis tool for 3-D data opens.



- 2 Load a 3-D array.

At the MATLAB command prompt, type

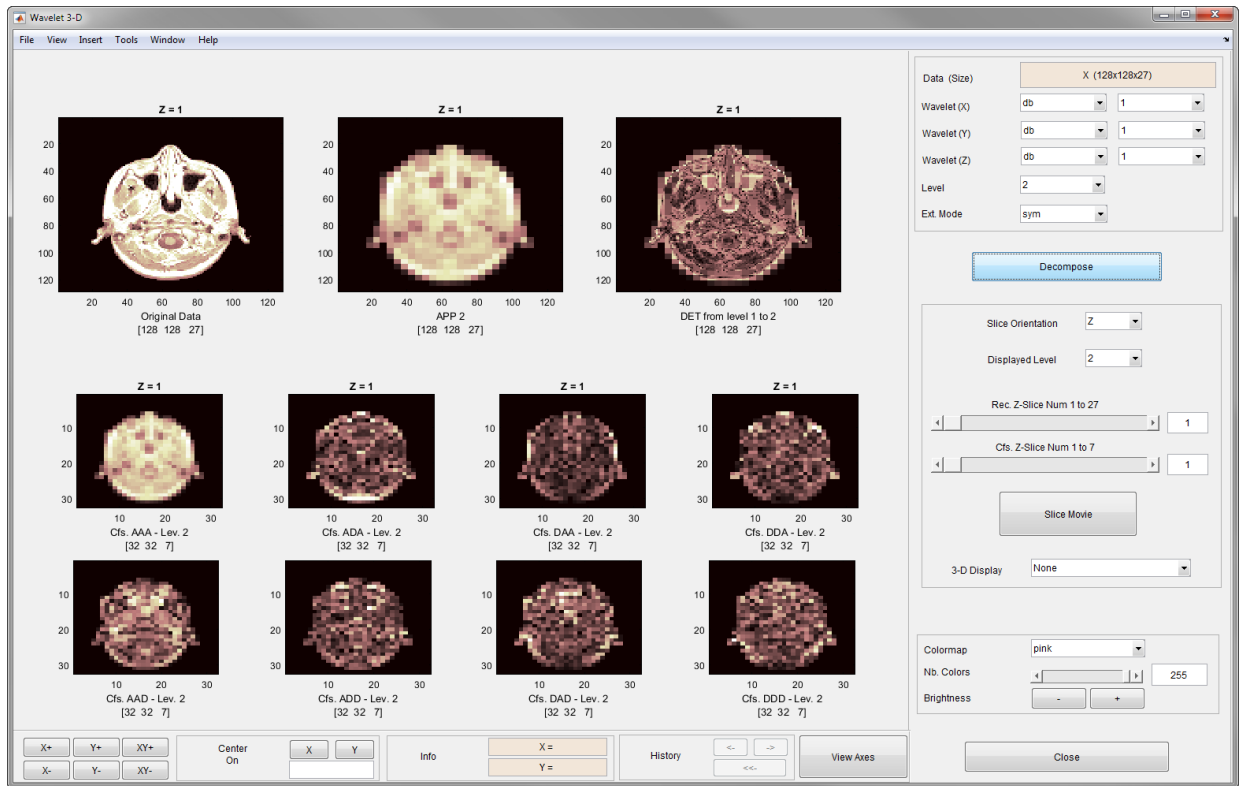
```
load wmri
```

In the **Wavelet 3-D** tool, select **File > Import Data**. When the **Import from Workspace** dialog box appears, select the X variable. Click **OK** to import the 3-D data.

- 3** Analyze the 3-D array. Using the **Wavelet** and **Level** menus located in the upper part of the tool, specify:
 - The wavelet families (one per direction X, Y and Z)
 - The decomposition level and the wavelet extension mode to be used for the analysis

For this analysis, accept the defaults: `db1` wavelet for each direction, decomposition at level 2 and symmetric extension mode (`sym`).

Click **Decompose**. After a pause for computation, the **Wavelet 3-D** tool displays its analysis.

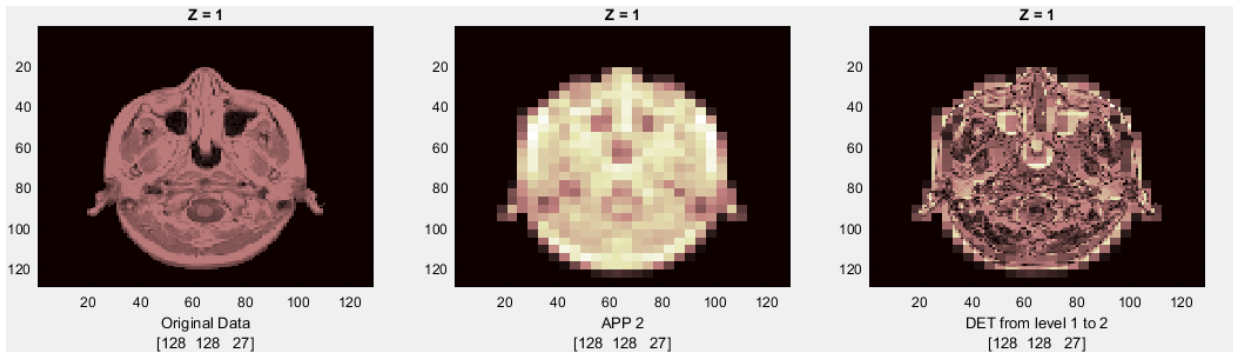


Review the slices of data and wavelet components in the graphical display. These slices are orthogonal to the z -direction as indicated by **Slice Orientation** in the command part of the window. This option lets you choose the desired slice orientation.

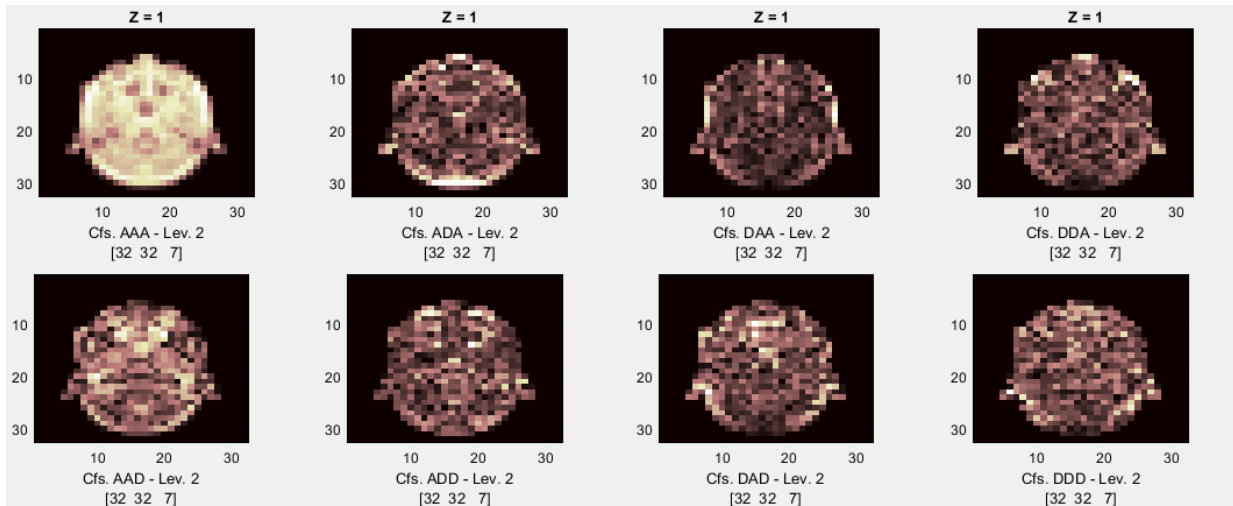
The first row of the graphical display area displays from left to right and for $Z = 1$:

- The original data slice
- The approximation at level 2 slice (low-pass component APP2)
- The slice which is the sum of all the components from level 1 to level 2, different from the lowpass one.

The x -labels of the three axes give you the name and the size of the displayed data.



The next two lines of axes, display the wavelet coefficients at level 2, which is the desired level of the analysis. In the first line, the first graph contains the coefficients of approximation at level 2. The remaining seven axes display the seven types of wavelet coefficients at level 2. These coefficients contain the x-labels of the eight axes and display the name, type and size of the displayed data.

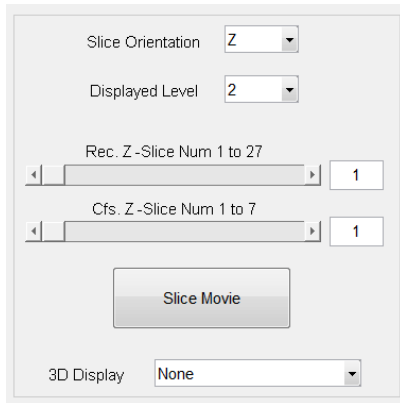


For example, in the third graphic of the bottom line, you can see the Cfs -DAD coefficients at level 2, which correspond to an array of size 32 x 32 x 7. The name of the DAD coefficients group indicates that it is obtained using

- The highpass filter in the x-direction (D) for detail

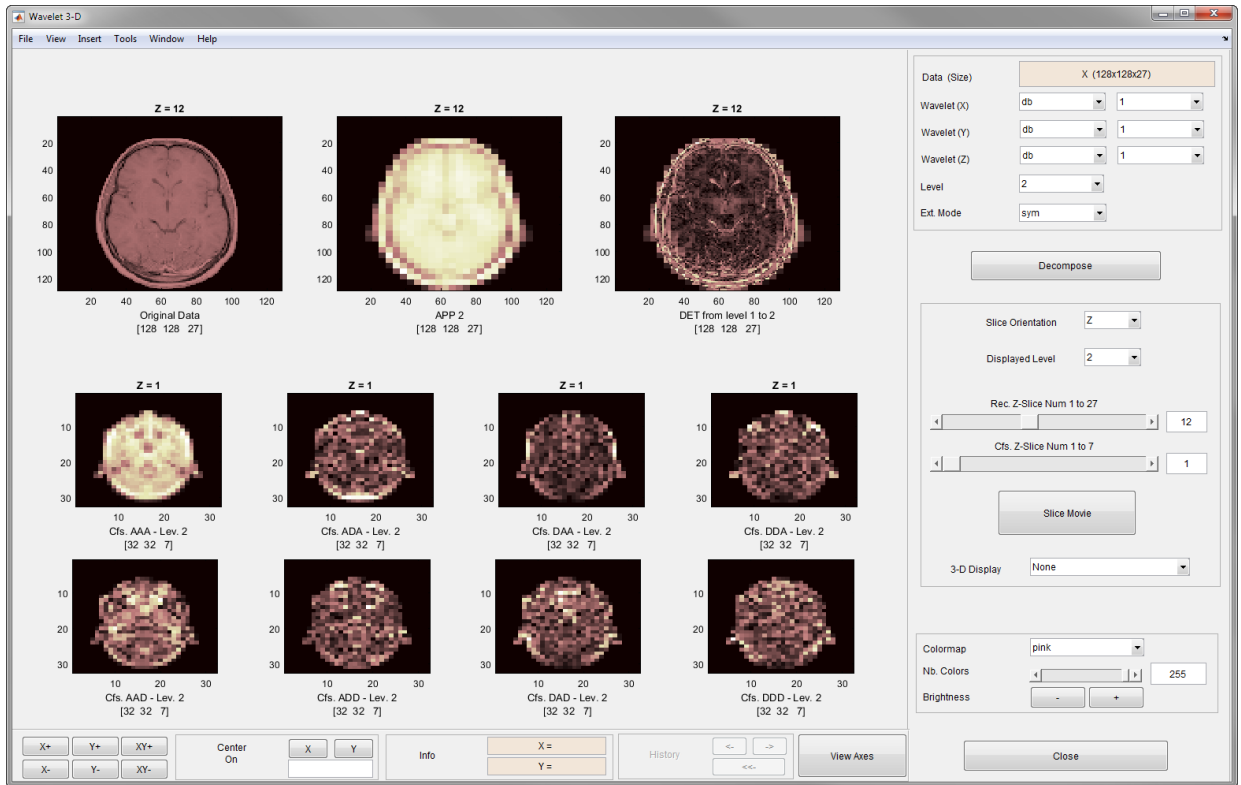
- The lowpass filter in the y-direction (A) for approximation
- The highpass filter in the z-direction (D), leading to the DAD component

You use the **Displayed Level** in the command part of the window to choose the level of the displayed component, from 1 to the decomposition level.



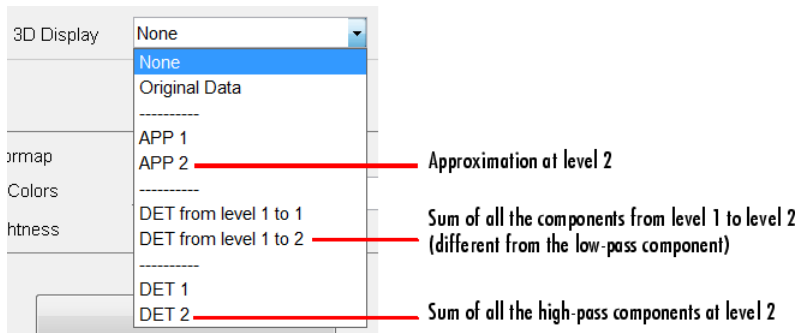
You can modify characteristics of the display using the options in the command part of the window. Each pair of sliders controls part of graphical array, the original and the reconstructed slices with the first pair or the coefficients slices with the second pair. Above each slider you can see the number of slices in the current slice orientation.

Using the slider (or by directly editing the values) of **Rec. Z-Slice**, choose slice number twelve. Similarly, choose slice number two using **Cfs. Z-Slice**.

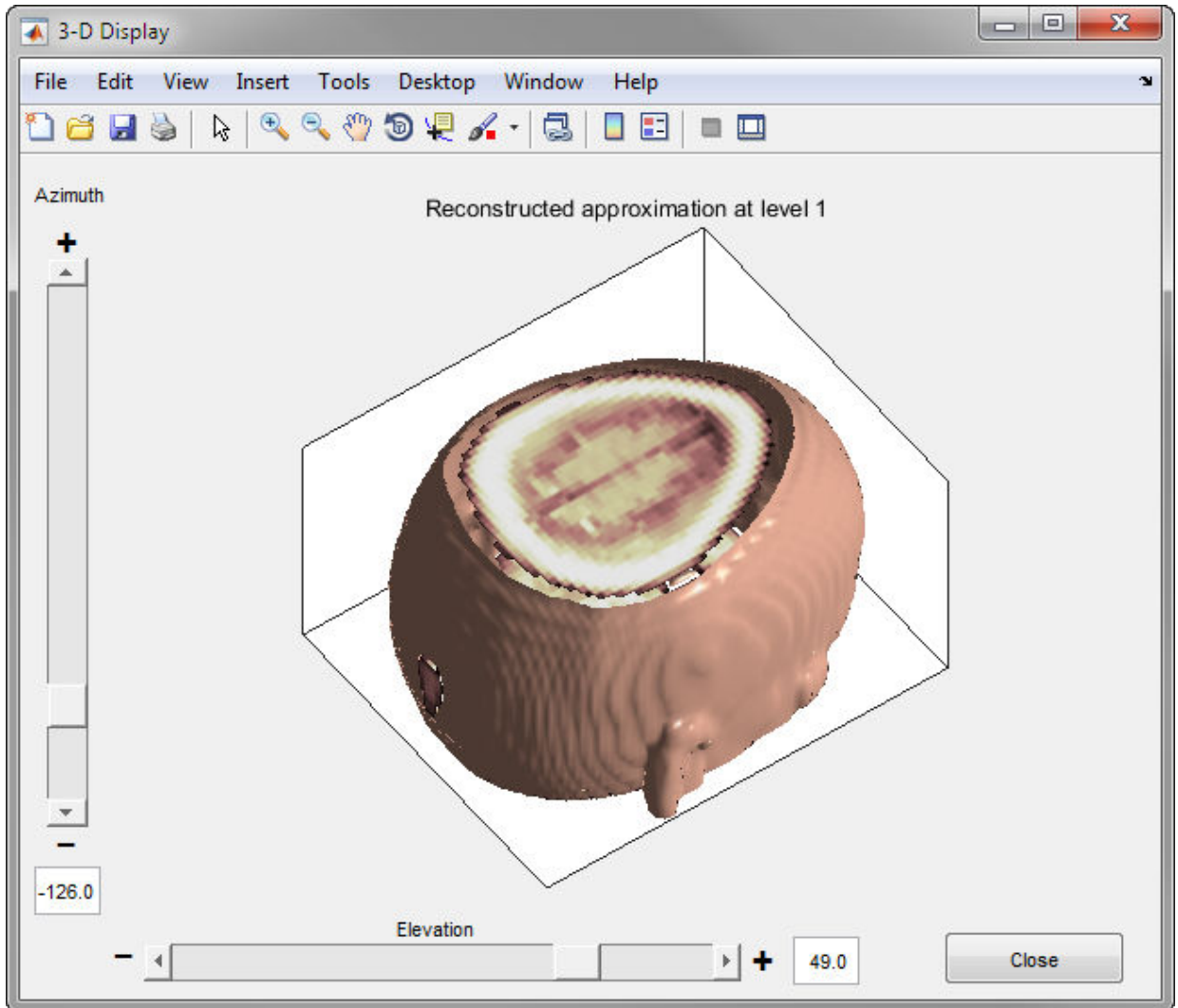


The **Slice Movie** button lets you see a movie of all the slices, first for the reconstructed slices and then for the coefficients slices. In this case, the movie contains 27 reconstructed images and 7 coefficients images.

3D Display lets you examine the original data and the wavelet components in true 3-D mode. Click **3D Display** and select APP1.



A rotated 3-D view of the approximation at level 1 opens in a new window. Use the sliders in the 3-D tool to examine the 3-D data.



Importing and Exporting Information from the Wavelet Analyzer App

You can import information from and export information either to disk or to the workspace using the **Wavelet 3-D** graphical tool.

Loading Information into the Wavelet 3-D Tool

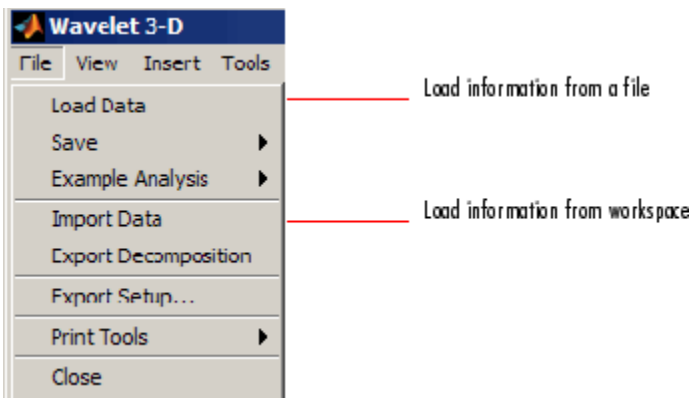
To load 3-D data you have constructed in your MATLAB workspace into the **Wavelet 3-D** tool, save the 3-D data in a MAT-file, using

```
M = magic(8);
X = repmat(M,[1 1 8]);
save magic3d X
whos
```

where M and X are

Name	Size	Bytes	Class
M	8x8	512	double
X	8x8x8	4096	double

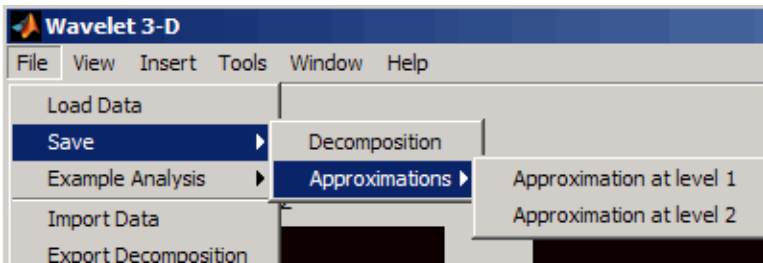
To load this 3-D data into the **Wavelet 3-D** tool, use the menu option **File > Load Data**. You then select the MAT-file to load.



Similarly, you can load information from the workspace using **File > Import Data**. You then select the variable to load.

Saving Information to a File

You can save decompositions and approximations from the **Wavelet 3-D** tool to a file or to the workspace.



Saving Decompositions

The **Wavelet 3-D** tool lets you save the entire set of data from a discrete wavelet analysis to a file. The toolbox creates a MAT-file in the current folder with a name you choose.

- 1 Open the **Wavelet 3-D** tool with **File > Load Data**, and select magic3d to load the 3-D data file.
- 2 After analyzing your data, save it by using **File > Save > Decomposition**.
- 3 In the dialog box that appears, specify a folder and file name for storing the decomposition data. Type the name `dec_magic3d`.
- 4 After saving the decomposition data to the file `dec_magic3d.mat`, load the variables into your workspace.

```
load dec_magic3d
whos
```

where `wdec` is

Name	Size	Bytes	Class
wdec	1x1	9182	struct

The variable `wdec` contains the wavelet decomposition structure.

```
wdec =
  sizeINI: [8 8 8]
  level: 2
  filters: [1x1 struct]
  mode: 'sym'
  dec: {15x1 cell}
  sizes: [3x3 double]
```

Saving Approximations

You can process a 3-D data in the **Wavelet 3-D** tool and then save any desired approximation, depending on the level chosen for the decomposition.

- 1** Open the **Wavelet 3-D** tool and load the file containing the 3-D data to analyze by using **File > Load Data**
- 2** Select magic3d.
- 3** Select the **File > Save > Approximations > Approximation at level 2** menu option.
- 4** In the dialog box that appears, select a folder and file name for the MAT-file. For this example, choose the name App2_magic3D.
- 5** Load the image data into your workspace.

```
load App2_magic3D  
whos
```

where x is

Name	Size	Bytes	Class
x	8x8x8	4096	double

Dual-Tree Wavelet Transforms

This example shows how the dual-tree complex discrete wavelet transform (DT-CWT) provides advantages over the critically sampled DWT for signal, image, and volume processing. The dual-tree DWT is implemented as two separate two-channel filter banks. To gain the advantages described in this example, you cannot arbitrarily choose the scaling and wavelet filters used in the two trees. The lowpass (scaling) and highpass (wavelet) filters of one tree, $\{h_0, h_1\}$, must generate a scaling function and wavelet that are approximate Hilbert transforms of the scaling function and wavelet generated by the lowpass and highpass filters of the other tree, $\{g_0, g_1\}$. Therefore, the complex-valued scaling functions and wavelets formed from the two trees are approximately analytic.

As a result, the dual-tree DWT exhibits less shift variance and more directional selectivity than the critically sampled DWT with only a 2^d redundancy factor for d -dimensional data. The redundancy in the dual-tree DWT is significantly less than the redundancy in the undecimated (stationary) DWT.

This example illustrates the approximate shift invariance of the dual-tree DWT, the selective orientation of the dual-tree analyzing wavelets in 2-D and 3-D, and the use of the dual-tree complex wavelet transform in image and volume denoising.

Near Shift Invariance of the Dual-Tree DWT

The DWT suffers from shift variance, meaning that small shifts in the input signal or image can cause significant changes in the distribution of signal/image energy across scales in the DWT coefficients. The DT-CWT is approximately shift invariant.

To demonstrate this on a test signal, construct two shifted discrete-time impulses 128 samples in length. One signal has the unit impulse at sample 60, while the other signal has the unit impulse at sample 64. Both signals clearly have unit energy (ℓ^2 norm).

```
kronDelta1 = zeros(128,1);
kronDelta1(60) = 1;
kronDelta2 = zeros(128,1);
kronDelta2(64) = 1;
```

Obtain the DWT and dual-tree DWT of the two signals down to level 3 with wavelet and scaling filters of length 14. Extract the level-3 detail coefficients for comparison.

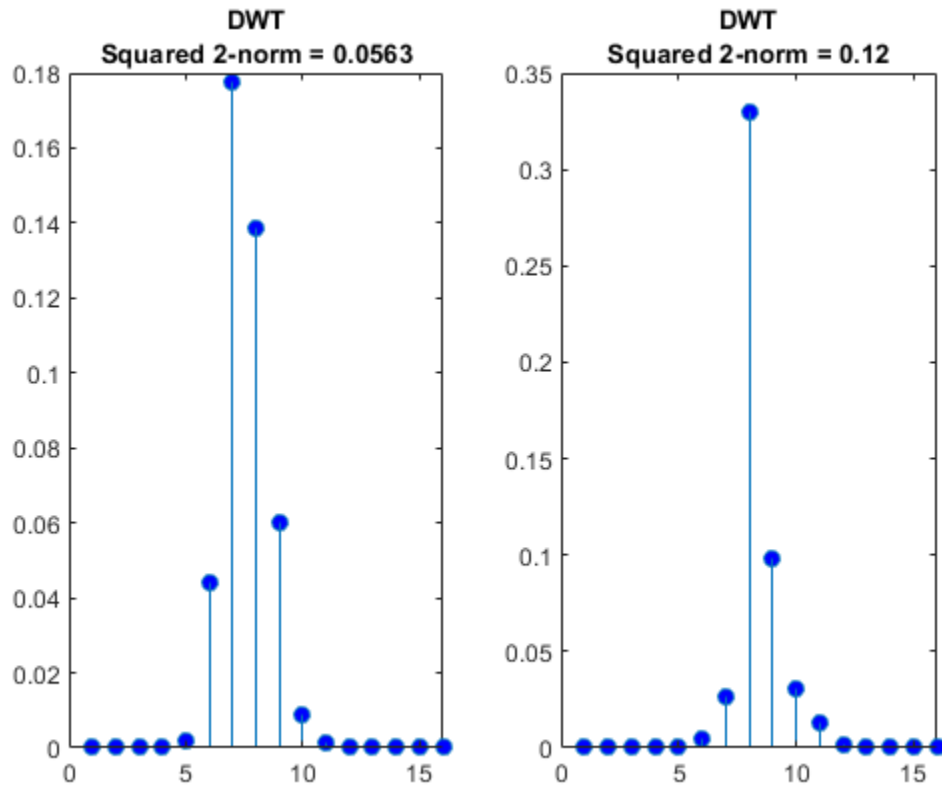
```
J = 3;
dwt1 = dddtree('dwt',kronDelta1,J,'sym7');
```

```
dwt2 = dddtree('dwt',kronDelta2,J,'sym7');
dwt1Cfs = dwt1.cfs{J};
dwt2Cfs = dwt2.cfs{J};

dt1 = dddtree('cplxdt',kronDelta1,J,'dtf3');
dt2 = dddtree('cplxdt',kronDelta2,J,'dtf3');
dt1Cfs = dt1.cfs{J}(:,:,1)+1i*dt1.cfs{J}(:,:,2);
dt2Cfs = dt2.cfs{J}(:,:,1)+1i*dt2.cfs{J}(:,:,2);
```

Plot the absolute values of the DWT and DT-CWT coefficients for the two signals at level 3 and compute the energy (squared ℓ^2 norms) of the coefficients.

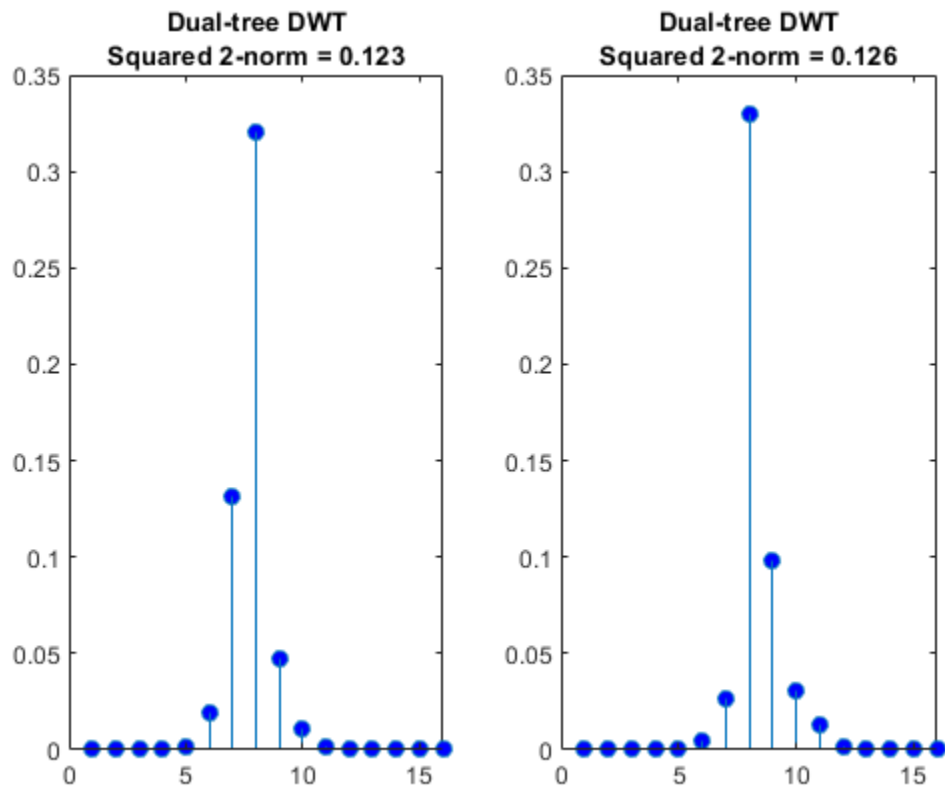
```
figure
subplot(1,2,1)
stem(abs(dwt1Cfs),'markerfacecolor',[0 0 1])
title({'DWT';['Squared 2-norm = ' num2str(norm(dwt1Cfs,2)^2,3)]},...
      'fontsize',10)
subplot(1,2,2)
stem(abs(dwt2Cfs),'markerfacecolor',[0 0 1])
title({'DWT';['Squared 2-norm = ' num2str(norm(dwt2Cfs,2)^2,3)]},...
      'fontsize',10)
```



```

figure
subplot(1,2,1)
stem(abs(dt1Cfs), 'markerfacecolor',[0 0 1])
title({'Dual-tree DWT'; ['Squared 2-norm = ' num2str(norm(dt1Cfs,2)^2,3)]},...
      'fontsize',10)
subplot(1,2,2)
stem(abs(dwt2Cfs), 'markerfacecolor',[0 0 1])
title({'Dual-tree DWT'; ['Squared 2-norm = ' num2str(norm(dt2Cfs,2)^2,3)]},...
      'fontsize',10)

```



Note the four sample shift in the signal has caused an almost 6.5% change in the energy of the level-3 DWT wavelet coefficients. However, the dual-tree wavelet coefficients show only a 0.3% change in energy.

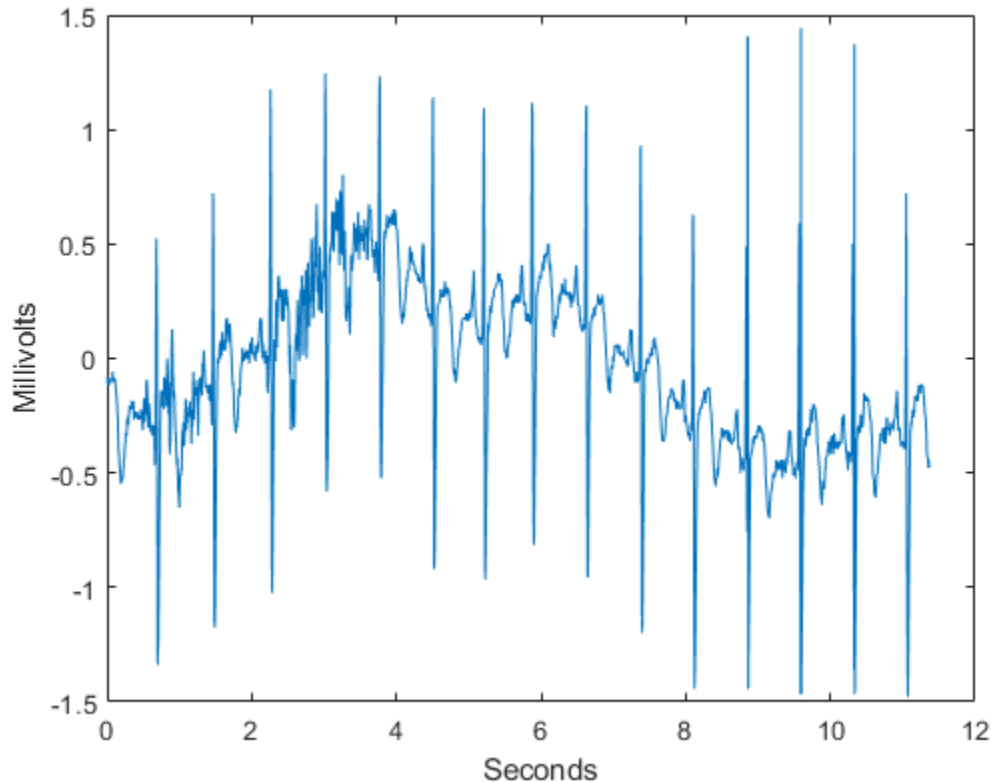
To demonstrate the utility of approximate shift invariance in real data, we analyze an electrocardiogram (ECG) signal. The sampling interval for the ECG signal is 1/180 seconds. The data are taken from Percival & Walden [3], p.125 (data originally provided by William Constantine and Per Reinhall, University of Washington). For convenience, we take the data to start at $t=0$.

```
load wecg
dt = 1/180;
t = 0:dt:(length(wecg)*dt)-dt;
figure
```

```

plot(t,wecg)
xlabel('Seconds')
ylabel('Millivolts')

```



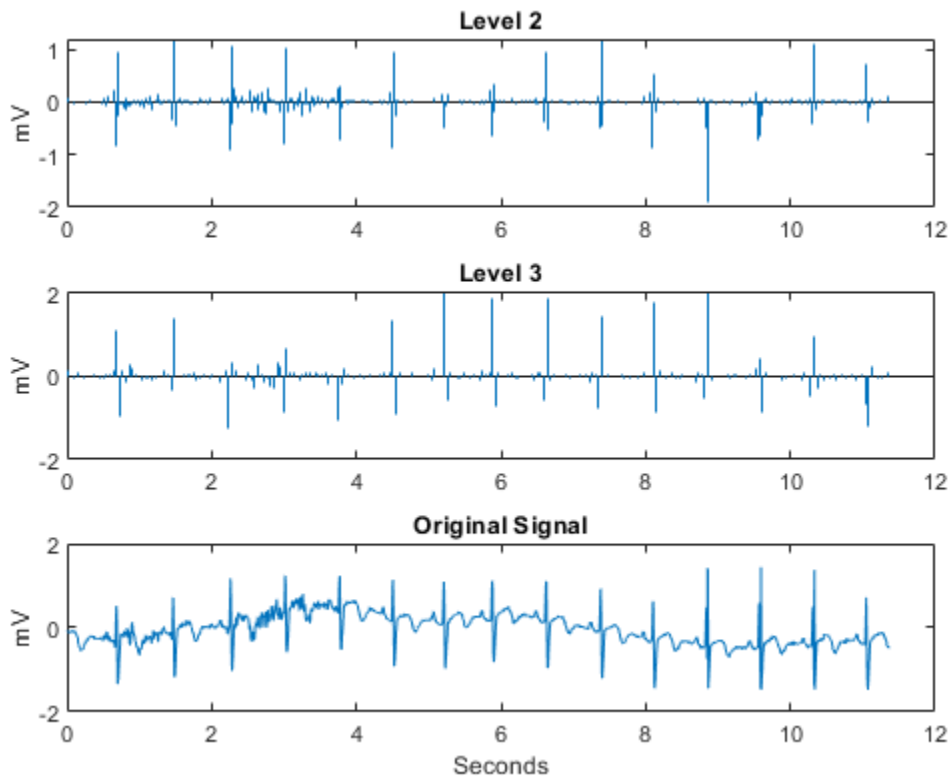
The large positive peaks approximately 0.7 seconds apart are the R waves of the cardiac rhythm. First, decompose the signal using the critically sampled DWT and plot the original signal along with the level-2 and level-3 wavelet coefficients. The level-2 and level-3 coefficients were chosen because the R waves are isolated most prominently in those scales for the given sampling rate.

```

J = 6;
dtDWT1 = dddtree('dwt',wecg,J,'farras');
details = zeros(2048,3);
details(2:4:end,2) = dtDWT1.cfs{2};

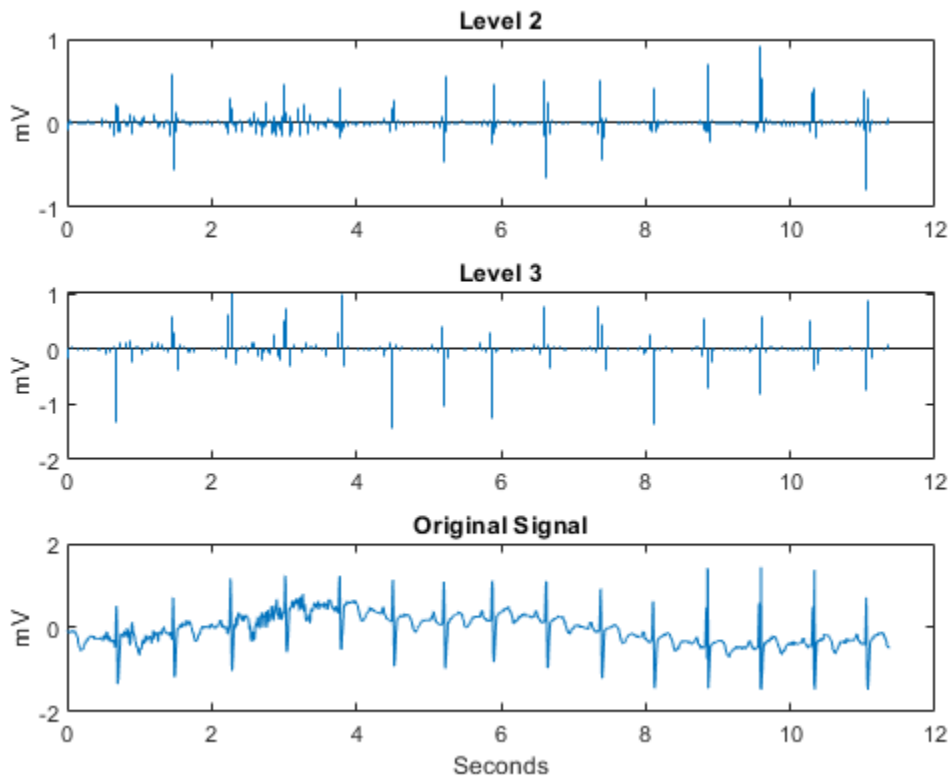
```

```
details(4:8:end,3) = dtDWT1.cfs{3};  
subplot(3,1,1)  
stem(t,details(:,2),'Marker','none','ShowBaseline','off')  
title('Level 2')  
ylabel('mV')  
subplot(3,1,2)  
stem(t,details(:,3),'Marker','none','ShowBaseline','off')  
title('Level 3')  
ylabel('mV')  
subplot(3,1,3)  
plot(t,wecg)  
title('Original Signal')  
xlabel('Seconds')  
ylabel('mV')
```



Repeat the above analysis for the dual-tree transform. In this case, just plot the real part of the dual-tree coefficients at levels 2 and 3.

```
dtcplx1 = dddtree('cplxdt',wecg,J,'dtf3');
details = zeros(2048,3);
details(2:4:end,2) = dtcplx1.cfs{2}(:,1,1)+1i*dtcplx1.cfs{2}(:,1,2);
details(4:8:end,3) = dtcplx1.cfs{3}(:,1,1)+1i*dtcplx1.cfs{3}(:,1,2);
subplot(3,1,1)
stem(t,real(details(:,2)),'Marker','none','ShowBaseline','off')
title('Level 2')
ylabel('mV')
subplot(3,1,2)
stem(t,real(details(:,3)),'Marker','none','ShowBaseline','off')
title('Level 3')
ylabel('mV')
subplot(3,1,3)
plot(t,wecg)
title('Original Signal')
xlabel('Seconds')
ylabel('mV')
```



Both the critically sampled and dual-tree wavelet transforms localize an important feature of the ECG waveform to similar scales.

An important application of wavelets in 1-D signals is to obtain an analysis of variance by scale. It stands to reason that this analysis of variance should not be sensitive to circular shifts in the input signal. Unfortunately, this is not the case with the critically sampled DWT. To demonstrate this, we circularly shift the ECG signal by 4 samples, analyze the unshifted and shifted signals with the critically sampled DWT, and calculate the distribution of energy across scales.

```
wecgShift = circshift(wecg,4);
dtDWT2 = ddtree('dwt',wecgShift,J,'farras');
sigenrgy = norm(wecg,2)^2;
enr1 = cell2mat(cellfun(@(x)(norm(x,2)^2/sigenrgy)*100,dtDWT1.cfs,'uni',0));
```



```

enr2 = cell2mat(cellfun(@(x)(norm(x,2)^2/sigenrgy)*100,dtDWT2.cfs,'uni',0));
levels = {'D1';'D2';'D3';'D4';'D5';'D6';'A6'};
enr1 = enr1(:);
enr2 = enr2(:);
table(levels,enr1,enr2,'VariableNames',{'Level','enr1','enr2'})

```

```

ans=7x3 table
    Level      enr1      enr2
    -----
    {'D1'}      4.1994    4.1994
    {'D2'}      8.425     8.425
    {'D3'}     13.381    10.077
    {'D4'}      7.0612    10.031
    {'D5'}      5.4606    5.4436
    {'D6'}      3.1273    3.4584
    {'A6'}     58.345    58.366

```

Note that the wavelet coefficients at levels 3 and 4 show approximately 3% changes in energy between the original and shifted signal. Next, we repeat this analysis using the complex dual-tree wavelet transform.

```

dteplx2 = ddtree('cplxd',wecgShift,J,'dtf3');
cfs1 = cellfun(@squeeze,dteplx1.cfs,'uni',0);
cfs2 = cellfun(@squeeze,dteplx2.cfs,'uni',0);
cfs1 = cellfun(@(x) complex(x(:,1),x(:,2)),cfs1,'uni',0);
cfs2 = cellfun(@(x) complex(x(:,1),x(:,2)),cfs2,'uni',0);
dtenr1 = cell2mat(cellfun(@(x)(norm(x,2)^2/sigenrgy)*100,cfs1,'uni',0));
dtenr2 = cell2mat(cellfun(@(x)(norm(x,2)^2/sigenrgy)*100,cfs2,'uni',0));
dtenr1 = dtenr1(:);
dtenr2 = dtenr2(:);
table(levels,dtenr1,dtenr2,'VariableNames',{'Level','dtenr1','dtenr2'})

```

```

ans=7x3 table
    Level      dtenr1    dtenr2
    -----
    {'D1'}      4.936     4.936
    {'D2'}     6.6691    6.6691
    {'D3'}     12.682    12.611
    {'D4'}      8.3891    8.4808
    {'D5'}      5.8868    5.8791
    {'D6'}      3.053     3.0415

```

```
{'A6'}    58.384    58.382
```

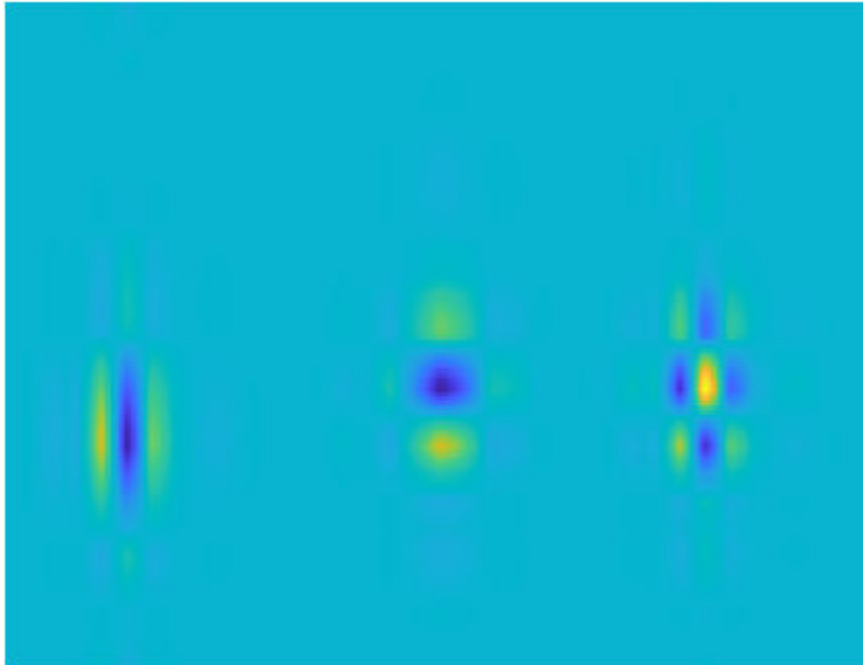
The dual-tree transform produces a consistent analysis of variance by scale for the original signal and its circularly shifted version.

Directional Selectivity in Image Processing

The standard implementation of the DWT in 2-D uses separable filtering of the columns and rows of the image. The LH, HL, and HH wavelets for Daubechies' least-asymmetric phase wavelet with 4 vanishing moments (sym4) are shown in the following figure.

```
figure
J = 5;
L = 3*2^(J+1);
N = L/2^J;
Y = zeros(L,3*L);
dt = dddtree2('dwt',Y,J,'sym4');
dt.cfs{J}(N/3,N/2,1) = 1;
dt.cfs{J}(N/2,N/2+N,2) = 1;
dt.cfs{J}(N/2,N/2+2*N,3) = 1;
dwtImage = idddtree2(dt);
imagesc(dwtImage)
axis xy
axis off
title({'Critically Sampled DWT';'2-D separable wavelets (sym4) -- LH, HL, HH'})
```

**Critically Sampled DWT
2-D separable wavelets (sym4) -- LH, HL, HH**



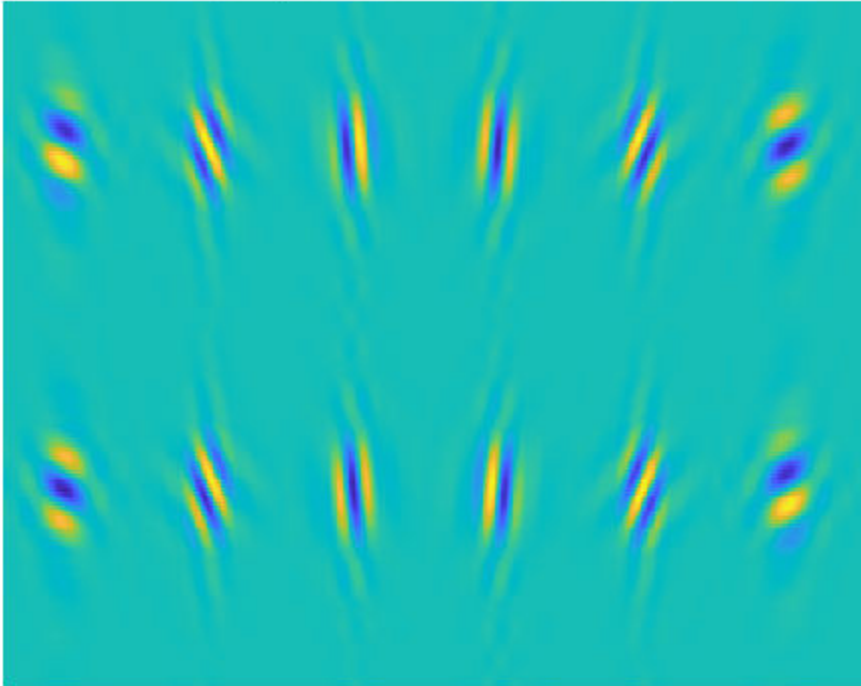
Note that the LH and HL wavelets have clear horizontal and vertical orientations respectively. However, the HH wavelet on the far right mixes both the +45 and -45 degree directions, producing a checkerboard artifact. This mixing of orientations is due to the use of real-valued separable filters. The HH real-valued separable filter has passbands in all four high frequency corners of the 2-D frequency plane.

The dual-tree DWT achieves directional selectivity by using wavelets that are approximately analytic, meaning that they have support on only one half of the frequency axis. In the dual-tree DWT, there are six subbands for both the real and imaginary parts. The six real parts are formed by adding the outputs of column filtering followed by row filtering of the input image in the two trees. The six imaginary parts are formed by subtracting the outputs of column filtering followed by row filtering.

The filters applied to the columns and rows may be from the same filter pair, $\{h_0, h_1\}$ or $\{g_0, g_1\}$, or from different filter pairs, $\{h_0, g_1\}$, $\{g_0, h_1\}$. The following code shows the orientation of the 12 wavelets corresponding to the real and imaginary parts of the complex oriented dual-tree DWT.

```
J = 4;
L = 3*2^(J+1);
N = L/2^J;
Y = zeros(2*L,6*L);
wt = dddtree2('cplxdt',Y,J,'dtf3');
wt.cfs{J}(N/2,N/2+0*N,2,2,1) = 1;
wt.cfs{J}(N/2,N/2+1*N,3,1,1) = 1;
wt.cfs{J}(N/2,N/2+2*N,1,2,1) = 1;
wt.cfs{J}(N/2,N/2+3*N,1,1,1) = 1;
wt.cfs{J}(N/2,N/2+4*N,3,2,1) = 1;
wt.cfs{J}(N/2,N/2+5*N,2,1,1) = 1;
wt.cfs{J}(N/2+N,N/2+0*N,2,2,2) = 1;
wt.cfs{J}(N/2+N,N/2+1*N,3,1,2) = 1;
wt.cfs{J}(N/2+N,N/2+2*N,1,2,2) = 1;
wt.cfs{J}(N/2+N,N/2+3*N,1,1,2) = 1;
wt.cfs{J}(N/2+N,N/2+4*N,3,2,2) = 1;
wt.cfs{J}(N/2+N,N/2+5*N,2,1,2) = 1;
waveIm = idddtree2(wt);
imagesc(waveIm)
axis off
title('Complex Dual-Tree 2-D Wavelets')
```

Complex Dual-Tree 2-D Wavelets



The top row of the preceding figure shows the six directional wavelets of the real oriented dual-tree wavelet transform. The second row shows the imaginary parts. Together the real and imaginary parts form the complex oriented dual-tree wavelet transform. The real and imaginary parts are oriented in the same direction. You can use `dddtree2` with the `'realdt'` option to obtain the real oriented dual-tree DWT, which uses only the real parts. Using the real oriented dual-tree wavelet transform, you can achieve directional selectivity, but you do not gain the full benefit of using analytic wavelets such as approximate shift invariance.

Edge Representation in Two Dimensions

The approximate analyticity and selective orientation of the complex dual-tree wavelets provide superior performance over the standard 2-D DWT in the representation of edges

in images. To illustrate this, we analyze test images with edges consisting of line and curve singularities in multiple directions using the critically sampled 2-D DWT and the 2-D complex oriented dual-tree transform. First, analyze an image of an octagon, which consists of line singularities.

```
load woctagon
figure
imagesc(woctagon)
colormap gray
title('Original Image')
axis equal
axis off
```

Original Image



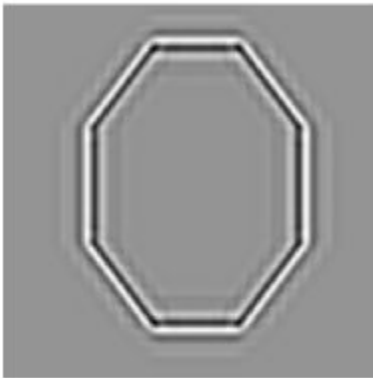
Decompose the image down to level 4 and reconstruct an image approximation based on the level-4 detail coefficients.

```
dtcplx = dddtree2('cplxdt',woctagon,4,'dtf3');
dtDWT = dddtree2('dwt',woctagon,4,'farras');

dtcplx.cfs{1} = zeros(size(dtcplx.cfs{1}));
dtcplx.cfs{2} = zeros(size(dtcplx.cfs{2}));
dtcplx.cfs{3} = zeros(size(dtcplx.cfs{3}));
dtcplx.cfs{5} = zeros(size(dtcplx.cfs{5}));

dtDWT.cfs{1} = zeros(size(dtDWT.cfs{1}));
dtDWT.cfs{2} = zeros(size(dtDWT.cfs{2}));
dtDWT.cfs{3} = zeros(size(dtDWT.cfs{3}));
dtDWT.cfs{5} = zeros(size(dtDWT.cfs{5}));

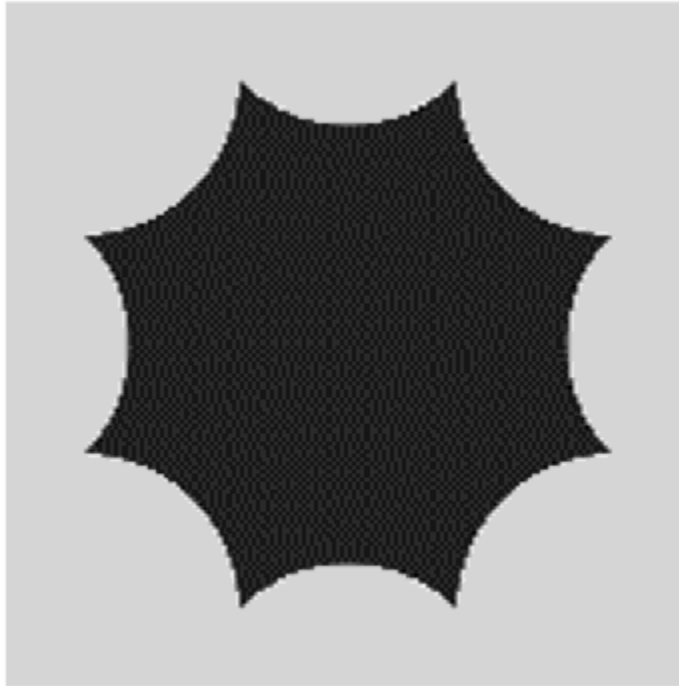
dtImage = idddtree2(dtcplx);
dwtImage = idddtree2(dtDWT);
subplot(1,2,1)
imagesc(dtImage)
axis equal
axis off
colormap gray
title('Complex Oriented Dual-Tree')
subplot(1,2,2)
imagesc(dwtImage)
axis equal
axis off
colormap gray
title('DWT')
```

Complex Oriented Dual-Tree**DWT**

Next, analyze an octagon with hyperbolic edges. The edges in the hyperbolic octagon are curve singularities.

```
load woctagonHyperbolic
figure
imagesc(woctagonHyperbolic)
colormap gray
title('Octagon with Hyperbolic Edges')
axis equal
axis off
```


Octagon with Hyperbolic Edges



Again, decompose the image down to level 4 and reconstruct an image approximation based on the level-4 detail coefficients for both the standard 2-D DWT and the complex oriented dual-tree DWT.

```
dtcplx = dddtree2('cplxdt',woctagonHyperbolic,4,'dtf3');
dtDWT = dddtree2('dwt',woctagonHyperbolic,4,'farras');
```

```
dtcplx.cfs{1} = zeros(size(dtcplx.cfs{1}));
dtcplx.cfs{2} = zeros(size(dtcplx.cfs{2}));
dtcplx.cfs{3} = zeros(size(dtcplx.cfs{3}));
dtcplx.cfs{5} = zeros(size(dtcplx.cfs{5}));
```

```
dtDWT.cfs{1} = zeros(size(dtDWT.cfs{1}));
dtDWT.cfs{2} = zeros(size(dtDWT.cfs{2}));
```

```
dtDWT.cfs{3} = zeros(size(dtDWT.cfs{3}));
dtDWT.cfs{5} = zeros(size(dtDWT.cfs{5}));

dtImage = idddtree2(dtcplx);
dwtImage = idddtree2(dtDWT);
subplot(1,2,1)
imagesc(dtImage)
axis equal
axis off
colormap gray
title('DT-CWT')
subplot(1,2,2)
imagesc(dwtImage)
axis equal
axis off
colormap gray
title('DWT')
```

DT-CWT



DWT



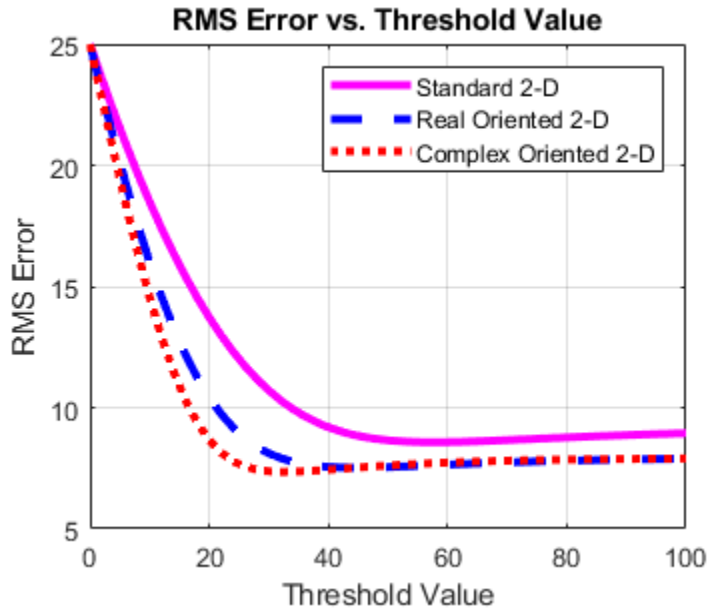
Note that the ringing artifacts evident in the 2-D critically sampled DWT are absent in the 2-D DT-CWT of both images. The DT-CWT more faithfully reproduces line and curve singularities.

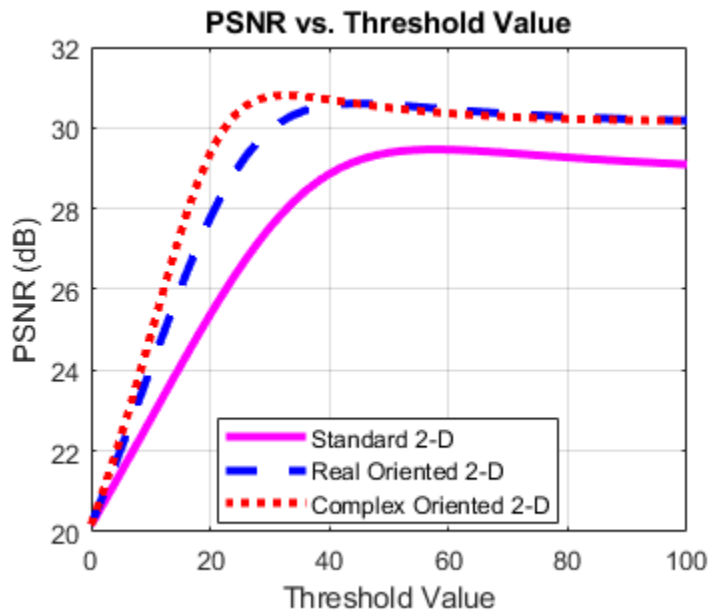
Image Denoising

Because of the ability to isolate distinct orientations in separate subbands, the dual-tree DWT is often able to outperform the standard separable DWT in applications like image denoising. To demonstrate this, use the helper function `helperCompare2DDenoising`. The helper function loads an image and adds zero-mean white Gaussian noise with $\sigma = 25$. For a user-supplied range of thresholds, the function compares denoising using soft thresholding for the critically sampled DWT, the real oriented dual-tree DWT, and the

complex oriented dual-tree DWT. For each threshold value, the root-mean-square (RMS) error and peak signal-to-noise ratio (PSNR) are displayed.

```
numex = 3;  
helperCompare2DDenoising(numex,0:2:100, 'PlotMetrics');
```





Both the real oriented and the complex oriented dual-tree DWTs outperform the standard DWT in RMS error and PSNR.

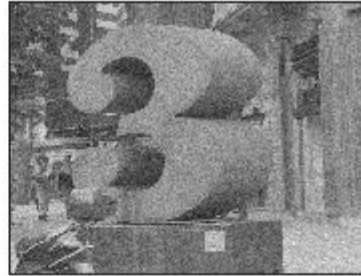
Next, obtain the denoised images for a threshold value of 25, which is equal to the standard deviation of the additive noise.

```
numex = 3;
helperCompare2DDenoising(numex, 25, 'PlotImage');
```

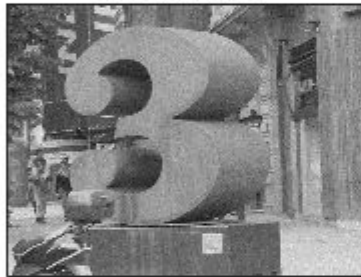
Original Image



Noisy Image



**Denoised Image
Standard 2-D**

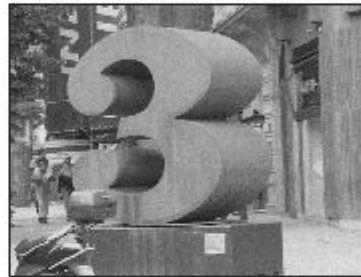


THR: 25.00

RMSE: 12.01

PSNR: 26.54

**Denoised Image
Real Oriented 2-D Dual-Tree**



THR: 25.00

RMSE: 8.95

PSNR: 29.10

**Denoised Image
Complex Oriented 2-D Dual-Tree**



THR: 25.00

With a threshold value equal to the standard deviation of the additive noise, the complex oriented dual-tree transform provides a PSNR almost 4 dB higher than the standard 2-D DWT.

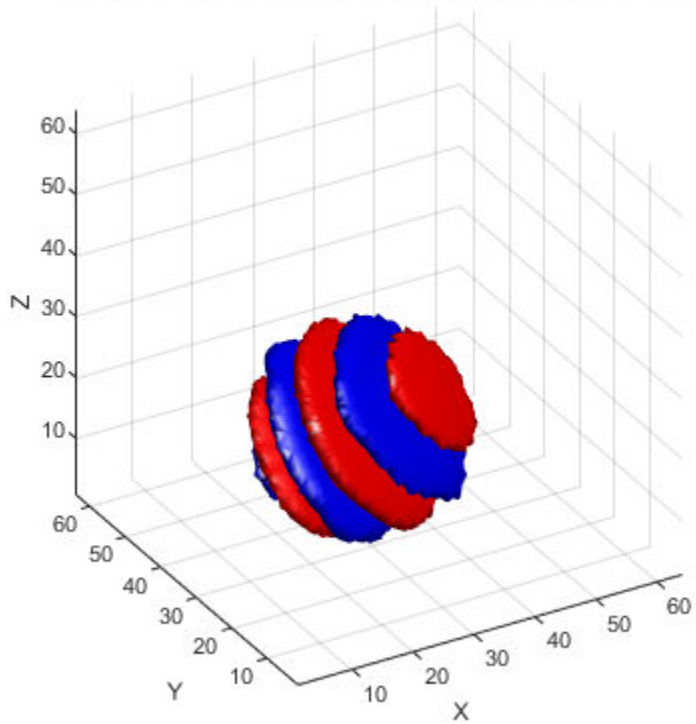
Directional Selectivity in 3-D

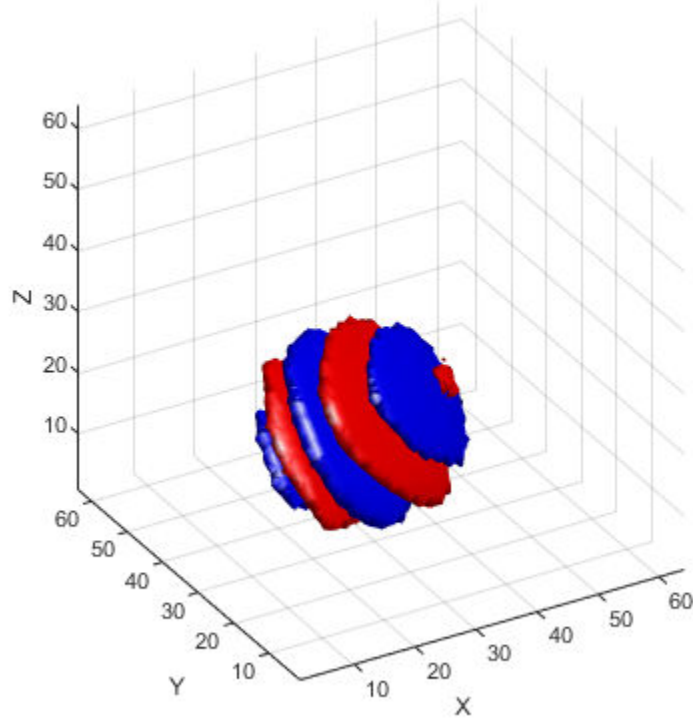
The ringing artifacts observed with the separable DWT in two dimensions is exacerbated when extending wavelet analysis to higher dimensions. The DT-CWT enables you to maintain directional selectivity in 3-D with minimal redundancy. In 3-D, there are 28 wavelet subbands in the dual-tree transform.

To demonstrate the directional selectivity of the 3-D dual-tree wavelet transform, visualize example 3-D isosurfaces of both 3-D dual-tree and separable DWT wavelets. First, visualize the real and imaginary parts separately of two dual-tree subbands.

```
helperVisualize3D('Dual-Tree',28,'separate');
```

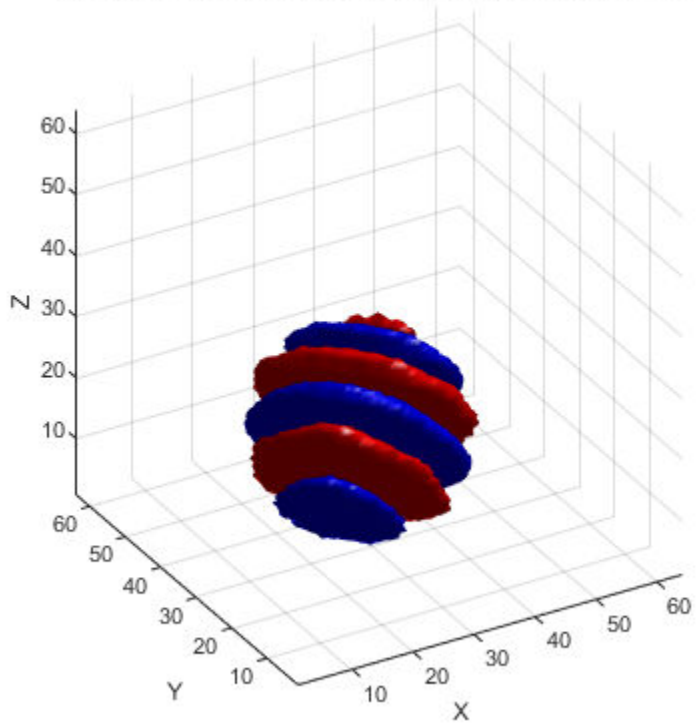
3-D WAVELET ISOSURFACE (REAL PART) SUBBAND 28

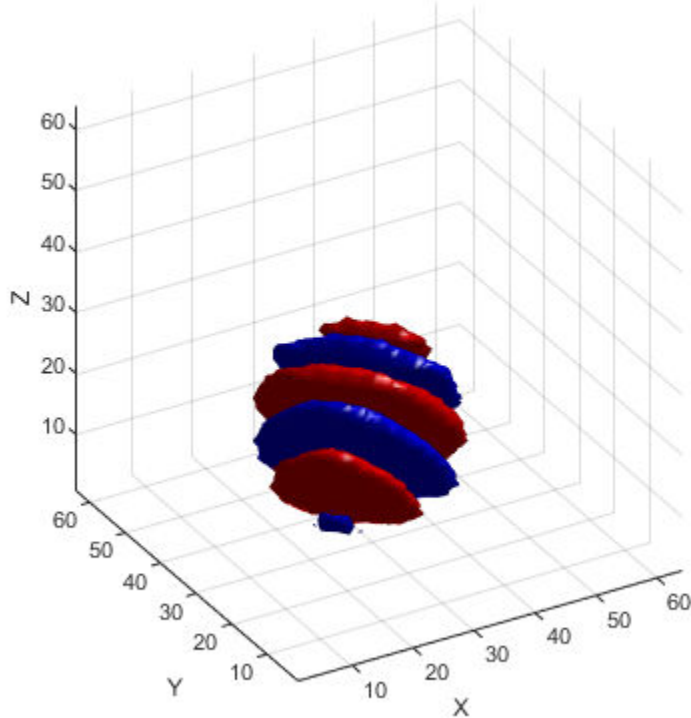


3-D WAVELET ISOSURFACE (IMAGINARY PART) SUBBAND 28

```
helperVisualize3D('Dual-Tree',25,'separate');
```

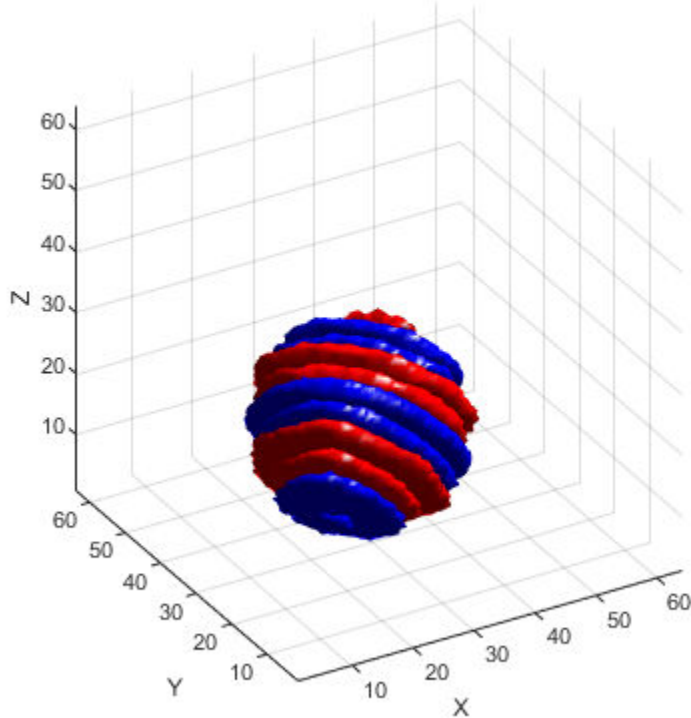
3-D WAVELET ISOSURFACE (REAL PART) SUBBAND 25



3-D WAVELET ISOSURFACE (IMAGINARY PART) SUBBAND 25

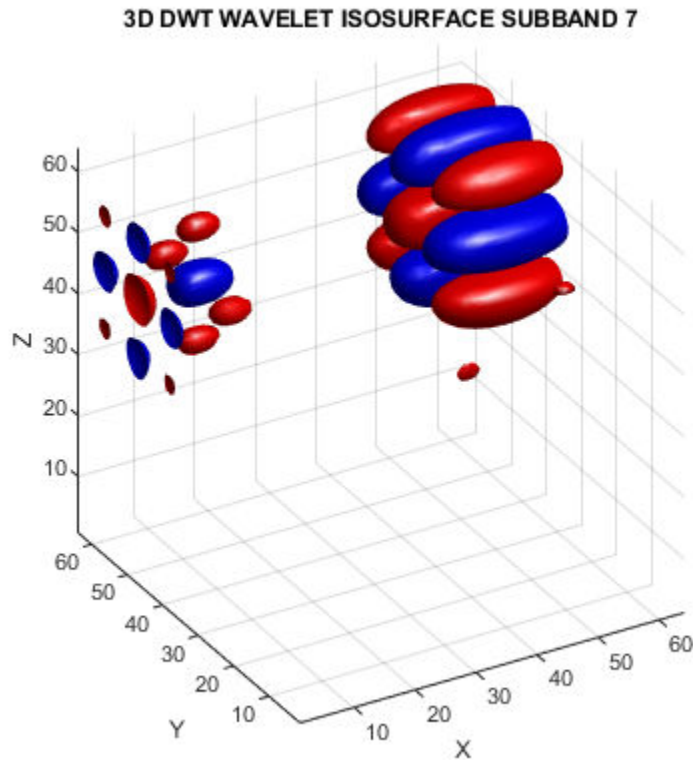
The red portion of the isosurface plot indicates the positive excursion of the wavelet from zero, while blue denotes the negative excursion. You can clearly see the directional selectivity in space of the real and imaginary parts of the dual-tree wavelets. Now visualize one of the dual-tree subbands with the real and imaginary plots plotted together as one isosurface.

```
helperVisualize3D('Dual-Tree',25,'real-imaginary');
```

3-D WAVELET ISOSURFACE (REAL AND IMAGINARY PARTS) SUBBAND 25

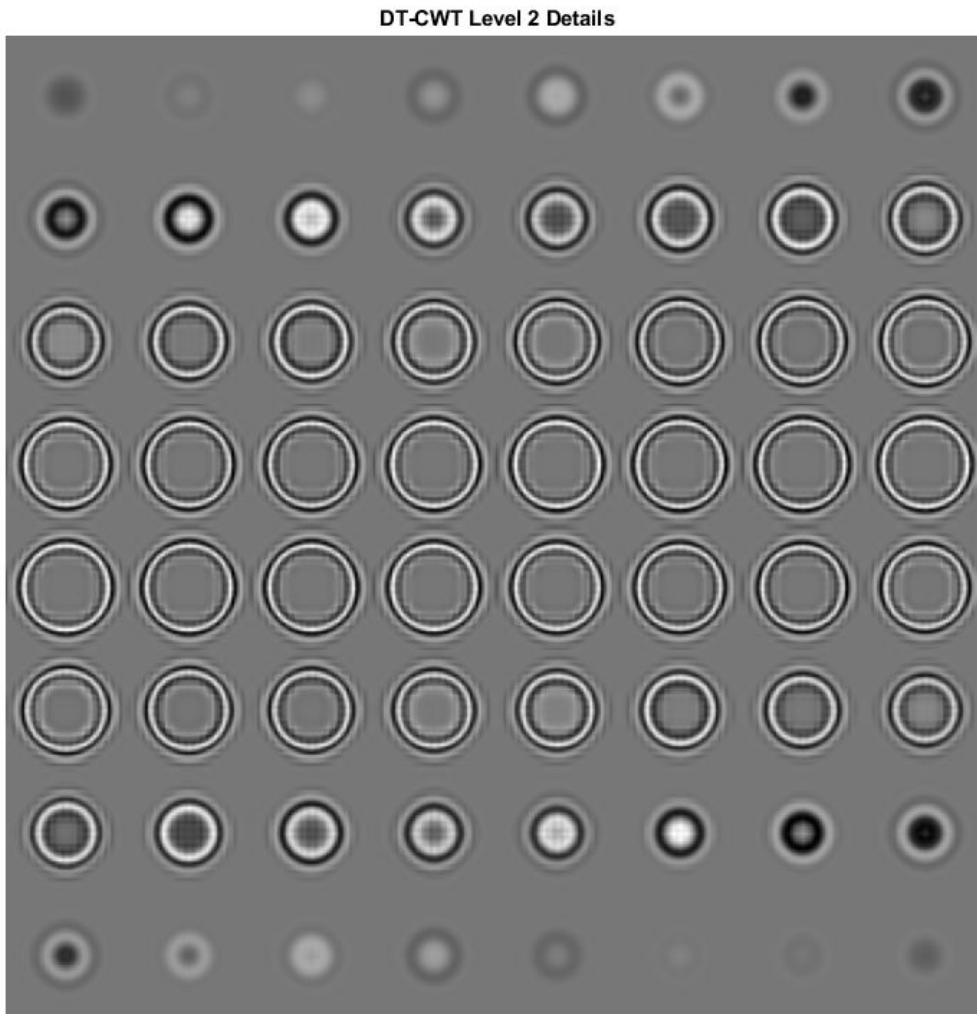
The preceding plot demonstrates that the real and imaginary parts are shifted versions of each other in space. This reflects the fact that the imaginary part of the complex wavelet is the approximate Hilbert transform of the real part. Next, visualize the isosurface of a real orthogonal wavelet in 3-D for comparison.

```
helperVisualize3D('DWT',7);
```



The mixing of orientations observed in the 2-D DWT is even more pronounced in 3-D. Just as in the 2-D case, the mixing of orientations in 3-D leads to significant ringing, or blocking artifacts. To demonstrate this, examine the 3-D DWT and DT-CWT wavelet details of a spherical volume. The sphere is 64-by-64-by-64.

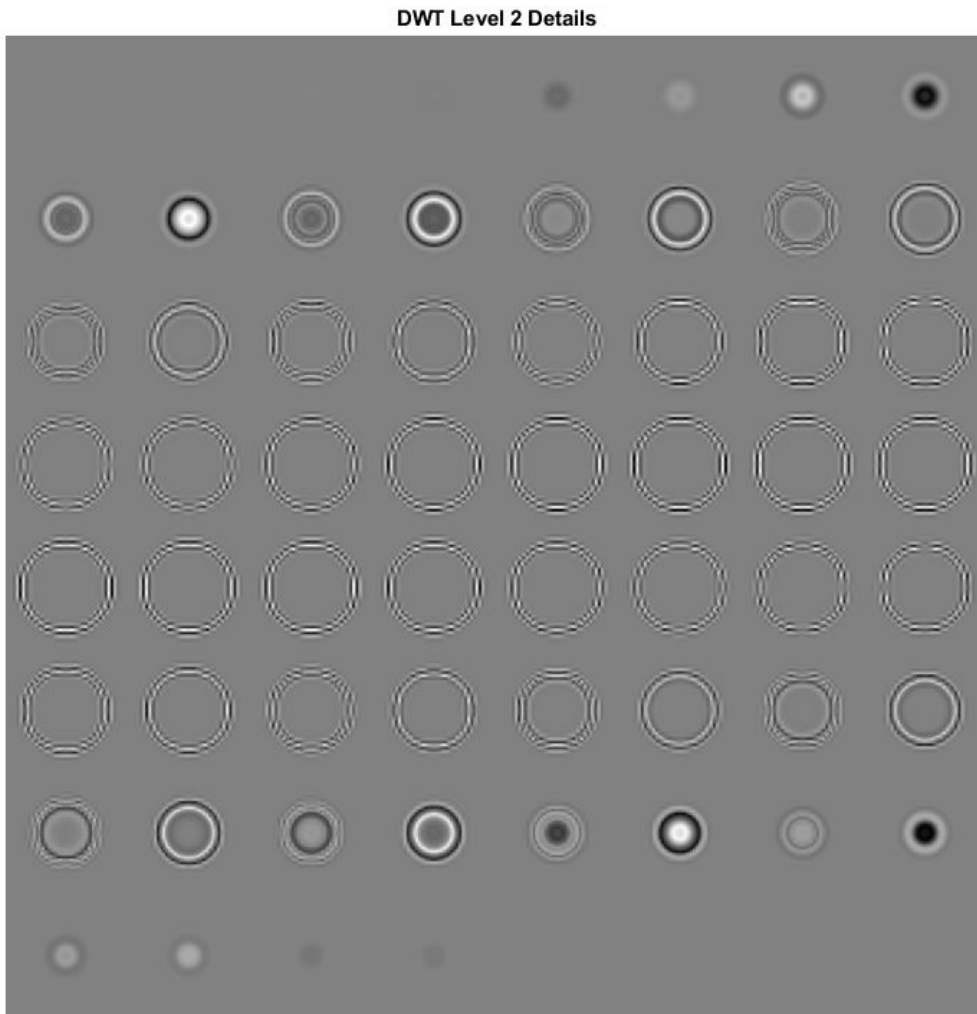
```
load sphr
[A,D] = dualtree3(sphr,2,'excludeL1');
A = zeros(size(A));
sphrDTCWT = idualtree3(A,D);
montage(reshape(sphrDTCWT,[64 64 1 64]),'DisplayRange',[])
title('DT-CWT Level 2 Details')
```



Compare the preceding plot against the second-level details based on the separable DWT.

```
sphrDEC = wavedec3(sphr,2,'sym4','mode','per');  
sphrDEC.dec{1} = zeros(size(sphrDEC.dec{1}));  
for kk = 2:8
```

```
    sphrDEC.dec{kk} = zeros(size(sphrDEC.dec{kk}));  
end  
sphrrcdWT = waverec3(sphrDEC);  
figure  
montage(reshape(sphrrcdWT,[64 64 1 64]),'DisplayRange',[])  
title('DWT Level 2 Details')
```



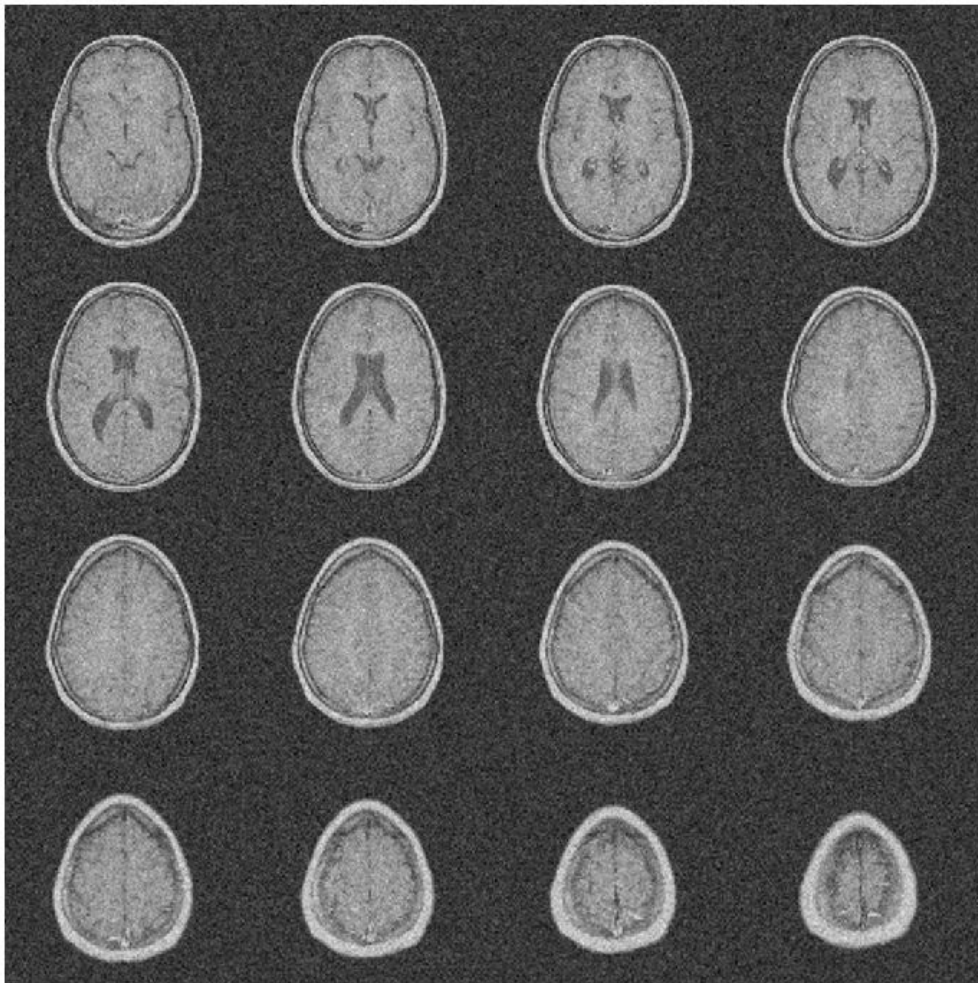
Zoom in on the images in both the DT-CWT and DWT montages and you will see how prominent the blocking artifacts in the DWT details are compared to the DT-CWT.

Volume Denoising

Similar to the 2-D case, the directional selectivity of the 3-D DT-CWT often leads to improvements in volume denoising.

To demonstrate this, consider an MRI dataset consisting of 16 slices. Gaussian noise with a standard deviation of 10 has been added to the original dataset. Display the noisy dataset.

```
load MRI3D
montage(reshape(noisyMRI,[128 128 1 16]),'DisplayRange',[1])
```



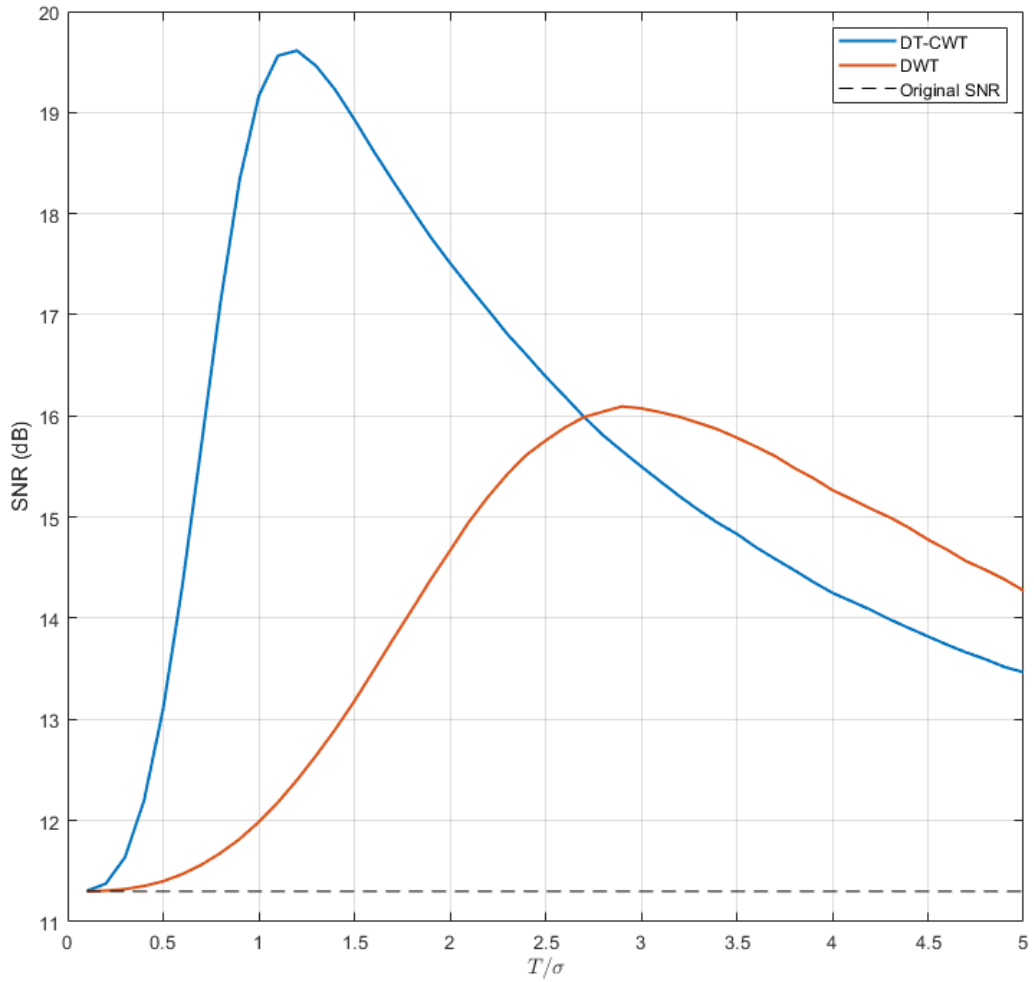
Note that the original SNR prior to denoising is approximately 11 dB.

```
20*log10(norm(origMRI(:),2)/norm(origMRI(:)-noisyMRI(:),2))
```

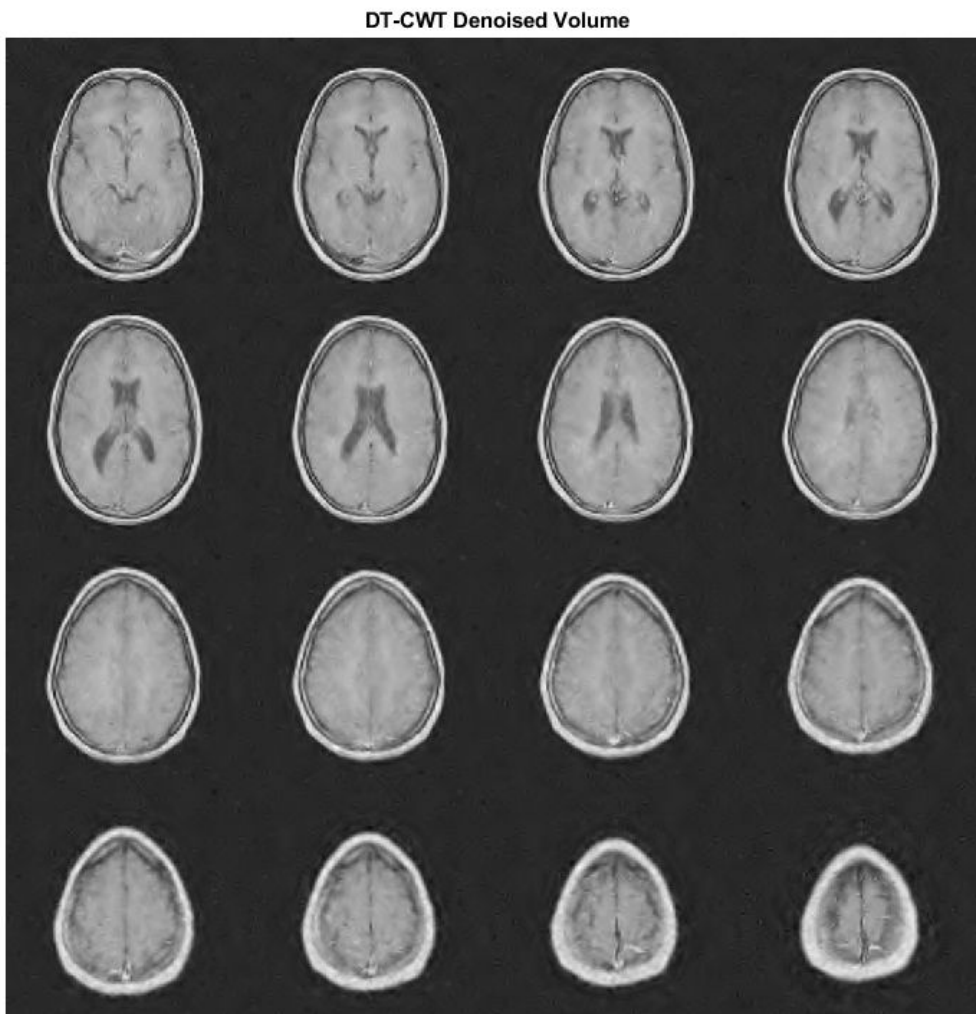
```
ans = 11.2997
```

Denoise the MRI dataset down to level 4 using both the DT-CWT and the DWT. Similar wavelet filter lengths are used in both cases. Plot the resulting SNR as a function of the threshold. Display the denoised results for both the DT-CWT and DWT obtained at the best SNR.

```
[imrecDTCWT,imrecDWT] = helperCompare3DDenoising(origMRI,noisyMRI);
```

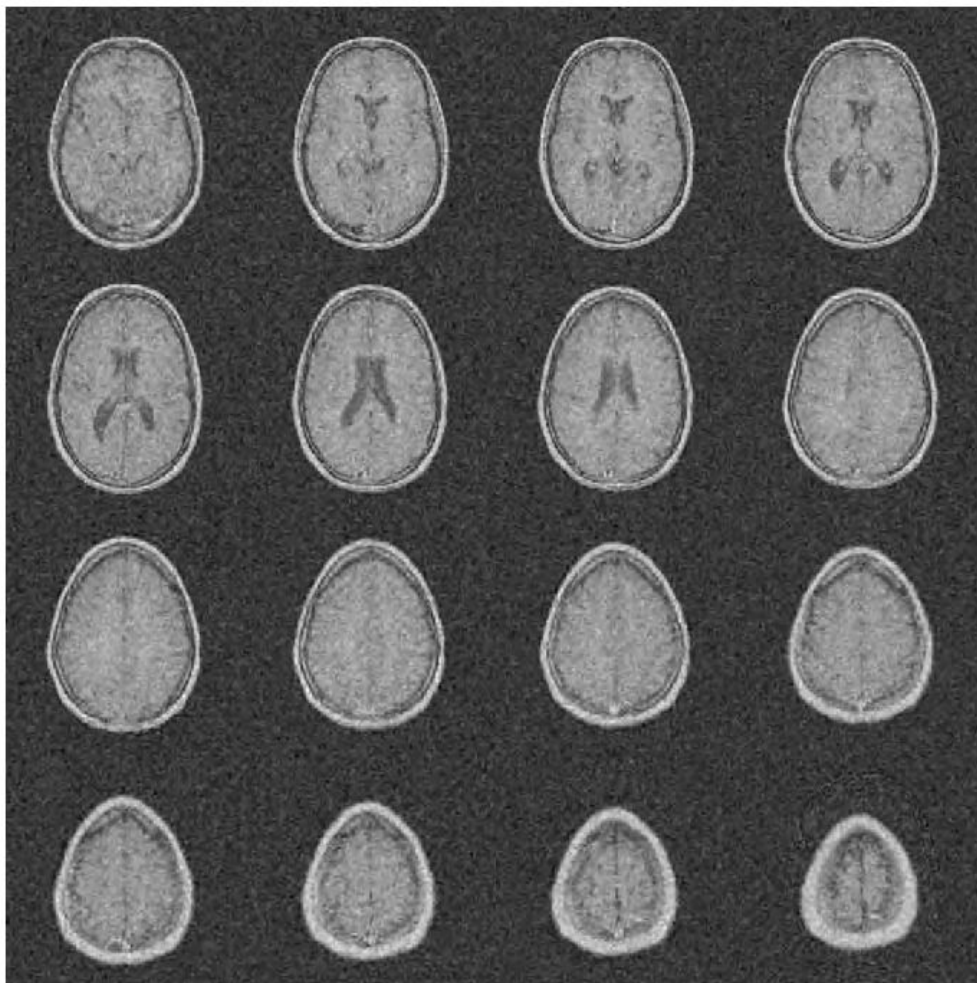


```
figure
montage(reshape(imrecDTCWT,[128 128 1 16]),'DisplayRange',[1])
title('DT-CWT Denoised Volume')
```



```
figure
montage(reshape(imrecDWT,[128 128 1 16]),'DisplayRange',[1])
title('DWT Denoised Volume')
```

DWT Denoised Volume



Summary

We have shown that the dual-tree DWT possesses the desirable properties of near shift invariance and directional selectivity not achievable with the critically sampled DWT. We

have demonstrated how these properties can result in improved performance in signal analysis, the representation of edges in images and volumes, and image and volume denoising.

Analytic Wavelets Using the Dual-Tree Wavelet Transform

This example shows how to create approximately analytic wavelets using the dual-tree complex wavelet transform. The example demonstrates that you cannot arbitrarily choose the analysis (decomposition) and synthesis (reconstruction) filters to obtain an approximately analytic wavelet. The FIR filters in the two filter banks must be carefully constructed in order to obtain an approximately analytic wavelet transform and derive the benefits of the dual-tree transform.

Obtain the lowpass and highpass analysis filters.

```
DF = dtfilters('dtf1');
```

DF is a 1-by-2 cell array of N -by-2 matrices containing the first-stage lowpass and highpass filters, DF{1}, and the lowpass and highpass filters for subsequent stages, DF{2}.

Create the zero signal 256 samples in length. Obtain two dual-tree transforms of the zero signal down to level 5.

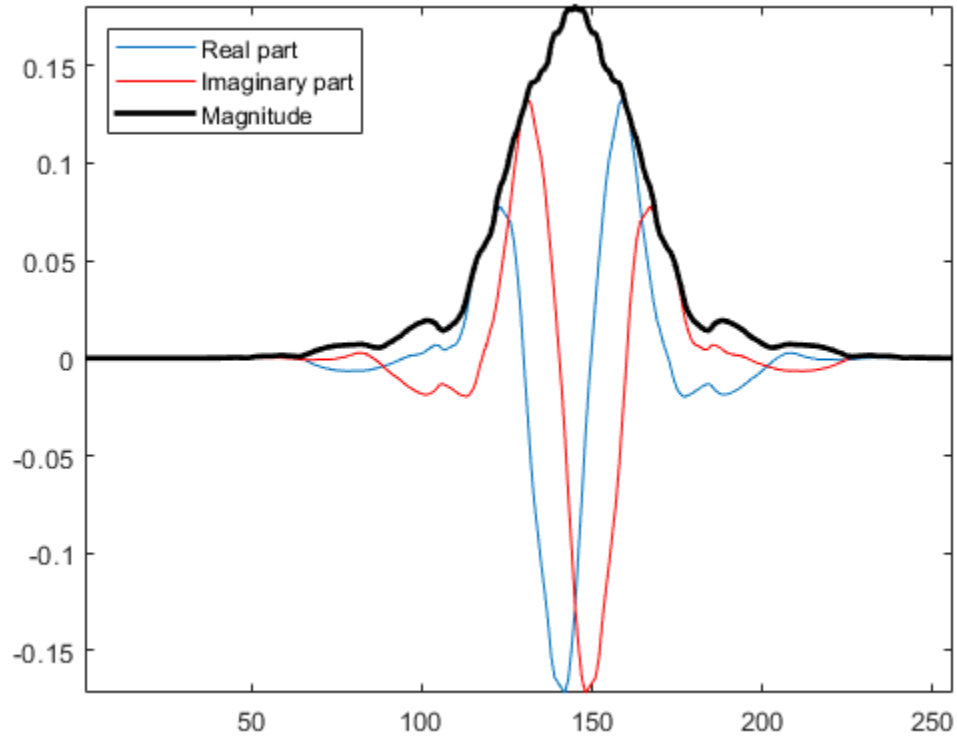
```
x = zeros(256,1);  
wt1 = dddtree('cplxdt',x,5,DF{1},DF{2});  
wt2 = dddtree('cplxdt',x,5,DF{1},DF{2});
```

Set a single level-five detail coefficient in each of the two trees to 1 and invert the transform to obtain the wavelets.

```
wt1.cfs{5}(5,1,1) = 1;  
wt2.cfs{5}(5,1,2) = 1;  
wav1 = idddtree(wt1);  
wav2 = idddtree(wt2);
```

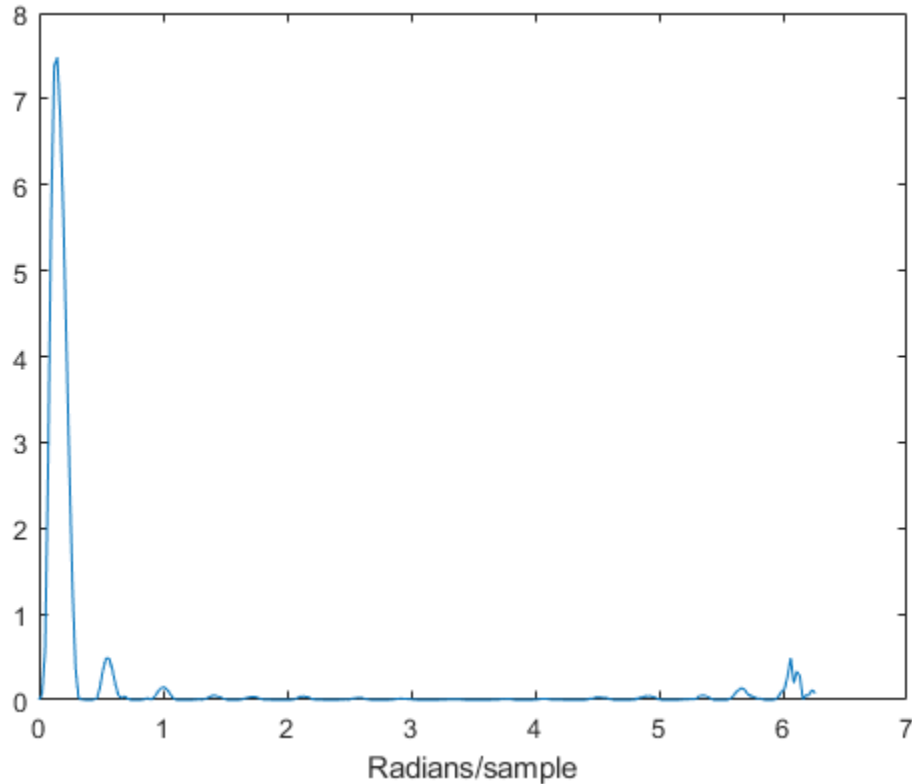
Form the complex wavelet using the first tree as the real part and the second tree as the imaginary part. Plot the real and imaginary parts of the wavelet.

```
analwav = wav1+1i*wav2;  
plot(real(analwav)); hold on;  
plot(imag(analwav),'r')  
plot(abs(analwav),'k','linewidth',2)  
axis tight;  
legend('Real part','Imaginary part','Magnitude','Location','Northwest');
```

Fourier transform the analytic wavelet and plot the magnitude.

```
zdfc = fft(analwav);  
domega = (2*pi)/length(analwav);  
omega = 0:domega:(2*pi)-domega;  
clf;  
plot(omega,abs(zdfc))  
xlabel('Radians/sample');
```

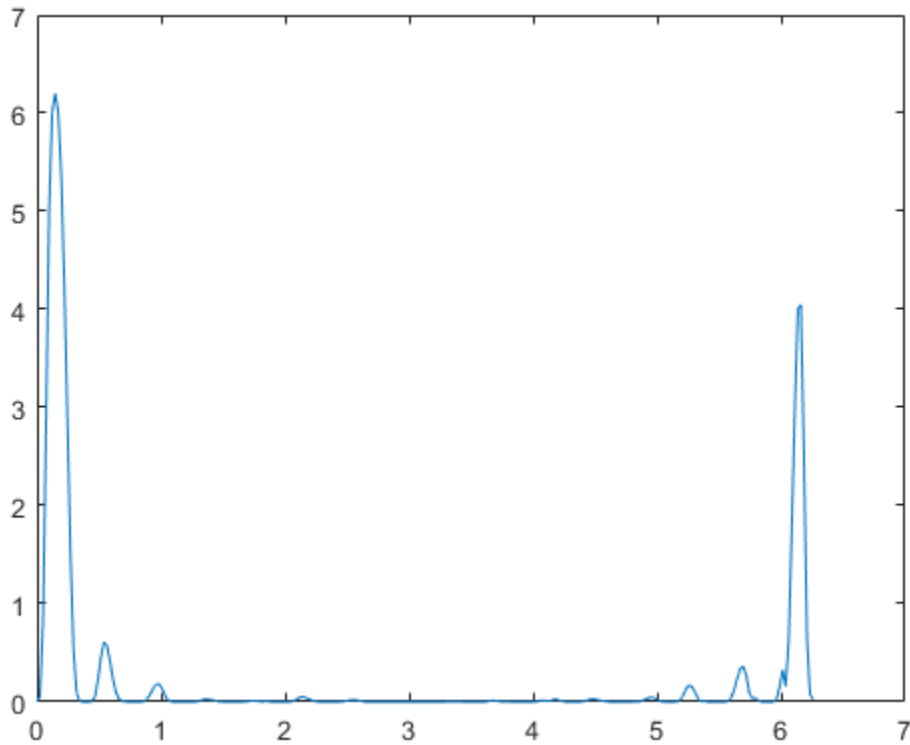


The Fourier transform of the wavelet has support on essentially only half of the frequency axis.

Repeat the preceding procedure with two arbitrarily chosen orthogonal wavelets, 'db4' and 'sym4'.

```
[LoD1,HiD1] = wfilters('db4');
[LoD2, HiD2] = wfilters('sym4');
df = {[LoD1' HiD1'],[LoD2',HiD2']};
wt1 = dddtree('cplxdt',x,5,df,df);
wt2 = dddtree('cplxdt',x,5,df,df);
wt1.cfs{5}(5,1,1) = 1;
wt2.cfs{5}(5,1,2) = 1;
wav1 = idddtree(wt1);
```

```
wav2 = iddtree(wt2);  
analwav = wav1+1i*wav2;  
zdft = fft(analwav);  
domega = (2*pi)/length(analwav);  
omega = 0:domega:(2*pi)-domega;  
clf;  
plot(omega,abs(zdft))
```



Using arbitrary orthogonal wavelets in the two trees does not result in approximately analytic wavelets. The Fourier transform of the resulting wavelet has support over the entire frequency axis.

Multifractal Analysis

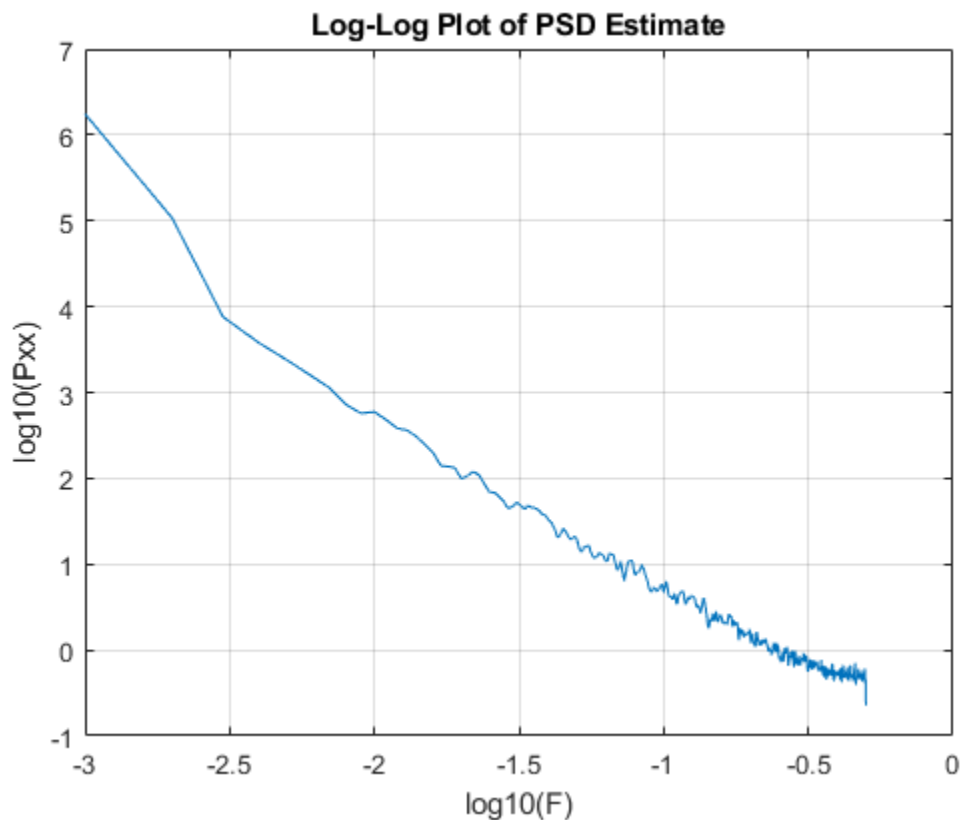
This example shows how to use wavelets to characterize local signal regularity. The ability to describe signal regularity is important when dealing with phenomena that have no characteristic scale. Signals with scale-free dynamics are widely observed in a number of different application areas including biomedical signal processing, geophysics, finance, and internet traffic. Whenever you apply some analysis technique to your data, you are invariably assuming something about the data. For example, if you use autocorrelation or power spectral density (PSD) estimation, you are assuming that your data is translation invariant, which means that signal statistics like mean and variance do not change over time. Signals with no characteristic scale are scale-invariant. This means that the signal statistics do not change if we stretch or shrink the time axis. Classical signal processing techniques typically fail to adequately describe these signals or reveal differences between signals with different scaling behavior. In these cases, fractal analysis can provide unique insights. Some of the following examples use `pwelch` and `xcorr` for illustration. To execute that code, you must have the Signal Processing Toolbox™.

Power Law Processes

An important class of signals with scale-free dynamics have autocorrelation or power spectral densities (PSD) that follow a power law. A power-law process has a PSD of the form $C|\omega|^{-\alpha}$ for some positive constant, C , and some exponent α . In some instances, the signal of interest exhibits a power-law PSD. In other cases, the signal of interest is corrupted by noise with a power-law PSD. These noises are often referred to as *colored*. Being able to estimate the exponent from realizations of these processes has important implications. For one, it allows you to make inferences about the mechanism generating the data as well as providing empirical evidence to support or reject theoretical predictions. In the case of an interfering noise with a power-law PSD, it is helpful in designing effective filters.

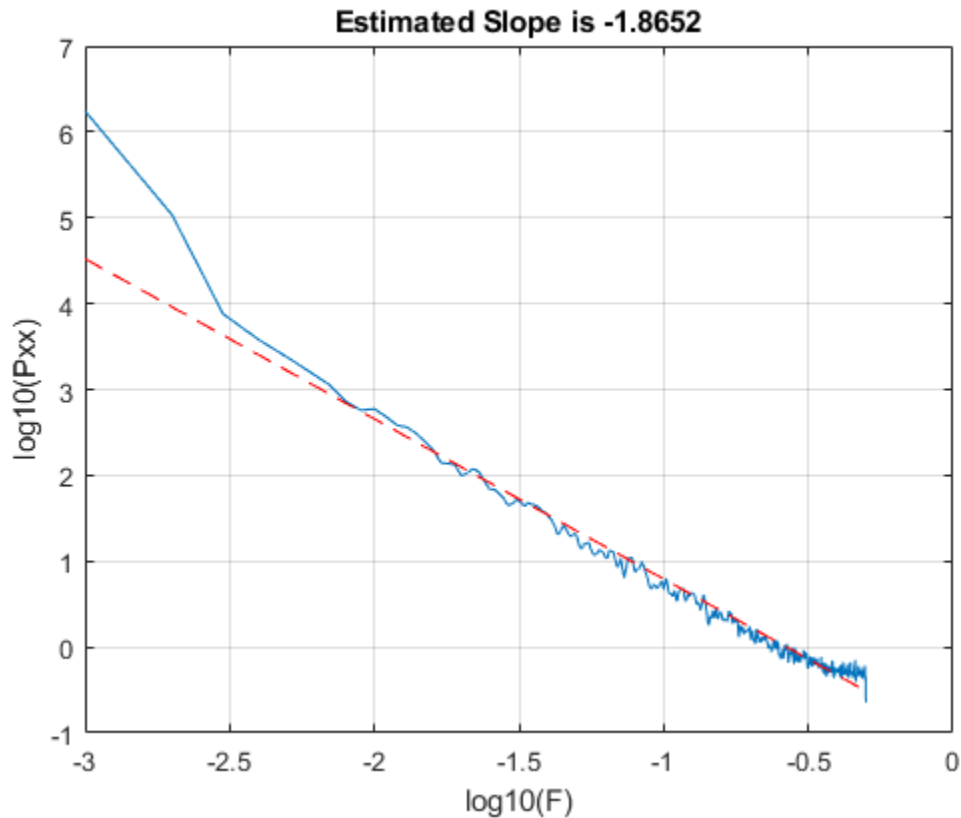
Brown noise, or a Brownian process, is one such colored noise process with a theoretical exponent of $\alpha = 2$. One way to estimate the exponent of a power law process is to fit a least-squares line to a log-log plot of the PSD.

```
load brownnoise;
[Pxx,F] = pwelch(brownnoise,kaiser(1000,10),500,1000,1);
plot(log10(F(2:end)),log10(Pxx(2:end)));
grid on;
xlabel('log10(F)'); ylabel('log10(Pxx)');
title('Log-Log Plot of PSD Estimate')
```



Regress the log PSD values on the log frequencies. Note you must ignore zero frequency to avoid taking the log of zero.

```
Xpred = [ones(length(F(2:end)),1) log10(F(2:end))];
b = lscov(Xpred,log10(Pxx(2:end)));
y = b(1)+b(2)*log10(F(2:end));
hold on;
plot(log10(F(2:end)),y,'r--');
title(['Estimated Slope is ' num2str(b(2))]);
```



Alternatively, you can use both discrete and continuous wavelet analysis techniques to estimate the exponent. The relationship between the Holder exponent, H , returned by `dwtleader` and `wtmm` and α in this scenario is $\alpha = 2H + 1$.

```
[dwbrown,hbrown,cpbrown] = dwtleader(brownnoise);
hexp = wtmm(brownnoise);
fprintf('Wavelet leader estimate is %1.2f\n',-2*cpbrown(1)-1);
```

```
Wavelet leader estimate is -1.91
```

```
fprintf('WTMM estimate is %1.2f\n',-2*hexp-1);
```

```
WTMM estimate is -2.00
```

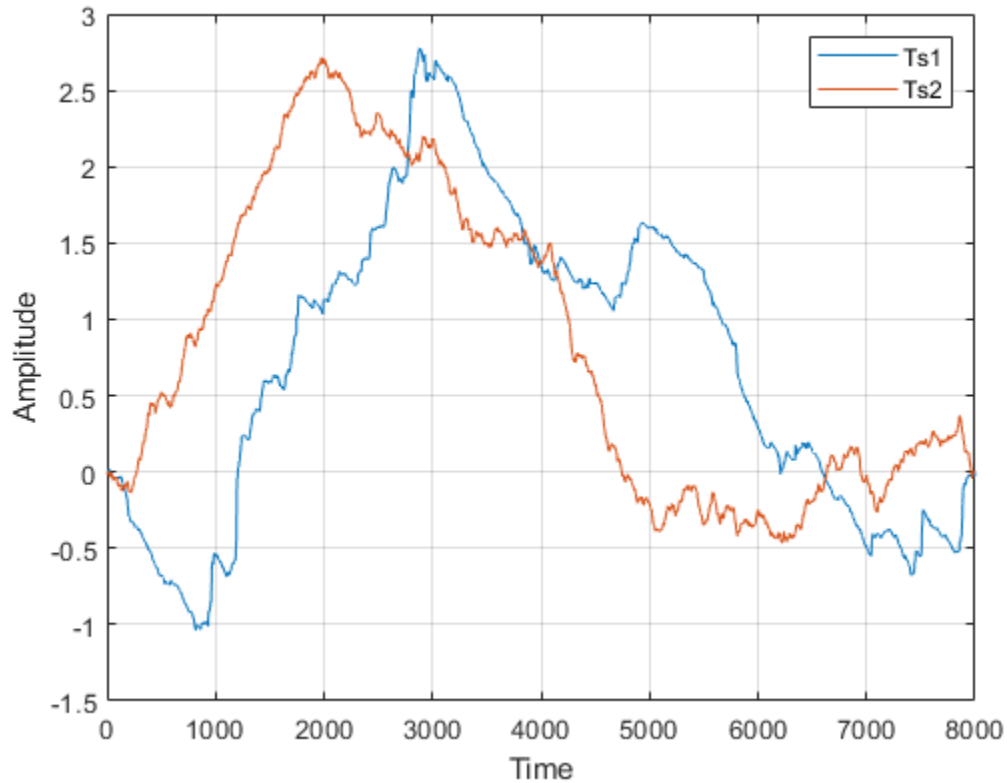
In this case, the estimate obtained by fitting a least-squares line to the log of the PSD estimate and those obtained using wavelet methods are in good agreement.

Multifractal Analysis

There are a number of real-world signals that exhibit nonlinear power-law behavior that depends on higher-order moments and scale. Multifractal analysis provides a way to describe these signals. Multifractal analysis consists of determining whether some type of power-law scaling exists for various statistical moments at different scales. If this scaling behavior is characterized by a single scaling exponent, or equivalently is a linear function of the moments, the process is *monofractal*. If the scaling behavior by scale is a nonlinear function of the moments, the process is *multifractal*. The brown noise from the previous section is an example of monofractal process and this is demonstrated in a later section.

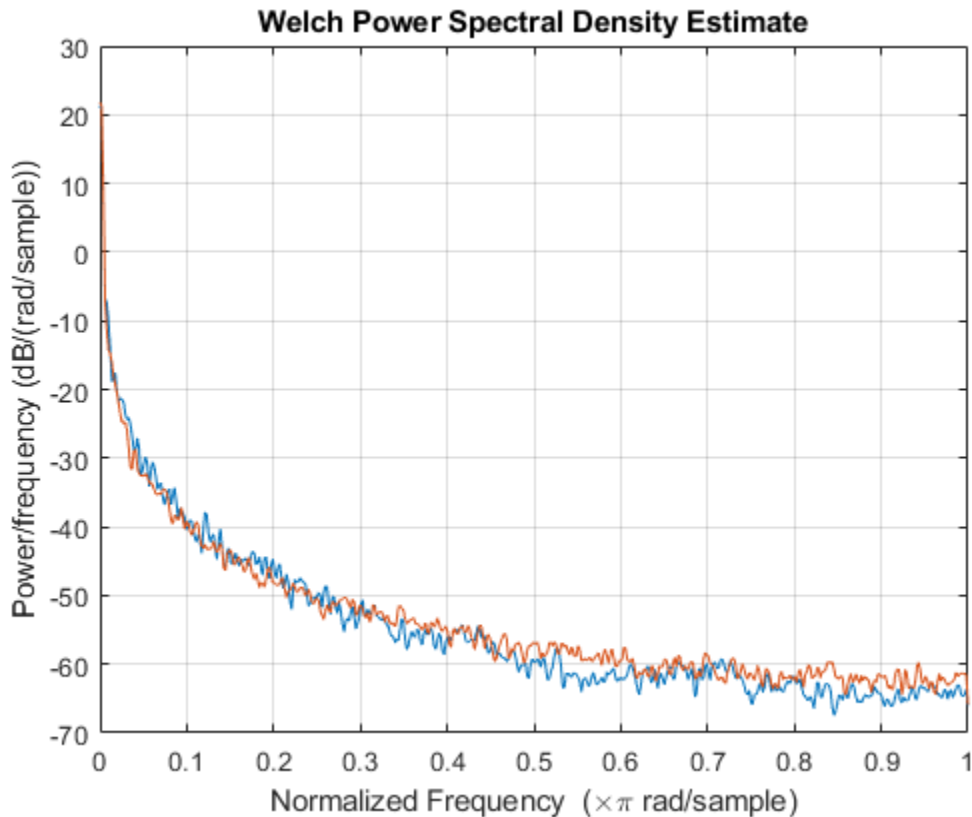
To illustrate how fractal analysis can reveal signal structure not apparent with more classic signal processing techniques, load `RWdata.mat` which contains two time series (`Ts1` and `Ts2`) with 8000 samples each. Plot the data.

```
load RWdata;  
figure;  
plot([Ts1 Ts2]); grid on;  
legend('Ts1', 'Ts2', 'Location', 'NorthEast');  
xlabel('Time'); ylabel('Amplitude');
```



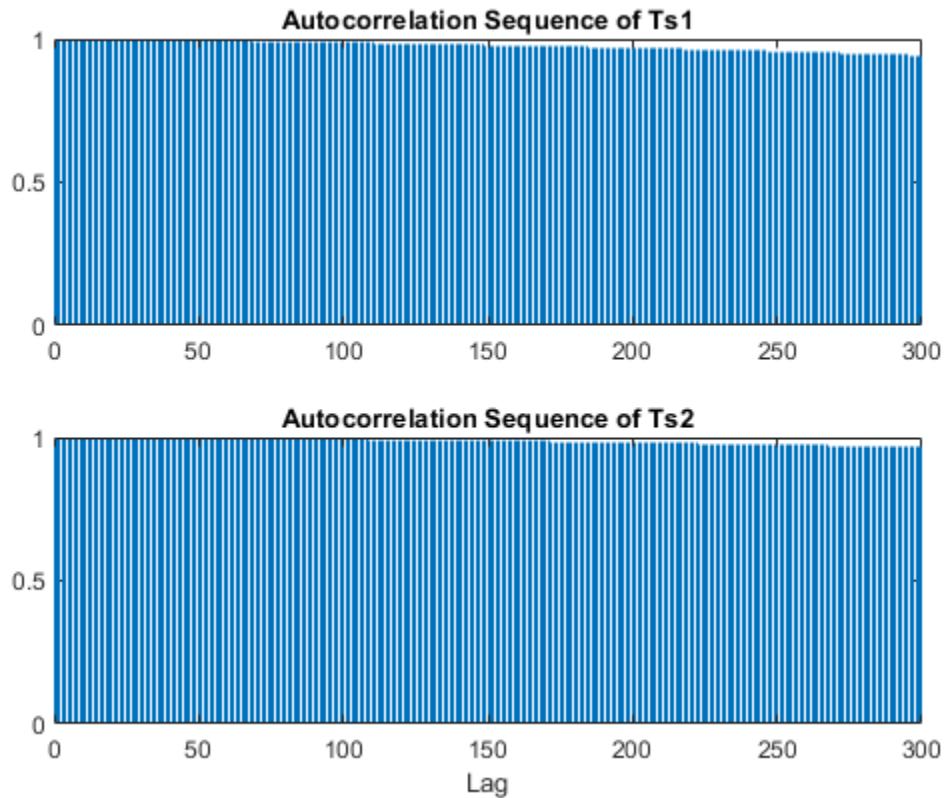
The signals have very similar second order statistics. If you look at the means, RMS values, and variances of Ts1 and Ts2, the values are almost identical. The PSD estimates are also very similar.

```
pwelch([Ts1 Ts2],kaiser(1e3,10))
```

The autocorrelation sequences decay very slowly for both time series and are not informative for differentiating the time series.

```
[xc1,lags] = xcorr(Ts1,300,'coef');
xc2 = xcorr(Ts2,300,'coef');
subplot(2,1,1)
hs1 = stem(lags(301:end),xc1(301:end));
hs1.Marker = 'none';
title('Autocorrelation Sequence of Ts1');
subplot(2,1,2)
hs2 = stem(lags(301:end),xc2(301:end));
hs2.Marker = 'none';
title('Autocorrelation Sequence of Ts2');
xlabel('Lag')
```



Even at a lag of 300, the autocorrelations are 0.94 and 0.96 respectively.

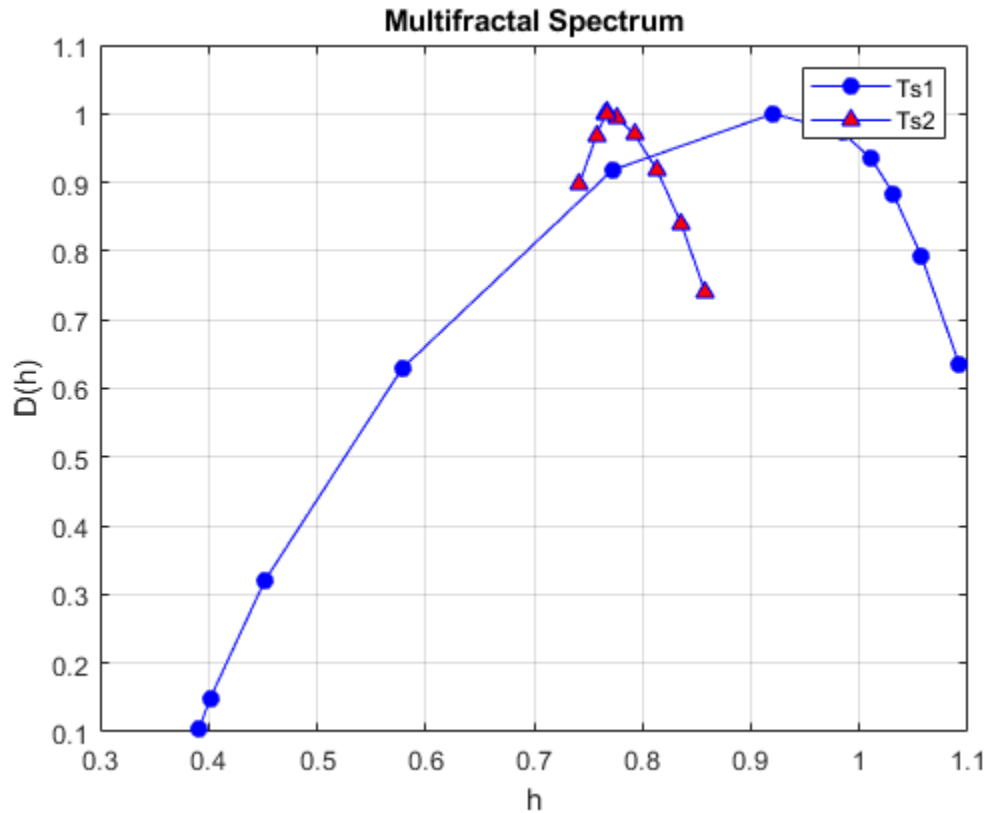
The fact that these signals are very different is revealed through fractal analysis. Compute and plot the multifractal spectra of the two signals. In multifractal analysis, discrete wavelet techniques based on the so-called *wavelet leaders* are the most robust.

```
[dh1,h1,cp1,tauq1] = dwtleader(Ts1);
[dh2,h2,cp2,tauq2] = dwtleader(Ts2);
figure;
hp = plot(h1,dh1,'b-o',h2,dh2,'b-^');
hp(1).MarkerFaceColor = 'b';
hp(2).MarkerFaceColor = 'r';
grid on;
xlabel('h'); ylabel('D(h)');
```

```

legend('Ts1','Ts2','Location','NorthEast');
title('Multifractal Spectrum');

```

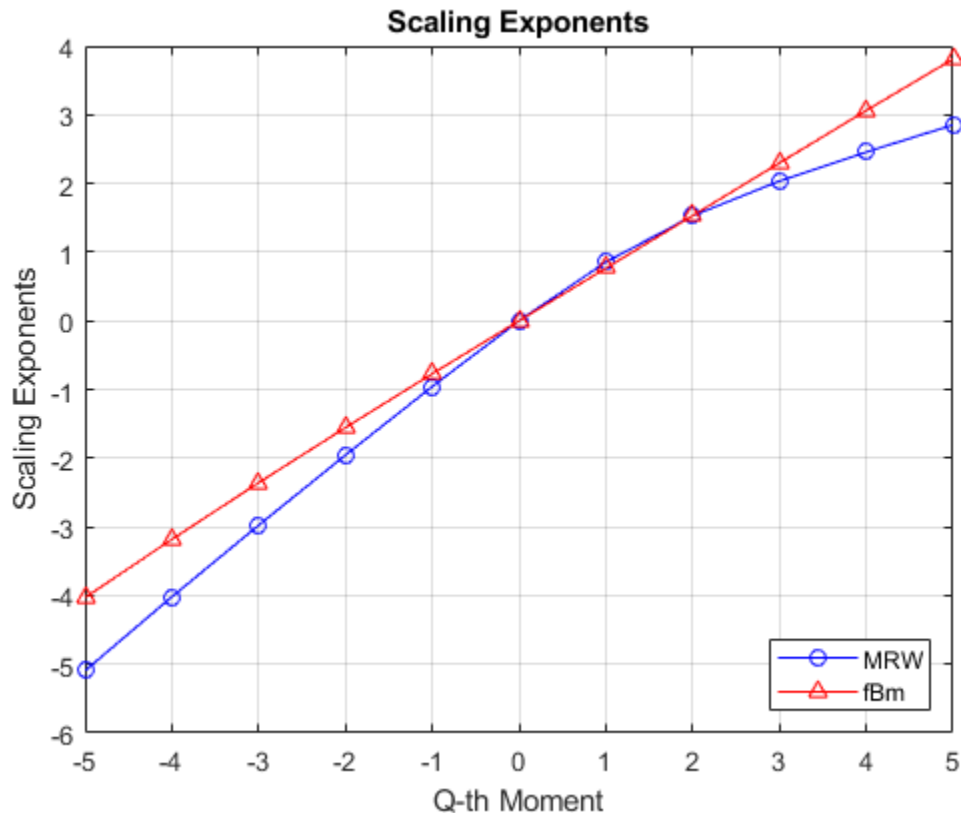


The multifractal spectrum effectively shows the distribution of scaling exponents for a signal. Equivalently, the multifractal spectrum provides a measure of how much the local regularity of a signal varies in time. A signal that is monofractal exhibits essentially the same regularity everywhere in time and therefore has a multifractal spectrum with narrow support. Conversely, A multifractal signal exhibits variations in signal regularity over time and has a multifractal spectrum with wider support. From the multifractal spectra shown here, Ts2, appears to be a monofractal signal characterized by a cluster of scaling exponents around 0.78. On the other hand, Ts1, demonstrates a wide-range of scaling exponents indicating that it is multifractal. Note the total range of scaling (Holder) exponents for Ts2 is just 0.14, while it is 4.6 times as big for Ts1. Ts2 is actually

an example of a monofractal fractional Brownian motion (fBm) process with a Holder exponent of 0.8 and Ts1 is a multifractal random walk.

You can also use the scaling exponent outputs from `dwtleader` along with the 2nd cumulant to help classify a process as monofractal vs. multifractal. Recall a monofractal process has a linear scaling law as a function of the statistical moments, while a multifractal process has a nonlinear scaling law. `dwtleader` uses the range of moments from -5 to 5 in estimating these scaling laws. A plot of the scaling exponents for the fBm and multifractal random walk (MRW) process shows the difference.

```
plot(-5:5,tauq1,'b-o',-5:5,tauq2,'r-^');  
grid on;  
xlabel('Q-th Moment'); ylabel('Scaling Exponents');  
title('Scaling Exponents');  
legend('MRW','fBm','Location','SouthEast');
```



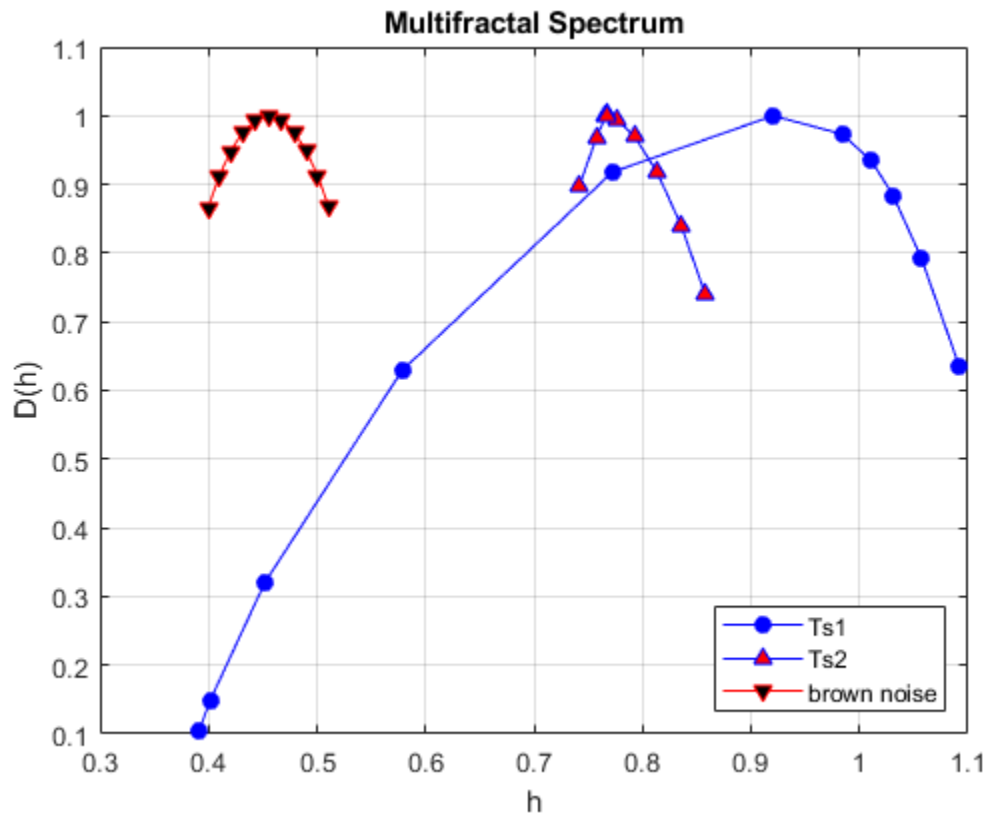
The scaling exponents for the fBm process are a linear function of the moments, while the exponents for the MRW process show a departure from linearity. The same information is summarized by the 1st, 2nd, and 3rd cumulants. The first cumulant is the estimate of the slope, in other words, it captures the linear behavior. The second cumulant captures the first departure from linearity. You can think of the second cumulant as the coefficients for a second-order (quadratic) term, while the third cumulant characterizes a more complicated departure of the scaling exponents from linearity. If you examine the 2nd and 3rd cumulants for the MRW process, they are 6 and 42 times as large as the corresponding cumulants for the fBm data. In the latter case, the 2nd and 3rd cumulants are almost zero as expected.

For comparison, add the multifractal spectrum for the brown noise computed in an earlier example.

```

hp = plot(h1,dh1, 'b-o',h2,dh2, 'b-^',hbrown,dhbrown, 'r-v');
hp(1).MarkerFaceColor = 'b';
hp(2).MarkerFaceColor = 'r';
hp(3).MarkerFaceColor = 'k';
grid on;
xlabel('h'); ylabel('D(h)');
legend('Ts1','Ts2','brown noise','Location','SouthEast');
title('Multifractal Spectrum');

```



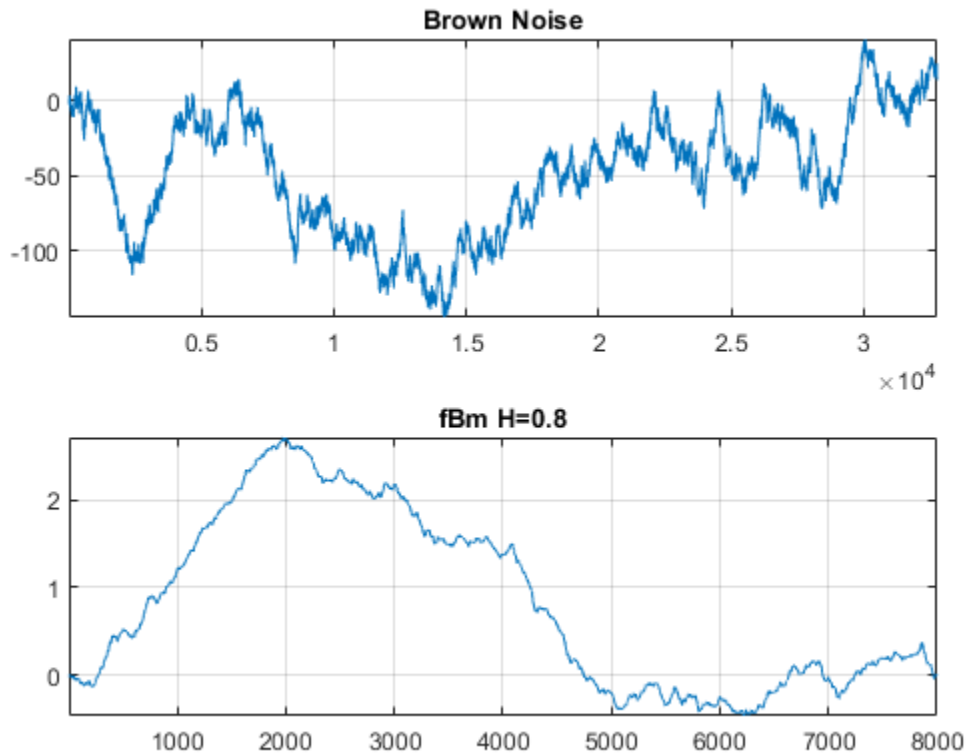
Where is the Process Going Next? Persistent and Antipersistent Behavior

Both the fractional Brownian process (Ts2) and the brown noise series are monofractal. However, a simple plot of the two time series shows that they appear quite different.

```

subplot(2,1,1)
plot(brownnoise); title('Brown Noise');
grid on; axis tight;
subplot(2,1,2)
plot(Ts2); title('fBm H=0.8'); grid on;
axis tight;

```

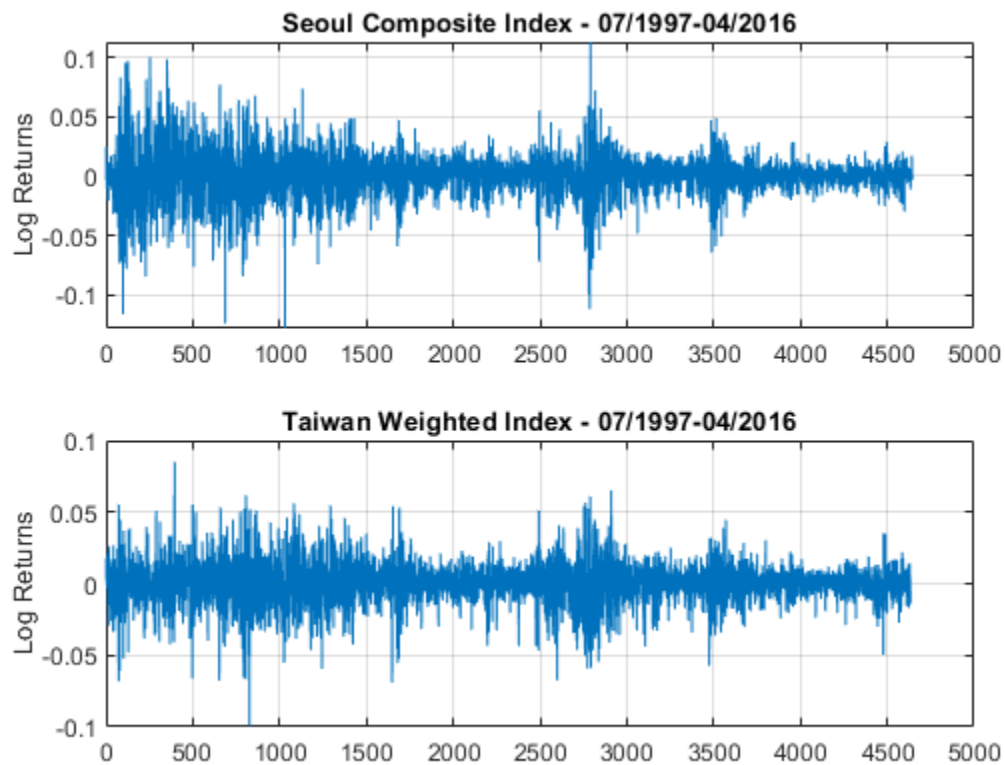


The fBm data is much smoother than the brown noise. Brown noise, also known as a random walk, has a theoretical Holder exponent of 0.5. This value forms a boundary between processes with Holder exponents, H , from $0 < H < 0.5$ and those with Holder exponents in the interval $0.5 < H < 1$. The former are called *antipersistent* and exhibit short memory. The latter are called *persistent* and exhibit long memory. In antipersistent time series, an increase in value at time t is followed with a decrease in value at time $t+1$ with

a high probability. Similarly, a decrease in value at time t is typically followed by an increase in value at time $t+1$. In other words, the time series tends to always revert to its mean value. In persistent time series, increases in value tend to be followed by subsequent increases while decreases in value tend to be followed by subsequent decreases.

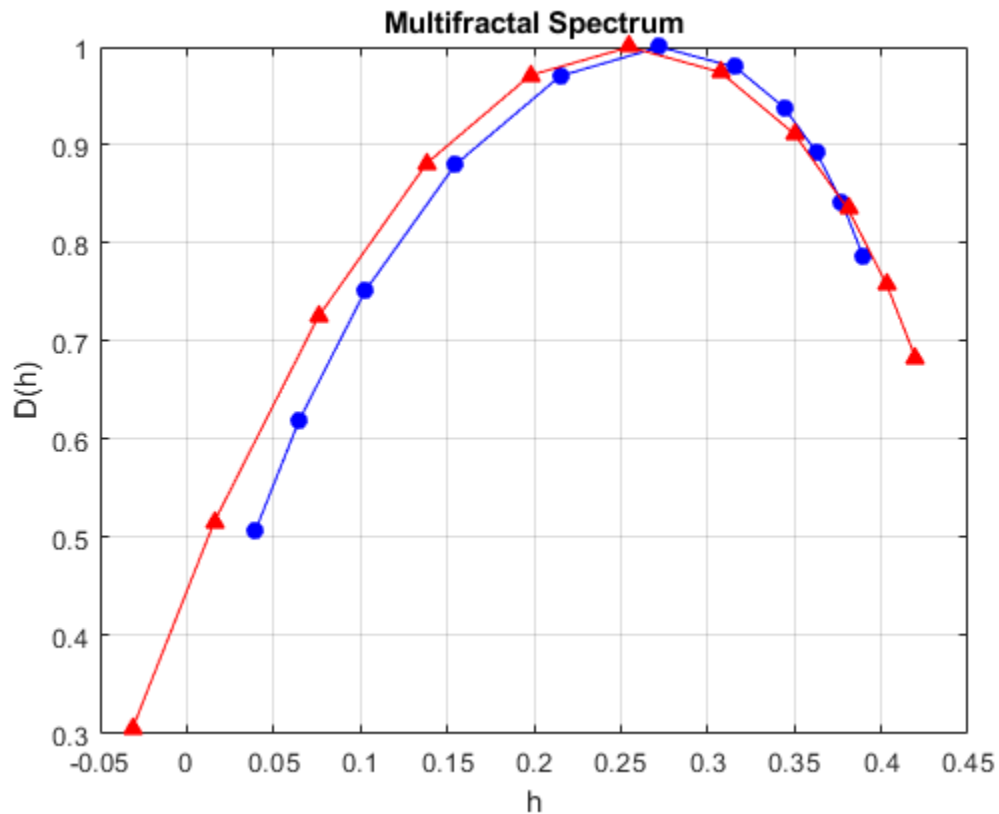
To see some real-world examples of antipersistent time series, load and analyze the daily log returns for the Taiwan Weighted and Seoul Composite stock indices. The daily returns for both indices cover the approximate period from July, 1997 through April, 2016.

```
load StockCompositeData;
subplot(2,1,1)
plot(SeoulComposite); title('Seoul Composite Index - 07/1997-04/2016');
ylabel('Log Returns'); grid on;
subplot(2,1,2);
plot(TaiwanWeighted); title('Taiwan Weighted Index - 07/1997-04/2016');
ylabel('Log Returns');
grid on;
```

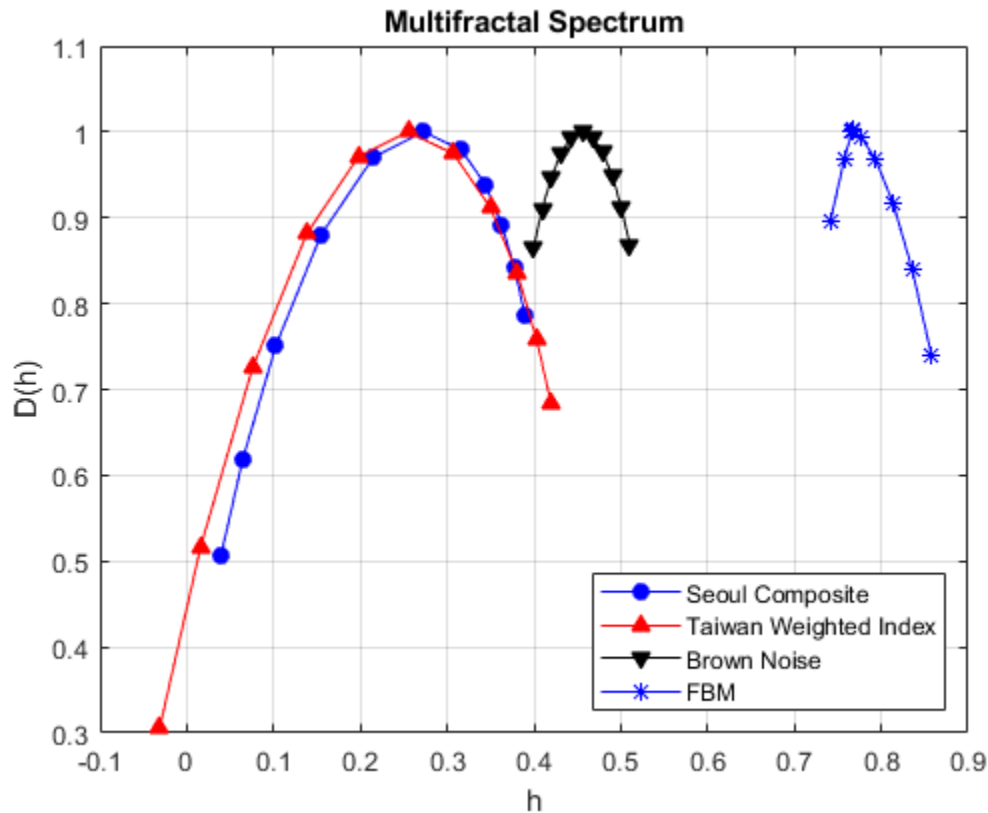
Obtain and plot the multifractal spectra of these two time series.

```
[dhseoul,hseoul,cpseoul] = dwtleader(SeoulComposite);  
[dhtaiwan,htaiwan,cptaiwan] = dwtleader(TaiwanWeighted);  
figure;  
plot(hseoul,dhseoul,'b-o','MarkerFaceColor','b');  
hold on;  
plot(htaiwan,dhtaiwan,'r-^','MarkerFaceColor','r');  
xlabel('h'); ylabel('D(h)'); grid on;  
title('Multifractal Spectrum');
```



From the multifractal spectrum, it is clear that both time series are antipersistent. For comparison, plot the multifractal spectra of the two financial time series along with the brown noise and fBm data shown earlier.

```
plot(hbrown,dhbrown,'k-v','MarkerFaceColor','k');
plot(h2,dh2,'b-*','MarkerFaceColor','b');
legend('Seoul Composite','Taiwan Weighted Index','Brown Noise','FBM',...
       'Location','SouthEast');
hold off;
```



Determining that a process is antipersistent or persistent is useful in predicting the future. For example, a time series with long memory that is increasing can be expected to continue increasing. While a time series that exhibits antipersistence can be expected to move in the opposite direction.

Measuring Fractal Dynamics of Heart Rate Variability

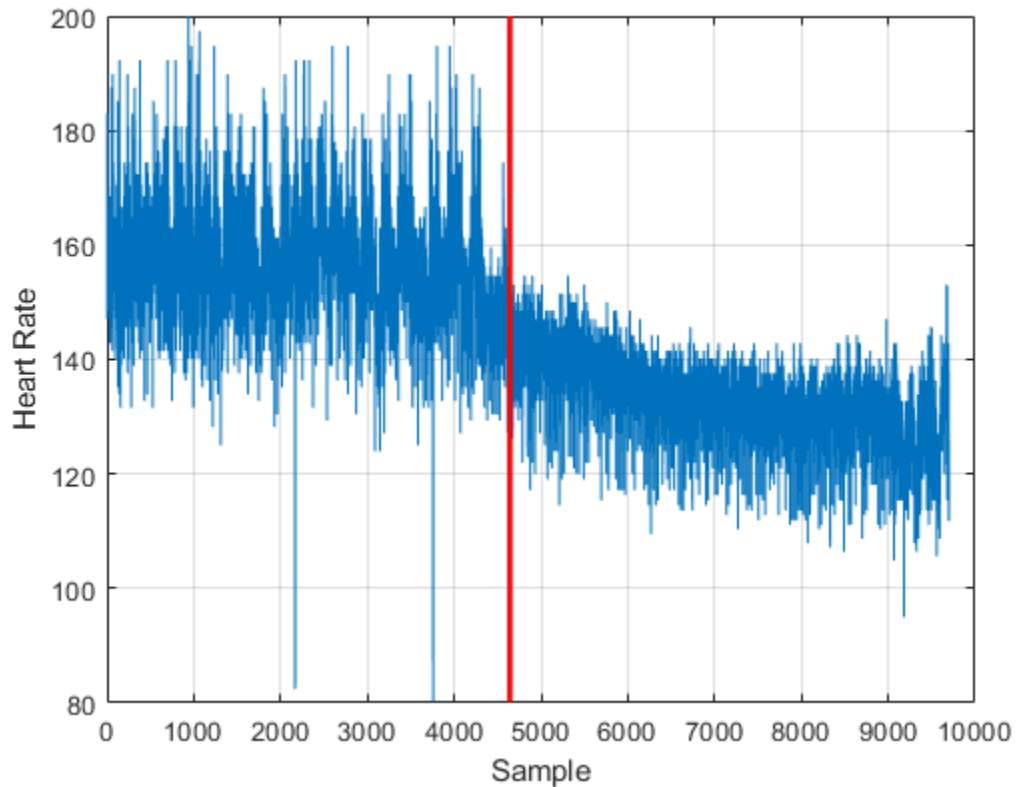
Normal human heart rate variability measured as RR intervals displays multifractal behavior. Further, reductions in this nonlinear scaling behavior are good predictors of cardiac disease and even mortality.

As an example of an induced change in the fractal dynamics of heart rate variability, consider a patient administered prostaglandin E1 due to a severe hypertensive episode.

The data is part of RHRV, an R-based software package for heart rate variability analysis. The authors have kindly granted permission for its use in this example.

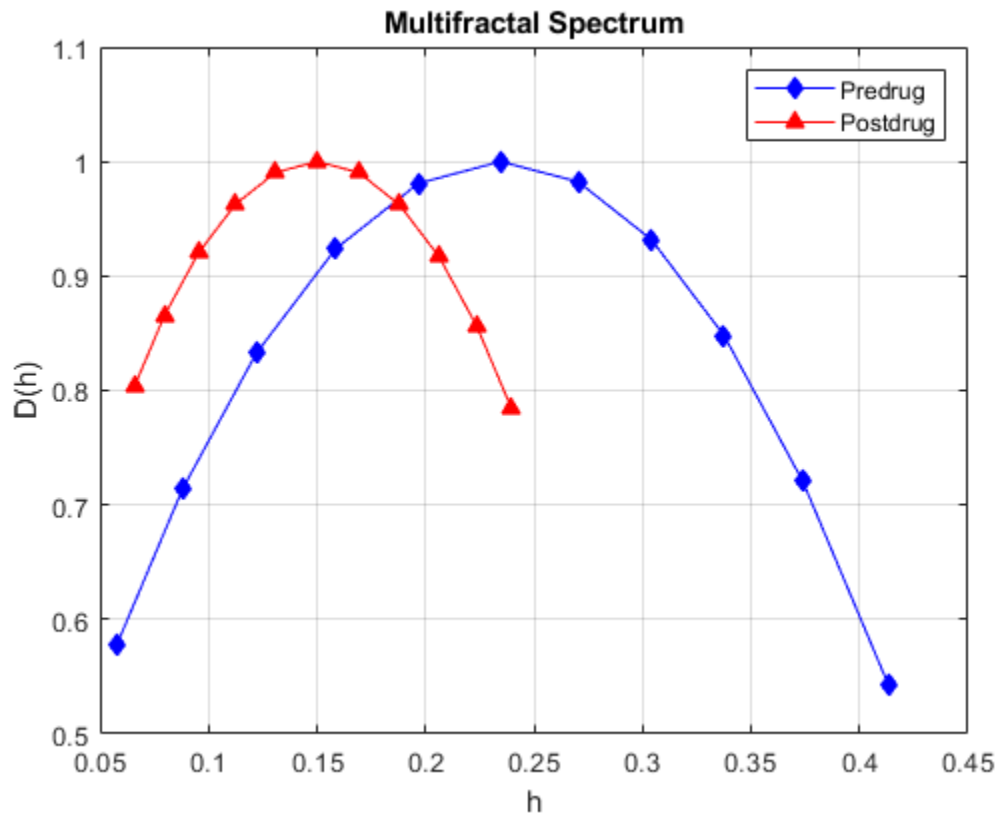
Load and plot the data. The vertical red line marks the beginning of the effect of the prostaglandin E1 on the heart rate and heart rate variability.

```
load hrvDrug;  
plot(hrvDrug); grid on;  
hold on;  
plot([4642 4642],[min(hrvDrug) max(hrvDrug)],'r','linewidth',2);  
hold off;  
ylabel('Heart Rate'); xlabel('Sample');
```



Split the data into pre-drug and post-drug data sets. Obtain and plot the multifractal spectra of the two time series.

```
predrug = hrvDrug(1:4642);
postdrug = hrvDrug(4643:end);
[dhpre,hpre] = dwtleader(predrug);
[dhpost,hpost] = dwtleader(postdrug);
figure;
hl = plot(hpre,dhpre,'b-d',hpost,dhpost,'r-^');
hl(1).MarkerFaceColor = 'b';
hl(2).MarkerFaceColor = 'r';
xlabel('h'); ylabel('D(h)');
grid on;
legend('Predrug','Postdrug');
title('Multifractal Spectrum'); xlabel('h'); ylabel('D(h)');
```



The induction of the drug has led to a 50% reduction in the width of the fractal spectrum. This indicates a significant reduction in the nonlinear dynamics of the heart as measured by heart rate variability. In this case, the reduction of the fractal dimension was part of a medical intervention. In a different context, studies on groups of healthy individuals and patients with congestive heart failure have shown that differences in the multifractal spectra can differentiate these groups. Specifically, significant reductions in the width of the multifractal spectrum is a marker of cardiac dysfunction.

References

L. Rodriguez-Linares, L., A.J. Mendez, M.J. Lado, D.N. Olivieri, X.A. Vila, and I. Gomez-Conde, "An open source tool for heart rate variability spectral analysis", *Computer Methods and Programs in Biomedicine*, 103(1):39-50,2011.

Wendt, H. and Abry, P. "Multifractality tests using bootstrapped wavelet leaders", IEEE Trans. Signal Processing, vol. 55, no. 10, pp. 4811-4820, 2007.

Wendt, H., Abry, P., and Jaffard, S. "Bootstrap for empirical multifractal analysis", IEEE Signal Processing Magazine, 24, 4, 38-48, 2007.

Jaffard, S., Lashermes, B., and Abry, P. "Wavelet leaders in multifractal analysis". In T. Qian, M.I. Vai and X. Yuesheng, editors. Wavelet Analysis and Applications, pp. 219-264, Birkhauser, 2006.

Wavelet Analysis of Financial Data

This example shows how to use wavelets to analyze financial data.

The separation of aggregate data into different time scales is a powerful tool for the analysis of financial data. Different market forces effect economic relationships over varying periods of time. Economic shocks are localized in time and within that time period exhibit oscillations of varying frequency.

Some economic indicators lag, lead, or are coincident with other variables. Different actors in financial markets view market mechanics over shorter and longer scales. Terms like "short-run" and "long-run" are central in modeling the complex relationships between financial variables.

Wavelets decompose time series data into different scales and can reveal relationships not obvious in the aggregate data. Further, it is often possible to exploit properties of the wavelet coefficients to derive scale-based estimators for variance and correlation and test for significant differences.

Maximal Overlap Discrete Wavelet Transform -- Volatility by Scale

There are a number of different variations of the wavelet transform. This example focuses on the maximal overlap discrete wavelet transform (MODWT). The MODWT is an undecimated wavelet transform over dyadic (powers of two) scales, which is frequently used with financial data. One nice feature of the MODWT for time series analysis is that it partitions the data variance by scale. To illustrate this, consider the quarterly chain-weighted U.S. real GDP data for 1947Q1 to 2011Q4. The data were transformed by first taking the natural logarithm and then calculating the year-over-year difference. Obtain the MODWT of the real GDP data down to level six with the 'db2' wavelet. Examine the variance of the data and compare that to the variances by scale obtained with the MODWT.

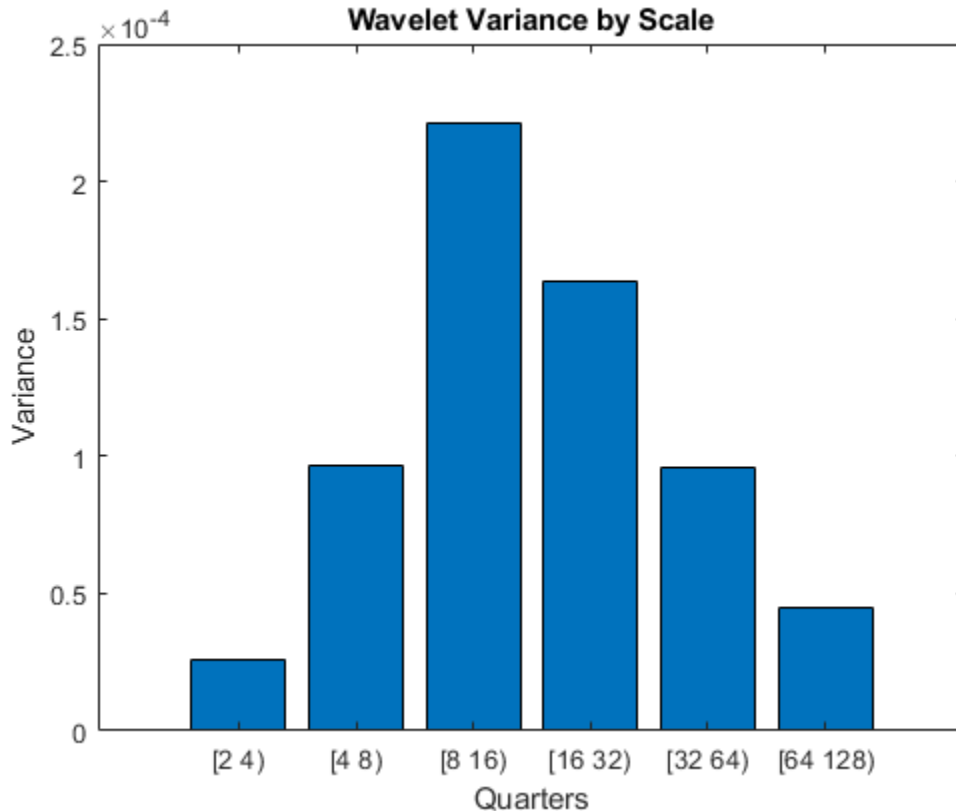
```
load GDPExampleData;  
realgdpwt = modwt(realgdp, 'db2', 6);  
vardata = var(realgdp, 1);  
varwt = var(realgdpwt, 1, 2);
```

In `vardata` you have the variance for the aggregate GDP time series. In `varwt` you have the variance by scale for the MODWT. There are seven elements in `varwt` because you obtained the MODWT down to level six resulting in six wavelet coefficient variances and one scaling coefficient variance. Sum the variances by scale to see that the variance is preserved. Plot the wavelet variances by scale ignoring the scaling coefficient variance.


```

totalMODWTvar = sum(varwt);
bar(varwt(1:end-1,:))
AX = gca;
AX.XTickLabels = {'[2 4)', '[4 8)', '[8 16)', '[16 32)', '[32 64)', '[64 128)'};
xlabel('Quarters')
ylabel('Variance')
title('Wavelet Variance by Scale')

```



Because this data is quarterly, the first scale captures variations between two and four quarters, the second scale between four and eight, the third between 8 and 16, and so on.

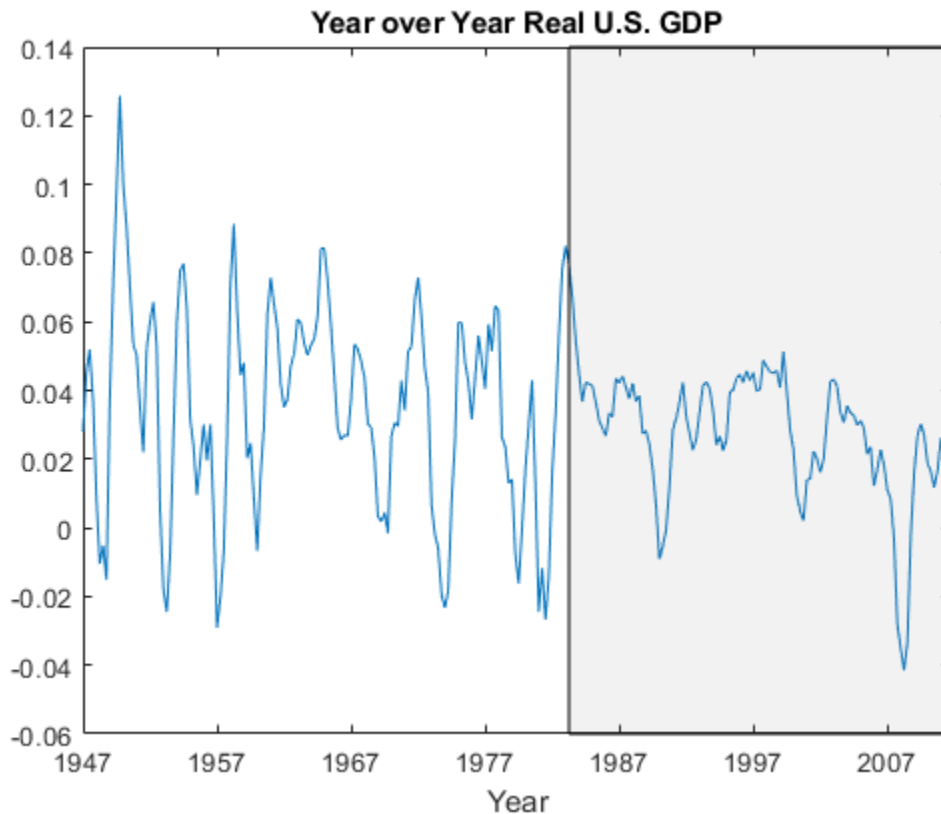
From the MODWT and a simple bar plot, you see that cycles in the data between 8 and 32 quarters account for the largest variance in the GDP data. If you consider the wavelet variances at these scales, they account for 57% of the variability in the GDP data. This

means that oscillations in the GDP over a period of 2 to 8 years account for most of the variability seen in the time series.

Great Moderation -- Testing for Changes in Volatility with the MODWT

Wavelet analysis can often reveal changes in volatility not evident in aggregate data. Begin with a plot of the GDP data.

```
helperFinancialDataExample1(realgdp, year, 'Year over Year Real U.S. GDP')
```



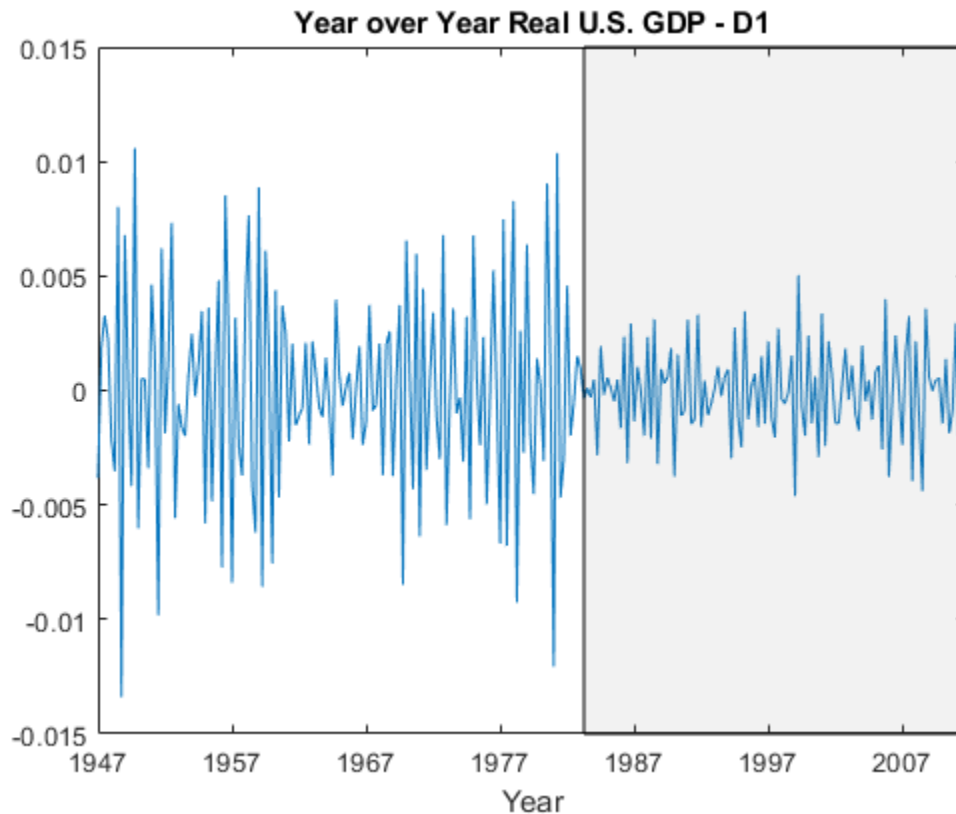
The shaded region is referred to as the "Great Moderation" signifying a period of decreased macroeconomic volatility in the U.S. beginning in the mid 1980s.

Examining the aggregate data, it is not clear that there is in fact reduced volatility in this period. Use wavelets to investigate this by first obtaining a multiresolution analysis of the real GDP data using the 'db2' wavelet down to level 6.

```
realgdpwt = modwt(realgdp, 'db2', 6, 'reflection');  
gdpmra = modwtmra(realgdpwt, 'db2', 'reflection');
```

Plot the level-one details, D1. These details capture oscillations in the data between two and four quarters in duration.

```
helperFinancialDataExample1(gdpmra(1,:), year, ...  
    'Year over Year Real U.S. GDP - D1')
```



Examining the level-one details, it appears there is a reduction of variance in the period of the Great Moderation.

Test the level-one wavelet coefficients for significant variance changepts.

```
[pts_0pt,kopt,t_est] = wvarchg(realgdpwt(1,1:numel(realgdp)),2);
years(pts_0pt)
```

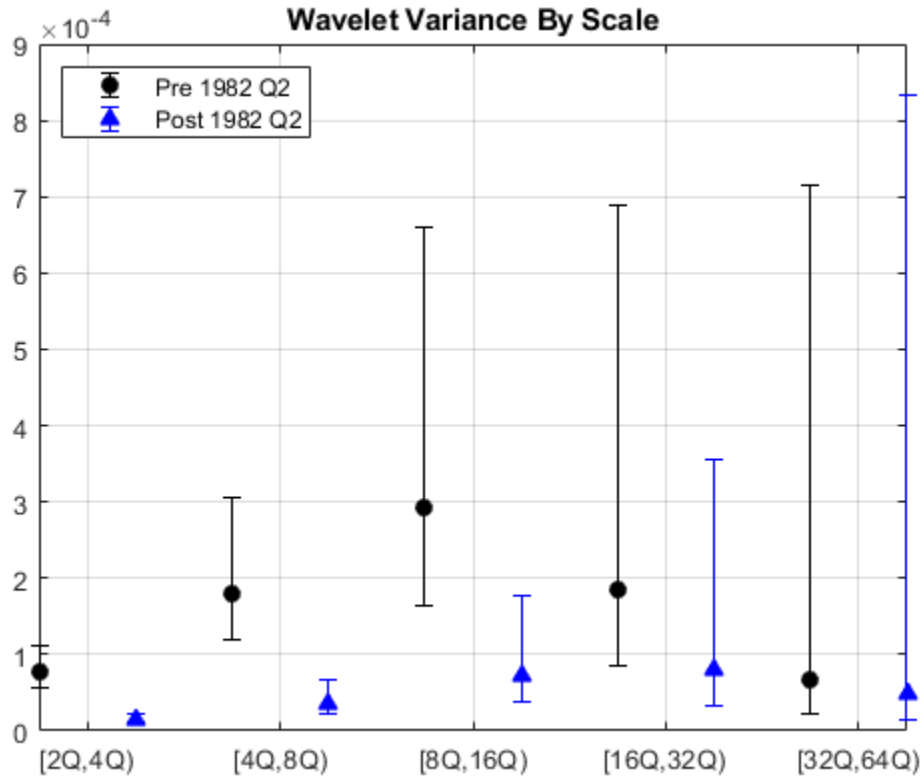
```
ans =
```

```
duration
142 yrs
```

There is a variance changepoint identified in 1982. This example does not correct for the delay introduced by the 'db2' wavelet at level one. However, that delay is only two samples so it does not appreciably affect the results.

To assess changes in the volatility of the GDP data pre and post 1982, split the original data into pre- and post-changept series. Obtain the wavelet transforms of the pre and post datasets. In this case, the series are relatively short so use the Haar wavelet to minimize the number of boundary coefficients. Compute unbiased estimates of the wavelet variance by scale and plot the result.

```
tspre = realgdp(1:pts_0pt);
tspost = realgdp(pts_0pt+1:end);
wtpre = modwt(tspre,'haar',5);
wtpost = modwt(tspost,'haar',5);
prevar = modwtvar(wtpre,'haar','table');
postvar = modwtvar(wtpost,'haar','table');
xlab = {'[20,40)', '[40,80)', '[80,160)', '[160,320)', '[320,640)'};
helperFinancialDataExampleVariancePlot(prevar,postvar,'table',xlab)
title('Wavelet Variance By Scale');
legend('Pre 1982 Q2', 'Post 1982 Q2', 'Location', 'NorthWest');
```

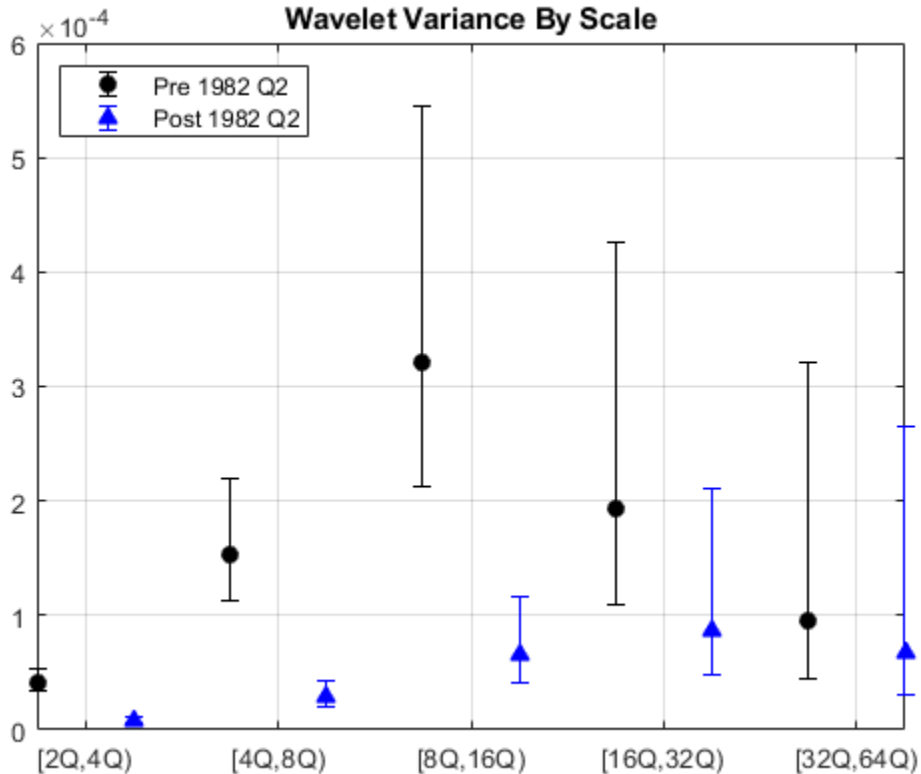


From the preceding plot, it appears there are significant differences between the pre-1982Q2 and post-1982Q2 variances at scales between 2 and 16 quarters.

Because the time series are so short in this example, it can be useful to use biased estimates of the variance. Biased estimates do not remove boundary coefficients. Use a 'db2' wavelet filter with four coefficients.

```
wtpre = modwt(tspre, 'db2', 5, 'reflection');
wtpost = modwt(tspost, 'db2', 5, 'reflection');
prevar = modwtvar(wtpre, 'db2', 0.95, 'EstimatorType', 'biased', 'table');
postvar = modwtvar(wtpost, 'db2', 0.95, 'EstimatorType', 'biased', 'table');
xlab = {'[2Q,4Q)', '[4Q,8Q)', '[8Q,16Q)', '[16Q,32Q)', '[32Q,64Q)'};
figure;
helperFinancialDataExampleVariancePlot(prevar, postvar, 'table', xlab)
```

```
title('Wavelet Variance By Scale');
legend('Pre 1982 Q2','Post 1982 Q2','Location','NorthWest');
```



The results confirm our original finding that the Great Moderation is manifested in volatility reductions over scales from 2 to 16 quarters.

Wavelet Correlation Analysis of GDP Component Data

You can also use wavelets to analyze correlation between two datasets by scale. Examine the correlation between the aggregate data on government spending and private investment. The data cover the same period as the real GDP data and are transformed in the exact same way.

```
[rho,pval] = corrcoef(privateinvest,govtexp);
```

Government spending and personal investment demonstrate a weak, but statistically significant, negative correlation of -0.215. Repeat this analysis using the MODWT.

```
wtPI = modwt(privateinvest, 'db2', 5, 'reflection');
wtGE = modwt(govtexp, 'db2', 5, 'reflection');
wcorrtable = modwtcorr(wtPI, wtGE, 'db2', 0.95, 'reflection', 'table');
display(wcorrtable)
```

wcorrtable =

6x6 table

	NJ	Lower	Rho	Upper	Pvalue	AdjustedPvalue
D1	257	-0.29187	-0.12602	0.047192	0.1531	0.7502
D2	251	-0.54836	-0.35147	-0.11766	0.0040933	0.060171
D3	239	-0.62443	-0.35248	-0.0043207	0.047857	0.35175
D4	215	-0.70466	-0.32112	0.20764	0.22523	0.82773
D5	167	-0.63284	0.12965	0.76448	0.75962	1
S5	167	-0.63428	0.12728	0.76347	0.76392	1

The multiscale correlation available with the MODWT shows a significant negative correlation only at scale 2, which corresponds to cycles in the data between 4 and 8 quarters. Even this correlation is only marginally significant when adjusting for multiple comparisons.

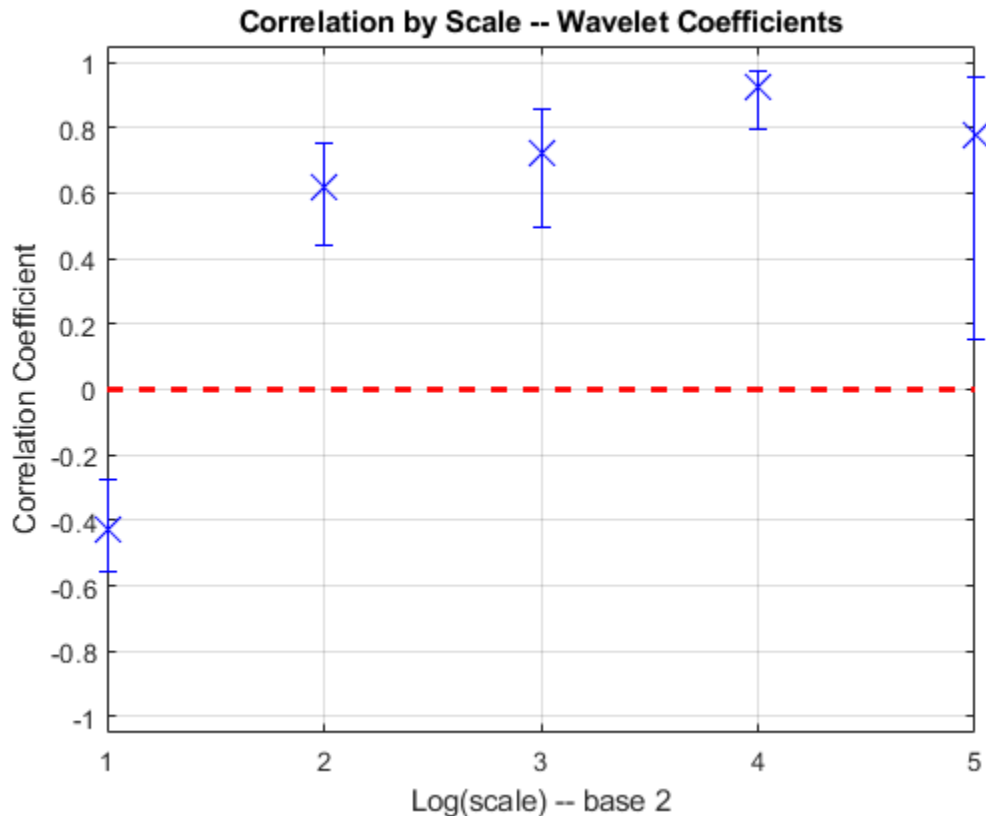
The multiscale correlation analysis reveals that the slight negative correlation in the aggregate data is driven by the behavior of the data over scales of four to eight quarters. When you consider the data over different time periods (scales), there is no significant correlation.

Wavelet Cross-Correlation Sequences -- Leading and Lagging Variables

With financial data, there is often a leading or lagging relationship between variables. In those cases, it is useful to examine the cross-correlation sequence to determine if lagging one variable with respect to another maximizes their cross-correlation. To illustrate this, consider the correlation between two components of the GDP -- personal consumption expenditures and gross private domestic investment.

```
piwt = modwt(privateinvest, 'fk8', 5);
pcwt = modwt(pc, 'fk8', 5);
```

```
figure;
modwtcorr(piwt,pcwt,'fk8')
```

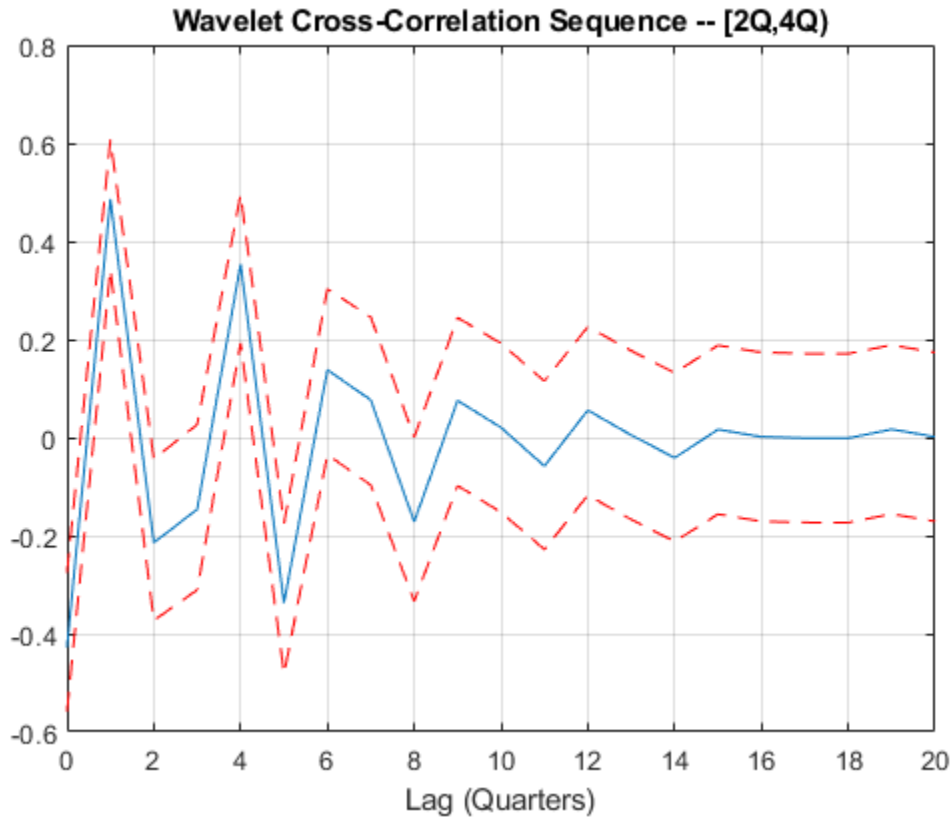


Personal expenditure and personal investment are negatively correlated over a period of 2-4 quarters. At longer scales, there is a strong positive correlation between personal expenditure and personal investment. Examine the wavelet cross-correlation sequence at the scale representing 2-4 quarter cycles.

```
[xcseq,xcseqci,lags] = modwtcorr(piwt,pcwt,'fk8');
zerolag = floor(numel(xcseq{1})/2)+1;
plot(lags{1}(zerolag:zerolag+20),xcseq{1}(zerolag:zerolag+20));
hold on;
plot(lags{1}(zerolag:zerolag+20),xcseqci{1}(zerolag:zerolag+20,:), 'r--');
xlabel('Lag (Quarters)');
```



```
grid on;
title('Wavelet Cross-Correlation Sequence -- [2Q,4Q]');
```



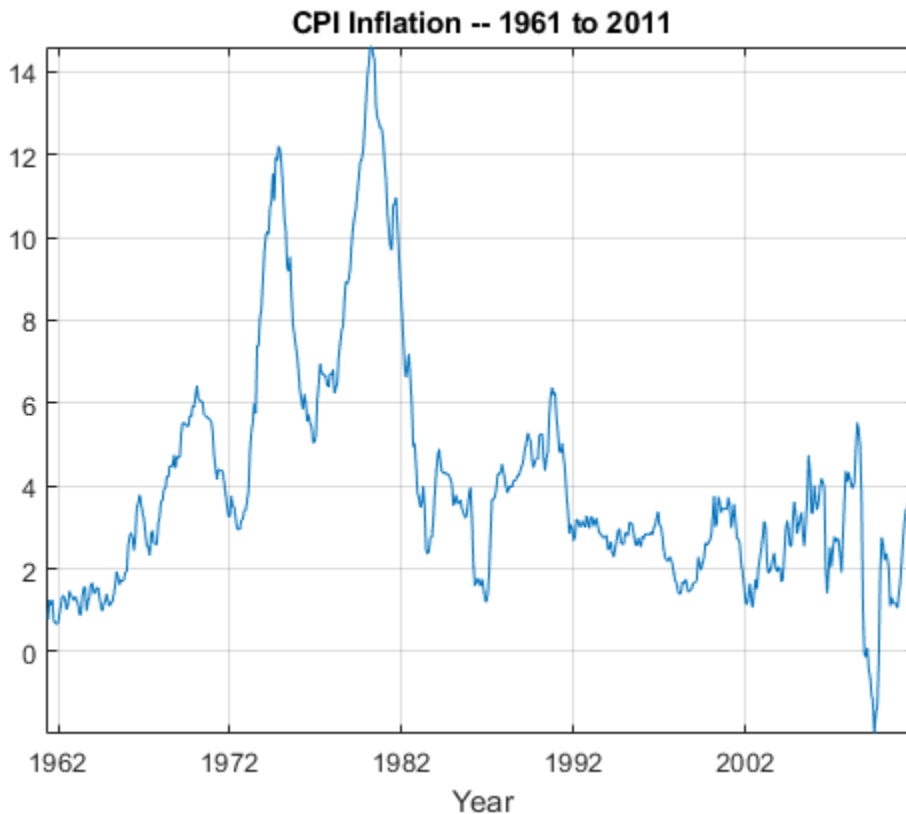
The finest-scale wavelet cross-correlation sequence shows a peak positive correlation at a lag of one quarter. This indicates that personal investment lags personal expenditures by one quarter.

Continuous Wavelet Analysis of U.S. Inflation Rate

Using discrete wavelet analysis, you are limited to dyadic scales. This limitation is removed when using continuous wavelet analysis.

Load U.S. inflation rate data from May, 1961 to November, 2011.

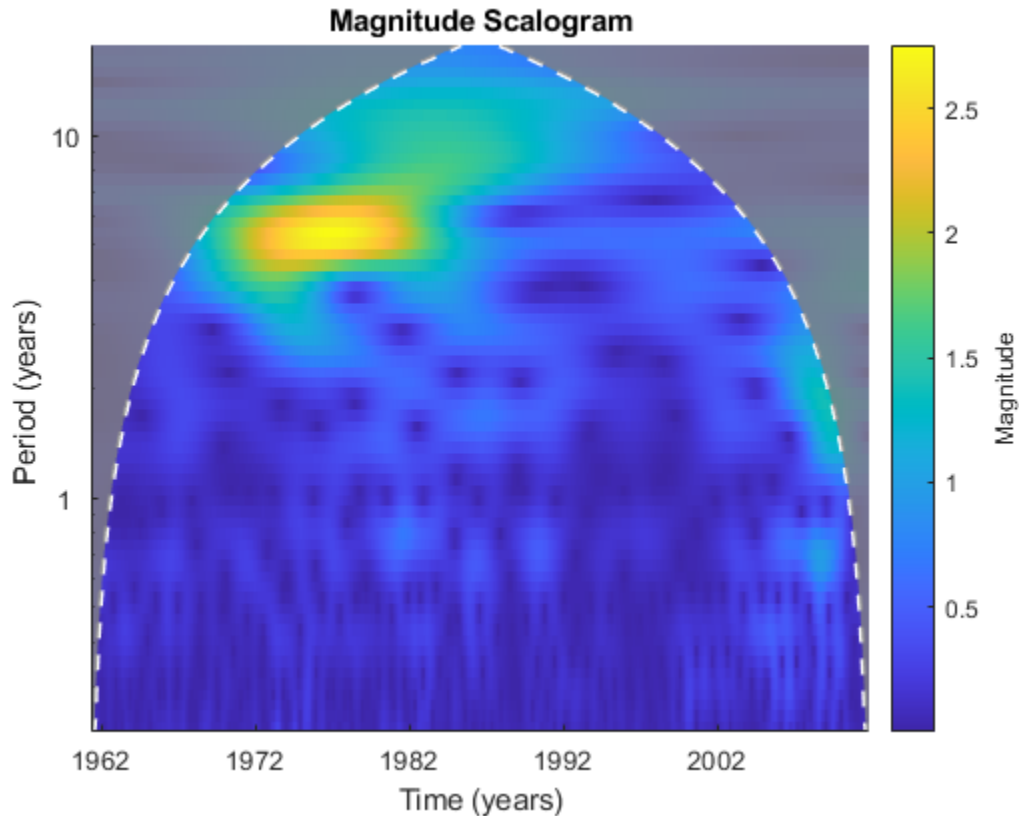
```
load CPIInflation;  
figure  
plot(yr,inflation)  
AX = gca;  
AX.XTick = 1962:10:2011;  
title('CPI Inflation -- 1961 to 2011')  
axis tight; grid on;  
xlabel('Year')
```



In the time data, a slow oscillation appears in the early 1970s and seems to dissipate by the late 1980s.

To characterize the periods of increased volatility, obtain the continuous wavelet transform (CWT) of the data using the analytic Morlet wavelet.

```
cwt(inflation, 'amor', years(1/12));  
AX = gca;  
AX.XTick = 8/12:10:596/12;  
AX.XTickLabels = yr(round(AX.XTick*12));
```



The CWT reveals the strongest oscillations in the inflation rate data in the approximate range of 4-6 years. This volatility begins to dissipate by the mid 1980s and is characterized by both a gradual reduction in inflation and a shift in volatility toward longer periods. The strong volatility cycles in the 1970s and into the early 1980s are a result of the 1970s energy crisis (oil shocks) which resulted in stagflation (stagnant growth and inflation) in the major industrial economies. See [1] for an in-depth CWT-based analysis of these and other macroeconomic data. This example reproduces a small part of the broader and more detailed analysis in that paper.

Conclusions

In this example you learned how to use the MODWT to analyze multiscale volatility and correlation in financial time series data. The example also demonstrated how wavelets can be used to detect changes in the volatility of a process over time. Finally, the example showed how the CWT can be used to characterize periods of increased volatility in financial time series. The references provide more detail on wavelet applications for financial data and time series analysis.

Appendix

The following helper functions are used in this example.

*helperFinancialDataExample1




*helperFinancialDataExampleVariancePlot




*helperCWTTimeFreqPlot

Time-Frequency Analysis

Time-Frequency Gallery

This gallery provides you with an overview of the time-frequency analysis features available in the Signal Processing Toolbox and Wavelet Toolbox. The descriptions and usage examples present various methods that you can use for your signal analysis.

Method	Features	Invertible	Examples
<p>“Short-Time Fourier Transform (Spectrogram)” on page 4-4</p>	<ul style="list-style-type: none"> The short-time Fourier transform (STFT) has fixed time-frequency resolution. The spectrogram is the magnitude squared of STFT. 	<ul style="list-style-type: none"> stft: Yes spectrogram: No 	<p>“Example: Whale Song” on page 4-7</p> 
<p>“Continuous Wavelet Transform (Scalogram)” on page 4-9</p>	<ul style="list-style-type: none"> The continuous wavelet transform (CWT) has a variable time-frequency resolution. The CWT preserves time shifts and time scalings. 	Yes	<p>“Example: ECG Signal” on page 4-10</p> 
<p>“Wigner-Ville Distribution” on page 4-11</p>	<ul style="list-style-type: none"> The Wigner-Ville distribution (WVD) is always real. Time and frequency marginals correspond to power and spectral energy density. Time resolution of the WVD is equal to the number of input samples. 	No	<p>“Example: Otoacoustic Emission” on page 4-12</p> 

Method	Features	Invertible	Examples
“Reassignment and Synchrosqueezing” on page 4-13	<ul style="list-style-type: none"> • Reassignment sharpens localization of spectral estimates. • Synchrosqueezing “condenses” time-frequency maps around curves of instantaneous frequency. • Both methods are especially suited to track and extract time-frequency ridges 	<ul style="list-style-type: none"> • pspectrum: No • fsst, wsst: Yes 	<p>“Example: Echolocation Pulse” on page 4-15</p> 
“Constant-Q Gabor Transform” on page 4-22	<ul style="list-style-type: none"> • The constant-Q Gabor transform (CQT) tiles the time-frequency plane with variable-sized windows. • The windows have adaptable bandwidth and sampling density. • The ratio of center frequency to bandwidth (Q-factor) for all windows is constant. 	Yes	<p>“Example: Rock Music” on page 4-23</p> 
“Empirical Mode Decomposition and Hilbert-Huang Transform” on page 4-24	<ul style="list-style-type: none"> • Empirical mode decomposition (EMD) decomposes signals into intrinsic mode functions. • The Hilbert-Huang transform (HHT) computes the instantaneous frequency of each empirical mode. 	No	<p>“Example: Bearing Vibration” on page 4-25</p> 

Short-Time Fourier Transform (Spectrogram)

Description

- The short-time Fourier transform is a linear time-frequency representation useful in the analysis of nonstationary multicomponent signals.
- The short-time Fourier transform is invertible.
- The spectrogram is the magnitude squared of the STFT.
- You can compute the cross-spectrogram of two signals to look for similarities in time-frequency space.
- The *persistence spectrum* of a signal is a time-frequency view that shows the percentage of the time that a given frequency is present in a signal. The persistence spectrum is a histogram in power-frequency space. The longer a particular frequency persists in a signal as the signal evolves, the higher its time percentage and thus the brighter or "hotter" its color in the display.

Potential Applications

The applications of this time-frequency method include, but are not limited to:

- *Audio signal processing*: Fundamental frequency estimation, cross synthesis, spectral envelope extraction, time-scale modification, time-stretching, and pitch shifting. (See "Phase Vocoder with Different Synthesis and Analysis Windows" (Signal Processing Toolbox) for more details.)
- *Crack detection*: Detect cracks in aluminum plates using dispersion curves of ultrasonic Lamb waves.
- *Sensor array processing*: Sonar exploration, geophysical exploration, and beamforming.
- *Digital communications*: Detection of frequency hopping signal.

How to Use

- `stft` computes the short-time Fourier transform. To invert the short-time Fourier transform, use the `istft` function.
- `pspectrum` or `spectrogram` computes the spectrogram.
- `xspectrogram` computes the cross spectrogram of two signals.
- You can also use the spectrogram view in **Signal Analyzer** to view the spectrogram of a signal.

- Use the persistence spectrum option in `pspectrum` or **Signal Analyzer** to identify signals hidden in other signals.

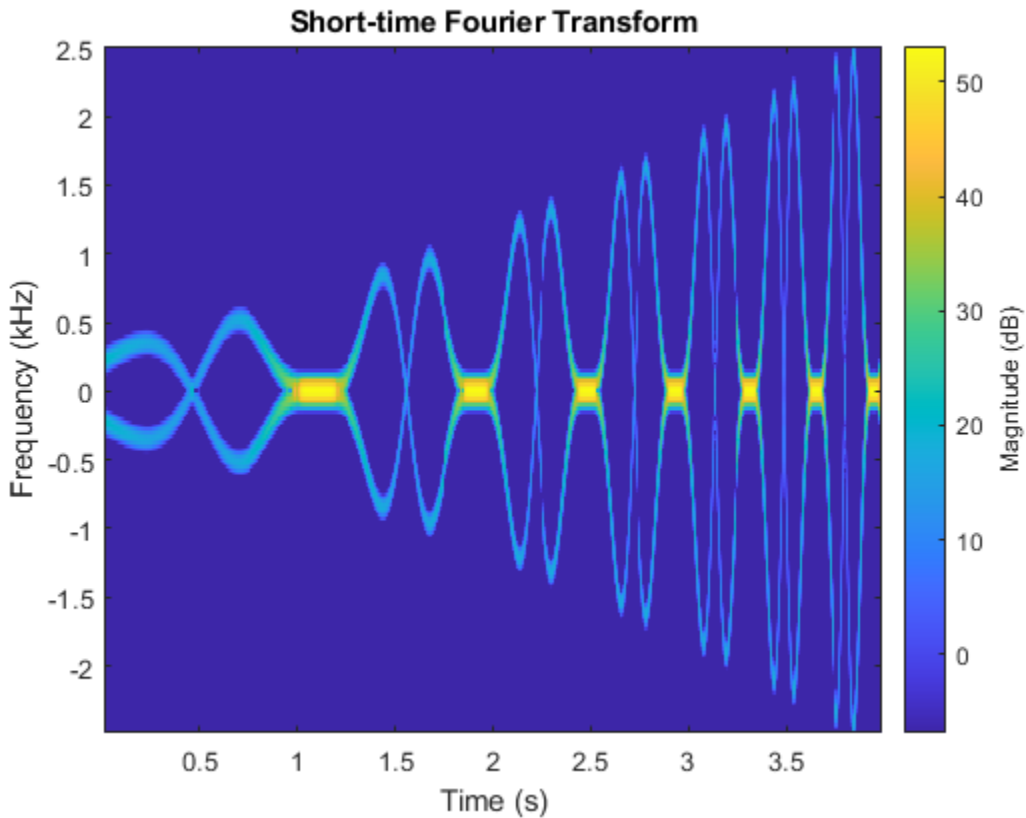
Example: Pulses and Oscillations

Generate a signal sampled at 5 kHz for 4 seconds. The signal consists of a set of pulses of decreasing duration separated by regions of oscillating amplitude and fluctuating frequency with an increasing trend.

```
fs = 5000;  
t = 0:1/fs:4-1/fs;  
  
x = 10*besselj(0,1000*(sin(2*pi*(t+2).^3/60).^5));
```

Compute and plot the short-time Fourier transform of the signal. Window the signal with a 200-sample Kaiser window with shape factor $\beta = 30$.

```
stft(x,fs,'Window',kaiser(200,30))
```



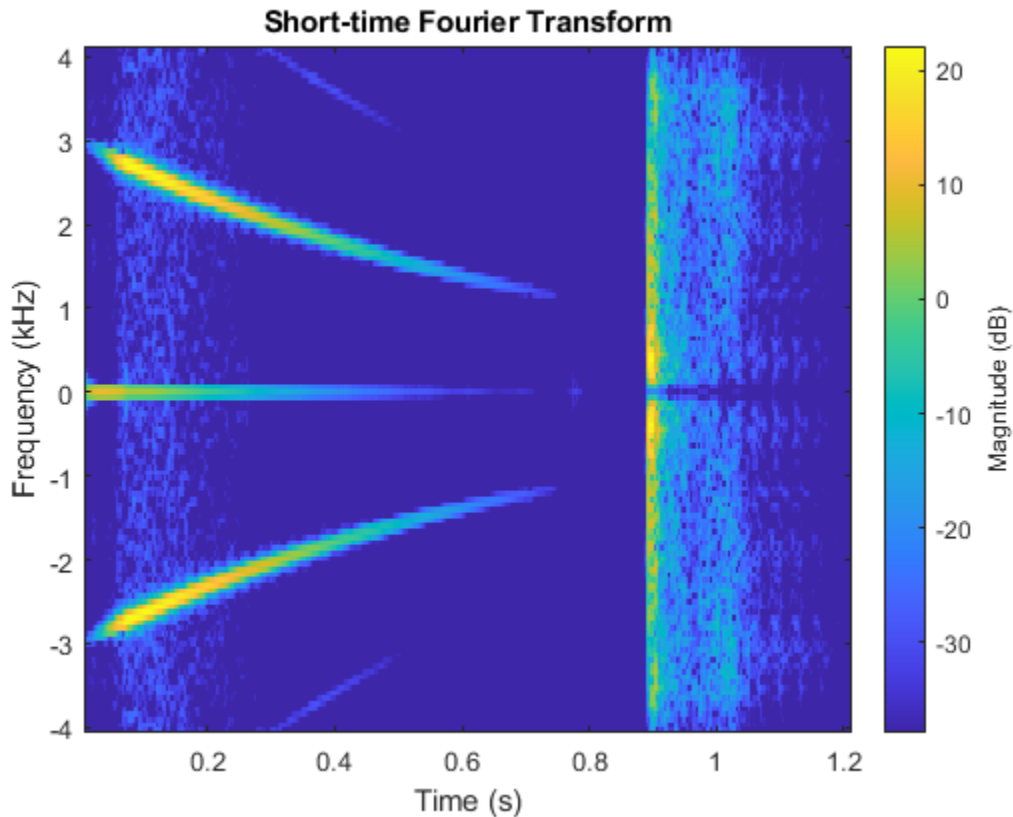
Example: Audio Signal with Decreasing Chirps

Load an audio signal that contains two decreasing chirps and a wideband splatter sound.

```
load splat
```

Set the overlap length to 96 samples. Plot the short-time Fourier transform.

```
stft(y,Fs,'OverlapLength',96)
```



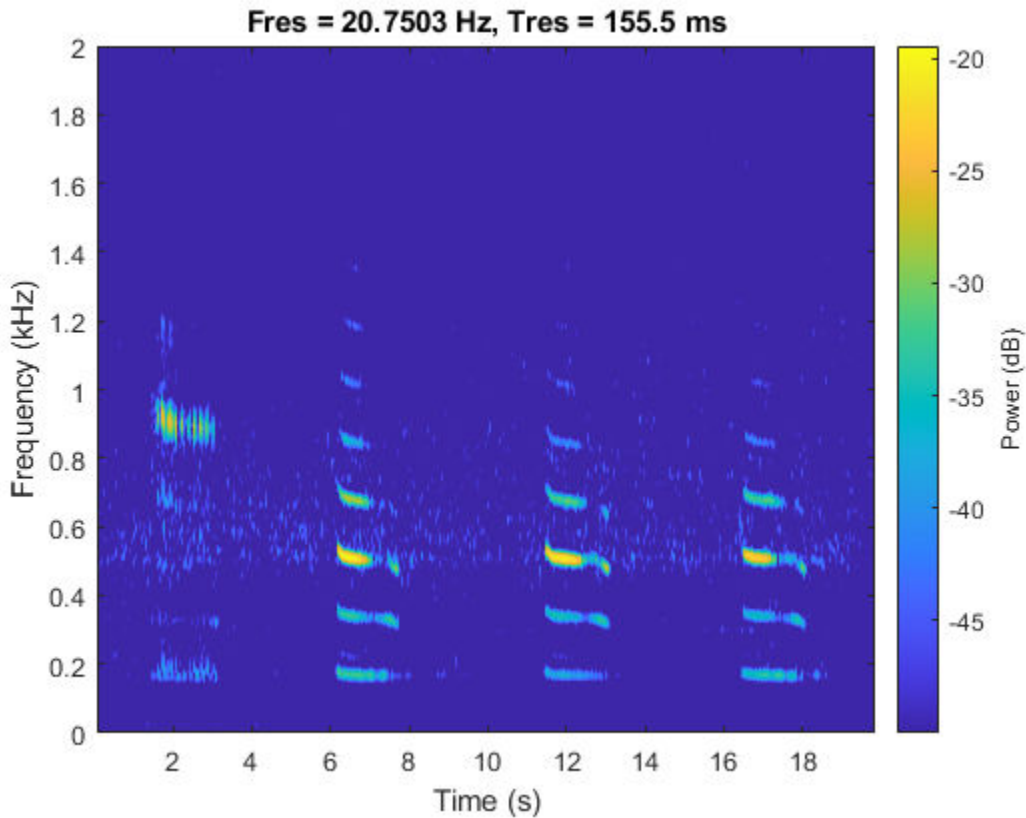
Example: Whale Song

Load a file that contains audio data from a Pacific blue whale, sampled at 4 kHz. The file is from the library of animal vocalizations maintained by the Cornell University Bioacoustics Research Program. The time scale in the data is compressed by a factor of 10 to raise the pitch and make the calls more audible.

```
whaleFile = fullfile(matlabroot, 'examples', 'matlab', 'bluewhale.au');
[w, fs] = audioread(whaleFile);
```

Compute the spectrogram of the whale song with an overlap percentage equal to eighty percent. Set the minimum threshold for the spectrogram to -50 dB.

```
pspectrum(w, fs, 'spectrogram', 'Leakage', 0.2, 'OverlapPercent', 80, 'MinThreshold', -50)
```



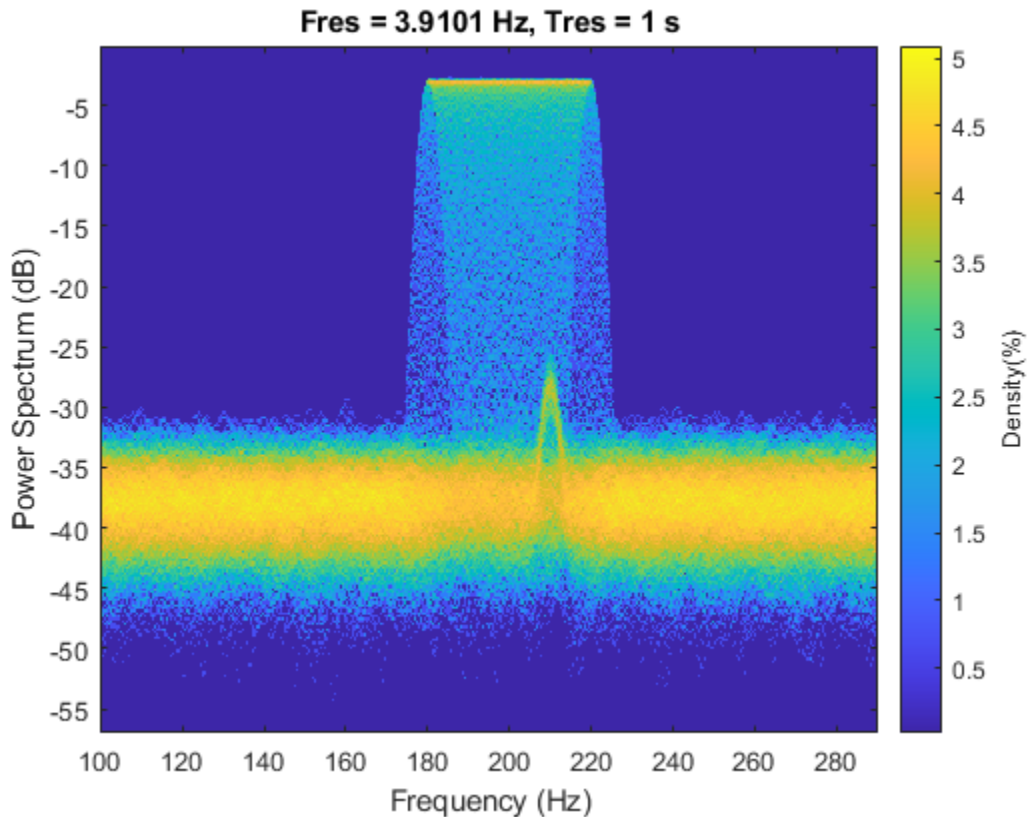
Example: Persistence Spectrum of Transient Signal

Load an interference narrowband signal embedded within a broadband signal.

```
load TransientSig
```

Compute the persistence spectrum of the signal. Both signal components are clearly visible.

```
pspectrum(x,fs,'persistence',...  
         'FrequencyLimits',[100 290],'TimeResolution',1)
```



Continuous Wavelet Transform (Scalogram)

Description

- The wavelet transform is a linear time-frequency representation that preserves time shifts and time scalings.
- The continuous wavelet transform is good at detecting transients in nonstationary signals, and for signals in which instantaneous frequency grows rapidly.
- The CWT is invertible.

- The CWT tiles the time-frequency plane with variable-sized windows. The window automatically widens in time, making it suitable for low-frequency phenomena, and narrows for high frequency phenomena.

Potential Applications

The applications of this time-frequency method include, but are not limited to:

- *Electrocardiograms (ECG)*: The most clinically useful information of the ECG signal is found in the time intervals between its consecutive waves and amplitudes defined by its features. The wavelet transform breaks down the ECG signal into scales, making it easier to analyze the ECG signal in different frequency ranges easier to analyze.
- *Electroencephalogram (EEG)*: Raw EEG signals suffer from poor spatial resolution, low signal-to-noise ratio, and artifacts. Continuous wavelet decomposition of a noisy signal concentrates intrinsic signal information in a few wavelet coefficients having large absolute values without modifying the random distribution of noise. Therefore, denoising can be achieved by thresholding the wavelet coefficients.
- *Signal demodulation*: Demodulate extended binary phase shift keying (EBPSK) using an adaptive wavelet construction method.
- *Deep learning*: The CWT can be used to create time-frequency representations that can be used to train a convolutional neural network. “Classify Time Series Using Wavelet Analysis and Deep Learning” shows how to classify ECG signals using scalograms and transfer learning.

How to Use

- `cwt` computes the continuous wavelet transform and displays the scalogram. Alternatively, create a CWT filter bank using `cwtfilterbank` and apply the `wt` function. Use this method to run in parallel applications or when computing the transform for several functions in a loop.
- `icwt` inverts the continuous wavelet transform.
- **Signal Analyzer** has a scalogram view to visualize the CWT of multiple time series.

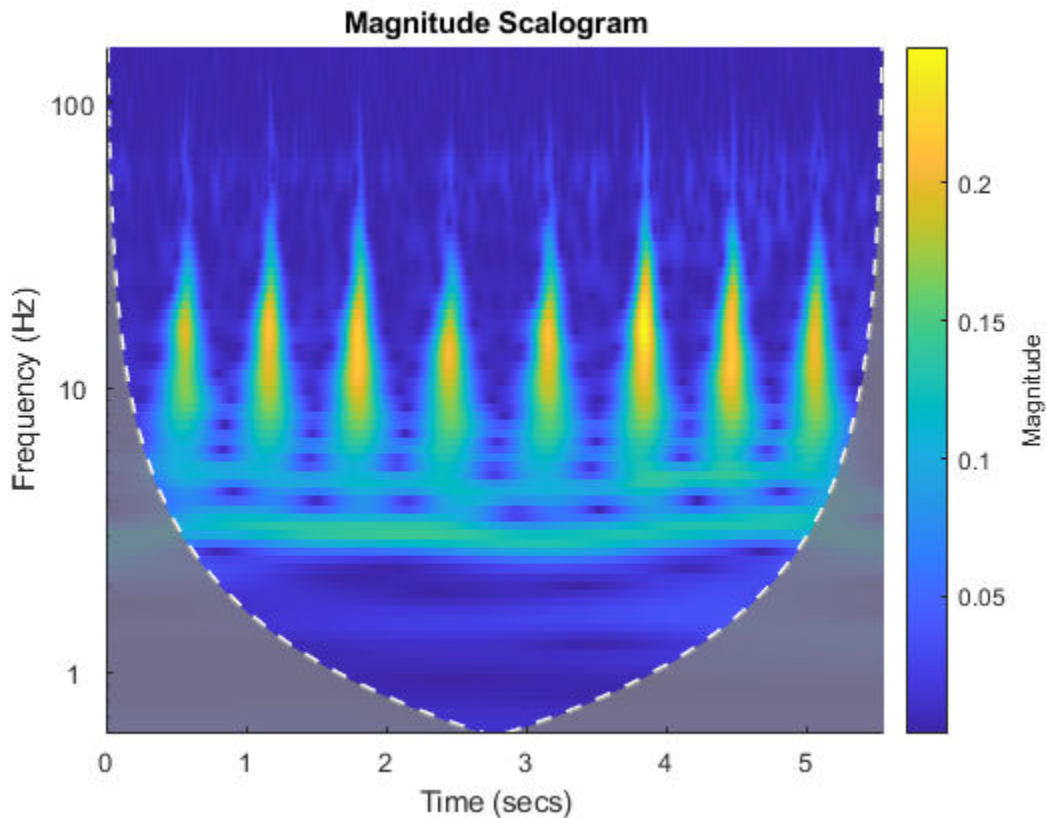
Example: ECG Signal

Load a noisy ECG waveform sampled at 360 Hz.

```
load ecg
Fs = 360;
```

Compute the continuous wavelet transform.

```
cwt(ecg, Fs)
```



The ECG data is taken from the MIT-BIH Arrhythmia Database [2].

Wigner-Ville Distribution

Description

- The Wigner-Ville distribution (WVD) is a quadratic energy density computed by correlating the signal with a time and frequency translated and complex-conjugated version of itself.

- The Wigner-Ville distribution is always real even if the signal is complex.
- The time- and frequency- integrals of the Wigner-Ville distribution correspond to the signal's instantaneous power and spectral energy density.
- The instantaneous frequency and group delay can be evaluated using local first-order moments of the Wigner distribution.
- The time resolution of the WVD is equal to the number of input samples.
- The Wigner distribution can locally assume negative values.

Potential Applications

The applications of this time-frequency method include, but are not limited to:

- *Otoacoustic emissions (OAEs)*: OAEs are narrowband oscillatory signals emitted by the cochlea (inner ear), and their presence is indicative of normal hearing.
- *Quantum mechanics*: Quantum corrections to classical statistical mechanics, model electron transport, and calculate static and dynamic properties of many-body quantum systems.

How to Use

- `wvd` computes the Wigner-Ville distribution.
- `xwvd` computes the cross Wigner-Ville distribution of two signals. See “Use Cross Wigner-Ville Distribution to Estimate Instantaneous Frequency” for more details.

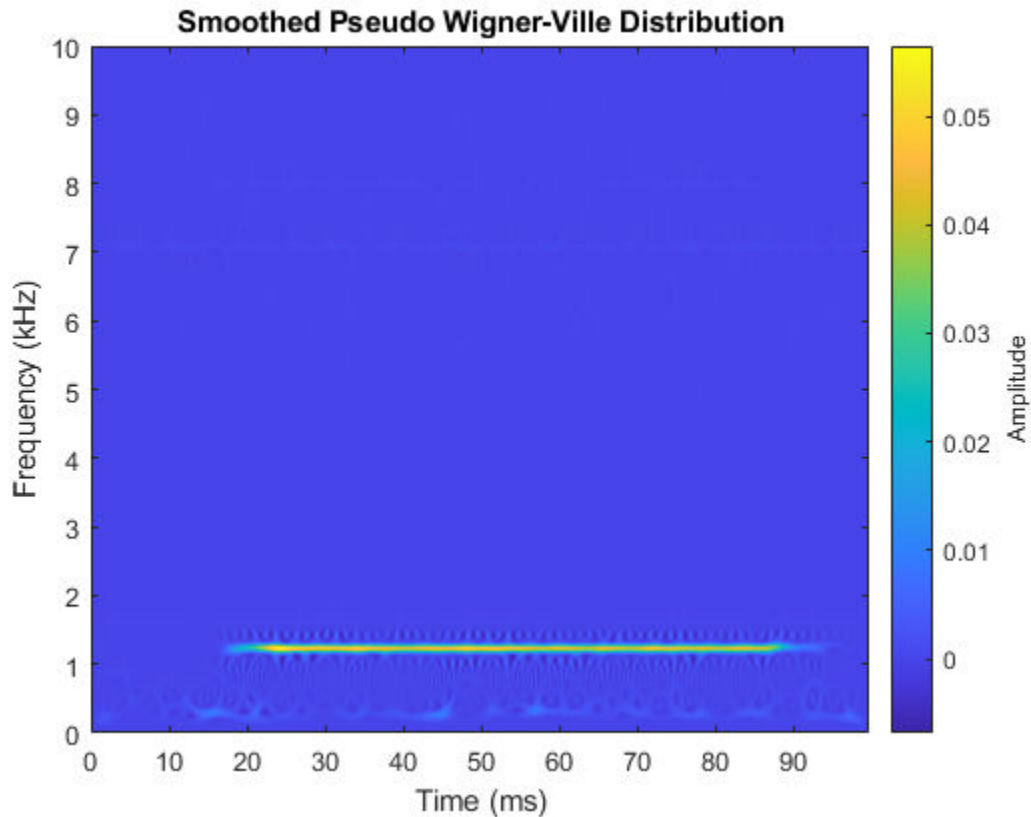
Example: Otoacoustic Emission

Load a data file containing otoacoustic emission data sampled at 20 kHz. The emission is produced by a stimulus beginning at 25 milliseconds and ending at 175 milliseconds.

```
load dpoae
Fs = 20e3;
```

Compute the smoothed-pseudo Wigner Ville distribution of the otoacoustic data. The convenience plot isolates the emission frequency at roughly the expected value 1.2 kHz.

```
wvd(dpoaets, Fs, 'smoothedPseudo', kaiser(511, 10), kaiser(511, 10), 'NumFrequencyPoints', 4000)
```

For more details on otoacoustic emissions, see "Determining Exact Frequency Through the Analytic CWT" in "CWT-Based Time-Frequency Analysis".

Reassignment and Synchrosqueezing

Description

- Reassignment sharpens the localization of spectral estimates and produces spectrograms that are easier to read and interpret. The technique relocates each spectral estimate to the center of energy of its bin instead of the bin's geometric center. It provides exact localization for chirps and impulses.

- The Fourier synchrosqueezed transform starts from the short-time Fourier transform and "squeezes" its values so that they concentrate around curves of instantaneous frequency in the time-frequency plane.
- The wavelet synchrosqueezed transform reassigns the signal energy in frequency.
- Both the Fourier synchrosqueezed transform and the wavelet synchrosqueezed transform are invertible.
- The reassigned and synchrosqueezing methods are especially suited to track and extract time-frequency ridges.

Potential Applications

The applications of this time-frequency method include, but are not limited to:

- *Audio signal processing:* Synchrosqueezing transform (SST) was originally introduced in the context of audio signal analysis.
- *Seismic data:* Analysis of seismic data to find oil and gas traps. Synchrosqueezing can also detect deep-layer weak signals that are usually smeared in seismic data.
- *Oscillations in power systems:* A steam turbine and electric generator can have mechanical subsynchronous oscillation (SSO) modes between the various turbine stages and the generator. The frequency of the SSO is generally between 5 Hz and 45 Hz, and the mode frequencies are often close to each other. The antinoise ability and time-frequency resolution of WSST improves the readability of the time-frequency view.
- *Deep learning:* Synchrosqueezed transforms can be used to extract time-frequency features and fed into a network that classifies time-series data. "Waveform Segmentation Using Deep Learning" (Signal Processing Toolbox) shows how `fsst` outputs can be fed into an LSTM network that classifies ECG signals.

How to Use

- Use the 'reassigned' option in `spectrogram`, set the 'Reassigned' argument to `true` in `pspectrum`, or check the **Reassign** box in the spectrogram view of **Signal Analyzer** to compute reassigned spectrograms.
- `fsst` computes the Fourier synchrosqueezed transform. Use the `ifsst` function to invert the Fourier synchrosqueezed transform. (See "Fourier Synchrosqueezed Transform of Speech Signal" (Signal Processing Toolbox) for reconstruction of speech signals using `ifsst`.)

- `wssst` computes the wavelet synchrosqueezed transform. Use the `iwsst` function to invert the wavelet synchrosqueezed transform. (See “Inverse Synchrosqueezed Transform of Chirp” for reconstruction of a quadratic chirp using `iwsst`.)

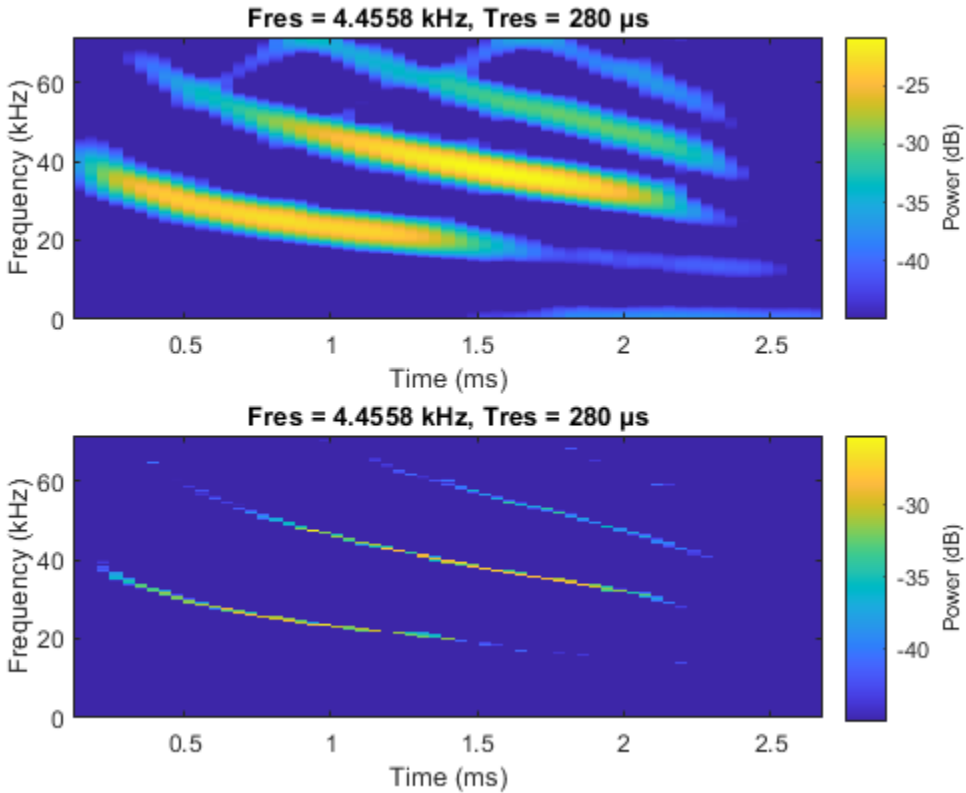
Example: Echolocation Pulse

Load an echolocation pulse emitted by a big brown bat (*Eptesicus Fuscus*). The sampling interval is 7 microseconds.

```
load batsignal
Fs = 1/DT;
```

Compute the reassigned spectrogram of the signal.

```
subplot(2,1,1)
pspectrum(batsignal,Fs,'spectrogram','TimeResolution',280e-6, ...
          'OverlapPercent',85,'MinThreshold',-45,'Leakage',0.9)
subplot(2,1,2)
pspectrum(batsignal,Fs,'spectrogram','TimeResolution',280e-6, ...
          'OverlapPercent',85,'MinThreshold',-45,'Leakage',0.9,'Reassign',true)
```



Thanks to Curtis Condon, Ken White, and Al Feng of the Beckman Center at the University of Illinois for the bat data and permission to use it in this example [3].

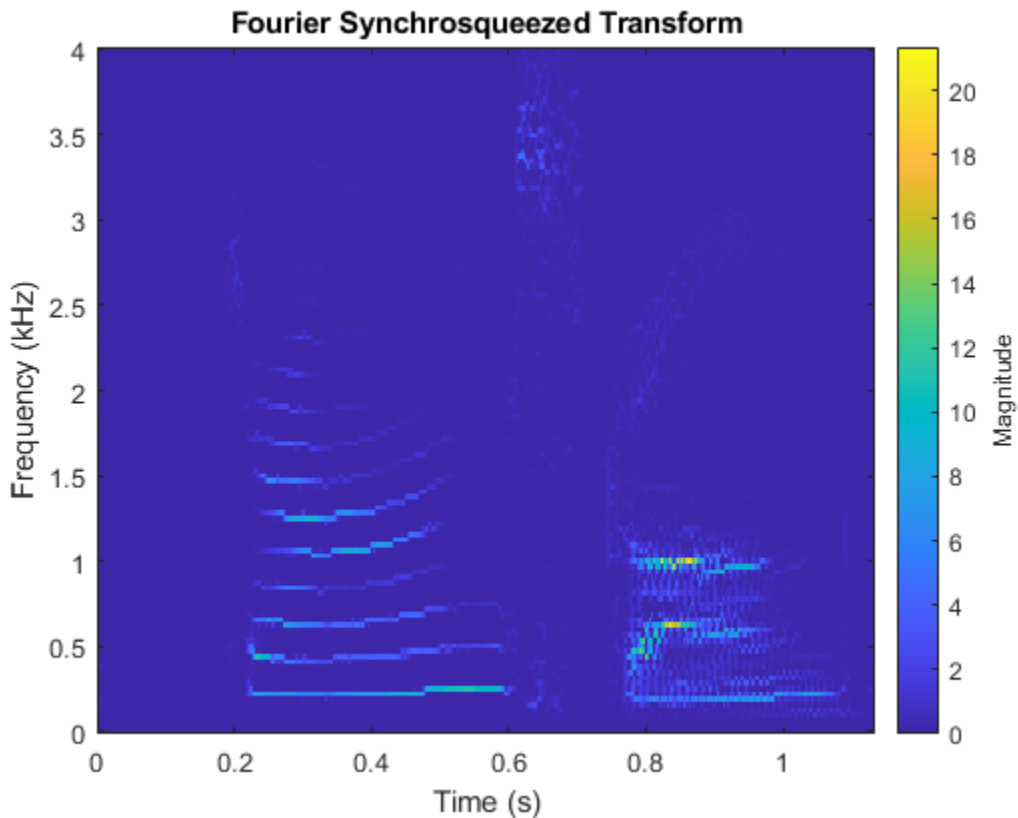
Example: Speech Signals

Load a file containing the word "strong," spoken by a woman and by a man. The signals are sampled at 8 kHz. Concatenate them into a single signal.

```
load Strong
x = [her' him'];
```

Compute the synchrosqueezed Fourier transform of the signal. Window the signal using a Kaiser window with shape factor $\beta = 20$.

```
fsst(x,Fs,kaiser(256,20),'yaxis')
```



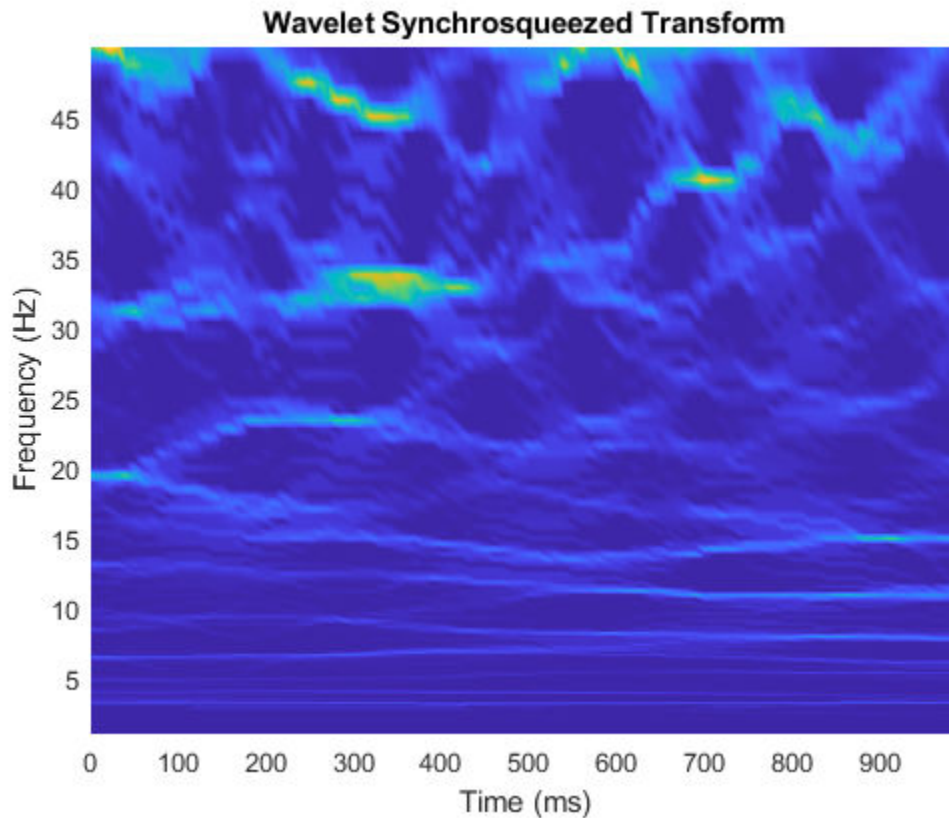
Example: Synthetic Seismic Data

Load the synthetic seismic data sampled at 100 Hz for 1 second.

```
load SyntheticSeismicData
```

Compute the wavelet synchrosqueezed transform of the seismic data using the bump wavelet and 30 voices per octave.

```
wsst(x,Fs,'bump','VoicesPerOctave',30,'ExtendSignal',true)
```



The seismic signal is generated using the two sinusoids mentioned in "Time-Frequency Analysis of Seismic Data Using Synchrosqueezing Transform" by Ping Wang, Jinghui Gao, and Zhiguo Wang [4].

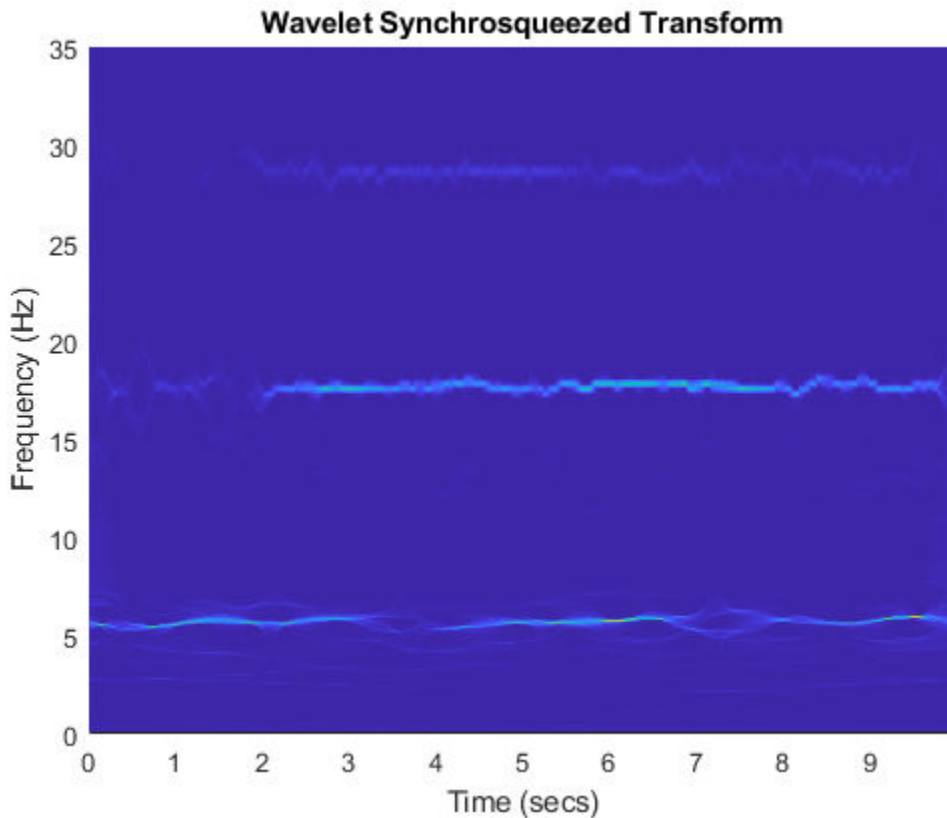
Example: Earthquake Vibration

Load acceleration measurements recorded on the first floor of a three story test structure under earthquake conditions. The measurements are sampled at 1 kHz.

```
load quakevib
Fs = 1e3;
```

Compute the wavelet synchrosqueezed transform of the acceleration measurements. You are analyzing vibration data that exhibit a cyclic behavior. The synchrosqueezed transform allows you to isolate the three frequency components, separated by roughly 11 Hz. The main vibration frequency is at 5.86 Hz, and the equispaced frequency peaks suggest that they are harmonically related. The cyclic behavior of the vibrations is also visible.

```
wsst(gfloor10L,Fs,'bump','VoicesPerOctave',48)  
ylim([0 35])
```



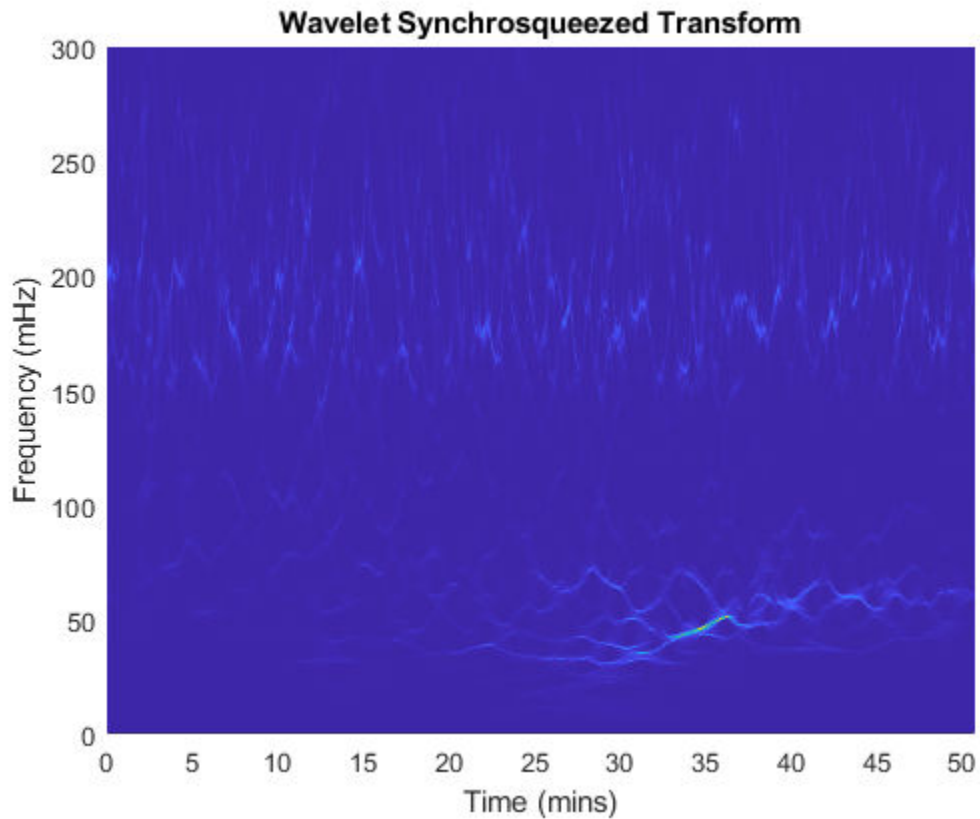
Example: Kobe Earthquake Data

Load seismograph data recorded during the 1995 Kobe earthquake. The data has a sample rate of 1 Hz.

```
load kobe  
Fs = 1;
```

Compute the wavelet synchrosqueezed transform that isolates the different frequency components of the seismic data.

```
wsst(kobe,Fs,'bump','VoicesPerOctave',48)  
ylim([0 300])
```



The data are seismograph (vertical acceleration, nm/sq.sec) measurements recorded at Tasmania University, Hobart, Australia on 16 January 1995 beginning at 20:56:51 (GMT) and continuing for 51 minutes at 1 second intervals [5].

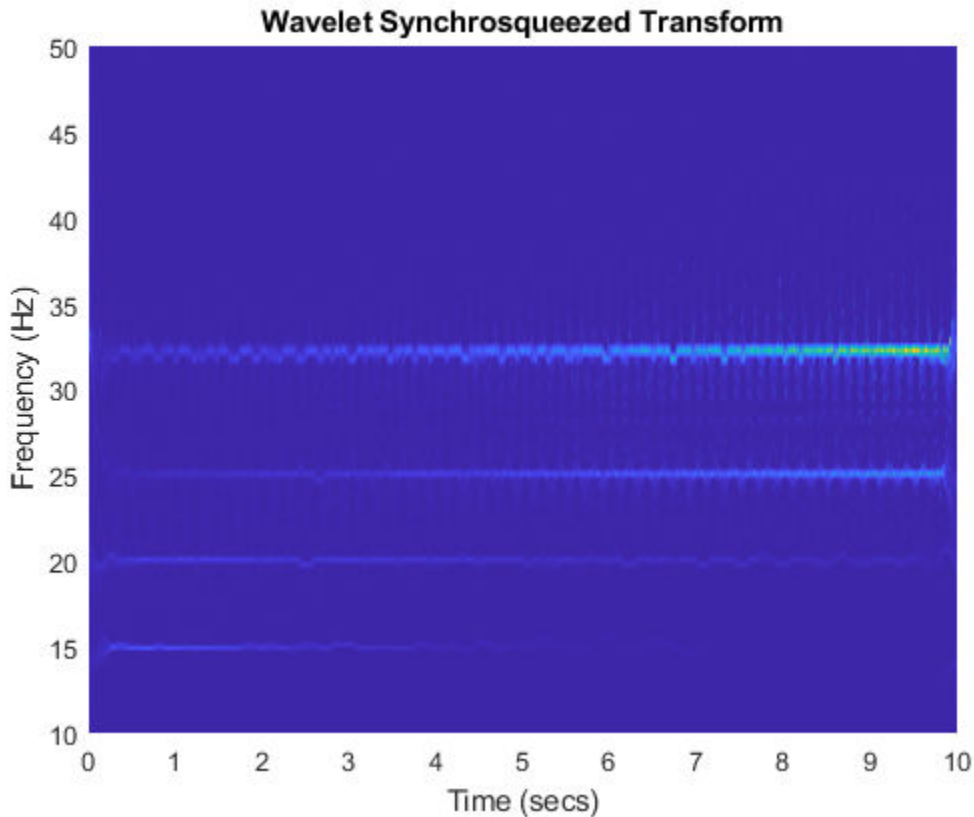
Example: Subsynchronous Oscillation in Power Systems

Load the subsynchronous oscillation data of a Power System.

```
load OscillationData
```

Compute the wavelet synchrosqueezed transform using the bump wavelet and 48 voices per octave. The four mode frequencies are at 15 Hz, 20 Hz, 25 Hz and 32 Hz. Notice that the energies of the modes at 15 Hz and 20 Hz decrease with time, whereas the energy of the modes at 25 Hz and 32 Hz increase gradually over time.

```
wsst(x,Fs,'bump','VoicesPerOctave',48)  
ylim([10 50])
```



This synthetic subsynchronous oscillation data was generated using the equation defined by Zhao et al in "Application of Synchrosqueezed Wavelet Transforms for Extraction of the Oscillatory Parameters of Subsynchronous Oscillation in Power Systems" [6].

Constant-Q Gabor Transform

Description

- The constant- Q nonstationary Gabor transform uses windows with different center frequencies and bandwidths such that the ratio of center frequency to bandwidth, the Q factor, remains constant.

- The constant- Q Gabor transform enables the construction of stable inverses, yielding perfect signal reconstruction.
- In frequency space, the windows are centered at logarithmically spaced center frequencies.

Potential Applications

The applications of this time-frequency method include, but are not limited to:

Audio signal processing: The fundamental frequencies of the tones in music are geometrically spaced. The frequency resolution of the human auditory system is approximately constant- Q , making this technique appropriate for music signal processing.

How to Use

- `cqt` computes the constant- Q Gabor transform.
- `icqt` inverts the constant- Q Gabor transform.

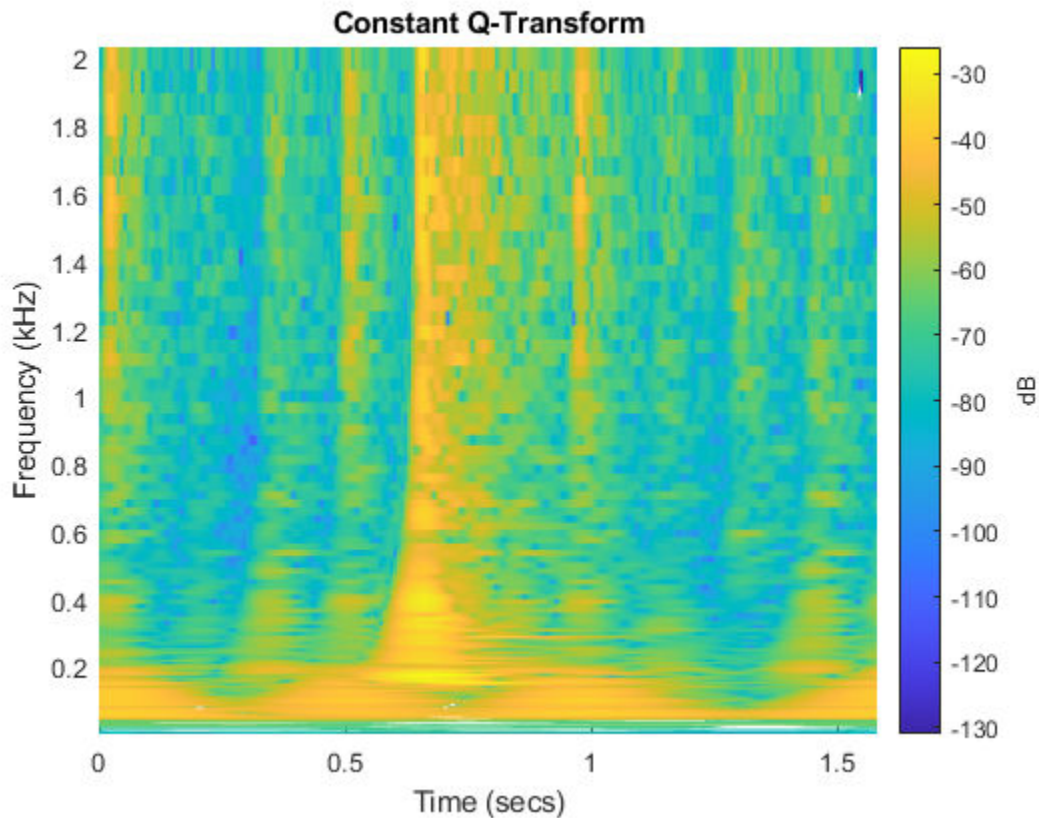
Example: Rock Music

Load an audio file containing a fragment of Rock music with vocals, drums, and guitar. The signal has a sample rate of 44.1 kHz.

```
load drums
```

Set the frequency range over which the CQT has a logarithmic frequency response to be the minimum allowable frequency to 2 kHz. Perform the CQT of the signal using 20 bins per octave.

```
minFreq = fs/length(audio);  
maxFreq = 2000;  
cqt(audio, 'SamplingFrequency', fs, 'BinsPerOctave', 20, 'FrequencyLimits', [minFreq maxFreq])
```



Empirical Mode Decomposition and Hilbert-Huang Transform

Description

- The empirical mode decomposition decomposes the signals into intrinsic mode functions which form a complete and nearly orthogonal basis for the original signal.
- The Hilbert-Huang transform computes the instantaneous frequency of each intrinsic mode function.
- These two methods combined are useful for analyzing nonlinear and nonstationary signals.

Potential Applications

The applications of this time-frequency method include, but are not limited to:

- *Physiological signal processing*: Analyze human EEG response to transcranial magnetic stimulation (TMS) of the brain cortex.
- *Structural applications*: Locate anomalies that appear as cracks, delamination, or stiffness loss in beams and plates.
- *System identification*: Isolate modal damping ratios of structures with closely spaced modal frequencies.
- *Ocean engineering*: Identify transient electromagnetic disturbances caused by humans in underwater electromagnetic environments.
- *Solar physics*: Extract periodic components of sunspot data.
- *Atmospheric turbulence*: Observe stable boundary layer to separate turbulent and nonturbulent motions.
- *Epidemiology*: Assess traveling speed of communicative diseases such as Dengue fever.

How to Use

- `emd` computes the empirical mode decomposition.
- `hht` computes the Hilbert Huang spectrum of an empirical mode decomposition.

Example: Bearing Vibration

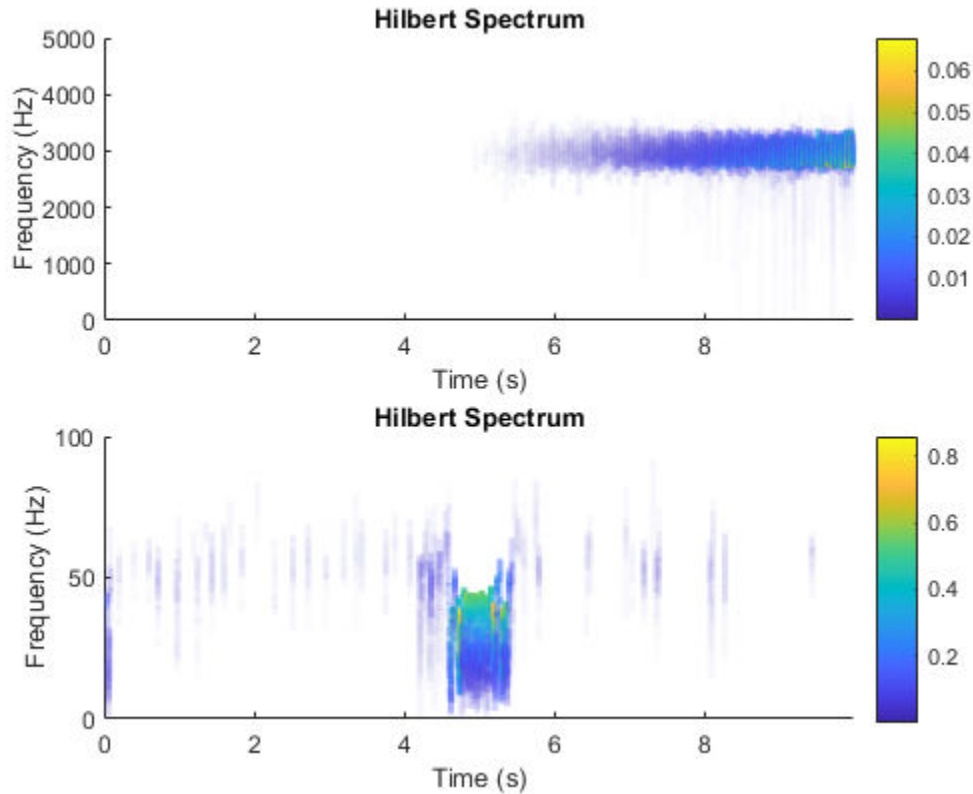
Load the vibration signal from a defective bearing generated in the “Compute Hilbert Spectrum of Vibration Signal” (Signal Processing Toolbox) example. The signal is sampled at a rate 10 kHz.

```
load bearingVibration
```

Compute the first five intrinsic mode functions (IMFs) of the signal. Plot the Hilbert spectrum of the first and third empirical modes. The first mode reveals increasing wear due to high-frequency impacts on the bearing's outer race. The third mode shows a resonance occurring halfway through the measurement process that caused the defect in the bearing.

```
imf = emd(y, 'MaxNumIMF', 5, 'Display', 0);  
subplot(2,1,1)  
hht(imf(:,1), fs)
```

```
subplot(2,1,2)  
hht(imf(:,3),fs,'FrequencyLimits',[0 100])
```



References

- [1] The Pacific blue whale file is obtained from the library of animal vocalizations maintained by the Cornell University Bioacoustics Research Program.
- [2] Moody G. B, Mark R. G. *The impact of the MIT-BIH Arrhythmia Database*. IEEE Eng in Med and Biol 20(3):45-50 (May-June 2001). (PMID: 11446209)
- [3] Thanks to Curtis Condon, Ken White, and Al Feng of the Beckman Center at the University of Illinois for the bat echolocation data.

- [4] Wang, Ping, Gao, J., and Wang, Z. *Time-Frequency Analysis of Seismic Data Using Synchrosqueezing Transform*, IEEE Geoscience and Remote Sensing Letters, Vol 12, Issue 11, Dec. 2014.
- [5] Seismograph (vertical acceleration, nm/sq.sec) of the Kobe earthquake, recorded at Tasmania University, Hobart, Australia on 16 January 1995 beginning at 20:56:51 (GMTRUE) and continuing for 51 minutes at 1 second intervals.
- [6] Zhao et al. *Application of Synchrosqueezed Wavelet Transforms for Extraction of the Oscillatory Parameters of Subsynchronous Oscillation in Power Systems* MDPI Energies; Published 12 June 2018.
- [7] Boashash, Boualem. *Time-Frequency Signal Analysis and Processing: A Comprehensive Reference* Elsevier, 2016.

See Also

Apps

Signal Analyzer

Functions

cqt | cwt | cwtfilterbank | emd | fsst | hht | icqt | icwt | ifsst | istft | iwsst | kurtogram | pkurtosis | pspectrum | spectrogram | stft | wsst | wt | wvd | xspectrogram | xwvd

Wavelet Packets

- “About Wavelet Packet Analysis” on page 5-2
- “1-D Wavelet Packet Analysis” on page 5-6
- “2-D Wavelet Packet Analysis” on page 5-19
- “Importing and Exporting from Wavelet Analyzer App” on page 5-25
- “Wavelet Packets” on page 5-32
- “Introduction to Object-Oriented Features” on page 5-51
- “Objects in the Wavelet Toolbox Software” on page 5-52
- “Examples Using Wavelet Packet Tree Objects” on page 5-53
- “Description of Objects in the Wavelet Toolbox Software” on page 5-65
- “Build Wavelet Tree Objects” on page 5-71

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 1-D and 2-D 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.

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 5-32.

Some object-oriented programming features are used for wavelet packet tree structures. For more detail, refer to “Introduction to Object-Oriented Features” on page 5-51.

This chapter takes you through the features of 1-D and 2-D 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 app functions can be found in the section “Examples Using Wavelet Packet Tree Objects” on page 5-53.

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

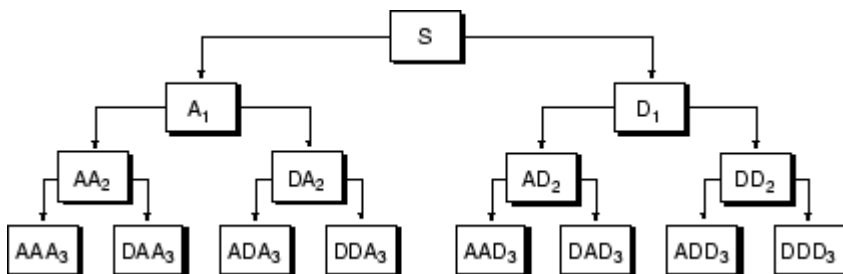
Decomposition Structure Utilities

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

Denoising and Compression

Function Name	Purpose
ddencmp	Default values for denoising and compression
wpbmpen	Penalized threshold for wavelet packet denoising
wpdencmp	Denoising and compression using wavelet packets
wpthcoef	Wavelet packets coefficients thresholding
wthrmngr	Threshold settings manager

In the wavelet packet framework, compression and denoising 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.

Denosing and compression are interesting applications of wavelet packet analysis. The wavelet packet denosing 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 1-D wavelet packet analysis to compress and to denoise a signal.

1-D Wavelet Packet Analysis

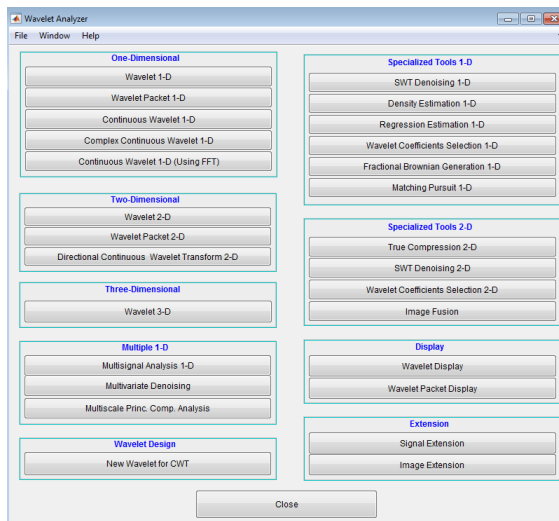
In this section...

- “Starting the Wavelet Packet 1-D Tool” on page 5-6
- “Importing a Signal” on page 5-7
- “Analyzing a Signal” on page 5-7
- “Computing the Best Tree” on page 5-9
- “Compressing a Signal Using Wavelet Packets” on page 5-10
- “De-Noising a Signal Using Wavelet Packets” on page 5-13

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

Starting the Wavelet Packet 1-D Tool

- 1 From the MATLAB prompt, type `waveletAnalyzer`. The **Wavelet Analyzer** appears.



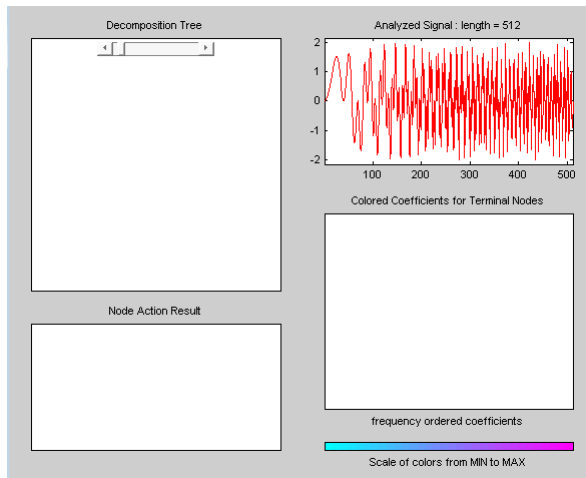
Click the **Wavelet Packet 1-D** menu item.

Importing a Signal

- 1 At the MATLAB command prompt, type

```
load sumlichr;
```
- 2 In the **Wavelet Packets 1-D** tool, select **File > Import from Workspace > Import Signal**. When the **Import from Workspace** dialog box appears, select the `sumlichr` variable. Click **OK** to import the data.

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



Analyzing a Signal

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

Data (Size)	sumlichr (512)	
Wavelet	db	2
Level	4	
Entropy	threshold	
Threshold	1	
<input type="button" value="Analyze"/>		

The available entropy types are listed below.

Type	Description
Shannon	Nonnormalized entropy involving the logarithm of the squared value of each signal sample — or, more formally, $-\sum s_i^2 \log(s_i^2).$
Threshold	The number of samples for which the absolute value of the signal exceeds a threshold ε .
Norm	The concentration in l^p norm with $1 \leq p$.
Log Energy	The logarithm of “energy,” defined as the sum over all samples: $\sum \log(s_i^2).$
SURE (Stein's Unbiased Risk Estimate)	A threshold-based method in which the threshold equals $\sqrt{2 \log_e(n \log_2(n))}$ where n is the number of samples in the signal.
User	An entropy type criterion you define in a file.

For more information about the available entropy types, user-defined entropy, and threshold parameters, see the `wentropy` reference page and “Choosing the Optimal Decomposition” on page 5-42.

Note Many capabilities are available using the command area on the right of the **Wavelet Packet 1-D** window.

Computing the Best Tree

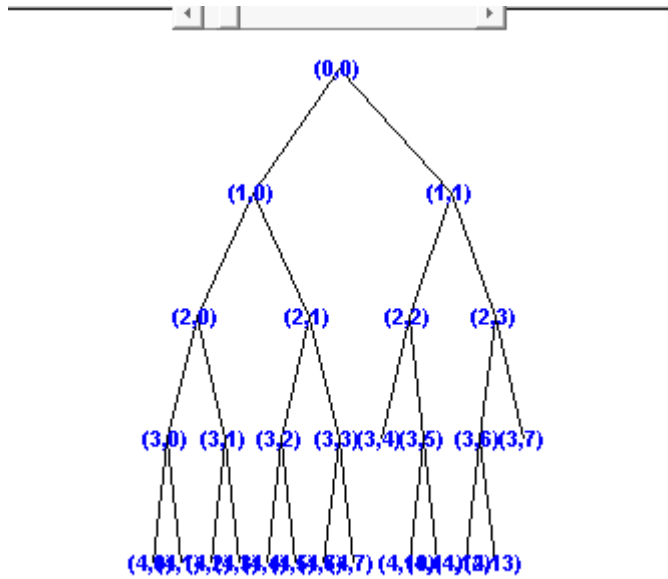
Because there are so many ways to reconstruct the original signal from the wavelet packet decomposition tree, we select the best tree before attempting to compress the signal.

- 1 Click the **Best Tree** button.



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

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

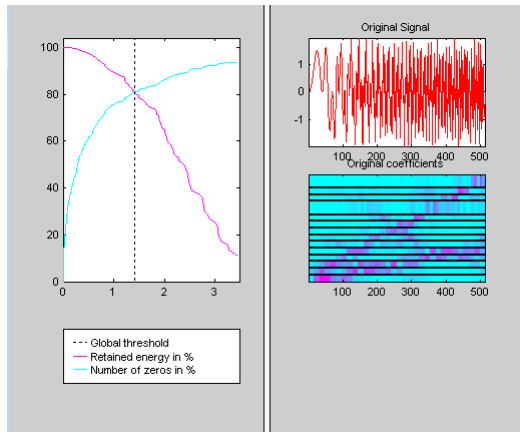


Compressing a Signal Using Wavelet Packets

Selecting a Threshold for Compression

- 1 Click the **Compress** button.

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

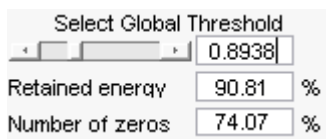


The leftmost graph shows how the threshold (vertical black dotted line) has been chosen automatically (1.482) to balance the number of zeros in the compressed signal (blue curve that increases as the threshold increases) with the amount of energy retained in the compressed signal (purple curve that decreases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

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

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

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



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

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

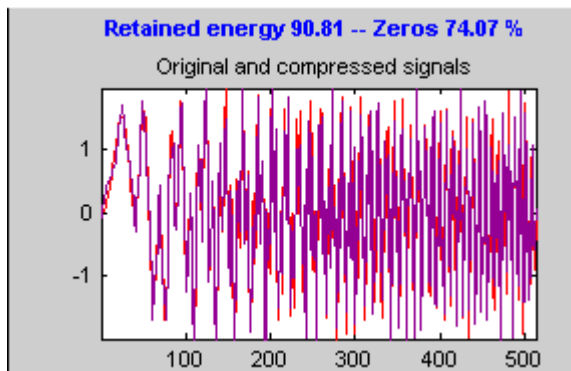
Note that, as we have *reduced* the threshold from 1.482 to 0.8938,

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

Compressing a Signal

- 1 Click the **Compress** button.

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



The original (red) and compressed (purple) 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.

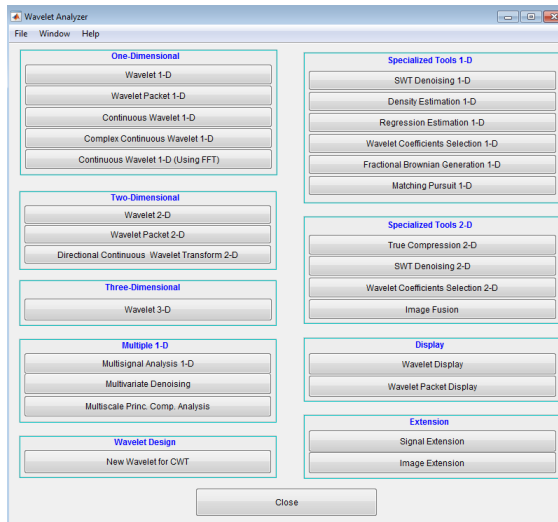
A more thorough discussion of the SURE criterion appears in "Choosing the Optimal Decomposition" on page 5-42. For now, suffice it to say that this method works well if your signal is normalized in such a way that the data fit the model $x(t) = f(t) + e(t)$, where $e(t)$ is a Gaussian white noise with 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

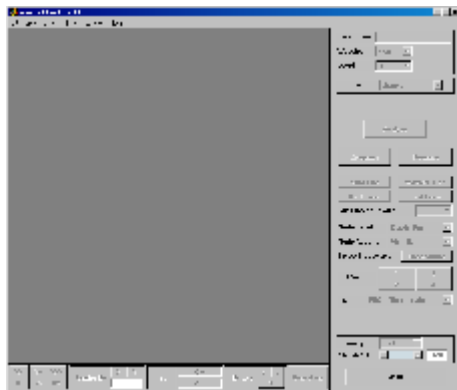
1 From the MATLAB prompt, type `waveletAnalyzer`.

The **Wavelet Analyzer** appears.



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

The tool appears on the desktop.



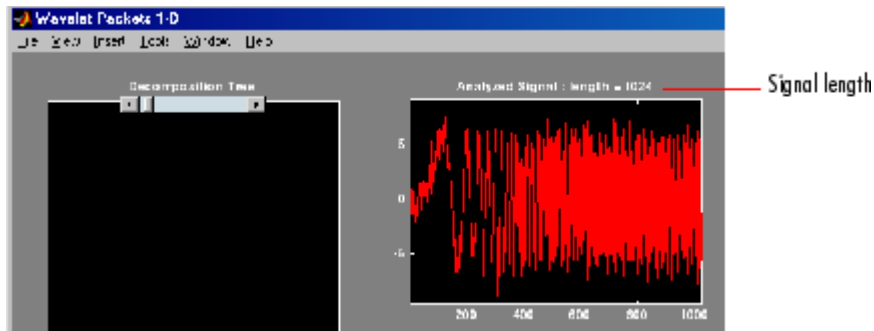
Importing a Signal

- 2 At the MATLAB command prompt, type
`load noisichir;`

In the **Wavelet Packet 1-D** tool, select **File > Import from Workspace > Import Signal**. When the **Import from Workspace** dialog box appears, select the `sumlichr` variable. Click **OK** to import the data

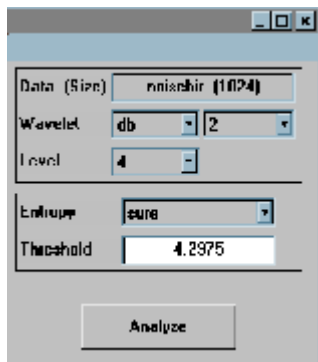
Note You can use **File > Load > Signal** to load a signal by navigating to its location.

- 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

- 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-14.

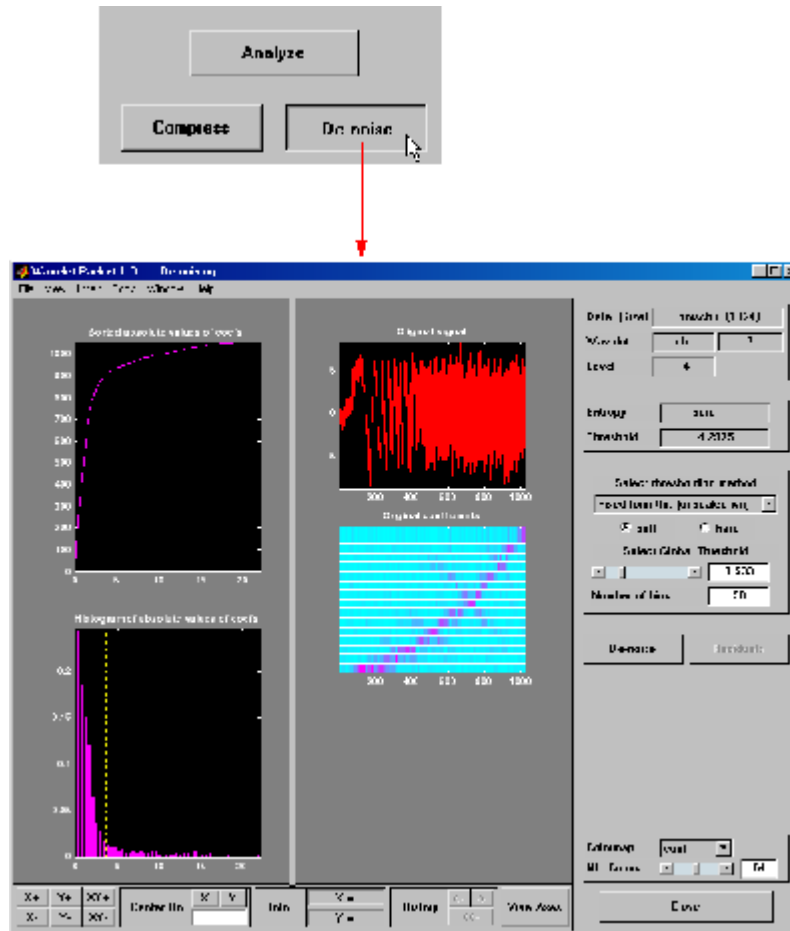
Computing the Best Tree and Performing De-Noising

- 5 Click the **Best Tree** button.

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

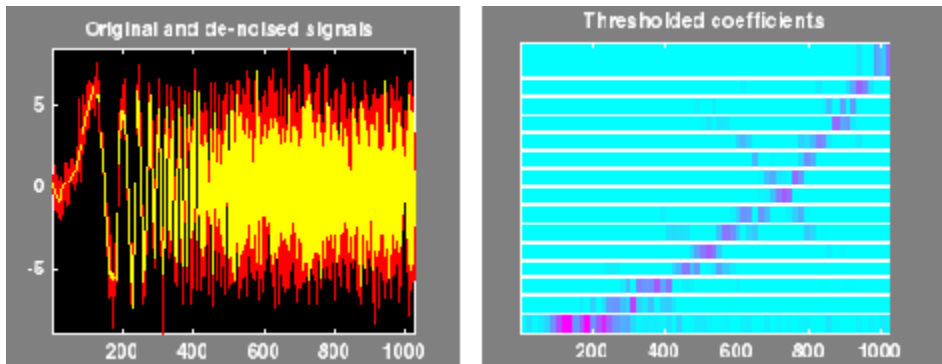


- 6 Click the **De-noise** button. This brings up the **Wavelet Packet 1-D De-Noising** window.



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

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



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

2-D Wavelet Packet Analysis

In this section...
“Starting the Wavelet Packet 2-D Tool” on page 5-19
“Compressing an Image Using Wavelet Packets” on page 5-21

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”).

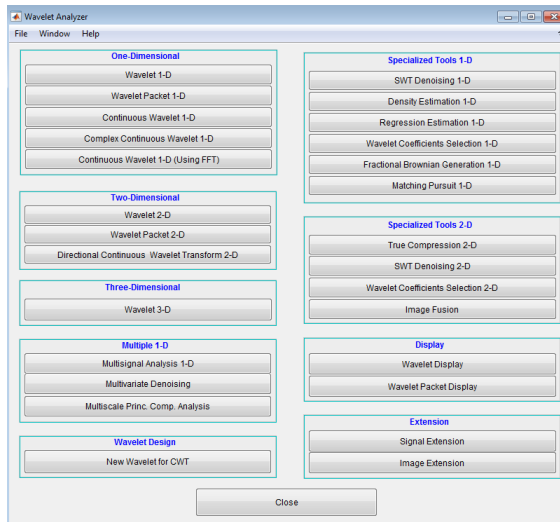
“The technique involves a 2-D DWT, uniform scalar quantization (a process that truncates, or quantizes, the precision of the floating-point DWT output) and Huffman entropy coding (i.e., encoding the quantized DWT output with a minimal number of bits).” (Brislawn, see the reference [Bris95] p. 1278, listed in “References”).

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 `waveletAnalyzer`. The **Wavelet Analyzer** appears.



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

Importing an Image

At the MATLAB command prompt, type

```
load detfingr;
```

In the **Wavelet Packet 2-D** tool, select **File > Import from Workspace > Import Image**. When the **Import from Workspace** dialog box appears, select the X variable. Click **OK** to import the fingerprint image.

Analyzing an Image

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

Wavelet	haar
Level	3
Entropy	shannon

Analyze

Note Many capabilities are available using the command area on the right of the **Wavelet Packet 2-D** window.

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

Compress	De-noise
Initial Tree	Wavelet Tree
Best Tree	Best Level

Compressing an Image Using Wavelet Packets

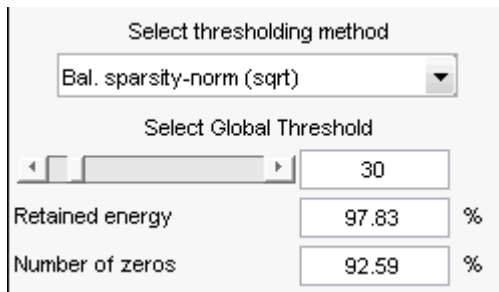
- Click the **Compress** button to bring up the **Wavelet Packet 2-D Compression** window. Select the **Bal. sparsity-norm (sqrt)** option from the **Select thresholding method** menu.

Select thresholding method	
Bal. sparsity-norm (sqrt)	▼
Select Global Threshold	
<input type="text" value="7.125"/>	
Retained energy	99.77 %
Number of zeros	63.41 %

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.

- 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.



The screenshot shows a software interface for wavelet packet thresholding. It features a dropdown menu for the thresholding method, a slider for the global threshold, and two data rows showing the results of the thresholding process.

Select thresholding method	
Bal. sparsity-norm (sqrt)	

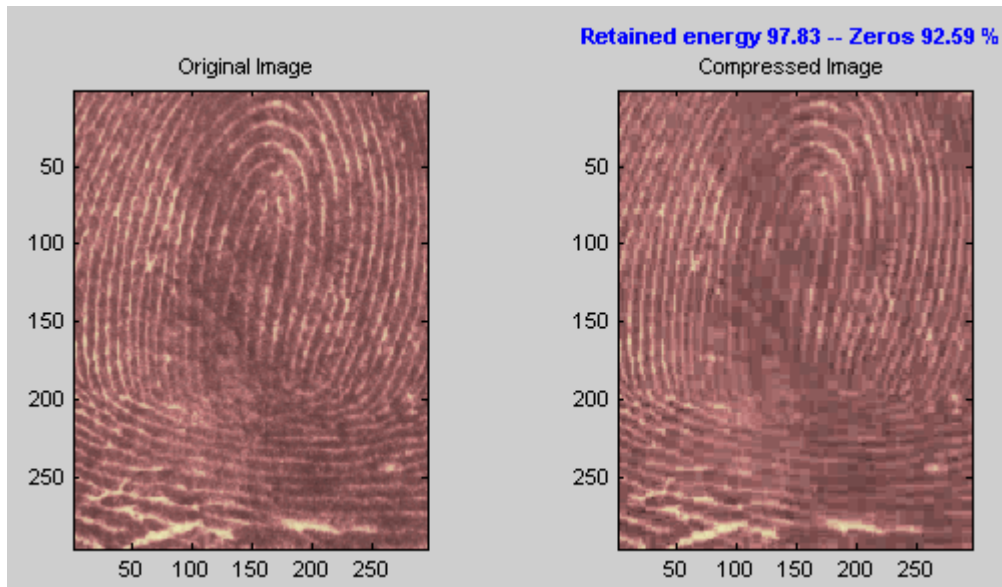
Select Global Threshold	
<input type="text" value="30"/>	

Retained energy	<input type="text" value="97.83"/> %
Number of zeros	<input type="text" value="92.59"/> %

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.

- 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.

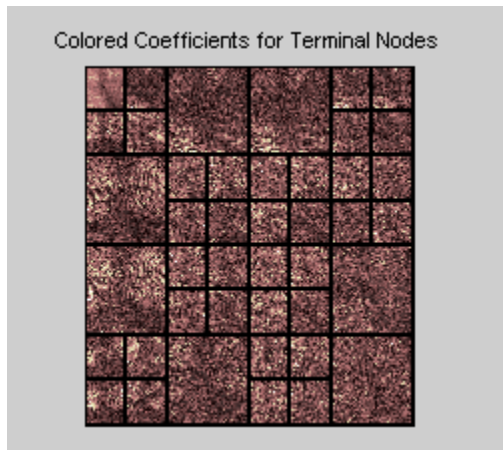


- 4 Click the **Close** button located at the bottom of the **Wavelet Packet 2-D Compression** window. Update the synthesized image by clicking **Yes** when the dialog box appears.

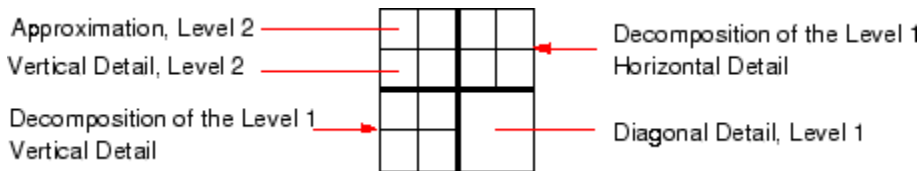
Take this opportunity to try out your own compression strategy. Adjust the threshold value, the entropy function, and the wavelet, and see if you can obtain better results.

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 6-82.

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 Wavelet Analyzer App

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 1-D and 2-D 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 1-D or 2-D 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 denoising operation on the original signal. When you close the **Wavelet Packet 1-D Denoising** or **Wavelet Packet 1-D Compression** window, update the synthesized signal by clicking **Yes** in the dialog box.

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

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

To load the signal into your workspace, simply type

```
load synthsig  
whos
```

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

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

```
valTHR
```

```
valTHR =  
    1.9961
```

Saving Synthesized Images

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

For example, load the example analysis:

File > Example Analysis > db1 - depth: 1 - ent: shannon > woman

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

Then, from the **Wavelet 2-D** tool, select the **File > Save > Synthesized Image** menu option.

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

To load the image into your workspace, simply type

```
load wpsymage  
whos
```

Name	Size	Bytes	Class
X	256x256	524288	double array

Name	Size	Bytes	Class
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 denoising or compression process are given by the wavelet name (`wname`) and the global threshold (`valTHR`).

Saving 1-D Decomposition Structures

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

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

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

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

A dialog box appears that lets you specify a folder and file name for storing the decomposition data. Type the name `wpdecex1d`.

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

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

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

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

Saving 2-D Decomposition Structures

The file format, variables, and conventions are exactly the same as in the 1-D case except for the extension, which is `wp2` (wavelet packet 2-D). The variables saved are the same as with the 1-D case, with the addition of the colormap matrix `map`:

Name	Size	Bytes	Class
<code>data_name</code>	1x5	10	char array
<code>map</code>	255x3	6120	double array
<code>tree_struct</code>	1x1	527400	wptree object
<code>valTHR</code>	1x1	8	double array

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

In the Wavelet Packet Denoising windows, you can save the denoised 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 Denoising 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 1-D and 2-D wavelet packet decompositions into the graphical interface tools. The information you load may have been previously exported from the graphical interface, and then manipulated in the workspace, or it may have been information you generated initially from the command line.

In either case, you must observe the strict file formats and data structures used by the graphical tools, or else errors will result when you try to load information.

Loading Signals

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

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

```
save warma warma
```

The workspace variable `warma` must be a vector.

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

```
sizwarma =  
          1          1000
```

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

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

Note The first 1-D 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 2-D 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 1-D and 2-D 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.

1-D data file contains the following variables:

Variable	Status	Description
tree_struct	Required	Object specifying the tree structure
data_name	Optional	Character vector specifying the name of the decomposition
valTHR	Optional	Global threshold (can be empty if neither compression nor denoising has been done)

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

2-D data file contains the following variables:

Variable	Status	Description
tree_struct	Required	Object specifying the tree structure
data_name	Optional	Character vector specifying the name of the decomposition
map	Optional	Image map
valTHR	Optional	Global threshold (can be empty if neither compression nor denoising 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.

Wavelet Packets

In this section...
“From Wavelets to Wavelet Packets” on page 5-32
“Wavelet Packets in Action: An Introduction” on page 5-33
“Building Wavelet Packets” on page 5-36
“Wavelet Packet Atoms” on page 5-39
“Organizing the Wavelet Packets” on page 5-40
“Choosing the Optimal Decomposition” on page 5-42
“Some Interesting Subtrees” on page 5-46
“Wavelet Packets 2-D Decomposition Structure” on page 5-50
“Wavelet Packets for Compression and Denoising” on page 5-50

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.

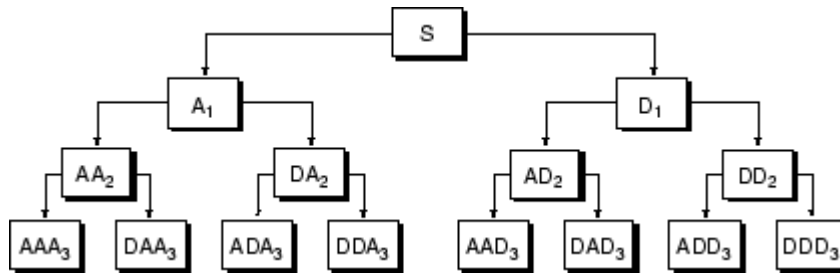
For a given orthogonal wavelet function, we generate a library of bases called *wavelet packet bases*. Each of these bases offers a particular way of coding signals, preserving global energy, and reconstructing exact features. The wavelet packets can be used for numerous expansions of a given signal. 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

In the orthogonal wavelet decomposition procedure, the generic step splits the approximation coefficients into two parts. After splitting we obtain a vector of approximation coefficients and a vector of detail coefficients, both at a coarser scale. The information lost between two successive approximations is captured in the detail coefficients. Then the next step consists of splitting the new approximation coefficient vector; successive details are never reanalyzed.

In the corresponding wavelet packet situation, each detail coefficient vector is also decomposed into two parts using the same approach as in approximation vector splitting. This offers the richest analysis: the complete binary tree is produced as shown in the following figure.



Wavelet Packet Decomposition Tree at Level 3

The idea of this decomposition is to start from a scale-oriented decomposition, and then to analyze the obtained signals on frequency subbands.

Wavelet Packets in Action: An Introduction

The following simple examples illustrate certain differences between wavelet analysis and wavelet packet analysis.

Wavelet Packet Spectrum

The spectral analysis of wide-sense stationary signals using the Fourier transform is well-established. For nonstationary signals, there exist local Fourier methods such as the short-time Fourier transform (STFT). See “Short-Time Fourier Transform” for a brief description.

Because wavelets are localized in time and frequency, it is possible to use wavelet-based counterparts to the STFT for the time-frequency analysis of nonstationary signals. For example, it is possible to construct the scalogram (*wscalogram*) based on the continuous wavelet transform (CWT). However, a potential drawback of using the CWT is that it is computationally expensive.

The discrete wavelet transform (DWT) permits a time-frequency decomposition of the input signal, but the degree of frequency resolution in the DWT is typically considered too coarse for practical time-frequency analysis.

As a compromise between the DWT- and CWT-based techniques, wavelet packets provide a computationally-efficient alternative with sufficient frequency resolution. You can use `wpspectrum` to perform a time-frequency analysis of your signal using wavelet packets.

The following examples illustrate the use of wavelet packets to perform a local spectral analysis. The following examples also use `spectrogram` from the Signal Processing Toolbox software as a benchmark to compare against the wavelet packet spectrum. If you do not have the Signal Processing Toolbox software, you can simply run the wavelet packet spectrum examples.

Wavelet packet spectrum of a sine wave.

```
fs = 1000; % sampling rate
t = 0:1/fs:2; % 2 secs at 1kHz sample rate
y = sin(256*pi*t); % sine of period 128
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');
```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```
figure;
window = 128;
window = hanning(window);
nfft = window;
noverlap = window-1;
[S,F,T] = spectrogram(y,window,noverlap,nfft,fs);
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')
```

Sum of two sine waves with frequencies of 64 and 128 hertz.

```
fs = 1000;
t = 0:1/fs:2;
y = sin(128*pi*t) + sin(256*pi*t); % sine of periods 64 and 128.
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');
```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```

figure;
windowSize = 128;
window = hanning(windowSize);
nfft = windowSize;
noverlap = windowSize-1;
[S,F,T] = spectrogram(y,window,noverlap,nfft,fs);
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')

```

Signal with an abrupt change in frequency from 16 to 64 hertz at two seconds.

```

fs = 500;
t = 0:1/fs:4;
y = sin(32*pi*t).*(t<2) + sin(128*pi*t).*(t>=2);
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');

```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```

figure;
windowSize = 128;
window = hanning(windowSize);
nfft = windowSize;
noverlap = windowSize-1;
[S,F,T] = spectrogram(y,window,noverlap,nfft,fs);
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')

```

Wavelet packet spectrum of a linear chirp.

```

fs = 1000;
t = 0:1/fs:2;
y = sin(256*pi*t.^2);
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');

```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```
figure;
windowSize = 128;
window = hanning(windowSize);
nfft = windowSize;
noverlap = windowSize-1;
[S,F,T] = spectrogram(y,window,noverlap,nfft,fs);
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')
```

Wavelet packet spectrum of quadratic chirp.

```
y = wnoise('quadchirp',10);
len = length(y);
t = linspace(0,5,len);
fs = 1/t(2);
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');
```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```
windowSize = 128;
window = hanning(windowSize);
nfft = windowSize;
noverlap = windowSize-1;
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')
```

Building Wavelet Packets

The computation scheme for wavelet packets generation is easy when using an orthogonal wavelet. We start with the two filters of length $2N$, where $h(n)$ and $g(n)$, correspond to the wavelet.

Now by induction let us define the following sequence of functions:

$$(W_n(x), \quad n \quad = \quad 0, \quad 1, \quad 2, \quad \dots)$$

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 $W_0(x) = \varphi(x)$ is the scaling function and $W_1(x) = \psi(x)$ is the wavelet function.

For example for the Haar wavelet we have

$$N = 1, h(0) = h(1) = \frac{1}{\sqrt{2}}$$

and

$$g(0) = -g(1) = \frac{1}{\sqrt{2}}$$

The equations become

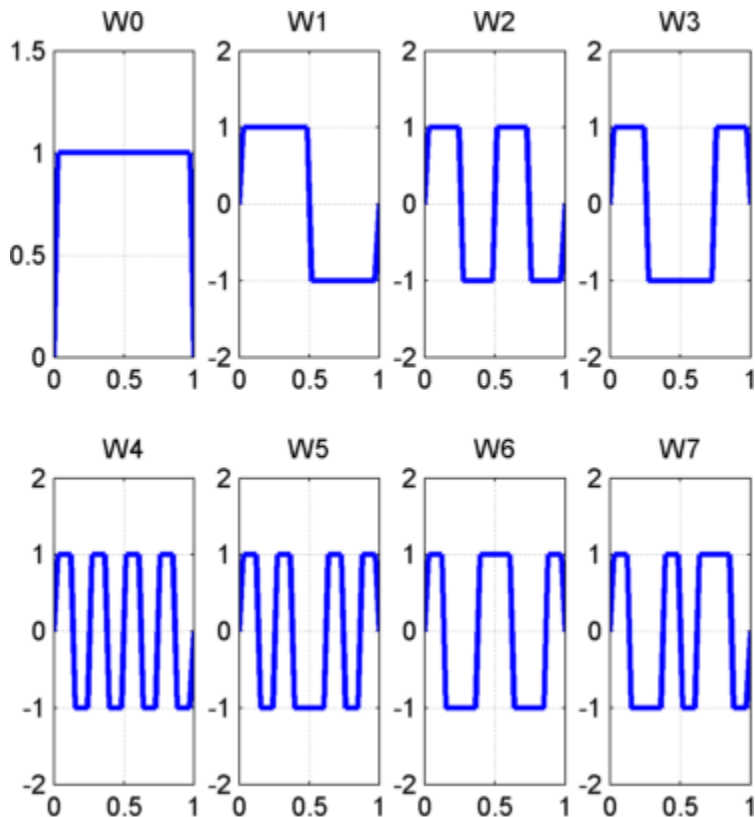
$$W_{2n}(x) = W_n(2x) + W_n(2x - 1)$$

and

$$W_{2n+1}(x) = W_n(2x) - W_n(2x - 1)$$

$W_0(x) = \varphi(x)$ is the *Haar* scaling function and $W_1(x) = \psi(x)$ is the Haar wavelet, both supported in $[0, 1]$. Then we can obtain W_{2n} by adding two $1/2$ -scaled versions of W_n with distinct supports $[0, 1/2]$ and $[1/2, 1]$ and obtain W_{2n+1} by subtracting the same versions of W_n .

For $n = 0$ to 7 , we have the W -functions shown in the figure “Haar Wavelet Packets” on page 5-38.



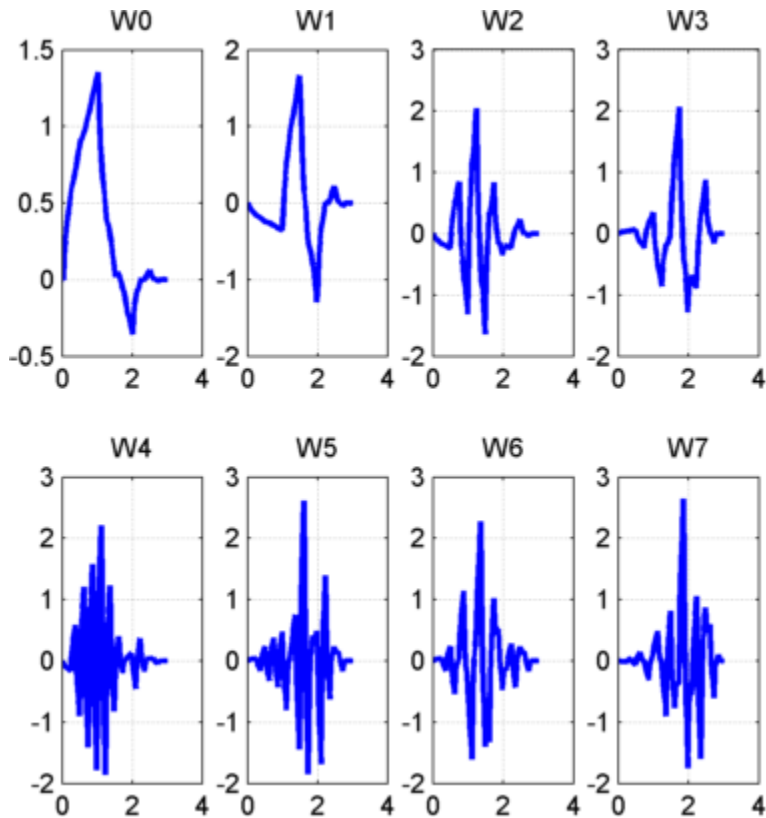
Haar Wavelet Packets

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 $[0, 2N-1]$. The figure “db2 Wavelet Packets” on page 5-39 presents the system of W -functions for the original db2 wavelet.



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 $(j,k) \in Z^2$.

As in the wavelet framework, k can be interpreted as a time-localization parameter and j as a scale parameter. So what is the interpretation of n ?

The basic idea of the wavelet packets is that for fixed values of j and k , $W_{j,n,k}$ analyzes the fluctuations of the signal roughly around the position $2^j \cdot k$, at the scale 2^j and at various frequencies for the different admissible values of the last parameter n .

In fact, examining carefully the wavelet packets displayed in “Haar Wavelet Packets” on page 5-38 and “db2 Wavelet Packets” on page 5-39, 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.

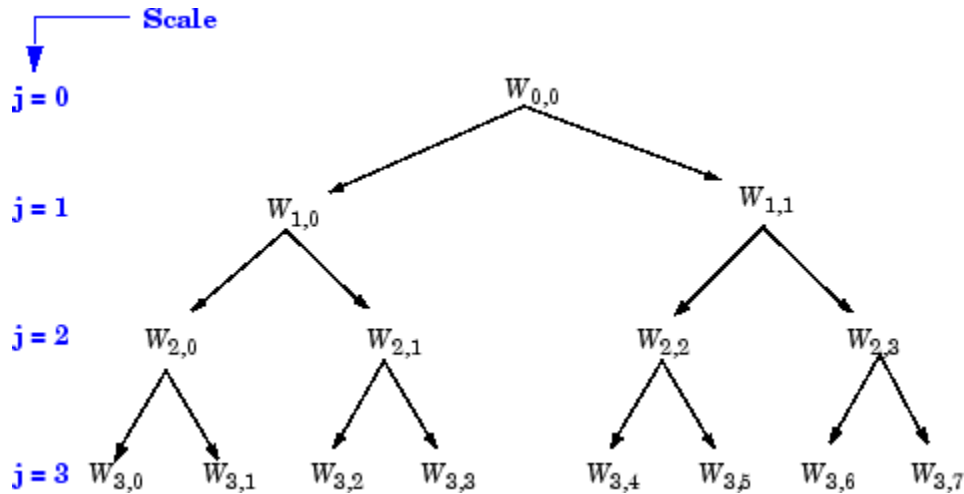
To analyze a signal (the chirp of Example 2 for instance), it is better to plot the wavelet packet coefficients following the frequency order from the low frequencies at the bottom to the high frequencies at the top, rather than naturally ordered coefficients.

When plotting the coefficients, the various options related to the “Frequency” or “Natural” order choice are available using the Wavelet Analyzer app.

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 the figure “Wavelet Packets Organized in a Tree; Scale j Defines Depth and Frequency n Defines Position in the Tree” on page 5-41 is created to give a maximum level decomposition equal to 3. For each scale j , the possible values of parameter n are $0, 1, \dots, 2^j - 1$.



Wavelet Packets Organized in a Tree; Scale j Defines Depth and Frequency n Defines Position in the Tree

The notation $W_{j,n}$, where j denotes scale parameter and n the frequency parameter, is consistent with the usual depth-position tree labeling.

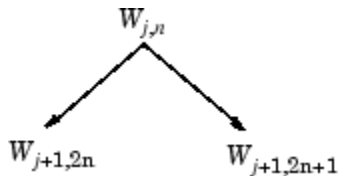
We have $W_{0,0} = (\phi(x - k), k \in Z)$, and $W_{1,1} = (\psi(\frac{x}{2} - k), k \in Z)$.

It turns out that the library of wavelet packet bases contains the wavelet basis and also several other bases. Let us have a look at some of those bases. More precisely, let V_0 denote the space (spanned by the family $W_{0,0}$) in which the signal to be analyzed lies; then $(W_{d,1}; d \geq 1)$ is an orthogonal basis of V_0 .

For every strictly positive integer D , $(W_{D,0}, (W_{d,1}; 1 \leq d \leq D))$ is an orthogonal basis of V_0 .

We also know that the family of functions $\{(W_{j+1,2n}), (W_{j+1,2n+1})\}$ is an orthogonal basis of the space spanned by $W_{j,n}$, which is split into two subspaces: $W_{j+1,2n}$ spans the first subspace, and $W_{j+1,2n+1}$ the second one.

This last property gives a precise interpretation of splitting in the wavelet packet organization tree, because all the developed nodes are of the form shown in the figure “Wavelet Packet Tree: Split and Merge” on page 5-42.



Wavelet Packet Tree: Split and Merge

It follows that the leaves of every connected binary subtree of the complete tree correspond to an orthogonal basis of the initial space.

For a finite energy signal belonging to V_0 , any wavelet packet basis will provide exact reconstruction and offer a specific way of coding the signal, using information allocation in frequency scale subbands.

Choosing the Optimal Decomposition

Based on the organization of the wavelet packet library, it is natural to count the decompositions issued from a given orthogonal wavelet.

A signal of length $N = 2^L$ can be expanded in α different ways, where α is the number of binary subtrees of a complete binary tree of depth L . As a result, $\alpha \geq 2^{N/2}$ (see [Mal98] page 323).

As this number may be very large, and since explicit enumeration is generally unmanageable, it is interesting to find an optimal decomposition with respect to a convenient criterion, computable by an efficient algorithm. We are looking for a minimum of the criterion.

Functions verifying an additivity-type property are well suited for efficient searching of binary-tree structures and the fundamental splitting. Classical entropy-based criteria match these conditions and describe information-related properties for an accurate representation of a given signal. Entropy is a common concept in many fields, mainly in signal processing. Let us list four different entropy criteria (see [CoiW92]); many others are available and can be easily integrated (type `help entropy`). 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 $1 \leq p \leq \infty$

$$E2(s_i) = |s_i|^p$$

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 ε .

These entropy functions are available using the `wentropy` file.

Example 1: Compute Various Entropies

- 1 Generate a signal of energy equal to 1.
 $s = \text{ones}(1, 16) * 0.25;$
- 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,  $w11$ , is zero; the entropy  $e11$  is zero. Due to the additivity property the entropy of decomposition is given by  $e10+e11=2.0794$ . This has to be compared to the initial entropy  $e00=2.7726$ . We have  $e10 + e11 < e00$ , so the splitting is interesting.

- 5 Now split  $w10$  (not  $w11$  because the splitting of a null vector is without interest since the entropy is zero).

```
[w20,w21] = dwt(w10,'db1');
```

- 6** We have  $w_{20}=0.5*\text{ones}(1,4)$  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 “Entropy Values” on page 5-46 shows the nodes labeled with original entropy numbers.



The complete binary tree of depth  $D$  corresponding to a wavelet packet decomposition tree developed at level  $D$  is denoted by WPT.

We have the following interesting subtrees.

| Decomposition Tree                         | Subtree Such That the Set of Leaves Is a Basis     |
|--------------------------------------------|----------------------------------------------------|
| Wavelet packets decomposition tree         | Complete binary tree: WPT of depth $D$             |
| Wavelet packets optimal decomposition tree | Binary subtree of WPT                              |
| Wavelet packets best-level tree            | Complete binary subtree of WPT                     |
| Wavelet decomposition tree                 | Left unilateral binary subtree of WPT of depth $D$ |
| Wavelet best-basis tree                    | Left unilateral binary subtree of WPT              |

We deduce the following definitions of optimal decompositions, with respect to an entropy criterion  $E$ .

| Decompositions                | Optimal Decomposition    | Best-Level Decomposition |
|-------------------------------|--------------------------|--------------------------|
| Wavelet packet decompositions | Search among $2^D$ trees | Search among $D$ trees   |
| Wavelet decompositions        | Search among $D$ trees   | Search among $D$ trees   |

For any nonterminal node, we use the following basic step to find the optimal subtree with respect to a given entropy criterion  $E$  (where  $E_{opt}$  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  $t$ .

### Reconstructing a Signal Approximation from a Node

You can use the function `wprcoef` to reconstruct an approximation to your signal from any node in the wavelet packet tree. This is true irrespective of whether you are working with a full wavelet packet tree, or a subtree determined by an optimality criterion. Use

`wpccoef` if you want to extract the wavelet packet coefficients from a node without reconstructing an approximation to the signal.

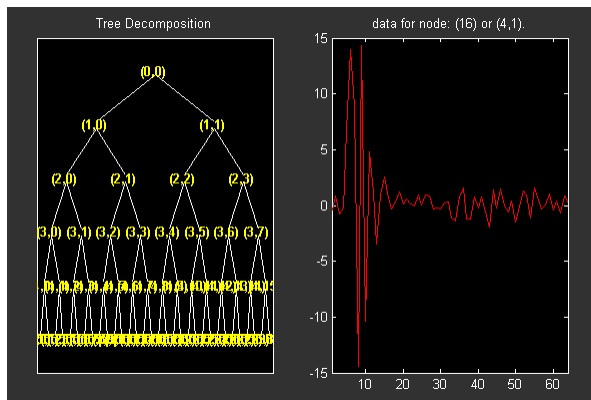
Load the noisy Doppler signal.

```
load noisdopp
```

Compute the wavelet packet decomposition down to level 5 using the `sym4` wavelet. Use the periodization mode.

```
dwtmode('per');
T = wpdec(noisdopp,5,'sym4');
plot(T)
```

Plot the binary wavelet packet tree and click on the (4,1) doublet (node 16).



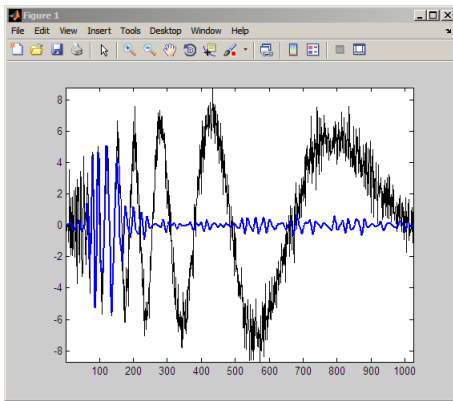
Extract the wavelet packet coefficients from node 16.

```
wpc = wpccoef(T,16);
% wpc is length 64
```

Obtain an approximation to the signal from node 16.

```
rwpc = wprcoef(T,16);
% rwpc is length 1024
plot(noisdopp,'k'); hold on;
plot(rwpc,'b','linewidth',2);
axis tight;
```



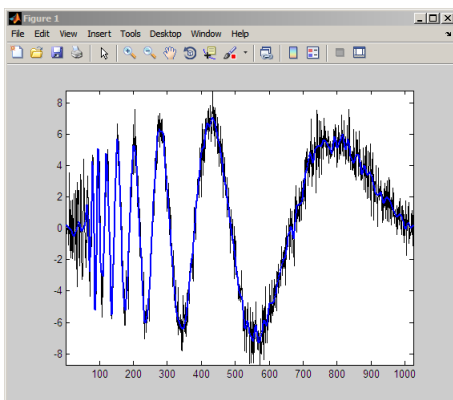


Determine the optimum binary wavelet packet tree.

```
Topt = besttree(T);
% plot the best tree
plot(Topt)
```

Reconstruct an approximation to the signal from the (3,0) doublet (node 7).

```
rsig = wprcoef(Topt,7);
% rsig is length 1024
plot(noisdopp,'k'); hold on;
plot(rsig,'b','linewidth',2);
axis tight;
```



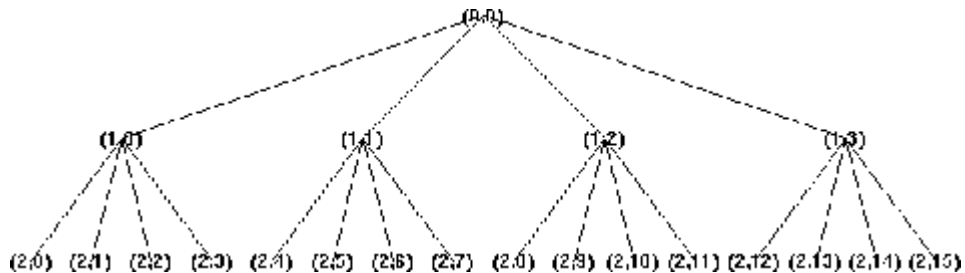
If you know which doublet in the binary wavelet packet tree you want to extract, you can determine the node corresponding to that doublet with `depo2ind`.

For example, to determine the node corresponding to the doublet (3,0), enter:

```
Node = depo2ind(2,[3 0]);
```

## Wavelet Packets 2-D Decomposition Structure

Exactly as in the wavelet decomposition case, the preceding 1-D framework can be extended to image analysis. Minor direct modifications lead to quaternary tree-related definitions. An example is shown the following figure for depth 2.



### Quaternary Tree of Depth 2

## Wavelet Packets for Compression and Denoising

In the wavelet packet framework, compression and denoising 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 `besttree` with an entropy function.

## 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 Wavelet Analyzer app without knowing about the underlying objects and classes. But, it is useful for **Save** and **Load** actions where objects are involved.

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.

## 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.

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.

## Examples Using Wavelet Packet Tree Objects

You can use command line functions, the Wavelet Analyzer app, 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

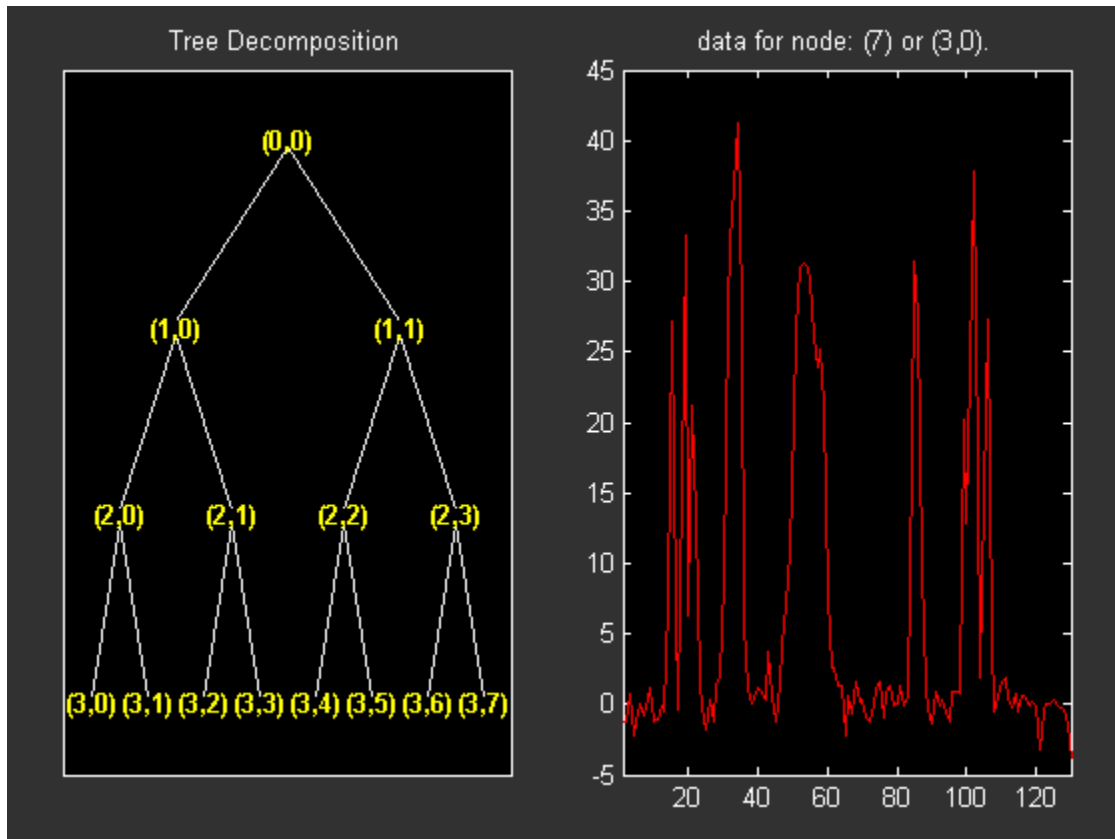
We can see some of these features in the following examples.

- “`plot` and `wpviewcf`” on page 5-53
- “`drawtree` and `readtree`” on page 5-57
- “Change Terminal Node Coefficients” on page 5-59
- “Thresholding Wavelet Packets” on page 5-61

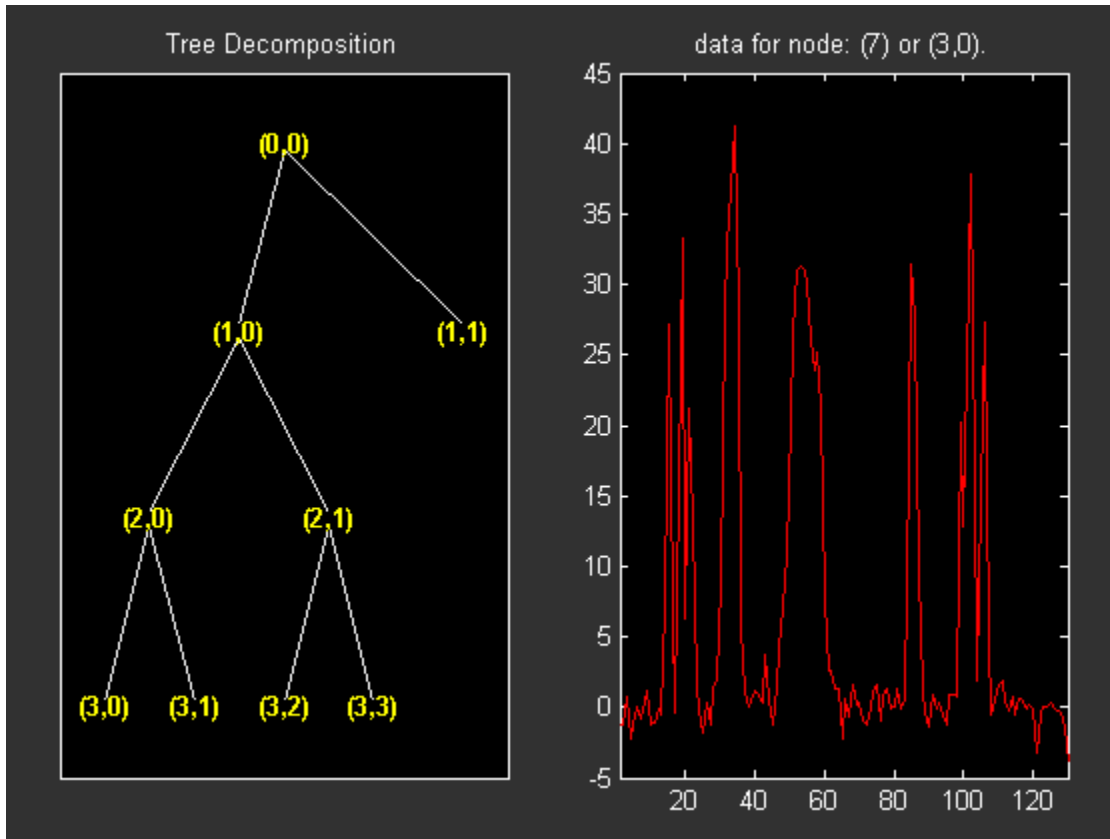
### **plot and wpviewcf**

```
load noisbump
x = noisbump;
t = wpdec(x,3,'db2');
fig = plot(t);
```

Click on node 7.



Change Node Action from **Visualize** to **Split-Merge** and merge the second node.



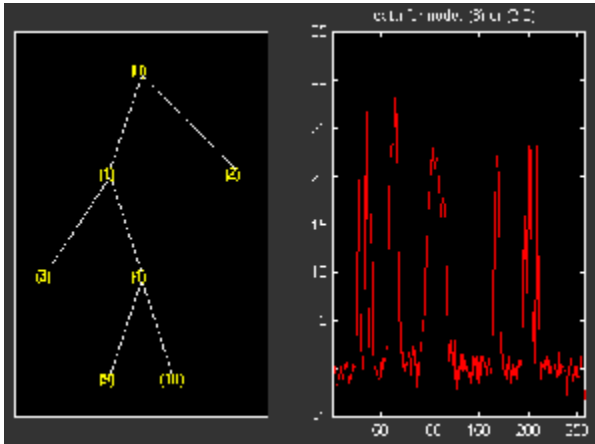
```
% From the command line, you can get the new tree.
newt = plot(t,'read',fig);
```

```
% 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.
```

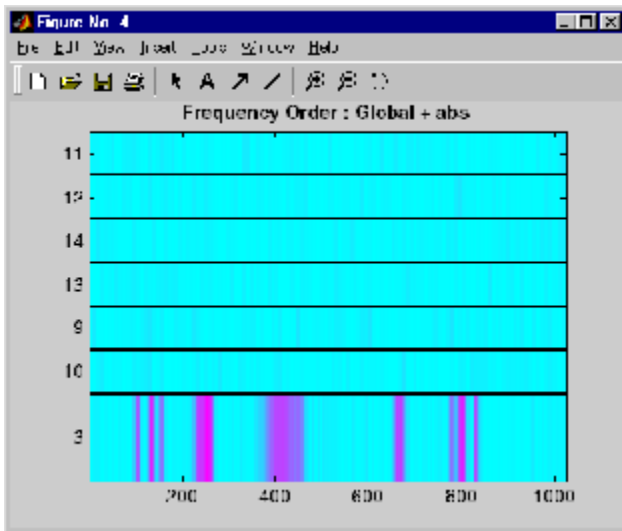


```
% Using plot(newt,fig), the plot is done in the figure fig,
% which already contains a tree object.

% 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.
```

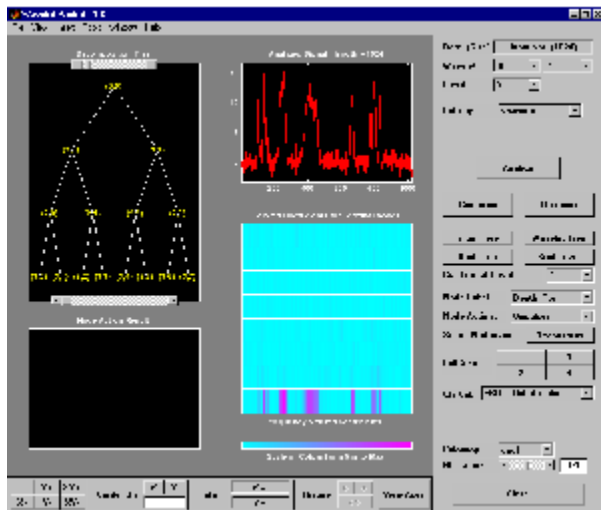




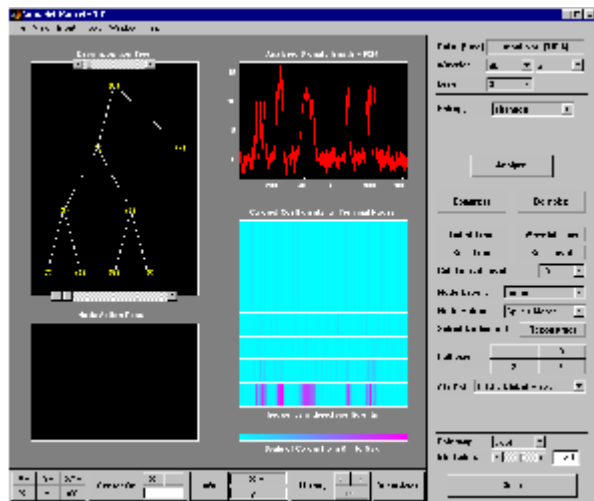
## drawtree and readtree

```
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 waveletAnalyzer 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.
```

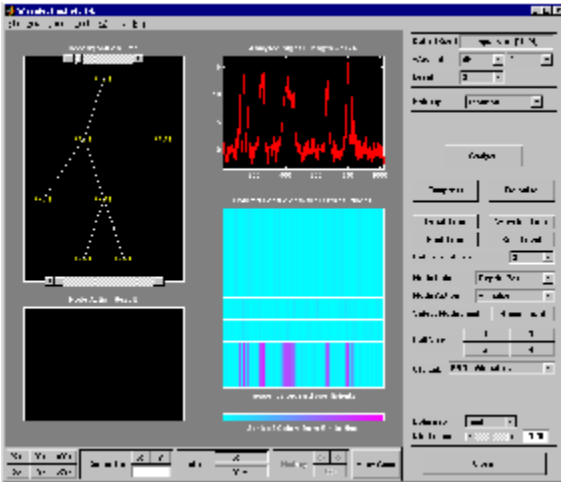


- % From the app, you can modify the tree.
- % For example, change Node label from Depth\_Position to Index,
- % change Node Action from Visualize to Split\_Merge and
- % merge the node 2.
- % You get the following figure.



```
% 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 portions of the Wavelet Analyzer app related to wavelet packets.

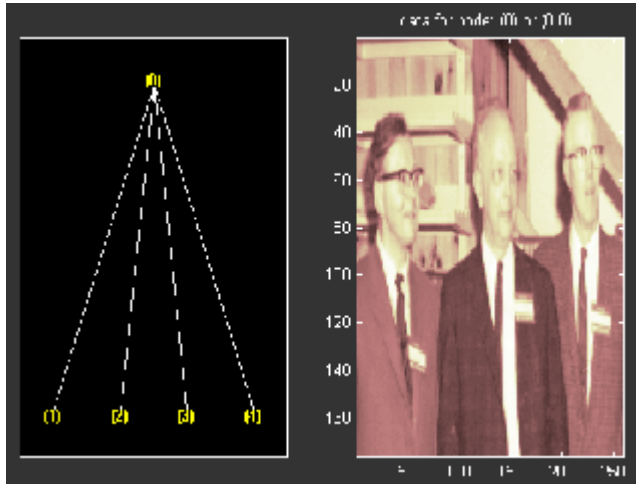
The methods associated with WPTREE objects let you do more complicated actions.

Namely, using `read` and `write` methods, you can change terminal node coefficients.

Let's illustrate this point with the following “funny” example.

## Change Terminal Node Coefficients

```
load gatin2
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.
```



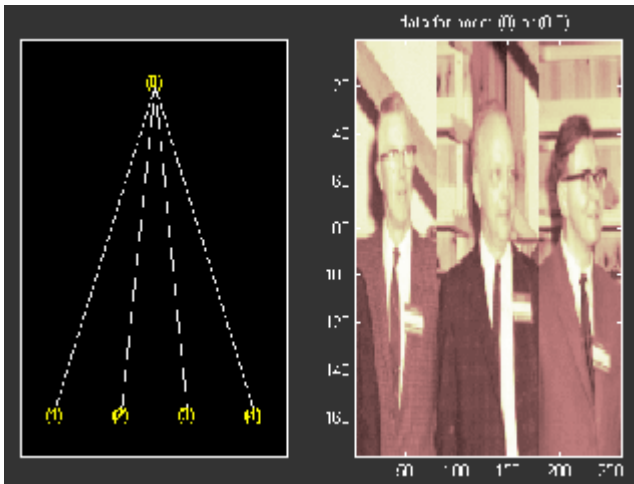
```

% 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.

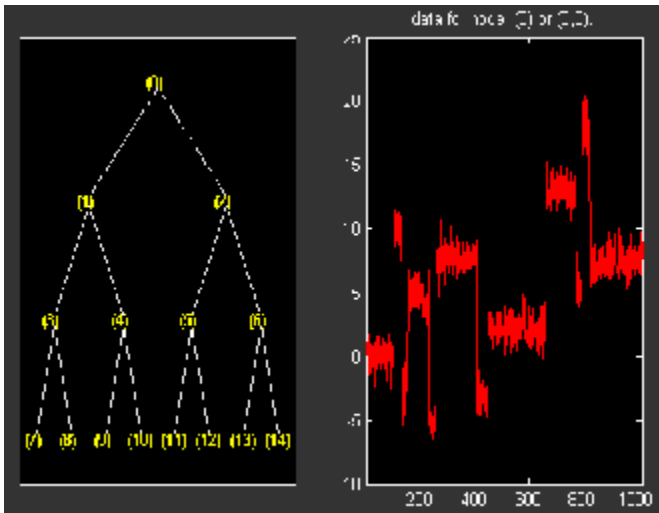
```



You can use this method for a more useful purpose. Let's see a denoising example.

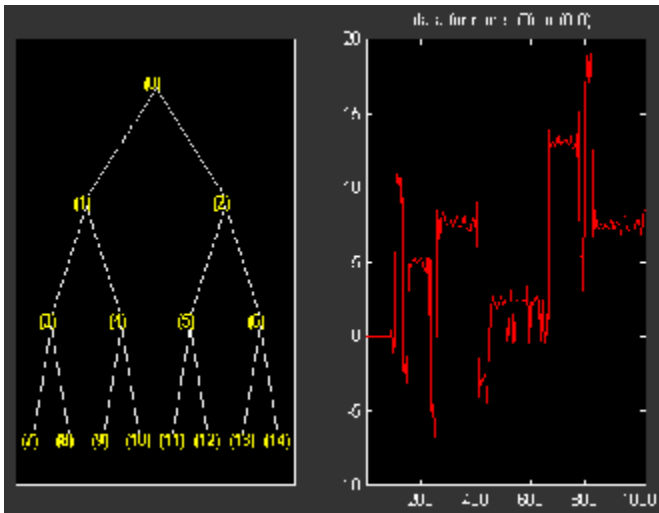
## Thresholding Wavelet Packets

```
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('wplddenoGBL','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.
```

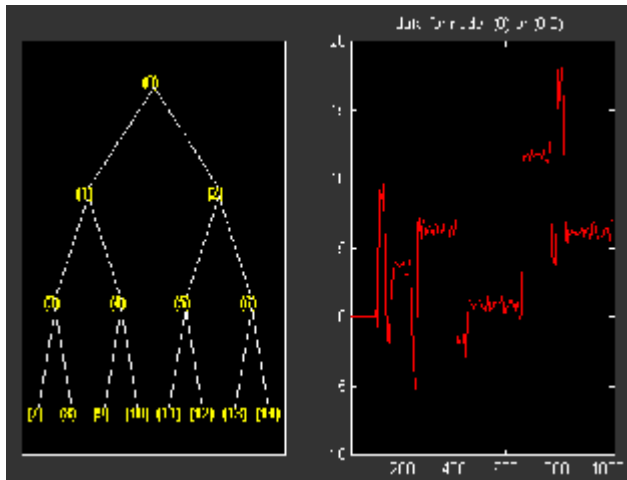


```

% Node by node thresholding.
t2 = t;
sorh = 's';
thr(1) = wthrmngr('wplddenoGBL','penalhi',t);
thr(2) = wthrmngr('wplddenoGBL','sqtwologswn',t);
tn = leaves(t);
for k=1:length(tn)
 node = tn(k);
 cfs = read(t,'data',node);
 numthr = rem(node,2)+1;
 cfs = wthresh(cfs,sorh,thr(numthr));
 t2 = write(t2,'data',node,cfs);
end
plot(t2)

% Change Node Label from Depth_position to Index and
% click the node (0). You get the following plot.

```





## Description of Objects in the Wavelet Toolbox Software

The following sections describe the objects in the Wavelet Toolbox software:

- “WTBO Object” on page 5-65
- “NTREE Object” on page 5-66
- “DTREE Object” on page 5-66
- “WPTREE Object” on page 5-68

### WTBO Object

Class **WTBO** (Wavelet Toolbox Object) -- Parent class: none

#### Fields

|          |                               |
|----------|-------------------------------|
| wtboInfo | Object information (Not used) |
| ud       | Userdata field                |

#### Methods

|      |                                 |
|------|---------------------------------|
| wtbo | Constructor for the class WTBO. |
| get  | Get WTBO object field contents. |
| set  | Set WTBO object field contents. |

#### Comments

Since any object in the toolbox is parented by a WTBO object, you can associate your own data to an object using the 'ud' field, and then access it.

If Obj is an object (parented by a WTBO object), use

```
Obj = set(Obj, 'ud', MyData)
```

to define the data.

To retrieve the data, use

```
MyData = get(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 NTREE. |
| findactn | Find active nodes.               |
| get      | Get NTREE object field contents. |
| nodejoin | Recompose node(s).               |
| nodesplt | Split (decompose) node(s).       |
| plot     | Plot NTREE object.               |
| set      | Set NTREE object field contents. |
| tlabels  | Labels for the nodes of a tree.  |
| wtreemgr | Manager for NTREE object.        |

### Private

|          |                               |
|----------|-------------------------------|
| locnumcn | Local number for a child node |
| tabofasc | Table of ascendants of nodes  |

## DTREE Object

Class **DTREE** (Data Tree) -- Parent class: **NTREE**

**Fields**

|                    |                            |
|--------------------|----------------------------|
| <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

`allNI(N,:) = [ind,size(1,1),size(1,2)]`

- `ind` = index of the node `N`
- `size` = size of data associated with the node `N`

`terNI` is a `1-by-2` cell array such that

- `terNI{1}` is an `NB_TerminalNodes-by-2` array such that
  - `terNI{1}(N, :)` is the size of coefficients associated with the `N`-th terminal node. The nodes are numbered from left to right and from top to bottom. The root index is 0.
- `terNI{2}` is a row vector containing the previous coefficients stored row-wise in the above specified order.

**Methods**

|                       |                                                |
|-----------------------|------------------------------------------------|
| <code>dtree</code>    | Constructor for the class <code>DTREE</code> . |
| <code>expand</code>   | Expand data tree.                              |
| <code>fmdtree</code>  | Field manager for <code>DTREE</code> 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 <code>DTREE</code> object field contents.  |

|                     |                                                |
|---------------------|------------------------------------------------|
| <code>plot</code>   | Plot DTREE object.                             |
| <code>read</code>   | Read values in DTREE object fields.            |
| <code>set</code>    | Set DTREE object field contents.               |
| <code>write</code>  | Write values in DTREE 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**

|         |                                 |
|---------|---------------------------------|
| dtree   | Parent object                   |
| wavInfo | Structure (wavelet information) |
| entInfo | Structure (entropy information) |

**Fields Description**

## wavInfo

|         |                            |
|---------|----------------------------|
| wavName | Wavelet Name               |
| Lo_D    | Low Decomposition filter   |
| Hi_D    | High Decomposition filter  |
| Lo_R    | Low Reconstruction filter  |
| Hi_R    | High Reconstruction filter |

## entInfo

|         |                   |
|---------|-------------------|
| entName | Entropy Name      |
| entPar  | Entropy Parameter |

allNI Array(nbnode,5) (field of the dtree parent object)

[ind,size,ent,ento]

|      |                 |
|------|-----------------|
| ind  | Index           |
| size | Size of data    |
| ent  | Entropy         |
| ento | Optimal Entropy |

**Methods****Constructor**

| Method | Description                             |
|--------|-----------------------------------------|
| wptree | Constructor for the class <b>WPTREE</b> |

**Methods That Overload Those of DTREE Class**

| <b>Method</b> | <b>Description</b>                             |
|---------------|------------------------------------------------|
| defaninf      | Define node information (all nodes).           |
| get           | Get WPTREE object field contents.              |
| merge         | Merge (recompose) the data of a node.          |
| read          | Read values in WPTREE object fields.           |
| recons        | Reconstruct wavelet packet coefficients.       |
| set           | Set WPTREE object field contents.              |
| split         | Split (decompose) the data of a terminal node. |
| tlabels       | Labels for the nodes of a wavelet packet tree. |
| write         | Write values in WPTREE object fields.          |

**Proper Methods of WPTREE Class**

| <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. |
| wpcoef        | Wavelet packet coefficients.                   |
| wpcutree      | Cut wavelet packet tree.                       |
| wpjoin        | Recompose wavelet packet.                      |
| wplotcf       | 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.     |

## Build Wavelet Tree Objects

The following sections explain how to extend the toolbox with new objects through four examples.

- “Building a Wavelet Tree Object (WTREE)” on page 5-71
- “Building a Right Wavelet Tree Object (RWVTREE)” on page 5-72
- “Building a Wavelet Tree Object (WVTREE)” on page 5-73
- “Building a Wavelet Tree Object (EDWTTREE)” on page 5-75

### 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 1-D example in the `ex1_wt` file and at a 2-D example in the `ex2_wt` file located in the `toolbox/wavelet/wavelet` folder. These examples can be used directly, but they are also useful to learn how to build new object-oriented programming functions.

The definition of the new class is described below.

Class **WTREE** (parent class: **DTREE**)

#### Fields

|                      |                                 |
|----------------------|---------------------------------|
| <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 |

|      |                            |
|------|----------------------------|
| Hi_D | High Decomposition filter  |
| Lo_R | Low Reconstruction filter  |
| Hi_R | High Reconstruction filter |

**Methods**

|       |                                                |
|-------|------------------------------------------------|
| wtree | Constructor for the class WTREE.               |
| merge | Merge (recompose) the data of a node.          |
| split | Split (decompose) the data of a terminal node. |

**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 1-D example in the `ex1_rwvt` file and at a 2-D example in the `ex2_rwvt` file located in the `toolbox/wavelet/wavelet` folder. These programs can be used directly, but they are also useful to learn how to build new object-oriented programming functions.

The definition of the new class is described below.

Class **RWVTREE** (parent class: **WTREE**)

**Fields**

|       |               |
|-------|---------------|
| dummy | Not used      |
| wtree | Parent object |

**Methods**

|         |                                       |
|---------|---------------------------------------|
| rwvtree | Constructor for the class RWVTREE.    |
| merge   | Merge (recompose) the data of a node. |

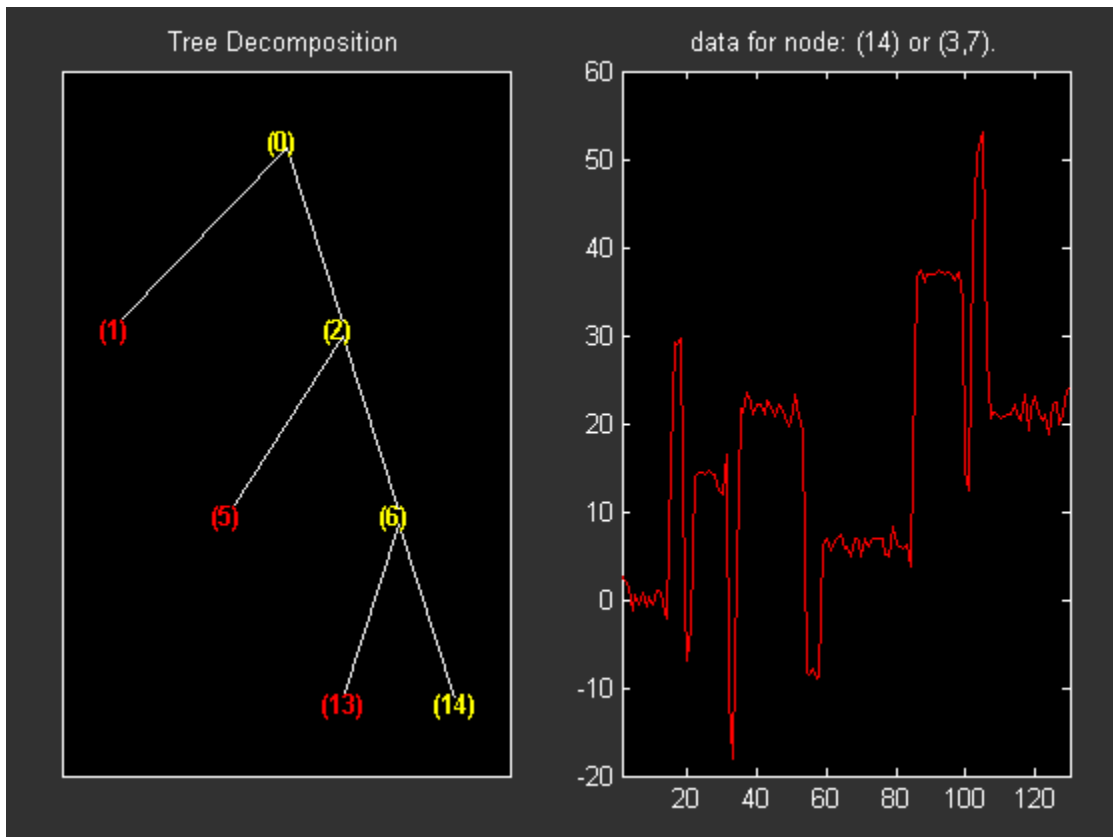


|       |                                                |
|-------|------------------------------------------------|
| plot  | Plot RWVTREE object.                           |
| split | Split (decompose) the data of a terminal node. |

### Running This Example

The following figure is obtained using the example `ex1_rwvt` and clicking the node 14.

The approximations are labeled in and the details are labeled in red. The last nodes cannot be split.



### 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 1-D example in the `ex1_wvt` file and at a 2-D example in the `ex2_wvt` file located in the `toolbox/wavelet/wavelet` folder. These programs can be used directly, but they are also useful to learn how to build new object-oriented programming functions.

The definition of the new class is described below.

Class **WVTREE** (parent class: **WTREE**)

### Fields

|                    |               |
|--------------------|---------------|
| <code>dummy</code> | Not used      |
| <code>wtree</code> | Parent object |

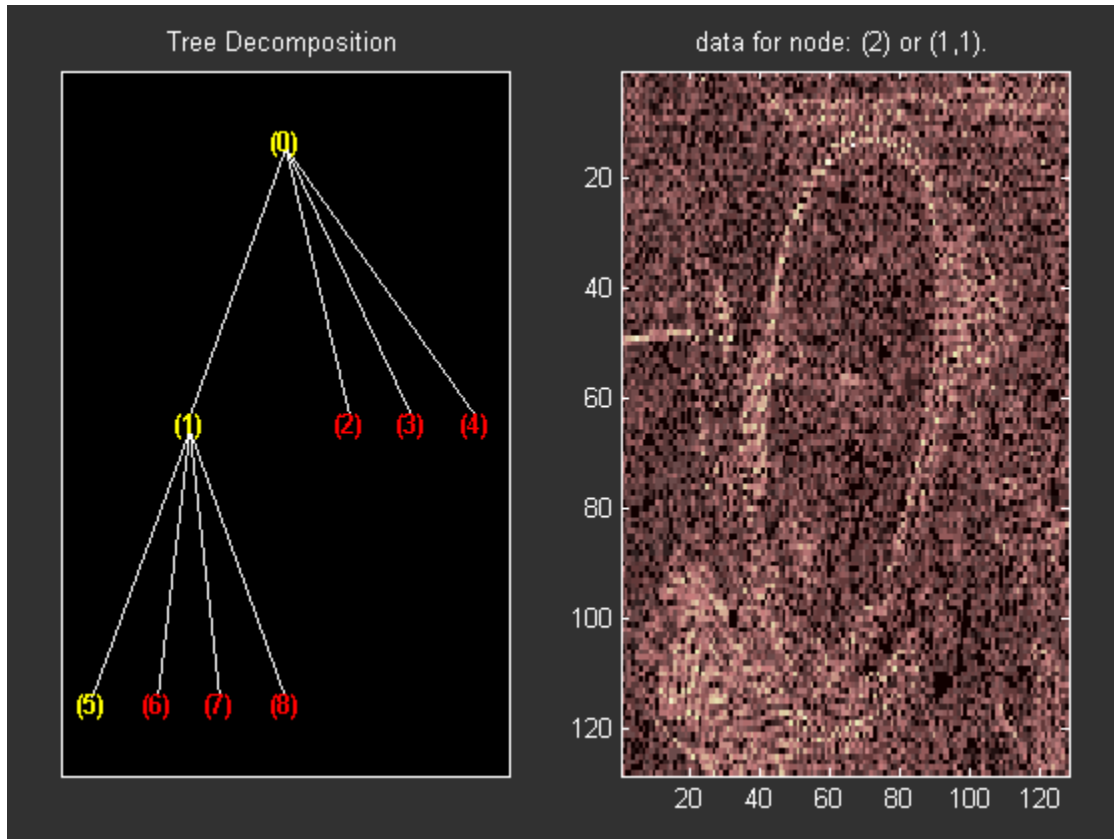
### Methods

|                     |                                                 |
|---------------------|-------------------------------------------------|
| <code>wvtree</code> | Constructor for the class <code>WVTREE</code> . |
| <code>get</code>    | Get <code>WVTREE</code> object field contents.  |
| <code>plot</code>   | Plot <code>WVTREE</code> 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 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).



## Building a Wavelet Tree Object (EDWTTREE)

This example creates a new class of objects: **EDWTTREE**.

We define an  $\epsilon$ -DWT tree class starting from the class DTREE and overloading the methods `merge`, `plot`, `recons`, and `split`.

For more information on the  $\epsilon$ -DWT, see the section “-Decimated DWT” on page 3-66.

The `plot` method shows how to add **Node Labels**, **Node Actions**, and **Tree Actions**.

You can have a look at the example in the `ex1_edwt` file located in the `toolbox/wavelet/wavelet` folder. This program can be used directly, but it is also useful to learn how to build new object-oriented programming functions.

The definition of the new class is described below.

Class **EDWTTREE** (parent class: **DTREE**)

### Fields

|                      |                                 |
|----------------------|---------------------------------|
| <code>dtree</code>   | Parent object                   |
| <code>dwtMode</code> | DWT extension mode              |
| <code>wavInfo</code> | Structure (wavelet 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 |

### Methods

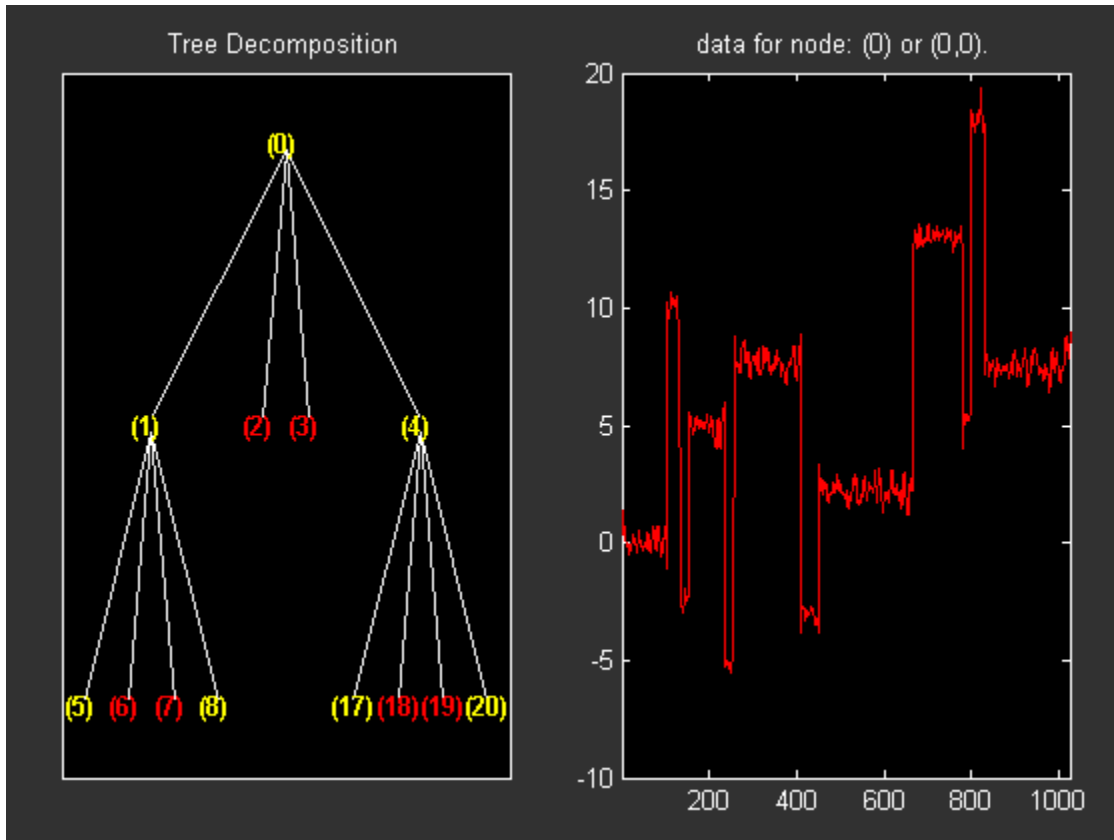
|                       |                                                |
|-----------------------|------------------------------------------------|
| <code>edwttree</code> | Constructor for the class <i>EDWTTREE</i> .    |
| <code>merge</code>    | Merge (recompose) the data of a node.          |
| <code>plot</code>     | Plot <i>EDWTTREE</i> object.                   |
| <code>recons</code>   | Reconstruct node coefficients.                 |
| <code>split</code>    | Split (decompose) the data of a terminal node. |

### Running This Example

The following figure is obtained using the example `ex1_edwt`, selecting the **Denoise** option in the **Tree Action** menu and clicking the node 0.

The approximations are labeled in 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 denoising method.





# **Denoising, Nonparametric Function Estimation, and Compression**

---

## Wavelet Denoising and Nonparametric Function Estimation

### In this section...

“Denoising Methods” on page 6-4

“Soft or Hard Thresholding” on page 6-6

“Dealing with Unscaled Noise and Nonwhite Noise” on page 6-7

“Wavelet Denoising in Action” on page 6-8

“Extension to Image Denoising” on page 6-14

“1-D Wavelet Variance Adaptive Thresholding” on page 6-16

“Wavelet Denoising Analysis Measurements” on page 6-20

The Wavelet Toolbox provides a number of functions for the estimation of an unknown function (signal or image) in noise. You can use these functions to denoise signals and as a method for nonparametric function estimation.

The most general 1-D model for this is

$$s(n) = f(n) + \sigma e(n)$$

where  $n = 0, 1, 2, \dots, N-1$ . The  $e(n)$  are Gaussian random variables distributed as  $N(0, 1)$ . The variance of the  $\sigma e(n)$  is  $\sigma^2$ .

In practice,  $s(n)$  is often a discrete-time signal with equal time steps corrupted by additive noise and you are attempting to recover that signal.

More generally, you can view  $s(n)$  as an  $N$ -dimensional random vector

$$\begin{pmatrix} f(0) + \sigma e(0) \\ f(1) + \sigma e(1) \\ f(2) + \sigma e(2) \\ \vdots \\ \vdots \\ \vdots \\ f(N-1) + \sigma e(N-1) \end{pmatrix} = \begin{pmatrix} f(0) \\ f(1) \\ f(2) \\ \vdots \\ \vdots \\ \vdots \\ f(N-1) \end{pmatrix} + \begin{pmatrix} \sigma e(0) \\ \sigma e(1) \\ \sigma e(2) \\ \vdots \\ \vdots \\ \vdots \\ \sigma e(N-1) \end{pmatrix}$$

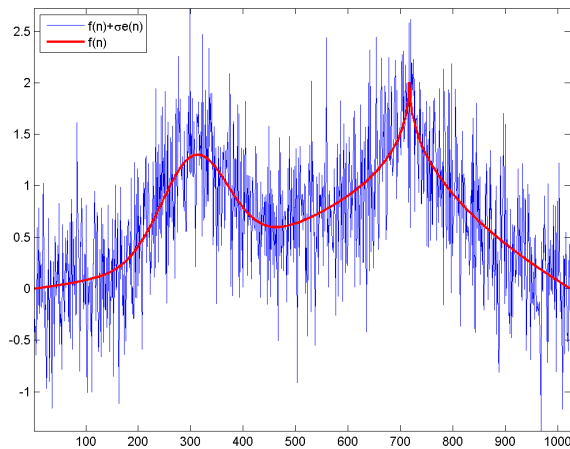
In this general context, the relationship between denoising and regression is clear.



You can replace the N-by-1 random vector by N-by-M random matrices to obtain the problem of recovering an image corrupted by additive noise.

You can obtain a 1-D example of this model with the following code.

```
load cuspamax;
y = cuspamax+0.5*randn(size(cuspamax));
plot(y); hold on;
plot(cuspamax,'r','linewidth',2);
axis tight;
legend('f(n)+\sigma e(n)', 'f(n)', 'Location', 'NorthWest');
```



For a broad class of functions (signals, images) that possess certain smoothness properties, wavelet techniques are optimal or near optimal for function recovery.

Specifically, 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.

The general wavelet-based method for denoising and nonparametric function estimation is to transform the data into the wavelet domain, threshold the wavelet coefficients, and invert the transform.

You can summarize these steps as:

**1** Decompose

Choose a wavelet and a level  $N$ . Compute the wavelet decomposition of the signal  $s$  down to level  $N$ .

**2** Threshold detail coefficients

For each level from 1 to  $N$ , threshold 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  $N$ .

## Denoising Methods

The Wavelet Toolbox supports a number of denoising methods. Four denoising methods are implemented in the `thselect`. Each method corresponds to a `tptr` option in the command

```
thr = thselect(y,tptr)
```

which returns the threshold value.

| Option     | Denoising Method                                                                                 |
|------------|--------------------------------------------------------------------------------------------------|
| 'rigrsure' | Selection using principle of Stein's Unbiased Risk Estimate (SURE)                               |
| 'sqtwolog' | Fixed form (universal) threshold equal to<br>$\sqrt{2\ln(N)}$ with $N$ the length of the signal. |
| 'heursure' | Selection using a mixture of the first two options                                               |
| 'minimaxi' | Selection using minimax principle                                                                |

- Option '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 'sqtwolog' uses a fixed form threshold yielding minimax performance multiplied by a small factor proportional to  $\log(\text{length}(s))$ .

- Option '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 '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 denoised 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.

The following example shows the denoising methods for a 1000-by-1  $N(0,1)$  vector. The signal here is

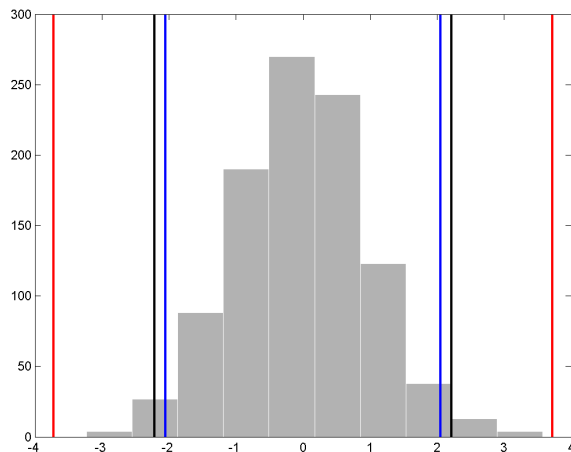
$$f(n) + e(n) \quad e(n) \sim N(0, 1)$$

with  $f(n) = 0$ .

```

rng default;
sig = randn(1e3,1);
thr_rigrsure = thselect(sig,'rigrsure')
thr_univthresh = thselect(sig,'sqtwolog')
thr_heursure = thselect(sig,'heursure')
thr_minimaxi = thselect(sig,'minimaxi')
histogram(sig);
h = findobj(gca,'Type','patch');
set(h,'FaceColor',[0.7 0.7 0.7],'EdgeColor','w');
hold on;
plot([thr_rigrsure thr_rigrsure], [0 300],'linewidth',2);
plot([thr_univthresh thr_univthresh], [0 300],'r','linewidth',2);
plot([thr_minimaxi thr_minimaxi], [0 300],'k','linewidth',2);
plot([-thr_rigrsure -thr_rigrsure], [0 300],'linewidth',2);
plot([-thr_univthresh -thr_univthresh], [0 300],'r','linewidth',2);
plot([-thr_minimaxi -thr_minimaxi], [0 300],'k','linewidth',2);

```



For Stein's Unbiased Risk Estimate (SURE) and minimax thresholds, approximately 3% of coefficients are retained. In the case of the universal threshold, all values are rejected.

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.

After you use `thselect` to determine a threshold, you can threshold each level of a . This second step can be done using `wthcoef`, directly handling the wavelet decomposition structure of the original signal  $s$ .

## Soft or Hard Thresholding

Hard and soft thresholding are examples of *shrinkage* rules. After you have determined your threshold, you have to decide how to apply that threshold to your data.

The simplest scheme is *hard* thresholding. Let  $T$  denote the threshold and  $x$  your data. The hard thresholding is

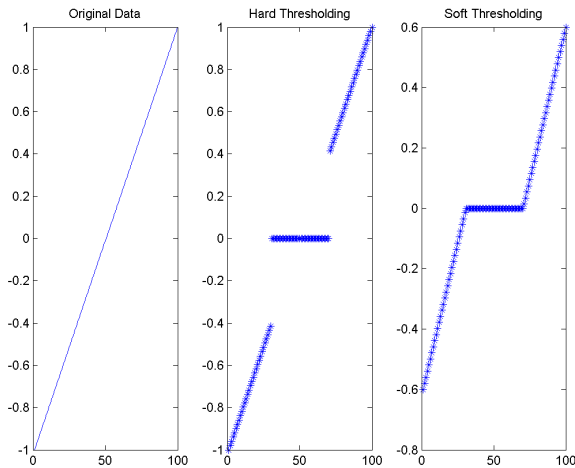
$$\eta(x) = \begin{cases} x & |x| \geq T \\ 0 & |x| < T \end{cases}$$

The soft thresholding is

$$\eta(x) = \begin{cases} x - T & x > T \\ 0 & |x| \leq T \\ x + T & x < -T \end{cases}$$

You can apply your threshold using the hard or soft rule with `wthresh`.

```
y = linspace(-1,1,100);
thr = 0.4;
ythard = wthresh(y,'h',thr);
ytsoft = wthresh(y,'s',thr);
subplot(131);
plot(y); title('Original Data');
subplot(132);
plot(ythard,'*'); title('Hard Thresholding');
subplot(133);
plot(ytsoft,'*'); title('Soft Thresholding');
```



## 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 denoising function `wdenoise`.

The simplest use of `wdenoise` is

```
sd = wdenoise(s)
```

which returns the denoised version `sd` of the original signal `s` obtained by using default settings for parameters including wavelet, denoising method, and soft or hard thresholding. Any of the default settings can be changed:

```
sd = wdenoise(s,n,'DenoisingMethod',tptr,'Wavelet',wav,...
 'ThresholdRule',sorr,'NoiseEstimate',scal)
```

which returns the denoised version `sd` of the original signal `s` obtained using the `tptr` denoising method. Other parameters needed are `sorr`, `scal`, and `wname`. The parameter `sorr` 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 the method of estimating variance of noise in the data.

| Option             | Noise Estimate Method                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------|
| 'LevelIndependent' | 'LevelIndependent' estimates the variance of the noise based on the finest-scale (highest-resolution) wavelet coefficients. |
| 'LevelDependent'   | 'LevelDependent' estimates the variance of the noise based on the wavelet coefficients at each resolution level.            |

For a more general procedure, the `wdencomp` function performs wavelet coefficients thresholding for both denoising and compression purposes, while directly handling 1-D and 2-D data. It allows you to define your own thresholding strategy selecting in

```
xd = wdencomp(opt,x,wav,n,thr,sorr,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 denoised and `wav`, `n`, `sorr` are the same as above.

## Wavelet Denoising in Action

We begin the examples of 1-D denoising methods with the first example credited to Donoho and Johnstone.

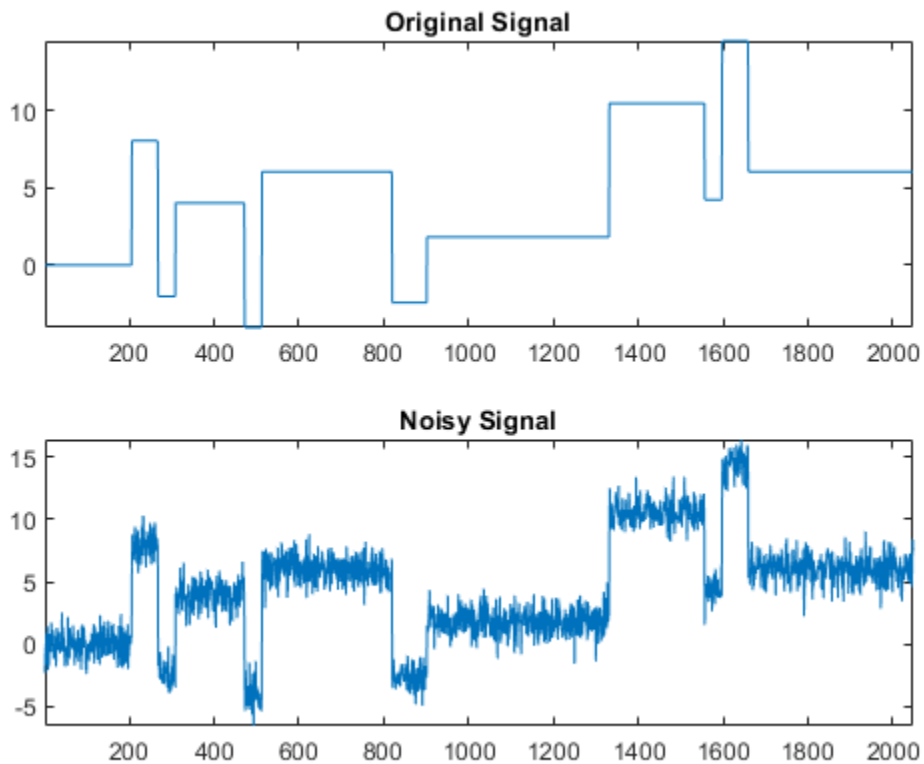
## Blocks Signal Thresholding

First set a signal-to-noise ratio (SNR) and set a random seed.

```
sqrt_snr = 4;
init = 2055615866;
```

Generate an original signal  $x_{ref}$  and a noisy version  $x$  by adding standard Gaussian white noise. Plot both signals.

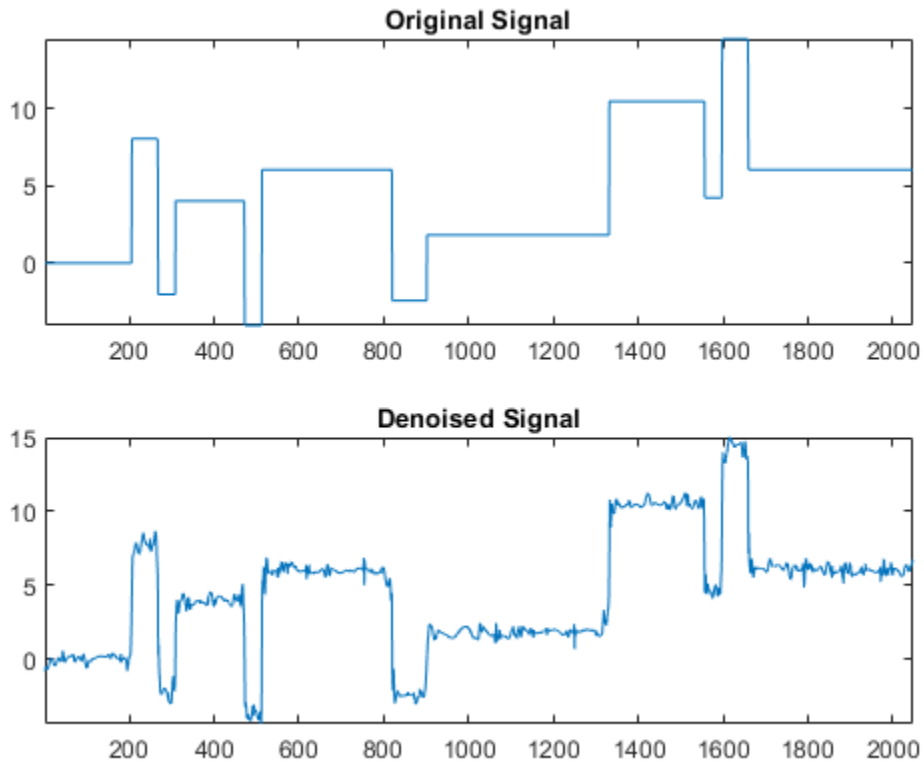
```
[xref,x] = wnoise(1,11,sqrt_snr,init);
subplot(2,1,1)
plot(xref)
axis tight
title('Original Signal')
subplot(2,1,2)
plot(x)
axis tight
title('Noisy Signal')
```



Denoise the noisy signal using soft heuristic SURE thresholding on detail coefficients obtained from the wavelet decomposition of  $x$  using the `sym8` wavelet. Use the default settings of `wdenoise` for the remaining parameters. Compare with the original signal.

```
xd = wdenoise(x, 'Wavelet', 'sym8', 'DenoisingMethod', 'SURE', 'ThresholdRule', 'Soft');
figure
subplot(2,1,1)
plot(xref)
axis tight
title('Original Signal')
subplot(2,1,2)
plot(xd)
axis tight
title('Denoised Signal')
```





Since only a small number of large coefficients characterize the original signal, the method performs very well.

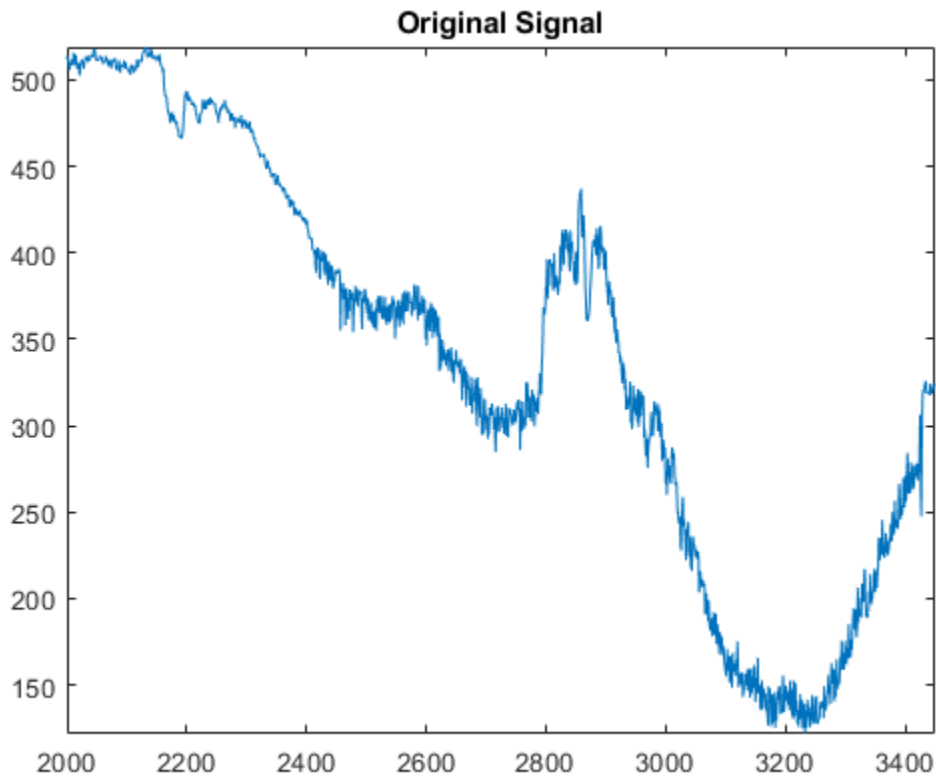
### Electrical Signal Denoising

When you suspect a non-white noise, thresholds must be rescaled by a level-dependent estimation of the level noise. As a second example, let us try the method on the highly perturbed part of an electrical signal.

First load the electrical signal and select a segment from it. Plot the segment.

```
load leleccum
indx = 2000:3450;
x = leleccum(indx);
```

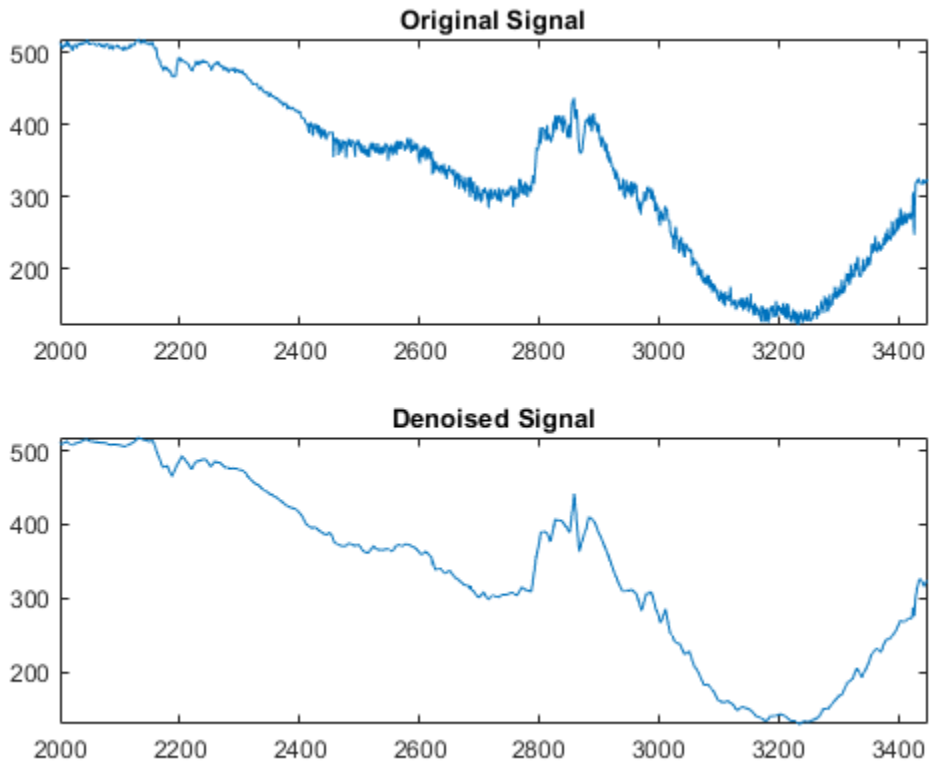
```
figure
plot(indx,x)
axis tight
title('Original Signal')
```



Denoise the signal using the db3 wavelet and a three-level wavelet decomposition and soft fixed form thresholding. To deal with the non-white noise, use level-dependent noise size estimation. Compare with the original signal.

```
xd = wdenoise(x,3,'Wavelet','db3',...
'DenoisingMethod','UniversalThreshold',...
'ThresholdRule','Soft',...
'NoiseEstimate','LevelDependent');
figure
```

```
subplot(2,1,1)
plot(indx,x)
axis tight
title('Original Signal')
subplot(2,1,2)
plot(indx,xd)
axis tight
title('Denoised Signal')
```



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 2410.

## Extension to Image Denoising

The denoising method described for the 1-D case applies also to images and applies well to geometrical images. A direct translation of the 1-D model is

$$s(i, j) = f(i, j) + \sigma e(i, j)$$

where  $e$  is a white Gaussian noise with unit variance.

The 2-D denoising procedure has the same three steps and uses 2-D wavelet tools instead of 1-D tools. 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" 1-D denoising case, 2-D denoising and compression are performed using `wdencomp`. To illustrate 2-D denoising, load an image and create a noisy version of it. For purposes of reproducibility, set the random seed.

```
init = 2055615866;
rng(init);
load woman
img = X;
imgNoisy = img + 15*randn(size(img));
```

Use `ddencomp` to find the denoising values. In this case, fixed form threshold is used with estimation of level noise, thresholding is soft and the approximation coefficients are kept.

```
[thr,sorh,keepapp] = ddencomp('den','wv',imgNoisy);
thr
```

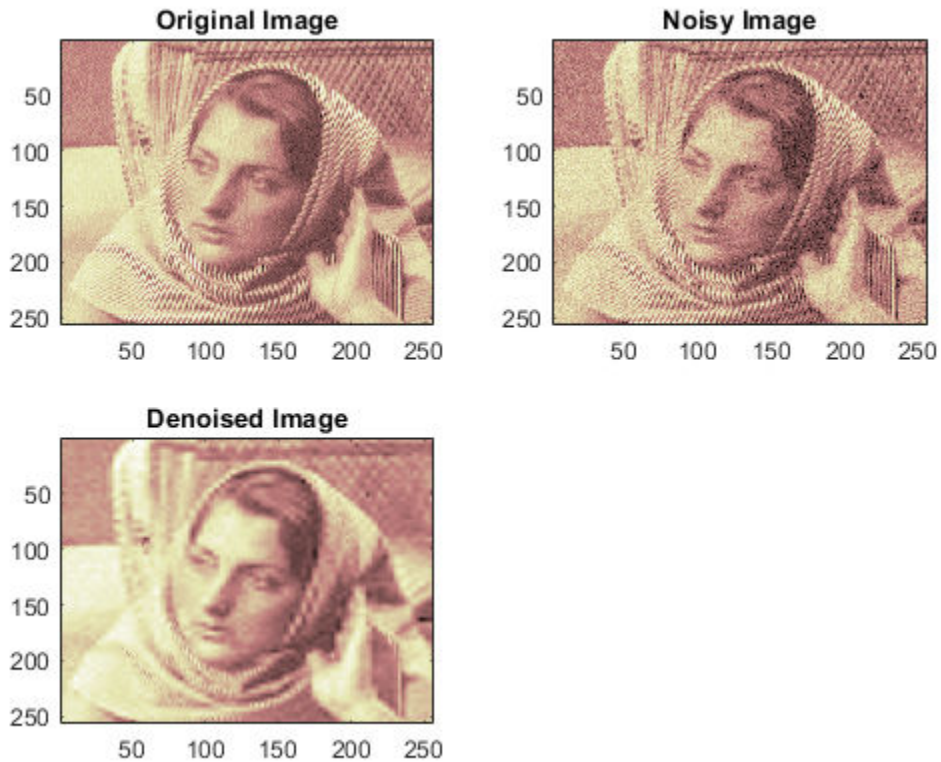
```
thr = 107.9838
```

`thr` is equal to `estimated_sigma*sqrt(log(prod(size(img))))`.

Denoise the noisy image using the global threshold option. Display the results.

```
imgDenoised = wdencomp('gbl',imgNoisy,'sym4',2,thr,sorh,keepapp);
figure
colormap(pink(255))
sm = size(map,1);
subplot(2,2,1)
```

```
image(wcodemat(img,sm))
title('Original Image')
subplot(2,2,2)
image(wcodemat(imgNoisy,sm))
title('Noisy Image')
subplot(2,2,3)
image(wcodemat(imgDenoised,sm))
title('Denoised Image')
```



The denoised image compares well with the original image.

## 1-D Wavelet Variance Adaptive Thresholding

The idea is to define level by level time-dependent thresholds, and then increase the capability of the denoising 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 noise

$$s(n) = f(n) + \sigma e(n)$$

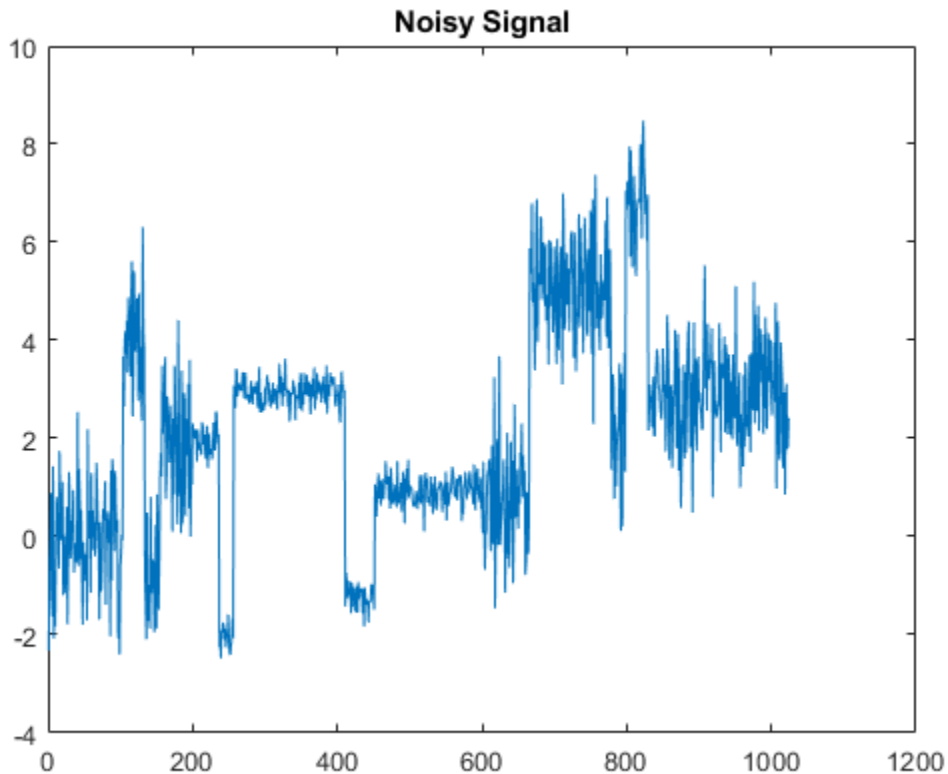
But the noise variance can vary with time. There are several different variance values on several time intervals. The values as well as the intervals are unknown.

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”).

Let us generate a signal from a fixed-design regression model with two noise variance change points located at positions 200 and 600. For purposes of reproducibility, set the random seed.

```
init = 2055615866;
rng(init);

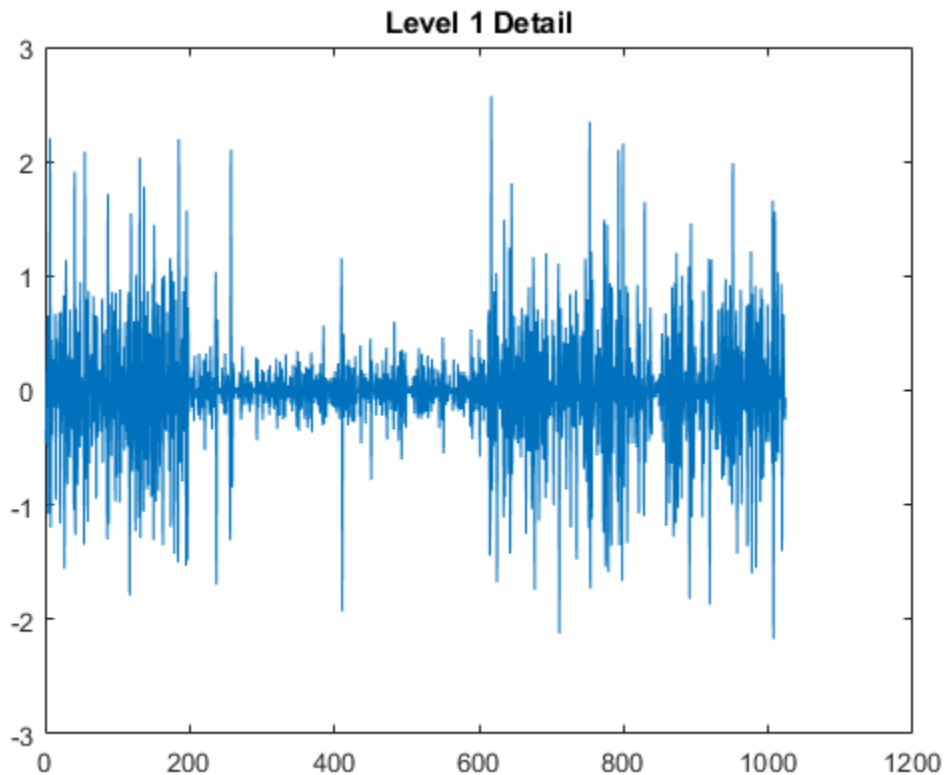
x = wnoise(1,10);
bb = randn(1,length(x));
cp1 = 200;
cp2 = 600;
x = x+[bb(1:cp1),bb(cp1+1:cp2)/4,bb(cp2+1:end)];
plot(x)
title('Noisy Signal')
```



The aim of this example is to recover the two change points from the signal  $x$ .

**Step 1.** Recover a noisy signal by suppressing an approximation. First perform a single-level wavelet decomposition using the `db3` wavelet. Then reconstruct the detail at level 1.

```
wname = 'db3';
lev = 1;
[c,l] = wavedec(x,lev,wname);
det = wrcoef('d',c,l,wname,1);
figure
plot(det)
title('Level 1 Detail')
```



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 = 1×2
```

```
259 611
```

```
kopt = 2
```

```
t_est = 6×6
```

|      |      |      |      |      |      |
|------|------|------|------|------|------|
| 1024 | 0    | 0    | 0    | 0    | 0    |
| 612  | 1024 | 0    | 0    | 0    | 0    |
| 259  | 611  | 1024 | 0    | 0    | 0    |
| 198  | 259  | 611  | 1024 | 0    | 0    |
| 198  | 235  | 259  | 611  | 1024 | 0    |
| 198  | 235  | 260  | 346  | 611  | 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:

```
for k=1:5
 cp_New = t_est(k+1, 1:k)
end
```

```
cp_New = 612
```

```
cp_New = 1×2
```

```
259 611
```

`cp_New = 1×3`

198 259 611

`cp_New = 1×4`

198 235 259 611

`cp_New = 1×5`

198 235 260 346 611

### Wavelet Denoising Analysis Measurements

The following measurements and settings are useful for analyzing wavelet signals and images:

- **M S E** — Mean square error (MSE) is the squared norm of the difference between the data and the signal or image approximation divided by the number of elements.
- **Max Error** — Maximum absolute squared deviation in the signal or image approximation.
- **L2-Norm Ratio** — Ratio of the squared L2-norm of the signal or image approximation to the input signal or image. For images, the image is reshaped as a column vector before taking the L2-norm
- **P S N R** — Peak signal-to-noise ratio (PSNR) in decibels. PSNR is meaningful only for data encoded in terms of bits per sample or bits per pixel.
- **B P P** — Bits per pixel ratio (BPP), which is the compression ratio (Comp. Ratio) multiplied by 8, assuming one byte per pixel (8 bits).
- **Comp Ratio** — Compression ratio, which is the number of elements in the compressed image divided by the number of elements in the original image expressed as a percentage.

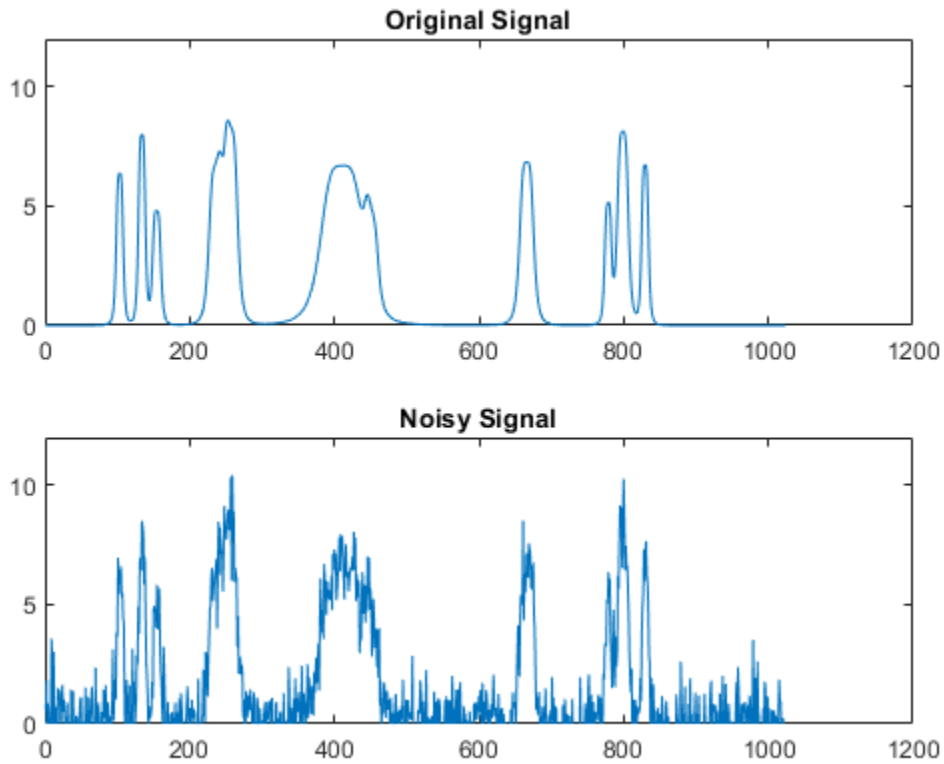
## Wavelet Denoising

This example shows how to use wavelets to denoise signals and images. Because wavelets localize features in your data to different scales, you can preserve important signal or image features while removing noise. The basic idea behind wavelet denoising, or wavelet thresholding, is that the wavelet transform leads to a sparse representation for many real-world signals and images. What this means is that the wavelet transform concentrates signal and image features in a few large-magnitude wavelet coefficients. Wavelet coefficients which are small in value are typically noise and you can "shrink" those coefficients or remove them without affecting the signal or image quality. After you threshold the coefficients, you reconstruct the data using the inverse wavelet transform.

### Denoise a Signal

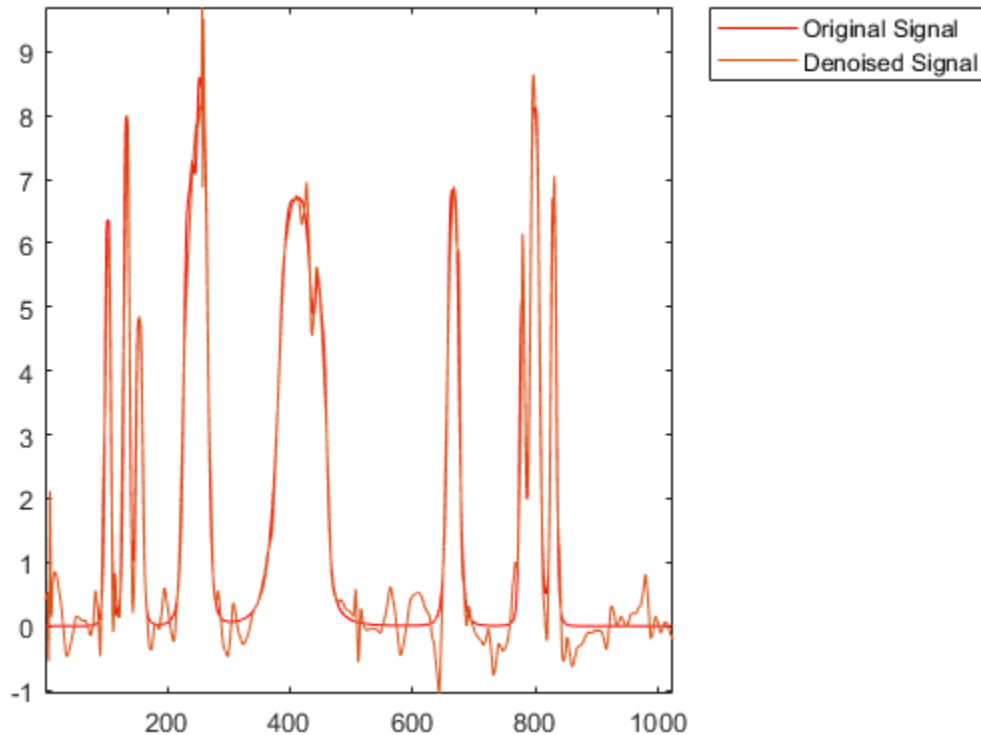
To illustrate wavelet denoising, create a noisy "bumps" signal. In this case you have both the original signal and the noisy version.

```
rng default;
[X,XN] = wnoise('bumps',10,sqrt(6));
subplot(211)
plot(X); title('Original Signal');
AX = gca;
AX.YLim = [0 12];
subplot(212)
plot(XN); title('Noisy Signal');
AX = gca;
AX.YLim = [0 12];
```



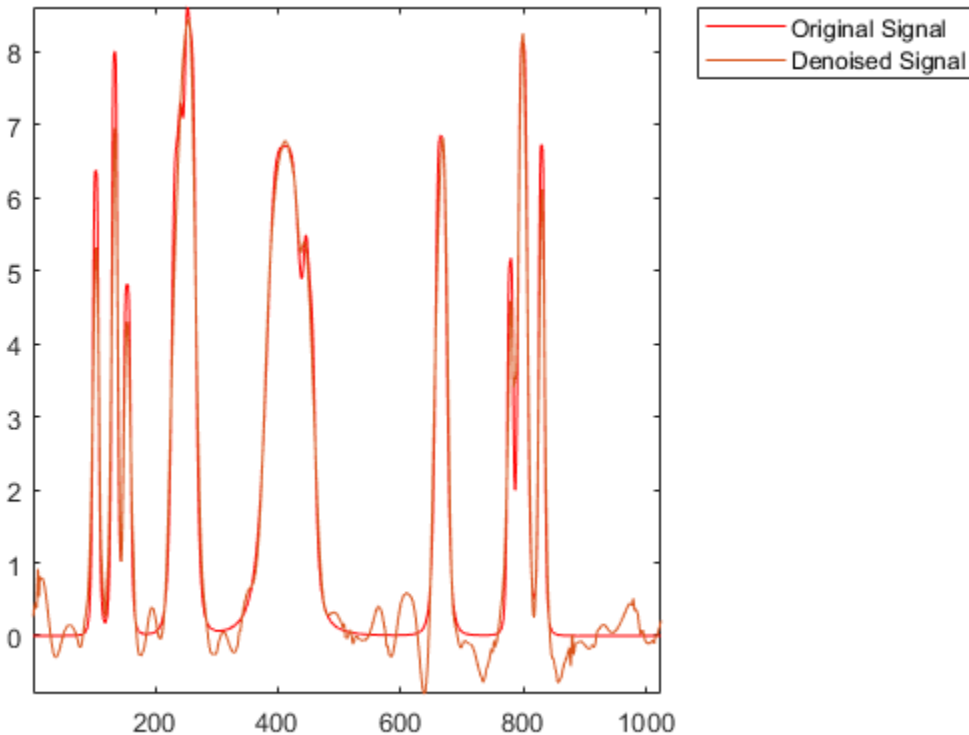
Denoise the signal down to level 4 using `wdenoise` with default settings. `wdenoise` uses the decimated wavelet transform. Plot the result along with the original signal.

```
xd = wdenoise(XN,4);
figure;
plot(X,'r')
hold on;
plot(xd)
legend('Original Signal','Denoised Signal','Location','NorthEastOutside')
axis tight;
hold off;
```



You can also denoise the signal using the undecimated wavelet transform. Denoise the signal again down to level 4 using the undecimated wavelet transform. Plot the result along with the original signal.

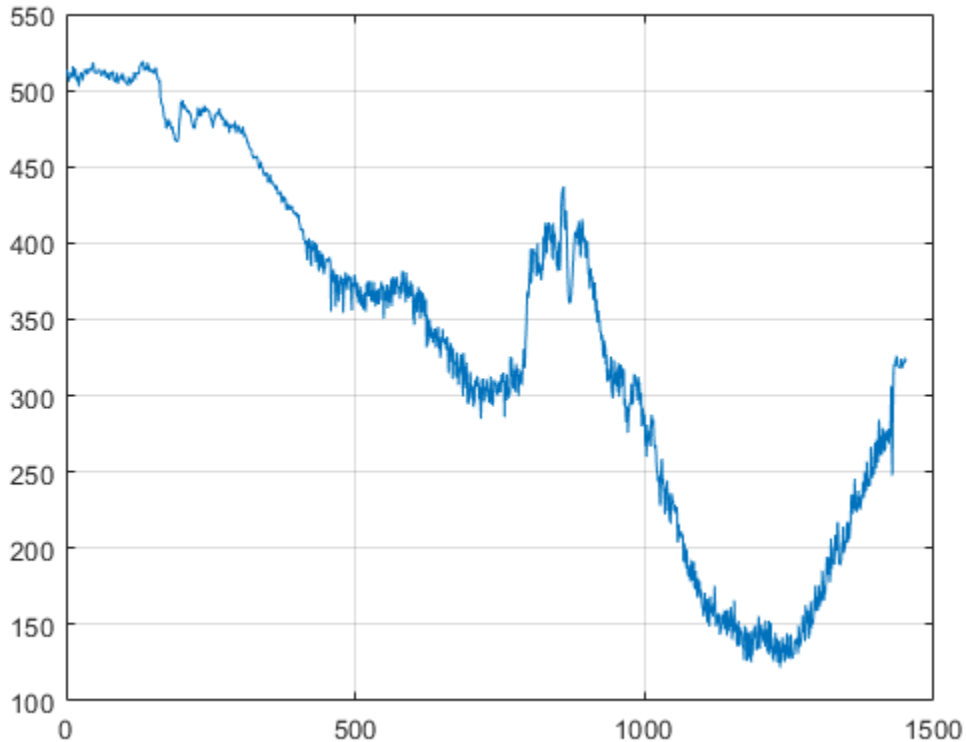
```
xdMODWT = wden(XN, 'modwtsqtwolog', 's', 'mln', 4, 'sym4');
figure;
plot(X, 'r')
hold on;
plot(xdMODWT)
legend('Original Signal', 'Denoised Signal', 'Location', 'NorthEastOutside')
axis tight;
hold off;
```



You see that in both cases, wavelet denoising has removed a considerable amount of the noise while preserving the sharp features in the signal. This is a challenge for Fourier-based denoising. In Fourier-based denoising, or filtering, you apply a lowpass filter to remove the noise. However, when the data has high-frequency features such as spikes in a signal or edges in an image, the lowpass filter smooths these out.

You can also use wavelets to denoise signals in which the noise is nonuniform. Import and examine a portion of a signal showing electricity consumption over time.

```
load leleccum;
indx = 2000:3450;
x = leleccum(indx);
plot(x)
grid on;
```



The signal appears to have more noise after approximately sample 500. Accordingly, you want to use different thresholding in the initial part of the signal. You can use `cmddenoise` to determine the optimal number of intervals to denoise and denoise the signal. In this example, use the 'db3' wavelet and decompose the data down to level 3.

```
[SIGDEN,~,thrParams,~,BestNbOfInt] = cmddenoise(x, 'db3', 3);
```

Display the number of intervals and the sample values that delimit the intervals.

```
BestNbOfInt
```

```
BestNbOfInt = 2
```

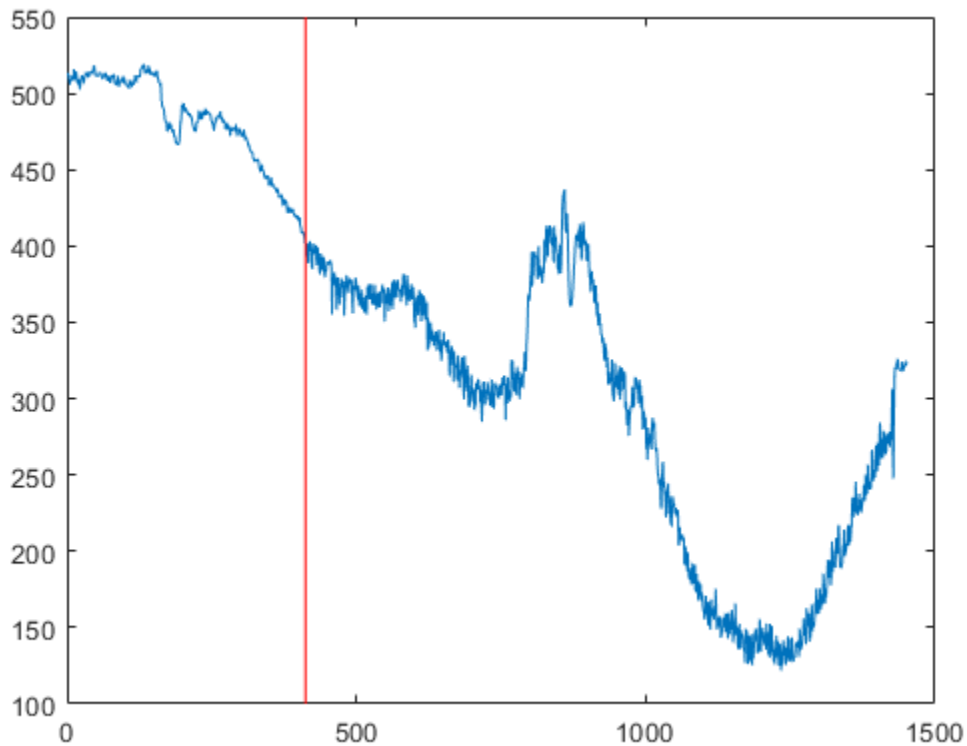
```
thrParams{1}(:,1:2)
```

```
ans = 2x2

 1 412
 412 1451
```

Two intervals were identified. The sample marking the boundary between the two segments is 412. If you plot the signal and mark the two signal segments, you see that the noise does appear different before and after sample 412.

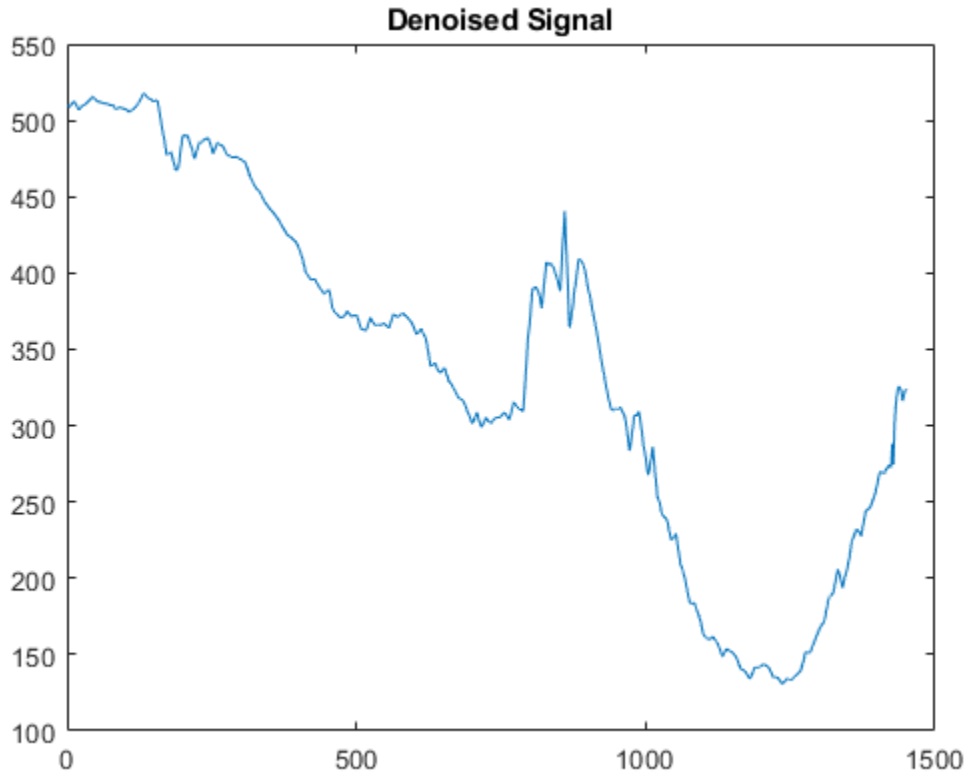
```
plot(x)
hold on;
plot([412 412],[100 550], 'r')
hold off;
```





Plot the denoised signal.

```
plot(SIGDEN)
title('Denoised Signal')
```

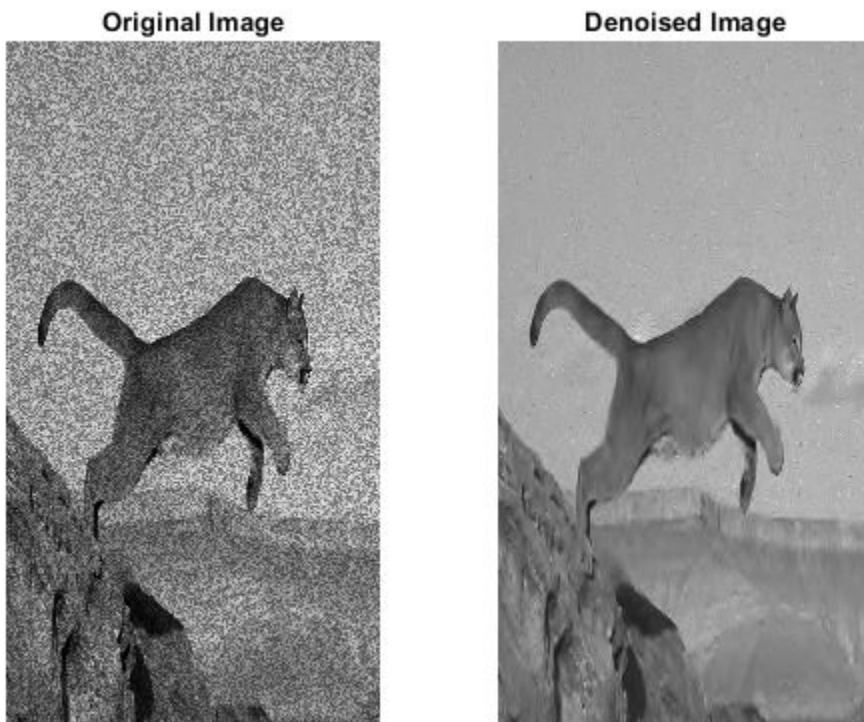


### Denoise an Image

You can also use wavelets to denoise images. In images, edges are places where the image brightness changes rapidly. Maintaining edges while denoising an image is critically important for perceptual quality. While traditional lowpass filtering removes noise, it often smooths edges and adversely affects image quality. Wavelets are able to remove noise while preserving the perceptually important features.

Load a noisy image. Denoise the image using `wdenoise2` with default settings. By default, `wdenoise2` uses the biorthogonal wavelet `bior4.4`. To display the original and denoised images, do not provide any output arguments.

```
load(fullfile(matlabroot, 'examples', 'wavelet', 'jump.mat'))
wdenoise2(jump)
```



Note that edges in the image are not smoothed out by the denoising process.

### See Also

`wdenoise` | `wdenoise2`

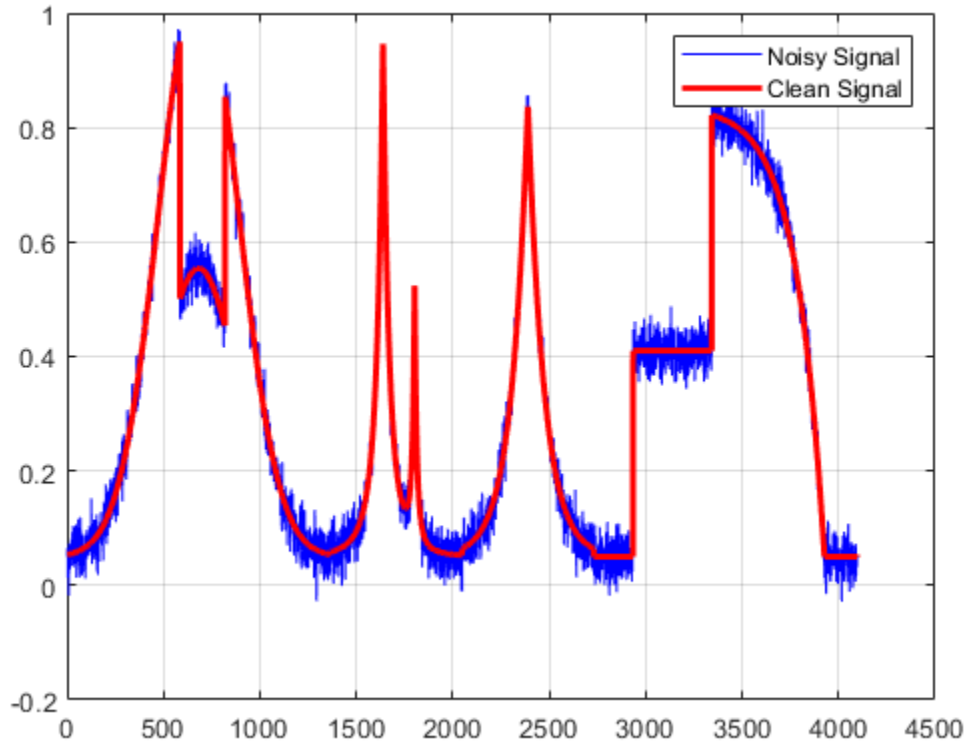
## Denoise a Signal with the Wavelet Signal Denoiser

This example shows how to use the Wavelet Signal Denoiser app to denoise a real-valued 1-D signal. You can create and compare multiple versions of a denoised signal with the app and export the desired denoised signal to your MATLAB® workspace. To reproduce the denoised signal in your workspace, or to apply the same denoising parameters to other data, you can generate and edit a MATLAB script. This example illustrates one possible workflow.

### Import Data into the App

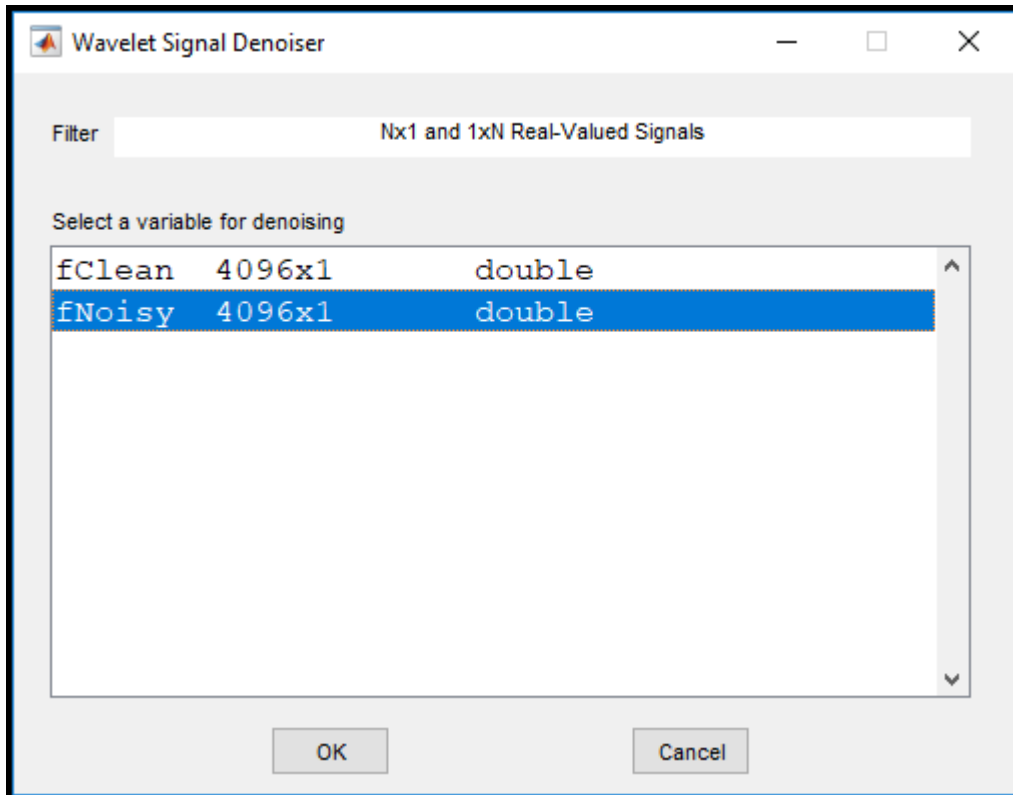
Load data into the MATLAB workspace. The `.mat` file contains a clean version and a noisy version of a signal. Plot both versions of the signal.

```
load fdata
plot(fNoisy,'b-')
hold on
plot(fClean,'r-','LineWidth',2)
legend('Noisy Signal','Clean Signal')
grid on
```

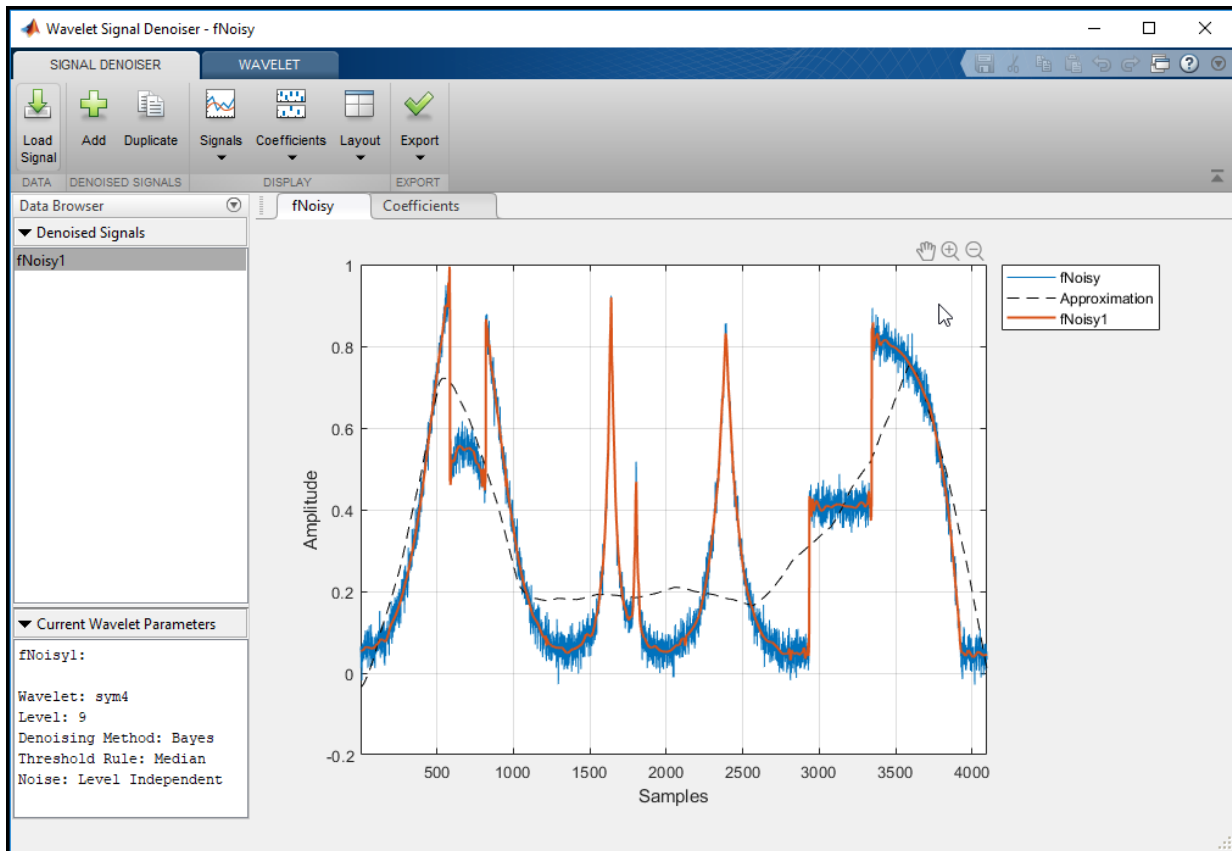


Open the Wavelet Signal Denoiser app. From the MATLAB Toolstrip, open the **Apps** tab and under **Signal Processing and Communications**, click **Wavelet Signal Denoiser**. You can also start the app by typing `waveletSignalDenoiser` at the MATLAB command prompt.

Load the noisy signal from the workspace into the app by clicking **Load Signal** in the toolstrip. From the list of workspace variables that can be loaded into the app, select `fNoisy` and click **OK**.



The app imports the noisy signal and immediately denoises it using default parameters.



### Examine Imported Data

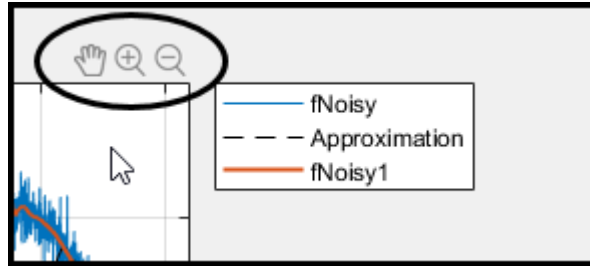
Examine the **fNoisy** plot. The app plots the original signal, **fNoisy**, the denoised signal, **fNoisy1**, and the coarse scale approximation of the signal, **Approximation**.

1. The **Denoised Signals** pane lists all versions of the denoised signal. The list currently contains only the signal that the app created using a default name, **fNoisy1**.

- You can change the default name by right-clicking it, choosing **Rename Denoised Signal** from the menu, entering the new name in the dialog box, and clicking **OK**.
- You can delete a denoised signal by right-clicking its name and choosing **Delete Denoised Signal** from the pop-up menu.

2. The **Current Wavelet Parameters** pane lists the denoising parameters applied to create `fNoisy1`.

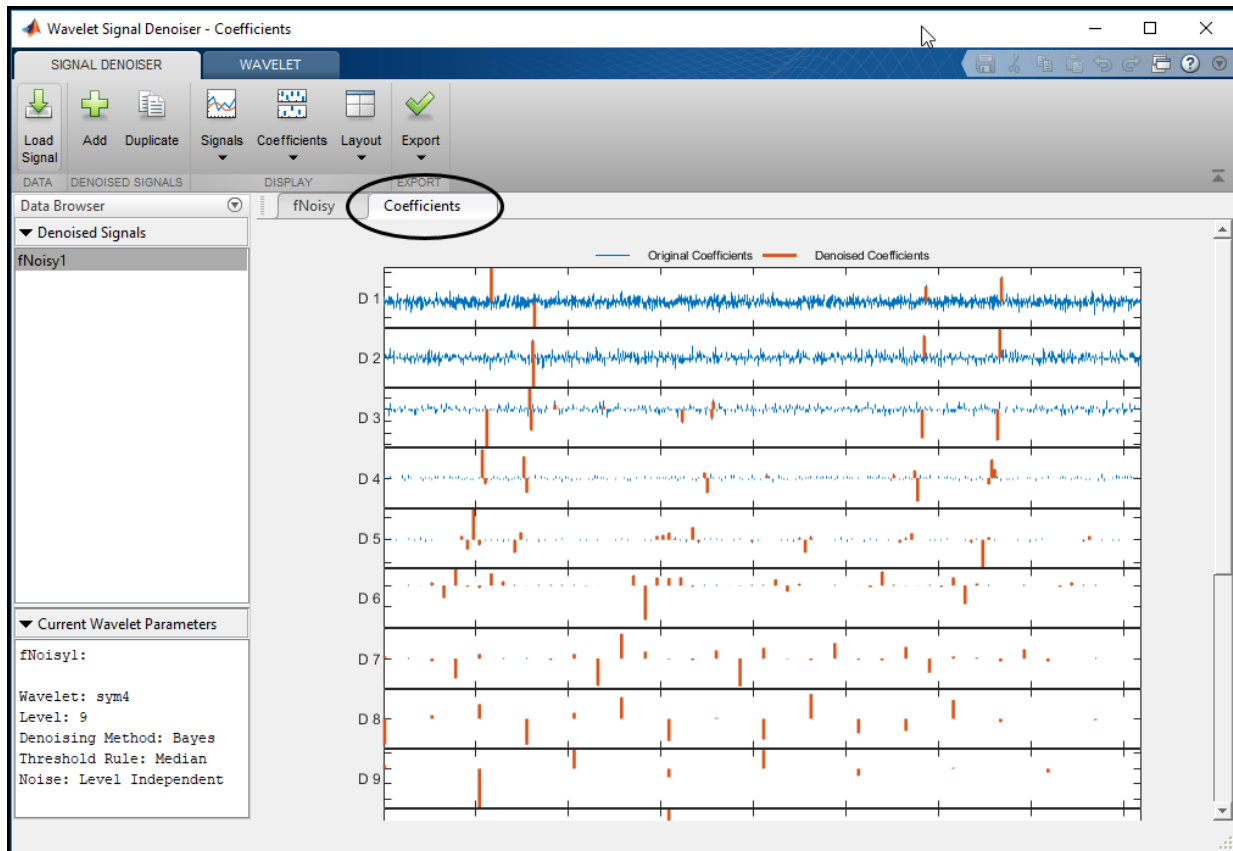
3. Zoom and pan a region of interest. First, place the cursor over the plot to reveal a



floating palette. Then select the desired action from the palette.

Then select the desired

- When you select to either zoom in or out, the mouse wheel controls the zoom.
4. Toggle what signals are visible in the **fNoisy** plot by:
- Clicking **Signals ▼** in the toolstrip and using the drop-down menu to toggle the visibility of the original and denoised signal plots.
  - Clicking individual signals in the plot legend.
5. Examine the **Coefficients** plot.



The coefficients in red are used to reconstruct the denoised signal. The **Current Wavelet Parameters** pane indicates that a 9-level wavelet decomposition was used to denoise the signal.

Zoom and pan a region of interest. First, place the cursor over the plot to reveal a floating palette. Then select the desired action from the palette.

- When you select to either zoom in or out, the mouse wheel controls the zoom and not the scrollbar.
- When you zoom in, zoom out, or pan in the D1 coefficients level, the zoom is applied to all levels.



## Modify Denoising Parameters

Click the **Wavelet** tab. Use this toolstrip to adjust and apply denoising parameters for the selected denoised signal.



The values listed are the parameters used to create the denoised signal, `fNoisy1`. To modify the values of these settings, working from left to right in the toolstrip:

1. In the **Wavelet** dropdown menu, choose the Daubechies wavelet family, `db`. Because of this action:

- The **Number** dropdown field changes to 1. Change the value to 2.
- A banner with blue text appears immediately in the **fNoisy** plot, stating that the wavelet parameter changes are in draft mode. This text appears whenever you have any pending changes to a signal. The banner disappears when you either apply the changes by using the **Denoise** button, or navigate away from the signal to another signal.

2. From the **Method** dropdown menu, select `Universal Threshold`.

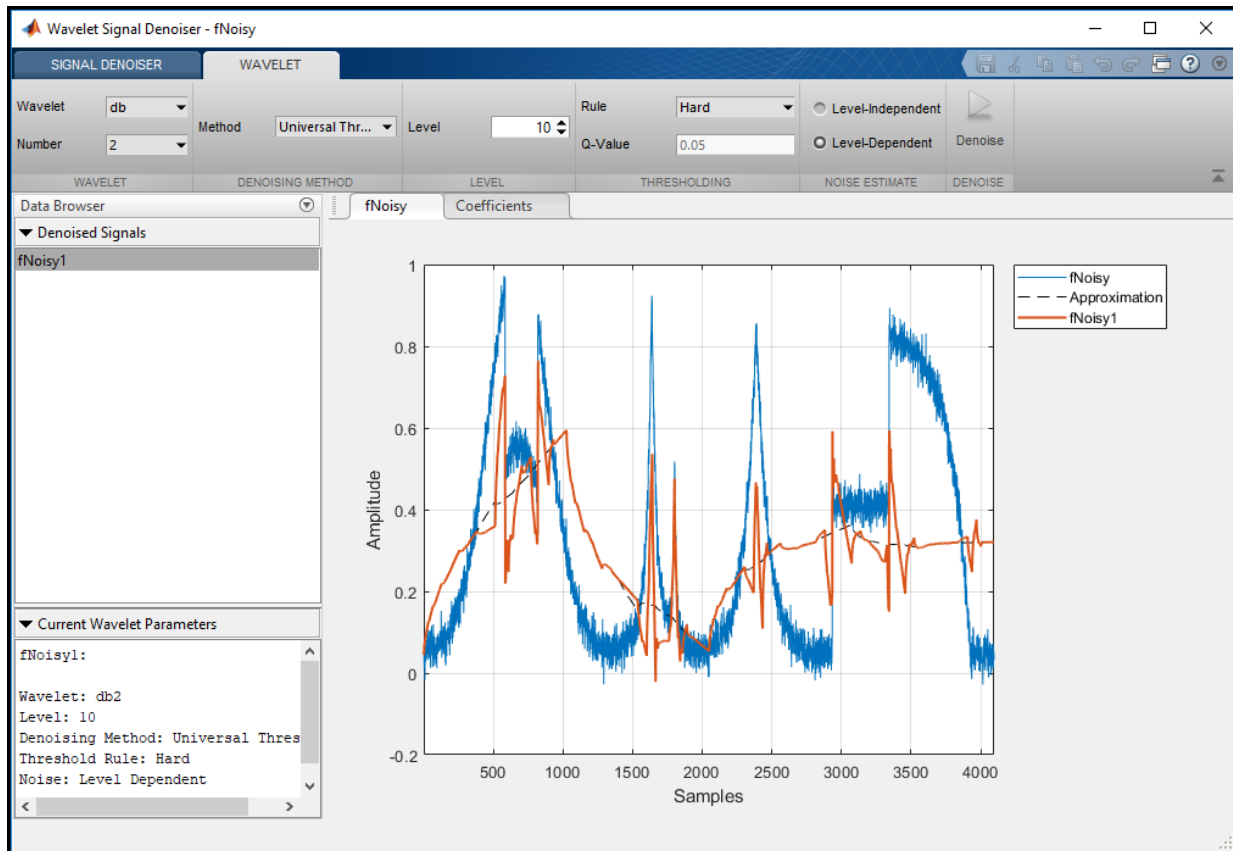
3. To define a 10-level wavelet decomposition, change the value of **Level** to 10.

4. By changing the **Method** to `Universal Threshold`, the **Rule** setting changed automatically from `Median` to `Soft`. Change the setting to `Hard`.

5. Click **Level-Dependent**.

6. Apply the new values for these settings by clicking **Denoise**.

The **Current Wavelet Parameters** pane updates with the new parameters used to denoise the signal, and the app replots the denoised signal, `fNoisy1`.



**Note:** All parameters are contextual. Possible values for one parameter can depend on the currently selected value of another parameter. You cannot make an incompatible selection. For example, when the denoising method is FDR, there is only possible thresholding rule: hard. In this instance, no other values are listed in the **Rule** dropdown menu.

### Duplicate a Denoised Signal and Compare Approximations

If you like a particular denoised signal but want to explore more denoising parameters, you can duplicate it. You can then modify the parameters for the duplicate, without losing the original parameters.

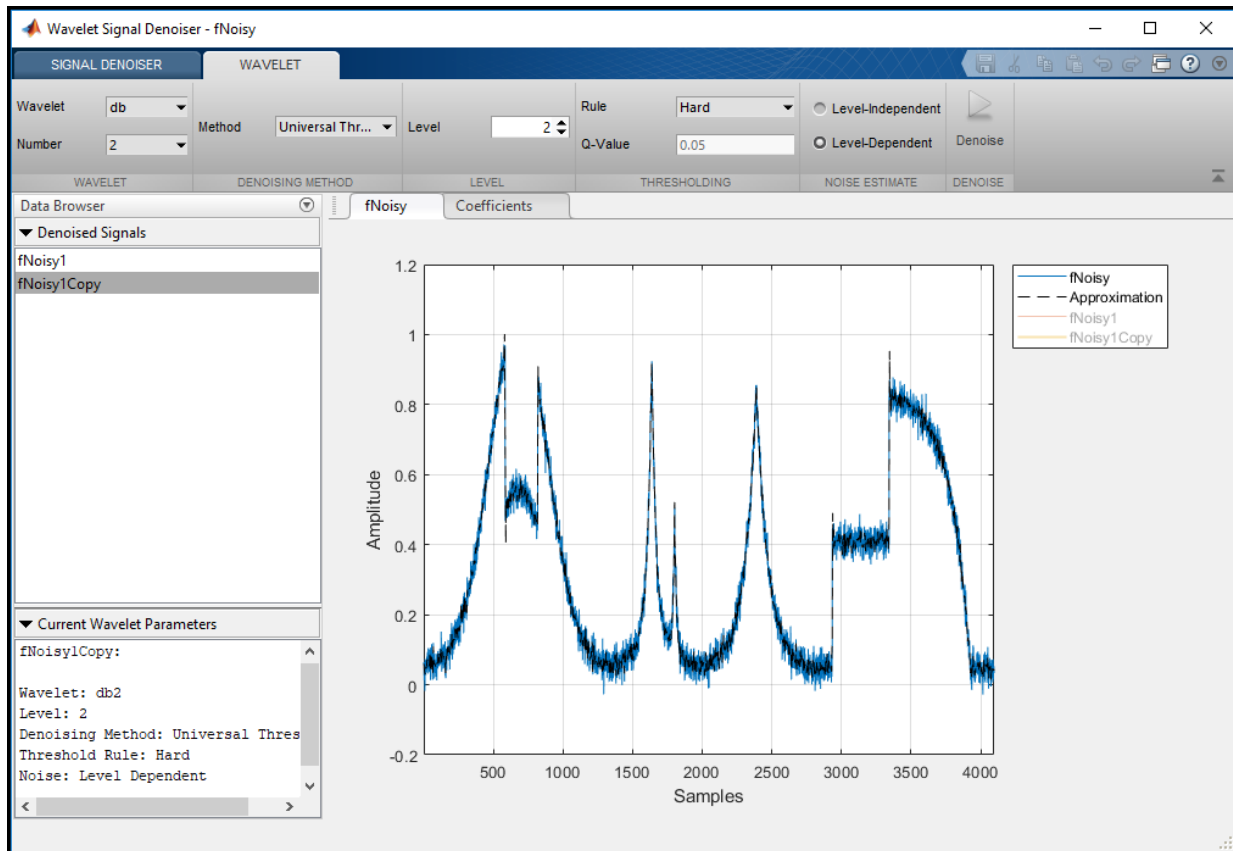
1. From the **Denoised Signals** pane, select `fNoisy1`. Then on the toolstrip, from the **Signal Denoiser** tab, click **Duplicate**.

- The duplicate signal, `fNoisy1Copy`, appears highlighted in **Denoised Signals**.
- The denoising parameters for the duplicate are listed in **Current Wavelet Parameters**.
- The duplicate is plotted as a thick line in the **fNoisy** plot. The plot legend updates to include the duplicate.

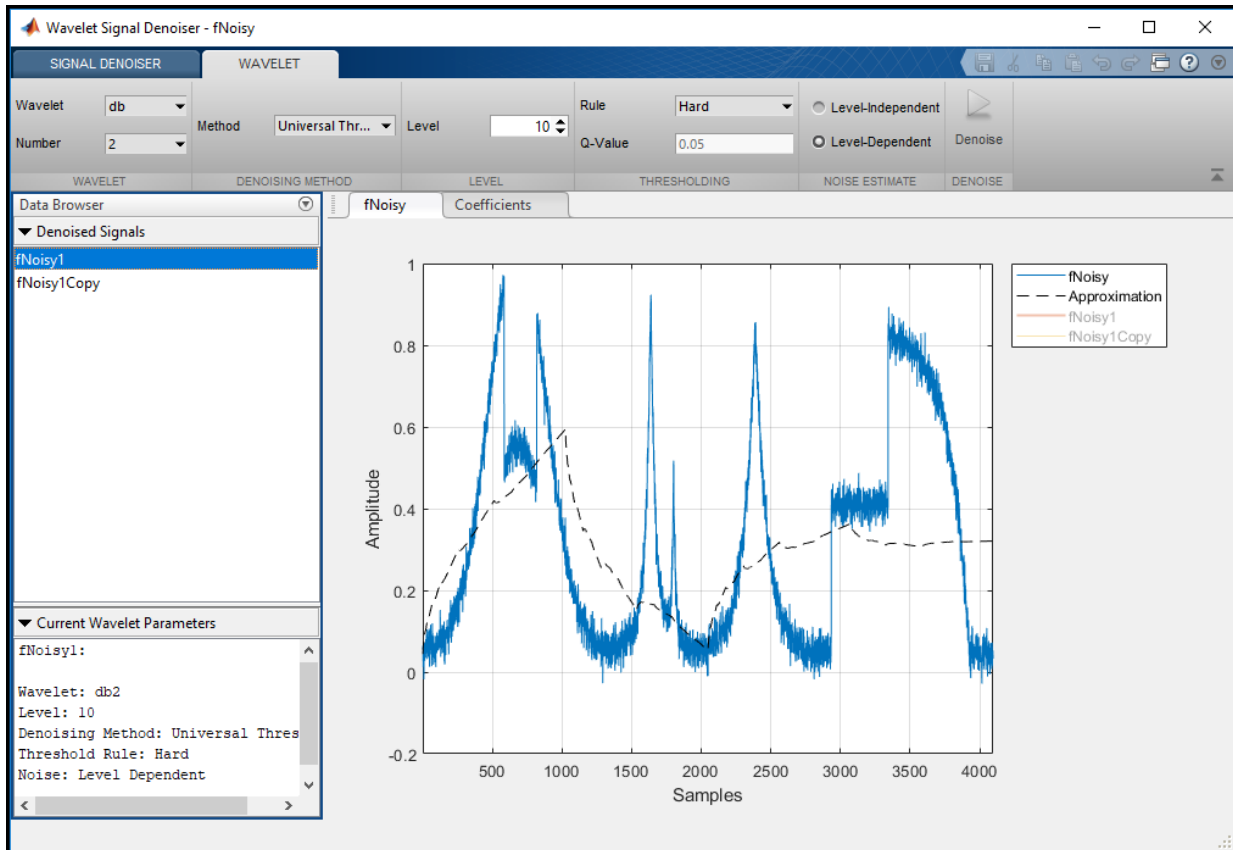
2. On the **Wavelet** tab, change the **Level** to 2, and then click **Denoise**. The app denoises the signal using a two-level wavelet decomposition. In addition, the app:

- Recalculates and plots the approximation for the duplicate.
- Updates the **Coefficients** plot to show the levels for the duplicate.

Because `fNoisy1Copy` is highlighted, its approximation is plotted. The app always plots the approximation for the currently selected denoised signal. You can demonstrate this behavior as follows. In the plot legend, click `fNoisy1` and `fNoisy1Copy`. The names of both denoised signals fade, and the two signals are no longer plotted. Only the original signal and approximation plots are visible.



The dashed line in the plot represents the approximation. Because `fNoisy1Copy` is highlighted in the **Denoised Signals** list, the approximation plotted is the result of a two-level wavelet decomposition. The approximation is relatively noisy. Now select `fNoisy1` in the list. The approximation of a 10-level wavelet decomposition is different.



## Restore Original Parameters

You can always return to using the original default parameters by adding a new denoised signal. From the toolbar, on the **Signal Denoiser** tab, click **Add**.

- The added denoised signal, `fNoisy2`, appears highlighted in the **Denoised Signals** list. The default denoising parameters are listed in **Current Wavelet Parameters**.
- The new denoised signal is plotted as a thick line. The approximation is calculated and plotted as well. The plot legend updates to include `fNoisy2`.

## Export Results

If you want to apply the same denoising parameters to other data, you can use the app to generate a script that reproduces the selected denoised signal. You can then modify and save the script for your own purposes. To do further analysis, you can export a denoised signal to your workspace.

## Export Script

Click `fNoisy2` in **Denoised Signals**. On the **Signal Denoiser** tab of the toolstrip, from the **Export ▼** menu, select **Generate MATLAB Script**. An untitled script opens in your MATLAB Editor with the following executable code:

```
fNoisy2 = wdenoise(fNoisy,9, ...
 'Wavelet', 'sym4', ...
 'DenoisingMethod', 'Bayes', ...
 'ThresholdRule', 'Median', ...
 'NoiseEstimate', 'LevelIndependent');
```

The `wdenoise` input arguments are populated with the values used to create `fNoisy2`. Save the script and then run. This will create the variable `fNoisy2` in your workspace.

Load the file `fdataTS`. The file contains noisy data of 100 time series. Each time series has 4096 data points. The data is contained in a type `TimeTable` variable called `fdataTS`.

```
load fdataTS
```

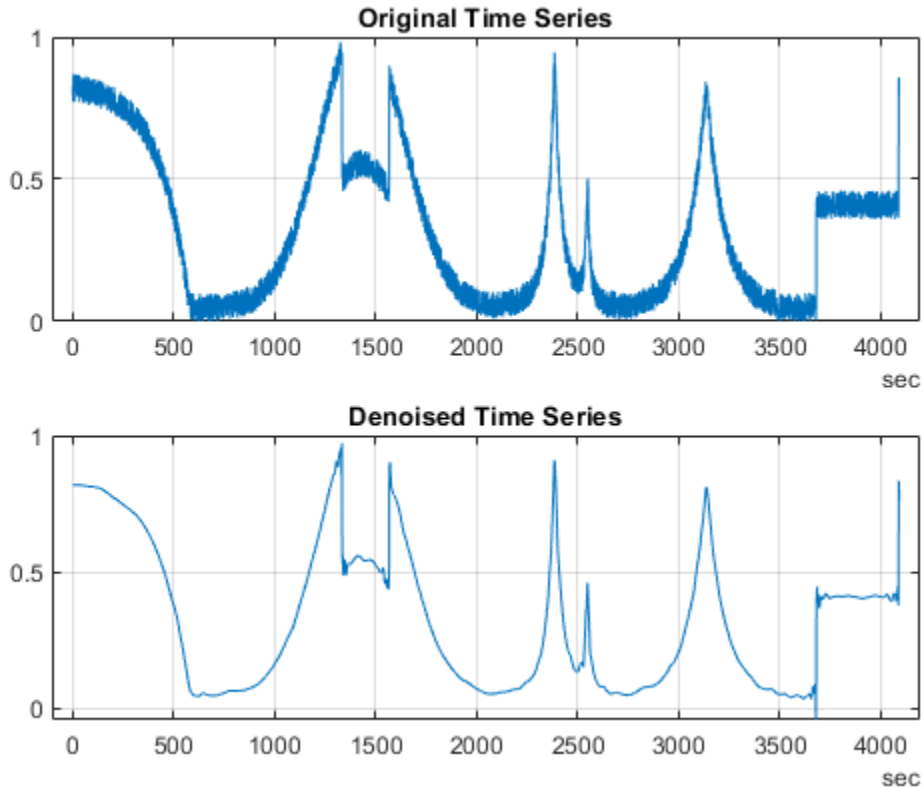
To apply the denoising parameters to `fdataTS`, edit the script by replacing `fNoisy` with `fdataTS` and `fNoisy2` with `fdataTSclean`. Then run the script.

```
fdataTSclean = wdenoise(fdataTS,9, ...
 'Wavelet', 'sym4', ...
 'DenoisingMethod', 'Bayes', ...
 'ThresholdRule', 'Median', ...
 'NoiseEstimate', 'LevelIndependent');
```

Compare the 15th noisy time series with its denoised version.

```
subplot(2,1,1)
plot(fdataTS.Time, fdataTS.fTS15)
title('Original Time Series')
grid on
subplot(2,1,2)
```

```
plot(fdataTSclean.Time, fdataTSclean.fTS15)
title('Denoised Time Series')
grid on
```



### Export Denoised Signal

Click `fNoisy2` in **Denoised Signals**. Then on the toolstrip, from the **Signal Denoiser** tab, click the green check mark on **Export ▼**. Since `fNoisy2` already exists in your workspace, you can force the export and overwrite the workspace variable. Alternatively, you can cancel the export, rename either the denoised signal in the app or the workspace variable, and export again. A banner appears confirming the signal is exported. To permanently remove the banner, either click X to close or import a new signal into the app.

Because you have the clean signal in your workspace, calculate the sign-to-noise ratio of the denoised signal.

```
snrWavelet = -20*log10(norm(abs(fClean-fNoisy2))/norm(fClean))
```

```
snrWavelet = 35.9623
```

If you have the Signal Processing Toolbox™ denoise the signal using a moving average filter and Savitzky-Golay filter and compute the SNR of each denoised signal.

```
fmv = smoothdata(fNoisy, 'movmean', 25);
```

```
snrMovingAverage = -20*log10(norm(abs(fClean-fmv))/norm(fClean))
```

```
snrMovingAverage = 26.0040
```

```
fsg = smoothdata(fNoisy, 'sgolay', 25);
```

```
snrSavitskyGolay = -20*log10(norm(abs(fClean-fsg))/norm(fClean))
```

```
snrSavitskyGolay = 28.8932
```

You achieve superior results denoising with the sym4 wavelet.



## Translation Invariant Wavelet Denoising with Cycle Spinning

Cycle spinning compensates for the lack of shift invariance in the critically-sampled wavelet transform by averaging over denoised cyclically-shifted versions of the signal or image. The appropriate inverse circulant shift operator is applied to the denoised signal/image and the results are averaged together to obtain the final denoised signal/image.

There are  $N$  unique cyclically-shifted versions of a signal of length,  $N$ . For an  $M$ -by- $N$  image, there are  $MN$  versions. This makes using all possible shifted versions computationally prohibitive. However, in practice, good results can be obtained by using a small subset of the possible circular shifts.

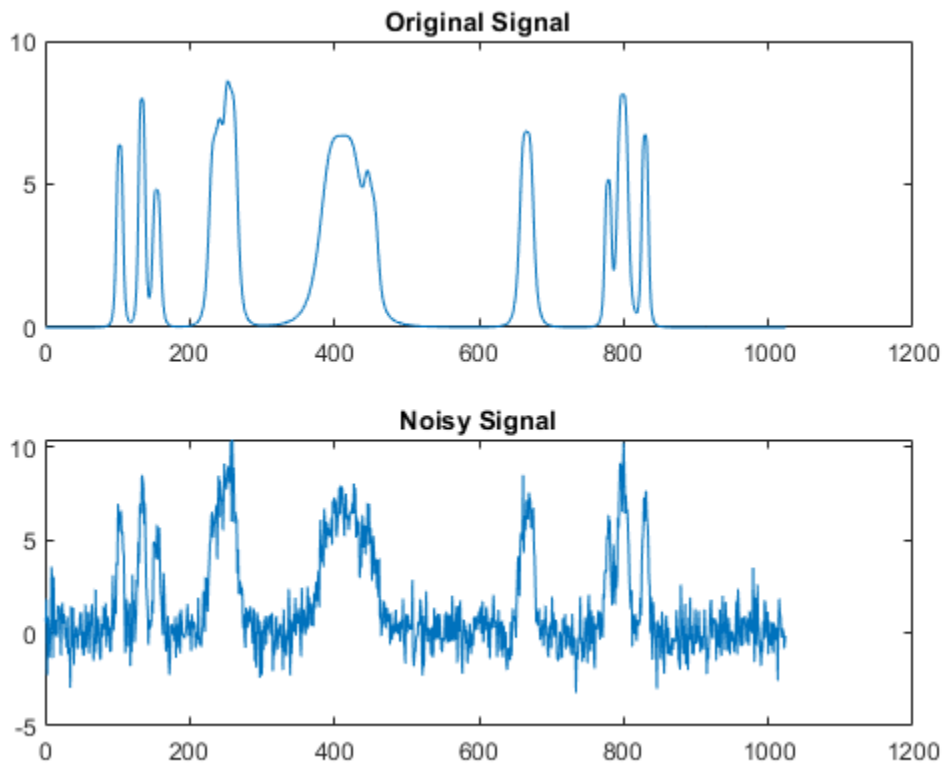
The following example shows how you use `wdenoise` and `circshift` to denoise a 1-D signal using cycle spinning. For denoising grayscale and RGB images, `wdenoise2` supports cycle spinning.

### 1-D Cycle Spinning

This example shows how to denoise a 1-D signal using cycle spinning and the shift-variant orthogonal nonredundant wavelet transform. The example compares the results of the two denoising methods.

Create a noisy 1-D *bumps* signal with a signal-to-noise ratio of 6. The signal-to-noise ratio is defined as  $\frac{N \|X\|_2^2}{\sigma^2}$  where  $N$  is the length of the signal,  $\|X\|_2^2$  is the squared L2 norm, and  $\sigma^2$  is the variance of the noise.

```
rng default
[X,XN] = wnoise('bumps',10,sqrt(6));
subplot(2,1,1)
plot(X)
title('Original Signal')
subplot(2,1,2)
plot(XN)
title('Noisy Signal')
```



Denoise the signal using cycle spinning with 15 shifts, 7 to the left and 7 to the right, including the zero-shifted signal. Use `wdenoise` with default settings. By default, `wdenoise` uses Daubechies' least-asymmetric wavelet with four vanishing moments, `sym4`. Denoising is down to the minimum of `floor(log2(N))` and `wmaxlev(N, 'sym4')` where `N` is the number of samples in the data.

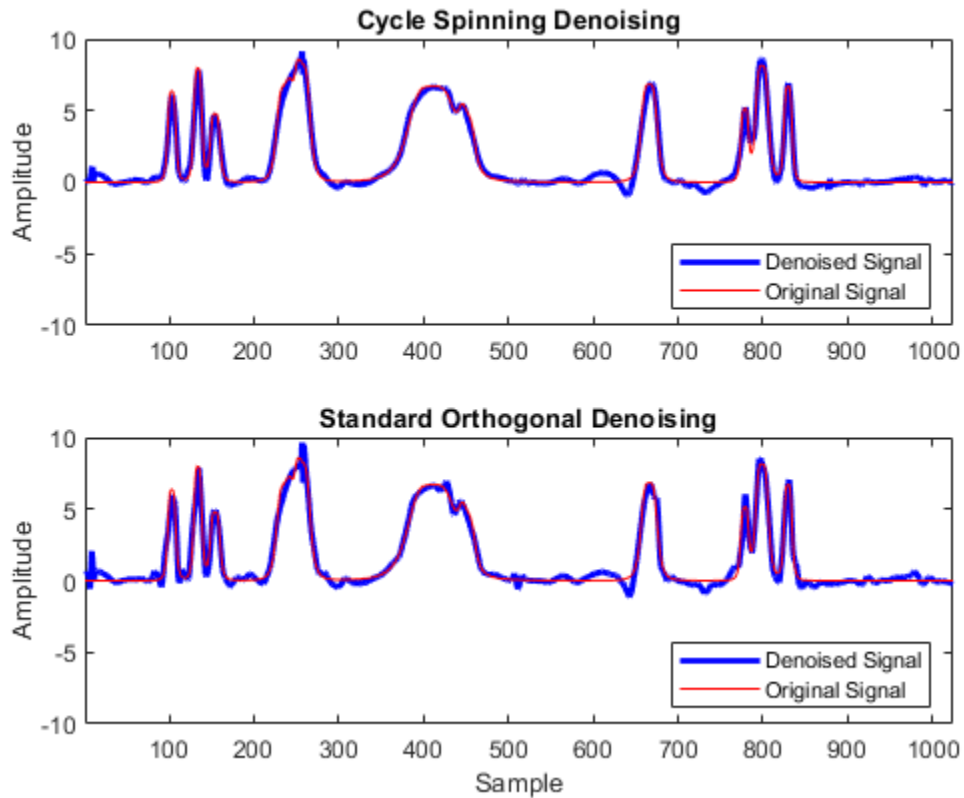
```

ydenoise = zeros(length(XN),15);
for nn = -7:7
 yshift = circshift(XN,[0 nn]);
 [yd,cyd] = wdenoise(yshift);
 ydenoise(:,nn+8) = circshift(yd,[0, -nn]);
end
ydenoise = mean(ydenoise,2);

```

Denoise the signal using `wdenoise`. Compare with the cycle spinning results.

```
xd = wdenoise(XN);
subplot(2,1,1)
plot(ydenoise,'b','linewidth',2)
hold on
plot(X,'r')
axis([1 1024 -10 10])
legend('Denoised Signal','Original Signal','Location','SouthEast')
ylabel('Amplitude')
title('Cycle Spinning Denoising')
hold off
subplot(2,1,2)
plot(xd,'b','linewidth',2)
hold on
plot(X,'r')
axis([1 1024 -10 10])
legend('Denoised Signal','Original Signal','Location','SouthEast')
xlabel('Sample')
ylabel('Amplitude')
title('Standard Orthogonal Denoising')
hold off
```



```
absDiffDWT = norm(X-xd,2)
```

```
absDiffDWT = 12.4248
```

```
absDiffCycleSpin = norm(X-ydenoise',2)
```

```
absDiffCycleSpin = 10.6124
```

Cycle spinning with only 15 shifts has reduced the approximation error.

## See Also

### Functions

wdenoise | wdenoise2

### Apps

Wavelet Signal Denoiser

## 1-D Adaptive Thresholding of Wavelet Coefficients

This section takes you through the features of local thresholding of wavelet coefficients for 1-D signals or data. This capability is available through Wavelet Analyzer app:

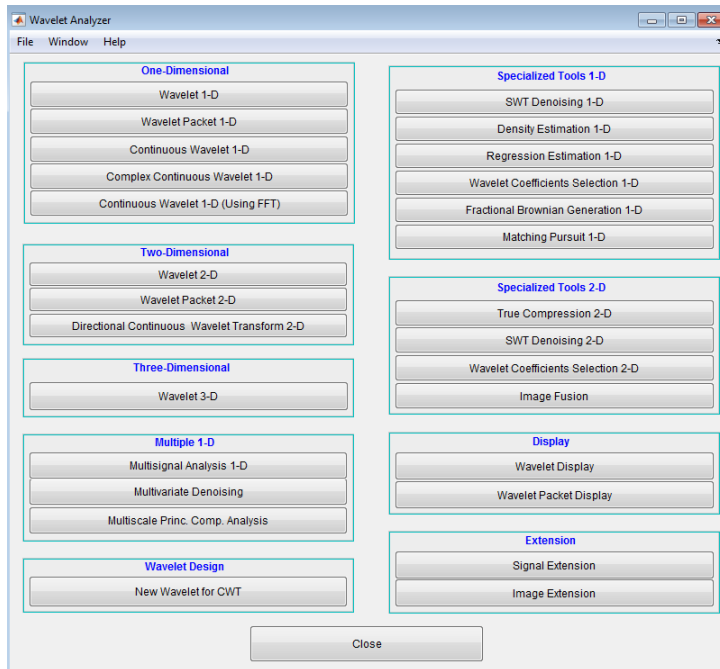
- Wavelet Denoising 1-D
- Wavelet Compression 1-D
- SWT Denoising 1-D
- Regression Estimation 1-D
- Density Estimation 1-D

This tool allows you to define, level by level, time-dependent (x-axis-dependent) thresholds, and then increase the capability of the denoising 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 Wavelet Analyzer app tools (**SWT Denoising 1-D**) to illustrate this capability. The behavior of all the above-mentioned tools is similar.

### 1-D Local Thresholding Using the Wavelet Analyzer App

**1** From the MATLAB prompt, type `waveletAnalyzer`.

The **Wavelet Analyzer** appears.



Click the **SWT Denoising 1-D** menu item.

The discrete stationary wavelet transform denoising tool for 1-D signals appears.

**2** Load data.

At the MATLAB command prompt, type

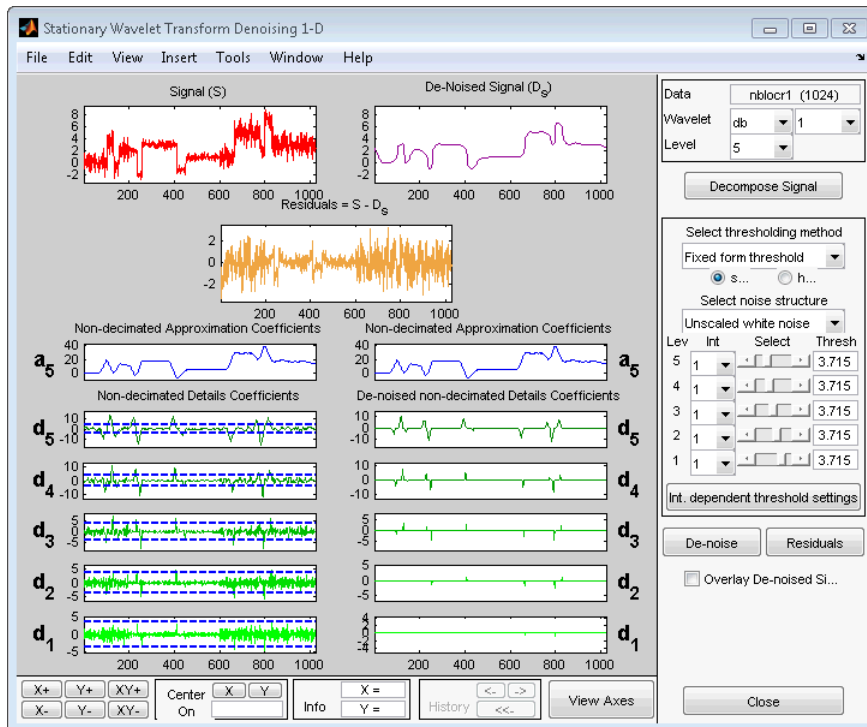
```
load nblocr1;
```

In the **SWT Denoising 1-D** tool, select **File > Import from Workspace**. When the **Import from Workspace** dialog box appears, select the `nblocr1` variable. Click **OK** to import the noisy blocks signal with two change points in the noise variance located at positions 200 and 600.

**3** Perform signal decomposition.

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

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



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

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

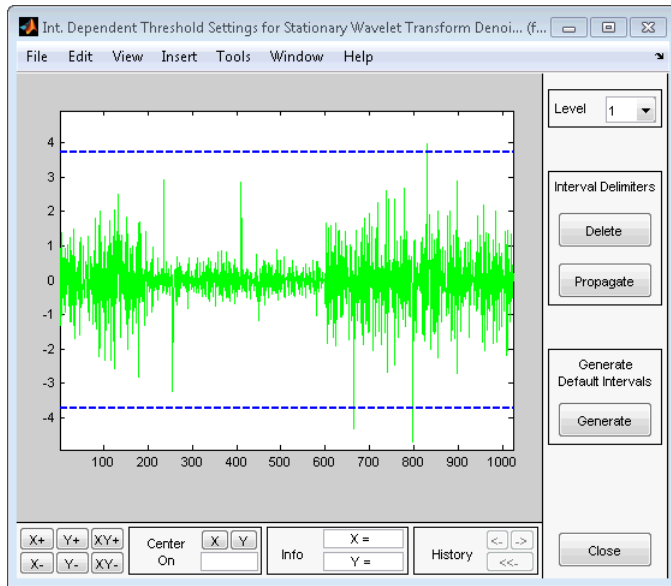




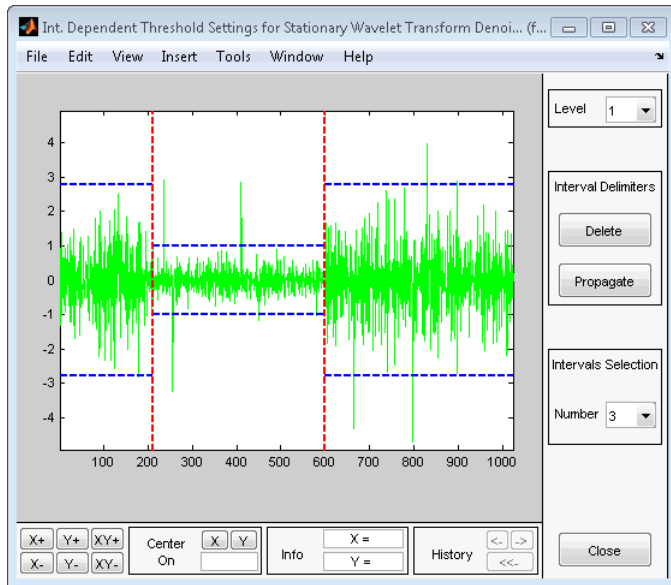
The result is not satisfactory. The denoised signal remains noisy before position 200 and after position 700. This illustrates the limits of the classical denoising strategies. In addition, the residuals obtained during the last trials clearly suggest to try a local thresholding strategy.

#### 4 Generate interval-dependent thresholds.

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



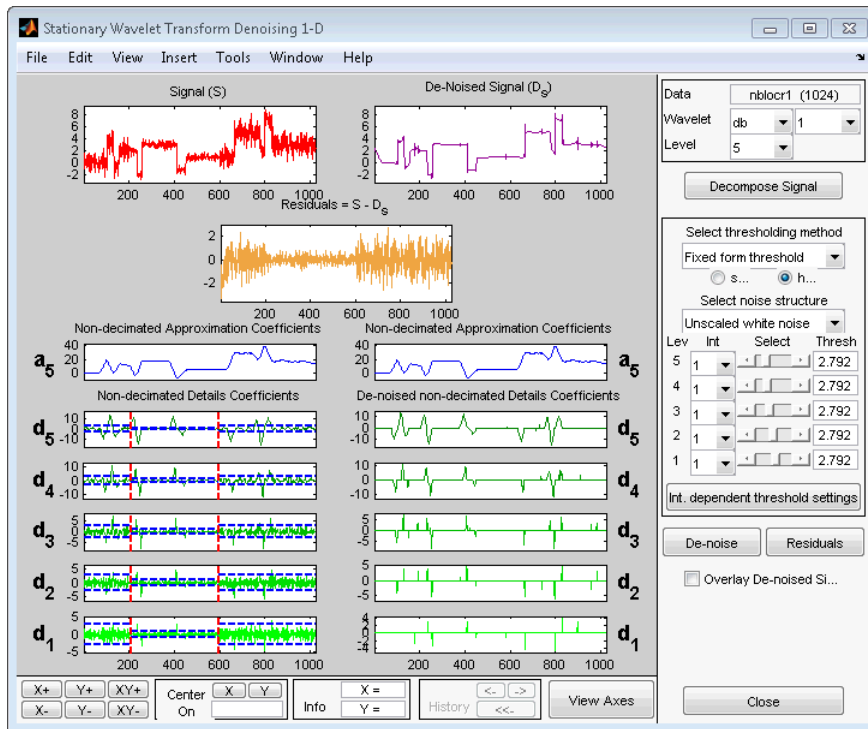
Click the **Generate** button. After a pause for computation, the tool displays the default intervals associated with adapted thresholds.



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

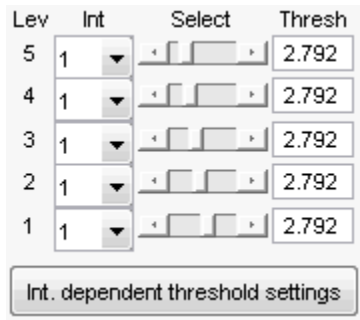
### Noise with Interval-Dependent Thresholds

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



### Modifying Interval Dependent Thresholds

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



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

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

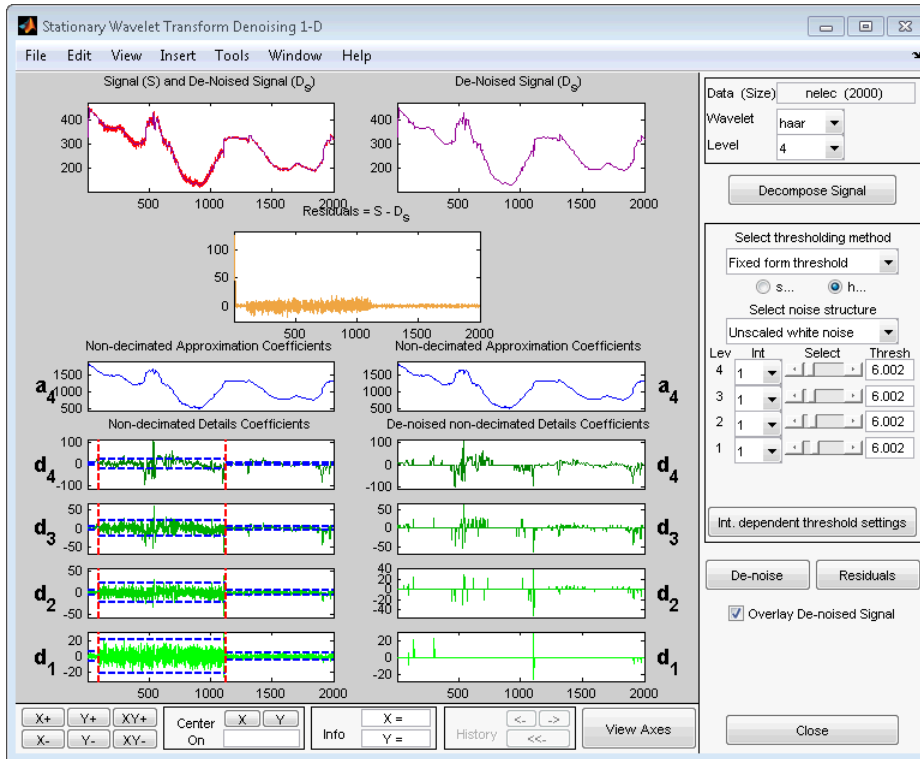
Note that

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

### Examples of Denoising with Interval Dependent Thresholds.

From the **File** menu, choose the **Example Analysis > Noisy Signals - Interval Dependent Noise Variance >** option. From the drop down men, choose with **haar** at level 4 ---> **Elec. consumption – 3 intervals**. 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.



## Importing and Exporting Information from the Wavelet Analyzer App

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

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

```
load dnelec
whos
```

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

The denoised signal is given by `dnelec`. In addition, the parameters of the denoising 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
```

## Multivariate Wavelet Denoising

This section demonstrates the features of multivariate denoising provided in the Wavelet Toolbox software. The toolbox includes the `wmldden` function and a **Wavelet Analyzer** app. This section also describes the command-line and app methods and includes information about transferring signal and parameter information between the disk and the app.

This multivariate wavelet denoising 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 Denoising – Command Line

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

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

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

```
load ex4mwden
whos
```

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

Usually, only the matrix of data  $x$  is available. Here, we also have the true noise covariance matrix (`covar`) and the original signals (`x_orig`). These signals are noisy



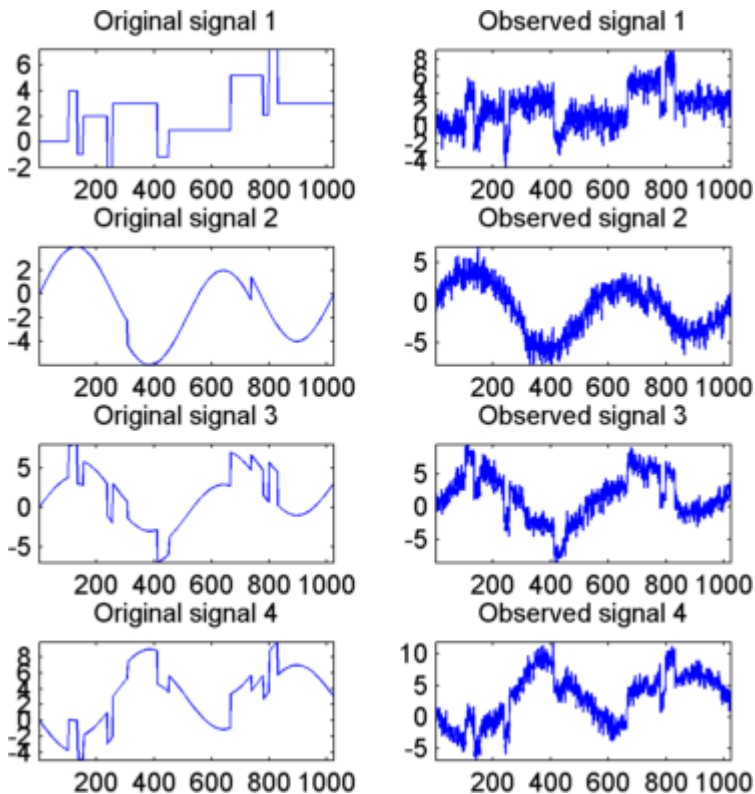
versions of simple combinations of the two original signals. The first one is “Blocks” which is irregular, and the second is “HeavySine,” which is regular except around time 750. The other two signals are the sum and the difference of the two original signals. Multivariate Gaussian white noise exhibiting strong spatial correlation is added to the resulting four signals, which leads to the observed data stored in  $x$ .

- 2 Display the original and observed signals by typing

```

kp = 0;
for i = 1:4
 subplot(4,2,kp+1), plot(x_orig(:,i)); axis tight;
 title(['Original signal ',num2str(i)])
 subplot(4,2,kp+2), plot(x(:,i)); axis tight;
 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 denoising strategy combines univariate wavelet denoising 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 denoising by typing the following to set the denoising 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 denoising by typing

```
x_den = wmulden(x, level, wname, npc_app, npc_fin, tptr, sorh);
```

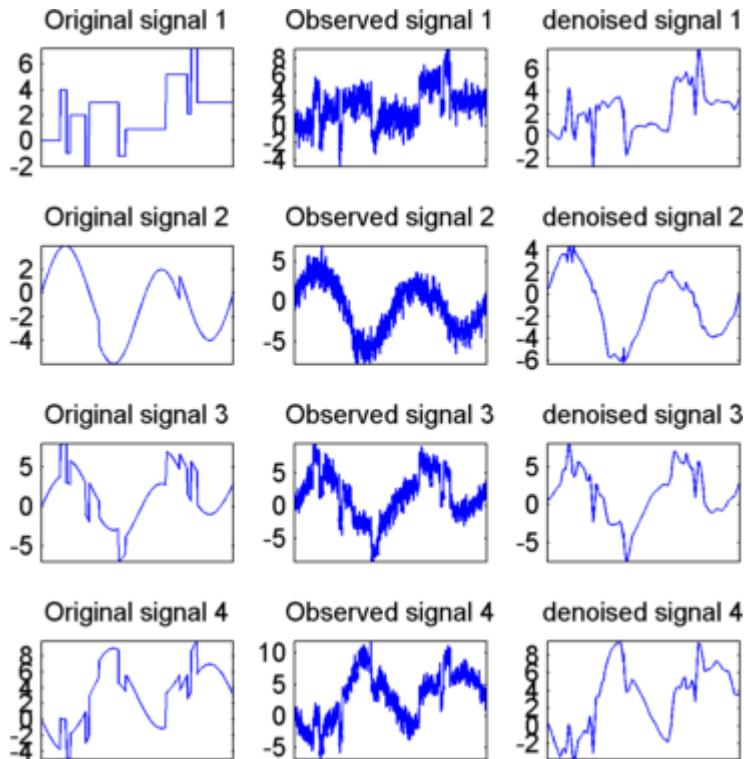
- 4 Display the original and denoised signals by typing

```
kp = 0;
for i = 1:4
 subplot(4,3,kp+1), plot(x_orig(:,i));
 set(gca,'xtick',[]); axis tight;
 title(['Original signal ',num2str(i)])
 subplot(4,3,kp+2), plot(x(:,i)); set(gca,'xtick',[]);
 axis tight;
 title(['Observed signal ',num2str(i)])
 subplot(4,3,kp+3), plot(x_den(:,i)); set(gca,'xtick',[]);
 axis tight;
 title(['denoised signal ',num2str(i)])
```

```

 kp = kp + 3;
end

```



- 5 Improve the first result by retaining fewer principal components.

The results are satisfactory. Focusing on the two first signals, note that they are correctly recovered, but the result can be improved by taking advantage of the relationships between the signals, leading to an additional denoising 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 denoising again by typing

```
[x_den, npc, nestco] = wmulden(x, level, wname, npc_app, ...
 npc_fin, tptr, sorh);
```

- 6** Display the number of retained principal components.

The second output argument gives the numbers of retained principal components for PCA for approximations and for final PCA.

```
npc
npc =
 2 2
```

As expected, since the signals are combinations of two initial ones, Kaiser's rule automatically detects that only two principal components are of interest.

- 7** Display the estimated noise covariance matrix.

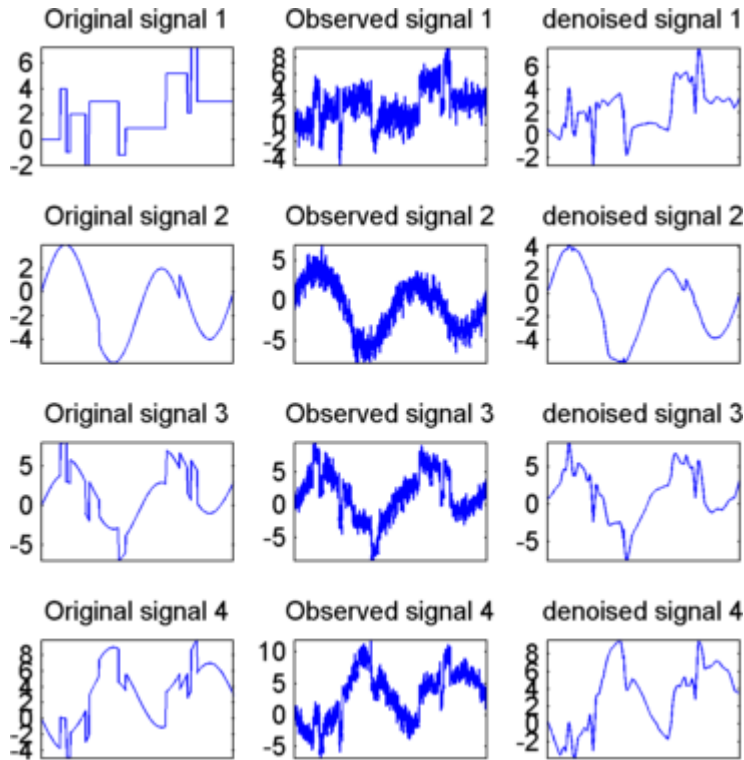
The third output argument contains the estimated noise covariance matrix:

```
nestco
nestco =
 1.0784 0.8333 0.6878 0.8141
 0.8333 1.0025 0.5275 0.6814
 0.6878 0.5275 1.0501 0.7734
 0.8141 0.6814 0.7734 1.0967
```

As you can see by comparing with the true matrix covar given previously, the estimation is satisfactory.

- 8** Display the original and final denoised signals by typing

```
kp = 0;
for i = 1:4
 subplot(4,3,kp+1), plot(x_orig(:,i));
 set(gca,'xtick',[]); axis tight;
 title(['Original signal ',num2str(i)]); set(gca,'xtick',[]);
 axis tight;
 subplot(4,3,kp+2), plot(x(:,i)); set(gca,'xtick',[]);
 axis tight;
 title(['Observed signal ',num2str(i)])
 subplot(4,3,kp+3), plot(x_den(:,i)); set(gca,'xtick',[]);
 axis tight;
 title(['denoised signal ',num2str(i)])
 kp = kp + 3;
end
```

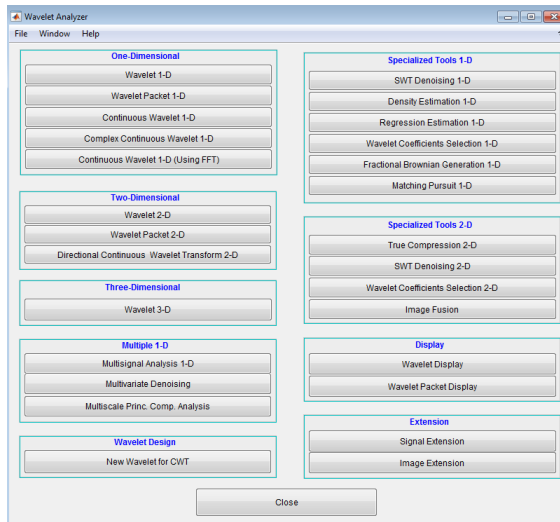


The results are better than those previously obtained. The first signal, which is irregular, is still correctly recovered, while the second signal, which is more regular, is denoised better after this second stage of PCA.

## Multivariate Wavelet Denoising Using the Wavelet Analyzer App

This section explores a denoising strategy for multivariate signals using the Wavelet Analyzer app.

- 1 Start the Multivariate Denoising Tool by first opening the Wavelet Analyzer app. Type `waveletAnalyzer` at the command line.

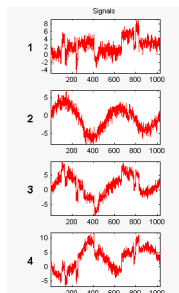


- 2 Click **Multivariate Denoising** to open the **Multivariate Denoising** portion of the app.
- 3 Load data.

At the MATLAB command prompt, type

```
load ex4mwden
```

In the **Multivariate Denoising** tool, select **File > Import from Workspace**. When the **Import from Workspace** dialog box appears, select the **x** variable. Click **OK** to import the noisy multivariate signal. The signal is a matrix containing four columns, where each column is a signal to be denoised.

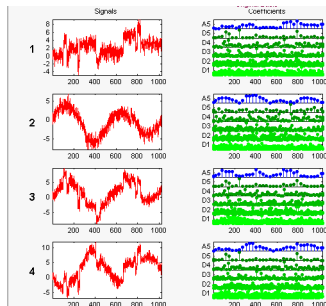


These signals are noisy versions from simple combinations of the two original signals. The first one is “Blocks” which is irregular and the second is “HeavySine” which is regular except around time 750. The other two signals are the sum and the difference between the original signals. Multivariate Gaussian white noise exhibiting strong spatial correlation is added to the resulting four signals.

The following example illustrates the two different aspects of the proposed denoising method. First, perform a convenient change of basis to cope with spatial correlation and denoise in the new basis. Then, use PCA to take advantage of the relationships between the signals, leading to an additional denoising effect.

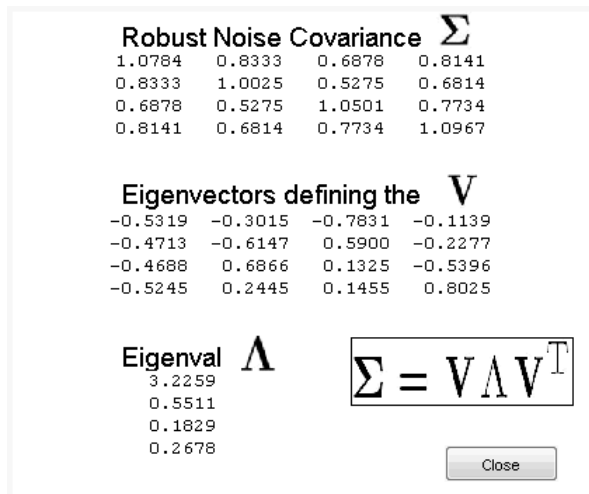
- 4 Perform a wavelet decomposition and diagonalize the noise covariance matrix.

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.

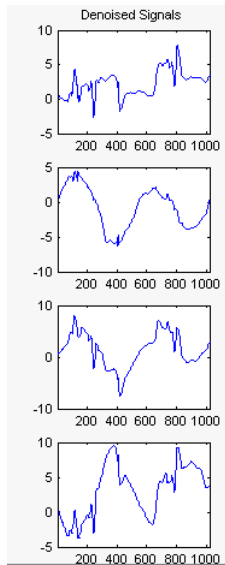
The multivariate denoising method proposed below is interesting if the noise covariance matrix is far from diagonal exhibiting spatial correlation, which, in this example, is the case.

- 5 denoise the multivariate signal.

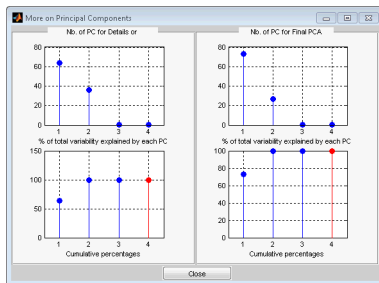
A number of options are available for fine-tuning the denoising algorithm. However, we will use the defaults: fixed form soft thresholding, scaled white noise model, and the proposed numbers of retained principal components. In this case, the default values for PCA lead to retaining all the components.

Select **Original Basis** to return to the original basis and then click **Denoise**.



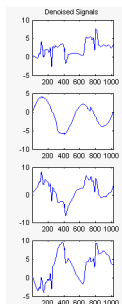


The results are satisfactory. Both of the two first signals are correctly recovered, but they can be improved by getting more information about the principal components. Click **More on Principal Components**.



A new figure displays information to select the numbers of components to keep for the PCA of approximations and for the final PCA after getting back to the original basis. You can see the percentages of variability explained by each principal component and the corresponding cumulative plot. Here, it is clear that only two principal components are of interest.

Close the **More on Principal Components** window. Select 2 as the **Nb. of PC for APP**. Select 2 as the **Nb. of PC for final PCA**, and then click **denoise**.



The results are better than those previously obtained. The first signal, which is irregular, is still correctly recovered. The second signal, which is more regular, is denoised better after this second stage of PCA. You can get more information by clicking **Residuals**.

## Importing and Exporting from the Wavelet Analyzer App

The tool lets you save denoised signals to disk by creating a MAT-file in the current folder with a name of your choice.

To save the signal denoised in the previous section,

- 1 Select **File > Save denoised Signals**.
- 2 Select **Save denoised Signals and Parameters**. A dialog box appears that lets you specify a folder and filename for storing the signal.
- 3 Type the name `s_ex4mwden` and click **OK** to save the data.
- 4 Load the variables into your workspace:

```
load s_ex4mwdent
whos
```

| Name       | Size   | Bytes | Class        |
|------------|--------|-------|--------------|
| DEN_Params | 1x1    | 430   | struct array |
| PCA_Params | 1x1    | 1536  | struct array |
| x          | 1024x4 | 32768 | struct array |

The denoised signals are in matrix `x`. The parameters (`PCA_Params` and `DEN_Params`) of the two-stage denoising 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 denoising 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.

## Wavelet 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 **Wavelet Analyzer** app. This section describes the command-line and app methods, and information about transferring signal and parameter information between the disk and the app.

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.

Since you can perform multiscale PCA either from the command line or using the app, this section has subsections covering each method.

### Multiscale Principal Components Analysis — Command Line

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

- Load a multivariate signal.
- Perform a simple multiscale PCA.
- Display the original and simplified signals.
- Improve the obtained result by retaining less principal components.

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

```
load ex4mwden
whos
```

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

The data stored in matrix  $x$  comes from two test signals, Blocks and HeavySine, and from their sum and difference, to which multivariate Gaussian white noise has been added.

**2** Perform a simple multiscale PCA.

The multiscale PCA combines noncentered PCA on approximations and details in the wavelet domain and a final PCA. At each level, the most significant principal components are selected.

First, set the wavelet parameters:

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

Then, automatically select the number of retained principal components using Kaiser's rule by typing

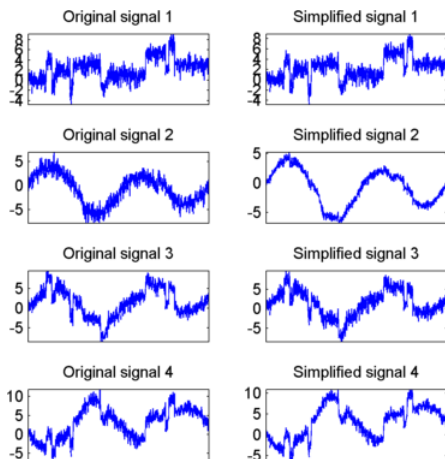
```
npc = 'kais';
```

Finally, perform multiscale PCA:

```
[x_sim, qual, npc] = wmspca(x ,level, wname, npc);
```

**3** Display the original and simplified signals:

```
kp = 0;
for i = 1:4
 subplot(4,2,kp+1), plot(x (:,i)); set(gca,'xtick',[]);
 axis tight;
 title(['Original signal ',num2str(i)])
 subplot(4,2,kp+2), plot(x_sim(:,i)); set(gca,'xtick',[]);
 axis tight;
 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,  $\text{npc}(d)$  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  $\text{npc}(6)$ , and  $\text{npc}(7)$  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

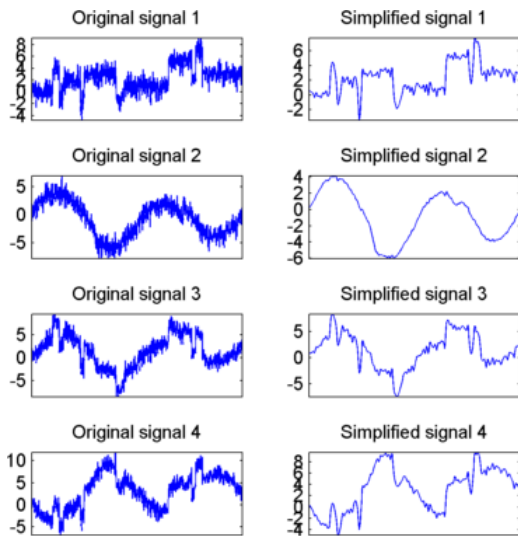
npc =
0 0 0 1 1 2 2
```

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)); set(gca,'xtick',[]);
 axis tight;
 title(['Original signal ',num2str(i)]); set(gca,'xtick',[]);
 axis tight;
 subplot(4,2,kp+2), plot(x_sim(:,i)); set(gca,'xtick',[]);
 axis tight;
 title(['Simplified signal ',num2str(i)])
 kp = kp + 2;
end
```



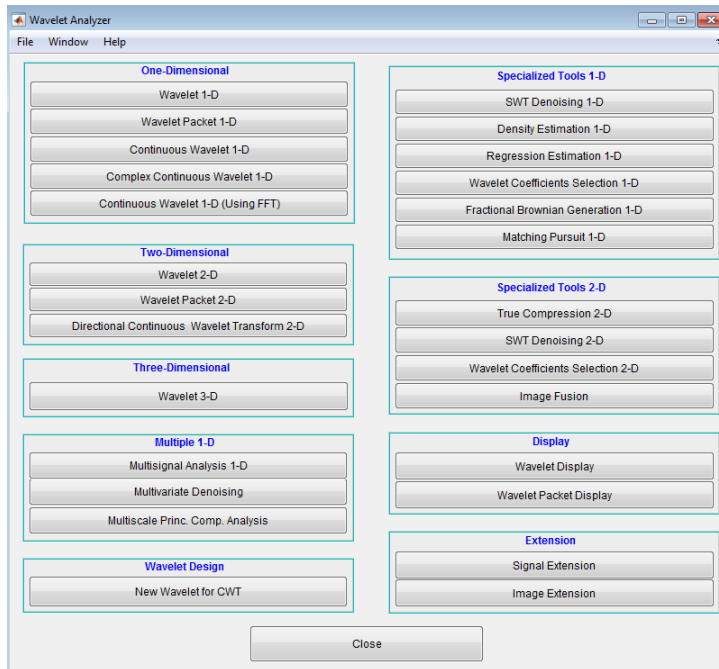
As shown, the results are improved.

### Multiscale Principal Components Analysis Using the Wavelet Analyzer App

This section explores multiscale PCA using the Wavelet Analyzer app.

- 1 Open the Wavelet Analyzer app by typing `waveletAnalyzer` at the command line.



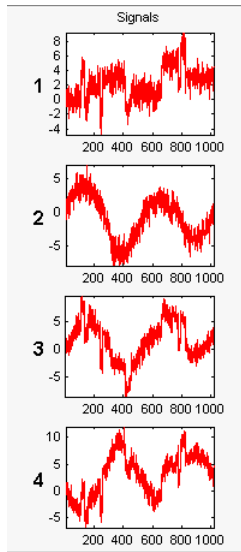


- 2 Click **Multiscale Princ. Comp. Analysis** to open the Multiscale Principal Components Analysis tool in the app.
- 3 Load data.

At the MATLAB command prompt, type

```
load ex4mwden
```

In the **Multiscale Princ. Comp. Analysis** tool, select **File > Import from Workspace**. When the **Import from Workspace** dialog box appears, select the **x** variable. Click **OK** to import the multivariate signal. The signal is a matrix containing four columns, where each column is a signal to be simplified.

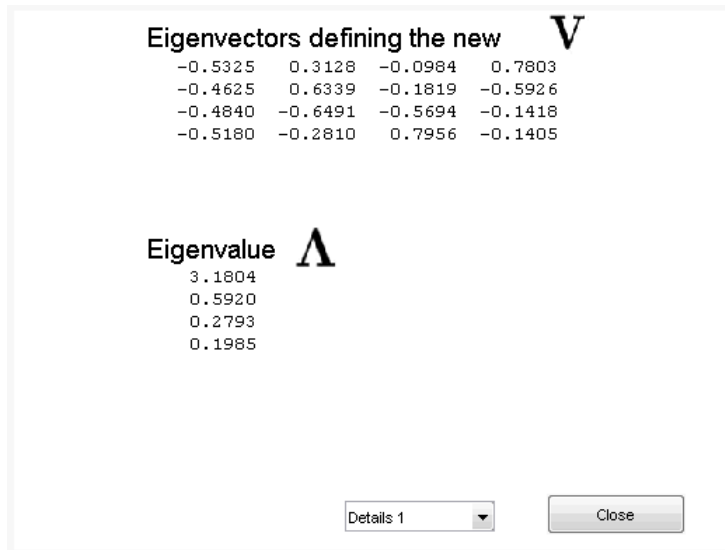


These signals are noisy versions from simple combinations of the two original signals, Blocks and HeavySine and their sum and difference, each with added multivariate Gaussian white noise.

- 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.

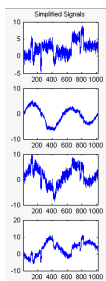
To get more information about the new bases allowed for performing a PCA for each scale, click **More on Adapted Basis**. A new figure displays the corresponding eigenvectors and eigenvalues for the matrix of the detail coefficients at level 1.



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**.



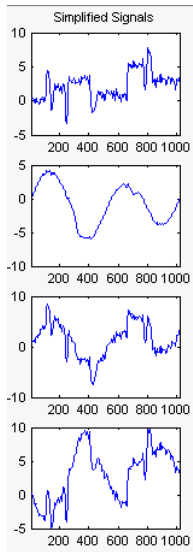
The results are good from a compression perspective.

**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, denoising effect.

For **D1**, **D2** and **D3**, select  $\theta$  as the **Nb. of non-centered PC** and click **Apply**.



The results are better than those previously obtained. The first signal, which is irregular, is still correctly recovered, while the second signal, which is more regular, is denoised better after this second stage of PCA. You can get more information by clicking **Residuals**.

### Importing and Exporting from the Wavelet Analyzer App

The Multiscale Principal Components Analysis tool lets you save the simplified signals to disk. The toolbox creates a MAT-file in the current folder with a name of your choice.

To save the simplified signals from the previous section:

- 1 Select **File > Save Simplified Signals**.
- 2 Select **Save Simplified Signals and Parameters**. A dialog box appears that lets you specify a folder and file name for storing the signal.
- 3 Type the name `s_ex4mwden` and click **OK** to save the data.

**4** Load the variables into your workspace:

```
load s_ex4mwden
whos
```

| Name       | Size   | Bytes | Class        |
|------------|--------|-------|--------------|
| PCA_Params | 1x7    | 2628  | struct array |
| x          | 1024x4 | 32768 | double array |

The simplified signals are in matrix `x`. The parameters of multiscale PCA are available in `PCA_Params`:

```
PCA_Params
```

```
PCA_Params =
1x7 struct array with fields:
 pc
 variances
 npc
```

`PCA_Params` is a structure array of length `d+2` (here, the maximum decomposition level `d=5`) such that `PCA_Params(d).pc` is the matrix of principal components. The columns are stored in descending order of the variances. `PCA_Params(d).variances` is the principal component variances vector, and `PCA_Params(d).npc` is the vector of selected numbers of retained principal components.

## Wavelet 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 denoising, 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  $N$ .

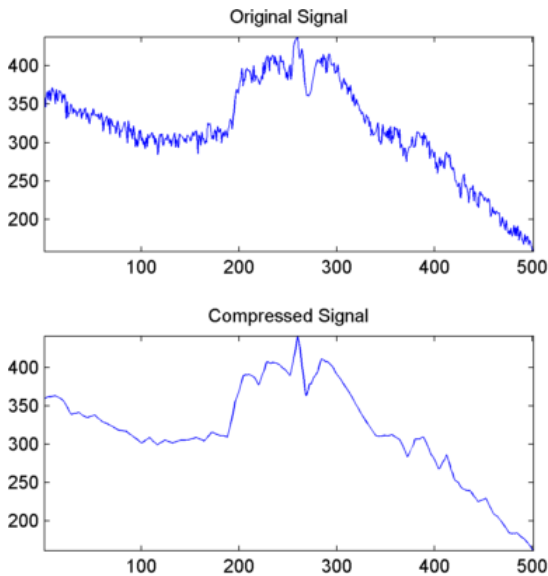
The difference of the denoising procedure is found in step **2**. There are two compression approaches available. The first consists of taking the wavelet expansion of the signal and keeping the largest absolute value coefficients. In this case, you can set a global threshold, a compression performance, or a relative square norm recovery performance.

Thus, only a single parameter needs to be selected. The second approach consists of applying visually determined level-dependent thresholds.

Let us examine two real-life examples of compression using global thresholding, for a given and unoptimized wavelet choice, to produce a nearly complete square norm recovery for a signal (see “Signal Compression” on page 6-81) and for an image (see “Image Compression” on page 6-82).

```
% 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);
```



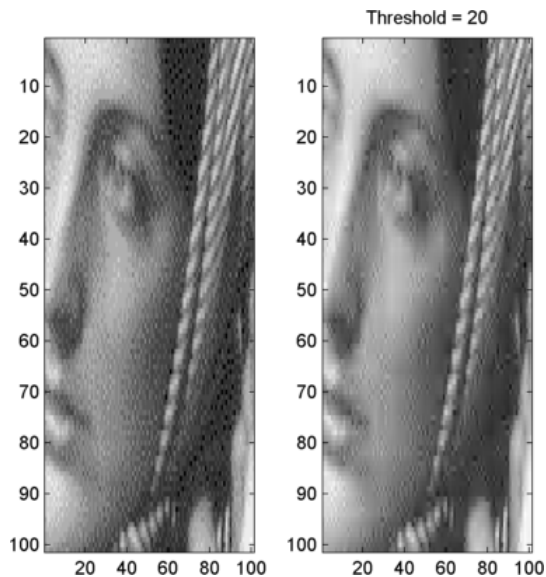
## Signal Compression

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);
```



### 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 (denoising or compression).

### Compression Scores

When compressing using orthogonal wavelets, the *Retained energy* in percentage is defined by

$$\frac{100 * (\text{vector-norm}(\text{coeffs of the current decomposition}, 2))^2}{(\text{vector-norm}(\text{original signal}, 2))^2}$$

When compressing using biorthogonal wavelets, the previous definition is not convenient. We use instead the *Energy ratio* in percentage defined by

$$\frac{100 * (\text{vector-norm}(\text{compressed signal}, 2))^2}{(\text{vector-norm}(\text{original signal}, 2))^2}$$



and as a tuning parameter the *Norm cfs recovery* defined by

$$\frac{100 * (\text{vector-norm}(\text{coeffs of the current decomposition}, 2))^2}{(\text{vector-norm}(\text{coeffs of the original decomposition}, 2))^2}$$

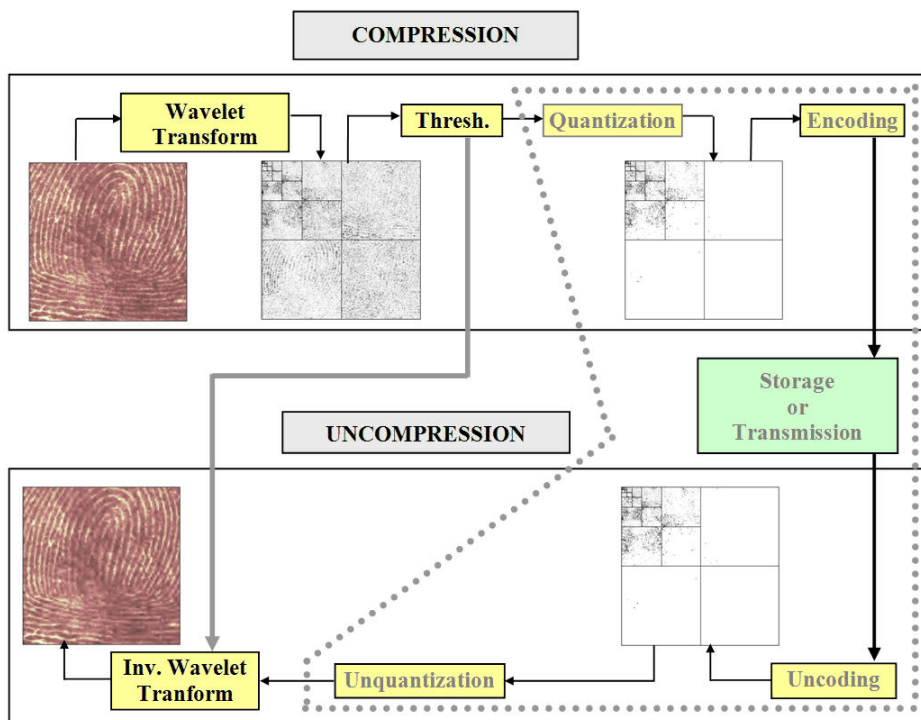
The *Number of zeros* in percentage is defined by

$$\frac{100 * (\text{number of zeros of the current decomposition})}{(\text{number of coefficients})}$$

## Wavelet Compression for Images

In “Wavelet Data Compression” on page 6-80, 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.

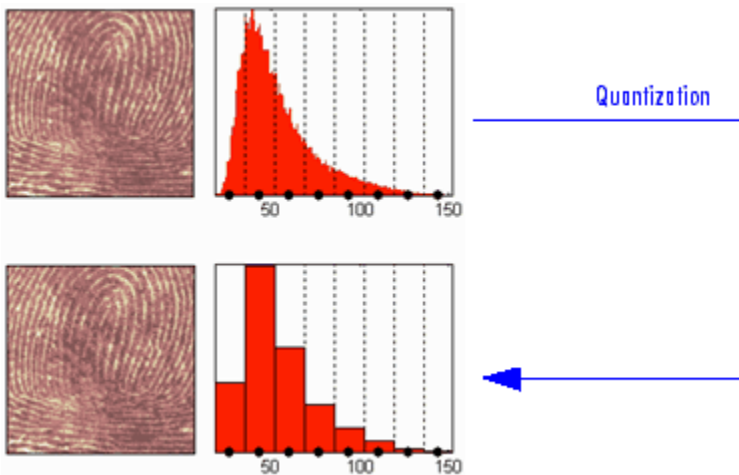
This more complex process can be represented by the following figure.



### 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.



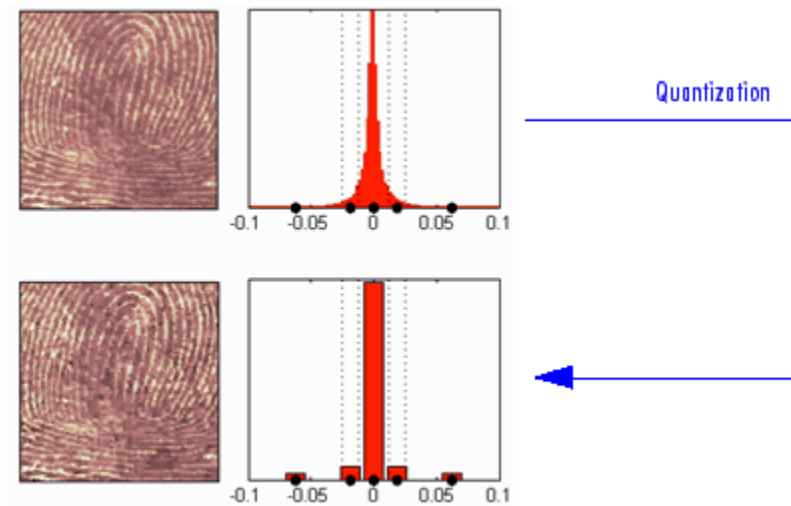
We can see on this figure:

- 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".

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".

The following table summarizes these methods, often called Progressive Coefficients Significance Methods (PCSM), and gives the MATLAB coded name used by the true compression tools for each of them.

| MATLAB Name | Compression Method Name                                        |
|-------------|----------------------------------------------------------------|
| 'ezw'       | Embedded Zerotree Wavelet                                      |
| 'spiht'     | Set Partitioning In Hierarchical Trees                         |
| 'stw'       | Spatial-orientation Tree Wavelet                               |
| 'wdr'       | Wavelet Difference Reduction                                   |
| 'aswdr'     | Adaptively Scanned Wavelet Difference Reduction                |
| 'spiht_3d'  | Set Partitioning In Hierarchical Trees 3D for truecolor images |

## Quantitative and Perceptual Quality Measures

The following quantitative measurements and measures of perceptual quality are useful for analyzing wavelet signals and images.

- **M S E** — Mean square error (MSE) is the squared norm of the difference between the data and the signal or image approximation divided by the number of elements. The MSE is defined by:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |X(i, j) - X_c(i, j)|^2$$

- **Max Error** — Maximum error is the maximum absolute squared deviation in the signal or image approximation.
- **L2-Norm Ratio** — L2-norm ratio is the ratio of the squared L2-norm of the signal or image approximation to the input signal or image. For images, the image is reshaped as a column vector before taking the L2-norm
- **P S N R** — Peak signal-to-noise ratio (PSNR) is a measure of the peak error in decibels. PSNR is meaningful only for data encoded in terms of bits per sample or bits per pixel. 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. When the PSNR is greater than 40 dB, then the two images are indistinguishable. The PSNR is defined by:

$$PSNR = 10 \cdot \log_{10} \left( \frac{255^2}{MSE} \right)$$

- **B P P** — Bits per pixel ratio (BPP) is the number of bits required to store one pixel of the image. The BPP is the compression ratio multiplied by 8, assuming one byte per pixel (8 bits).
- **Comp Ratio** — Compression ratio is ratio of the number of elements in the compressed image divided by the number of elements in the original image, expressed as a percentage.

## More Information on True Compression

You can find examples illustrating command-line mode and app tools for true compression in “Wavelet Compression for Images” on page 6-84 and the reference page for `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”.

## 2-D Wavelet Compression

This section takes you through the features of wavelet 2-D true compression using the Wavelet Toolbox software.

For more information on the compression methods see “Wavelet Compression for Images” on page 6-84 in the *Wavelet Toolbox User's Guide*.

For more information on the main function available when using command-line mode, see the `wcompress` reference pages.

Starting from a given image, the goal of the true compression is to minimize the length of the sequence of bits needed to represent it, while preserving information of acceptable quality. Wavelets contribute to effective solutions for this problem.

The complete chain of compression includes phases of quantization, coding and decoding in addition of the wavelet processing itself.

The purpose of this section is to show how to compress and uncompress a grayscale or truecolor image using various compression methods.

In this section, you'll learn to

- Compress using global thresholding and Huffman encoding
- Uncompress
- Compress using progressive methods
- Handle truecolor images

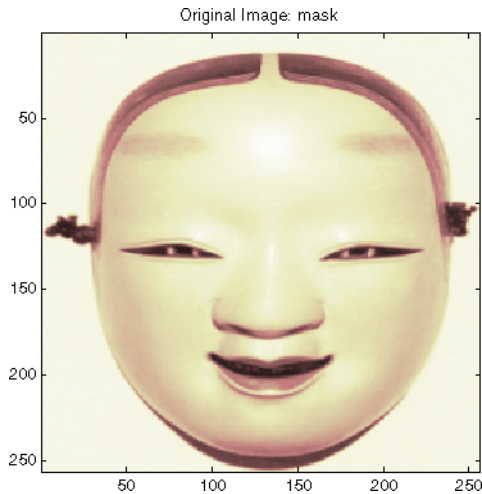
### 2-D Wavelet Compression Commands

#### 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 =

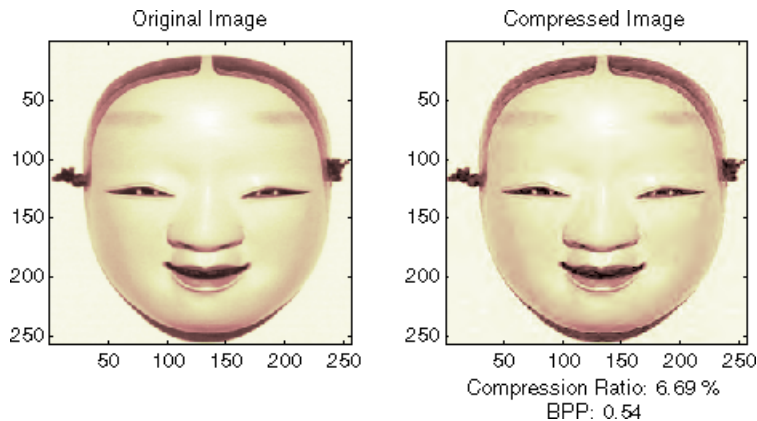
0.5354

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.

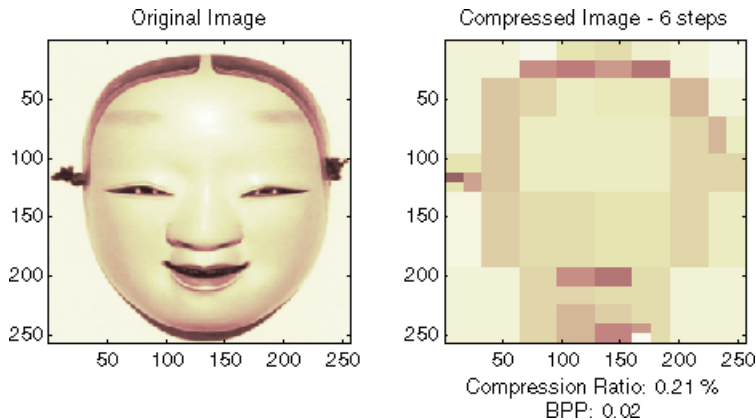
```
option = 'u'; % 'u' stands for uncompression
Xc = wcompress(option,'mask.wtc');
colormap(pink(255))
subplot(1,2,1); image(X);
axis square;
title('Original Image')
subplot(1,2,2); image(Xc);
axis square;
title('Compressed Image')
xlabel({'Compression Ratio: ' num2str(CR,'%1.2f %')'}, ...
 ['BPP: ' 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.

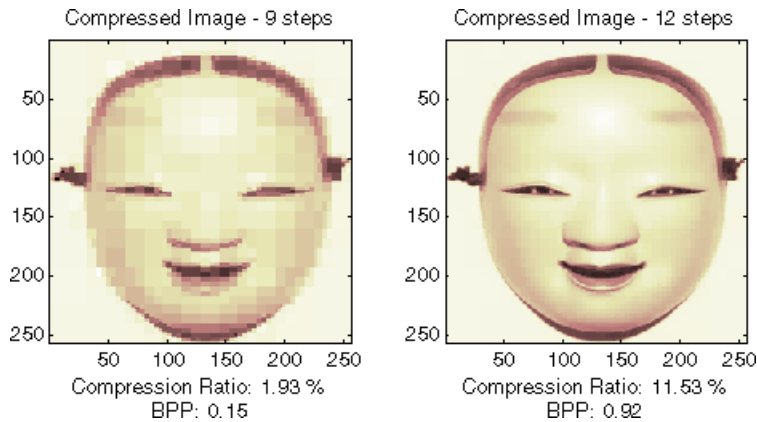
```
[CR,BPP]= wcompress('c',X,'mask.wtc',meth,'maxloop',9, ...
 'wname','haar');
Xc = wcompress('u','mask.wtc');
colormap(pink(255))
subplot(1,2,1); image(Xc);
```

```

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.

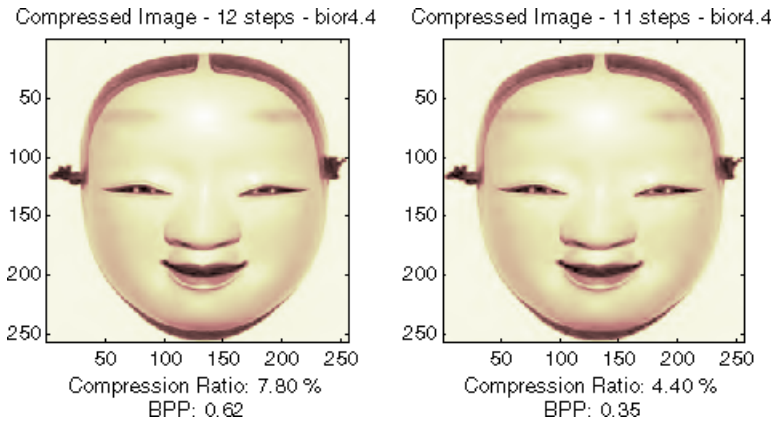
Let us try to improve it by using the wavelet *bior4.4* instead of *haar* and looking at obtained results for steps 12 and 11.

```

[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');

```

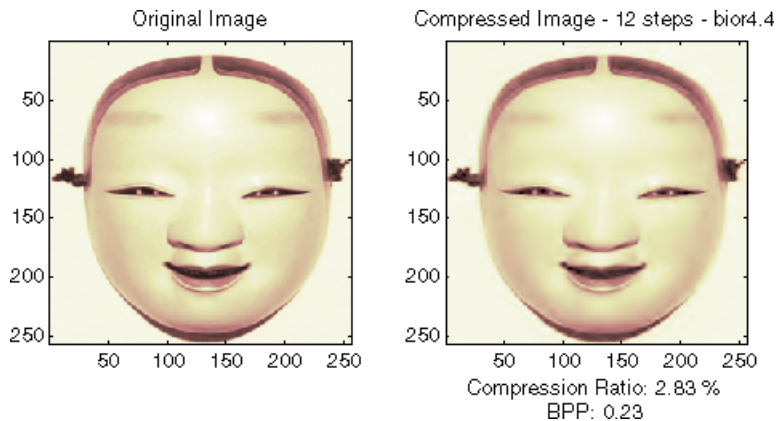
```
Xc = wcompress('u','mask.wtc');
subplot(1,2,2); image(Xc);
axis square;
title('Compressed Image - 11 steps - bior4.4')
xlabel({'Compression Ratio: ' num2str(CR,'%1.2f %%')', ...
 ['BPP: ' num2str(BPP,'%3.2f')']})
```



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,maxerr,l2rat] = measerr(X,Xc)

delete('mask.wtc')
```



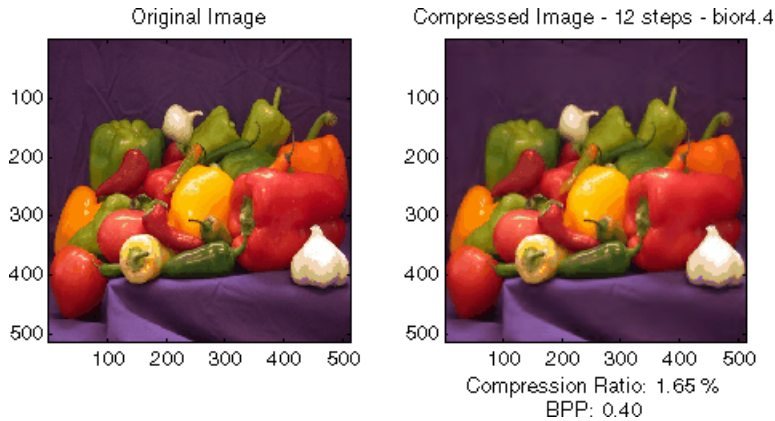
The final compression ratio (2.8%) and the Bit-Per-Pixel ratio (0.23) are very satisfactory. Let us recall that the first ratio means that the compressed image is stored using only 2.8% of the initial storage size.

### 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.

## 2-D Wavelet Compression using the Wavelet Analyzer App

In this section, we explore the different methods for 2-D true compression, using the Wavelet Analyzer app.

- 1 Start the True Compression 2-D Tool.

From the MATLAB prompt, type `waveletAnalyzer`.

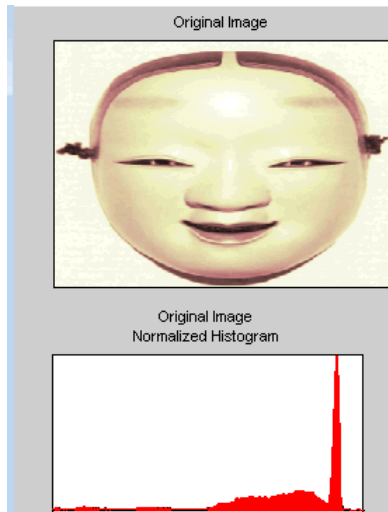
The **Wavelet Analyzer** appears. Click the **True Compression 2-D** item. The true compression tool for images appears.

- 2 Load the image.

At the MATLAB command prompt, type

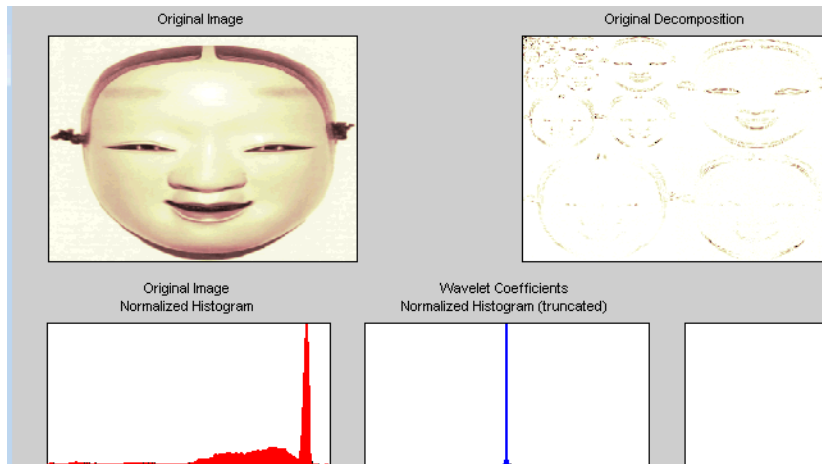
```
load mask
```

In the **True Compression 2-D** tool, select **File > Import Image from Workspace**. When the **Import from Workspace** dialog box appears, select the X variable. Click **OK** to import the data. The image 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.

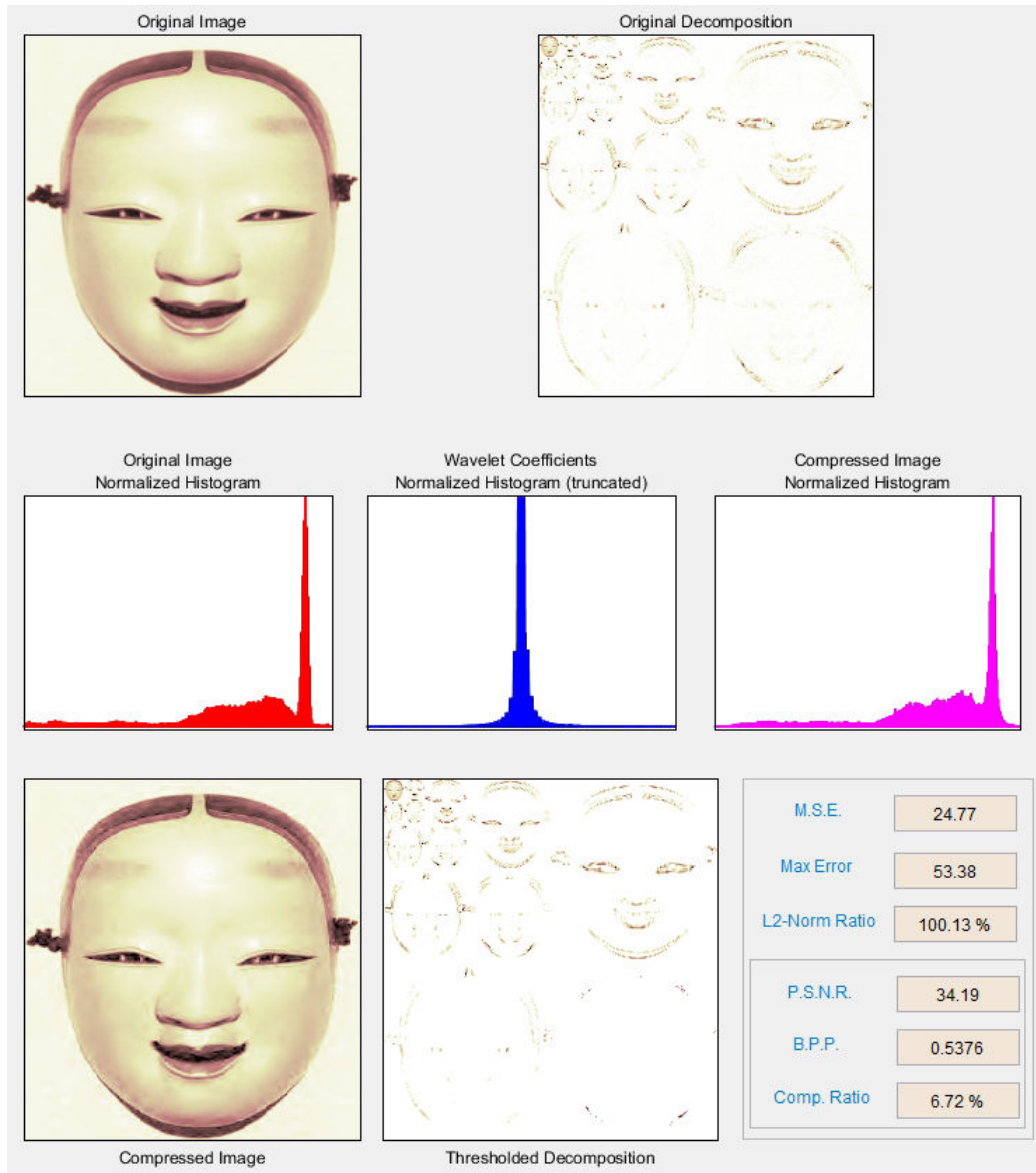
Begin with a simple method cascading global coefficients thresholding and Huffman encoding.

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 “Wavelet Compression for Images” on page 6-84 in the *Wavelet Toolbox User's Guide*.

Set the desired Bit-Per-Pixel ratio to **0.5**.

The image shows a software dialog box titled "Compression Parameters". It contains several input fields and a button. The "Compression Method" is a dropdown menu set to "GBL\_MMC\_H". Below it, "Bit Per Pixel (BPP)" is a text box with "0.500". "Compression Ratio" is a text box with "6.25" and a "%" symbol. "Nb. Kept Cfs." has a text box with "4325", a bidirectional arrow "<=>", another text box with "6.60", and a "%" symbol. "Threshold" is a text box with "12.2596". "Nb. Symbols for Quantization" is a text box with "75". At the bottom is a "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%.

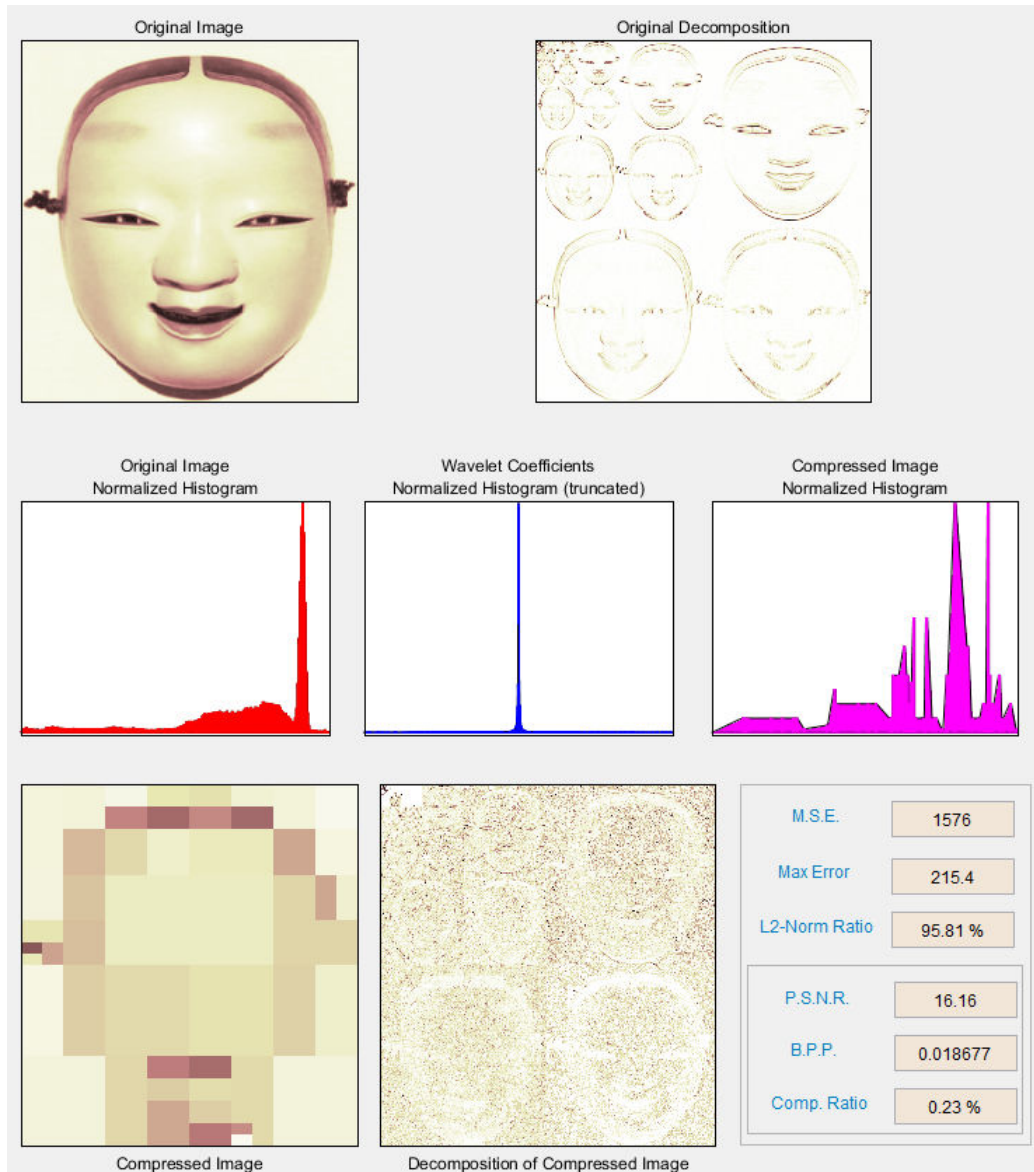
The compressed image, at the bottom left, can be compared with the original image.

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.

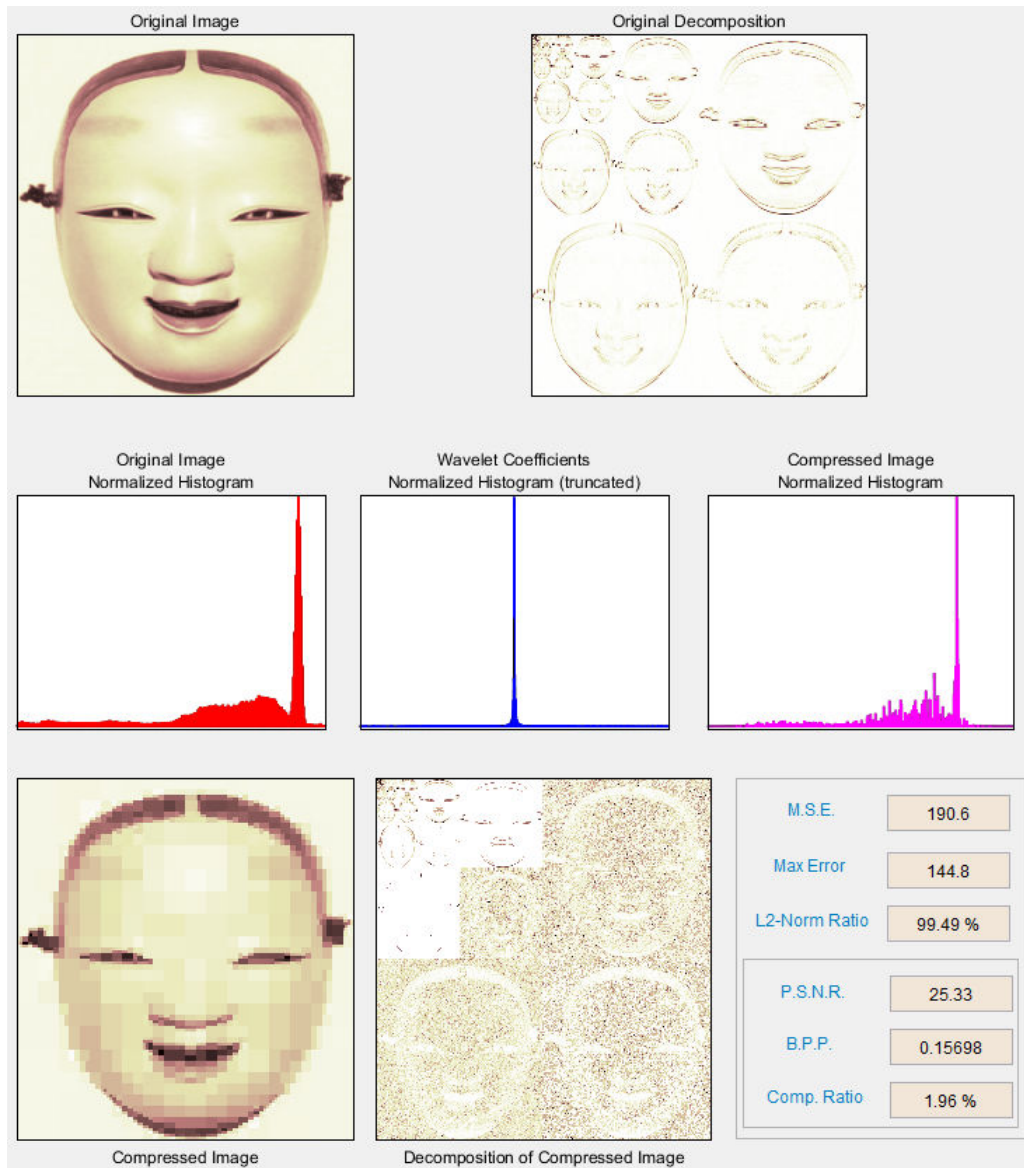
Let us now illustrate the use of progressive methods starting with the well known EZW algorithm. Let us start by selecting the wavelet **haar** from the **Wavelet** menu and select **8** from the **Level** menu. Then click the **Decompose** button.

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.

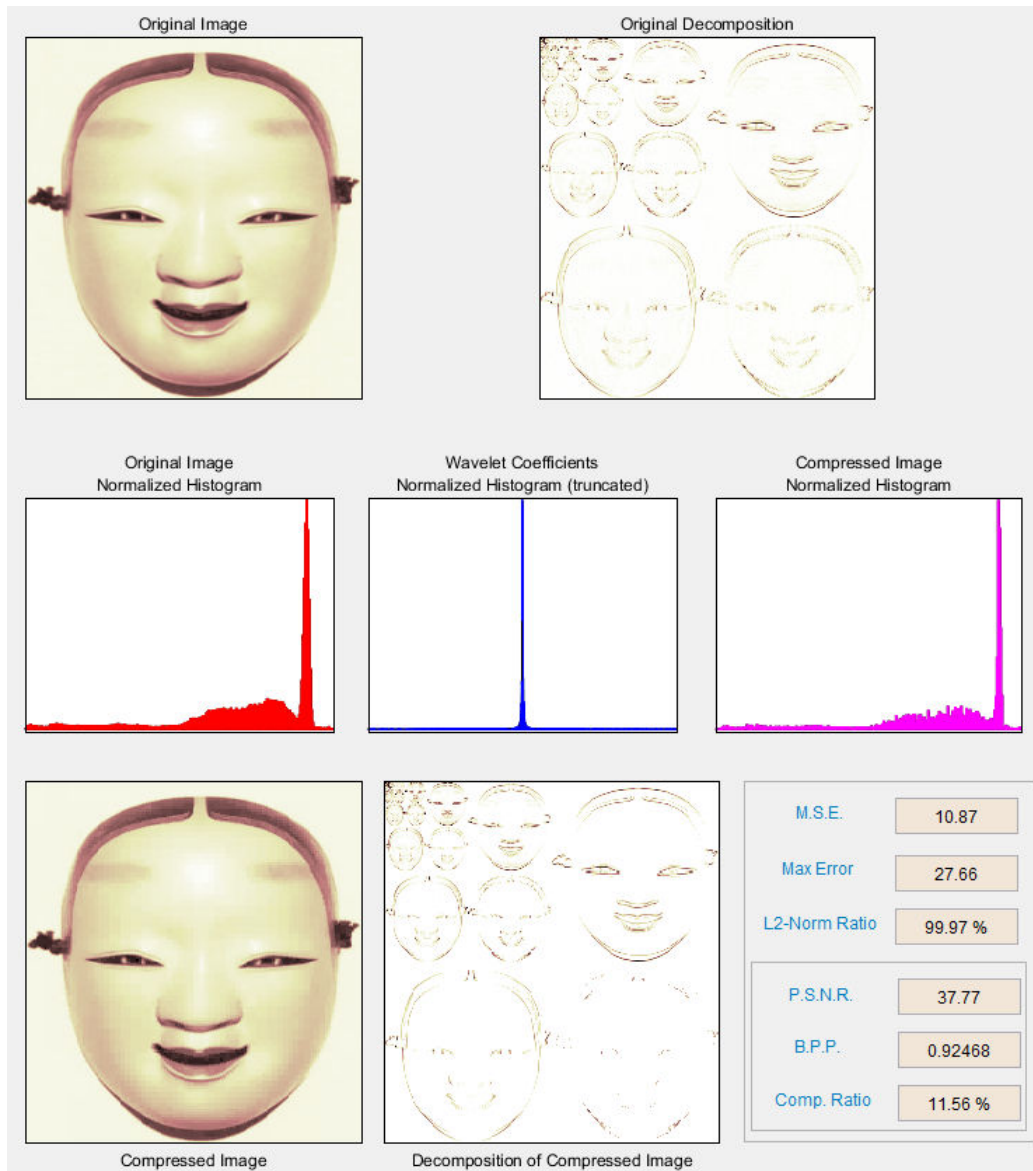
Too few steps produce a very coarse compressed image. So let us examine a little better result for **9** steps. Set the number of encoding loops to 9 and click the **Compress** button.



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     | 144.8   |
| L2-Norm Ratio | 99.49 % |
| P.S.N.R.      | 25.33   |
| B.P.P.        | 0.15698 |
| Comp. Ratio   | 1.96 %  |

Now try a large enough number of steps to get a satisfactory result. Set the number of encoding loops to **12** and click the **Compress** button.



The result is now acceptable. But for 12 steps, we attain a Bit-Per-Pixel ratio about 0.92.



- 7 Get better compression performance by changing the wavelet and selecting the best adapted number of loops.

Let us try to improve the compression performance by changing the wavelet: select **bior4.4** instead of **haar** and then click the **Decompose** button.

In order to select the number of loops, the Wavelet Analyzer app 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.

The image shows a dialog box titled "Compression Parameters". It contains the following elements:

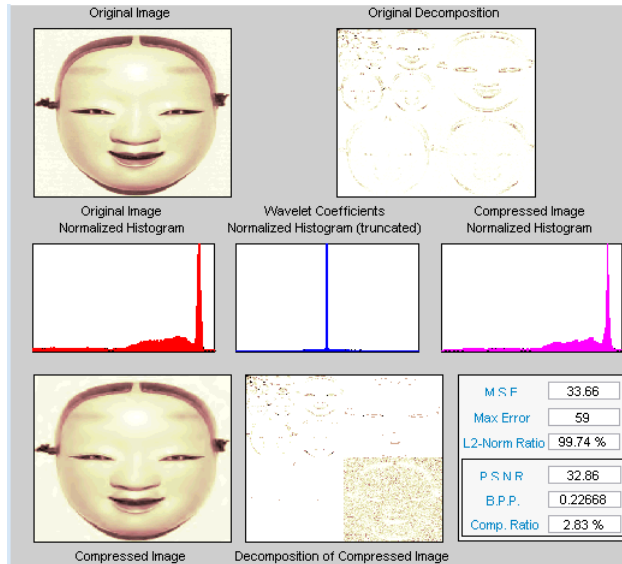
- Compression Method:** A dropdown menu currently showing "EZW".
- Nb. Encoding Loops:** A dropdown menu currently showing "13".
- Show Compression Steps:** A checked checkbox.
- Stepwise:** A checked checkbox.
- Next >>** and **Finish** buttons.
- Compress** button at the bottom.

Then, click the **Compress** button. Thirteen progressively more finely compressed images appear, and you can then select visually a convenient value for the number of loops. A satisfactory result seems to be obtained after 11 loops. So, you can set the number of encoding loops to **11** and click the **Compress** button.

The reached BPP ratio is now about 0.35 which is commonly considered a very satisfactory result. Nevertheless, it can be slightly improved by using a more recent method SPIHT (Set Partitioning In Hierarchical Trees).

- 8 Obtain a final compressed image by using a third method.

Choose the **SPIHT** option from the menu **Compression method**, set the number of encoding loops to **12**, and click the **Compress** button.



The final compression ratio and the Bit-Per-Pixel ratio are very satisfactory: 2.8% and 0.22. Let us recall that the first ratio means that the compressed image is stored using only 2.8% of the initial storage size.

9 Handle truecolor images.

Finally, let us illustrate how to compress truecolor images. The truecolor images can be compressed along the same scheme by applying the same strategies to each of the three-color components.

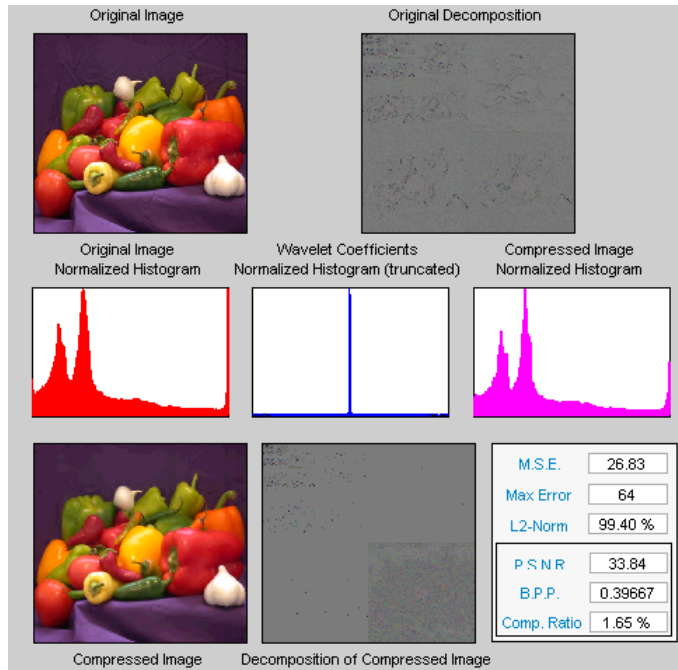
From the **File** menu, choose the **Load Image** option and select the **Matlab Supported Formats** item.

When the **Load Image** dialog box appears, select the MAT-file `wpeppers.jpg` which should reside in the MATLAB folder `toolbox/wavelet/wavelet`.

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.

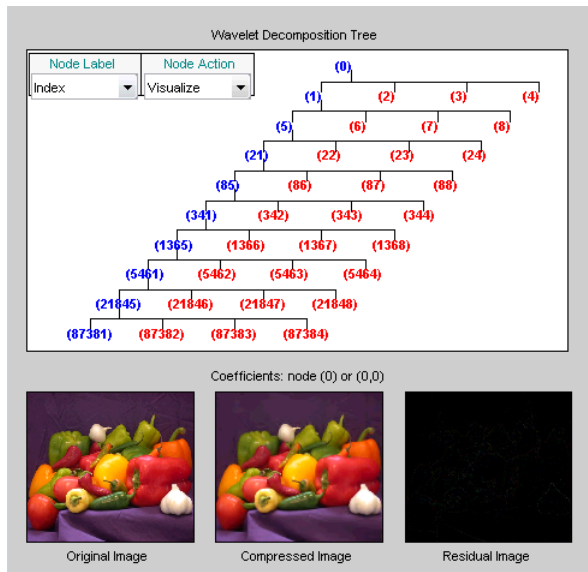
Then, accept the default compression method **SPIHT** and set the number of encoding loops to **12**. Click the **Compress** button.

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 Wavelet Analyzer App

You can save the compressed image to disk in the current folder with a name of your choice.

The Wavelet Toolbox compression tools can create a file using either one of the Matlab Supported Format types or a specific format which can be recognized by the extension of the file: **wtc** (Wavelet Toolbox Compressed).

To save the above compressed image, use the menu option **File > Save Compressed Image > Wavelet Toolbox Compressed Image**. A dialog box appears that lets you specify a folder and filename for storing the image. Of course, the use of the **wtc** format requires you to uncompress the stored image using the Wavelet Toolbox true compression tools.

# Univariate Wavelet Regression

This section takes you through the features of 1-D wavelet regression estimation using one of the Wavelet Toolbox specialized tools. The toolbox provides a Wavelet Analyzer app to explore some denoising schemes for equally or unequally sampled data.

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

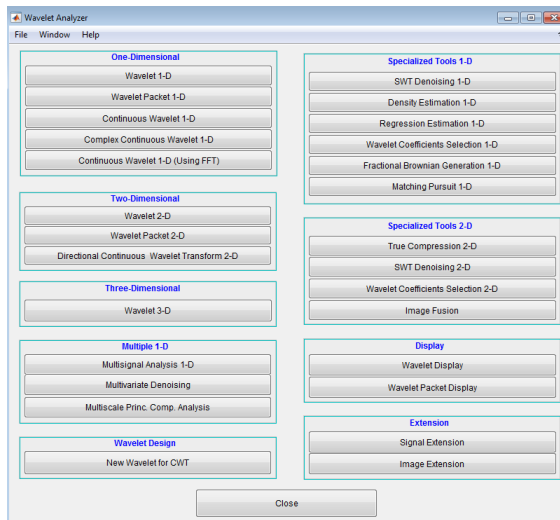
```
dwtmode('sym')
```

## Regression for Equally-Spaced Observations

- 1 Start the Regression Estimation 1-D Tool.

From the MATLAB prompt, type `waveletAnalyzer`.

The **Wavelet Analyzer** appears.



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

- 2 Load data.

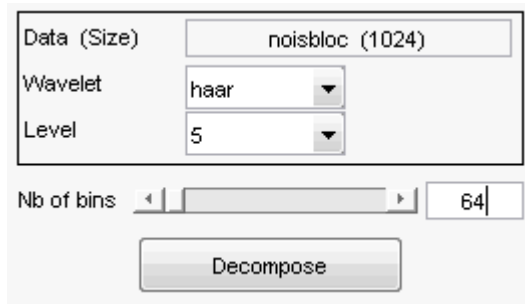
At the MATLAB command prompt, type

```
load blocregdata;
```

In the **Regression Estimation 1-D** tool, select **File > Import from Workspace**. When the **Import from Workspace** dialog box appears, select the `blocregdata` data. Click **OK** to import the data. The loaded data and 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.

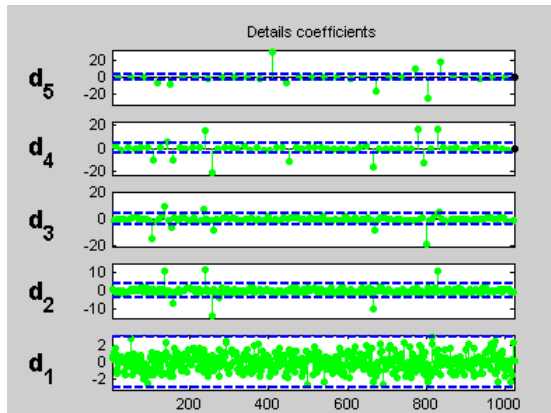


The binned data appears to be very smoothed. Select 1000 from the **Nb bins** edit and press **Enter** or use the slider. The new data to be processed appears.

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.

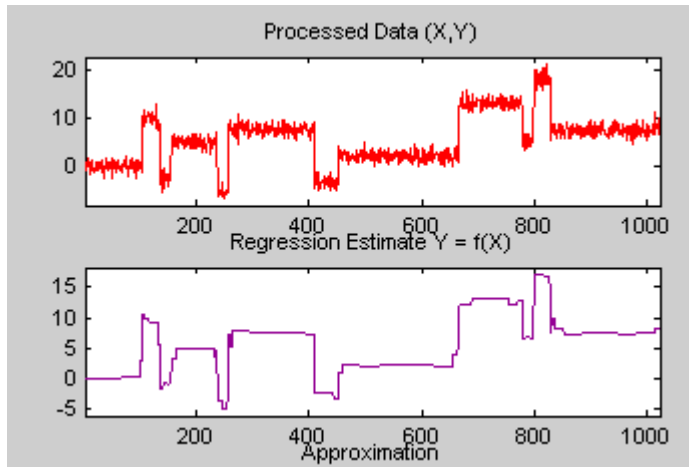
Select the `haar` wavelet from the **Wavelet** menu and select **5** from the **Level** menu, and then click the **Decompose** button. After a pause for computation, the tool displays the detail coefficients of the decomposition.



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 Wavelet Analyzer app for randomly or irregularly spaced observations, focusing on the differences from the previous situation.

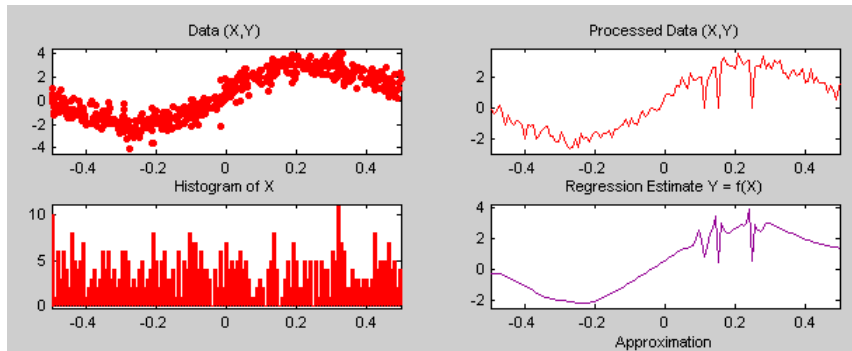
### Regression for Randomly-Spaced Observations

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

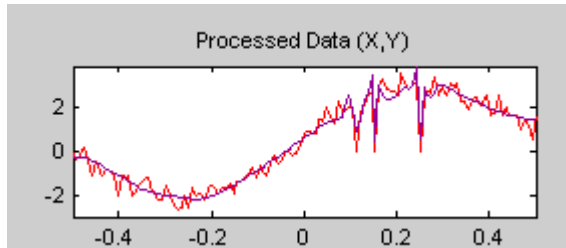
The loaded data denoted  $(X,Y)$ , the histogram of  $X$ , and the processed data obtained after a binning are displayed. The histogram is interesting, because the values of  $X$  are randomly distributed. The binning step is essential since it transforms a problem of regression estimation for irregularly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

- 2 Select the `sym4` wavelet from the **Wavelet** menu, select **5** from the **Level** menu, and enter 125 in the **Nb bins** edit box. Click the **Decompose** button. The tool displays the detail coefficients of the decomposition.
- 3 From the **Select thresholding method** menu, select the item **Penalize low** and click the **Estimate** button.





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



## Importing and Exporting Information from the Wavelet Analyzer App

### Saving Function

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

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

```
load fex1nsto
whos
```

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

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

```
wname
```

```
wname =
 sym4
```

and the level dependent thresholds contained in `thrParams`, which is a cell array of length 5 (the level of the decomposition). For `i` from 1 to 5, `thrParams{i}` contains the lower and upper bounds of the interval of thresholding and the threshold value (since interval dependent thresholds are allowed). For more information, see “1-D Adaptive Thresholding of Wavelet Coefficients” on page 6-48 in the *Wavelet Toolbox User's Guide*.

For example, for level 1,

```
thrParams{1}
```

```
ans =
 -0.4987 0.4997 1.0395
```

### Loading Data

To load data for regression estimation, your data must be in the form of a structure array with exactly two fields. The fields must be named `xdata` and `ydata`, and must be the same length.

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

```
clear
load ex1nsto
whos
```

| <b>Name</b> | <b>Size</b> | <b>Bytes</b> | <b>Class</b> |
|-------------|-------------|--------------|--------------|
| xdata       | 1x500       | 4000         | double array |
| ydata       | 1x500       | 4000         | double array |

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

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



# Matching Pursuit

---

- “Matching Pursuit Algorithms” on page 7-2
- “Matching Pursuit” on page 7-8
- “Matching Pursuit Using Wavelet Analyzer App” on page 7-23

## Matching Pursuit Algorithms

| In this section...                                    |
|-------------------------------------------------------|
| “Redundant Dictionaries and Sparsity” on page 7-2     |
| “Nonlinear Approximation in Dictionaries” on page 7-3 |
| “Basic Matching Pursuit” on page 7-3                  |
| “Orthogonal Matching Pursuit” on page 7-6             |
| “Weak Orthogonal Matching Pursuit” on page 7-6        |

### Redundant Dictionaries and Sparsity

Representing a signal in a particular basis involves finding the unique set of expansion coefficients in that basis. While there are many advantages to signal representation in a basis, particularly an orthogonal basis, there are also disadvantages.

The ability of a basis to provide a sparse representation depends on how well the signal characteristics match the characteristics of the basis vectors. For example, smooth continuous signals are sparsely represented in a Fourier basis, while impulses are not. A smooth signal with isolated discontinuities is sparsely represented in a wavelet basis. However, a wavelet basis is not efficient at representing a signal whose Fourier transform has narrow high frequency support.

Real-world signals often contain features that prohibit sparse representation in any single basis. For these signals, you want the ability to choose vectors from a set not limited to a single basis. Because you want to ensure that you can represent every vector in the space, the *dictionary* of vectors you choose from must span the space. However, because the set is not limited to a single basis, the dictionary is not linearly independent.

Because the vectors in the dictionary are not a linearly independent set, the signal representation in the dictionary is not unique. However, by creating a redundant dictionary, you can expand your signal in a set of vectors that adapt to the time-frequency or time-scale characteristics of your signal. You are free to create a dictionary consisting of the union of several bases. For example, you can form a basis for the space of square-integrable functions consisting of a wavelet packet basis and a local cosine basis. A wavelet packet basis is well adapted to signals with different behavior in different frequency intervals. A local cosine basis is well adapted to signals with different behavior in different time intervals. The ability to choose vectors from each of these bases greatly increases your ability to sparsely represent signals with varying characteristics.

## Nonlinear Approximation in Dictionaries

Define a *dictionary* as a collection of unit-norm elementary building blocks for your signal space. These unit-norm vectors are called *atoms*. If the atoms of the dictionary span the entire signal space, the dictionary is *complete*.

If the dictionary atoms form a linearly-dependent set, the dictionary is *redundant*. In most applications of matching pursuit, the dictionary is complete and redundant.

Let  $\{\phi_k\}$  denote the atoms of a dictionary. Assume the dictionary is complete and redundant. There is no unique way to represent a signal from the space as a linear combination of the atoms.

$$x = \sum_k \alpha_k \phi_k$$

An important question is whether there exists a *best* way. An intuitively satisfying way to choose the *best* representation is to select the  $\phi_k$  yielding the largest inner products (in absolute value) with the signal. For example, the best single  $\phi_k$  is

$$\max_k \left| \langle x, \phi_k \rangle \right|$$

which for a unit-norm atom is the magnitude of the scalar projection onto the subspace spanned by  $\phi_k$ .

The central problem in *matching pursuit* is how you choose the optimal  $M$ -term expansion of your signal in a dictionary.

## Basic Matching Pursuit

Let  $\Phi$  denote the dictionary of atoms as a  $N$ -by- $M$  matrix with  $M > N$ . If the complete, redundant dictionary forms a frame for the signal space, you can obtain the minimum  $L_2$  norm expansion coefficient vector by using the frame operator.

$$\Phi^\dagger = \Phi^*(\Phi \Phi^*)^{-1}$$

However, the coefficient vector returned by the frame operator does not preserve sparsity. If the signal is sparse in the dictionary, the expansion coefficients obtained with the canonical frame operator generally do not reflect that sparsity. Sparsity of your signal in the dictionary is a trait that you typically want to preserve. Matching pursuit addresses sparsity preservation directly.

Matching pursuit is a greedy algorithm that computes the best nonlinear approximation to a signal in a complete, redundant dictionary. Matching pursuit builds a sequence of sparse approximations to the signal stepwise. Let  $\Phi = \{\phi_k\}$  denote a dictionary of unit-norm atoms. Let  $f$  be your signal.

- 1 Start by defining  $R^0 f = f$
- 2 Begin the matching pursuit by selecting the atom from the dictionary that maximizes the absolute value of the inner product with  $R^0 f = f$ . Denote that atom by  $\phi_p$ .
- 3 Form the residual  $R^1 f$  by subtracting the orthogonal projection of  $R^0 f$  onto the space spanned by  $\phi_p$ .

$$R^1 f = R^0 f - \langle R^0 f, \phi_p \rangle \phi_p$$

- 4 Iterate by repeating steps 2 and 3 on the residual.

$$R^{m+1} f = R^m f - \langle R^m f, \phi_k \rangle \phi_k$$

- 5 Stop the algorithm when you reach some specified stopping criterion.

In *nonorthogonal* (or basic) matching pursuit, the dictionary atoms are not mutually orthogonal vectors. Therefore, subtracting subsequent residuals from the previous one can introduce components that are not orthogonal to the span of previously included atoms.

To illustrate this, consider the following example. The example is not intended to present a working matching pursuit algorithm.

Consider the following dictionary for Euclidean 2-space. This dictionary is an equal-norm frame.

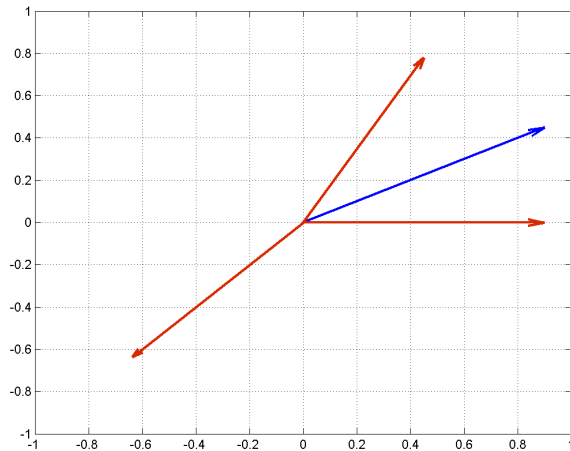
$$\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1/2 \\ \sqrt{3}/2 \end{pmatrix}, \begin{pmatrix} -1/\sqrt{2} \\ -1/\sqrt{2} \end{pmatrix} \right\}$$

Assume you have the following signal.

$$\begin{pmatrix} 1 \\ 1/2 \end{pmatrix}$$

The following figure illustrates this example. The dictionary atoms are in red. The signal vector is in blue.





Construct this dictionary and signal in MATLAB.

```
dictionary = [1 0; 1/2 sqrt(3)/2; -1/sqrt(2) -1/sqrt(2)]';
x = [1 1/2]';
```

Compute the inner (scalar) products between the signal and the dictionary atoms.

```
scalarproducts = dictionary'*x;
```

The largest scalar product in absolute value occurs between the signal and  $[-1/\sqrt{2}; -1/\sqrt{2}]$ . This is clear because the angle between the two vectors is almost  $\pi$  radians.

Form the residual by subtracting the orthogonal projection of the signal onto  $[-1/\sqrt{2}; -1/\sqrt{2}]$  from the signal. Next, compute the inner products of the residual (new signal) with the remaining dictionary atoms. It is not necessary to include  $[-1/\sqrt{2}; -1/\sqrt{2}]$  because the residual is orthogonal to that vector by construction.

```
residual = x - scalarproducts(3)*dictionary(:,3);
scalarproducts = dictionary(:,1:2)'*residual;
```

The largest scalar product in absolute value is obtained with  $[1; 0]$ . The best two atoms in the dictionary from two iterations are  $[-1/\sqrt{2}; -1/\sqrt{2}]$  and  $[1; 0]$ . If you iterate on the residual, you see that the output is no longer orthogonal to the first

atom chosen. In other words, the algorithm has introduced a component that is not orthogonal to the span of the first atom selected. This fact and the associated complications with convergence argues in favor of “Orthogonal Matching Pursuit” on page 7-6 (OMP).

## Orthogonal Matching Pursuit

In orthogonal matching pursuit (OMP), the residual is always orthogonal to the span of the atoms already selected. This results in convergence for a  $d$ -dimensional vector after at most  $d$  steps.

Conceptually, you can do this by using Gram-Schmidt to create an orthonormal set of atoms. With an orthonormal set of atoms, you see that for a  $d$ -dimensional vector, you can find at most  $d$  orthogonal directions.

The OMP algorithm is:

- 1 Denote your signal by  $f$ . Initialize the residual  $R^0 f = f$ .
- 2 Select the atom that maximizes the absolute value of the inner product with  $R^0 f = f$ . Denote that atom by  $\varphi_p$ .
- 3 Form a matrix,  $\Phi$ , with previously selected atoms as the columns. Define the orthogonal projection operator onto the span of the columns of  $\Phi$ .

$$P = \Phi (\Phi^* \Phi)^{-1} \Phi^*$$

- 4 Apply the orthogonal projection operator to the residual.
- 5 Update the residual.

$$R^{m+1} f = (I - P) R^m f$$

$I$  is the identity matrix.

Orthogonal matching pursuit ensures that components in the span of previously selected atoms are not introduced in subsequent steps.

## Weak Orthogonal Matching Pursuit

It can be computationally efficient to relax the criterion that the selected atom maximizes the absolute value of the inner product to a less strict one.

$$|\langle x, \phi_p \rangle| \geq \alpha \max_k |\langle x, \phi_k \rangle|, \quad \alpha \in (0, 1]$$

This is known as *weak* matching pursuit.

## Matching Pursuit

### In this section...

“Matching Pursuit Dictionary Creation and Visualization” on page 7-8

“Orthogonal Matching Pursuit on a 1-D Signal” on page 7-10

“Electricity Consumption Analysis Using Matching Pursuit” on page 7-12

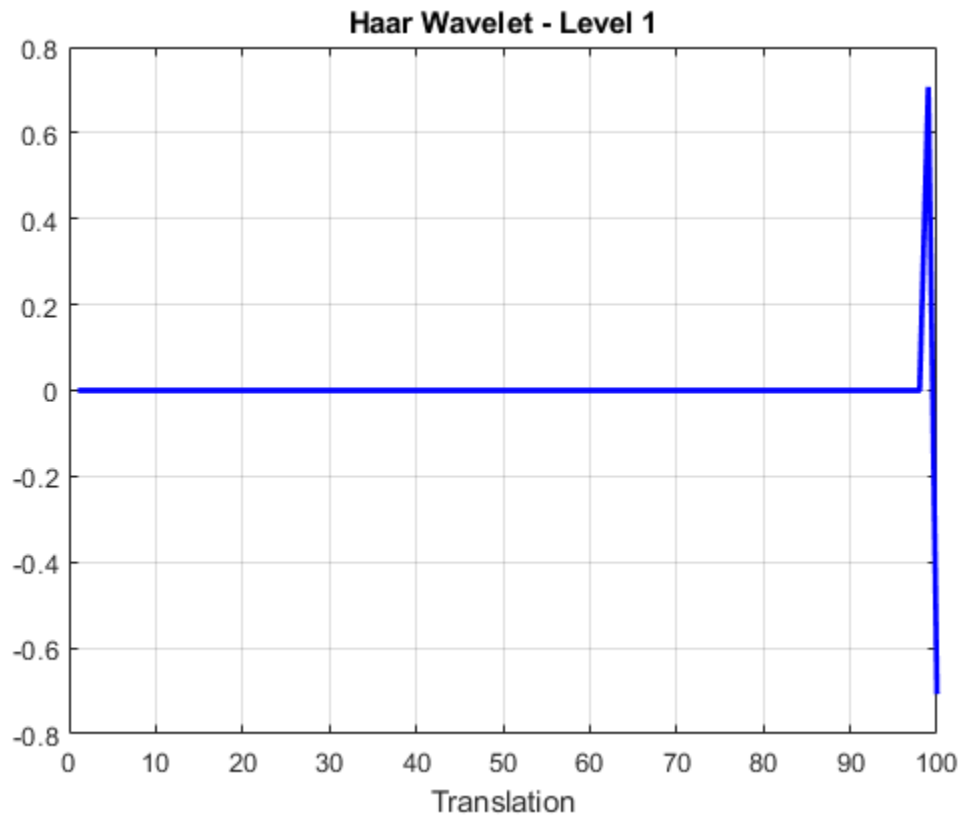
### Matching Pursuit Dictionary Creation and Visualization

This example shows how to create and visualize a dictionary consisting of a Haar wavelet down to level 2.

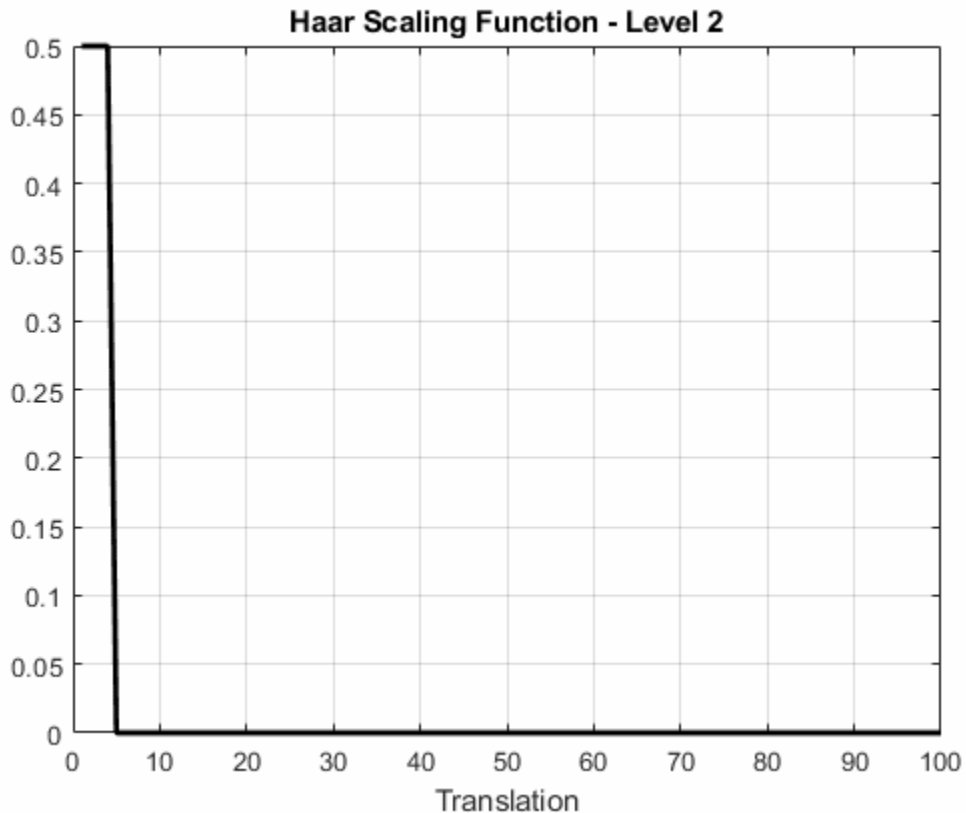
```
[mpdict,~,~,longs] = wmpdictionary(100,'lstdcpt',{{'haar',2}});
```

Use the `longs` output argument to divide the wavelet dictionary by level and type of function (scaling or wavelet). Step through a plot of the translated scaling functions and wavelets by level.

```
for nn = 1:size(mpdict,2)
 if (nn <= longs{1}(1))
 plot(mpdict(:,nn),'k','linewidth',2)
 grid on
 xlabel('Translation')
 title('Haar Scaling Function - Level 2')
 elseif (nn>longs{1}(1) && nn<= longs{1}(1)+longs{1}(2))
 plot(mpdict(:,nn),'r','linewidth',2)
 grid on
 xlabel('Translation')
 title('Haar Wavelet - Level 2')
 else
 plot(mpdict(:,nn),'b','linewidth',2)
 grid on
 xlabel('Translation')
 title('Haar Wavelet - Level 1')
 end
 pause(0.2)
end
```



This animation infinitely loops through all the plots generated.



## Orthogonal Matching Pursuit on a 1-D Signal

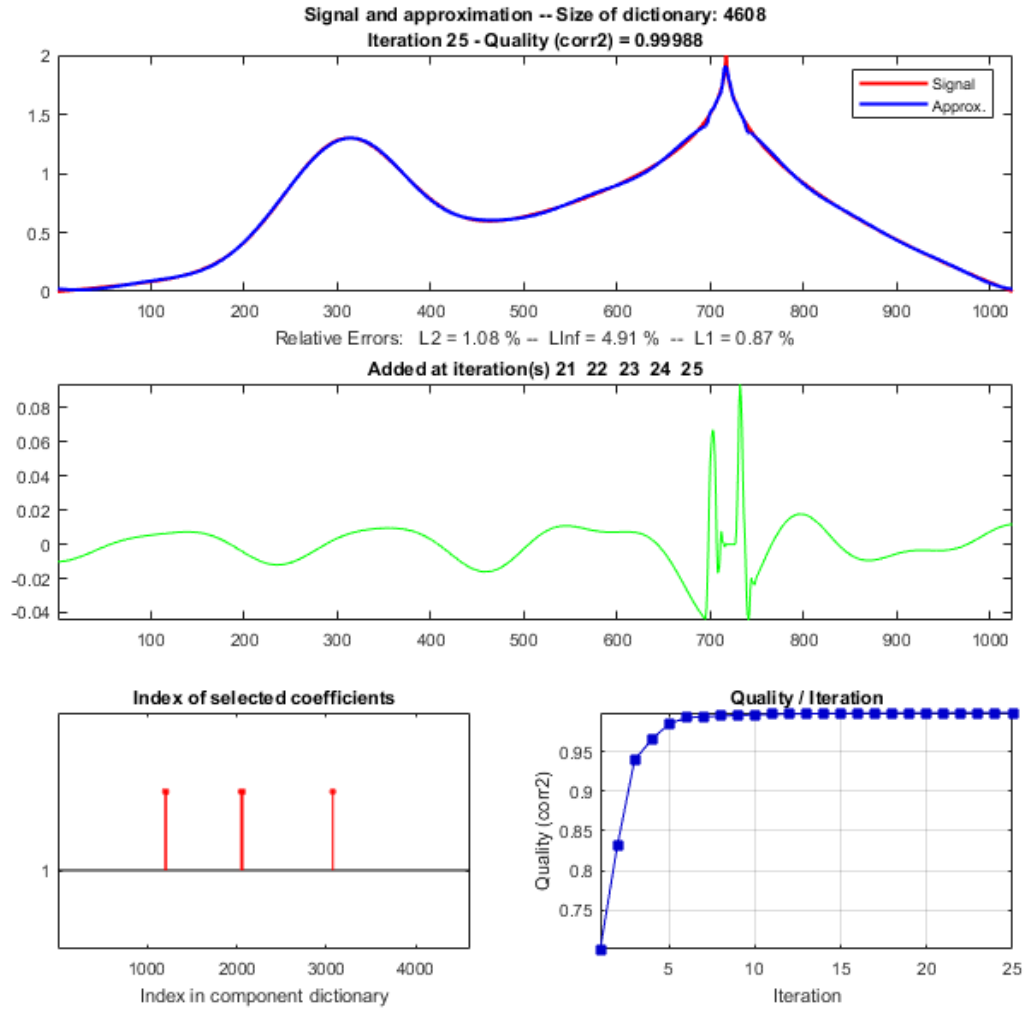
This example shows how to perform orthogonal matching pursuit on a 1-D input signal that contains a cusp.

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

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

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

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



## Electricity Consumption Analysis Using Matching Pursuit

This example shows how to compare matching pursuit with a nonlinear approximation in the discrete Fourier transform basis. The data is electricity consumption data collected



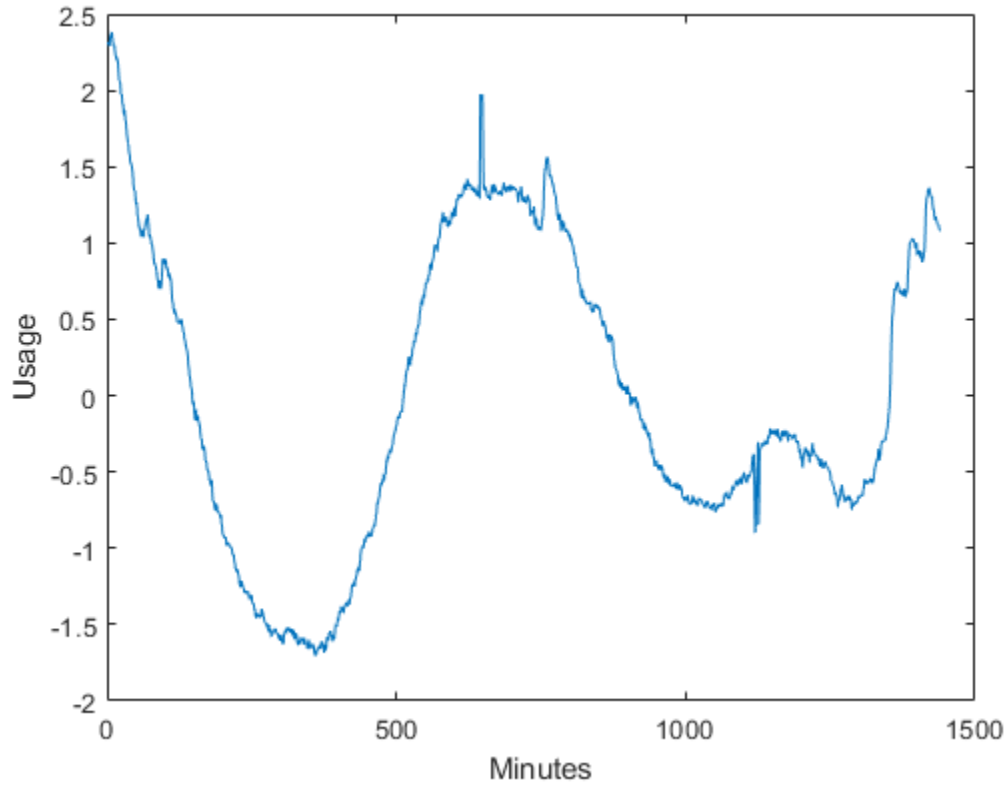
over a 24-hour period. The example demonstrates that by selecting atoms from a dictionary, matching pursuit is often able to approximate a vector more efficiently with  $M$  vectors than any single basis.

### **Matching Pursuit Using DCT, Sine, and Wavelet Dictionaries**

Load the dataset and plot the data. The dataset contains 35 days of electric consumption. Choose day 32 for further analysis. The data is centered and scaled, so the actual units of usage are not relevant.

```
load elec35_nor
x = signals(32,:);

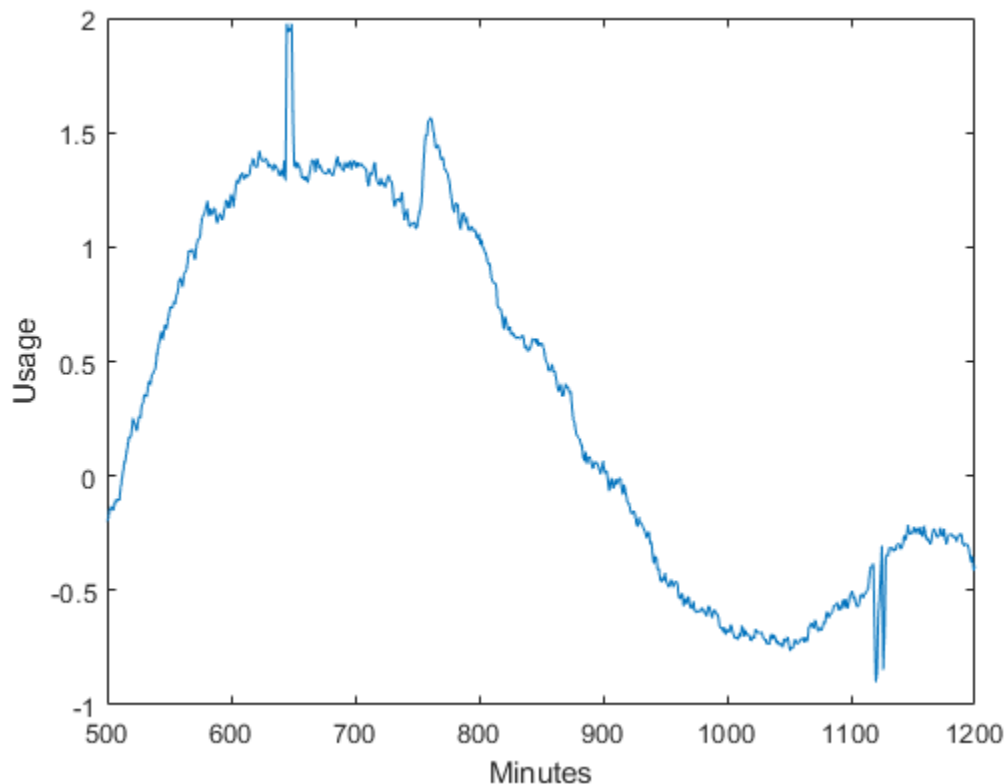
plot(x)
xlabel('Minutes')
ylabel('Usage')
```



The electricity consumption data contains smooth oscillations punctuated by abrupt increases and decreases in usage.

Zoom in on a time interval from 500 minutes to 1200 minutes.

```
xlim([500 1200])
```

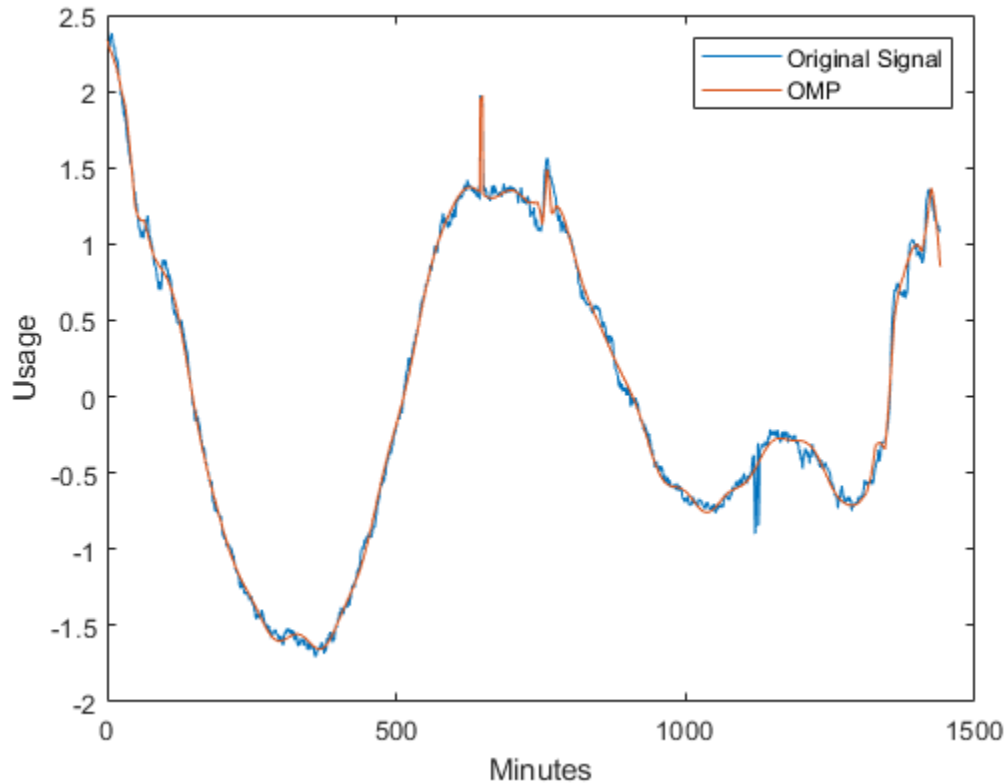


You can see the abrupt changes in the slowly-varying signal at approximately 650, 760, and 1120 minutes. In many real-world signals like these data, the interesting and important information is contained in the transients. It is important to model these transient phenomena.

Construct a signal approximation using 35 vectors chosen from a dictionary with orthogonal matching pursuit (OMP). The dictionary consists of the Daubechies extremal phase wavelet and scaling vectors at level 2, the discrete cosine transform (DCT) basis, a sine basis, the Kronecker delta basis, and the Daubechies least asymmetric phase wavelet and scaling vectors with 4 vanishing moments at levels 1 and 4. Then, use OMP to find the best 35-term greedy approximation of the electric consumption data. Plot the result.

```
dictionary = {{'db1',2},{'db1',3},'dct','sin','RnIdent',{'sym4',4}};
[mpdict,nbvect] = wmpdictionary(length(x),'lstcpt',dictionary);
[y,~,~,iopt] = wmpalg('OMP',x,mpdict);

plot(x)
hold on
plot(y)
hold off
xlabel('Minutes')
ylabel('Usage')
legend('Original Signal','OMP')
```



You can see that with 35 coefficients, orthogonal matching pursuit approximates both the smoothly-oscillating part of the signal and the abrupt changes in electricity usage.

Determine how many vectors the OMP algorithm selected from each of the subdictionaries.

```
basez = cumsum(nbvect);
k = 1;
for nn = 1:length(basez)
 if (nn == 1)
 basezind{nn} = 1:basez(nn);
 else
 basezind{nn} = basez(nn-1)+1:basez(nn);
 end
end
dictvectors = cellfun(@(x)intersect(iopt,x),basezind, ...
 'UniformOutput',false);
```

The majority (60%) of the vectors come from the DCT and sine basis. Given the overall slowly-varying nature of the electricity consumption data, this is expected behavior. The additional 14 vectors from the wavelet subdictionaries captures the abrupt signal changes. The number of vectors of each type is:

- 3 Daubechies wavelet (db4) level 2 vectors
- 16 Discrete cosine transform vectors
- 5 Sine vectors
- 2 Daubechies least-asymmetric wavelet (sym4) level 1 vectors
- 9 Daubechies least-asymmetric wavelet (sym4) level 4 vectors

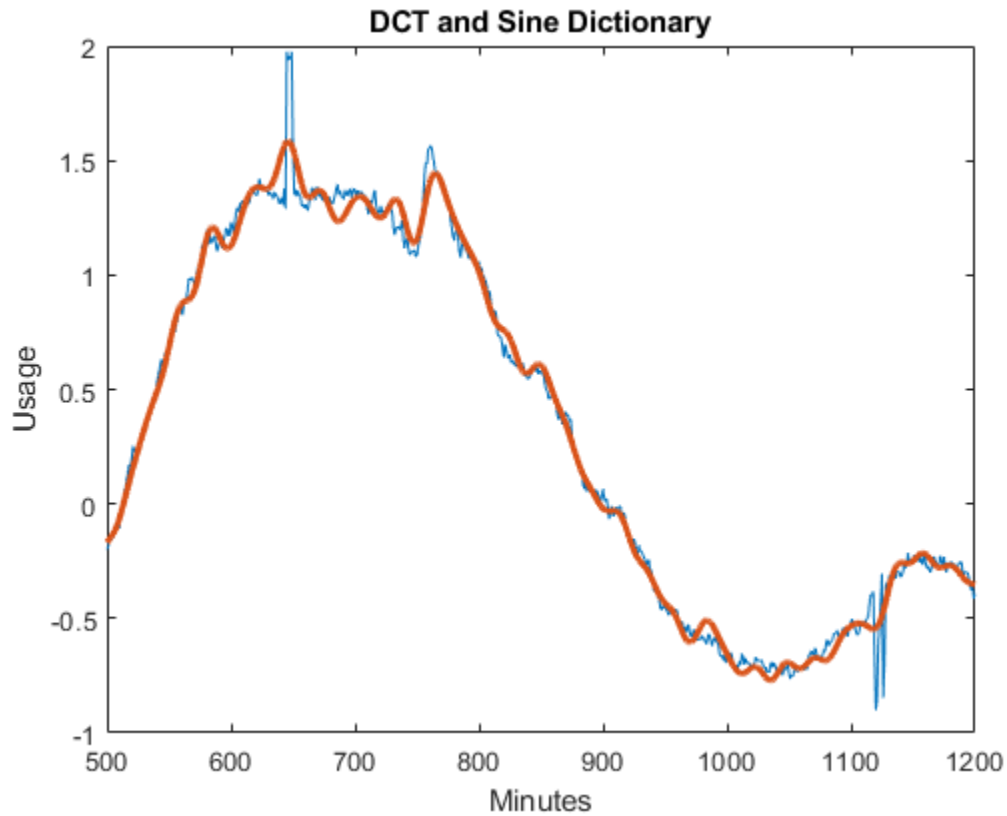
### Matching Pursuit Using DCT and Sine Dictionary vs. Full Dictionary

Repeat the OMP with only the DCT and sine subdictionaries. Set the OMP to select the 35 best vectors from the DCT-sine dictionary. Construct the dictionary and perform the OMP. Compare the OMP with the DCT-sine dictionary to the OMP with the additional wavelet subdictionaries. Notice that adding the wavelet subdictionaries shows the abrupt changes in electricity usage more accurately. The advantage of including the wavelet bases is especially clear especially in approximating the upward and downward spikes in usage at approximately 650 and 1120 minutes.

```
dictionary2 = {'dct','sin'};
[mpdict2,nbvect2] = wmpdictionary(length(x),'lstcpt',dictionary2);
y2 = wmpalg('OMP',x,mpdict2,'itermax',35);

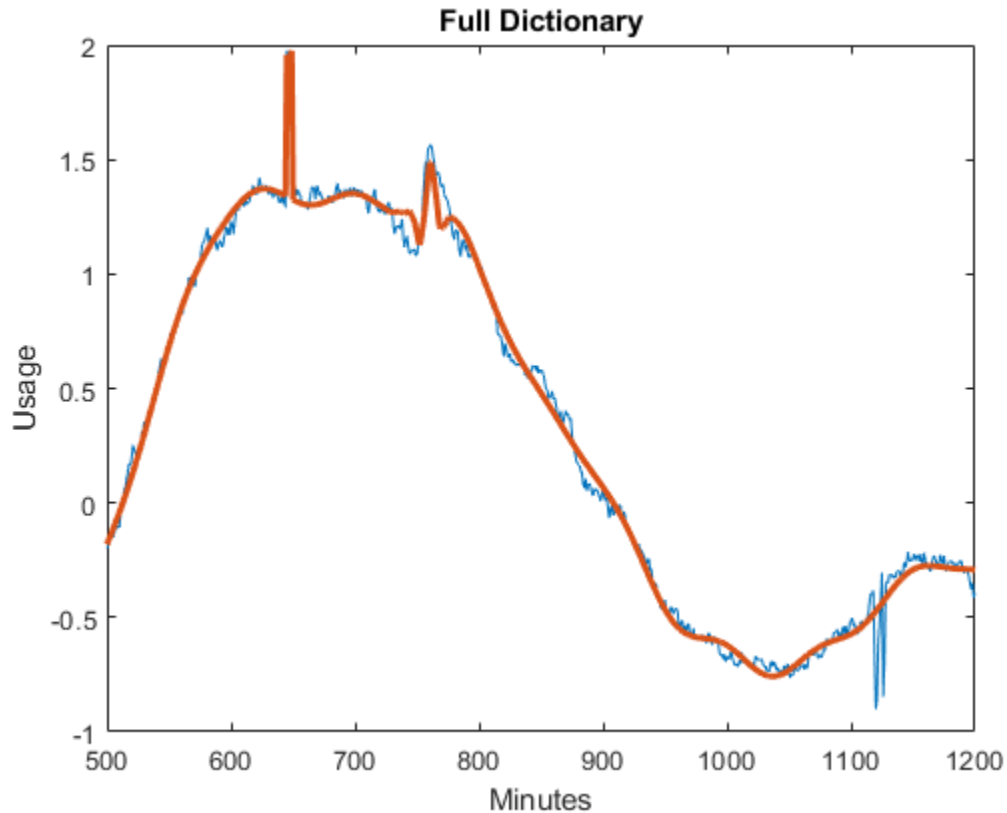
plot(x)
hold on
```

```
plot(y2,'linewidth',2)
hold off
title('DCT and Sine Dictionary')
xlabel('Minutes')
ylabel('Usage')
xlim([500 1200])
```



```
figure
plot(x)
hold on
plot(y,'linewidth',2)
hold off
title('Full Dictionary')
```

```
xlabel('Minutes')
ylabel('Usage')
xlim([500 1200])
```



Obtain the best 35-term nonlinear approximation of the signal in the discrete Fourier basis. Obtain the DFT of the data, sort the DFT coefficients, and select the 35 largest coefficients. The DFT of a real-valued signal is conjugate symmetric, so only consider frequencies from 0 (DC) to the Nyquist (1/2 cycles/minute).

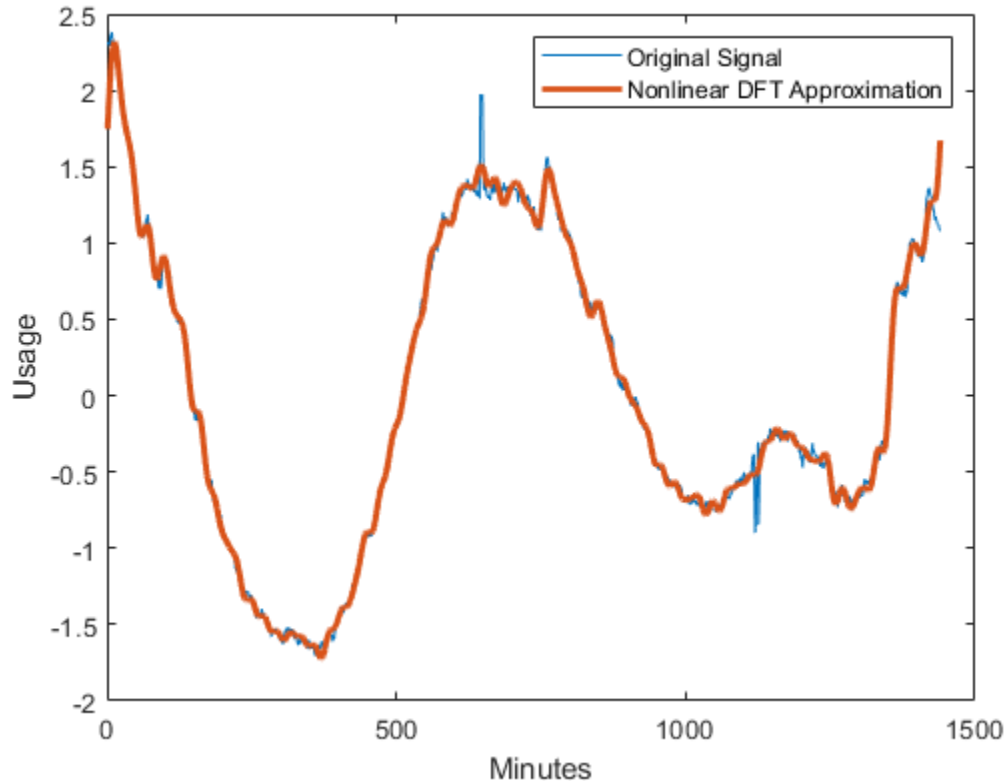
```
xdft = fft(x);
[~,I] = sort(xdft(1:length(x)/2+1), 'descend');
ind = I(1:35);
```

Examine the vector `ind`. None of the indices correspond to 0 or the Nyquist. Add the corresponding complex conjugate to obtain the nonlinear approximation in the DFT basis. Plot the approximation and the original signal.

```
indconj = length(xdft)-ind+2;
ind = [ind indconj];
xdftapp = zeros(size(xdft));
xdftapp(ind) = xdft(ind);
xrec = ifft(xdftapp);

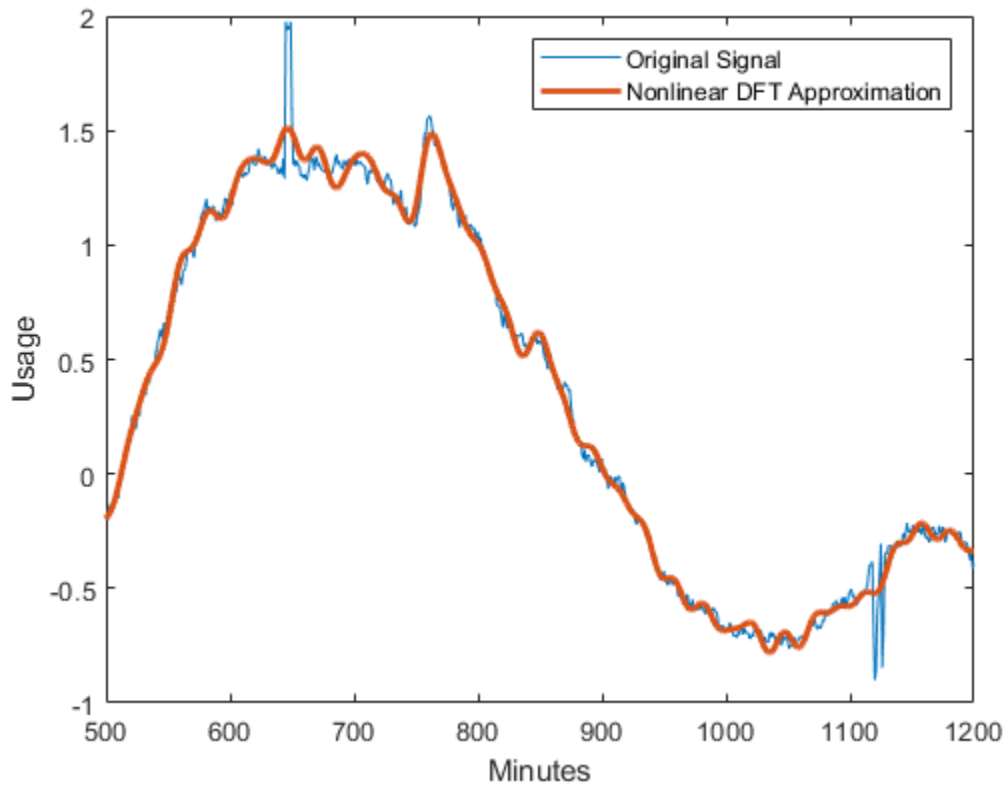
plot(x)
hold on
plot(xrec,'LineWidth',2)
hold off
xlabel('Minutes')
ylabel('Usage')
legend('Original Signal','Nonlinear DFT Approximation')
```





Similar to the DCT-sine dictionary, the nonlinear DFT approximation performs well at matching the smooth oscillations in electricity consumption data. However, the nonlinear DFT approximation does not approximate the abrupt changes accurately. Zoom in on the interval of the data containing the abrupt changes in consumption.

```
plot(x)
hold on
plot(xrec, 'LineWidth', 2)
hold off
xlabel('Minutes')
ylabel('Usage')
legend('Original Signal', 'Nonlinear DFT Approximation')
xlim([500 1200])
```

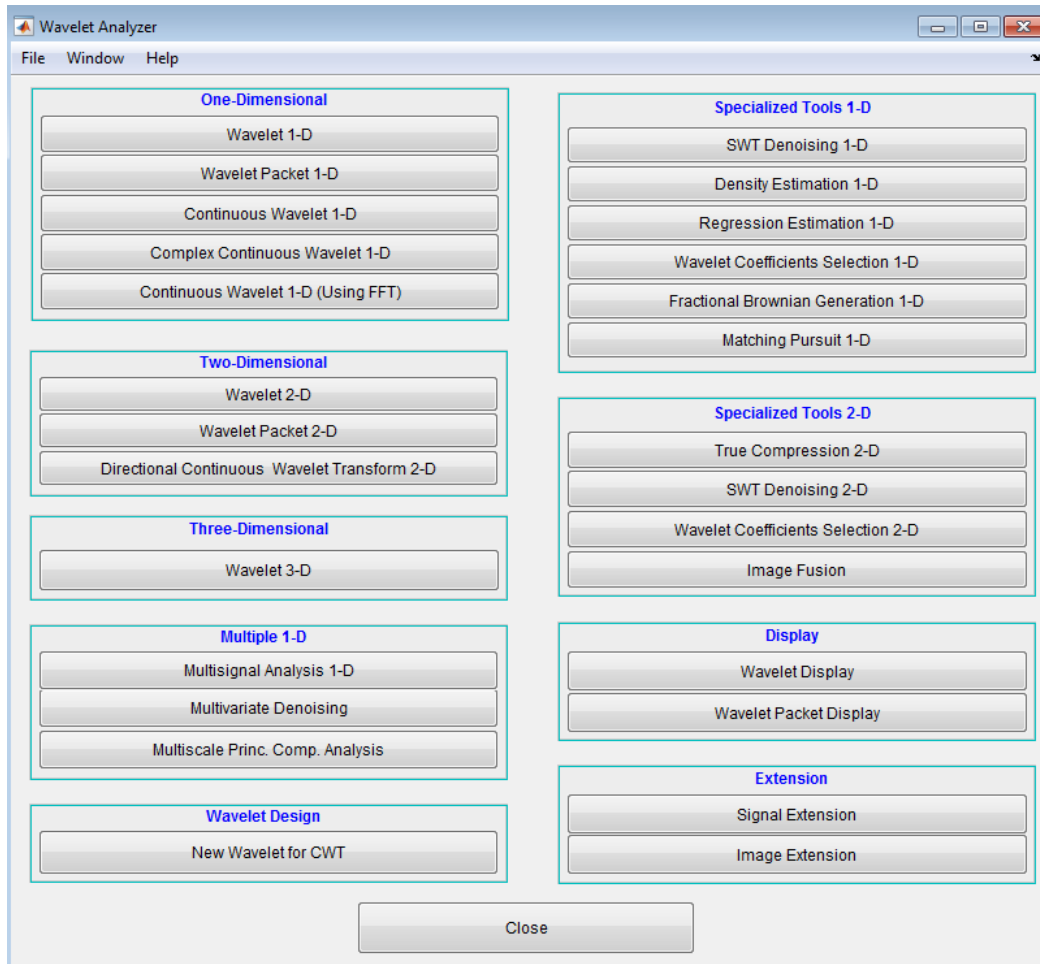


## Matching Pursuit Using Wavelet Analyzer App

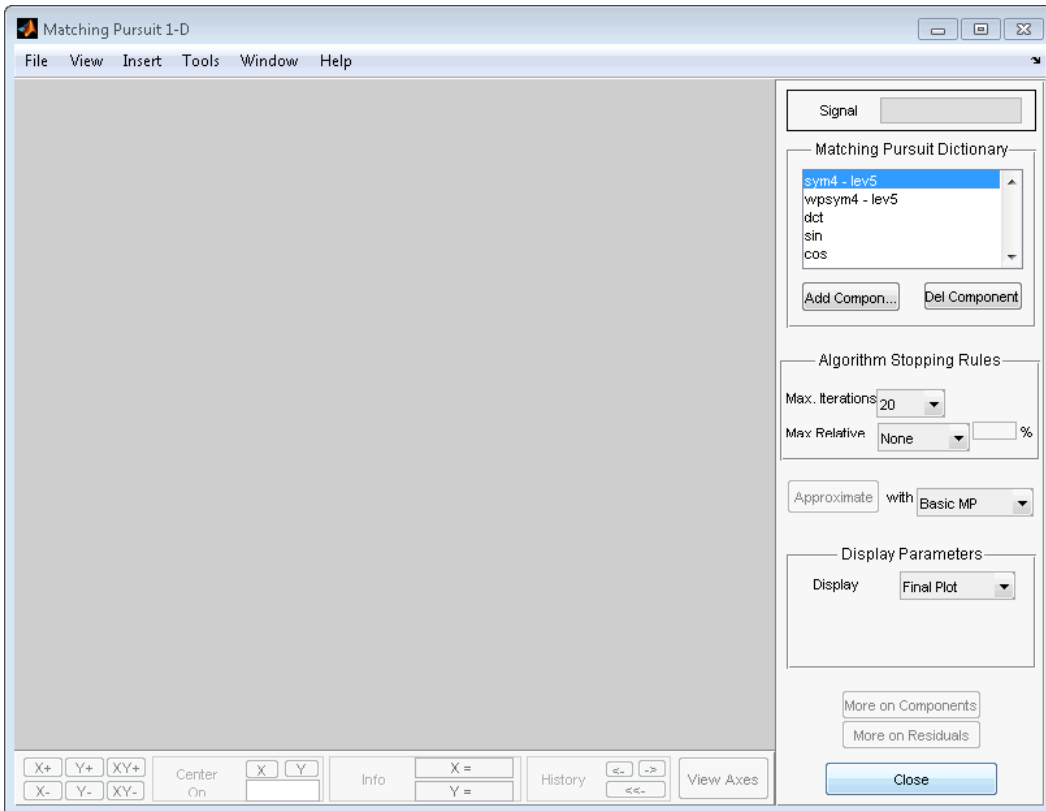
| In this section...                                                          |
|-----------------------------------------------------------------------------|
| “Matching Pursuit 1-D Interactive Tool” on page 7-23                        |
| “Interactive Matching Pursuit of Electricity Consumption Data” on page 7-39 |

### Matching Pursuit 1-D Interactive Tool

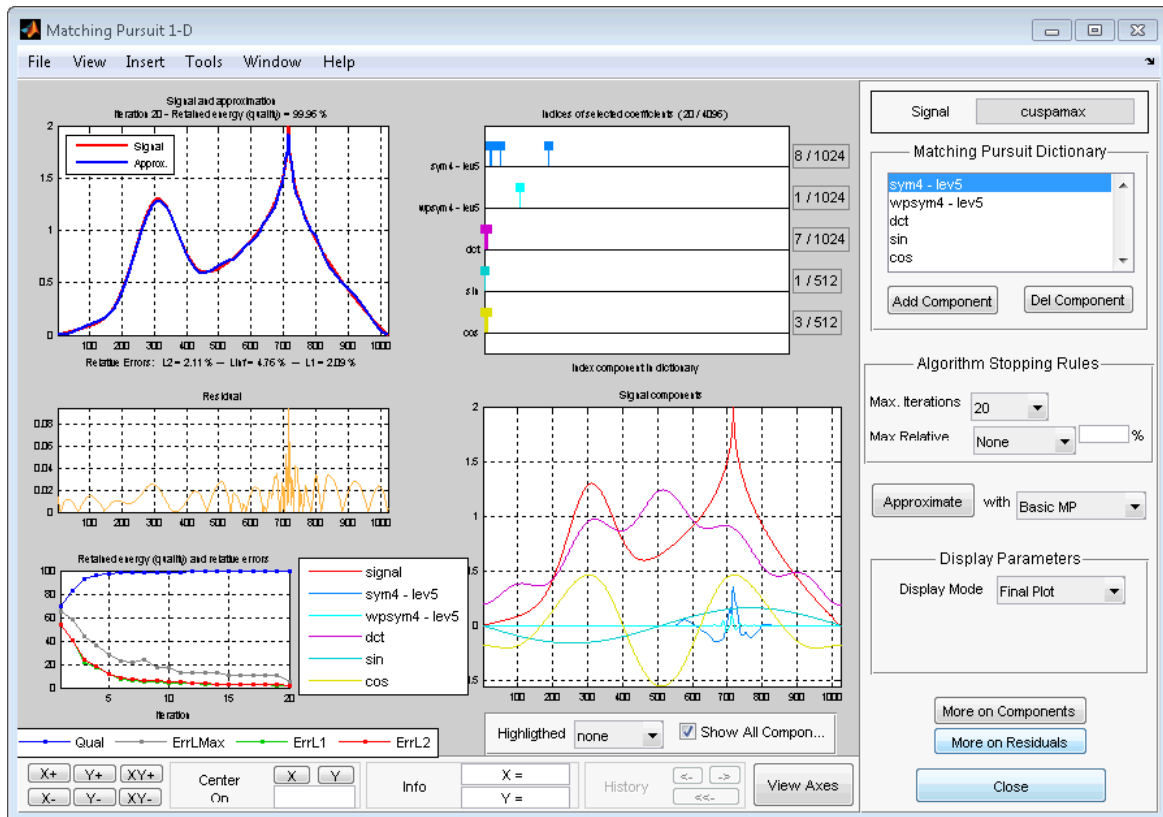
You can perform basic, orthogonal, and weak orthogonal matching pursuit using the Wavelet Analyzer app. To access the **Matching Pursuit 1-D**, enter `waveletAnalyzer` at the MATLAB command prompt.



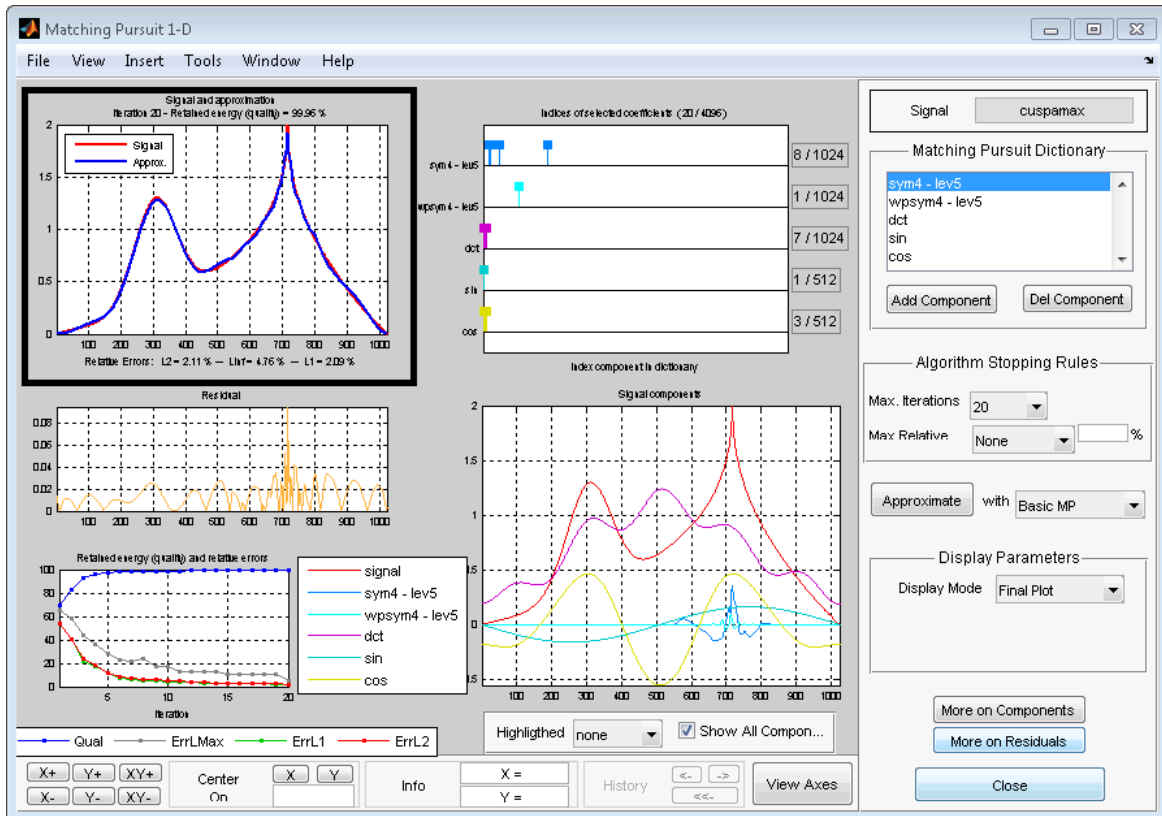
Click **Matching Pursuit 1-D**.



To demonstrate the **Matching Pursuit 1-D** tool, select **File** → **Example** → **Cusp signal**.



In the upper left corner, you see the plot of the signal with the matching pursuit approximation superimposed.



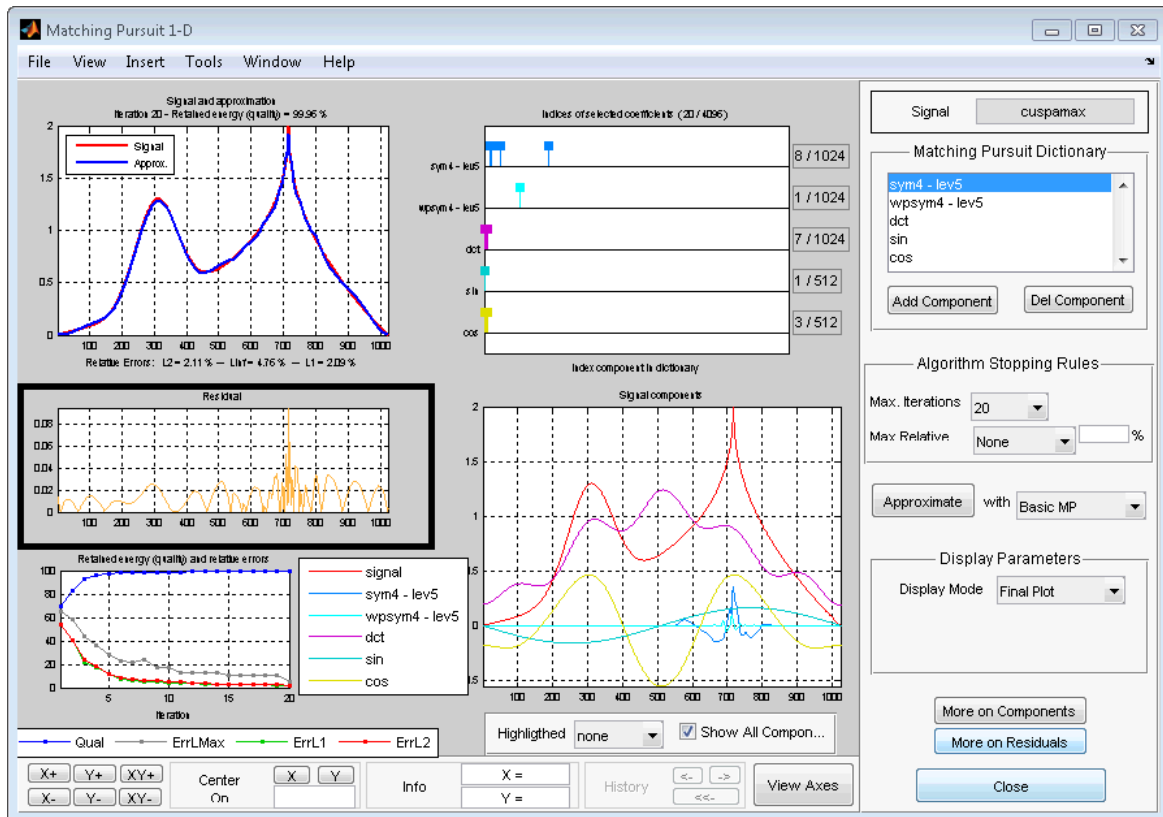
Underneath the plot, you see the relative errors using the L1, L2, and L-infinity norms.

The maximum relative error in a given norm is

$$100 \frac{\|R\|}{\|Y\|},$$

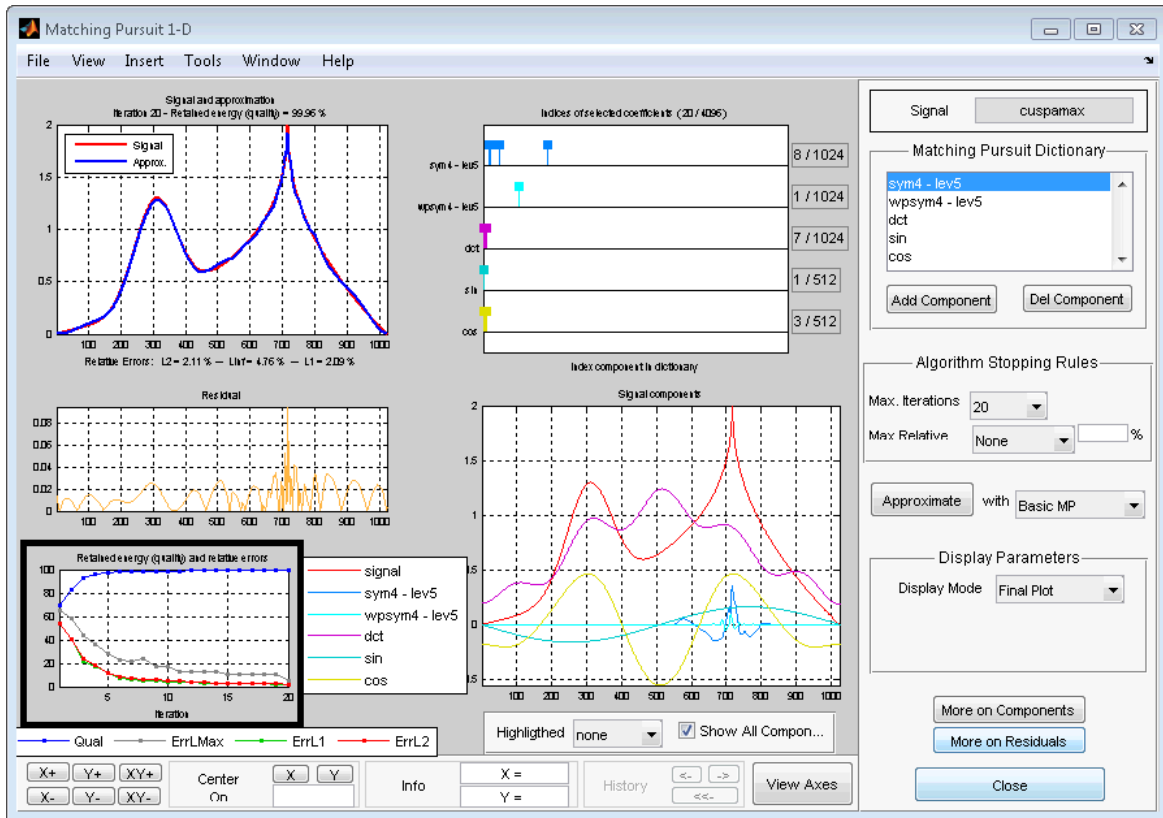
where  $\| \cdot \|$  denotes the specified norm,  $R$  is the residual vector at each iteration in the matching pursuit algorithm, and  $Y$  is the signal.

In the middle panel on the left is the plot of the final residual vector after the matching pursuit algorithm terminates.

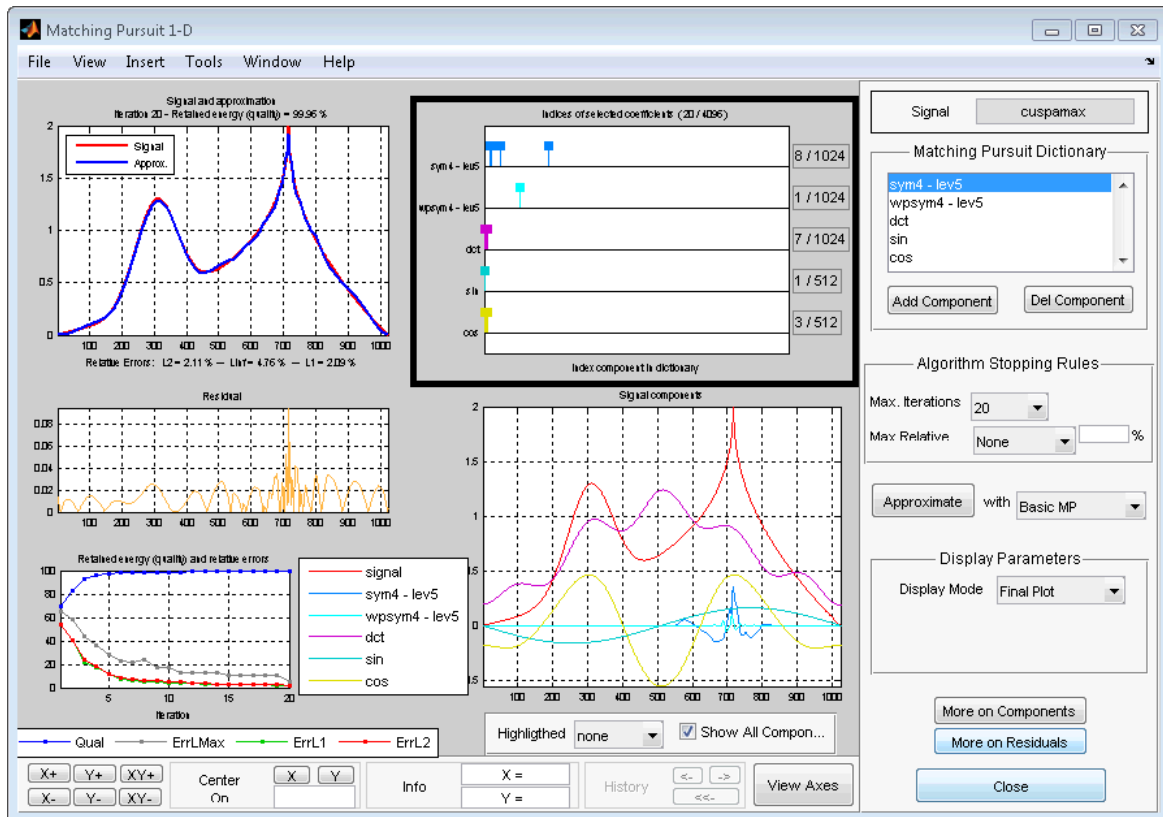


The bottom left panel displays the percentage of retained signal energy (L2 norm) and the relative error percentages for the L1, L2, and L-infinity norms over the algorithm iterations.





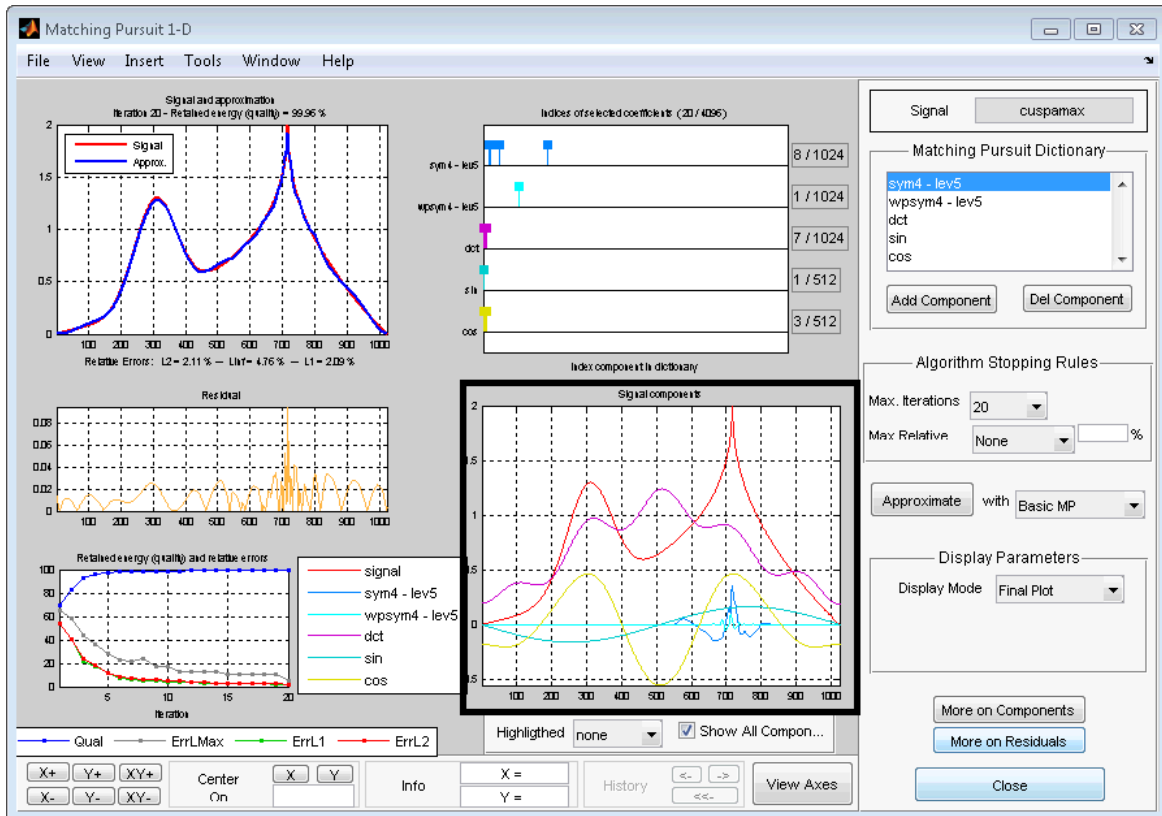
In the top middle panel of the **Matching Pursuit 1-D** tool, you see the indices of the selected coefficients from the subdictionaries.



The left vertical axis shows the name of the subdictionary. The right vertical axis gives the ratio of selected vectors to the total number of vectors in the subdictionary. The location of the vertical bars along the horizontal axis gives the relative positions of the selected vectors in the subdictionaries.

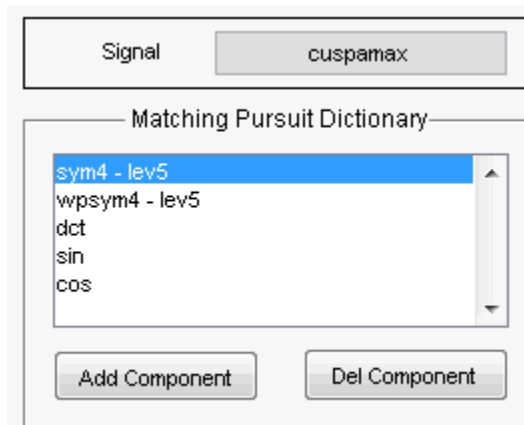
More detailed information on selected components is available by clicking **More on Components** in the bottom right panel.

The bottom middle panel displays the superposition of selected vectors from the subdictionaries.



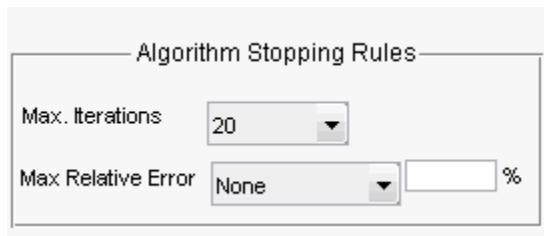
This plot enables you to assess the relative contribution of the subdictionary to the signal approximation. In this example, you can see that the cosine and DCT subdictionary contribute significantly to the approximation of the slowly-varying portions of the signal. The Daubechies least asymmetric wavelet with 4 vanishing moments (sym4) enables the matching pursuit to sparsely represent the cusp around index 700.

In the top right panel of the **Matching Pursuit 1-D** tool, you see the dictionary used in the analysis.



You have the ability to add or delete subdictionaries with **Add Component** and **Del Component**.

The next panel contains the algorithm stopping rules.



- **Max. Iterations** — This controls the number of iterations of the greedy matching pursuit algorithm. The value is equal to the number of expansion coefficients (vectors) used in the approximation. The utility of matching pursuit is that you can approximate many real-world signals efficiently with far fewer vectors than needed to span the signal space.
- **Max Relative Error** — Specifies the stopping criterion based on the maximum relative error. Choose one of None, L2 norm, L1 norm, or Linf norm.

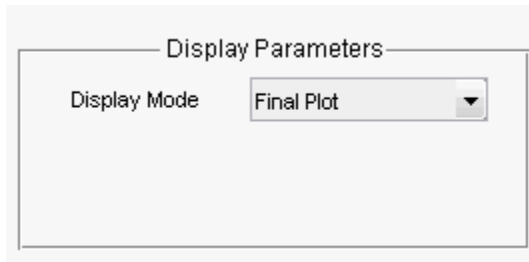
The maximum relative error in a given norm is

$$100 \frac{\|R\|}{\|Y\|},$$

where  $\| \cdot \|$  denotes the specified norm,  $R$  is the residual vector at each iteration in the matching pursuit algorithm, and  $Y$  is the signal.

In the next panel you select the algorithm used in the matching pursuit. Choose one of **Basic MP** for basic matching pursuit, **Orthogonal MP** for orthogonal matching pursuit, and **Weak MP** for weak orthogonal matching pursuit. See “Matching Pursuit Algorithms” on page 7-2 for a brief description of these algorithms.

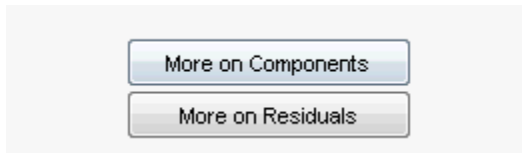
In the **Display Parameters** panel, you can control how the progress of the matching pursuit is displayed.



Select one of

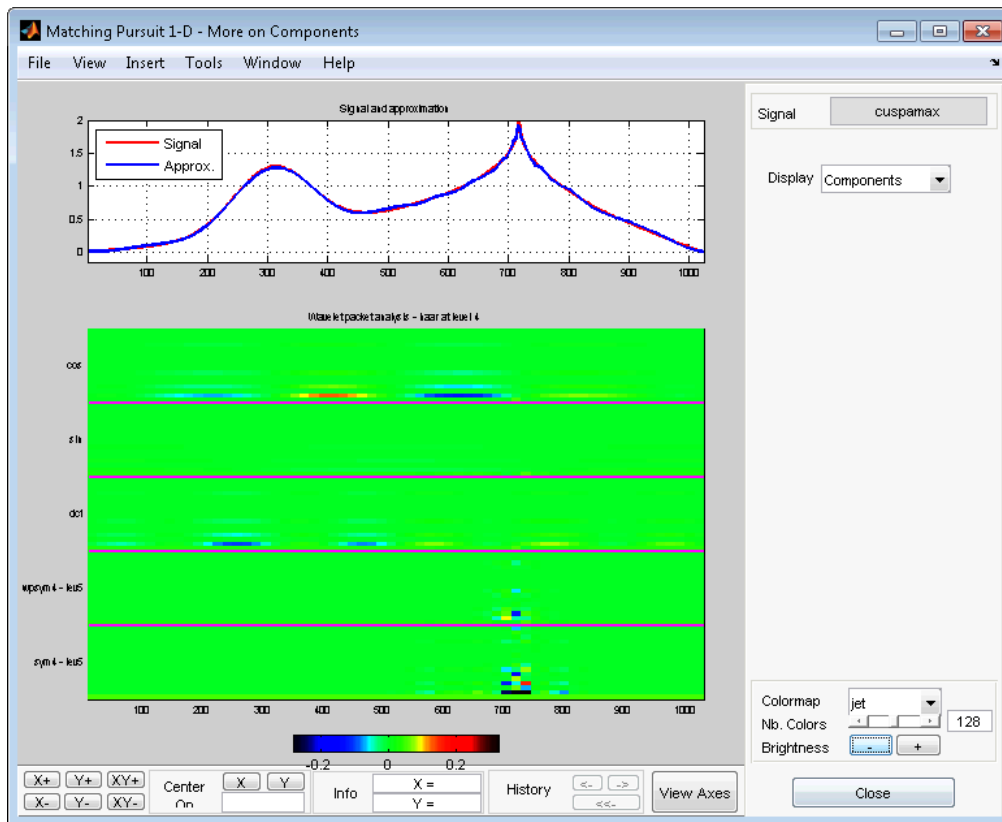
- **Final Plot** — Plots the result of matching pursuit only after the algorithm terminates.
- **Stepwise** — Updates the result every  $N$  iterations where  $N$  is a positive integer. If you select **Stepwise**, the **Display every iterations** item becomes visible. Select the number of iterations from the drop down menu. You are prompted to step through the algorithm with the **Next** or **Final Plot**.
- **Movie** — Updates the result every  $N$  iterations where  $N$  is a positive integer in a continuous manner. If you select **Movie**, the **Display every iterations** item becomes visible. Select the number of iterations from the drop down menu. Click **Continue** to step through the algorithm as a movie, which continues until the algorithm terminates. Click **Pause** to pause the algorithm, or **Final Plot** to update only at the termination of the algorithm.

After you obtain a matching pursuit of a signal, use



to obtain detailed interactive plots and information on the selected dictionary atoms and the final residual vector.

Click **More on Components**.



From the above figure, you can see that while the DCT and cosine subdictionary contribute energy across the extent of the signal, the wavelet and wavelet packet contributions are localized at the cusp around sample 700. This result is expected

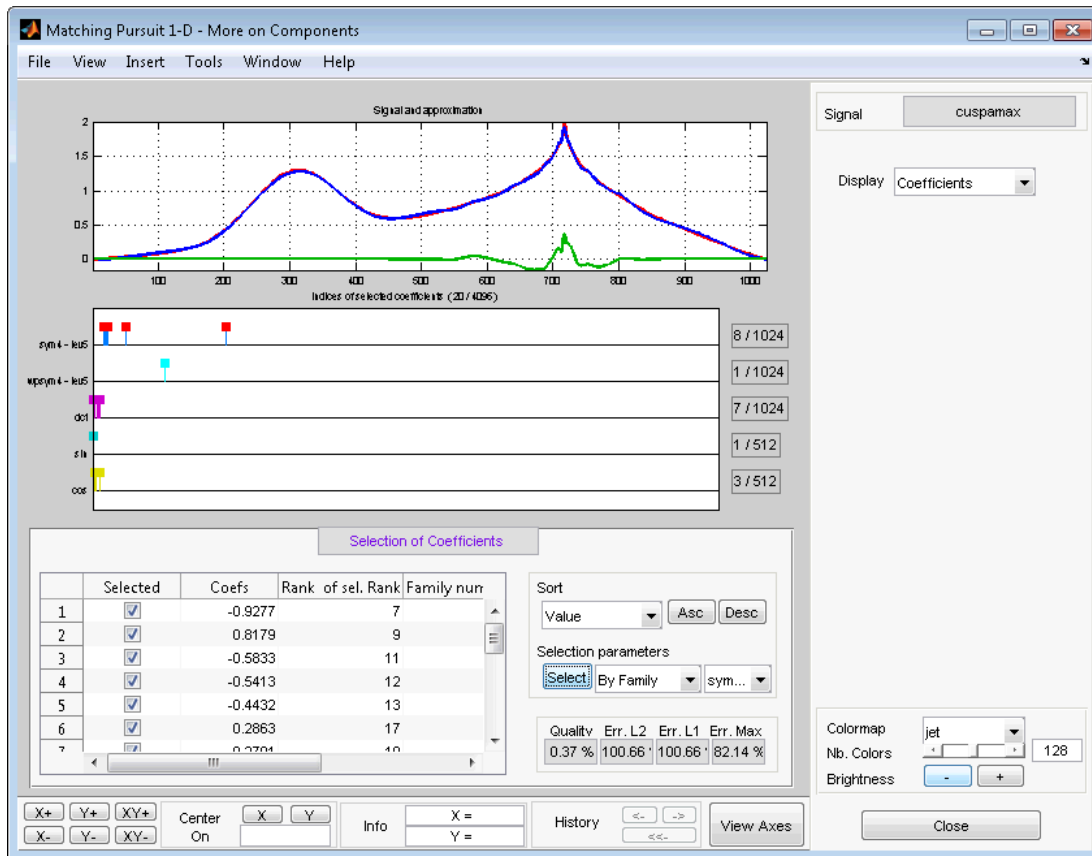
because wavelets and wavelet packets excel at sparsely representing abrupt changes in a signal or image.

Change the **Display** to the **Coefficients** view.

The **Selection of Coefficients** panel enables you to selectively sort and display contributions to the signal approximation by the various subdictionaries.

|   | Selected                 | Coefs   | Rank of sel. | Rank | Family number | Ndx in Dictior |
|---|--------------------------|---------|--------------|------|---------------|----------------|
| 1 | <input type="checkbox"/> | 21.9376 | 1            | 3    | 1             | 2              |
| 2 | <input type="checkbox"/> | -9.5213 | 2            | 3    | 2             | 2              |
| 3 | <input type="checkbox"/> | -8.6156 | 3            | 5    | 3             | 3              |
| 4 | <input type="checkbox"/> | 4.2498  | 4            | 5    | 4             | 3              |
| 5 | <input type="checkbox"/> | -3.6537 | 5            | 4    | 5             | 3              |
| 6 | <input type="checkbox"/> | -2.3496 | 6            | 3    | 6             | 2              |
| 7 | <input type="checkbox"/> | -0.9277 | 7            | 1    | 7             | 1              |

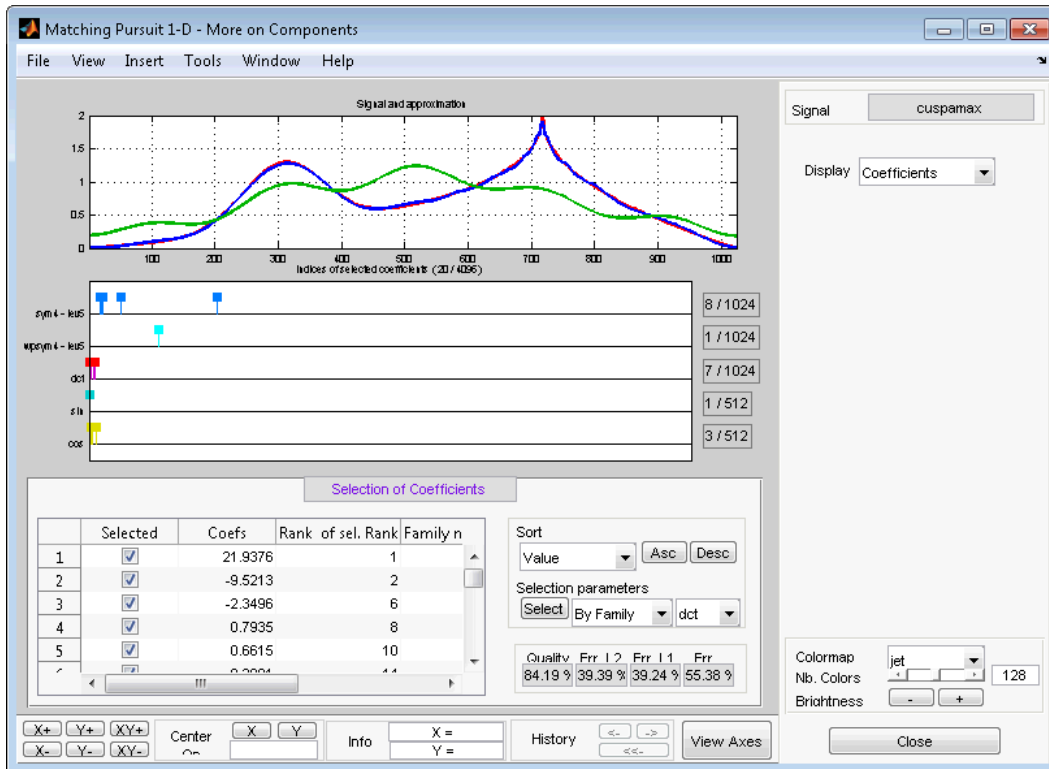
Under **Selection parameters**, choose **By Family** and **sym4 – lev5**. Click **Select**



From the preceding operation, you see that the wavelet packet contributes to the approximation of the cusp, but does not contribute significantly to the global approximation.

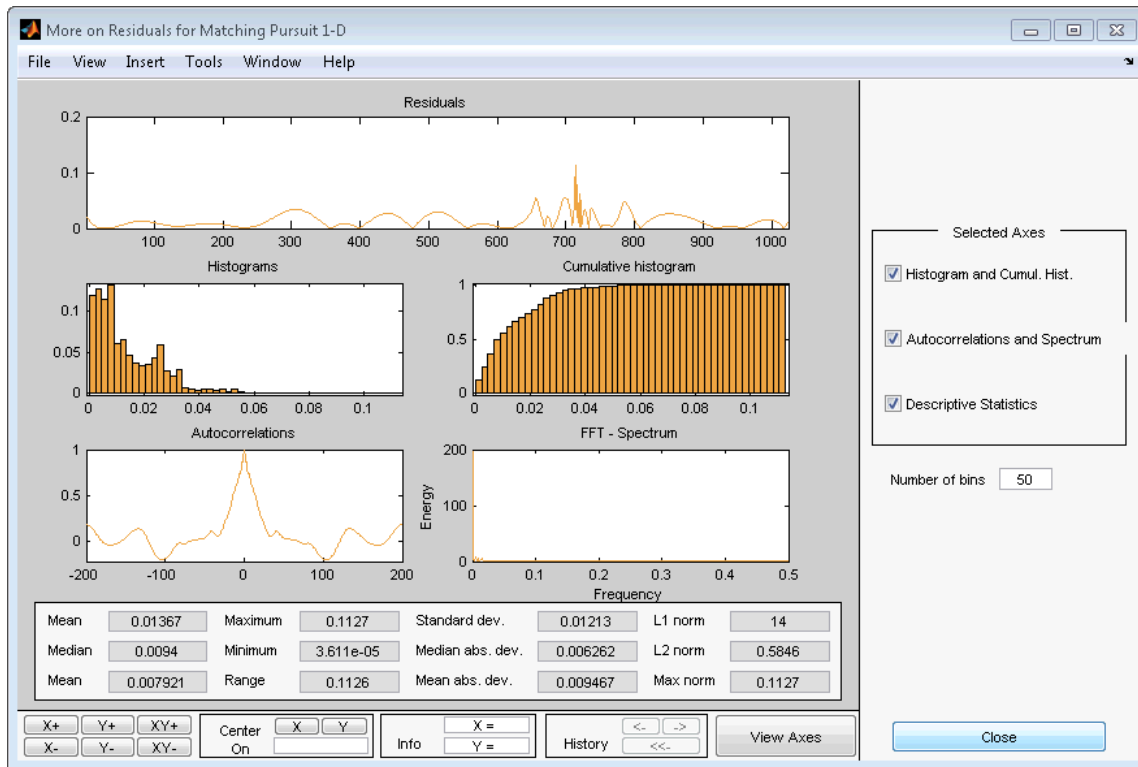
Choose **dct** and click **Select**.



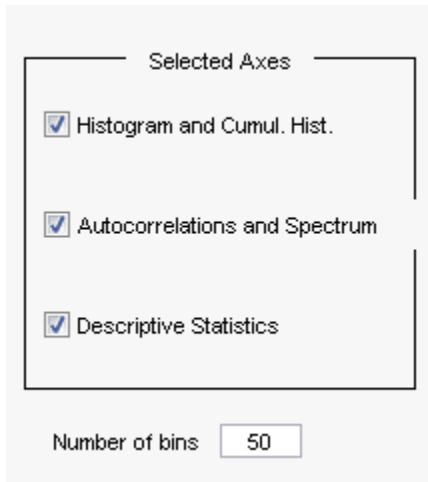


The DCT basis contributes significantly to the global approximation of the signal but the smooth DCT basis vectors are not able to sparsely represent the cusp.

Selecting **More on Residuals** on the **Matching Pursuit 1-D** tool allows you to examine the residual vector, a histogram of the residuals, a cumulative histogram, the estimated autocorrelation sequence, and the magnitude-squared discrete Fourier transform.



You can control which plots are displayed and the appearance of the histogram by the options in the right panel.



## Interactive Matching Pursuit of Electricity Consumption Data

This example shows how to perform an interactive matching pursuit of electricity consumption data collected over a 24-hour period.

Load the electricity consumption signals in the workspace. Select the data for the 32nd day for further matching pursuit.

```
load elec35_nor;
x = signals(32,:);
```

To start the app, enter `waveletAnalyzer` at the MATLAB command prompt.

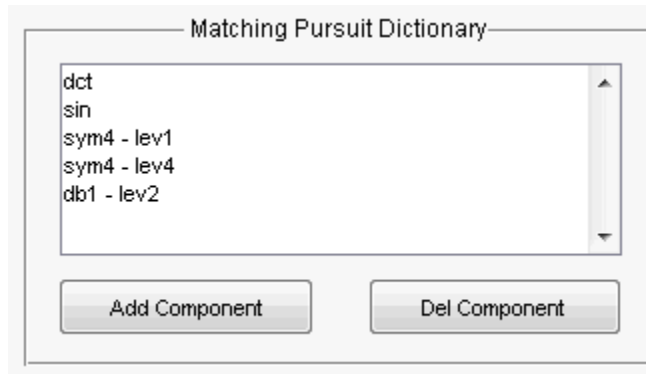
Click the **Matching Pursuit 1-D** tool.



Select **File** → **Import Signal from Workspace**

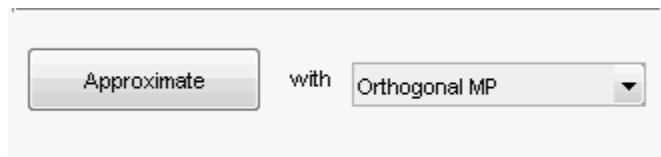
Load x.

Construct the following matching pursuit dictionary.

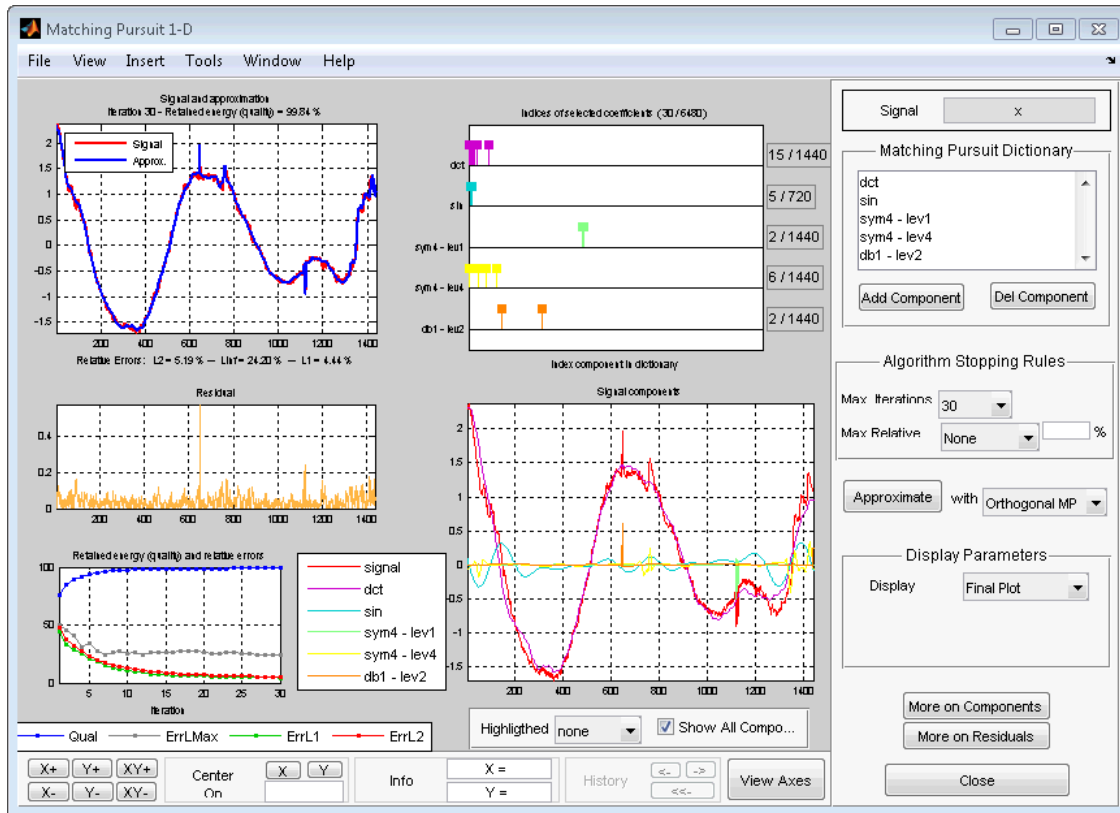


In the **Algorithm Stopping Rules** panel, set **Max. Iterations** to 30.

Select **Orthogonal MP** to use orthogonal matching pursuit.



Click **Approximate**.



# Generating MATLAB Code from Wavelet Toolbox Wavelet Analyzer App

---

You can denoise or compress a signal or image in the **Wavelet Analyzer** app and export the MATLAB code to implement that operation at the command line. This approach allows you to set denoising thresholds or compression ratios aided by visualization tools and save the commands to reproduce those operations at the command line.

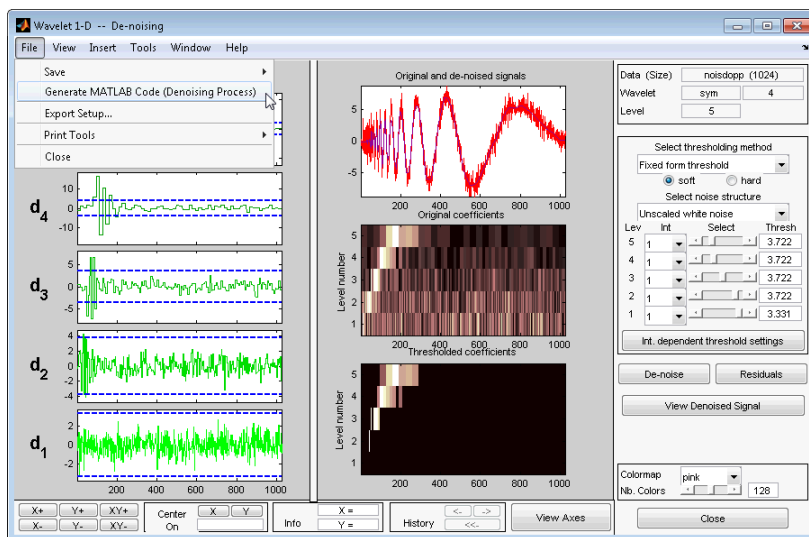
- “Generate MATLAB Code for 1-D Decimated Wavelet Denoising and Compression” on page 8-2
- “Generate MATLAB Code for 2-D Decimated Wavelet Denoising and Compression” on page 8-11
- “Generate MATLAB Code for 1-D Stationary Wavelet Denoising” on page 8-17
- “Generate MATLAB Code for 2-D Stationary Wavelet Denoising” on page 8-23
- “Generate MATLAB Code for 1-D Wavelet Packet Denoising and Compression” on page 8-27
- “Generate MATLAB Code for 2-D Wavelet Packet Denoising and Compression” on page 8-31
- “Generate Code to Denoise a Signal” on page 8-35
- “Code Generation Support, Usage Notes, and Limitations” on page 8-37

## Generate MATLAB Code for 1-D Decimated Wavelet Denoising and Compression

### Wavelet 1-D Denoising

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

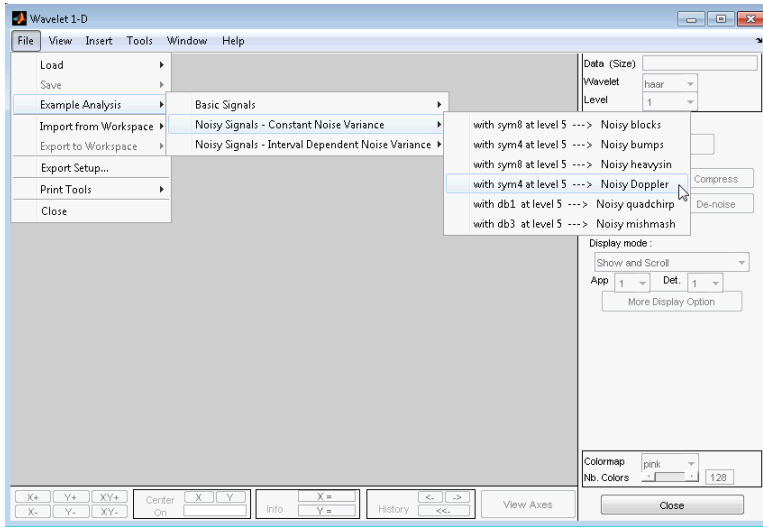
The generated MATLAB code does not include the calculation of the thresholds using `thselect` or `wbmpen`.



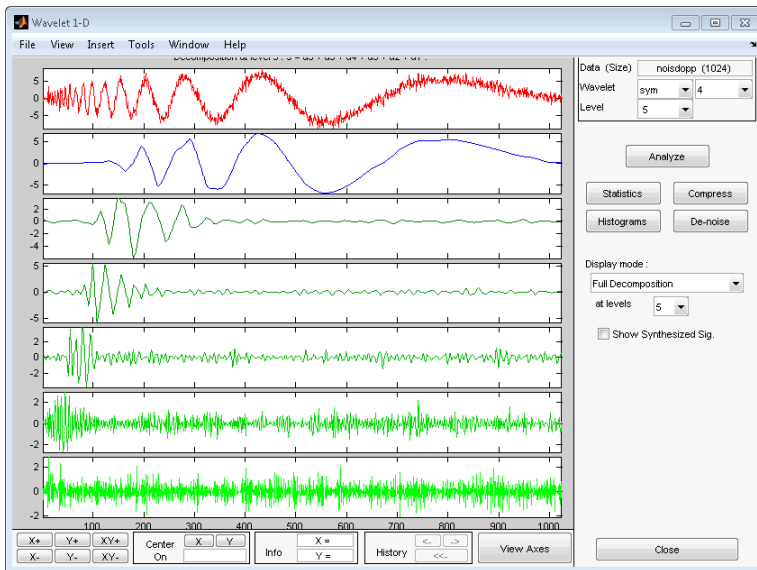
### Denoise Doppler Signal

- 1 Enter `waveletAnalyzer` at the MATLAB command prompt.
- 2 Select **Wavelet 1-D** in the **Wavelet Analyzer**.
- 3 Load the noisy Doppler example analysis. Select **File > Example Analysis > Noisy Signals - Constant Noise Variance > with sym4 at level 5 - - -> Noisy Doppler**.





After selecting the analysis, the wavelet decomposition appears.

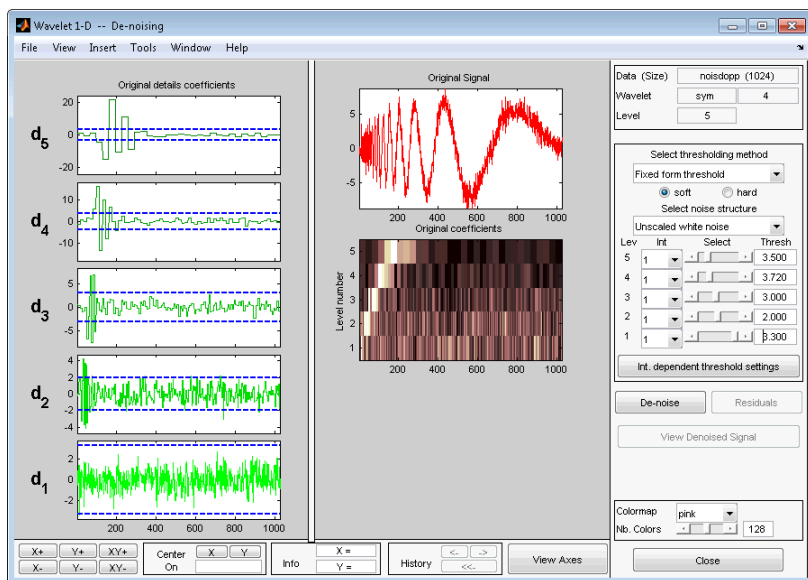


4 Click **Denoise**.

- The original details coefficients appear on the left side of the display. In order to time align decomposition levels across all scales, wavelet coefficients are replicated at each scale to account for the missing time points. Therefore, as the scale becomes coarser, the coefficients assume a staircase-like appearance.

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

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



Click **Denoise**.

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

The operation generates the following MATLAB code.

```

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

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

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

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

```

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

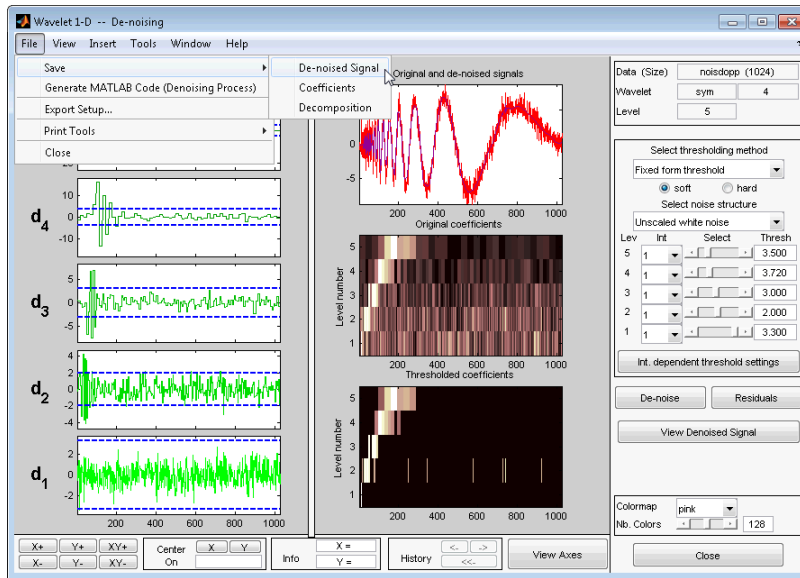
```

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

```

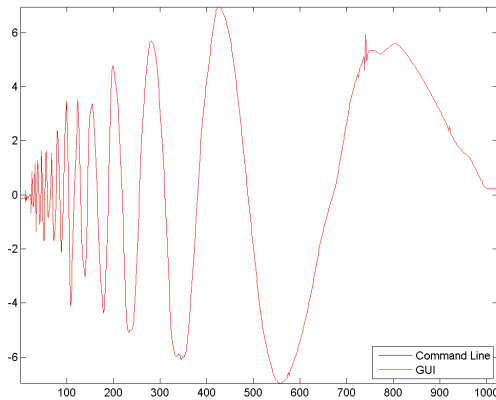
- 8 Export the denoised signal from the app by selecting **File > Save > Denoised Signal**.

## 8 Generating MATLAB Code from Wavelet Toolbox Wavelet Analyzer App



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

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



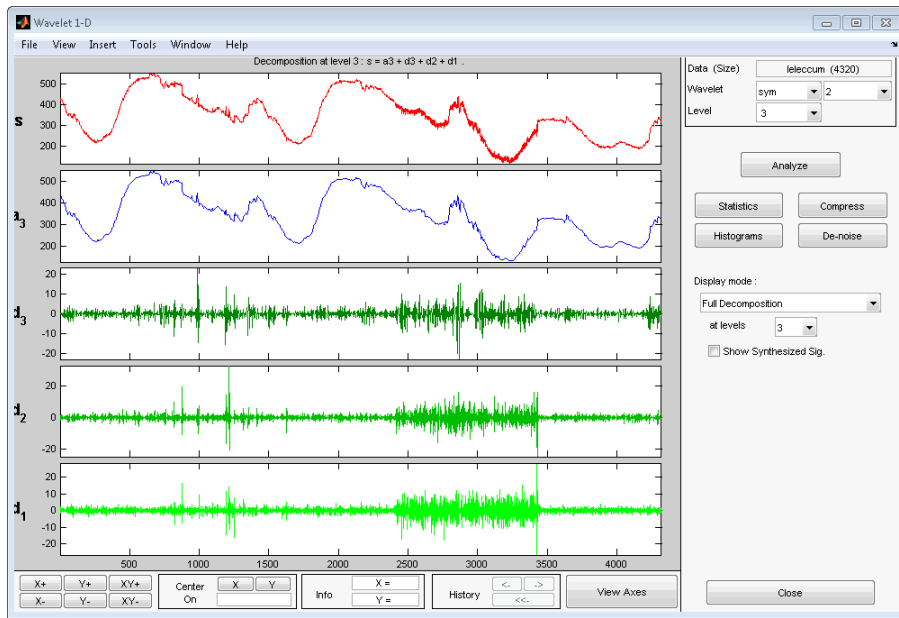
### Interval Dependent 1-D Wavelet Denoising

- 1 Enter `waveletAnalyzer` at the MATLAB command prompt.
- 2 Select **Wavelet 1-D**.
- 3 At the MATLAB command prompt, type

```
load leleccum;
```

In the **Wavelet 1-D** tool, select **File > Import from Workspace > Import Signal**. When the **Import from Workspace** dialog box appears, select the `leleccum` variable. Click **OK** to import the data.

- 4 Select the `sym4` wavelet, and set `Level` equal to 3. Click **Analyze**.



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

- 1 Click **Denoise**.
- 2 Under **Select thresholding method**, select **Rigorous SURE**.
- 3 Select **Int. dependent threshold settings**.
- 4 In the **Interval Dependent Threshold Settings for Wavelet 1-D** tool, choose **Generate Default Intervals**. Three intervals are created. Click **Propagate** to propagate the intervals to all levels.
- 5 Click **Close**, and answer **Yes** to Update Thresholds?.
- 6 Select **Denoise**.
- 7 Generate the MATLAB code by selecting **File > Generate Matlab Code (Denoising Process)**.

The operation generates the following MATLAB code.

```
function sigDEN = func_denoise_dwld(SIG)
% FUNC_DENOISE_DWLD Saved Denoising Process.
```

```

% SIG: vector of data
% -----
% sigDEN: vector of denoised data

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

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

% Denoise using CMDDENNOISE.
%-----
sigDEN = cmdddenoise(SIG,wname,level,sorgh,NaN,thrSettings);

```

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

```
function sigDEN = func_IDdenoise_dwld(SIG)
```

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

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

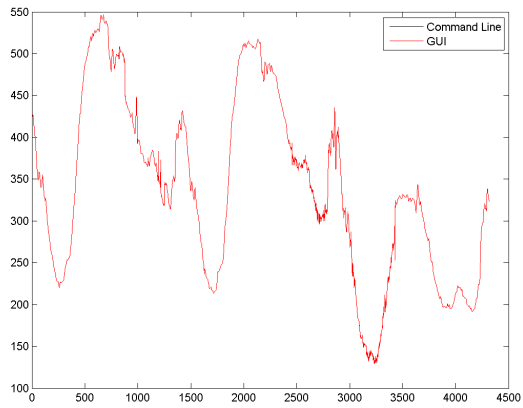
Execute the following code.

```

load leleccum;
load denoisedleleccum;
sigDEN = func_IDdenoise_dwld(leleccum);

```

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





# Generate MATLAB Code for 2-D Decimated Wavelet Denoising and Compression

## In this section...

“2-D Decimated Discrete Wavelet Transform Denoising” on page 8-11

“2-D Decimated Discrete Wavelet Transform Compression” on page 8-14

## 2-D Decimated Discrete Wavelet Transform Denoising

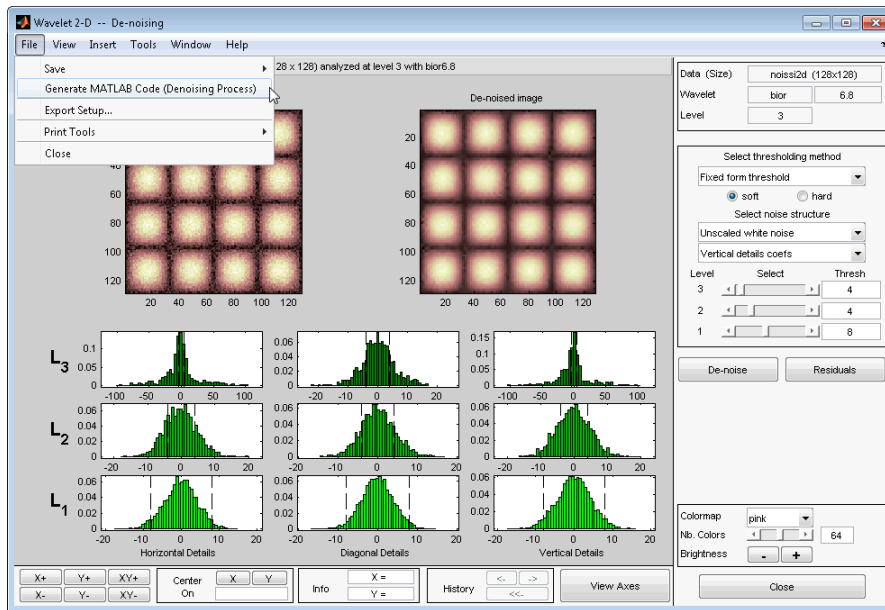
You can generate MATLAB code to reproduce app-based 2-D decimated wavelet denoising at the command line. You must perform this operation in the **Wavelet 2-D - - Denoising** tool. You must first denoise your image before you can enable the **File > Generate Matlab Code (Denoising Process)** operation.

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

Enter these thresholds for the horizontal, diagonal, and vertical coefficients.

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

## 8 Generating MATLAB Code from Wavelet Toolbox Wavelet Analyzer App



The operation generates the following MATLAB code.

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

% Analysis parameters.
%-----
wname = 'bior6.8';
level = 3;

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

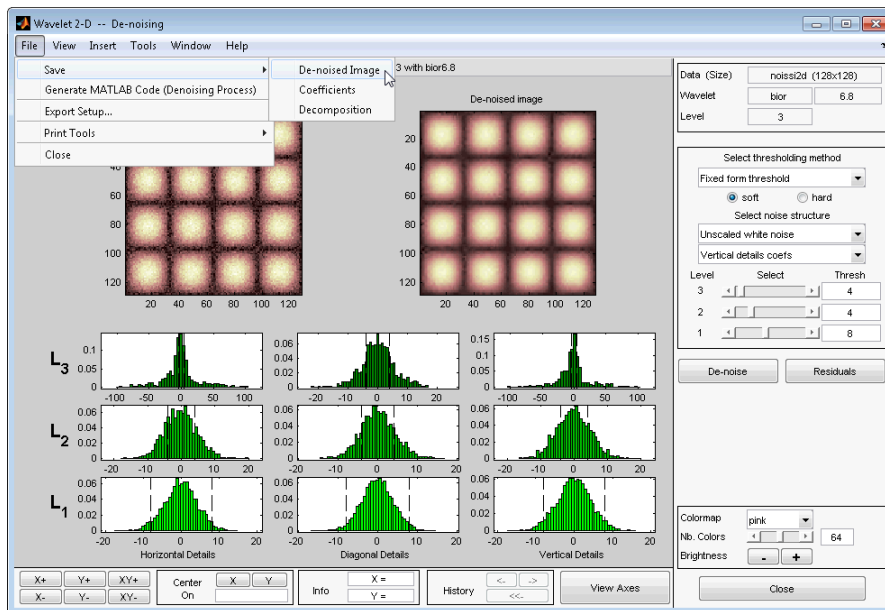
```
% Denoise using CMDENOISE.
%-----
[coefs,sizes] = wavedec2(X,level,wname);
[XDEN,cfsDEN,dimCFS] = wdencomp('lvd',coefs,sizes, ...
 wname,level,thrParams,sorh);

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

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

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

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



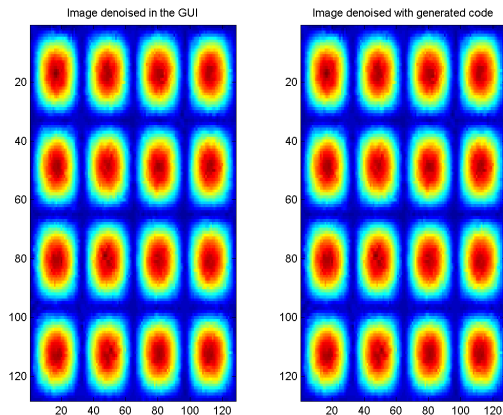
Load the denoised image in the MATLAB workspace. Compare the result with your generated code.

```
load denoisedsin.mat;
% denoised image loaded in variable X
subplot(121);
```

```

imagesc(X); title('Image denoised in the GUI');
subplot(122);
imagesc(XDEN); title('Image denoised with generated code');
% Norm of the difference is zero
norm(XDEN-X,2)

```



## 2-D Decimated Discrete Wavelet Transform Compression

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

- 1 Enter waveletAnalyzer at the MATLAB command prompt.
- 2 Select **Wavelet 2-D**.
- 3 At the MATLAB command prompt, type

```
load detfingr
```

In the **Wavelet 2-D** tool, select **File > Import from Workspace > Load Image**. When the **Import from Workspace** dialog box appears, select the X variable. Click **OK** to import the data.

- 4 Select the bior3.5 wavelet, and set Level to 3.
- 5 Click **Analyze**, then click **Compress**.

- 6 Using the default Global thresholding, set Select thresholding method to Bal.sparsity-norm (sqrt).
- 7 Click **Compress**.
- 8 **File > Generate Code (Compression Process)** generates the following code.

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

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

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

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

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

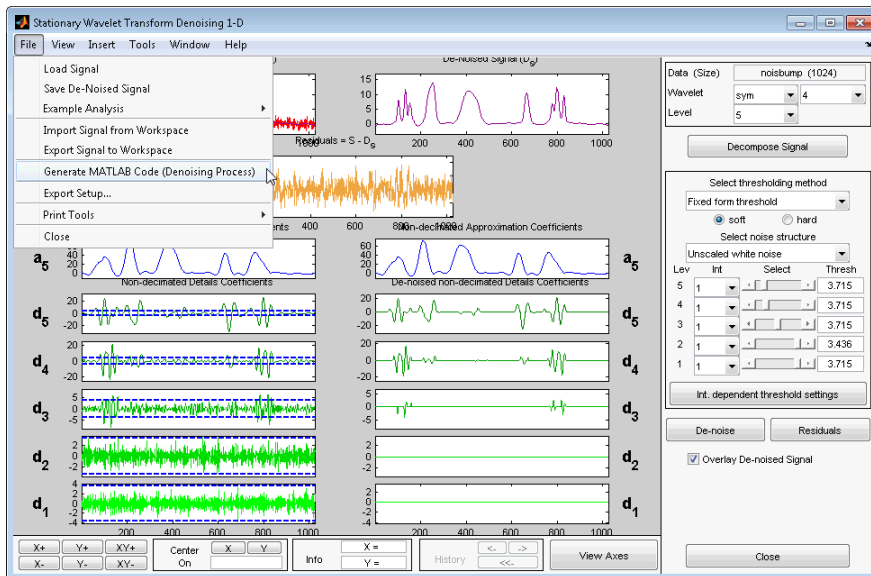
```
load detfingr.mat;
% Image data is in X
[XCMP,cfsCMP,dimCFS] = func_compress_dw2d(X);
```

- 10 Save the compressed image from the **Wavelet 2-D - - Compression** tool in a folder on the MATLAB search path. Use **File > Save > Compressed Image**, and name the file `compressed_fingerprint.mat`. Execute the following code.

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

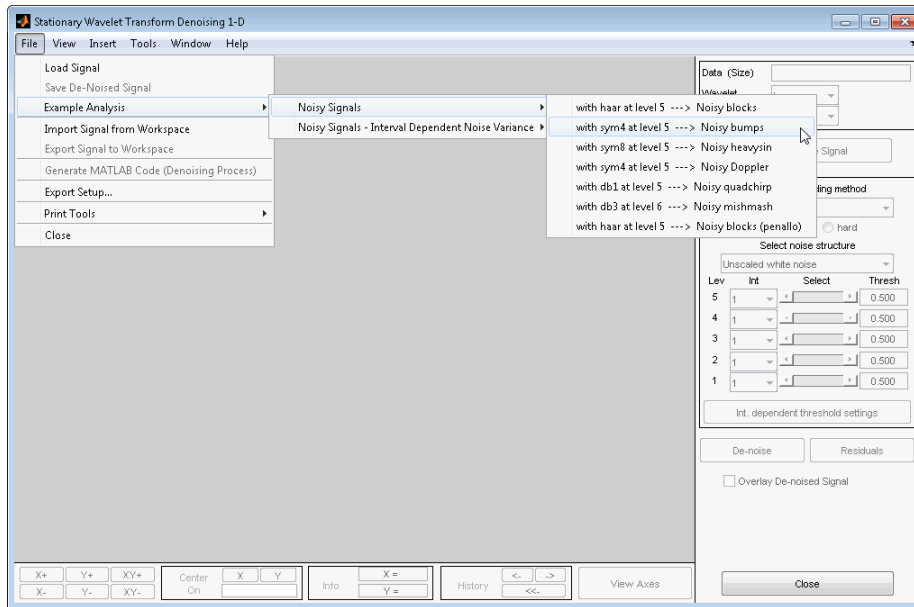
# Generate MATLAB Code for 1-D Stationary Wavelet Denoising

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



## 1-D Stationary Wavelet Transform Denoising

- 1 Enter waveletAnalyzer at the MATLAB command prompt.
- 2 Select **SWT Denoising 1-D**.
- 3 Load the Noisy bumps example. Select **File > Example Analysis > Noisy Signals > with sym4 at level 5 - -> Noisy bumps**

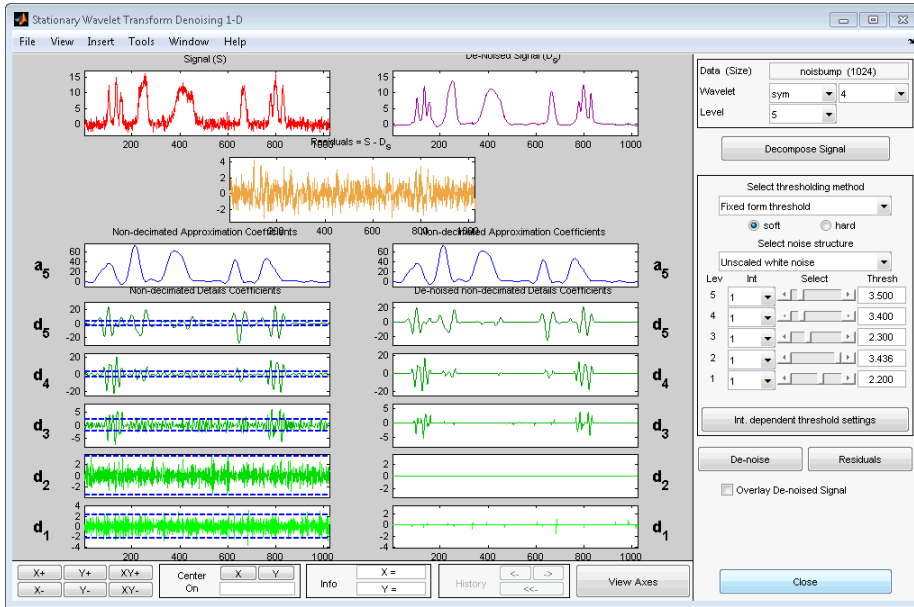


#### 4 Set the thresholds as follows:

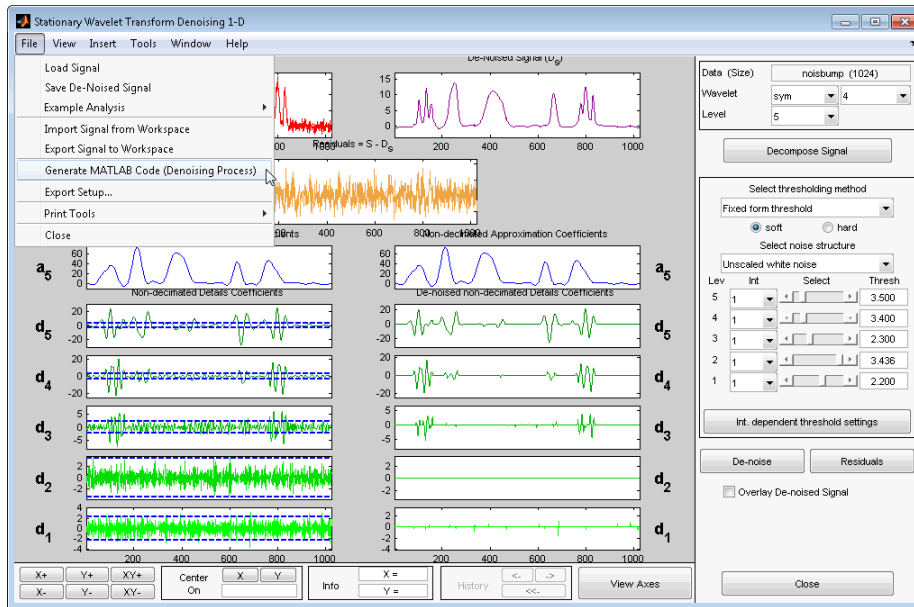
- Level 1 — 3.5
- Level 2 — 3.4
- Level 3 — 2.3
- Level 4 — 5.3
- Level 5 — 2.2

Click **Denoise**.





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



The operation generates the following MATLAB code.

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

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

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

```

]; ...
[...
1.00000000 1024.00000000 5.29965570; ...
]; ...
[...
1.00000000 1024.00000000 2.20000000; ...
]; ...
};

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

% Denoise.
%-----
len = length(SIG);
for k = 1:level
 thr_par = thrParams{k};
 if ~isempty(thr_par)
 NB_int = size(thr_par,1);
 x = [thr_par(:,1) ; thr_par(NB_int,2)];
 x = round(x);
 x(x<1) = 1;
 x(x>len) = len;
 thr = thr_par(:,3);
 for j = 1:NB_int
 if j==1 , d_beg = 0; else d_beg = 1; end
 j_beg = x(j)+d_beg;
 j_end = x(j+1);
 j_ind = (j_beg:j_end);
 wDEC(k,j_ind) = wthresh(wDEC(k,j_ind),sorb,thr(j));
 end
 end
end

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

```

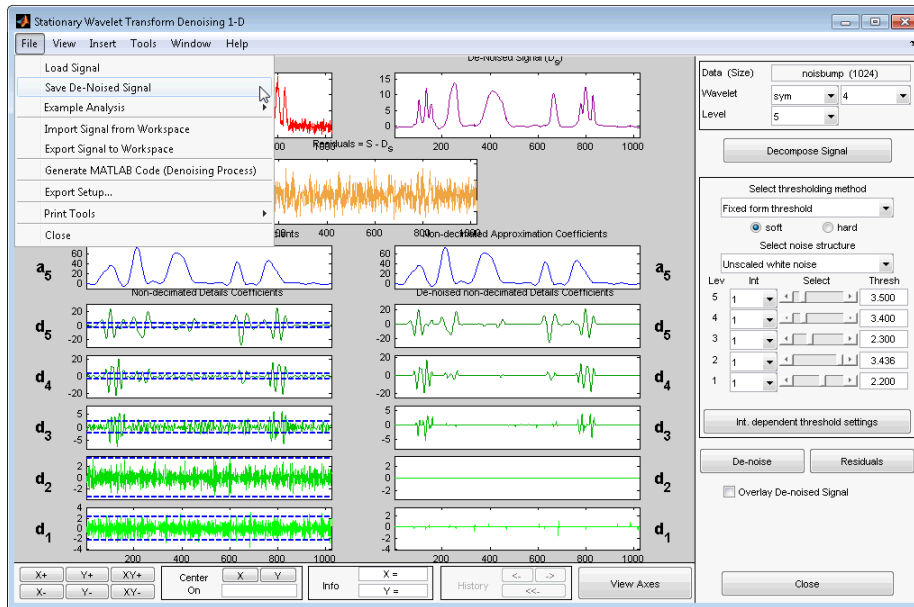
- 6** Save `func_denoise_sw1d.m` in a folder on the MATLAB search path. Execute the following code.

```

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

```

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



Execute the following code.

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

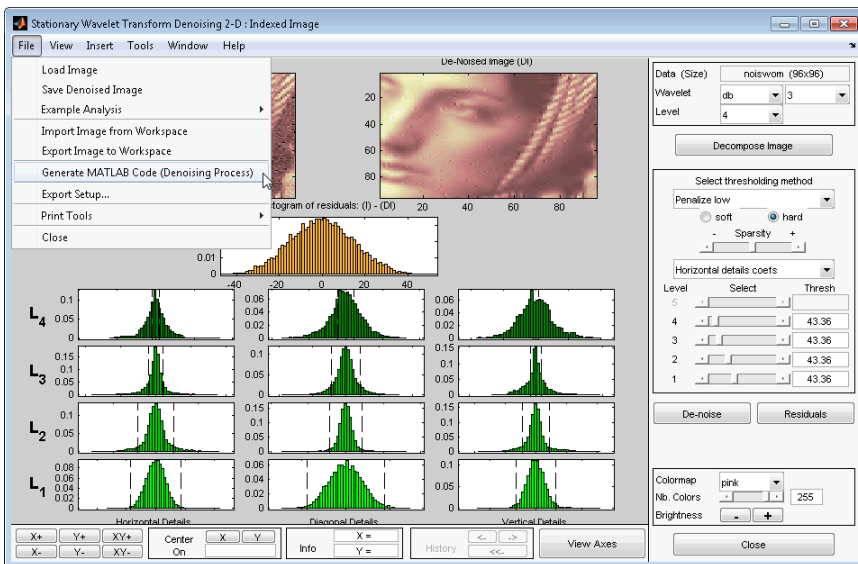
---

**Note** Thresholds are derived from a subset of the coefficients in the stationary wavelet decomposition. For more information, see “Coefficient Selection”.

---

# Generate MATLAB Code for 2-D Stationary Wavelet Denoising

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



## 2-D Stationary Wavelet Transform Denoising

- 1 Enter `waveletAnalyzer` at the MATLAB command prompt.
- 2 Select **SWT Denoising 2-D**.
- 3 At the MATLAB command prompt, type

```
load noiswom;
```

In the **SWT Denoising 2-D** tool, select **File > Import Image from Workspace**. When the **Import from Workspace** dialog box appears, select the X variable. Click **OK** to import the image.

- 4 Select the db4 wavelet, and set the **Level** to 5.
- 5 Click **Decompose Image**.
- 6 Use the default soft thresholding method with Fixed form threshold and Unscaled white noise for **Select noise structure**.
- 7 Set the following thresholds for the horizontal, diagonal, and vertical details. Ensure that you set the thresholds for the three detail coefficient types.
  - Level 1 — 5
  - Level 2 — 4
  - Level 3 — 3
  - Level 4 — 2
  - Level 5 — 1
- 8 Click **Denoise**.
- 9 Select **File > Generate Matlab Code (Denoising Process)**.

The operation generates the following MATLAB code.

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

% Analysis parameters.
%-----
wname = 'db4';
level = 5;

% Denoising parameters.
%-----
% meth = 'sqtwolog';
% scal_OR_alfa = one;
sorth = 's'; % Specified soft or hard thresholding
% Order of thresholds down each column is H,D,V
% Order in SWT2 output is H,V,D where each coefficient
% matrix is repeated L times where L is the number of levels.
thrSettings = [...
 5.0000 4.0000 3.0000 2.0000 1.0000 ; ...
 5.0000 4.0000 3.0000 2.0000 1.0000 ; ...
```

```

 5.0000 4.0000 3.0000 2.0000 1.0000 ...
];
 roundFLAG = false;

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

 isRGB = ndims(wDEC) == 4 && size(wDEC,3) == 3;
 % Denoise
 permDir = [1 3 2];

 for j = 1:level
 for kk=1:3
 ind = (permDir(kk)-1)*level+j;
 thr = thrSettings(kk,j);
 if isRGB
 wDEC(:,:,:,ind) = wthresh(wDEC(:,:,:,ind),sorh,thr);
 else
 wDEC(:,:,ind) = wthresh(wDEC(:,:,ind),sorh,thr);
 end
 end
 end

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

```

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

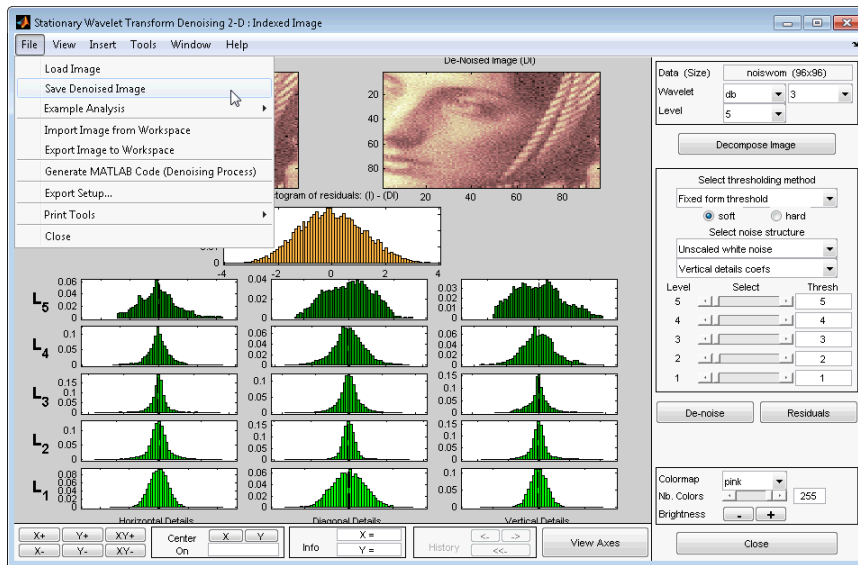
Execute the following code.

```

load noiswom
[XDEN,wDEC] = func_denoise_sw2d(X);

```

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



12 Execute the following code.

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

---

**Note** Thresholds are derived from a subset of the coefficients in the stationary wavelet decomposition. For more information, see “Coefficient Selection”.

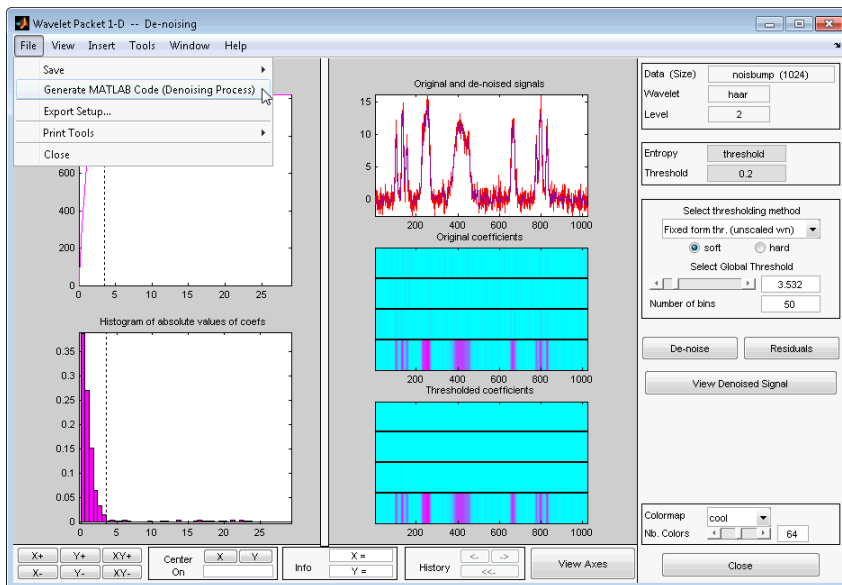
---



# Generate MATLAB Code for 1-D Wavelet Packet Denoising and Compression

## 1-D Wavelet Packet Denoising

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



- 1 Enter `waveletAnalyzer` at the MATLAB command prompt.
- 2 Select **Wavelet Packet 1-D**.
- 3 At the MATLAB command prompt, type

```
load noisbump
```

In the **\*\*\*** tool, select **File > Import from Workspace > Import Signal**. When the **Import from Workspace** dialog box appears, select the `noisbump` variable. Click **OK** to import the data.

- 4 Select the db4 wavelet, and set the **Level** to 4. Accept the default value Shannon for **Entropy**.
- 5 Click **Analyze**.
- 6 Click **Denoise**.
- 7 Under **Select thresholding method**, accept the default Fixed form thr. (unscaled wn) with the **soft** radio button enabled.

Set **Select Global Threshold** to 2.75.

- 8 Click **Denoise**.
- 9 Select **File > Generate Matlab Code (Denoising Process)**

The operation generates the following MATLAB code.

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

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

% Denoising parameters.
%-----
% meth = 'sqrtwologuwn';
sorgh = 's'; % Specified soft or hard thresholding
thrSettings = {sorgh,'nobest',2.75000000000000,1};

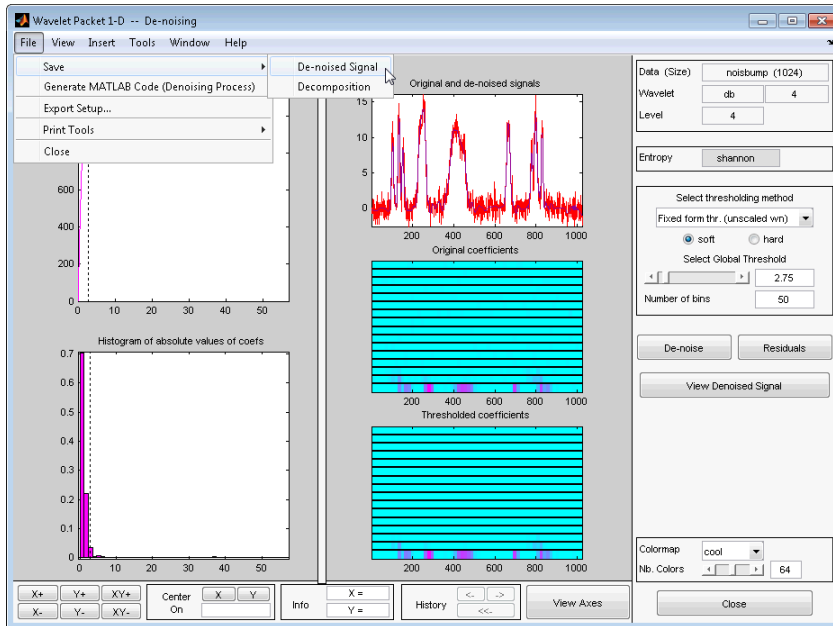
% Decompose using WPDEC.
%-----
wpt = wpdec(SIG,Lev_Anal,Wav_Nam,Ent_Nam,Ent_Par);

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

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

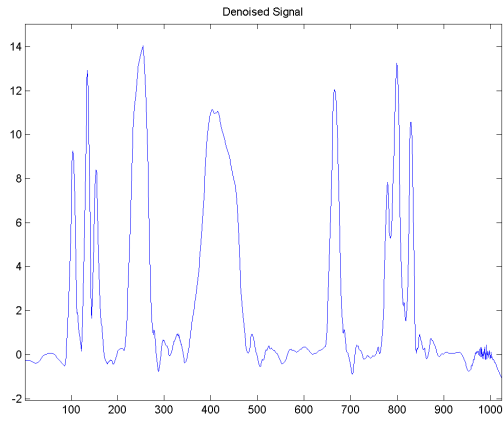
Save `func_denoise_wp1d.m` in a folder on the MATLAB search path.

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



Execute the following code.

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

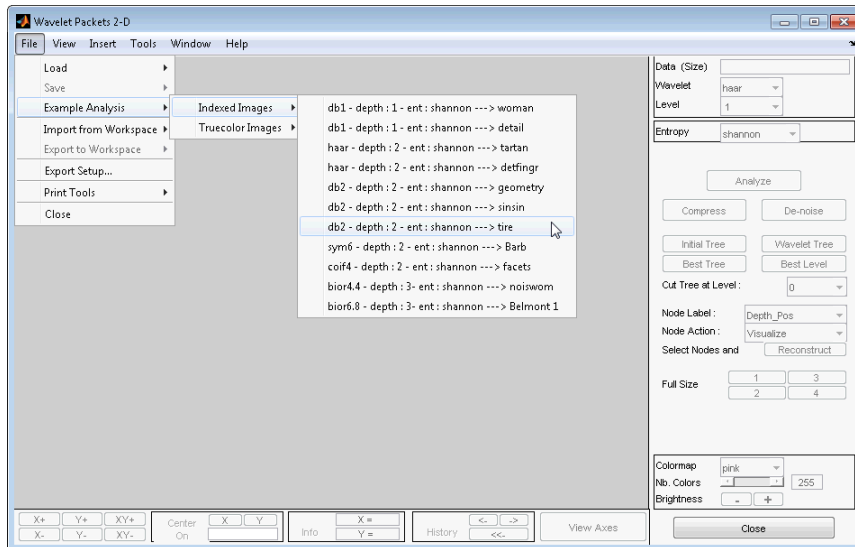


# Generate MATLAB Code for 2-D Wavelet Packet Denoising and Compression

## 2-D Wavelet Packet Compression

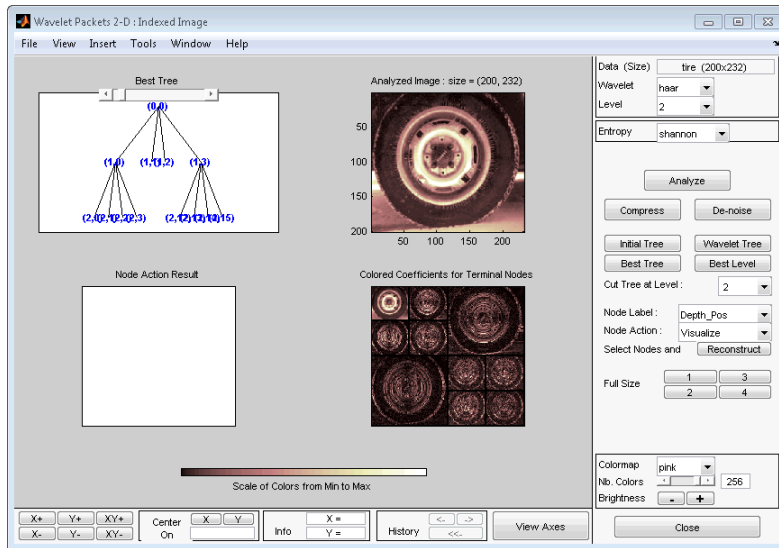
You can generate MATLAB code to reproduce app-based 2-D wavelet packet compression at the command line. You must perform this operation in the **Wavelet 2-D - - Compression** tool. You must first compress your image before you can enable the **File > Generate Matlab Code (Compression Process)** operation.

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

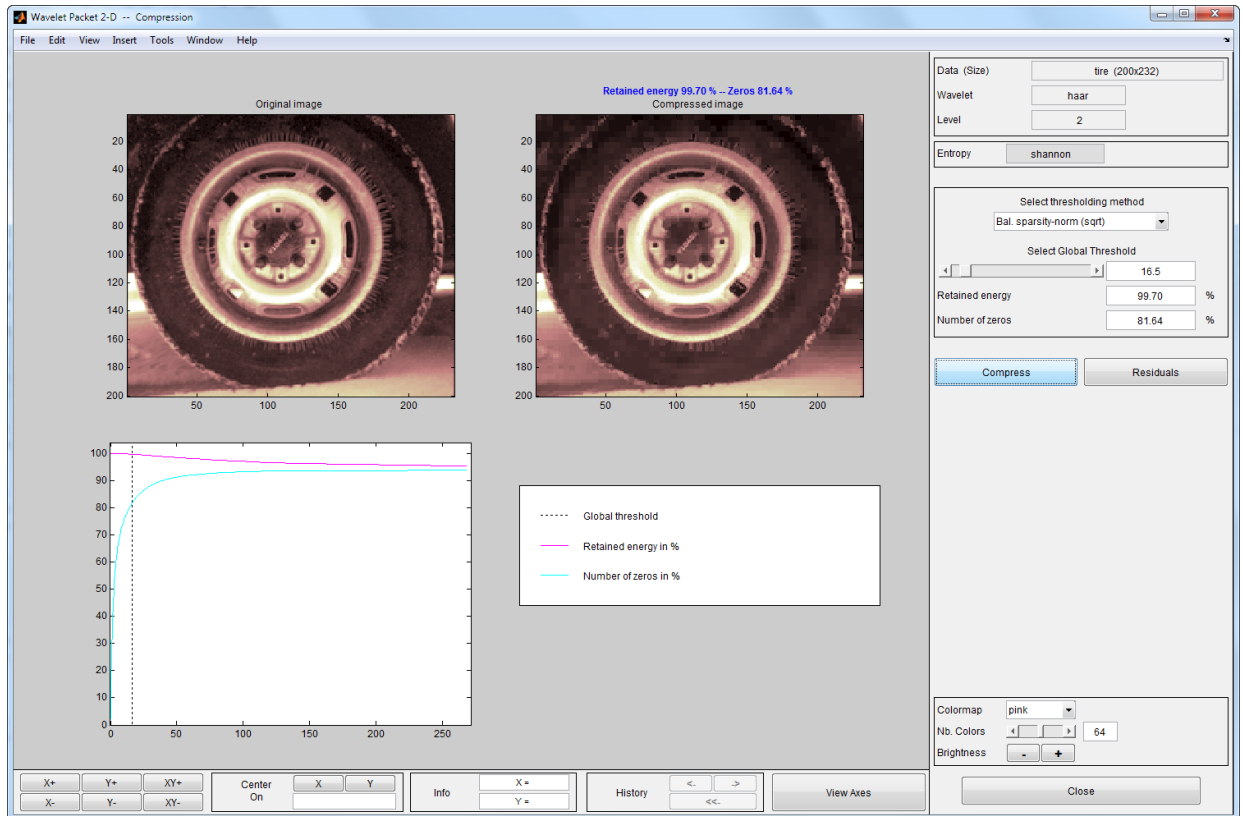


- 4 Using the default parameter settings, click **Best Tree**.

## 8 Generating MATLAB Code from Wavelet Toolbox Wavelet Analyzer App



- 5 Click **Compress**.
- 6 Set Select thresholding method to Bal.sparsity-norm (sqrt).
- 7 Click **Compress**.



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

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

% Analysis parameters.
%-----
Wav_Nam = 'haar';
Lev_Anal = 2;
Ent_Nam = 'shannon';
Ent_Par = 0;

% Compression parameters.
%-----
% meth = 'sqrtbal_sn';
```

```
sorh = 'h'; % Specified soft or hard thresholding
thrSettings = {sorh,'nobest',16.499999999999886,1};
roundFLAG = true;

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

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

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

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

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

- 10 Save the compressed image from the **Wavelet 2-D -- Compression** tool as `compressed_tire.mat` in a folder on the MATLAB search path. Use **File > Save > Compressed Image** to save the compressed image.
- 11 Execute the following code to compare the command line and **Wavelet Analyzer** app result.

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



## Generate Code to Denoise a Signal

This example shows how to use MATLAB Coder™ to generate executable code. The Wavelet Toolbox supports code generation for functions that support discrete wavelet transform (DWT), maximal overlap discrete wavelet transform (MODWT), maximal overlap wavelet packet transform (MODWPT), and denoising workflows. This example requires a MATLAB Coder license.

Define a function that uses `wden` to denoise a signal. You also specify the level to which to denoise the signal when you run the generated code.

- 1 From the MATLAB command prompt, create the file, `sigdenoise.m`.

```
edit sigdenoise
```

If you do not have permission to write to the current working folder, change the current folder to one that is writable.

- 2 Copy this `sigdenoise` function code into the `sigdenoise.m` file. Your file must include `%#codegen` to indicate that this function will generate code.

```
function xdenoise = sigdenoise(x,level)
```

```
%#codegen
```

```
wname = 'sym4';
xdenoise = wden(x,'sqrtwolog','s','mln',level,wname);
```

- 3 Save the file.

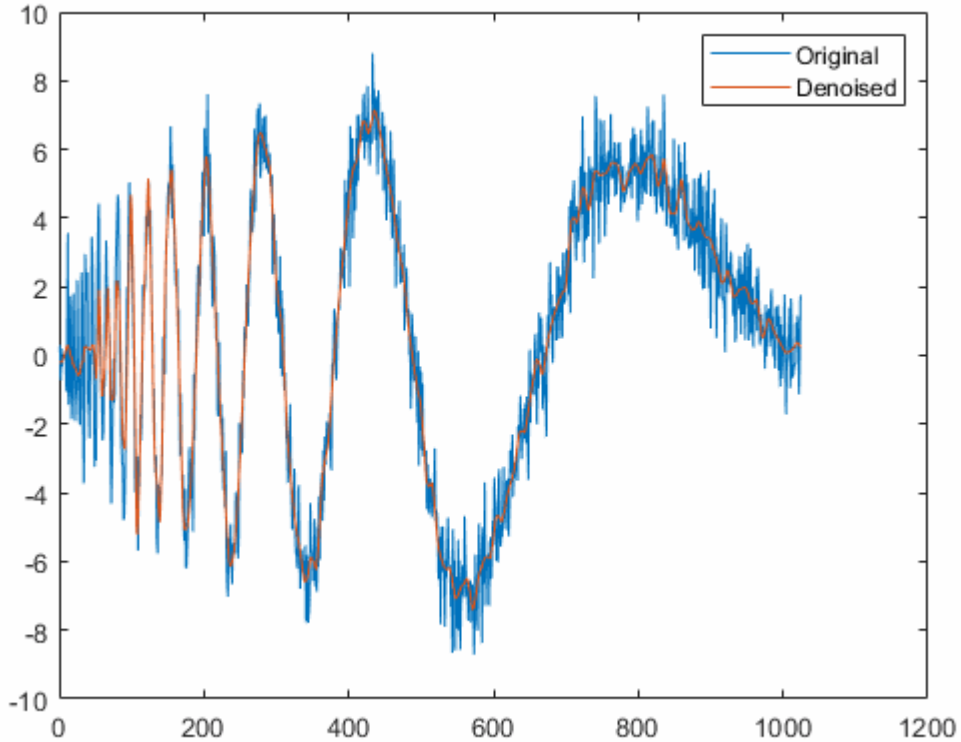
- 4 At the MATLAB command line, use the `codegen` function to compile the `sigdenoise` function into a MEX file. You can use the `-o` option to specify the name of the executable. If you do not use the `-o` option, the generated MEX file has the same name as the original MATLAB file with `_mex` appended. You can include the `-report` option to generate a compilation report. This report shows the original MATLAB code and the associated files created during code generation. The `-args` option specifies the data types of the inputs required to run the generated code. In this case, a variable-size row vector and a scalar input are required.

```
codegen sigdenoise.m -config:mex -args {coder.typeof(1,[1,inf],[false,true]),0}
```

- 5 At the MATLAB command line, run the generated code on noisy Doppler data and denoise it to level three. Compare the original and denoised signals.

```
load noisdopp
xden = sigdenoise_mex(noisdopp,3);
```

```
plot([noisdopp',xden'])
legend('Original','Denoised')
```



For a list of Wavelet Toolbox functions supported for code generation and associated limitations, see “Code Generation Support, Usage Notes, and Limitations” on page 8-37. For more information on code generation, see “Getting Started with MATLAB Coder” (MATLAB Coder).

## Code Generation Support, Usage Notes, and Limitations

An asterisk (\*) indicates that the reference page has usage notes and limitations for C/C++ code generation.

|                |                                                           |
|----------------|-----------------------------------------------------------|
| appcoef*       | 1-D approximation coefficients                            |
| appcoef2*      | 2-D approximation coefficients                            |
| cwtfilterbank* | Continuous wavelet transform filter bank                  |
| cwtfreqbounds* | CWT maximum and minimum frequency or period               |
| ddencomp*      | Default values for denoising or compression               |
| detcoef        | 1-D detail coefficients                                   |
| detcoef2       | 2-D detail coefficients                                   |
| dwt            | Single-level 1-D discrete wavelet transform               |
| dwt2           | Single-level discrete 2-D wavelet transform               |
| dyadup*        | Dyadic upsampling                                         |
| emd            | Empirical mode decomposition                              |
| filterbank     | Shearlet system filters                                   |
| framebounds    | Shearlet system frame bounds                              |
| idwt           | Single-level inverse discrete 1-D wavelet transform       |
| idwt2*         | Single-level inverse discrete 2-D wavelet transform       |
| imodwpt        | Inverse maximal overlap discrete wavelet packet transform |
| imodwt         | Inverse maximal overlap discrete wavelet transform        |
| isheart2       | Inverse shearlet transform                                |
| mdwtdec*       | Multisignal 1-D wavelet decomposition                     |
| mdwtrec*       | Multisignal 1-D wavelet reconstruction                    |
| meyeraux       | Meyer wavelet auxiliary function                          |
| modwpt         | Maximal overlap discrete wavelet packet transform         |
| modwptdetails  | Maximal overlap discrete wavelet packet transform details |
| modwt          | Maximal overlap discrete wavelet transform                |
| modwtmra       | Multiresolution analysis based on MODWT                   |

|                |                                                                                     |
|----------------|-------------------------------------------------------------------------------------|
| numshears      | Number of shearlets                                                                 |
| qmf            | Scaling and Wavelet Filter                                                          |
| shearletSystem | Bandlimited shearlet system                                                         |
| sheart2        | Shearlet transform                                                                  |
| thselect       | Threshold selection for denoising                                                   |
| wavedec*       | 1-D wavelet decomposition                                                           |
| wavedec2*      | 2-D wavelet decomposition                                                           |
| waverec*       | 1-D wavelet reconstruction                                                          |
| waverec2*      | 2-D wavelet reconstruction                                                          |
| wden*          | Automatic 1-D denoising                                                             |
| wdencomp*      | Denoising or compression                                                            |
| wextend*       | Extend vector or matrix                                                             |
| wnoisest       | Estimate noise of 1-D wavelet coefficients                                          |
| wthcoef        | 1-D wavelet coefficient thresholding                                                |
| wthcoef2       | Wavelet coefficient thresholding 2-D                                                |
| wthresh        | Soft or hard thresholding                                                           |
| wvd*           | Wigner-Ville distribution and smoothed pseudo Wigner-Ville distribution             |
| xwvd*          | Cross Wigner-Ville distribution and cross smoothed pseudo Wigner-Ville distribution |

# Wavelet Analyzer App Features Summary

This appendix explains some of the features of the Wavelet Analyzer app.

## 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
- Zooming in on plots
- 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.

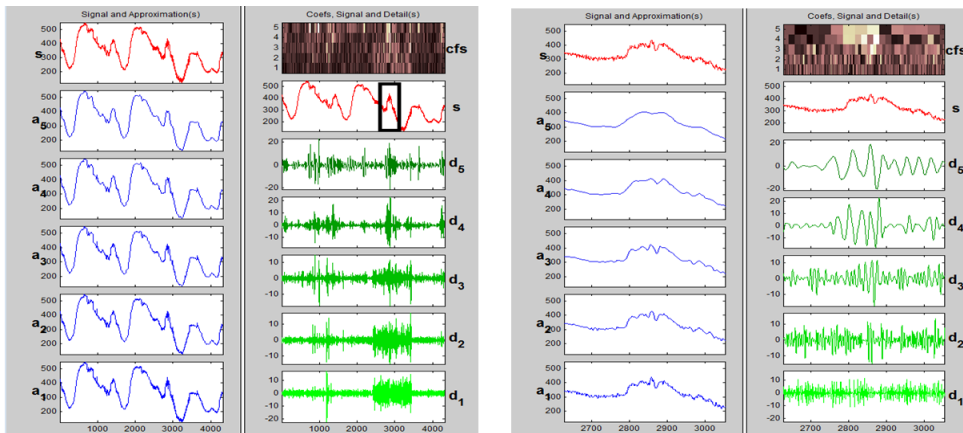
| <b>Signal</b>                | <b>Shown In</b>                                     |
|------------------------------|-----------------------------------------------------|
| Original                     | Red                                                 |
| Reconstructed or synthesized | Yellow                                              |
| Approximations               | Variegated shades of blue<br>(high level = darker)  |
| Details                      | Variegated shades of green<br>(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.

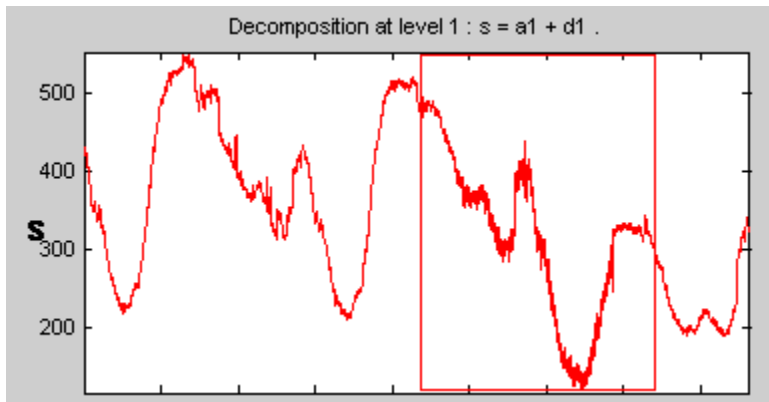
Click and drag your mouse over the region you want to zoom. Clicking **XY+** results in the zoom being applied to all the plots.



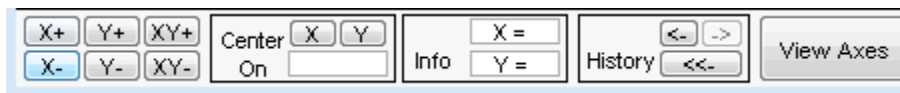
- Zoom in on relevant detail.

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.

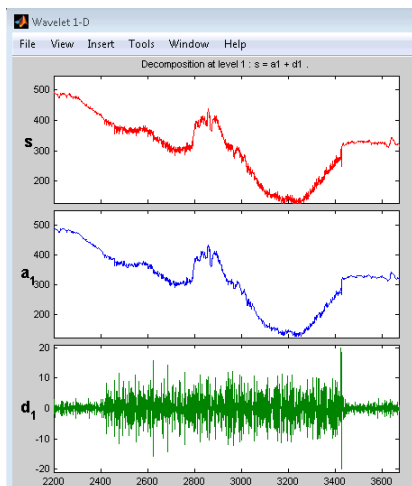
Drag a rubber band box (by holding down the left mouse button) over the portion of the signal you want to magnify. Here, we've selected the noisy part of the original signal.



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



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



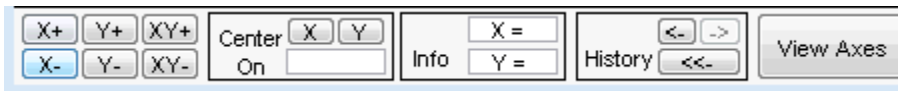
The other zoom controls do more or less what you'd expect them to. The **X-** button, for example, zooms out horizontally. The history function keeps track of all your views of the signal. Return to a previous zoom level by clicking the left arrow button.



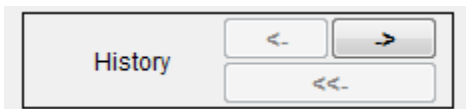
- **Zooming in on Detail**

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.


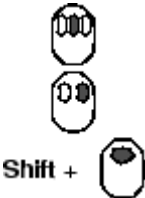
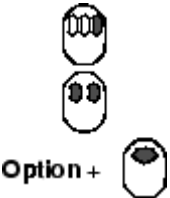


- The **History** pane enables you to remember how you zoom the axes so that you can toggle back and forth between views.



## Using the Mouse

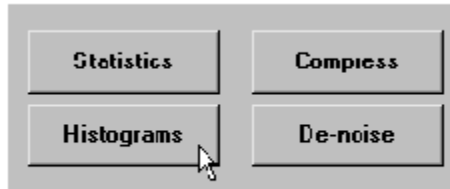
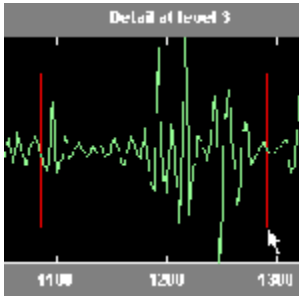
Wavelet Toolbox software uses three types of mouse control.

| 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..

## 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).

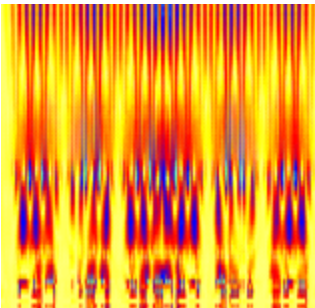


This is more than an aesthetic adjustment because you are likely to see different features depending on your colormap selection.

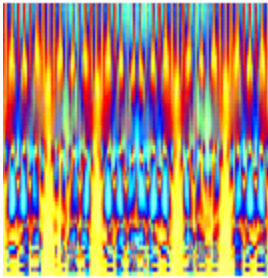
## Controlling the Number of Colors

The **Nb. Colors** slider, located at the bottom right of the window, allows you to adjust how many colors the tool uses to plot images or coefficients (wavelet or wavelet packet). You can also use the edit control to adjust the number of colors. Adjusting the number of colors can highlight different features of the plot.

Consider the coefficients plot of the Koch curve generated in the **Continuous Wavelet** tool, shown here using 129 colors.



and here using 68 colors.



### Controlling the Coloration Mode

In the **Continuous Wavelet** tools, the coloration of coefficients can be done in several different ways.

Coloration mode — Three parameters are used color the coefficients.

- **init** or **current** — When you select **init**, coloration is performed with all the coefficient values. When you select **current**, only the coefficients displayed in the current axis limits are used.
- **by scales** or **all scales** — When you select **by scale**, the coloration is done separately for each scale. When you select **all scales**, all scales are used.
- **abs** — When you select **abs**, the absolute values of the coefficients are used.

init + by scale + abs

init + by scale

init + all scales + abs

init + all scales

current + by scale + abs

current + by scale

current + all scales + abs

current + all scales

---

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

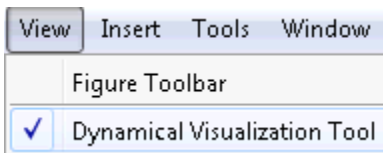
### General Menu Bar

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.

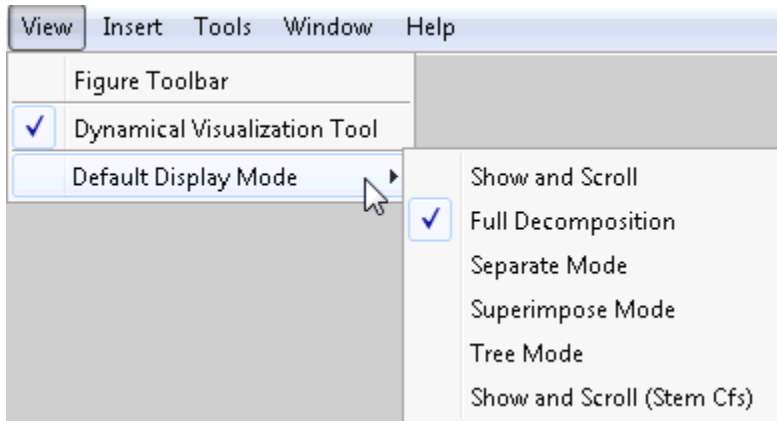


Enabling the **Dynamical Visualization Tool** activates the zoom, center, history, and axes options at the bottom of the interactive tool.

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.

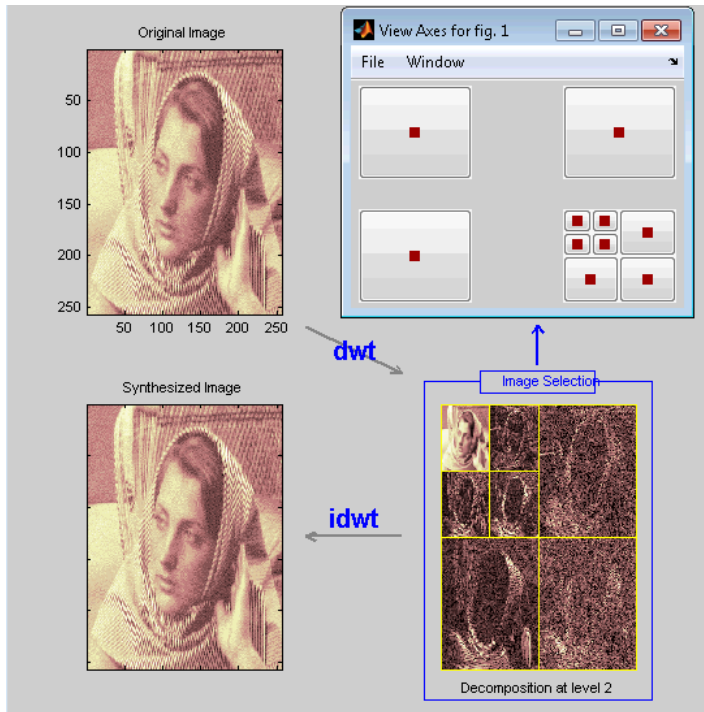
### Default Display Mode Option

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.



## 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.

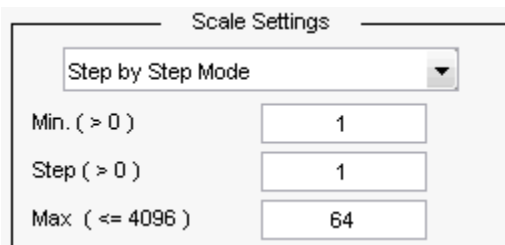
Clicking the **View Axes** toggle button again closes the **View Axes** figure.

## Continuous Wavelet Tool Features

Here is an example of an option that allows you to perform analysis using different scale modes.

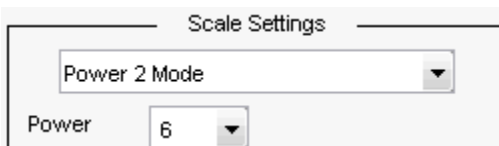
### Scale Settings

- **Step by Step Mode** — Specify the initial scale, the step size, and the maximum scale.



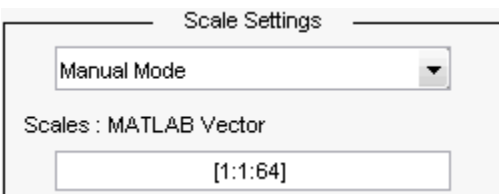
The screenshot shows a dialog box titled "Scale Settings". At the top, there is a dropdown menu set to "Step by Step Mode". Below this, there are three input fields: "Min. (> 0)" with the value "1", "Step (> 0)" with the value "1", and "Max (<= 4096)" with the value "64".

- **Power 2 Mode** — The scales are  $2^0, 2^1$ , up to power you select in the Power drop down menu. These are the same scales used for discrete analysis.



The screenshot shows a dialog box titled "Scale Settings". At the top, there is a dropdown menu set to "Power 2 Mode". Below this, there is a "Power" dropdown menu set to "6".

- **Manual Mode** — Enter a vector of scales.

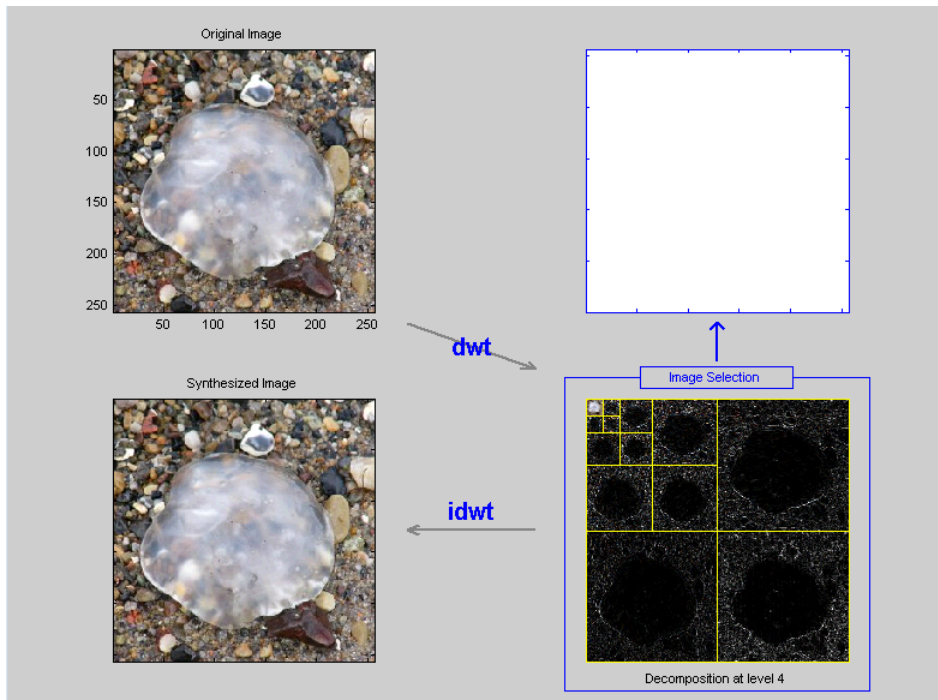


The screenshot shows a dialog box titled "Scale Settings". At the top, there is a dropdown menu set to "Manual Mode". Below this, there is a label "Scales : MATLAB Vector" and a text input field containing the MATLAB vector "[1:1:64]".

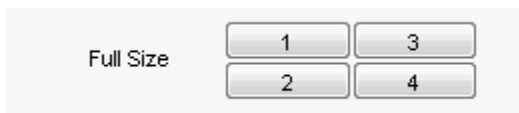


## Wavelet 2-D Tool Features

The **Wavelet 2-D** tool is described in “2-D Wavelet Analysis Using the Wavelet Analyzer App” on page 3-195. Here is an example of an option that allows you to view a selected part of the window at a full window resolution.



In the **Full Size** menu on the right side of the interactive tool



choose the image you want to view as full size. Click your selection again to restore the original view.

## Wavelet Packet Tool Features (1-D and 2-D)

### Coefficients Coloration

**NAT** or **FRQ** is for Natural or Frequency order.

**By level** or **Global** is for a coloration made level by level or taking all detail levels.

**abs** is used to take the absolute values of coefficients.

### Node Action

When you select a node in the tree, the selected option is performed. A complete description of options is provided in the following sections.

### Node Label

The node labels can be changed using the pop-up menu. For example, the **Type** option labels the nodes with **(a)** for approximation and **(d)** for detail.

### Node Action Functionality

The available options in the **Node Action** menu are

- **Visualize:** When you select a node in the wavelet packet tree the corresponding signal appears.
- **Split/Merge:** If a terminal node is selected, it is split, growing the wavelet packet tree. Selecting other nodes has the behavior of merging all the nodes below it in the wavelet packet tree.
- **Recons.:** When you select a node in the wavelet packet tree, the corresponding reconstructed signal appears.
- **Select On/Off:** When **On**, you can select many nodes in the wavelet packet tree. Then you can reconstruct a synthesized signal from the selected nodes using the **Reconstruct** button on the main window. Use the **Off** selection to deselect all the previous selected nodes.
- **Statistics:** When you select a node in the wavelet packet tree, the **Statistics** tool appears using the signal corresponding to the selected node.

- **View Col. Cfs.:** When active, this option removes all the colored coefficients displayed, and lets you redraw only the corresponding coefficients by selecting a node in the wavelet packet tree.

## Wavelet Display Tool

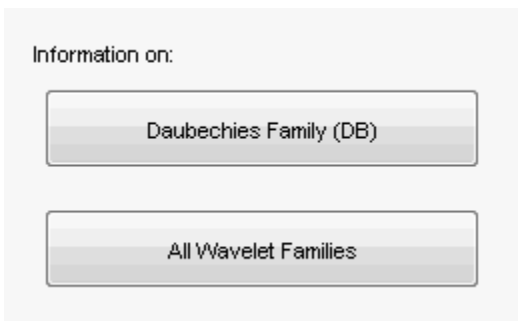
The **Wavelet Display** tool is mentioned in the section “Introduction to Wavelet Families” in the *Wavelet Toolbox Getting Started Guide*.

The **Refinement** drop down menu allows you choose the number of points that the wavelet and scaling functions are computed over. The number of points are in powers of 2. In the following figure, the db2 scaling and wavelet functions are computed over a grid of  $2^8$  points.



The screenshot shows a control panel for the Wavelet Display tool. It features a 'Wavelet' section with a dropdown menu set to 'db' and a numeric input field set to '2'. Below this is a 'Refinement' section with a dropdown menu set to '8'. At the bottom of the panel is a large 'Display' button.

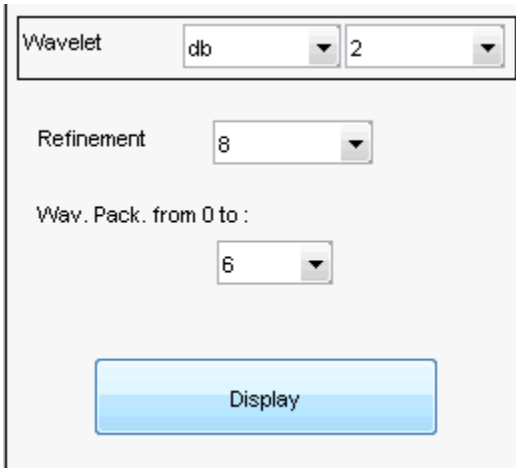
The **Information on:** selections allow you to obtain more detailed information on the current wavelet family, or all supported families.



The screenshot shows the 'Information on:' section of the tool. It contains two buttons: 'Daubechies Family (DB)' and 'All Wavelet Families'.

## Wavelet Packet Display Tool

The Refinement drop down menu allows you choose the number of points that the wavelet packets are computed over. The number of points are in powers of 2. In the following figure, the db2 wavelet packets are computed over a grid of  $2^8$  points.

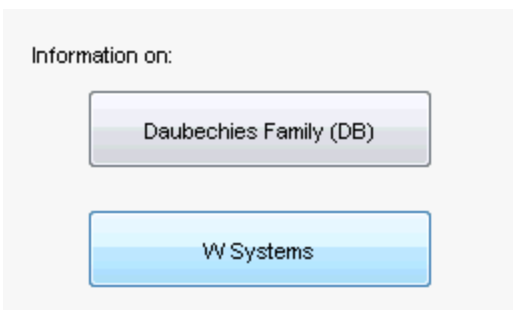


The screenshot shows the configuration interface of the Wavelet Packet Display Tool. It includes the following elements:

- A "Wavelet" section with a dropdown menu set to "db" and a numeric dropdown set to "2".
- A "Refinement" section with a numeric dropdown menu set to "8".
- A "Wav. Pack. from 0 to :" section with a numeric dropdown menu set to "6".
- A blue "Display" button at the bottom.

The Wav. Pack. from 0 to: allows you to choose the number of wavelet packets to display.

The **Information on:** selections allow you to obtain more detailed information on the current wavelet family, **Daubechies Family (DB)**, or wavelet packets in general, **W Systems**.



The screenshot shows the "Information on:" section of the tool. It contains two buttons:

- A grey button labeled "Daubechies Family (DB)".
- A blue button labeled "W Systems".

