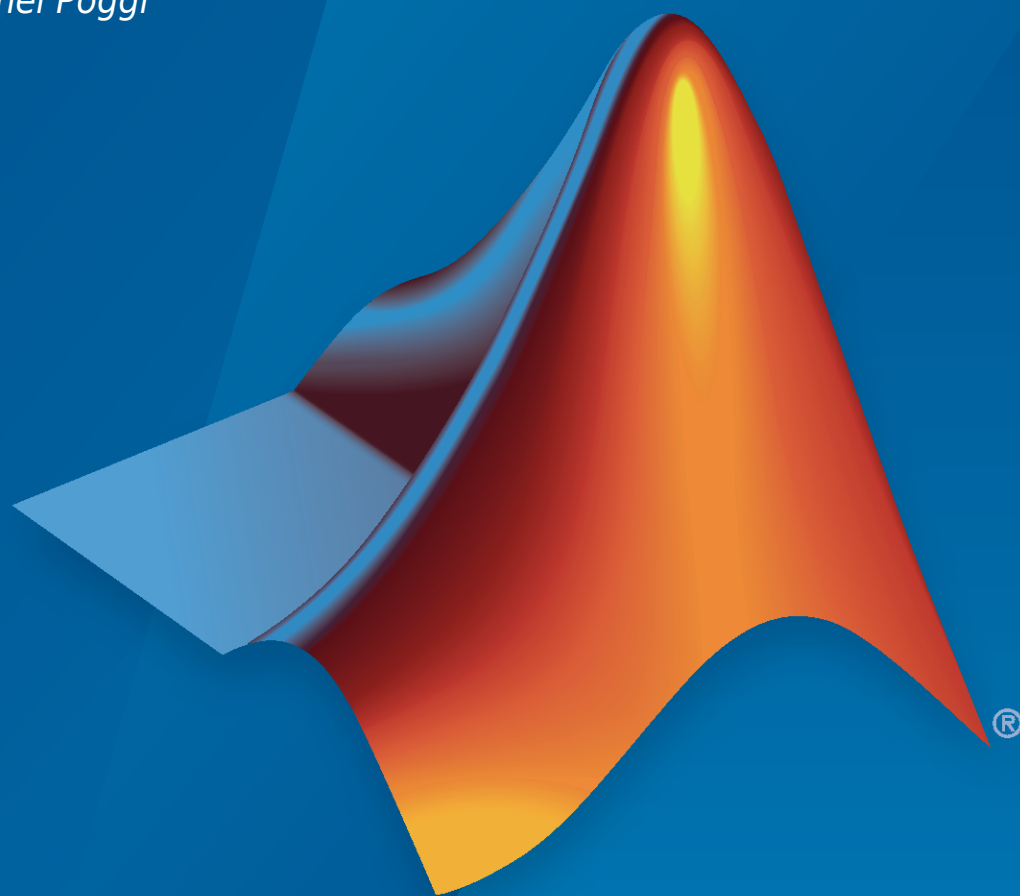# Wavelet Toolbox™

User's Guide

Michel Misiti
Yves Misiti
Georges Oppenheim
Jean-Michel Poggi

MATLAB®

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

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)
March 2020	Online only	Revised for Version 5.4 (Release 2020a)
September 2020	Online only	Revised for Version 5.5 (Release 2020b)

Contents

Wavelets, Scaling Functions, and Conjugate Quadrature Mirror Filters

1

3

Time-Frequency Analysis

4

5

Denoising, Nonparametric Function Estimation, and Compression

6

Matching Pursuit

7

Code Generation from MATLAB Support in Wavelet Toolbox

8

Special Topics

9

Featured Examples — Time-Frequency Analysis

10

Featured Examples — Discrete Multiresolution Analysis

11

Generating MATLAB Code from Wavelet Toolbox Wavelet Analyzer App

14

Wavelet Analyzer App Features Summary

A

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

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" on page 10-27 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
`'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

Wavelet Family Short Name	Wavelet Family Name
`'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');
```
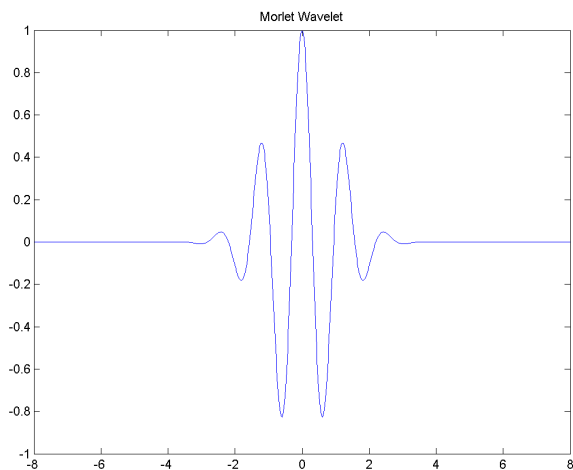

For wavelets associated with a multiresolution analysis, you can compute both the scaling function and wavelet. The following code returns the scaling function and wavelet for the Daubechies' extremal phase wavelet with 4 vanishing moments.

```
[phi,psi,xval] = wavefun('db4',10);
subplot(211);
plot(xval,phi);
title('db4 Scaling Function');
subplot(212);
plot(xval,psi);
title('db4 Wavelet');
```
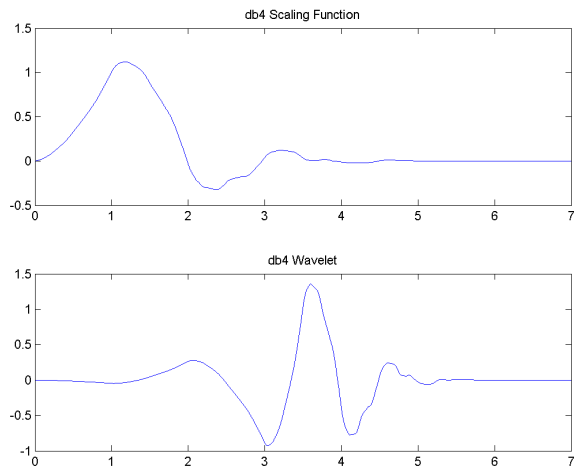
In discrete wavelet analysis, the analysis and synthesis filters are of more interest than the associated scaling function and wavelet. You can use `wfilters` to obtain the analysis and synthesis filters.

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');
```
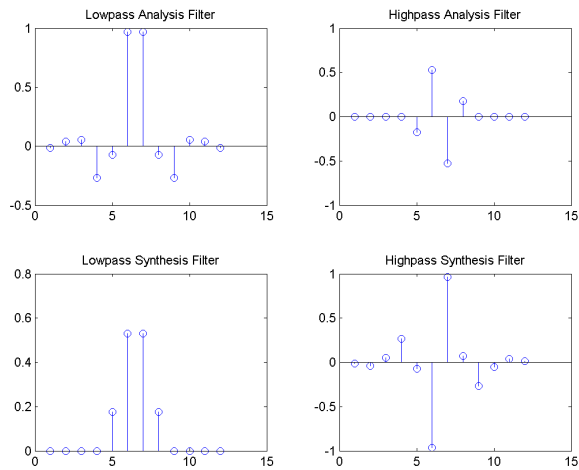
Daubechies Wavelets: dbN

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.
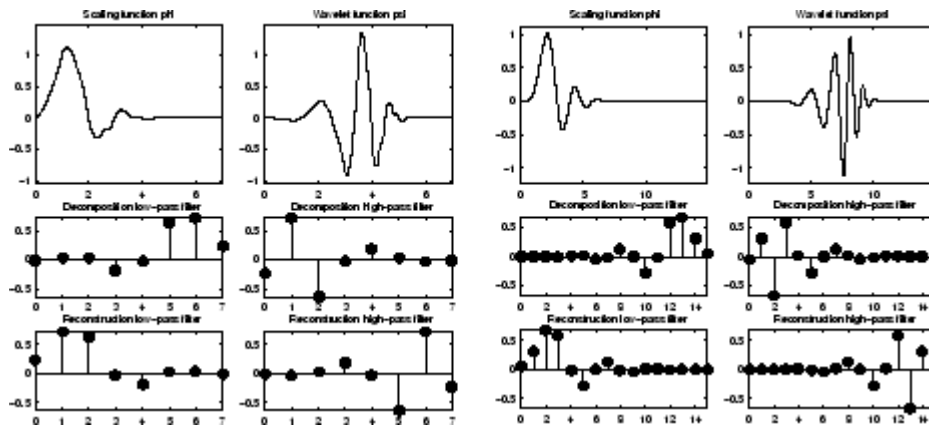
Daubechies Wavelets db4 on the Left and db8 on the Right

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.
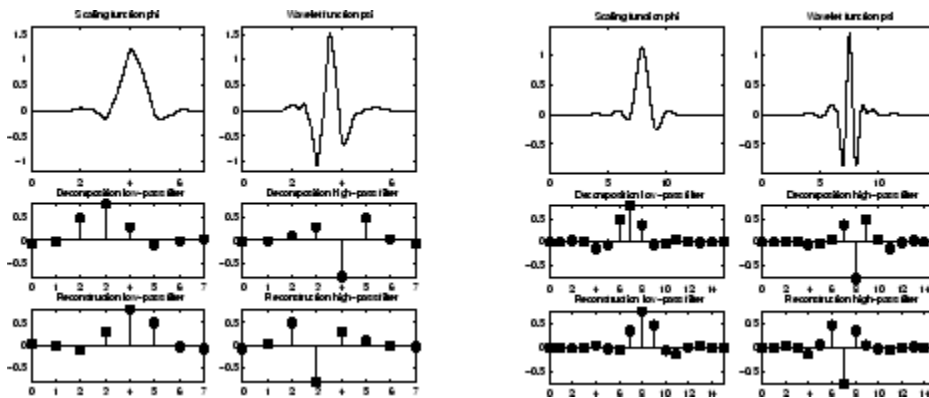
Symlets sym4 on the Left and sym8 on the Right

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.
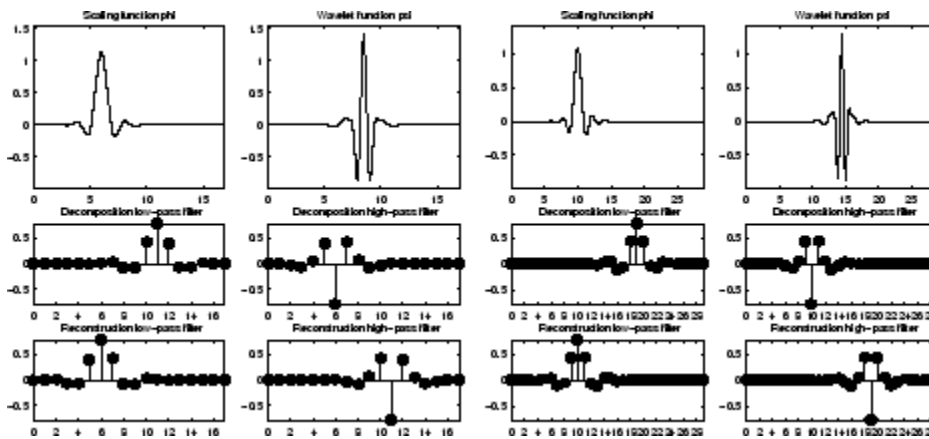
Coiflets coif3 on the Left and coif5 on the Right

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 $\varphi_{j,l}$ 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, 262–285 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');
```
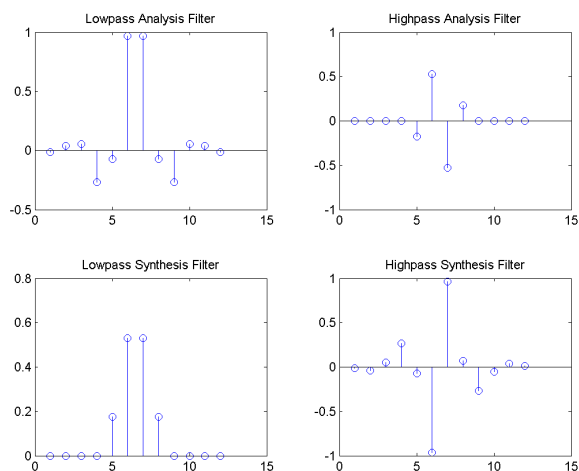

Reverse Biorthogonal Wavelet Pairs: rbioNr.Nd

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.
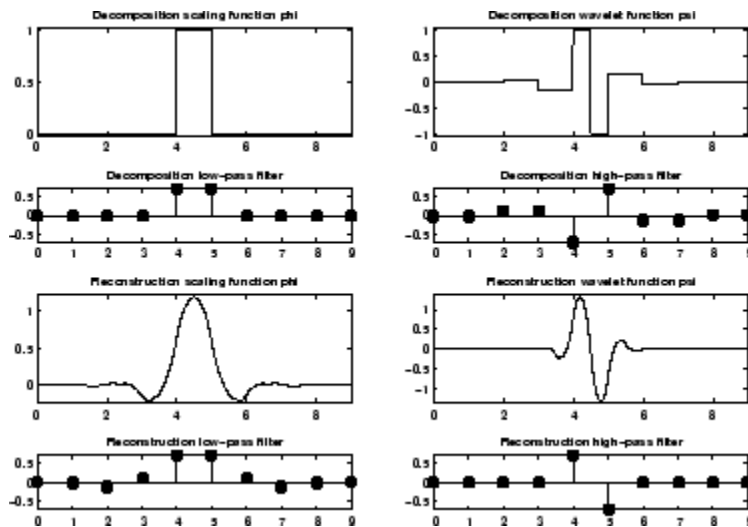
Reverse Biorthogonal Wavelet rbio1.5

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.
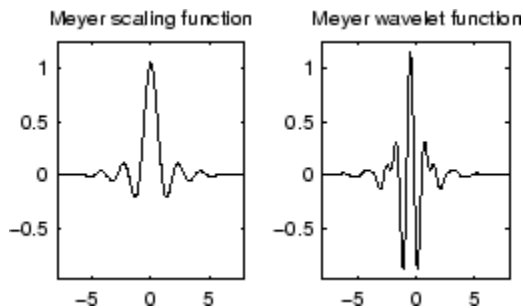
Meyer Wavelet

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

- Wavelet function

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

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

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

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

- Scaling function

$$\widehat{\phi}(\omega) = \big(^2 \quad if \quad |\omega| \le \frac{2\pi}{3}$$

$$\widehat{\phi}(\omega) = \Big(^2\cos\Big(\frac{\pi}{2}\nu\Big(\frac{3}{2\pi}|\omega| - 1\Big)\Big) \quad if \quad \frac{2\pi}{3} \le |\omega| \le \frac{4\pi}{3}$$

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

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

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

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

This property holds also for the derivatives

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

The wavelet is infinitely differentiable.

Note Although the Meyer wavelet is not compactly supported, there exists a good approximation leading to FIR filters 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 pth derivative of f.

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

You can obtain a survey of the main properties of this family by typing `waveinfo('gaus')` from the MATLAB command line.
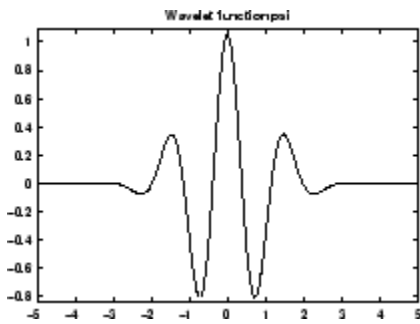
Gaussian Derivative Wavelet gaus8

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');
```
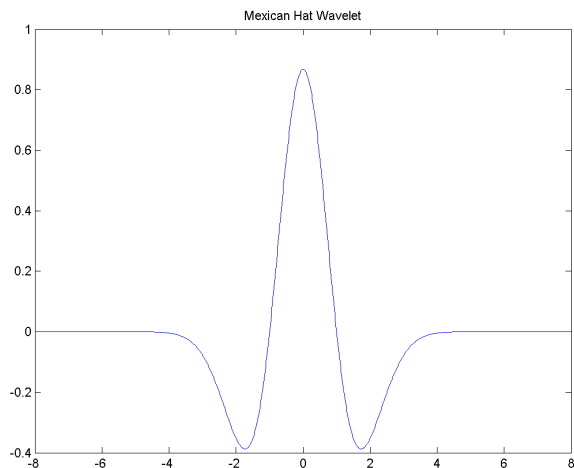

Morlet Wavelet: morl

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');
```
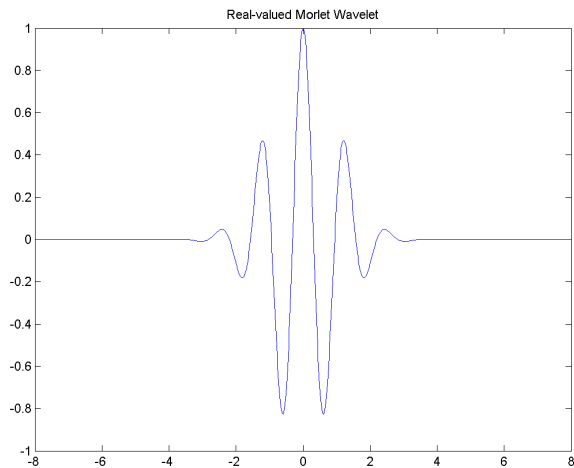

The Morlet wavelet does not technically satisfy the admissibility condition..

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

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

You can obtain a survey of the main properties of this family by typing `waveinfo('cgau')` from the MATLAB command line.
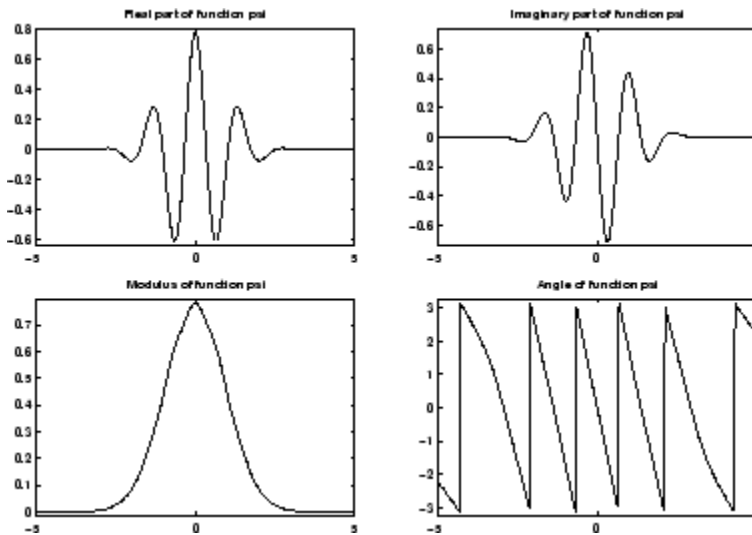
Complex Gaussian Wavelet cgau8

Complex Morlet Wavelets: cmor

See [Teo98] pages 62–65.

A complex Morlet wavelet is defined by

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

depending on two parameters:

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

You can obtain a survey of the main properties of this family by typing `waveinfo('cmor')` from the MATLAB command line.
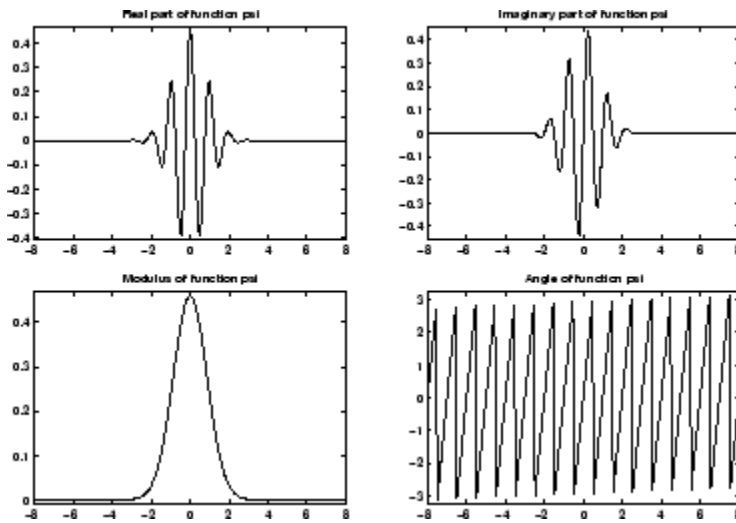
Complex Morlet Wavelet morl 1.5-1

Complex Frequency B-Spline Wavelets: fbsp

See [Teo98] pages 62–65.

A complex frequency B-spline wavelet is defined by

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

depending on three parameters:

- m is an integer order parameter ($m \geq 1$).
- f_b is a bandwidth parameter.
- f_c is a wavelet center frequency.

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

Real part of function psi   **Imaginary part of function psi**

Modulus of function psi   **Angle of function psi**
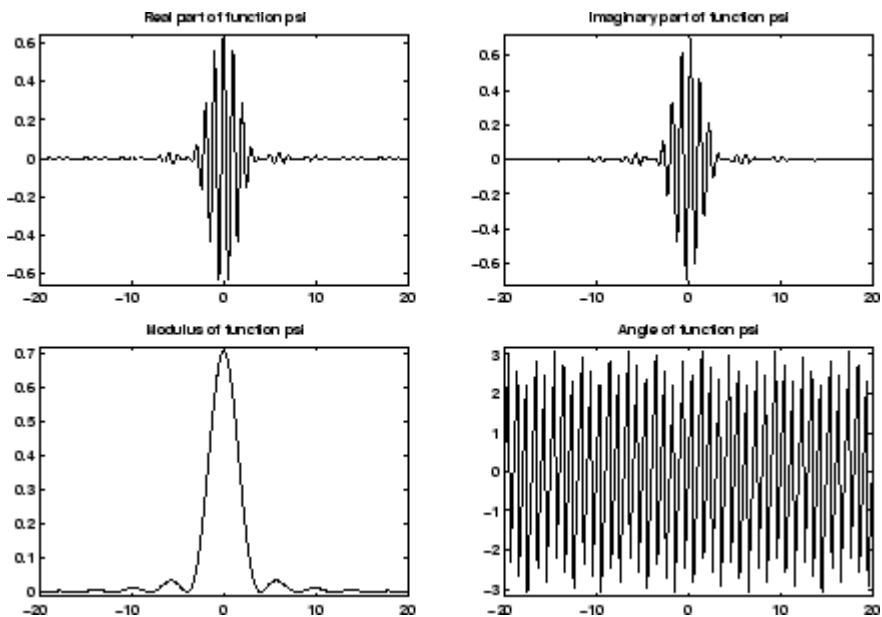
Complex Frequency B-Spline Wavelet fbsp 2-0.5-1

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.
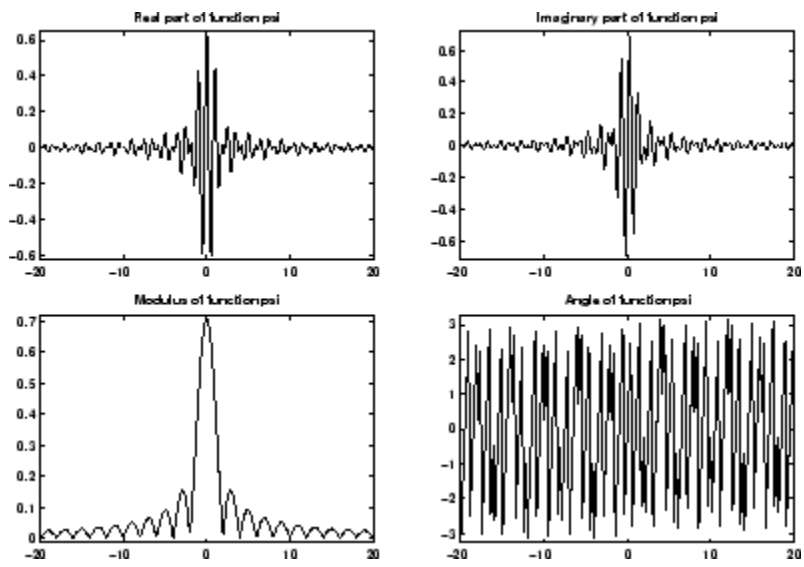
Complex Shannon Wavelet shan 0.5-1

Wavelet Families and Associated Properties — I

Property	morl	mexh	meyr	haar	*db*N	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" in "Choose a Wavelet".

Orthogonal — See "General Considerations" in "Choose a Wavelet".

Biorthogonal — See "Biorthogonal Wavelet Pairs: biorNr.Nd" on page 1-6.

Vanishing moments —See "General Considerations" 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" in "Choose a Wavelet".

Orthogonal — See "General Considerations" in "Choose a Wavelet".

Biorthogonal — See "Biorthogonal Wavelet Pairs: biorNr.Nd" on page 1-6.

Vanishing moments — See "General Considerations" 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 3-34.

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 1-19
- "Lifting Functions" on page 1-24

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 3-34. 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..
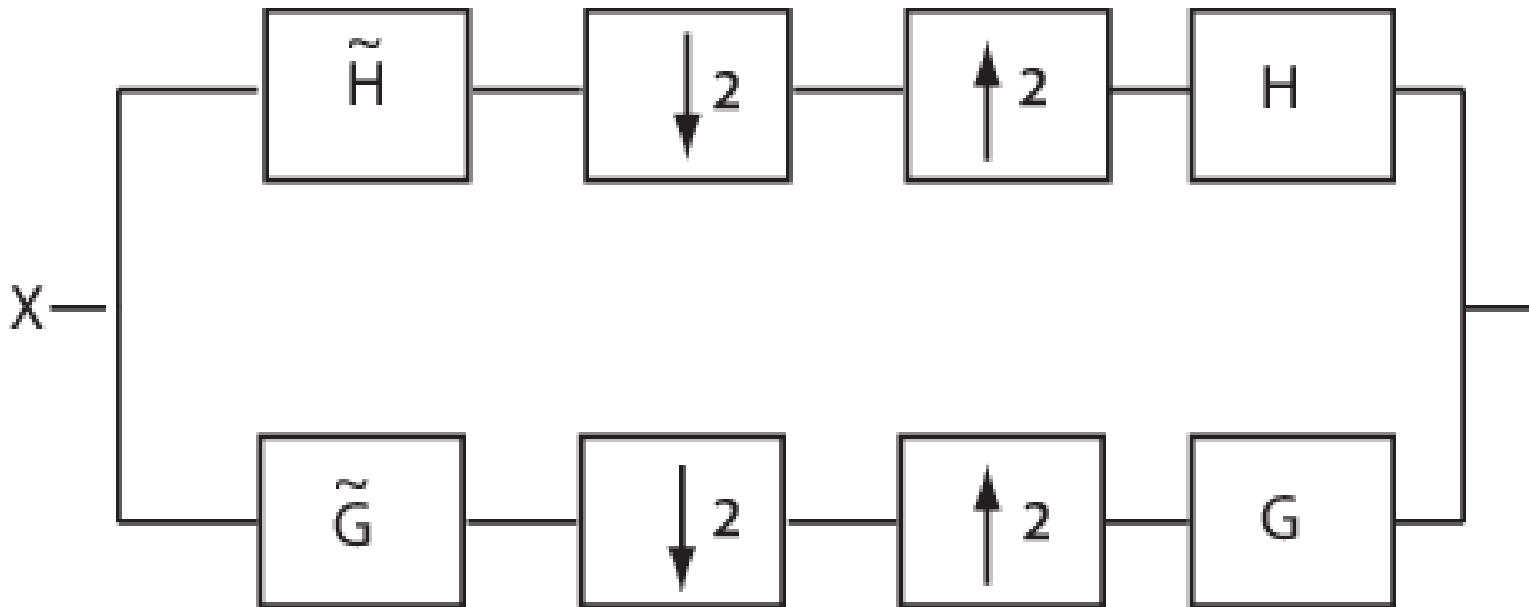
This leads to the following two conditions for a perfect reconstruction (PR) filter bank:

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

and

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

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;
```
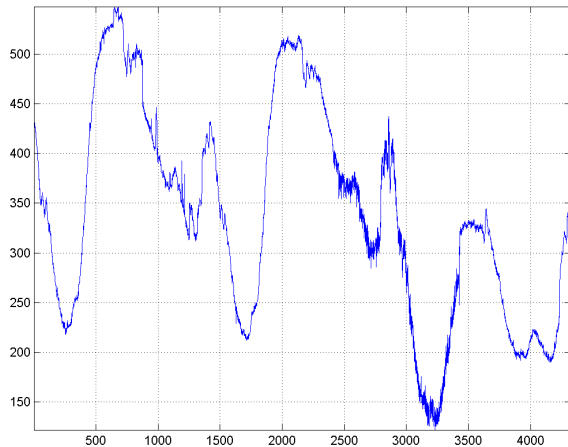
The data do not exhibit arbitrary changes from sample to sample. Neighboring samples exhibit correlation. A relatively low (high) value at index (sample) n is associated with a relatively low (high) value at index $n-1$ and $n+1$. This implies that if you have only the odd or even samples from the data, 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.
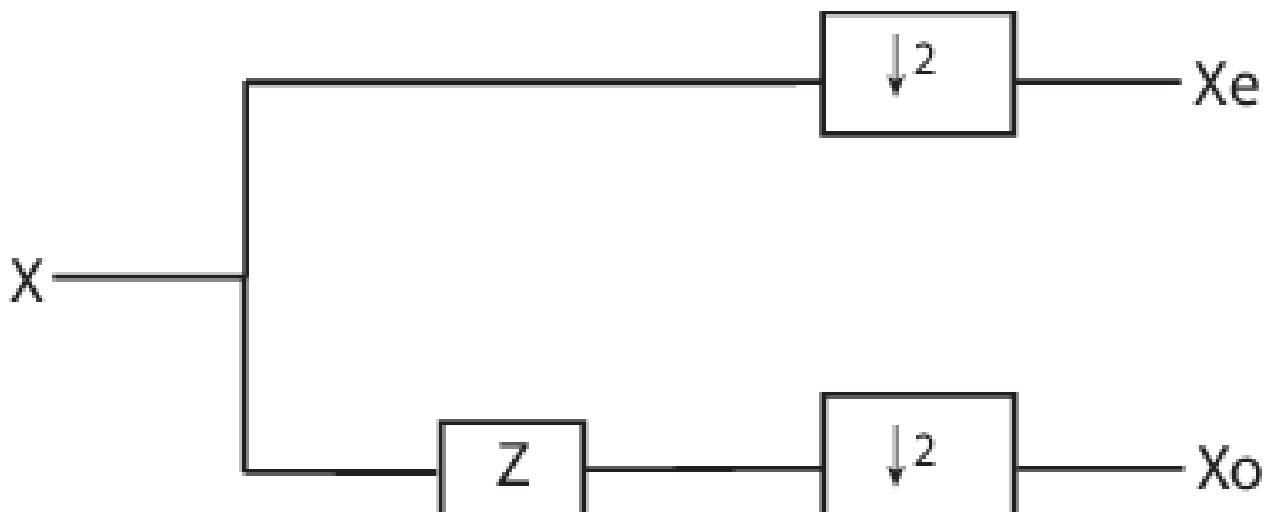
where Z denotes the unit advance operator and the downward arrow with the number 2 represents downsampling by two. In the language of lifting, the operation of separating an input signal into even and odd components is known as the *split* operation, or the *lazy wavelet*.

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 1-21. 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.
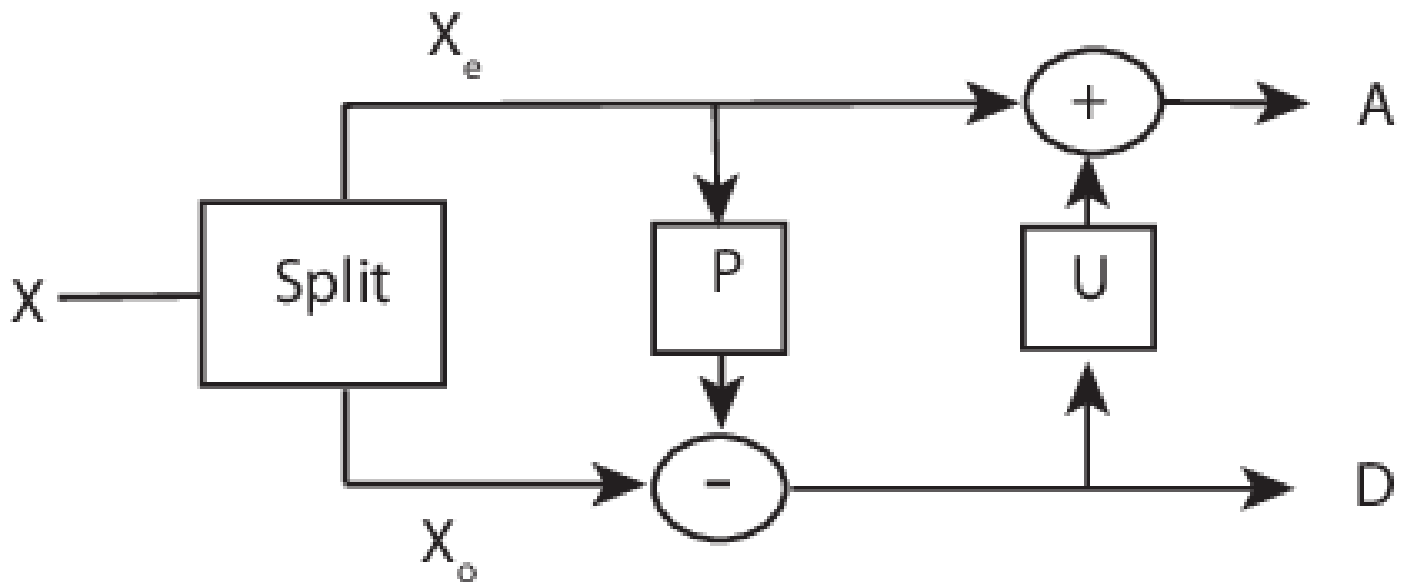
Haar Wavelet Via Lifting

Using the operations defined in "Split, Predict, and Update" on page 1-22, 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.

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

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 1-23. 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

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
lwt	1-D lifting wavelet transform
ilwt	Inverse 1-D lifting wavelet transform
lwtcoef	Extract or reconstruct 1-D LWT wavelet coefficients

Function Name	Description
lwt2	2-D lifting wavelet transform
ilwt2	Inverse 2-D lifting wavelet transform
lwtcoef2	Extract or reconstruct 2-D LWT wavelet coefficients

Laurent Polynomials and Matrices

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

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

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

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');
```
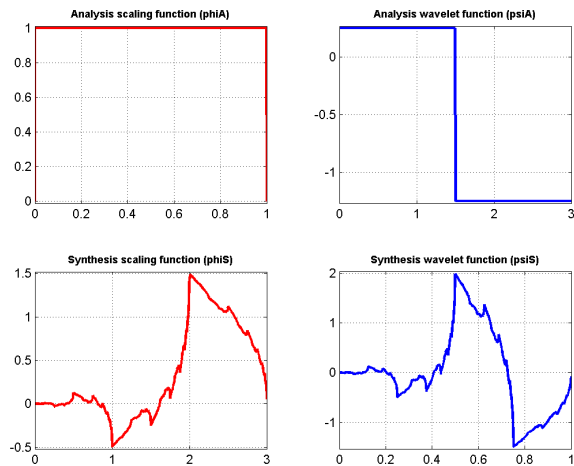
Integer-to-Integer Wavelet Transform

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.
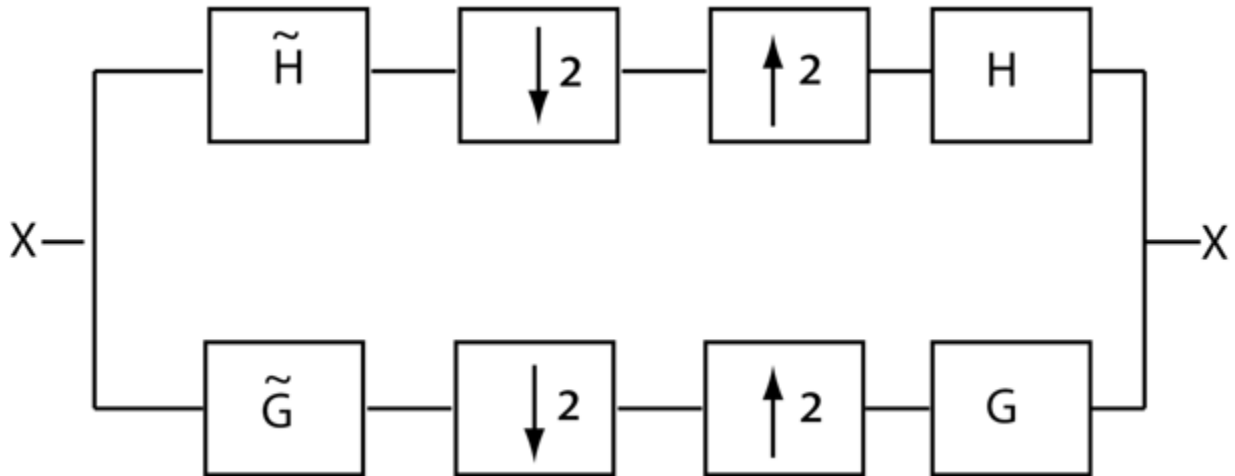
Let $\tilde{G}$ and $\tilde{H}$ denote the lowpass and highpass analysis filters and G and H denote the corresponding lowpass and highpass synthesis filters. A two-channel critically sampled filter bank filters the input signal using a lowpass and highpass filter. The subband outputs of the filters are 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')
```

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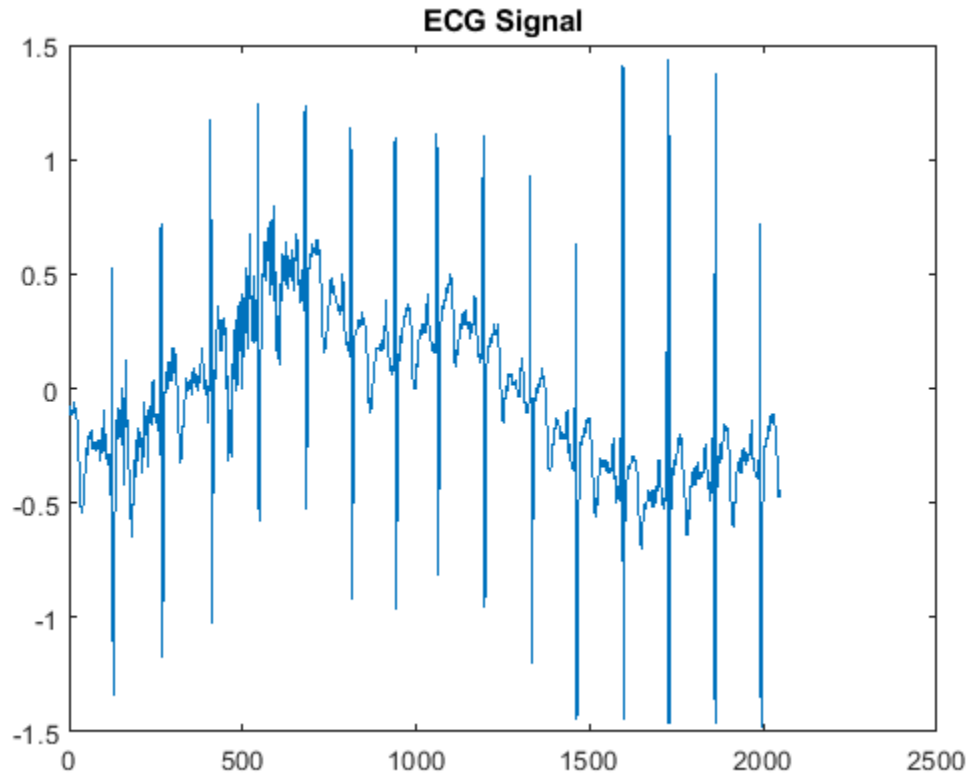
Obtain the lowpass (scaling) and highpass (wavelet) analysis and synthesis filters.

```
[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
```

Original ECG Waveform

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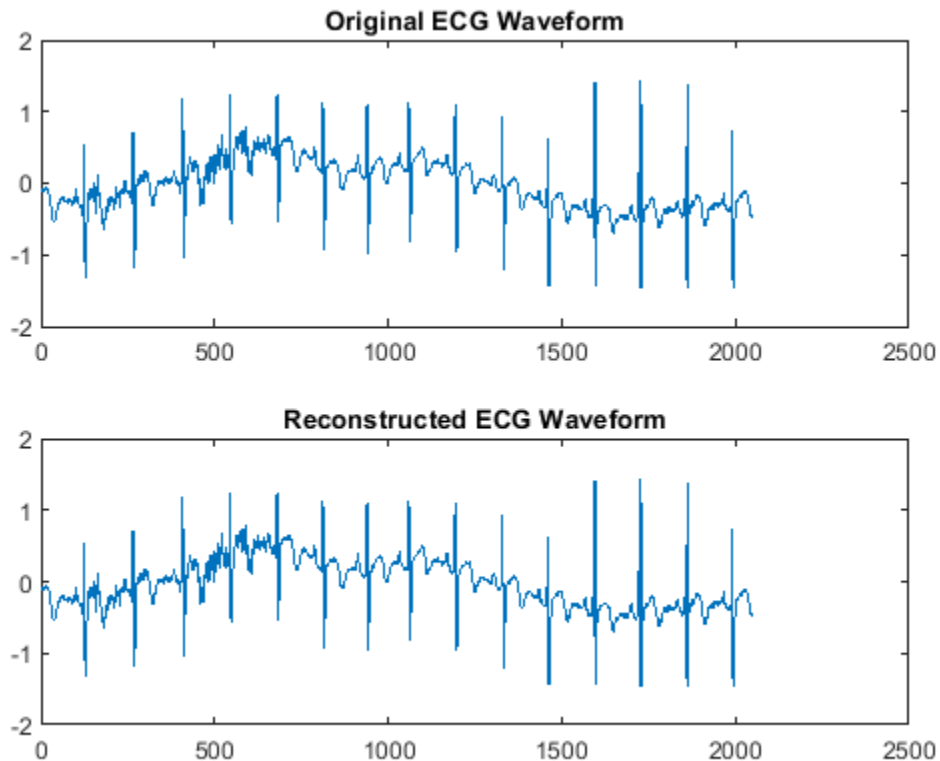
The analysis and synthesis filters for the 'db2' wavelet are just time reverses of each other. You can see this by comparing the following.

```
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)
```

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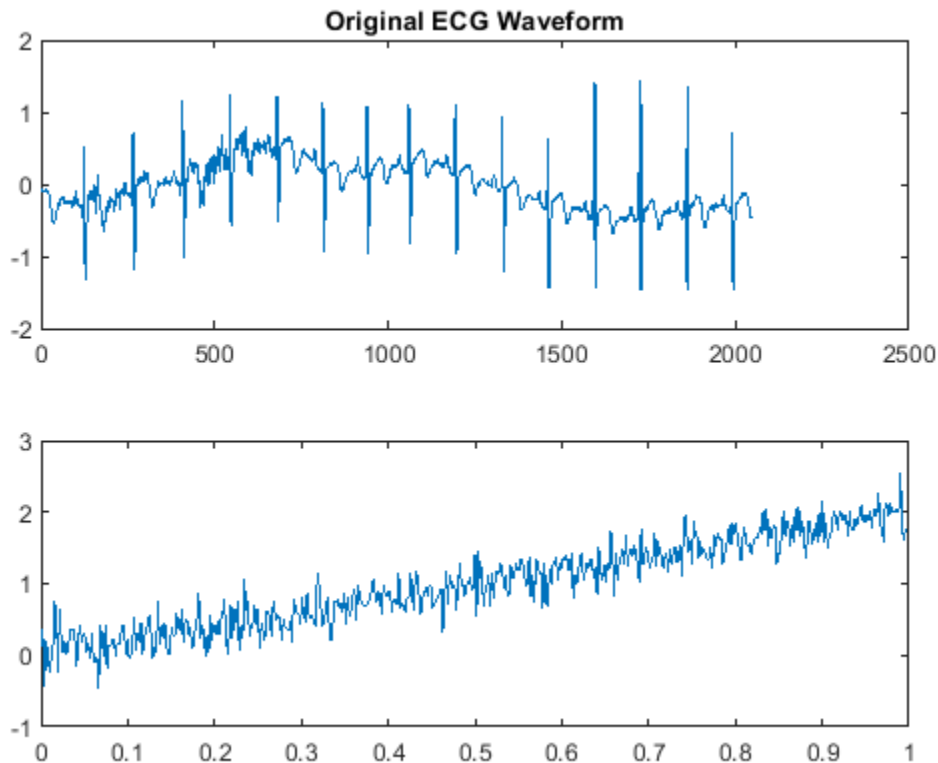
A linear trend is a polynomial of degree 1. Therefore, a wavelet with two vanishing moments removes this polynomial. The linear trend is preserved in the scaling coefficients and the wavelet coefficients can be regarded as consisting of only noise. Obtain the level-one DWT of the signal with the 'db2'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

Scaling Coefficients

Wavelet Coefficients
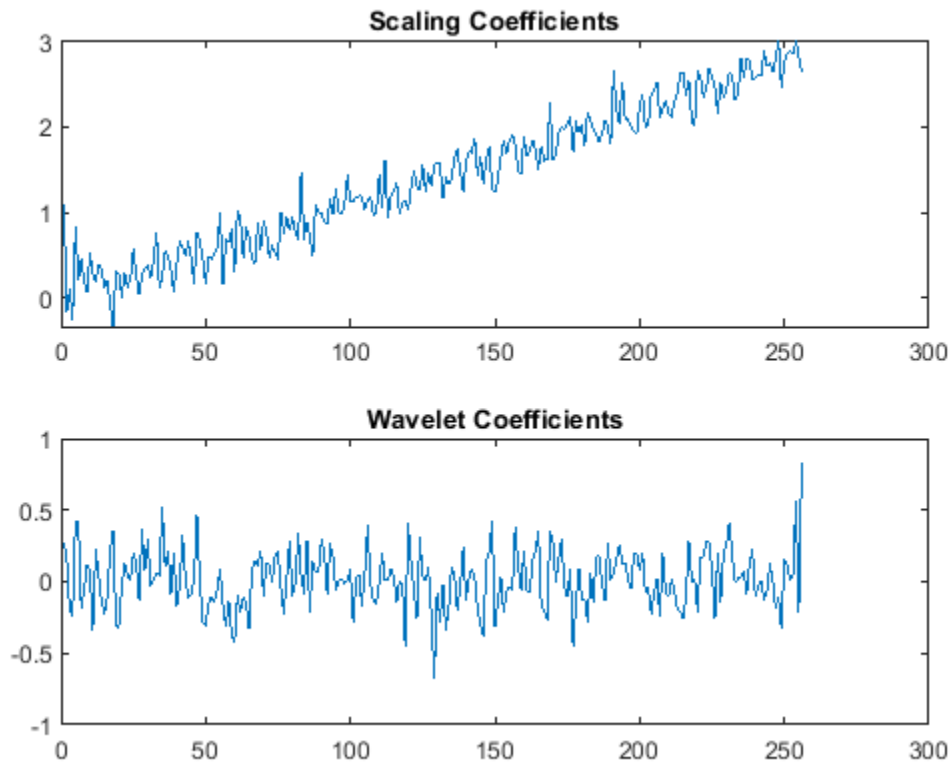
You can use `dwt` and `idwt` to implement a two-channel orthogonal filter bank, but it is often more convenient to implement a multi-level two-channel filter bank using `wavedec`. The multi-level DWT iterates on the output of the lowpass (scaling) filter. In other words, the input to the second level 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.
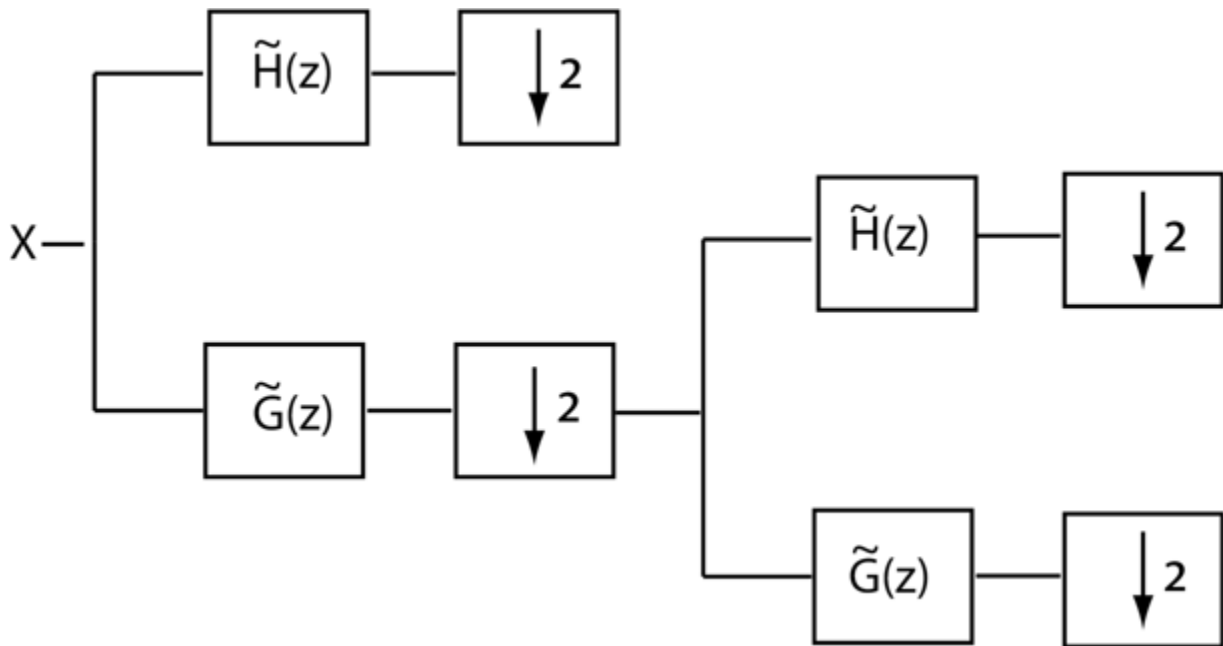
At each successive level, the number of scaling and wavelet coefficients is downsampled by two so the total number of coefficients are preserved. Obtain the level three DWT of the ECG signal using the 'sym4' orthogonal filter bank.

```
[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;
```
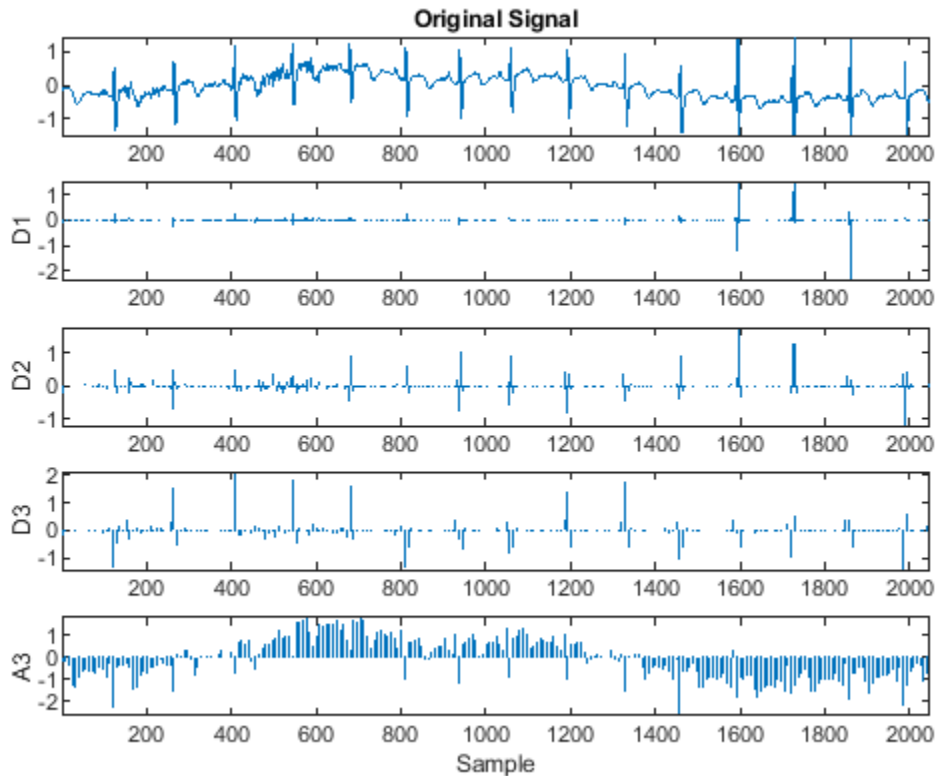

Because the critically sampled wavelet filter bank downsamples the data at each level, the analysis must stop when you have only one coefficient left. In the case of the ECG signal with 2048 samples, this must occur when $L = \log_2 2048$.

```
[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');
```
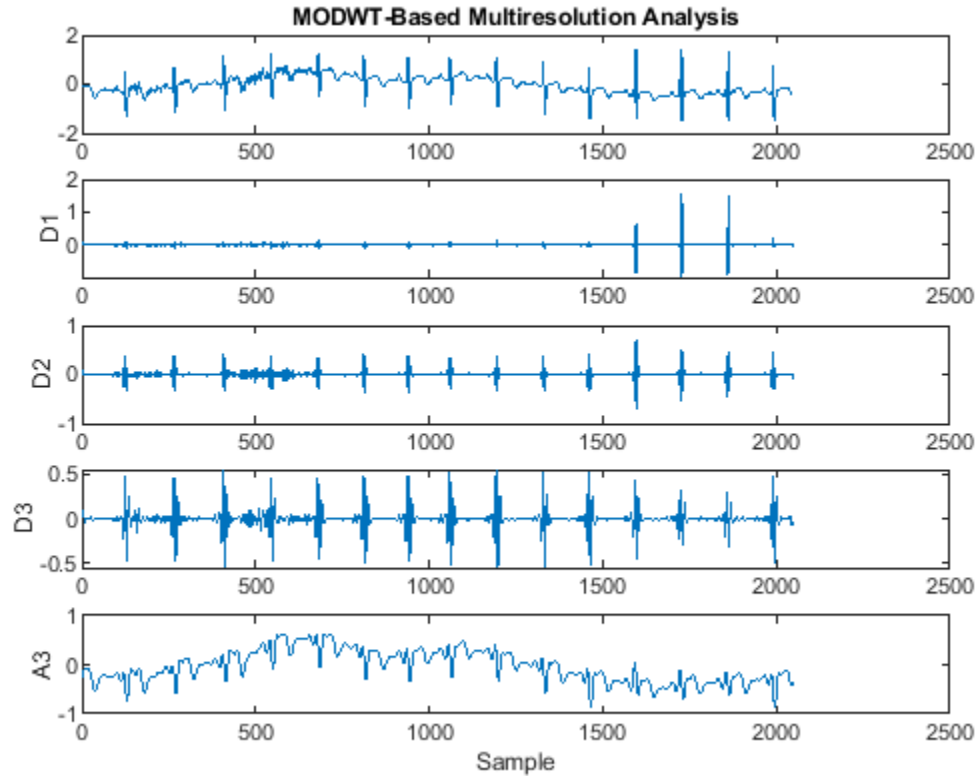

In a biorthogonal filter bank, the synthesis filters are not simply time-reversed versions of the analysis filters. The family of biorthogonal spline wavelet filters are an example of such filter banks.

```
[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 =

   4.4409e-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 Signal Processing Toolbox™, 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');
```
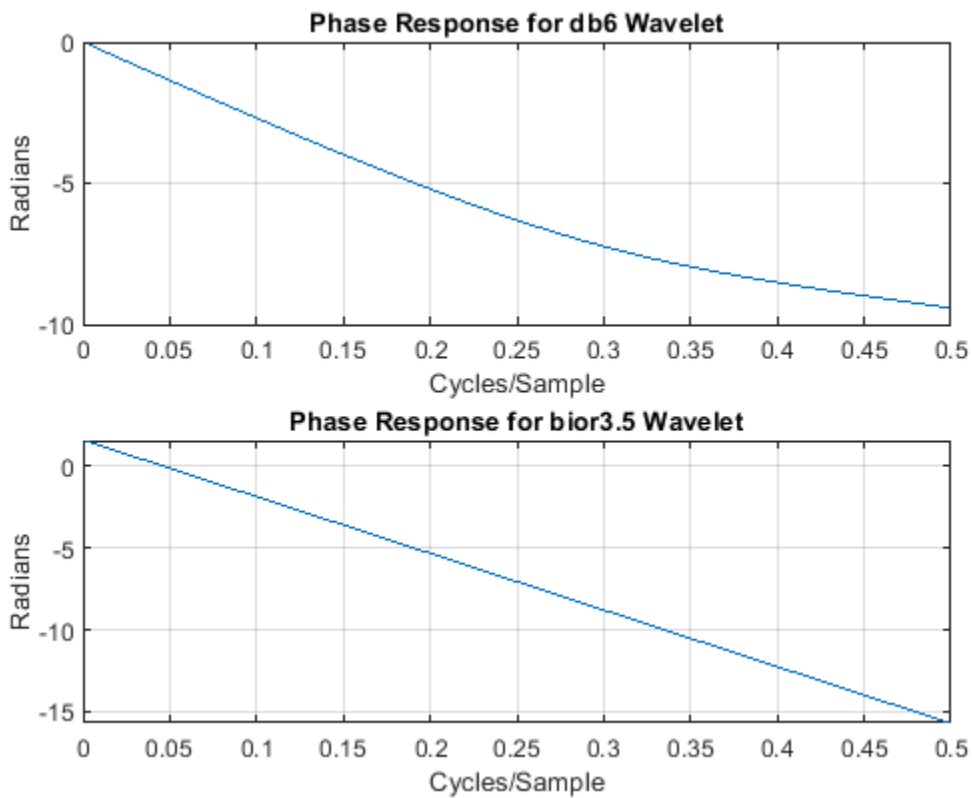

Set the dwtmode back to the original setting.

```
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');
```
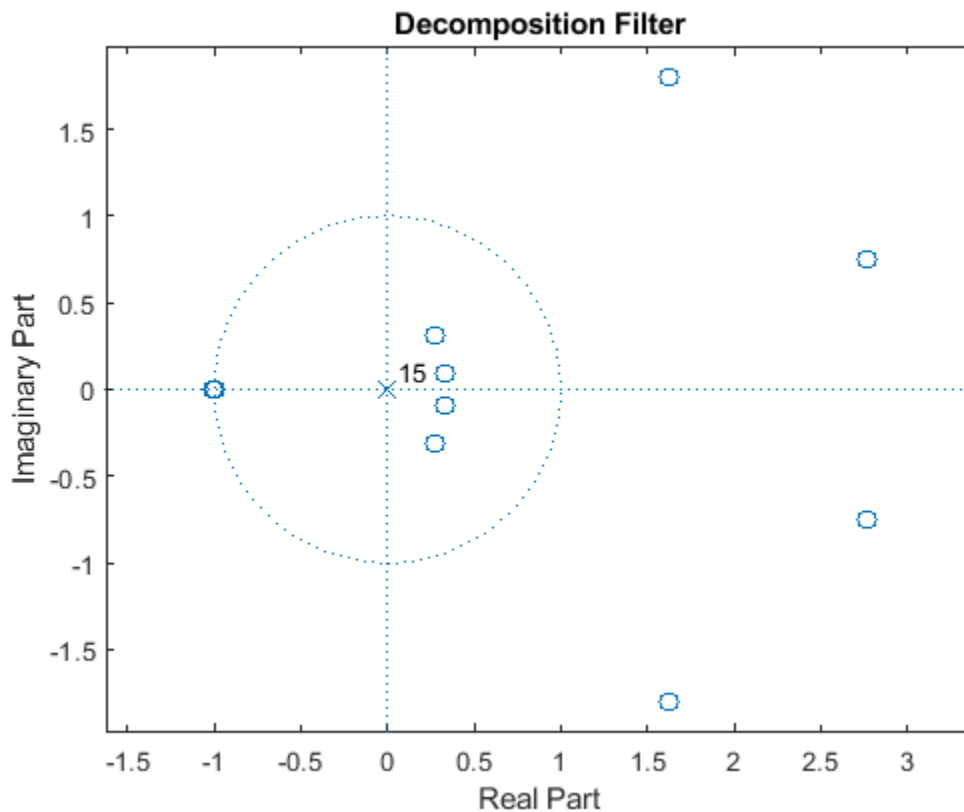


```
figure;
zplane(LoR); title('Reconstruction Filter');
```

Reconstruction Filter
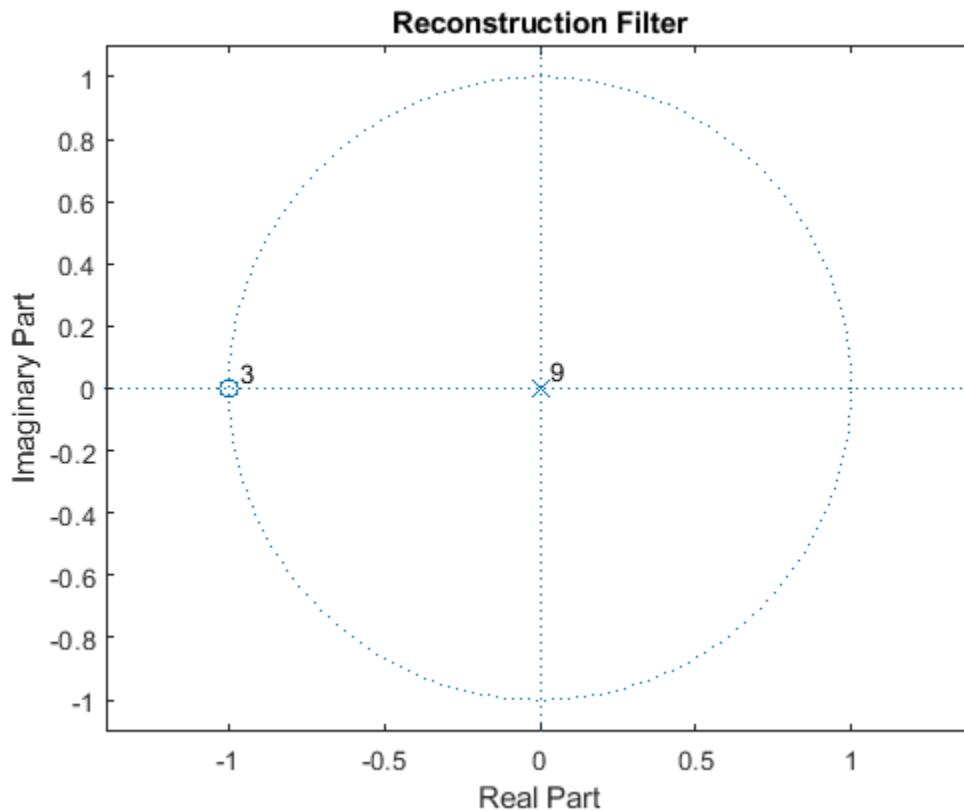
If you zoom in on the region around -1+i0, you find there are 7 zeros in the decomposition filter and 3 zeros in the reconstruction filter. This has important consequences for the smoothness of the corresponding scaling functions and wavelets. For biorthogonal wavelets, the more zeros at -1+i0 in the 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');
```
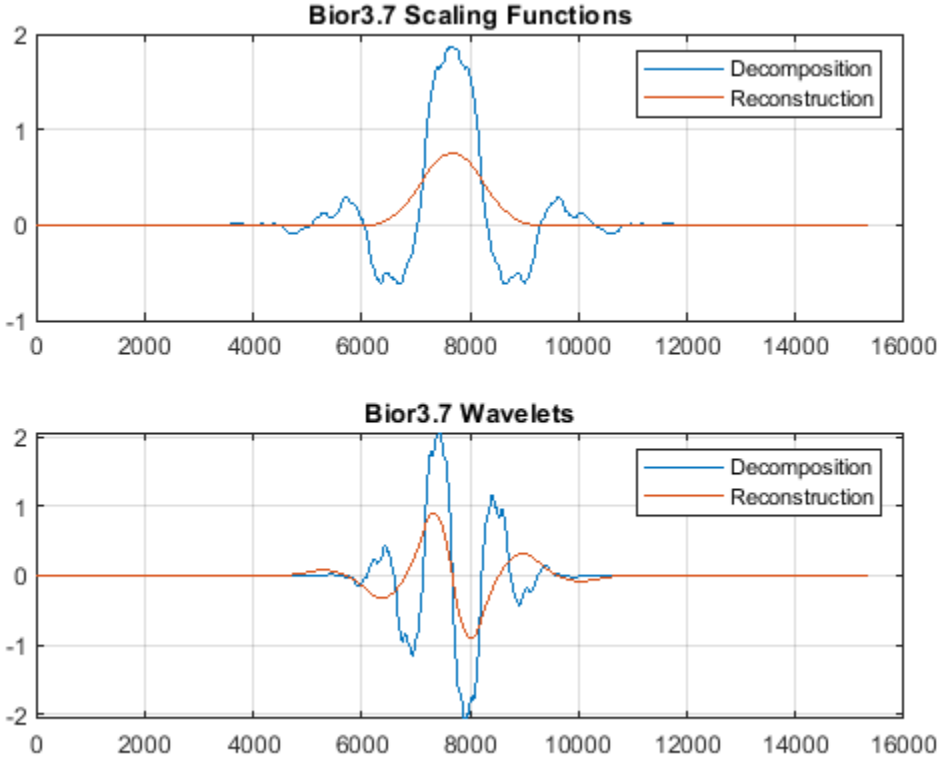
Because there are more than twice the number of zeros at -1+i0 for the lowpass decomposition filter, the dual (reconstruction) scaling function and wavelet are much smoother than the analysis (decomposition) scaling function and wavelet.

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.
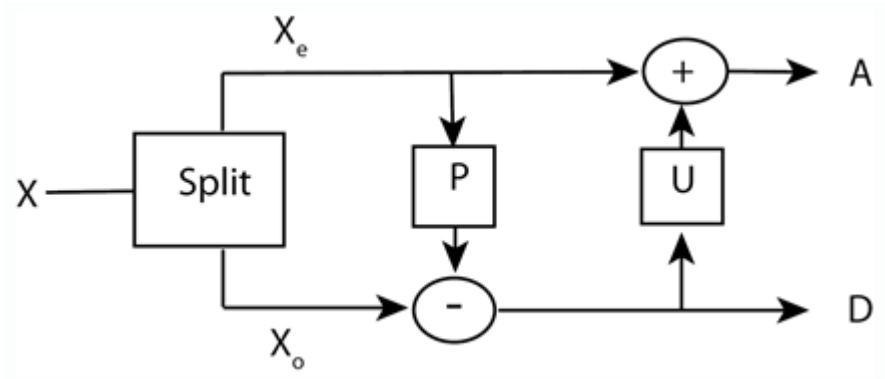
The first step in lifting is simply to split the signal into its even- and odd-indexed samples. These are called polyphase components and that step in the lifting process is often referred to as the "lazy" lifting step because you really are not doing that much work. You can do this in MATLAB by creating 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 Du 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=4×3 cell array
    {'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');
```
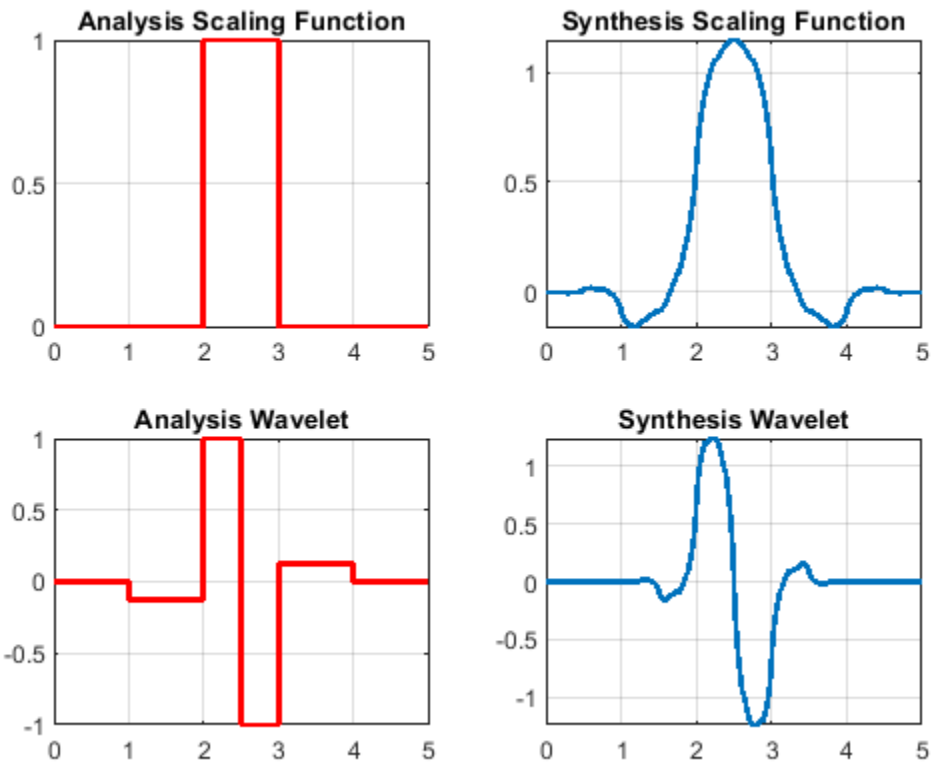
If you plot the analysis and synthesis scaling functions and wavelets for the 'bior1.3' wavelet, you see that lifting the Haar wavelet as in the previous example has essentially provided the 'bior1.3' wavelet to within a change of sign on the synthesis wavelet.

```
[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')
```

Original Image

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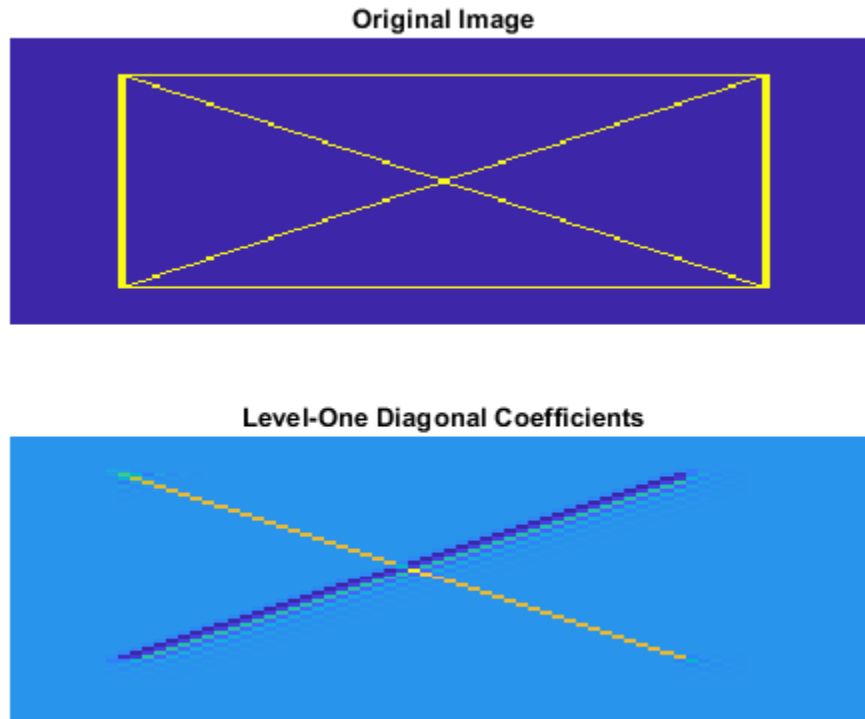
Finally, verify that the new filter satisfies the conditions for an orthogonal QMF pair. Obtain the scaling (lowpass) and wavelet (highpass) filters.

```
[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');
```
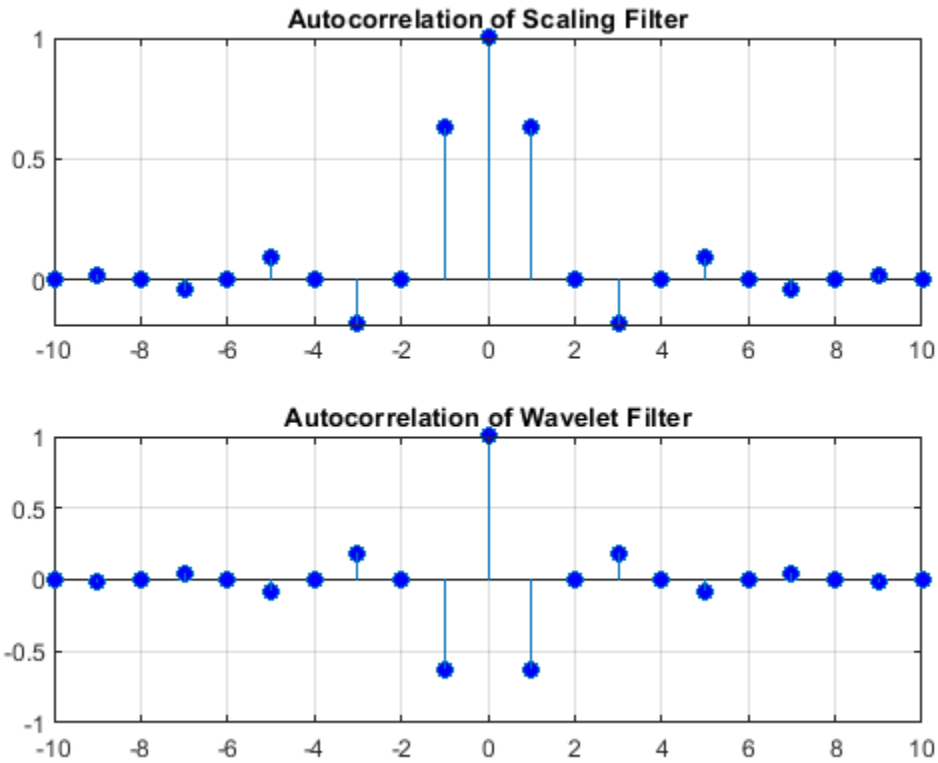

Note that the autocorrelation values in both plots is zero for nonzero even lags. Show that the cross-correlation of the scaling and wavelet filter is zero at all even lags.

```
[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 is: $|G_{2^m k \bmod N}|^2 + |H_{2^m k \bmod N}|^2 = 2$ for any m. Check this for the Beylkin(18) filter with $m = 0$.

```
N = numel(Lo);
LoDFT = fft(Lo);
HiDFT = fft(Hi);
k = 0:N-1;
m = 0;
sumDFTmags = abs(LoDFT(1+mod(2^m*k,N))).^2+abs(HiDFT(1+mod(2^m*k,N))).^2

sumDFTmags = 18×1

    2.0000
    2.0000
    2.0000
    2.0000
    2.0000
    2.0000
    2.0000
    2.0000
    2.0000
    2.0000
      ⋮
```

All the values are equal to 2 as expected. To understand why these filters are called quadrature mirror filters, visualize the squared-magnitude frequency responses of the scaling and wavelet filters.

```matlab
nfft = 512;
F = 0:1/nfft:1/2;
LoDFT = fft(Lo,nfft);
HiDFT = fft(Hi,nfft);
figure
plot(F,abs(LoDFT(1:nfft/2+1)).^2,'DisplayName','Scaling Filter');
hold on
plot(F,abs(HiDFT(1:nfft/2+1)).^2,'r','DisplayName','Wavelet Filter');
xlabel('Frequency'); ylabel('Squared Magnitude')
title('Beylkin(18) QMF Filter Pair')
grid on
plot([1/4 1/4], [0 2],'k','HandleVisibility','off');
legend show;
```
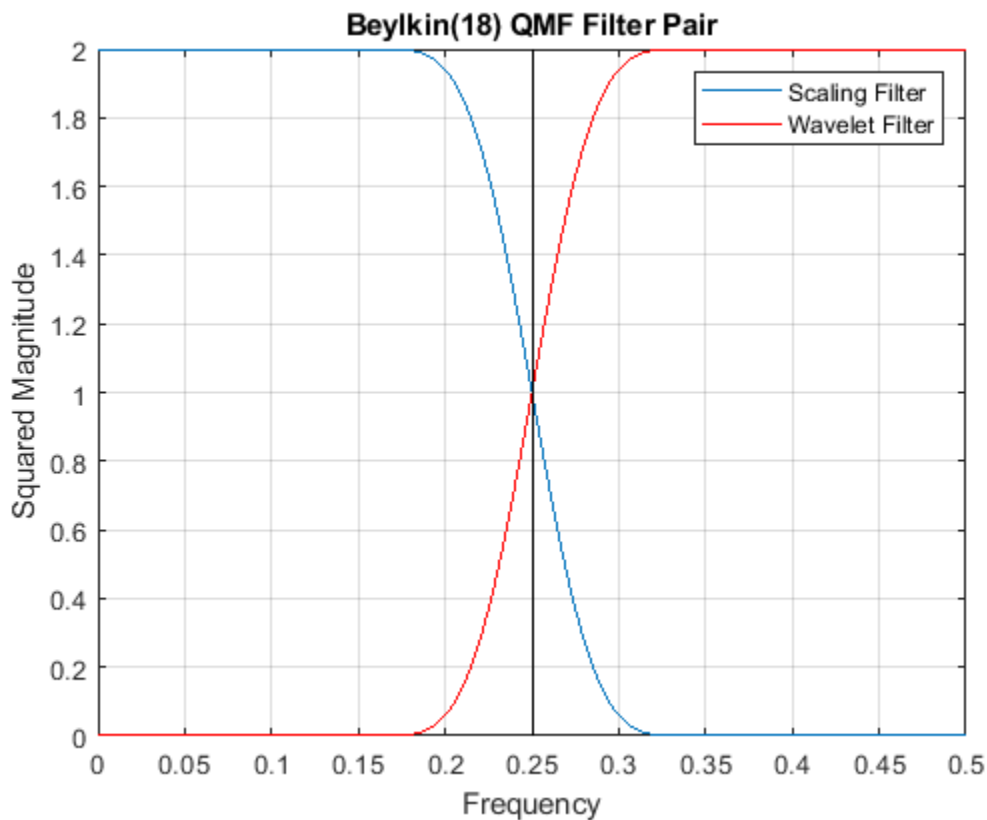

Note the magnitude responses are symmetric, or mirror images, of each other around the quadrature frequency of 1/4.

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

```matlab
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->->fbsp         '
    '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)
```
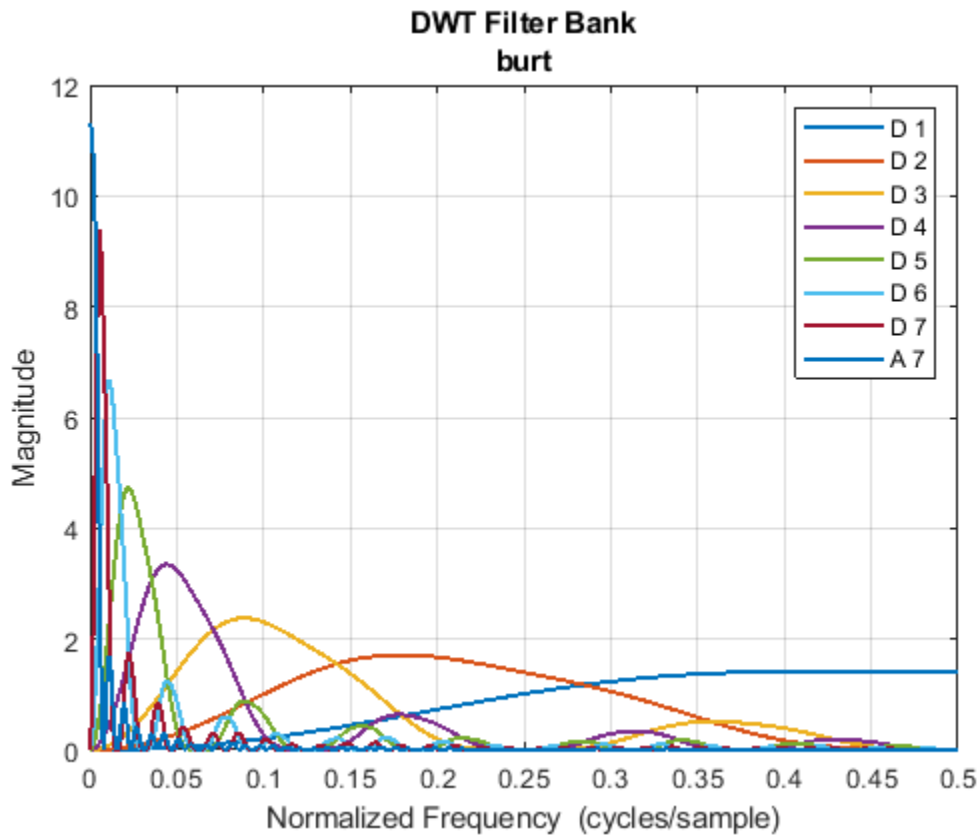

Obtain the wavelet and scaling functions of the filter bank. Plot the wavelet and scaling functions at the coarsest scale.

```
[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')
```
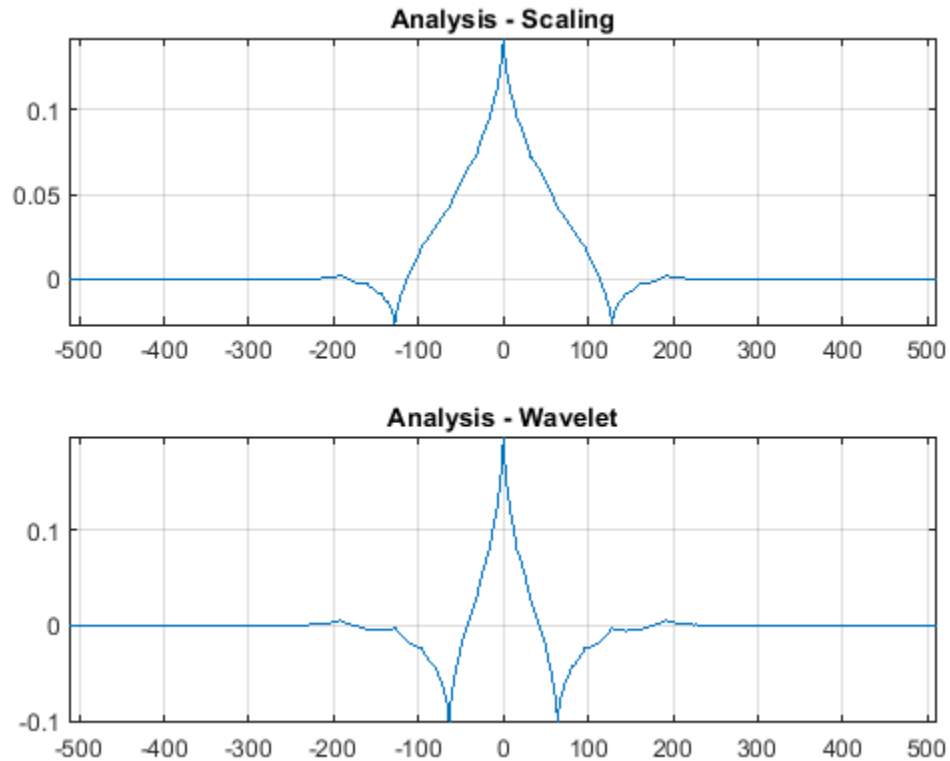
Create a synthesis DWT filter bank using the `burt` wavelet. Compute the framebounds.

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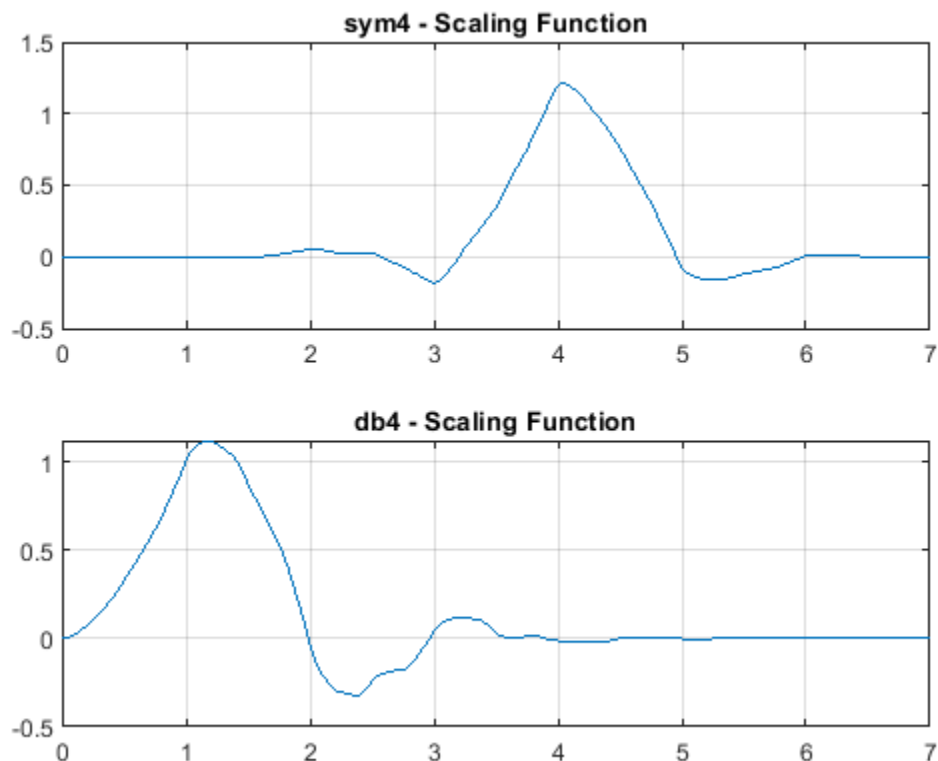
Generate the filters associated with the order 4 symlet and Daubechies wavelets.

```
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')
```
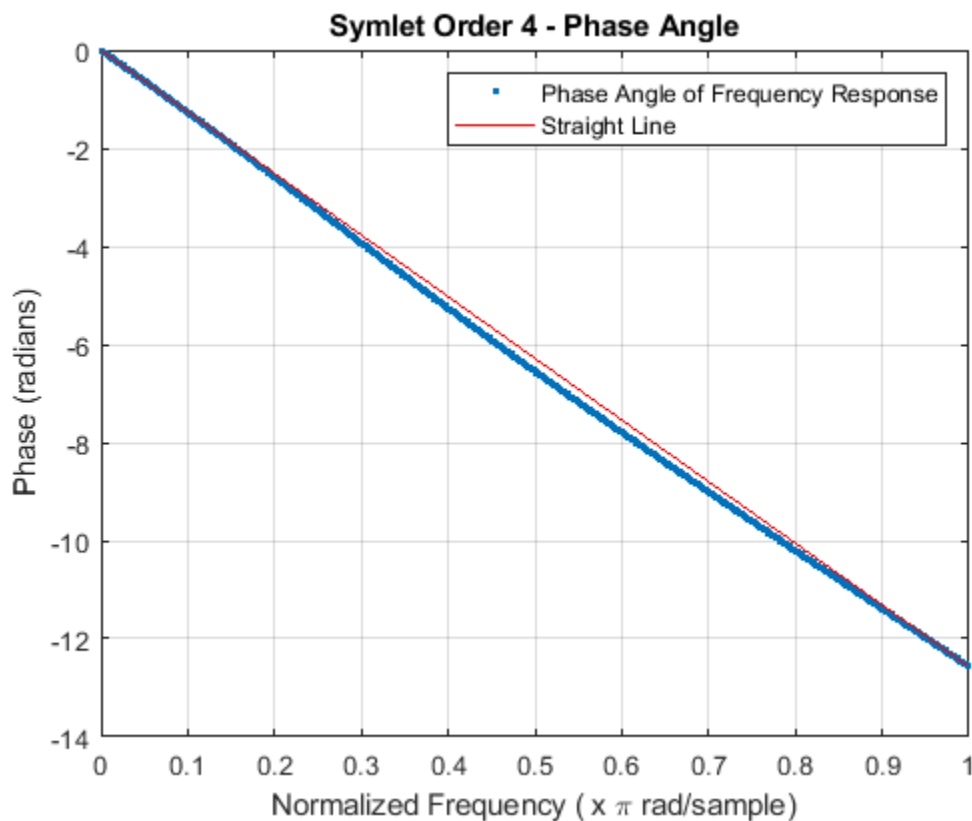


```
figure
plot(w_db/pi,h_db_u,'.')
hold on
plot(w_db([1 end])/pi,h_db_u([1 end]),'r')
grid on
xlabel('Normalized Frequency ( x \pi rad/sample)')
ylabel('Phase (radians)')
```

```
legend('Phase Angle of Frequency Response','Straight Line')
title('Daubechies Order 4 - Phase Angle')
```
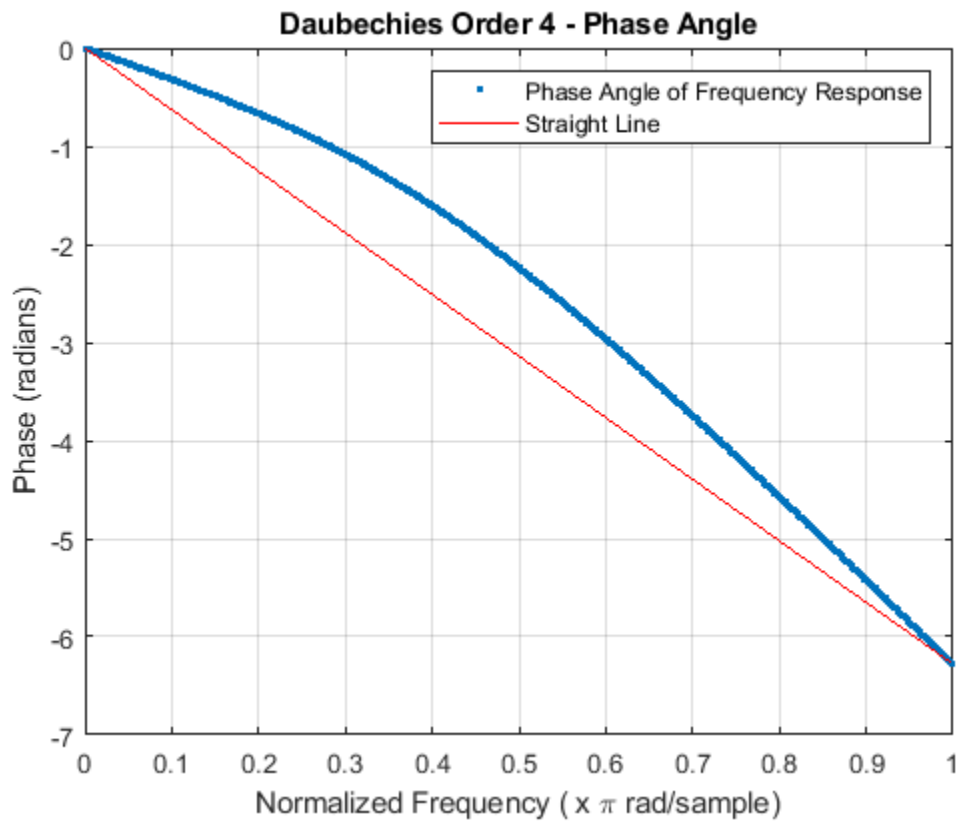

The sym4 and db4 wavelets are not symmetric, but the biorthogonal wavelet is. Plot the scaling function associated with the bior3.5 wavelet. Compute the frequency response of the synthesis scaling filter for the wavelet and verify that it has linear phase.

```
[~,~,phi_bior_r,~,xval_bior]=wavefun('bior3.5',10);
figure
plot(xval_bior,phi_bior_r)
title('bior3.5 - Scaling Function')
grid on
```
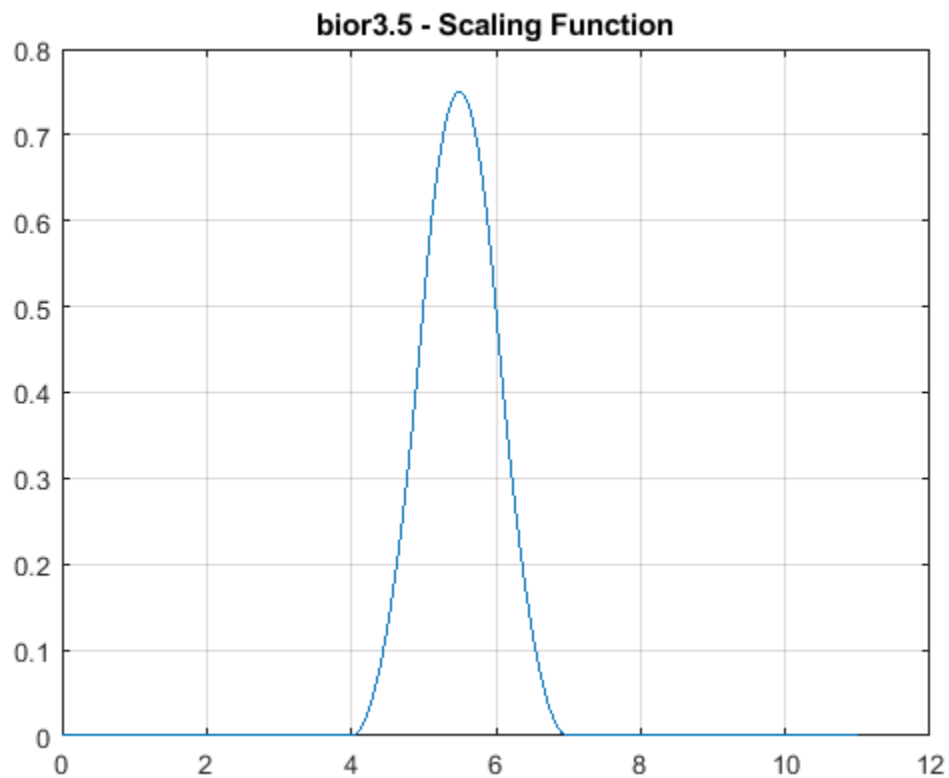
```
[LoD_bior,HiD_bior,LoR_bior,HiR_bior] = wfilters('bior3.5');
[h_bior,w_bior] = freqz(LoR_bior);
h_bior_u = unwrap(angle(h_bior));
figure
plot(w_bior/pi,h_bior_u,'.')
hold on
plot(w_bior([1 end])/pi,h_bior_u([1 end]),'r')
grid on
xlabel('Normalized Frequency ( x \pi rad/sample)')
ylabel('Phase (radians)')
legend('Phase Angle of Frequency Response','Straight Line')
title('Biorthogonal 3.5 - Phase Angle')
```
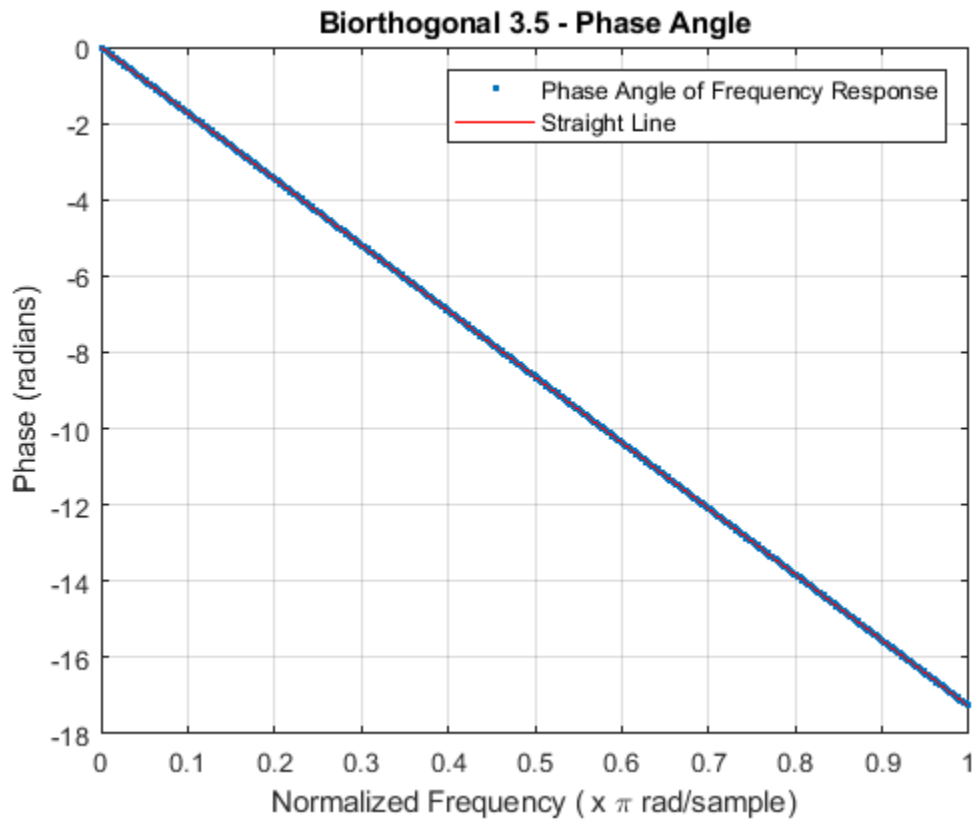
See Also

dbaux | symaux

Continuous Wavelet Analysis

1-D Continuous Wavelet Analysis

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" on page 10-64 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.
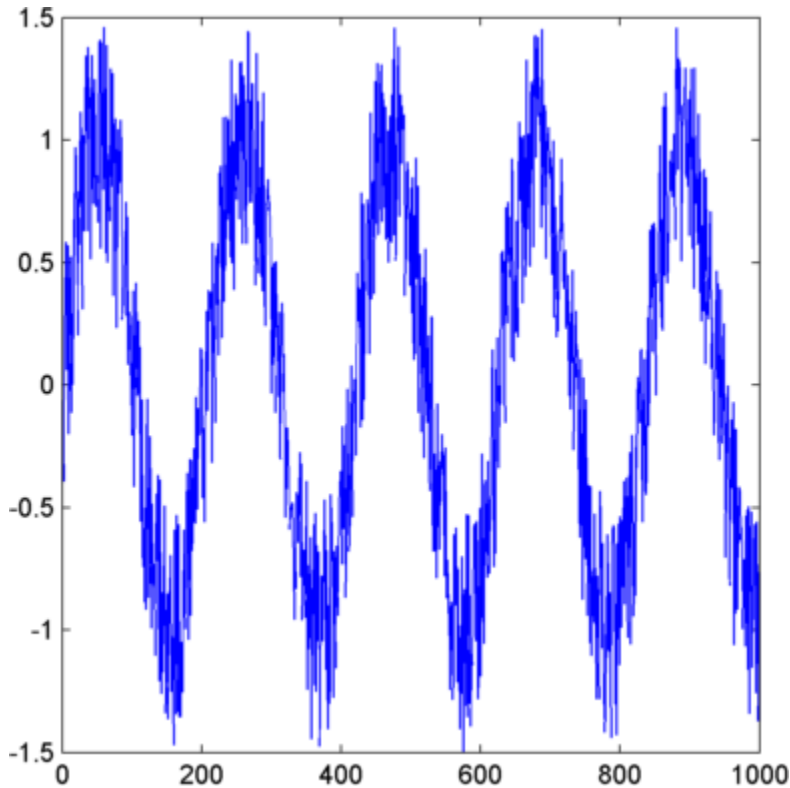
1   Load a signal.

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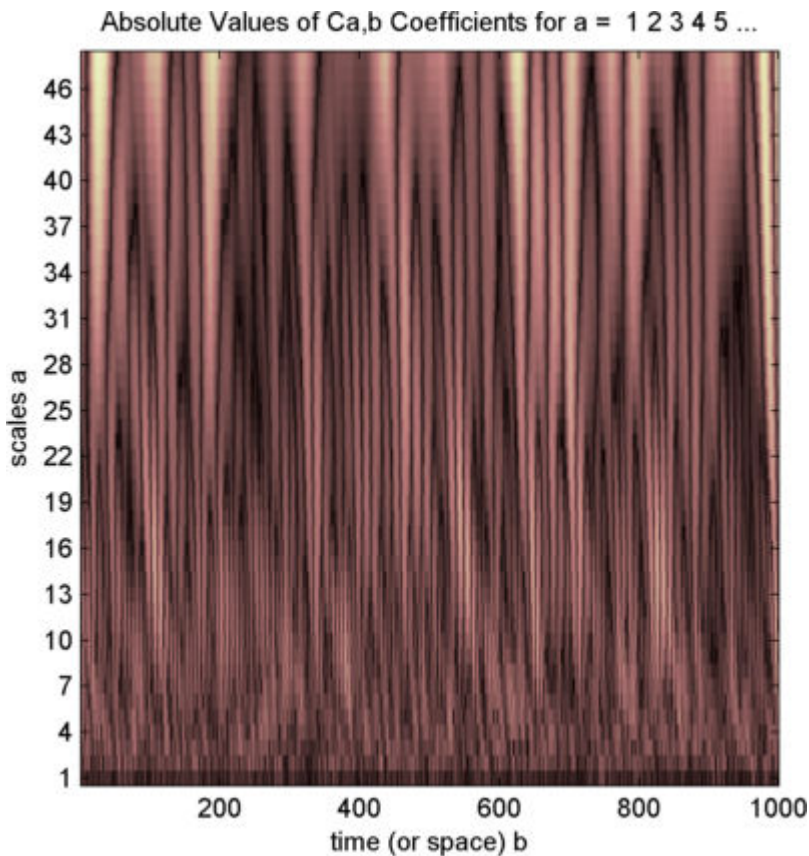
Of course, coefficient plots generated from the command line can be manipulated using ordinary MATLAB graphics commands.

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:

Absolute Values of Ca,b Coefficients for a = 2 4 6 8 10 ...
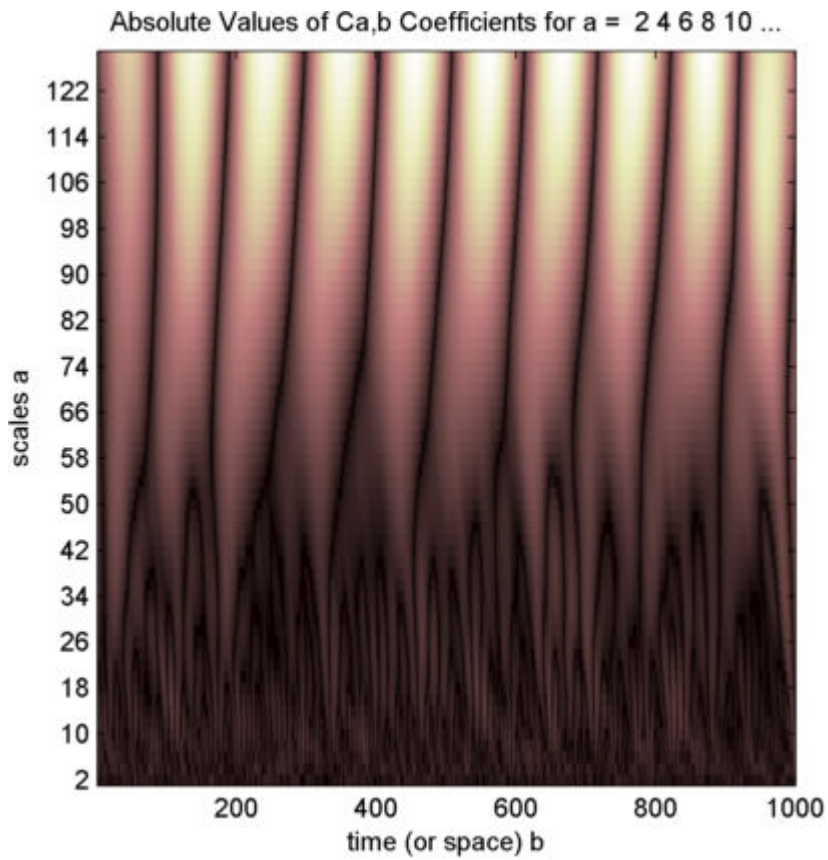
This plot gives a clearer picture of what's happening with the signal, highlighting the periodicity.

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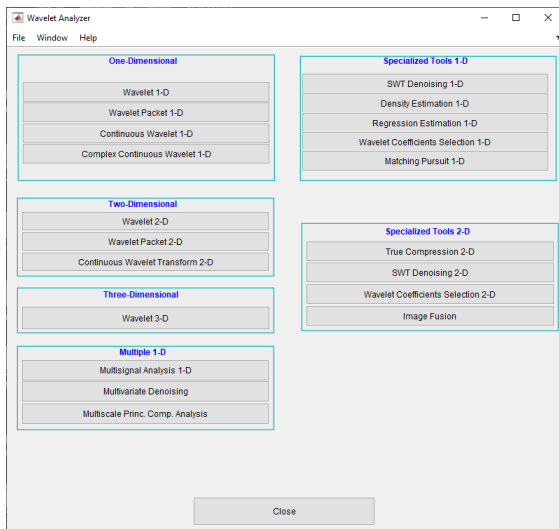
Click the **Continuous Wavelet 1-D** menu item.

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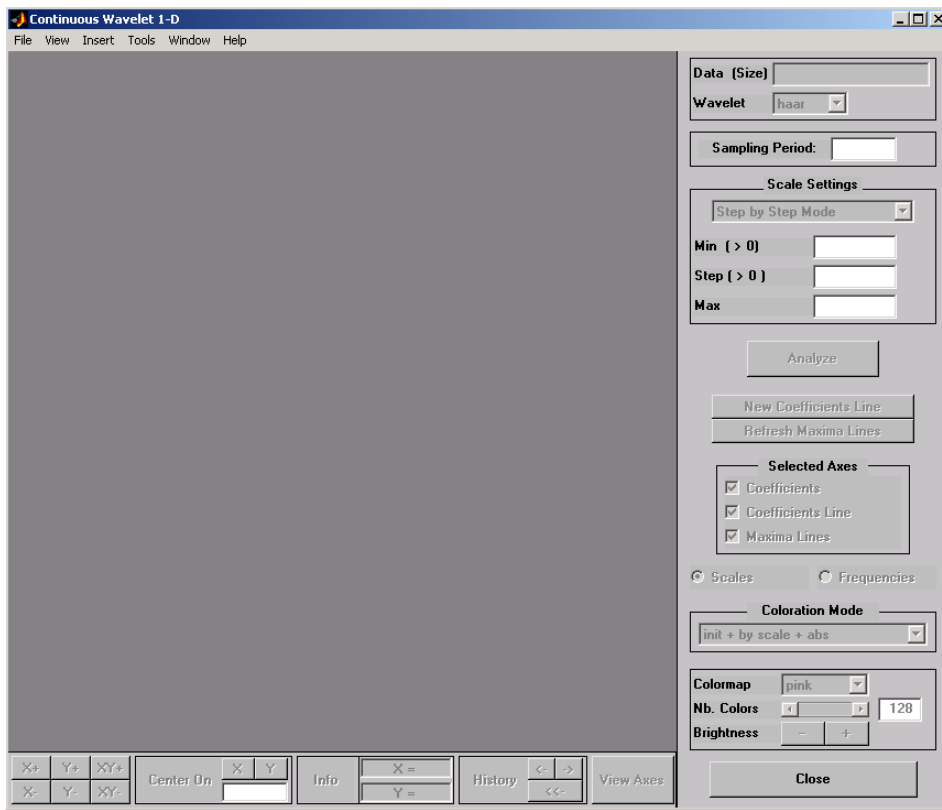
2 Load a signal.

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.
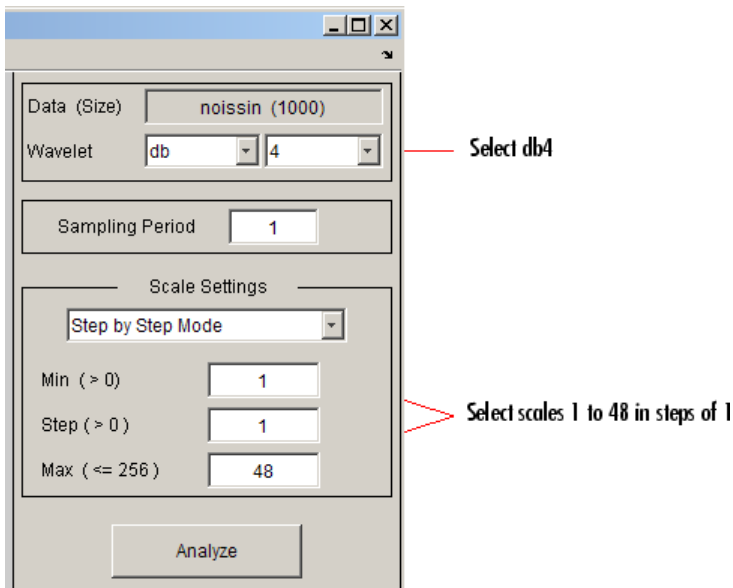
4 Click the **Analyze** button.

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.
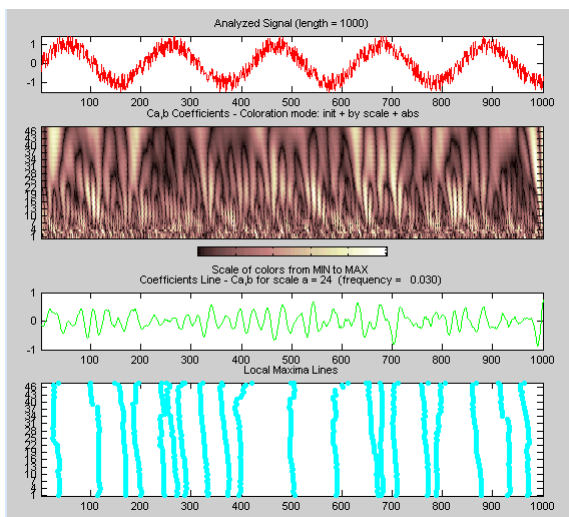
5 View Wavelet Coefficients Line.

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.

Coefficients Line - Ca,b for scale a = 40 (frequency = 0.018)
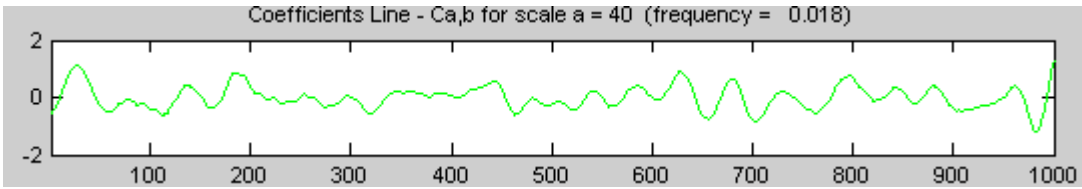
6 View Maxima Line.

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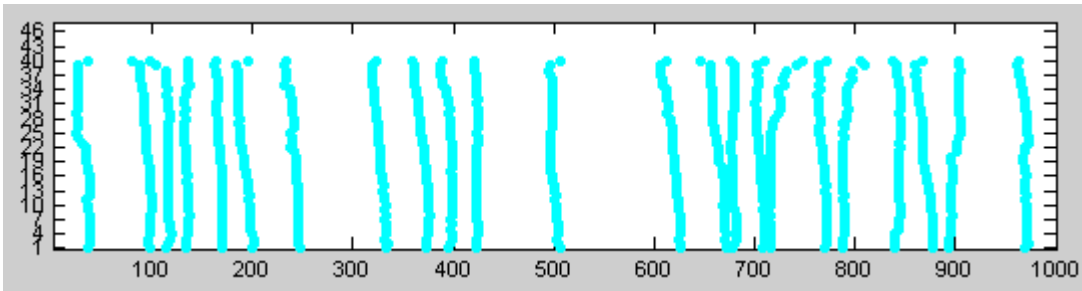
Hold down the right mouse button over the coefficients plot. The position of the mouse is given by the **Info** frame (located at the bottom of the screen) in terms of location (**X**) and scale (**Sca**).
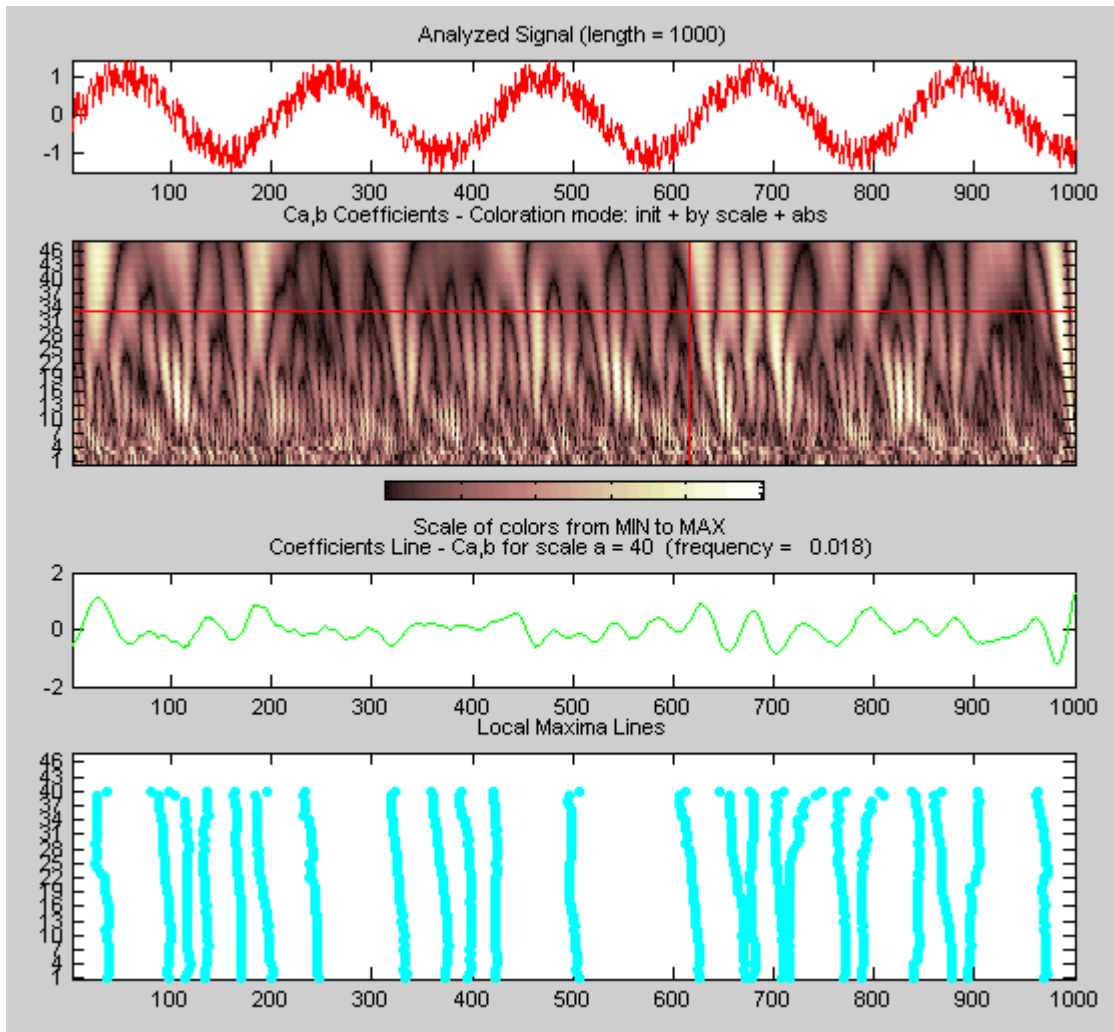
7 Switch from scale to Pseudo-Frequency Information.

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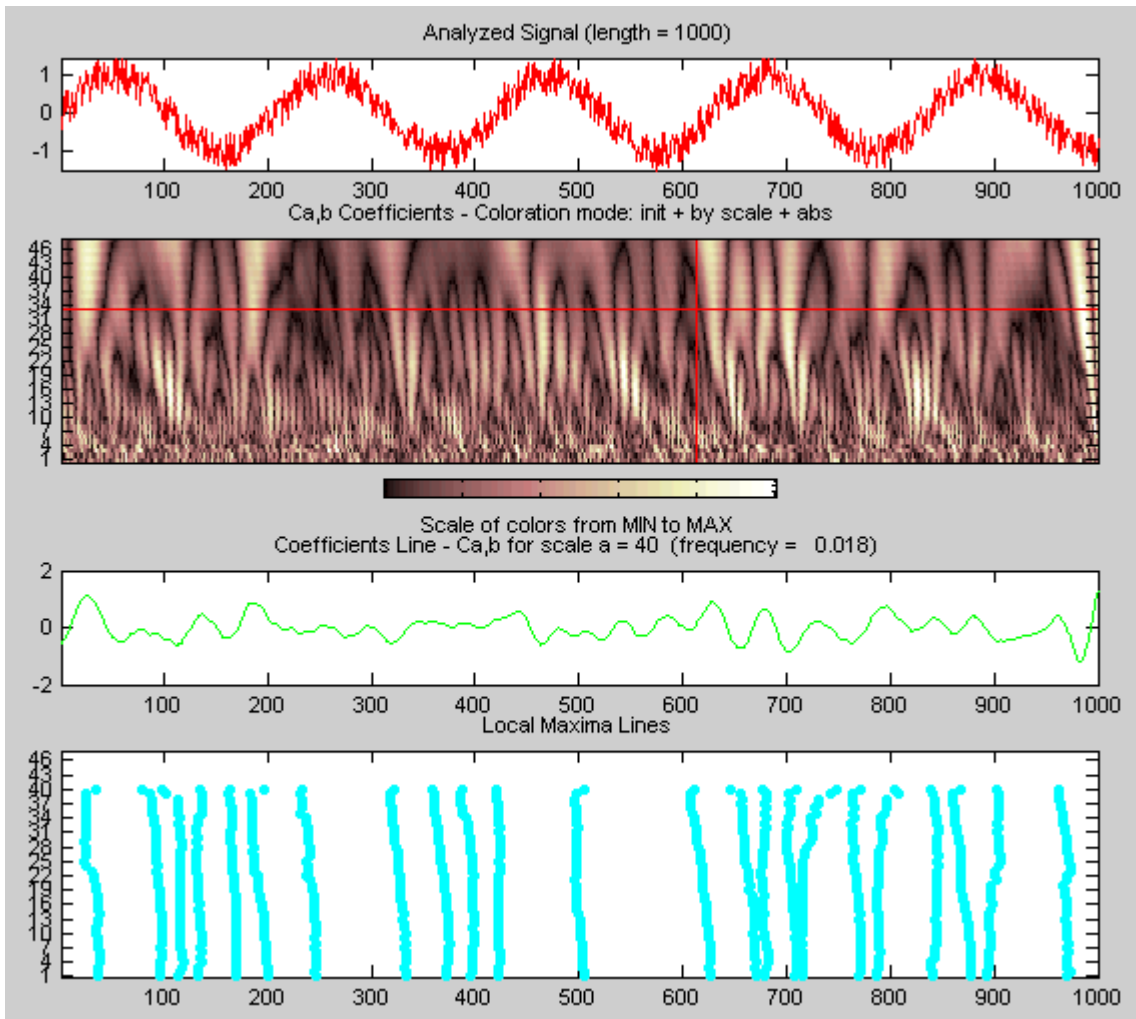
This facility allows you to interpret scale in terms of an associated pseudo-frequency, which depends on the wavelet and the sampling period..

8 Deselect the last two plots using the check boxes in the **Selected Axes** frame.
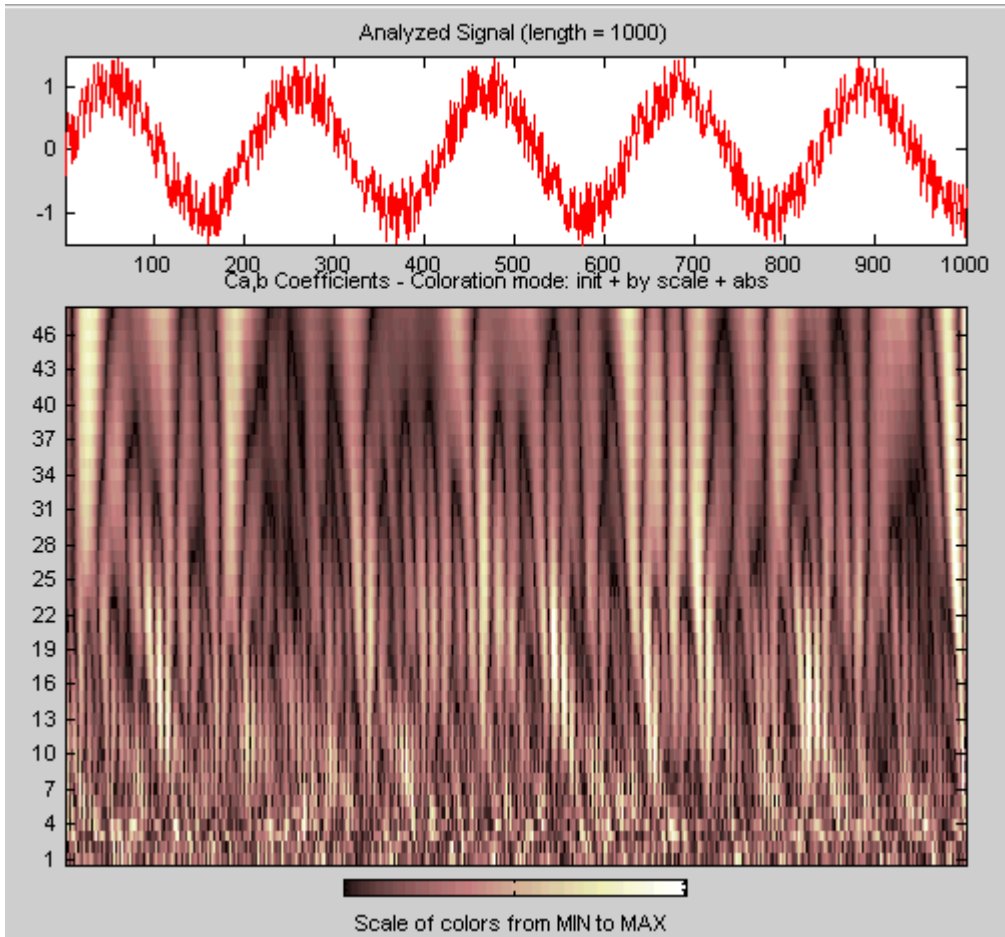
Analyzed Signal (length = 1000)

Ca,b Coefficients - Coloration mode: init + by scale + abs

Scale of colors from MIN to MAX

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.

Analyzed Signal (length = 1000)

ients - Coloration mode: init + by scale + abs
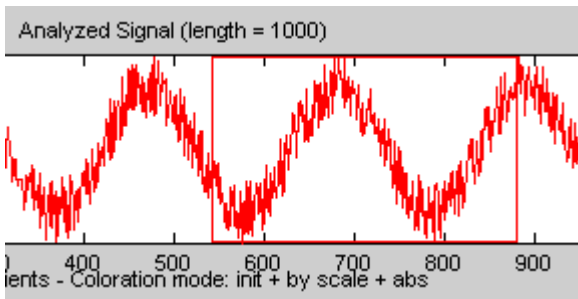
10 Click the **X+** button (located at the bottom of the screen) to zoom horizontally only.
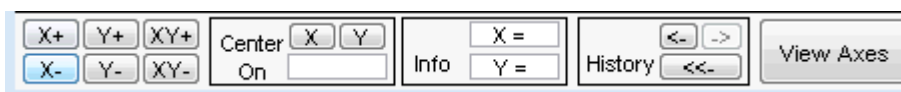
The **Continuous Wavelet 1-D** tool enlarges the displayed signal and coefficients plot (for more information on zooming, see "Connection of Plots" on page A-2 in the *Wavelet Toolbox User's Guide*).
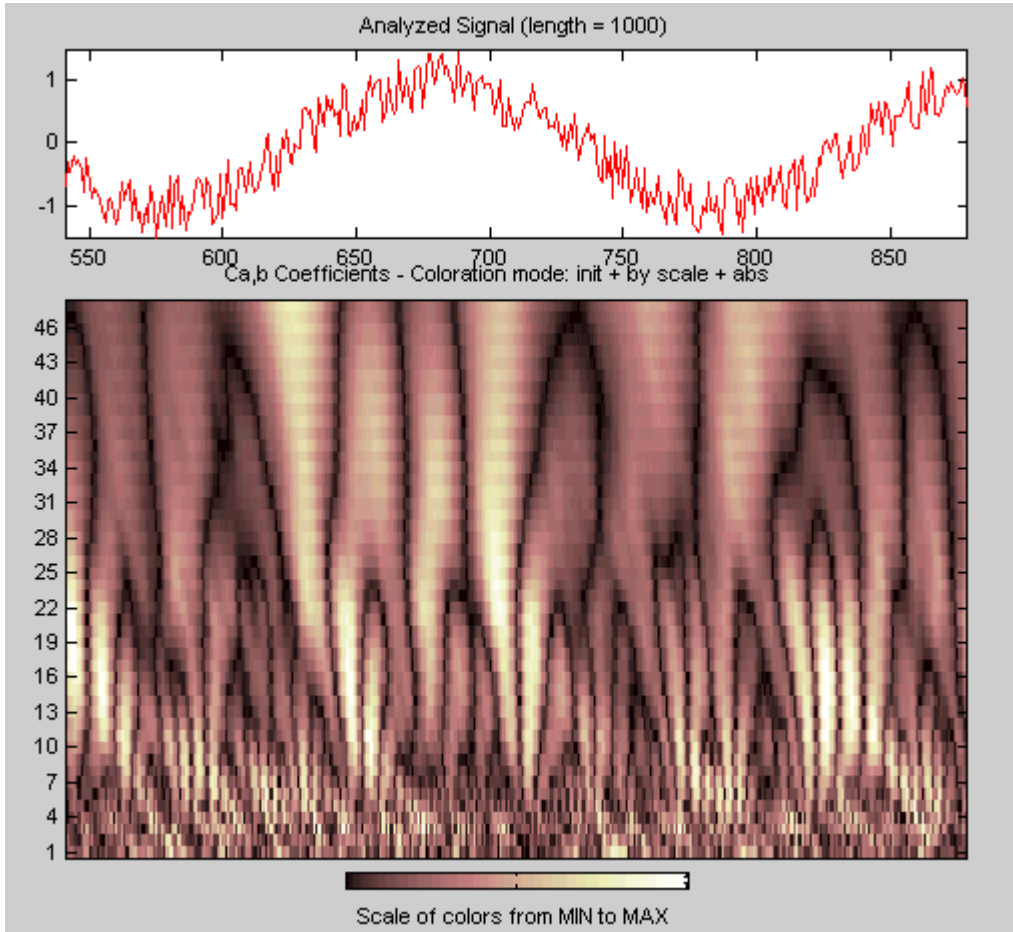
As with the command line analysis on the preceding pages, you can change the scales or the analyzing wavelet and repeat the analysis. To do this, just edit the necessary fields and click the **Analyze** button.

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-6.

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
coefs	64x2188	1120256	double array
scales	1x64	512	double array
wname	1x4	8	char array

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

Morse Wavelets

In this section...

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 $\gamma = 0.25$. See "Generalized Morse and Analytic Morlet Wavelets".

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 $\left(\dfrac{P^2}{\gamma}\right)^{\frac{1}{\gamma}}$.

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.

Figure with four subplots. Top-left titled "$\gamma = 3$ and $P^2 = 60$" (Time vs amplitude). Top-right titled "$\gamma = 3$ and $P^2 = 60$" (Time vs Frequency contour). Bottom-left titled "$\gamma = 3$ and $P^2 = 9.3$" (Time vs amplitude). Bottom-right titled "$\gamma = 3$ and $P^2= 9.3$" (Time vs Frequency contour).
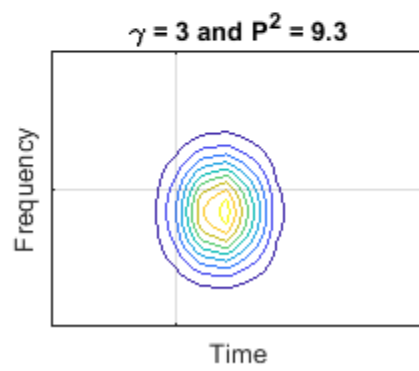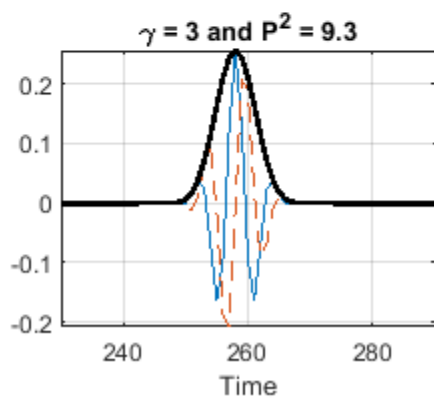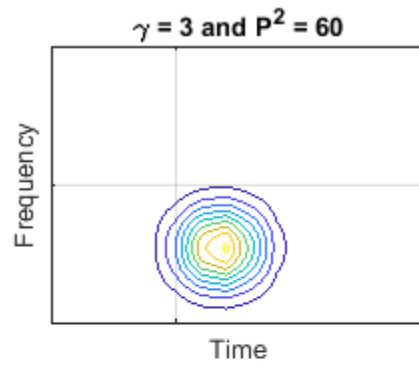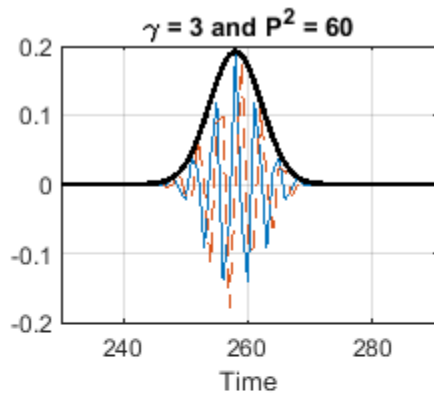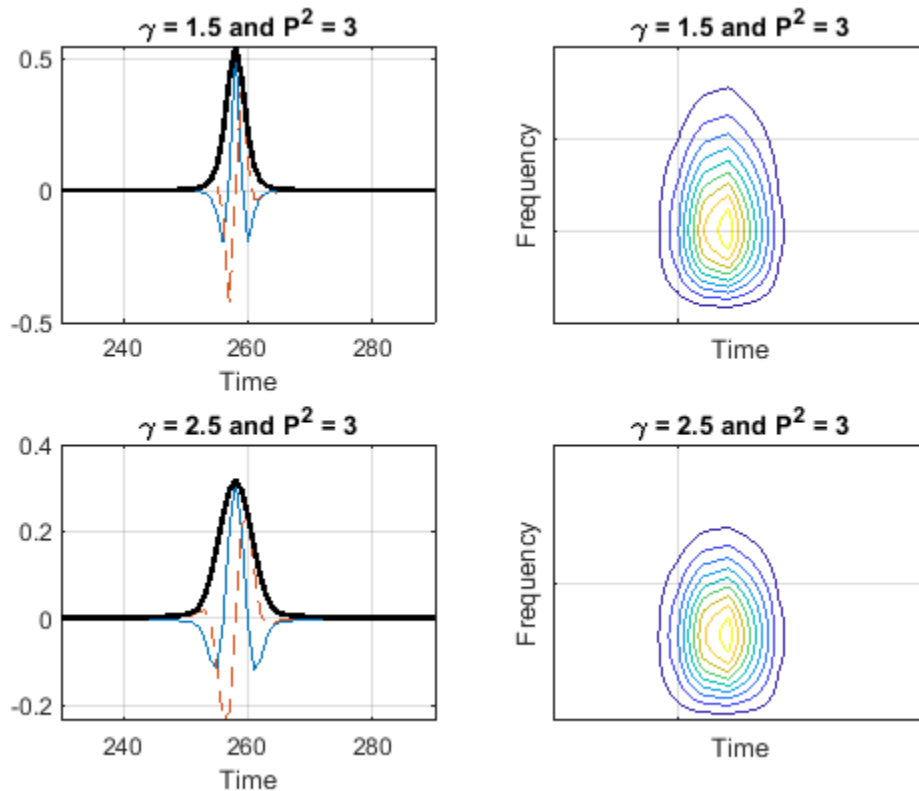
Relationship Between Analytic Morse Wavelet and Analytic Signal

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

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

The value of the analytic signal depends on ω.

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

- For ω = 0, the Fourier transform of an analytic signal is equal to the Fourier transform of the corresponding nonanalytic signal.

- For ω < 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');
```
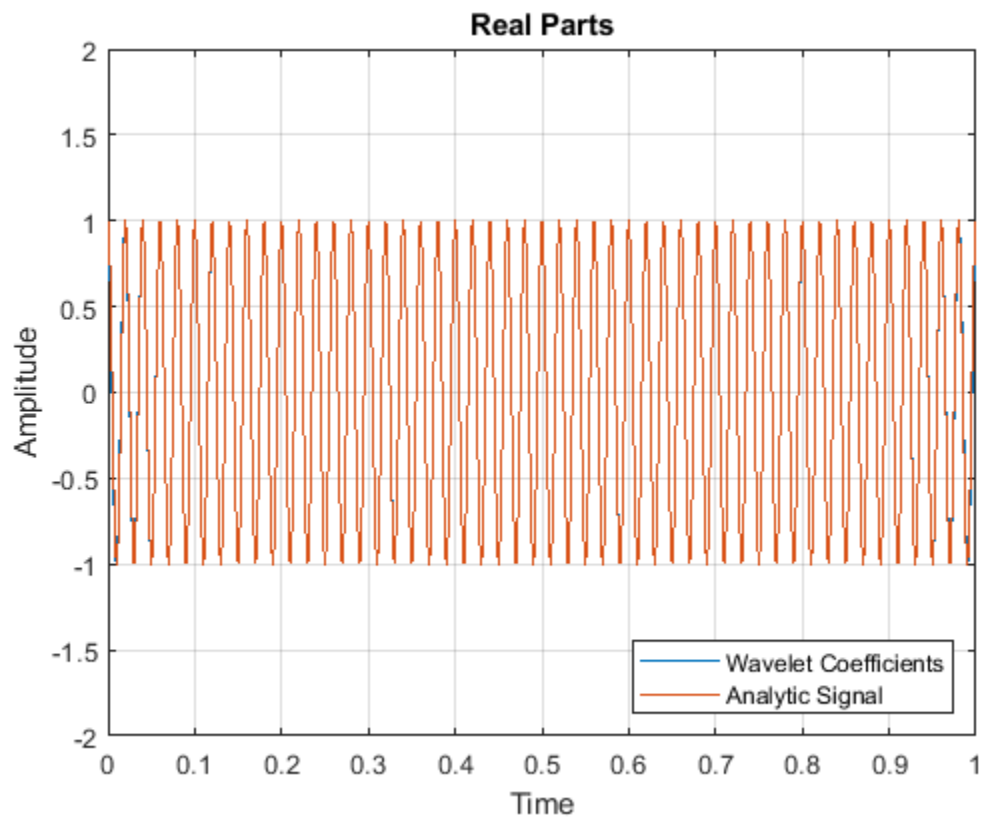
```
figure;
plot(t,[imag(morsecoefx)' imag(analytsig)']);
title('Imaginary Parts');
ylim([-2 2]); grid on;
legend('Wavelet Coefficients','Analytic Signal','Location','SouthEast');
xlabel('Time'); ylabel('Amplitude');
```
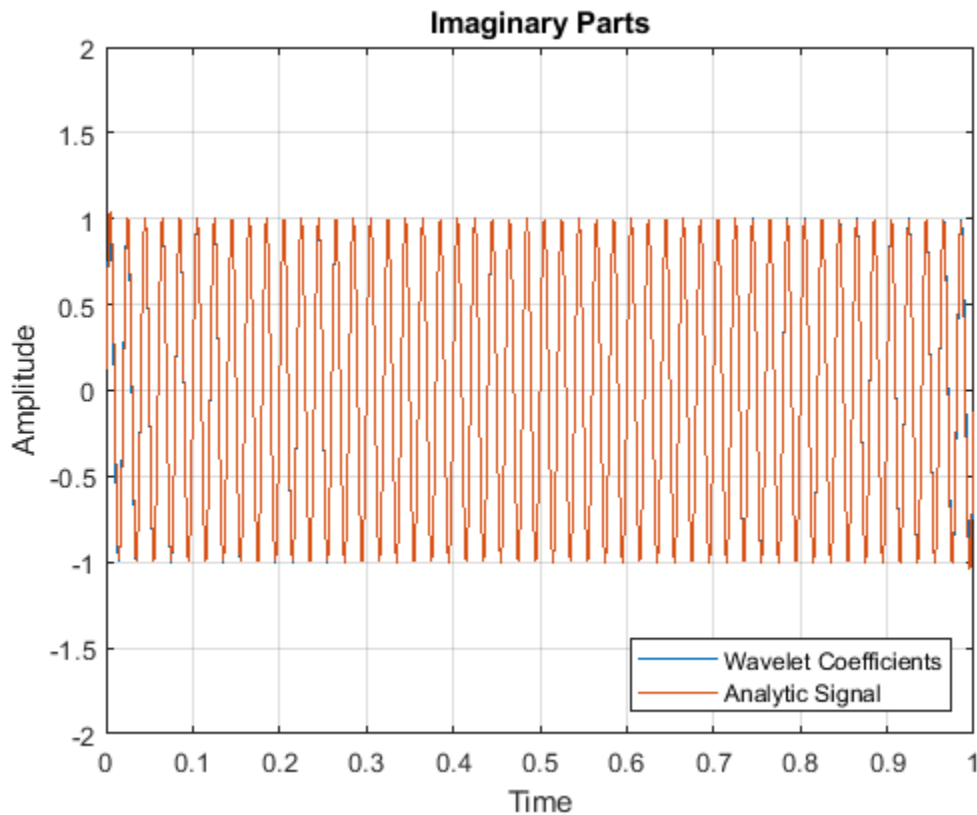
cwt uses L1 normalization and scales the wavelet bandpass filters to have a peak magnitude of 2. The factor of 1/2 in the above equation is canceled by the peak magnitude value.

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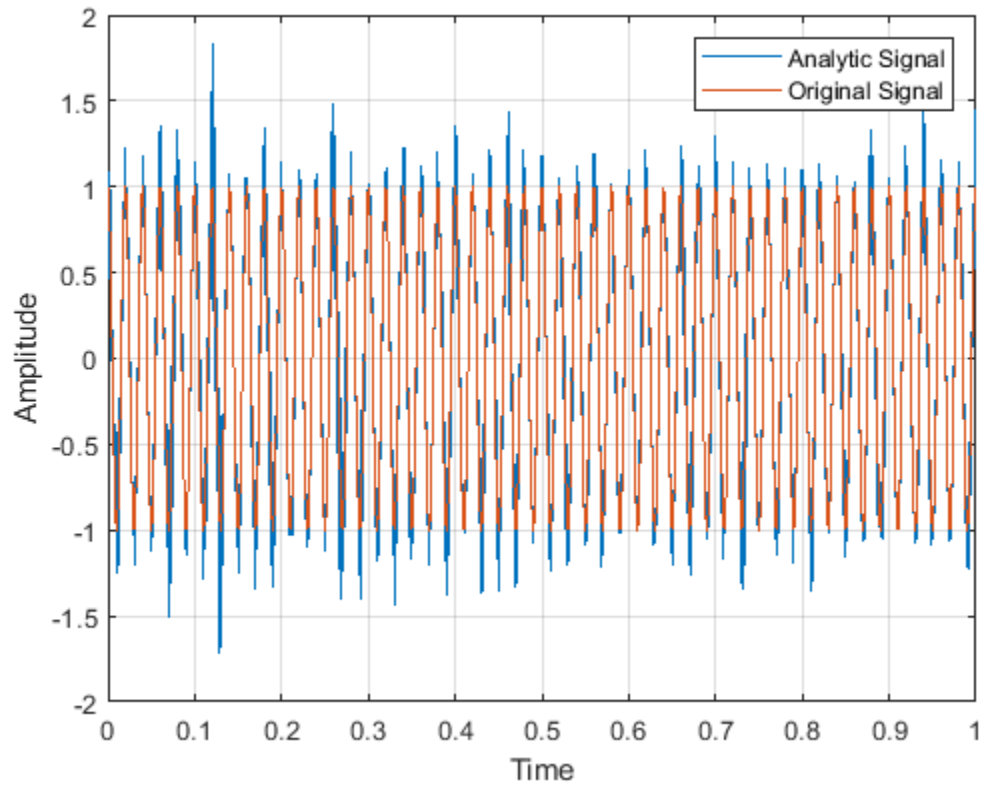
```
figure;
plot(t,[real(morsecoefy)' x']);
legend('Wavelet Coefficients','Original Signal');
grid on;
xlabel('Time'); ylabel('Amplitude');
ylim([-2 2])
```
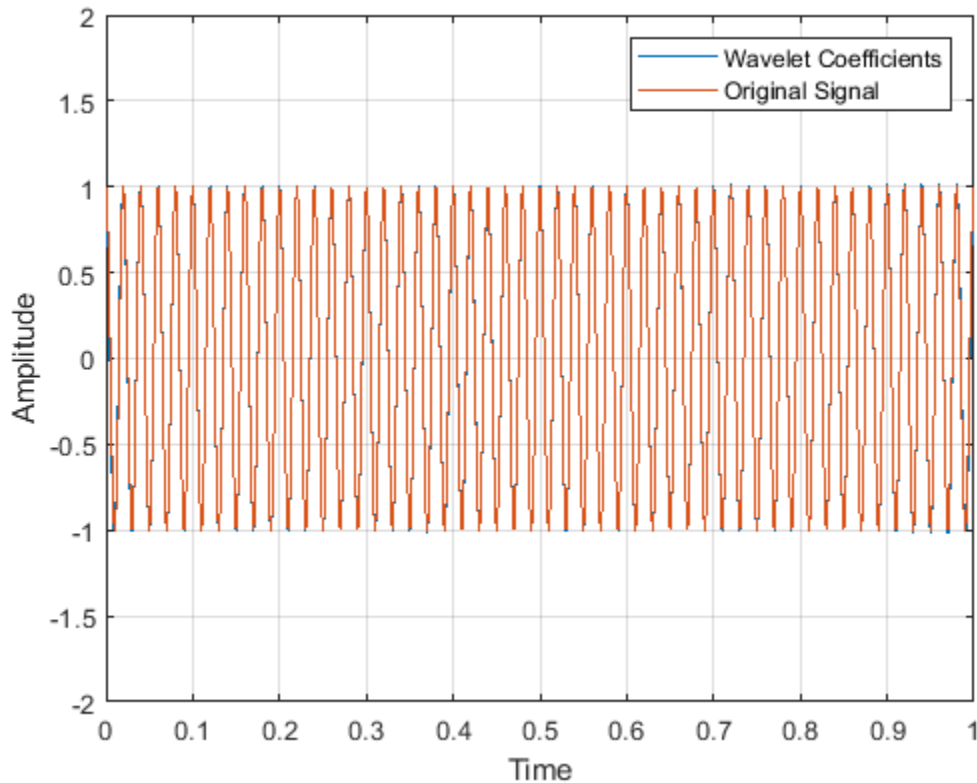
Recommended Morse Wavelet Settings for the CWT

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.

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)
```
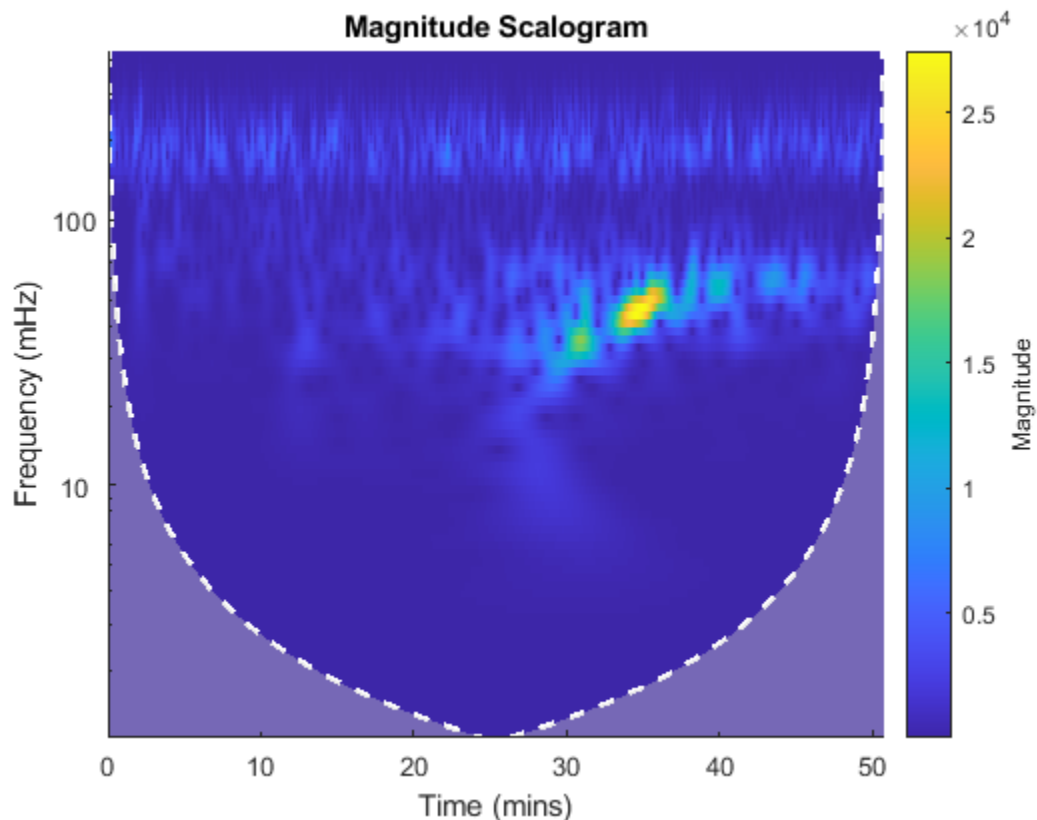

In addition to the scalogram, the plot also features a dashed white line and shaded gray regions from the edge of the white line to the time and frequency axes. Plot the same data using the sampling interval instead of sampling rate. Now the scalogram is displayed in periods instead of frequency.

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

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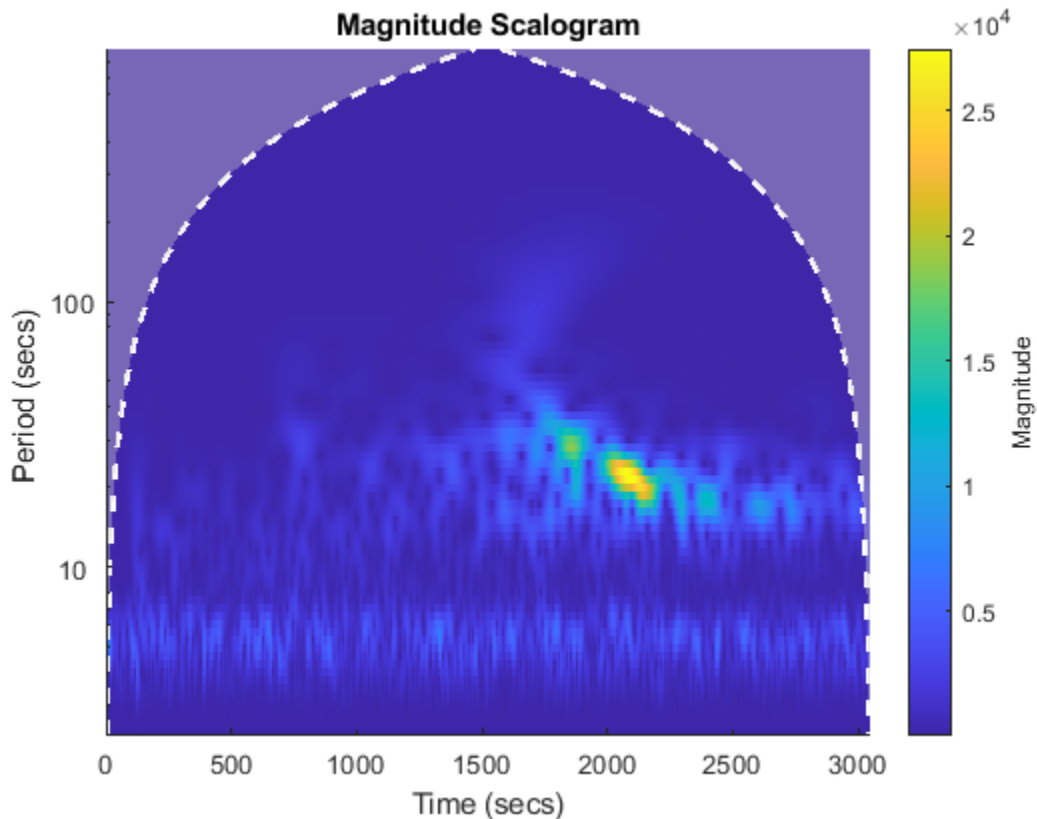
The orientation of the dashed white line has flipped upside down, but the line and the shaded regions are still present.

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.

2-27

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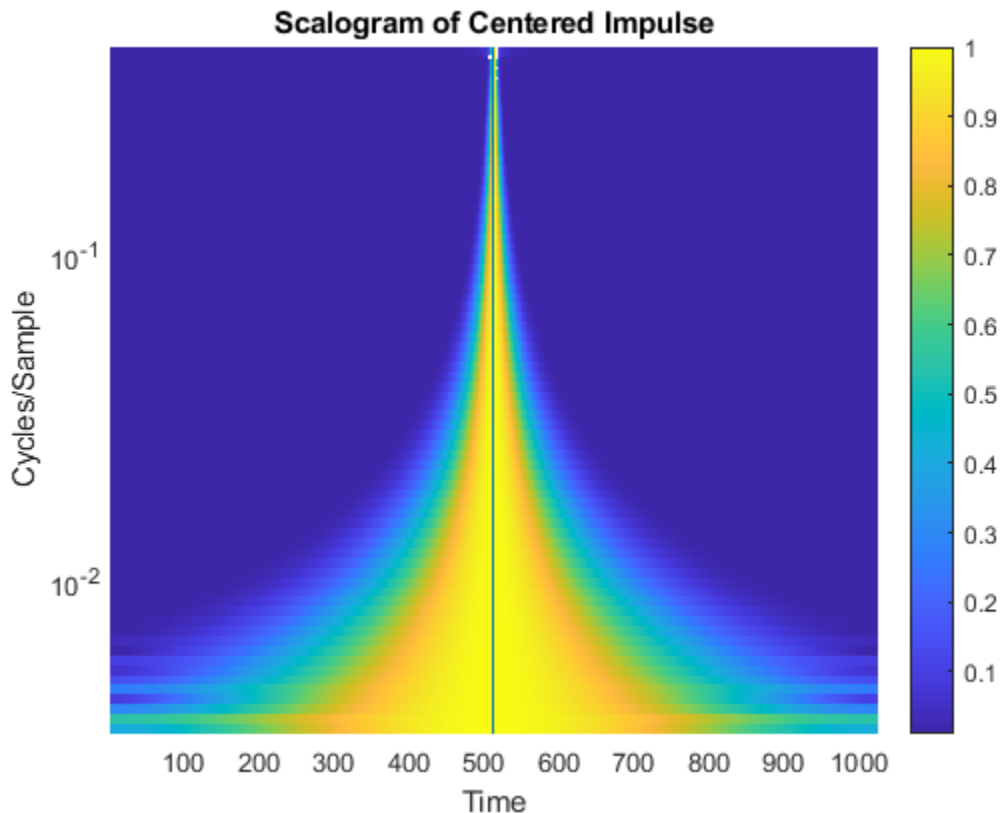

The solid black line shows the location of the impulse in time. Note that as the frequency decreases, the width of the CWT coefficients in time that are nonzero and centered on the impulse increases. Conversely, as the frequency increases, the width of the CWT coefficients that are nonzero decreases 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')
```
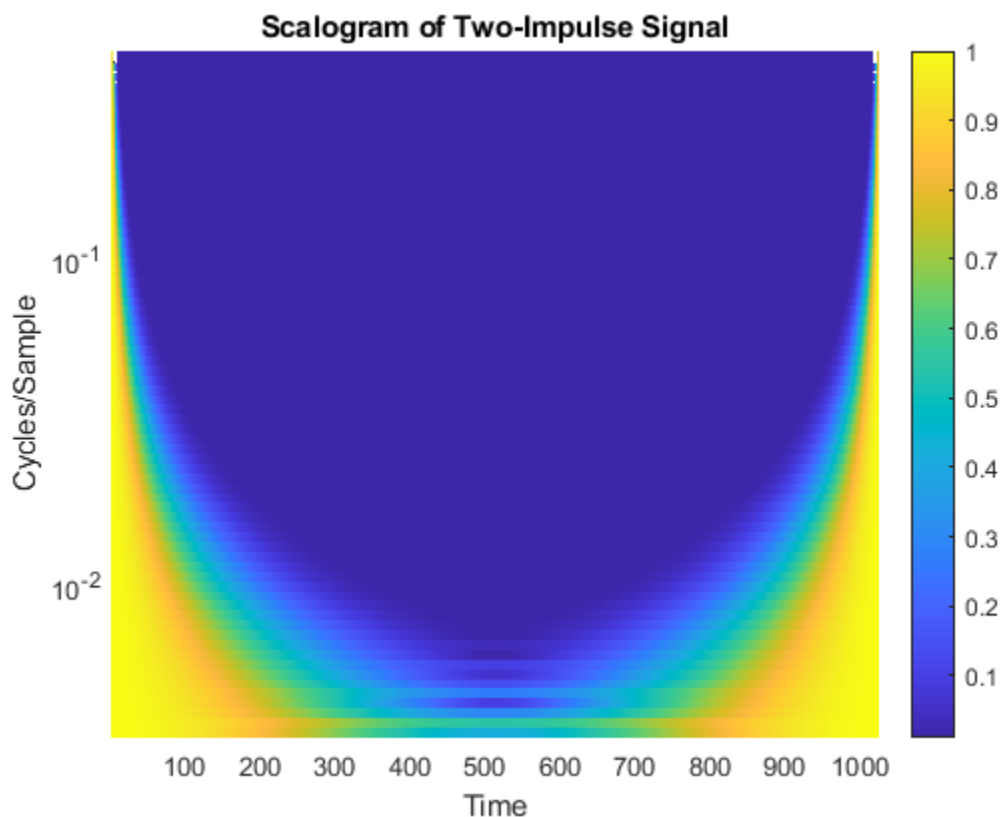

Here it is clear that the cone of influence for the extreme boundaries of the observation interval extends into the interval to a degree that depends on the scale of the wavelet. Therefore, wavelet coefficients well inside the observation interval can be affected by what data the wavelet sees at the 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')
```
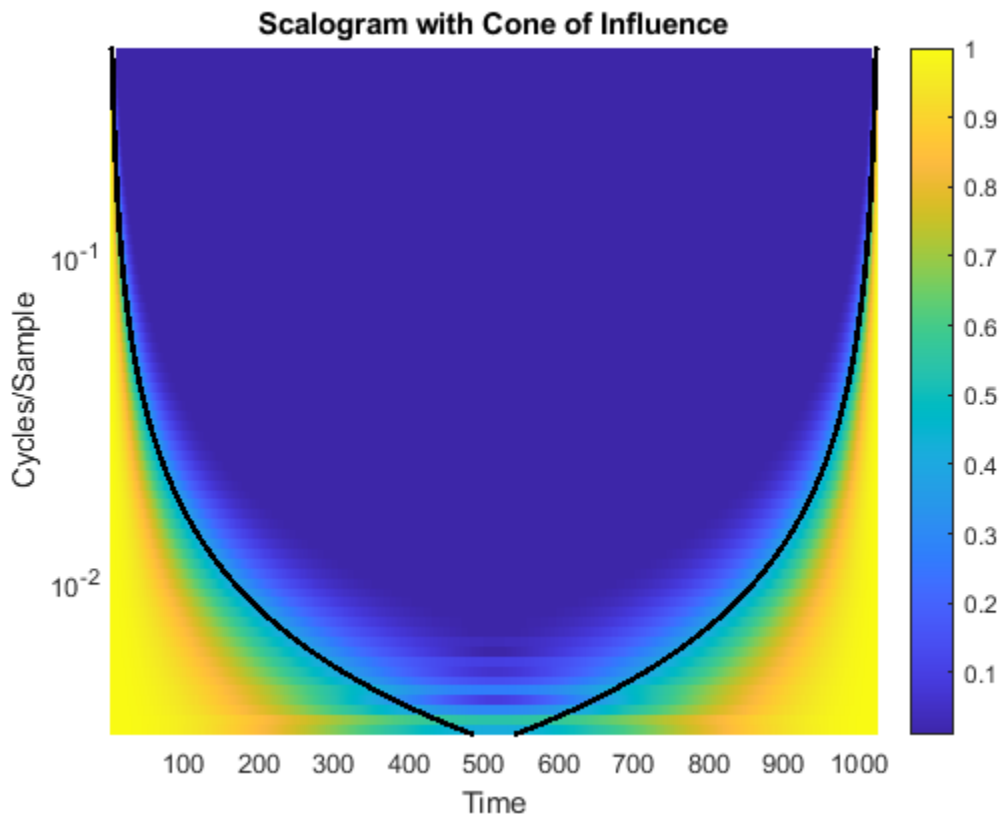

You see that the computed COI is a good approximation to boundaries of the significant effects of an impulse at the beginning and end of the signal.

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')
```
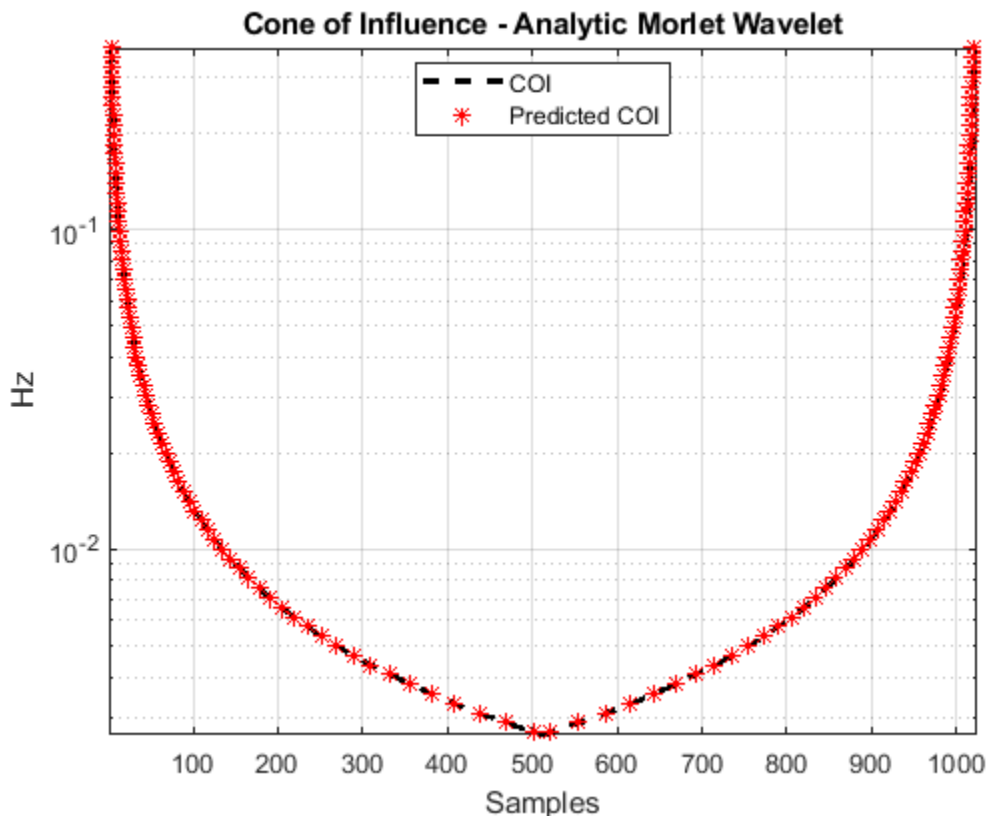

The last example shows the same correspondence for the default Morse wavelet in `cwtfilterbank` and `cwt`. The time-domain standard deviation of the default Morse wavelet is 5.5008, and the peak frequency is 0.2995 cycles/sample. Use the center frequencies of the wavelet bandpass filters as well as the 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')
```
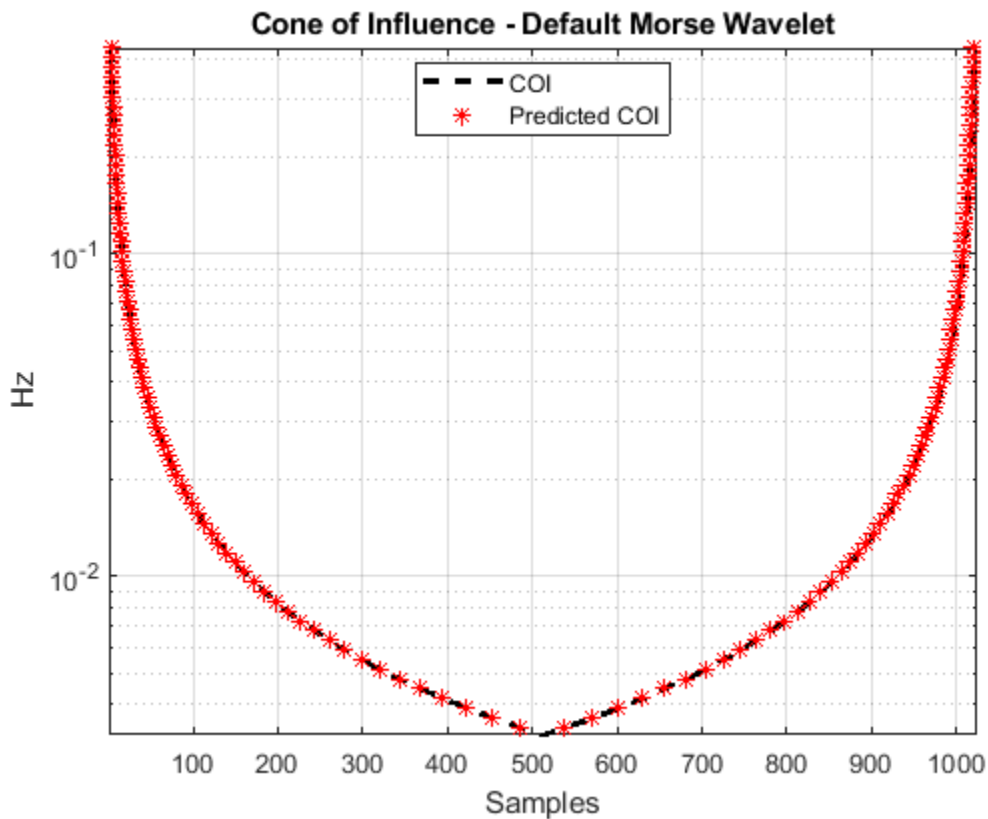

Appendix

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')
```

```matlab
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.

[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

Apps
Signal Analyzer

Functions
cwt | cwtfilterbank | pspectrum

More About

- "Morse Wavelets" on page 2-16
- "Continuous and Discrete Wavelet Transforms"

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 $\dfrac{15\pi}{(0.8 - t)^2}/2\pi$. The instantaneous frequency of the second chirp at time t is $\dfrac{5\pi}{(0.8 - t)^2}/2\pi$. Plot the signal.

```
load hychirp
plot(t,hychirp)
grid on
title('Signal')
axis tight
xlabel('Time (s)')
ylabel('Amplitude')
```
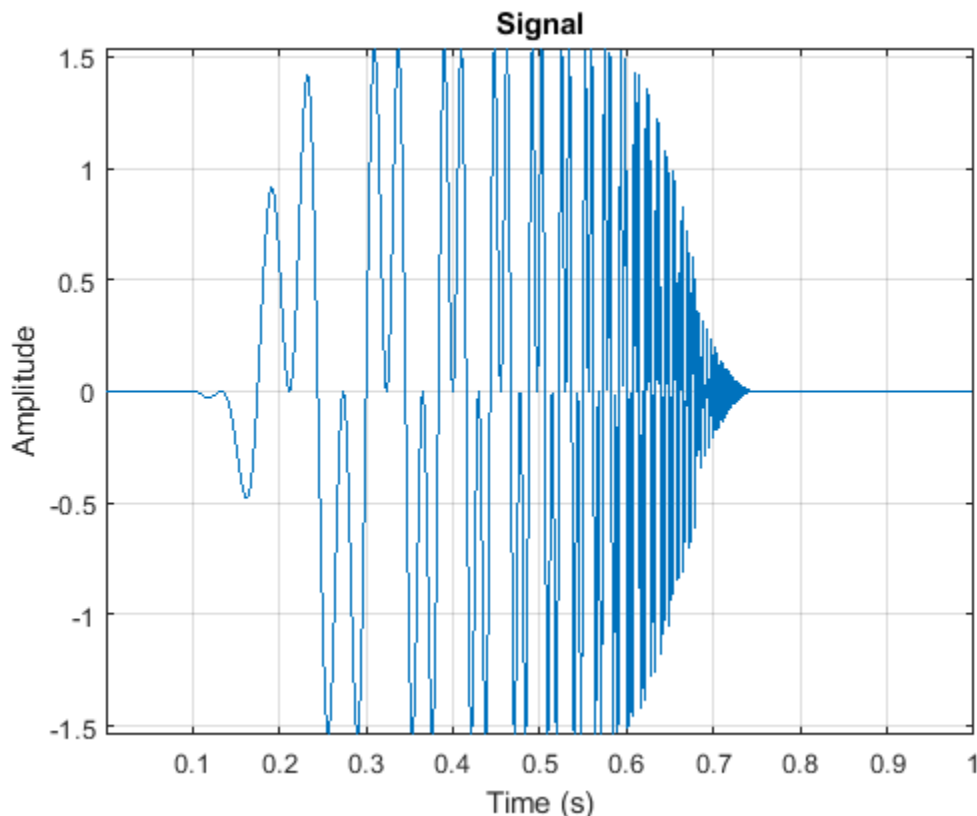
Time-Frequency Analysis: Fourier Transform

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])
```
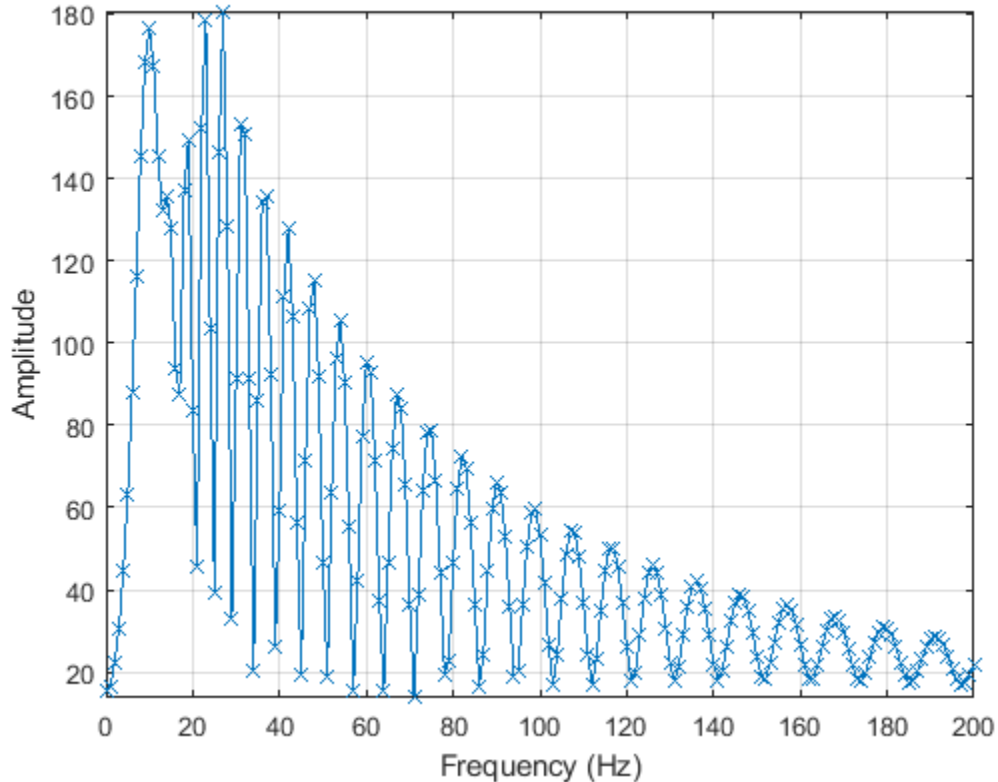

Time-Frequency Analysis: Short-Time Fourier Transform

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.
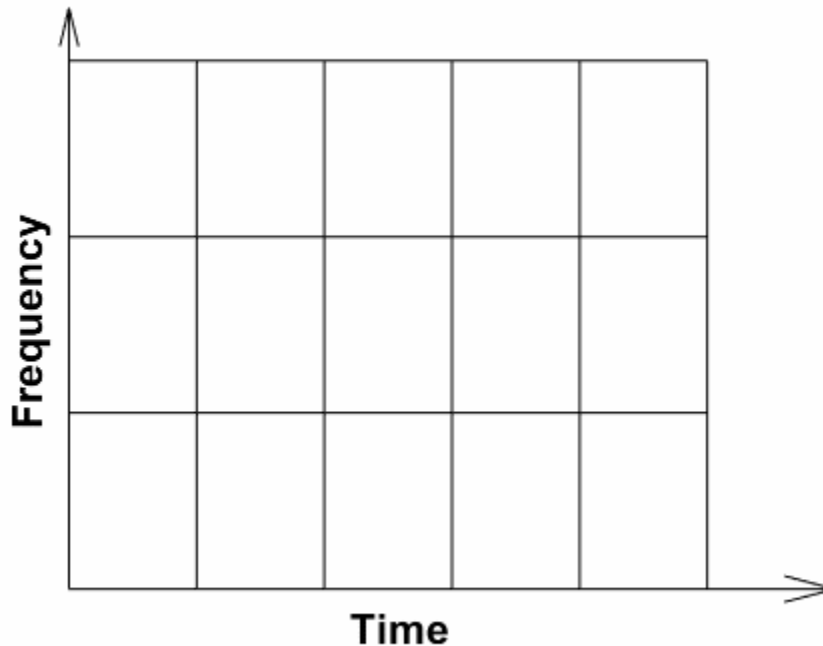
The STFT provides some information on both the timing and the frequencies at which a signal event occurs. However, choosing a window (segment) size is key. For time-frequency analysis using the STFT, choosing a shorter window size helps obtain good time resolution at the expense of frequency resolution. 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)
```
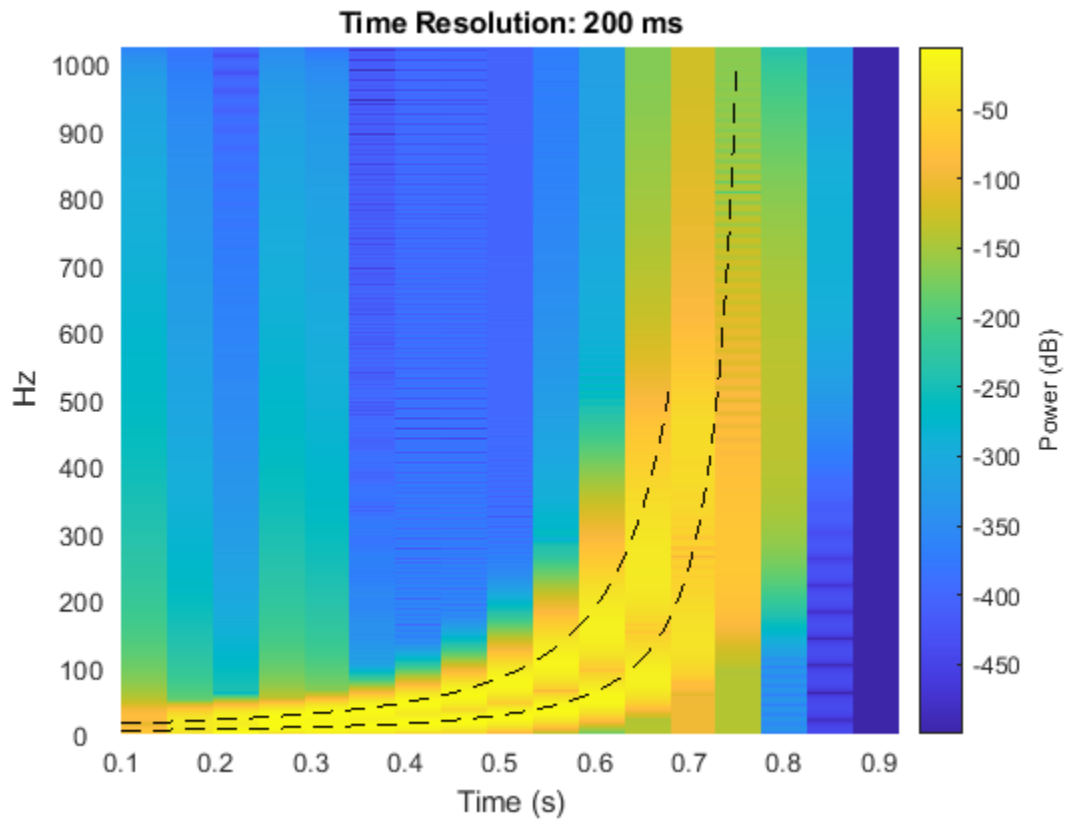
Now use `helperPlotSpectrogram` to plot the spectrogram with a time window size of 50 milliseconds. The higher frequencies, which occur later in the signal, are now resolved, but the lower frequencies at the beginning of the signal are not.

```
helperPlotSpectrogram(hychirp,t,Fs,50)
```
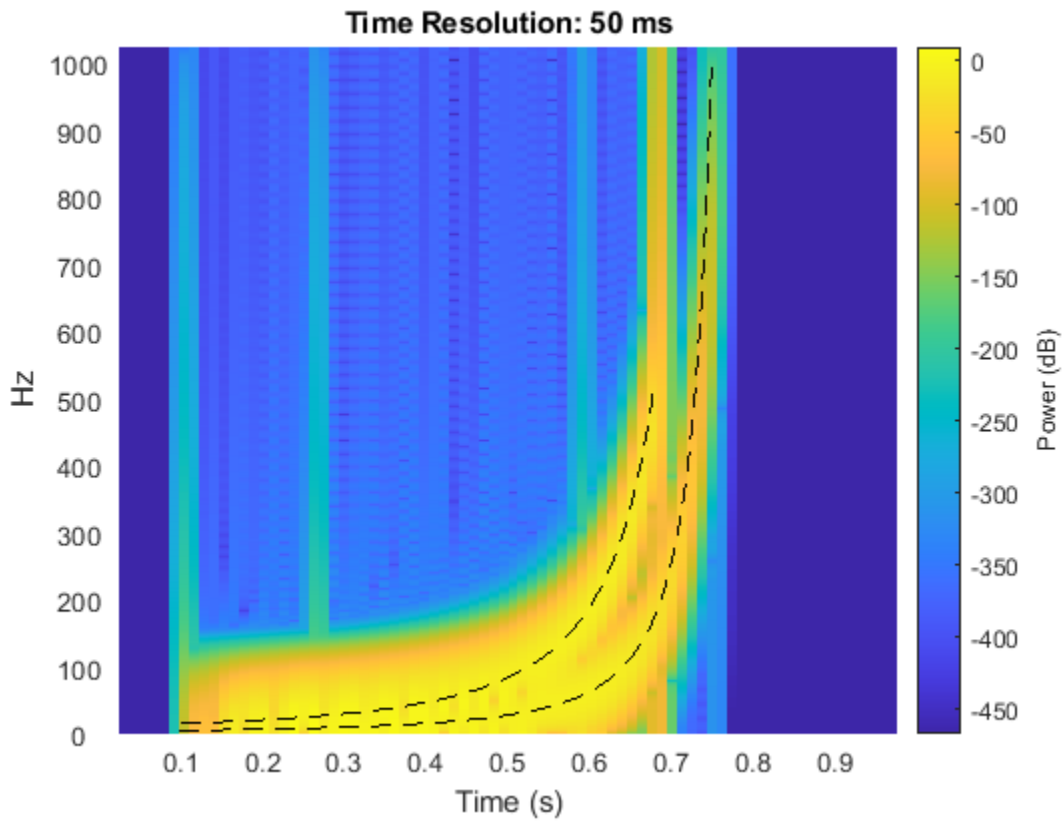
For nonstationary signals like the hyperbolic chirp, using the STFT is problematic. No single window size can resolve the entire frequency content of such signals.

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.
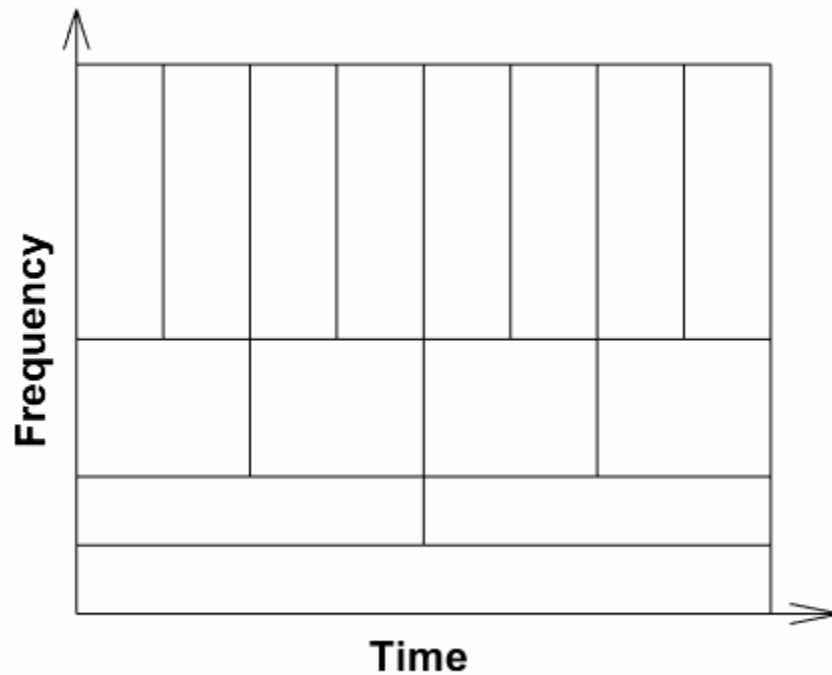
The CWT tiling of the plane is useful because many real-world signals have slowly oscillating content that occurs on long scales, while high frequency events tend to be abrupt or transient. However, if it were natural for high-frequency events to be long in duration, then using the CWT would not be appropriate. 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)
```

Magnitude Scalogram
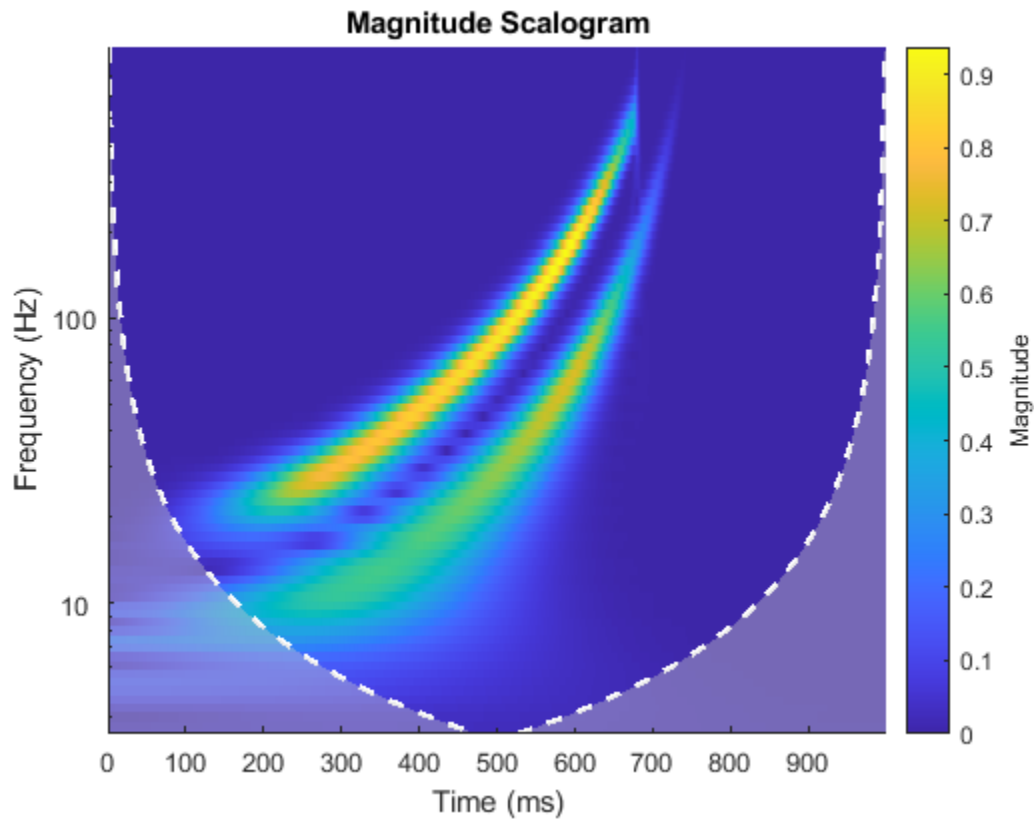
The white dashed line marks what is known as the *cone of influence*. The cone of influence shows areas in the scalogram potentially affected by boundary effects. For more information, see "Boundary Effects and the Cone of Influence" on page 2-26.

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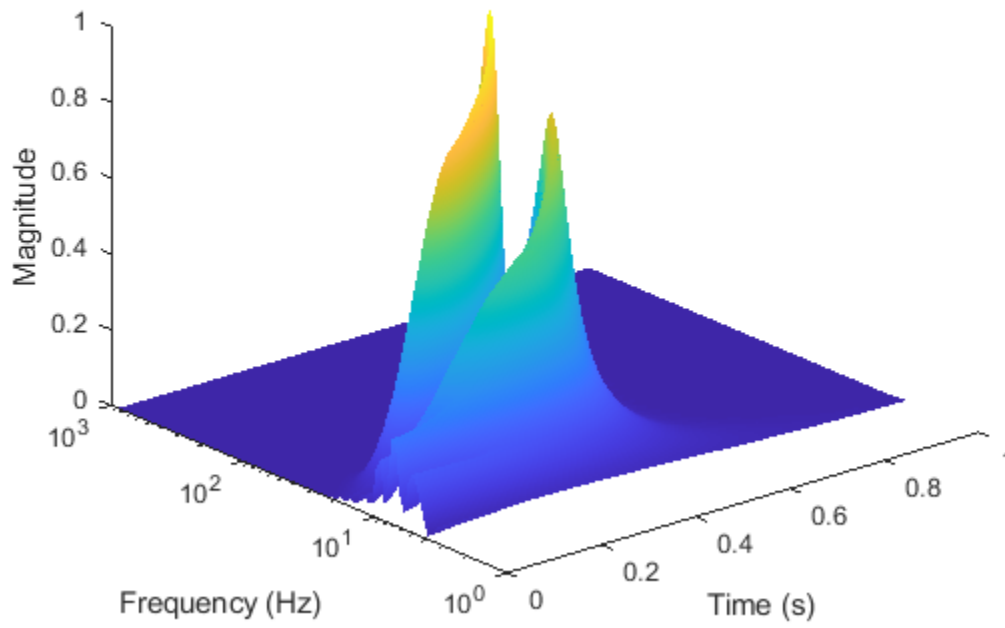
Use the helper function `helperPlotScalogram` to plot the scalogram of the signal and the instantaneous frequencies. The source code for `helperPlotScalogram` is listed in the appendix. The instantaneous frequencies align well with the scalogram features.

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

Scalogram and Instantaneous Frequencies
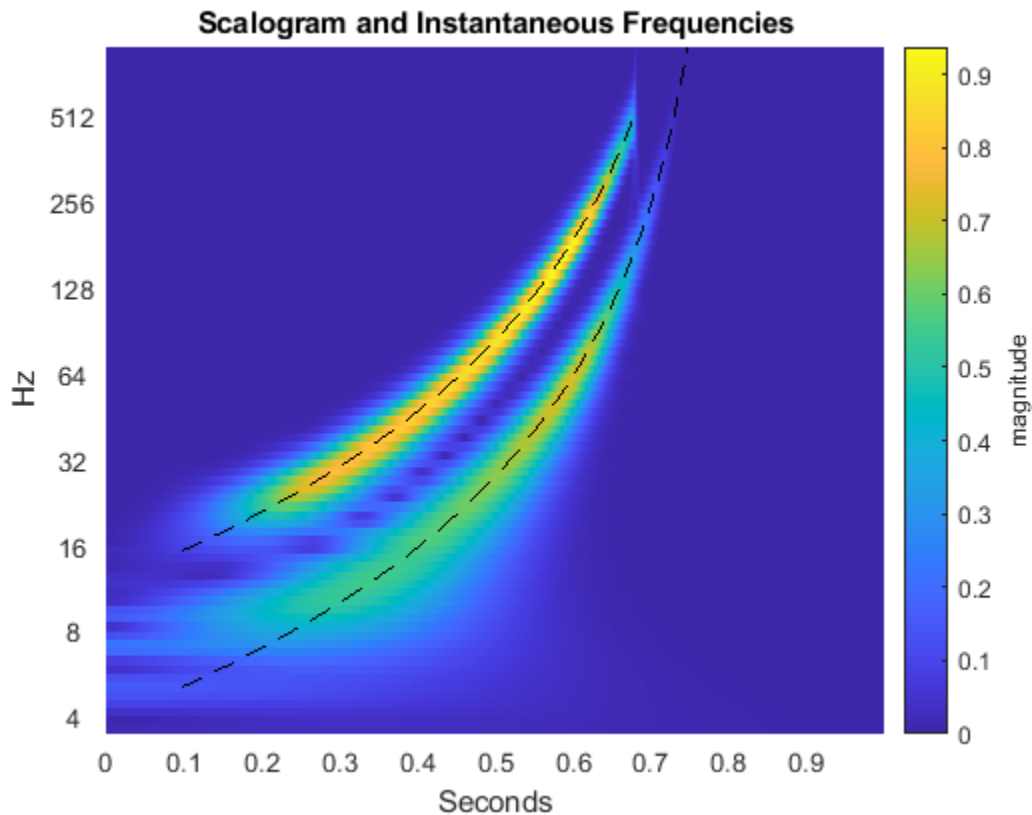
Appendix – Helper Functions

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 2-26
- "Classify Time Series Using Wavelet Analysis and Deep Learning" on page 13-79
- "Wavelet Time Scattering Classification of Phonocardiogram Data" on page 13-43

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')
```
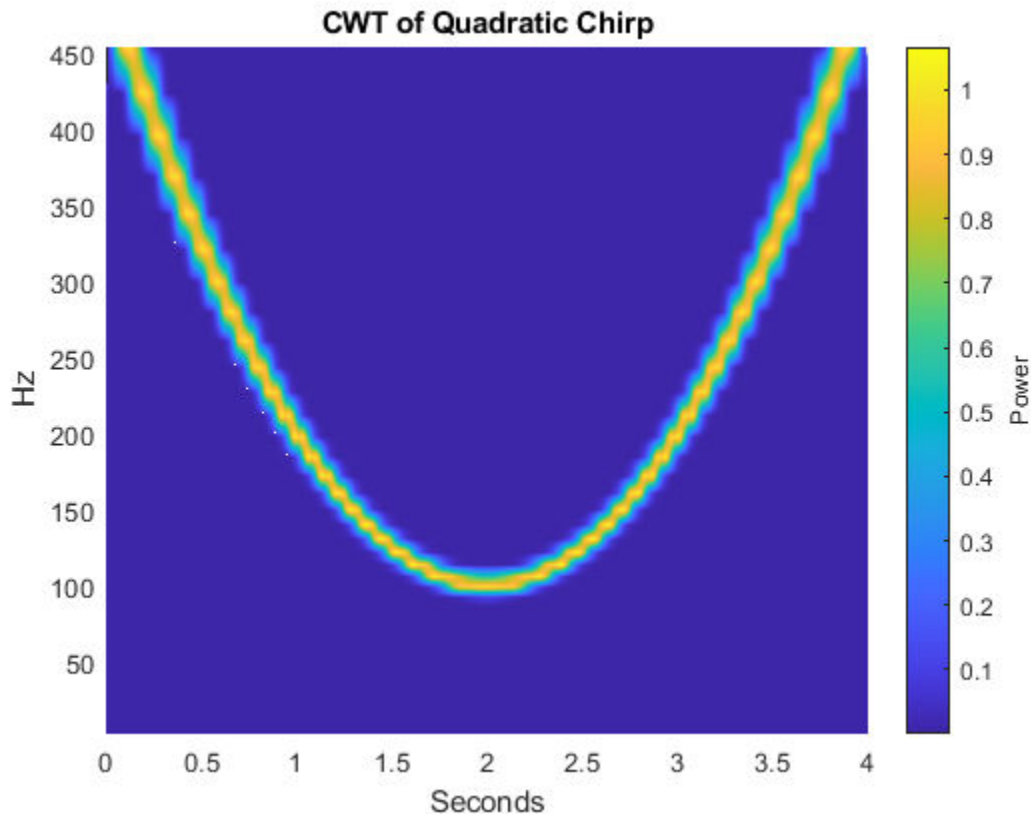

The CWT clearly shows the time evolution of the quadratic chirp's frequency. The quadratic chirp is a frequency-modulated signal. While that signal is synthetic, frequency and amplitude modulation occur frequently in natural signals as well. Use the CWT to obtain a time-frequency analysis of an echolocation 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')
```
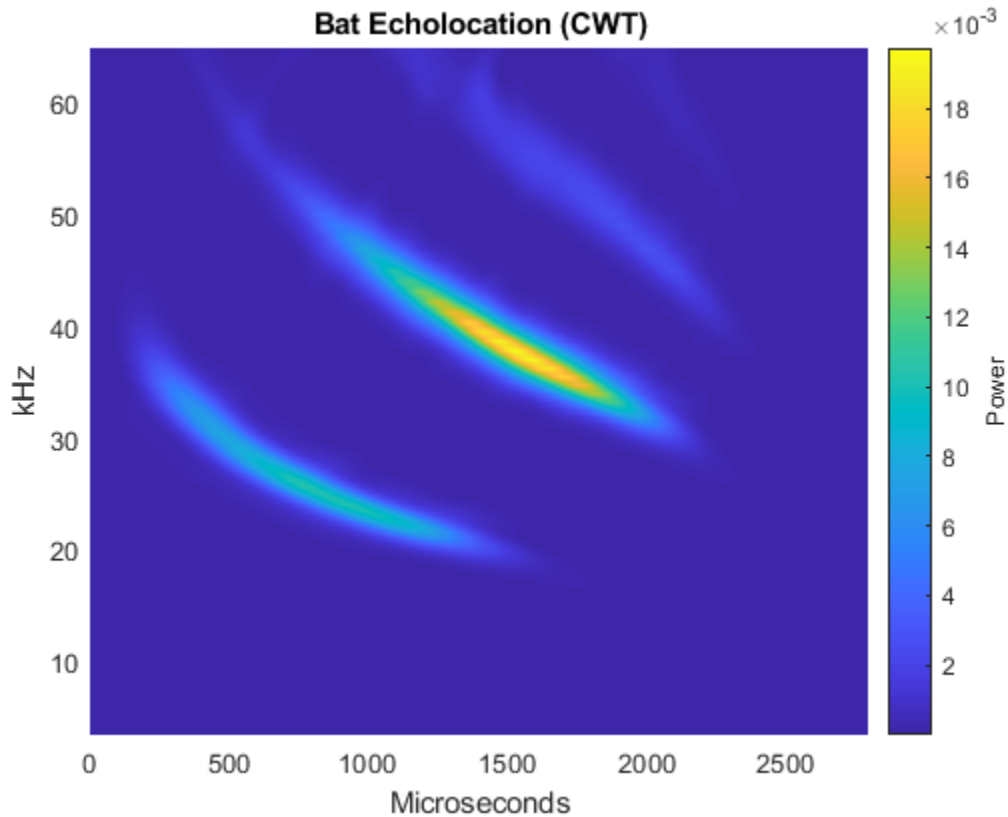

For the final example, obtain a time-frequency analysis of some seismograph data recorded during the 1995 Kobe earthquake. The data are seismograph (vertical acceleration, nm/sq.sec) measurements recorded at Tasmania University, Hobart, Australia on 16 January 1995 beginning at 20:56:51 (GMT) and continuing 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');
```
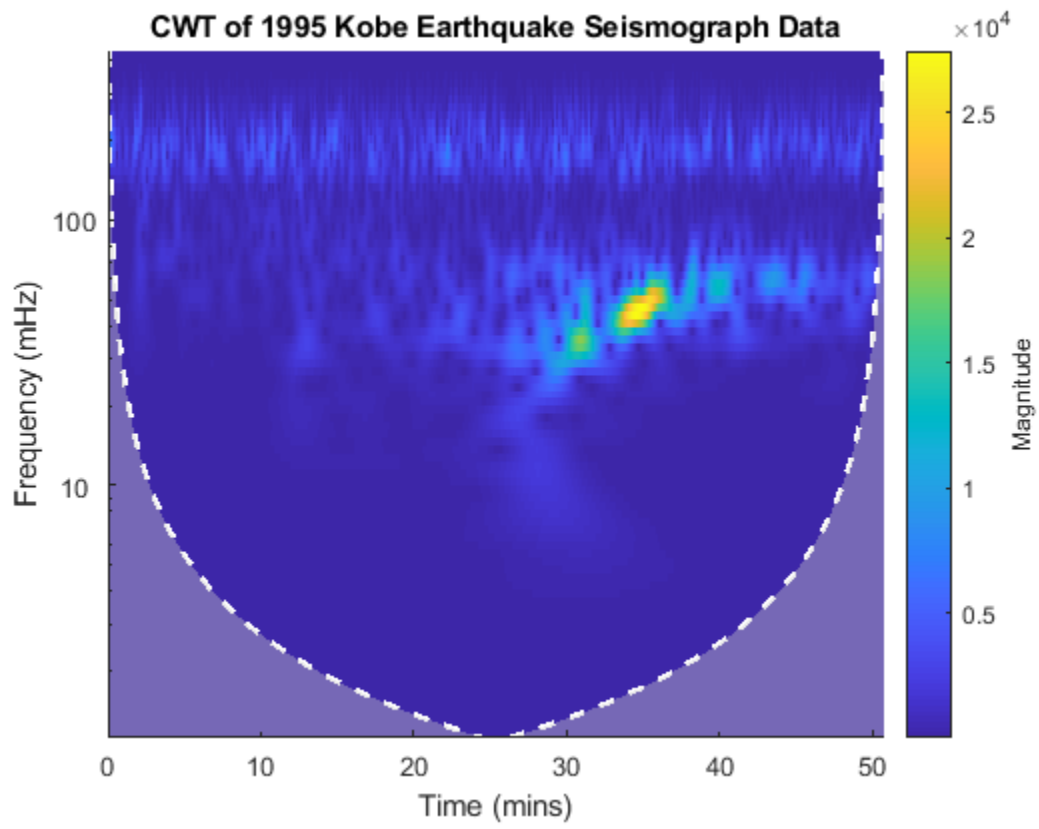
Remove Time-Localized Frequency Components

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')
```
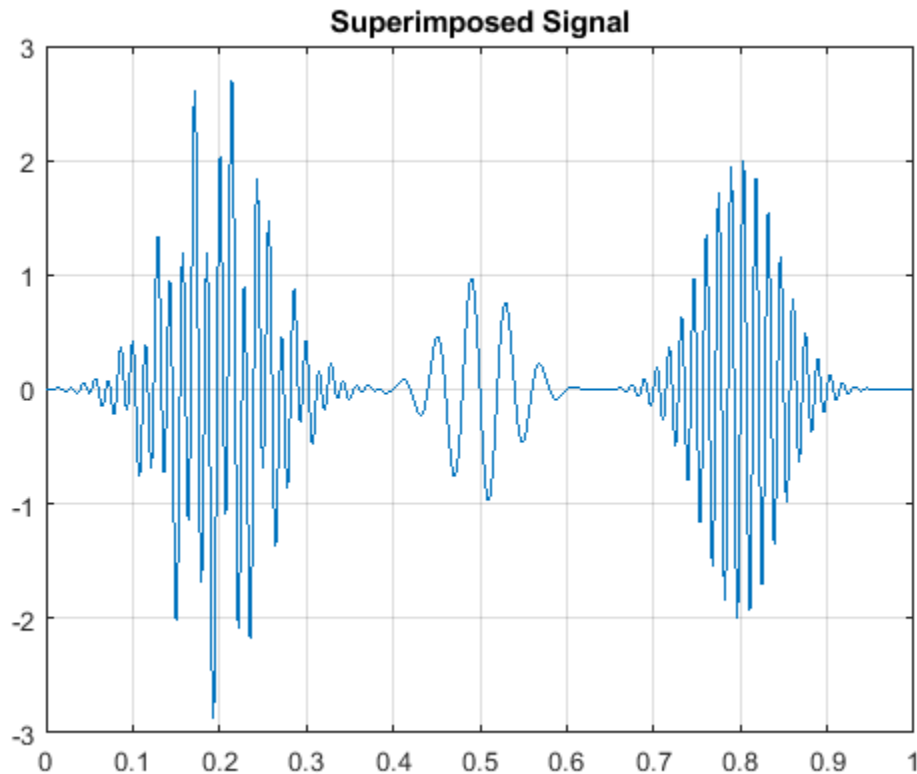

Obtain and display the CWT.

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

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)
```

Magnitude Scalogram
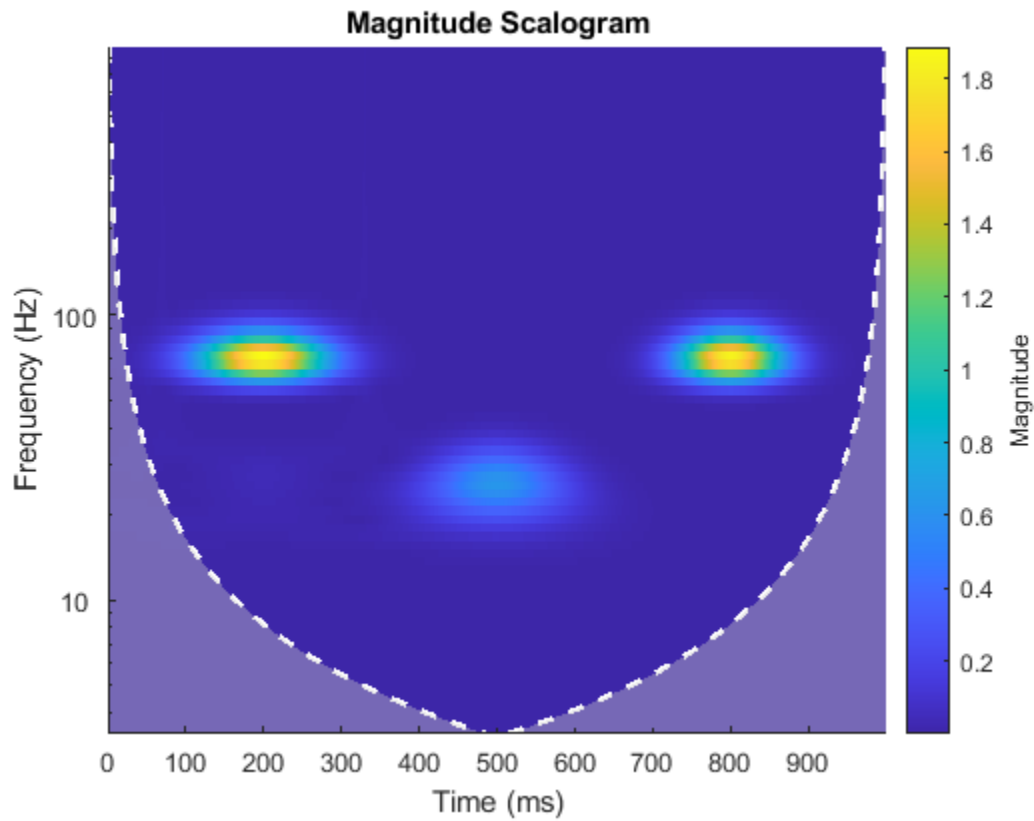
Plot the original signal and the reconstruction.

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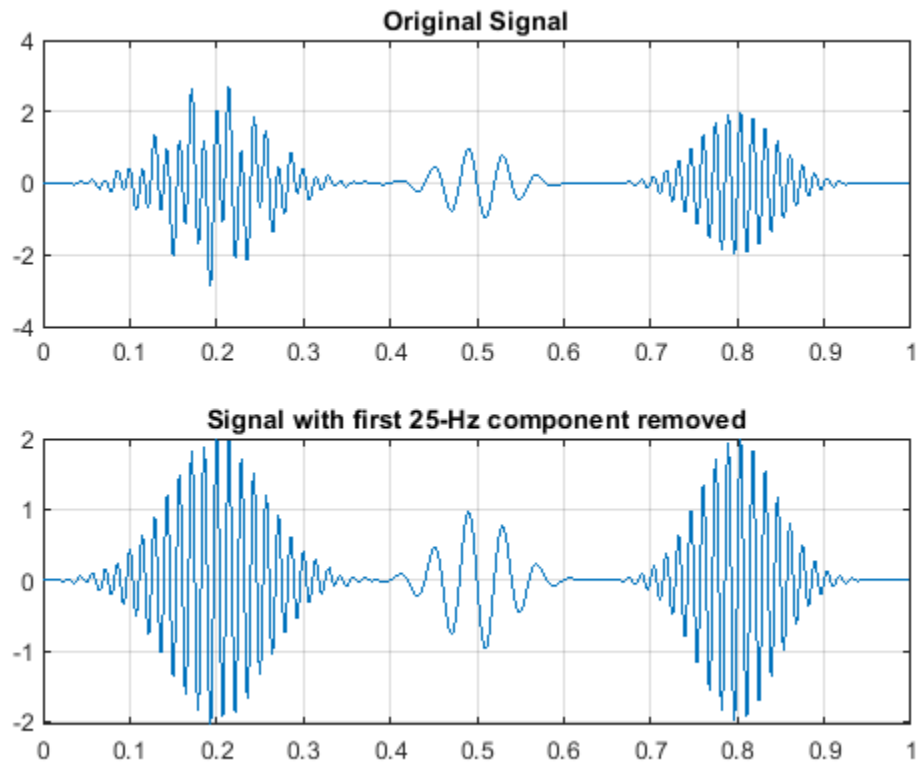
Compare the reconstructed signal with the original signal without the 25-Hz component centered at 0.2 seconds.

```
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
```
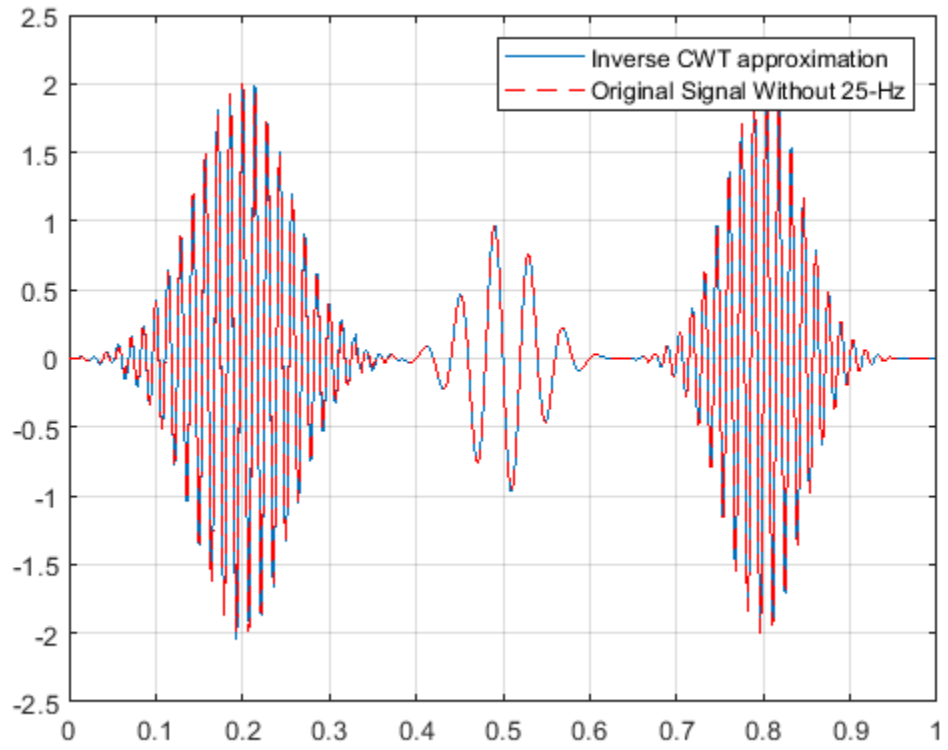
Time-Varying Coherence

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;
```

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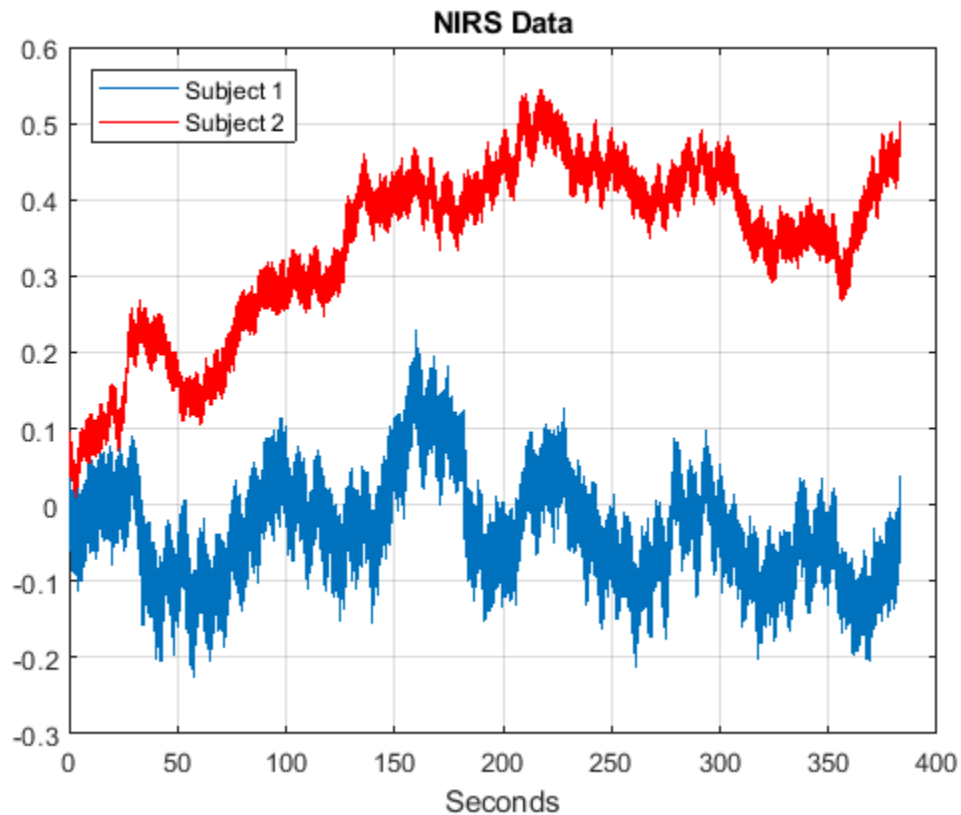
Examining the time-domain data, it is not clear what oscillations are present in the individual time series, or what oscillations are common to both data sets. Use wavelet analysis to answer both questions.

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,'numscales',16);
helperPlotCoherence(wcoh,tm,f,coi,'Seconds','Hz');
```

Wavelet Coherence

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),...
    'numscales',16);
helperPlotCoherence(wcoh,tm,seconds(P),seconds(coi),'Time (secs)','Periods (Seconds)');
```

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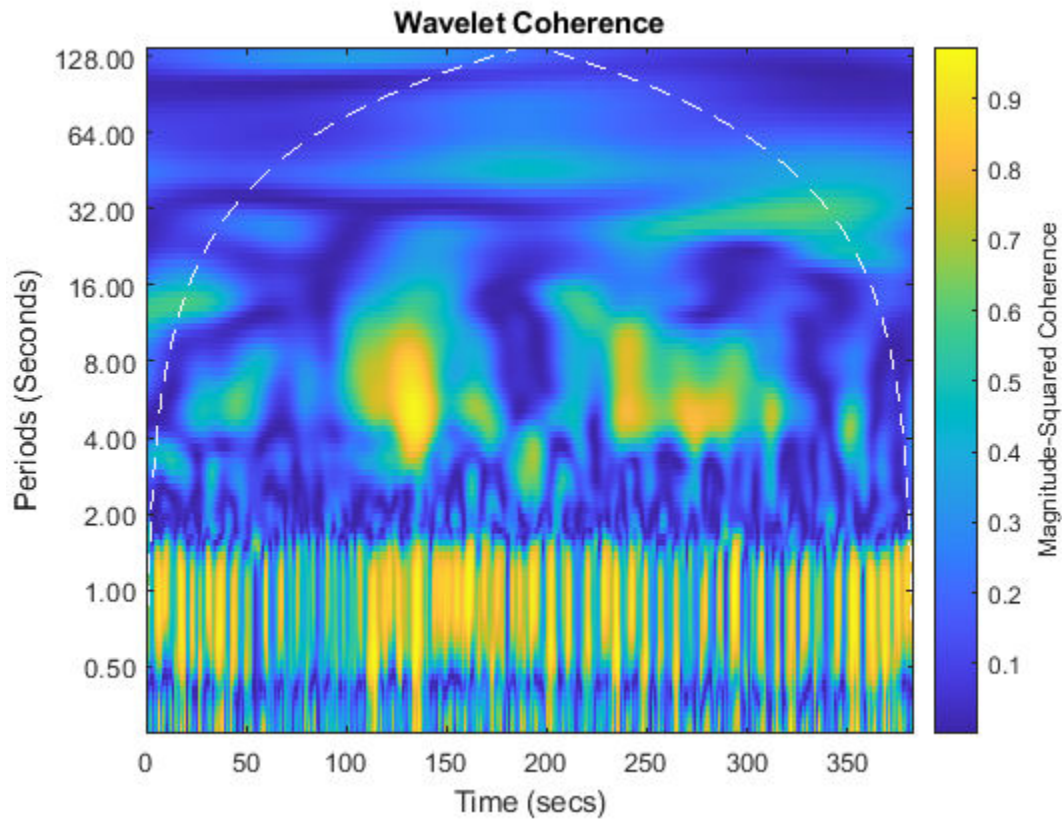
Again, note the coherent oscillations corresponding to the subjects' cardiac activity occurring throughout the recordings with a period of approximately one second. The task-related activity is also apparent with a period of approximately 8 seconds. Consult Cui, Bryant, & Reiss (2012) for a more detailed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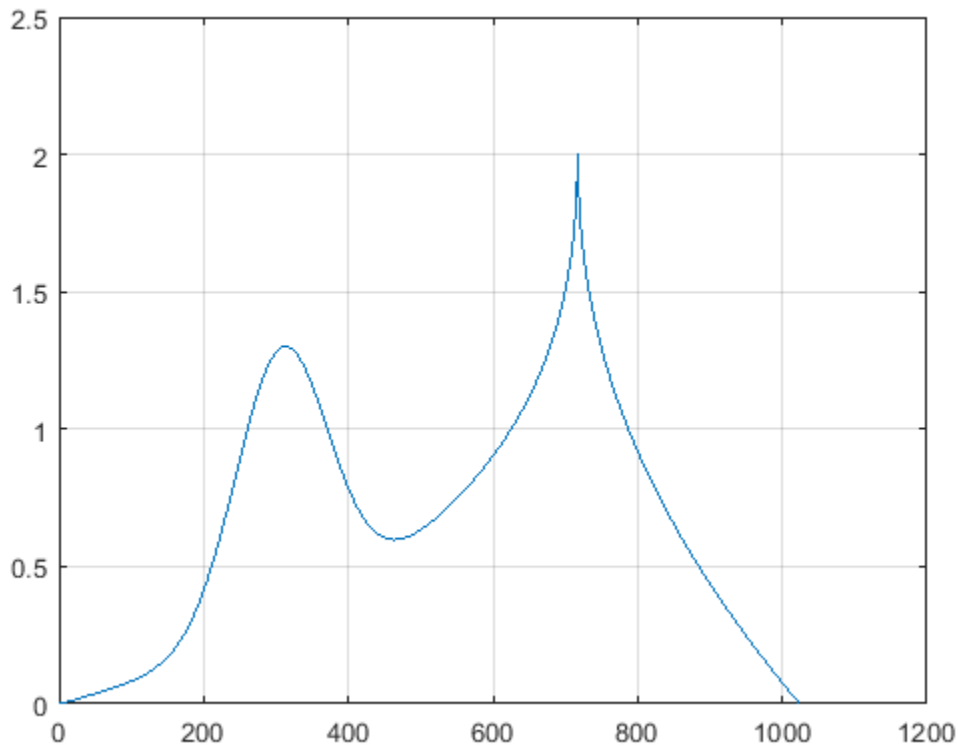


```
disp(caption)
```

```
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(cuspamax,'WaveletParameters',[2 2.5]);
```
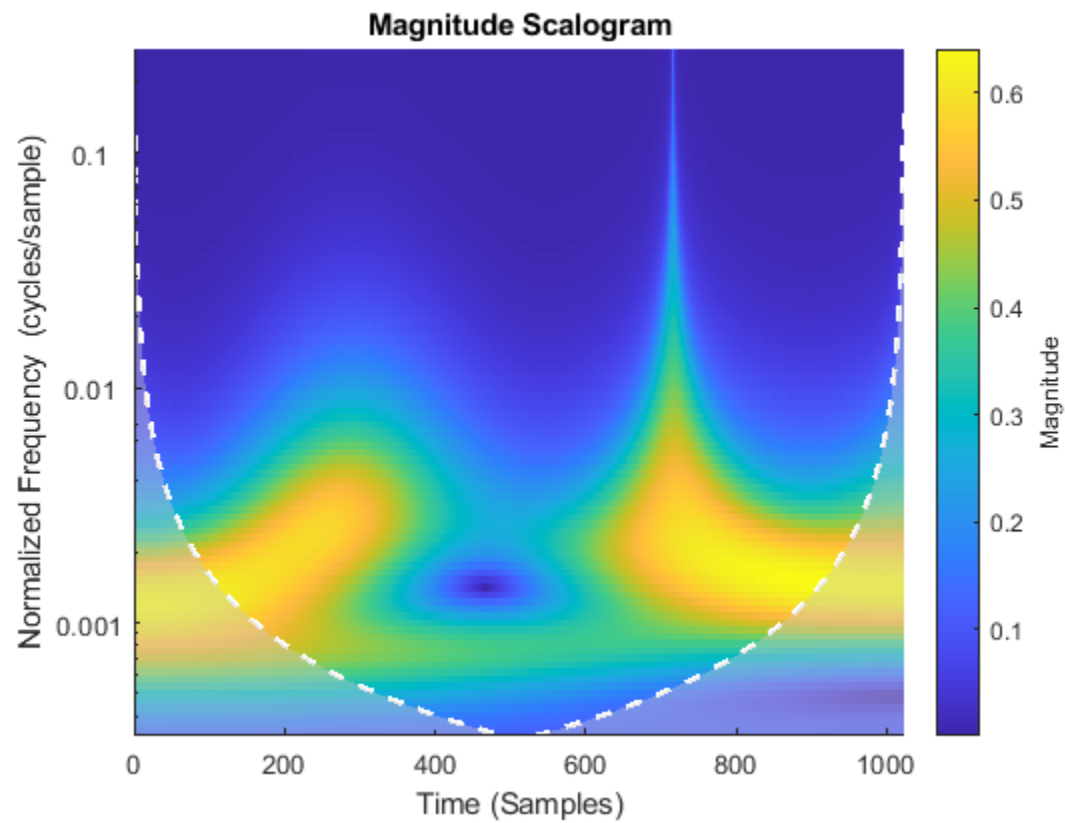
Obtain a plot of the wavelet maxima lines using wavelet transform modulus maxima. `wtmm` returns estimates of the Holder exponents, which characterize isolated singularities in a signal. Notice that the cusp is shown very clearly using `wtmm`.

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

Wavelet Transform Maxima Lines

	Sample	Holder Exponent
1	197	1.7353
2	308	1.8870
3	419	1.7950
4	633	0.3851
5	717	0.3879
6	720	0.3929
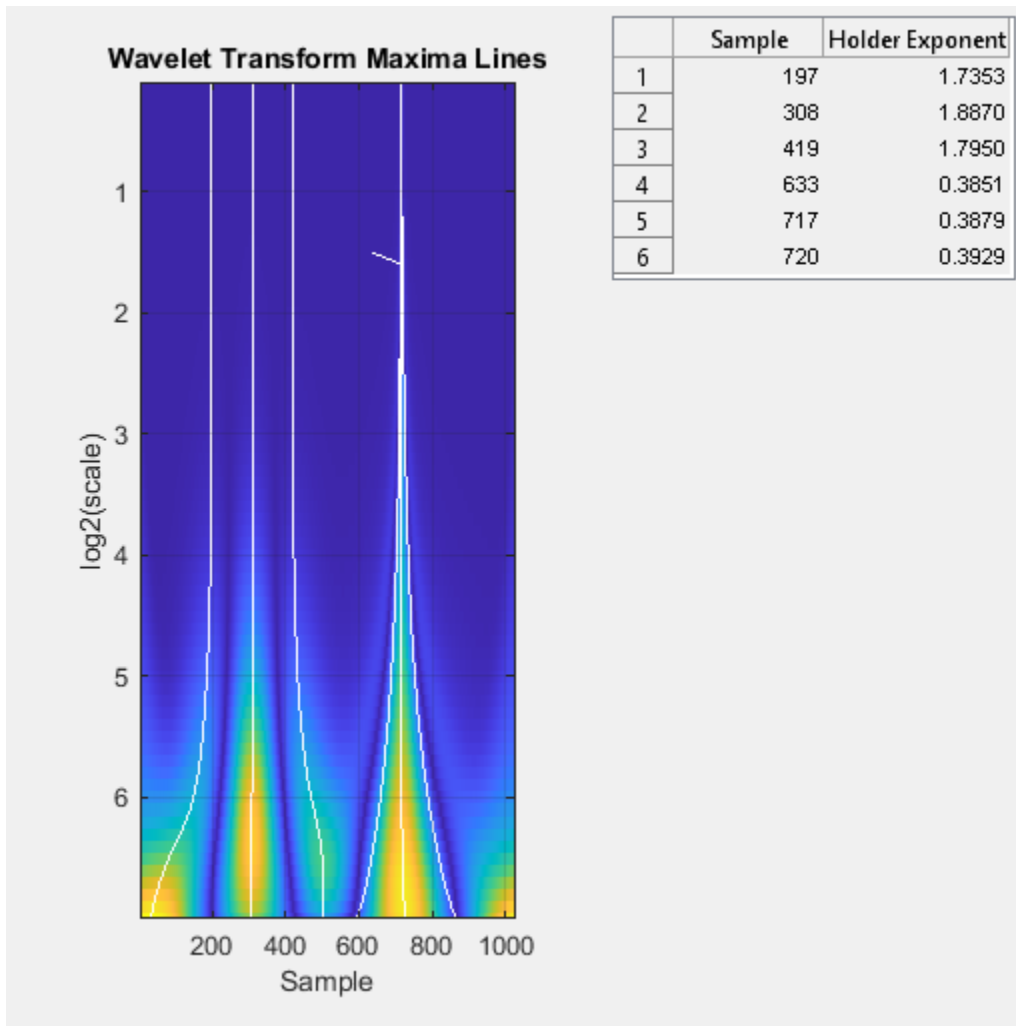
Complex Continuous Analysis Using the Wavelet Analyzer App

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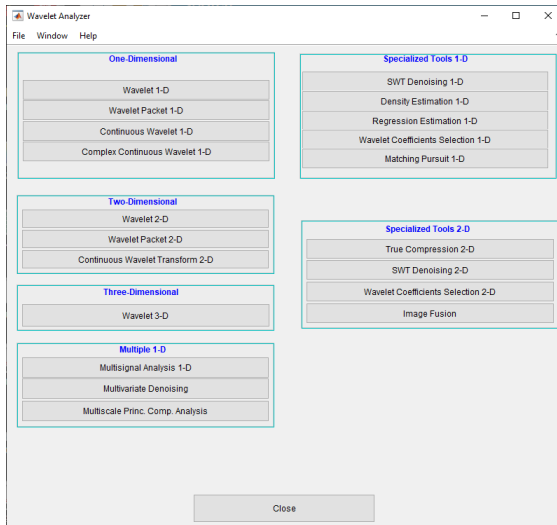
Click the **Complex Continuous Wavelet 1-D** menu item.

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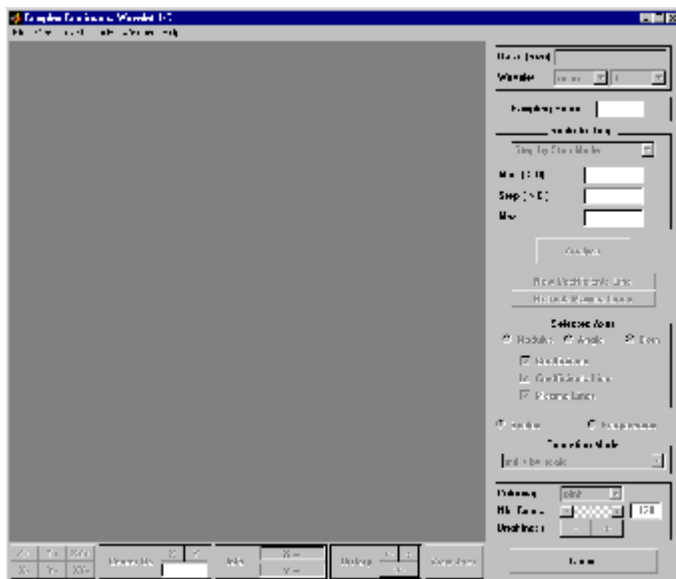
2   Load a signal.

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 2-57.

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

Click the **Analyze** button.

After a pause for computation, the tool displays the usual plots associated to the modulus of the coefficients on the left side, and the angle of the coefficients on the right side.
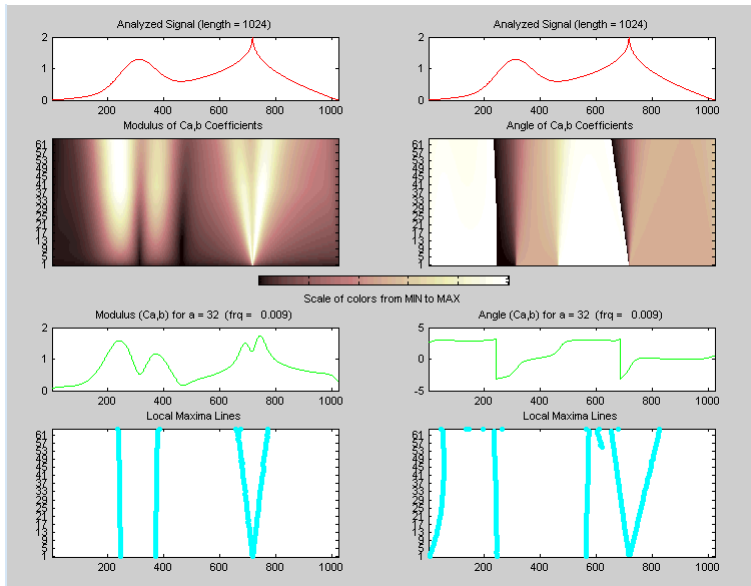
Each side has exactly the same representation that we found in "Continuous Wavelet Analysis of Noisy Sinusoid Using the Wavelet Analyzer App" on page 2-6.

Select the plots related to the modulus of the coefficients using the **Modulus** option button in the **Selected Axes** frame.
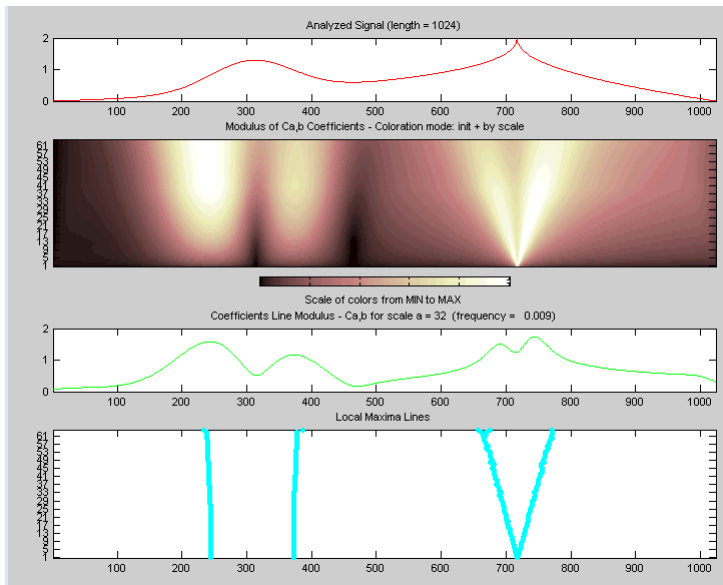
The figure now looks like the one in the real **Continuous Wavelet 1-D** tool.

To import and export information from the Complex Continuous Wavelet tool, see "Importing and Exporting Information from the Wavelet Analyzer App" on page 2-14. The only difference is that the variable `coefs` is a complex matrix (see "Saving Wavelet Coefficients" on page 2-14).

DFT-Based Continuous Wavelet Analysis Using the Graphical User Interface

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

1 At the MATLAB command prompt, enter

```
cwtfttool
```
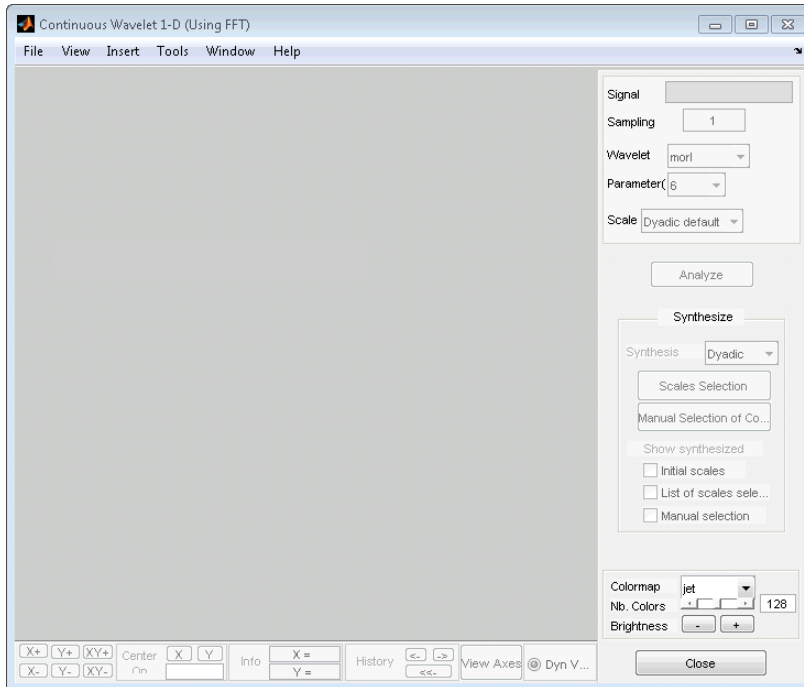

2 At the MATLAB command prompt, type

```
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.
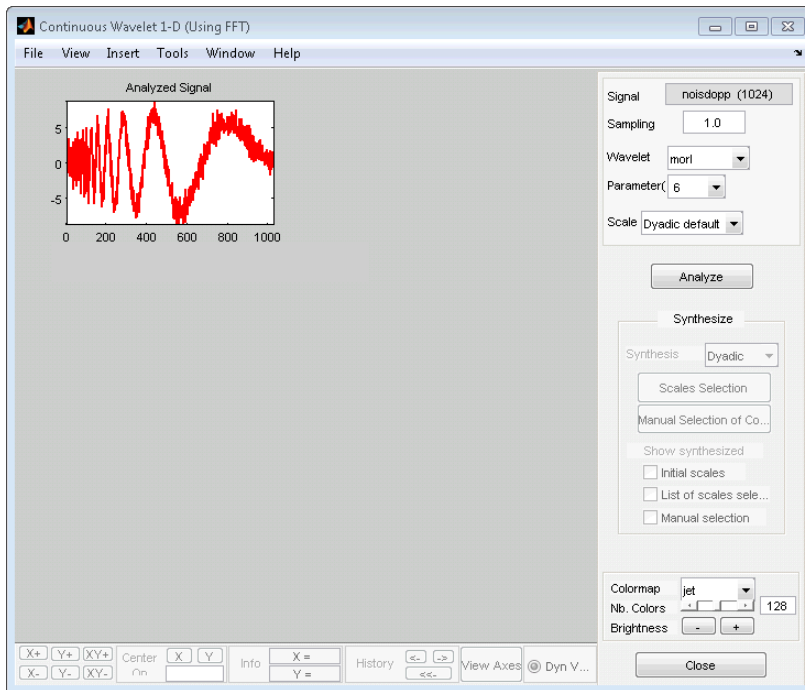
3 Using the menu default parameters, click **Analyze**.
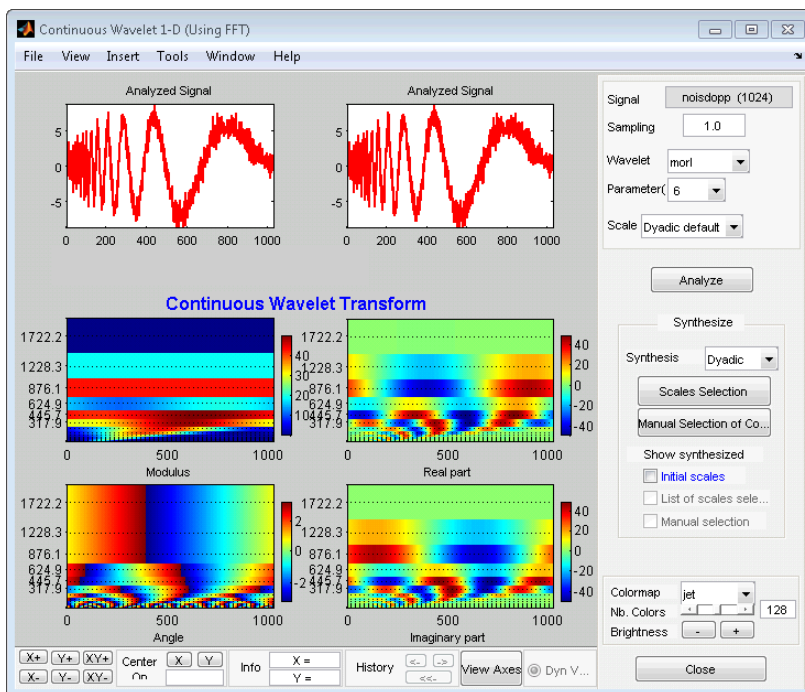
4 Reconstruct the signal based on all the default dyadic scales. Click **Scales Selection**.
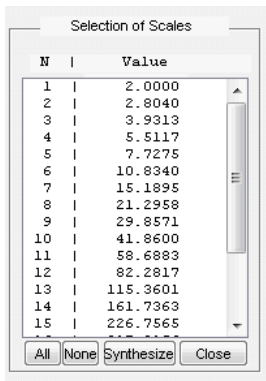
Select all scales by clicking **All**. Click **Synthesize**.
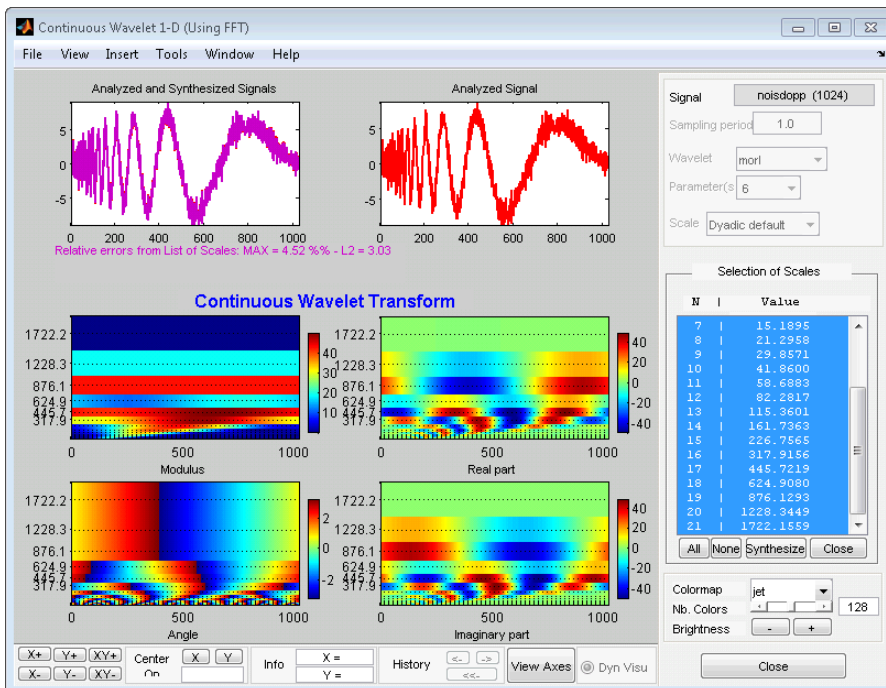
In the top left, the synthesized signal plot is superimposed on the original signal. The relative maximum and L2 errors are displayed under the plot.

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

5 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);
```

6 Click **Synthesize**.

The reconstructed signal from scale indices 11–21 is a lowpass approximation to the noisy Doppler signal.
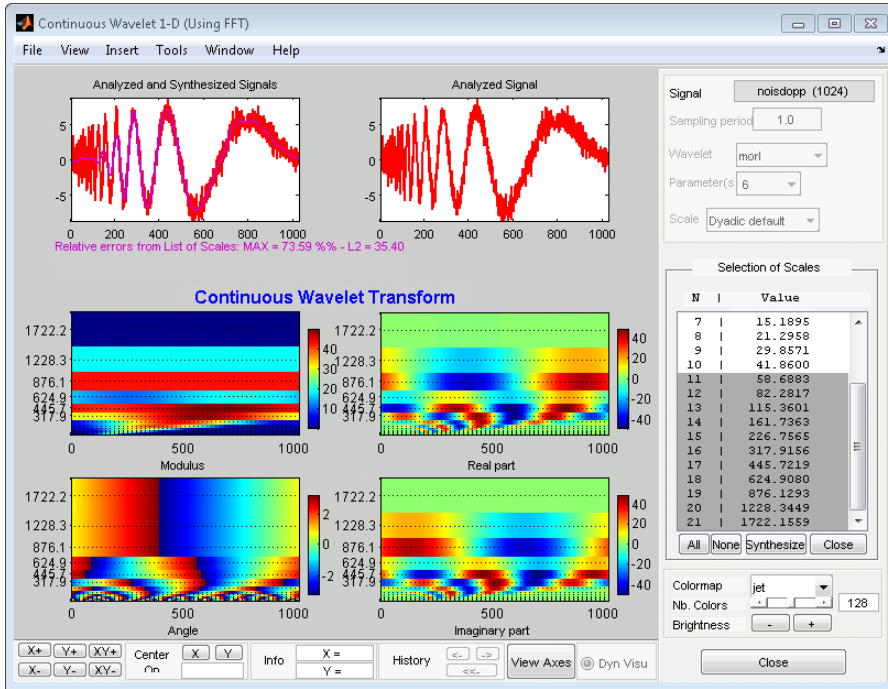
7 Analyze using linear scales. In the **Scales** drop-down menu in the upper right, select `Linear default` and click **Analyze**.
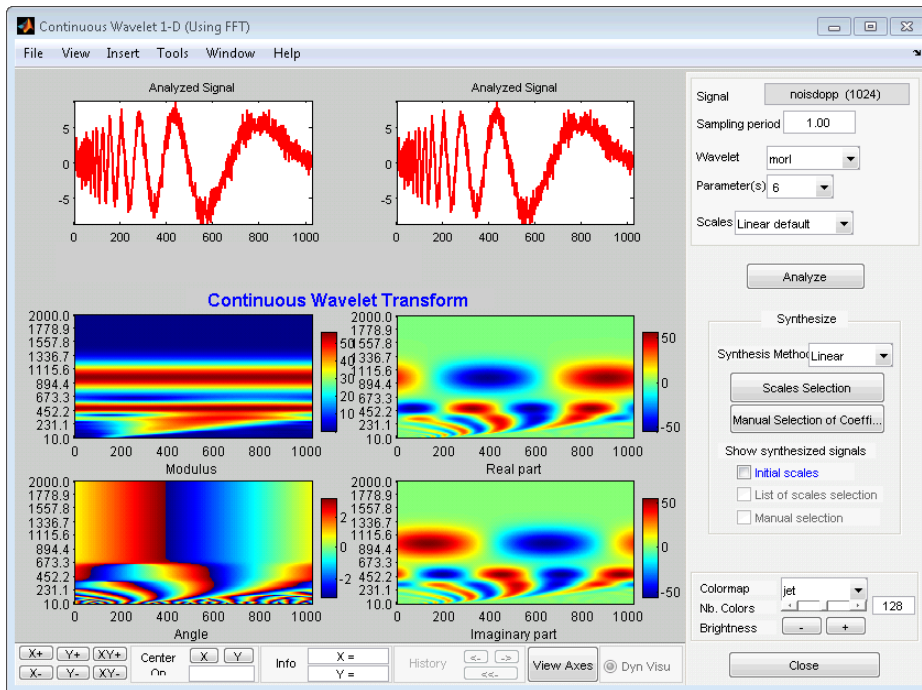
Note The other options under **Scales** include `Dyadic default` and `Manual`.

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

Manual Selection of CWT Coefficients

Select coefficients manually by graphically selecting the CWT coefficients. Reconstruct the signal from the selected coefficients. Click **Manual Selection of Coefficients**. The **Select the Coefficients Manually** panel appears with a single box containing all the CWT coefficient moduli.
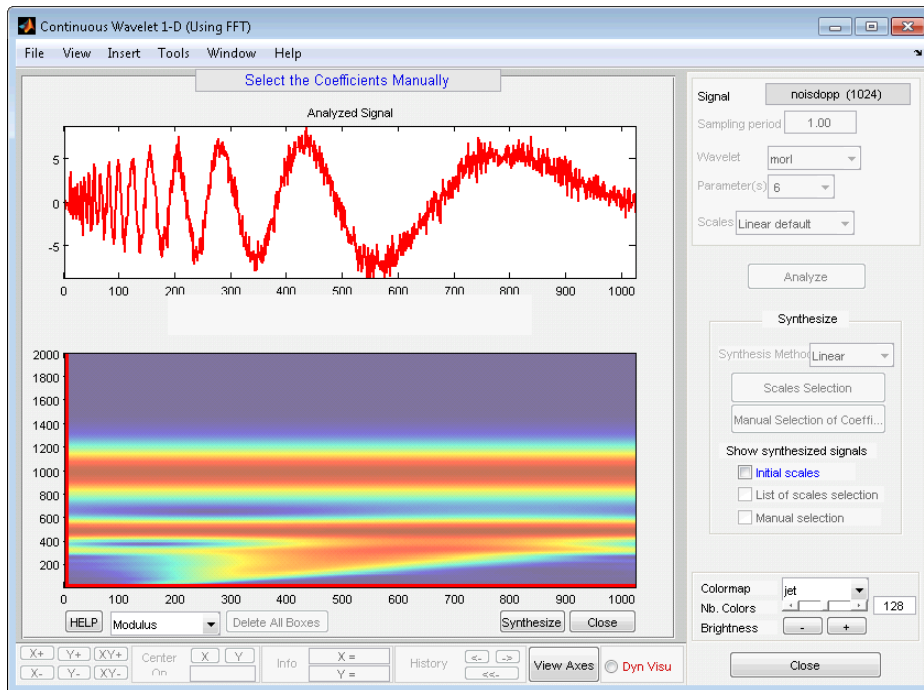
You can change the CWT coefficient view to `Angle`, `Real`, or `Imaginary`.

To select a subset of coefficients, draw a box by left-clicking and dragging the mouse. When you release the mouse button, a semi-transparent box with a green border is superimposed on the plot.
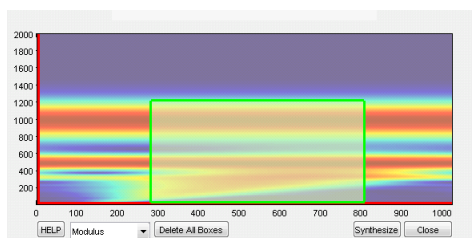
You can place multiple boxes on the same plot. To synthesize a signal based on the selected coefficients, click **Synthesize**.
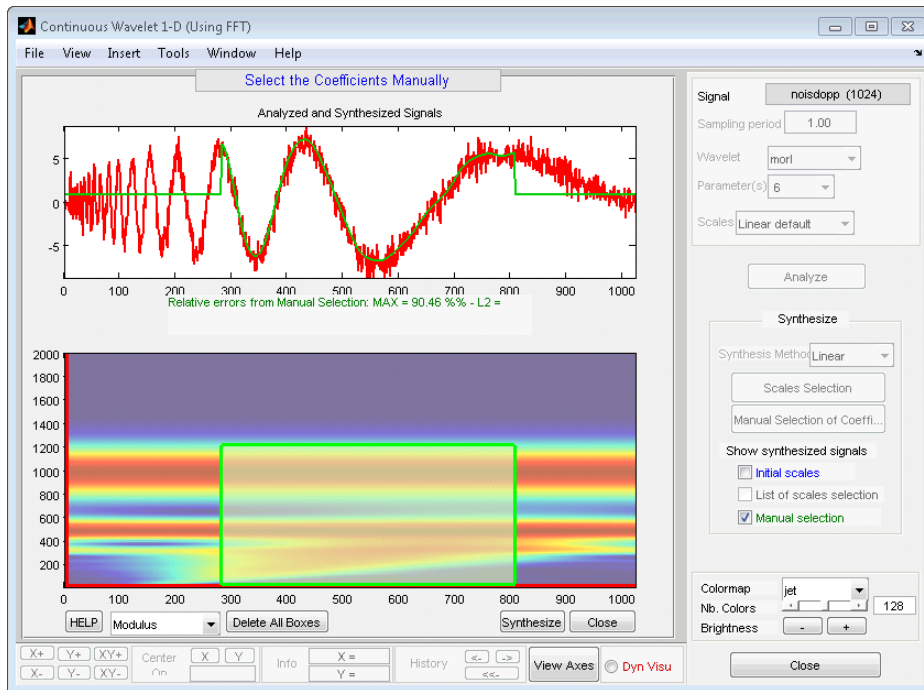
To select, unselect, or delete a box, right-click in the box. A context menu appears that allows you to **select**, **unselect**, or **delete** the box. After you select the coefficients within the box, the border of the box displays in green. When the coefficients within the box are not selected, the border of the box displays in red.

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

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

In the **Show synthesized signals from** panel on the right, you can turn the plot of your synthesized signal on and off by checking and unchecking **Manual selection**.
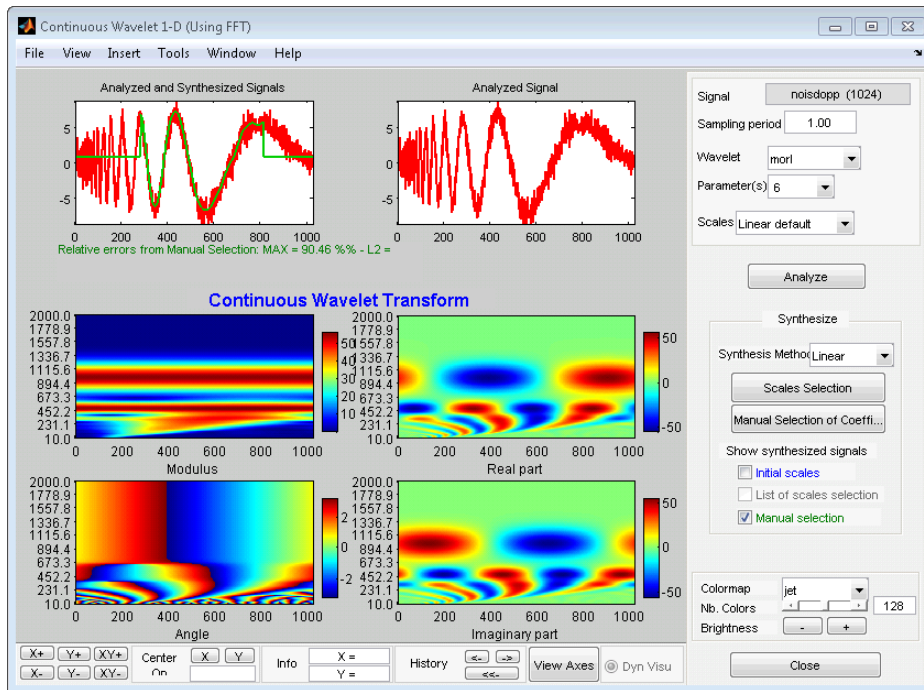
Using the **File > Save > Synthesized signal** menu, you can save the available synthesized signals.

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

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');
```

Angle = 0 Radians

Extract the wavelet corresponding to an angle of *π*/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');
```
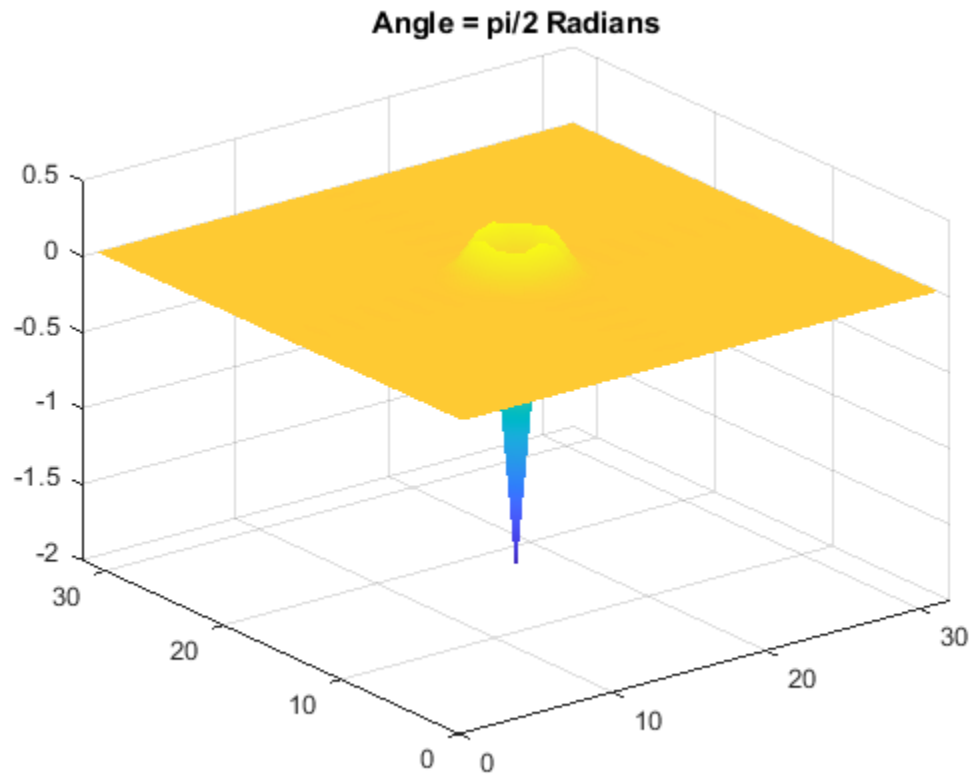
Repeat the preceding steps for the complex-valued Morlet wavelet. The Morlet wavelet has a larger spatial support than the Mexican hat wavelet, therefore this example uses a larger matrix. The wavelet is complex-valued, so the modulus is plotted.

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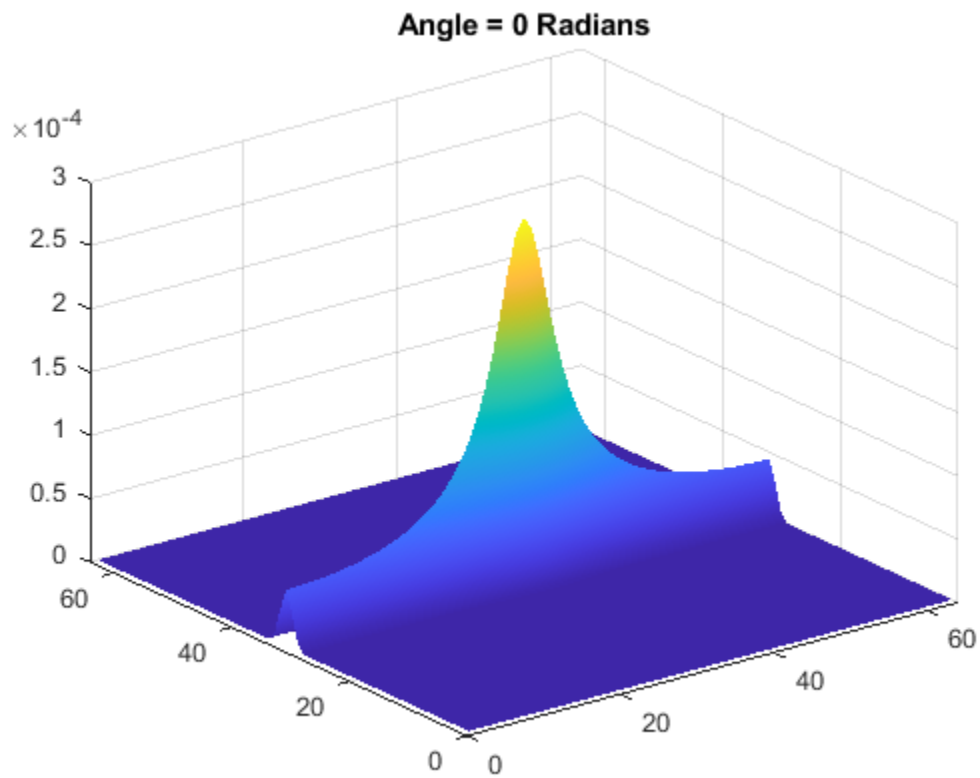
Extract the wavelet corresponding to an angle of $\pi/2$ radians. Unlike the Mexican hat wavelet, the Morlet wavelet is not isotropic and therefore is sensitive to the direction of features in the data.

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

Angle = pi/2 Radians
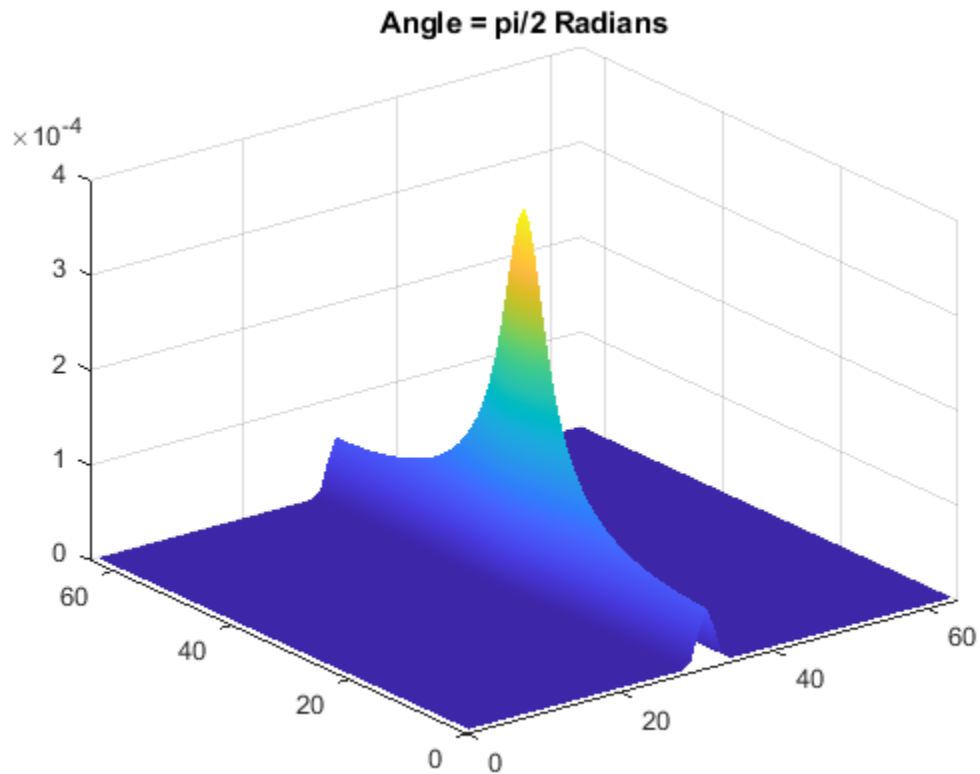
Apply the Mexican hat and Morlet wavelets to the detection of a pattern in noise. Create a pattern consisting of line segments joined at a 90-degree angle. The amplitude of the pattern is 3 and it occurs in additive N(0,1) white Gaussian noise.

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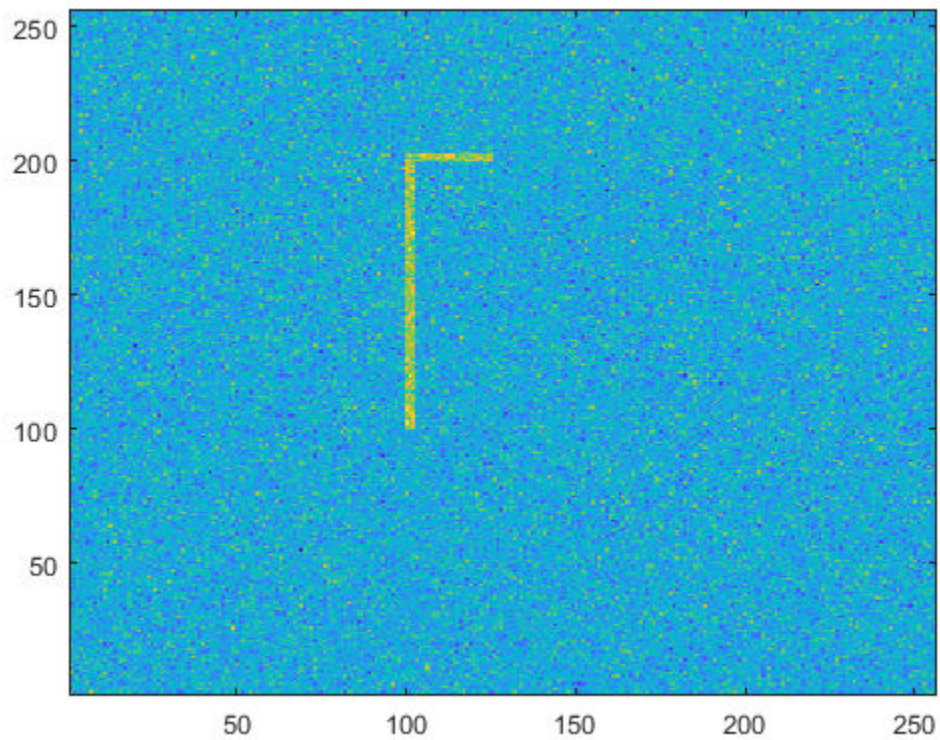
Obtain the 2-D CWT at scales 3 to 8 in 0.5 increments with the Mexican hat wavelet. Visualize the magnitude-squared 2-D wavelet coefficients at scale 3.

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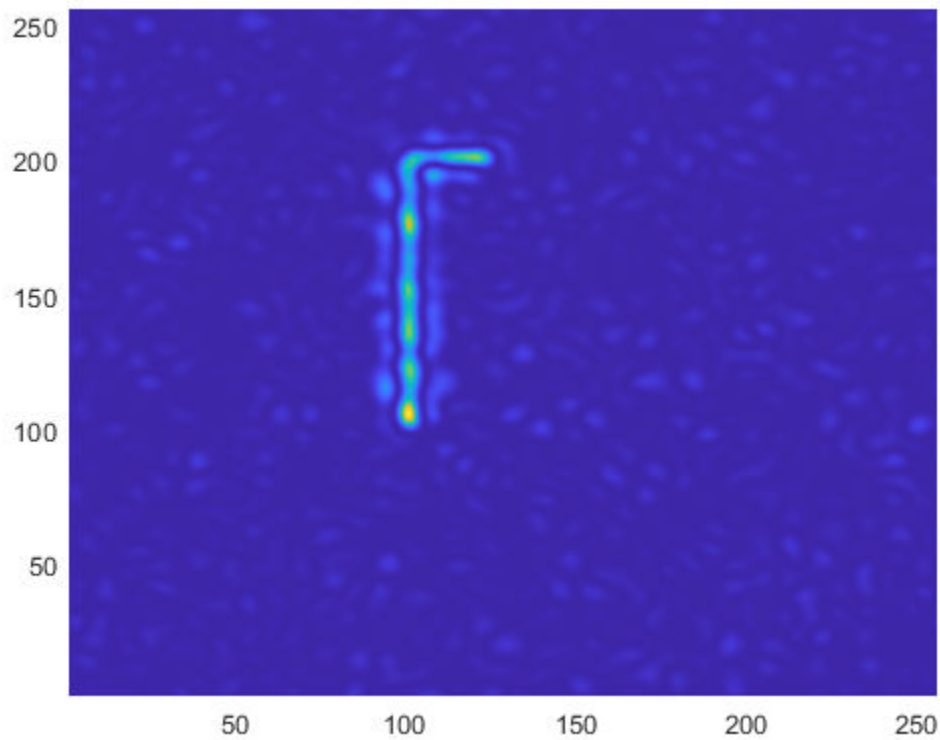
Use a directional Morlet wavelet to extract the vertical and horizontal line segments separately. The vertical line segment is extracted by one angle. The horizontal line segment is extracted by another angle.

```
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;
```
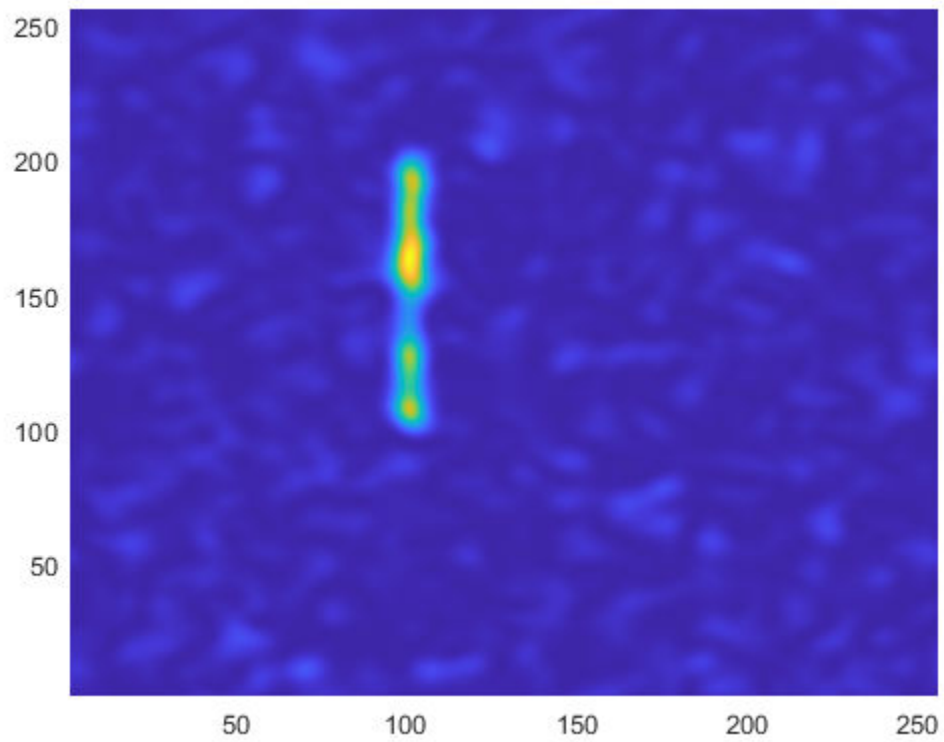
```
figure;
surf(abs(cwtmorl.cfs(:,:,1,4,2)).^2);
view(0,90); shading interp; axis tight;
```
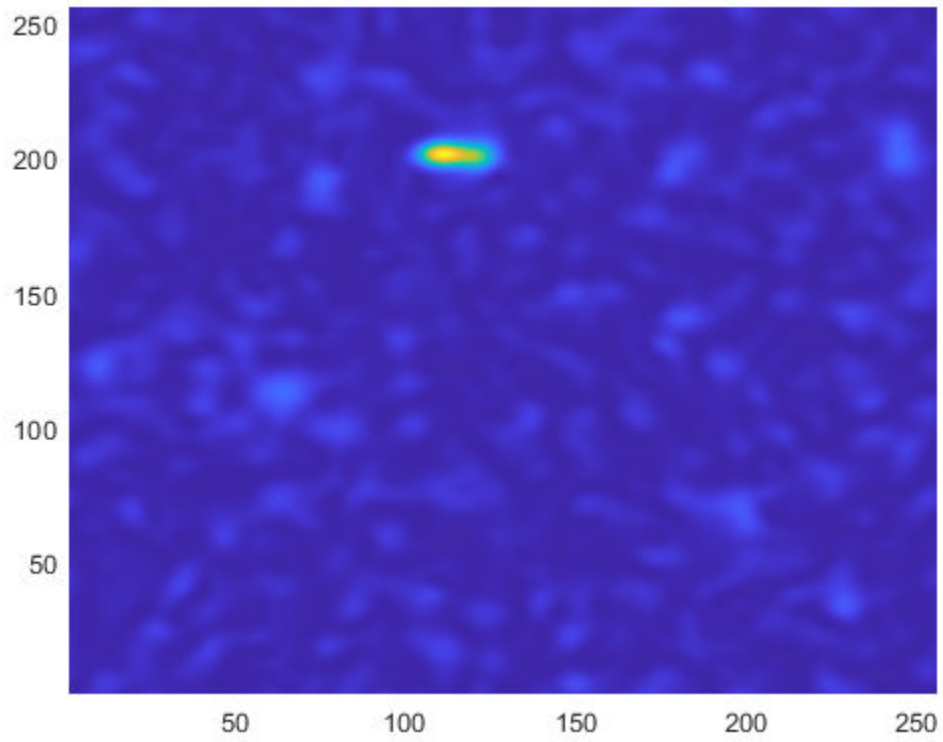
2-D Continuous Wavelet Transform App

In this section...
"2-D Continuous Wavelet Transform" on page 2-79
"2-D CWT App Example" on page 2-79

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

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

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 2-71 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

```
cwtfttool2
```

at the MATLAB command prompt.

Select **File -> Import Data** to import the `imdata` variable.
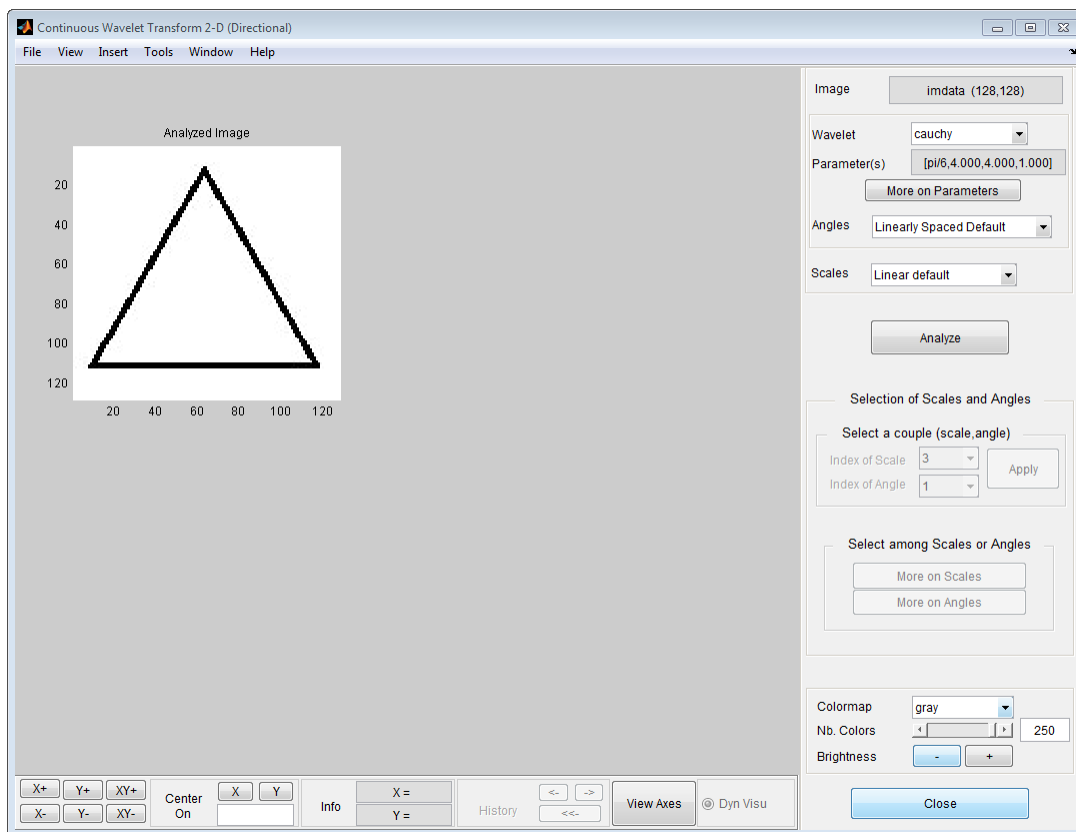
From the `Wavelet` drop down menu, select the **cauchy** wavelet.

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*pi/8` radians in increments of `pi/8` radians, `0:pi/8:(7*pi)/8`. Click **Apply** to apply your choice of angles.
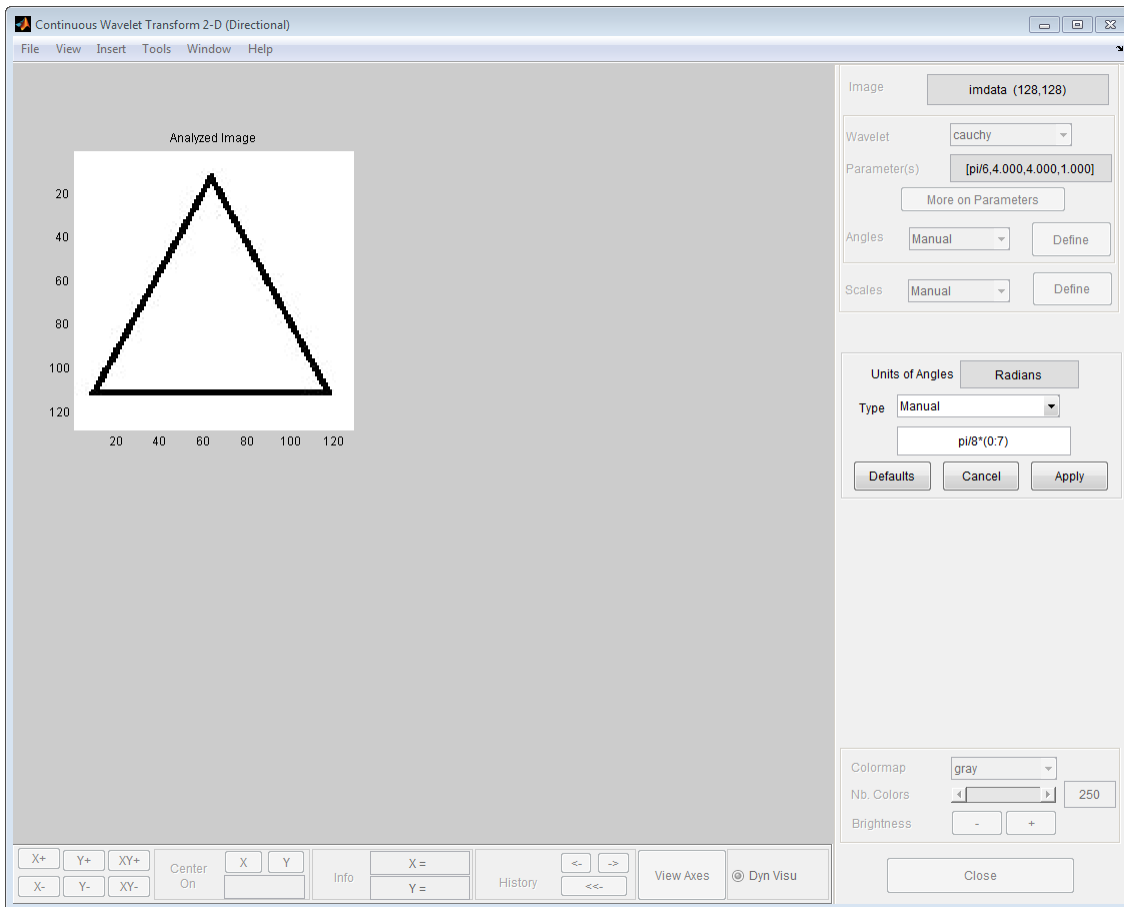
Click **Define** to specify a vector of scales from 0.5 to 4 in increments of 0.5. Select **Linear** from the `Type` drop-down list. Set **First Scale** equal to 0.5, **Gap between two scales** equal to 0.5, and **Number of Scales** equal to 8. Equivalently, you can select **Manual** from the `Type` 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.
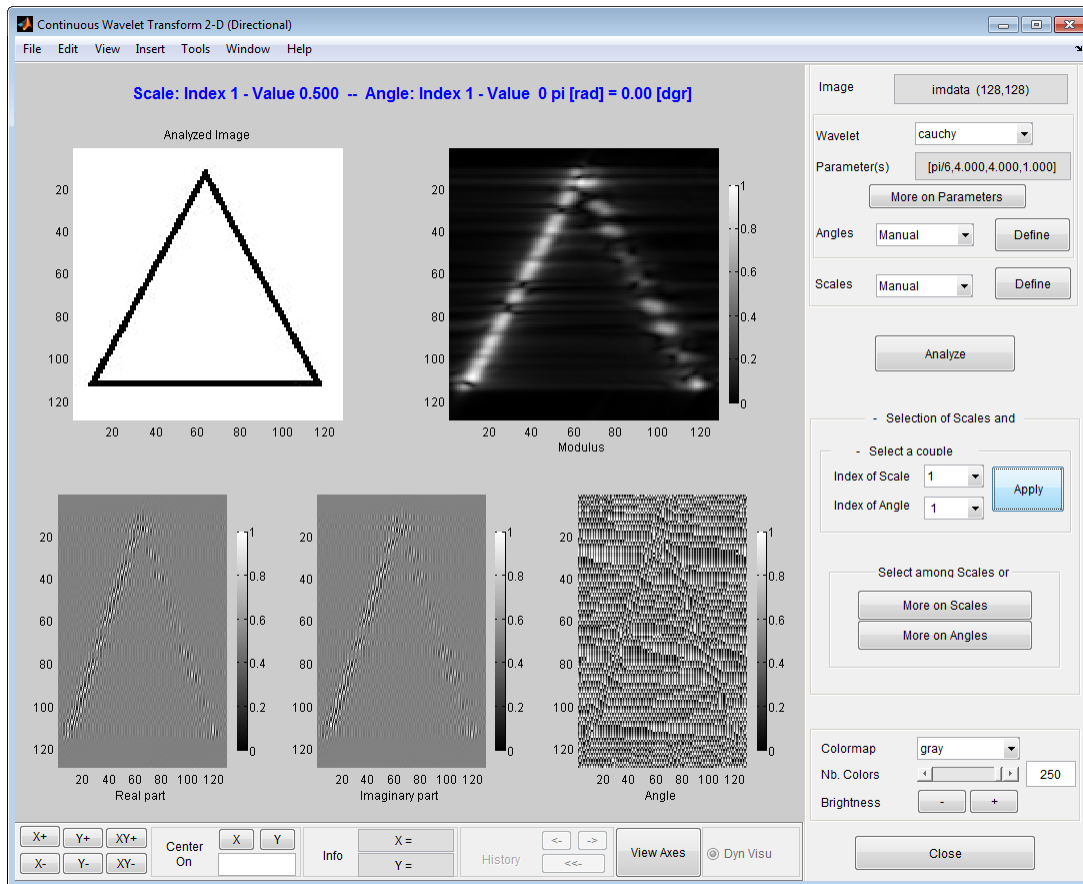
Set the `Index of Scale` to be 1 and click **More on Angles**. Click **Movie** to step through the manually-defined angles for the 2-D CWT coefficients at scale 0.5.

Select **File -> Export Data -> Export CWTFT 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

In this section...
"Double-Density Wavelet Transform" on page 3-3
"Dual-Tree Complex Wavelet Transform" on page 3-5
"Dual-Tree Double-Density Wavelet Transforms" on page 3-7

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.
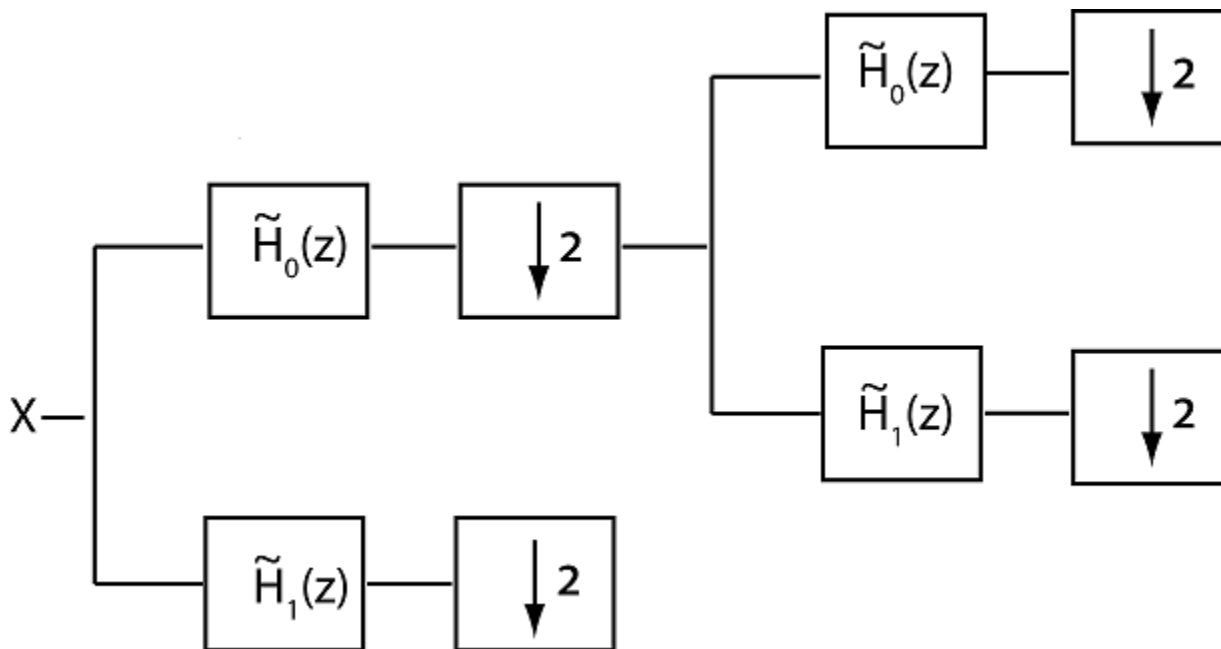
The filter system functions, $\tilde{H}_0(z)$ and $\tilde{H}_1(z)$, are typically designed to approximately partition the input signal, X, into disjoint subbands. In wavelet tree-structured filter banks, the filter $\tilde{H}_0(z)$ is a lowpass, or scaling, filter, with a non-zero frequency response on the interval [-π/2, π/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 [-π, -π/2] ∪ [π/2, π] radians/sample or [-1/2, -1/4] ∪ [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.
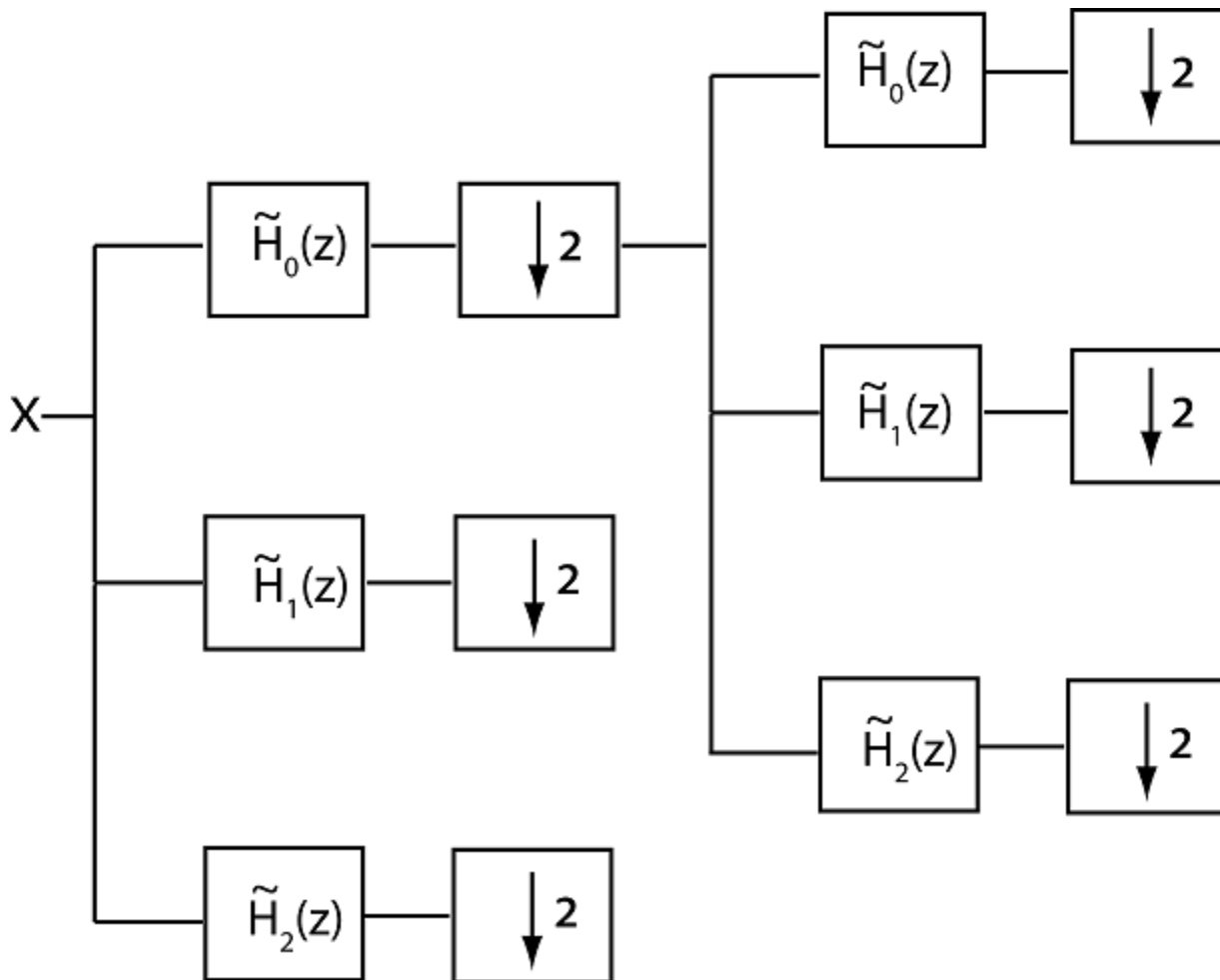
Assume the filter $\tilde{H}_0(z)$, is a lowpass half-band filter and the filters $\tilde{H}_1(z)$ and $\tilde{H}_2(z)$ are highpass half-band filters.

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)

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

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

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

from which it follows that $\left| H_0(e^{j\omega}) \right|^2 = 2 \quad 0 \le \omega \le \pi$. 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.
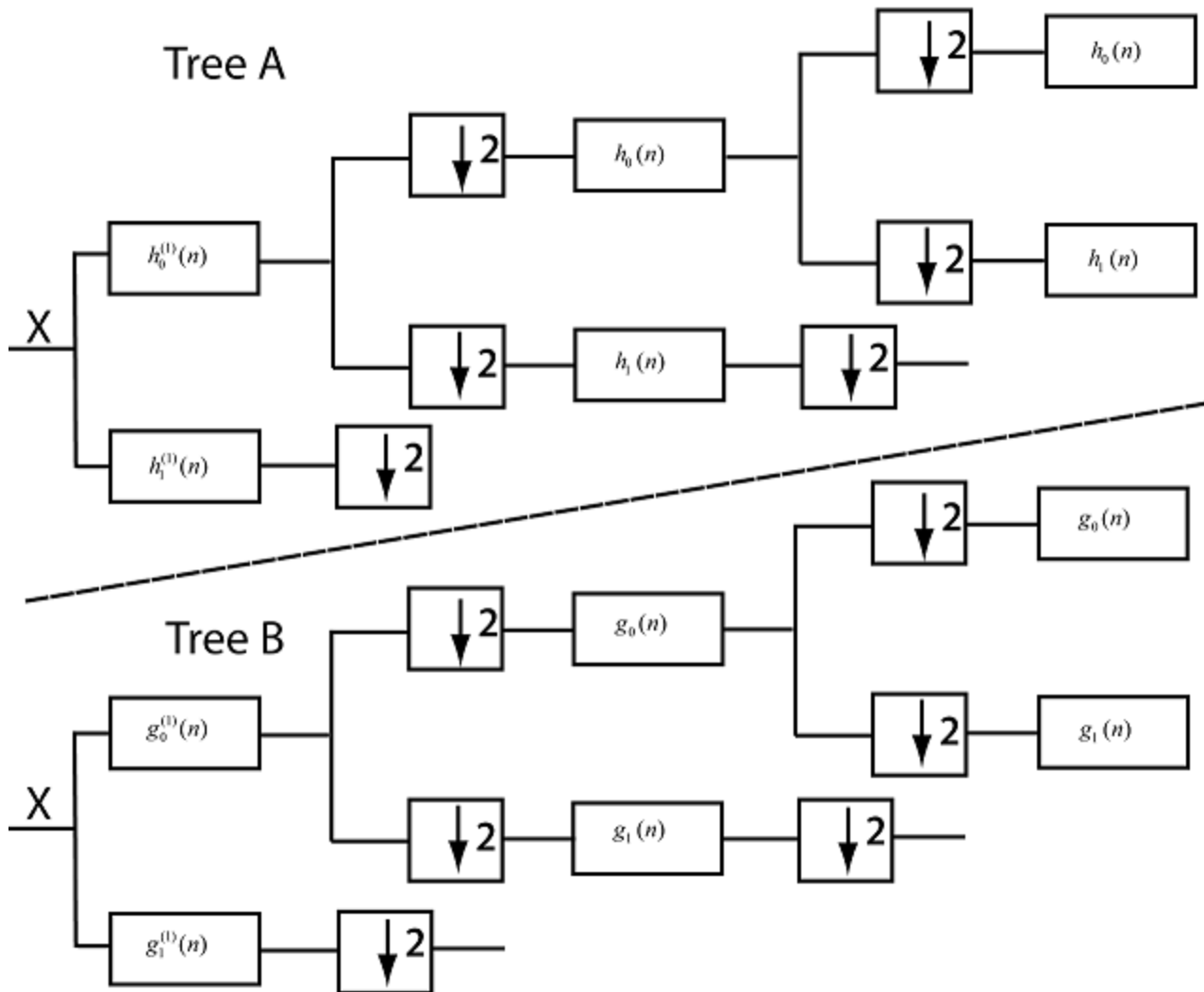
2-D Dual-Tree Wavelet Transforms

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 `'cplxdddt'` 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 `'realdddt'` option in `dddtree2`. To obtain the complex oriented dual-tree double-density wavelet transform, use the `'cplxdddt'` 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.

See Also

dddtree | dddtree2 | dualtree | dualtree2

More About

- "Dual-Tree Complex Wavelet Transforms" on page 3-191

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
wdenoise	Automatic wavelet signal denoising (recommended)
wdenoise2	Automatic wavelet image denoising (recommended)
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.

Note To denoise a signal, `wdenoise` and **Wavelet Signal Denoiser** are recommended.

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')
```
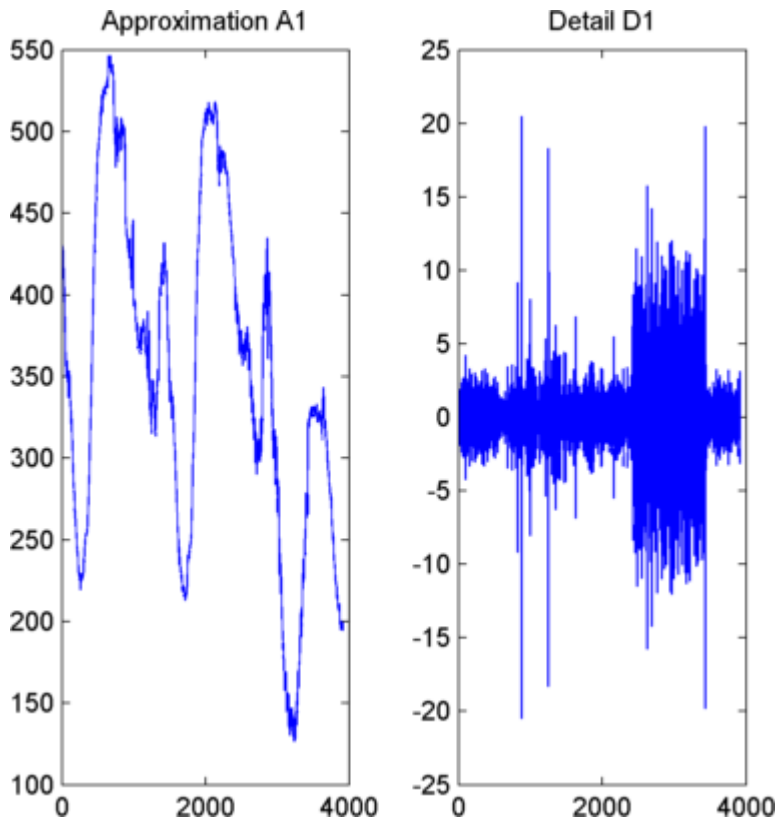

5   Regenerate a signal by using the Inverse Wavelet Transform.

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.
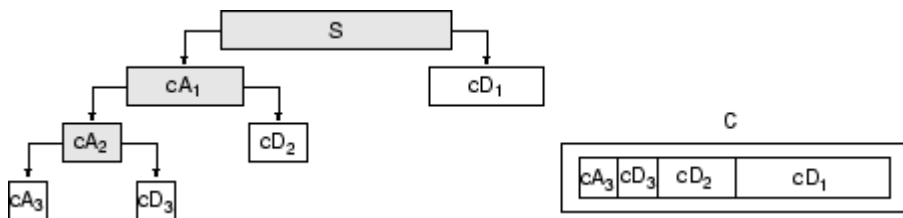
7 Extract approximation and detail coefficients.

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')
```
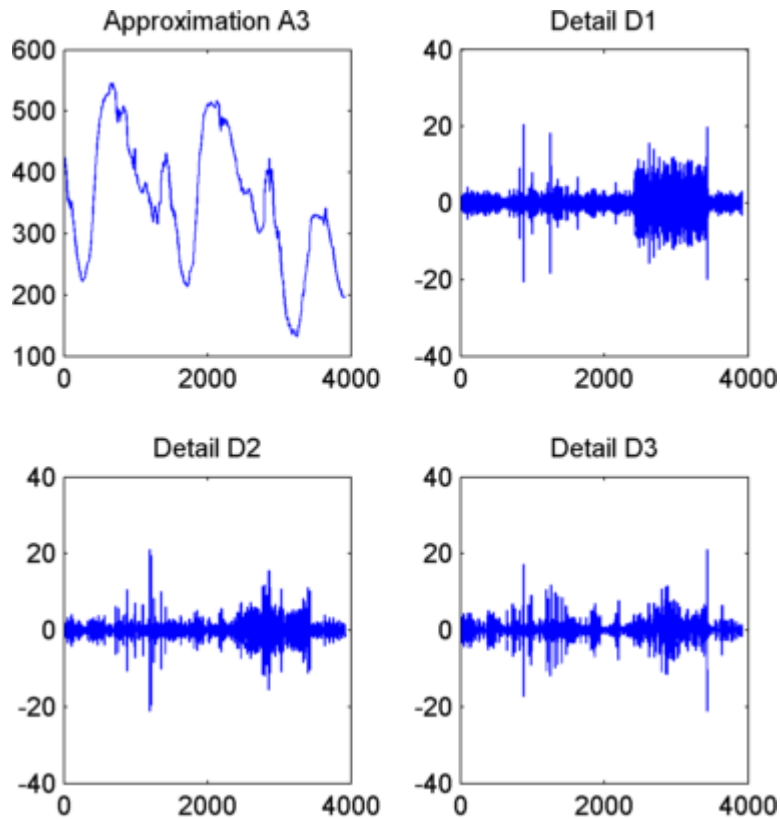
10 Reconstruct the original signal from the Level 3 decomposition.

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
```

Original

Level 3 Approximation
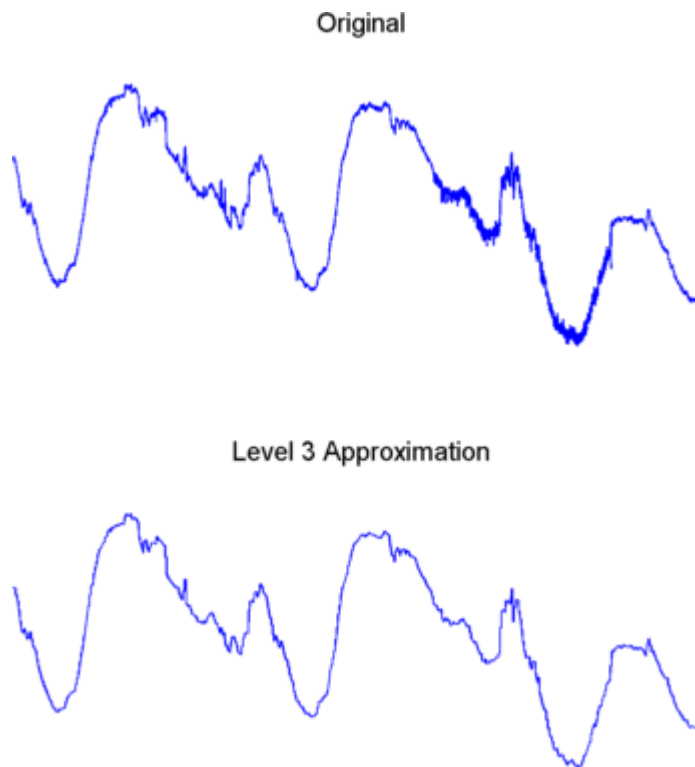
Of course, in discarding all the high-frequency information, we've also lost many of the original signal's sharpest features.

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 `wdencmp` command to perform the actual denoising, type

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

Note that `wdencmp` uses the results of the decomposition (`C` and `L`) that we 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 `wdencmp` in the reference pages for more information about the use of these commands.

To display both the original and denoised signals, type

```
subplot(2,1,1); plot(s(2000:3920)); title('Original')
subplot(2,1,2); plot(clean(2000:3920)); title('denoised')
```
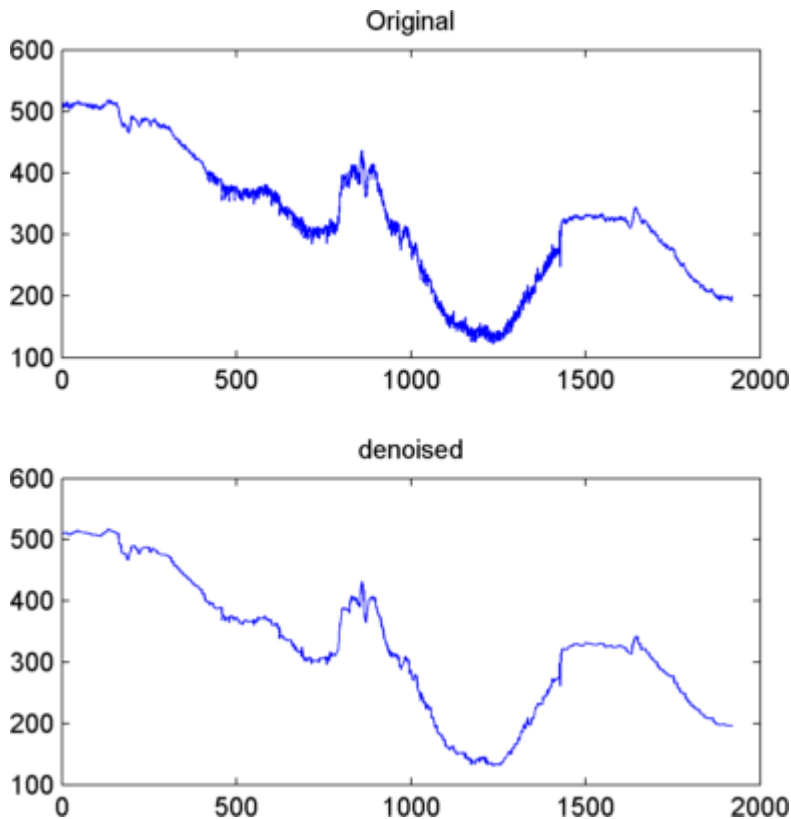
We've plotted here only the noisy latter part of the signal. Notice how we've removed the noise without compromising the sharp detail of the original signal. This is a strength of wavelet analysis.

While using command line functions to remove the noise from a signal can be cumbersome, the software's 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 3-16
- "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 6-39
- "1-D Wavelet Variance Adaptive Thresholding" on page 6-13 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.

Note Using the Wavelet Analyzer app to denoise a signal is no longer recommended. Use **Wavelet Signal Denoiser** instead.
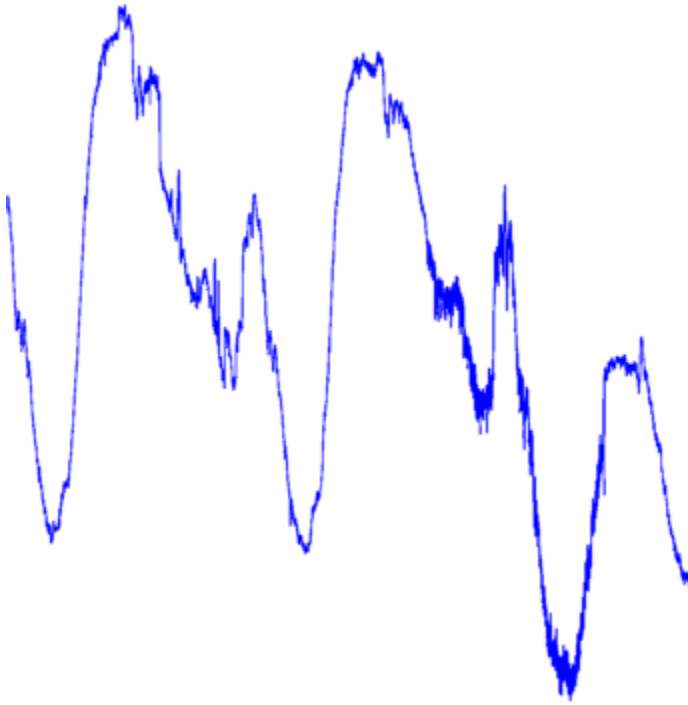
1   Start the 1-D Wavelet Analysis Tool.

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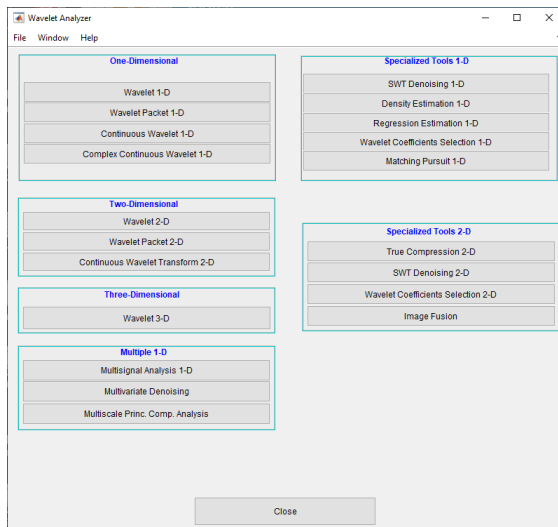
In the **Wavelet 1-D** tool, select **File > Import from Workspace**. When the **Import from Workspace** dialog box appears, select the `leleccum` variable. Click **OK** to import the electrical consumption signal.

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 3-10.

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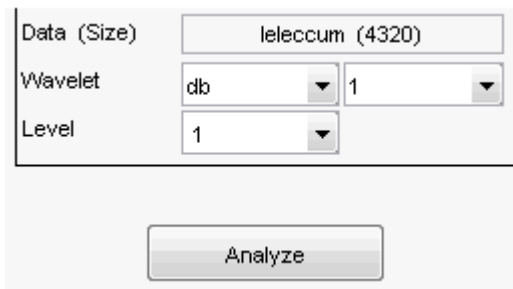
Click the **Analyze** button.

After a pause for computation, the tool displays the decomposition.
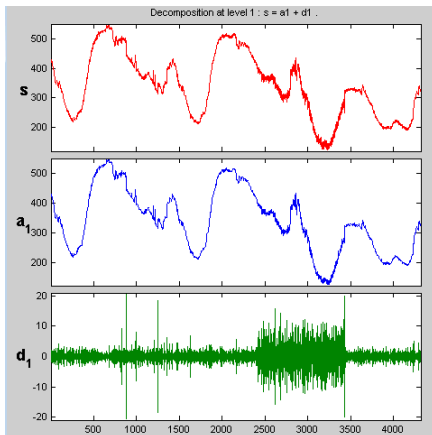
4 Zoom in on relevant detail.

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.
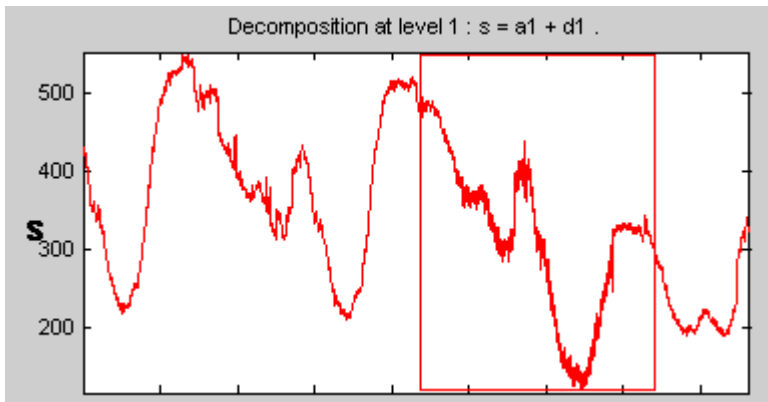
Click the **X+** button (located at the bottom of the screen) to zoom horizontally.
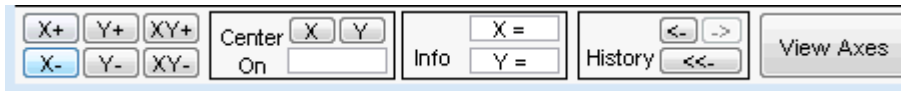
The **Wavelet 1-D** tool zooms all the displayed signals.
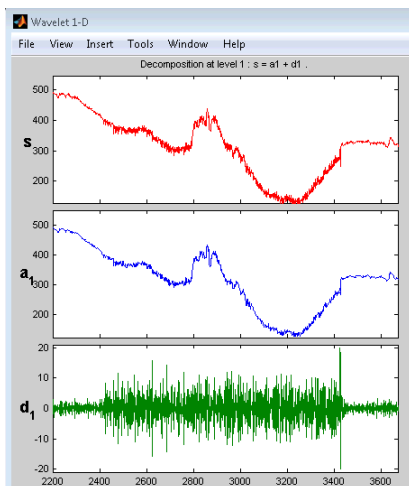
The other zoom controls do more or less what you'd expect them to. The **X-** button, for example, zooms out horizontally. The history function keeps track of all your views of the signal. Return to a previous zoom level by clicking the left arrow button.

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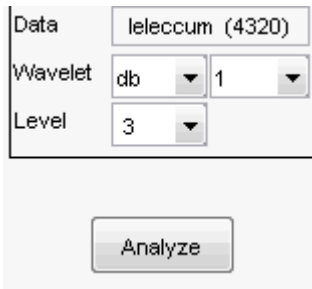
After the decomposition is performed, you'll see a new analysis appear in the **Wavelet 1-D** tool.

Selecting Different Views of the Decomposition

The **Display mode** menu (middle right) lets you choose different views of the wavelet decomposition.
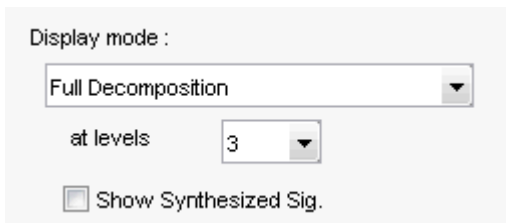
The default display mode is called "Full Decomposition Mode." Other alternatives include:

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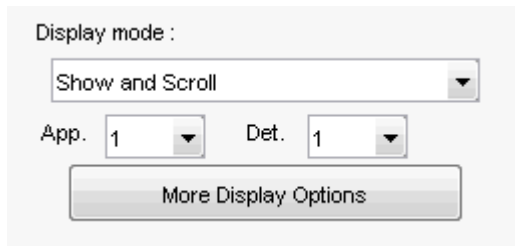
These options include the ability to suppress the display of various components, and to choose whether or not to display the original signal along with the details and approximations.

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.

Note The denoising option is no longer recommended. Use **Wavelet Signal Denoiser** instead.

Bring up the denoising tool: click the **denoise** button, located in the middle right of the window, underneath the **Analyze** button.
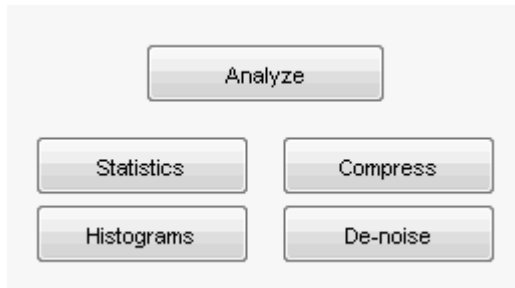
The **Wavelet 1-D Denoising** window appears.

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. The `Unscaled white noise` option corresponds to setting the multiplicative threshold input argument `SCAL` of `wden` to `'one'`. Choosing `Scaled white noise` corresponds to `'sln'`, and `Non-white noise` corresponds to `'mln'`. For more information, see `wden`.

Continue by clicking the **denoise** button.

The denoised signal appears superimposed on the original. The tool also plots the wavelet coefficients of both signals. The original detail 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.
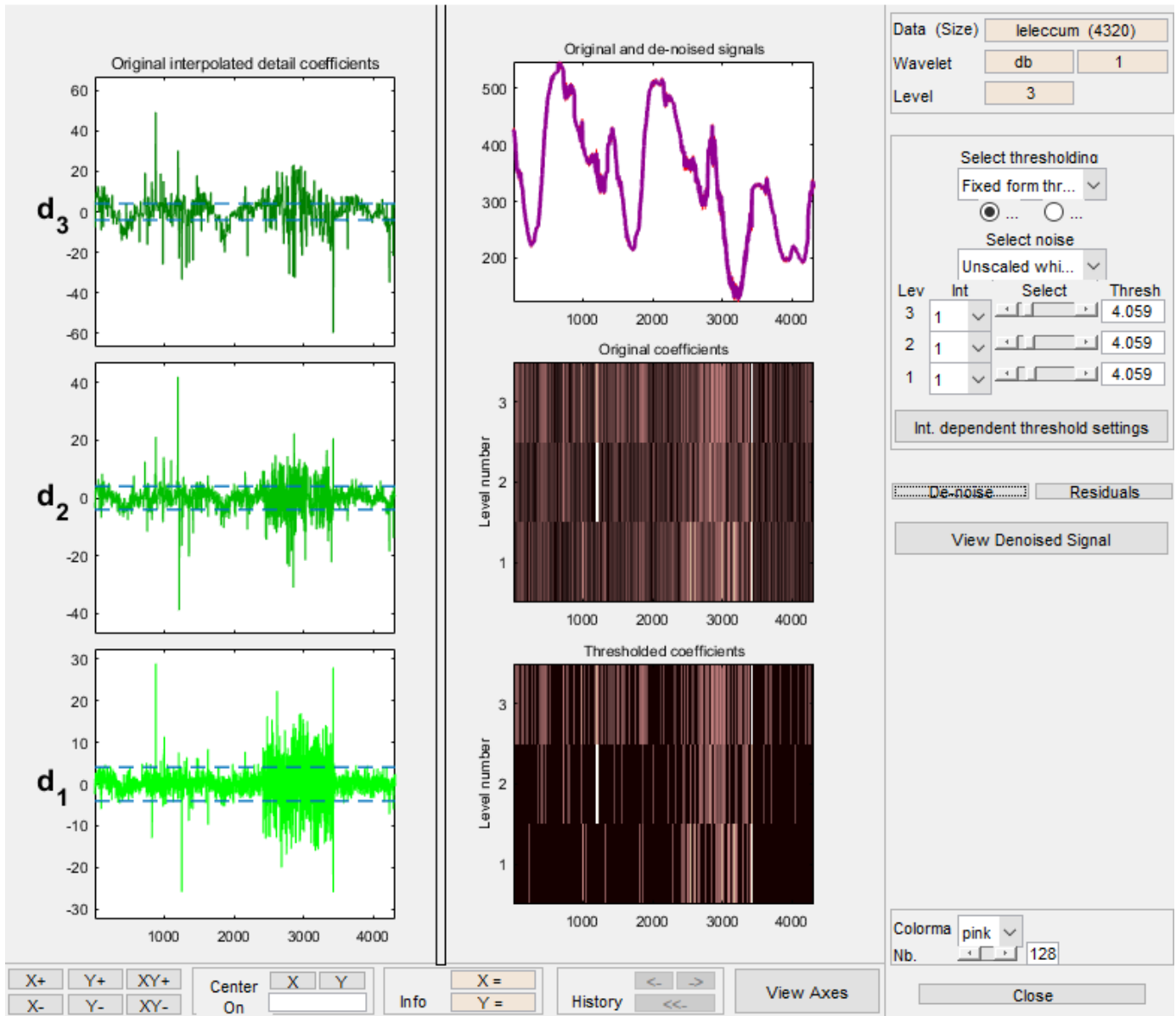
Zoom in on the plot of the original and denoised signals for a closer look.

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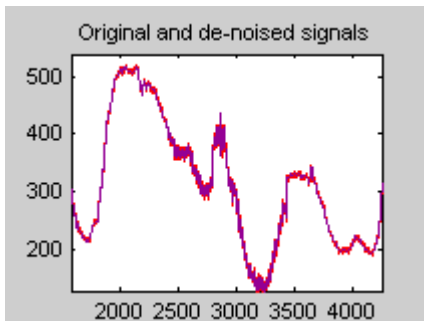
Dismiss the **Wavelet 1-D Denoising** window: click the **Close** button.

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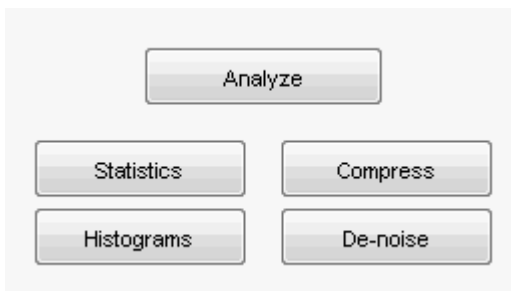
Bring up the **Compression** window: click the **Compress** button, located in the middle right of the window, underneath the **Analyze** button.

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

3-23

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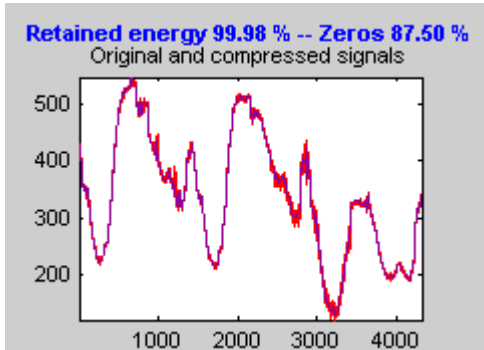
You can see that the compression process removed most of the noise, but preserved 99.99% of the energy of the signal.

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.
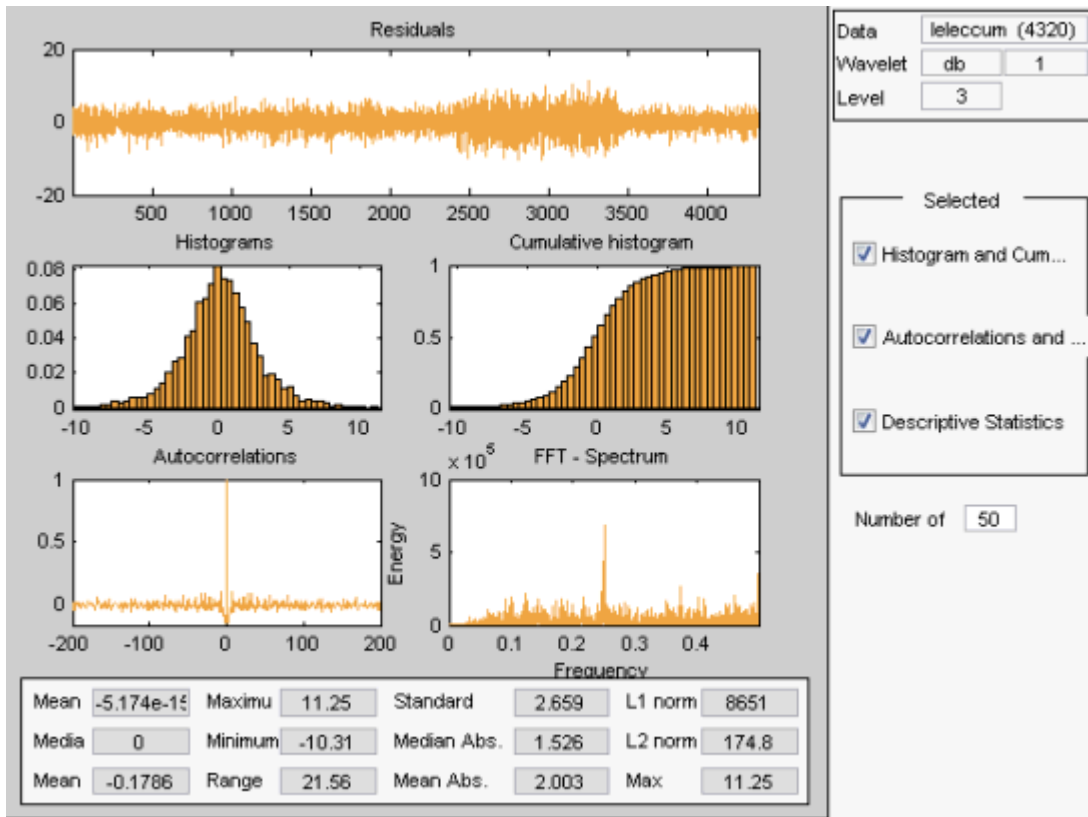
Displayed statistics include measures of tendency (mean, mode, median) and dispersion (range, standard deviation). In addition, the tool provides frequency-distribution diagrams (histograms and cumulative histograms), as well as time-series diagrams: autocorrelation function and spectrum. The same feature 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.
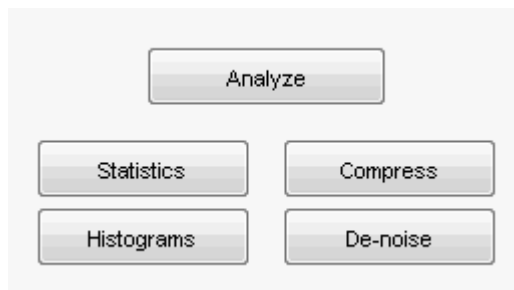
The **Wavelet 1-D Statistics** window appears displaying by default statistics on the original signal.
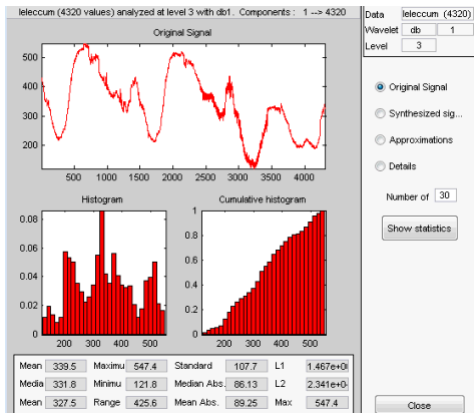
Select the synthesized signal or signal component whose statistics you want to examine. Click the appropriate option button, and then click the **Show Statistics** button. Here, we've chosen to examine the synthesized signal using 100 bins instead of 30, which is the default:
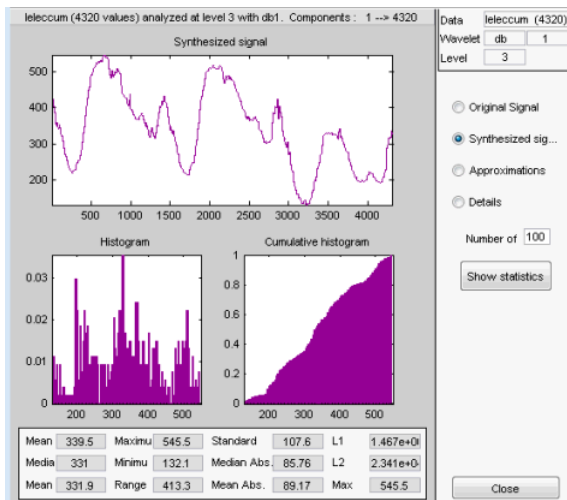
Displayed statistics include measures of tendency (mean, mode, median) and dispersion (range, standard deviation).

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.
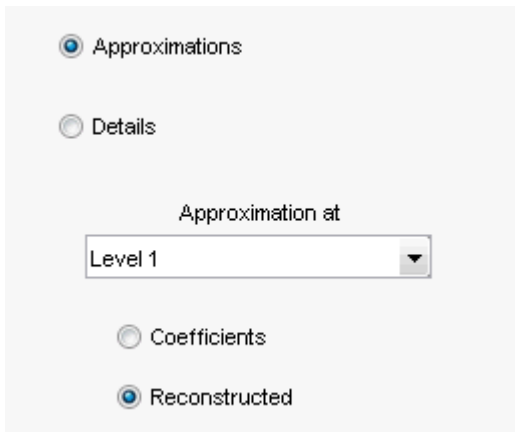
Select Level 1 and again click the **Show Statistics** button. Statistics appear for the level 1 approximation.
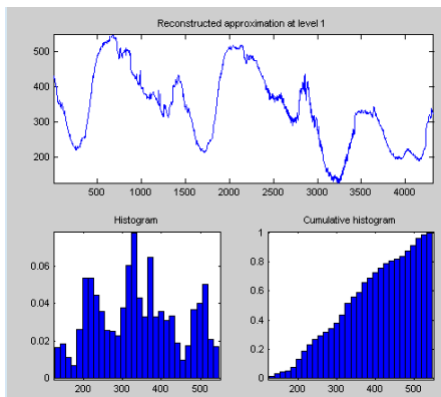
Importing and Exporting Information from the Wavelet Analyzer App

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.
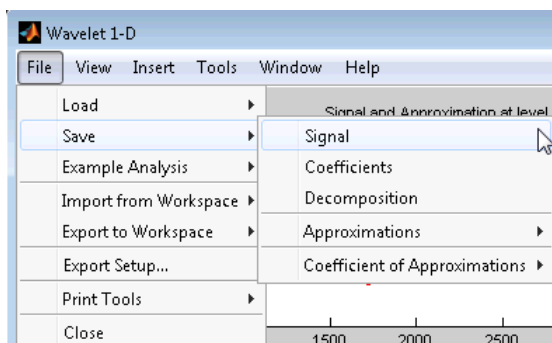
Saving Synthesized Signals

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 6-39).

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
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.
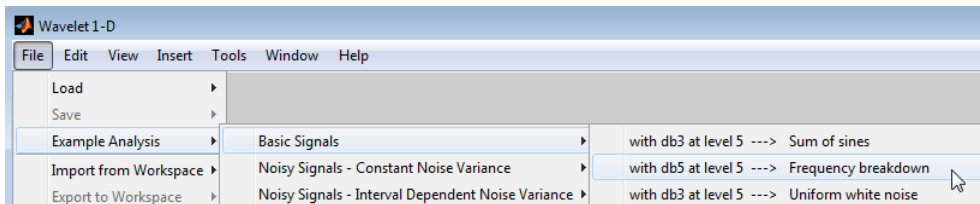
After the wavelet 1-D analysis loads, select File -> Export to Workspace.
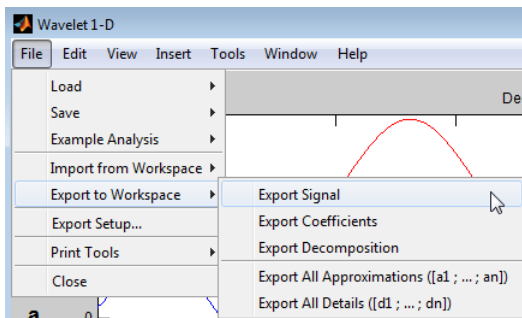
You have the option to

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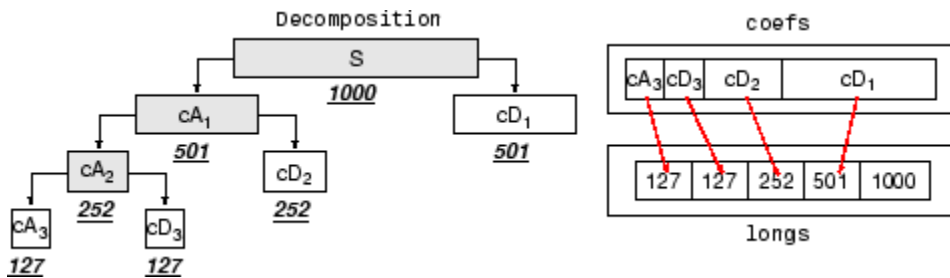
After constructing or editing the appropriate data in your workspace, type

```
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 `wa1` or other).

The MAT-file contains the following variables.

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

See Also
Wavelet Signal Denoiser | `wdenoise`

More About
- "Denoise a Signal with the Wavelet Signal Denoiser" on page 6-25

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

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):

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

All the filters used in DWT and IDWT are intimately related to the sequence

$(w_n)_{n \epsilon Z}$

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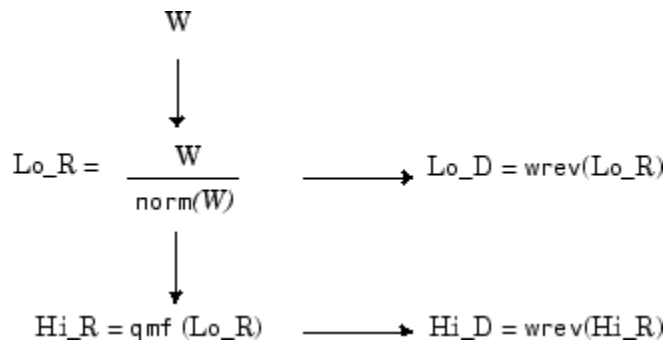
where `qmf` is such that Hi_R and Lo_R are quadrature mirror filters (i.e., Hi_R(*k*) = (–1) ^{k} Lo_R(2*N* + 1 – *k*)) for *k* = 1, 2, ..., 2*N*.

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');
```
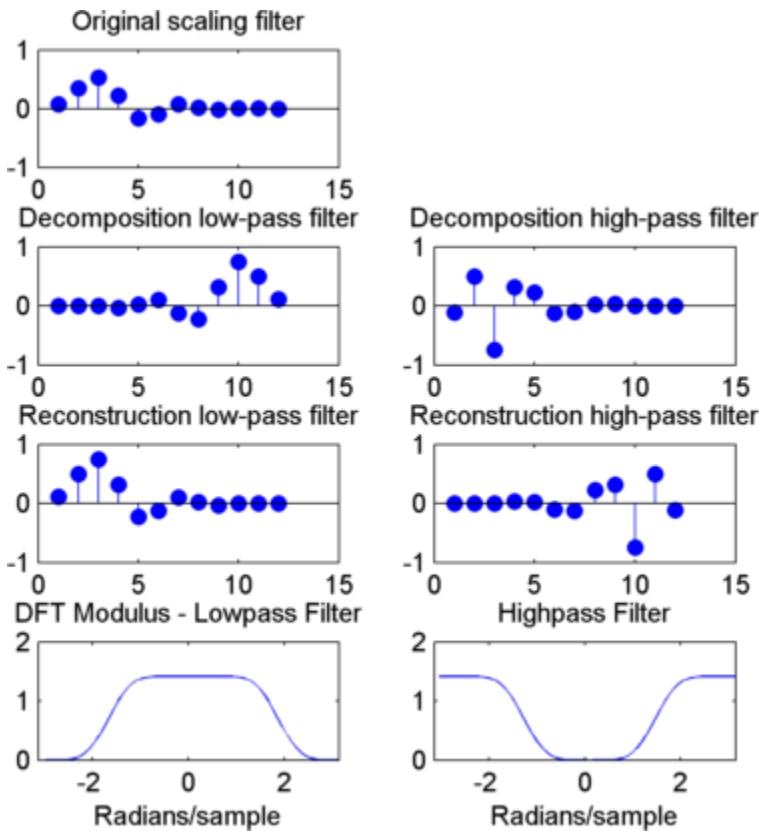

Algorithms

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

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
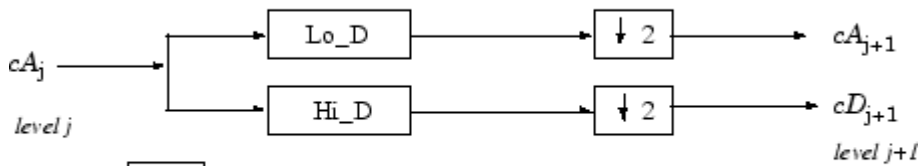
where | X | Convolve with filter X.

| ↓ 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, ..., cD_1].

This structure contains for J = 3 the terminal nodes of the following tree.
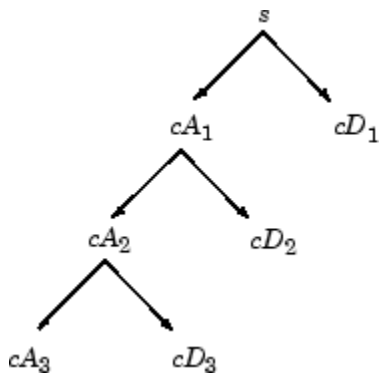
- Conversely, starting from cA_j and cD_j, the IDWT reconstructs cA_{j-1}, inverting the decomposition step by inserting zeros and convolving the results with the reconstruction filters.

One-Dimensional IDWT

Reconstruction Step

cA_j → upsample [↑ 2] → low-pass [Lo_R] ⟶ [wkeep] → cA_{j-1}

cD_j → [↑ 2] upsample → [Hi_R] high-pass

level j   *level j-1*
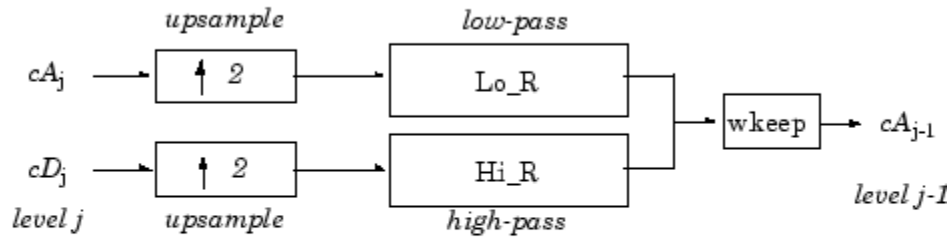
where
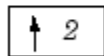
[↑ 2] Insert zeros at odd-indexed elements.
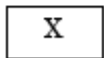
[X] Convolve with filter X.
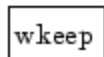
[wkeep] Take the central part of U with the convenient length.

- For images, a similar algorithm is possible for two-dimensional wavelets and scaling functions obtained from 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
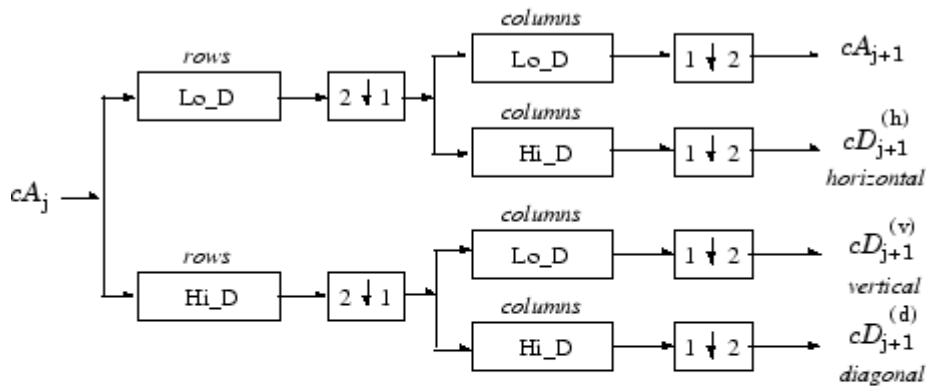
where
$\boxed{2\downarrow1}$ Downsample columns: keep the even indexed columns.

$\boxed{1\downarrow2}$ Downsample rows: keep the even indexed rows.

$\boxed{\overset{rows}{X}}$ Convolve with filter X the rows of the entry.

$\boxed{\overset{columns}{X}}$ Convolve with filter X the columns of the entry.

Initialization $\quad CA_0 = s$ for the decomposition initialization.

Two-Dimensional IDWT

Reconstruction Step
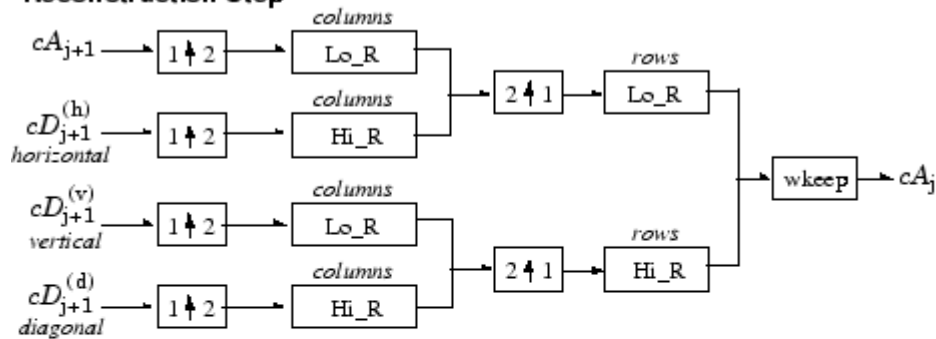
where
$\boxed{2\uparrow1}$ Upsample columns: insert zeros at odd-indexed columns.

$\boxed{1\uparrow2}$ Upsample rows: insert zeros at odd-indexed rows.

$\boxed{\overset{rows}{X}}$ Convolve with filter X the rows of the entry.

$\boxed{\overset{columns}{X}}$ Convolve with filter X the columns of the entry.

So, for $J = 2$, the 2-D wavelet tree has the following form.
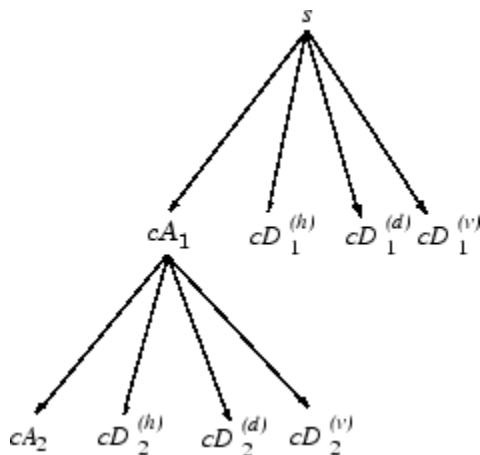
Finally, let us mention that, for biorthogonal wavelets, the same algorithms hold but the decomposition filters on one hand and the reconstruction filters on the other hand are obtained from two distinct scaling functions associated with two multiresolution analyses in duality.

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 3-45.

Let us denote h = Lo_R and g = 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 $(A_k^{(j)})k \epsilon Z$ be the coordinates of the vector A_j:

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

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

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

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

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

This formula resembles a convolution formula.

The computation is very simple.

Let us define

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

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

We obtain

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

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

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

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

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

Let us now describe some of them.

1   The family $(\phi_{0,k}, k \in Z)$ is formed of orthonormal functions. As a consequence for any j, the family $(\phi_{j,k}, k \in Z)$ is orthonormal.

2   The double indexed family

$(\psi_{j,k}, j \in Z, k \in Z)$

is orthonormal.

3   For any j, the $(\phi_{j,k}, k \in Z)$ are orthogonal to $(\psi_{j',k}, j' \le j, k \in Z)$.

4   Between two successive scales, we have a fundamental relation, called the *twin-scale relation*.

Twin-Scale Relation for ϕ	
$\phi_{1,0} = \sum\limits_{k \in Z} h_k \phi_{0,k}$	$\phi_{j+1,0} = \sum\limits_{k \in Z} h_k \phi_{j,k}$

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 3-34.

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 $\left\langle \phi_{j+1,n}, \phi_{j,k} \right\rangle = h_{k-2n}$.

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	
$\psi_{1,0} = \sum\limits_{k \in Z} g_k \phi_{0,k}$	$\psi_{j+1,0} = \sum\limits_{k \in Z} g_k \phi_{j,k}$

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

Let us define α_n, δ_n, α_k^0 by

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

Let us assess the α_k^0 coordinates as

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

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)

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

If we transform the (α_n) sequence into a new sequence $(\tilde{\alpha}_n)$ defined by

$\ldots, \alpha_{-1}, 0, \alpha_0, 0, \alpha_1, 0, \alpha_2, 0, \ldots$ 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
	$[cA_j, cD_j, ..., cD_1]$	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	$[cA_j, cD_j, ..., cD_1]$	wavedec

Synthesis-Reconstruction Capabilities

Purpose	Input	Output	File
Single-level reconstruction	cA_1, cD_1	s or A_0	idwt
Single-level reconstruction	cA_{j+1}, cD_{j+1}	cA_j	idwt
Full reconstruction	$[cA_j, cD_j, ..., cD_1]$	s or A_0	waverec
Selective reconstruction	$[cA_j, cD_j, ..., cD_1]$	A_l, D_m	wrcoef

Decomposition Structure Utilities

Purpose	Input	Output	File
Extraction of detail coefficients	$[cA_j, cD_j, ..., cD_1]$	$cD_k, 1 \leq k \leq j$	detcoef
Extraction of approximation coefficients	$[cA_j, cD_j, ..., cD_1]$	$cA_k, 0 \leq k \leq j$	appcoef
Recomposition of the decomposition structure	$[cA_j, cD_j, ..., cD_1]$	$[cA_k, cD_k, ..., cD_1] \; 1 \leq k \leq j$	upwlev

To illustrate command-line mode for 1-D capabilities, see "1-D Analysis Using the Command Line" on page 3-10. .

2-D Wavelet Capabilities

Basic 2-D Objects

	Objects	Description
Image in original resolution	s	Original image
	A_0	Approximation at level 0
	A_k, $1 \leq k \leq j$	Approximation at level k
	D_k, $1 \leq k \leq j$	Details at level k
Coefficients in scale-related resolution	cA_k, $1 \leq k \leq j$	Approximation coefficients at level k
	cD_k, $1 \leq k \leq j$	Detail coefficients at level k
	$[cA_j, cD_j, ..., cD_1]$	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 3-43.

To illustrate command-line mode for 2-D capabilities, see "Wavelet Image Analysis and Compression" on page 3-150..

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" [1]. 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 [2].

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.

Practical Considerations

The DWT iterative step consists of filtering followed by downsampling:

- Apply lowpass filter then downsample by two to obtain approximation coefficients.
- Apply highpass filter then downsample by two to obtain detail coefficients.

So conceptually, the number of approximation coefficients is one half the number of samples, and similarly for the detail coefficients.

In the real world, we deal with signals of finite length. With the desire of applying the theoretical DWT algorithm to the practical, the question of boundary conditions must be addressed: how should the signal be extended?

Before investigating the different scenarios, save the current boundary extension mode.

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

Periodic, Power of 2

Consider the following example. Load the `noisdopp` data. The signal has 1024 samples, which is a power of 2. Use `dwtmode` to set the extension mode to periodic. Then use `wavedec` to obtain the level-3 DWT of the signal using the orthogonal `db4` wavelet.

```
load noisdopp;
x = noisdopp;
lev = 3;
wav = 'db4';
dwtmode('per','nodisp')
[c,bk] = wavedec(x,lev,wav);
bk


bk =

        128         128         256         512        1024
```

The bookkeeping vector `bk` contains the number of coefficients by level. At each stage, the number of detail coefficients reduces exactly by a factor of 2. At the end, there are $1024/2^3 = 128$ approximation coefficients.

Compare the l^2-norms.

```
fprintf('l2-norm difference: %.5g\n',sum(x.^2)-sum(c.^2))

l2-norm difference: 9.0658e-09
```

Obtain the reconstructed approximations and details by setting to 0 the appropriate segments of the coefficients vector `c` and taking the inverse DWT.

```
cx = c;
cx(bk(1)+1:end) = 0;
reconApp = waverec(cx,bk,wav);
cx = c;
cx(1:bk(1)) = 0;
reconDet = waverec(cx,bk,wav);
```

Check for orthogonality.

```
fprintf('Orthogonality difference %.4g\n',...
    sum(x.^2)-(sum(reconApp.^2)+sum(reconDet.^2)))

Orthogonality difference 1.816e-08
```

Check for perfect reconstruction.

```
fprintf('Perfect reconstruction difference: %.5g\n',...
    max(abs(x-(reconApp+reconDet))));
```

```
Perfect reconstruction difference: 1.6741e-11
```

The three theoretical DWT properties are preserved.

Periodic, Not Power of 2

Now obtain the three-level DWT of a signal with 1026 samples. Use the same wavelet and extension mode is above. The number of coefficients at stage n does not evenly divide the signal length.

```
x = [0 0 noisdopp];
[c,bk] = wavedec(x,lev,wav);
bk
```

```
bk =

        129         129         257         513        1026
```

Check for l^2-norm preservation, orthogonality, and perfect reconstruction.

```
cx = c;
cx(bk(1)+1:end) = 0;
reconApp = waverec(cx,bk,wav);
cx = c;
cx(1:bk(1)) = 0;
reconDet = waverec(cx,bk,wav);

fprintf('l2-norm difference: %.5g\n',sum(x.^2)-sum(c.^2))
fprintf('Orthogonality difference %.4g\n',...
    sum(x.^2)-(sum(reconApp.^2)+sum(reconDet.^2)))
fprintf('Perfect reconstruction difference: %.5g\n',...
    max(abs(x-(reconApp+reconDet))));
```

```
l2-norm difference: -1.4028
Orthogonality difference -0.3319
Perfect reconstruction difference: 1.6859e-11
```

Perfect reconstruction is satisfied, but the l^2-norm and orthogonality are not preserved.

Not Periodic, Power of 2

Obtain the three-level DWT of a signal with 1024. Use the same wavelet as above, but this time change the extension mode to smooth extension of order 1. The number of coefficients at stage n does not evenly divide the signal length.

```
dwtmode('sp1','nodisp')
[c,bk] = wavedec(x,lev,wav);
bk
```

```
bk =
```

134	134	261	516	1026

Check for l^2-norm preservation, orthogonality, and perfect reconstruction.

```
cx = c;
cx(bk(1)+1:end) = 0;
reconApp = waverec(cx,bk,wav);
cx = c;
cx(1:bk(1)) = 0;
reconDet = waverec(cx,bk,wav);

fprintf('l2-norm difference: %.5g\n',sum(x.^2)-sum(c.^2))
fprintf('Orthogonality difference %.4g\n',...
    sum(x.^2)-(sum(reconApp.^2)+sum(reconDet.^2)))
fprintf('Perfect reconstruction difference: %.5g\n',...
    max(abs(x-(reconApp+reconDet))));

l2-norm difference: -113.58
Orthogonality difference -2.678
Perfect reconstruction difference: 1.637e-11
```

Again, only perfect reconstruction is satisfied.

Restore the original extension mode.

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

To support perfect reconstruction for arbitrary choices of signal length, wavelet, and extension mode, we use *frames* of wavelets.

A frame is a set of functions $\{\phi_k\}$ that satisfy the following condition: there exist constants $0 < A \le B$ such that for any function f, the *frame inequality* holds: $A\|f\|^2 \le \Sigma_k |\langle f, \phi_k \rangle|^2 \le B\|f\|^2$.

The functions in a frame are generally not linearly independent. This means that the function f does not have a unique expansion in ϕ_k. Daubechies [3] shows that if $\{\tilde{\phi}_k\}$ is the dual frame and $f = \Sigma_{k \in K} c_k \phi_k$ for some $c = (c_k) \in l^2(K)$, and if not all c_k equal $\langle f, \tilde{\phi}_k \rangle$, then $\Sigma_{k \in K} |c_k|^2 \ge \Sigma_{k \in K} |\langle f, \tilde{\phi}_k \rangle|^2$.

If $A = B$, the frame is called a *tight frame*. If $A = B = 1$ and $\|\phi_k\|^2 = 1$ for all ϕ_k, the frame is an orthonormal basis. If $A \ne B$, then energy is not necessarily preserved, and the total number of coefficients might exceed the length of the signal. If the extension mode is periodic, the wavelet is orthogonal, and the signal length is divisible by 2^J, where J is the maximum level of the wavelet decomposition, all three theoretical DWT properties are satisfied.

Arbitrary Extensions

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

Create a signal and obtain the filters associated with the `db9` wavelet.

```
x = sin(0.3*[1:451]);
w = 'db9';
[LoD,HiD,LoR,HiR] = wfilters(w);
```

Append and prepend `length(LoD)` random numbers to the signal. Plot the original and extended signals.

```
lx = length(x);
lf = length(LoD);
ex = [randn(1,lf) x randn(1,lf)];
ymin = min(ex);
ymax = max(ex);
subplot(2,1,1)
plot(lf+1:lf+lx,x)
axis([1 lx+2*lf ymin ymax]);
title('Original Signal')
subplot(2,1,2)
plot(ex)
title('Extended Signal')
axis([1 lx+2*lf ymin ymax])
```

Original Signal
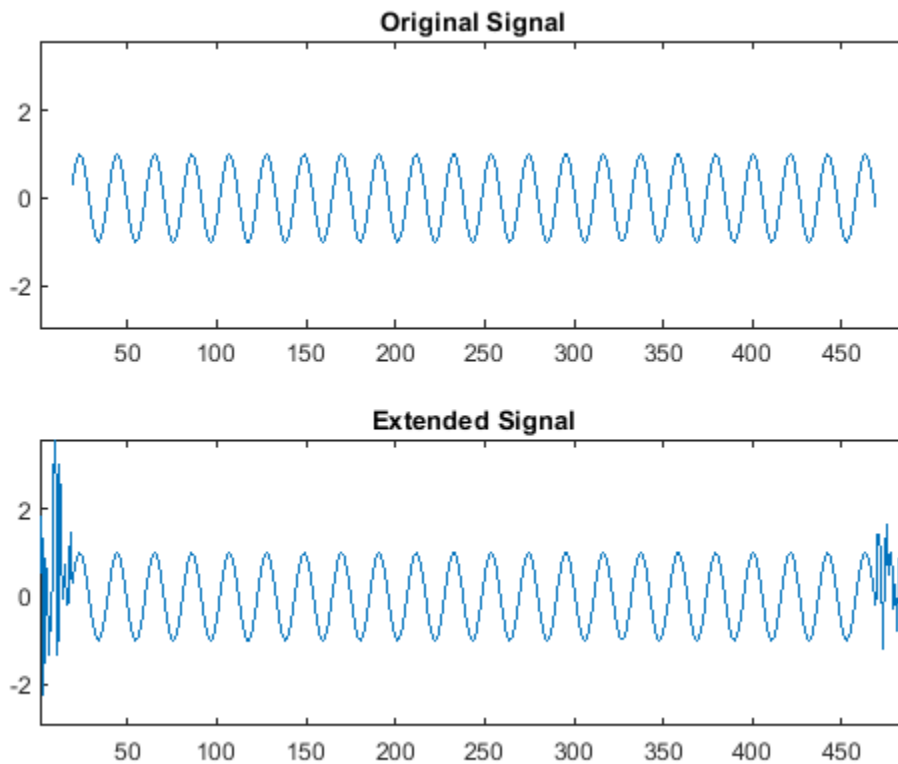
Extended Signal

Use the lowpass and highpass wavelet decomposition filters to obtain the one-level wavelet decomposition of the extended signal.

```
la = floor((lx+lf-1)/2);
ar = wkeep(dyaddown(conv(ex,LoD)),la);
dr = wkeep(dyaddown(conv(ex,HiD)),la);
```

Confirm perfect reconstruction of the signal.

```
xr = idwt(ar,dr,w,lx);
err0 = max(abs(x-xr))
```

```
err0 = 5.4700e-11
```

Comparing Extension Differences

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.
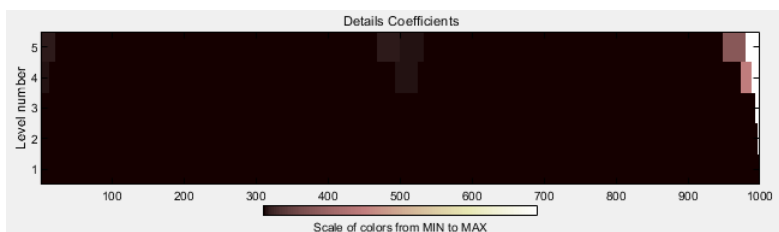
Symmetric Extension

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

The detail coefficients clearly show the signal end effects.
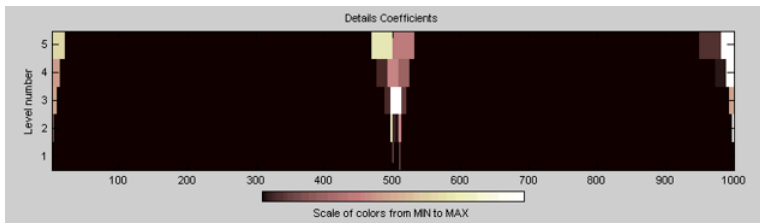
Smooth Padding

10 From the MATLAB prompt, type

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

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

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

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

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

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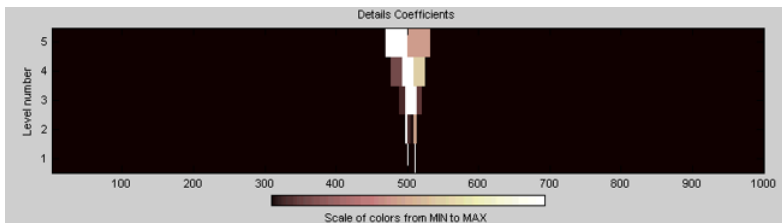
Image Extensions

Let us now consider an image example. Save the current extension mode. Load and display the geometry image.

```
origmodestatus = dwtmode('status','nodisplay');
load geometry
nbcol = size(map,1);
colormap(pink(nbcol))
image(wcodemat(X,nbcol))
```
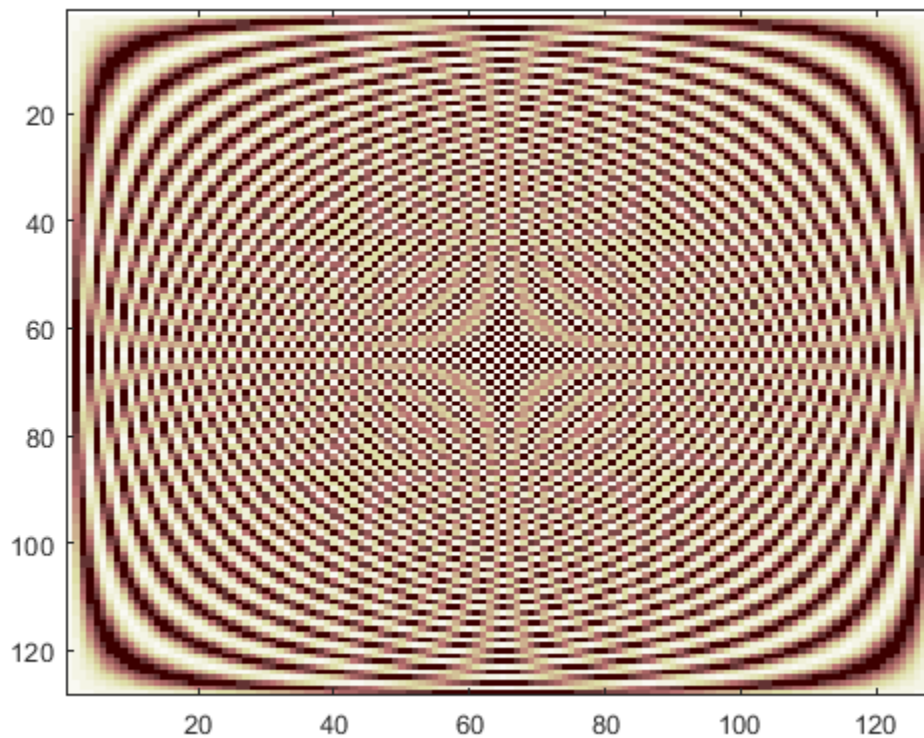
Zero-Padding

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

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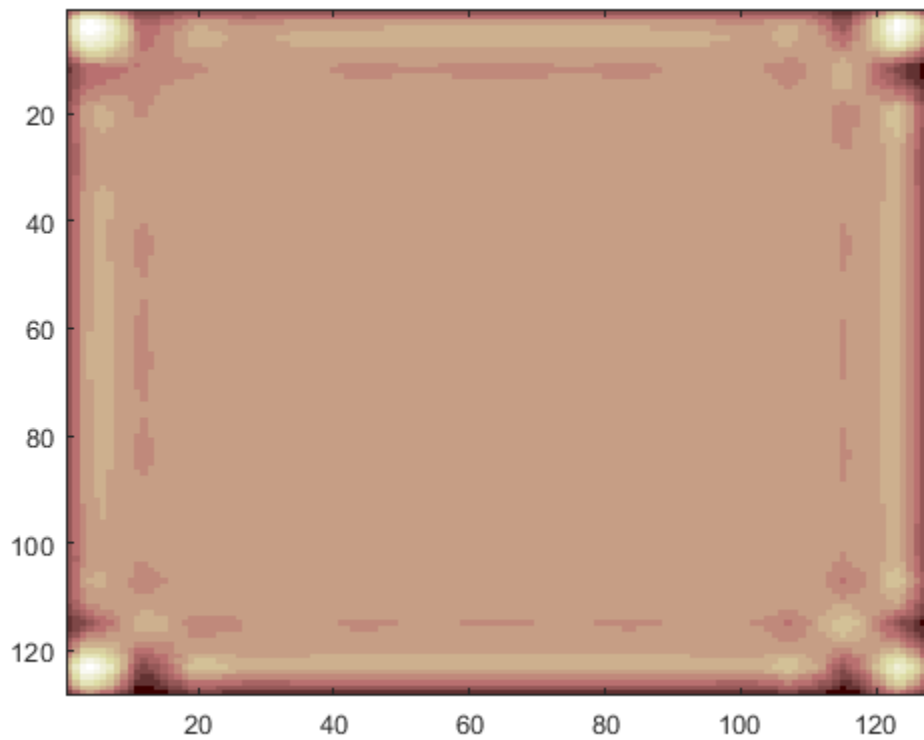
Symmetric Extension

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

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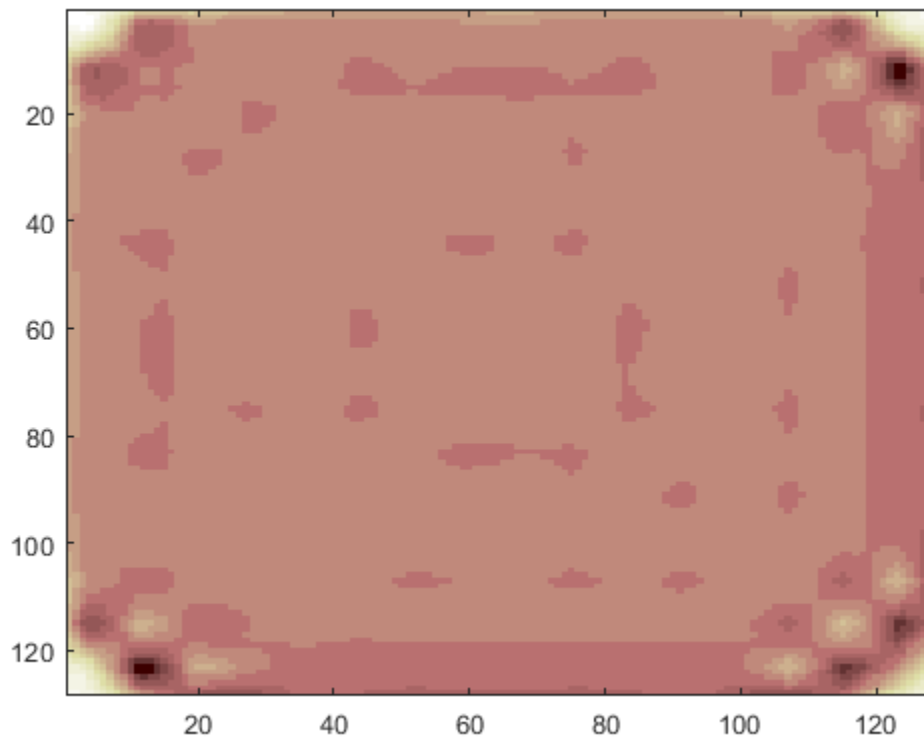
Smooth Padding

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

```
dwtmode('spd','nodisp')
[c,s] = wavedec2(X,lev,wname);
a = wrcoef2('a',c,s,wname,lev);
image(wcodemat(a,nbcol))
```
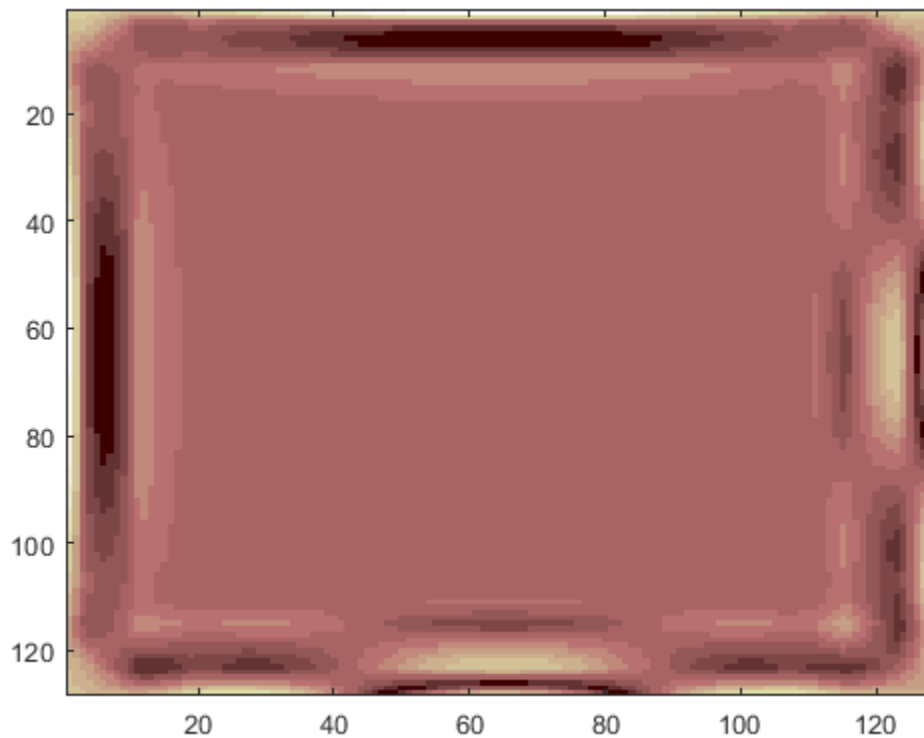
Restore the original extension mode.

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

References

[1] Cohen, A., I. Daubechies, B. Jawerth, and P. Vial. "Multiresolution analysis, wavelets and fast algorithms on an interval." *Comptes Rendus Acad. Sci. Paris Sér. A*, Vol. 316, pp. 417–421, 1993.

[2] Strang, G., and T. Nguyen. *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1996.

[3] Daubechies, I. *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: SIAM Ed, 1992.

See Also
```
dwtfilterbank | wavedec
```

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 3-62 and "2-D Stationary Wavelet Transform" on page 3-169.

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: $\varepsilon = \varepsilon_1...,\varepsilon_J$. This transform is called the ε-decimated DWT.

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

How to Calculate the ε-Decimated DWT: SWT

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

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

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

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

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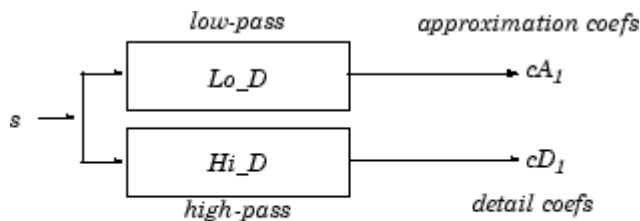
The general step j convolves the approximation coefficients at level j–1, with upsampled versions of the appropriate original filters, to produce the approximation and detail coefficients at level j. This can be visualized in the following figure.

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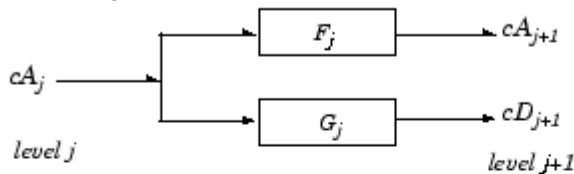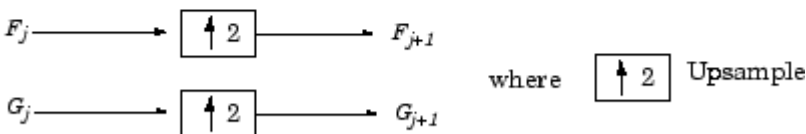
Next, we illustrate how to extract a given ε-decimated DWT from the approximation and detail coefficients structure of the SWT.

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

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

Step 0 (Original Data)

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

Step 1

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

Step 2

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

Step 3

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

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

```
[tmpAPP,tmpDET] =
dwt(A(j,ε₁, ,εⱼ),wname,'mode','per','shift',εⱼ₊₁);
A(j+1,ε₁, ,εⱼ,εⱼ₊₁) = 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 $\varepsilon = [\varepsilon_1,...,\varepsilon_j]$ with $\varepsilon_i = 0$ or 1. We have $2^J = 2^3 =$ eight different ε-decimated DWTs at level 3. Choosing ε, we can retrieve the corresponding ε-decimated DWT from the SWT array.

Now, consider the last step, $J = 3$, and let $[C\varepsilon,L\varepsilon]$ 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:

Cε =

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

Lε = [1,1,2,4,8]

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

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

Two-Dimensional SWT

Decomposition Step
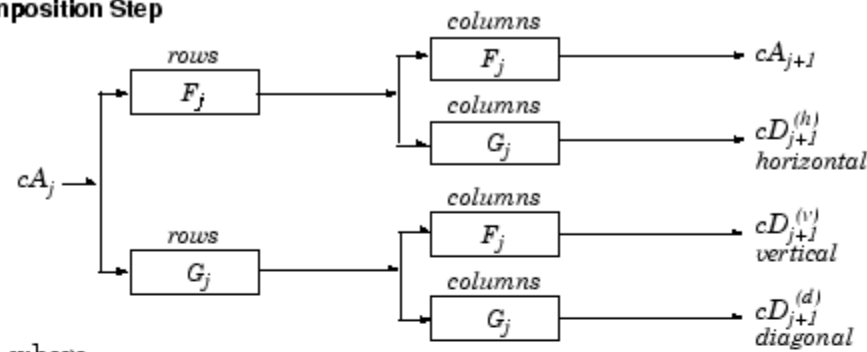
where

Filter Computation
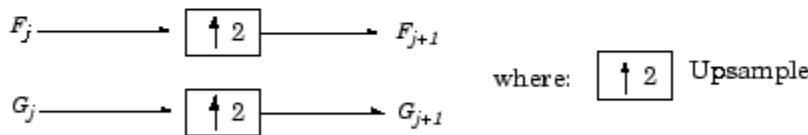
Initialization

$cA_0 = s$ for the decomposition initialization

$F_0 = Lo\_D$

$G_0 = Hi\_D$

Note $size(cA_j) = size(cD_j^{(h)}) = size(cD_j^{(v)}) = size(cD_j^{(d)}) = s$
Where $s = $ size of the analyzed image

Inverse Discrete Stationary Wavelet Transform (ISWT)

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

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

```
FOR j = J:-1:1
   A(j-1, ε₁, ,εⱼ₋₁) = ...
   idwt(A(j,ε₁, ,εⱼ),D(S,ε₁, ,εⱼ)],wname,'mode','per','shift',εⱼ);
END
```

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

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

The ISWT is obtained with the following abstract algorithm:

```
FOR j = J:-1:1
    X0 = idwt(A(j,ε₁, ,εⱼ),D(j,ε₁, ,εⱼ)],wname, ...
            'mode','per','shift',0);
    X1 = idwt(A(j,ε₁, ,εⱼ),D(j,ε₁, ,εⱼ)],wname, ...
            'mode','per','shift',1);
    X1 = circshift(X1,-1);
    A(j-1, ε₁, ,εⱼ₋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 3-57 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 `wextend` function to extend it.

3   Perform a single-level Stationary Wavelet Decomposition.

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

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

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

```
whos
```

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

4   Display the coefficients of approximation and detail.

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

```
subplot(1,2,1), plot(swa); title('Approximation cfs')
subplot(1,2,2), plot(swd); title('Detail cfs')
```
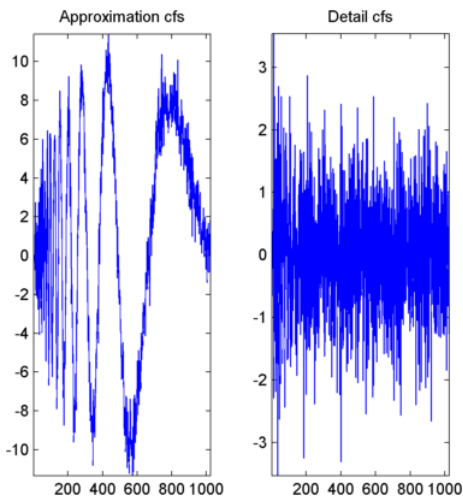
5   Regenerate the signal by Inverse Stationary Wavelet Transform.

To find the inverse transform, type

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

To check the perfect reconstruction, type

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

6   Construct and display approximation and detail from the coefficients.

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

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

To display the approximation and detail at level 1, type

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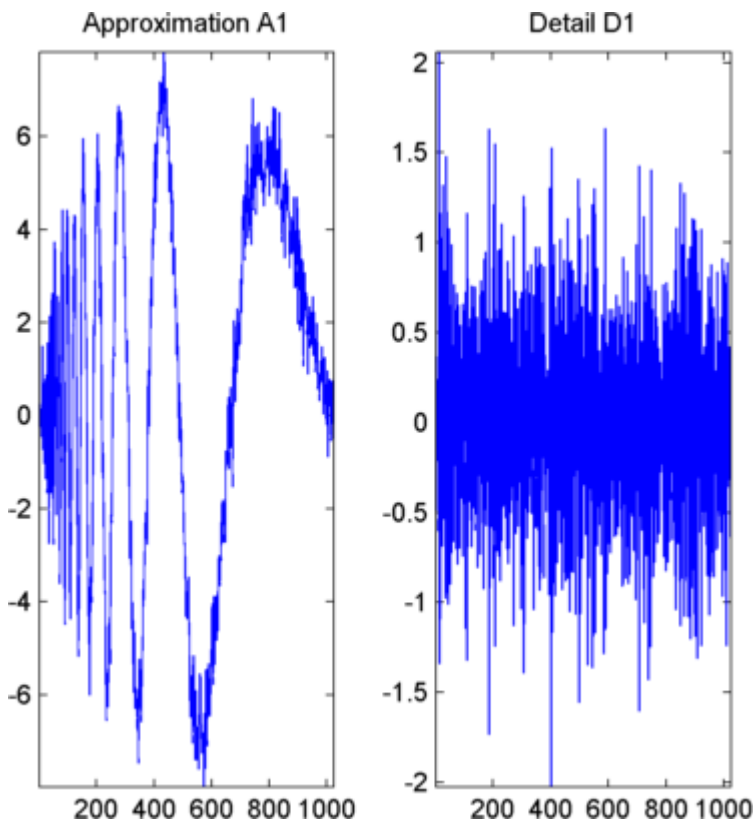

7   Perform a multilevel Stationary Wavelet Decomposition.

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

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

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

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

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

8 Display the coefficients of approximations and details.

To display the coefficients of approximations and details, type

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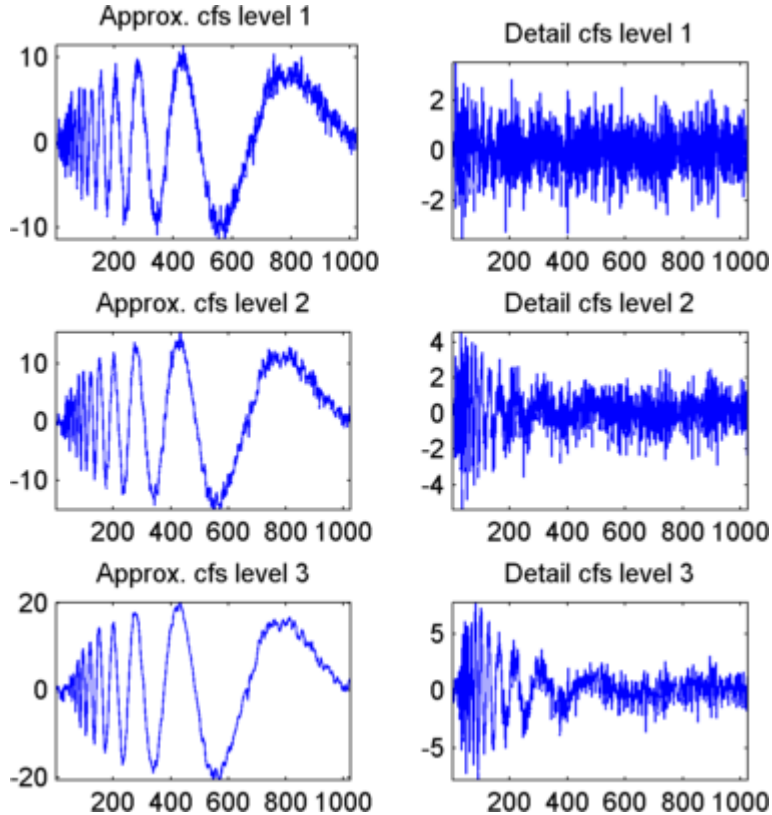

9 Reconstruct approximation at Level 3 From coefficients.

To reconstruct the approximation at level 3, type

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

10 Reconstruct details from coefficients.

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

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

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

To reconstruct the approximations at levels 2 and 3, type

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

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

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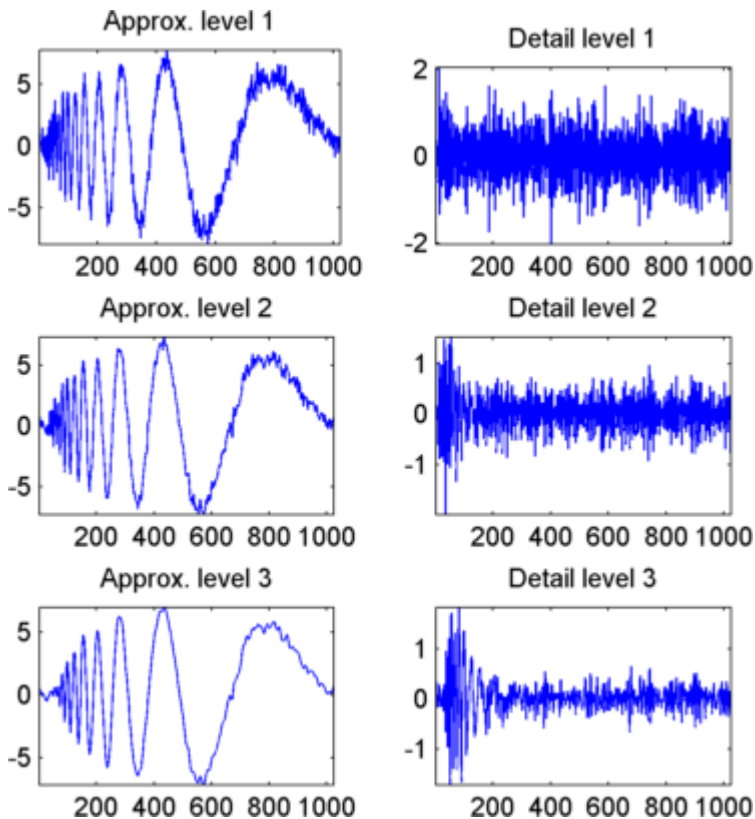
12 Remove noise by thresholding.

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

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

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

To display both the original and denoised signals, type

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

Original signal
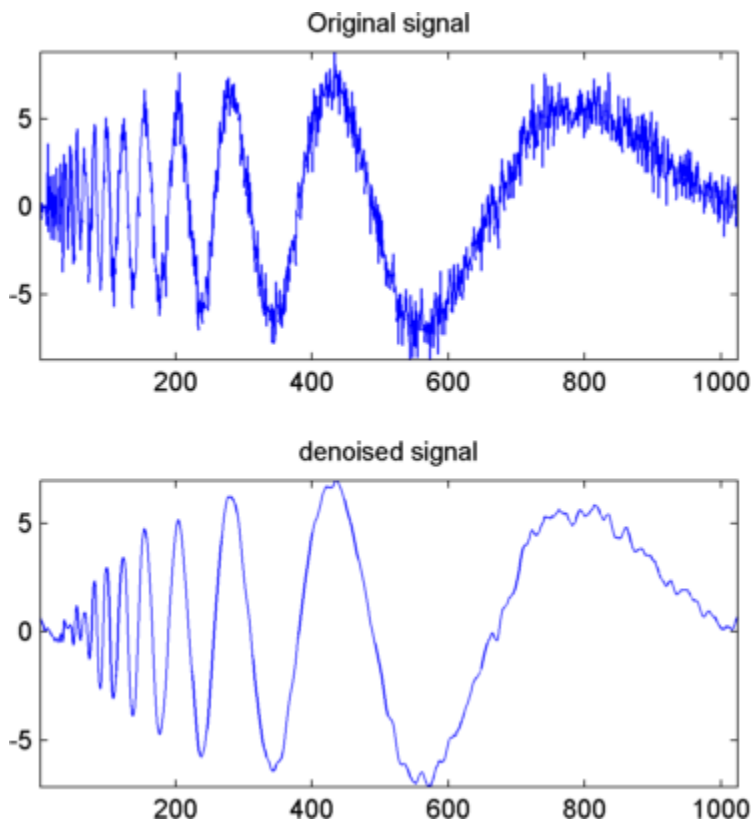
denoised signal

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

Original signal

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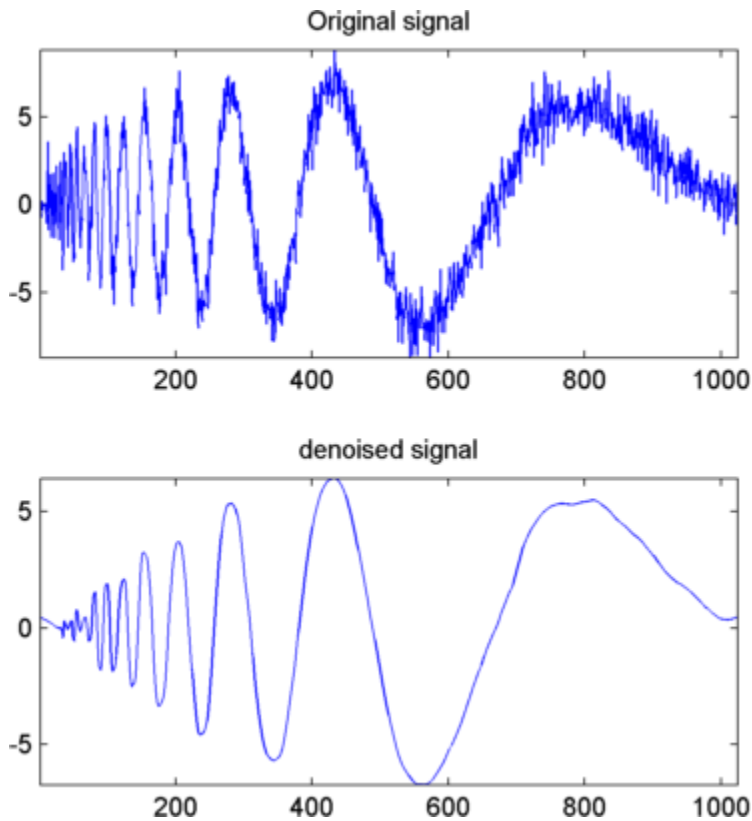
A second syntax can be used for the `swt` and `iswt` functions, giving the same results:

```
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`.
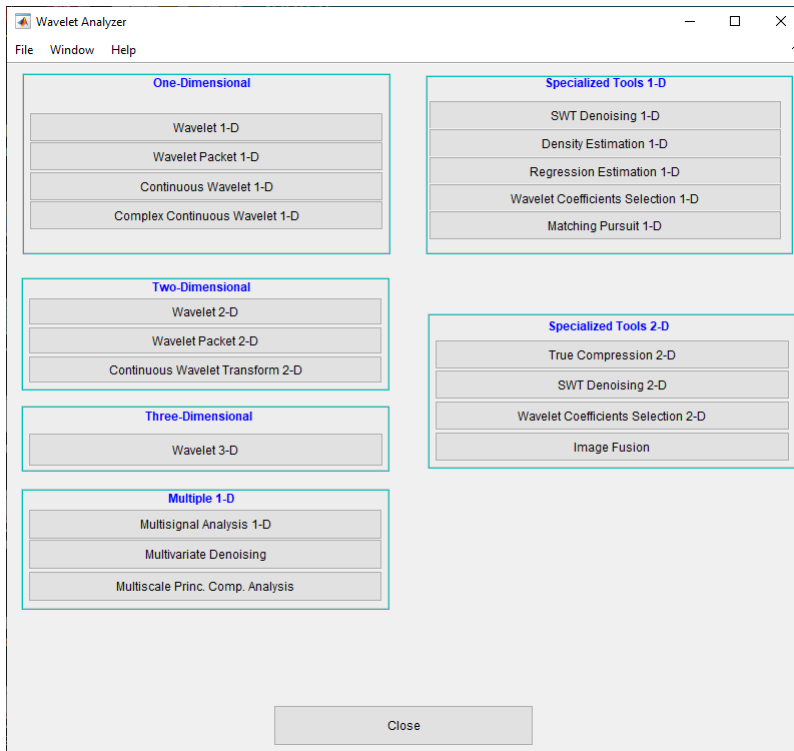
The **Wavelet Analyzer** appears.

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 3-34 in the *Wavelet Toolbox User's Guide*).
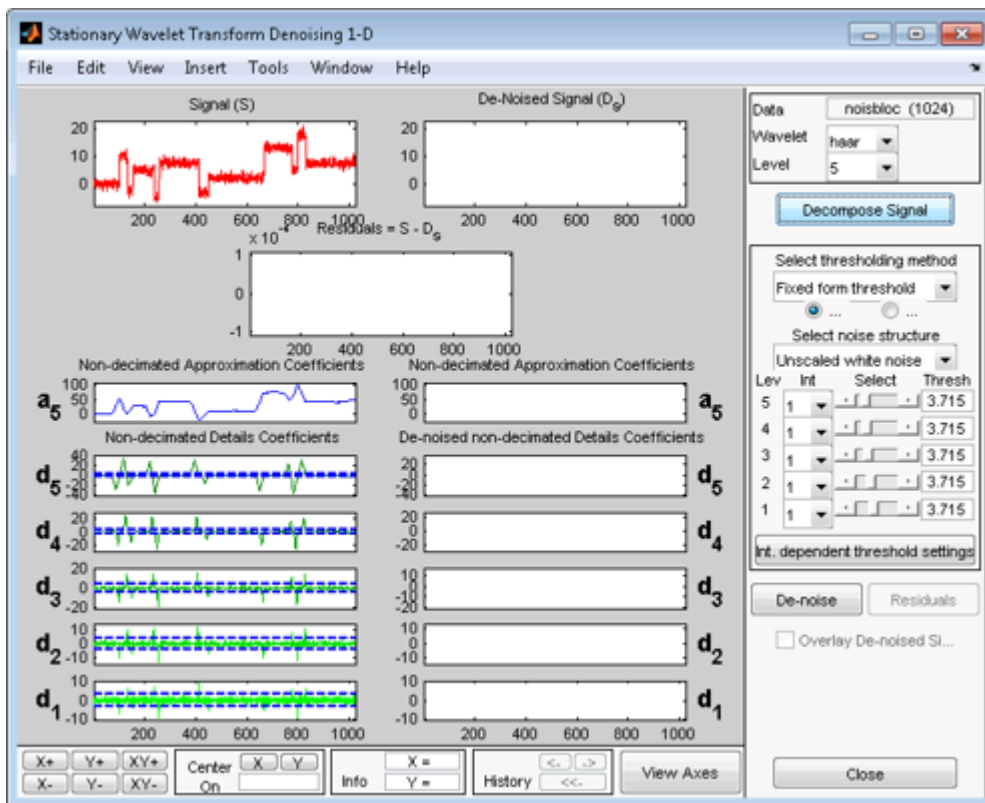
4   denoise the signal using the Stationary Wavelet Transform.

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.
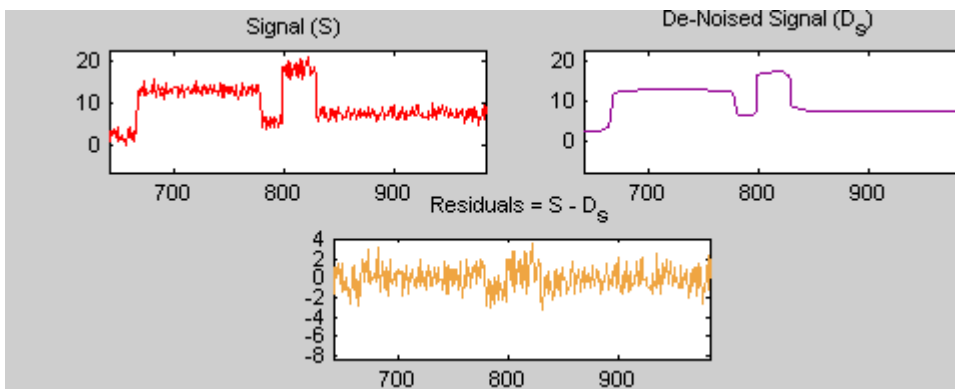
Selecting a Thresholding Method

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
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 6-39.

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')
```

Nile River Minimum Levels
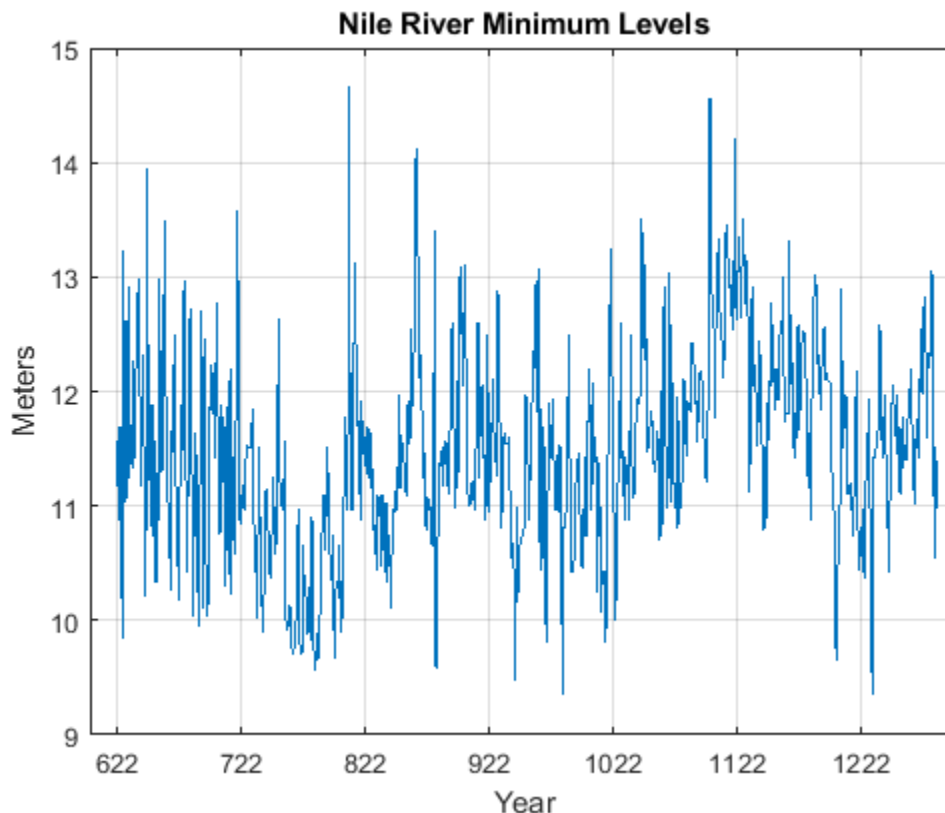
Construction began on a new measuring device around 715 AD. Examining the data prior to and after approximately 722 AD, there appears to be a change in the variability of the data. You can use wavelets to explore the hypothesis that the variability of the measurements has been affected by the introduction 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')
```
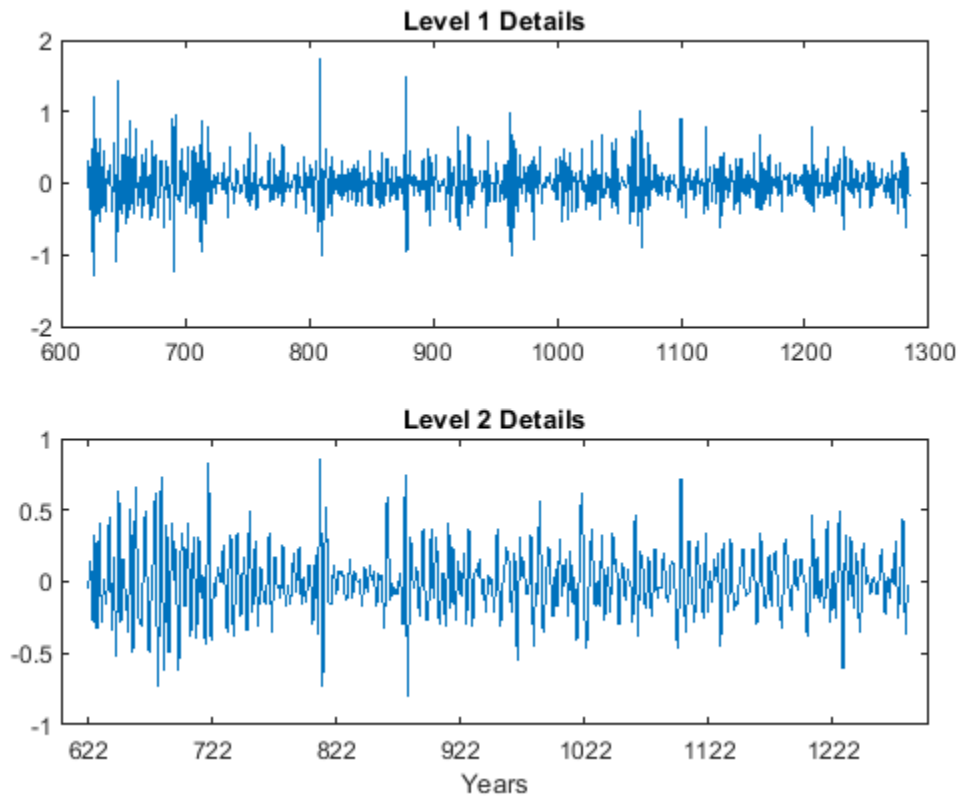

Apply an overall change of variance test to the wavelet coefficients.

```
for JJ = 1:5
    pts_Opt = wvarchg(wt(JJ,:),2);
    changepoints{JJ} = pts_Opt;
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')
```
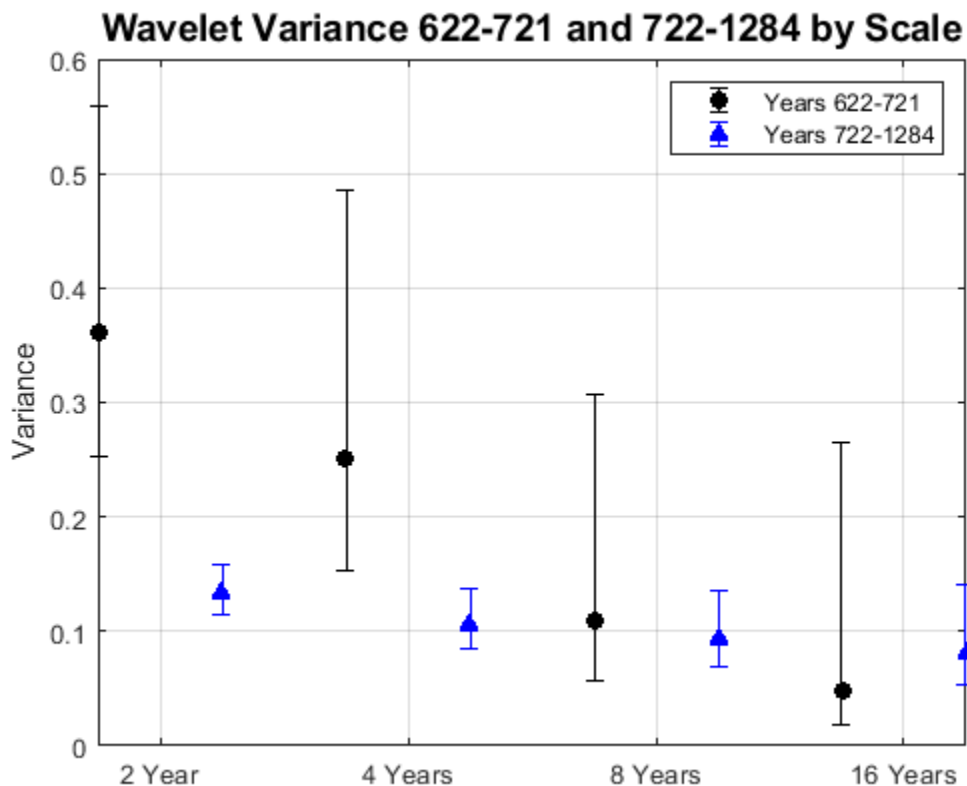

The wavelet variance indicates a significant change in variance between the 622-721 and 722-1284 data over scales of 2 and 4 years.

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.

```matlab
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.

```matlab
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
```

Wavelet Variance 622-721 and 722-1284 by Scale
Biased Estimate -- Reflection Boundary
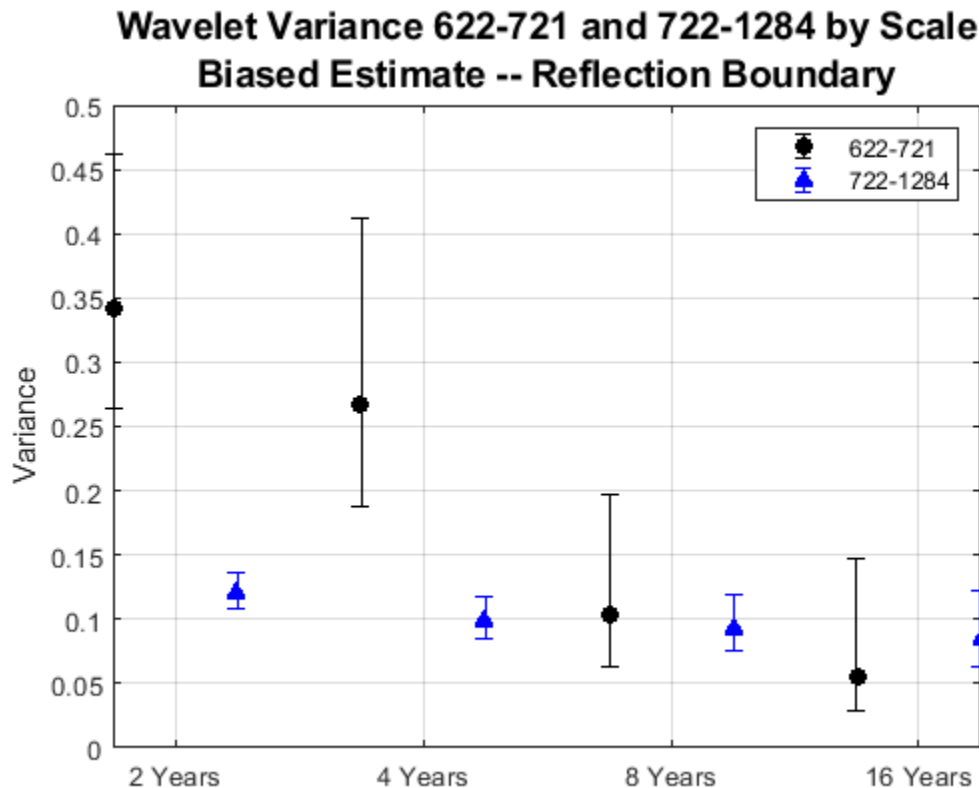
The conclusion is reinforced. There is a significant difference in the variance of the data over scales of 2 and 4 years, but not at longer scales. You can conclude that something has changed about the process variance.

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')
```

Year over Year Real U.S. GDP - D1
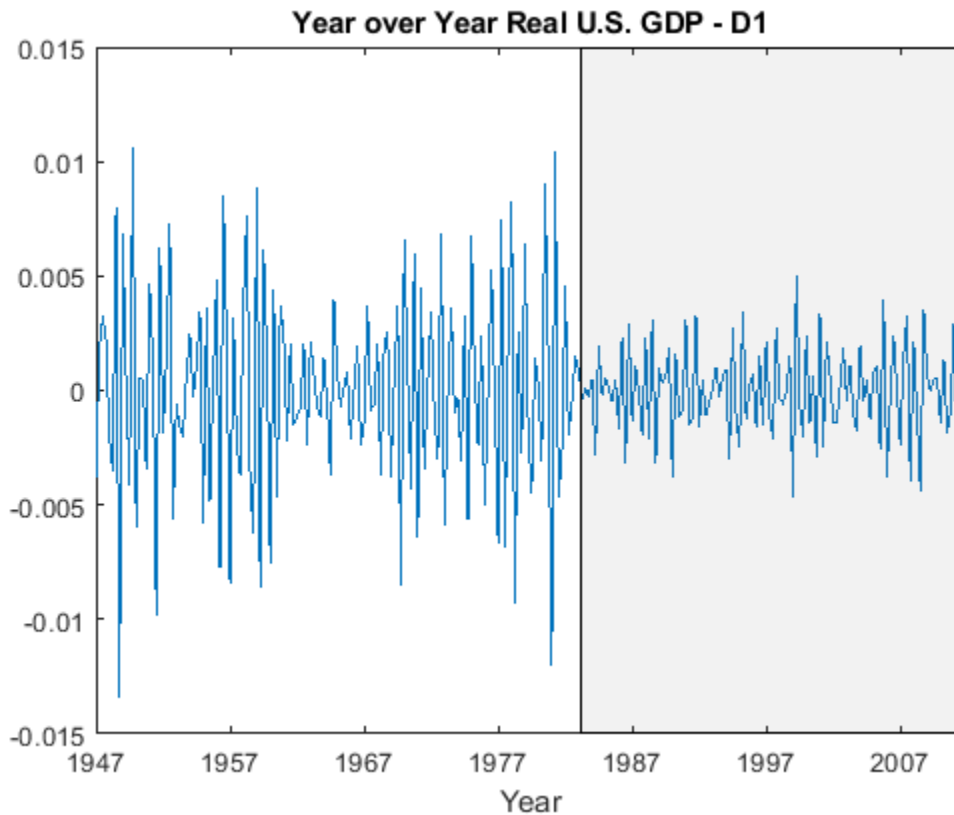
Examining the level-one details, it appears there is a reduction of variance beginning in the 1980s.

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_Opt);
tspost = realgdp(pts_Opt+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')
```
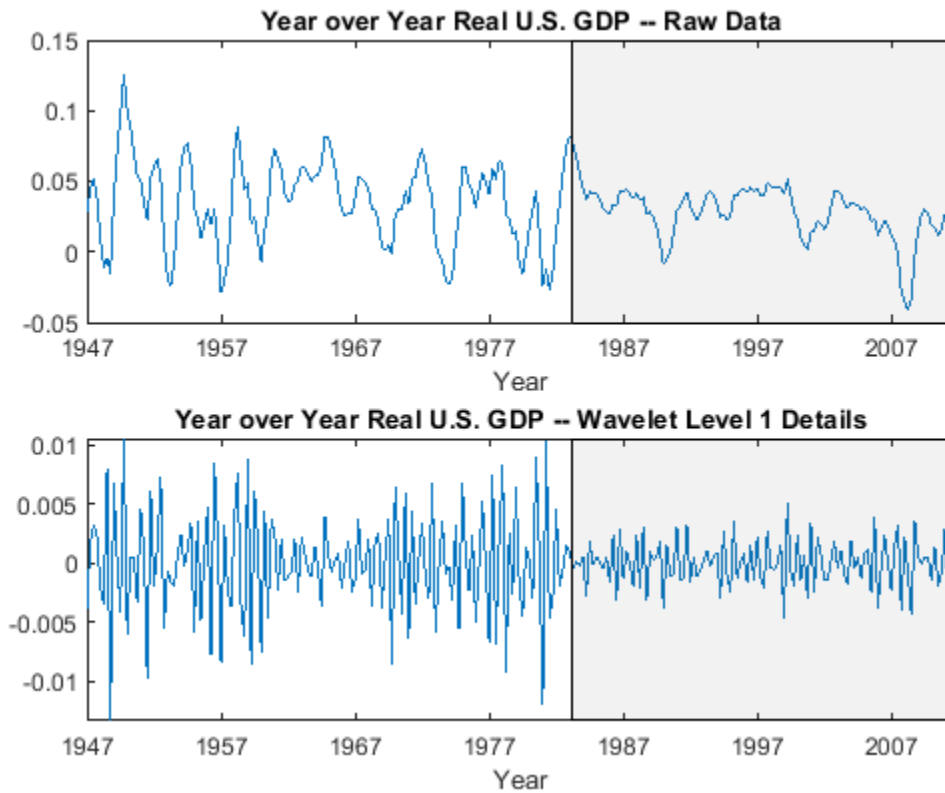
The shaded region is referred to as the "Great Moderation" signifying a period of decreased macroeconomic volatility in the U.S. beginning in the mid 1980s.

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.

```matlab
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.

```matlab
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')
```

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_Opt,kopt,t_est] = 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_Opt);
tspost = realgdp(pts_Opt+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 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] = modwtxcorr(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 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');
```

R-Waves Localized by Wavelet Transform

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:

$< x_n, y_{n-k} >_n \ = \sum_n x_n \bar{y}_{n-k}$ 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 $< x_n, x_{n-10-k} >_n \ = \sum_n x_n \bar{x}_{n-10-k}$. 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 $k = L$. 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.

```
n = 20;
x0 = 1:n/2;
x1 = (2*x0-1)/n;
x = [x1 fliplr(x1)]';
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')
```
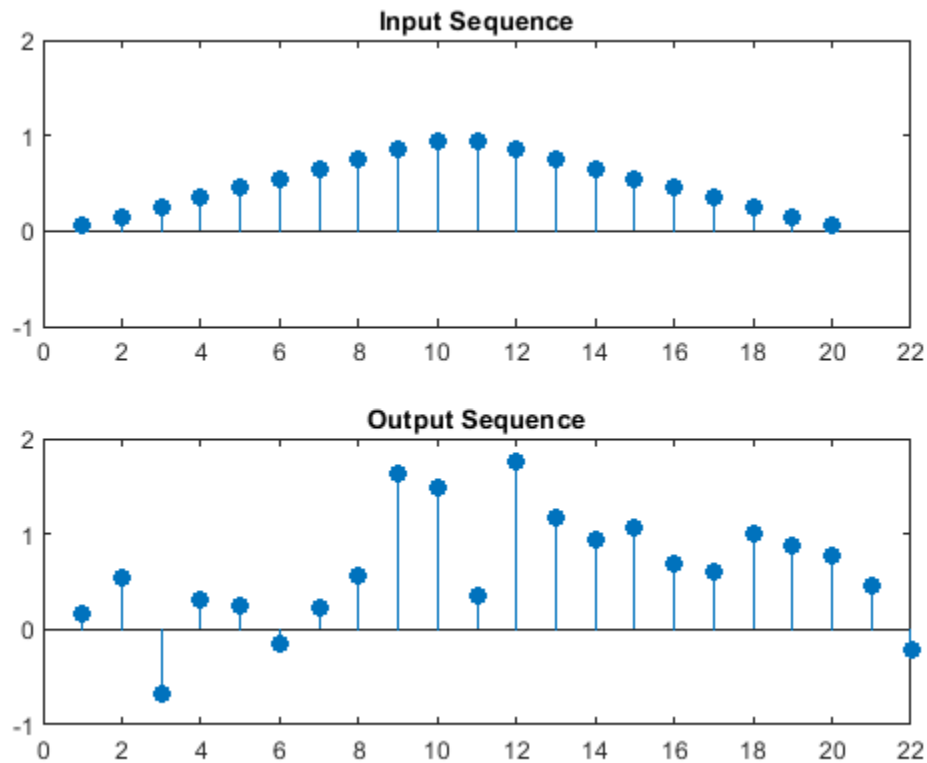
Use `xcorr` to obtain the cross-correlation sequence and determine the lag where the maximum is obtained.

```
[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')
```
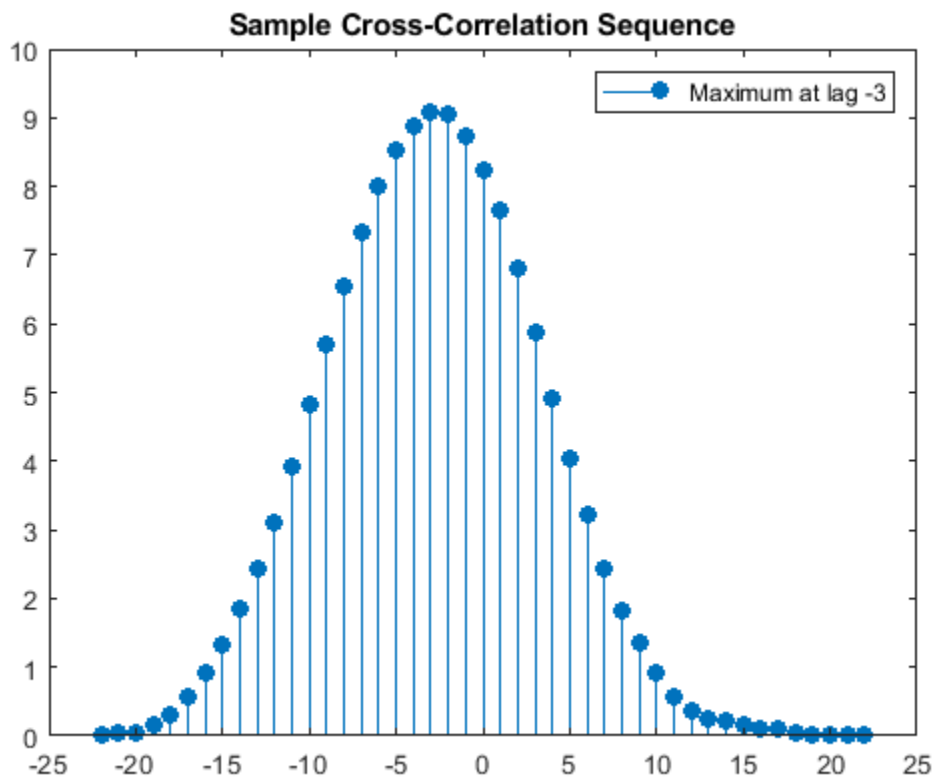
The maximum is found at a lag of -3. The signal y is the second input to xcorr and it is a delayed version of x. You have to shift y 3 samples to the left (a negative shift) to maximize the cross correlation. If you reverse the roles of x and y as inputs to xcorr, the maximum lag now occurs at a positive 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')
```
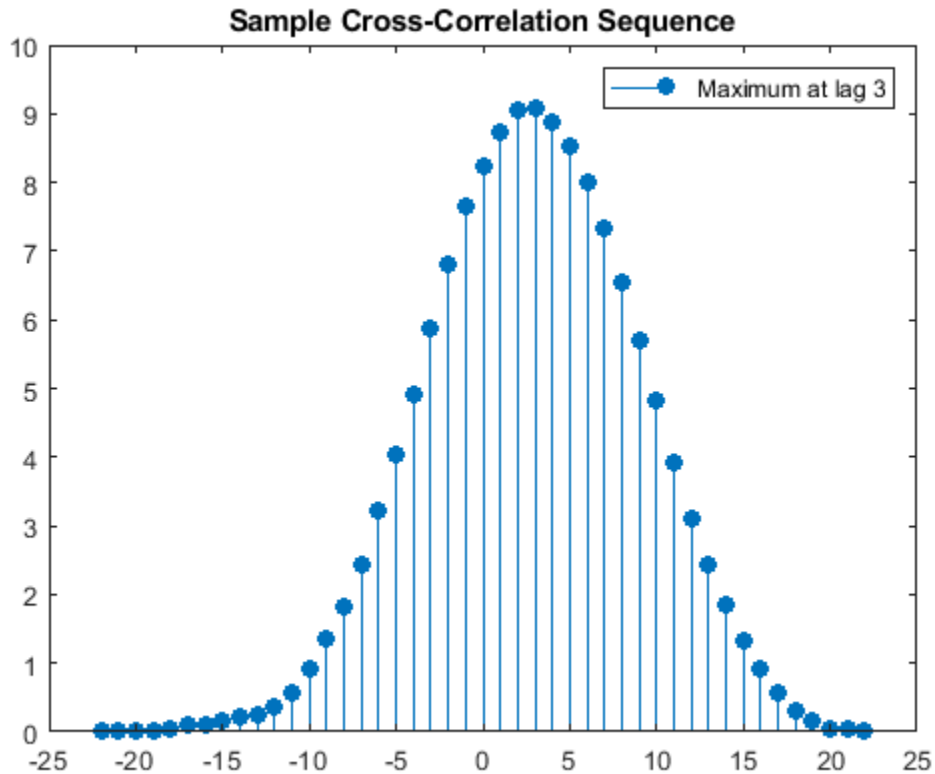
x is an advanced version of y and you delay x by three samples to maximize the cross correlation.

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')
```
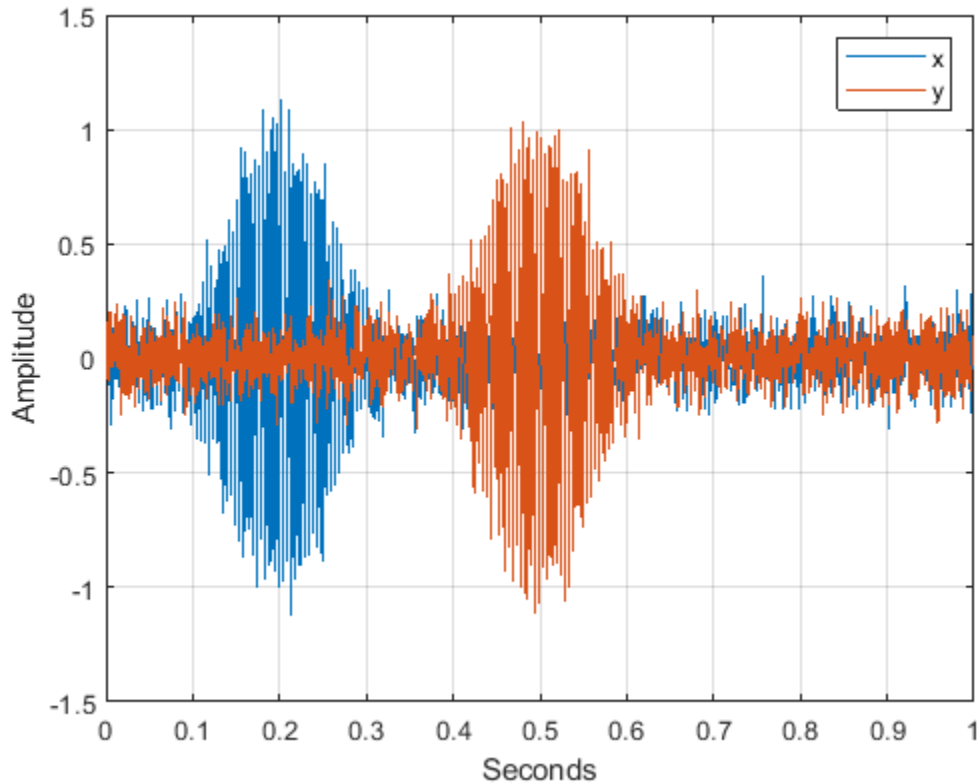
You see that x and y are very similar except that y is delayed by 0.3 seconds. Obtain the nondecimated wavelet transform of x and y down to level 5 using the Fejer-Korovkin (14) wavelet. The wavelet coefficients at level 3 with a sampling frequency of 2 kHz are an approximate $[2000/2^4, 2000/2^3)$ bandpass 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] = modwtxcorr(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')
```

Wavelet Cross-Correlation Sequence (level 3)
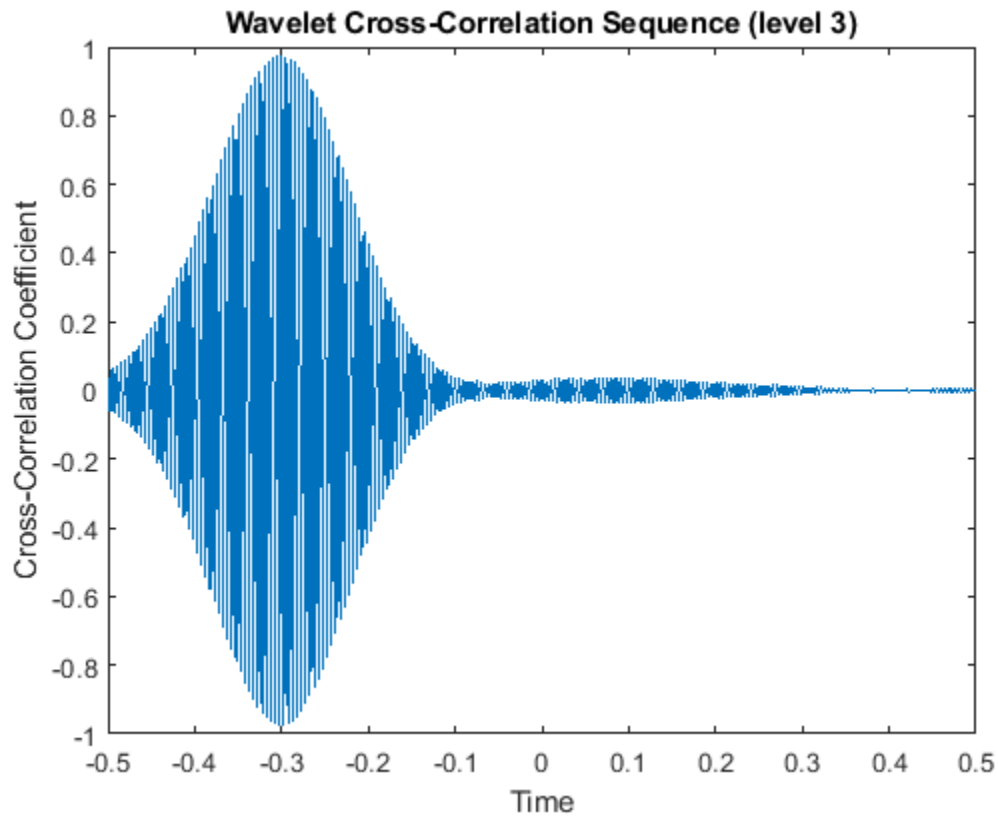
The cross-correlation sequence peaks at a delay of -0.3 seconds. The wavelet transform of y is the second input to `modwtxcorr`. Because the second input of `modwtxcorr` is shifted relative to the first, the peak correlation occurs at a negative delay. You have to left shift (advance) the cross-correlation sequence to align the time series. If you reverse the roles of the inputs to `modwtxcorr`, you obtain the peak correlation at a positive lag.

```
[xc,~,lags] = modwtxcorr(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')
```
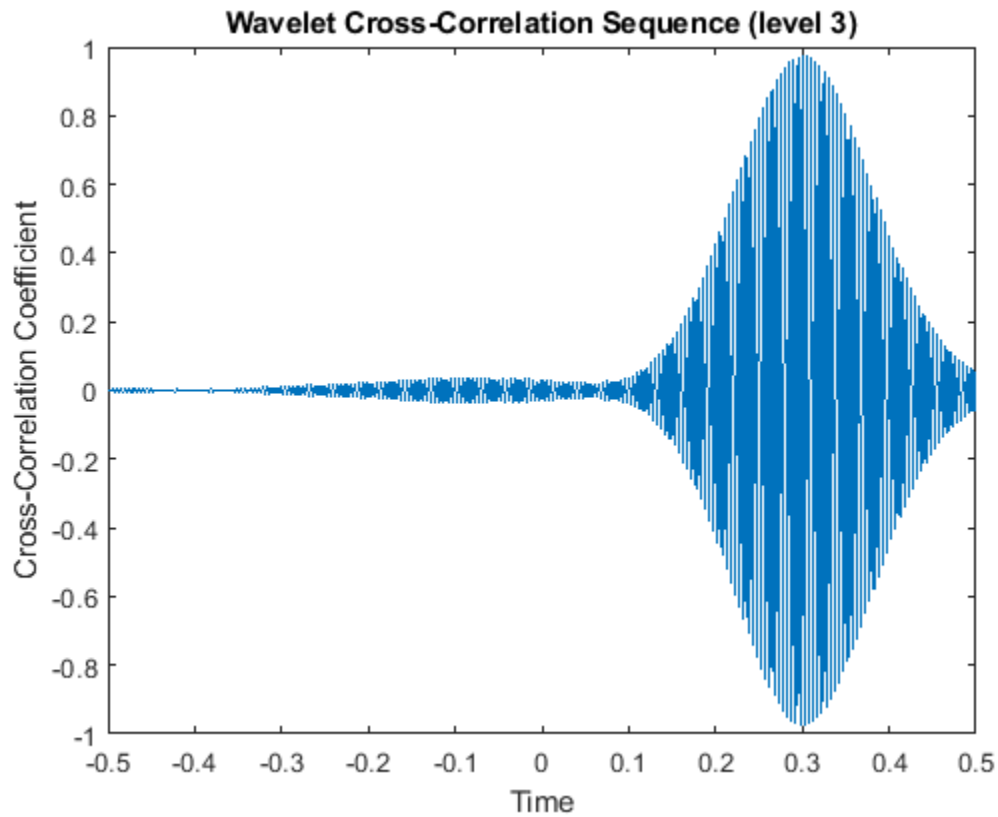
To show that wavelet cross-correlation enables scale(frequency)-localized correlation, plot the cross-correlation sequences at levels 1 and 5.

```
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')
```

Wavelet Cross-Correlation Sequence (level 1)
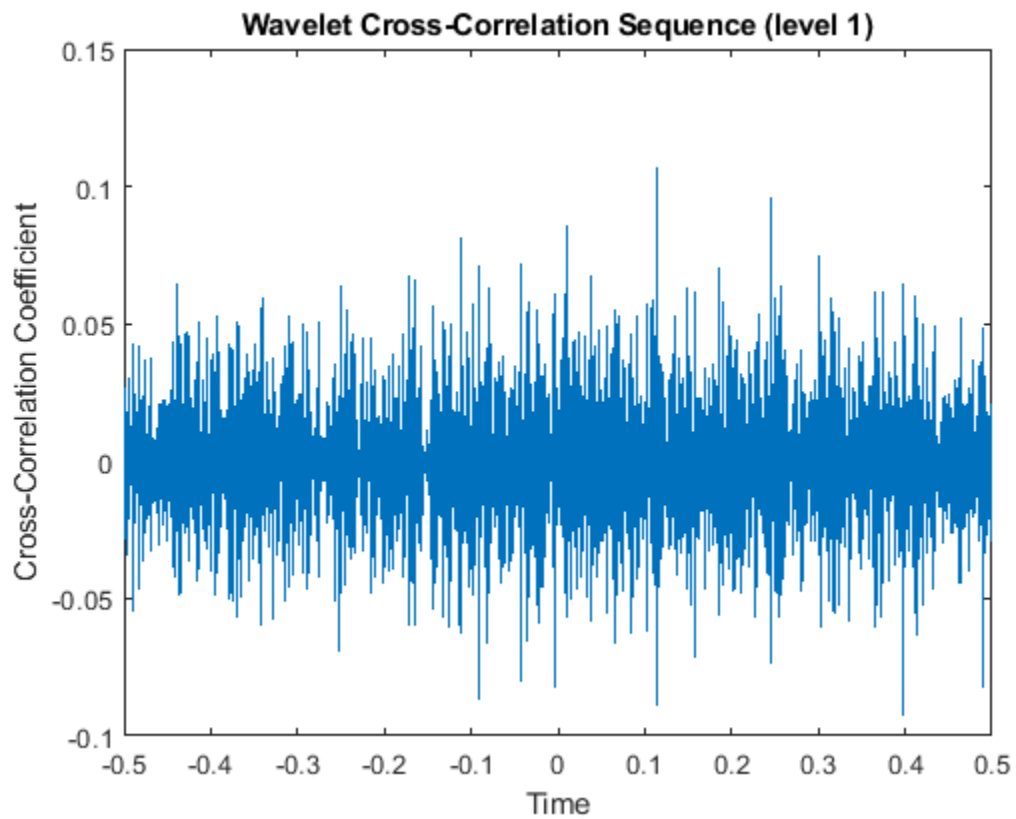

```
figure
lev = 5;
zerolag = floor(numel(xc{lev})/2+1);
tlag = lags{lev}(zerolag-999:zerolag+1000).*(1/2000);
plot(tlag,xc{lev}(zerolag-999:zerolag+1000))
title('Wavelet Cross-Correlation Sequence (level 5)')
xlabel('Time')
ylabel('Cross-Correlation Coefficient')
```
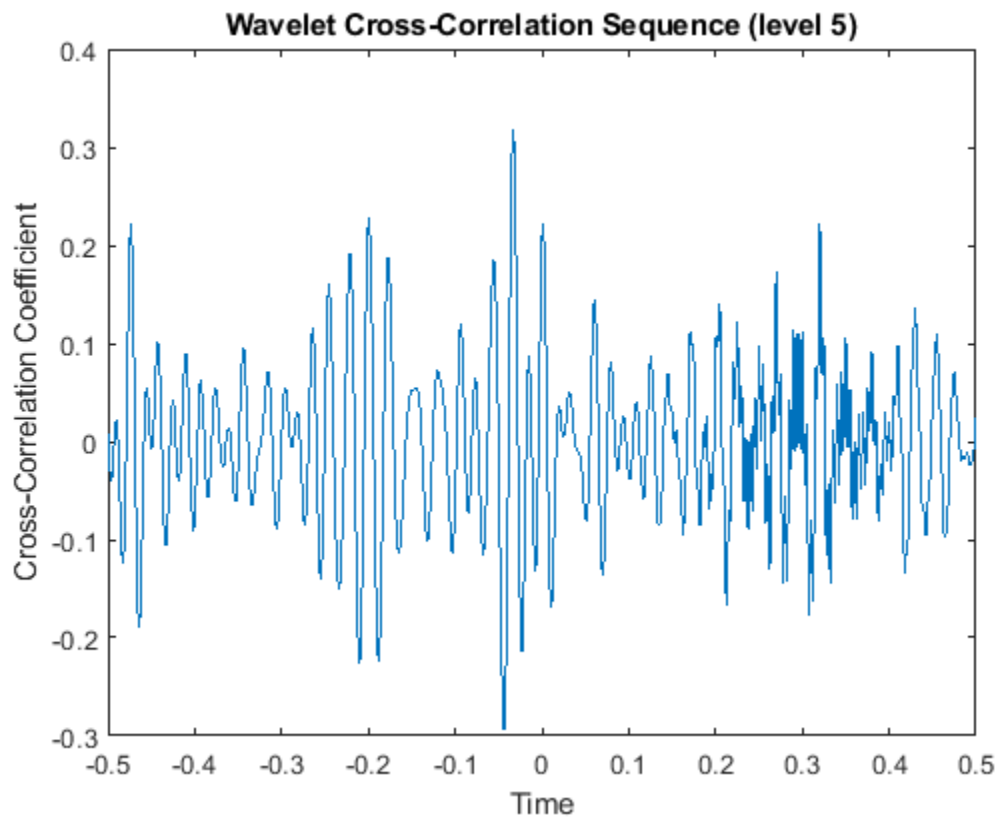
The wavelet cross-correlation sequences at levels 1 and 5 do not show any evidence of the exponentially-weighted sinusoids due to the bandpass nature of the wavelet transform.

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')
```
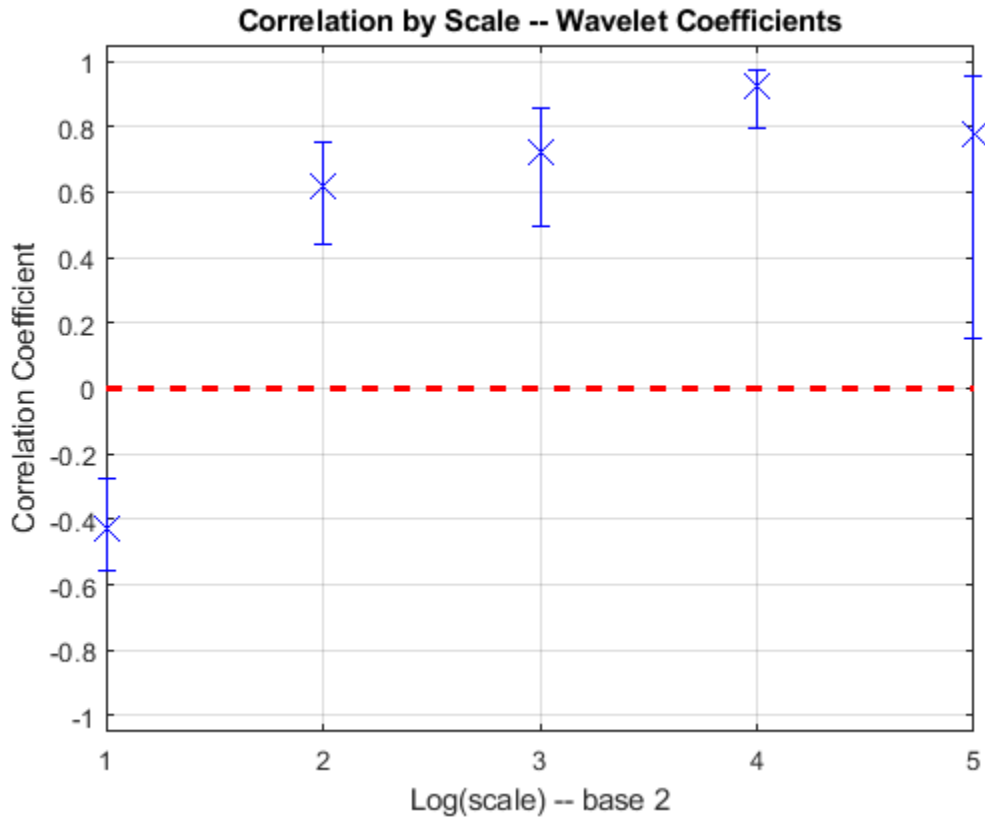
Personal expenditure and personal investment are negatively correlated over a period of 2-4 quarters. At longer scales, there is a strong positive correlation between personal expenditure and personal investment. Examine the wavelet cross-correlation sequence at the scale representing 2-4 quarter cycles. Plot the cross-correlation sequence along with 95% confidence intervals.

```
[xcseq,xcseqci,lags] = modwtxcorr(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'})
```
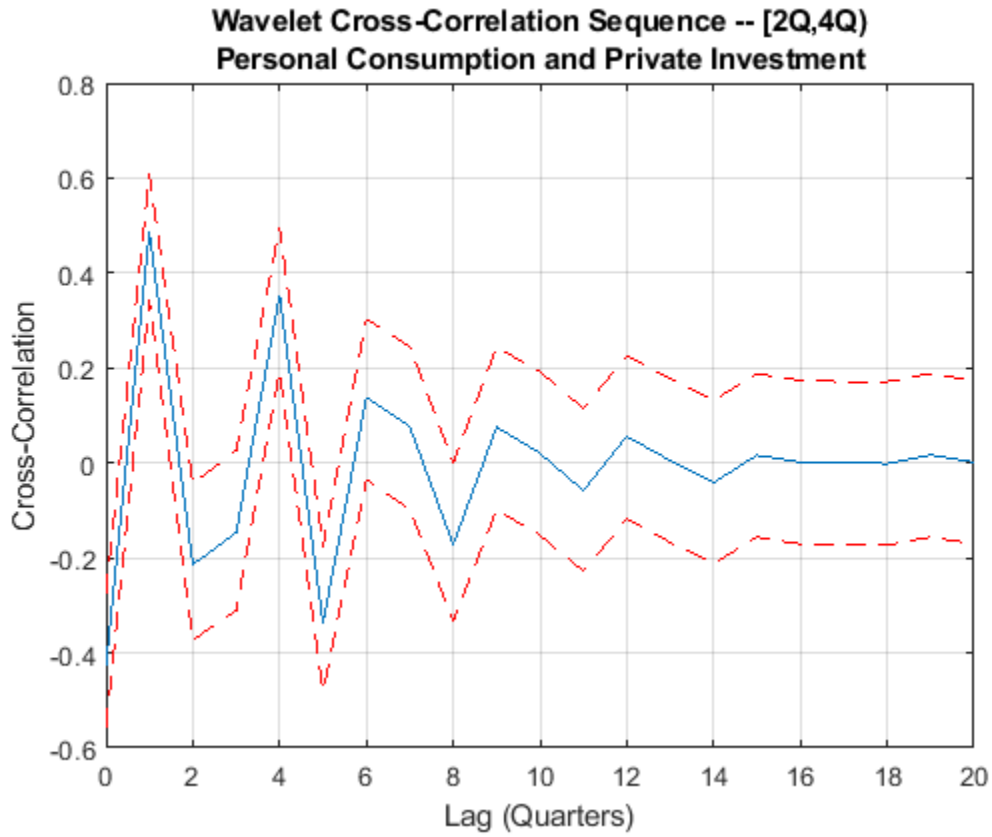
The finest-scale wavelet cross-correlation sequence shows a peak positive correlation at a lag of one quarter. This indicates that personal investment lags personal expenditures by one quarter. If you take that lagging relationship into account, then there is a positive correlation between the GDP components 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
chgwdeccfs	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
```
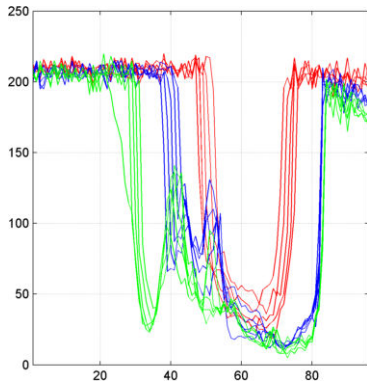

3 Perform a wavelet decomposition of signals at level 2 of row signals using the db2 wavelet.

```
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])
```
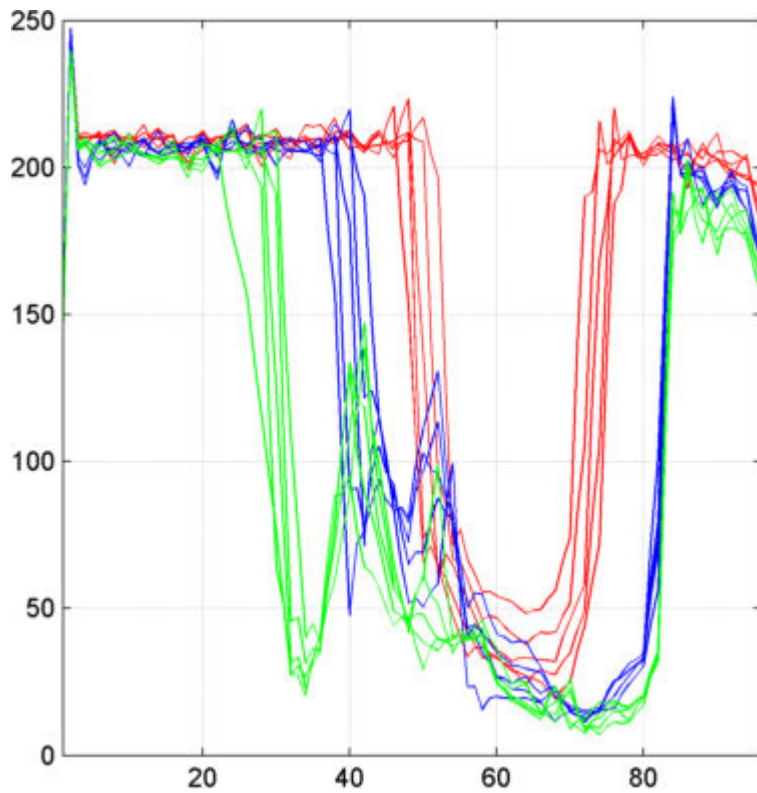
Compare old and new signals by plotting them together.

```
figure; idxSIG = [1 31];
plot(X(idxSIG,:)','r','linewidth',2);    hold on
plot(Xbis(idxSIG,:)','b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
```
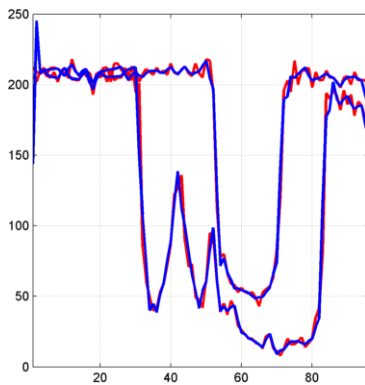

6 Set the wavelet coefficients at level 1 and 2 for signals 31 to 35 to the value zero, perform a wavelet reconstruction of signal 31, and compare some of the old and new signals.

```
decTER = chgwdeccfs(dec,'cd',0,1:2,31:35);
Y = mdwtrec(decTER,'a',0,31);
figure;
plot(X([1 31],:)','r','linewidth',2);    hold on
plot([Xbis(1,:)
```

```
; Y]','b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
```
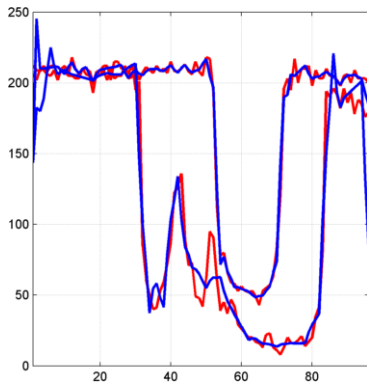

7 Compute the energy of signals and the percentage of energy for wavelet components.

```
[E,PEC,PECFS] = wdecenergy(dec);
```

Energy of signals 1 and 31:

```
Ener_1_31 = E([1 31])
Ener_1_31 =

   1.0e+006 *
     3.7534
     2.2411
```

8 Compute the percentage of energy for wavelet components of signals 1 and 31.

```
PEC_1_31 = PEC([1 31],:)

PEC_1_31 =
    99.7760     0.1718     0.0522
    99.3850     0.2926     0.3225
```

The first column shows the percentage of energy for approximations at level 2. Columns 2 and 3 show the percentage of energy for details at level 2 and 1, respectively.

9 Display the percentage of energy for wavelet coefficients of signals 1 and 31. As we can see in the dec structure, there are 26 coefficients for the approximation and the detail at level 2, and 49 coefficients for the detail at level 1.

```
PECFS_1 = PECFS(1,:); PECFS_31 = PECFS(31,:);
figure;
plot(PECFS_1,'r','linewidth',2); hold on
plot(PECFS_31,'b','linewidth',2);
grid; set(gca,'Xlim',[1,size(PECFS,2)])
```
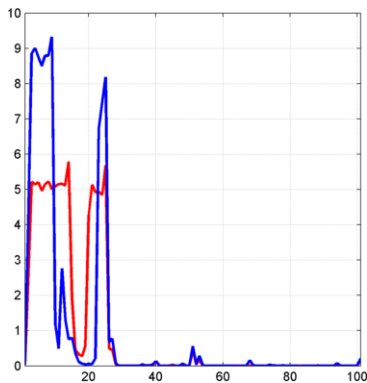
10 Compress the signals to obtain a percentage of zeros near 95% for the wavelet coefficients.

```
[XC,decCMP,THRESH] = mswcmp('cmp',dec,'N0_perf',95);
[Ecmp,PECcmp,PECFScmp] = wdecenergy(decCMP);
```

Plot the original signals 1 and 31, and the corresponding compressed signals.

```
figure;
plot(X([1 31],:)','r','linewidth',2);    hold on
plot(XC([1 31],:)','b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
```
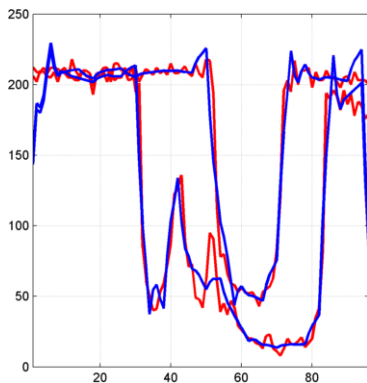

Compute thresholds, percentage of energy preserved and percentage of zeros associated with the `L2_perf` method preserving at least 95% of energy.

```
[THR_VAL,L2_Perf,N0_Perf] = mswcmptp(dec,'L2_perf',95);
idxSIG = [1,31];

Thr    = THR_VAL(idxSIG)
Thr =
   256.1914
   158.6085

L2per = L2_Perf(idxSIG)
L2per =
   96.5488
   94.7197

N0per = N0_Perf(idxSIG)
N0per =
```

```
79.2079
86.1386
```

Compress the signals to obtain a percentage of zeros near 60% for the wavelet coefficients.

```
[XC,decCMP,THRESH] = mswcmp('cmp',dec,'N0_perf',60);
```

XC signals are the compressed versions of the original signals in the row direction.

Compress the XC signals in the column direction

```
XX = mswcmp('cmpsig','c',XC,'db2',2,'N0_perf',60);
```

Plot original signals X and the compressed signals XX as images.

```
figure;
subplot(1,2,1); image(X)
subplot(1,2,2); image(XX)
colormap(pink(222))
```
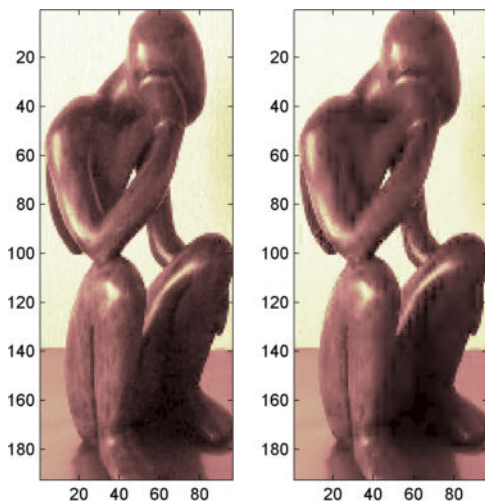

11 Denoise the signals using the universal threshold:

```
[XD,decDEN,THRESH] = mswden('den',dec,'sqtwolog','sln'); figure;
plot(X([1 31],:)','r','linewidth',2); hold on
plot(XD([1 31],:)','b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
```
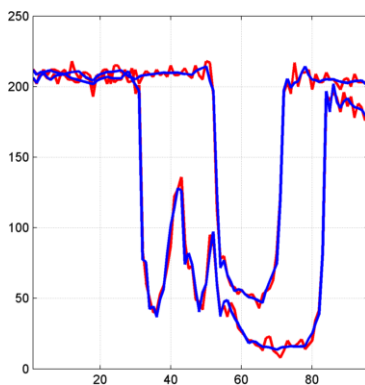

XD signals are the denoised versions of the original signals in the row direction.

Denoise the XD signals in column direction

```
XX = mswden('densig','c',XD,'db2',2,'sqtwolog','sln');
```

Plot original signals X and the denoised signals XX as images.

```
figure;
subplot(1,2,1); image(X)
subplot(1,2,2); image(XX)
colormap(pink(222))
```
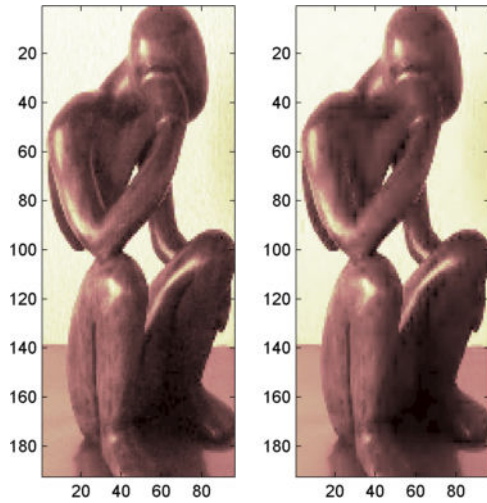

1-D Multisignal Analysis Using the Wavelet Analyzer App

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.
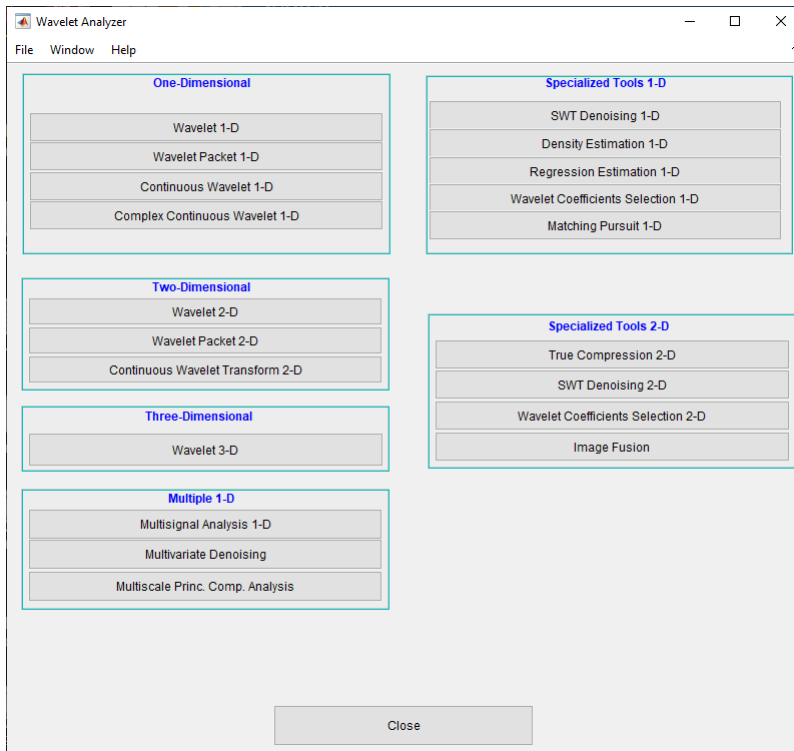
Click **Multisignal Analysis 1-D** to open the Wavelet 1-D Multisignal Analysis tool.
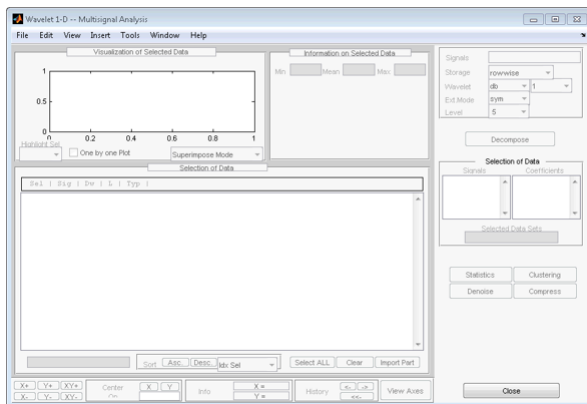
The tool is divided into five panes. Two of them are the same as in all Wavelet Toolbox app tools, the Command Frame on the right side of the figure and the Dynamic Visualization tool at the bottom. The Command Frame contains a special component found in all multisignal tools: the **Selection of Data 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.
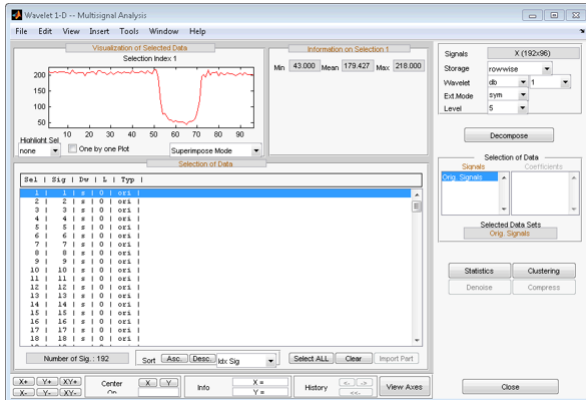
The **Selection of Data** pane contains a list of selectable signals. At the beginning, only the originally loaded signals are available. You can generate and add new signals to the list by decomposing, compressing, or denoising original signals.

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.
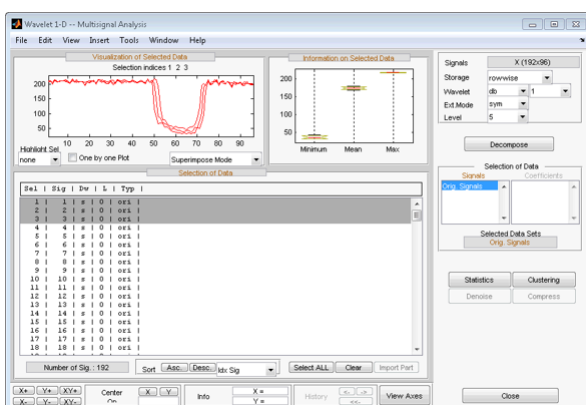
4 Highlight a signal.

Using the **Highlight Sel** button in the lower-left corner of the **Visualization of Selected Data** pane, select signal 3.
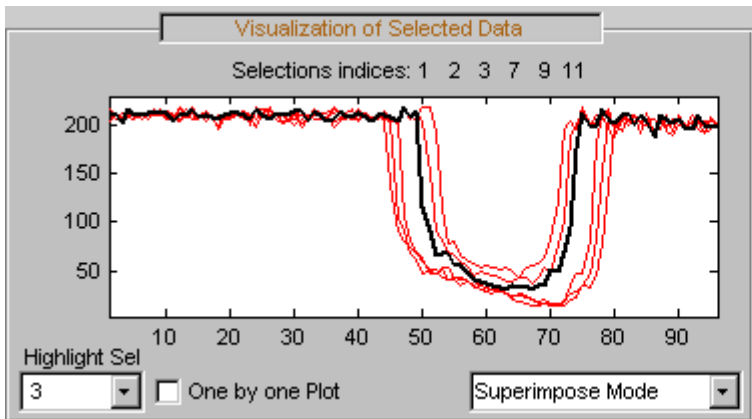
5 Select Different Views.

In the **Visualization of Selected Data** pane, change the view mode using the pop-up in the lower-right corner. Choose **Separate Mode**. The selected signals appear.

6 Decompose a multisignal.

Perform an analysis at level 4 using the `db2` wavelet and the same file used in the command line section: `thinker.mat`.

In the upper right portion of the Wavelet 1-D Multisignal Analysis tool, select `db2` and level 4 in the Wavelet fields.
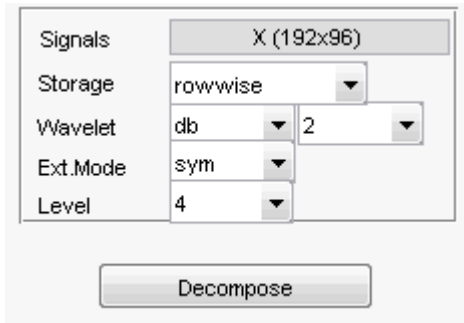
Click **Decompose**. After a pause for computation, all the original signals are decomposed and signal 1 is automatically selected
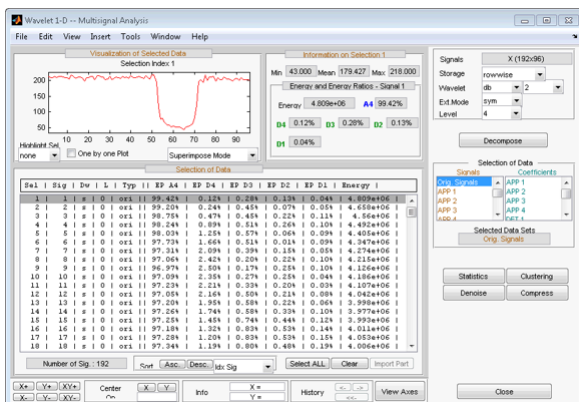
In the **Selection of Data** pane, new information is added for each original signal — the percentage of energy of the wavelet components (**D1**,...,**D4** and **A4**) and the total energy. The **Information on Selected Data** pane contains information on the single selected signal: **Min**, **Mean**, **Max** and the energy distribution of the signal.
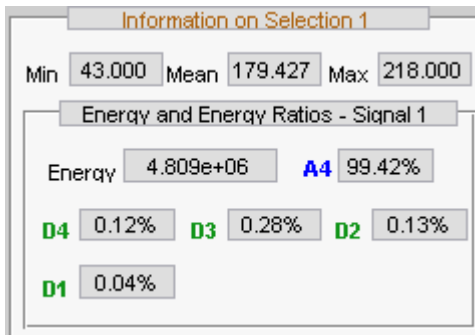
Since the original signals are decomposed, new objects appear and the **Selection of Data Sets** pane in the Command Frame updates.

The **Selection of Data Sets** pane defines the available signals that are now selectable from the **Selection of Data** pane.
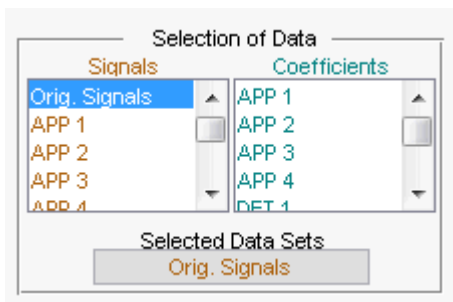
The list on the left allows you to select sets of signals and the right list allows you to select sets of corresponding coefficients: original signals (`Orig. Signals`), approximations (`APP 1`,...)and details from levels 1 to 4 (`DET 1`,...).

In the list on the right, the coefficients vectors can be of different lengths, but only components of the same length can be selected together.

After a decomposition the original signals (`Orig. Signals`) data set appears automatically selected.

Select signals 1, 2, 3, 7, 9 and 11.
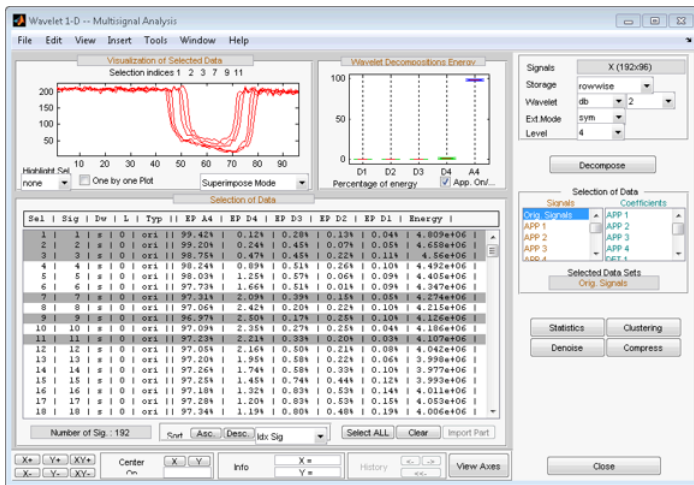
The energy of selected signals is primarily concentrated in the approximation A4, so the box plot is crushed (see following figure on the left). Deselect **App. On/Off** to see a better representation of details energy (see following figure on the right).
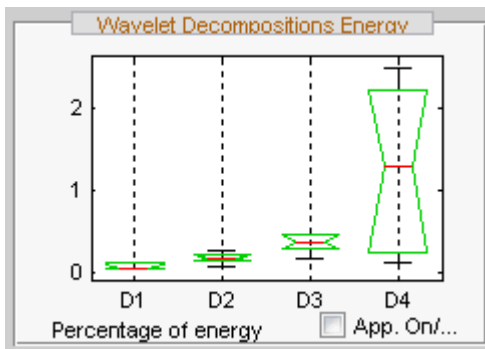
7   Display multisignal decompositions.

In the **Visualization of Selected Data** pane, change the view mode using the pop-up below the plots and select Full Dec Mode. The decompositions of the selected signals display.
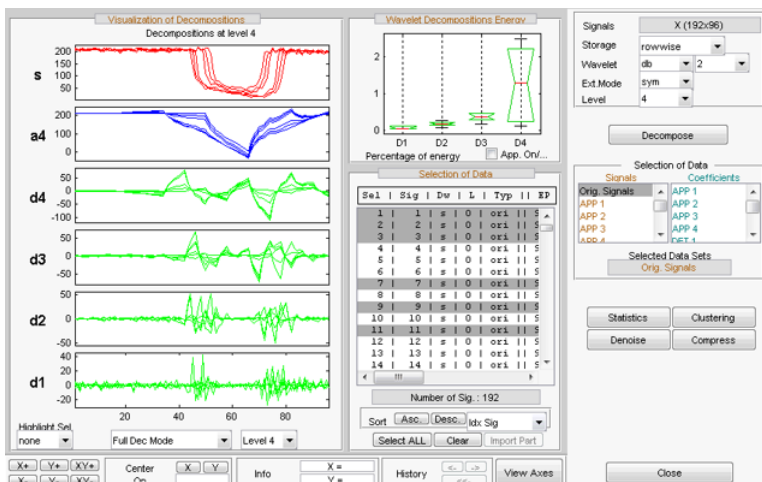
Change the **Level** to 2.

Select the signal 7 in **Highlight Sel.**

8 Change the visualization modes.

Using the second pop-up from the left at the bottom of the pane, select `Full Dec Mode (Cfs)`. The coefficients of the decompositions of the selected signals display. At level k, coefficients are duplicated 2k times.

Change the view mode to `Stem Mode (Abs)`, and then change to `Tree Mode`. The wavelet tree corresponding to the decompositions of the selected signals displays.

Select the level 4 and click the node a3. Then highlight signal 7.

9 Select Different Wavelet Components.

Ctrl-click **Orig. Signals**, **APP 1**, **APP 3** and **DET 1** to select these four sets of signals from the list on the left in the **Selection of Data Sets** pane.

The total number of selected data (`Number of Sig`.) appears in the **Selection of Data Sets** pane: four sets of 192 signals each is a total of 768 signals.
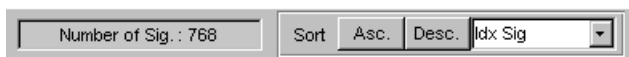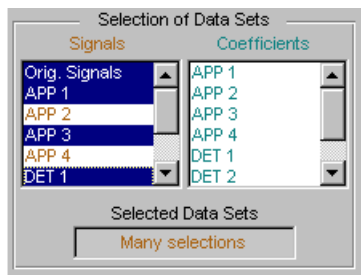
Click the **Asc**. button in the **Sort** pane. The selected data are sorted in ascending order with respect to the `Idx Sig` parameter
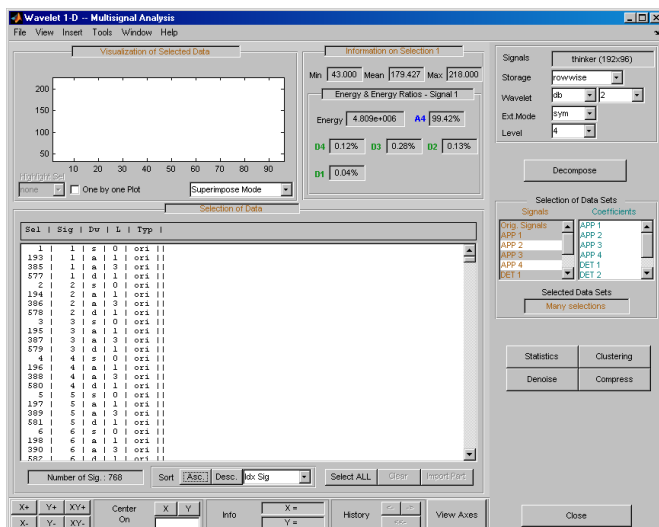
Note that DWT attributes of each selectable signal have been updated where **a** stands for approximation, **d** for detail and **s** for signal.

Click the `Idx Sel 1` signal and then shift-click the `Idx Sel 579` signal.
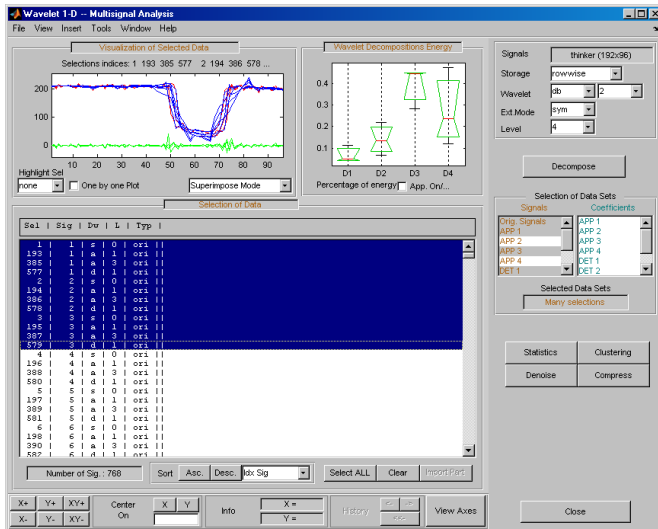
Choose `Separate Mode`.
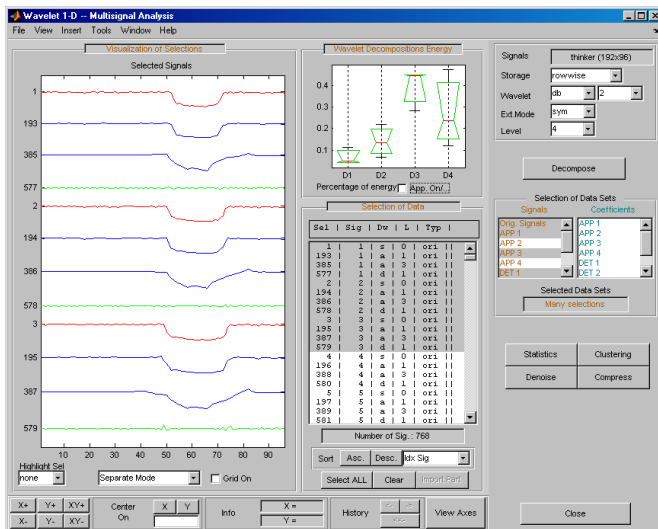
Ctrl-click to select two sets of signals from the right-most list of the **Selection of Data Sets** pane: `APP 1` and `DET 1`.
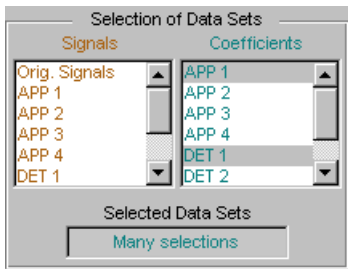
Note that in this list of coefficients sets, the selected vectors must be of same length, which means that you must select components of the same level.

Click the **Asc.** button in the Sort pane. The selected data are sorted in ascending order with respect to Idx Sig parameter.

Select the ten first signals.
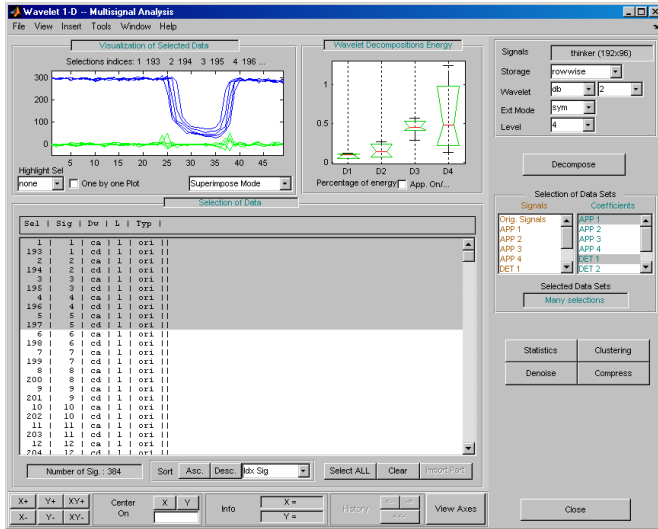
10 Compress a multisignal.

The 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.
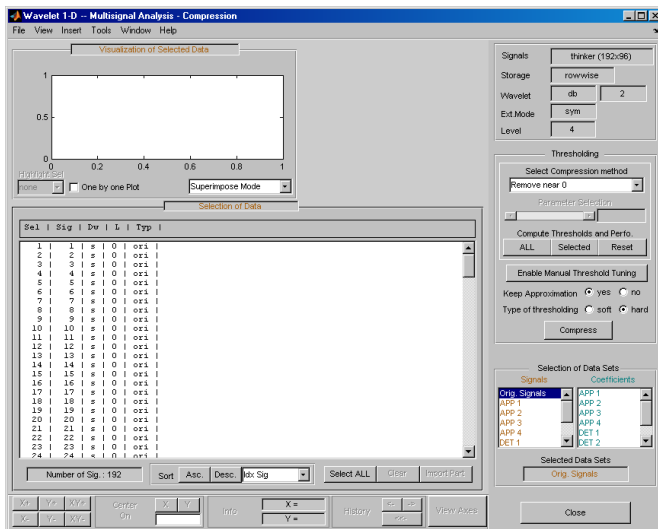
Note The tool always compresses all the original signals when you click the **Compress** button.

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.

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.
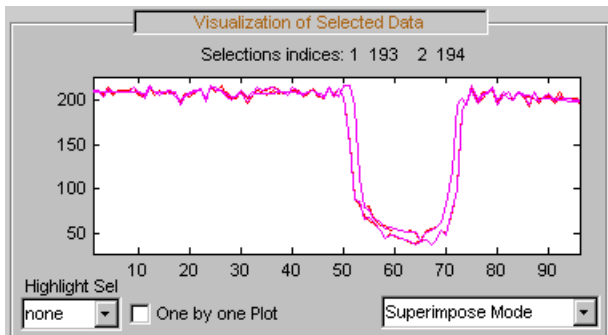
Click the **Close** button to close the Compression window.

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.
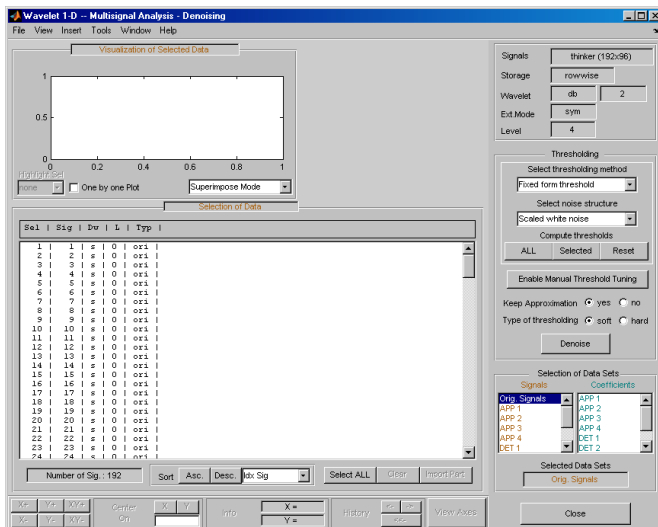
A number of options are available for fine-tuning the denoising algorithm. For this example, accept the defaults: `soft` type of thresholding, `Fixed form threshold` method, and `Scaled white noise` as noise structure.

Click the **ALL** button in the Thresholding pane. The threshold for each level (**ThrD1**, …, **ThrD4**) computes and displays in the **Selection of Data** pane.
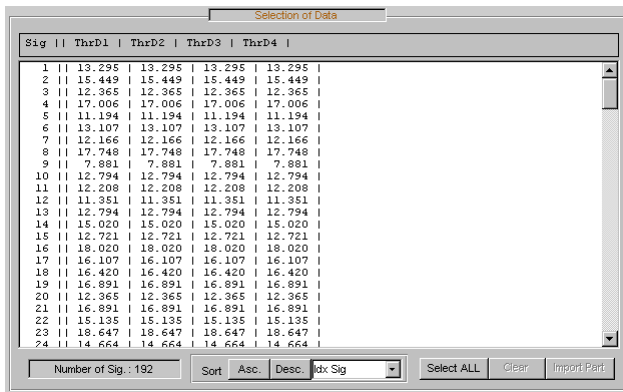
Then click the **Denoise** button at the bottom of the **Thresholding** pane.

Ctrl-click the **Denoised** item in the list on the left of the **Selection of Data Sets** pane. The original signals and the corresponding denoised ones are selected (2 x 192 = 384 signals).

Click the **Asc.** button at the bottom of the **Selection of Data** pane to sort the signals according to the Idx Sig parameter.

With the mouse, select the first four signals. They correspond to the original signals 1, 2 and the corresponding denoised signals 193, 194
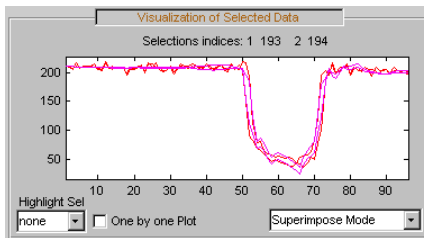
Choose `Separate Mode`.

12 To view residuals, Ctrl-click the `Orig. Signal`, the `Denoised` and the `Residuals` items in the list on the left of the **Selection of Data Sets** pane. Original, denoised and residual signals are selected (3 x 192 = 576 signals).

Click the **Asc.** button at the bottom of the **Selection of Data** pane to sort the signals using the `Idx Sig` parameter.

With the mouse, select the first six signals. They correspond to the original signals 1, 2, the corresponding denoised signals 193, 194 and the residuals 385, 386.

Then, choose `Separate Mode`.
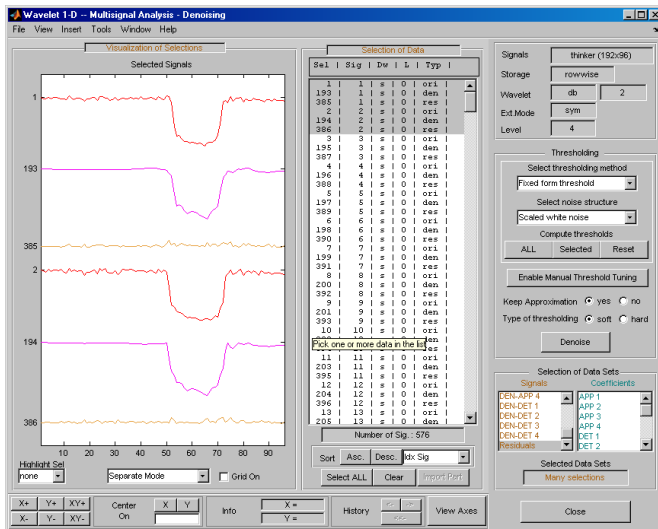
13 Click **Close** to close the denoising tool. Then, click the **Yes** button to update the synthesized signals.
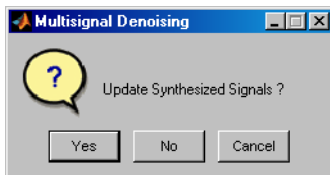
Manual Threshold Tuning

1 Choose a method, select one or several signals in the **Selection of Data** pane using the mouse and keys. Then click the **Selected** button. You can select another group of signals using the same method. Press the **Denoise** button to denoise the selected signal(s).
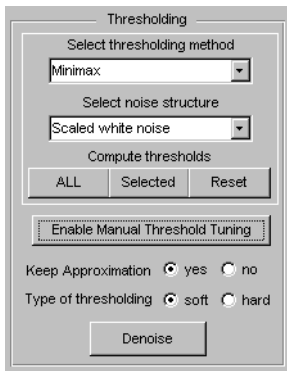
You can also use manual threshold tuning. Click the **Enable Manual Thresholding Tuning** button.
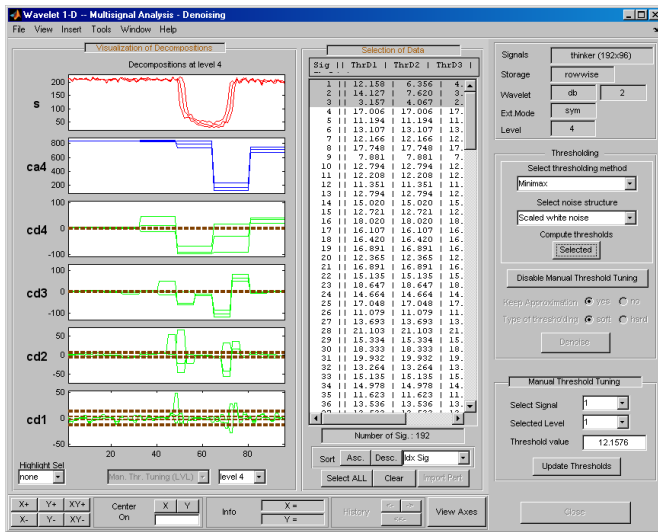
The horizontal lines in the wavelet coefficient axes (`cd1`, ..., `cd4`) can be dragged using the mouse. This may be done individually, by group or all together depending on the values in the **Select Signal** and **Selected Level** fields in the **Manual Threshold Tuning** pane.
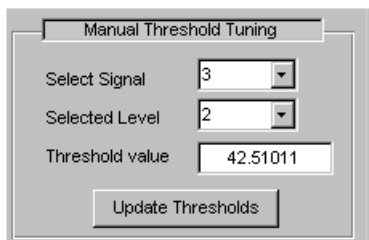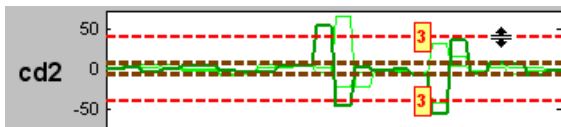
2 In the Wavelet 1-D Multisignal Analysis Compression tool, you can use two methods for threshold tuning: the **By level thresholding** method which is used in the Wavelet 1-D Multisignal Analysis Denoising tool, and the **Global thresholding method**.
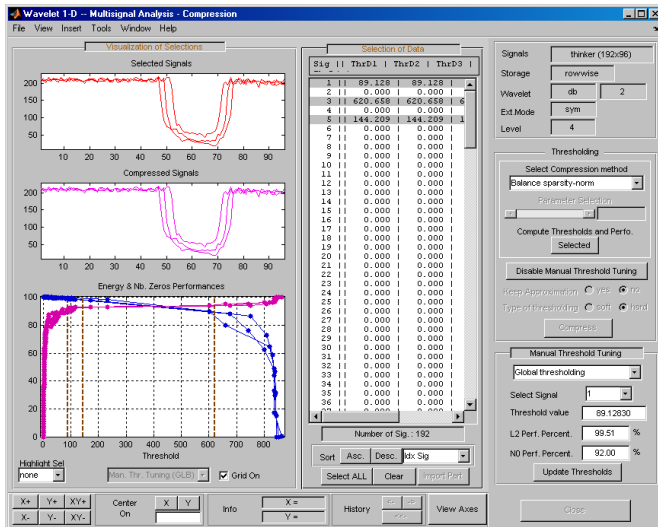
You can drag the vertical lines in the **Energy and Nb. Zeros Performances** axes using the mouse. This can be done individually or all together depending on the values of **Select Signal** in the **Manual Threshold Tuning** pane.
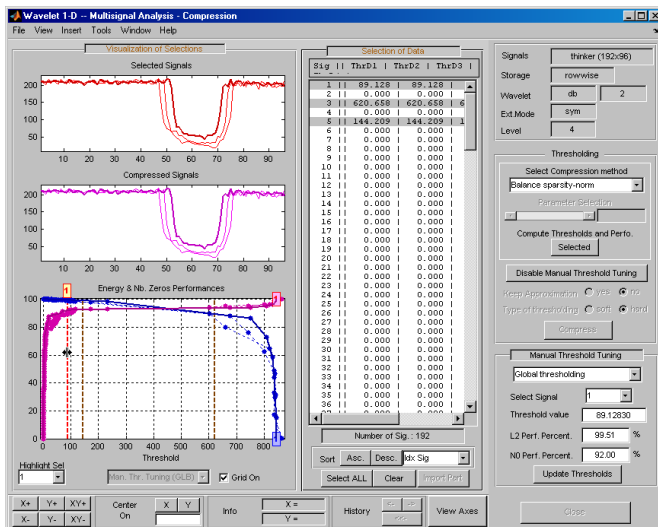
The threshold value, L2 performance, and number of zeros performance are updated in the corresponding edit buttons in the **Manual Threshold Tuning** pane.

Statistics on Signals

1   You can display various statistical parameters related to the signals and their components. From the Wavelet 1-D Multisignal Analysis tool, click the **Statistics** button. Then select the signal 1 in the **Selection of Data Sets** pane.
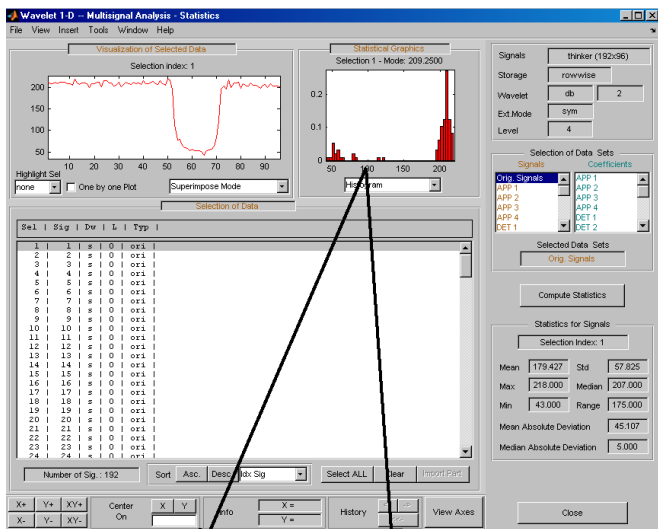
Select the signals 1, 2, 3, 7, 9 and 11 in the **Selection of Data** pane, and display the corresponding boxplots and correlation plots.
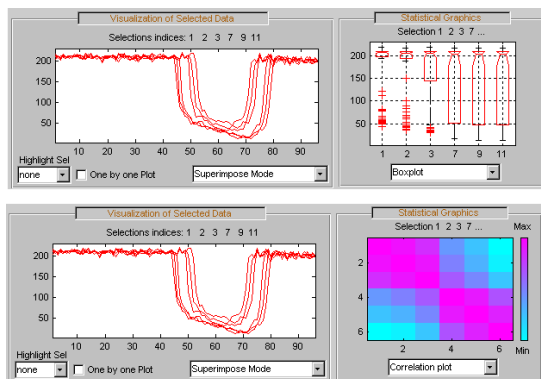
2 To display statistics on many wavelet components, in the **Selection Data Sets** pane, in the left column, select `Orig.Signals`, `APP 1`, `DET 1`, Denoised and `Residuals` signals. Then choose `Separate Mode`, and click the **Asc.** button in the **Sort** pane. The selected data are sorted in ascending order with respect to `Idx Sig` parameter. In the **Selection of Data** pane, select data related to signal 1.
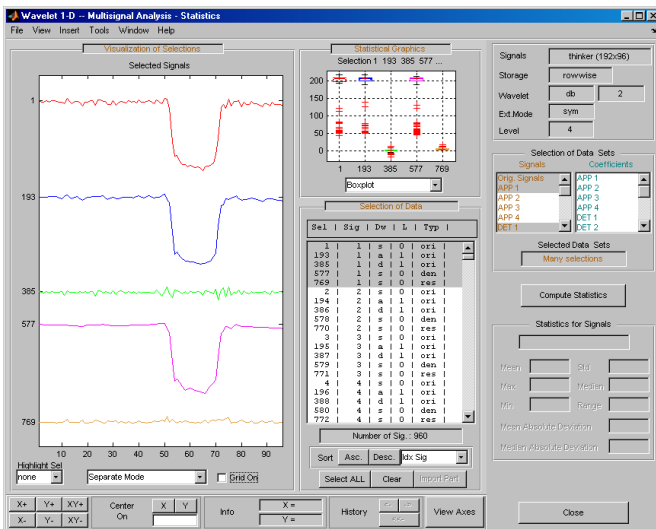
Clustering Signals

Note To use clustering, you must have Statistics 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).

1   Click the **Clustering** button located in the Command Frame, which is in the lower right of the Wavelet 1-D Multisignal Analysis window to open the Clustering tool.
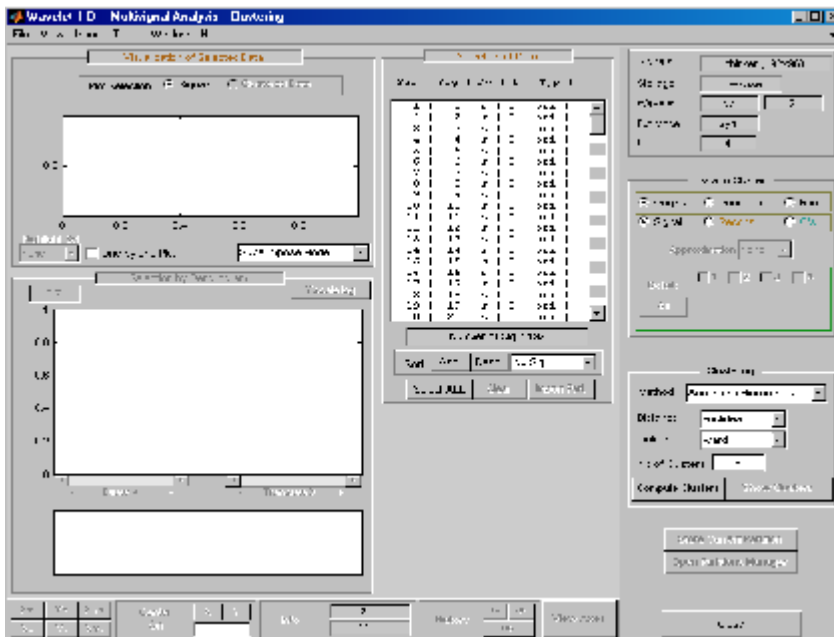
You can cluster various type of signals and wavelet components: original, denoised or compressed, residuals, and approximations or details (reconstructed or coefficients). Similarly, there are several methods for constructing partitions of data.
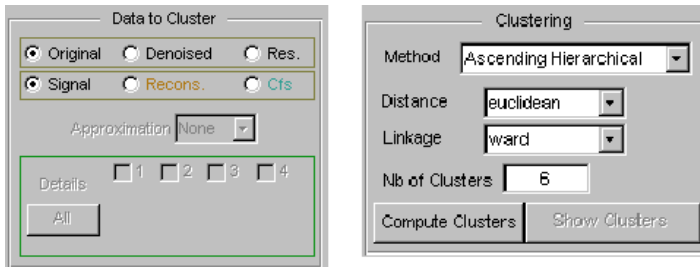
Use the default parameters (`Original` and `Signal` in Data to Cluster, and in `Ascending Hierarchical, euclidean, ward`, and `6` in Clustering) and click the **Compute Clusters** button.

A full dendrogram and a restricted dendrogram display in the **Selection by Dendrogram** pane. For each signal, the cluster number displays in the **Selection of Data** pane.
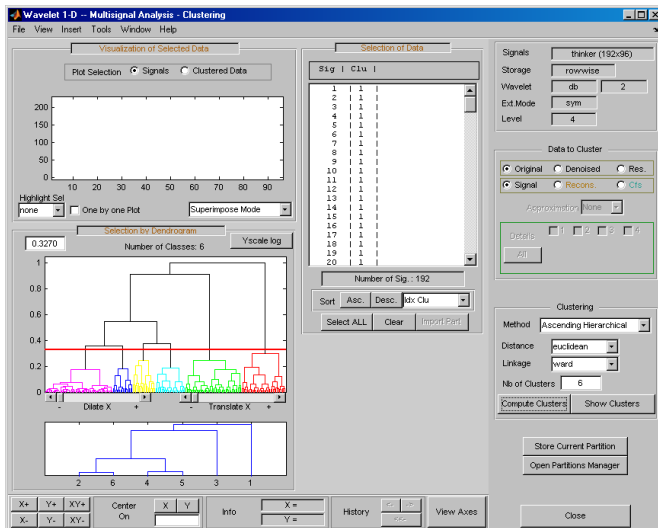
2 Select one cluster, several clusters, or a part of a cluster.

Click the `xticklabel 3` at the bottom of the restricted dendrogram. The links of the third cluster blink in the full dendrogram and the 24 signals of this class display in the **Visualization of Selected Data** pane. You can see their numbers in the **Selection of Data** pane.

Clicking the line in the restricted or in the full dendrogram lets you select one cluster, several linked clusters, or a part of a cluster. For a more accurate selection, use the `Dilate X` and the `Translate X` sliders under the full dendrogram. You can also use the **Yscale** button located above the full dendrogram. The corresponding signals display in the **Visualization of Selected Data** pane and in the list of the **Selection of Data** pane.
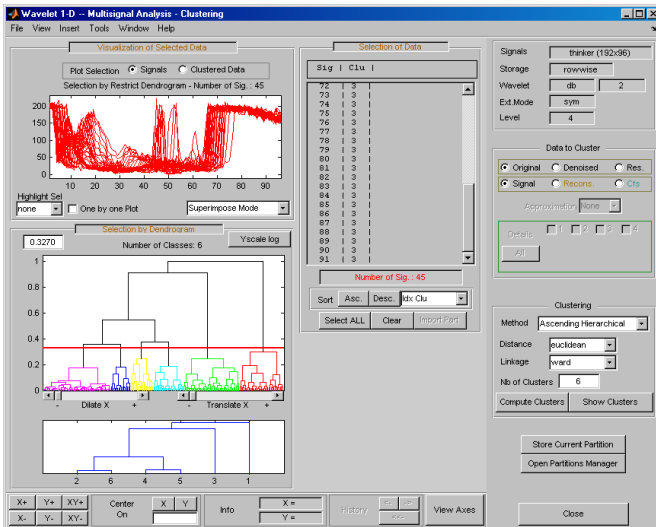
You can use the horizontal line in the full dendrogram to change the number of clusters. Use the left mouse button to drag the line up or down.
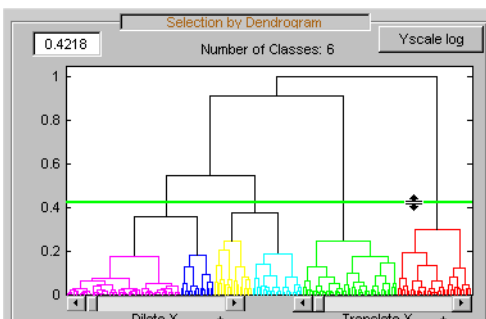
3 Use the **Show Clusters** button to examine the clusters of the current partition. You can display the mean (or the median) of each cluster, the global standard deviation and the pointwise standard deviation distance around the mean (or the median). Each plot title contains: the number of signals in 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 $Q2 = 37.208/(220\text{-}5) = 0.173$.
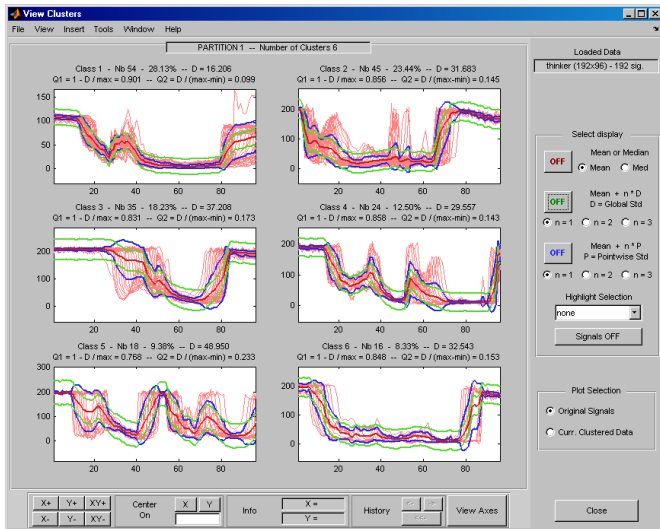
4   Click the **Store Current Partition** button below the **Clustering** pane to store the current partition for further comparisons. A default name is suggested. Note that the 1-D Wavelet Multisignal Analysis tool stores the partitions and they are not saved on the disk.
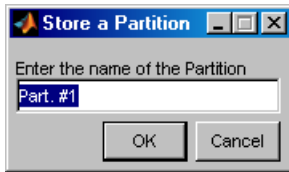
Partitions

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.
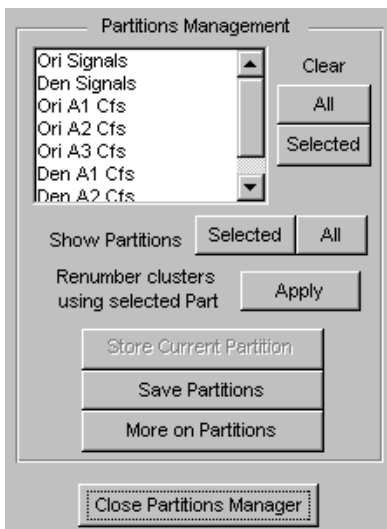
Now, you can show, clear, or save the partitions (individually, selected ones, or all together).

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.
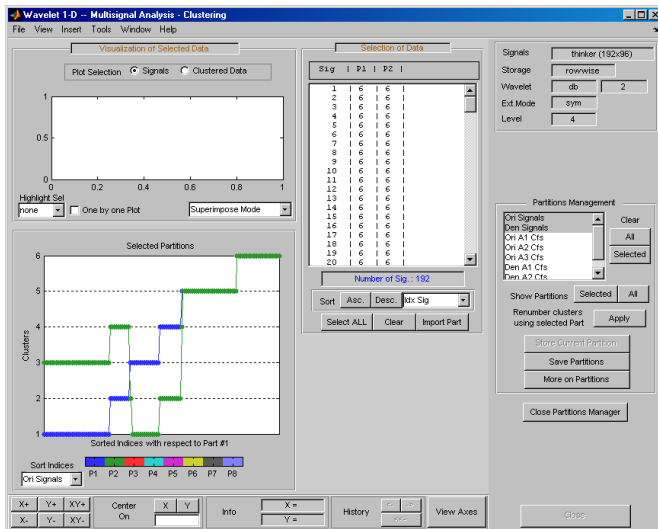
Only three signals are not classified in the same cluster for the two considered partitions.
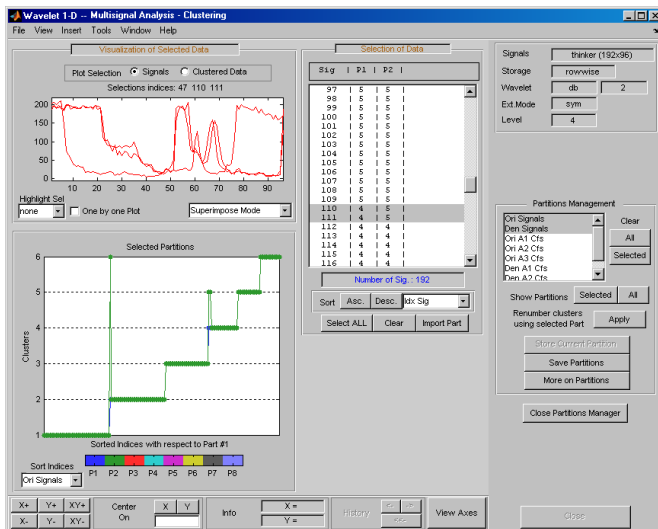
3 Select the partitions you want to save and click the **Save Partitions** button below the **Store Current Partition** button in the **Partitions Management** pane.
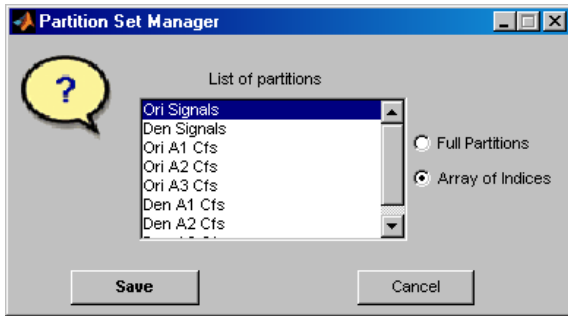
Partitions are saved as an array of integers, where each column corresponds to one partition and contains the indices of clusters. When you choose the `Full Partitions` option, an array object (`wpartobj`) is saved.

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.)
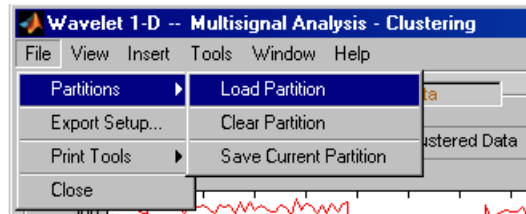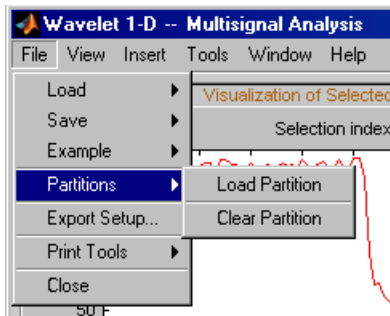
To clear one or more stored partitions, select **File > Partitions > Clear Partition**.
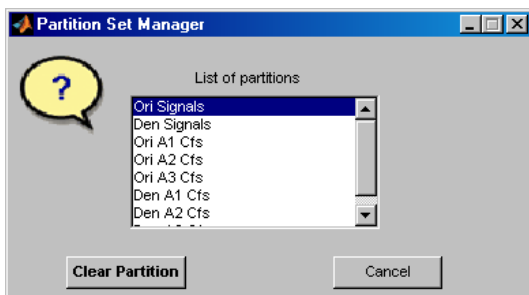
Select **File > Partitions > Load Partition** to load one or several partitions from the disk. The loaded partitions are stored in Wavelet 1-D Multisignal Analysis tool with any previously stored partitions. A partition can also be a manually created column vector.

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.
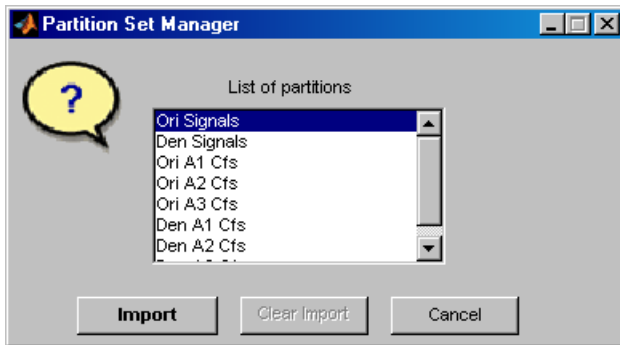
For this example, go back to the main window, import the `Ori Signals` partition and sort the signals in descending order with respect to **A4** energy percentage.
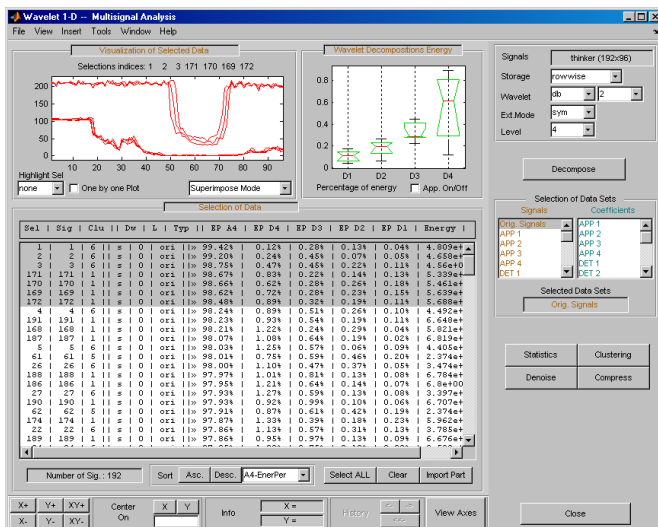
6 You can compare partitions with the Partition tool. To display the Partition tool, click the **More on Partitions** button at the bottom of the Partitions Management pane. By default, when the Partition tool opens, the currently selected partition, in this case `Ori Signals`, is compared with itself. In the lower left plot, an integer N at `(i,j)` means there is a group of N signals in the ith cluster of P1 and the jth cluster of P2. Since the Partition tool is comparing a partition with itself, all the numbers are plotted on the main diagonal.
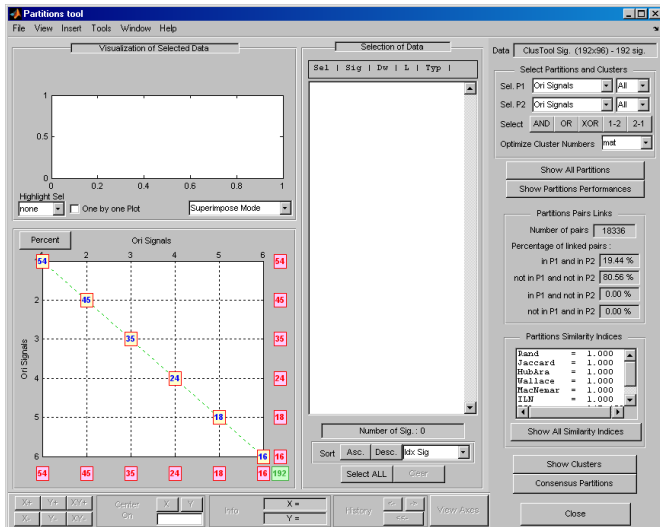
Statistics measuring the similarity of the partitions P1 and P2 are displayed in the panels on the right. The **Partitions Pairs Links** panel counts pairs of signals. Two signals x and y are considered a *pair* if they are in the same cluster. For any two signals x and y, there are four possibilities.

- 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*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): $(R\text{-}ER)/(MR\text{-}ER)$ 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): $R/\text{sqrt}(|pi(P1)| \times |pi(P2)|)$ where $|pi(Pi)|$ is the number of joined pairs in the partition P*i*.

- McNemar Index (`MacNemar`): If abs($U+V$) = 0, then the index is equal to 1. Otherwise, the index is equal to 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) × 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.
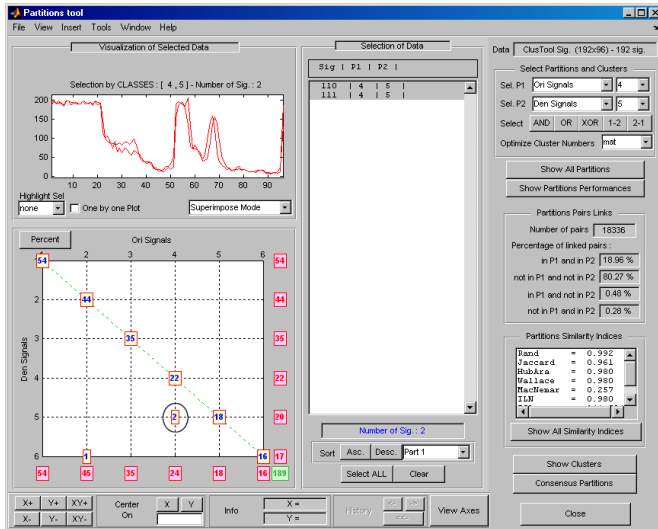
More on Clustering

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.
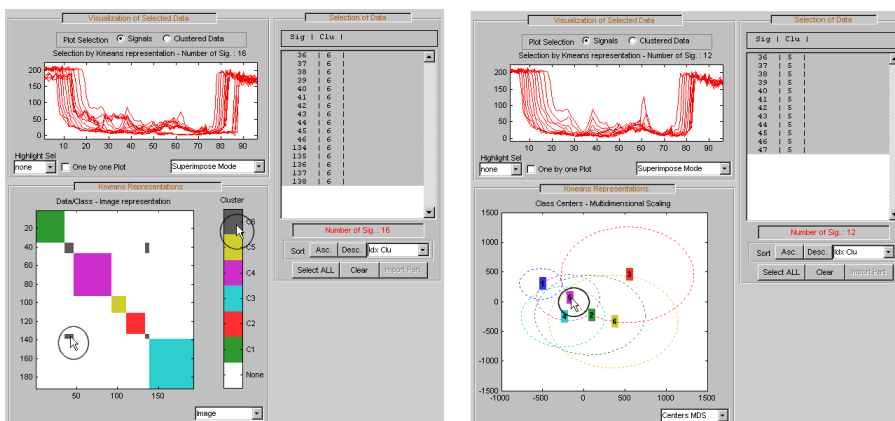
Importing and Exporting Information from the Wavelet Analyzer App

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.
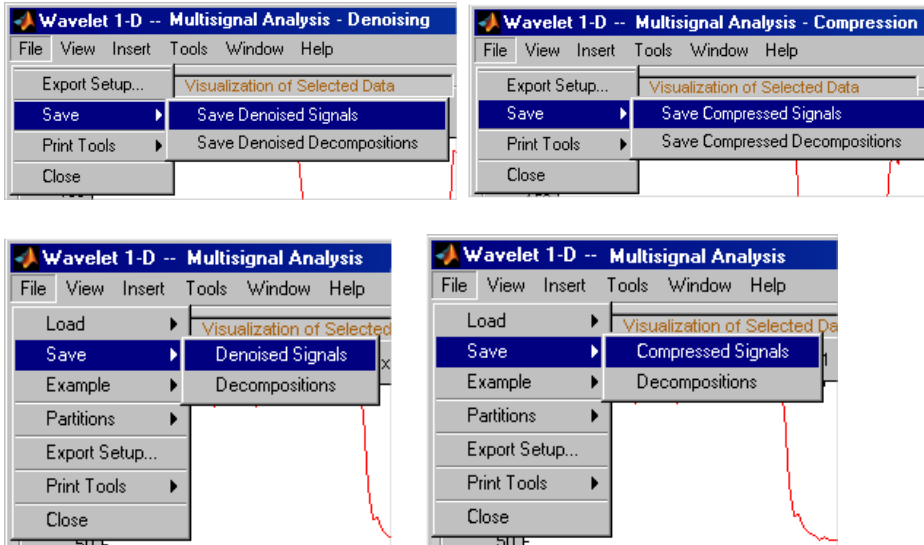
Saving Decompositions

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 3-146.

Loading Information into the Wavelet 1-D Multisignal Analysis Tool

You can load signals or decompositions into the graphical interface. The information you load may be previously exported from the graphical interface, and then manipulated in the workspace; or it may be information you initially generated from the command line. In either case, you must observe the strict file formats and data structures used by the Wavelet 1-D Multisignal Analysis tools or errors will occur when you try to load information.
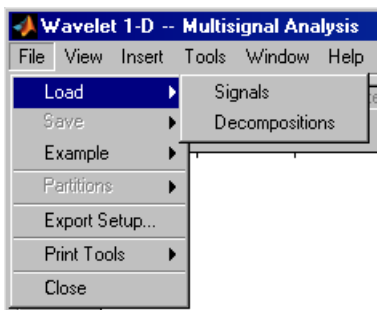
Loading Signals

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

For example, if you design a signal called `magic128` and want to analyze it in the Wavelet 1-D Multisignal Analysis tool, type

```
save magic128 magic128
```

> **Note** The workspace variable `magic128` must be a matrix and the number of rows and columns must be greater than 1.

```
sizmag = size(magic128)

sizmag =
    128   128
```

To load this signal into the Wavelet 1-D Multisignal Analysis tool, use the **File > Load Signal** menu item. A dialog box appears in which you select the appropriate MAT-file to be loaded.

> **Note** When you load a matrix of signals from the disk, the name of 2-D variables are inspected in the following order: `x`, `X`, `sigDATA`, and `signals`. Then, the 2-D variables encountered in the file are inspected in alphabetical order.

Loading Decompositions

To load decompositions that you constructed in the MATLAB workspace into the Wavelet 1-D Multisignal Analysis tool, save the signal in a MAT-file (with extension `mat`).

For instance, if you design a signal called `magic128` and want to analyze it in the Wavelet 1-D Multisignal Analysis too, the structure `dec` must have the following fields:

`'dirDec'`	Direction indicator with `'r'` for row or `'c'` for column
`'level'`	Level of DWT decomposition
`'wname'`	Wavelet name
`'dwtFilters'`	Structure with four fields: `LoD`, `HiD`, `LoR`, `HiR`
`'dwtEXTM'`	DWT extension mode (see `dwtmode`)
`'dwtShift'`	DWT shift parameter (0 or 1)
`'dataSize'`	Size of original matrix X
`'ca'`	Approximation coefficients at level `dec.level`
`'cd'`	Cell array of detail coefficients, from 1 to `dec.level`

The coefficients `cA` and `cD{k}`, for (`k = 1` to `dec.level`), are matrices and are stored 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.
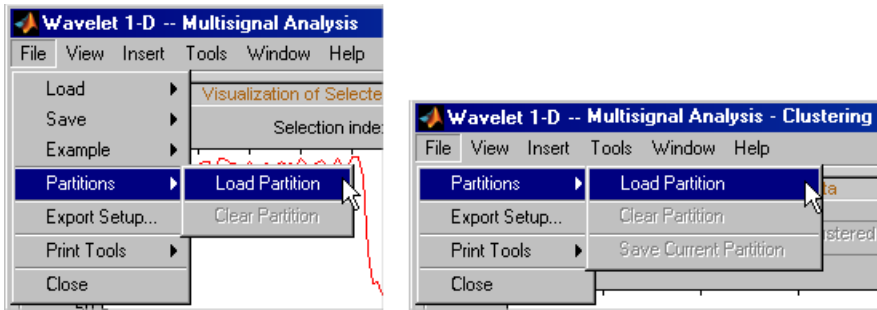
Saving Partitions

The Wavelet 1-D Multisignal Analysis clustering tool lets you save a set of partitions to disk.
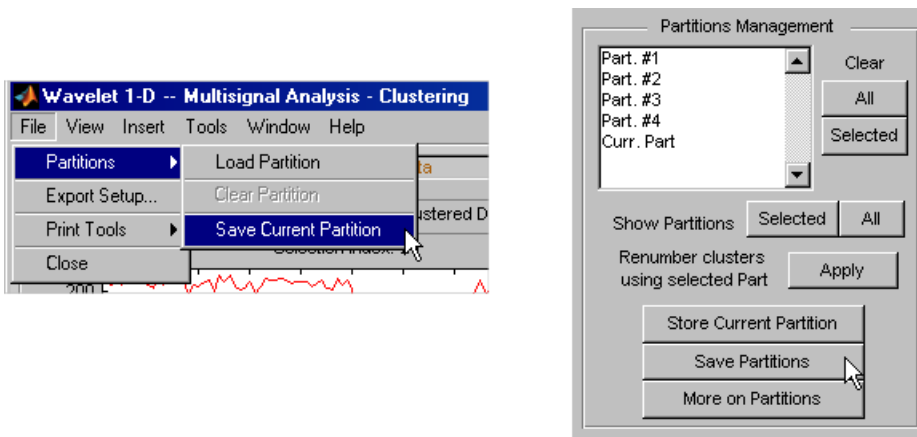
For example:

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.
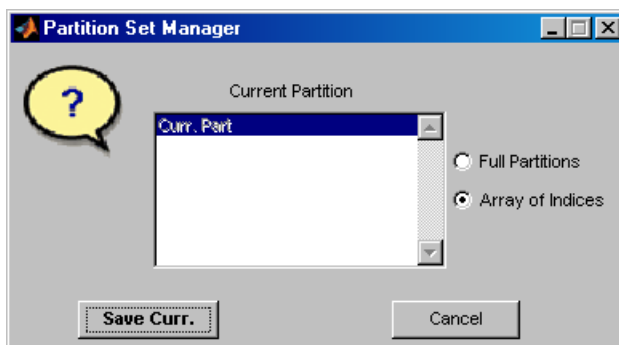
5 Another dialog box appears that lets you specify a folder and filename for storing the partition data. Type the name `curPART`.

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
dwt2	Single-level decomposition
wavedec2	Decomposition
wmaxlev	Maximum wavelet decomposition level

Synthesis-Reconstruction Functions

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

Decomposition Structure Utilities

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

Denoising and Compression

Function Name	Purpose
wdenoise2	Wavelet image denoising
ddencmp	Provide default values for denoising and compression
wbmpen	Penalized threshold for wavelet 1-D or 2-D denoising
wdcbm2	Thresholds for wavelet 2-D using Birgé-Massart strategy
wdencmp	Wavelet denoising and compression

Function Name	Purpose
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

Wavelet Image Analysis and Compression

This example shows 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`).

Load an image.

```
load wbarb
whos X map
```

```
  Name        Size             Bytes  Class      Attributes

  X         256x256           524288  double
  map       192x3               4608  double
```

Display the image.

```
image(X)
colormap(map)
colorbar
```
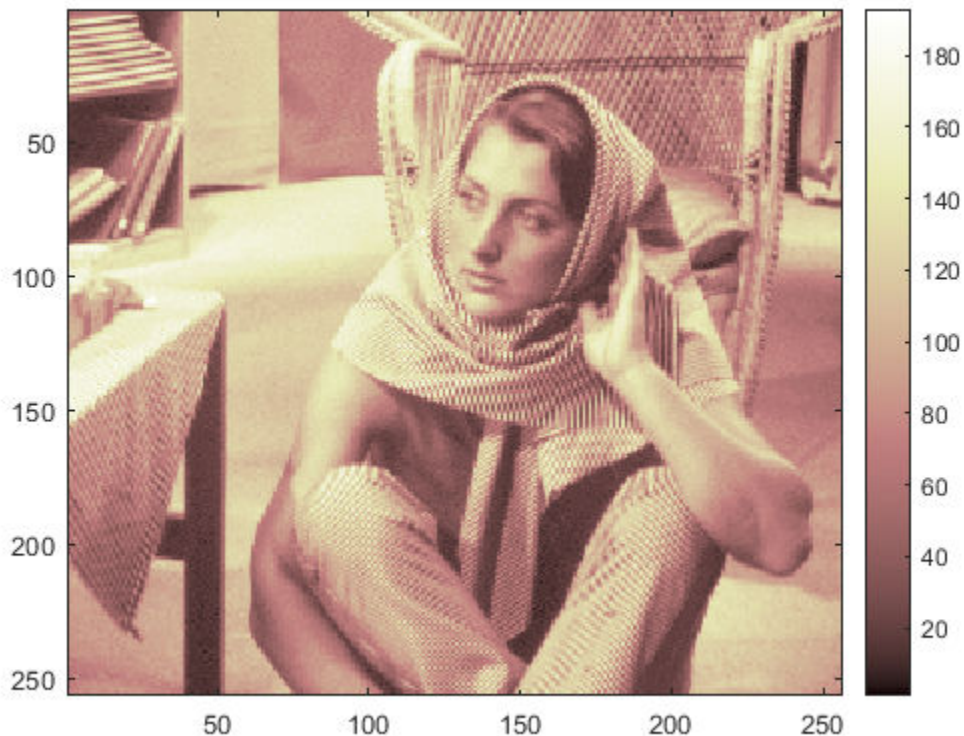
If the colormap is smooth, the wavelet transform can be directly applied to the indexed image; otherwise the indexed image should be converted to grayscale format. For more information, see "Wavelets: Working with Images". Since the colormap is smooth in this image, you can now perform the decomposition.

Perform a single-level wavelet decomposition of the image using the `bior3.7` wavelet. The coefficient matrix `cA1` are the approximation coefficients. The horizontal, vertical, and diagonal details are in the matrices `cH1`, `cV1`, and `cD1`, respectively.

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

Use `idwt1` to construct the approximations and details from the coefficients. (Note: You can also use `upcoef2`.

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

Display the approximations and details.

```
figure
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')
colormap(map)
```
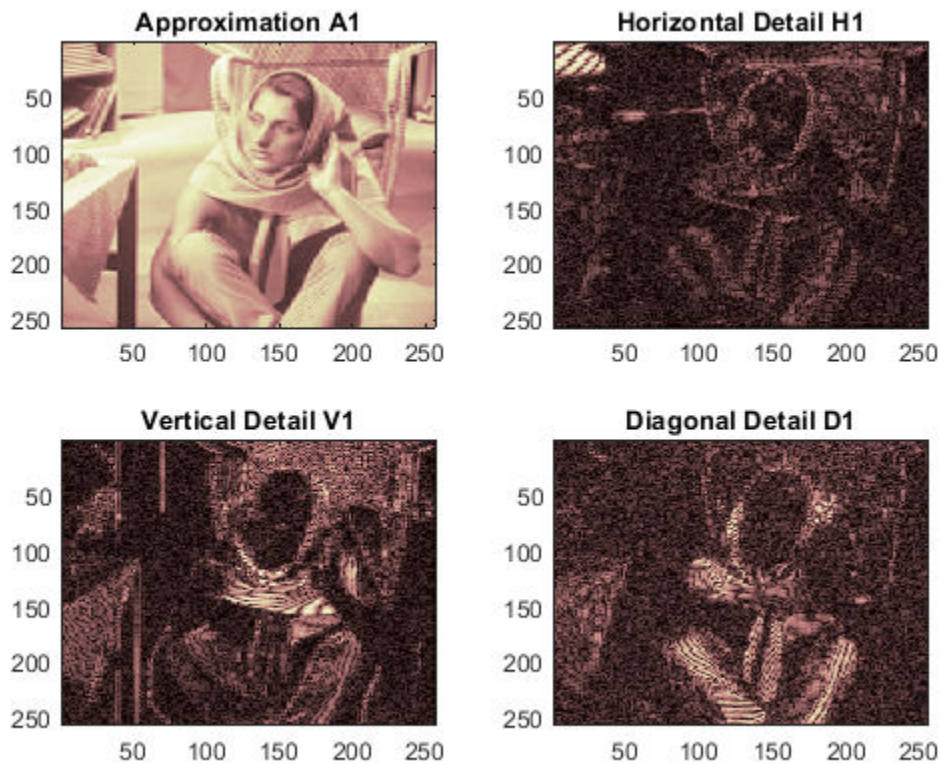

Regenerate the image by the single-level inverse discrete wavelet transform. Confirm the difference between the regenerated and original images are small.

```
Xrec = idwt2(cA1,cH1,cV1,cD1,wv);
max(abs(X(:)-Xrec(:)))
```

```
ans = 1.4211e-13
```

Perform a level-2 wavelet decomposition of the image using the same `bior3.7` wavelet. 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.

```
[c,s] = wavedec2(X,2,wv);
```

Extract the level 2 approximation coefficients. Extract the first- and second-level detail coefficients.

```
cA2 = appcoef2(c,s,wv,2);
[cH2,cV2,cD2] = detcoef2('all',c,s,2);
[cH1,cV1,cD1] = detcoef2('all',c,s,1);
```

Reconstruct the level 2 approximation and the level 1 and level 2 details.

```
A2 = wrcoef2('a',c,s,wv,2);
H1 = wrcoef2('h',c,s,wv,1);
V1 = wrcoef2('v',c,s,wv,1);
D1 = wrcoef2('d',c,s,wv,1);
H2 = wrcoef2('h',c,s,wv,2);
V2 = wrcoef2('v',c,s,wv,2);
D2 = wrcoef2('d',c,s,wv,2);
```

Display the approximation and details.

```
figure
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')
colormap(map)
```
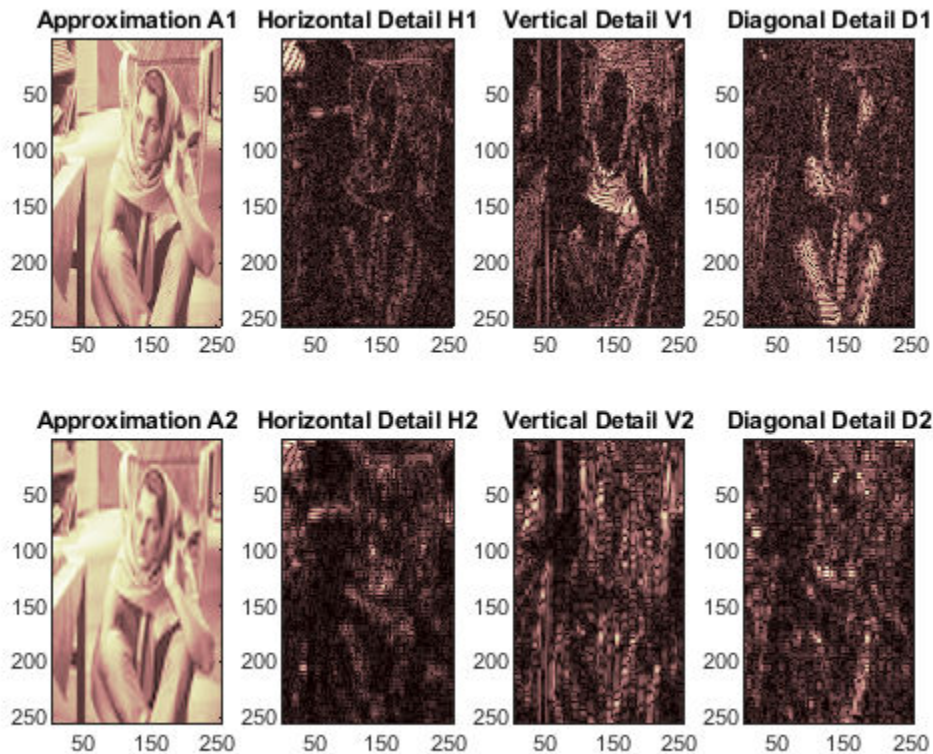
Compress the image. Use **ddencmp** to calculate the default parameters and **wdencmp** to perform the actual compression.

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

Compare the compressed image with the original image.

```
fprintf('Percentage of wavelet coefficients set to zero: %.4f\nPercentage of energy preserved: %
    PERF0,PERFL2);
```

```
Percentage of wavelet coefficients set to zero: 49.8011
Percentage of energy preserved: 99.9817
```

```
figure
subplot(121)
image(X)
title('Original Image')
axis square
subplot(122)
image(Xcomp)
title('Compressed Image')
axis square
colormap(map)
```
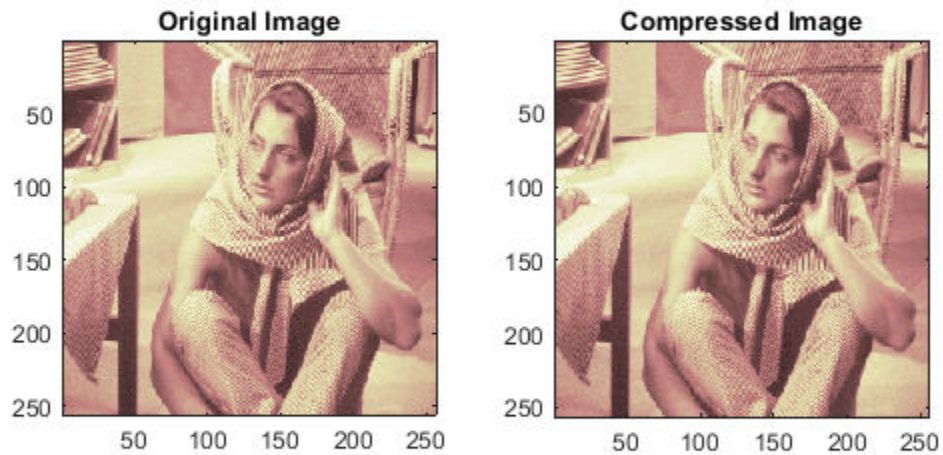
Note that, even though the compressed image is constructed from only about half as many nonzero wavelet coefficients as the original, there is almost no detectable deterioration in the image quality.

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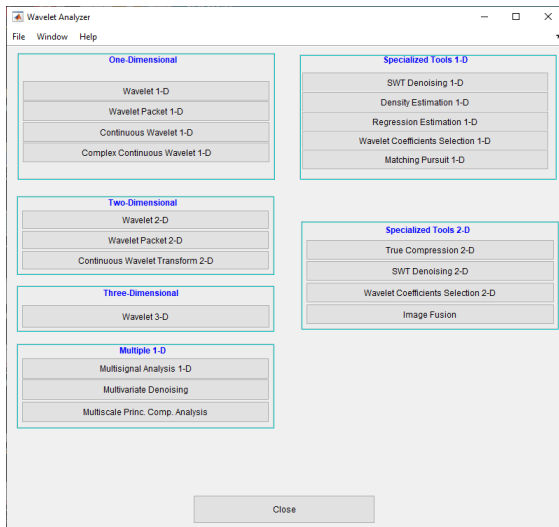
Click the **Wavelet 2-D** menu item. The discrete wavelet analysis tool for 2-D image data appears.
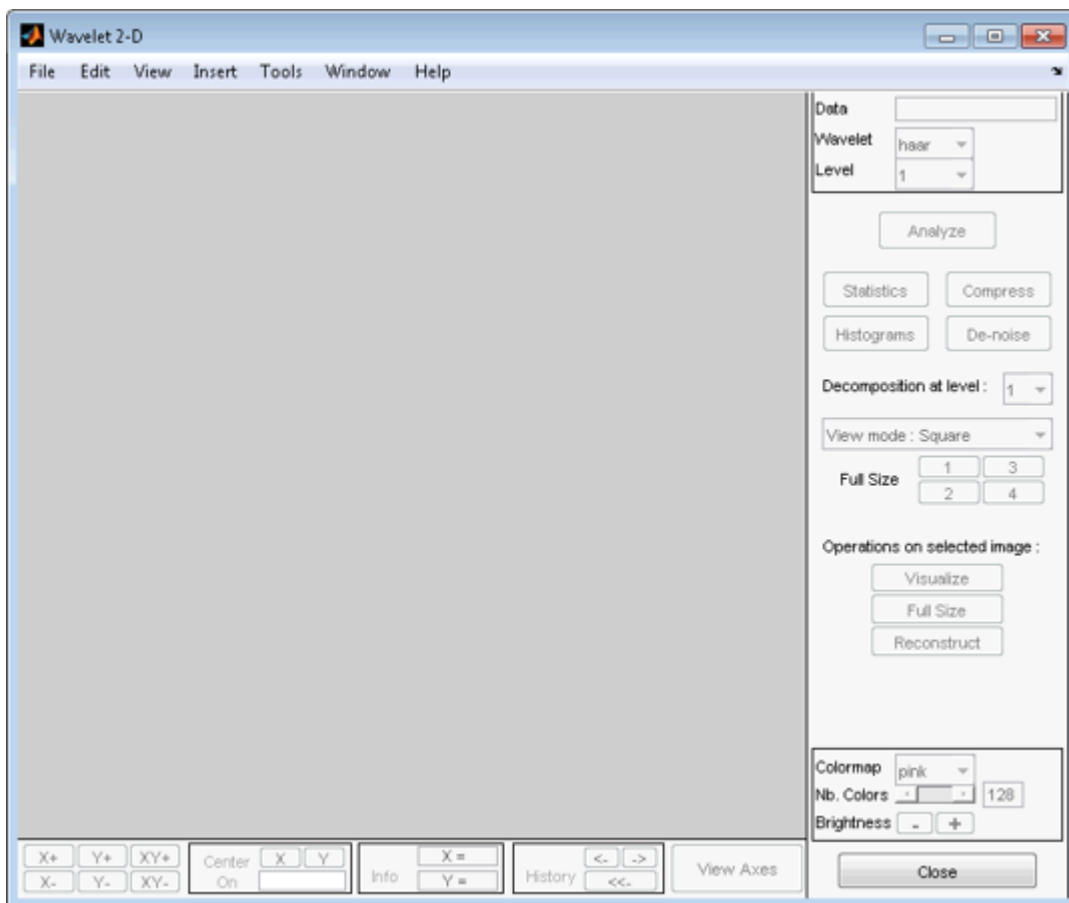
2   Load an image.

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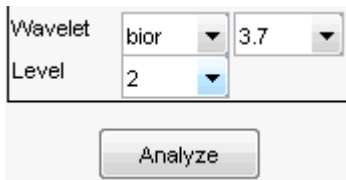
Click the **Analyze** button. After a pause for computation, the **Wavelet 2-D** tool displays its analysis.
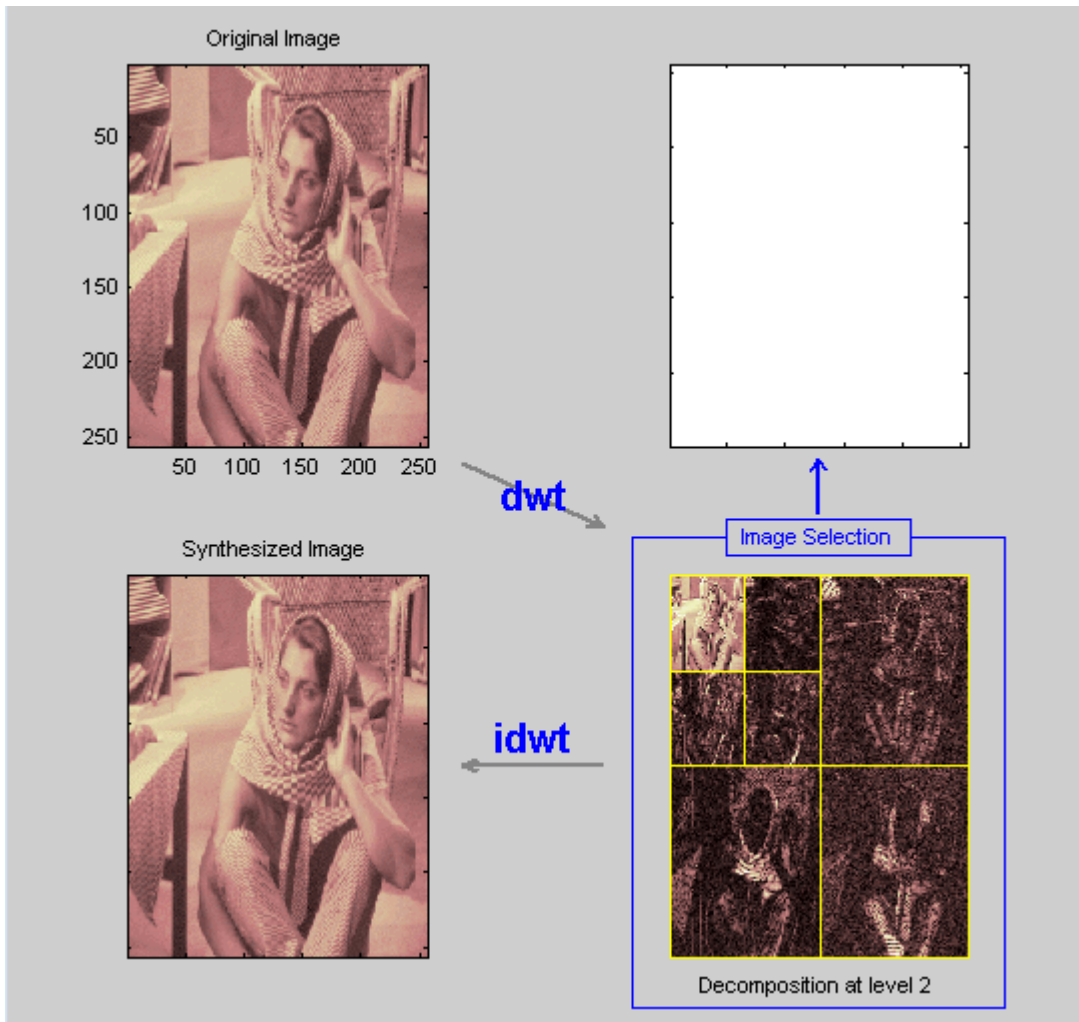
Using Square Mode Features

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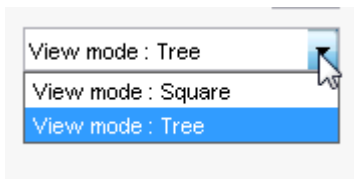
Your display changes to reveal the following.
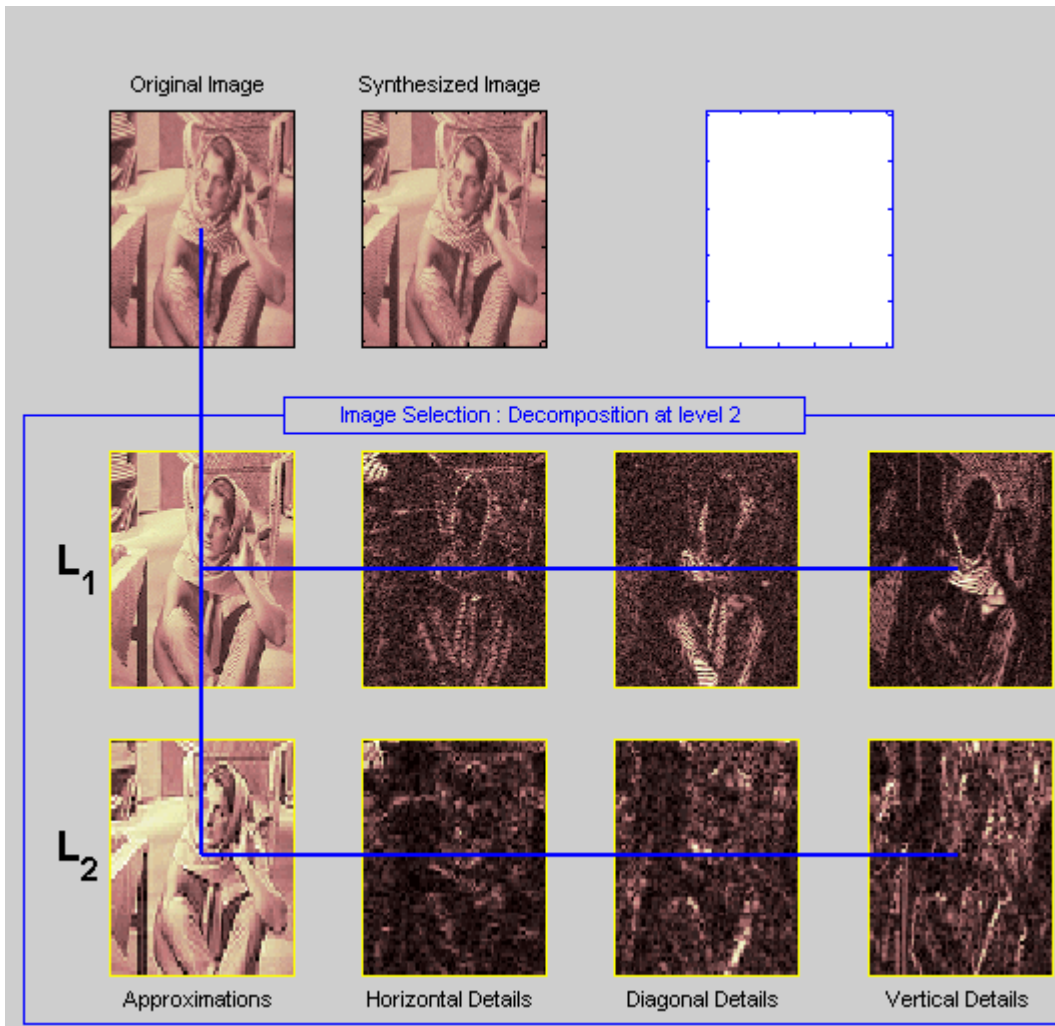
This is the same information shown in square mode, with in addition all the approximation coefficients, but arranged to emphasize the tree structure of the decomposition. The various buttons and menus work just the same as they do in square mode.

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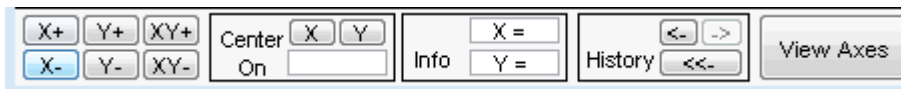
The **Wavelet 2-D** tool enlarges the displayed images.

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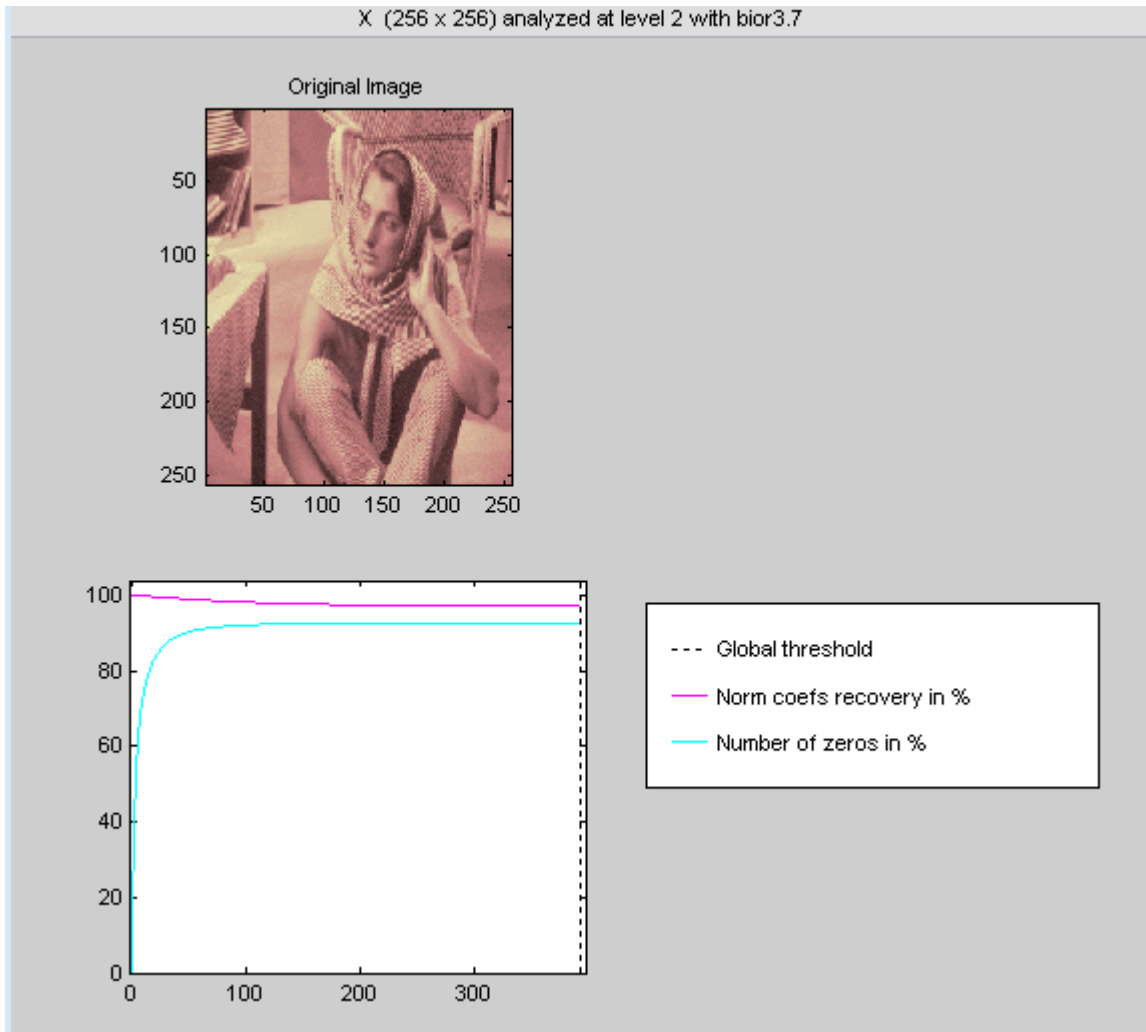
The tool automatically selects thresholding levels to provide a good initial balance between retaining the image's energy while minimizing the number of coefficients needed to represent the image.

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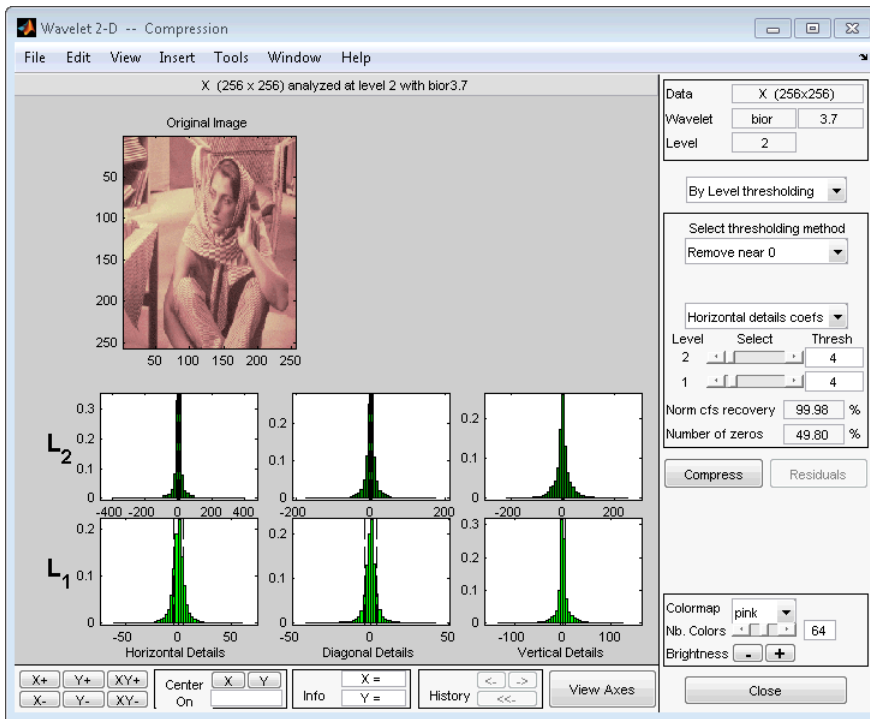
Select from the direction menu whether you want to adjust thresholds for horizontal, diagonal or vertical details. To make the actual adjustments for each level, use the sliders or use the left mouse button to directly drag the dashed lines.

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.
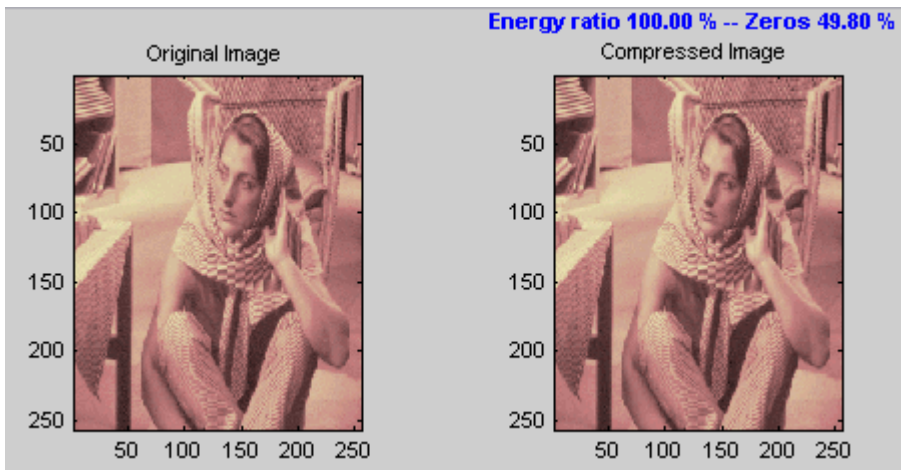
5 Show the residuals.

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.
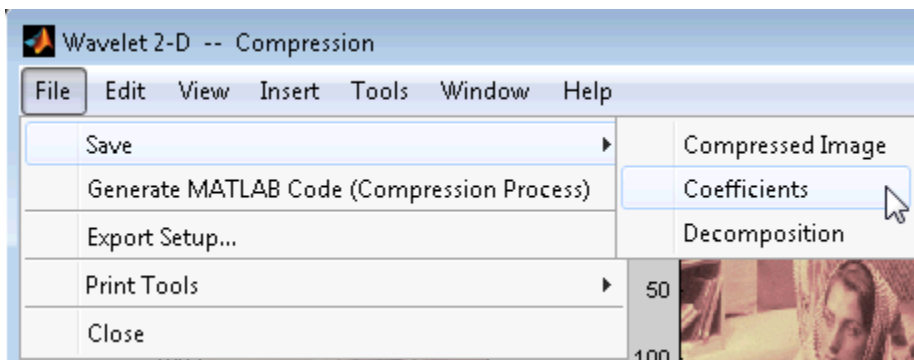
Saving Synthesized Images

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

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

Note Save options are also available when performing 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.
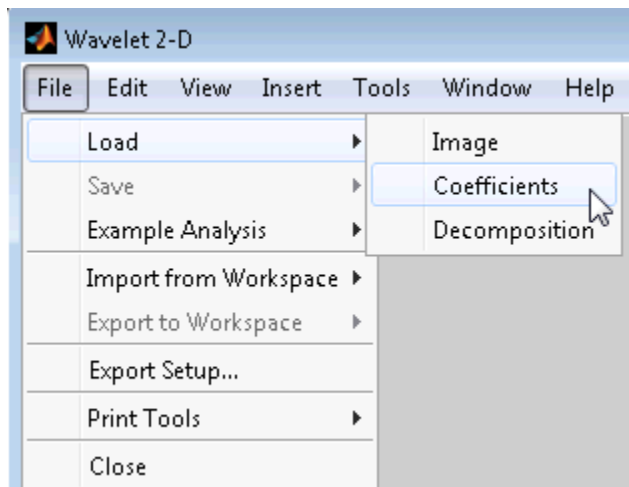
Loading Images

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.

Loading Discrete Wavelet Transform Coefficients

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

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

For an indexed image the matrix `sizes` is an n+2-by-2 array:
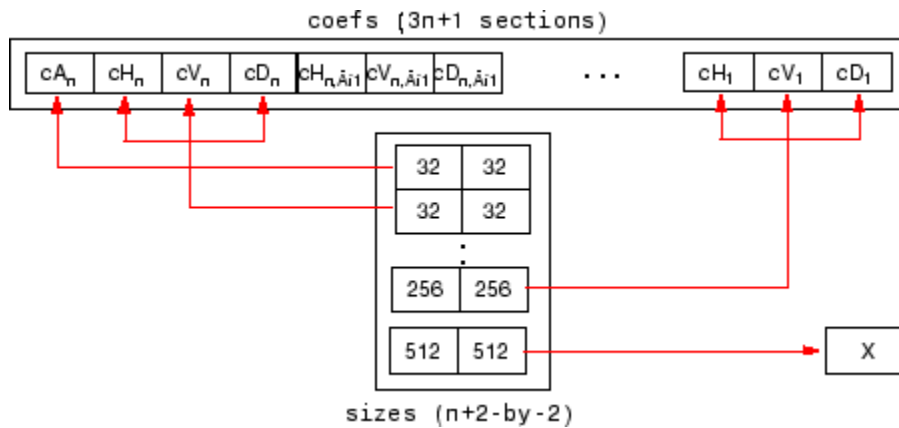
For a truecolor image, the matrix `sizes` is a n+2-by-3:
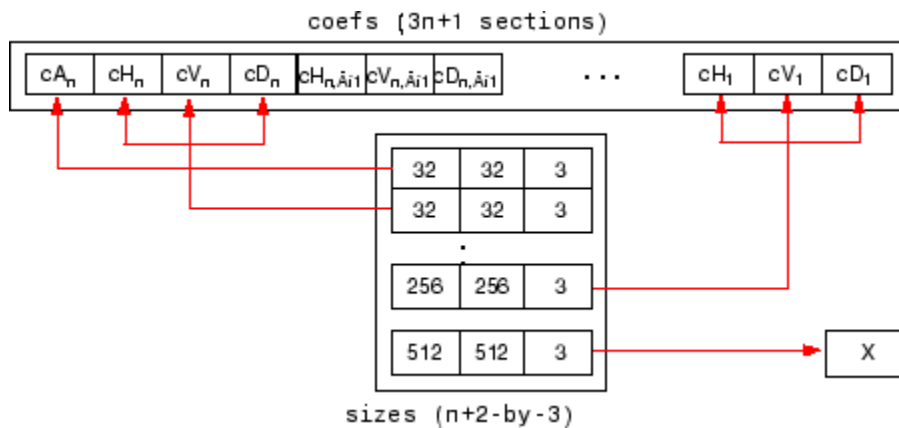
Variable `coefs` must be a vector of concatenated DWT coefficients. The `coefs` vector for an n-level decomposition contains 3n+1 sections, consisting of the level-n approximation coefficients, followed by the horizontal, vertical, and diagonal detail coefficients, in that order, for each level. Variable `sizes` is a matrix, the rows of which specify the size of cA_n, the size of cH_n (or cV_n, or cD_n),..., the size of cH_1 (or cV_1, or cD_1), and the size of the original image X. The sizes of vertical and diagonal details are the same as the horizontal detail.

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

```
save myfile coefs sizes
```

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

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

Loading Decompositions

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

The MAT-file contains these variables.

Variable	Status	Description
`coefs`	Required	Vector of concatenated DWT coefficients
`sizes`	Required	Matrix specifying sizes of components of `coefs` and of the original image
`wave_name`	Required	Character vector specifying name of wavelet used for decomposition (e.g., `db3`)
`map`	Optional	n-by-3 colormap matrix.
`data_name`	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 `size(X,1)` and `size(X,2)`. If your original image is not of correct size, you can use the function `wextend` to extend it.

2 Perform a single-level Stationary Wavelet Decomposition.

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

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

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

```
whos
```

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

3 Display the coefficients of approximation and details.

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

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

Approximation swa   Horiz. Detail swh

Vertical Detail swv   Diag. Detail swd
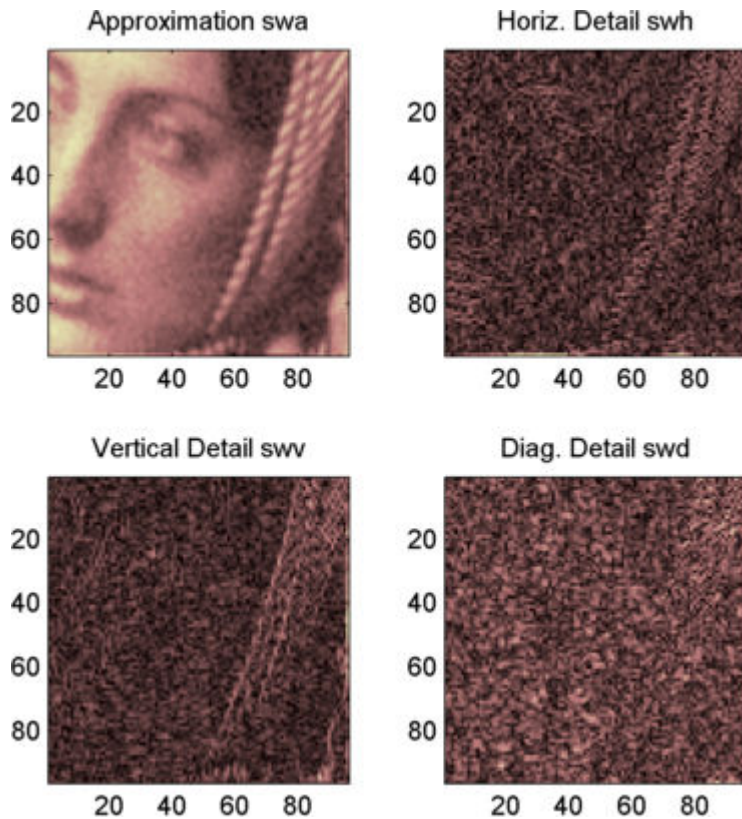
4 Regenerate the image by Inverse Stationary Wavelet Transform.

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')
```
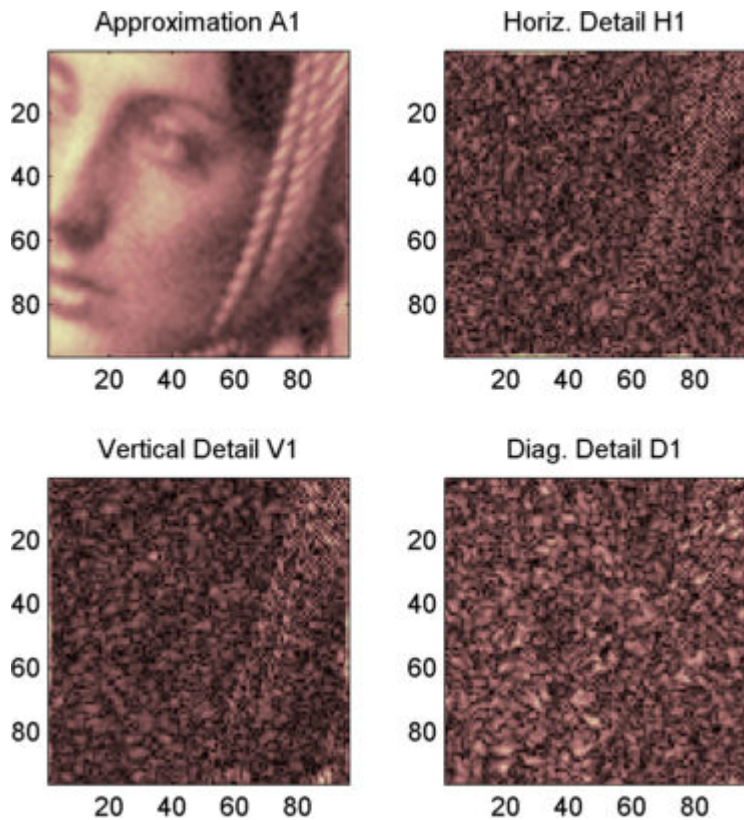

6   Perform a multilevel Stationary Wavelet Decomposition.

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

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

To display both the original and denoised images, type

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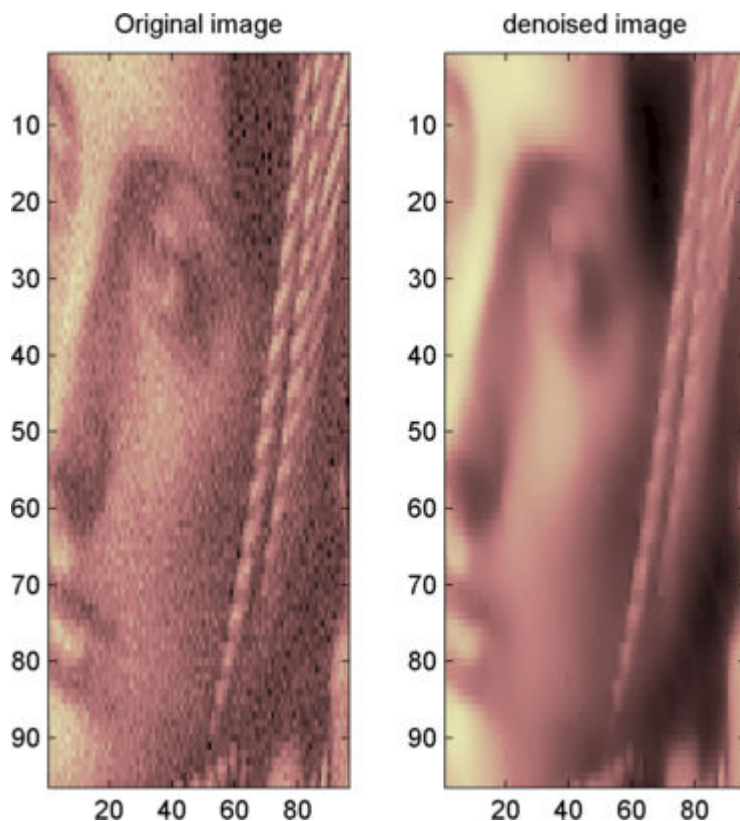

A second syntax can be used for the `swt2` and `iswt2` functions, giving the same results:

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

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

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:
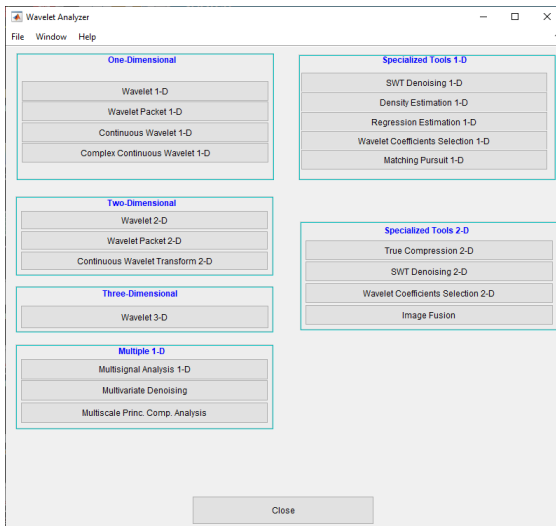
Click the **SWT Denoising 2-D** menu item.
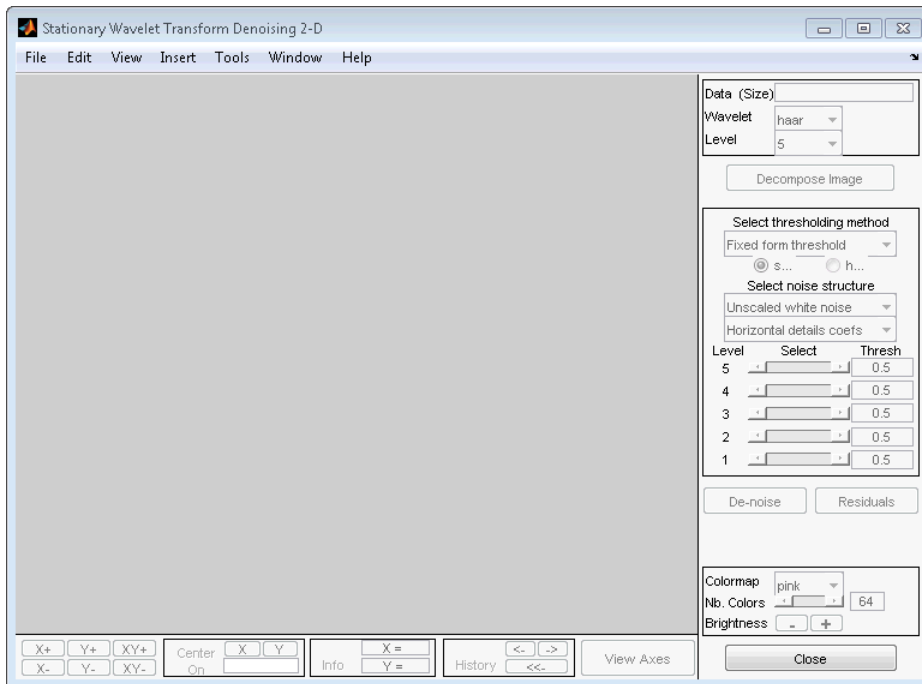
2 Load data.

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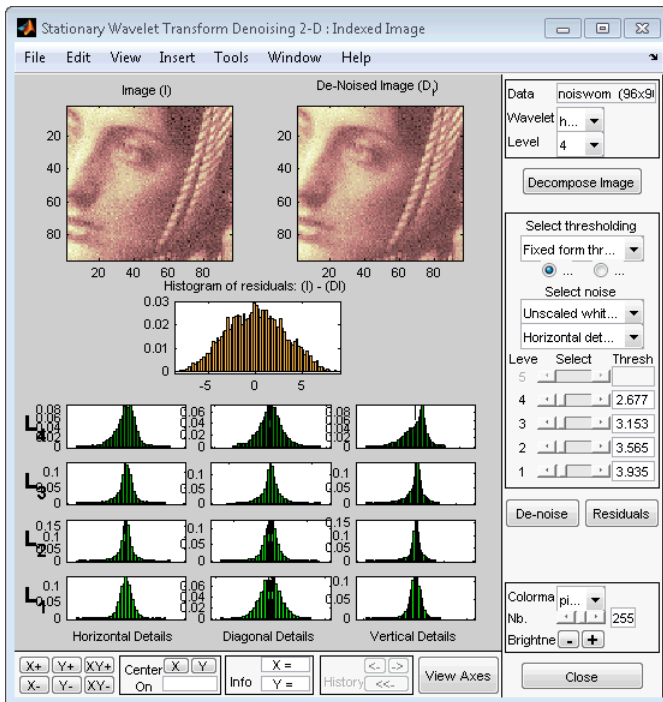
The result seems to be oversmoothed and the selected thresholds too aggressive. Nevertheless, the histogram of the residuals is quite good since it is close to a Gaussian distribution, which is the noise introduced to produce the analyzed image `noiswom.mat` from a piece of the original image `woman.mat`.

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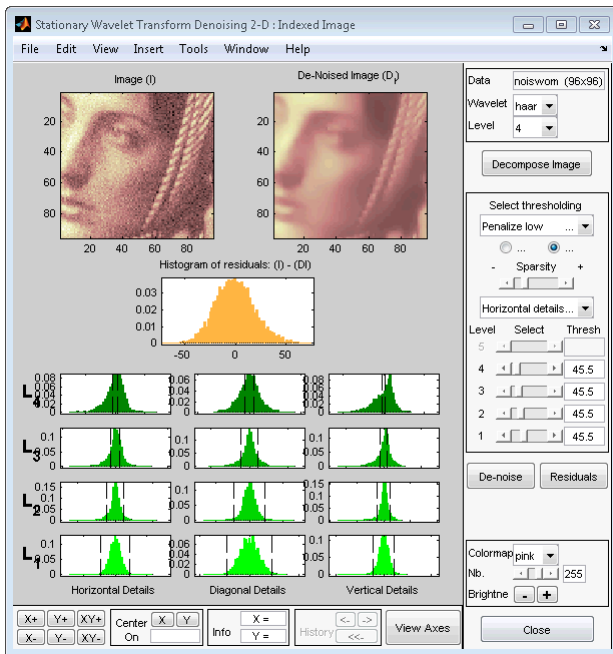
The result is quite satisfactory, although it is possible to improve it slightly.

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

$$SH\{\psi\} = \left\{ \psi_{a,s,t} = a^{3/4} \psi(S_s A_a(\,\cdot\, - t)) \right\}$$

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*:

$$SH(\psi) = \left\{ \psi_{j,k,m} = 2^{3/4j} \psi(S_k A_{2^j} \cdot\, - m) : j, k \in \mathbb{Z}, m \in \mathbb{Z}^2 \right\}.$$

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 `sheart2` to take the discrete shearlet transform of an image. For additional information, see the "References" on page 3-180.

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.
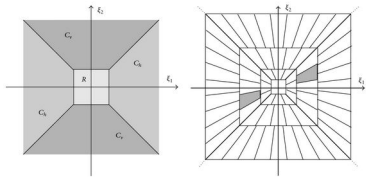
The spectra of the real-valued shearlets are the same over the positive and negative ξ_1, ξ_2 supports. Shearlets in complex-valued shearlet systems partition individually, not in pairs.

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`

More About

- "Boundary Effects in Real-Valued Bandlimited Shearlet Systems" on page 11-83

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
dwt3	Single-level decomposition
wavedec3	Decomposition

Synthesis-Reconstruction Functions

Function Name	Purpose
idwt3	Single-level reconstruction
waverec3	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.
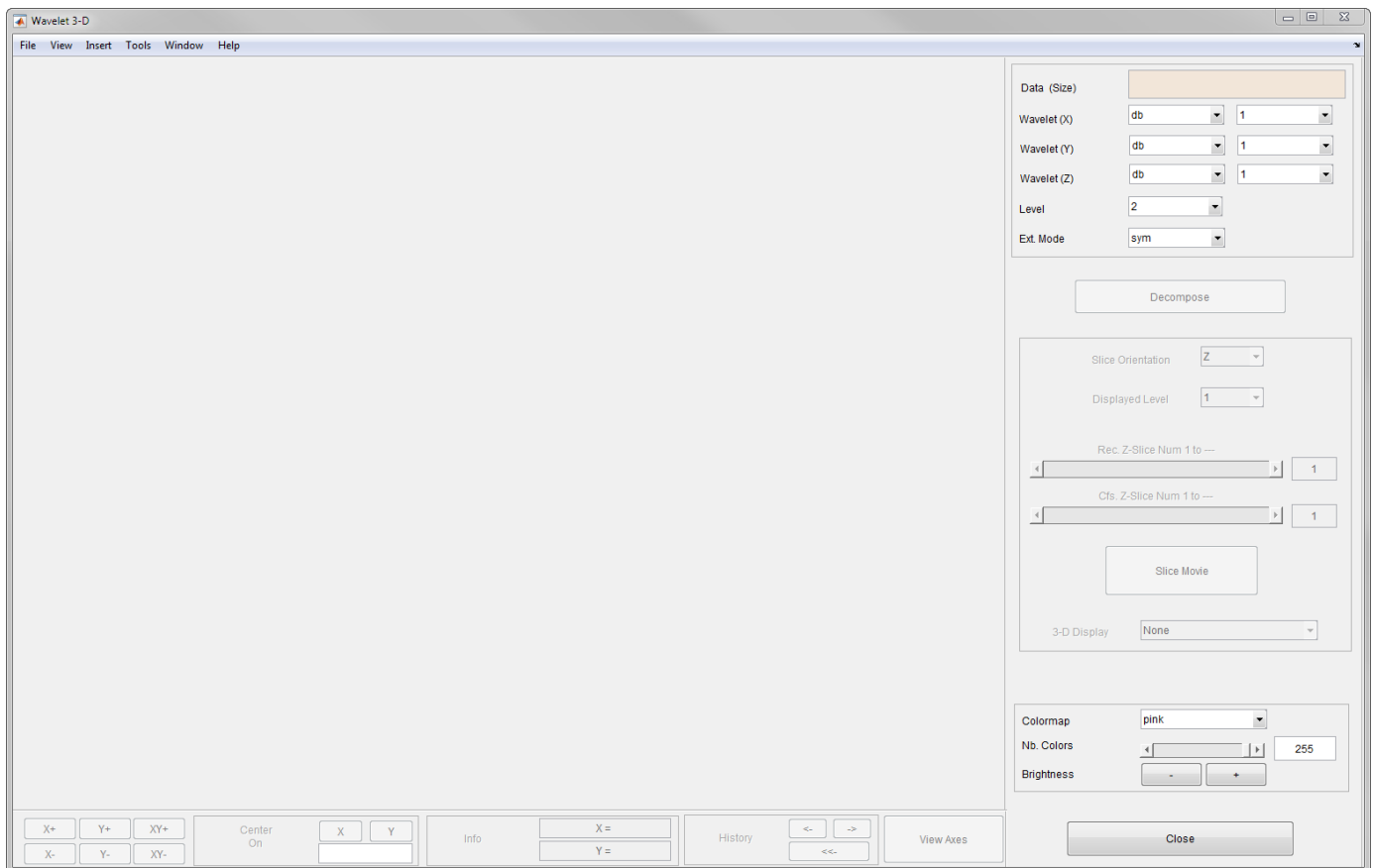
2 Load a 3-D array.

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.
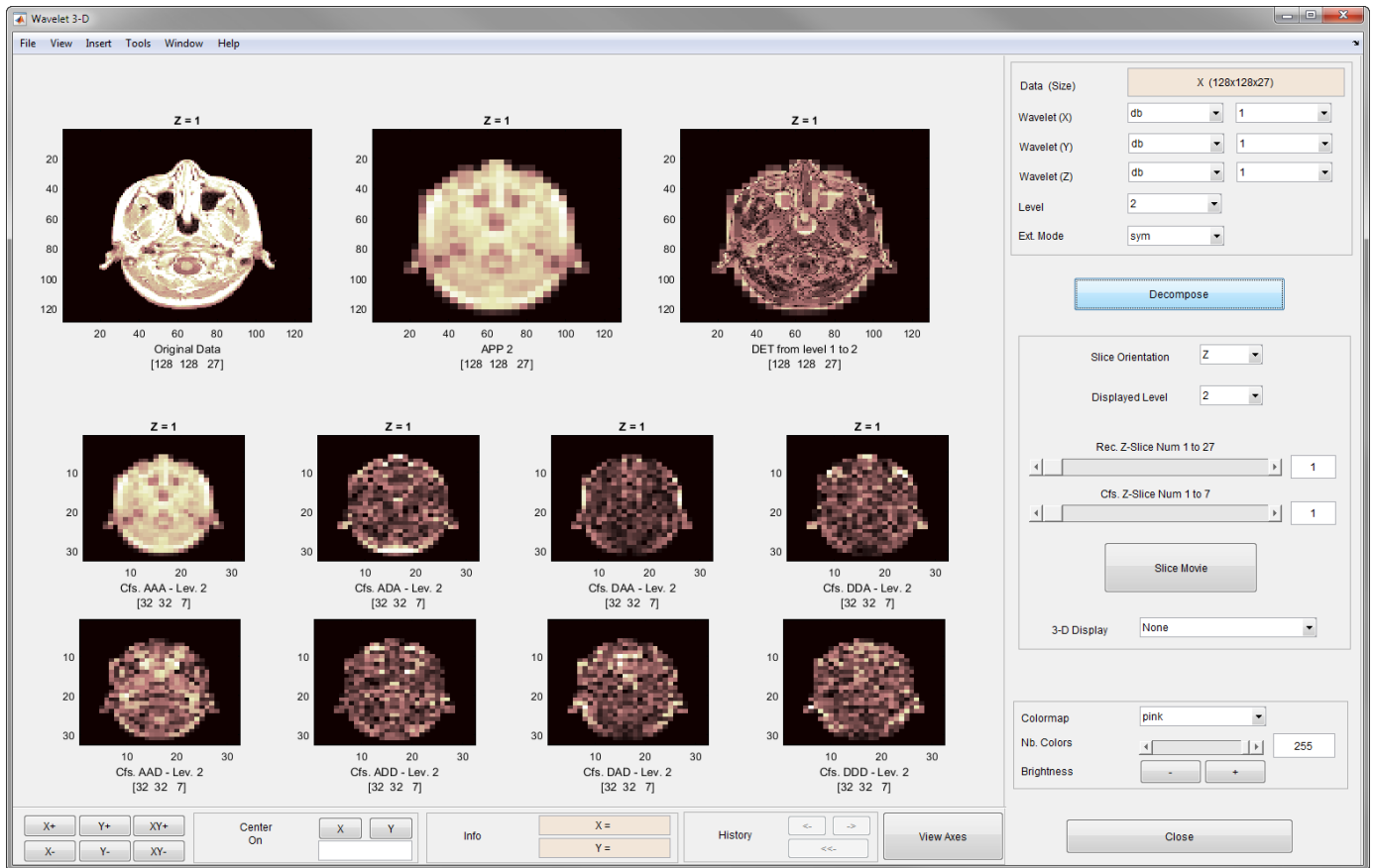
Review the slices of data and wavelet components in the graphical display. These slices are orthogonal to the *z*-direction as indicated by **Slice Orientation** in the command part of the window. This option lets you choose the desired slice orientation.

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.
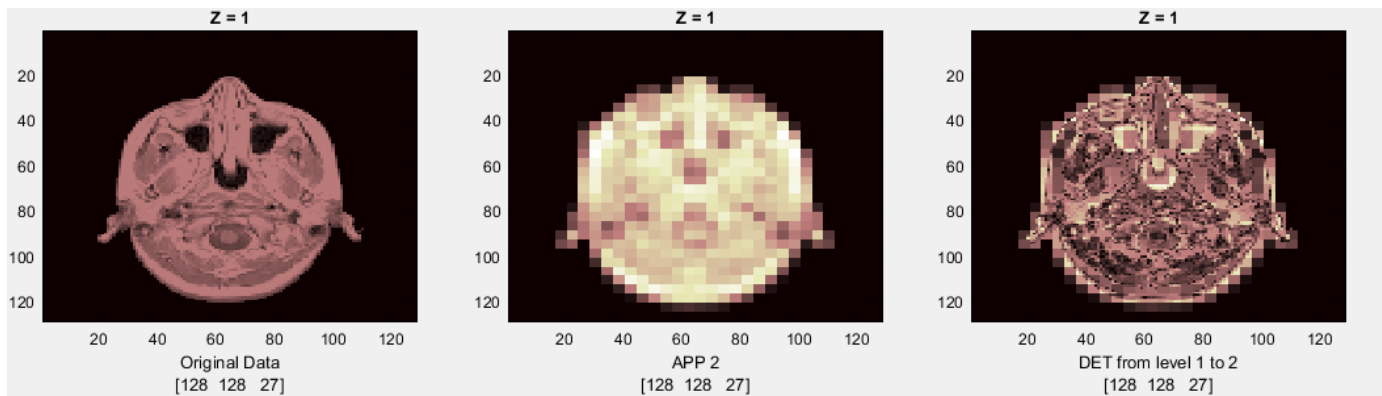
The next two lines of axes, display the wavelet coefficients at level 2, which is the desired level of the analysis. In the first line, the first graph contains the coefficients of approximation at level 2. The remaining seven axes display the seven types of wavelet coefficients at level 2. These coefficients contain the *x*-labels of the eight axes and display the name, type and size of the displayed data.
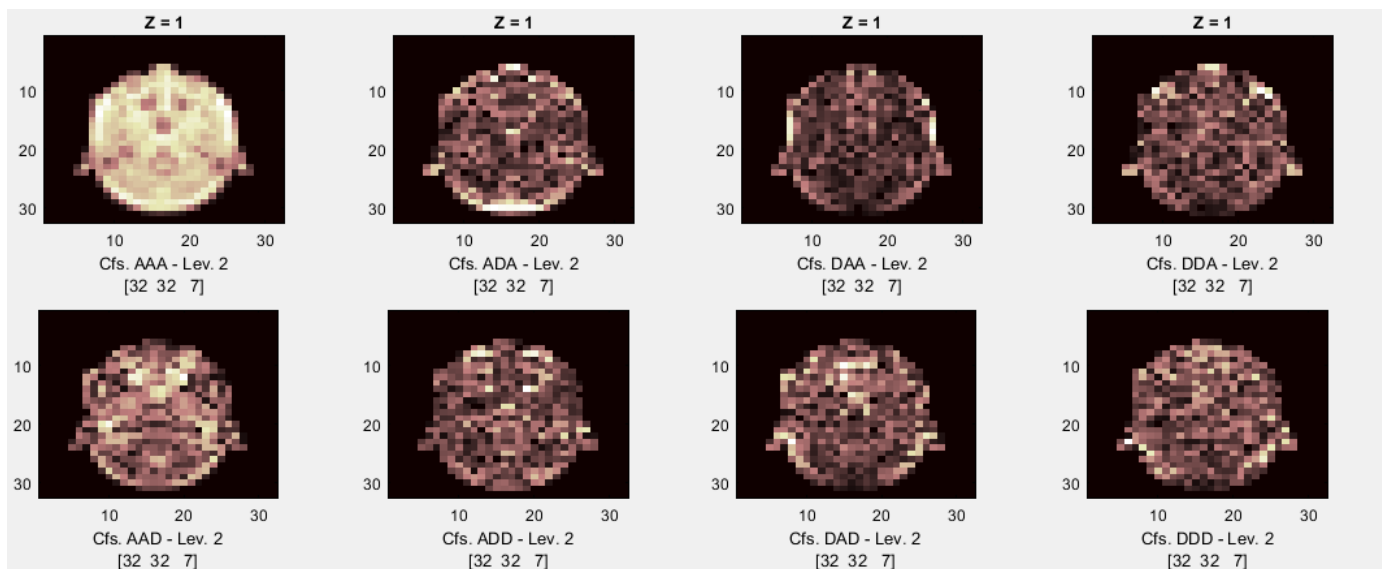
For example, in the third graphic of the bottom line, you can see the `Cfs-DAD` coefficients at level 2, which correspond to an array of size 32 x 32 x 7. The name of the `DAD` coefficients group indicates that it is obtained using

- 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.
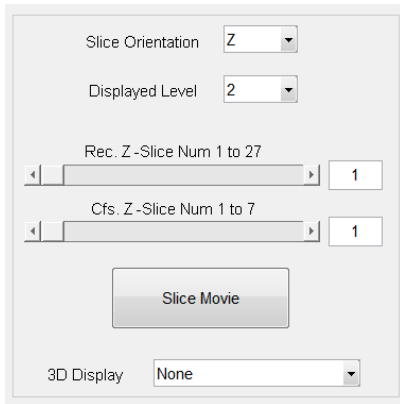
You can modify characteristics of the display using the options in the command part of the window. Each pair of sliders controls part of graphical array, the original and the reconstructed slices with the first pair or the coefficients slices with the second pair. Above each slider you can see the number 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**.
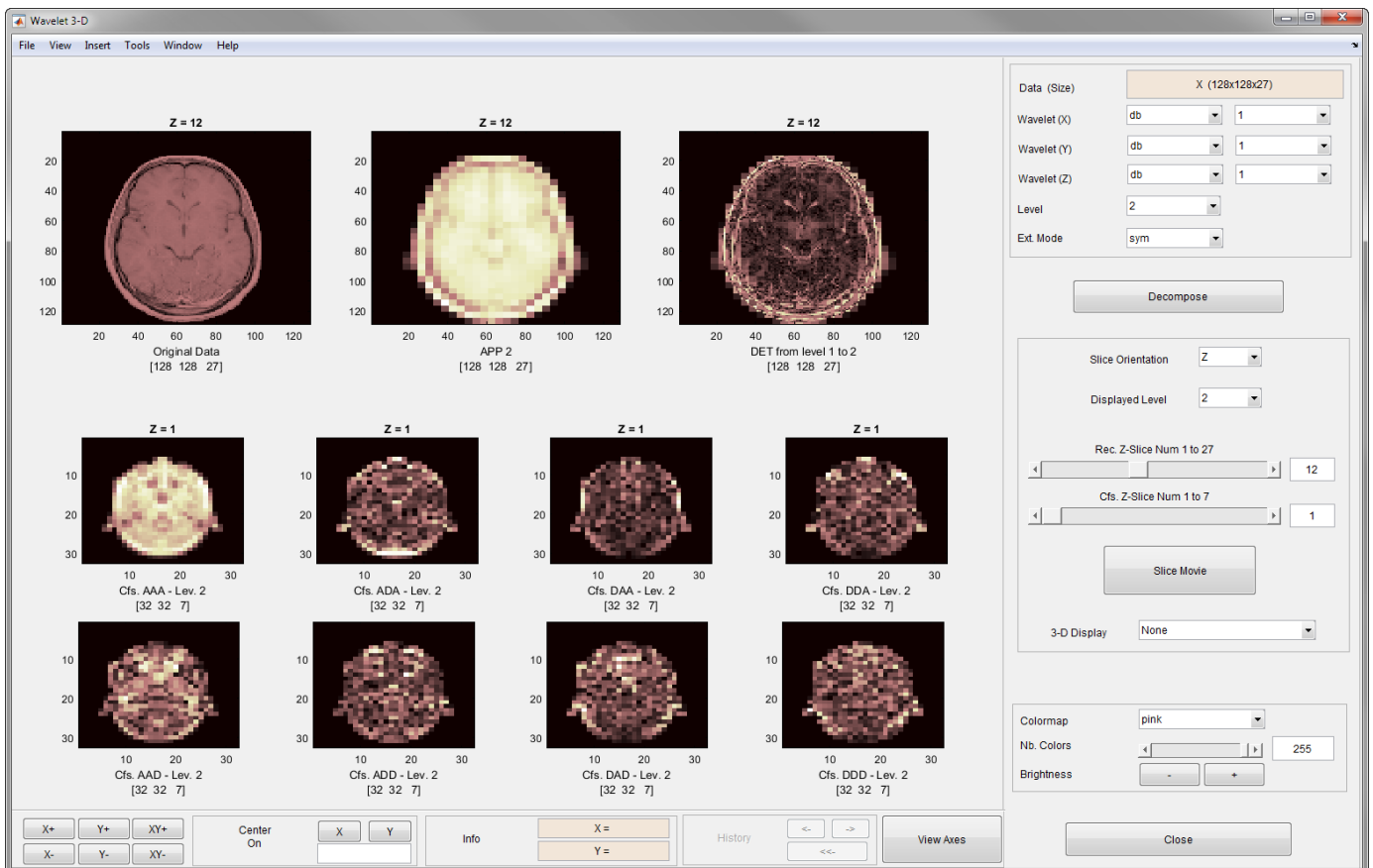
The **Slice Movie** button lets you see a movie of all the slices, first for the reconstructed slices and then for the coefficients slices. In this case, the movie contains 27 reconstructed images and 7 coefficients images.

3D Display lets you examine the original data and the wavelet components in true 3-D mode. Click **3D Display** and select APP1.
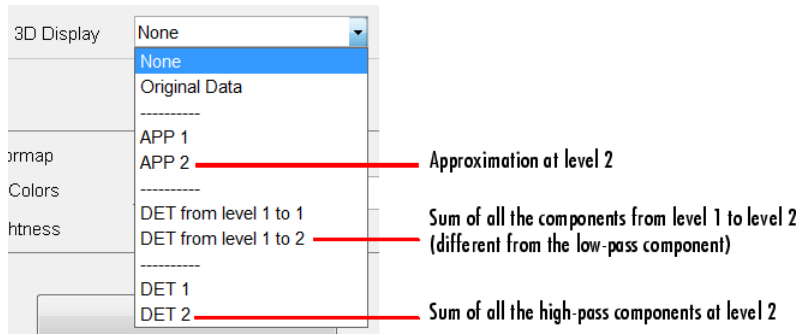
A rotated 3-D view of the approximation at level 1 opens in a new window. Use the sliders in the 3-D tool to examine the 3-D data.
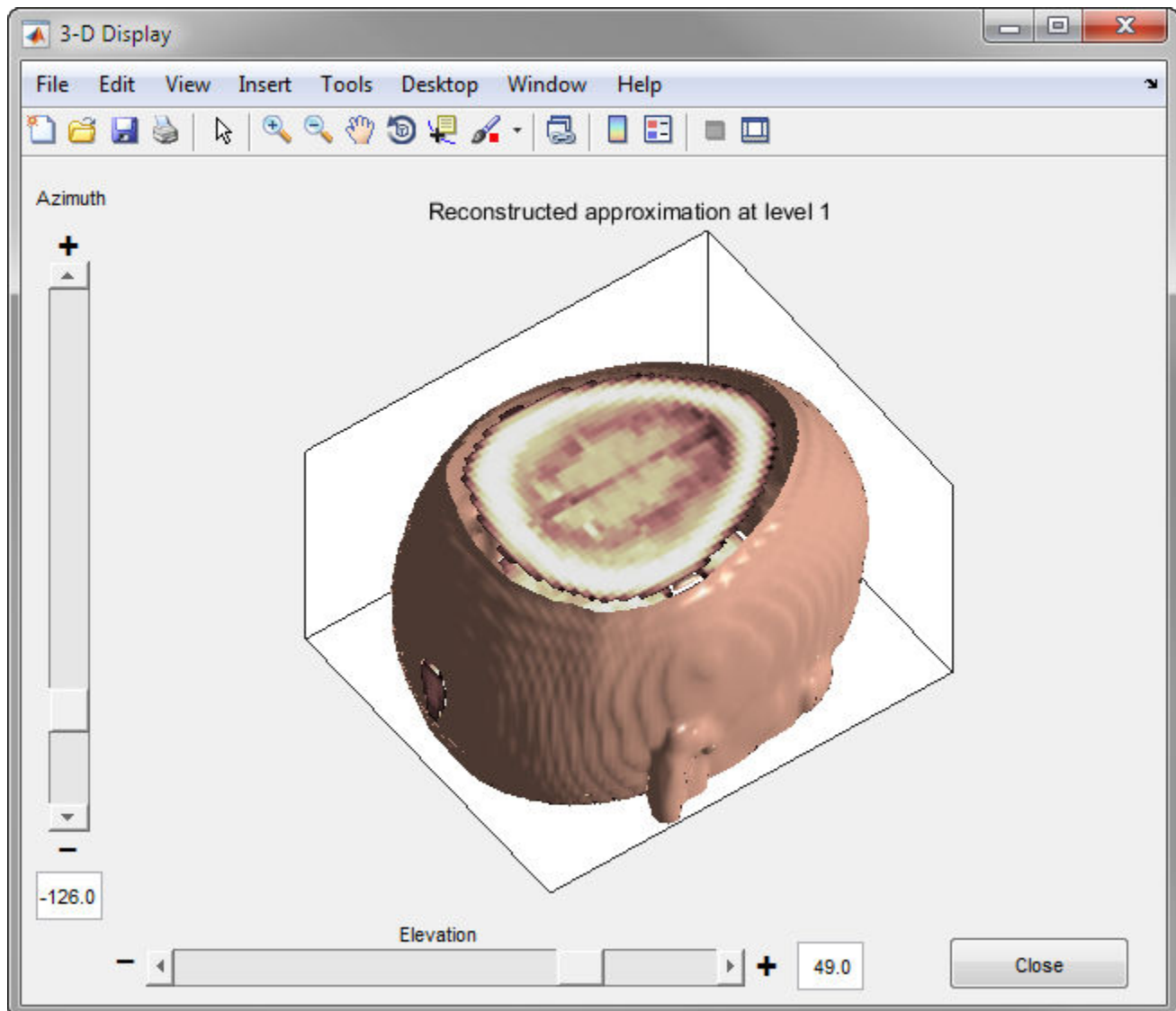
Importing and Exporting Information from the Wavelet Analyzer App

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.
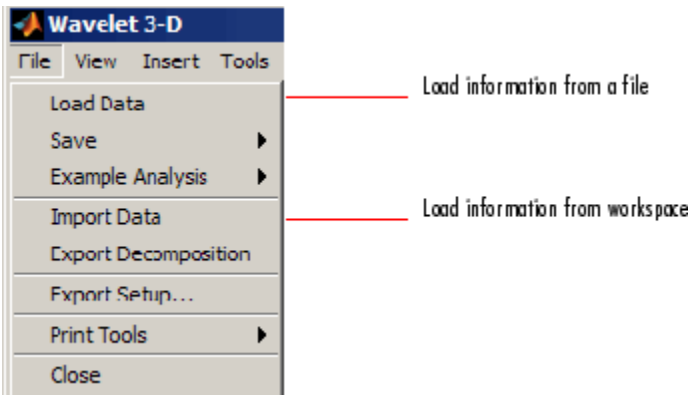
Similarly, you can load information from the workspace using **File > Import Data**. You then select the variable to load.

Saving Information to a File

You can save decompositions and approximations from the **Wavelet 3-D** tool to a file or to the workspace.
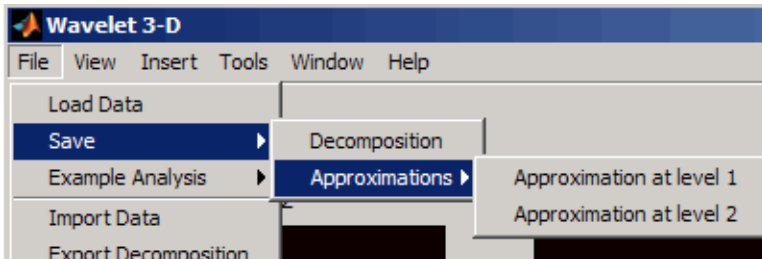
Saving Decompositions

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
    ```

3-189

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 Complex Wavelet Transforms

This example shows how the dual-tree complex wavelet transform (DTCWT) provides advantages over the critically sampled DWT for signal, image, and volume processing. The DTCWT 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 DTCWT 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 DTCWT is significantly less than the redundancy in the undecimated (stationary) DWT.

This example illustrates the approximate shift invariance of the DTCWT, the selective orientation of the dual-tree analyzing wavelets in 2-D and 3-D, and the use of the dual-tree complex discrete wavelet transform in image and volume denoising.

Near Shift Invariance of the DTCWT

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 DTCWT 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;
```

Set the DWT extension mode to periodic. Obtain the DWT and DTCWT of the two signals down to level 3 with wavelet and scaling filters of length 14. Extract the level-3 detail coefficients for comparison.

```
origmode = dwtmode('status','nodisplay');
dwtmode('per','nodisp')
J = 3;
[dwt1C,dwt1L] = wavedec(kronDelta1,J,'sym7');
[dwt2C,dwt2L] = wavedec(kronDelta2,J,'sym7');
dwt1Cfs = detcoef(dwt1C,dwt1L,3);
dwt2Cfs = detcoef(dwt2C,dwt2L,3);

[dt1A,dt1D] = dualtree(kronDelta1,'Level',J,'FilterLength',14);
[dt2A,dt2D] = dualtree(kronDelta2,'Level',J,'FilterLength',14);
dt1Cfs = dt1D{3};
dt2Cfs = dt2D{3};
```

Plot the absolute values of the DWT and DTCWT coefficients for the two signals at level 3 and compute the energy (squared l^2 norms) of the coefficients. Plot the coefficients on the same scale.

The four sample shift in the signal has caused a significant change in the energy of the level-3 DWT coefficients. The energy in the level-3 DTCWT coefficients has changed by only approximately 3%.

```
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)
ylim([0 0.4])
subplot(1,2,2)
stem(abs(dwt2Cfs),'markerfacecolor',[0 0 1])
title({'DWT';['Squared 2-norm = ' num2str(norm(dwt2Cfs,2)^2,3)]},...
    'fontsize',10)
ylim([0 0.4])
```
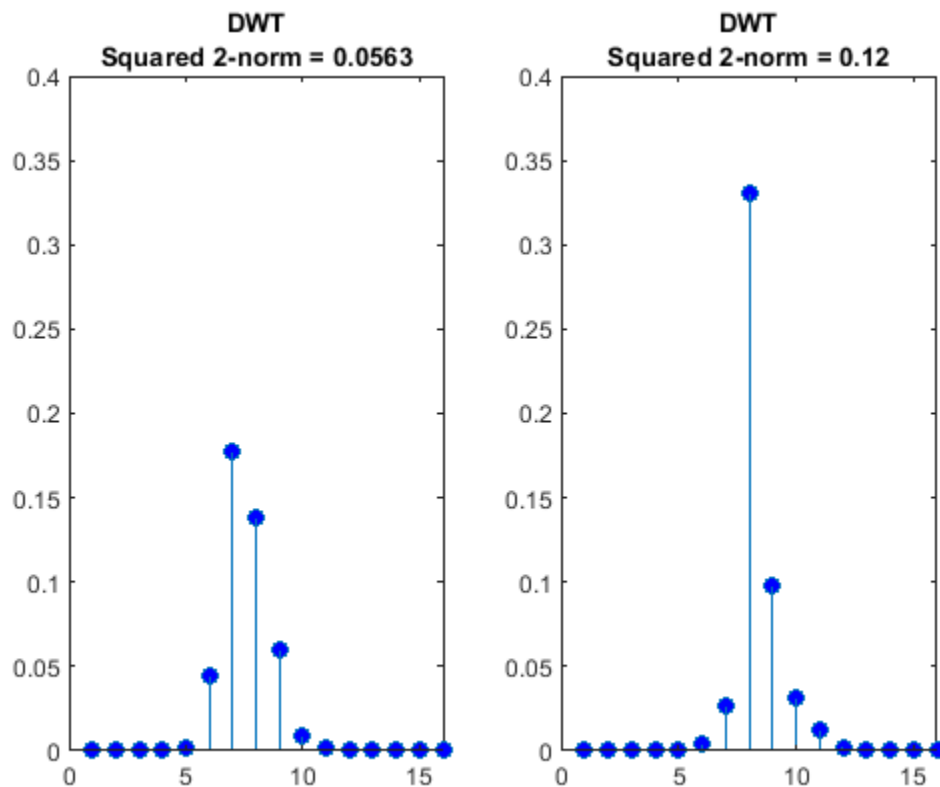


```
figure
subplot(1,2,1)
stem(abs(dt1Cfs),'markerfacecolor',[0 0 1])
title({'Dual-tree CWT';['Squared 2-norm = ' num2str(norm(dt1Cfs,2)^2,3)]},...
    'fontsize',10)
ylim([0 0.4])
subplot(1,2,2)
stem(abs(dwt2Cfs),'markerfacecolor',[0 0 1])
title({'Dual-tree CWT';['Squared 2-norm = ' num2str(norm(dt2Cfs,2)^2,3)]},...
    'fontsize',10)
ylim([0 0.4])
```
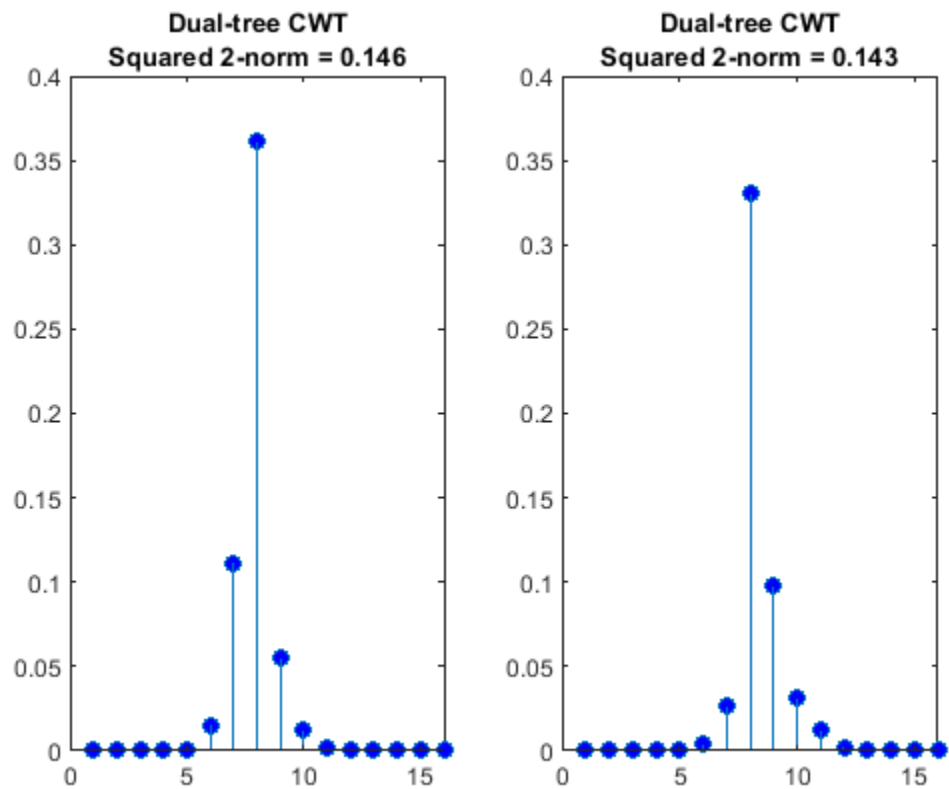
To demonstrate the utility of approximate shift invariance in real data, we analyze an electrocardiogram (ECG) signal. The sampling interval for the ECG signal is 1/180 seconds. The data are taken from Percival & Walden [3], p.125 (data originally provided by William Constantine and Per Reinhall, University 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')
```
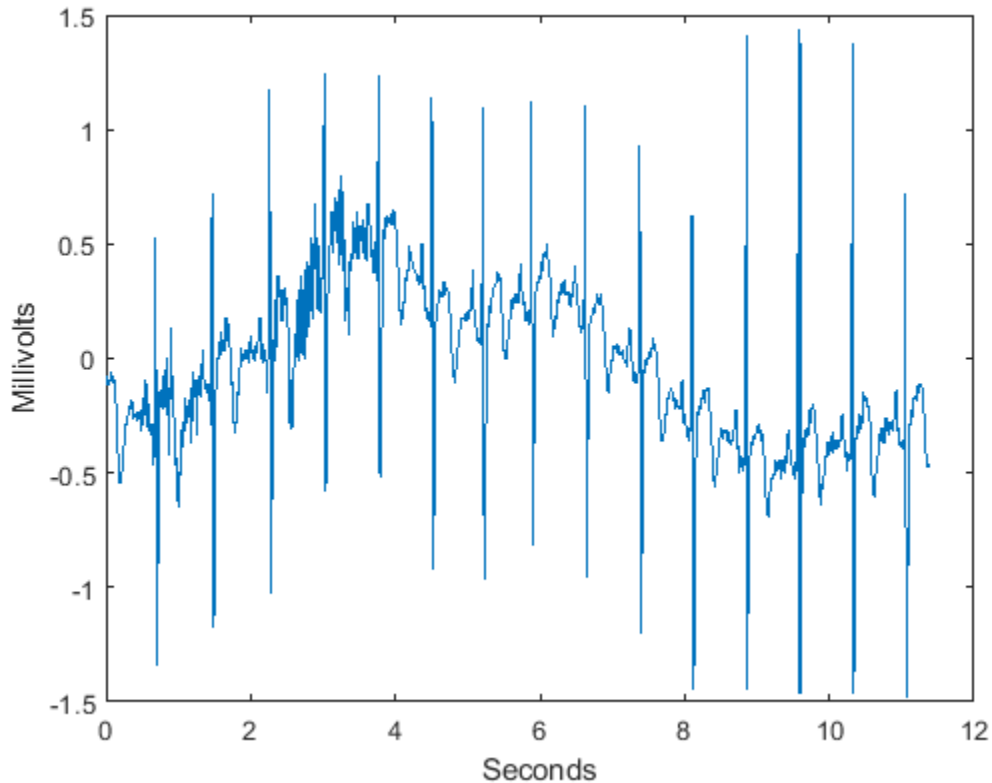
The large positive peaks approximately 0.7 seconds apart are the R waves of the cardiac rhythm. First, decompose the signal using the critically sampled DWT with the Farras nearly symmetric filters. Plot the original signal along with the level-2 and level-3 wavelet coefficients. The level-2 and level-3 coefficients were chosen because the R waves are isolated most prominently in those scales for the given sampling rate.

```
figure
J = 6;
[df,rf] = dtfilters('farras');
[dtDWT1,L1] = wavedec(wecg,J,df(:,1),df(:,2));
details = zeros(2048,3);
details(2:4:end,2) = detcoef(dtDWT1,L1,2);
details(4:8:end,3) = detcoef(dtDWT1,L1,3);
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')
```
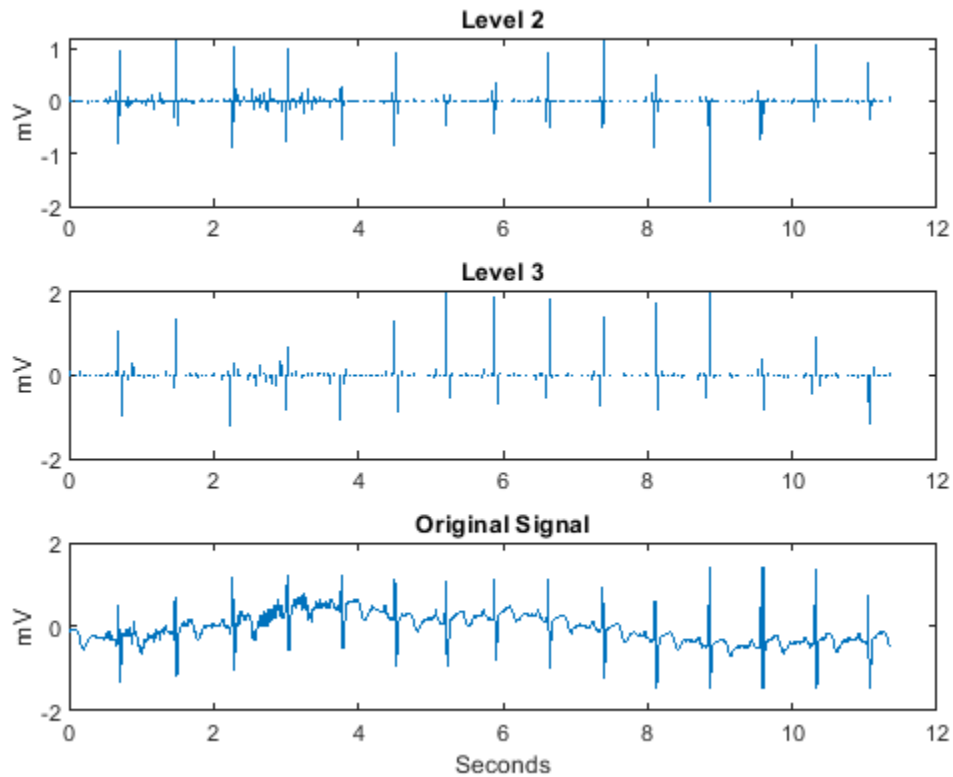
Repeat the above analysis for the dual-tree transform. In this case, just plot the real part of the dual-tree coefficients at levels 2 and 3.

```
[dtcplxA,dtcplxD] = dualtree(wecg,'Level',J,'FilterLength',14);
details = zeros(2048,3);
details(2:4:end,2) = dtcplxD{2};
details(4:8:end,3) = dtcplxD{3};
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')
```
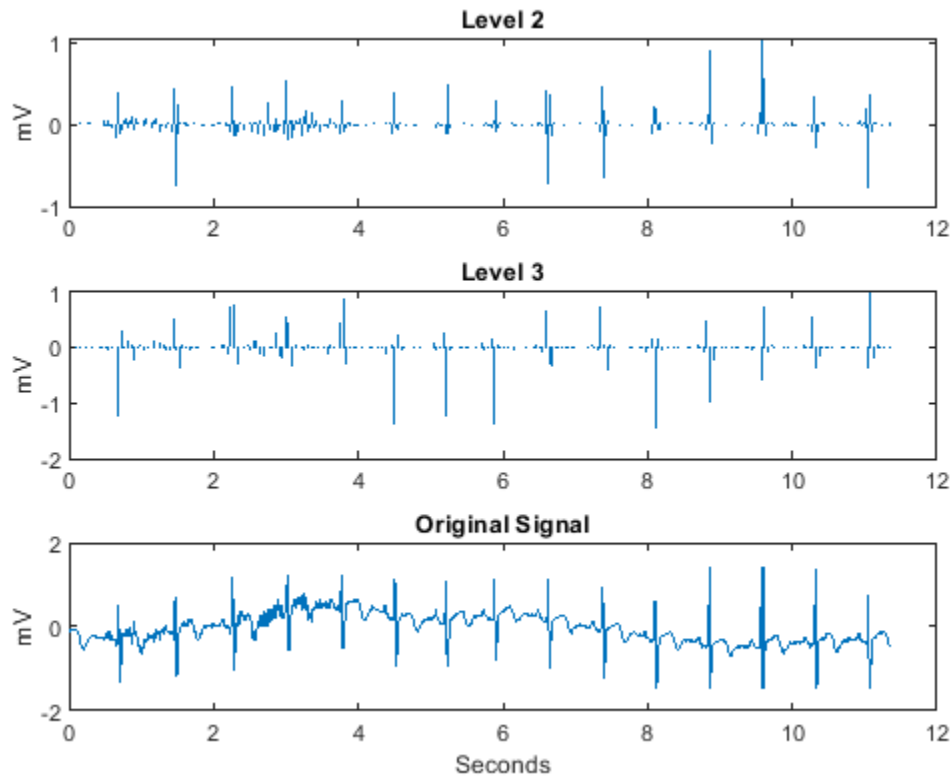
Both the critically sampled and dual-tree wavelet transforms localize an important feature of the ECG waveform to similar scales

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,L2] = wavedec(wecgShift,J,df(:,1),df(:,2));

detCfs1 = detcoef(dtDWT1,L1,1:J,'cells');
apxCfs1 = appcoef(dtDWT1,L1,rf(:,1),rf(:,2),J);
cfs1 = horzcat(detCfs1,{apxCfs1});
detCfs2 = detcoef(dtDWT2,L2,1:J,'cells');
apxCfs2 = appcoef(dtDWT2,L2,rf(:,1),rf(:,2),J);
cfs2 = horzcat(detCfs2,{apxCfs2});

sigenrgy = norm(wecg,2)^2;
enr1 = cell2mat(cellfun(@(x)(norm(x,2)^2/sigenrgy)*100,cfs1,'uni',0));
enr2 = cell2mat(cellfun(@(x)(norm(x,2)^2/sigenrgy)*100,cfs2,'uni',0));
levels = {'D1';'D2';'D3';'D4';'D5';'D6';'A6'};
enr1 = enr1(:);
enr2 = enr2(:);
table(levels,enr1,enr2,'VariableNames',{'Level','enr1','enr2'})
```

```
ans=7×3 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 discrete wavelet transform.

```
[dtcplx2A,dtcplx2D] = dualtree(wecgShift,'Level',J,'FilterLength',14);
dtCfs1 = vertcat(dtcplxD,{dtcplxA});
dtCfs2 = vertcat(dtcplx2D,{dtcplx2A});

dtenr1 = cell2mat(cellfun(@(x)(norm(x,2)^2/sigenrgy)*100,dtCfs1,'uni',0));
dtenr2 = cell2mat(cellfun(@(x)(norm(x,2)^2/sigenrgy)*100,dtCfs2,'uni',0));
dtenr1 = dtenr1(:);
dtenr2 = dtenr2(:);
table(levels,dtenr1,dtenr2, 'VariableNames',{'Level','dtenr1','dtenr2'})
```

```
ans=7×3 table
    Level      dtenr1     dtenr2

    _____     _____     _____

    {'D1'}     5.3533     5.3619
    {'D2'}     6.2672     6.2763
    {'D3'}     12.155      12.19
    {'D4'}     8.2831      8.325
    {'D5'}       5.81     5.8577
    {'D6'}     3.1768     3.0526
    {'A6'}     58.403     58.384
```

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. Use the helper function `helperPlotCritSampDWT` to plot the LH, HL, and HH wavelets for Daubechies' least-asymmetric phase wavelet with 4 vanishing moments, `sym4`.

```
 helperPlotCritSampDWT
```

Critically Sampled DWT
2-D separable wavelets (sym4) -- LH, HL, HH
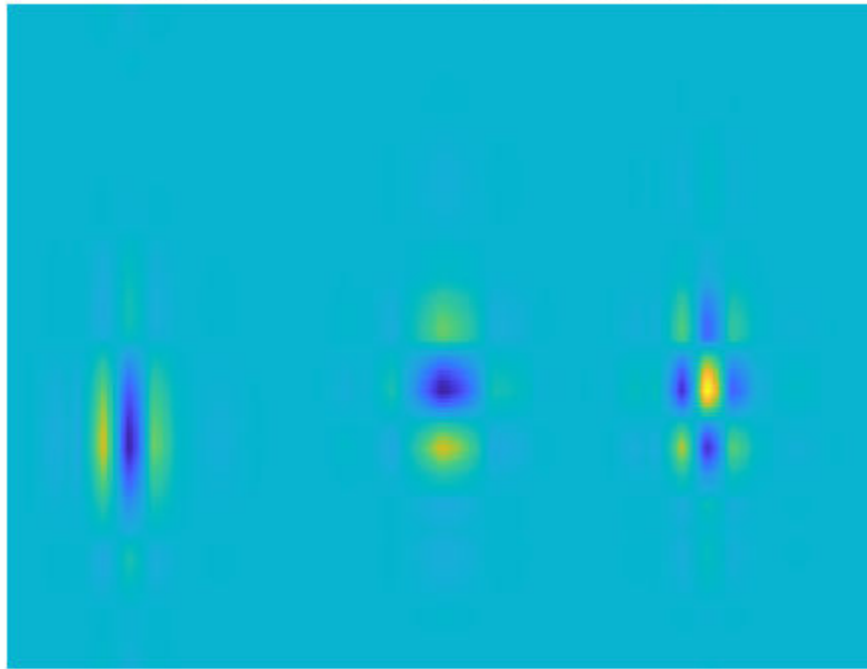
Note that the LH and HL wavelets have clear horizontal and vertical orientations respectively. However, the HH wavelet on the far right mixes both the +45 and -45 degree directions, producing a checkerboard artifact. This mixing of orientations is due to the use of real-valued separable filters. The 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, $\{h_0, g_1\}$, $\{g_0, h_1\}$. Use the helper function `helperPlotWaveletDTCWT` to plot the orientation of the 12 wavelets corresponding to the real and imaginary parts of the DTCWT. The top row of the preceding figure shows the real parts of the six wavelets, and the second row shows the imaginary parts.

`helperPlotWaveletDTCWT`

DTCWT 2-D Wavelets
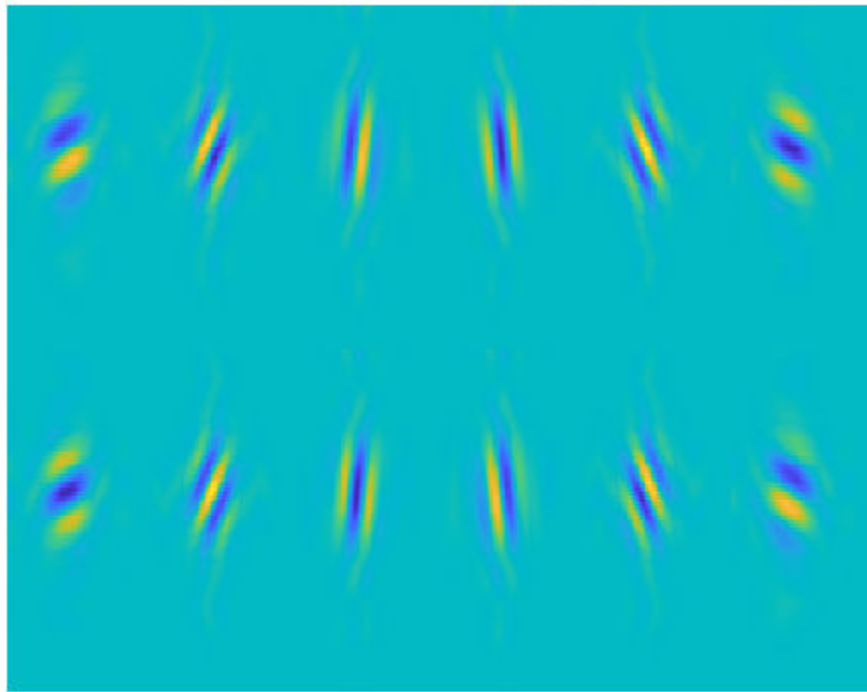
Edge Representation in Two Dimensions

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
imagesc(woctagon)
colormap gray
title('Original Image')
axis equal
axis off
```

Original Image

Use the helper function `helprPlotOctagon` to decompose the image down to level 4 and reconstruct an image approximation based on the level-4 detail coefficients.

```
helperPlotOctagon(woctagon)
```

DTCWT

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
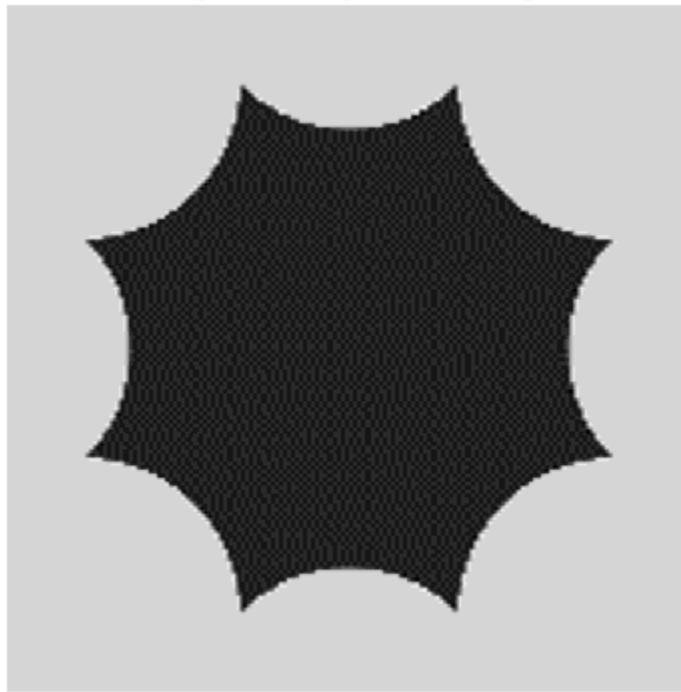
Again, use the helper function `helprPlotOctagon` to decompose the image down to level 4 and reconstruct an image approximation based on the level-4 detail coefficients for both the standard 2-D DWT and the complex oriented dual-tree DWT.

```
helperPlotOctagon(woctagonHyperbolic)
```
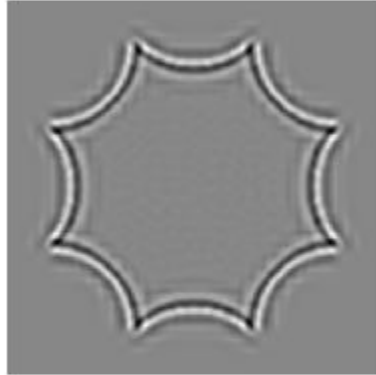
DTCWT   **DWT**

Note that the ringing artifacts evident in the 2-D critically sampled DWT are absent in the 2-D DTCWT of both images. The DTCWT more faithfully reproduces line and curve singularities.

Image Denoising

Because of the ability to isolate distinct orientations in separate subbands, the DTCWT is often able to outperform the standard separable DWT in applications like image denoising. To demonstrate this, use the helper function `helperCompare2DDenoisingDTCWT`. 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, and the DTCWT. For each threshold value, the root-mean-square (RMS) error and peak signal-to-noise ratio (PSNR) are displayed.

```
numex = 3;
helperCompare2DDenoisingDTCWT(numex,0:2:100,'PlotMetrics');
```
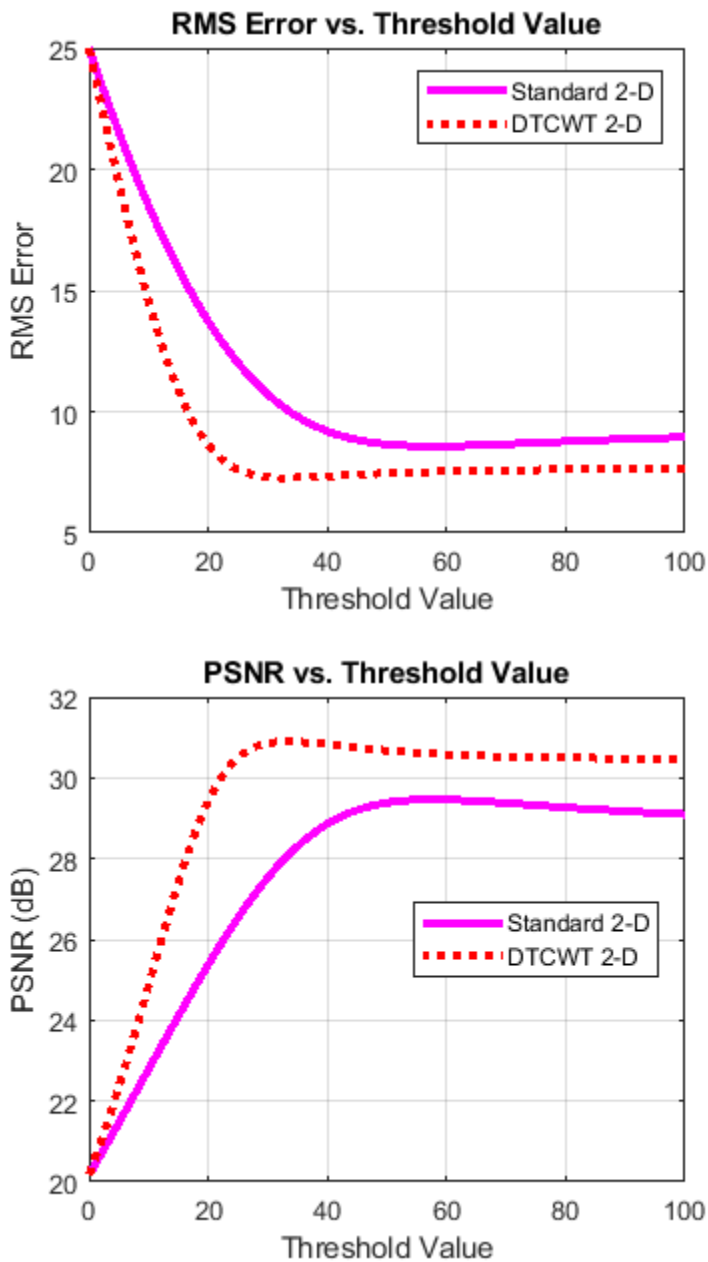
The DTCWT outperforms the standard DWT in RMS error and PSNR.

Next, obtain the denoised images for a threshold value of 25, which is equal to the standard deviation of the additive noise.

```
numex = 3;
helperCompare2DDenoisingDTCWT(numex,25,'PlotImage');
```

Original Image

Noisy Image
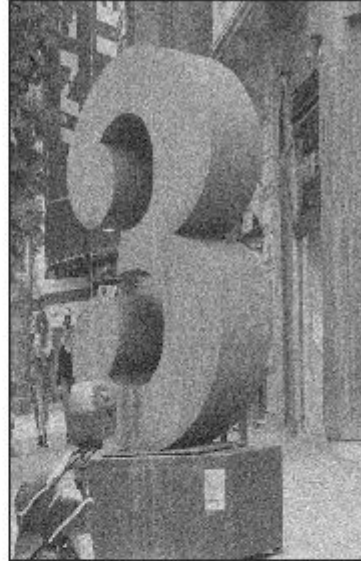
**Denoised Image
Standard 2-D**

**Denoised Image
2-D DTCWT**
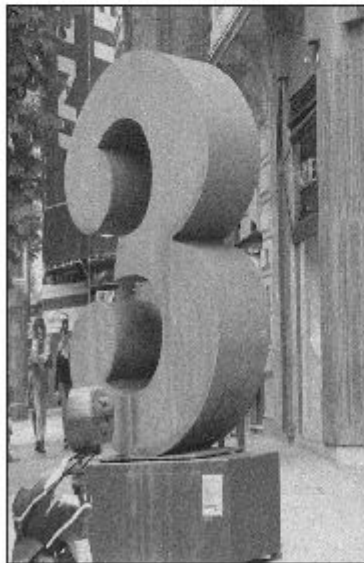
THR: 25.00

RMSE: 12.01
PSNR: 26.54

THR: 25.00

RMSE: 7.61
PSNR: 30.51

With a threshold value equal to the standard deviation of the additive noise, the DTCWT 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 DTCWT 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
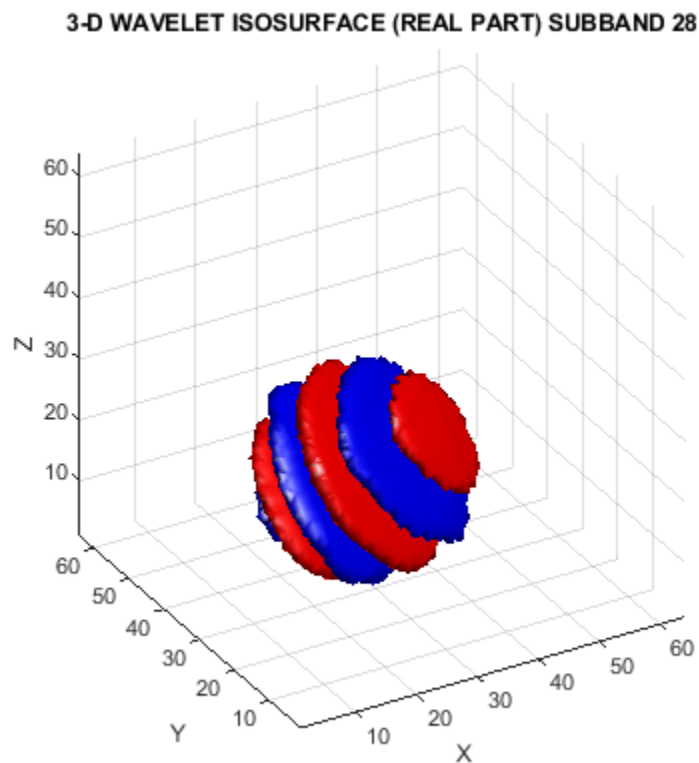
3-D WAVELET ISOSURFACE (IMAGINARY PART) SUBBAND 28
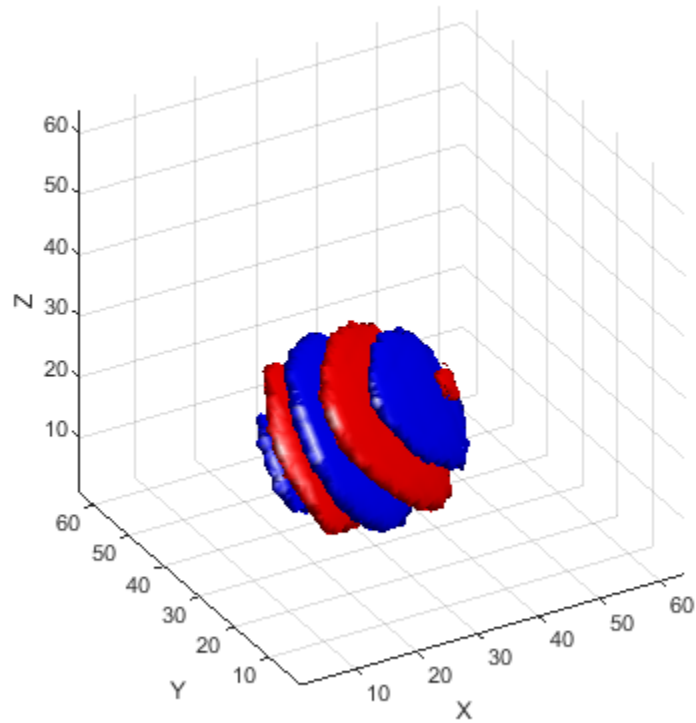

```
helperVisualize3D('Dual-Tree',25,'separate')
```

3-D WAVELET ISOSURFACE (REAL PART) SUBBAND 25
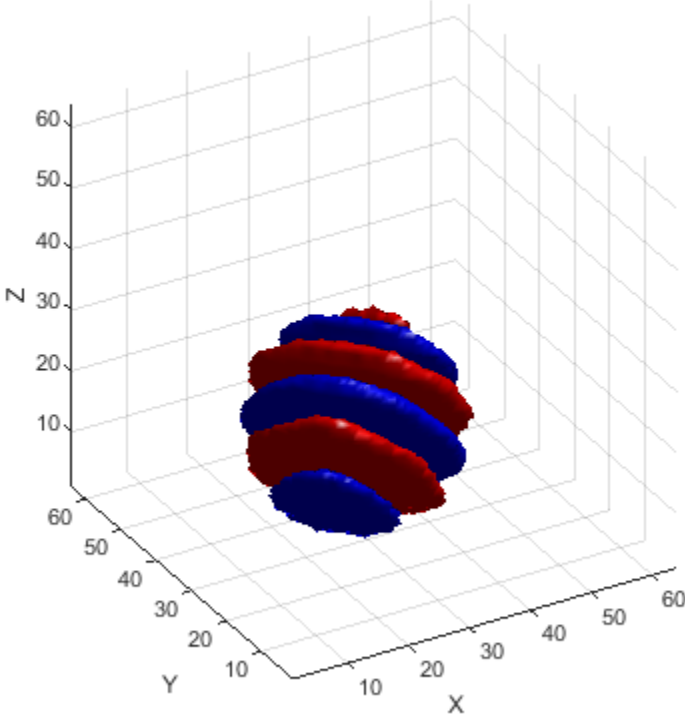
3-D WAVELET ISOSURFACE (IMAGINARY PART) SUBBAND 25
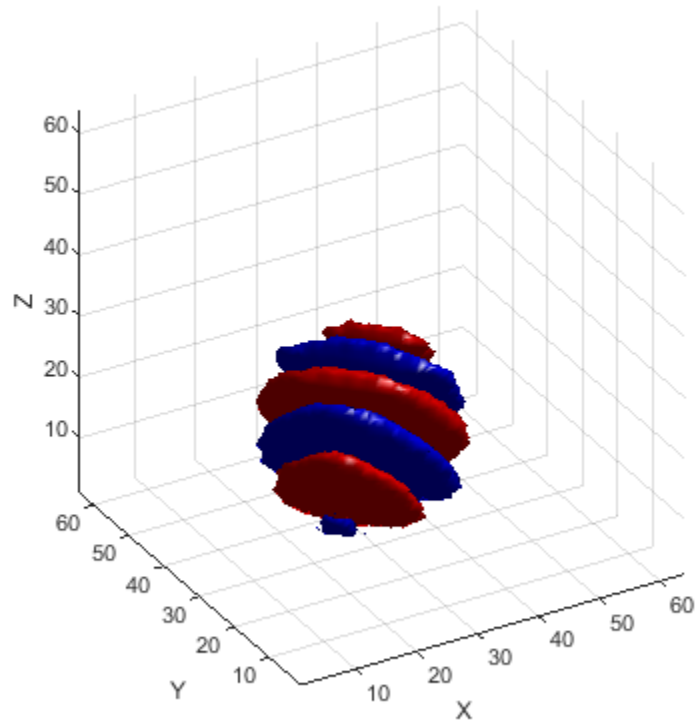
The red portion of the isosurface plot indicates the positive excursion of the wavelet from zero, while blue denotes the negative excursion. You can clearly see the directional selectivity in space of the real and imaginary parts of the dual-tree wavelets. Now visualize one of the dual-tree subbands 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
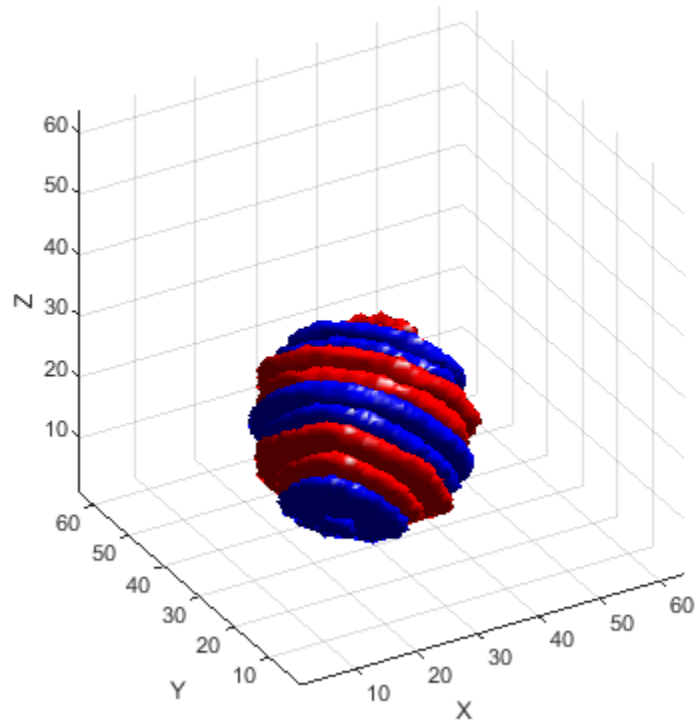
The preceding plot demonstrates that the real and imaginary parts are shifted versions of each other in space. This reflects the fact that the imaginary part of the complex wavelet is the approximate Hilbert transform of the real part. Next, visualize the isosurface of a real orthogonal wavelet in 3-D for comparison.

```
helperVisualize3D('DWT',7)
```

3D DWT WAVELET ISOSURFACE SUBBAND 7
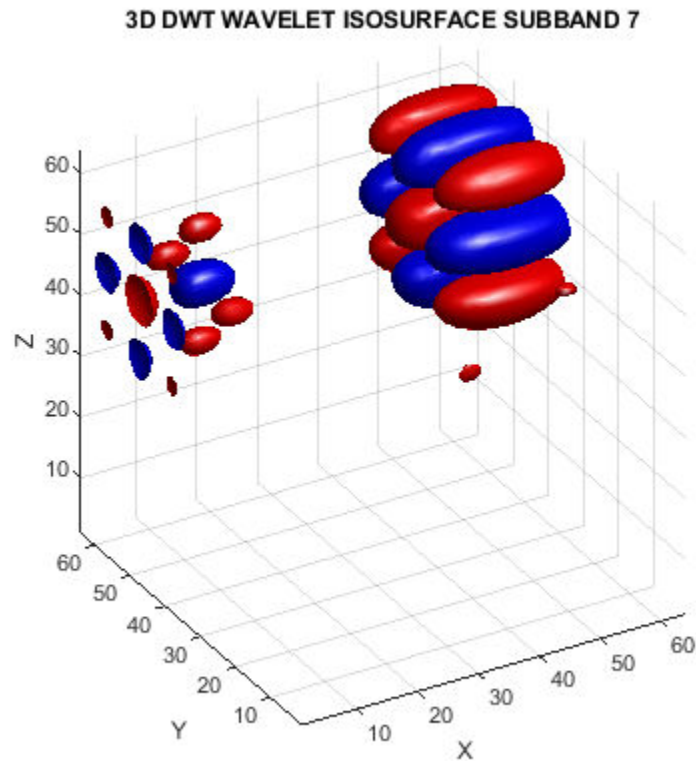
The mixing of orientations observed in the 2-D DWT is even more pronounced in 3-D. Just as in the 2-D case, the mixing of orientations in 3-D leads to significant ringing, or blocking artifacts. To demonstrate this, examine the 3-D DWT and DTCWT wavelet details of a spherical volume. The sphere is 64-by-64-by-64.

```
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('DTCWT Level 2 Details')
```

DTCWT Level 2 Details
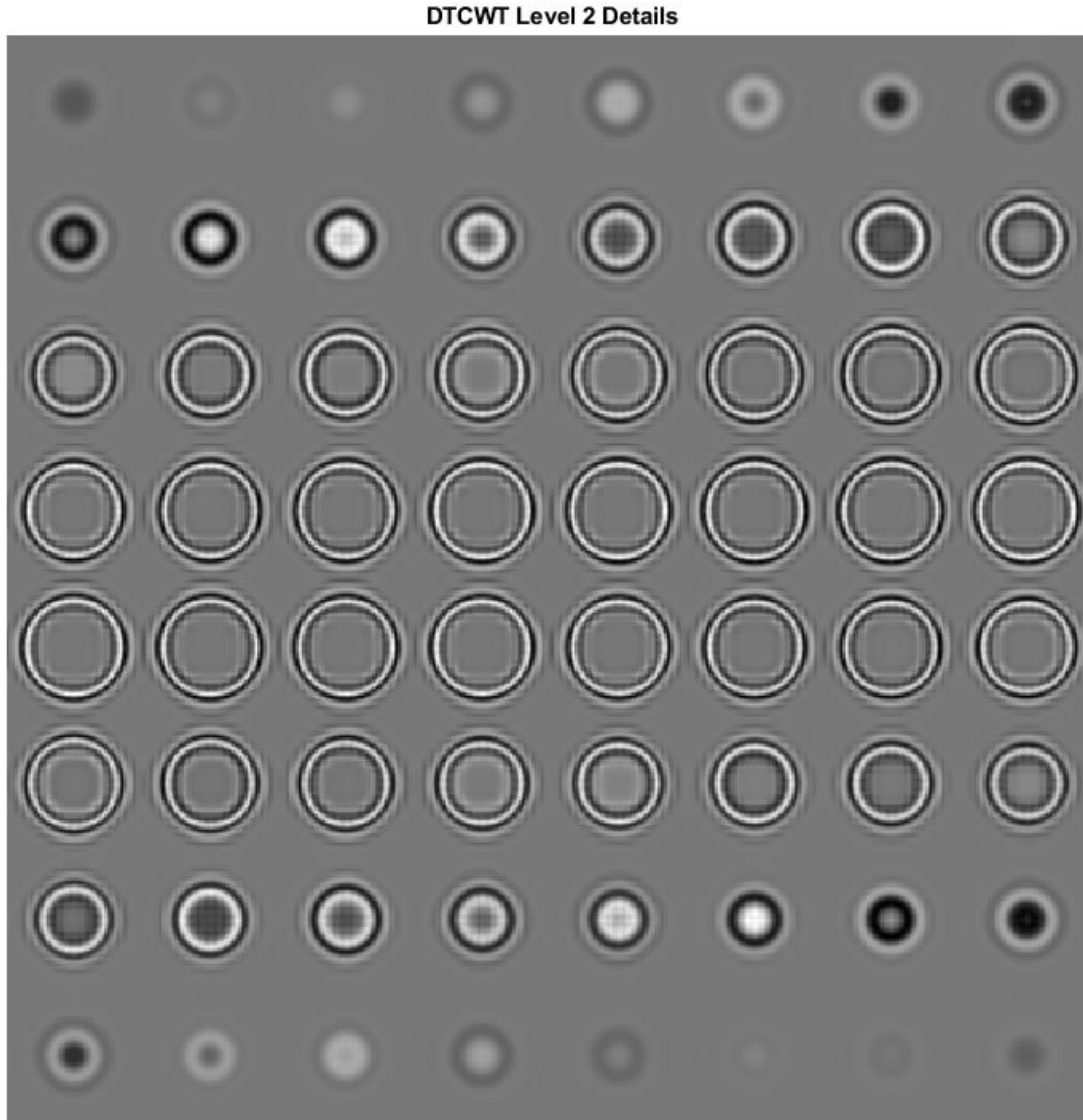
Compare the preceding plot against the second-level details based on the separable DWT.

```
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
sphrrecDWT = waverec3(sphrDEC);
figure
montage(reshape(sphrrecDWT,[64 64 1 64]),'DisplayRange',[])
title('DWT Level 2 Details')
```

DWT Level 2 Details
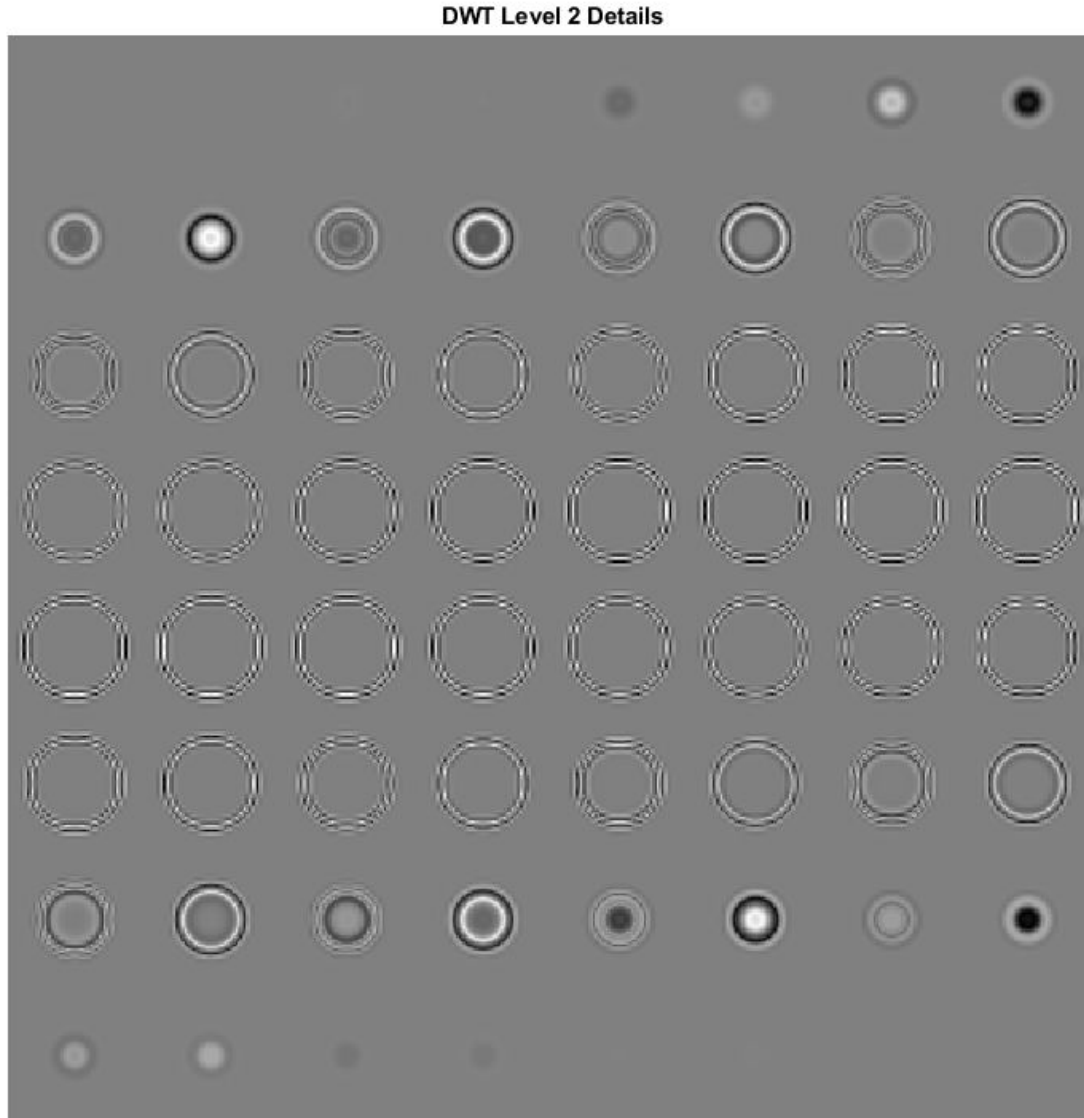
Zoom in on the images in both the DTCWT and DWT montages and you will see how prominent the blocking artifacts in the DWT details are compared to the DTCWT.

Volume Denoising

Similar to the 2-D case, the directional selectivity of the 3-D DTCWT 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',[])
```
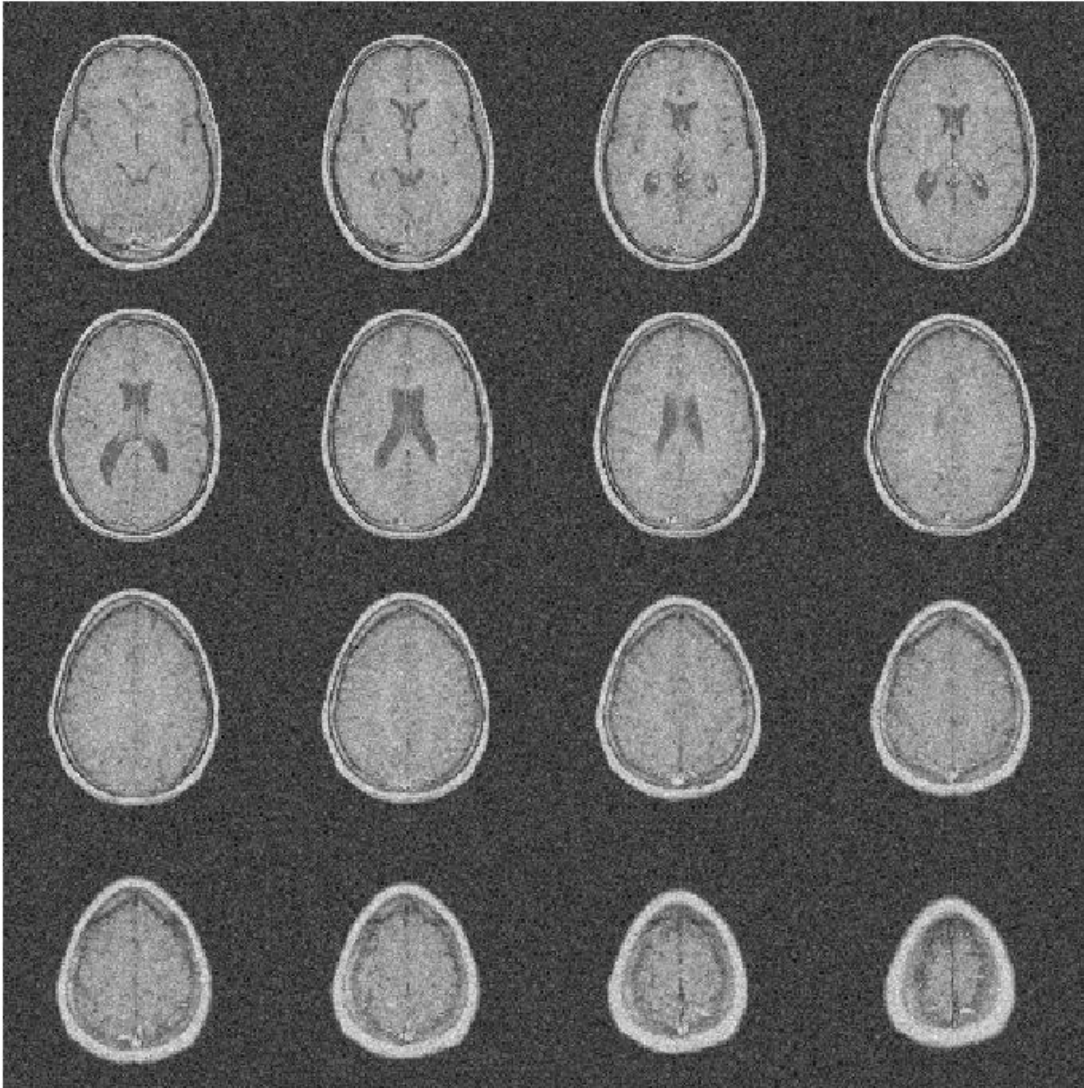
Note that the original SNR prior to denoising is approximately 11 dB.

```
20*log10(norm(origMRI(:),2)/norm(origMRI(:)-noisyMRI(:),2))
```

```
ans = 11.2997
```

Denoise the MRI dataset down to level 4 using both the DTCWT 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 DTCWT and DWT obtained at the best SNR.

```
[imrecDTCWT,imrecDWT] = helperCompare3DDenoising(origMRI,noisyMRI);
```
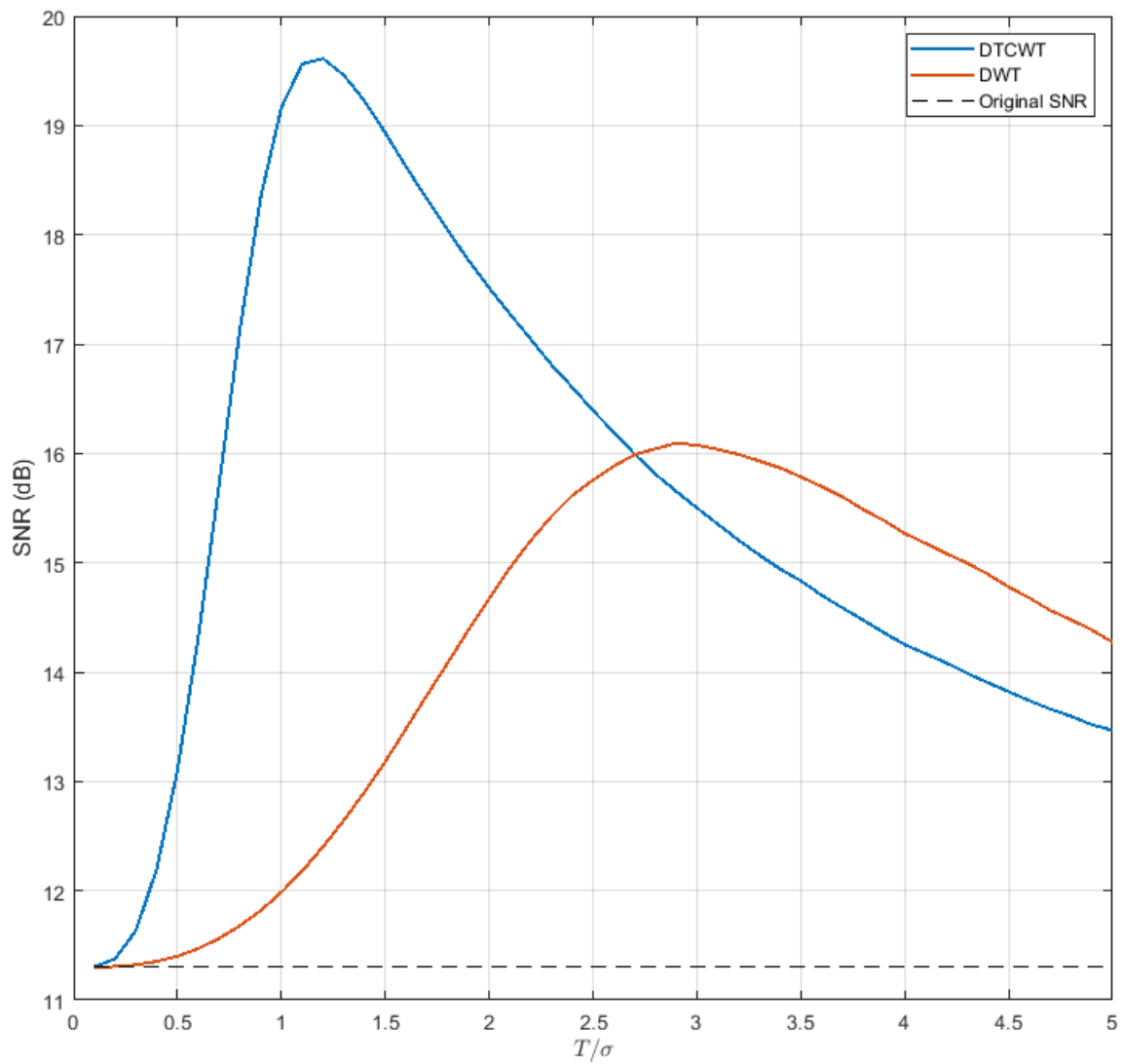
```
figure
montage(reshape(imrecDTCWT,[128 128 1 16]),'DisplayRange',[])
title('DTCWT Denoised Volume')
```

DTCWT Denoised Volume
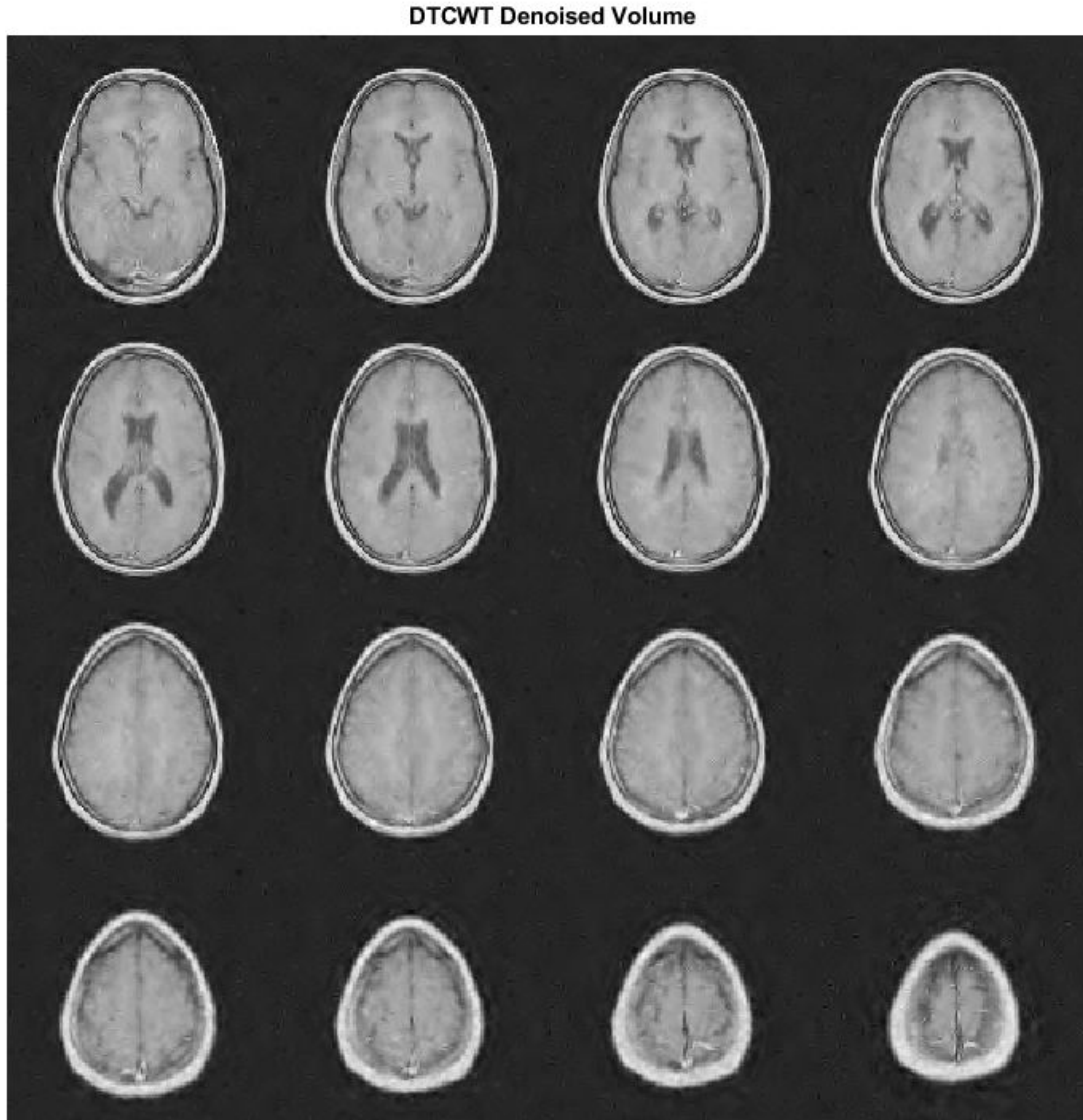
```
figure
montage(reshape(imrecDWT,[128 128 1 16]),'DisplayRange',[])
title('DWT Denoised Volume')
```

DWT Denoised Volume
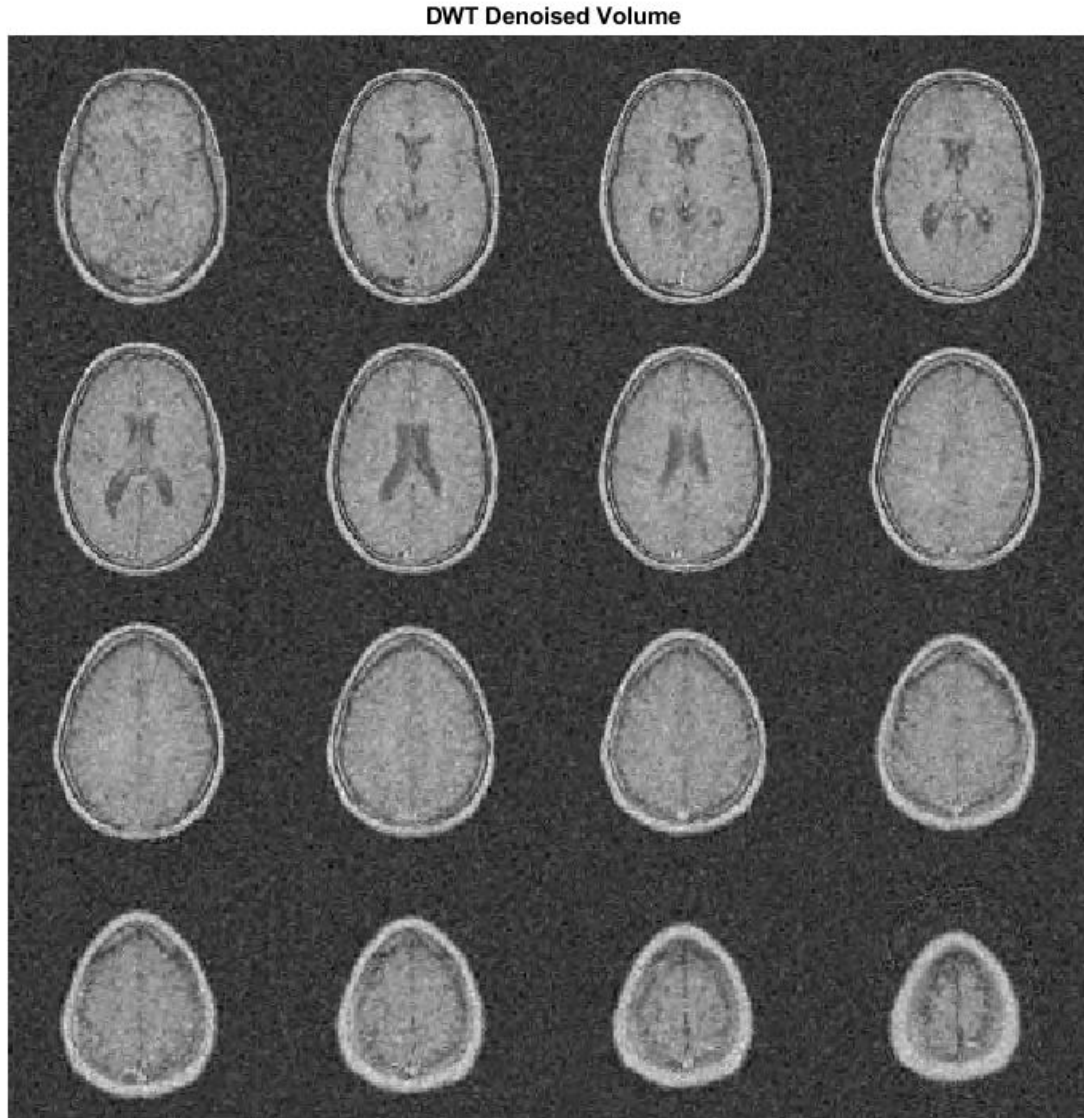
Restore the original extension mode.

```
dwtmode(origmode,'nodisplay')
```

Summary

We have shown that the dual-tree CWT 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.

References

1 Huizhong Chen, and N. Kingsbury. "Efficient Registration of Nonrigid 3-D Bodies." *IEEE Transactions on Image Processing* 21, no. 1 (January 2012): 262–72. https://doi.org/10.1109/TIP.2011.2160958.

2 Kingsbury, Nick. "Complex Wavelets for Shift Invariant Analysis and Filtering of Signals." *Applied and Computational Harmonic Analysis* 10, no. 3 (May 2001): 234–53. https://doi.org/10.1006/acha.2000.0343.

3 Percival, Donald B., and Andrew T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge ; New York: Cambridge University Press, 2000.

4 Selesnick, I.W., R.G. Baraniuk, and N.C. Kingsbury. "The Dual-Tree Complex Wavelet Transform." IEEE Signal Processing Magazine 22, no. 6 (November 2005): 123–51. https://doi.org/10.1109/MSP.2005.1550194.

5 Selesnick, I. "The Double Density DWT." *Wavelets in Signal and Image Analysis: From Theory to Practice* (A. A. Petrosian, and F. G. Meyer, eds.). Norwell, MA: Kluwer Academic Publishers, 2001, pp. 39–66.

6 Selesnick, I.W. "The Double-Density Dual-Tree DWT." *IEEE Transactions on Signal Processing* 52, no. 5 (May 2004): 1304–14. https://doi.org/10.1109/TSP.2004.826174.

See Also

dualtree | dualtree2 | dualtree3

Related Examples

- "Analytic Wavelets Using the Dual-Tree Wavelet Transform" on page 3-219

More About

- "Critically Sampled and Oversampled Wavelet Filter Banks" on page 3-2

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 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.

Create the zero signal 256 samples in length. Obtain two dual-tree transforms of the zero signal down to level 5. By default, `dualtree` uses the default near-symmetric biorthogonal filter pair `nearsym5_7` for level 1, and the orthogonal Q-shift Hilbert wavelet filter pair of length 10 for levels greater than 1.

```
x = zeros(256,1);
[a1,d1] = dualtree(x,'Level',5);
[a2,d2] = dualtree(x,'Level',5);
```

Set a single level-five detail coefficient in each of the two trees to 1 and invert the transform to obtain the wavelets.

```
d1{5}(5) = 1;
d2{5}(5) = 1i;
wav1 = idualtree(a1,d1);
wav2 = idualtree(a2,d2);
```

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')
```
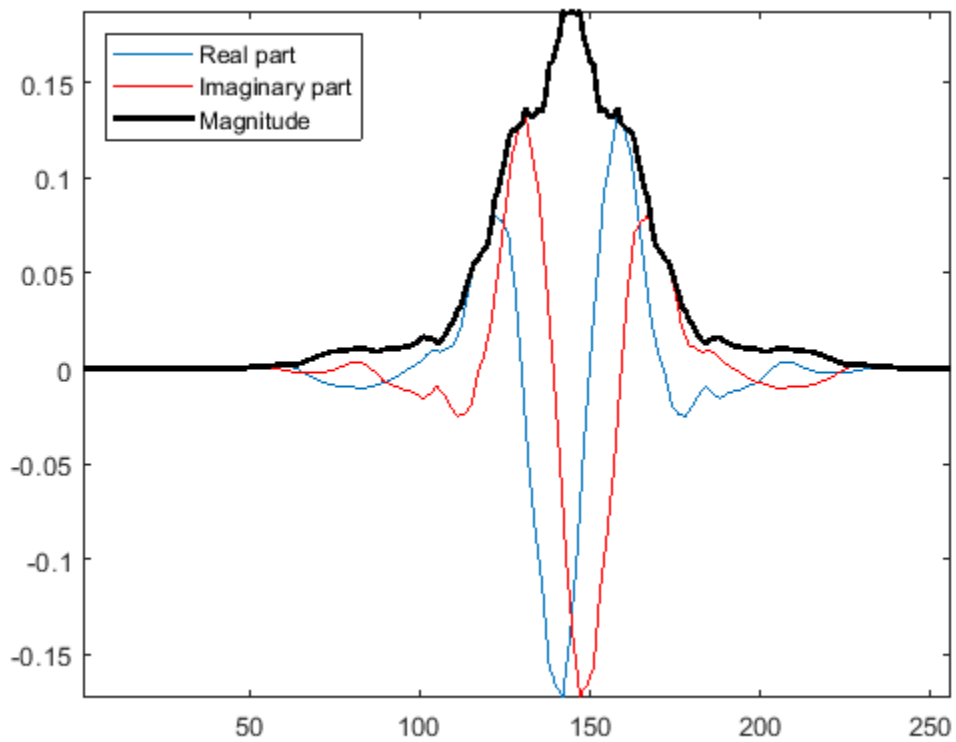
Fourier transform the analytic wavelet and plot the magnitude.

```
zdft = fft(analwav);
domega = (2*pi)/length(analwav);
omega = 0:domega:(2*pi)-domega;
clf;
plot(omega,abs(zdft))
xlabel('Radians/Sample')
axis tight
```
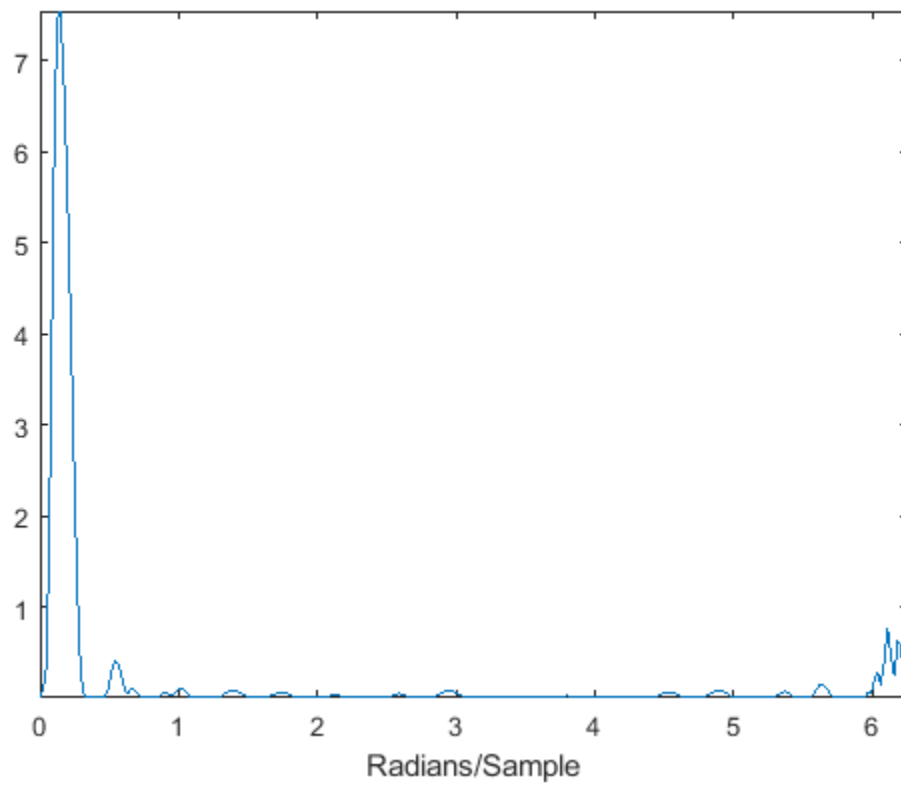
The Fourier transform of the wavelet has support on essentially only half of the frequency axis.

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` for illustration. To execute that code, you must have 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')
```
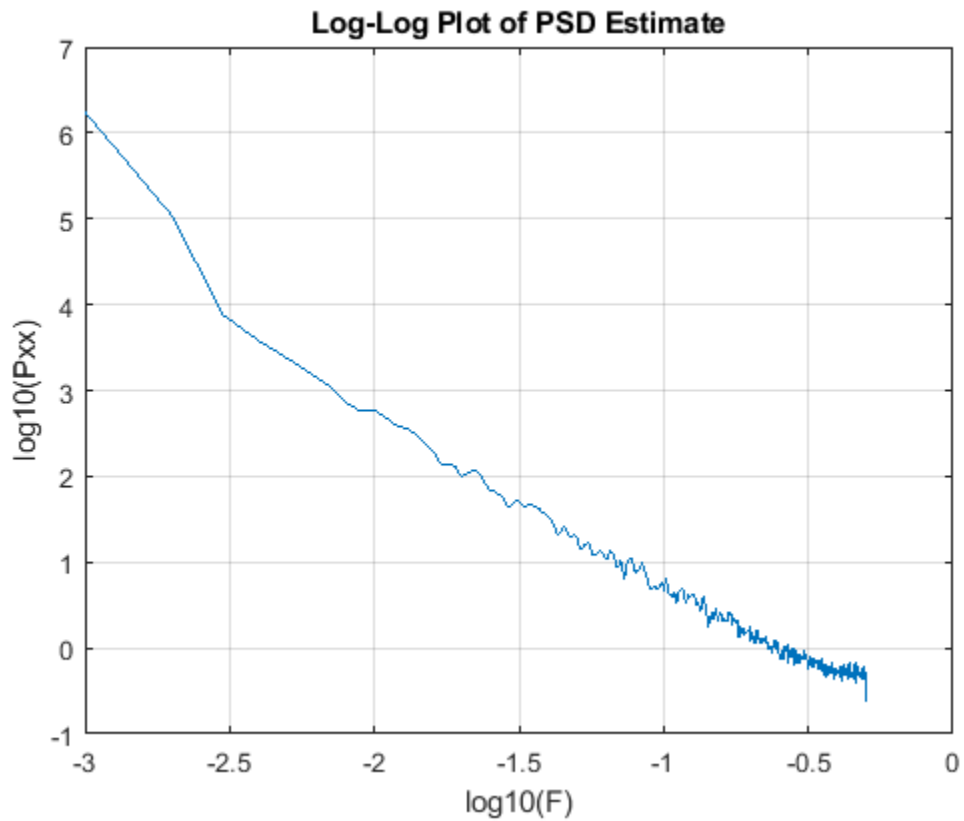
Regress the log PSD values on the log frequencies. Note you must ignore zero frequency to avoid taking the log of zero.

```
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))]);
```

Estimated Slope is -1.8652
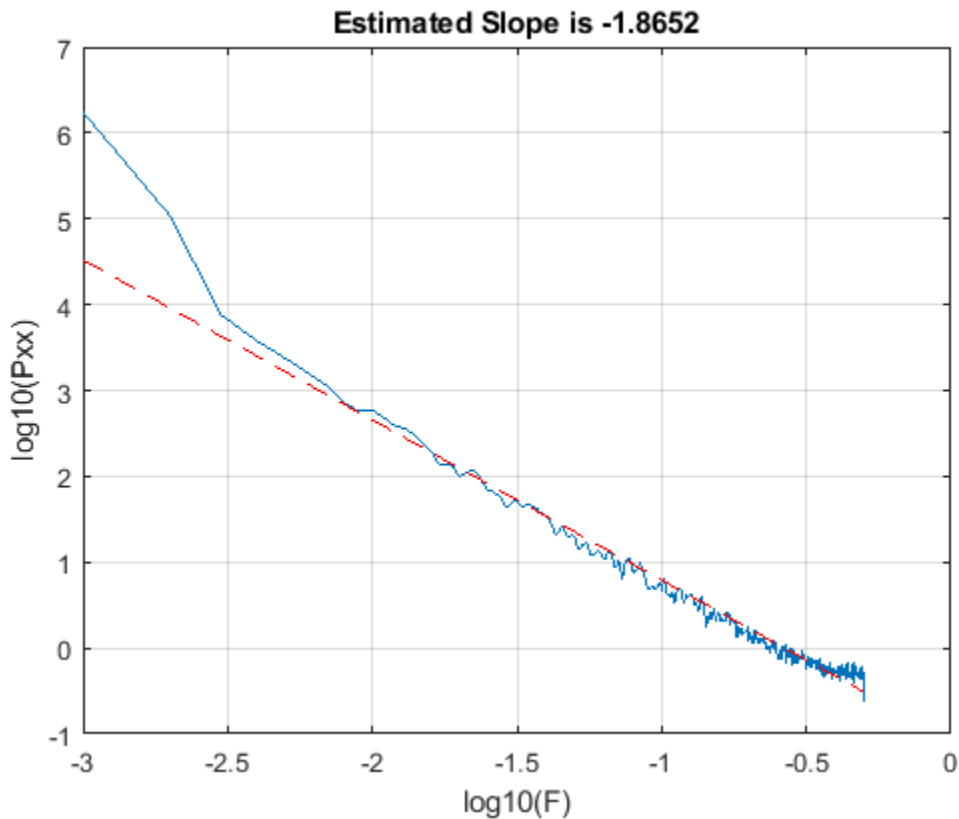
Alternatively, you can use both discrete and continuous wavelet analysis techniques to estimate the exponent. The relationship between the Holder exponent, H, returned by dwtleader and wtmm and α is this scenario is $\alpha = 2H + 1$.

```
[dhbrown,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');
```
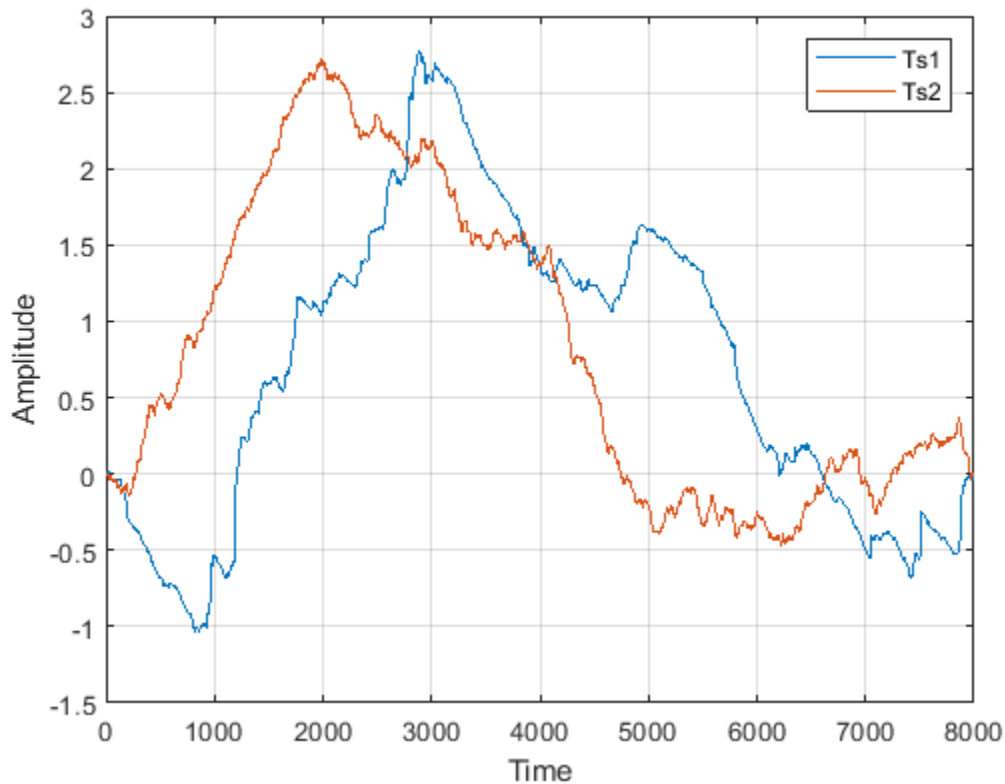

The signals have very similar second order statistics. If you look at the means, RMS values, and variances of `Ts1` and `Ts2`, the values are almost identical. The PSD estimates are also very similar.

```
pwelch([Ts1 Ts2],kaiser(1e3,10))
```

Welch Power Spectral Density Estimate
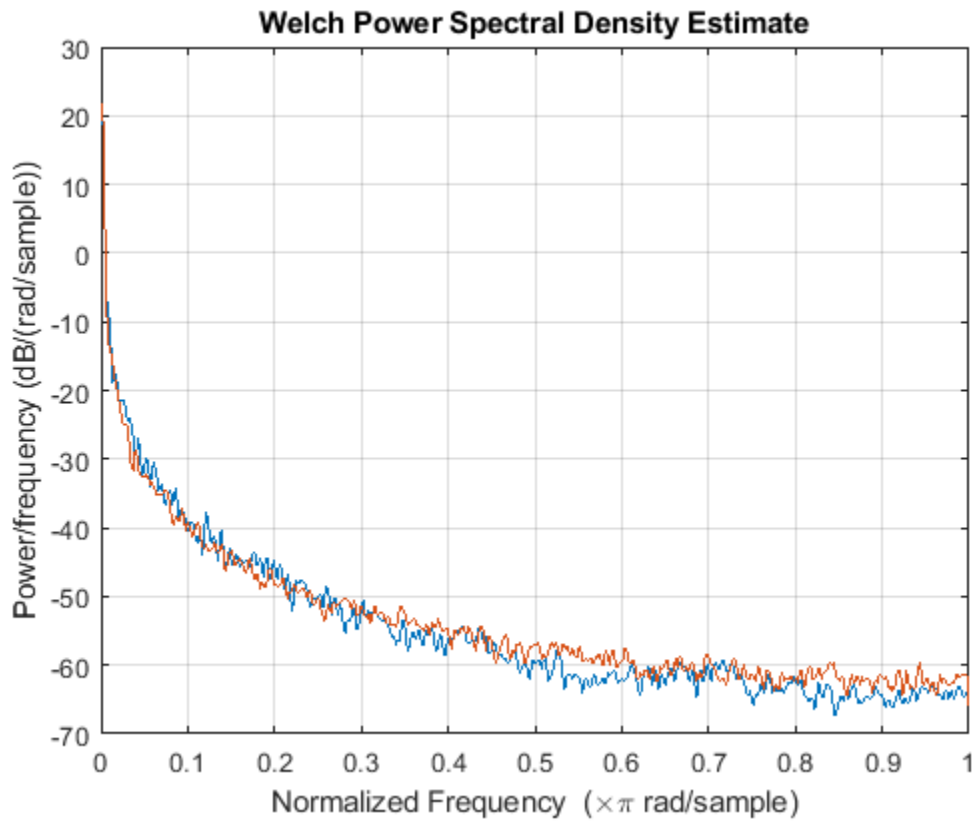
The autocorrelation sequences decay very slowly for both time series and are not informative for differentiating the time series.

```
[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')
```
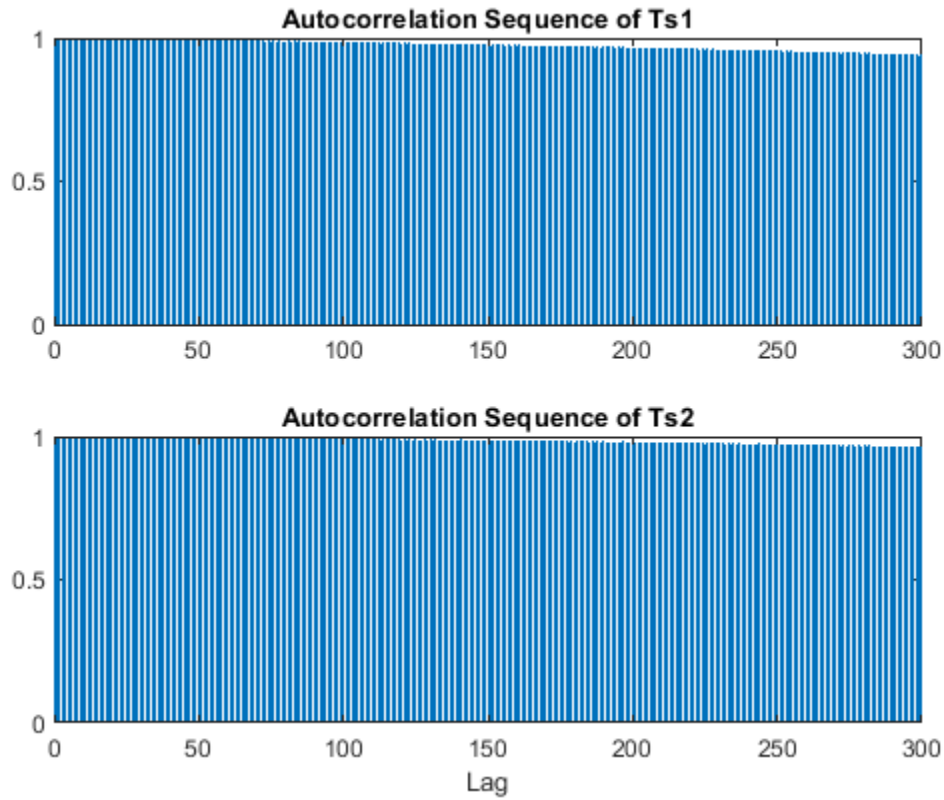
Even at a lag of 300, the autocorrelations are 0.94 and 0.96 respectively.

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');
```
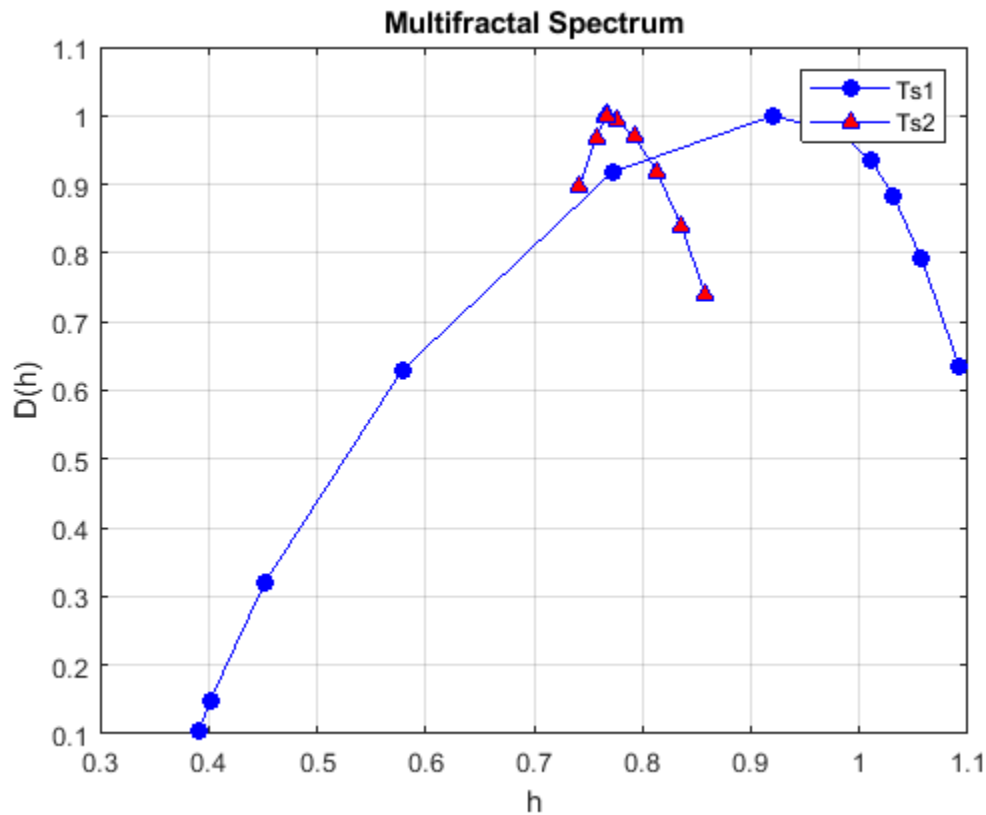
The multifractal spectrum effectively shows the distribution of scaling exponents for a signal. Equivalently, the multifractal spectrum provides a measure of how much the local regularity of a signal varies in time. A signal that is monofractal exhibits essentially the same regularity everywhere in time 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');
```
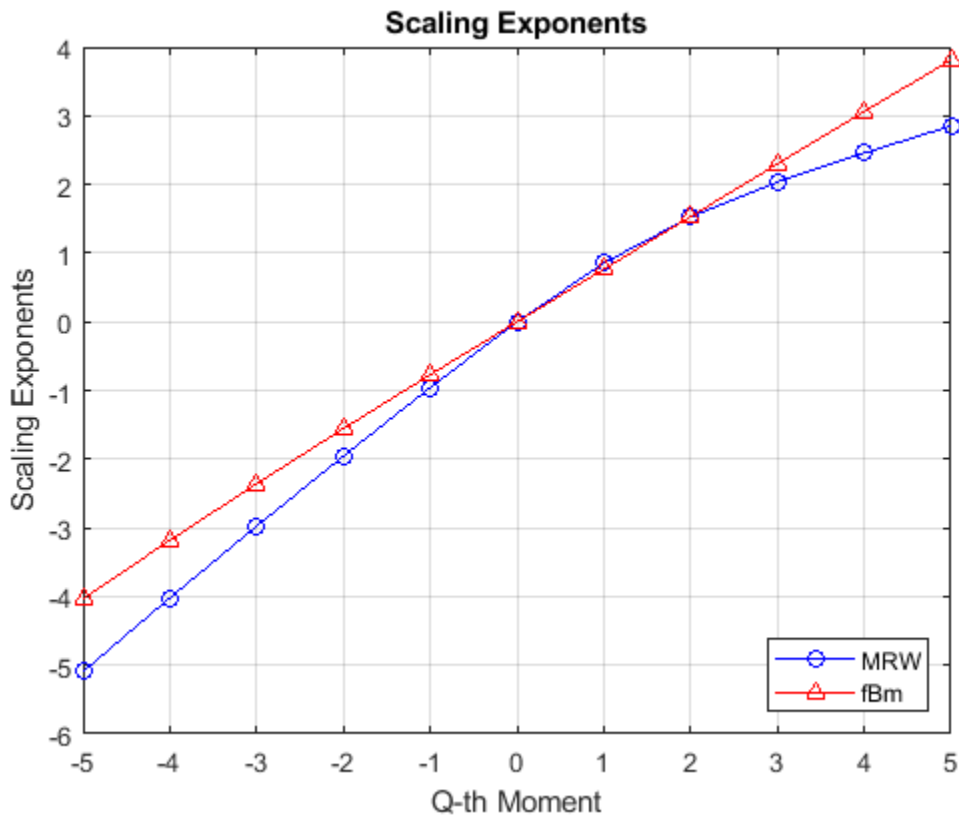
The scaling exponents for the fBm process are a linear function of the moments, while the exponents for the MRW process show a departure from linearity. The same information is summarized by the 1st, 2nd, and 3rd cumulants. The first cumulant is the estimate of the slope, in other words, it captures 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');
```

Multifractal Spectrum
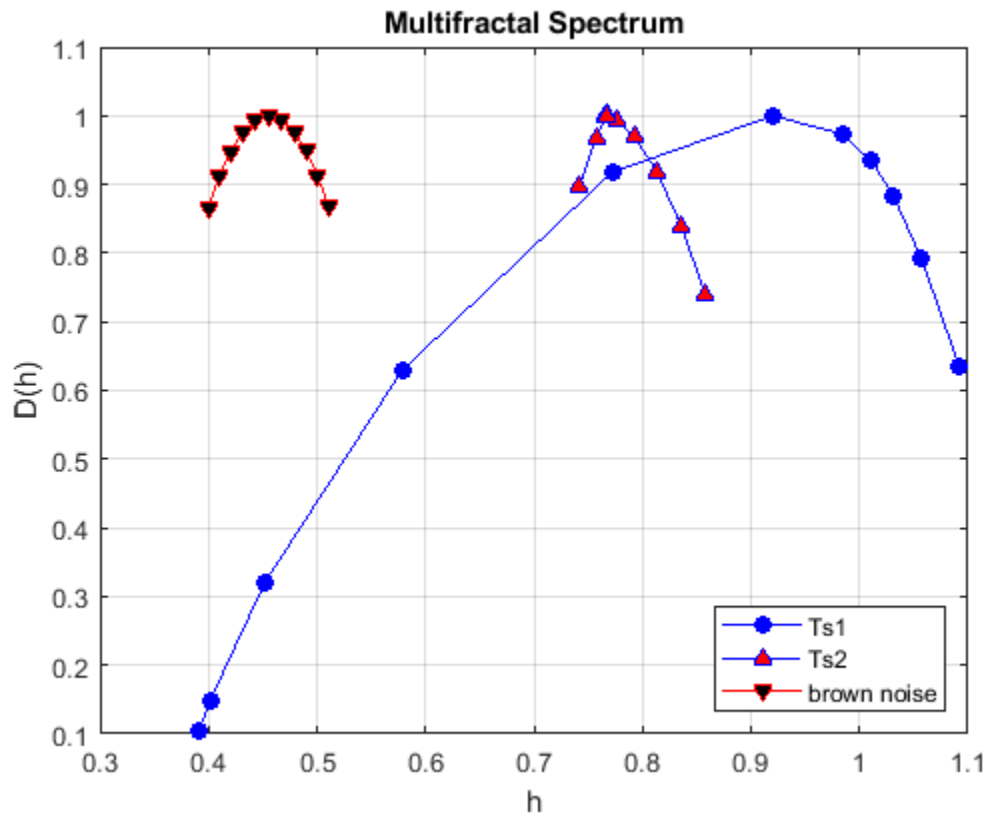
Where is the Process Going Next? Persistent and Antipersistent Behavior

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;
```
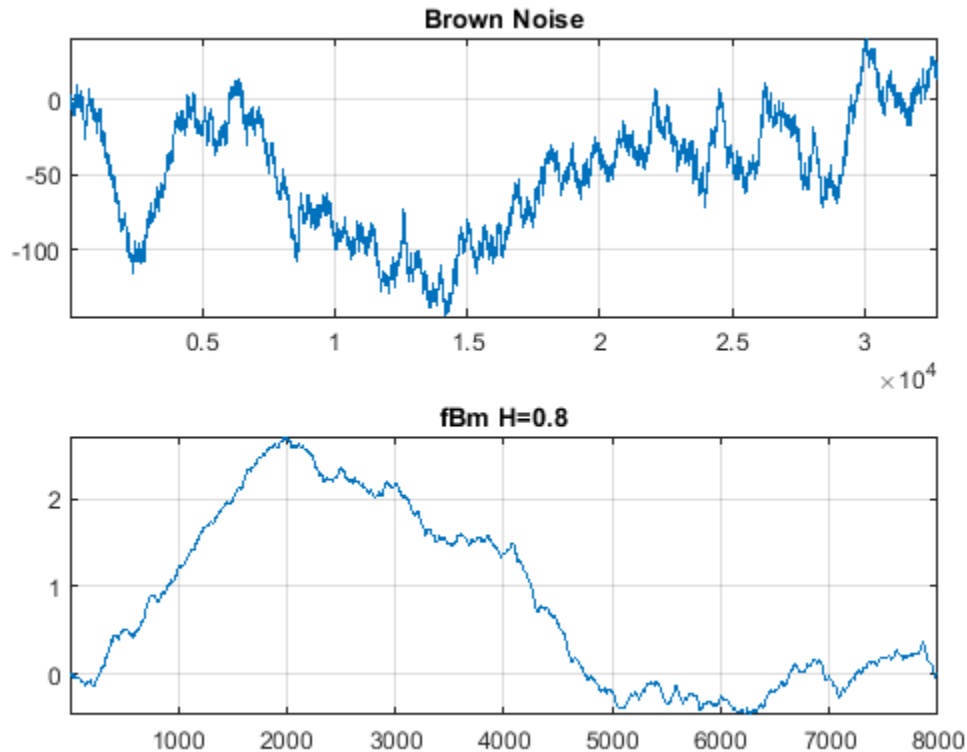
The fBm data is much smoother than the brown noise. Brown noise, also known as a random walk, has a theoretical Holder exponent of 0.5. This value forms a boundary between processes with Holder exponents, H, from 0<H<0.5 and those with Holder exponents in the interval 0.5<H<1. The former are called *antipersistent* 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;
```

Seoul Composite Index - 07/1997-04/2016

Taiwan Weighted Index - 07/1997-04/2016
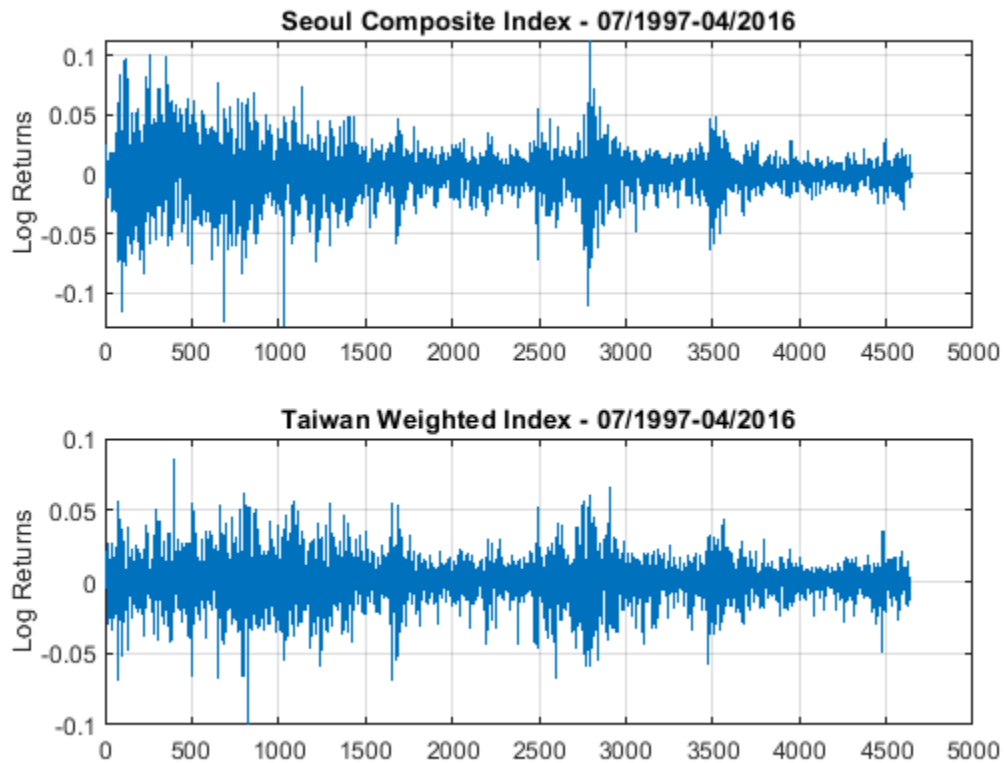
Obtain and plot the multifractal spectra of these two time series.

```
[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');
```

Multifractal Spectrum
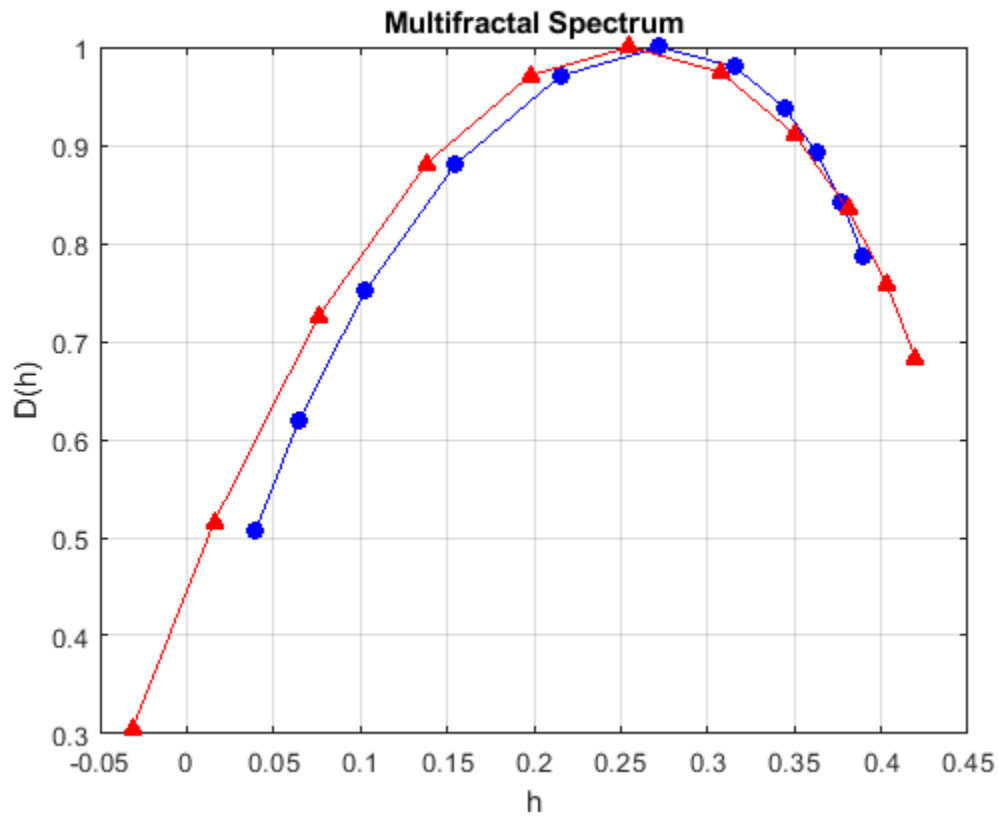
From the multifractal spectrum, it is clear that both time series are antipersistent. For comparison, plot the multifractal spectra of the two financial time series along with the brown noise and fBm data shown earlier.

```
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;
```
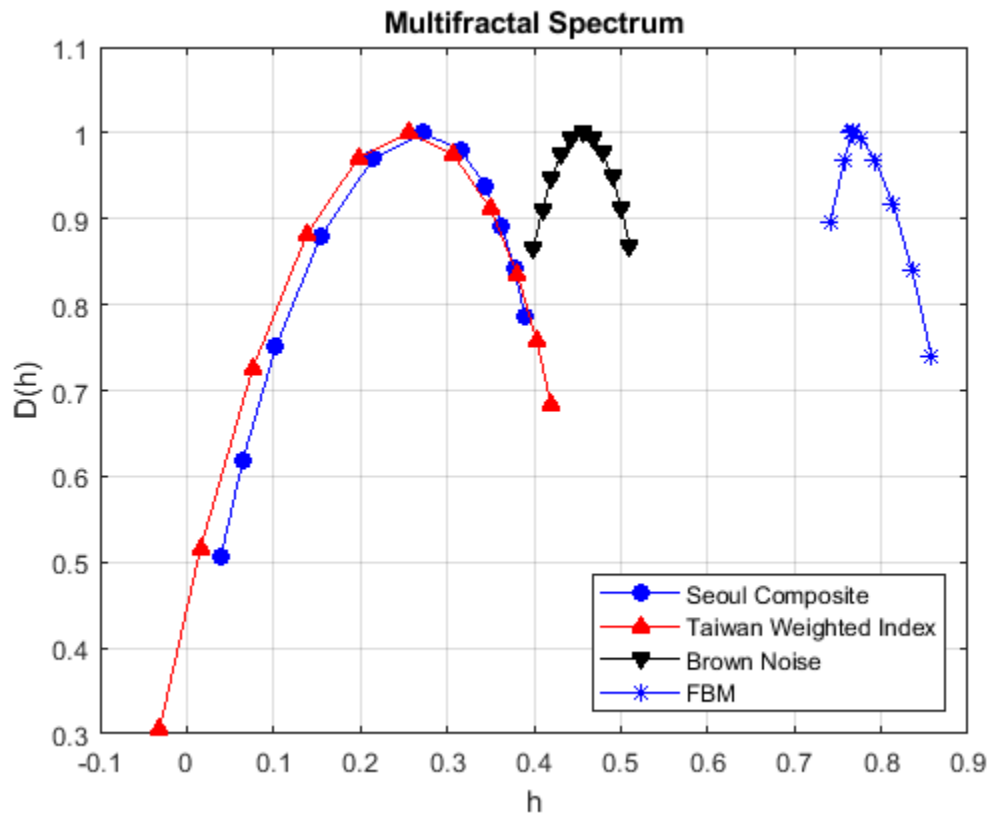
Determining that a process is antipersistent or persistent is useful in predicting the future. For example, a time series with long memory that is increasing can be expected to continue increasing. While a time series that exhibits antipersistence can be expected to move in the opposite direction.

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');
```
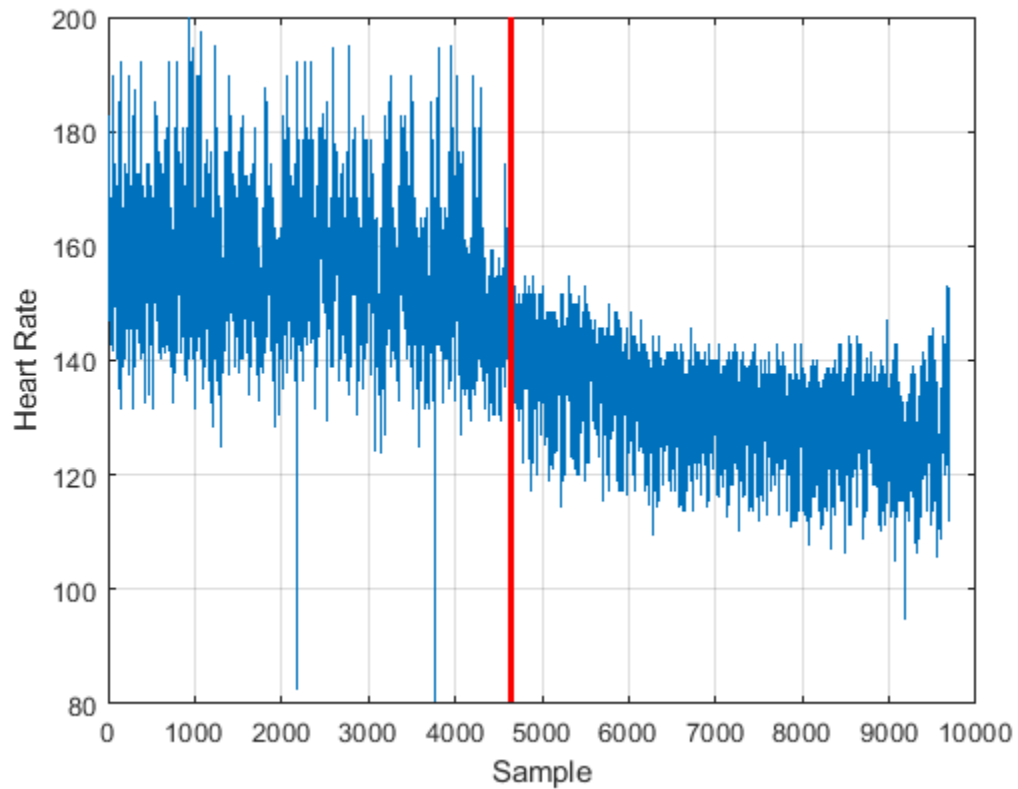
Split the data into pre-drug and post-drug data sets. Obtain and plot the multifractal spectra of the two time series.

```
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)');
```
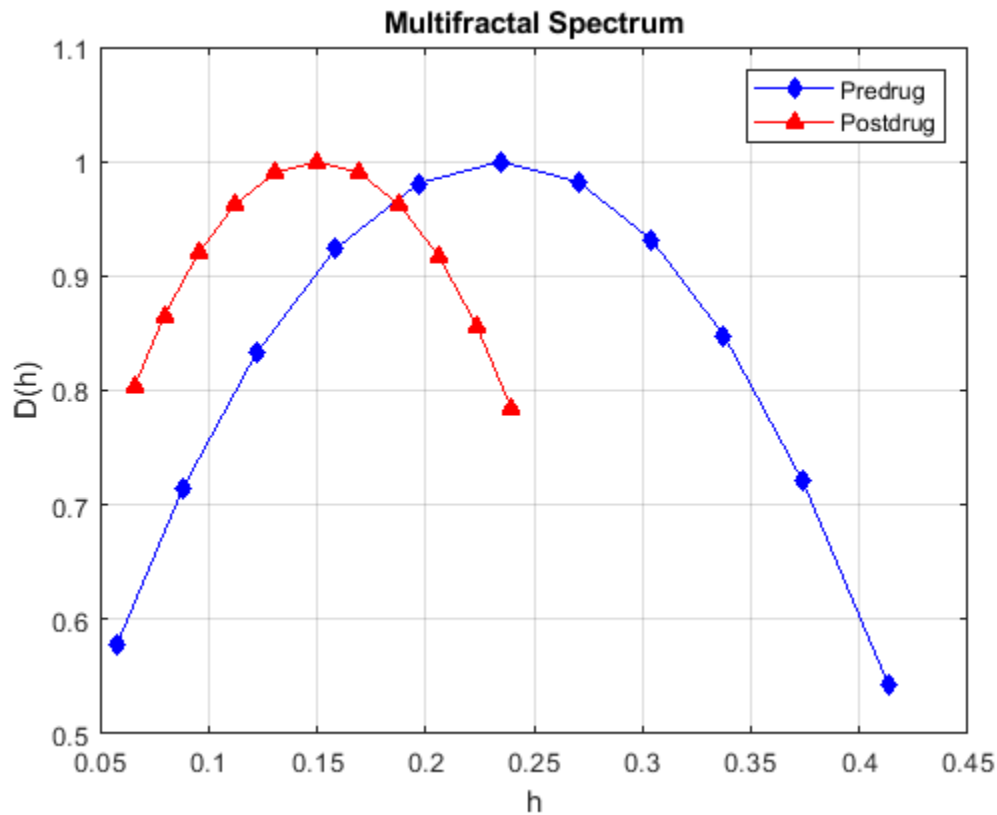
The induction of the drug has led to a 50% reduction in the width of the fractal spectrum. This indicates a significant reduction in the nonlinear dynamics of the heart as measured by heart rate variability. In this case, the reduction of the fractal dimension was part of a medical intervention. In a 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.

Time-Frequency Analysis

Time-Frequency Gallery

This gallery provides you with an overview of the time-frequency analysis features available in 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
"Short-Time Fourier Transform (Spectrogram)" on page 4-3	• The short-time Fourier transform (STFT) has fixed time-frequency resolution. • The spectrogram is the magnitude squared of STFT.	• `stft`: Yes • `spectrogram`: No	"Example: Whale Song" on page 4-5
"Continuous Wavelet Transform (Scalogram)" on page 4-7	• The continuous wavelet transform (CWT) has a variable time-frequency resolution. • The CWT preserves time shifts and time scalings.	Yes	"Example: ECG Signal" on page 4-8
"Wigner-Ville Distribution" on page 4-9	• The Wigner-Ville distribution (WVD) is always real. • Time and frequency marginal densities correspond to instantaneous power and spectral energy density, respectively. • The time resolution of the WVD is equal to the number of input samples.	No	"Example: Otoacoustic Emission" on page 4-10
"Reassignment and Synchrosqueezing" on page 4-11	• 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	• `pspectrum`: No • `fsst`, `wsst`: Yes	"Example: Echolocation Pulse" on page 4-11

Method	Features	Invertible	Examples
"Constant-Q Gabor Transform" on page 4-17	• 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	"Example: Rock Music" on page 4-18
"Empirical Mode Decomposition and Hilbert-Huang Transform" on page 4-18	• Empirical mode decomposition (EMD) decomposes signals into intrinsic mode functions. • The Hilbert-Huang transform (HHT) computes the instantaneous frequency of each empirical mode.	No	"Example: Bearing Vibration" on page 4-19

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))
```
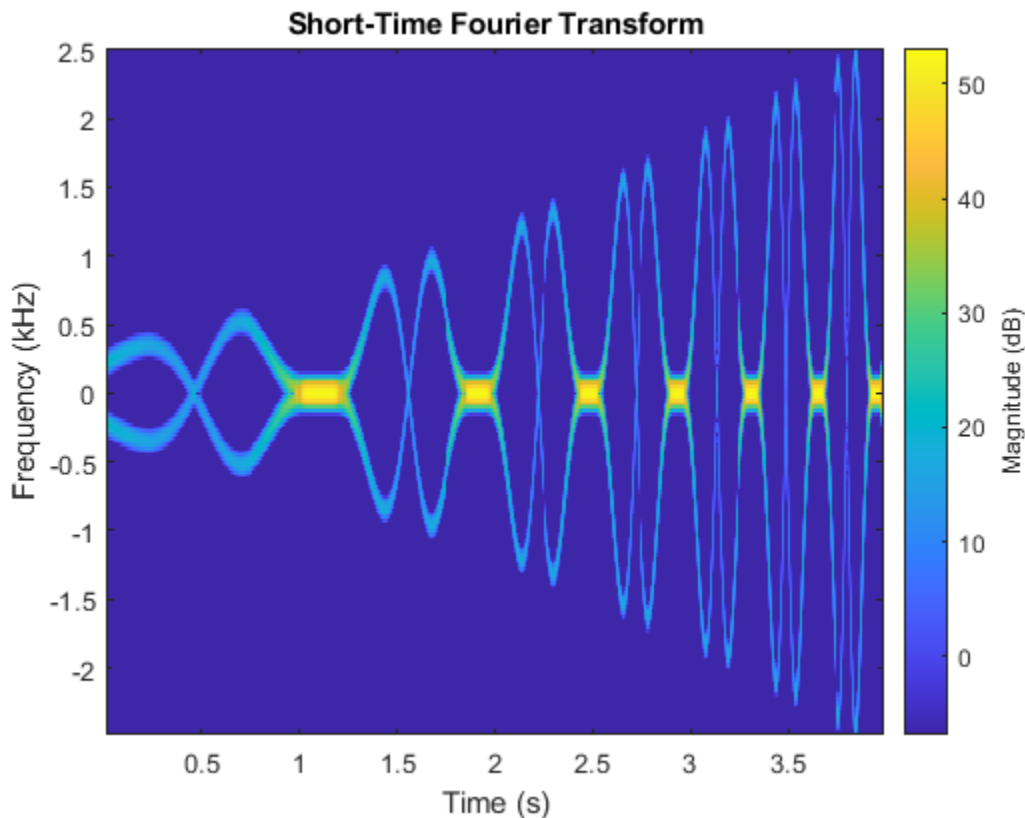
Example: Audio Signal with Decreasing Chirps

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)
```
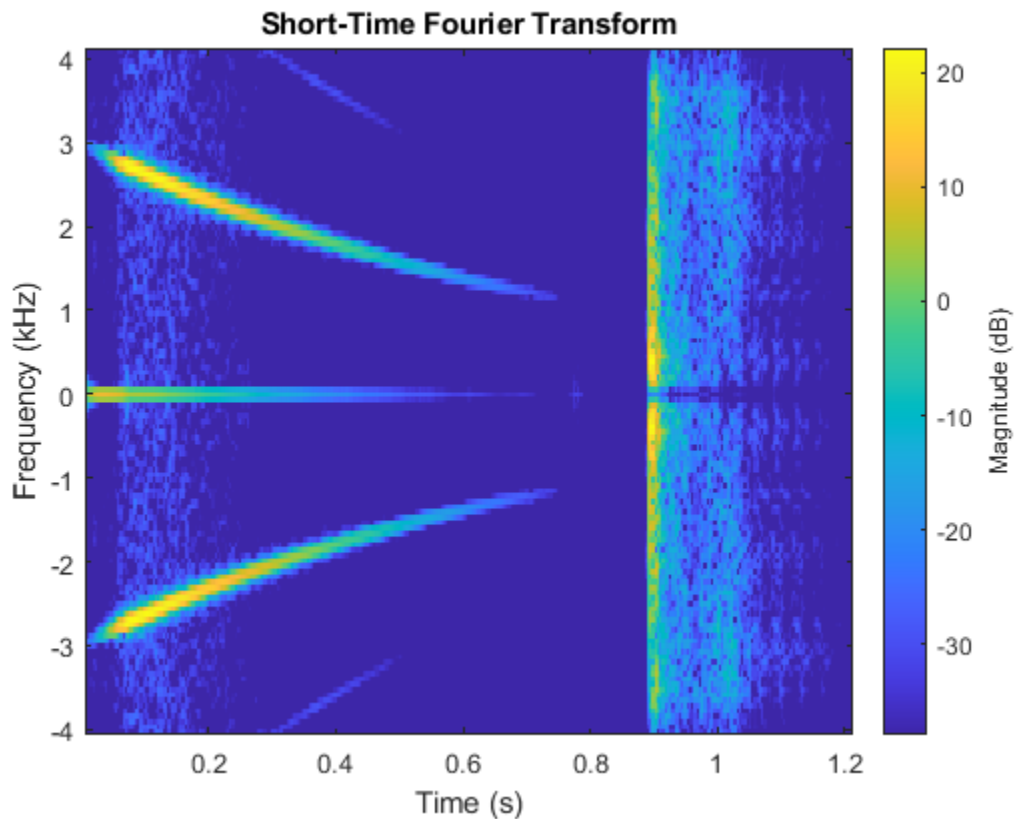

Example: Whale Song

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','data','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)
```

Fres = 20.7503 Hz, Tres = 155.5 ms
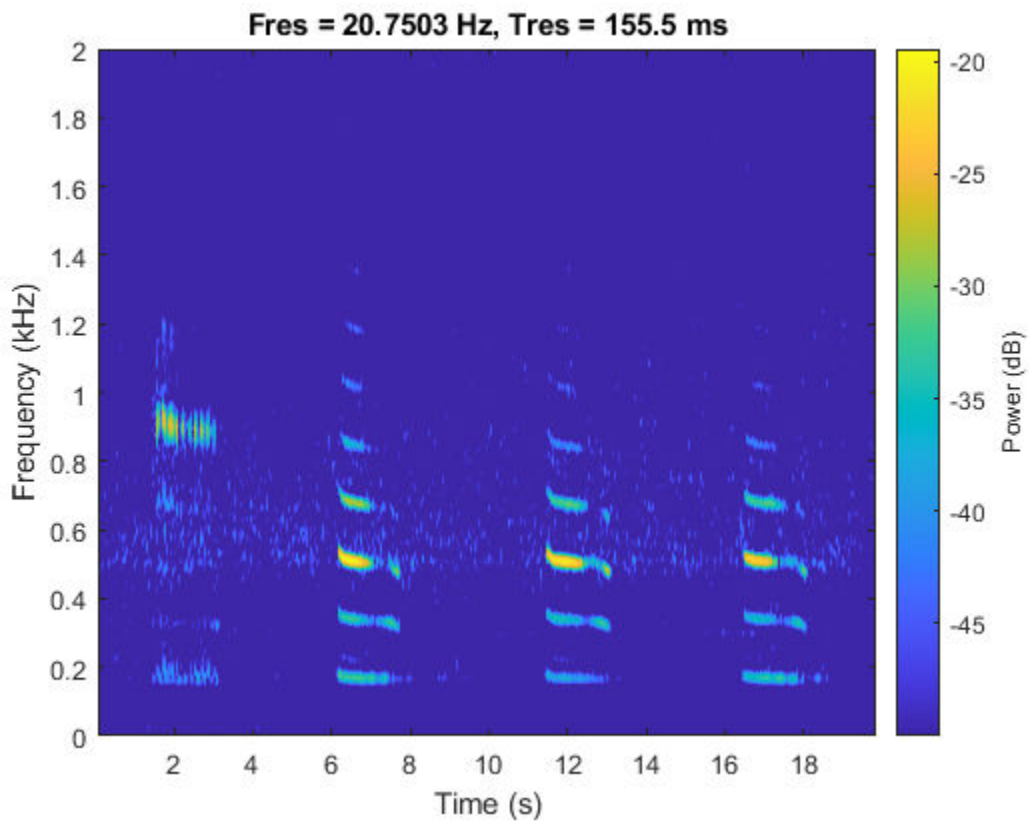
Example: Persistence Spectrum of Transient Signal

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)
```

Fres = 3.9101 Hz, Tres = 1 s
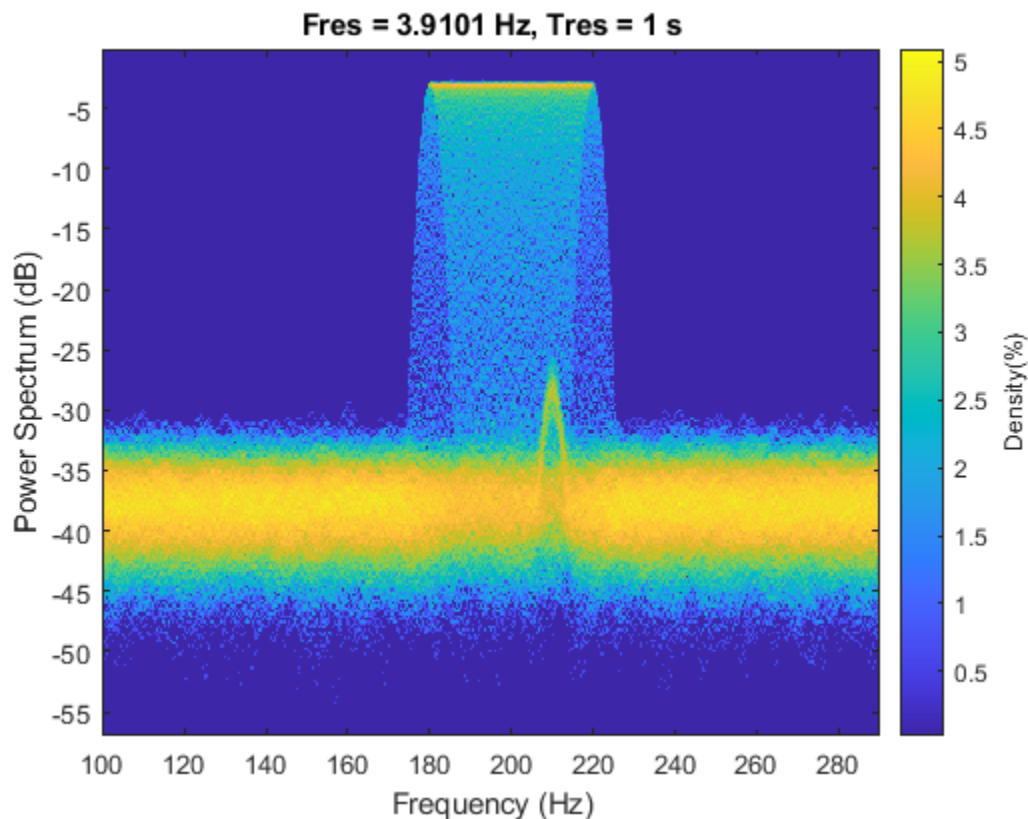
Continuous Wavelet Transform (Scalogram)

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" on page 13-79 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 a 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)
```
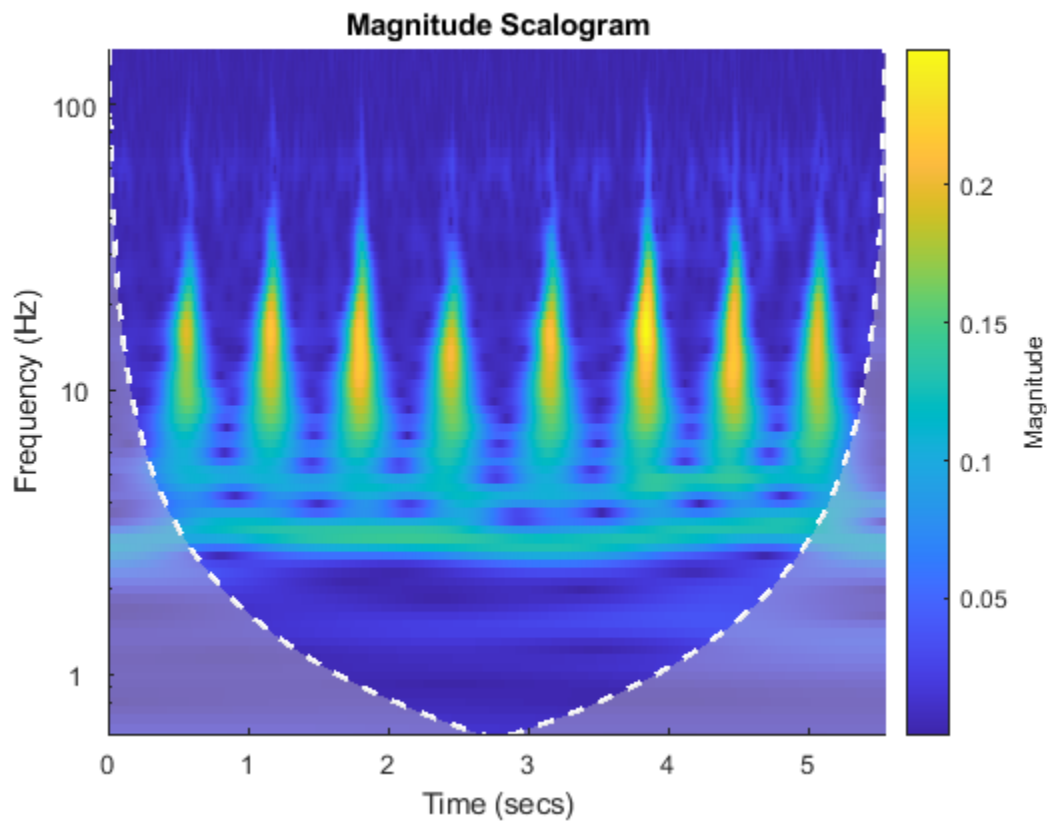
The ECG data is taken from the MIT-BIH Arrhythmia Database [2].

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.
- Time and frequency marginal densities correspond to instantaneous power and spectral energy density, respectively.
- 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,'NumTimel
```
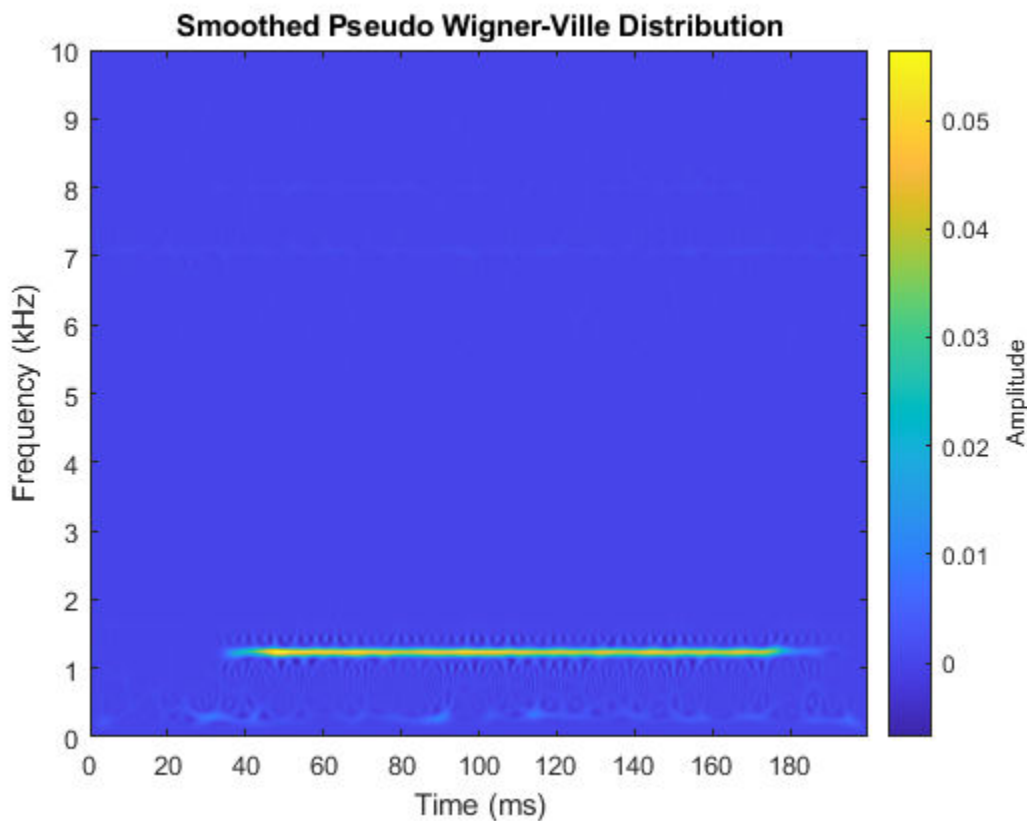

For more details on otoacoustic emissions, see "Determining Exact Frequency Through the Analytic CWT" in "CWT-Based Time-Frequency Analysis" on page 10-27.

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`.)
- `wsst` 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)
```
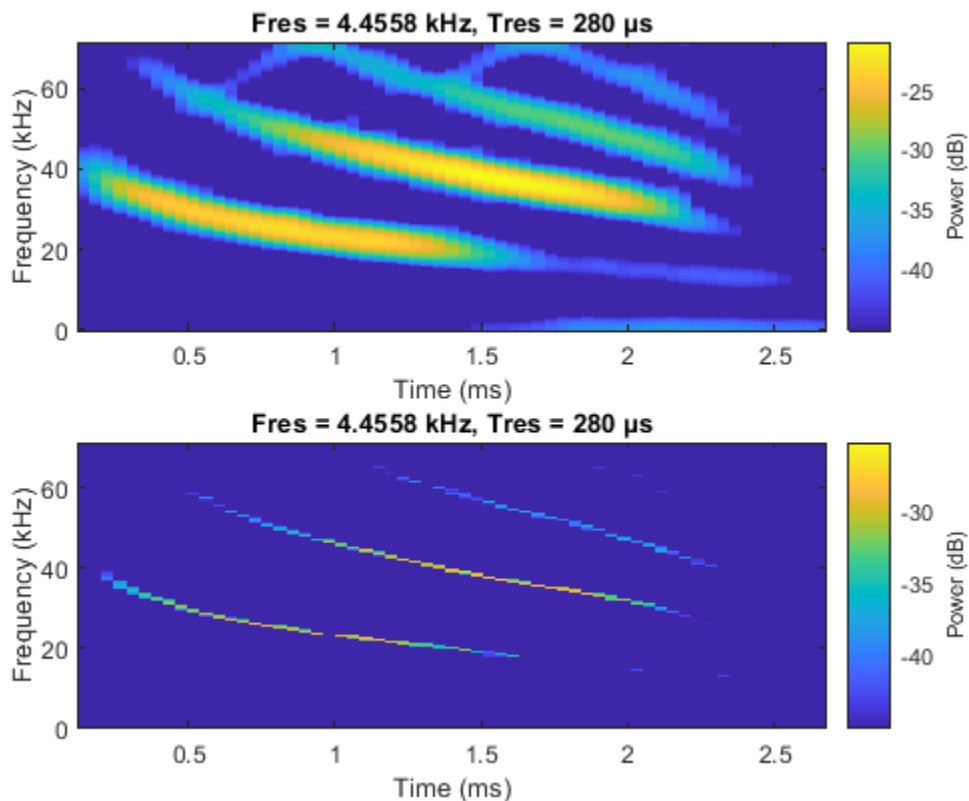

Thanks to Curtis Condon, Ken White, and Al Feng of the Beckman Center at the University of Illinois for the bat data and permission to use it in this example [3].

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')
```
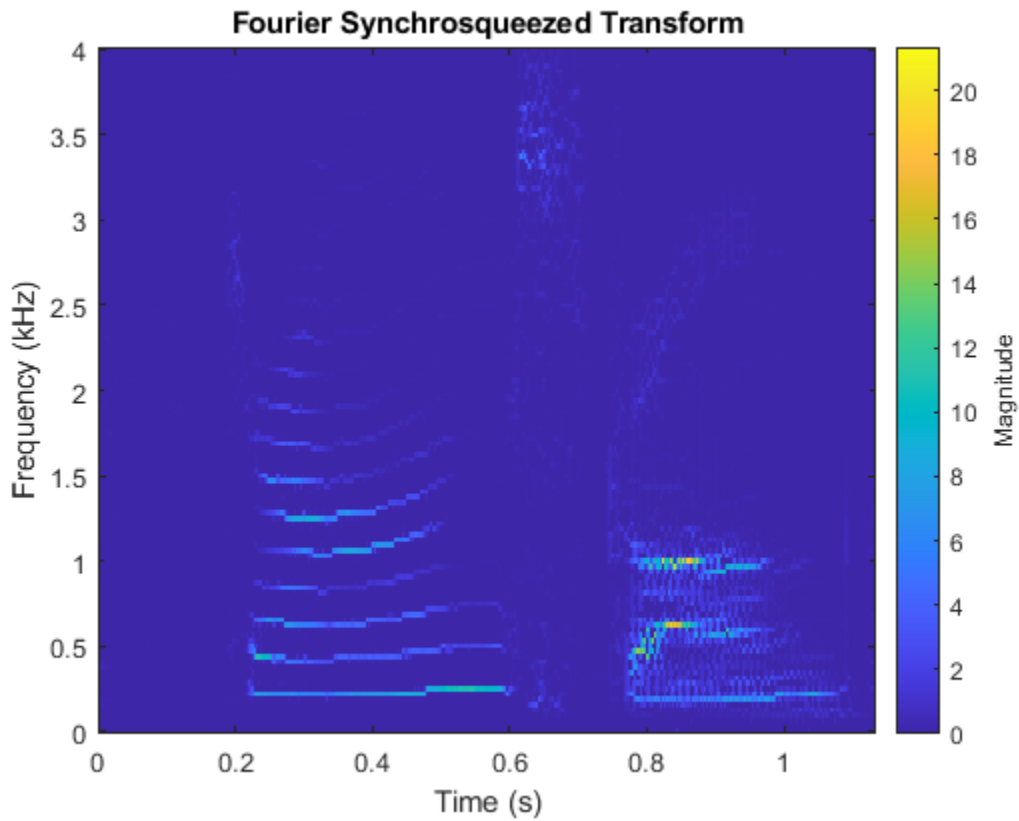
Example: Synthetic Seismic Data

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)
```
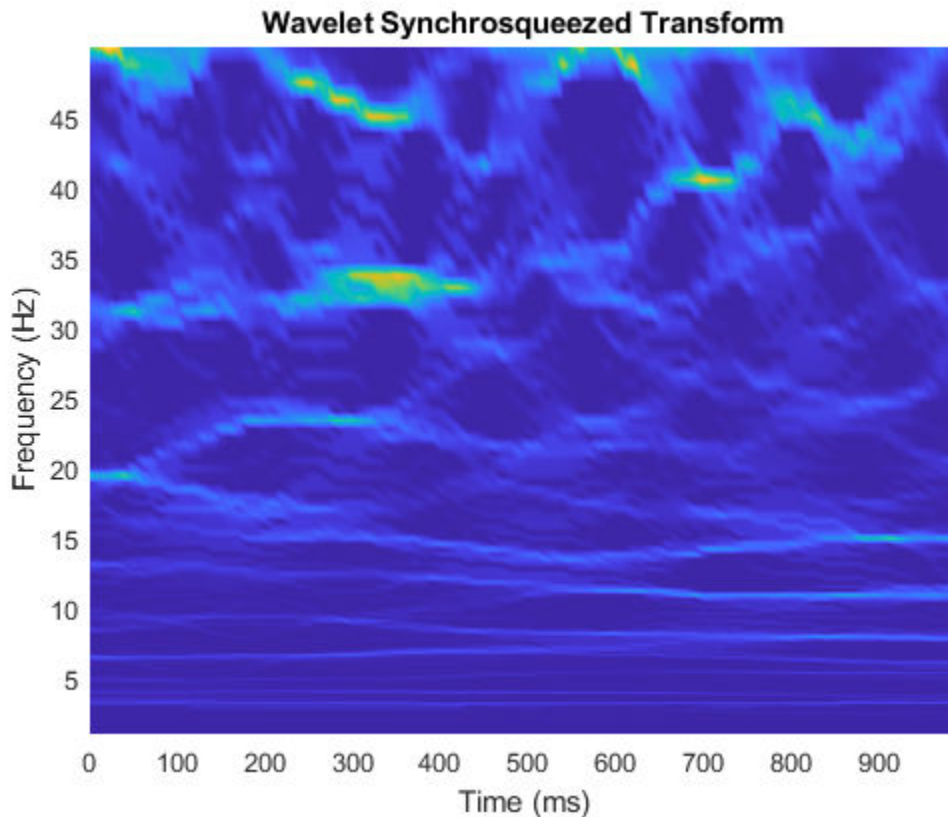
The seismic signal is generated using the two sinusoids mentioned in "Time-Frequency Analysis of Seismic Data Using Synchrosqueezing Transform" by Ping Wang, Jinghuai Gao, and Zhiguo Wang [4].

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(gfloor1OL,Fs,'bump','VoicesPerOctave',48)
ylim([0 35])
```
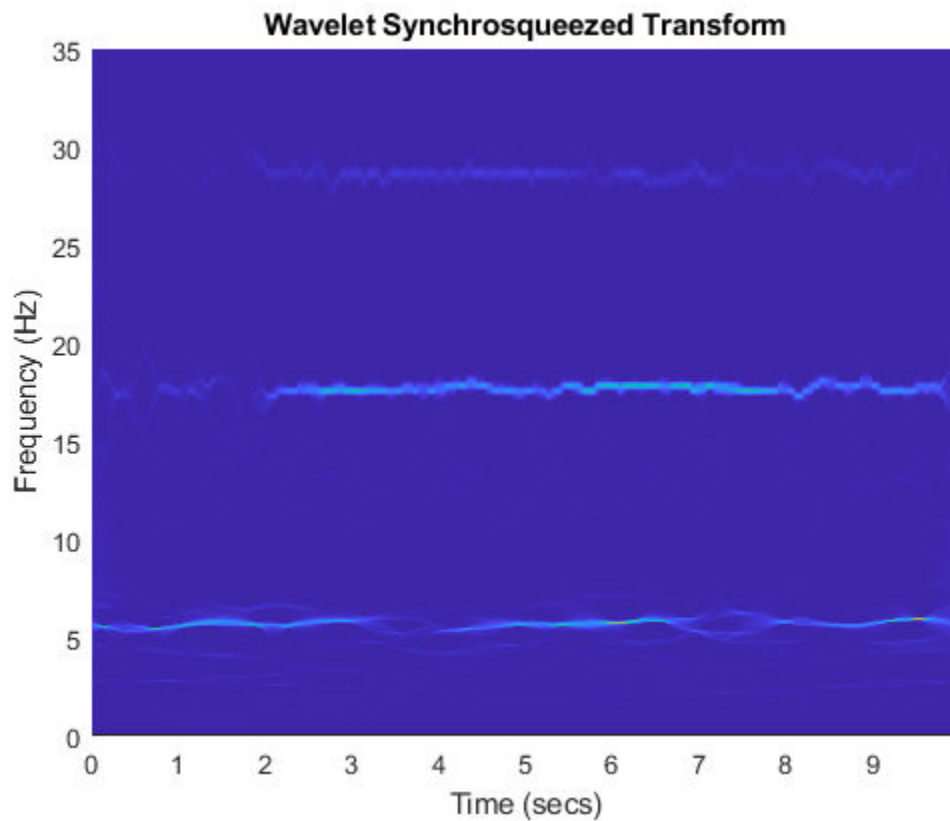
Example: Kobe Earthquake Data

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])
```
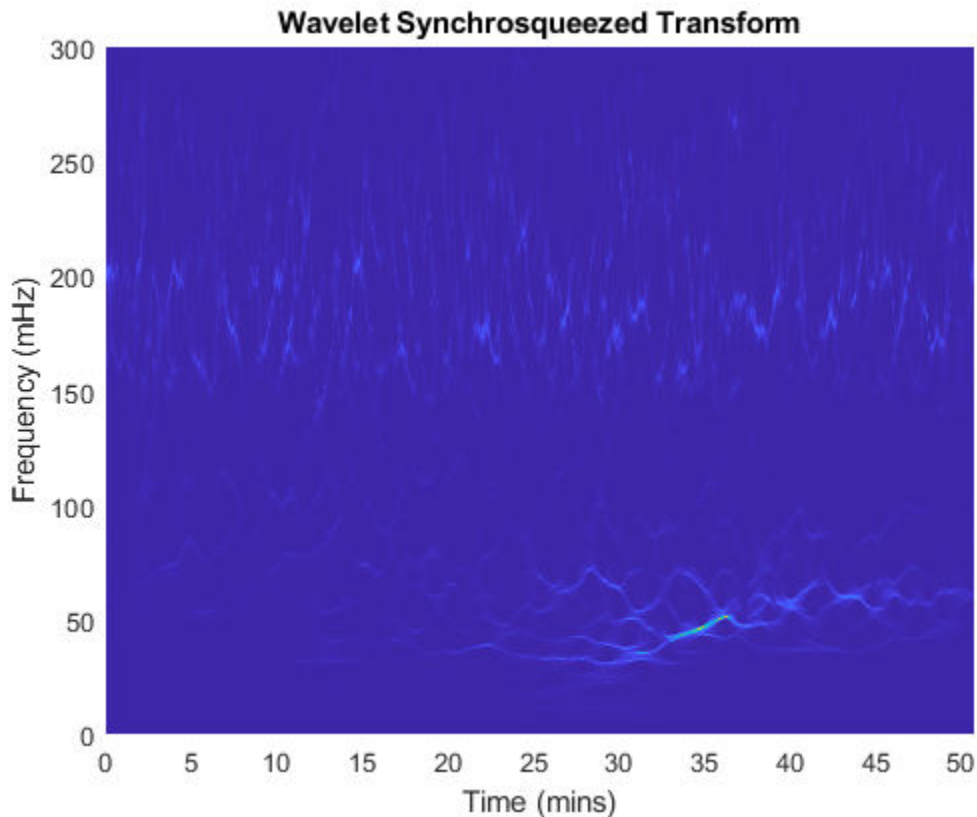
The data are seismograph (vertical acceleration, nm/sq.sec) measurements recorded at Tasmania University, Hobart, Australia on 16 January 1995 beginning at 20:56:51 (GMT) and continuing for 51 minutes at 1 second intervals [5].

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])
```

Wavelet Synchrosqueezed Transform
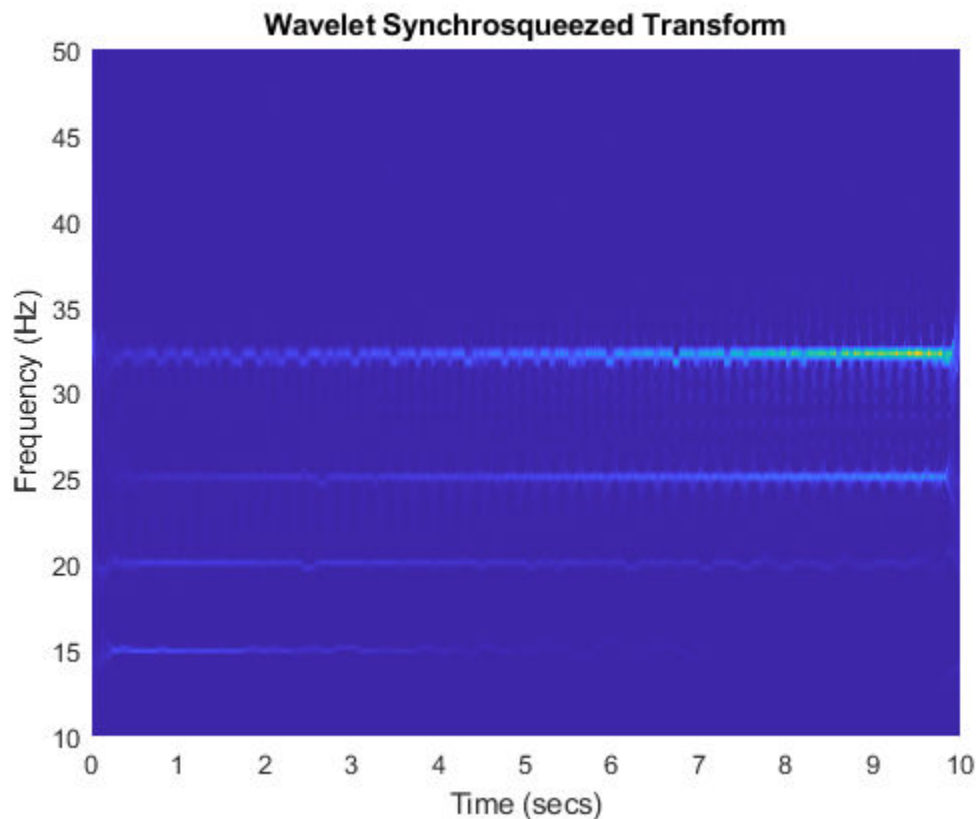
This synthetic subsynchronous oscillation data was generated using the equation defined by Zhao et al in "Application of Synchrosqueezed Wavelet Transforms for Extraction of the Oscillatory Parameters of Subsynchronous Oscillation in Power Systems" [6].

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])
```
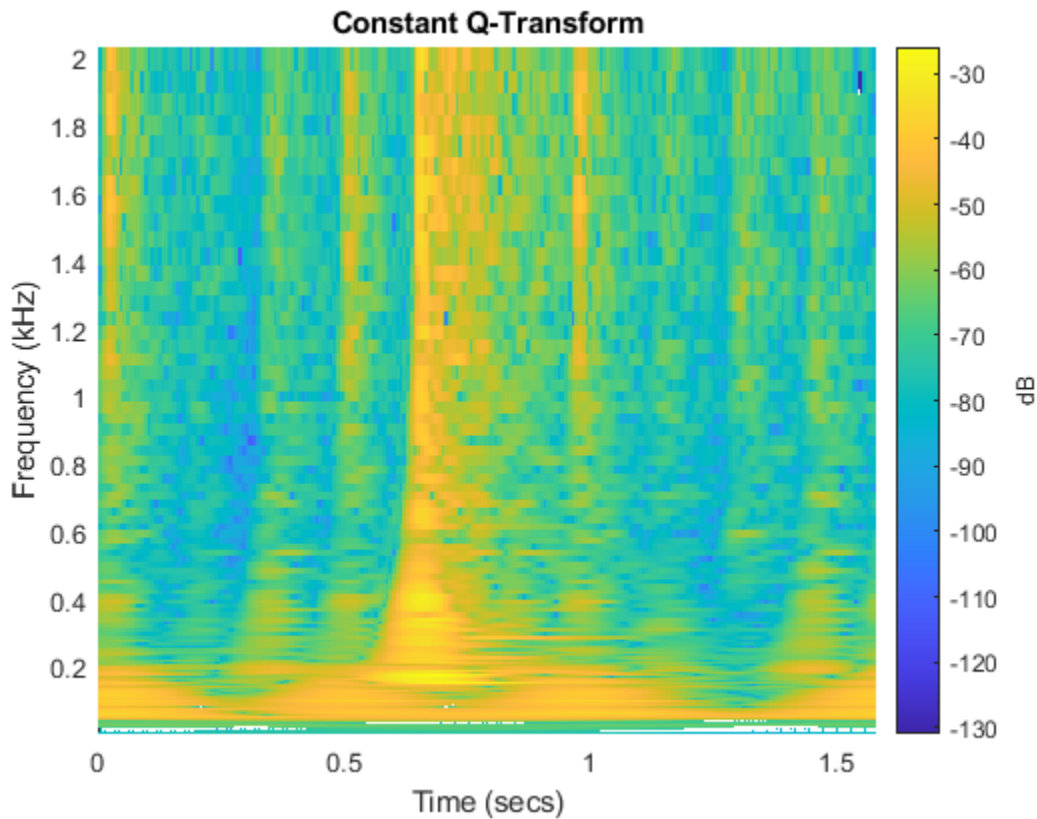

Empirical Mode Decomposition and Hilbert-Huang Transform

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])
```
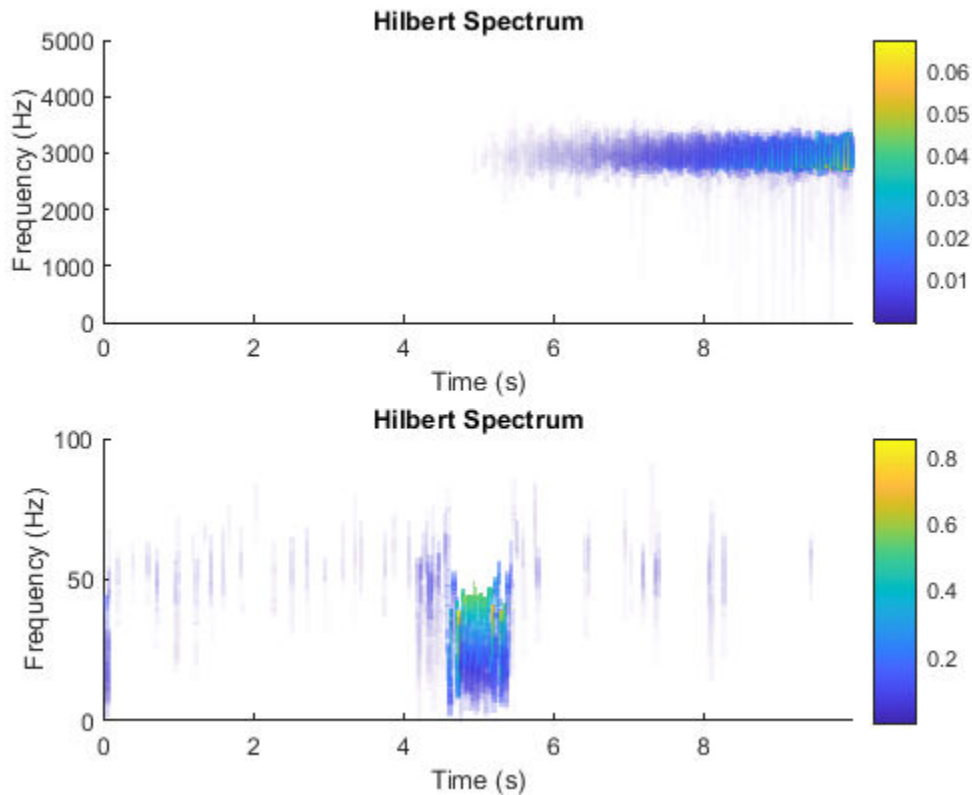
References

[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 | **Signal Multiresolution 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

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 5-23.

Some object-oriented programming features are used for wavelet packet tree structures. For more detail, refer to "Introduction to Object-Oriented Features" on page 5-38.

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 5-40.

Analysis-Decomposition Functions

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

Synthesis-Reconstruction Functions

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

Decomposition Structure Utilities

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

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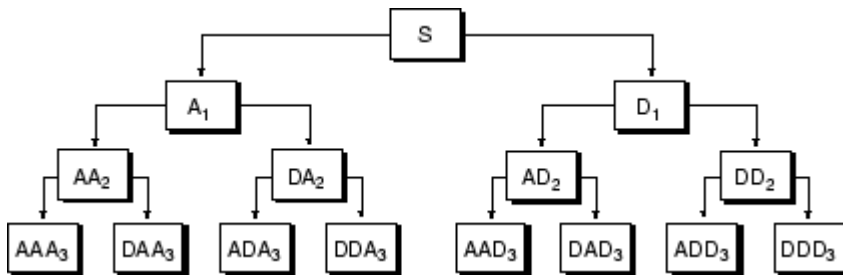
A single wavelet packet decomposition gives a lot of bases from which you can look for the best representation with respect to a design objective. This can be done by finding the "best tree" based on an entropy criterion.

Denoising and compression are interesting applications of wavelet packet analysis. The wavelet packet denoising 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 5-5
"Importing a Signal" on page 5-5
"Analyzing a Signal" on page 5-6
"Computing the Best Tree" on page 5-7
"Compressing a Signal Using Wavelet Packets" on page 5-8
"De-Noising a Signal Using Wavelet Packets" on page 5-10

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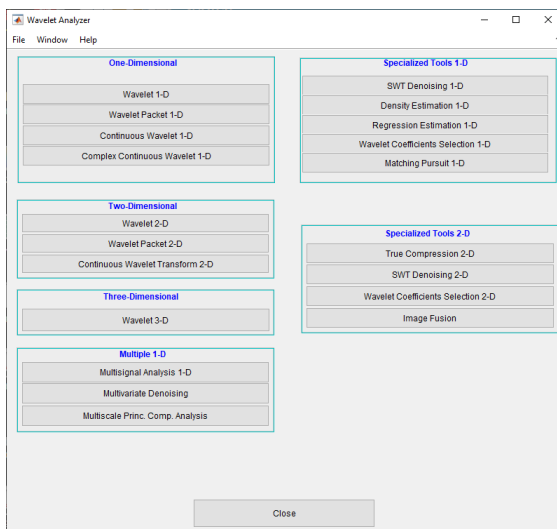
   Click the **Wavelet Packet 1-D** menu item.

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.
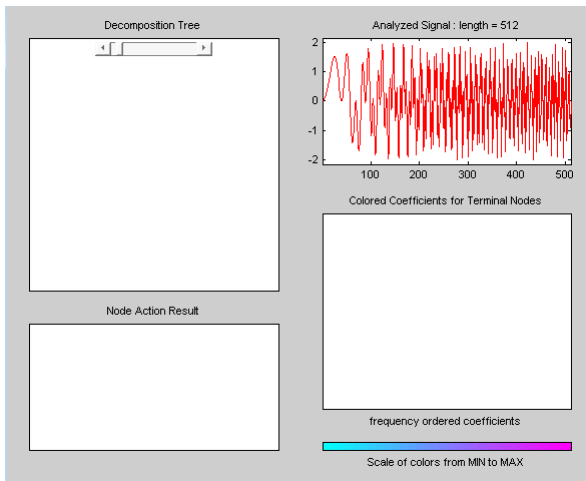
Analyzing a Signal

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.

The available entropy types are listed below.

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

Type	Description
SURE (Stein's Unbiased Risk Estimate)	A threshold-based method in which the threshold equals $$\sqrt{2\log_e(n\log_2(n))}$$ where n is the number of samples in the signal.
User	An entropy type criterion you define in 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 5-31.

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.
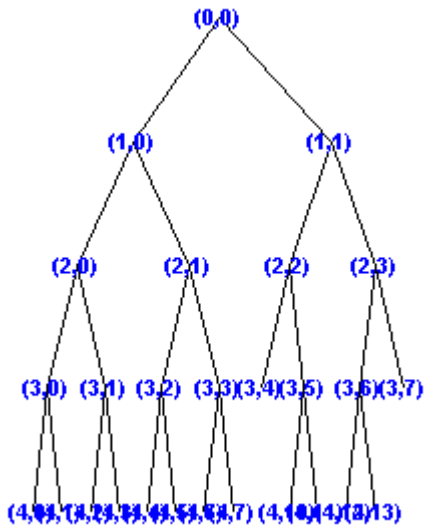
Compressing a Signal Using Wavelet Packets

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.
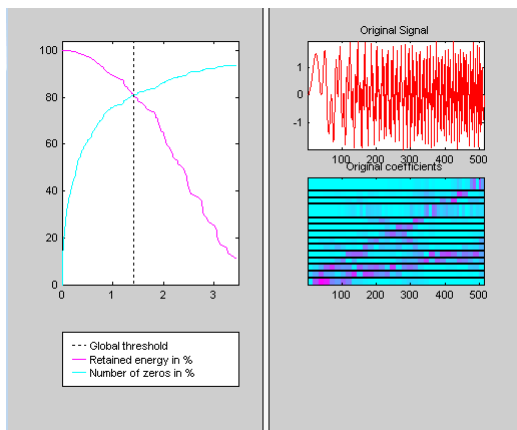
The leftmost graph shows how the threshold (vertical black dotted line) has been chosen automatically (1.482) to balance the number of zeros in the compressed signal (blue curve that increases as the threshold increases) with the amount of energy retained in the compressed signal (purple curve that decreases 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.
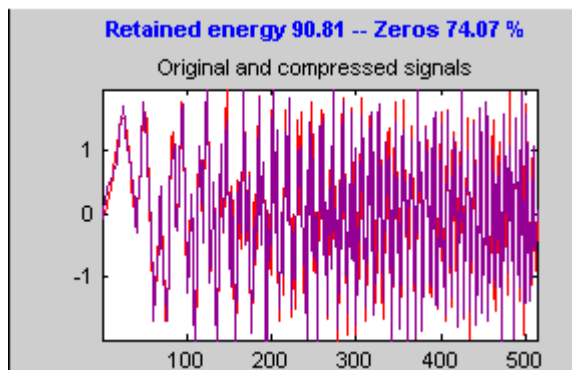
The original (red) and compressed () signals are displayed superimposed. Visual inspection suggests the compression quality is quite good.

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 5-31. 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.
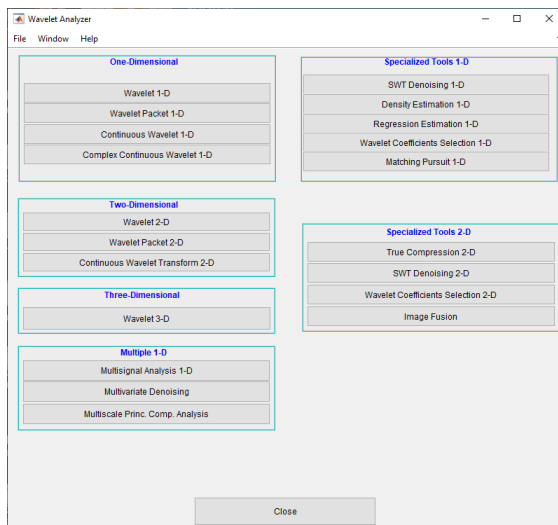
    Click the **Wavelet Packet 1-D** menu item.

    The tool appears on the desktop.
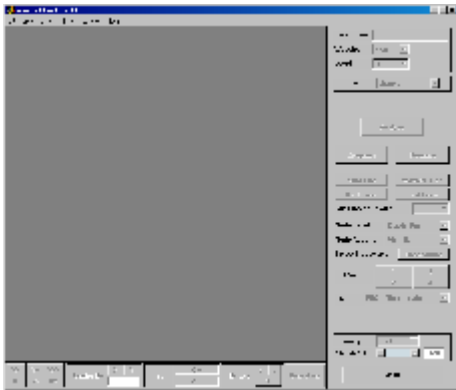
Importing a Signal

2 At the MATLAB command prompt, type

```
load noischir;
```

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.

3 The signal's length is 1024. This means we should set the SURE criterion threshold equal to `sqrt(2.*log(1024.*log2(1024)))`, or 4.2975.
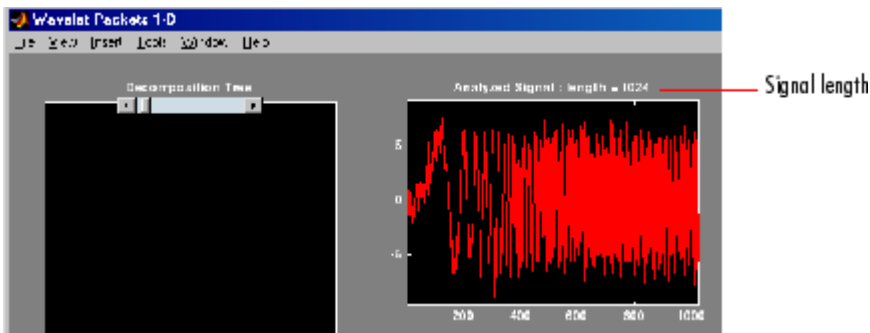
Analyzing a Signal

4 Make the appropriate settings for the analysis. Select the `db2` wavelet, level 4, entropy type `sure`, and threshold parameter 4.2975. Click the **Analyze** button.
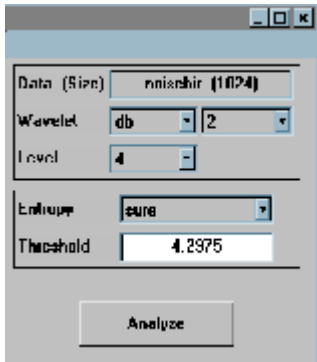
There is a pause while the wavelet packet analysis is computed.

Note Many capabilities are available using the command area on the right of the **Wavelet Packet 1-D** window. Some of them are used in the sequel. For a more complete description, see "Wavelet Packet Tool Features (1-D and 2-D)" on page A-11.

Computing the Best Tree and Performing De-Noising

5 Click the **Best Tree** button.

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

6 Click the **De-noise** button. This brings up the **Wavelet Packet 1-D De-Noising** window.
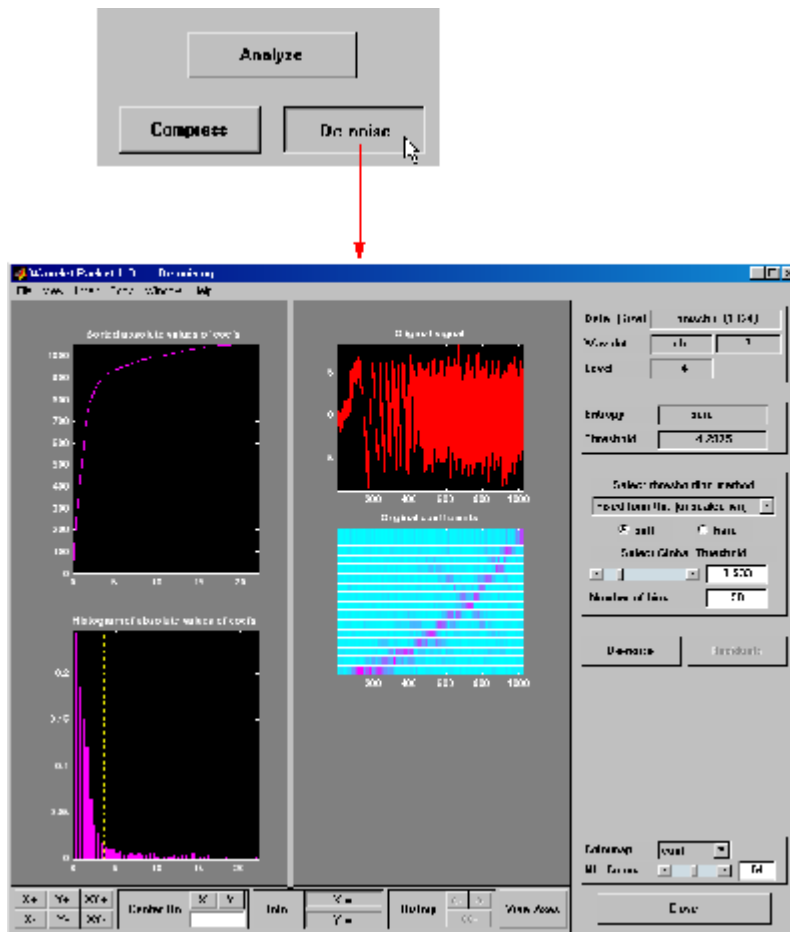
7 Click the **De-noise** button located at the center right side of the **Wavelet Packet 1-D De-Noising** window.

The results of the de-noising operation are quite good, as can be seen by looking at the thresholded coefficients. The frequency of the chirp signal increases quadratically over time, and the thresholded coefficients essentially capture the quadratic curve in the time-frequency plane.
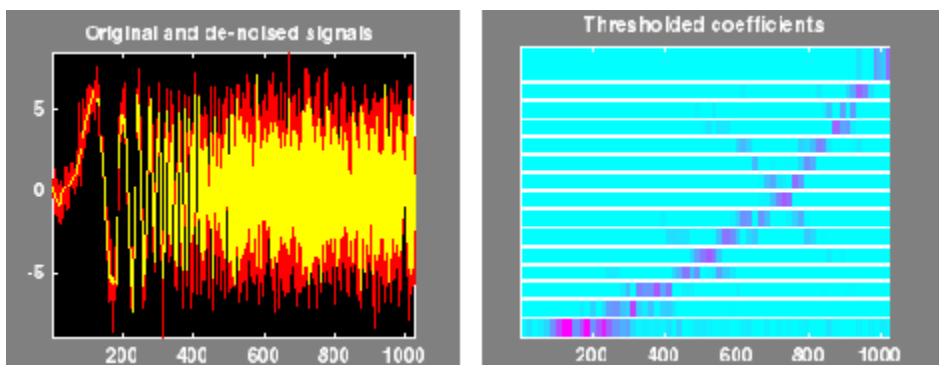
You can also use the `wpdencmp` function to perform wavelet packet de-noising or compression from the command line.

2-D Wavelet Packet Analysis

In this section...
"Starting the Wavelet Packet 2-D Tool" on page 5-14
"Compressing an Image Using Wavelet Packets" on page 5-15

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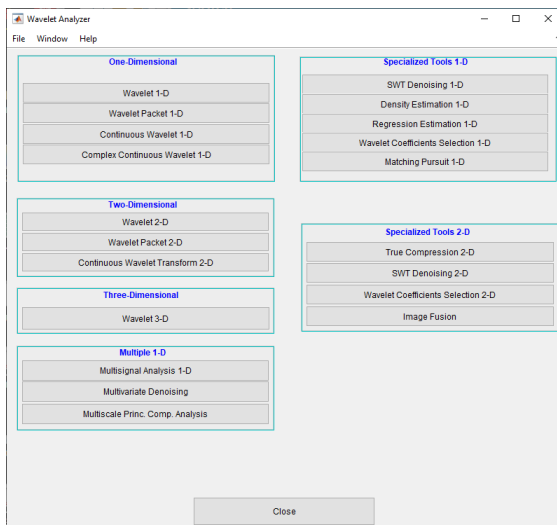
Click the **Wavelet Packet 2-D** menu item.

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.

3   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

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

Select thresholding method

Bal. sparsity-norm (sqrt)

Select Global Threshold

	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.

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

Setting all wavelet packet coefficients whose value falls below 30 to zero yields much better results. Note that the new threshold achieves around 92% of zeros, while still retaining nearly 98% of the image energy.

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

You can see the result obtained by wavelet packet coefficients thresholding and image reconstruction. The visual recovery is correct, but not perfect. The compressed image, shown side by side with the original, shows some artifacts.
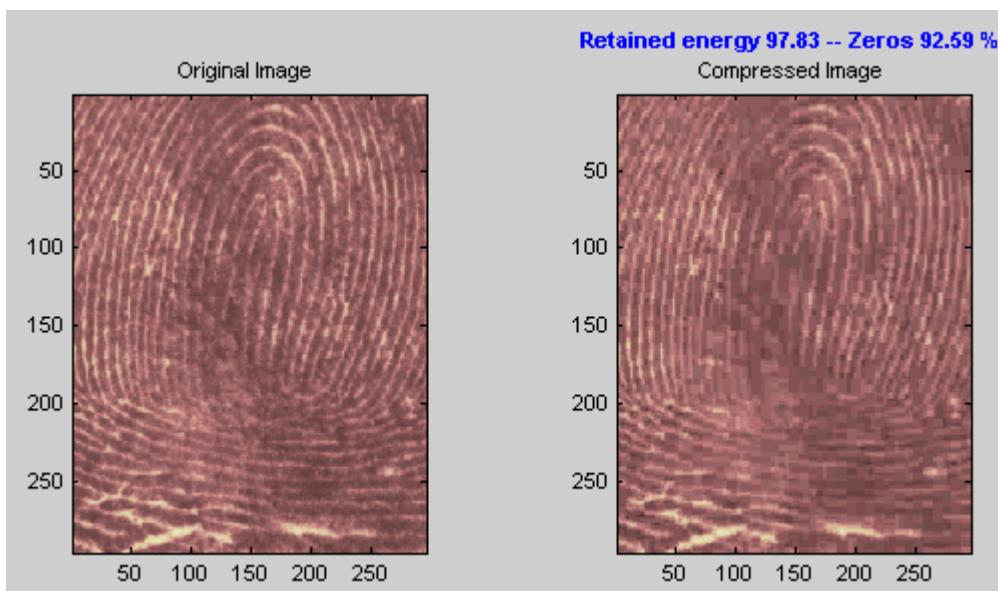
4 Click the **Close** button located at the bottom of the **Wavelet Packet 2-D Compression** window. Update the synthesized image by clicking **Yes** when the dialog box appears.

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

Hint The `bior6.8` wavelet is better suited to this analysis than is `haar`, and can lead to a better compression ratio. When a biorthogonal wavelet is used, then instead of "Retained energy" the

information displayed is "Energy ratio." For more information, see "Compression Scores" on page 6-64.

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.
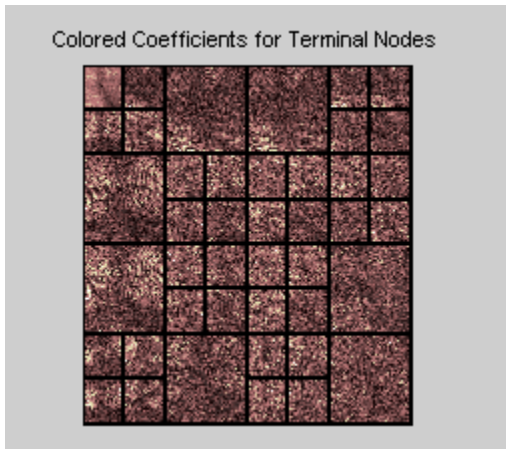
This plot is shown in the lower right side of the **Wavelet Packet 2-D** tool. The plot shows us which details have been decomposed and which have not. Larger squares represent details that have not been broken down to as many levels as smaller squares. Consider, for example, this level 2 decomposition pattern:
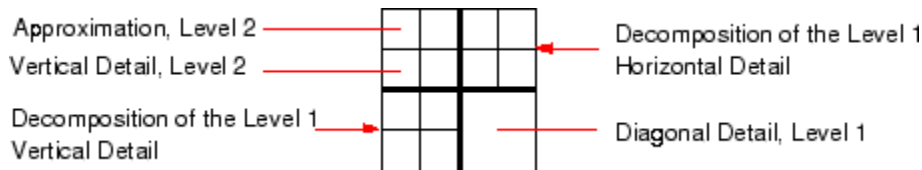
Looking at the pattern of small and large squares in the fingerprint analysis shows that the best tree algorithm has apparently singled out the diagonal details, often sparing these from further decomposition. Why is this?

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

Save options are also available when performing 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...

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.
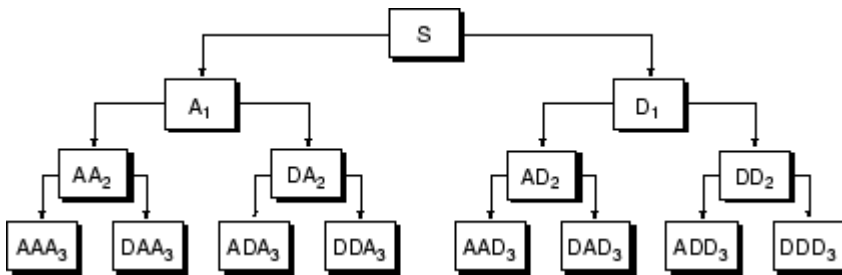
Wavelet Packet Decomposition Tree at Level 3

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;
windowsize = 128;
window = hanning(windowsize);
nfft = windowsize;
noverlap = windowsize-1;
[S,F,T] = spectrogram(y,window,noverlap,nfft,fs);
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')
```

Sum of two sine waves with frequencies of 64 and 128 hertz.

```
fs = 1000;
t = 0:1/fs:2;
y = sin(128*pi*t) + sin(256*pi*t); % sine of periods 64 and 128.
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');
```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```
figure;
windowsize = 128;
window = hanning(windowsize);
nfft = windowsize;
noverlap = windowsize-1;
[S,F,T] = spectrogram(y,window,noverlap,nfft,fs);
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')
```

Signal with an abrupt change in frequency from 16 to 64 hertz at two seconds.

```
fs = 500;
t = 0:1/fs:4;
y = sin(32*pi*t).*(t<2) + sin(128*pi*t).*(t>=2);
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');
```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```
figure;
windowsize = 128;
window = hanning(windowsize);
nfft = windowsize;
noverlap = windowsize-1;
[S,F,T] = spectrogram(y,window,noverlap,nfft,fs);
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')
```

Wavelet packet spectrum of a linear chirp.

```
fs = 1000;
t = 0:1/fs:2;
y = sin(256*pi*t.^2);
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');
```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```
figure;
windowsize = 128;
window = hanning(windowsize);
nfft = windowsize;
noverlap = windowsize-1;
[S,F,T] = spectrogram(y,window,noverlap,nfft,fs);
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')
```

Wavelet packet spectrum of quadratic chirp.

```
y = wnoise('quadchirp',10);
len = length(y);
t = linspace(0,5,len);
fs = 1/t(2);
level = 6;
wpt = wpdec(y,level,'sym8');
[Spec,Time,Freq] = wpspectrum(wpt,fs,'plot');
```

If you have the Signal Processing Toolbox software, you can compute the short-time Fourier transform.

```
windowsize = 128;
window = hanning(windowsize);
nfft = windowsize;
noverlap = windowsize-1;
imagesc(T,F,log10(abs(S)))
set(gca,'YDir','Normal')
xlabel('Time (secs)')
ylabel('Freq (Hz)')
title('Short-time Fourier Transform spectrum')
```

Building Wavelet Packets

The computation scheme for wavelet packets generation is easy when using an orthogonal wavelet. We start with the two filters of length $2N$, where $h(n)$ and $g(n)$, correspond to the wavelet.

Now by induction let us define the following sequence of functions:

$(W_n(x), n = 0, 1, 2, ...)$

by

$$W_{2n}(x) = \sqrt{2} \sum_{k=0}^{2N-1} h(k) W_n(2x - k)$$

$$W_{2n+1}(x) = \sqrt{2} \sum_{k=0}^{2N-1} g(k) W_n(2x - k)$$

where $W_0(x) = \varphi(x)$ is the scaling function and $W_1(x) = \psi(x)$ is the wavelet function.

For example for the Haar wavelet we have

$$N = 1, h(0) = h(1) = \frac{1}{\sqrt{2}}$$

and

$$g(0) = -g(1) = \frac{1}{\sqrt{2}}$$

The equations become

$$W_{2n}(x) = W_n(2x) + W_n(2x - 1)$$

and

$$W_{2n+1}(x) = W_n(2x) - W_n(2x - 1)$$

$W_0(x) = \varphi(x)$ is the *Haar* scaling function and $W_1(x) = \psi(x)$ is the Haar wavelet, both supported in [0, 1]. Then we can obtain W_{2n} by adding two 1/2-scaled versions of W_n with distinct supports [0,1/2] and [1/2,1] and obtain W_{2n+1} by subtracting the same versions of W_n.

For $n = 0$ to 7, we have the *W*-functions shown in the figure "Haar Wavelet Packets" on page 5-28.
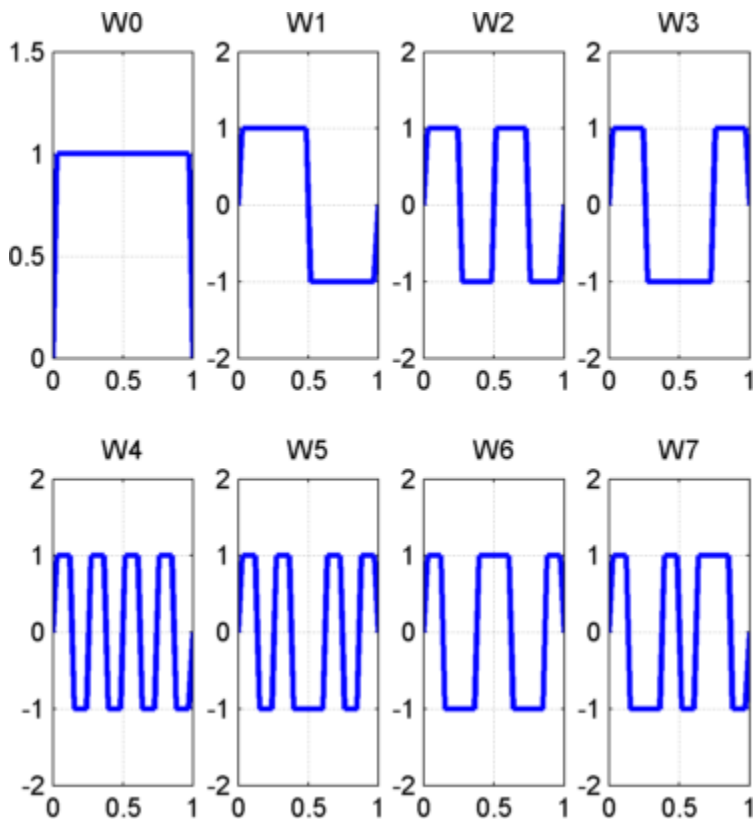
Haar Wavelet Packets

This can be obtained using the following command:

```
[wfun,xgrid] = wpfun('db1',7,5);
```

which returns in `wfun` the approximate values of W_n for $n = 0$ to 7, computed on a $1/2^5$ grid of the support `xgrid`.

Starting from more regular original wavelets and using a similar construction, we obtain smoothed versions of this system of *W*-functions, all with support in the interval [0, 2*N*–1]. The figure "db2 Wavelet Packets" on page 5-29 presents the system of *W*-functions for the original `db2` wavelet.
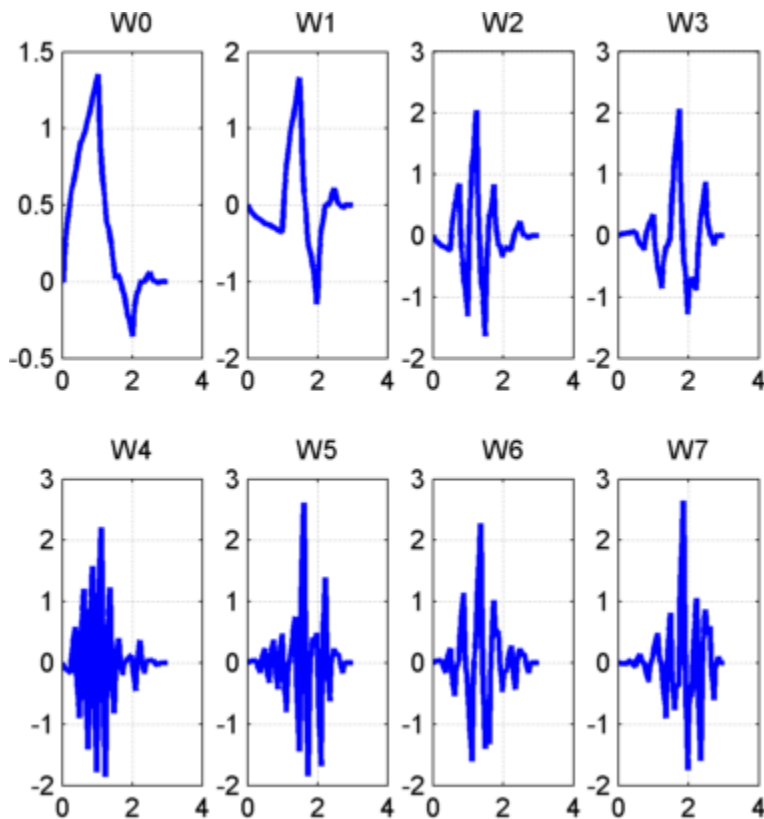
*db*2 **Wavelet Packets**

Wavelet Packet Atoms

Starting from the functions ($W_n(x), n \in N$) and following the same line leading to orthogonal wavelets, we consider the three-indexed family of analyzing functions (the waveforms):

$$(W_{j,n,k}(x) = 2^{-j/2}W_n(2^{-j}x - k)$$

where $n \in N$ 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 5-28 and "db2 Wavelet Packets" on page 5-29, the naturally ordered W_n for $n = 0, 1, ..., 7$, does not match exactly the order defined by the number of oscillations. More precisely, counting the number of zero crossings (up-crossings and down-crossings) for the db1 wavelet packets, we have the following.

Natural order n	0	1	2	3	4	5	6	7

Number of zero crossings for $db1\ W_n$	2	3	5	4	9	8	6	7

So, to restore the property that the main frequency increases monotonically with the order, it is convenient to define the *frequency order* obtained from the natural one recursively.

Natural order n	0	1	2	3	4	5	6	7
Frequency order r(n)	0	1	3	2	6	7	5	4

As can be seen in the previous figures, $W_{r(n)}(x)$ "oscillates" approximately n times.

To analyze a signal (the chirp of Example 2 for instance), it is better to plot the wavelet packet coefficients following the frequency order 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 $W_{j,n} = (W_{j,n,k}(x), k \in Z)$ 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 5-30 is created to give a maximum level decomposition equal to 3. For each scale j, the possible values of parameter n are 0, 1, ..., 2^j-1.
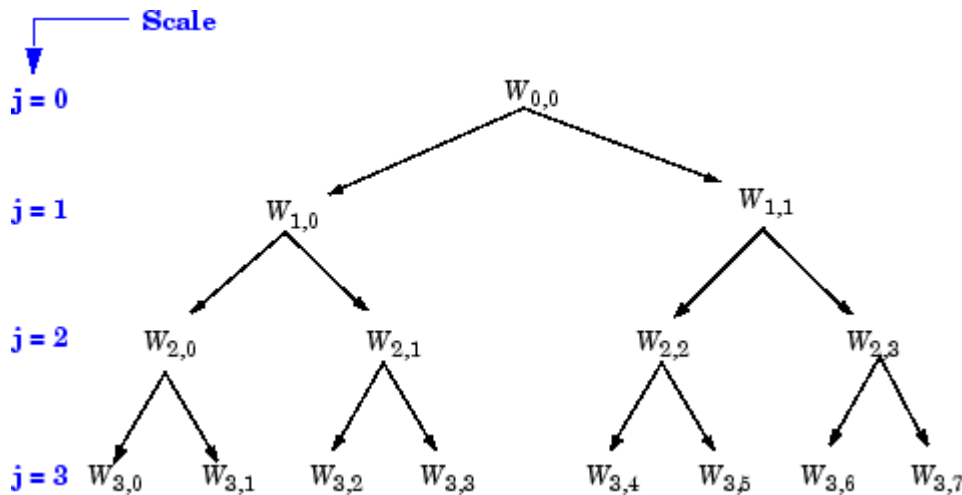
Wavelet Packets Organized in a Tree; Scale *j* Defines Depth and Frequency n Defines Position in the Tree

The notation W$_{j,n}$, where j denotes scale parameter and n the frequency parameter, is consistent with the usual depth-position tree labeling.

We have $W_{0,0} = (\phi(x - k), k \in Z)$, and $W_{1,1} = (\psi(\frac{x}{2} - k), k \in Z)$.

It turns out that the library of wavelet packet bases contains the wavelet basis and also several other bases. Let us have a look at some of those bases. More precisely, let V_0 denote the space (spanned by the family $W_{0,0}$) in which the signal to be analyzed lies; then $(W_{d,1}; d \geq 1)$ is an orthogonal basis of V_0.

For every strictly positive integer D, $(W_{D,0}, (W_{d,1}; 1 \leq d \leq D))$ is an orthogonal basis of V_0.

We also know that the family of functions $\{(W_{j+1,2n}), (W_{j+1,2n+1})\}$ is an orthogonal basis of the space spanned by $W_{j,n}$, which is split into two subspaces: $W_{j+1,2n}$ spans the first subspace, and $W_{j+1,2n+1}$ the second one.

This last property gives a precise interpretation of splitting in the wavelet packet organization tree, because all the developed nodes are of the form shown in the figure "Wavelet Packet Tree: Split and Merge" on page 5-31.
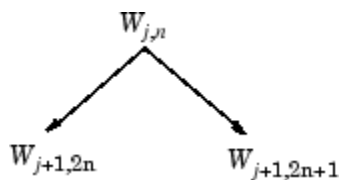
Wavelet Packet Tree: Split and Merge

It follows that the leaves of every connected binary subtree of the complete tree correspond to an orthogonal basis of the initial space.

For a finite energy signal belonging to V_0, any wavelet packet basis will provide exact reconstruction and offer a specific way of coding the signal, using information allocation in frequency scale subbands.

Choosing the Optimal Decomposition

Based on the organization of the wavelet packet library, it is natural to count the decompositions issued from a given orthogonal wavelet.

A signal of length $N = 2^L$ can be expanded in α different ways, where α is the number of binary subtrees of a complete binary tree of depth L. As a result, $\alpha \geq 2^{N/2}$ (see [Mal98] page 323).

As this number may be very large, and since explicit enumeration is generally unmanageable, it is interesting to find an optimal decomposition with respect to a convenient criterion, computable by an efficient algorithm. We are looking for a minimum of the criterion.

Functions verifying an additivity-type property are well suited for efficient searching of binary-tree structures and the fundamental splitting. Classical entropy-based criteria match these conditions and describe information-related properties for an accurate representation of a given signal. Entropy is a common concept in many fields, mainly in signal processing. Let us list four different entropy criteria (see [CoiW92]); many others are available and can be easily integrated (type `help wentropy`). In the following expressions s is the signal and (s_i) are the coefficients of s in an orthonormal basis.

The entropy E must be an additive cost function such that $E(0) = 0$ and

$$E(s) = \sum_i E(s_i)$$

- The (nonnormalized) Shannon entropy

$$E1(s_i) = -s_i^2 \log(s_i^2)$$

so

$$E1(s) = -\sum_i s_i^2 \log(s_i^2)$$

with the convention $0\log(0) = 0$.
- The concentration in l^p norm with $1 \, \Re \leq p$

$$E2(s_i) = |s_i|^p$$

so

$$E2(s) = \sum_i |s_i|^p = |s|_p^p$$

- The logarithm of the "energy" entropy

$$E3(s_i) = \log(s_i^2)$$

so

$$E3(s) = \sum_i \log(s_i^2)$$

with the convention $\log(0) = 0$.
- The threshold entropy

$E4(s_i) = 1$ if $|s_i| > \varepsilon$ and 0 elsewhere, so $E4(s) = \#\ \{i$ such that $|s_i| > \varepsilon\}$ is the number of time instants when the signal is greater than a threshold ε.

These entropy functions are available using the `wentropy` file.

Example 1: Compute Various Entropies

1 Generate a signal of energy equal to 1.

```
s = ones(1,16)*0.25;
```

2 Compute the Shannon entropy of s.

```
e1 = wentropy(s,'shannon')
    e1 = 2.7726
```

3 Compute the $l^{1.5}$ entropy of s, equivalent to `norm(s,1.5)`$^{1.5}$.

```
e2 = wentropy(s,'norm',1.5)
    e2 = 2
```

4 Compute the "log energy" entropy of s.

```
e3 = wentropy(s,'log energy')
    e3 = -44.3614
```

5 Compute the threshold entropy of s using a threshold value of 0.24.

```
e4 = wentropy(s,'threshold', 0.24)
    e4 = 16
```

Example 2: Minimum-Entropy Decomposition

This simple example illustrates the use of entropy to determine whether a new splitting is of interest to obtain a minimum-entropy decomposition.

1 We start with a constant original signal. Two pieces of information are sufficient to define and to recover the signal (i.e., length and constant value).

```
w00 = ones(1,16)*0.25;
```

2 Compute entropy of original signal.

```
e00 = wentropy(w00,'shannon')
    e00 = 2.7726
```

3 Then split w00 using the haar wavelet.

```
[w10,w11] = dwt(w00,'db1');
```

4 Compute entropy of approximation at level 1.

```
e10 = wentropy(w10,'shannon')
    e10 = 2.0794
```

The detail of level 1, w11, is zero; the entropy e11 is zero. Due to the additivity property the entropy of decomposition is given by e10+e11=2.0794. This has to be compared to the initial entropy e00=2.7726. We have e10 + e11 < e00, so the splitting is interesting.

5 Now split w10 (not w11 because the splitting of a null vector is without interest since the entropy is zero).

```
[w20,w21] = dwt(w10,'db1');
```

6 We have w20=0.5*ones(1,4) and w21 is zero. The entropy of the approximation level 2 is

```
e20 = wentropy(w20,'shannon')
    e20 = 1.3863
```

Again we have e20 + 0 < e10, so splitting makes the entropy decrease.

7 Then

```
[w30,w31] = dwt(w20,'db1');
e30 = wentropy(w30,'shannon')
    e30 = 0.6931

[w40,w41] = dwt(w30,'db1')
    w40 = 1.0000
    w41 = 0

e40 = wentropy(w40,'shannon')
    e40 = 0
```

In the last splitting operation we find that only one piece of information is needed to reconstruct the original signal. The wavelet basis at level 4 is a best basis according to Shannon entropy (with null optimal entropy since e40+e41+e31+e21+e11 = 0).

8 Perform wavelet packets decomposition of the signal *s* defined in example 1.

```
t = wpdec(s,4,'haar','shannon');
```

The wavelet packet tree in "Entropy Values" on page 5-34 shows the nodes labeled with original entropy numbers.

Entropy Values

9   Compute the best tree.

```
bt = besttree(t);
```

The best tree is shown in the following figure. In this case, the best tree corresponds to the wavelet tree. The nodes are labeled with optimal entropy.

Optimal Entropy Values

Some Interesting Subtrees

Using wavelet packets requires tree-related actions and labeling. The implementation of the user interface is built around this consideration. For more information on the technical details, see the reference pages.

The complete binary tree of depth D corresponding to a wavelet packet decomposition tree developed at level D is denoted by WPT.

We have the following interesting subtrees.

Decomposition Tree	Subtree Such That the Set of Leaves Is a Basis
Wavelet packets decomposition tree	Complete binary tree: WPT of depth D
Wavelet packets optimal decomposition tree	Binary subtree of WPT

Decomposition Tree	Subtree Such That the Set of Leaves Is a Basis
Wavelet packets best-level tree	Complete binary subtree of WPT
Wavelet decomposition tree	Left unilateral binary subtree of WPT of depth D
Wavelet best-basis tree	Left unilateral binary subtree of WPT

We deduce the following definitions of optimal decompositions, with respect to an entropy criterion E.

Decompositions	Optimal Decomposition	Best-Level Decomposition
Wavelet packet decompositions	Search among 2^D trees	Search among D trees
Wavelet decompositions	Search among D trees	Search among D trees

For any nonterminal node, we use the following basic step to find the optimal subtree with respect to a given entropy criterion E (where $Eopt$ denotes the optimal entropy value).

Entropy Condition	Action on Tree and on Entropy Labeling
$E(node) \leq \sum\limits_{c \text{ child of node}} Eopt(c)$	If ($node \neq root$), merge and set $Eopt(node) = E(node)$
$E(node) > \sum\limits_{c \text{ child of node}} Eopt(c)$	Split and set $Eopt(node) = \sum\limits_{c \text{ child of node}} Eopt(c)$

with the natural initial condition on the reference tree, $Eopt(t) = E(t)$ for each terminal node t.

Reconstructing a Signal Approximation from a Node

You can use the function `wprcoef` to reconstruct an approximation to your signal from any node in the wavelet packet tree. This is true irrespective of whether you are working with a full wavelet packet tree, or a subtree determined by an optimality criterion. Use `wpcoef` if you want to extract the wavelet packet coefficients from a node without reconstructing an approximation to the signal.

Load the noisy Doppler signal.

```
load noisdopp
```

Compute the wavelet packet decomposition down to level 5 using the `sym4` wavelet. Use the periodization mode.

```
dwtmode('per');
T = wpdec(noisdopp,5,'sym4');
plot(T)
```

Plot the binary wavelet packet tree and click on the (4,1) doublet (node 16).
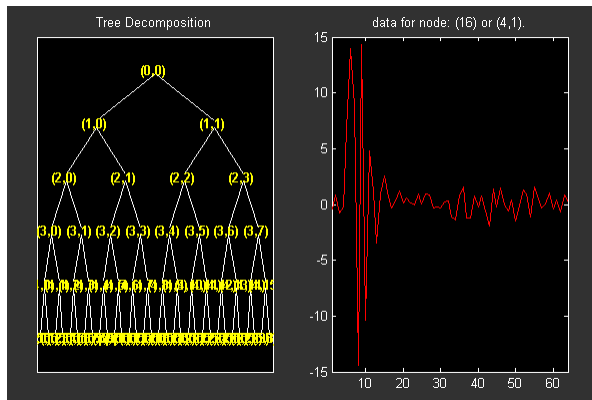
Extract the wavelet packet coefficients from node 16.

```
wpc = wpcoef(T,16);
% wpc is length 64
```

Obtain an approximation to the signal from node 16.

```
rwpc = wprcoef(T,16);
% rwpc is length 1024
plot(noisdopp,'k'); hold on;
plot(rwpc,'b','linewidth',2);
axis tight;
```
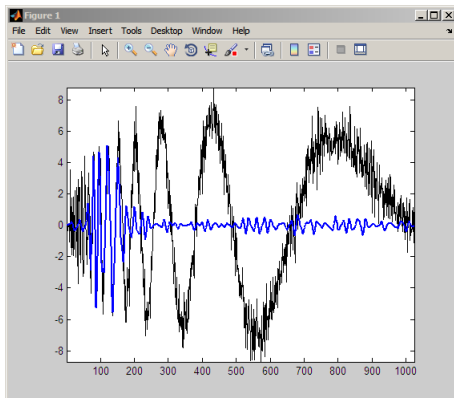

Determine the optimum binary wavelet packet tree.

```
Topt = besttree(T);
% plot the best tree
plot(Topt)
```

Reconstruct an approximation to the signal from the (3,0) doublet (node 7).

```
rsig = wprcoef(Topt,7);
% rsig is length 1024
plot(noisdopp,'k'); hold on;
plot(rsig,'b','linewidth',2);
axis tight;
```
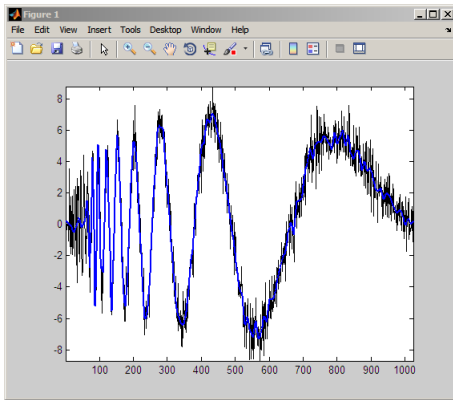
If you know which doublet in the binary wavelet packet tree you want to extract, you can determine the node corresponding to that doublet with `depo2ind`.

For example, to determine the node corresponding to the doublet (3,0), enter:

```
Node = depo2ind(2,[3 0]);
```

Wavelet Packets 2-D Decomposition Structure

Exactly as in the wavelet decomposition case, the preceding 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.
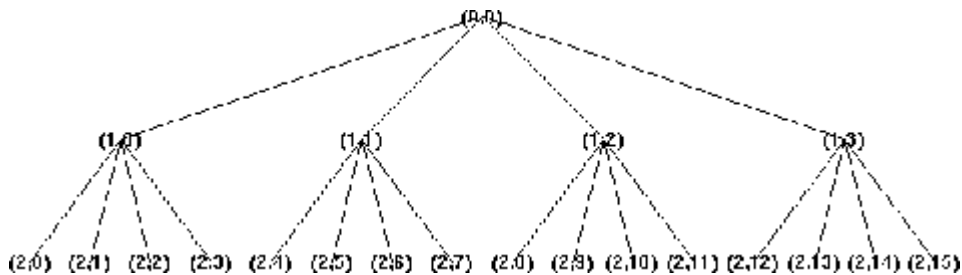
Quaternary Tree of Depth 2

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 5-40
- "drawtree and readtree" on page 5-42
- "Change Terminal Node Coefficients" on page 5-44
- "Thresholding Wavelet Packets" on page 5-45

plot and wpviewcf

```
load noisbump
x = noisbump;
t = wpdec(x,3,'db2');
fig = plot(t);
```

Click on node 7.
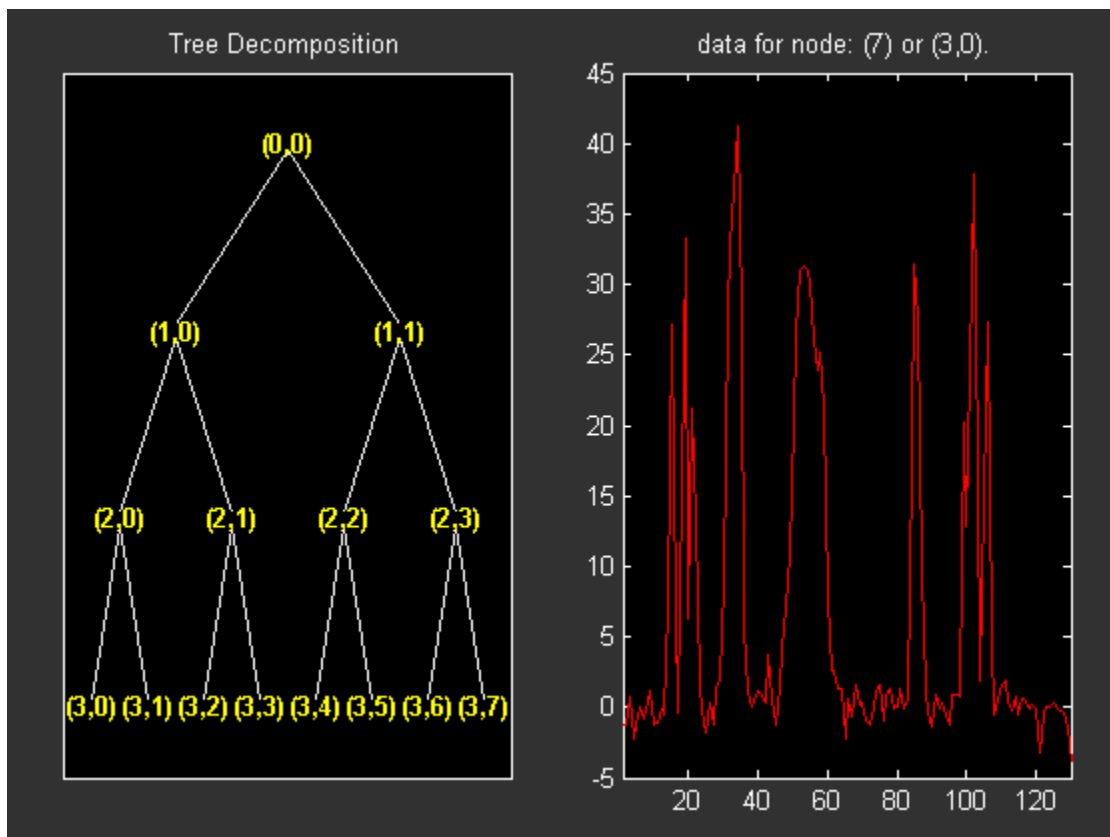
Change Node Action from **Visualize** to **Split-Merge** and merge the second node.
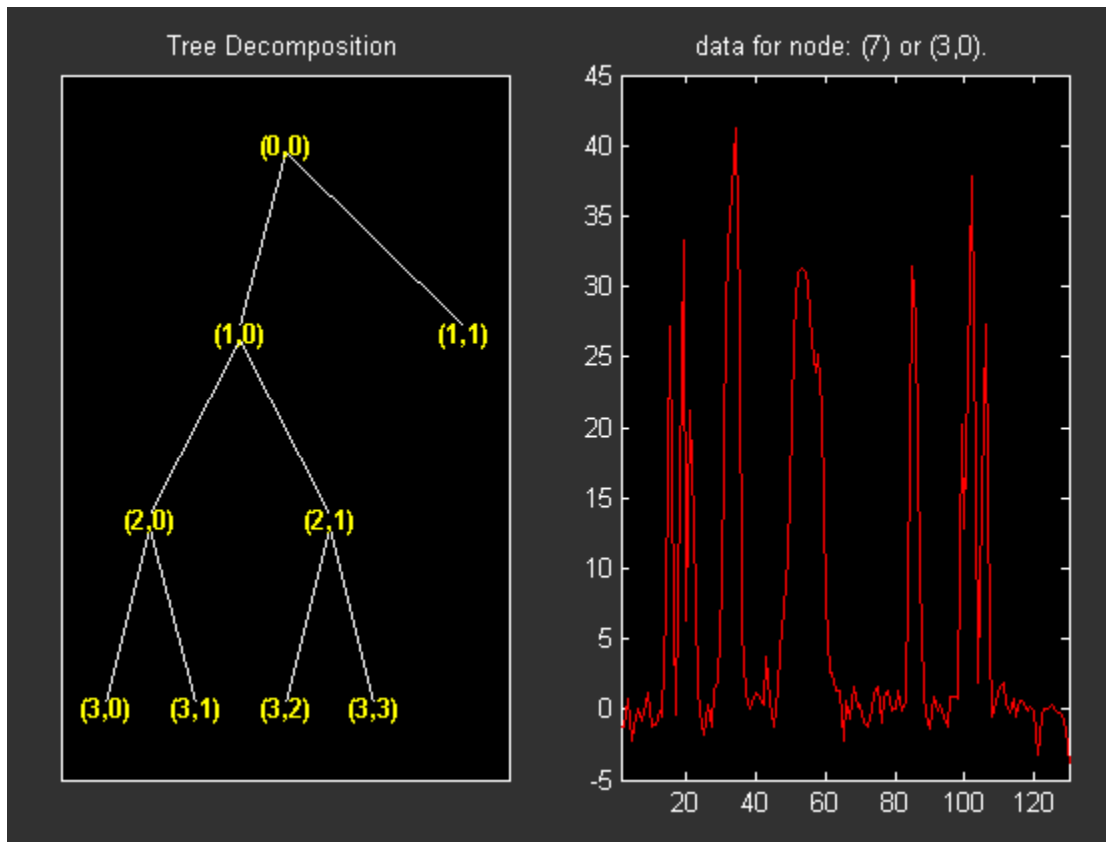

```
% From the command line, you can get the new tree.
newt = plot(t,'read',fig);

% 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.
```
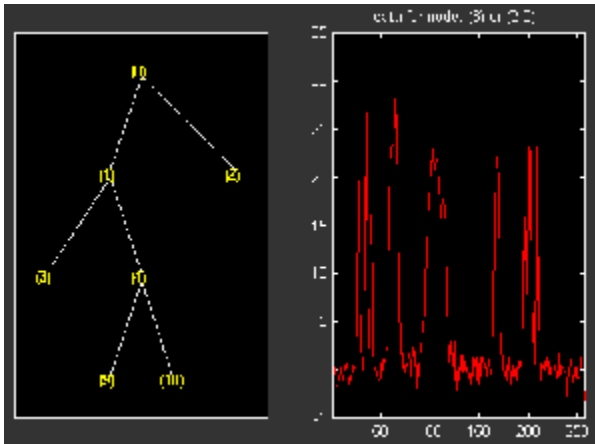
```
% Using plot(newt,fig), the plot is done in the figure fig,
% which already contains a tree object.

% 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.
```
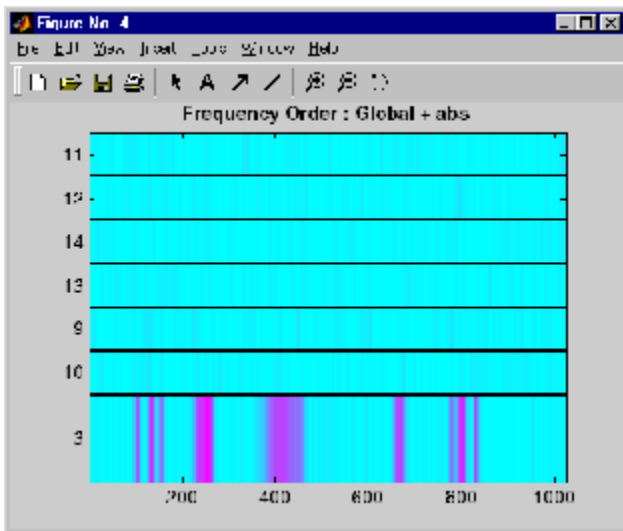

drawtree and readtree

```
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.
```
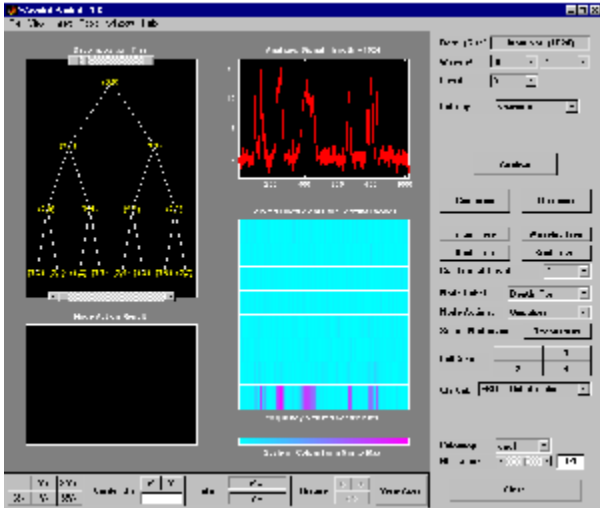


```
% From the app, you can modify the tree.
% For example, change Node label from Depth_Position to Index,
% change Node Action from Visualize to Split_Merge and
% merge the node 2.
% You get the following figure.
```
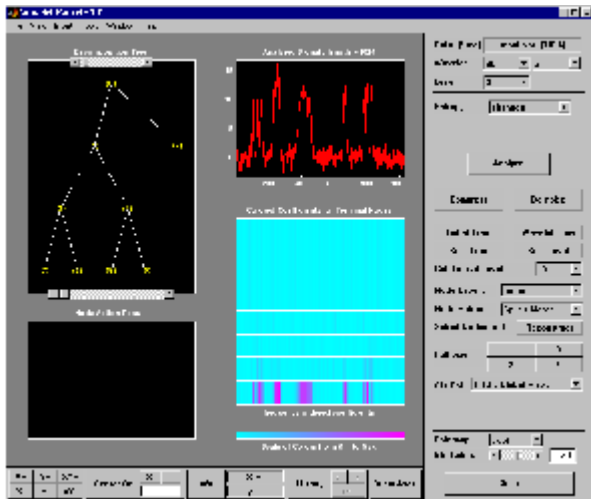


```
% From the command line, you can get the new tree.
newt = readtree(fig);

% From the command line you can modify the new tree;
% then plot it in the same figure.
newt = wpjoin(newt,3);
drawtree(newt,fig);
```
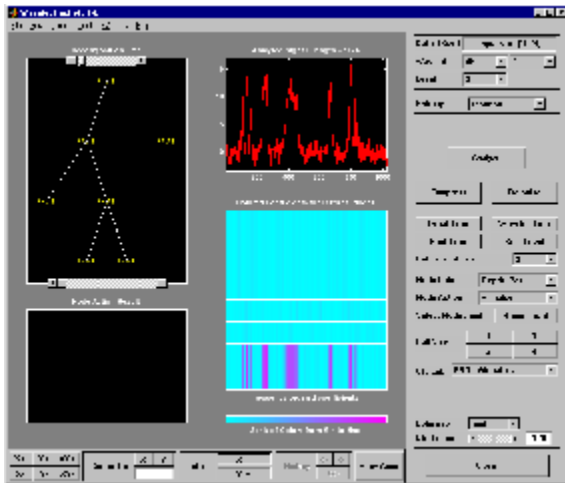
You can mix previous commands. The GUI associated with the `plot` command is simpler and quicker, but more actions and information are available using portions of the Wavelet Analyzer app related to wavelet packets.

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 gatlin2
t = wpdec2(X,1,'haar');
plot(t);
% Change Node Label from Depth_position to Index and
% click the node (0). You get the following figure.
```
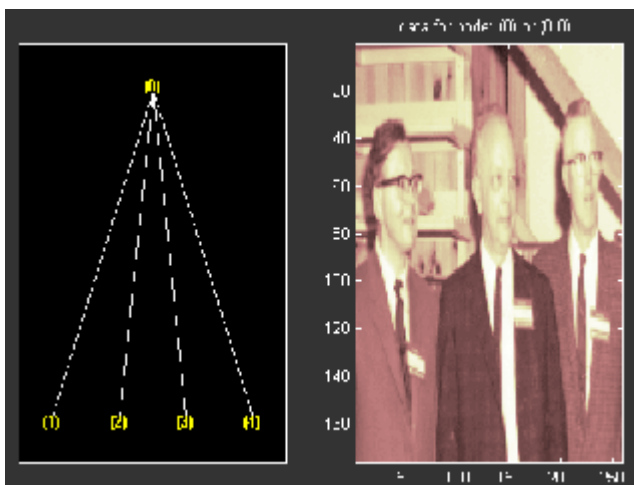


```
% Now modify the coefficients of the four terminal nodes.
newt = t;
NBcols = 40;
```

```
for node = 1:4
  cfs = read(t,'data',node);
  tmp = cfs(1:end,1:NBcols);
  cfs(1:end,1:NBcols) = cfs(1:end,end-NBcols+1:end);
  cfs(1:end,end-NBcols+1:end) = tmp;
  newt = write(newt,'data',node,cfs);
end
plot(newt)

% Change Node Label from Depth_position to Index and
% click on the node (0). You get the following figure.
```
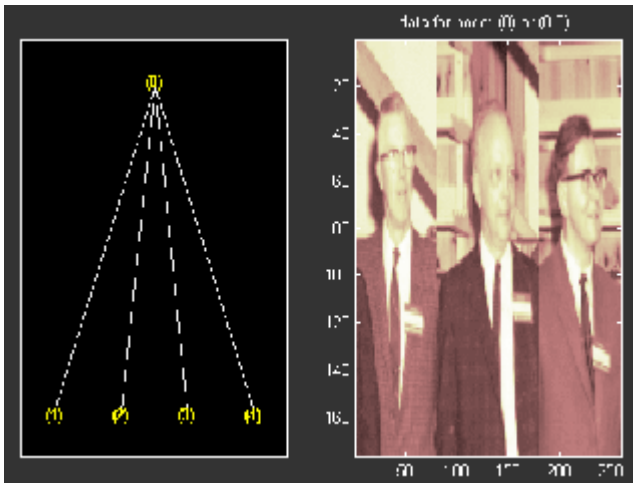

You can use this method for a more useful purpose. Let's see a denoising example.

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.
```
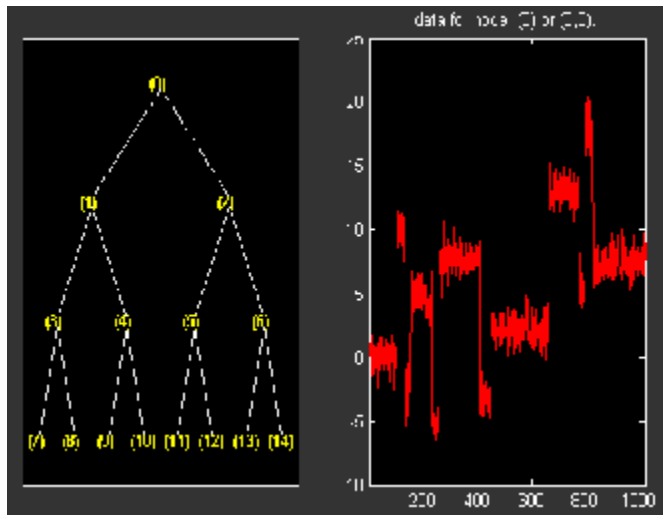
```
% Global thresholding.
t1 = t;
sorh = 'h';
thr = wthrmngr('wp1ddenoGBL','penalhi',t);
cfs = read(t,'data');
cfs = wthresh(cfs,sorh,thr);
t1  = write(t1,'data',cfs);
plot(t1)

% Change Node Label from Depth_position to Index and
% click the node (0). You get the following plot.
```
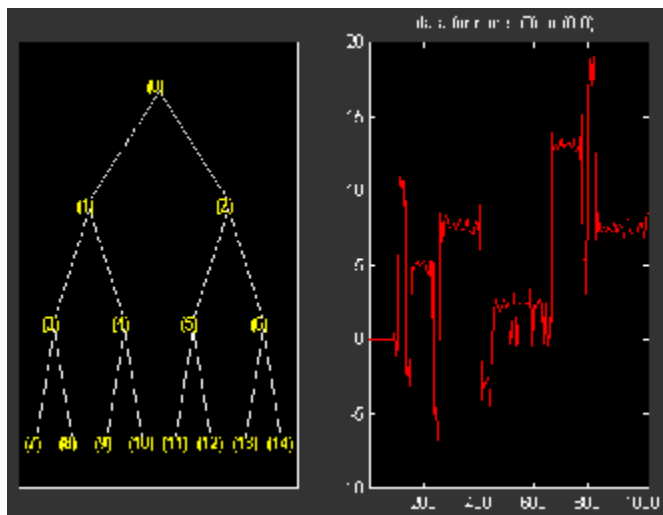


```
% Node by node thresholding.
t2 = t;
sorh = 's';
thr(1) = wthrmngr('wp1ddenoGBL','penalhi',t);
thr(2) = wthrmngr('wp1ddenoGBL','sqtwologswn',t);
tn  = leaves(t);
for k=1:length(tn)
  node = tn(k);
```

```
  cfs = read(t,'data',node);
  numthr = rem(node,2)+1;
  cfs = wthresh(cfs,sorh,thr(numthr));
  t2 = write(t2,'data',node,cfs);
end
plot(t2)

% Change Node Label from Depth_position to Index and
% click the node (0). You get the following plot.
```
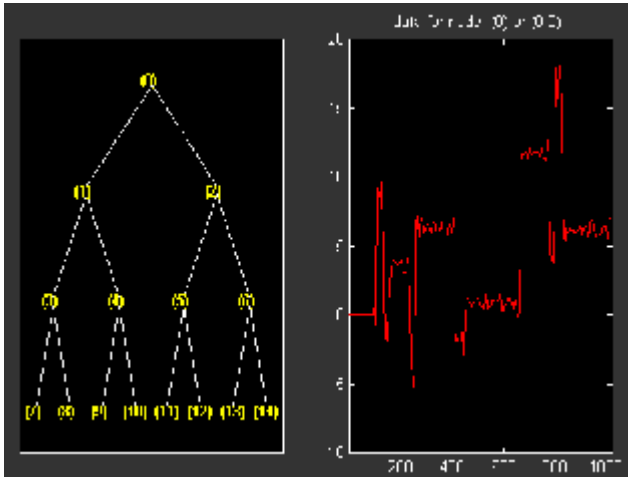
Description of Objects in the Wavelet Toolbox Software

The following sections describe the objects in the Wavelet Toolbox software:

- "WTBO Object" on page 5-48
- "NTREE Object" on page 5-48
- "DTREE Object" on page 5-49
- "WPTREE Object" on page 5-50

WTBO Object

Class **WTBO** (Wavelet Toolbox Object) -- Parent class: none

Fields

wtboInfo	Object information (Not used)
ud	Userdata field

Methods

wtbo	Constructor for the class WTBO.
get	Get WTBO object field contents.
set	Set WTBO object field contents.

Comments

Since any object in the toolbox is parented by a WTBO object, you can associate your own data to an object using the `'ud'` field, and then access it.

If `Obj` is an object (parented by a WTBO object), use

```
Obj = set(Obj,'ud',MyData)
```

to define the data.

To retrieve the data, use

```
MyData = get(O,'ud')
```

NTREE Object

Class **NTREE** (New Tree) -- Parent class: **WTBO**

Fields

wtbo	Parent object
order	Tree order
depth	Tree depth
spsch	Split scheme for nodes

tn	Column vector with terminal nodes indices

Methods

ntree	Constructor for the class NTREE.
findactn	Find active nodes.
get	Get NTREE object field contents.
nodejoin	Recompose node(s).
nodesplt	Split (decompose) node(s).
plot	Plot NTREE object.
set	Set NTREE object field contents.
tlabels	Labels for the nodes of a tree.
wtreemgr	Manager for NTREE object.

Private

locnumcn	Local number for a child node
tabofasc	Table of ascendants of nodes

DTREE Object

Class **DTREE** (Data Tree) -- Parent class: **NTREE**

Fields

ntree	Parent object
allNI	All Nodes Information
terNI	Terminal Nodes Information

Fields Description

allNI is a NBnodes-by-3 array such that

allNI(N,:) = [ind,size(1,1),size(1,2)]

- ind = index of the node N
- size = size of data associated with the node N

terNI is a 1-by-2 cell array such that

- terNI{1} is an NB_TerminalNodes-by-2 array such that

    - terNI{1}(N,:) is the size of coefficients associated with the N-th terminal node. The nodes are numbered from left to right and from top to bottom. The root index is 0.
- terNI{2} is a row vector containing the previous coefficients stored row-wise in the above specified order.

Methods

dtree	Constructor for the class DTREE.

expand	Expand data tree.
fmdtree	Field manager for DTREE object.
nodejoin	Recompose node.
nodesplt	Split (decompose) node.
rnodcoef	Reconstruct node coefficients.

defaninf	Define node information (all nodes).
get	Get DTREE object field contents.
plot	Plot DTREE object.
read	Read values in DTREE object fields.
set	Set DTREE object field contents.
write	Write values in DTREE object fields.

merge	Merge (recompose) the data of a node.
recons	Reconstruct node coefficients.
split	Split (decompose) the data of a terminal node.

Comments

- After the constructor, the first set of methods (between line separators) might not be overloaded (or only with great care). The second set of methods can be overloaded. The third set of methods must be overloaded to recompose, reconstruct, or decompose nodes data.

- The method nodejoin calls the method merge, the method nodesplt calls the method split, and the method rnodcoef calls the method recons.

- To define nodes information, you must overload the method defaninf. For each node N, the basic information is given by

  allNI(N,1:3): [index,size(1,1),size(1,2)];

  You can add other information by adding columns to allNI.

  See the WPTREE object method for an example.

- If the method get is not overloaded, using the DTREE get method you can get some object field contents (but not all).

For example, if T is parented by a DTREE object of order 2 and if 'Tfield' is a field of T, whose content is Tval, [a,b] = get(t,'order','Tfield') returns a = 2 and b = 'errorWTBX'. Nevertheless, using a nondocumented method you can get the right values. Namely: [a,b] = getwtbo(t,'order','Tfield') returns a = 2 and b=Tval.

WPTREE Object

Class **WPTREE** (Wavelet Packet Tree) -- Parent class: **DTREE**

Fields

dtree	Parent object
wavInfo	Structure (wavelet information)
entInfo	Structure (entropy information)

Fields Description

wavInfo

wavName	Wavelet Name
Lo_D	Low Decomposition filter
Hi_D	High Decomposition filter
Lo_R	Low Reconstruction filter
Hi_R	High Reconstruction filter

entInfo

entName	Entropy Name
entPar	Entropy Parameter

allNI Array(nbnode,5)   (field of the dtree parent object)

[ind,size,ent,ento]

ind	Index
size	Size of data
ent	Entropy
ento	Optimal Entropy

Methods

Constructor

Method	Description
wptree	Constructor for the class **WPTREE**

Methods That Overload Those of DTREE Class

Method	Description
defaninf	Define node information (all nodes).
get	Get WPTREE object field contents.
merge	Merge (recompose) the data of a node.
read	Read values in WPTREE object fields.
recons	Reconstruct wavelet packet coefficients.
set	Set WPTREE object field contents.

Method	Description
split	Split (decompose) the data of a terminal node.
tlabels	Labels for the nodes of a wavelet packet tree.
write	Write values in WPTREE object fields.

Proper Methods of WPTREE Class

Method	Description
bestlevt	Best level of a wavelet packet tree.
besttree	Best wavelet packet tree.
entrupd	Entropy update (wavelet packet tree).
wp2wtree	Extract wavelet tree from wavelet packet tree.
wpcoef	Wavelet packet coefficients.
wpcutree	Cut wavelet packet tree.
wpjoin	Recompose wavelet packet.
wpplotcf	Plot wavelet packets colored coefficients.
wprcoef	Reconstruct wavelet packet coefficients.
wprec	Wavelet packet reconstruction 1-D.
wprec2	Wavelet packet reconstruction 2-D.
wpsplt	Split (decompose) wavelet packet.
wpthcoef	Wavelet packet coefficients thresholding.
wpviewcf	Plot wavelet packets colored coefficients.

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 5-53
- "Building a Right Wavelet Tree Object (RWVTREE)" on page 5-60
- "Building a Wavelet Tree Object (WVTREE)" on page 5-66
- "Building a Wavelet Tree Object (EDWTTREE)" on page 5-73

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.

The definition of the new class is described below.

Class **WTREE** (parent class: **DTREE**)

Fields

dtree	Parent object
dwtMode	DWT extension mode
wavInfo	Structure (wavelet information)

wavInfo Structure information

wavName	Wavelet Name
Lo_D	Low Decomposition filter
Hi_D	High Decomposition filter
Lo_R	Low Reconstruction filter
Hi_R	High Reconstruction filter

Methods

wtree	Constructor for the class WTREE.
merge	Merge (recompose) the data of a node.
split	Split (decompose) the data of a terminal node.

Working With Wavelet Tree Objects (WTREE)

1-D Object

Load a signal.

```
load noisbloc
x = noisbloc;
```

Define the level and the wavelet.

```
lev = 3;
wav = 'db2';
```

Create the wavelet tree.

```
t = wtree(x,lev,wav);
```

Plot the wavelet tree. The approximations are labeled in yellow and the details are labeled in red. The detail nodes cannot be split.

```
plot(t)
```
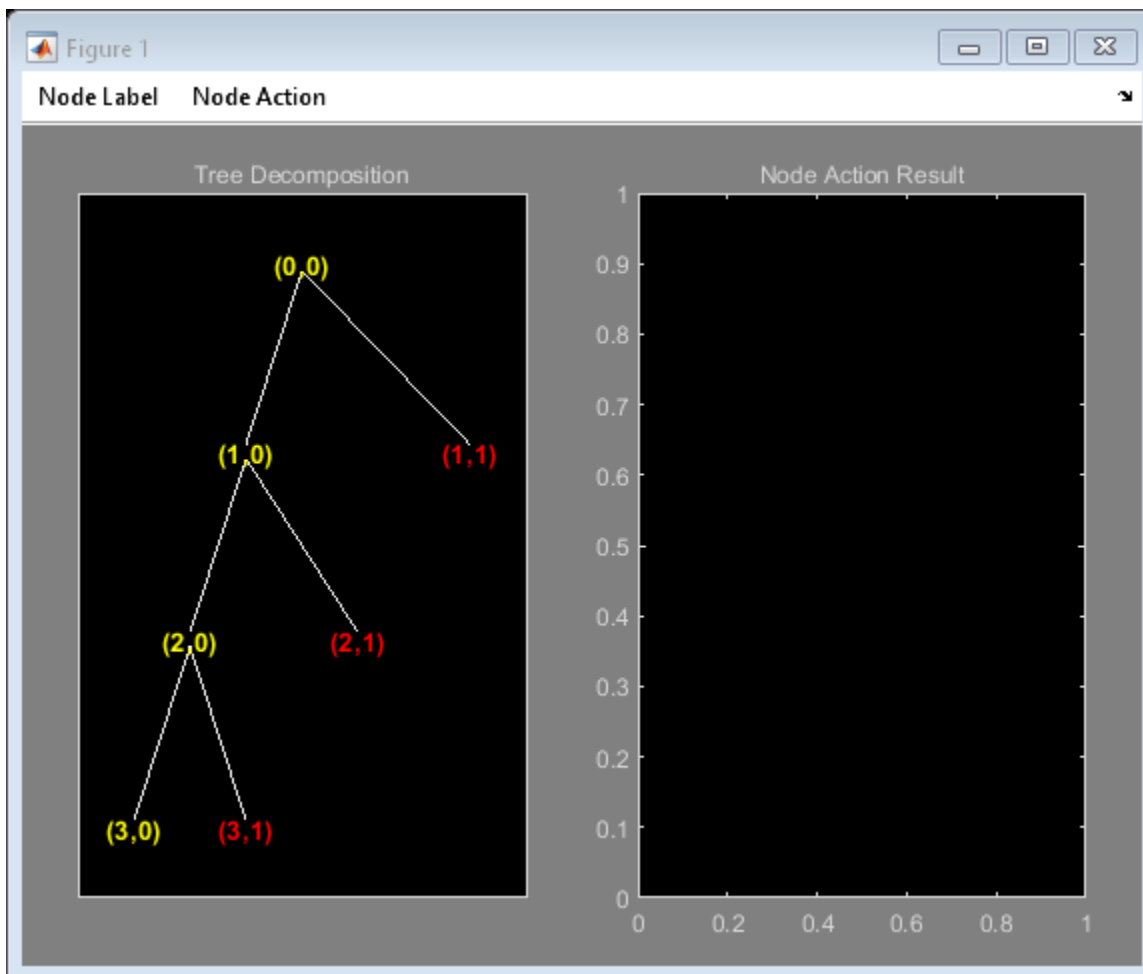

Change **Node Label** from **Visualize** to **Split-Merge**, and then click node (7), to get this figure:
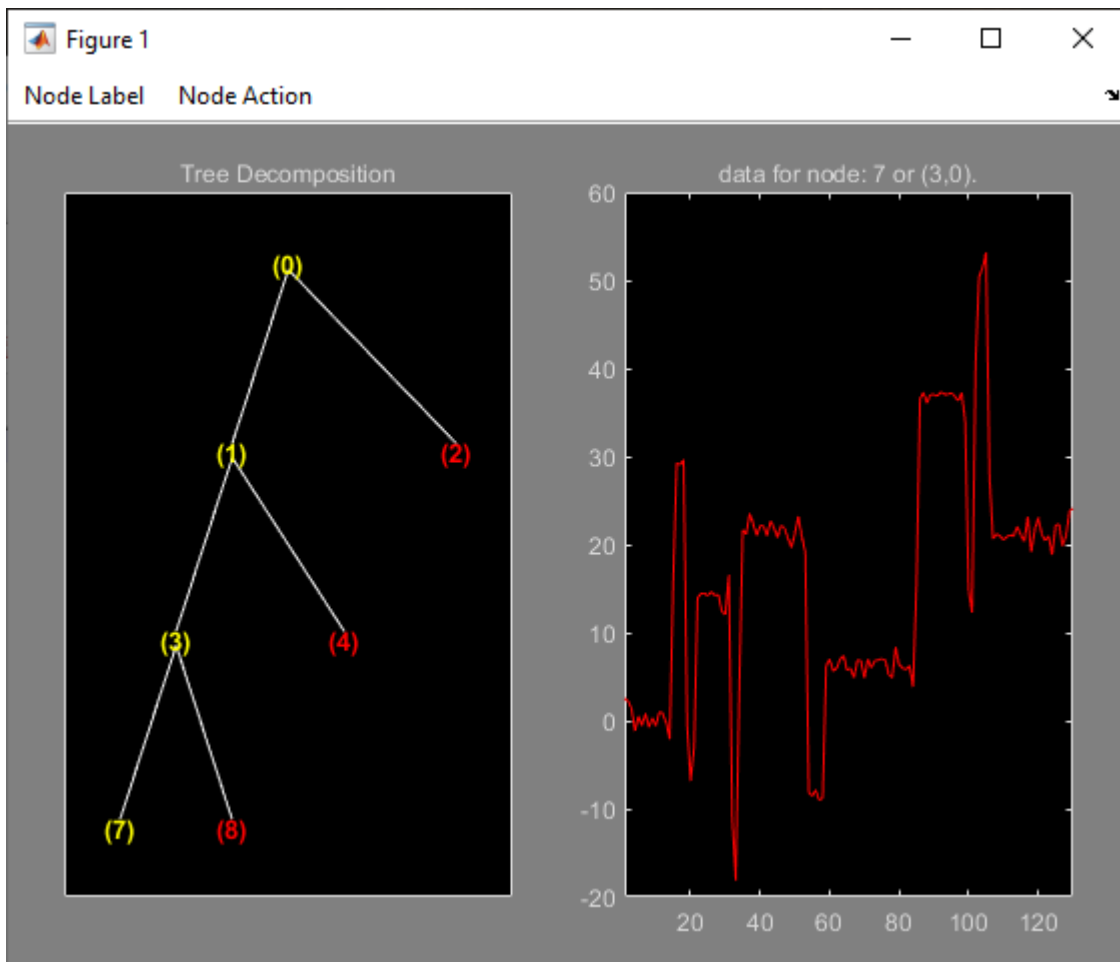
Change **Node Action** from **Visualize** to **Split-Merge**, and merge node (3). Then change **Node Action** from **Split-Merge** to **Visualize**, and click node (3) to get this figure:
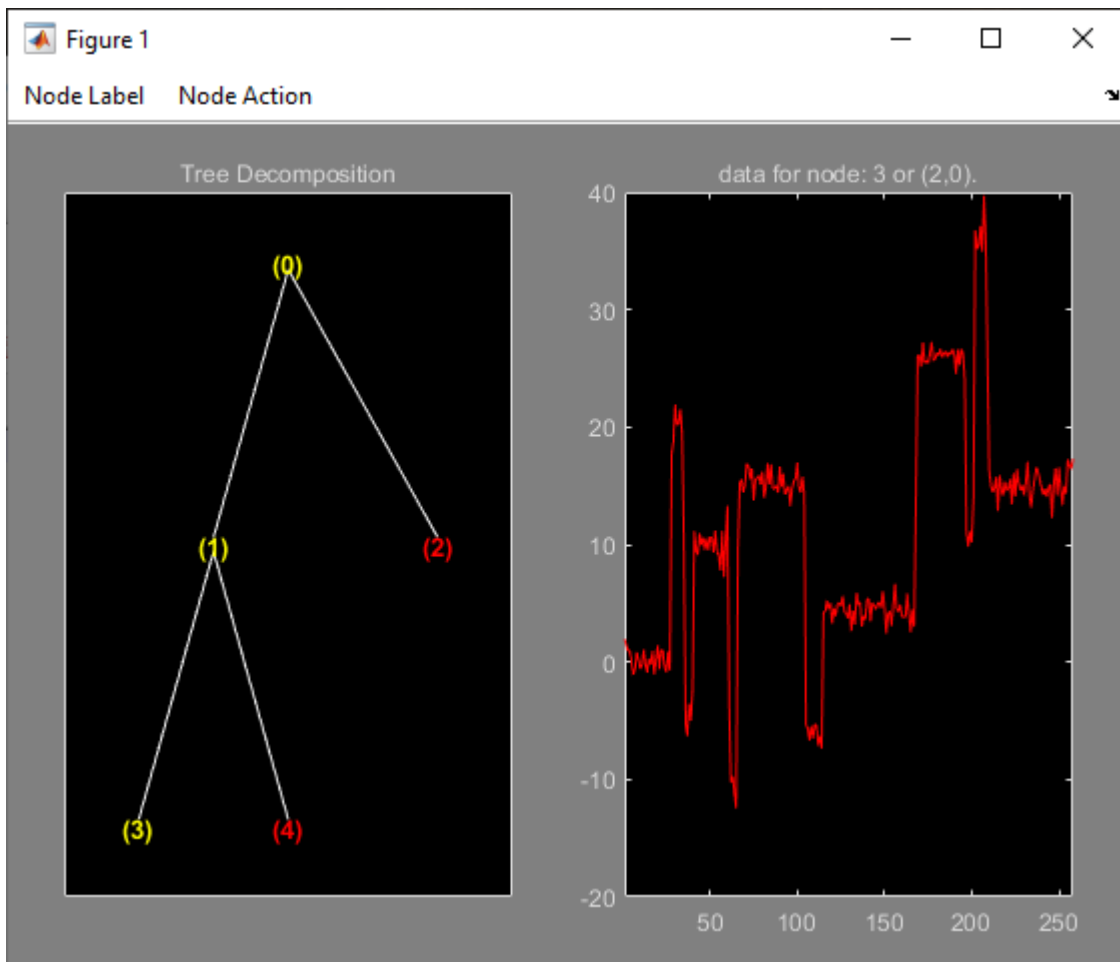
2-D Object

Load an image.

```
load woman
```

Define the level and the wavelet.

```
lev = 2;
wav = 'db2';
```

Create the wavelet tree.

```
t = wtree(X,lev,wav);
```

Plot the tree.

```
plot(t)
```
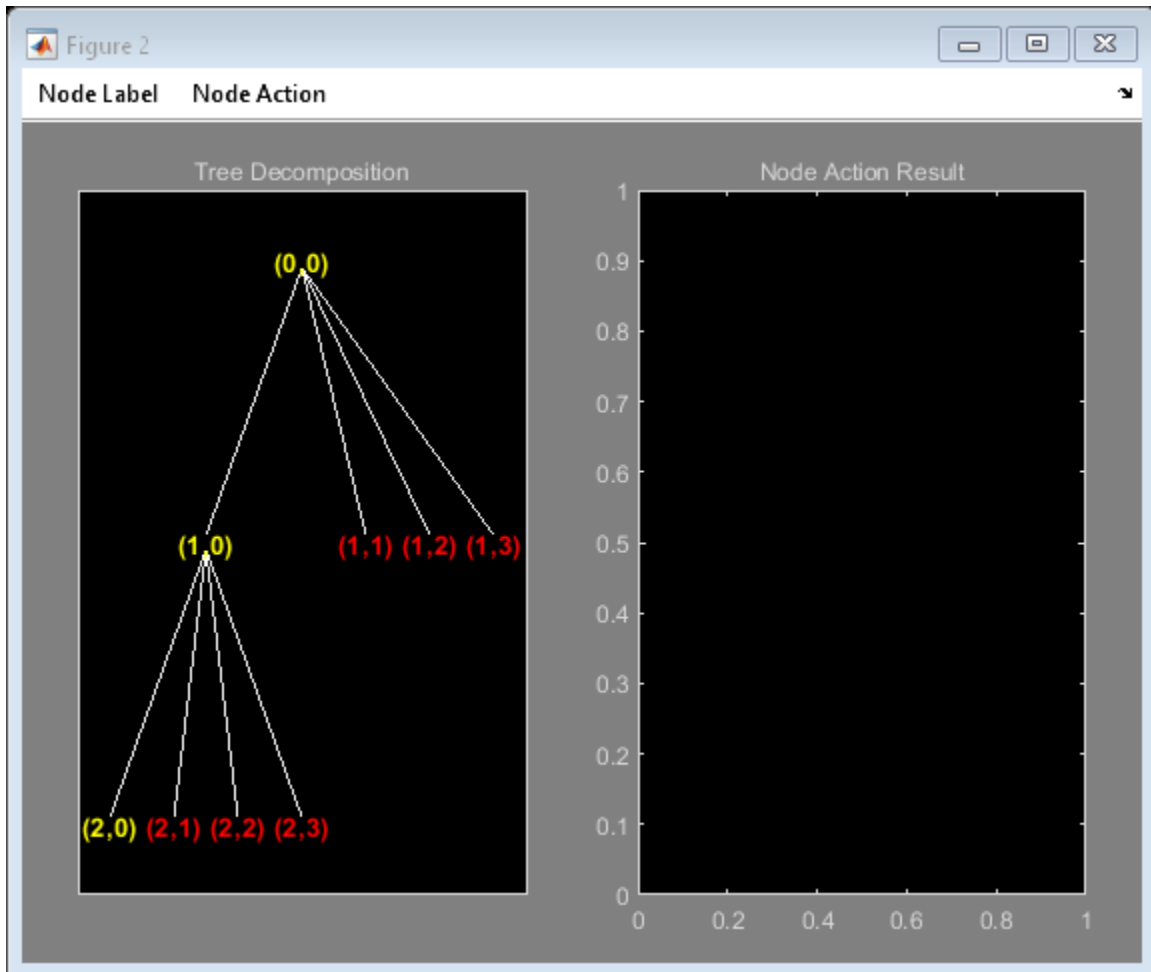
Change **Node Label** from **Depth_Position** to **Index**. Click the node (5). You get the following plot.
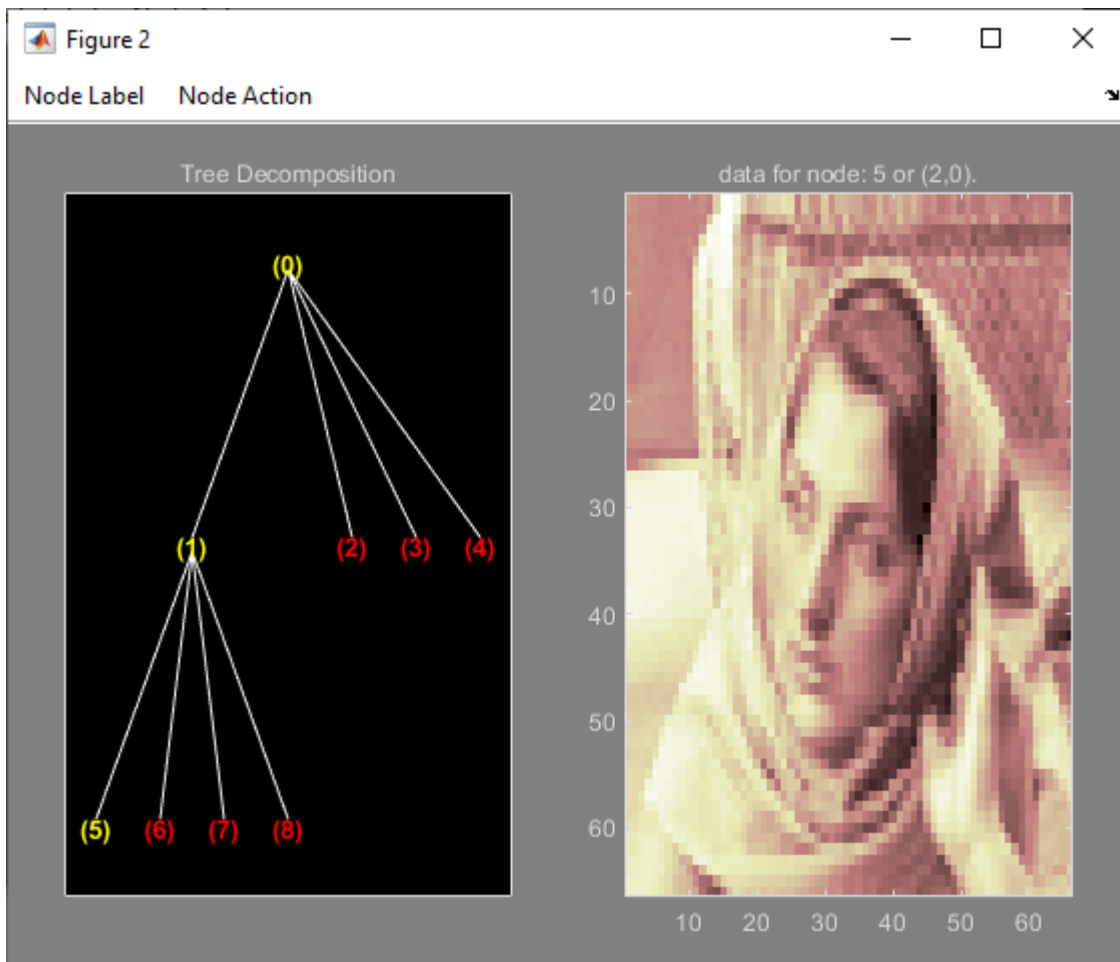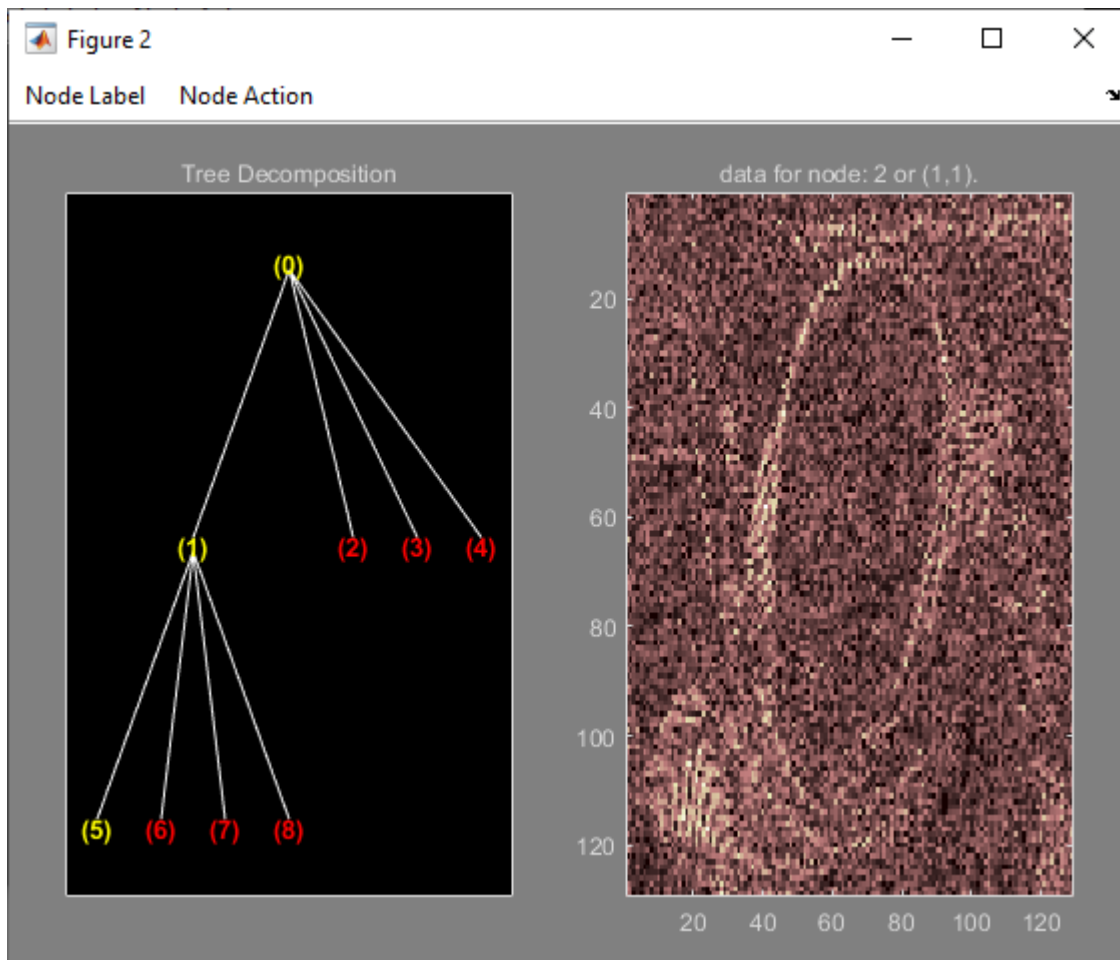
Click the node (2). You obtain the following plot.

Change **Node Action** from **Visualize** to **Split-Merge**. Split the node (5). Change **Node Action** from **Split-Merge** to **Visualize**. Click the node (21). You obtain the following plot.
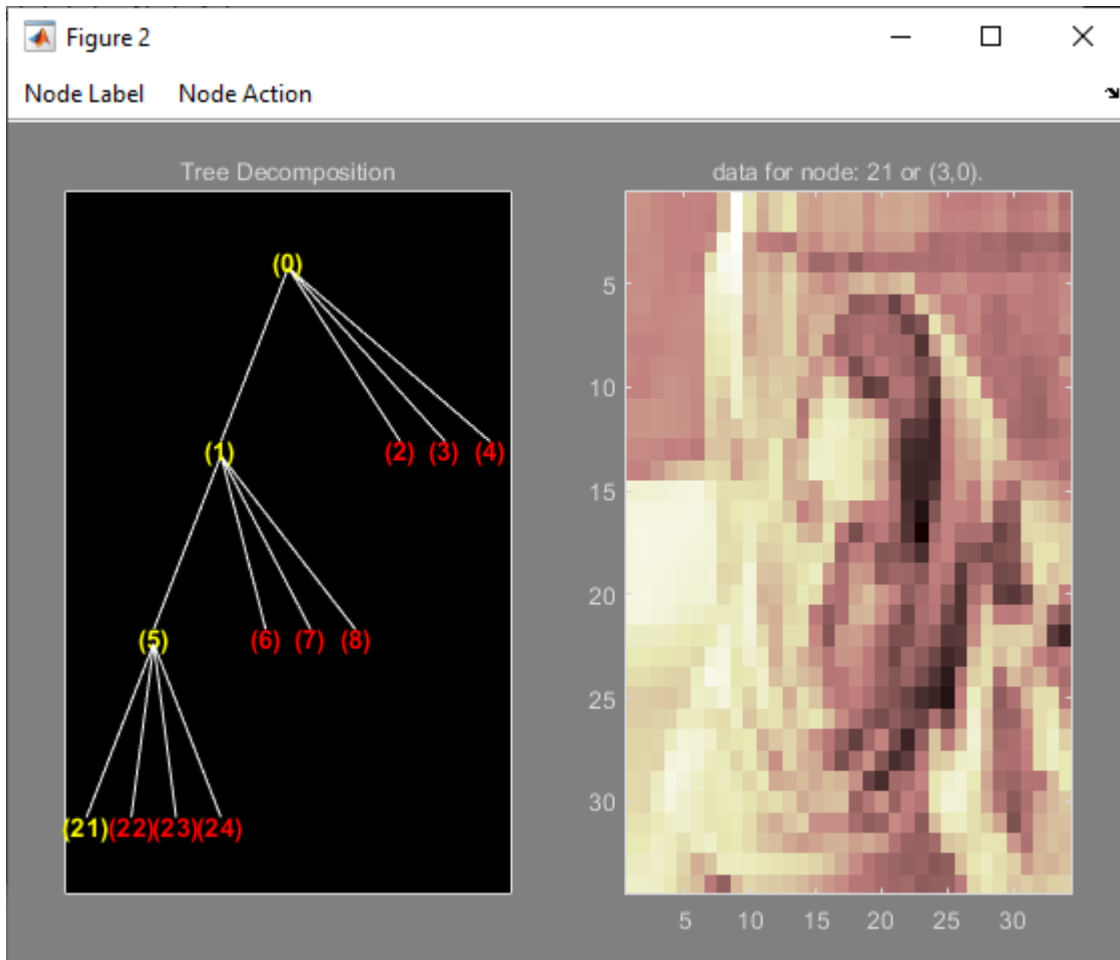
Building a Right Wavelet Tree Object (RWVTREE)

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**.

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.

Working With Right Wavelet Tree Objects (RWVTREE)

1-D Object

Load a signal.

```
load noisbloc
x = noisbloc;
```

Define the level and the wavelet.

```
lev = 3;
wav = 'db2';
```

Create the wavelet tree.

```
t = rwvtree(x,lev,wav);
```

Plot the tree. The approximations are labeled in yellow and the details are labeled in red. The detail nodes cannot be split.

```
plot(t)
```
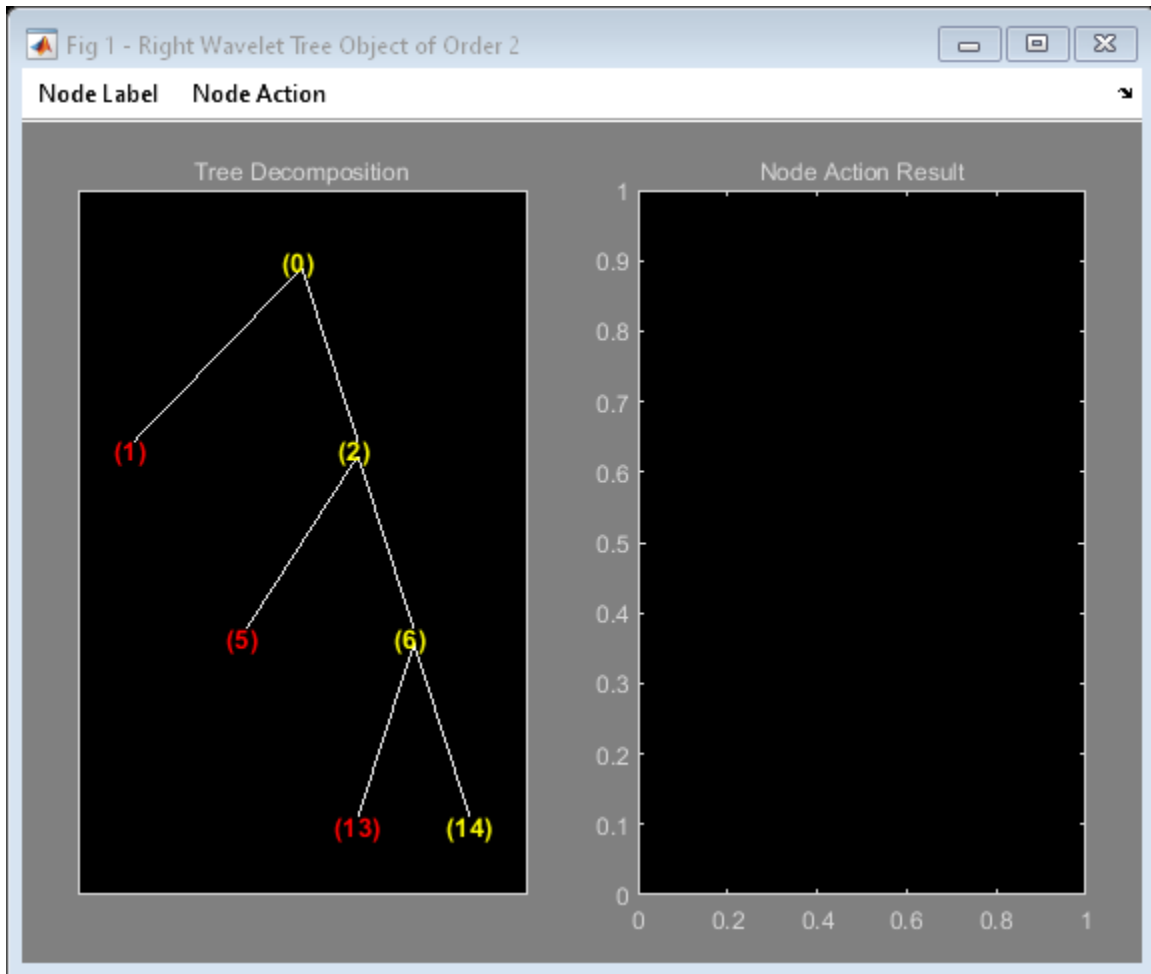
Change **Node Action** from **Visualize** to **Split-Merge**. Merge the node (6). Change **Node Action** from **Split-Merge** to **Visualize**. Click the node (6). You obtain the following plot.
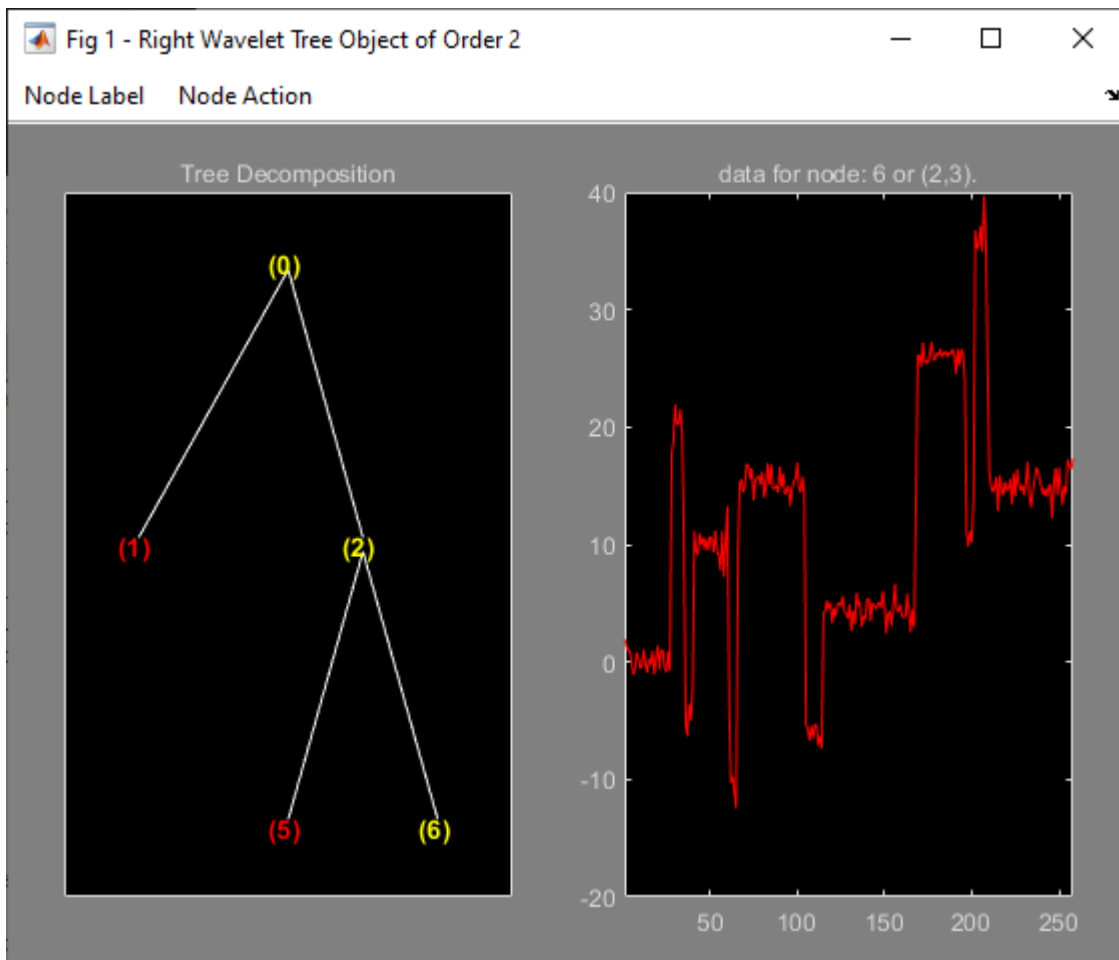
2-D Object

Load an image.

```
load woman
```

Define the level and the wavelet.

```
lev = 2;
wav = 'db2';
```

Create the wavelet tree.

```
t = wtree(X,lev,wav);
```

Plot the tree.

```
plot(t)
```
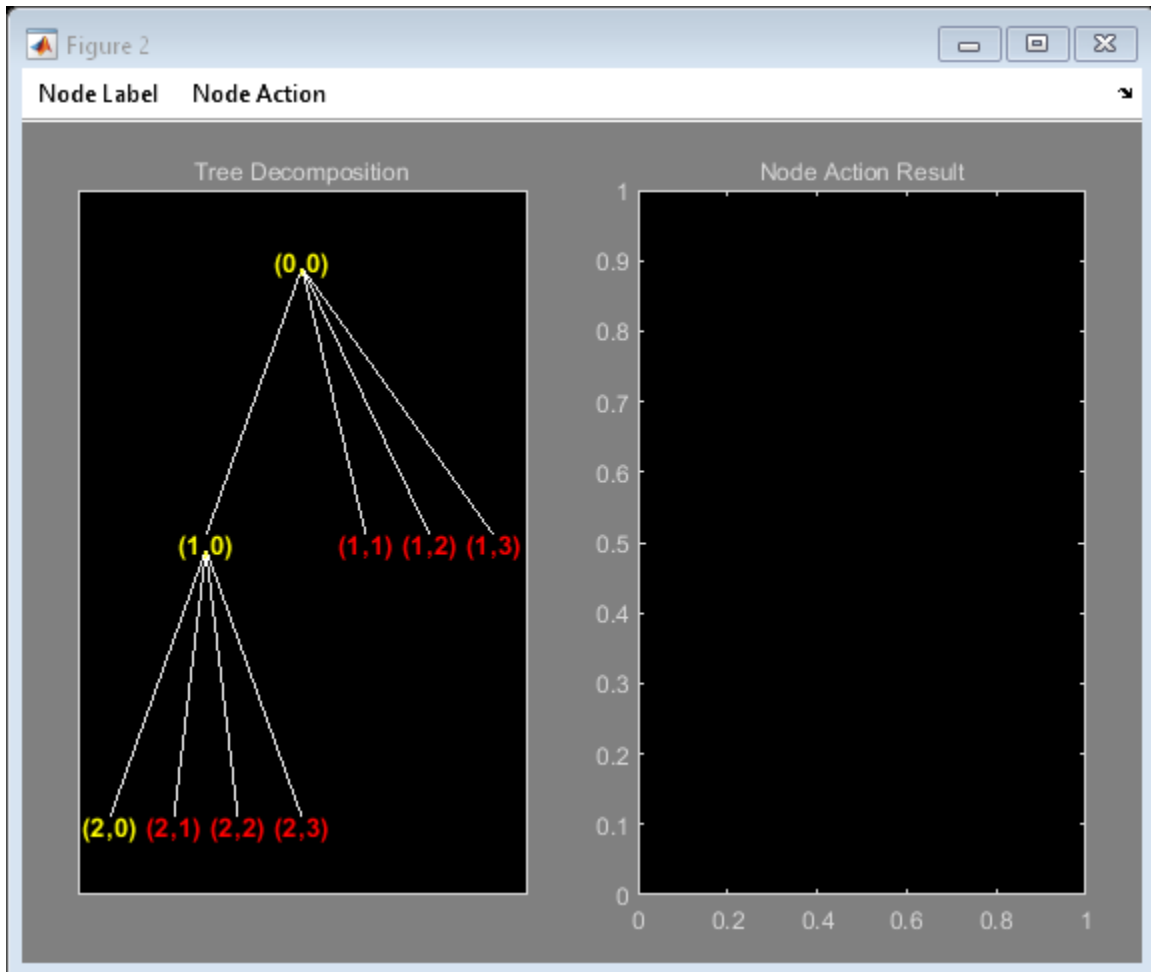
Click the node (2,0). You get the following plot.
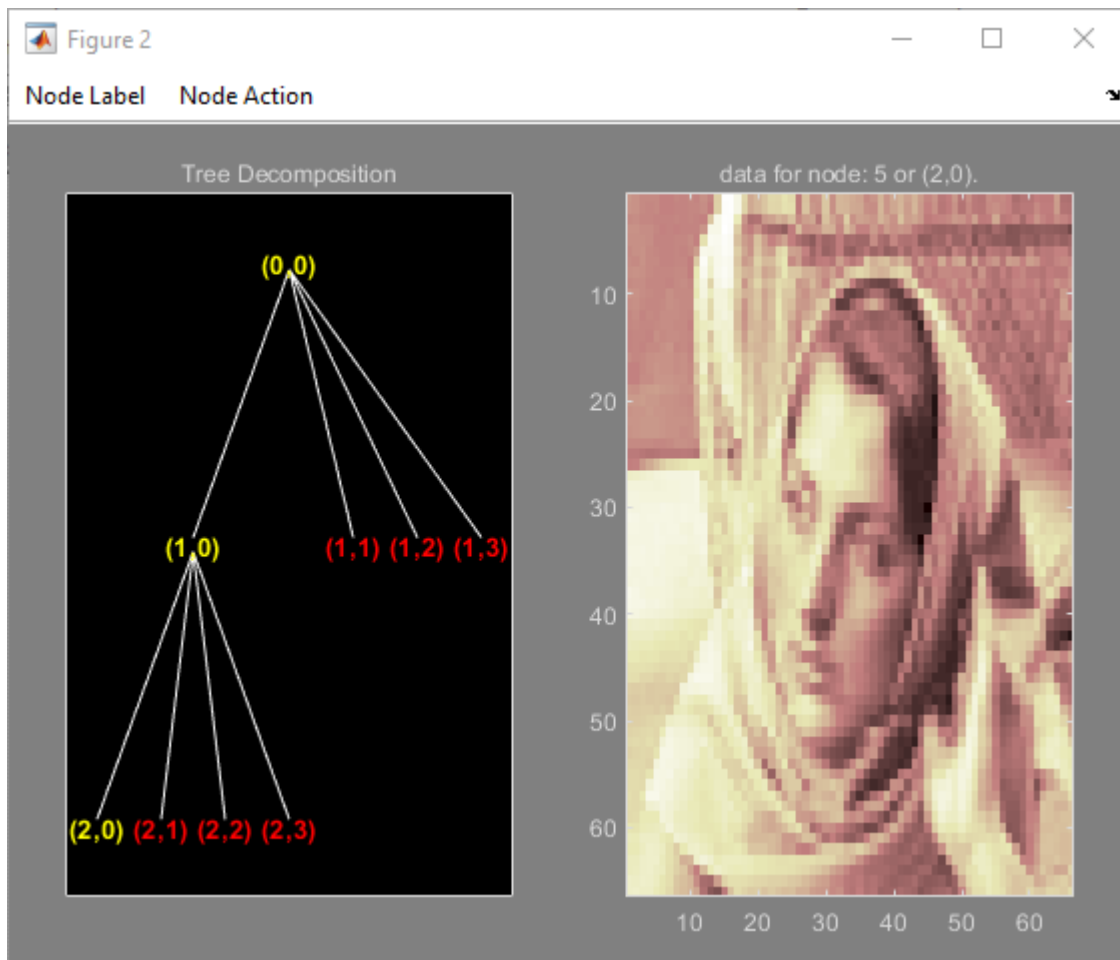
Change **Node Action** from **Visualize** to **Split-Merge**. Split the node (2,0). Change **Node Action** from **Split-Merge** to **Visualize**. Click the node (3,0). You obtain the following plot.
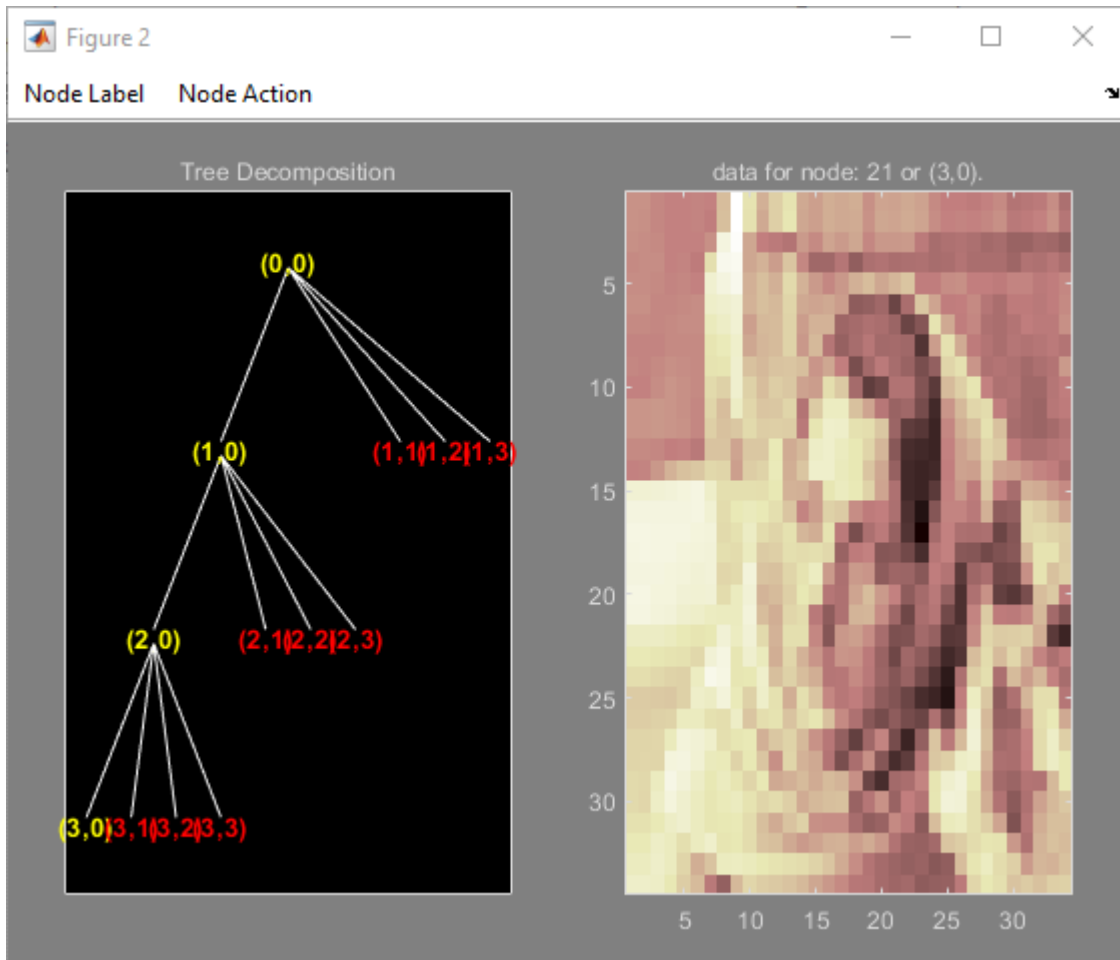
Building a Wavelet Tree Object (WVTREE)

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**.

The definition of the new class is described below.

Class **WVTREE** (parent class: **WTREE**)

Fields

dummy	Not used
wtree	Parent object

Methods

wvtree	Constructor for the class WVTREE.
get	Get WVTREE object field contents.
plot	Plot WVTREE object.
recons	Reconstruct node coefficients.

Working With Wavelet Tree Objects (WVTREE)

1-D Object

Load the signal.

```
load noisbloc
x = noisbloc;
```

Define the level and the wavelet.

```
lev = 3;
wav = 'db2';
```

Create the wavelet tree.

```
t = wvtree(x,lev,wav);
```

Plot the tree. The approximations are labeled in yellow and the details are labeled in red. The detail nodes cannot be split.

```
plot(t)
```
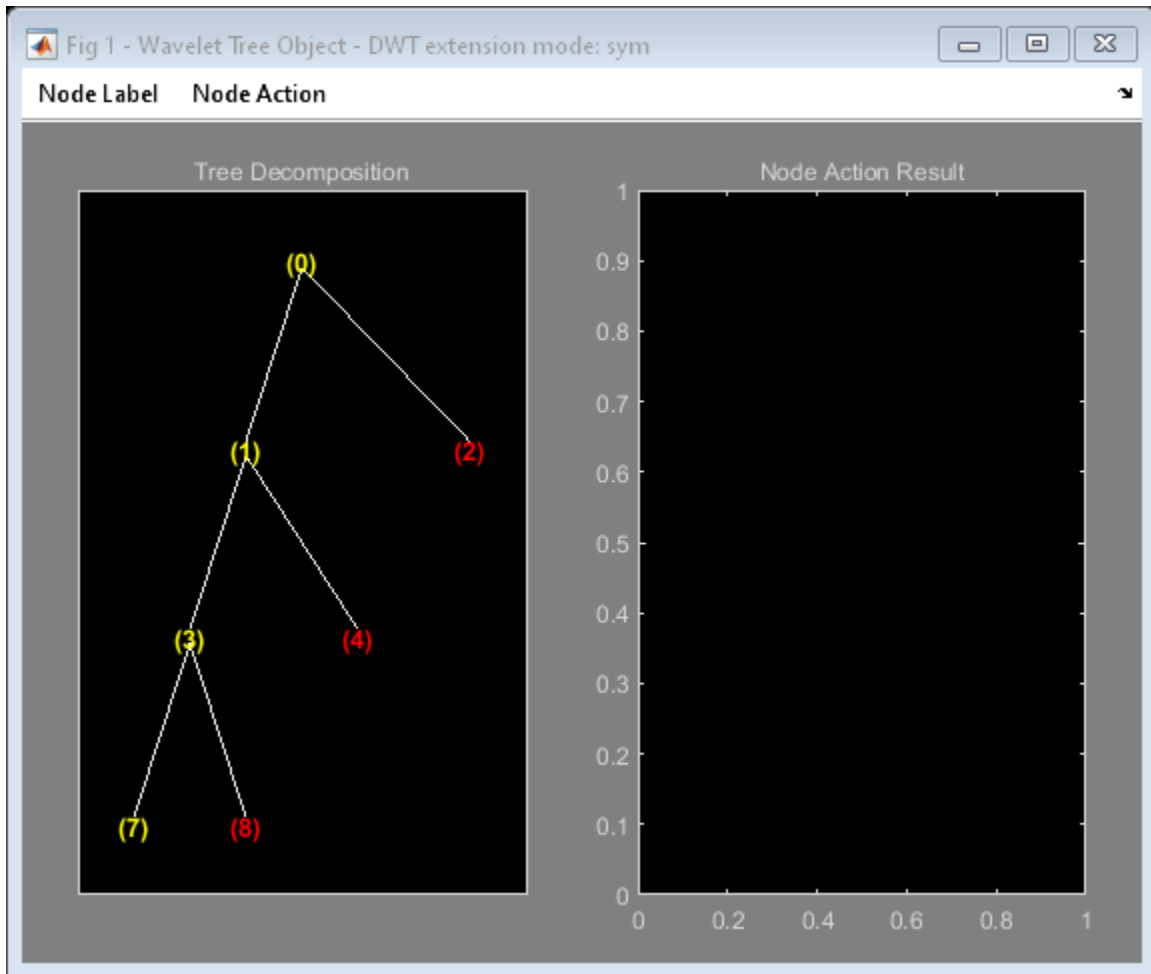
Change **Node Action** from **Visualize** to **Split-Merge**. Merge the node (3). Change **Node Action** from **Split-Merge** to **Reconstruct**. Click the node (3). You obtain the following plot.

2-D Object

Load an image.

```
load woman
```
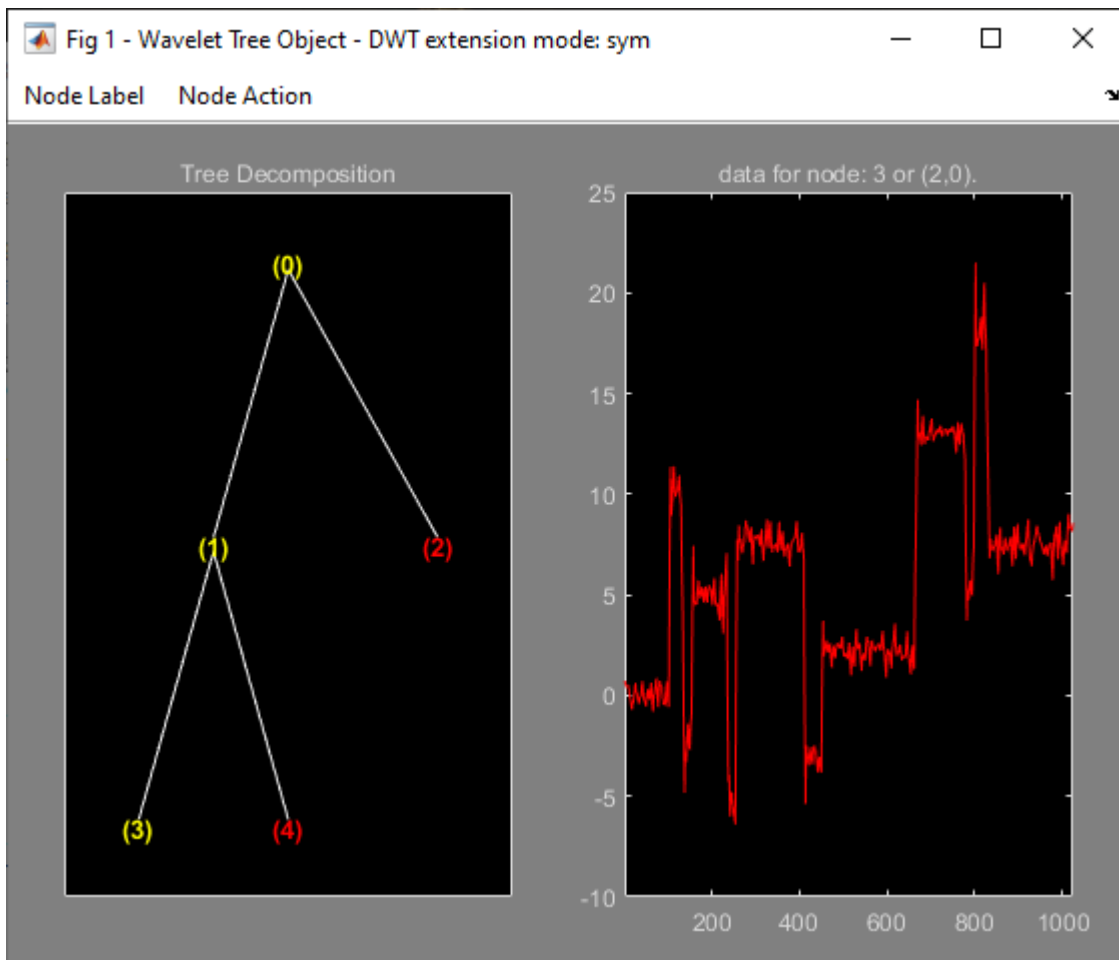
Define the level and the wavelet.

```
lev = 2;
wav = 'db1';
```

Create the wavelet tree.
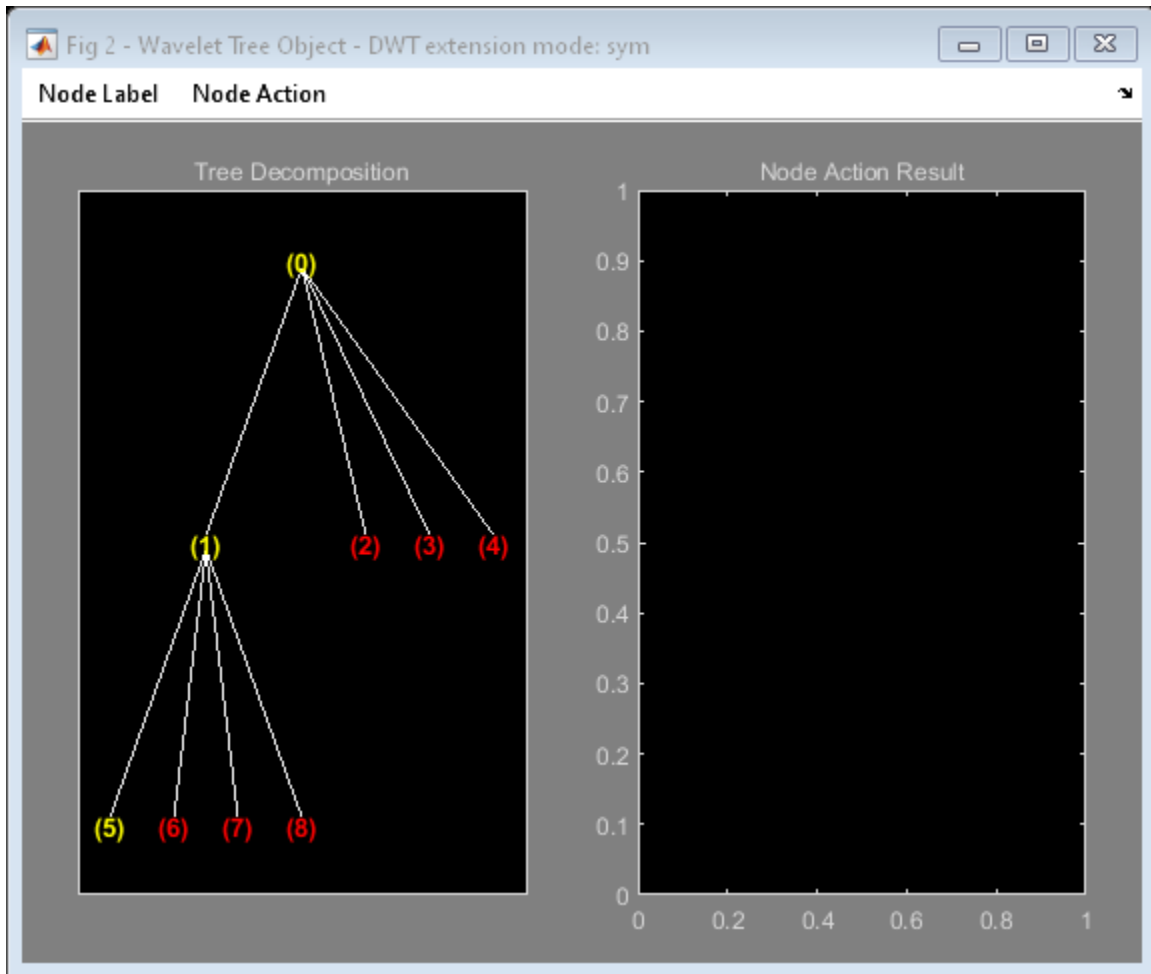
```
t = wvtree(X,lev,wav);
```

Plot the tree.

```
plot(t)
```

Click the node (5). You get the following plot.
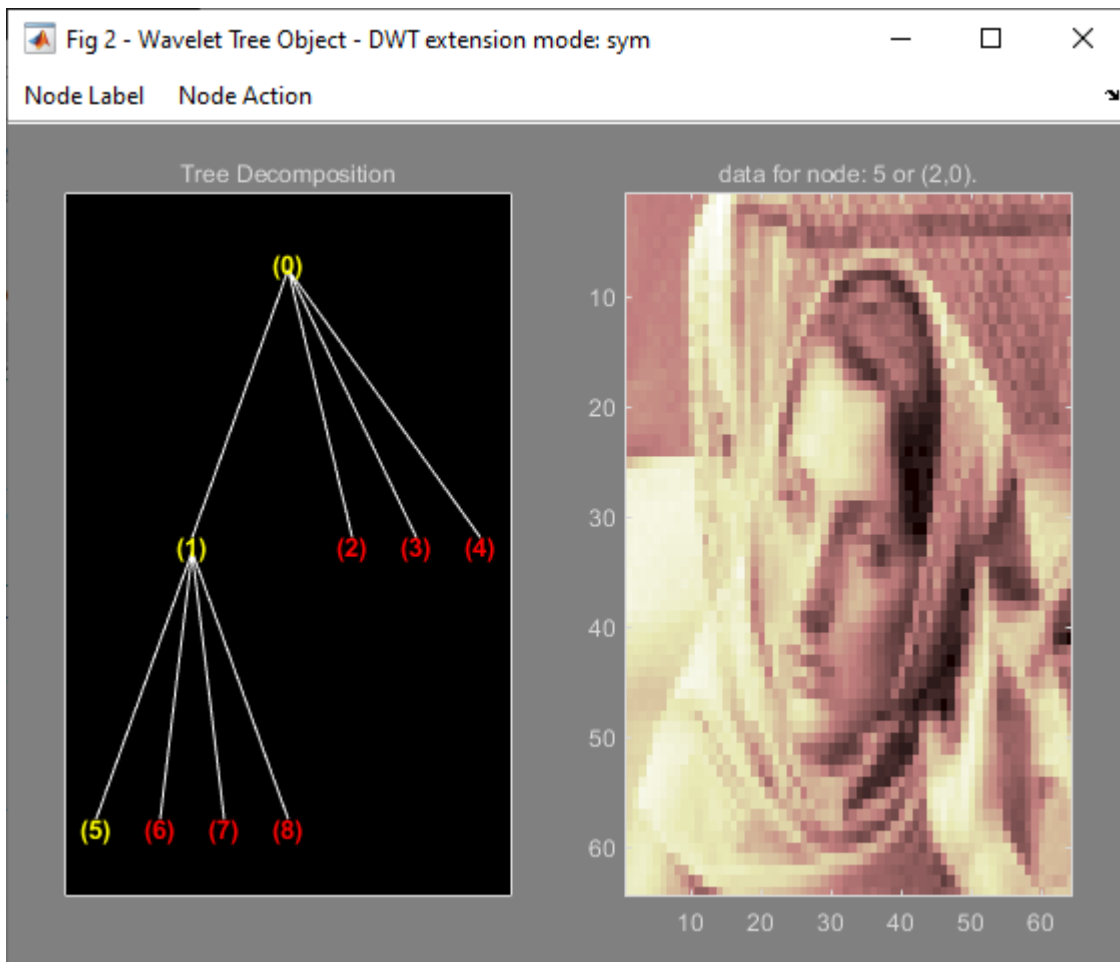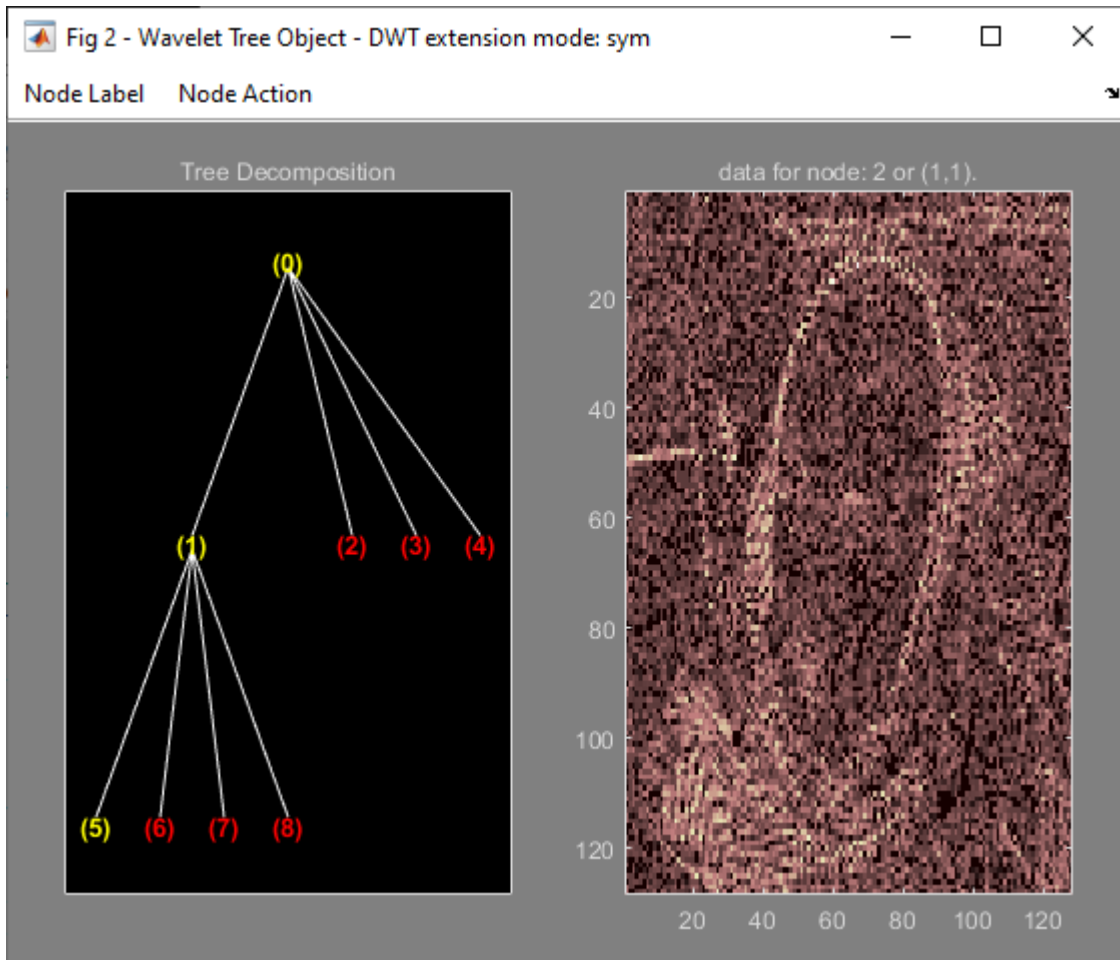
Click the node (2). You obtain the following plot.

Change **Node Action** from **Visualize** to **Split-Merge**. Split the node (5). Change **Node Action** from **Split-Merge** to **Reconstruct**. Click the node (21). You obtain the following plot.
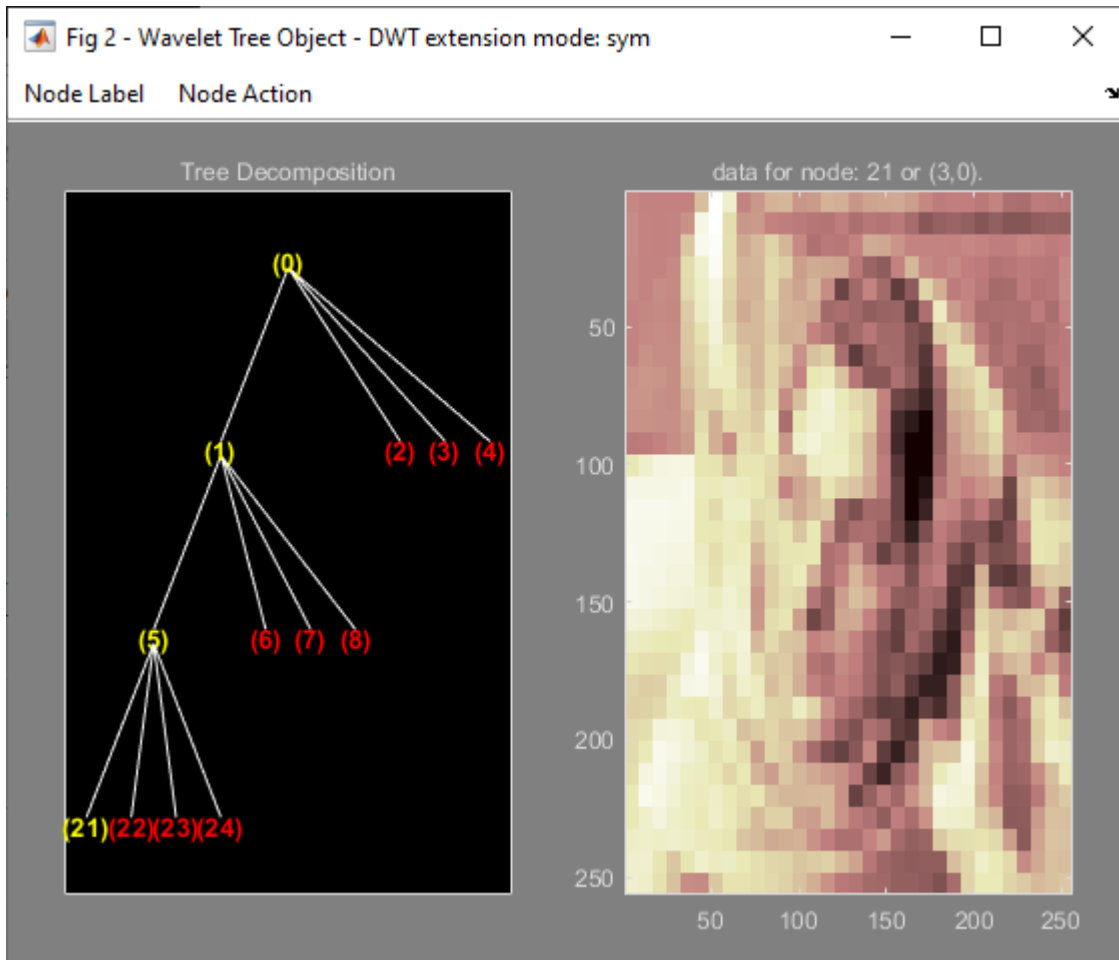
Building a Wavelet Tree Object (EDWTTREE)

This example creates a new class of objects: **EDWTTREE**.

We define an ε-DWT tree class starting from the class DTREE and overloading the methods `merge`, `plot`, `recons`, and `split`.

For more information on the ε-DWT, see the section "ε-Decimated DWT" on page 3-57.

The `plot` method shows how to add **Node Labels**, **Node Actions**, and **Tree Actions**.

The definition of the new class is described below.

Class **EDWTTREE** (parent class: **DTREE**)

Fields

dtree	Parent object
dwtMode	DWT extension mode
wavInfo	Structure (wavelet information)

Fields Description

`wavInfo`

`wavName`	Wavelet Name
`Lo_D`	Low Decomposition filter
`Hi_D`	High Decomposition filter
`Lo_R`	Low Reconstruction filter
`Hi_R`	High Reconstruction filter

Methods

`edwttree`	Constructor for the class *EDWTTREE*.
`merge`	Merge (recompose) the data of a node.
`plot`	Plot *EDWTTREE* object.
`recons`	Reconstruct node coefficients.
`split`	Split (decompose) the data of a terminal node.

Working With Wavelet Tree Object (EDWTTREE)

Load the signal.

```
load noisbloc
x = noisbloc;
```

Define the level and the wavelet.

```
lev = 2;
wav = 'haar';
```

Create the wavelet tree.

```
t = edwttree(x,lev,wav);
```

Plot the tree. The approximations are labeled in yellow and the details are labeled in red. The detail nodes cannot be split. The title of the figure contains the DWT extension mode used (`'per'` in the present example) and the name of the denoising method.

```
plot(t)
```
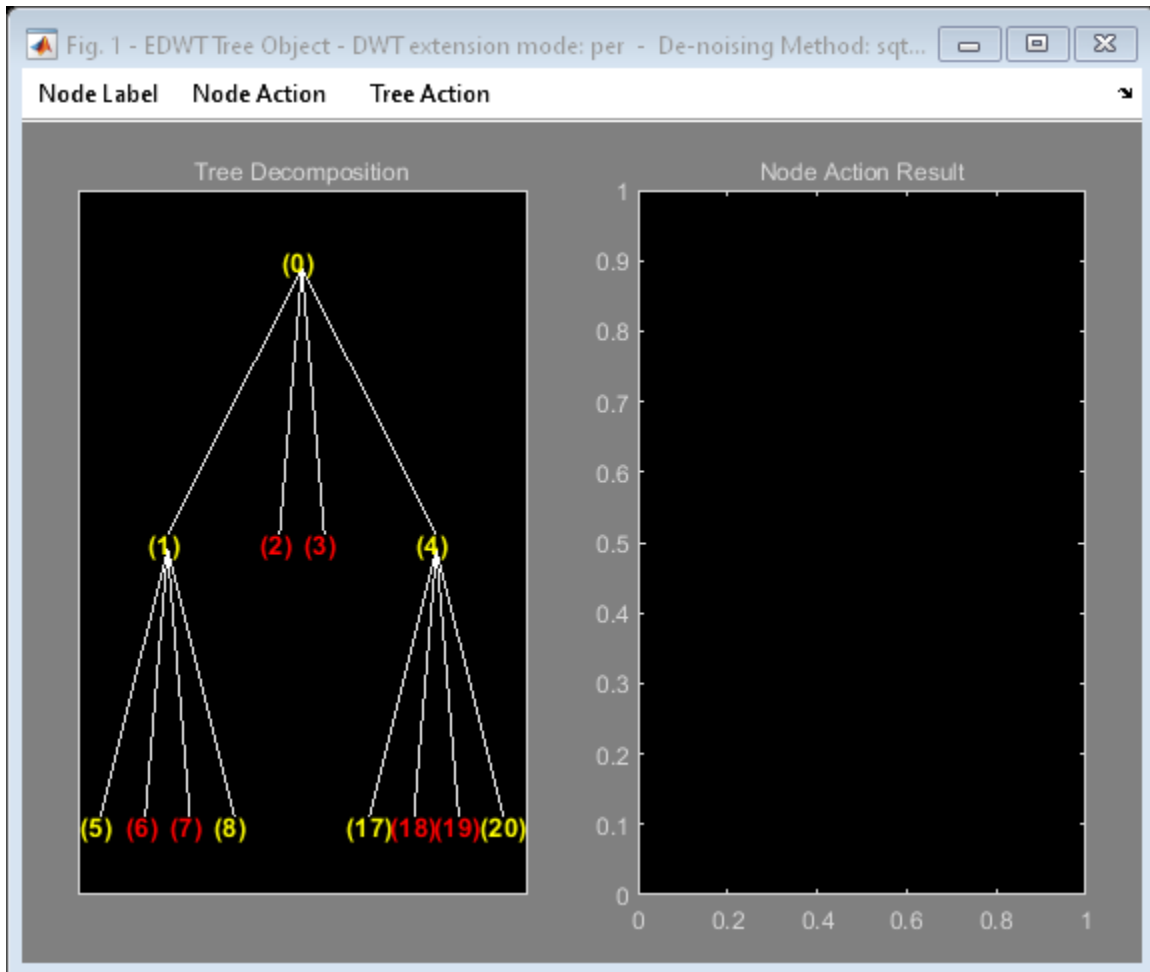
Click the node (0). You obtain the following plot.
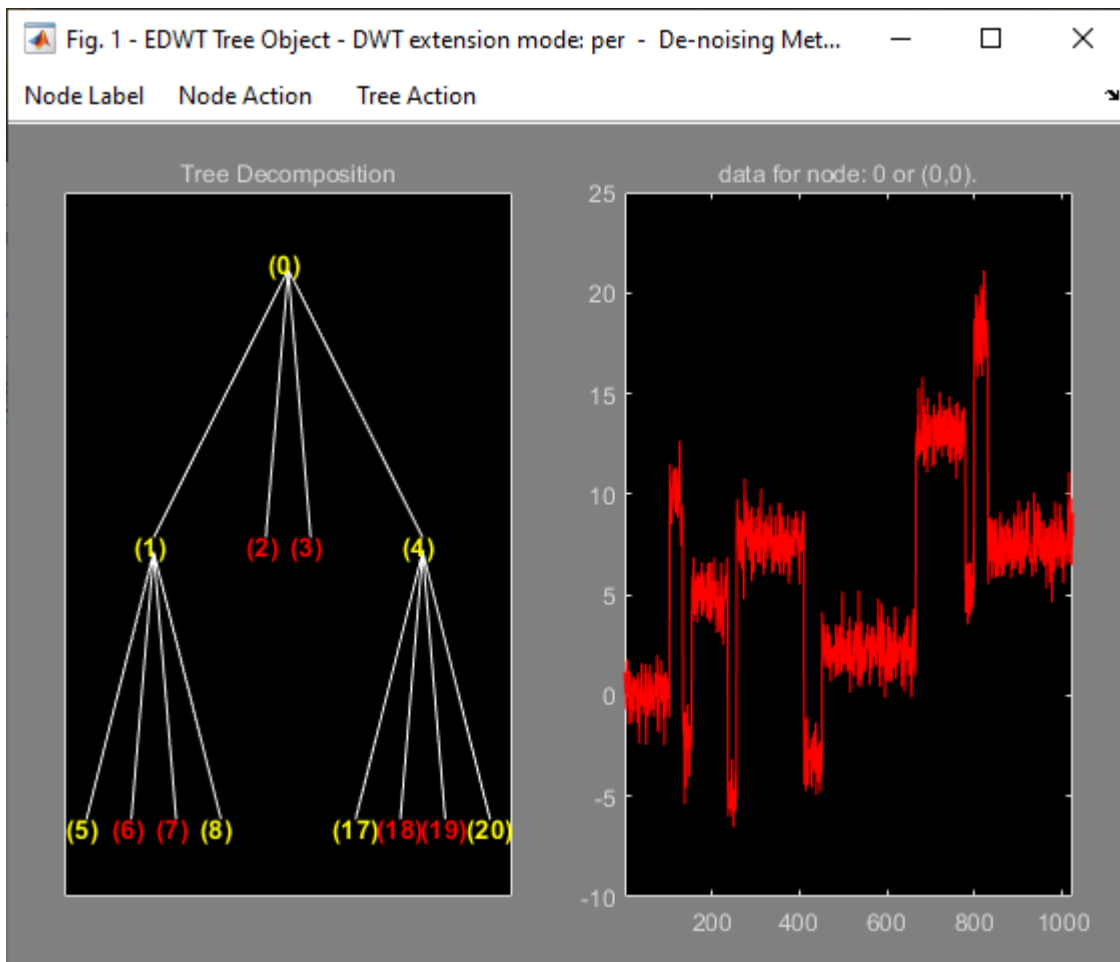
Change **Node Action** from **Visualize** to **Split-Merge**. Split the nodes (5) and (20).
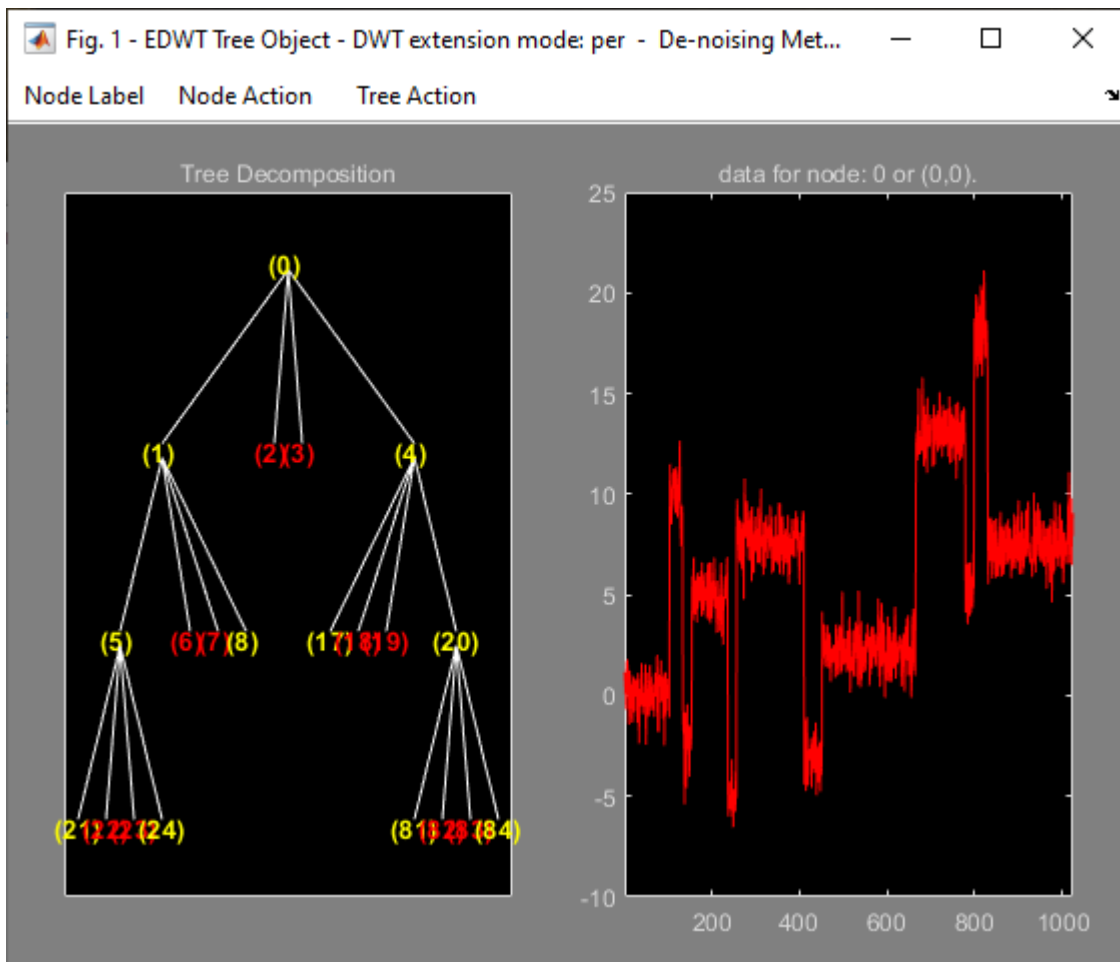
Select **Tree Action**: **De-noise**. Click the node (0). You obtain the following plot.
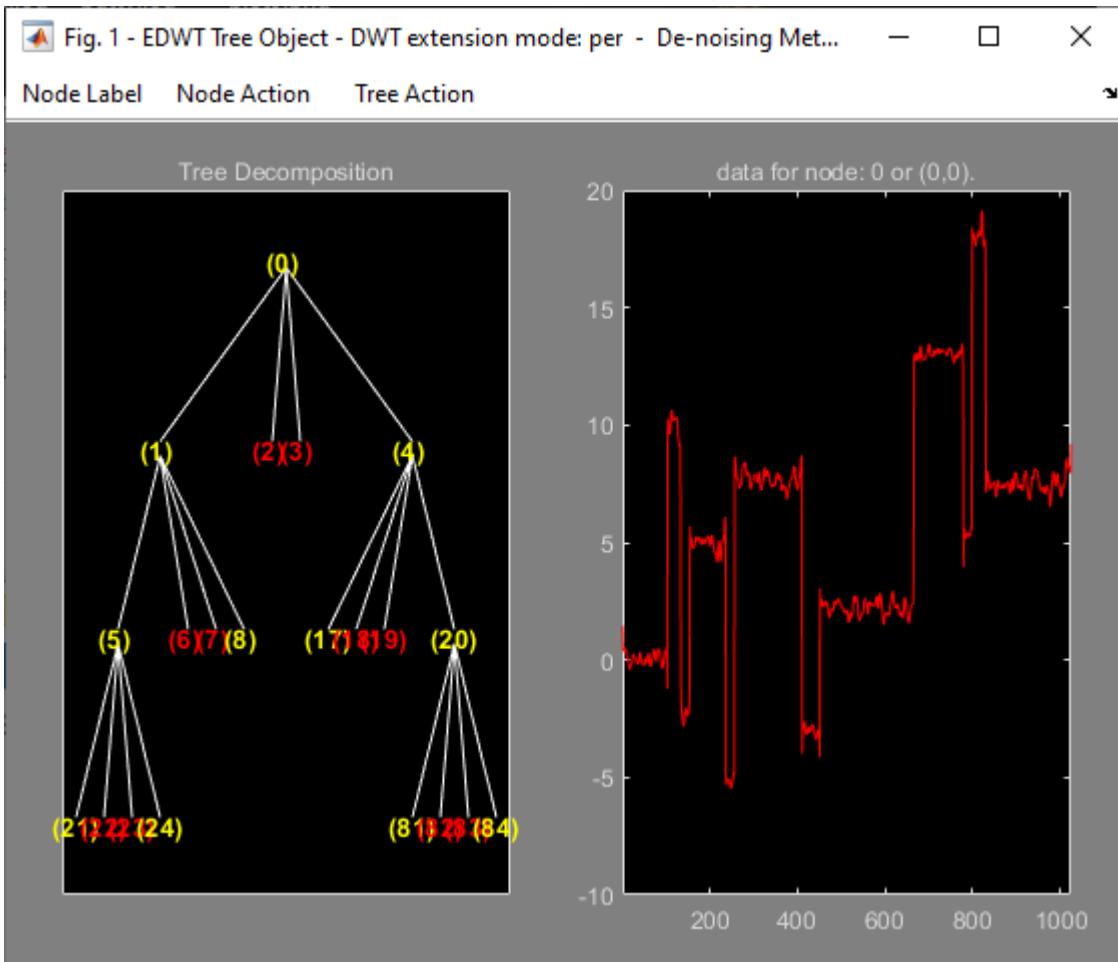
Denoising, Nonparametric Function Estimation, and Compression

Wavelet Denoising and Nonparametric Function Estimation

In this section...
"Denoising Methods" on page 6-3
"Soft or Hard Thresholding" on page 6-5
"Dealing with Unscaled Noise and Nonwhite Noise" on page 6-6
"Wavelet Denoising in Action" on page 6-7
"Extension to Image Denoising" on page 6-11
"1-D Wavelet Variance Adaptive Thresholding" on page 6-13
"Wavelet Denoising Analysis Measurements" on page 6-16

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 $n = 0,1,2,...N-1$. The $e(n)$ are Gaussian random variables distributed as $N(0,1)$. The variance of the $\sigma e(n)$ is σ^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

$$
\begin{pmatrix}
f(0) + \sigma e(0) \\
f(1) + \sigma e(1) \\
f(2) + \sigma e(2) \\
. \\
. \\
. \\
f(N-1) + \sigma e(N-1)
\end{pmatrix}
=
\begin{pmatrix}
f(0) \\
f(1) \\
f(2) \\
. \\
. \\
. \\
f(N-1)
\end{pmatrix}
+
\begin{pmatrix}
\sigma e(0) \\
\sigma e(1) \\
\sigma e(2) \\
. \\
. \\
. \\
\sigma e(N-1)
\end{pmatrix}
$$

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');
```
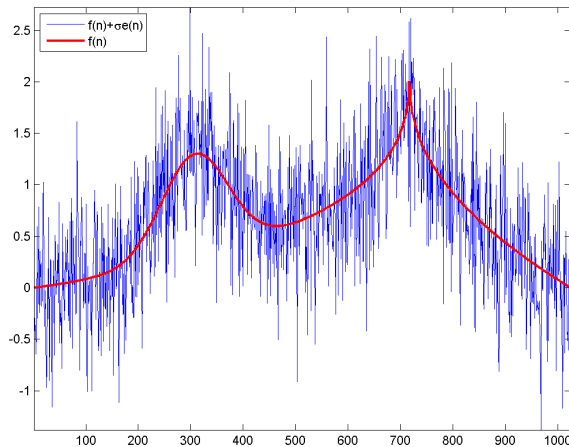
For a broad class of functions (signals, images) that possess certain smoothness properties, wavelet techniques are optimal or near optimal for function recovery.

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 $$\sqrt{2\ln(N)}$$ 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 $log(length(s))$.

- 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);
```
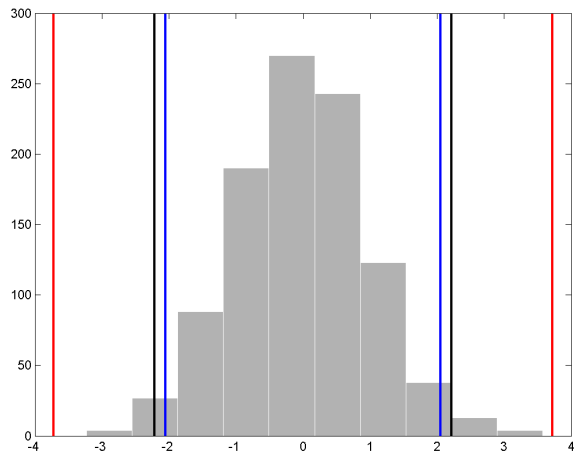
For Stein's Unbiased Risk Estimate (SURE) and minimax thresholds, approximately 3% of coefficients are retained. In the case of the universal threshold, all values are rejected.

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');
```
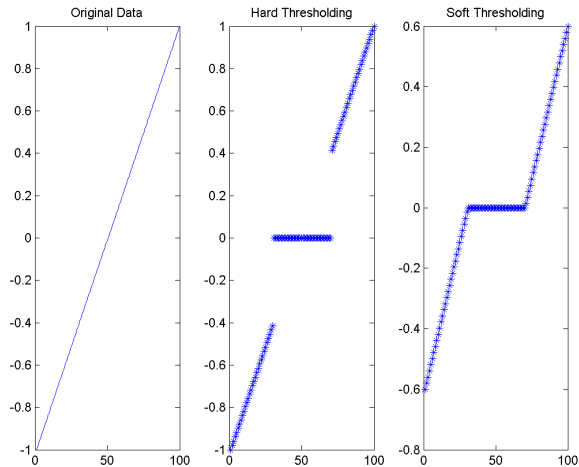

Dealing with Unscaled Noise and Nonwhite Noise

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',sorh,'NoiseEstimate',scal)
```

which returns the denoised version `sd` of the original signal `s` obtained using the `tptr` denoising method. Other parameters needed are `sorh`, `scal`, and `wname`. The parameter `sorh` specifies the thresholding of details coefficients of the decomposition at level `n` of `s` by the wavelet called `wav`. The remaining parameter `scal` is to be specified. It corresponds to 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 `wdencmp` 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 = wdencmp(opt,x,wav,n,thr,sorh,keepapp);
```

where

- `opt = 'gbl'` and `thr` is a positive real number for uniform threshold.
- `opt = 'lvd'` and `thr` is a vector for level dependent threshold.
- `keepapp = 1` to keep approximation coefficients, as previously and
- `keepapp = 0` to allow approximation coefficients thresholding.
- `x` is the signal to be denoised and `wav, n, sorh` 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 `xref` 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')
```
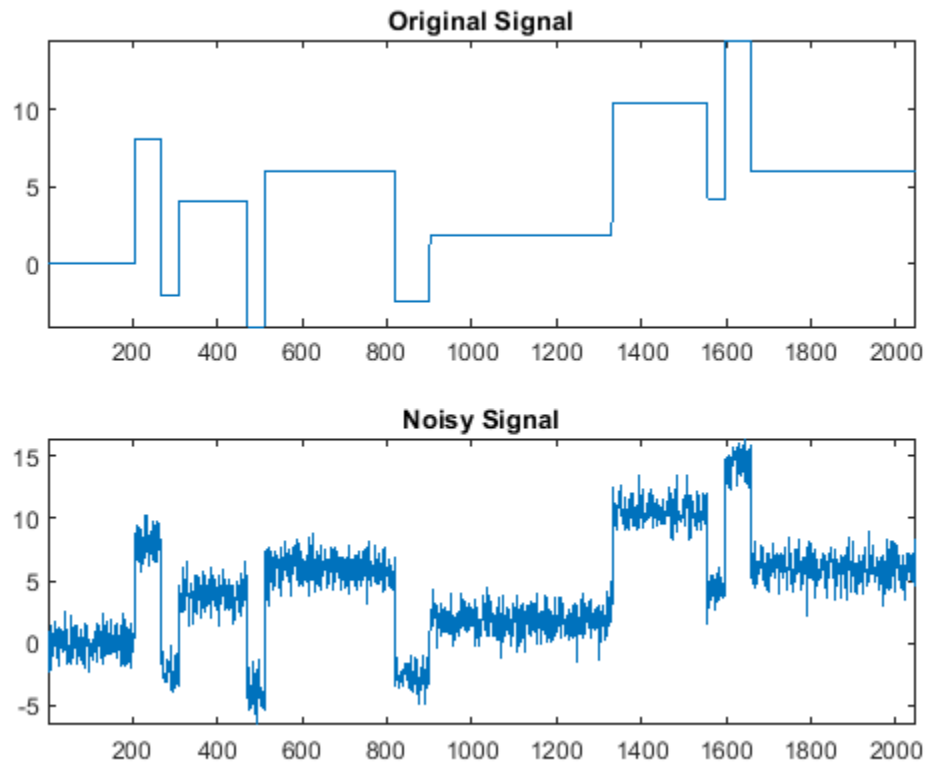
Denoise the noisy signal using soft heuristic SURE thresholding on detail coefficients obtained from the wavelet decomposition of x using the sym8 wavelet. Use the default settings of wdenoise for the remaining parameters. Compare with the original signal.

```
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')
```

Original Signal
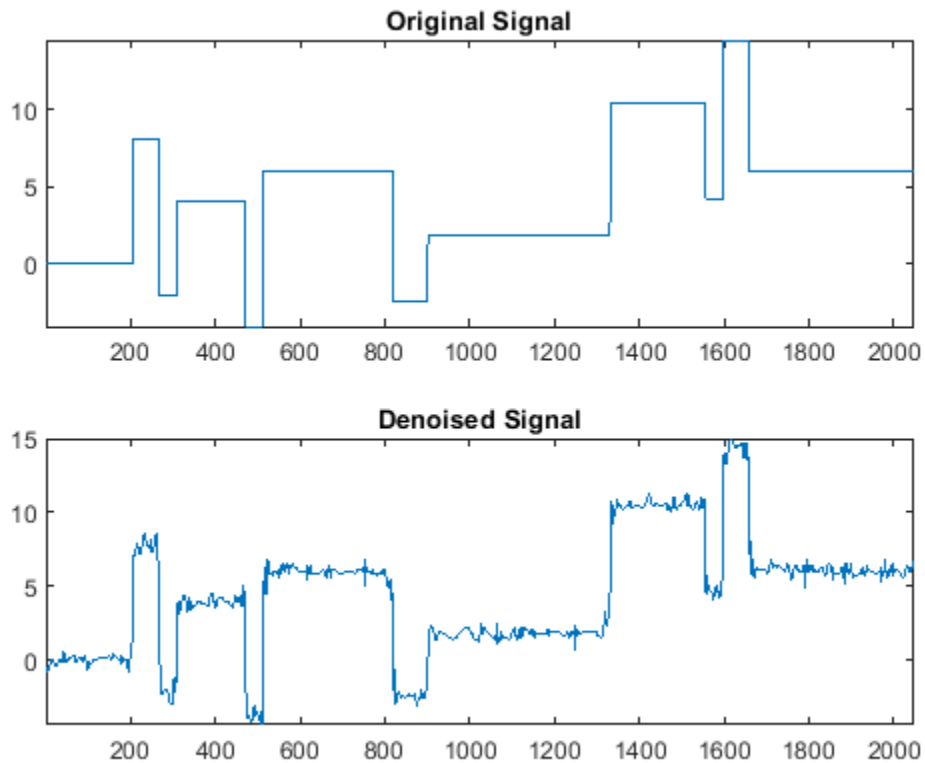
Denoised Signal

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')
```
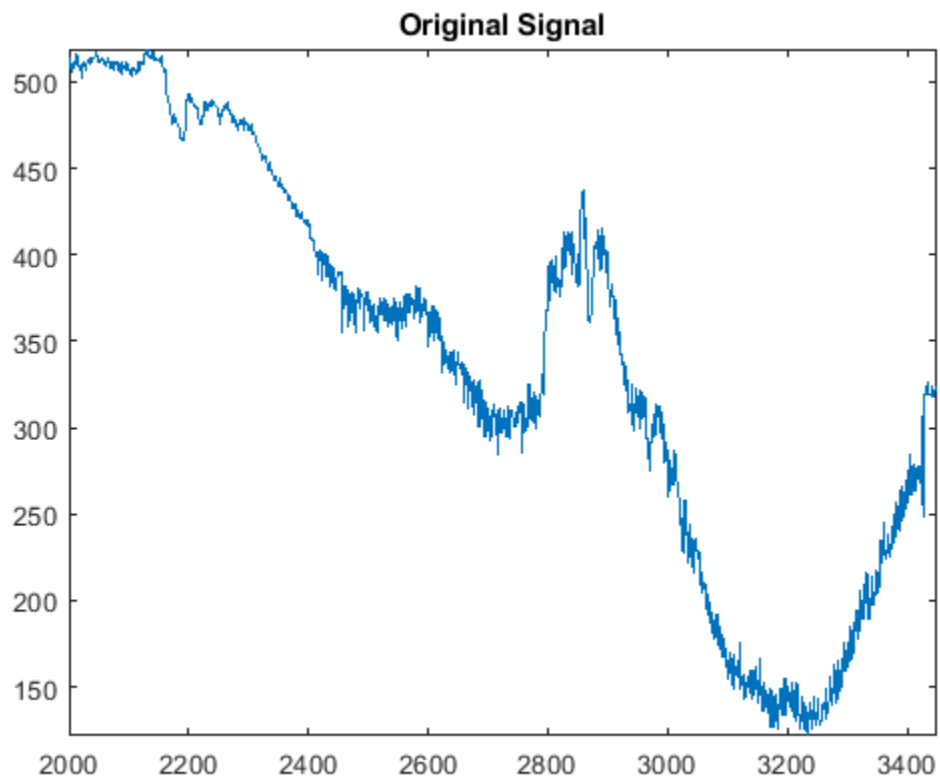
Denoise the signal using the db3 wavelet and a three-level wavelet decomposition and soft fixed form thresholding. To deal with the non-white noise, use level-dependent noise size estimation. Compare with the original signal.

```
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')
```

Original Signal

Denoised Signal
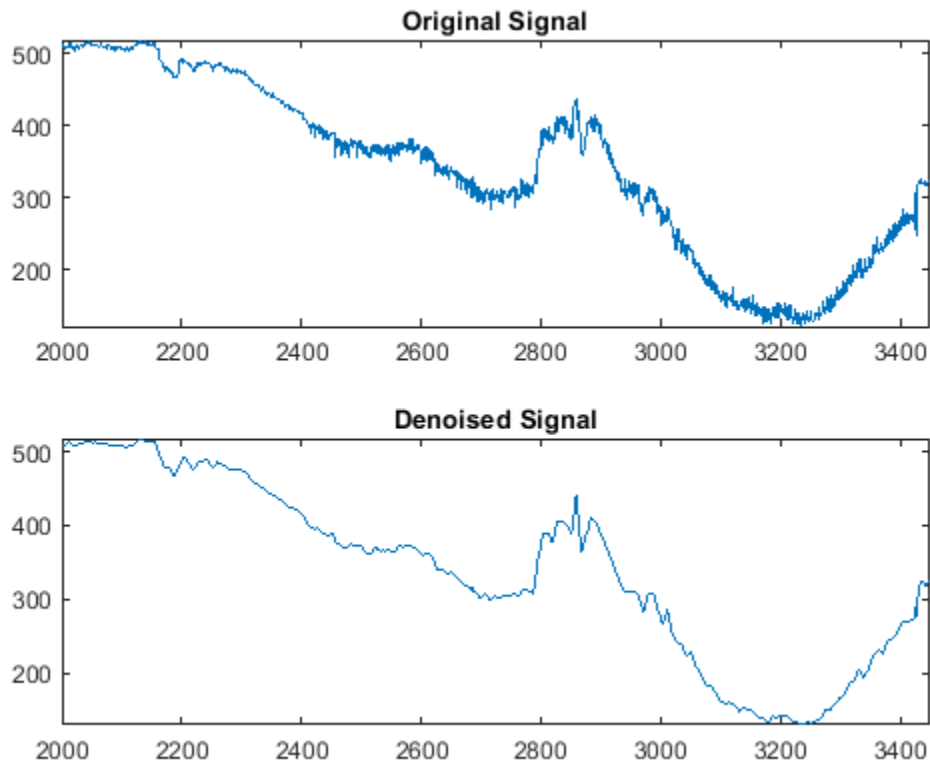
The result is quite good in spite of the time heterogeneity of the nature of the noise after and before the beginning of the sensor failure around time 2410.

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 `wdencmp`. 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 `ddencmp` 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] = ddencmp('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 = wdencmp('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')
```
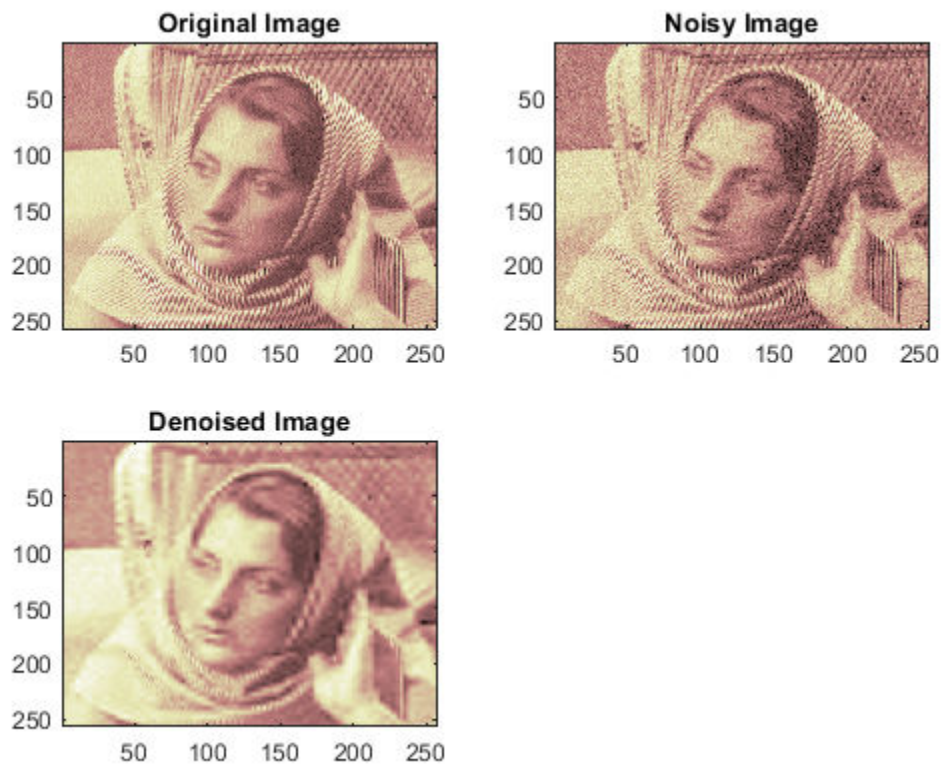

The denoised image compares well with the original image.

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')
```
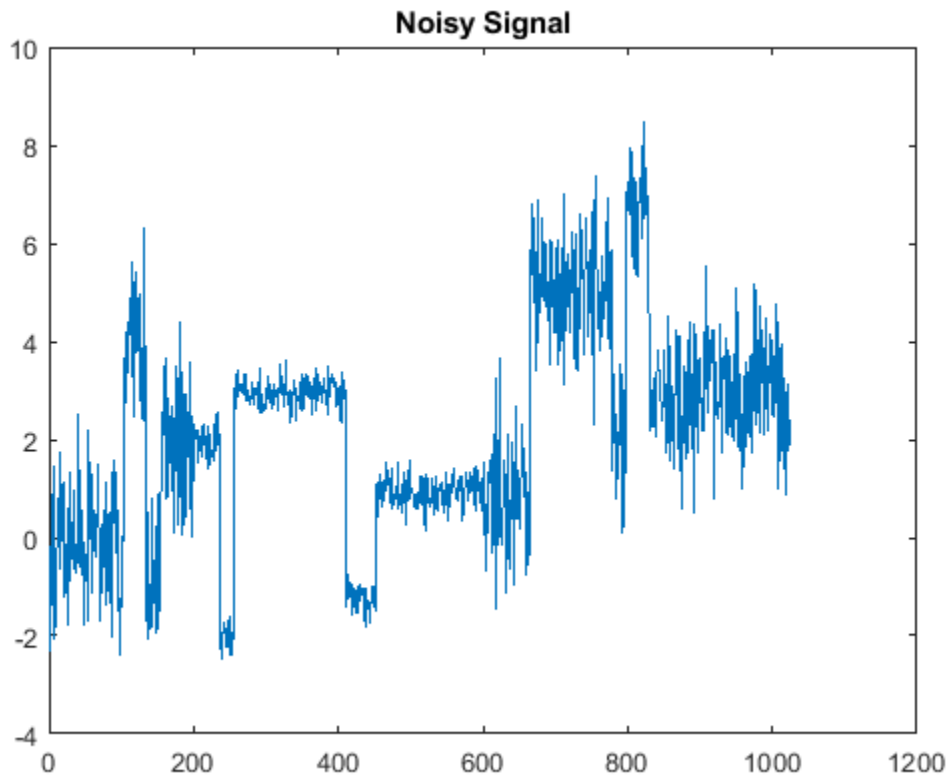
The aim of this example is to recover the two change points from the signal x.

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')
```
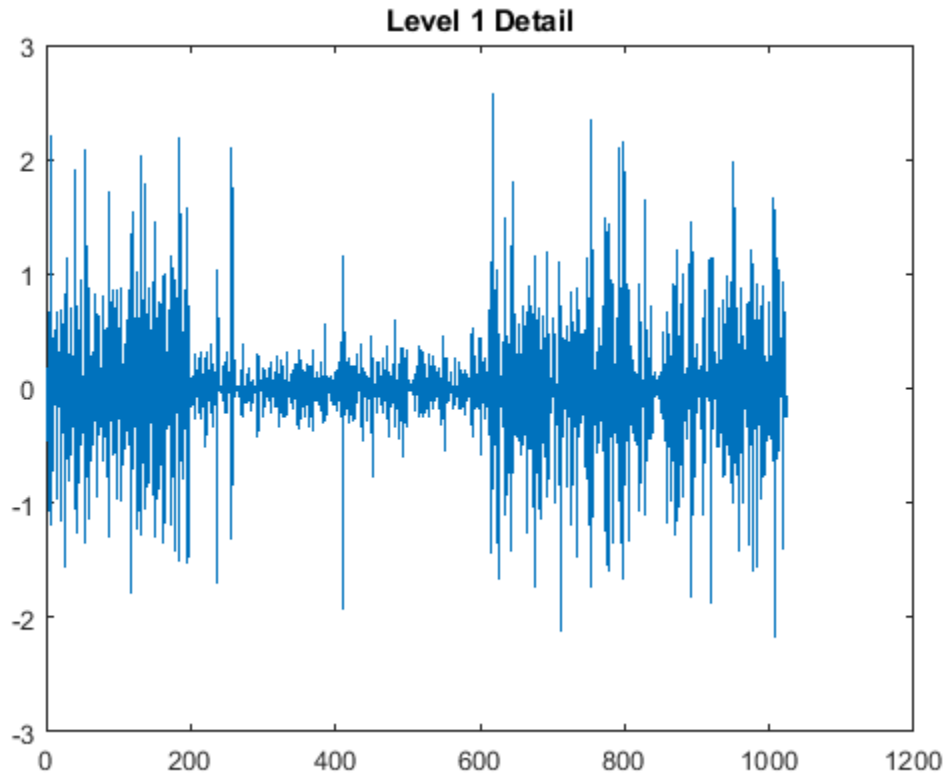
Level 1 Detail

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];
```
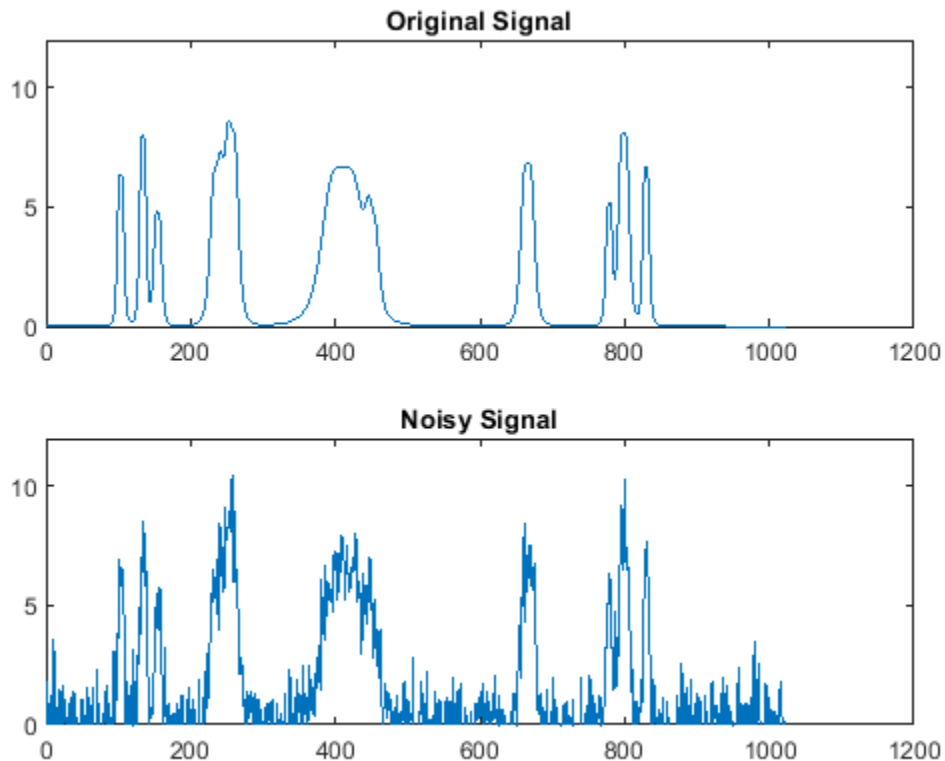
Denoise the signal down to level 4 using `wdenoise` with default settings. `wdenoise` uses the decimated wavelet transform. Plot the result along with the original signal.

```
xd = wdenoise(XN,4);
figure;
plot(X,'r')
hold on;
plot(xd)
legend('Original Signal','Denoised Signal','Location','NorthEastOutside')
axis tight;
hold off;
```
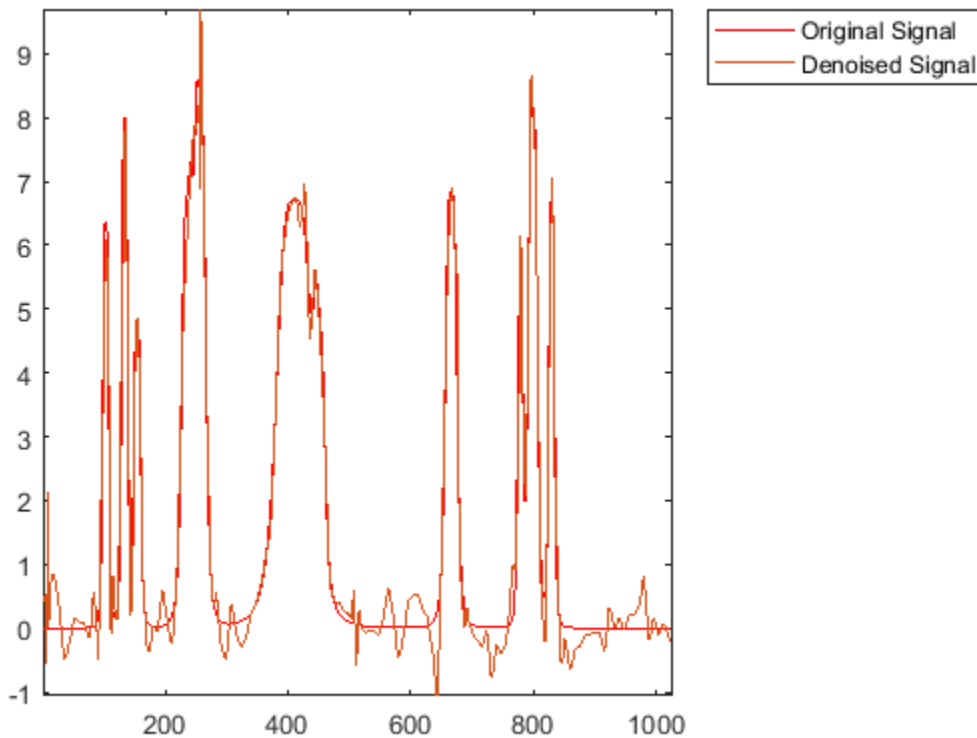

You can also denoise the signal using the undecimated wavelet transform. Denoise the signal again down to level 4 using the undecimated wavelet transform. Plot the result along with the original signal.

```
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;
```
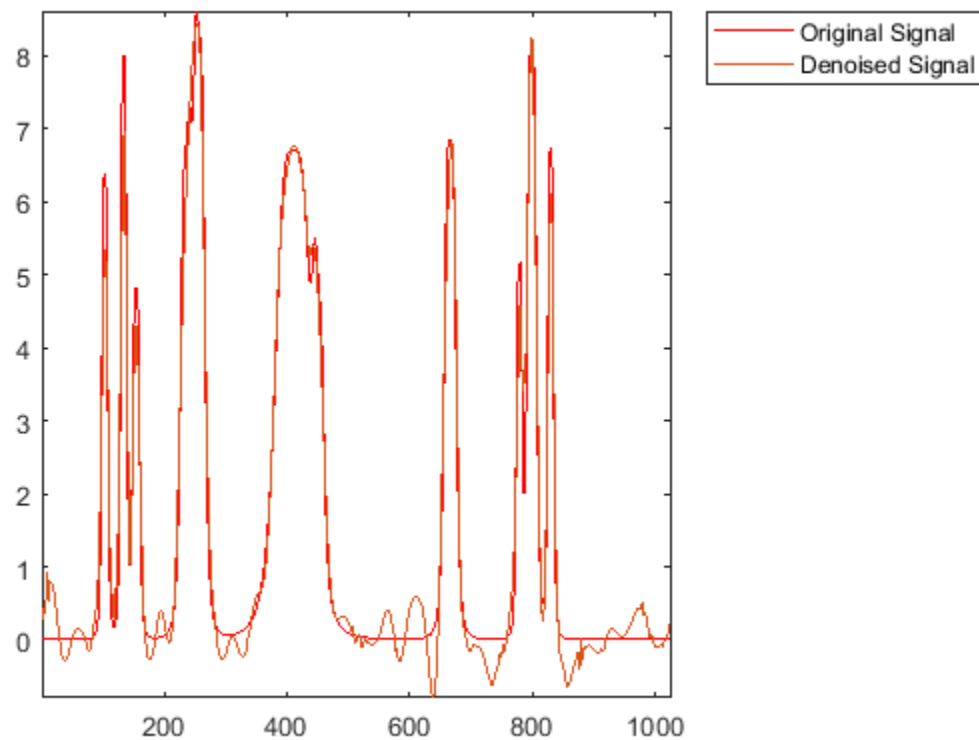
You see that in both cases, wavelet denoising has removed a considerable amount of the noise while preserving the sharp features in the signal. This is a challenge for Fourier-based denoising. In Fourier-based denoising, or filtering, you apply a lowpass filter to remove the noise. However, when the 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;
```
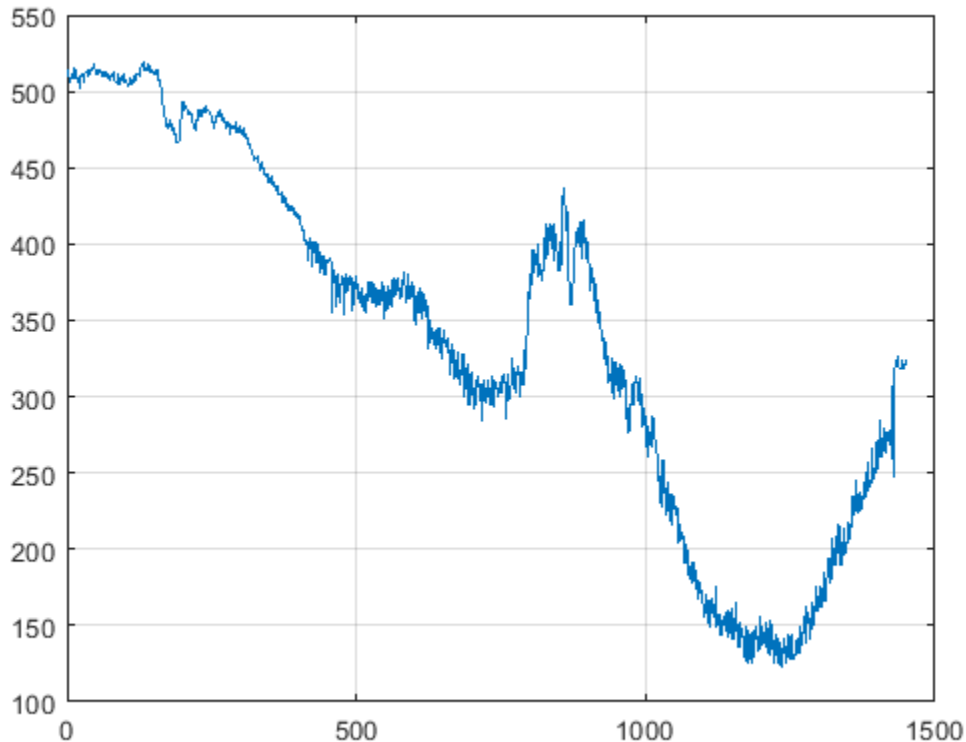
The signal appears to have more noise after approximately sample 500. Accordingly, you want to use different thresholding in the initial part of the signal. You can use `cmddenoise` to determine the optimal number of intervals to denoise and denoise the signal. In this example, use the 'db3' wavelet 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 = 2×2

        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;
```
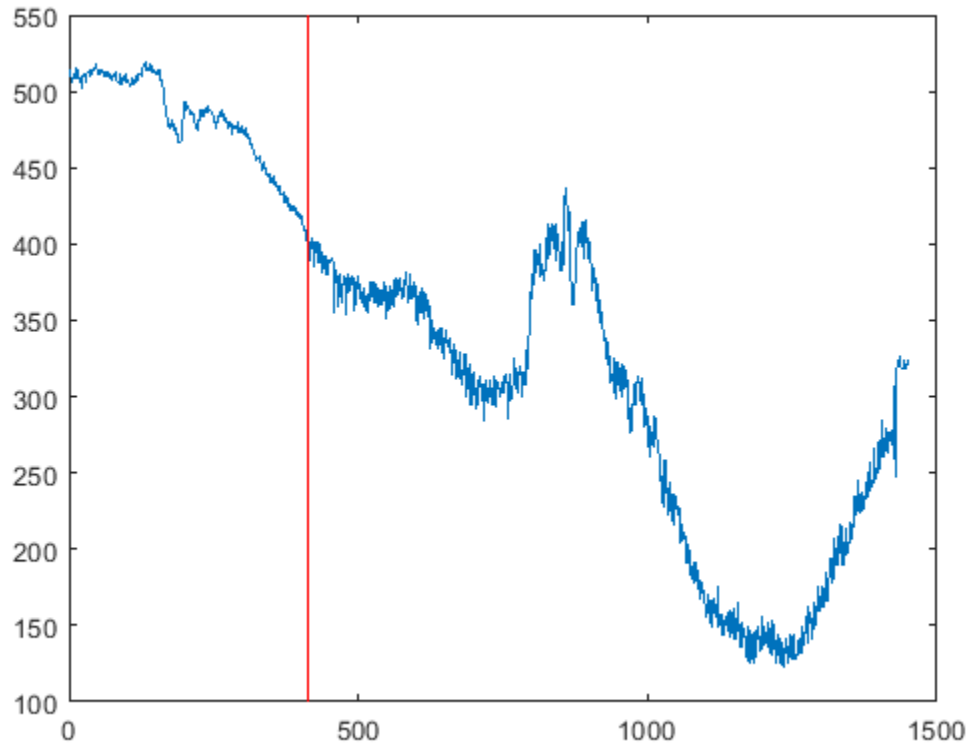

Plot the denoised signal.

```
plot(SIGDEN)
title('Denoised Signal')
```
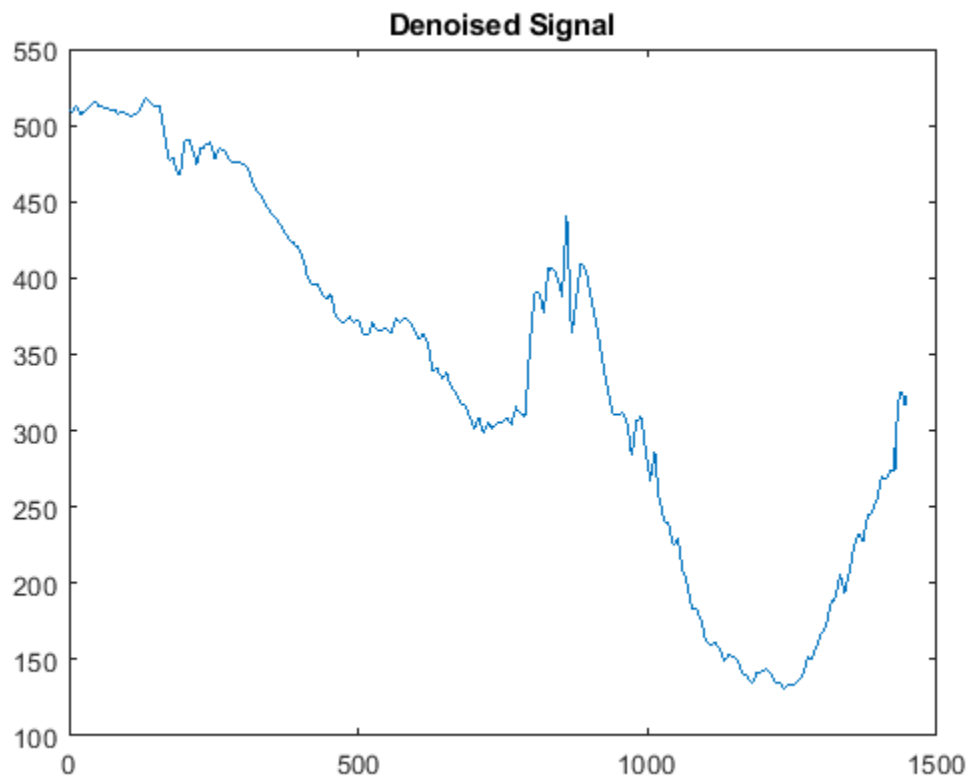
Denoise an Image

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('jump.mat')
wdenoise2(jump)
```
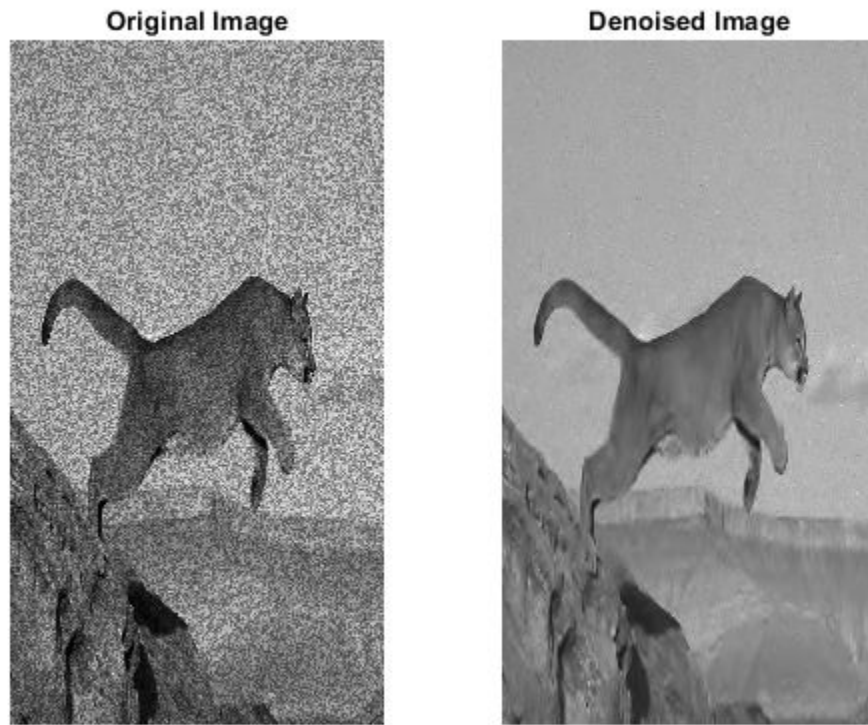
Note that edges in the image are not smoothed out by the denoising process.

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
```
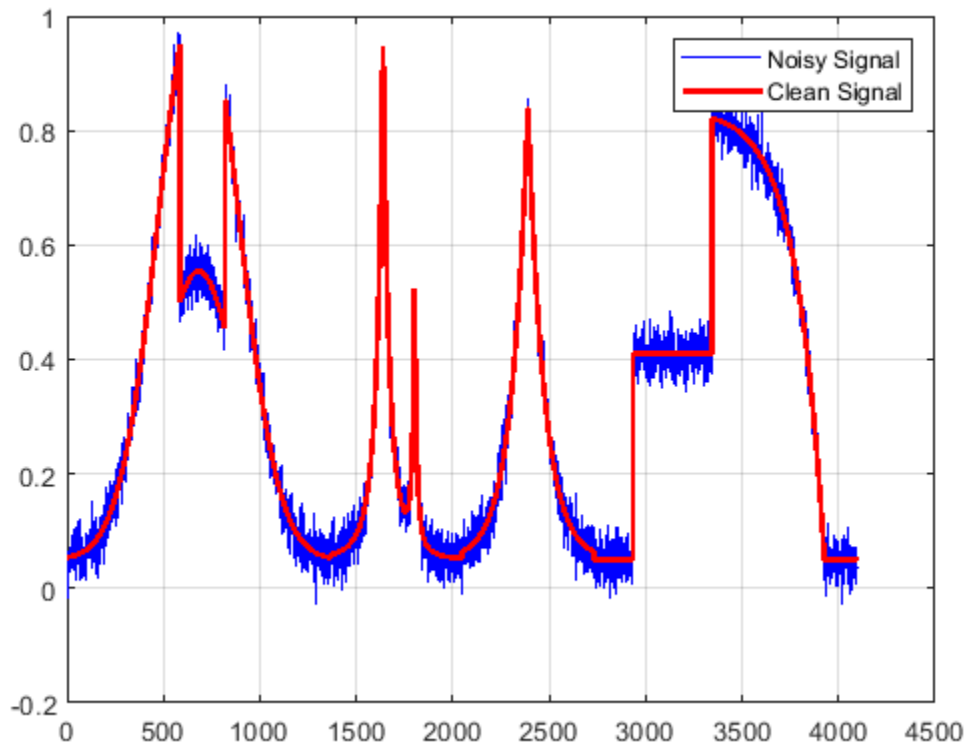

Open the Wavelet Signal Denoiser app. From the MATLAB Toolstrip, open the **Apps** tab and under **Signal Processing and Communications**, click **Wavelet Signal Denoiser**. You can also start the app by typing `waveletSignalDenoiser` at the MATLAB command prompt.

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**.
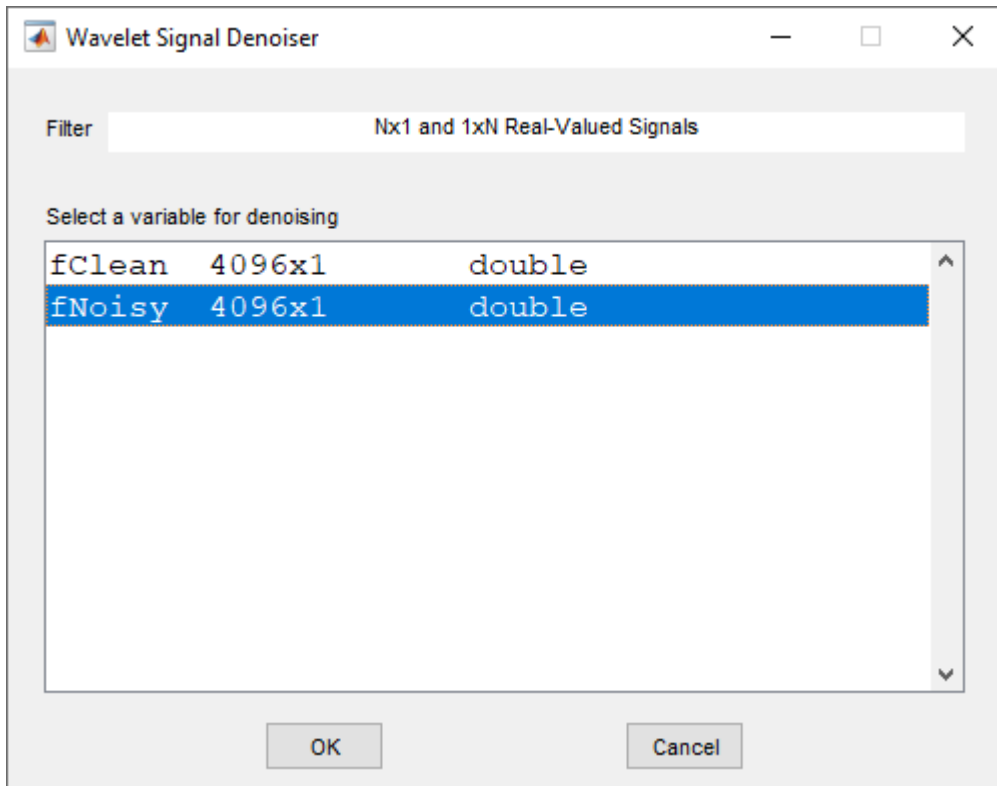
The app imports the noisy signal and immediately denoises it using default parameters.
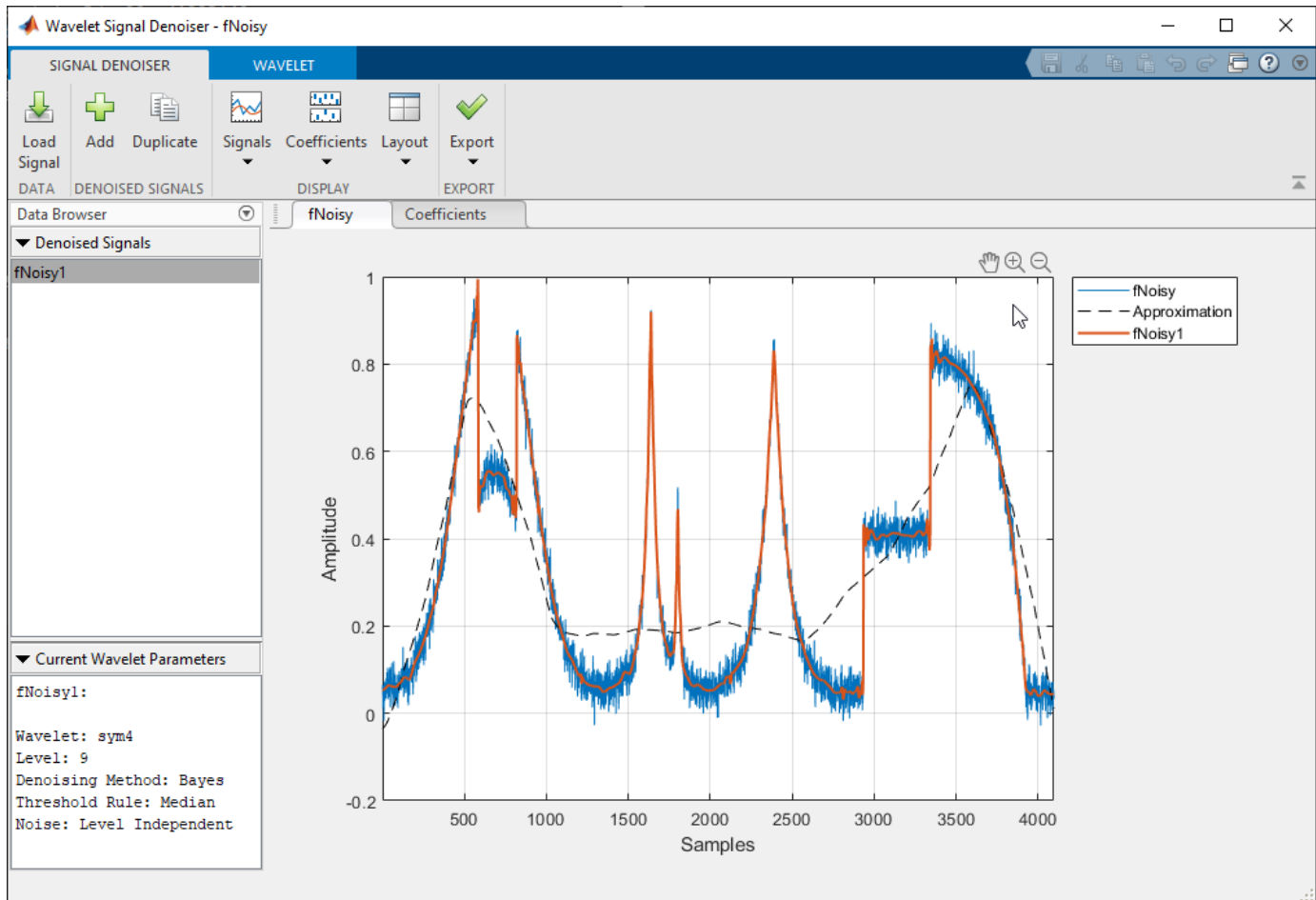
Examine Imported Data

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.
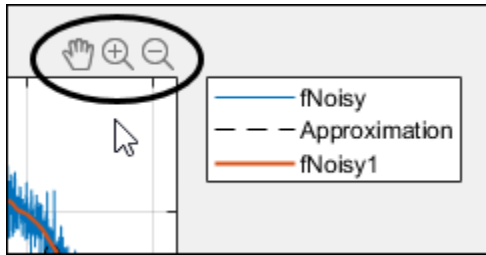
 Then select the desired action from the palette.

- 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.
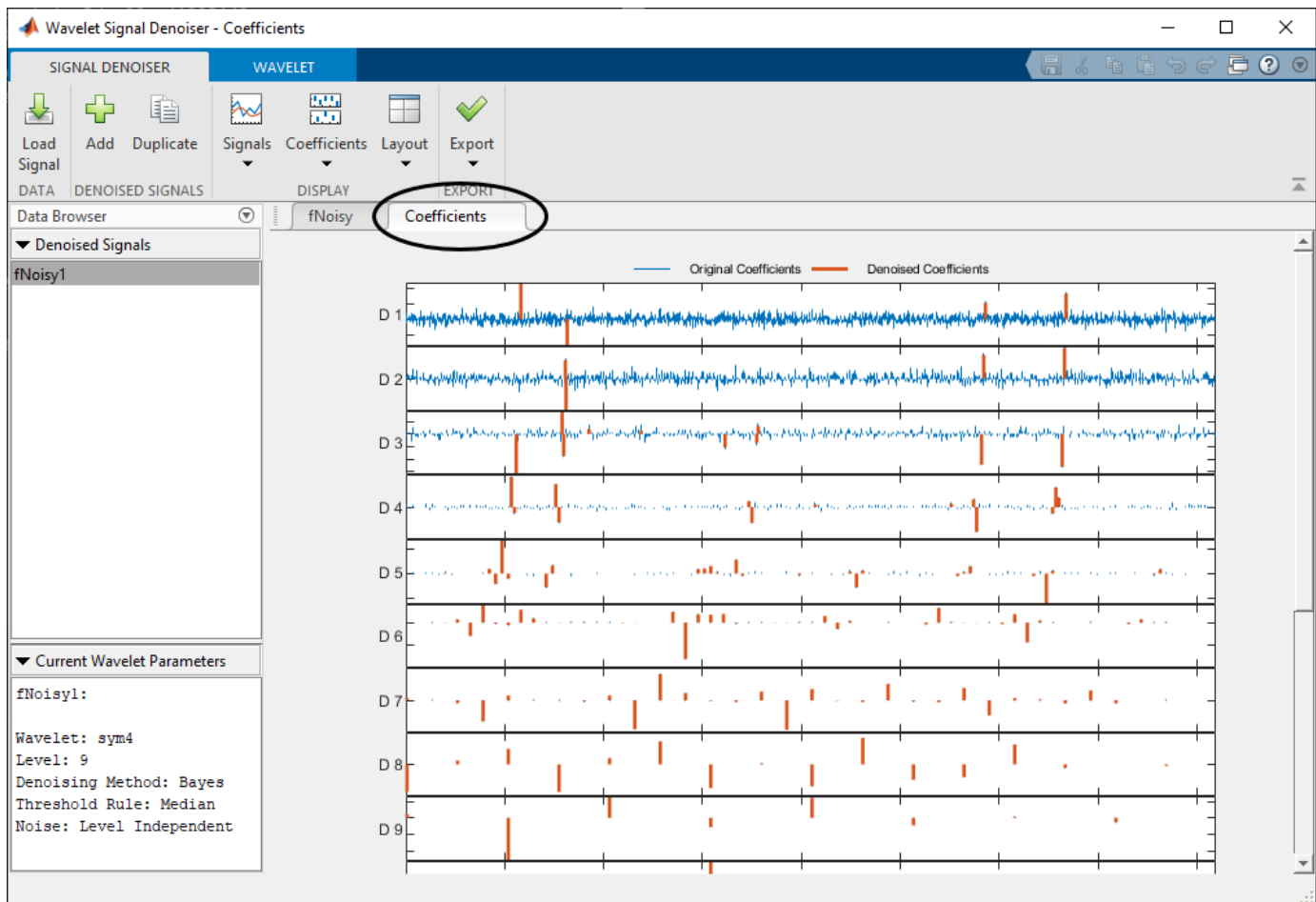
The coefficients in red are used to reconstruct the denoised signal. The **Current Wavelet Parameters** pane indicates that a 9-level wavelet decomposition was used to denoise the signal.

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.
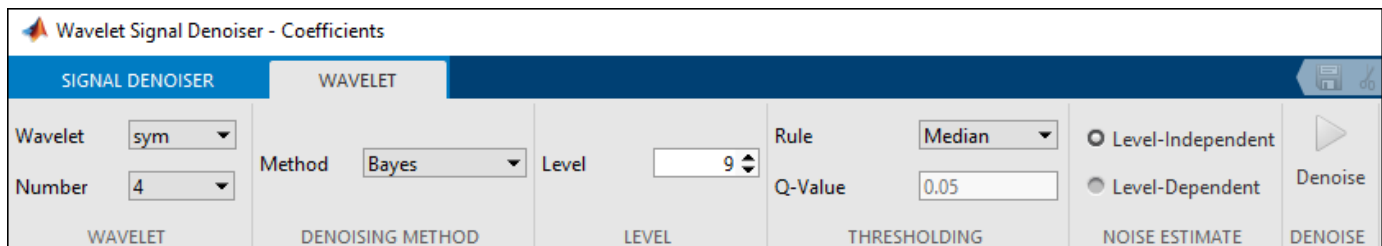
The values listed are the parameters used to create the denoised signal, fNoisy1. To modify the values of these settings, working from left to right in the toolstrip:

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.
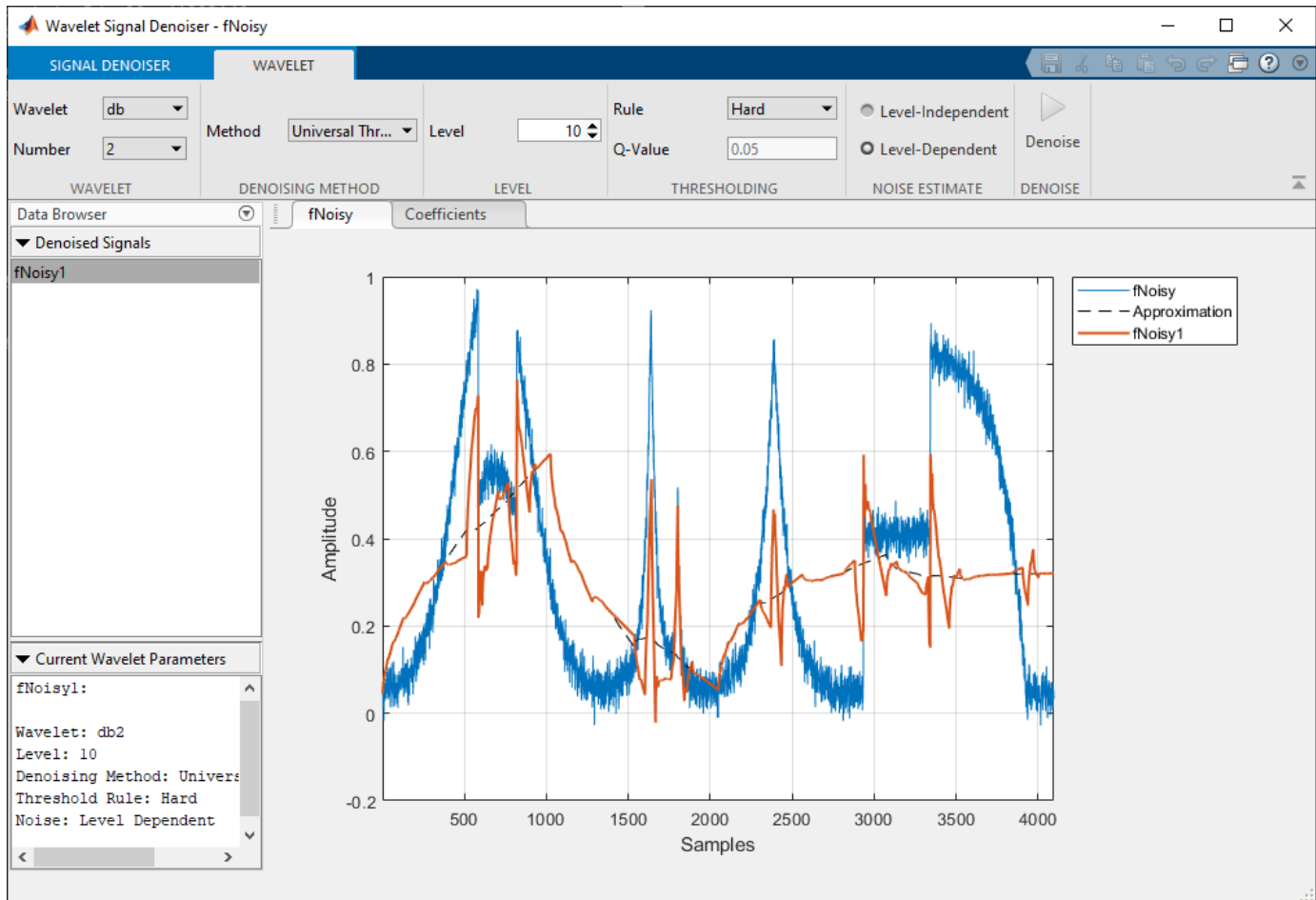
Note: All parameters are contextual. Possible values for one parameter can depend on the currently selected value of another parameter. You cannot make an incompatible selection. For example, when the denoising method is FDR, there is only possible thresholding rule: hard. In this instance, no other 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.
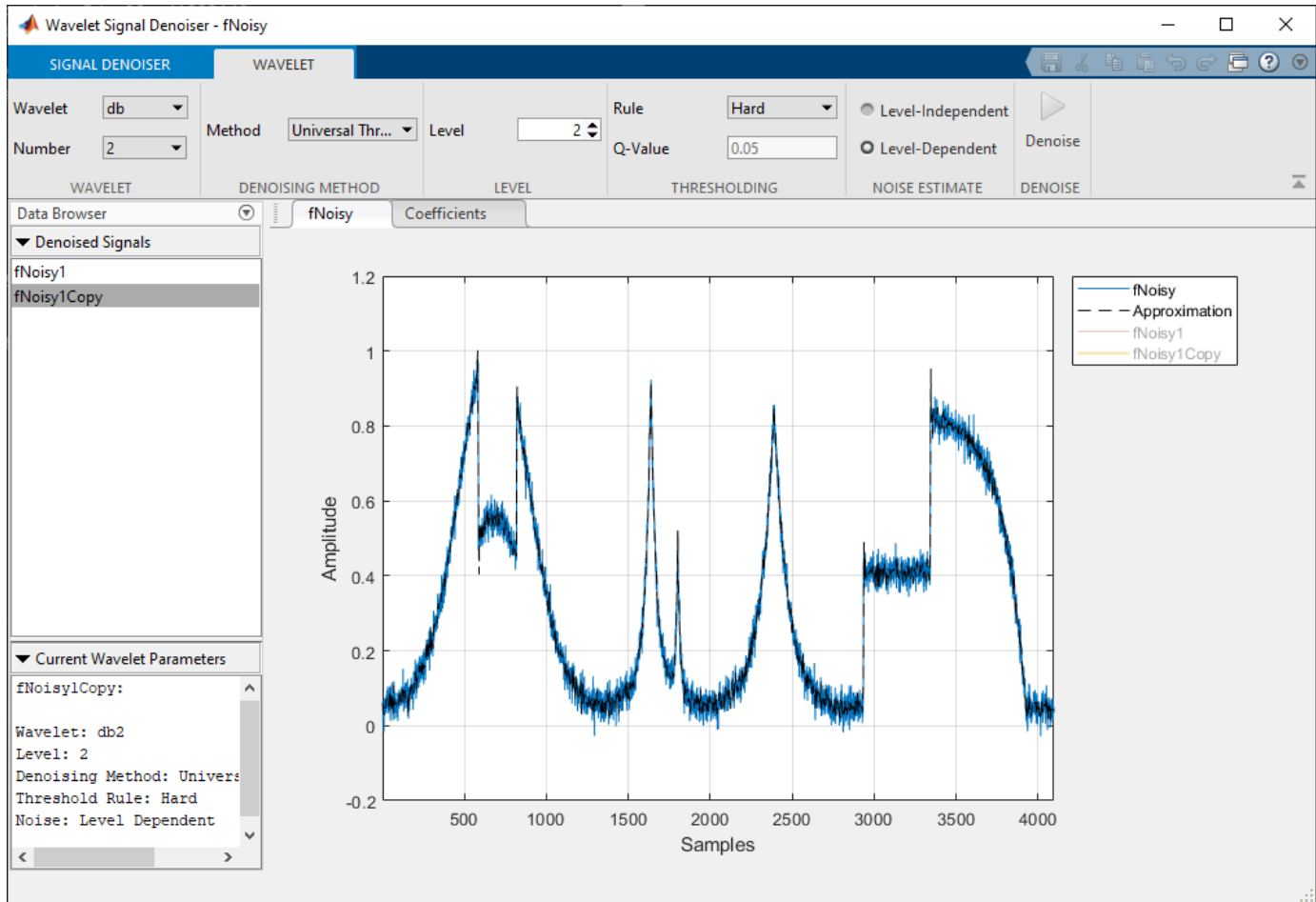
The dashed line in the plot represents the approximation. Because `fNoisy1Copy` is highlighted in the **Denoised Signals** list, the approximation plotted is the result of a two-level wavelet decomposition. The approximation is relatively noisy. Now select `fNoisy1` in the list. The approximation of a 10-level wavelet decomposition is different.
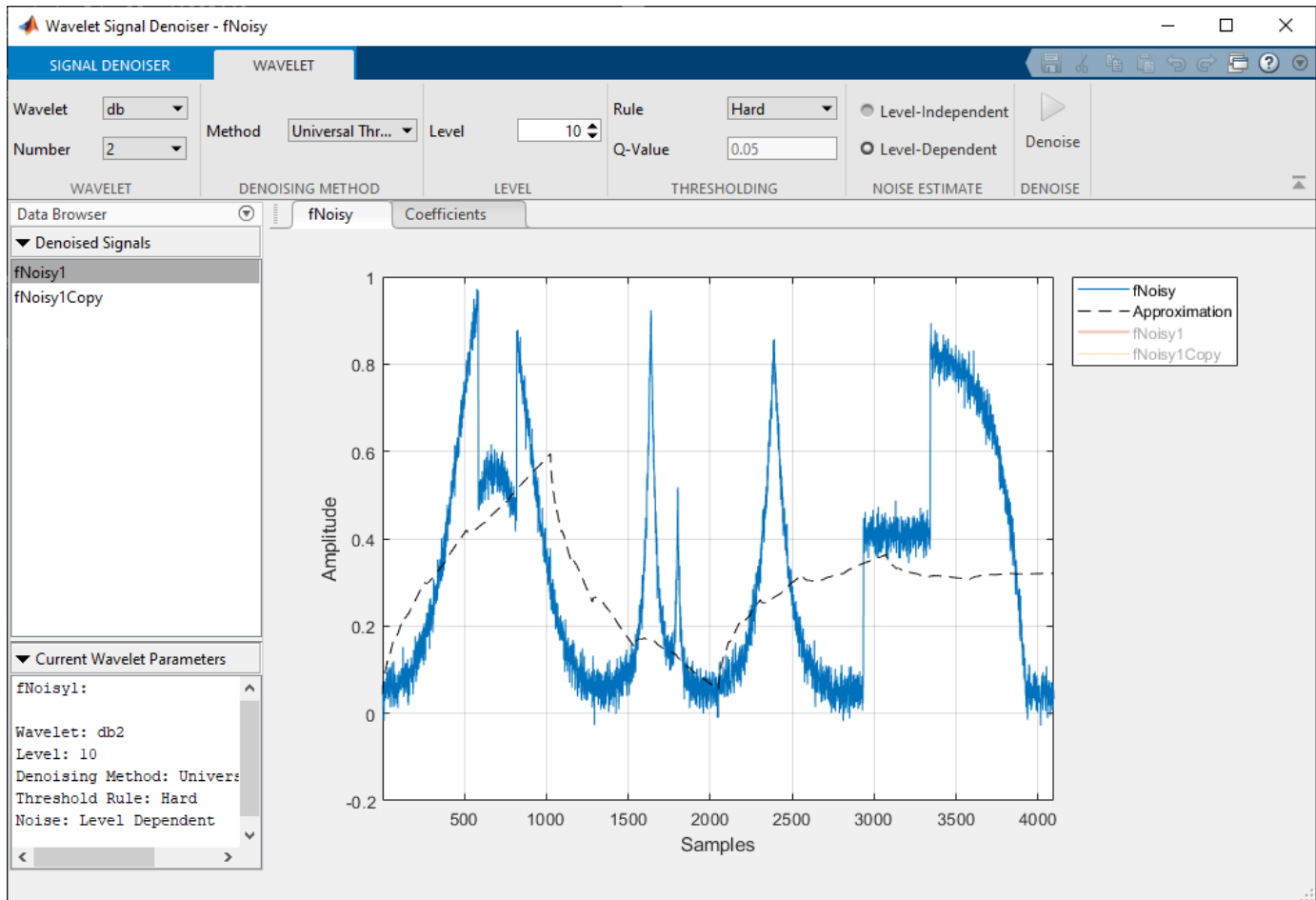
Restore Original Parameters

You can always return to using the original default parameters by adding a new denoised signal. From the toolstrip, 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
```

Original Time Series

Denoised Time Series
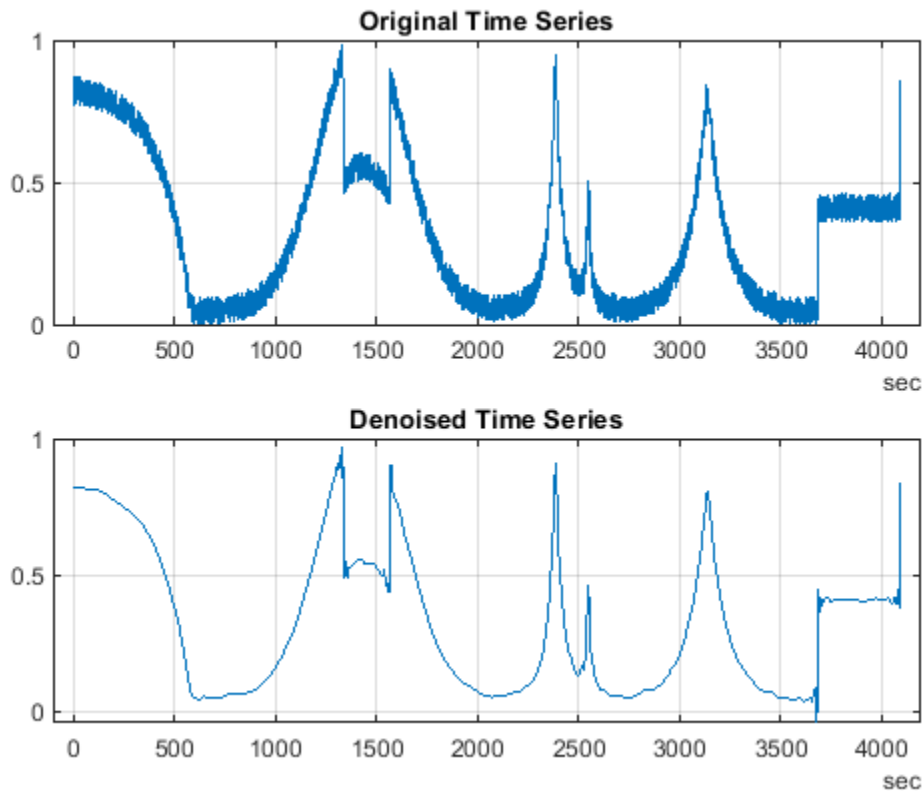
Export Denoised Signal

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.

See Also

Functions
wdenoise | wdenoise2

Apps
Wavelet Signal Denoiser

More About

- "Wavelet Denoising" on page 6-18
- "Smoothing Nonuniformly Sampled Data" on page 12-41
- "Translation Invariant Wavelet Denoising with Cycle Spinning" on page 6-36

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 $\dfrac{N\left|\left|X\right|\right|_2^2}{\sigma}$ where N is the length of the signal, $\left|\left|X\right|\right|_2^2$ is the squared L2 norm, and σ^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')
```
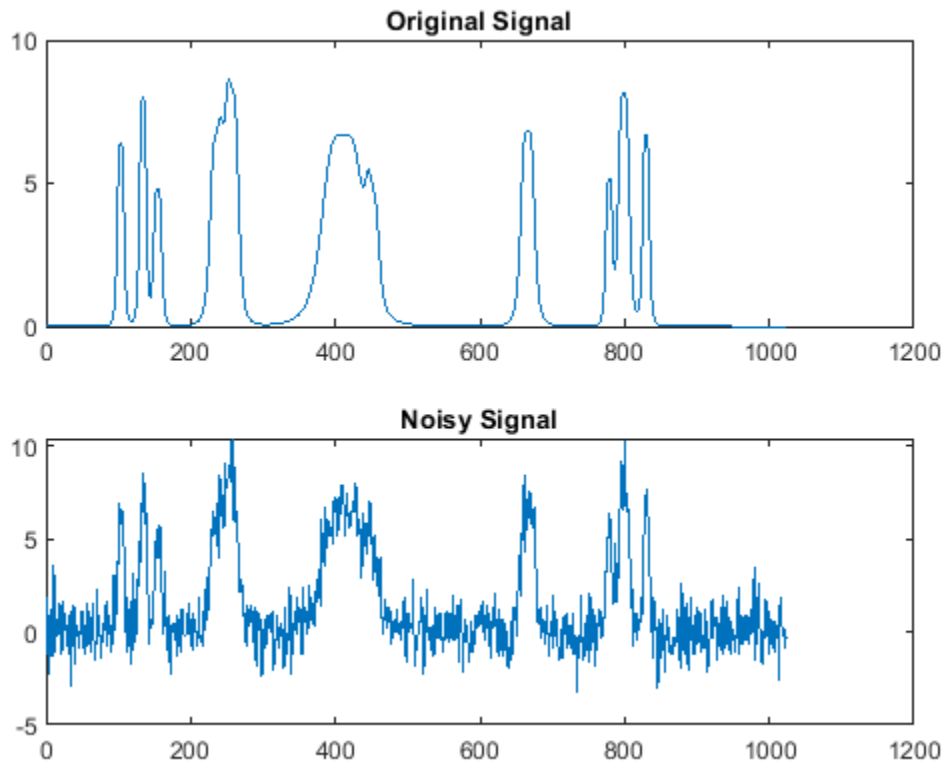
Denoise the signal using cycle spinning with 15 shifts, 7 to the left and 7 to the right, including the zero-shifted signal. Use `wdenoise` with default settings. By default, `wdenoise` uses Daubechies' least-asymmetric wavelet with four vanishing moments, `sym4`. Denoising is down to the minimum of `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
```
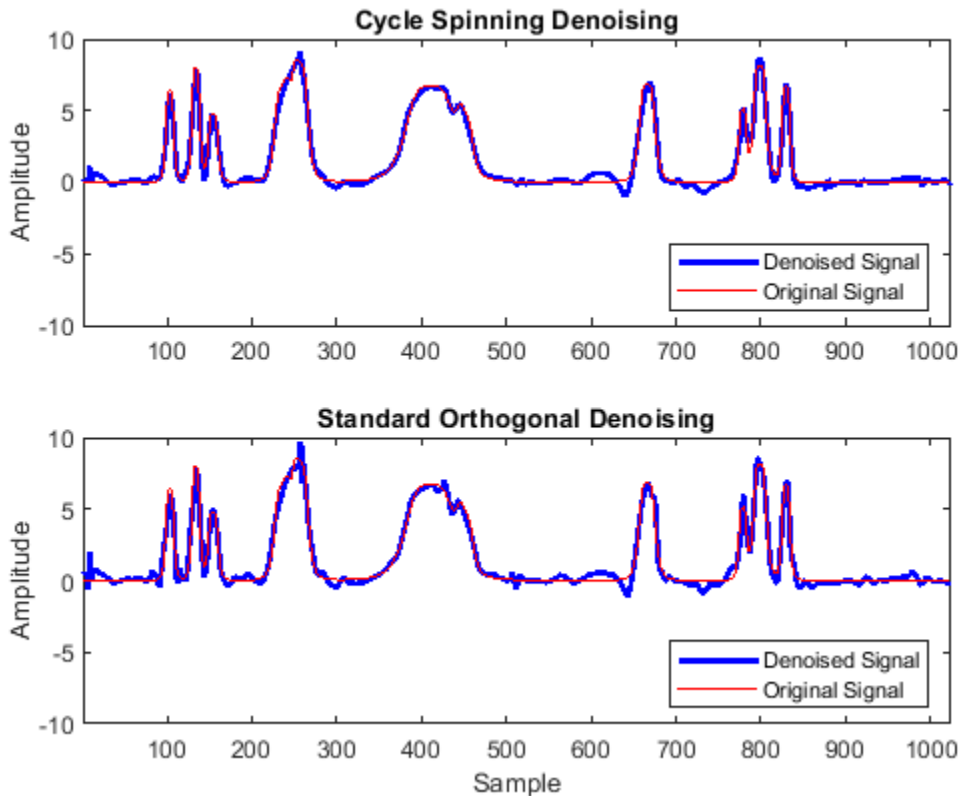


```
absDiffDWT = norm(X-xd,2)

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`.
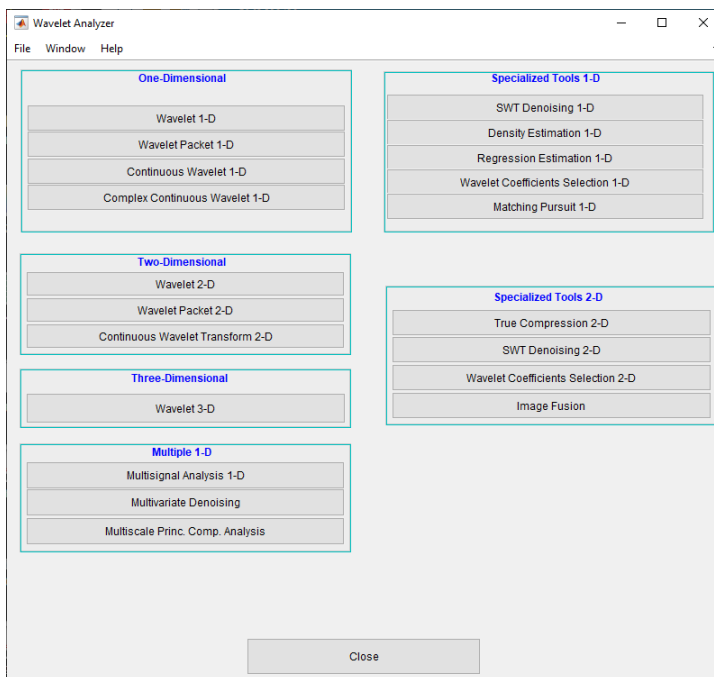
The **Wavelet Analyzer** appears.

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.
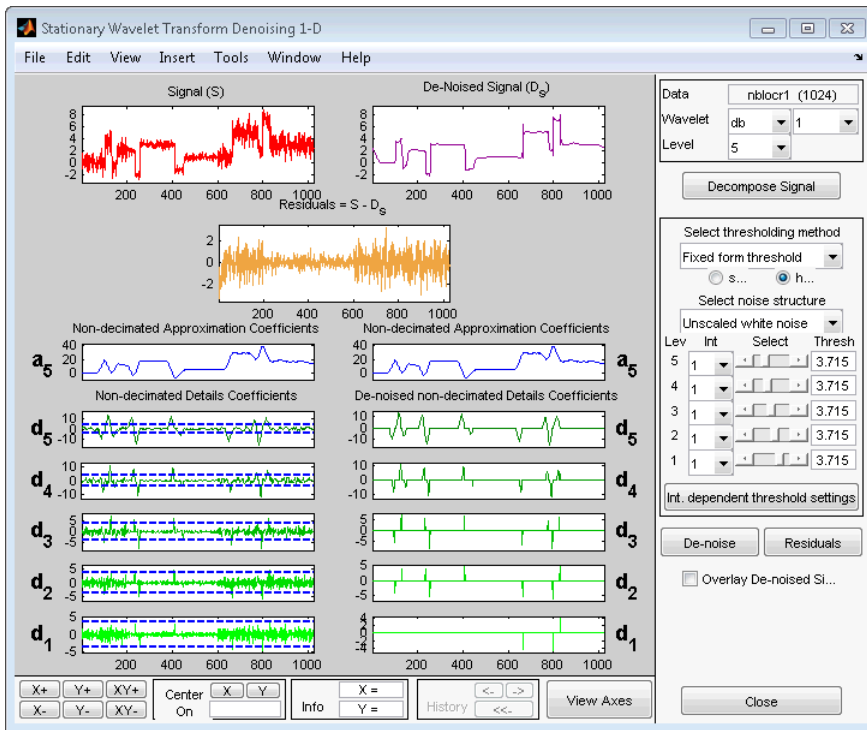
The result is not satisfactory. The denoised signal remains noisy before position 200 and after position 700. This illustrates the limits of the classical denoising strategies. In addition, the residuals obtained during the last trials clearly suggest to try a local thresholding strategy.

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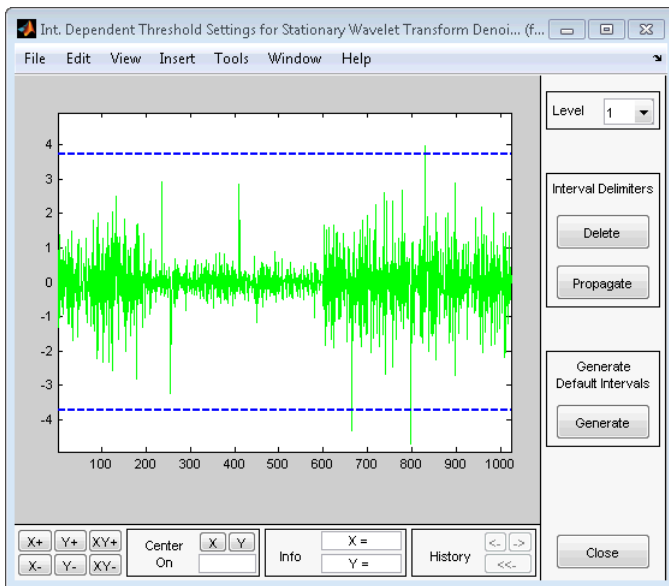
Click the **Generate** button. After a pause for computation, the tool displays the default intervals associated with adapted thresholds.
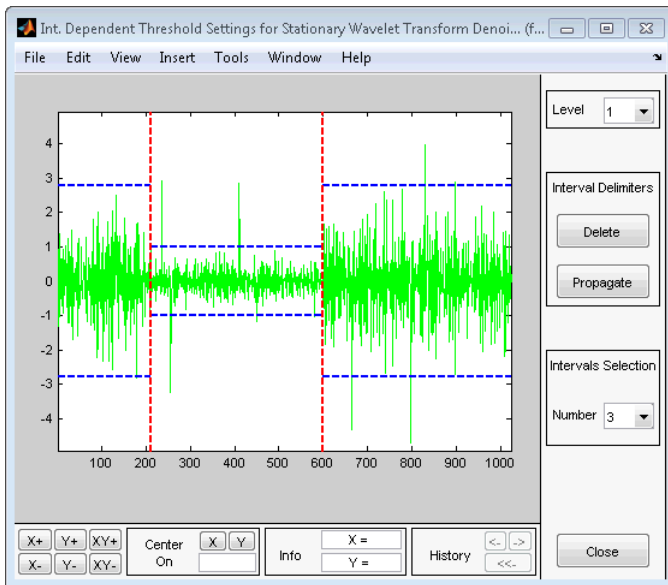
Three intervals are proposed. Since the variances for the three intervals are very different, the optimization program easily detects the correct structure. Nevertheless, you can visualize the intervals proposed for a number of intervals from 1 to 6 using the **Select Number of Intervals** menu (which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

Denoise with Interval-Dependent Thresholds

Click the **Close** button in the **Int. Dependent Threshold Settings for ...** window. When the **Update thresholds** dialog box appears, click **Yes**. The **SWT 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.
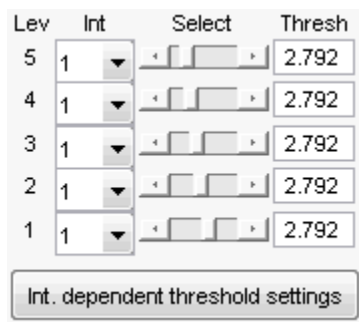
Note that you can also change the interval limits by holding down the left mouse button over the vertical dotted red lines, and dragging them.

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.
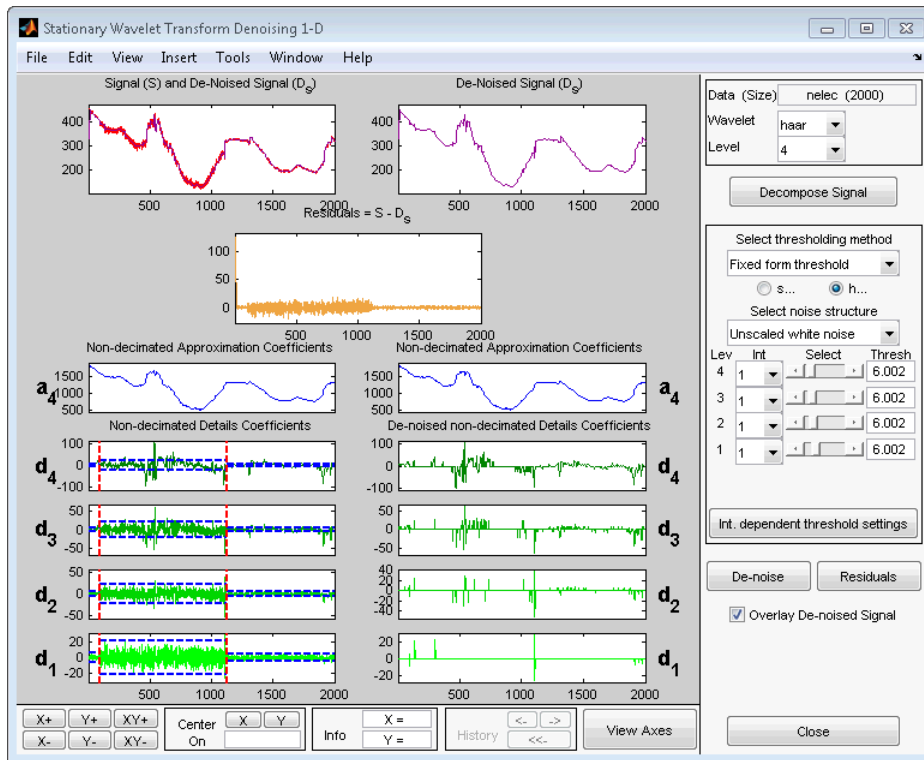
Importing and Exporting Information from the Wavelet Analyzer App

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 `wmulden` 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
```
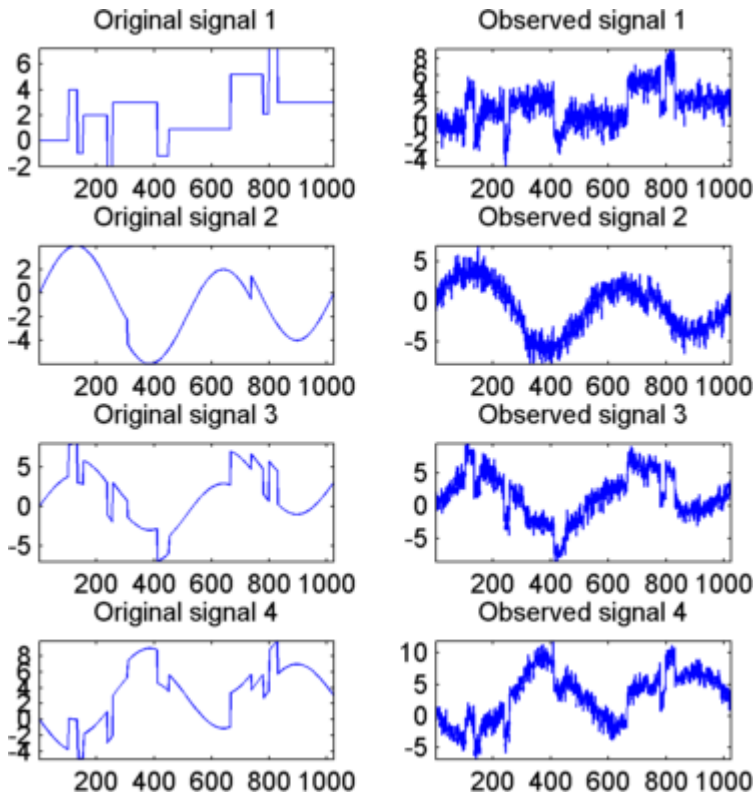

The true noise covariance matrix is given by

```
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
```
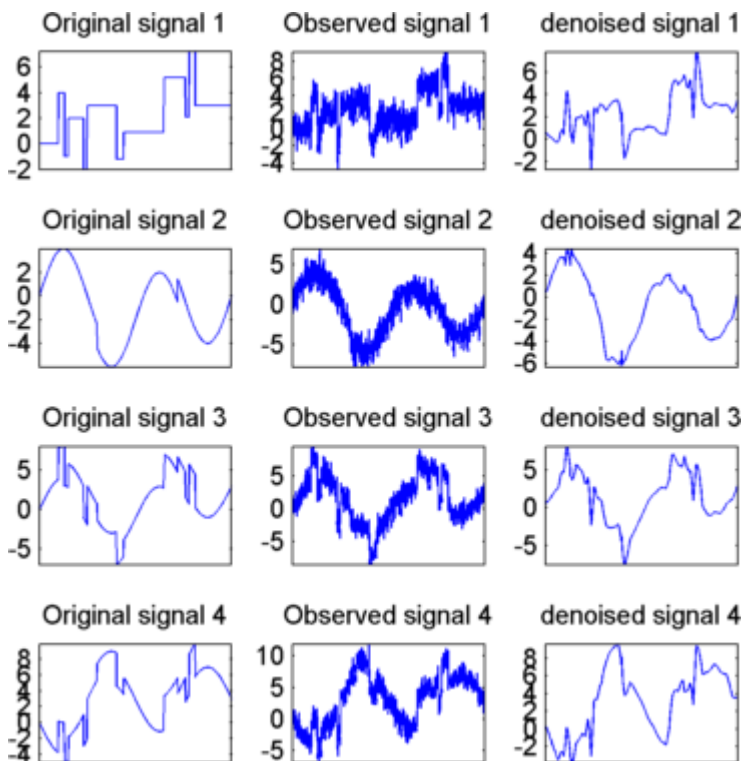

5   Improve the first result by retaining fewer principal components.

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
```
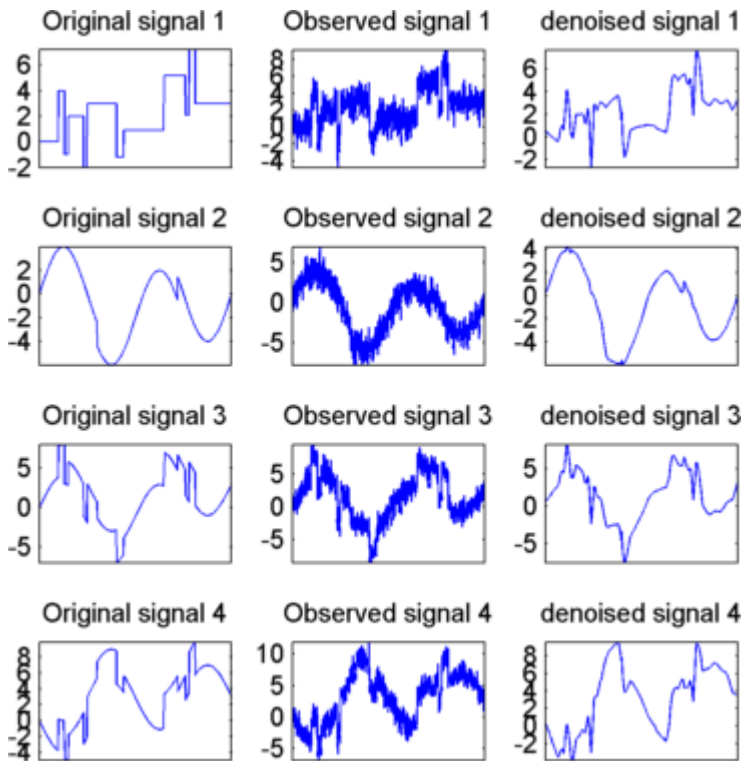
The results are better than those previously obtained. The first signal, which is irregular, is still correctly recovered, while the second signal, which is more regular, is denoised better after this second stage of PCA.

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.
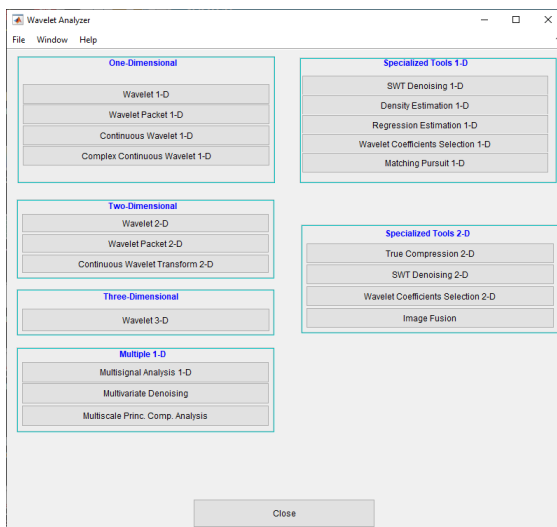
2 Click **Multivariate Denoising** to open the **Multivariate Denoising** portion of the app.

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.
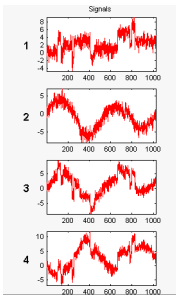
These signals are noisy versions from simple combinations of the two original signals. The first one is "Blocks" which is irregular and the second is "HeavySine" which is regular except around time 750. The other two signals are the sum and the difference between the original signals. Multivariate Gaussian 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.
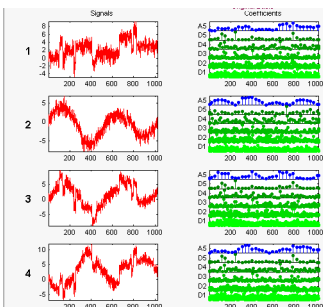
Select **Noise Adapted Basis** to display the signals and their coefficients in the noise-adapted basis.

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.

Robust Noise Covariance Σ

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

Eigenvectors defining the V

−0.5319	−0.3015	−0.7831	−0.1139
−0.4713	−0.6147	0.5900	−0.2277
−0.4688	0.6866	0.1325	−0.5396
−0.5245	0.2445	0.1455	0.8025

Eigenval Λ

3.2259
0.5511
0.1829
0.2678

$$\Sigma = V \Lambda V^{\mathrm{T}}$$

Close
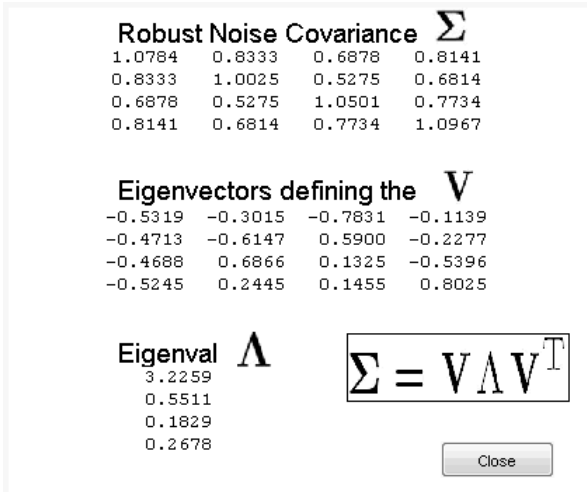
Eigenvectors define the change of basis, and eigenvalues are the variances of uncorrelated noises in the new basis.

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**.
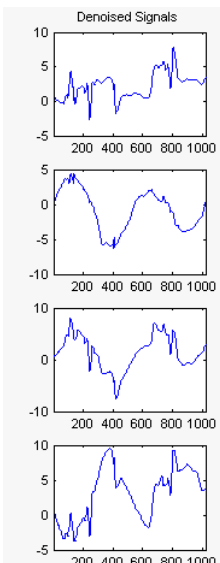
The results are satisfactory. Both of the two first signals are correctly recovered, but they can be improved by getting more information about the principal components. Click **More on Principal Components**.
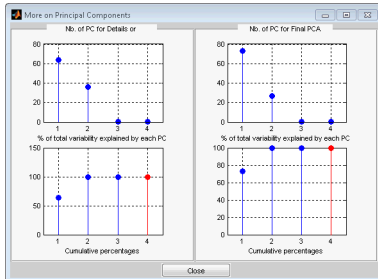
A new figure displays information to select the numbers of components to keep for the PCA of approximations and for the final PCA after getting back to the original basis. You can see the percentages of variability explained by each principal component and the corresponding cumulative plot. Here, it 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**.
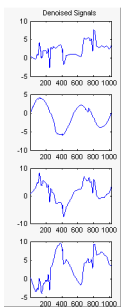
The results are better than those previously obtained. The first signal, which is irregular, is still correctly recovered. The second signal, which is more regular, is denoised better after this second stage of PCA. You can get more information by clicking **Residuals**.

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: 'sqtwolog'
    thrTYPE: 's'
```

The thresholds are encoded in thrVAL. For j from 1 to 5, thrVAL(j) contains the value used to threshold the detail coefficients at level j. The thresholding method is given by thrMETH and the thresholding mode is given by thrTYPE.

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
```
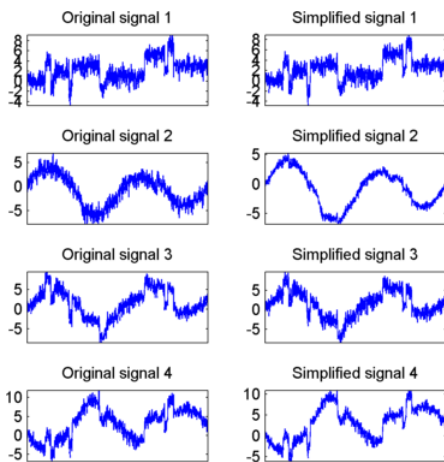

The results from a compression perspective are good. The percentages reflecting the quality of column reconstructions given by the relative mean square errors are close to 100%.

```
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, `npc(d)` is the number of retained noncentered principal components (PCs) for details at level d. The number of retained noncentered PCs for approximations at level 5 is `npc(6)`, and `npc(7)` is the number of retained PCs for final PCA after wavelet reconstruction. As expected, the rule keeps two principal components, both for the PCA approximations and the final PCA, but one principal component is kept for details at each level.

To suppress the details at levels 1 to 3, update the `npc` argument as follows:

```
npc(1:3) = zeros(1,3);

npc

npc =
0     0     0     1     1     2     2
```

Then, perform multiscale PCA again:

```
[x_sim, qual, npc] = wmspca(x, level, wname, npc);
```

5 Display the original and final simplified signals:

```
kp = 0;
for i = 1:4
    subplot(4,2,kp+1), plot(x (:,i)); 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
```
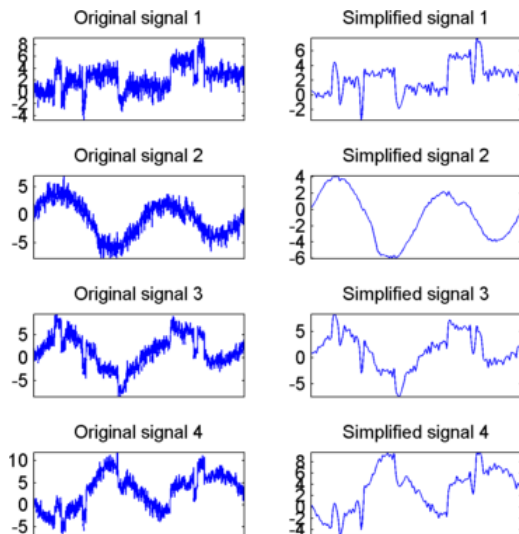

As shown, the results are improved.

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.
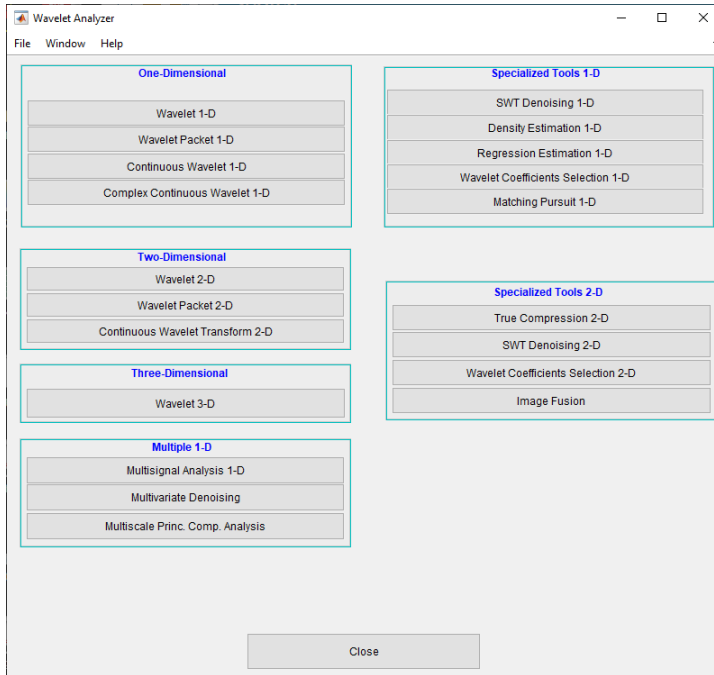
2 Click **Multiscale Princ. Comp. Analysis** to open the Multiscale Principal Components Analysis tool in the app.

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.
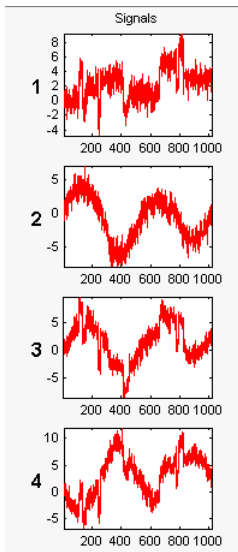
These signals are noisy versions from simple combinations of the two original signals, Blocks and HeavySine and their sum and difference, each with added multivariate Gaussian white noise.

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.
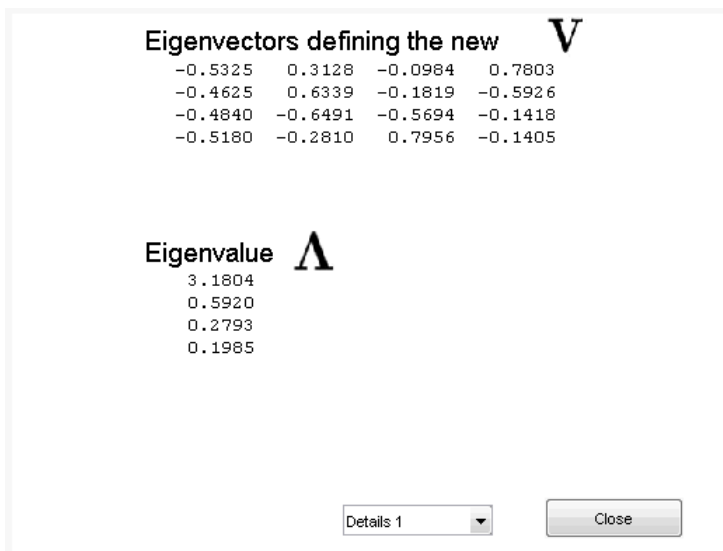
You can change the level or select the coarser approximations or the reconstructed matrix to investigate the different bases. When you finish, click **Close**.

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**.
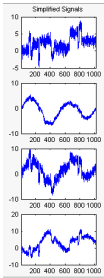
The results are good from a compression perspective.

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 `0` as the **Nb. of non-centered PC** and click **Apply**.
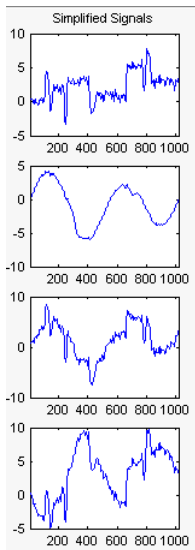
The results are better than those previously obtained. The first signal, which is irregular, is still correctly recovered, while the second signal, which is more regular, is denoised better after this second stage of PCA. You can get more information by clicking **Residuals**.

Importing and Exporting from the 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 6-63) and for an image (see "Image Compression" on page 6-64).

```
% Load electrical signal and select a part.
load leleccum; indx = 2600:3100;
x = leleccum(indx);
% Perform wavelet decomposition of the signal.
n = 3; w = 'db3';
[c,l] = wavedec(x,n,w);
% Compress using a fixed threshold.
thr = 35;
keepapp = 1;
[xd,cxd,lxd,perf0,perfl2] = ...
    wdencmp('gbl',c,l,w,n,thr,'h',keepapp);
```
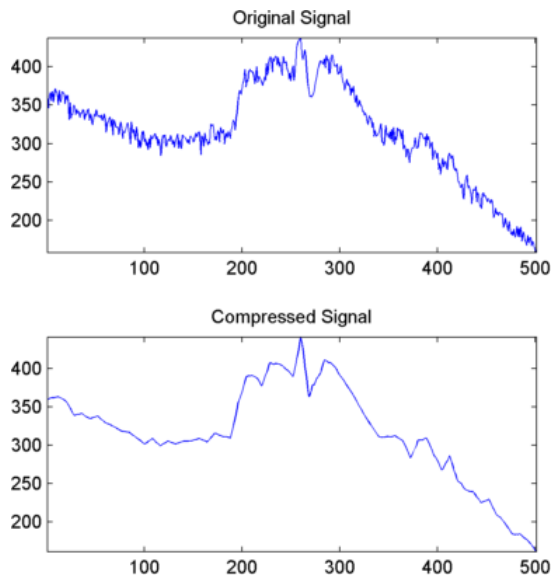
Signal Compression

The result is quite satisfactory, not only because of the norm recovery criterion, but also on a visual perception point of view. The reconstruction uses only 15% of the coefficients.

```
% Load original image.
load woman; x = X(100:200,100:200);
nbc = size(map,1);

% Wavelet decomposition of x.
n = 5; w = 'sym2'; [c,l] = wavedec2(x,n,w);

% Wavelet coefficients thresholding.
thr = 20;
keepapp = 1;
[xd,cxd,lxd,perf0,perfl2] = ...
                wdencmp('gbl',c,l,w,n,thr,'h',keepapp);
```

Threshold = 20
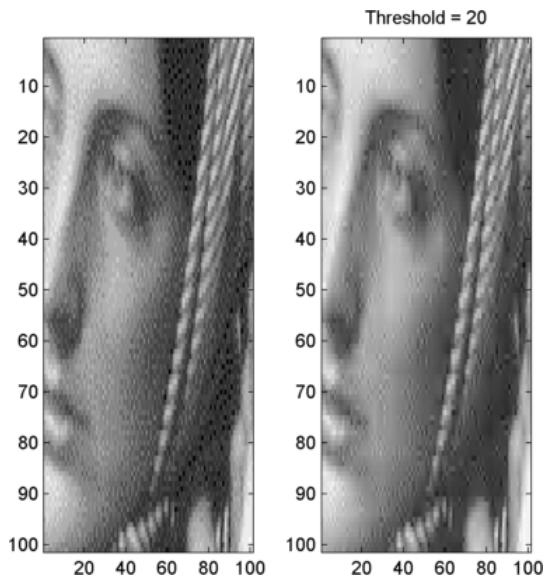
Image Compression

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

$$\frac{100 * (\text{vector-norm(coeffs of the current decomposition, 2)})^2}{(\text{vector-norm(original signal, 2)})^2}$$

When compressing using biorthogonal wavelets, the previous definition is not convenient. We use instead the *Energy ratio* in percentage defined by

$$\frac{100 * (\text{vector-norm(compressed signal, 2)})^2}{(\text{vector-norm(original signal, 2)})^2}$$

and as a tuning parameter the *Norm cfs recovery* defined by

$$\frac{100 * (\text{vector-norm(coeffs of the current decomposition, 2)})^2}{(\text{vector-norm(coeffs of the original decomposition, 2)})^2}$$

The *Number of zeros* in percentage is defined by

$$\frac{100 * (\text{number of zeros of the current decomposition})}{(\text{number of coefficients})}$$

Wavelet Compression for Images

In "Wavelet Data Compression" on page 6-62, we addressed the aspects specifically related to compression using wavelets. However, in addition to the algorithms related to wavelets like DWT and IDWT, it is necessary to use other ingredients concerning the quantization mode and the coding type in order to deal with true compression.

This more complex process can be represented by the following figure.
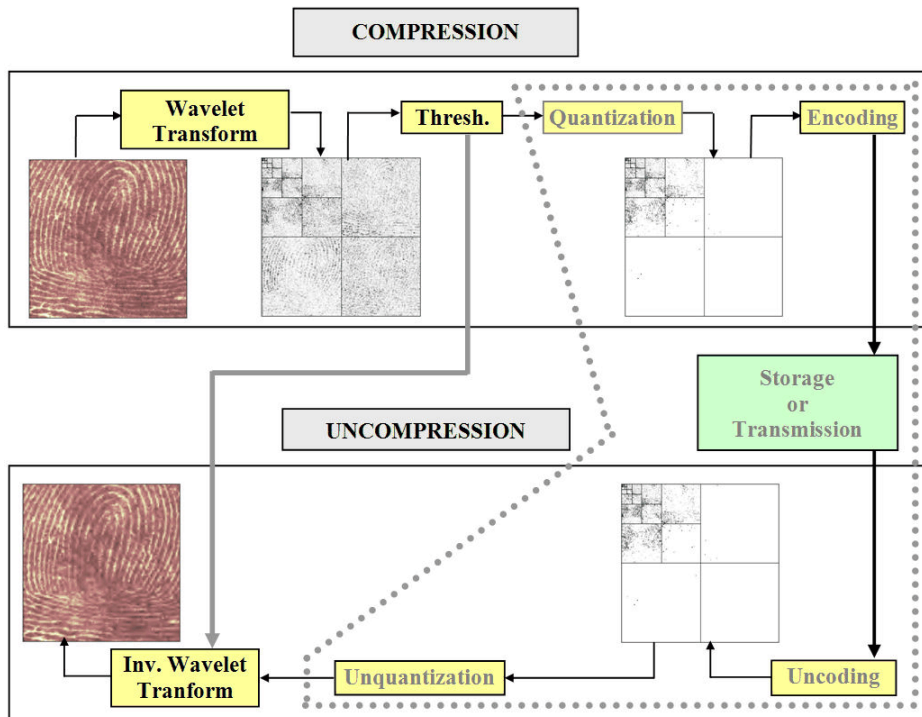
Effects of Quantization

Let us show the effects of quantization on the visualization of the fingerprint image. This indexed image corresponds to a matrix of integers ranging between 0 and 255. Through quantization we can decrease the number of colors which is here equal to 256.

The next figure illustrates how to decrease from 256 to 16 colors by working on the *values* of the original image.
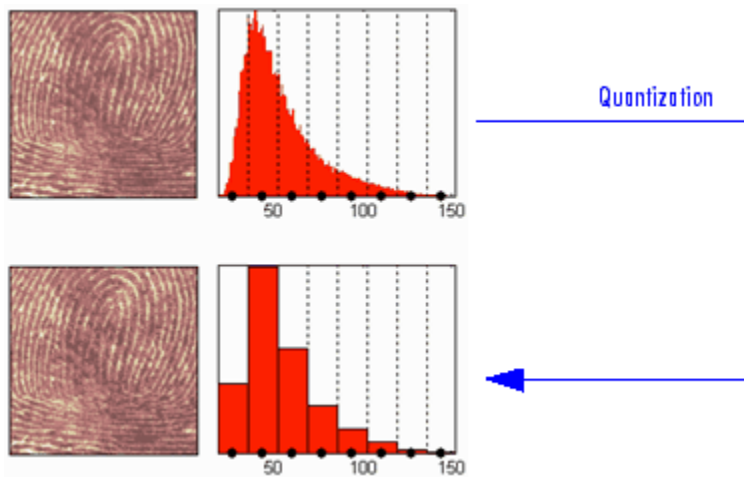
We can see on this figure:

- At the top

  - On the left: the original image
  - On the right: the corresponding histogram of *values*

- At the bottom

  - On the left: the reconstructed image
  - On the right: the corresponding histogram of *quantized values*

This quantization leads to a compression of the image. Indeed, with a fixed length binary code, 8 bits per pixel are needed to code 256 colors and 4 bits per pixel to code 16 colors. We notice that the image obtained after quantization is of good quality. However, within the framework of true compression, quantization is not used on the original image, but on its wavelet decomposition.

Let us decompose the fingerprint image at level 4 with the Haar wavelet. The histogram of wavelet coefficients and the quantized histogram are normalized so that the values vary between –1 and +1. The 15 intervals of quantization do not have the same length.

The next figure illustrates how to decrease information by binning on the wavelet *coefficient values* of the original image.
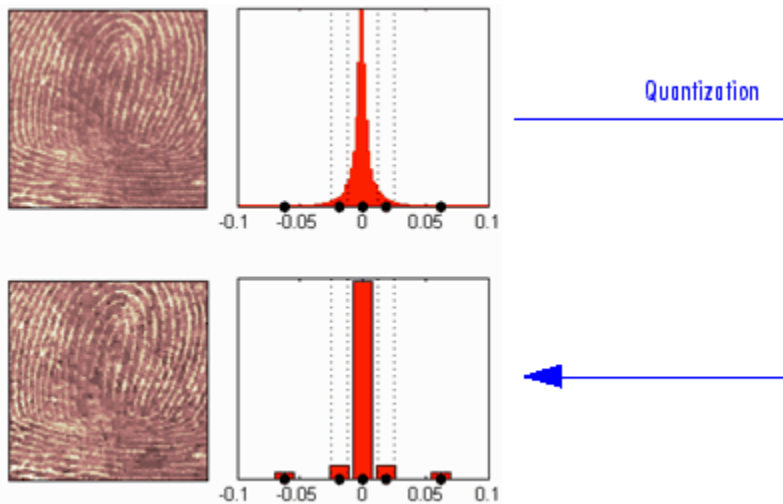
We can see on this figure:

- At the top

  - On left: the original image
  - On the right: the corresponding histogram (central part) of *coefficient values*

- At the bottom

  - On the left: the reconstructed image
  - On the right: the corresponding histogram (central part) of *quantized coefficient values*

The key point is that the histogram of the quantized coefficients is massively concentrated in the class centered in 0. Let us note that yet again the image obtained is of good quality.

True Compression Methods

The basic ideas presented above are used by three methods which cascade in a single step, coefficient thresholding (global or by level), and encoding by quantization. Fixed or Huffman coding can be used for the quantization depending on the method.

The following table summarizes these methods, often called Coefficients Thresholding Methods (CTM), and gives the MATLAB name used by the true compression tools for each of them.

MATLAB Name	Compression Method Name
`'gbl_mmc_f'`	Global thresholding of coefficients and fixed encoding
`'gbl_mmc_h'`	Global thresholding of coefficients and Huffman encoding
`'lvl_mmc'`	Subband thresholding of coefficients and Huffman encoding

More sophisticated methods are available which combine wavelet decomposition and quantization. This is the basic principle of progressive methods.

On one hand, progressivity makes it possible during decoding to obtain an image whose resolution increases gradually. In addition, it is possible to obtain a set of compression ratios based on the

length of the preserved code. This compression usually involves a loss of information, but this kind of algorithm enables also lossless compression.

Such methods are based on three ideas. The two first, already mentioned, are the use of wavelet decomposition to ensure sparsity (a large number of zero coefficients) and classical encoding methods. The third idea, decisive for the use of wavelets in image compression, is to exploit fundamentally the tree structure of the wavelet decomposition. Certain codes developed from 1993 to 2000 use this idea, in particular, the EZW coding algorithm introduced by Shapiro. See [Sha93] in "References".

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 6-65 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 6-65 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')
```

Original Image: mask
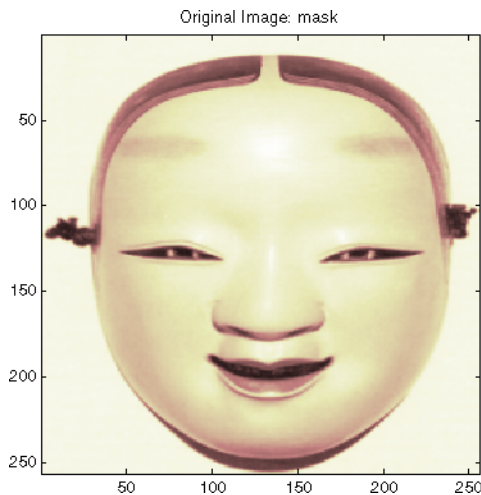
A synthetic performance of the compression is given by the compression ratio and the Bit-Per-Pixel ratio which are equivalent.

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')]})
```
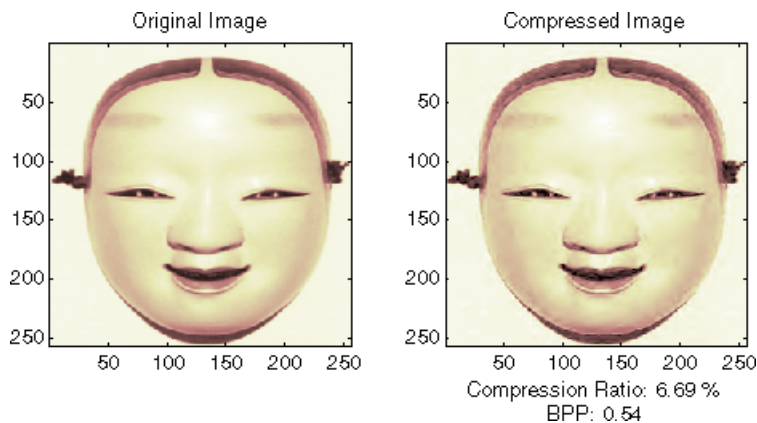

Compression by Progressive Methods

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')]})
```
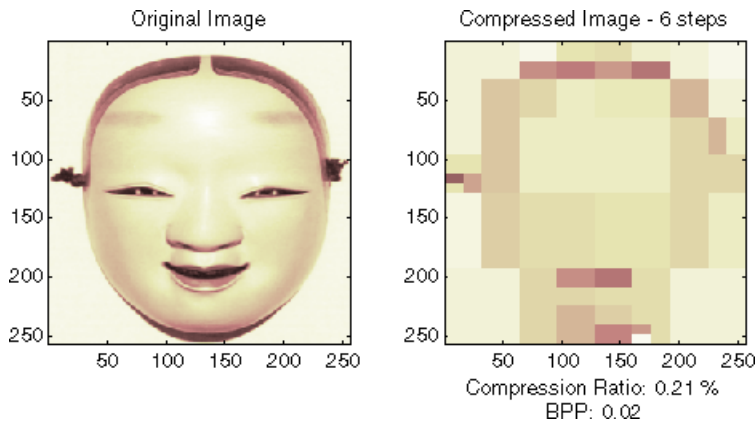
A too small number of steps (here 6) produces a very coarse compressed image. So let us examine a little better result for 9 steps and a satisfactory result for 12 steps.

```
[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')]})
```
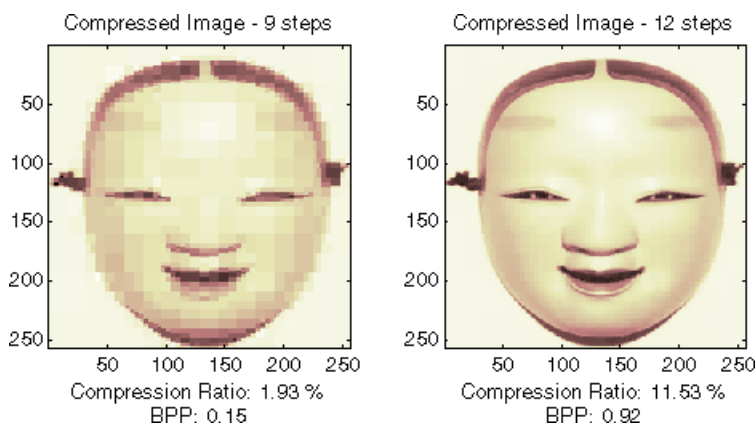

As can be seen, the reached BPP ratio is about 0.92 when using 12 steps.

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')]})
```


Compressed Image - 12 steps - bior4.4   Compressed Image - 11 steps - bior4.4

Compression Ratio: 7.80 %   Compression Ratio: 4.40 %
BPP: 0.62   BPP: 0.35
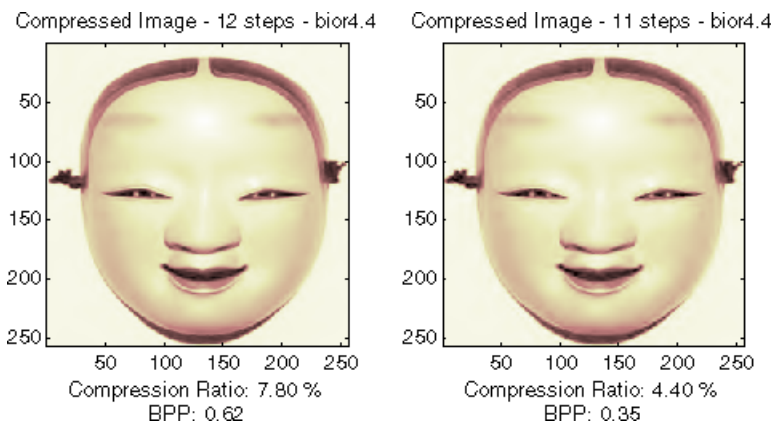
Starting from the eleventh loop, the result can be considered satisfactory. The reached BPP ratio is now about 0.35. It can even be slightly improved by using a more recent method: SPIHT (Set Partitioning In Hierarchical Trees).

```
[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')
```

Original Image — Compressed Image - 12 steps - bior4.4
Compression Ratio: 2.83 %
BPP: 0.23
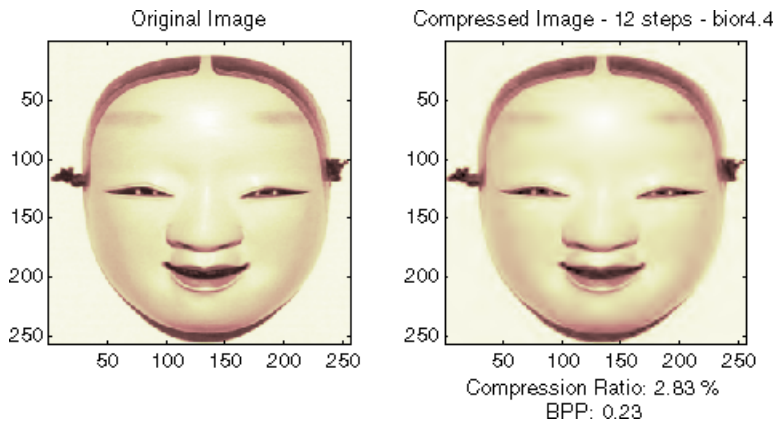
The final compression ratio (2.8%) and the Bit-Per-Pixel ratio (0.23) are very satisfactory. Let us recall that the first ratio means that the compressed image is stored using only 2.8% of the initial storage size.

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')
```
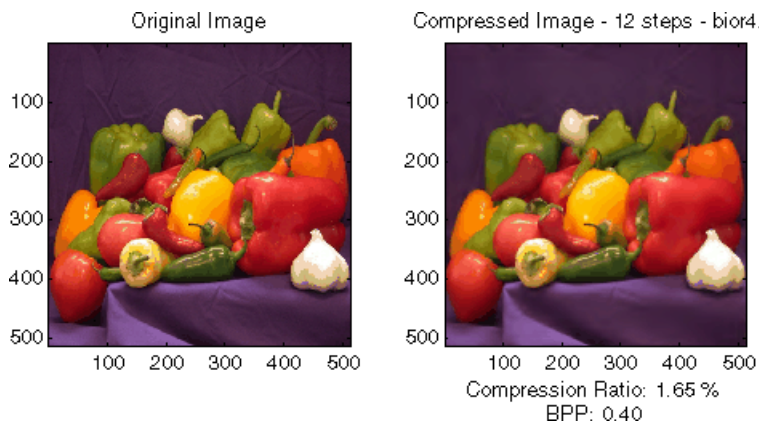

Original Image — Compressed Image - 12 steps - bior4.4
Compression Ratio: 1.65 %
BPP: 0.40

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.
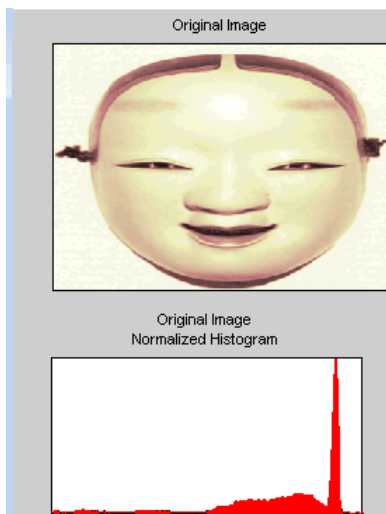
3 Perform a Wavelet Decomposition.

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.
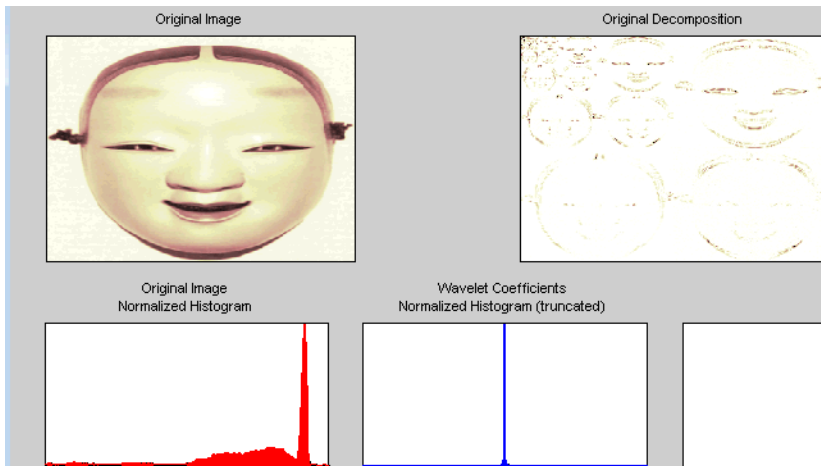
The peak of the bin containing zero illustrates the capability of wavelets to concentrate the image energy into a few nonzero coefficients.

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 6-65 in the *Wavelet Toolbox User's Guide*.

Set the desired Bit-Per-Pixel ratio to **0.5**.

Values of the other parameters are automatically updated. Note that these values are only approximations of the true performances since the quantization step cannot be exactly predicted without performing it. Click the **Compress** button.

M.S.E.	24.77
Max Error	53.38
L2-Norm Ratio	100.13 %
P.S.N.R.	34.19
B.P.P.	0.5376
Comp. Ratio	6.72 %
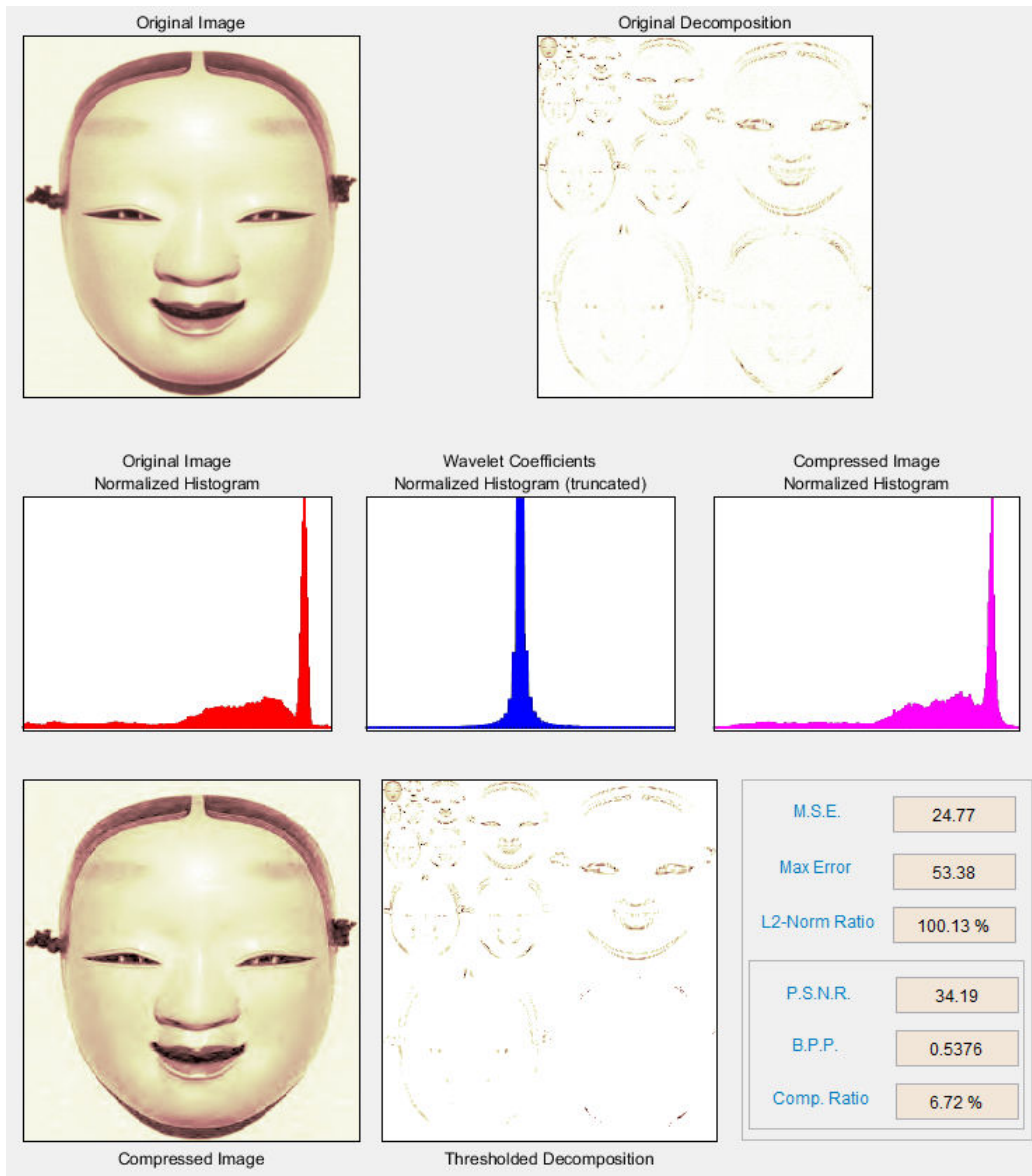
Synthetic performance is given by the compression ratio and the computed Bit-Per-Pixel (BPP). This last one is actually about 0.53 (close to the desired one 0.5) for a compression ratio of 6.7%.

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.
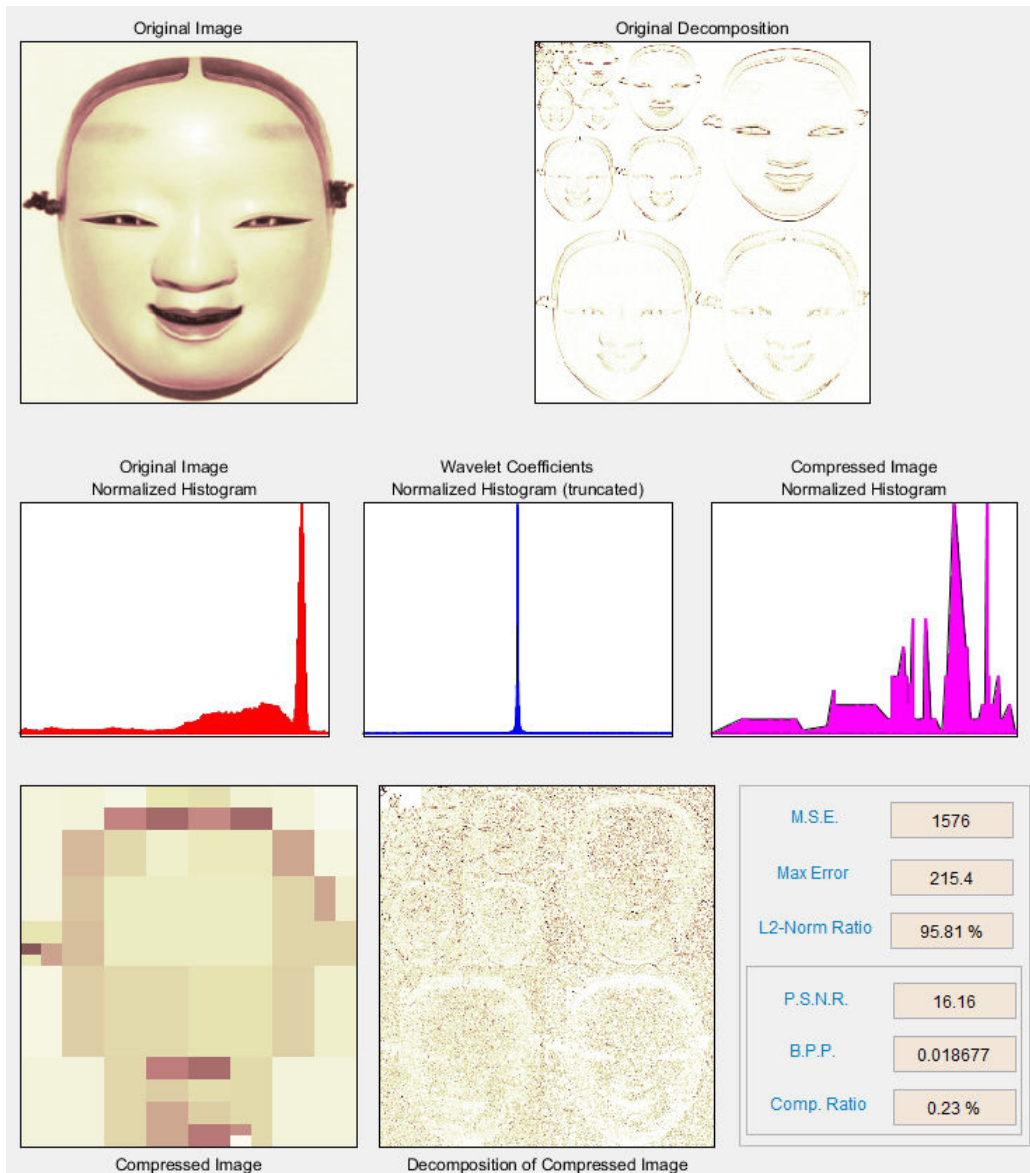
6 Refine the result by increasing the number of loops.

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.
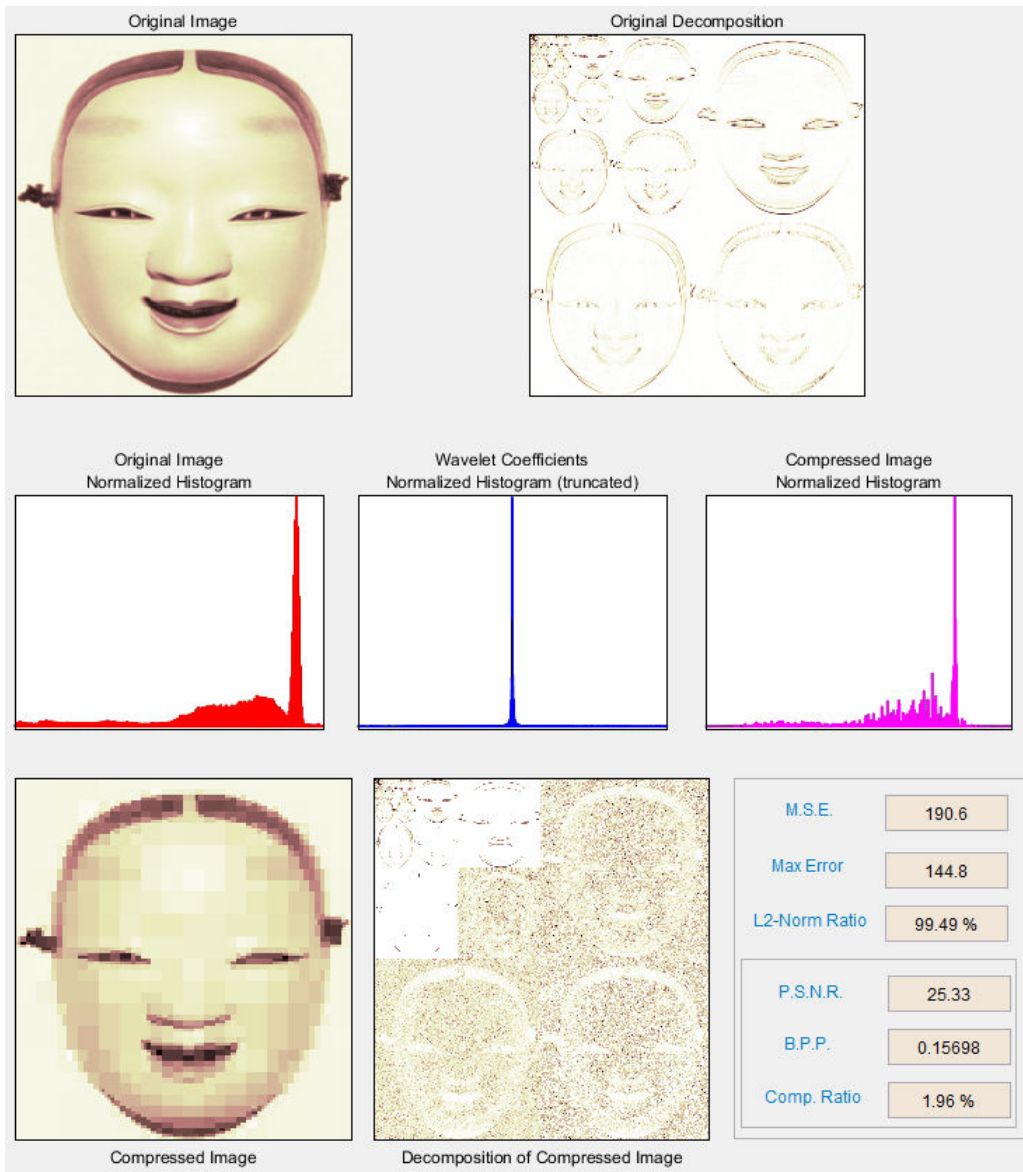
As can be seen, the result is better but not satisfactory, both by visual inspection and by calculating the Peak Signal to Noise Ratio (PSNR) which is less than 30.

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.
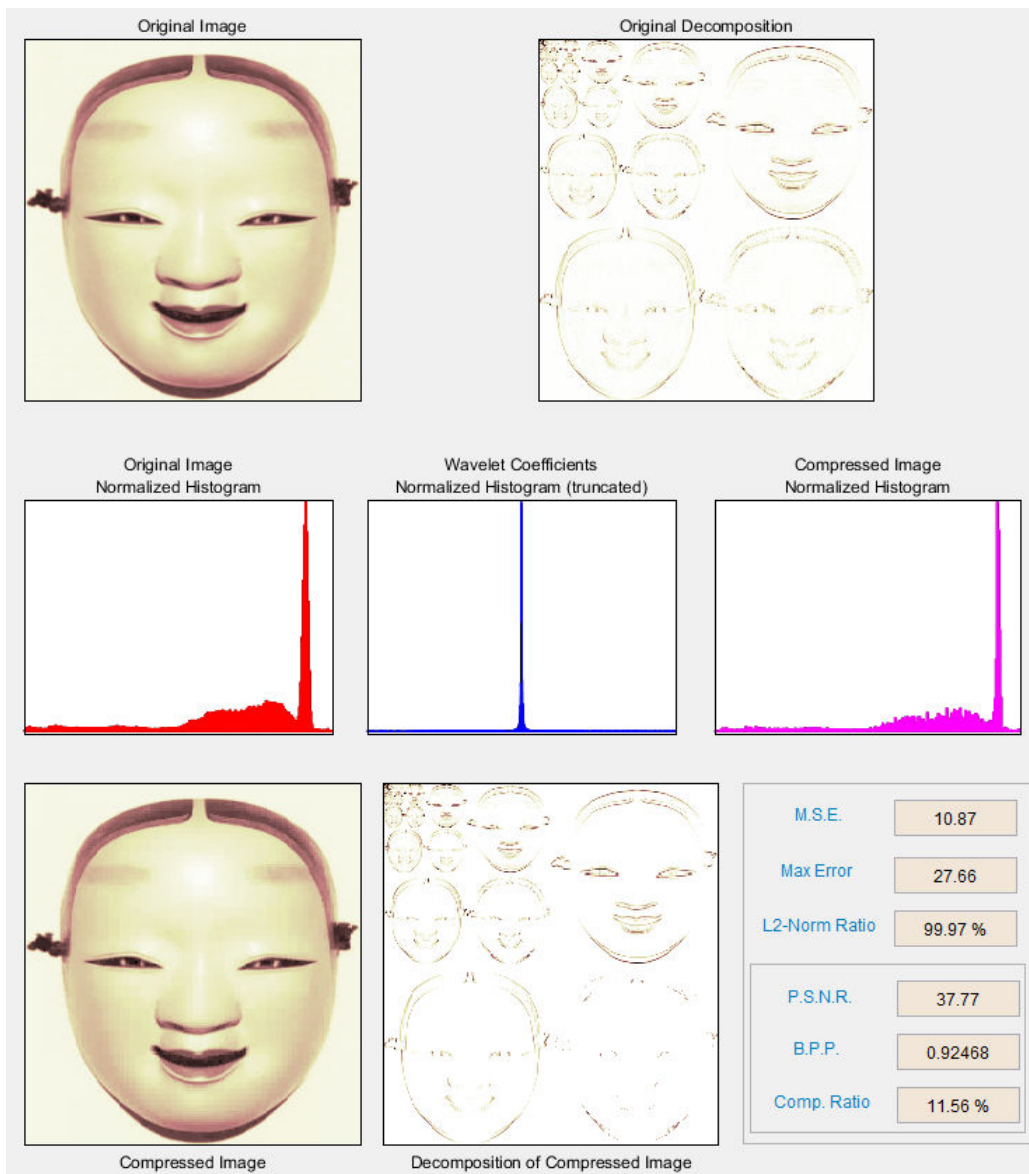
The result is now acceptable. But for 12 steps, we attain a Bit-Per-Pixel ratio about 0.92.

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.
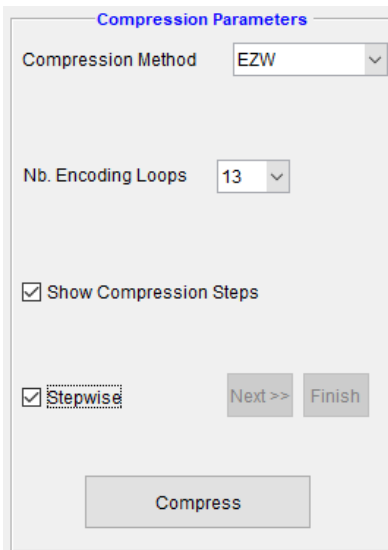
Then, click the **Compress** button. Thirteen progressively more finely compressed images appear, and you can then select visually a convenient value for the number of loops. A satisfactory result seems to be obtained after 11 loops. So, you can set the number of encoding loops to **11** and click the **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.
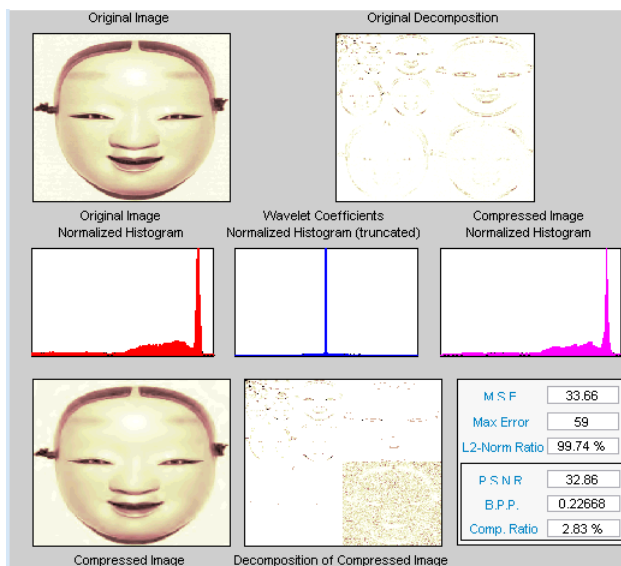
The final compression ratio and the Bit-Per-Pixel ratio are very satisfactory: 2.8% and 0.22. Let us recall that the first ratio means that the compressed image is stored using only 2.8% of the initial storage size.

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.
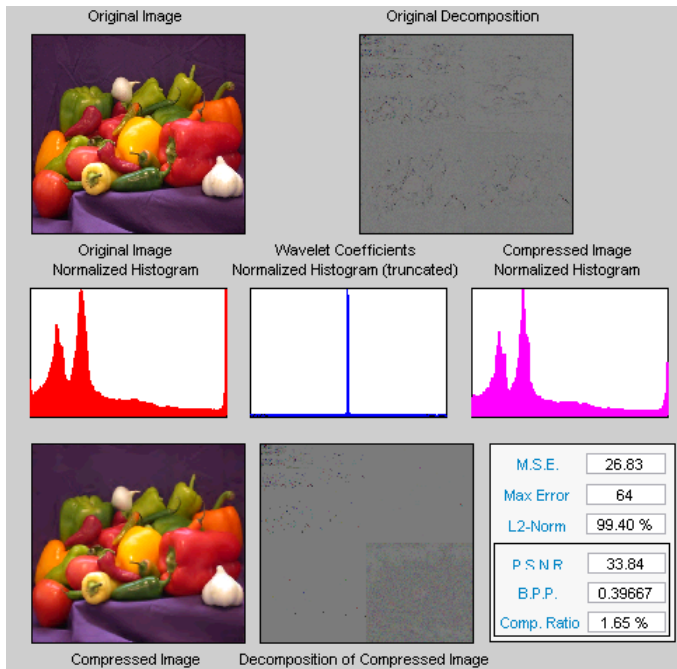
10 Inspect the wavelet tree.

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.
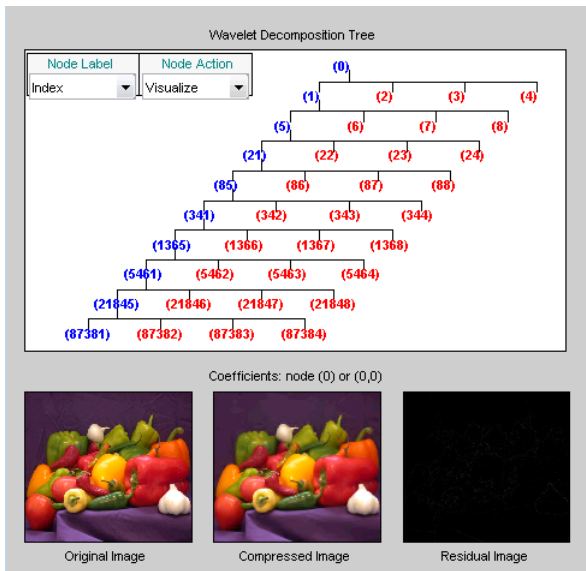
Importing and Exporting from the Wavelet Analyzer App

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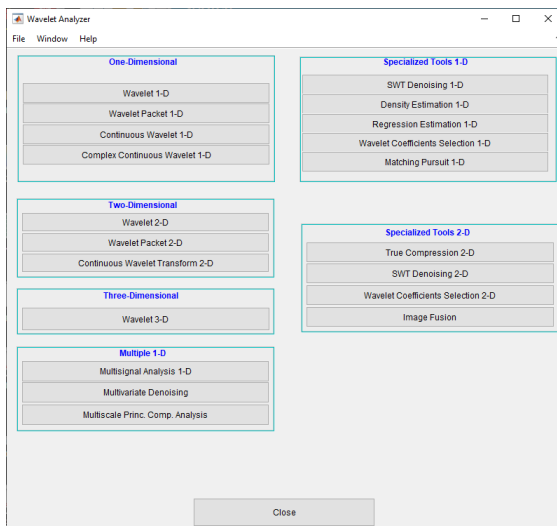
Click the **Regression Estimation 1-D** menu item. The discrete wavelet analysis tool for 1-D regression estimation appears.

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.
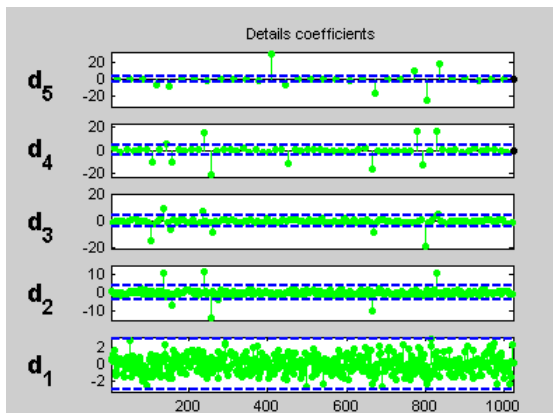
5   Perform a regression estimation.

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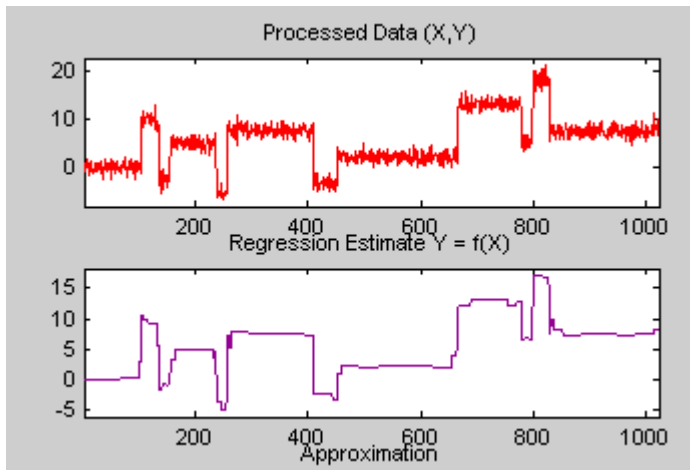
You can see that the process removed the noise and that the blocks are well reconstructed. The regression estimate (in yellow) is the sum of the signals located below it: the approximation a5 and the reconstructed details after coefficient thresholding.

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

Let us now illustrate the regression estimation using the 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 `exlnsto.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.
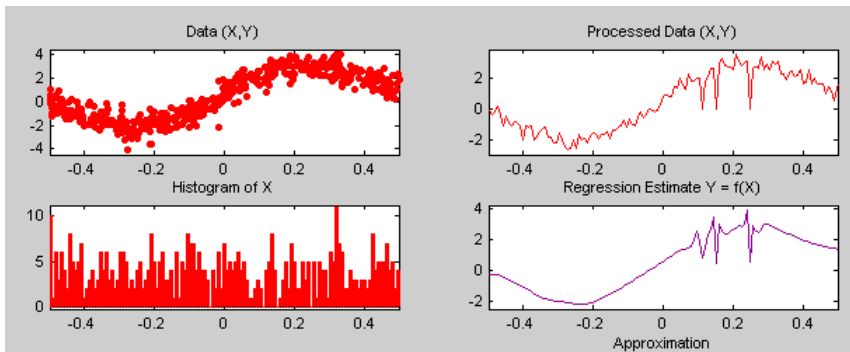
4 Check **Overlay Estimated Function** to validate the fit of the original data.
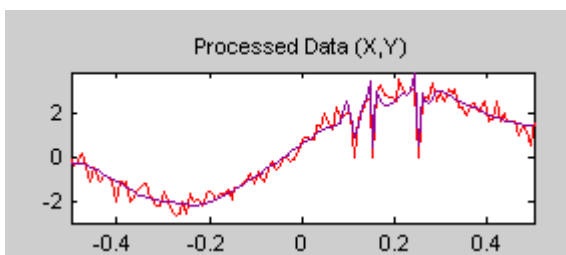
Importing and Exporting Information from the Wavelet Analyzer App

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 6-39 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
```

Name	Size	Bytes	Class
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')
```

7

Matching Pursuit

Matching Pursuit Algorithms

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 $\{\varphi_k\}$ 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 φ_k yielding the largest inner products (in absolute value) with the signal. For example, the best single φ_k is

$$\max_k \left| <x, \phi_k> \right|$$

which for a unit-norm atom is the magnitude of the scalar projection onto the subspace spanned by φ_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 Φ 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 L2 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 $\Phi = \{\varphi_k\}$ 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 φ_p.

3   Form the residual $R^1 f$ by subtracting the orthogonal projection of $R^0 f$ onto the space spanned by φ_p.

$$R^1 f = R^0 f - \left\langle R^0 f, \phi_p \right\rangle \phi_p$$

4   Iterate by repeating steps 2 and 3 on the residual.

$$R^{m+1} f = R^m f - \left\langle R^m f, \phi_k \right\rangle \phi_k$$

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.
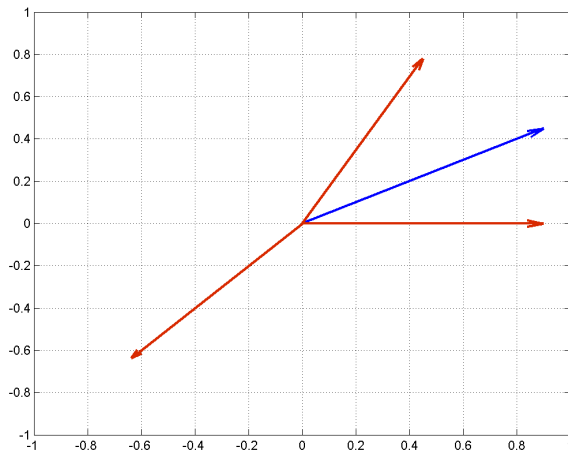
Construct this dictionary and signal in MATLAB.

```
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 π 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 7-5 (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 φ_p.

3   Form a matrix, Φ, with previously selected atoms as the columns. Define the orthogonal projection operator onto the span of the columns of Φ.

$$P = \Phi \, (\Phi^* \Phi)^{-1} \Phi^*$$

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.

$$\left| \langle x, \phi_p \rangle \right| \geq \alpha \max_k \left| \langle x, \phi_k \rangle \right|, \quad \alpha \in (0, 1]$$

This is known as *weak* matching pursuit.

Matching Pursuit

In this section...

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,'lstcpt',{{'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
```
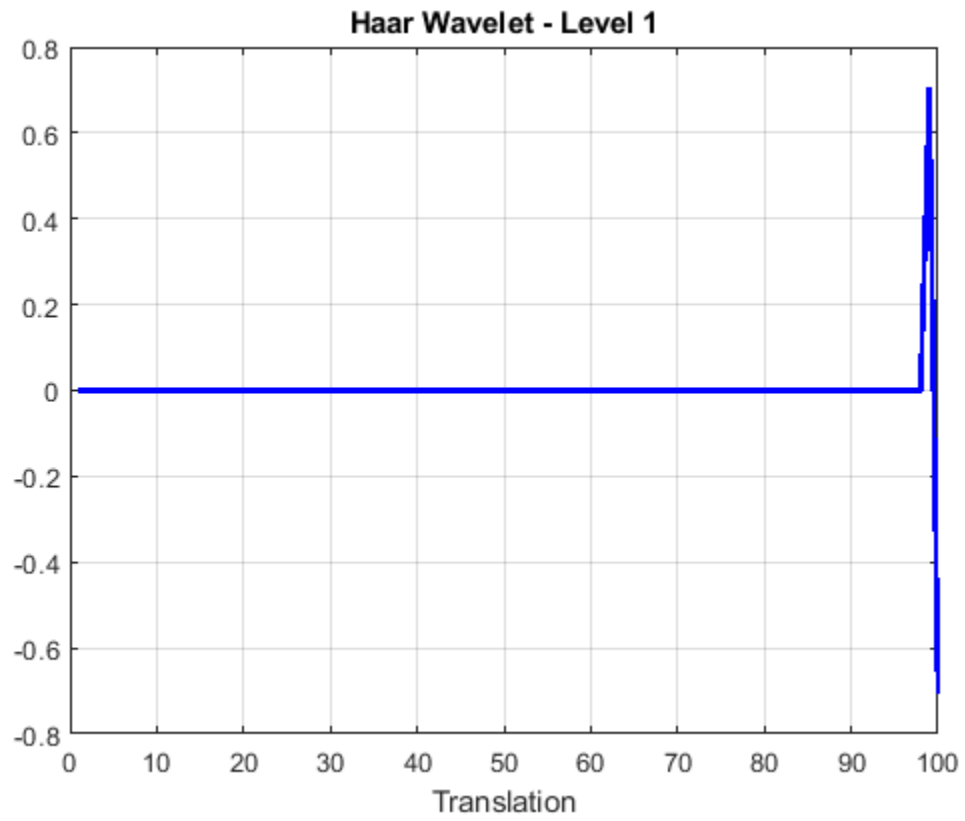
This animation infinitely loops through all the plots generated.

Haar Scaling Function - Level 2
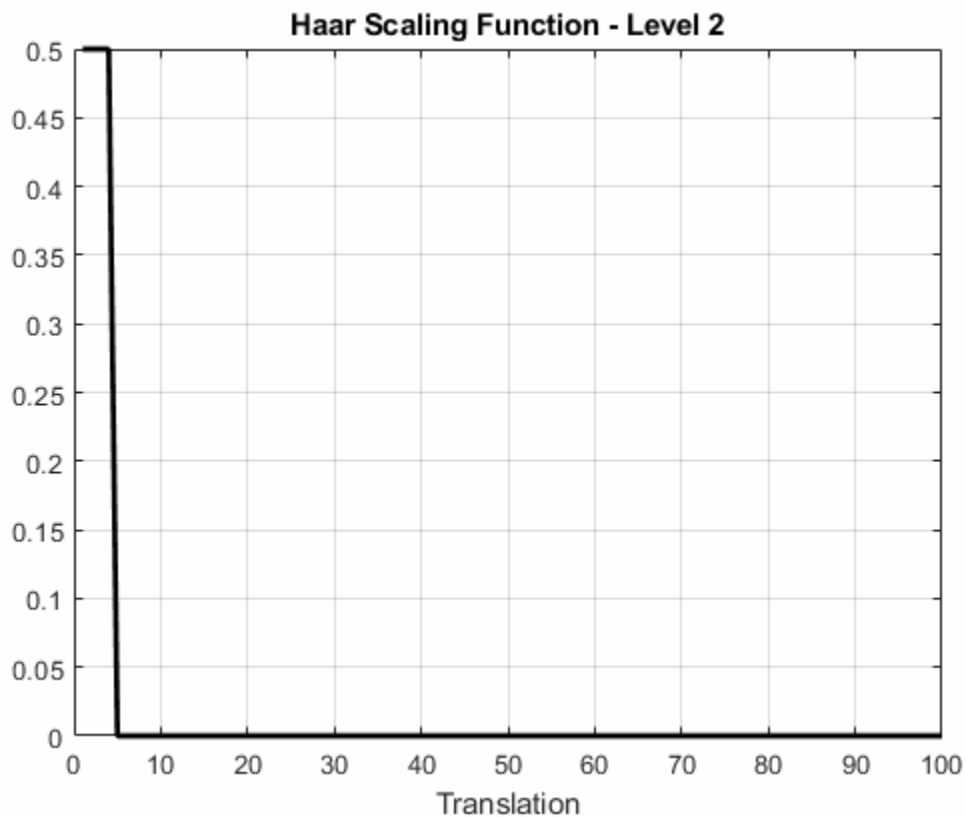
Orthogonal Matching Pursuit on a 1-D Signal

This example shows how to perform orthogonal matching pursuit on a 1-D input signal that contains a cusp.

Load the cuspamax 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 cuspamax;
dict = {{'wpsym4',1},{'db4',2},'dct','sin','RnIdent'};
mpdict = wmpdictionary(length(cuspamax),'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);
```
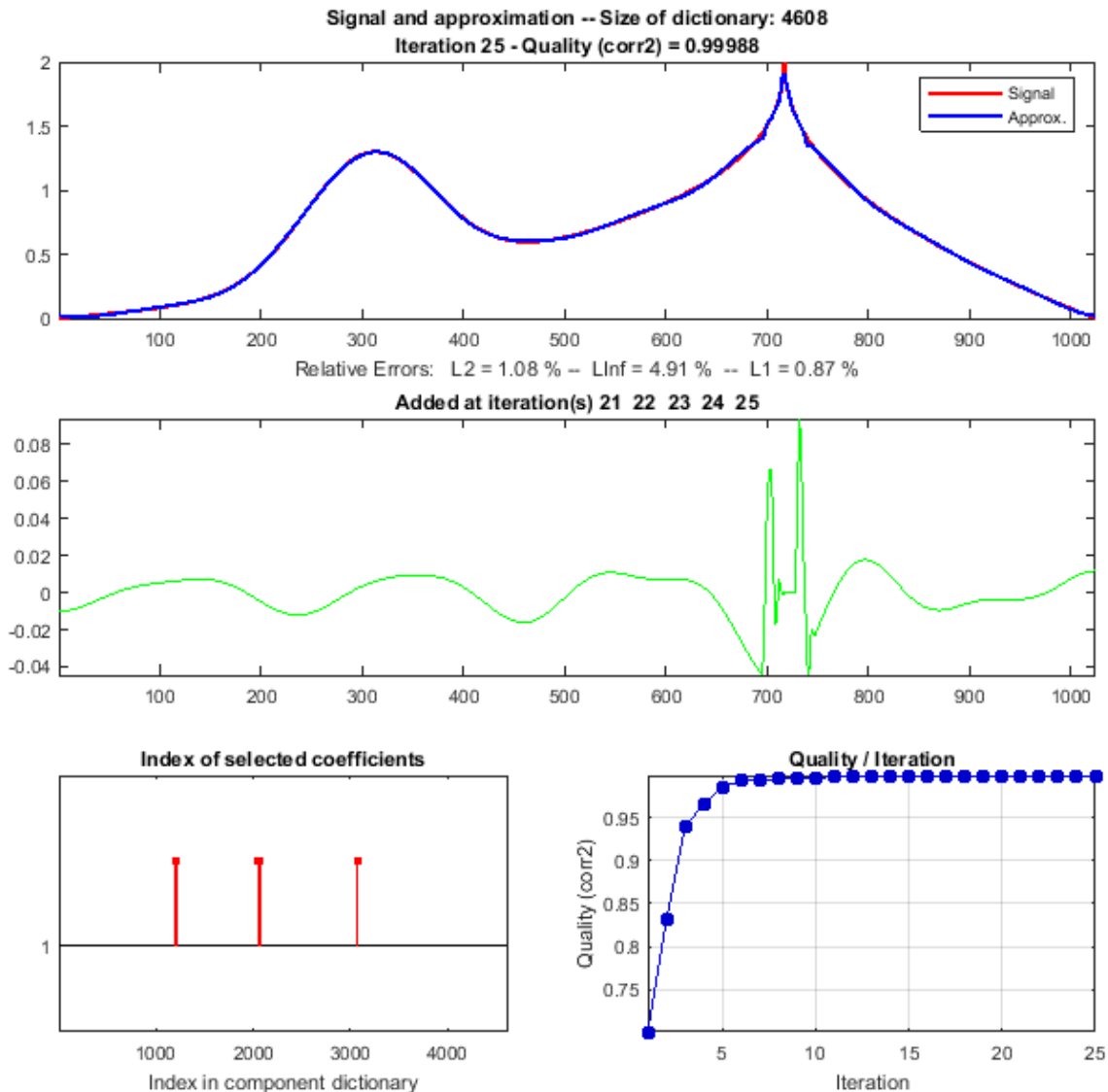
Electricity Consumption Analysis Using Matching Pursuit

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')
```
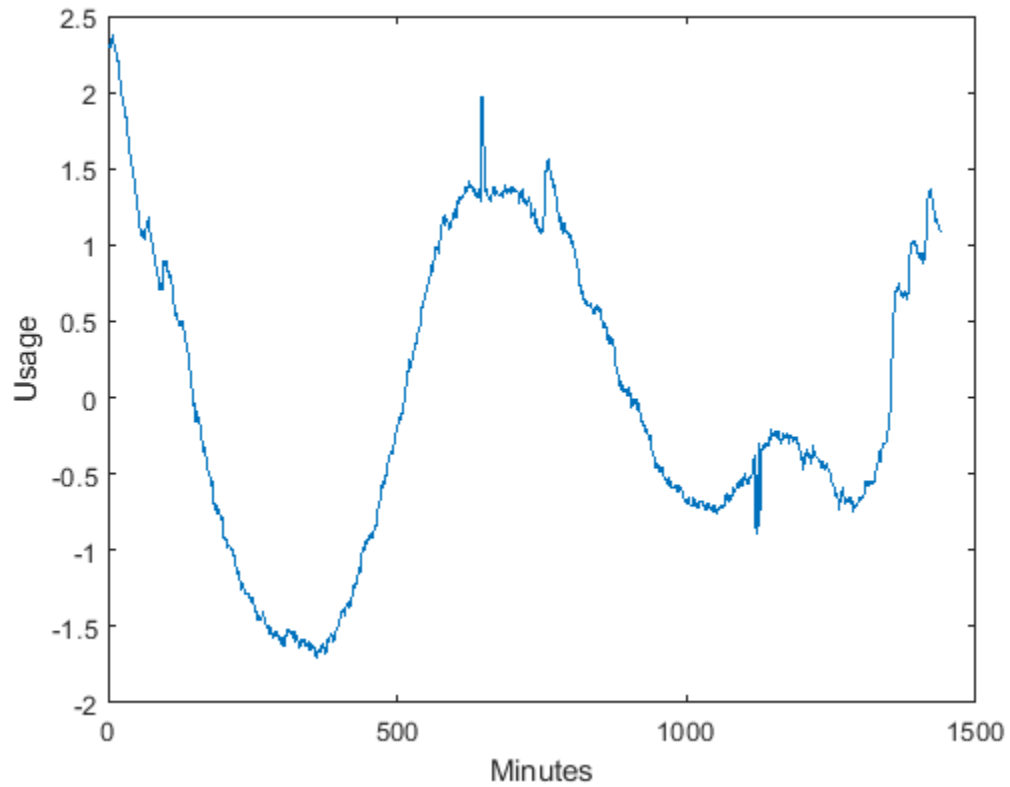

The electricity consumption data contains smooth oscillations punctuated by abrupt increases and decreases in usage.

Zoom in on a time interval from 500 minutes to 1200 minutes.

```
xlim([500 1200])
```
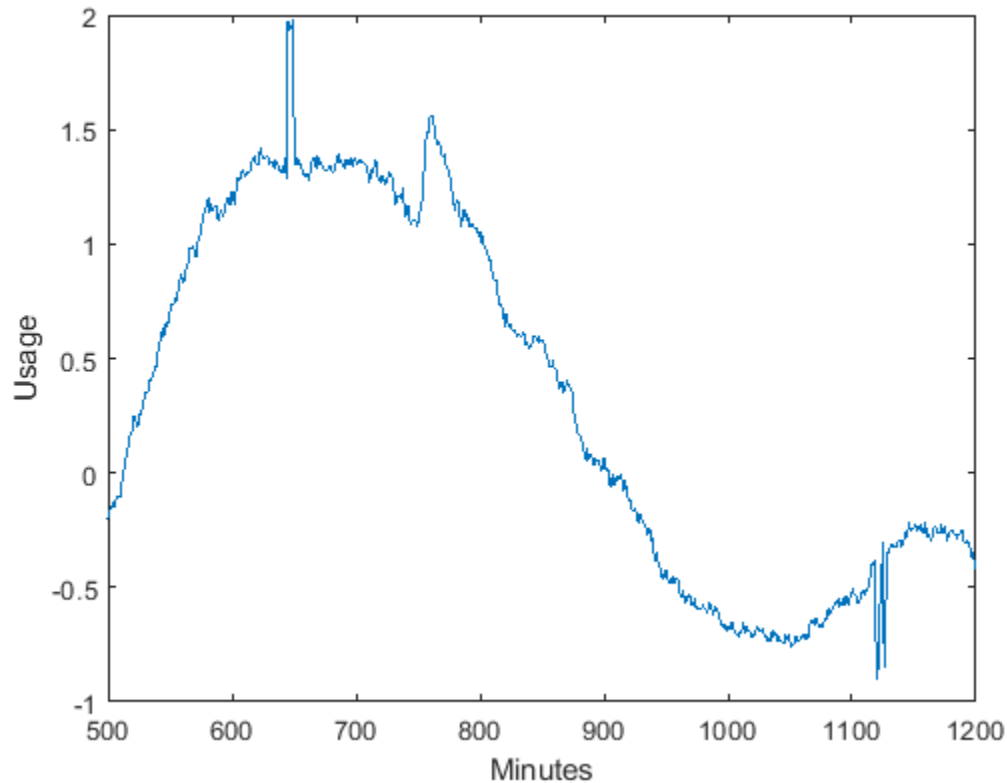
You can see the abrupt changes in the slowly-varying signal at approximately 650, 760, and 1120 minutes. In many real-world signals like these data, the interesting and important information is contained in the transients. It is important to model these transient phenomena.

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')
```
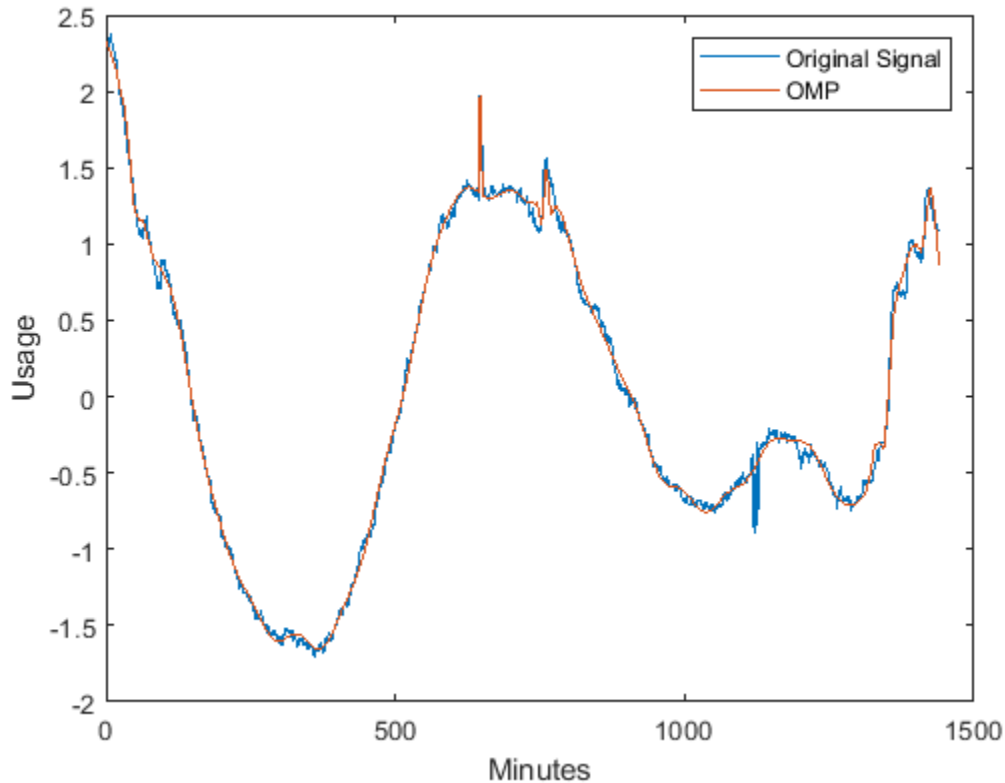
You can see that with 35 coefficients, orthogonal matching pursuit approximates both the smoothly-oscillating part of the signal and the abrupt changes in electricity usage.

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
```
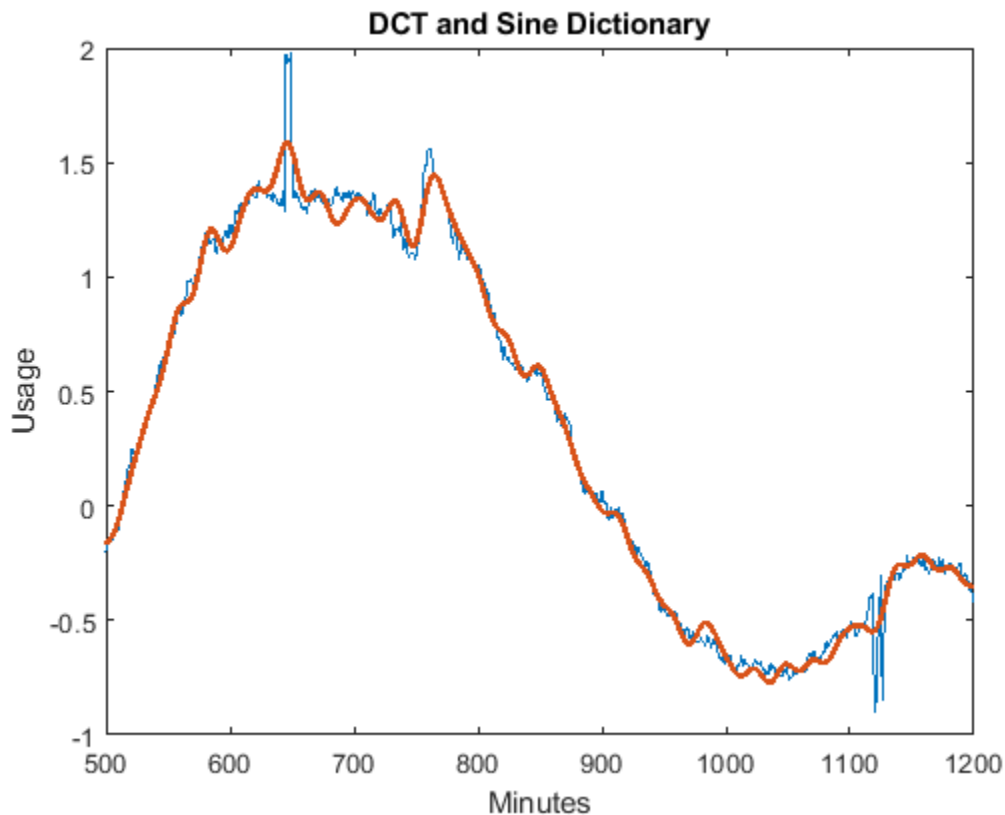
```matlab
plot(x)
hold on
plot(y,'linewidth',2)
hold off
title('Full Dictionary')
xlabel('Minutes')
ylabel('Usage')
xlim([500 1200])
```
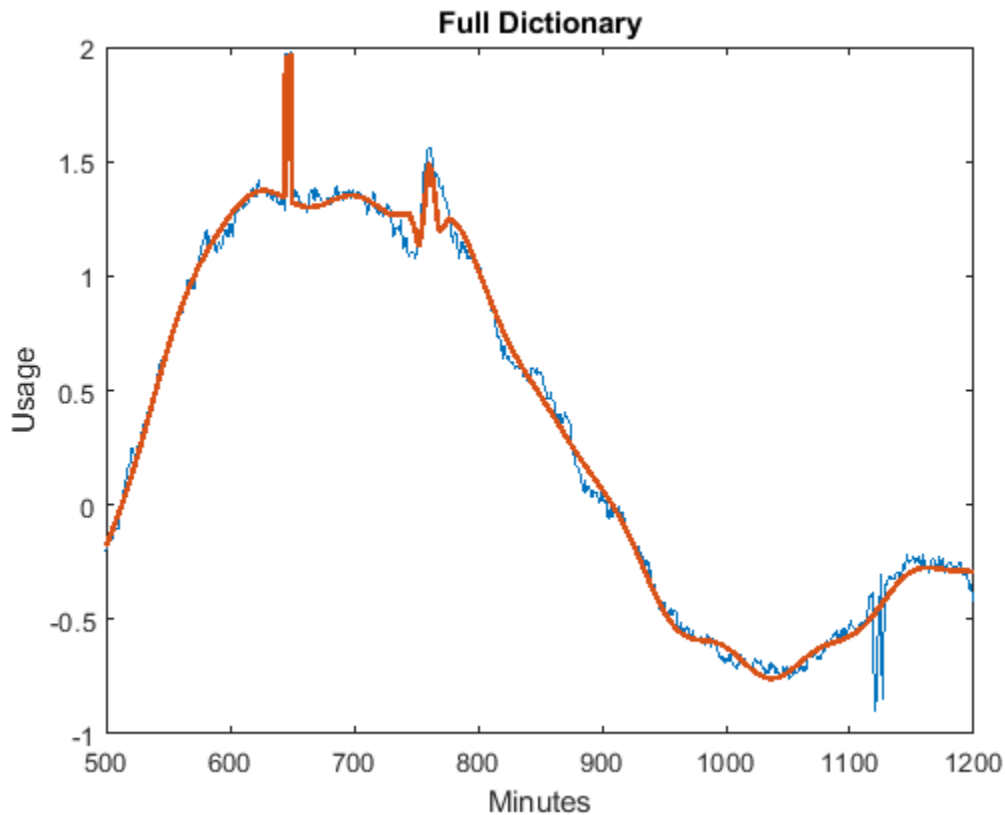

Obtain the best 35-term nonlinear approximation of the signal in the discrete Fourier basis. Obtain the DFT of the data, sort the DFT coefficients, and select the 35 largest coefficients. The DFT of a real-valued signal is conjugate symmetric, so only consider frequencies from 0 (DC) to the Nyquist (1/2 cycles/minute).

```matlab
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.

```matlab
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')
```
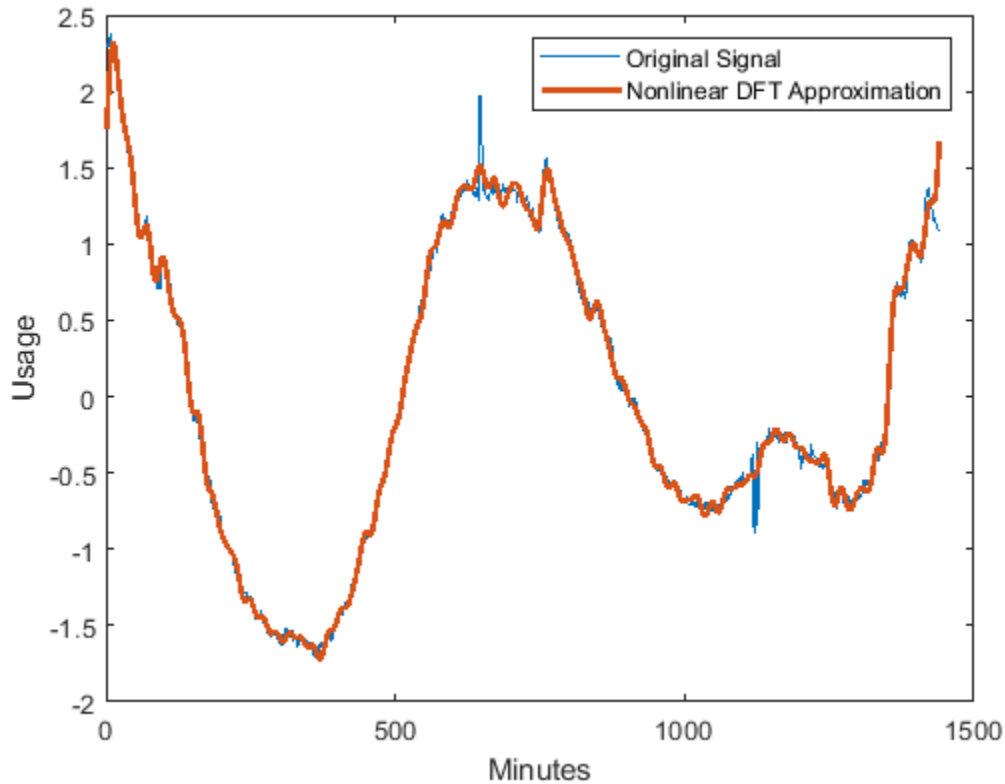

Similar to the DCT-sine dictionary, the nonlinear DFT approximation performs well at matching the smooth oscillations in electricity consumption data. However, the nonlinear DFT approximation does not approximate the abrupt changes accurately. Zoom in on the interval of the data containing the abrupt 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])
```
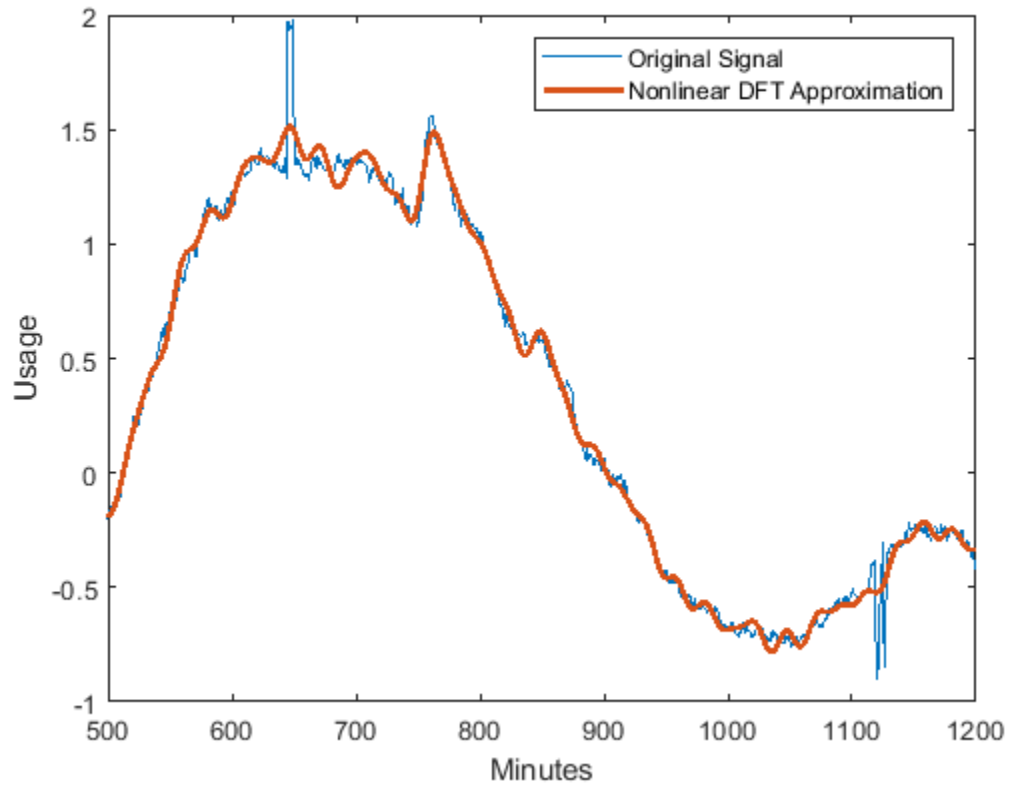
Matching Pursuit Using Wavelet Analyzer App

In this section...
"Matching Pursuit 1-D Interactive Tool" on page 7-17
"Interactive Matching Pursuit of Electricity Consumption Data" on page 7-29

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.
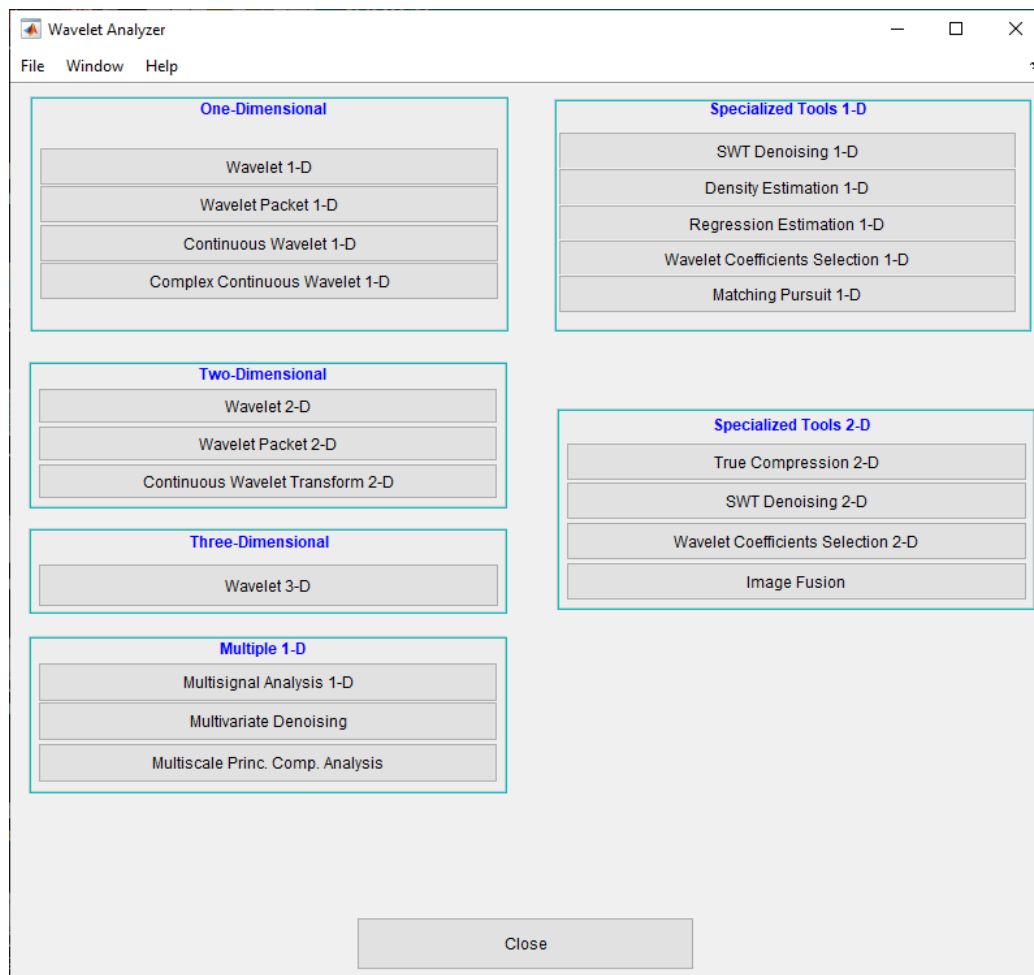
Click **Matching Pursuit 1-D**.
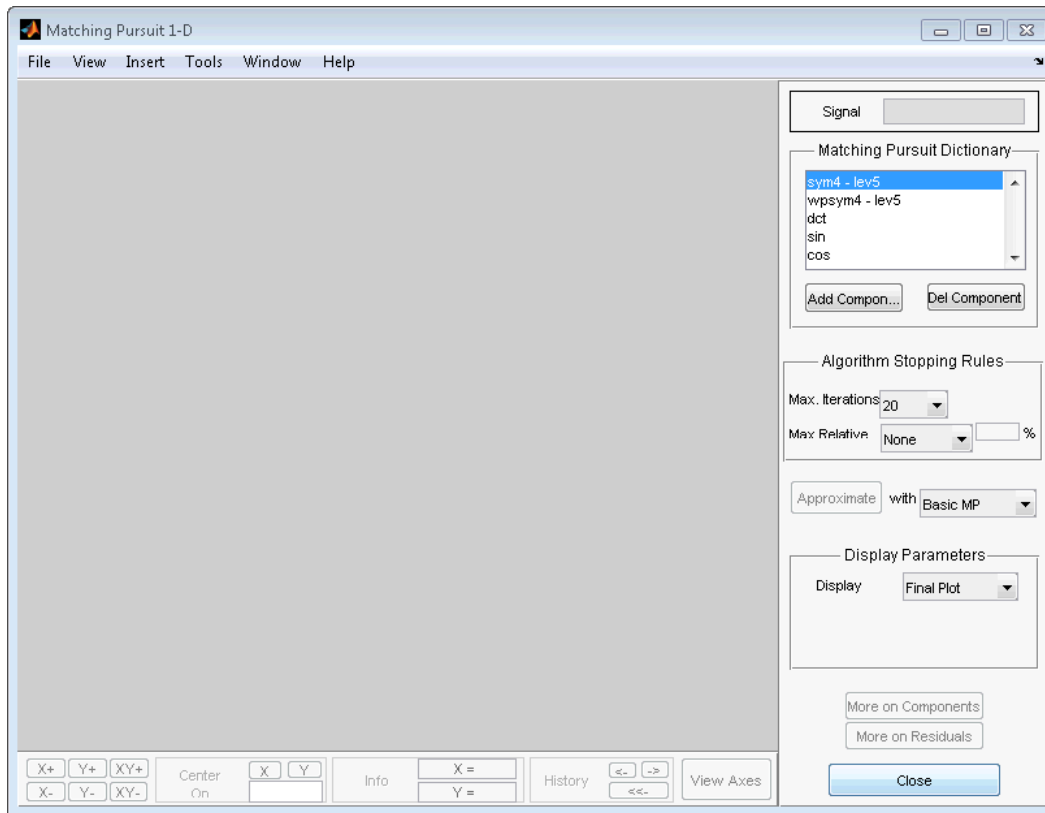
To demonstrate the **Matching Pursuit 1-D** tool, select **File —> Example —> Cusp signal**.
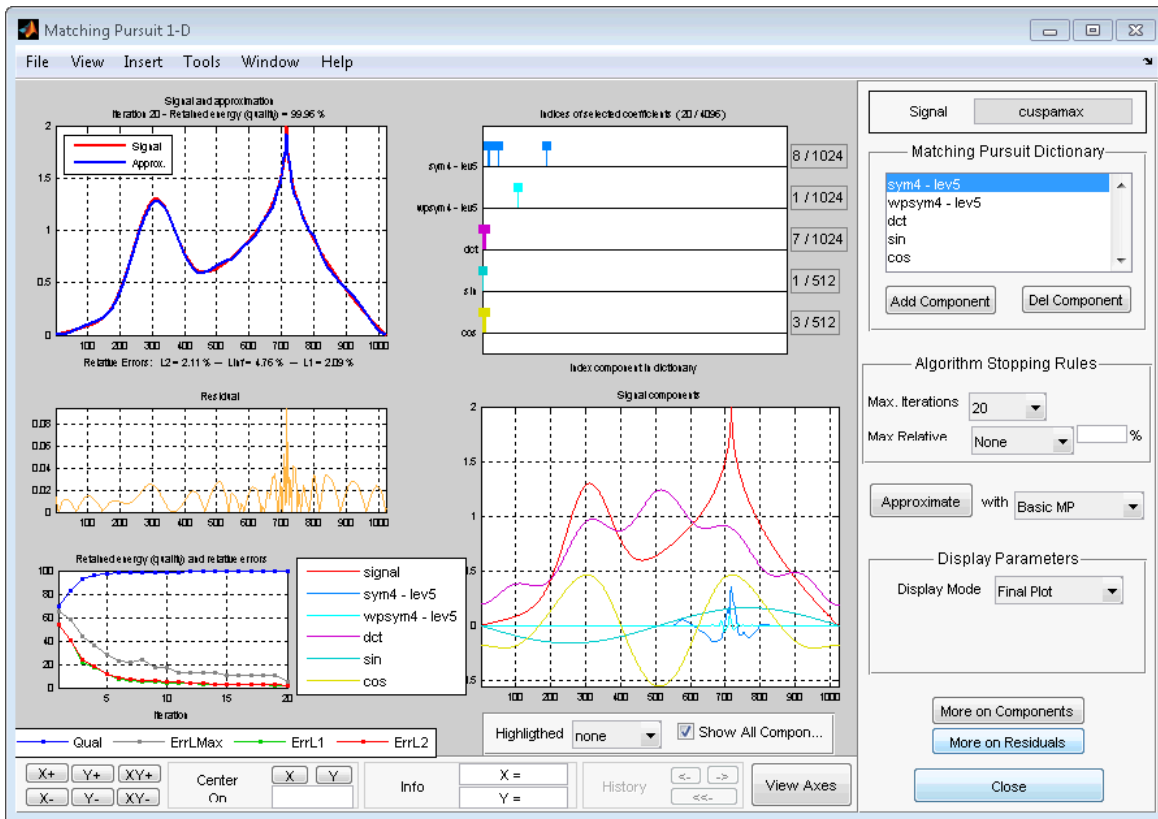
In the upper left corner, you see the plot of the signal with the matching pursuit approximation superimposed.
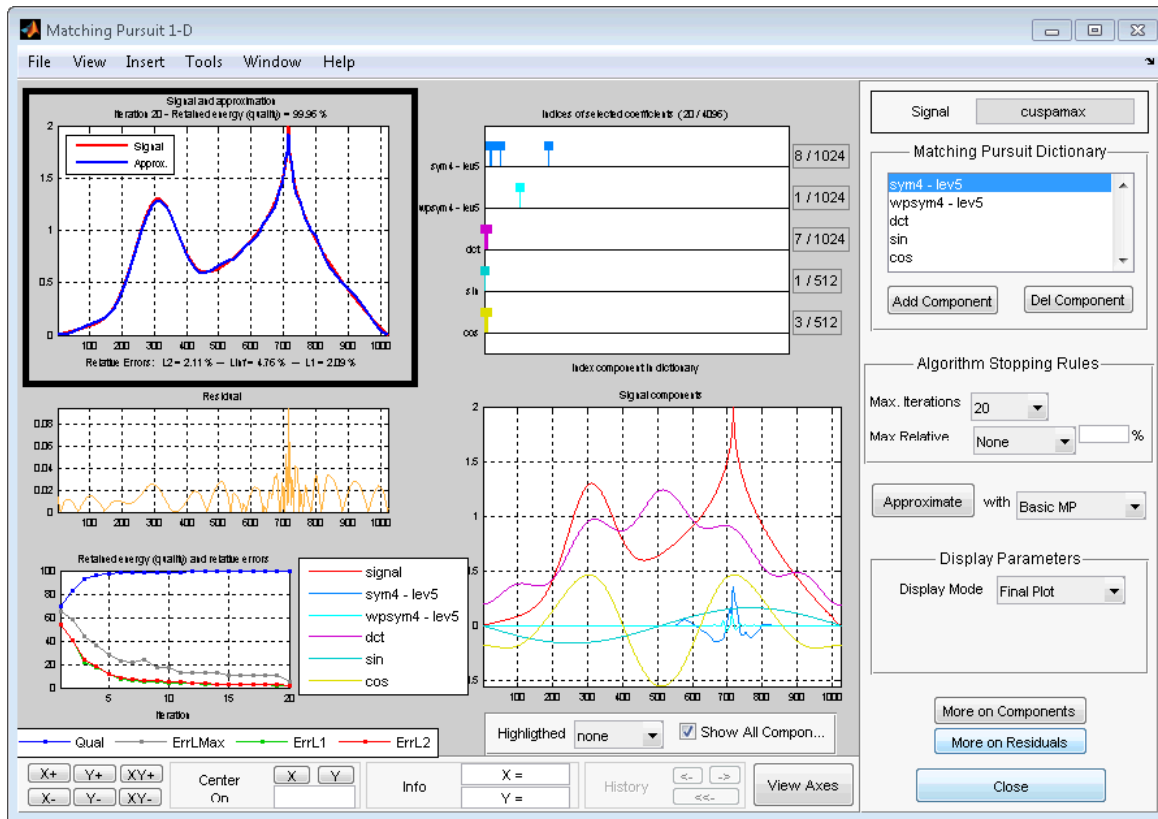
Underneath the plot, you see the relative errors using the L1, L2, and L-infinity norms.

The maximum relative error in a given norm is

$$100\frac{||R||}{||Y||},$$

where $|| \ ||$ 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.
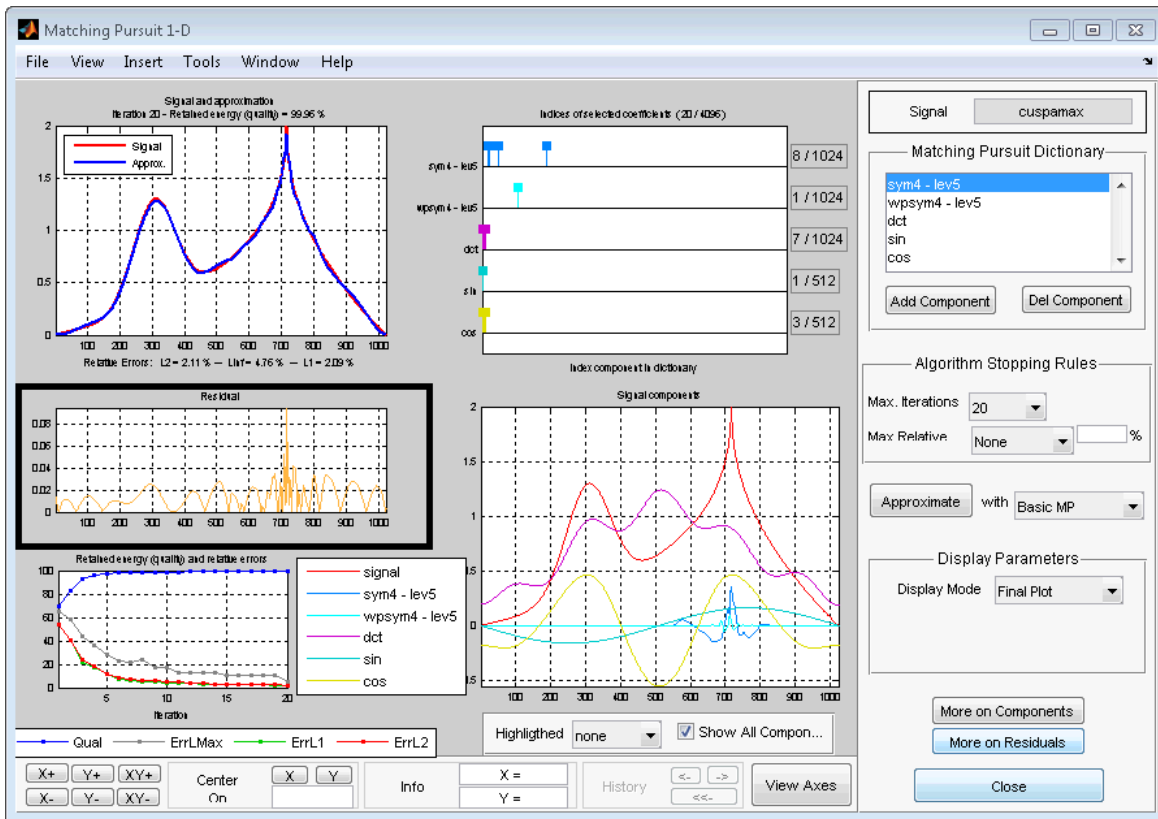
The bottom left panel displays the percentage of retained signal energy (L2 norm) and the relative error percentages for the L1, L2, and L-infinity norms over the algorithm iterations.
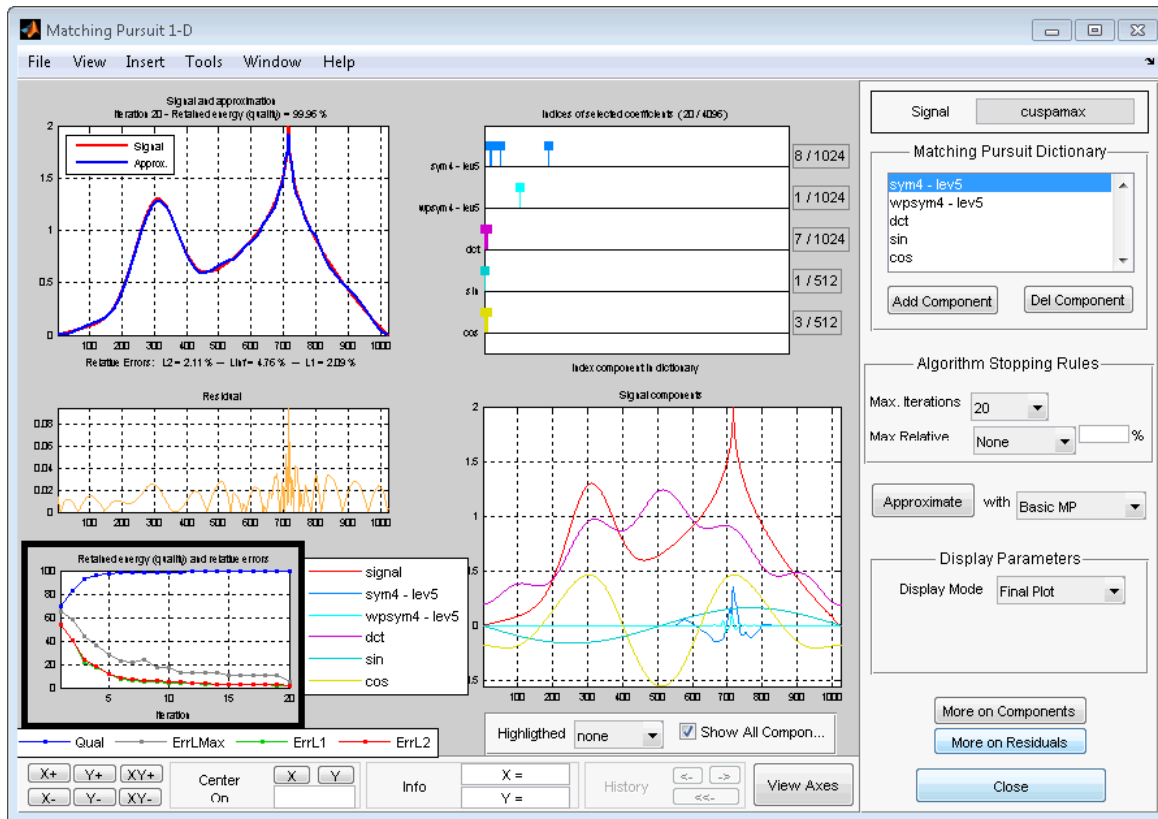
In the top middle panel of the **Matching Pursuit 1-D** tool, you see the indices of the selected coefficients from the subdictionaries.
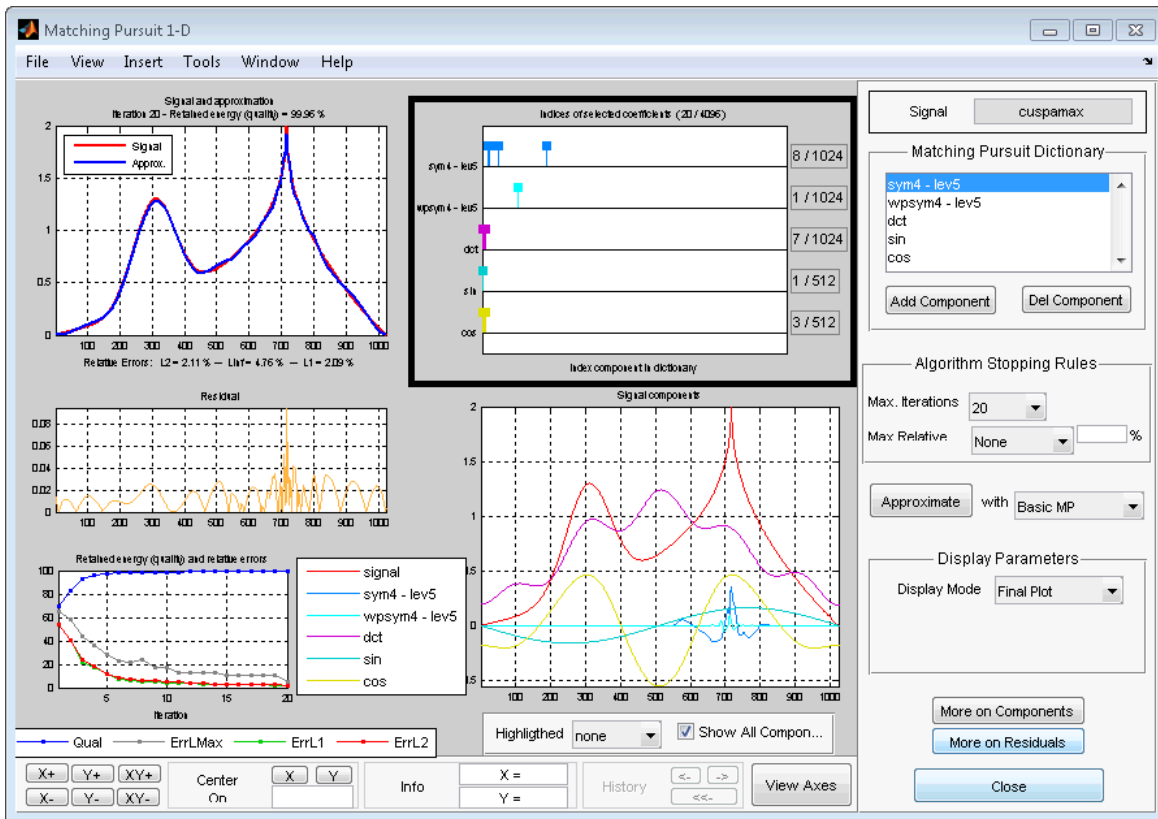
The left vertical axis shows the name of the subdictionary. The right vertical axis gives the ratio of selected vectors to the total number of vectors in the subdictionary. The location of the vertical bars along the horizontal axis gives the relative positions of the selected vectors in the subdictionaries.

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.
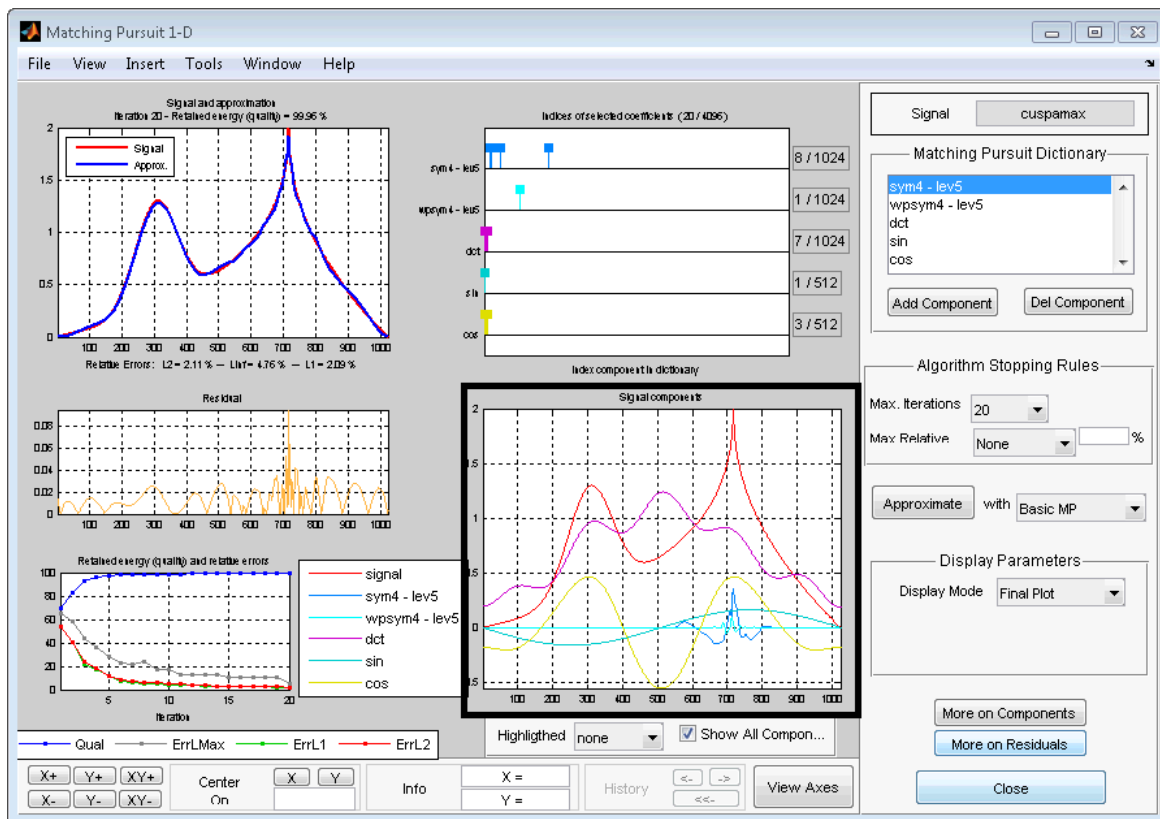
This plot enables you to assess the relative contribution of the subdictionaries to the signal approximation. In this example, you can see that the cosine and DCT subdictionaries contribute significantly to the approximation of the slowly-varying portions of the signal. The Daubechies least asymmetric 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.
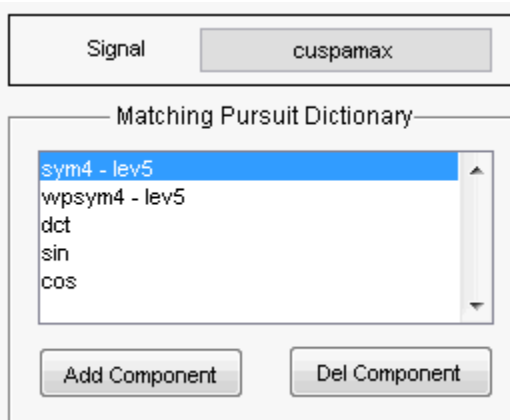
You have the ability to add or delete subdictionaries with **Add Component** and **Del Component**.

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 || || 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
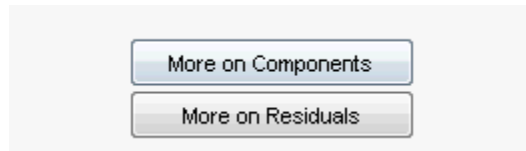
to obtain detailed interactive plots and information on the selected dictionary atoms and the final residual vector.

Click **More on Components**.
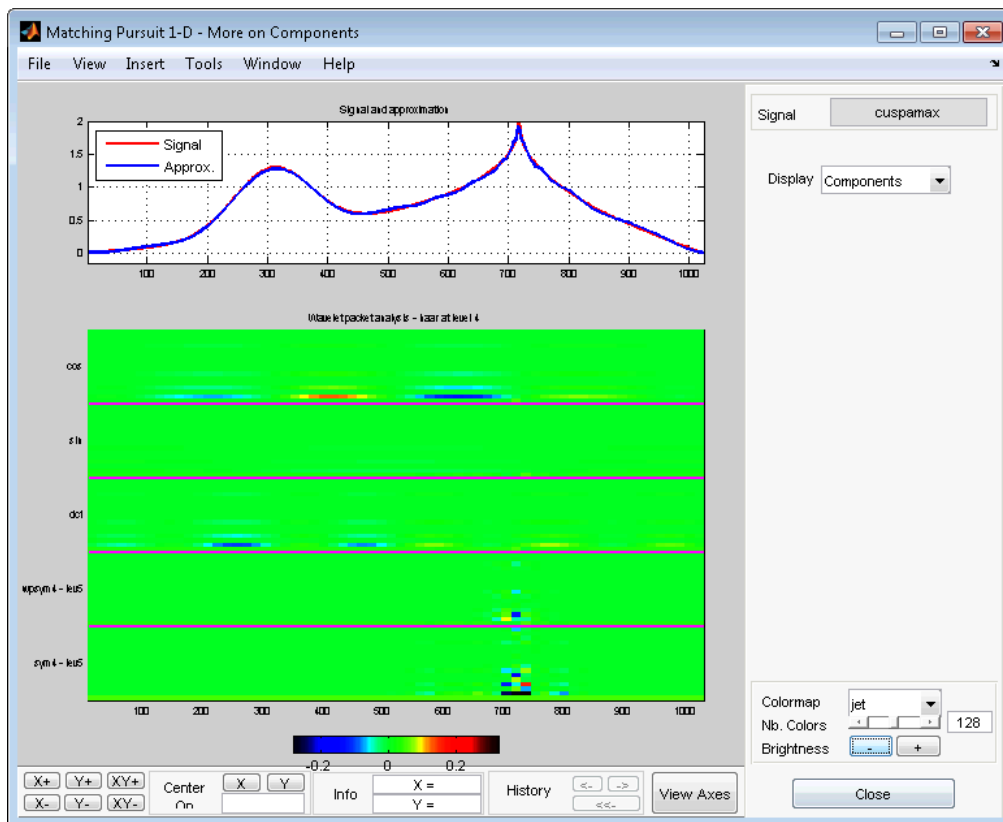
From the above figure, you can see that while the DCT and cosine subdictionaries contribute energy across the extent of the signal, the wavelet and wavelet packet contributions are localized at the cusp around sample 700. This result is expected because wavelets and wavelet packets excel at sparsely 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.

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**.
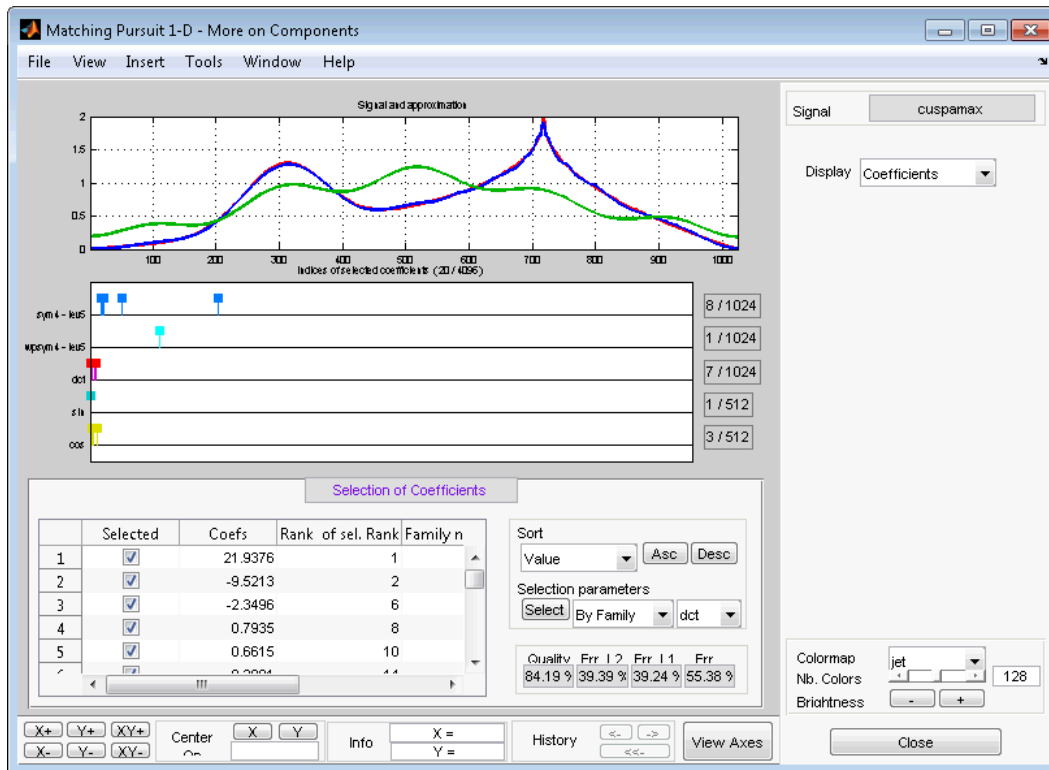
The DCT basis contributes significantly to the global approximation of the signal but the smooth DCT basis vectors are not able to sparsely represent the cusp.

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.
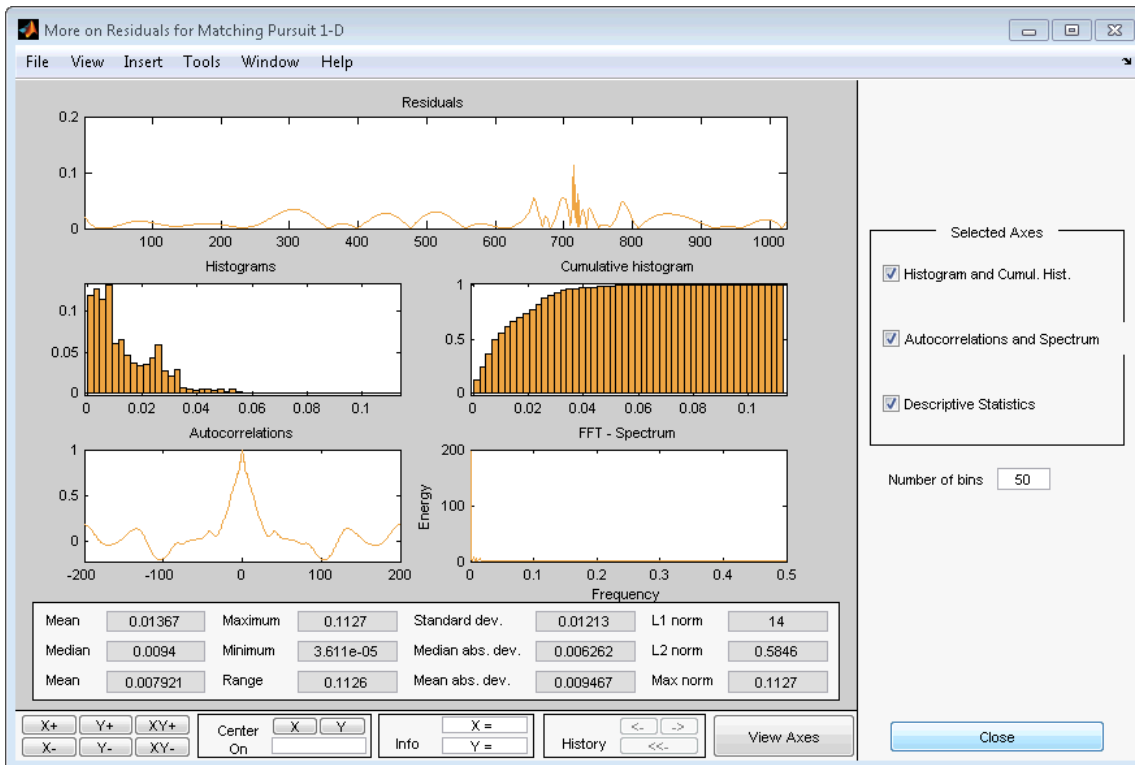
You can control which plots are displayed and the appearance of the histogram by the options in the right panel.
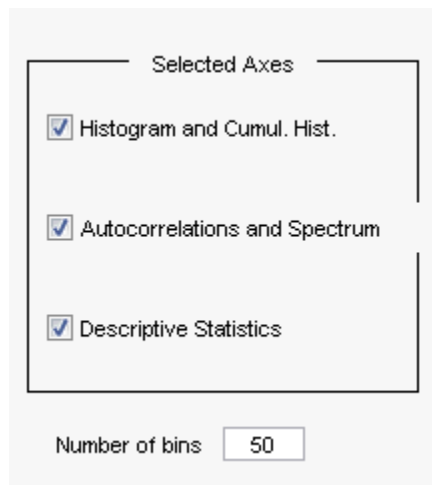
Interactive Matching Pursuit of Electricity Consumption Data

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.
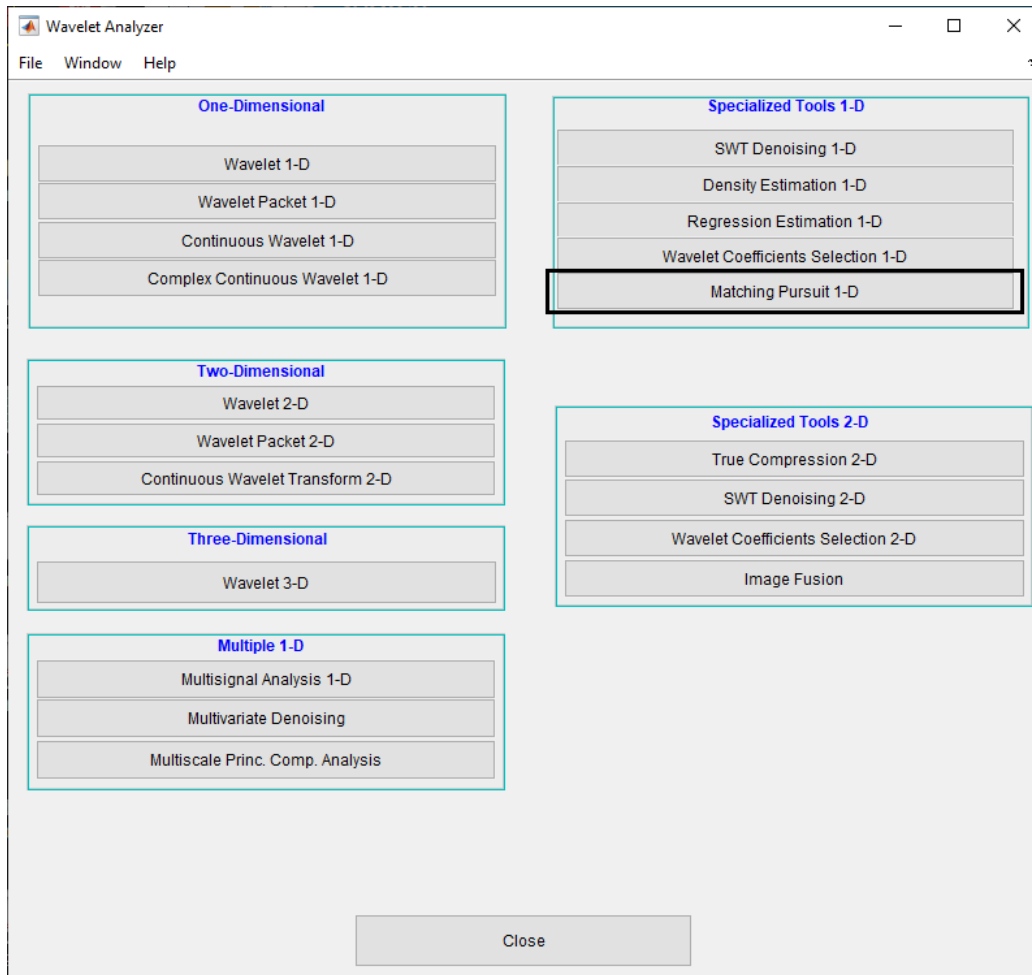
Select **File —> Import Signal from Workspace**

Load `x`.

Construct the following matching pursuit dictionary.
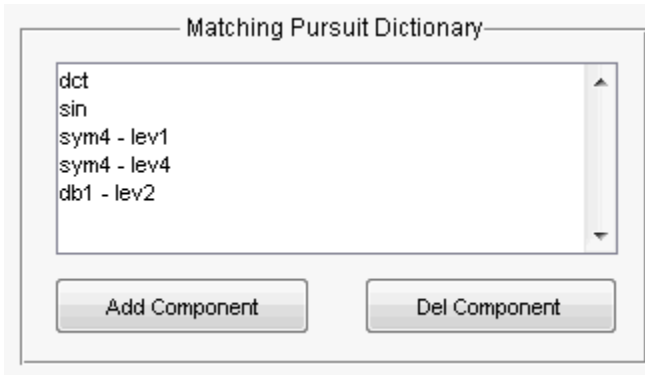
In the **Algorithm Stopping Rules** panel, set **Max. Iterations** to 30.

Select `Orthogonal MP` to use orthogonal matching pursuit.
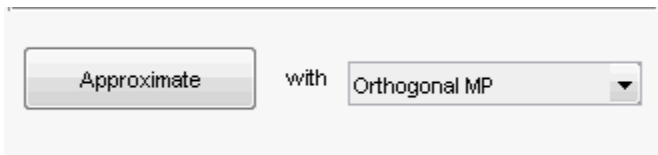
Click **Approximate**.
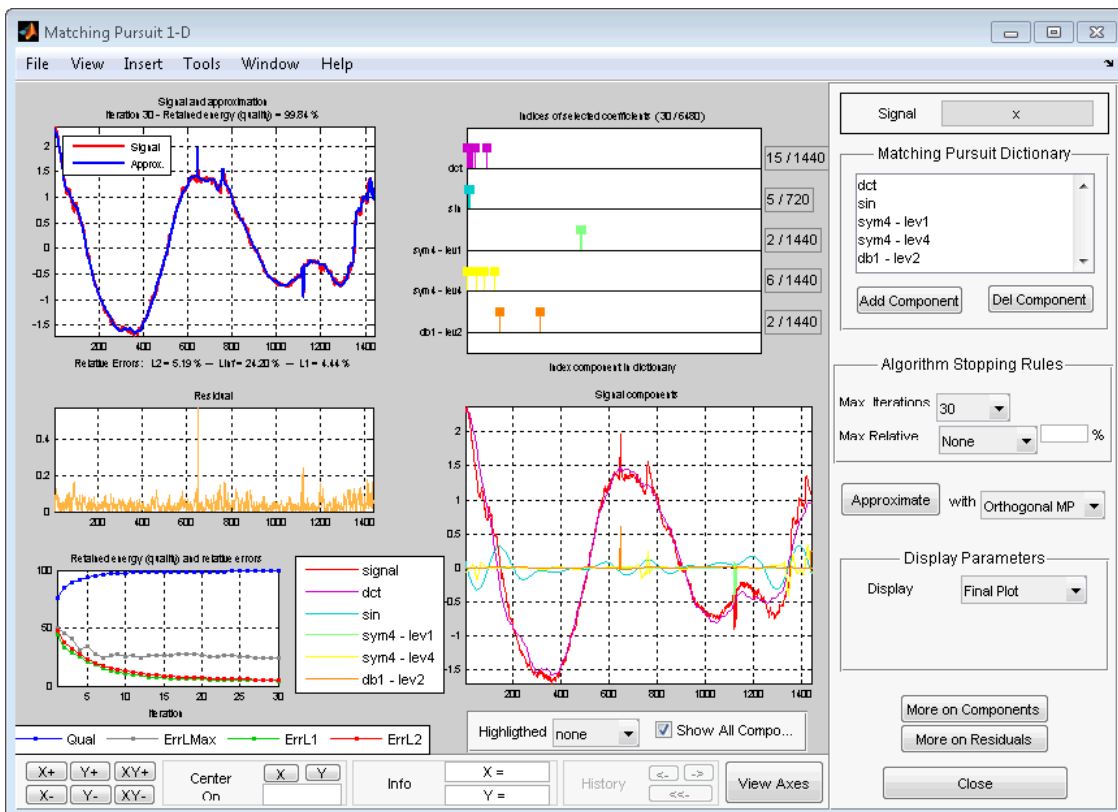
Code Generation from MATLAB Support in Wavelet Toolbox

Code Generation Support, Usage Notes, and Limitations

To generate code from MATLAB code that contains Wavelet Toolbox functions, you must have MATLAB Coder™.

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
centerFrequencies	CWT filter bank bandpass center frequencies
conv*	Convolution and polynomial multiplication
conv2	2-D convolution
cwtfilterbank*	Continuous wavelet transform filter bank
cwtfreqbounds*	CWT maximum and minimum frequency or period
ddencmp*	Default values for denoising or compression
detcoef	1-D detail coefficients
detcoef2	2-D detail coefficients
dualtree*	Kingsbury Q-shift 1-D dual-tree complex wavelet transform
dualtree2	Kingsbury Q-shift 2-D dual-tree complex wavelet transform
dwpt*	Multisignal 1-D wavelet packet transform
dwt*	Single-level 1-D discrete wavelet transform
dwt2*	Single-level discrete 2-D wavelet transform
dyadup*	Dyadic upsampling
emd*	Empirical mode decomposition
ewt*	Empirical wavelet transform
fft*	Fast Fourier transform
fft2*	2-D fast Fourier transform
fftshift	Shift zero-frequency component to center of spectrum
filter*	1-D digital filter
filter2	2-D digital filter
filterbank	Shearlet system filters
framebounds	Shearlet system frame bounds
freqz*	CWT filter bank frequency responses
haart	Haar 1-D wavelet transform
haart2	2-D Haar wavelet transform
hht*	Hilbert-Huang transform
idualtree	Kingsbury Q-shift 1-D inverse dual-tree complex wavelet transform
idualtree2	Kingsbury Q-shift 2-D inverse dual-tree complex wavelet transform
idwpt*	Multisignal 1-D inverse wavelet packet transform

idwt*	Single-level inverse discrete 1-D wavelet transform
idwt2*	Single-level inverse discrete 2-D wavelet transform
ifft*	Inverse fast Fourier transform
ifft2*	2-D inverse fast Fourier transform
ifftshift	Inverse zero-frequency shift
ihaart	Inverse 1-D Haar wavelet transform
ihaart2	Inverse 2-D Haar wavelet transform
imodwpt*	Inverse maximal overlap discrete wavelet packet transform
imodwt*	Inverse maximal overlap discrete wavelet transform
isheart2	Inverse shearlet transform
iswt*	Inverse discrete stationary wavelet transform 1-D
iswt2*	Inverse discrete stationary wavelet transform 2-D
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
modwtvar*	Multiscale variance of maximal overlap discrete wavelet transform
numshears	Number of shearlets
qbiorthfilt	First-level dual-tree biorthogonal filters
qfactor	CWT filter bank quality factor
qmf	Scaling and Wavelet Filter
qorthwavf	Kingsbury Q-shift filters
scales	CWT filter bank scales
scaleSpectrum*	Scale-averaged wavelet spectrum
shearletSystem	Cone-adapted bandlimited shearlet system
sheart2	Shearlet transform
swt*	Discrete stationary wavelet transform 1-D
swt2*	Discrete stationary wavelet transform 2-D
thselect	Threshold selection for denoising
timeSpectrum	Time-averaged wavelet spectrum
vmd*	Variational mode decomposition
wavedec*	1-D wavelet decomposition
wavedec2*	2-D wavelet decomposition
wavelets	CWT filter bank time-domain wavelets

waverec*	1-D wavelet reconstruction
waverec2*	2-D wavelet reconstruction
wcoherence*	Wavelet coherence and cross-spectrum
wden*	Automatic 1-D denoising
wdencmp*	Denoising or compression
wdenoise*	Wavelet signal denoising
wdenoise2*	Wavelet image denoising
wextend*	Extend vector or matrix
wnoisest	Estimate noise of 1-D wavelet coefficients
wt	Continuous wavelet transform with filter bank
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

Denoise Signal Using Generated C Code

This example shows how to denoise a signal using C code generated from the wdenoise function. Two techniques are demonstrated. The first technique uses a wrapper function which calls wdenoise. The second technique generates C code directly from wdenoise. Depending on your workflow, one technique may be preferred over the other. For example, to streamline code generation for large MATLAB® code, you can use wrapper functions to modularize the code.

Use Wrapper Function

Create a MATLAB function denoiseSignal that executes wdenoise and returns a denoised signal. The function takes two input arguments: a 1-D signal, and a denoising method. It passes these arguments to wdenoise. The function is included in the directory containing this example. The code for denoiseSignal follows.

```
function out = denoiseSignal(input,denMthd)
%#codegen
out = wdenoise(input,'DenoisingMethod',denMthd);
end
```

The %#codegen directive indicates that the function is intended for code generation.

Use codegen (MATLAB Coder) to generate a MEX function. Code generation defaults to MEX code generation when you do not specify a build target. By default, codegen names the generated MEX function denoiseSignal_mex. To allow the generation of MEX file, specify the properties (class, size, and complexity) of the two input parameters:

- coder.typeof(0,[1 Inf]) specifies a row vector of arbitrary length containing real double values.
- coder.typeof('c',[1 Inf]) specifies a character array of arbitrary length.

```
codegen denoiseSignal -args {coder.typeof(0,[1 Inf]),coder.typeof('c',[1 Inf])}
```

Load a signal.

```
load noisdopp
plot(noisdopp)
title('Signal')
```
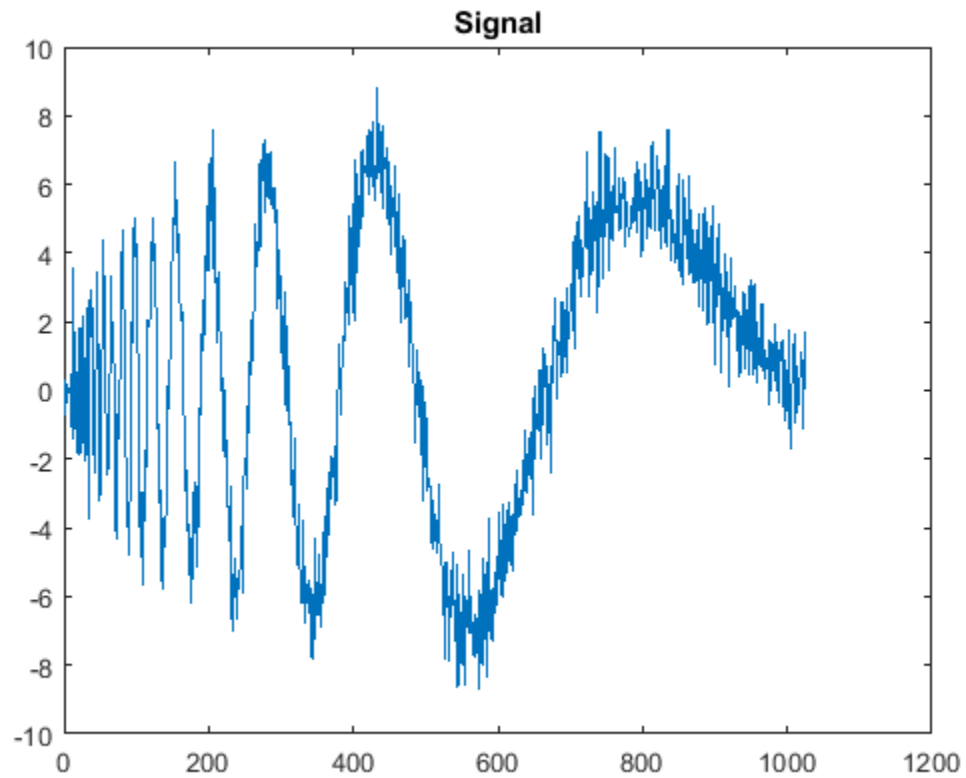
Signal

Denoise the signal twice using `denoiseSignal_mex`. First, use the denoising method `UniversalThreshold`. Then, use the SURE method. Plot both results.

```
dn = denoiseSignal_mex(noisdopp,'UniversalThreshold');
dn2 = denoiseSignal_mex(noisdopp,'SURE');
plot([dn' dn2'])
legend('Universal','SURE')
```
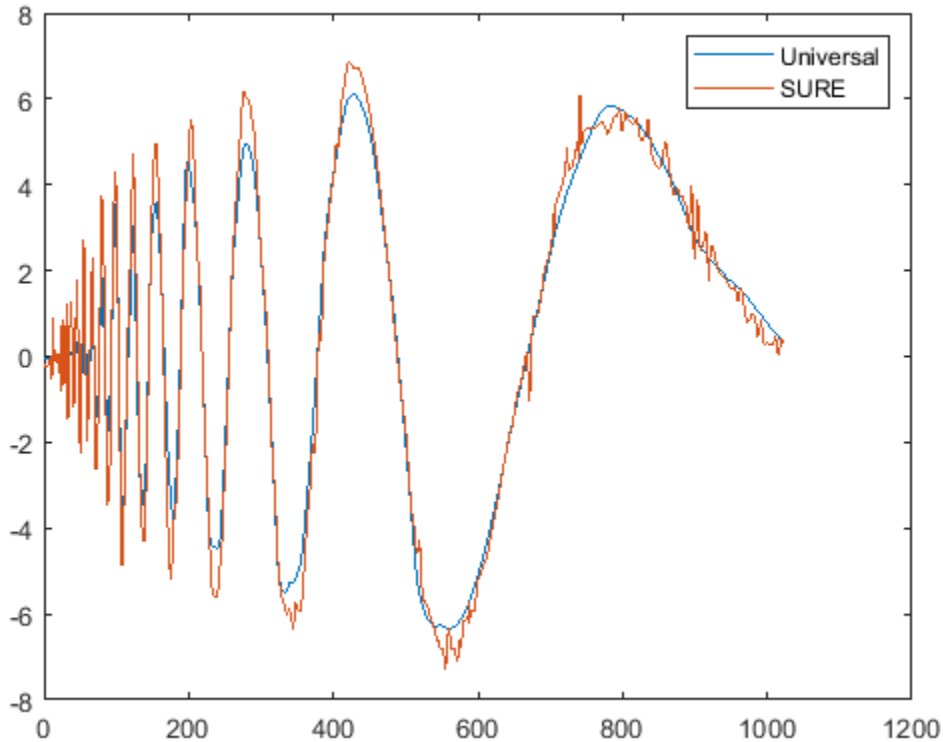
Use the MATLAB function and MEX function to denoise the signal using the `Minimax` method. Confirm the results are equal.

```
dnA = denoiseSignal_mex(noisdopp,'Minimax');
dnB = denoiseSignal(noisdopp,'Minimax');
max(abs(dnA-dnB))
```

```
ans = 6.2172e-15
```

Use Explicitly

You can also generate C code directly from `wdenoise`. Generate a MEX file that denoises a signal using the `db4` wavelet. The generated MEX file is called `wdenoise_mex`.

```
codegen wdenoise -args {coder.typeof(0,[1 Inf]),coder.Constant('Wavelet'),coder.Constant('db4')}
```

Denoise the original signal.

```
dn3 = wdenoise_mex(noisdopp,'Wavelet','db4');
```

To confirm the MEX file accepts variable length inputs, denoise the first 200 samples of the original signal.

```
dn4 = wdenoise_mex(noisdopp(1:200),'Wavelet','db4');
plot([noisdopp(1:200)' dn4'])
legend('Original','Denoised')
```
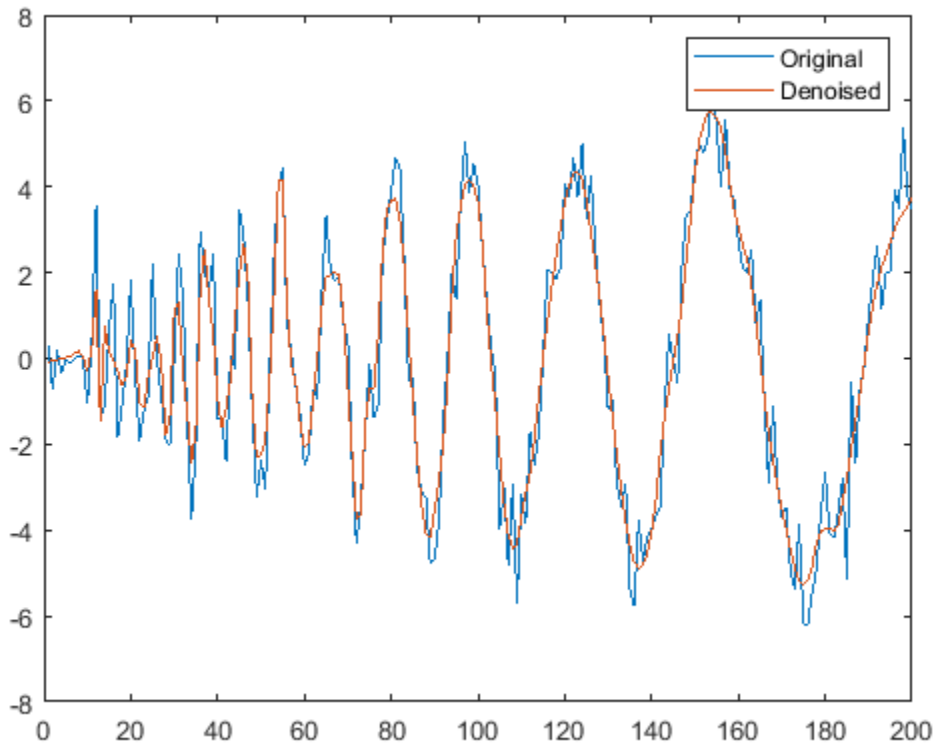
See Also
`wdenoise`

More About
- "Code Generation Support, Usage Notes, and Limitations" on page 8-2

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), wavelet packet transform (WPT), maximal overlap wavelet packet transform (MODWPT), and denoising workflows. This example requires a MATLAB Coder license.

Define a function that uses `wdenoise` 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 = wdenoise(x,level,'Wavelet',wname,...
       'DenoisingMethod','SURE',...
       'ThresholdRule','soft',...
       'NoiseEstimate','LevelIndependent');
   end
   ```

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]),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')
   ```
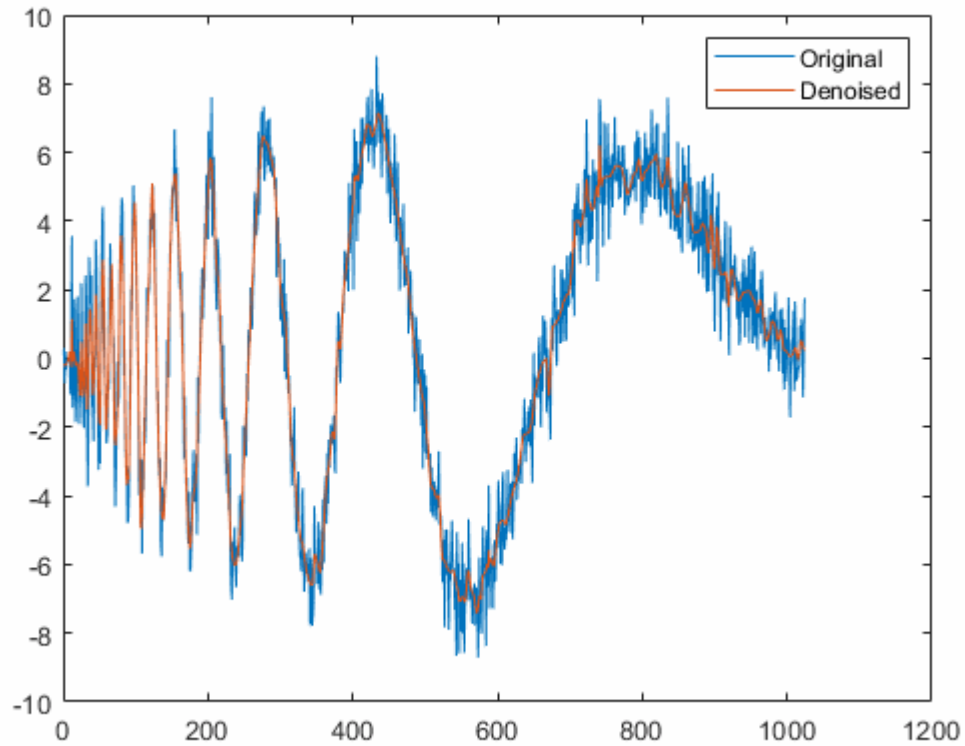
For a list of Wavelet Toolbox functions supported for code generation and associated limitations, see "Code Generation Support, Usage Notes, and Limitations" on page 8-2. For more information on code generation, see "Get Started with MATLAB Coder" (MATLAB Coder).

See Also

wdenoise

More About

CUDA Code from CWT

This example shows how to generate a MEX file to perform the continuous wavelet transform (CWT) using generated CUDA code.

First, ensure that you have a CUDA-enabled GPU and the NVCC compiler. See "The GPU Environment Check and Setup App" (GPU Coder) to ensure you have the proper configuration.

Create a GPU coder configuration object.

```
cfg = coder.gpuConfig('mex');
```

Generate a signal of 100,000 samples at 1,000 Hz. The signal consists of two cosine waves with disjoint time supports.

```
t = 0:.001:(1e5*0.001)-0.001;
x = cos(2*pi*32*t).*(t > 10 & t<=50)+cos(2*pi*64*t).*(t >= 60 & t < 90)+ ...
    0.2*randn(size(t));
```

Cast the signal to use single precision. GPU calculations are often more efficiently done in single precision. You can however also generate code for double precision if your NVIDIA GPU supports it.

```
x = single(x);
```

Generate the GPU MEX file and a code generation report. To allow generation of the MEX file, you must specify the properties (class, size, and complexity) of the three input parameters:

- `coder.typeof(single(0),[1 1e5])` specifies a row vector of length 100,000 containing real `single` values.
- `coder.typeof('c',[1 inf])` specifies a character array of arbitrary length.
- `coder.typeof(0)` specifies a real `double` value.

```
codegen cwt -config cfg -args {coder.typeof(single(0),[1 1e5]),coder.typeof('c',[1 inf]),coder.ty
```

```
Code generation successful: To view the report, open('codegen/mex/cwt/html/report.mldatx').
```

The -report flag is optional. Using `-report` generates a code generation report. In the **Summary** tab of the report, you can find a **GPU code metrics** link, which provides detailed information such as the number of CUDA kernels generated and how much memory was allocated.

Run the MEX file on the data and plot the scalogram. Confirm the plot is consistent with the two disjoint cosine waves.

```
[cfs,f] = cwt_mex(x,'Morse',1e3);
image('XData',t,'YData',f,'CData',abs(cfs),'CDataMapping','scaled')
set(gca,'YScale','log')
axis tight
xlabel('Seconds')
ylabel('Hz')
title('CWT of Two-Tone Signal')
```

CWT of Two-Tone Signal
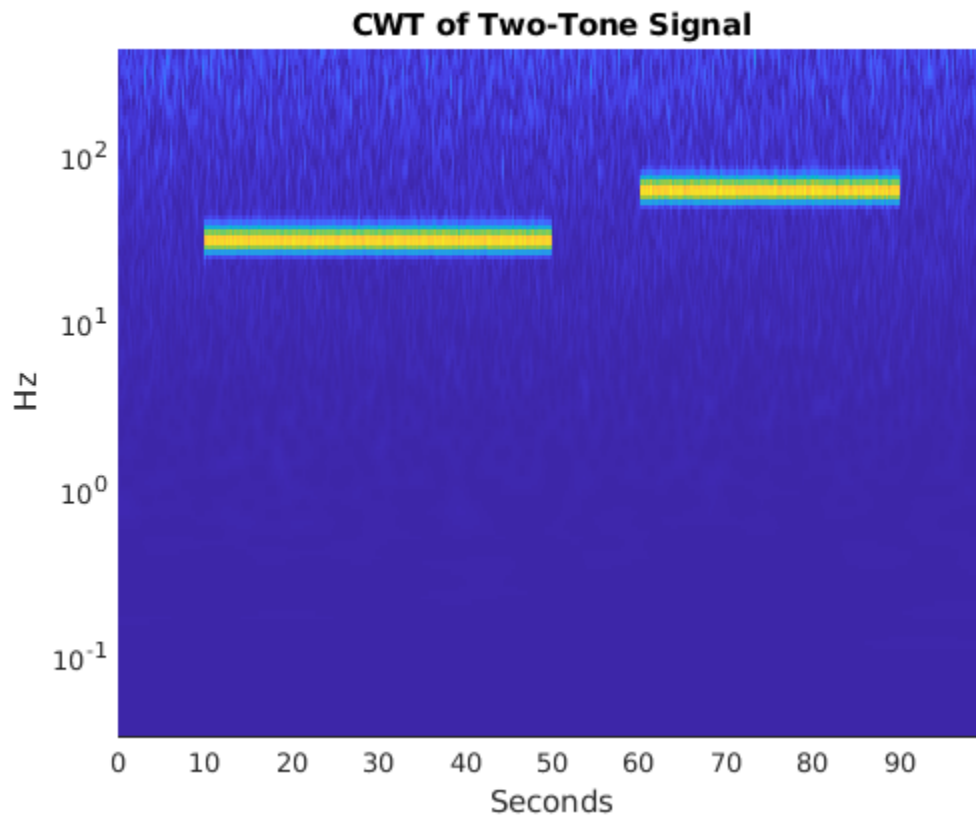
Run the CWT command above without appending the _mex. Confirm the MATLAB and the GPU MEX scalograms are identical.

```
[cfs2,f2] = cwt(x,'Morse',1e3);
max(abs(cfs2(:)-cfs(:)))
```

```
ans =

  single

  6.7583e-07
```

See Also

cwt | cwtfilterbank

Special Topics

Wavelet Scattering

A wavelet scattering framework enables you to derive, with minimal configuration, low-variance features from real-valued time series and image data for use in machine learning and deep learning applications. The features are insensitive to translations of the input on an invariance scale that you define and are continuous with respect to deformations. In the 2-D case, features are also insensitive to rotations. The scattering framework uses predefined wavelet and scaling filters.

Mallat, with Bruna and Andén, pioneered the creation of a mathematical framework for studying convolutional neural architectures [2][3][4][5]. Andén and Lostanlen developed efficient algorithms for wavelet scattering of 1-D signals [4] [6]. Oyallon developed efficient algorithms for 2-D scattering [7]. Andén, Lostanlen, and Oyallon are major contributors to the ScatNet [10] and Kymatio [11] software for computing scattering transforms.

Mallat and others characterized three properties that deep learning architectures possess for extracting useful features from data:

- Multiscale contractions
- Linearization of hierarchical symmetries
- Sparse representations

The wavelet scattering framework exhibits all these properties. Wavelet transforms linearize small deformations such as dilations by separating the variations across different scales. For many natural signals, the wavelet transform also provides a sparse representation. By combining wavelet transforms with other features of the scattering framework described below, the scattering transform produces data representations that minimize differences *within* a class while preserving discriminability *across* classes. An important distinction between the scattering transform and deep learning frameworks is that the filters are defined a priori as opposed to being learned. Because the scattering transform is not required to learn the filter responses, you can often use scattering successfully in situations where there is a shortage of training data.

Wavelet Scattering Transform

A wavelet scattering transform processes data in stages. The output of one stage becomes input for the next stage. Each stage consists of three operations.
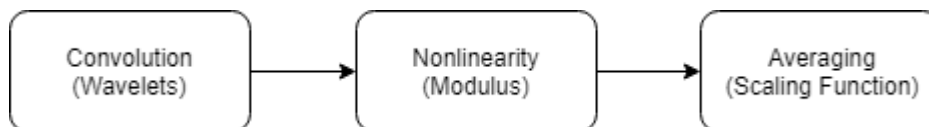
The zeroth-order scattering coefficients are computed by simple averaging of the input. Here is a tree view of the algorithm:

$f * \phi_J \quad \leftarrow - -$

$|f * \psi_{1,1}| * \phi_J \quad \leftarrow - -$
$|f * \psi_{1,2}| * \phi_J$
$|f * \psi_{1,3}| * \phi_J$
$\vdots$

$||f * \psi_{1,1}| * \psi_{2,1}| * \phi_J \quad \leftarrow - -$
$||f * \psi_{1,1}| * \psi_{2,2}| * \phi_J$
$||f * \psi_{1,1}| * \psi_{2,3}| * \phi_J$
$\vdots$
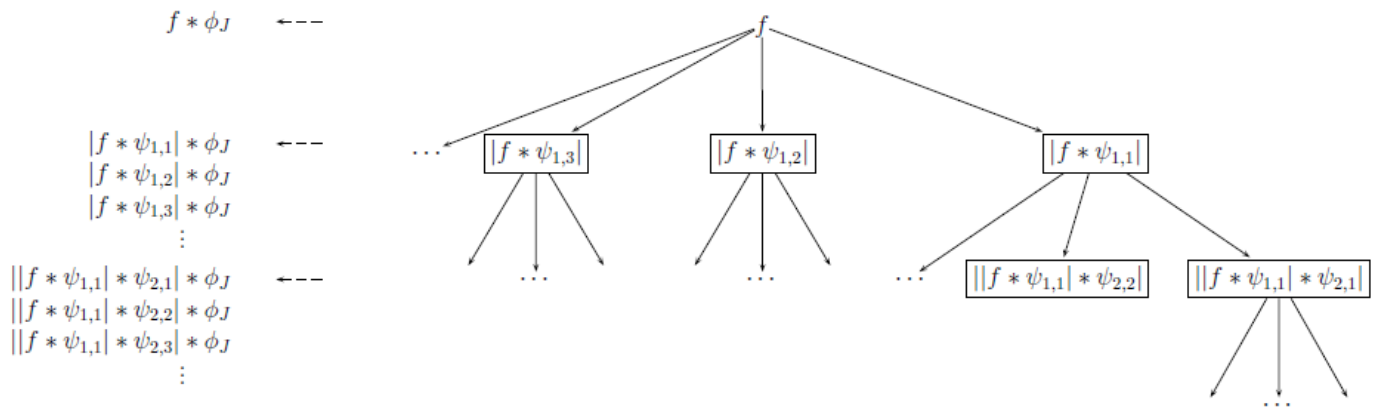
The $\{\psi_{j,k}\}$ are wavelets, ϕ_J is the scaling function, and f is the input data. In the case of image data, for each $\psi_{j,k}$, there are a number of user-specified rotations of the wavelet. A sequence of edges from the root to a node is referred to as a *path*. The tree nodes are the *scalogram coefficients*. The *scattering coefficients* are the scalogram coefficients convolved with the scaling function ϕ_J. The set of scattering coefficients are the low-variance features derived from the data. Convolution with the scaling function is lowpass filtering and information is lost. However, the information is recovered when computing the coefficients in the next stage.

To extract features from the data, first use `waveletScattering` (for time series) or `waveletScattering2` (for image data) to create and configure the framework. Parameters you set include the size of invariance scale, the number of filter banks, and the number of wavelets per octave in each filter bank. In `waveletScattering2` you can also set the number of rotations per wavelet. To derive features from time series, use the `waveletScattering` object functions `scatteringTransform` or `featureMatrix`. To derive features from image data, use the `waveletScattering2` object functions `scatteringTransform` or `featureMatrix`.

The scattering transform generates features in an iterative fashion. First, you convolve the data with the scaling function, $f * \phi_J$ to obtain S[0], the zeroth-order scattering coefficients. Next, proceed as follows:

1 Take the wavelet transform of the input data with each wavelet filter in the first filter bank.

2 Take the modulus of each of the filtered outputs. The nodes are the scalogram, U[1].

3 Average each of the moduli with the scaling filter. The results are the first-order scattering coefficients, S[1].

Repeat the process at every node.

The `scatteringTransform` function returns the scattering and scalogram coefficients. The `featureMatrix` function returns the scattering features. Both outputs can be made easily consumable by learning algorithms, as demonstrated in "Wavelet Time Scattering for ECG Signal Classification" on page 13-17 or "Texture Classification with Wavelet Image Scattering" on page 13-97.

Invariance Scale

The scaling filter plays a crucial role in the wavelet scattering framework. When you create a wavelet scattering framework, you specify the invariance scale. The framework is invariant to translations up

to the invariance scale. The support of the scaling function determines the size of the invariant in time or space.

Time Invariance

For time series data, the invariance scale is a duration. The time support of the scaling function does not exceed the size of the invariant. This plot shows the support of the scaling function in a framework with an invariance scale of two seconds and a sampling frequency of 100 Hz. Also shown are the real and imaginary parts of the coarsest-scale wavelet from the first filter bank. Observe the time supports of the functions do not exceed two seconds.
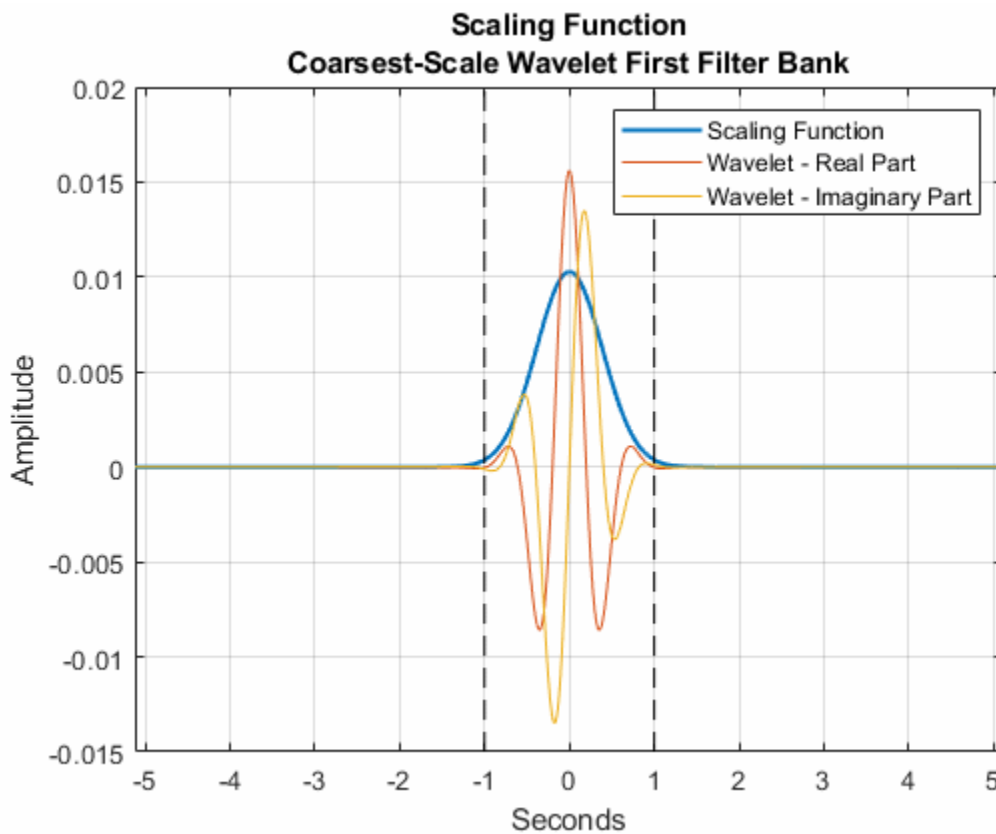
The invariance scale also affects the spacings of the center frequencies of the wavelets in the filter banks. In a filter bank created by `cwtfilterbank`, the bandpass center frequencies are logarithmically spaced and the bandwidths of the wavelets decrease with center frequency.

In a scattering framework, however, the time support of a wavelet cannot exceed the invariance scale. This property is illustrated in the coarsest-scale wavelet plot. Frequencies lower than the invariant scale are linearly spaced with scale held constant so that the size of the invariant is not exceeded. The next plot shows the center frequencies of the wavelets in the first filter bank in the scattering framework. The center frequencies are plotted on linear and logarithmic scales. Note the logarithmic spacing of the higher center frequencies and the linear spacing of the lower center frequencies.

First Filter Bank Center Frequencies - Linear Scale

First Filter Bank Center Frequencies - Logarithmic Scale
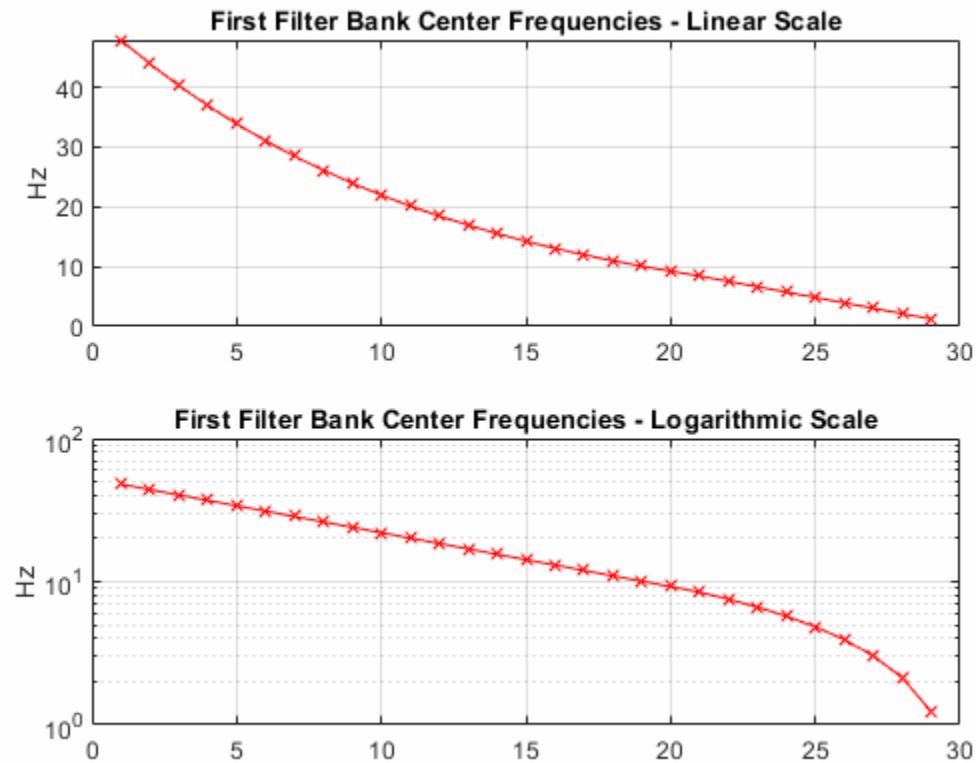
Image Invariance

For image data, the invariance scale specifies the *N*-by-*N* spatial support, in pixels, of the scaling filter. For example, by default the `waveletScattering2` function creates a wavelet image scattering framework for image size 128-by-128 and an invariance scale of 64. The following surface plot shows the scaling function used in the framework. The intersecting red lines form a 64-by-64 square.

$$\phi(x, y)$$
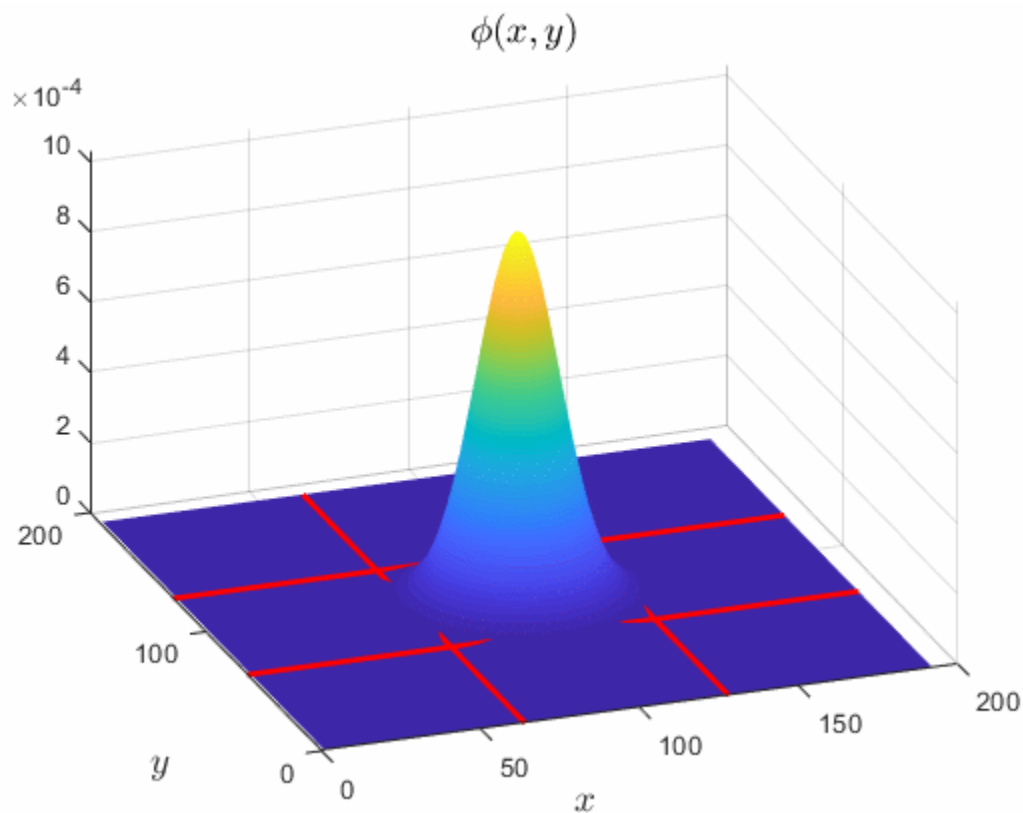
Quality Factors and Filter Banks

When creating a wavelet scattering framework, in addition to the invariance scale, you also set the quality factors for the scattering filter banks. The quality factor for each filter bank is the number of wavelet filters per octave. The wavelet transform discretizes the scales using the specified number of wavelet filters.

This plot shows the wavelet filters in the framework created by `waveletScattering`. The invariance scale is one second and sampling frequency is 200 Hz. The first filter bank has the default quality value of 8, and the second filter bank has the default quality factor of 1.

1st Filter Bank Wavelet Filters

2nd Filter Bank Wavelet Filters
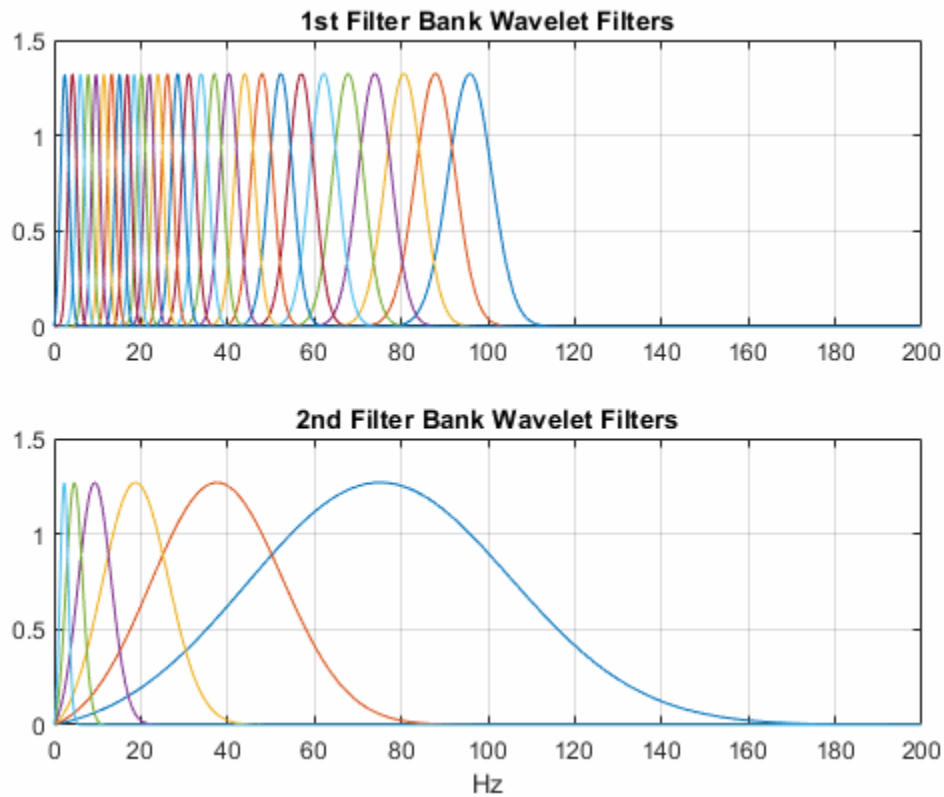
For image data, large quality factors are not necessary. Large values also result in significant computational overhead. By default `waveletScattering2` creates a framework with two filter banks each with a quality factor of 1. This plot shows the wavelet center frequencies for a wavelet image scattering framework with two filter banks. The first filter bank has a quality factor of 2, and the second filter bank has a quality factor of 1. The number of rotations per filter bank is 6.
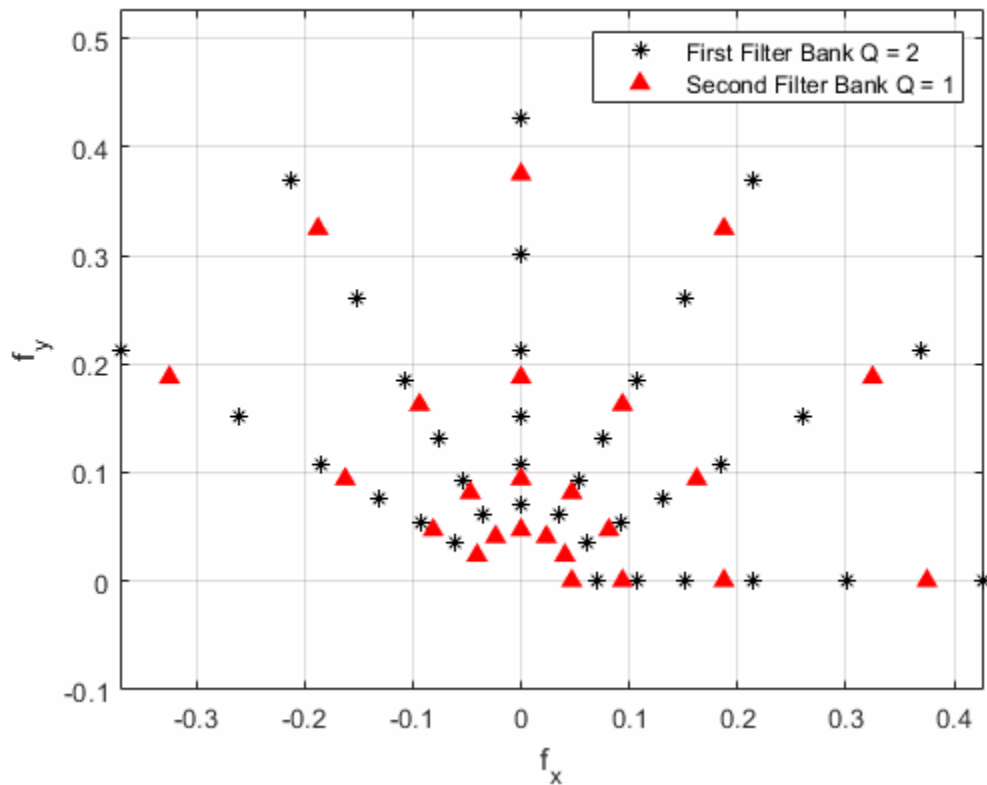
In Practice

With the proper choice of wavelets, the scattering transform is nonexpansive. Energy dissipates as you iterate through the framework. As the order m increases, the energy of the mth-order scalogram coefficients and scattering coefficients rapidly converges to 0 [3]. Energy dissipation has a practical benefit. You can limit the number of wavelet filter banks in the framework with a minimal loss of signal energy. Published results show that the energy of the third-order scattering coefficients can fall below one percent. For most applications, a framework with two wavelet filter banks is sufficient.

Consider the tree view of the wavelet time scattering framework. Suppose that there are M wavelets in the first filter bank, and N wavelets in the second filter bank. The number of wavelet filters in each filter bank do not have to be large before a naive implementation becomes unfeasible. Efficient implementations take advantage of the lowpass nature of the modulus function and critically downsample the scattering and scalogram coefficients. These strategies were pioneered by Andén, Mallat, Lostanlen, and Oyallon [4] [6] [7] in order to make scattering transforms computationally practical while maintaining their ability to produce low-variance data representations for learning. By default, `waveletScattering` and `waveletScattering2` create frameworks that critically downsample the coefficients.

References

[1] LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Handwritten Digit Recognition with a Back-Propagation Network." In *Advances in Neural*

Information Processing Systems (NIPS 1989) (D. Touretzky, ed.). 396–404. Denver, CO: Morgan Kaufmann, Vol 2, 1990.

[2] Mallat, S. "Group Invariant Scattering." *Communications in Pure and Applied Mathematics*. Vol. 65, Number 10, 2012, pp. 1331–1398.

[3] Bruna, J., and S. Mallat. "Invariant Scattering Convolution Networks." *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 35, Number 8, 2013, pp. 1872–1886.

[4] Andén, J., and S. Mallat. "Deep Scattering Spectrum." *IEEE Transactions on Signal Processing*. Vol. 62, Number 16, 2014, pp. 4114–4128.

[5] Mallat, S. "Understanding deep convolutional networks." *Philosophical Transactions of the Royal Society A.* Volume 374: 20150203, 2016, pp. 1–16. dx.doi.org/10.1098/rsta.2015.0203.

[6] Lostanlen, V. *Scattering.m — a MATLAB toolbox for wavelet scattering.* https://github.com/lostanlen/scattering.m.

[7] Oyallon, Edouard. *Webpage of Edouard Oyallon*. https://edouardoyallon.github.io/.

[8] Sifre, L., and S. Mallat. "Rigid-Motion Scattering for Texture Classification". arXiv preprint. 2014, pp. 1–19. https://arxiv.org/abs/1403.1687.

[9] Sifre, L., and S. Mallat. "Rotation, scaling and deformation invariant scattering for texture discrimination." *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp 1233–1240.

[10] *ScatNet*. https://www.di.ens.fr/data/software/scatnet/.

[11] *Kymatio*. https://www.kymat.io/.

See Also
waveletScattering | waveletScattering2

Related Examples

Wavelet Scattering Invariance Scale and Oversampling

This example shows how changing the invariance scale and oversampling factor affects the output of the wavelet scattering transform.

Invariance Scale

The `InvarianceScale` property of a wavelet time scattering framework sets the time scale of the scaling (lowpass) filter. Create a wavelet scattering framework with a signal length of 10,000 and invariance scale of 500. Obtain the filter bank.

```
sigLength = 1e4;
sf = waveletScattering('SignalLength',sigLength,'InvarianceScale',500);
[fb,f] = filterbank(sf);
```

Use the helper function `helperPlotScalingWavelet` to plot the scaling filter in time along with the real and imaginary parts of the coarsest-scale wavelet from the first filter bank. The source code for `helperPlotScalingWavelet` is listed in the appendix. The supports of the scaling filter and wavelet are essentially the size of the invariance scale.

```
helperPlotScalingWavelet(fb,f,500)
```
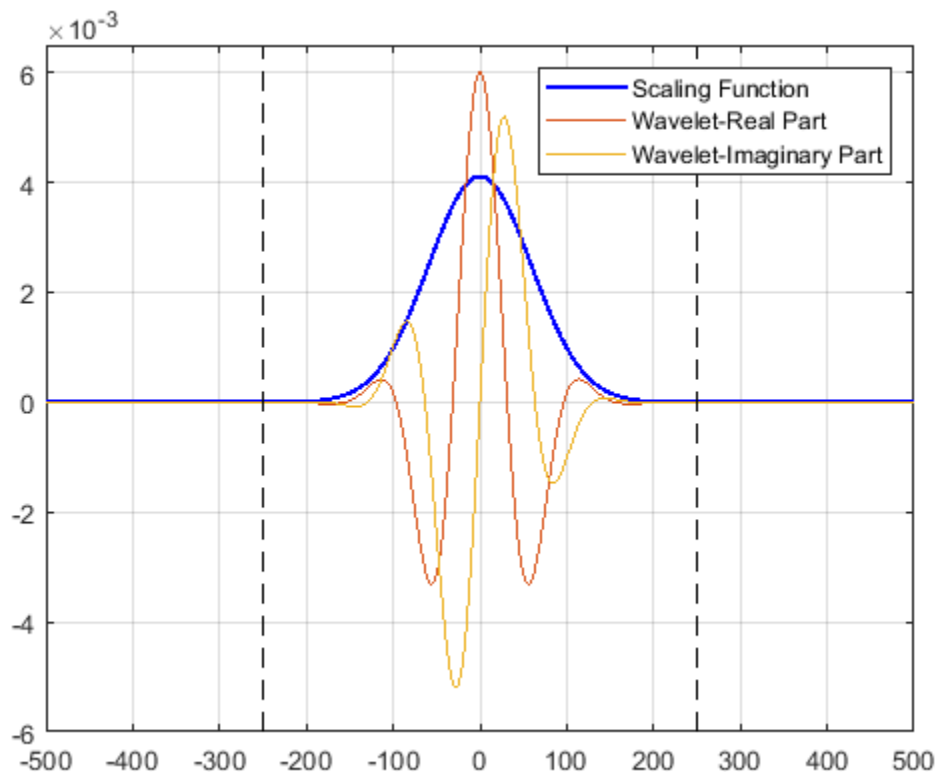

Generate a random signal and use `featureMatrix` to obtain the scattering feature matrix for the signal and scattering framework.

```
x = randn(1,sigLength);
smat = featureMatrix(sf,x);
whos smat
```

```
  Name          Size                Bytes  Class     Attributes

  smat        109x79                68888  double
```

Each row of the feature matrix is a vector which has been convolved (filtered) with the lowpass filter (after other wavelet filtering). The second dimension of the feature matrix is the time resolution. The output of the filtering is downsampled as much as possible without aliasing, what is called "critically downsampling". The amount of downsampling depends on the bandwidth of the filter. The bigger the invariance scale, the larger the time support of the lowpass (scaling) function and accordingly the more we can downsample.

Obtain the scattering transform of the signal.

```
[S,~] = scatteringTransform(sf,x);
S{2}
```

ans=40×4 table

signals	path		bandwidth	resolution
{79x1 double}	0	1	0.0084478	-7
{79x1 double}	0	2	0.0084478	-7
{79x1 double}	0	3	0.0084478	-7
{79x1 double}	0	4	0.0084478	-7
{79x1 double}	0	5	0.0084478	-7
{79x1 double}	0	6	0.0084478	-7
{79x1 double}	0	7	0.0084478	-7
{79x1 double}	0	8	0.0084478	-7
{79x1 double}	0	9	0.0084478	-7
{79x1 double}	0	10	0.0084478	-7
{79x1 double}	0	11	0.0084478	-7
{79x1 double}	0	12	0.0084478	-7
{79x1 double}	0	13	0.0084478	-7
{79x1 double}	0	14	0.0084478	-7
{79x1 double}	0	15	0.0084478	-7
{79x1 double}	0	16	0.0084478	-7
⋮				

The scattering coefficient vectors, `signals`, have length 79, and the resolution is -7. This means that we expect approximately $10^4/2^7$ coefficients in each vector.

Create a wavelet scattering framework with an invariance scale of 200. Obtain the scattering transform of the signal. Because the invariance scale is smaller, the bandwidth of the scaling filter increases and we cannot downsample as much without aliasing. Therefore, the number of scattering coefficients increases.

```
sf = waveletScattering('SignalLength',sigLength,'InvarianceScale',200);
[S,~] = scatteringTransform(sf,x);
S{2}
```

ans=29×4 table

signals	path	bandwidth	resolution

9-11

signals		path	bandwidth	resolution
{157x1 double}	0	1	0.02112	-6
{157x1 double}	0	2	0.02112	-6
{157x1 double}	0	3	0.02112	-6
{157x1 double}	0	4	0.02112	-6
{157x1 double}	0	5	0.02112	-6
{157x1 double}	0	6	0.02112	-6
{157x1 double}	0	7	0.02112	-6
{157x1 double}	0	8	0.02112	-6
{157x1 double}	0	9	0.02112	-6
{157x1 double}	0	10	0.02112	-6
{157x1 double}	0	11	0.02112	-6
{157x1 double}	0	12	0.02112	-6
{157x1 double}	0	13	0.02112	-6
{157x1 double}	0	14	0.02112	-6
{157x1 double}	0	15	0.02112	-6
{157x1 double}	0	16	0.02112	-6
⋮				

Oversampling Factor

Because the invariance scale is such an important hyperparameter for scattering frameworks (one of the most important for performance), you should set the value based on the problem at hand and not because you want a certain number of coefficients. You can use the `OversamplingFactor` property for adjusting the number of coefficients for a given `InvarianceScale`. The `OversamplingFactor` specifies how much the scattering coefficients are oversampled with respect to the critically downsampled values. The factor is on a $\log_2$ scale.

Create a scattering framework with an invariance scale of 500, and an `OversamplingFactor` equal to 1. Obtain the scattering transform of the signal. As expected, the number of scattering paths is greater than in the case where `InvarianceScale` is 200. By setting `OversamplingFactor` to 1, the scattering transform returns two times as many coefficients for each scattering path with respect to the critically sampled number. The size of the scattering coefficient vectors returned is equal to the size when the scattering framework has an invariance scale of 200 with default critical downsampling.

```
sf = waveletScattering('SignalLength',sigLength,'InvarianceScale',500,'OversamplingFactor',1);
[S,~] = scatteringTransform(sf,x);
S{2}
```

```
ans=40×4 table
```

signals		path	bandwidth	resolution
{157x1 double}	0	1	0.0084478	-6
{157x1 double}	0	2	0.0084478	-6
{157x1 double}	0	3	0.0084478	-6
{157x1 double}	0	4	0.0084478	-6
{157x1 double}	0	5	0.0084478	-6
{157x1 double}	0	6	0.0084478	-6
{157x1 double}	0	7	0.0084478	-6
{157x1 double}	0	8	0.0084478	-6
{157x1 double}	0	9	0.0084478	-6
{157x1 double}	0	10	0.0084478	-6
{157x1 double}	0	11	0.0084478	-6

```
{157x1 double}    0    12    0.0084478         -6
{157x1 double}    0    13    0.0084478         -6
{157x1 double}    0    14    0.0084478         -6
{157x1 double}    0    15    0.0084478         -6
{157x1 double}    0    16    0.0084478         -6
        ⋮
```

Appendix

```matlab
function helperPlotScalingWavelet(fb,f,invScale)
% This function is in support of wavelet scattering examples only. It may
% change or be removed in a future release.
fBin = diff(f(1:2));
time = (-1/2:fBin:1/2-fBin)*1e4;
phi = ifftshift(ifft(fb{1}.phift));
psiL1 = ifftshift(ifft(fb{2}.psift(:,end)));
figure
plot(time,phi,'b','LineWidth',1.5)
grid on
hold on
plot(time,real(psiL1));
plot(time,imag(psiL1));
plot([-invScale/2 -invScale/2],[-6e-3 6.5e-3],'k--')
plot([invScale/2 invScale/2],[-6e-3 6.5e-3],'k--')
ylim([-6e-3 6.5e-3])
xlim([-invScale invScale])
legend('Scaling Function','Wavelet-Real Part','Wavelet-Imaginary Part')
end
```

See Also

filterbank | scatteringTransform | waveletScattering

Empirical Wavelet Transform

The empirical wavelet transform (EWT) is a technique that creates a multiresolution analysis (MRA) of a signal using an adaptive wavelet subdivision scheme. The EWT starts with a segmentation of the signal's spectrum. The EWT provides perfect reconstruction of the input signal. The EWT coefficients partition the energy of the input signal into separate passbands.

The EWT was developed by Gilles [1 on page 9-0   ]. Gilles and Heal [3 on page 9-0   ] proposed and use a histogram-based approach for segmenting the spectrum.

An MRA is a decomposition of a signal into components on different scales, or equivalently, on different frequency bands in such a way that the original signal is recovered by summing the components at each point in time (see "Practical Introduction to Multiresolution Analysis" on page 11-2). Many MRA techniques exist. The maximal overlap discrete wavelet transform (MODWT) and its associated MRA formulation use a basis or frame designed independently of the signal (see `modwt` and `modwtmra`). The empirical mode decomposition (EMD) algorithm is a data-adaptive technique that decomposes a nonlinear or nonstationary process into its intrinsic modes of oscillation. The EMD iterates on an input signal to extract natural AM-FM modes, also known as intrinsic mode functions, contained in the data (see `emd`).

EWT Algorithm

You can use the `ewt` function to obtain the MRA of a signal. The structure of the EWT algorithm is as follows.

1 We obtain a multitaper power spectral estimate of the signal using five sine tapers. This is a smooth, low-variance estimate of the power spectrum (see "Bias and Variability in the Periodogram" (Signal Processing Toolbox)). We normalize the estimate to lie in the range [0,1]. By default, we identify all peaks strictly greater than 70% of the peak value. If there is a dominant peak and many smaller ones, you can use the "'LogSpectrum'" option.

2 The empirical wavelet passbands are constructed by default, so that their transition bands cross at the geometric mean frequency of the adjacent peaks. The Meyer wavelets are constructed as described in [1 on page 9-0   ] along with the way we determine the γ parameter. The wavelets are overlapped in such a way that they form a Parseval tight frame.

3 For determining the boundaries between adjacent passbands, you have the option of using the first local minima between adjacent peaks. If no local minimum is identified, we revert to the geometric mean (default).

4 You also have the option to override the automatic thresholding of peaks using "'MaxNumPeaks'". The largest peaks up to `MaxNumPeaks` are used. The approximate bandwidth of the multitaper estimate is (K+1/2)/(N+1), where K is the number of tapers, and N is the data length (which may include padding). Because peaks must be minimally separated by the approximate bandwidth to qualify as a peak, it is possible that fewer than `MaxNumPeaks` peaks, including no peaks, are identified.

Because the wavelets form a Parseval tight frame, the filter bank is self-dual: the analysis filter bank is equal to the synthese filter bank. The EWT uses the wavelets to filter the signal in the frequency domain and then inverts the transform to obtain the analysis coefficients. The EWT uses the corresponding synthesis wavelets to reconstruct the MRA components.

Spectrum Segmentation

If you have Signal Processing Toolbox™, you can see how using multitapers can produce a smoothed estimate of the power spectrum.

Create the third test signal defined in [1 on page 9-0   ] and add white noise. Set the random number generator to the default settings to produce repeatable results. Subtract its mean value and plot the result.

```
rng default
fs = 500;
t = 0:1/fs:1-1/fs;
f1 = 1./(6/5+cos(2*pi*t));
f2 = 1./(3/2+sin(2*pi*t));
f3 = cos(32*pi*t+cos(64*pi*t));
sig = f1+f2.*f3;
sig = sig+randn(1,length(sig))/2;
sig = sig-mean(sig);
plot(t,sig)
xlabel('Time (sec)')
ylabel('Amplitude')
title('Test Signal')
```
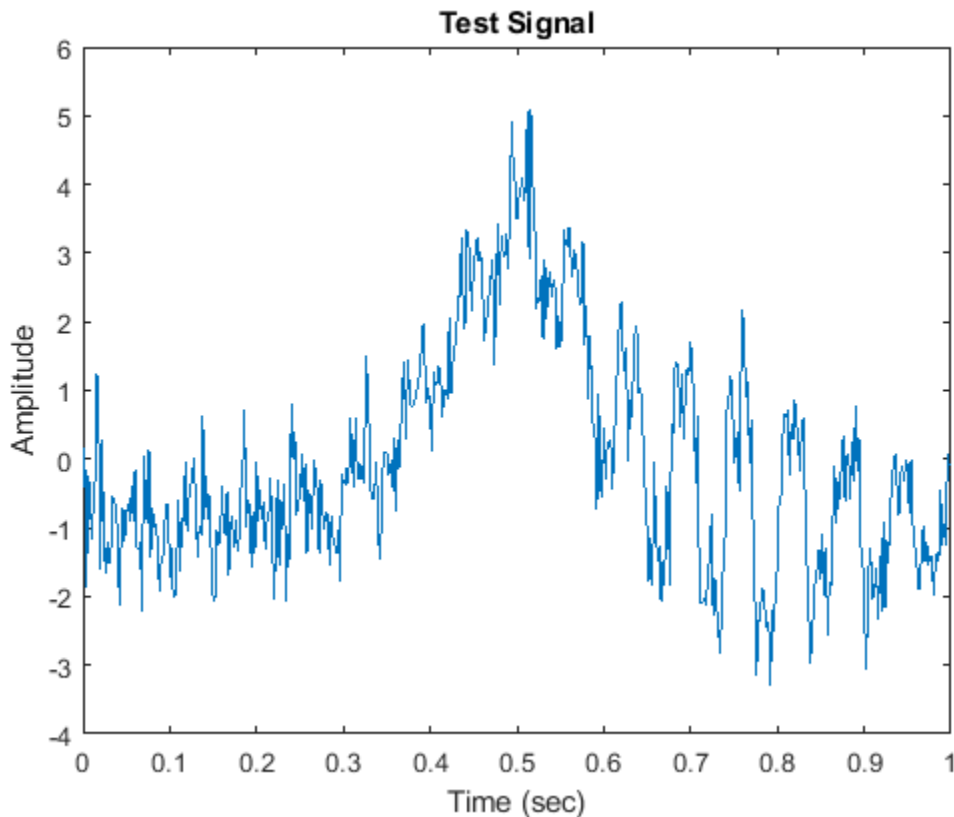

Use the `periodogram` function to plot the periodogram of the signal. Then use the `pmtm` function to plot the smoothed multitaper estimate.

```
[Pxx,F] = periodogram(sig,[],[],500);
Pxxmt = pmtm(sig,5,[],500,'Tapers','sine','power');
subplot(2,1,1)
plot(F,Pxx)
title('Periodogram')
subplot(2,1,2)
```

```
plot(F,Pxxmt)
title('Smoothed Estimate')
xlabel('Frequency (Hz)')
```
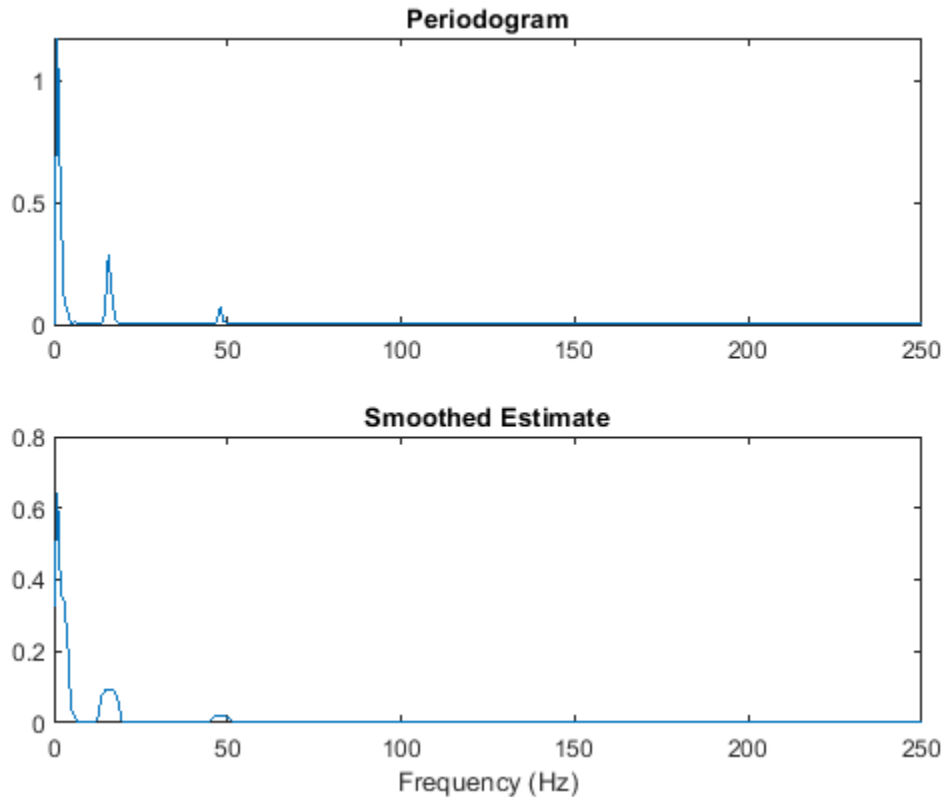

Computing EWT

You have several ways to control how the `ewt` function obtains the MRA of a signal. This section demonstrates a few options.

Use the `ewt` function with default settings to obtain the MRA of the signal and information about the filter bank.

```
[mra,~,~,info] = ewt(sig);
size(mra)
```

ans = *1×2*

    500   2

Specify Peaks

By default, `ewt` finds two MRA components. Inspect the filter bank passbands. Because the passbands are returned in normalized frequencies, multiply them by the sampling frequency.

```
info.FilterBank.Passbands*fs
```

ans = *2×2*

```
        65     250
         0      65
```

Note that there is a segment boundary at 22 Hz. The first segment has two peaks. Set `MaxNumPeaks` equal to 3 so that `ewt` determines the filter passbands using the three largest peaks.

```
[mra,cfs,~,info] = ewt(sig,'MaxNumPeaks',3);
info.FilterBank.Passbands*fs
```

ans = *3×2*

```
   62.0000   250.0000
   28.0000    62.0000
        0    28.0000
```

Verify that summing the MRA components results in perfect reconstruction of the signal, and verify that the EWT analysis coefficients are energy preserving.

```
max(abs(sig'-sum(mra,2)))
```

ans = 1.7764e-15

```
sum(sum(abs(cfs).^2))
```

ans = 1.2985e+03

```
norm(sig,2)^2
```

ans = 1.2985e+03

Instead of specifying a maximum number of peaks, you can set the percentage threshold used to determine which peaks are retained in the multitaper power spectrum. Local maxima in the multitaper power spectral estimate of the signal are normalized to lie in the range [0,1] with the maximum peak equal to 1. Set `PeakThresholdPercent` to 2.

```
[~,~,~,info] = ewt(sig,'PeakThresholdPercent',2);
info.FilterBank.Passbands*fs
```

ans = *5×2*

```
  141.0000   250.0000
   74.0000   141.0000
   57.0000    74.0000
   28.0000    57.0000
        0    28.0000
```

Specify Segmentation Method

By default, `ewt` uses the geometric mean of adjacent peaks to determine the filter passbands. The `ewt` function gives you the option to instead use the first local minimum between peaks. Set `SegmentMethod` to `'localmin'`, so that `ewt` uses the first local minimum, and specify a maximum of three peaks. Confirm that using the first local minimum results in a different segmentation.

```
[~,~,~,info] = ewt(sig,'MaxNumPeaks',3,'SegmentMethod','localmin');
info.FilterBank.Passbands*fs
```

```
ans = 3×2

   54.0000   250.0000
   28.0000    54.0000
        0     28.0000
```

Specify Frequency Resolution

You can also specify the frequency resolution bandwidth of the multitaper power spectral estimate. The frequency resolution bandwidth determines how many sine tapers are used in the multitaper power spectrum estimate. Specify a frequency resolution of 0.2 and a maximum of three peaks. Note that even though `MaxNumPeaks` is set at 3, three peaks are not found using the specified frequency resolution.

```
[mra,~,~,info] = ewt(sig,'MaxNumPeaks',3,'FrequencyResolution',0.2);
info.FilterBank.Passbands*fs
```

```
ans = 2×2

   83.0000   250.0000
        0     83.0000
```

References

[1] Gilles, Jérôme. "Empirical Wavelet Transform." *IEEE Transactions on Signal Processing* 61, no. 16 (August 2013): 3999–4010. https://doi.org/10.1109/TSP.2013.2265222.

[2] Gilles, Jérôme, Giang Tran, and Stanley Osher. "2D Empirical Transforms. Wavelets, Ridgelets, and Curvelets Revisited." *SIAM Journal on Imaging Sciences* 7, no. 1 (January 2014): 157–86. https://doi.org/10.1137/130923774.

[3] Gilles, Jérôme, and Kathryn Heal. "A Parameterless Scale-Space Approach to Find Meaningful Modes in Histograms — Application to Image and Spectrum Segmentation." *International Journal of Wavelets, Multiresolution and Information Processing* 12, no. 06 (November 2014): 1450044. https://doi.org/10.1142/S0219691314500441.

See Also

Functions
emd | ewt | modwt | modwtmra

Apps
Signal Multiresolution Analyzer

Featured Examples — Time-Frequency Analysis

Practical Introduction to Continuous Wavelet Analysis

This example shows how to perform and interpret continuous wavelet analysis. The example helps you answer common questions such as: What is the difference between continuous and discrete wavelet analysis? Why are the frequencies, or scales in continuous wavelet analysis logarithmically spaced? In what types of signal analysis problems are continuous wavelet techniques particularly useful?

Continuous versus Discrete Wavelet Analysis

One question that people often have is: What is *continuous* about continuous wavelet analysis? After all, you are presumably interested in doing wavelet analysis in a computer and not with pencil and paper, and in a computer, nothing is really continuous. When the term *continuous wavelet analysis* is used in a scientific computing setting, it means a wavelet analysis technique with more than one wavelet per octave, or doubling of frequency, and where the shift between wavelets in time is one sample. This provides the resulting continuous wavelet transform (CWT) has two properties that are very useful in applications:

- The frequency content of a signal is more finely captured than with discrete wavelet techniques.
- The CWT has the same time resolution as the original data in each frequency band.

Additionally, in most applications of the CWT, it is useful to have complex-valued wavelets as opposed to real-valued wavelets. The main reason for this is that complex-valued wavelets contain phase information. See Compare Time-Varying Frequency Content in Two Signals on page 10-0 for an example where phase information is useful. The examples in this tutorial use complex-valued wavelets exclusively.

See [1] for a detailed treatment of wavelet signal processing including continuous wavelet analysis with complex-valued wavelets.

Filters or Voices Per Octave

A term commonly used to designate the number of wavelet filters per octave is *voices per octave*. To illustrate this, construct a default continuous wavelet filter bank.

```
fb = cwtfilterbank
```

```
fb =
  cwtfilterbank with properties:

      VoicesPerOctave: 10
              Wavelet: 'Morse'
    SamplingFrequency: 1
        SamplingPeriod: []
          PeriodLimits: []
          SignalLength: 1024
       FrequencyLimits: []
         TimeBandwidth: 60
     WaveletParameters: []
              Boundary: 'reflection'
```

If you examine the properties of the filter bank, by default there are 10 wavelet filters per octave (or `VoicesPerOctave`). Plot the frequency responses of the wavelet filters.

```
freqz(fb)
```

CWT Filter Bank
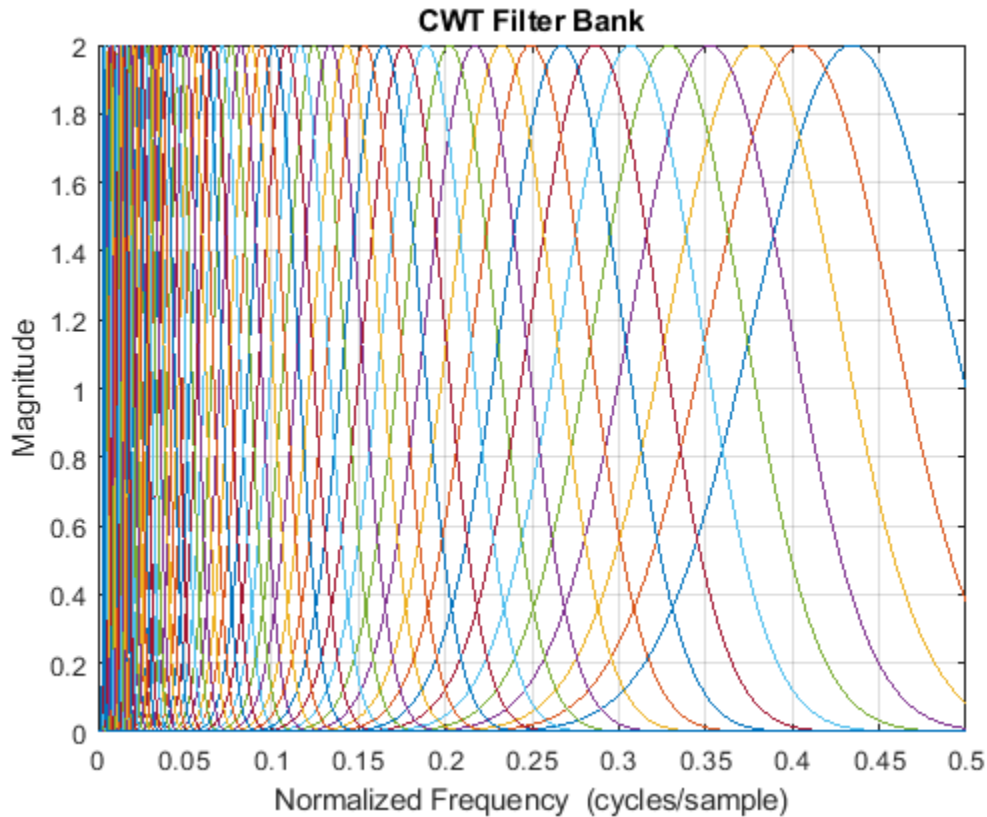
The wavelet filters in a continuous analysis share the important *constant-Q* property with all wavelet filters, namely that their spread in frequency, or bandwidth, is proportional to their center frequency. In other words, wavelet filters are broader at higher frequencies than they are at lower frequencies. Because of the reciprocal relationship between time and frequency, this means that higher frequency wavelets are better localized in time (have smaller time support) than lower frequency wavelets. To see this, extract the center frequencies and time-domain wavelets from the filter bank. Plot the real parts of two wavelets, one high-frequency and one low frequency, for comparison.

```
psi = wavelets(fb);
F = centerFrequencies(fb);
figure
plot(real(psi(9,:)))
yyaxis right
plot(real(psi(end-16,:)))
axis tight
S1 = 'Wavelets With Center Frequencies';
S2 = [num2str(F(9),'%1.2f') ' and ' num2str(F(end-16),'%1.2f') ' cycles/sample'];
title({S1 ; S2})
legend('0.25 c/s Wavelet','0.01 c/s Wavelet')
```

Wavelets With Center Frequencies 0.25 and 0.01 cycles/sample
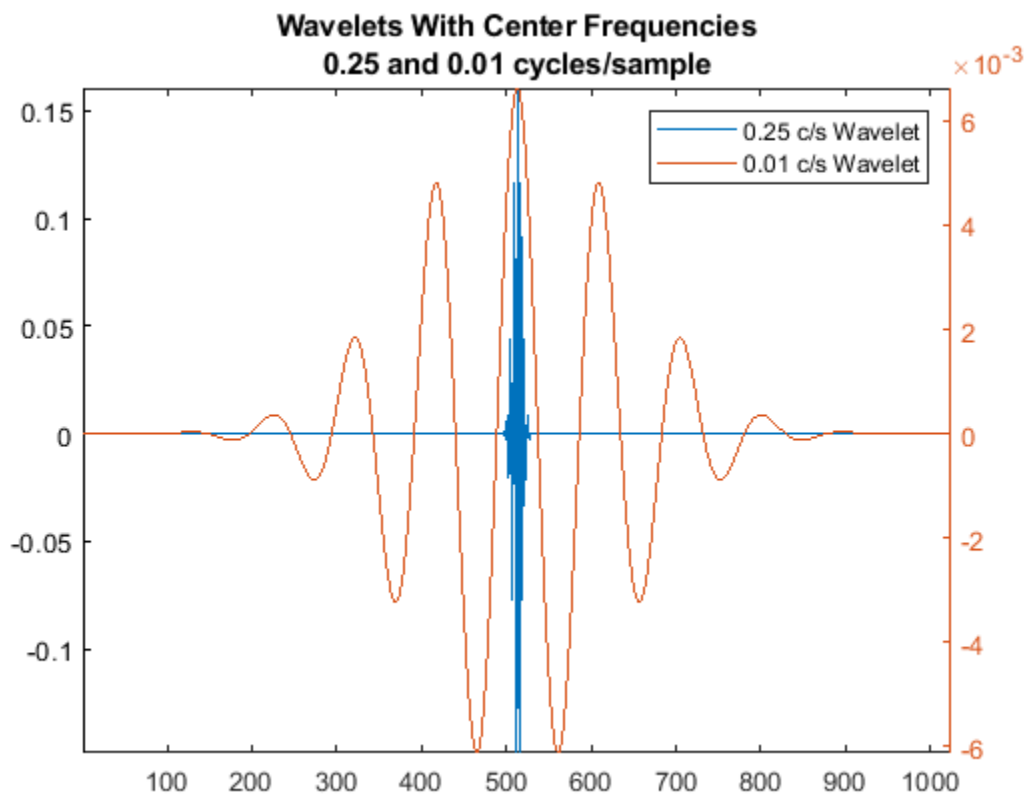
In the frequency responses plot, the center frequencies do not exceed 0.45 cycles/sample. Find how many wavelet filters the filter bank has in one octave, or doubling of frequency. Confirm the number is equal to `VoicesPerOctave`.

```
numFilters10 = nnz(F >= 0.2 & F<= 0.4)
```

```
numFilters10 = 10
```

You can specify a different number of filters when you create the filter bank. Create a filter bank with 8 voices per octave and plot the frequency responses.

```
fb = cwtfilterbank('VoicesPerOctave',8);
figure
freqz(fb)
```
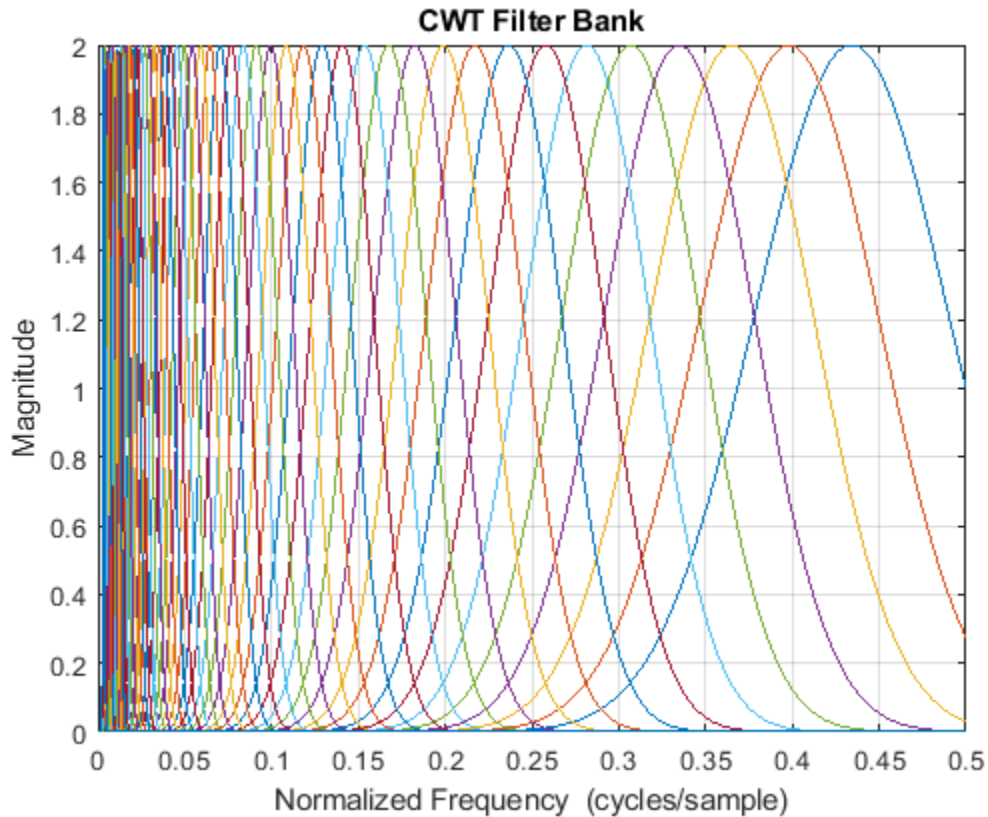
CWT Filter Bank

Confirm that there are eight filters per octave in the filter bank.

```
F = centerFrequencies(fb);
numFilters8 = nnz(F >= 0.2 & F<= 0.4)
```

```
numFilters8 = 8
```

You can read a more detailed explanation of the differences between continuous and discrete wavelet analysis at "Continuous and Discrete Wavelet Transforms".

Logarithmically Spaced Center Frequencies

One aspect of wavelet analysis that people can find a bit confusing is the logarithmic spacing of the filters.

Plot the center frequencies for the wavelet filter bank.

```
subplot(2,1,1)
plot(F)
ylabel('Cycles/Sample')
title('Wavelet Center Frequencies')
grid on
subplot(2,1,2)
semilogy(F)
grid on
ylabel('Cycles/Sample')
title('Semi-Log Scale')
```
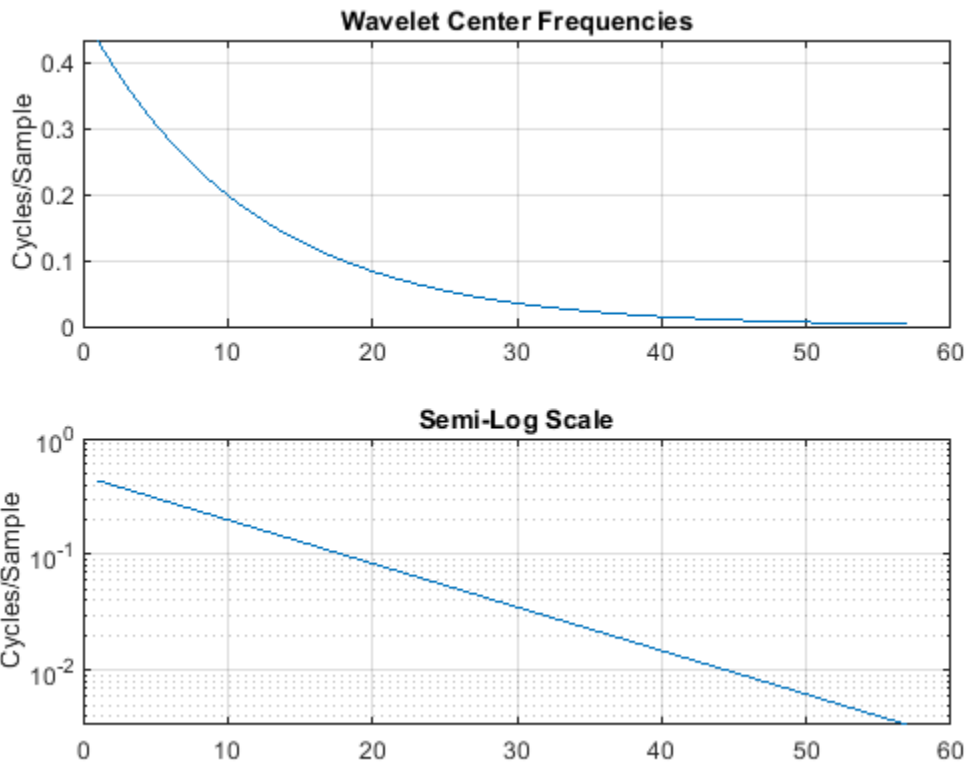
The plots show that the wavelet center frequencies are not linearly spaced as is commonly the case with other filter banks. Specifically, the center frequencies are exponentially decreasing, so that the step size between higher center frequencies is larger than the step size between lower center frequencies. Why does this make sense for wavelets? Recall that wavelets are constant-Q filters, which means their bandwidths are proportional to their center frequencies. If you have a filter bank where every filter has the same bandwidth, like in the short-time Fourier transform, or spectrogram, filter bank, then to maintain a constant frequency overlap between filters you need a constant step size. However, with wavelets, the step size should be proportional to frequency like the bandwidth. For continuous wavelet analysis, the most common spacing is the base $2^{(1/NV)}$, where NV is the number of filters per octave, raised to integer powers. For comparison, the spacing used exclusively in discrete wavelet analysis is the base 2 raised to integer powers.

See [2] for a thorough treatment of discrete wavelet analysis. The following table summarizes the main similarities and differences between discrete and continuous wavelet techniques. For discrete techniques, the names of representative algorithms in MATLAB are provided in parentheses.

Method	Wavelet Filters Per Octave	Logarithmically Spaced Center Frequencies	Base	Time Shift
Continuous	> 1 Usually 8-16 in practice	✓	2^(1/NV) NV = Number of filters per octave	1 sample
Discrete - Critically Downsampled (DWT,WAVEDEC)	1	✓	2	2^L samples L = level of DWT Shift is proportional to scale
Maximally Redundant Discrete (MODWT)	1	✓	2	1 sample

Time-Frequency Analysis

Because wavelets are simultaneously localized in time and frequency, they are useful for a number of applications. For continuous wavelet analysis, the most common application area is time-frequency analysis. Furthermore, the following properties of the CWT make it particularly useful for certain classes of signals.

- The constant-Q property of the wavelet filters, which means higher frequency wavelets are shorter in duration and lower frequency wavelets are longer in duration.
- The one-sample time shift between wavelet filters in continuous analysis

To understand which classes of signals are good candidates for continuous wavelet analysis, consider a hyperbolic chirp signal.

```
load hyperbolicChirp
figure
plot(t,hyperbolchirp)
axis tight, xlabel('Seconds')
ylabel('Amplitude')
title('Hyperbolic Chirp')
```
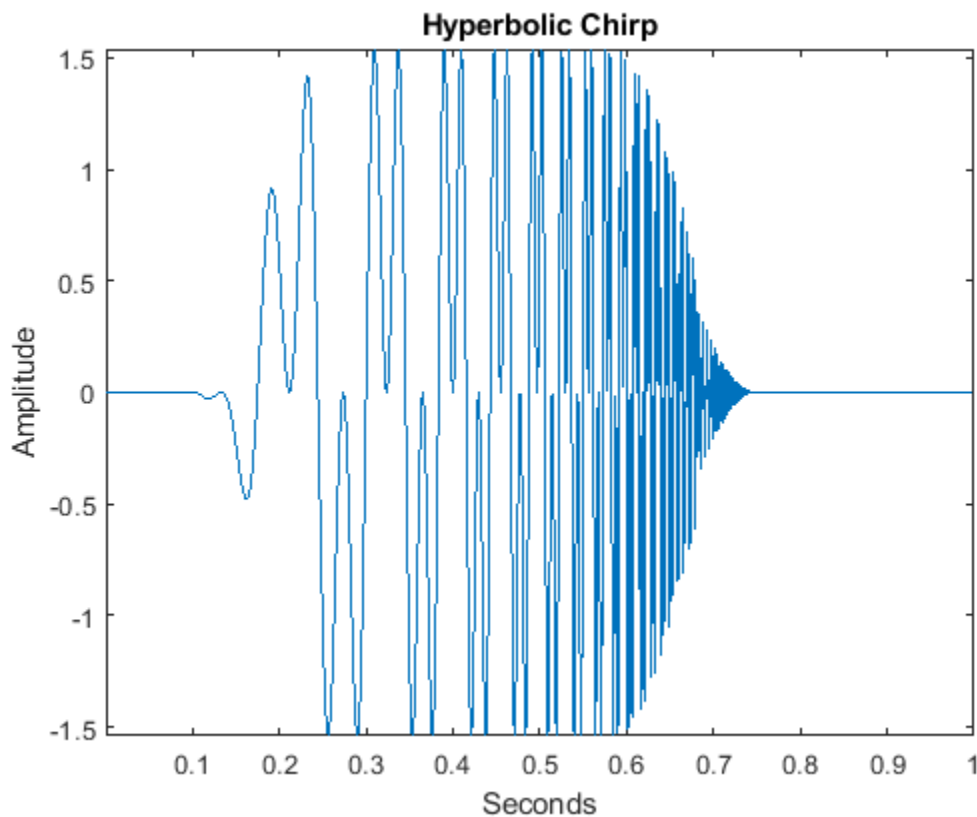
Hyperbolic Chirp

The hyperbolic chirp is the sum of $\sin\left(\frac{15\pi}{0.8-t}\right)$ and $\sin\left(\frac{5\pi}{0.8-t}\right)$. The first component is active from 0.1 to 0.68 seconds, and the second component from 0.1 to 0.75 seconds. The frequency at each instant in time, or the instantaneous frequencies, of these components are $\frac{7.5}{(0.8-t)^2}$ and $\frac{2.5}{(0.8-t)^2}$ cycles/second respectively. This means that the instantaneous frequencies are very low near $t = 0$ and increase rapidly as the time approaches 0.8 seconds.

Obtain the CWT of the chirp signal and plot the CWT along with the true instantaneous frequencies plotted as dashed lines.

```
Fs = 1/mean(diff(t));
[cfs,f] = cwt(hyperbolchirp,Fs);
helperHyperbolicChirpPlot(cfs,f,t)
```

Magnitude Scalogram
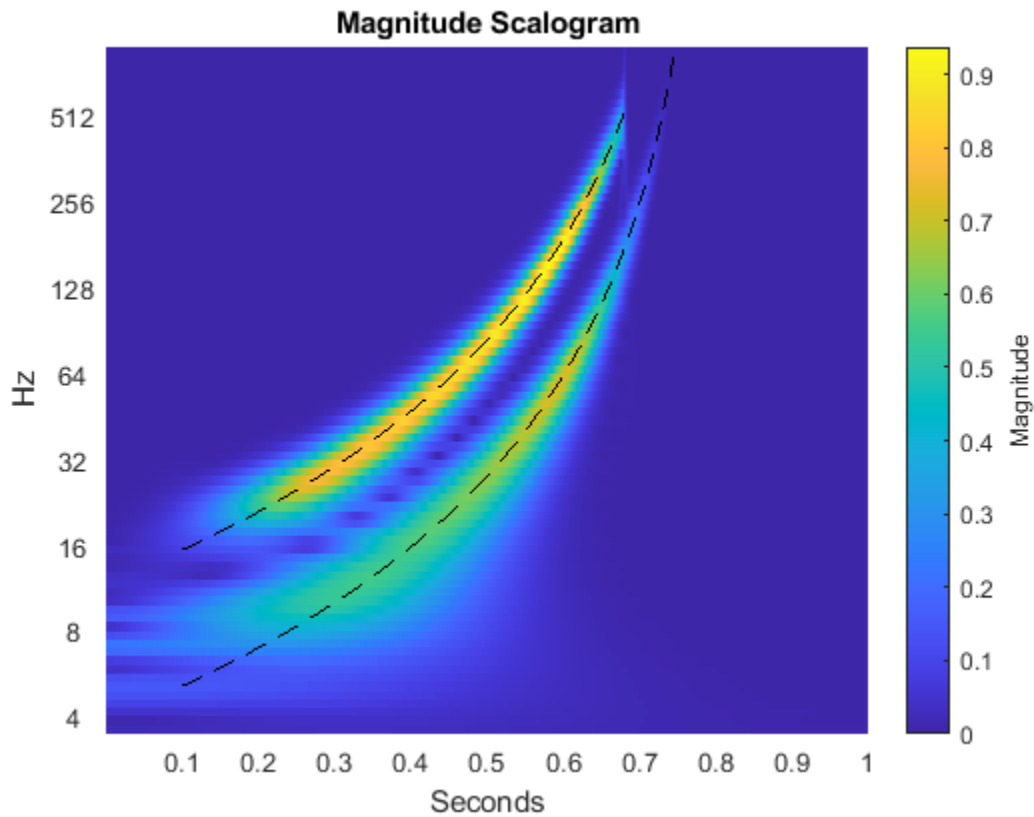
We see that the CWT shows a time-frequency representation that accurately captures the instantaneous frequencies of the hyperbolic chirp. Now analyze the same signal with the short-time Fourier transform which uses constant bandwidth filters. Use the default window size and overlap.

```
pspectrum(hyperbolchirp,2048,'spectrogram')
```

Fres = 41.0679 Hz, Tres = 62.5 ms
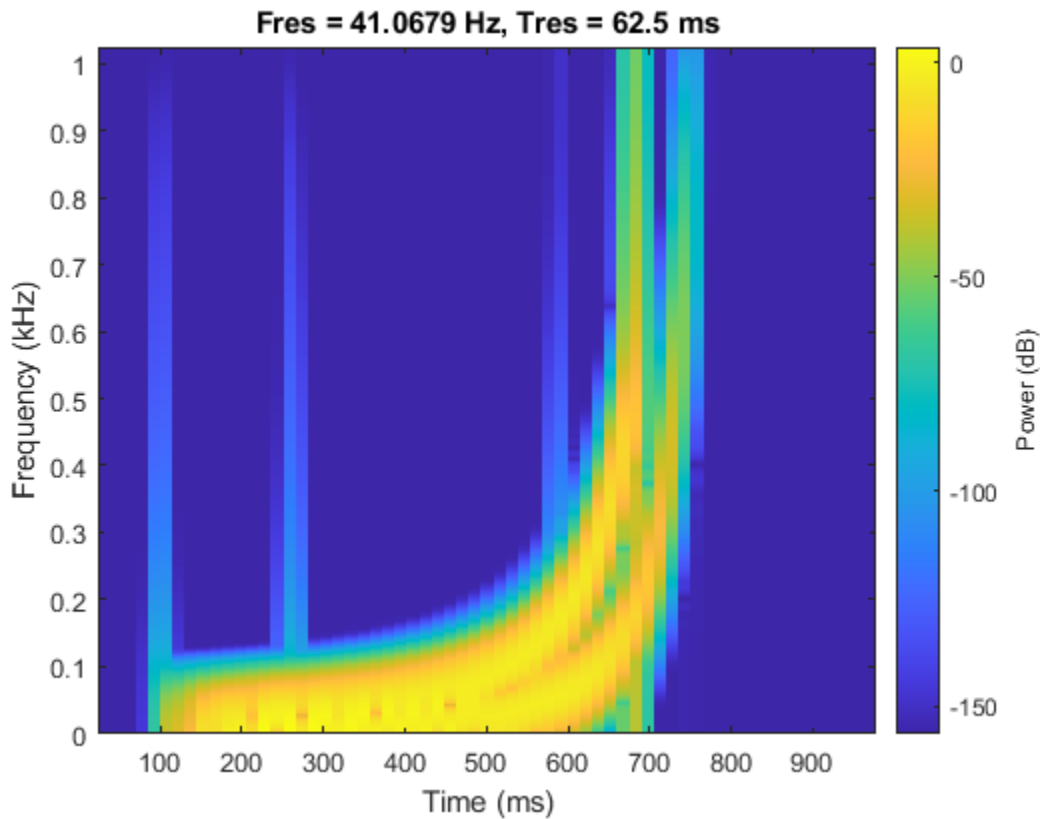
Note that the spectrogram is unable to distinguish the two different instantaneous frequencies when the frequencies are low, or equivalently when the two instantaneous frequencies are close together. Based on the input length of 2048, the spectrogram is computed by default using a window length of 128 samples. This is equivalent to a time resolution of 62.5 msec given the sampling rate of 2048 Hz. Based on the default Kaiser window, this results in a frequency resolution of 41 Hz. This is too large to differentiate the instantaneous frequencies near t=0. What happens if we reduce the frequency resolution by approximately 1/2 in the hopes of better resolving the lower frequencies?

```
pspectrum(hyperbolchirp,2048,'spectrogram','FrequencyResolution',20, ...
    'OverlapPercent',90)
```

Fres = 20.0637 Hz, Tres = 127.9297 ms
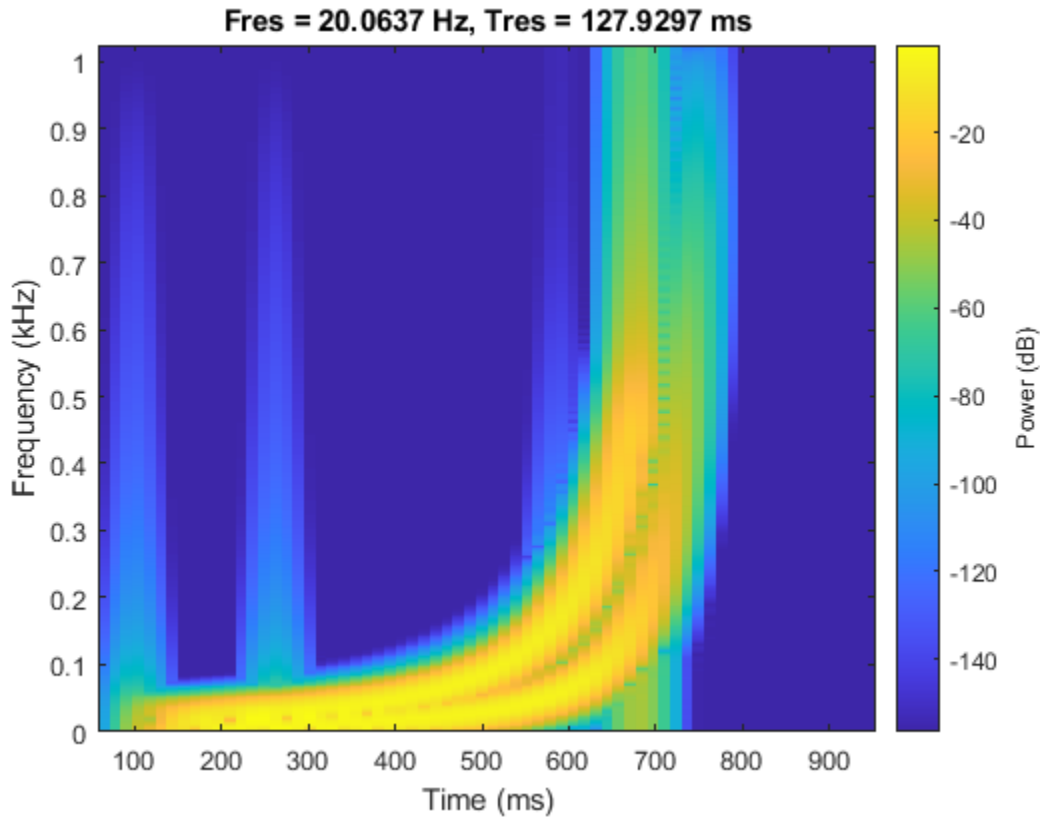
The result is no better because any frequency separation we have gained at the lower frequencies results in the higher instantaneous frequencies smearing together in time. The spectrogram is a powerful time-frequency analysis technique, in fact arguably optimal in many applications, but like all techniques it has limitations. This particular type of signal presents a major challenge for fixed bandwidth filters. Ideally you want a long time response with narrow frequency support at low frequencies and a short time support with broader frequency support at high frequencies. The wavelet transform provides exactly that and is therefore quite successful in this case. An equivalent way to state this is that the wavelet transform has better time resolution at higher frequencies and better frequency resolution at lower frequencies.

In real-world signals, high-frequency events are often of short duration while lower frequency events are longer in duration. As an example, load and plot a human electrocardiogram (ECG) signal. The data is sampled at 180 Hz.

```
load wecg
tm = 0:1/180:numel(wecg)*1/180-1/180;
plot(tm,wecg)
grid on
axis tight
title('Human ECG')
xlabel('Seconds')
```

Human ECG
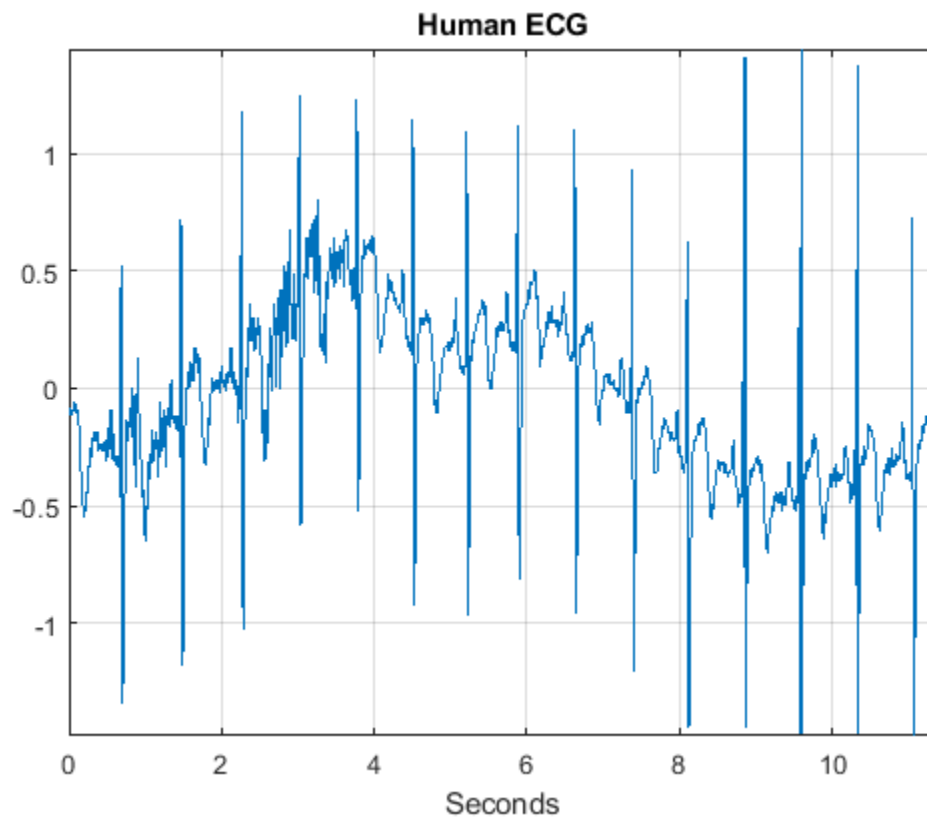
The data consists of slowly varying components punctuated by high frequency transients, which represent the electrical impulse as it spreads through the heart. If we are interested in localizing these impulsive events accurately in time, we want the analysis window (filter) to be short. We are willing to sacrifice frequency resolution in this case for more accurate time localization. On the other hand, to identify the lower frequency oscillations, it is preferred to have a longer analysis window.

```
[cfs,f] = cwt(wecg,180);
imagesc(tm,f,abs(cfs));
xlabel('Seconds'); ylabel('Hz');
axis xy;
caxis([0.025 0.25])
title('CWT of ECG Data')
```

CWT of ECG Data
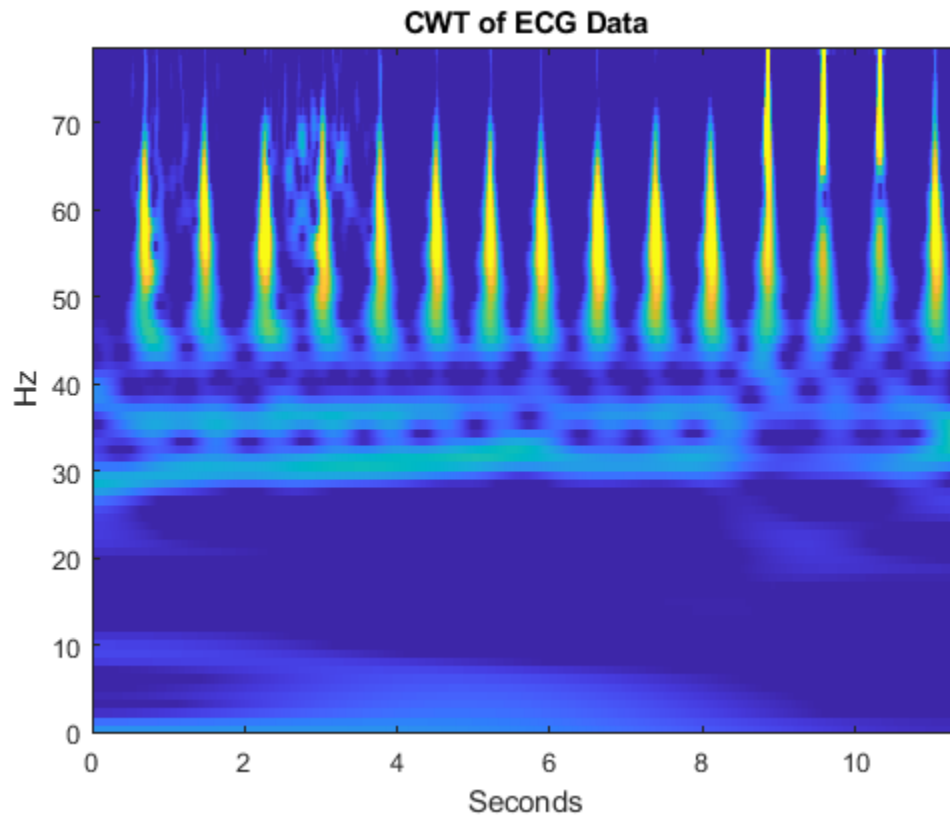
The CWT shows nearly steady-state oscillations near 30 Hz and 37 Hz as well as the transient events denoting the heartbeats (QRS complexes). With this analysis, you are able to analyze both phenomena simultaneously in the same time-frequency representation.

As another example of high frequency transients punctuating slowly varying components, consider a time series of solar magnetic field magnitudes recorded hourly over the south pole of the sun by the Ulysses spacecraft from 21:00 UT on December 4, 1993 to 12:00 UT on May 24, 1994. See [2] pp. 218–220 for a complete description of this data.

```
load solarMFmagnitudes
plot(sm_dates,sm)
axis tight
title('Solar Magnetic Field Magnitudes');
ylabel('Magnitude')
```

Solar Magnetic Field Magnitudes
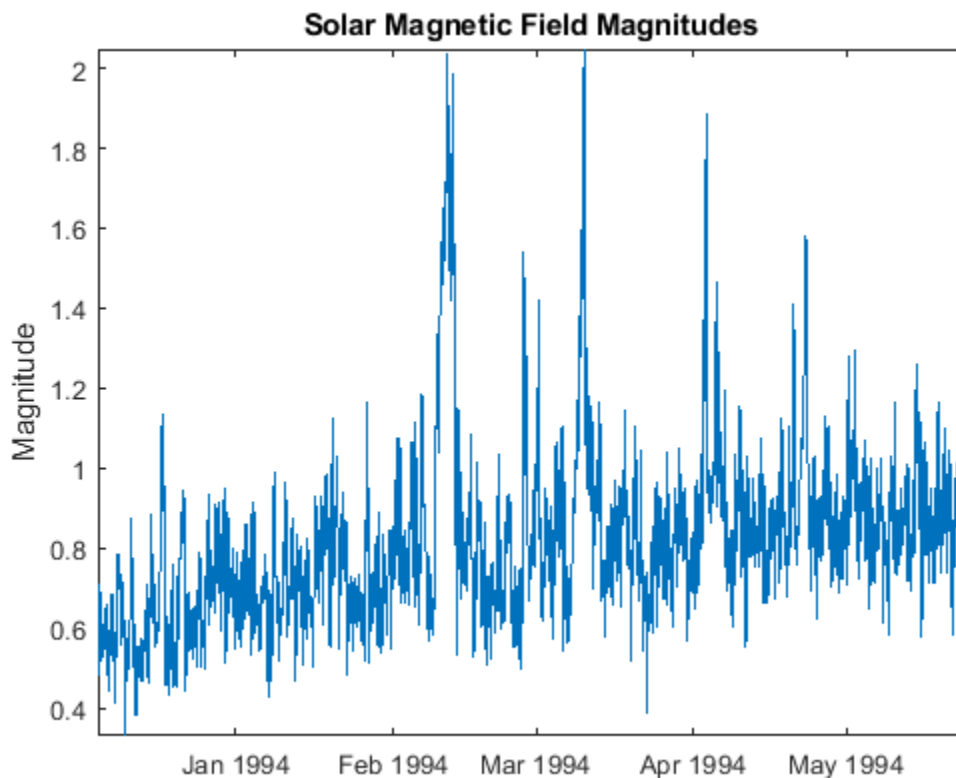
The raw time data shows an overall roughly linear trend with several impulsive events, or shock waves. Shock waves are produced when a fast solar wind or fast coronal mass ejection overtakes a slow solar wind. In the time series, we see significant shock wave structures occurring approximately on the following dates: February 11, February 26, March 10, April 3, and April 23.

Perform a continuous wavelet analysis of the data and plot the CWT magnitudes with the time series data on the same plot.

```
helperSolarMFDataPlot(sm,sm_dates)
```
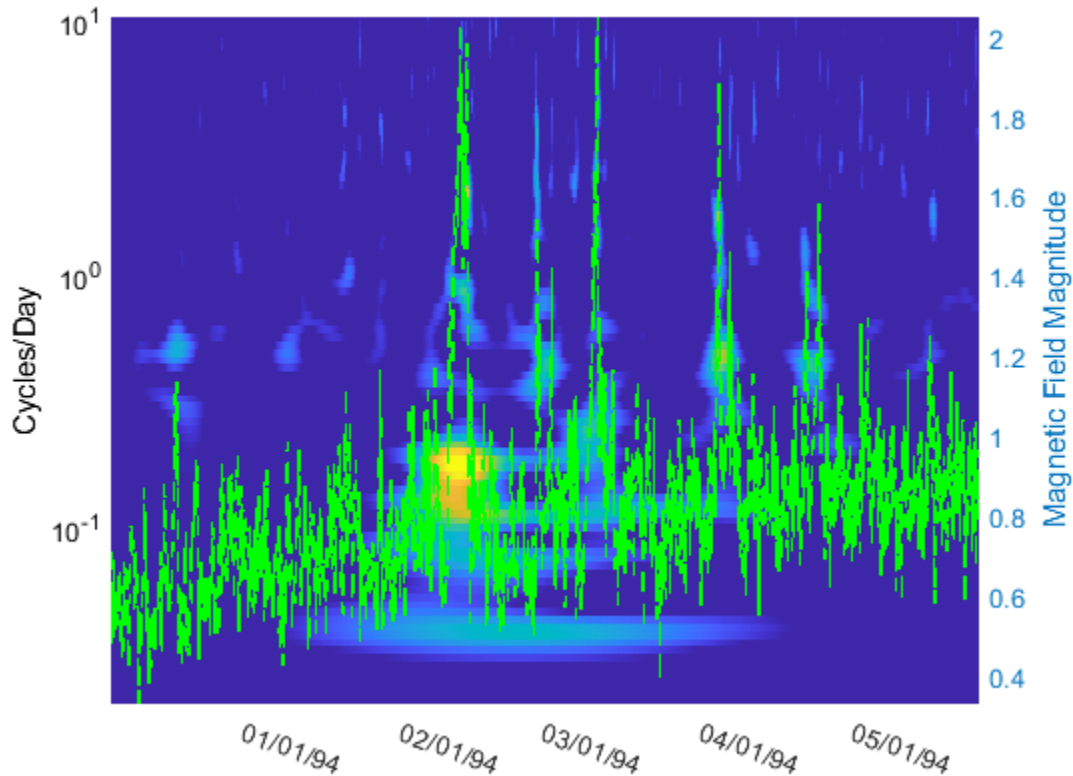
The CWT captures the impulsive events at the same times they occur in the time series. However, the CWT also reveals lower frequency features of the data hidden in the time series. For example, starting before and extending beyond the shock wave structure on February 11, 1994, there is a low-frequency steady state event (near 0.04 cycles/day). This results from a coronal mass ejection (CME) event likely occurring in a different solar region which reached the Ulysses spacecraft at the same time as the closer shock wave.

With the ECG and solar magnetic field magnitude examples, we have two time series from very disparate mechanisms generating similar data: signals with both short-duration transients and longer duration low frequency oscillations. Characterizing both in the same time-frequency representation is advantageous.

Localize Transients

Transient events in time series data are often one of the most informative events. Transient events can indicate an abrupt change in the data-generating mechanism, which you would like to detect and localize in time. Examples include faults in machinery, problems with sensors, financial "shocks" in economic time series, and many others. As an example, consider the following signal.

```
rng default;
dt = 0.001;
t = 0:dt:1.5-dt;
addNoise = 0.025*randn(size(t));
x = cos(2*pi*150*t).*(t>=0.1 & t<0.5)+sin(2*pi*200*t).*(t>0.7 & t <= 1.2);
x = x+addNoise;
x([222 800]) = x([222 800 ])+[-2 2];
```

```
figure;
plot(t.*1000,x);
xlabel('Milliseconds'); ylabel('Amplitude');
```
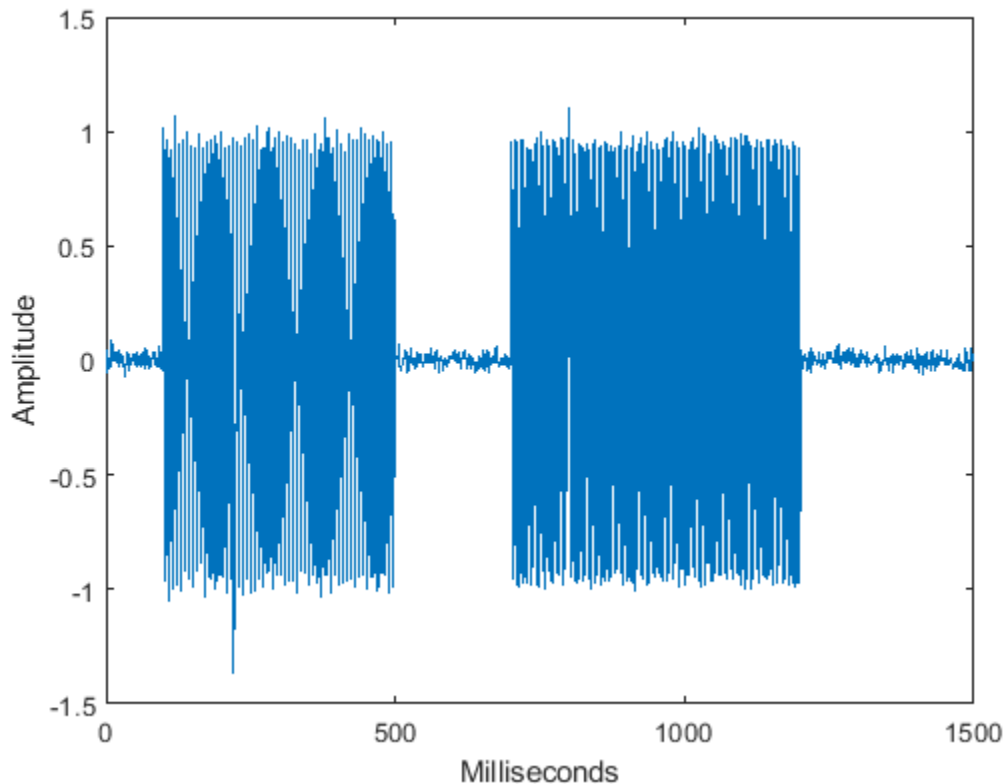

The signal consists of two sinusoidal components that turn on and off abruptly with additional "defects" at 222 milliseconds and 800 milliseconds. Obtain the CWT of the signal and plot only the finest-scale, or highest center frequency, wavelet coefficients. Create a second axis and plot the original time data.

```
cfs = cwt(x,1000,'amor');
plot(t,abs(cfs(1,:)),'linewidth',2)
ylabel('Magnitude')
set(gca,'XTick',[0.1 0.222 0.5 0.7 0.8 1.2])
yyaxis right
plot(t,x,'--')
ylabel('Amplitude')
xlabel('Seconds')
hold off
```
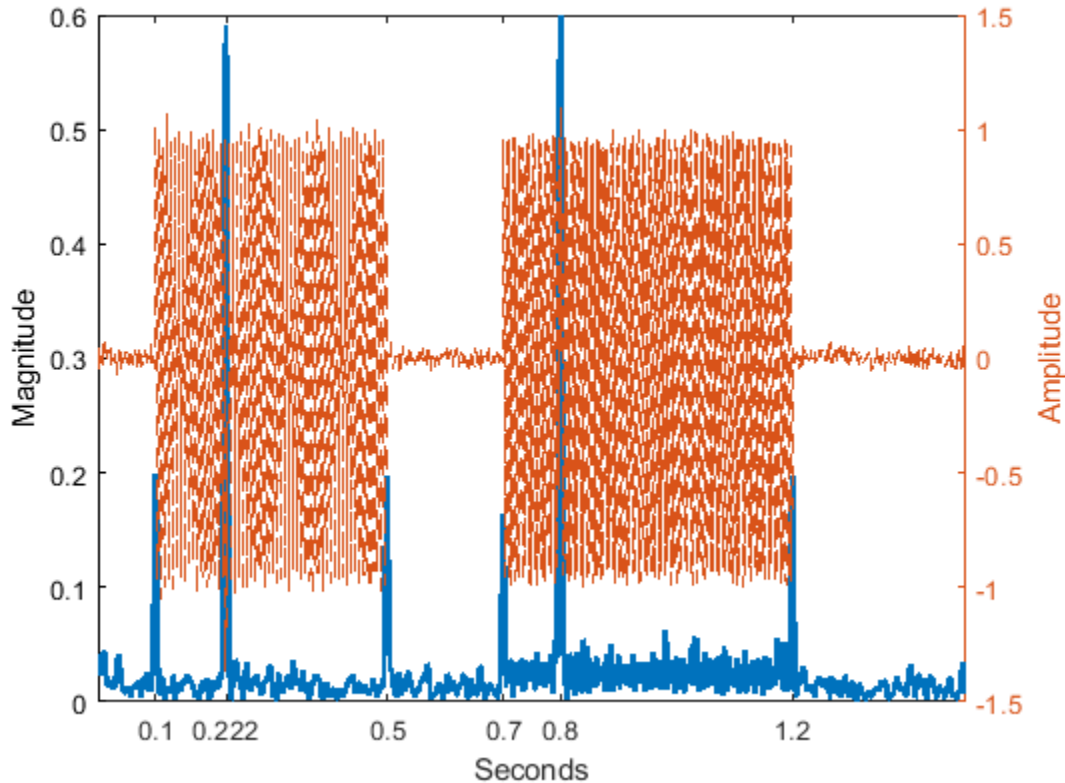
The finest-scale CWT coefficients localize all the abrupt changes in the data. The CWT coefficients show peaks where the sinusoidal components turn on and off as well as localizing the defects in the sinusoids at 222 milliseconds and 800 milliseconds. Because the CWT coefficients have the same time resolution as the data, it is often useful to use the CWT fine-scale magnitudes in order to detect transient changes in the data. To most accurately localize these transients, use the finest-scale (highest frequency) coefficients.

Sharpen Time-Frequency Analysis

Any time-frequency transform that uses filters, like wavelets in the case of the CWT, or modulated windows in the case of the short-time Fourier transform, necessarily smears the picture of the signal in time and frequency. The uncertainty in the localization of the signal's energy in time and frequency comes from the spread of the filters in time and frequency. Synchrosqueezing is a technique that attempts to compensate for this smearing by "squeezing" the transform along the frequency axis. To illustrate this with wavelets, obtain the CWT of a signal consisting of an amplitude and frequency modulated (AM-FM) signal and an 18-Hz sinusoid. The AM-FM signal is defined by the equation

$$(2 + \cos(4\pi t))\sin(2\pi 231 t + 90\sin(3\pi t))$$

```
load multicompsig
sig = sig1+sig2;
cwt(sig,sampfreq,'amor')
```

Magnitude Scalogram
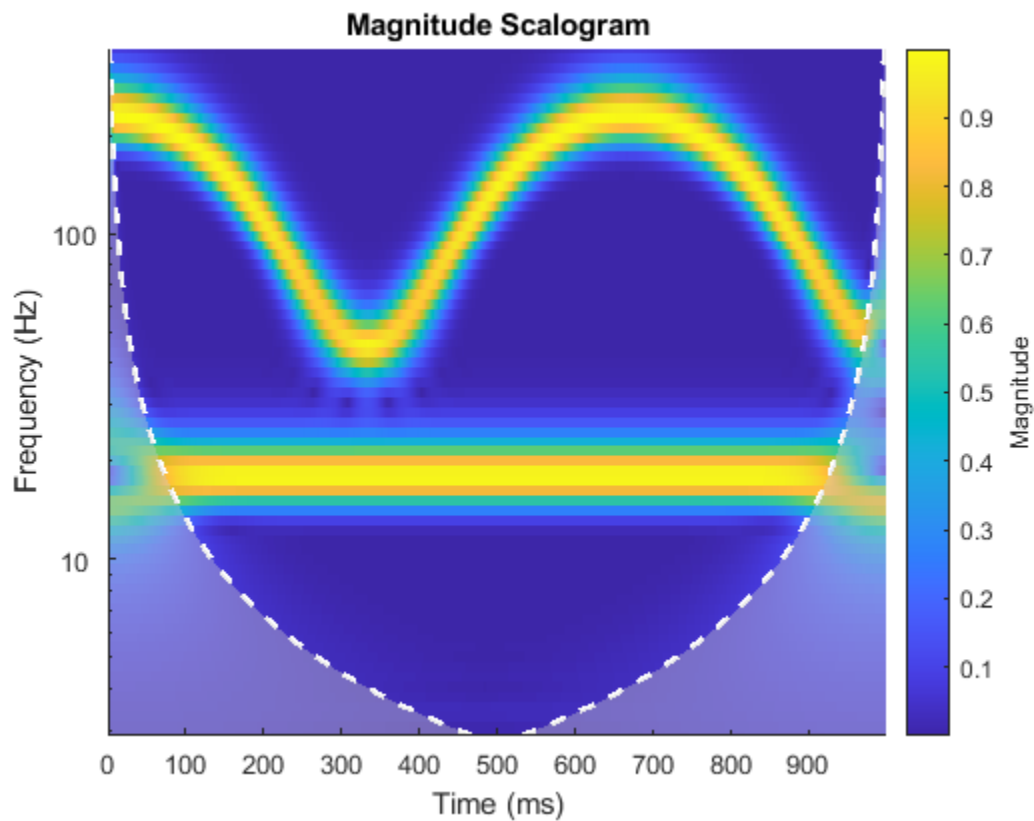
While the CWT shows both the 18-Hz sine wave as well as the AM-FM component, we see that the 18-Hz sine wave certainly appears spread out in frequency. Now use synchrosqueezing to sharpen the frequency estimates.

```
[sst,f] = wsst(sig,sampfreq);
figure
helperSSTPlot(sst,t,f)
```
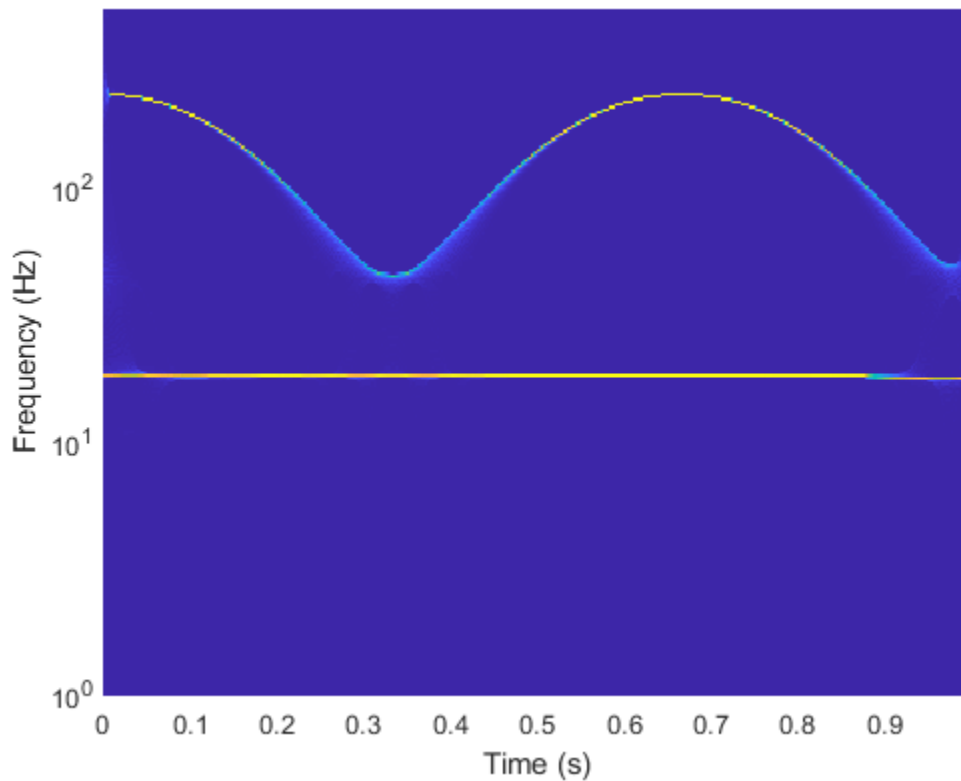
The synchrosqueezed transform has largely compensated for the spread in time frequency introduced by the wavelet filters. Synchrosqueezing has squeezed the relatively broad peaks in time-frequency into narrow *ridges*. You can actually extract these ridges and reconstruct the individual components from the time-frequency ridges.

```
[fridge,iridge] = wsstridge(sst,5,f,'NumRidges',2);
figure;
contour(t,f,abs(sst));
grid on;
title('Synchrosqueezed Transform with Ridges');
xlabel('Time (secs)'); ylabel('Hz');
hold on;
plot(t,fridge,'k--','linewidth',2);
hold off;
```
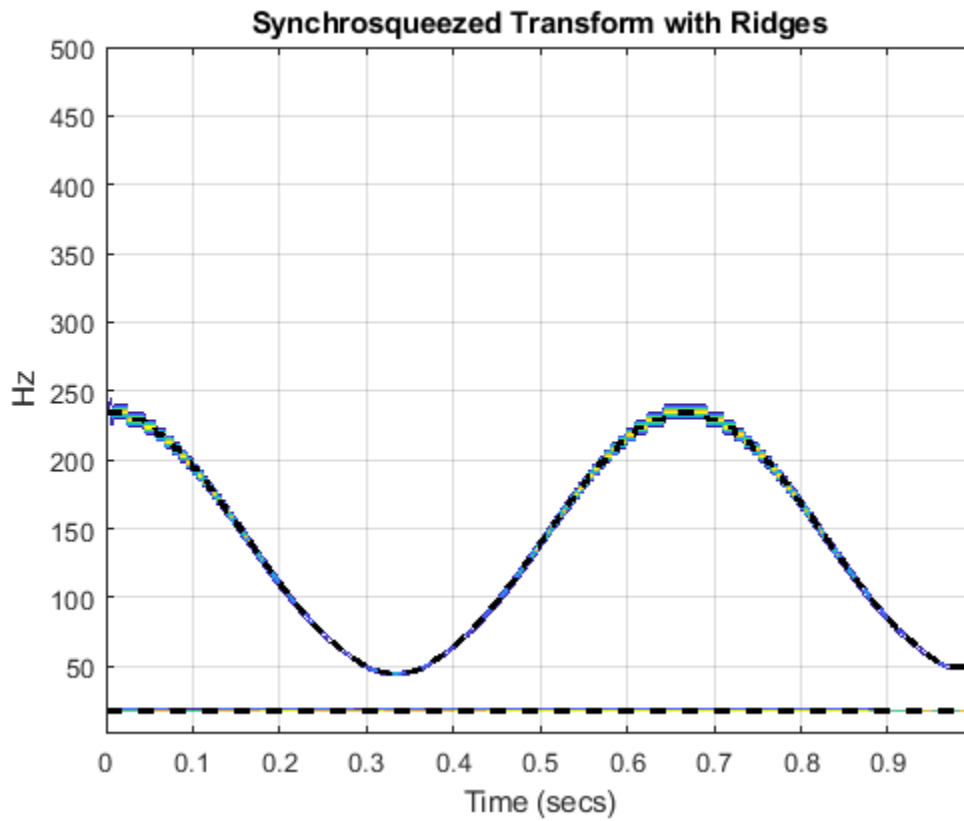
Finally reconstruct approximations to the components from the time-frequency ridges and compare the results to the original components.

```
xrec = iwsst(sst,iridge);
helperPlotComponents(xrec,sig1,sig2,t)
```
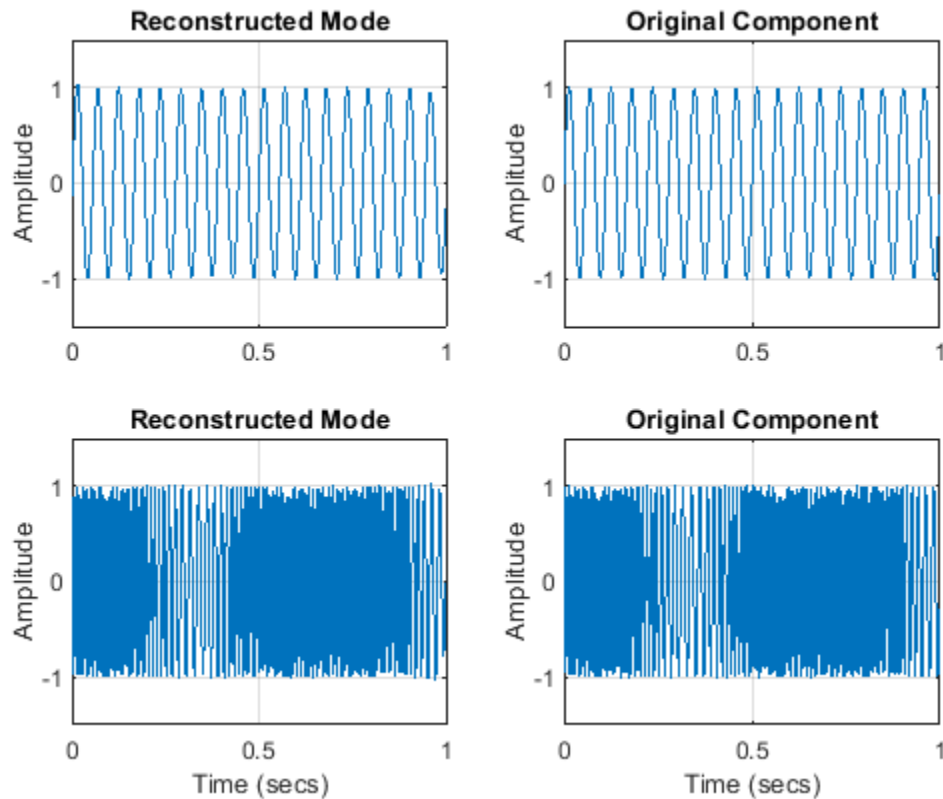
Compare Time-Varying Frequency Content in Two Signals

Often you have two signals which may be related in some way. One signal may determine behavior in another, or the signals may simply be correlated because of some influence extrinsic to both signals. If you obtain the CWT of two signals, you can use those transforms to obtain the *wavelet coherence,* a measure of time-varying correlation.

Wavelet coherence for each time instant and center frequency produces a coherence value between 0 and 1 which quantifies the strength of the correlation between the two signals. Because complex-valued wavelets are used, there is also phase information, which can be used to infer the lead-lag relationship between the signals. As an example, consider the following two signals.

```
t = 0:0.001:2;
X = cos(2*pi*10*t).*(t>=0.5 & t<1.1)+ ...
        cos(2*pi*50*t).*(t>= 0.2 & t< 1.4)+0.25*randn(size(t));
Y = sin(2*pi*10*t).*(t>=0.6 & t<1.2)+...
        sin(2*pi*50*t).*(t>= 0.4 & t<1.6)+ 0.35*randn(size(t));
figure
subplot(2,1,1)
plot(t,X)
ylabel('Amplitude')
subplot(2,1,2)
plot(t,Y)
ylabel('Amplitude')
xlabel('Time (s)')
```
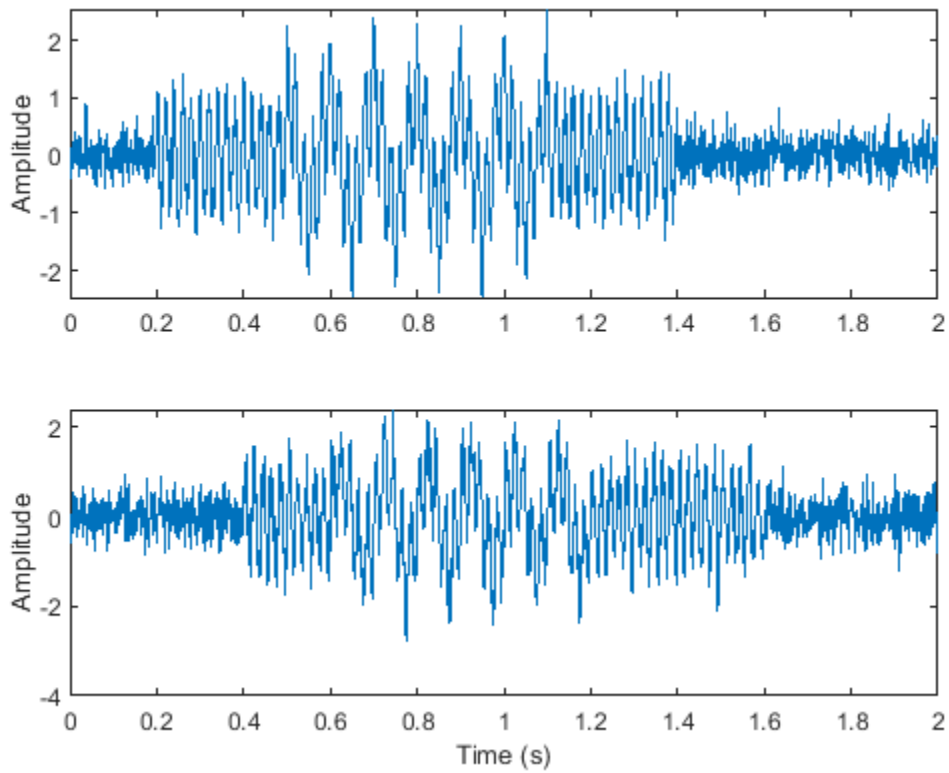
Both signals consist two sine waves (10 Hz and 50 Hz) in white noise. The sine waves have slightly different time supports. Note that the 10-Hz and 50-Hz components in Y are lagged by 1/4 cycle with respect to the corresponding components in X. Plot the wavelet coherence of the two signals.

```
figure
wcoherence(X,Y,1000,'PhaseDisplayThreshold',0.7)
```
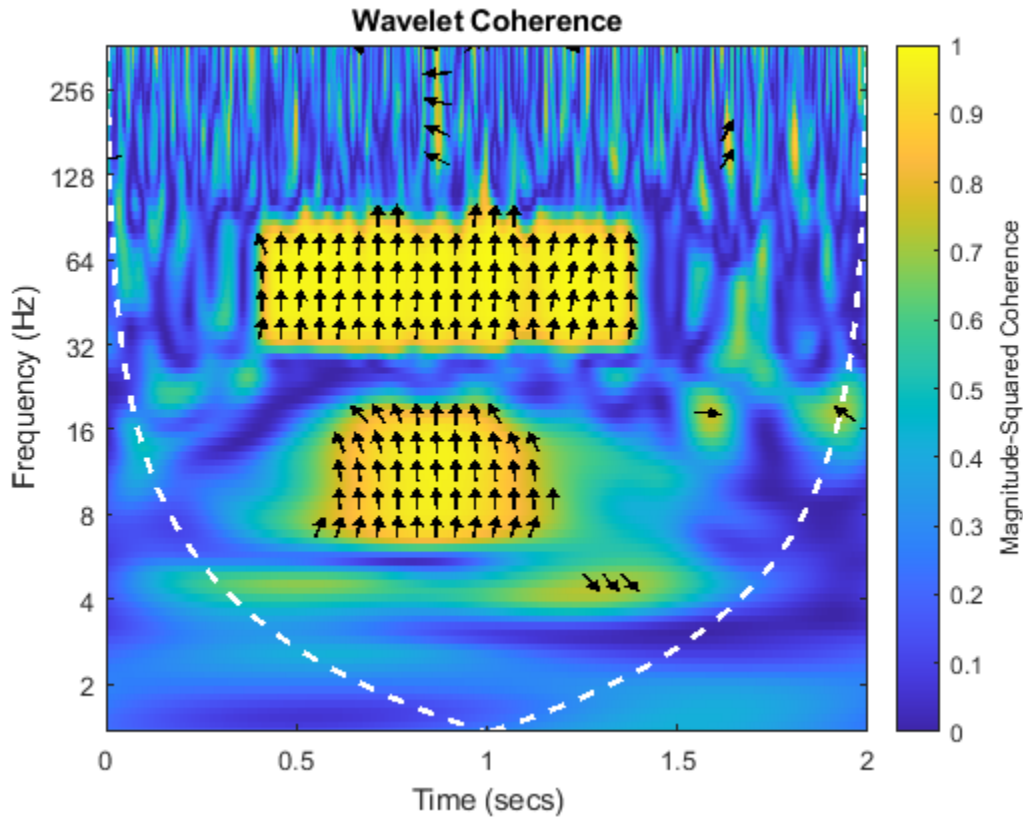
The wavelet coherence estimate captures that the signals are highly correlated near 50 and 10 Hz and only during the correct time intervals. The dark arrows on the plot show the phase relationship between the signals. Here the arrows point straight up which indicates a 90-degree phase relationship between the two signals. That is exactly the relationship dictated by the cosine-sine terms in the signals.

For a real-world example, consider the El Nino Region 3 data and deseasonalized All Indian Rainfall Index from 1871 to late 2003. The data are sampled monthly. The Nino 3 time series is a record of monthly sea surface temperature anomalies in degrees Celsius recorded from the equatorial Pacific at longitudes 90 degrees west to 150 degrees west, and latitudes 5 degrees north to 5 degrees south. The All Indian Rainfall Index represents average Indian rainfalls in millimeters with seasonal components removed.

```
load ninoairdata
figure
subplot(2,1,1)
plot(datayear,nino)
title('El Nino Region 3 -- SST Anomalies')
ylabel('Degrees Celsius')
axis tight
subplot(2,1,2)
plot(datayear,air)
axis tight
title('Deseasonalized All Indian Rainfall Index')
ylabel('Millimeters')
xlabel('Year')
```

10-23

Compute the wavelet coherence between the two time series. Set the phase display threshold to 0.7. To display the periods in years, specify the sampling interval as 1/12 of a year.

```
figure
wcoherence(nino,air,years(1/12),'PhaseDisplayThreshold',0.7);
```
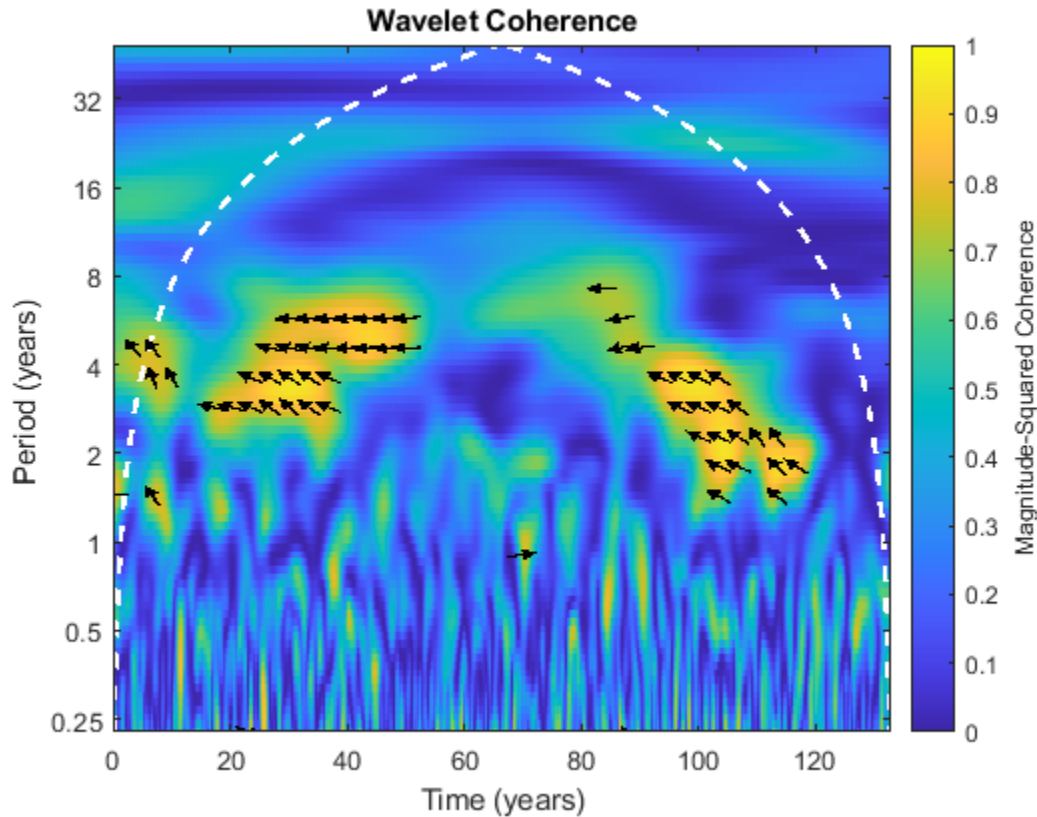
The plot shows time-localized areas of strong coherence occurring in periods that correspond to the typical El Nino cycles of 2 to 7 years. The plot also shows that there is an approximate 3/8-to-1/2 cycle delay between the two time series at those periods. This indicates that periods of sea warming consistent with El Nino recorded off the coast of South America are correlated with rainfall amounts in India approximately 17,000 km away, but that this effect is delayed by approximately 1/2 a cycle (1 to 3.5 years).

Conclusions

In this example you learned:

- What differentiates continuous from discrete wavelet analysis.
- How the constant-Q and one-sample time shift properties of the CWT allow you to analyze very different signal structures simultaneously.
- A time-frequency reassignment technique using the CWT.
- How you can use the CWT to compare frequency content in two signals.

See "Time-Frequency Analysis" for many more highlighted topics and featured examples on continuous wavelet analysis.

References

[1] Mallat, S. G. *A Wavelet Tour of Signal Processing: The Sparse Way*. 3rd ed. Amsterdam ; Boston: Elsevier/Academic Press, 2009.

[2] Percival, Donald B., and Andrew T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge ; New York: Cambridge University Press, 2000.

See Also

Functions
cwt | cwtfilterbank | pspectrum | wcoherence | wsst | wsstridge

Apps
Signal Analyzer

More About

- "Practical Introduction to Multiresolution Analysis" on page 11-2
- "Continuous and Discrete Wavelet Transforms"
- "Practical Introduction to Time-Frequency Analysis" (Signal Processing Toolbox)
- "Time-Frequency Gallery" on page 4-2
- "Wavelet Synchrosqueezing"

CWT-Based Time-Frequency Analysis

This example shows how to use the continuous wavelet transform (CWT) to analyze signals jointly in time and frequency. The example discusses the localization of transients where the CWT outperforms the short-time Fourier transform (STFT). The example also shows how to synthesize time-frequency localized signal approximations using the inverse CWT.

The CWT is compared with the STFT in a number of the examples. You must have Signal Processing Toolbox™ to run the `spectrogram` examples.

Time-Frequency Analysis of Modulated Signals

Load a quadratic chirp signal and show a plot of its spectrogram. 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;
[S,F,T] = spectrogram(quadchirp,100,98,128,fs);
helperCWTTimeFreqPlot(S,T,F,'surf','STFT of Quadratic Chirp','Seconds','Hz')
```
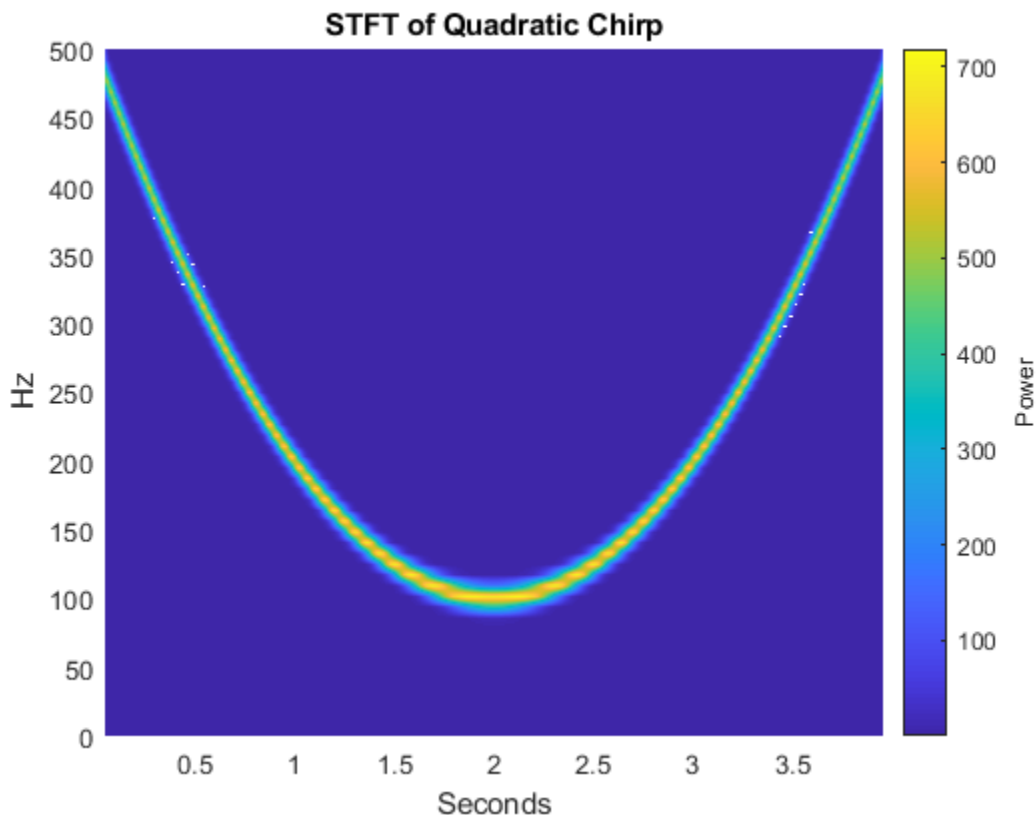

Obtain a time-frequency plot of this signal using the CWT.

```
[cfs,f] = cwt(quadchirp,fs,'WaveletParameters',[14,200]);
helperCWTTimeFreqPlot(cfs,tquad,f,'surf','CWT of Quadratic Chirp','Seconds','Hz')
```

10-27

CWT of Quadratic Chirp
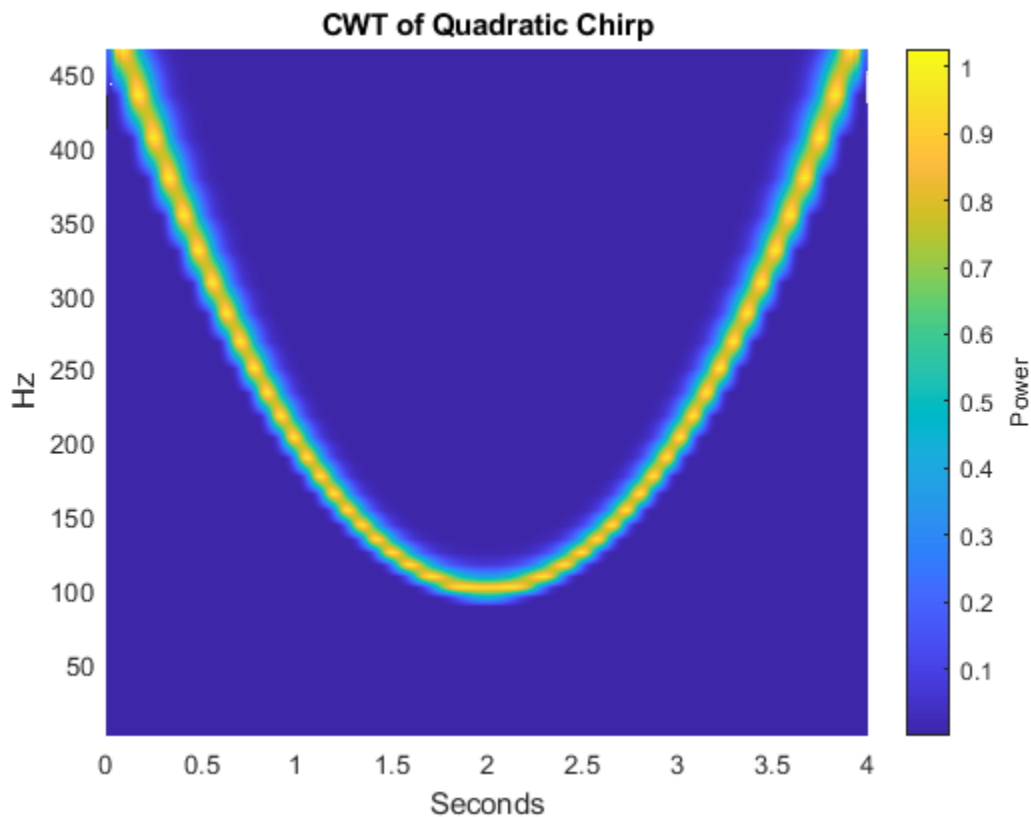
The CWT with the bump wavelet produces a time-frequency analysis very similar to the STFT.

Frequency and amplitude modulation occur frequently in natural signals. 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./1000,'surf','CWT of Bat Echolocation',...
    'Microseconds','kHz')
```
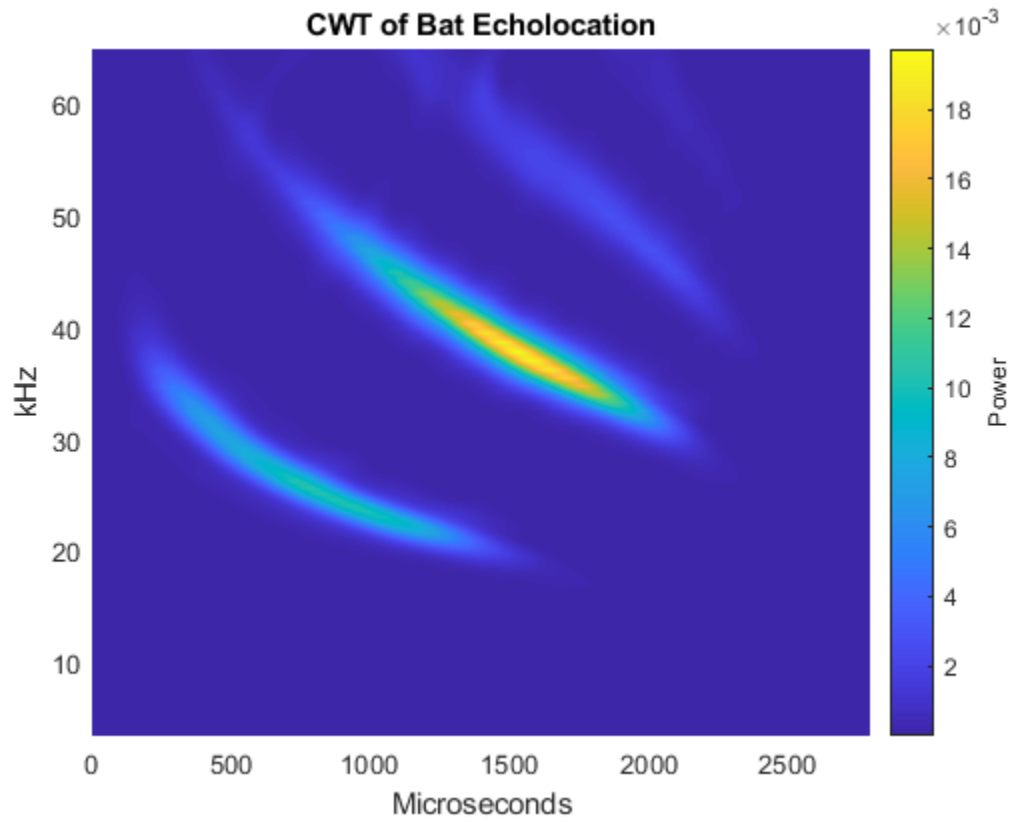
Obtain and plot the STFT of the bat data.

```
[S,F,T] = spectrogram(batsignal,50,48,128,1/DT);
helperCWTTimeFreqPlot(S,T.*1e6,F./1e3,'surf','Bat Echolocation (STFT)',...
    'Microseconds','kHz')
```

Bat Echolocation (STFT)
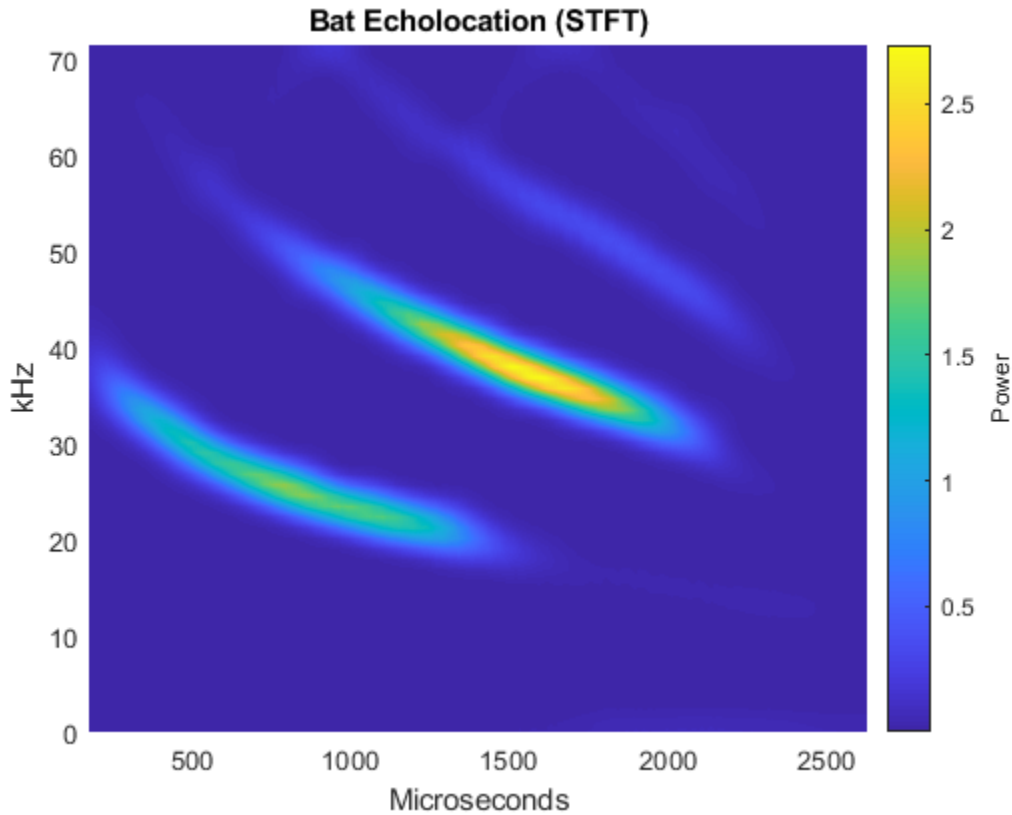
For both the simulated and natural modulated signals, the CWT provides results similar to the STFT.

Detection of Transients in Oscillations Using the CWT

There are certain situations in time-frequency analysis where the CWT can provide a more informative time-frequency transform than the short-time Fourier transform. One such situation occurs when the signal is corrupted by transients. The appearance and disappearance of these transients often has physical significance. Therefore, it is important to be able to localize these transients in addition to characterizing oscillatory components in the signal. To simulate this, create a signal consisting of two sine waves with frequencies of 150 and 200 Hz. The sampling frequency is 1 kHz. The sine waves have disjoint time supports. The 150-Hz sine wave occurs between 100 and 300 milliseconds. The 200-Hz sine wave occurs from 700 milliseconds to 1 second. Additionally, there are two transients at 222 and 800 milliseconds. The signal is corrupted by noise.

```
rng default;
dt = 0.001;
t = 0:dt:1-dt;
addNoise = 0.025*randn(size(t));
x = cos(2*pi*150*t).*(t>=0.1 & t<0.3)+sin(2*pi*200*t).*(t>0.7);
x = x+addNoise;
x([222 800]) = x([222 800 ])+[-2 2];
figure;
plot(t.*1000,x);
xlabel('Milliseconds'); ylabel('Amplitude');
```
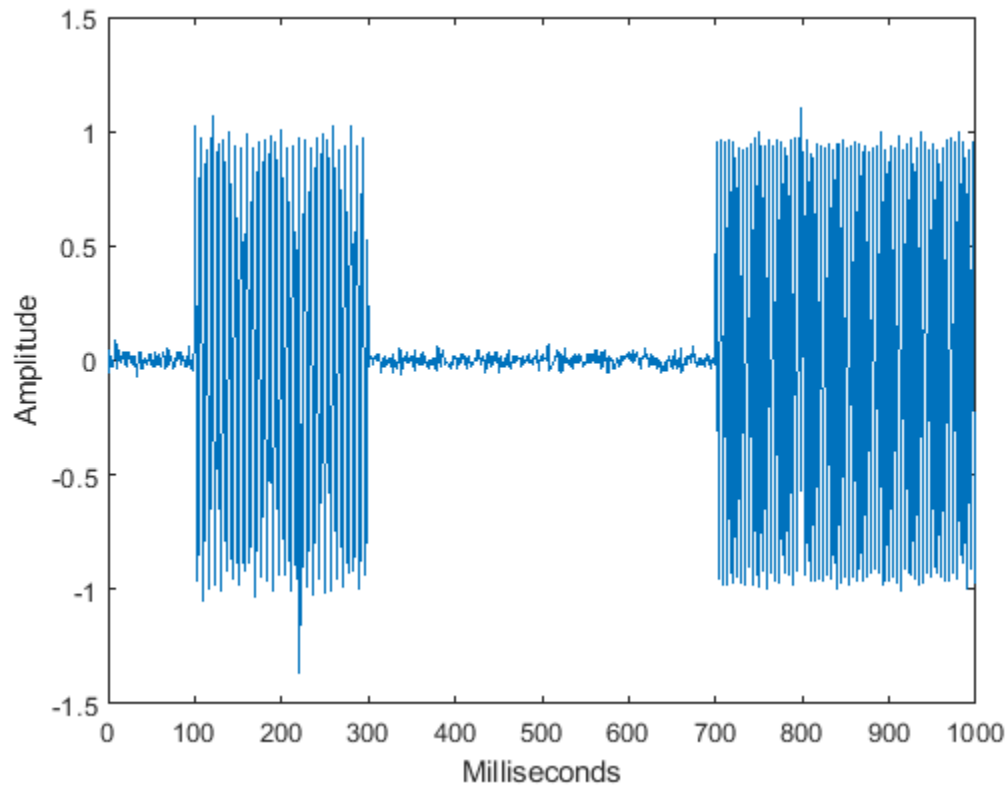
Zoom in on the two transients to see that they represent disturbances in the oscillations at 150 and 200 Hz.

```
subplot(2,1,1)
plot(t(184:264).*1000,x(184:264));
ylabel('Amplitude')
title('Transients')
axis tight;
subplot(2,1,2)
plot(t(760:840).*1000,x(760:840));
ylabel('Amplitude')
axis tight;
xlabel('Milliseconds')
```

10-31
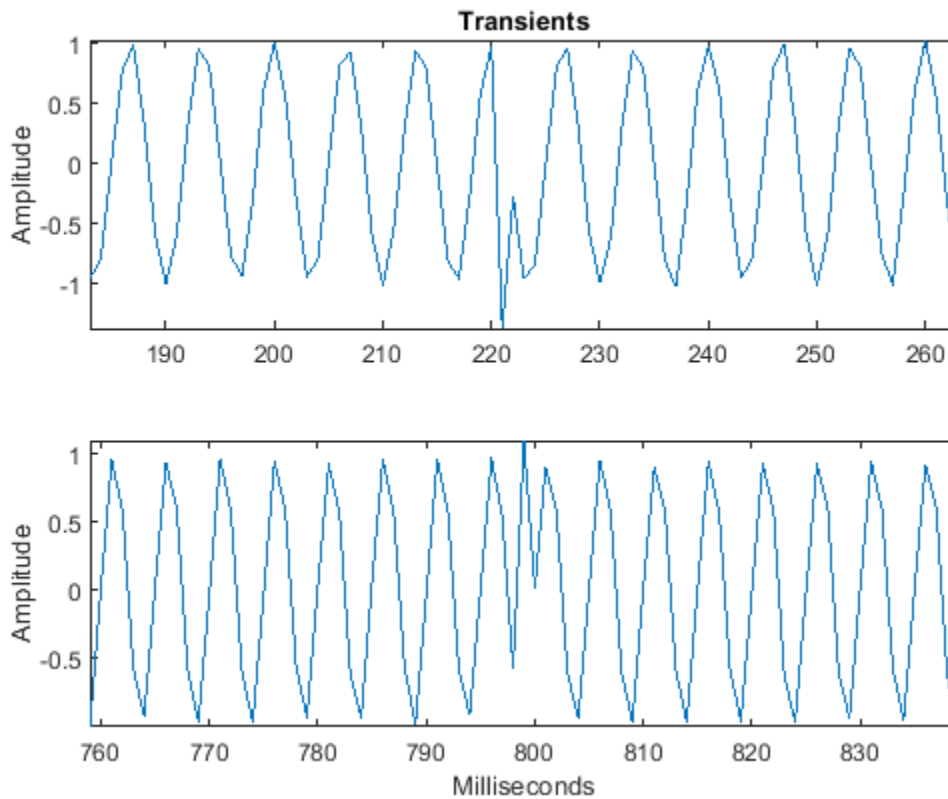
Obtain and plot the CWT using the analytic Morlet wavelet.

```
figure;
[cfs,f] = cwt(x,1/dt,'amor');
contour(t*1000,f,abs(cfs))
grid on
c = colorbar;
xlabel('Milliseconds')
ylabel('Frequency')
c.Label.String = 'Magnitude';
```
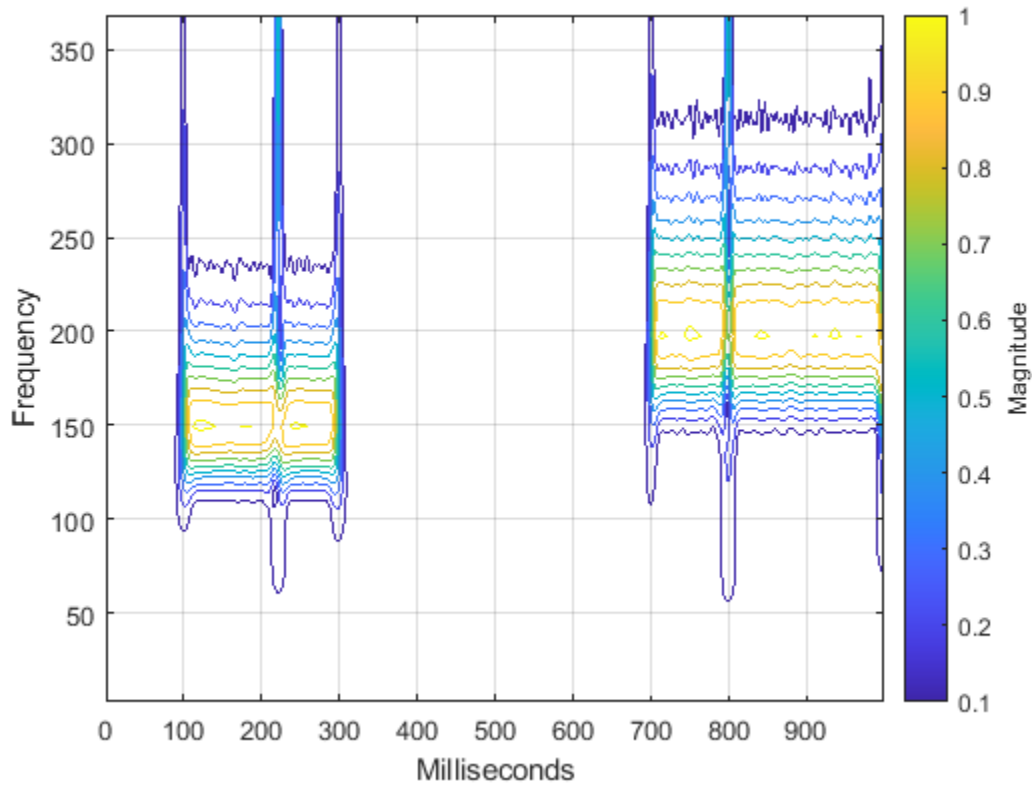
The analytic Morlet wavelet exhibits poorer frequency localization than the bump wavelet, but superior time localization. This makes the Morlet wavelet a better choice for transient localization. Plot the magnitude-squared fine scale coefficients to demonstrate the localization of the transients.

```
figure;
wt = cwt(x,1/dt,'amor');
plot(t.*1000,abs(wt(1,:)).^2);
xlabel('Milliseconds'); ylabel('Magnitude');
grid on;
title('Analytic Morlet Wavelet -- Fine Scale Coefficients');
hold off
```
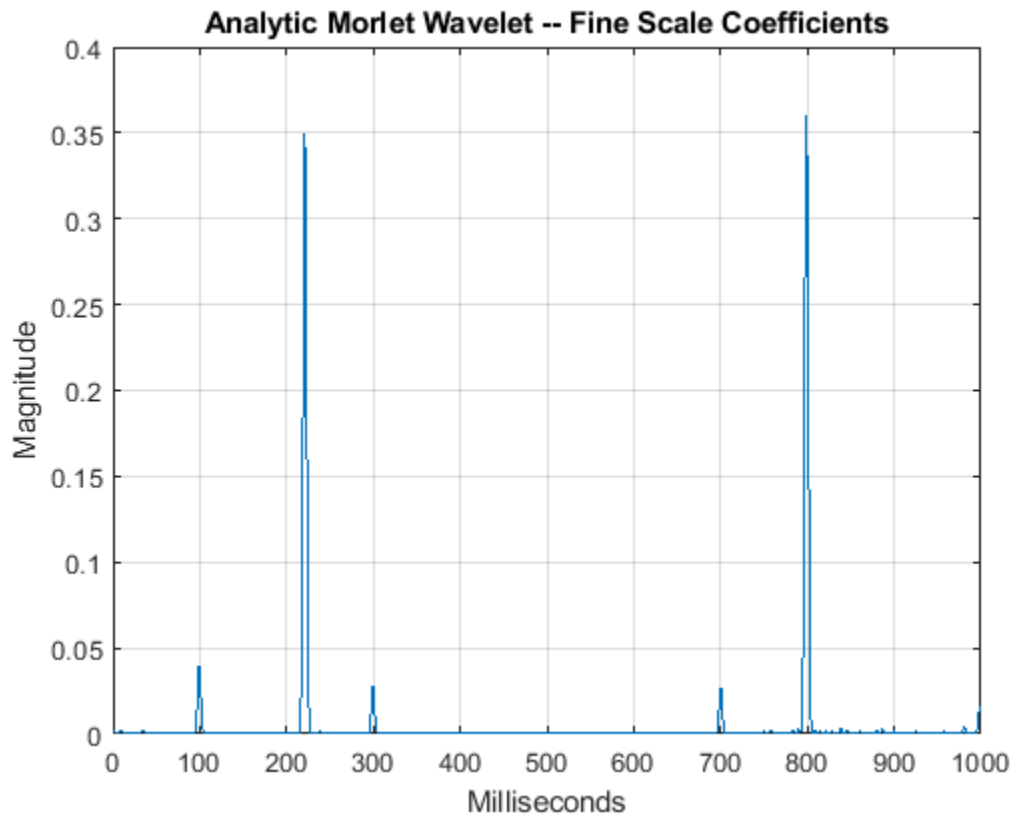
The wavelet shrinks to enable time localization of the transients with a high degree of accuracy while stretching to permit frequency localization of the oscillations at 150 and 200 Hz.

The STFT can only localize the transients to the width of the window. You have to plot the STFT in decibels (dB) to be able to visualize the transients.

```
[S,F,T] = spectrogram(x,50,48,128,1000);
surf(T.*1000,F,20*log10(abs(S)),'edgecolor','none'); view(0,90);
axis tight;
shading interp;
colorbar
xlabel('Time'), ylabel('Hz');
title('STFT')
```
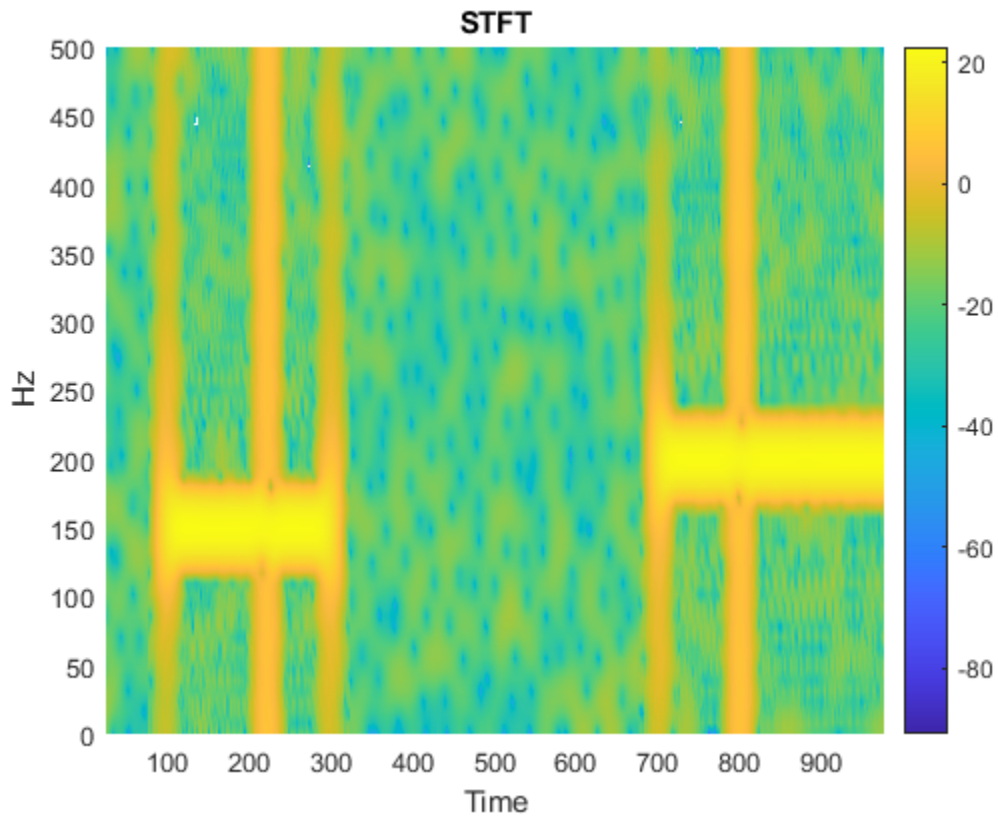
The transients appear in the STFT only as a broadband increase in power. Compare short-time power estimates obtained from the STFT before (centered at 183 msec) and after (centered at 223 msec) the appearance of the first transient.

```
figure;
plot(F,20*log10(abs(S(:,80))));
hold on;
plot(F,20*log10(abs(S(:,100))),'r');
legend('T = 183 msec','T = 223 msec')
xlabel('Hz');
ylabel('dB');
hold off;
```
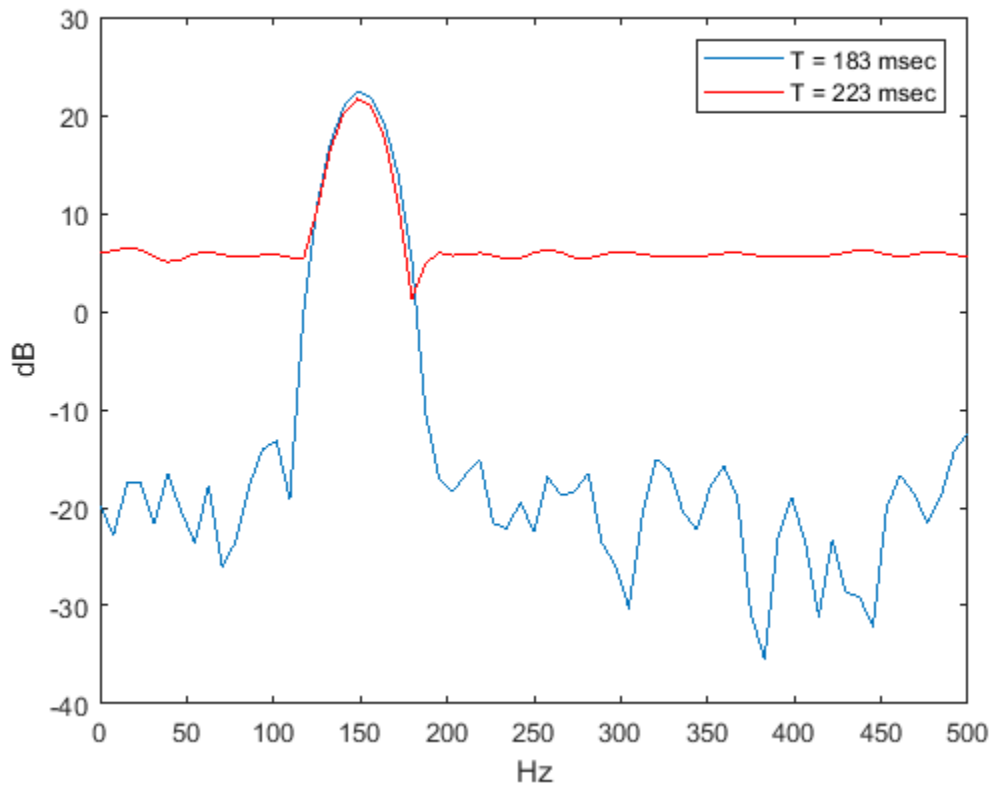
The STFT does not provide the ability to localize transients to the same degree as the CWT.

Removing A Time-Localized Frequency Component Using the Inverse CWT

Create a signal consisting of exponentially weighted sine waves. There are two 25-Hz components -- one centered at 0.2 seconds and one centered at 0.5 seconds. There are 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;
figure;
plot(t,x)
title('Superimposed Signal')
```
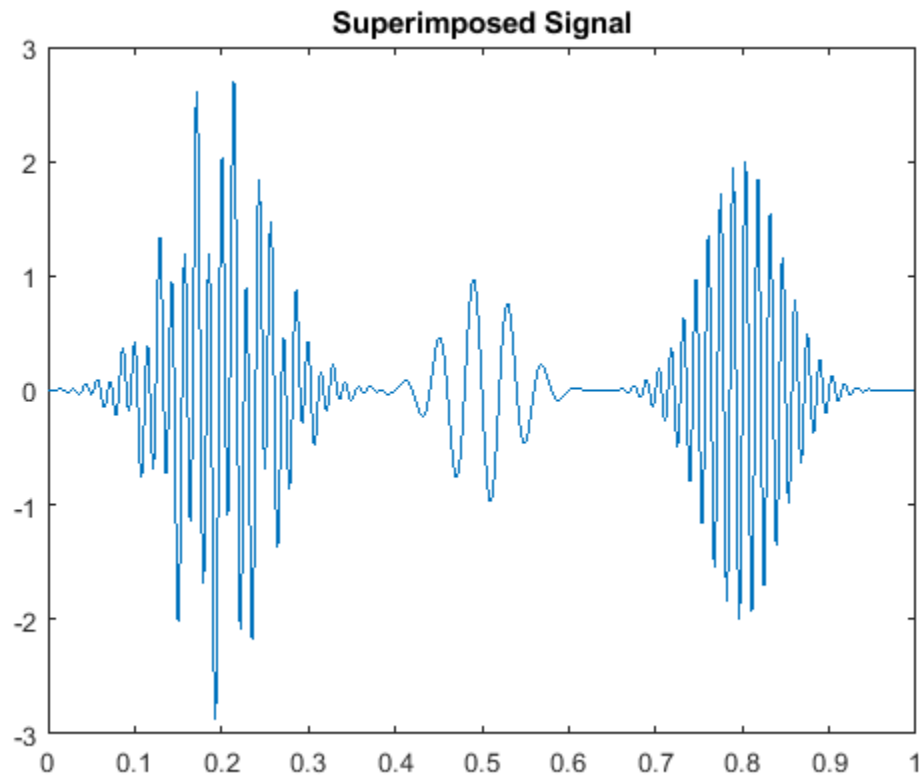
Obtain and display the CWT.

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

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)
```

Magnitude Scalogram
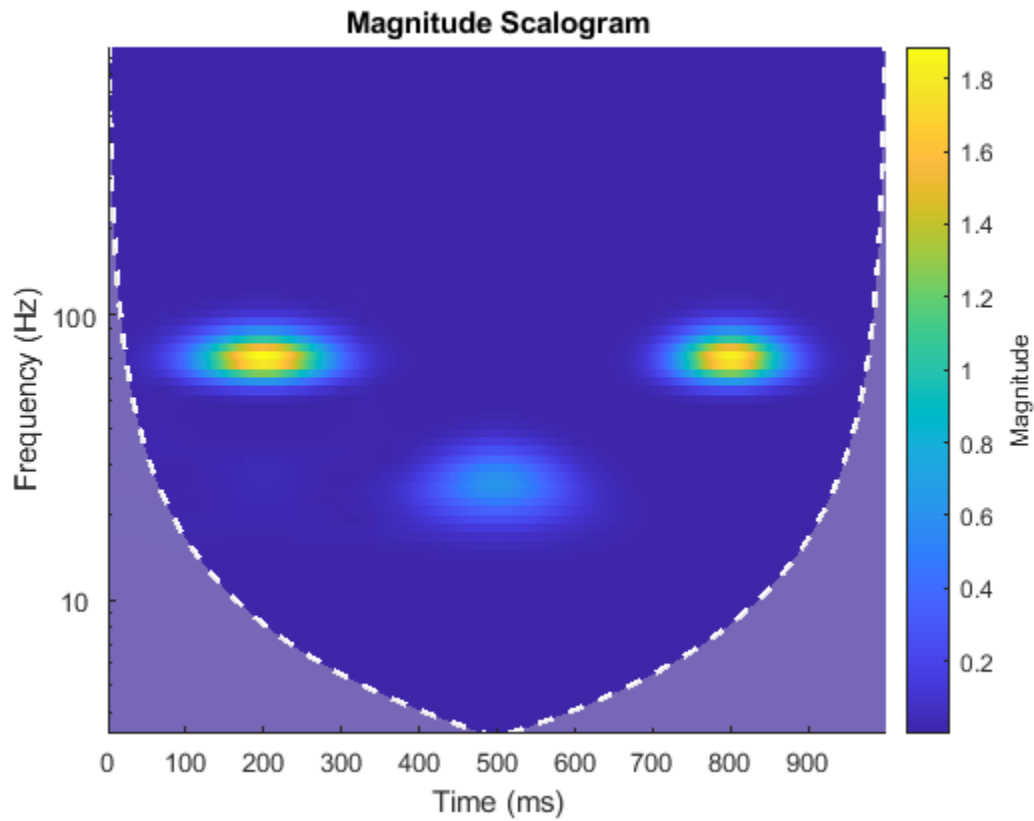
Plot the original signal and the reconstruction.

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

Original Signal

Signal with first 25-Hz component removed
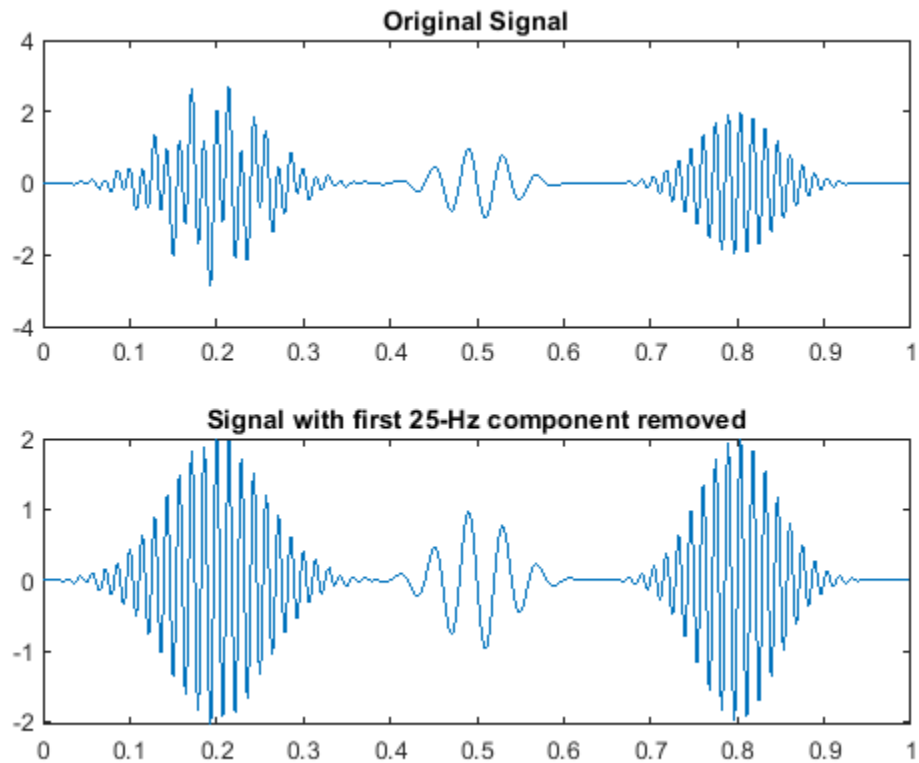
Finally, compare the reconstructed signal with the original signal without the 25-Hz component centered at 0.2 seconds.

```
y = x2+x3+x4;
figure;
plot(t,xrec)
hold on
plot(t,y,'r--')
legend('Inverse CWT approximation','Original signal without 25-Hz');
hold off
```
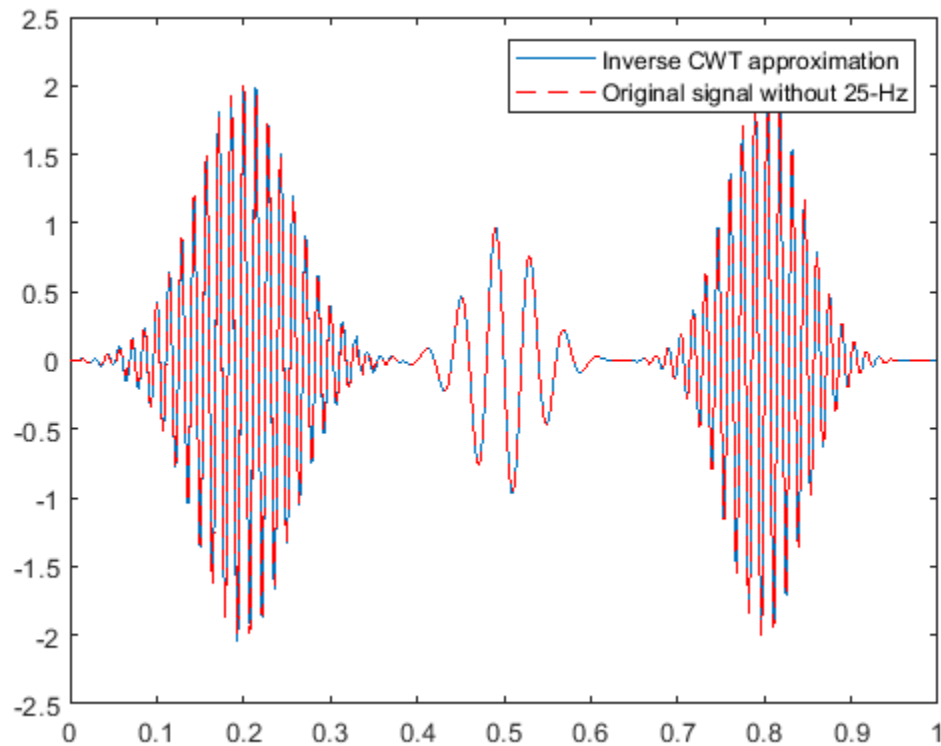
Determining Exact Frequency Through the Analytic CWT

When you obtain the wavelet transform of a sine wave using an analytic wavelet, the analytic CWT coefficients actually encode the frequency.

To illustrate this, consider an otoacoustic emission obtained from a human ear. Otoacoustic emissions (OAEs) are emitted by the cochlea (inner ear) and their presence are indicative of normal hearing. Load and plot the OAE data. The data are sampled at 20 kHz.

```
load dpoae
plot(t.*1000,dpoaets)
xlabel('Milliseconds')
ylabel('Amplitude')
```
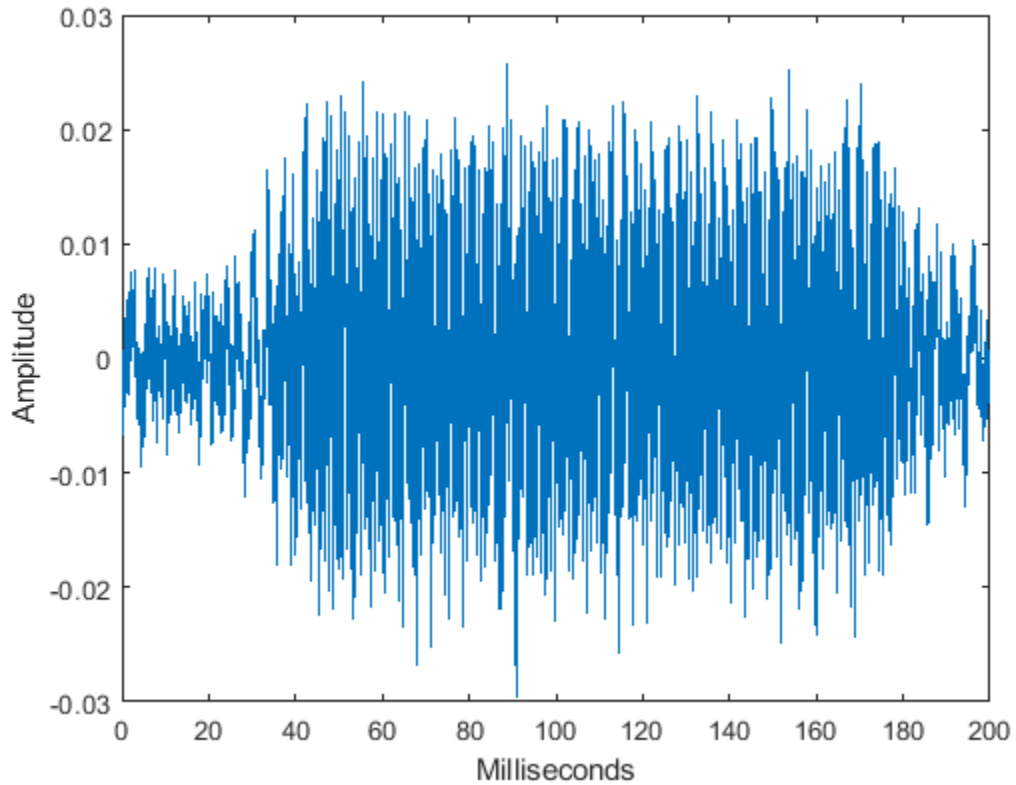
The emission was evoked by a stimulus beginning at 25 milliseconds and ending at 175 milliseconds. Based on the experimental parameters, the emission frequency should be 1230 Hz. Obtain and plot the CWT.

```
cwt(dpoaets,'bump',20000);
xlabel('Milliseconds');
```
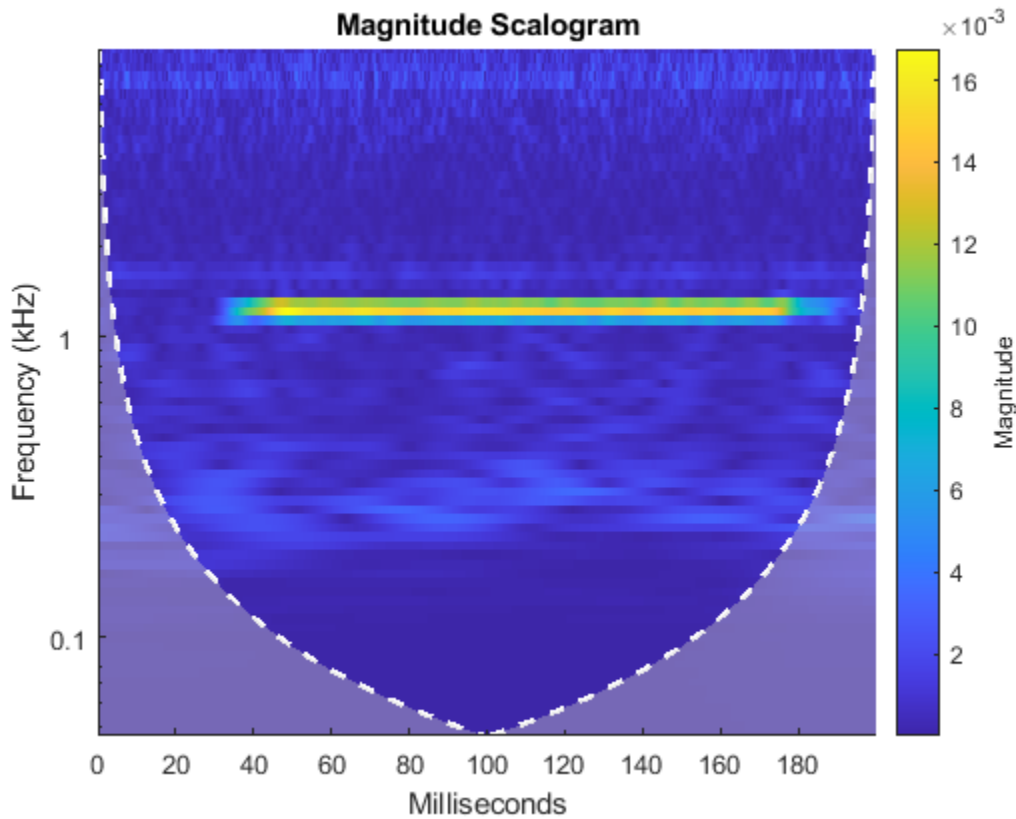
Magnitude Scalogram

You can investigate the time evolution of the OAE by finding the CWT coefficients closest in frequency to 1230 Hz and examining their magnitudes as a function of time. Plot the magnitudes along with time markers designating the beginning and end of the evoking stimulus.

```
[dpoaeCWT,f] = cwt(dpoaets,'bump',20000);
[~,idx1230] = min(abs(f-1230));
cfsOAE = dpoaeCWT(idx1230,:);
plot(t.*1000,abs(cfsOAE));
hold on
AX = gca;
plot([25 25],[AX.YLim(1) AX.YLim(2)],'r')
plot([175 175],[AX.YLim(1) AX.YLim(2)],'r')
xlabel('Milliseconds'), ylabel('Magnitude');
title('CWT Coefficient Magnitudes')
```
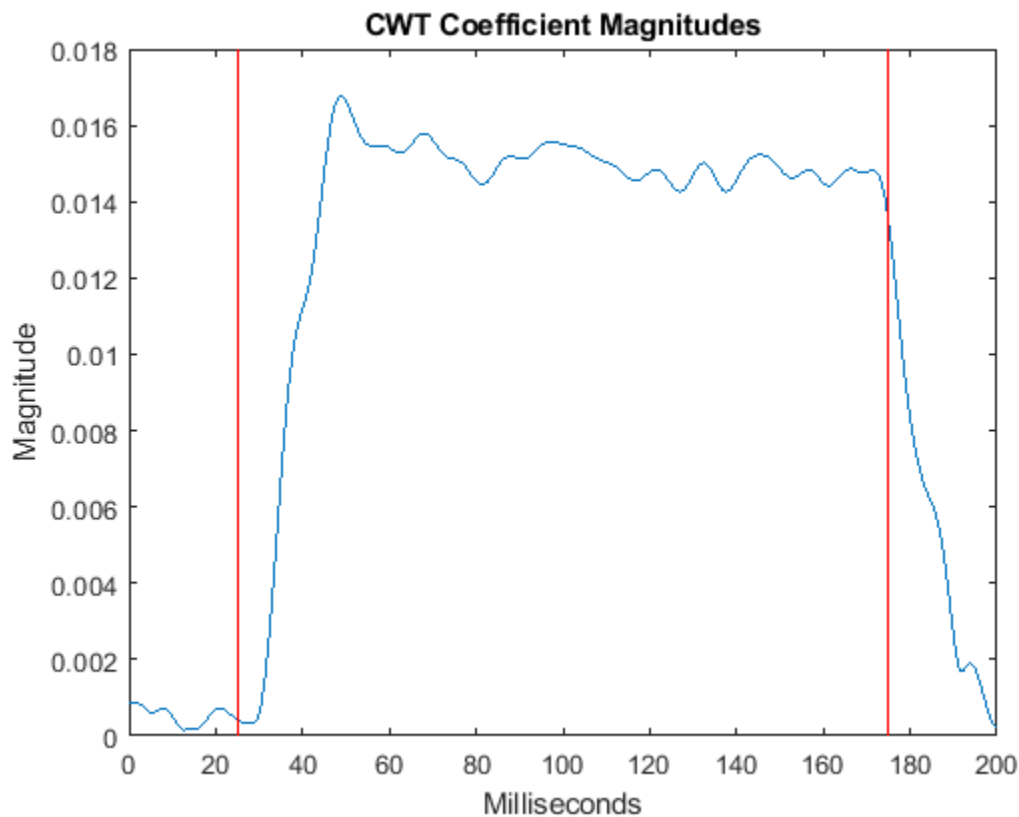
There is some delay between the onset of the evoking stimulus and the OAE. Once the evoking stimulus is terminated, the OAE immediately begins to decay in magnitude.

Another way to isolate the emission is to use the inverse CWT to reconstruct a frequency-localized approximation in the time domain.

Reconstruct a frequency-localized emission approximation by inverting the CWT in the frequency range [1150 1350] Hz. Plot the original data along with the reconstruction and markers indicating the beginning and end of the evoking stimulus.

```
xrec = icwt(dpoaeCWT,f,[1150 1350]);
figure
plot(t.*1000,dpoaets)
hold on;
plot(t.*1000,xrec,'r')
AX = gca;
ylim = AX.YLim;
plot([25 25],ylim,'k')
plot([175 175],ylim,'k')
xlabel('Milliseconds')
ylabel('Amplitude')
title('Frequency-Localized Reconstruction of Emission')
```

Frequency-Localized Reconstruction of Emission

In the time-domain data, you clearly see how the emission ramps on and off at the application and termination of the evoking stimulus.

It is important to note that even though a range of frequencies were selected for the reconstruction, the analytic wavelet transform actually encodes the exact frequency of the emission. To demonstrate this, take the Fourier transform of the emission approximation reconstructed from the analytic CWT.

```
xdft = fft(xrec);
freq = 0:2e4/numel(xrec):1e4;
xdft = xdft(1:numel(xrec)/2+1);
figure
plot(freq,abs(xdft));
xlabel('Hz'); ylabel('Magnitude')
title('Fourier Transform of CWT-Based Signal Approximation');
[~,maxidx] = max(abs(xdft));
fprintf('The frequency is %4.2f Hz\n',freq(maxidx));

The frequency is 1230.00 Hz
```

Fourier Transform of CWT-Based Signal Approximation

Conclusions and Further Reading

In this example you learned how to use the CWT to obtain a time-frequency analysis of a 1-D signal using an analytic wavelet with `cwt`. You saw examples of signals where the CWT provides similar results to the STFT and an example where the CWT can provide more interpretable results than the STFT. Finally, you learned how to reconstruct time-scale (frequency) localized approximations to a signal using `icwt`.

For more information on the CWT see the Wavelet Toolbox documentation.

Reference: Mallat, S. "A Wavelet Tour of Signal Processing: The Sparse Way", Academic Press, 2009.

Appendix

The following helper functions are used in this example.

- helperCWTTimeFreqPlot

Time-Frequency Reassignment and Mode Extraction with Synchrosqueezing

This example shows how to use wavelet synchrosqueezing to obtain a higher resolution time-frequency analysis. The example also shows how to extract and reconstruct oscillatory modes in a signal.

In many practical applications across a wide range of disciplines, signals occur which consist of a number of oscillatory components, or modes. These components often exhibit slow variations in amplitude and smooth changes in frequency over time. Signals consisting of one or more such components are called amplitude and frequency modulated (AM-FM). Individual AM-FM components of signals are also referred to as intrinsic modes, or intrinsic mode functions (IMF).

Wavelet synchrosqueezing is a method for analyzing signals consisting of a sum of well-separated AM-FM components, or IMFs. With synchrosqueezing, you can sharpen the time-frequency analysis of a signal as well as reconstruct individual oscillatory modes for isolated analysis.

Sharpen Time-Frequency Analysis

Synchrosqueezing can compensate for the spreading in time and frequency caused by linear transforms like the short-time Fourier and continuous wavelet transforms (CWT). In the CWT, the wavelet acts like a measuring device for the input signal. Accordingly, any time frequency analysis depends not only on the intrinsic time-frequency properties of the signal, but also on the properties of the wavelet.

To demonstrate this, first obtain and plot the CWT of 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;
[cfs,f] = cwt(quadchirp,'bump',fs);
helperCWTTimeFreqPlot(cfs,tquad,f,'surf','CWT of Quadratic Chirp','Seconds','Hz')
```

CWT of Quadratic Chirp
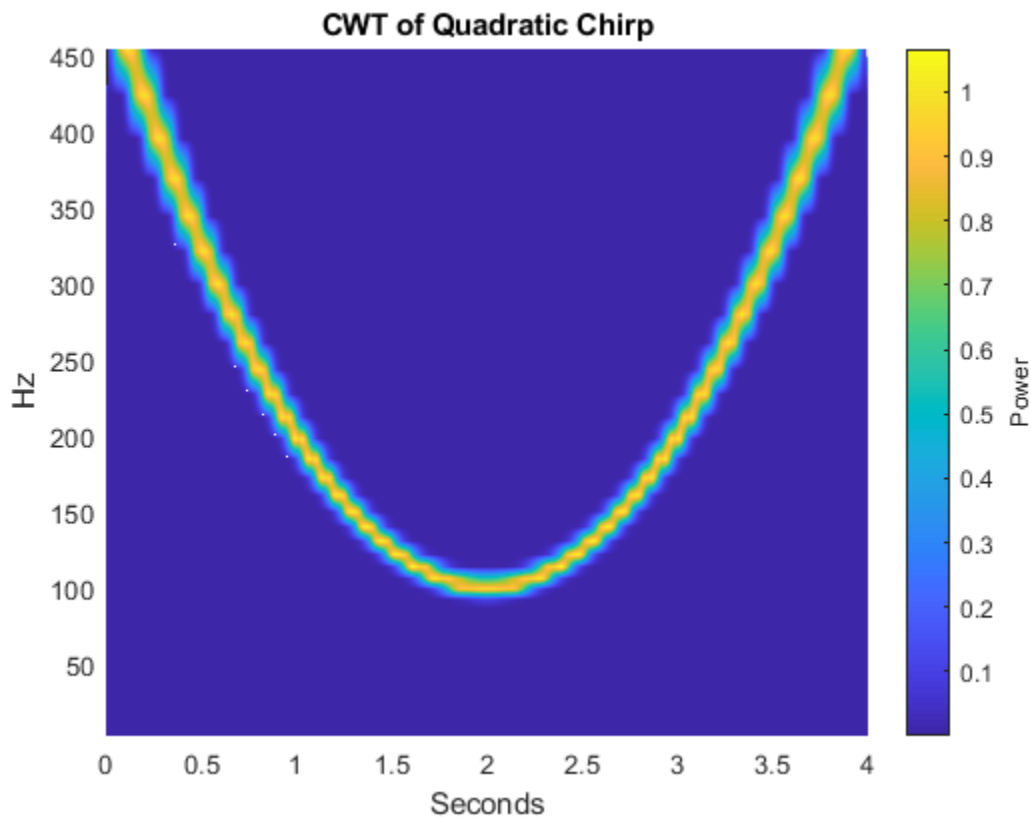
Note that the energy of the quadratic chirp is smeared in the time-frequency plane by the time-frequency concentration of the wavelet. For example, if you focus on the time-frequency concentration of the CWT magnitudes near 100 Hz, you see that it is narrower than that observed near 500 Hz. This is not an intrinsic property of the chirp. It is an artifact of the measuring device (the CWT). Compare the time-frequency analysis of the same signal obtained with the synchrosqueezed transform.

```
wsst(quadchirp,1000,'bump')
```
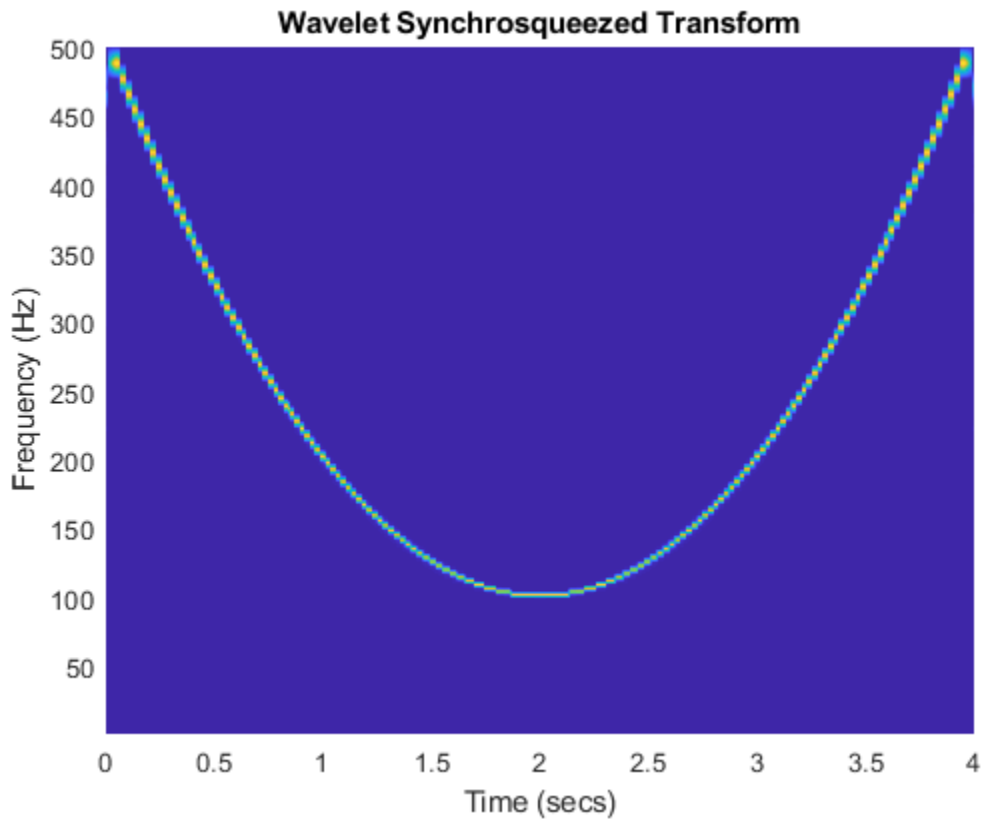
The synchrosqueezed transform uses the phase information in the CWT to sharpen the time-frequency analysis of the chirp.

Reconstruct Signal from Synchrosqueezed Transform

You can reconstruct a signal from the synchrosqueezed transform. This is an advantage synchrosqueezing has over other time-frequency reassignment techniques. The transform does not provide a perfect inversion, but the reconstruction is stable and the results are typically quite good. To demonstrate this, load, plot, and play a recording of a female speaker saying "I saw the sheep".

```
load wavsheep;
plot(tsh,sheep)
title(' "I saw the sheep." ');
xlabel('Time (secs)'); ylabel('Amplitude');
grid on;
hplayer = audioplayer(sheep,fs);
play(hplayer);
```

"I saw the sheep."
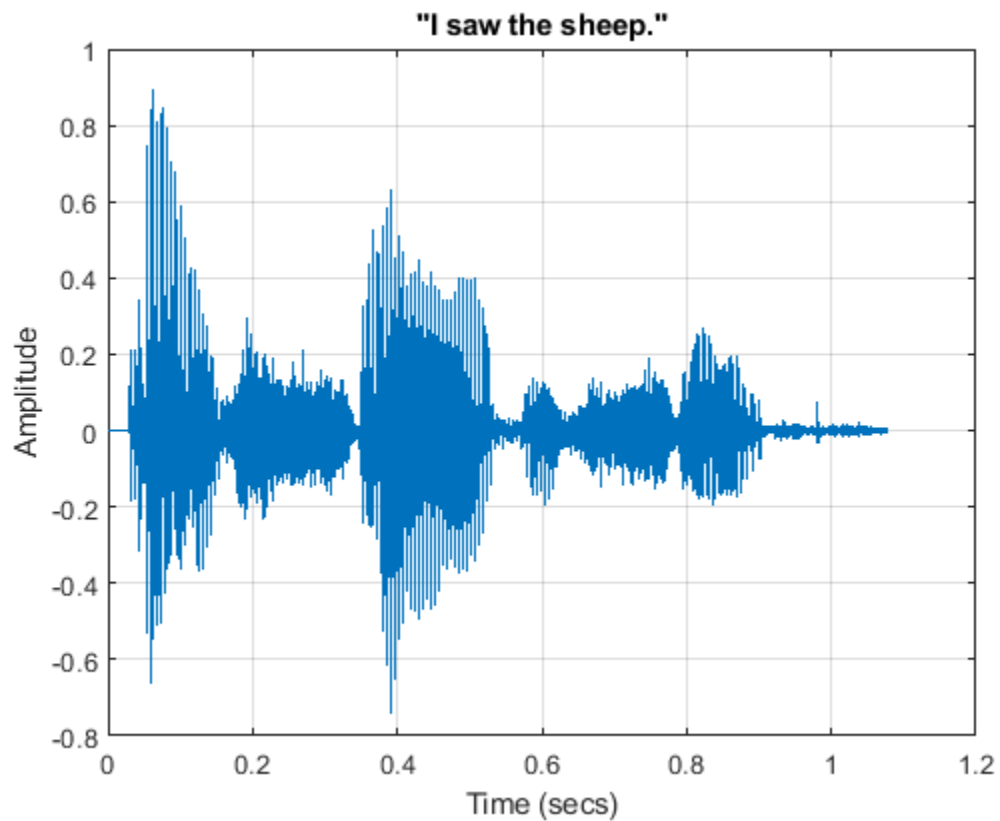
Plot the synchrosqueezed transform of the speech sample.

```
[sst,f] = wsst(sheep,fs,'bump');
contour(tsh,f,abs(sst));
title('Wavelet Synchrosqueezed Transform');
xlabel('Time (secs)'); ylabel('Hz');
grid on;
```
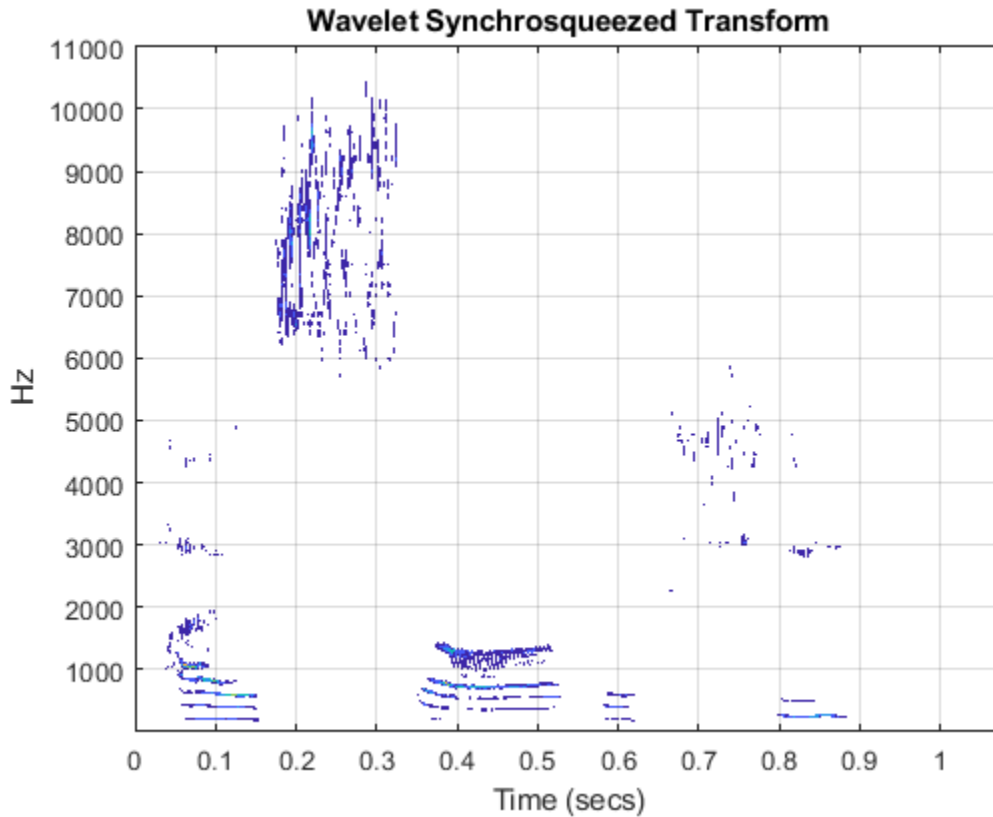
Reconstruct the signal from the synchrosqueezed transform and compare the reconstruction to the original waveform.

```
xrec = iwsst(sst,'bump');
plot(tsh,sheep)
hold on;
plot(tsh,sheep,'r');
xlabel('Time (secs)'); ylabel('Amplitude');
grid on;
title('Reconstruction From Synchrosqueezed Transform');
legend('Original Waveform','Synchrosqueezed Reconstruction');
sprintf('The maximum sample-by-sample difference is %1.3f', ...
    norm(abs(xrec-sheep),Inf))
hold off;
```

```
ans =

    'The maximum sample-by-sample difference is 0.004'
```
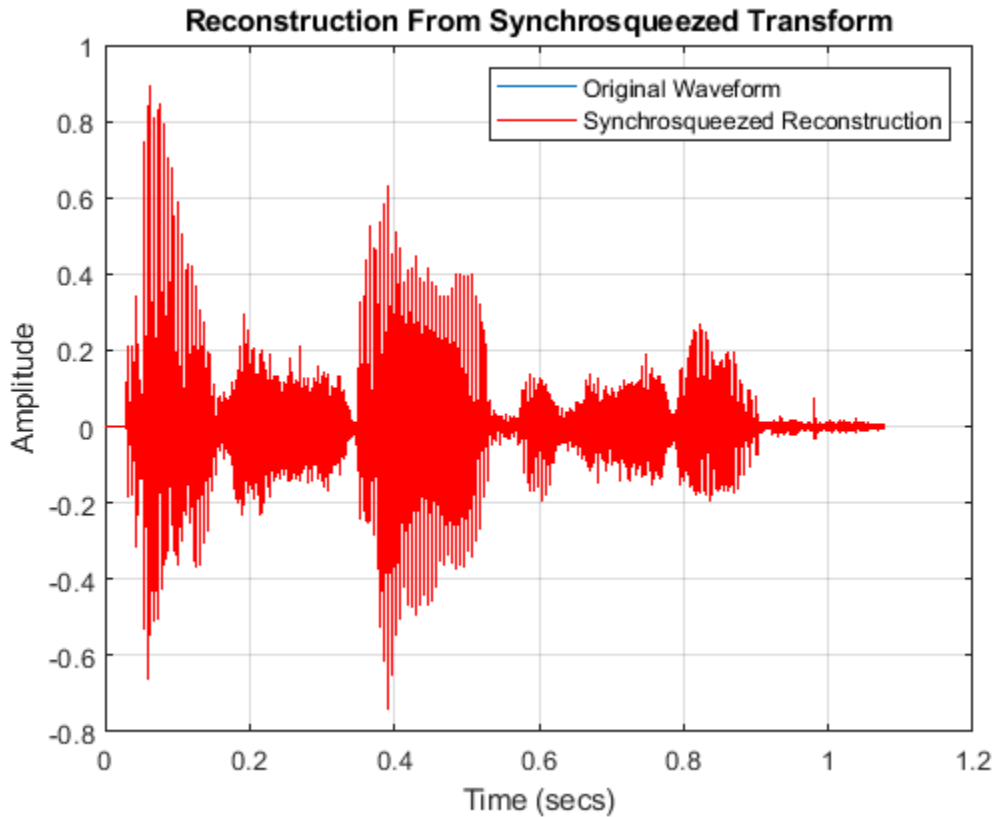
Play the reconstructed signal and compare with the original.

```
hplayerRecon = audioplayer(xrec,fs);
play(hplayerRecon)
```

The ability to reconstruct from the synchrosqueezed transform enables you to extract signal components from localized regions of the time-frequency plane. The next section demonstrates this idea.

Identify Ridges and Reconstruct Modes

A time-frequency ridge is defined by the local maxima of a time-frequency transform. Because the synchrosqueezed transform concentrates oscillatory modes in a narrow region of the time-frequency plane and is invertible, you can reconstruct individual modes by:

1   Identifying ridges in the magnitudes of the synchrosqueezed transform
2   Reconstructing along the ridge

This allows you to isolate and analyze modes which may be difficult or impossible to extract with conventional bandpass filtering.

To illustrate this, consider an echolocation pulse emitted by a big brown bat (Eptesicus Fuscus). The sampling interval is 7 microseconds. 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 the data and plot the synchrosqueezed transform.

```
load batsignal;
time = 0:DT:(numel(batsignal)*DT)-DT;
[sst,f] = wsst(batsignal,1/DT);
contour(time.*1e6,f./1000,abs(sst));
grid on;
xlabel('microseconds'); ylabel('kHz');
title('Bat Echolocation Pulse');
```


Bat Echolocation Pulse
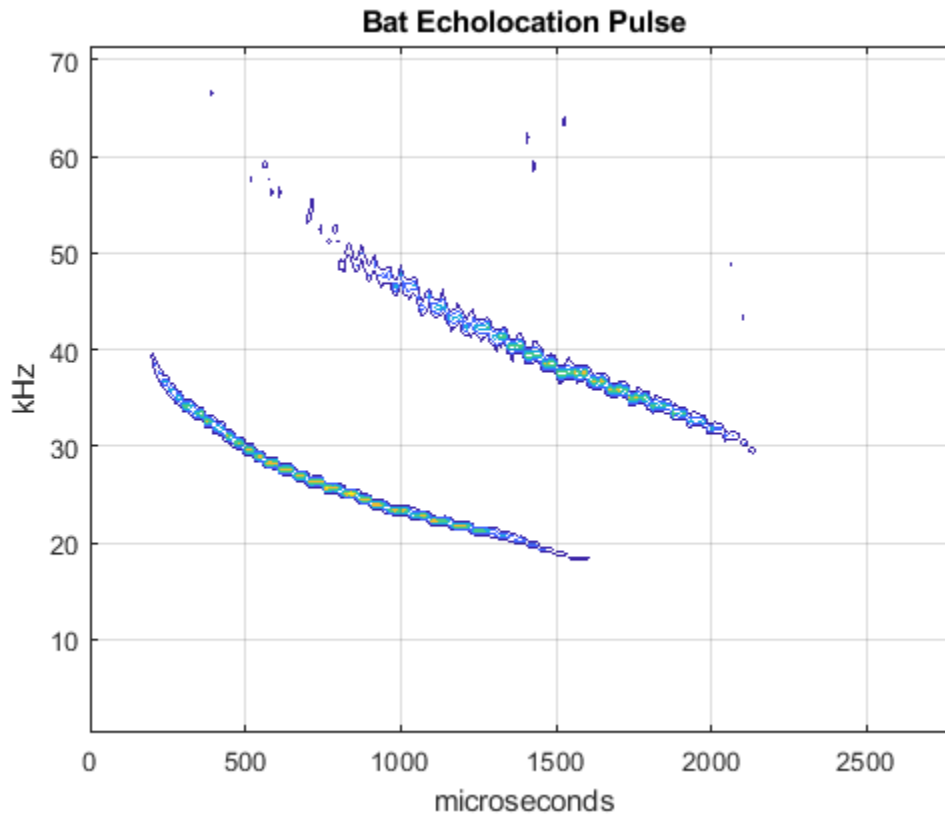
Note that there are two modulated modes that trace out curved paths in the time-frequency plane. Attempting to separate these components with conventional bandpass filtering does not work because the filter would need to operate in a time-varying manner. For example, a conventional filter with a passband of 18 to 40 kHz would capture the energy in the earliest-occurring pulse, but would also capture energy from the later pulse.

Synchrosqueezing can separate these components by filtering and reconstructing the synchrosqueezed transform in a time-varying manner.

First, extract the two highest-energy ridges from the synchrosqueezed transform.

```
[fridge,iridge] = wsstridge(sst,5,f,'NumRidges',2);
hold on;
plot(time.*1e6,fridge./1e3,'k--','linewidth',2);
hold off;
```

Bat Echolocation Pulse
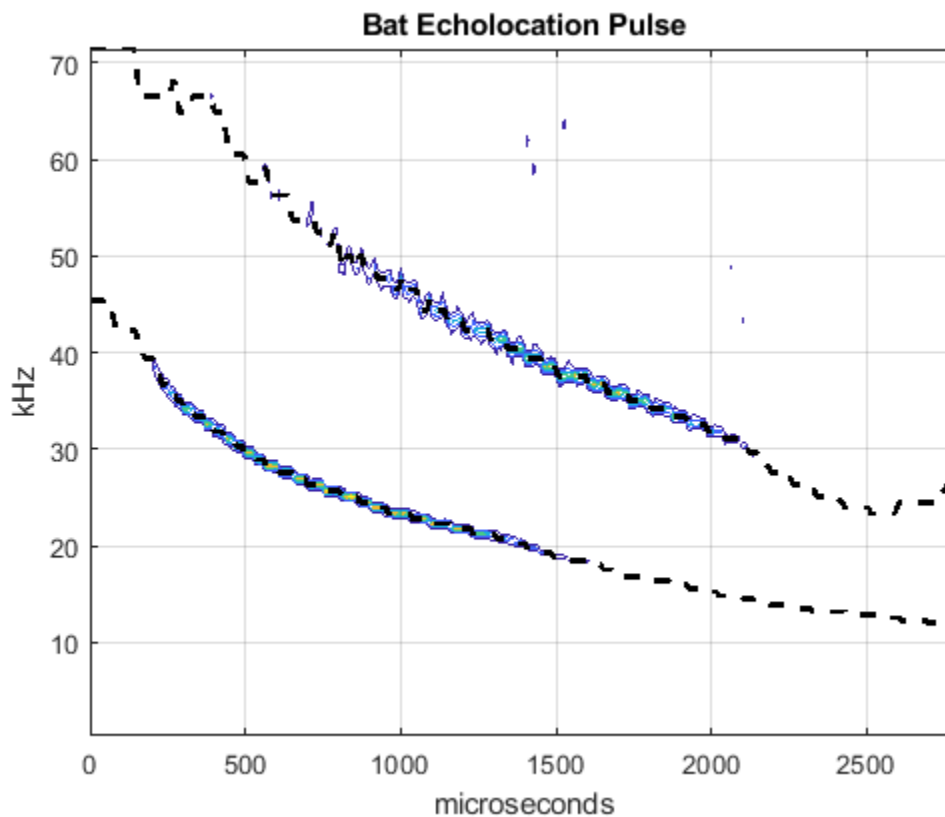
The ridges follow the time-varying nature of the modulated pulses. Reconstruct the signal modes by inverting the synchrosqueezed transform.

```
xrec = iwsst(sst,iridge);
subplot(2,1,1)
plot(time.*1e6,batsignal); hold on;
plot(time.*1e6,xrec(:,1),'r');
grid on; ylabel('Amplitude');
title('Bat Echolocation Pulse with Reconstructed Modes');
legend('Original Signal','Mode 1','Location','SouthEast');
subplot(2,1,2);
plot(time.*1e6,batsignal); hold on;
grid on;
plot(time.*1e6,xrec(:,2),'r');
xlabel('microseconds'); ylabel('Amplitude');
legend('Original Signal','Mode 2','Location','SouthEast');
```

Bat Echolocation Pulse with Reconstructed Modes
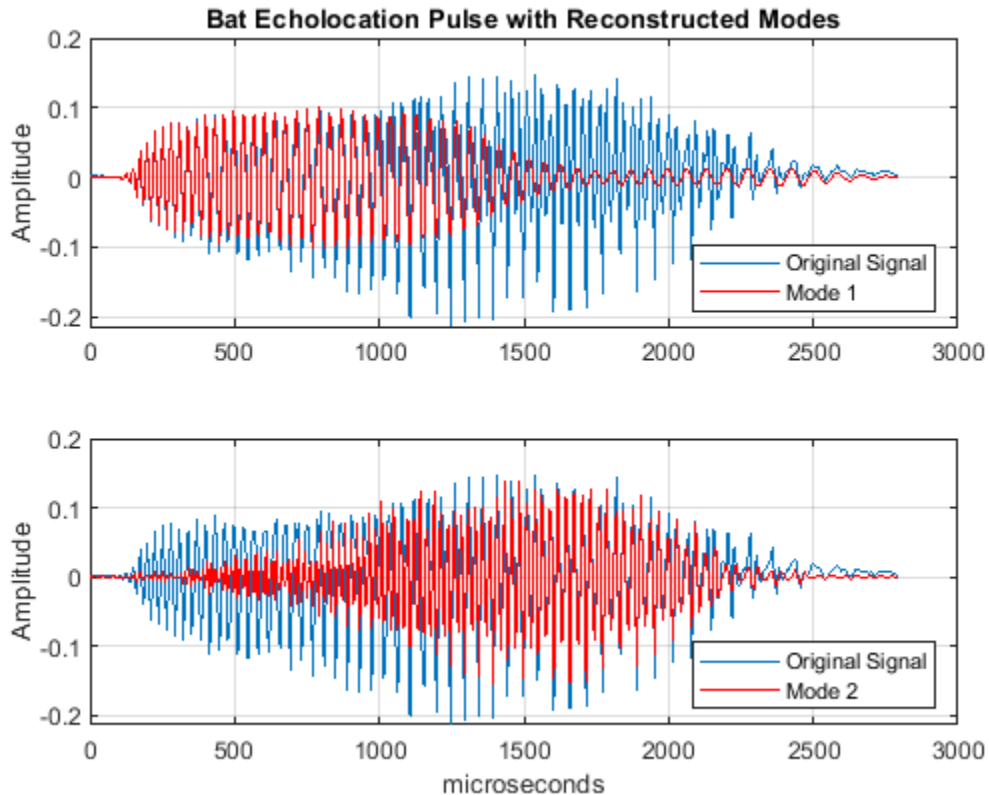
You see that synchrosqueezing has extracted the two modes. If you sum the two dominant modes at each point in time, you essentially recover the entire echolocation pulse. In this application, synchrosqueezing allows you to isolate signal components where a traditional bandpass filter would fail.

Penalty Term in Ridge Extraction

The previous mode extraction example used a penalty term in the ridge extraction without explanation.

When you extract multiple ridges, or you have a single modulated component in additive noise, it is important to use a penalty term in the ridge extraction. The penalty term serves to prevent jumps in frequency as the region of highest energy in the time-frequency plane moves.

To demonstrate this, consider a two-component signal consisting of an amplitude and frequency-modulated signal plus a sine wave. The signal is sampled at 1000 Hz. The sine wave frequency is 18 Hz. The AM-FM signal is defined by:

$$(2 + \cos(4\pi t)) \sin(2\pi 231 t + 90 \sin(3\pi t))$$

Load the signal and obtain the synchrosqueezed transform.

```
load multicompsig;
sig = sig1+sig2;
[sst,f] = wsst(sig,sampfreq);
figure;
```

10-55

```
contour(t,f,abs(sst));
grid on;
title('Synchrosqueezed Transform of AM-FM Signal Plus Sine Wave');
xlabel('Time (secs)'); ylabel('Hz');
```
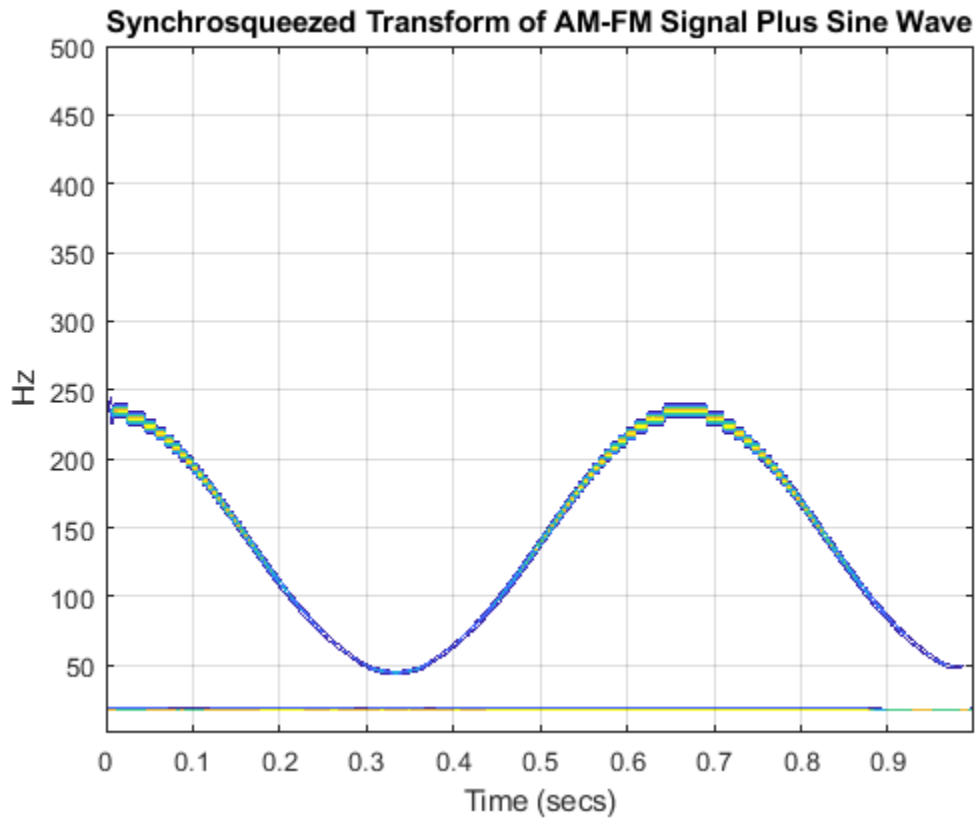

First attempt to extract two ridges from the synchrosqueezed transform without using a penalty.

```
[fridge,iridge] = wsstridge(sst,f,'NumRidges',2);
hold on;
plot(t,fridge,'k--','linewidth',2);
```
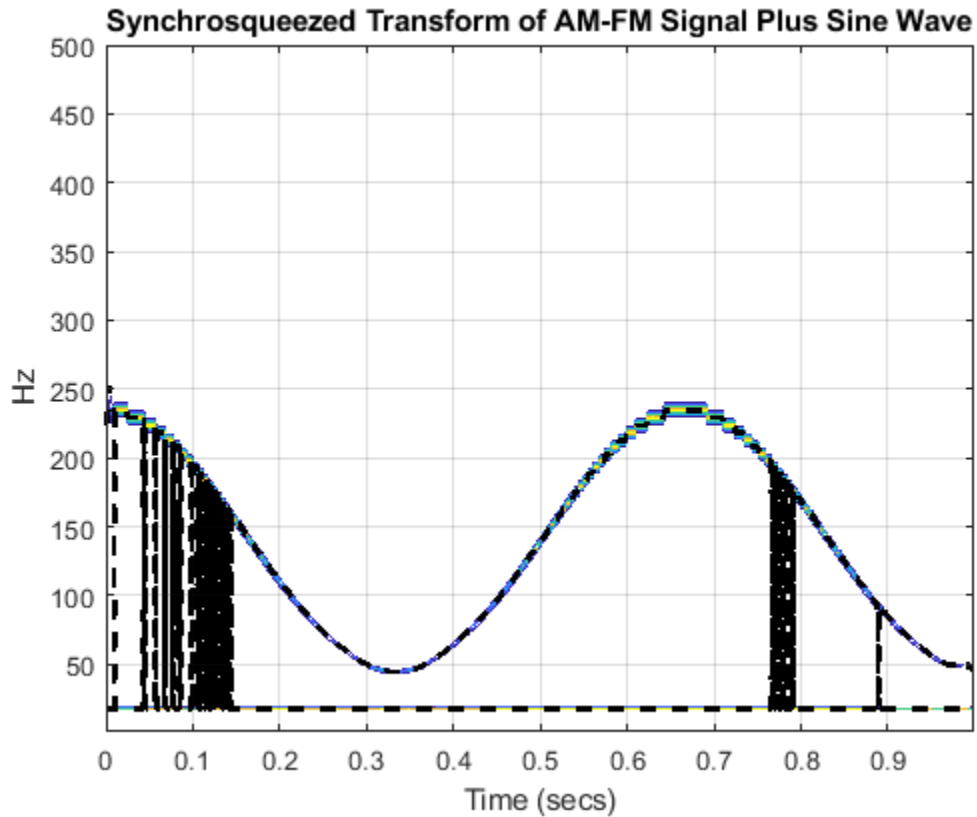
You see that the ridge jumps between the AM-FM signal and the sine wave as the region of highest energy in the time-frequency plane changes between the two signals. Add a penalty term of 5 to the ridge extraction. In this case, jumps in frequency are penalized by a factor of 5 times the distance between the frequencies in terms of bins (not actual frequency in hertz).

```
[fridge,iridge] = wsstridge(sst,5,f,'NumRidges',2);
figure;
contour(t,f,abs(sst));
grid on;
title('Synchrosqueezed Transform of AM-FM Signal Plus Sine Wave');
xlabel('Time (secs)'); ylabel('Hz');
hold on;
plot(t,fridge,'k--','linewidth',2);
hold off;
```

Synchrosqueezed Transform of AM-FM Signal Plus Sine Wave
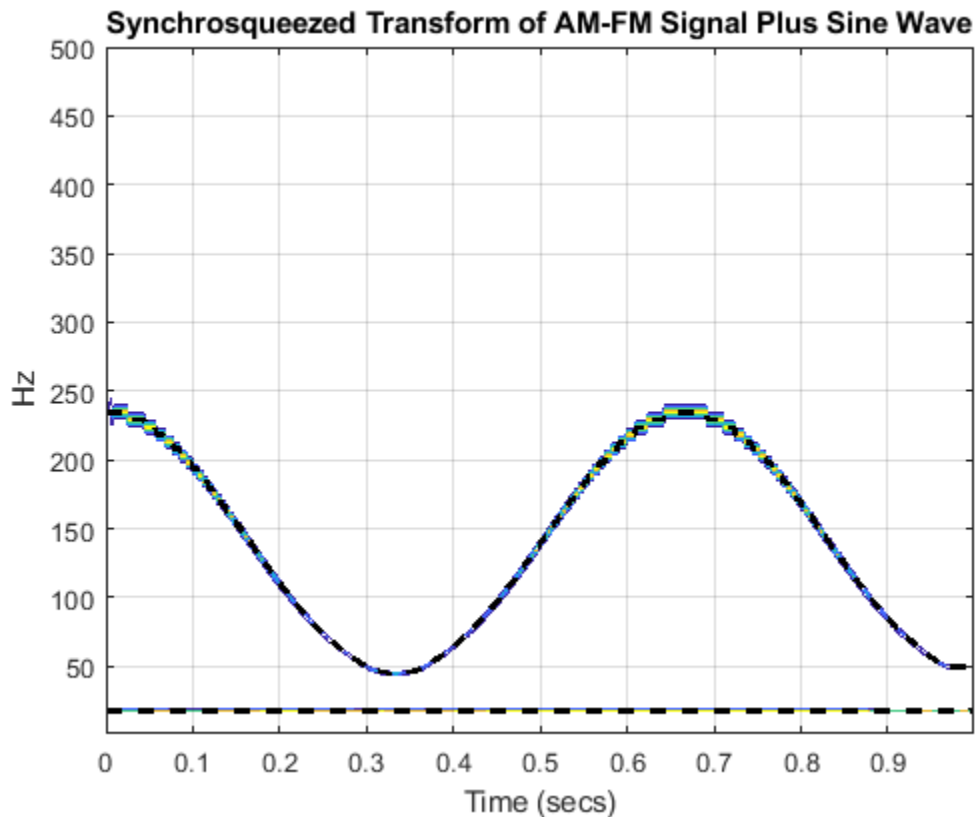
With the penalty term, the two modes of oscillation are isolated in two separate ridges. Reconstruct the modes along the time-frequency ridges from the synchrosqueezed transform.

```
xrec = iwsst(sst,iridge);
```

Plot the reconstructed modes along with the original signals for comparison.

```
subplot(2,2,1)
plot(t,xrec(:,1)); grid on;
ylabel('Amplitude');
title('Reconstructed Mode');
ylim([-1.5 1.5]);
subplot(2,2,2);
plot(t,sig2); grid on;
ylabel('Amplitude');
title('Original Component');
ylim([-1.5 1.5]);
subplot(2,2,3);
plot(t,xrec(:,2)); grid on;
xlabel('Time (secs)'); ylabel('Amplitude');
title('Reconstructed Mode');
ylim([-1.5 1.5]);
subplot(2,2,4);
plot(t,sig1); grid on;
xlabel('Time (secs)'); ylabel('Amplitude');
title('Original Component');
ylim([-1.5 1.5]);
```
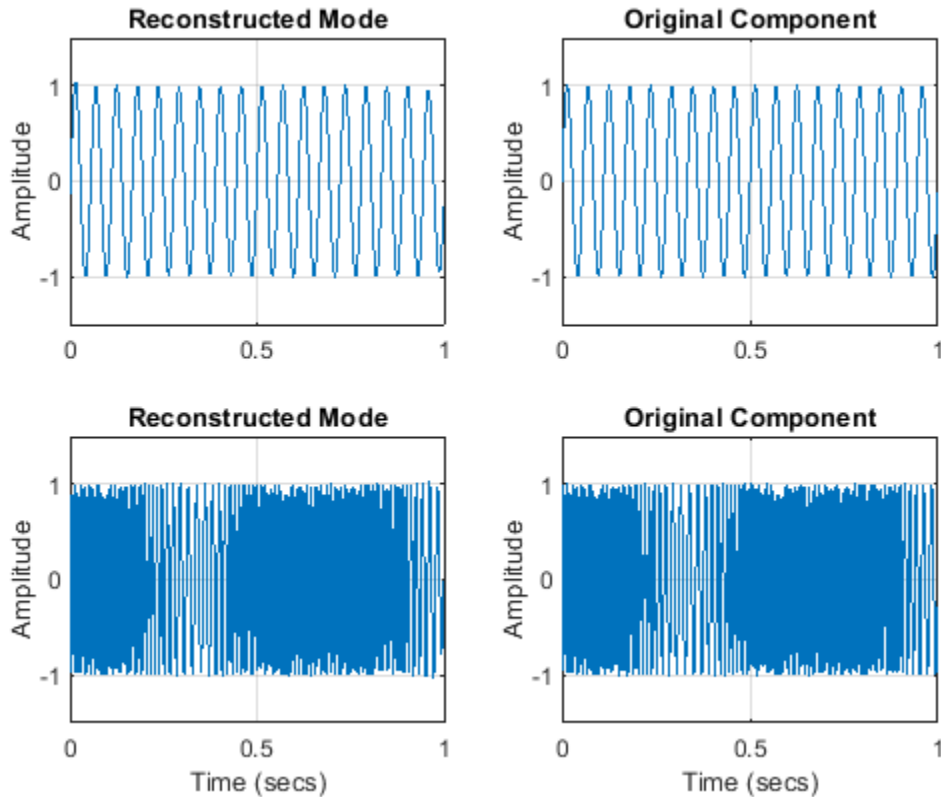
Conclusions

In this example, you learned how to use wavelet synchrosqueezing to obtain a higher-resolution time-frequency analysis.

You also learned how to identify maxima ridges in the synchrosqueezed transform and reconstruct the time waveforms corresponding to those modes. Mode extraction from the synchrosqueezing transform can help you isolate signal components, which are difficult or impossible to isolate with conventional bandpass filtering.

References

Daubechies, I., Lu, J., and Wu, H-T. "Synchrosqueezed wavelet transforms: an empirical mode decomposition-like tool." Appl. Comput. Harmonic Anal., 30(2):243-261, 2011.

Thakur, G., Brevdo, E., Fuckar, N.S. and Wu H-T. "The synchrosqueezing algorithm for time-varying spectral analysis: robustness properties and new paleoclimate applications." Signal Processing, 93(5), 1079-1094, 2013.

Meignen, S., Oberlin, T., and McLaughlin, S. "A new algorithm for multicomponent signals analysis based on synchrosqueezing: with an application to signal sampling and denoising." IEEE Transactions on Signal Processing, vol. 60, no. 12, pp. 5787-5798, 2012.

Appendix

The following helper function is used in this example.

- helperScaleVector

Frequency- and Time-Localized Reconstruction from the Continuous Wavelet Transform

Reconstruct a frequency-localized approximation of Kobe earthquake data. Extract information from the CWT for frequencies in the range of [0.030, 0.070] Hz.

```
load kobe
```

Obtain the CWT of the data.

```
[wt,f] = cwt(kobe,1);
```

Reconstruct the earthquake data, adding the signal mean back into the transformed data.

```
xrec = icwt(wt,f,[0.030 0.070],'SignalMean',mean(kobe));
```

Plot and compare the original data and the data for frequencies in the range of [0.030, 0.070] Hz.

```
subplot(2,1,1)
plot(kobe)
grid on
title('Original Data')
subplot(2,1,2)
plot(xrec)
grid on
title('Bandpass Filtered Reconstruction [0.030 0.070] Hz')
```
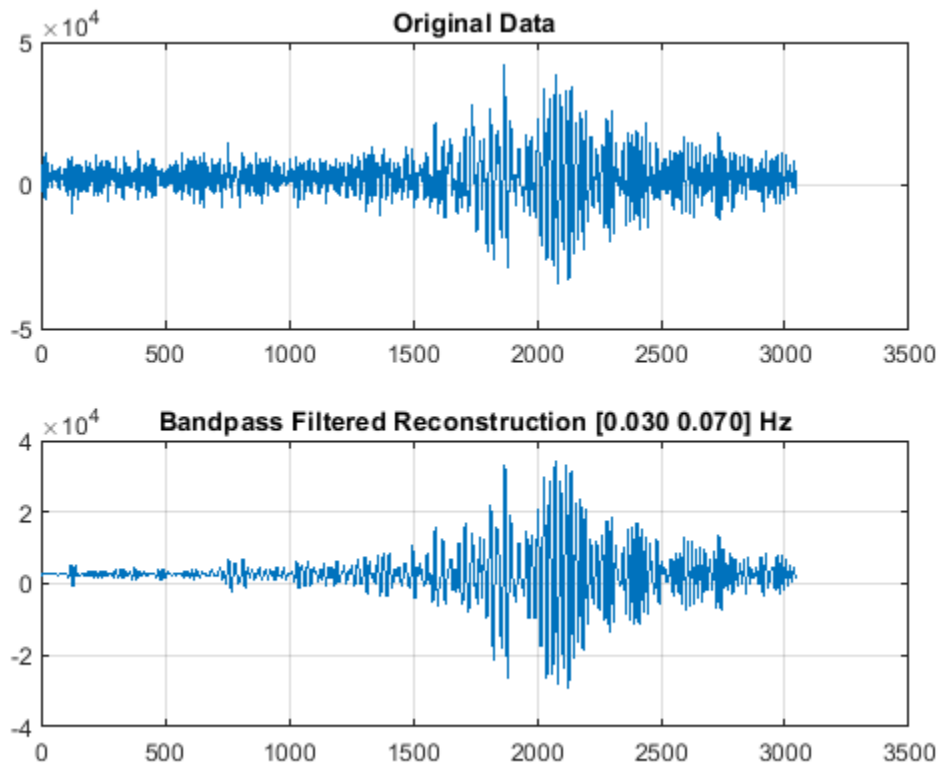
You can also use time periods, instead of frequency, with the CWT. Load the El Nino data and obtain its CWT, specifying the time period in years.

```
load ninoairdata
[cfs,period] = cwt(nino,years(1/12));
```

Obtain the inverse CWT for years 2 through 8.

```
xrec = icwt(cfs,period,[years(2) years(8)]);
```

Plot the CWT of the reconstructed data. Note the absence of energy outside the band of periods from 2 to 8 years.

```
figure
cwt(xrec,years(1/12))
```
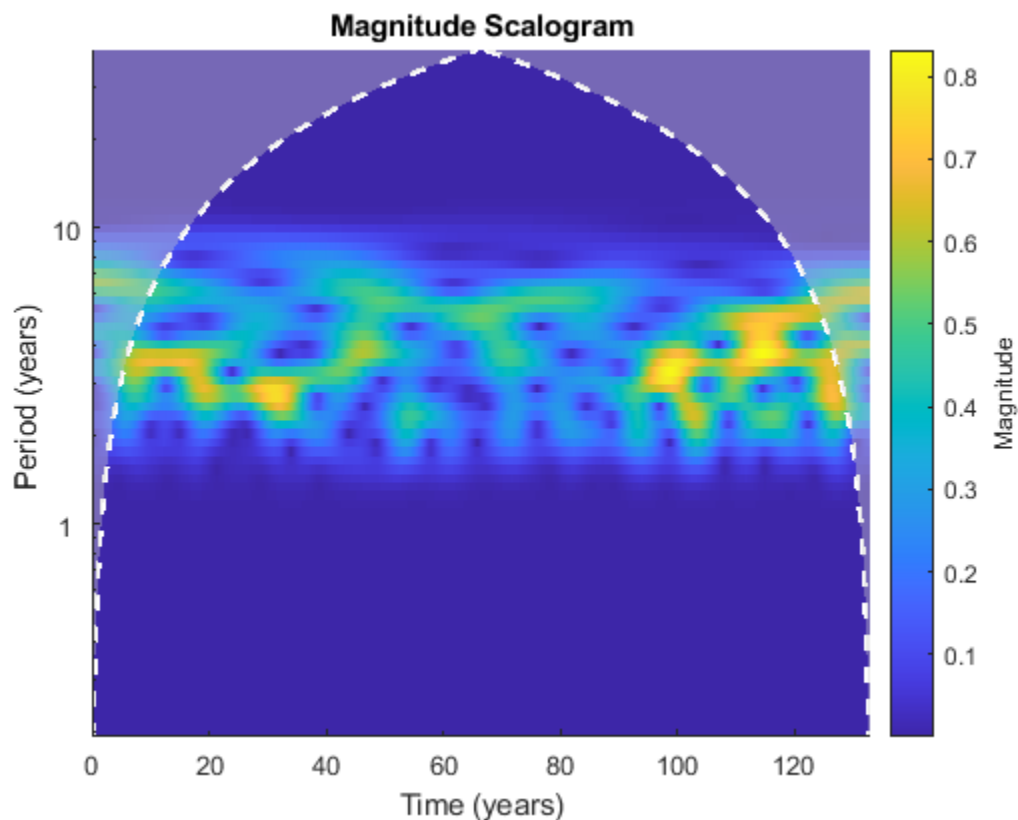

Compare the original data with the reconstructed data for years 2 through 8.

```
figure
subplot(2,1,1)
plot(nino)
grid on
title('Original Data')
subplot(2,1,2)
plot(xrec)
grid on
title('El Nino Data - Years 2-8')
```
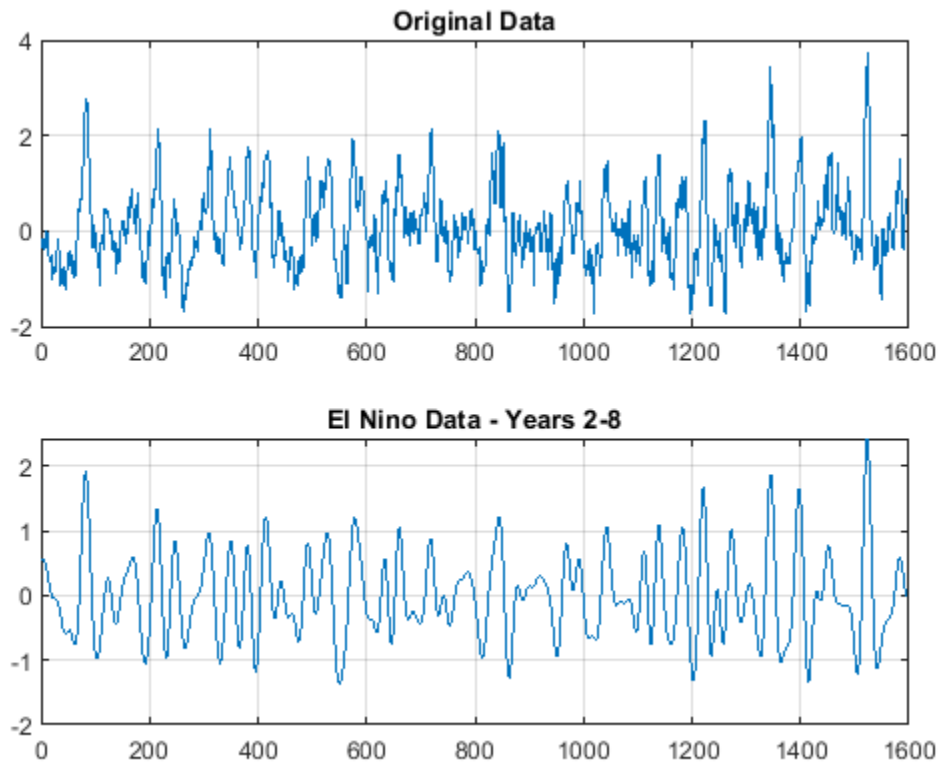
Compare Time-Frequency Content in Signals with Wavelet Coherence

This example shows how to use wavelet coherence and the wavelet cross-spectrum to identify time-localized common oscillatory behavior in two time series. The example also compares the wavelet coherence and cross-spectrum against their Fourier counterparts. You must have Signal Processing Toolbox™ to run the examples using `mscohere` (Signal Processing Toolbox) and `cpsd` (Signal Processing Toolbox).

Many applications involve identifying and characterizing common patterns in two time series. In some situations, common behavior in two time series results from one time series driving or influencing the other. In other situations, the common patterns result from some unobserved mechanism influencing both time series.

For jointly stationary time series, the standard techniques for characterizing correlated behavior in time or frequency are cross-correlation, the (Fourier) cross-spectrum, and coherence. However, many time series are nonstationary, meaning that their frequency content changes over time. For these time series, it is important to have a measure of correlation or coherence in the time-frequency plane.

You can use wavelet coherence to detect common time-localized oscillations in nonstationary signals. In situations where it is natural to view one time series as influencing another, you can use the phase of the wavelet cross-spectrum to identify the relative lag between the two time series.

Locate Common Time-Localized Oscillations and Determine Phase Lag

For the first example, use two signals consisting of time-localized oscillations at 10 and 75 Hz. The signals are six seconds in duration and are sampled at 1000 Hz. The 10-Hz oscillation in the two signals overlaps between 1.2 and 3 seconds. The overlap for the 75-Hz oscillation occurs between 0.4 and 4.4 seconds. The 10 and 75-Hz components are delayed 1/4 of a cycle in the Y-signal. This means there is a $\pi/2$ (90 degree) phase lag between the 10 and 75-Hz components in the two signals. Both signals are corrupted by additive white Gaussian noise.

```
load wcoherdemosig1
subplot(2,1,1)
plot(t,x1)
title('X Signal')
grid on
ylabel('Amplitude')
subplot(2,1,2)
plot(t,y1)
title('Y Signal')
ylabel('Amplitude')
grid on
xlabel('Seconds')
```
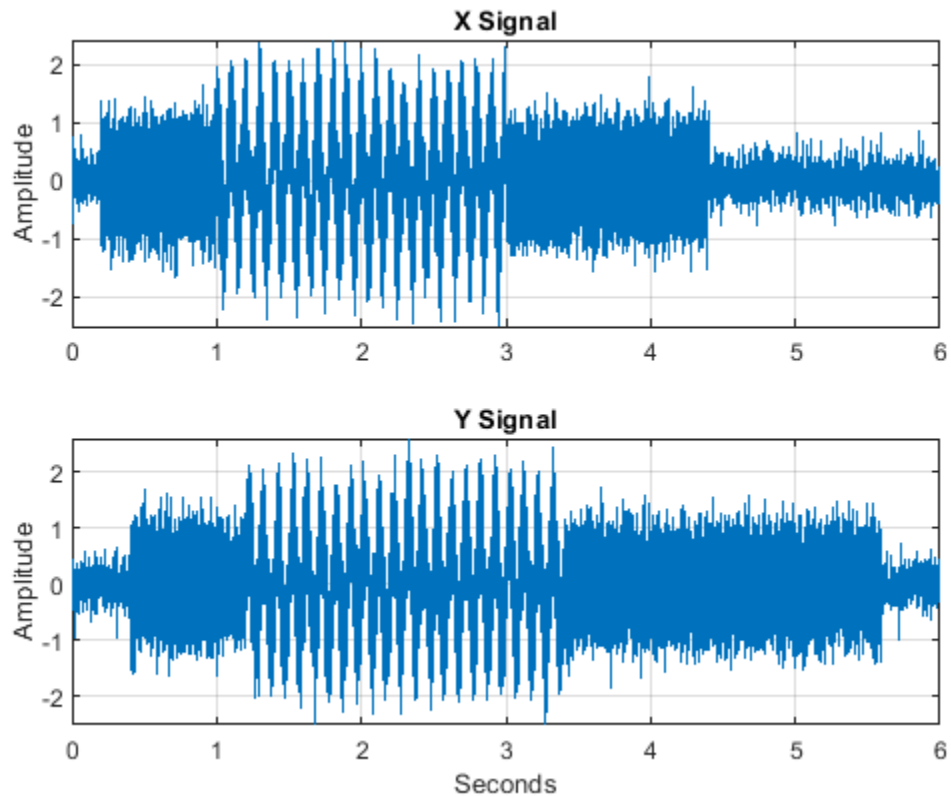
Obtain the wavelet coherence and display the result. Enter the sampling frequency (1000 Hz) to obtain a time-frequency plot of the wavelet coherence. In regions of the time-frequency plane where coherence exceeds 0.5, the phase from the wavelet cross-spectrum is used to indicate the relative lag between coherent components. Phase is indicated by arrows oriented in a particular direction. Note that a 1/4 cycle lag in the Y-signal at a particular frequency is indicated by an arrow pointing vertically.

```
figure
wcoherence(x1,y1,1000)
```
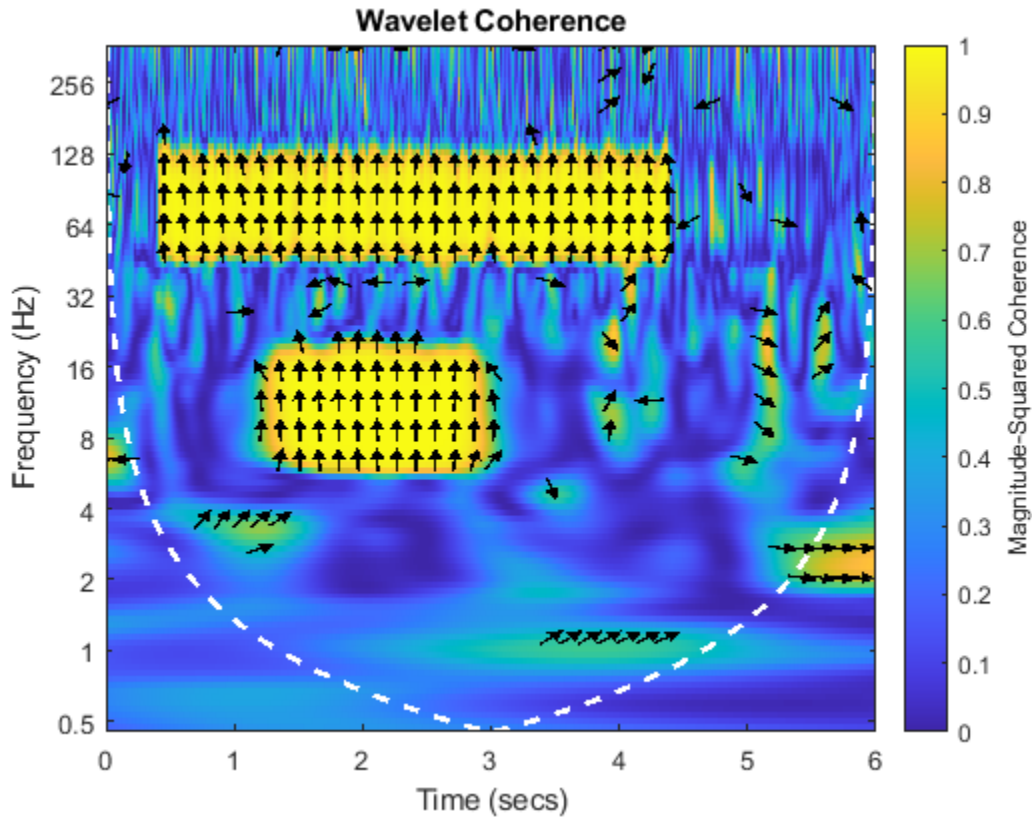
The two time-localized regions of coherent oscillatory behavior at 10 and 75 Hz are evident in the plot of the wavelet coherence. The phase relationship is shown by the orientation of the arrows in the regions of high coherence. In this example, you see that the wavelet cross-spectrum captures the $\pi/2$ (1/4 cycle) phase lag between the two signals at 10 and 75 Hz.

The white dashed line shows the cone of influence where edge effects become significant at different frequencies (scales). Areas of high coherence occurring outside or overlapping the cone of influence should be interpreted with caution.

Repeat the same analysis using the Fourier magnitude-squared coherence and cross-spectrum.

```
figure
mscohere(x1,y1,500,450,[],1000)
```
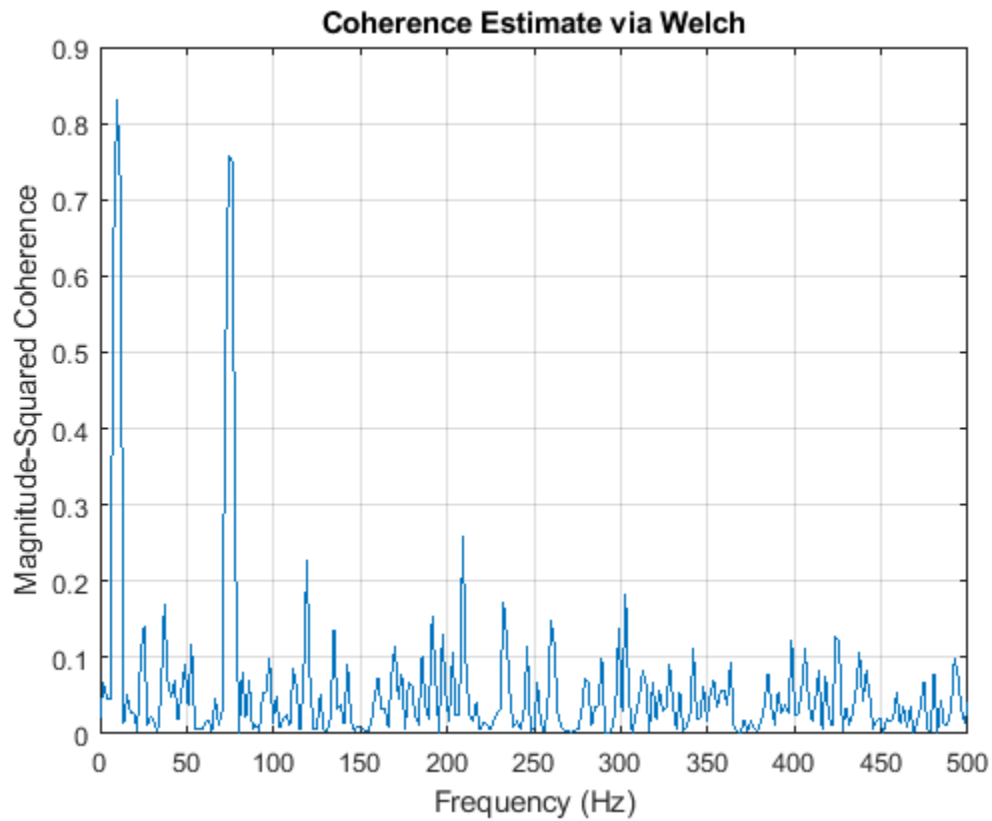
```
[Pxy,F] = cpsd(x1,y1,500,450,[],1000);
Phase = (180/pi)*angle(Pxy);
figure
plot(F,Phase)
title('Cross-Spectrum Phase')
xlabel('Hz')
ylabel('Phase (Degrees)')
grid on
ylim([0 180])
xlim([0 200])
hold on
plot([10 10],[0 180],'r--')
plot([75 75],[0 180],'r--')
plot(F,90*ones(size(F)),'r--')
hold off
```
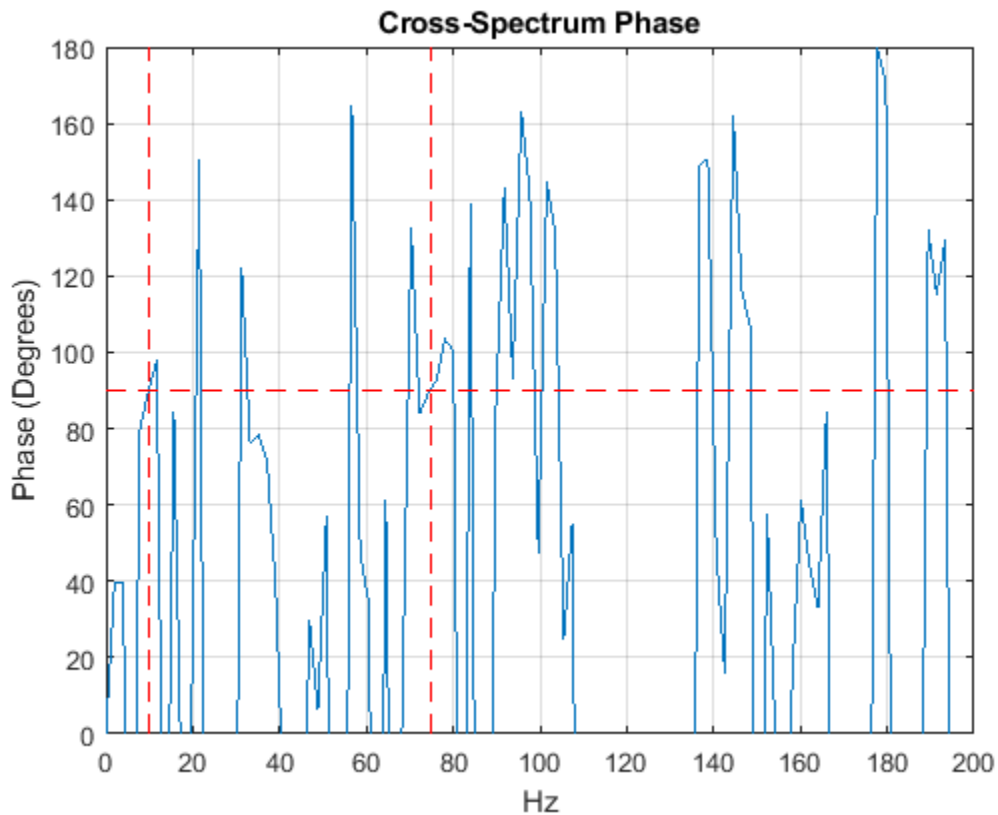
The Fourier magnitude-squared coherence obtained from `mscohere` (Signal Processing Toolbox) clearly identifies the coherent oscillations at 10 and 75 Hz. In the phase plot of the Fourier cross-spectrum, the vertical red dashed lines mark 10 and 75 Hz while the horizontal line marks an angle of 90 degrees. You see that the phase of the cross-spectrum does a reasonable job of capturing the relative phase lag between the components.

However, the time-dependent nature of the coherent behavior is completely obscured by these techniques. For nonstationary signals, characterizing coherent behavior in the time-frequency plane is much more informative.

The following example repeats the preceding one while changing the phase relationship between the two signals. In this case, the 10-Hz component in the Y-signal is delayed by 3/8 of a cycle ($3\pi/4$ radians). The 75-Hz component in the Y-signal is delayed by 1/8 of a cycle ($\pi/4$ radians). Plot the wavelet coherence and threshold the phase display to only show areas where the coherence exceeds 0.75.

```
load wcoherdemosig2
wcoherence(x2,y2,1000,'phasedisplaythreshold',0.75)
```
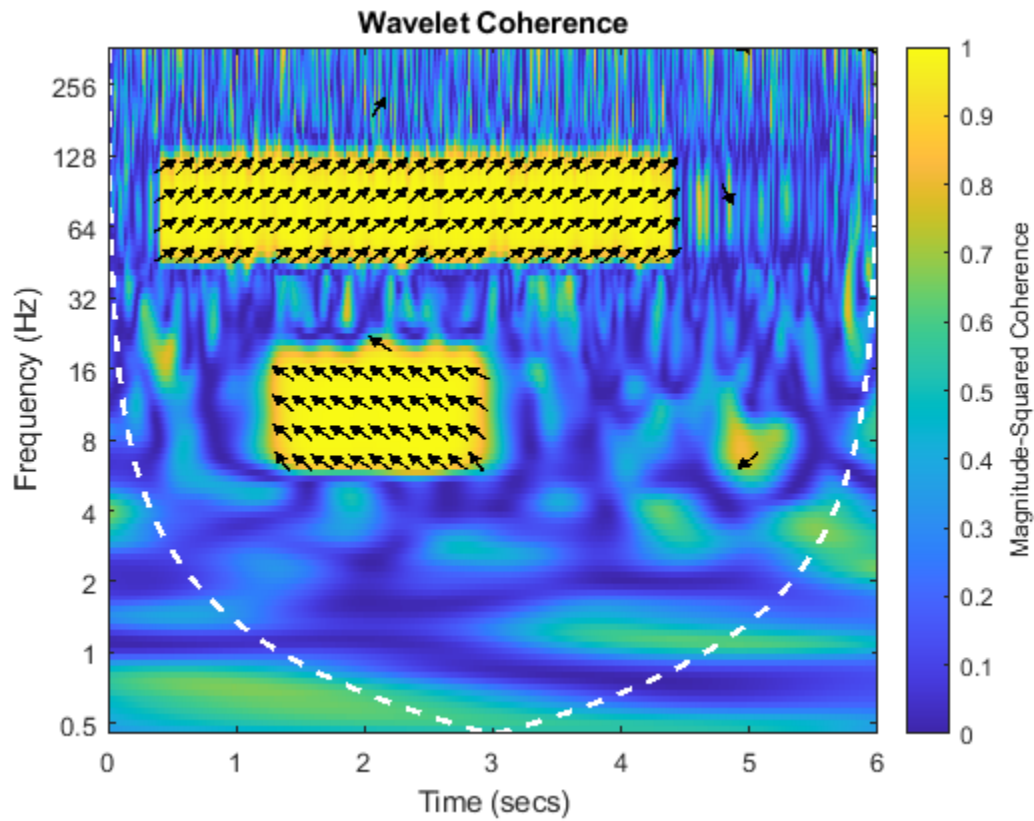
Phase estimates obtained from the wavelet cross-spectrum capture the relative lag between the two time series at 10 and 75 Hz.

If you prefer to view data in terms of periods rather than frequency, you can use a MATLAB duration object to provide wcoherence with a sample time.

```
wcoherence(x2,y2,seconds(0.001),'phasedisplaythreshold',0.75)
```
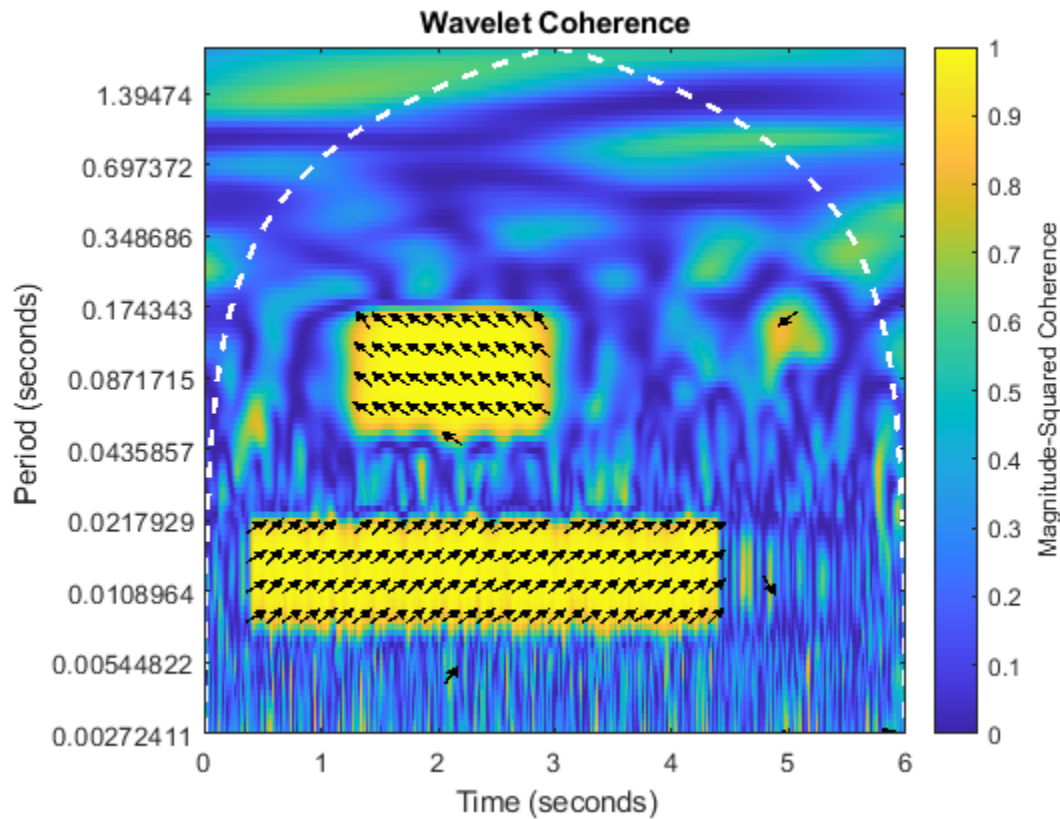
Note that cone of influence has inverted because the wavelet coherence is now plotted in terms of periods.

Determine Coherent Oscillations and Delay in Climate Data

Load and plot the El Nino Region 3 data and deseasonalized All Indian Rainfall Index from 1871 to late 2003. The data are sampled monthly. The Nino 3 time series is a record of monthly sea surface temperature anomalies in degrees Celsius recorded from the equatorial Pacific at longitudes 90 degrees west to 150 degrees west and latitudes 5 degrees north to 5 degrees south. The All Indian Rainfall Index represents average Indian rainfalls in millimeters with seasonal components removed.

```
load ninoairdata
figure
subplot(2,1,1)
plot(datayear,nino)
title('El Nino Region 3 -- SST Anomalies')
ylabel('Degrees Celsius')
axis tight
subplot(2,1,2)
plot(datayear,air)
axis tight
title('Deseasonalized All Indian Rainfall Index')
ylabel('Millimeters')
xlabel('Year')
```

El Nino Region 3 -- SST Anomalies

Deseasonalized All Indian Rainfall Index

Plot the wavelet coherence with phase estimates. For this data, it is more natural to look at oscillations in terms of their periods in years. Enter the sampling interval (period) as a duration object with units of years so that the output periods are in years. Show the relative lag between the two climate time series only where the magnitude-squared coherence exceeds 0.7.

```
figure
wcoherence(nino,air,years(1/12),'phasedisplaythreshold',0.7)
```
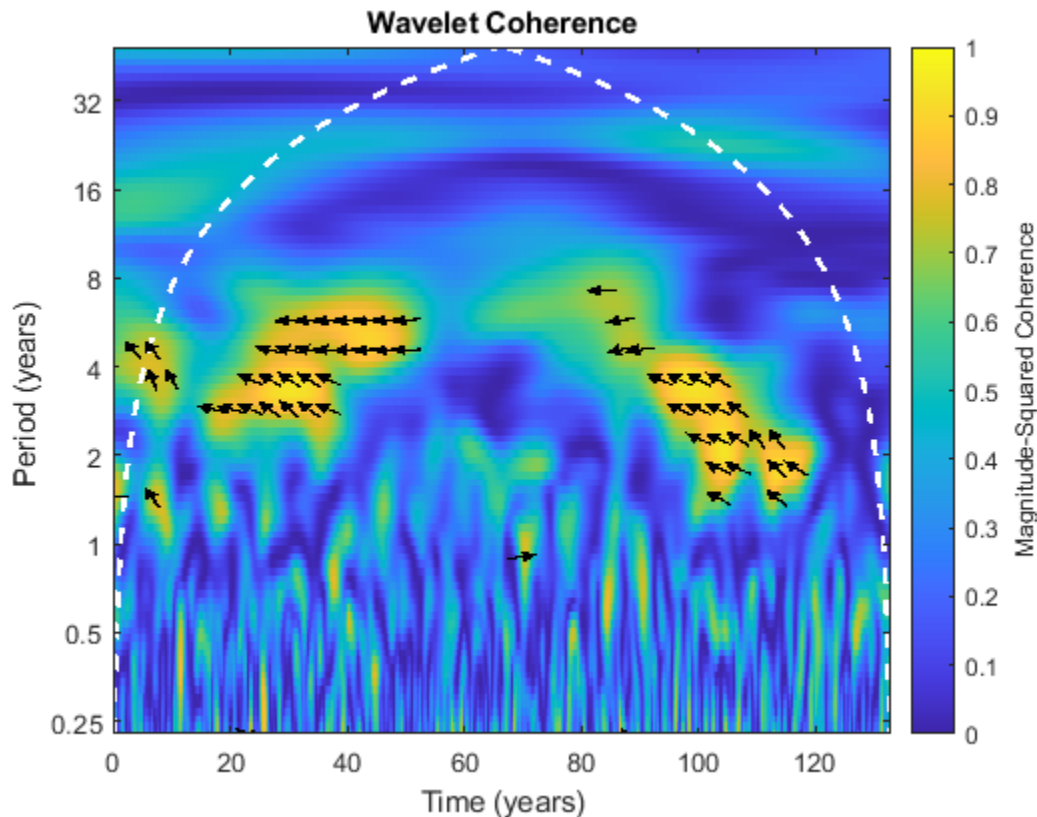
The plot shows time-localized areas of strong coherence occurring in periods that correspond to the typical El Nino cycles of 2 to 7 years. The plot also shows that there is an approximate 3/8-to-1/2 cycle delay between the two time series at those periods. This indicates that periods of sea warming consistent with El Nino recorded off the coast of South America are correlated with rainfall amounts in India approximately 17,000 km away, but that this effect is delayed by approximately 1/2 a cycle (1 to 3.5 years).

Find Coherent Oscillations in Brain Activity

In the previous examples, it was natural to view one time series as influencing the other. In these cases, examining the lead-lag relationship between the data is informative. In other cases, it is more natural to examine the coherence alone.

For an example, consider 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 data is taken from Cui, Bryant, & Reiss [1] and was kindly provided by the authors for this example. The recording site was the superior frontal cortex for both subjects. The data is sampled at 10 Hz. 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')
```

```matlab
xlabel('Seconds')
title('NIRS Data')
grid on
hold off
```

NIRS Data

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`.

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

Wavelet Coherence

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 as a duration object. `wcoherence` provides scale-to-period conversions.

```
[wcoh,~,P,coi] = wcoherence(NIRSData(:,1),NIRSData(:,2),seconds(0.1),...
    'numscales',16);
helperPlotCoherence(wcoh,tm,seconds(P),seconds(coi),...
    'Time (secs)','Periods (Seconds)')
```
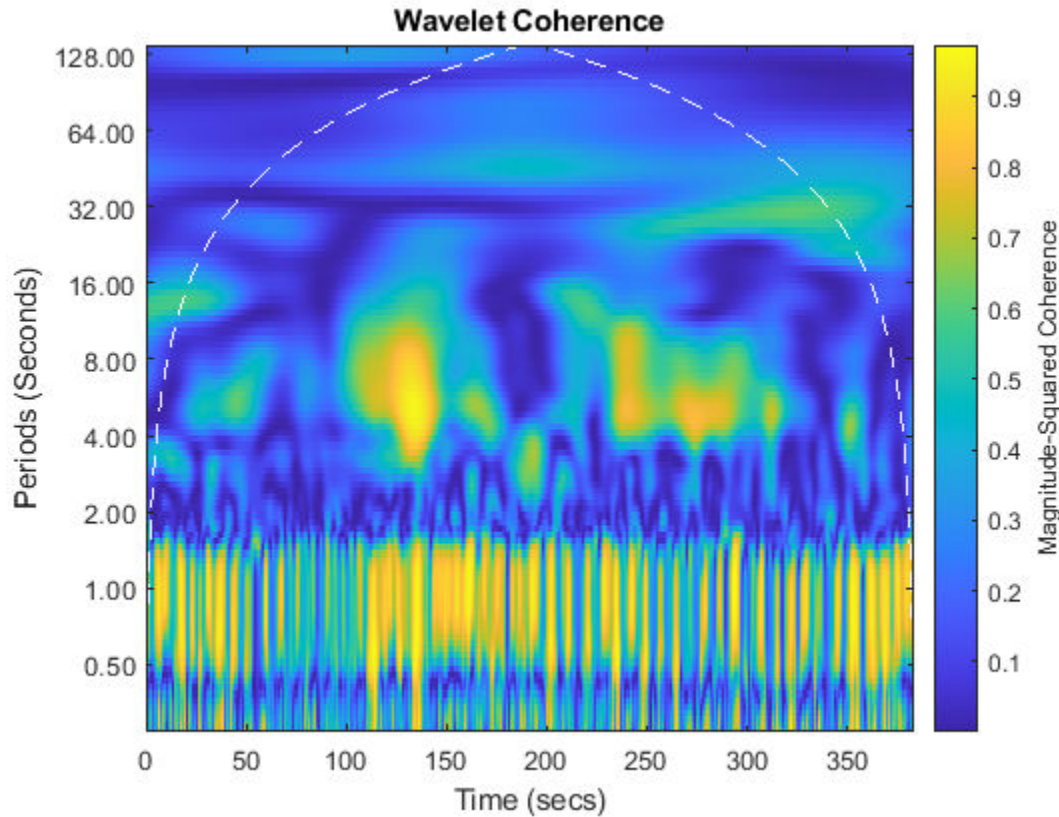
Again, note the coherent oscillations corresponding to the subjects' cardiac activity occurring throughout the recordings with a period of approximately one second. The task-related activity is also apparent with a period of approximately 8 seconds. Consult Cui, Bryant, & Reiss [1] for a more detailed wavelet analysis of this data.

Conclusions

In this example you learned how to use wavelet coherence to look for time-localized coherent oscillatory behavior in two time series. For nonstationary signals, it is often more informative if you have a measure of coherence that provides simultaneous time and frequency (period) information. The relative phase information obtained from the wavelet cross-spectrum can be informative when one time series directly affects oscillations in the other.

Appendix

The following helper function is used in this example.

- helperPlotCoherence

References

[1] 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.

[2] Grinsted, A., J. C. Moore, and S. Jevrejeva. "Application of the cross wavelet transform and wavelet coherence to geophysical time series." *Nonlinear Processes in Geophysics*. Vol. 11, 2004, pp. 561–566. doi:10.5194/npg-11-561-2004.

[3] Maraun, D., J. Kurths, and M. Holschneider. "Nonstationary Gaussian processes in wavelet domain: synthesis, estimation, and significance testing." *Physical Review E*. Vol. 75, 2007, pp. 016707(1)–016707(14). doi:10.1103/PhysRevE.75.016707.

[4] Torrence, C., and P. Webster. "Interdecadal Changes in the ENSO-Monsoon System." *Journal of Climate*. Vol. 12, Number 8, 1999, pp. 2679–2690. doi:10.1175/1520-0442(1999)012<2679:ICITEM>2.0.CO;2.

See Also
wcoherence

Continuous and Discrete Wavelet Analysis of Frequency Break

This example shows the difference between the discrete wavelet transform (**DWT**) and the continuous wavelet transform (**CWT**).

When is Continuous Analysis More Appropriate than Discrete Analysis?

To answer this, consider the related questions: Do you need to know all values of a continuous decomposition to reconstruct the signal exactly? Can you perform nonredundant analysis? When the energy of the signal is finite, not all values of a decomposition are needed to exactly reconstruct the original signal, provided that you are using a wavelet that satisfies some admissibility condition. Usual wavelets satisfy this condition. In this case, a continuous-time signal is characterized by the knowledge of the discrete transform. In such cases, discrete analysis is sufficient and continuous analysis is redundant.

Continuous analysis is often easier to interpret, since its redundancy tends to reinforce the traits and makes all information more visible. This is especially true of very subtle information. Thus, the analysis gains in "readability" and in ease of interpretation what it loses in terms of saving space.

DWT and CWT of a Signal with a Frequency Break

Show how analysis using wavelets can detect the exact instant when a signal changes. Use a discontinuous signal that consists of a slow sine wave abruptly followed by a medium sine wave.

```
load freqbrk;
signal = freqbrk;
```

Perform the discrete wavelet transform (DWT) at level 5 using the Haar wavelet.

```
lev   = 5;
wname = 'db1';
nbcol = 64;
[c,l] = wavedec(signal,lev,wname);
```

Expand discrete wavelet coefficients for plot.

```
len = length(signal);
cfd = zeros(lev,len);
for k = 1:lev
    d = detcoef(c,l,k);
    d = d(:)';
    d = d(ones(1,2^k),:);
    cfd(k,:) = wkeep(d(:)',len);
end
cfd =  cfd(:);
I = find(abs(cfd)<sqrt(eps));
cfd(I) = zeros(size(I));
cfd    = reshape(cfd,lev,len);
cfd = wcodemat(cfd,nbcol,'row');

h211 = subplot(2,1,1);
h211.XTick = [];
plot(signal,'r');
title('Analyzed signal.');
ax = gca;
ax.XLim = [1 length(signal)];
subplot(2,1,2);
```

```
colormap(cool(128));
image(cfd);
tics = 1:lev;
labs = int2str(tics');
ax = gca;
ax.YTickLabelMode = 'manual';
ax.YDir = 'normal';
ax.Box = 'On';
ax.YTick = tics;
ax.YTickLabel = labs;
title('Discrete Transform, absolute coefficients.');
ylabel('Level');
```
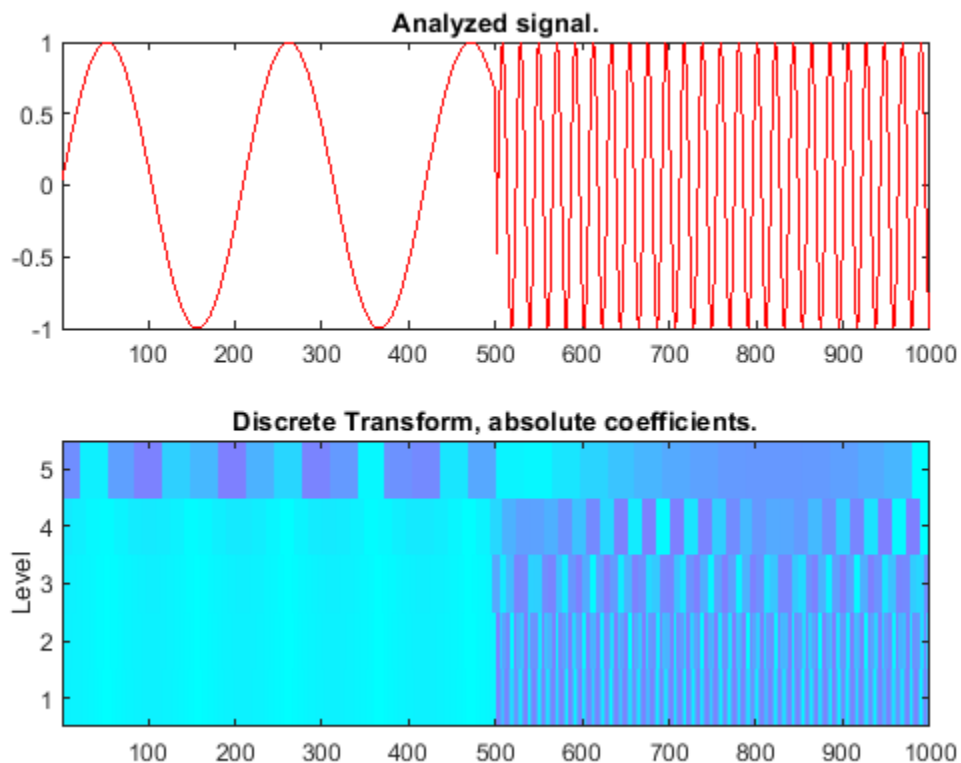

Perform the continuous wavelet transform (CWT) and visualize results

```
figure;
[cfs,f] = cwt(signal,1,'waveletparameters',[3 3.1]);
hp = pcolor(1:length(signal),f,abs(cfs)); hp.EdgeColor = 'none';
set(gca,'YScale','log');
xlabel('Sample'); ylabel('log10(f)');
```
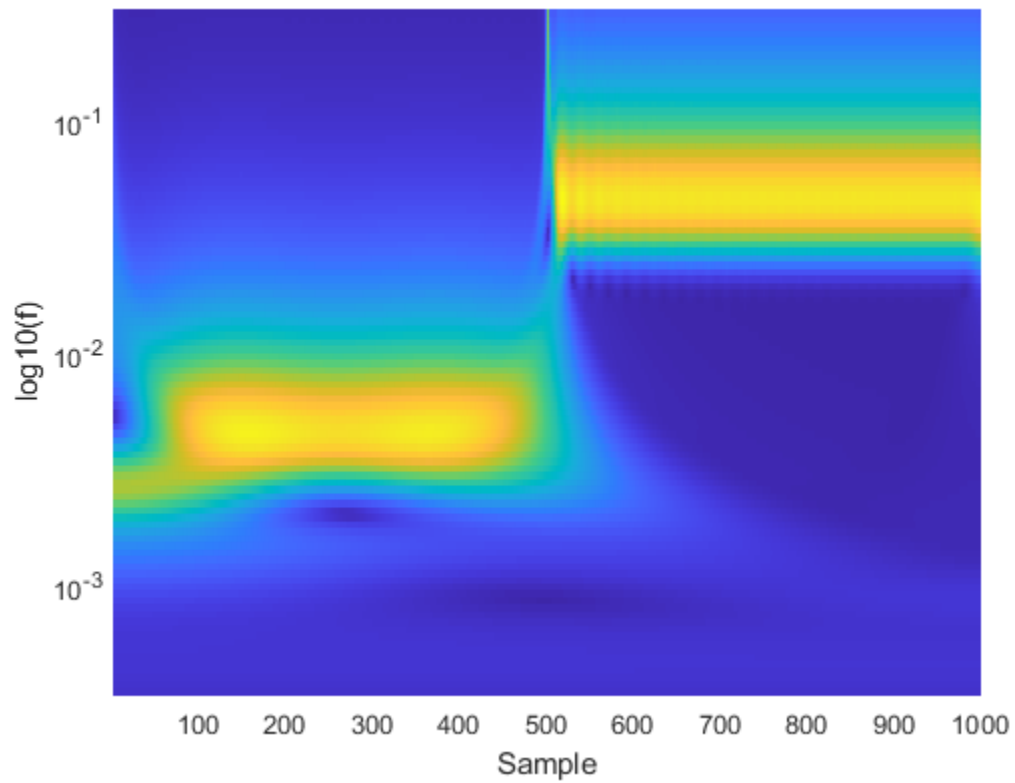
If you just look at the finest scale CWT coefficients, you can localize the frequency change precisely.

```
plot(abs(cfs(1,:))); grid on;
```
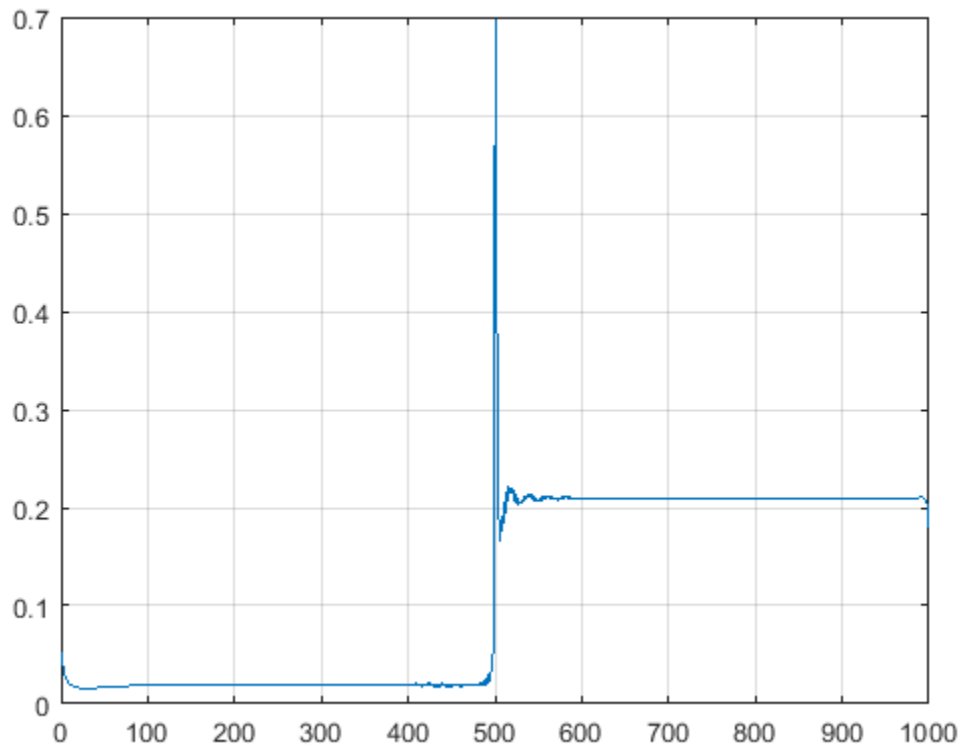
This example shows an important advantage of wavelet analysis over Fourier. If the same signal had been analyzed by the Fourier transform, we would not have been able to detect the instant when the signal's frequency changed, whereas it is clearly observable here.

Wavelet Packets: Decomposing the Details

This example shows how wavelet packets differ from the discrete wavelet transform (DWT). The example shows how the wavelet packet transform results in equal-width subband filtering of signals as opposed to the coarser octave band filtering found in the DWT.

This makes wavelet packets an attractive alternative to the DWT in a number of applications. Two examples presented here are time-frequency analysis and signal classification. You must have Statistics and Machine Learning Toolbox™ and Signal Processing Toolbox™ to perform the classification.

If you use an orthogonal wavelet with the wavelet packet transform, you additionally end up with a partitioning of the signal energy among the equal-width subbands.

The example focuses on the 1-D case, but many of the concepts extend directly to the wavelet packet transform of images.

Discrete Wavelet and Discrete Wavelet Packet Transforms

The following figure shows a DWT tree for a 1-D input signal.

Discrete Wavelet Transform Tree
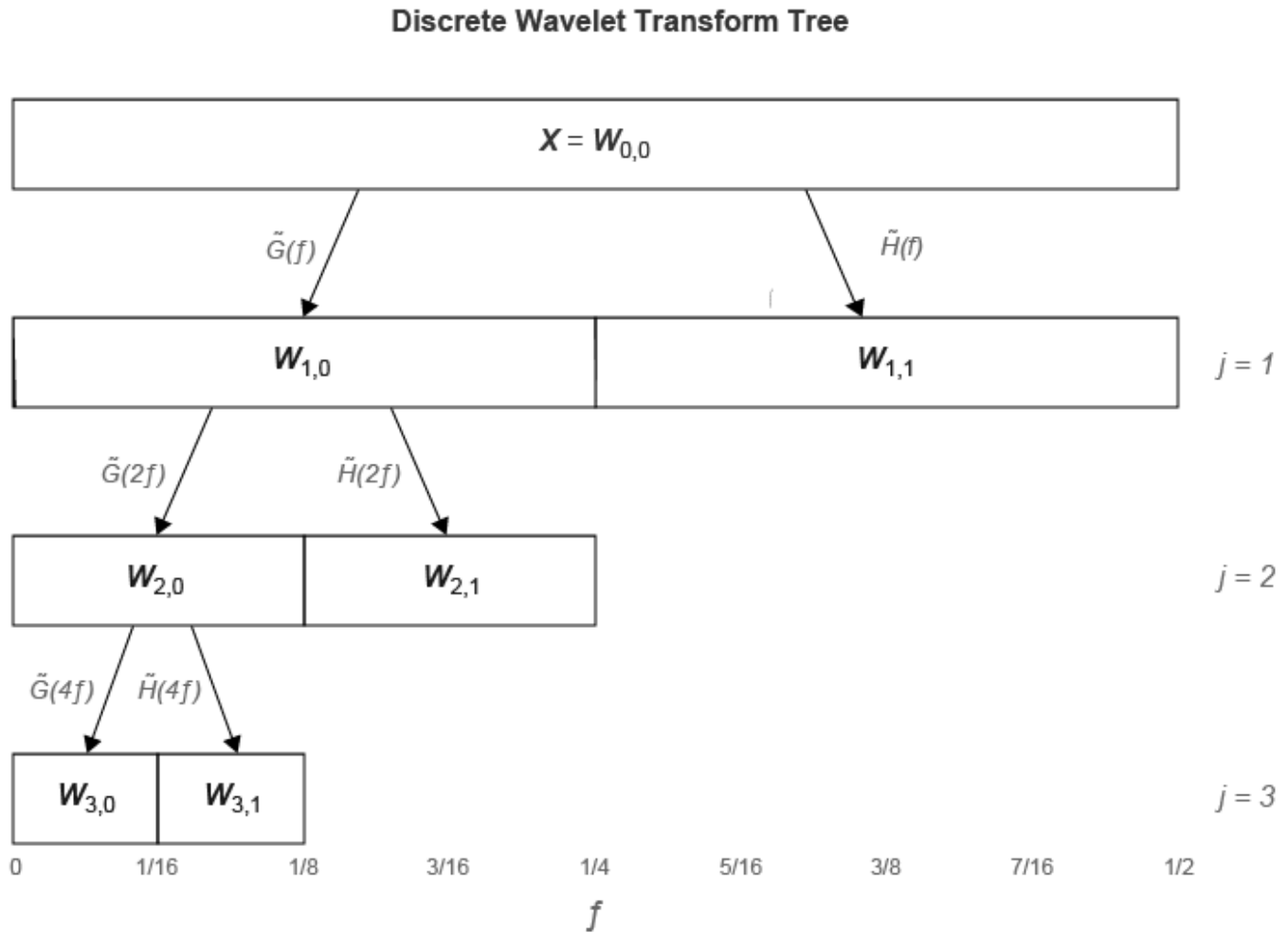
Figure 1: DWT Tree down to level 3 for a 1-D input signal.

$\widetilde{G}(f)$ is the scaling (lowpass) analysis filter and $\widetilde{H}(f)$ represents the wavelet (highpass) analysis filter. The labels at the bottom show the partition of the frequency axis $[0,1/2]$ into subbands.

The figure shows that subsequent levels of the DWT operate only on the outputs of the lowpass (scaling) filter. At each level, the DWT divides the signal into octave bands. In the critically-sampled DWT, the outputs of the bandpass filters are downsampled by two at each level. In the undecimated discrete wavelet transform, the outputs are not downsampled.

Compare the DWT tree with the full wavelet packet tree.

Natural-Ordered Wavelet Packet Tree
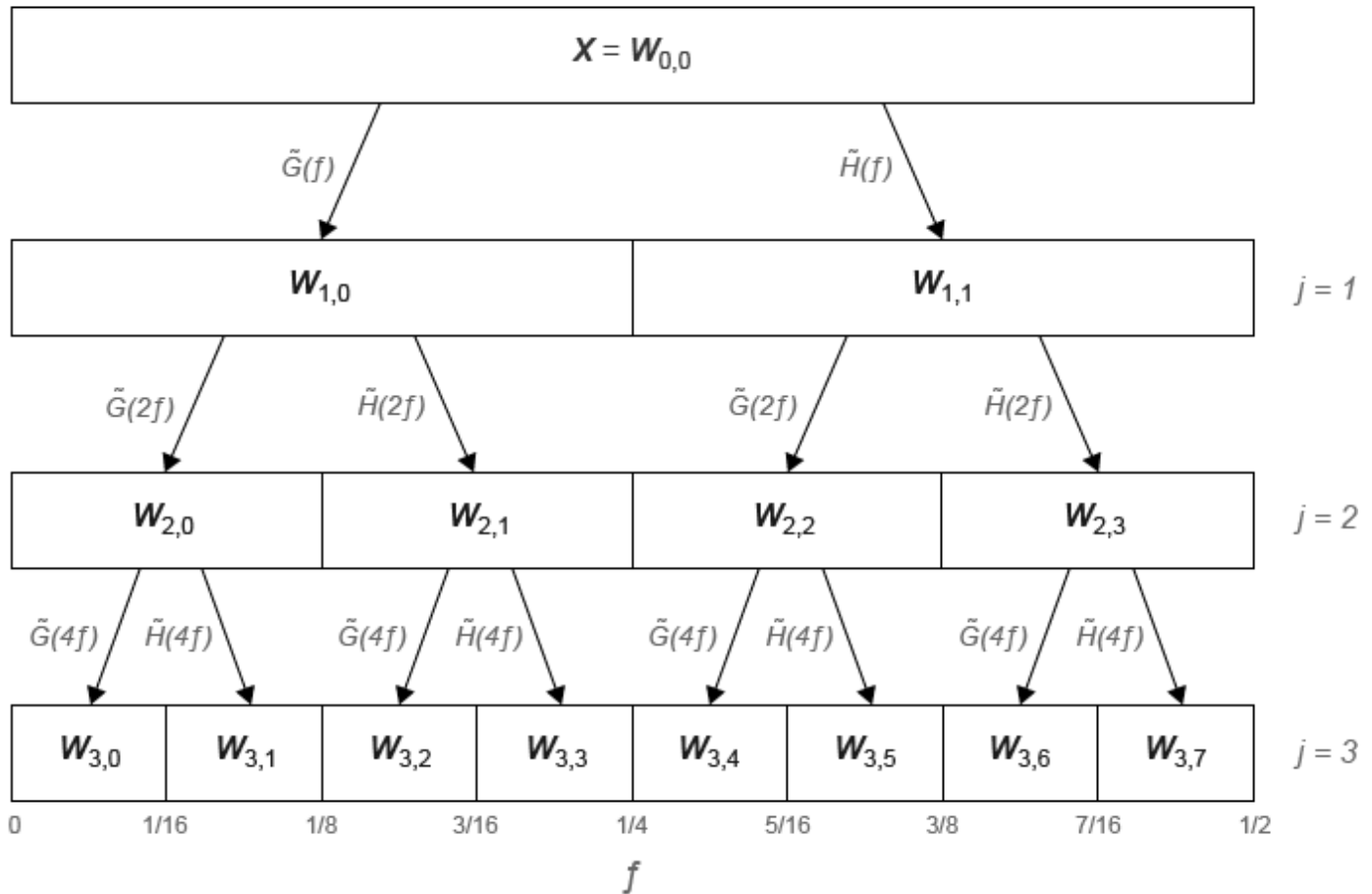
Figure 2: Full wavelet packet tree down to level 3.

In the wavelet packet transform, the filtering operations are also applied to the wavelet, or detail, coefficients. The result is that wavelet packets provide a subband filtering of the input signal into progressively finer equal-width intervals. At each level, j, the frequency axis $[0,1/2]$ is divided into 2^j subbands. The subbands in hertz at level j are approximately

$$[\frac{n\text{Fs}}{2^{j+1}}, \frac{(n+1)\text{Fs}}{2^{j+1}}) \quad n = 0, 1, ...2^j - 1$$ where Fs is the sampling frequency.

How good this bandpass approximation is depends on how frequency-localized the filters are. For Fejér-Korovkin filters like `'fk18'` (18 coefficients), the approximation is quite good. For a filter like the Haar (`'haar'`), the approximation is not accurate.

In the critically-sampled wavelet packet transform the outputs of the bandpass filters are downsampled by two. In the undecimated wavelet packet transform, the outputs are not downsampled.

Time-Frequency Analysis with Wavelet Packets

Because wavelet packets divide the frequency axis into finer intervals than the DWT, wavelet packets are superior at time-frequency analysis. As an example, consider two intermittent sine waves with frequencies of 150 and 200 Hz in additive noise. The data are sampled at 1 kHz. To prevent the loss of time resolution inherent in the critically-sampled wavelet packet transform, use the undecimated transform.

```
dt = 0.001;
t = 0:dt:1-dt;
x = ...
cos(2*pi*150*t).*(t>=0.2 & t<0.4)+sin(2*pi*200*t).*(t>0.6 & t<0.9);
y = x+0.05*randn(size(t));
[wpt,~,F] = modwpt(x,'TimeAlign',true);
contour(t,F.*(1/dt),abs(wpt).^2)
grid on
xlabel('Time (secs)')
ylabel('Hz')
title('Time-Frequency Analysis -- Undecimated Wavelet Packet Transform')
```
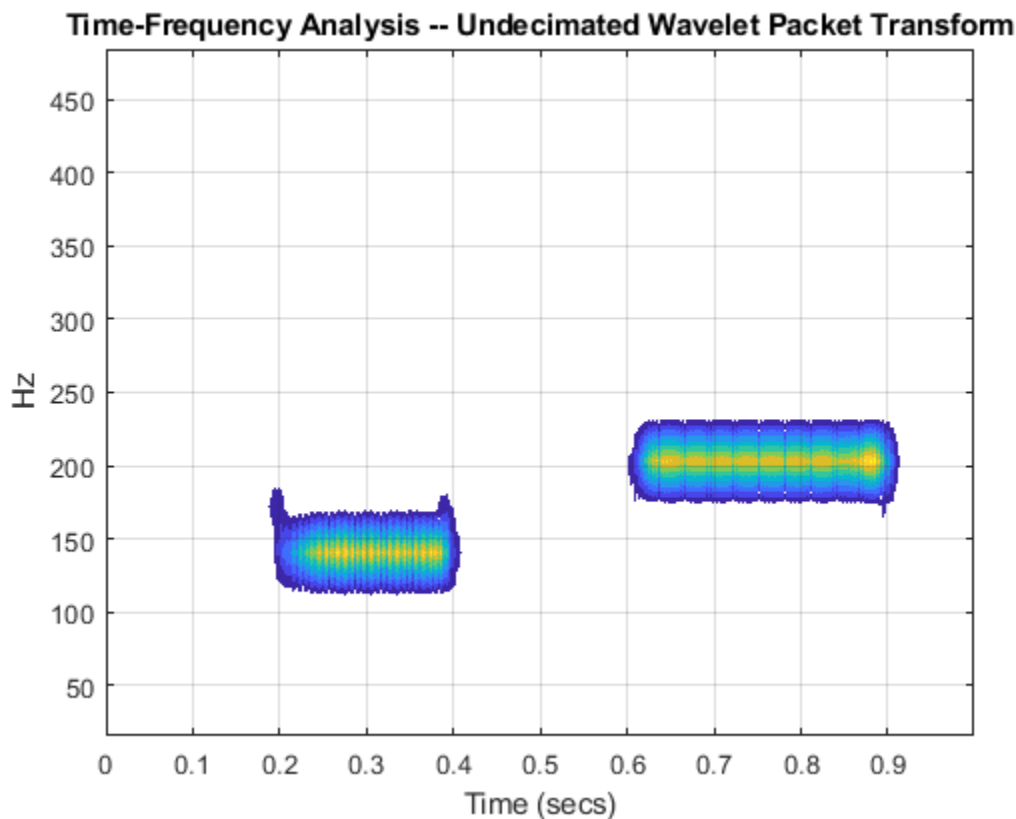

Note that the wavelet packet transform is able to separate the 150 and 200 Hz components. This is not true of the DWT, because 150 and 200 Hz fall in the same octave band. The octave bands for a level-four DWT are (in Hz)

- [0,31.25)
- [31.25,62.5)

- [62.5,125)
- [125,250)
- [250,500)

Demonstrate that these two components are time-localized by the DWT but not separated in frequency.

```
mra  = modwtmra(modwt(y,'fk18',4),'fk18');
J = 5:-1:1;
freq = 1/2*(1000./2.^J);
contour(t,freq,flipud(abs(mra).^2))
grid on
ylim([0 500])
xlabel('Time (secs)')
ylabel('Hz')
title('Time-Frequency Analysis -- Undecimated Wavelet Transform')
```
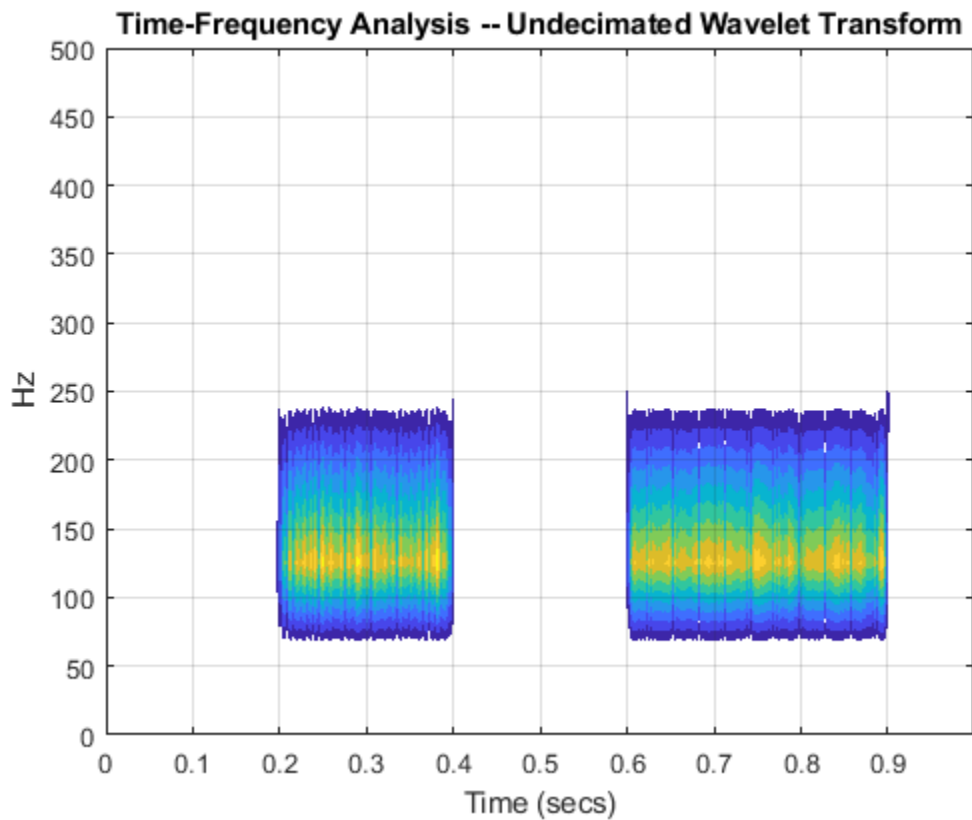

Of course, the continuous wavelet transform (CWT) provides a higher resolution time-frequency analysis than the wavelet packet transform, but wavelet packets have the added benefit of being an orthogonal transform (when using an orthogonal wavelet). An orthogonal transform means that the energy in the signal is preserved and partitioned among the coefficients as the next section demonstrates.

Energy Preservation in the Wavelet Packet Transform

In addition to filtering a signal into equal-width subbands at each level, the wavelet packet transform partitions the signal's energy among the subbands. This is true for both the decimated and

undecimated wavelet packet transforms. To demonstrate this, load a dataset containing a seismic recording of an earthquake. This data is similar to the time series used in the signal classification example below.

```
load kobe
plot(kobe)
grid on
xlabel('Seconds')
title('Kobe Earthquake Data')
axis tight
```
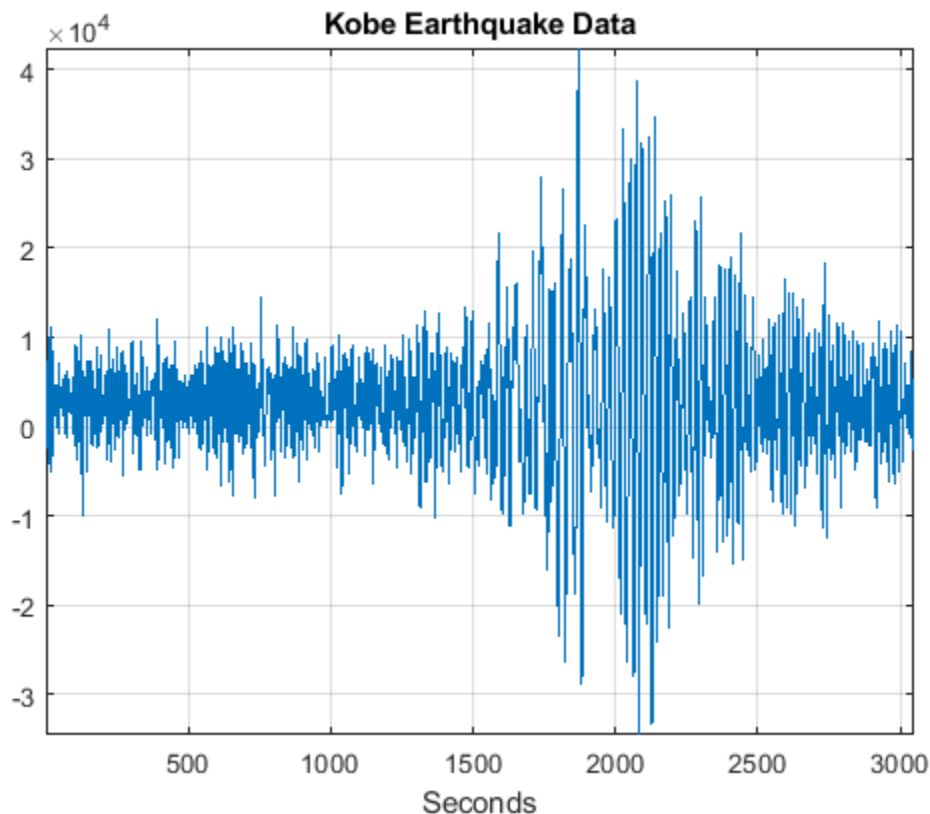

Obtain both the decimated and undecimated wavelet packet transforms of the data down to level 3. To ensure consistent results for the decimated wavelet packet transform, set the boundary extension mode to `'periodic'`.

```
wptreeDecimated = dwpt(kobe,'Level',3,'Boundary','periodic');
wptUndecimated = modwpt(kobe,3);
```

Compute the total energy in both the decimated and undecimated wavelet packet level-three coefficients and compare to the energy in the original signal.

```
decimatedEnergy = sum(cell2mat(cellfun(@(x) sum(abs(x).^2),wptreeDecimated,'UniformOutput',false)
```

```
decimatedEnergy = 2.0189e+11
```

```
undecimatedEnergy = sum(sum(abs(wptUndecimated).^2,2))
```

```
undecimatedEnergy = 2.0189e+11
```

```
signalEnergy = norm(kobe,2)^2
```

```
signalEnergy = 2.0189e+11
```

The DWT shares this important property with the wavelet packet transform.

```
wt = modwt(kobe,'fk18',3);
undecimatedWTEnergy = sum(sum(abs(wt).^2,2))
```

```
undecimatedWTEnergy = 2.0189e+11
```

Because the wavelet packet transform at each level divides the frequency axis into equal-width intervals and partitions the signal energy among those intervals, you can often construct useful feature vectors for classification just by retaining the relative energy in each wavelet packet. The next example demonstrates this.

Wavelet Packet Classification -- Earthquake or Explosion?

Seismic recordings pick up activity from many sources. It is important to be able to classify this activity based on its origin. For example, earthquakes and explosions may present similar time-domain signatures, but it is very important to differentiate between these two events. The dataset `EarthQuakeExplosionData` contains 16 recordings with 2048 samples. The first 8 recordings (columns) are seismic recordings of earthquakes, the last 8 recordings (columns) are seismic recordings of explosions. Load the data and plot an earthquake and explosion recording for comparison. The data is taken from the R package astsa Stoffer [4], which supplements Shumway and Stoffer [3]. The author has kindly allowed its use in this example.

Plot a time series from each group for comparison.

```
load EarthQuakeExplosionData
subplot(2,1,1)
plot(EarthQuakeExplosionData(:,3))
xlabel('Time')
title('Earthquake Recording')
grid on
axis tight
subplot(2,1,2)
plot(EarthQuakeExplosionData(:,9))
xlabel('Time')
title('Explosion Recording')
grid on
axis tight
```

Earthquake Recording

Explosion Recording
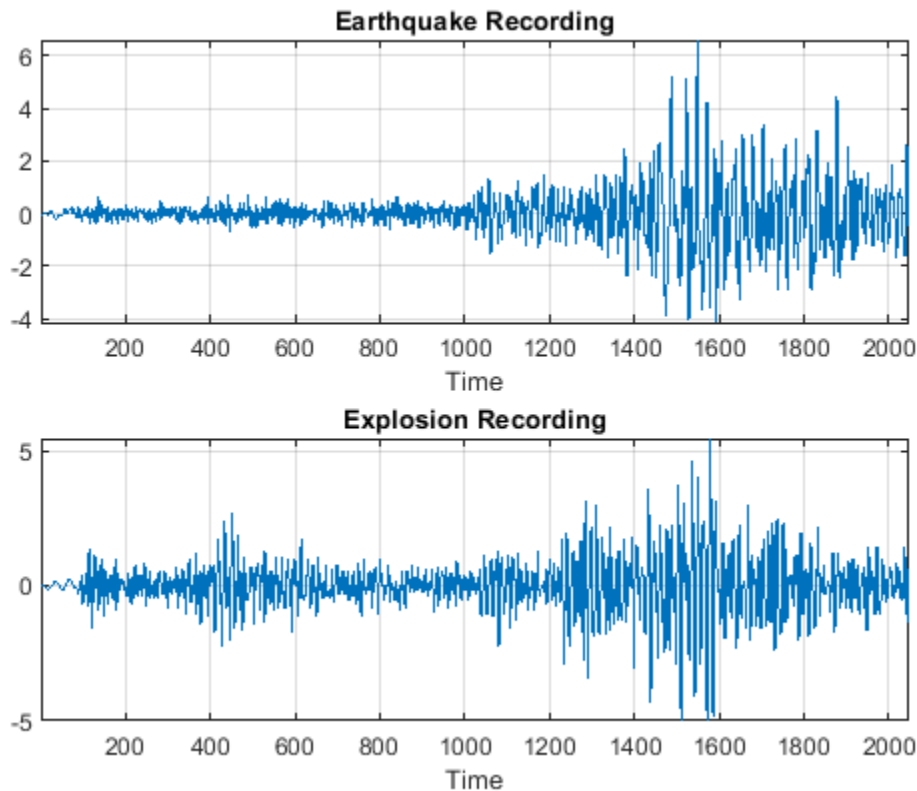
Form a wavelet packet feature vector by decomposing each time series down to level three using the `'fk6'` wavelet with an undecimated wavelet packet transform. This results in 8 subbands with an approximate width of 1/16 cycles/sample. Use the relative energy in each subband to create a feature vector. As an example, obtain a wavelet packet relative energy feature vector for the first earthquake recording.

```
[wpt,~,F,E,RE] = modwpt(EarthQuakeExplosionData(:,1),3,'fk6');
```

RE contains the relative energy in each of the 8 subbands. If you sum all the elements in RE, it is equal to 1. Note that this is a significant reduction in data. A time series of length 2048 is reduced to a vector with 8 elements, each element representing the relative energy in the wavelet packet nodes at level 3. The helper function `helperEarthQuakeExplosionClassifier` obtains the relative energies for each of the 16 recordings at level 3 using the `'fk6'` wavelet. This results in a 16-by-8 matrix and uses these feature vectors as inputs to a kmeans classifier. The classifier uses the Gap statistic criterion to determine the optimal number of clusters for the feature vectors between 1 and 6 and classifies each vector. Additionally, the classifier performs the exact same classification on the undecimated wavelet transform coefficients at level 3 obtained with the `'fk6'` wavelet and power spectra for each of the time series. The undecimated wavelet transform results in a 16-by-4 matrix (3 wavelet subbands and 1 scaling subband). The power spectra result in a 16-by-1025 matrix. The source code for `helperEarthQuakeExplosionClassifier` is listed in the appendix. You must have Statistics and Machine Learning Toolbox™ and Signal Processing Toolbox™ to run the classifier.

```
Level = 3;
[WavPacketCluster,WtCluster,PspecCluster] = ...
helperEarthQuakeExplosionClassifier(EarthQuakeExplosionData,Level)
```

```
WavPacketCluster =
  GapEvaluation with properties:

    NumObservations: 16
         InspectedK: [1 2 3 4 5 6]
    CriterionValues: [-0.0039 0.0779 0.1314 0.0862 0.0296 -0.0096]
           OptimalK: 2


WtCluster =
  GapEvaluation with properties:

    NumObservations: 16
         InspectedK: [1 2 3 4 5 6]
    CriterionValues: [-0.0095 0.0474 0.0706 0.0260 -0.0280 -0.0346]
           OptimalK: 1


PspecCluster =
  GapEvaluation with properties:

    NumObservations: 16
         InspectedK: [1 2 3 4 5 6]
    CriterionValues: [0.3804 0.3574 0.3466 0.2879 0.3256 0.3326]
           OptimalK: 1
```

`WavPacketCluster` is the clustering output for the undecimated wavelet packet feature vectors.
`WtCluster` is the clustering output using the undecimated DWT feature vectors, and `PspecCluster`
is the output for the clustering based on the power spectra. The wavelet packet classification has
identified two clusters (`OptimalK: 2`) as the optimal number. Examine the results of the wavelet
packet clustering.

```
res = [ WavPacketCluster.OptimalY(1:8)' ; WavPacketCluster.OptimalY(9:end)']
```

res = *2×8*

```
    1     2     2     2     2     2     2     2
    1     1     1     1     1     1     1     2
```

You see that only two errors have been made. Of the eight earthquake recordings, seven are classified
together (group 2) and one is mistakenly classified as belonging to group 1. The same is true of the 8
explosion recordings -- 7 are classified together and 1 is misclassified. Classification based on the
level-three undecimated DWT and power spectra return one cluster as the optimal solution.

If you repeat the above with the level equal to 4, the undecimated wavelet transform and wavelet
packet transform perform identically. Both identify two clusters as optimal, and both misclassify the
first and last time series as belonging to the other group.

Conclusions

In this example, you learned how the wavelet packet transform differs from the discrete wavelet
transform. Specifically, the wavelet packet tree also filters the wavelet (detail) coefficients, while the
wavelet transform only iterates on the scaling (approximation) coefficients.

You learned that the wavelet packet transform shares the important energy-preserving property of the DWT while providing superior frequency resolution. In some applications, this superior frequency resolution makes the wavelet packet transform an attractive alternative to the DWT.

Appendix

helperEarthQuakeExplosionClassifier

```matlab
function [WavPacketCluster,WtCluster,PspecCluster] = helperEarthQuakeExplosionClassifier(Data,Lev
%   This function is provided solely in support of the featured example
%   waveletpacketsdemo.mlx.
%   This function may be changed or removed in a future release.
%   Data - The data matrix with individual time series as column vectors.
%   Level - The level of the wavelet packet and wavelet transforms

NP = 2^Level;
NW = Level+1;
WpMatrix = zeros(16,NP);
WtMatrix = zeros(16,NW);

for jj = 1:8
    [~,~,~,~,RE] = modwpt(Data(:,jj),Level,'fk6');
    wt = modwt(Data(:,jj),Level,'fk6');
    WtMatrix(jj,:) = sum(abs(wt).^2,2)./norm(Data(:,jj),2)^2;
    WpMatrix(jj,:) = RE;
end

for kk = 9:16
    [~,~,~,~,RE] = modwpt(Data(:,kk),Level,'fk6');
    wt = modwt(Data(:,kk),Level,'fk6');
    WtMatrix(kk,:) = sum(abs(wt).^2,2)./norm(Data(:,kk),2)^2;
    WpMatrix(kk,:) = RE;
end

% For repeatability
rng('default');

WavPacketCluster = evalclusters(WpMatrix,'kmeans','gap','KList',1:6,...
    'Distance','cityblock');

WtCluster = evalclusters(WtMatrix,'kmeans','gap','KList',1:6,...
'Distance','cityblock');

Pxx = periodogram(Data,hamming(numel(Data(:,1))),[],1,'power');

PspecCluster = evalclusters(Pxx','kmeans','gap','KList',1:6,...
    'Distance','cityblock');
end
```

References

[1] Wickerhauser, Mladen Victor. *Adapted Wavelet Analysis from Theory to Software*. Wellesley, MA: A.K. Peters, 1994.

[2] Percival, Donald B., and Andrew T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge ; New York: Cambridge University Press, 2000.

[3] Shumway, Robert H., and David S. Stoffer. *Time Series Analysis and Its Applications: With R Examples*. 3rd ed. Springer Texts in Statistics. New York: Springer, 2011.

[4] Stoffer, D. H. "asta: Applied Statistical Time Series Analysis." R package version 1.3. `https://CRAN.R-project.org/package=astsa`, 2014.

See Also

dwpt | modwpt | modwt | modwtmra

Wavelet Analysis of Physiologic Signals

This example shows how to use wavelets to analyze physiologic signals.

Physiologic signals are frequently nonstationary meaning that their frequency content changes over time. In many applications, 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.

This example presents a few illustrative cases where the ability of wavelets to provide a local time-frequency analysis of signals is beneficial.

R Wave Detection in the Electrocardiogram with MODWT

The QRS complex consists of three deflections in the electrocardiogram (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')
```

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 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')
```


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')
```

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')
```

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. Examine the scale 4 and 5 magnitude-squared wavelet details plotted along with R peak times as annotated by the experts. The level-4 details are shifted for visualization.

```
ecgmra = modwtmra(wt);
figure
plot(tm,ecgmra(5,:).^2,'b')
hold on
plot(tm,ecgmra(4,:).^2+0.6,'b')
set(gca,'xlim',[14.3 25.5])
timemarks = repelem(tm(ann),2);
N = numel(timemarks);
markerlines = reshape(repmat([0;1],1,N/2),N,1);
h = stem(timemarks,markerlines,'k--');
h.Marker = 'none';
set(gca,'ytick',[0.1 0.6]);
set(gca,'yticklabels',{'D5','D4'})
xlabel('Seconds')
title('Magnitude-Squared Level 4 and 5 Details')
```

Magnitude-Squared Level 4 and 5 Details
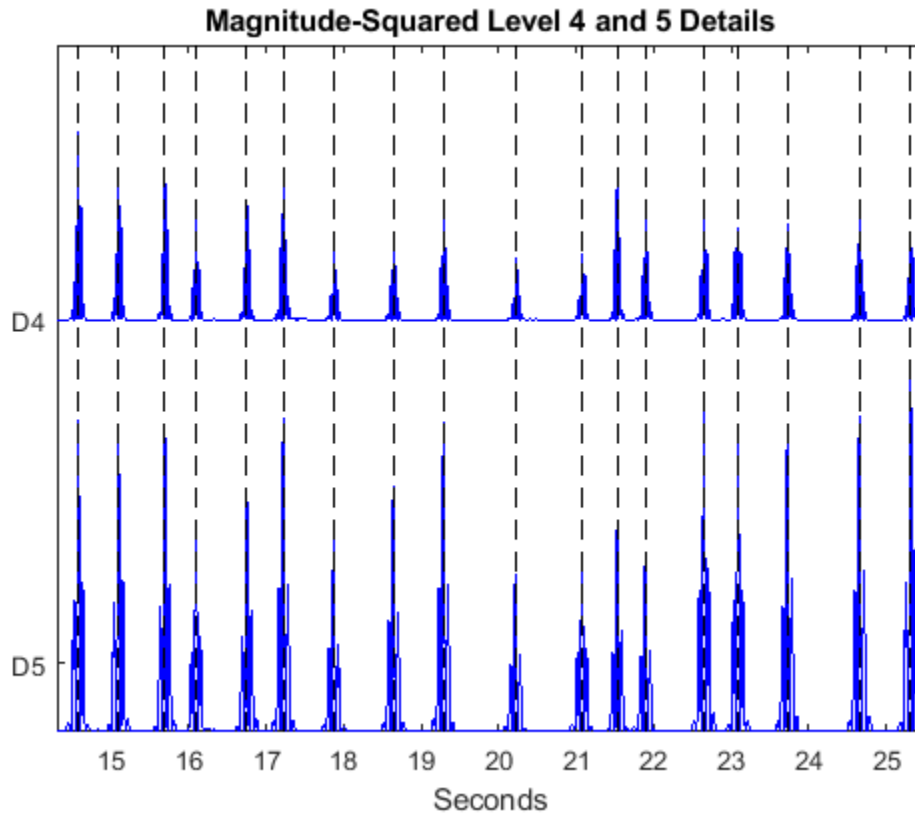
You see that peaks in the level 4 and level 5 details tend to co-occur. A more advanced wavelet peak-finding algorithm could exploit this by using information from multiple scales simultaneously.

Time-Varying Wavelet Coherence Analysis of Brain Dynamics

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 data is taken from Cui, Bryant, & Reiss (2012) and was kindly provided by the authors for this example. The recording site was the superior frontal cortex for both subjects. The data is sampled at 10 Hz.

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

```
load NIRSData
figure
plot(tm,[NIRSData(:,1) NIRSData(:,2)])
legend('Subject 1','Subject 2','Location','NorthWest')
xlabel('Seconds')
title('NIRS Data')
grid on
```
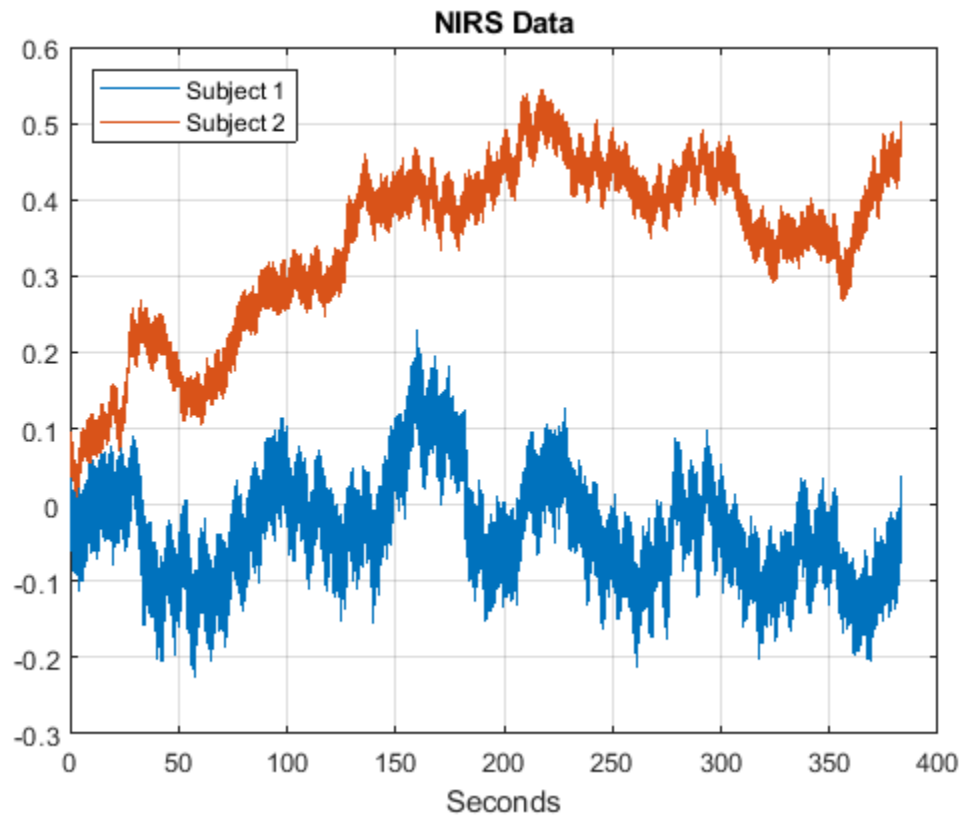
Examining the time-domain data, it is not clear what oscillations are present in the individual time series, or what oscillations are common to both data sets. Use wavelet analysis to answer both questions.

```
cwt(NIRSData(:,1),10,'bump')
figure
cwt(NIRSData(:,2),10,'bump')
```

Magnitude Scalogram
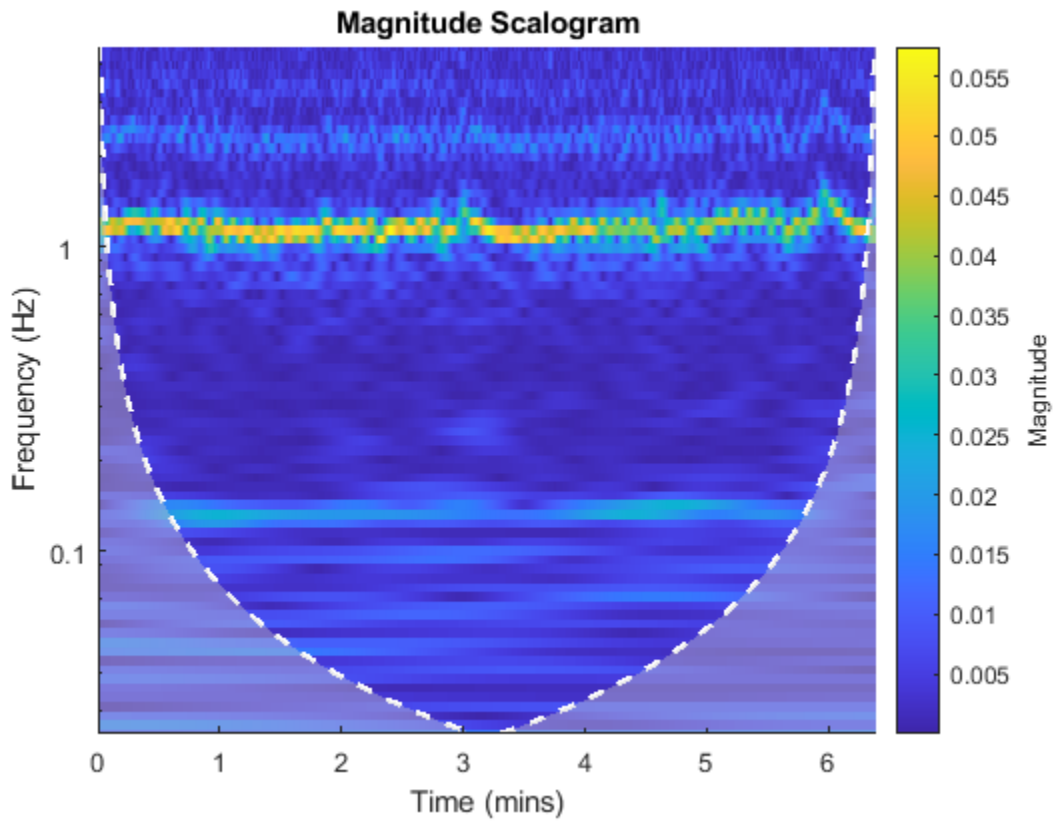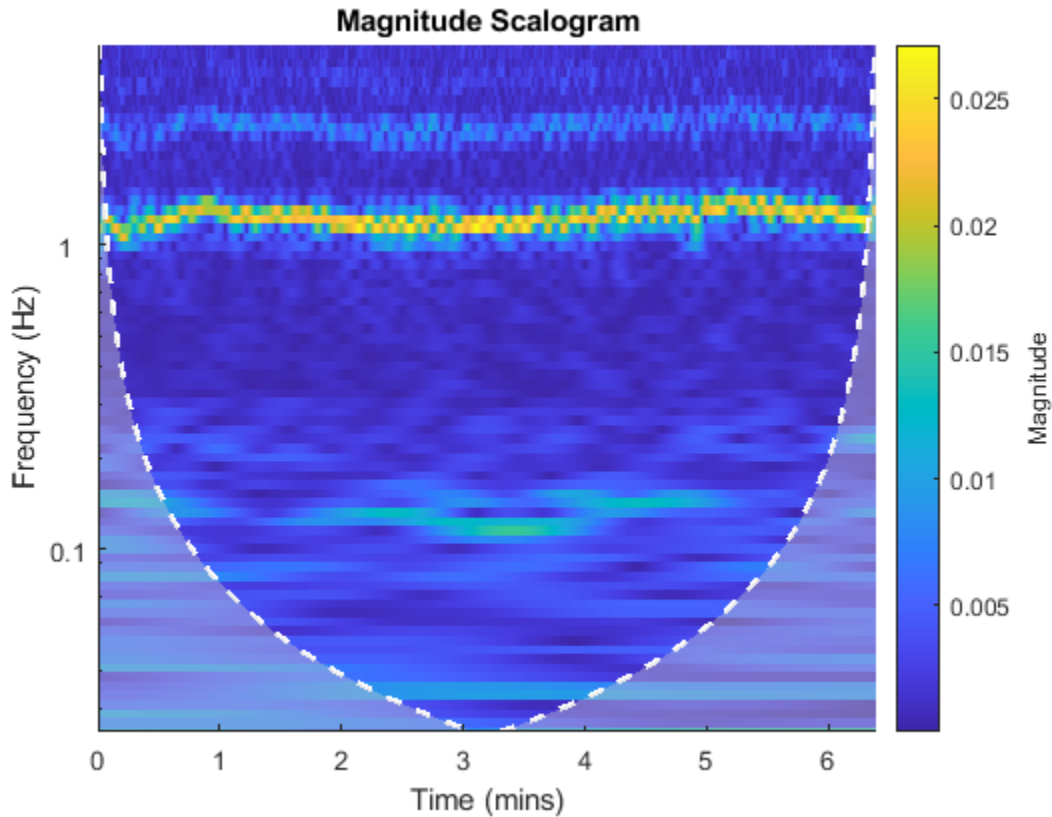
The CWT analyses reveal strong frequency-modulated oscillations in both datasets around 1 Hz. These are due to the cardiac cycles in the two subjects. Additionally, there appears to be a weaker oscillation in both datasets around 0.15 Hz. This activity is stronger and more consistent in subject 1 than subject 2. Wavelet coherence can enhance the detection of weak oscillations that are jointly present in two time series.

```
[wcoh,~,F] = wcoherence(NIRSData(:,1),NIRSData(:,2),10);
figure
surf(tm,F,abs(wcoh).^2); view(0,90)
shading interp
axis tight
hc = colorbar;
hc.Label.String = 'Coherence';
title('Wavelet Coherence')
xlabel('Seconds')
ylabel('Hz')
ylim([0 2.5])
set(gca,'ytick',[0.15 1.2 2])
```
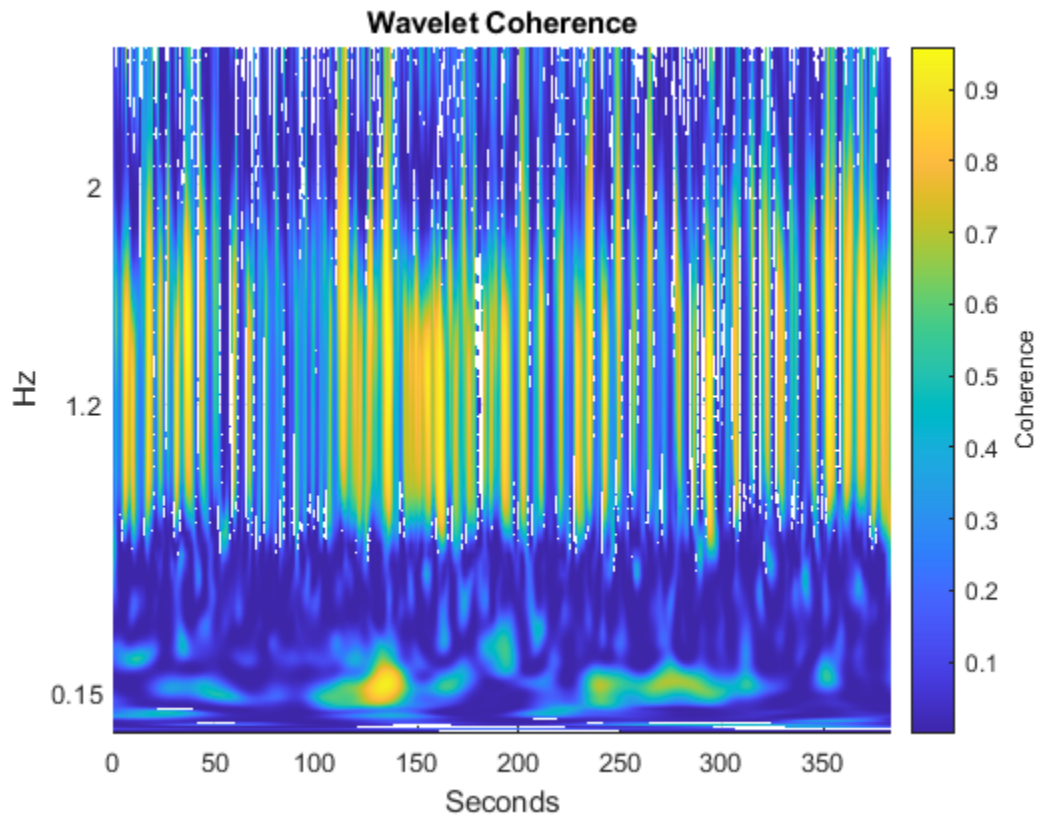
In the wavelet coherence, there is a strong correlation around 0.15 Hz. This is in the frequency band corresponding to the experimental task and represents task-related coherent oscillations in brain activity in the two subjects. Add to the plot time markers which indicate two task periods. The period between the tasks is a rest period.

```
taskbd = [245 1702 2065 3474];
tvec = repelem(tm(taskbd),2);
yvec = [0 max(F)]';
yvec = reshape(repmat(yvec,1,4),8,1);
hold on
stemPlot = stem(tvec,yvec,'w--','linewidth',2);
stemPlot.Marker = 'none';
```
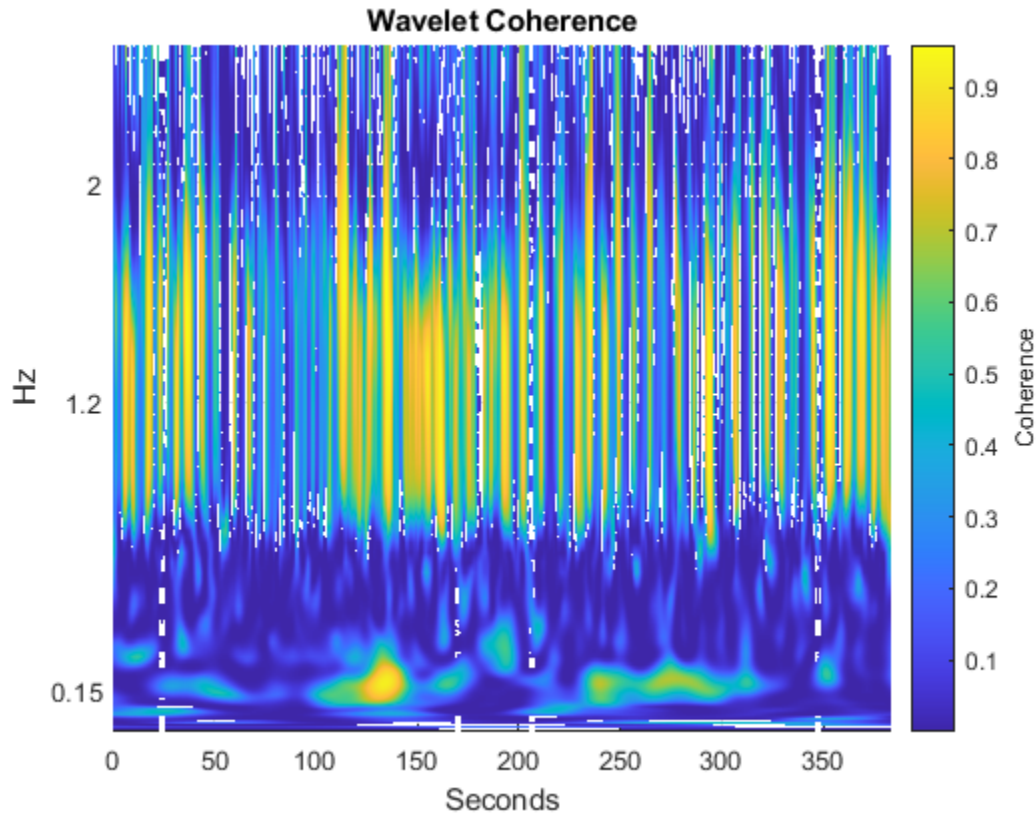
This example used `cwt` to obtain and plot a time-frequency analysis of the individual NIRS time series. The example also used `wcoherence` to obtain the wavelet coherence of the two time series. The use of wavelet coherence often enables you to detect coherent oscillatory behavior in two time series which may be fairly weak in each individual series. Consult Cui, Bryant, & Reiss (2012) for a more detailed wavelet-coherence analysis of this data.

Time-Frequency Analysis of Otoacoustic Emission Data with the CWT

Otoacoustic emissions (OAEs) are narrowband oscillatory signals emitted by the cochlea (inner ear) and their presence is indicative of normal hearing. Load and plot some example OAE data. The data are sampled at 20 kHz.

```
load dpoae
figure
plot(t.*1000,dpoaets)
xlabel('Milliseconds')
ylabel('Amplitude')
```
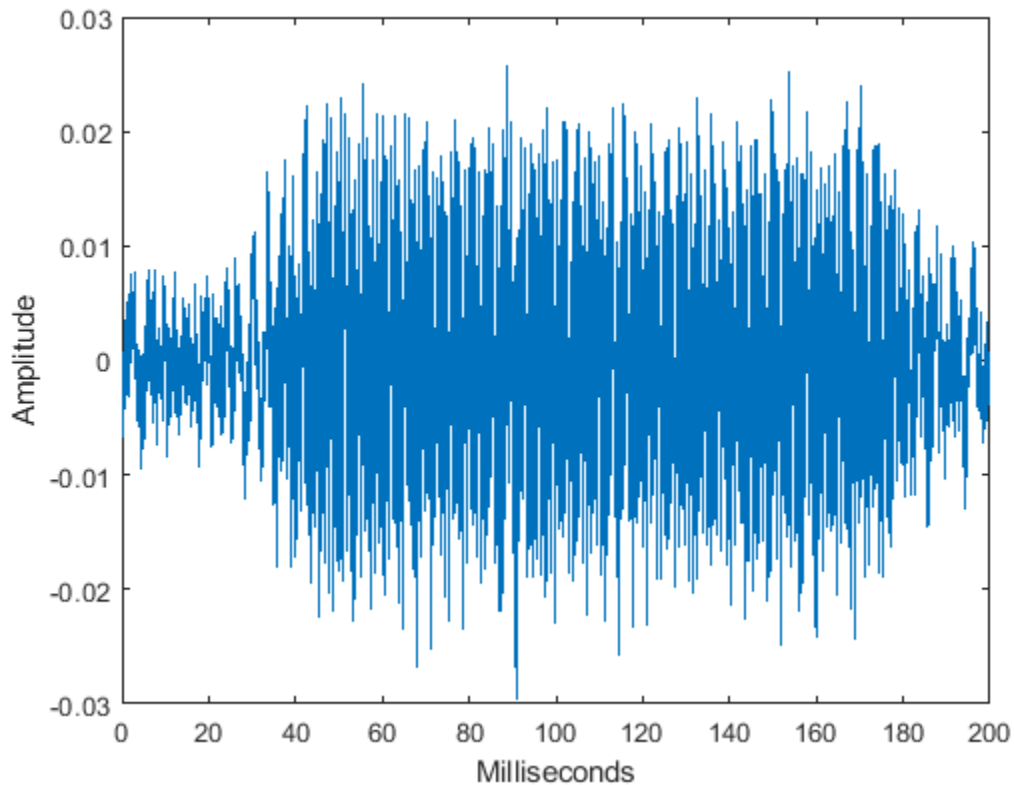
The emission was evoked by a stimulus beginning at 25 milliseconds and ending at 175 milliseconds. Based on the experimental parameters, the emission frequency should be 1230 Hz. Obtain and plot the CWT as a function of time and frequency. Use the default analytic Morse wavelet with 16 voices per octave.

```
[dpoaeCWT,f] = cwt(dpoaets,2e4,'VoicesPerOctave',16);
helperCWTTimeFreqPlot(dpoaeCWT,t.*1000,f,...
    'surf','CWT of OAE','milliseconds','Hz')
```
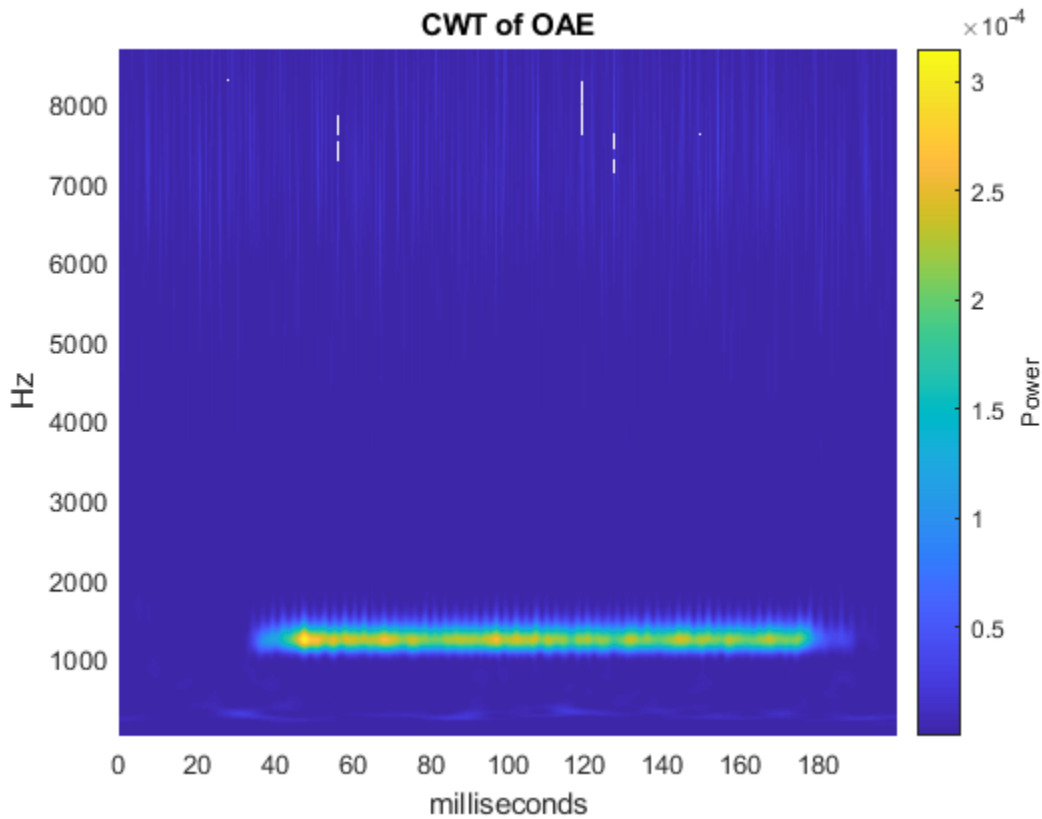
You can investigate the time evolution of the OAE by finding the CWT coefficients closest in frequency to 1230 Hz and examining their magnitudes as a function of time. Plot the magnitudes along with time markers designating the beginning and end of the evoking stimulus.

```
[~,idx1230] = min(abs(f-1230));
cfsOAE = dpoaeCWT(idx1230,:);
plot(t.*1000,abs(cfsOAE))
hold on
AX = gca;
plot([25 25],[AX.YLim(1) AX.YLim(2)],'r')
plot([175 175],[AX.YLim(1) AX.YLim(2)],'r')
xlabel('msec')
title('CWT Coefficient Magnitudes')
```
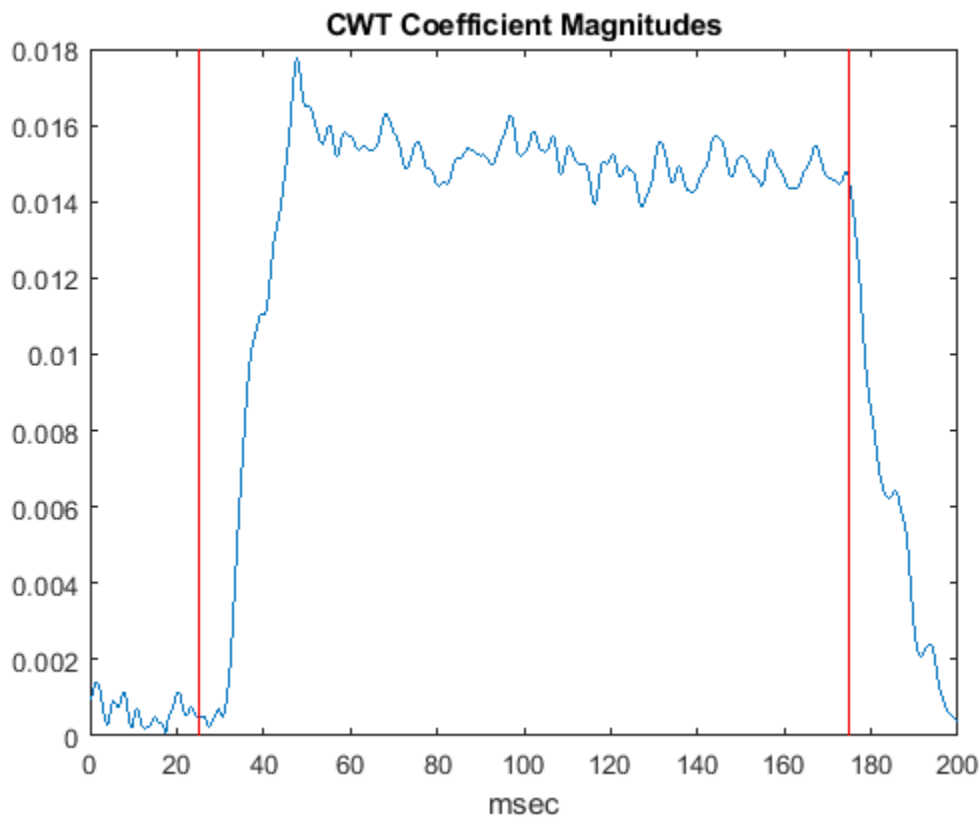
CWT Coefficient Magnitudes

There is some delay between the onset of the evoking stimulus and the OAE. Once the evoking stimulus is terminated, the OAE immediately begins to decay in magnitude.

Another way to isolate the emission is to use the inverse CWT to reconstruct a frequency-localized approximation in the time domain.

Construct a frequency-localized emission approximation by extracting the CWT coefficients corresponding to frequencies between 1150 and 1350 Hz. Use these coefficients and invert the CWT. Plot the original data along with the reconstruction and markers indicating the beginning and end of the evoking stimulus.

```
frange = [1150 1350];
xrec = icwt(dpoaeCWT,f,frange);
figure
plot(t.*1000,dpoaets)
hold on
plot(t.*1000,xrec,'r')
AX = gca;
ylimits = AX.YLim;
plot([25 25],ylimits,'k')
plot([175 175],ylimits,'k')
grid on
xlabel('Milliseconds')
ylabel('Amplitude')
title('Frequency-Localized Reconstruction of Emission')
```

Frequency-Localized Reconstruction of Emission

In the time-domain data, you clearly see how the emission ramps on and off at the application and termination of the evoking stimulus.

It is important to note that even though a range of frequencies were selected for the reconstruction, the analytic wavelet transform actually encodes the exact frequency of the emission. To demonstrate this, take the Fourier transform of the emission approximation reconstructed from the analytic CWT.

```
xdft = fft(xrec);
freq = 0:2e4/numel(xrec):1e4;
xdft = xdft(1:numel(xrec)/2+1);
figure
plot(freq,abs(xdft))
xlabel('Hz')
ylabel('Magnitude')
title('Fourier Transform of CWT-Based Signal Approximation')
[~,maxidx] = max(abs(xdft));
fprintf('The frequency is %4.2f Hz\n',freq(maxidx))
```

```
The frequency is 1230.00 Hz
```

This example used `cwt` to obtain a time-frequency analysis of the OAE data and `icwt` to obtain a frequency-localized approximation to the signal.

References

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

Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng C-K, Stanley HE. "PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals." Circulation 101(23):e215-e220, 2000. `http://circ.ahajournals.org/cgi/content/full/101/23/e215`

Mallat, S. "A Wavelet Tour of Signal Processing: The Sparse Way", Academic Press, 2009.

Moody, G.B. "Evaluating ECG Analyzers". `http://www.physionet.org/physiotools/wfdb/doc/wag-src/eval0.tex`

Moody GB, Mark RG. "The impact of the MIT-BIH Arrhythmia Database." IEEE Eng in Med and Biol 20(3):45-50 (May-June 2001).

Appendix - Supporting Functions

The following helper functions are used in this example.

• helperCWTTimeFreqPlot.m

- helperCWTTimeFreqVector.m

Featured Examples — Discrete Multiresolution Analysis

Practical Introduction to Multiresolution Analysis

This example shows how to perform and interpret basic signal multiresolution analysis (MRA). The example uses both simulated and real data to answer questions such as: What does multiresolution analysis mean? What insights about my signal can I gain performing a multiresolution analysis? What are some of the advantages and disadvantages of different MRA techniques? Many of the analyses presented here can be replicated using the **Signal Multiresolution Analyzer** app.

What Is Multiresolution Analysis?

Signals often consist of multiple physically meaningful components. Quite often, you want to study one or more of these components in isolation on the same time scale as the original data. *Multiresolution analysis* refers to breaking up a signal into components, which produce the original signal exactly when added back together. To be useful for data analysis, how the signal is decomposed is important. The components ideally decompose the variability of the data into physically meaningful and interpretable parts. The term *multiresolution analysis* is often associated with wavelets or wavelet packets, but there are non-wavelet techniques which also produce useful MRAs.

As a motivating example of the insights you can gain from an MRA, consider the following synthetic signal.

```
Fs = 1e3;
t = 0:1/Fs:1-1/Fs;
comp1 = cos(2*pi*200*t).*(t>0.7);
comp2 = cos(2*pi*60*t).*(t>=0.1 & t<0.3);
trend = sin(2*pi*1/2*t);
rng default
wgnNoise = 0.4*randn(size(t));
x = comp1+comp2+trend+wgnNoise;
plot(t,x)
xlabel('Seconds')
ylabel('Amplitude')
```
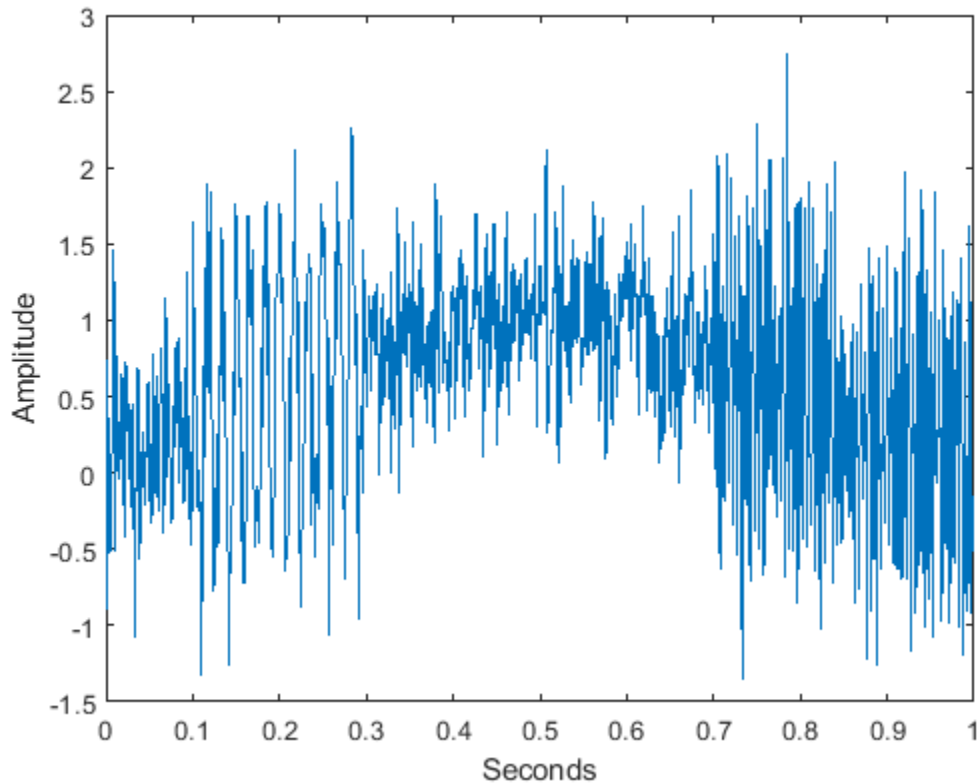
The signal is explicitly composed of three main components: a time-localized oscillation with a frequency of 60 cycles/second, a time-localized oscillation with a frequency of 200 cycles/second, and a trend term. The trend term here is also sinusoidal but has a frequency of 1/2 cycle per second, so it completes only 1/2 cycle in the one-second interval. The 60 cycles/second or 60 Hz oscillation occurs between 0.1 and 0.3 seconds, while the 200 Hz oscillation occurs between 0.7 and 1 second.

Not all of this is immediately evident from the plot of the raw data because these components are mixed.

Now, plot the signal from a frequency point of view.

```
xdft = fft(x);
N = numel(x);
xdft = xdft(1:numel(xdft)/2+1);
freq = 0:Fs/N:Fs/2;
plot(freq,20*log10(abs(xdft)))
xlabel('Cycles/second'); ylabel('dB')
grid on
```
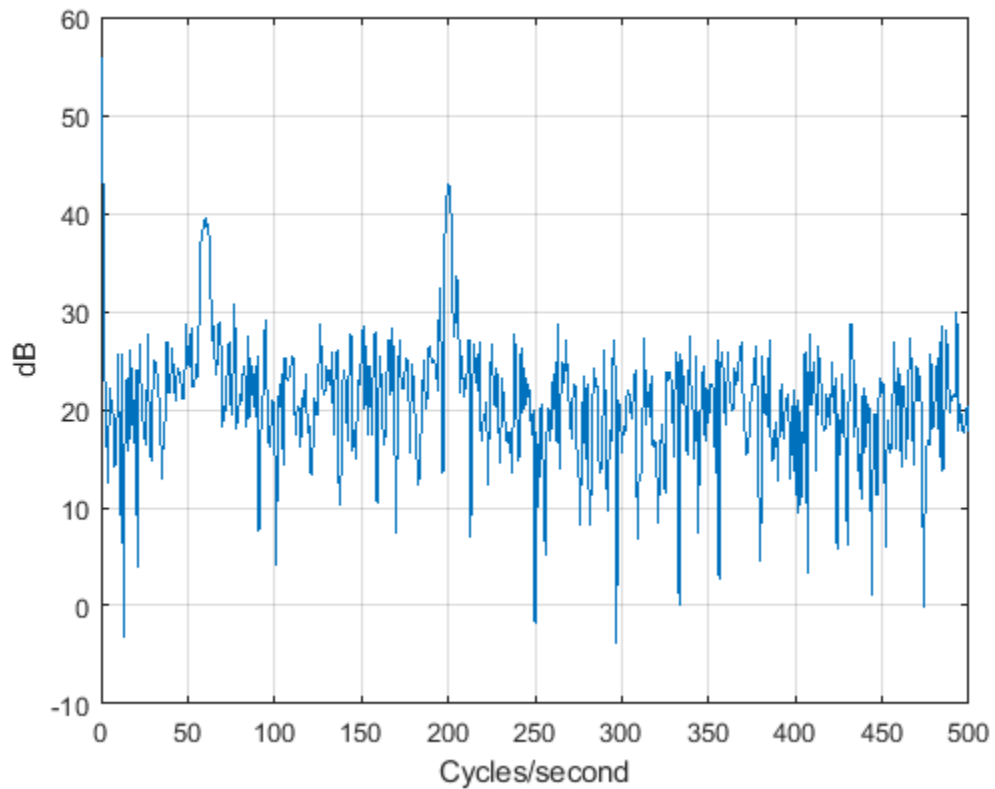
From the frequency analysis, it is much easier for us to discern the frequencies of the oscillatory components, but we have lost their time-localized nature. It is also difficult to visualize the trend in this view.

To gain some simultaneous time and frequency information, we can use a time-frequency analysis technique like the continuous wavelet transform.

```
cwt(x,Fs)
```
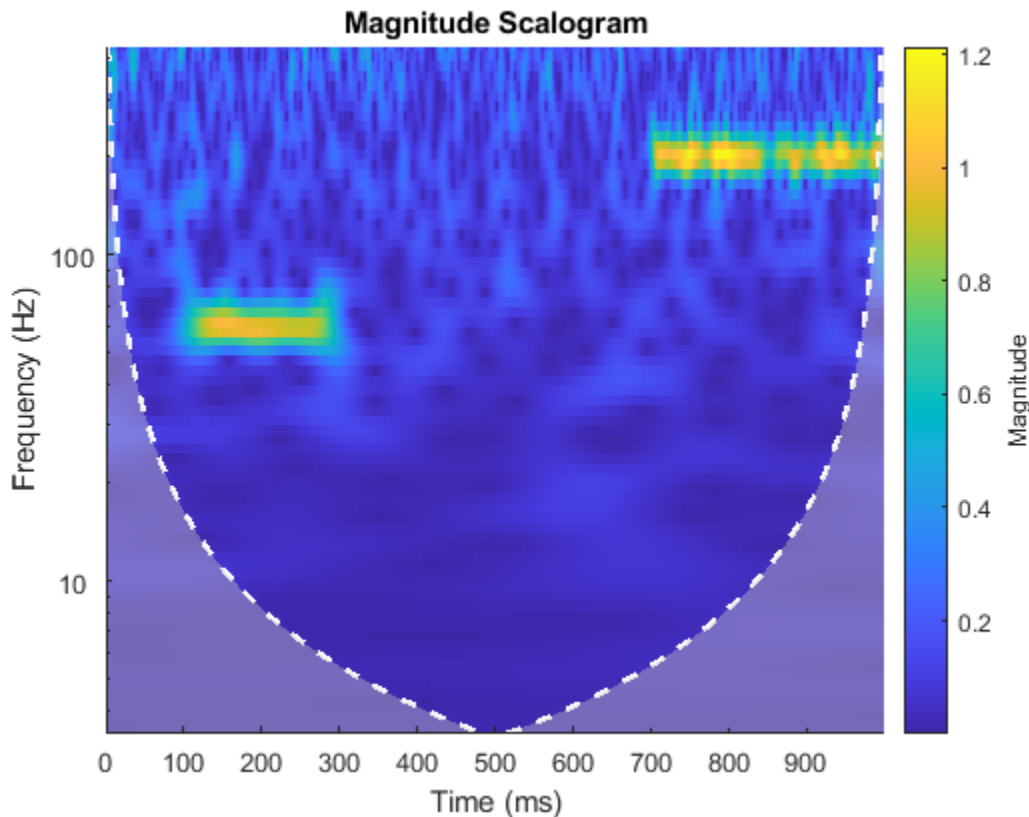
Now you see the time extents of the 60 Hz and 200 Hz components. However, we still do not have any useful visualization of the trend.

The time-frequency view provides useful information, but in many situations you would like to separate out components of the signal in time and examine them individually. Ideally, you want this information to be available on the same time scale as the original data.

Multiresolution analysis accomplishes this. In fact, a useful way to think about multiresolution analysis is that it provides a way of avoiding the need for time-frequency analysis while allowing you to work directly in the time domain.

Separating Signal Components in Time

Real-world signals are a mixture of different components. Often you are only interested in a subset of these components. Multiresolution analysis allows you to narrow your analysis by separating the signal into components at different *resolutions*.

Extracting signal components at different resolutions amounts to decomposing variations in the data on different time scales, or equivalently in different frequency bands (different rates of oscillation). Accordingly, you can visualize signal variability at different scales, or frequency bands simultaneously.

Analyze and plot the synthetic signal using a wavelet MRA. The signal is analyzed at eight resolutions or levels.

```
mra = modwtmra(modwt(x,8));
helperMRAPlot(x,mra,t,'wavelet','Wavelet MRA',[2 3 4 9])
```
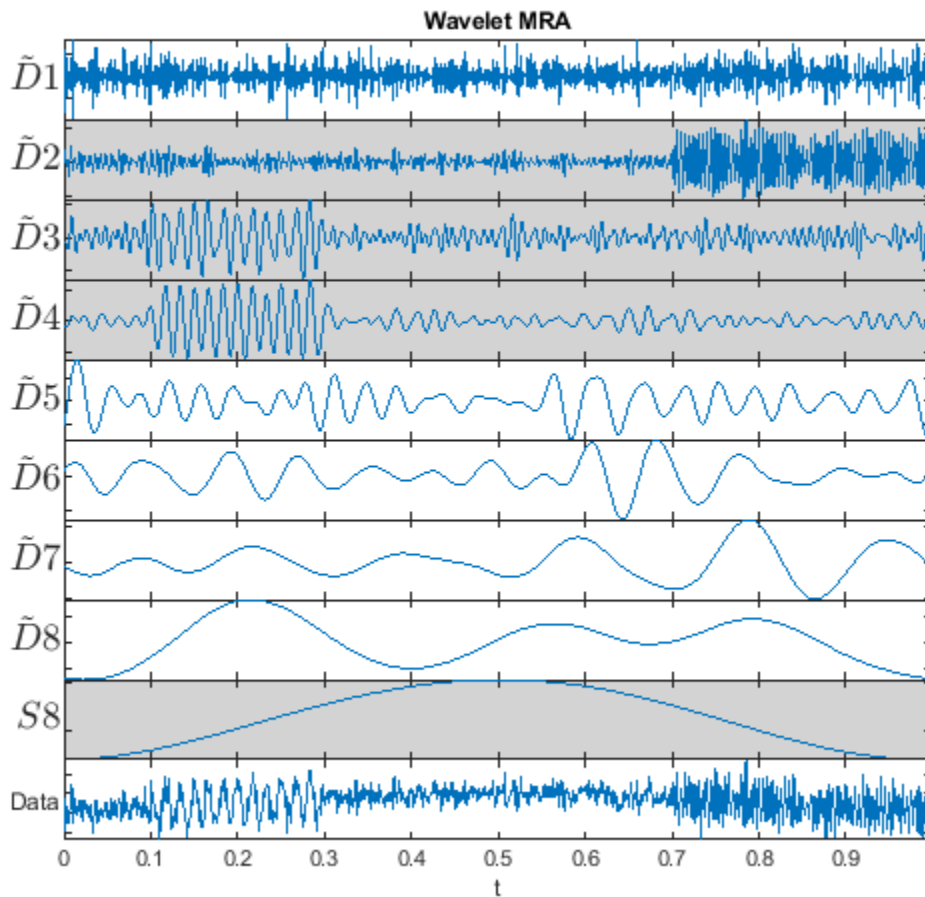
Without explaining what the notations on the plot mean, let us use our knowledge of the signal and try to understand what this wavelet MRA is showing us. If you start from the uppermost plot and proceed down until you reach the plot of the original data, you see that the components are become progressively smoother. If you prefer to think about data in terms of frequency, the frequencies contained in the components are becoming lower. Recall that the original signal had three main components, a high frequency oscillation at 200 Hz, a lower frequency oscillation of 60 Hz, and a trend term, all corrupted by additive noise.

If you look at the $\tilde{D}2$ plot, you can see that the time-localized high frequency component is isolated there. You can see and investigate this important signal feature essentially in isolation. The next two plots contain the lower frequency oscillation. This is an important aspect of multiresolution analysis, namely important signal components may not end up isolated in one MRA component, but they are rarely located in more than two. Finally, we see the $S8$ plot contains the trend term. For convenience, the color of the axes in those components has been changed to highlight them in the MRA. If you prefer to visualize this plot or subsequent ones without the highlighting, leave out the last numeric input to `helperMRAPlot`.

The wavelet MRA uses fixed functions called *wavelets* to separate the signal components. The kth wavelet MRA component, denoted by $\tilde{D}k$ in the previous plot, can be regarded as a filtering of the

signal into frequency bands of the form $[\frac{1}{2^{k+1}\Delta t}, \frac{1}{2^{k}\Delta t}]$ where Δt is the sampling period, or sampling interval. The final smooth component, denoted in the plot by *SL*, where *L* is the level of the MRA, captures the frequency band $[0, \frac{1}{2^{L+1}\Delta t}]$. The accuracy of this approximation depends on the wavelet used in the MRA. See [4] for detailed descriptions of wavelet and wavelet packet MRAs.

However, there are other MRA techniques to consider.

The *empirical mode decomposition* (EMD) is a data-adaptive multiresolution technique. EMD recursively extracts different resolutions from the data without the use of fixed functions or filters. EMD regards a signal as consisting of a *fast* oscillation superimposed on a *slower* one. After the fast oscillation is extracted, the process treats the remaining slower component as the new signal and again regards it as a fast oscillation superimposed on a slower one. The process continues until some stopping criterion is reached. While EMD does not use fixed functions like wavelets to extract information, the EMD approach is conceptually very similar to the wavelet method of separating the signal into *details* and *approximations* and then separating the approximation again into details and an approximation. The MRA components in EMD are referred to as *intrinsic mode functions* (IMF). See [3] for a detailed treatment of EMD.

Plot the EMD analysis of the same signal.

```
[imf_emd,resid_emd] = emd(x);
helperMRAPlot(x,imf_emd,t,'emd','Empirical Mode Decomposition',[1 2 3 6])
```

Empirical Mode Decomposition
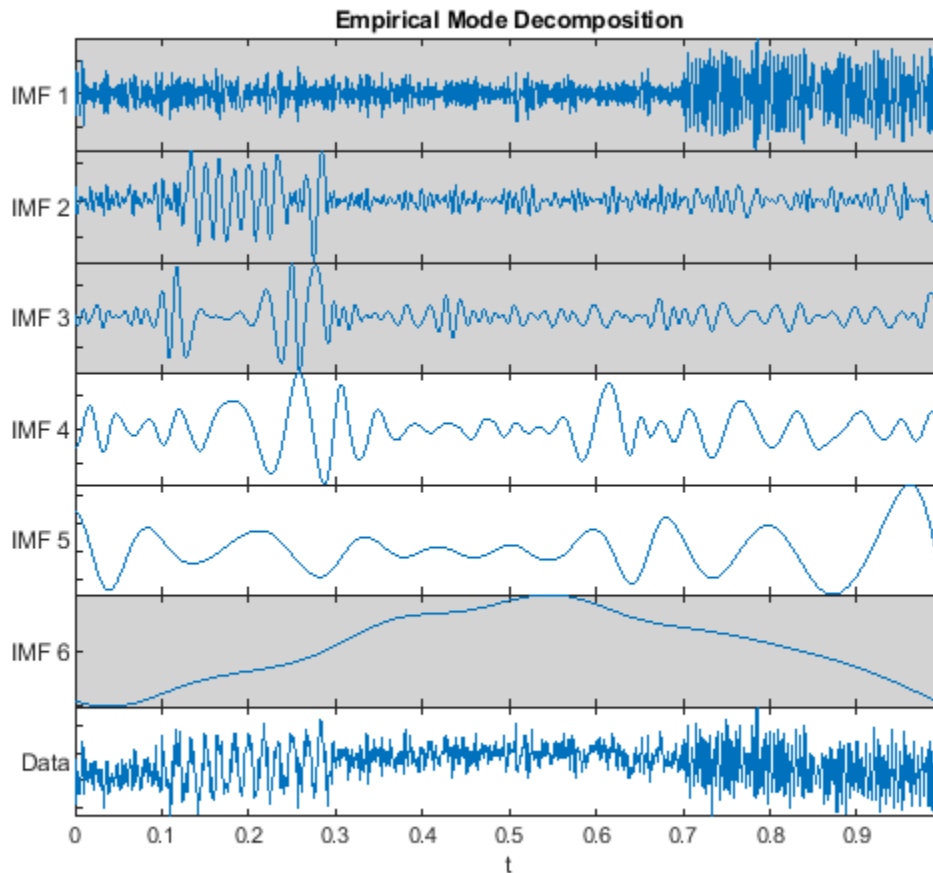
While the number of MRA components is different, the EMD and wavelet MRAs produce a similar picture of the signal. This is not accidental. See [2] for a description of the similarities between the wavelet transform and EMD.

In the EMD decomposition, the high-frequency oscillation is localized to the first intrinsic mode function (IMF 1). The lower frequency oscillation is localized largely to IMF 2, but you can see some effect also in IMF 3. The trend component in IMF 6 is very similar to the trend component extracted by the wavelet technique.

Yet another technique for adaptive multiresolution analysis is *variational mode decomposition* (VMD). Like EMD, VMD attempts to extract intrinsic mode functions, or modes of oscillation from the signal without using fixed functions for analysis. But EMD and VMD determine the modes in very different ways. EMD works recursively on the time domain signal to extract progressively lower frequency IMFs. VMD starts by identifying signal peaks in the frequency domain and extracts all modes concurrently. See [1] for a treatment of VMD.

```
[imf_vmd,resid_vmd] = vmd(x);
helperMRAPlot(x,imf_vmd,t,'vmd','Variational Mode Decomposition',[2 4 5])
```

Variational Mode Decomposition
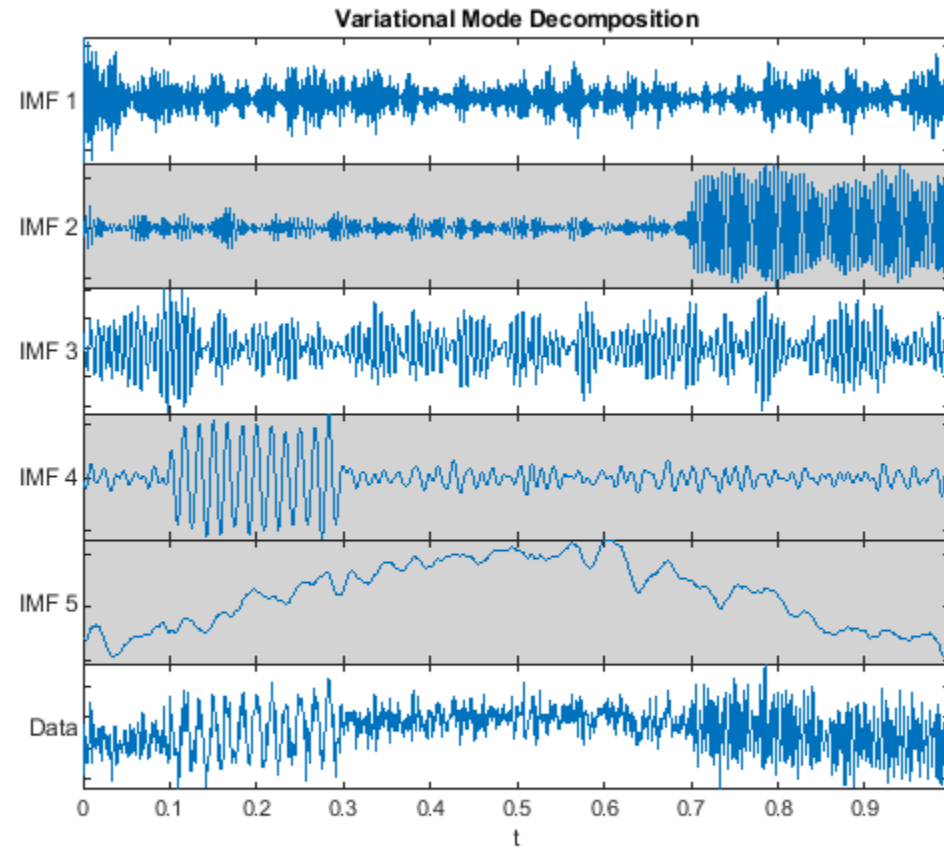
The key thing to note is that similar to the wavelet and EMD decompositions, VMD segregates the three components of interest into completely separate modes or into a small number of adjacent modes. All three techniques allow you to visualize signal components on the same time scale as the original signal.

For a real-world example of useful component separation, consider a seismograph (vertical acceleration, nm/s^2) of the Kobe earthquake, 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.

```
load KobeTimeTable
T = KobeTimeTable.t;
kobe = KobeTimeTable.kobe;
figure
plot(T,kobe)
title('Kobe Earthquake Seismograph')
axis tight
grid on
```
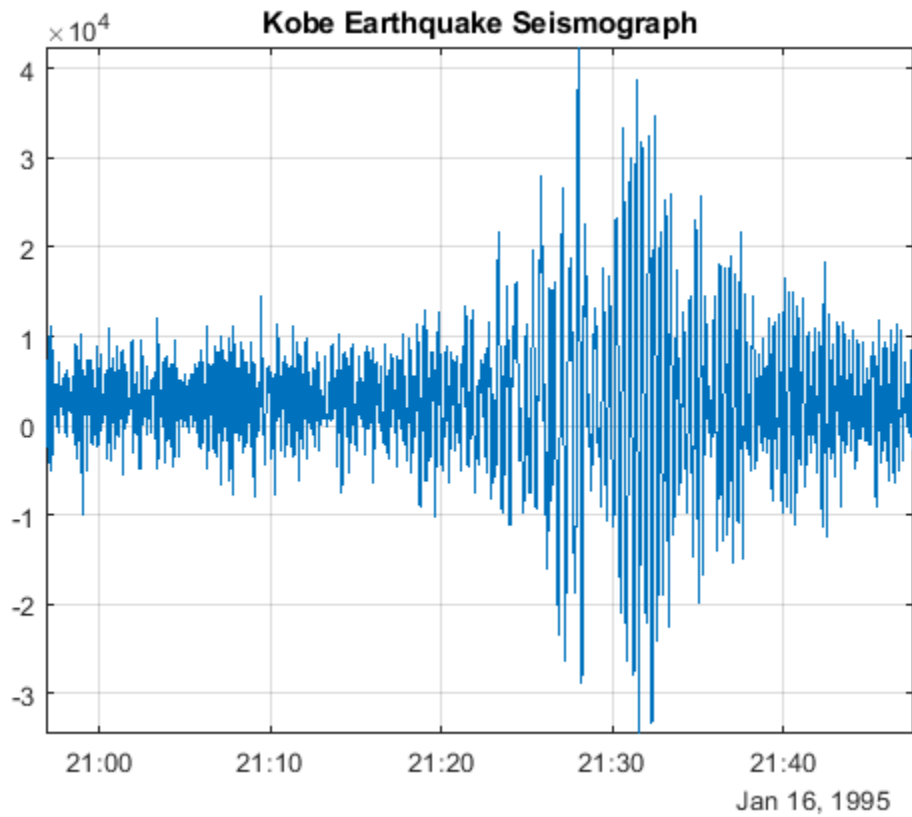
Obtain and plot a wavelet MRA of the data.

```
mraKobe = modwtmra(modwt(kobe,8));
figure
helperMRAPlot(kobe,mraKobe,T,'Wavelet','Wavelet MRA Kobe Earthquake',[4 5])
```

Wavelet MRA Kobe Earthquake
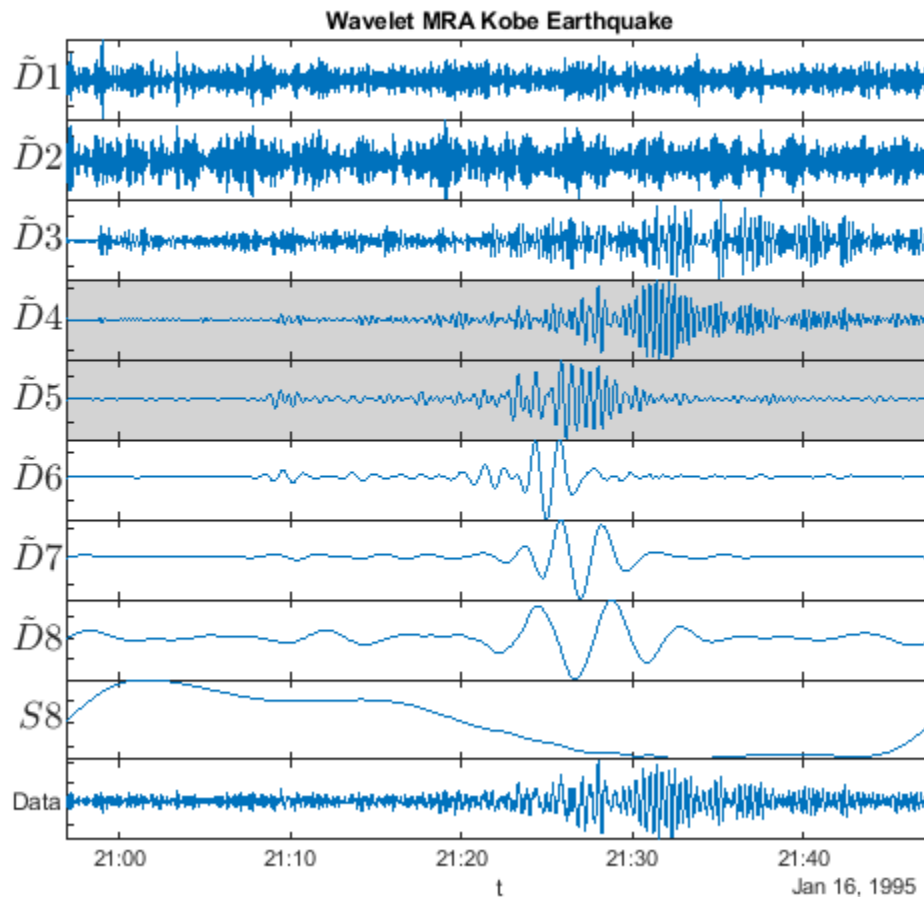
The plot shows the separation of primary and delayed secondary wave components in MRA components $\widetilde{D}4$ and $\widetilde{D}5$. Components in a seismic wave travel at different velocities with primary (compressional) waves traveling faster than secondary (shear) waves. MRA techniques can enable you to study these components in isolation on the original time scale.

Reconstructing Signals from MRA

The point of separating signals into components is often to remove certain components or mitigate their effect on the signal. Crucial to MRA techniques is the ability to reconstruct the original signal.

First, let us demonstrate that all these multiresolution techniques allow you to perfectly reconstruct the signal from the components.

```
sigrec_wavelet = sum(mra);
sigrec_emd = sum(imf_emd,2)+resid_emd;
sigrec_vmd = sum(imf_vmd,2)+resid_vmd;
figure
subplot(3,1,1)
plot(t,sigrec_wavelet); title('Wavelet reconstruction');
set(gca,'XTickLabel',[]);
ylabel('Amplitude');
```

11-11

```
subplot(3,1,2);
plot(t,sigrec_emd); title('EMD reconstruction');
set(gca,'XTickLabel',[]);
ylabel('Amplitude');
subplot(3,1,3)
plot(t,sigrec_vmd); title('VMD reconstruction');
ylabel('Amplitude');
xlabel('Time');
```
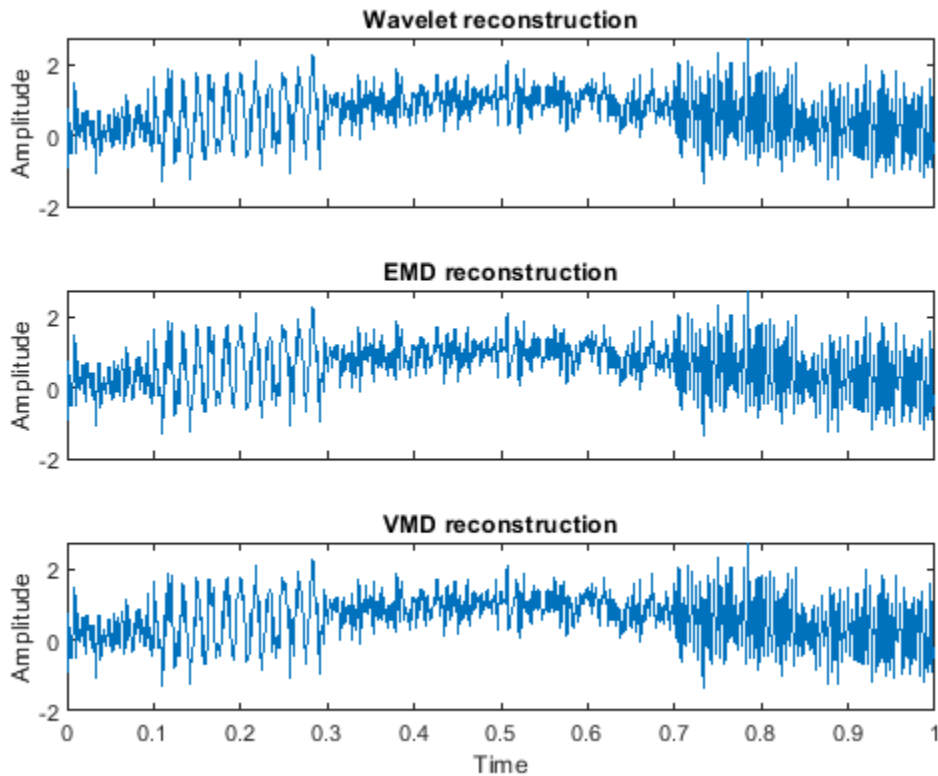

The maximum reconstruction error on a sample-by-sample basis for each of the methods is on the order of 10^{-12} or smaller, indicating they are perfect reconstruction methods. Because the sum of the MRA components reconstructs the original signal, it stands to reason that including or excluding a subset of components could produce a useful approximation.

Return to our original wavelet MRA of the synthetic signal and suppose that you were not interested in in the trend term. Because the trend term is localized in the last MRA component, just exclude that component from the reconstruction.

```
sigWOtrend = sum(mra(1:end-1,:));
figure
plot(t,sigWOtrend)
title('Trend Term Removed')
```

Trend Term Removed
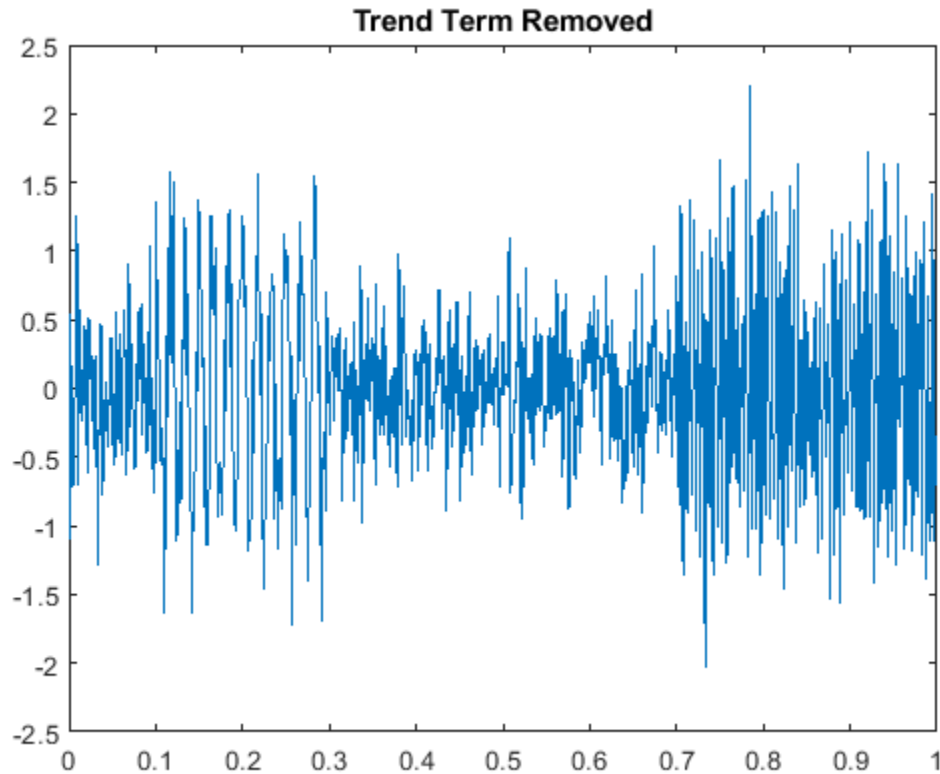
To remove other components, you can create a logical vector with `false` values in components you do not wish to include. Here we remove the trend and highest frequency component along with the first MRA component (which looks to be largely noise). Plot the actual second signal component (60 Hz) along with the reconstruction for comparison.

```
include = true(size(mra,1),1);
include([1 2 9]) = false;
ts = sum(mra(include,:));
plot(t,comp2,'b')
hold on
plot(t,ts,'r')
title('Trend Term and Highest Frequency Component Removed')
xlabel('Seconds');
legend('Component 2','Partial Reconstruction')
xlim([0.0 0.4])
```
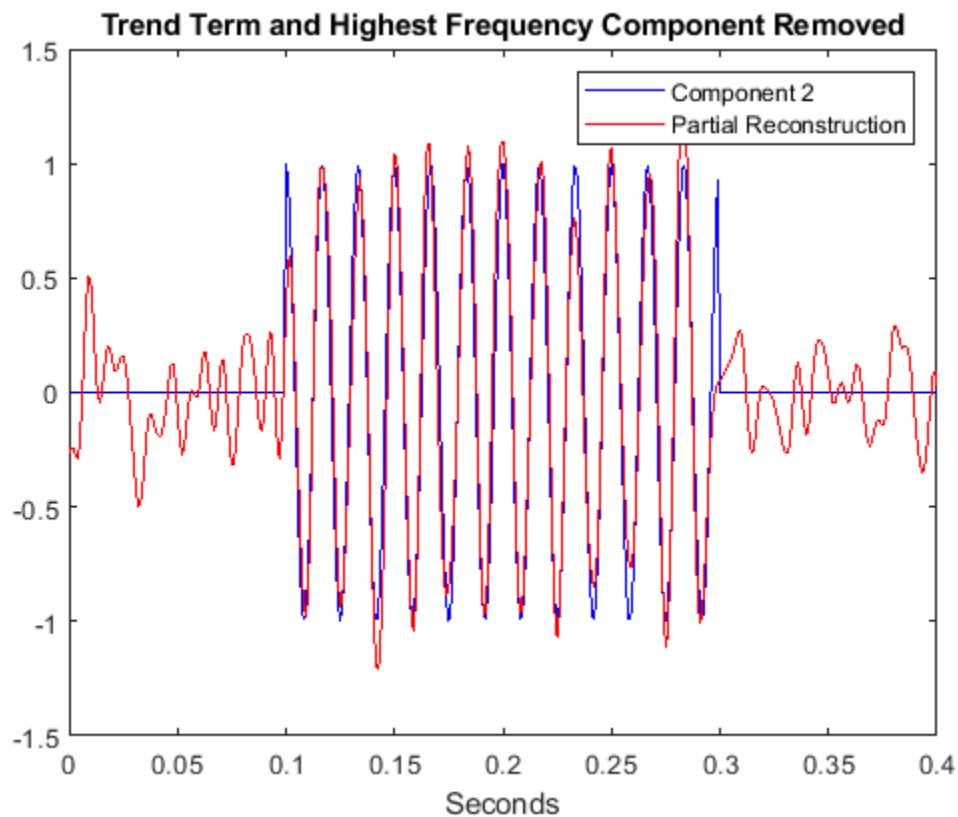
In the previous example, we treated the trend term as a nuisance component to be removed. There are a number of applications where the trend may be the primary component of interest. Let us visualize the trend terms extracted by our three example MRAs.

```
figure
plot(t,trend,'LineWidth',2)
hold on
plot(t,[mra(end,:)' imf_vmd(:,end) imf_emd(:,end)])
grid on
legend('Trend','Wavelet','VMD','EMD')
title('Trend in three MRAs')
```

Trend in three MRAs
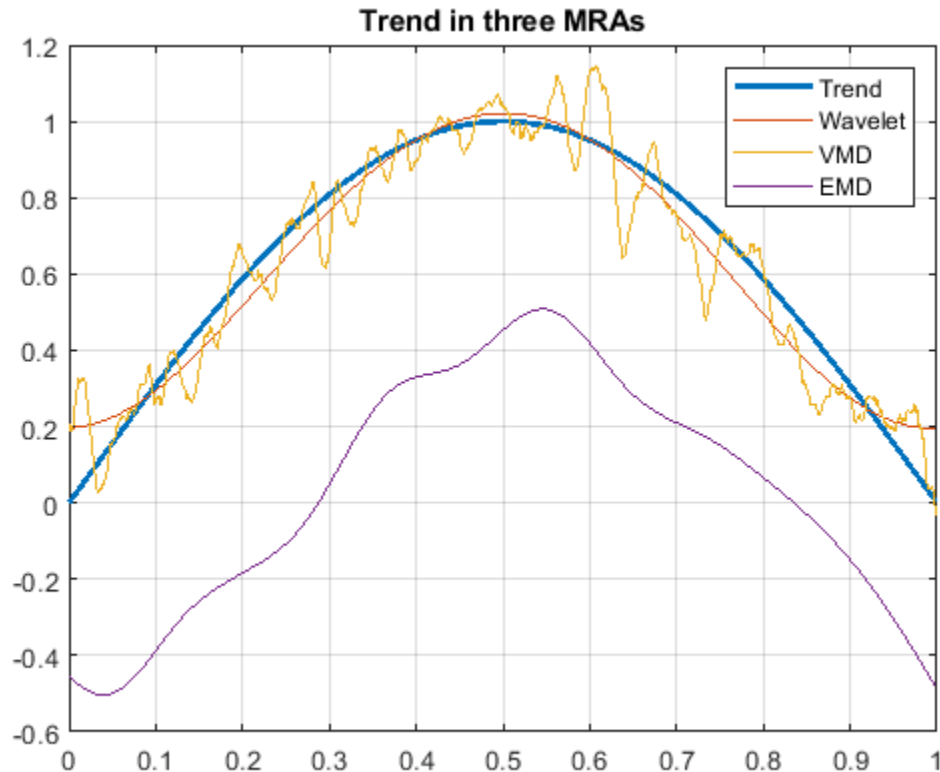
Note that the trend is smoother and most accurately captured by the wavelet technique. EMD finds a smooth trend term, but it is shifted with respect to the true trend amplitude while the VMD technique seems inherently more biased toward finding oscillations than the wavelet and EMD techniques. The implications of this are further discussed in the MRA Techniques — Advantages and Disadvantages on page 11-0   section.

Detecting Transient Changes Using MRA

In the first examples, the role of multiresolution analysis in the detection of oscillatory components in the data and an overall trend was emphasized. However, these are not the only signal features that can be analyzed using multiresolution analysis. MRA can also help localize and detect transient features in a signal like impulsive events, or reductions or increases in variability in certain components. Changes in variability localized to certain scales or frequency bands often indicate significant changes in the process generating the data. These changes are frequently more easily visualized in the MRA components than in the raw data.

To illustrate this, consider the quarterly chain-weighted U.S. real gross domestic product (GDP) data for 1947Q1 to 2011Q4. Quarterly samples correspond to a sampling frequency of 4 samples/year. The vertical black line marks the beginning of the "Great Moderation" signifying a period of decreased macroeconomic volatility in the U.S. beginning in the mid-1980s. Note that this is difficult to discern looking at the raw data.

```
load GDPData
figure
plot(year,realgdp)
hold on
```

```
plot([year(146) year(146)],[-0.06 0.14],'k')
title('GDP Data'); xlabel('Year');
hold off
```
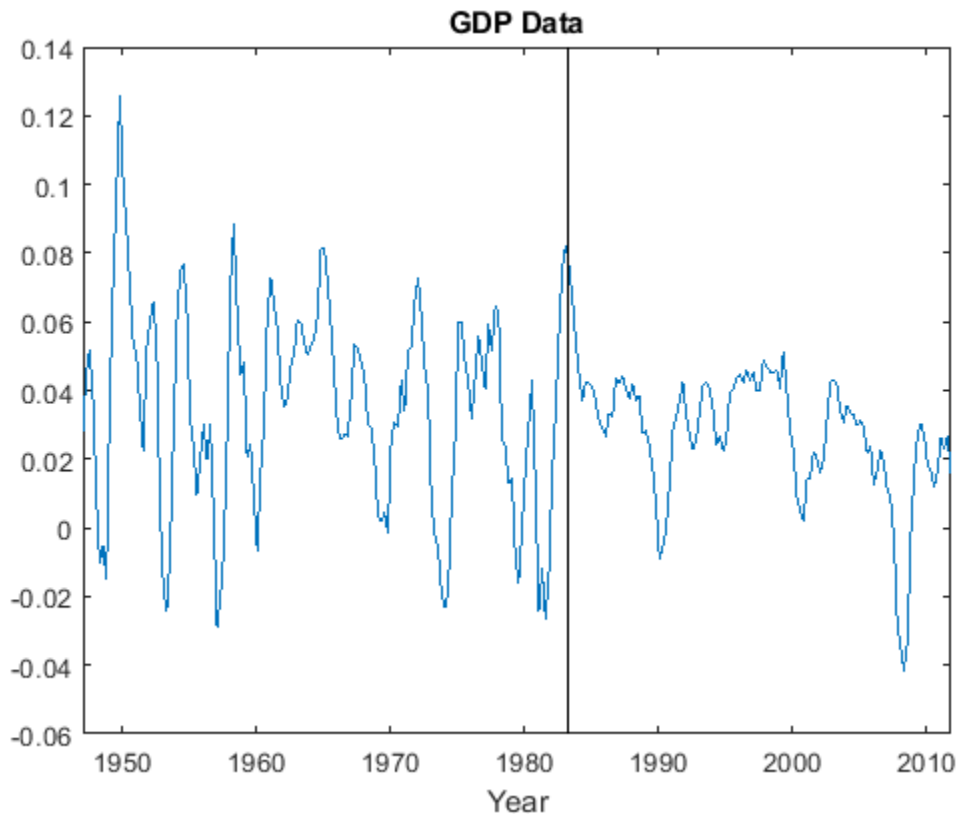
GDP Data

Obtain a wavelet MRA of the GDP data. Plot the finest resolution MRA component with the period of the great moderation marked. Because the wavelet MRA is obtained using fixed filters, we can associate the finest-scale MRA component with frequencies of 1 cycle per year to 2 cycles per year. A component which oscillated with a period of two quarters has a frequency of 2 cycles per year. In this case, the finest-resolution MRA component captures changes in the GDP that occur between adjacent two-quarter intervals to changes that occur from quarter to quarter.

```
mra = modwtmra(modwt(realgdp,'db2'));
figure
plot(year,mra(1,:))
hold on
plot([year(146) year(146)],[-0.015 0.015],'k')
title('Wavelet MRA - Quarter-to-Quarter Changes');
hold off
```
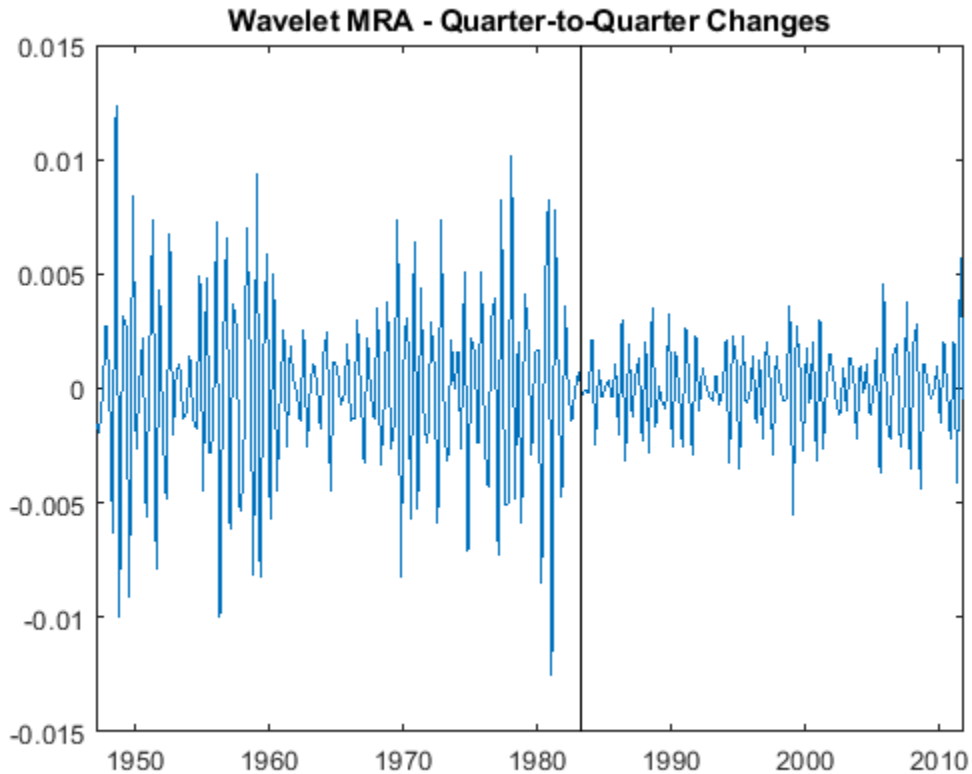
The reduction in variability, or in economic terms volatility, is much more readily apparent in the finest resolution MRA component than in the raw data. Techniques to detect changes in variance like the MATLAB `findchangepts` (Signal Processing Toolbox) often work better on MRA components than on the raw data.

MRA Techniques — Advantages and Disadvantages

In this example, we discussed wavelet and data-adaptive techniques for multiresolution analysis. What are some of the advantages and disadvantages of the various techniques? In other words, for what applications might I choose one over the other? Let us start with wavelets. The wavelet techniques in this example use fixed filters to obtain the MRA. This means that the wavelet MRA has a well-defined mathematical explanation and we can predict the behavior of the MRA. We are also able to tie events in the MRA to specific time scales in the data as was done in the GDP example. The disadvantage is that the wavelet transform divides the signal into octave bands (a reduction in center frequency by 1/2 in each component) so that at high center frequencies the bandwidths are much larger than those at lower frequencies. This means two closely spaced high frequency oscillations can easily end up in the same MRA component with a wavelet technique.

Repeat the first synthetic example but move the two oscillatory components within one octave of each other.

```
Fs = 1e3;
t = 0:1/Fs:1-1/Fs;
comp1 = cos(2*pi*150*t).*(t>=0.1 & t<0.3);
comp2 = cos(2*pi*200*t).*(t>0.7);
trend = sin(2*pi*1/2*t);
```

```
rng default;
wgnNoise = 0.4*randn(size(t));
x = comp1+comp2+trend+wgnNoise;
plot(t,x)
xlabel('Seconds')
ylabel('Amplitude')
```
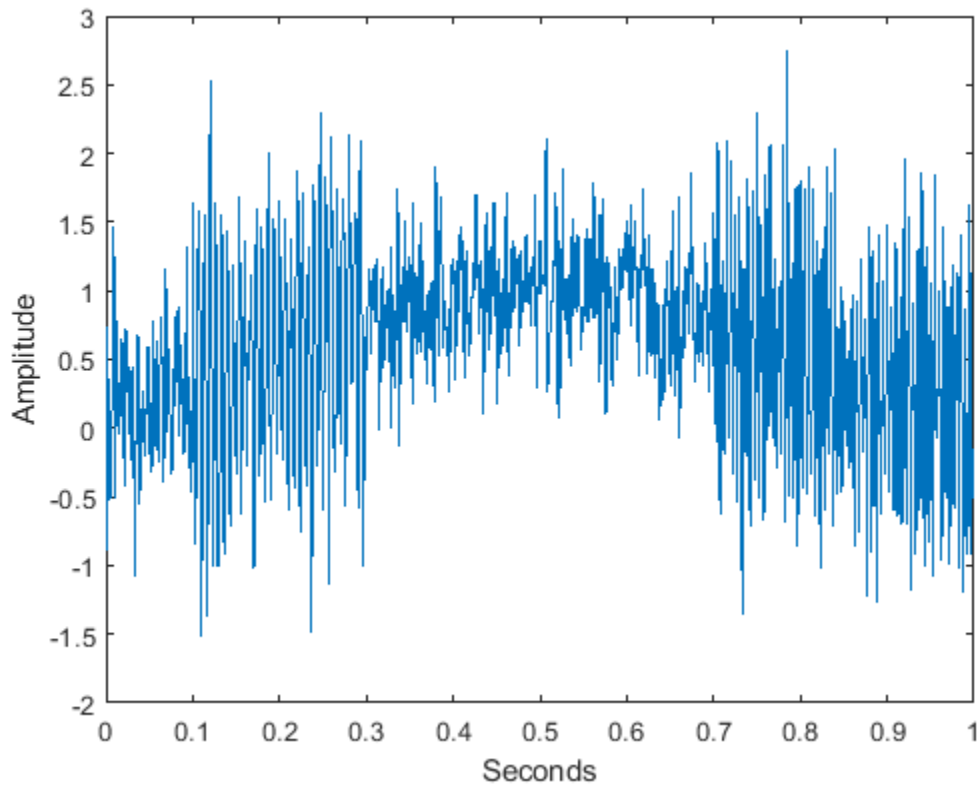

Repeat and plot the wavelet MRA.

```
mra = modwtmra(modwt(x,8));
helperMRAPlot(x,mra,t,'wavelet','Wavelet MRA',[2 9])
```

Wavelet MRA
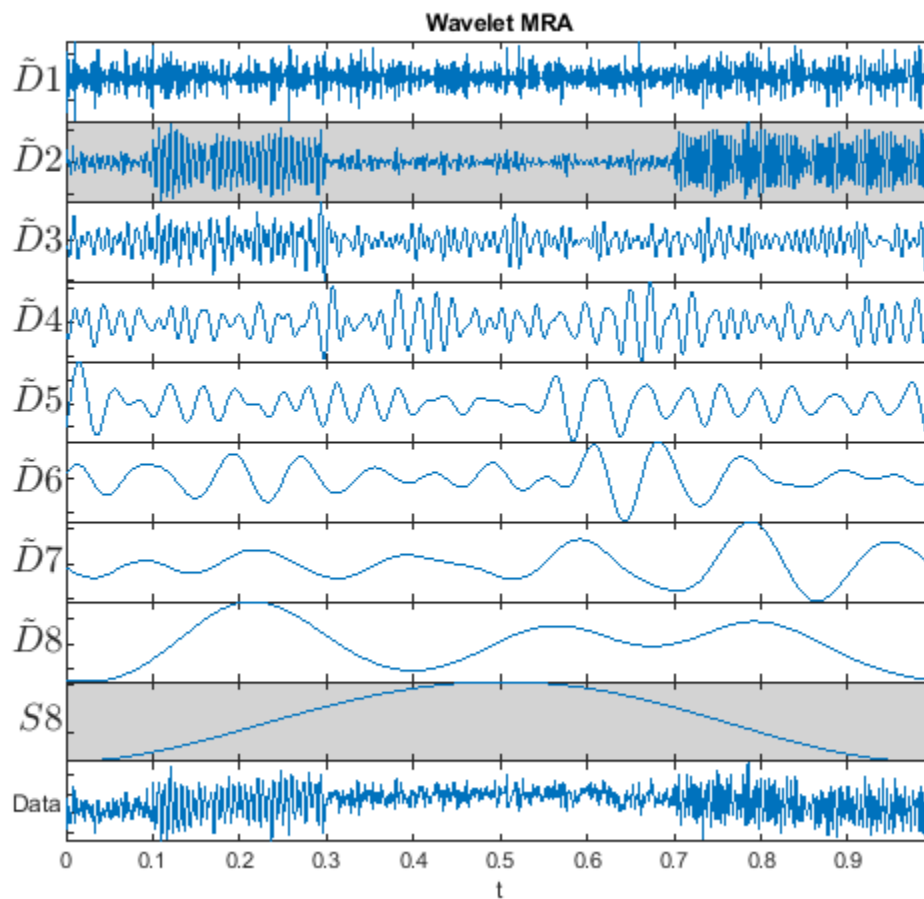
Now we see that $\widetilde{D}2$ contains both the 150 Hz and 200 Hz components. If you repeat this analysis using EMD, you see the same result.

Now let us use the VMD.

```
[imf_vmd,~,info_vmd] = vmd(x);
helperMRAPlot(x,imf_vmd,t,'vmd','VMD',[1 2 3 5]);
```
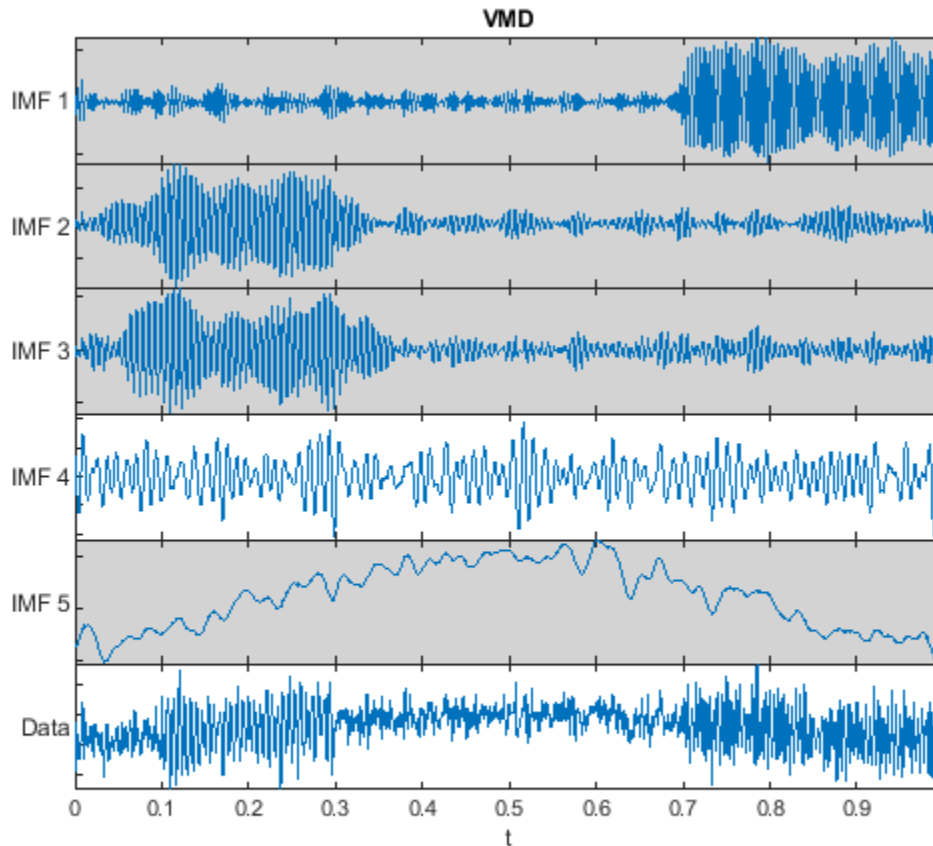
VMD is able to separate the two components. The high frequency oscillation localized to IMF 1, while the second component is spread over two adjacent IMFs. If you look at the estimated central frequencies of the VMD modes, the technique has localized the first two components around 200 and 150 Hz. The third IMF has a center frequency close to 150 Hz, which is why we see the second component in two MRA components.

```
info_vmd.CentralFrequencies*Fs
```

ans = *5×1*

```
  202.7204
  153.3278
  148.8022
   84.2802
    0.2667
```

VMD is able to do this because it starts by identifying candidate center frequencies for IMFs by looking at a frequency domain analysis of the data.

While the wavelet MRA is not able to separate the two high-frequency components, there is an additional *wavelet packet* MRA which can.

```
wpt = modwptdetails(x,3);
helperMRAPlot(x,flip(wpt),t,'wavelet','Wavelet Packet MRA',[5 6 8]);
```


Wavelet Packet MRA
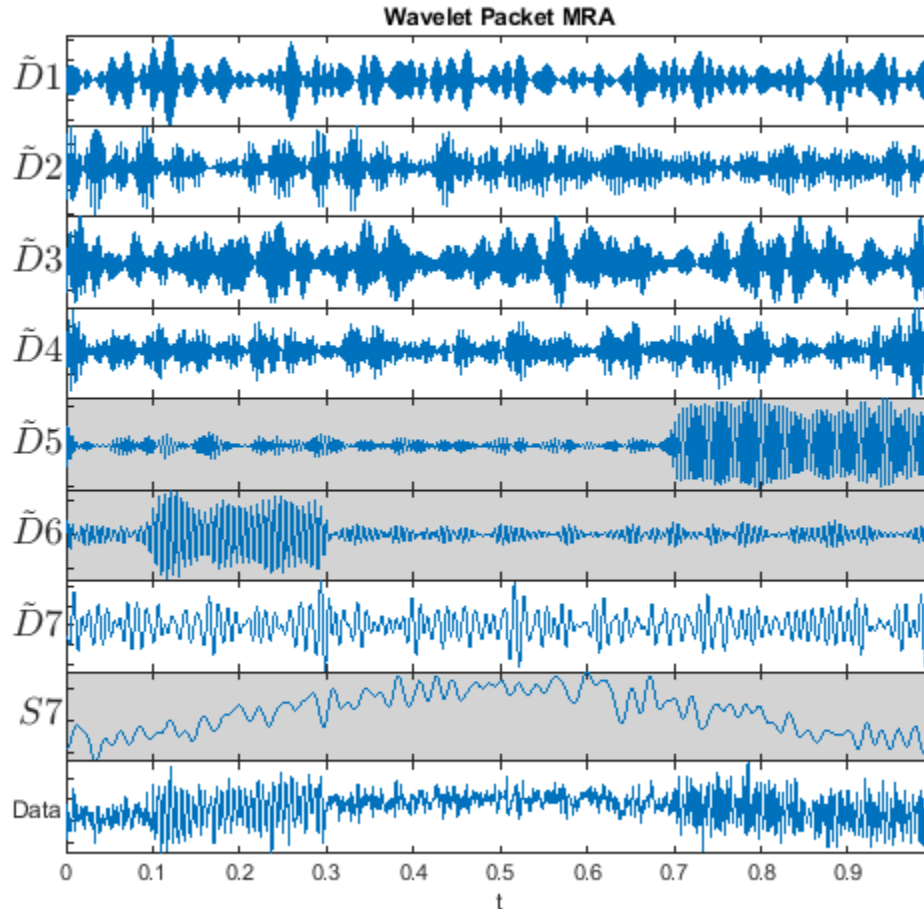
Now you see the two oscillations are separated in $\widetilde{D}5$ and $\widetilde{D}6$. From this example we can extract a general rule. If an initial wavelet or EMD decomposition shows components with apparently different rates of oscillation in the same component, consider VMD or a wavelet packet MRA. If you suspect that your data has high frequency components close together in frequency, VMD or a wavelet packet approach will generally work better than a wavelet or EMD approach.

Recall the problem of extracting a smooth trend. Repeat both the wavelet MRA and the EMD.

```
mra = modwtmra(modwt(x,8));
helperMRAPlot(x,mra,t,'wavelet','Wavelet MRA',[2 9])
```
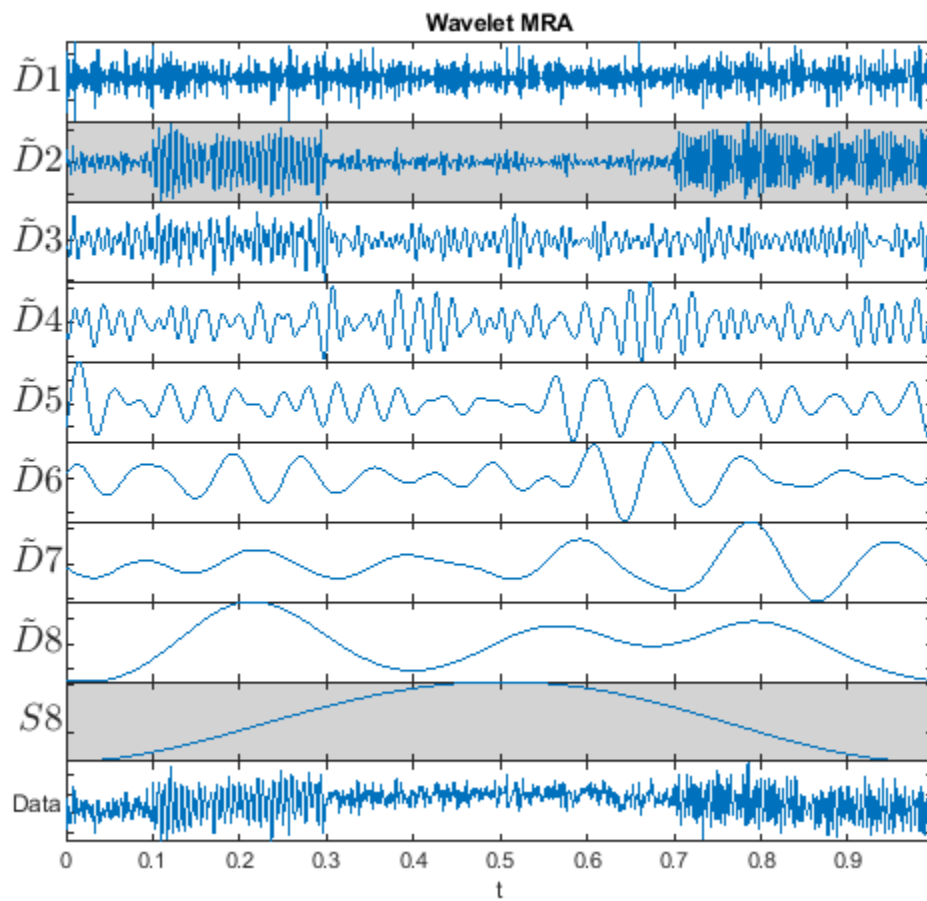
Wavelet MRA

```
imf_emd = emd(x);
helperMRAPlot(x,imf_emd,t,'EMD','Empirical Mode Decomposition',[1 2 6])
```

Empirical Mode Decomposition
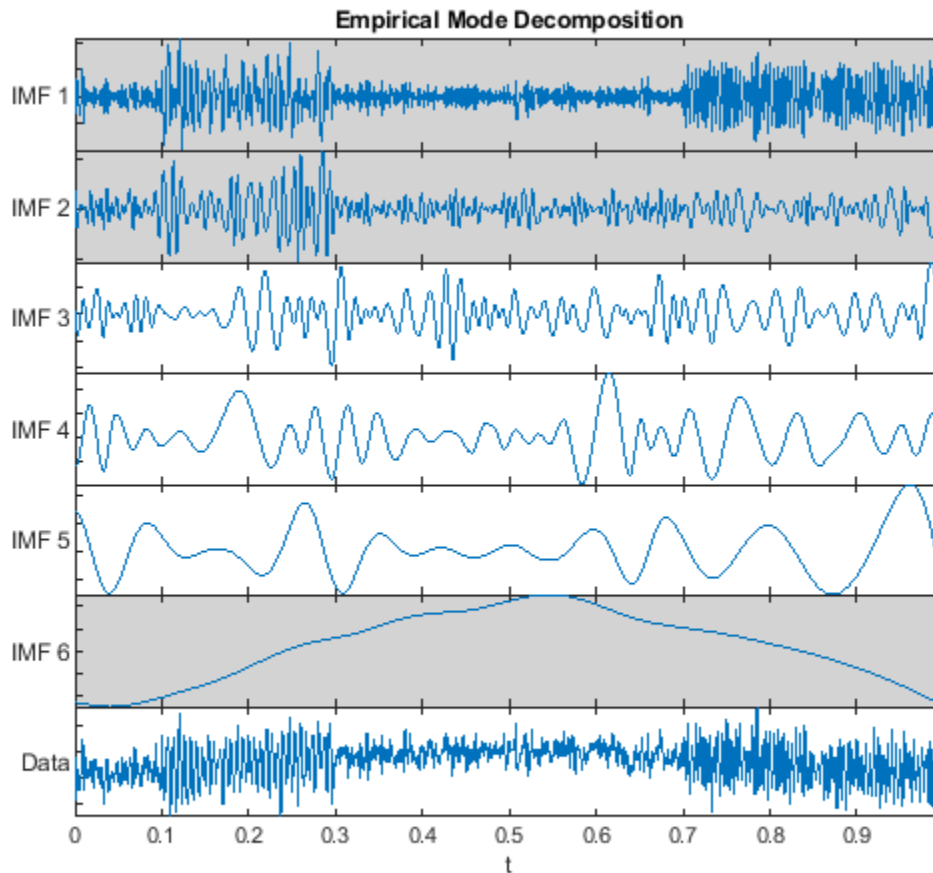
The trends extracted by the wavelet and EMD techniques are closer to the true trend than those extracted by VMD and the wavelet packet technique. VMD is inherently biased toward finding narrowband oscillatory components. This is a strength of VMD in detecting closely spaced oscillations, but a disadvantage when extracting smooth trends in the data. The same is true of wavelet packet MRAs when the decomposition goes beyond a few levels. This leads to a second general recommendation. If you are interested in characterizing a smooth trend in your data for identification or removal, try a wavelet or EMD technique.

What about detecting transient changes as we saw in the GDP data? Let us repeat the GDP analysis using VMD.

```
[imf_vmd,~,vmd_info] = vmd(realgdp);
figure
subplot(2,1,1)
plot(year,realgdp)
title('Real GDP');
hold on
plot([year(146) year(146)],[-0.1 0.15],'k')
hold off
subplot(2,1,2)
plot(year,imf_vmd(:,1));
```

```
title('First VMD IMF');
hold on
plot([year(146) year(146)],[-0.02 0.02],'k')
hold off
```

Real GDP

First VMD IMF
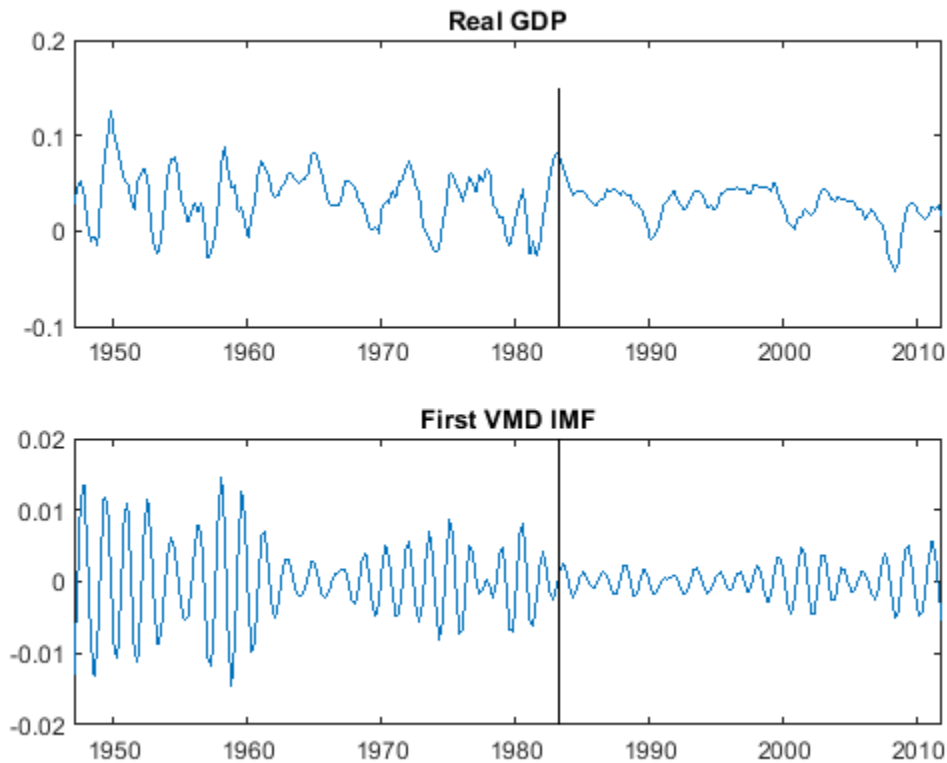
While the highest-frequency VMD component also appears to show some reduction in variability beginning in the mid-1980s, is it not as readily apparent as in the wavelet MRA. Because the VMD technique is biased toward finding oscillations, there is substantial ringing in the first VMD IMF which obscures the volatility changes.

Repeat this analysis using EMD.

```
imf_emd = emd(realgdp);
figure
subplot(2,1,1)
plot(year,realgdp)
title('Real GDP');
hold on
plot([year(146) year(146)],[-0.1 0.15],'k')
hold off
subplot(2,1,2)
plot(year,imf_emd(:,1));
title('First EMD IMF');
hold on
plot([year(146) year(146)],[-0.06 0.05],'k')
hold off
```
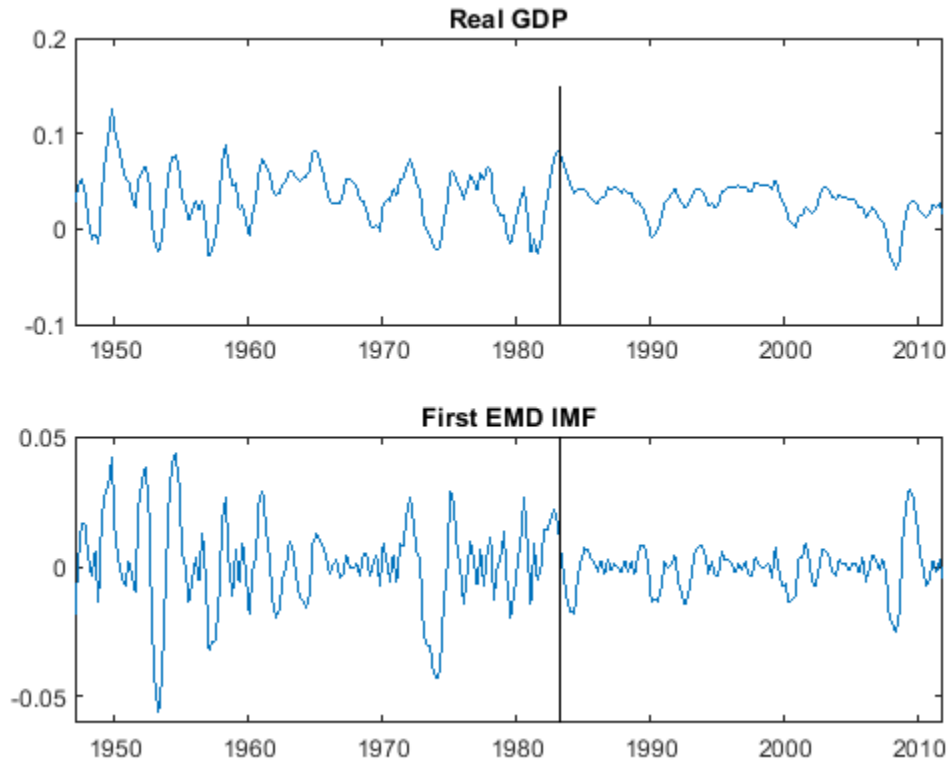
The EMD technique is less useful in finding the change in volatility (variance). In this case, the fixed functions using in the wavelet MRA were more advantageous than the data adaptive techniques.

This leads to our final general rule. If you are interested in detecting transient changes in a signal like impulsive events are reductions and increases in variability, try a wavelet technique.

Conclusions

This example showed how multiresolution decomposition techniques such as wavelet, wavelet packet, empirical mode decomposition, and variational mode decomposition allow you to study signal components in relative isolation on the same time scale as the original data. Each technique has proven itself powerful in a number of applications. The example has given a few rules of thumb to get you started, but these should not be regarded as absolute. The following table recaps properties of the MRA techniques presented here along with some general rules of thumb. Properties are denoted in blue and recommendations in orange.

Method	Fixed Bandwidth	Data-Adaptive	Separate Closely Spaced Frequencies	Localize Transient Changes	Extract Smooth Trend
Wavelet	✓			✓	✓
Wavelet Packet	✓		✓		
EMD		✓			✓
VMD		✓	✓		

- If your data appears to contain oscillatory components close in frequency, try VMD or wavelet packets. VMD identifies the important center frequencies directly from the data, while wavelet packets use a fixed frequency analysis, which may be less flexible than VMD.
- If you are interested in transient events in your data like impulsive events or transient reductions and increases in variability, try a wavelet MRA. In any MRA, these events are usually localized to the finest scale (highest center frequency) MRA components.
- If you are interested in representing a smooth trend term in the data, consider EMD or a wavelet MRA.

With **Signal Multiresolution Analyzer**, you can perform multiresolution analysis on a signal, obtain metrics on various MRA components, experiment with partial reconstructions, and generate MATLAB scripts to reproduce the analysis at the command line.

References

[1] Dragomiretskiy, Konstantin, and Dominique Zosso. "Variational Mode Decomposition." *IEEE Transactions on Signal Processing* 62, no. 3 (February 2014): 531–44. https://doi.org/10.1109/TSP.2013.2288675.

[2] Flandrin, P., G. Rilling, and P. Goncalves. "Empirical Mode Decomposition as a Filter Bank." *IEEE Signal Processing Letters* 11, no. 2 (February 2004): 112–14. https://doi.org/10.1109/LSP.2003.821662.

[3] Huang, Norden E., Zheng Shen, Steven R. Long, Manli C. Wu, Hsing H. Shih, Quanan Zheng, Nai-Chyuan Yen, Chi Chao Tung, and Henry H. Liu. "The Empirical Mode Decomposition and the Hilbert Spectrum for Nonlinear and Non-Stationary Time Series Analysis." *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454, no. 1971 (March 8, 1998): 903–95. https://doi.org/10.1098/rspa.1998.0193.

[4] Percival, Donald B., and Andrew T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge ; New York: Cambridge University Press, 2000.

See Also

Functions
emd | modwptdetails | modwt | modwtmra | vmd

Apps
Signal Multiresolution Analyzer

More About

- "Practical Introduction to Continuous Wavelet Analysis" on page 10-2
- "Continuous and Discrete Wavelet Transforms"

Wavelet Analysis for 3D Data

This example shows how to analyze 3D data using the three-dimensional wavelet analysis tool, and how to display low-pass and high-pass components along a given slice. The example focuses on magnetic resonance images.

A key feature of this analysis is to track the optimal, or at least a good, wavelet-based sparsity of the image which is the lowest percentage of transform coefficients sufficient for diagnostic-quality reconstruction.

To illustrate this, we keep the approximation of a 3D MRI to show the complexity reduction. The result can be improved if the images were transformed and reconstructed from the largest transform coefficients where the definition of the quality is assessed by medical specialists.

We will see that Wavelet transform for brain images allows efficient and accurate reconstructions involving only 5-10% of the coefficients.

Load and Display 3D MRI Data

First, load the `wmri.mat` file which is built from the MRI data set that comes with MATLAB®.

```
load wmri
```

We now display some slices along the Z-orientation of the original brain data.

```
map = pink(90);
idxImages = 1:3:size(X,3);
figure('DefaultAxesXTick',[],'DefaultAxesYTick',[],...
    'DefaultAxesFontSize',8,'Color','w')
colormap(map)
for k = 1:9
    j = idxImages(k);
    subplot(3,3,k);
    image(X(:,:,j));
    xlabel(['Z = ' int2str(j)]);
    if k==2
        title('Some slices along the Z-orientation of the original brain data');
    end
end
```

Some slices along the Z-orientation of the original brain data
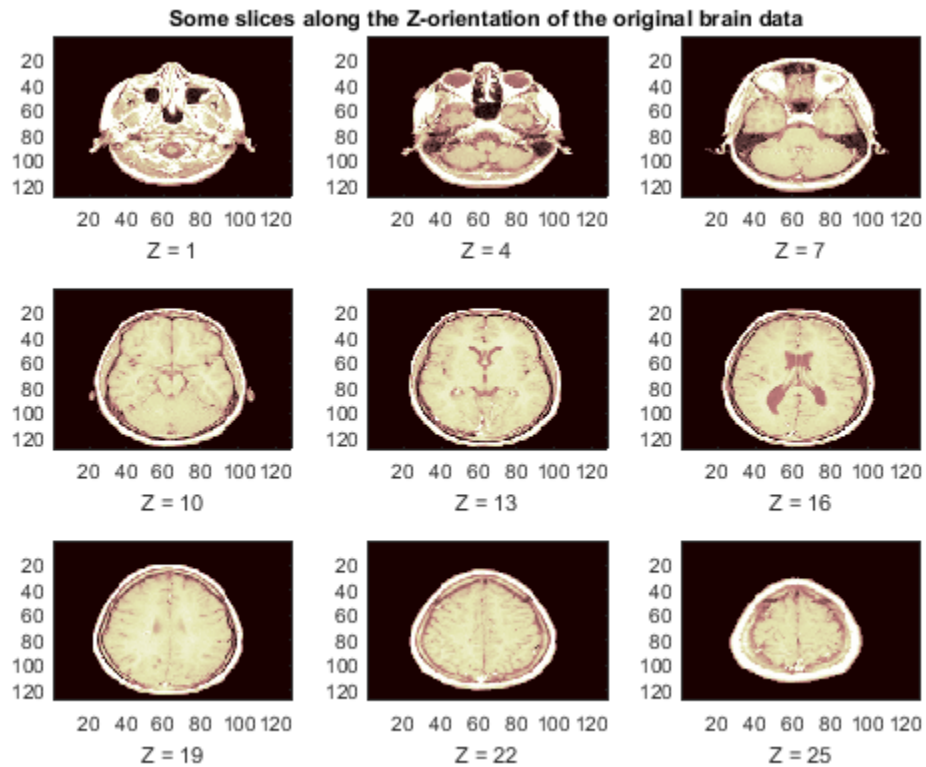
We now switch to the Y-orientation slice.

```
perm = [1 3 2];
XP = permute(X,perm);
figure('DefaultAxesXTick',[],'DefaultAxesYTick',[],...
    'DefaultAxesFontSize',8,'Color','w')
colormap(map)
for k = 1:9
    j = idxImages(k);
    subplot(3,3,k);
    image(XP(:,:,j));
    xlabel(['Y = ' int2str(j)]);
    if k==2
        title('Some slices along the Y-orientation');
    end
end
```

Some slices along the Y-orientation
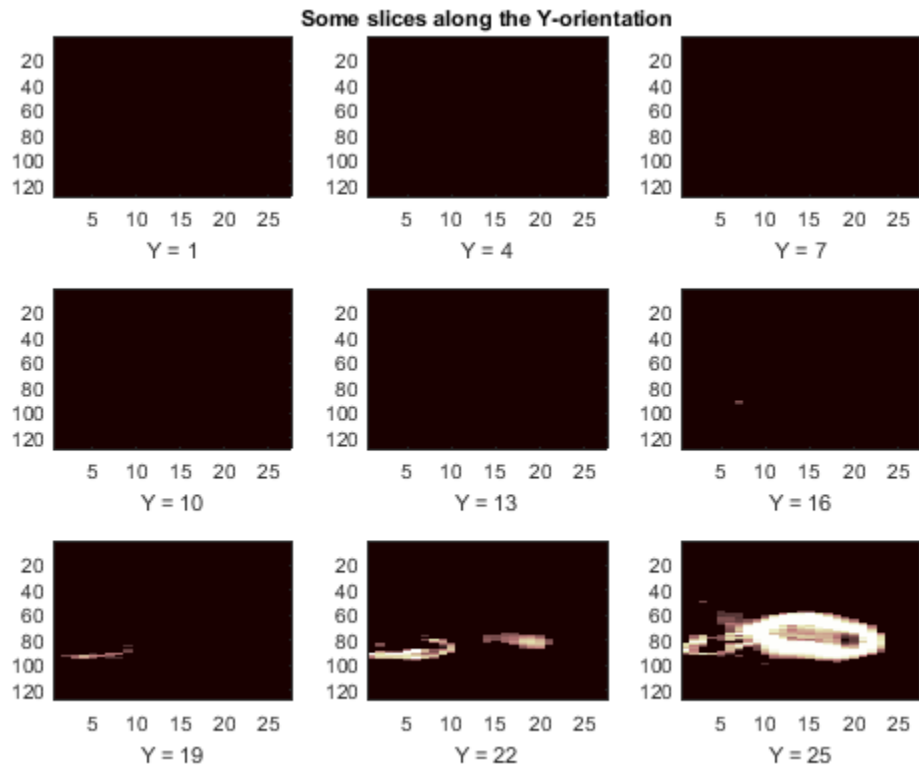
```
clear XP
```

Multilevel 3D Wavelet Decomposition

Compute the wavelet decomposition of the 3D data at level 3.

```
n = 3;                      % Decomposition Level
w = 'sym4';                 % Near symmetric wavelet
WT = wavedec3(X,n,w);       % Multilevel 3D wavelet decomposition.
```

Multilevel 3D Wavelet Reconstruction

Reconstruct from coefficients the approximations and details for levels 1 to 3.

```
A = cell(1,n);
D = cell(1,n);
for k = 1:n
    A{k} = waverec3(WT,'a',k);   % Approximations (low-pass components)
    D{k} = waverec3(WT,'d',k);   % Details (high-pass components)
end
```

Check for perfect reconstruction.

```
err = zeros(1,n);
for k = 1:n
    E = double(X)-A{k}-D{k};
    err(k) = max(abs(E(:)));
end
disp(err)
```

```
     1.0e-09 *

      0.3043     0.3043     0.3043
```

Display Low-Pass and High-Pass Components

The reconstructed approximations and details along the Z-orientation are displayed below.

```
nbIMG = 6;
idxImages_New = [1 7 10 16 19 25];
for ik = 1:nbIMG
    j = idxImages_New(ik);
    figure('DefaultAxesXTick',[],'DefaultAxesYTick',[],...
        'DefaultAxesFontSize',8,'Color','w')
    colormap(map)
    for k = 1:n
        labstr = [int2str(k) ' - Z = ' int2str(j)];
        subplot(2,n,k);
        image(A{k}(:,:,j));
        xlabel(['A' labstr]);
        if k==2
            title(['Approximations and details at level 1 to 3 - Slice = ' num2str(j)]);
        end
        subplot(2,n,k+n);
        imagesc(abs(D{k}(:,:,j)));
        xlabel(['D' labstr]);
    end
end
```


Approximations and details at level 1 to 3 - Slice = 1

Approximations and details at level 1 to 3 - Slice = 7

Approximations and details at level 1 to 3 - Slice = 10

Approximations and details at level 1 to 3 - Slice = 16

Approximations and details at level 1 to 3 - Slice = 19

Approximations and details at level 1 to 3 - Slice = 25
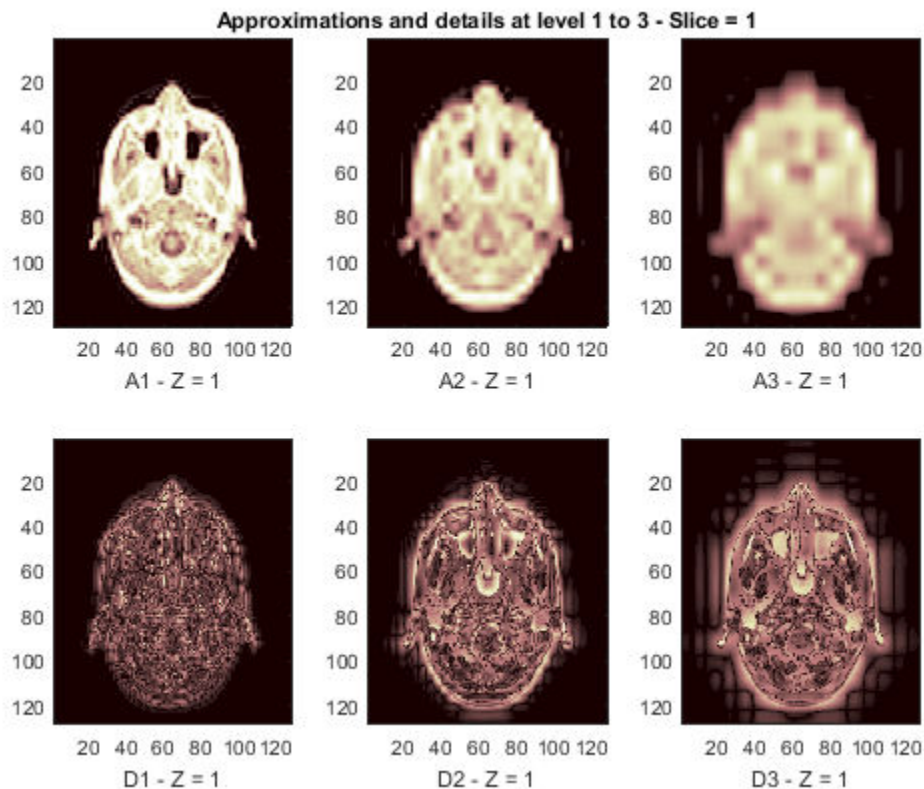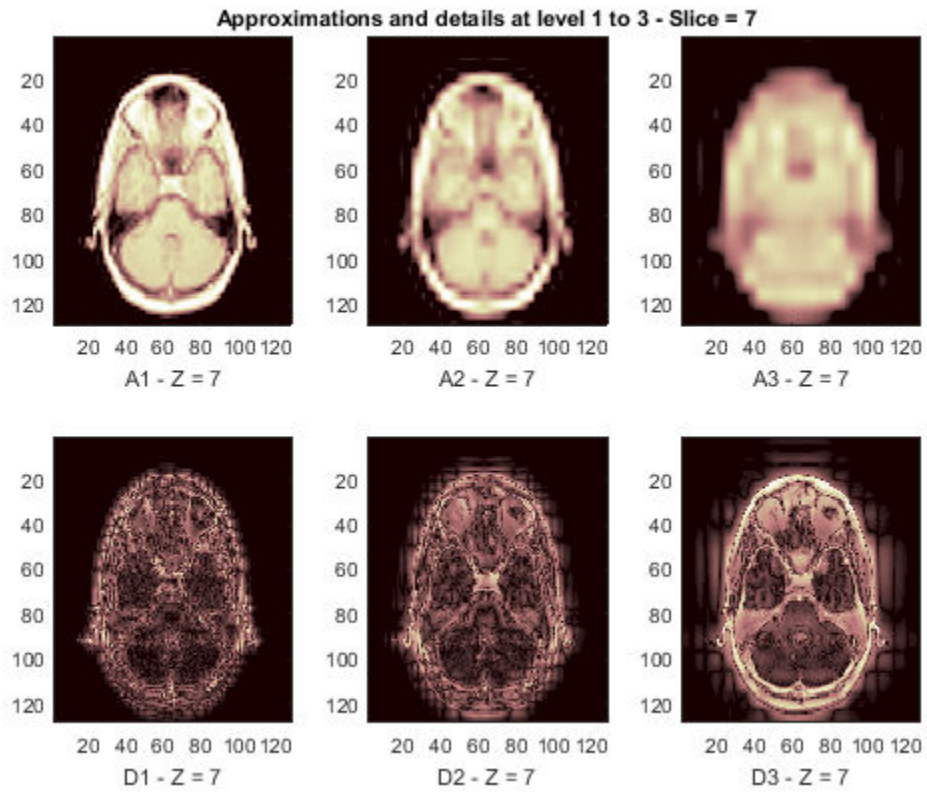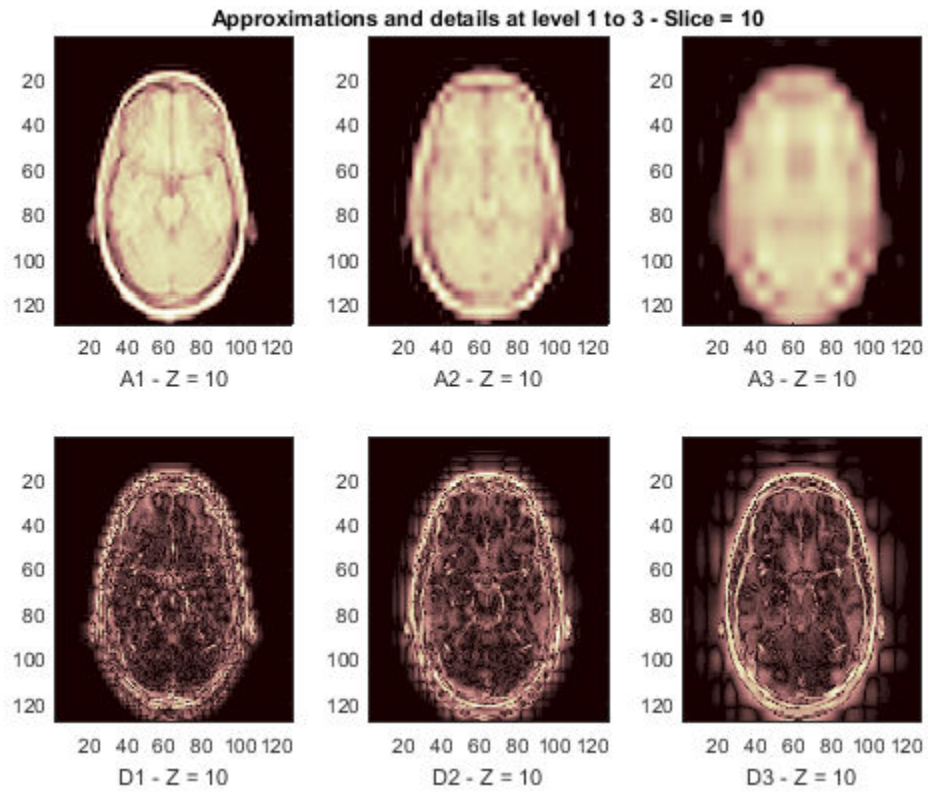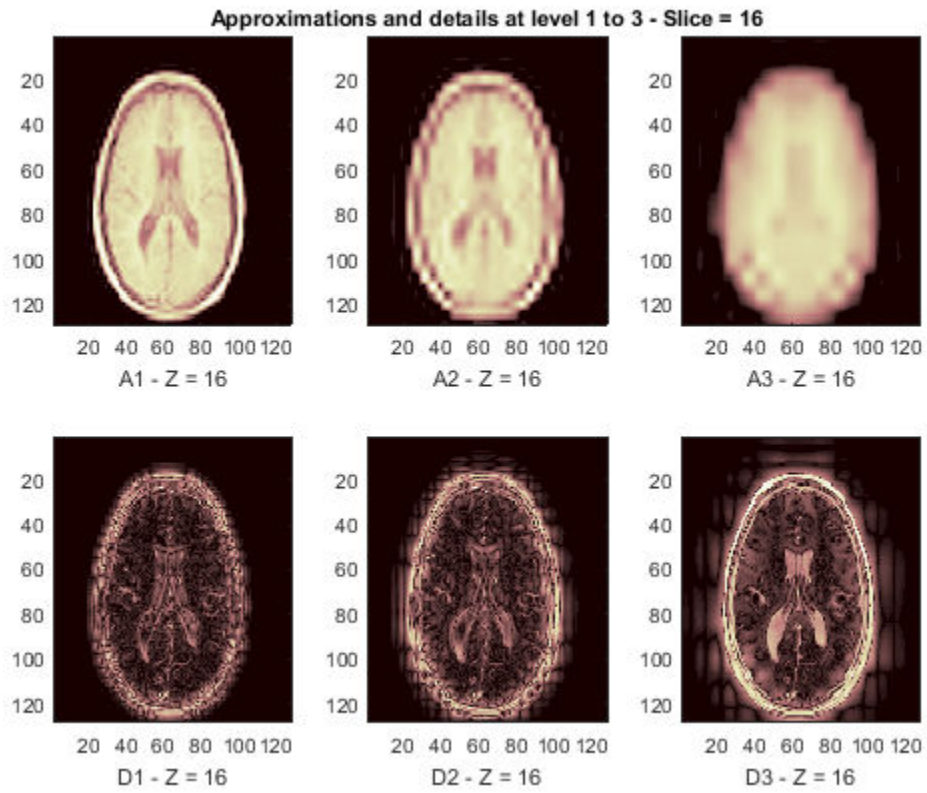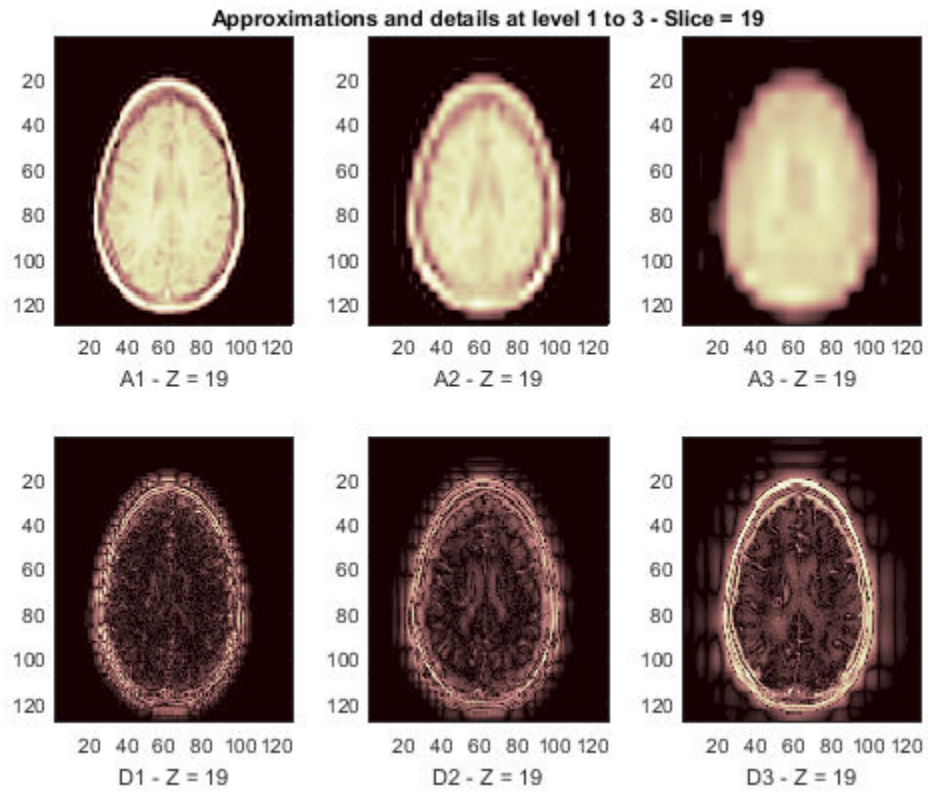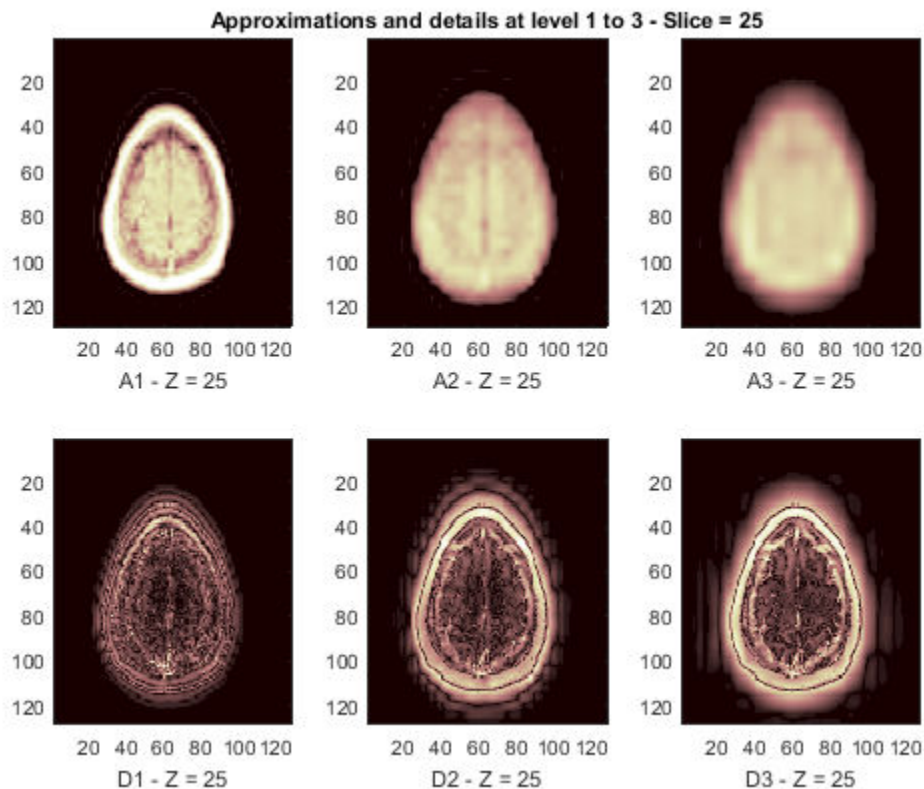
3D Display of Original Data and Approximation at Level 2

The size of the 3D original array X is **(128 x 128 x 27) = 442368**. We can use a 3D display to show it.

```
figure('DefaultAxesXTick',[],'DefaultAxesYTick',[],...
       'DefaultAxesFontSize',8,'Color','w')
XR = X;
Ds = smooth3(XR);
hiso = patch(isosurface(Ds,5),'FaceColor',[1,.75,.65],'EdgeColor','none');
hcap = patch(isocaps(XR,5),'FaceColor','interp','EdgeColor','none');
colormap(map)
daspect(gca,[1,1,.4])
lightangle(305,30);
fig = gcf;
fig.Renderer = 'zbuffer';
lighting phong
isonormals(Ds,hiso)
hcap.AmbientStrength = .6;
hiso.SpecularColorReflectance = 0;
hiso.SpecularExponent = 50;
ax = gca;
ax.View = [215,30];
ax.Box = 'On';
axis tight
title('Original Data');
```
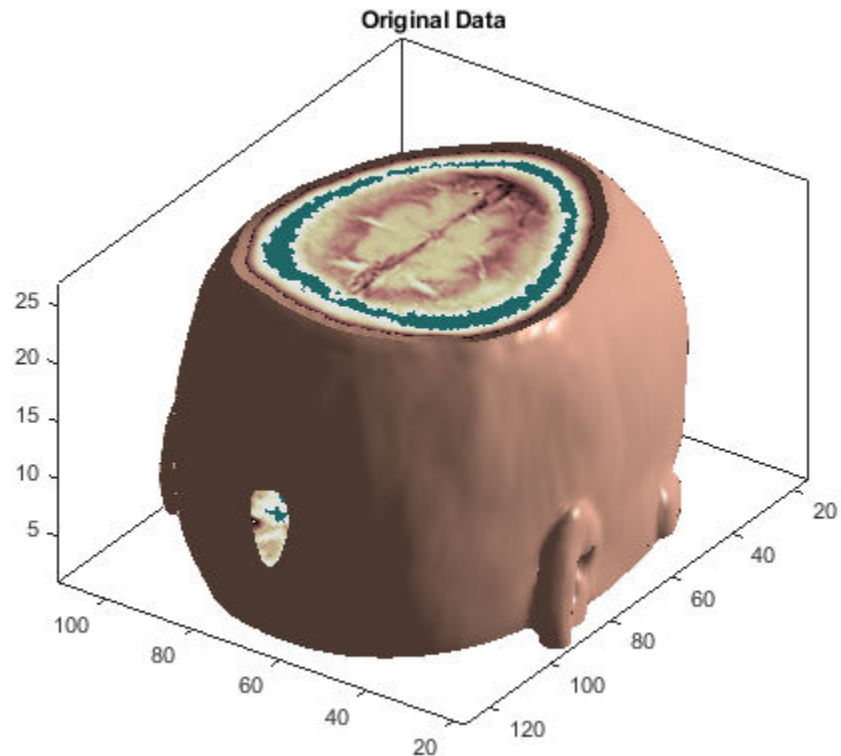
Original Data

The 3D array of the coefficients of approximation at level 2, whose size is **(22 x 22 x 9) = 4356**, is less than 1% the size of the original data. With these coefficients, we can reconstruct A2, the approximation at level 2, which is a kind of compression of the original 3D array. A2 can also be shown using a 3D display.

```
figure('DefaultAxesXTick',[],'DefaultAxesYTick',[],...
       'DefaultAxesFontSize',8,'Color','w')
XR = A{2};
Ds = smooth3(XR);
hiso = patch(isosurface(Ds,5),'FaceColor',[1,.75,.65],'EdgeColor','none');
hcap = patch(isocaps(XR,5),'FaceColor','interp','EdgeColor','none');
colormap(map)
daspect(gca,[1,1,.4])
lightangle(305,30);
fig = gcf;
fig.Renderer = 'zbuffer';
lighting phong
isonormals(Ds,hiso)
hcap.AmbientStrength = .6;
hiso.SpecularColorReflectance = 0;
hiso.SpecularExponent = 50;
ax = gca;
ax.View = [215,30];
ax.Box = 'On';
axis tight
title('Approximation at level 2');
```

Approximation at level 2
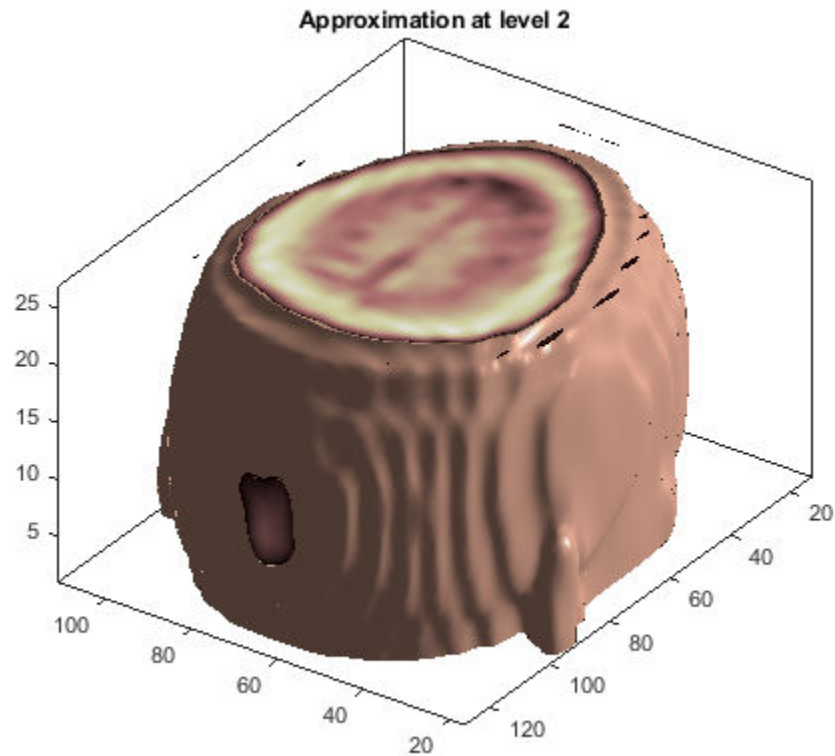
Summary

This example shows how to use 3D wavelet functions to analyze 3D data. The Wavelet Analyzer app, `waveletAnalyzer`, lets you perform the same steps more easily, such as simulating 3D visualization using an animation across different slices.

Multisignal 1-D Wavelet Analysis

A 1-D multisignal is a set of 1-D signals of same length stored as a matrix organized rowwise (or columnwise).

The purpose of this example is to show how to analyze, denoise or compress a multisignal, and then to cluster different representations or simplified versions of the signals composing the multisignal.

In this example, the signals are first analyzed and different representations or simplified versions are produced:

- Reconstructed approximations at given levels,

- Denoised versions,

- Compressed versions.

Denoising and compressing are two of the main applications of wavelets, often used as a preprocessing step before clustering.

The last step of this example performs several clustering strategies and compare them. It allows to summarize a large set of signals using sparse wavelet representations.

Load and Plot a Multisignal

To illustrate these capabilities, let us consider a real-life multisignal representing 35 days of an electrical load consumption, centered and standardized.

```
load elec35_nor;        % Load normalized original data.
X = signals;
[nbSIG,nbVAL] = size(X);
plot(X','r')
axis tight
title('Original Data: 35 days of electrical consumption');
xlabel('Minutes from 1 to 1440');
```
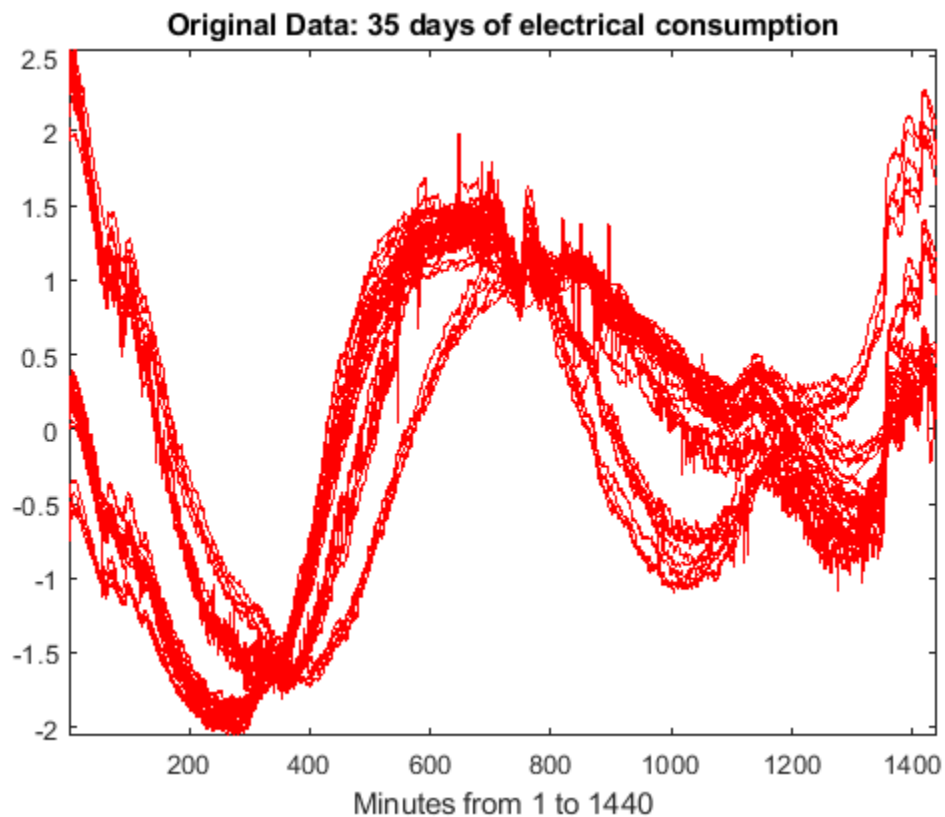
We can see that the signals are locally irregular and noisy but nevertheless three different general shapes can be distinguished.

Surface Representation of Data

In order to highlight the periodicity of the multisignal, let us now examine the data using a 3-D representation.

```
surf(X)
shading interp
axis tight
title('Original Data: 35 days of electrical consumption')
xlabel('Minutes from 1 to 1440','Rotation',4)
ylabel('Days from 1 to 35','Rotation',-60)
ax = gca;
ax.View = [-13.5 48];
```

Original Data: 35 days of electrical consumption
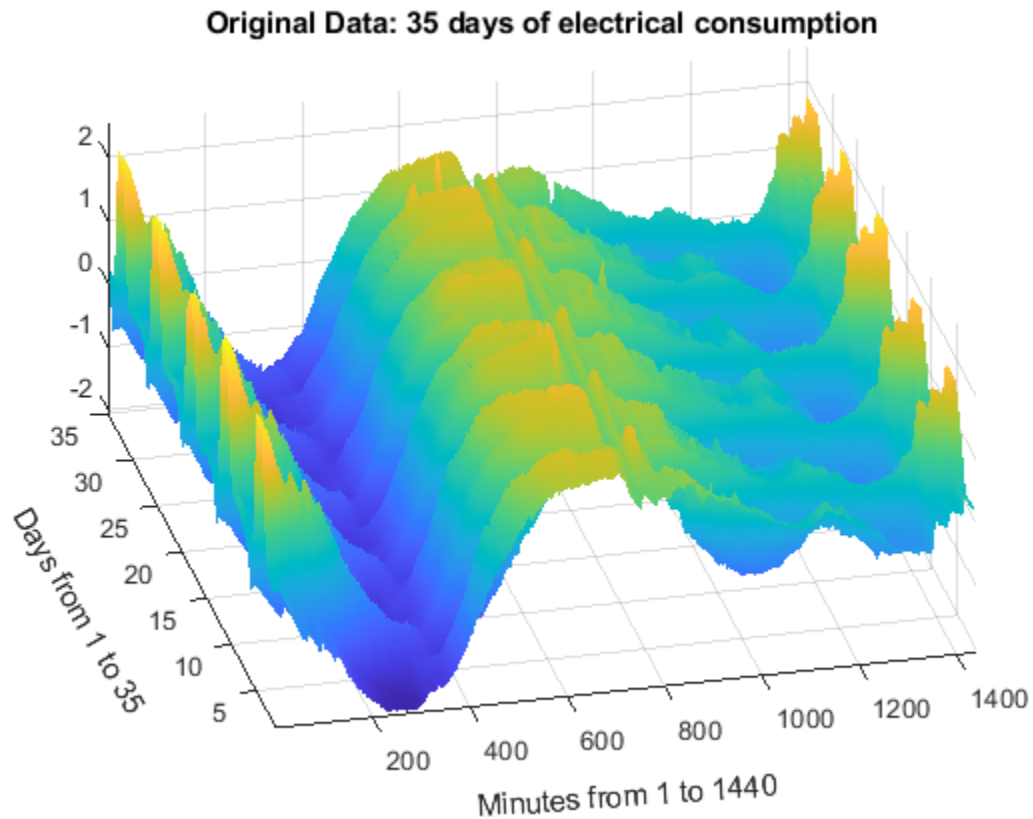
The five weeks represented can now be seen more clearly.

Multisignal Row Decomposition

Perform a wavelet decomposition at level 7 using the ' *sym4* ' wavelet.

```
dirDec = 'r';          % Direction of decomposition
level  = 7;            % Level of decomposition
wname  = 'sym4';       % Near symmetric wavelet
decROW = mdwtdec(dirDec,X,level,wname);
```

The generated decomposition structure is as follows:

```
decROW
```

```
decROW =

  struct with fields:

        dirDec: 'r'
         level: 7
         wname: 'sym4'
    dwtFilters: [1x1 struct]
       dwtEXTM: 'sym'
       dwtShift: 0
       dataSize: [35 1440]
            ca: [35x18 double]
```

11-41

```
                    cd: {1x7 cell}
```

Signals and Approximations at Level 7

Let us reconstruct the approximations at level 7 for each row signal. Then, to compare the approximations with the original signals, let us display two plots (see figure below). The first one shows all the original signals and the second one shows all the corresponding approximations at level 7.

```
A7_ROW  = mdwtrec(decROW,'a',7);

subplot(2,1,1)
plot(X(:,:)','r')
title('Original Data')
axis tight
subplot(2,1,2)
plot(A7_ROW(:,:)','b')
axis tight
title('Corresponding approximations at level 7')
```
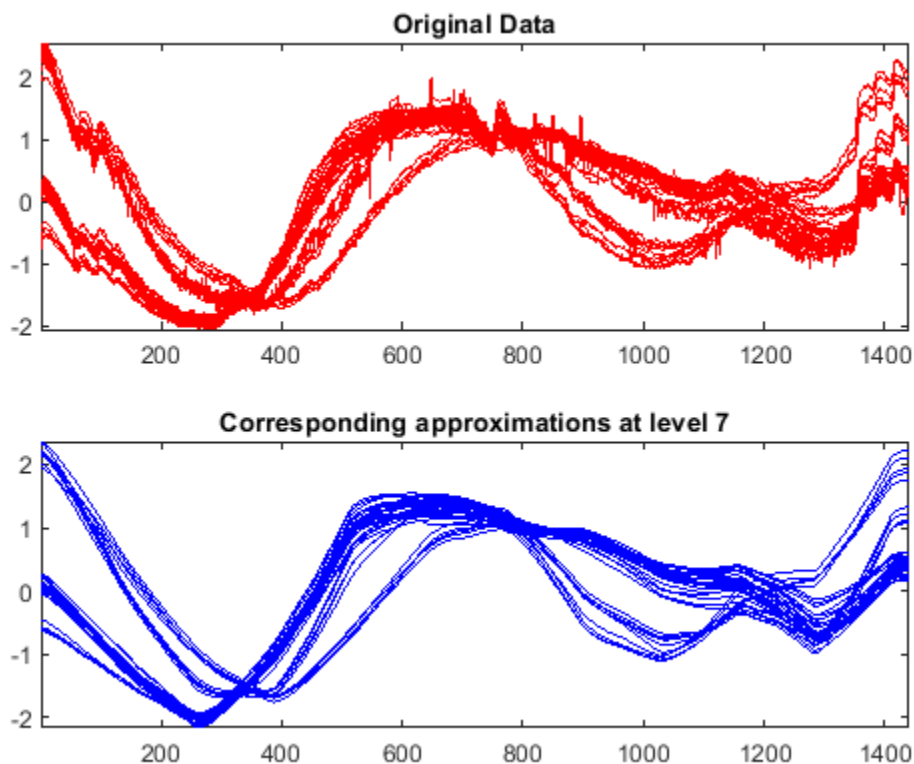

As it can be seen, the general shape is captured by the approximations at level 7, but some interesting features are lost. For example, the bumps at the beginning and at the end of the signals disappeared.

Superimposed Signals and Approximations

To compare more closely the original signals with their corresponding approximations at level 7, let us display two plots (see figure below). The first one concerns the four first signals and the second one the five last signals. Each of these plots represents the original signals superimposed with their corresponding approximation at level 7.

```
subplot(2,1,1)
idxDays = 1:4;
plot(X(idxDays,:)','r')
hold on
plot(A7_ROW(idxDays,:)','b')
axis tight
title(['Days ' int2str(idxDays), ' - Signals and Approximations at level 7'])
subplot(2,1,2)
idxDays = 31:35;
plot(X(idxDays,:)','r')
hold on
plot(A7_ROW(idxDays,:)','b')
axis tight
title(['Days ' int2str(idxDays), ' - Signals and Approximations at level 7'])
```
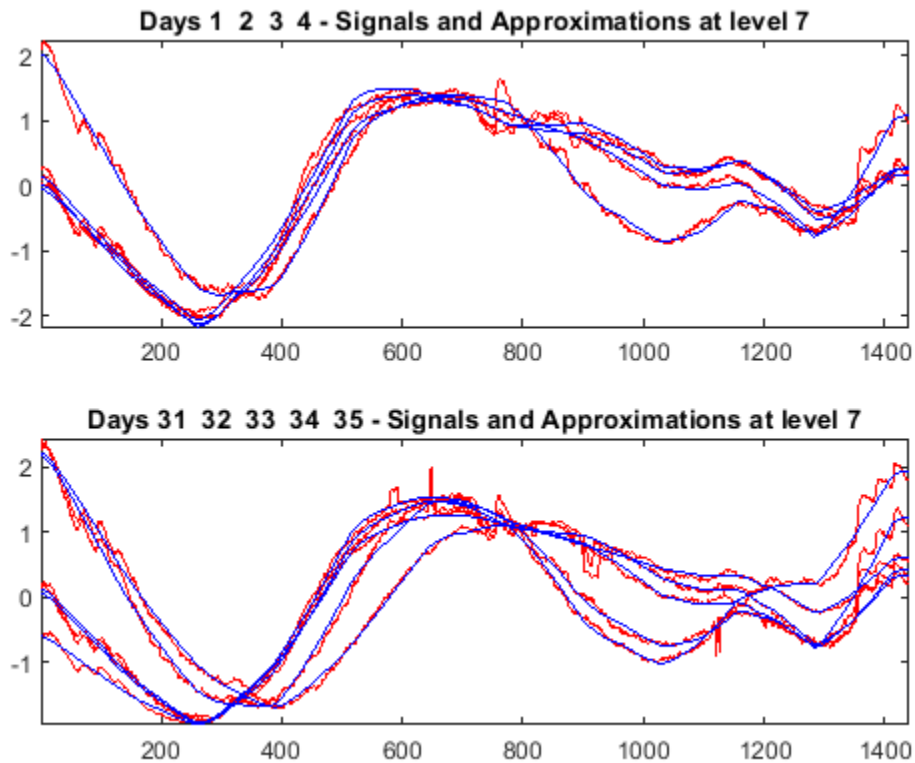

As it can be seen, the approximation of the original signal is visually accurate in terms of general shape.

11-43

Multisignal Denoising

To perform a more subtle simplification of the multisignal preserving these bumps which are clearly attributable to the electrical signal, let us denoise the multisignal.

The denoising procedure is performed using three steps:

1) **Decomposition** : Choose a wavelet and a level of decomposition N, and then compute the wavelet decompositions of the signals at level N.

2) **Thresholding** : For each level from 1 to N and for each signal, a threshold is selected and thresholding is applied to the detail coefficients.

3) **Reconstruction** : Compute wavelet reconstructions using the original approximation coefficients of level N and the modified detail coefficients of levels from 1 to N.

Let us choose now the level of decomposition N = 5 instead of N = 7 used previously.

```matlab
dirDec = 'r';           % Direction of decomposition
level  = 5;             % Level of decomposition
wname  = 'sym4';        % Near symmetric wavelet
decROW = mdwtdec(dirDec,X,level,wname);

[XD,decDEN] = mswden('den',decROW,'sqtwolog','mln');
Residuals = X-XD;

subplot(3,1,1)
plot(X','r')
axis tight
title('Original Data: 35 days of electrical consumption')
subplot(3,1,2)
plot(XD','b')
axis tight
title('Denoised Data: 35 days of electrical consumption')
subplot(3,1,3)
plot(Residuals','k')
axis tight
title('Residuals')
xlabel('Minutes from 1 to 1440')
```

Original Data: 35 days of electrical consumption

Denoised Data: 35 days of electrical consumption

Residuals

Minutes from 1 to 1440
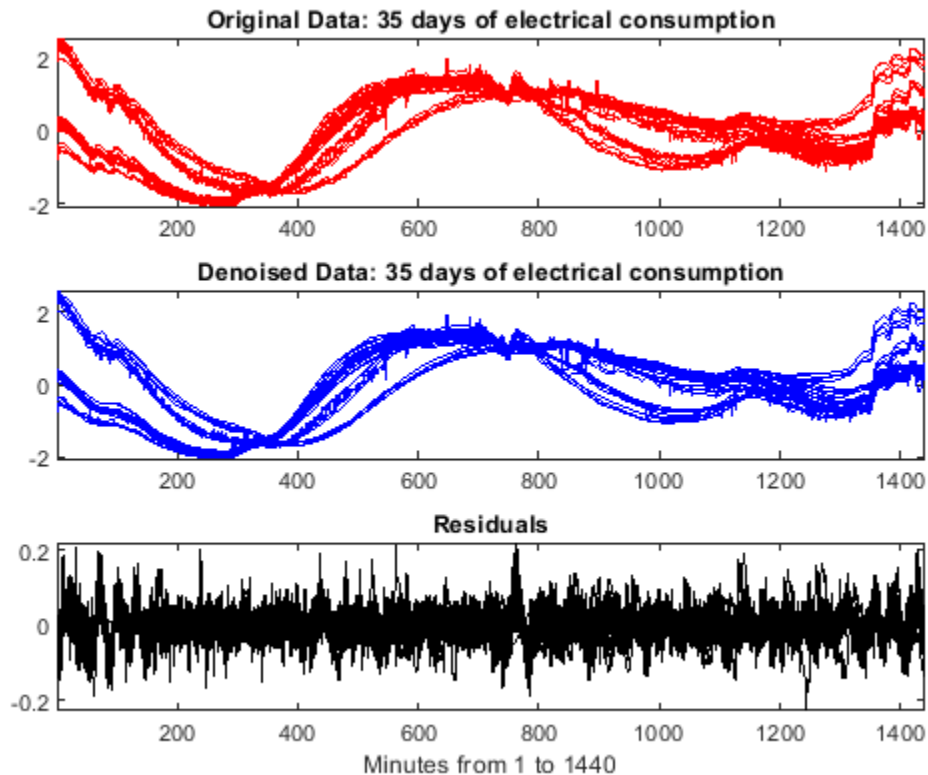
The quality of the results is good. The bumps at the beginning and at the end of the signals are well recovered and, conversely, the residuals look like a noise except for some remaining bumps due to the signals. Furthermore, the magnitude of these remaining bumps is of a small order.

Multisignal Compressing Row Signals

Like denoising, the compression procedure is performed using three steps (see above).

The difference with the denoising procedure is found in step 2. There are two compression approaches available:

- The first one consists of taking the wavelet expansions of the signals 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 signal-dependent parameter needs to be selected.

- The second approach consists of applying visually determined level-dependent thresholds.

To simplify the data representation and to make more efficient the compression, let us go back to the decomposition at level 7 and compress each row of the multisignal by applying a global threshold leading to recover 99% of the energy.

```
dirDec = 'r';          % Direction of decomposition
level  = 7;            % Level of decomposition
wname  = 'sym4';       % Near symmetric wavelet
decROW = mdwtdec(dirDec,X,level,wname);
```

11-45

```
[XC,decCMP,THRESH] = mswcmp('cmp',decROW,'L2_perf',99);

subplot(3,1,1)
plot(X','r')
axis tight
title('Original Data: 35 days of electrical consumption')
subplot(3,1,2)
plot(XC','b')
axis tight
title('Compressed Data: 35 days of electrical consumption')
subplot(3,1,3)
plot((X-XC)','k')
axis tight
title('Residuals')
xlabel('Minutes from 1 to 1440')
```
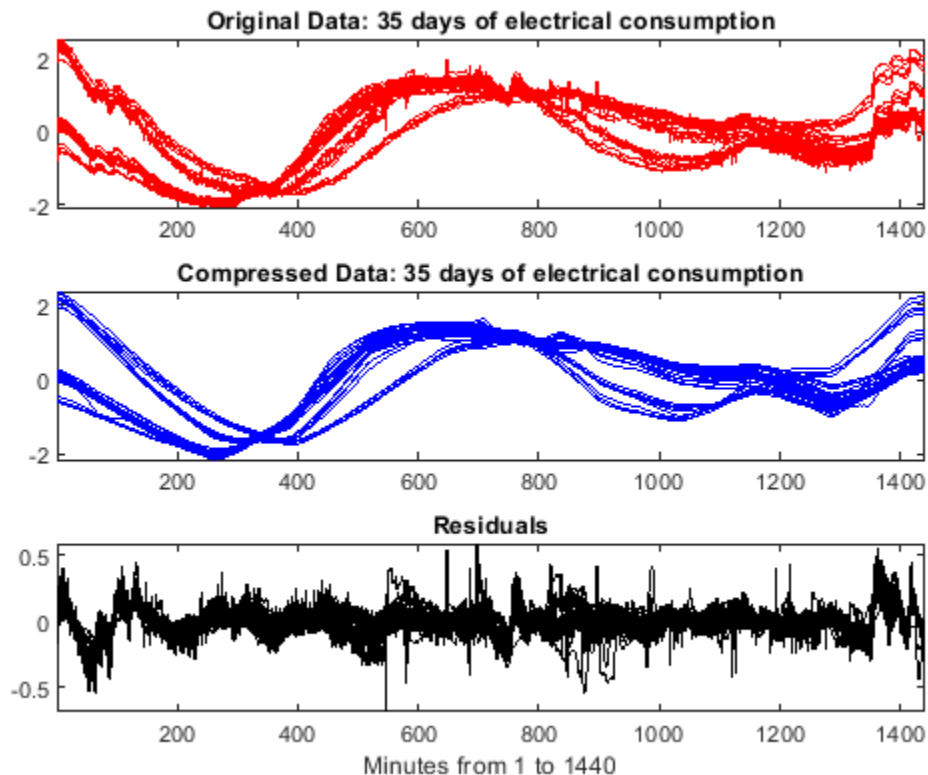
As it can be seen, the general shape is preserved while the local irregularities are neglected. The residuals contain noise as well as components due to the small scales essentially.

Compression Performance

Let us now compute the corresponding densities of nonzero elements.

```
cfs = cat(2,[decCMP.cd{:},decCMP.ca]);
cfs = sparse(cfs);
perf = zeros(1,nbSIG);
for k = 1:nbSIG
    perf(k) = 100*nnz(cfs(k,:))/nbVAL;
```

```
end

figure
plot(perf,'r-*')
title('Percentages of nonzero coefficients for the 35 days')
xlabel('Signal indices')
ylabel('% of nonzero coefficients')
```
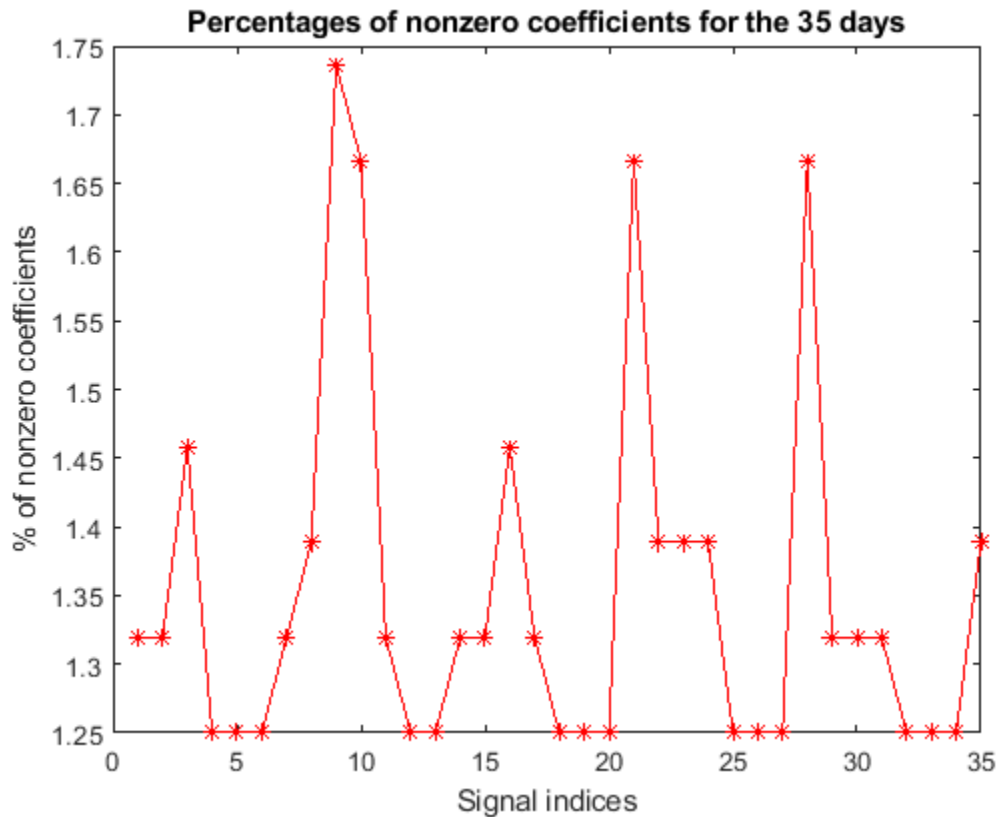

For each signal, the percentage of required coefficients to recover 99% of the energy lies between 1.25% and 1.75%. This illustrates the capacity of wavelets to concentrate signal energy in few coefficients.

Clustering Row Signals

Clustering offers a convenient procedure to summarize a large set of signals using sparse wavelet representations. You can implement hierarchical clustering using `mdwtcluster`. You must have the Statistics and Machine Learning Toolbox™ to use `mdwtcluster`.

Compare three different clusterings of the 35-day data. The first one is based on the original multisignal, the second one on the approximation coefficients at level 7, and the last one is based on the denoised multisignal.

Set to the number of clusters to 3. Compute the structures P1 and P2 which contain respectively the two first partitions and the third one.

```
P1 = mdwtcluster(decROW,'lst2clu',{'s','ca7'},'maxclust',3);
P2 = mdwtcluster(decDEN,'lst2clu',{'s'},'maxclust',3);
Clusters = [P1.IdxCLU P2.IdxCLU];
```

We can now test the equality of the three partitions.

```
EqualPART = isequal(max(diff(Clusters,[],2)),[0 0])
```

```
EqualPART =

  logical

   1
```

So the three partitions are the same. Let us now plot and examine the clusters.

```
figure
stem(Clusters,'filled','b:')
title('The three clusters of the original 35 days')
xlabel('Signal indices')
ylabel('Index of cluster')
ax = gca;
xlim([1 35])
ylim([0.5 3.1])
ax.YTick = 1:3;
```
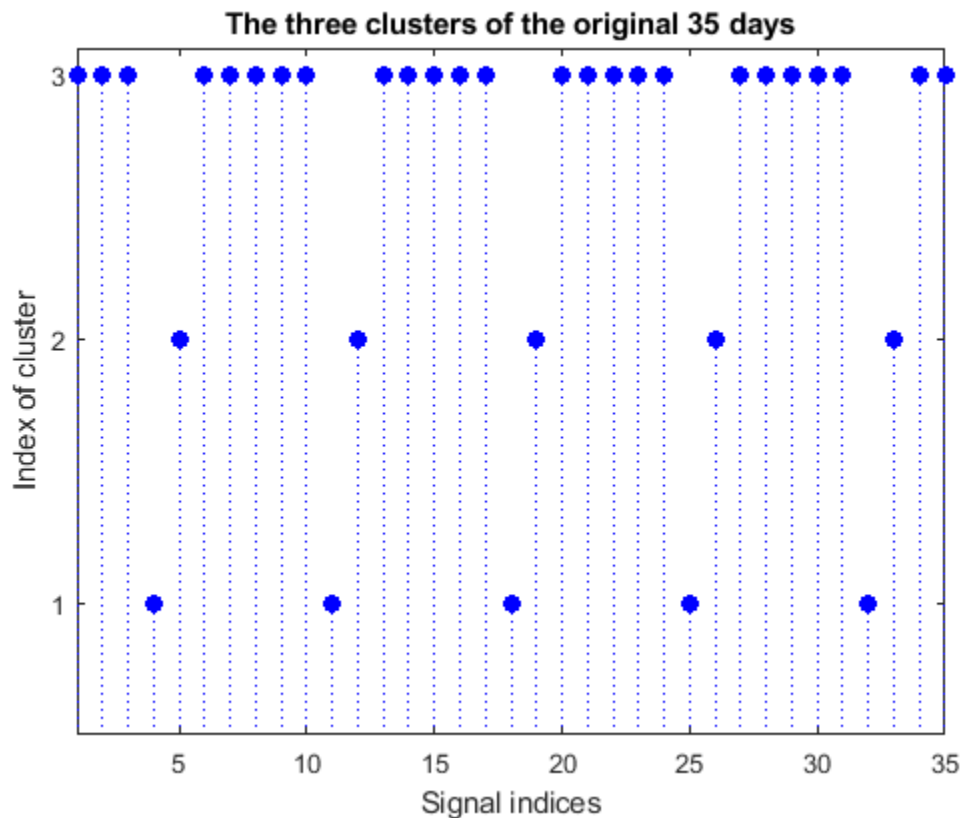
The first cluster (labelled 3) contains 25 mid-week days and the two others (labelled 2 and 1) contain five Saturdays and five Sundays respectively. This illustrates again the periodicity of the underlying time series and the three different general shapes seen in the two first plots displayed at the beginning of this example.

Let us now display the original signals and the corresponding approximation coefficients used to obtain two of the three partitions.

```matlab
CA7 = mdwtrec(decROW,'ca');
IdxInCluster = cell(1,3);
for k = 1:3
    IdxInCluster{k} = find(P2.IdxCLU==k);
end
figure('Units','normalized','Position',[0.2 0.2 0.6 0.6])
for k = 1:3
    idxK = IdxInCluster{k};
    subplot(2,3,k)
    plot(X(idxK,:)','r')
    axis tight
    ax = gca;
    ax.XTick = [200 800 1400];
    if k==2
        title('Original signals')
    end
    xlabel(['Cluster: ' int2str(k) ' (' int2str(length(idxK)) ')'])
    subplot(2,3,k+3)
    plot(CA7(idxK,:)','b')
    axis tight
    if k==2
        title('Coefficients of approximations at level 7')
    end
    xlabel(['Cluster: ' int2str(k) ' (' int2str(length(idxK)) ')'])
end
```

Original signals

Coefficients of approximations at level 7
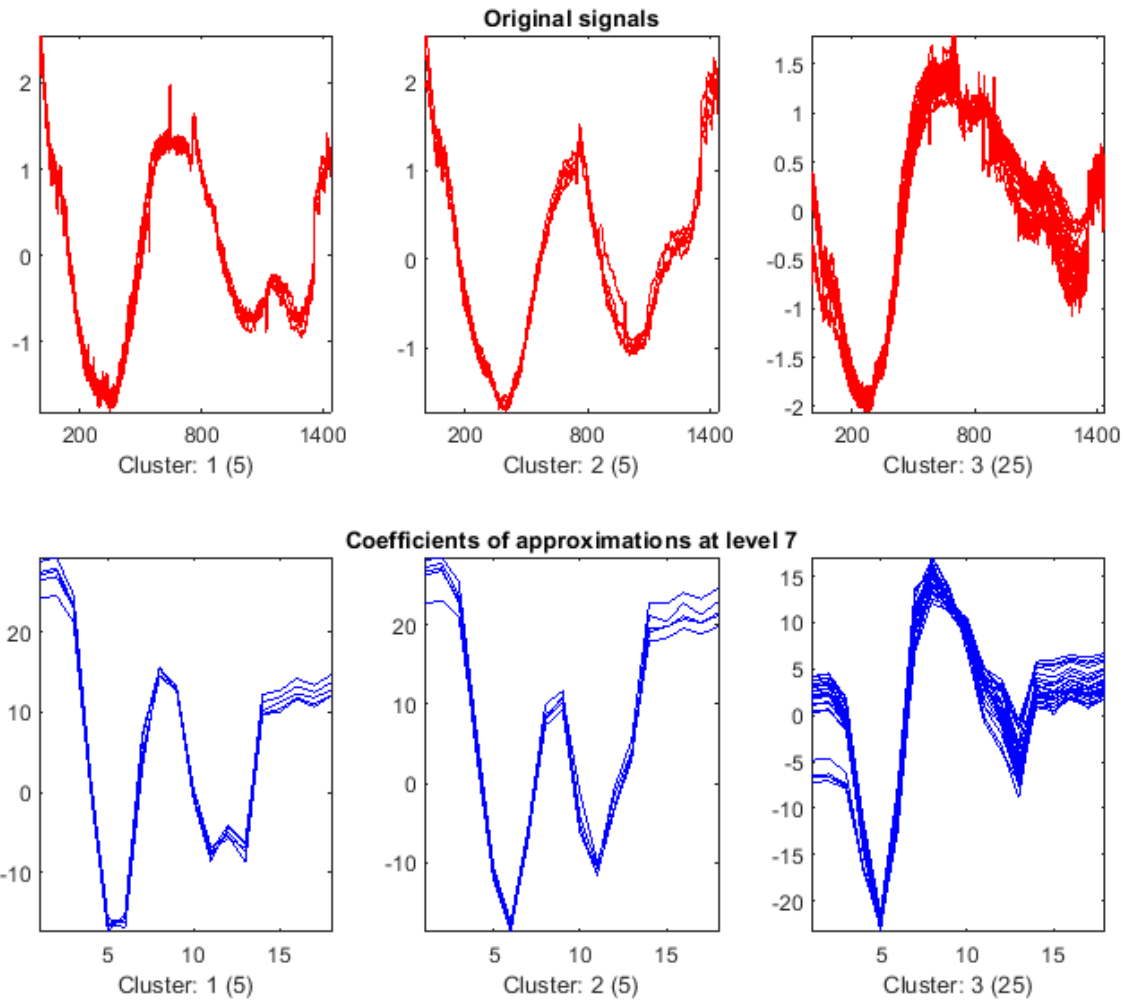
The same partitions are obtained from the original signals (1440 samples for each signal) and from the coefficients of approximations at level 7 (18 samples for each signal). This illustrates that using less than 2% of the coefficients is enough to get the same clustering partitions of the 35 days.

Summary

Denoising, compression and clustering using wavelets are very efficient tools. The capacity of wavelet representations to concentrate signal energy in few coefficients is the key of efficiency. In addition, clustering offers a convenient procedure to summarize a large set of signals using sparse wavelet representations.

Detecting Discontinuities and Breakdown Points

Signals with very rapid evolutions such as transient signals in dynamic systems may undergo abrupt changes such as a jump, or a sharp change in the first or second derivative. Fourier analysis is usually not able to detect those events. The purpose of this example is to show how analysis by wavelets can detect the exact instant when a signal changes and also the type (a rupture of the signal, or an abrupt change in its first or second derivative) and amplitude of the change. In image processing, one of the major applications is edge detection, which also involves detecting abrupt changes.

Frequency Breakdown

Short wavelets are often more effective than long ones in detecting a signal rupture. Therefore, to identify a signal discontinuity, we will use the Haar wavelet. The discontinuous signal consists of a slow sine wave abruptly followed by a medium sine wave.

```
load freqbrk
x = freqbrk;
```

Compute the multilevel 1-D wavelet decomposition at level 1.

```
level = 1;
[c,l] = wavedec(x,level,'haar');
```

Extract the detail coefficients at level 1 from the wavelet decomposition and visualize the result. Interpolate the detail coefficients so they and the signal have the same length.

```
d1 = detcoef(c,l,level);
subplot(2,1,1)
plot(x)
subplot(2,1,2)
plot(interpft(d1,2*length(d1)))
ylabel('d1')
```
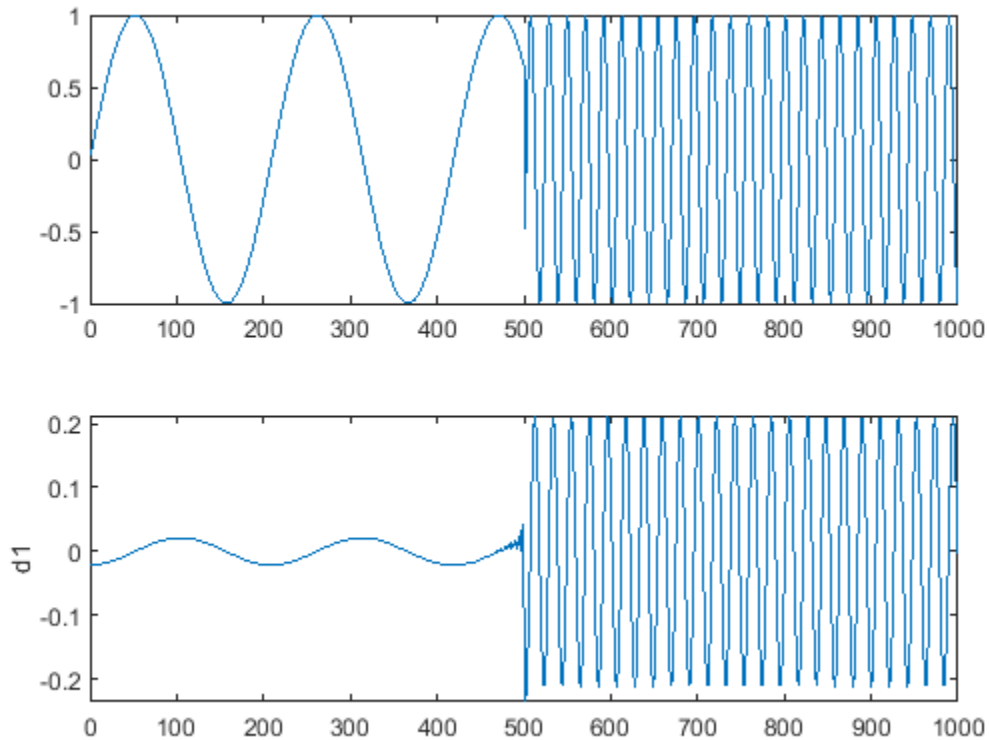
The first-level details (d1) show the discontinuity most clearly, because the rupture contains the high-frequency part. The discontinuity is localized very precisely around time = 500.

The presence of noise makes identification of discontinuities more complicated. If the finer levels of the decomposition can be used to eliminate a large part of the noise, the rupture is sometimes visible at coarser levels in the decomposition.

Second Derivation Breakdown

The purpose of this example is to show how analysis by wavelets can detect a discontinuity in one of a signal's derivatives. The signal, while apparently a single smooth curve, is actually composed of two separate exponentials.

```
load scddvbrk;
x = scddvbrk;
level = 2;
[c,l] = wavedec(x,level,'db4');
[d1,d2] = detcoef(c,l,1:level);
d1up = dyadup(d1,0);
d2up = dyadup(dyadup(d2,0),0);
subplot(3,1,1)
plot(x)
xlim([400 600])
subplot(3,1,2)
plot(d1up)
ylabel('d1')
xlim([400 600])
```

```
subplot(3,1,3)
plot(d2up)
ylabel('d2')
xlim([400 600])
```
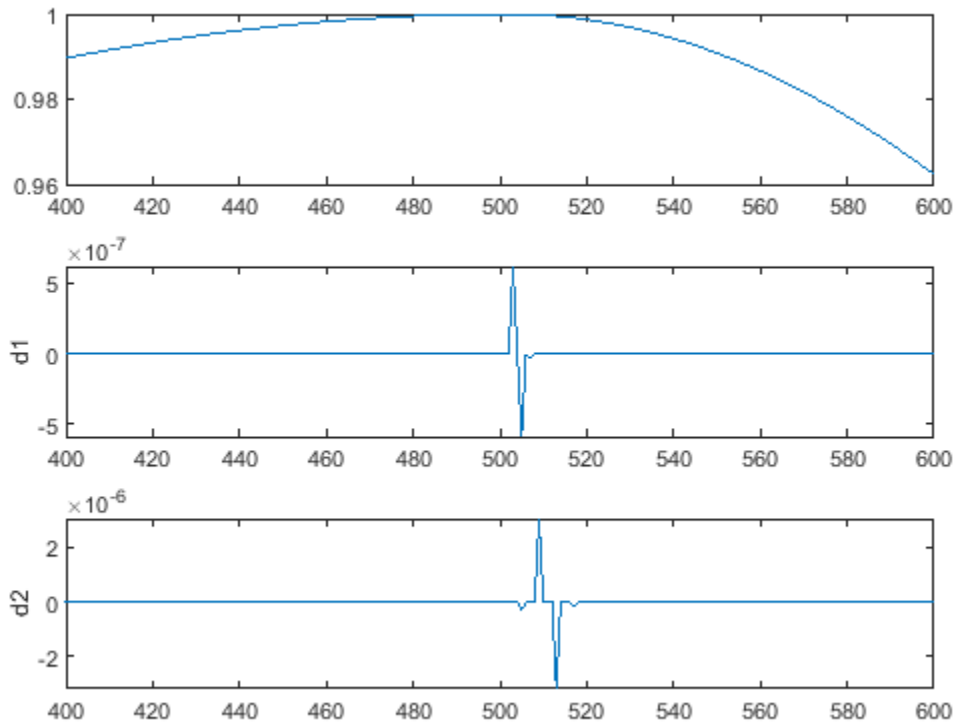

We have zoomed in on the middle part of the signal to show more clearly what happens around time = 500. The details are high only in the middle of the signal and are negligible elsewhere. This suggests the presence of high-frequency information — a sudden change or discontinuity — around time = 500.

Note that to detect a singularity, the selected wavelet must be sufficiently regular, which implies a longer filter impulse response. Regularity can be an important criterion in selecting a wavelet. We have chosen to use db4, which is sufficiently regular for this analysis. Had we chosen the Haar wavelet, the discontinuity would not have been detected. If you try repeating this analysis using the Haar wavelet at level two, you will notice that the details are equal to zero at time = 500.

Edge Detection

For images, a two-dimensional discrete wavelet transform 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).

```
load tartan;
level = 1;
[c,s] = wavedec2(X,level,'coif2');
[chd1,cvd1,cdd1] = detcoef2('all',c,s,level);
```

```
subplot(1,2,1)
image(X)
colormap(map)
title('Original Image')
subplot(1,2,2)
image(chd1)
colormap(map)
title('Horizontal Edges')
```
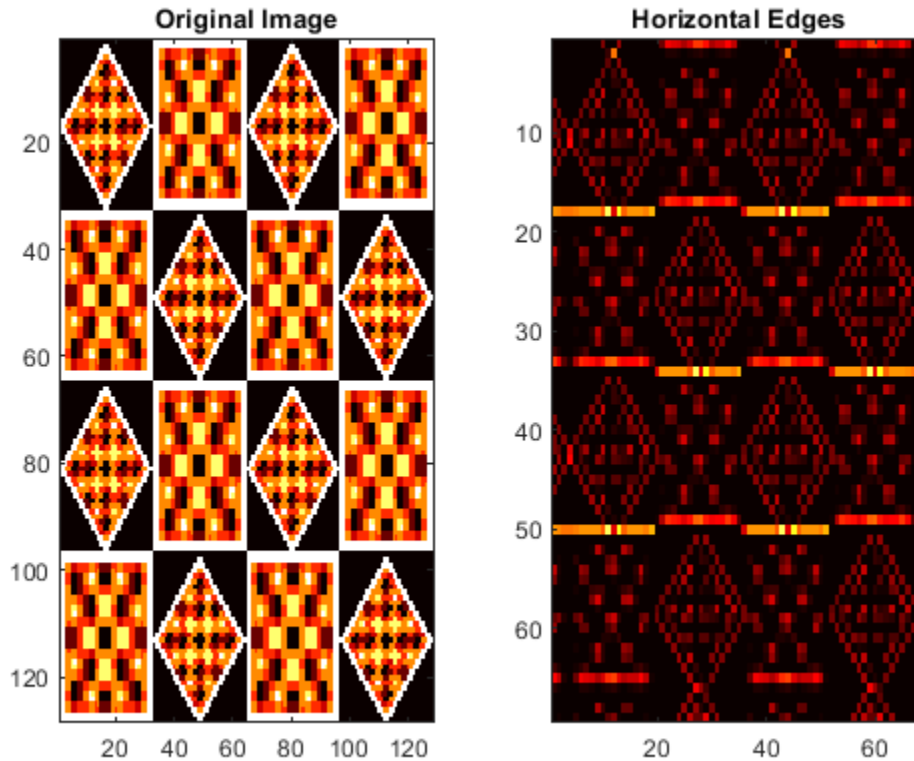


```
subplot(1,2,1)
image(cvd1)
colormap(map)
title('Vertical Edges')
subplot(1,2,2)
image(cdd1)
colormap(map)
title('Diagonal Edges')
```
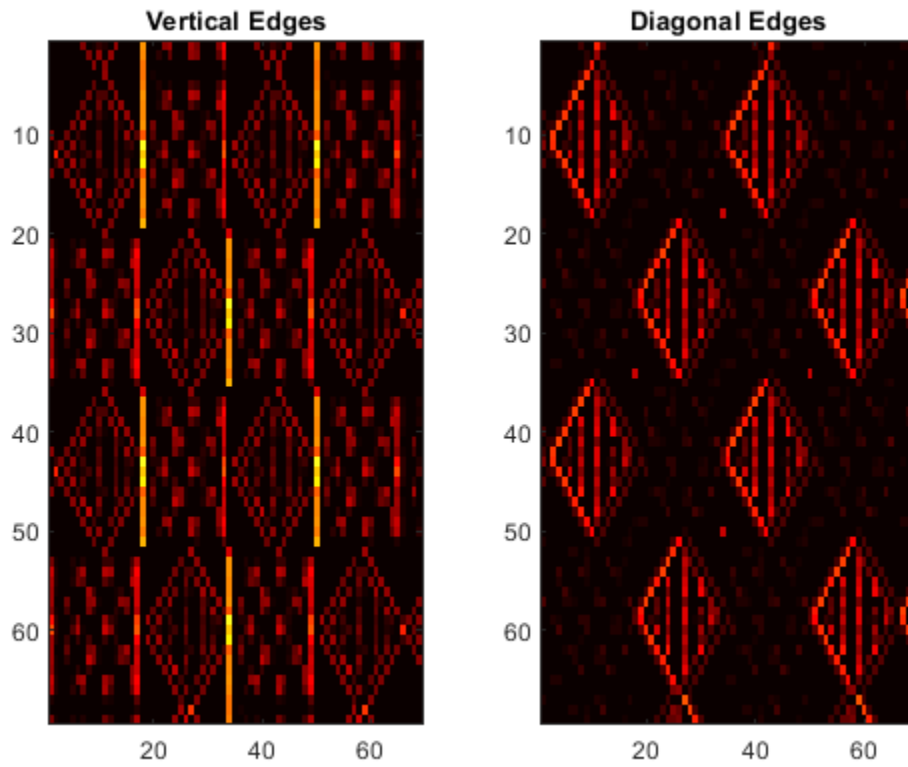
See Also
wavedec | wavedec2

Haar Transforms for Time Series Data and Images

This example shows how to use Haar transforms to analyze time series data and images. To run all the code in this example, you must have Signal Processing Toolbox™ and Image Processing Toolbox™.

First, visualize the Haar wavelet.

```
[~,psi,x] = wavefun('haar',10);
x = x(2:end-1);
psi = psi(2:end-1);
hl = plot(x(1:512),psi(1:512));
grid on
hold on
line(x(513:end),psi(513:end))
xlabel('t')
ylabel('\psi(t)','fontsize',14)
title('Haar Wavelet')
```
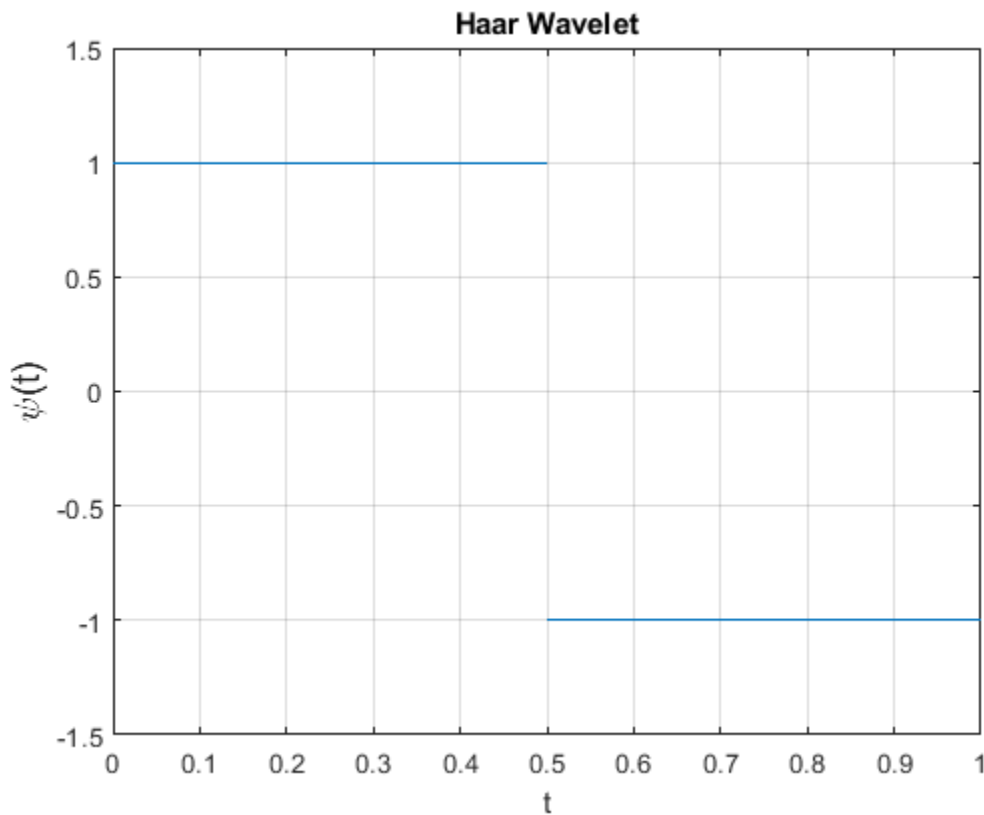

The Haar wavelet is discontinuous. As a result, it is typically not used in denoising or compression applications where smoothness of the reconstruction wavelet is an important consideration. However, Haar transforms are useful in a number of applications due to their superior time (spatial) localization and computational efficiency. The Wavelet Toolbox™ supports Haar analysis in most of the discrete wavelet analysis tools. This example features Haar lifting implementations which support integer-to-integer wavelet transforms for both 1-D and 2-D data and multichannel (multivariate) 1-D data.

Analyze Signal Variability By Scale

Load and plot the `clock_571` dataset. This example is essentially a recreation of the analysis presented in Percival & Walden [3], pp 13–16.

```
load clock_571;
figure;
plot(clock_571)
xlabel('Days')
grid on;
title('Daily Average Fractional Frequency Deviates -- Cesium Clock');
```
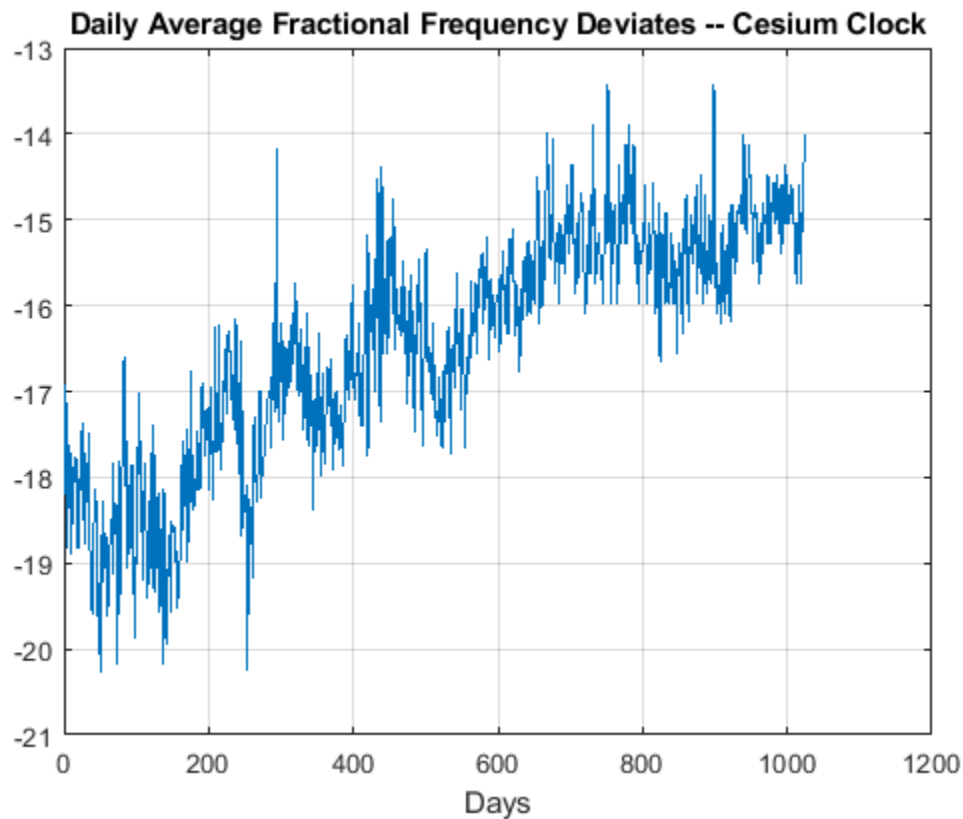

The data are daily average fractional frequency deviates for a particular cesium beam atomic clock with respect to the U.S. Naval Observatory master clock. If the value of the time series is 0, that means the cesium clock has neither lost nor gained time with respect to the master clock over the duration of the day. If the value is negative, the clock has lost time that day, a positive value means that the clock has gained time. For this data, the values are all negative. For certain applications, like geodesy, it is important to know whether there are certain time scales where the clock's deviation from the master clock is at its lowest value. In other words, are there certain scales where the clock agrees most closely with the master clock? The Haar transform is useful here because it possesses two important properties: It decorrelates data by scale and it partitions signal energy among scale.

To illustrate the decorrelating property, obtain the Haar transform down to level 6. Plot the autocorrelation sequence of the original data along with the autocorrelation of the wavelet coefficients by scale for scales of 2,4,8,16, and 32 days. Dashed lines on the plots delineate 95%

confidence intervals for white noise inputs. Values exceeding those lines are indicative of significant autocorrelation in the data.

```
[s,w] = haart(clock_571,6);
helperAutoCorr(clock_571,w);
```
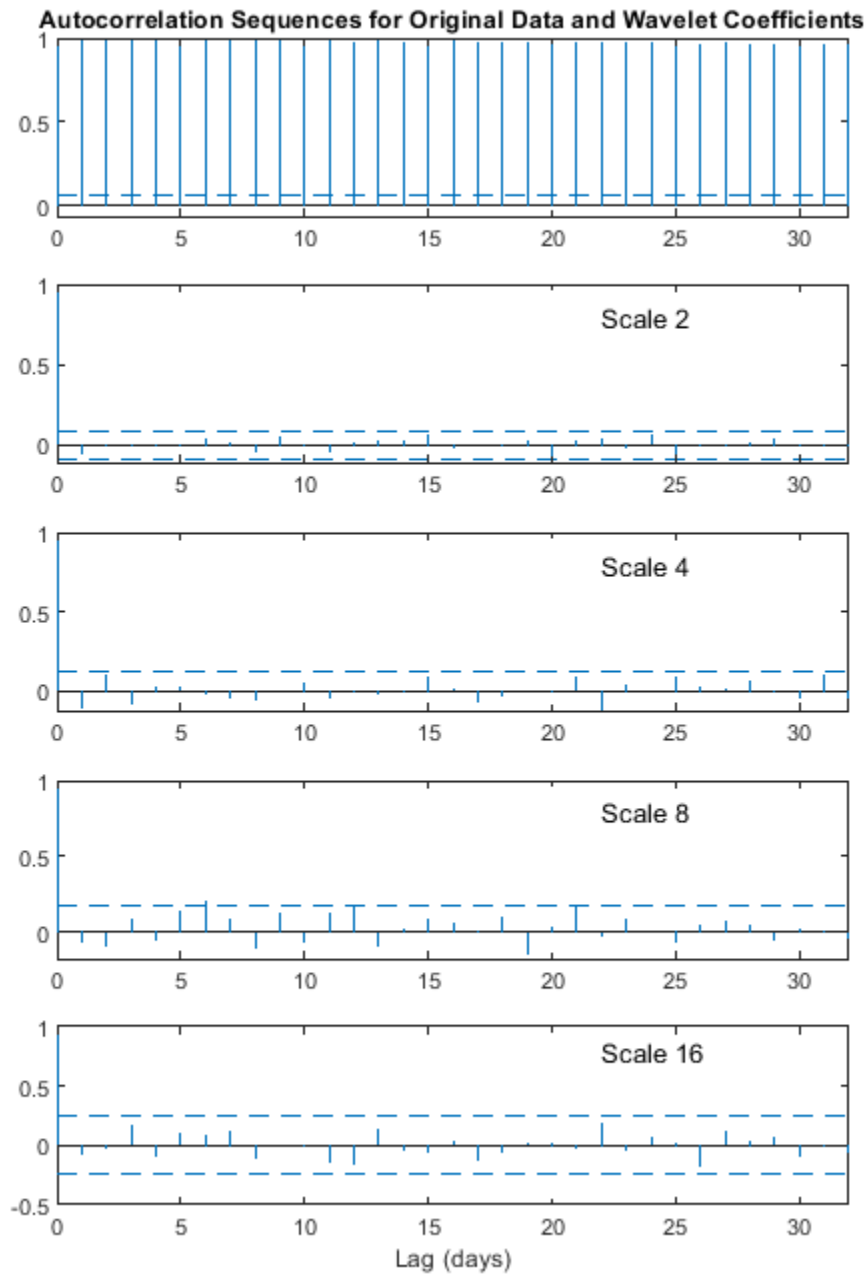
The top plot shows the autocorrelation sequence for the original data. Subsequent plots show the autocorrelation sequences for wavelet coefficients at increasingly coarser scales. It is clear that the autocorrelation sequence of the original data exhibits correlation at all lags while the Haar transform coefficients are decorrelated. Next, demonstrate energy conservation.

```
sigenergy = norm(clock_571,2)^2
energyByScale = cellfun(@(x)norm(x,2)^2,w);
haarenergy = norm(s,2)^2+sum(energyByScale)
```

```
sigenergy =

    2.7964e+05


haarenergy =

    2.7964e+05
```

The total signal energy is preserved by the Haar transform. Because of these properties, you can make meaningful inferences based on the proportion of signal energy captured by the wavelet coefficients at each scale.

```
scales = 2.^(1:6);
figure
plot(scales,energyByScale,'-o')
xlabel('Scale (days)')
set(gca,'xscale','log')
set(gca,'xtick',2.^(1:6))
ylabel('Proportion of Signal Energy')
grid on
```
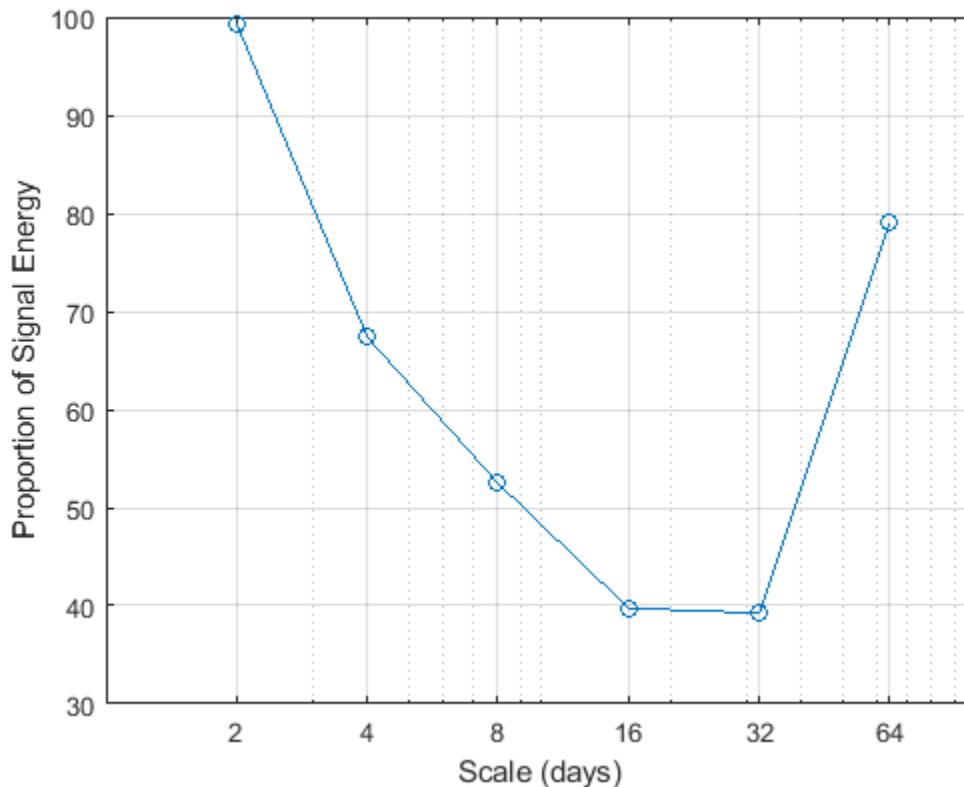
You see that the energy is at a minimum for scales of 16 and 32 days. For the Haar wavelet (and all Daubechies wavelets), the wavelet coefficients at a given scale represent differences between weighted averages of the data over a duration 1/2 the length of the scale. This plot indicates scales over which the cesium clock is in best agreement with the master clock. This means that considering data over approximately two-week or even one-month periods is more accurate than data on smaller or longer scales. As previously mentioned, this has important implications for geodesy where extremely precise time measurements are critical.

Although the Haar wavelet is discontinuous, it is still effective at representing various kinds of time series. Examples include count data and data where values of a time series are tied to some specific state, which affects the level of the time series. As an example, consider the relationship between heart rate and sleep state.

Create Signal Approximations

The data consist of two time series. One time series is the heart rate of a 66-day old infant sampled every 16 seconds for just over 9 hours. The heart rate time series is integer-valued. The other time series is the expertly scored sleep state of the same infant over the same period with the same sampling rate. The sleep state data was scored based on the infant's EEG and EOG (eye movement) data, not based on the heart rate. The sleep state codes are 1=quiet sleep, 2=between quiet and active sleep, 3=active sleep, and 4=awake. Both time series were recorded by Prof. Peter Fleming, Dr Andrew Sawczenko, and Jeanine Young of the Institute of Child Health, Royal Hospital for Sick Children, Bristol and kindly provided for use in this example. Plot the heart rate data along with the sleep states.

```
load BabyECGData
figure
yyaxis left
plot(times,HR)
ylabel('HR')
xlabel('Hrs')
YLim = [min(HR)-1 max(HR)+1];
yyaxis right
plot(times,SS)
ylabel('Sleep State')
YLim = [0.5 4.5];
title('Baby ECG and Sleep State')
```
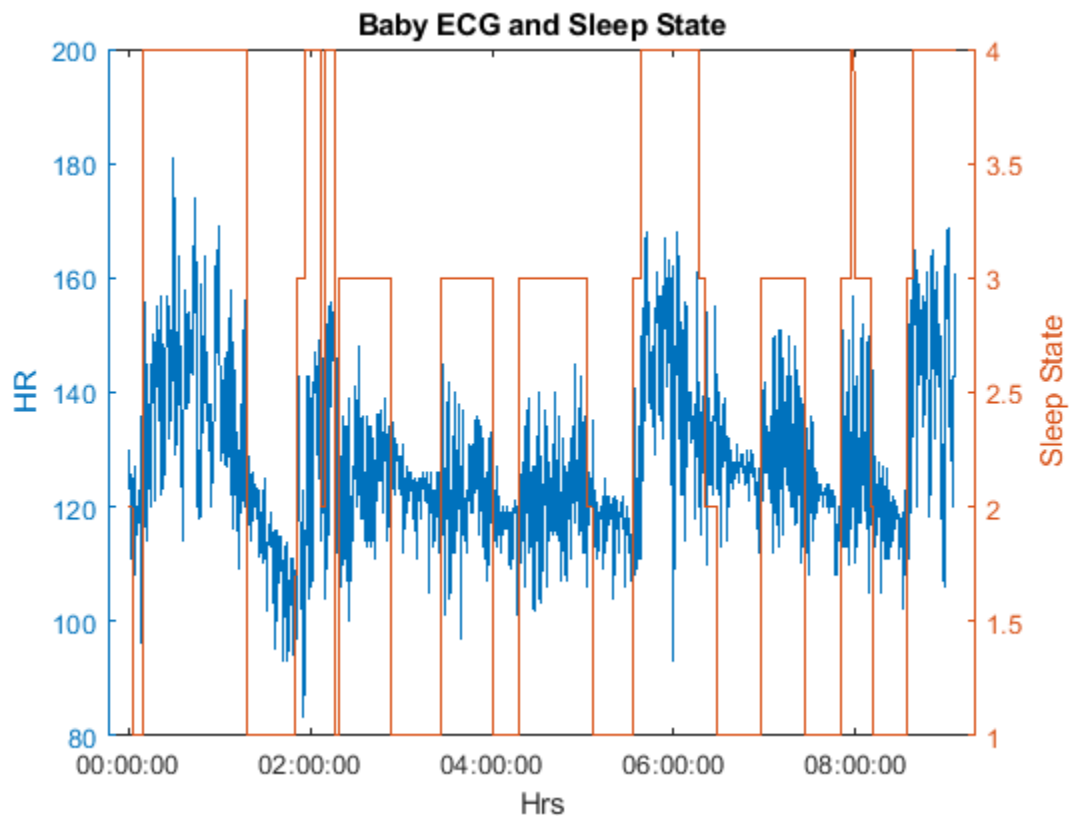

An inspection of the data reveals an apparent correlation between sleep state and heart rate, but the data is extremely noisy. Because the Haar transform provides a staircase approximation to a signal, it is often useful in situations where a response is dependent on a predictor variable with a small number of discrete states. Here the discrete states are the four sleep stages. Obtain the Haar approximation of the heart rate data using a level 5 approximation. Because the heart rate data is integer-valued, use the `'integer'` flag to ensure that integer-valued data is returned. Plot the result.

```
[S,W] = haart(HR,'integer');
HaarHR = ihaart(S,W,5,'integer');
figure
hL = plotyy(times,HaarHR,times,SS);
Ax1 = hL(1);
Ax2 = hL(2);
```

```
Ax1.YLim = [min(HaarHR)-1 max(HaarHR)+1];
Ax1.YLabel.String = 'HR';
Ax2.YLim = [0.5 4.5];
Ax2.YLabel.String = 'Sleep State';
xlabel('Hrs')
title('Haar Approximation and Sleep State')
```
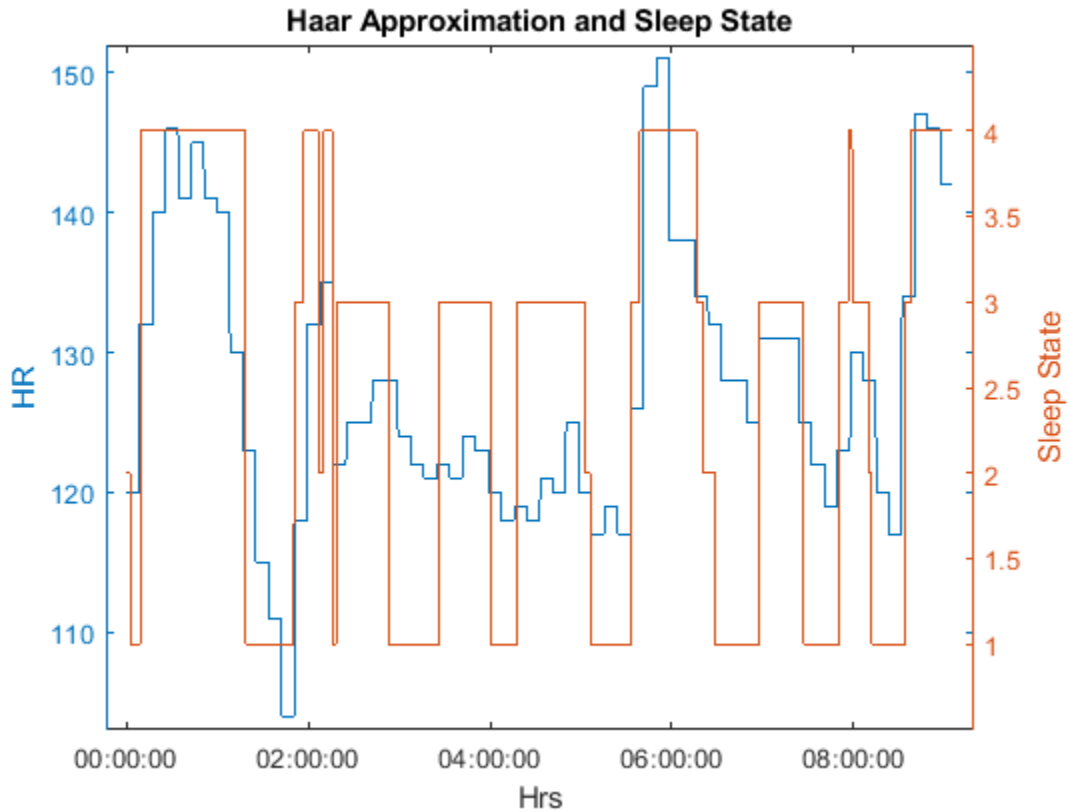

The Haar approximation more clearly shows the relationship between the sleep state and the heart rate data. You can assess this change by looking at the correlation between the raw data and the sleep state time series.

```
corr(SS,HR)
```

```
ans =

    0.5576
```

Now compare the value of 0.56 with the correlation between the sleep state data and the Haar approximation

```
corr(SS,HaarHR)
```

```
ans =
```

```
0.6907
```

The correlation has increased from 0.56 to 0.69. More advanced wavelet analysis and modeling of this data is presented in Nason, von Sachs, & Kroisandt [1] and Nason, Sapatinas, & Sawczenko [2].

Digital Watermarking of Images

Watermarking is an important data protection tool. It is a passive protection technique where a marker is covertly inserted in some data in order to verify the authenticity or integrity of the data. Wavelet techniques in general and the Haar transform in particular are frequently employed in watermarking images. This example illustrates the use of the Haar transform in watermarking an image and recovering the watermark. The example employs a simple watermarking scheme chosen for ease of illustration. In this scheme, the watermark is inserted in the approximation coefficients at level 3.

Watermark an image of a Mandril with one of a honey badger. Read in the Mandrill image. Resize it to 2048x2048 and display the result.

```
coverIM = imread('mandrill.jpg');
coverIM = rgb2gray(coverIM);
coverIM = imresize(im2double(coverIM),[2048 2048]);
imagesc(coverIM)
colormap gray
title('Original Image to Watermark')
axis off
axis square
```


Original Image to Watermark
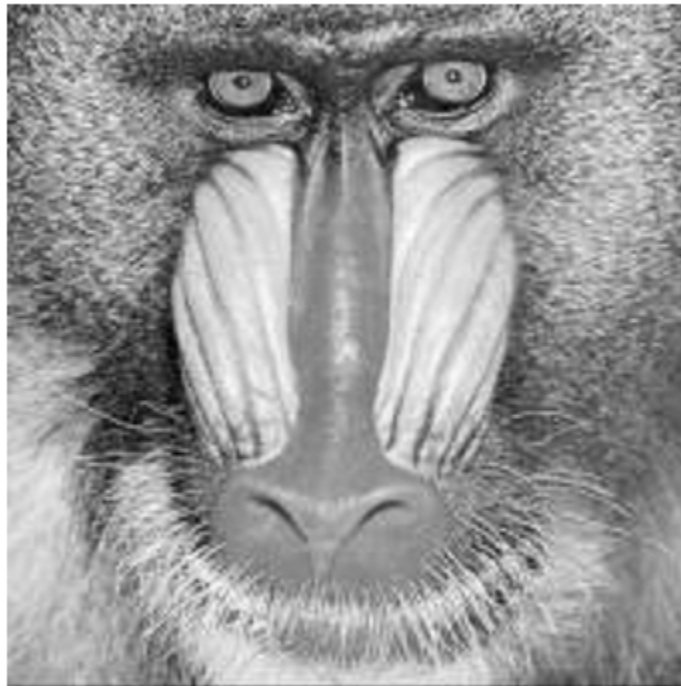
Obtain the Haar transform of the Mandrill image down to level 3.

```
[LLorig,LHorig,HLorig,HHorig] = haart2(coverIM,3);
imagesc(LLorig)
title('Level 3 Haar Approximation')
axis off
axis square
```

Level 3 Haar Approximation
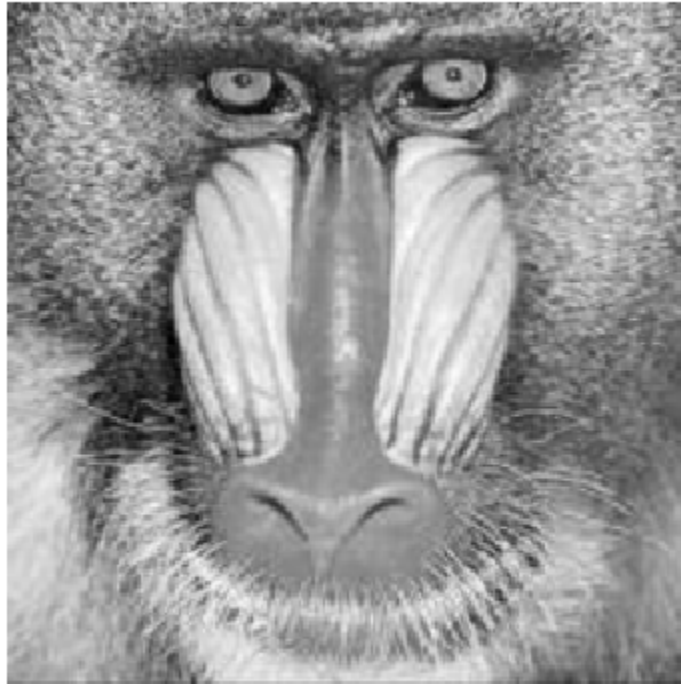
Read in watermark image and resize it.

```
watermark = imread('honeybadger.jpg');
watermark = im2double(rgb2gray(watermark));
watermark = imresize(watermark,[2048 2048]);
```

Obtain the Haar transform of the watermark image down to level 3.

```
[LLw,LHw,HLw,HHw] = haart2(watermark,3);
imagesc(LLw)
colormap gray
title('Level 3 Haar Approximation--Watermark')
axis off
axis square
```

Level 3 Haar Approximation--Watermark

Add the honey badger watermark to the Mandril image by attenuating the level-3 approximation coefficients of the watermark and inserting the attenuated coefficients into the level-3 Mandrill approximation coefficients.

```
LLwatermarked = LLorig+1e-4*LLw;
markedIM = ihaart2(LLwatermarked,LHorig,HLorig,HHorig);
imagesc(markedIM)
title('Watermarked Image')
axis off
axis square
colormap gray
```

Watermarked Image
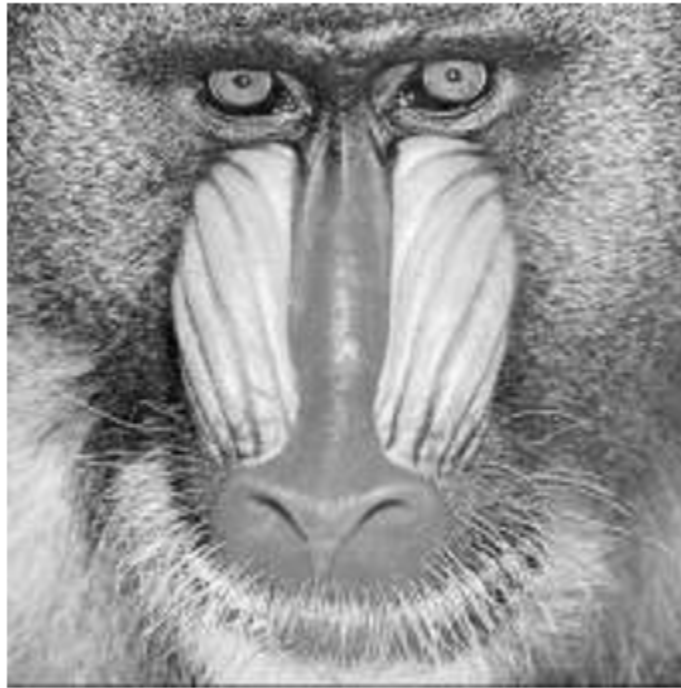
The watermark (honey badger) is not visible in the watermarked image. Because you know what algorithm was used to insert the watermark, you can recover the watermark using the Haar transform.

```
[LLr,LHr,HLr,HHr] = haart2(markedIM,3);
LLmarked = (LLr-LLorig).*1e4;
imagesc(LLmarked)
title('Recovered Watermark')
colormap gray
axis off
axis square
```

Recovered Watermark

References

[1] Nason, G. P., R. von Sachs, and G. Kroisandt. "Wavelet processes and adaptive estimation of the evolutionary wavelet spectrum." *J. R. Statist. Soc. Series B*. Volume 62, Issue 2, 2000, pp. 271–292.

[2] Nason, G. P., T. Sapatinas, and A. Sawczenko. "Wavelet packet modelling of infant sleep state using heart rate data." *Sankhyā: The Indian Journal of Statistics. Series B*. Volume 63, No. 2, 2001, pp. 199–217.

[3] Percival, D. B., and A. T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge, UK: Cambridge University Press, 2000.

See Also
haart | haart2

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')
```

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')
```
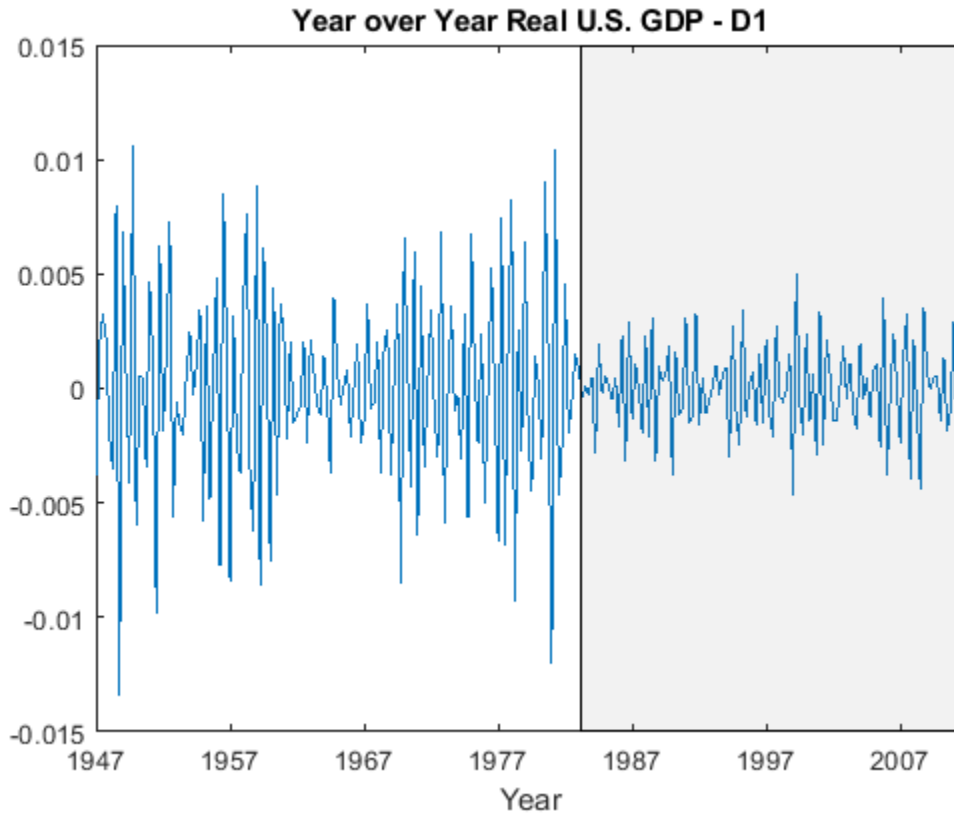
Examining the level-one details, it appears there is a reduction of variance in the period of the Great Moderation.

Test the level-one wavelet coefficients for significant variance changepoints.

```
[pts_Opt,kopt,t_est] = wvarchg(realgdpwt(1,1:numel(realgdp)),2);
years(pts_Opt)
```

```
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-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_Opt);
tspost = realgdp(pts_Opt+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 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] = modwtxcorr(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)');
```

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')
```

CPI Inflation -- 1961 to 2011
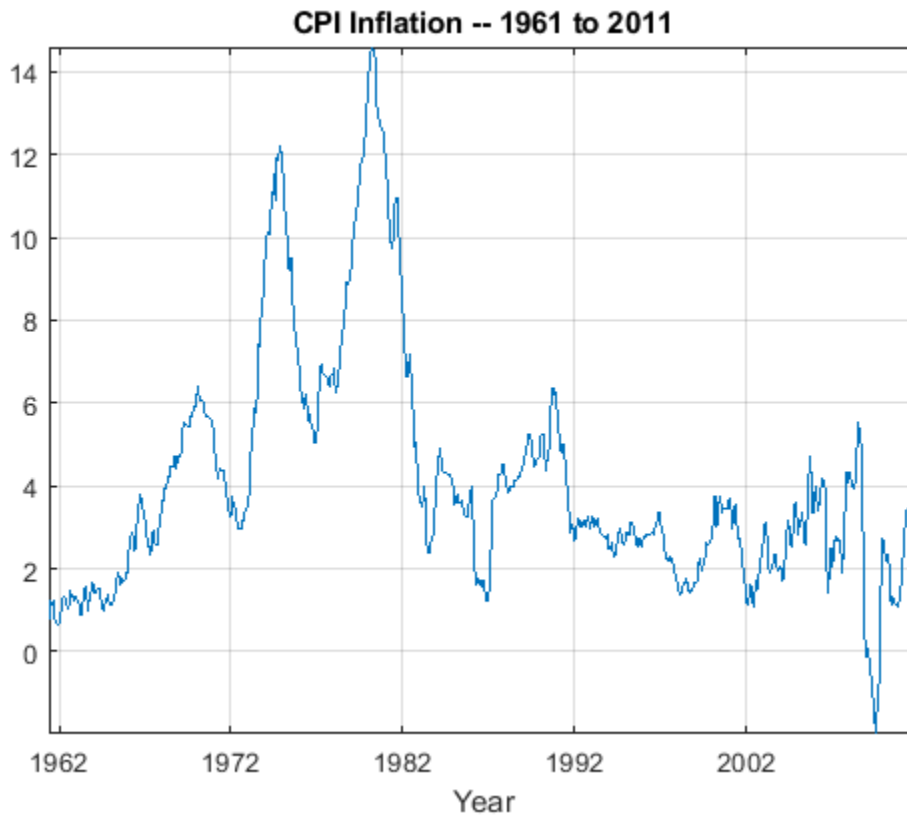
In the time data, a slow oscillation appears in the early 1970s and seems to dissipate by the late 1980s.

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));
```

Magnitude Scalogram
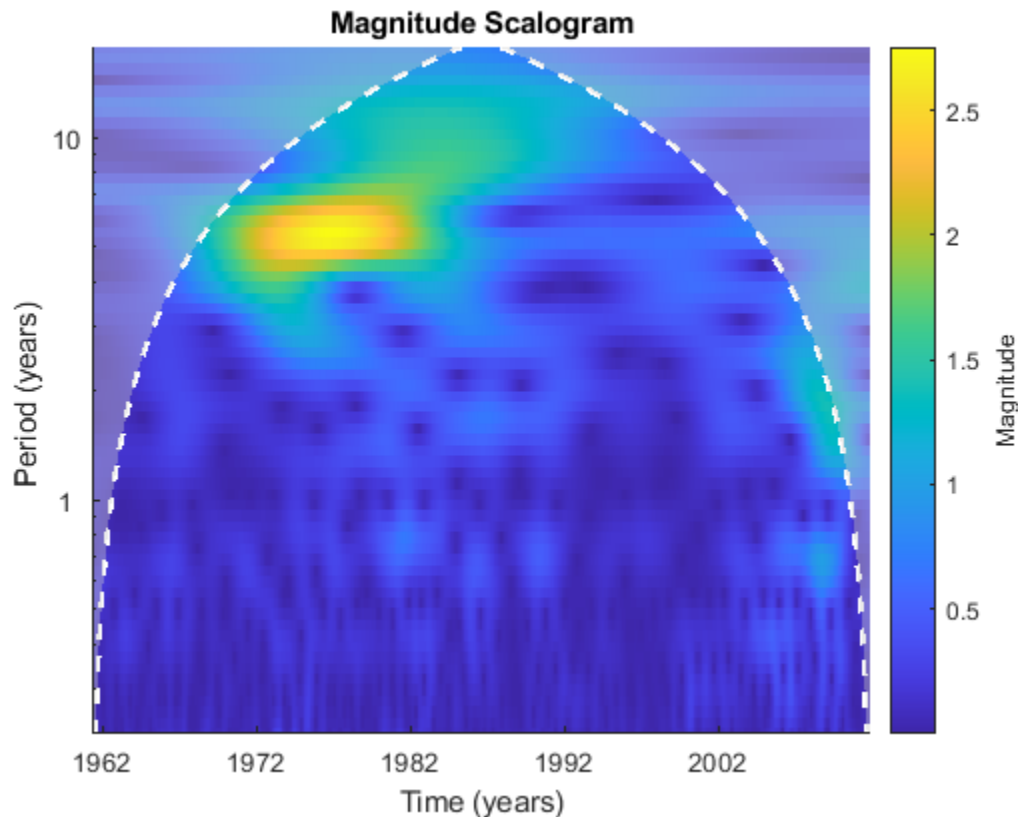
The CWT reveals the strongest oscillations in the inflation rate data in the approximate range of 4-6 years. This volatility begins to dissipate by the mid 1980s and is characterized by both a gradual reduction in inflation and a shift in volatility toward longer periods. The strong volatility cycles 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

References

[1] Aguiar-Conraria, L., M. M. F. Martins, and M. J. Soares. "The yield curve and the macro-economy across time and frequencies." *Journal of Economic Dynamics and Control*. Vol. 36, Number 12, 2012, pp. 1950–1970.

[2] Crowley, P. M. "A Guide to Wavelets for Economists." *Journal of Economic Surveys*. Vol. 21, Number 2, 2007, pp. 207–267.

[3] Gallegati, M., and W. Semmler (Eds.). *Wavelet Applications in Economics and Finance*. Heidelberg, Germany: Springer International Publishing, 2014.

[4] Percival, D. B., and A. T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge, UK: Cambridge University Press, 2000.

See Also
modwt | modwtcorr | modwtvar

Image Fusion

The principle of image fusion using wavelets is to merge the wavelet decompositions of the two original images using fusion methods applied to approximations coefficients and details coefficients. The two images must be of the same size and are supposed to be associated with indexed images on a common colormap (see wextend to resize images).

Two examples are examined: the first one merges two different images leading to a new image and the second restores an image from two fuzzy versions of an original image.

Fusion of Two Different Images

Load two original images: a mask and a bust.

```
load mask; X1 = X;
load bust; X2 = X;
```

Merge the two images from wavelet decompositions at level 1 using db2 by taking two different fusion methods: fusion by taking the mean for both approximations and details:

```
XFUSmean = wfusimg(X1,X2,'db2',1,'mean','mean');
```

and fusion by taking the maximum for approximations and the minimum for the details.

```
XFUSmaxmin = wfusimg(X1,X2,'db2',1,'max','min');
```

Plot original and synthesized images.

```
colormap(map);
subplot(221), image(X1), axis square, title('Mask')
subplot(222), image(X2), axis square, title('Bust')
subplot(223), image(XFUSmean), axis square,
title('Synthesized image, mean-mean')
subplot(224), image(XFUSmaxmin), axis square,
title('Synthesized image, max-min')
```

Mask

Bust

Synthesized image, mean-mean

Synthesized image, max-min
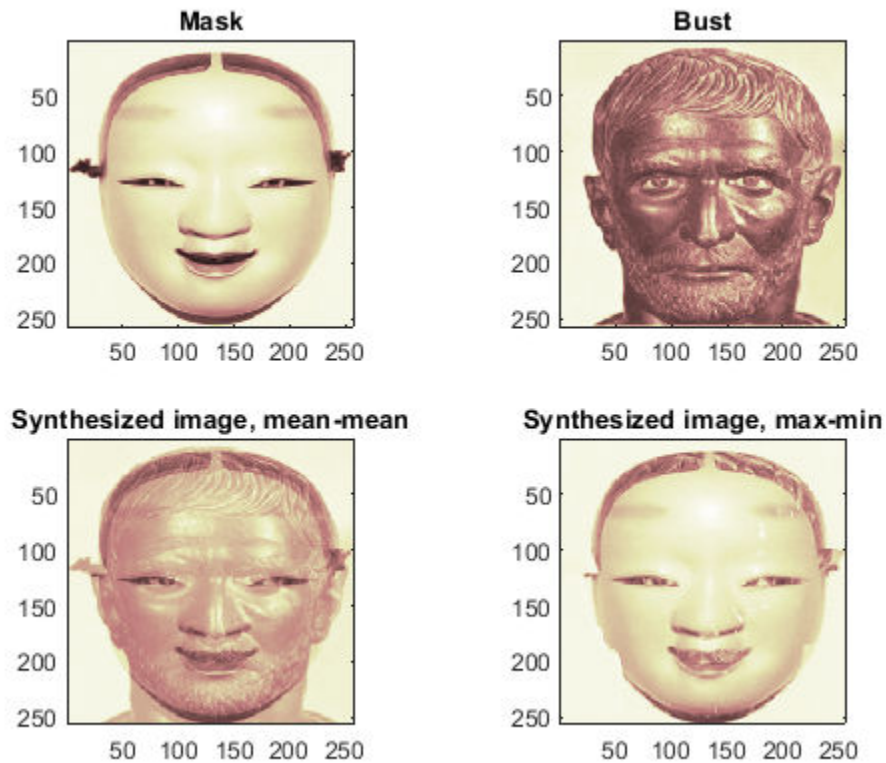
Restoration by Fusion from Fuzzy Images

Load two fuzzy versions of an original image.

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

Merge the two images from wavelet decompositions at level 5 using sym4 by taking the maximum of absolute value of the coefficients for both approximations and details.

```
XFUS = wfusimg(X1,X2,'sym4',5,'max','max');
```

Plot original and synthesized images.

```
figure('Color','white'),colormap(map);
subplot(221), image(X1), axis square,
title('Catherine 1')
subplot(222), image(X2), axis square,
title('Catherine 2')
subplot(223), image(XFUS), axis square,
title('Synthesized image')
```

Catherine 1

Catherine 2

Synthesized image
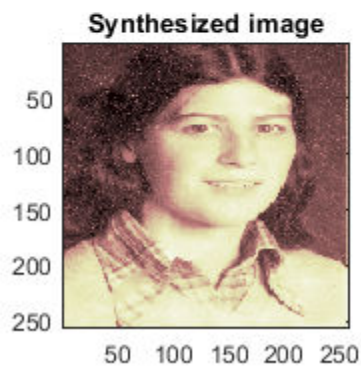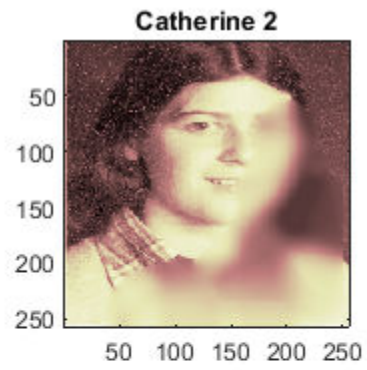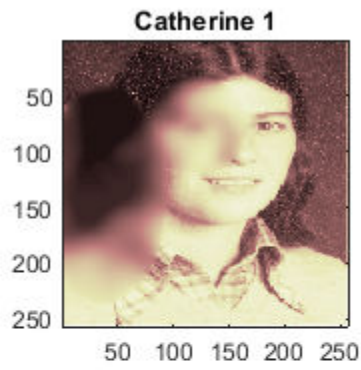
Boundary Effects in Real-Valued Bandlimited Shearlet Systems

This example shows how edge effects can result in shearlet coefficients with nonzero imaginary parts even in a real-valued shearlet system. The example also discusses two strategies for mitigating these boundary effects: resizing or padding the image and changing the number of scales.

The bandlimited cone-adapted finite shearlet transform implemented in the `shearletSystem` object is developed in [1] on page 11-0 and [2] on page 11-0. The problem of potential edge effects in this system is discussed at length in [1] on page 11-0.

Frequency-Ordering Convention in Shearlet Filters

Bandlimited cone-adapted shearlets are constructed in the frequency domain, see "Shearlet Systems" on page 3-179 for more detail. The shearlet filters in `shearletSystem` use a slight modification of the spatial intrinsic coordinate system described in "Image Coordinate Systems" (Image Processing Toolbox). If you are familiar with the MATLAB conventions for ordering spatial frequencies in the 2-D discrete Fourier transform, you can skip this section of the example.

In MATLAB, the 2-D Fourier transform orders frequencies in the X and Y direction starting from (X,Y) = (0,0) in the top left corner of the image. To see this, create an image consisting of only a DC or mean component.

```
imMean = 10*ones(5,5);
imMeanDFT = fft2(imMean)

imMeanDFT = 5×5

   250     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

Spatial frequencies increase down the row dimension (Y) and across the columns (X) until reaching the Nyquist frequency and then decrease again. In the above example, there are no oscillations in the image, so all the energy is concentrated in `imMeanDFT(1,1)`, which corresponds to (0,0) in spatial frequencies. Now construct an image consisting of a nonzero mean value and a sinusoid with a low negative frequency in both the X and Y directions.

```
x = 0:15;
y = 0:15;
[X,Y] = meshgrid(x,y);
Z = 5+4*exp(2*pi*1j*(12/16*X+12/16*Y));
Zdft = fft2(Z);
imagesc(abs(Zdft))
```
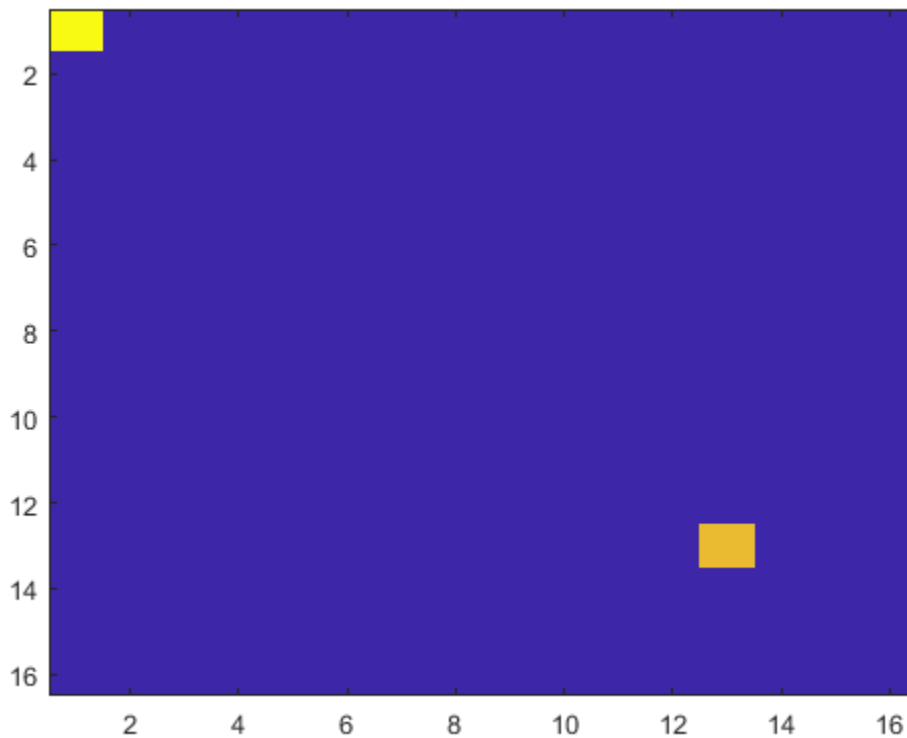
The mean component is in the top left corner of the 2-D DFT and the low negative frequency in the X and Y direction is in the lower right corner of the image as expected.

Now apply `fftshift`, which for matrices (images) swaps the first quadrant with the third, and the second quadrant with the fourth. This means that high negative spatial frequencies in X and Y are now in the top left corner. Spatial frequency decreases in the Y direction to zero and then increases again up to the Nyquist as you move down the image. Similarly, spatial frequencies in X decrease as you move along the image to the right until zero and then increase up to the Nyquist. The (0,0) spatial frequency component is moved to the center of the image.

```
imagesc(abs(fftshift(Zdft)))
```
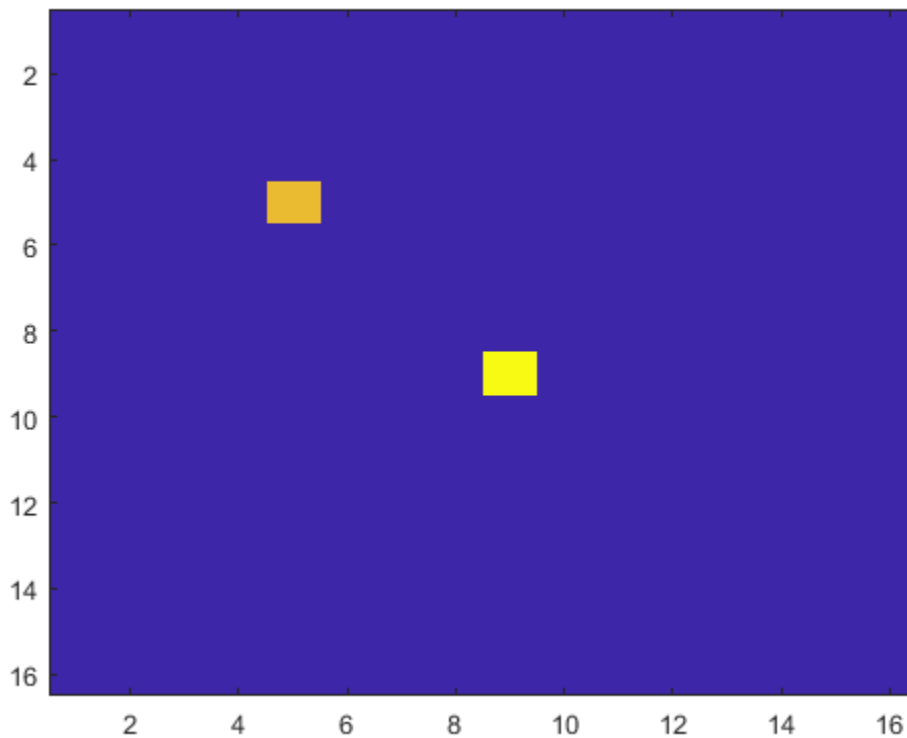
The frequency ordering of the shearlet filters returned by the `filterbank` function follows this shifted convention for the spatial frequencies in X, but the convention is flipped for spatial frequencies in Y. Spatial frequencies in Y are large and positive in the top left corner, decrease toward zero as you move down the rows, and increase again toward large negative spatial frequencies. Accordingly, the Y spatial frequencies are flipped with respect to MATLAB's `fftshift` output. This has no impact on the computation of the shearlet coefficients. This needs to be taken into account only when visualizing the shearlet filters in the 2-D frequency domain.

Real-Valued Band-Limited Shearlets and Symmetry in Frequency

Obtain a real-valued shearlet system with the default image size of 128-by-128. Visualize the three horizontal cone-adapted shearlets for shearing factors of -1,0,1, respectively, at the coarsest spatial scale (finest spatial frequency resolution).

```
sls = shearletSystem;
[psi,scale,shear,cone] = filterbank(sls);
omegax = -1/2:1/128:1/2-1/128;
omegay = -1/2:1/128:1/2-1/128;
figure
subplot(2,2,1)
surf(omegax,flip(omegay),psi(:,:,2),'EdgeColor',"none")
view(0,90)
title('Horizontal Cone — Shear = -1')
xlabel('$\omega_x$','Interpreter',"latex",'FontSize',14)
ylabel('$\omega_y$','Interpreter',"latex","FontSize",14)
subplot(2,2,2)
surf(omegax,flip(omegay),psi(:,:,3),'EdgeColor',"none")
```

```
xlabel('$\omega_x$',"Interpreter","latex",'FontSize',14)
ylabel('$\omega_y$',"Interpreter","latex","FontSize",14)
view(0,90)
title('Horizontal Cone — Shear = 0')
subplot(2,2,3)
surf(omegax,flip(omegay),psi(:,:,4),'EdgeColor',"none")
xlabel('$\omega_x$',"Interpreter","latex","FontSize",14)
ylabel('$\omega_y$',"Interpreter","latex","FontSize",14)
view(0,90)
title('Horizontal Cone — Shear = 1')
```
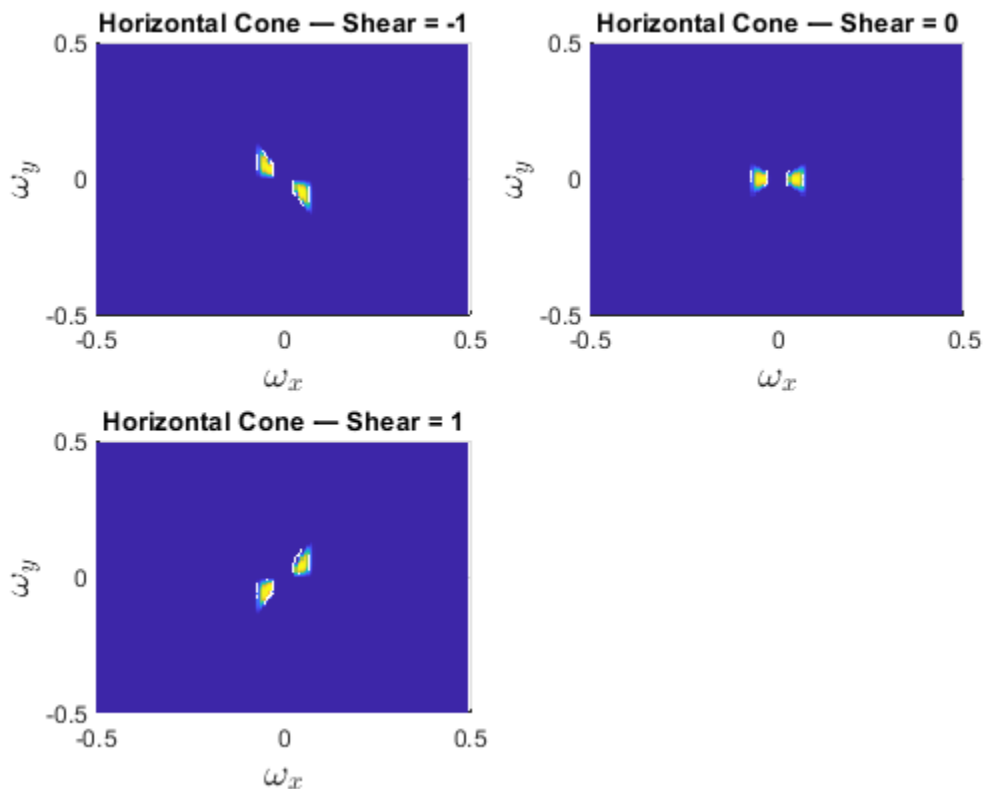

As expected and required for the shearlets to be real-valued in the spatial domain, their real-valued Fourier transforms must be symmetric with respect to the positive and negative X and Y spatial frequencies. Repeat the steps for the vertical cone-adapted shearlets at the finest spatial frequency scale.

```
figure
subplot(2,2,1)
surf(omegax,flip(omegay),psi(:,:,5),'EdgeColor',"none")
view(0,90)
title('Vertical Cone — Shear = -1')
xlabel('$\omega_x$','Interpreter',"latex",'FontSize',14)
ylabel('$\omega_y$','Interpreter',"latex","FontSize",14)
subplot(2,2,2)
surf(omegax,flip(omegay),psi(:,:,6),'EdgeColor',"none")
xlabel('$\omega_x$',"Interpreter","latex",'FontSize',14)
ylabel('$\omega_y$',"Interpreter","latex","FontSize",14)
```

```
view(0,90)
title('Vertical Cone — Shear = 0')
subplot(2,2,3)
surf(omegax,flip(omegay),psi(:,:,7),'EdgeColor',"none")
xlabel('$\omega_x$',"Interpreter","latex","FontSize",14)
ylabel('$\omega_y$',"Interpreter","latex","FontSize",14)
view(0,90)
title('Vertical Cone — Shear = 1')
```
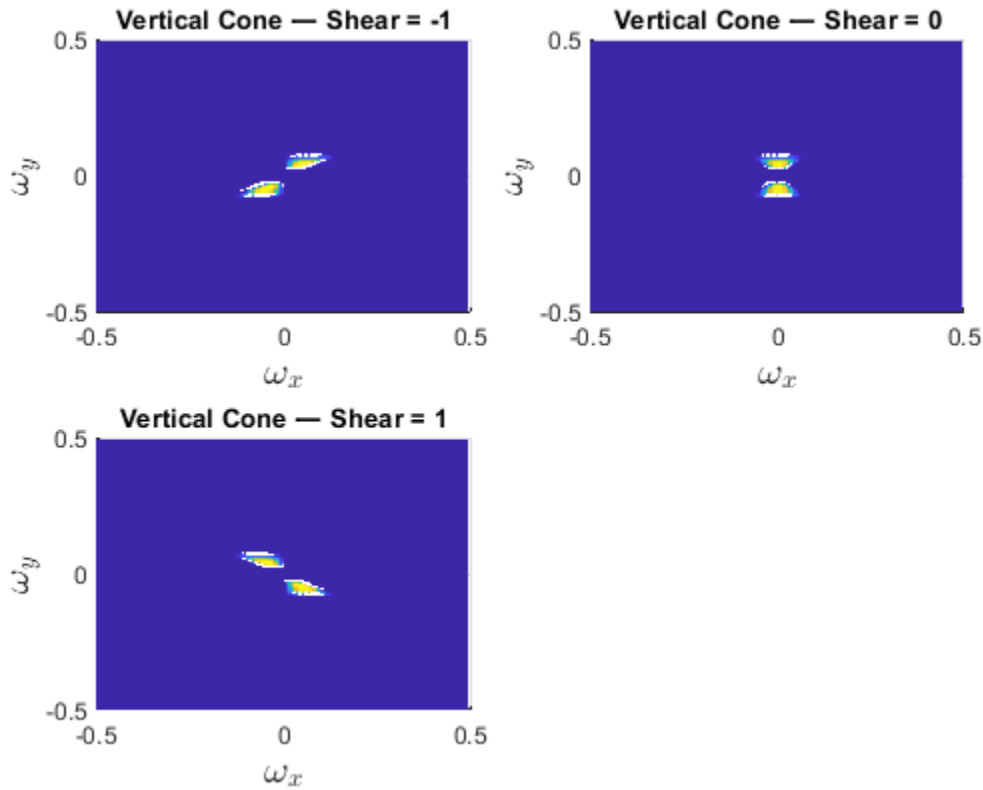

In a square image, it is easier to ensure the required symmetry for all shearing factors and dilations than in an image where the aspect ratio differs from one. The more pronounced the difference between the height and width dimensions of an image, the harder it becomes to ensure the necessary symmetry. Ensuring perfect symmetry in the 2-D frequency plane is also harder when the shearlet overlaps the edges of the image.

In shearletSystem, the number of scales determines the support of the lowpass filter. The greater the number of the scales the more concentrated the frequency support of the lowpass filter around (0,0). For a real-valued shearlet system, the six scale-0 shearlets (3 horizontal and 3 vertical shearlets) fill in the frequency support around the lowpass filter. Accordingly, while the number of shears is constant at a given scale, the actual frequency support of the shearlets is not. To see this, construct another shearlet system with only one scale and plot the vertical-cone scale-0 shearlets.

```
sls1 = shearletSystem('NumScales',1);
psi1 = filterbank(sls1);
figure
subplot(2,2,1)
surf(omegax,flip(omegay),psi1(:,:,5),'EdgeColor',"none");
```

```
view(0,90)
title('Vertical Cone — Shear = -1')
xlabel('$\omega_x$','Interpreter',"latex",'FontSize',14)
ylabel('$\omega_y$','Interpreter',"latex","FontSize",14)
subplot(2,2,2)
surf(omegax,flip(omegay),psi1(:,:,6),'EdgeColor',"none");
xlabel('$\omega_x$',"Interpreter","latex",'FontSize',14)
ylabel('$\omega_y$',"Interpreter","latex",'FontSize',14)
view(0,90)
title('Vertical Cone — Shear = 0')
subplot(2,2,3)
surf(omegax,flip(omegay),psi1(:,:,7),'EdgeColor',"none");
xlabel('$\omega_x$',"Interpreter","latex","FontSize",14)
ylabel('$\omega_y$',"Interpreter","latex","FontSize",14)
view(0,90)
title('Vertical Cone — Shear = 1')
```
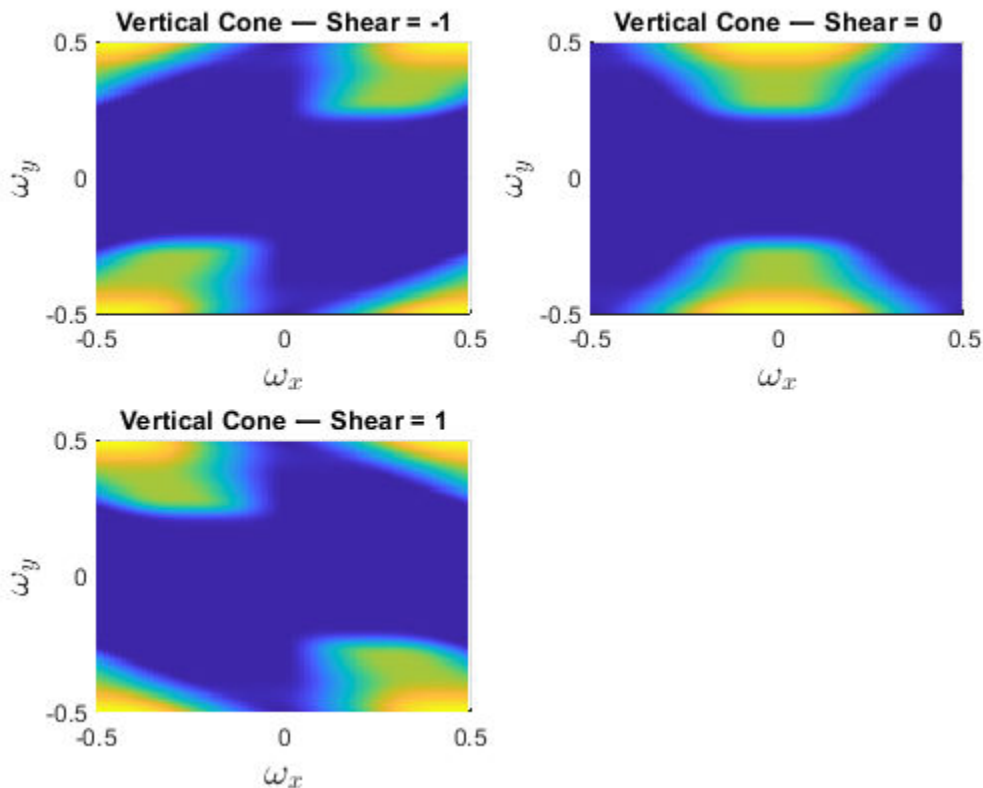

In this shearlet system, the scale-0 shearlets have all their energy in the highest X and Y spatial frequencies. At these edges of the 2-D frequency plane, the symmetry required for purely real-valued shearlet coefficients can suffer numerical instabilities. The important point is that the scale at which this can happen depends on the number of scales in the shearlet system as well as the aspect ratio of the analyzed image.

Plot the frequency responses of the lowpass filter and six shearlet filters to show how the 2-D frequency plane is filled ensuring a perfect reconstruction system.

```
figure
for ns = 1:size(psi1,3)
    surf(omegax,flip(omegay),psi1(:,:,ns),'edgecolor','none');
    xlabel('$\omega_x$','Interpreter',"latex","FontSize",14)
    ylabel('$\omega_y$','Interpreter',"latex",'FontSize',14)
    view(0,90);
    hold on;
end
title('One-Scale Shearlet Transform Filters')
```
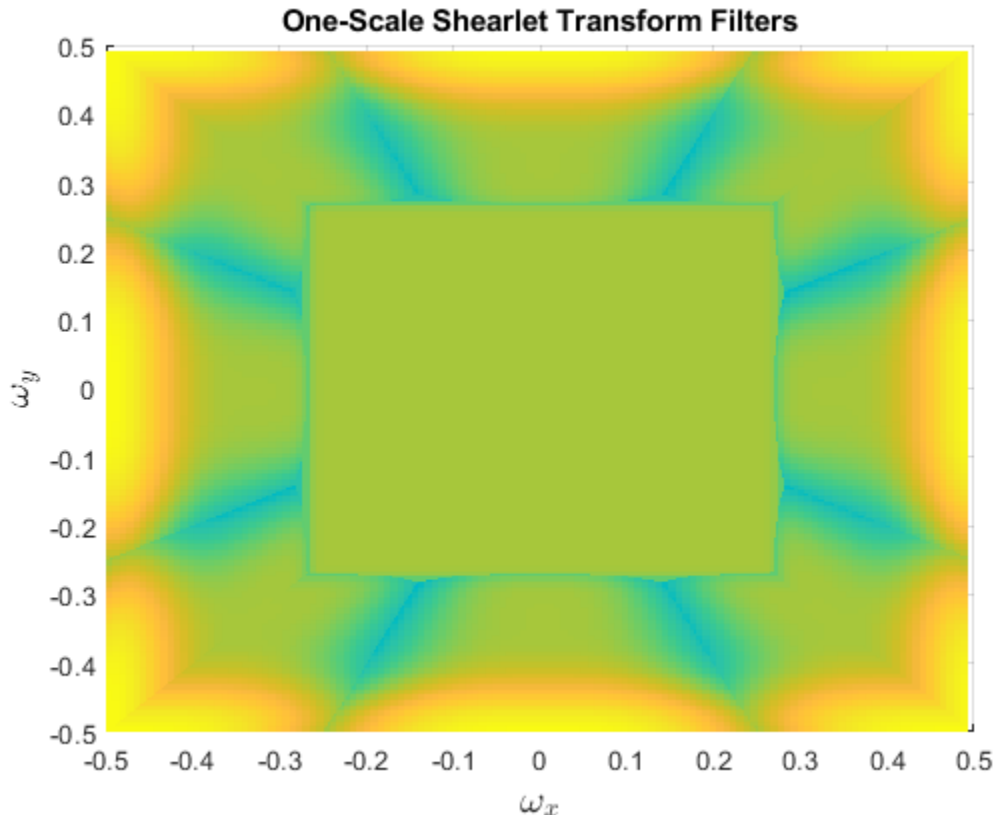

With only one scale in the shearlet system, the six shearlets must fill any region of the 2-D frequency space not covered by the lowpass filter. Accordingly, a breakdown in the symmetry required for purely real-valued shearlet coefficients may occur when there is only one scale present because those shearlets have support overlapping the edges. The next section discusses two ways to guard against the appearance of complex-valued coefficients in a real-valued shearlet transform.

Resize or Pad Rectangular Images

Whenever possible use square images. It is easier to ensure perfect symmetry in the shearlet frequency response when the aspect ratio is one. This is true even when the shearlet filters overlap the edge of the image. If you have Image Processing Toolbox™, you can use `imresize.` Read in the following image and display the image and its size.

```
im = imread('pout.tif');
size(im)
```

ans = *1×2*

```
    291   240
```

```
figure
imshow(im)
```


The image is 291-by-240. The default number of scales in the shearlet transform for this image size is 4. Obtain the real-valued shearlet transform for the default number of scales. Check whether any of the coefficients have a nonzero imaginary part.

```
sls = shearletSystem('ImageSize',size(im));
cfs = sheart2(sls,double(im));
max(imag(cfs(:)))
```

```
ans = 0
```

```
~any(imag(cfs(:)))
```

```
ans = logical
   1
```

The shearlet coefficients are purely real-valued as expected. Repeat the steps with the maximum number of scales equal to 2.

```
sls = shearletSystem('ImageSize',size(im),'NumScales',2);
cfs = sheart2(sls,double(im));
max(imag(cfs(:)))
```

```
ans = 0.0021
```

```
~any(imag(cfs(:)))
```

```
ans = logical
   0
```

Now the shearlet coefficients are not purely real. The perfect reconstruction property of the transform is not affected.

```
imrec = isheart2(sls,cfs);
max(abs(imrec(:)-double(im(:))))
```

```
ans = 2.7001e-13
```

However, the presence of complex-valued shearlet coefficients may be undesirable.

One strategy to mitigate this if you want a transform with two scales is to resize the image to 291-by-291.

```
imr = imresize(im,[291 291]);
imshow(imr)
```



```
sls = shearletSystem('ImageSize',size(imr),'NumScales',2);
cfs = sheart2(sls,double(imr));
max(imag(cfs(:)))
```

```
ans = 0
```

```
~any(imag(cfs(:)))
```

```
ans = logical
   1
```

By simply resizing the image to have an aspect ratio equal to one, you are able to obtain a purely real shearlet transform with the desired number of scales.

As an alternative to resizing, you can simply pad the image to the desired size. Use the `wextend` function to zero-pad the image columns to 291. Obtain the shearlet transform of the zero-padded image and verify that the coefficients are purely real. If you are interested only in the analysis coefficients, you can discard any padding after obtaining the shearlet transform. This is because the shearlet coefficient images have the same spatial resolution as the original image.

```
impad = wextend('ac','zpd',im,51,'r');
cfs = sheart2(sls,double(impad));
max(imag(cfs(:)))
```

```
ans = 0
```

```
~any(imag(cfs(:)))
```

```
ans = logical
   1
```

Adjust Number of Scales

There are cases where simply increasing or decreasing the number of scales by one is enough to eliminate any edge effects in the symmetry of the shearlet transform. For example, with the original image, simply incrementing the number of scales by one results in purely real-valued coefficients.

```
sls = shearletSystem('ImageSize',size(im),'NumScales',3);
cfs = sheart2(sls,double(im));
max(imag(cfs(:)))
```

```
ans = 0
```

```
~any(imag(cfs(:)))
```

```
ans = logical
   1
```

References

[1] Häuser, Sören, and Gabriele Steidl. "Fast Finite Shearlet Transform: A Tutorial." arXiv preprint arXiv:1202.1773 (2014).

[2] Voigtlaender, Felix, and Anne Pein. "Analysis vs. Synthesis Sparsity for α-Shearlets." arXiv preprint arXiv:1702.03559 (2017).

See Also
`isheart2` | `shearletSystem` | `sheart2`

More About

- "Shearlet Systems" on page 3-179

Featured Examples — Denoising and Compression

Denoising Signals and Images

This example discusses the problem of signal recovery from noisy data. The general denoising procedure involves three steps. The basic version of the procedure follows the steps described below:

- Decompose: Choose a wavelet, choose a level N. Compute the wavelet decomposition of the signal at level N.
- Threshold detail coefficients: For each level from 1 to N, select a threshold and apply soft thresholding to the detail coefficients.
- Reconstruct: Compute wavelet reconstruction using the original approximation coefficients of level N and the modified detail coefficients of levels from 1 to N.

Two points must be addressed in particular:

- how to choose the threshold,
- and how to perform the thresholding.

Soft or Hard Thresholding?

Thresholding can be done using the function `wthresh` which returns soft or hard thresholding of the input signal. Hard thresholding is the simplest method but soft thresholding has nice mathematical properties. Let `thr` denote the threshold.

```
thr = 0.4;
```

Hard thresholding can be described as the usual process of setting to zero the elements whose absolute values are lower than the threshold. The hard threshold signal is x if x > `thr`, and is 0 if x ≤ `thr`.

```
y = linspace(-1,1,100);
ythard = wthresh(y,'h',thr);
```

Soft thresholding is an extension of hard thresholding, first setting to zero the elements whose absolute values are lower than the threshold, and then shrinking the nonzero coefficients towards 0. The soft threshold signal is sign(x)(x-`thr`) if x > `thr` and is 0 if x ≤ `thr`.

```
ytsoft = wthresh(y,'s',thr);
subplot(1,3,1)
plot(y)
title('Original')
subplot(1,3,2)
plot(ythard)
title('Hard Thresholding')
subplot(1,3,3)
plot(ytsoft)
title('Soft Thresholding')
```
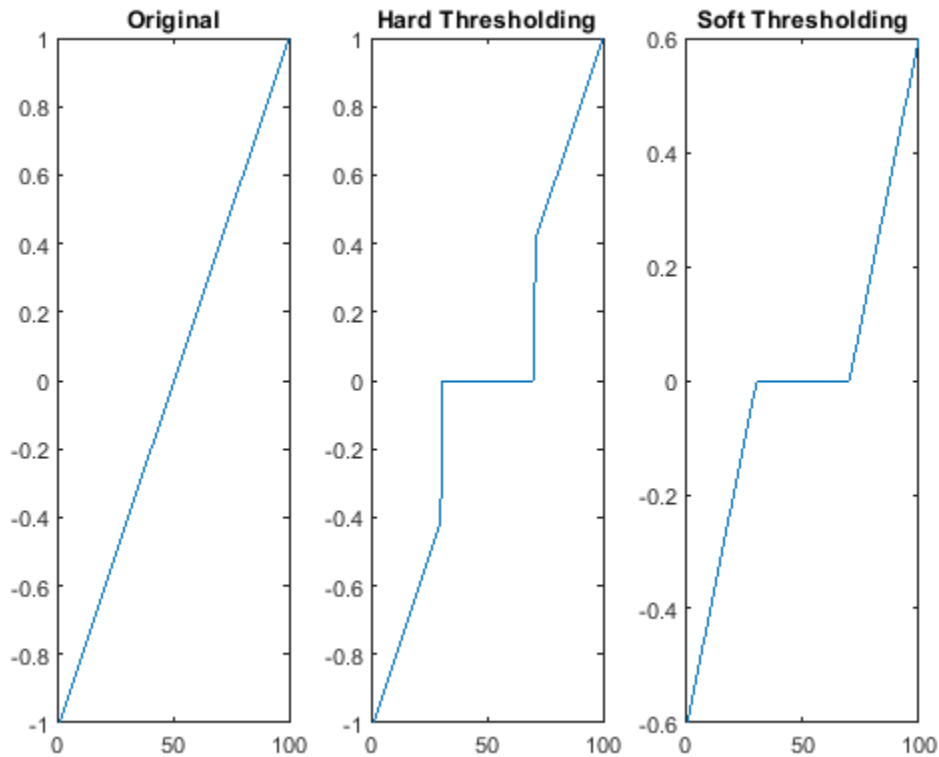
As can be seen in the figure above, the hard procedure creates discontinuities at x = ± t, while the soft procedure does not.

Threshold Selection Rules

Recalling step 2 of the denoise procedure, the function `thselect` performs a threshold selection, and then each level is thresholded. This second step can be done using `wthcoeff`, directly handling the wavelet decomposition structure of the original signal. Four threshold selection rules are implemented in the function `thselect`. Typically it is interesting to show them in action when the input signal is a Gaussian white noise.

```
rng default
y = randn(1,1000);
```

Rule 1: Selection using principle of Stein's Unbiased Risk Estimate (SURE)

```
thr = thselect(y,'rigrsure')
```

```
thr = 2.0518
```

Rule 2: Fixed form threshold equal to sqrt(2*log(length(y)))

```
thr = thselect(y,'sqtwolog')
```

```
thr = 3.7169
```

Rule 3: Selection using a mixture of the first two options

```
thr = thselect(y,'heursure')
```

```
thr = 3.7169
```

Rule 4: Selection using minimax principle

```
thr = thselect(y,'minimaxi')
```

```
thr = 2.2163
```

Minimax and SURE threshold selection rules are more conservative and would be more convenient when small details of the signal lie near the noise range. The two other rules remove the noise more efficiently.

Let us use the "blocks" test data credited to Donoho and Johnstone as the first example. Generate original signal xref and a noisy version x adding a standard Gaussian white noise.

```
sqrt_snr = 4;        % Set signal to noise ratio
init = 2055615866; % Set rand seed
[xref,x] = wnoise(1,11,sqrt_snr,init);
```

First denoise the signal using `wdenoise` with default settings. Compare the result with the original and noisy signals.

```
xd = wdenoise(x);
subplot(3,1,1)
plot(xref)
axis tight
title('Original Signal')
subplot(3,1,2)
plot(x)
axis tight
title('Noisy Signal')
subplot(3,1,3)
plot(xd)
axis tight
title('Denoised Signal - Signal to Noise Ratio = 4')
```
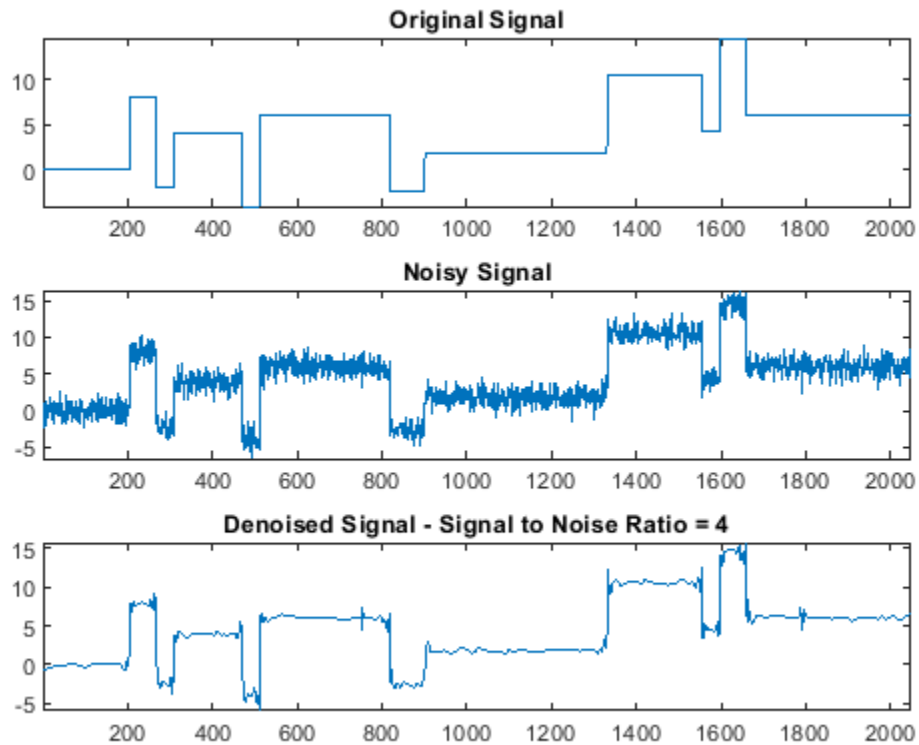
Denoise the noisy signal a second time, this time using soft heuristic SURE thresholding on detail coefficients obtained from the decomposition of x, at level 3 by `sym8` wavelet. Compare with the previous denoised signal.

```
xd2 = wdenoise(x,3,'Wavelet','sym8',...
    'DenoisingMethod','SURE',...
    'ThresholdRule','Soft');
figure
subplot(3,1,1)
plot(xref)
axis tight
title('Original Signal')
subplot(3,1,2)
plot(xd)
axis tight
title('First Denoised Signal: Default Settings')
subplot(3,1,3)
plot(xd2)
axis tight
title('Second Denoised Signal')
```

Original Signal

First Denoised Signal: Default Settings

Second Denoised Signal
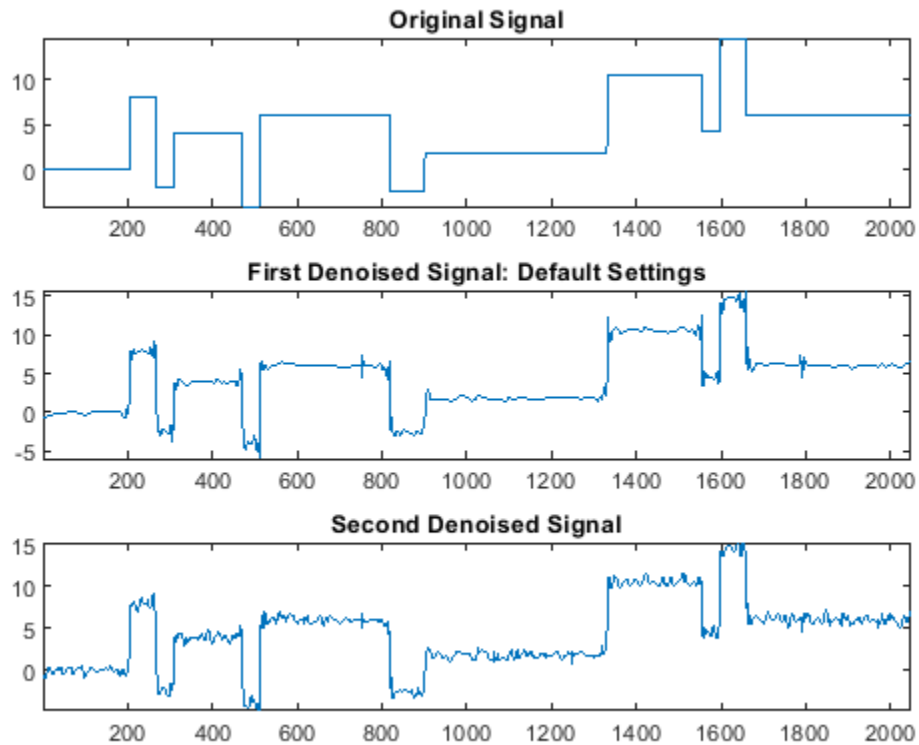
Since only a small number of large coefficients characterize the original signal, both denoised signals compare well with the original signal. You can use the Wavelet Signal Denoiser to explore the effects other denoising parameters have on the noisy signal.

Dealing with Non-White Noise

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. Let us use db3 wavelet and decompose at level 3. To deal with the composite noise nature, let us try a level-dependent noise size estimation.

```
load leleccum
indx = 2000:3450;
x = leleccum(indx); % Load electrical signal and select part of it.
```

Denoise the signal using soft fixed form thresholding and level-dependent noise size estimation.

```
xd = wdenoise(x,3,'Wavelet','db3',...
    'DenoisingMethod','UniversalThreshold',...
    'ThresholdRule','Soft',...
    'NoiseEstimate','LevelDependent');
Nx = length(x);
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')
```
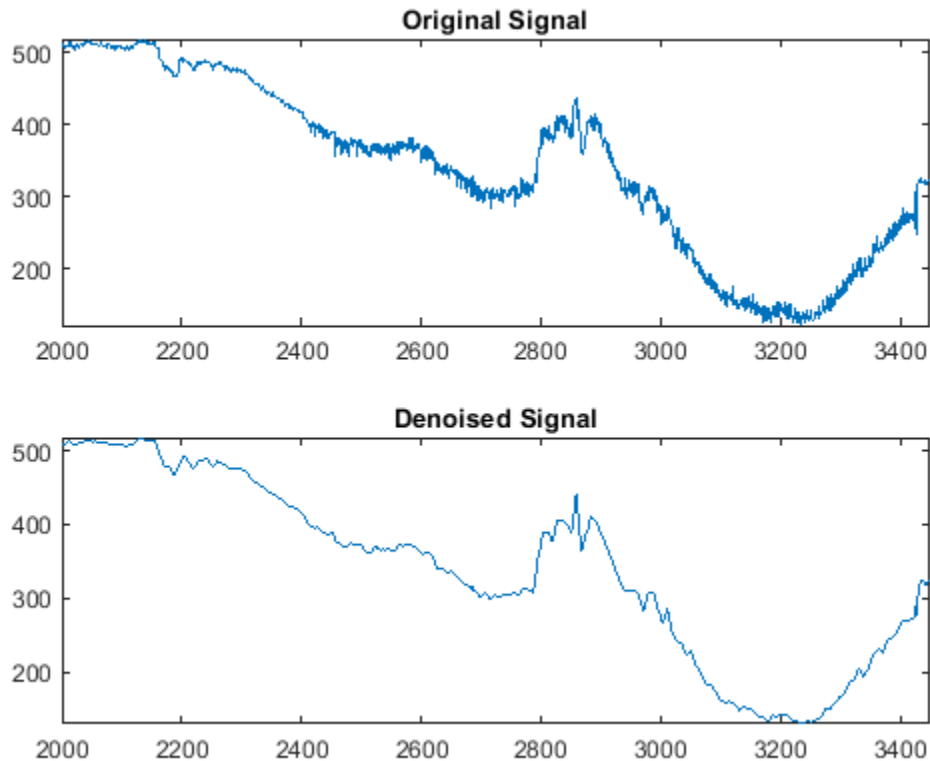

The result is quite good in spite of the time heterogeneity of the nature of the noise after and before the beginning of the sensor failure around time 2410.

Image Denoising

The denoising method described for the one-dimensional case applies also to images and applies well to geometrical images. The two-dimensional denoising procedure has the same three steps and uses two-dimensional wavelet tools instead of one-dimensional ones. For the threshold selection, prod(size(y)) is used instead of length(y) if the fixed form threshold is used.

Generate a noisy image.

```
load  woman
init = 2055615866;
rng(init);
x = X + 15*randn(size(X));
```

In this case fixed form threshold is used with estimation of level noise, thresholding mode is soft and the approximation coefficients are kept.

```
[thr,sorh,keepapp] = ddencmp('den','wv',x);
thr
```

```
thr = 107.9838
```

Denoise image using global thresholding option.

```
xd = wdencmp('gbl',x,'sym4',2,thr,sorh,keepapp);
figure('Color','white')
colormap(pink(255))
sm = size(map,1);
image(wcodemat(X,sm))
title('Original Image')
```


Original Image
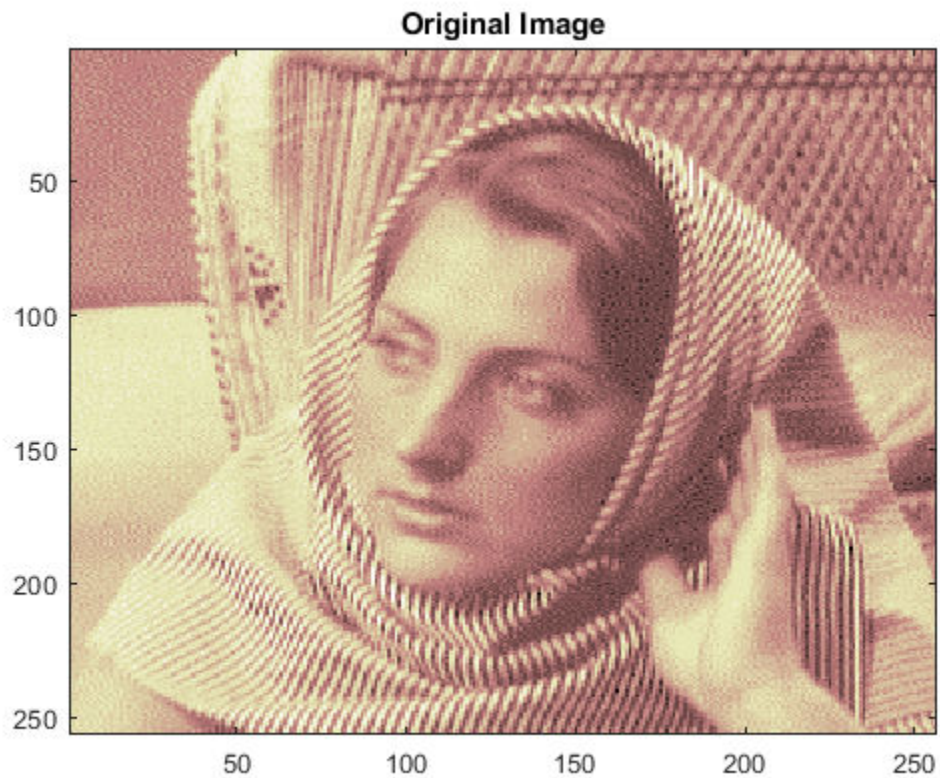
```
figure('Color','white')
colormap(pink(255))
image(wcodemat(x,sm))
title('Noisy Image')
```

Noisy Image
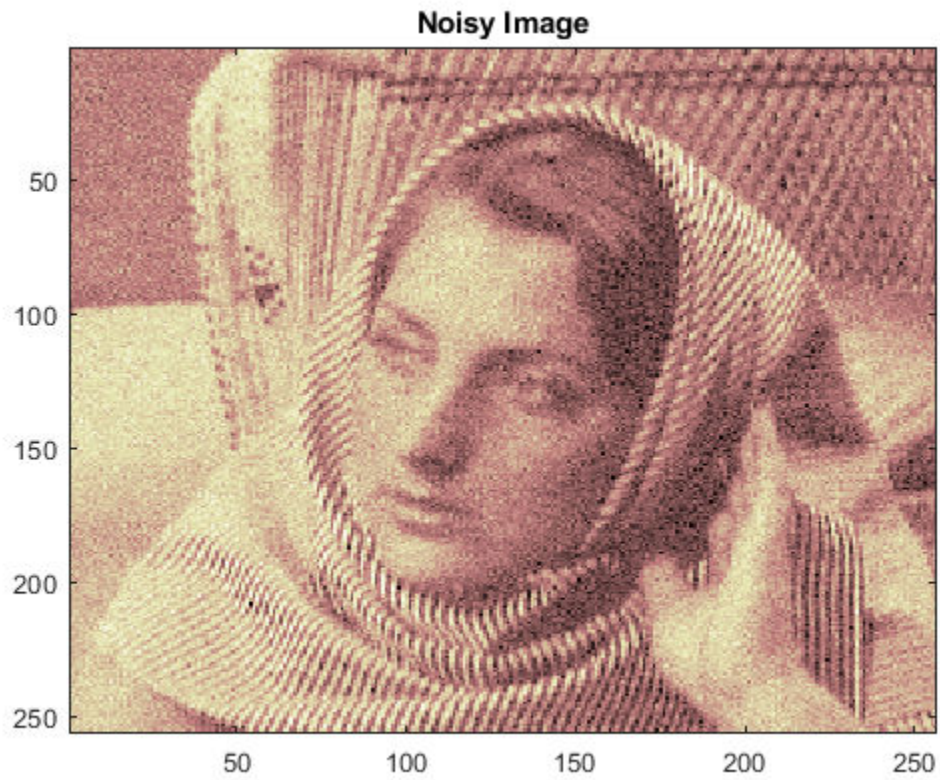

```
image(wcodemat(xd,sm))
title('Denoised Image')
```

Denoised Image

Summary

Denoising methods based on wavelet decomposition is one of the most significant applications of wavelets. See wdenoise and Wavelet Signal Denoiser for more information.

See Also

wdenoise | wdenoise2

More About

- "Denoise a Signal with the Wavelet Signal Denoiser" on page 6-25

Wavelet Interval-Dependent Denoising

The example shows how to denoise a signal using interval-dependent thresholds.

Wavelet GUI tools provide an accurate denoising process by allowing us to fine tune the parameters required to denoise a signal. Then, we can save the denoised signal, the wavelet decomposition and all denoising parameters.

It can be tedious and sometimes impossible to treat many signals using the same denoising settings. A batch mode, using the command line, can be much more efficient. This example shows how to work at the command line to simplify and solve this problem.

In this example, we perform six trials to denoise the electrical signal **nelec** using these procedures:

1   Using one interval with the minimum threshold value: 4.5

2   Using one interval with the maximum threshold value: 19.5

3   Manually selecting three intervals and three thresholds, and using the `wthresh` function to threshold the coefficients.

4   Using the `utthrset_cmd` function to automatically find the intervals and the thresholds.

5   Using the `cmddenoise` function to perform all the processes automatically.

6   Using the `cmddenoise` function with additional parameters.

Denoising Using a Single Interval

Load the electrical consumption signal **nelec**.

```
load nelec.mat
sig = nelec;
```

Now we use the discrete wavelet analysis at **level 5** with the **sym4** wavelet. We set the thresholding type to `'s'` (soft).

```
wname = 'sym4';
level = 5;
sorh  = 's';
```

Denoise the signal using the function `wdencmp` with the threshold set at **4.5**, which is the minimum value provided by the GUI tools.

```
thr = 4.5;
[sigden_1,~,~,perf0,perfl2] = wdencmp('gbl',sig,wname,level,thr,sorh,1);
res = sig-sigden_1;
subplot(3,1,1)
plot(sig,'r')
axis tight
title('Original Signal')
subplot(3,1,2)
plot(sigden_1,'b')
axis tight
title('Denoised Signal')
subplot(3,1,3)
plot(res,'k')
axis tight
title('Residual')
```

12-11
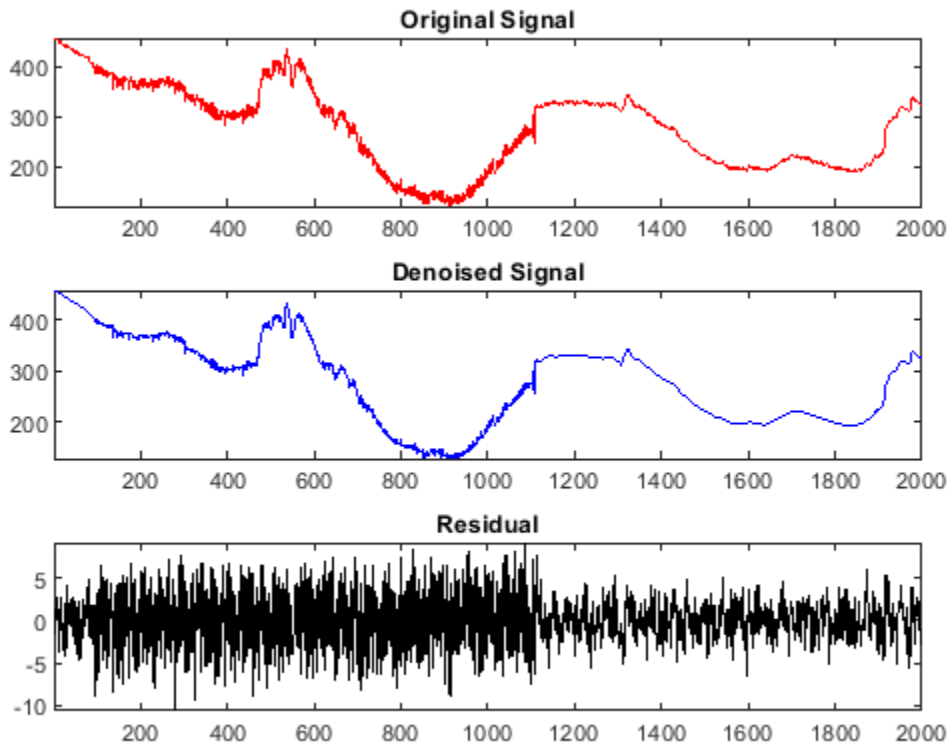
```
perf0,perfl2

perf0 = 66.6995

perfl2 = 99.9756
```

The obtained result is not good. The denoising is very efficient at the beginning and end of the signal, but between 100 and 1100 the noise is not removed. Note that the **perf0** value gives the percentage of coefficients set to zero and the **perfl2** value gives the percentage of preserved energy.

Now, we denoise the signal with the maximum value provided by the GUI tools for the threshold, **19.5**

```
thr = 19.5;
[sigden_2,cxd,lxd,perf0,perfl2] = wdencmp('gbl',sig,wname,level,thr,sorh,1);
res = sig-sigden_2;
subplot(3,1,1)
plot(sig,'r')
axis tight
title('Original Signal')
subplot(3,1,2)
plot(sigden_2,'b')
axis tight
title('Denoised Signal')
subplot(3,1,3)
plot(res,'k')
axis tight
title('Residual')
```
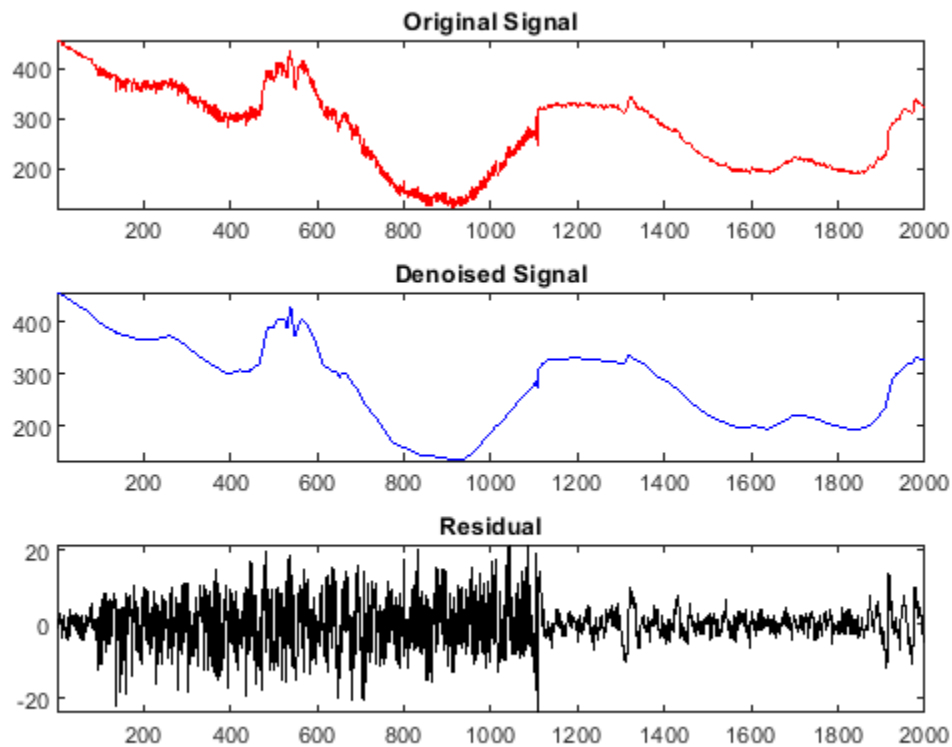
```
perf0,perfl2
```

```
perf0 = 94.7860
```

```
perfl2 = 99.9564
```

The denoised signal is very smooth. It seems quite good, but if we look at the residual after position 1100, we can see that the variance of the underlying noise is not constant. Some components of the signal have likely remained in the residual, such as, near the position 1300 and between positions 1900 and 2000.

Denoising Using the Interval Dependent Thresholds (IDT)

Now we will use an interval-dependent thresholding, as in the denoising GUI tools.

Perform a discrete wavelet analysis of the signal.

```
[coefs,longs] = wavedec(sig,level,wname);
```

Using the GUI tools to perform interval-dependent thresholding for the signal **nelec**, and setting the number of intervals at three, we get the content of the denPAR variable, which can be interpreted as follows:

- I1 = [1 94] with a threshold **thr1 = 5.9**
- I2 = [94 1110] with a threshold **thr2 = 19.5**
- I3 = [1110 2000] with a threshold **thr3 = 4.5**

Define the interval-dependent thresholds.

```
denPAR = {[1 94 5.9 ; 94 1110 19.5 ; 1110 2000 4.5]};
thrParams = cell(1,level);
thrParams(:) = denPAR;
```

Show the wavelet coefficients of the signal and the interval-dependent threshold for each level of the discrete analysis.

```
% Replicate the coefficients
cfs_beg = wrepcoef(coefs,longs);

% Display the coefficients of the decomposition
figure
subplot(6,1,1)
plot(sig,'r')
axis tight
title('Original Signal and Detail Coefficients from 1 to 5')
ylabel('S','Rotation',0)
for k = 1:level
    subplot(6,1,k+1)
    plot(cfs_beg(k,:),'Color',[0.5 0.8 0.5])
    ylabel(['D' int2str(k)],'Rotation',0)
    axis tight
    hold on
    maxi = max(abs(cfs_beg(k,:)));
    hold on
    par = thrParams{k};
    plotPar = {'Color','m','LineStyle','-.'};
    for j = 1:size(par,1)-1
        plot([par(j,2),par(j,2)],[-maxi maxi],plotPar{:})
    end
    for j = 1:size(par,1)
        plot([par(j,1),par(j,2)],[par(j,3) par(j,3)],plotPar{:})
        plot([par(j,1),par(j,2)],-[par(j,3) par(j,3)],plotPar{:})
    end
    ylim([-maxi*1.05 maxi*1.05])
    %hold off
end
subplot(6,1,level+1)
xlabel('Time or Space')
```

Original Signal and Detail Coefficients from 1 to 5

For each level `k`, the variable `thrParams{k}` contains the intervals and the corresponding thresholds for the denoising procedure.

Threshold the wavelet coefficients level-by-level, and interval-by- interval, using the values contained in the `thrParams` variable.

Using the function `wthresh`, we threshold the wavelet coefficients values between the horizontal lines by replacing them with zeros, while other values are either reduced if `sorh = 's'` or remain unchanged if `sorh = 'h'`.

```
first = cumsum(longs)+1;
first = first(end-2:-1:1);
tmp   = longs(end-1:-1:2);
last  = first+tmp-1;
for k = 1:level
    thr_par = thrParams{k};
    if ~isempty(thr_par)
        cfs = coefs(first(k):last(k));
        nbCFS = longs(end-k);
        NB_int = size(thr_par,1);
        x = [thr_par(:,1) ; thr_par(NB_int,2)];
        alf = (nbCFS-1)/(x(end)-x(1));
        bet = 1 - alf*x(1);
        x = round(alf*x+bet);
        x(x<1) = 1;
        x(x>nbCFS) = nbCFS;
        thr = thr_par(:,3);
```

```matlab
        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);
            cfs(j_ind) = wthresh(cfs(j_ind),sorh,thr(j));
        end
        coefs(first(k):last(k)) = cfs;
    end
end
```

Show the thresholded wavelet coefficients of the signal.

```matlab
% Replicate the coefficients.
cfs_beg = wrepcoef(coefs,longs);

% Display the decomposition coefficients.
figure
subplot(6,1,1)
plot(sig,'r')
axis tight
title('Original Signal and Detail Coefficients from 1 to 5')
ylabel('S','Rotation',0)
for k = 1:level
    subplot(6,1,k+1)
    plot(cfs_beg(k,:),'Color',[0.5 0.8 0.5])
    ylabel(['D' int2str(k)],'Rotation',0)
    axis tight
    hold on
    maxi = max(abs(cfs_beg(k,:)));
    %hold on
    par = thrParams{k};
    plotPar = {'Color','m','LineStyle','-.'};
    for j = 1:size(par,1)-1
        plot([par(j,2),par(j,2)],[-maxi maxi],plotPar{:})
    end
    for j = 1:size(par,1)
        plot([par(j,1),par(j,2)], [par(j,3) par(j,3)],plotPar{:})
        plot([par(j,1),par(j,2)],-[par(j,3) par(j,3)],plotPar{:})
    end
    ylim([-maxi*1.05 maxi*1.05])
    hold off
end
subplot(6,1,level+1)
xlabel('Time or Space')
```

Original Signal and Detail Coefficients from 1 to 5

Reconstruct the denoised signal.

```
sigden = waverec(coefs,longs,wname);
res = sig - sigden;
```

Display the original, denoised, and residual signals.

```
figure
subplot(3,1,1)
plot(sig,'r')
hold on
plot(sigden,'b')
axis tight
title('Original and Denoised Signals')
subplot(3,1,2)
plot(sigden,'b')
axis tight
title('Denoised Signal')
subplot(3,1,3)
plot(res,'k')
axis tight
title('Residual')
```

Original and Denoised Signals

Denoised Signal

Residual
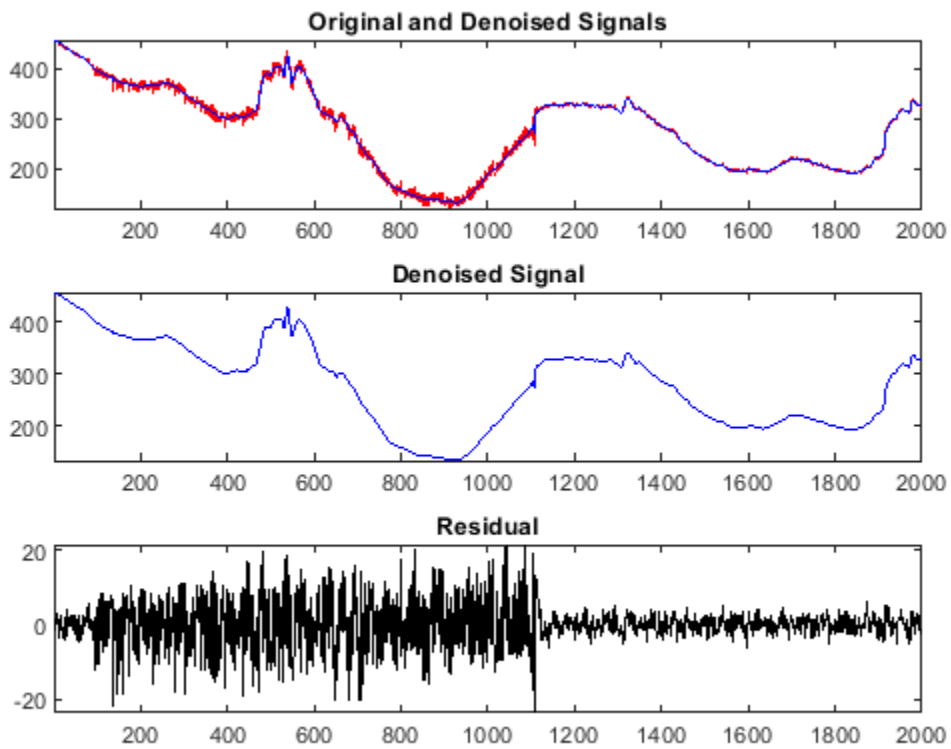
Compare the three denoised versions of the signal.

```
figure
plot(sigden_1,'g')
hold on
plot(sigden_2,'r')
plot(sigden,'k')
axis tight
hold off
legend('Denoised Min','Denoised Max','Denoised IDT','Location','North')
```
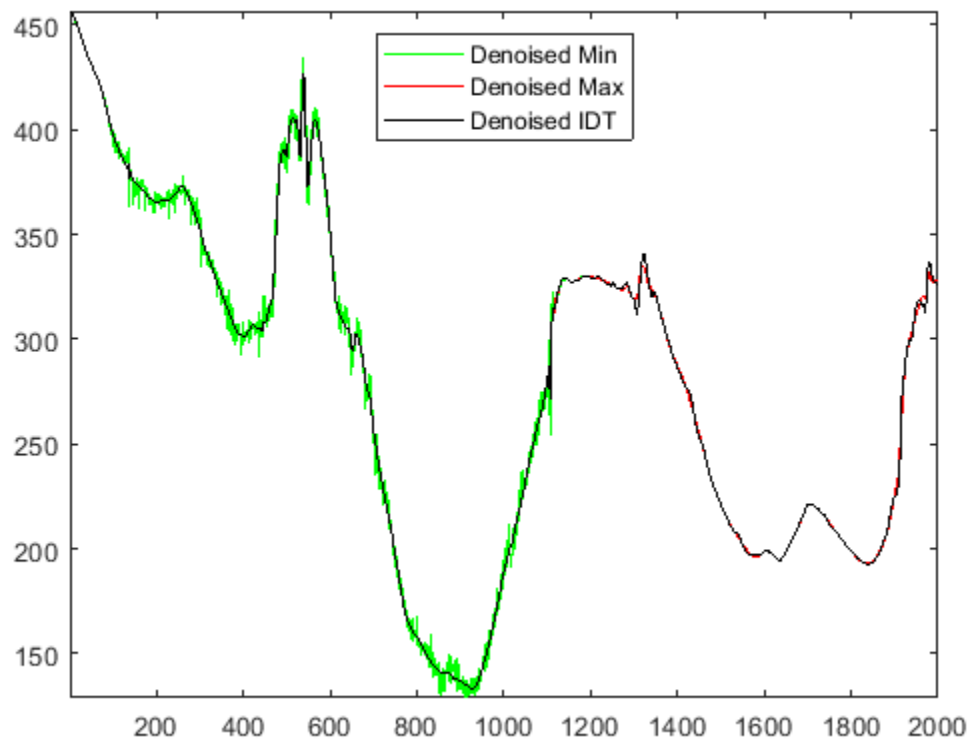
Looking at the first half of the signals, it is clear that denoising using the minimum value of the threshold is not good. Now, we zoom on the end of the signal for more details.

```
xlim([1200 2000])
ylim([180 350])
```
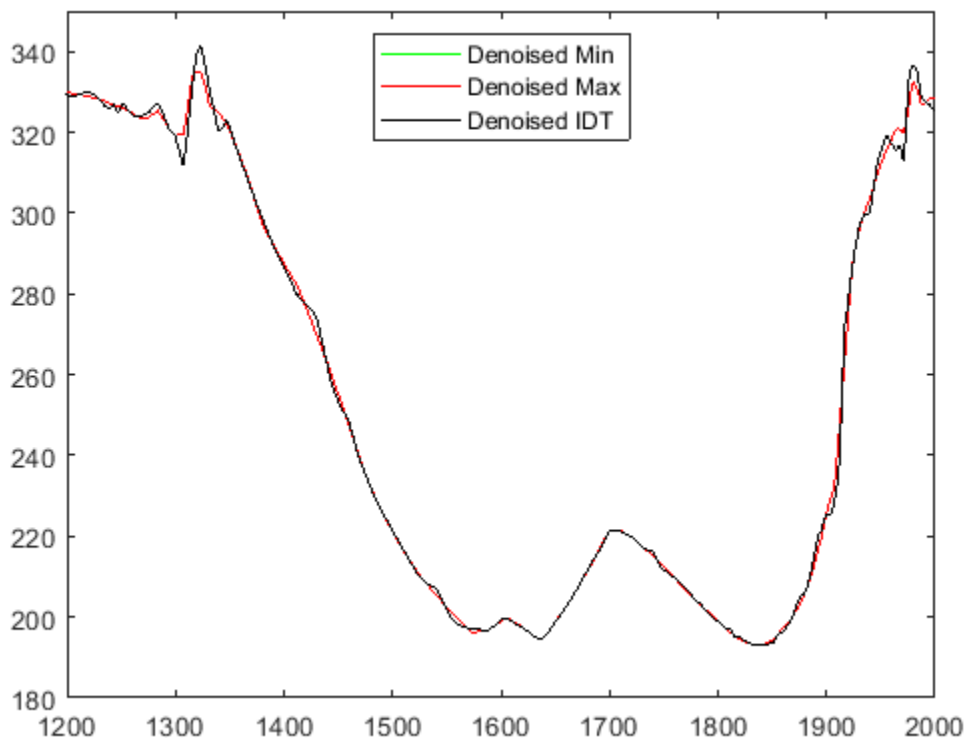
We can see that when the maximum threshold value is used, the denoised signal is smoothed too much and information is lost.

The best result is given by using the threshold based on the interval- dependent thresholding method, as we will show now.

Automatic Computation of Interval-Dependent Thresholds

Instead of manually setting the intervals and the thresholds for each level, we can use the function `utthrset_cmd` to automatically compute the intervals and the thresholds for each interval. Then, we complete the procedure by applying the thresholds, reconstructing, and displaying the signal.

```
% Wavelet Analysis.
[coefs,longs] = wavedec(sig,level,wname);
siz = size(coefs);
thrParams = utthrset_cmd(coefs,longs);
first = cumsum(longs)+1;
first = first(end-2:-1:1);
tmp   = longs(end-1:-1:2);
last  = first+tmp-1;
for k = 1:level
    thr_par = thrParams{k};
    if ~isempty(thr_par)
        cfs = coefs(first(k):last(k));
        nbCFS = longs(end-k);
        NB_int = size(thr_par,1);
        x = [thr_par(:,1) ; thr_par(NB_int,2)];
```

```
        alf = (nbCFS-1)/(x(end)-x(1));
        bet = 1 - alf*x(1);
        x = round(alf*x+bet);
        x(x<1) = 1;
        x(x>nbCFS) = nbCFS;
        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);
            cfs(j_ind) = wthresh(cfs(j_ind),sorh,thr(j));
        end
        coefs(first(k):last(k)) = cfs;
    end
end
sigden = waverec(coefs,longs,wname);
figure
subplot(2,1,1)
plot(sig,'r')
axis tight
hold on
plot(sigden,'k')
title('Original and Denoised Signals')
subplot(2,1,2)
plot(sigden,'k')
axis tight
hold off
title('Denoised Signal')
```

Original and Denoised Signals
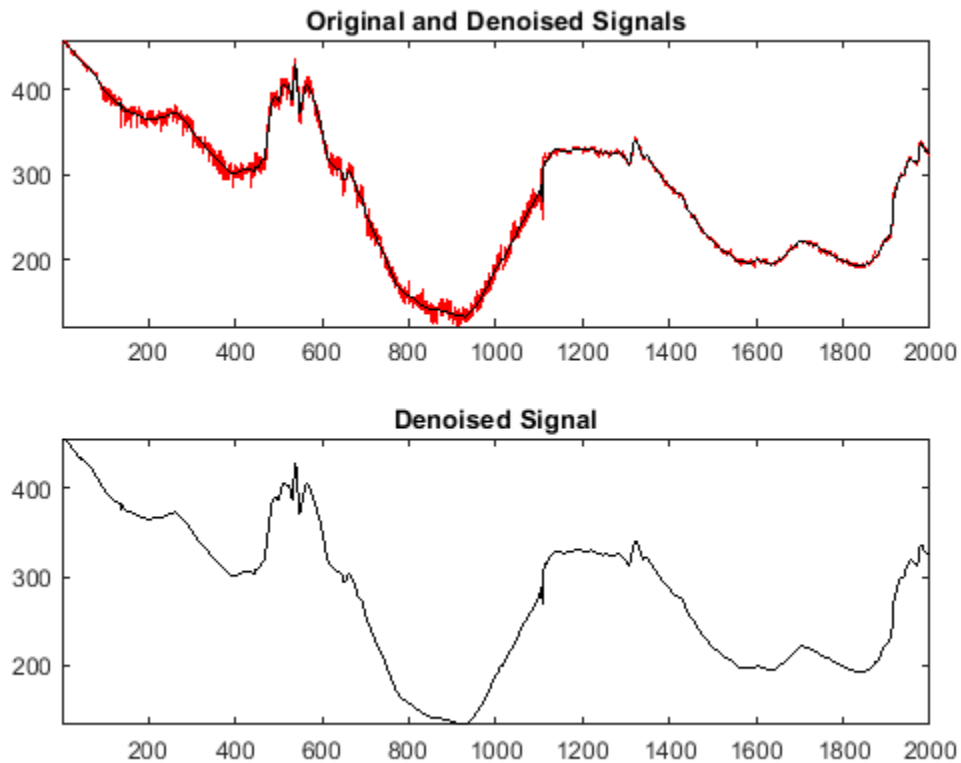
Denoised Signal

Automatic Interval-Dependent Denoising

In command-line mode, we can use the function `cmddenoise` to automatically compute the denoised signal and coefficients based on the interval-dependent denoising method. This method performs the whole process of denoising using only this one function, which includes all the steps described earlier in this example.

```
[sigden,~,thrParams] = cmddenoise(sig,wname,level);
thrParams{1}    % Denoising parameters for level 1.

ans = 2×3
10³ ×

    0.0010    1.1100    0.0176
    1.1100    2.0000    0.0045
```

The automatic procedure finds two intervals for denoising:

- `I1 = [1 1110]` with a threshold **thr1 = 17.6**
- `I2 = [1110 2000]` with a threshold **thr2 = 4.5**.

We can display the result of denoising and see that the result is fine.

```
figure
subplot(2,1,1)
plot(sig,'r')
```

```
axis tight
hold on
plot(sigden,'k')
title('Original and Denoised Signals')
hold off
subplot(2,1,2)
plot(sigden,'k')
axis tight
title('Denoised Signal')
```

Original and Denoised Signals

Denoised Signal
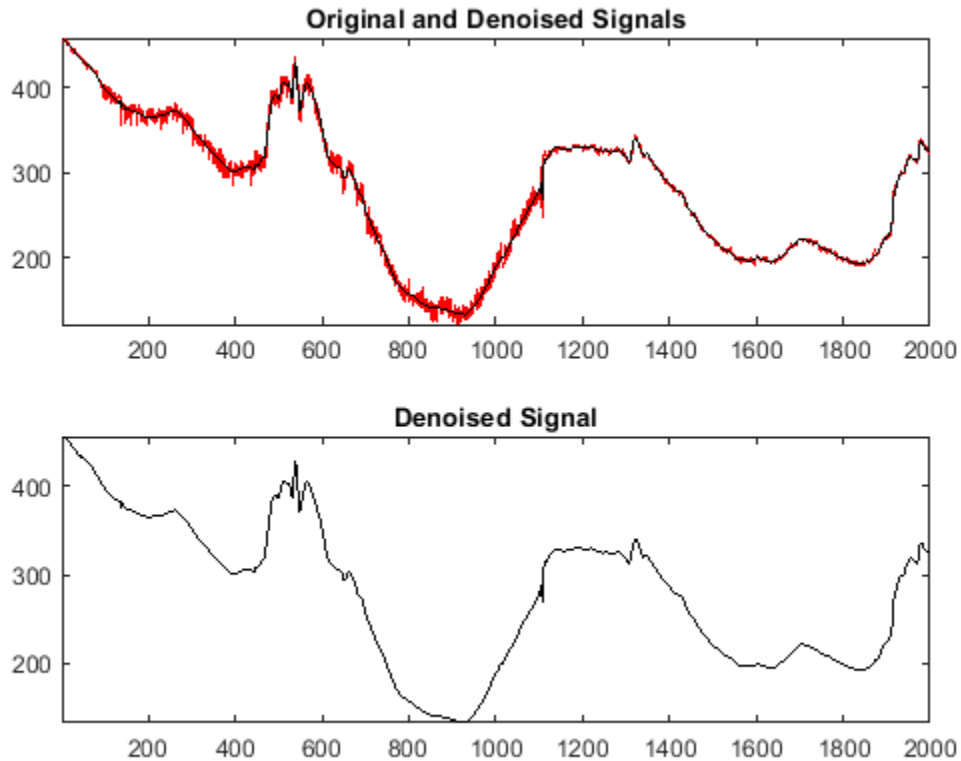
Advanced Automatic Interval-Dependent Denoising

Now, we look at a more complete example of the automatic denoising.

Rather than using the default values for the input parameters, we can specify them when calling the function. Here, the type of threshold is chosen as s (soft) and the number of intervals is set at 3.

```
load nelec.mat;
sig = nelec;   % Signal to analyze.
wname  = 'sym4';              % Wavelet for analysis.
level  = 5;                   % Level for wavelet decomposition.
sorh   = 's';                 % Type of thresholding.
nb_Int = 3;                   % Number of intervals for thresholding.

[sigden,coefs,thrParams,int_DepThr_Cell,BestNbOfInt] = ...
            cmddenoise(sig,wname,level,sorh,nb_Int);
```

12-23

For the output parameters, the variable `thrParams{1}` gives the denoising parameters for levels from 1 to 5. For example, here are the denoising parameters for level 1.

`thrParams{1}`

```
ans = 3×3
10³ ×

    0.0010    0.0940    0.0059
    0.0940    1.1100    0.0195
    1.1100    2.0000    0.0045
```

We find the same values that were set earlier this example. They correspond to the choice we have made by fixing the number of intervals to three in the input parameter: `nb_Int = 3`.

The automatic procedure suggests **2** as the best number of intervals for denoising. This output value `BestNbOfInt = 2` is the same as used in the previous step of this example.

`BestNbOfInt`

```
BestNbOfInt = 2
```

The variable `int_DepThr_Cell` contains the interval locations and the threshold values for a number of intervals from 1 to 6.

`int_DepThr_Cell`

```
int_DepThr_Cell=1×6 cell array
  Columns 1 through 4

    {1x3 double}    {2x3 double}    {3x3 double}    {4x3 double}

  Columns 5 through 6

    {5x3 double}    {6x3 double}
```

Finally, we look at the values corresponding to the locations and thresholds for 5 intervals.

`int_DepThr_Cell{5}`

```
ans = 5×3
10³ ×

    0.0010    0.0940    0.0059
    0.0940    0.5420    0.0184
    0.5420    0.5640    0.0056
    0.5640    1.1100    0.0240
    1.1100    2.0000    0.0045
```

Summary

This example shows how to use command-line mode to achieve the same capabilities as the GUI tools for denoising, while giving you more control over particular parameter values to obtain better results.

See Also

Wavelet Signal Denoiser | wdenoise | wdenoise2

More About

•   "Denoise a Signal with the Wavelet Signal Denoiser" on page 6-25

Multivariate Wavelet Denoising

The purpose of this example is to show the features of multivariate denoising provided in Wavelet Toolbox™.

Multivariate wavelet denoising problems deal 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 example uses a number of noise signals and performs the following steps to denoise the deterministic signal.

Loading a Multivariate Signal

To load the multivariate signal, type the following code at the MATLAB® prompt:

```
load ex4mwden
whos
```

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

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 signal is "Blocks" which is irregular, and the second one is "HeavySine" which is regular, except around time 750. The other two signals are the sum and the difference of the two original signals, respectively. Multivariate Gaussian white noise exhibiting strong spatial correlation is added to the resulting four signals, which produces the observed data stored in x.

Displaying the Original and Observed Signals

To display the original and observed signals, type:

```
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
```
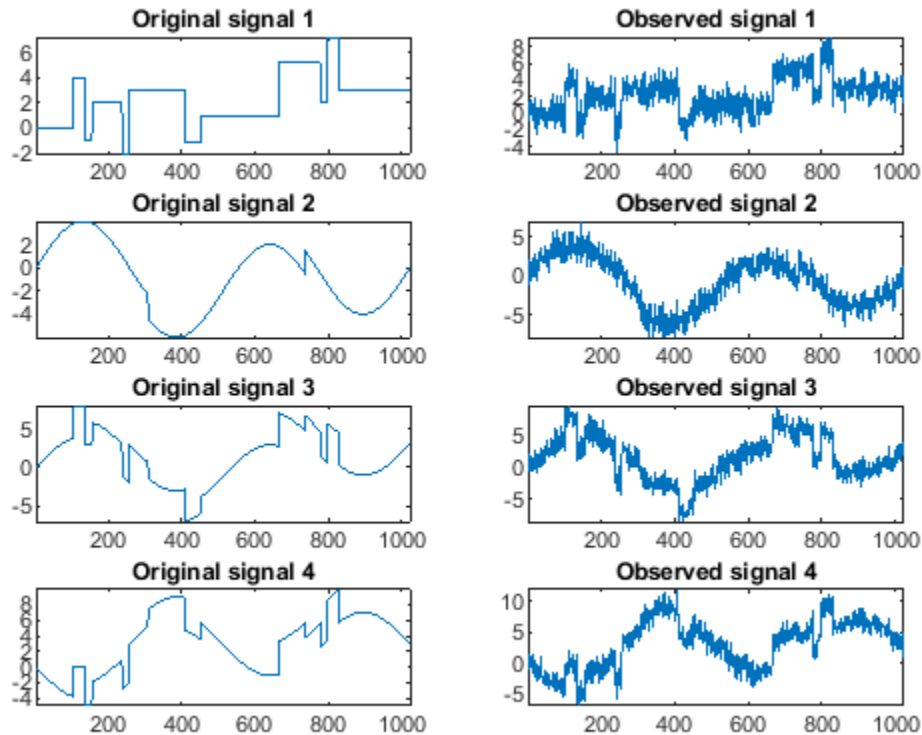
The true noise covariance matrix is given by:

```
covar
```

```
covar = 4×4

    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
```

Removing 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 lines 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);
```

Displaying the Original and Denoised Signals

To display the original and denoised signals type the following:

```
clf
kp = 0;
for i = 1:4
    subplot(4,3,kp+1), plot(x_orig(:,i)); axis tight;
    title(['Original signal ',num2str(i)])
    subplot(4,3,kp+2), plot(x(:,i)); axis tight;
    title(['Observed signal ',num2str(i)])
    subplot(4,3,kp+3), plot(x_den(:,i)); axis tight;
    title(['Denoised signal ',num2str(i)])
    kp = kp + 3;
end
```
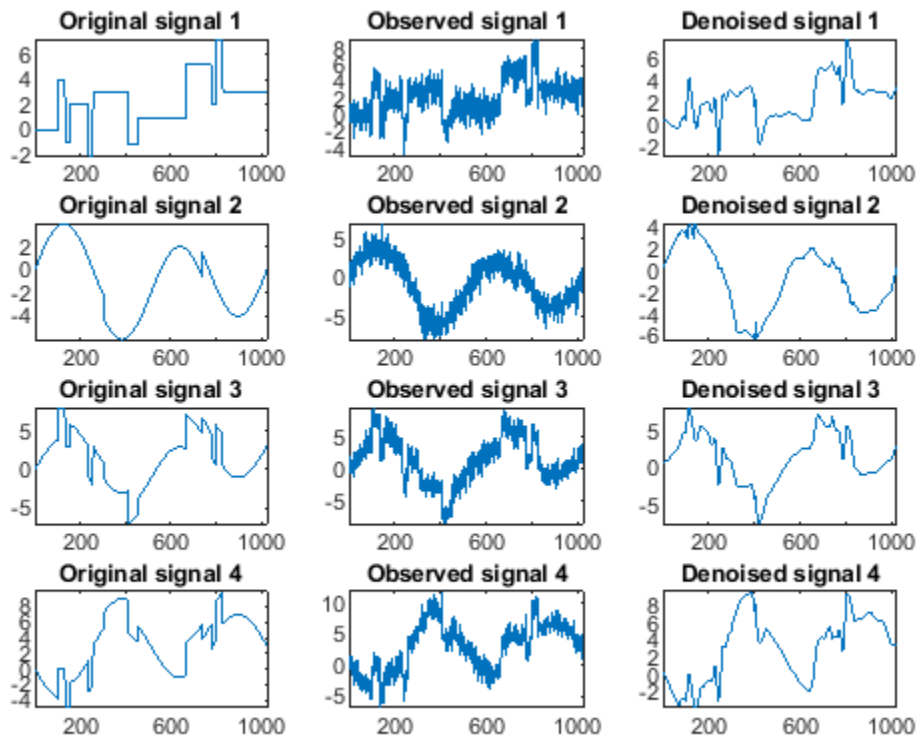

Improving the First Result by Retaining Fewer Principal Components

We can see that, overall, the results are satisfactory. Focusing on the two first signals, note that they are correctly recovered, but we can improve the result by taking advantage of the relationships between the signals, leading to an additional denoising effect.

To automatically select the numbers of retained principal components using Kaiser's rule, which retains 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);
```

Displaying the Number of Retained Principal Components

The second output argument `npc` is the number of retained principal components for PCA for approximations and for final PCA.

```
npc
```

npc = *1×2*

```
    2     2
```

As expected, because the signals are combinations of two original signals, Kaiser's rule automatically detects that only two principal components are of interest.

Displaying the Estimated Noise Covariance Matrix

The third output argument `nestco` contains the estimated noise covariance matrix:

```
nestco
```

nestco = *4×4*

```
    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 it can be seen by comparing it with the true matrix covar given previously, the estimation is satisfactory.

Displaying the Original and Final Denoised Signals

To display the original and final denoised signals type:

```
kp = 0;
for i = 1:4
    subplot(4,3,kp+1), plot(x_orig(:,i)); axis tight;
    title(['Original signal ',num2str(i)])
    subplot(4,3,kp+2), plot(x(:,i)); axis tight;
    title(['Observed signal ',num2str(i)])
    subplot(4,3,kp+3), plot(x_den(:,i)); axis tight;
    title(['Denoised signal ',num2str(i)])
    kp = kp + 3;
end
```
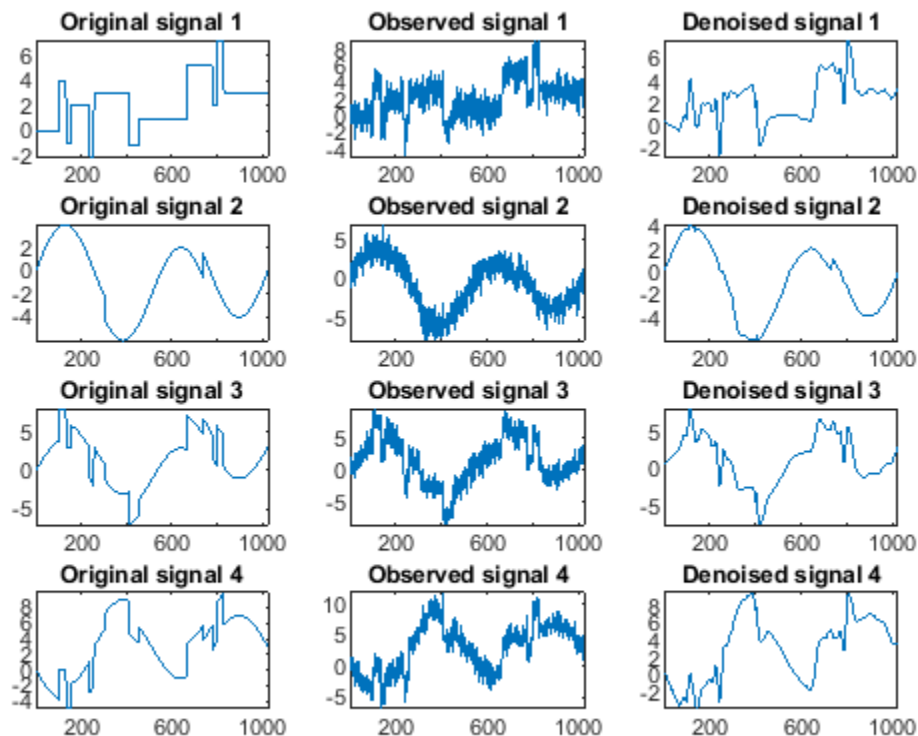
These results are better than those previously obtained. The first signal, which is irregular, is still correctly recovered, while the second signal, which is more regular, is better denoised after this second stage of PCA.

Learning More About Multivariate Denoising

You can find more information about multivariate denoising, including some theory, simulations, and real examples, in the following reference:

M. Aminghafari, N. Cheze and J-M. Poggi (2006), "Multivariate denoising using wavelets and principal component analysis," Computational Statistics & Data Analysis, 50, pp. 2381-2398.

Multiscale Principal Components Analysis

The purpose of this example is to show the features of multiscale principal components analysis (PCA) provided in the Wavelet Toolbox™.

The aim of multiscale PCA is to reconstruct a simplified multivariate signal, starting from a multivariate signal and using a simple representation at each resolution level. Multiscale principal components analysis generalizes the PCA of a multivariate signal represented as a matrix by simultaneously performing a PCA on the matrices of details of different levels. 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. This example uses a number of noisy test signals and performs the following steps.

Load Multivariate Signal

Load the multivariate signal by typing the following at the MATLAB(R) prompt:

```
load ex4mwden
whos
```

```
  Name         Size           Bytes  Class      Attributes

  covar        4x4              128  double
  x            1024x4         32768  double
  x_orig       1024x4         32768  double
```

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 signal is "Blocks" which is irregular, and the second one is "HeavySine" which is regular, except around time 750. The other two signals are the sum and the difference of the two original signals, respectively. Multivariate Gaussian white noise exhibiting strong spatial correlation is added to the resulting four signals, which produces the observed data stored in `x`.

Perform Simple Multiscale PCA

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, which retains components associated with eigenvalues exceeding the mean of all eigenvalues, by typing:

```
npc = 'kais';
```

Finally, perform multiscale PCA:

```
[x_sim, qual, npc] = wmspca(x ,level, wname, npc);
```

Display the Original and Simplified Signals

To display the original and simplified signals type:

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

Original signal 1 **Simplified signal 1**
Original signal 2 **Simplified signal 2**
Original signal 3 **Simplified signal 3**
Original signal 4 **Simplified signal 4**
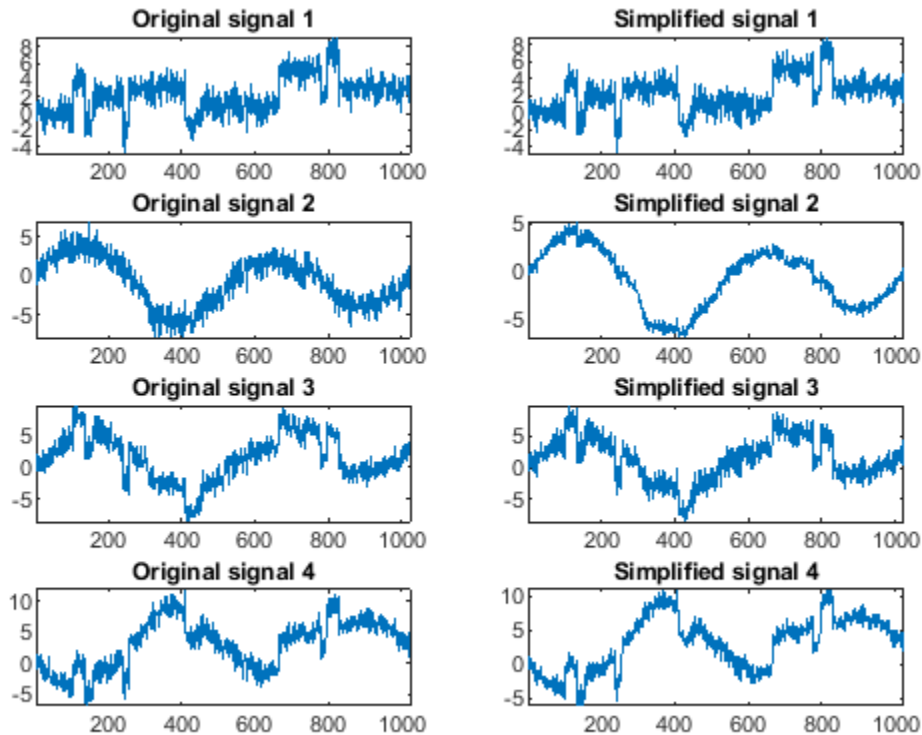
We can see that the results from a compression perspective are good. The percentages reflecting the quality of column reconstructions given by the relative mean square errors are close to 100%.

```
qual
```

```
qual = 1×4

   98.0545   93.2807   97.1172   98.8603
```

Improve the First Result by Retaining Fewer Principal Components

We can improve the results 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 efficient, denoising effect.

The output argument `npc` is the number of retained principal components selected by Kaiser's rule:

```
npc
```

```
npc = 1×7

    1     1     1     1     1     2     2
```

For *d* from 1 to 5, `npc`(*d*) is the number of retained noncentered principal components (PCs) for details at level *d*. `npc`(6) is the number of retained non-centered PCs for approximations at level 5, and `npc` (7) is the number of retained PCs for final PCA after wavelet reconstruction. As expected, the rule keeps two principal components, both for the PCA approximations and the final PCA, but one principal component is kept for details at each level.

To suppress the details at levels 1 to 3, update the `npc` argument as follows:

```
npc(1:3) = zeros(1,3);
npc
```

```
npc = 1×7

    0     0     0     1     1     2     2
```

Then, perform multiscale PCA again by typing:

```
[x_sim, qual, npc] = wmspca(x, level, wname, npc);
```

Display the Original and Final Simplified Signals

To display the original and final simplified signals type:

```
kp = 0;
for i = 1:4
    subplot(4,2,kp+1), plot(x (:,i)); axis tight;
    title(['Original signal ',num2str(i)])
    subplot(4,2,kp+2), plot(x_sim(:,i)); axis tight;
    title(['Simplified signal ',num2str(i)])
    kp = kp + 2;
end
```
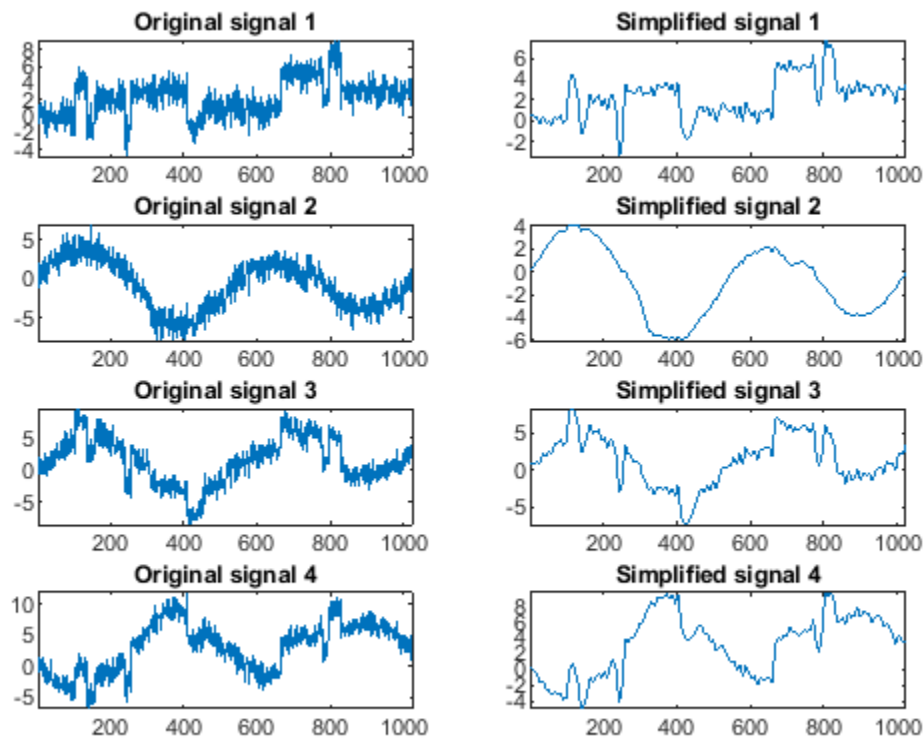
As we can see above, the results are improved.

More about Multiscale Principal Components Analysis

More about multiscale PCA, including some theory, simulations and real examples, can be found in the following reference:

Aminghafari, M.; Cheze, N.; Poggi, J-M. (2006), "Multivariate denoising using wavelets and principal component analysis," Computational Statistics & Data Analysis, 50, pp. 2381-2398.

Data Compression using 2D Wavelet Analysis

The purpose of this example is to show how to compress an image using two-dimensional wavelet analysis. Compression is one of the most important applications of wavelets. Like de-noising, the compression procedure contains three steps:

- Decompose: Choose a wavelet, choose a level N. Compute the wavelet decomposition of the signal at level N.
- Threshold detail coefficients: For each level from 1 to N, a threshold is selected and hard thresholding is applied to the detail coefficients.
- 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 with the de-noising procedure is found in step 2. There are two compression approaches available:

- The first consists of taking the wavelet expansion of the signal and keeping the largest absolute value coefficients. In this case, you can set a global threshold, a compression performance, or a relative square norm recovery performance. Thus, only a single parameter needs to be selected.
- The second approach consists of applying visually determined level-dependent thresholds.

Load an Image

Let us examine a real-life example of compression for a given and unoptimized wavelet choice, to produce a nearly complete square norm recovery for an image.

```
load woman;               % Load original image.
image(X)
title('Original Image')
colormap(map)
```

Original Image
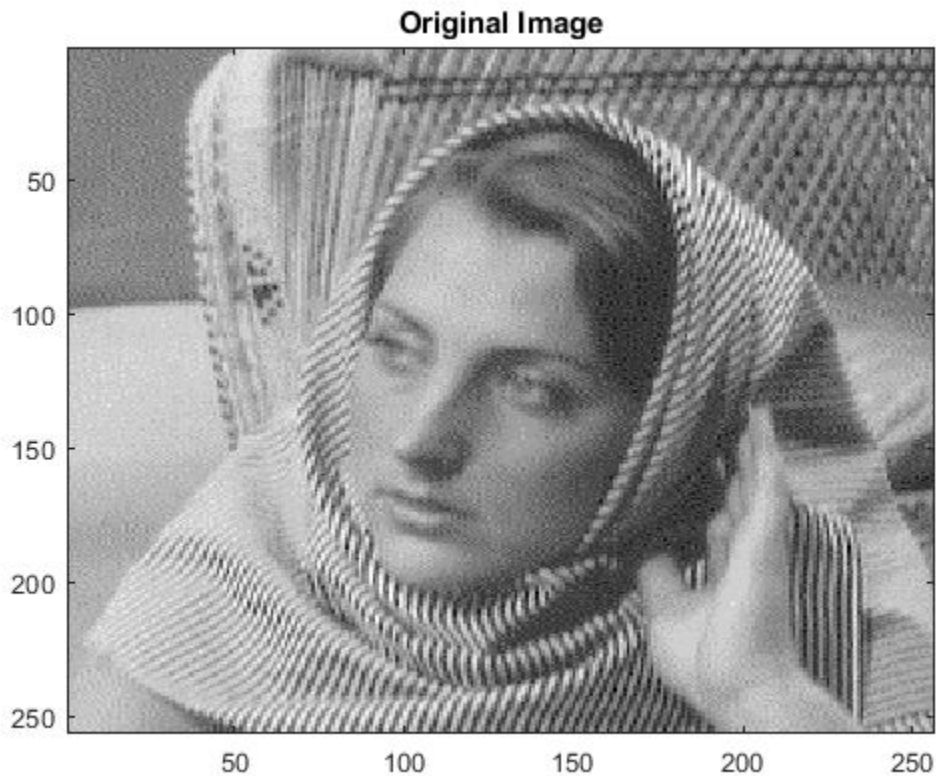
```
x = X(100:200,100:200);  % Select ROI
```

Method 1: Global Thresholding

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.

```
n = 5;                    % Decomposition Level
w = 'sym8';               % Near symmetric wavelet
[c l] = wavedec2(x,n,w);  % Multilevel 2-D wavelet decomposition.
```

In this first method, the WDENCMP function performs a compression process from the wavelet decomposition structure [c,l] of the image.

```
opt = 'gbl'; % Global threshold
thr = 20;    % Threshold
sorh = 'h';  % Hard thresholding
keepapp = 1; % Approximation coefficients cannot be thresholded
[xd,cxd,lxd,perf0,perfl2] = wdencmp(opt,c,l,w,n,thr,sorh,keepapp);
image(x)
title('Original Image')
colormap(map)
```

Original Image
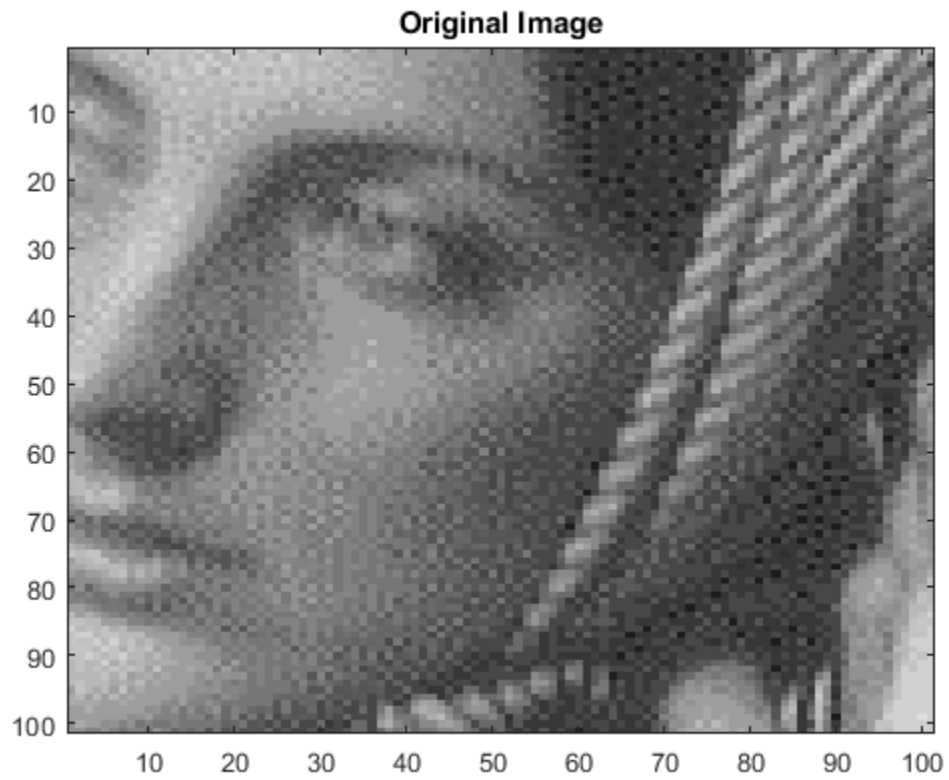

```
figure('Color','white'),image(xd)
title('Compressed Image - Global Threshold = 20')
colormap(map)
```

Compressed Image - Global Threshold = 20
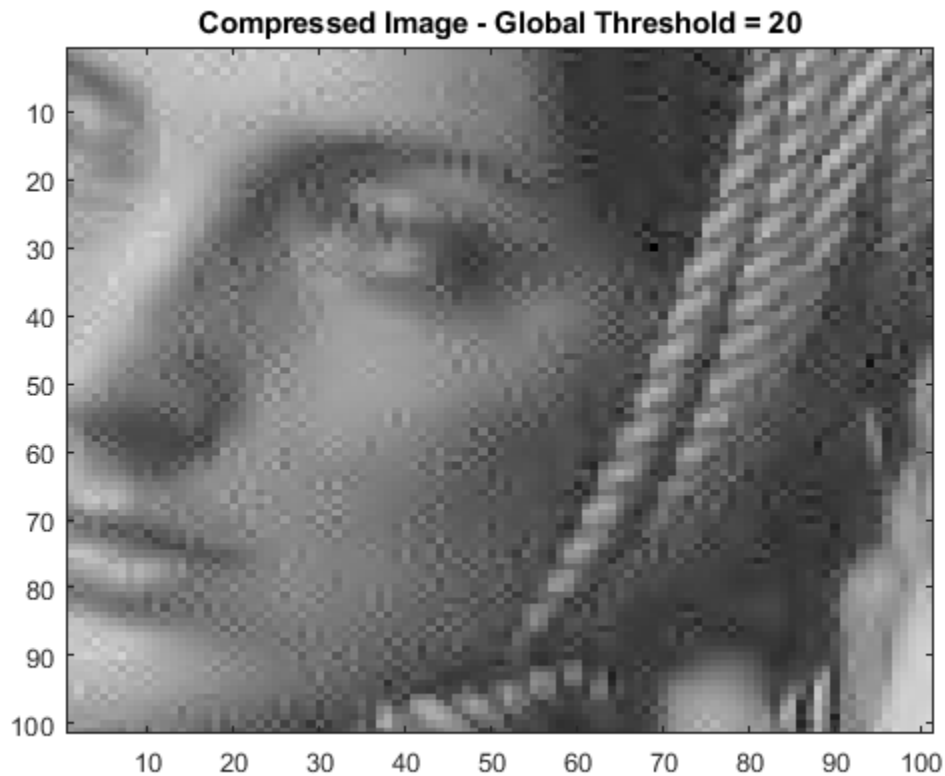
L2-norm recovery (%)

`perf0`

`perf0 = 74.3067`

Compression score (%)

`perfl2`

`perfl2 = 99.9772`

The density of the current decomposition sparse matrix is:

```
cxd = sparse(cxd);
cxd_density = nnz(cxd)/prod(size(cxd))
```

`cxd_density = 0.2569`

Method 2: Level-Dependent Thresholding

The WDENCMP function also allows level and orientation-dependent thresholds. In this case the approximation is kept. The level-dependent thresholds in the three orientations horizontal, diagonal and vertical are as follows:

```
opt = 'lvd';          % Level dependent thresholds
thr_h = [17 18];      % Horizontal thresholds.
thr_d = [19 20];      % Diagonal thresholds.
```

```
thr_v = [21 22];     % Vertical thresholds.
thr = [thr_h ; thr_d ; thr_v];
```

In this second example, notice that the WDENCMP function performs a compression process from the image x.

```
[xd2,cxd2,lxd2,perf02,perfl22] = wdencmp(opt,x,w,2,thr,sorh);
image(x)
title('Original Image')
colormap(map)
```

Original Image
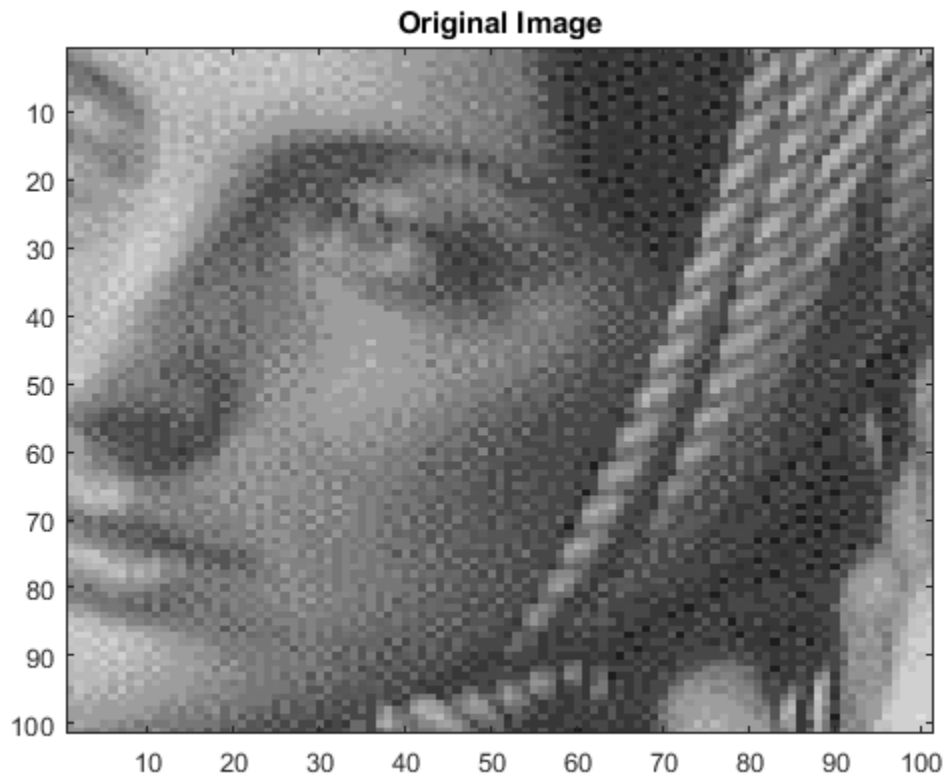
```
figure('Color','white'),image(xd2)
title('Compressed Image - Level-Dependent Thresholding')
colormap(map)
```

Compressed Image - Level-Dependent Thresholding
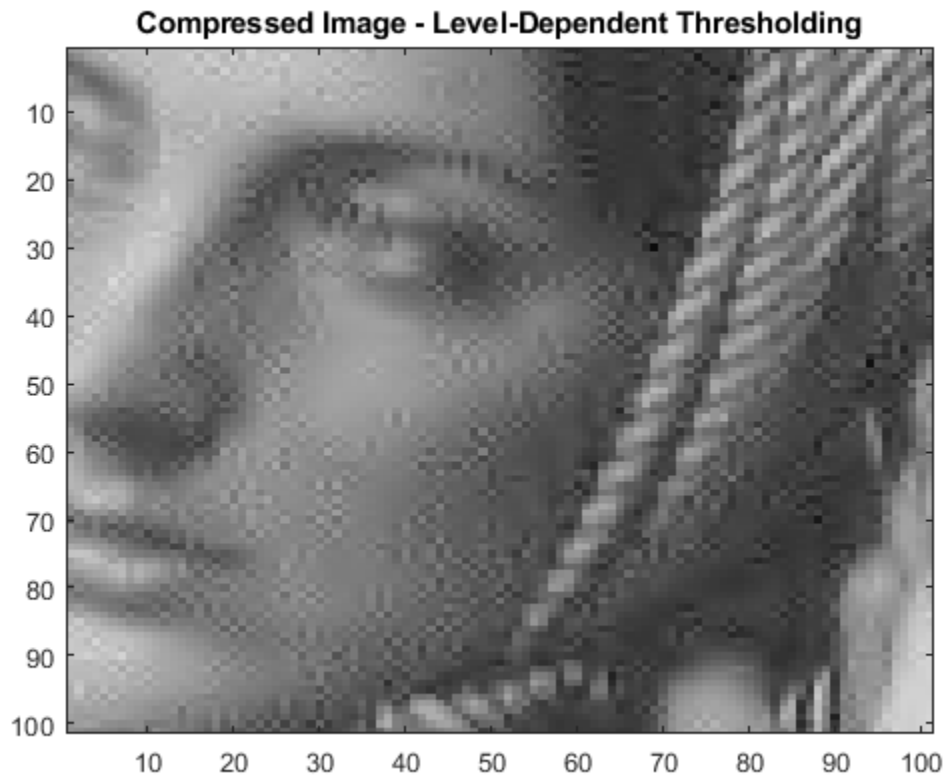
L2-norm recovery (%)

```
perf02
```

```
perf02 = 77.3435
```

Compression score (%)

```
perfl22
```

```
perfl22 = 99.6132
```

The density of the current decomposition sparse matrix is:

```
cxd2 = sparse(cxd2);
cxd2_density = nnz(cxd2)/prod(size(cxd2))
```

```
cxd2_density = 0.2266
```

Summary

By using level-dependent thresholding, the density of the wavelet decomposition was reduced by 3% while improving the L2-norm recovery by 3%. If the wavelet representation is too dense, similar strategies can be used in the wavelet packet framework to obtain a sparser representation. You can then determine the best decomposition with respect to a suitably selected entropy-like criterion, which corresponds to the selected purpose (de-noising or compression).

Smoothing Nonuniformly Sampled Data

This example shows to smooth and denoise nonuniformly sampled data using the multiscale local polynomial transform (MLPT). The MLPT is a lifting scheme (Jansen, 2013) that shares many characteristics of the discrete wavelet transform and works with nonuniformly sampled data.

Many real-world time series have observations that are not recorded at regularly spaced intervals. This may be a result from a nonuniform sampling of the data or from missing or corrupted observations. The discrete wavelet transform (DWT) is a powerful tool for denoising data or performing nonparametric regression, but the classic DWT is defined for uniformly sampled data. The concept of scale, which is central to the DWT, crucially depends on a regular interval between observations.

The lifting scheme (second-generation wavelets) (Jansen & Oonincx, 2005) provides a way to design wavelets and implement the wavelet transform entirely in the time (spatial) domain. While the classic DWT may also be represented by a lifting scheme, lifting is also flexible enough to handle nonuniformly sampled data.

Denoising Data

Load and plot a noisy nonuniformly sampled time series.

```
load skyline
plot(T,y)
xlabel('Seconds')
grid on
```
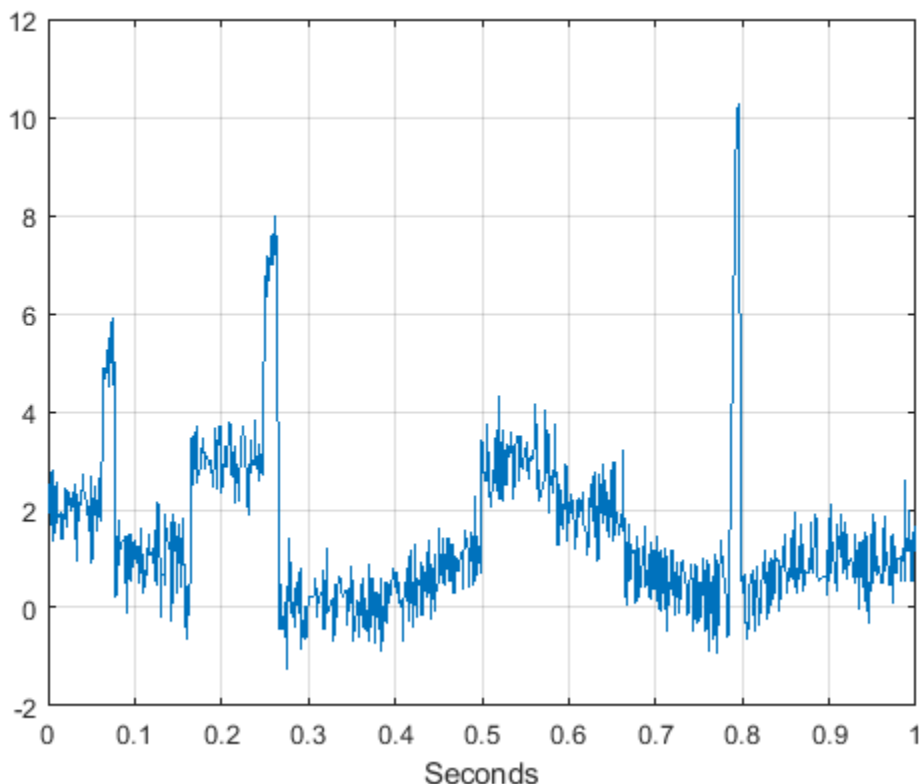
If you plot the difference of the sampling times, you see that the data is nonuniformly sampled.

```
plot(diff(T))
title('First Difference of Sampling Times')
axis tight
ylabel('\Delta T')
xlabel('Sample')
grid on
```
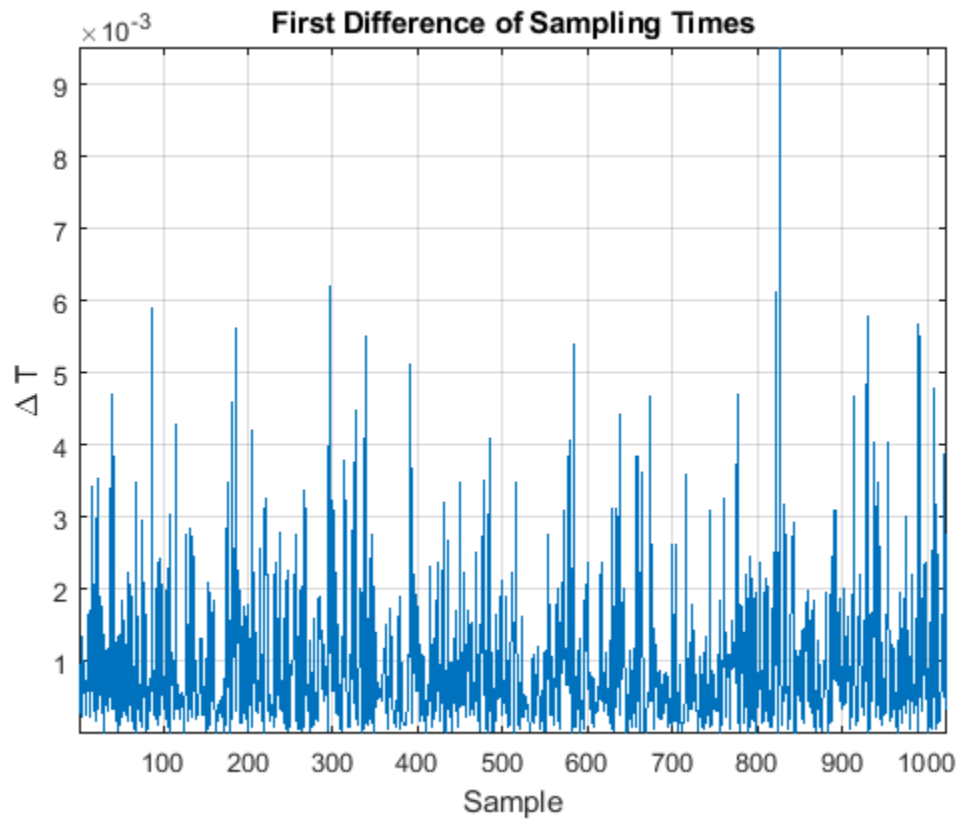

In this synthetic dataset, you have access to the original noise-free signal. If you plot that data, you see that the data are characterized by smoothly varying features as well as abrupt transient features near 0.07, 0.26, and 0.79 seconds.

```
plot(T,f)
xlabel('Seconds')
grid on
title('Original (Noise-Free) Data')
```

Original (Noise-Free) Data
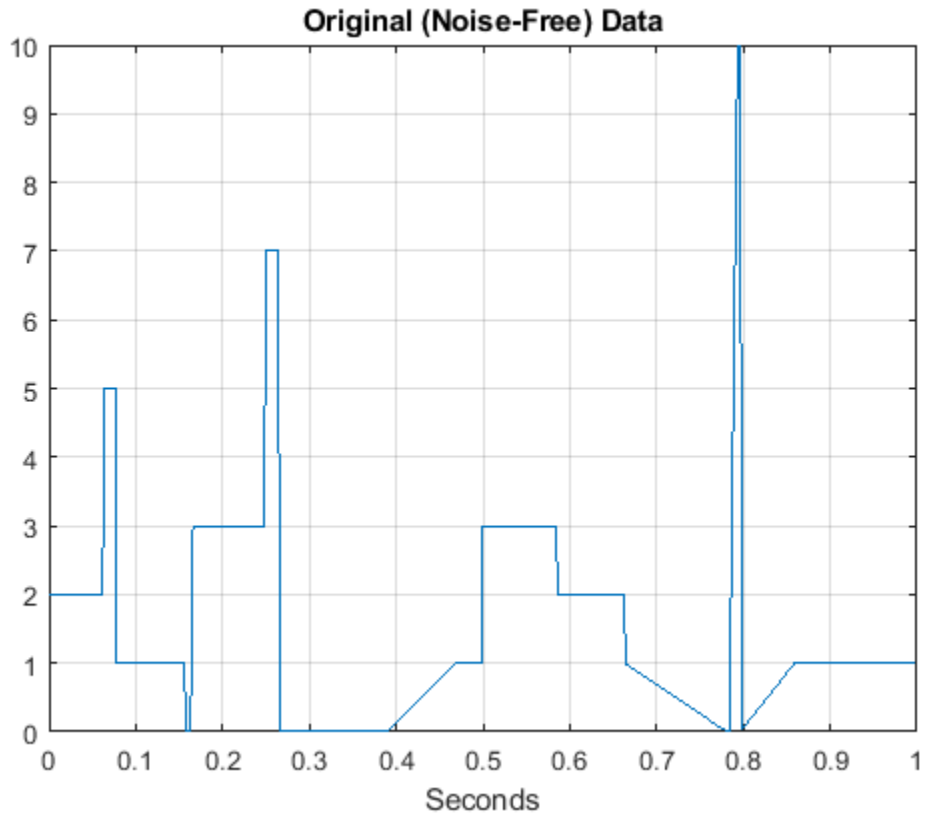
In real-world applications, abrupt changes in the data often signify important events. When you denoise such data, it is important not to smooth out the transients. Wavelets typically excel at denoising such data because the wavelets stretch to match longer duration smoothly varying features and shrink to match transients. The MLPT has similar characteristics, but it naturally handles nonuniformly sampled data. Denoise the data using the MLPT.

```
xden = mlptdenoise(y,T,3);
hline = plot(T,[f xden]);
grid on
hline(1).LineWidth = 2;
title('MLPT Denoising')
```
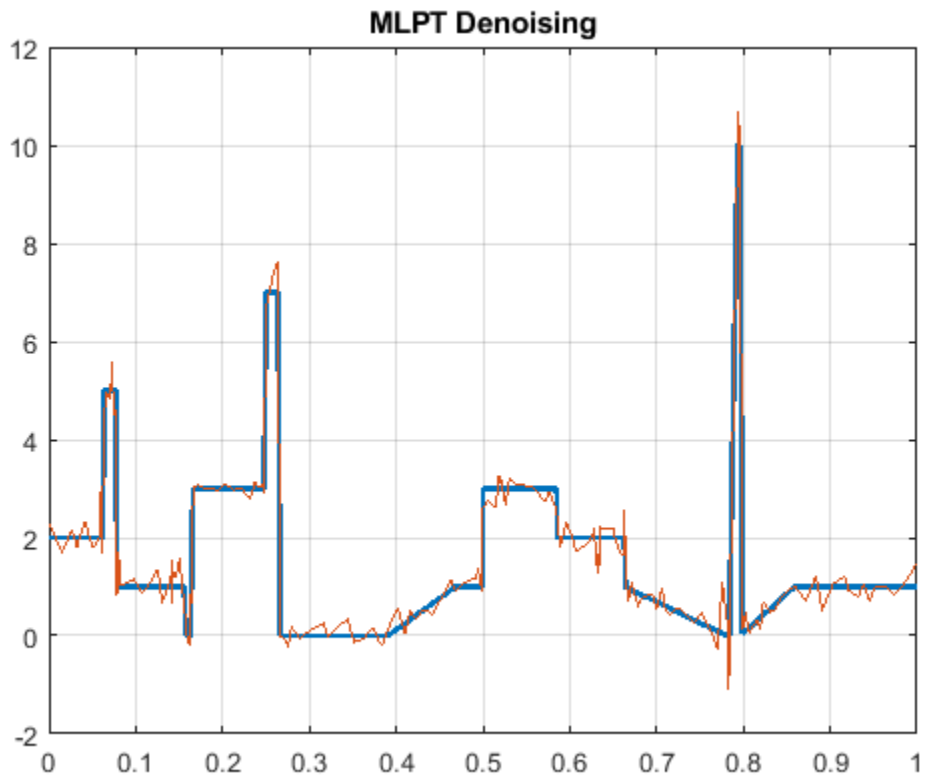
MLPT Denoising

The MLPT does a good job of denoising the data. The smoothly varying features are well represented and the transients are preserved. In this synthetic data set, you can actually measure the signal-to-noise ratio (SNR) of the denoised version.

```
SNR = 20*log10(norm(f,2)/norm(xden-f,2));
title(['SNR  ' num2str(SNR,'%2.2f') 'dB'])
```
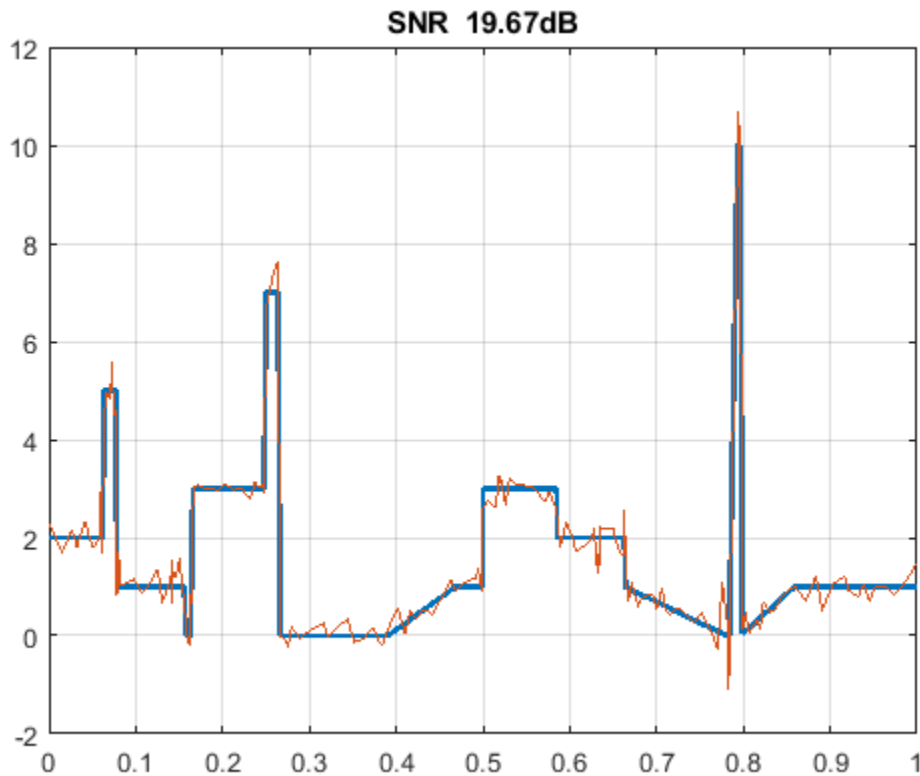
The SNR is almost 20 dB. Of course, you can ignore the fact that the data are nonuniformly sampled and treat the samples as uniformly sampled. Denoise the previous data set using the DWT.

```
xd = wdenoise(y,3,'Wavelet','bior2.2','DenoisingMethod','SURE','NoiseEstimate','LevelDependent')
SNR = 20*log10(norm(f,2)/norm(xd-f,2));
hline = plot(T,[f xd]);
title(['SNR  ' num2str(SNR,'%2.2f') 'dB'])
grid on
hline(1).LineWidth = 2;
```
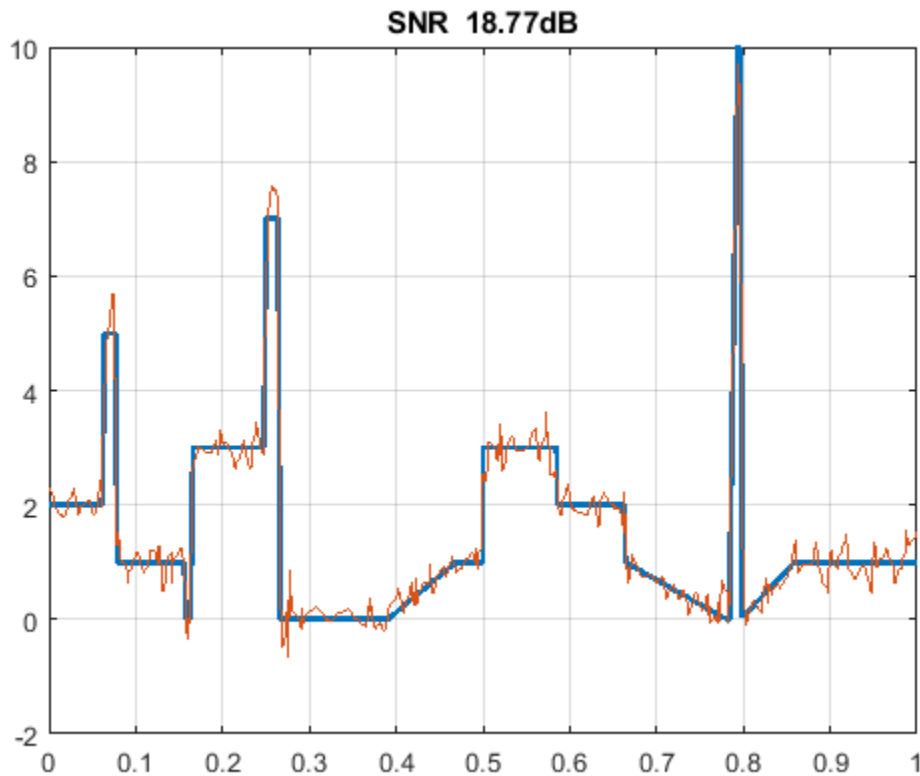
Here the DWT does a good job of denoising the data, but it is outperformed by the MLPT, which explicitly takes the nonuniform sampling instants into account.

Compare the wavelet and MLPT denoising results against the Savitzky-Golay method, which implements a local polynomial approximation of the data. The variant of Savitzky-Golay implemented in `smoothdata` handles uniformly and nonuniformly sampled data.

```
xdsg = smoothdata(y,'sgolay','degree',4,'samplepoints',T);
SNR = 20*log10(norm(f,2)/norm(xdsg-f,2));
hline = plot(T,[f xdsg]);
title(['SNR  ' num2str(SNR,'%2.2f') ' dB'])
grid on
hline(1).LineWidth = 2;
```
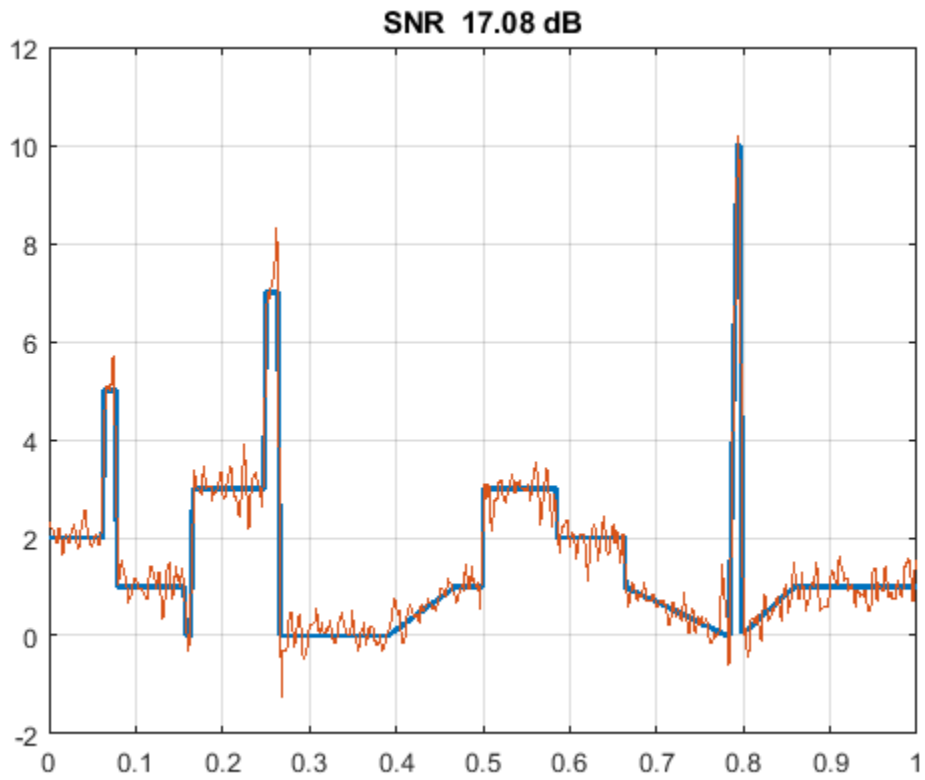
Both the classic DWT and the MLPT outperform Savitzky-Golay on this data.

Consider another nonuniformly sampled synthetic dataset.

```
load nonuniformheavisine
plot(t,x)
grid on
```
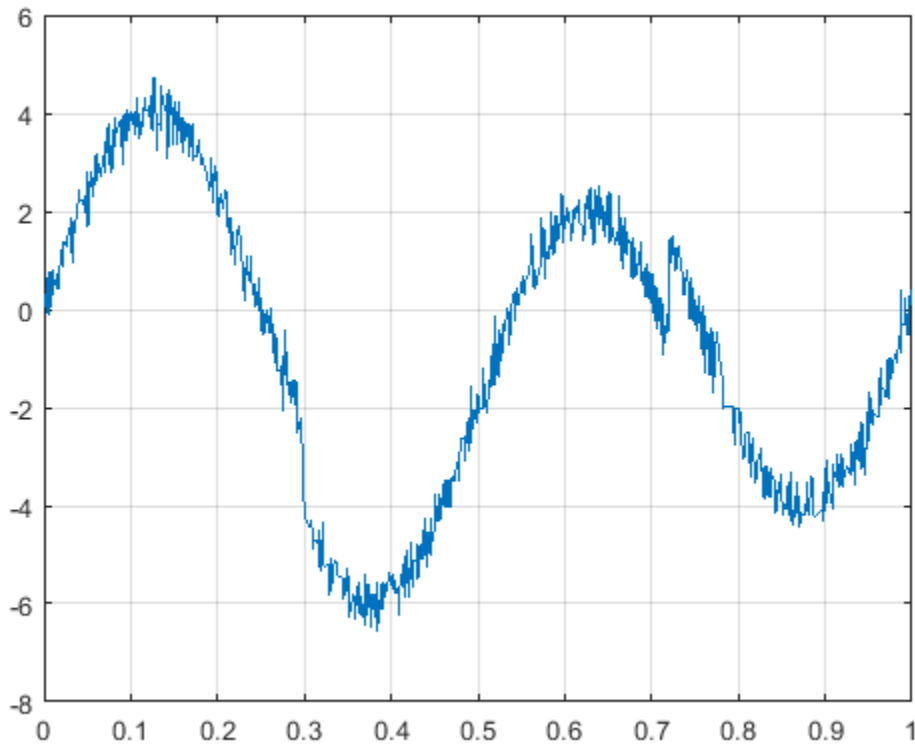
This data is generally smoother than the previous example with the notable exception of two transients at 0.3 and 0.7 seconds. Data like these present a challenge for methods like Savitzky-Golay because a low-order polynomial is required to fit the smoothly oscillating data, while a higher-order polynomial is required to approximate the jumps.

Denoise the data using the MLPT and measure the SNR. Return both the original MLPT coefficients and the denoised coefficients.

```
[xden,t,wthr,w] = mlptdenoise(x,t,3,'denoisingmethod','SURE');
plot(t,[f xden])
grid on
SNR = 20*log10(norm(f,2)/norm(xden-f,2));
title({'MLPT Denoising'; ['SNR  ' num2str(SNR,'%2.2f') ' dB']})
```
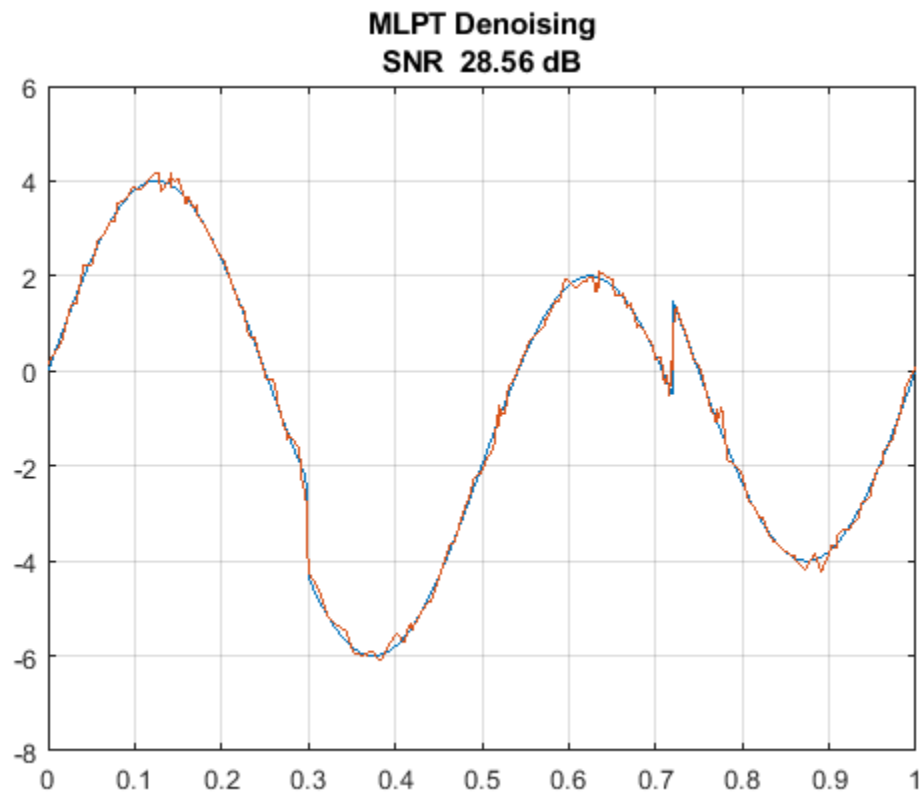
Plot the original and denoised coefficients.

```
figure
subplot(2,1,1)
stem(w,'ShowBaseLine','off','Marker','None')
title('Original MLPT Coefficients')
subplot(2,1,2)
stem(wthr,'ShowBaseLine','off','Marker','None')
title('Denoised MLPT Coefficients')
```
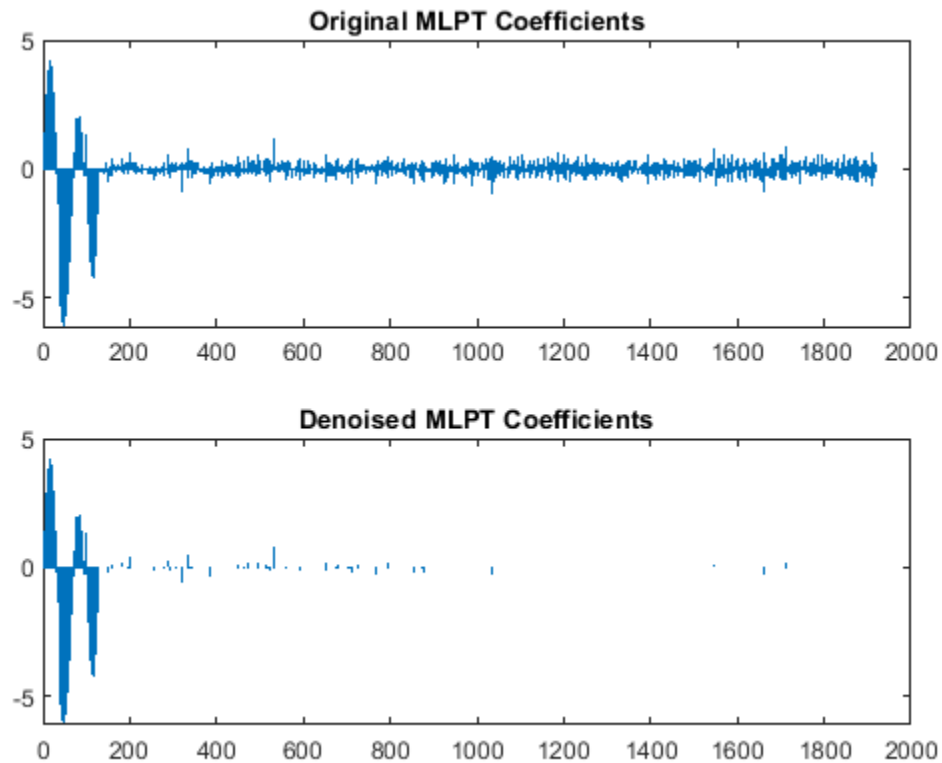
Compare the MLPT result with the Savitzky-Golay method.

```
xdsg = smoothdata(x,'sgolay','degree',4,'samplepoints',t);
figure
plot(t,[f xdsg])
grid on
SNR = 20*log10(norm(f,2)/norm(xdsg-f,2));
title({'Savitzky-Golay Denoising'; ['SNR  ' num2str(SNR,'%2.2f') ' dB']})
```

Savitzky-Golay Denoising
SNR 23.75 dB
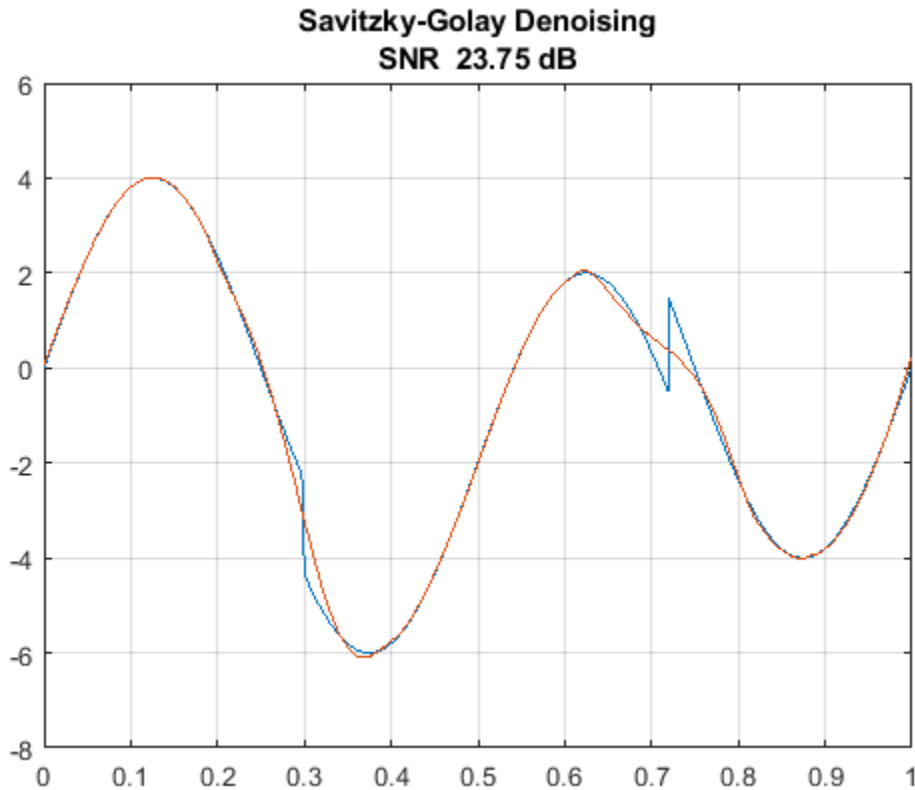
Here the MLPT method significantly outperforms Savitzky-Golay. This is especially evident in the inability of the Savitzky-Golay method to capture the transients near 0.3 and 0.7 seconds.

Nonparametric Regression

In many applications, the goal is to estimate some unknown response function as one or more predictor variables vary. When the exact shape of the response function is unknown, this is an example of *nonparametric regression*.

A principal objective in these cases is often to obtain a smooth estimate assuming that the unknown response function changes smoothly with changes in the predictor.

As an example of nonparametric regression with nonuniformly sampled data, consider the following data consisting of G-force measurements made on the motorcycle helmet of a crash-test dummy under crash conditions.

```
load motorcycledata
plot(times,gmeasurements)
grid on
xlabel('Time')
ylabel('G-force')
```

12-51
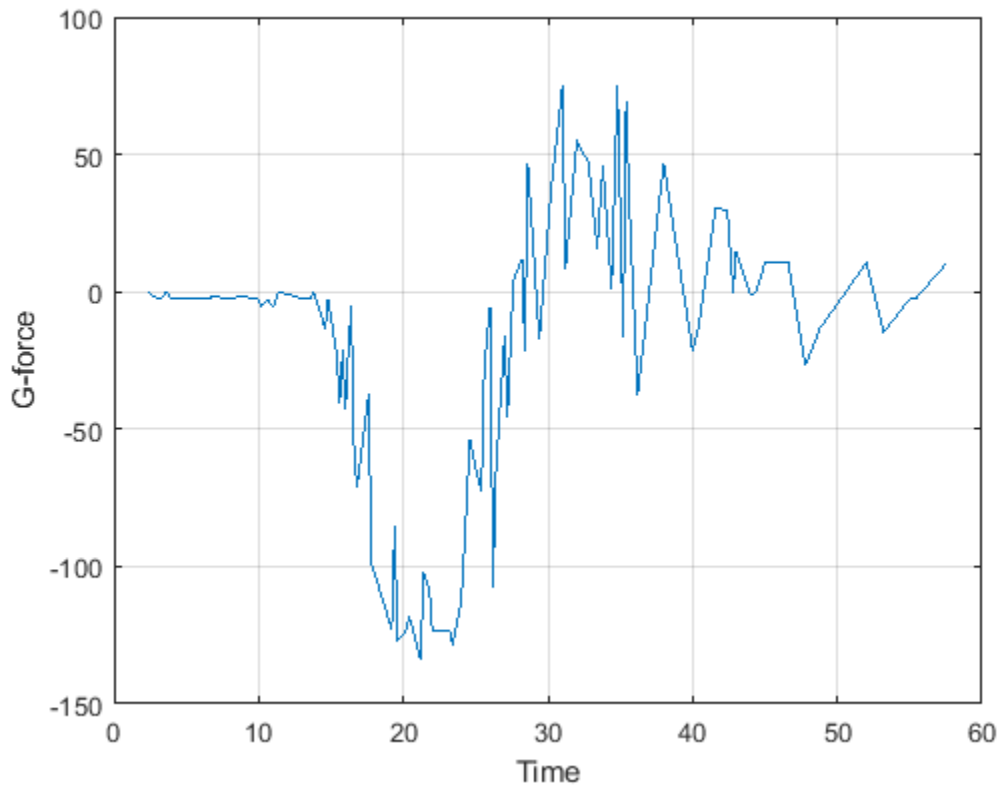
If you plot the difference of the time data, you see that the data is sampled nonuniformly.

```
plot(diff(times))
title('First Difference of Time Data')
grid on
```
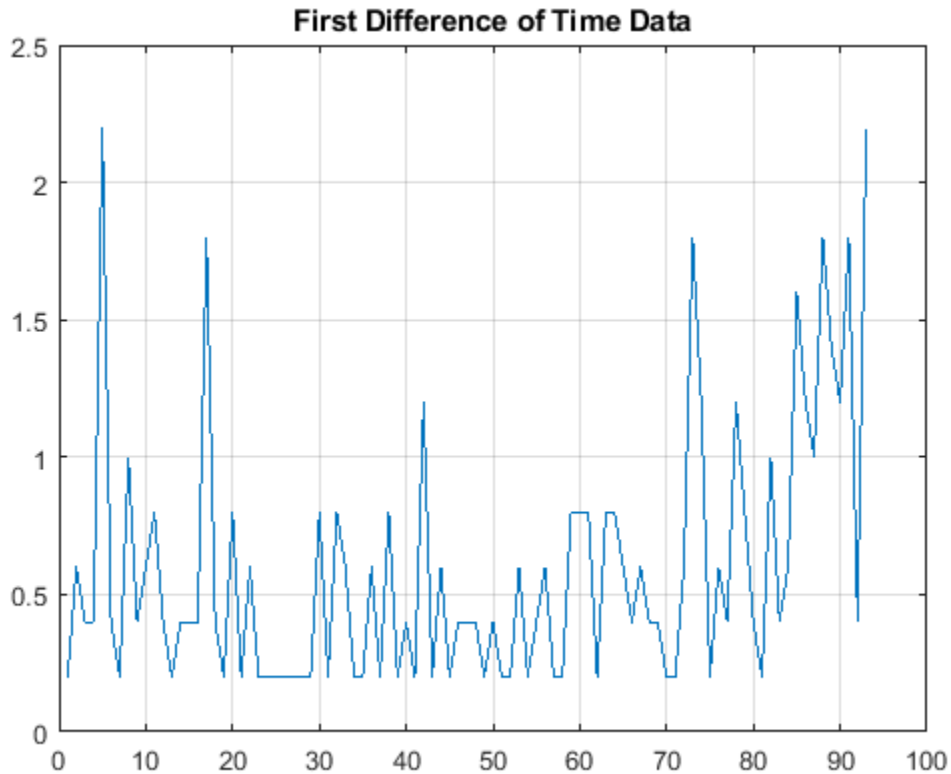
The data are noisy but there appears to be a clear overall trend in the data. Specifically, there is an initial negative G-force experienced which then begins to turn positive over time. The positive rebound continues past 0 and then recovers to baseline. You can use `mlptrecon` to obtain a nonparametric regression for this data.

```
[w,t,nj,scalingmoments] = mlpt(gmeasurements,times,'dualmoments',4,...
    'primalmoments',4,'prefilter','none');
a4 = mlptrecon('a',w,t,nj,scalingmoments,4,'dualmoments',4);
hline = plot(times,[gmeasurements a4]);
grid on
hline(2).LineWidth = 2;
legend('Original Data','Smooth Fit')
xlabel('Time')
ylabel('G-force')
title('G-Force Measurements')
```
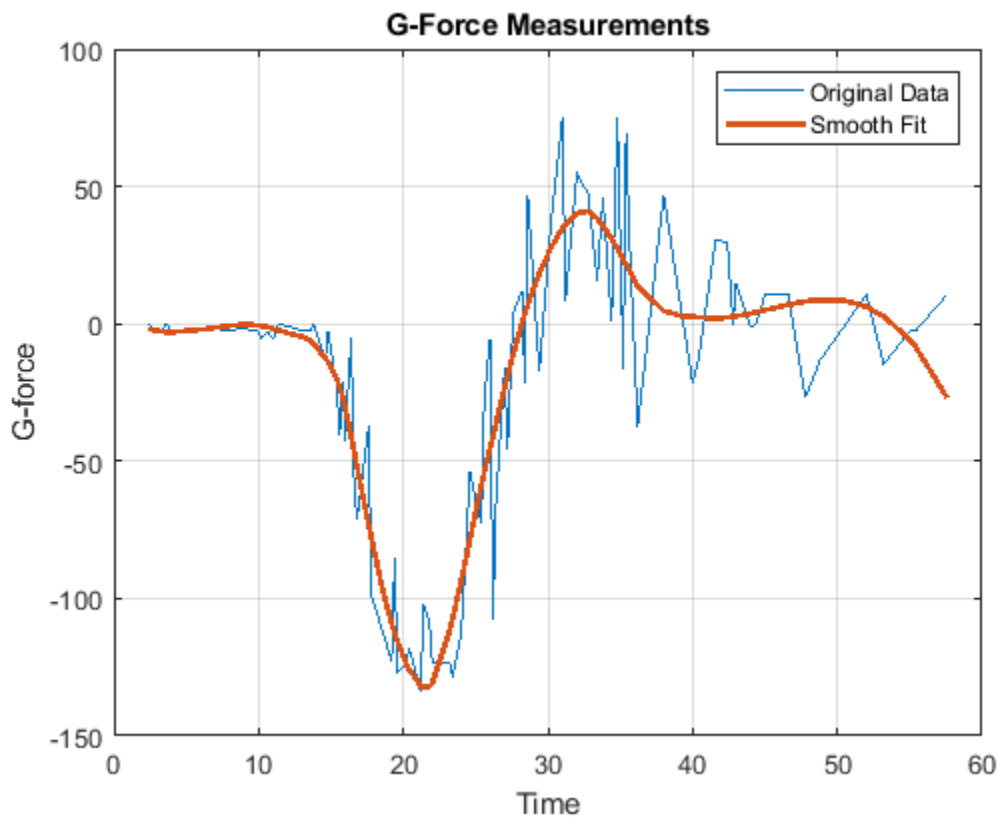
The MLPT approximation at level 4 provides a nice smooth fit to the data that allows you to capture the essential nature of the response without the effect of noise.

Summary

This example demonstrated the multiscale local polynomial transform (MLPT), a lifting scheme which is amenable to nonuniformly sampled data. The MLPT is useful for denoising or nonparametric regression in cases of missing or nonuniformly sampled data. Minimally, it is a useful benchmark for comparison against wavelet denoising techniques designed to work on uniformly sampled data. In other cases, it outperforms conventional wavelet denoising by explicitly taking the nonuniform sampling grid into consideration.

References

Jansen, M. "Multiscale Local Polynomial Smoothing in a Lifted Pyramid for Non-Equispaced Data". IEEE Transactions on Signal Processing. Vol. 61, Number 3, 2013, pp. 545-555.

Jansen, M., and Patrick Oonincx. "Second Generation Wavelets and Applications." London: Springer, 2005.

Two-Dimensional True Compression

Starting from a given image, the goal of true compression is to minimize the number of bits needed to represent it, while storing information of acceptable quality. Wavelets contribute to effective solutions for this problem. The complete chain of compression includes iterative phases of quantization, coding, and decoding, in addition to the wavelet processing itself.

The purpose of this example is to show how to decompose, compress, and decompress a grayscale or truecolor image using various compression methods. To illustrate these capabilities, we consider a grayscale image of a mask and a truecolor image of peppers.

Compression by Global Thresholding and Huffman Encoding

First we load and display the *mask* grayscale image.

```
load mask
image(X)
axis square
colormap(pink(255))
title('Original Image: mask')
```
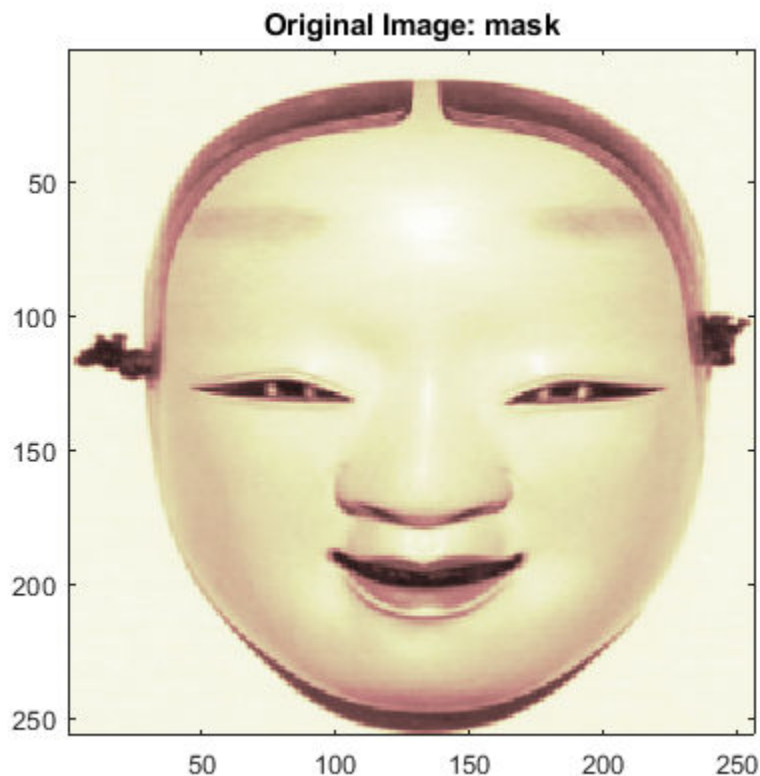

A measure of achieved compression is given by the compression ratio (CR) and the Bit-Per-Pixel (BPP) ratio. CR and BPP represent equivalent information. CR indicates that the compressed image is stored using CR % of the initial storage size while BPP is the number of bits used to store one pixel of the image. For a grayscale image the initial BPP is 8. 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 low compression ratio and a good perceptual result.

We begin with a simple method of 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 BPP is set to 0.5 and the compressed image is 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 = 6.7200
```

```
BPP = 0.5376
```

The achieved Bit-Per-Pixel ratio is actually about 0.53 (closed to the desired one of 0.5) for a compression ratio of 6.7%.

Uncompression

Now we decompress 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))
```


Original Image: mask

```
subplot(1,2,1); image(X);
axis square;
```
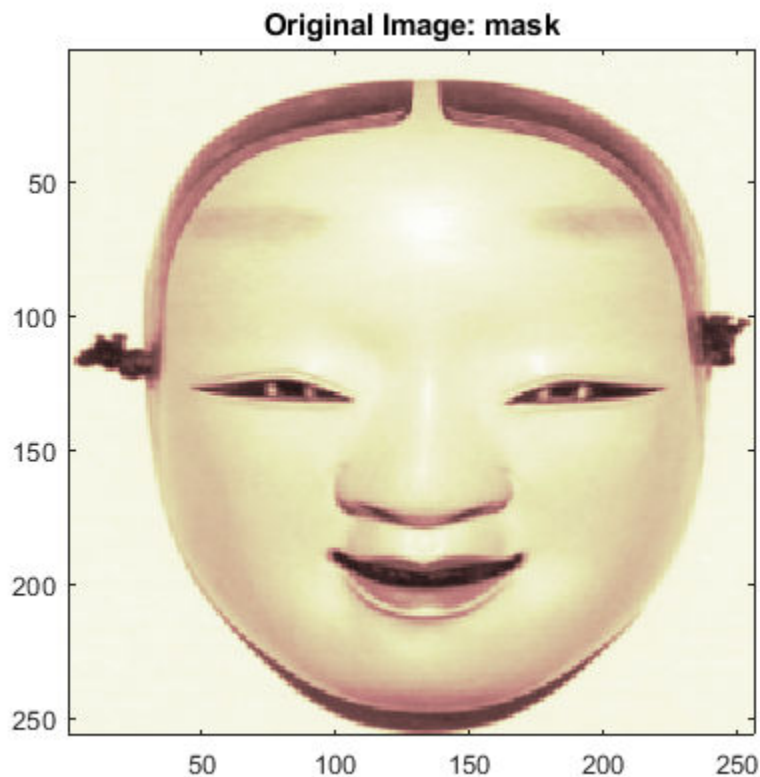
```
title('Original Image')
subplot(1,2,2); image(Xc);
axis square;
title('Compressed Image')
xlabel({['Compression Ratio: ' num2str(CR,'%1.2f %%')], ...
        ['BPP: ' num2str(BPP,'%3.2f')]})
```
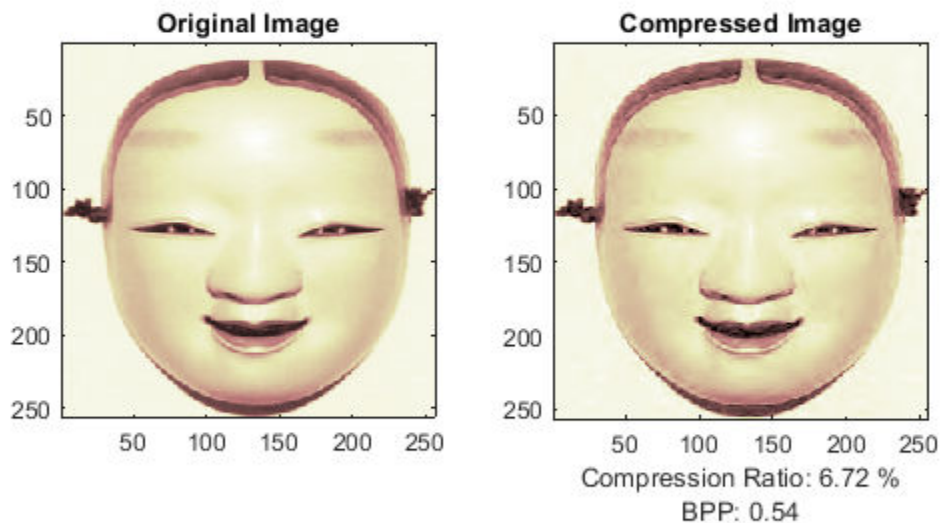

The result is satisfactory, but a better compromise between compression ratio and visual quality can be obtained using more sophisticated true compression methods, which involve tighter thresholding and quantization steps.

Compression by Progressive Methods

We now illustrate the use of progressive methods of compression, starting with the EZW algorithm using the Haar wavelet. The key parameter is the number of loops; increasing it leads to better recovery, but a 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','haar');
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')]})
```
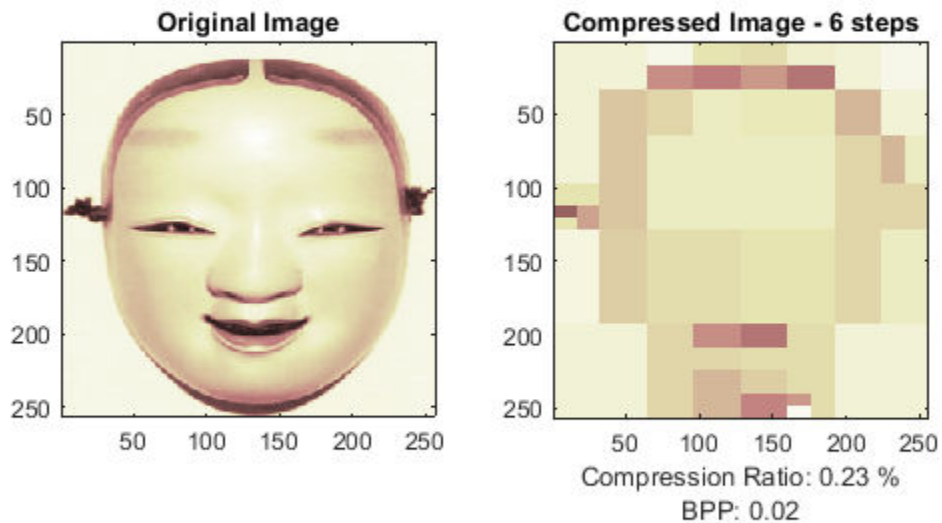

Here, using only 6 steps produces a very coarse decompressed image. Now we examine a slightly better result using 9 steps and finally a satisfactory result using 12 steps.

```
[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')]})
```
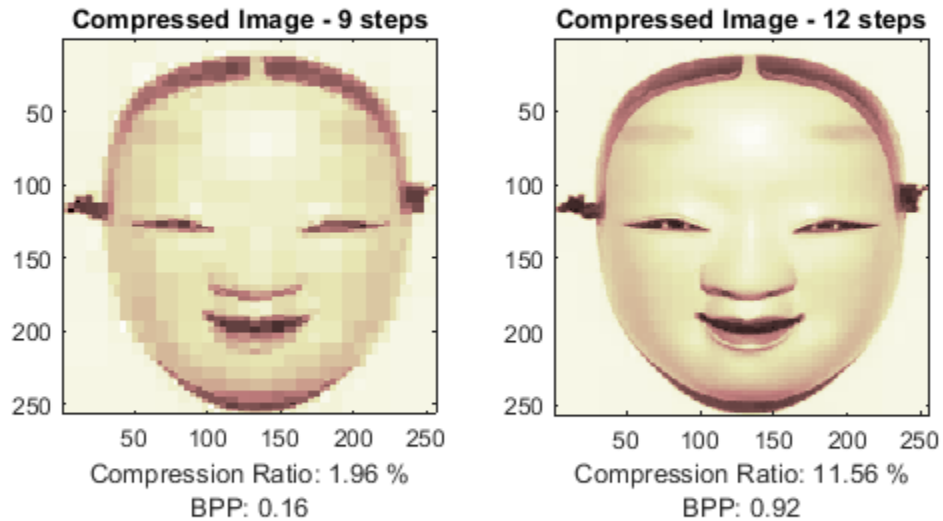
The final BPP ratio is approximately 0.92 when using 12 steps.

Now we try to improve the results by using the wavelet *bior4.4* instead of *haar* and looking at loops of 12 and 11 steps.

```matlab
[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')
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')
xlabel({['Compression Ratio: ' num2str(CR,'%1.2f %%')], ...
        ['BPP: ' num2str(BPP,'%3.2f')]})
```
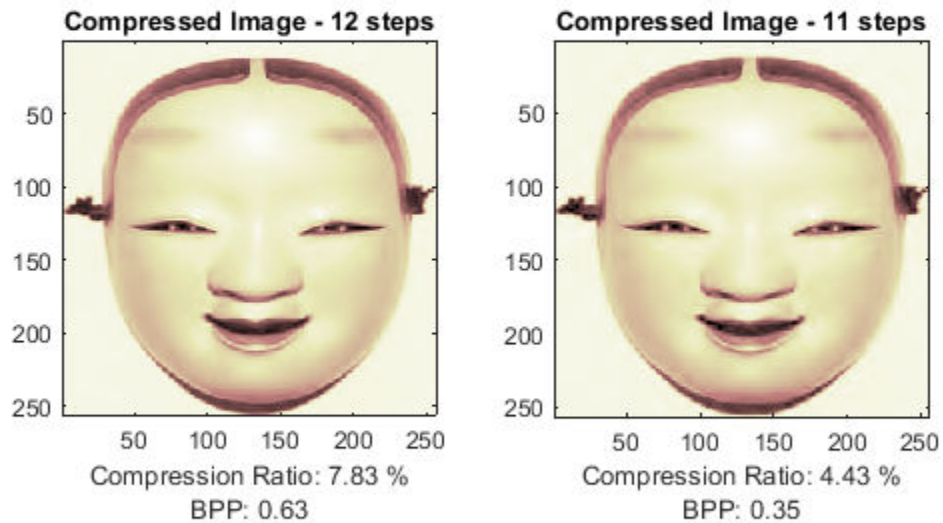
For the eleventh loop, we see that the result can be considered satisfactory, and the obtained BPP ratio is approximately 0.35. By using a more recent method, SPIHT (Set Partitioning in Hierarchical Trees), the BPP can be improved further.

```
[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')
xlabel({['Compression Ratio: ' num2str(CR,'%1.2f %%')], ...
        ['BPP: ' num2str(BPP,'%3.2f')]})
```

Original Image

Compressed Image - 12 steps

Compression Ratio: 2.86 %
BPP: 0.23
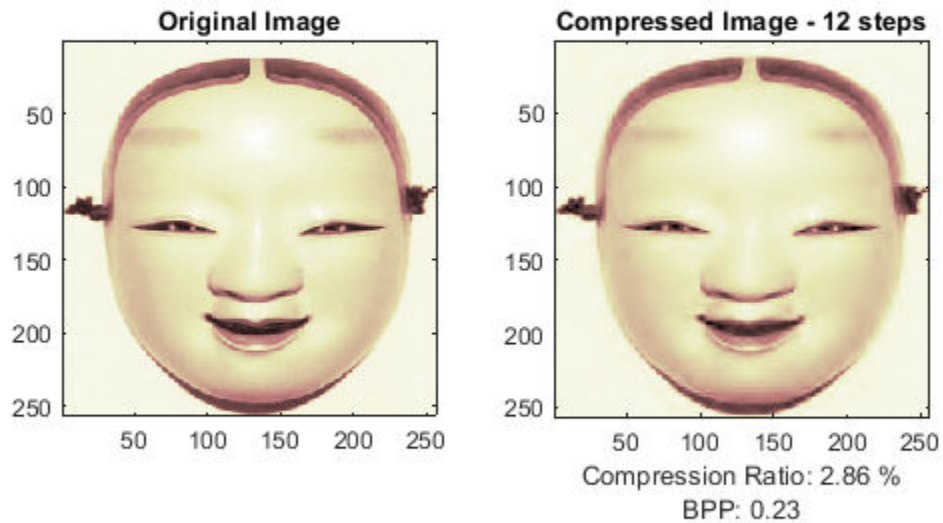
The final compression ratio (2.8%) and Bit-Per-Pixel ratio (0.23) are very satisfactory. Recall that the CR means that the compressed image is stored using only 2.8% of the initial storage size.

Handling Truecolor Images

Finally, we illustrate how to compress the *wpeppers.jpg* truecolor image. Truecolor images can be compressed using the same scheme as 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');
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')
xlabel({['Compression Ratio: ' num2str(CR,'%1.2f %%')], ...
        ['BPP: ' num2str(BPP,'%3.2f')]})
```

12-61
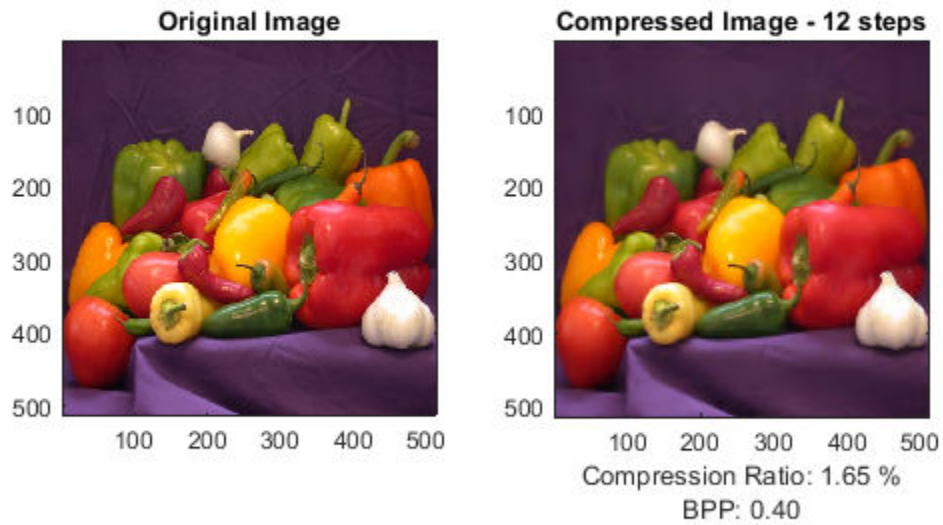
```
delete('wpeppers.wtc')
```

The compression ratio (1.65%) and the Bit-Per-Pixel ratio (0.4) are very satisfactory while maintaining a good visual perception.

More about True Compression of Images

For more information about True Compression of images, including some theory and examples, see the following reference:

Misiti, M., Y. Misiti, G. Oppenheim, J.-M. Poggi (2007), "Wavelets and their applications", ISTE DSP Series.

Featured Examples — Machine Learning and Deep Learning

Signal Classification Using Wavelet-Based Features and Support Vector Machines

This example shows how to classify human electrocardiogram (ECG) signals using wavelet-based feature extraction and a support vector machine (SVM) classifier. The problem of signal classification is simplified by transforming the raw ECG signals into a much smaller set of features that serve in aggregate to differentiate different classes. You must have Wavelet Toolbox™, Signal Processing Toolbox™, and Statistics and Machine Learning Toolbox™ to run this example. The data used in this example are publicly available from PhysioNet.

Data Description

This example uses ECG data obtained from three groups, or classes, of people: persons with cardiac arrhythmia, persons with congestive heart failure, and persons with normal sinus rhythms. The example uses 162 ECG recordings from three PhysioNet databases: MIT-BIH Arrhythmia Database [3] [7], MIT-BIH Normal Sinus Rhythm Database [3], and The BIDMC Congestive Heart Failure Database [1][3]. In total, there are 96 recordings from persons with arrhythmia, 30 recordings from persons with congestive heart failure, and 36 recordings from persons with normal sinus rhythms. The goal is to train a classifier to distinguish between arrhythmia (ARR), congestive heart failure (CHF), and normal sinus rhythm (NSR).

Download Data

The first step is to download the data from the GitHub repository. To download the data, click `Clone or download` and select `Download ZIP`. Save the file `physionet_ECG_data-master.zip` in a folder where you have write permission. The instructions for this example assume you have downloaded the file to your temporary directory, (`tempdir` in MATLAB). Modify the subsequent instructions for unzipping and loading the data if you choose to download the data in folder different from `tempdir`. If you are familiar with Git, you can download the latest version of the tools (git) and obtain the data from a system command prompt using `git clone https://github.com/ mathworks/physionet_ECG_data/` .

The file `physionet_ECG_data-master.zip` contains

- ECGData.zip
- README.md

and ECGData.zip contains

- ECGData.mat
- Modified_physionet_data.txt
- License.txt.

ECGData.mat holds the data used in this example. The .txt file, Modified_physionet_data.txt, is required by PhysioNet's copying policy and provides the source attributions for the data as well as a description of the pre-processing steps applied to each ECG recording.

Load Files

If you followed the download instructions in the previous section, enter the following commands to unzip the two archive files.

```
unzip(fullfile(tempdir,'physionet_ECG_data-master.zip'),tempdir)
unzip(fullfile(tempdir,'physionet_ECG_data-master','ECGData.zip'),...
    fullfile(tempdir,'ECGData'))
```

After you unzip the ECGData.zip file, load the data into MATLAB.

```
load(fullfile(tempdir,'ECGData','ECGData.mat'))
```

ECGData is a structure array with two fields: Data and Labels. Data is a 162-by-65536 matrix where each row is an ECG recording sampled at 128 hertz. Labels is a 162-by-1 cell array of diagnostic labels, one for each row of Data. The three diagnostic categories are: 'ARR' (arrhythmia), 'CHF' (congestive heart failure), and 'NSR' (normal sinus rhythm).

Create Training and Test Data

Randomly split the data into two sets - training and test data sets. The helper function helperRandomSplit performs the random split. helperRandomSplit accepts the desired split percentage for the training data and ECGData. The helperRandomSplit function outputs two data sets along with a set of labels for each. Each row of trainData and testData is an ECG signal. Each element of trainLabels and testLabels contains the class label for the corresponding row of the data matrices. In this example, we randomly assign 70% percent of the data in each class to the training set. The remaining 30% is held out for testing (prediction) and are assigned to the test set.

```
percent_train = 70;
[trainData,testData,trainLabels,testLabels] = ...
    helperRandomSplit(percent_train,ECGData);
```

There are 113 records in the trainData set and 49 records in testData. By design the training data contains 69.75% (113/162) of the data. Recall that the ARR class represents 59.26% of the data (96/162), the CHF class represents 18.52% (30/162), and the NSR class represents 22.22% (36/162). Examine the percentage of each class in the training and test sets. The percentages in each are consistent with the overall class percentages in the data set.

```
Ctrain = countcats(categorical(trainLabels))./numel(trainLabels).*100
Ctest = countcats(categorical(testLabels))./numel(testLabels).*100


Ctrain =

   59.2920
   18.5841
   22.1239


Ctest =

   59.1837
   18.3673
   22.4490
```

Plot Samples

Plot the first few thousand samples of four randomly selected records from ECGData. The helper function helperPlotRandomRecords does this. helperPlotRandomRecords accepts ECGData and a random seed as input. The initial seed is set at 14 so that at least one record from each class is

plotted. You can execute `helperPlotRandomRecords` with `ECGData` as the only input argument as many times as you wish to get a sense of the variety of ECG waveforms associated with each class. You can find the source code for this helper function in the Supporting Functions section at the end of this example.

`helperPlotRandomRecords(ECGData,14)`

Feature Extraction

Extract the features used in the signal classification for each signal. This example uses the following features extracted on 8 blocks of each signal approximately one minute in duration (8192 samples):

- Autoregressive model (AR) coefficients of order 4 [8].
- Shannon entropy (SE) values for the maximal overlap discrete wavelet packet transform (MODPWT) at level 4 [5].
- Multifractal wavelet leader estimates of the second cumulant of the scaling exponents and the range of Holder exponents, or singularity spectrum [4].

Additionally, multiscale wavelet variance estimates are extracted for each signal over the entire data length [6]. An unbiased estimate of the wavelet variance is used. This requires that only levels with at least one wavelet coefficient unaffected by boundary conditions are used in the variance estimates. For a signal length of 2^16 (65,536) and the 'db2' wavelet this results in 14 levels.

These features were selected based on published research demonstrating their effectiveness in classifying ECG waveforms. This is not intended to be an exhaustive or optimized list of features.

The AR coefficients for each window are estimated using the Burg method, `arburg`. In [8], the authors used model order selection methods to determine that an AR(4) model provided the best fit for ECG waveforms in a similar classification problem. In [5], an information theoretic measure, the Shannon entropy, was computed on the terminal nodes of a wavelet packet tree and used with a random forest classifier. Here we use the nondecimated wavelet packet transform, `modwpt`, down to level 4.

The definition of the Shannon entropy for the undecimated wavelet packet transform following [5] is given by: $SE_j = -\sum_{k=1}^{N} p_{j,k} * \log p_{j,k}$ where N is the number of the corresponding coefficients in the j-th node and $p_{j,k}$ are the normalized squares of the wavelet packet coefficients in the j-th terminal node.

Two fractal measures estimated by wavelet methods are used as features. Following [4], we use the width of the singularity spectrum obtained from `dwtleader` as a measure of the multifractal nature of the ECG signal. We also use the second cumulant of the scaling exponents. The scaling exponents are scale-based exponents describing power-law behavior in the signal at different resolutions. The second cumulant broadly represents the departure of the scaling exponents from linearity.

The wavelet variance for the entire signal is obtained using `modwtvar`. Wavelet variance measures variability in a signal by scale, or equivalently variability in a signal over octave-band frequency intervals.

In total there are 190 features: 32 AR features (4 coefficients per block), 128 Shannon entropy values (16 values per block), 16 fractal estimates (2 per block), and 14 wavelet variance estimates.

The `helperExtractFeatures` function computes these features and concatenates them into a feature vector for each signal. You can find the source code for this helper function in the Supporting Functions section at the end of this example.

```
timeWindow = 8192;
ARorder = 4;
MODWPTlevel = 4;
[trainFeatures,testFeatures,featureindices] = ...
    helperExtractFeatures(trainData,testData,timeWindow,ARorder,MODWPTlevel);
```

`trainFeatures` and `testFeatures` are 113-by-190 and 49-by-190 matrices, respectively. Each row of these matrices is a feature vector for the corresponding ECG data in `trainData` and `testData`, respectively. In creating feature vectors, the data is reduced from 65536 samples to 190 element vectors. This is a significant reduction in data, but the goal is not just a reduction in data. The goal is to reduce the data to a much smaller set of features which captures the difference between the classes so that a classifier can accurately separate the signals. The indices for the features, which make up both `trainFeatures` and `testFeatures` are contained in the structure array, `featureindices`. You can use these indices to explore features by group. As an example, examine the range of Holder exponents in the singularity spectra for the first time window. Plot the data for the entire data set.

```
allFeatures = [trainFeatures;testFeatures];
allLabels = [trainLabels;testLabels];
figure
boxplot(allFeatures(:,featureindices.HRfeatures(1)),allLabels,'notch','on')
ylabel('Holder Exponent Range')
title('Range of Singularity Spectrum by Group (First Time Window)')
grid on
```

Range of Singularity Spectrum by Group (First Time Window)
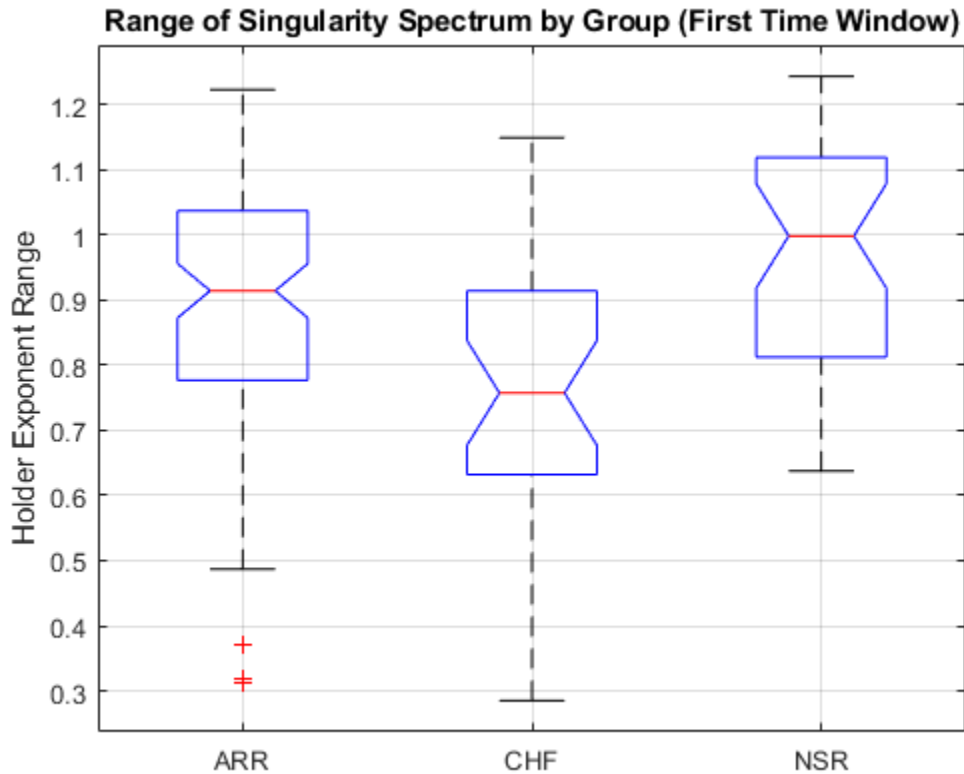
You can perform a one-way analysis of variance on this feature and confirm what appears in the boxplot, namely that the ARR and NSR groups have a significantly larger range than the CHF group.

```
[p,anovatab,st] = anova1(allFeatures(:,featureindices.HRfeatures(1)),...
    allLabels);
c = multcompare(st,'display','off')
```

```
c =

    1.0000    2.0000    0.0176    0.1144    0.2112    0.0155
    1.0000    3.0000   -0.1591   -0.0687    0.0218    0.1764
    2.0000    3.0000   -0.2975   -0.1831   -0.0687    0.0005
```

ANOVA Table

Source	SS	df	MS	F	Prob>F
Groups	0.55664	2	0.27832	7.14	0.0011
Error	6.20009	159	0.03899		
Total	6.75673	161			
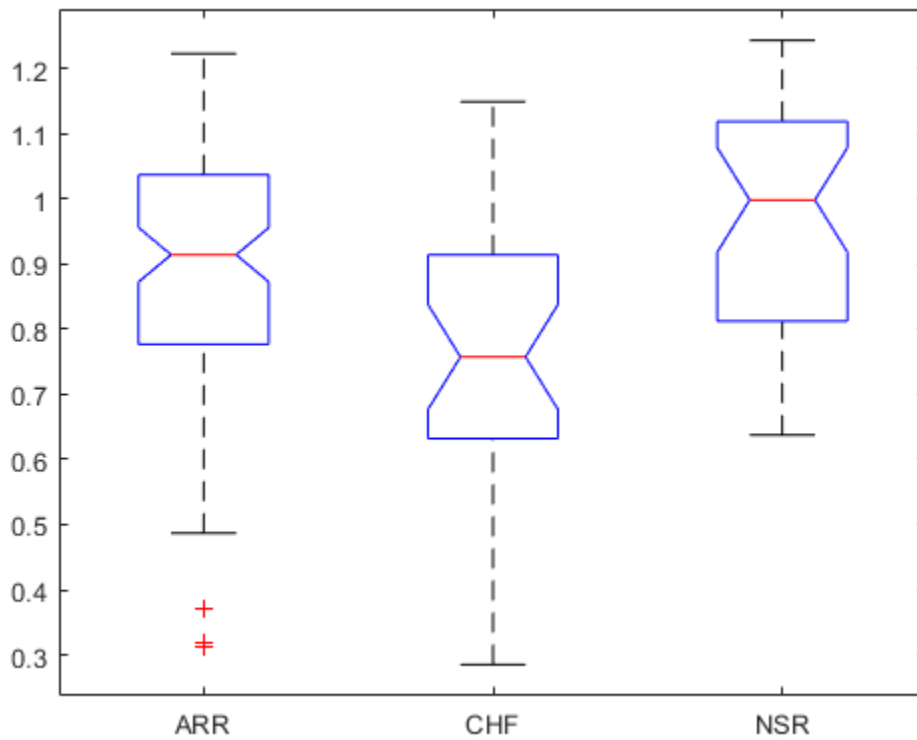
As an additional example, consider the difference in variance in the second-lowest frequency (second-largest scale) wavelet subband for the three groups.

```
boxplot(allFeatures(:,featureindices.WVARfeatures(end-1)),allLabels,'notch','on')
ylabel('Wavelet Variance')
title('Wavelet Variance by Group')
grid on
```

Wavelet Variance by Group
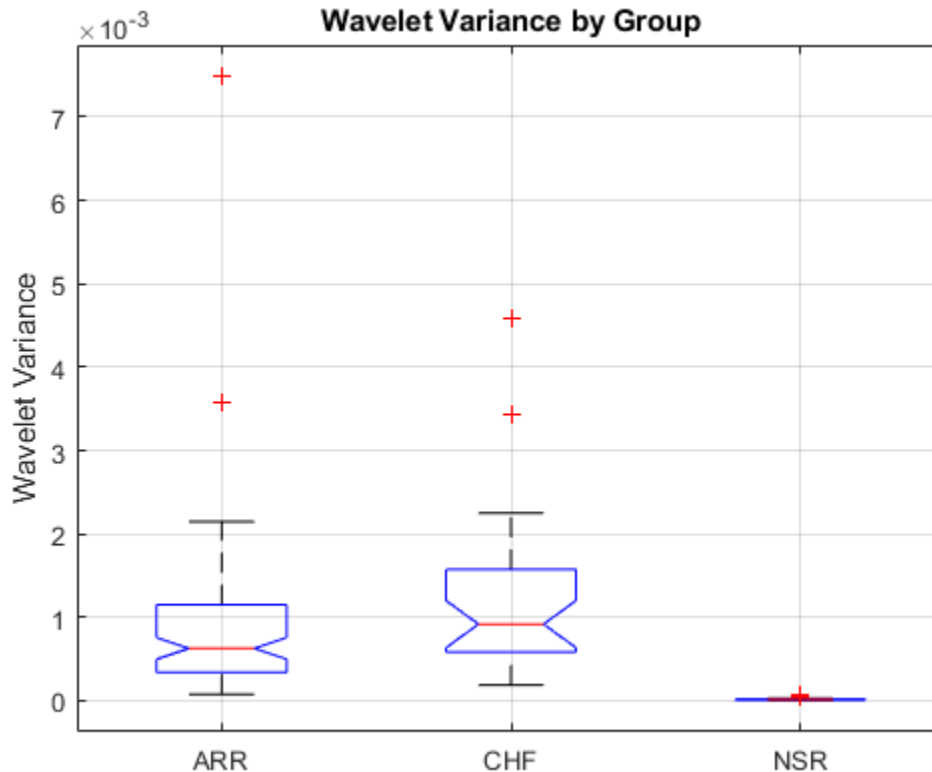
If you perform an analysis of variance on this feature, you find that the NSR group has significantly lower variance in this wavelet subband than the ARR and CHF groups. These examples are just intended to illustrate how individual features serve to separate the classes. While one feature alone is not sufficient, the goal is to obtain a rich enough feature set to enable a classifier to separate all three classes.

Signal Classification

Now that the data has been reduced to a feature vector for each signal, the next step is to use these feature vectors for classifying the ECG signals. You can use the Classification Learner app to quickly evaluate a large number of classifiers. In this example, a multi-class SVM with a quadratic kernel is used. Two analyses are performed. First we use the entire dataset (training and testing sets) and estimate the misclassification rate and confusion matrix using 5-fold cross-validation.

```
features = [trainFeatures; testFeatures];
rng(1)
template = templateSVM(...
    'KernelFunction','polynomial',...
    'PolynomialOrder',2,...
    'KernelScale','auto',...
    'BoxConstraint',1,...
    'Standardize',true);
model = fitcecoc(...
    features,...
    [trainLabels;testLabels],...
    'Learners',template,...
    'Coding','onevsone',...
```

```
    'ClassNames',{'ARR','CHF','NSR'});
kfoldmodel = crossval(model,'KFold',5);
classLabels = kfoldPredict(kfoldmodel);
loss = kfoldLoss(kfoldmodel)*100
[confmatCV,grouporder] = confusionmat([trainLabels;testLabels],classLabels);


loss =

    8.0247
```

The 5-fold classification error is 8.02% (91.98% correct). The confusion matrix, `confmatCV`, shows which records were misclassified. `grouporder` gives the ordering of the groups. Two of the ARR group were misclassified as CHF, eight of the CHF group were misclassified as ARR and one as NSR, and two from the NSR group were misclassified as ARR.

Precision, Recall, and F1 Score

In a classification task, the precision for a class is the number of correct positive results divided by the number of positive results. In other words, of all the records that the classifier assigns a given label, what proportion actually belong to the class. Recall is defined as the number of correct labels divided by the number of labels for a given class. Specifically, of all the records belonging to a class, what proportion did our classifier label as that class. In judging the accuracy your machine learning system, you ideally want to do well on both precision and recall. For example, suppose we had a classifier that labeled every single record as ARR. Then our recall for the ARR class would be 1 (100%). All records belonging to the ARR class would be labeled ARR. However, the precision would be low. Because our classifier labeled all records as ARR, there would be 66 false positives in this case for a precision of 96/162, or 0.5926. The F1 score is the harmonic mean of precision and recall and therefore provides a single metric that summarizes the classifier performance in terms of both recall and precision. The following helper function computes the precision, recall, and F1 scores for the three classes. You can see how `helperPrecisionRecall` computes precision, recall, and the F1 score based on the confusion matrix by examining the code in the Supporting Functions section.

```
CVTable = helperPrecisionRecall(confmatCV);
```

You can display the table returned by `helperPrecisionRecall` with the following command.

```
disp(CVTable)
```

```
          Precision     Recall     F1_Score
          _____     _____     _____

    ARR    90.385       97.917         94
    CHF    91.304          70       79.245
    NSR    97.143       94.444       95.775
```

Both precision and recall are good for the ARR and NSR classes, while recall is significantly lower for the CHF class.

For the next analysis, we fit a multi-class quadratic SVM to the training data only (70%) and then use that model to make predictions on the 30% of the data held out for testing. There are 49 data records in the test set.

```
model = fitcecoc(...
    trainFeatures,...
```

```
        trainLabels,...
        'Learners',template,...
        'Coding','onevsone',...
        'ClassNames',{'ARR','CHF','NSR'});
predLabels = predict(model,testFeatures);
```

Use the following to determine the number of correct predictions and obtain the confusion matrix.

```
correctPredictions = strcmp(predLabels,testLabels);
testAccuracy = sum(correctPredictions)/length(testLabels)*100
[confmatTest,grouporder] = confusionmat(testLabels,predLabels);
```

```
testAccuracy =

    97.9592
```

The classification accuracy on the test dataset is approximately 98% and the confusion matrix shows that one CHF record was misclassified as NSR.

Similar to what was done in the cross-validation analysis, obtain precision, recall, and the F1 scores for the test set.

```
testTable = helperPrecisionRecall(confmatTest);
disp(testTable)
```

	Precision	Recall	F1_Score
ARR	100	100	100
CHF	100	88.889	94.118
NSR	91.667	100	95.652

Classification on Raw Data and Clustering

Two natural questions arise from the previous analysis. Is feature extraction necessary in order to achieve good classification results? Is a classifier necessary, or can these features separate the groups without a classifier? To address the first question, repeat the cross-validation results for the raw time series data. Note that the following is a computationally expensive step because we are applying the SVM to a 162-by-65536 matrix. If you do not wish to run this step yourself, the results are described in the next paragraph.

```
rawData = [trainData;testData];
Labels = [trainLabels;testLabels];
rng(1)
template = templateSVM(...
    'KernelFunction','polynomial', ...
    'PolynomialOrder',2, ...
    'KernelScale','auto', ...
    'BoxConstraint',1, ...
    'Standardize',true);
model = fitcecoc(...
    rawData,...
    [trainLabels;testLabels],...
    'Learners',template,...
    'Coding','onevsone',...
```

```
    'ClassNames',{'ARR','CHF','NSR'});
kfoldmodel = crossval(model,'KFold',5);
classLabels = kfoldPredict(kfoldmodel);
loss = kfoldLoss(kfoldmodel)*100
[confmatCVraw,grouporder] = confusionmat([trainLabels;testLabels],classLabels);
rawTable = helperPrecisionRecall(confmatCVraw);
disp(rawTable)
```

```
loss =

   33.3333
```

	Precision	Recall	F1_Score
ARR	64	100	78.049
CHF	100	13.333	23.529
NSR	100	22.222	36.364

The misclassification rate for the raw time series data is 33.3%. Repeating the precision, recall, and F1 score analysis reveals very poor F1 scores for both the CHF (23.52) and NSR groups (36.36). Obtain the magnitude discrete Fourier transform (DFT) coefficients for each signal to perform the analysis in frequency domain. Because the data are real-valued, we can achieve some data reduction using the DFT by exploiting the fact that Fourier magnitudes are an even function.

```
rawDataDFT = abs(fft(rawData,[],2));
rawDataDFT = rawDataDFT(:,1:2^16/2+1);
rng(1)
template = templateSVM(...
    'KernelFunction','polynomial',...
    'PolynomialOrder',2,...
    'KernelScale','auto',...
    'BoxConstraint',1,...
    'Standardize',true);
model = fitcecoc(...
    rawDataDFT,...
    [trainLabels;testLabels],...
    'Learners',template,...
    'Coding','onevsone',...
    'ClassNames',{'ARR','CHF','NSR'});
kfoldmodel = crossval(model,'KFold',5);
classLabels = kfoldPredict(kfoldmodel);
loss = kfoldLoss(kfoldmodel)*100
[confmatCVDFT,grouporder] = confusionmat([trainLabels;testLabels],classLabels);
dftTable = helperPrecisionRecall(confmatCVDFT);
disp(dftTable)
```

```
loss =

   19.1358
```

	Precision	Recall	F1_Score
ARR	76.423	97.917	85.845

```
        CHF         100        26.667     42.105
        NSR      93.548        80.556     86.567
```

Using the DFT magnitudes reduces the misclassification rate to 19.13% but that is still more than twice the error rate obtained with our 190 features. These analyses demonstrate that the classifier has benefited from a careful selection of features.

To answer the question concerning the role of the classifier, attempt to cluster the data using only the feature vectors. Use k-means clustering along with the gap statistic to determine both the optimal number of clusters and cluster assignment. Allow for the possibility of 1 to 6 clusters for the data.

```
rng default
eva = evalclusters(features,'kmeans','gap','KList',[1:6]);
eva
```

```
eva =

  GapEvaluation with properties:

    NumObservations: 162
         InspectedK: [1 2 3 4 5 6]
    CriterionValues: [1.2777 1.3539 1.3644 1.3570 1.3591 1.3752]
           OptimalK: 3
```

The gap statistic indicates that the optimal number of clusters is three. However, if you look at the number of records in each of the three clusters, you see that the k-means clustering based on the feature vectors has done a poor job of separating the three diagnostic categories.

```
countcats(categorical(eva.OptimalY))
```

```
ans =

    61
    74
    27
```

Recall that there are 96 persons in the ARR class, 30 in the CHF class, and 36 in the NSR class.

Summary

This example used signal processing to extract wavelet features from ECG signals and used those features to classify ECG signals into three classes. Not only did the feature extraction result in a significant amount of data reduction, it also captured the differences between the ARR, CHF, and NSR classes as demonstrated by the cross-validation results and the performance of the SVM classifier on the test set. The example further demonstrated that applying a SVM classifier to the raw data resulted in poor performance as did clustering the feature vectors without using a classifier. Neither the classifier nor the features alone were sufficient to separate the classes. However, when feature extraction was used as a data reduction step prior to the use of a classifier, the three classes were well separated.

References

1 Baim DS, Colucci WS, Monrad ES, Smith HS, Wright RF, Lanoue A, Gauthier DF, Ransil BJ, Grossman W, Braunwald E. Survival of patients with severe congestive heart failure treated with oral milrinone. J American College of Cardiology 1986 Mar; 7(3):661-670.

2 Engin, M., 2004. ECG beat classification using neuro-fuzzy network. Pattern Recognition Letters, 25(15), pp.1715-1722.

3 Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng C-K, Stanley HE. PhysioBank, PhysioToolkit,and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. *Circulation*. Vol. 101, No. 23, 13 June 2000, pp. e215-e220. `http://circ.ahajournals.org/content/101/23/e215.full`

4 Leonarduzzi, R.F., Schlotthauer, G., and Torres. M.E. 2010. Wavelet leader based multifractal analysis of heart rate variability during myocardial ischaemia. Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE.

5 Li, T. and Zhou, M., 2016. ECG classification using wavelet packet entropy and random forests. Entropy, 18(8), p.285.

6 Maharaj, E.A. and Alonso, A.M. 2014. Discriminant analysis of multivariate time series: Application to diagnosis based on ECG signals. Computational Statistics and Data Analysis, 70, pp. 67-87.

7 Moody GB, Mark RG. The impact of the MIT-BIH Arrhythmia Database. IEEE Eng in Med and Biol 20(3):45-50 (May-June 2001). (PMID: 11446209)

8 Zhao, Q. and Zhang, L., 2005. ECG feature extraction and classification using wavelet transform and support vector machines. IEEE International Conference on Neural Networks and Brain,2, pp. 1089-1092.

Supporting Functions

helperPlotRandomRecords Plots four ECG signals randomly chosen from `ECGData`.

```
function helperPlotRandomRecords(ECGData,randomSeed)
% This function is only intended to support the XpwWaveletMLExample. It may
% change or be removed in a future release.

if nargin==2
    rng(randomSeed)
end

M = size(ECGData.Data,1);
idxsel = randperm(M,4);
for numplot = 1:4
    subplot(2,2,numplot)
    plot(ECGData.Data(idxsel(numplot),1:3000))
    ylabel('Volts')
    if numplot > 2
        xlabel('Samples')
    end
    title(ECGData.Labels{idxsel(numplot)})
end

end
```

helperExtractFeatures Extracts the wavelet features and AR coefficients for blocks of the data of a specified size. The features are concatenated into feature vectors.

```matlab
function [trainFeatures, testFeatures,featureindices] = helperExtractFeatures(trainData,testData
% This function is only in support of XpwWaveletMLExample. It may change or
% be removed in a future release.
trainFeatures = [];
testFeatures = [];

for idx =1:size(trainData,1)
    x = trainData(idx,:);
    x = detrend(x,0);
    arcoefs = blockAR(x,AR_order,T);
    se = shannonEntropy(x,T,level);
    [cp,rh] = leaders(x,T);
    wvar = modwtvar(modwt(x,'db2'),'db2');
    trainFeatures = [trainFeatures; arcoefs se cp rh wvar']; %#ok<AGROW>

end

for idx =1:size(testData,1)
    x1 = testData(idx,:);
    x1 = detrend(x1,0);
    arcoefs = blockAR(x1,AR_order,T);
    se = shannonEntropy(x1,T,level);
    [cp,rh] = leaders(x1,T);
    wvar = modwtvar(modwt(x1,'db2'),'db2');
    testFeatures = [testFeatures;arcoefs se cp rh wvar']; %#ok<AGROW>

end

featureindices = struct();
% 4*8
featureindices.ARfeatures = 1:32;
startidx = 33;
endidx = 33+(16*8)-1;
featureindices.SEfeatures = startidx:endidx;
startidx = endidx+1;
endidx = startidx+7;
featureindices.CP2features = startidx:endidx;
startidx = endidx+1;
endidx = startidx+7;
featureindices.HRfeatures = startidx:endidx;
startidx = endidx+1;
endidx = startidx+13;
featureindices.WVARfeatures = startidx:endidx;
end


function se = shannonEntropy(x,numbuffer,level)
numwindows = numel(x)/numbuffer;
y = buffer(x,numbuffer);
se = zeros(2^level,size(y,2));
for kk = 1:size(y,2)
    wpt = modwpt(y(:,kk),level);
    % Sum across time
    E = sum(wpt.^2,2);
    Pij = wpt.^2./E;
    % The following is eps(1)
    se(:,kk) = -sum(Pij.*log(Pij+eps),2);
end
```

```matlab
se = reshape(se,2^level*numwindows,1);
se = se';
end


function arcfs = blockAR(x,order,numbuffer)
numwindows = numel(x)/numbuffer;
y = buffer(x,numbuffer);
arcfs = zeros(order,size(y,2));
for kk = 1:size(y,2)
    artmp =  arburg(y(:,kk),order);
    arcfs(:,kk) = artmp(2:end);
end
arcfs = reshape(arcfs,order*numwindows,1);
arcfs = arcfs';
end


function [cp,rh] = leaders(x,numbuffer)
y = buffer(x,numbuffer);
cp = zeros(1,size(y,2));
rh = zeros(1,size(y,2));
for kk = 1:size(y,2)
    [~,h,cptmp] = dwtleader(y(:,kk));
    cp(kk) = cptmp(2);
    rh(kk) = range(h);
end
end
```

helperPrecisionRecall returns the precision, recall, and F1 scores based on the confusion matrix. Outputs the results as a MATLAB table.

```matlab
function PRTable = helperPrecisionRecall(confmat)
% This function is only in support of XpwWaveletMLExample. It may change or
% be removed in a future release.
precisionARR = confmat(1,1)/sum(confmat(:,1))*100;
precisionCHF = confmat(2,2)/sum(confmat(:,2))*100 ;
precisionNSR = confmat(3,3)/sum(confmat(:,3))*100 ;
recallARR = confmat(1,1)/sum(confmat(1,:))*100;
recallCHF = confmat(2,2)/sum(confmat(2,:))*100;
recallNSR = confmat(3,3)/sum(confmat(3,:))*100;
F1ARR = 2*precisionARR*recallARR/(precisionARR+recallARR);
F1CHF = 2*precisionCHF*recallCHF/(precisionCHF+recallCHF);
F1NSR = 2*precisionNSR*recallNSR/(precisionNSR+recallNSR);
% Construct a MATLAB Table to display the results.
PRTable = array2table([precisionARR recallARR F1ARR;...
    precisionCHF recallCHF F1CHF; precisionNSR recallNSR...
    F1NSR],'VariableNames',{'Precision','Recall','F1_Score'},'RowNames',...
    {'ARR','CHF','NSR'});

end
```

See Also

Functions
modwpt | modwt | modwtvar

Apps
Classification Learner App

Wavelet Time Scattering for ECG Signal Classification

This example shows how to classify human electrocardiogram (ECG) signals using wavelet time scattering and a support vector machine (SVM) classifier. In wavelet scattering, data is propagated through a series of wavelet transforms, nonlinearities, and averaging to produce low-variance representations of time series. Wavelet time scattering yields signal representations insensitive to shifts in the input signal without sacrificing class discriminability. You must have the Wavelet Toolbox™ and the Statistics and Machine Learning Toolbox™ to run this example. The data used in this example are publicly available from PhysioNet. You can find a deep learning approach to this classification problem in this example "Classify Time Series Using Wavelet Analysis and Deep Learning" on page 13-79 and a machine learning approach in this example "Signal Classification Using Wavelet-Based Features and Support Vector Machines" on page 13-2.

Data Description

This example uses ECG data obtained from three groups, or classes, of people: persons with cardiac arrhythmia, persons with congestive heart failure, and persons with normal sinus rhythms. The example uses 162 ECG recordings from three PhysioNet databases: MIT-BIH Arrhythmia Database [3] [5], MIT-BIH Normal Sinus Rhythm Database [3], and The BIDMC Congestive Heart Failure Database [2][3]. In total, there are 96 recordings from persons with arrhythmia, 30 recordings from persons with congestive heart failure, and 36 recordings from persons with normal sinus rhythms. The goal is to train a classifier to distinguish between arrhythmia (ARR), congestive heart failure (CHF), and normal sinus rhythm (NSR).

Download Data

The first step is to download the data from the GitHub repository. To download the data, click `Clone or download` and select `Download ZIP`. Save the file `physionet_ECG_data-master.zip` in a folder where you have write permission. The instructions for this example assume you have downloaded the file to your temporary directory, (`tempdir` in MATLAB). Modify the subsequent instructions for unzipping and loading the data if you choose to download the data in folder different from `tempdir`. If you are familiar with Git, you can download the latest version of the tools (git) and obtain the data from a system command prompt using `git clone https://github.com/mathworks/physionet_ECG_data/`.

The file `physionet_ECG_data-master.zip` contains

* ECGData.zip
* README.md

and ECGData.zip contains

* ECGData.mat
* Modified_physionet_data.txt
* License.txt.

ECGData.mat holds the data used in this example. The .txt file, Modified_physionet_data.txt, is required by PhysioNet's copying policy and provides the source attributions for the data as well as a description of the pre-processing steps applied to each ECG recording.

Load Files

If you followed the download instructions in the previous section, enter the following commands to unzip the two archive files.

```
unzip(fullfile(tempdir,'physionet_ECG_data-master.zip'),tempdir)
unzip(fullfile(tempdir,'physionet_ECG_data-master','ECGData.zip'),...
    fullfile(tempdir,'ECGData'))
```

After you unzip the ECGData.zip file, load the data into MATLAB.

```
load(fullfile(tempdir,'ECGData','ECGData.mat'))
```

ECGData is a structure array with two fields: Data and Labels. Data is a 162-by-65536 matrix where each row is an ECG recording sampled at 128 hertz. Each ECG time series has a total duration of 512 seconds. Labels is a 162-by-1 cell array of diagnostic labels, one for each row of Data. The three diagnostic categories are: 'ARR' (arrhythmia), 'CHF' (congestive heart failure), and 'NSR' (normal sinus rhythm).

Create Training and Test Data

Randomly split the data into two sets - training and test data sets. The helper function helperRandomSplit performs the random split. helperRandomSplit accepts the desired split percentage for the training data and ECGData. The helperRandomSplit function outputs two data sets along with a set of labels for each. Each row of trainData and testData is an ECG signal. Each element of trainLabels and testLabels contains the class label for the corresponding row of the data matrices. In this example, we randomly assign 70% percent of the data in each class to the training set. The remaining 30% is held out for testing (prediction) and are assigned to the test set.

```
percent_train = 70;
[trainData,testData,trainLabels,testLabels] = ...
    helperRandomSplit(percent_train,ECGData);
```

There are 113 records in the trainData set and 49 records in testData. By design the training data contains 69.75% (113/162) of the data. Recall that the ARR class represents 59.26% of the data (96/162), the CHF class represents 18.52% (30/162), and the NSR class represents 22.22% (36/162). Examine the percentage of each class in the training and test sets. The percentages in each are consistent with the overall class percentages in the data set.

```
Ctrain = countcats(categorical(trainLabels))./numel(trainLabels).*100
Ctest = countcats(categorical(testLabels))./numel(testLabels).*100


Ctrain =

    59.2920
    18.5841
    22.1239


Ctest =

    59.1837
    18.3673
    22.4490
```

Plot Samples

Plot the first few thousand samples of four randomly selected records from ECGData. The helper function helperPlotRandomRecords does this. helperPlotRandomRecords accepts ECGData

and a random seed as input. The initial seed is set at 14 so that at least one record from each class is plotted. You can execute `helperPlotRandomRecords` with `ECGData` as the only input argument as many times as you wish to get a sense of the variety of ECG waveforms associated with each class. You can find the source code for this and all helper functions in the Supporting Functions section at the end of this example.

```
helperPlotRandomRecords(ECGData,14)
```
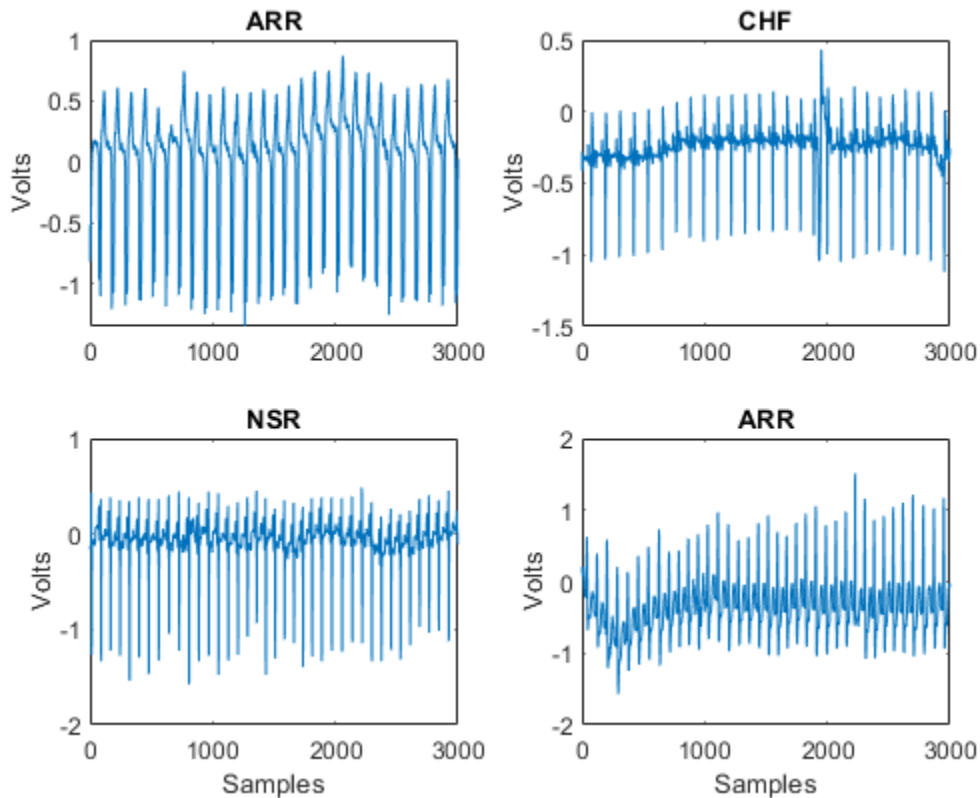

Wavelet Time Scattering

The key parameters to specify in a wavelet time scattering decomposition are the scale of the time invariant, the number of wavelet transforms, and the number of wavelets per octave in each of the wavelet filter banks. In many applications, the cascade of two filter banks is sufficient to achieve good performance. In this example, we construct a wavelet time scattering decomposition with the default filter banks: 8 wavelets per octave in the first filter bank and 1 wavelet per octave in the second filter bank. The invariance scale is set to 150 seconds.

```
N = size(ECGData.Data,2);
sf = waveletScattering('SignalLength',N, 'InvarianceScale',150,'SamplingFrequency',128);
```

You can visualize the wavelet filters in the two filter banks with the following.

```
[fb,f,filterparams] = filterbank(sf);
subplot(211)
plot(f,fb{2}.psift)
xlim([0 128])
grid on
```

```
title('1st Filter Bank Wavelet Filters');
subplot(212)
plot(f,fb{3}.psift)
xlim([0 128])
grid on
title('2nd Filter Bank Wavelet Filters');
xlabel('Hz');
```

1st Filter Bank Wavelet Filters

2nd Filter Bank Wavelet Filters
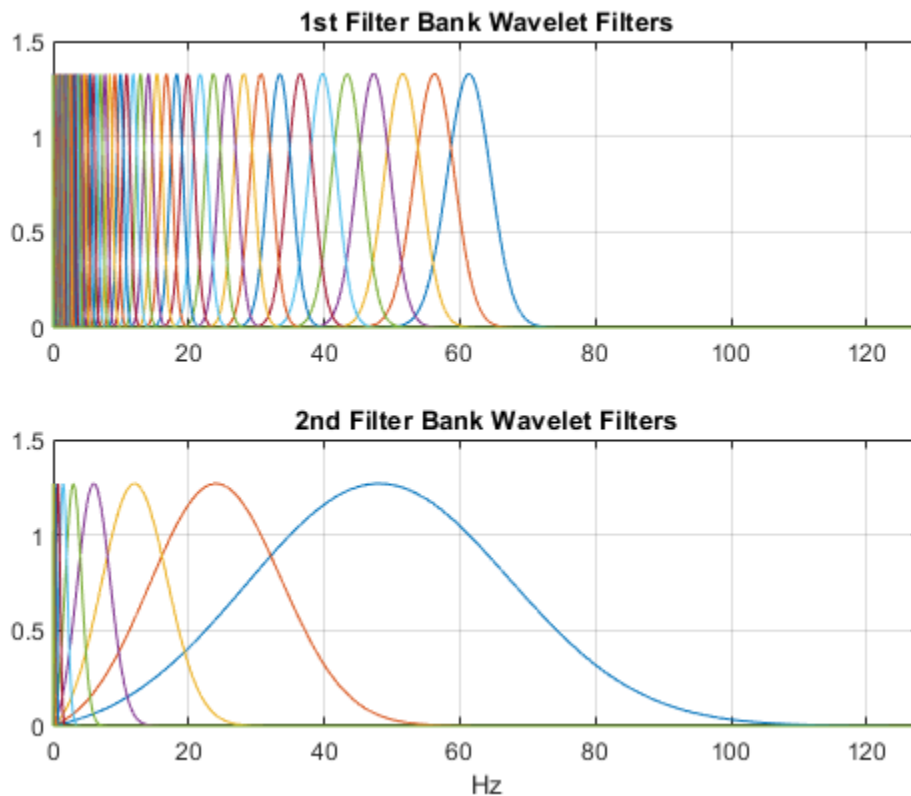
To demonstrate the invariance scale, obtain the inverse Fourier transform of the scaling function and center it at 0 seconds in time. The two vertical black lines mark the -75 and 75 second boundaries. Additionally, plot the real and imaginary parts of the coarsest-scale (lowest frequency) wavelet from the first filter bank. Note that the coarsest-scale wavelet does not exceed the invariant scale determined by the time support of the scaling function. This is an important property of wavelet time scattering.

```
figure;
phi = ifftshift(ifft(fb{1}.phift));
psiL1 = ifftshift(ifft(fb{2}.psift(:,end)));
t = (-2^15:2^15-1).*1/128;
scalplt = plot(t,phi);
hold on
grid on
ylim([-1.5e-4 1.6e-4]);
plot([-75 -75],[-1.5e-4 1.6e-4],'k--');
plot([75 75],[-1.5e-4 1.6e-4],'k--');
xlabel('Seconds'); ylabel('Amplitude');
wavplt = plot(t,[real(psiL1) imag(psiL1)]);
```

```
legend([scalplt wavplt(1) wavplt(2)],{'Scaling Function','Wavelet-Real Part','Wavelet-Imaginary
title({'Scaling Function';'Coarsest-Scale Wavelet First Filter Bank'})
```
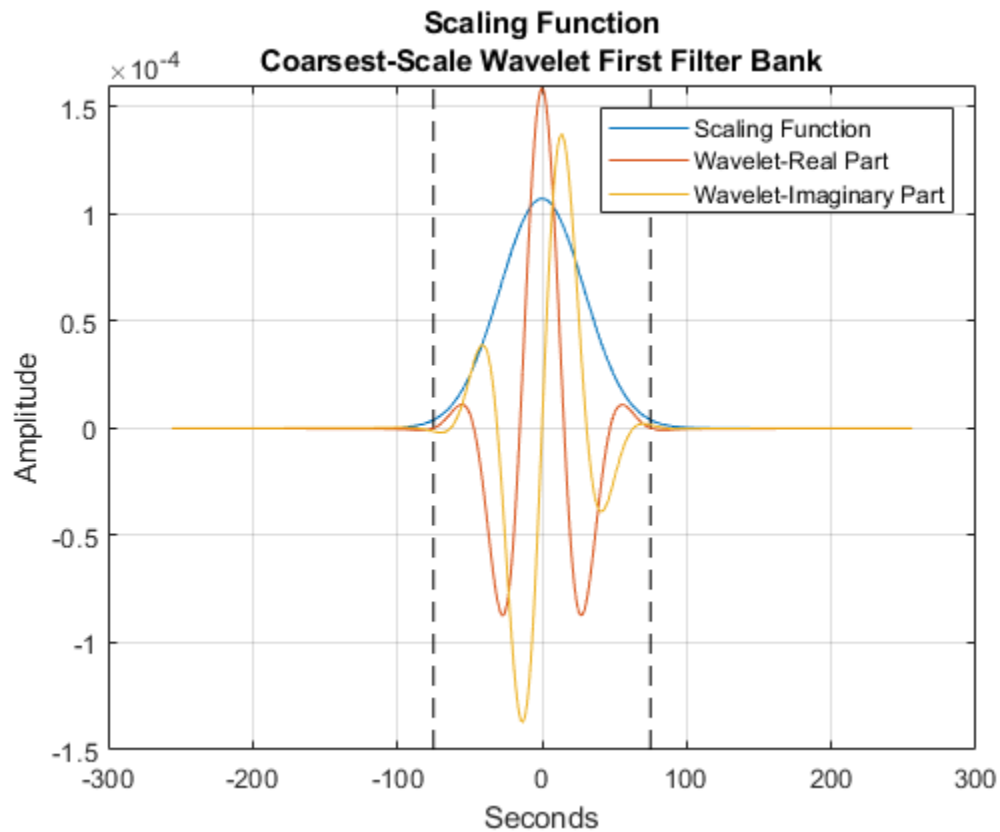

After constructing the scattering decomposition framework, obtain the scattering coefficients for the training data as a matrix. When you run `featureMatrix` with multiple signals, each column is treated a single signal.

```
scat_features_train = featureMatrix(sf,trainData');
```

The output of `featureMatrix` in this case is 416-by-16-by-113. Each page of the tensor, `scat_features_train`, is the scattering transform of one signal. The wavelet scattering transform is critically downsampled in time based on the bandwidth of the scaling function. In this case, this results in 16 time windows for each of the 416 scattering paths.

In order to obtain a matrix compatible with the SVM classifier, reshape the multisignal scattering transform into a matrix where each column corresponds to a scattering path and each row is a scattering time window. In this case, you obtain 1808 rows because there are 16 time windows for each of the 113 signals in the training data.

```
Nwin = size(scat_features_train,2);
scat_features_train = permute(scat_features_train,[2 3 1]);
scat_features_train = reshape(scat_features_train,...
    size(scat_features_train,1)*size(scat_features_train,2),[]);
```

Repeat the process for the test data. Initially, `scat_features_test` is 416-by-16-by-49 because there are 49 ECG waveforms in the training set. After reshaping for the SVM classifier, the feature matrix is 784-by-416.

```
scat_features_test = featureMatrix(sf,testData');
scat_features_test = permute(scat_features_test,[2 3 1]);
scat_features_test = reshape(scat_features_test,...
    size(scat_features_test,1)*size(scat_features_test,2),[]);
```

Because for each signal we obtained 16 scattering windows, we need to create labels to match the number of windows. The helper function `createSequenceLabels` does this based on the number of windows.

```
[sequence_labels_train,sequence_labels_test] = createSequenceLabels(Nwin,trainLabels,testLabels)
```

Cross Validation

For classification, two analyses are performed. The first uses all the scattering data and fits a multi-class SVM with a quadratic kernel. In total there 2592 scattering sequences in the entire dataset, 16 for each of the 162 signals. The error rate, or loss, is estimated using 5-fold cross validation.

```
scat_features = [scat_features_train; scat_features_test];
allLabels_scat = [sequence_labels_train; sequence_labels_test];
rng(1);
template = templateSVM(...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 2, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true);
classificationSVM = fitcecoc(...
    scat_features, ...
    allLabels_scat, ...
    'Learners', template, ...
    'Coding', 'onevsone', ...
    'ClassNames', {'ARR';'CHF';'NSR'});
kfoldmodel = crossval(classificationSVM, 'KFold', 5);
```

Compute the loss and the confusion matrix. Display the accuracy.

```
predLabels = kfoldPredict(kfoldmodel);
loss = kfoldLoss(kfoldmodel)*100;
confmatCV = confusionmat(allLabels_scat,predLabels)
fprintf('Accuracy is %2.2f percent.\n',100-loss);


confmatCV =

        1535           0           1
           1         479           0
           0           0         576

Accuracy is 99.92 percent.
```

The accuracy is 99.92%, which is quite good but the actual results are better considering that here each time window is classified separately. There are 16 separate classifications for each signal. Use a simple majority vote to obtain a single class prediction for each scattering representation.

```
classes = categorical({'ARR','CHF','NSR'});
[ClassVotes,ClassCounts] = helperMajorityVote(predLabels,[trainLabels; testLabels],classes);
```

Determine the actual cross-validation accuracy based on the mode of the class predictions for each set of scattering time windows. If the mode is not unique for a given set, `helperMajorityVote`

returns a classification error indicated by `'NoUniqueMode'`. This results in an extra column in the confusion matrix, which in this case is all zeros because a unique mode exists for each set of scattering predictions.

```
CVaccuracy = sum(eq(ClassVotes,categorical([trainLabels; testLabels])))/162*100;
fprintf('True cross-validation accuracy is %2.2f percent.\n',CVaccuracy);
MVconfmatCV = confusionmat(ClassVotes,categorical([trainLabels; testLabels]));
MVconfmatCV
```

```
True cross-validation accuracy is 100.00 percent.

MVconfmatCV =

    96     0     0     0
     0    30     0     0
     0     0    36     0
     0     0     0     0
```

Scattering has correctly classified all the signals in the cross-validated model. If you examine `ClassCounts`, you see that the 2 misclassified time windows in `confmatCV` are attributable to 2 signals where 15 of the 16 scattering windows were classified correctly.

SVM Classification

For the next analysis, we fit a multi-class quadratic SVM to the training data only (70%) and then use that model to make predictions on the 30% of the data held out for testing. There are 49 data records in the test set. Use a majority vote on the individual scattering windows.

```
model = fitcecoc(...
    scat_features_train, ...
    sequence_labels_train, ...
    'Learners', template, ...
    'Coding', 'onevsone', ...
    'ClassNames', {'ARR','CHF','NSR'});
predLabels = predict(model,scat_features_test);
[TestVotes,TestCounts] = helperMajorityVote(predLabels,testLabels,classes);
testaccuracy = sum(eq(TestVotes,categorical(testLabels)))/numel(testLabels)*100;
fprintf('The test accuracy is %2.2f percent. \n',testaccuracy);
testconfmat = confusionmat(TestVotes,categorical(testLabels));
testconfmat
```

```
The test accuracy is 97.96 percent.

testconfmat =

    29     1     0     0
     0     8     0     0
     0     0    11     0
     0     0     0     0
```

The classification accuracy on the test dataset is approximately 98%. The confusion matrix shows that one CHF record is misclassified as ARR. All 48 other signals are correctly classified.

Summary

This example used wavelet time scattering and an SVM classifier to classify ECG waveforms into one of three diagnostic classes. Wavelet scattering proved to be a powerful feature extractor, which

required only a minimal set of user-specified parameters to yield a set of robust features for classification. Compare this with the example "Signal Classification Using Wavelet-Based Features and Support Vector Machines" on page 13-2, which required a significant amount of expertise to handcraft features to use in classification. With wavelet time scattering, you are only required to specify the scale of the time invariance, the number of filter banks (or wavelet transforms), and the number of wavelets per octave. The combination of a wavelet scattering transform and an SVM classifier yielded 100% classification on a cross-validated model and 98% correct classification when applying an SVM to the scattering transforms of a hold-out test set.

References

1 Anden, J., Mallat, S. 2014. Deep scattering spectrum, IEEE Transactions on Signal Processing, 62, 16, pp. 4114-4128.

2 Baim DS, Colucci WS, Monrad ES, Smith HS, Wright RF, Lanoue A, Gauthier DF, Ransil BJ, Grossman W, Braunwald E. Survival of patients with severe congestive heart failure treated with oral milrinone. J American College of Cardiology 1986 Mar; 7(3):661-670.

3 Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng C-K, Stanley HE. PhysioBank, PhysioToolkit,and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. *Circulation*. Vol. 101, No. 23, 13 June 2000, pp. e215-e220. `http://circ.ahajournals.org/content/101/23/e215.full`

4 Mallat, S., 2012. Group invariant scattering. Communications in Pure and Applied Mathematics, 65, 10, pp. 1331-1398.

5 Moody GB, Mark RG. The impact of the MIT-BIH Arrhythmia Database. IEEE Eng in Med and Biol 20(3):45-50 (May-June 2001). (PMID: 11446209)

Supporting Functions

helperPlotRandomRecords Plots four ECG signals randomly chosen from `ECGData`.

```matlab
function helperPlotRandomRecords(ECGData,randomSeed)
% This function is only intended to support the XpwWaveletMLExample. It may
% change or be removed in a future release.

if nargin==2
    rng(randomSeed)
end

M = size(ECGData.Data,1);
idxsel = randperm(M,4);
for numplot = 1:4
    subplot(2,2,numplot)
    plot(ECGData.Data(idxsel(numplot),1:3000))
    ylabel('Volts')
    if numplot > 2
        xlabel('Samples')
    end
    title(ECGData.Labels{idxsel(numplot)})
end

end
```

helperMajorityVote Finds the mode in the predicted class labels for each set of scattering time windows. The function returns both the class label modes and the number of class predictions for each set of scattering time windows. If there is no unique mode, `helperMajorityVote` returns a class label of "error" to indicate that set of scattering windows is a classification error.

```matlab
function [ClassVotes,ClassCounts] = helperMajorityVote(predLabels,origLabels,classes)
% This function is in support of ECGWaveletTimeScatteringExample. It may
% change or be removed in a future release.

% Make categorical arrays if the labels are not already categorical
predLabels = categorical(predLabels);
origLabels = categorical(origLabels);
% Expects both predLabels and origLabels to be categorical vectors
Npred = numel(predLabels);
Norig = numel(origLabels);
Nwin = Npred/Norig;
predLabels = reshape(predLabels,Nwin,Norig);
ClassCounts = countcats(predLabels);
[mxcount,idx] = max(ClassCounts);
ClassVotes = classes(idx);
% Check for any ties in the maximum values and ensure they are marked as
% error if the mode occurs more than once
modecnt = modecount(ClassCounts,mxcount);
ClassVotes(modecnt>1) = categorical({'error'});
ClassVotes = ClassVotes(:);

%--------------------------------------------------------------------------
function modecnt = modecount(ClassCounts,mxcount)
modecnt = Inf(size(ClassCounts,2),1);
for nc = 1:size(ClassCounts,2)
    modecnt(nc) = histc(ClassCounts(:,nc),mxcount(nc));
end

end

% EOF
end
```

See Also

waveletScattering

More About

- "Wavelet Scattering" on page 9-2

Music Genre Classification Using Wavelet Time Scattering

This example shows how to classify the genre of a musical excerpt using wavelet time scattering and the audio datastore. In wavelet scattering, data is propagated through a series of wavelet transforms, nonlinearities, and averaging to produce low-variance representations of the data. These low-variance representations are then used as inputs to a classifier.

GTZAN Dataset

The data set used in this example is the GTZAN Genre Collection [7][8]. The data is provided as a zipped tar archive which is approximately 1.2 GB. The uncompressed data set requires about 3 GB of disk space. Extracting the compressed tar file from the link provided in the references creates a folder with ten subfolders. Each subfolder is named for the genre of music samples it contains. The genres are: blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, and rock. There are 100 examples of each genre and each audio file consists of about 30 seconds of data sampled at 22050 Hz. In the original paper, the authors used a number of time-domain and frequency-domain features including mel-frequency cepstral (MFC) coefficients extracted from each music example and a Gaussian mixture model (GMM) classification to achieve an accuracy of 61 percent [7]. Subsequently, deep learning networks have been applied to this data. In most cases, these deep learning approaches consist of convolutional neural networks (CNN) with the MFC coefficients or spectrograms as the input to the deep CNN. These approaches have resulted in performance of around 84% [4]. An LSTM approach with spectrogram time slices resulted in 79% accuracy and time-domain and frequency-domain features coupled with an ensemble learning approach (AdaBoost) resulted in 82% accuracy on a test set [2][3]. Recently, a sparse representation machine learning approach achieved approximately 89% accuracy [6].

Wavelet Scattering Framework

The only parameters to specify in a wavelet time scattering framework are the duration of the time invariance, the number of wavelet filter banks, and the number of wavelets per octave. For most applications, cascading the data through two wavelet filter banks is sufficient. In this example, we use the default scattering framework which uses two wavelet filter banks. The first filter bank has 8 wavelets per octave and the second filter bank has 1 wavelet per octave. For this example, set the invariant scale to be 0.5 seconds, which corresponds to slightly more than 11,000 samples for the given sampling rate. Create the wavelet time scattering decomposition framework.

```matlab
sf = waveletScattering('SignalLength',2^19,'SamplingFrequency',22050,...
    'InvarianceScale',0.5);
```

To understand the role of the invariance scale, obtain and plot the scaling filter in time along with the real and imaginary parts of the coarsest-scale wavelet from the first filter bank. Note that the time-support of the scaling filter is essentially 0.5 seconds as designed. Further, the time support of the coarsest-scale wavelet does not exceed the invariant scale of the wavelet scattering decomposition.

```matlab
[fb,f,filterparams] = filterbank(sf);
phi = ifftshift(ifft(fb{1}.phift));
psiL1 = ifftshift(ifft(fb{2}.psift(:,end)));
dt = 1/22050;
time = -2^18*dt:dt:2^18*dt-dt;
scalplt = plot(time,phi,'linewidth',1.5);
hold on
grid on
ylimits = [-3e-4 3e-4];
ylim(ylimits);
plot([-0.25 -0.25],ylimits,'k--');
```

```
plot([0.25 0.25],ylimits,'k--');
xlim([-0.6 0.6]);
xlabel('Seconds'); ylabel('Amplitude');
wavplt = plot(time,[real(psiL1) imag(psiL1)]);
legend([scalplt wavplt(1) wavplt(2)],{'Scaling Function','Wavelet-Real Part','Wavelet-Imaginary I
title({'Scaling Function';'Coarsest-Scale Wavelet First Filter Bank'})
hold off
```

Scaling Function
Coarsest-Scale Wavelet First Filter Bank
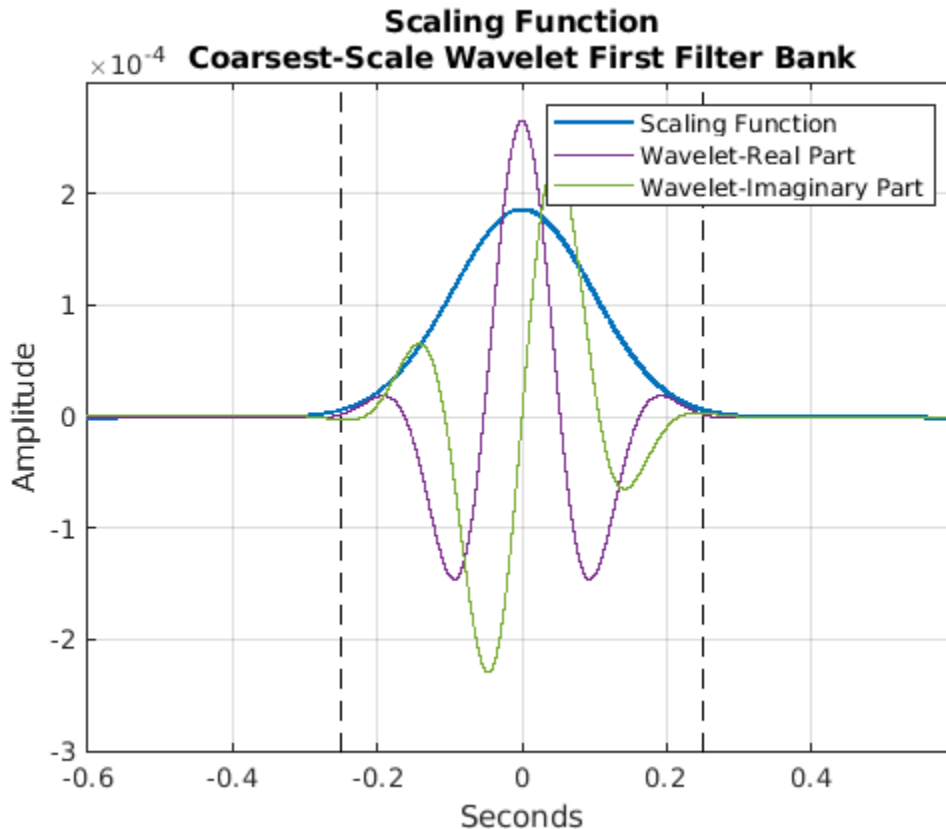
Audio Datastore

The audio datastore enables you to manage collections of audio data files. For machine or deep learning, the audio datastore not only manages the flow of audio data from files and folders, the audio datastore also manages the association of labels with the data and provides the ability to randomly partition your data into different sets for training, validation, and testing. In this example, use the audio datastore to manage the GTZAN music genre collection. Recall each subfolder of the collection is named for the genre it represents. Set the `'IncludeSubFolders'` property to `true` to instruct the audio datastore to use subfolders and set the `'LabelSource'` property to `'foldernames'` to create data labels based on the subfolder names. This example assumes the top-level directory is inside your MATLAB `tempdir` directory and is called 'genres'. Ensure that `location` is the correct path to the top-level data folder on your machine. The top-level data folder on your machine should contain ten subfolders each named for the ten genres and must only contain audio files corresponding to those genres.

```
location = fullfile(tempdir,'genres');
ads = audioDatastore(location,'IncludeSubFolders',true,...
    'LabelSource','foldernames');
```

Run the following to obtain a count of the musical genres in the data set.

```
countEachLabel(ads)
```

```
ans =

  10×2 table

    Label        Count
   _____    _____

    blues         100
    classical     100
    country       100
    disco         100
    hiphop        100
    jazz          100
    metal         100
    pop           100
    reggae        100
    rock          100
```

As previously stated, there are 10 genres with 100 files each.

Training and Test Sets

Create training and test sets to develop and test our classifier. We use 80% of the data for training and hold out the remaining 20% for testing. The `shuffle` function of the audio datastore randomly shuffles the data. Do this prior to splitting the data by label to randomize the data. In this example, we set the random number generator seed for reproducibility. Use the audio datastore `splitEachLabel` function to perform the 80-20 split. `splitEachLabel` ensures that all classes are equally represented.

```
rng(100);
ads = shuffle(ads);
[adsTrain,adsTest] = splitEachLabel(ads,0.8);
countEachLabel(adsTrain)
countEachLabel(adsTest)
```

```
ans =

  10×2 table

    Label        Count
   _____    _____

    blues          80
    classical      80
    country        80
    disco          80
    hiphop         80
    jazz           80
    metal          80
    pop            80
    reggae         80
```

```
    rock            80


ans =

  10×2 table

     Label       Count
    _____    _____

    blues         20
    classical     20
    country       20
    disco         20
    hiphop        20
    jazz          20
    metal         20
    pop           20
    reggae        20
    rock          20
```

You see that there are 800 records in the training data as expected and 200 records in the test data. Additionally, there are 80 examples of each genre in the training set and 20 examples of each genre in the test set.

`audioDatastore` works with MATLAB tall arrays. Create tall arrays for both the training and test sets. Depending on your system, the number of workers in the parallel pool MATLAB creates may be different.

```
Ttrain = tall(adsTrain);
Ttest = tall(adsTest);

Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 6).
```

To obtain the scattering features, define a helper function, `helperscatfeatures`, that obtains the natural logarithm of the scattering features for 2^19 samples of each audio file and subsamples the number of scattering windows by 8. The source code for `helperscatfeatures` is listed in the appendix. We will compute the wavelet scattering features for both the training and test data.

```
scatteringTrain = cellfun(@(x)helperscatfeatures(x,sf),Ttrain,'UniformOutput',false);
scatteringTest = cellfun(@(x)helperscatfeatures(x,sf),Ttest,'UniformOutput',false);
```

Compute the scattering features on the training data and bundle all the features together in a matrix. This process takes several minutes.

```
TrainFeatures = gather(scatteringTrain);
TrainFeatures = cell2mat(TrainFeatures);

Evaluating tall expression using the Parallel Pool 'local':
- Pass 1 of 1: Completed in 3 min 40 sec
Evaluation completed in 3 min 40 sec
```

Repeat this process for the test data.

```
TestFeatures = gather(scatteringTest);
TestFeatures = cell2mat(TestFeatures);
```

```
Evaluating tall expression using the Parallel Pool 'local':
- Pass 1 of 1: Completed in 56 sec
Evaluation completed in 56 sec
```

Each row of `TrainFeatures` and `TestFeatures` is one scattering time window across the 341 paths in the scattering transform of each audio signal. For each music sample, we have 32 such time windows. Accordingly, the feature matrix for the training data is 25600-by-341. The number of rows is equal to the number of training examples (800) multiplied by the number of scattering windows per example (32). Similarly, the scattering feature matrix for the test data is 6400-by-341. There are 200 test examples and 32 windows per example. Create a genre label for each of the 32 windows in the wavelet scattering feature matrix for the training data.

```
numTimeWindows = 32;
trainLabels = adsTrain.Labels;
numTrainSignals = numel(trainLabels);
trainLabels = repmat(trainLabels,1,numTimeWindows);
trainLabels = reshape(trainLabels',numTrainSignals*numTimeWindows,1);
```

Repeat the process for the test data.

```
testLabels = adsTest.Labels;
numTestSignals = numel(testLabels);
testLabels = repmat(testLabels,1,numTimeWindows);
testLabels = reshape(testLabels',numTestSignals*numTimeWindows,1);
```

In this example, use a multi-class support vector machine (SVM) classifier with a cubic polynomial kernel. Fit the SVM to the training data.

```
template = templateSVM(...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 3, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true);
Classes = {'blues','classical','country','disco','hiphop','jazz',...
    'metal','pop','reggae','rock'};
classificationSVM = fitcecoc(...
    TrainFeatures, ...
    trainLabels, ...
    'Learners', template, ...
    'Coding', 'onevsone','ClassNames',categorical(Classes));
```

Test Set Prediction

Use the SVM model fit to the scattering transforms of the training data to predict the music genres for the test data. Recall there are 32 time windows for each signal in the scattering transform. Use a simple majority vote to predict the genre. The helper function `helperMajorityVote` obtains the mode of the genre labels over all 32 scattering windows. If there is no unique mode, `helperMajorityVote` returns a classification error indicated by `'NoUniqueMode'`. This results in an extra column in the confusion matrix. The source code for `helperMajorityVote` is listed in the appendix.

```
predLabels = predict(classificationSVM,TestFeatures);
[TestVotes,TestCounts] = helperMajorityVote(predLabels,adsTest.Labels,categorical(Classes));
testAccuracy = sum(eq(TestVotes,adsTest.Labels))/numTestSignals*100;
```

The test accuracy, `testAccuracy`, is 88 percent. This accuracy is comparable with the state of the art of the GTZAN dataset.

Display the confusion matrix to inspect the genre-by-genre accuracy rates. Recall there are 20 examples in each class.

```
confusionchart(TestVotes,adsTest.Labels)
```

True Class \ Predicted Class	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock	NoUniqueMode
blues	19										
classical		19		1							
country	1		18	1		1		1	1	1	
disco				17							
hiphop				1	17				2		
jazz		1				19				2	
metal					2		20				
pop								17		3	
reggae					1			1	16		
rock			2						1	14	
NoUniqueMode								1			
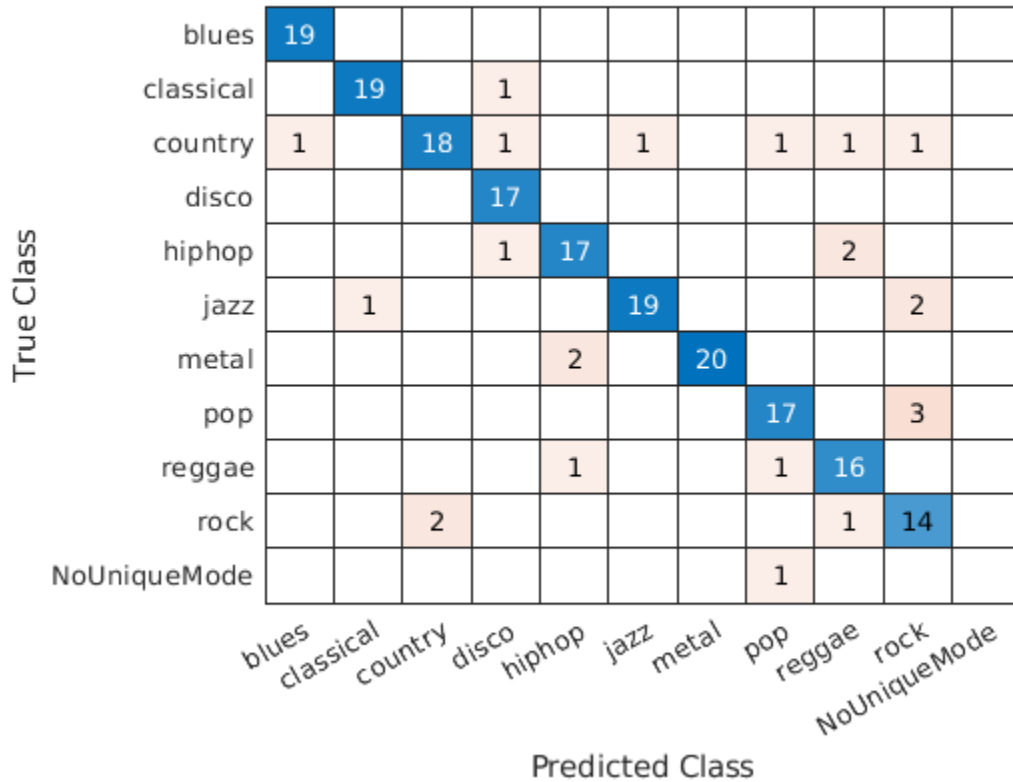
The diagonal of the confusion matrix plot shows that the classification accuracies for the individual genres is quite good in general. Extract these genre accuracies and plot separately.

```
cm = confusionmat(TestVotes,adsTest.Labels);
cm(:,end) = [];
genreAccuracy = diag(cm)./20*100;
figure;
bar(genreAccuracy)
set(gca,'XTickLabels',Classes);
xtickangle(gca,30);
title('Percentage Correct by Genre - Test Set');
```

Percentage Correct by Genre - Test Set

Summary

This example demonstrated the use of wavelet time scattering and the audio datastore in music genre classification. In this example, wavelet time scattering achieved an classification accuracy comparable to state of the art performance for the GTZAN dataset. As opposed to other approaches requiring the extraction of a number of time-domain and frequency-domain features, wavelet scattering only required the specification of a single parameter, the scale of the time invariant. The audio datastore enabled us to efficiently manage the transfer of a large dataset from disk into MATLAB and permitted us to randomize the data and accurately retain genre membership of the randomized data through the classification workflow.

References

1   Anden, J. and Mallat, S. 2014. Deep scattering spectrum. IEEE Transactions on Signal Processing, Vol. 62, 16, pp. 4114-4128.

2   Bergstra, J., Casagrande, N., Erhan, D., Eck, D., and Kegl, B. Aggregate features and AdaBoost for music classification. Machine Learning, Vol. 65, Issue 2-3, pp. 473-484.

3   Irvin, J., Chartock, E., and Hollander, N. 2016. Recurrent neural networks with attention for genre classification. https://www.semanticscholar.org/paper/Recurrent-Neural-Networks-with-Attention-for-Genre-Irvin/6da301817851f19107447e4c72e682e3f183ae8a

4   Li, T., Chan, A.B., and Chun, A. 2010. Automatic musical pattern feature extraction using convolutional neural network. International Conference Data Mining and Applications.

5   Mallat. S. 2012. Group invariant scattering. Communications on Pure and Applied Mathematics, Vol. 65, 10, pp. 1331-1398.

6 Panagakis, Y., Kotropoulos, C.L., and Arce, G.R. 2014. Music genre classification via joint sparse low-rank representation of audio features. IEEE Transactions on Audio, Speech, and Language Processing, 22, 12, pp. 1905-1917.

7 Tzanetakis, G. and Cook, P. 2002. Music genre classification of audio signals. IEEE Transactions on Speech and Audio Processing, Vol. 10, No. 5, pp. 293-302.

8 *GTZAN Genre Collection.* `http://marsyas.info/downloads/datasets.html`

Appendix -- Supporting Functions

helperMajorityVote -- This function returns the mode of the class labels predicted over a number of feature vectors. In wavelet time scattering, we obtain a class label for each time window. If no unique mode is found a label of 'NoUniqueMode' is returned to denote a classification error.

```matlab
function [ClassVotes,ClassCounts] = helperMajorityVote(predLabels,origLabels,classes)
% This function is in support of wavelet scattering examples only. It may
% change or be removed in a future release.

% Make categorical arrays if the labels are not already categorical
predLabels = categorical(predLabels);
origLabels = categorical(origLabels);
% Expects both predLabels and origLabels to be categorical vectors
Npred = numel(predLabels);
Norig = numel(origLabels);
Nwin = Npred/Norig;
predLabels = reshape(predLabels,Nwin,Norig);
ClassCounts = countcats(predLabels);
[mxcount,idx] = max(ClassCounts);
ClassVotes = classes(idx);
% Check for any ties in the maximum values and ensure they are marked as
% error if the mode occurs more than once
modecnt = modecount(ClassCounts,mxcount);
ClassVotes(modecnt>1) = categorical({'NoUniqueMode'});
ClassVotes = ClassVotes(:);

%-------------------------------------------------------------------------
    function modecnt = modecount(ClassCounts,mxcount)
        modecnt = Inf(size(ClassCounts,2),1);
        for nc = 1:size(ClassCounts,2)
            modecnt(nc) = histc(ClassCounts(:,nc),mxcount(nc));
        end
    end
end
```

helperscatfeatures - This function returns the wavelet time scattering feature matrix for a given input signal. In this case, we use the natural logarithm of the wavelet scattering coefficients. The scattering feature matrix is computed on 2^{19} samples of a signal. The scattering features are subsampled by a factor of 8.

```matlab
function features = helperscatfeatures(x,sf)
% This function is in support of wavelet scattering examples only. It may
% change or be removed in a future release.

features = featureMatrix(sf,x(1:2^19),'Transform','log');
```

```
features = features(:,1:8:end)';
end
```

See Also
`waveletScattering`

More About

Wavelet Time Scattering with GPU Acceleration — Music Genre Classification

This example shows how to accelerate the computation of wavelet scattering features using `gpuArray` (Parallel Computing Toolbox) and a parallelized "depth-first" version of wavelet time scattering. You must have a CUDA-enabled NVIDIA GPU with compute capability 3.0 or higher. See "GPU Support by Release" (Parallel Computing Toolbox) for details.

This example reproduces the CPU version found in "Music Genre Classification Using Wavelet Time Scattering" on page 13-26. The example uses wavelet scattering features with a support vector machine to classify the genre of a musical excerpt. The example uses the audio datastore to manage reading of the audio files from disk as well as creating training and test sets.

GTZAN Dataset

The data set used in this example is the GTZAN Genre Collection [7][8]. The data is provided as a zipped tar archive which is approximately 1.2 GB. The uncompressed data set requires about 3 GB of disk space. Extracting the compressed tar file from the link provided in the references creates a folder with ten subfolders. Each subfolder is named for the genre of music samples it contains. The genres are: blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, and rock. There are 100 examples of each genre and each audio file consists of about 30 seconds of data sampled at 22050 Hz. In the original paper, the authors used a number of time-domain and frequency-domain features including mel-frequency cepstral (MFC) coefficients extracted from each music example and a Gaussian mixture model (GMM) classification to achieve an accuracy of 61 percent [7]. Subsequently, deep learning networks have been applied to this data. In most cases, these deep learning approaches consist of convolutional neural networks (CNN) with the MFC coefficients or spectrograms as the input to the deep CNN. These approaches have resulted in performance of around 84% [4]. An LSTM approach with spectrogram time slices resulted in 79% accuracy and time-domain and frequency-domain features coupled with an ensemble learning approach (AdaBoost) resulted in 82% accuracy on a test set [2][3]. Recently, a sparse representation machine learning approach achieved approximately 89% accuracy [6].

Wavelet Scattering Framework

The wavelet scattering framework in this example uses two filter banks with 8 wavelets per octave in the first filter bank and 1 wavelet per octave in the second filter bank. The invariant scale is set to 0.5 seconds, which corresponds to slightly more than 11000 samples for the sample rate of 22050 Hz.

Create the wavelet time scattering network.

```
sf = helpergpuscat1('SignalLength',2^19,'SamplingFrequency',22050,...
    'InvarianceScale',0.5,'OversamplingFactor',0);
```

Audio Datastore

Use `audioDatastore` to manage the GTZAN music genre collection. Each subfolder of the collection is named for the genre it represents. Set the `'IncludeSubFolders'` property to `true` to instruct the audio datastore to use subfolders and set the `'LabelSource'` property to `'foldernames'` to create data labels based on the subfolder names. Set `pathToData` to a character vector containing the top-level data folder on your machine. The top-level data folder on your machine should contain ten subfolders each one named for one of the ten genres and must only contain audio files corresponding to those genres. This example assumes the top-level directory is inside your MATLAB™ `tempdir` directory and is called `'genres'`. Extracting the compressed tar file should provide a folder containing the genre subfolders.

```
pathToData = fullfile(tempdir,'genres');
location = pathToData;
ads = audioDatastore(location,'IncludeSubFolders',true,...
    'LabelSource','foldernames');
```

Run the following to obtain a count of the musical genres in the data set.

```
countEachLabel(ads)
```

```
ans=10×2 table
     Label        Count
    _____      _____

    blues          100
    classical      100
    country        100
    disco          100
    hiphop         100
    jazz           100
    metal          100
    pop            100
    reggae         100
    rock           100
```

As previously stated, there are 10 genres with 100 files each.

Training and Test Sets

Create training and test sets to develop and test our classifier. Allocate 80% of the data for training and hold out the remaining 20% for testing. The `shuffle` function of the audio datastore randomly shuffles the data. Do this prior to splitting the data by label to randomize the data. In this example, we set the random number generator seed for reproducibility. Use the audio datastore `splitEachLabel` function to perform the 80-20 split. `splitEachLabel` ensures that all classes are equally represented.

```
rng(100);
ads = shuffle(ads);
[adsTrain,adsTest] = splitEachLabel(ads,0.8);
countEachLabel(adsTrain)
```

```
ans=10×2 table
     Label        Count
    _____      _____

    blues          80
    classical      80
    country        80
    disco          80
    hiphop         80
    jazz           80
    metal          80
    pop            80
    reggae         80
    rock           80
```

```
countEachLabel(adsTest)
```

```
ans=10×2 table
     Label        Count
    _____     _____

    blues          20
    classical      20
    country        20
    disco          20
    hiphop         20
    jazz           20
    metal          20
    pop            20
    reggae         20
    rock           20
```

You see that there are 800 records in the training data as expected and 200 records in the test data. Additionally, there are 80 examples of each genre in the training set and 20 examples of each genre in the test set.

To obtain the scattering features, define a helper function, `helperbatchscatfeatures`, that obtains the natural logarithm of the scattering features for 2^19 samples of each audio file and subsamples the number of scattering windows by 8. The source code for `helperscatfeatures` is listed in the appendix. Wavelet scattering features are computed using a batch size of 64 signals. The use of `gpuArray` (Parallel Computing Toolbox) with a CUDA-enabled NVIDIA GPU provides a significant acceleration for this batch computation. With this scattering framework, batch size, and GPU (NVIDIA Titan XP), the GPU implementation reduces the time needed to compute the scattering features by approximately a factor of 7.

```
N = 2^19;
batchsize = 64;
scTrain = [];
while hasdata(adsTrain)
    sc = helperbatchscatfeatures(adsTrain,sf,N,batchsize);
    scTrain = cat(3,scTrain,sc);
end
```

Record the number of time windows in the scattering transform for label creation.

```
numTimeWindows = size(sc,2);
```

Repeat the feature extraction process for the test data.

```
scTest = [];
while hasdata(adsTest)
    sc = helperbatchscatfeatures(adsTest,sf,N,batchsize);
    scTest = cat(3,scTest,sc);
end
```

Determine the number of paths in the scattering network and reshape the training and test features into 2-D matrices.

```
[~,npaths] = sf.dfspaths();
Npaths = sum(npaths);
TrainFeatures = permute(scTrain,[2 3 1]);
TrainFeatures = reshape(TrainFeatures,[],Npaths,1);
TestFeatures = permute(scTest,[2 3 1]);
TestFeatures = reshape(TestFeatures,[],Npaths,1);
```

Each row of `TrainFeatures` and `TestFeatures` is one scattering time window across the `Npaths` (341) paths in the scattering transform of each audio signal. For each music sample, there are `numTimeWindows` such time windows. Accordingly, the feature matrix for the training data is 25600-by-341. The number of rows is equal to the number of training examples (800) multiplied by the number of scattering windows per example (32). Similarly, the scattering feature matrix for the test data is 6400-by-341. There are 200 test examples and 32 windows per example. Create a genre label for each of the 32 windows in the wavelet scattering feature matrix for the training data.

```
trainLabels = adsTrain.Labels;
numTrainSignals = numel(trainLabels);
trainLabels = repmat(trainLabels,1,numTimeWindows);
trainLabels = reshape(trainLabels',numTrainSignals*numTimeWindows,1);
```

Repeat the process for the test data.

```
testLabels = adsTest.Labels;
numTestSignals = numel(testLabels);
testLabels = repmat(testLabels,1,numTimeWindows);
testLabels = reshape(testLabels',numTestSignals*numTimeWindows,1);
```

In this example, use a multi-class support vector machine (SVM) classifier with a cubic polynomial kernel. Fit the SVM to the training data.

```
template = templateSVM(...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 3, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true);
Classes = {'blues','classical','country','disco','hiphop','jazz',...
    'metal','pop','reggae','rock'};
trainingOptions = struct('UseParallel',true);
classificationSVM = fitcecoc(...
    TrainFeatures, ...
    trainLabels, ...
    'Learners', template, ...
    'Coding', 'onevsone','ClassNames',categorical(Classes));
```

Test Set Prediction

Use the SVM model fit to the scattering transforms of the training data to predict music genres for the test data. Recall there are 32 time windows for each signal in the scattering transform. Use a simple majority vote to predict the genre. The helper function `helperMajorityVote` obtains the mode of the genre labels over all 32 scattering windows. If there is no unique mode, `helperMajorityVote` returns a classification error indicated by `'NoUniqueMode'`. This results in an extra column in the confusion matrix. The source code for `helperMajorityVote` is listed in the appendix.

```
predLabels = predict(classificationSVM,TestFeatures);
[TestVotes,TestCounts] = helperMajorityVote(predLabels,adsTest.Labels,categorical(Classes));
testAccuracy = sum(eq(TestVotes,adsTest.Labels))/numTestSignals*100
```

```
testAccuracy = 87.5000
```

The test accuracy, `testAccuracy`, is approximately 87.5 percent. This accuracy is comparable with the state of the art of the GTZAN dataset.

Display the confusion matrix to inspect the genre-by-genre accuracy rates. Recall there are 20 examples in each class.

```
confusionchart(TestVotes,adsTest.Labels)
```

True Class \ Predicted Class	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock	NoUniqueMode
blues	18										
classical		20		1							
country	2		18	1		1		1	2	1	
disco				16							
hiphop				2	17				2		
jazz						19				2	
metal					2		20				
pop								17		1	
reggae								1	15	1	
rock			2						1	15	
NoUniqueMode					1			1			
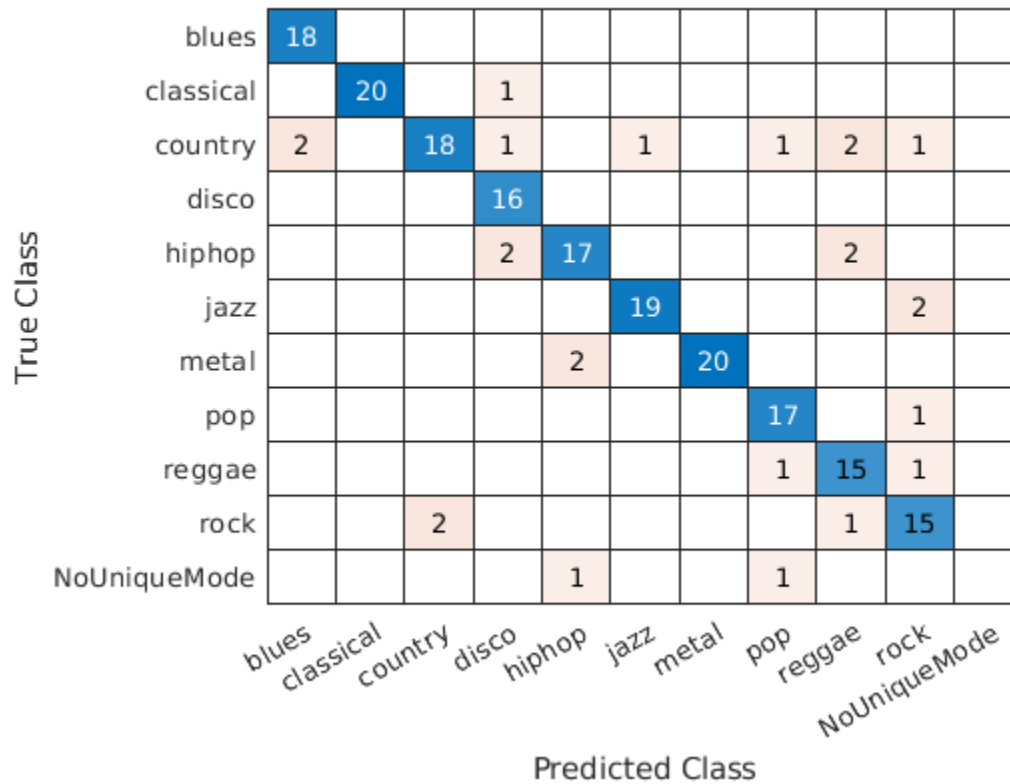
The diagonal of the confusion matrix plot shows that the classification accuracies for the individual genres is quite good in general. Extract these genre accuracies and plot separately.

```
cm = confusionmat(TestVotes,adsTest.Labels);
cm(:,end) = [];
genreAccuracy = diag(cm)./20*100;
figure;
bar(genreAccuracy)
set(gca,'XTickLabels',Classes);
xtickangle(gca,30);
title('GPU Example: Percent Correct by Genre (Test Set)');
```

GPU Example: Percent Correct by Genre (Test Set)

Summary

This example demonstrated the use of wavelet time scattering and the audio datastore in music genre classification. In this example, wavelet time scattering achieved an classification accuracy comparable to state of the art performance for the GTZAN dataset. As opposed to other approaches requiring the extraction of a number of time-domain and frequency-domain features, wavelet scattering only required the specification of a single parameter, the scale of the time invariant. The audio datastore enabled us to efficiently manage the transfer of a large dataset from disk into MATLAB and permitted us to randomize the data and accurately retain genre membership of the randomized data through the classification workflow.

References

1 Anden, J. and Mallat, S. 2014. Deep scattering spectrum. IEEE Transactions on Signal Processing, Vol. 62, 16, pp. 4114-4128.

2 Bergstra, J., Casagrande, N., Erhan, D., Eck, D., and Kegl, B. Aggregate features and AdaBoost for music classification. Machine Learning, Vol. 65, Issue 2-3, pp. 473-484.

3 Irvin, J., Chartock, E., and Hollander, N. 2016. Recurrent neural networks with attention for genre classification. https://www.semanticscholar.org/paper/Recurrent-Neural-Networks-with-Attention-for-Genre-Irvin/6da301817851f19107447e4c72e682e3f183ae8a

4 Li, T., Chan, A.B., and Chun, A. 2010. Automatic musical pattern feature extraction using convolutional neural network. International Conference Data Mining and Applications.

5 Mallat. S. 2012. Group invariant scattering. Communications on Pure and Applied Mathematics, Vol. 65, 10, pp. 1331-1398.

6 Panagakis, Y., Kotropoulos, C.L., and Arce, G.R. 2014. Music genre classification via joint sparse low-rank representation of audio features. IEEE Transactions on Audio, Speech, and Language Processing, 22, 12, pp. 1905-1917.

7 Tzanetakis, G. and Cook, P. 2002. Music genre classification of audio signals. IEEE Transactions on Speech and Audio Processing, Vol. 10, No. 5, pp. 293-302.

8 *GTZAN Genre Collection.* `http://marsyas.info/downloads/datasets.html`

Appendix — Supporting Functions

helperMajorityVote — This function returns the mode of the class labels predicted over a number of feature vectors. In wavelet time scattering, we obtain a class label for each time window. If no unique mode is found a label of 'NoUniqueMode' is returned to denote a classification error.

```matlab
function [ClassVotes,ClassCounts] = helperMajorityVote(predLabels,origLabels,classes)
% This function is in support of wavelet scattering examples only. It may
% change or be removed in a future release.

% Make categorical arrays if the labels are not already categorical
predLabels = categorical(predLabels);
origLabels = categorical(origLabels);
% Expects both predLabels and origLabels to be categorical vectors
Npred = numel(predLabels);
Norig = numel(origLabels);
Nwin = Npred/Norig;
predLabels = reshape(predLabels,Nwin,Norig);
ClassCounts = countcats(predLabels);
[mxcount,idx] = max(ClassCounts);
ClassVotes = classes(idx);
% Check for any ties in the maximum values and ensure they are marked as
% error if the mode occurs more than once
modecnt = modecount(ClassCounts,mxcount);
ClassVotes(modecnt>1) = categorical({'NoUniqueMode'});
ClassVotes = ClassVotes(:);

%--------------------------------------------------------------------------
    function modecnt = modecount(ClassCounts,mxcount)
        modecnt = Inf(size(ClassCounts,2),1);
        for nc = 1:size(ClassCounts,2)
            modecnt(nc) = histc(ClassCounts(:,nc),mxcount(nc));
        end
    end
end
```

helperbatchscatfeatures — This function returns the wavelet time scattering feature matrix for a given input signal. In this case, we use the natural logarithm of the wavelet scattering coefficients. The scattering feature matrix is computed on 2^19 samples of a signal. The scattering features are subsampled by a factor of 8.

```matlab
function sc = helperbatchscatfeatures(ds,sf,N,batchsize)
% This function is only intended to support examples in the Wavelet
% Toolbox. It may be changed or removed in a future release.

% Read batch of data from audio datastore
batch = helperReadBatch(ds,N,batchsize);
% Obtain scattering features
S = sf.featureMatrix(batch,'transform','log');
```

```
% Subsample the features
sc = S(:,1:8:end,:);
end
```

helperReadBatch — This function reads batches of a specified size from a datastore. Each column of the output is a separate signal from the datastore. The output may have fewer columns than the batchsize if the datastore does not have enough records.

```
function batchout = helperReadBatch(ds,N,batchsize)
% This function is only in support of Wavelet Toolbox examples. It may
% change or be removed in a future release.
%
% batchout = readBatch(ds,batchsize) where ds is the Datastore and
%   ds is the Datastore
%   batchsize is the batchsize

kk = 1;

while(hasdata(ds)) && kk <= batchsize
    tmpRead = read(ds);
    batchout(:,kk) = tmpRead(1:N); %#ok<AGROW>
    kk = kk+1;
end

end
```

See Also
waveletScattering

More About
- "Wavelet Scattering" on page 9-2
- "Music Genre Classification Using Wavelet Time Scattering" on page 13-26
- "Wavelet Time Scattering with GPU Acceleration — Spoken Digit Recognition" on page 13-54

Wavelet Time Scattering Classification of Phonocardiogram Data

This example shows how to classify human phonocardiogram (PCG) recordings using wavelet time scattering and a support vector machine (SVM) classifier. Phonocardiograms are acoustic recordings of sounds produced by the systolic and diastolic phases of the heart. Auscultation of the heart continues to play an important diagnostic role in assessing cardiac health. Unfortunately, many areas of the world lack sufficient numbers of medical personnel trained in heart auscultation. Accordingly, it is necessary to develop reliable automated ways of interpreting phonocardiogram data.

This example uses wavelet scattering as a feature extractor for PCG classification. In wavelet scattering, data is propagated through a series of wavelet transforms, nonlinearities, and averaging to produce low-variance representations of the data. These low-variance representations are then used as inputs to a classifier. This example is a binary classification problem where each PCG recording is either "normal" or "abnormal".

Data Description

This example uses phonocardiogram (PCG) data obtained from persons with normal and abnormal cardiac function. The dataset consists of 3829 recordings, 2575 from persons with normal cardiac function and 1254 records from persons with abnormal cardiac function. Each recording is 10,000 samples long and is sampled at 2 kHz. This represents five seconds of phonocardiogram data. The dataset is constructed from the training and validation data used in the PhysioNet Computing in Cardiology Challenge 2016 [1][2].

Download Data

The first step is to download the data from the GitHub repository. To download the data, click `Clone or download` and select `Download ZIP`. Save the file `physionet_phonocardiogram-master` in a folder where you have write permission. The instructions for this example assume you have downloaded the file to your temporary directory, (`tempdir` in MATLAB™). Modify the subsequent instructions for unzipping and loading the data if you choose to download the data in folder different from `tempdir`. If you are familiar with Git, you can download the latest version of the tools (git) and obtain the data from a system command prompt using `git clone https://github.com/mathworks/physionet_phonocardiogram`.

The file `physionet_phonocardiogram-master.zip` contains

- PCG_Data.zip
- README.md

and PCG_Data.zip contains

- heartSoundData.mat
- extrafiles.mat
- Modified_physionet_data.txt
- License.txt.

`heartSoundData.mat` holds the data and class labels used in this example. The .txt file, Modified_physionet_data.txt, is required by PhysioNet's copying policy and provides the source attributions for the data as well as a description of how each signal in `heartSoundData.mat` corresponds to a file in the original PhysioNet data. `extrafiles.mat` also contains source file

attributions and is explained in the Modified_physionet_data.txt file. The only file required to run the example is `heartSoundData.mat`.

Load Data

If you followed the download instructions in the previous section, enter the following commands to unzip the two archive files.

```
unzip(fullfile(tempdir,'physionet_phonocardiogram-master.zip'),tempdir)
unzip(fullfile(tempdir,'physionet_phonocardiogram-master','PCG_Data.zip'),...
    fullfile(tempdir,'PCG_Data'))
```

After you unzip the PCG_Data.zip file, load the data into MATLAB.

```
load(fullfile(tempdir,'PCG_Data','heartSoundData.mat'))
```

`heartSoundData` is a structure array with two fields: `Data` and `Classes`. `Data` is a 10000-by-3829 matrix where each column is an PCG recording. `Classes` is a 3829-by-1 categorical array of diagnostic labels, one for each column of `Data`. Because this is a binary classification problem, the classes are "normal" and "abnormal". As previously stated, there are 2575 normal records and 1254 abnormal records. Equivalently, 67.25% of the examples in the data are from persons with normal cardiac function while 32.75% are from persons with abnormal cardiac function. You can verify this by entering:

```
summary(heartSoundData.Classes)
```

```
    normal        2575
    abnormal      1254
```

```
countcats(heartSoundData.Classes)./sum(countcats(heartSoundData.Classes))
```

```
ans = 2×1

    0.6725
    0.3275
```

Wavelet Scattering Framework

Use `waveletScattering` to construct a wavelet time scattering framework. Set the invariant scale to match the signal length. The default scattering framework has two wavelet transforms (filter banks). The first wavelet filter bank has eight wavelets per octave. The second filter bank has one wavelet per octave.

```
N = 1e4;
sn = waveletScattering('SignalLength',N,'InvarianceScale',N);
```

Create Training and Test Sets

The helper function, `partition_heartsounds`, partitions the 3829 observations so that 70% (2680) are in the training set with 1802 normal and 878 abnormal. The remaining 1149 records (773 normal and 376 abnormal) are held out in the test set for prediction. The random number generator is seeded inside of the helper function so the results are repeatable. The code for `partition_heartsounds` and all other helper functions used in this example is given in the Supporting Functions section at the end of the example.

```
[trainData, testData, trainLabels, testLabels] = ...
    partition_heartsounds(70,heartSoundData.Data,heartSoundData.Classes);
```

You can check the numbers of each class in the training and test sets.

summary(trainLabels)

```
    normal        1802
    abnormal       878
```

summary(testLabels)

```
    normal         773
    abnormal       376
```

Note that the training and test sets have been partitioned so that the proportion of "normal" and "abnormal" records in the training and test sets are the same as their proportions in the overall data. You can confirm this with the following.

countcats(trainLabels)./sum(countcats(trainLabels))

ans = *2×1*

```
    0.6724
    0.3276
```

countcats(testLabels)./sum(countcats(testLabels))

ans = *2×1*

```
    0.6728
    0.3272
```

Scattering Features

Obtain the scattering transform of all 2680 recordings in the training set. For multivariate time series, the scattering transform assumes each column is a separate signal. Use the `'log'` option to obtain the natural logarithm of the scattering coefficients.

scat_features_train = featureMatrix(sn,trainData,'Transform','log');

For the given scattering parameters, `scat_features_train` is a 340-by-5-by-2680 matrix. There are 340 scattering paths and five scattering windows for each of the 2680 signals. In order to pass this to the SVM classifier, reshape the tensor into a 13400-by-340 matrix where each row represents a single scattering window across the 340 scattering paths. The total number of rows is equal to the product of 5 and 2680 (number of recordings in the training data).

```
Nseq = size(scat_features_train,2);
scat_features_train = permute(scat_features_train,[2 3 1]);
scat_features_train = reshape(scat_features_train,...
    size(scat_features_train,1)*size(scat_features_train,2),[]);
```

Repeat the process for the test data.

```
scat_features_test = featureMatrix(sn,testData,'Transform','log');
scat_features_test = permute(scat_features_test,[2 3 1]);
scat_features_test = reshape(scat_features_test,...
    size(scat_features_test,1)*size(scat_features_test,2),[]);
```

Here we replicate the labels so that we have a label for each scattering time window.

```
[sequence_labels_train,sequence_labels_test] = ...
    createSequenceLabels_heartsounds(Nseq,trainLabels,testLabels);
```

Fit the SVM to the training data. In this example, we use a cubic polynomial kernel. After fitting the SVM to the training data, we perform a 5-fold cross-validation to estimate the generalization error on the training data.

```
rng default;
classificationSVM = fitcsvm(...
    scat_features_train, ...
    sequence_labels_train , ...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 3, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true, ...
    'ClassNames', categorical({'normal','abnormal'}));
kfoldmodel = crossval(classificationSVM, 'KFold', 5);
```

Compute the loss as a percentage and display the confusion matrix.

```
predLabels = kfoldPredict(kfoldmodel);
loss = kfoldLoss(kfoldmodel)*100;
fprintf('Loss is %2.2f percent\n',loss);
```

```
Loss is 0.79 percent
```

```
accuracy = 100-loss;
fprintf('Accuracy is %2.2f percent\n',accuracy);
```

```
Accuracy is 99.21 percent
```

```
confmatCV = confusionchart(sequence_labels_train,predLabels);
```

Note that the scattering framework results in 99.21 percent accuracy when each time window is classified separately. However, the performance is actually better than this value because we have five scattering windows per recording and the 99.21 percent accuracy is based on classifying all windows separately. In this case, use a majority vote to obtain a single class assignment per recording. The class vote corresponds to the mode of the votes for the five windows. If no unique mode is found, `helperMajorityVote` classifies that set of scattering windows as `'NoUniqueMode'` to indicate a classification error. This results in an extra column in the confusion matrix.

```
classes = categorical({'abnormal','normal'});
ClassVotes = helperMajorityVote(predLabels,trainLabels,classes);
CVaccuracy = sum(eq(ClassVotes,trainLabels))./numel(trainLabels)*100;
fprintf('The true cross-validation accuracy is %2.2f percent.\n',CVaccuracy);
```

```
The true cross-validation accuracy is 99.93 percent.
```

Display the confusion matrix for the majority vote classifications.

```
cmCV = confusionchart(ClassVotes,trainLabels);
```

The cross-validation accuracy on the training data is actually 99.93 percent. There are two normal records, which are misclassified as abnormal.

Use the SVM model fit to the training data to make class predictions on the held-out test data.

```
predTestLabels = predict(classificationSVM,scat_features_test);
```

Determine the accuracy of the predictions on the test set using a majority vote.

```
ClassVotes = helperMajorityVote(predTestLabels,testLabels,classes);
testaccuracy = sum(eq(ClassVotes,testLabels))./numel(testLabels)*100;
fprintf('The test accuracy is %2.2f percent.\n',testaccuracy);
```

```
The test accuracy is 92.25 percent.
```

```
cmTest = confusionchart(ClassVotes,testLabels);
```

Of the 1149 test records, 92.25% are correctly classified as "Normal" or "Abnormal". Of the 773 normal PCG recordings in the test set, 728 are correctly classified. Of the 376 abnormal recordings in the test set, 332 are correctly classified.

Precision, Recall, and F1 Score

In a classification task, the precision for a class is the number of correct positive results divided by the number of positive results. In other words, of all the records that the classifier assigns a given label, what proportion actually belong to the class. Recall is defined as the number of correct labels divided by the number of labels for a given class. Specifically, of all the records belonging to a class, what proportion did our classifier label as that class. In judging the accuracy your classifier, you ideally want to do well on both precision and recall. For example, suppose we had a classifier that labeled every PCG recording as abnormal. Then our recall for the abnormal class would be 100%. All records belonging to the abnormal class would be labeled abnormal. However, the precision would be low. Because our classifier labeled all records as abnormal, there would be 2575 false positives in this case for a precision of 1254/3829, or 32.75%. The F1 score is the harmonic mean of precision and recall and provides a single metric that summarizes the classifier performance in terms of both recall and precision. The helper function, `helperF1heartSounds`, computes the precision, recall, and F1 scores for the classification results on the test set and returns those results in a table.

```
PRTable = helperF1heartSounds(cmTest.NormalizedValues);
disp(PRTable)
```

```
        Precision    Recall    F1_Score
        _____    _____    _____
```

```
Abnormal      88.298      88.064      88.181
Normal        94.179      94.301      94.239
```

In this case, the F1 scores for the abnormal and normal groups confirm that our model has both good precision and recall. In binary classification it is simple to determine both precision and recall directly from the confusion matrix. To see this, plot the confusion matrix again for convenience.

```
cmTest = confusionchart(ClassVotes,testLabels);
```
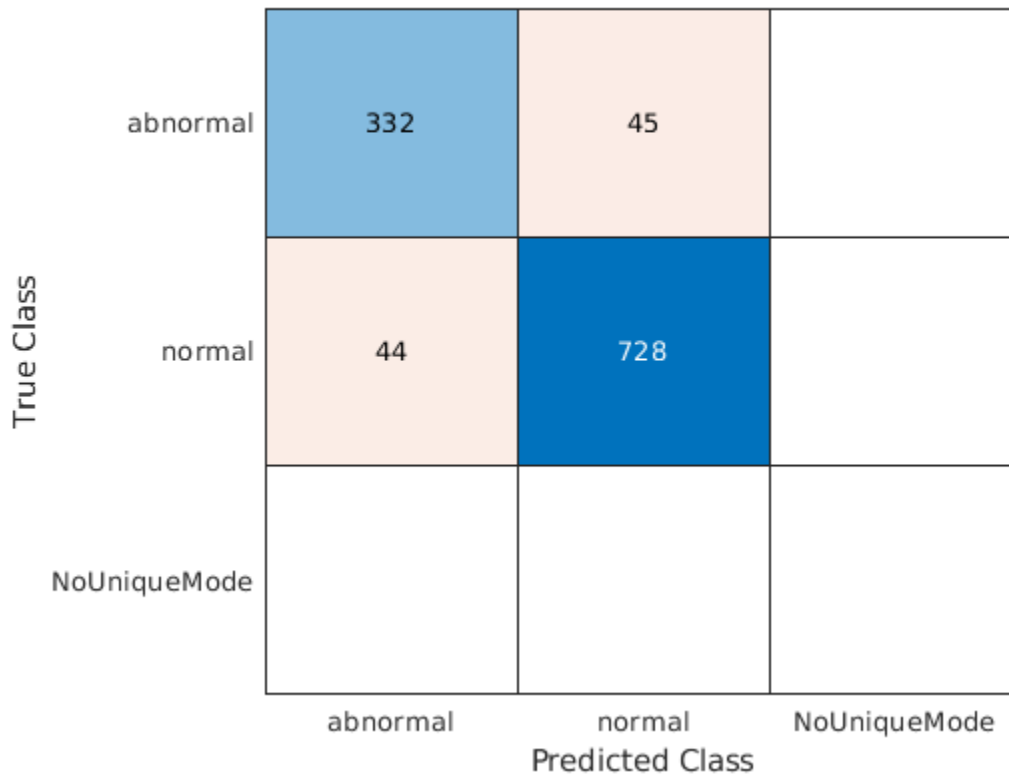

The recall for the abnormal class is the number of abnormal records identified as abnormal, which is the entry in the first row and first column of the confusion matrix divided by the sum of entries in the first row. Precision for the abnormal class is the proportion of true abnormal records in the total number identified as abnormal by the classifier. That corresponds to the entry in the first row and first column of the confusion matrix divided by the sum of entries in the first column. The F1 score is the harmonic mean of the two.

```
RecallAbnormal = cmTest.NormalizedValues(1,1)/sum(cmTest.NormalizedValues(1,:));
PrecisionAbnormal = cmTest.NormalizedValues(1,1)/sum(cmTest.NormalizedValues(:,1));
F1Abnormal = harmmean([RecallAbnormal PrecisionAbnormal]);
fprintf('RecallAbnormal = %2.3f\nPrecisionAbnormal = %2.3f\nF1Abnormal = %2.3f\n',...
    100*RecallAbnormal,100*PrecisionAbnormal,100*F1Abnormal);
```

```
RecallAbnormal = 88.064
PrecisionAbnormal = 88.298
F1Abnormal = 88.181
```

Repeat the above for the normal class.

```
RecallNormal = cmTest.NormalizedValues(2,2)/sum(cmTest.NormalizedValues(2,:));
PrecisionNormal = cmTest.NormalizedValues(2,2)/sum(cmTest.NormalizedValues(:,2));
F1Normal = harmmean([RecallNormal PrecisionNormal]);
fprintf('RecallNormal = %2.3f\nPrecisionNormal = %2.3f\nF1Normal = %2.3f\n',...
    100*RecallNormal,100*PrecisionNormal,100*F1Normal);
```

```
RecallNormal = 94.301
PrecisionNormal = 94.179
F1Normal = 94.239
```

Summary

This example used wavelet time scattering to robustly identify human phonocardiogram recordings as normal or abnormal in a binary classification problem. Wavelet scattering required only the specification of a single parameter, the length of the scale invariant, in order to produce low-variance representations of the PCG data that enabled the support vector machine classifier to accurately model the difference between the two groups. The support vector machine classifier with wavelet scattering was able to achieve superior performance in both precision and recall for both groups in spite of significantly unbalanced numbers of normal and abnormal PCG recordings in both the training and test set.

References

1 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, and H. E. Stanley. "PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals". *Circulation*. Vol. 101, No. 23, 13 June 2000, pp. e215-e220. http://circ.ahajournals.org/content/101/23/e215.full

2 Liu et al. "An open access database for the evaluation of heart sound algorithms". *Physiological Measurement*. Vol. 37, No. 12, 21 November 2016, pp. 2181-2213. https://www.ncbi.nlm.nih.gov/pubmed/27869105

Supporting Functions

partition_heartsounds creates training and test sets consisting of specified proportions of the data. The function also preserves the proportion of abnormal and normal PCG recordings in each set.

```
function [trainData, testData, trainLabels, testLabels] = partition_heartsounds(percent_train_sp
% This function is only in support of the Wavelet Time Scattering
% Classification of Phonocardiogram Data example. It may change or be
% removed in a future release.

% Labels in heart sound data are not sequential.
percent_train_split = percent_train_split/100;
% Each column is an observation
NormalData = Data(:,Labels == 'normal');
AbnormalData = Data(:,Labels == 'abnormal');
LabelsNormal = Labels(Labels == 'normal');
LabelsAbnormal = Labels(Labels == 'abnormal');
Nnormal = size(NormalData,2);
Nabnormal = size(AbnormalData,2);
num_train_normal = round(percent_train_split*Nnormal);
num_train_abnormal = round(percent_train_split*Nabnormal);
rng default;
Pnormal = randperm(Nnormal,num_train_normal);
Pabnormal = randperm(Nabnormal,num_train_abnormal);
notPnormal = setdiff(1:Nnormal,Pnormal);
notPabnormal = setdiff(1:Nabnormal,Pabnormal);
```

```
trainNormalData = NormalData(:,Pnormal);
trainNormalLabels = LabelsNormal(Pnormal);
trainAbnormalData = AbnormalData(:,Pabnormal);
trainAbnormalLabels = LabelsAbnormal(Pabnormal);
testNormalData = NormalData(:,notPnormal);
testNormalLabels = LabelsNormal(notPnormal);
testAbnormalData = AbnormalData(:,notPabnormal);
testAbnormalLabels = LabelsAbnormal(notPabnormal);
trainData = [trainNormalData trainAbnormalData];
trainData = (trainData-mean(trainData))./std(trainData,1);
trainLabels = [trainNormalLabels;  trainAbnormalLabels];
testData = [testNormalData  testAbnormalData];
testData = (testData-mean(testData))./std(testData,1);
testLabels = [testNormalLabels; testAbnormalLabels];

end
```

createSequenceLabels_heartsounds creates class labels for the wavelet time scattering sequences.

```
function [sequence_labels_train,sequence_labels_test] = createSequenceLabels_heartsounds(Nseq,tra
% This function is only in support of the Wavelet Time Scattering
% Classification of Phonocardiogram Data example. It may change or be
% removed in a future release.
Ntrain = numel(trainLabels);
trainLabels = repmat(trainLabels',Nseq,1);
sequence_labels_train = reshape(trainLabels,Nseq*Ntrain,1);
Ntest = numel(testLabels);
testLabels = repmat(testLabels',Nseq,1);
sequence_labels_test = reshape(testLabels,Ntest*Nseq,1);
end
```

helperMajorityVote implements a majority vote for a classification based on the mode. If no unique mode is present, a vote of NoUniqueMode is returned to ensure a classification error is recorded.

```
function [ClassVotes,ClassCounts] = helperMajorityVote(predLabels,origLabels,classes)
% This function is in support of ECGWaveletTimeScatteringExample. It may
% change or be removed in a future release.

% Make categorical arrays if the labels are not already categorical
predLabels = categorical(predLabels);
origLabels = categorical(origLabels);
% Expects both predLabels and origLabels to be categorical vectors
Npred = numel(predLabels);
Norig = numel(origLabels);
Nwin = Npred/Norig;
predLabels = reshape(predLabels,Nwin,Norig);
ClassCounts = countcats(predLabels);
[mxcount,idx] = max(ClassCounts);
ClassVotes = classes(idx);
% Check for any ties in the maximum values and ensure they are marked as
% error if the mode occurs more than once
modecnt = modecount(ClassCounts,mxcount);
ClassVotes(modecnt>1) = categorical({'NoUniqueMode'});
ClassVotes = ClassVotes(:);

%-------------------------------------------------------------------
```

```matlab
function modecnt = modecount(ClassCounts,mxcount)
modecnt = Inf(size(ClassCounts,2),1);
for nc = 1:size(ClassCounts,2)
    modecnt(nc) = histc(ClassCounts(:,nc),mxcount(nc));
end

end

% EOF
end
```

helperF1heartSounds calculate precision, recall, and F1 scores for the classifier results.

```matlab
function PRTable = helperF1heartSounds(confmat)
% This function is only in support of the Wavelet Time Scattering
% Classification of Phonocardiogram Data example. It may change or be
% removed in a future release.

precisionAB = confmat(1,1)/sum(confmat(:,1))*100;
precisionNR = confmat(2,2)/sum(confmat(:,2))*100 ;
recallAB = confmat(1,1)/sum(confmat(1,:))*100;
recallNR = confmat(2,2)/sum(confmat(2,:))*100;
F1AB = 2*(precisionAB*recallAB)/(precisionAB+recallAB);
F1NR = 2*(precisionNR*recallNR)/(precisionNR+recallNR);
% Construct a MATLAB Table to display the results.
PRTable = array2table([precisionAB recallAB F1AB;...
    precisionNR recallNR F1NR],...
    'VariableNames',{'Precision','Recall','F1_Score'},'RowNames',...
    {'Abnormal','Normal'});

end
```

See Also
```
waveletScattering
```

More About
- "Wavelet Scattering" on page 9-2

Wavelet Time Scattering with GPU Acceleration — Spoken Digit Recognition

This example shows how to accelerate the computation of wavelet scattering features using `gpuArray` (Parallel Computing Toolbox) and a parallelized "depth-first" version of wavelet time scattering. You must have a CUDA-enabled NVIDIA GPU with compute capability 3.0 or higher. See "GPU Support by Release" (Parallel Computing Toolbox) for details. This example uses an NVIDIA Titan XP GPU with compute capability 6.1.

This example reproduces the CPU version of the scattering transform found in "Spoken Digit Recognition with Wavelet Scattering and Deep Learning" on page 13-61.

Data

Clone or download the Free Spoken Digit Dataset (FSDD), available at `https://github.com/Jakobovski/free-spoken-digit-dataset`. FSDD is an open data set, which means that it can grow over time. This example uses the version committed on 01/29/2019 which consists of 2000 recordings of the English digits 0 through 9 obtained from four speakers. Two of the speakers in this version are native speakers of American English and two speakers are nonnative speakers of English with a Belgium French and German accent respectively. The data is sampled at 8000 Hz.

Use `audioDatastore` to manage data access and ensure random division of the recordings into training and test sets. Set the `location` property to the location of the FSDD recordings folder on your computer.

```
location = fullfile(tempdir,'free-spoken-digit-dataset','recordings');
ads = audioDatastore(location);
```

The helper function, `helpergenLabels`, defined at the end of this example, creates a categorical array of labels from the FSDD files. List the classes and the number of examples in each class.

```
ads.Labels = helpergenLabels(ads);
summary(ads.Labels)
```

```
     0      200
     1      200
     2      200
     3      200
     4      200
     5      200
     6      200
     7      200
     8      200
     9      200
```

The FSDD dataset consists of 10 balanced classes with 200 recordings each. The recordings in the FSDD are not of equal duration. Read through the FSDD files and construct a histogram of the signal lengths.

```
LenSig = zeros(numel(ads.Files),1);
nr = 1;
while hasdata(ads)
    digit = read(ads);
    LenSig(nr) = numel(digit);
    nr = nr+1;
```

```
end
reset(ads)
histogram(LenSig)
grid on
xlabel('Signal Length (Samples)')
ylabel('Frequency')
```
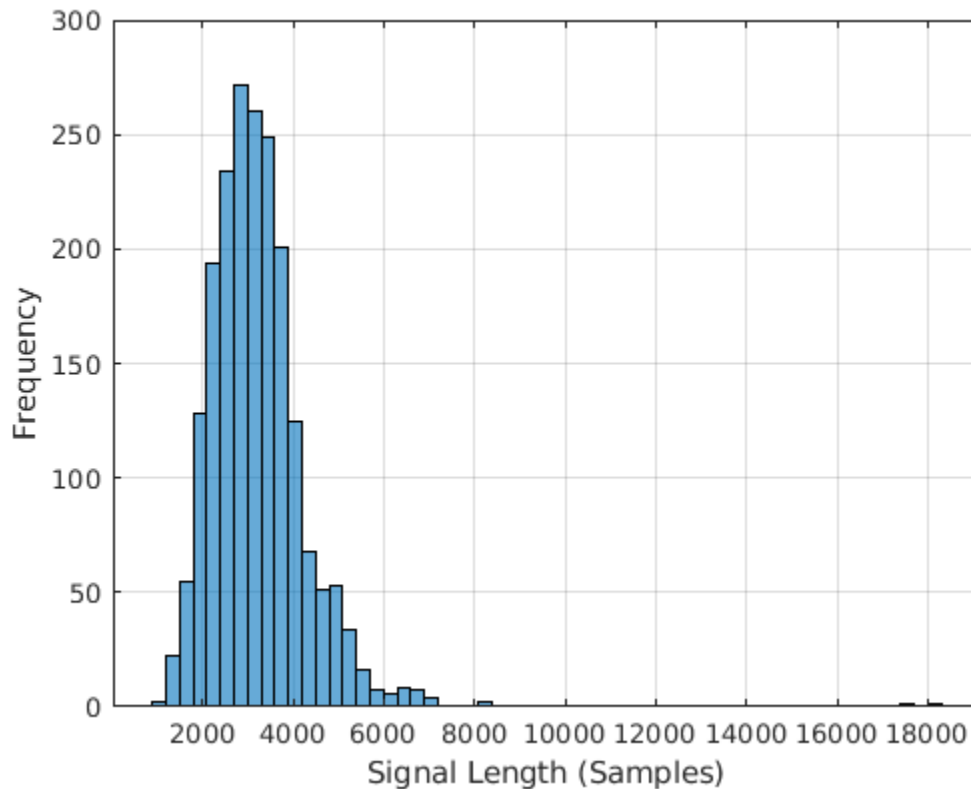

The histogram shows that the distribution of recording lengths is positively skewed. For classification, this example uses a common signal length of 8192 samples. The value 8192, a conservative choice, ensures that truncating longer recordings does not affect (cut off) the speech content. If the signal is greater than 8192 samples, or 1.024 seconds, in length, the recording is truncated to 8192 samples. If the signal is less than 8192 samples in length, the signal is symmetrically prepended and appended with zeros out to a length of 8192 samples.

Wavelet Time Scattering

Create a wavelet time scattering framework using an invariant scale of 0.22 seconds. Because the feature vectors will be created by averaging the scattering transform over all time samples, set the `OversamplingFactor` to 2. Setting the value to 2 will result in a four-fold increase in the number of scattering coefficients for each path with respect to the critically downsampled value.

```
sf = helpergpuscat1('SignalLength',8192,'InvarianceScale',0.22,...
    'SamplingFrequency',8000,'OversamplingFactor',2);
```

Split the FSDD into training and test sets. Allocate 80% of the data to the training set and retain 20% for the test set. The training data is for training the classifier based on the scattering transform. The test data is for validating the model.

```
rng default;
ads = shuffle(ads);
[adsTrain,adsTest] = splitEachLabel(ads,0.8);
countEachLabel(adsTrain)
```

ans=*10×2 table*

Label	Count
0	160
1	160
2	160
3	160
4	160
5	160
6	160
7	160
8	160
9	160

```
countEachLabel(adsTest)
```

ans=*10×2 table*

Label	Count
0	40
1	40
2	40
3	40
4	40
5	40
6	40
7	40
8	40
9	40

Form a 8192-by-1600 matrix where each column is a spoken-digit recording. The helper function `helperReadSPData` truncates or pads the data to length 8192 and normalizes each record by its maximum value.

```
Xtrain = [];
scatds_Train = transform(adsTrain,@(x)helperReadSPData(x));
while hasdata(scatds_Train)
    smat = read(scatds_Train);
    Xtrain = cat(2,Xtrain,smat);

end
```

Repeat the process for the held-out test set. The resulting matrix is 8192-by-400.

```
Xtest = [];
scatds_Test = transform(adsTest,@(x)helperReadSPData(x));
while hasdata(scatds_Test)
    smat = read(scatds_Test);
    Xtest = cat(2,Xtest,smat);
```

```
end
```

Apply the scattering transform to the training and test sets. The use of `gpuArray` with a CUDA-enabled NVIDIA GPU provides a significant acceleration. With this scattering framework, batch size, and GPU, the GPU implementation computes the scattering features approximately 8 times faster than the CPU version.

```
Strain = sf.featureMatrix(Xtrain);
Stest = sf.featureMatrix(Xtest);
```

Obtain the scattering features for the training and test sets.

```
TrainFeatures = Strain(2:end,:,:);
TrainFeatures = squeeze(mean(TrainFeatures,2))';
TestFeatures = Stest(2:end,:,:);
TestFeatures = squeeze(mean(TestFeatures,2))';
```

This example uses a support vector machine (SVM) classifier with a quadratic polynomial kernel. Fit the SVM model to the scattering features.

```
template = templateSVM(...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 2, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true);
classificationSVM = fitcecoc(...
    TrainFeatures, ...
    adsTrain.Labels, ...
    'Learners', template, ...
    'Coding', 'onevsone', ...
    'ClassNames', categorical({'0'; '1'; '2'; '3'; '4'; '5'; '6'; '7'; '8'; '9'}));
```

Use *k*-fold cross-validation to predict the generalization accuracy of the model. Split the training set into five groups for cross-validation.

```
partitionedModel = crossval(classificationSVM, 'KFold', 5);
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);
validationAccuracy = (1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError'))*100
```

```
validationAccuracy = 96.8125
```

The estimated generalization accuracy is approximately 97%. Now use the SVM model to predict the held-out test set.

```
predLabels = predict(classificationSVM,TestFeatures);
testAccuracy = sum(predLabels==adsTest.Labels)/numel(predLabels)*100
```

```
testAccuracy = 97.7500
```

The test accuracy is approximately 98% on the held-out test set.

Summarize the performance of the model on the test set with a confusion chart. Display the precision and recall for each class by using column and row summaries. The table at the bottom of the confusion chart shows the precision values for each class. The table to the right of the confusion chart shows the recall values.

```
figure('Units','normalized','Position',[0.2 0.2 0.5 0.5]);
ccscat = confusionchart(adsTest.Labels,predLabels);
ccscat.Title = 'Wavelet Scattering Classification';
ccscat.ColumnSummary = 'column-normalized';
ccscat.RowSummary = 'row-normalized';
```

Wavelet Scattering Classification

True Class	0	1	2	3	4	5	6	7	8	9		
0	39					1					97.5%	2.5%
1		40									100.0%	
2	1		39								97.5%	2.5%
3			2	38							95.0%	5.0%
4		1			39						97.5%	2.5%
5					1	38		1			95.0%	5.0%
6				2			38				95.0%	5.0%
7								40			100.0%	
8									40		100.0%	
9										40	100.0%	
	97.5%	97.6%	95.1%	95.0%	97.5%	97.4%	100.0%	97.6%	100.0%	100.0%		
	2.5%	2.4%	4.9%	5.0%	2.5%	2.6%		2.4%				
	0	1	2	3	4	5	6	7	8	9		

Predicted Class
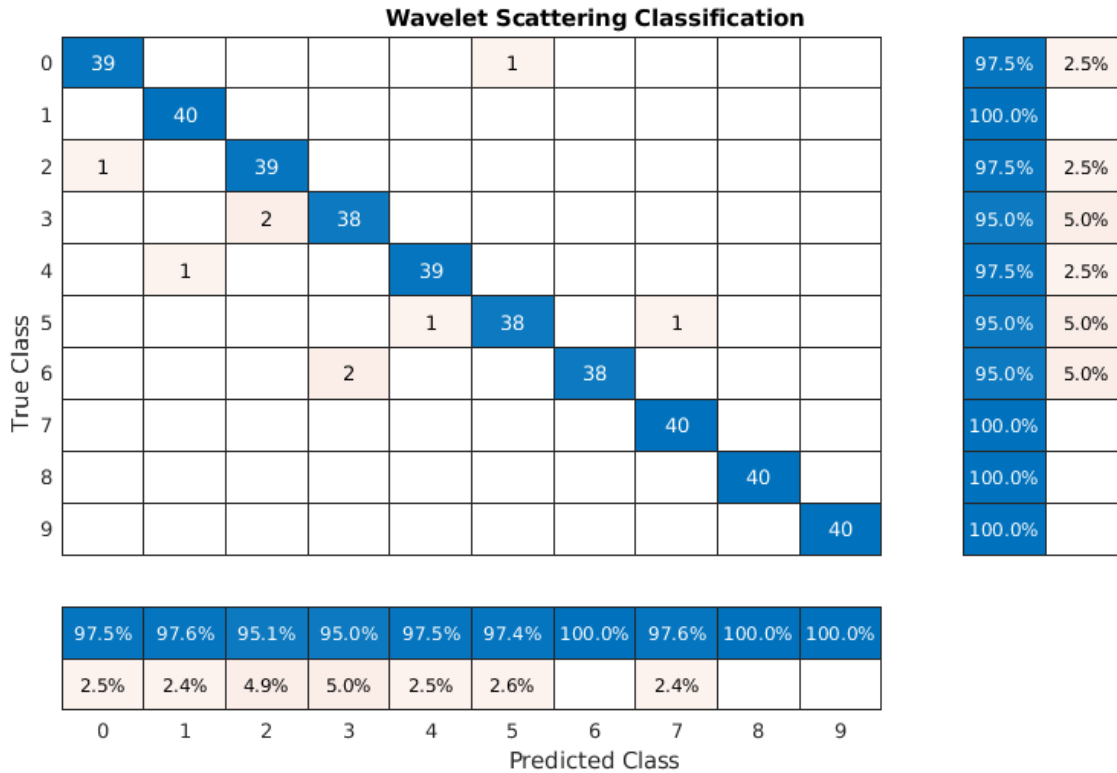
As a final example, read the first two records from the dataset, calculate the scattering features, and predict the spoken digit using the SVM trained with scattering features.

```
reset(ads);
sig1 = helperReadSPData(read(ads));
scat1 = sf.featureMatrix(sig1);
scat1 = mean(scat1(2:end,:),2)';
plab1 = predict(classificationSVM,scat1);
```

Read the next record and predict the digit.

```
sig2 = helperReadSPData(read(ads));
scat2 = sf.featureMatrix(sig2);
scat2 = mean(scat2(2:end,:),2)';
plab2 = predict(classificationSVM,scat2);

t = 0:1/8000:(8192*1/8000)-1/8000;
plot(t,[sig1 sig2])
grid on
axis tight
legend(char(plab1),char(plab2))
title('Spoken Digit Prediction - GPU')
```

Spoken Digit Prediction - GPU
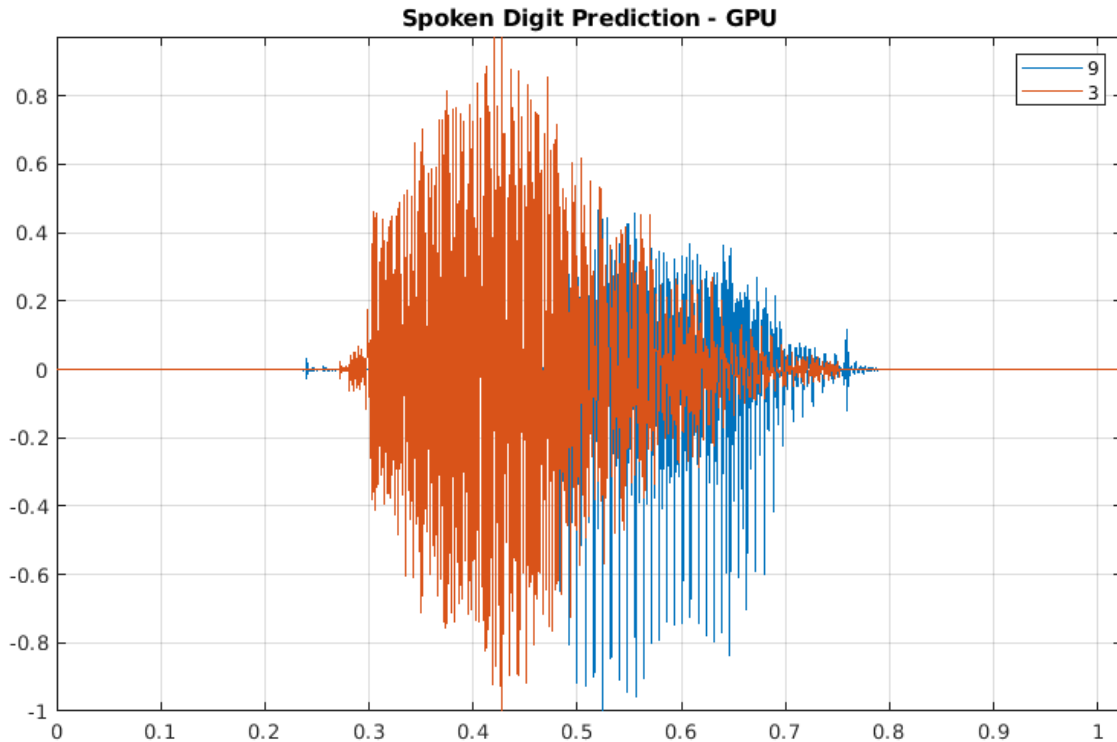
Appendix

The following helper functions are used in this example.

helpergenLabels — generates labels based on the file names in the FSDD.

```
function Labels = helpergenLabels(ads)
% This function is only for use in Wavelet Toolbox examples. It may be
% changed or removed in a future release.
tmp = cell(numel(ads.Files),1);
expression = "[0-9]+_";
for nf = 1:numel(ads.Files)
    idx = regexp(ads.Files{nf},expression);
    tmp{nf} = ads.Files{nf}(idx);
end
Labels = categorical(tmp);

end
```

helperReadSPData — Ensures that each spoken-digit recording is 8192 samples long.

```
function x = helperReadSPData(x)
% This function is only for use Wavelet Toolbox examples. It may change or
% be removed in a future release.
N = numel(x);
if N > 8192
    x = x(1:8192);
elseif N < 8192
```

```
    pad = 8192-N;
    prepad = floor(pad/2);
    postpad = ceil(pad/2);
    x = [zeros(prepad,1) ; x ; zeros(postpad,1)];
end
x = x./max(abs(x));

end
```

See Also
`waveletScattering`

More About

Spoken Digit Recognition with Wavelet Scattering and Deep Learning

This example shows how to classify spoken digits using both machine and deep learning techniques. In the example, you perform classification using wavelet time scattering with a support vector machine (SVM) and with a long short-term memory (LSTM) network. You also apply Bayesian optimization to determine suitable hyperparameters to improve the accuracy of the LSTM network. In addition, the example illustrates an approach using a deep convolutional neural network (CNN) and mel-frequency spectrograms.

Data

Clone or download the Free Spoken Digit Dataset (FSDD), available at https://github.com/Jakobovski/free-spoken-digit-dataset. FSDD is an open data set, which means that it can grow over time. This example uses the version committed on January 29, 2019, which consists of 2000 recordings in English of the digits 0 through 9 obtained from four speakers. In this version, two of the speakers are native speakers of American English, one speaker is a nonnative speaker of English with a Belgian French accent, and one speaker is a nonnative speaker of English with a German accent. The data is sampled at 8000 Hz.

Use `audioDatastore` to manage data access and ensure the random division of the recordings into training and test sets. Set the `location` property to the location of the FSDD recordings folder on your computer, for example:

```
pathToRecordingsFolder = fullfile(tempdir,'free-spoken-digit-dataset','recordings');
location = pathToRecordingsFolder;
```

Point `audioDatastore` to that location.

```
ads = audioDatastore(location);
```

The helper function `helpergenLabels` creates a categorical array of labels from the FSDD files. The source code for `helpergenLabels` is listed in the appendix. List the classes and the number of examples in each class.

```
ads.Labels = helpergenLabels(ads);
summary(ads.Labels)
```

```
     0      200
     1      200
     2      200
     3      200
     4      200
     5      200
     6      200
     7      200
     8      200
     9      200
```

The FSDD data set consists of 10 balanced classes with 200 recordings each. The recordings in the FSDD are not of equal duration. The FSDD is not prohibitively large, so read through the FSDD files and construct a histogram of the signal lengths.

```
LenSig = zeros(numel(ads.Files),1);
nr = 1;
```

```
while hasdata(ads)
    digit = read(ads);
    LenSig(nr) = numel(digit);
    nr = nr+1;
end
reset(ads)
histogram(LenSig)
grid on
xlabel('Signal Length (Samples)')
ylabel('Frequency')
```
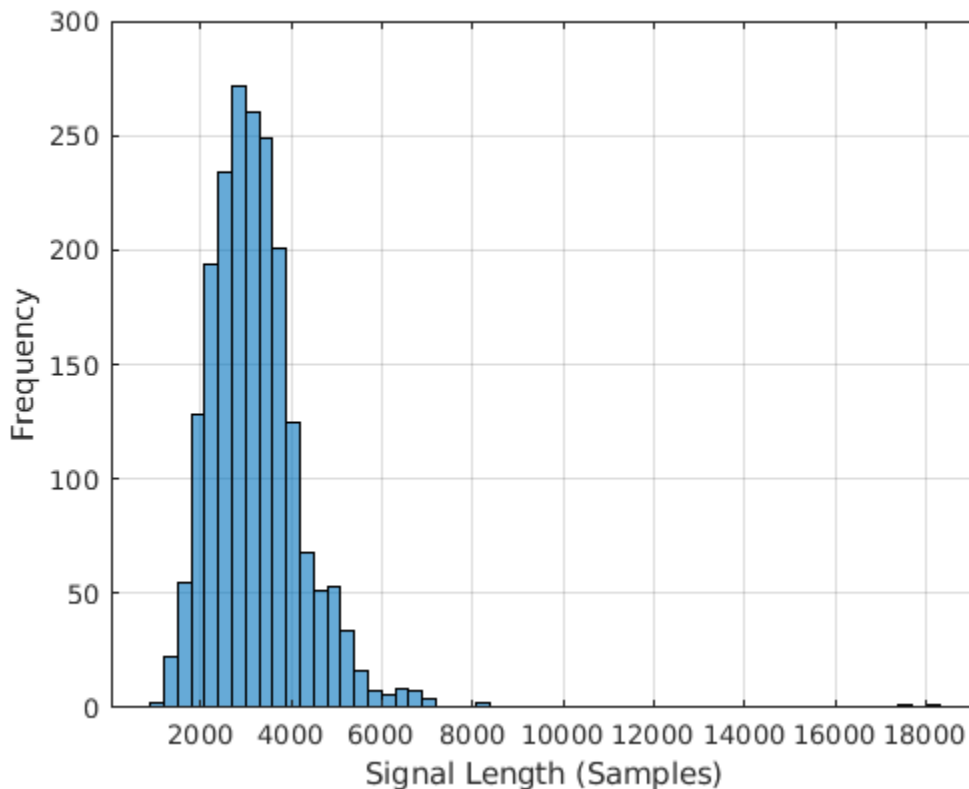

The histogram shows that the distribution of recording lengths is positively skewed. For classification, this example uses a common signal length of 8192 samples, a conservative value that ensures that truncating longer recordings does not cut off speech content. If the signal is greater than 8192 samples (1.024 seconds) in length, the recording is truncated to 8192 samples. If the signal is less than 8192 samples in length, the signal is prepadded and postpadded symmetrically with zeros out to a length of 8192 samples.

Wavelet Time Scattering

Use `waveletScattering` to create a wavelet time scattering framework using an invariant scale of 0.22 seconds. In this example, you create feature vectors by averaging the scattering transform over all time samples. To have a sufficient number of scattering coefficients per time window to average, set `OversamplingFactor` to 2 to produce a four-fold increase in the number of scattering coefficients for each path with respect to the critically downsampled value.

```
sf = waveletScattering('SignalLength',8192,'InvarianceScale',0.22,...
    'SamplingFrequency',8000,'OversamplingFactor',2);
```

Split the FSDD into training and test sets. Allocate 80% of the data to the training set and retain 20% for the test set. The training data is for training the classifier based on the scattering transform. The test data is for validating the model.

```
rng default;
ads = shuffle(ads);
[adsTrain,adsTest] = splitEachLabel(ads,0.8);
countEachLabel(adsTrain)
```

ans=*10×2 table*

Label	Count
0	160
1	160
2	160
3	160
4	160
5	160
6	160
7	160
8	160
9	160

```
countEachLabel(adsTest)
```

ans=*10×2 table*

Label	Count
0	40
1	40
2	40
3	40
4	40
5	40
6	40
7	40
8	40
9	40

The helper function `helperReadSPData` truncates or pads the data to a length of 8192 and normalizes each recording by its maximum value. The source code for `helperReadSPData` is listed in the appendix. Create an 8192-by-1600 matrix where each column is a spoken-digit recording.

```
Xtrain = [];
scatds_Train = transform(adsTrain,@(x)helperReadSPData(x));
while hasdata(scatds_Train)
    smat = read(scatds_Train);
    Xtrain = cat(2,Xtrain,smat);

end
```

Repeat the process for the test set. The resulting matrix is 8192-by-400.

```
Xtest = [];
scatds_Test = transform(adsTest,@(x)helperReadSPData(x));
while hasdata(scatds_Test)
    smat = read(scatds_Test);
    Xtest = cat(2,Xtest,smat);

end
```

Apply the wavelet scattering transform to the training and test sets.

```
Strain = sf.featureMatrix(Xtrain);
Stest = sf.featureMatrix(Xtest);
```

Obtain the mean scattering features for the training and test sets. Exclude the zeroth-order scattering coefficients.

```
TrainFeatures = Strain(2:end,:,:);
TrainFeatures = squeeze(mean(TrainFeatures,2))';
TestFeatures = Stest(2:end,:,:);
TestFeatures = squeeze(mean(TestFeatures,2))';
```

SVM Classifier

Now that the data has been reduced to a feature vector for each recording, the next step is to use these features for classifying the recordings. Create an SVM learner template with a quadratic polynomial kernel. Fit the SVM to the training data.

```
template = templateSVM(...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 2, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true);
classificationSVM = fitcecoc(...
    TrainFeatures, ...
    adsTrain.Labels, ...
    'Learners', template, ...
    'Coding', 'onevsone', ...
    'ClassNames', categorical({'0'; '1'; '2'; '3'; '4'; '5'; '6'; '7'; '8'; '9'}));
```

Use k-fold cross-validation to predict the generalization accuracy of the model based on the training data. Split the training set into five groups.

```
partitionedModel = crossval(classificationSVM, 'KFold', 5);
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);
validationAccuracy = (1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError'))*100
```

```
validationAccuracy = 96.8125
```

The estimated generalization accuracy is approximately 97%. Use the trained SVM to predict the spoken-digit classes in the test set.

```
predLabels = predict(classificationSVM,TestFeatures);
testAccuracy = sum(predLabels==adsTest.Labels)/numel(predLabels)*100
```

```
testAccuracy = 97.7500
```

Summarize the performance of the model on the test set with a confusion chart. Display the precision and recall for each class by using column and row summaries. The table at the bottom of the

confusion chart shows the precision values for each class. The table to the right of the confusion chart shows the recall values.

```
figure('Units','normalized','Position',[0.2 0.2 0.5 0.5]);
ccscat = confusionchart(adsTest.Labels,predLabels);
ccscat.Title = 'Wavelet Scattering Classification';
ccscat.ColumnSummary = 'column-normalized';
ccscat.RowSummary = 'row-normalized';
```

Wavelet Scattering Classification

True Class \ Predicted Class	0	1	2	3	4	5	6	7	8	9		
0	39					1					97.5%	2.5%
1		40									100.0%	
2	1		39								97.5%	2.5%
3			2	38							95.0%	5.0%
4		1		39							97.5%	2.5%
5				1	38		1				95.0%	5.0%
6			2			38					95.0%	5.0%
7							40				100.0%	
8								40			100.0%	
9									40		100.0%	
	97.5%	97.6%	95.1%	95.0%	97.5%	97.4%	100.0%	97.6%	100.0%	100.0%		
	2.5%	2.4%	4.9%	5.0%	2.5%	2.6%		2.4%				
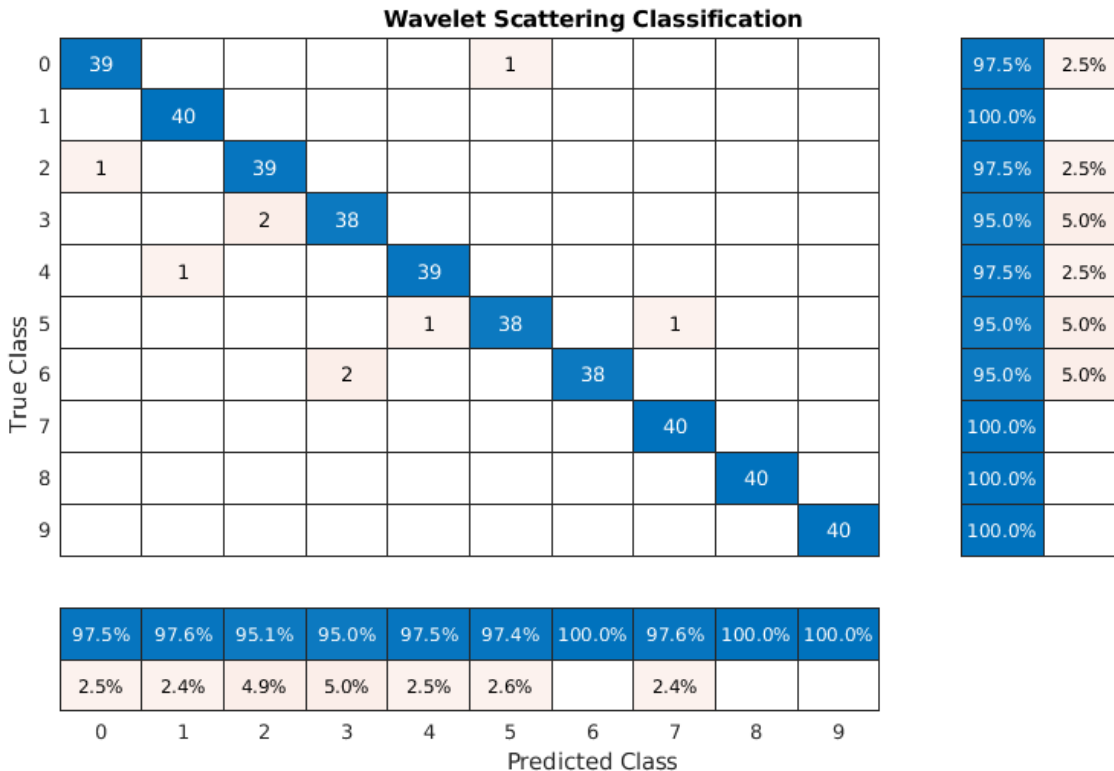
The scattering transform coupled with a SVM classifier classifies the spoken digits in the test set with an accuracy of 98% (or an error rate of 2%).

Long Short-Term Memory (LSTM) Networks

An LSTM network is a type of recurrent neural network (RNN). RNNs are neural networks that are specialized for working with sequential or temporal data such as speech data. Because the wavelet scattering coefficients are sequences, they can be used as inputs to an LSTM. By using scattering features as opposed to the raw data, you can reduce the variability that your network needs to learn.

Modify the training and testing scattering features to be used with the LSTM network. Exclude the zeroth-order scattering coefficients and convert the features to cell arrays.

```
TrainFeatures = Strain(2:end,:,:);
TrainFeatures = squeeze(num2cell(TrainFeatures,[1 2]));
TestFeatures = Stest(2:end,:,:);
TestFeatures = squeeze(num2cell(TestFeatures, [1 2]));
```

Construct a simple LSTM network with 512 hidden layers.

```matlab
[inputSize, ~] = size(TrainFeatures{1});
YTrain = adsTrain.Labels;

numHiddenUnits = 512;
numClasses = numel(unique(YTrain));

layers = [ ...
    sequenceInputLayer(inputSize)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

Set the hyperparameters. Use Adam optimization and a mini-batch size of 50. Set the maximum number of epochs to 300. Use a learning rate of 1e-4. You can turn off the training progress plot if you do not want to track the progress using plots. The training uses a GPU by default if one is available. Otherwise, it uses a CPU. For more information, see `trainingOptions` (Deep Learning Toolbox).

```matlab
maxEpochs = 300;
miniBatchSize = 50;

options = trainingOptions('adam', ...
    'InitialLearnRate',0.0001,...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'SequenceLength','shortest', ...
    'Shuffle','every-epoch',...
    'Verbose', false, ...
    'Plots','training-progress');
```

Train the network.
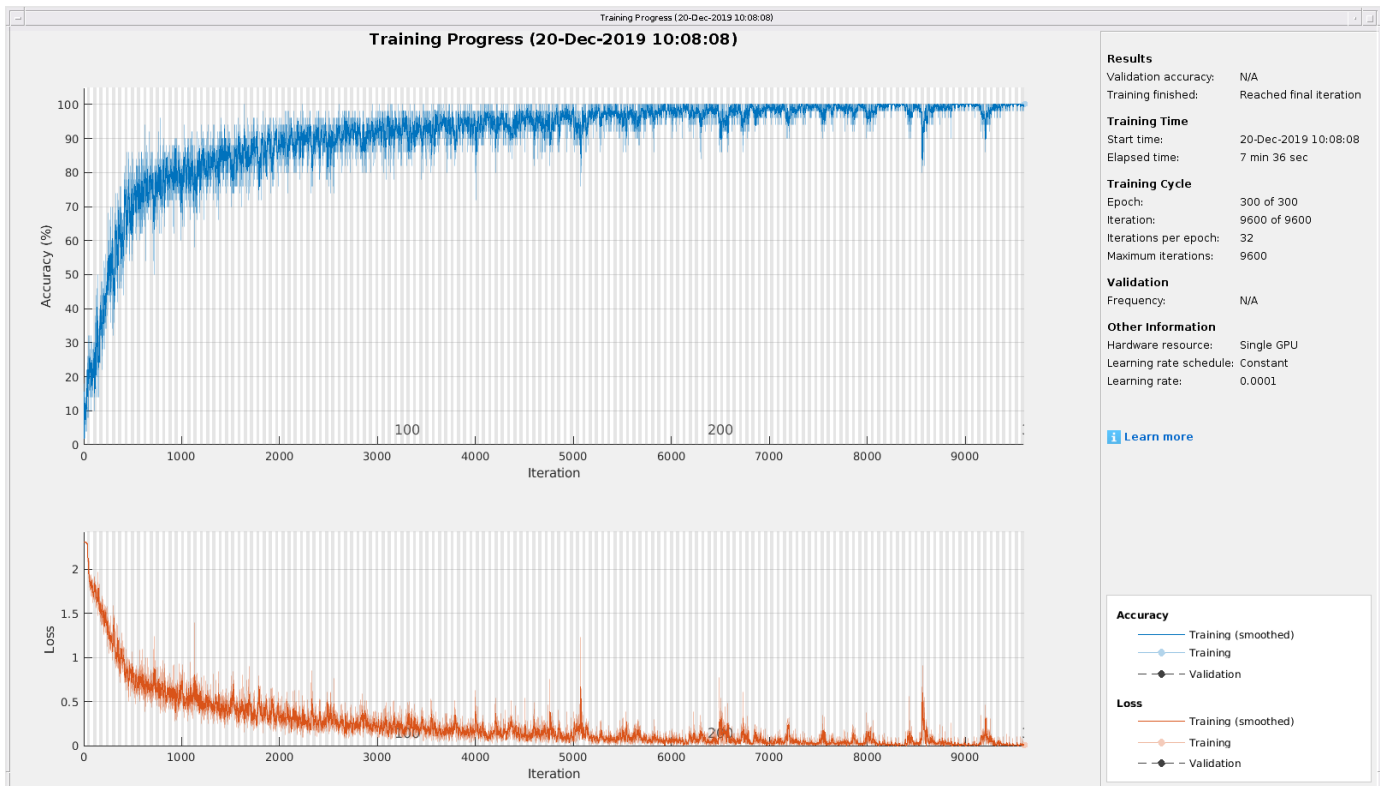
```matlab
net = trainNetwork(TrainFeatures,YTrain,layers,options);
```

```
predLabels = classify(net,TestFeatures);
testAccuracy = sum(predLabels==adsTest.Labels)/numel(predLabels)*100
```

testAccuracy = 94

Bayesian Optimization

Determining suitable hyperparameter settings is often one of the most difficult parts of training a deep network. To mitigate this, you can use Bayesian optimization. In this example, you optimize the number of hidden layers and the initial learning rate by using Bayesian techniques. Create a new directory to store the MAT-files containing information about hyperparameter settings and the network along with the corresponding error rates.

```
YTrain = adsTrain.Labels;
YTest = adsTest.Labels;

if ~exist("results/",'dir')
    mkdir results
end
```

Initialize the variables to be optimized and their value ranges. Because the number of hidden layers must be an integer, set `'type'` to `'integer'`.

```
optVars = [
    optimizableVariable('InitialLearnRate',[1e-5, 1e-1],'Transform','log')
    optimizableVariable('NumHiddenUnits',[10, 1000],'Type','integer')
    ];
```

Bayesian optimization is computationally intensive and can take several hours to finish. For the purposes of this example, set `optimizeCondition` to `false` to load predetermined optimized

13-67

hyperparameter settings. If you set `optimizeCondition` to `true`, the objective function `helperBayesOptLSTM` is minimized using Bayesian optimization. The objective function, listed in the appendix, is the error rate of the network given specific hyperparameter settings. The loaded settings are for the objective function minimum of 0.02 (2% error rate).

```
ObjFcn = helperBayesOptLSTM(TrainFeatures, YTrain, TestFeatures, YTest);

optimizeCondition = false;
if optimizeCondition
    BayesObject = bayesopt(ObjFcn,optVars,...
            'MaxObjectiveEvaluations',15,...
            'IsObjectiveDeterministic',false,...
            'UseParallel',true);
else
    load 0.02.mat
end
```

If you perform Bayesian optimization, figures similar to the following are generated to track the objective function values with the corresponding hyperparameter values and the number of iterations. You can increase the number of Bayesian optimization iterations to ensure that the global minimum of the objective function is reached.

Objective function model

Min objective vs. Number of function evaluations
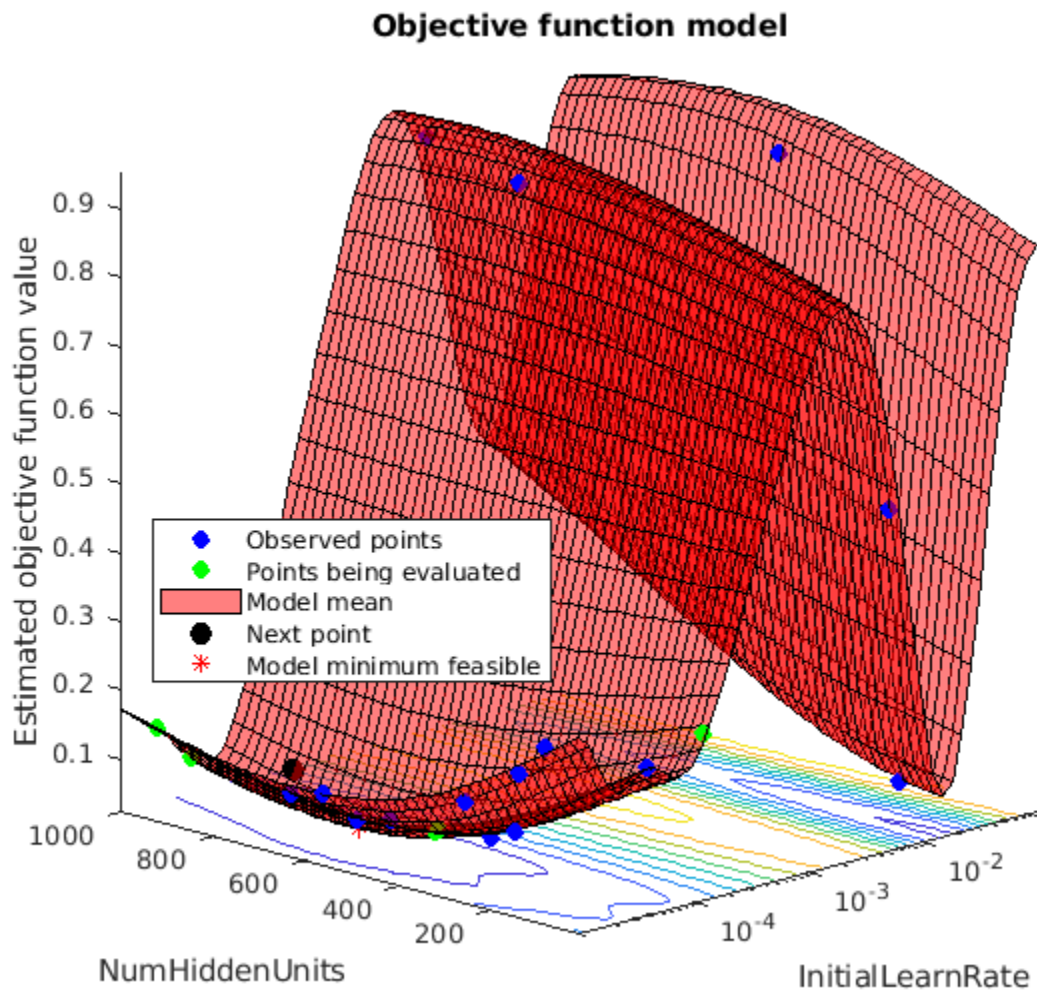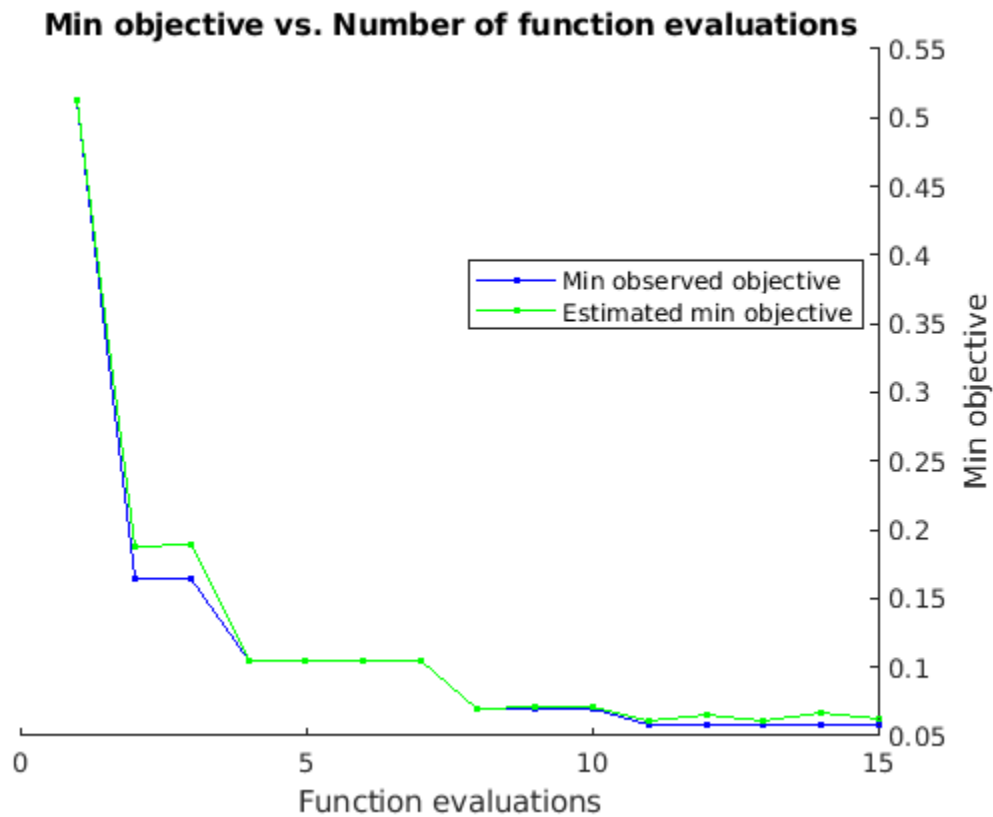
Use the optimized values for the number of hidden units and initial learning rate and retrain the network.

```
numHiddenUnits = 768;
numClasses = numel(unique(YTrain));

layers = [ ...
    sequenceInputLayer(inputSize)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];

maxEpochs = 300;
miniBatchSize = 50;

options = trainingOptions('adam', ...
    'InitialLearnRate',2.198827960269379e-04,...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'SequenceLength','shortest', ...
    'Shuffle','every-epoch',...
    'Verbose', false, ...
    'Plots','training-progress');

net = trainNetwork(TrainFeatures,YTrain,layers,options);
```
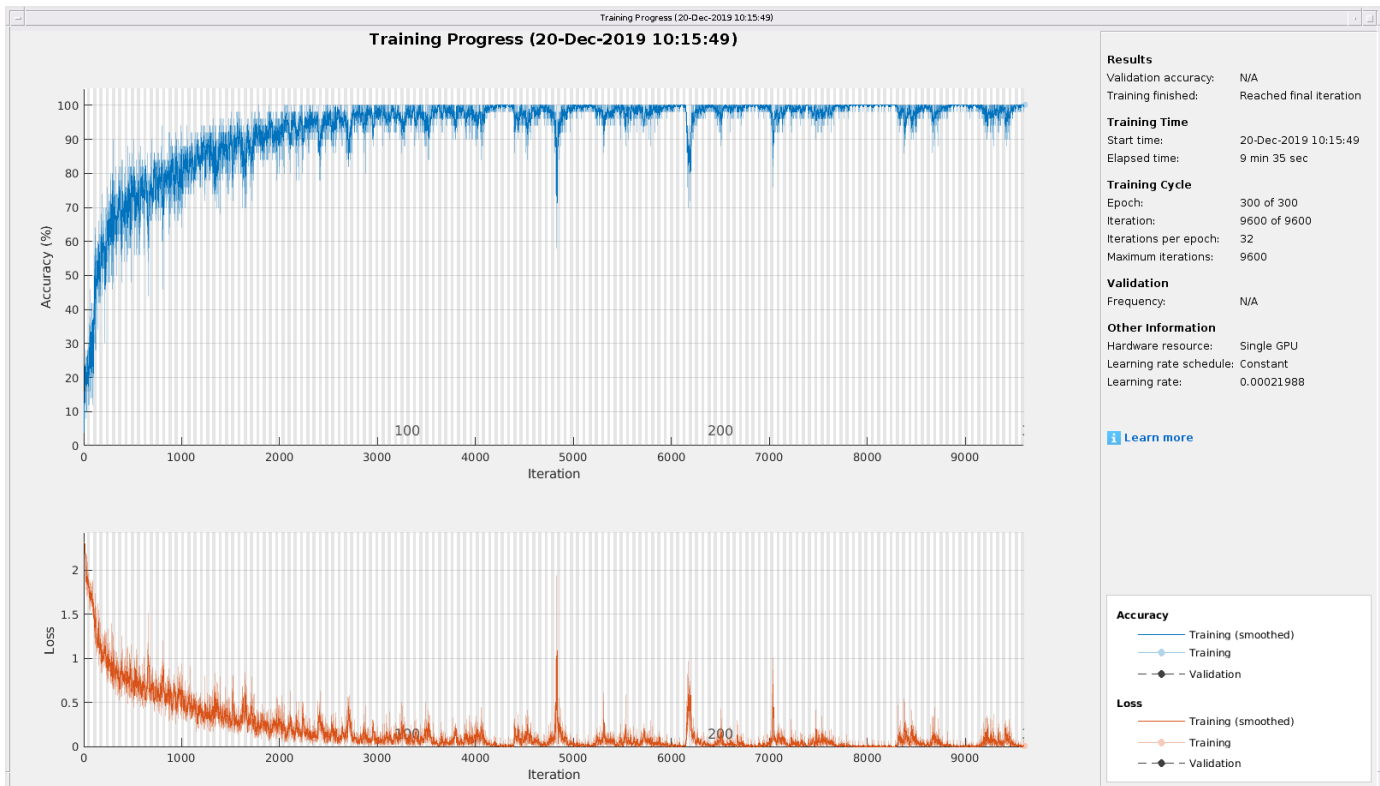
```
predLabels = classify(net,TestFeatures);
testAccuracy = sum(predLabels==adsTest.Labels)/numel(predLabels)*100
```

```
testAccuracy = 97.7500
```

As the plot shows, using Bayesian optimization yields an LSTM with a higher accuracy.

Deep Convolutional Network Using Mel-Frequency Spectrograms

As another approach to the task of spoken digit recognition, use a deep convolutional neural network (DCNN) based on mel-frequency spectrograms to classify the FSDD data set. Use the same signal truncation/padding procedure as in the scattering transform. Similarly, normalize each recording by dividing each signal sample by the maximum absolute value. For consistency, use the same training and test sets as for the scattering transform.

Set the parameters for the mel-frequency spectrograms. Use the same window, or frame, duration as in the scattering transform, 0.22 seconds. Set the hop between windows to 10 ms. Use 40 frequency bands.

```
segmentDuration = 8192*(1/8000);
frameDuration = 0.22;
hopDuration = 0.01;
numBands = 40;
```

Reset the training and test datastores.

```
reset(adsTrain);
reset(adsTest);
```

13-71

The helper function `helperspeechSpectrograms`, defined at the end of this example, uses `melSpectrogram` to obtain the mel-frequency spectrogram after standardizing the recording length and normalizing the amplitude. Use the logarithm of the mel-frequency spectrograms as the inputs to the DCNN. To avoid taking the logarithm of zero, add a small epsilon to each element.

```
epsil = 1e-6;
XTrain = helperspeechSpectrograms(adsTrain,segmentDuration,frameDuration,hopDuration,numBands);
```

```
Computing speech spectrograms...
Processed 500 files out of 1600
Processed 1000 files out of 1600
Processed 1500 files out of 1600
...done
```

```
XTrain = log10(XTrain + epsil);
```

```
XTest = helperspeechSpectrograms(adsTest,segmentDuration,frameDuration,hopDuration,numBands);
```

```
Computing speech spectrograms...
...done
```

```
XTest = log10(XTest + epsil);
```

```
YTrain = adsTrain.Labels;
YTest = adsTest.Labels;
```

Define DCNN Architecture

Construct a small DCNN as an array of layers. Use convolutional and batch normalization layers, and downsample the feature maps using max pooling layers. To reduce the possibility of the network memorizing specific features of the training data, add a small amount of dropout to the input to the last fully connected layer.

```
sz = size(XTrain);
specSize = sz(1:2);
imageSize = [specSize 1];

numClasses = numel(categories(YTrain));

dropoutProb = 0.2;
numF = 12;
layers = [
    imageInputLayer(imageSize)

    convolution2dLayer(5,numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,2*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
```

```
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2)

    dropoutLayer(dropoutProb)
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer('Classes',categories(YTrain));
    ];
```

Set the hyperparameters to use in training the network. Use a mini-batch size of 50 and a learning rate of 1e-4. Specify Adam optimization. Because the amount of data in this example is relatively small, set the execution environment to `'cpu'` for reproducibility. You can also train the network on an available GPU by setting the execution environment to either `'gpu'` or `'auto'`. For more information, see `trainingOptions` (Deep Learning Toolbox).

```
miniBatchSize = 50;
options = trainingOptions('adam', ...
    'InitialLearnRate',1e-4, ...
    'MaxEpochs',30, ...
    'MiniBatchSize',miniBatchSize, ...
    'Shuffle','every-epoch', ...
    'Plots','training-progress', ...
    'Verbose',false, ...
    'ExecutionEnvironment','cpu');
```

Train the network.

```
trainedNet = trainNetwork(XTrain,YTrain,layers,options);
```
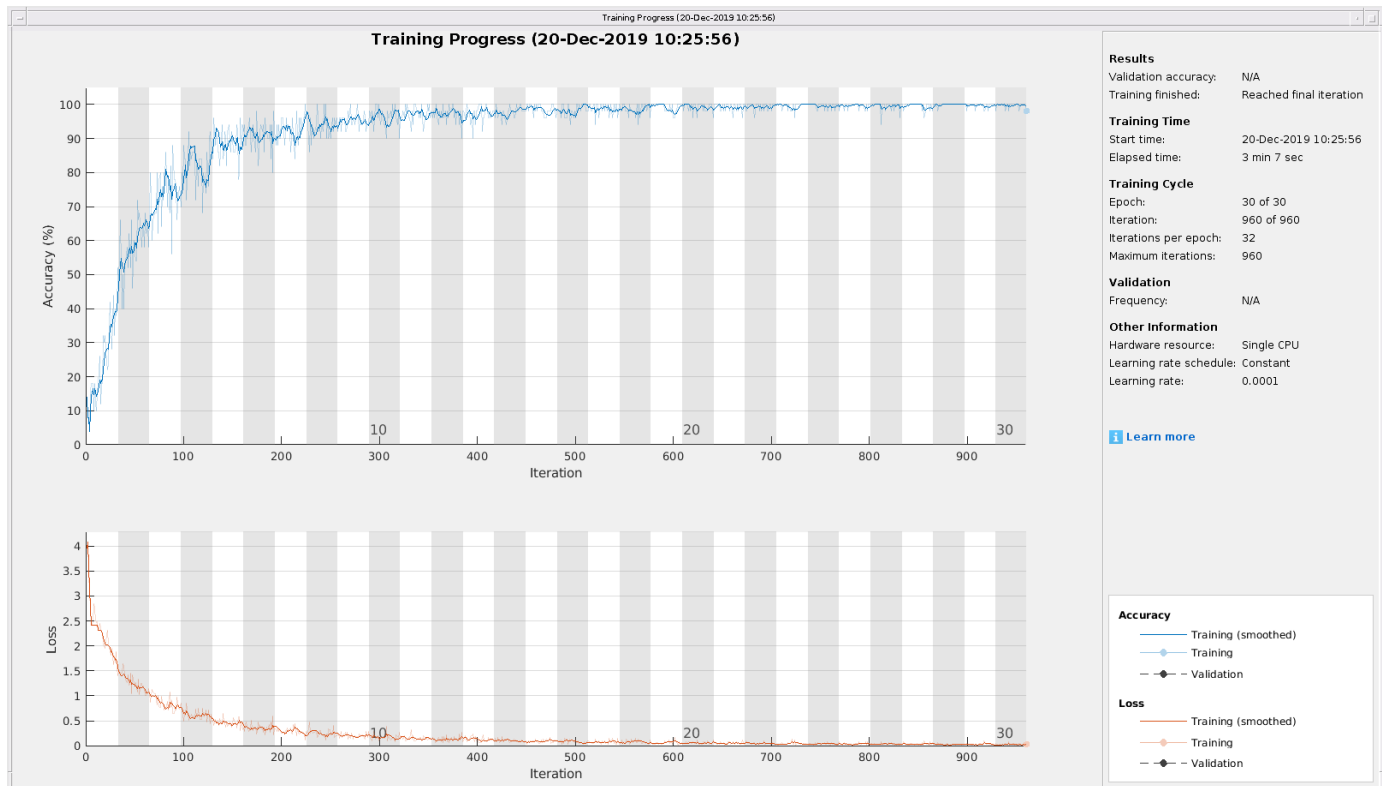
Use the trained network to predict the digit labels for the test set.

```
[Ypredicted,probs] = classify(trainedNet,XTest,'ExecutionEnvironment','CPU');
cnnAccuracy = sum(Ypredicted==YTest)/numel(YTest)*100
```

```
cnnAccuracy = 98.2500
```

Summarize the performance of the trained network on the test set with a confusion chart. Display the precision and recall for each class by using column and row summaries. The table at the bottom of the confusion chart shows the precision values. The table to the right of the confusion chart shows the recall values.

```
figure('Units','normalized','Position',[0.2 0.2 0.5 0.5]);
ccDCNN = confusionchart(YTest,Ypredicted);
ccDCNN.Title = 'Confusion Chart for DCNN';
ccDCNN.ColumnSummary = 'column-normalized';
ccDCNN.RowSummary = 'row-normalized';
```

Confusion Chart for DCNN

True Class	0	1	2	3	4	5	6	7	8	9		
0	39		1								97.5%	2.5%
1		40									100.0%	
2			40								100.0%	
3			1	38					1		95.0%	5.0%
4					40						100.0%	
5					2	38					95.0%	5.0%
6				1			39				97.5%	2.5%
7								40			100.0%	
8									40		100.0%	
9								1		39	97.5%	2.5%
	100.0%	100.0%	95.2%	97.4%	95.2%	100.0%	100.0%	97.6%	97.6%	100.0%		
			4.8%	2.6%	4.8%			2.4%	2.4%			
Predicted Class	0	1	2	3	4	5	6	7	8	9		
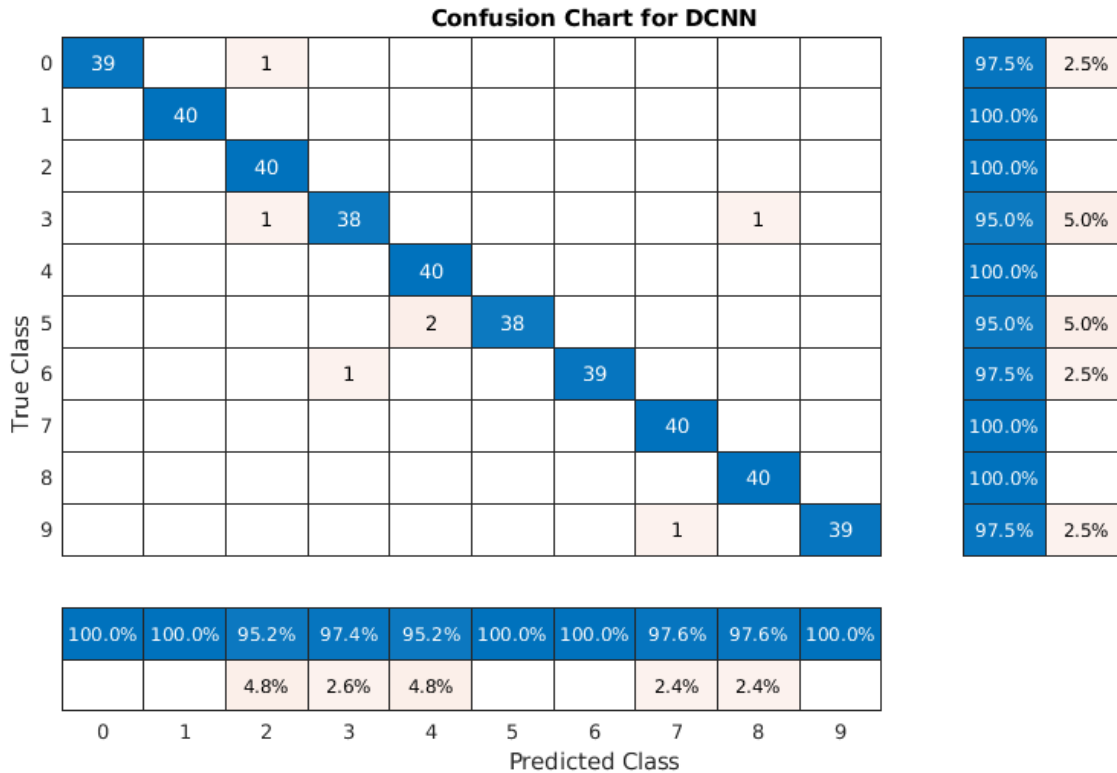
The DCNN using mel-frequency spectrograms as inputs classifies the spoken digits in the test set with an accuracy rate of approximately 98% as well.

Summary

This example shows how to use different machine and deep learning approaches for classifying spoken digits in the FSDD. The example illustrated wavelet scattering paired with both an SVM and a LSTM. Bayesian techniques were used to optimize LSTM hyperparameters. Finally, the example shows how to use a CNN with mel-frequency spectrograms.

The goal of the example is to demonstrate how to use MathWorks® tools to approach the problem in fundamentally different but complementary ways. All workflows use `audioDatastore` to manage flow of data from disk and ensure proper randomization.

All approaches used in this example performed equally well on the test set. This example is not intended as a direct comparison between the various approaches. For example, you can also use Bayesian optimization for hyperparameter selection in the CNN. An additional strategy that is useful in deep learning with small training sets like this version of the FSDD is to use data augmentation. How manipulations affect class is not always known, so data augmentation is not always feasible. However, for speech, established data augmentation strategies are available through `audioDataAugmenter`.

In the case of wavelet time scattering, there are also a number of modifications you can try. For example, you can change the invariant scale of the transform, vary the number of wavelet filters per filter bank, and try different classifiers.

Appendix: Helper Functions

```matlab
function Labels = helpergenLabels(ads)
% This function is only for use in Wavelet Toolbox examples. It may be
% changed or removed in a future release.
tmp = cell(numel(ads.Files),1);
expression = "[0-9]+_";
for nf = 1:numel(ads.Files)
    idx = regexp(ads.Files{nf},expression);
    tmp{nf} = ads.Files{nf}(idx);
end
Labels = categorical(tmp);
end

function x = helperReadSPData(x)
% This function is only for use Wavelet Toolbox examples. It may change or
% be removed in a future release.

N = numel(x);
if N > 8192
    x = x(1:8192);
elseif N < 8192
    pad = 8192-N;
    prepad = floor(pad/2);
    postpad = ceil(pad/2);
    x = [zeros(prepad,1) ; x ; zeros(postpad,1)];
end
x = x./max(abs(x));

end

function x = helperBayesOptLSTM(X_train, Y_train, X_val, Y_val)
% This function is only for use in the
% "Spoken Digit Recognition with Wavelet Scattering and Deep Learning"
% example. It may change or be removed in a future release.
x = @valErrorFun;

    function [valError,cons, fileName] = valErrorFun(optVars)
        %% LSTM Architecture
        [inputSize,~] = size(X_train{1});
        numClasses = numel(unique(Y_train));

        layers = [ ...
            sequenceInputLayer(inputSize)
            bilstmLayer(optVars.NumHiddenUnits,'OutputMode','last') % Using number of hidden laye
            fullyConnectedLayer(numClasses)
            softmaxLayer
            classificationLayer];

        % Plots not displayed during training
        options = trainingOptions('adam', ...
            'InitialLearnRate',optVars.InitialLearnRate, ... % Using initial learning rate value
            'MaxEpochs',300, ...
            'MiniBatchSize',30, ...
            'SequenceLength','shortest', ...
            'Shuffle','never', ...
            'Verbose', false);
```

```matlab
        %% Train the network
        net = trainNetwork(X_train, Y_train, layers, options);
        %% Training accuracy
        X_val_P = net.classify(X_val);
        accuracy_training  = sum(X_val_P == Y_val)./numel(Y_val);
        valError = 1 - accuracy_training;
        %% save results of network and options in a MAT file in the results folder along with the
        fileName = fullfile('results', num2str(valError) + ".mat");
        save(fileName,'net','valError','options')
        cons = [];
    end % end for inner function
end % end for outer function


function X = helperspeechSpectrograms(ads,segmentDuration,frameDuration,hopDuration,numBands)
% This function is only for use in the
% "Spoken Digit Recognition with Wavelet Scattering and Deep Learning"
% example. It may change or be removed in a future release.
%
% helperspeechSpectrograms(ads,segmentDuration,frameDuration,hopDuration,numBands)
% computes speech spectrograms for the files in the datastore ads.
% segmentDuration is the total duration of the speech clips (in seconds),
% frameDuration the duration of each spectrogram frame, hopDuration the
% time shift between each spectrogram frame, and numBands the number of
% frequency bands.
disp("Computing speech spectrograms...");

numHops = ceil((segmentDuration - frameDuration)/hopDuration);
numFiles = length(ads.Files);
X = zeros([numBands,numHops,1,numFiles],'single');

for i = 1:numFiles

    [x,info] = read(ads);
    x = normalizeAndResize(x);
    fs = info.SampleRate;
    frameLength = round(frameDuration*fs);
    hopLength = round(hopDuration*fs);

    spec = melSpectrogram(x,fs, ...
        'Window',hamming(frameLength,'periodic'), ...
        'OverlapLength',frameLength - hopLength, ...
        'FFTLength',2048, ...
        'NumBands',numBands, ...
        'FrequencyRange',[50,4000]);

    % If the spectrogram is less wide than numHops, then put spectrogram in
    % the middle of X.
    w = size(spec,2);
    left = floor((numHops-w)/2)+1;
    ind = left:left+w-1;
    X(:,ind,1,i) = spec;

    if mod(i,500) == 0
        disp("Processed " + i + " files out of " + numFiles)
    end

end
```

```matlab
    disp("...done");

end

%-------------------------------------------------------------------------
function x = normalizeAndResize(x)
% This function is only for use in the
% "Spoken Digit Recognition with Wavelet Scattering and Deep Learning"
% example. It may change or be removed in a future release.

N = numel(x);
if N > 8192
    x = x(1:8192);
elseif N < 8192
    pad = 8192-N;
    prepad = floor(pad/2);
    postpad = ceil(pad/2);
    x = [zeros(prepad,1) ; x ; zeros(postpad,1)];
end
x = x./max(abs(x));
end
```

Copyright 2018, The MathWorks, Inc.

See Also
waveletScattering

More About
- "Wavelet Scattering" on page 9-2
- "Wavelet Time Scattering with GPU Acceleration — Spoken Digit Recognition" on page 13-54

Classify Time Series Using Wavelet Analysis and Deep Learning

This example shows how to classify human electrocardiogram (ECG) signals using the continuous wavelet transform (CWT) and a deep convolutional neural network (CNN).

Training a deep CNN from scratch is computationally expensive and requires a large amount of training data. In various applications, a sufficient amount of training data is not available, and synthesizing new realistic training examples is not feasible. In these cases, leveraging existing neural networks that have been trained on large data sets for conceptually similar tasks is desirable. This leveraging of existing neural networks is called transfer learning. In this example we adapt two deep CNNs, GoogLeNet and SqueezeNet, pretrained for image recognition to classify ECG waveforms based on a time-frequency representation.

GoogLeNet and SqueezeNet are deep CNNs originally designed to classify images in 1000 categories. We reuse the network architecture of the CNN to classify ECG signals based on images from the CWT of the time series data. The data used in this example are publicly available from PhysioNet.

Data Description

In this example, you use ECG data obtained from three groups of people: persons with cardiac arrhythmia (ARR), persons with congestive heart failure (CHF), and persons with normal sinus rhythms (NSR). In total you use 162 ECG recordings from three PhysioNet databases: MIT-BIH Arrhythmia Database [3][7], MIT-BIH Normal Sinus Rhythm Database [3], and The BIDMC Congestive Heart Failure Database [1][3]. More specifically, 96 recordings from persons with arrhythmia, 30 recordings from persons with congestive heart failure, and 36 recordings from persons with normal sinus rhythms. The goal is to train a classifier to distinguish between ARR, CHF, and NSR.

Download Data

The first step is to download the data from the GitHub repository. To download the data from the website, click `Clone or download` and select `Download ZIP`. Save the file `physionet_ECG_data-master.zip` in a folder where you have write permission. The instructions for this example assume you have downloaded the file to your temporary directory, `tempdir`, in MATLAB. Modify the subsequent instructions for unzipping and loading the data if you choose to download the data in folder different from `tempdir`. If you are familiar with Git, you can download the latest version of the tools (git) and obtain the data from a system command prompt using `git clone https://github.com/mathworks/physionet_ECG_data/`.

After downloading the data from GitHub, unzip the file in your temporary directory.

```
unzip(fullfile(tempdir,'physionet_ECG_data-master.zip'),tempdir)
```

Unzipping creates the folder `physionet-ECG_data-master` in your temporary directory. This folder contains the text file `README.md` and `ECGData.zip`. The `ECGData.zip` file contains

- `ECGData.mat`
- `Modified_physionet_data.txt`
- `License.txt`

`ECGData.mat` holds the data used in this example. The text file, `Modified_physionet_data.txt`, is required by PhysioNet's copying policy and provides the source attributions for the data as well as a description of the preprocessing steps applied to each ECG recording.

Unzip `ECGData.zip` in `physionet-ECG_data-master`. Load the data file into your MATLAB workspace.

```
unzip(fullfile(tempdir,'physionet_ECG_data-master','ECGData.zip'),...
    fullfile(tempdir,'physionet_ECG_data-master'))
load(fullfile(tempdir,'physionet_ECG_data-master','ECGData.mat'))
```

`ECGData` is a structure array with two fields: `Data` and `Labels`. The `Data` field is a 162-by-65536 matrix where each row is an ECG recording sampled at 128 hertz. `Labels` is a 162-by-1 cell array of diagnostic labels, one for each row of `Data`. The three diagnostic categories are: `'ARR'`, `'CHF'`, and `'NSR'`.

To store the preprocessed data of each category, first create an ECG data directory `dataDir` inside `tempdir`. Then create three subdirectories in `'data'` named after each ECG category. The helper function `helperCreateECGDirectories` does this. `helperCreateECGDirectories` accepts `ECGData`, the name of an ECG data directory, and the name of a parent directory as input arguments. You can replace `tempdir` with another directory where you have write permission. You can find the source code for this helper function in the Supporting Functions section at the end of this example.

```
parentDir = tempdir;
dataDir = 'data';
helperCreateECGDirectories(ECGData,parentDir,dataDir)
```

Plot a representative of each ECG category. The helper function `helperPlotReps` does this. `helperPlotReps` accepts `ECGData` as input. You can find the source code for this helper function in the Supporting Functions section at the end of this example.

```
helperPlotReps(ECGData)
```
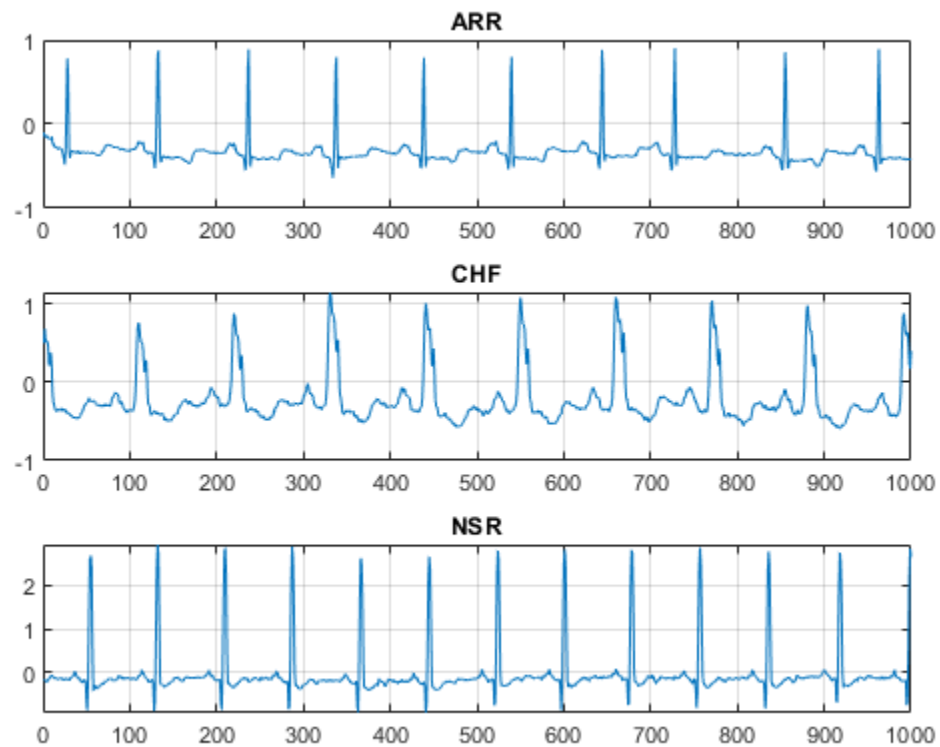
Create Time-Frequency Representations

After making the folders, create time-frequency representations of the ECG signals. These representations are called scalograms. A scalogram is the absolute value of the CWT coefficients of a signal.

To create the scalograms, precompute a CWT filter bank. Precomputing the CWT filter bank is the preferred method when obtaining the CWT of many signals using the same parameters.

Before generating the scalograms, examine one of them. Create a CWT filter bank using `cwtfilterbank` for a signal with 1000 samples. Use the filter bank to take the CWT of the first 1000 samples of the signal and obtain the scalogram from the coefficients.

```
Fs = 128;
fb = cwtfilterbank('SignalLength',1000,...
    'SamplingFrequency',Fs,...
    'VoicesPerOctave',12);
sig = ECGData.Data(1,1:1000);
[cfs,frq] = wt(fb,sig);
t = (0:999)/Fs;figure;pcolor(t,frq,abs(cfs))
set(gca,'yscale','log');shading interp;axis tight;
title('Scalogram');xlabel('Time (s)');ylabel('Frequency (Hz)')
```
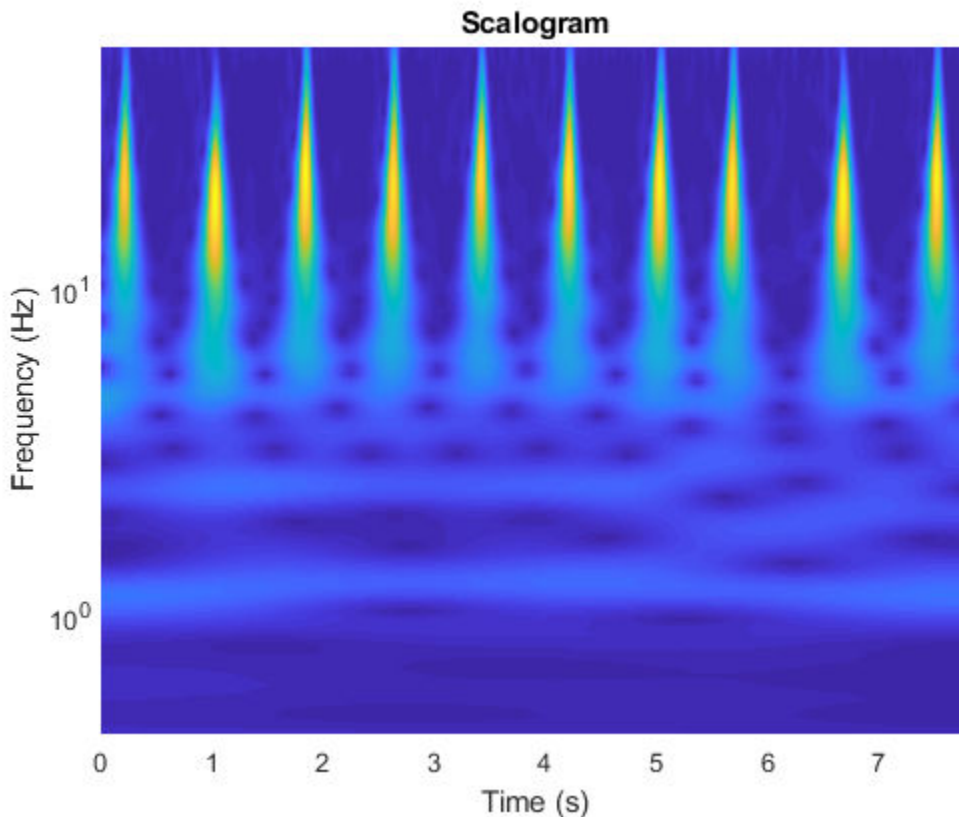
Use the helper function `helperCreateRGBfromTF` to create the scalograms as RGB images and write them to the appropriate subdirectory in `dataDir`. The source code for this helper function is in the Supporting Functions section at the end of this example. To be compatible with the GoogLeNet architecture, each RGB image is an array of size 224-by-224-by-3.

```
helperCreateRGBfromTF(ECGData,parentDir,dataDir)
```

Divide into Training and Validation Data

Load the scalogram images as an image datastore. The `imageDatastore` function automatically labels the images based on folder names and stores the data as an ImageDatastore object. An image datastore enables you to store large image data, including data that does not fit in memory, and efficiently read batches of images during training of a CNN.

```
allImages = imageDatastore(fullfile(parentDir,dataDir),...
    'IncludeSubfolders',true,...
    'LabelSource','foldernames');
```

Randomly divide the images into two groups, one for training and the other for validation. Use 80% of the images for training, and the remainder for validation. For purposes of reproducibility, we set the random seed to the default value.

```
rng default
[imgsTrain,imgsValidation] = splitEachLabel(allImages,0.8,'randomized');
disp(['Number of training images: ',num2str(numel(imgsTrain.Files))]);
```

```
Number of training images: 130
```

```
disp(['Number of validation images: ',num2str(numel(imgsValidation.Files))]);
```

```
Number of validation images: 32
```

GoogLeNet

Load

Load the pretrained GoogLeNet neural network. If Deep Learning Toolbox™ Model *for GoogLeNet Network* support package is not installed, the software provides a link to the required support package in the Add-On Explorer. To install the support package, click the link, and then click **Install**.

```
net = googlenet;
```

Extract and display the layer graph from the network.

```
lgraph = layerGraph(net);
numberOfLayers = numel(lgraph.Layers);
figure('Units','normalized','Position',[0.1 0.1 0.8 0.8]);
plot(lgraph)
title(['GoogLeNet Layer Graph: ',num2str(numberOfLayers),' Layers']);
```
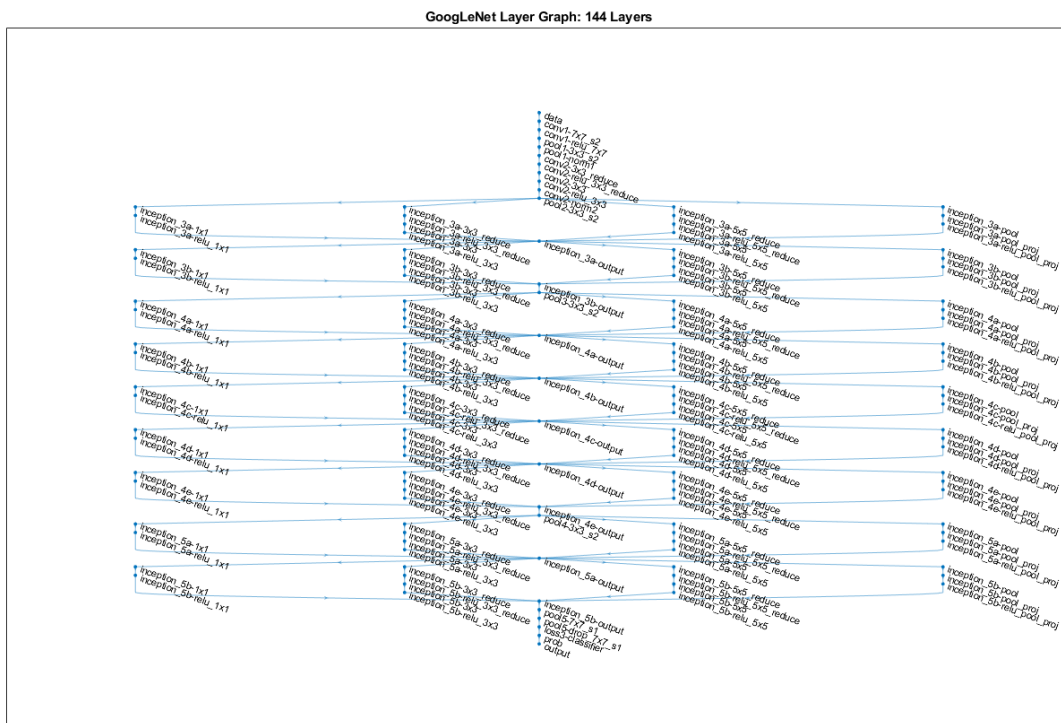

Inspect the first element of the network Layers property. Confirm that GoogLeNet requires RGB images of size 224-by-224-by-3.

```
net.Layers(1)
```

```
ans =
  ImageInputLayer with properties:

                Name: 'data'
           InputSize: [224 224 3]

   Hyperparameters
    DataAugmentation: 'none'
       Normalization: 'zerocenter'
                Mean: [224×224×3 single]
```

Modify GoogLeNet Network Parameters

Each layer in the network architecture can be considered a filter. The earlier layers identify more common features of images, such as blobs, edges, and colors. Subsequent layers focus on more specific features in order to differentiate categories. GoogLeNet is pretrained to classify images into 1000 object categories. You must retrain GoogLeNet for our ECG classification problem.

To prevent overfitting, a dropout layer is used. A dropout layer randomly sets input elements to zero with a given probability. See `dropoutLayer` (Deep Learning Toolbox) for more information. The default probability is 0.5. Replace the final dropout layer in the network, `'pool5-drop_7x7_s1'`, with a dropout layer of probability 0.6.

```
newDropoutLayer = dropoutLayer(0.6,'Name','new_Dropout');
lgraph = replaceLayer(lgraph,'pool5-drop_7x7_s1',newDropoutLayer);
```

The convolutional layers of the network extract image features that the last learnable layer and final classification layer use to classify the input image. These two layers, `'loss3-classifier'` and `'output'` in GoogLeNet, contain information on how to combine the features that the network extracts into class probabilities, a loss value, and predicted labels. To retrain GoogLeNet to classify the RGB images, replace these two layers with new layers adapted to the data.

Replace the fully connected layer `'loss3-classifier'` with a new fully connected layer with the number of filters equal to the number of classes. To learn faster in the new layers than in the transferred layers, increase the learning rate factors of the fully connected layer.

```
numClasses = numel(categories(imgsTrain.Labels));
newConnectedLayer = fullyConnectedLayer(numClasses,'Name','new_fc',...
    'WeightLearnRateFactor',5,'BiasLearnRateFactor',5);
lgraph = replaceLayer(lgraph,'loss3-classifier',newConnectedLayer);
```

The classification layer specifies the output classes of the network. Replace the classification layer with a new one without class labels. `trainNetwork` automatically sets the output classes of the layer at training time.

```
newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,'output',newClassLayer);
```

Set Training Options and Train GoogLeNet

Training a neural network is an iterative process that involves minimizing a loss function. To minimize the loss function, a gradient descent algorithm is used. In each iteration, the gradient of the loss function is evaluated and the descent algorithm weights are updated.

Training can be tuned by setting various options. `InitialLearnRate` specifies the initial step size in the direction of the negative gradient of the loss function. `MiniBatchSize` specifies how large of a

subset of the training set to use in each iteration. One epoch is a full pass of the training algorithm over the entire training set. MaxEpochs specifies the maximum number of epochs to use for training. Choosing the right number of epochs is not a trivial task. Decreasing the number of epochs has the effect of underfitting the model, and increasing the number of epochs results in overfitting.

Use the trainingOptions (Deep Learning Toolbox) function to specify the training options. Set MiniBatchSize to 10, MaxEpochs to 10, and InitialLearnRate to 0.0001. Visualize training progress by setting Plots to training-progress. Use the stochastic gradient descent with momentum optimizer. By default, training is done on a GPU if one is available (requires Parallel Computing Toolbox™ and a CUDA® enabled GPU with compute capability 3.0 or higher). For purposes of reproducibility, set ExecutionEnvironment to cpu so that trainNetwork used the CPU. Set the random seed to the default value. Run times will be faster if you are able to use a GPU.

```
options = trainingOptions('sgdm',...
    'MiniBatchSize',15,...
    'MaxEpochs',20,...
    'InitialLearnRate',1e-4,...
    'ValidationData',imgsValidation,...
    'ValidationFrequency',10,...
    'Verbose',1,...
    'ExecutionEnvironment','cpu',...
    'Plots','training-progress');
rng default
```

Train the network. The training process usually takes 1-5 minutes on a desktop CPU. The command window displays training information during the run. Results include epoch number, iteration number, time elapsed, mini-batch accuracy, validation accuracy, and loss function value for the validation data.

```
trainedGN = trainNetwork(imgsTrain,lgraph,options);
```
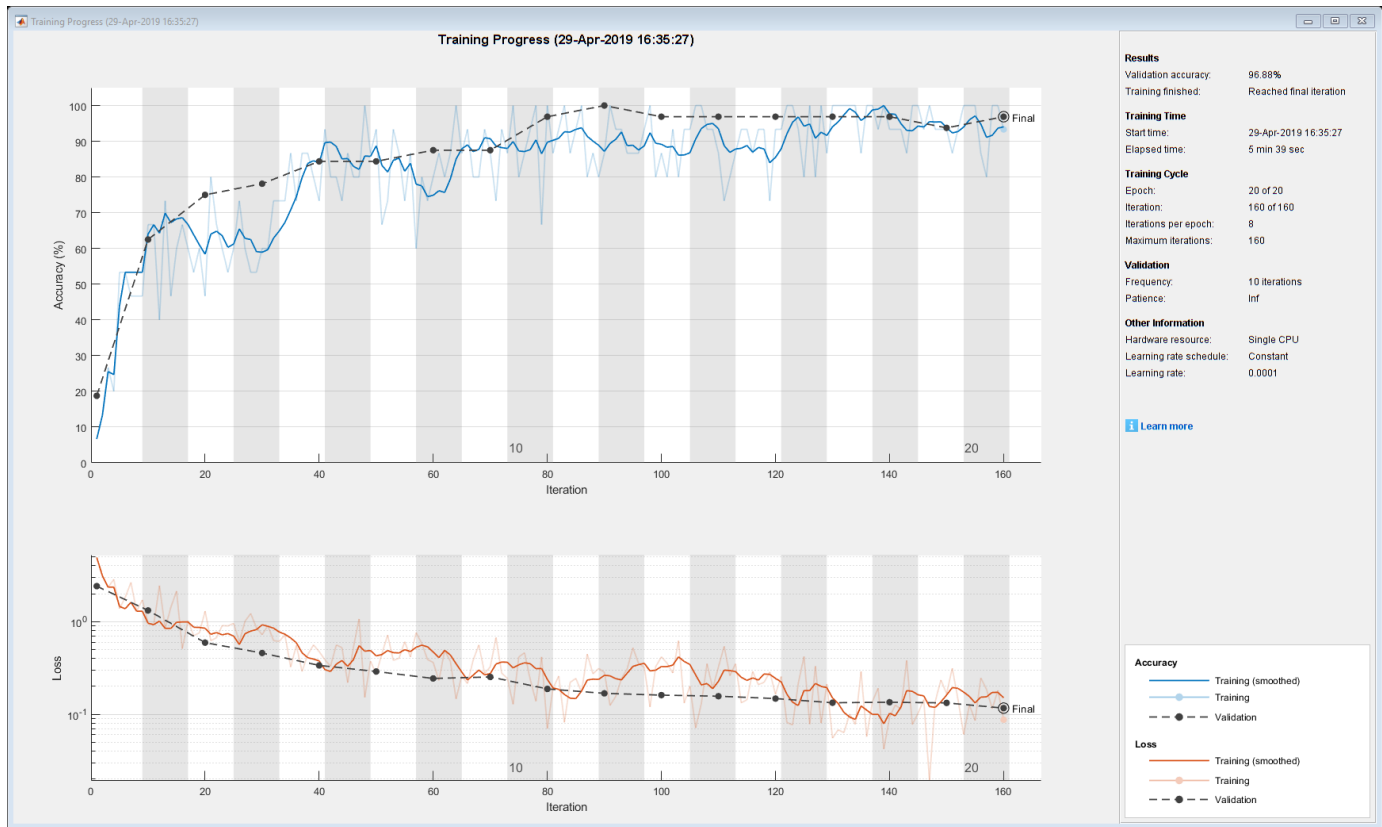
Initializing input data normalization.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validati Loss
1	1	00:00:03	6.67%	18.75%	4.9207	2.4
2	10	00:00:23	66.67%	62.50%	0.9589	1.1
3	20	00:00:43	46.67%	75.00%	1.2973	0.5
4	30	00:01:04	60.00%	78.13%	0.7219	0.4
5	40	00:01:25	73.33%	84.38%	0.4750	0.3
7	50	00:01:46	93.33%	84.38%	0.2714	0.2
8	60	00:02:07	80.00%	87.50%	0.3617	0.2
9	70	00:02:29	86.67%	87.50%	0.3246	0.2
10	80	00:02:50	100.00%	96.88%	0.0701	0.1
12	90	00:03:11	86.67%	100.00%	0.2836	0.1
13	100	00:03:32	86.67%	96.88%	0.4160	0.1
14	110	00:03:53	86.67%	96.88%	0.3237	0.1
15	120	00:04:14	93.33%	96.88%	0.1646	0.1
17	130	00:04:35	100.00%	96.88%	0.0551	0.1
18	140	00:04:57	93.33%	96.88%	0.0927	0.1
19	150	00:05:18	93.33%	93.75%	0.1666	0.1
20	160	00:05:39	93.33%	96.88%	0.0873	0.1

Inspect the last layer of the trained network. Confirm the Classification Output layer includes the three classes.

```
trainedGN.Layers(end)
```

```
ans =
  ClassificationOutputLayer with properties:

           Name: 'new_classoutput'
        Classes: [ARR    CHF    NSR]
     OutputSize: 3

   Hyperparameters
    LossFunction: 'crossentropyex'
```

Evaluate GoogLeNet Accuracy

Evaluate the network using the validation data.

```
[YPred,probs] = classify(trainedGN,imgsValidation);
accuracy = mean(YPred==imgsValidation.Labels);
disp(['GoogLeNet Accuracy: ',num2str(100*accuracy),'%'])
```

```
GoogLeNet Accuracy: 96.875%
```

The accuracy is identical to the validation accuracy reported on the training visualization figure. The scalograms were split into training and validation collections. Both collections were used to train GoogLeNet. The ideal way to evaluate the result of the training is to have the network classify data it has not seen. Since there is an insufficient amount of data to divide into training, validation, and testing, we treat the computed validation accuracy as the network accuracy.

Explore GoogLeNet Activations

Each layer of a CNN produces a response, or activation, to an input image. However, there are only a few layers within a CNN that are suitable for image feature extraction. The layers at the beginning of the network capture basic image features, such as edges and blobs. To see this, visualize the network filter weights from the first convolutional layer. There are 64 individual sets of weights in the first layer.

```
wghts = trainedGN.Layers(2).Weights;
wghts = rescale(wghts);
wghts = imresize(wghts,5);
figure
montage(wghts)
title('First Convolutional Layer Weights')
```

First Convolutional Layer Weights
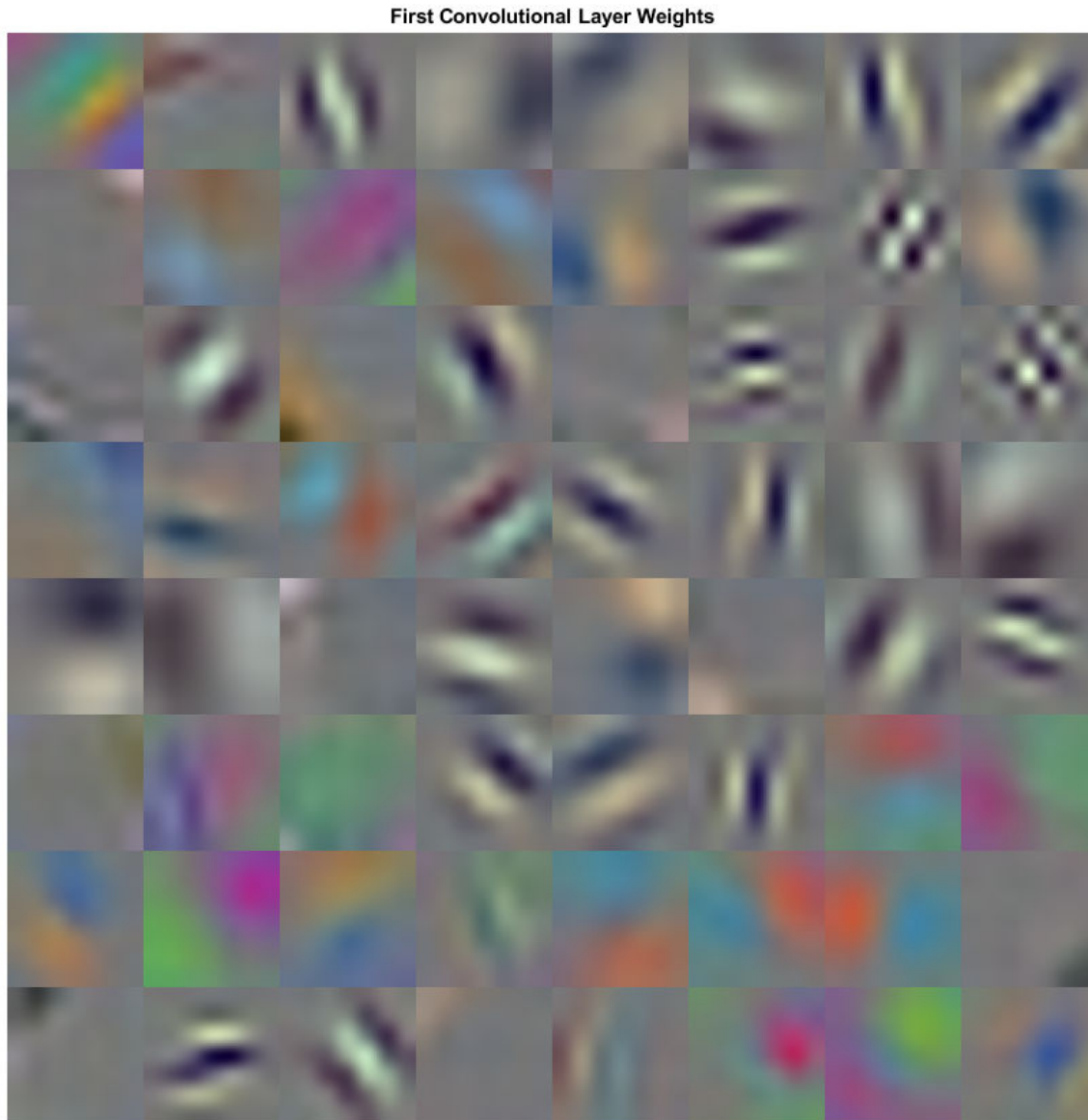
You can examine the activations and discover which features GoogLeNet learns by comparing areas of activation with the original image. For more information, see "Visualize Activations of a Convolutional Neural Network" (Deep Learning Toolbox) and "Visualize Features of a Convolutional Neural Network" (Deep Learning Toolbox).

Examine which areas in the convolutional layers activate on an image from the ARR class. Compare with the corresponding areas in the original image. Each layer of a convolutional neural network consists of many 2-D arrays called *channels*. Pass the image through the network and examine the output activations of the first convolutional layer, `conv1-7x7_s2`.

```
convLayer = 'conv1-7x7_s2';
```

```
imgClass = 'ARR';
imgName = 'ARR_10.jpg';
imarr = imread(fullfile(parentDir,dataDir,imgClass,imgName));

trainingFeaturesARR = activations(trainedGN,imarr,convLayer);
sz = size(trainingFeaturesARR);
trainingFeaturesARR = reshape(trainingFeaturesARR,[sz(1) sz(2) 1 sz(3)]);
figure
montage(rescale(trainingFeaturesARR),'Size',[8 8])
title([imgClass,' Activations'])
```


ARR Activations
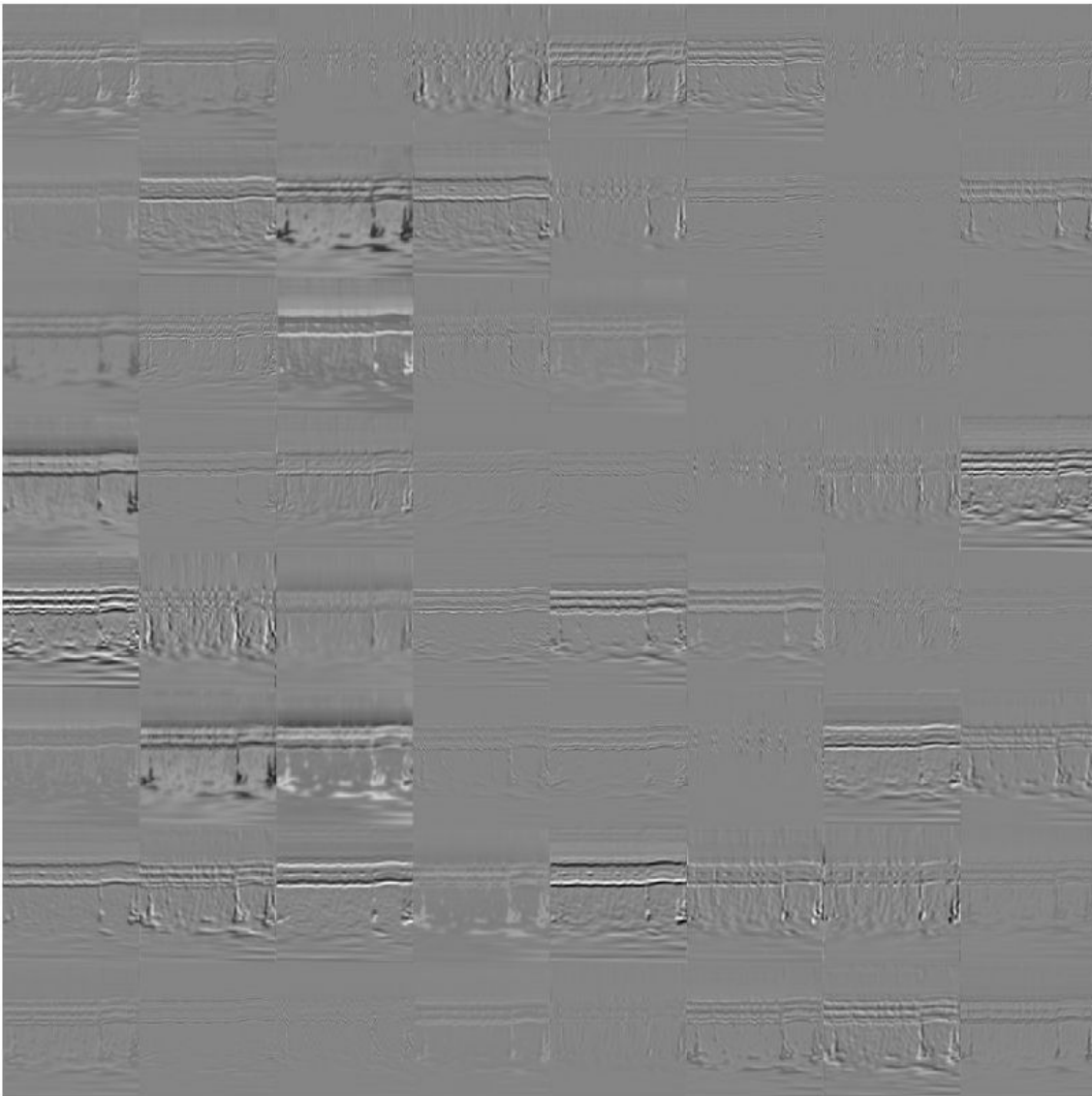
Find the strongest channel for this image. Compare the strongest channel with the original image.

```
imgSize = size(imarr);
imgSize = imgSize(1:2);
[~,maxValueIndex] = max(max(max(trainingFeaturesARR)));
arrMax = trainingFeaturesARR(:,:,:,maxValueIndex);
arrMax = rescale(arrMax);
arrMax = imresize(arrMax,imgSize);
figure;
imshowpair(imarr,arrMax,'montage')
title(['Strongest ',imgClass,' Channel: ',num2str(maxValueIndex)])
```

Strongest ARR Channel: 53
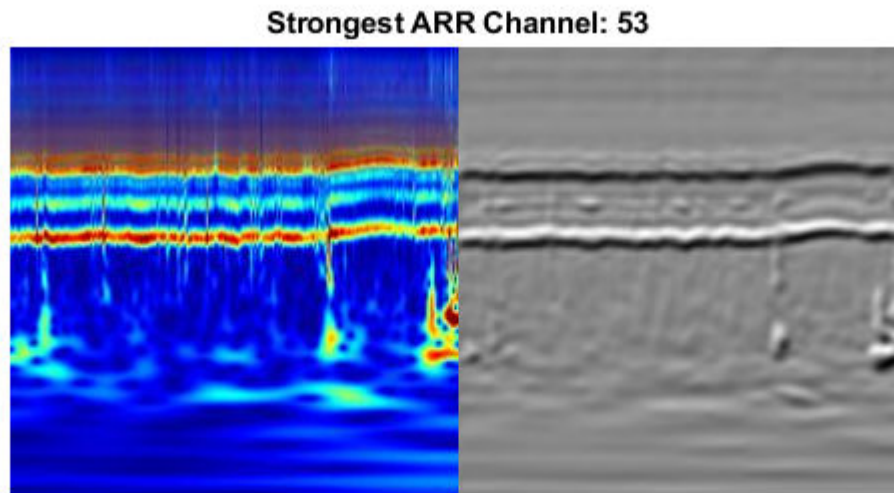
SqueezeNet

SqueezeNet is a deep CNN whose architecture supports images of size 227-by-227-by-3. Even though the image dimensions are different for GoogLeNet, you do not have to generate new RGB images at the SqueezeNet dimensions. You can use the original RGB images.

Load

Load the pretrained SqueezeNet neural network. If Deep Learning Toolbox™ Model *for SqueezeNet Network* support package is not installed, the software provides a link to the required support package in the Add-On Explorer. To install the support package, click the link, and then click **Install**.

```
sqz = squeezenet;
```

Extract the layer graph from the network. Confirm SqueezeNet has fewer layers than GoogLeNet. Also confirm that SqueezeNet is configured for images of size 227-by-227-by-3

```
lgraphSqz = layerGraph(sqz);
disp(['Number of Layers: ',num2str(numel(lgraphSqz.Layers))])
```

```
Number of Layers: 68
```

```
disp(lgraphSqz.Layers(1).InputSize)
```

```
   227   227     3
```

Modify SqueezeNet Network Parameters

To retrain SqueezeNet to classify new images, make changes similar to those made for GoogLeNet.

Inspect the last six network layers.

```
lgraphSqz.Layers(end-5:end)
```

```
ans =
  6x1 Layer array with layers:

     1   'drop9'                             Dropout                50% dropout
     2   'conv10'                            Convolution            1000 1x1x512 convolutions w:
     3   'relu_conv10'                       ReLU                   ReLU
     4   'pool10'                            Average Pooling        14x14 average pooling with :
     5   'prob'                              Softmax                softmax
     6   'ClassificationLayer_predictions'   Classification Output  crossentropyex with 'tench'
```

Replace the `'drop9'` layer, the last dropout layer in the network, with a dropout layer of probability 0.6.

```
tmpLayer = lgraphSqz.Layers(end-5);
newDropoutLayer = dropoutLayer(0.6,'Name','new_dropout');
lgraphSqz = replaceLayer(lgraphSqz,tmpLayer.Name,newDropoutLayer);
```

Unlike GoogLeNet, the last learnable layer in SqueezeNet is a 1-by-1 convolutional layer, `'conv10'`, and not a fully connected layer. Replace the `'conv10'` layer with a new convolutional layer with the number of filters equal to the number of classes. As was done with GoogLeNet, increase the learning rate factors of the new layer.

```
numClasses = numel(categories(imgsTrain.Labels));
tmpLayer = lgraphSqz.Layers(end-4);
newLearnableLayer = convolution2dLayer(1,numClasses, ...
        'Name','new_conv', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);
lgraphSqz = replaceLayer(lgraphSqz,tmpLayer.Name,newLearnableLayer);
```

Replace the classification layer with a new one without class labels.

```
tmpLayer = lgraphSqz.Layers(end);
newClassLayer = classificationLayer('Name','new_classoutput');
lgraphSqz = replaceLayer(lgraphSqz,tmpLayer.Name,newClassLayer);
```

Inspect the last six layers of the network. Confirm the dropout, convolutional, and output layers have been changed.

```
lgraphSqz.Layers(63:68)
```

```
ans =
  6x1 Layer array with layers:

     1   'new_dropout'       Dropout                60% dropout
     2   'new_conv'          Convolution            3 1x1 convolutions with stride [1  1] and pa
     3   'relu_conv10'       ReLU                   ReLU
     4   'pool10'            Average Pooling        14x14 average pooling with stride [1  1] and
     5   'prob'              Softmax                softmax
     6   'new_classoutput'   Classification Output  crossentropyex
```

13-91

Prepare RGB Data for SqueezeNet

The RGB images have dimensions appropriate for the GoogLeNet architecture. Create augmented image datastores that automatically resize the existing RGB images for the SqueezeNet architecture. For more information, see `augmentedImageDatastore` (Deep Learning Toolbox).

```
augimgsTrain = augmentedImageDatastore([227 227],imgsTrain);
augimgsValidation = augmentedImageDatastore([227 227],imgsValidation);
```

Set Training Options and Train SqueezeNet

Create a new set of training options to use with SqueezeNet. Set the random seed to the default value and train the network. The training process usually takes 1-5 minutes on a desktop CPU.

```
ilr = 3e-4;
miniBatchSize = 10;
maxEpochs = 15;
valFreq = floor(numel(augimgsTrain.Files)/miniBatchSize);
opts = trainingOptions('sgdm',...
    'MiniBatchSize',miniBatchSize,...
    'MaxEpochs',maxEpochs,...
    'InitialLearnRate',ilr,...
    'ValidationData',augimgsValidation,...
    'ValidationFrequency',valFreq,...
    'Verbose',1,...
    'ExecutionEnvironment','cpu',...
    'Plots','training-progress');

rng default
trainedSN = trainNetwork(augimgsTrain,lgraphSqz,opts);
```

```
Initializing input data normalization.
|========================================================================================...
| Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Validation  |  Mini-batch  |  Validati
|        |             |   (hh:mm:ss)   |   Accuracy   |   Accuracy   |     Loss     |     Loss
|========================================================================================...
|      1 |          1  |    00:00:01    |    20.00%    |    43.75%    |    5.2508    |    1.
|      1 |         13  |    00:00:11    |    60.00%    |    50.00%    |    0.9912    |    1.
|      2 |         26  |    00:00:20    |    60.00%    |    59.38%    |    0.8554    |    0.
|      3 |         39  |    00:00:30    |    60.00%    |    59.38%    |    0.8120    |    0.
|      4 |         50  |    00:00:38    |    50.00%    |              |    0.7885    |
|      4 |         52  |    00:00:40    |    60.00%    |    65.63%    |    0.7091    |    0.
|      5 |         65  |    00:00:49    |    90.00%    |    87.50%    |    0.4639    |    0.
|      6 |         78  |    00:00:59    |    70.00%    |    87.50%    |    0.6021    |    0.
|      7 |         91  |    00:01:08    |    90.00%    |    90.63%    |    0.2307    |    0.
|      8 |        100  |    00:01:15    |    90.00%    |              |    0.1827    |
|      8 |        104  |    00:01:18    |    90.00%    |    93.75%    |    0.2139    |    0.
|      9 |        117  |    00:01:28    |   100.00%    |    90.63%    |    0.0521    |    0.
|     10 |        130  |    00:01:38    |    90.00%    |    90.63%    |    0.1134    |    0.
|     11 |        143  |    00:01:47    |   100.00%    |    90.63%    |    0.0855    |    0.
|     12 |        150  |    00:01:52    |    90.00%    |              |    0.2394    |
|     12 |        156  |    00:01:57    |   100.00%    |    90.63%    |    0.0606    |    0.
|     13 |        169  |    00:02:06    |   100.00%    |    90.63%    |    0.0090    |    0.
|     14 |        182  |    00:02:16    |   100.00%    |    93.75%    |    0.0127    |    0.
|     15 |        195  |    00:02:25    |   100.00%    |    93.75%    |    0.0016    |    0.
|========================================================================================...
```

Inspect the last layer of the network. Confirm the Classification Output layer includes the three classes.

```
trainedSN.Layers(end)
```

```
ans =
  ClassificationOutputLayer with properties:

          Name: 'new_classoutput'
       Classes: [ARR    CHF    NSR]
    OutputSize: 3

  Hyperparameters
   LossFunction: 'crossentropyex'
```

Evaluate SqueezeNet Accuracy

Evaluate the network using the validation data.

```
[YPred,probs] = classify(trainedSN,augimgsValidation);
accuracy = mean(YPred==imgsValidation.Labels);
disp(['SqueezeNet Accuracy: ',num2str(100*accuracy),'%'])
```

```
SqueezeNet Accuracy: 93.75%
```

Conclusion

This example shows how to use transfer learning and continuous wavelet analysis to classify three classes of ECG signals by leveraging the pretrained CNNs GoogLeNet and SqueezeNet. Wavelet-based time-frequency representations of ECG signals are used to create scalograms. RGB images of the scalograms are generated. The images are used to fine-tune both deep CNNs. Activations of different network layers were also explored.

This example illustrates one possible workflow you can use for classifying signals using pretrained CNN models. Other workflows are possible. "Deploy Signal Classifier on NVIDIA Jetson Using Wavelet Analysis and Deep Learning" on page 13-170 and "Deploy Signal Classifier Using Wavelets and Deep Learning on Raspberry Pi" on page 13-146 show how to deploy code onto hardware for signal classification. GoogLeNet and SqueezeNet are models pretrained on a subset of the ImageNet database [10], which is used in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [8]. The ImageNet collection contains images of real-world objects such as fish, birds, appliances, and fungi. Scalograms fall outside the class of real-world objects. In order to fit into the GoogLeNet and SqueezeNet architecture, the scalograms also underwent data reduction. Instead of fine-tuning pretrained CNNs to distinguish different classes of scalograms, training a CNN from scratch at the original scalogram dimensions is an option.

References

1   Baim, D. S., W. S. Colucci, E. S. Monrad, H. S. Smith, R. F. Wright, A. Lanoue, D. F. Gauthier, B. J. Ransil, W. Grossman, and E. Braunwald. "Survival of patients with severe congestive heart failure treated with oral milrinone." *Journal of the American College of Cardiology*. Vol. 7, Number 3, 1986, pp. 661–670.

2   Engin, M. "ECG beat classification using neuro-fuzzy network." *Pattern Recognition Letters*. Vol. 25, Number 15, 2004, pp.1715–1722.

3   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, and H. E. Stanley. "PhysioBank, PhysioToolkit,and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals." Circulation. Vol. 101, Number 23: e215–e220. [Circulation Electronic Pages; http://circ.ahajournals.org/content/101/23/e215.full]; 2000 (June 13). doi: 10.1161/01.CIR.101.23.e215.

4 Leonarduzzi, R. F., G. Schlotthauer, and M. E. Torres. "Wavelet leader based multifractal analysis of heart rate variability during myocardial ischaemia." In *Engineering in Medicine and Biology Society (EMBC), Annual International Conference of the IEEE*, 110–113. Buenos Aires, Argentina: IEEE, 2010.

5 Li, T., and M. Zhou. "ECG classification using wavelet packet entropy and random forests." *Entropy*. Vol. 18, Number 8, 2016, p.285.

6 Maharaj, E. A., and A. M. Alonso. "Discriminant analysis of multivariate time series: Application to diagnosis based on ECG signals." *Computational Statistics and Data Analysis*. Vol. 70, 2014, pp. 67–87.

7 Moody, G. B., and R. G. Mark. "The impact of the MIT-BIH Arrhythmia Database." *IEEE Engineering in Medicine and Biology Magazine*. Vol. 20. Number 3, May-June 2001, pp. 45–50. (PMID: 11446209)

8 Russakovsky, O., J. Deng, and H. Su et al. "ImageNet Large Scale Visual Recognition Challenge." *International Journal of Computer Vision*. Vol. 115, Number 3, 2015, pp. 211–252.

9 Zhao, Q., and L. Zhang. "ECG feature extraction and classification using wavelet transform and support vector machines." In *IEEE International Conference on Neural Networks and Brain*, 1089–1092. Beijing, China: IEEE, 2005.

10 *ImageNet*. http://www.image-net.org

Supporting Functions

helperCreateECGDataDirectories creates a data directory inside a parent directory, then creates three subdirectories inside the data directory. The subdirectories are named after each class of ECG signal found in `ECGData`.

```
function helperCreateECGDirectories(ECGData,parentFolder,dataFolder)
% This function is only intended to support the ECGAndDeepLearningExample.
% It may change or be removed in a future release.

rootFolder = parentFolder;
localFolder = dataFolder;
mkdir(fullfile(rootFolder,localFolder))

folderLabels = unique(ECGData.Labels);
for i = 1:numel(folderLabels)
    mkdir(fullfile(rootFolder,localFolder,char(folderLabels(i))));
end
end
```

helperPlotReps plots the first thousand samples of a representative of each class of ECG signal found in `ECGData`.

```
function helperPlotReps(ECGData)
% This function is only intended to support the ECGAndDeepLearningExample.
% It may change or be removed in a future release.

folderLabels = unique(ECGData.Labels);

for k=1:3
    ecgType = folderLabels{k};
    ind = find(ismember(ECGData.Labels,ecgType));
    subplot(3,1,k)
    plot(ECGData.Data(ind(1),1:1000));
    grid on
```

```
        title(ecgType)
    end
end
```

helperCreateRGBfromTF uses `cwtfilterbank` to obtain the continuous wavelet transform of the ECG signals and generates the scalograms from the wavelet coefficients. The helper function resizes the scalograms and writes them to disk as jpeg images.

```
function helperCreateRGBfromTF(ECGData,parentFolder,childFolder)
% This function is only intended to support the ECGAndDeepLearningExample.
% It may change or be removed in a future release.

imageRoot = fullfile(parentFolder,childFolder);

data = ECGData.Data;
labels = ECGData.Labels;

[~,signalLength] = size(data);

fb = cwtfilterbank('SignalLength',signalLength,'VoicesPerOctave',12);
r = size(data,1);

for ii = 1:r
    cfs = abs(fb.wt(data(ii,:)));
    im = ind2rgb(im2uint8(rescale(cfs)),jet(128));

    imgLoc = fullfile(imageRoot,char(labels(ii)));
    imFileName = strcat(char(labels(ii)),'_',num2str(ii),'.jpg');
    imwrite(imresize(im,[224 224]),fullfile(imgLoc,imFileName));
end
end
```

See Also

`augmentedImageDatastore` | `cwtfilterbank` | `googlenet` | `imageDatastore` | `squeezenet` | `trainNetwork` | `trainingOptions`

Related Examples

- "Deploy Signal Classifier Using Wavelets and Deep Learning on Raspberry Pi" on page 13-146
- "Deploy Signal Classifier on NVIDIA Jetson Using Wavelet Analysis and Deep Learning" on page 13-170
- "Train Deep Learning Network to Classify New Images" (Deep Learning Toolbox)

Texture Classification with Wavelet Image Scattering

This example shows how to classify textures using wavelet image scattering. In addition to Wavelet Toolbox™, this example also requires Parallel Computing Toolbox™ and Image Processing Toolbox™.

In a digital image, texture provides information about the spatial arrangement of color or pixel intensities. Particular spatial arrangements of color or pixel intensities correspond to different appearances and consistencies of the physical material being imaged. Texture classification and segmentation of images has a number of important application areas. A particularly important example is biomedical image analysis where normal and pathologic states are often characterized by morphological and histological characteristics which manifest as differences in texture [4].

Wavelet Image Scattering

For classification problems, it is often useful to map the data into some alternative representation which discards irrelevant information while retaining the discriminative properties of each class. Wavelet image scattering constructs low-variance representations of images which are insensitive to translations and small deformations. Because translations and small deformations in the image do not affect class membership, scattering transform coefficients provide features from which you can build robust classification models.

Wavelet scattering works by cascading the image through a series of wavelet transforms, nonlinearities, and averaging [1][3][5]. The result of this *deep* feature extraction is that images in the same class are moved closer to each other in the scattering transform representation, while images belonging to different classes are moved farther apart.

KTH-TIPS

This example uses a publicly available texture database, the KTH-TIPS (Textures under varying Illumination, Pose, and Scale) image database [6]. The KTH-TIPS dataset used in this example is the grayscale version. There are 810 images in total with 10 textures and 81 images per texture. The majority of images are 200-by-200 in size. This example assumes you have downloaded the KTH-TIPS grayscale dataset and untarred it so that the 10 texture classes are contained in separate subfolders of a common folder. Each subfolder is named for the class of textures it contains. Untarring the downloaded `kth_tips_grey_200x200.tar` file is sufficient to provide a top-level folder KTH_TIPS and the required subfolder structure.

Use the `imageDatastore` to read the data. Set the `location` property of the `imageDatastore` to the folder containing the KTH-TIPS database that you have access to.

```
location = fullfile(tempdir,'kth_tips_grey_200x200','KTH_TIPS');
Imds = imageDatastore(location,'IncludeSubFolders',true,'FileExtensions','.png','LabelSource','fc
```

Randomly select and visualize 20 images from the dataset.

```
numImages = 810;
perm = randperm(numImages,20);
for np = 1:20
    subplot(4,5,np)
    im = imread(Imds.Files{perm(np)});
    imagesc(im);
    colormap gray; axis off;
end
```
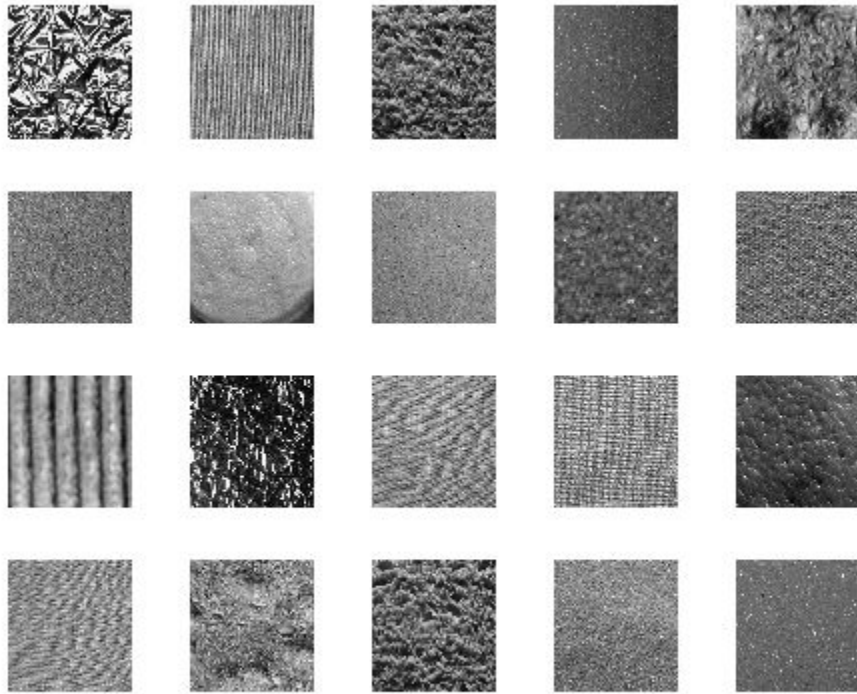
Texture Classification

This example uses MATLAB™'s parallel processing capability through the `tall` array interface. Start the parallel pool if one is not currently running.

```
if isempty(gcp)
    parpool;
end
```

```
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 6).
```

For reproducibility, set the random number generator. Shuffle the files of the KTH-TIPS dataset and split the 810 images into two randomly selected sets, one for training and one held-out set for testing. Use approximately 80% of the images for building a predictive model from the scattering transform and use the remainder for testing the model.

```
rng(100)
Imds = imageDatastore(location,'IncludeSubFolders',true,'FileExtensions','.png','LabelSource','fc
Imds = shuffle(Imds);
[trainImds,testImds] = splitEachLabel(Imds,0.8);
```

We now have two datasets. The training set consists of 650 images, with 65 images per texture. The testing set consists of 160 images, with 16 images per texture. To verify, count the labels in each dataset.

```
countEachLabel(trainImds)
```

```
ans=10×2 table
        Label            Count
    _____      _____

    aluminium_foil        65
    brown_bread           65
    corduroy              65
    cotton                65
    cracker               65
    linen                 65
    orange_peel           65
    sandpaper             65
    sponge                65
    styrofoam             65
```

```
countEachLabel(testImds)
```

```
ans=10×2 table
        Label            Count
    _____      _____

    aluminium_foil        16
    brown_bread           16
    corduroy              16
    cotton                16
    cracker               16
    linen                 16
    orange_peel           16
    sandpaper             16
    sponge                16
    styrofoam             16
```

Create `tall` arrays for the resized images.

```
Ttrain = tall(trainImds);
Ttest = tall(testImds);
```

Create a scattering framework for an image input size of 200-by-200 with an `InvarianceScale` of 150. The invariance scale hyperparameter is the only one we set in this example. For the other hyperparameters of the scattering transform, use the default values.

```
sn = waveletScattering2('ImageSize',[200 200],'InvarianceScale',150);
```

To extract features for classification for each the training and test sets, use the `helperScatImages_mean` function. The code for `helperScatImages_mean` is at the end of this example. `helperScatImages_mean` resizes the images to a common 200-by-200 size and uses the scattering framework, `sn`, to obtain the feature matrix. In this case, each feature matrix is 391-by-7-by-7. There are 391 scattering paths and each scattering coefficient image is 7-by-7. Finally, `helperScatImages_mean` obtains the mean along the 2nd and 3rd dimensions to obtain a 391 element feature vector for each image. This is a significant reduction in data from 40,000 elements down to 391.

```
trainfeatures = cellfun(@(x)helperScatImages_mean(sn,x),Ttrain,'Uni',0);
testfeatures = cellfun(@(x)helperScatImages_mean(sn,x),Ttest,'Uni',0);
```

Using `tall`'s `gather` capability, gather all the training and test feature vectors and concatenate them into matrices.

```
Trainf = gather(trainfeatures);
```

```
Evaluating tall expression using the Parallel Pool 'local':
- Pass 1 of 1: Completed in 1 min 39 sec
Evaluation completed in 1 min 39 sec
```

```
trainfeatures = cat(2,Trainf{:});
Testf = gather(testfeatures);
```

```
Evaluating tall expression using the Parallel Pool 'local':
- Pass 1 of 1: Completed in 23 sec
Evaluation completed in 23 sec
```

```
testfeatures = cat(2,Testf{:});
```

The previous code results in two matrices with row dimensions 391 and column dimension equal to the number of images in the training and test sets, respectively. So each column is a feature vector.

PCA Model and Prediction

This example constructs a simple classifier based on the principal components of the scattering feature vectors for each class. The classifier is implemented in the functions `helperPCAModel` and `helperPCAClassifier`. The function `helperPCAModel` determines the principal components for each digit class based on the scattering features. The code for `helperPCAModel` is at the end of this example. The function `helperPCAClassifier` classifies the held-out test data by finding the closest match (best projection) between the principal components of each test feature vector with the training set and assigning the class accordingly. The code for `helperPCAClassifier` is at the end of this example.

```
model = helperPCAModel(trainfeatures,30,trainImds.Labels);
predlabels = helperPCAClassifier(testfeatures,model);
```

After constructing the model and classifying the test set, determine the accuracy of the test set classification.

```
accuracy = sum(testImds.Labels == predlabels)./numel(testImds.Labels)*100
```

```
accuracy = 99.3750
```

We have achieved 99.375% correct classification, or a 0.625% error rate for the 160 images in the test set. A plot of the confusion matrix shows that our simple model misclassified one texture.

```
figure
confusionchart(testImds.Labels,predlabels)
```

Summary

In this example, we used wavelet image scattering to create low-variance representations of textures for classification. Using the scattering transform and a simple principal components classifier, we achieved 99.375% correct classification on a held-out test set. This result is comparable to state-of-the-art performance on the KTH-TIPS database.[2]

References

[1] Bruna, J., and S. Mallat. "Invariant Scattering Convolution Networks." *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 35, Number 8, 2013, pp. 1872–1886.

[2] Hayman, E., B. Caputo, M. Fritz, and J. O. Eklundh. "On the Significance of Real-World Conditions for Material Classification." In *Computer Vision - ECCV 2004*, edited by Tomás Pajdla and Jiří Matas, 3024:253–66. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. https://doi.org/10.1007/978-3-540-24673-2_21.

[3] Mallat, S. "Group Invariant Scattering." *Communications in Pure and Applied Mathematics*. Vol. 65, Number 10, 2012, pp. 1331–1398.

[4] Pujol, O., and P. Radeva. "Supervised Texture Classification for Intravascular Tissue Characterization." In *Handbook of Biomedical Image Analysis*, edited by Jasjit S. Suri, David L. Wilson, and Swamy Laxminarayan, 57–109. Boston, MA: Springer US, 2005. https://doi.org/10.1007/0-306-48606-7_2.

[5] Sifre, L., and S. Mallat. "Rotation, scaling and deformation invariant scattering for texture discrimination." *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp 1233–1240. 10.1109/CVPR.2013.163.

[6] *KTH-TIPS image databases homepage.* `https://www.csc.kth.se/cvap/databases/kth-tips/`

Appendix — Supporting Functions

helperScatImages_mean

```matlab
function features = helperScatImages_mean(sf,x)
x = imresize(x,[200 200]);
smat = featureMatrix(sf,x);
features = mean(mean(smat,2),3);
end
```

helperPCAModel

```matlab
function model = helperPCAModel(features,M,Labels)
% This function is only to support wavelet image scattering examples in
% Wavelet Toolbox. It may change or be removed in a future release.
% model = helperPCAModel(features,M,Labels)

% Copyright 2018 MathWorks

% Initialize structure array to hold the affine model
model = struct('Dim',[],'mu',[],'U',[],'Labels',categorical([]),'s',[]);
model.Dim = M;
% Obtain the number of classes
LabelCategories = categories(Labels);
Nclasses = numel(categories(Labels));
for kk = 1:Nclasses
    Class = LabelCategories{kk};
    % Find indices corresponding to each class
    idxClass = Labels == Class;
    % Extract feature vectors for each class
    tmpFeatures = features(:,idxClass);
    % Determine the mean for each class
    model.mu{kk} = mean(tmpFeatures,2);
    [model.U{kk},model.S{kk}] = scatPCA(tmpFeatures);
    if size(model.U{kk},2) > M
        model.U{kk} = model.U{kk}(:,1:M);
        model.S{kk} = model.S{kk}(1:M);

    end
    model.Labels(kk) = Class;
end

function [u,s,v] = scatPCA(x,M)
    % Calculate the principal components of x along the second dimension.

    if nargin > 1 && M > 0
        % If M is non-zero, calculate the first M principal components.
        [u,s,v] = svds(x-sig_mean(x),M);
        s = abs(diag(s)/sqrt(size(x,2)-1)).^2;
    else
        % Otherwise, calculate all the principal components.
```

```matlab
        % Each row is an observation, i.e. the number of scattering paths
        % Each column is a class observation
        [u,d] = eig(cov(x'));
        [s,ind] = sort(diag(d),'descend');
        u = u(:,ind);
    end
end
end
```

helperPCAClassifier

```matlab
function labels = helperPCAClassifier(features,model)
% This function is only to support wavelet image scattering examples in
% Wavelet Toolbox. It may change or be removed in a future release.
% model is a structure array with fields, M, mu, v, and Labels
% features is the matrix of test data which is Ns-by-L, Ns is the number of
% scattering paths and L is the number of test examples. Each column of
% features is a test example.

% Copyright 2018 MathWorks

labelIdx = determineClass(features,model);
labels = model.Labels(labelIdx);
% Returns as column vector to agree with imageDatastore Labels
labels = labels(:);


%--------------------------------------------------------------------------
function labelIdx = determineClass(features,model)
% Determine number of classes
Nclasses = numel(model.Labels);
% Initialize error matrix
errMatrix = Inf(Nclasses,size(features,2));
for nc = 1:Nclasses
    % class centroid
    mu = model.mu{nc};
    u = model.U{nc};
    % 1-by-L
    errMatrix(nc,:) = projectionError(features,mu,u);
end
% Determine minimum along class dimension
[~,labelIdx] = min(errMatrix,[],1);


%--------------------------------------------------------------------------
function totalerr = projectionError(features,mu,u)
    %
    Npc = size(u,2);
    L = size(features,2);
    % Subtract class mean: Ns-by-L minus Ns-by-1
    s = features-mu;
    % 1-by-L
    normSqX = sum(abs(s).^2,1)';
    err = Inf(Npc+1,L);
    err(1,:) = normSqX;
    err(2:end,:) = -abs(u'*s).^2;
    % 1-by-L
    totalerr = sqrt(sum(err,1));
```

```
end
end
end
```

See Also

`waveletScattering2`

More About

- "Wavelet Scattering" on page 9-2

Digit Classification with Wavelet Scattering

This example shows how to use wavelet scattering for image classification. This example requires Wavelet Toolbox™, Deep Learning Toolbox™, and Parallel Computing Toolbox™.

For classification problems, it is often useful to map the data into some alternative representation which discards irrelevant information while retaining the discriminative properties of each class. Wavelet image scattering constructs low-variance representations of images which are insensitive to translations and small deformations. Because translations and small deformations in the image do not affect class membership, scattering transform coefficients provide features from which you can build robust classification models.

Wavelet scattering works by cascading the image through a series of wavelet transforms, nonlinearities, and averaging [1][3][4]. The result of this *deep* feature extraction is that images in the same class are moved closer to each other in the scattering transform representation, while images belonging to different classes are moved farther apart. While the wavelet scattering transform has a number of architectural similarities with deep convolutional neural networks, including convolution operators, nonlinearities, and averaging, the filters in a scattering transform are pre-defined and fixed.

Digit Images

The dataset used in this example contains 10,000 synthetic images of digits from 0 to 9. The images are generated by applying random transformations to images of those digits created with different fonts. Each digit image is 28-by-28 pixels. The dataset contains an equal number of images per category. Use the `imageDataStore` to read the images.

```
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos','nndatasets','DigitDataset');
Imds = imageDatastore(digitDatasetPath,'IncludeSubfolders',true, 'LabelSource','foldernames');
```

Randomly select and plot 20 images from the dataset.

```
figure
numImages = 10000;
rng(100);
perm = randperm(numImages,20);
for np = 1:20
    subplot(4,5,np);
    imshow(Imds.Files{perm(np)});
end
```
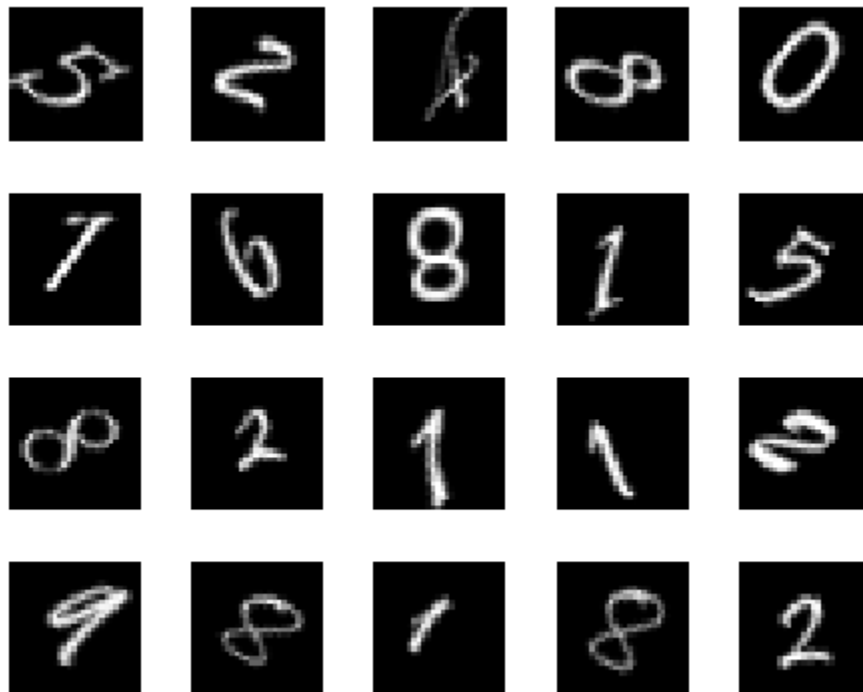
You can see that the 8s exhibit considerable variability while all being identifiable as an 8. The same is true of the other repeated digits in the sample. This is consistent with natural handwriting where any digit differs non-trivially between individuals and even within the same individual's handwriting with respect to translation, rotation, and other small deformations. Using wavelet scattering, we hope to build representations of these digits which obscure this irrelevant variability.

Wavelet Image Scattering Feature Extraction

The synthetic images are 28-by-28. Create a wavelet image scattering framework and set the invariance scale to equal the size of the image. Set the number of rotations to 8 in each of the two wavelet scattering filter banks. The construction of the wavelet scattering framework requires that we set only two hyperparameters: the `InvarianceScale` and `NumRotations`.

```
sf = waveletScattering2('ImageSize',[28 28],'InvarianceScale',28, ...
    'NumRotations',[8 8]);
```

This example uses MATLAB™'s parallel processing capability through the `tall` array interface. If a parallel pool is not currently running, you can start one by executing the following code. Alternatively, the first time you create a `tall` array, the parallel pool is created.

```
if isempty(gcp)
    parpool;
end
```

```
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 6).
```

For reproducibility, set the random number generator. Shuffle the files of the `imageDatastore` and split the 10,000 images into two sets, one for training and one held-out set for testing. Allocate 80% of the data, or 8,000 images, to the training set and hold out the remaining 2,000 images for testing. Create `tall` arrays from the training and test datasets. Use the helper function `helperScatImages` to create feature vectors from the scattering transform coefficients. `helperScatImages` obtains the log of the scattering transform feature matrix as well as the mean along both the row and column dimensions of each image. The code for `helperScatImages` is at the end of this example. For each image in this example, the helper function creates a 217-by-1 feature vector.

```
rng(10);
Imds = shuffle(Imds);
[trainImds,testImds] = splitEachLabel(Imds,0.8);
Ttrain = tall(trainImds);
Ttest = tall(testImds);
trainfeatures = cellfun(@(x)helperScatImages(sf,x),Ttrain,'UniformOutput',false);
testfeatures = cellfun(@(x)helperScatImages(sf,x),Ttest,'UniformOutput',false);
```

Use `tall`'s `gather` capability to concatenate all the training and test features.

```
Trainf = gather(trainfeatures);
```

```
Evaluating tall expression using the Parallel Pool 'local':
- Pass 1 of 1: Completed in 3 min 51 sec
Evaluation completed in 3 min 51 sec
```

```
trainfeatures = cat(2,Trainf{:});
Testf = gather(testfeatures);
```

```
Evaluating tall expression using the Parallel Pool 'local':
- Pass 1 of 1: Completed in 49 sec
Evaluation completed in 49 sec
```

```
testfeatures = cat(2,Testf{:});
```

The previous code results in two matrices with row dimensions 217 and column dimension equal to the number of images in the training and test sets respectively. Accordingly, each column is a feature vector for its corresponding image. The original images contained 784 elements. The scattering coefficients represent an approximate 4-fold reduction in the size of each image.

PCA Model and Prediction

This example constructs a simple classifier based on the principal components of the scattering feature vectors for each class. The classifier is implemented in the functions `helperPCAModel` and `helperPCAClassifier`. `helperPCAModel` determines the principal components for each digit class based on the scattering features. The code for `helperPCAModel` is at the end of this example. `helperPCAClassifier` classifies the held-out test data by finding the closest match (best projection) between the principal components of each test feature vector with the training set and assigning the class accordingly. The code for `helperPCAClassifier` is at the end of this example.

```
model = helperPCAModel(trainfeatures,30,trainImds.Labels);
predlabels = helperPCAClassifier(testfeatures,model);
```

After constructing the model and classifying the test set, determine the accuracy of the test set classification.

```
accuracy = sum(testImds.Labels == predlabels)./numel(testImds.Labels)*100
```

```
accuracy = 99.6000
```

We have achieved 99.6% correct classification of the test data. To see how the 2,000 test images have been classified, plot the confusion matrix. There are 200 examples in the test set for each of the 10 classes.

```
figure;
confusionchart(testImds.Labels,predlabels)
title('Test-Set Confusion Matrix -- Wavelet Scattering')
```
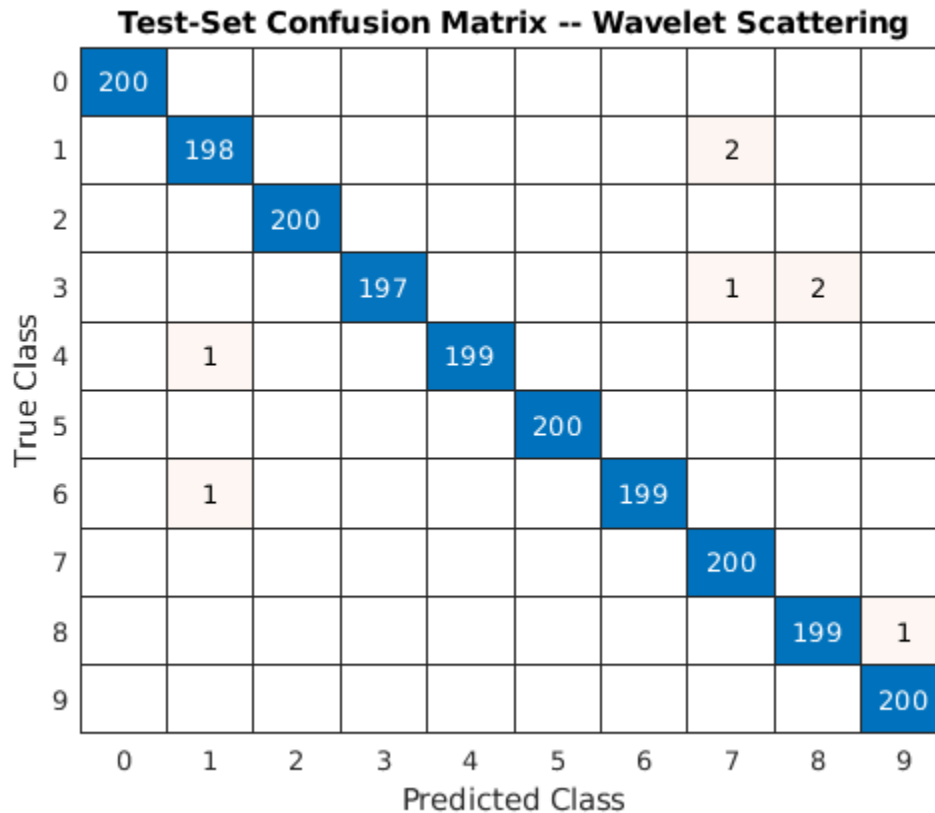

CNN

In this section, we train a simple convolutional neural network (CNN) to recognize digits. Construct the CNN to consist of a convolution layer with 20 5-by-5 filters with 1-by-1 strides. Follow the convolution layer with a RELU activation and max pooling layer. Use a fully connected layer, followed by a softmax layer to normalize the output of the fully connected layer to probabilities. Use a cross entropy loss function for learning.

```
imageSize = [28 28 1];
layers = [ ...
    imageInputLayer([28 28 1])
    convolution2dLayer(5,20)
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
```

Use stochastic gradient descent with momentum and a learning rate of 0.0001 for training. Set the maximum number of epochs to 20. For reproducibility, set the ExecutionEnvironment to 'cpu'.

```
options = trainingOptions('sgdm', ...
    'MaxEpochs',20,...
    'InitialLearnRate',1e-4, ...
    'Verbose',false, ...
    'Plots','training-progress','ExecutionEnvironment','cpu');
```

Train the network. For training and testing we use the same data sets used in the scattering transform.

```
reset(trainImds);
reset(testImds);
net = trainNetwork(trainImds,layers,options);
```
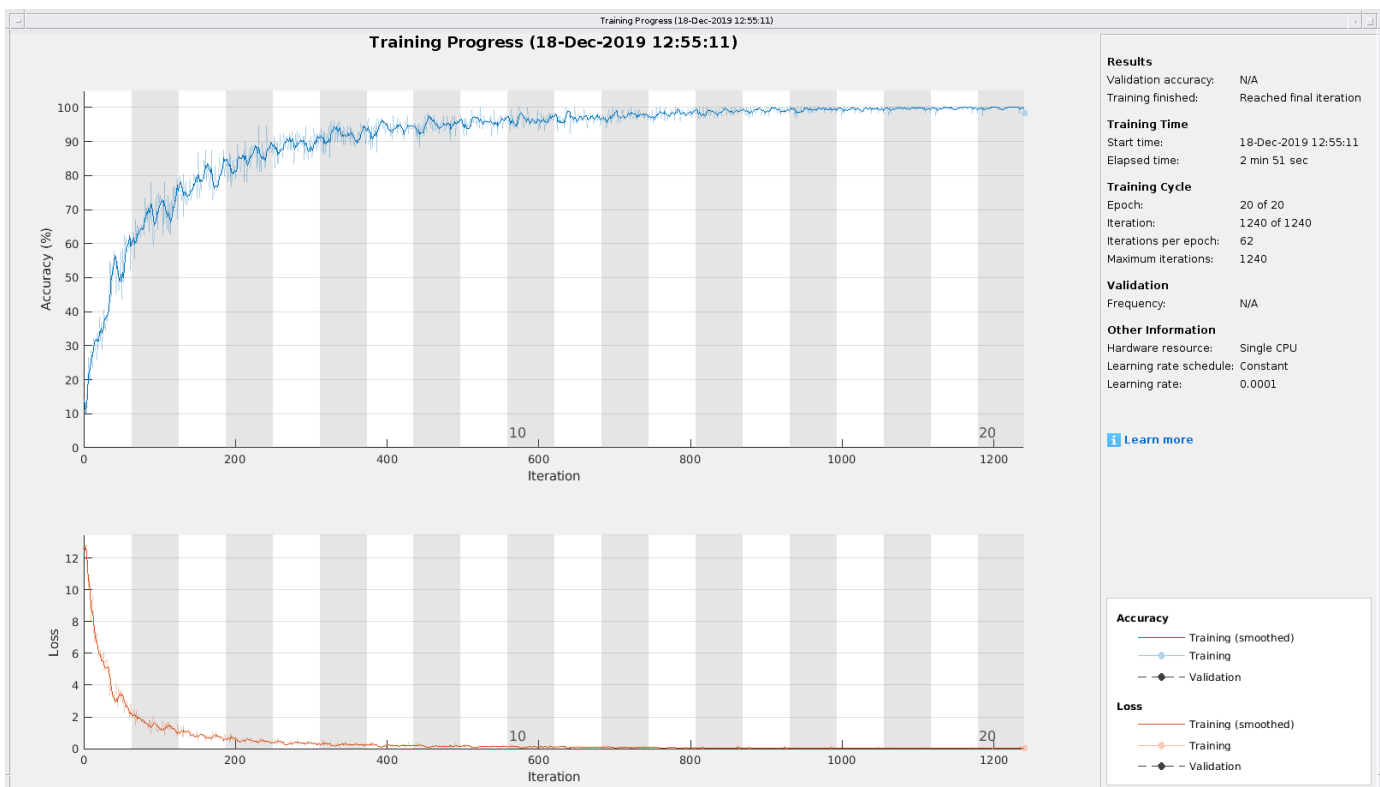

By the end of training, the CNN is performing near 100% on the training set. Use the trained network to make predictions on the held-out test set.

```
YPred = classify(net,testImds,'ExecutionEnvironment','cpu');
DCNNaccuracy = sum(YPred == testImds.Labels)/numel(YPred)*100
```

```
DCNNaccuracy = 95.5000
```

The simple CNN has achieved 95.5% correct classification on the held-out test set. Plot the confusion chart for the CNN.

```
figure;
confusionchart(testImds.Labels,YPred)
title('Test-Set Confusion Chart -- CNN')
```

Test-Set Confusion Chart -- CNN

True Class \ Predicted Class	0	1	2	3	4	5	6	7	8	9
0	193		2		1				1	3
1		195		1				4		
2	1		193	2	2				1	1
3			2	191		3			2	2
4		1			198	1				
5				1	1	197				1
6		3	1	1	1	7	181		6	
7	1	3	1					195		
8		3	2	5	1	2	2		181	4
9			1	1	7				5	186

Summary

This example used wavelet image scattering to create low-variance representations of digit images for classification. Using the scattering transform with fixed filter weights and a simple principal components classifier, we achieved 99.6% correct classification on a held-out test set. With a simple CNN in which the filters are learned, we achieved 95.5% correct. This example is not intended as a direct comparison of the scattering transform and CNNs. There are multiple hyperparameter and architectural changes you can make in each case, which significantly affect the results. The goal of this example was simply to demonstrate the potential of deep feature extractors like the wavelet scattering transform to produce robust representations of data for learning.

References

[1] Bruna, J., and S. Mallat. "Invariant Scattering Convolution Networks." *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 35, Number 8, 2013, pp. 1872–1886.

[2] Mallat, S. "Group Invariant Scattering." *Communications in Pure and Applied Mathematics*. Vol. 65, Number 10, 2012, pp. 1331–1398.

[3] Sifre, L., and S. Mallat. "Rotation, scaling and deformation invariant scattering for texture discrimination." *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp 1233–1240. 10.1109/CVPR.2013.163.

Appendix — Supporting Functions

helperScatImages

```matlab
function features = helperScatImages(sf,x)
% This function is only to support examples in the Wavelet Toolbox.
% It may change or be removed in a future release.

% Copyright 2018 MathWorks

smat = featureMatrix(sf,x,'transform','log');
features = mean(mean(smat,2),3);
end
```

helperPCAModel

```matlab
function model = helperPCAModel(features,M,Labels)
% This function is only to support wavelet image scattering examples in
% Wavelet Toolbox. It may change or be removed in a future release.
% model = helperPCAModel(features,M,Labels)

% Copyright 2018 MathWorks

% Initialize structure array to hold the affine model
model = struct('Dim',[],'mu',[],'U',[],'Labels',categorical([]),'s',[]);
model.Dim = M;
% Obtain the number of classes
LabelCategories = categories(Labels);
Nclasses = numel(categories(Labels));
for kk = 1:Nclasses
    Class = LabelCategories{kk};
    % Find indices corresponding to each class
    idxClass = Labels == Class;
    % Extract feature vectors for each class
    tmpFeatures = features(:,idxClass);
    % Determine the mean for each class
    model.mu{kk} = mean(tmpFeatures,2);
    [model.U{kk},model.S{kk}] = scatPCA(tmpFeatures);
    if size(model.U{kk},2) > M
        model.U{kk} = model.U{kk}(:,1:M);
        model.S{kk} = model.S{kk}(1:M);

    end
    model.Labels(kk) = Class;
end

function [u,s,v] = scatPCA(x,M)
    % Calculate the principal components of x along the second dimension.

    if nargin > 1 && M > 0
        % If M is non-zero, calculate the first M principal components.
        [u,s,v] = svds(x-sig_mean(x),M);
        s = abs(diag(s)/sqrt(size(x,2)-1)).^2;
    else
        % Otherwise, calculate all the principal components.
        % Each row is an observation, i.e. the number of scattering paths
        % Each column is a class observation
        [u,d] = eig(cov(x'));
        [s,ind] = sort(diag(d),'descend');
        u = u(:,ind);
    end
```

```
        end
    end
```

helperPCAClassifier

```
function labels = helperPCAClassifier(features,model)
% This function is only to support wavelet image scattering examples in
% Wavelet Toolbox. It may change or be removed in a future release.
% model is a structure array with fields, M, mu, v, and Labels
% features is the matrix of test data which is Ns-by-L, Ns is the number of
% scattering paths and L is the number of test examples. Each column of
% features is a test example.

% Copyright 2018 MathWorks

labelIdx = determineClass(features,model);
labels = model.Labels(labelIdx);
% Returns as column vector to agree with imageDatastore Labels
labels = labels(:);


%--------------------------------------------------------------------------
function labelIdx = determineClass(features,model)
% Determine number of classes
Nclasses = numel(model.Labels);
% Initialize error matrix
errMatrix = Inf(Nclasses,size(features,2));
for nc = 1:Nclasses
    % class centroid
    mu = model.mu{nc};
    u = model.U{nc};
    % 1-by-L
    errMatrix(nc,:) = projectionError(features,mu,u);
end
% Determine minimum along class dimension
[~,labelIdx] = min(errMatrix,[],1);


%--------------------------------------------------------------------------
function totalerr = projectionError(features,mu,u)
    %
    Npc = size(u,2);
    L = size(features,2);
    % Subtract class mean: Ns-by-L minus Ns-by-1
    s = features-mu;
    % 1-by-L
    normSqX = sum(abs(s).^2,1)';
    err = Inf(Npc+1,L);
    err(1,:) = normSqX;
    err(2:end,:) = -abs(u'*s).^2;
    % 1-by-L
    totalerr = sqrt(sum(err,1));
end
end
```

```
end
```

See Also
waveletScattering2

More About
- "Wavelet Scattering" on page 9-2

Acoustic Scene Recognition Using Late Fusion

This example shows how to create a multi-model late fusion system for acoustic scene recognition. The example trains a convolutional neural network (CNN) using mel spectrograms and an ensemble classifier using wavelet scattering. The example uses the TUT dataset for training and evaluation [1].

Introduction

Acoustic scene classification (ASC) is the task of classifying environments from the sounds they produce. ASC is a generic classification problem that is foundational for context awareness in devices, robots, and many other applications [1]. Early attempts at ASC used mel-frequency cepstral coefficients (`mfcc` (Audio Toolbox)) and Gaussian mixture models (GMMs) to describe their statistical distribution. Other popular features used for ASC include zero crossing rate, spectral centroid (`spectralCentroid` (Audio Toolbox)), spectral rolloff (`spectralRolloffPoint` (Audio Toolbox)), spectral flux (`spectralFlux` (Audio Toolbox)), and linear prediction coefficients (`lpc` (Signal Processing Toolbox)) [5]. Hidden Markov models (HMMs) were trained to describe the temporal evolution of the GMMs. More recently, the best performing systems have used deep learning, usually CNNs, and a fusion of multiple models. The most popular feature for top-ranked systems in the DCASE 2017 contest was the mel spectrogram (`melSpectrogram` (Audio Toolbox)). The top-ranked systems in the challenge used late fusion and data augmentation to help their systems generalize.

To illustrate a simple approach that produces reasonable results, this example trains a CNN using mel spectrograms and an ensemble classifier using wavelet scattering. The CNN and ensemble classifier produce roughly equivalent overall accuracy, but perform better at distinguishing different acoustic scenes. To increase overall accuracy, you merge the CNN and ensemble classifier results using late fusion.

Load Acoustic Scene Recognition Data Set

To run the example, you must first download the data set [1]. The full data set is approximately 15.5 GB. Depending on your machine and internet connection, downloading the data can take about 4 hours.

```
downloadFolder = tempdir;
datasetFolder = fullfile(downloadFolder,'TUT-acoustic-scenes-2017');

if ~exist(datasetFolder,'dir')
    disp('Downloading TUT-acoustic-scenes-2017 (15.5 GB)...')
    HelperDownload_TUT_acoustic_scenes_2017(datasetFolder);
end
```

Read in the development set metadata as a table. Name the table variables `FileName`, `AcousticScene`, and `SpecificLocation`.

```
metadata_train = readtable(fullfile(datasetFolder,'TUT-acoustic-scenes-2017-development','meta.t
    'Delimiter',{'\t'}, ...
    'ReadVariableNames',false);
metadata_train.Properties.VariableNames = {'FileName','AcousticScene','SpecificLocation'};
head(metadata_train)
```

```
ans =

  8×3 table
```

```
          FileName                AcousticScene        SpecificLocation
    _____     _____        _____

    {'audio/b020_90_100.wav' }       {'beach'}             {'b020'}
    {'audio/b020_110_120.wav'}       {'beach'}             {'b020'}
    {'audio/b020_100_110.wav'}       {'beach'}             {'b020'}
    {'audio/b020_40_50.wav'  }       {'beach'}             {'b020'}
    {'audio/b020_50_60.wav'  }       {'beach'}             {'b020'}
    {'audio/b020_30_40.wav'  }       {'beach'}             {'b020'}
    {'audio/b020_160_170.wav'}       {'beach'}             {'b020'}
    {'audio/b020_170_180.wav'}       {'beach'}             {'b020'}
```

```matlab
metadata_test = readtable(fullfile(datasetFolder,'TUT-acoustic-scenes-2017-evaluation','meta.txt
    'Delimiter',{'\t'}, ...
    'ReadVariableNames',false);
metadata_test.Properties.VariableNames = {'FileName','AcousticScene','SpecificLocation'};
head(metadata_test)
```

```
ans =

  8×3 table

        FileName          AcousticScene      SpecificLocation
    _____     _____      _____

    {'audio/1245.wav'}       {'beach'}           {'b174'}
    {'audio/1456.wav'}       {'beach'}           {'b174'}
    {'audio/1318.wav'}       {'beach'}           {'b174'}
    {'audio/967.wav' }       {'beach'}           {'b174'}
    {'audio/203.wav' }       {'beach'}           {'b174'}
    {'audio/777.wav' }       {'beach'}           {'b174'}
    {'audio/231.wav' }       {'beach'}           {'b174'}
    {'audio/768.wav' }       {'beach'}           {'b174'}
```

Note that the specific recording locations in the test set do not intersect with the specific recording locations in the development set. This makes it easier to validate that the trained models can generalize to real-world scenarios.

```matlab
sharedRecordingLocations = intersect(metadata_test.SpecificLocation,metadata_train.SpecificLocati
fprintf('Number of specific recording locations in both train and test sets = %d\n',numel(sharedF
```

```
Number of specific recording locations in both train and test sets = 0
```

The first variable of the metadata tables contains the file names. Concatenate the file names with the file paths.

```matlab
train_filePaths = fullfile(datasetFolder,'TUT-acoustic-scenes-2017-development',metadata_train.F:
```

```matlab
test_filePaths = fullfile(datasetFolder,'TUT-acoustic-scenes-2017-evaluation',metadata_test.FileN
```

Create audio datastores for the train and test sets. Set the `Labels` property of the `audioDatastore` (Audio Toolbox) to the acoustic scene. Call `countEachLabel` (Audio Toolbox) to verify an even distribution of labels in both the train and test sets.

```matlab
adsTrain = audioDatastore(train_filePaths, ...
    'Labels',categorical(metadata_train.AcousticScene), ...
```

```
    'IncludeSubfolders',true);
display(countEachLabel(adsTrain))

adsTest = audioDatastore(test_filePaths, ...
    'Labels',categorical(metadata_test.AcousticScene), ...
    'IncludeSubfolders',true);
display(countEachLabel(adsTest))
```

```
  15×2 table

      Label              Count
    _____     _____

    beach                312
    bus                  312
    cafe/restaurant      312
    car                  312
    city_center          312
    forest_path          312
    grocery_store        312
    home                 312
    library              312
    metro_station        312
    office               312
    park                 312
    residential_area     312
    train                312
    tram                 312

  15×2 table

      Label              Count
    _____     _____

    beach                108
    bus                  108
    cafe/restaurant      108
    car                  108
    city_center          108
    forest_path          108
    grocery_store        108
    home                 108
    library              108
    metro_station        108
    office               108
    park                 108
    residential_area     108
    train                108
    tram                 108
```

You can reduce the data set used in this example to speed up the run time at the cost of performance. In general, reducing the data set is a good practice for development and debugging. Set reduceDataset to true to reduce the data set.

```
reduceDataset = false;
if reduceDataset
    adsTrain = splitEachLabel(adsTrain,20);
```

```
    adsTest = splitEachLabel(adsTest,10);
end
```

Call `read` (Audio Toolbox) to get the data and sample rate of a file from the train set. Audio in the database has consistent sample rate and duration. Normalize the audio and listen to it. Display the corresponding label.

```
[data,adsInfo] = read(adsTrain);
data = data./max(data,[],'all');

fs = adsInfo.SampleRate;
sound(data,fs)

fprintf('Acoustic scene = %s\n',adsTrain.Labels(1))
```

```
Acoustic scene = beach
```

Call `reset` (Audio Toolbox) to return the datastore to its initial condition.

```
reset(adsTrain)
```

Feature Extraction for CNN

Each audio clip in the dataset consists of 10 seconds of stereo (left-right) audio. The feature extraction pipeline and the CNN architecture in this example are based on [3]. Hyperparameters for the feature extraction, the CNN architecture, and the training options were modified from the original paper using a systematic hyperparameter optimization workflow.

First, convert the audio to mid-side encoding. [3] suggests that mid-side encoded data provides better spatial information that the CNN can use to identify moving sources (such as a train moving across an acoustic scene).

```
dataMidSide = [sum(data,2),data(:,1)-data(:,2)];
```

Divide the signal into one-second segments with overlap. The final system uses a probability-weighted average on the one-second segments to predict the scene for each 10-second audio clip in the test set. Dividing the audio clips into one-second segments makes the network easier to train and helps prevent overfitting to specific acoustic events in the training set. The overlap helps to ensure all combinations of features relative to one another are captured by the training data. It also provides the system with additional data that can be mixed uniquely during augmentation.

```
segmentLength = 1;
segmentOverlap = 0.5;

[dataBufferedMid,~] = buffer(dataMidSide(:,1),round(segmentLength*fs),round(segmentOverlap*fs),'r
[dataBufferedSide,~] = buffer(dataMidSide(:,2),round(segmentLength*fs),round(segmentOverlap*fs),
dataBuffered = zeros(size(dataBufferedMid,1),size(dataBufferedMid,2)+size(dataBufferedSide,2));
dataBuffered(:,1:2:end) = dataBufferedMid;
dataBuffered(:,2:2:end) = dataBufferedSide;
```

Use `melSpectrogram` (Audio Toolbox) to transform the data into a compact frequency-domain representation. Define parameters for the mel spectrogram as suggested by [3].

```
windowLength = 2048;
samplesPerHop = 1024;
samplesOverlap = windowLength - samplesPerHop;
fftLength = 2*windowLength;
numBands = 128;
```

`melSpectrogram` operates along channels independently. To optimize processing time, call `melSpectrogram` with the entire buffered signal.

```
spec = melSpectrogram(dataBuffered,fs, ...
    'Window',hamming(windowLength,'periodic'), ...
    'OverlapLength',samplesOverlap, ...
    'FFTLength',fftLength, ...
    'NumBands',numBands);
```

Convert the mel spectrogram into the logarithmic scale.

```
spec = log10(spec+eps);
```

Reshape the array to dimensions (Number of bands)-by-(Number of hops)-by-(Number of channels)-by-(Number of segments). When you feed an image into a neural network, the first two dimensions are the height and width of the image, the third dimension is the channels, and the fourth dimension separates the individual images.

```
X = reshape(spec,size(spec,1),size(spec,2),size(data,2),[]);
```

Call `melSpectrogram` without output arguments to plot the mel spectrogram of the mid channel for the first six of the one-second increments.

```
for channel = 1:2:11
    figure
    melSpectrogram(dataBuffered(:,channel),fs, ...
        'Window',hamming(windowLength,'periodic'), ...
        'OverlapLength',samplesOverlap, ...
        'FFTLength',fftLength, ...
        'NumBands',numBands);
    title(sprintf('Segment %d',ceil(channel/2)))
end
```
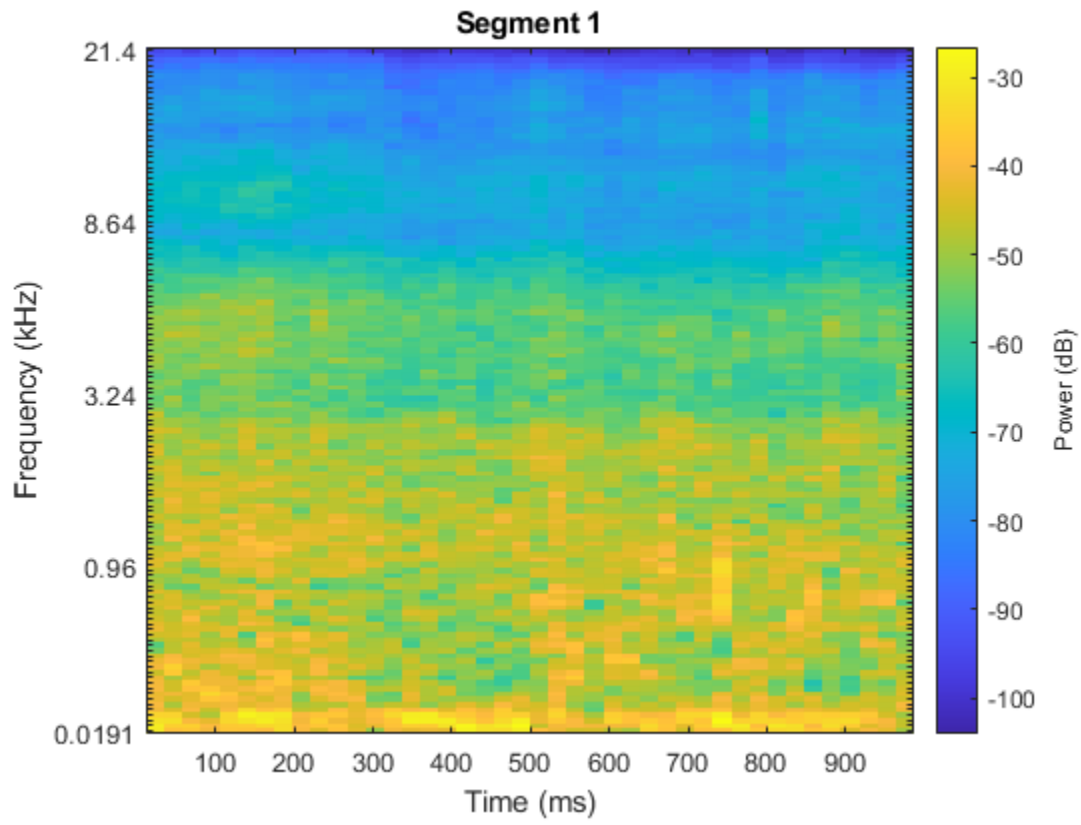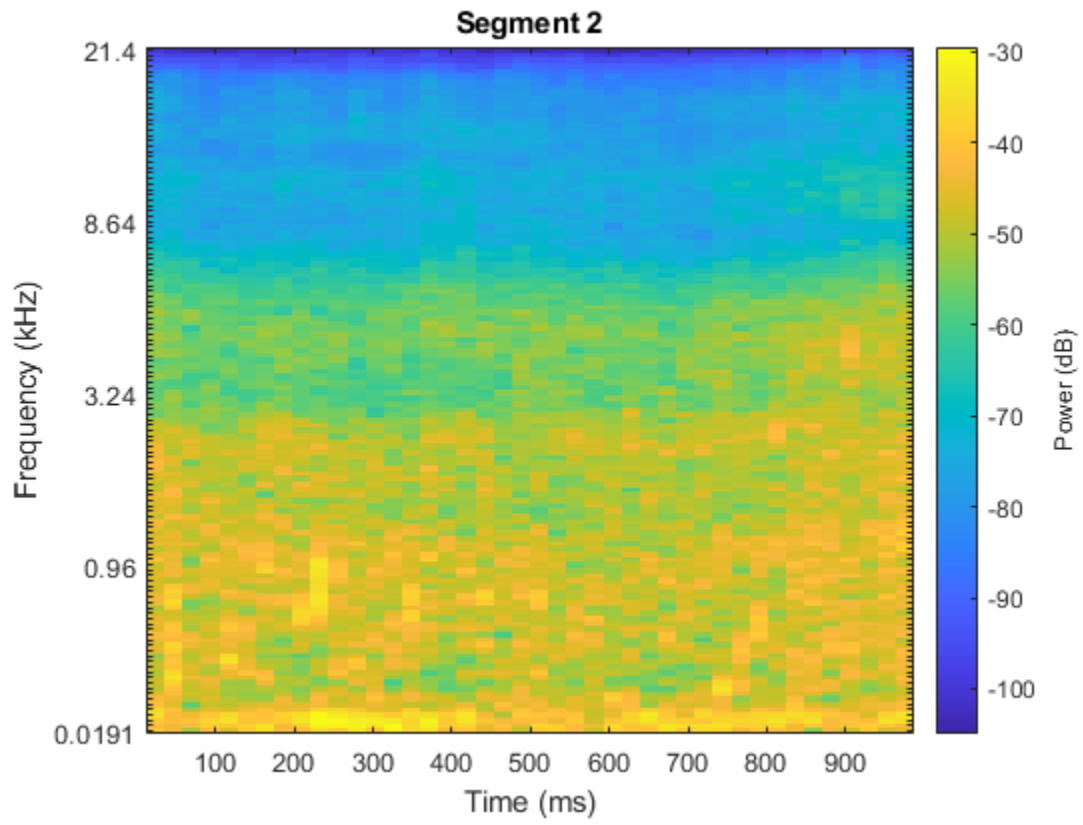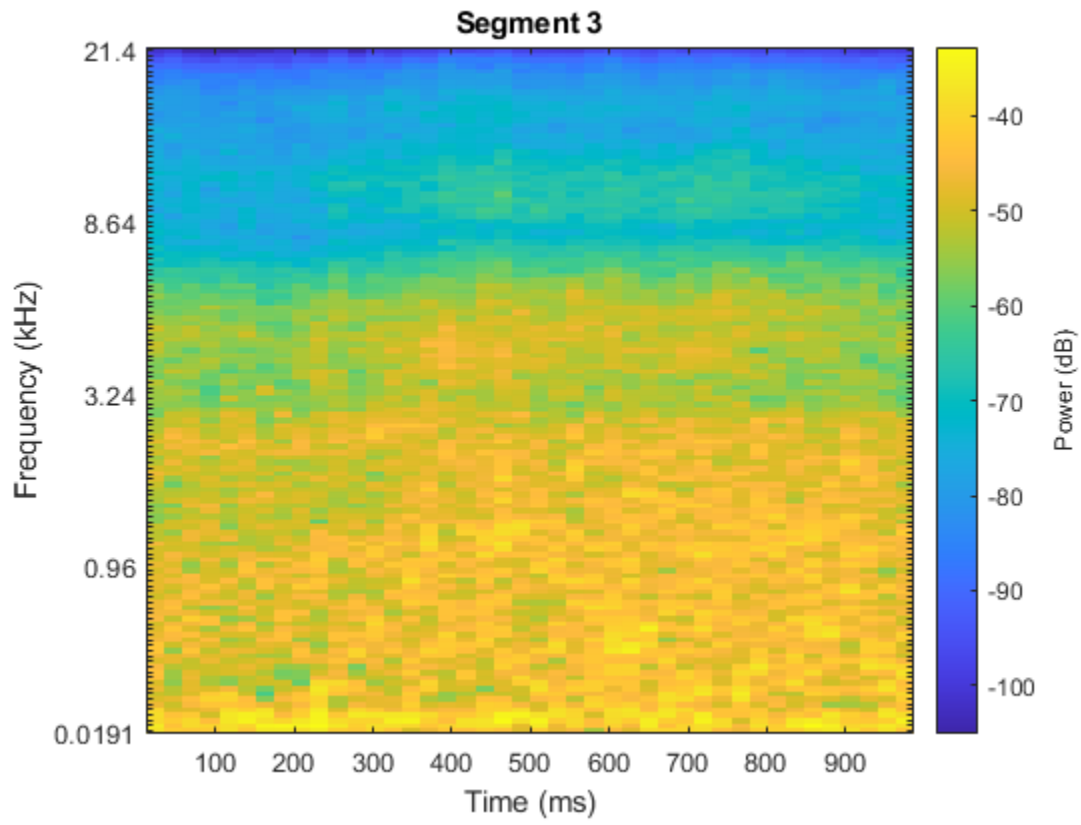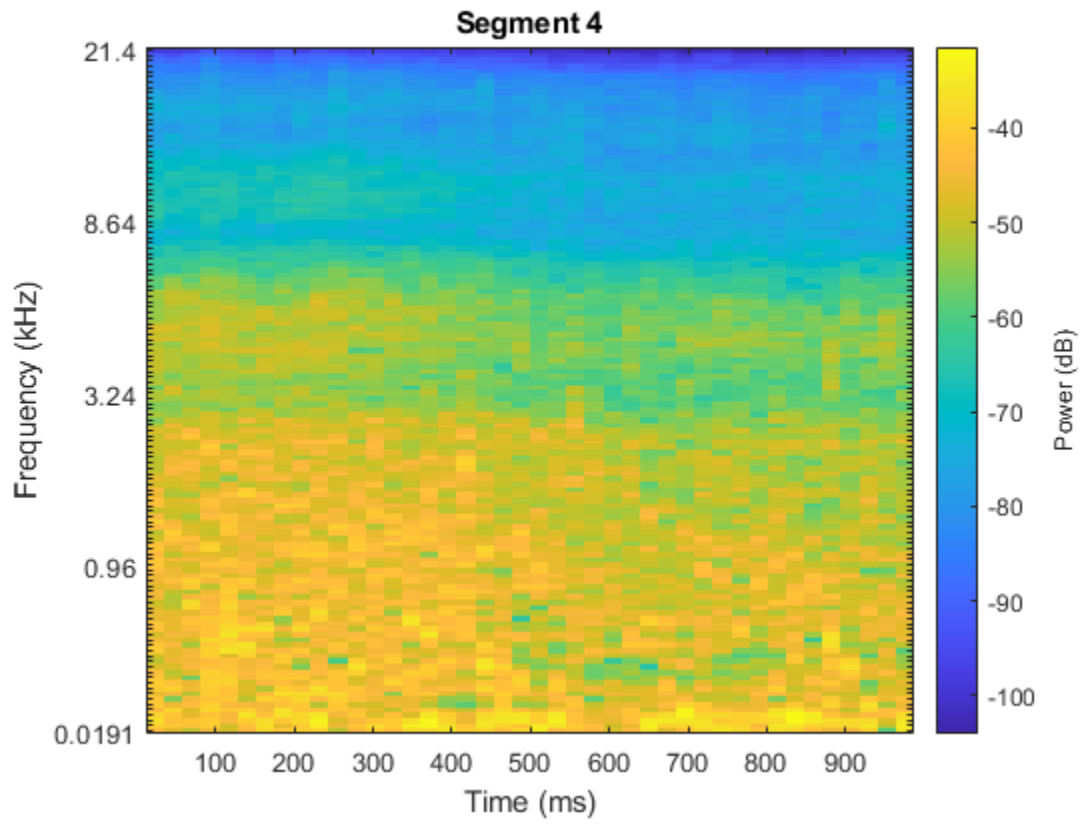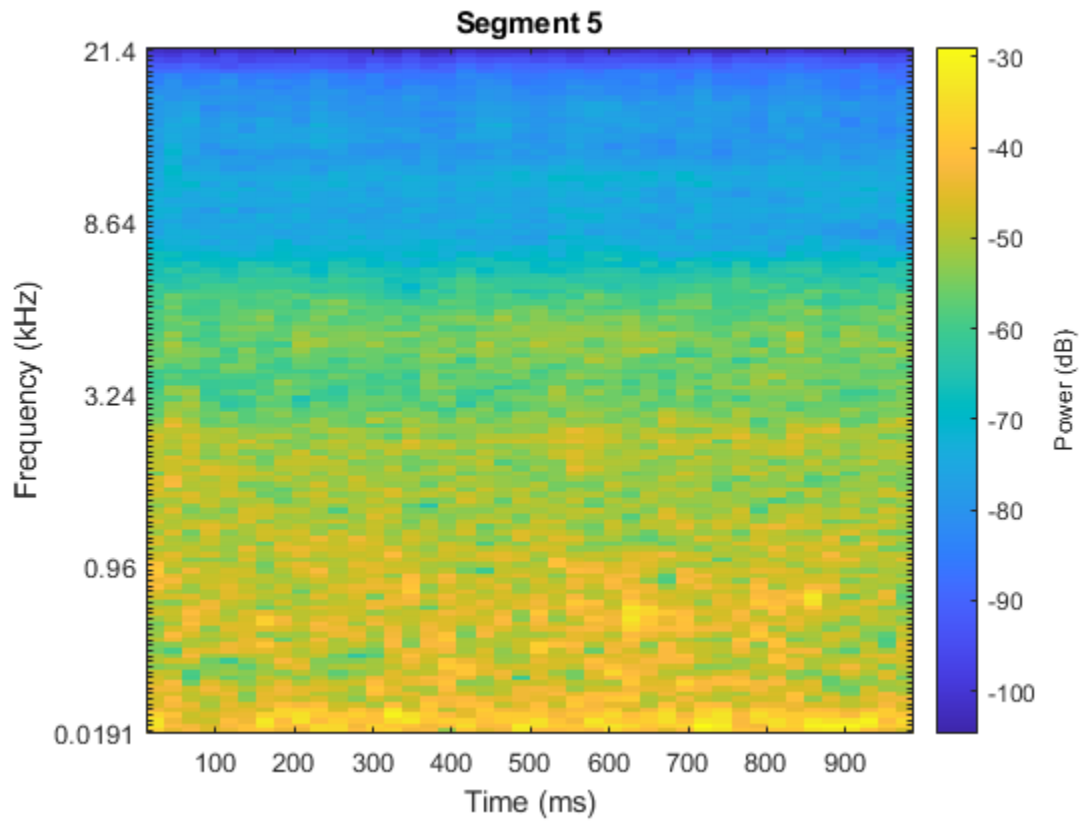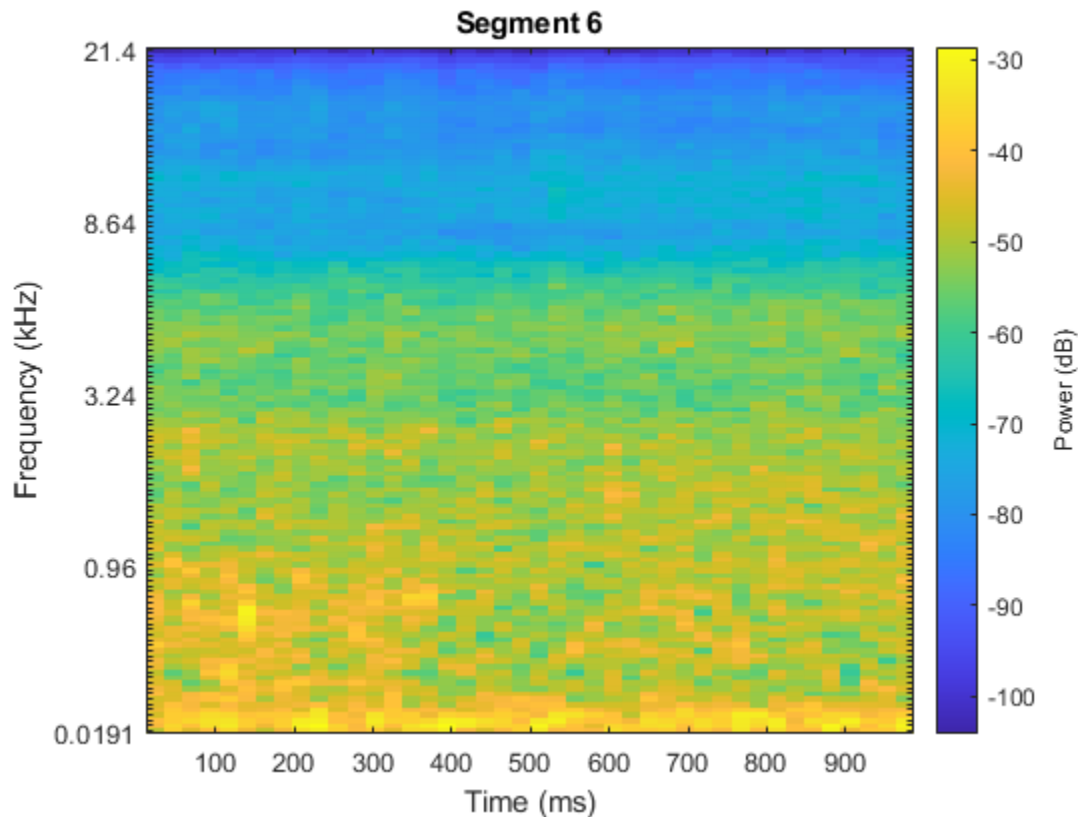
Segment 1

Segment 3

The helper function `HelperSegmentedMelSpectrograms` performs the feature extraction steps outlined above.

To speed up processing, extract mel spectrograms of all audio files in the datastores using `tall` arrays. Unlike in-memory arrays, tall arrays remain unevaluated until you request that the calculations be performed using the `gather` function. This deferred evaluation enables you to work quickly with large data sets. When you eventually request the output using `gather`, MATLAB combines the queued calculations where possible and takes the minimum number of passes through the data. If you have Parallel Computing Toolbox™, you can use tall arrays in your local MATLAB session, or on a local parallel pool. You can also run tall array calculations on a cluster if you have MATLAB® Parallel Server™ installed.

If you do not have Parallel Computing Toolbox™, the code in this example still runs.

```matlab
pp = parpool('IdleTimeout',inf);

train_set_tall = tall(adsTrain);
xTrain = cellfun(@(x)HelperSegmentedMelSpectrograms(x,fs, ...
    'SegmentLength',segmentLength, ...
    'SegmentOverlap',segmentOverlap, ...
    'WindowLength',windowLength, ...
    'HopLength',samplesPerHop, ...
    'NumBands',numBands, ...
    'FFTLength',fftLength), ...
    train_set_tall, ...
    'UniformOutput',false);
xTrain = gather(xTrain);
```

```
xTrain = cat(4,xTrain{:});

test_set_tall = tall(adsTest);
xTest = cellfun(@(x)HelperSegmentedMelSpectrograms(x,fs, ...
    'SegmentLength',segmentLength, ...
    'SegmentOverlap',segmentOverlap, ...
    'WindowLength',windowLength, ...
    'HopLength',samplesPerHop, ...
    'NumBands',numBands, ...
    'FFTLength',fftLength), ...
    test_set_tall, ...
    'UniformOutput',false);
xTest = gather(xTest);
xTest = cat(4,xTest{:});
```

```
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 6).
Evaluating tall expression using the Parallel Pool 'local':
- Pass 1 of 1: Completed in 4 min 48 sec
Evaluation completed in 4 min 48 sec
Evaluating tall expression using the Parallel Pool 'local':
- Pass 1 of 1: Completed in 1 min 40 sec
Evaluation completed in 1 min 40 sec
```
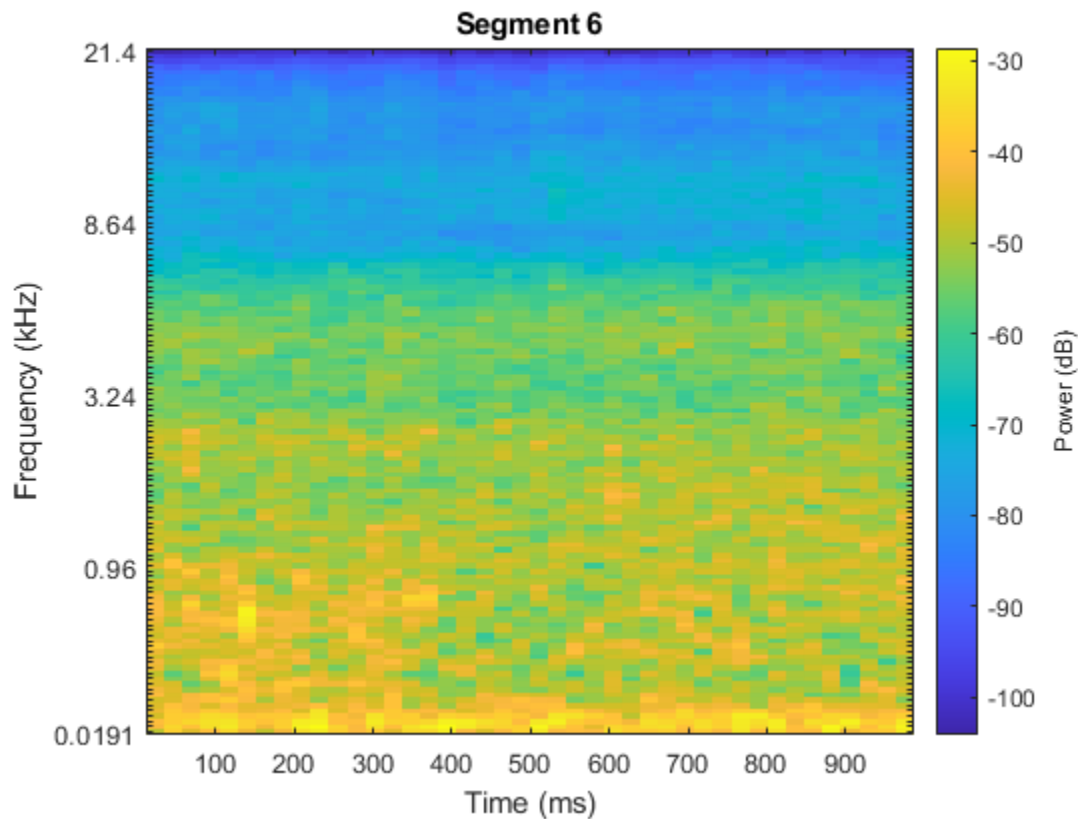

Replicate the labels of the training set so that they are in one-to-one correspondence with the segments.

```
numSegmentsPer10seconds = size(dataBuffered,2)/2;
yTrain = repmat(adsTrain.Labels,1,numSegmentsPer10seconds)';
yTrain = yTrain(:);
```

Data Augmentation for CNN

The DCASE 2017 dataset contains a relatively small number of acoustic recordings for the task, and the development set and evaluation set were recorded at different specific locations. As a result, it is easy to overfit to the data during training. One popular method to reduce overfitting is *mixup*. In mixup, you augment your dataset by mixing the features of two different classes. When you mix the features, you mix the labels in equal proportion. That is:

$$\tilde{x} = \lambda x_i + (1 - \lambda) x_j$$
$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j$$

Mixup was reformulated by [2] as labels drawn from a probability distribution instead of mixed labels. The implementation of mixup in this example is a simplified version of mixup: each spectrogram is mixed with a spectrogram of a different label with lambda set to 0.5. The original and mixed datasets are combined for training.

```
xTrainExtra = xTrain;
yTrainExtra = yTrain;
lambda = 0.5;
for i = 1:size(xTrain,4)

    % Find all available spectrograms with different labels.
    availableSpectrograms = find(yTrain~=yTrain(i));

    % Randomly choose one of the available spectrograms with a different label.
    numAvailableSpectrograms = numel(availableSpectrograms);
    idx = randi([1,numAvailableSpectrograms]);

    % Mix.
    xTrainExtra(:,:,:,i) = lambda*xTrain(:,:,:,i) + (1-lambda)*xTrain(:,:,:,availableSpectrograms

    % Specify the label as randomly set by lambda.
    if rand > lambda
        yTrainExtra(i) = yTrain(availableSpectrograms(idx));
    end
end
xTrain = cat(4,xTrain,xTrainExtra);
yTrain = [yTrain;yTrainExtra];
```

Call `summary` to display the distribution of labels for the augmented training set.

```
summary(yTrain)
```

```
    beach               11769
    bus                 11904
    cafe/restaurant     11873
    car                 11820
    city_center         11886
    forest_path         11936
    grocery_store       11914
    home                11923
    library             11817
    metro_station       11804
```

```
office                  11922
park                    11871
residential_area        11704
train                   11773
tram                    11924
```

Define and Train CNN

Define the CNN architecture. This architecture is based on [1] and modified through trial and error. See "List of Deep Learning Layers" (Deep Learning Toolbox) to learn more about deep learning layers available in MATLAB®.

```matlab
imgSize = [size(xTrain,1),size(xTrain,2),size(xTrain,3)];
numF = 32;
layers = [ ...
    imageInputLayer(imgSize)

    batchNormalizationLayer

    convolution2dLayer(3,numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,2*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,2*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,8*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,8*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    averagePooling2dLayer(ceil(imgSize(1:2)/8))

    dropoutLayer(0.5)

    fullyConnectedLayer(15)
```

```
        softmaxLayer
        classificationLayer];
```

Define `trainingOptions` (Deep Learning Toolbox) for the CNN. These options are based on [3] and modified through a systematic hyperparameter optimization workflow.

```
miniBatchSize = 128;
tuneme = 128;
lr = 0.05*miniBatchSize/tuneme;
options = trainingOptions('sgdm', ...
    'InitialLearnRate',lr, ...
    'MiniBatchSize',miniBatchSize, ...
    'Momentum',0.9, ...
    'L2Regularization',0.005, ...
    'MaxEpochs',8, ...
    'Shuffle','every-epoch', ...
    'Plots','training-progress', ...
    'Verbose',false, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropPeriod',2, ...
    'LearnRateDropFactor',0.2);
```

Call `trainNetwork` (Deep Learning Toolbox) to train the network.

```
trainedNet = trainNetwork(xTrain,yTrain,layers,options);
```

Evaluate CNN

Call `predict` (Deep Learning Toolbox) to predict responses from the trained network using the held-out test set.

```
cnnResponsesPerSegment = predict(trainedNet,xTest);
```

Average the responses over each 10-second audio clip.

```
classes = trainedNet.Layers(end).Classes;
numFiles = numel(adsTest.Files);

counter = 1;
cnnResponses = zeros(numFiles,numel(classes));
for channel = 1:numFiles
    cnnResponses(channel,:) = sum(cnnResponsesPerSegment(counter:counter+numSegmentsPer10seconds
    counter = counter + numSegmentsPer10seconds;
end
```

For each 10-second audio clip, choose the maximum of the predictions, then map it to the corresponding predicted location.

```
[~,classIdx] = max(cnnResponses,[],2);
cnnPredictedLabels = classes(classIdx);
```

Call `confusionchart` (Deep Learning Toolbox) to visualize the accuracy on the test set. Return the average accuracy to the Command Window.

```
figure
cm = confusionchart(adsTest.Labels,cnnPredictedLabels,'title','Test Accuracy - CNN');
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';
```

```
fprintf('Average accuracy of CNN = %0.2f\n',mean(adsTest.Labels==cnnPredictedLabels)*100)
```

Average accuracy of CNN = 75.12
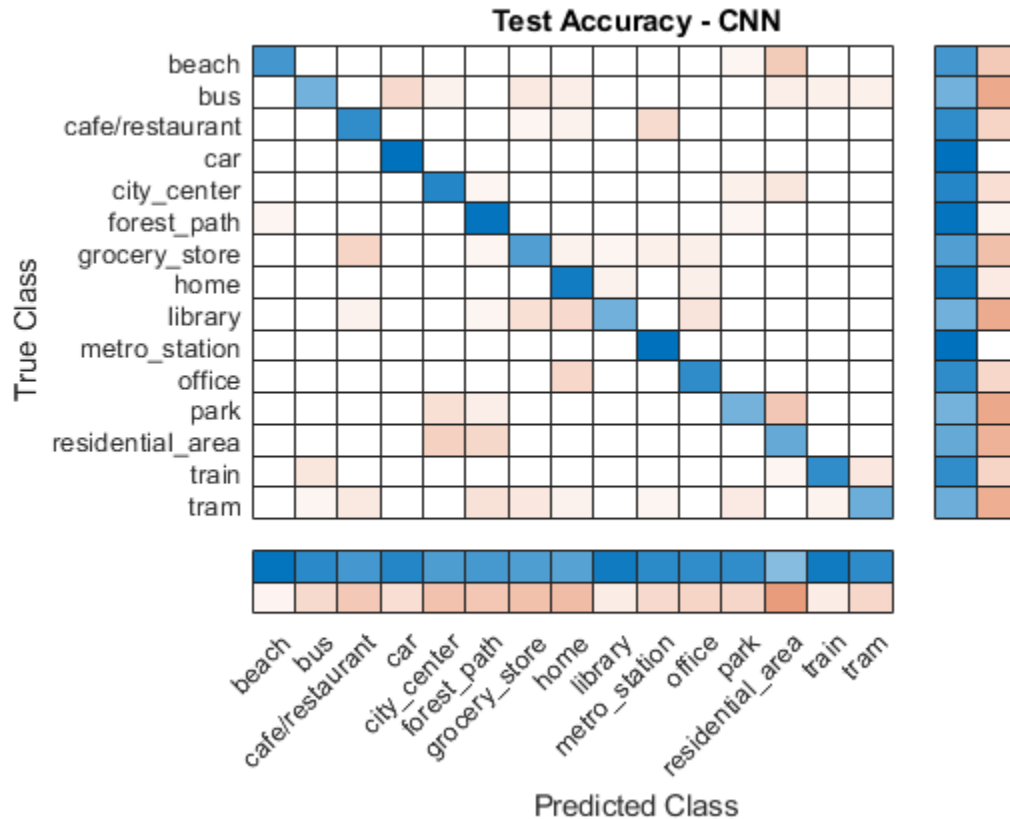
Feature Extraction for Ensemble Classifier

Wavelet scattering has been shown in [4] to provide a good representation of acoustic scenes. Define a `waveletScattering` object. The invariance scale and quality factors were determined through trial and error.

```
sf = waveletScattering('SignalLength',size(data,1), ...
                       'SamplingFrequency',fs, ...
                       'InvarianceScale',0.75, ...
                       'QualityFactors',[4 1]);
```

Convert the audio signal to mono, and then call `featureMatrix` to return the scattering coefficients for the scattering decomposition framework, `sf`.

```
dataMono = mean(data,2);
scatteringCoeffients = featureMatrix(sf,dataMono,'Transform','log');
```

Average the scattering coefficients over the 10-second audio clip.

```
featureVector = mean(scatteringCoeffients,2);
fprintf('Number of wavelet features per 10-second clip = %d\n',numel(featureVector))
```

Number of wavelet features per 10-second clip = 290

The helper function `HelperWaveletFeatureVector` performs the above steps. Use a `tall` array with `cellfun` and `HelperWaveletFeatureVector` to parallelize the feature extraction. Extract wavelet feature vectors for the train and test sets.

```
scatteringTrain = cellfun(@(x)HelperWaveletFeatureVector(x,sf),train_set_tall,'UniformOutput',fal
xTrain = gather(scatteringTrain);
xTrain = cell2mat(xTrain')';

scatteringTest = cellfun(@(x)HelperWaveletFeatureVector(x,sf),test_set_tall,'UniformOutput',false
xTest = gather(scatteringTest);
xTest = cell2mat(xTest')';
```

```
Evaluating tall expression using the Parallel Pool 'local':
- Pass 1 of 1: Completed in 30 min 22 sec
Evaluation completed in 30 min 22 sec
Evaluating tall expression using the Parallel Pool 'local':
- Pass 1 of 1: Completed in 10 min 30 sec
Evaluation completed in 10 min 30 sec
```

Define and Train Ensemble Classifier

Use `fitcensemble` to create a trained classification ensemble model (`ClassificationEnsemble`).

```
subspaceDimension = min(150,size(xTrain,2) - 1);
numLearningCycles = 30;
classificationEnsemble = fitcensemble(xTrain,adsTrain.Labels, ...
    'Method','Subspace', ...
    'NumLearningCycles',numLearningCycles, ...
    'Learners','discriminant', ...
    'NPredToSample',subspaceDimension, ...
    'ClassNames',removecats(unique(adsTrain.Labels)));
```

Evaluate Ensemble Classifier

For each 10-second audio clip, call `predict` to return the labels and the weights, then map it to the corresponding predicted location. Call `confusionchart` (Deep Learning Toolbox) to visualize the accuracy on the test set. Print the average.

```
[waveletPredictedLabels,waveletResponses] = predict(classificationEnsemble,xTest);

figure
cm = confusionchart(adsTest.Labels,waveletPredictedLabels,'title','Test Accuracy - Wavelet Scatte
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';

fprintf('Average accuracy of classifier = %0.2f\n',mean(adsTest.Labels==waveletPredictedLabels)*1
```

```
Average accuracy of classifier = 76.23
```

Test Accuracy - Wavelet Scattering
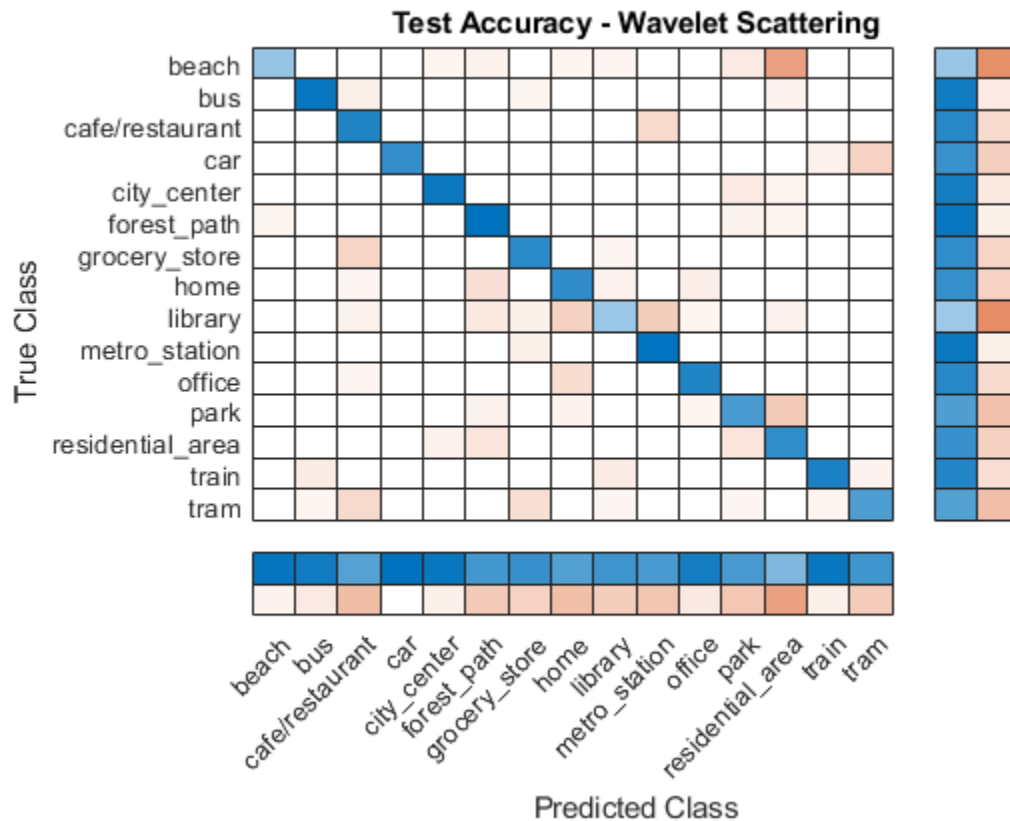
Apply Late Fusion

For each 10-second clip, calling predict on the wavelet classifier and the CNN returns a vector indicating the relative confidence in their decision. Multiply the waveletResponses with the cnnResponses to create a late fusion system.

```
fused = waveletResponses .* cnnResponses;
[~,classIdx] = max(fused,[],2);

predictedLabels = classes(classIdx);
```

Evaluate Late Fusion

Call confusionchart to visualize the fused classification accuracy. Print the average accuracy to the Command Window.

```
figure
cm = confusionchart(adsTest.Labels,predictedLabels,'title','Test Accuracy - Fusion');
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';

fprintf('Average accuracy of fused models = %0.2f\n',mean(adsTest.Labels==predictedLabels)*100)

Average accuracy of fused models = 79.57
```

13-131

Test Accuracy - Fusion
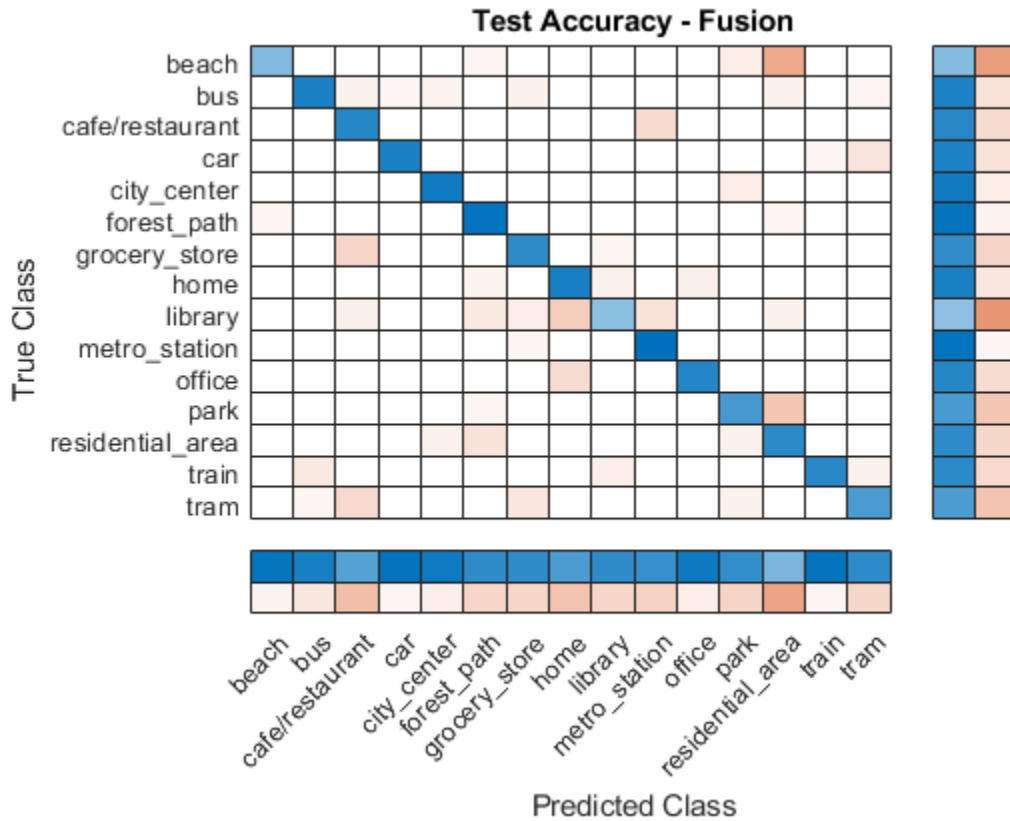
Close the parallel pool.

```
delete(pp)
```

Parallel pool using the 'local' profile is shutting down.

References

[1] A. Mesaros, T. Heittola, and T. Virtanen. Acoustic Scene Classification: an Overview of DCASE 2017 Challenge Entries. In proc. International Workshop on Acoustic Signal Enhancement, 2018.

[2] Huszar, Ferenc. "Mixup: Data-Dependent Data Augmentation." InFERENCe. November 03, 2017. Accessed January 15, 2019. https://www.inference.vc/mixup-data-dependent-data-augmentation/.

[3] Han, Yoonchang, Jeongsoo Park, and Kyogu Lee. "Convolutional neural networks with binaural representations and background subtraction for acoustic scene classification." the Detection and Classification of Acoustic Scenes and Events (DCASE) (2017): 1-5.

[4] Lostanlen, Vincent, and Joakim Anden. Binaural scene classification with wavelet scattering. Technical Report, DCASE2016 Challenge, 2016.

[5] A. J. Eronen, V. T. Peltonen, J. T. Tuomi, A. P. Klapuri, S. Fagerlund, T. Sorsa, G. Lorho, and J. Huopaniemi, "Audio-based context recognition," IEEE Trans. on Audio, Speech, and Language Processing, vol 14, no. 1, pp. 321-329, Jan 2006.

[6] TUT Acoustic scenes 2017, Development dataset

[7] TUT Acoustic scenes 2017, Evaluation dataset

Appendix -- Supporting Functions

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%HelperSegmentedMelSpectrograms
function X = HelperSegmentedMelSpectrograms(x,fs,varargin)

p = inputParser;
addParameter(p,'WindowLength',1024);
addParameter(p,'HopLength',512);
addParameter(p,'NumBands',128);
addParameter(p,'SegmentLength',1);
addParameter(p,'SegmentOverlap',0);
addParameter(p,'FFTLength',1024);
parse(p,varargin{:})
params = p.Results;

x = [sum(x,2),x(:,1)-x(:,2)];
x = x./max(max(x));

[xb_m,~] = buffer(x(:,1),round(params.SegmentLength*fs),round(params.SegmentOverlap*fs),'nodelay
[xb_s,~] = buffer(x(:,2),round(params.SegmentLength*fs),round(params.SegmentOverlap*fs),'nodelay
xb = zeros(size(xb_m,1),size(xb_m,2)+size(xb_s,2));
xb(:,1:2:end) = xb_m;
xb(:,2:2:end) = xb_s;

spec = melSpectrogram(xb,fs, ...
    'Window',hamming(params.WindowLength,'periodic'), ...
    'OverlapLength',params.WindowLength - params.HopLength, ...
    'FFTLength',params.FFTLength, ...
    'NumBands',params.NumBands, ...
    'FrequencyRange',[0,floor(fs/2)]);
spec = log10(spec+eps);

X = reshape(spec,size(spec,1),size(spec,2),size(x,2),[]);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%HelperWaveletFeatureVector
function features = HelperWaveletFeatureVector(x,sf)
x = mean(x,2);
features = featureMatrix(sf,x,'Transform','log');
features = mean(features,2);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

See Also
waveletScattering

More About
- "Wavelet Scattering" on page 9-2

GPU Acceleration of Scalograms for Deep Learning

This example shows how you can accelerate scalogram computation using GPUs. The computed scalograms are used as the input features to deep convolution neural networks (CNN) for ECG and spoken digit classification.

You must have a CUDA-enabled NVIDIA GPU with compute capability 3.0 or higher. See "GPU Support by Release" (Parallel Computing Toolbox) for details. The audio section of this example requires Audio Toolbox™ to use the audio datastore and transformed datastore.

Scalogram Computation Using GPU

The most efficient way to compute scalograms on the GPU is to use `cwtfilterbank`. The steps to compute scalograms on the GPU are:

1   Construct `cwtfilterbank` with the desired property settings.
2   Move the signal to the GPU using `gpuArray`.
3   Use the filter bank `WT` method to compute the continuous wavelet transform (CWT).

The first time you use the `WT` method, `cwtfilterbank` caches the wavelet filters on the GPU. As a result, substantial time savings in computation are realized when you obtain scalograms of multiple signals using the same filter bank and same datatype. The following demonstrates the recommended workflow. As an example, use a sample of guitar music containing 661,500 samples.

```
[y,fs] = audioread('guitartune.wav');
plot(y)
grid on
```
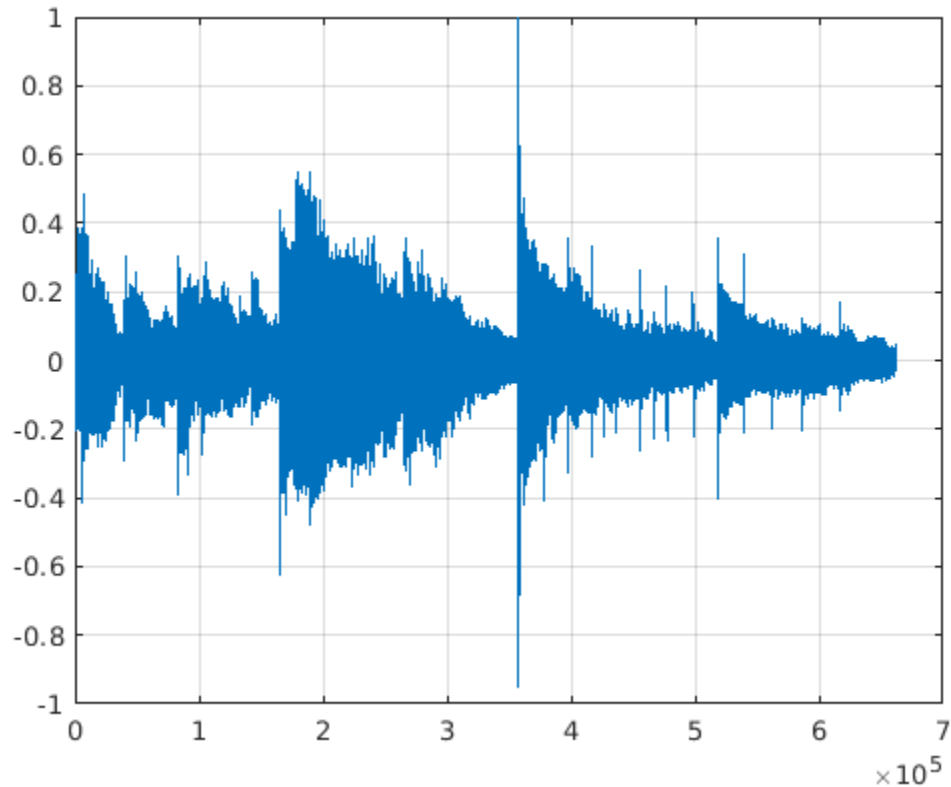
Because most NVIDIA GPUs are significantly more efficient with single rather than double-precision data, cast the signal to single precision.

```
y = single(y);
```

Construct the filter bank to match the signal length and the sampling frequency. For deep learning, the sampling frequency is normally not necessary and therefore can be excluded.

```
fb = cwtfilterbank('SignalLength',length(y),'SamplingFrequency',fs);
```

Finally move the signal to the GPU using `gpuArray` and compute the CWT of the data. Plot the resulting scalogram.

```
[cfs,f] = fb.wt(gpuArray(y));
t = 0:1/fs:(length(y)*1/fs)-1/fs;
imagesc(t,f,abs(cfs))
axis xy
ylabel('Hz')
xlabel('Seconds')
```
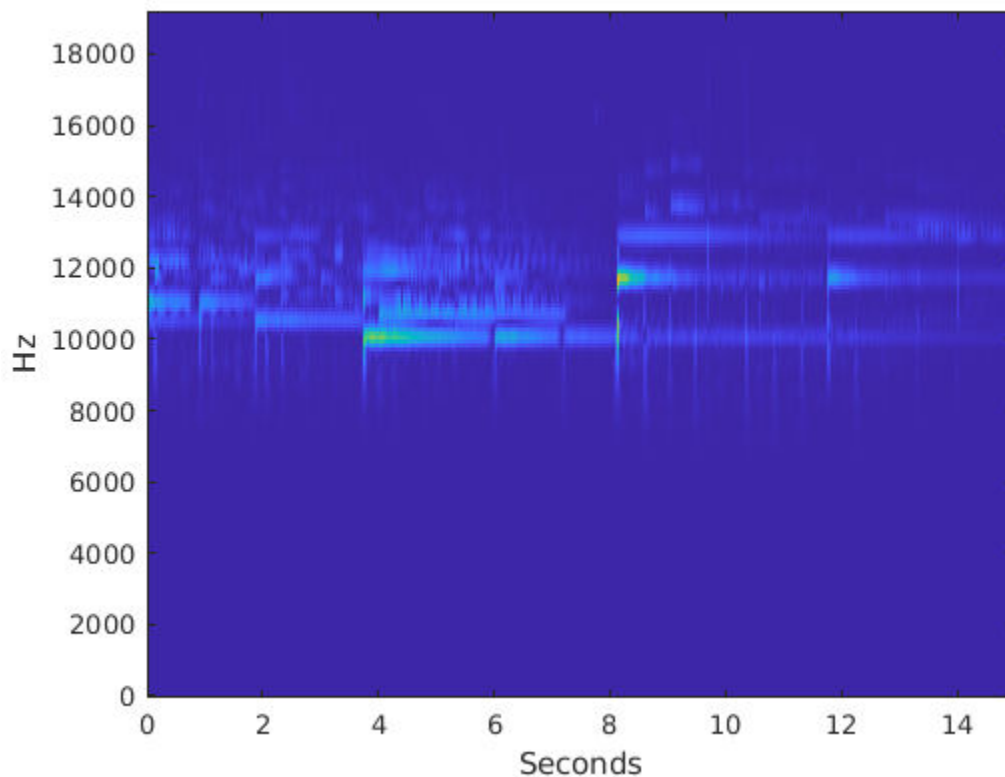
Use `gather` to bring the CWT coefficients and any other outputs back to the CPU.

```
cfs = gather(cfs);
f = gather(f);
```

To demonstrate the efficiency gained in using the GPU, time the CWT computation on the GPU and CPU for the same signal. The GPU compute times reported here are obtained using an NVIDIA Titan V with a compute capability of 7.0.

```
ygpu = gpuArray(y);
fgpu = @()fb.wt(ygpu);
Tgpu = gputimeit(fgpu)
```

```
Tgpu = 0.2658
```

Repeat the same measurement on the CPU and examine the ratio of GPU to CPU time to see the reduction in computation time.

```
fcpu = @()fb.wt(y);
Tcpu = timeit(fcpu)
```

```
Tcpu = 3.7088
```

```
Tcpu/Tgpu
```

```
ans = 13.9533
```

Scalograms in Deep Learning

A common application of the CWT in deep learning is to use the scalogram of a signal as the input "image" to a deep CNN. This necessarily mandates the computation of multiple scalograms, one for each signal in the training, validation, and test sets. While GPUs are often used to speed up training and inference in the deep network, it is also beneficial to use GPUs to accelerate any feature extraction, or data preprocessing needed to make the deep network more robust.

To illustrate this, the following section applies wavelet scalograms to human electrocardiogram (ECG) classification. Scalograms are used with the same data treated in "Classify Time Series Using Wavelet Analysis and Deep Learning" on page 13-79. In that example, transfer learning with GoogLeNet and SqueezeNet was used to classify ECG waveforms into one of three categories. The description of the data and how to obtain it is repeated here for convenience.

ECG Data Description and Download

The ECG data is obtained from three groups of people: persons with cardiac arrhythmia (ARR), persons with congestive heart failure (CHF), and persons with normal sinus rhythms (NSR). In total there are 162 ECG recordings from three PhysioNet databases: MIT-BIH Arrhythmia Database [2][3], MIT-BIH Normal Sinus Rhythm Database [3], and The BIDMC Congestive Heart Failure Database [1] [3]. More specifically, 96 recordings from persons with arrhythmia, 30 recordings from persons with congestive heart failure, and 36 recordings from persons with normal sinus rhythms. The goal is to train a model to distinguish between ARR, CHF, and NSR.

You can obtain this data from the MathWorks GitHub repository. To download the data from the website, click `Clone or download` and select `Download ZIP`. Save the file `physionet_ECG_data-master.zip` in a folder where you have write permission. The instructions for this example assume you have downloaded the file to your temporary directory, `tempdir`, in MATLAB. Modify the subsequent instructions for unzipping and loading the data if you choose to download the data in folder different from `tempdir`. If you are familiar with Git, you can download the latest version of the tools (git) and obtain the data from a system command prompt using `git clone https://github.com/mathworks/physionet_ECG_data/`.

After downloading the data from GitHub, unzip the file in your temporary directory.

```
unzip(fullfile(tempdir,'physionet_ECG_data-master.zip'),tempdir)
```

Unzipping creates the folder `physionet-ECG_data-master` in your temporary directory. This folder contains the text file `README.md` and `ECGData.zip`. The `ECGData.zip` file contains

- `ECGData.mat`
- `Modified_physionet_data.txt`
- `License.txt`

`ECGData.mat` holds the data used in this example. The text file, `Modified_physionet_data.txt`, is required by PhysioNet's copying policy and provides the source attributions for the data as well as a description of the preprocessing steps applied to each ECG recording.

Unzip `ECGData.zip` in `physionet-ECG_data-master`. Load the data file into your MATLAB workspace.

```
unzip(fullfile(tempdir,'physionet_ECG_data-master','ECGData.zip'),...
    fullfile(tempdir,'physionet_ECG_data-master'))
load(fullfile(tempdir,'physionet_ECG_data-master','ECGData.mat'))
```

ECGData is a structure array with two fields: Data and Labels. The Data field is a 162-by-65536 matrix where each row is an ECG recording sampled at 128 hertz. Labels is a 162-by-1 cell array of diagnostic labels, one for each row of Data. The three diagnostic categories are: 'ARR', 'CHF', and 'NSR'. Use the helper function, helperRandomSplit, to split the data into training and validation sets with 80% of the data allocated for training and 20% for validation. Convert the ECG diagnostic labels into categoricals.

```
[trainData, validationData, trainLabels, validationLabels] = helperRandomSplit(80,ECGData);
trainLabels = categorical(trainLabels);
validationLabels = categorical(validationLabels);
```

There are 130 records in the trainData set and 32 records in validationData. By design, the training data contains 80.25% (130/162) of the data. Recall that the ARR class represents 59.26% of the data (96/162), the CHF class represents 18.52% (30/162), and the NSR class represents 22.22% (36/162). Examine the percentage of each class in the training and test sets. The percentages in each are consistent with the overall class percentages in the data set.

```
Ctrain = countcats(trainLabels)./numel(trainLabels).*100
```

```
Ctrain = 3×1

   59.2308
   18.4615
   22.3077
```

```
Cvalid = countcats(validationLabels)./numel(validationLabels).*100
```

```
Cvalid = 3×1

   59.3750
   18.7500
   21.8750
```

Scalograms With Deep CNN — ECG Data

Scalogram Computation on the GPU

Compute the scalograms for both the training and validation sets. Set useGPU to true to use the GPU and false to compute the scalograms on the CPU. To mitigate the effect of large input matrices on the CNN and create more training and validation examples, helperECGScalograms splits each ECG waveform into four nonoverlapping segments of 16384 samples each and computes scalograms for all four segments. Replicate the labels to match the expanded dataset. In this case, obtain an estimate of the expended computation time.

```
frameLength = 16384;
useGPU = true;
tic;
Xtrain = helperECGScalograms(trainData,frameLength,useGPU);

Computing scalograms...
Processed 50 files out of 130
Processed 100 files out of 130
...done

T = toc;
sprintf('Elapsed time is %1.2f seconds',T)
```

```
ans =
'Elapsed time is 4.22 seconds'

trainLabels = repelem(trainLabels,4);
```

With the Titan V GPU, 502 scalograms have been computed in approximately 4.2 seconds. Setting useGPU to false and repeating the above computation demonstrates the speed up obtained by using the GPU. In this case, using the CPU required 33.3 seconds to compute the scalograms. The GPU computation was more than 7 times faster.

Repeat the same process for the validation data.

```
useGPU = true;
Xvalid = helperECGScalograms(validationData,frameLength,useGPU);

Computing scalograms...
...done

validationLabels = repelem(validationLabels,4);
```

Next set up a deep CNN to process both the training and validation sets. The simple network used here is not optimized. This CNN is only used to illustrate the end-to-end workflow for cases where the scalograms fit in memory.

```
sz = size(Xtrain);
specSize = sz(1:2);
imageSize = [specSize 1];
dropoutProb = 0.3;

layers = [
    imageInputLayer(imageSize)

    convolution2dLayer(3,12,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)


    convolution2dLayer(3,20,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)


    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer
    dropoutLayer(dropoutProb)
    fullyConnectedLayer(3)
    softmaxLayer
    classificationLayer];
```

Use the following training options.

```
options = trainingOptions('sgdm',...
    'InitialLearnRate', 1e-4,...
```

```
        'LearnRateDropPeriod',18,...
        'MiniBatchSize', 20,...
        'MaxEpochs',25,...
        'L2Regularization',1e-1,...
        'Plots', 'training-progress',...
        'Verbose',false,...
        'Shuffle','every-epoch',...
        'ExecutionEnvironment','auto',...
        'ValidationData',{Xvalid,validationLabels});
```

Train the network and measure the validation error.

```
trainNetwork(Xtrain,trainLabels,layers,options);
```
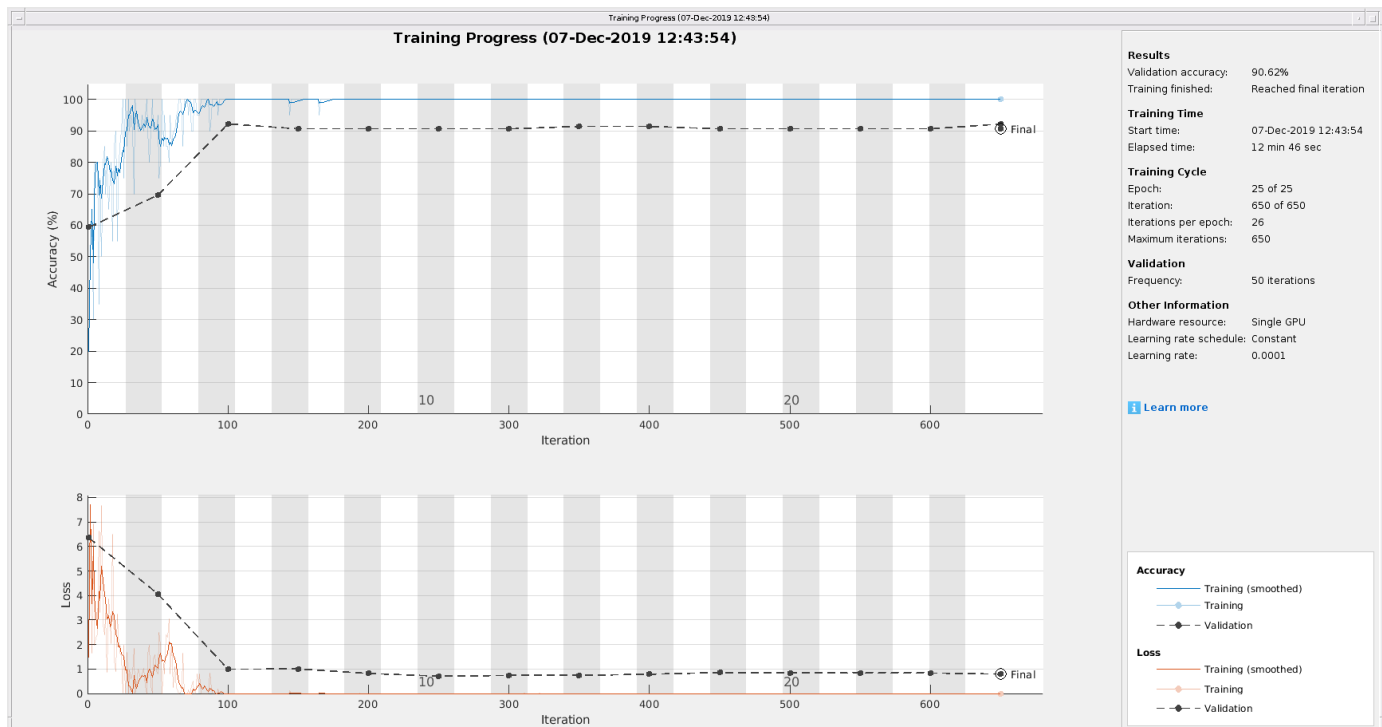

Even though the simple CNN used here is not optimized, the validation accuracy is consistently in the high 80 to low 90 percent range. This is comparable to the validation accuracy achieved with the more powerful and optimized SqueezeNet shown in "Classify Time Series Using Wavelet Analysis and Deep Learning" on page 13-79 example. Further, this is a much more efficient use of the scalogram, because in that example the scalograms had to be rescaled as RGB images compatible with SqueezeNet, saved to disk in an appropriate image format, and then fed to the deep network using `imageDatastore`.

Spoken Digit Recognition — GPU Computing using Transform Datastore

This section shows how to accelerate scalogram computation using GPU in a transformed datastore workflow.

Data

Clone or download the Free Spoken Digit Dataset (FSDD), available at https://github.com/Jakobovski/ free-spoken-digit-dataset. FSDD is an open data set, which means that it can grow over time. This

example uses the version committed on 01/29/2019 which consists of 2000 recordings of the English digits 0 through 9 obtained from four speakers. Two of the speakers in this version are native speakers of American English and two speakers are nonnative speakers of English with a Belgium French and German accent. The data is sampled at 8000 Hz.

For other approaches to this dataset including wavelet scattering, see "Spoken Digit Recognition with Wavelet Scattering and Deep Learning" on page 13-61.

Use `audioDatastore` to manage data access and ensure the random division of the recordings into training and test sets. Set the `location` property to the location of the FSDD recordings folder on your computer, for example:

```
pathToRecordingsFolder = '/home/user/free-spoken-digit-dataset/recordings';
location = pathToRecordingsFolder;
```

Point `audioDatastore` to that location.

```
ads = audioDatastore(location);
```

The helper function `helpergenLabels` creates a categorical array of labels from the FSDD files. List the classes and the number of examples in each class.

```
ads.Labels = helpergenLabels(ads);
summary(ads.Labels)
```

```
     0      200
     1      200
     2      200
     3      200
     4      200
     5      200
     6      200
     7      200
     8      200
     9      200
```

Transformed Datastore

First split the FSDD into training and test sets. Allocate 80% of the data to the training set and retain 20% for the test set.

```
rng default;
ads = shuffle(ads);
[adsTrain,adsTest] = splitEachLabel(ads,0.8);
countEachLabel(adsTrain)
```

```
ans=10×2 table
    Label     Count
    _____     _____

      0        160
      1        160
      2        160
      3        160
      4        160
      5        160
      6        160
      7        160
```

```
8      160
9      160
```

Next create the CWT filter bank and the transformed datastores for both the training and test data using the helper function, `helperDigitScalogram`. The transformed datastore converts each recording into a signal of length 8192, computes the scalogram on the GPU, and gathers the data back onto the CPU.

```
reset(gpuDevice(1))
fb = cwtfilterbank('SignalLength',8192);
adsSCTrain = transform(adsTrain,@(audio,info)helperDigitScalogram(audio,info,fb),'IncludeInfo',tr
adsSCTest = transform(adsTest,@(audio,info)helperDigitScalogram(audio,info,fb),'IncludeInfo',true
```

Deep CNN

Construct a deep CNN to train with the transformed datastore, `adscTrain`. As in the first example, the network is not optimized. The point is to show the workflow using scalograms computed on the GPU for out-of-memory data.

```
numClasses = 10;

dropoutProb = 0.2;
numF = 12;
layers = [
    imageInputLayer([101 8192 1])

    convolution2dLayer(5,numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,2*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2)

    dropoutLayer(dropoutProb)
    fullyConnectedLayer(numClasses)
    softmaxLayer
```

```
    classificationLayer('Classes',categories(ads.Labels));
    ];
```

Set the training options for the network.

```
miniBatchSize = 25;
options = trainingOptions('adam', ...
    'InitialLearnRate',1e-4, ...
    'MaxEpochs',30, ...
    'MiniBatchSize',miniBatchSize, ...
    'Shuffle','every-epoch', ...
    'Plots', 'training-progress',...
    'Verbose',false,...
    'ExecutionEnvironment','gpu');
```

Train the network.

```
trainedNet = trainNetwork(adsSCTrain,layers,options);
```
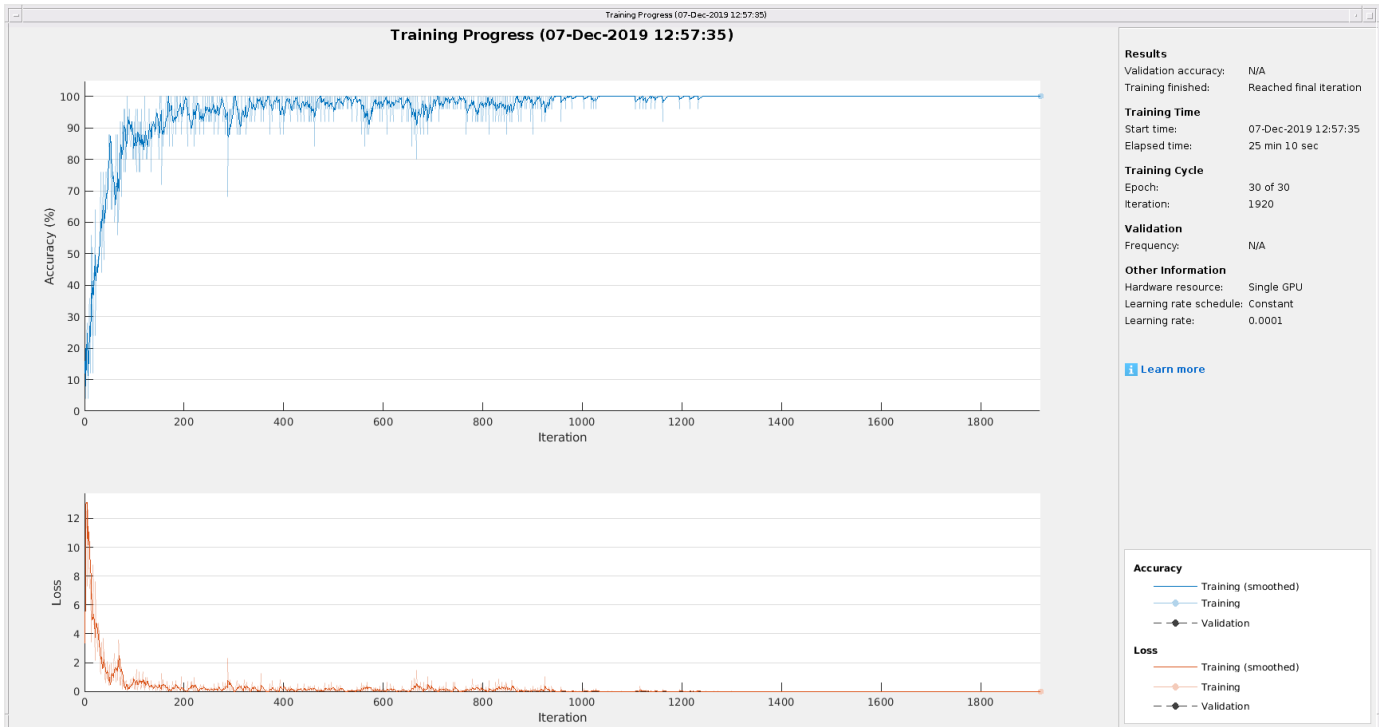

In this instance, training was completed in 25 minutes and 10 seconds. If you comment out the call to `gpuArray` in `helperDigitScalogram` and use the CPU to obtain the scalograms, training time increases significantly. In this case, an increase from 25 minutes and 10 seconds to 45 minutes and 38 seconds was observed.

Use the trained network to predict the digit labels for the test set.

```
Ypredicted = classify(trainedNet,adsSCTest,'ExecutionEnvironment','CPU');
cnnAccuracy = sum(Ypredicted == adsTest.Labels)/numel(Ypredicted)*100
```

```
cnnAccuracy = 96.2500
```

The inference time using the GPU was approximately 22 seconds. Using the CPU, inference time doubled to 45 seconds.

The performance of trained network on the test data is close to 96%. This is comparable to the performance in "Spoken Digit Recognition with Wavelet Scattering and Deep Learning" on page 13-61.

Summary

This example has showcased how to use the GPU to accelerate scalogram computation. The example presented the optimal workflow for efficiently computing scalograms both for in-memory data and for out-of-memory data read from disk using transformed datastores.

References

1 Baim, D. S., W. S. Colucci, E. S. Monrad, H. S. Smith, R. F. Wright, A. Lanoue, D. F. Gauthier, B. J. Ransil, W. Grossman, and E. Braunwald. "Survival of patients with severe congestive heart failure treated with oral milrinone." *Journal of the American College of Cardiology*. Vol. 7, Number 3, 1986, pp. 661–670.

2 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, and H. E. Stanley. "PhysioBank, PhysioToolkit,and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals." Circulation. Vol. 101, Number 23: e215–e220. [Circulation Electronic Pages; `http://circ.ahajournals.org/content/101/23/e215.full`]; 2000 (June 13). doi: 10.1161/01.CIR.101.23.e215.

3 Moody, G. B., and R. G. Mark. "The impact of the MIT-BIH Arrhythmia Database." *IEEE Engineering in Medicine and Biology Magazine*. Vol. 20. Number 3, May-June 2001, pp. 45–50. (PMID: 11446209)

```matlab
function Labels = helpergenLabels(ads)
% This function is only for use in Wavelet Toolbox examples. It may be
% changed or removed in a future release.
tmp = cell(numel(ads.Files),1);
expression = "[0-9]+_";
for nf = 1:numel(ads.Files)
    idx = regexp(ads.Files{nf},expression);
    tmp{nf} = ads.Files{nf}(idx);
end
Labels = categorical(tmp);

end

function X = helperECGScalograms(data,window,useGPU)

disp("Computing scalograms...");
Nsig = size(data,1);
Nsamp = size(data,2);
Nsegment = Nsamp/window;

fb = cwtfilterbank('SignalLength',window,'Voices',10);
Ns = length(fb.Scales);
X = zeros([Ns,window,1,Nsig*Nsegment],'single');
start = 0;
if useGPU
    data = gpuArray(single(data'));
else
    data = single(data');
```

```matlab
    end
for ii = 1:Nsig
    ts = data(:,ii);
    ts = reshape(ts,window,Nsegment);
    ts = (ts-mean(ts))./max(abs(ts));

    for kk = 1:size(ts,2)
        cfs = fb.wt(ts(:,kk));
        X(:,:,1,kk+start) = gather(abs(cfs));

    end
    start = start+Nsegment;

    if mod(ii,50) == 0
        disp("Processed " + ii + " files out of " + Nsig)
    end

end

disp("...done");
data = gather(data);

end

function [x,info] = helperReadSPData(x,info)
% This function is only for use Wavelet Toolbox examples. It may change or
% be removed in a future release.

N = numel(x);
if N > 8192
    x = x(1:8192);
elseif N < 8192
    pad = 8192-N;
    prepad = floor(pad/2);
    postpad = ceil(pad/2);
    x = [zeros(prepad,1) ; x ; zeros(postpad,1)];
end
x = x./max(abs(x));

end

function [dataout,info] = helperDigitScalogram(audioin,info,fb)
audioin = single(audioin);
audioin = gpuArray(audioin);
audioin = helperReadSPData(audioin);
cfs = gather(abs(fb.wt(audioin)));
audioin = gather(audioin);
dataout = {cfs,info.Label};
end
```

Deploy Signal Classifier Using Wavelets and Deep Learning on Raspberry Pi

This example shows the workflow to classify human electrocardiogram (ECG) signals using the Continuous Wavelet Transform (CWT) and a deep convolutional neural network (CNN). This example also provides information on how to generate and deploy the code and CNN for prediction on a Raspberry Pi target (ARM®-based device).

SqueezeNet is a deep CNN originally designed to classify images in 1000 categories. In the example "Classify Time Series Using Wavelet Analysis and Deep Learning" on page 13-79, SqueezeNet is retrained to classify ECG waveforms based on their *scalograms*. A scalogram is a time-frequency representation of the signal and is the absolute value of the CWT of the signal. We reuse the retrained SqueezeNet in this example.

ECG Data Description

In this example, ECG data from PhysioNet is used. The ECG data is obtained from three groups of people: persons with cardiac arrhythmia (ARR), persons with congestive heart failure (CHF), and persons with normal sinus rhythms (NSR). The data set includes 96 recordings from persons with ARR, 30 recordings from persons with CHF, and 36 recordings from persons with NSR. The 162 ECG recordings are from three PhysioNet databases: MIT-BIH Arrhythmia Database [2][3], MIT-BIH Normal Sinus Rhythm Database [3], and The BIDMC Congestive Heart Failure Database [1][3]. Shortened ECG data of the above references can be downloaded from the GitHub repository.

Prerequisites

- ARM processor that supports the NEON extension
- ARM Compute Library version 19.05 (on the target ARM hardware)
- Environment variables for the compilers and libraries
- MATLAB Support Package for Raspberry Pi Hardware
- MATLAB Coder Interface for Deep Learning Libraries support package

For supported versions of libraries and for information about setting up environment variables, see "Prerequisites for Deep Learning with MATLAB Coder" (MATLAB Coder). This example is not supported in MATLAB Online™.

Functionality of Generated Code

The core function in the generated executable, `processECG`, uses 65,536 samples of single-precision ECG data as input. The function:

1 Takes the CWT of the ECG data.
2 Obtains scalogram from wavelet coefficients.
3 Converts the scalogram to an RGB mage of dimension 227-by-227-by-3. This makes the image compatible with the SqueezeNet network architecture.
4 Performs prediction to classify the image using SqueezeNet.

```
type processECG

function [YPred] = processECG(input)
% processECG function - converts 1D ECG to image and predicts the syndrome
% of heart disease
```

```matlab
%
% This function is only intended to support the example:
% Signal Classification Code Generation Using Wavelets and
% Deep Learning on Raspberry Pi. It may change or be removed in a
% future release.

% Copyright 2020 The MathWorks, Inc.

    % colourmap for image transformation
    persistent net jetdata;
    if(isempty(jetdata))
        jetdata = colourmap(128,class(input));
    end

    % Squeezenet trained network
    if(isempty(net))
        net = coder.loadDeepLearningNetwork('trainedNet.mat');
    end

    % Wavelet Transformation & Image conversion
    cfs = ecg_to_Image(input);
    image = ind2rgb(im2uint8(rescale(cfs)),single(jetdata));
    image = im2uint8(imresize(image,[227,227]));

    % figure
    if isempty(coder.target)
        imshow(image);
    end

    % Prediction
    [YPred] = predict(net,image);


    %% ECG to image conversion
    function cfs = ecg_to_Image(input)

        %Wavelet Transformation
        persistent filterBank
        [~,siglen] = size(input);
        if isempty(filterBank)
            filterBank = cwtfilterbank('SignalLength',siglen,'VoicesPerOctave',6);
        end
        %CWT conversion
        cfs = abs(filterBank.wt(input));
    end


    %% Colourmap
    function J = colourmap(m,class)

        n = ceil(m/4);
        u = [(1:1:n)/n ones(1,n-1) (n:-1:1)/n]';
        g = ceil(n/2) - (mod(m,4)==1) + (1:length(u))';
        r = g + n;
        b = g - n;
        r1 = r(r<=128);
        g1 = g(g<=128);
        b1 = b(b >0);
```

```
        J = zeros(m,3);
        J(r1,1) = u(1:length(r1));
        J(g1,2) = u(1:length(g1));
        J(b1,3) = u(end-length(b1)+1:end);
        feval = str2func(class);
        J = feval(J);
    end
end
```

Create Code Generation Configuration Object

Create a code generation configuration object for generation of an executable program. Specify generation of C++ code.

```
cfg = coder.config('exe');
cfg.TargetLang = 'C++';
```

Set Up Configuration Object for Deep Learning Code Generation

Create a `coder.ARMNEONConfig` object. Specify the same version of the ARM Compute library as the one on the Raspberry Pi. Specify the architecture of the Raspberry Pi.

```
dlcfg = coder.DeepLearningConfig('arm-compute');
dlcfg.ArmComputeVersion = '19.05';
dlcfg.ArmArchitecture = 'armv7';
```

Attach Deep Learning Configuration Object to Code Generation Configuration Object

Set the `DeepLearningConfig` property of the code generation configuration object to the deep learning configuration object. Make the MATLAB Source Comments visible in the configuration object at the time of code generation.

```
cfg.DeepLearningConfig = dlcfg;
cfg.MATLABSourceComments = 1;
```

Create a Connection to the Raspberry Pi

Use the MATLAB Support Package for Raspberry Pi Support Package function, `raspi`, to create a connection to the Raspberry Pi. In the following code, replace:

- `'raspiname'` with the name of your Raspberry Pi
- `'pi'` with your user name
- `'password'` with your password

```
r = raspi('172.18.76.69','pi','raspberry');
```

Configure Code Generation Hardware Parameters for Raspberry Pi

Create a `coder.Hardware` object for Raspberry Pi and attach it to the code generation configuration object.

```
hw = coder.hardware('Raspberry Pi');
cfg.Hardware = hw;
```

Specify the build folder on the Raspberry Pi.

```
buildDir = '~/remdirECG';
cfg.Hardware.BuildDir = buildDir;
```

Provide C++ Main File for Code Execution

The C++ main file reads the input ECG data, calls the `processECG` function to perform preprocessing and deep learning using CNN on the ECG data, and displays the classification probability.

Specify the main file in the code generation configuration object. To learn more about generating and customizing `main_ecg_raspi.cpp`, refer to "Generating Standalone C/C++ Executables from MATLAB Code" (MATLAB Coder).

```
cfg.CustomSource = 'main_ecg_raspi.cpp';
```

Generate Source C++ Code Using codegen

Use the `codegen` function to generate the C++ code. When `codegen` is used with the MATLAB Support Package for Raspberry Pi Hardware, the executable is built on the Raspberry Pi board.

Make sure to set the environment variables `ARM_COMPUTELIB` and `LD_LIBRARY_PATH` on the Raspberry Pi. See "Prerequisites for Deep Learning with MATLAB Coder" (MATLAB Coder).

```
codegen -config cfg processECG -args {ones(1,65536,'single')} -d arm_compute
```

```
 Deploying code. This may take a few minutes.
```

Fetch Generated Executable Directory

To test the generated code on the Raspberry Pi, copy the input ECG signal to the generated code directory. You can find this directory manually or by using the `raspi.utils.getRemoteBuildDirectory` API. This function lists the directories of the binary files that are generated by using `codegen`.

```
applicationDirPaths = raspi.utils.getRemoteBuildDirectory('applicationName','processECG')
```

```
applicationDirPaths=1×4 cell array
    {1×1 struct}    {1×1 struct}    {1×1 struct}    {1×1 struct}
```

The complete path to the remote build directory is derived from the present working directory. If you do not know which `applicationDirPaths` entry contains the generated code, use the helper function `helperFindTargetDir`. Otherwise, specify the proper directory.

```
directoryUnknown = true;

if directoryUnknown
    targetDirPath = helperFindTargetDir(applicationDirPaths);
else
    targetDirPath = applicationDirPaths{1}.directory;
end
```

Copy Input File to Raspberry Pi

The text file `input_ecg_raspi.csv` contains the ECG samples of a representative ARR signal. To copy the file required to run the executable program, use `putFile`, which is available with the MATLAB Support Package for Raspberry Pi Hardware.

```
r.putFile('input_ecg_raspi.csv', targetDirPath);
```

For a pictorial representation, the first 1000 samples can be plotted by using these steps.

```
input = dlmread('input_ecg_raspi.csv');
plot(input(1:1000))
title('ARR Signal')
```

ARR Signal
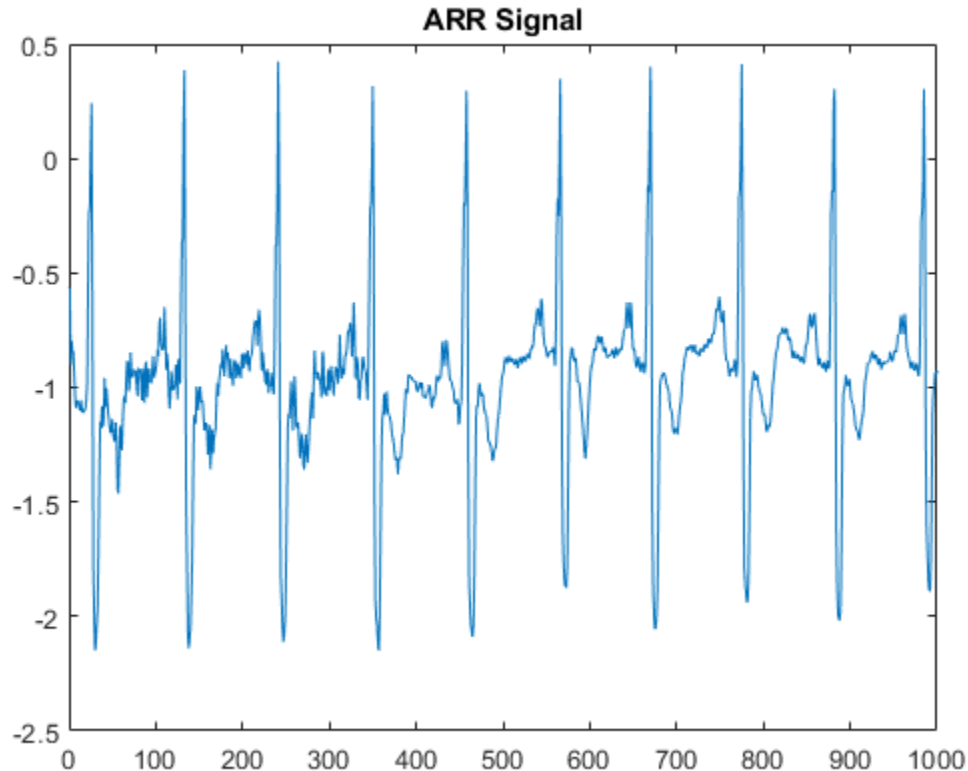
Run Executable on Raspberry Pi

Run the executable program on the Raspberry Pi from MATLAB and direct the output back to MATLAB. The input file name is passed as the command line argument for the executable.

```
exeName = 'processECG.elf';          % executable name
fileName = 'input_ecg_raspi.csv';    % Input ECG file that is pushed to target
command = ['cd ' targetDirPath ';./' exeName ' ' fileName];
output = system(r,command)

output =
    'Predicted Values on the Target Hardware
     ARR          CHF          NSR
     0.806078    0.193609    0.000313103
     '
```

References

1   Baim, D. S., W. S. Colucci, E. S. Monrad, H. S. Smith, R. F. Wright, A. Lanoue, D. F. Gauthier, B. J. Ransil, W. Grossman, and E. Braunwald. "Survival of patients with severe congestive heart failure treated with oral milrinone." *Journal of the American College of Cardiology*. Vol. 7, Number 3, 1986, pp. 661–670.

2 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, and H. E. Stanley. "PhysioBank, PhysioToolkit,and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals." Circulation. Vol. 101, Number 23: e215–e220. [Circulation Electronic Pages; `http://circ.ahajournals.org/content/101/23/e215.full`]; 2000 (June 13). doi: 10.1161/01.CIR.101.23.e215.

3 Moody, G. B., and R. G. Mark. "The impact of the MIT-BIH Arrhythmia Database." *IEEE Engineering in Medicine and Biology Magazine*. Vol. 20. Number 3, May-June 2001, pp. 45–50. (PMID: 11446209)

Supporting Functions

helperFindTargetDir

```matlab
function targetDir = helperFindTargetDir(dirPaths)
%
% This function is only intended to support wavelet deep learning examples.
% It may change or be removed in a future release.

% find pwd
p = pwd;
if ispc
    % replace blank spaces with underscores
    p = strrep(p,' ','_');

    % split path into component folders
    pSplit = regexp(p,filesep,'split');

    % Since Windows uses colons, remove any colons that occur
    for k=1:numel(pSplit)
        pSplit{k} = erase(pSplit{k},':');
    end

    % now build the path using Linux file separation
    pLinux = '';
    for k=1:numel(pSplit)-1
        pLinux = [pLinux,pSplit{k},'/'];
    end
    pLinux = [pLinux,pSplit{end}];
else
    pLinux = p;
end

targetDir = '';
for k=1:numel(dirPaths)
    d = strfind(dirPaths{k}.directory,pLinux);
    if ~isempty(d)
        targetDir = dirPaths{k}.directory;
        break
    end
end

if numel(targetDir) == 0
    disp('Target directory not found.');
```

```
end
end
```

See Also

Crack Identification From Accelerometer Data

This example shows how to use wavelet and deep learning techniques to detect transverse pavement cracks and localize their position. The example demonstrates the use of wavelet scattering sequences as inputs to a gated recurrent unit (GRU) and 1-D convolutional network to classify time series based on the presence or absence of a crack. The data are vertical acceleration measurements obtained from a sensor mounted on the suspension knuckle of the front passenger seat wheel. Early identification of developing transverse cracks is important for pavement performance evaluation and maintenance. Reliable automatic detection methods enable more frequent and extensive monitoring.

Please read the Data - Description and Required Attributions on page 13-0 before running this example. All data import and preprocessing is described in the Data — Import on page 13-0 and the Data — Preprocessing on page 13-0 sections. If you want to skip the training-test set creation, preprocessing, feature extraction, and model training, you can go directly to the Classification and Analysis on page 13-0 section. There you can load the preprocessed test data as well as the extracted features and trained models.

Data — Description and Required Attributions

The data used in this example was retrieved from the Mendeley Data open data repository [2 on page 13-0]. The data is distributed under a Creative Commons (CC) BY 4.0 license. Download the data and models used in this example in your temporary directory specified by MATLAB® `tempdir` command. If you choose to download the data in a folder different from `tempdir`, change the directory name in the subsequent instructions. Unzip the data into a folder specified as `TransverseCrackData`.

```
dataURL = 'https://ssd.mathworks.com/supportfiles/wavelet/crackDetection/transverse_crack.zip';
saveFolder = fullfile(tempdir,'TransverseCrackData');
zipFile = fullfile(tempdir,'transverse_crack.zip');
websave(zipFile,dataURL);
unzip(zipFile,saveFolder)
```

After you unzip the data, the `TransverseCrackData` folder contains a subfolder called `transverse_crack_latest`. All subsequent commands must be run in this folder, or you can place this folder on the `matlabpath`.

The text file, *vehiclevibrationdata.rights*, included in the zip file contains the text of the CC BY 4.0 license. The data has been repackaged from the original Excel format into MAT-files.

Data acquisition is described in detail in [1 on page 13-0]. Twelve four meter long sections of asphalt containing a centrally-located transverse crack and twelve equal-length uncracked sections were used. The data is obtained from three separate roads. The transverse cracks ranged in width from 2-13 mm with crack spacings from 7-35 mm. The sections were driven at three different speeds: 30 km/hr, 40 km/hr, and 50 km/hr. Vertical acceleration measurements from the front passenger suspension knuckle are acquired at a sampling frequency of 1.28 kHz. The speeds of 30, 40, and 50 km/hr correspond to sensor measurements every 6.5 mm at 30 km/hr, 8.68 mm at 40 km/hr, and 10.85 mm at 50 km/hour. See [1] on page 13-0 for a detailed wavelet analysis of these data distinct from the analyses in this example.

In keeping with the stipulations of the CC BY 4.0 license, we note that the speed information of the original data is not retained in the data used in this example. The speed and road information are retained in the `road1.mat`, `road2.mat`, `road3.mat` data files included in the data folder for completeness.

Data — Download and Import

Load the accelerometer data and their corresponding labels. There are 327 accelerometer recordings.

```
load(fullfile(saveFolder,"transverse_crack_latest","allroadData.mat"));
load(fullfile(saveFolder,"transverse_crack_latest","allroadLabel.mat"));
```

Data — Preprocessing

Obtain the length of all the time series. Display a bar graph of the number of time series per length.

```
tslen = cellfun(@length,allroadData);
uLen = unique(tslen);
Ng = histcounts(tslen);
Ng = Ng(Ng > 0);
bar(uLen,Ng,0.5)
grid on
AX = gca;
AX.YLim = [0 max(Ng)+15];
text(uLen(1),Ng(1)+10,num2str(Ng(1)))
text(uLen(2),Ng(2)+10,num2str(Ng(2)))
text(uLen(3),Ng(3)+10,num2str(Ng(3)))
xlabel('Length in Samples')
ylabel('Number of Series')
title('Time Series Length')
```
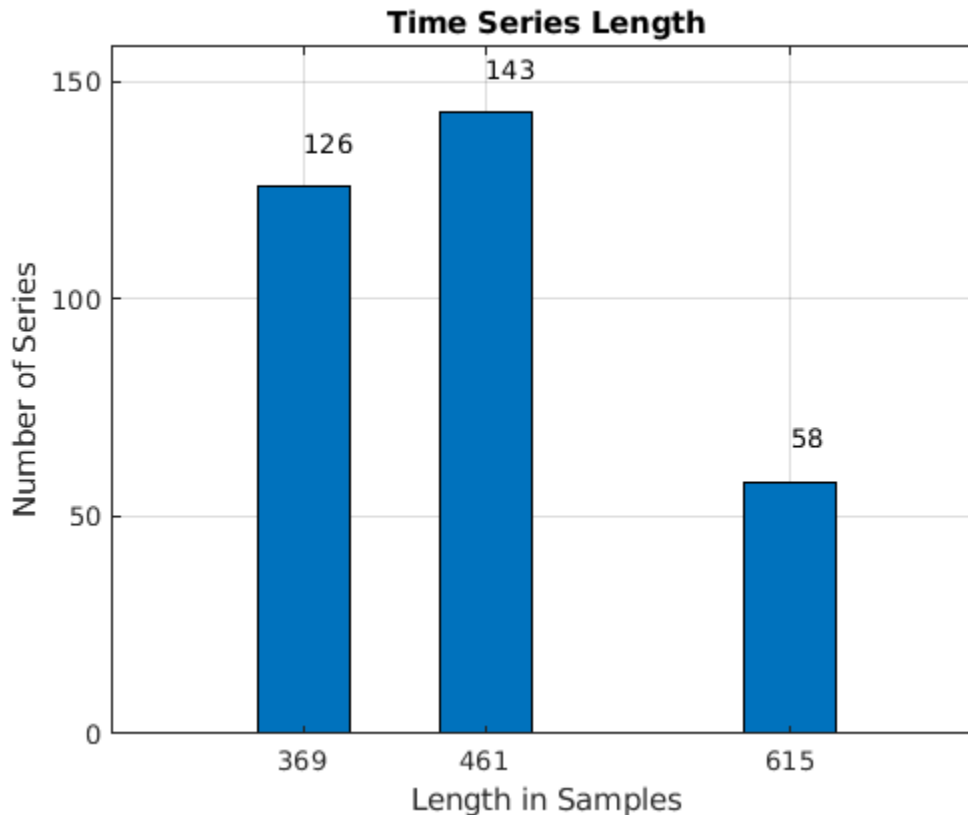
There are three unique lengths in the dataset: 369, 461, and 615 samples. The vehicle is traveling at three different speeds but the distance traversed and sample rate is constant resulting in different data record lengths. Determine how many records are in the "Cracked" (CR) and "Uncracked" (UNCR) classes.

```
tabulate(allroadLabel)
```

```
  Value    Count    Percent
     CR      109     33.33%
   UNCR      218     66.67%
```

This dataset is significantly imbalanced. There are twice as many time series without a crack (UNCR) as series containing a crack (CR). This means that a classifier which predicts "Uncracked" on each record would achieve an accuracy of 67% without any learning.

The time series are also of different lengths. To use a convolution network, a common input shape is needed. In recurrent networks it is possible to use unequal length time series as inputs, but all time series in a mini-batch are padded or truncated based on the training options. This requires care in creating mini-batches for both training and testing to ensure the proper distribution of padded sequences. Further, it requires that you do not shuffle the data during training. With this small dataset, shuffling the training data for each epoch is desirable. Accordingly, a common time series length is used.

The most common length is 461 samples. Further, the crack, if present, is centrally located in the recording. Accordingly, we can symmetrically extend the series with 369 samples to length 461 by reflecting the initial and final 46 samples. In the recordings with 615 samples, remove the initial 77 and final 77 samples.

Training — Feature Extraction and Network Training

The following sections generate the training and test sets, create the wavelet scattering sequences, and train both gated recurrent unit (GRU) and 1-D convolutional networks. First, split the data and corresponding labels into "Cracked" vs "Uncracked" sets.

```
[~,~,pop] = unique(allroadLabel);
CRidx = find(pop == 1);
UCRidx = find(pop == 2);
CRdata = allroadData(CRidx);
CRlabel = allroadLabel(CRidx);
UCRdata = allroadData(UCRidx);
UCRlabel = allroadLabel(UCRidx);
```

Extend or truncate the series in both sets to obtain a common length of 461 samples.

```
CRdataUL = equalLenTS(CRdata);
UCRdataUL = equalLenTS(UCRdata);
```

Now each time series in both the cracked and uncracked datasets has 461 samples. Split the data in a training set consisting of 80% of the time series in each class and hold out the remaining 20% of each class for testing.

```
NtrainCR = floor(0.8*length(CRlabel));
NtrainUCR = floor(0.8*length(UCRlabel));
idxTrainCR = randperm(length(CRlabel),NtrainCR);
idxTrainUCR = randperm(length(UCRlabel),NtrainUCR);
idxTestCR = setdiff(1:length(CRlabel),idxTrainCR);
idxTestUCR = setdiff(1:length(UCRlabel),idxTrainUCR);
```

Create the training and test sets and print a summary of each class.

```
TrainData = [CRdataUL(idxTrainCR) ; UCRdataUL(idxTrainUCR)];
TrainLabels = [CRlabel(idxTrainCR) ;  UCRlabel(idxTrainUCR)];
TestData = [CRdataUL(idxTestCR) ; UCRdataUL(idxTestUCR)];
TestLabels = [CRlabel(idxTestCR) ; UCRlabel(idxTestUCR)];
tabulate(TrainLabels)
```

```
  Value    Count   Percent
     CR       87    33.33%
   UNCR      174    66.67%
```

```
tabulate(TestLabels)
```

```
  Value    Count   Percent
     CR       22    33.33%
   UNCR       44    66.67%
```

Shuffle the data and labels once before training.

```
idxS = randperm(length(TrainData));
TrainData = TrainData(idxS);
TrainLabels = TrainLabels(idxS);
idxS = randperm(length(TrainData));
TrainData = TrainData(idxS);
TrainLabels = TrainLabels(idxS);
```

Compute the scattering sequences for each of the training series. The scattering sequences are stored in a cell array to be compatible with the GRU network. These sequences are subsequently reshaped into a 4-D array input for use with a 1-D convolutional network.

```
XTrainSCAT = cell(size(TrainData));
for kk = 1:numel(TrainData)
    XTrainSCAT{kk} = helperscat(TrainData{kk});
end
npaths = cellfun(@(x)size(x,1),XTrainSCAT);
inputSize = npaths(1);
```

Training — GRU Network

Construct the GRU network layers. Use two GRU layers with 30 hidden units each as well as two dropout layers. The input size is the number of scattering paths. To train this network on the raw time series, change the `inputSize` to 1 and transpose each time series to a row vector (1-by-461). If you wish to skip the network training, you may go directly to the Classification and Analysis on page 13-0   section. There you can load the trained GRU network as well as the preprocessed training and test sets.

```
numHiddenUnits1 = 30;
numHiddenUnits2 = 30;
numClasses = 2;

GRUlayers = [ ...
    sequenceInputLayer(inputSize,'Name','InputLayer',...
        'Normalization','zerocenter')
    gruLayer(numHiddenUnits1,'Name','GRU1','OutputMode','sequence')
    dropoutLayer(0.35,'Name','Dropout1')
    gruLayer(numHiddenUnits2,'Name','GRU2','OutputMode','last')
    dropoutLayer(0.2,'Name','Dropout2')
```

```
        fullyConnectedLayer(numClasses,'Name','FullyConnected')
        softmaxLayer('Name','smax');
        classificationLayer('Name','ClassificationLayer')];
```

Train the GRU network. Use a mini-batch size of 15 with 150 epochs.

```
maxEpochs = 150;
miniBatchSize = 15;

options = trainingOptions('adam', ...
    'ExecutionEnvironment','gpu', ...
    'L2Regularization',1e-3,...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'Shuffle','every-epoch',...
    'Verbose',0,...
    'Plots','none');
iterPerEpoch = floor(length(XTrainSCAT)/miniBatchSize);
[scatGRUnet,infoGRU] = trainNetwork(XTrainSCAT,TrainLabels,GRUlayers,options);
```

Plot the smoothed training accuracy and loss per iteration.

```
figure
subplot(2,1,1)
smoothedACC = filter(1/iterPerEpoch*ones(iterPerEpoch,1),1,...
    infoGRU.TrainingAccuracy);
smoothedLoss = filter(1/iterPerEpoch*ones(iterPerEpoch,1),1,...
    infoGRU.TrainingLoss);
plot(smoothedACC)
title(['Training Accuracy (Smoothed) '...
    num2str(iterPerEpoch) ' iterations per epoch'])
ylabel('Accuracy (%)')
ylim([0 100.1])
grid on
xlim([1 length(smoothedACC)])
subplot(2,1,2)
plot(smoothedLoss)
ylim([-0.01 1])
grid on
xlim([1 length(smoothedLoss)])
ylabel('Loss')
xlabel('Iteration')
```
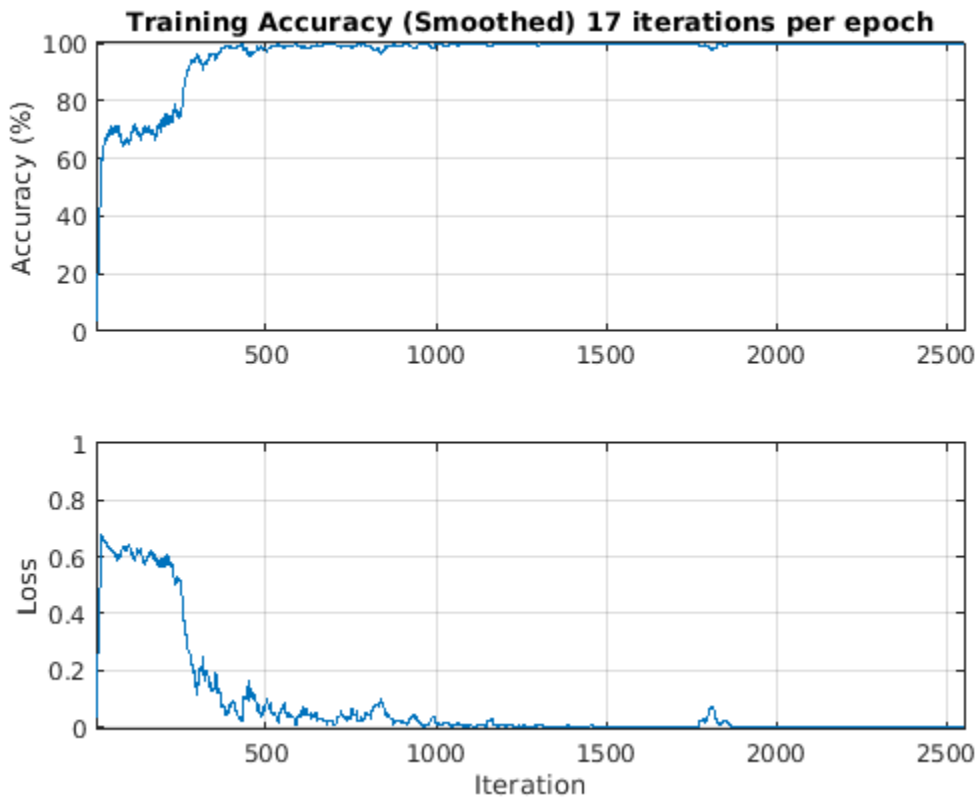
Obtain the wavelet scattering transforms of the held-out test data for classification.

```
XTestSCAT = cell(size(TestData));
for kk = 1:numel(TestData)
    XTestSCAT{kk} = helperscat(TestData{kk});
end
```

Training — 1-D Convolutional Network

Train a 1-D convolutional network with wavelet scattering sequences. The scattering sequences are 38-by-58 where 58 is the number of time steps and 38 is the number of scattering paths. To use this in a 1-D convolutional network, transpose and reshape the scattering sequences to be 58-by-1-by-38-by-Nsig where Nsig is the number of training examples. If you wish to skip the network training, you may go directly to the Classification and Analysis on page 13-0   section. There you can load the trained convolutional network as well as the preprocessed training and test sets.

```
scatCONVTrain = zeros(58,1,38,length(XTrainSCAT),'single');
for kk = 1:length(XTrainSCAT)
    scatCONVTrain(:,:,:,kk) = reshape(XTrainSCAT{kk}',[58,1,38]);
end
```

Construct and train the 1-D convolutional network.

```
conv1dLayers = [
    imageInputLayer([58 1 38]);
    convolution2dLayer([3 1],64,'stride',1);
    batchNormalizationLayer;
```

```
    reluLayer;
    maxPooling2dLayer([2 1]);
    convolution2dLayer([3 1],32);
    reluLayer;
    maxPooling2dLayer([2 1]);
    fullyConnectedLayer(500);
    fullyConnectedLayer(2);
    softmaxLayer;
    classificationLayer;
    ];

convoptions = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropFactor',0.5, ...
    'LearnRateDropPeriod',5, ...
    'Plots','training-progress',...
    'MaxEpochs',50,...
    'Verbose',0,...
    'Plots','none',...
    'MiniBatchSize',15);
[scatCONV1dnet,infoCONV] = ...
    trainNetwork(scatCONVTrain,TrainLabels,conv1dLayers,convoptions);
```

Plot the smoothed training accuracy and loss per iteration.

```
iterPerEpoch = floor(size(scatCONVTrain,4)/15);
figure
subplot(2,1,1)
smoothedACC = filter(1/iterPerEpoch*ones(iterPerEpoch,1),1,...
    infoCONV.TrainingAccuracy);
smoothedLoss = filter(1/iterPerEpoch*ones(iterPerEpoch,1),1,...
    infoCONV.TrainingLoss);
plot(smoothedACC)
title(['Training Accuracy (Smoothed) '...
    num2str(iterPerEpoch) ' iterations per epoch'])
ylabel('Accuracy (%)')
ylim([0 100.1])
grid on
xlim([1 length(smoothedACC)])
subplot(2,1,2)
plot(smoothedLoss)
ylim([-0.01 1])
grid on
xlim([1 length(smoothedLoss)])
ylabel('Loss')
xlabel('Iteration')
```
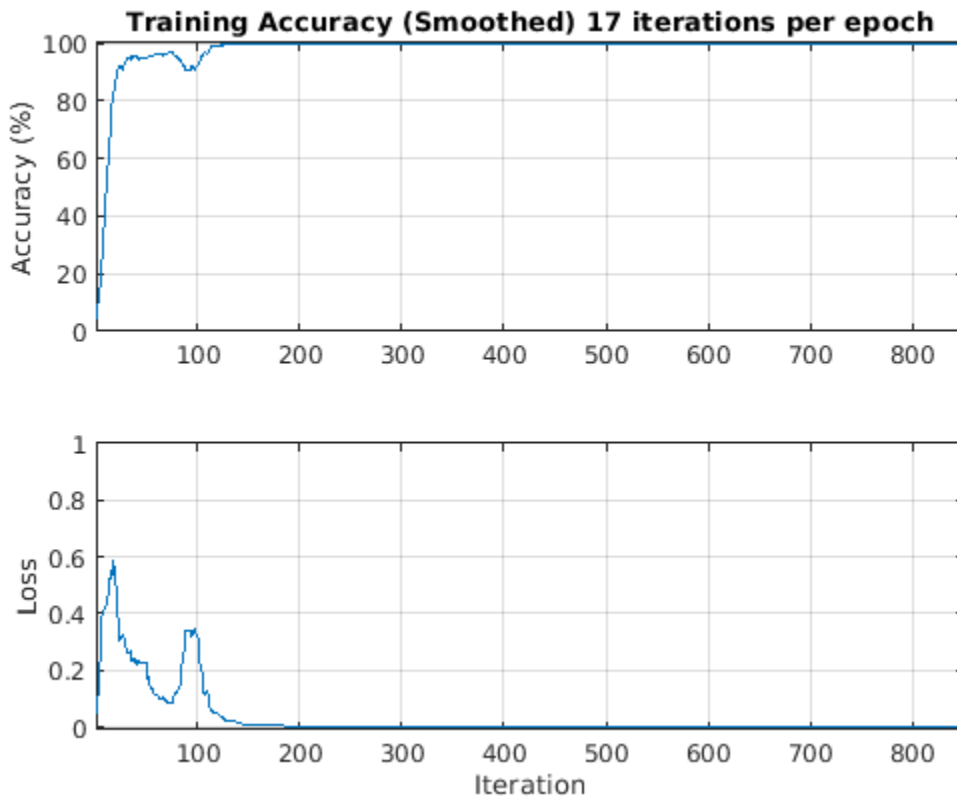
Reshape the test set of scattering sequences to be compatible with the network for classification.

```
scatCONVTest = zeros(58,1,38,length(XTestSCAT));
for kk = 1:length(XTestSCAT)
    scatCONVTest(:,:,:,kk) = reshape(XTestSCAT{kk}',58,1,38);
end
```

Classification and Analysis

Load the trained gated recurrent unit (GRU) and 1-D convolutional networks along with the test data and scattering sequences. All data, features, and networks were created in the Training — Feature Extraction and Network Training on page 13-0   section.

```
load(fullfile(saveFolder,"transverse_crack_latest","TestData.mat"))
load(fullfile(saveFolder,"transverse_crack_latest","TestLabels.mat"))
load(fullfile(saveFolder,"transverse_crack_latest","XTestSCAT.mat"))
load(fullfile(saveFolder,"transverse_crack_latest","scatGRUnet"))
load(fullfile(saveFolder,"transverse_crack_latest","scatCONV1Dnet.mat"))
load(fullfile(saveFolder,"transverse_crack_latest","scatCONVTest.mat"))
```

If you additionally want the preprocessed data, labels, and wavelet scattering sequences for the training data, you can load those with the following commands. These data and labels are not used in the remainder of this example if you wish to skip the following load commands.

```
load(fullfile(saveFolder,"transverse_crack_latest","TrainData.mat"))
load(fullfile(saveFolder,"transverse_crack_latest","TrainLabels.mat"))
load(fullfile(saveFolder,"transverse_crack_latest","XTrainSCAT.mat"))
load(fullfile(saveFolder,"transverse_crack_latest","scatCONVTrain.mat"))
```

Examine the number of time series per class in the test set. Note the test set is significantly imbalanced as discussed in Data — Preprocessing on page 13-0 section.

```
tabulate(TestLabels)
```

```
  Value    Count    Percent
    CR        22      33.33%
   UNCR       44      66.67%
```

XTestSCAT contains the wavelet scattering sequences computed for the raw time series in TestData.

Show the GRU model performance on the test data not used in model training.

```
miniBatchSize = 15;
ypredSCAT = classify(scatGRUnet,XTestSCAT, ...
    'MiniBatchSize',miniBatchSize);
figure
confusionchart(TestLabels,ypredSCAT,'RowSummary','row-normalized',...
    'ColumnSummary','column-normalized');
title({'GRU network -- Wavelet Scattering Sequences'; ...
    'Confusion chart with precision and recall'});
```
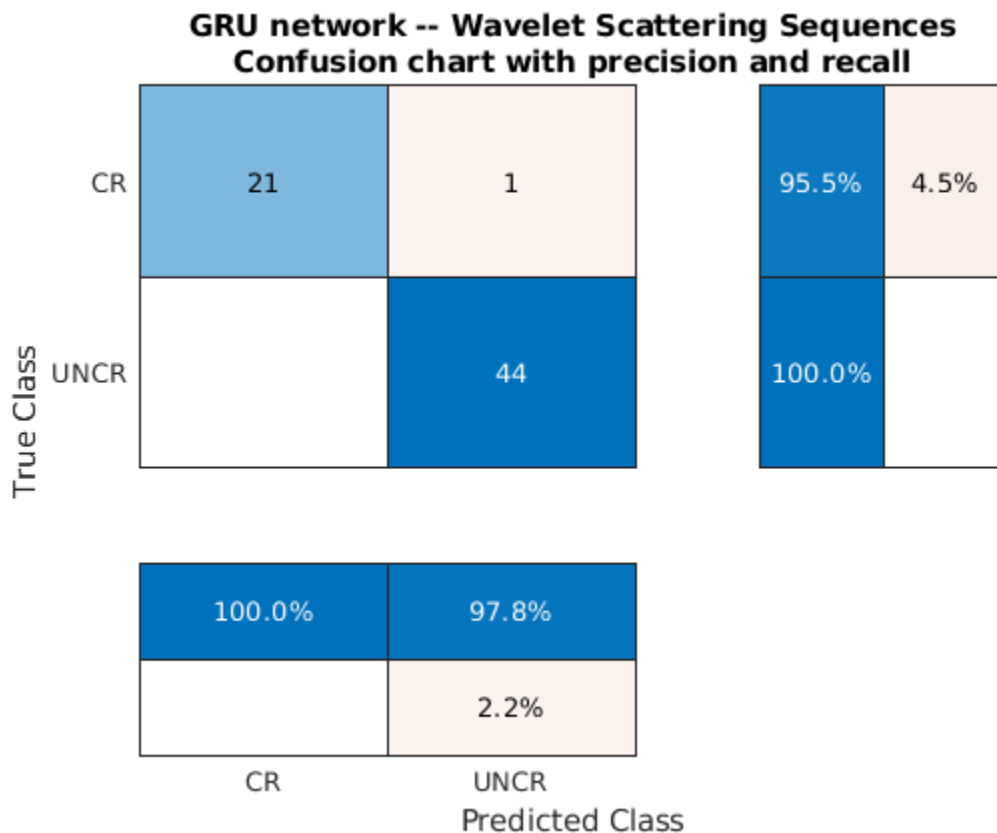

In spite of the large imbalance in the classes and the small dataset, the precision and recall values indicate the network performs well on the test data. Specifically, the precision value for "Cracked" data is approximately 100% and the recall is approximately 95%. These values are especially good given that a full 67% of the records in the training set were "Uncracked". The network has not overlearned to classify the time series as "Uncracked" in spite of the imbalance.

If you set the inputSize = 1 and transpose the time series in the training data, you can retrain the GRU network on the raw time series data. This was done on the same data in the training set. You can load that network and check the performance on the test set.

```
load(fullfile(saveFolder,"transverse_crack_latest","tsGRUnet.mat"));
rawTest = cellfun(@transpose,TestData,'UniformOutput',false);
miniBatchSize = 15;
YPredraw = classify(tsGRUnet,rawTest, ...
    'MiniBatchSize',miniBatchSize);
confusionchart(TestLabels,YPredraw,'RowSummary','row-normalized',...
    'ColumnSummary','column-normalized');
title({'GRU network -- Raw Time Series'; ...
    'Confusion chart with precision and recall'});
```


GRU network -- Raw Time Series
Confusion chart with precision and recall
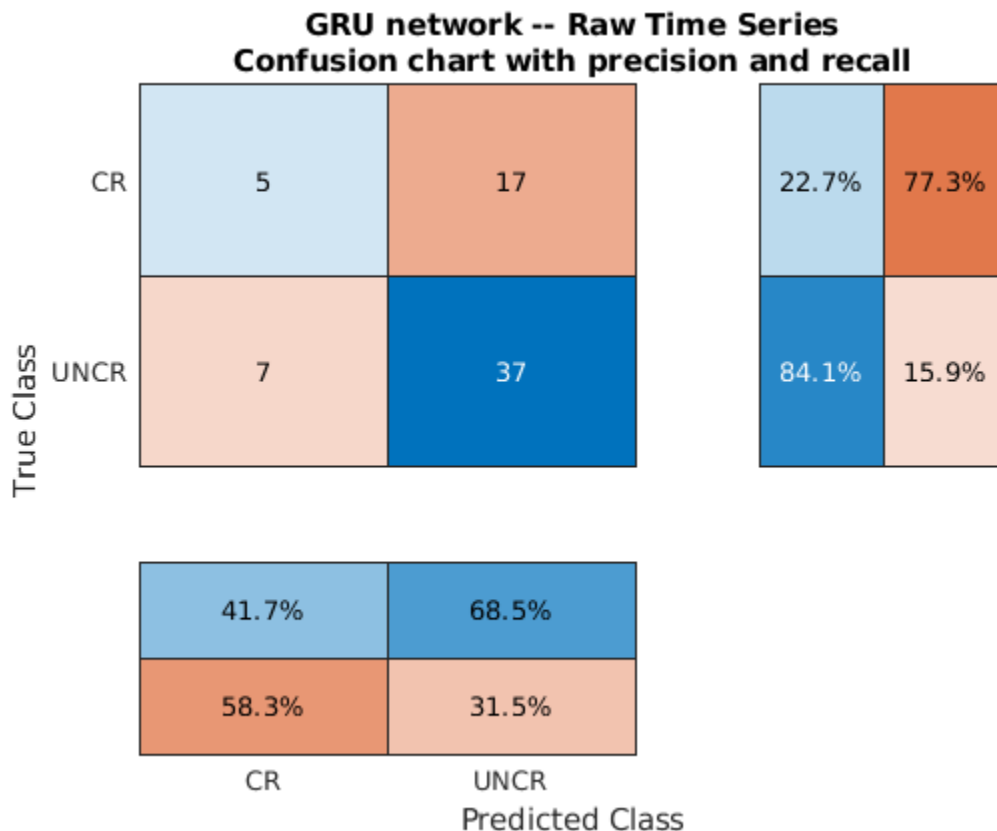
For this network, the performance is not good. Specifically, the recall for the "Cracked" data is poor. The number of false negatives for the "Crack" data is quite large. This is exactly what you would expect with an imbalanced dataset when the model has not learned well.

Test the 1-D convolutional network trained with the wavelet scattering sequences.

```
miniBatchSize = 15;
YPredSCAT = classify(scatCONV1Dnet,scatCONVTest,...
    'MiniBatchSize',miniBatchSize);
figure
confusionchart(TestLabels,YPredSCAT,'RowSummary','row-normalized',...
    'ColumnSummary','column-normalized');
title({'1-D Convolutional Network-- Wavelet Scattering Sequences'; ...
    'Confusion chart with precision and recall'});
```

1-D Convolutional Network-- Wavelet Scattering Sequences
Confusion chart with precision and recall
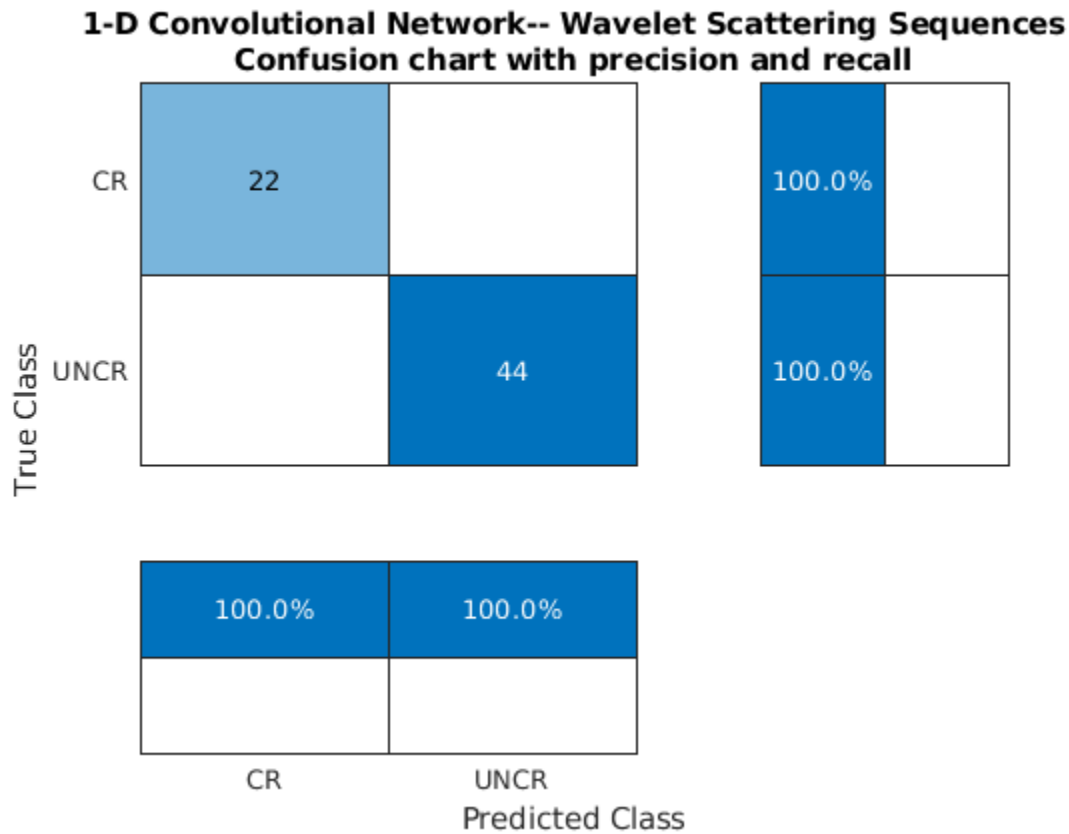
The performance of the convolutional network with scattering sequences is excellent and exceeds the performance of the GRU network. Precision and recall on the minority class demonstrate robust learning.

To train the 1-D convolutional network on the raw sequences set `inputSize = [461 1]` in the `imageInputLayer`. You can load and test that network.

```
load(fullfile(saveFolder,"transverse_crack_latest","tsCONV1Dnet.mat"));
XTestData = cell2mat(TestData');
XTestData = reshape(XTestData,[461 1 1 66]);
miniBatchSize = 15;
YPredRAW = classify(tsCONV1Dnet,XTestData,...
    'MiniBatchSize',miniBatchSize);
confusionchart(TestLabels,YPredRAW,'RowSummary','row-normalized',...
    'ColumnSummary','column-normalized');
title({'1-D Convolutional Network-- Raw Sequences'; ...
    'Confusion chart with precision and recall'});
```

1-D Convolutional Network-- Raw Sequences
Confusion chart with precision and recall
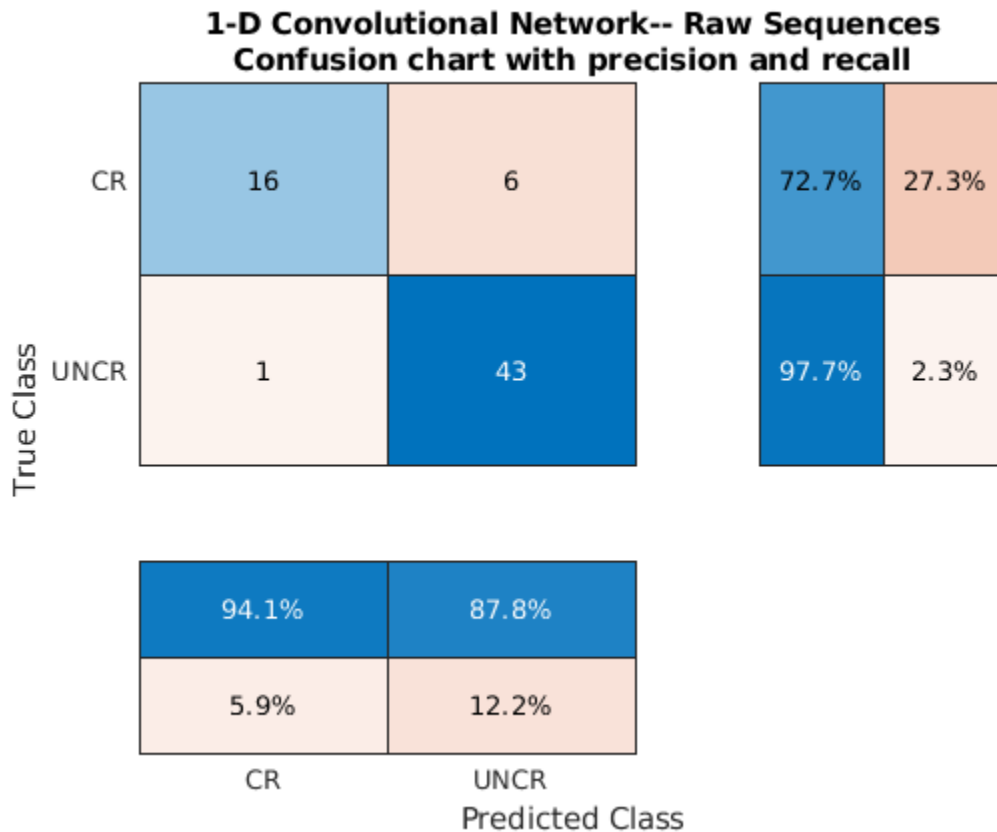
The 1-D convolution network with the raw sequences performs better than the GRU network, but not as well as the convolutional network trained with the wavelet scattering sequences. Further, the reduction along the time dimension of the data with the scattering transform has resulted in a network which is approximately 8.5 times smaller than the convolutional network trained on the raw data.

Wavelet Inference and Analysis

This section demonstrates how to classify a single time series using wavelet analysis with a pretrained model. The model used is the 1-D convolutional network trained on wavelet scattering sequences. Load the trained network and some test data if you have not already loaded these in the previous section.

```
load(fullfile(saveFolder,"transverse_crack_latest","scatCONV1Dnet.mat"));
load(fullfile(saveFolder,"transverse_crack_latest","TestData.mat"));
```

Construct the wavelet scattering network to transform the data. Select a time series from the test data and classify the data. If the model classifies the time series as "Cracked", investigate the series for the position of the crack in the waveform.

```
sf = waveletScattering('SignalLength',461,...
    'OversamplingFactor',1,'qualityfactors',[8 1],...
    'InvarianceScale',0.05,'Boundary','reflect','SamplingFrequency',1280);
idx = 10;
data = TestData{idx};
[smat,x] = featureVectors(data,sf);
PredictedClass = classify(scatCONV1Dnet,smat);
```

```matlab
if isequal(PredictedClass,'CR')
    fprintf('Crack detected. Computing wavelet transform modulus maxima.\n')
    wtmm(data,'Scaling','local');
end
```

Crack detected. Computing wavelet transform modulus maxima.

	Sample	Holder Expone
1	5	-2.155
2	9	-1.828
3	11	-0.398
4	15	-0.693
5	21	-0.302
6	23	-1.165
7	26	-2.234
8	28	-2.111
9	31	-1.882
10	33	-1.781
11	39	-0.762
12	41	-0.595
13	45	-2.265
14	48	-1.327
15	54	-2.104
16	56	-3.382
17	59	-2.367
18	61	-0.270
19	64	-2.207
20	69	-1.525
21	71	-1.280
22	76	-2.822
23	78	-1.699
24	84	-1.134
25	87	-0.178
26	92	-2.173
27	94	-1.624
28	98	-3.131
29	100	-1.484
30	102	-2.678

The wavelet transform modulus maxima (WTMM) technique shows a maxima line converging to the finest scale at sample 205. Maxima lines that converge to fine scales are a good estimate of where singularities are in a time series. This makes sample 205 a good estimate of the location of the crack.

```matlab
plot(x,data)
axis tight
hold on
plot([x(205) x(205)],[min(data) max(data)],'k')
grid on
title(['Crack located at ' num2str(x(205)) 'meters'])
xlabel('Distance (m)')
ylabel('Amplitude')
```

Crack located at 1.7708 meters
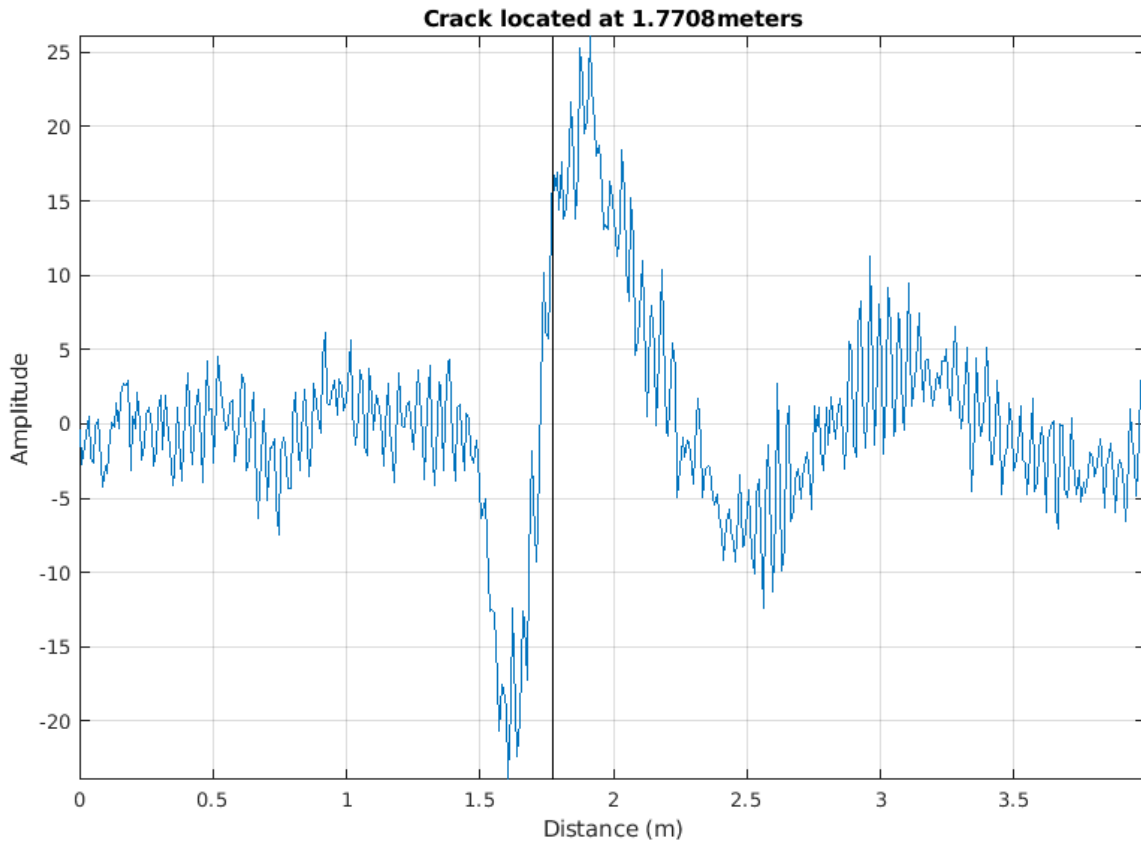
You can increase your confidence in this location by using multiresolution analysis (MRA) techniques and identifying changes in slope in long-scale wavelet MRA series. See "Practical Introduction to Multiresolution Analysis" on page 11-2 for an introduction to MRA techniques. In [1] on page 13-0 the difference in energy between "Cracked" and "Uncracked" series occurred in the low frequency bands, specifically in the interval of [10,20] Hz. Accordingly, the following MRA is focused on signal components in the frequency bands from [10,80] Hz. In these bands, identify linear changes in the data. Plot the change points along with the MRA components.

```
[mra,chngpts] = helperMRA(data,x);
```

MRA Components
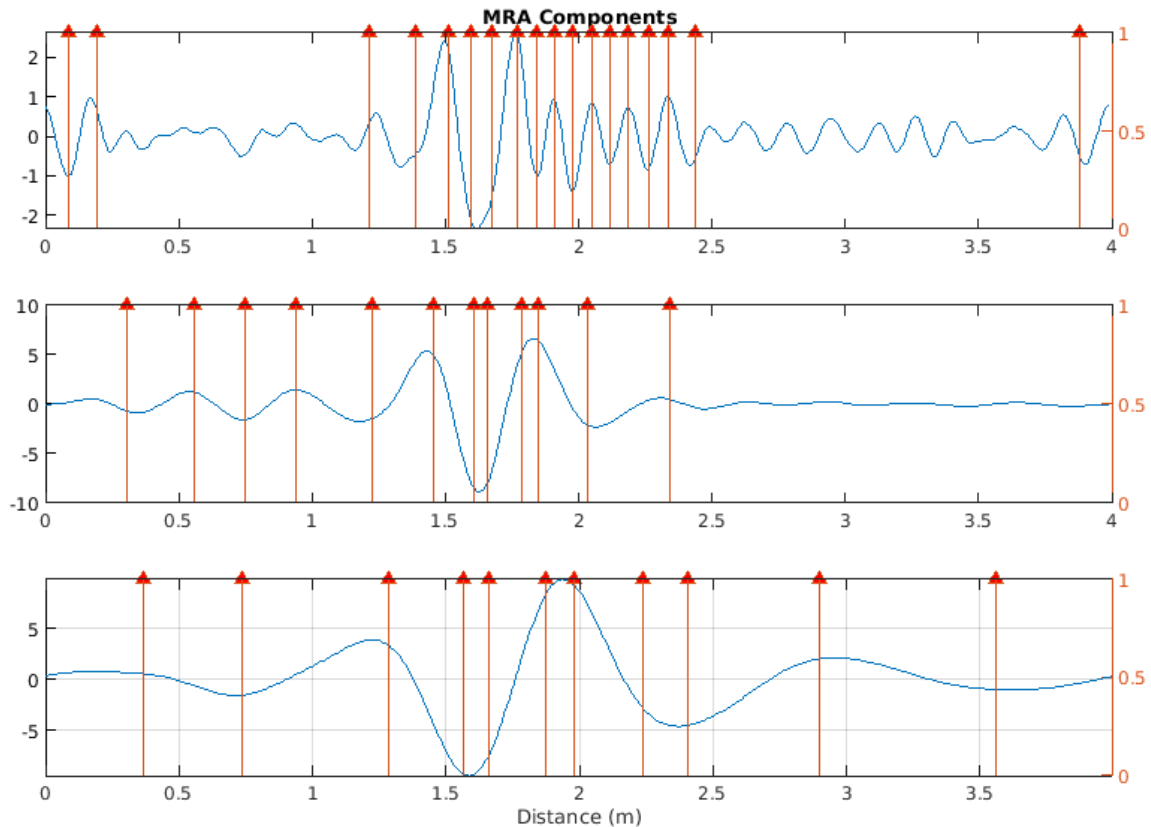
The MRA-based changepoint analysis has helped to confirm the WTMM analysis in identifying the region around 1.8 meters as the probable location of the crack.

Summary

This example showed how to use wavelet scattering sequences with both recurrent and convolutional networks to classify time series. The example further demonstrated how wavelet techniques can help to localize features on the same spatial (time) scale as the original data.

References

[1] Yang, Qun and Shishi Zhou. "Identification of asphalt pavement transverse cracking based on vehicle vibration signal analysis.", Road Materials and Pavement Design, 2020, 1-19. https://doi.org/10.1080/14680629.2020.1714699.

[2] Zhou,Shishi. "Vehicle vibration data." https://data.mendeley.com/datasets/3dvpjy4m22/1. Data is used under CC BY 4.0. Data is repackaged from original Excel data format to MAT-files. Speed label removed and only "crack" or "nocrack" label retained.

Appendix

Helper functions used in this example.

```
function smat = helperscat(datain)
% This helper function is only in support of Wavelet Toolbox examples.
```

```matlab
% It may change or be removed in a future release.
datain = single(datain);

sf = waveletScattering('SignalLength',length(datain),...
    'OversamplingFactor',1,'qualityfactors',[8 1],...
    'InvarianceScale',0.05,'Boundary','reflect','SamplingFrequency',1280);
smat = sf.featureMatrix(datain);

end

%-------------------------------------------------------------------------
function dataUL = equalLenTS(data)
% This function in only in support of Wavelet Toolbox examples.
% It may change or be removed in a future release.
N = length(data);
dataUL = cell(N,1);
for kk = 1:N
    L = length(data{kk});
    switch L
        case 461
            dataUL{kk} = data{kk};
        case 369
            Ndiff = 461-369;
            pad = Ndiff/2;
            dataUL{kk} = [flip(data{kk}(1:pad)); data{kk} ; ...
                flip(data{kk}(L-pad+1:L))];
        otherwise
            Ndiff = L-461;
            zrs = Ndiff/2;
            dataUL{kk} = data{kk}(zrs:end-zrs-1);
    end
end

end

%-------------------------------------------------------------------------
function [fmat,x] = featureVectors(data,sf)
% This function is only in support of Wavelet Toolbox examples.
% It may change or be removed in a future release.
data = single(data);
N = length(data);
dt = 1/1280;
if N < 461
    Ndiff = 461-N;
    pad = Ndiff/2;
    dataUL = [flip(data(1:pad)); data ; ...
            flip(data(N-pad+1:N))];
     rate = 5e4/3600;
     dx = rate*dt;
     x = 0:dx:(N*dx)-dx;

elseif N > 461
    Ndiff = N-461;
    zrs = Ndiff/2;
    dataUL = data(zrs:end-zrs-1);
    rate = 3e4/3600;
    dx = rate*dt;
    x = 0:dx:(N*dx)-dx;
```
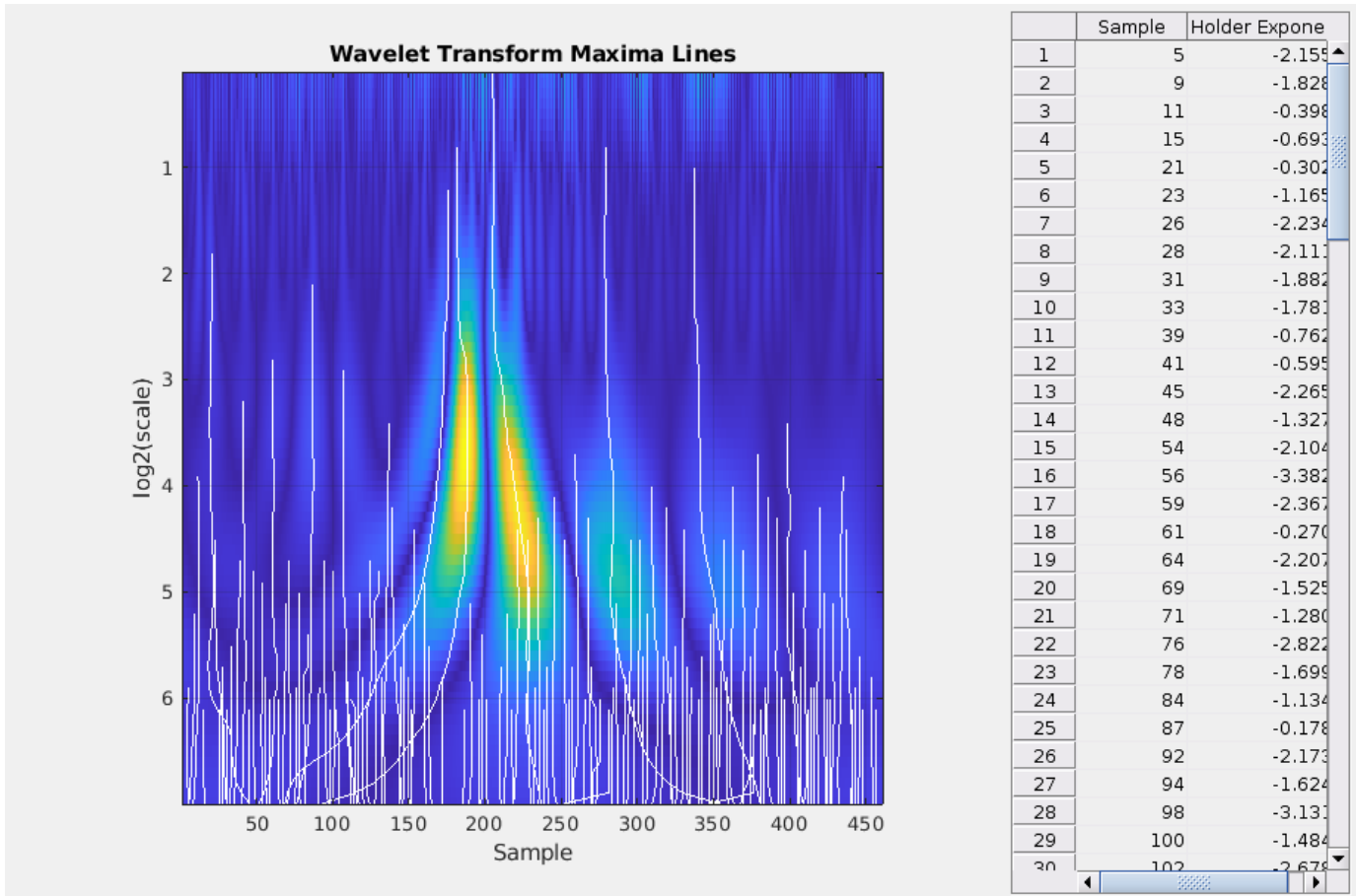
```matlab
else
    dataUL = data;
    rate = 4e4/3600;
    dx = rate*dt;
    x = 0:dx:(N*dx)-dx;
end
fmat = sf.featureMatrix(dataUL);
fmat = reshape(fmat',[58 1 38]);
end

%--------------------------------------------------------------------------------
function [mra,chngpts] = helperMRA(data,x)
% This function is only in support of Wavelet Toolbox examples.
% It may change or be removed in a future release.
mra = modwtmra(modwt(data,'sym3'),'sym3');
mraLev = mra(4:6,:);
Ns = size(mraLev,1);
thresh = [2, 4, 8];
chngpts = false(size(mraLev));
% Determine changepoints. We want different thresholds for different
% resolution levels.
for ii = 1:Ns
    chngpts(ii,:) = ischange(mraLev(ii,:),"linear",2,"Threshold",thresh(ii));
end

for kk = 1:Ns
    idx = double(chngpts(kk,:));
    idx(idx == 0) = NaN;
    subplot(Ns,1,kk)
    plot(x,mraLev(kk,:))
    if kk == 1
        title('MRA Components')
    end
    yyaxis right
    hs = stem(x,idx);
    hs.ShowBaseLine = 'off';
    hs.Marker = '^';
    hs.MarkerFaceColor = [1 0 0];
end
grid on
axis tight
xlabel('Distance (m)')
end
```

See Also

waveletScattering

More About

- "Wavelet Scattering" on page 9-2

Deploy Signal Classifier on NVIDIA Jetson Using Wavelet Analysis and Deep Learning

This example shows how to generate and deploy a CUDA® executable that classifies human electrocardiogram (ECG) signals using features extracted by the continuous wavelet transform (CWT) and a pretrained convolutional neural network (CNN).

SqueezeNet is a deep CNN originally designed to classify images in 1000 categories. We reuse the network architecture of the CNN to classify ECG signals based on their scalograms. A scalogram is the absolute value of the CWT of the signal. After training SqueezeNet to classify ECG signals, you create a CUDA executable that generates a scalogram of an ECG signal and then uses the CNN to classify the signal. The executable and CNN are both deployed to the NVIDIA hardware.

This example uses the same data as used in "Classify Time Series Using Wavelet Analysis and Deep Learning" on page 13-79. In that example, transfer learning with GoogLeNet and SqueezeNet are used to classify ECG waveforms into one of three categories. The description of the data and how to obtain it are repeated here for convenience.

ECG Data Description and Download

The ECG data is obtained from three groups of people: persons with cardiac arrhythmia (ARR), persons with congestive heart failure (CHF), and persons with normal sinus rhythms (NSR). In total there are 162 ECG recordings from three PhysioNet databases: MIT-BIH Arrhythmia Database [2][3], MIT-BIH Normal Sinus Rhythm Database [3], and The BIDMC Congestive Heart Failure Database [1] [3]. More specifically, 96 recordings from persons with arrhythmia, 30 recordings from persons with congestive heart failure, and 36 recordings from persons with normal sinus rhythms. The goal is to train a model to distinguish between ARR, CHF, and NSR.

You can obtain this data from the MathWorks GitHub repository. To download the data from the website, click `Code` and select `Download ZIP`. Save the file `physionet_ECG_data-master.zip` in a folder where you have write permission. The instructions for this example assume you have downloaded the file to your temporary directory, `tempdir`, in MATLAB. Modify the subsequent instructions for unzipping and loading the data if you choose to download the data in a folder different from `tempdir`.

After downloading the data from GitHub, unzip the file in your temporary directory.

```
unzip(fullfile(tempdir,'physionet_ECG_data-master.zip'),tempdir)
```

Unzipping creates the folder `physionet-ECG_data-master` in your temporary directory. This folder contains the text file `README.md` and `ECGData.zip`. The `ECGData.zip` file contains:

- `ECGData.mat`
- `Modified_physionet_data.txt`
- `License.txt`

`ECGData.mat` holds the data used in this example. The text file `Modified_physionet_data.txt` is required by PhysioNet's copying policy and provides the source attributions for the data as well as a description of the preprocessing steps applied to each ECG recording.

Unzip `ECGData.zip` in `physionet-ECG_data-master`. Load the data file into your MATLAB workspace.

```
unzip(fullfile(tempdir,'physionet_ECG_data-master','ECGData.zip'),...
    fullfile(tempdir,'physionet_ECG_data-master'))
load(fullfile(tempdir,'physionet_ECG_data-master','ECGData.mat'))
```

ECGData is a structure array with two fields: `Data` and `Labels`. The `Data` field is a 162-by-65536 matrix where each row is an ECG recording sampled at 128 hertz. `Labels` is a 162-by-1 cell array of diagnostic labels, one label for each row of `Data`. The three diagnostic categories are: `'ARR'`, `'CHF'`, and `'NSR'`.

Feature Extraction

After downloading the data, you must generate scalograms of the signals. The scalograms are the "input" images to the CNN.

To store the scalograms of each category, first create an ECG data directory `'data'` inside `tempdir`. Then create three subdirectories in `'data'` named after each ECG category. The helper function `helperCreateECGDirectories` does this for you. `helperCreateECGDirectories` accepts `ECGData`, the name of an ECG data directory, and the name of a parent directory as input arguments. You can replace `tempdir` with another directory where you have write permission. You can find the source code for this helper function in the Supporting Functions on page 13-0 section at the end of this example.

```
parentDir = tempdir;
dataDir = 'data';
helperCreateECGDirectories(ECGData,parentDir,dataDir)
```

After making the folders, create scalograms of the ECG signals as RGB images and write them to the appropriate subdirectory in `dataDir`. To create the scalograms, first precompute a CWT filter bank. Precomputing the filter bank is the preferred method when obtaining the CWT of many signals using the same parameters. The helper function `helperCreateRGBfromTF` does this.The source code for this helper function is in the Supporting Functions on page 13-0 section at the end of this example. To be compatible with the SqueezeNet architecture, each RGB image is an array of size 227-by-227-by-3.

```
helperCreateRGBfromTF(ECGData,parentDir,dataDir)
```

Divide Data Set into Training and Validation Data

Load the scalogram images as an image datastore. The `imageDatastore` function automatically labels the images based on folder names and stores the data as an `ImageDatastore` object. An image datastore enables you to store large image data, including data that does not fit in memory, and efficiently read batches of images when training a CNN.

```
allImages = imageDatastore(fullfile(tempdir,dataDir),...
    'IncludeSubfolders',true,...
    'LabelSource','foldernames');
```

Randomly divide the images into two groups, one for training and the other for validation. Use 80% of the images for training and the remainder for validation. For purposes of reproducibility, we set the random seed to the default value.

```
rng default
[imgsTrain,imgsValidation] = splitEachLabel(allImages,0.8,'randomized');
disp(['Number of training images: ',num2str(numel(imgsTrain.Files))]);
```

```
Number of training images: 130
```

```
disp(['Number of validation images: ',num2str(numel(imgsValidation.Files))]);
```

```
Number of validation images: 32
```

SqueezeNet

SqueezeNet is a pretrained CNN that can classify images into 1000 categories. You need to retrain SqueezeNet for our ECG classification problem. Prior to retraining, you modify several network layers and set various training options. After retraining is complete, you save the CNN in a `.mat` file. The CUDA executable use the `.mat` file.

Specify an experiment trial index and a results directory. If necessary, create the directory.

```
trial = 1;
ResultDir = 'results';
if ~exist(ResultDir,'dir')
    mkdir(ResultDir)
end
MatFile = fullfile(ResultDir,sprintf('SqueezeNet_Trial%d.mat',trial));
```

Load SqeezeNet. Extract the layer graph and inspect the last five layers.

```
sqz = squeezenet;
lgraph = layerGraph(sqz);
lgraph.Layers(end-4:end)
```

```
ans =
  5×1 Layer array with layers:

     1   'conv10'                           Convolution           1000 1×1×512 convolutions
     2   'relu_conv10'                      ReLU                  ReLU
     3   'pool10'                           Global Average Pooling   Global average pooling
     4   'prob'                             Softmax               softmax
     5   'ClassificationLayer_predictions'  Classification Output   crossentropyex with 'tench
```

To retrain SqueezeNet to classify the three classes of ECG signals, replace the `'conv10'` layer with a new convolutional layer with the number of filters equal to the number of ECG classes. Replace the classification layer with a new one without class labels.

```
numClasses = numel(categories(imgsTrain.Labels));
new_conv10_WeightLearnRateFactor = 1;
new_conv10_BiasLearnRateFactor = 1;
newConvLayer = convolution2dLayer(1,numClasses,...
        'Name','new_conv10',...
        'WeightLearnRateFactor',new_conv10_WeightLearnRateFactor,...
        'BiasLearnRateFactor',new_conv10_BiasLearnRateFactor);
lgraph = replaceLayer(lgraph,'conv10',newConvLayer);
newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,'ClassificationLayer_predictions',newClassLayer);
lgraph.Layers(end-4:end)
```

```
ans =
  5×1 Layer array with layers:

     1   'new_conv10'        Convolution           3 1×1 convolutions with stride [1  1] and
     2   'relu_conv10'       ReLU                  ReLU
     3   'pool10'            Global Average Pooling   Global average pooling
     4   'prob'              Softmax               softmax
     5   'new_classoutput'   Classification Output   crossentropyex
```

Create a set of training options to use with SqueezeNet.

```
OptimSolver = 'sgdm';
MiniBatchSize = 15;
MaxEpochs = 20;
InitialLearnRate = 1e-4;
Momentum = 0.9;
ExecutionEnvironment = 'cpu';

options = trainingOptions(OptimSolver,...
    'MiniBatchSize',MiniBatchSize,...
    'MaxEpochs',MaxEpochs,...
    'InitialLearnRate',InitialLearnRate,...
    'ValidationData',imgsValidation,...
    'ValidationFrequency',10,...
    'ExecutionEnvironment',ExecutionEnvironment,...
    'Momentum',Momentum);
```

Save all the parameters in a structure. The trained network and structure will be later saved in a .mat file.

```
TrialParameter.new_conv10_WeightLearnRateFactor = new_conv10_WeightLearnRateFactor;
TrialParameter.new_conv10_BiasLearnRateFactor = new_conv10_BiasLearnRateFactor;
TrialParameter.OptimSolver = OptimSolver;
TrialParameter.MiniBatchSize = MiniBatchSize;
TrialParameter.MaxEpochs = MaxEpochs;
TrialParameter.InitialLearnRate = InitialLearnRate;
TrialParameter.Momentum = Momentum;
TrialParameter.ExecutionEnvironment = ExecutionEnvironment;
```

Set the random seed to the default value and train the network. Save the trained network, trial parameters, training run time, and image datastore containing the validation images. The training process usually takes 1-5 minutes on a desktop CPU. If you want to use a trained CNN from a previous trial, set trial to the index number of that trial and LoadModel to true.

```
LoadModel = false;
if ~LoadModel
    rng default
    tic;
    trainedModel = trainNetwork(imgsTrain,lgraph,options);
    trainingTime = toc;
    fprintf('Total training time: %.2e sec\n',trainingTime);
    save(MatFile,'TrialParameter','trainedModel','trainingTime','imgsValidation');
else
    disp('Load ML model from the file')
    load(MatFile,'trainedModel','imgsValidation');
end
```

```
Initializing input data normalization.
|==========================================================================================...
| Epoch | Iteration | Time Elapsed | Mini-batch | Validation | Mini-batch | Validat:
|       |           | (hh:mm:ss)   | Accuracy   | Accuracy   | Loss       | Loss
|==========================================================================================...
|     1 |         1 |     00:00:02 |     26.67% |     25.00% |     4.1769 |     2.9
|     2 |        10 |     00:00:12 |     73.33% |     59.38% |     0.9877 |     1.1
|     3 |        20 |     00:00:21 |     60.00% |     56.25% |     0.9164 |     0.9
|     4 |        30 |     00:00:31 |     86.67% |     68.75% |     0.6698 |     0.7
|     5 |        40 |     00:00:41 |     66.67% |     68.75% |     0.9053 |     0.7
```

```
|      7 |        50 |    00:00:50 |     80.00% |    78.13% |    0.5422 |    0.(
|      8 |        60 |    00:01:00 |    100.00% |    81.25% |    0.4187 |    0.(
|      9 |        70 |    00:01:09 |     93.33% |    84.38% |    0.3561 |    0.5
|     10 |        80 |    00:01:18 |     73.33% |    84.38% |    0.5141 |    0.4
|     12 |        90 |    00:01:28 |     86.67% |    84.38% |    0.4220 |    0.4
|     13 |       100 |    00:01:37 |     93.33% |    90.63% |    0.1923 |    0.1
|     14 |       110 |    00:01:47 |    100.00% |    90.63% |    0.1472 |    0.1
|     15 |       120 |    00:01:56 |    100.00% |    93.75% |    0.0791 |    0.2
|     17 |       130 |    00:02:05 |     86.67% |    93.75% |    0.2486 |    0.2
|     18 |       140 |    00:02:15 |    100.00% |    93.75% |    0.0386 |    0.2
|     19 |       150 |    00:02:24 |    100.00% |    93.75% |    0.0487 |    0.2
|     20 |       160 |    00:02:33 |    100.00% |    93.75% |    0.0224 |    0.2
|========================================================================
```

Total training time: 1.62e+02 sec

Save only the trained network in a separate `.mat` file. This file will be used by the CUDA executable.

```
ModelFile = fullfile(ResultDir,sprintf('SqueezeNet_Trial%d.mat',trial));
OutMatFile = fullfile('ecg_model.mat');

data = load(ModelFile,'trainedModel');
net = data.trainedModel;
save(OutMatFile,'net');
```

Use the trained network to predict the classes for the validation set.

```
[YPred, probs] = classify(trainedModel,imgsValidation);
accuracy = mean(YPred==imgsValidation.Labels)

accuracy = 0.9375
```

Summarize the performance of the trained network on the validation set with a confusion chart. Display the precision and recall for each class by using column and row summaries. Save the figure. The table at the bottom of the confusion chart shows the precision values. The table to the right of the confusion chart shows the recall values.

```
figure
confusionMat = confusionmat(imgsValidation.Labels,YPred);
confusionchart(imgsValidation.Labels,YPred, ...
    'Title',sprintf('Confusion Matrix on Validation (overall accuracy: %.4f)',accuracy),...
    'ColumnSummary','column-normalized','RowSummary','row-normalized');
```

Confusion Matrix on Validation (overall accuracy: 0.9375)
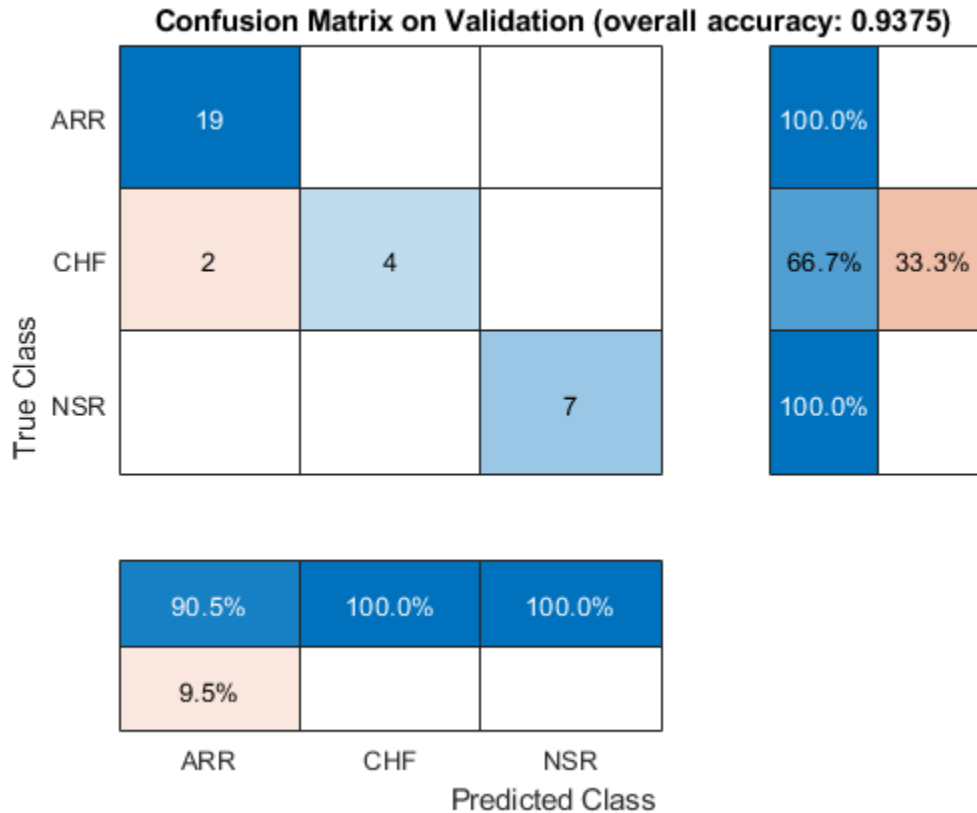

```
AccFigFile = fullfile(ResultDir,sprintf('SqueezeNet_ValidationAccuracy_Trial%d.fig',trial));
saveas(gcf,AccFigFile);
```

Display the size of the trained network.

```
info = whos('trainedModel');
ModelMemSize = info.bytes/1024;
fprintf('Trained network size: %g kB\n',ModelMemSize)
```

Trained network size: 2981.71 kB

Determine the average time it takes the network to classify an image.

```
NumTestForPredTime = 20;
TrialParameter.NumTestForPredTime = NumTestForPredTime;

fprintf('Test prediction time (number of tests: %d)... ',NumTestForPredTime)
```

Test prediction time (number of tests: 20)...

```
imageSize = trainedModel.Layers(1).InputSize;
PredTime = zeros(NumTestForPredTime,1);
for i = 1:NumTestForPredTime
    x = randn(imageSize);
    tic;
    [YPred, probs] = classify(trainedModel,x,'ExecutionEnvironment',ExecutionEnvironment);
    PredTime(i) = toc;
end
```

13-175

```
AvgPredTimePerImage = mean(PredTime);
fprintf('Average prediction time (execution environment: %s): %.2e sec \n',...
    ExecutionEnvironment,AvgPredTimePerImage);
```

```
Average prediction time (execution environment: cpu): 2.97e-02 sec
```

Save the results.

```
if ~LoadModel
    save(MatFile,'accuracy','confusionMat','PredTime','ModelMemSize', ...
        'AvgPredTimePerImage','-append')
end
```

GPU Code Generation — Define Functions

The scalogram of a signal is the input "image" to a deep CNN. Create a function, `cwt_ecg_jetson_ex`, that computes the scalogram of an input signal and returns an image at the user-specified dimensions. The image uses the `jet(128)` colormap. The `%#codegen` directive in the function indicates that the function is intended for code generation. When using the `coder.gpu.kernelfun` pragma, code generation attempts to map the computations in the `cwt_ecg_jetson_ex` function to the GPU.

```
type cwt_ecg_jetson_ex.m
```

```
function im = cwt_ecg_jetson_ex(TimeSeriesSignal, ImgSize) %#codegen
% This function is only intended to support wavelet deep learning examples.
% It may change or be removed in a future release.

coder.gpu.kernelfun();

%% Create Scalogram
cfs = cwt(TimeSeriesSignal, 'morse', 1, 'VoicesPerOctave', 12);
cfs = abs(cfs);

%% Image generation
cmapj128 = coder.load('cmapj128');
imx = ind2rgb_custom_ecg_jetson_ex(round(255*rescale(cfs))+1,cmapj128.cmapj128);

% resize to proper size and convert to uint8 data type
im = im2uint8(imresize(imx, ImgSize));

end
```

Create the entry-point function, `model_predict_ecg.m`, for code generation. The function takes an ECG signal as input and calls the `cwt_ecg_jetson_ex` function to create an image of the scalogram. The `model_predict_ecg` function uses the network contained in the `ecg_model.mat` file to classify the ECG signal.

```
type model_predict_ecg.m
```

```
function PredClassProb = model_predict_ecg(TimeSeriesSignal) %#codegen
% This function is only intended to support wavelet deep learning examples.
% It may change or be removed in a future release.
    coder.gpu.kernelfun();

    % parameters
    ModFile = 'ecg_model.mat'; % file that saves neural network model
    ImgSize = [227 227]; % input image size for the ML model
```

```
    % sanity check signal is a row vector of correct length
    assert(isequal(size(TimeSeriesSignal), [1 65536]))
    %% cwt transformation for the signal
    im = cwt_ecg_jetson_ex(TimeSeriesSignal, ImgSize);

    %% model prediction
    persistent model;
    if isempty(model)
        model = coder.loadDeepLearningNetwork(ModFile, 'mynet');
    end

    PredClassProb = predict(model, im);

end
```

To generate a CUDA executable that can be deployed to an NVIDIA target, create a custom main file (`main_ecg_jetson_ex.cu`) and a header file (`main_ecg_jetson_ex.h`). You can generate an example main file and use that as a template to rewrite new main and header files. For more information, see the `GenerateExampleMain` property of `coder.CodeConfig` (MATLAB Coder). The main file calls the code generated for the MATLAB entry-point function. The main file first reads the ECG signal from a text file, passes the data to the entry-point function, and writes the prediction results to a text file (`predClassProb.txt`). To maximize computation efficiency on the GPU, the executable processes single-precision data.

type main_ecg_jetson_ex.cu

```
//
// File: main_ecg_jetson_ex.cu
//
// This file is only intended to support wavelet deep learning examples.
// It may change or be removed in a future release.

//*************************************************************************
// Include Files
#include "rt_nonfinite.h"
#include "model_predict_ecg.h"
#include "main_ecg_jetson_ex.h"
#include "model_predict_ecg_terminate.h"
#include "model_predict_ecg_initialize.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function Definitions

/* Read data from a file*/
int readData_real32_T(const char * const file_in, real32_T data[65536])
{
  FILE* fp1 = fopen(file_in, "r");
  if (fp1 == 0)
  {
    printf("ERROR: Unable to read data from %s\n", file_in);
    exit(0);
  }
  for(int i=0; i<65536; i++)
  {
      fscanf(fp1, "%f", &data[i]);
```

```
  }
  fclose(fp1);
  return 0;
}

/* Write data to a file*/
int writeData_real32_T(const char * const file_out, real32_T data[3])
{
  FILE* fp1 = fopen(file_out, "w");
  if (fp1 == 0)
  {
    printf("ERROR: Unable to write data to %s\n", file_out);
    exit(0);
  }
  for(int i=0; i<3; i++)
  {
    fprintf(fp1, "%f\n", data[i]);
  }
  fclose(fp1);
  return 0;
}

// model predict function
static void main_model_predict_ecg(const char * const file_in, const char * const file_out)
{
  real32_T PredClassProb[3];
  //   real_T b[65536];
  real32_T b[65536];

  // readData_real_T(file_in, b);
  readData_real32_T(file_in, b);

  model_predict_ecg(b, PredClassProb);

  writeData_real32_T(file_out, PredClassProb);

}

// main function
int32_T main(int32_T argc, const char * const argv[])
{
  const char * const file_out = "predClassProb.txt";
  // Initialize the application.
  model_predict_ecg_initialize();

  // Run prediction function
  main_model_predict_ecg(argv[1], file_out); // argv[1] = file_in

  // Terminate the application.
  model_predict_ecg_terminate();
  return 0;
}

type main_ecg_jetson_ex.h

//
// File: main_ecg_jetson_ex.h
//
```

```
// This file is only intended to support wavelet deep learning examples.
// It may change or be removed in a future release.

//
//***************************************************************************
#ifndef MAIN_H
#define MAIN_H

// Include Files
#include <stddef.h>
#include <stdlib.h>
#include "rtwtypes.h"
#include "model_predict_ecg_types.h"

// Function Declarations
extern int32_T main(int32_T argc, const char * const argv[]);

#endif

//
// File trailer for main_ecg_jetson_ex.h
//
// [EOF]
//
```

GPU Code Generation — Specify Target

To create an executable that can be deployed to the target device, set `CodeGenMode` equal to 1. If you want to create an executable that runs locally and connects remotely to the target device, set `CodeGenMode` equal to 2.

The `main` function reads data from the text file specified by `signalFile` and writes the classification results to `resultFile`. Set `ExampleIndex` to choose a representative ECG signal. You will use this signal to test the executable against the `classify` function. `Jetson_BuildDir` specifies the directory for performing the remote build process on the target. If the specified build directory does not exist on the target, then the software creates a directory with the given name.

```
CodeGenMode = 1 ;
signalFile = 'signalData.txt';
resultFile = 'predClassProb.txt'; % consistent with "main_ecg_jetson_ex.cu"
Jetson_BuildDir = '~/projectECG';
ExampleIndex = 1; % 1,4: type ARR; 2,5: type CHF; 3,6: type NSR

Function_to_Gen = 'model_predict_ecg';
ModFile = 'ecg_model.mat'; % file that saves neural network model; consistent with "main_ecg_jets
ImgSize = [227 227]; % input image size for the ML model

switch ExampleIndex
    case 1 % ARR 7
        SampleSignalIdx = 7;
    case 2 % CHF 97
        SampleSignalIdx = 97;
    case 3 % NSR 132
        SampleSignalIdx = 132;
    case 4 % ARR 31
        SampleSignalIdx = 31;
    case 5 % CHF 101
```

```
        SampleSignalIdx = 101;
    case 6 % NSR 131
        SampleSignalIdx = 131;
end
signal_data = single(ECGData.Data(SampleSignalIdx,:));
ECGtype = ECGData.Labels{SampleSignalIdx};
```

GPU Code Generation — Connect to Hardware

To communicate with the NVIDIA hardware, you create a live hardware connection object using the `jetson` function. You must know the host name or IP address, username, and password of the target board to create a live hardware connection object.

Create a live hardware connection object for the Jetson hardware. During the hardware live object creation checking of hardware, IO server installation and gathering peripheral information on target are performed. This information is displayed in the Command Window.

```
hwobj = jetson('gpucoder-nano-2','ubuntu','ubuntu');
```

```
Checking for CUDA availability on the Target...
Checking for 'nvcc' in the target system path...
Checking for cuDNN library availability on the Target...
Checking for TensorRT library availability on the Target...
Checking for prerequisite libraries is complete.
Gathering hardware details...
Checking for third-party library availability on the Target...

Gathering hardware details is complete.
 Board name          : NVIDIA Jetson TX1, NVIDIA Jetson Nano
 CUDA Version        : 10.0
 cuDNN Version       : 7.3
 TensorRT Version    : 5.0
 GStreamer Version   : 1.14.5
 V4L2 Version        : 1.14.2-1
 SDL Version         : 1.2
 Available Webcams   :
 Available GPUs      : NVIDIA Tegra X1
```

Use the `coder.checkGpuInstall` (GPU Coder) function and verify that the compilers and libraries needed for running this example are set up correctly on the hardware.

```
envCfg = coder.gpuEnvConfig('jetson');
envCfg.DeepLibTarget = 'cudnn';
envCfg.DeepCodegen = 1;
envCfg.HardwareObject = hwobj;
envCfg.Quiet = 1;
coder.checkGpuInstall(envCfg)
```

```
ans = struct with fields:
                 gpu: 1
                cuda: 1
               cudnn: 1
            tensorrt: 0
         basiccodegen: 0
        basiccodeexec: 0
          deepcodegen: 1
          deepcodeexec: 0
     tensorrtdatatype: 0
```

```
        profiling: 0
```

GPU Code Generation — Compile

Create a GPU code configuration object necessary for compilation. Use the `coder.hardware` function to create a configuration object for the Jetson platform and assign it to the `Hardware` property of the code configuration object `cfg`. Use `'NVIDIA Jetson'` for the Jetson TX1 or TX2 boards. The custom main file is a wrapper that calls the entry-point function in the generated code. The custom file is required for a deployed executable.

Use the `coder.DeepLearningConfig` (GPU Coder) function to create a `CuDNN` deep learning configuration object and assign it to the `DeepLearningConfig` property of the GPU code configuration object. The code generator takes advantage of NVIDIA® CUDA® deep neural network library (cuDNN) for NVIDIA GPUs. cuDNN is a GPU-accelerated library of primitives for deep neural networks.

```
if CodeGenMode == 1
    cfg = coder.gpuConfig('exe');
    cfg.Hardware = coder.hardware('NVIDIA Jetson');
    cfg.Hardware.BuildDir = Jetson_BuildDir;
    cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');
    cfg.CustomSource = fullfile('main_ecg_jetson_ex.cu');
elseif CodeGenMode == 2
    cfg = coder.gpuConfig('lib');
    cfg.VerificationMode = 'PIL';
    cfg.Hardware = coder.hardware('NVIDIA Jetson');
    cfg.Hardware.BuildDir = Jetson_BuildDir;
    cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');
end
```

To generate CUDA code, use the `codegen` function and pass the GPU code configuration along with the size and type of the input for the `model_predict_ecg` entry-point function. After code generation on the host is complete, the generated files are copied over and built on the target.

```
codegen('-config ',cfg,Function_to_Gen,'-args',{signal_data},'-report');
```

```
Code generation successful: View report
```

GPU Code Generation — Execute

If you compiled an executable to be deployed to the target, write the example ECG signal to a text file. Use the `putFile()` function of the hardware object to place the text file on the target. The `workspaceDir` property contains the path to the `codegen` folder on the target.

```
if CodeGenMode == 1
    fid = fopen(signalFile,'w');
    for i = 1:length(signal_data)
        fprintf(fid,'%f\n',signal_data(i));
    end
    fclose(fid);
    hwobj.putFile(signalFile,hwobj.workspaceDir);
end
```

Run the executable.

When running the deployed executable, delete the previous result file if it exists. Use the `runApplication()` function to launch the executable on the target hardware, and then the

getFile() function to retrieve the results. Because the results may not exist immediately after the runApplication() function call returns, and to allow for communication delays, set a maximum time for fetching the results to 90 seconds. Use the evalc function to suppress the command-line output.

```matlab
if CodeGenMode == 1 % run deployed executable
    maxFetchTime = 90;
    resultFile_hw = fullfile(hwobj.workspaceDir,resultFile);
    if ispc
        resultFile_hw = strrep(resultFile_hw,'\','/');
    end

    ta = tic;

    hwobj.deleteFile(resultFile_hw)
    hwobj.runApplication(Function_to_Gen,signalFile);

    tf = tic;
    success = false;
    while toc(tf) < maxFetchTime
        try
            evalc('hwobj.getFile(resultFile_hw)');
            success = true;
        catch ME
        end
        if success
            break
        end
    end
    fprintf('Fetch time = %.3e sec\n',toc(tf));
    assert(success,'Unable to fetch the prediction')
    PredClassProb = readmatrix(resultFile);
    PredTime = toc(ta);
elseif CodeGenMode == 2 % run PIL executable
    ta = tic;
    eval(sprintf('PredClassProb = %s_pil(signal_data);',Function_to_Gen));
    PredTime = toc(ta);
    eval(sprintf('clear %s_pil;',Function_to_Gen)); % terminate PIL execution
end
```

```
### Launching the executable on the target...
Executable launched successfully with process ID 15078.
Displaying the simple runtime log for the executable...

Note: For the complete log, run the following command in the MATLAB command window:
system(hwobj,'cat /home/ubuntu/projectECG/MATLAB_ws/R2020b/C/Users/pkostele/OneDrive_-_MathWorks,

Fetch time = 9.927e+00 sec
```

Use the classify function to predict the class labels for the example signal.

```matlab
ModData = load(ModFile,'net');
im = cwt_ecg_jetson_ex(signal_data,ImgSize);
[ModPred, ModPredProb] = classify(ModData.net,im);
PredCat = categories(ModPred)';
```

Compare the results.

```
PredTableJetson = array2table(PredClassProb(:)','VariableNames',matlab.lang.makeValidName(PredCa
fprintf('tPred = %.3e sec\nExample ECG Type: %s\n',PredTime,ECGtype)
```

```
tPred = 1.302e+01 sec
Example ECG Type: ARR
```

```
disp(PredTableJetson)
```

ARR	CHF	NSR
0.99872	0.001131	0.000153

```
PredTableMATLAB = array2table(ModPredProb(:)','VariableNames',matlab.lang.makeValidName(PredCat)
disp(PredTableMATLAB)
```

ARR	CHF	NSR
0.99872	0.0011298	0.00015316

Close the hardware connection.

```
clear hwobj
```

Summary

This example shows how to create and deploy a CUDA executable that uses a CNN to classify ECG signals. You also have the option to create an executable the runs locally and connects to the remote target. A complete workflow is presented in this example. After the data is downloaded, the CWT is used to extract features from the ECG signals. Then SqueezeNet is retrained to classify the signals based on their scalograms. Two user-defined functions are created and compiled on the target NVIDIA device. Results of the executable are compared with MATLAB.

References

1   Baim, D. S., W. S. Colucci, E. S. Monrad, H. S. Smith, R. F. Wright, A. Lanoue, D. F. Gauthier, B. J. Ransil, W. Grossman, and E. Braunwald. "Survival of patients with severe congestive heart failure treated with oral milrinone." *Journal of the American College of Cardiology*. Vol. 7, Number 3, 1986, pp. 661–670.

2   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, and H. E. Stanley. "PhysioBank, PhysioToolkit,and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals." Circulation. Vol. 101, Number 23: e215–e220. [Circulation Electronic Pages; `http://circ.ahajournals.org/content/101/23/e215.full`]; 2000 (June 13). doi: 10.1161/01.CIR.101.23.e215.

3   Moody, G. B., and R. G. Mark. "The impact of the MIT-BIH Arrhythmia Database." *IEEE Engineering in Medicine and Biology Magazine*. Vol. 20. Number 3, May-June 2001, pp. 45–50. (PMID: 11446209)

Supporting Functions

helperCreateECGDirectories

```
function helperCreateECGDirectories(ECGData,parentFolder,dataFolder)
% This function is only intended to support wavelet deep learning examples.
% It may change or be removed in a future release.
```

```matlab
rootFolder = parentFolder;
localFolder = dataFolder;
mkdir(fullfile(rootFolder,localFolder))

folderLabels = unique(ECGData.Labels);
for i = 1:numel(folderLabels)
    mkdir(fullfile(rootFolder,localFolder,char(folderLabels(i))));
end
end
```

helperPlotReps

```matlab
function helperPlotReps(ECGData)
% This function is only intended to support wavelet deep learning examples.
% It may change or be removed in a future release.

folderLabels = unique(ECGData.Labels);

for k=1:3
    ecgType = folderLabels{k};
    ind = find(ismember(ECGData.Labels,ecgType));
    subplot(3,1,k)
    plot(ECGData.Data(ind(1),1:1000));
    grid on
    title(ecgType)
end
end
```

helperCreateRGBfromTF

```matlab
function helperCreateRGBfromTF(ECGData,parentFolder, childFolder)
% This function is only intended to support wavelet deep learning examples.
% It may change or be removed in a future release.

imageRoot = fullfile(parentFolder,childFolder);

data = ECGData.Data;
labels = ECGData.Labels;

[~,signalLength] = size(data);

fb = cwtfilterbank('SignalLength',signalLength,'VoicesPerOctave',12);
r = size(data,1);

for ii = 1:r
    cfs = abs(fb.wt(data(ii,:)));
    im = ind2rgb(im2uint8(rescale(cfs)),jet(128));

    imgLoc = fullfile(imageRoot,char(labels(ii)));
    imFileName = strcat(char(labels(ii)),'_',num2str(ii),'.jpg');
    imwrite(imresize(im,[227 227]),fullfile(imgLoc,imFileName));
end
end
```

See Also
coder.DeepLearningConfig | cwt | cwtfilterbank

More About

- "GPU Acceleration of Scalograms for Deep Learning" on page 13-134

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

> **Note** The **Wavelet 1-D — Denoising** tool is no longer recommended. Use **Wavelet Signal Denoiser** instead.

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`.
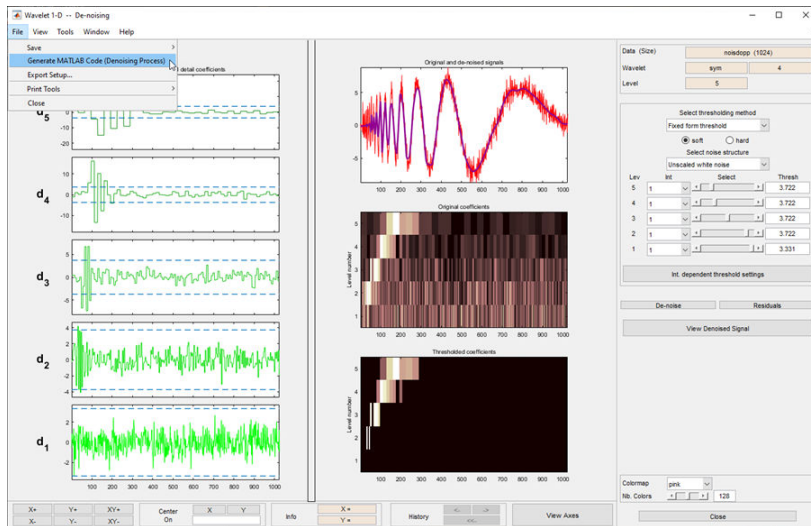
Denoise Doppler Signal

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**.
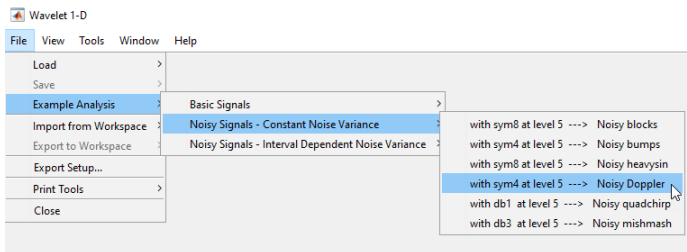
After selecting the analysis, the wavelet decomposition appears.
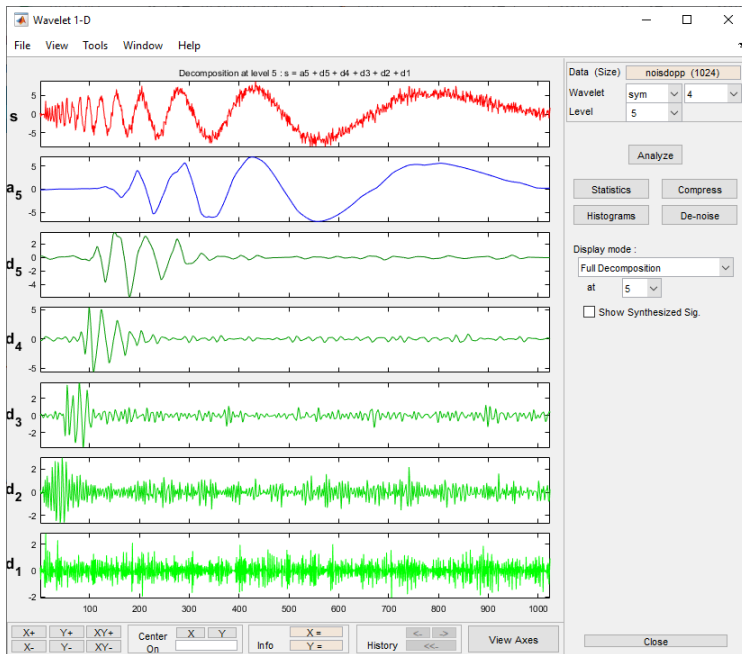
4 Click **Denoise**.

5 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 — 3.500
- level 4 — 3.720
- level 3 — 3.000
- level 2 — 2.000
- level 1 — 3.000
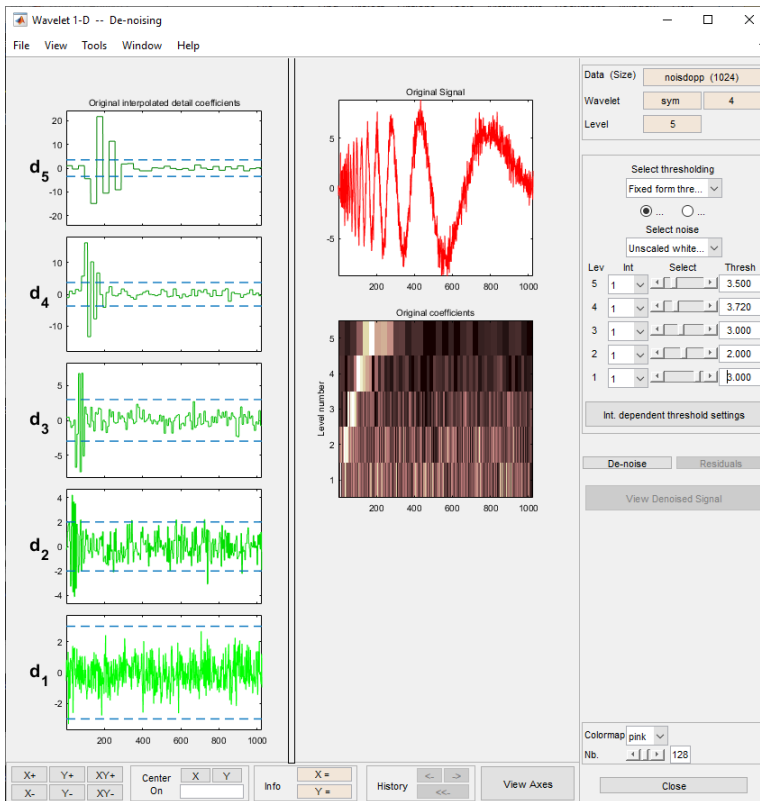
Click **Denoise**.

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

The operation generates the following MATLAB code.

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


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

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

```
        ];

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

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

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

8 Export the denoised signal from the app by selecting **File > Save > Denoised Signal**.
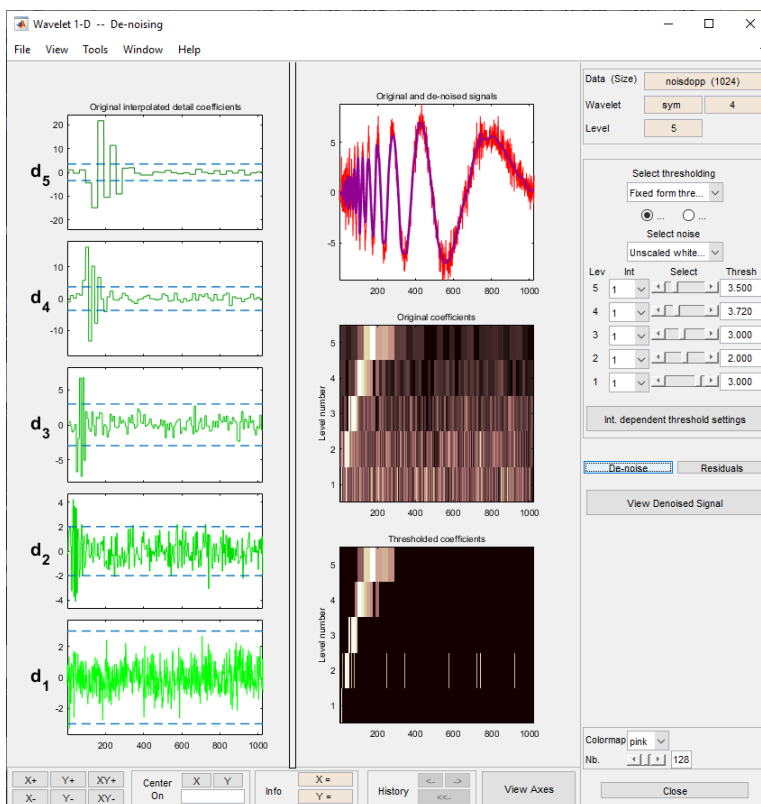
Save the denoised signal as `denoiseddoppler.mat` in a folder on the MATLAB search path. Load `denoiseddoppler.mat` in the MATLAB workspace. Compare `denoiseddoppler` with your command line result.

```
load denoiseddoppler;
plot(sigDEN,'k'); axis tight;
hold on;
plot(denoiseddoppler,'r');
legend('Command Line','GUI','Location','SouthEast');
```
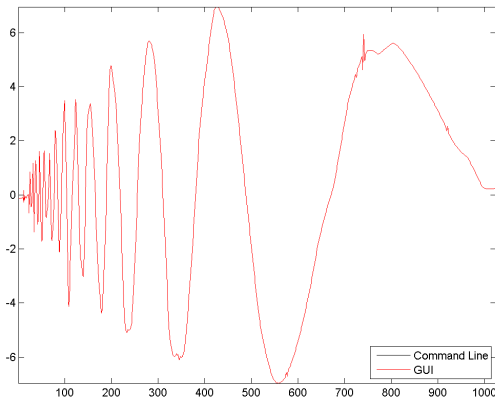
Interval Dependent 1-D Wavelet Denoising

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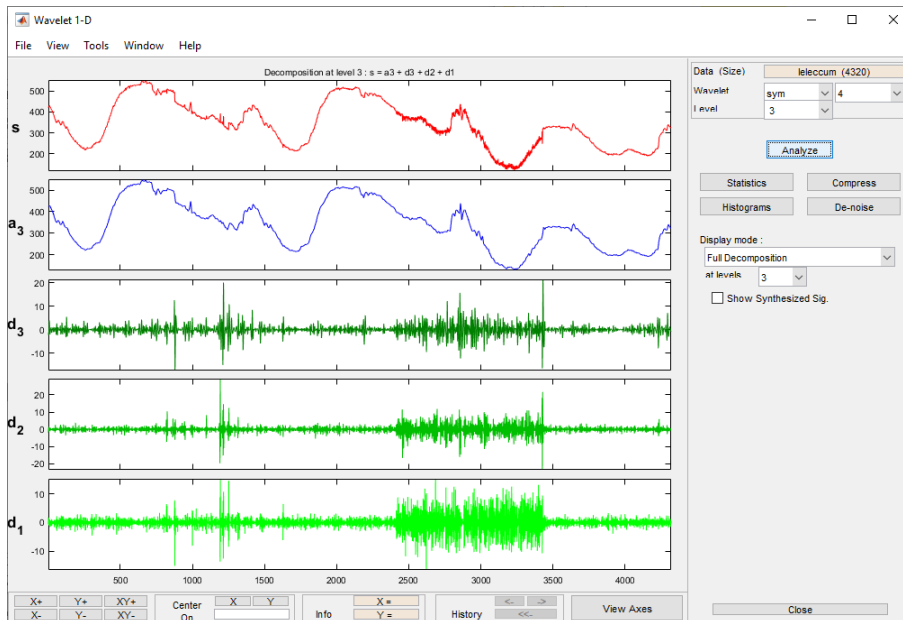
When you inspect the original signal and the finest-scale wavelet coefficients, you see that the noise variance is not constant. In this situation, interval-dependent thresholding is useful. To implement interval-dependent denoising:

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_dw1d(SIG)
% FUNC_DENOISE_DW1D Saved Denoising Process.
%   SIG: vector of data
%   -------------------
%   sigDEN: vector of denoised data


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

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

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

8   To avoid confusion with the MATLAB code generated in "Denoise Doppler Signal" on page 14-2, change the function definition line. Change the function definition to:

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

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

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

Execute the following code.

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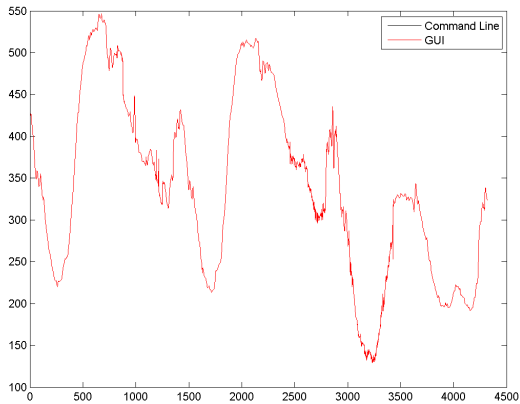

See Also
Wavelet Signal Denoiser | wdenoise

More About
- "Denoise a Signal with the Wavelet Signal Denoiser" on page 6-25

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

In this section...
"2-D Decimated Discrete Wavelet Transform Denoising" on page 14-9
"2-D Decimated Discrete Wavelet Transform Compression" on page 14-11

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)**.
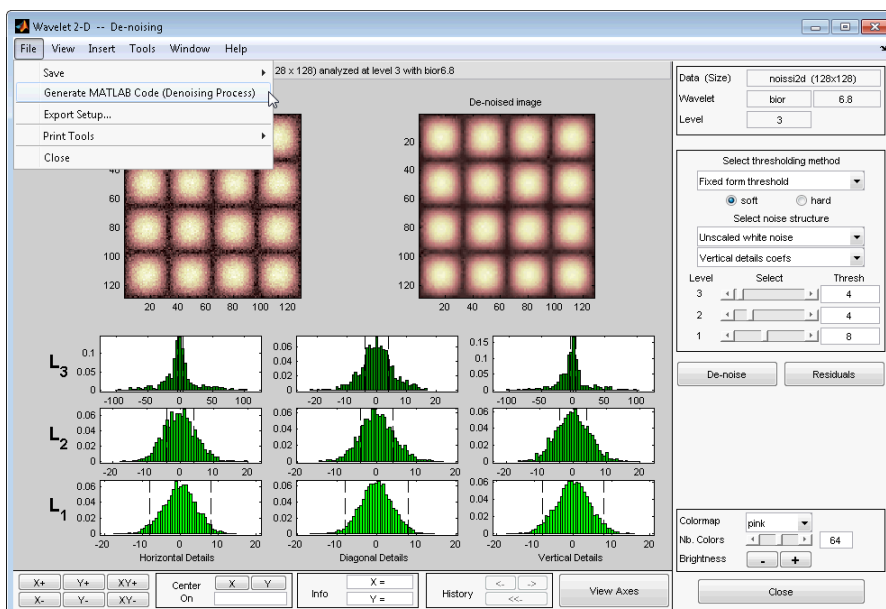
The operation generates the following MATLAB code.

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


% Analysis parameters.
%---------------------
wname = 'bior6.8';
level = 3;

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

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

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

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

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

8 Save your denoised image in a folder on the MATLAB search path as `denoisedsin.mat`.
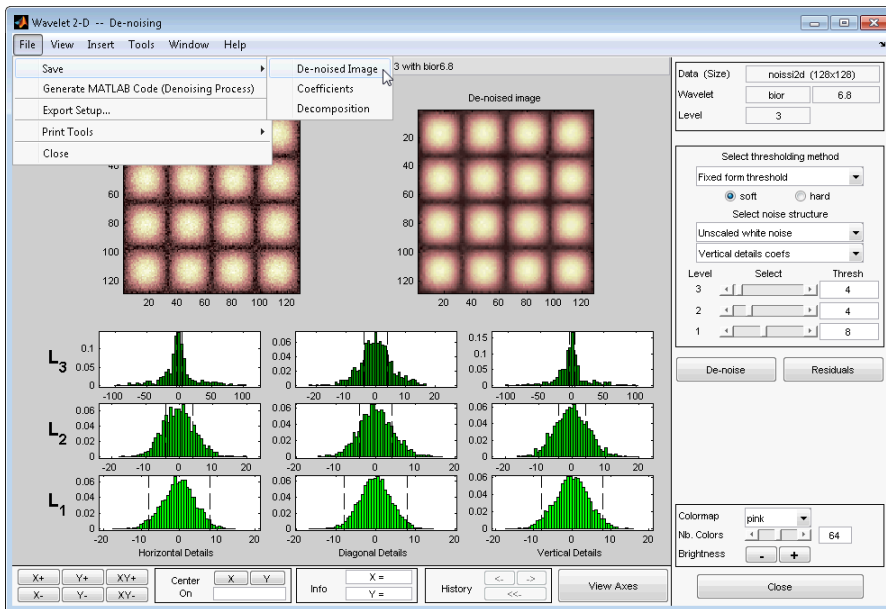
Load the denoised image in the MATLAB workspace. Compare the result with your generated code.

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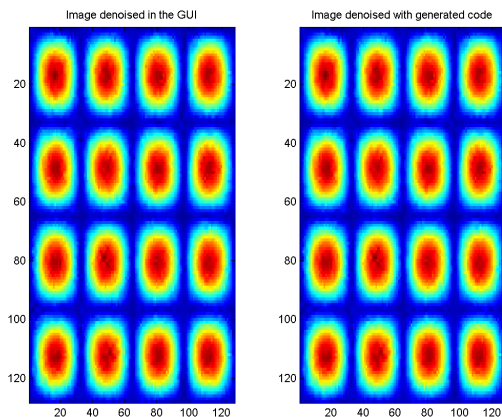

2-D Decimated Discrete Wavelet Transform Compression

You can generate MATLAB code to reproduce 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';
sorh = 'h';     % Specified soft or hard thresholding
thrSettings = 10.064453124999996;
roundFLAG = true;

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

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

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

10 Save the compressed image from the **Wavelet 2-D - - Compression** tool in a folder on the MATLAB search path. Use **File > Save > Compressed Image**, and name the file `compressed_fingerprint.mat`. Execute the following code.

```matlab
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.
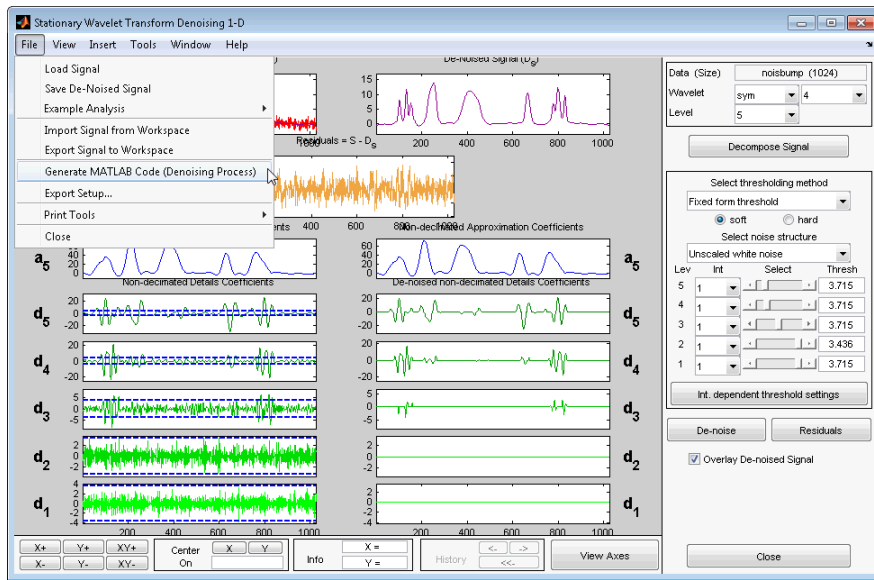
1-D Stationary Wavelet Transform Denoising

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**
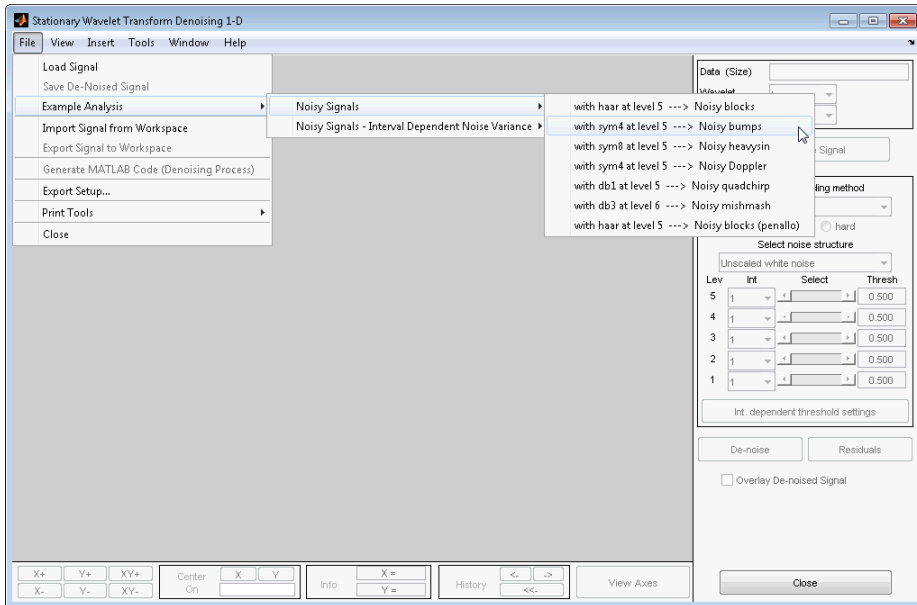
4 Set the thresholds as follows:

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

Click **Denoise**.
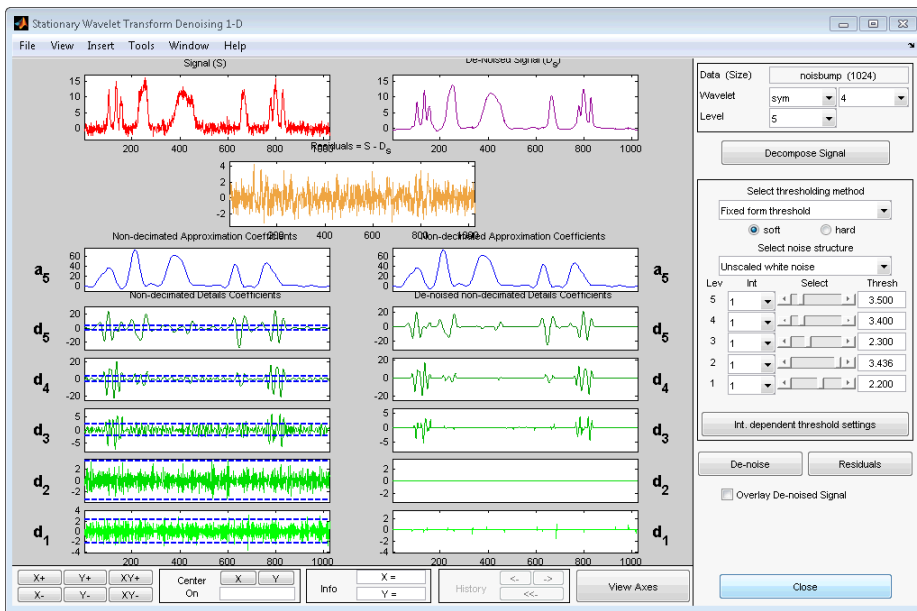
5 Generate the MATLAB code with **File > Generate Matlab Code (Denoising Process)**.
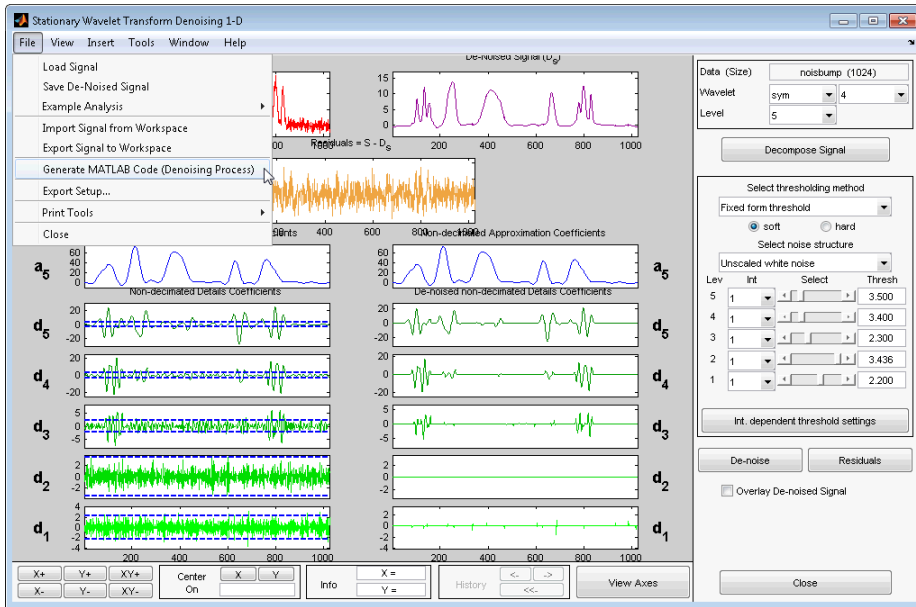
The operation generates the following MATLAB code.

```
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;
sorh = 's';    % Specified soft or hard thresholding
thrParams =  {...
    [...
    1.00000000  1024.00000000     3.50000000; ...
    ]; ...
    [...
    1.00000000  1024.00000000     3.40000000; ...
    ]; ...
    [...
    1.00000000  1024.00000000     2.30000000; ...
    ]; ...
    [...
    1.00000000  1024.00000000     5.29965570; ...
    ]; ...
    [...
    1.00000000  1024.00000000     2.20000000; ...
    ]; ...
    };

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

% Denoise.
%---------
len = length(SIG);
```

```
for k = 1:level
    thr_par = thrParams{k};
    if ~isempty(thr_par)
        NB_int = size(thr_par,1);
        x      = [thr_par(:,1) ; thr_par(NB_int,2)];
        x      = round(x);
        x(x<1) = 1;
        x(x>len) = len;
        thr = thr_par(:,3);
        for j = 1:NB_int
            if j==1 , d_beg = 0; else d_beg = 1; end
            j_beg = x(j)+d_beg;
            j_end = x(j+1);
            j_ind = (j_beg:j_end);
            wDEC(k,j_ind) = wthresh(wDEC(k,j_ind),sorh,thr(j));
        end
    end
end

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

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

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

7 Select **File > Save Denoised Signal**, and save the denoised signal as `denoisedbumps.mat` in a folder on the MATLAB search path.
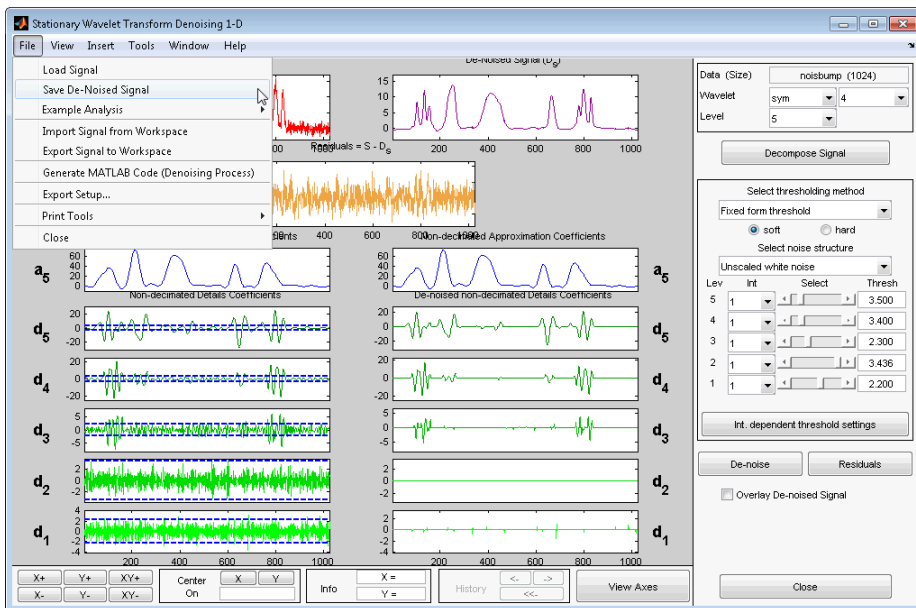
Execute the following code.

```
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.
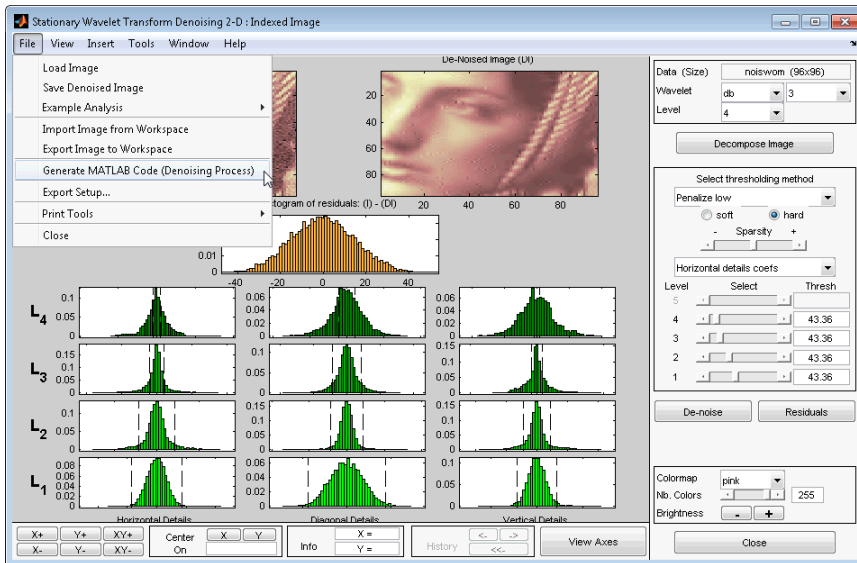
2-D Stationary Wavelet Transform Denoising

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;
sorh = '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.
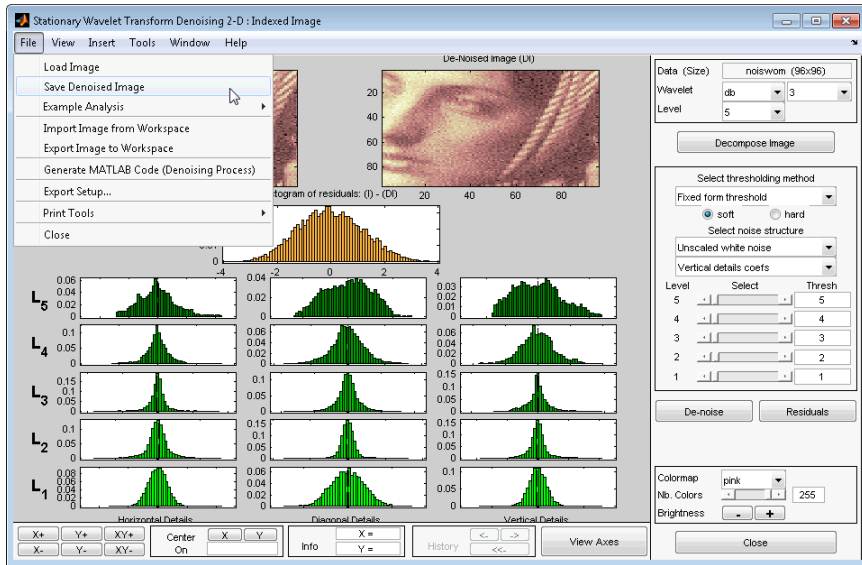
12 Execute the following code.

```
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.
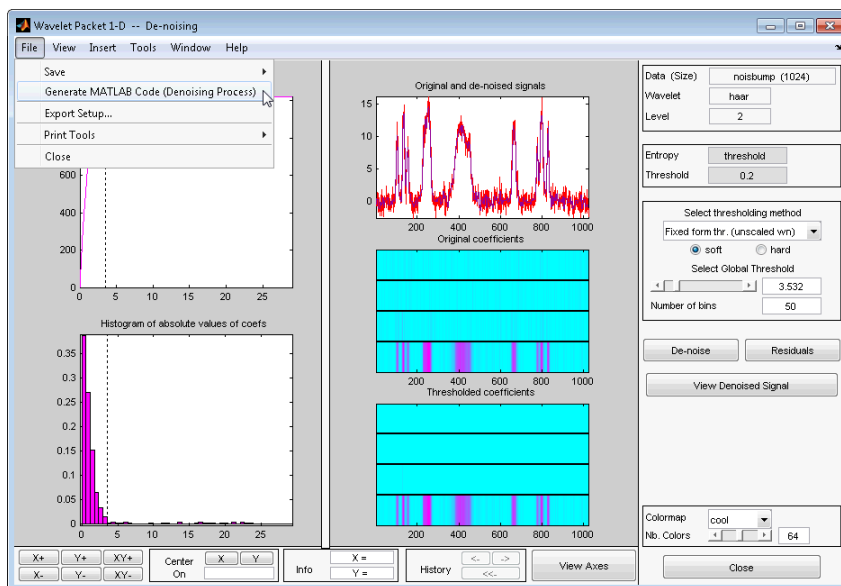

1. Enter `waveletAnalyzer` at the MATLAB command prompt.

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 = 'sqtwologuwn';
sorh = 's';     % Specified soft or hard thresholding
thrSettings = {sorh,'nobest',2.750000000000000,1};

% Decompose using WPDEC.
%----------------------
wpt = wpdec(SIG,Lev_Anal,Wav_Nam,Ent_Nam,Ent_Par);

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

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

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

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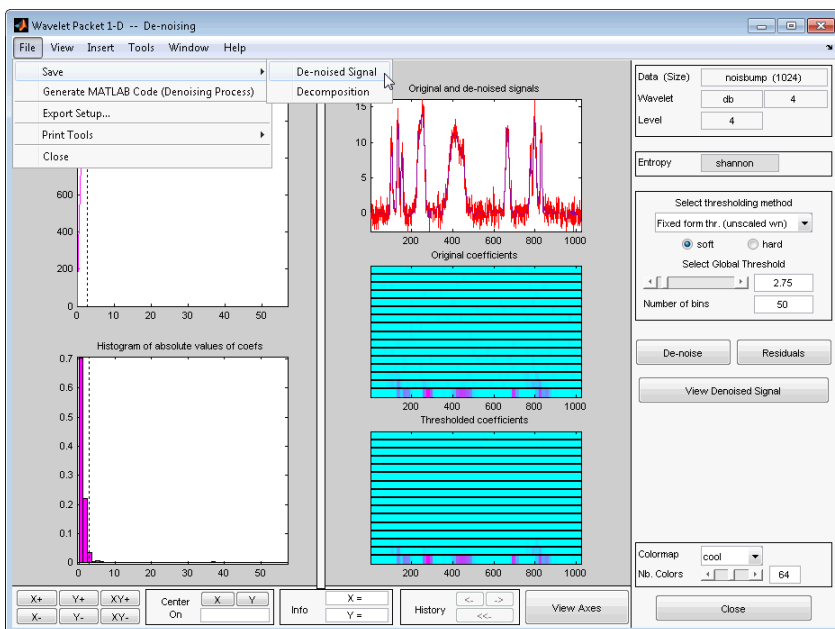
Execute the following code.

```
load noisbump;
[sigDEN,wptDEN] = func_denoise_wp1d(noisbump);
load wp_denoisedbump;
plot(sigDEN); title('Denoised Signal');
axis([1 1024 min(sigDEN)-1 max(sigDEN+1)]);
norm(sigDEN-wp_denoisedbump,2)
```
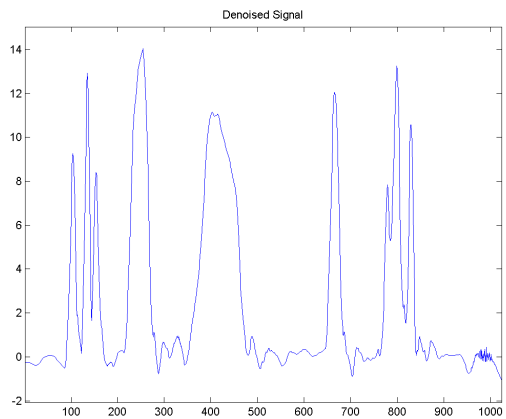
Generate MATLAB Code for 2-D Wavelet Packet Denoising and Compression

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`.
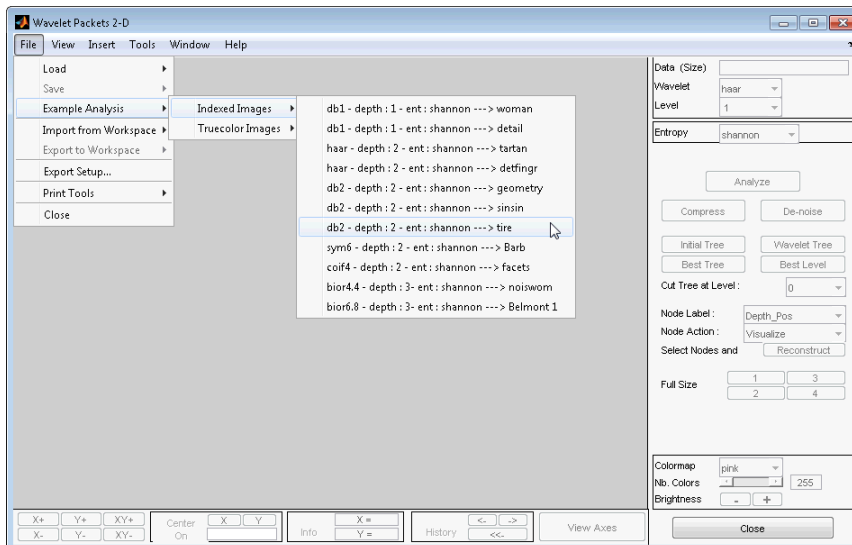
4 Using the default parameter settings, click **Best Tree**.
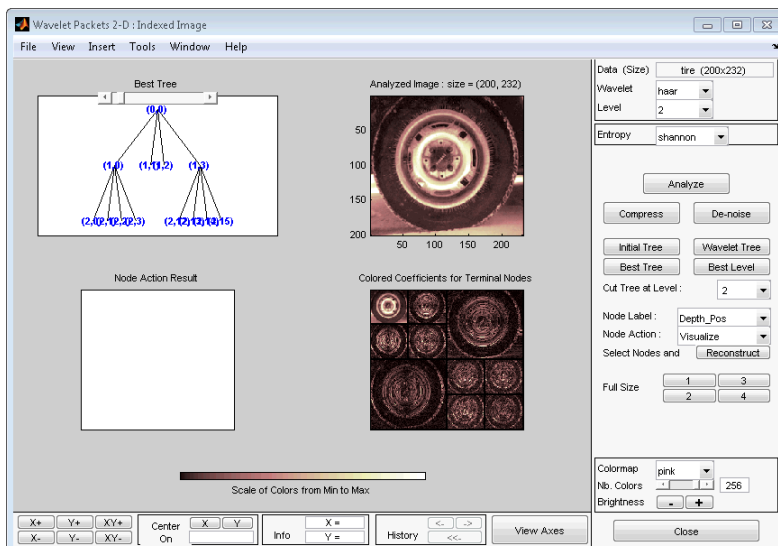
5 Click **Compress**.

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

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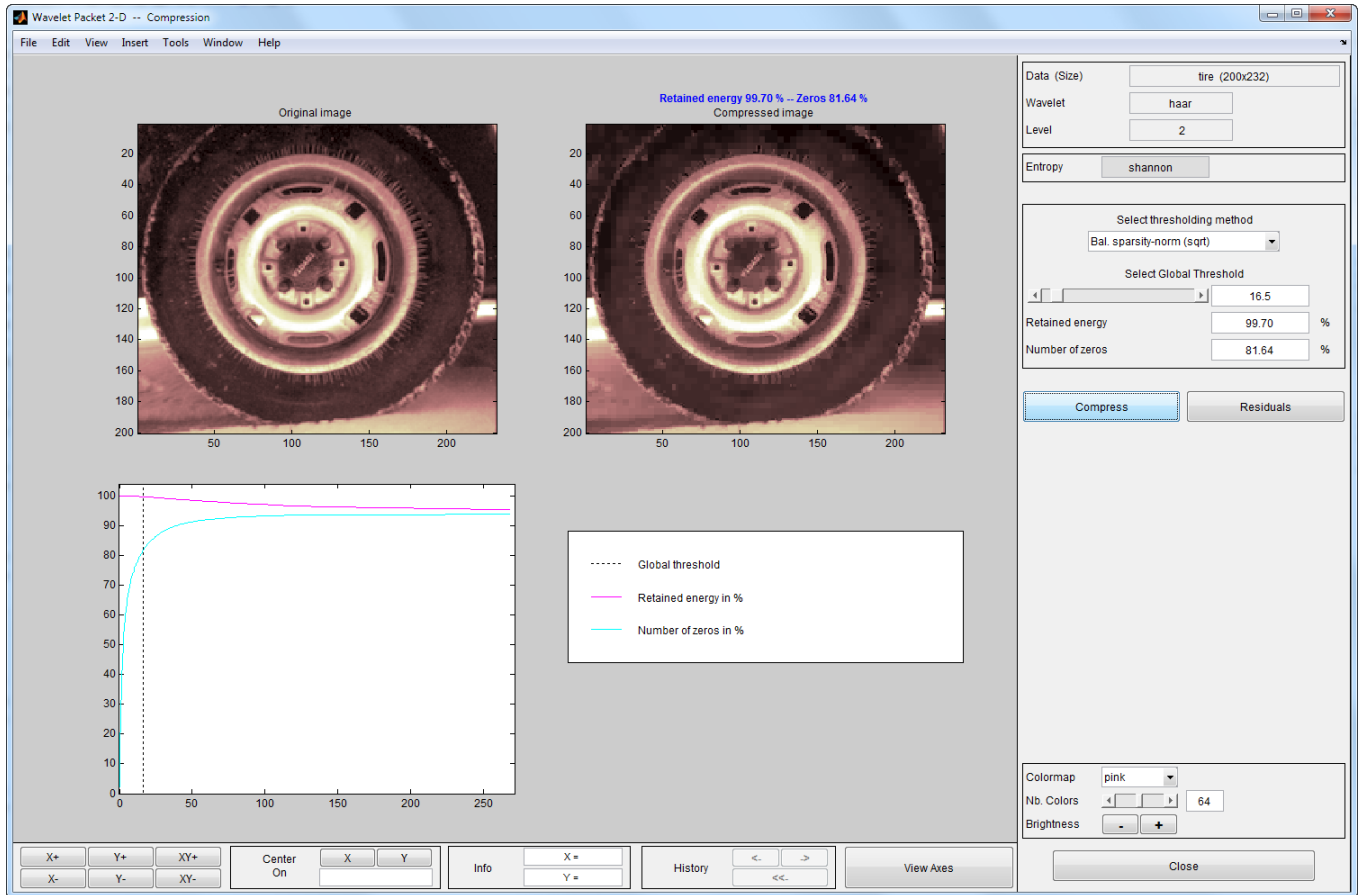
8 **File > Generate Code (Compression Process)** generates the following code.

```
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)
```

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.

Signal	Shown In
Original	Red
Reconstructed or synthesized	Yellow
Approximations	Variegated shades of blue (high level = darker)
Details	Variegated shades of green (high level = darker)

Connection of Plots

Plots containing related information and graphed on the same abscissa are connected in the sense that manipulations performed on one plot affect all others in the same way. For images, the connection holds in both abscissa and ordinate. You can manipulate all plots along an individual axis (X or Y) or you can manipulate all plots along both axes at the same time (XY).

For example, the approximations and details shown in the separate mode view of a decomposition all respond together when any of the plots is magnified or zoomed.

Click and drag your mouse over the region you want to zoom. Clicking **XY+** results in the zoom being applied to all the plots.
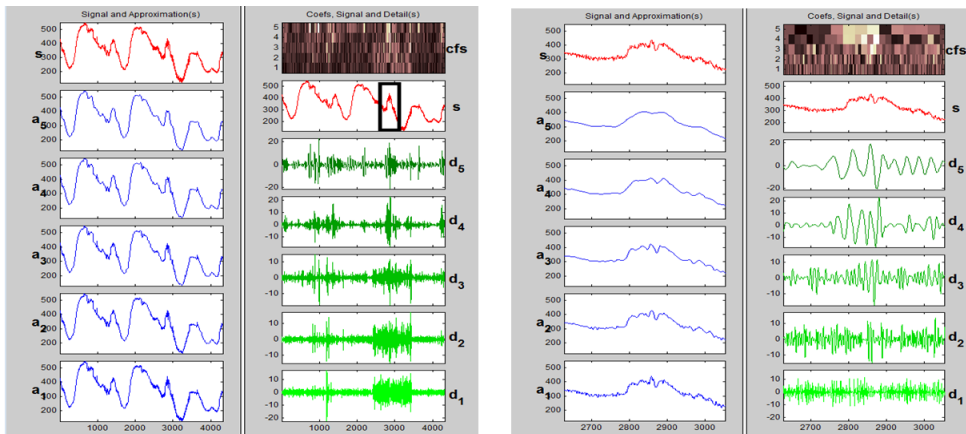
- Zoom in on relevant detail.

  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.
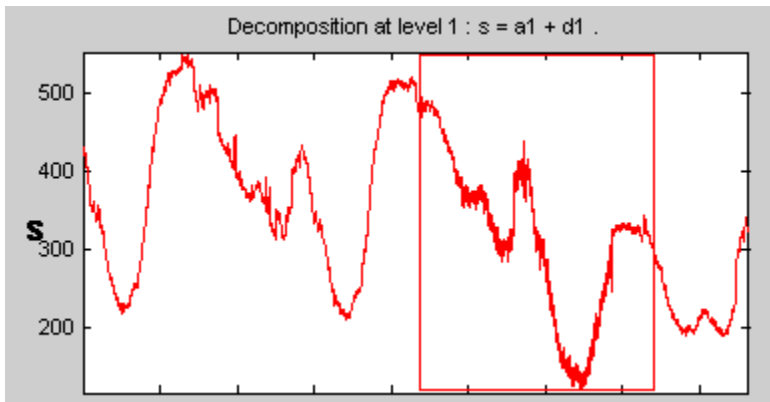
  Click the **X+** button (located at the bottom of the screen) to zoom horizontally.
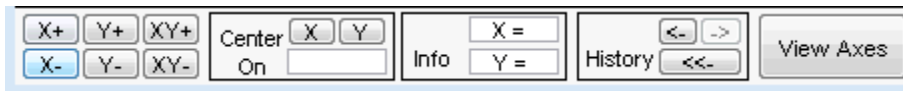
  The **Wavelet 1-D** tool zooms all the displayed signals.
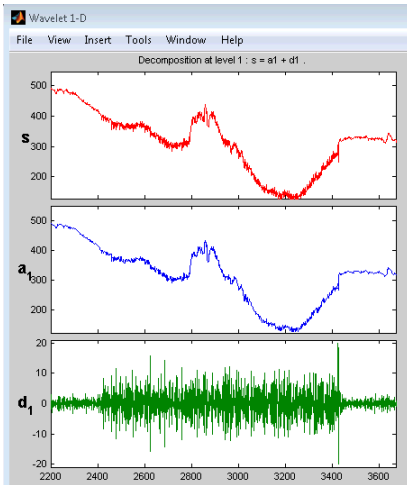
The other zoom controls do more or less what you'd expect them to. The **X-** button, for example, zooms out horizontally. The history function keeps track of all your views of the signal. Return to a previous zoom level by clicking the left arrow button.

- **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.
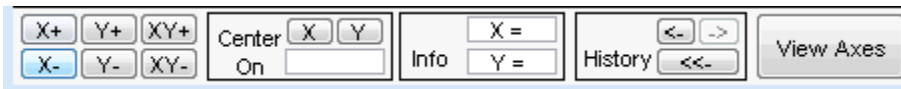
- The **History** pane enables you to remember how you zoom the axes so that you can toggle back and forth between views.

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.
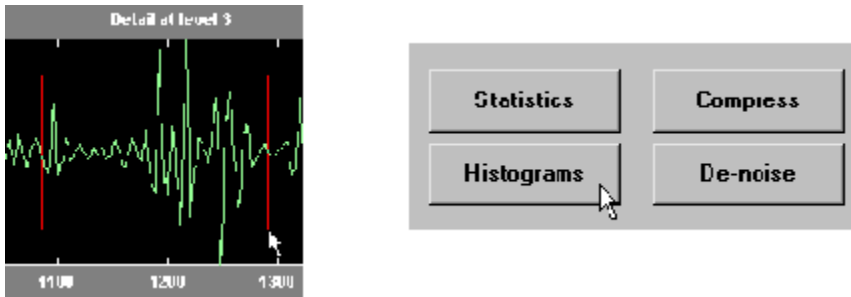
Translating Plots

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.
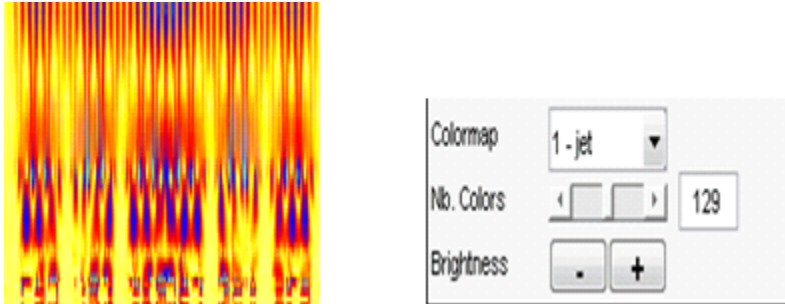
and here using 68 colors.
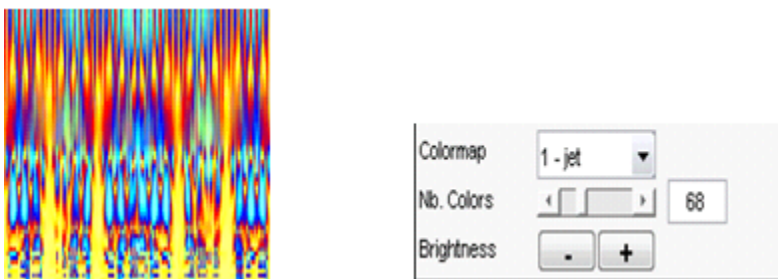
Controlling the Coloration Mode

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.

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.
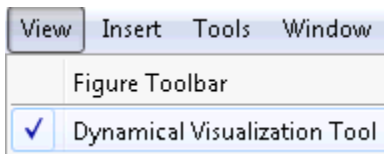
Enabling the **Dynamical Visualization Tool** activates the zoom, center, history, and axes options at the bottom of the interactive tool.

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.
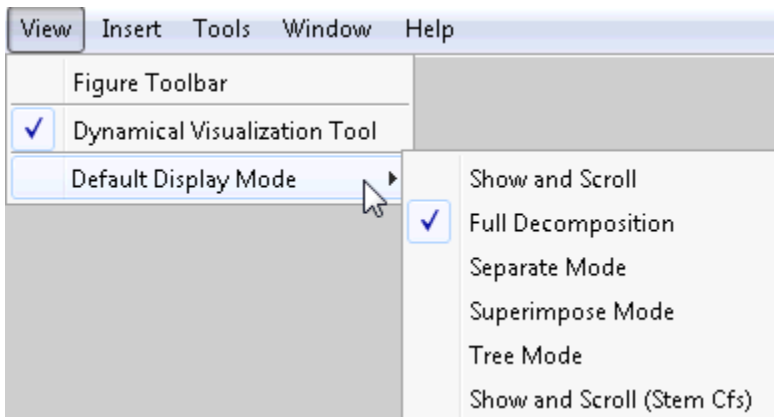
Using the View Axes Button

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.
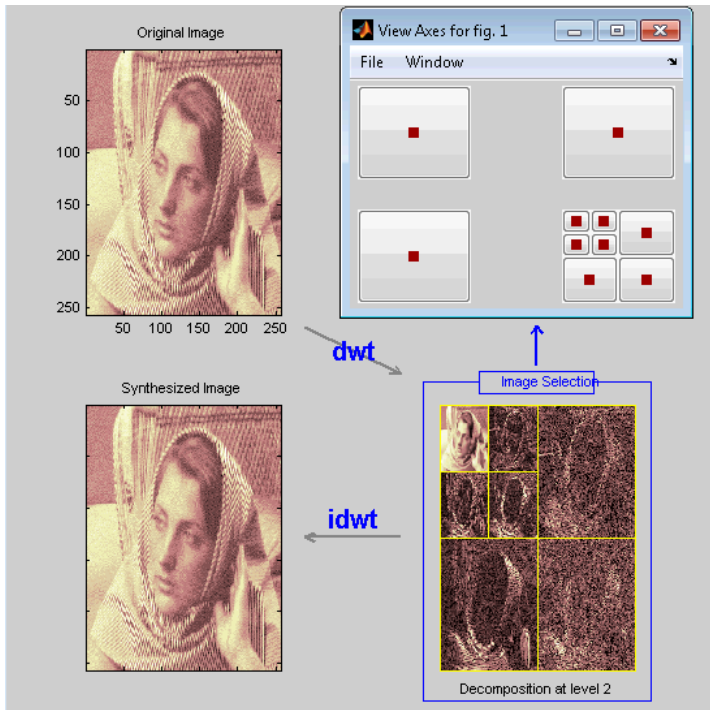
The toggle buttons in the **View Axes** figure are positioned so that you can understand which axis is correlated with a button.

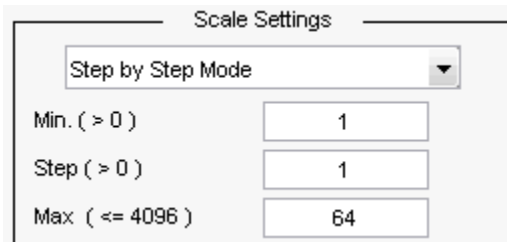When you click the same toggle button again, you restore the original view.

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.
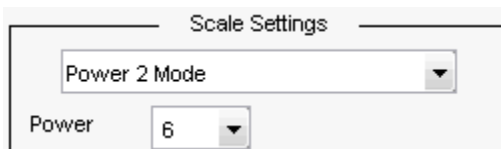
Scale Settings

- `Step by Step Mode` — Specify the initial scale, the step size, and the maximum scale.
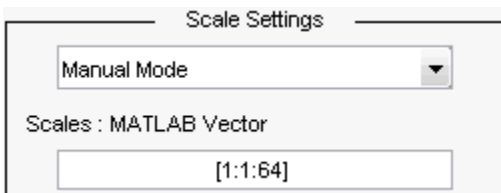
- `Power 2 Mode` — The scales are 2^0, 2^1, up to power you select in the `Power` drop down menu. These are the same scales used for discrete analysis.

- `Manual Mode` — Enter a vector of scales.

Wavelet 2-D Tool Features

The **Wavelet 2-D** tool is described in "2-D Wavelet Analysis Using the Wavelet Analyzer App" on page 3-155. Here is an example of an option that allows you to view a selected part of the window at a full window resolution.
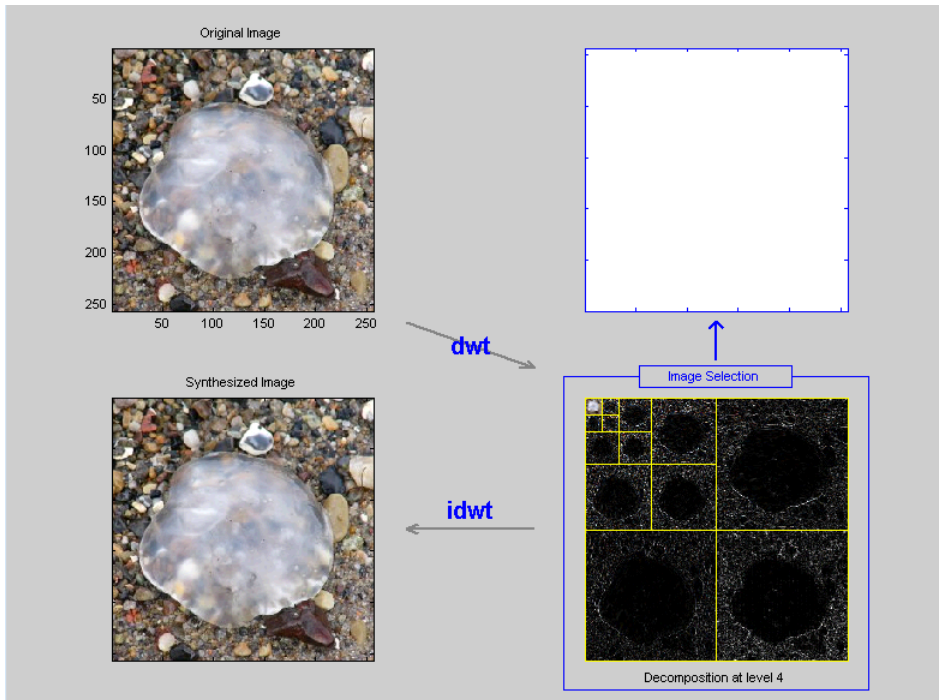
In the **Full Size** menu on the right side of the interactive tool

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.
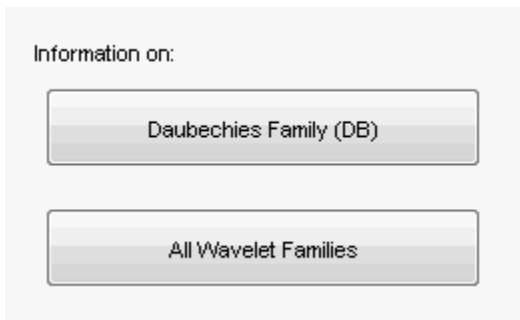
Wavelet Display Tool

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.
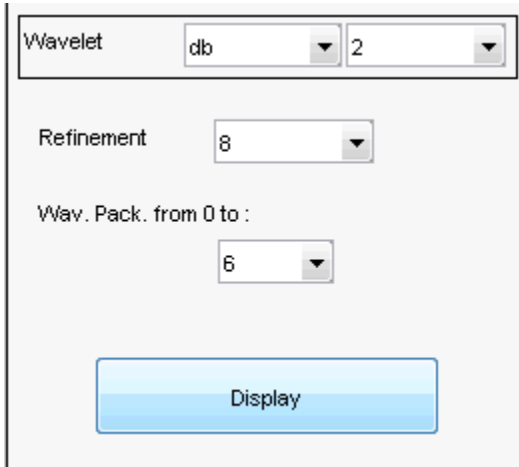
The **Information on:** selections allow you to obtain more detailed information on the current wavelet family, or all supported families.
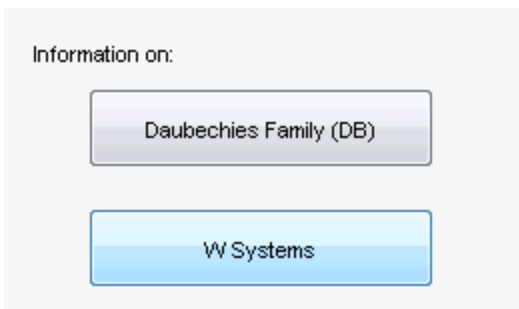
Wavelet Packet Display Tool

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 `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**.