

Signal Processing Toolbox

For Use with **MATLAB®**

- Computation
- Visualization
- Programming

Getting Started

Version 6



How to Contact The MathWorks



www.mathworks.com
comp.soft-sys.matlab
www.mathworks.com/contact_TS.html

Web
Newsgroup
Technical Support



suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
service@mathworks.com
info@mathworks.com

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

Getting Started with the Signal Processing Toolbox

© COPYRIGHT 2006 by The MathWorks, Inc.

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

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

Trademarks

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks, and SimBiology, SimEvents, and SimHydraulics are trademarks of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Patents

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

Revision History

September 2006 First printing New for Version 6.6 (Release 2006b)

Overview

1

What Is the Signal Processing Toolbox?	1-2
Central Features	1-4
Signals and Systems	1-4
Filter Design, Analysis, and Implementation	1-4
Linear System Transformations	1-4
Windows	1-4
Spectral Analysis	1-4
Transforms	1-5
Statistical Signal Processing	1-5
Parametric Modeling	1-5
Linear Prediction	1-5
Multirate Signal Processing	1-5
Waveform Generation	1-5
Other Operations	1-6
Interactive Tools	1-7
Extensibility	1-8
Finding More Information	1-9

Basic Signal Processing Concepts

2

Representing Signals	2-2
Vector Representation	2-2
Waveform Generation: Time Vectors and Sinusoids ...	2-4

Common Sequences: Unit Impulse, Unit Step, and Unit Ramp	2-5
Multichannel Signals	2-5
Common Periodic Waveforms	2-5
Common Aperiodic Waveforms	2-6
The pulstran Function	2-7
The Sinc Function	2-8
The Dirichlet Function	2-9
Working with Data	2-11
Selected Bibliography	2-12

Filter Design with the FDATool GUI

3

Introduction	3-2
Designing the Filter	3-3
Analyzing the Filter	3-8
Designing Additional Filters	3-10
Viewing and Annotating the Filter	3-11
Viewing the Filter in FVTool	3-11
Using FVTool for Annotation	3-15
Exporting Filters from FDATool	3-17
Using Discrete Filter Objects (dfilts)	3-19
Filtering with dfilt	3-19
Where to Find More Information	3-22

4

Introduction	4-2
Creating a Spectral Analysis Object	4-4
Generating a Spectrum	4-6
Changing Spectral Analysis Object Property Values ..	4-9
Using the set command to Set Property Values	4-9
Using Options Objects to Set Property Values	4-10
Measuring Signal Power	4-11
Where to Find More Information	4-15

Index

Overview

What Is the Signal Processing
Toolbox? (p. 1-2)

Central Features (p. 1-4)

Interactive Tools (p. 1-7)

Extensibility (p. 1-8)

Finding More Information (p. 1-9)

Major features and key areas of the
toolbox

Key areas of the toolbox

Descriptions of available graphical
user interfaces (GUIs)

Extending the toolbox with M-files
and other MathWorks products

Where to find more detailed and
advanced information

What Is the Signal Processing Toolbox?

The Signal Processing Toolbox is a collection of tools based on the MATLAB® numeric computing environment. The toolbox supports a wide range of signal processing operations, from waveform generation to filter design and implementation, parametric modeling, and spectral analysis. The toolbox provides two categories of tools, command-line functions and graphical user interfaces:

Command-line functions are available in the following categories:

- Discrete-time filter design, analysis, and implementation
- Analog filter design, analysis, and implementation
- Linear system transformations
- Windows
- Spectral analysis and cepstral analysis
- Transforms
- Statistical signal processing
- Parametric modeling
- Linear prediction
- Multirate signal processing
- Waveform generation

A suite of interactive graphical user interfaces are available for

- Filter design and analysis
- Window design and analysis
- Signal plotting and analysis, spectral analysis, and filtering

Note The Signal Processing Toolbox supports only double-precision inputs. If you input single-precision floating-point or integer data, you should not expect to receive correct results, and in many cases, an error will occur. The Filter Design Toolbox, in conjunction with the Fixed-Point Toolbox, enables single-precision floating-point and fixed-point support for filtering and filter design.

Central Features

The Signal Processing Toolbox functions are algorithms, expressed mostly in M-files, that implement a variety of signal processing tasks. These toolbox functions are a specialized extension of the MATLAB computational and graphical environment.

Signals and Systems

The basic entities that toolbox functions work with are signals and systems. The functions emphasize digital (or discrete) signals and filters, as opposed to analog (or continuous) signals. The principal filter type the toolbox supports is the linear, time-invariant digital filter with a single input and a single output. You can represent linear time-invariant systems using one of several models (such as transfer function, state-space, zero-pole-gain, and second-order section), and you can convert between representations.

Filter Design, Analysis, and Implementation

The Signal Processing Toolbox provides customizable support for filter design. The major filter design functions included in the toolbox are FIR and IIR filter design, analysis, and implementation, filter order estimation, and analog filter prototyping and transformations.

Linear System Transformations

The toolbox has a number of transformation functions, including conversions to and from second-order sections, state-space, pole-zero, lattice or ladder, and transfer functions.

Windows

The toolbox provides many commonly used window functions as well as graphical user interfaces to view and compare windows and design filters using these windows.

Spectral Analysis

Toolbox functions are available for estimating the power spectral density, mean-square spectral estimate, and pseudo spectrum, using parametric and

nonparametric techniques. Some of the spectral analysis methods included in the toolbox are Burg, covariance, eigenvector, Thomson multitaper, periodogram, Welch, and Yule-Walker. Other functions are available for computing the average power of a power spectral density, computing a one-sided spectrum, and shifting the DC component to the center of a spectrum.

Transforms

The toolbox includes a variety of transforms and inverse transforms, including Fourier, chirp-Z, discrete cosine, Goertzel, and Hilbert.

Statistical Signal Processing

The toolbox has functions for computing correlation, cross-correlation, covariance, and autocorrelation.

Parametric Modeling

The toolbox includes these methods for autoregressive parametric modeling: Burg, covariance, Yule-Walker, and Steiglitz-McBride (for ARMA modeling). The toolbox also has functions for fitting a frequency response to an analog or discrete-time filter.

Linear Prediction

The toolbox has functions for computing linear prediction coefficients and for converting between autocorrelations and prediction polynomials, reflection coefficients, and line spectral frequencies.

Multirate Signal Processing

The toolbox includes a number of functions for multirate signal processing, including decimation, up- and downsampling, resampling, and spline interpolation.

Waveform Generation

The toolbox has functions to generate many types of periodic and aperiodic waveforms, including chirp, Dirichlet function, Gaussian RF pulse, Gaussian

monopulse, pulse train, rectangle, sawtooth, sinc, square wave, triangle, and voltage-controlled oscillator. See “Waveform Generation: Time Vectors and Sinusoids” on page 2-4 for more information.

Other Operations

A number of other operations are also available in the toolbox. Some of these are cepstral analysis, modulation, demodulation, Slepian sequences, and various plotting methods.

Interactive Tools

The power of the Signal Processing Toolbox is greatly enhanced by its easy-to-use interactive tools.

- The Filter Design and Analysis Tool (`fdatool`) provides a more comprehensive collection of features for addressing filter design. `FDATool` offers seamless access to the additional filter design methods, quantization features, C-code generation and other enhanced filtering features of the Filter Design Toolbox when that product is installed. If you have the Filter Design HDL Coder installed, you can also generate HDL code from `FDATool`.
- The Filter Visualization Tool (`fvtool`) provides a graphical environment for viewing, annotating, and printing filter response plots.
- The Window Design and Analysis Tool (`wintool`) provides an environment for designing and comparing spectral windows.
- The Window Visualization Tool (`wvtool`) provides a graphical environment for viewing, annotating, and printing window plots.
- The Signal Processing Tool (`sptool`) provides a rich graphical environment for signal viewing, filter design, and spectral analysis.

Extensibility

One of the most important features of the MATLAB environment is that it is extensible. MATLAB lets you create your own M-files to meet numeric computation needs for research, design, or engineering of signal processing systems. Simply copy the M-files provided with the Signal Processing Toolbox and modify them as needed, or create new functions to expand the functionality of the toolbox.

A number of other MATLAB toolboxes expand and enhance the Signal Processing Toolbox. These include:

- Filter Design Toolbox — toolbox for advanced filter design
- Signal Processing Blockset — product based on Simulink® for creating and analyzing signal processing models
- Fixed Point Toolbox — toolbox for using fixed-point arithmetic
- Filter Design HDL Coder — toolbox for creating and exporting HDL code
- Link for Code Composer Studio™ — toolbox for working with embedded software on Texas Instruments DSPs

Finding More Information

This Getting Started guide provides an introduction to the Signal Toolbox and examples, which give you a quick start at using some of the commands and graphical user interfaces. It is assumed that you have basic knowledge and understanding of signals and systems, including such topics as filter and linear system theory and basic Fourier analysis.

More detailed and advanced information on using the Signal Processing Toolbox is available in the online help system by typing `doc` at the MATLAB command line or by viewing the documentation on the MathWorks Web site (www.mathworks.com). The toolbox also includes a number of introductory and advanced demos which you can access by typing `demos` at the MATLAB command line.

Basic Signal Processing Concepts

Representing Signals (p. 2-2)	Vector and matrix representation of signals
Waveform Generation: Time Vectors and Sinusoids (p. 2-4)	Periodic and aperiodic waveforms, sequences (impulse, step, ramp), multichannel signals, pulse trains, sinc and Dirichlet functions
Working with Data (p. 2-11)	Methods of inputting and importing data
Selected Bibliography (p. 2-12)	Sources for additional information

Representing Signals

The central data construct in MATLAB is the *numeric array*, an ordered collection of real or complex numeric data with two or more dimensions. The basic data objects of signal processing (one-dimensional signals or sequences, multichannel signals, and two-dimensional signals) are all naturally suited to array representation.

Vector Representation

MATLAB represents ordinary one-dimensional sampled data signals, or sequences, as *vectors*. Vectors are 1-by- n or n -by-1 arrays, where n is the number of samples in the sequence. One way to introduce a sequence into MATLAB is to enter it as a list of elements at the command prompt. The statement

```
x = [4 3 7 -9 1]
```

creates a simple five-element real sequence in a row vector. Transposition turns the sequence into a column vector

```
x = x'
```

resulting in

```
x =  
 4  
 3  
 7  
-9  
 1
```

Column orientation is preferable for single channel signals because it extends naturally to the multichannel case. For multichannel data, each column of a matrix represents one channel. Each row of such a matrix then corresponds to a sample point. A three-channel signal that consists of x , $2x$, and x/π is

```
y = [x 2*x x/pi]
```

This results in

y =

4.0000	8.0000	1.2732
3.0000	6.0000	0.9549
7.0000	14.0000	2.2282
-9.0000	-18.0000	-2.8648
1.0000	2.0000	0.3183

Waveform Generation: Time Vectors and Sinusoids

A variety of toolbox functions generate waveforms. Most require you to begin with a vector representing a time base. Consider generating data with a 1000 Hz sample frequency, for example. An appropriate time vector is

```
t = (0:0.001:1)';
```

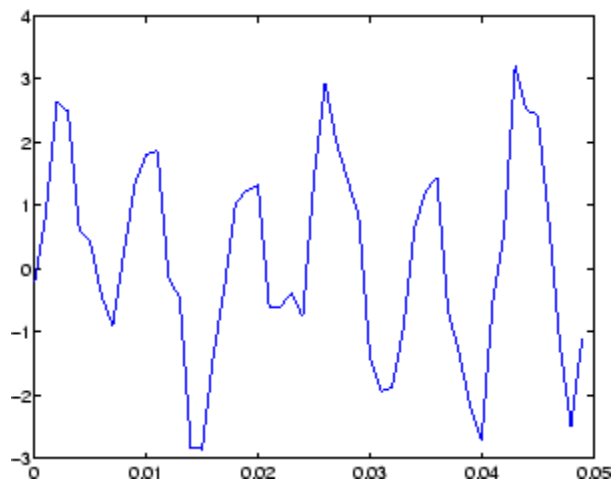
where the MATLAB colon operator creates a 1001-element row vector that represents time running from 0 to 1 s in steps of 1 ms. The transpose operator (') changes the row vector into a column; the semicolon (;) tells MATLAB to compute, but not display the result.

Given t , you can create a sample signal y consisting of two sinusoids, one at 50 Hz and one at 120 Hz with twice the amplitude.

```
y = sin(2*pi*50*t) + 2*sin(2*pi*120*t);
```

The new variable y , formed from vector t , is also 1001 elements long. You can add normally distributed white noise to the signal and plot the first 50 points using

```
randn('state',0);  
yn = y + 0.5*randn(size(t));  
plot(t(1:50),yn(1:50))
```



Common Sequences: Unit Impulse, Unit Step, and Unit Ramp

Since MATLAB is a programming language, an endless variety of different signals is possible. Here are some statements that generate several commonly used sequences, including the unit impulse, unit step, and unit ramp functions:

```
t = (0:0.001:1)';
y = [1; zeros(99,1)]; % impulse
y = ones(100,1);      % step (filter assumes 0 initial cond.)
y = t;                % ramp
y = t.^2;
y = square(4*t);
```

All of these sequences are column vectors. The last three inherit their shapes from `t`.

Multichannel Signals

Use standard MATLAB array syntax to work with multichannel signals. For example, a multichannel signal consisting of the last three signals generated above is

```
z = [t t.^2 square(4*t)];
```

You can generate a multichannel unit sample function using the outer product operator. For example, a six-element column vector whose first element is one, and whose remaining five elements are zeros, is

```
a = [1 zeros(1,5)]';
```

To duplicate column vector `a` into a matrix without performing any multiplication, use the MATLAB colon operator and the ones function:

```
c = a(:,ones(1,3));
```

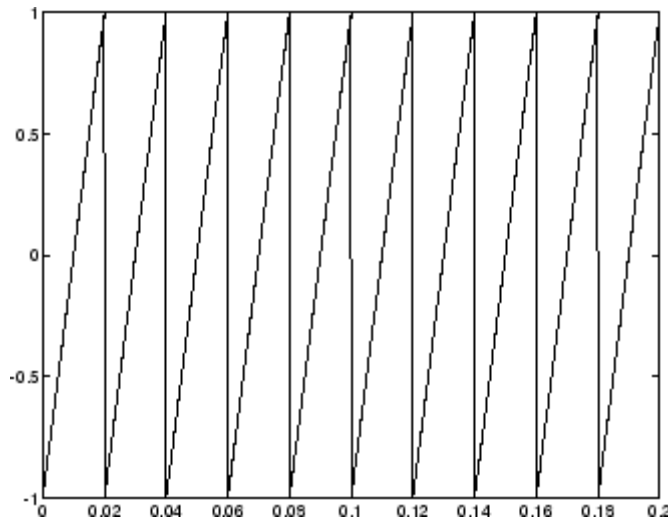
Common Periodic Waveforms

The toolbox provides functions for generating widely used periodic waveforms:

- `sawtooth` generates a sawtooth wave with peaks at ± 1 and a period of 2π . An optional width parameter specifies a fractional multiple of 2π at which the signal's maximum occurs.
- `square` generates a square wave with a period of 2π . An optional parameter specifies *duty cycle*, the percent of the period for which the signal is positive.

To generate 1.5 s of a 50 Hz sawtooth wave with a sample rate of 10 kHz and plot 0.2 s of the generated waveform, use

```
fs = 10000;  
t = 0:1/fs:1.5;  
x = sawtooth(2*pi*50*t);  
plot(t,x), axis([0 0.2 -1 1])
```



Common Aperiodic Waveforms

The toolbox also provides functions for generating several widely used aperiodic waveforms:

- `gauspuls` generates a Gaussian-modulated sinusoidal pulse with a specified time, center frequency, and fractional bandwidth. Optional parameters return in-phase and quadrature pulses, the RF signal envelope, and the cutoff time for the trailing pulse envelope.

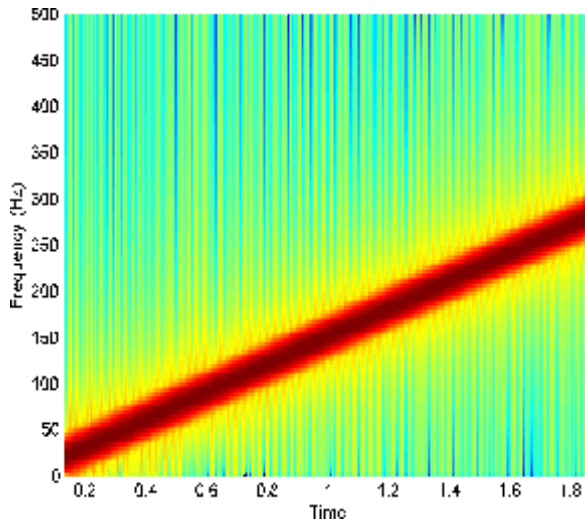
- `chirp` generates a linear, log, or quadratic swept-frequency cosine signal. An optional parameter specifies alternative sweep methods. An optional parameter `phi` allows initial phase to be specified in degrees.

To compute 2 s of a linear chirp signal with a sample rate of 1 kHz, that starts at DC and crosses 150 Hz at 1 s, use

```
t = 0:1/1000:2;
y = chirp(t,0,1,150);
```

To plot the spectrogram, use

```
spectrogram(y,256,250,256,1000,'yaxis')
```

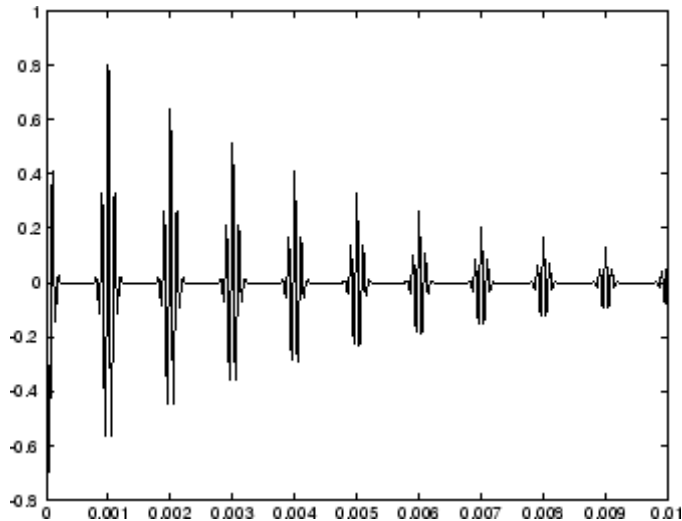


The pulstran Function

The `pulstran` function generates pulse trains from either continuous or sampled prototype pulses. The following example generates a pulse train consisting of the sum of multiple delayed interpolations of a Gaussian pulse. The pulse train is defined to have a sample rate of 50 kHz, a pulse train length of 10 ms, and a pulse repetition rate of 1 kHz; `D` specifies the delay to each pulse repetition in column 1 and an optional attenuation for each repetition in column 2. The pulse train is constructed by passing the name of

the `gauspuls` function to `pulstran`, along with additional parameters that specify a 10 kHz Gaussian pulse with 50% bandwidth:

```
T = 0:1/50E3:10E-3;  
D = [0:1/1E3:10E-3;0.8.^(0:10)]';  
Y = pulstran(T,D,'gauspuls',10E3,0.5);  
plot(T,Y)
```



The Sinc Function

The `sinc` function computes the mathematical sinc function for an input vector or matrix x . The sinc function is the continuous inverse Fourier transform of the rectangular pulse of width 2π and height 1.

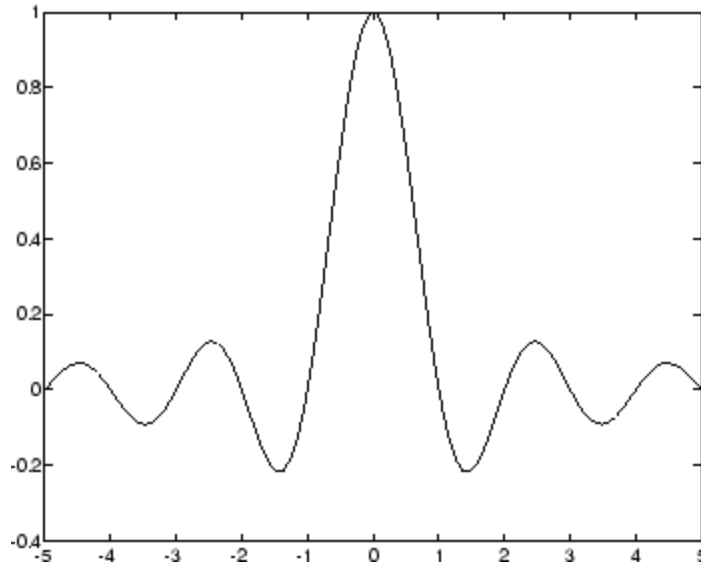
The sinc function has a value of 1 where x is zero, and a value of

$$\frac{\sin(\pi x)}{\pi x}$$

for all other elements of x .

To plot the sinc function for a linearly spaced vector with values ranging from -5 to 5, use the following commands:


```
x = linspace(-5,5);
y = sinc(x);
plot(x,y)
```



The Dirichlet Function

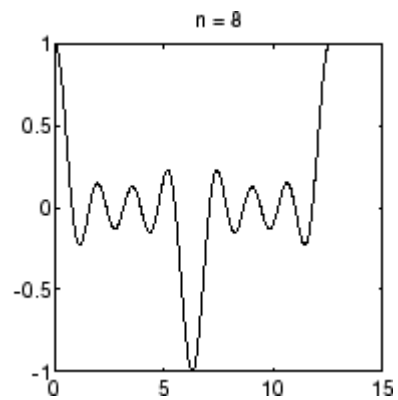
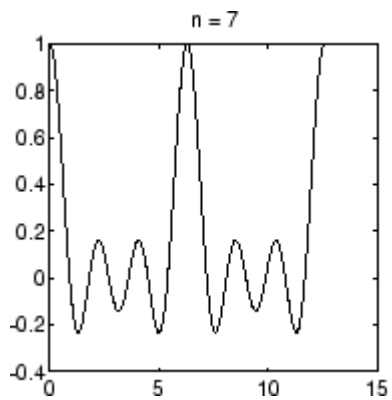
The toolbox function `diric` computes the Dirichlet function, sometimes called the *periodic sinc* or *aliased sinc* function, for an input vector or matrix x . The Dirichlet function is

$$\text{diric}(x) = \begin{cases} -1^{k(n-1)} & x = 2\pi k, k = 0, \pm 1, \pm 2, \dots \\ \frac{\sin(nx/2)}{n \sin(x/2)} & \text{otherwise} \end{cases}$$

where n is a user-specified positive integer. For n odd, the Dirichlet function has a period of 2π ; for n even, its period is 4π . The magnitude of this function is $(1/n)$ times the magnitude of the discrete-time Fourier transform of the n -point rectangular window.

To plot the Dirichlet function over the range 0 to 4π for $n = 7$ and $n = 8$, use

```
x = linspace(0,4*pi,300);  
plot(x,diric(x,7)); axis tight;  
plot(x,diric(x,8)); axis tight;
```



Working with Data

The examples in the preceding sections obtain data in one of two ways:

- By direct input, that is, entering the data manually at the keyboard
- By using a MATLAB or toolbox function, such as `sin`, `cos`, `sawtooth`, `square`, or `sinc`

Some applications, however, may need to import data from outside MATLAB. Depending on your data format, you can do this in the following ways:

- Load data from an ASCII file or MAT-file with the MATLAB `load` command.
- Read the data into MATLAB with a low-level file I/O function, such as `fopen`, `fread`, and `fscanf`.
- Develop a MEX-file to read the data.

Other resources are also useful, such as a high-level language program (in Fortran or C, for example) that converts your data into MAT-file format. See the “MATLAB External Interfaces” documentation for details. MATLAB reads such files using the `load` command.

Similar techniques are available for exporting data generated within MATLAB. See the “Importing and Exporting Data” documentation for more details.

Note All Signal Processing Toolbox functions accept double-precision inputs. If you input single-precision floating-point or integer data types, you should not expect to receive correct results and in many cases, an error will occur. The Filter Design Toolbox, along with the Fixed-Point Toolbox, enables single-precision floating-point and fixed-point support for most `dfilt` structures.

Selected Bibliography

Algorithm development for the Signal Processing Toolbox has drawn heavily upon the references listed below. All are recommended to the interested reader who needs to know more about signal processing than is covered in this manual.

[1] Crochiere, R.E., and L.R. Rabiner. *Multi-Rate Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1983. pp. 88-91.

[2] IEEE. *Programs for Digital Signal Processing*. IEEE Press. New York: John Wiley & Sons, 1979.

[3] Jackson, L.B. *Digital Filters and Signal Processing*. Third Ed. Boston: Kluwer Academic Publishers, 1989.

[4] Kay, S.M. *Modern Spectral Estimation*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[5] Oppenheim, A.V., and R.W. Schaffer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[6] Parks, T.W., and C.S. Burrus. *Digital Filter Design*. New York: John Wiley & Sons, 1987.

[7] Percival, D.B., and A.T. Walden. *Spectral Analysis for Physical Applications: Multitaper and Conventional Univariate Techniques*. Cambridge: Cambridge University Press, 1993.

[8] Pratt, W.K. *Digital Image Processing*. New York: John Wiley & Sons, 1991.

[9] Proakis, J.G., and D.G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*. Upper Saddle River, NJ: Prentice Hall, 1996.

[10] Rabiner, L.R., and B. Gold. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1975.

[11] Welch, P.D. "The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified

Periodograms." *IEEE Trans. Audio Electroacoust.* Vol. AU-15 (June 1967).
Pgs. 70-73.

Filter Design with the FDATool GUI

Introduction (p. 3-2)	Overview of filtering using Signal Processing Toolbox GUIs
Designing the Filter (p. 3-3)	Details of designing a filter using FDATool
Analyzing the Filter (p. 3-8)	Viewing different filter responses in FDATool
Designing Additional Filters (p. 3-10)	Creating more filters in FDATool
Viewing and Annotating the Filter (p. 3-11)	Using the FVTool GUI to view and annotate the filter
Exporting Filters from FDATool (p. 3-17)	Exporting the filter designs as objects
Using Discrete Filter Objects (dfilts) (p. 3-19)	Using discrete filter objects
Where to Find More Information (p. 3-22)	Finding more information and more complex examples

Introduction

This section describes how to graphically design and implement digital filters using the Signal Processing Toolbox. Filter design is the process of creating the filter coefficients to meet specific frequency specifications. Filter implementation involves choosing and applying a particular filter structure to those coefficients. Only after both design and implementation have been performed can your data be filtered.

Although many methods exist for designing the filter coefficients, this chapter focuses on using the basic features of the Filter Design and Analysis Tool (FDATool) GUI. For filter implementation, this chapter uses `dfilt`.

This section includes a brief discussion of applying the completed filter design and filter implementation using MATLAB command line functions, such as `filter`.

For an interactive FDATool demo, type `demos` at the MATLAB command line, and select **Toolboxes**. Expand the tree, scroll down, and select **Signal Processing Toolbox**. Under Filter Design and Analysis, click **Introduction to Filter Design and Analysis Tool**.

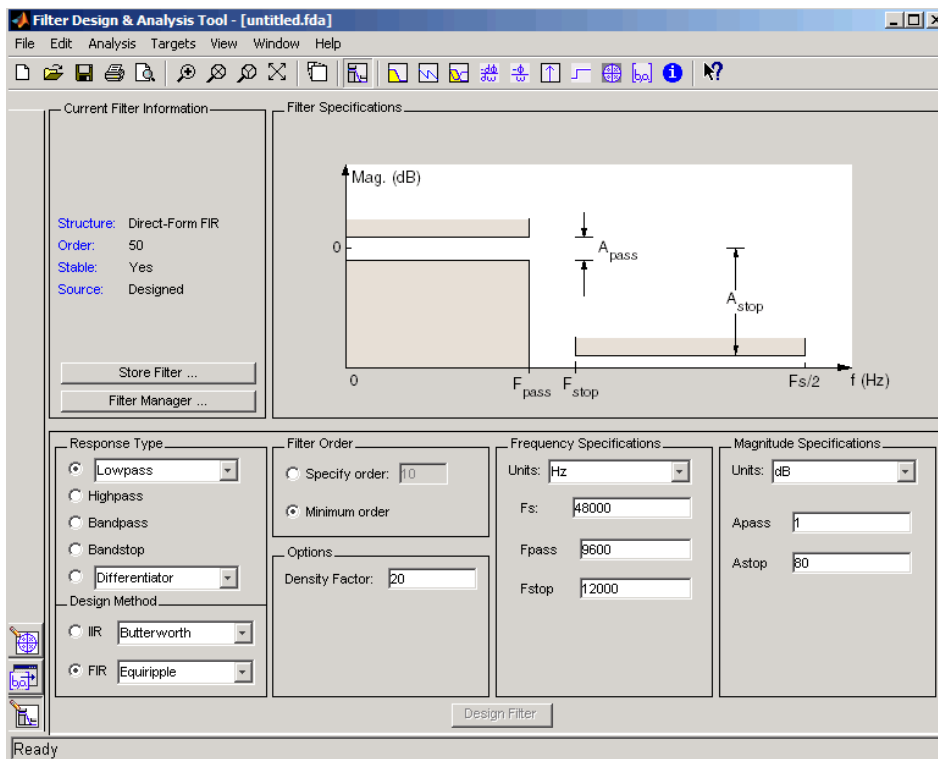
Designing the Filter

This section is a step-by-step introduction to using the Filter Design and Analysis Tool (FDATool) to design an octave-band filter. An octave is the interval between two frequencies having a ratio of 2:1. An octave-band filter is a bandpass filter with high cutoff frequency approximately twice that of the low cutoff frequency. The class of an octave filter is determined by its allowable passband ripple and its stopband attenuation. Refer to the ANSI S1.11–2004 standard for more information. For more information on designing filters, see “FDATool: A Filter Design and Analysis GUI” in the Signal Processing Toolbox User’s Guide. (Note that you can also access FDATool from SPTool).

- 1 Start FDATool from the MATLAB command line.

```
fdatool
```

The FDATool dialog opens with a default filter. Its filter information is summarized in the upper left (**Current Filter Information**) and its filter specifications are depicted in the upper right. In addition to displaying filter specification, this upper right pane displays filter responses and filter coefficients.



The bottom half of FDATool shows the Filter Design panel, where you specify the filter parameters. Other panels, such as Import filter from workspace and Pole/Zero Editor, which you access with the buttons on the lower left, are also displayed in this area. If you have other products installed, you may see additional buttons.

- 2 In the **Response Type** pane, select **Bandpass**.
- 3 In the **Design Method** pane, select **IIR**, and then select Butterworth from the selection list.

Response Type

Lowpass

Highpass

Bandpass

Bandstop

Differentiator

Design Method

IIR Butterworth

FIR Equiripple

- 4 For the Filter Order, select **Specify order**, and then enter 6.

Filter Order

Specify order: 6

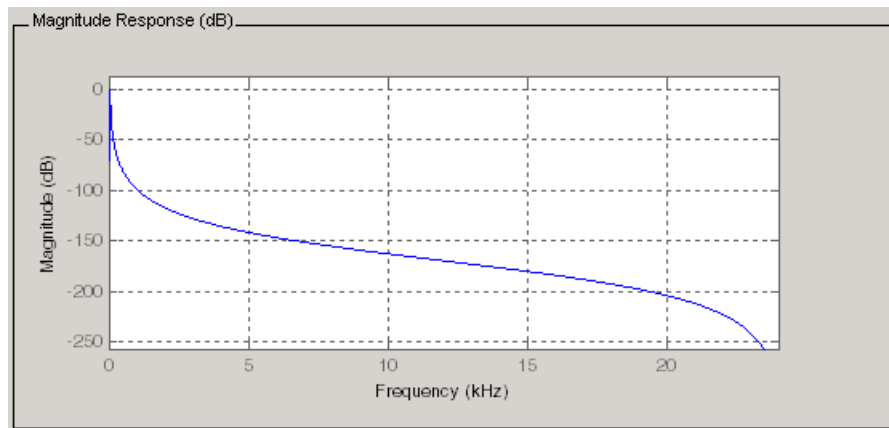
Minimum order

- 5 Set the Frequency Specifications as follows:

Parameter	Setting	Description
Units	Hz	Units for the parameters
F_s	48000	Sampling frequency
F_{c1}	22	First cutoff frequency (i.e., the frequency preceding the passband at which the magnitude response is 3 dB below the passband gain)
F_{c2}	45	Second cutoff frequency (i.e., the frequency following the passband at which the magnitude response is 3 dB below the passband gain)

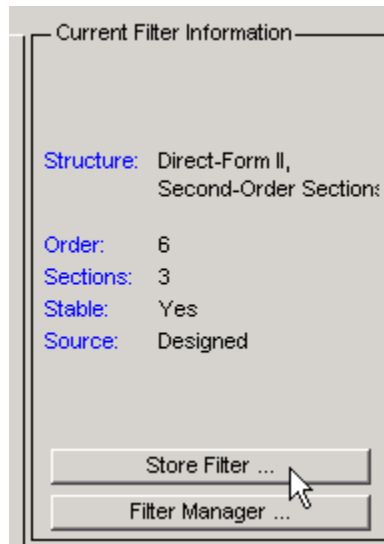
Frequency Specifications	Magnitude Specifications
Units: <input type="text" value="Hz"/>	
Fs: <input type="text" value="48000"/>	
Fc1: <input type="text" value="22"/>	The attenuation at cutoff frequencies is fixed at 3 dB (half the passband gain)
Fc2: <input type="text" value="45"/>	

- 6 After specifying the filter design parameters, click the **Design Filter** button at the bottom of the design panel to compute the filter coefficients. The display updates to show the magnitude response of the designed filter.

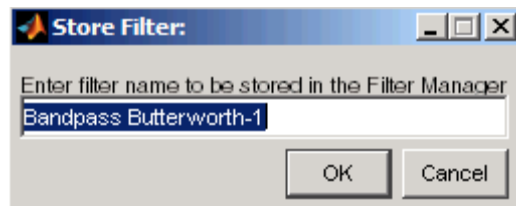


Notice that the **Design Filter** button is disabled after you compute the coefficients for your filter design. This button is enabled again if you make any changes to the filter specifications.

- 7 Click the **Store Filter** button.













- 8 In the Store Filter dialog, change the filter name to Bandpass Butterworth-1 and click **OK** to save the filter in the Filter Manager.



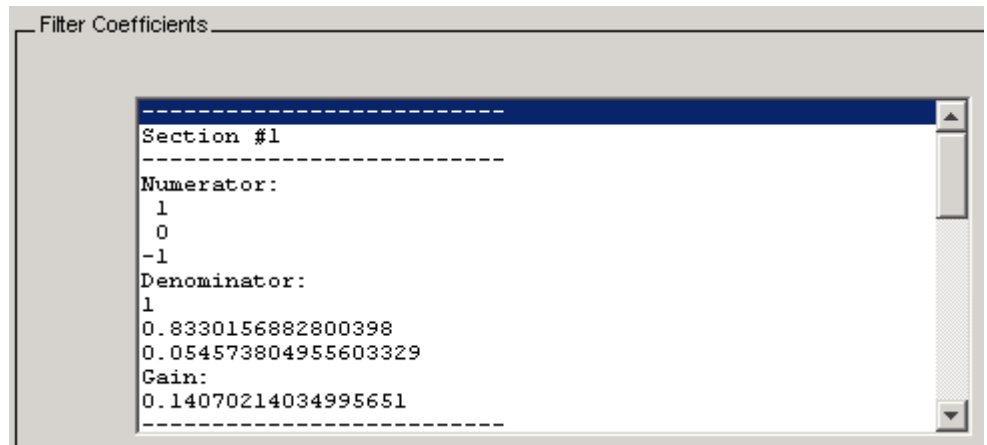
Analyzing the Filter

After designing the filter, you can view the following filter responses in the display region by clicking on the associated toolbar button or by selecting the desired response from the **Analysis** menu.

Response	Toolbar Button Image
Magnitude response	
Phase response	
Magnitude and Phase responses	
Group delay	
Phase delay	
Impulse response	
Step response	
Pole-zero plot	
Filter coefficients	
Filter information	
<p>Zero-phase response—only available from a context menu. Right-click on the y-axis of a Magnitude or Magnitude and Phase response plot or select Analysis > Analysis Parameters.</p>	

Note Other analyses are available if you have the Filter Design Toolbox installed.

- 1 Examine the displayed magnitude response of the filter.
- 2 Display other responses, as desired. Click the appropriate buttons, shown in the table above or select the desired response from the **Analysis** menu.
- 3 Click the **Filter coefficients** button to display the filter coefficients.



```
Filter Coefficients
-----
Section #1
-----
Numerator:
  1
  0
 -1
Denominator:
  1
 0.8330156882800398
 0.054573804955603329
Gain:
 0.14070214034995651
-----
```

Designing Additional Filters

You have designed one of the bands of an octave filter bank. This section shows you how to design and save the other nine bands. The following table defines the parameters for the remaining bands. Note that all of the bands use these parameters: **Bandpass, IIR – Butterworth**, **order = 6, Fs = 48000 Hz**.

Fc1	Fc2	Filter Name
45	89	Bandpass Butterworth-2
89	178	Bandpass Butterworth-3
178	355	Bandpass Butterworth-4
355	708	Bandpass Butterworth-5
708	1413	Bandpass Butterworth-6
1413	2818	Bandpass Butterworth-7
2818	5623	Bandpass Butterworth-8
5623	11220	Bandpass Butterworth-9
11220	22387	Bandpass Butterworth-10

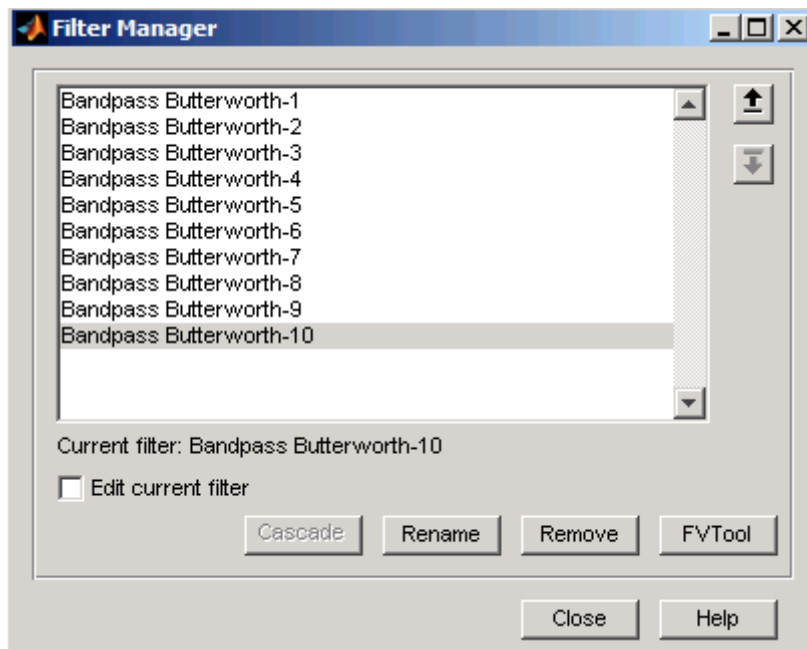
- 1 Using the parameters listed in the table above, for each table row, set the appropriate the **Fc1** and **Fc2** values.
- 2 Design the filter by clicking the **Design Filter** button.
- 3 Click **Store Filter** to save the filter.
- 4 Change the name to the appropriate filter name shown in the table above.
- 5 Repeat these steps until all 10 filters are designed and stored.

Viewing and Annotating the Filter

Viewing the Filter in FVTool

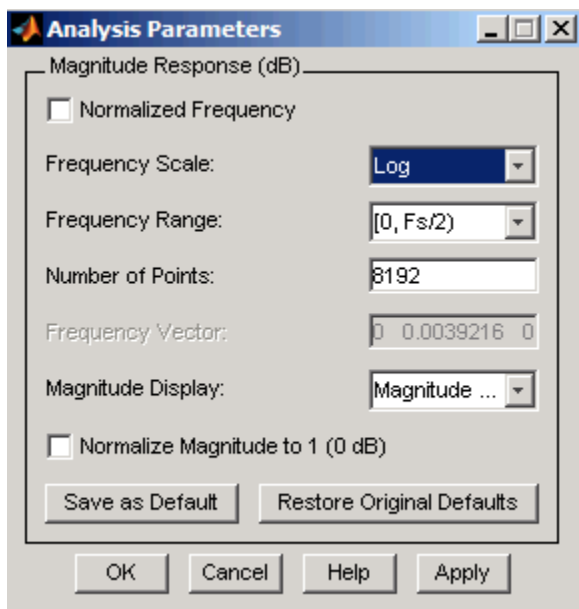
This section teaches you how to use the Filter Visualization Tool (FVTool) to view the octave-band filter. It also describes how to annotate your filter.

- 1 Click the **Filter Manager** button to display the Filter Manager, which lists your saved filters.

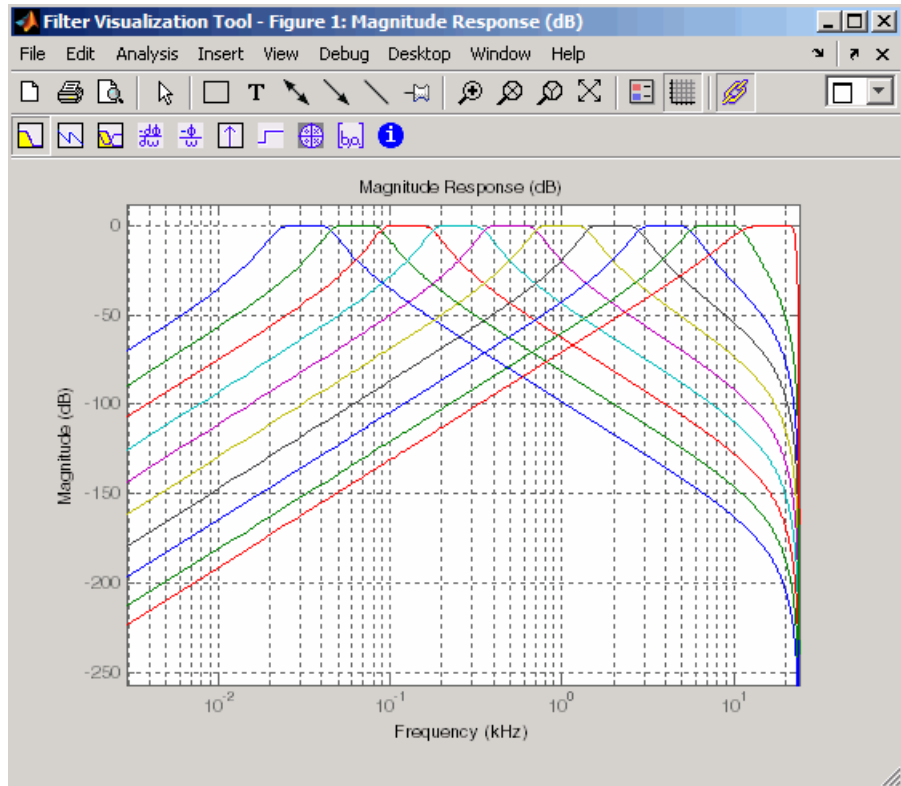



- 2 Press Ctrl+click on each filter name to select all the filters, and then click **FVTool**. FVTool opens with the filter responses overlaid for easy comparison. (If you want to view a single filter in FVTool, click the **Full View Analysis** button when that filter is shown in the FDATool display panel or select **View > Filter Visualization Tool**).
- 3 Change the x-axis scale to logarithmic by selecting **Analysis > Analysis Parameters** to display the Analysis Parameters dialog.

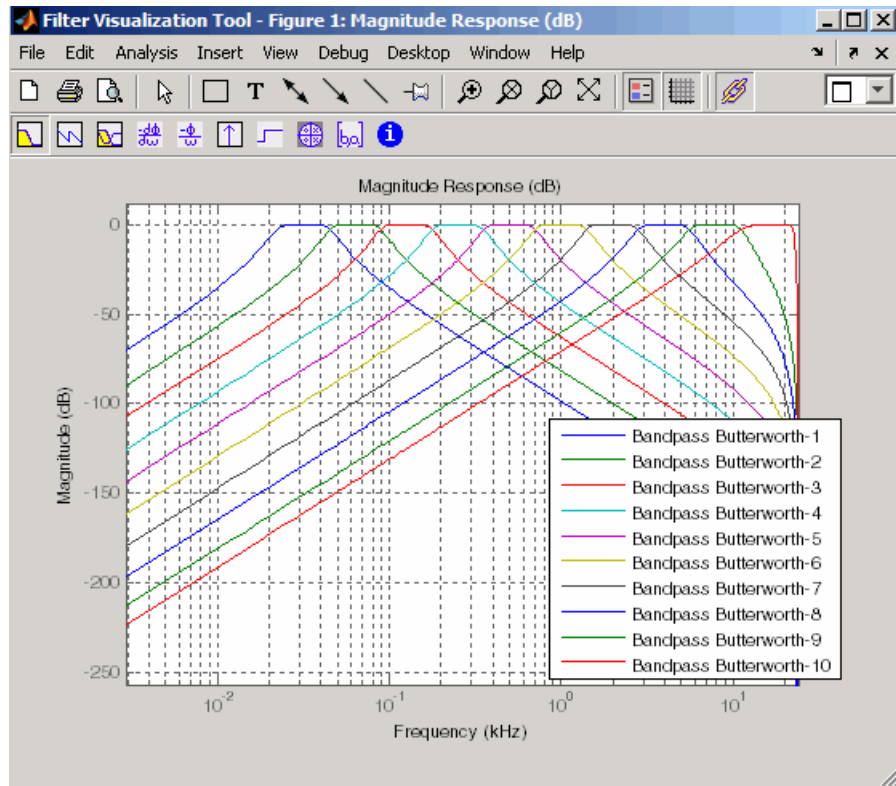
4 Change the **Frequency Scale** to Log.




5 Click **OK**.

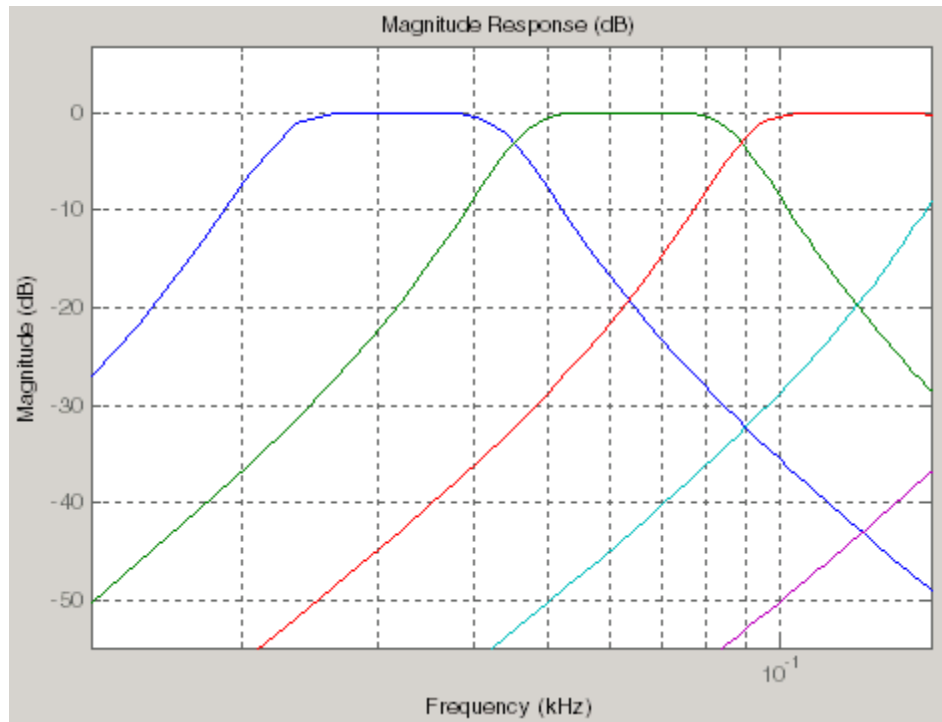



- 6 Click the **Legend** button  to turn on the legend, which you can drag to the desired location.



7 Click the **Legend** button again to turn off the legend.

Use the **Zoom** button  and drag a rectangle around the first few passbands to zoom in.



8 Click the **Restore Default View** button  to return to the full view.

9 Display other responses, as desired. (The FVTool Analysis toolbar buttons and **Analysis** menu are the same as in FDATool. See “Analyzing the Filter” on page 3-8 for descriptions of the buttons.)

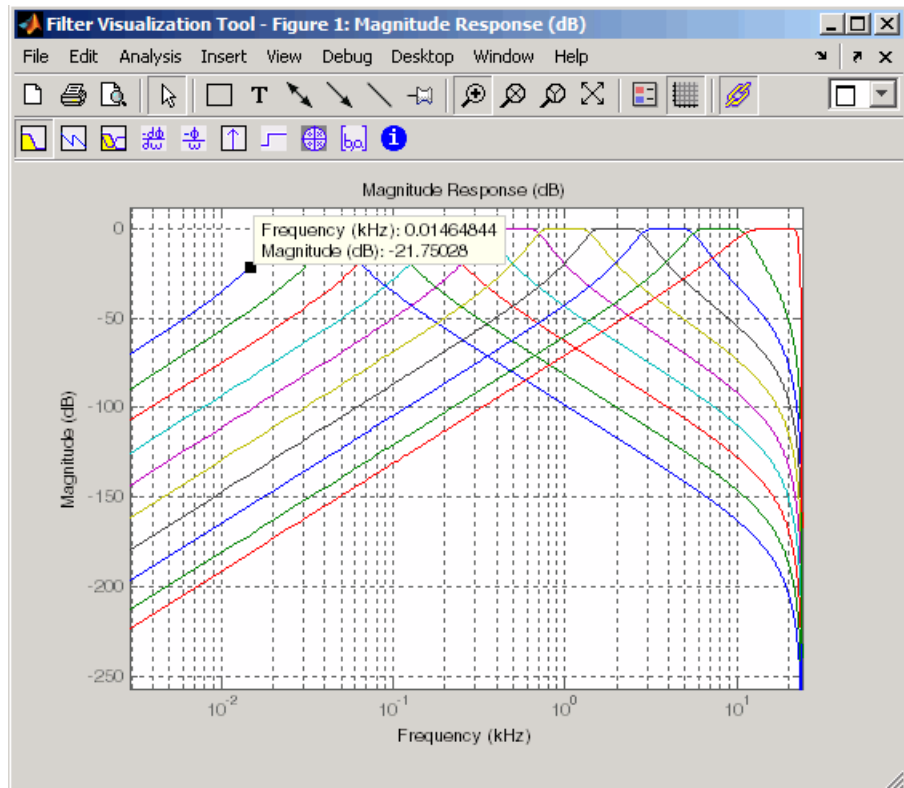
Using FVTool for Annotation

FVTool is also useful for doing further analysis, adding annotations, and printing. Available annotations include adding rectangles, text boxes, arrows and lines, and adding data markers.

For a demo about FVTool, type demos at the MATLAB command line, and select **Toolboxes**. Expand the tree, scroll down, and select **Signal Processing Toolbox**. Under Filter Design and Analysis, click **Filter Analysis using FVTool and its API**.

Note Do not close FDATool at this time. You will use it in future sections.

- 1 Use the toolbar buttons to annotate your response plot. Add a line by clicking one of the line buttons, and then use your mouse to draw the line on your plot.
- 2 Add a data maker by clicking on a plot at the desired point. The data marker shows the frequency and magnitude at that point.

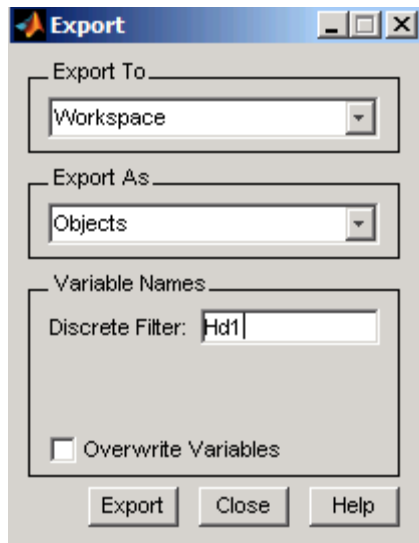


- 3 Close FVTool by selecting **File > Close**.

Exporting Filters from FDATool

FDATool provides a simple way to create filter objects (dfilts) from your filter designs. This is particularly useful for saving your filter design to the MATLAB workspace for use with command line functions. You can also save your filters to M-files using **File > Generate M-file** to run in scripts or batch files.

- 1 In FDATool, click **Filter Manager** and highlight only the Bandpass Butterworth-1 filter.
- 2 Select **File > Export**.
- 3 Set **Export to** to Workspace. Set **Export as** to Objects. In **Discrete Filter** type Hd1. Click **Export** to export the first filter in your filter bank to an Hd1 dfilt object in the workspace.



- 4 Repeat steps 1 through 3 for each of the remaining nine filters. Highlight each filter individually to make it the active filter and change the **Discrete Filter** name to match the filter number. When you finish you will have 10 dfilt objects in the workspace.

5 Close FDATool by selecting **File > Close**.

6 On the MATLAB command line, verify that your objects were exported by using the whos command.

```
whos
      Name      Size      Bytes  Class      Attributes
      Hd1       1x1           8      dfilt.df2sos
      Hd10      1x1           8      dfilt.df2sos
      Hd2       1x1           8      dfilt.df2sos
      Hd3       1x1           8      dfilt.df2sos
      Hd4       1x1           8      dfilt.df2sos
      Hd5       1x1           8      dfilt.df2sos
      Hd6       1x1           8      dfilt.df2sos
      Hd7       1x1           8      dfilt.df2sos
      Hd8       1x1           8      dfilt.df2sos
      Hd9       1x1           8      dfilt.df2sos
```


Using Discrete Filter Objects (dfilts)

In the previous section, you created `dfilt` objects containing your filters. In this section, you explore some `dfilt` command line methods. See the `dfilt` reference page for descriptions of all available methods. `dfilts` are also used in the Filter Design Toolbox and the Filter Design HDL Coder, which extend the capabilities of the Signal Processing Toolbox, and in the Filter D.

For an interactive demo about `dfilts`, type `demof` at the MATLAB command line, and select **Toolboxes**. Expand the tree, scroll down, and select **Signal Processing Toolbox**. Under Filter Design and Analysis, click **Getting Started with Discrete-Time Filter Objects**.

Filtering with `dfilt`

- 1 Type the following on the MATLAB command line to concatenate your filter bank filter objects into a single `dfilt` object.

```
Hd = [Hd1 Hd2 Hd3 Hd4 Hd5 Hd6 Hd7 Hd8 Hd9 Hd10];
```

- 2 To view the first filter, type `Hd(1)`.

```
Hd(1)
```

```
ans =
```

```
FilterStructure: 'Direct-Form II, Second-Order Sections'
  sosMatrix: [3x6 double]
  ScaleValues: [3.40097054256801e-009;1;1;1]
PersistentMemory: false
```

- 3 A number of methods can be used to view and manipulate the `Hd1` `dfilt` object. Try the `info` command:

```
info(Hd1)           % Displays filter information
```

```
Discrete-Time IIR Filter (real)
```

```
-----
Filter Structure      : Direct-Form II, Second-Order Sections
Number of Sections   : 3
Stable                : Yes
Linear Phase         : No
```

- 4** You can open FVTool from the MATLAB command line and specify display parameters as follows.

```
F = fvtool(Hd, 'Analysis', 'magnitude')    % Open FVTool with
                                           % magnitude display
set(F, 'FrequencyScale', 'Log')          % Change to log scale
```

This produces the same display as step 5 of “Viewing the Filter in FVTool” on page 3-11 above.

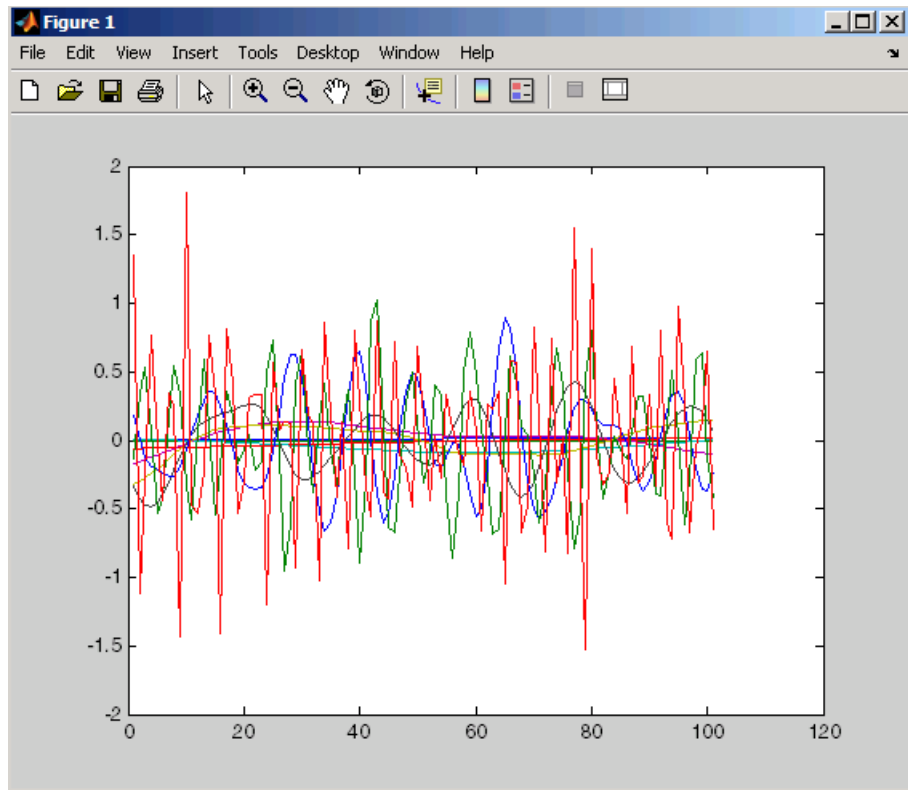
- 5** Now using the MATLAB command line, create some discrete white Gaussian noise data, which you can then filter using the filter bank.

```
rand('state',0);                          % Initialize random number generator
Nx = 100000;                               % Number of noise data points
xw = randn(Nx,1);                          % Create white noise
for i=1:10,
    yw(:,i)=filter(Hd(i),xw); % Filter the white noise through
end                                       % the entire filter bank.
                                           % (:,i) means all rows of column i
```

Note Do not delete this filtered data. You will use it in the Spectral Analysis section.

- 6** Plot the filtered data.

```
plot(yw)
```



The next section discusses spectral analysis, where you analyze this data.

Where to Find More Information

The previous sections described how to use the fundamental features of FDATool, FVTool, and the `dfilt` command. For more advanced information and more complex examples, refer to other sections of the Signal Processing Toolbox online help or the Signal Processing Toolbox documentation available on The MathWorks Web site (www.mathworks.com).

Spectral Analysis

Introduction (p. 4-2)

Summary of algorithms and methods available for spectral analysis

Creating a Spectral Analysis Object (p. 4-4)

How to create a new spectral analysis object

Generating a Spectrum (p. 4-6)

Using PSDs, mean-square spectra, and pseudo spectra

Changing Spectral Analysis Object Property Values (p. 4-9)

How to edit a spectral analysis object

Measuring Signal Power (p. 4-11)

Measuring average power of a signal

Where to Find More Information (p. 4-15)

Finding more information and more complex examples

Introduction

This section has two major examples. The first example shows you how to compute and display a mean-square spectrum. The second example shows you how to measure the average power of a signal.

Spectral analysis includes three types of spectral estimators—power spectral density (PSD), mean-square spectrum (MSS) and pseudo spectrum.

- Power spectral density (psd) measures power per unit of frequency and has power/frequency units.
- Mean-square (power) spectrum (msspectrum) measures power at a specific frequency.
- Pseudo spectrum (pseudospectrum) returns a pseudo spectrum that does not have any units.

The Signal Processing Toolbox provides algorithms to compute the above spectral estimators. The following table indicates which algorithms are available to compute each type of estimator and produce a spectrum object. See the `spectrum` reference page for more information.

Spectral Estimator	Algorithms
Power spectral density (psd)	Burg (<code>spectrum.burg</code>), Covariance (<code>spectrum.cov</code>), Modified covariance (<code>spectrum.mcov</code>), Thomson multitaper method (MTM) (<code>spectrum.mtm</code>), Periodogram (<code>spectrum.periodogram</code>), Welch (<code>spectrum.welch</code>), Yule-Walker autoregressive (<code>spectrum.yulear</code>)
Mean-square spectrum (msspectrum)	Periodogram (<code>spectrum.periodogram</code>), Welch (<code>spectrum.welch</code>)
Pseudo spectrum (pseudospectrum)	Eigenvector (<code>spectrum.eigenvector</code>), MUSIC (Multiple Signal Classification) (<code>spectrum.music</code>)

Spectral analysis objects contain property values for the particular algorithm. To calculate a spectrum estimate, you first create an estimator object using one of the algorithms (e.g., `h = spectrum.burg`), and then pass it to one of the spectrum estimator objects with your data (e.g., `Hpsd = psd(h,x)`).

For more information and examples, see the **Getting Started with Spectral Analysis Objects** demo, which you access by typing `demodemos` at the MATLAB command line. Then expand **Toolboxes**, and then expand **Signal Processing**. Next, select **Spectral Analysis and Statistical Signal Processing** and click **Getting Started with Spectral Analysis Objects**.

Creating a Spectral Analysis Object

Continuing the example in Chapter 3, “Filter Design with the FDATool GUI”, you will create a spectral analysis object that uses default property values, and then, use it to generate a mean-square spectrum of the data filtered through the octave-band filter bank.

Create a Welch’s method spectrum object using default values by typing the following at the MATLAB command line.

```
h = spectrum.welch

h =
  EstimationMethod: 'Welch'
  SegmentLength: 64
  OverlapPercent: 50
  WindowName: 'Hamming'
  SamplingFlag: 'symmetric'
```

If you want to specify parameters instead of using default values, you use syntax similar to the following, which uses a Kaiser window and a segment length of 66. Note that the Kaiser window has an additional parameter (Beta). See the `spectrum.welch` reference page for more information.

```
h = spectrum.welch('kaiser',66,50)

h =
  EstimationMethod: 'Welch'
  SegmentLength: 66
  OverlapPercent: 50
  WindowName: 'Kaiser'
  Beta: 0.5000
```

Note For the rest of this example, use the default `h = spectrum.welch` object.

You can also change some of the property values of a created spectrum object. See “Changing Spectral Analysis Object Property Values” on page 4-9 for more information.

Generating a Spectrum

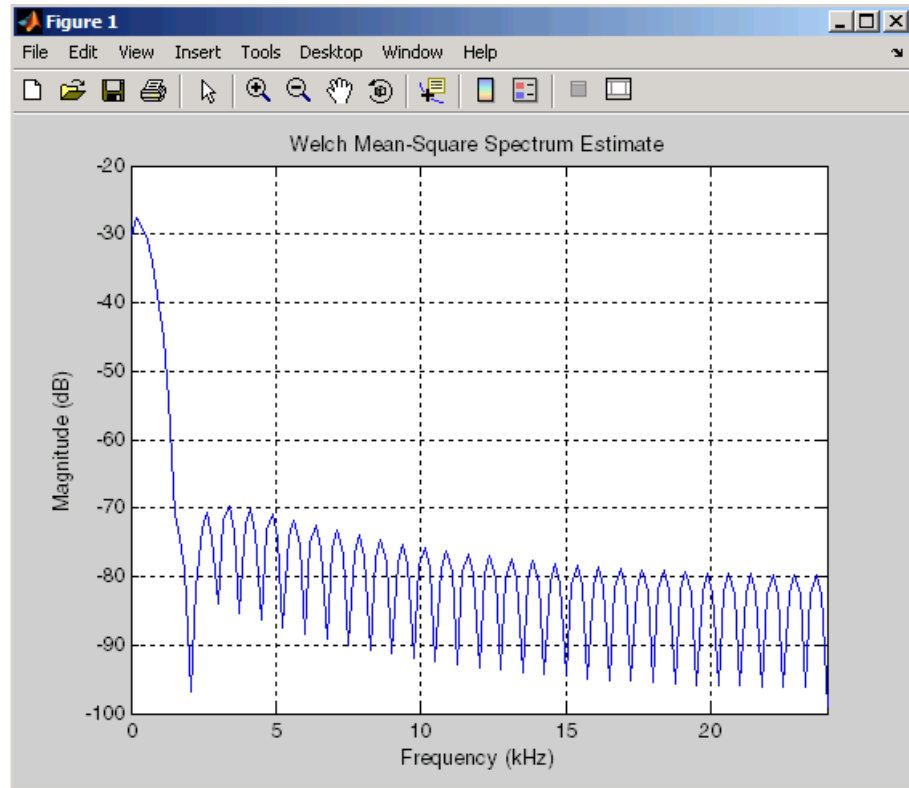
To generate a spectrum, you use a spectral estimation method on your spectrum object and data. To create a mean-square spectrum, type

```
Fs = 48000      % Set the sampling frequency

% Create spectrum object of first filter
msPyyw = msspectrum(h,yw(:,1),'Fs',Fs)

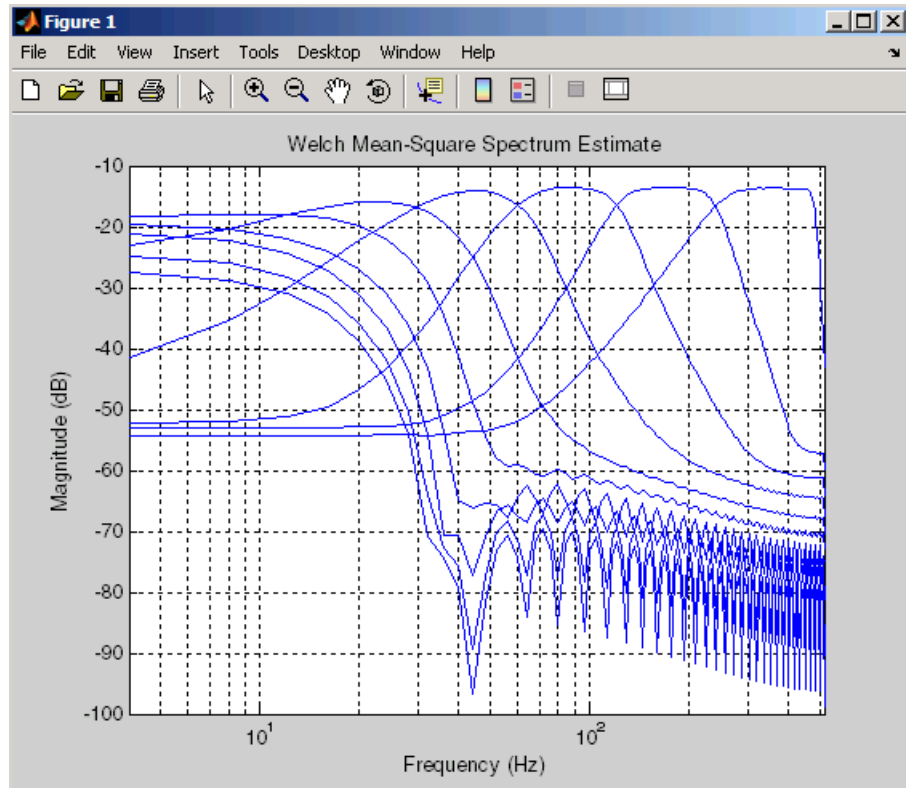
msPyyw =
           Name: 'Mean-Square Spectrum'
           Data: [129x1 double]
      SpectrumType: 'Onesided'
NormalizedFrequency: false
                Fs: 48000
      Frequencies: [129x1 double]

plot(msPyyw)
```



To see the spectra for all the filters, type

```
for i=1:10
    msspectrum(h,yw(:,i),'Fs',Fs);
    hold on; % Plot each spectrum in the same
end % figure window
set(gca,'Xscale','log') % Change to log scale
```



The syntax for using `psd`, `msspectrum`, or `pseudospectrum` is the same. As shown in the example above, the first input is the spectrum object (`h`) and the second input (`yw`) is the signal to measure, followed by any settable properties using property-value pairs (e.g., `'Fs'`, `Fs`). To set property-value pairs, you list the property first and then the value for that property.

Changing Spectral Analysis Object Property Values

There are two ways to change the property values for a spectrum object—using the set command or dot notation and using options objects.

Using the set command to Set Property Values

After you have created a spectrum object, you can use the set method or dot notation to change any of its properties, except its EstimationMethod property. (Since EstimationMethod is the central property of a particular spectrum object, you cannot change it.) To change the window from Hamming (used in the Welch object created above) to Chebyshev, use dot notation as follows:

```
h.WindowName = 'Chebyshev'

h =
  EstimationMethod: 'Welch'
  SegmentLength: 66
  OverlapPercent: 50
  WindowName: 'Chebyshev'
  SidelobeAtten: 100
```

or use the set method,

```
set(h, 'WindowName', 'Chebyshev')
```

As illustrated above, Chebyshev windows have a sidelobe attenuation parameter that automatically appears in the list of properties. To change a window parameter, you must use a cell array containing the window name and parameter value, such as,

```
h = spectrum.welch({'Chebyshev',80})

h =
  EstimationMethod: 'Welch'
  SegmentLength: 64
  OverlapPercent: 50
  WindowName: 'Chebyshev'
  SidelobeAtten: 80
```

To view the amplitude and magnitude responses of various windows, type `wintool` to open the Window Design and Analysis Tool.

Using Options Objects to Set Property Values

Another way to set properties for a particular estimation method is to use the options object associated with that method. For example, use the following syntax to create an options object from the spectrum object for use with the mean-square spectrum:

```
Hopts = msspectrumopts(h)    % Create options object

Hopts =
           NFFT: 'Nextpow2'
  NormalizedFrequency: true
                Fs: 'Normalized'
   SpectrumType: 'Onesided'
         CenterDC: false
```

You can then change any of the options properties using `set`, followed by the options object and property-value pair(s), such as

```
set(Hopts, 'Fs', 48000)
```

To pass the options object to the first `msspectrum` in the filtered data, use

```
msspectrum(h, yw(:, 1), Hopts)
```

Options objects that correspond to `psd` and `pseudospectrum` are `psdopts` and `pseudospectrumopts`, respectively.

Measuring Signal Power

This section shows you how to measure the average power of a deterministic periodic signal. This type of signal is continuous in time, but produces a discrete power spectra. A signal made up of sinusoids is an example of a power signal that has infinite energy, but finite average power . The example below shows how to measure and estimate the average power of a single sinusoidal signal with a peak amplitude of 1 v and a frequency component at 128 Hz.

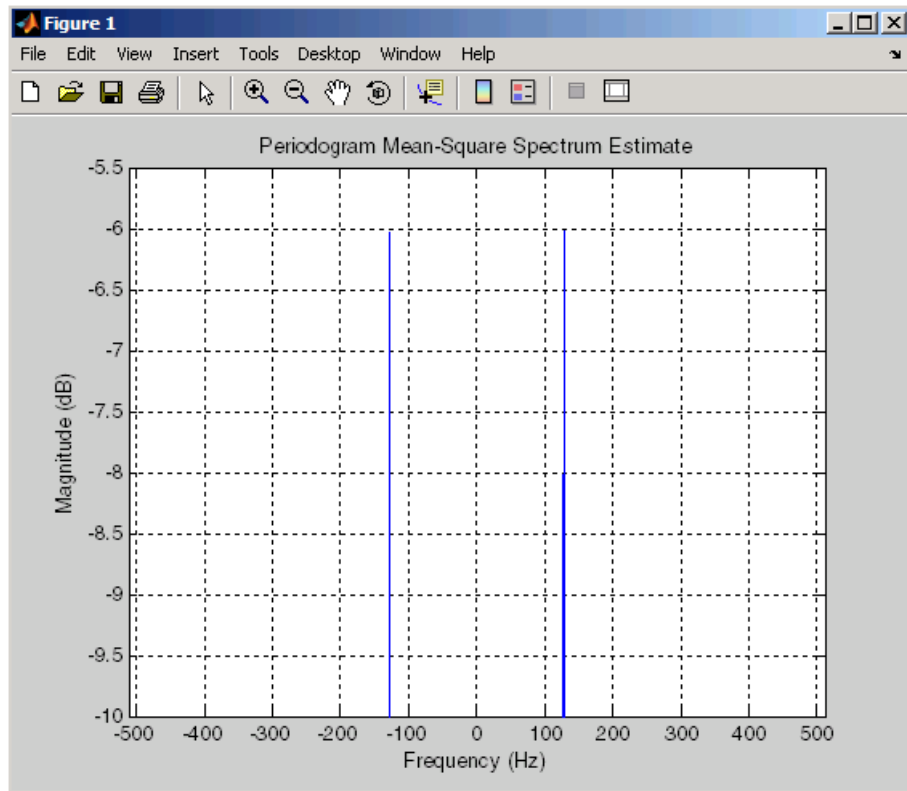
First, you measure the signal's average power using a periodogram spectrum object, and then calculate and plot the mean-square (power) spectrum.

```
Fs = 1024;                % Sampling frequency
t = 0:1/Fs:1-(1/Fs);     % Time vector
A = 1;                   % Peak amplitude
F1 = 128;                 % Hz
x = A*sin(2*pi*t*F1);    % Sinusoidal signal

hp = spectrum.periodogram('hamming'); % Create periodogram

% Create options object and set properties
hpopts = psdopts(hp,x);
set(hpopts,'Fs',Fs,'SpectrumType','twosided','centerdc',true);

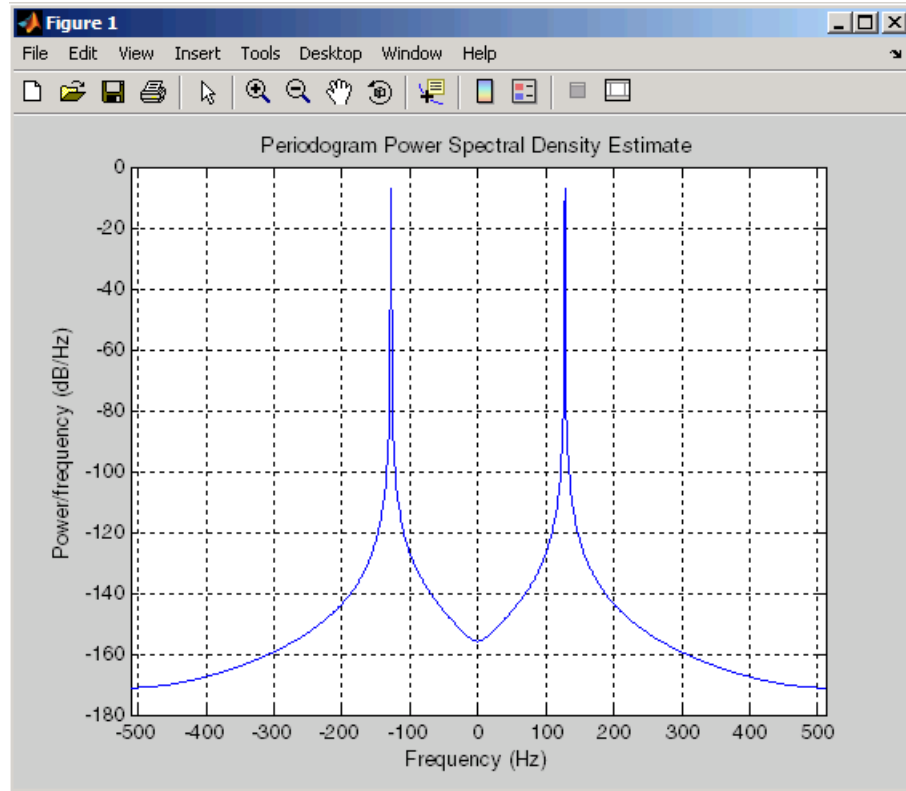
msspectrum(hp,x,hpopts);
v = axis; axis([v(1) v(2) -10 -5.5]); % Zoom in Y.
```



The average power of each complex sinusoid is approximately -6 dB.

Another way to calculate a signal's average power is by "integrating" the area under the power spectral density (PSD) curve. To do this, you use the `psd` method on the spectrum object created above (`hp`) to get a PSD data object, and then use the `avgpower` method.

```
hpsd = psd(hp,x,hopts);  
plot(hpsd);
```

Notice that the peaks of this plot are not the same height as the mean-square spectrum peaks plotted above. The area under the PSD curve is the measure of the average power, not the peak heights. By using the `avgpower` method and converting the result to dB, you can see that the average power is the same for both of them.

```
power_freqdomain = avgpower(hpsd)
```

```
power_freqdomain =
```

```
0.5000
```

According to Parseval's theorem, the total average power in a sinusoid is the same, whether it's computed in the time domain or the frequency domain.

You can verify the signal's estimated total average power by summing the signal in the time domain.

```
power_timedomain = sum(abs(x).^2)/length(x)

power_timedomain =

    0.5000
```

Now, converting this linear value to a logarithmic value (and taking only the positive frequencies), you see that the average power is the same as that shown in the mean-square spectrum plot above.

```
10*log10(power_freqdomain/2)

ans =

    -6.0206
```

Where to Find More Information

The previous sections described how to use the fundamental features of spectral analysis. For more advanced information and more complex examples, refer to other sections of the Signal Processing Toolbox online help or the Signal Processing Toolbox documentation available on The MathWorks Web site (www.mathworks.com).

A

- aliasing
 - sinc functions 2-9
- analog signals, *see* signals
- annotation 3-15
- ASCII files
 - importing 2-11
- average power 4-11

B

- bandpass filters 3-3
- buttons
 - filter analysis 3-8

C

- cell array 4-9
- chirp signals 2-7
- continuous signals, *see* signals
- cycles
 - duty 2-6

D

- data
 - importing 2-11
 - matrices 2-2
 - multichannel matrix 2-2
 - multichannel signals 2-5
 - supported types 1-3
 - time vectors 2-4
 - vectors 2-2
- demos 1-9
 - dfilt 3-19
 - FDATool 3-2
 - FVTool 3-15
 - spectral analysis 4-3
- dfilt 3-19
- digital filters 1-4

- digital signals 1-4
- Dirichlet functions
 - definition 2-9
- dot notation 4-9
- double-precision inputs 1-3
- duty cycles 2-6

E

- export 3-17
- extensibility 1-8

F

- FDATool
 - analysis buttons 3-8
- fdatool GUI
 - analysis buttons 3-8
 - group delay 3-8
 - impulse response 3-8
 - magnitude response 3-8
 - phase delay 3-8
 - phase response 3-8
 - pole-zero plots 3-8
 - step response 3-8
- files
 - M 2-11
 - MAT 2-11
 - MEX 2-11
- filter coefficients 3-8
- filter information 3-8
- filters
 - linear time-invariant digital 1-4
 - octave-band 3-3
- fopen function 2-11
- fread function 2-11
- functions
 - Dirichlet 2-9
 - sinc 2-8

G

gauspuls function
 pulse trains 2-8
graphical user interface (GUI) 1-7
group delay 3-8

I

import
 data 2-11
impulse 2-5
impulse response 3-8
inverse Fourier transforms
 See sinc function 2-8

M

M-files 1-8
magnitude and phase response 3-8
magnitude response 3-8
MAT-files
 converting to 2-11
 importing 2-11
matrices
 data 2-2
matrix 2-2
mean-square spectrum 4-2
MEX-files 2-11
multichannel data 2-5

N

noise data 3-20

O

octave-band filters 3-3
options object 4-10

P

P-V pairs 4-8
periodic sinc functions 2-9
 See also Dirichlet functions
periodogram 4-11
phase delay 3-8
phase response 3-8
pole-zero plot 3-8
power spectral density 4-2
power spectrum 4-2
property values
 changing 4-9
property-value pairs 4-8
PSD 4-2
pseudo spectrum 4-2
pulse trains
 example 2-7
 pulstran function 2-7

R

ramp 2-5
references
 general DSP 2-12

S

sawtooth function
 example 2-6
sawtooth wave 2-6
set 4-9 to 4-10
signal
 average power 4-11
signals
 adding noise 2-4
 aperiodic 2-6
 chirp 2-7
 continuous (analog) 1-4
 diric function 2-9
 discrete (digital) 1-4

- generating 2-5
- multichannel 2-5
- periodic 2-5
- plotting 2-4
- pulstran function 2-7
- representing 2-2
- sawtooth 2-6
- sinc 2-8
- sinusoidal 2-4
- square wave 2-6
- sinc function 2-8
- single-precision inputs 1-3
- spectrum
 - creating 4-6
 - mean-square spectrum 4-2
 - options object 4-10
 - PSD 4-2
 - pseudo spectrum 4-2
- spectrum object
 - creating 4-4
 - Welch 4-4
- sptool GUI
 - data entering 2-11
- square function
 - example 2-6
- square wave
 - See* square function 2-6
- step 2-5
- step response 3-8

T

- time vectors 2-4
- tools
 - interactive GUIs 1-7

U

- unit impulse function 2-5
- unit ramp function 2-5
- unit sample multichannel 2-5
- unit step function 2-5

V

- vectors
 - data representation 2-2
 - waveform generation 2-4

W

- waveforms, *see* signals
- Welch spectrum object 4-4
- white noise 2-4

Z

- zero-phase response 3-8