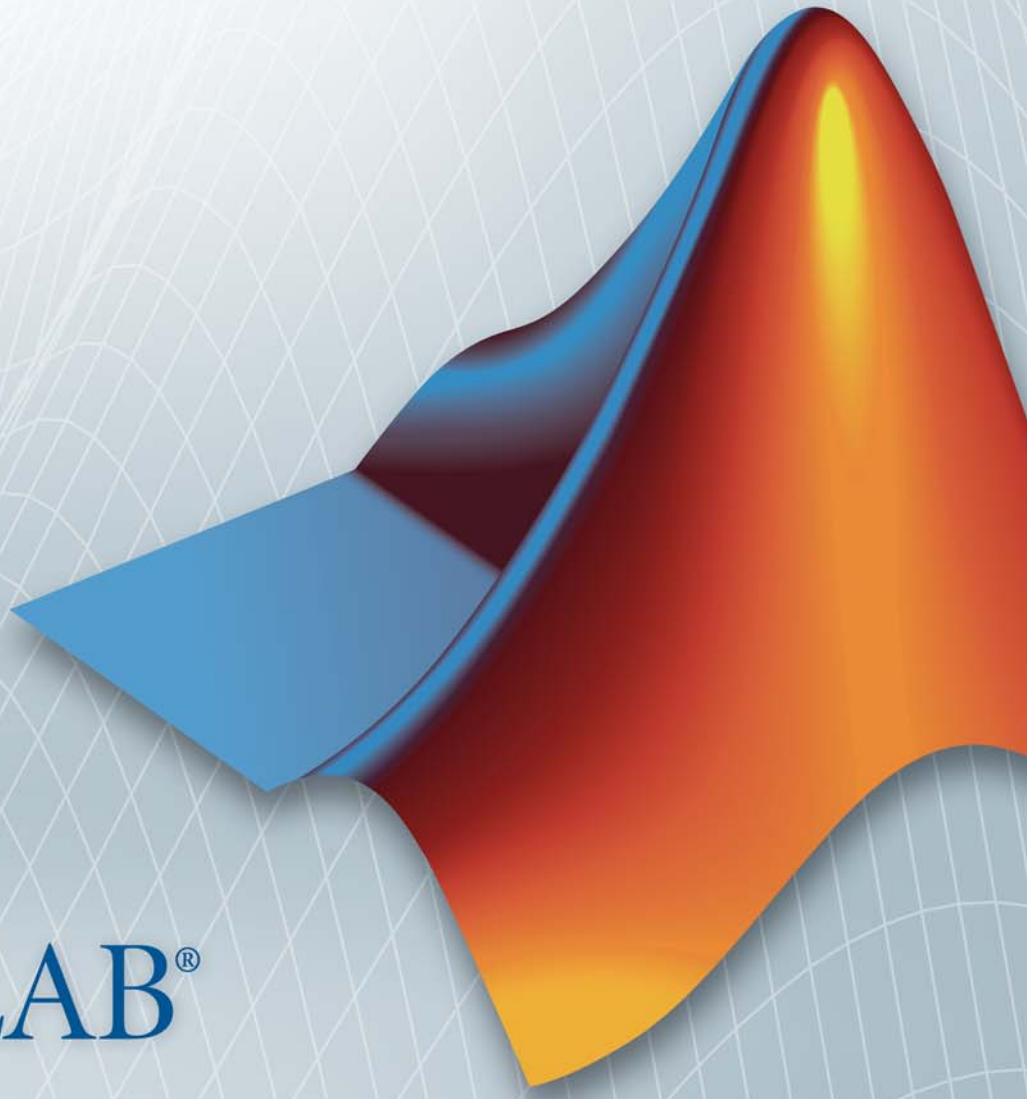
Signal Processing Toolbox™

Getting Started Guide

R2012b



MATLAB®

MathWorks®

How to Contact MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Signal Processing Toolbox™ Getting Started Guide

© COPYRIGHT 2006–2012 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

September 2006	First printing	New for Version 6.6 (Release 2006b)
March 2007	Online only	Revised for Version 6.7 (Release 2007a)
September 2007	Online only	Revised for Version 6.8 (Release 2007b)
March 2008	Online only	Revised for Version 6.9 (Release 2008a)
October 2008	Online only	Revised for Version 6.10 (Release 2008b)
March 2009	Online only	Revised for Version 6.11 (Release 2009a)
September 2009	Online only	Revised for Version 6.12 (Release 2009b)
March 2010	Online only	Revised for Version 6.13 (Release 2010a)
September 2010	Online only	Revised for Version 6.14 (Release 2010b)
April 2011	Online only	Revised for Version 6.15 (Release 2011a)
September 2011	Online only	Revised for Version 6.16 (Release 2011b)
March 2012	Online only	Revised for Version 6.17 (Release 2012a)
September 2012	Online only	Revised for Version 6.18 (Release 2012b)

Contents

1 Overview

Product Description 1-2
Key Features 1-2

Interactive Tools 1-3

Extensibility 1-4
Expanding the Toolbox 1-4
Expanding the Toolbox with Other Toolboxes 1-4

2 Basic Signal Processing Concepts

Representing Signals 2-2
Numeric Arrays 2-2
Vector Representation 2-2

Waveform Generation: Time Vectors and Sinusoids . . . 2-4
Time Vectors 2-4
Impulse, Step, and Ramp Functions 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
Importing Data from Within the MATLAB Environment . . 2-11
Importing Data from Outside the MATLAB Environment 2-11

Converting Data into a MAT-File 2-11
Exporting Data 2-12
Data Precision 2-12

Selected Bibliography 2-13

3 Design a Filter with fdesign and filterbuilder

Filter Design Process Overview 3-2

Design a Filter Using fdesign 3-4

Design a Filter Using filterbuilder 3-10

4 Filter Design with the FDATool GUI

Introduction 4-2

Designing the Filter 4-3

Analyzing the Filter 4-8

Designing Additional Filters 4-10

Viewing and Annotating the Filter 4-11
Viewing the Filter in FVTool 4-11
Using FVTool for Annotation 4-15

Exporting Filters from FDATool 4-17
Filtering with dfilt 4-18

Designing Filters Using Command Line Functions . . . **4-21**

5 Spectral Analysis

Introduction **5-2**
Spectral Estimators **5-2**
Spectral Analysis Algorithms **5-2**
Spectral Analysis Objects **5-3**

Creating a Spectral Analysis Object **5-4**

Producing a PSD Estimate **5-6**

Changing Spectral Analysis Object Property Values . . **5-8**
Using the set command to Set Property Values **5-8**
Using Options Objects to Set Property Values **5-11**

Measuring Signal Power **5-13**

Index

1

Overview

- “Product Description” on page 1-2
- “Interactive Tools” on page 1-3
- “Extensibility” on page 1-4

Product Description

Perform signal processing, analysis, and algorithm development

Signal Processing Toolbox™ provides industry-standard algorithms for analog and digital signal processing (DSP). You can use the toolbox to visualize signals in time and frequency domains, compute FFTs for spectral analysis, design FIR and IIR filters, and implement convolution, modulation, resampling, and other signal processing techniques. Algorithms in the toolbox can be used as a basis for developing custom algorithms for audio and speech processing, instrumentation, and baseband wireless communications.

Key Features

- Signal and linear system models
- Signal transforms, including fast Fourier transform (FFT), discrete Fourier transform (DFT), and short-time Fourier transform (STFT)
- Waveform and pulse generation functions, including sine, square, sawtooth, and Gaussian pulse
- Transition metrics, pulse metrics, and state-level estimation functions for bilevel waveforms
- Statistical signal measurements and data windowing functions
- Power spectral density estimation algorithms, including periodogram, Welch, and Yule-Walker
- Digital FIR and IIR filter design, analysis, and implementation methods
- Analog filter design methods, including Butterworth, Chebyshev, and Bessel
- Linear prediction and parametric time-series modeling

Interactive Tools

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

- The Filter Design and Analysis Tool (`fdatool`) and `filterbuilder` provide a comprehensive collection of features for addressing filter design. Both FDAtool and filterbuilder offer seamless access to the additional filter design methods, quantization features, C-code generation and other enhanced filtering features of the DSP System Toolbox™ product when that product is installed. If you have the Filter Design HDL Coder™ product installed, you can also generate HDL code from both FDATool and filterbuidler.
- The Filter Visualization Tool (`fvtool`) provides a graphical environment for viewing, annotating, and printing filter response plots.
- The Signal Processing Tool (`sptool`) provides a rich graphical environment for signal viewing, filter design, and spectral analysis.
- 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.

Extensibility

In this section...
"Expanding the Toolbox" on page 1-4
"Expanding the Toolbox with Other Toolboxes" on page 1-4

Expanding the Toolbox

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

Expanding the Toolbox with Other Toolboxes

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

- DSP System Toolbox — Toolbox for advanced filter design and producing Simulink® models.
- Fixed-Point Toolbox™ — Toolbox for using fixed-point arithmetic
- Filter Design HDL Coder — Toolbox for creating and exporting HDL code

2

Basic Signal Processing Concepts

- "Representing Signals" on page 2-2
- "Waveform Generation: Time Vectors and Sinusoids" on page 2-4
- "Working with Data" on page 2-11
- "Selected Bibliography" on page 2-13

Representing Signals

In this section...

"Numeric Arrays" on page 2-2

"Vector Representation" on page 2-2

Numeric Arrays

The central data construct in the MATLAB environment 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 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
```

If the sequence has complex-valued elements, the transpose operator takes the conjugate of the sequence elements. To transform a complex-valued row vector into a column vector without taking conjugates, use the `.'` or non-conjugate transpose:

```
x=[1-i 3+i 2+3*i 4-2*i]; %1X4
x=x.'; %4X1
```

Waveform Generation: Time Vectors and Sinusoids

In this section...

"Time Vectors" on page 2-4
"Impulse, Step, and Ramp Functions" on page 2-5
"Multichannel Signals" on page 2-5
"Common Periodic Waveforms" on page 2-5
"Common Aperiodic Waveforms" on page 2-6
"The pulstran Function" on page 2-7
"The Sinc Function" on page 2-8
"The Dirichlet Function" on page 2-9

Time Vectors

Most toolbox functions 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

```
rng default;
yn = y + 0.5*randn(size(t));
plot(t(1:50),yn(1:50))
```

Impulse, Step, and Ramp Functions

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)';
imp= [1; zeros(99,1)];       % Impulse
unit_step = ones(100,1);     % Step (with 0 initial cond.)
ramp_sig= t;                 % Ramp
quad_sig=t.^2;               % Quadratic
sq_wave = square(4*pi*t);    % Square wave with period 0.5
```

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 = [ramp_sig quad_sig sq_wave];
```

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));
```

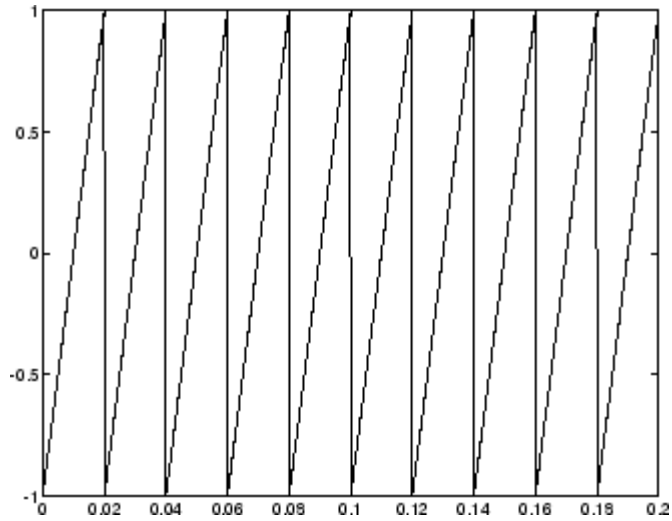
Common Periodic Waveforms

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.

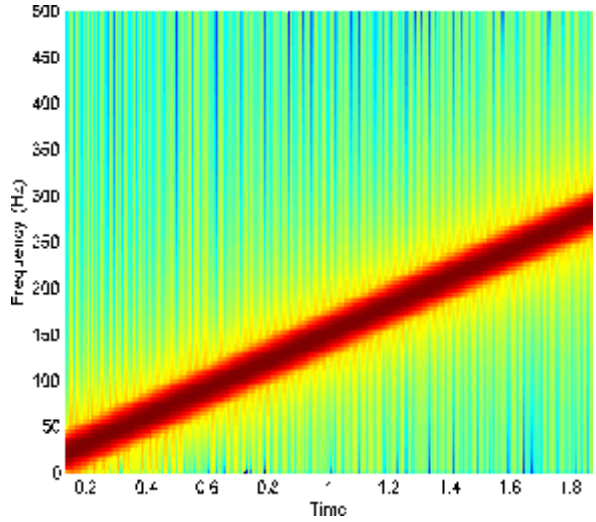
- `chirp` generates a linear, log, or quadratic swept-frequency cosine signal. An optional parameter specifies alternative sweep methods. An optional parameter `phi` allows initial phase to be specified in degrees.

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')
```

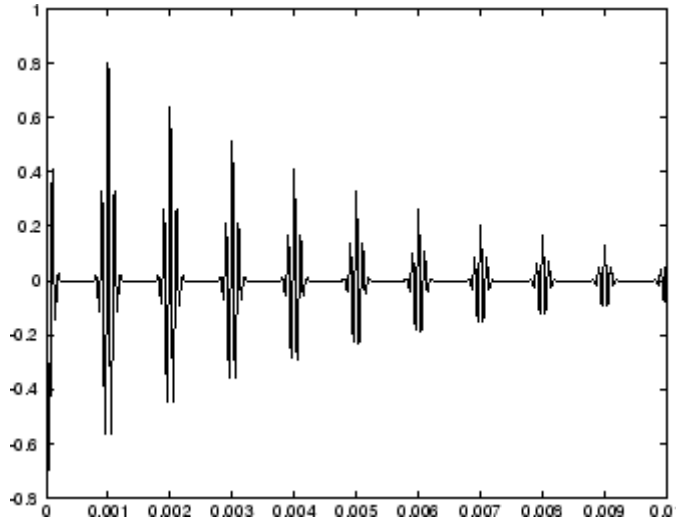


The pulstran Function

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. Viewed as a function of time, or space, the sinc function is the inverse Fourier transform of the rectangular pulse in frequency centered at zero of width 2π and height 1. The following equation defines the sinc function:

$$\frac{\sin(\pi x)}{\pi x} = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{i\omega x} d\omega.$$

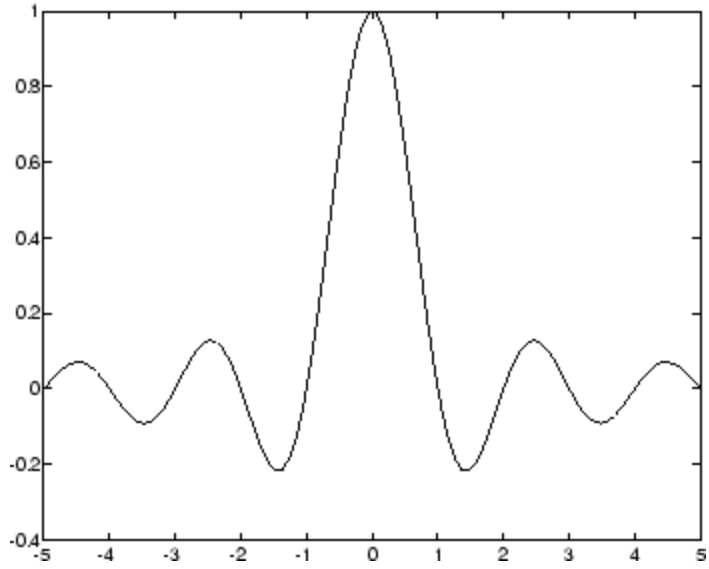
The sinc function has a value of 1 when x is equal to zero, and a value of

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

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

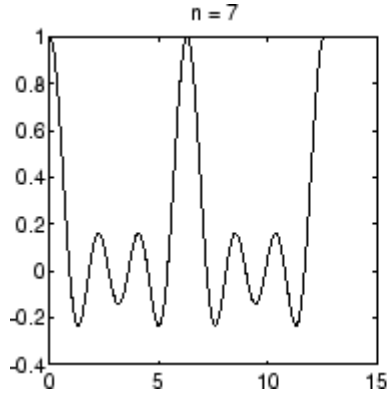
The toolbox function `diric` computes the Dirichlet function, sometimes called the *periodic sinc* or *aliased sinc* function, for an input vector or matrix `x`. The Dirichlet function, *D(x)* is:

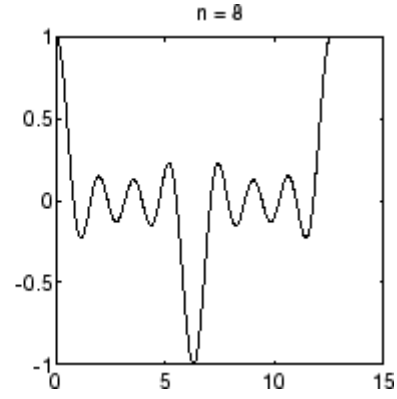
$$D(x) = \begin{cases} \dfrac{\sin(Nx/2)}{N\sin(x/2)} & x \neq 2\pi k, \quad k = 0, \pm 1, \pm 2, \pm 3, \ldots \\ (-1)^{k(N-1)} & x = 2\pi k, \quad k = 0, \pm 1, \pm 2, \pm 3, \ldots \end{cases}$$

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

In this section...
"Importing Data from Within the MATLAB Environment" on page 2-11
"Importing Data from Outside the MATLAB Environment" on page 2-11
"Converting Data into a MAT-File" on page 2-11
"Exporting Data" on page 2-12
"Data Precision" on page 2-12

Importing Data from Within the MATLAB Environment

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`

Importing Data from Outside the MATLAB Environment

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.

Converting Data into a MAT-File

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 "Manually Converting Data Passed to Functions" documentation for details. MATLAB reads such files using the `load` command.

Exporting Data

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

Data Precision

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. DSP System Toolbox and Fixed-Point Toolbox products enable single-precision floating-point and fixed-point support for most `dfilt` structures.

Selected Bibliography

Algorithm development for Signal Processing Toolbox functions 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.

References

[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. Schafer. *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.

3

Design a Filter with fdesign and filterbuilder

- "Filter Design Process Overview" on page 3-2
- "Design a Filter Using fdesign" on page 3-4
- "Design a Filter Using filterbuilder" on page 3-10

Filter Design Process Overview

Note You must have the Signal Processing Toolbox installed to use `fdesign` and `filterbuilder`. Advanced capabilities are available if your installation additionally includes the DSP System Toolbox license. You can verify the presence of both toolboxes by typing `ver` at the command prompt.

Filter design through user-defined specifications is the core of the `fdesign` approach. This specification-centric approach places less emphasis on the choice of specific filter algorithms, and more emphasis on performance during the design a good working filter. For example, you can take a given set of design parameters for the filter, such as a stopband frequency, a passband frequency, and a stopband attenuation, and— using these parameters— design a specification object for the filter. You can then implement the filter using this specification object. Using this approach, it is also possible to compare different algorithms as applied to a set of specifications.

There are two distinct objects involved in filter design:

- Specification Object — Captures the required design parameters of a filter
- Implementation Object — Describes the designed filter; includes the array of coefficients and the filter structure

The distinction between these two objects is at the core of the filter design methodology. The basic attributes of each of these objects are outlined in the following table.

Specification Object	Implementation Object
High-level specification	Filter coefficients
Algorithmic properties	Filter structure

You can run the code in the following examples from the Help browser (select the code, right-click the selection, and choose **Evaluate Selection** from the context menu), or you can enter the code on the MATLAB command line. Before you begin this example, start MATLAB and verify that you have installed the Signal Processing Toolbox software. If you wish to access the

full functionality of `fdesign` and `filterbuilder`, you should additionally obtain the DSP System Toolbox software. You can verify the presence of these products by typing `ver` at the command prompt.

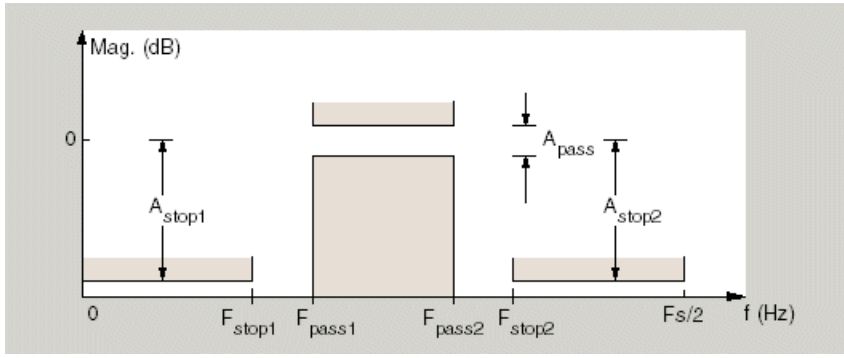
Design a Filter Using fdesign

Use the following two steps to design a simple filter.

1. Create a filter specification object.
2. Design your filter.

Design a Filter in Two Steps

Assume that you want to design a bandpass filter. Typically a bandpass filter is defined as shown in the following figure.



In this example, a sampling frequency of `Fs` = 48 kHz is used. This bandpass filter has the following specifications, specified here using MATLAB code:

```
A_stop1 = 60;  % Attenuation in the first stopband = 60 dB
F_stop1 = 8400;  % Edge of the stopband = 8400 Hz
F_pass1 = 10800; % Edge of the passband = 10800 Hz
F_pass2 = 15600; % Closing edge of the passband = 15600 Hz
F_stop2 = 18000; % Edge of the second stopband = 18000 Hz
A_stop2 = 60;  % Attenuation in the second stopband = 60 dB
A_pass = 1;  % Amount of ripple allowed in the passband = 1 dB
```

In the following two steps, these specifications are passed to the `fdesign.bandpass` method as parameters.

Step 1

To create a filter specification object, evaluate the following code at the MATLAB prompt:

```
d = fdesign.bandpass
```

Now, pass the filter specifications that correspond to the default `Specification` — `fst1`,`fp1`,`fp2`,`fst2`,`ast1`,`ap`,`ast2`. This example adds `fs` as the final input argument to specify the sampling frequency of 48 kHz.

```
>> BandPassSpecObj = ...
   fdesign.bandpass('Fst1,Fp1,Fp2,Fst2,Ast1,Ap,Ast2', ...
  F_stop1, F_pass1, F_pass2, F_stop2, A_stop1, A_pass, ...
  A_stop2, 48000)
```

Note The order of the filter is not specified, allowing a degree of freedom for the algorithm design in order to achieve the specification. The design will be a minimum order design.

The specification parameters, such as `Fstop1`, are all given default values when none are provided. You can change the values of the specification parameters after the filter specification object has been created. For example, if there are two values that need to be changed, `Fpass2` and `Fstop2`, use the `set` command, which takes the object first, and then the parameter value pairs. Evaluate the following code at the MATLAB prompt:

```
>> set(BandPassSpecObj, 'Fpass2', 15800, 'Fstop2', 18400)
```

`BandPassSpecObj` is the new filter specification object which contains all the required design parameters, including the filter type.

You may also change parameter values in filter specification objects by accessing them as if they were elements in a `struct` array.

```
>> BandPassSpecObj.Fpass2=15800;
```

Step 2

Design the filter by using the `design` command. You can access the design methods available for you specification object by calling the `designmethods` function. For example, in this case, you can execute the command

```
>> designmethods(BandPassSpecObj)


Design Methods for class
fdesign.bandpass (Fst1,Fp1,Fp2,Fst2,Ast1,Ap,Ast2):


butter
cheby1
cheby2
ellip
equiripple
kaiserwin
```

After choosing a design method use, you can evaluate the following at the MATLAB prompt (this example assumes you've chosen '`equiripple`'):

```
>> BandPassFilt = design(BandPassSpecObj, 'equiripple')

BandPassFilt =

     FilterStructure: 'Direct-Form FIR'
          Arithmetic: 'double'
           Numerator: [1x44 double]
    PersistentMemory: false
```

If you have the DSP System Toolbox installed, you can also design your filter with a filter System object™. To create a filter System object with the same specification object `BandPassSpecObj`, you can execute the commands

```
>> designmethods(BandPassSpecObj,...
'SystemObject',true)
```

```
Design Methods that support System objects for class
fdesign.bandpass (Fst1,Fp1,Fp2,Fst2,Ast1,Ap,Ast2):


butter
cheby1
cheby2
ellip
equiripple
kaiserwin


>> BandPassFiltSysObj = design(BandPassSpecObj,...
'equiripple','SystemObject',true)


  System: dsp.FIRFilter

  Properties:
              Structure: 'Direct form'
        NumeratorSource: 'Property'
              Numerator: [1x44 double]
      InitialConditions: 0
    FrameBasedProcessing: true

  Show fixed-point properties
```

Available design methods and design options for filter System objects are not necessarily the same as those for filter objects.

Note If you do not specify a design method, a default method will be used. For example, you can execute the command

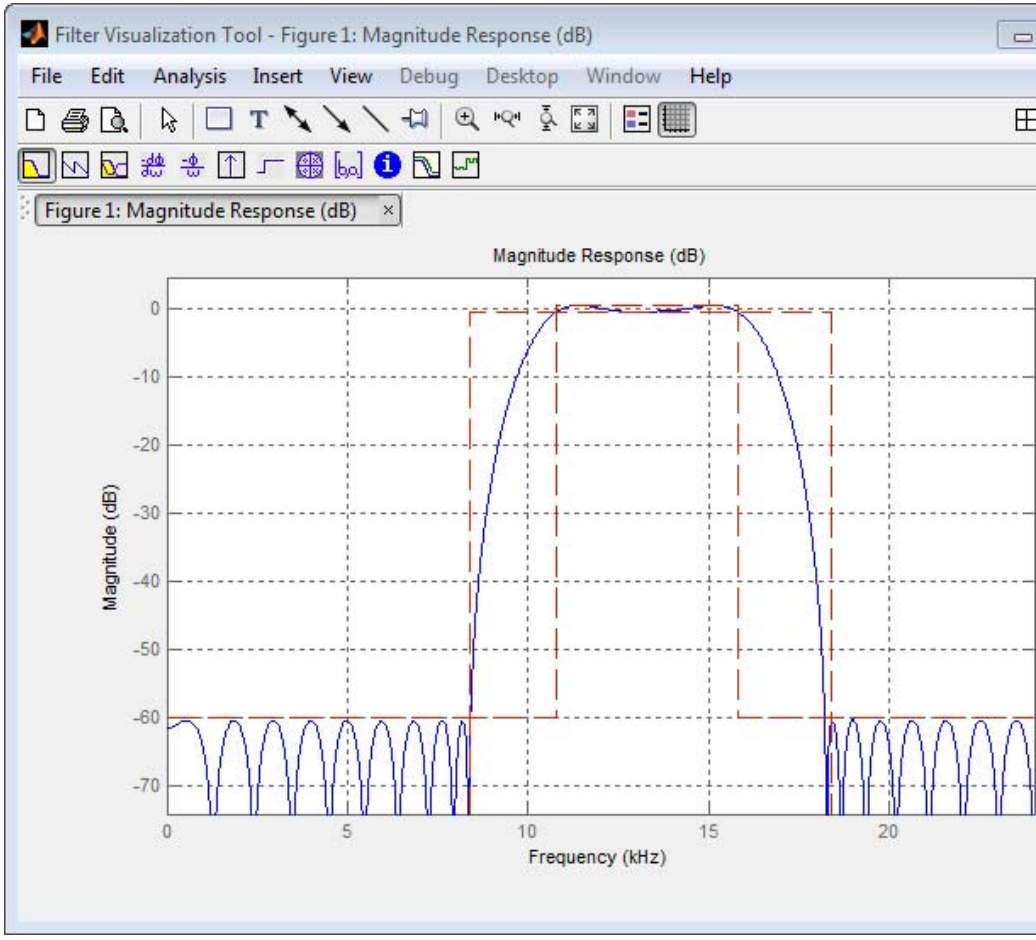
```
>> BandPassFilt = design(BandPassSpecObj)

BandPassFilt =

     FilterStructure: 'Direct-Form FIR'
          Arithmetic: 'double'
           Numerator: [1x44 double]
    PersistentMemory: false
```

and a design method will be selected automatically.

To check your work, you can plot the filter magnitude response using the Filter Visualization tool. Verify that all the design parameters are met:

```
>> fvtool(BandPassFilt) %plot the filter magnitude response
```

If you have the DSP System Toolbox installed, the Filter Visualization tool produces the following figure with the dashed red lines indicating the transition bands and unity gain (0 in dB) over the passband.


Filter Visualization Tool - Figure 1: Magnitude Response (dB)
File
Edit
Analysis
Insert
View
Debug
Desktop
Window
Help
Figure 1: Magnitude Response (dB)
Magnitude Response (dB)
Magnitude (dB)
0
-10
-20
-30
-40
-50
-60
-70
0
5
10
15
20
Frequency (kHz)

Design a Filter Using filterbuilder

Filterbuilder presents the option of designing a filter using a GUI dialog box as opposed to the command line instructions. You can use Filterbuilder to design the same bandpass filter designed in the previous section, "Design a Filter Using fdesign" on page 3-4

Design a Simple Filter in Filterbuilder

To design the filter using the Filterbuilder GUI:

1. Type the following at the MATLAB prompt:

   ```
   filterbuilder
   ```

2. Select `Bandpass` filter response from the list in the dialog box, and hit the **OK** button.

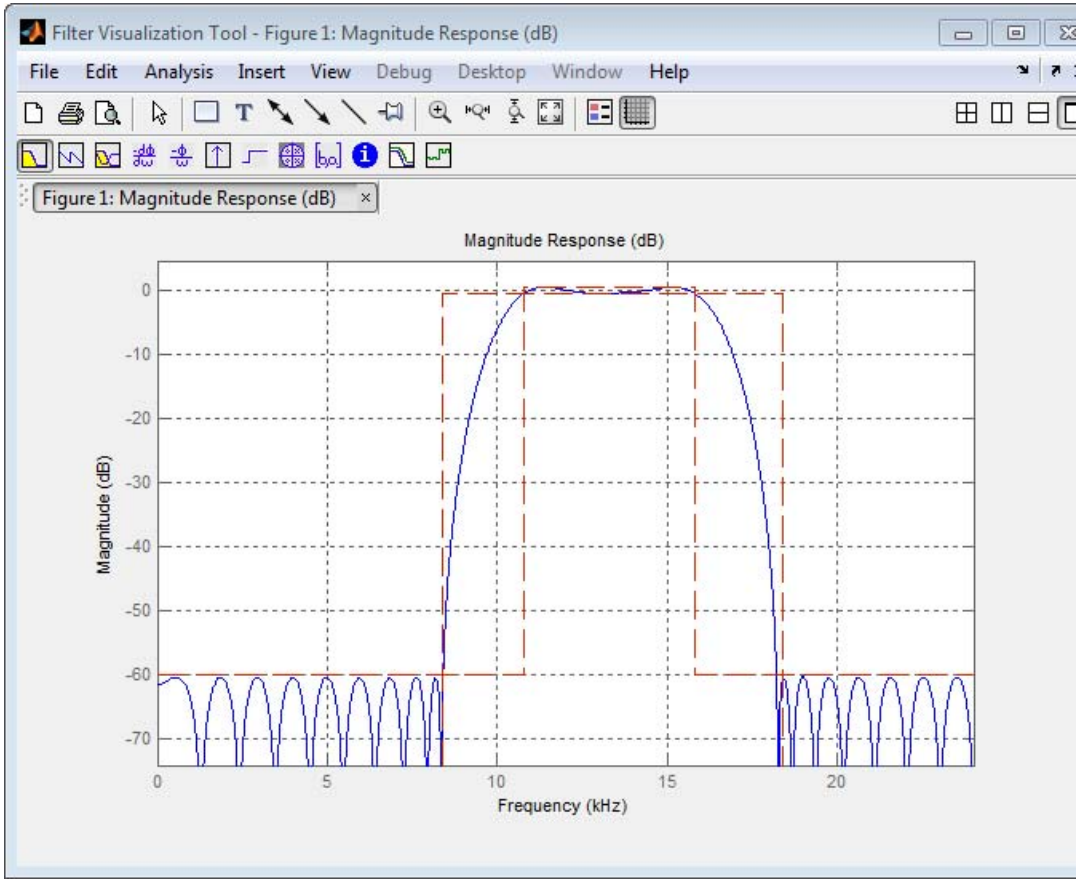
3. Enter the correct frequencies for **Fpass2** and **Fstop2**, then click **OK**. Here the specification uses normalized frequency, so that the passband and stopband edges are expressed as a fraction of the Nyquist frequency (in this case, 48/2 kHz). The following message appears at the MATLAB prompt:

   ```
   The variable 'Hbp' has been exported to the command window.
   ```

   If you display the Workspace tab, you see the object `Hbp` has been placed on your workspace.

4. To check your work, plot the filter magnitude response using the Filter Visualization tool. Verify that all the design parameters are met:

   ```
   fvtool(Hbp) %plot the filter magnitude response
   ```



Note that the dashed red lines on the preceding figure will only appear if you are using the DSP System Toolbox software.

4

Filter Design with the FDATool GUI

- "Introduction" on page 4-2
- "Designing the Filter" on page 4-3
- "Analyzing the Filter" on page 4-8
- "Designing Additional Filters" on page 4-10
- "Viewing and Annotating the Filter" on page 4-11
- "Exporting Filters from FDATool" on page 4-17
- "Designing Filters Using Command Line Functions" on page 4-21

Introduction

This section describes how to graphically design and implement digital filters using the Signal Processing Toolbox FDATool GUI. 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.

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

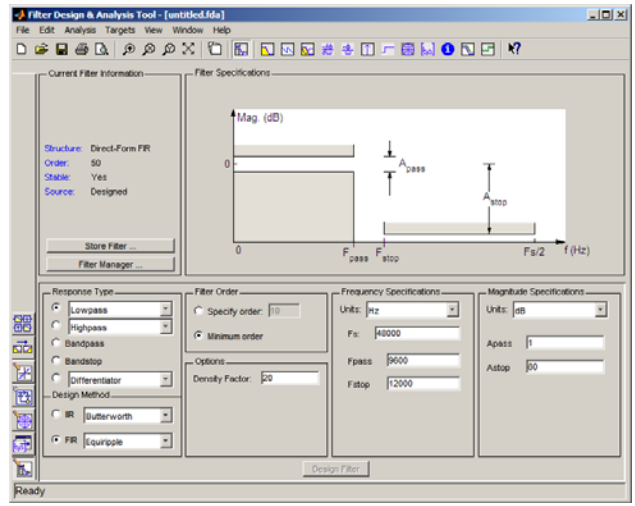
Designing the Filter

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.

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.



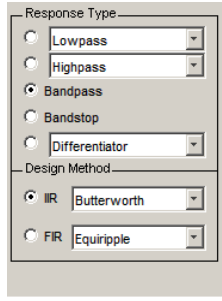
The bottom half of FDATool shows the Filter Design panel, where you specify the filter parameters. Other panels, such as Import filter from workspace and Pole/Zero Editor, which you access with the buttons on

the lower left, are also displayed in this area. If you have other products installed, you may see additional buttons.

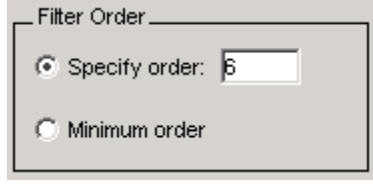
Note that when you open FDATool, **Design Filter** is not enabled. You must make a change to the default filter design in order to enable **Design Filter**. This is true each time you want to change the filter design. Changes to radio button items or drop down menu items such as those under **Response Type** or **Filter Order** enable **Design Filter** immediately. Changes to specifications in text boxes such as **Fs**, **Fpass**, and **Fstop** require you to click outside the text box to enable **Design Filter**.

2 In the **Response Type** pane, select **Bandpass**.

3 In the **Design Method** pane, select **IIR**, and then select `Butterworth` from the selection list.



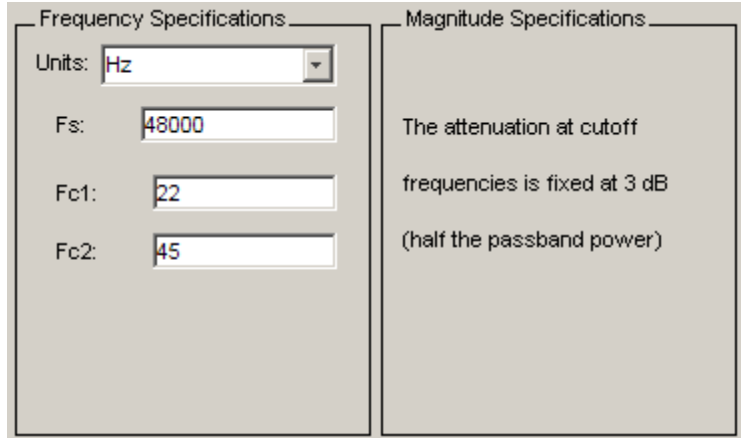
4 For the Filter Order, select **Specify order**, and then enter `6`.



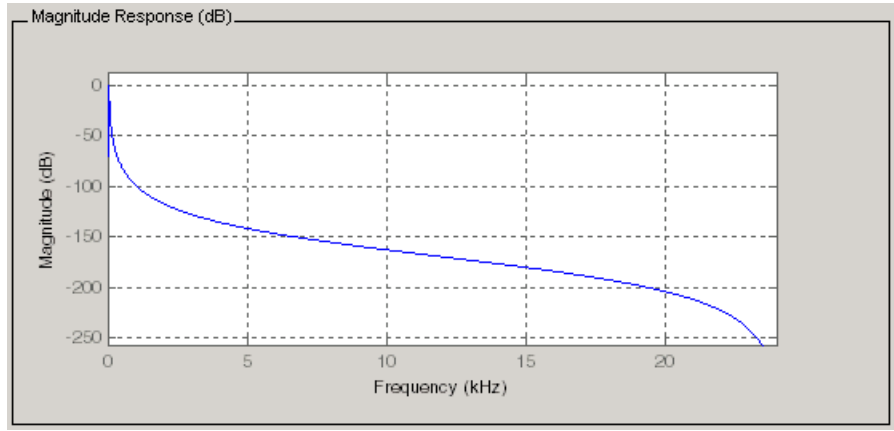
5 Set the Frequency Specifications as follows:

Parameter	Setting	Description
Units	`Hz`	Units for the parameters
Fs	`48000`	Sampling frequency

Parameter	Setting	Description
Fc1	22	First cutoff frequency (i.e., the frequency preceding the passband at which the magnitude response is 3 dB below the passband gain)
Fc2	45	Second cutoff frequency (i.e., the frequency following the passband at which the magnitude response is 3 dB below the passband gain)



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.

Store Filter:
Enter filter name to be stored in the Filter Manager
Bandpass Butterworth-1
OK
Cancel

‘

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.

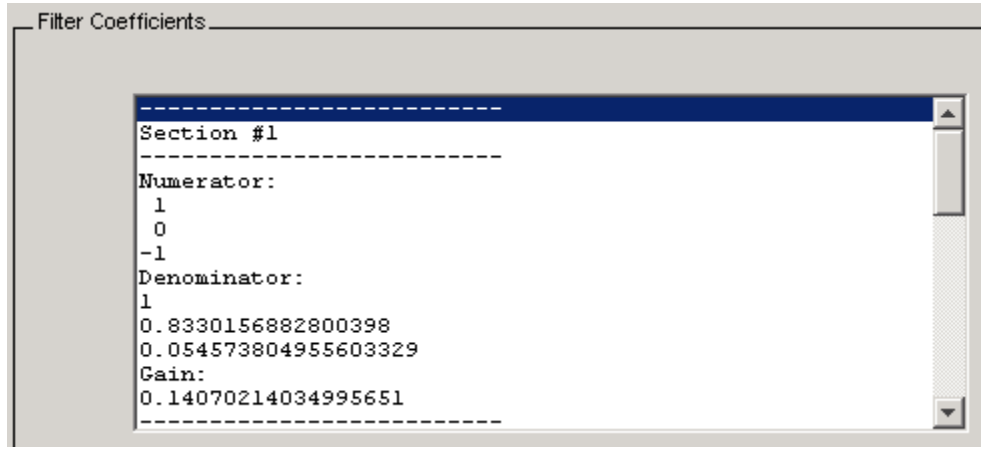
Response	Toolbar Button Image
Filter specifications	
Magnitude response	
Phase response	
Magnitude and Phase responses	
Group delay	
Phase delay	
Impulse response	
Step response	
Pole-zero plot	
Filter coefficients	
Filter information	

Note Other analyses are available if you have the DSP System Toolbox product 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.



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.

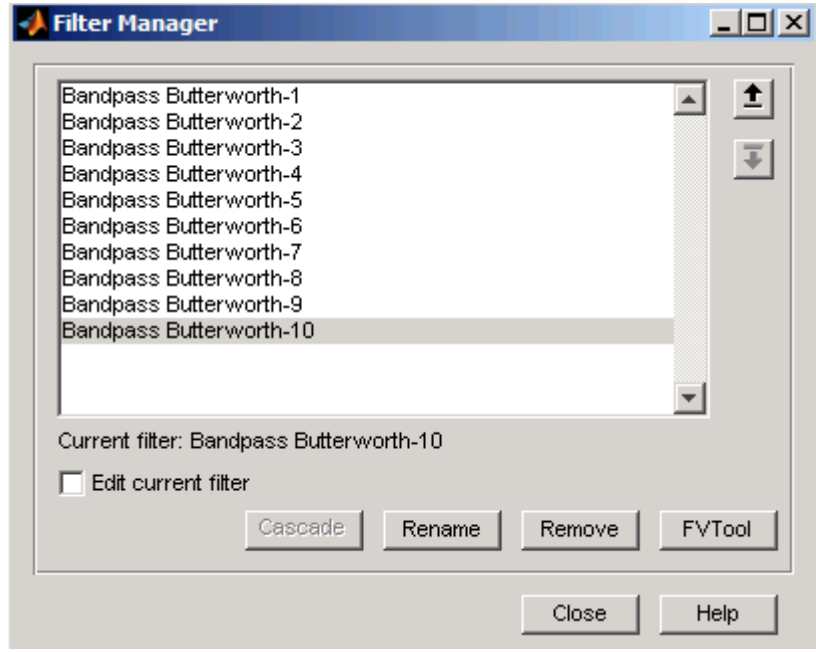
Viewing and Annotating the Filter

In this section...
"Viewing the Filter in FVTool" on page 4-11
"Using FVTool for Annotation" on page 4-15

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.

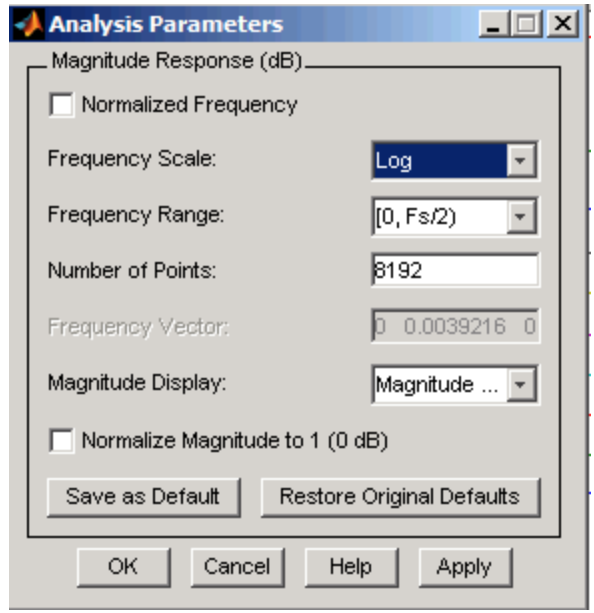


2. Press Ctrl+click on each filter name to select all the filters, and then click **FVTool**. FVTool opens with the filter responses overlaid for easy comparison. (If you want to view a single filter in FVTool, click the **Full**

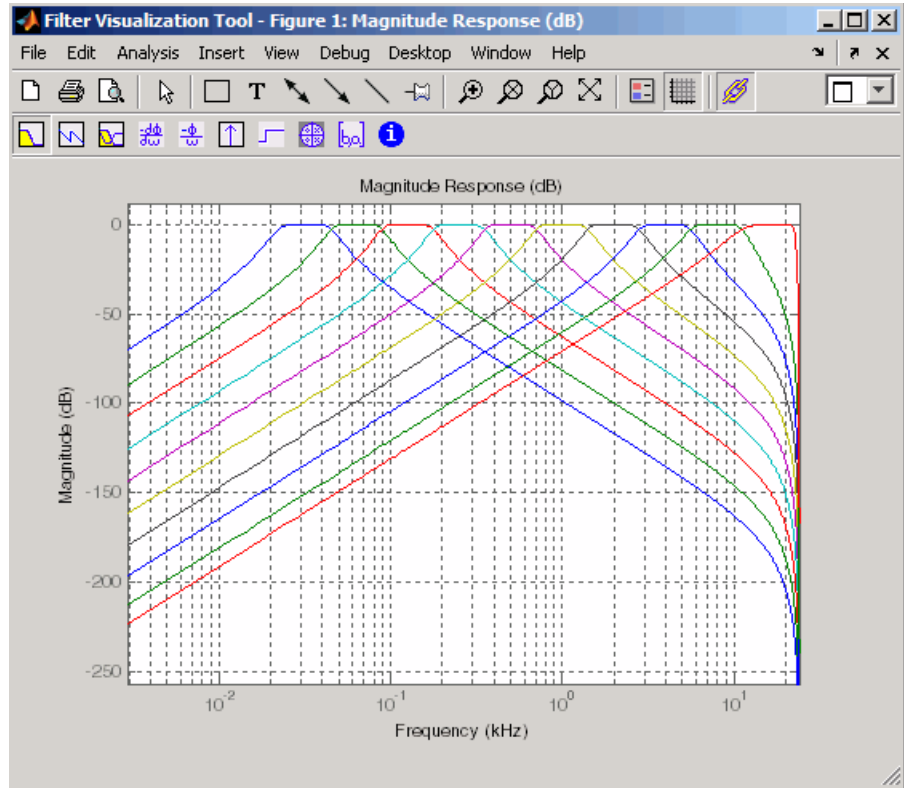
View Analysis button when that filter is shown in the FDATool display panel or select **View > Filter Visualization Tool**).

3 Change the x-axis scale to logarithmic by selecting **Analysis > Analysis Parameters** to display the Analysis Parameters dialog.

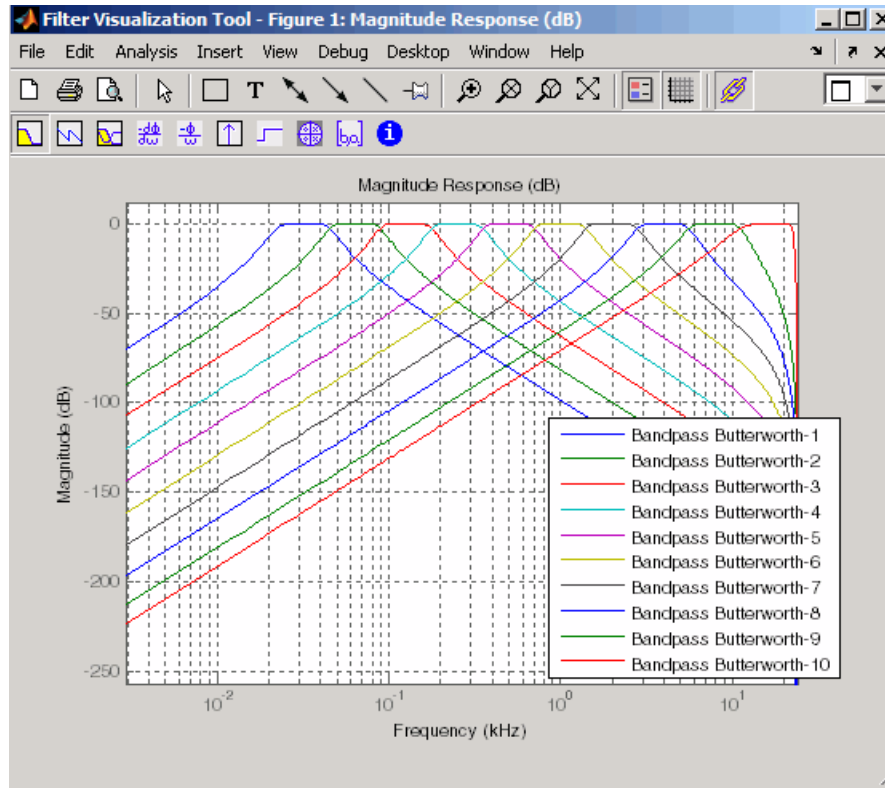
4 Change the **Frequency Scale** to `Log`.



5 Click **OK**.

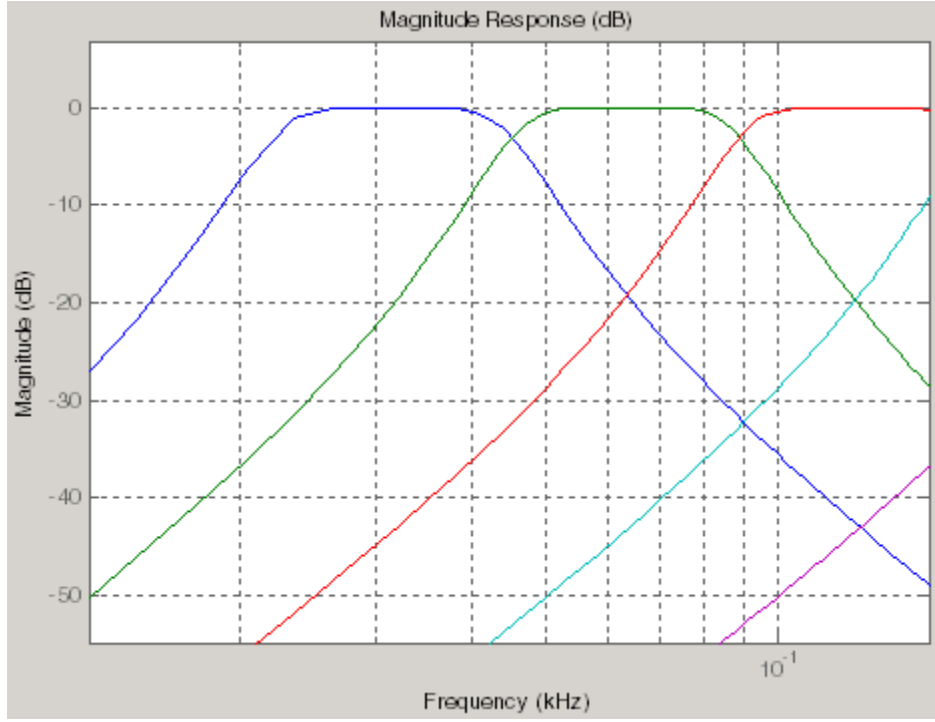


6 Click the **Legend** button to turn on the legend, which you can drag to the desired location.



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 4-8 for descriptions of the buttons.

Using FVTool for Annotation

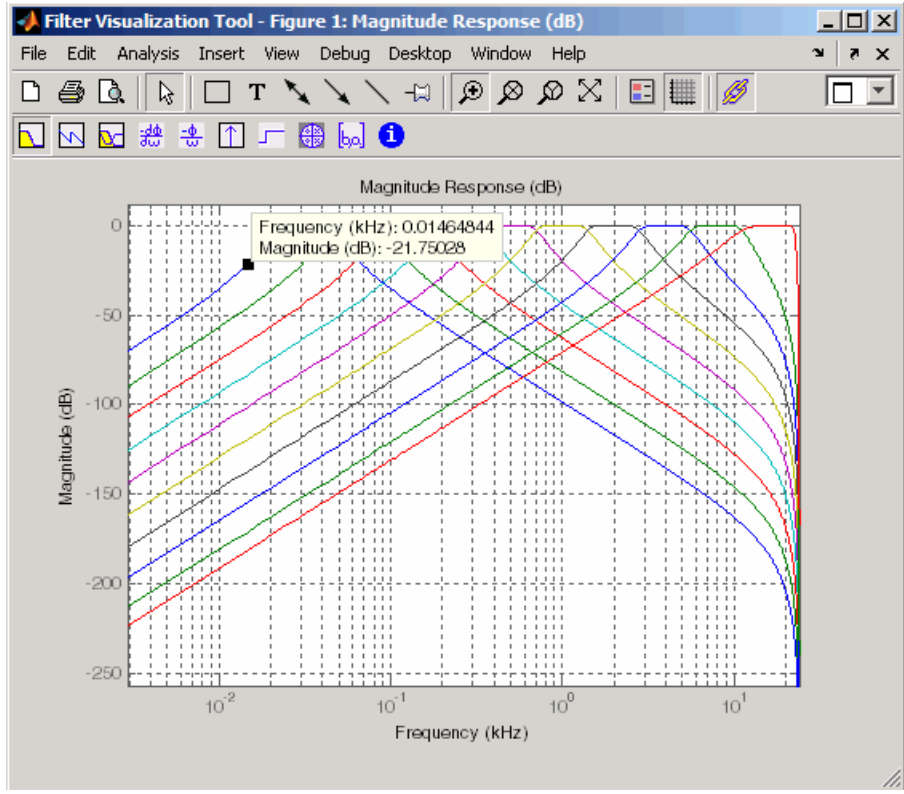
FVTool is also useful for doing further analysis, adding annotations, and printing. Available annotations include adding rectangles, text boxes, arrows and lines, and adding data tips.

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 tip by clicking on a plot at the desired point. The data tip shows the frequency and magnitude at that point.



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

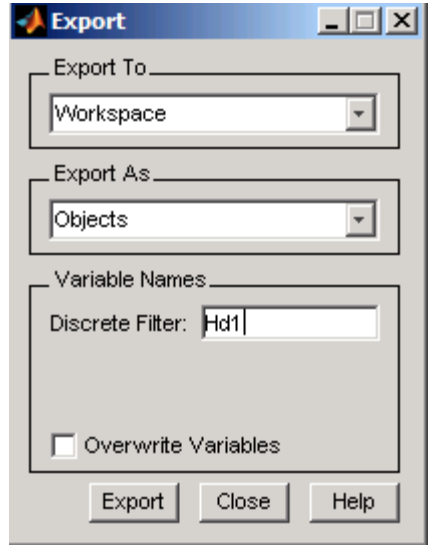
Exporting Filters from FDATool

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 as MATLAB code by using **File > Generate MATLAB code** 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                     dfilt.df2sos
  Hd10      1x1                     dfilt.df2sos
  Hd2       1x1                     dfilt.df2sos
  Hd3       1x1                     dfilt.df2sos
  Hd4       1x1                     dfilt.df2sos
  Hd5       1x1                     dfilt.df2sos
  Hd6       1x1                     dfilt.df2sos
  Hd7       1x1                     dfilt.df2sos
  Hd8       1x1                     dfilt.df2sos
  Hd9       1x1                     dfilt.df2sos
```

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 4-11 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;                   % 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
```

6 Plot the filtered data.

```
plot(yw)
```

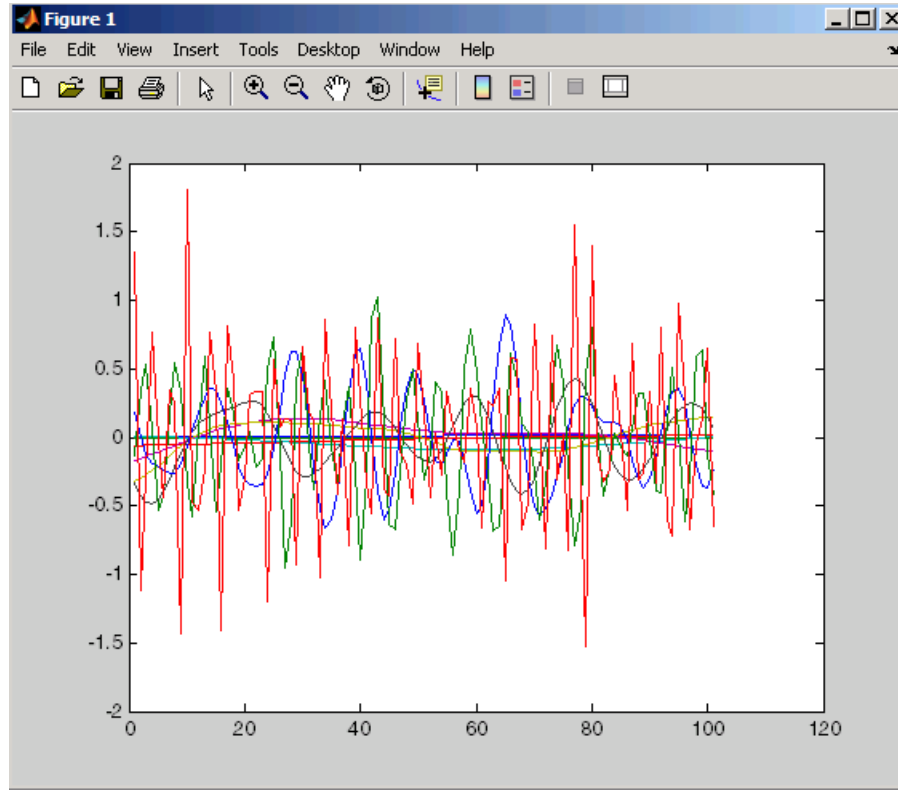


Figure 1
File Edit View Insert Tools Desktop Window Help
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
20
40
60
80
100
120

Designing Filters Using Command Line Functions

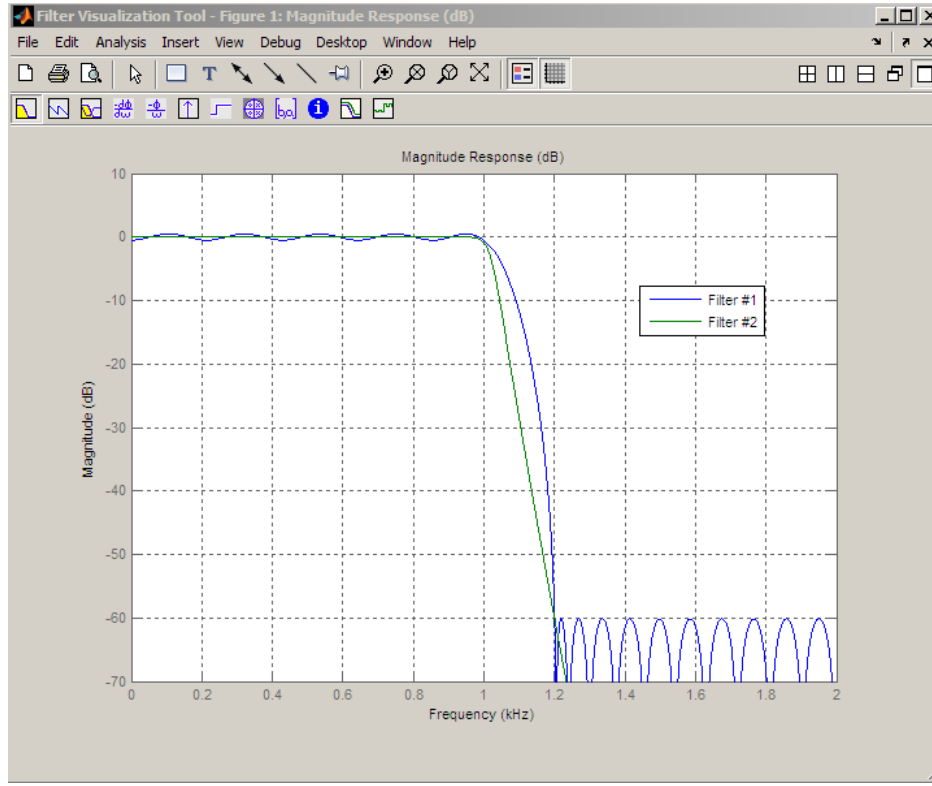
You can specify and design filters at the command line using `fdesign` and `design`. The use of `fdesign` and `design` provides a powerful and efficient way to specify and implement digital filters. With `fdesign` and `design`, digital filter design is a two-step process. In the first step, the user specifies the desired characteristics of the filter and saves those specifications in a Filter Specification Object. In the second step, the user implements the filter as a dfilt object, which can then be used to filter data. As an example of this two-step process, we will design a lowpass filter for data sampled at 20 kHz. The desired passband frequency is 1 kHz with a stopband frequency of 1.2 kHz. We will limit the passband ripple to 1 dB and require 60 dB of attenuation between the passband and stopband frequencies.

1. To specify the filter use `fdesign.lowpass`with the parameters given above. You can copy and paste the following code at the MATLAB command prompt.

   ```
   d=fdesign.lowpass('Fp,Fst,Ap,Ast',1000,1200,1,60,20000);
   ```

2. To design the filter use `design` with the appropriate design methods. In this example, we create FIR equiripple and IIR Butterworth designs and compare the filters' magnitude responses.

   ```
   Hd1=design(d,'equiripple'); %FIR equiripple design
   Hd2=design(d,'butter'); %Butterworth design
   Hd=[Hd1 Hd2];%filter object with both designs
   %compare filters
   fvtool(Hd,'legend','on'); axis([0 2 -70 10])
   ```



To determine which filter design methods are available for a given Filter Specification Object, use `designmethods`. For the lowpass filter example above, you have a choice of two FIR and four IIR digital filter designs:

```
designmethods(d)
Design Methods for class fdesign.lowpass (Fp,Fst,Ap,Ast):


butter
cheby1
cheby2
ellip
equiripple
kaiserwin
```

5

Spectral Analysis

- "Introduction" on page 5-2
- "Creating a Spectral Analysis Object" on page 5-4
- "Producing a PSD Estimate" on page 5-6
- "Changing Spectral Analysis Object Property Values" on page 5-8
- "Measuring Signal Power" on page 5-13

Introduction

In this section...

"Spectral Estimators" on page 5-2

"Spectral Analysis Algorithms" on page 5-2

"Spectral Analysis Objects" on page 5-3

Spectral Estimators

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.
- Pseudospectrum (`pseudospectrum`) returns a pseudo spectrum that does not have any units.

Spectral Analysis Algorithms

Signal Processing Toolbox software provides several algorithms to compute of spectral estimates. The following table indicates which algorithms are available to compute each type of estimator and produce a spectrum object. For details on spectrum objects, see the `spectrum` reference page.

Spectral Estimator	Algorithms
Power spectral density (`psd`)	Burg (`spectrum.burg`), Covariance (`spectrum.cov`), Modified covariance (`spectrum.mcov`), Thomson multitaper method (MTM) (`spectrum.mtm`), Periodogram (`spectrum.periodogram`), Welch (`spectrum.welch`),

Spectral Estimator	Algorithms
	Yule-Walker autoregressive (`spectrum.yulear`) See also `dspdata.psd`
Mean-square spectrum (`msspectrum`)	Periodogram (`spectrum.periodogram`), Welch (`spectrum.welch`) See also `dspdata.msspectrum`
Pseudo spectrum (`pseudospectrum`)	Eigenvector (`spectrum.eigenvector`), MUSIC (Multiple Signal Classification) (`spectrum.music`) See also `dspdata.pseudospectrum`

Spectral Analysis Objects

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 (`h = spectrum.burg`). You then pass your data and the estimator object to a spectrum estimation algorithm (`Hpsd = psd(h,x)`). In this example, `h` is a Burg spectrum object, `x` is the original input data, and `Hpsd` is the resulting PSD estimate.

For more information and examples, see Getting Started with Spectral Analysis Objects.

Creating a Spectral Analysis Object

In this example, we construct a PSD estimate of a signal using Welch's overlapped segment method. To run the example, copy and paste the following code at the MATLAB command prompt.

```
Fs=10000; %sampling frequency
t=0:(1/Fs):1; %one second time vector
y=0.4*cos(2*pi*2000*t)+0.2*sin(2*pi*1000*t)+randn(size(t));
```

Next create a default Welch spectrum object.

```
h = spectrum.welch;
```

Entering `h` at the command prompt shows the default settings for the Welch spectrum object.

```
h =

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

If you want to specify parameters instead of using default values, you can use syntax like the following:

```
h=spectrum.welch('kaiser',128,50);
```

The code creates a Welch spectrum object using a Kaiser window (see kaiser). We have set the segment length equal to 128 with an overlap percentage of 50. The Kaiser window has an additional parameter, `beta`, which governs the tradeoff between the width of the main lobe and level of energy in the sidelobes. Larger values of `beta` decrease the height of the sidelobes at the expense of widening the main lobe. You can specify additional parameters for a chosen window by passing them to the spectrum object in a cell array. For example,

```
h=spectrum.welch({'Kaiser',0.2},128,50)
h =
```

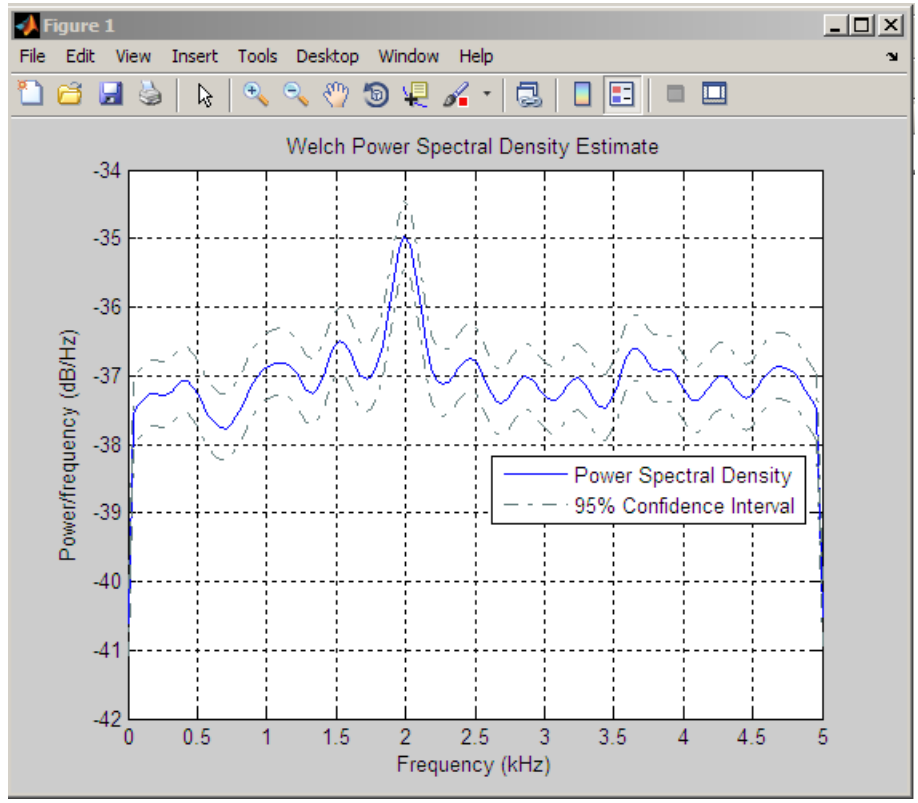
```
EstimationMethod: 'Welch'
   SegmentLength: 128
  OverlapPercent: 50
      WindowName: 'Kaiser'
            Beta: 0.2000
```

For additional information on changing the property values of spectrum objects, see “Changing Spectral Analysis Object Property Values” on page 5-8 for more information.

Producing a PSD Estimate

We now produce a PSD estimate of the signal we created in the previous section. To generate a PSD estimate, you apply a spectral estimation method on your spectrum object and data. ,

```
h = spectrum.welch;
Hpsd=psd(h,y,'Fs',Fs,'ConfLevel',0.95);
plot(Hpsd)
```



The syntax for using `psd`, `msspectrum`, or `pseudospectrum` is the same. The first input is the spectrum object (`h`) and the second input (`y`) is the signal (data), followed by any settable properties using property-value pairs (`'Fs'`,

`Fs,'ConfLevel',0.95,'`). To set property-value pairs, you list the property first and then the value for that property.

Changing Spectral Analysis Object Property Values

In this section...

"Using the set command to Set Property Values" on page 5-8

"Using Options Objects to Set Property Values" on page 5-11

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 the`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) 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')
```

Chebyshev windows have a sidelobe attenuation parameter that automatically appears in the list of properties. To change a window parameter, you 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 see a list of all available window functions for the Welch spectral analysis object, enter the following at the MATLAB command prompt:

```
set(h,'windowname')
ans =

    'Bartlett'
    'Bartlett-Hanning'
    'Blackman'
    'Blackman-Harris'
    'Bohman'
    'Chebyshev'
    'Flat Top'
    'Gaussian'
    'Hamming'
    'Hann'
    'Kaiser'
    'Nuttall'
    'Parzen'
    'Rectangular'
    'Taylor'
    'Triangular'
    'Tukey'
    'User Defined'
```

The Signal Processing Toolbox provides the most common window functions used in nonparametric spectral analysis. However, you also have the flexibility to use a user-defined window function as the next section demonstrates.

Using a User-Defined Window

This section demonstrates how to construct a Welch PSD estimate with a user-defined window, a discrete prolate spheroidal sequence (see `dpss`). First, construct a Welch spectral analysis object with a user-defined window function using the following code:

```
h=spectrum.welch('user defined')

h =
```

```
    EstimationMethod: 'Welch'
       SegmentLength: 64
      OverlapPercent: 50
          WindowName: 'User Defined'
    MATLABExpression: ''
          Parameters: []
```

To specify a discrete prolate spheroidal sequence as the window function, set the `MATLABExpression` property to `'dpss'`. You use the syntax `dpss(N,NW,1)` to construct the window function. `N` is the `SegmentLength` property, `NW` represents the time half bandwidth product, and the scalar 1 indicates use of only the first discrete prolate spheroidal sequence.

```
 set(h,'matlabexpression','dpss')

h =

    EstimationMethod: 'Welch'
       SegmentLength: 64
      OverlapPercent: 50
          WindowName: 'User Defined'
    MATLABExpression: 'dpss'
          Parameters: []
```

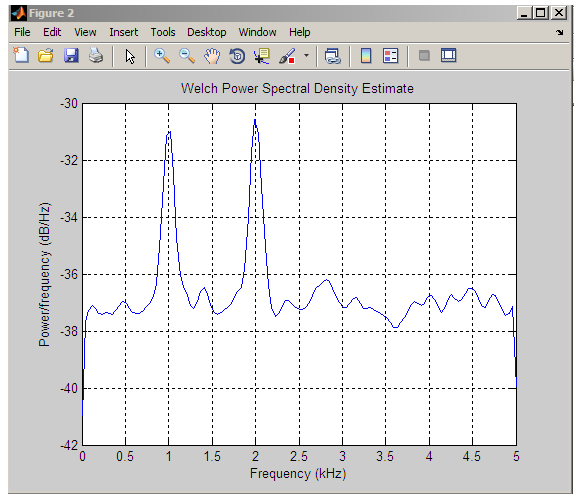
Finally, use the `Parameters` property to supply required or optional input arguments for the window function defined by `MATLABExpression`. This example uses a time half bandwidth product of 2.5. You specify the length by the SegmentLength property. Because you are not constructing a multitaper spectral estimate, you use only the first discrete prolate spheroidal sequence.

```
set(h,'parameters',{2.5,1})
h

h =

    EstimationMethod: 'Welch'
       SegmentLength: 64
      OverlapPercent: 50
          WindowName: 'User Defined'
    MATLABExpression: 'dpss'
          Parameters: {[2.5000]  [1]}
```

MATLAB evaluates the user-defined window as `dpss(64,2.5,1)`.

The following MATLAB code obtains the Welch PSD estimate of a signal with a user-defined window. Use the first discrete prolate spheroidal sequence of length 128 and a time half bandwidth product of 2.5 as the window function

```
Fs=10000; %sampling frequency
t=0:(1/Fs):1; %one second time vector
y=0.4*cos(2*pi*2000*t)+0.4*sin(2*pi*1000*t)+randn(size(t));
h=spectrum.welch('user defined');
set(h,'matlabexpression','dpss',...
'parameters',{2.5,1},'segmentlength',128);
Hs=psd(h,y,'Fs',Fs);
plot(Hs);
```



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 change any of the options properties using `set`, followed by the options object and property-value pairs, 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)
```

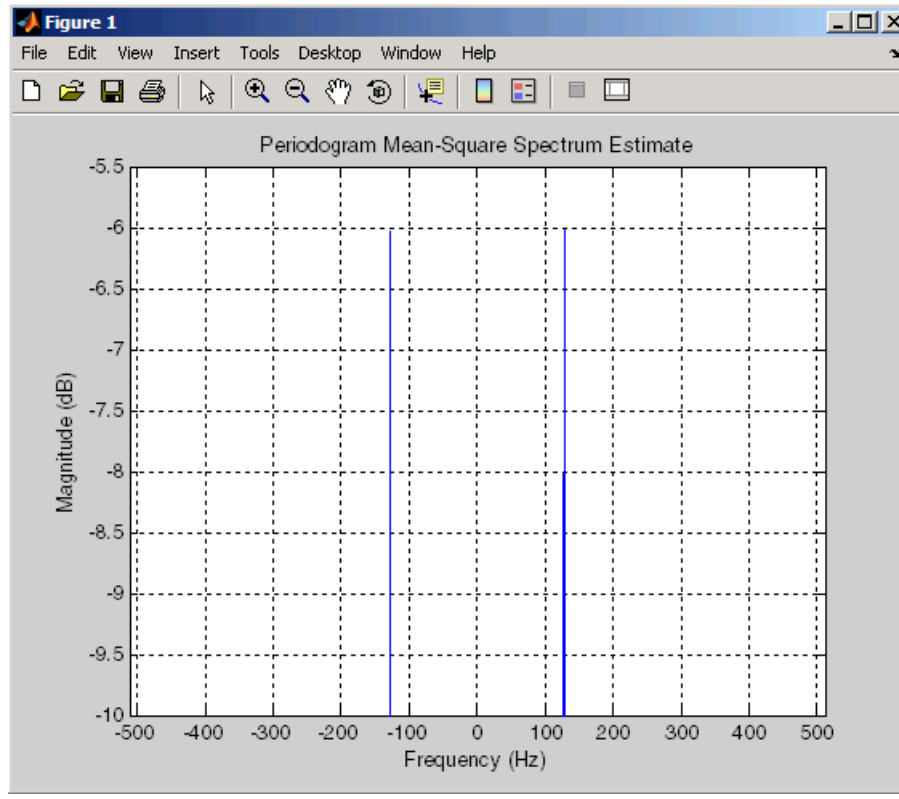
Options objects that correspond to `psd` and `pseudospectrum` are `psdopts` and `pseudospectrumopts`, respectively.

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 spectrum. A signal made up of sinusoids is an example of a power signal that has infinite energy, but finite average power . The example shows how to estimate the average power of a sinewave with a peak amplitude of 1.

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

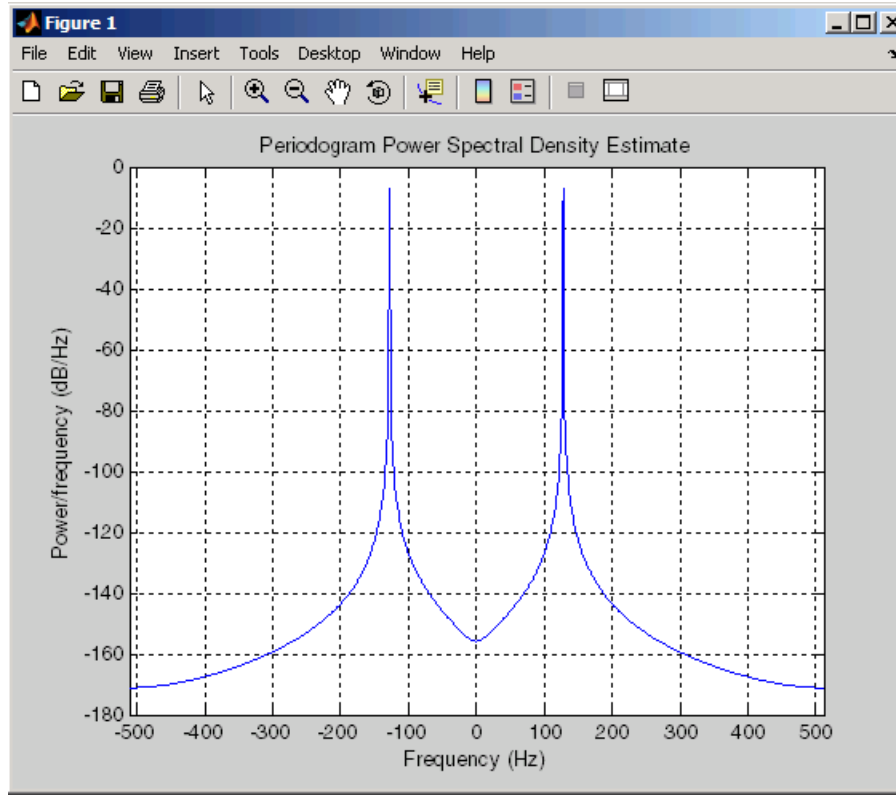
```
Fs = 1024;                      % Sampling frequency
t  = 0:1/Fs:1-(1/Fs);           % Time vector
A  = 1;                         % Peak amplitude
F1 = 128;                       % Hz
x  = A*sin(2*pi*t*F1);          % Sinusoidal signal

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.

To calculate the average power by "integrating" under the power spectral density (PSD) curve, you use the `psd` method on the spectrum object (`hp`) , and then use the `avgpower` method.

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



Notice that the peaks of this plot are not the same height as the mean-square spectrum peaks. 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 you compute the power in the time or frequency domain. You

can verify the estimated average power by summing the signal in the time domain.

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

power_timedomain =

    0.5000
```

Converting this linear value to a logarithmic value, you see that the average power is the same as shown in the mean-square spectrum plot.

```
10*log10(power_freqdomain/2)

ans =

   -6.0206
```

Index

A

aliasing
    sinc functions 2-9
annotation 4-15
ASCII files
    importing 2-11
average power 5-13

B

bandpass filters 4-3
buttons
    filter analysis 4-8

C

cell array 5-8
chirp signals 2-7
cycles
    duty 2-6

D

data
    importing 2-11
    matrices 2-2
    multichannel matrix 2-2
    multichannel signals 2-5
    time vectors 2-4
    vectors 2-2
demos
    spectral analysis 5-3
design a filter 3-4
    filterbuilder 3-10
`dfilt` 4-18
Dirichlet functions
    definition 2-9
dot notation 5-8
duty cycles 2-6

E

Expanding the toolbox 1-4
export 4-17
extensibility 1-4

F

`fdatool` GUI
    analysis buttons 4-8
    group delay 4-8
    impulse response 4-8
    magnitude response 4-8
    phase delay 4-8
    phase response 4-8
    pole-zero plots 4-8
    step response 4-8
files
    M 2-11
    MAT 2-11
    MEX 2-11
filter coefficients
    icon 4-8
filter information
    icon 4-8
filter specification
    icon 4-8
filterbuilder 3-10
filters
    octave-band 4-3
`fopen` function 2-11
`fread` function 2-11
functions
    Dirichlet 2-9
    sinc 2-8

G

`gauspuls` function
    pulse trains 2-8
getting started 3-2

getting started example 3-2
graphical user interface (GUI) 1-3
    See also `sptool` GUI, `fdatool` GUI, `wintool` GUI, `fvtool` GUI, `wvtool` GUI
group delay
    icon 4-8

I

import
    data 2-11
impulse 2-5
impulse response
    icon 4-8
inverse Fourier transforms
    See `sinc` function 2-8

M

magnitude and phase response
    icon 4-8
magnitude response
    icon 4-8
MAT-files
    converting to 2-11
    importing 2-11
matrices
    data 2-3
matrix 2-2
MEX-files 2-11
multichannel data 2-5

N

noise data 4-19

O

octave-band filters 4-3
options object 5-11

P

P-V pairs 5-7
periodogram 5-13
phase delay
    icon 4-8
phase response
    icon 4-8
pole-zero plot
    icon 4-8
power spectral density 5-2
power spectrum 5-2
property values
    changing 5-8
property-value pairs 5-7
pulse trains
    example 2-7
    `pulstran` function 2-7

R

ramp 2-5
references
    DSP 2-13

S

`sawtooth` function
    example 2-6
sawtooth wave 2-6
set 5-8
signal
    average power 5-13
signals
    adding noise 2-4
    aperiodic 2-6
    chirp 2-7
    `diric` function 2-9
    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
spectrum
creating 5-6
mean-square spectrum 5-2
options object 5-11
PSD 5-2
pseudo spectrum 5-2
`sptool` GUI
data entering 2-11
`square` function
example 2-6
square wave
See `square` function 2-6
step 2-5
step response
icon 4-8

T

time vectors 2-4
toolbox
getting started 3-2
tools
interactive GUIs 1-3

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
white noise 2-4