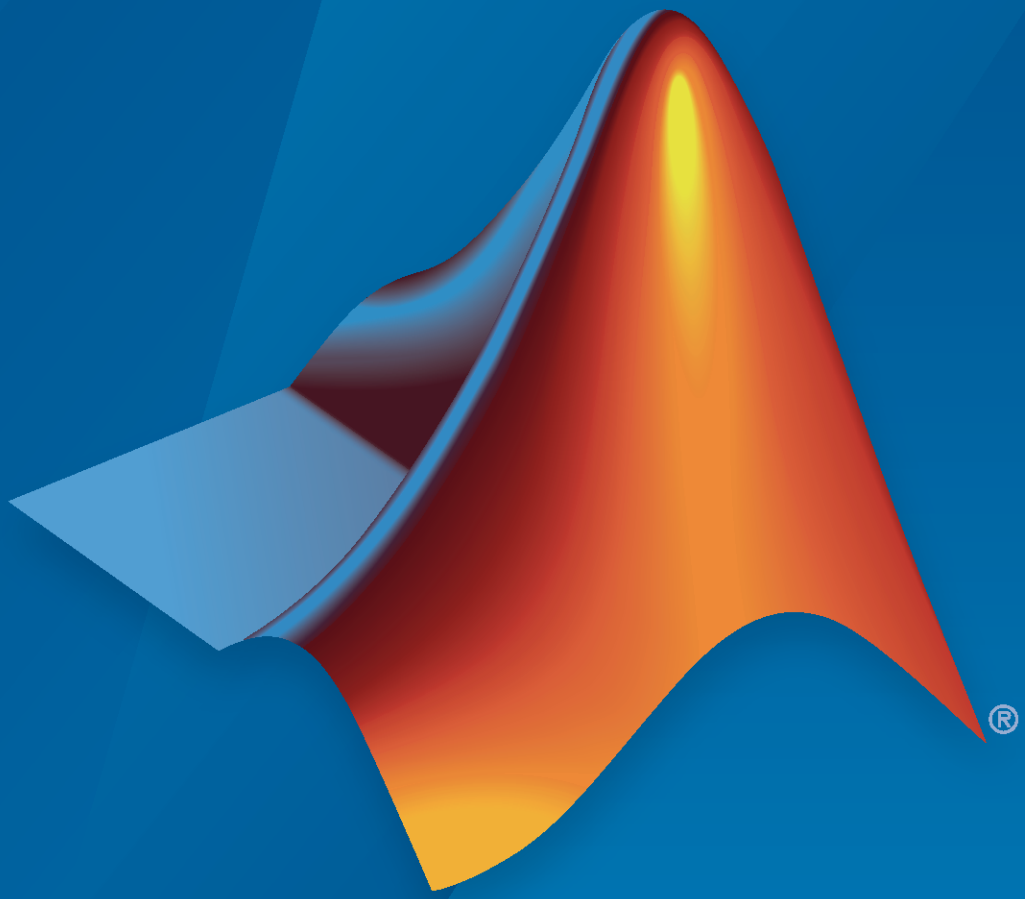
Signal Processing Toolbox™

Getting Started Guide



MATLAB®

R2022b

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

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

Signal Processing Toolbox™ Getting Started Guide

© COPYRIGHT 2006–2022 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)
March 2013	Online only	Revised for Version 6.19 (Release 2013a)
September 2013	Online only	Revised for Version 6.20 (Release 2013b)
March 2014	Online only	Revised for Version 6.21 (Release 2014a)
October 2014	Online only	Revised for Version 6.22 (Release 2014b)
March 2015	Online only	Revised for Version 7.0 (Release 2015a)
September 2015	Online only	Revised for Version 7.1 (Release 2015b)
March 2016	Online only	Revised for Version 7.2 (Release 2016a)
September 2016	Online only	Revised for Version 7.3 (Release 2016b)
March 2017	Online only	Revised for Version 7.4 (Release 2017a)
September 2017	Online only	Revised for Version 7.5 (Release 2017b)
March 2018	Online only	Revised for Version 8.0 (Release 2018a)
September 2018	Online only	Revised for Version 8.1 (Release 2018b)
March 2019	Online only	Revised for Version 8.2 (Release 2019a)
September 2019	Online only	Revised for Version 8.3 (Release 2019b)
March 2020	Online only	Revised for Version 8.4 (Release 2020a)
September 2020	Online only	Revised for Version 8.5 (Release 2020b)
March 2021	Online only	Revised for Version 8.6 (Release 2021a)
September 2021	Online only	Revised for Version 8.7 (Release 2021b)
March 2022	Online only	Revised for Version 9.0 (Release 2022a)
September 2022	Online only	Revised for Version 9.1 (Release 2022b)

Contents

1 Overview

Signal Processing Toolbox Product Description **1-2**

2 Basic Signal Processing Concepts

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

Waveform Generation: Time Vectors and Sinusoids **2-3**

Impulse, Step, and Ramp Functions **2-4**

Common Periodic Waveforms **2-6**

Common Aperiodic Waveforms **2-9**

The pulstran Function **2-11**

The Sinc Function **2-13**

The Dirichlet Function **2-14**

Working with Data **2-16**
Data Precision **2-16**

Selected Bibliography **2-17**

3 Design a Filter with fdesign and Filter Builder

Filter Design Process Overview **3-2**

Design a Filter Using fdesign **3-3**

Design a Filter Using Filter Builder **3-7**

4 Filter Design with the Filter Designer App

Introduction **4-2**

Designing the Filter **4-3**

Analyzing the Filter **4-6**

Designing Additional Filters **4-8**

Viewing and Annotating the Filter **4-9**
Viewing the Filter in FVTool **4-9**
Using FVTool for Annotation **4-12**

Exporting Filters from Filter Designer **4-13**
Filtering with dfilt **4-14**

Overview

Signal Processing Toolbox Product Description

Perform signal processing and analysis

Signal Processing Toolbox provides functions and apps to manage, analyze, preprocess, and extract features from uniformly and nonuniformly sampled signals. The toolbox includes tools for filter design and analysis, resampling, smoothing, detrending, and power spectrum estimation. You can use the Signal Analyzer app for visualizing and processing signals simultaneously in time, frequency, and time-frequency domains. With the Filter Designer app you can design and analyze FIR and IIR digital filters. Both apps generate MATLAB® scripts to reproduce or automate your work.

Using toolbox functions, you can prepare signal datasets for AI model training by engineering features that reduce dimensionality and improve the quality of signals. You can access and process collections of files and large datasets using signal datastores. With the Signal Labeler app, you can annotate signal attributes, regions, and points of interest to create labeled signal sets. The toolbox supports GPU acceleration in addition to C/C++ and CUDA® code generation for desktop prototyping and embedded system deployment.

Basic Signal Processing Concepts

- “Representing Signals” on page 2-2
- “Waveform Generation: Time Vectors and Sinusoids” on page 2-3
- “Impulse, Step, and Ramp Functions” on page 2-4
- “Common Periodic Waveforms” on page 2-6
- “Common Aperiodic Waveforms” on page 2-9
- “The pulstran Function” on page 2-11
- “The Sinc Function” on page 2-13
- “The Dirichlet Function” on page 2-14
- “Working with Data” on page 2-16
- “Selected Bibliography” on page 2-17

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';
```

```
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]
```

```
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]; % 1-by-4 vector
x = x.';                   % 4-by-1 vector
```

Waveform Generation: Time Vectors and Sinusoids

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)';
```

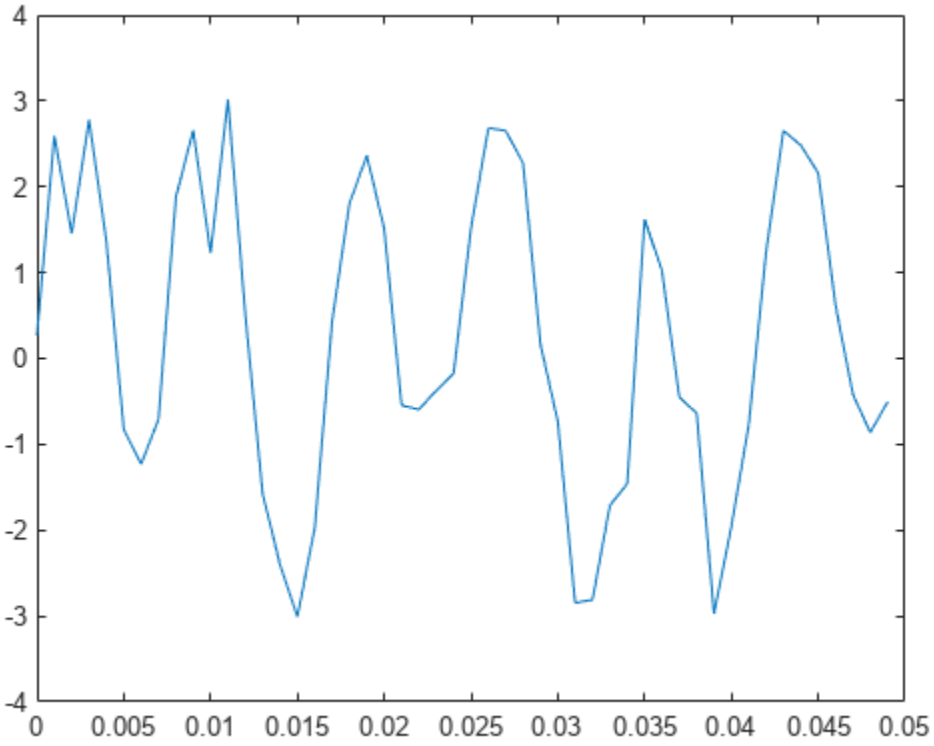
where the MATLAB® colon operator (`:`) creates a 1001-element row vector that represents time running from 0 to 1 seconds in steps of 1 ms. The transpose operator (`'`) changes the row vector into a column; the semicolon (`;`) tells MATLAB to compute, but not display, the result.

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:

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



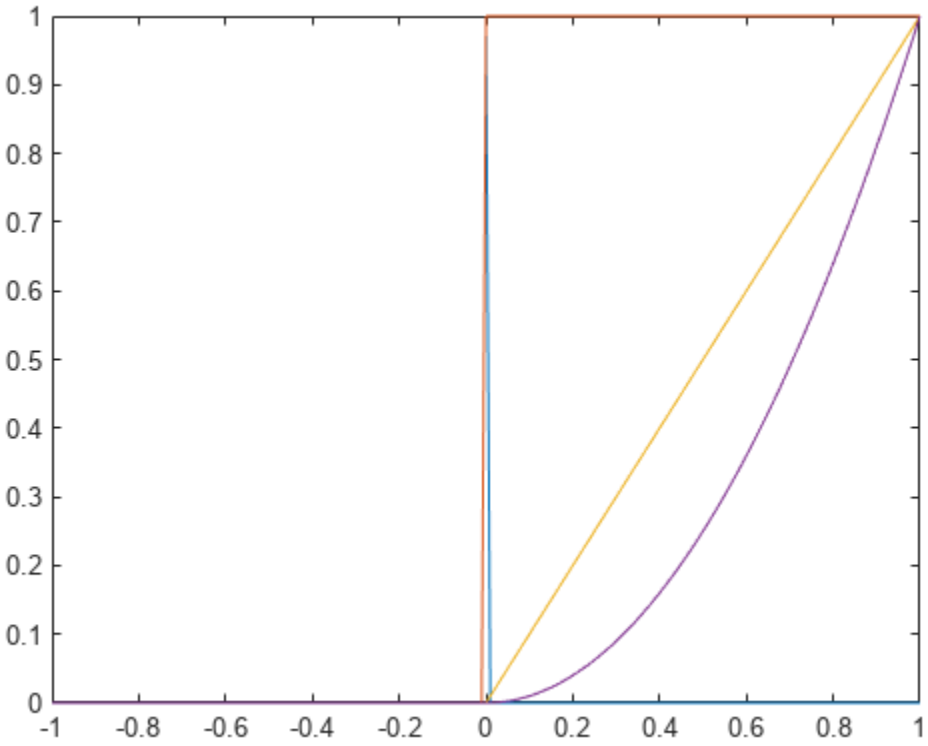
Impulse, Step, and Ramp Functions

Since MATLAB® is a programming language, an endless variety of different signals is possible. Here are some statements that generate a unit impulse, a unit step, a unit ramp, and a unit parabola.

```
t = (-1:0.01:1)';

impulse = t==0;
unitstep = t>=0;
ramp = t.*unitstep;
quad = t.^2.*unitstep;
```

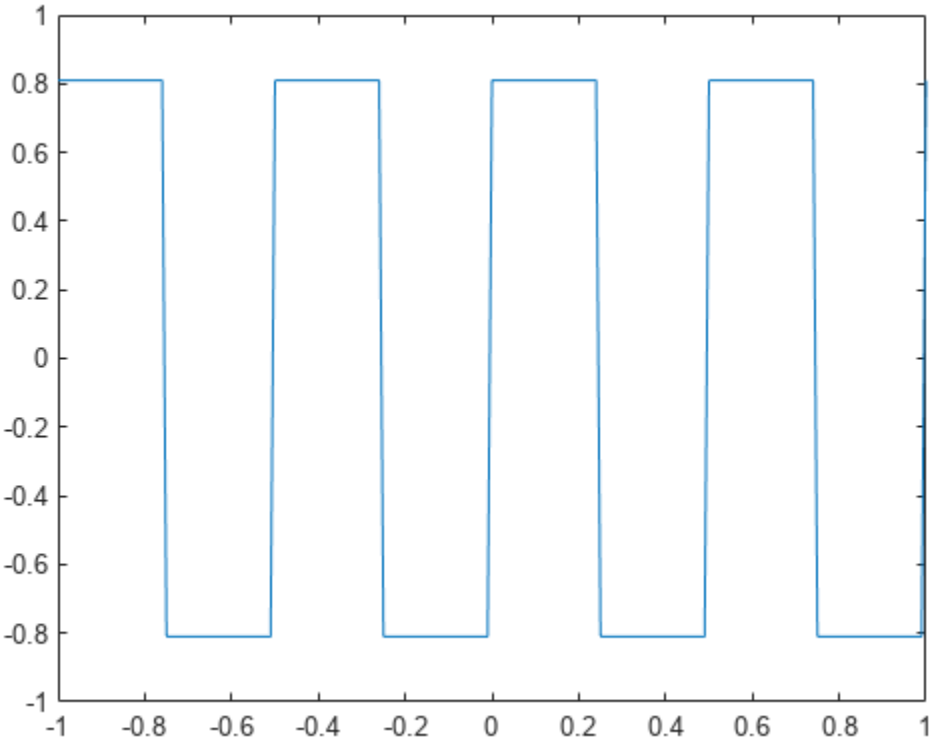
All of these sequences are column vectors that inherit their shapes from `t`. Plot the sequences.

```
plot(t,[impulse unitstep ramp quad])
```



Generate and plot a square wave with period 0.5 and amplitude 0.81.

```
sqwave = 0.81*square(4*pi*t);
plot(t,sqwave)
```


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

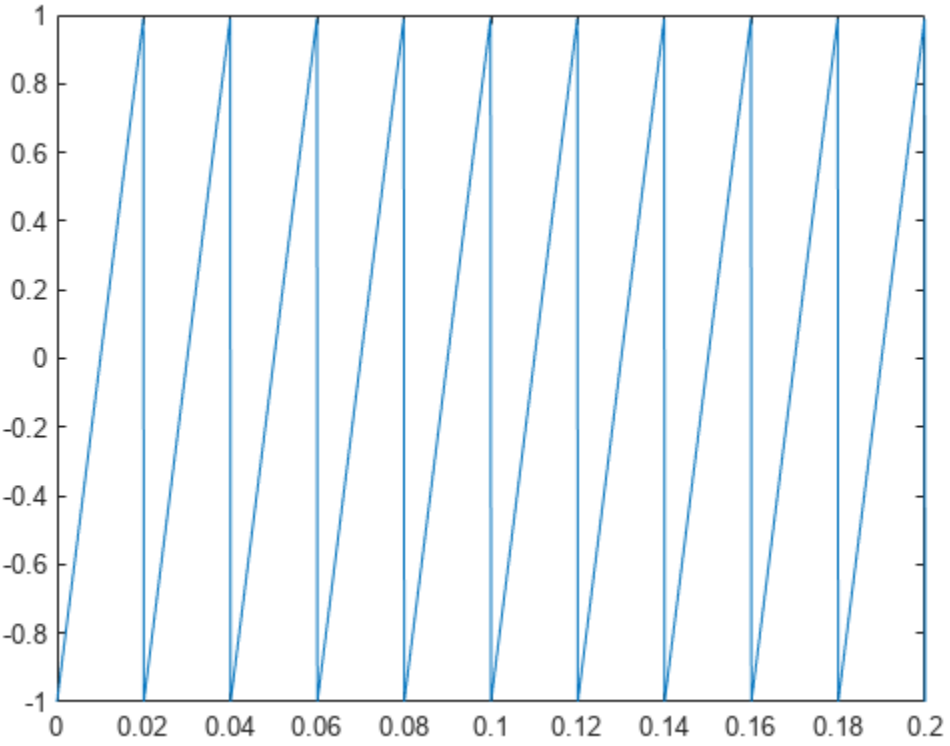
Common Periodic Waveforms

Signal Processing 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 maximum occurs.
- `square` generates a square wave with a period of 2π. An optional parameter specifies the *duty cycle*, the percent of the period for which the signal is positive.

Generate 1.5 seconds of a 50 Hz sawtooth wave with a sample rate of 10 kHz. Plot 0.2 seconds of the generated waveform.

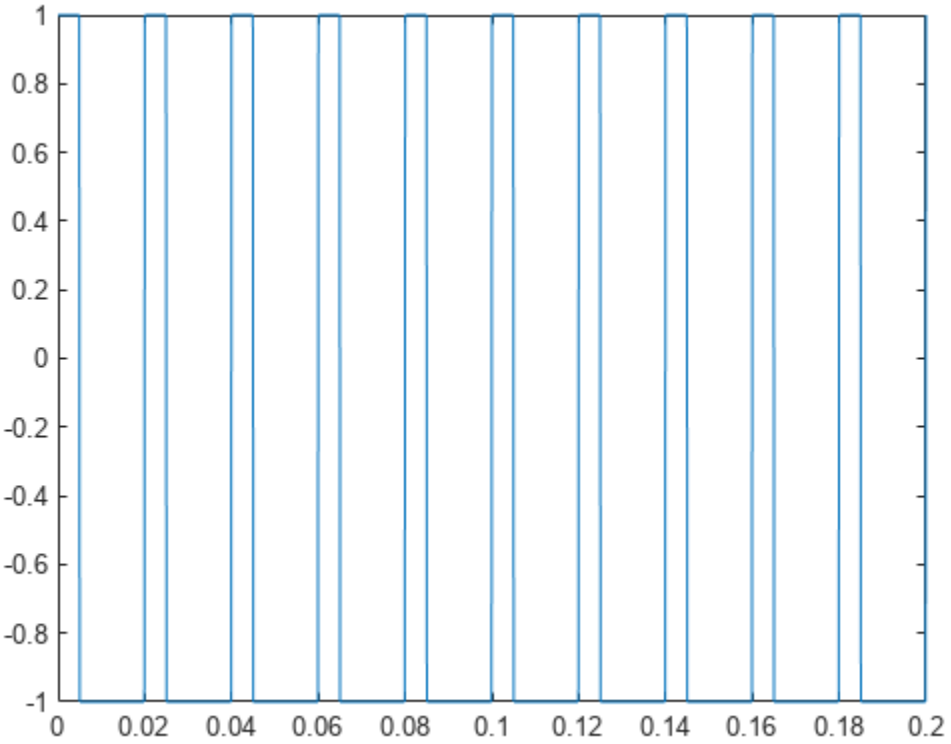
```
fs = 10e3;
t = 0:1/fs:1.5;
x = sawtooth(2*pi*50*t);

plot(t,x)
axis([0 0.2 -1 1])
```



Generate 1.5 seconds of a 50 Hz square wave with a sample rate of 10 kHz. Specify a duty cycle of 25%. Plot 0.2 seconds of the generated waveform.

```
fs = 10e3;
t = 0:1/fs:1.5;
x = square(2*pi*50*t,25);
```

```
plot(t,x)
axis([0 0.2 -1 1])
```

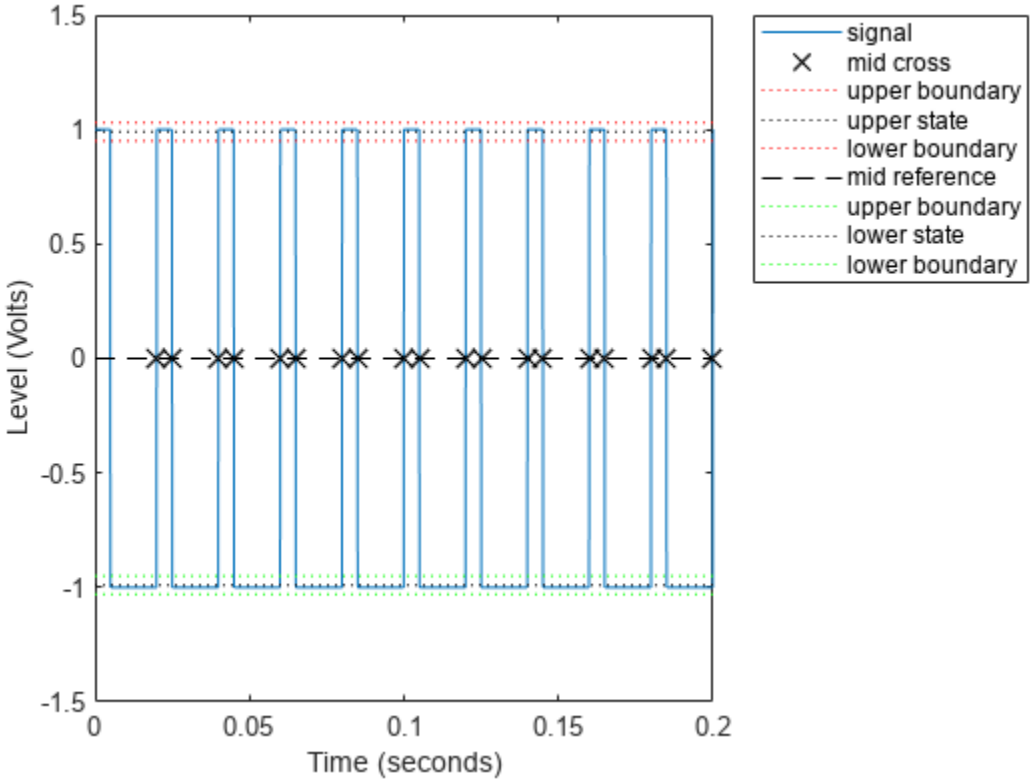


Use the `dutycycle` function to verify that the duty cycle of the square wave is the specified value. Use the function with no output arguments to plot the waveform, the location of the mid-reference level instants, the associated reference levels, the state levels, and the associated lower and upper state boundaries.

```
dc = dutycycle(x,fs);
dc = dc(1)
```

```
dc = 0.2500
```

```
dutycycle(x,fs);
xlim([0 0.2])
```



See Also

`dutycycle` | `sawtooth` | `square`

Common Aperiodic Waveforms

Signal Processing Toolbox™ provides functions for generating several widely used aperiodic waveforms.

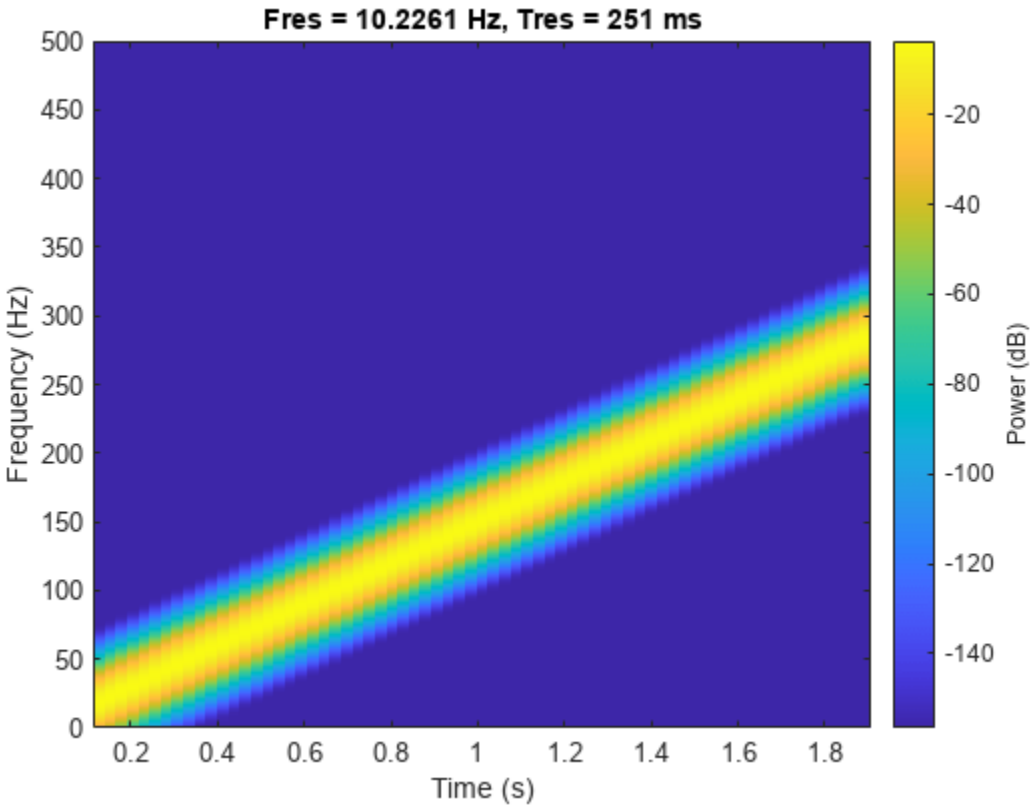
- `gauspuls` generates a Gaussian-modulated sinusoidal pulse with a specified time, center frequency, and fractional bandwidth. Optional parameters return in-phase and quadrature pulses, the RF signal envelope, and the cutoff time for the trailing pulse envelope.
- `chirp` generates a linear, logarithmic, or quadratic swept-frequency sinusoidal signal. An optional parameter specifies alternative sweep methods. An optional parameter allows an initial phase to be specified in degrees.

Compute 2 seconds of a linear chirp signal with a sample rate of 1 kHz that starts at DC and crosses 150 Hz at 1 second.

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

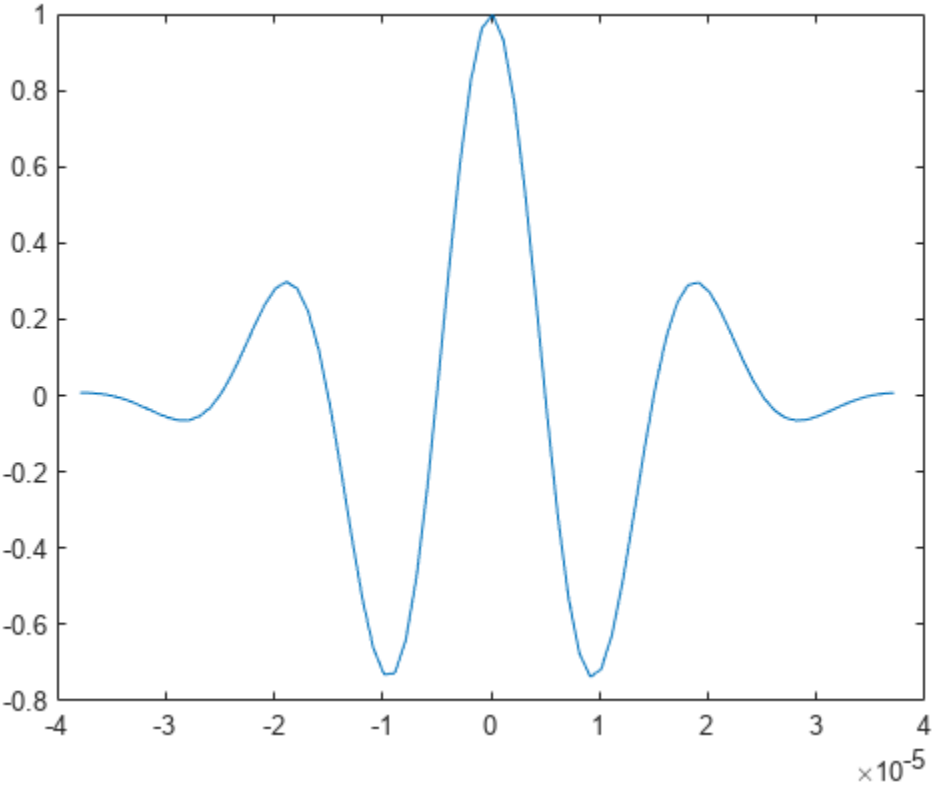
Plot the spectrogram of the chirp. Specify 90% of overlap between adjoining windowed segments.

```
pspectrum(y,t,'spectrogram','OverlapPercent',90)
```



Use `gauspuls` to plot a 50 kHz Gaussian RF pulse with 60% bandwidth, sampled at a rate of 1 MHz. Truncate the pulse where the envelope falls 40 dB below the peak.

```
tc = gauspuls('cutoff',50e3,0.6,[],-40);
t = -tc : 1e-6 : tc;
yi = gauspuls(t,50e3,0.6);
plot(t,yi)
```



See Also

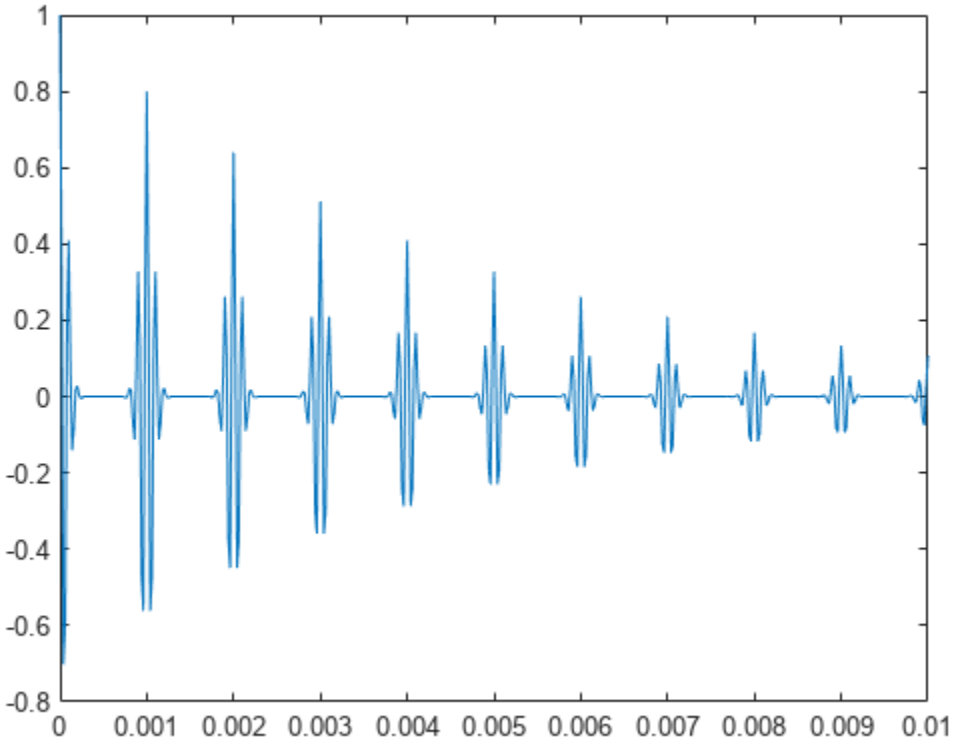
chirp | gauspuls | pspectrum

The pulstran Function

The `pulstran` function generates pulse trains from either continuous or sampled prototype pulses. This 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. `T` specifies the time instants at which the pulse train is sampled. `D` specifies the delay to each pulse repetition in the first column and an optional attenuation for each repetition in the second column. To construct the pulse train, pass 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)
```



See “Compute Envelope Spectrum of Vibration Signal” for an example that uses the `pulstran` function to generate vibration data for bearing analysis.

See Also

`pulstran`

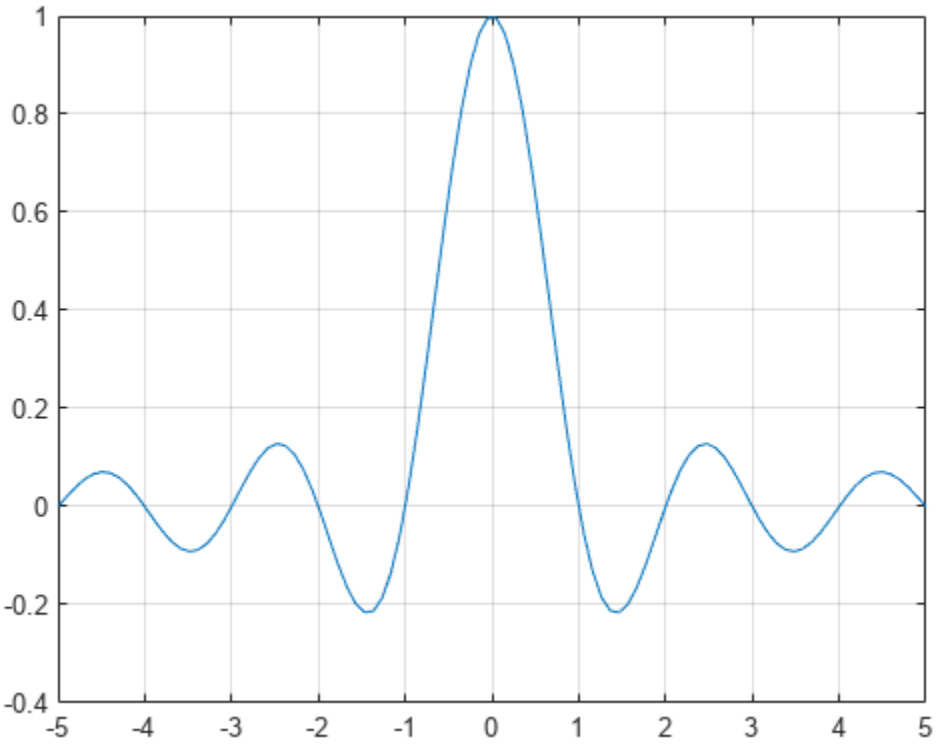
The Sinc Function

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, with width 2π and unit height:

$$\operatorname{sinc} x = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{j\omega x}\, d\omega = \begin{cases} \dfrac{\sin \pi x}{\pi x}, & x \neq 0, \\ 1, & x = 0\,. \end{cases}$$

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

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



See Also

`diric` | `sinc`

The Dirichlet Function

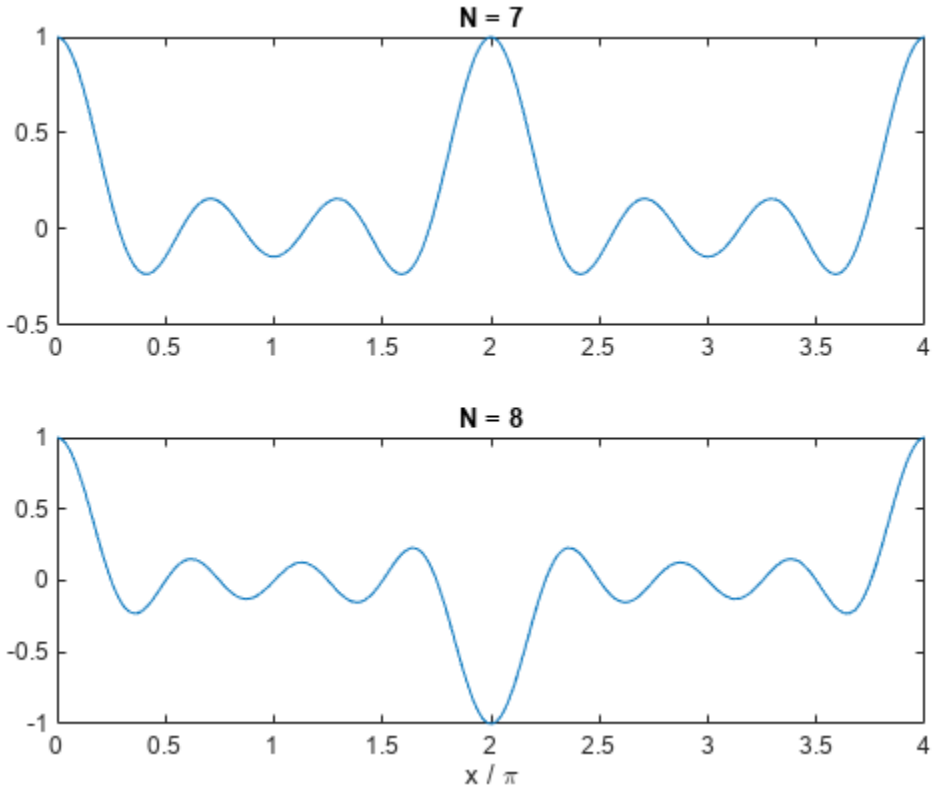
The function `diric` computes the Dirichlet function, sometimes called the *periodic sinc* or *aliased sinc* function, for an input vector or matrix `x`. The Dirichlet function is defined by

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

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 between 0 and 4π for $N = 7$ and $N = 8$, use

```
x = linspace(0,4*pi,300);

subplot(2,1,1)
plot(x/pi,diric(x,7))
title('N = 7')

subplot(2,1,2)
plot(x/pi,diric(x,8))
title('N = 8')
xlabel('x / \pi')
```



See Also

`diric` | `sinc`

Working with Data

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 Designer™ 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 Lawrence 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, Steven M. *Modern Spectral Estimation*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[5] Oppenheim, Alan V., and Ronald W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[6] Parks, Thomas W., and C. Sidney 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, John G., and Dimitris G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*. Upper Saddle River, NJ: Prentice Hall, 1996.

[10] Rabiner, Lawrence R., and Bernard 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® Transactions on Audio and Electroacoustics*. Vol. AU-15, 1967. pp. 70–73.

Design a Filter with fdesign and Filter Builder

- “Filter Design Process Overview” on page 3-2
- “Design a Filter Using fdesign” on page 3-3
- “Design a Filter Using Filter Builder” on page 3-7

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.

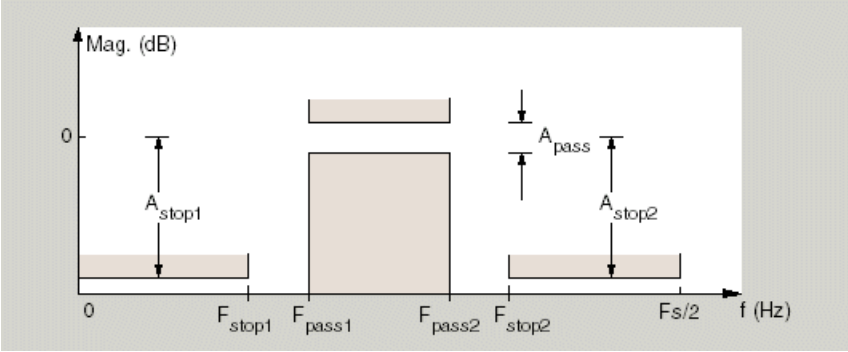
Design a Filter Using fdesign

Use the following two steps to design a simple filter.

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

Example 3.1. 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

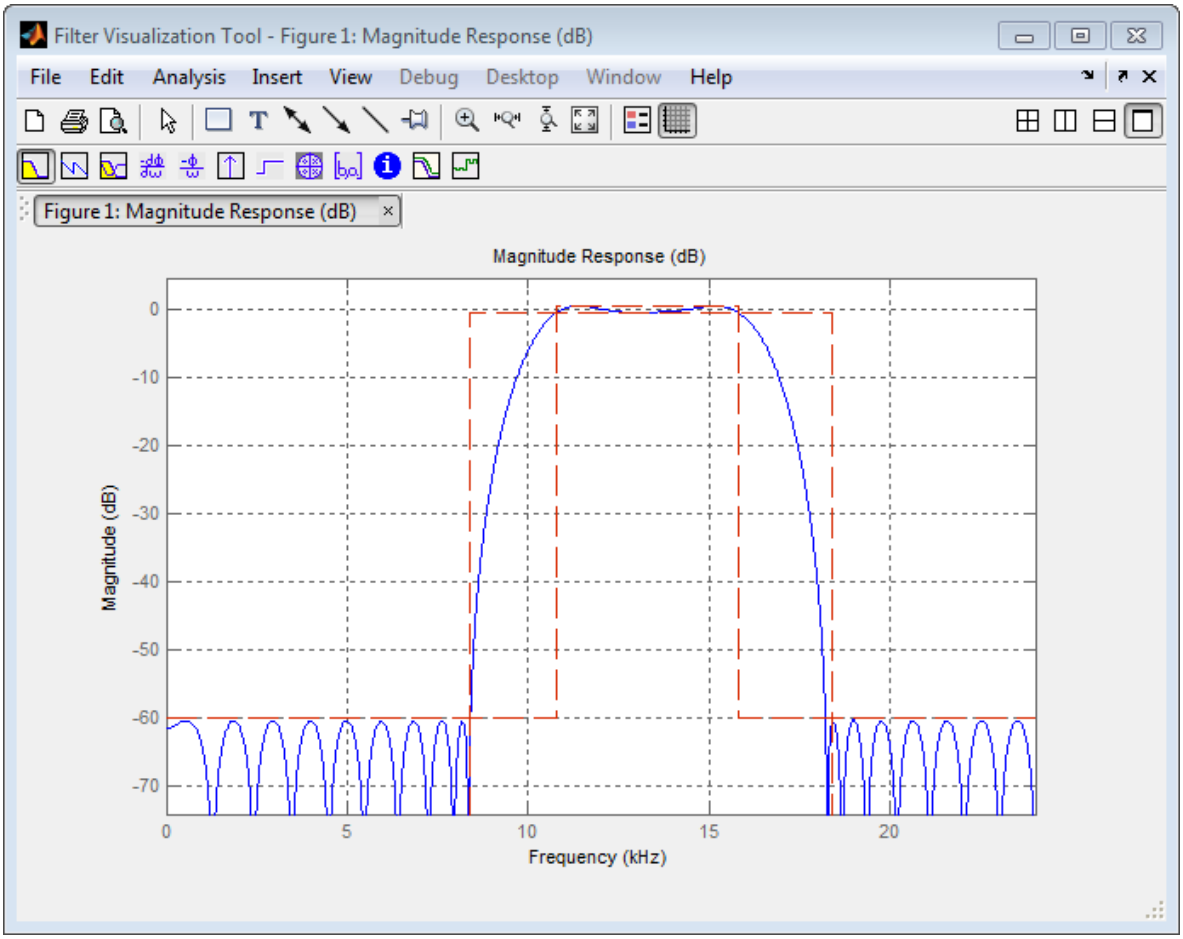
```
>> BandPassFilt = design(BandPassSpecObj)

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 Filter Builder

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

Example 3.2. Design a Simple Filter in Filter Builder

To design the filter using the Filter Builder GUI:

1. Type the following at the MATLAB prompt:

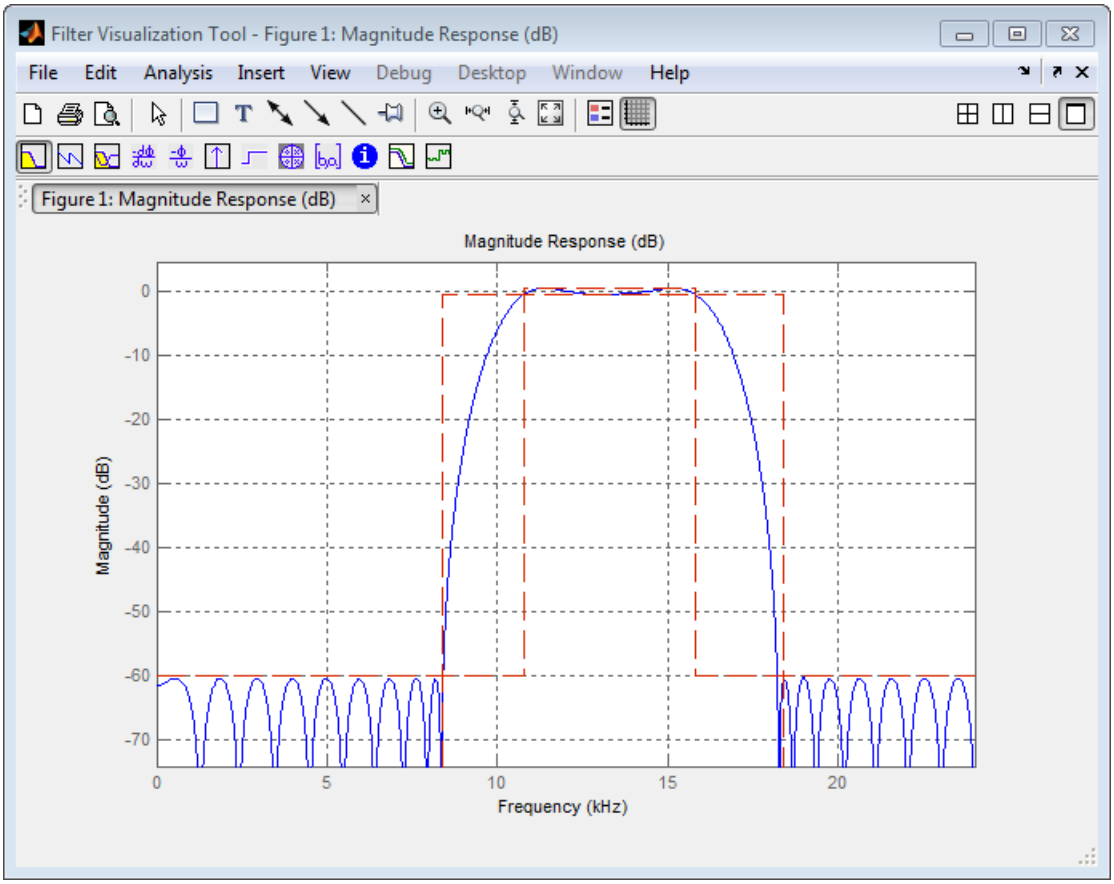
   ```
   filterBuilder
   ```

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.

Filter Design with the Filter Designer App

- “Introduction” on page 4-2
- “Designing the Filter” on page 4-3
- “Analyzing the Filter” on page 4-6
- “Designing Additional Filters” on page 4-8
- “Viewing and Annotating the Filter” on page 4-9
- “Exporting Filters from Filter Designer” on page 4-13

Introduction

This section describes how to graphically design and implement digital filters using 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.

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

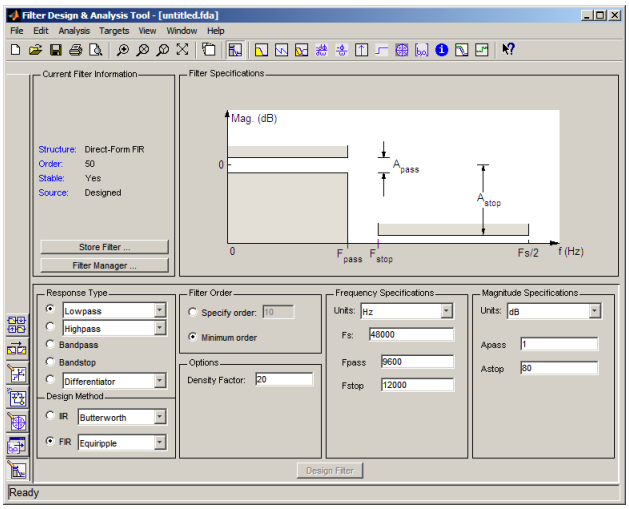
Designing the Filter

This section is a step-by-step introduction to using the **Filter Designer** app 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 the app from the MATLAB command line.

   ```
   filterDesigner
   ```

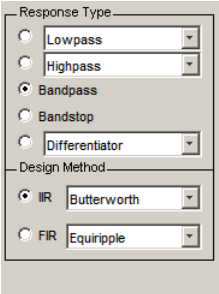
   The app opens with a default filter. Its filter information is summarized in the upper left (**Current Filter Information**) and its filter specifications are depicted in the upper right. In addition to displaying filter specification, this upper right pane displays filter responses and filter coefficients.



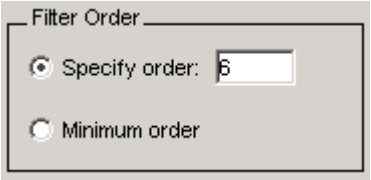
   The bottom half of the app shows the Filter Design panel, where you specify the filter parameters. Other panels, such as Import filter from workspace and Pole/Zero Editor, which you access with the buttons on the lower left, are also displayed in this area. If you have other products installed, you may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

   Note that when you open the app, **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.

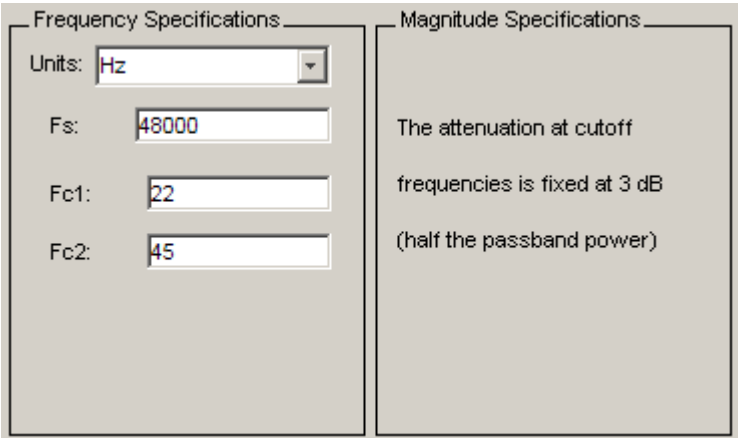


4 For the Filter Order, select **Specify order**, and then enter 6.

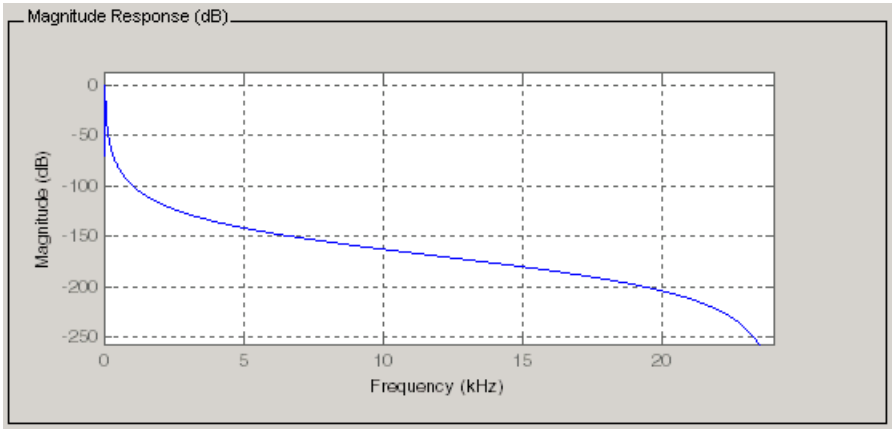


5 Set the Frequency Specifications as follows:

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

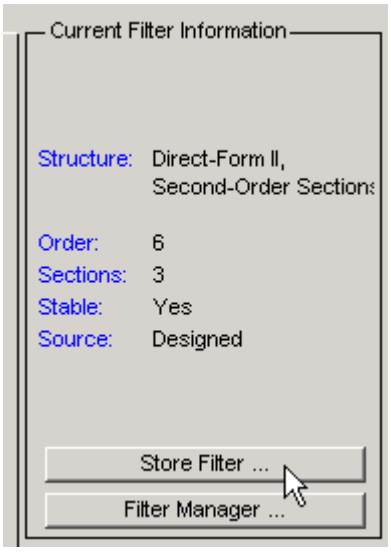


6 After specifying the filter design parameters, click the **Design Filter** button at the bottom of the design panel to compute the filter coefficients. The display updates to show the magnitude response of the designed filter.



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.

`

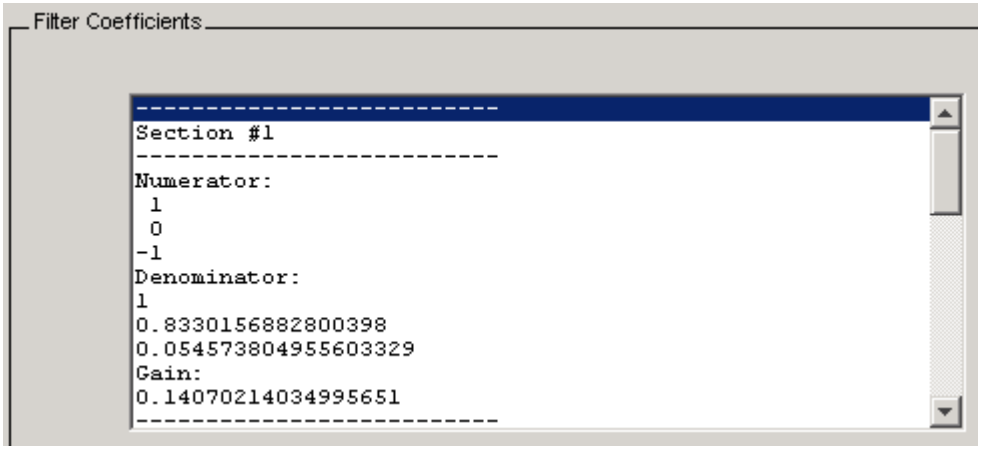
Analyzing the Filter

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.


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.

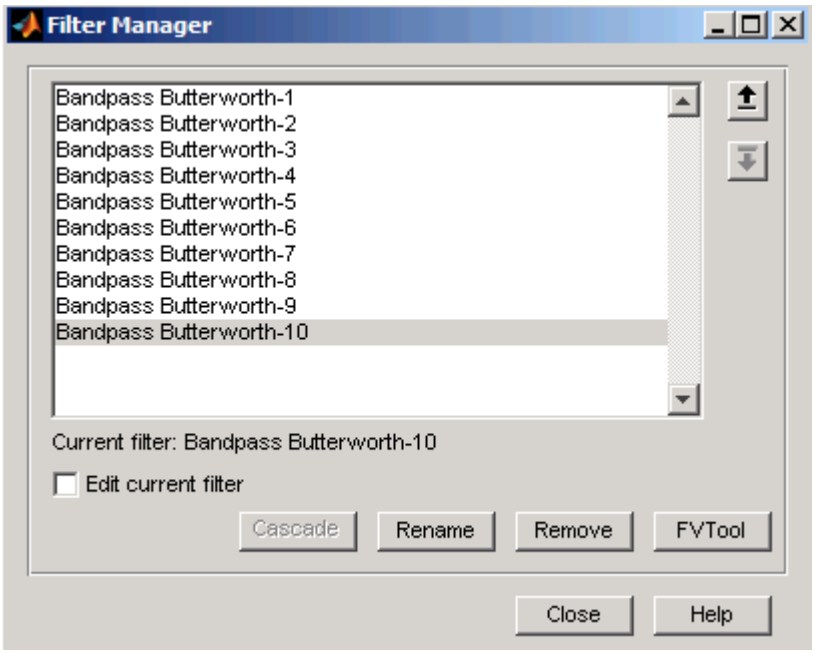
Viewing and Annotating the Filter

In this section...
“Viewing the Filter in FVTool” on page 4-9 “Using FVTool for Annotation” on page 4-12

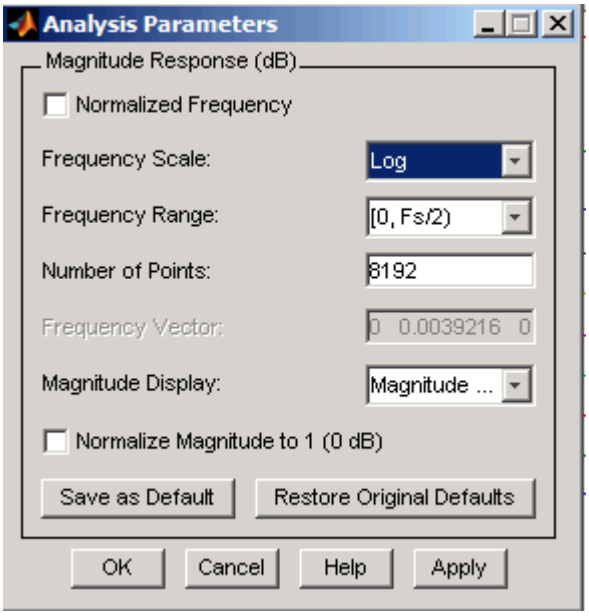
Viewing the Filter in FVTool

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.

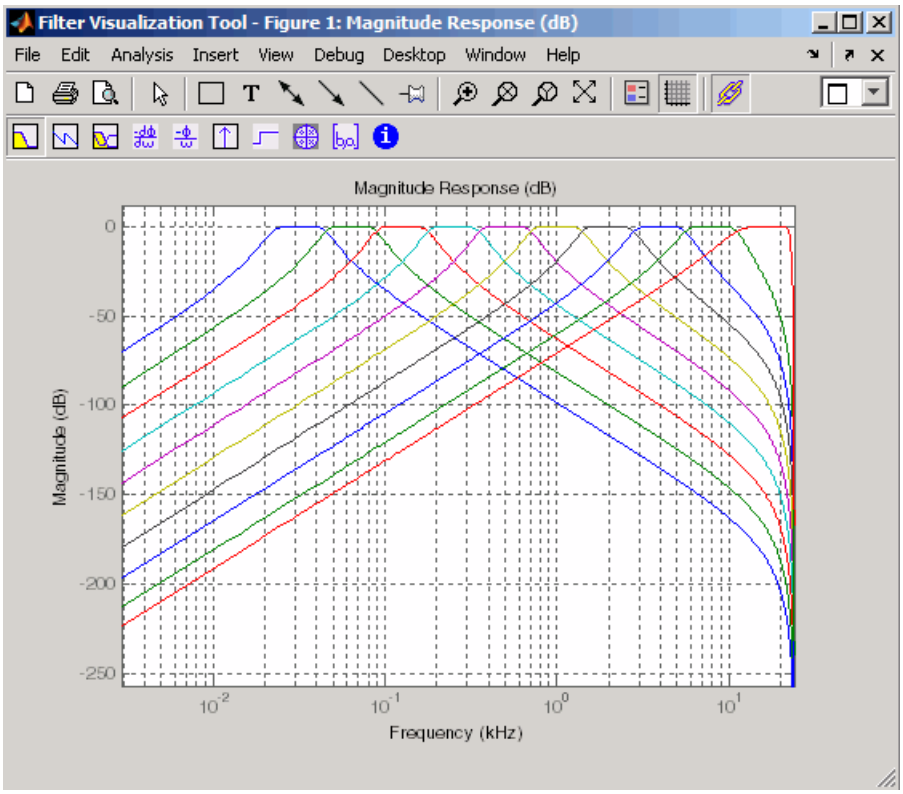
1. Click the **Filter Manager** button to display the Filter Manager, which lists your saved filters.



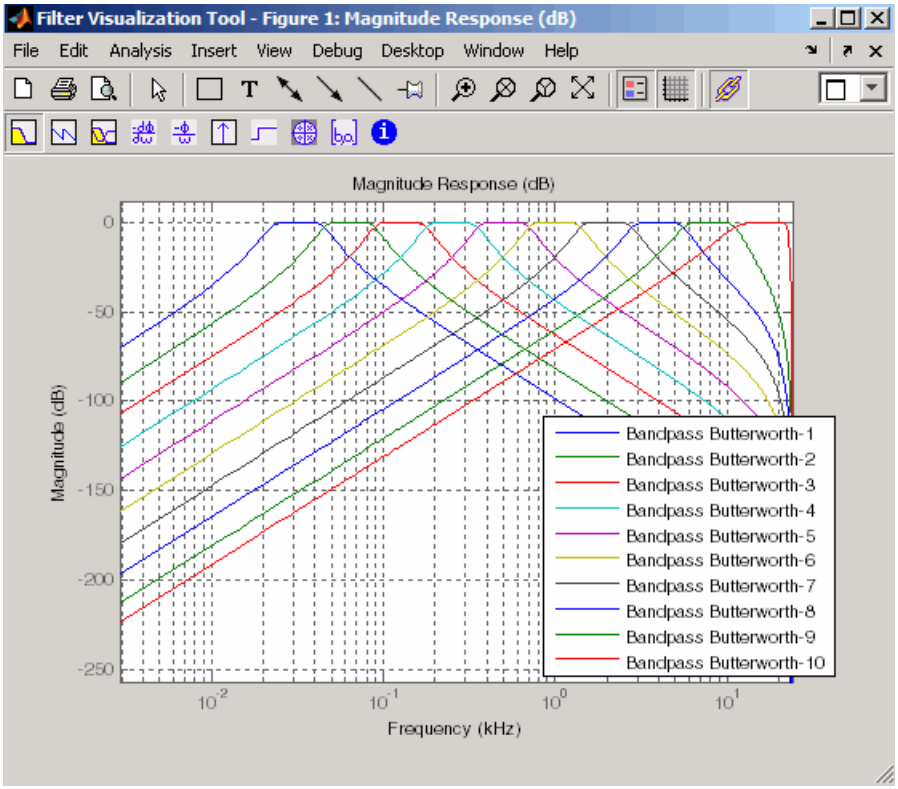
2. Press Ctrl+click on each filter name to select all the filters, and then click **FVTool**. FVTool opens with the filter responses overlaid for easy comparison. (If you want to view a single filter in FVTool, click the **Full View Analysis** button when that filter is shown in the app’s display panel 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**.

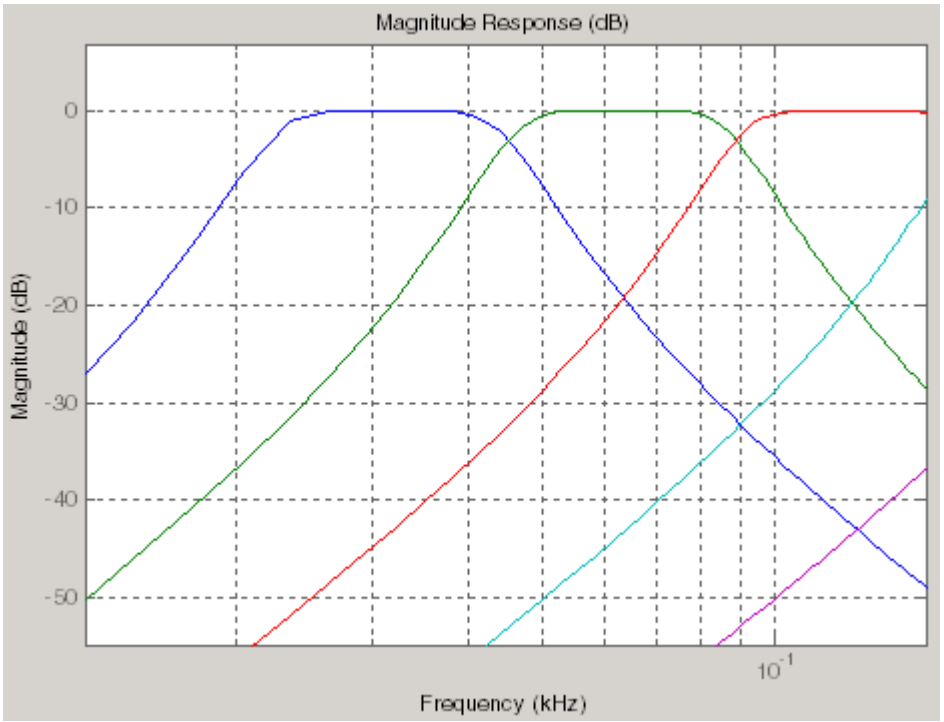


6 Click the **Legend** button to turn on the legend, which you can drag to the desired location.



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 **Filter Designer**. See “Analyzing the Filter” on page 4-6 for descriptions of the buttons.

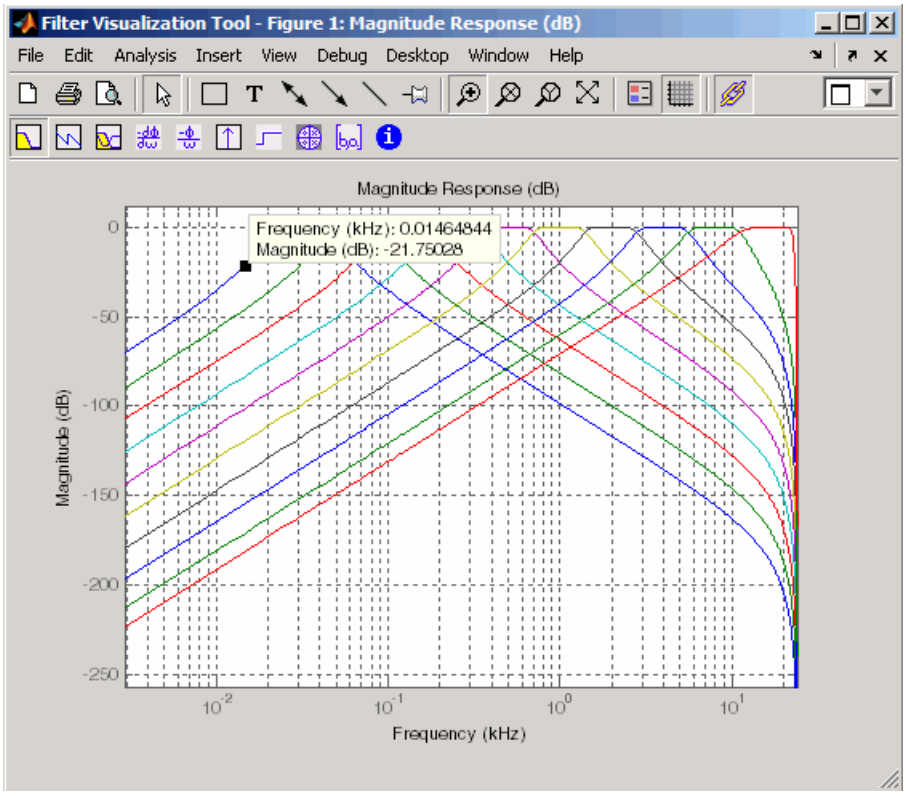
Using FVTool for Annotation

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 **Filter Designer** 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.

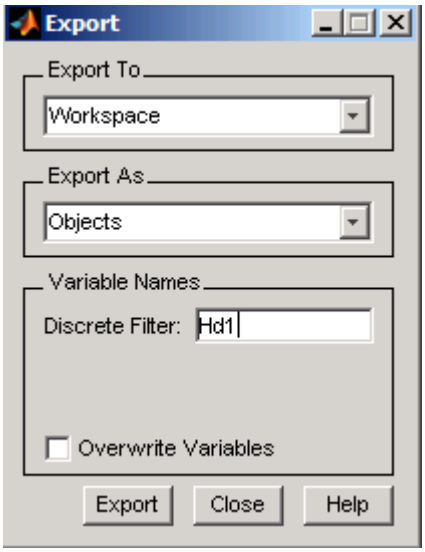


3 Close FVTool by selecting **File** > **Close**.

Exporting Filters from Filter Designer

The **Filter Designer** app 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 **Filter Designer**, 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 the app 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 4-9.

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)
```

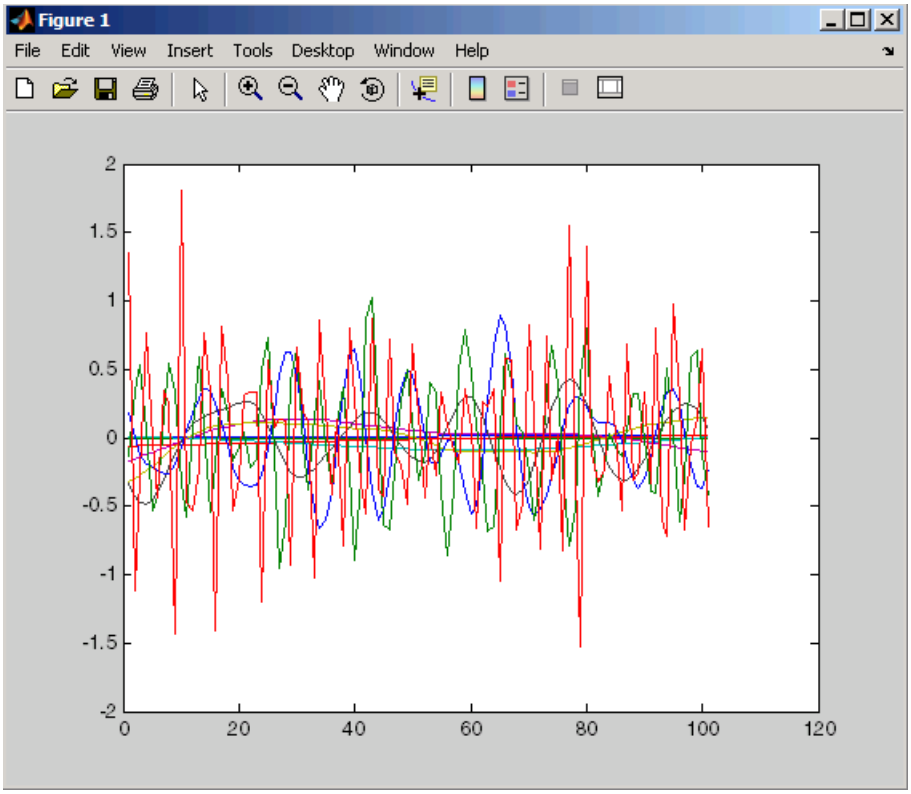


Figure 1
File Edit View Insert Tools Desktop Window Help
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
20
40
60
80
100
120