

FIR Filter Design

In this section...
“FIR vs. IIR Filters” on page 1-17
“FIR Filter Summary” on page 1-18
“Linear Phase Filters” on page 1-18
“Windowing Method” on page 1-19
“Multiband FIR Filter Design with Transition Bands” on page 1-24
“Constrained Least Squares FIR Filter Design” on page 1-31
“Arbitrary-Response Filter Design” on page 1-37

FIR vs. IIR Filters

Digital filters with finite-duration impulse response (all-zero, or FIR filters) have both advantages and disadvantages compared to infinite-duration impulse response (IIR) filters.

FIR filters have the following primary advantages:

- They can have exactly linear phase.
- They are always stable.
- The design methods are generally linear.
- They can be realized efficiently in hardware.
- The filter startup transients have finite duration.

The primary disadvantage of FIR filters is that they often require a much higher filter order than IIR filters to achieve a given level of performance. Correspondingly, the delay of these filters is often much greater than for an equal performance IIR filter.

FIR Filter Summary

FIR Filters

Filter Design Method	Description	Filter Functions
Windowing	Apply window to truncated inverse Fourier transform of desired “brick wall” filter	fir1, fir2, kaiserord
Multiband with Transition Bands	Equiripple or least squares approach over sub-bands of the frequency range	firls, firpm, firpmord
Constrained Least Squares	Minimize squared integral error over entire frequency range subject to maximum error constraints	fircls, fircls1
Arbitrary Response	Arbitrary responses, including nonlinear phase and complex filters	cfirpm
Raised Cosine	Lowpass response with smooth, sinusoidal transition	firrcos

Linear Phase Filters

Except for `cfirpm`, all of the FIR filter design functions design linear phase filters only. The filter coefficients, or “taps,” of such filters obey either an even or odd symmetry relation. Depending on this symmetry, and on whether the order n of the filter is even or odd, a linear phase filter (stored in length $n+1$ vector b) has certain inherent restrictions on its frequency response.

Linear Phase Filter Type	Filter Order	Symmetry of Coefficients	Response $H(f)$, $f = 0$	Response $H(f)$, $f = 1$ (Nyquist)
Type I	Even	even: $b(k) = b(n+2-k)$, $k = 1, \dots, n+1$	No restriction	No restriction
Type II	Odd	No restriction	$H(1) = 0$	

Linear Phase Filter Type	Filter Order	Symmetry of Coefficients	Response $H(f)$, $f = 0$	Response $H(f)$, $f = 1$ (Nyquist)
Type III	Even	odd: $b(k) = -b(n+2-k)$, $k = 1, \dots, n+1$	$H(0) = 0$	$H(1) = 0$
Type IV	Odd	$H(0) = 0$	No restriction	

The phase delay and group delay of linear phase FIR filters are equal and constant over the frequency band. For an order n linear phase FIR filter, the group delay is $n/2$, and the filtered signal is simply delayed by $n/2$ time steps (and the magnitude of its Fourier transform is scaled by the filter's magnitude response). This property preserves the wave shape of signals in the passband; that is, there is no phase distortion.

The functions `fir1`, `fir2`, `firls`, `firpm`, `fircls`, `fircls1`, and `firrcos` all design type I and II linear phase FIR filters by default. Both `firls` and `firpm` design type III and IV linear phase FIR filters given a 'hilbert' or 'differentiator' flag. `cfirpm` can design any type of linear phase filter, and nonlinear phase filters as well.

Note Because the frequency response of a type II filter is zero at the Nyquist frequency ("high" frequency), `fir1` does not design type II highpass and bandstop filters. For odd-valued n in these cases, `fir1` adds 1 to the order and returns a type I filter.

Windowing Method

Consider the ideal, or "brick wall," digital lowpass filter with a cutoff frequency of ω_0 rad/s. This filter has magnitude 1 at all frequencies with magnitude less than ω_0 , and magnitude 0 at frequencies with magnitude between ω_0 and π . Its impulse response sequence $h(n)$ is

$$h(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H(\omega) e^{j\omega n} d\omega = \frac{1}{2\pi} \int_{-\omega_0}^{\omega_0} e^{j\omega n} d\omega = \frac{\omega_0}{\pi} \operatorname{sinc}\left(\frac{\omega_0}{\pi} n\right)$$

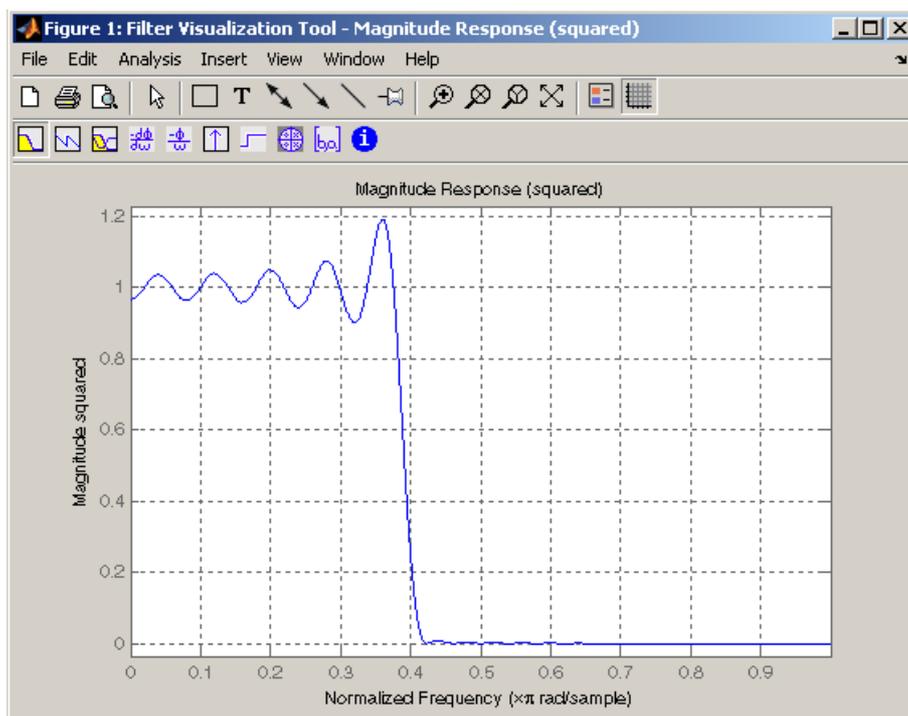
This filter is not implementable since its impulse response is infinite and noncausal. To create a finite-duration impulse response, truncate it by applying a window. By retaining the central section of impulse response in this truncation, you obtain a linear phase FIR filter. For example, a length 51 filter with a lowpass cutoff frequency ω_0 of 0.4π rad/s is

```
b = 0.4*sinc(0.4*(-25:25));
```

The window applied here is a simple rectangular window. By Parseval's theorem, this is the length 51 filter that best approximates the ideal lowpass filter, in the integrated least squares sense. The following command displays the filter's frequency response in FVTool:

```
fvtool(b,1)
```

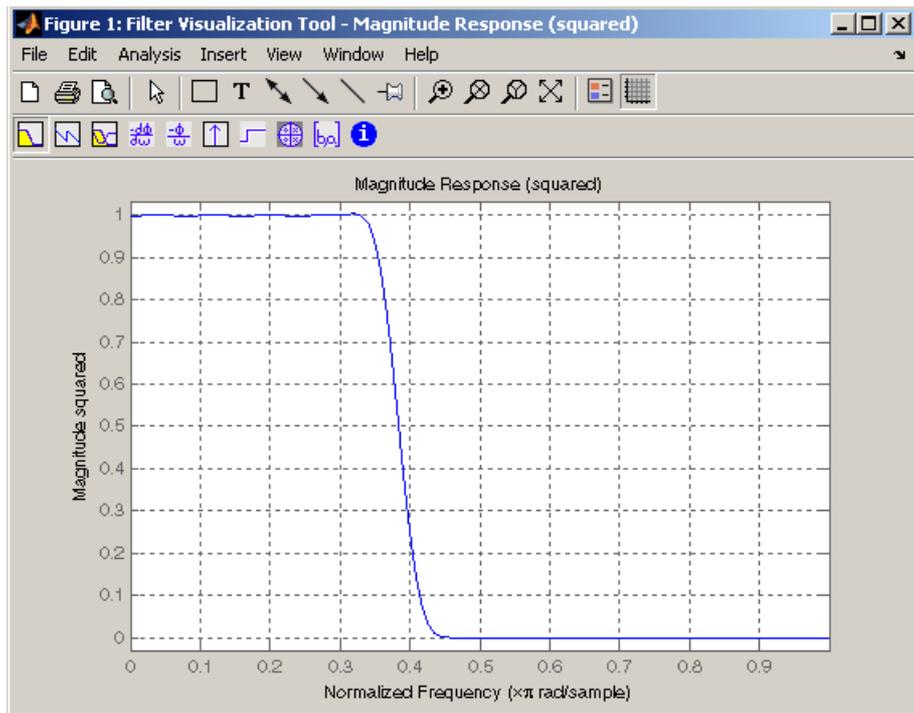
Note that the y -axis shown in the figure below is in Magnitude Squared. You can set this by right-clicking on the axis label and selecting **Magnitude Squared** from the menu.



Ringings and ripples occur in the response, especially near the band edge. This “Gibbs effect” does not vanish as the filter length increases, but a nonrectangular window reduces its magnitude. Multiplication by a window in the time domain causes a convolution or smoothing in the frequency domain. Apply a length 51 Hamming window to the filter and display the result using FVTool:

```
b = 0.4*sinc(0.4*(-25:25));
b = b.*hamming(51)';
fvtool(b,1)
```

Note that the y -axis shown in the figure below is in Magnitude Squared. You can set this by right-clicking on the axis label and selecting **Magnitude Squared** from the menu.



Using a Hamming window greatly reduces the ringing. This improvement is at the expense of transition width (the windowed version takes longer to ramp from passband to stopband) and optimality (the windowed version does not minimize the integrated squared error).

The functions `fir1` and `fir2` are based on this windowing process. Given a filter order and description of an ideal desired filter, these functions return a windowed inverse Fourier transform of that ideal filter. Both use a Hamming window by default, but they accept any window function. See “Windows” on page 3-2 for an overview of windows and their properties.

Standard Band FIR Filter Design: `fir1`

`fir1` implements the classical method of windowed linear phase FIR digital filter design. It resembles the IIR filter design functions in that it is formulated to design filters in standard band configurations: lowpass, bandpass, highpass, and bandstop.

The statements

```
n = 50;
Wn = 0.4;
b = fir1(n,Wn);
```

create row vector `b` containing the coefficients of the order `n` Hamming-windowed filter. This is a lowpass, linear phase FIR filter with cutoff frequency `Wn`. `Wn` is a number between 0 and 1, where 1 corresponds to the Nyquist frequency, half the sampling frequency. (Unlike other methods, here `Wn` corresponds to the 6 dB point.) For a highpass filter, simply append the string 'high' to the function's parameter list. For a bandpass or bandstop filter, specify `Wn` as a two-element vector containing the passband edge frequencies; append the string 'stop' for the bandstop configuration.

`b = fir1(n,Wn>window)` uses the window specified in column vector `window` for the design. The vector `window` must be `n+1` elements long. If you do not specify a window, `fir1` applies a Hamming window.

Kaiser Window Order Estimation. The `kaiserord` function estimates the filter order, cutoff frequency, and Kaiser window beta parameter needed to meet a given set of specifications. Given a vector of frequency band edges and a corresponding vector of magnitudes, as well as maximum allowable ripple, `kaiserord` returns appropriate input parameters for the `fir1` function.

Multiband FIR Filter Design: `fir2`

The `fir2` function also designs windowed FIR filters, but with an arbitrarily shaped piecewise linear frequency response. This is in contrast to `fir1`, which only designs filters in standard lowpass, highpass, bandpass, and bandstop configurations.

The commands

```
n = 50;  
f = [0 .4 .5 1];  
m = [1 1 0 0];  
b = fir2(n,f,m);
```

return row vector *b* containing the *n*+1 coefficients of the order *n* FIR filter whose frequency-magnitude characteristics match those given by vectors *f* and *m*. *f* is a vector of frequency points ranging from 0 to 1, where 1 represents the Nyquist frequency. *m* is a vector containing the desired magnitude response at the points specified in *f*. (The IIR counterpart of this function is `yulewalk`, which also designs filters based on arbitrary piecewise linear magnitude responses. See “IIR Filter Design” on page 1-4 for details.)

Multiband FIR Filter Design with Transition Bands

The `firls` and `firpm` functions provide a more general means of specifying the ideal desired filter than the `fir1` and `fir2` functions. These functions design Hilbert transformers, differentiators, and other filters with odd symmetric coefficients (type III and type IV linear phase). They also let you include transition or “don’t care” regions in which the error is not minimized, and perform band dependent weighting of the minimization.

The `firls` function is an extension of the `fir1` and `fir2` functions in that it minimizes the integral of the square of the error between the desired frequency response and the actual frequency response.

The `firpm` function implements the Parks-McClellan algorithm, which uses the Remez exchange algorithm and Chebyshev approximation theory to design filters with optimal fits between the desired and actual frequency responses. The filters are optimal in the sense that they minimize the maximum error between the desired frequency response and the actual frequency response; they are sometimes called *minimax* filters. Filters designed in this way exhibit an equiripple behavior in their frequency response, and hence are also known as *equiripple* filters. The Parks-McClellan FIR filter design algorithm is perhaps the most popular and widely used FIR filter design methodology.

The syntax for `firls` and `firpm` is the same; the only difference is their minimization schemes. The next example shows how filters designed with `firls` and `firpm` reflect these different schemes.

Basic Configurations

The default mode of operation of `firls` and `firpm` is to design type I or type II linear phase filters, depending on whether the order you desire is even or odd, respectively. A lowpass example with approximate amplitude 1 from 0 to 0.4 Hz, and approximate amplitude 0 from 0.5 to 1.0 Hz is

```
n = 20;                % Filter order
f = [0 0.4 0.5 1];    % Frequency band edges
a = [1 1 0 0];        % Desired amplitudes
b = firpm(n,f,a);
```

From 0.4 to 0.5 Hz, `firpm` performs no error minimization; this is a transition band or “don’t care” region. A transition band minimizes the error more in the bands that you do care about, at the expense of a slower transition rate. In this way, these types of filters have an inherent trade-off similar to FIR design by windowing.

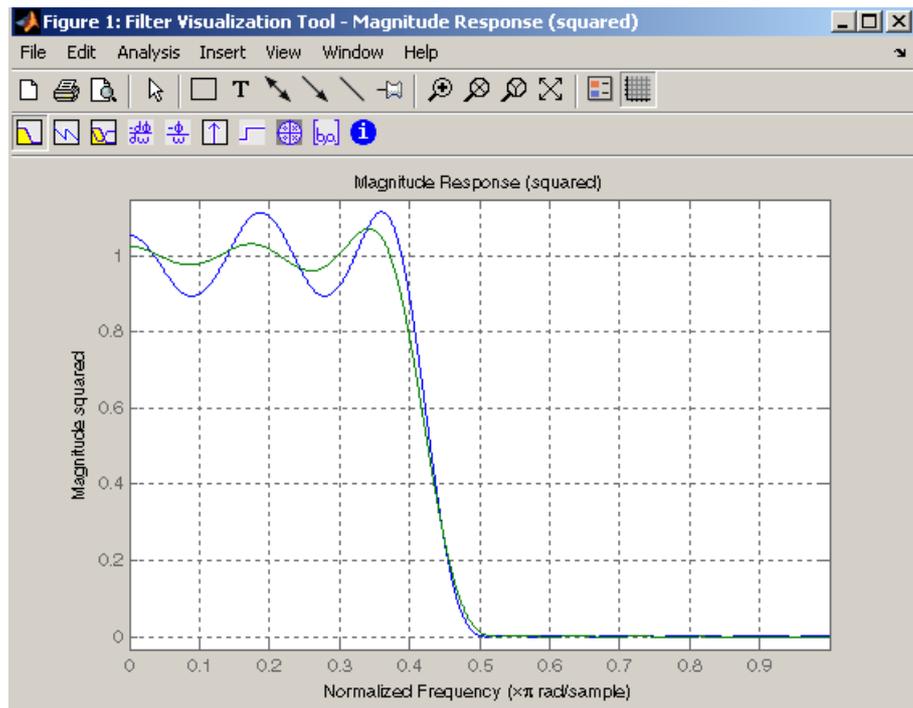
To compare least squares to equiripple filter design, use `firls` to create a similar filter. Type

```
bb = firls(n,f,a);
```

and compare their frequency responses using `FVTool`:

```
fvtool(b,1,bb,1)
```

Note that the y -axis shown in the figure below is in Magnitude Squared. You can set this by right-clicking on the axis label and selecting **Magnitude Squared** from the menu.



The filter designed with `firpm` exhibits equiripple behavior. Also note that the `fir1s` filter has a better response over most of the passband and stopband, but at the band edges ($f = 0.4$ and $f = 0.5$), the response is further away from the ideal than the `firpm` filter. This shows that the `firpm` filter's *maximum* error over the passband and stopband is smaller and, in fact, it is the smallest possible for this band edge configuration and filter length.

Think of frequency bands as lines over short frequency intervals. `firpm` and `fir1s` use this scheme to represent any piecewise linear desired function with any transition bands. `fir1s` and `firpm` design lowpass, highpass, bandpass, and bandstop filters; a bandpass example is

```
f = [0 0.3 0.4 0.7 0.8 1]; % Band edges in pairs
```

```
a = [0 0 1 1 0 0]; % Bandpass filter amplitude
```

Technically, these f and a vectors define five bands:

- Two stopbands, from 0.0 to 0.3 and from 0.8 to 1.0
- A passband from 0.4 to 0.7
- Two transition bands, from 0.3 to 0.4 and from 0.7 to 0.8

Example highpass and bandstop filters are

```
f = [0 0.7 0.8 1]; % Band edges in pairs
a = [0 0 1 1]; % Highpass filter amplitude
f = [0 0.3 0.4 0.5 0.8 1]; % Band edges in pairs
a = [1 1 0 0 1 1]; % Bandstop filter amplitude
```

An example multiband bandpass filter is

```
f = [0 0.1 0.15 0.25 0.3 0.4 0.45 0.55 0.6 0.7 0.75 0.85 0.9 1];
a = [1 1 0 0 1 1 0 0 1 1 0 0 1 1];
```

Another possibility is a filter that has as a transition region the line connecting the passband with the stopband; this can help control “runaway” magnitude response in wide transition regions:

```
f = [0 0.4 0.42 0.48 0.5 1];
a = [1 1 0.8 0.2 0 0]; % Passband, linear transition,
                        % stopband
```

The Weight Vector

Both `firls` and `firpm` allow you to place more or less emphasis on minimizing the error in certain frequency bands relative to others. To do this, specify a weight vector following the frequency and amplitude vectors. An example lowpass equiripple filter with 10 times less ripple in the stopband than the passband is

```
n = 20; % Filter order
f = [0 0.4 0.5 1]; % Frequency band edges
a = [1 1 0 0]; % Desired amplitudes
```

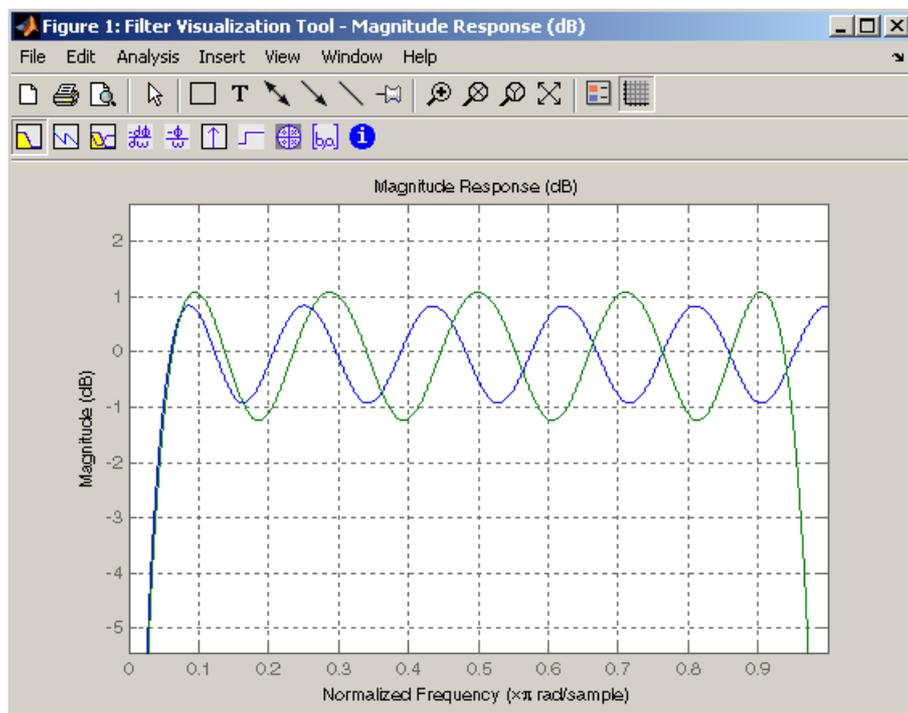
```
w = [1 10];           % Weight vector
b = firpm(n,f,a,w);
```

A legal weight vector is always half the length of the `f` and `a` vectors; there must be exactly one weight per band.

Anti-Symmetric Filters / Hilbert Transformers

When called with a trailing 'h' or 'Hilbert' option, `firpm` and `firls` design FIR filters with odd symmetry, that is, type III (for even order) or type IV (for odd order) linear phase filters. An ideal Hilbert transformer has this anti-symmetry property and an amplitude of 1 across the entire frequency range. Try the following approximate Hilbert transformers and plot them using `FVTool`:

```
b = firpm(21,[0.05 1],[1 1],'h');           % Highpass Hilbert
bb = firpm(20,[0.05 0.95],[1 1],'h');      % Bandpass Hilbert
fvtool(b,1,bb,1)
```



You can find the delayed Hilbert transform of a signal x by passing it through these filters.

```
fs = 1000;           % Sampling frequency
t = (0:1/fs:2)';    % Two second time vector
x = sin(2*pi*300*t); % 300 Hz sine wave example signal
xh = filter(bb,1,x); % Hilbert transform of x
```

The analytic signal corresponding to x is the complex signal that has x as its real part and the Hilbert transform of x as its imaginary part. For this FIR method (an alternative to the `hilbert` function), you must delay x by half the filter order to create the analytic signal:

```
xd = [zeros(10,1); x(1:length(x)-10)]; % Delay 10 samples
xa = xd + j*xh;                       % Analytic signal
```

This method does not work directly for filters of odd order, which require a noninteger delay. In this case, the `hilbert` function, described in “Specialized Transforms” on page 3-40, estimates the analytic signal. Alternatively, use the `resample` function to delay the signal by a noninteger number of samples.

Differentiators

Differentiation of a signal in the time domain is equivalent to multiplication of the signal’s Fourier transform by an imaginary ramp function. That is, to differentiate a signal, pass it through a filter that has a response $H(\omega) = j\omega$. Approximate the ideal differentiator (with a delay) using `firpm` or `firls` with a 'd' or 'differentiator' option:

```
b = firpm(21,[0 1],[0 pi],'d');
```

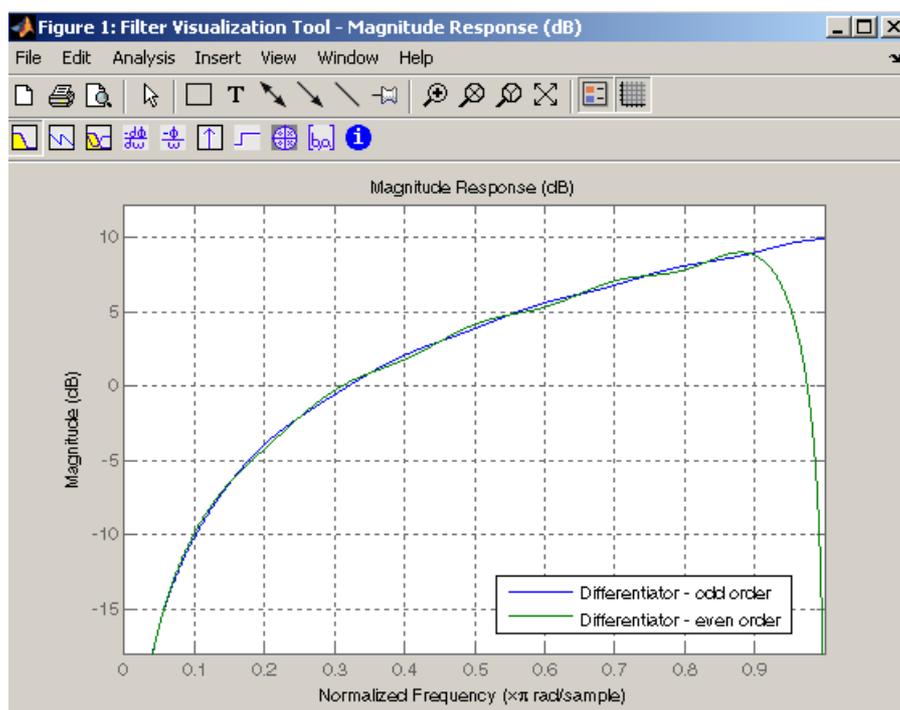
For a type III filter, the differentiation band should stop short of the Nyquist frequency, and the amplitude vector must reflect that change to ensure the correct slope:

```
bb = firpm(20,[0 0.9],[0 0.9*pi],'d');
```

In the 'd' mode, `firpm` weights the error by $1/\omega$ in nonzero amplitude bands to minimize the maximum *relative* error. `firls` weights the error by $(1/\omega)^2$ in nonzero amplitude bands in the 'd' mode.

The following plots show the magnitude responses for the differentiators above.

```
fvtool(b,1,bb,1)
```



Constrained Least Squares FIR Filter Design

The Constrained Least Squares (CLS) FIR filter design functions implement a technique that enables you to design FIR filters without explicitly defining the transition bands for the magnitude response. The ability to omit the specification of transition bands is useful in several situations. For example, it may not be clear where a rigidly defined transition band should appear if noise and signal information appear together in the same frequency band. Similarly, it may make sense to omit the specification of transition bands if they appear only to control the results of Gibbs phenomena that appear in the filter's response. See Selesnick, Lang, and Burrus [2] for discussion of this method.

Instead of defining passbands, stopbands, and transition regions, the CLS method accepts a cutoff frequency (for the highpass, lowpass, bandpass, or bandstop cases), or passband and stopband edges (for multiband cases), for the desired response. In this way, the CLS method defines transition regions implicitly, rather than explicitly.

The key feature of the CLS method is that it enables you to define upper and lower thresholds that contain the maximum allowable ripple in the magnitude response. Given this constraint, the technique applies the least square error minimization technique over the frequency range of the filter's response, instead of over specific bands. The error minimization includes any areas of discontinuity in the ideal, "brick wall" response. An additional benefit is that the technique enables you to specify arbitrarily small peaks resulting from Gibbs' phenomena.

There are two toolbox functions that implement this design technique.

Description	Function
Constrained least square multiband FIR filter design	fircls
Constrained least square filter design for lowpass and highpass linear phase filters	fircls1

For details on the calling syntax for these functions, see their reference descriptions in the Function Reference.

Basic Lowpass and Highpass CLS Filter Design

The most basic of the CLS design functions, `fircls1`, uses this technique to design lowpass and highpass FIR filters. As an example, consider designing a filter with order 61 impulse response and cutoff frequency of 0.3 (normalized). Further, define the upper and lower bounds that constrain the design process as:

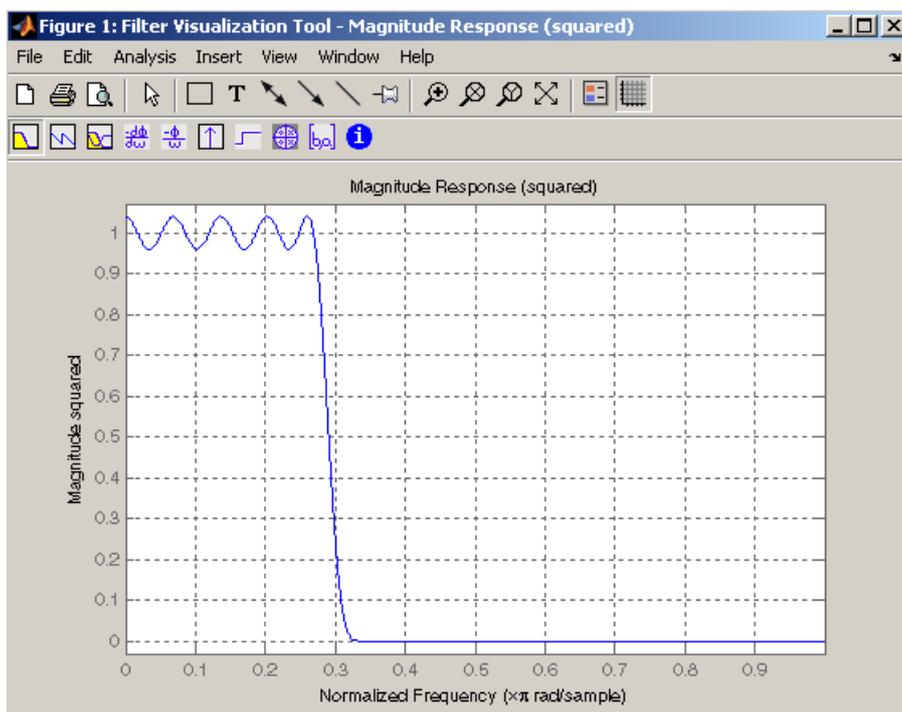
- Maximum passband deviation from 1 (passband ripple) of 0.02.
- Maximum stopband deviation from 0 (stopband ripple) of 0.008.



To approach this design problem using `fircls1`, use the following commands:

```
n = 61;
wo = 0.3;
dp = 0.02;
ds = 0.008;
h = fircls1(n,wo,dp,ds);
fvtool(h,1)
```

Note that the y-axis shown below is in Magnitude Squared. You can set this by right-clicking on the axis label and selecting **Magnitude Squared** from the menu.



Multiband CLS Filter Design

`fircls` uses the same technique to design FIR filters with a desired piecewise constant magnitude response. In this case, you can specify a vector of band edges and a corresponding vector of band amplitudes. In addition, you can specify the maximum amount of ripple for each band.

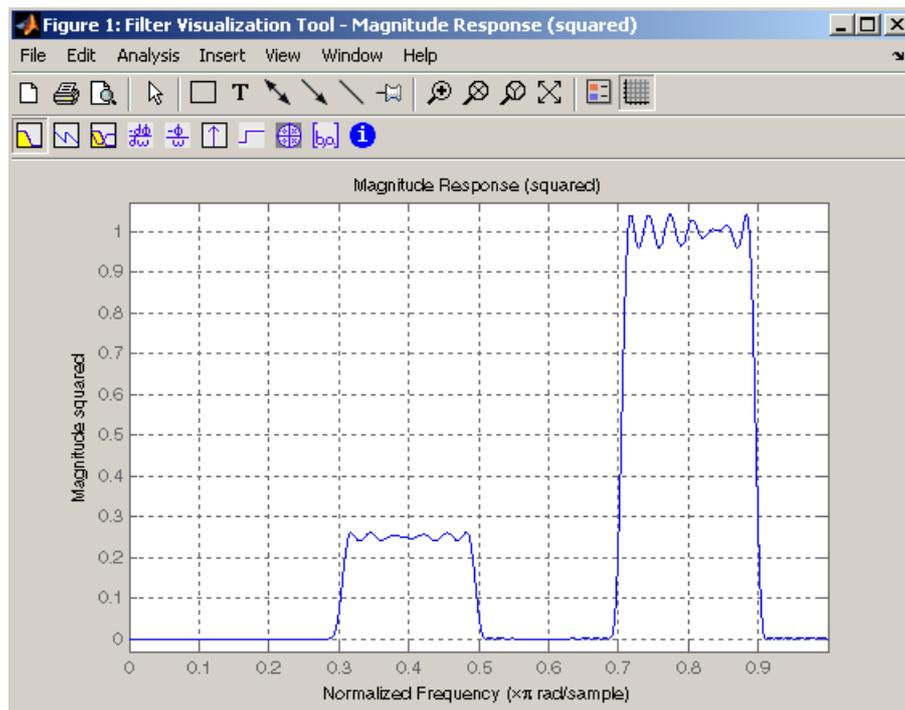
For example, assume the specifications for a filter call for:

- From 0 to 0.3 (normalized): amplitude 0, upper bound 0.005, lower bound -0.005
- From 0.3 to 0.5: amplitude 0.5, upper bound 0.51, lower bound 0.49
- From 0.5 to 0.7: amplitude 0, upper bound 0.03, lower bound -0.03
- From 0.7 to 0.9: amplitude 1, upper bound 1.02, lower bound 0.98
- From 0.9 to 1: amplitude 0, upper bound 0.05, lower bound -0.05

Design a CLS filter with impulse response order 129 that meets these specifications:

```
n = 129;  
f = [0 0.3 0.5 0.7 0.9 1];  
a = [0 0.5 0 1 0];  
up = [0.005 0.51 0.03 1.02 0.05];  
lo = [-0.005 0.49 -0.03 0.98 -0.05];  
h = fircls(n,f,a,up,lo);  
fvtool(h,1)
```

Note that the y -axis shown below is in Magnitude Squared. You can set this by right-clicking on the axis label and selecting **Magnitude Squared** from the menu.



Weighted CLS Filter Design

Weighted CLS filter design lets you design lowpass or highpass FIR filters with relative weighting of the error minimization in each band. The `fircls1` function enables you to specify the passband and stopband edges for the least squares weighting function, as well as a constant k that specifies the ratio of the stopband to passband weighting.

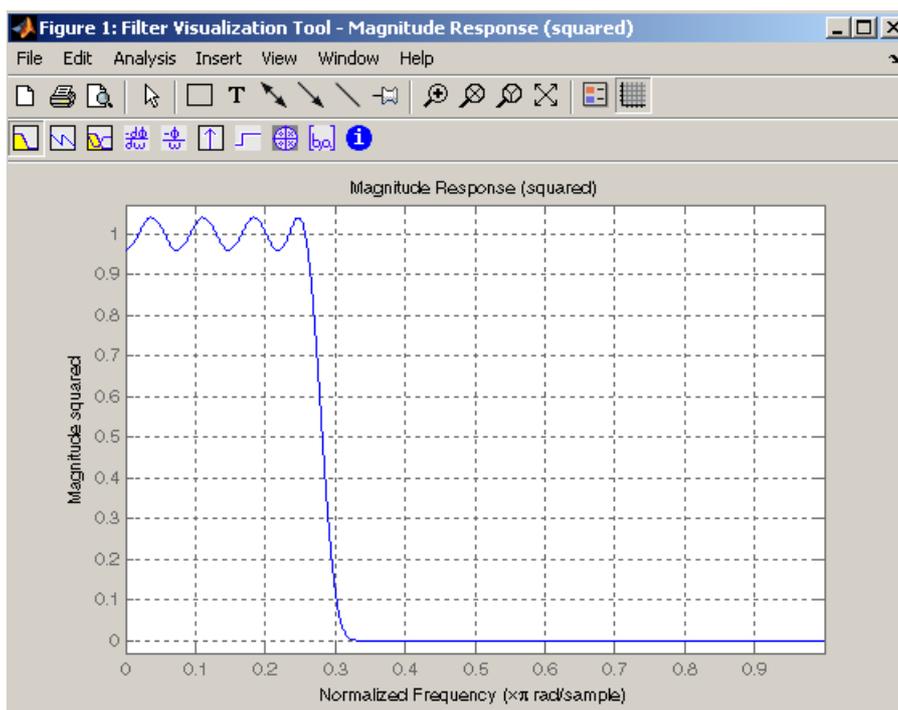
For example, consider specifications that call for an FIR filter with impulse response order of 55 and cutoff frequency of 0.3 (normalized). Also assume maximum allowable passband ripple of 0.02 and maximum allowable stopband ripple of 0.004. In addition, add weighting requirements:

- Passband edge for the weight function of 0.28 (normalized)
- Stopband edge for the weight function of 0.32
- Weight error minimization 10 times as much in the stopband as in the passband

To approach this using `fircls1`, type

```
n = 55;
wo = 0.3;
dp = 0.02;
ds = 0.004;
wp = 0.28;
ws = 0.32;
k = 10;
h = fircls1(n,wo,dp,ds,wp,ws,k);
fvtool(h,1)
```

Note that the y -axis shown below is in Magnitude Squared. You can set this by right-clicking on the axis label and selecting **Magnitude Squared** from the menu.



Arbitrary-Response Filter Design

The `cfirpm` filter design function provides a tool for designing FIR filters with arbitrary complex responses. It differs from the other filter design functions in how the frequency response of the filter is specified: it accepts the name of a function which returns the filter response calculated over a grid of frequencies. This capability makes `cfirpm` a highly versatile and powerful technique for filter design.

This design technique may be used to produce nonlinear-phase FIR filters, asymmetric frequency-response filters (with complex coefficients), or more symmetric filters with custom frequency responses.

The design algorithm optimizes the Chebyshev (or minimax) error using an extended Remez-exchange algorithm for an initial estimate. If this exchange method fails to obtain the optimal filter, the algorithm switches to an ascent-descent algorithm that takes over to finish the convergence to the optimal solution.

Multiband Filter Design

Consider a multiband filter with the following special frequency-domain characteristics.

Band	Amplitude	Optimization Weighting
[-1 -0.5]	[5 1]	1
[-0.4 +0.3]	[2 2]	10
[+0.4 +0.8]	[2 1]	5

A linear-phase multiband filter may be designed using the predefined frequency-response function `multiband`, as follows:

```
b = cfirpm(38, [-1 -0.5 -0.4 0.3 0.4 0.8], ...
             {'multiband', [5 1 2 2 2 1]}, [1 10 5]);
```

For the specific case of a multiband filter, we can use a shorthand filter design notation similar to the syntax for `firpm`:

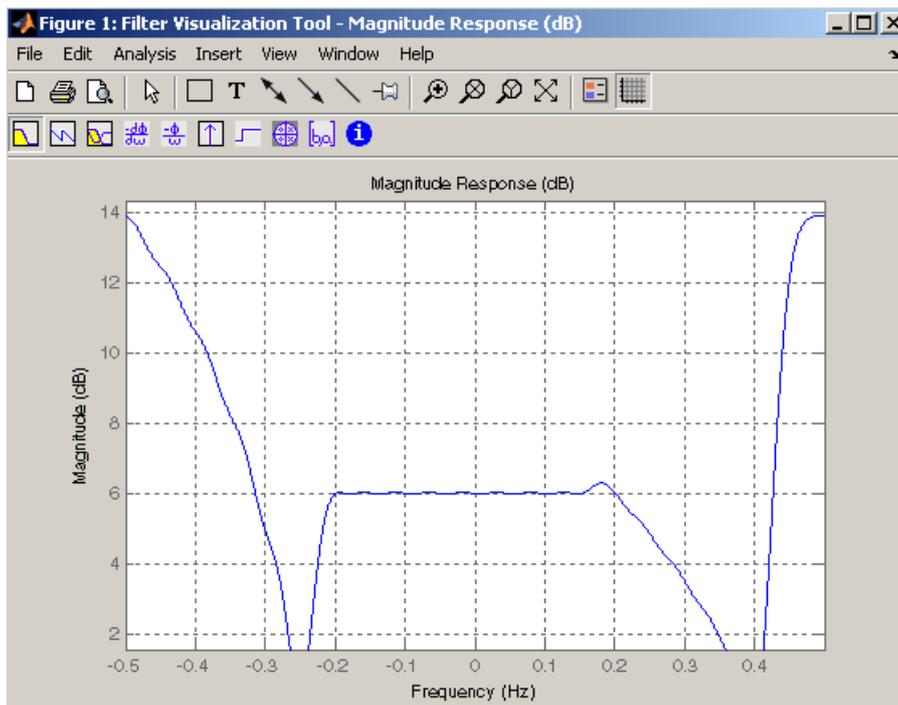
```
b = cfirpm(38, [-1 -0.5 -0.4 0.3 0.4 0.8], ...
            [5 1 2 2 2 1], [1 10 5]);
```

As with `firpm`, a vector of band edges is passed to `cfirpm`. This vector defines the frequency bands over which optimization is performed; note that there are two transition bands, from -0.5 to -0.4 and from 0.3 to 0.4.

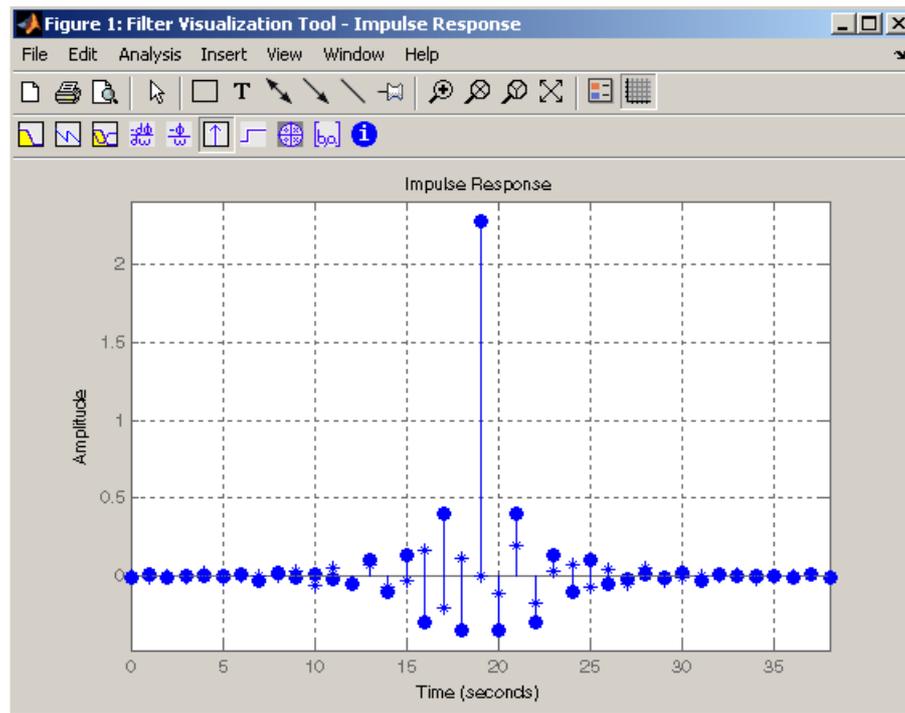
In either case, the frequency response is obtained and plotted using linear scale in `FVTool`:

```
fvtool(b,1)
```

Note that the range of data shown below is $(-F_s/2, F_s/2)$. You can set this range by changing the x -axis units to **Frequency ($F_s = 1$ Hz)**.



The filter response for this multiband filter is complex, which is expected because of the asymmetry in the frequency domain. The impulse response, which you can select from the FVTool toolbar, is shown below.



Filter Design with Reduced Delay

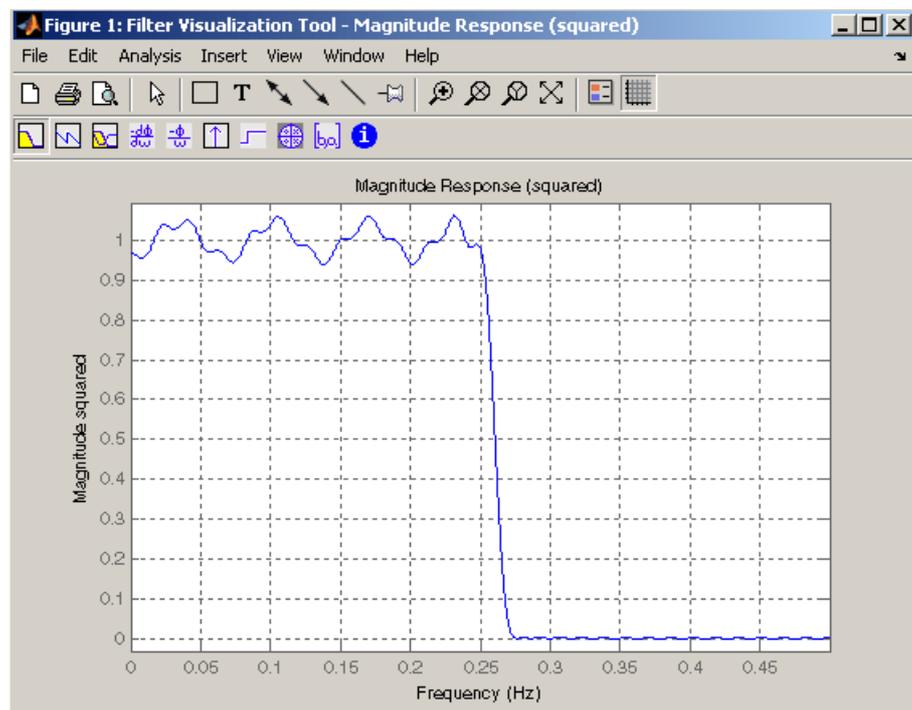
Consider the design of a 62-tap lowpass filter with a half-Nyquist cutoff. If we specify a negative offset value to the lowpass filter design function, the group delay offset for the design is significantly less than that obtained for a standard linear-phase design. This filter design may be computed as follows:

```
b = cfirpm(61,[0 0.5 0.55 1],{'lowpass',-16});
```

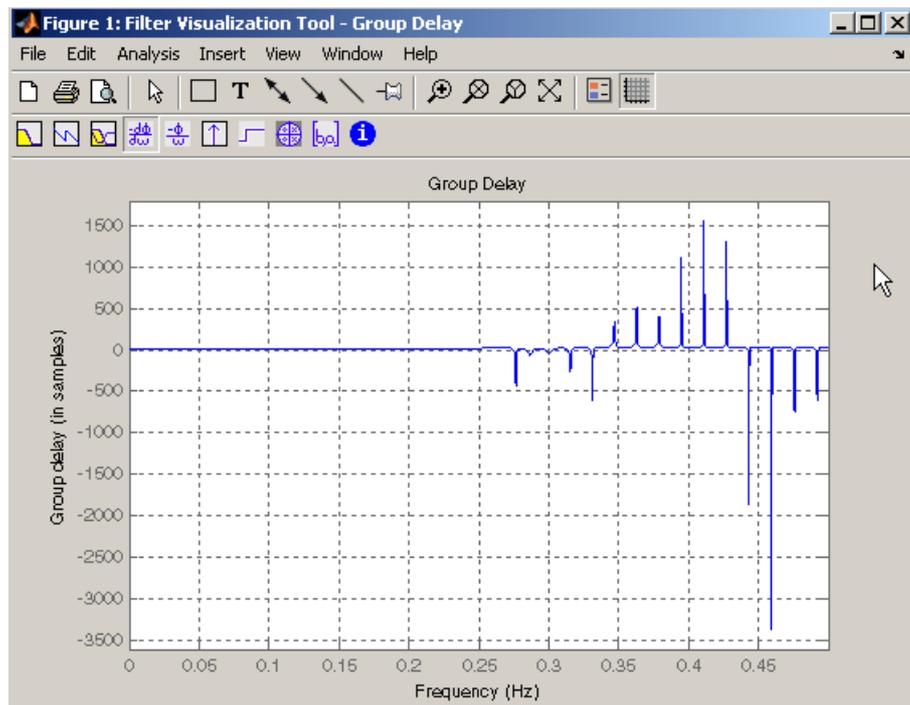
The resulting magnitude response is

```
fvtool(b,1)
```

Note that the range of data in this plot is $(-F_s/2, F_s/2)$, which you can set changing the x-axis units to **Frequency**. The y-axis is in Magnitude Squared, which you can set by right-clicking on the axis label and selecting **Magnitude Squared** from the menu.



The group delay of the filter reveals that the offset has been reduced from $N/2$ to $N/2 - 16$ (i.e., from 30.5 to 14.5). Now, however, the group delay is no longer flat in the passband region. To create this plot, click the **Group Delay** button on the toolbar.



If we compare this nonlinear-phase filter to a linear-phase filter that has exactly 14.5 samples of group delay, the resulting filter is of order 2×14.5 , or 29. Using `b = cfirpm(29,[0 0.5 0.55 1], 'lowpass')`, the passband and stopband ripple is much greater for the order 29 filter. These comparisons can assist you in deciding which filter is more appropriate for a specific application.