Digital Filter Design Using Matlab

By Timothy J. Schlichter

EE 4000 Introduction to Digital Filtering

5/2/99

Submitted to: Dr. Joseph Picone
Mississippi State University
Department of Electrical and Computer Engineering

EXECUTIVE SUMMARY

A fundamental aspect of signal processing is filtering. Filtering involves the manipulation of the spectrum of a signal by passing or blocking certain portions of the spectrum, depending on the frequency of those portions. Filters are designed according to what kind of manipulation of the signal is required for a particular application. Digital filters are implemented using three fundamental building blocks: an adder, a multiplier, and a delay element.

The design process of a digital filter is long and tedious if done by hand. With the aid of computer programs performing filter design algorithms, designing and optimizing filters can be done relatively quickly. This paper discusses the use of Matlab, a mathematical software package, to design, manipulate, and analyze digital filters.

The design options in Matlab allow the user to either create a code for designing filters that calls built-in functions, or to design filters in Sptool, a graphical user interface. Each of these methods are examined in this paper. The strengths and weaknesses of each are detailed in the following discussion.

This paper concludes with a discussion of how the data given by Matlab for various filters can be used to implement filters on real digital signal processors. Matlab provides all the information necessary for building a hardware replica of the filter designed in software.

TABLE OF CONTENTS

Abstract

Matlab provides different options for digital filter design, which include function calls to filter algorithms and a graphical user interface called Sptool. A variety of filter design algorithms are available in Matlab for both IIR and FIR filters. This paper discusses the different options in Matlab and gives examples of lowpass, highpass, and bandpass filter designs.

Results show that the graphical user interface Sptool is a quicker and simpler option than the option of making function calls to the filter algorithms. Sptool has a more user-friendly environment since the spectrum of the filter is immediately displayed to the user, and the user can quickly zoom in and examine particular areas of interest in the spectrum (i.e. the passband). However, the shortcoming of Sptool is that it only displays the magnitude response of the filter, not the phase response.

Introduction

A key element in processing digital signals is the filter. Filters perform direct manipulations on the spectra of signals. To completely describe digital filters, three basic elements (or building blocks) are needed: an adder, a multiplier, and a delay element. The adder has two inputs and one output, and it simply adds the two inputs together. The multiplier is a gain element, and it multiplies the input signal by a constant. The delay element delays the incoming signal by one sample. Digital filters can be implemented using either a block diagram or a signal flow graph. Figure 1 shows the three basic elements in block diagram form, and Figure 2 shows them in signal flow graph form.

Figure 1: Block Diagram of Filter Elements

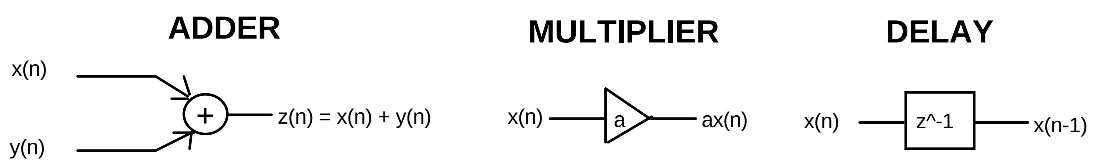
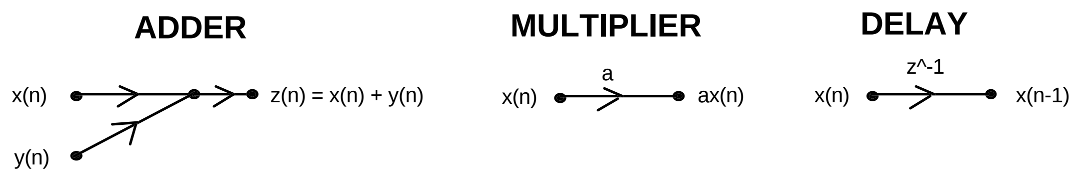


Figure 2: Signal Flow Graph of Filter Elements



With the basic building blocks at hand, the two different filter structures can easily be implemented. These two structures are Infinite Impulse Response (IIR) and Finite Impulse Response (FIR), depending on the form of the system's response to a unit pulse input. IIR filters are commonly implemented using a feedback (recursive) structure, while FIR filters usually require no feedback (non-recursive).

In the design of IIR filters, a commonly used approach is called the bilinear transformation. This design begins with the transfer function of an analog filter, then performs a mapping from the s-domain to the z-domain. Using differential equations, it can be shown (Proakis 677) that the mapping from the s-plane to the z-plane is

$$s = \frac{2}{T}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)$$

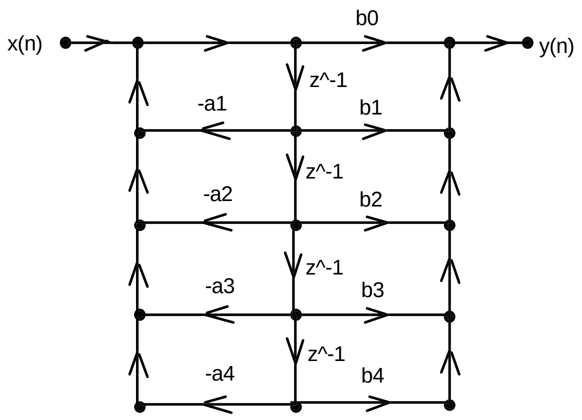
This mapping results in a general form for an IIR filter with an arbitrary number of poles and zeros. The system response and the difference equation for this filter is as follows:

$$H(z) = \frac{B(z)}{A(z)} = \frac{\sum_{n=0}^{M} b_n z^{-n}}{\sum_{n=0}^{N} a_n z^{-n}} = \frac{b_0 + b_1 z^{-1} + \ldots\ldots + b_M z^{-M}}{1 + a_1 z^{-1} + \ldots\ldots + a_N z^{-N}}, a_0 = 1 \qquad \text{(Ingle 183)}$$

$$y(n) = \sum_{m=0}^{M} b_m x(n-m) - \sum_{n=0}^{N} a_m y(n-m)$$

This system response can be easily realized using a signal flow graph.

Figure 3: Signal Flow Graph of IIR Filter



An FIR filter has a difference equation of

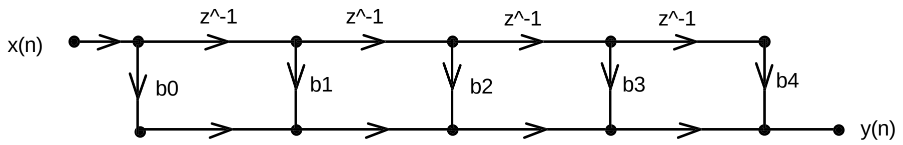
$$y(n) = \sum_{k=0}^{M-1} b_k x(n-k) \qquad \text{(Proakis 620)}$$

By taking the z-transform, the system response is

$$H(z) = b_0 + b_1 z^{-1} + \ldots\ldots + b_{M-1} z^{1-M} = \sum_{k=0}^{M-1} b_k z^{-k} \qquad \text{(Ingle 197)}$$

The realization of an FIR filter using a signal flow graph is straightforward.

Figure 4: Signal Flow Graph of FIR Filter



Matlab has several design algorithms that can be used to create and analyze both IIR and FIR digital filters. The IIR filters that can be created in Matlab are Butterworth, Chebyshev type 1 and 2, and elliptic. The FIR filter algorithms in Matlab are equiripple, least squares, and Kaiser window. The Matlab code required to implement these filters involves bilinear transformations and function calls to analog prototype filters. The following sections give examples of Matlab implementation of the IIR filters listed above.

Lowpass Filter Design

Using Matlab, a lowpass digital filter is designed using various analog prototypes: Chebyshev, Butterworth, and Elliptic. The optimum filter type is chosen on the basis of implementation complexity, magnitude response, and phase response. The design specifications for the filter are as follows:

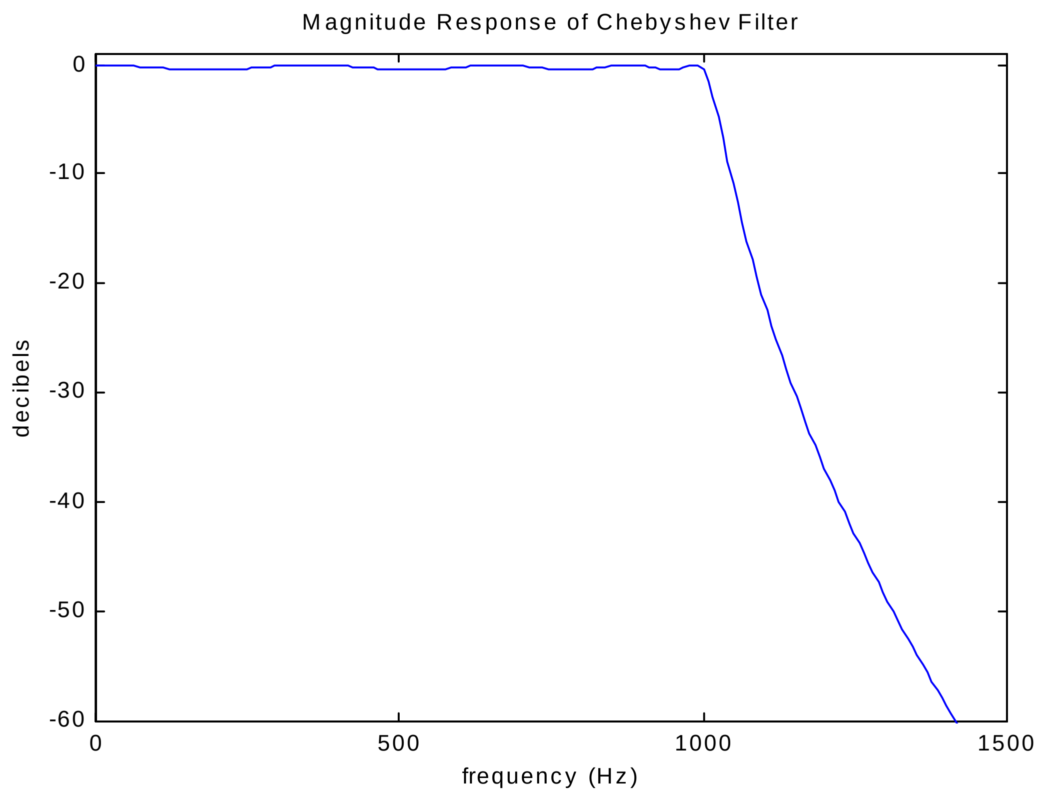
- Cutoff frequency = 1000Hz
- Sample frequency = 8000Hz
- Passband ripple = 0.5dB
- Stopband attn. = 60dB
- Transition band = 100Hz

Matlab Code (Chebyshev):

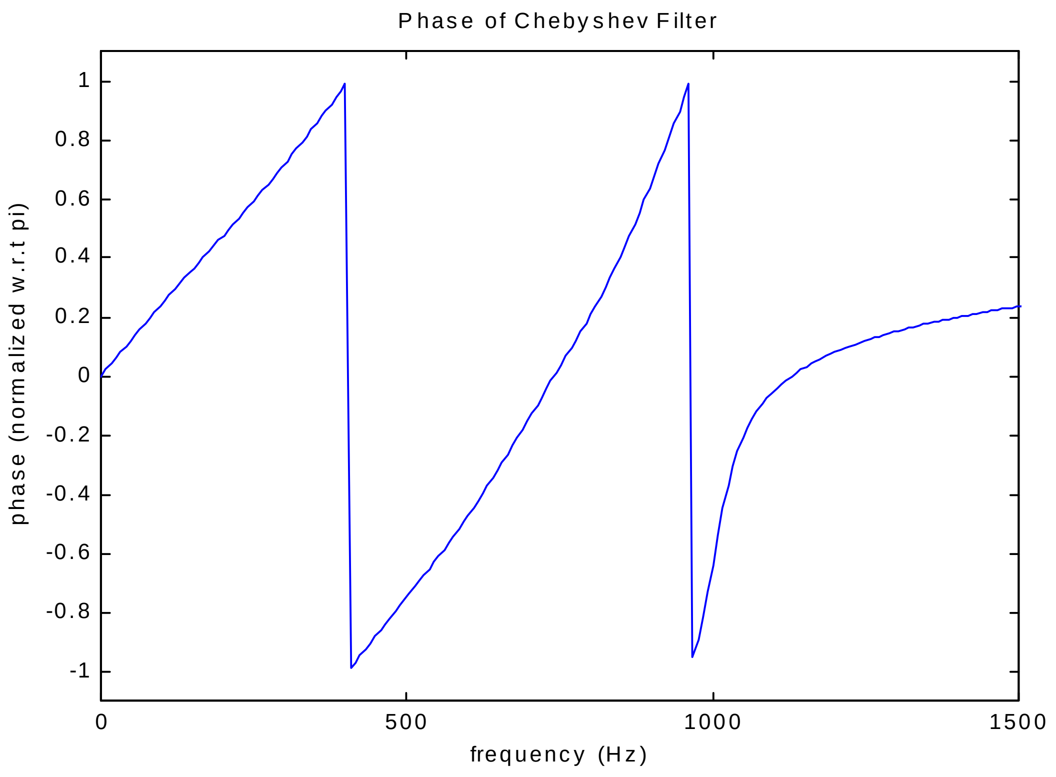
```
% Lowpass digital filter with Chebyshev-I analog prototype
%
% Digital Filter Specifications:
wp = 0.125*2*pi;  % digital passband frequency in Hz (normalized)
ws = 0.1375*2*pi; % digital stopband frequency in Hz (normalized)
Rp = 0.5;         % passband ripple in dB
As = 20;          % stopband attenuation in dB

% Analog Prototype Specifications:
Fs = 1; T = 1/Fs;
OmegaP = (2/T)*tan(wp/2);  % prewarp prototype passband frequency
OmegaS = (2/T)*tan(ws/2);  % prewarp prototype stopband frequency

% Analog Chebyshev-1 Prototype Filter Calculation:
[c, d] = chb1(OmegaP, OmegaS, Rp, As);

% Bilinear Transformation:
[b, a] = bilinear(cs, ds, Fs);
%
[db,mag,pha,grd,w] = freqz(b,a);
plot(w*8000/2/pi,db);
xlabel('frequency (Hz)'); ylabel('decibels'); title('Magnitude in
dB');
```

This exact code is also used for the elliptic and Butterworth designs. The only change is in the filter calculations of each type. Instead of calling **`chb1`**(), the elliptic filter design calls a function “**`elliptic`**()” and the Butterworth design calls a function “**`butterworth`**()”. See the appendix for the Matlab code of the function **`chb1`**().

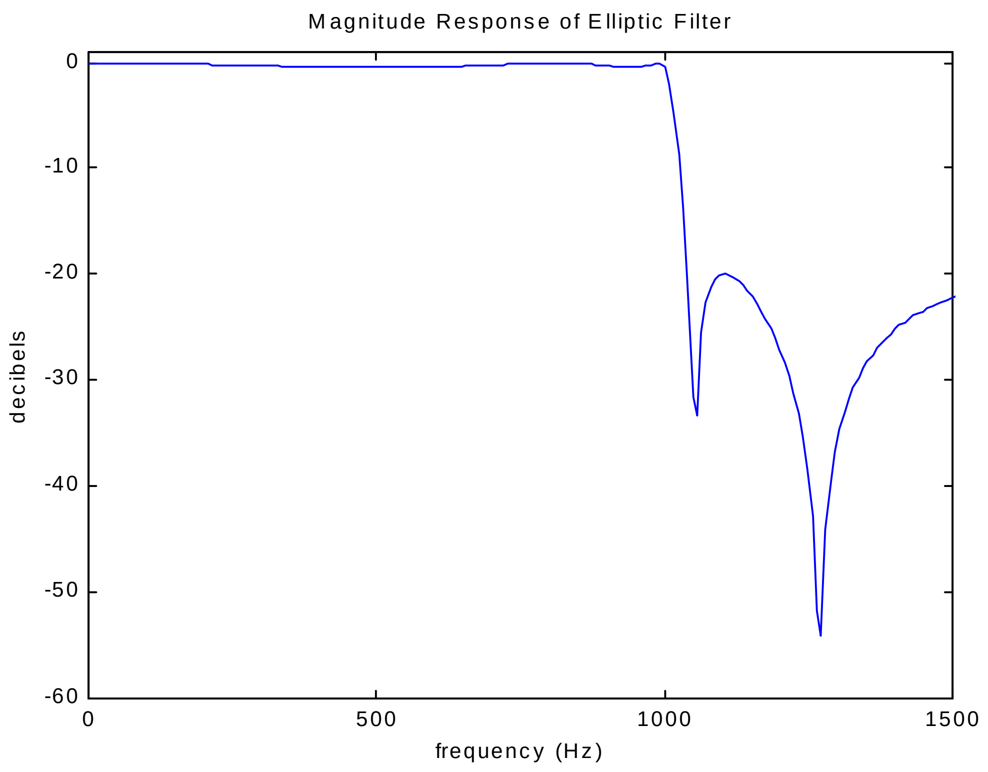
The following figures show the magnitude and phase responses of each type of filter.

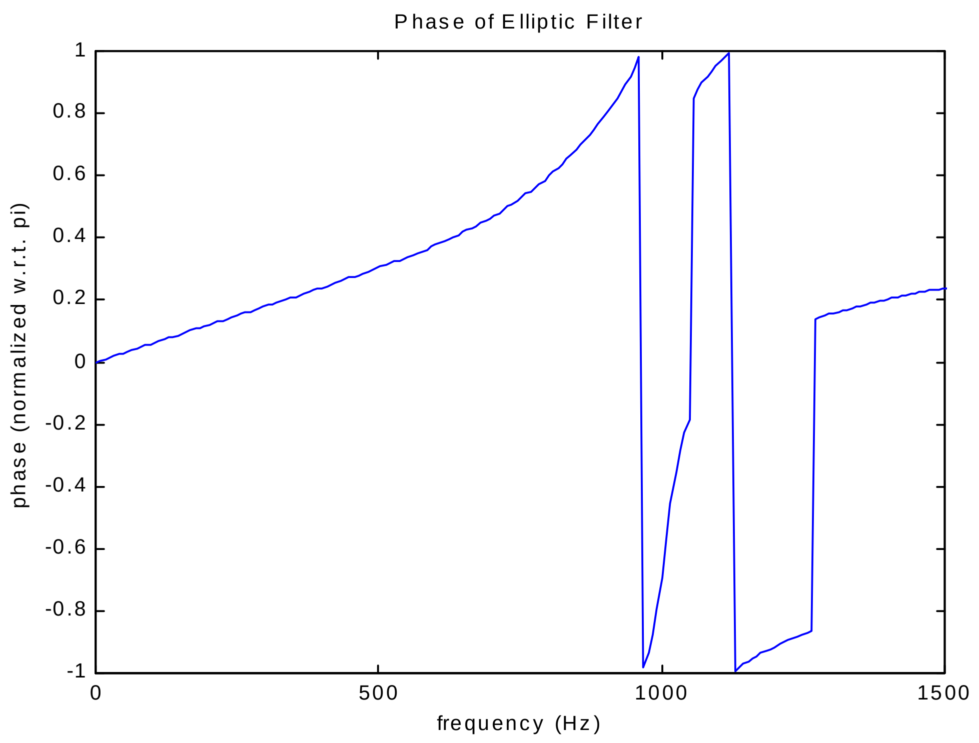


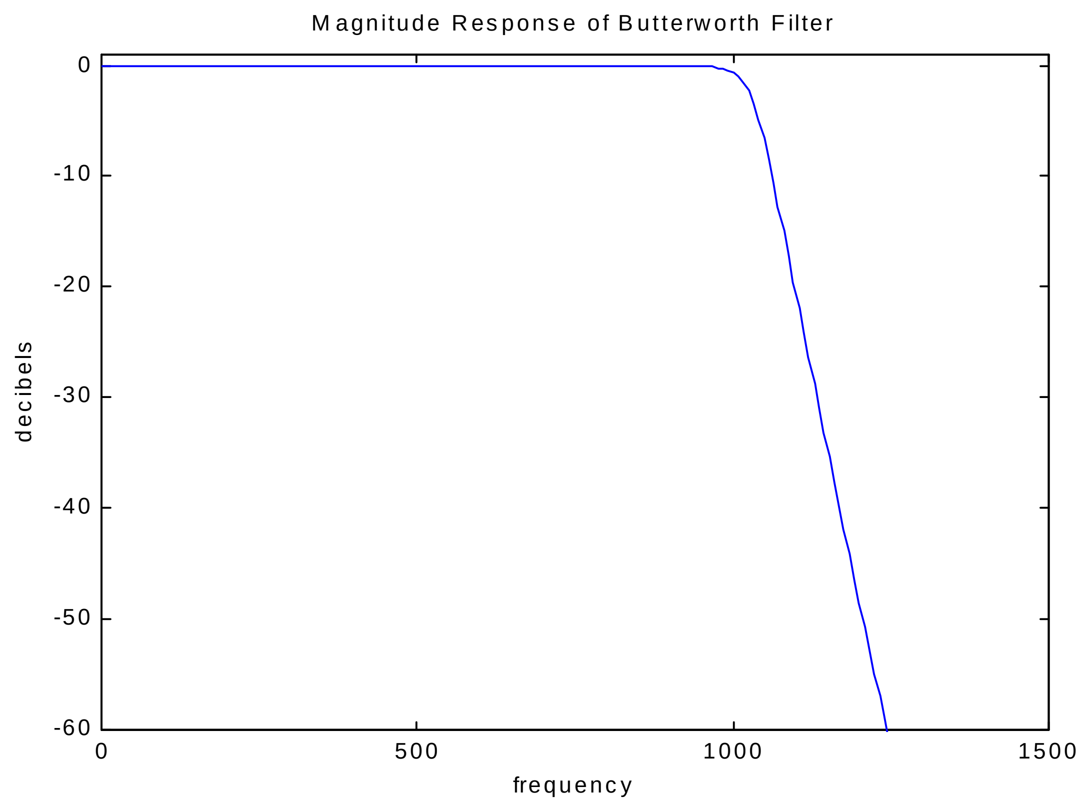
Magnitude Response of Chebyshev Filter
decibels
0
-10
-20
-30
-40
-50
-60
0
500
1000
1500
frequency (Hz)

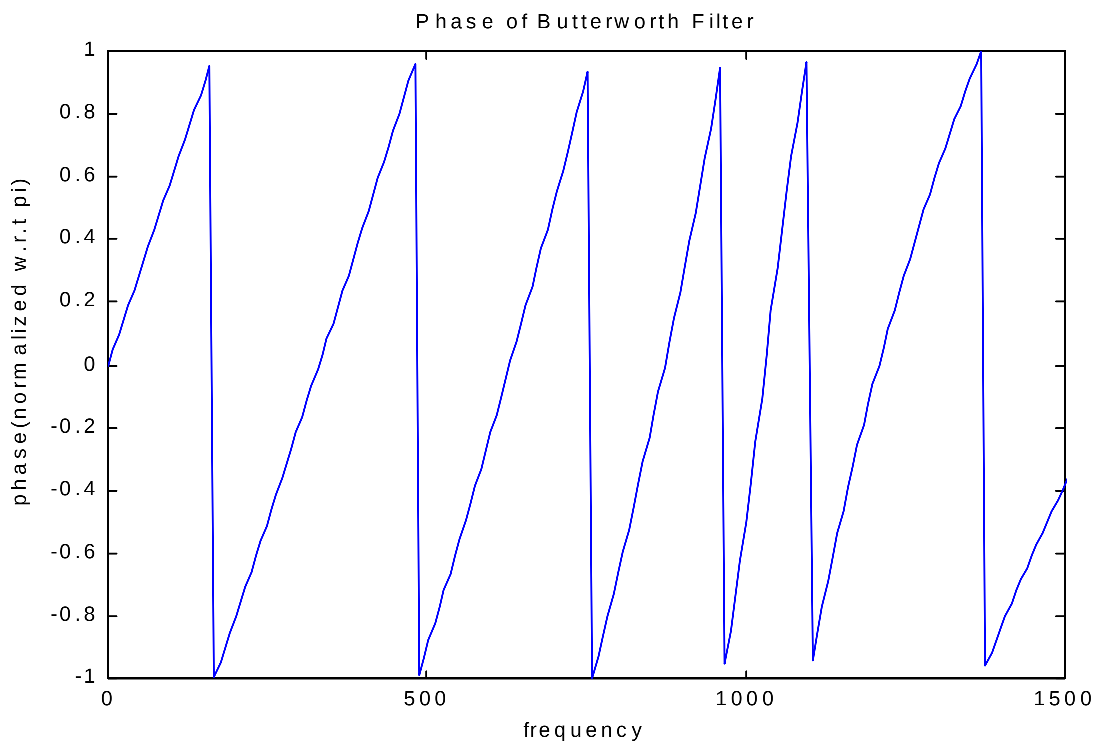


Phase of Chebyshev Filter
phase (normalized w.r.t pi)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
500
1000
1500
frequency (Hz)


Magnitude Response of Elliptic Filter
decibels
0
-10
-20
-30
-40
-50
-60
0
500
1000
1500
frequency (Hz)


Phase of Elliptic Filter
phase (normalized w.r.t. pi)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
500
1000
1500
frequency (Hz)


Magnitude Response of Butterworth Filter
0
-10
-20
-30
-40
-50
-60
decibels
0
500
1000
1500
frequency


Phase of Butterworth Filter
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
phase(normalized w.r.t pi)
0
500
1000
1500
frequency

The Matlab code outputs the filter order and the filter coefficients. For this example, the Chebyshev filter order was nine. The elliptic filter had an order of five, and the Butterworth filter order was thirty-two.

Several conclusions can be drawn about these low-pass filter designs from this simple example. First, in general, for a given set of design constraints, the elliptic filter design algorithm will result in the simplest filter (in terms of complexity). The most complex filter is the Butterworth filter with an order of thirty-two. In terms of passband ripple, the Butterworth filter gives the optimum response. In the passband, there is almost no ripple (monotonic). The elliptic and Chebyshev filters both have much more ripple in the passband. So, there is a tradeoff between these three different types of filters. In terms of magnitude response and complexity, the elliptic ripple is most likely the optimum choice. However, the elliptic ripple has a phase response that is more nonlinear than the Chebyshev and Butterworth filters. Therefore, if a sharp cutoff and relatively low complexity is required, the choice would be the elliptic filter. If the phase response would need to be more linear, a Chebyshev or Butterworth filter should be chosen over the elliptic filter.

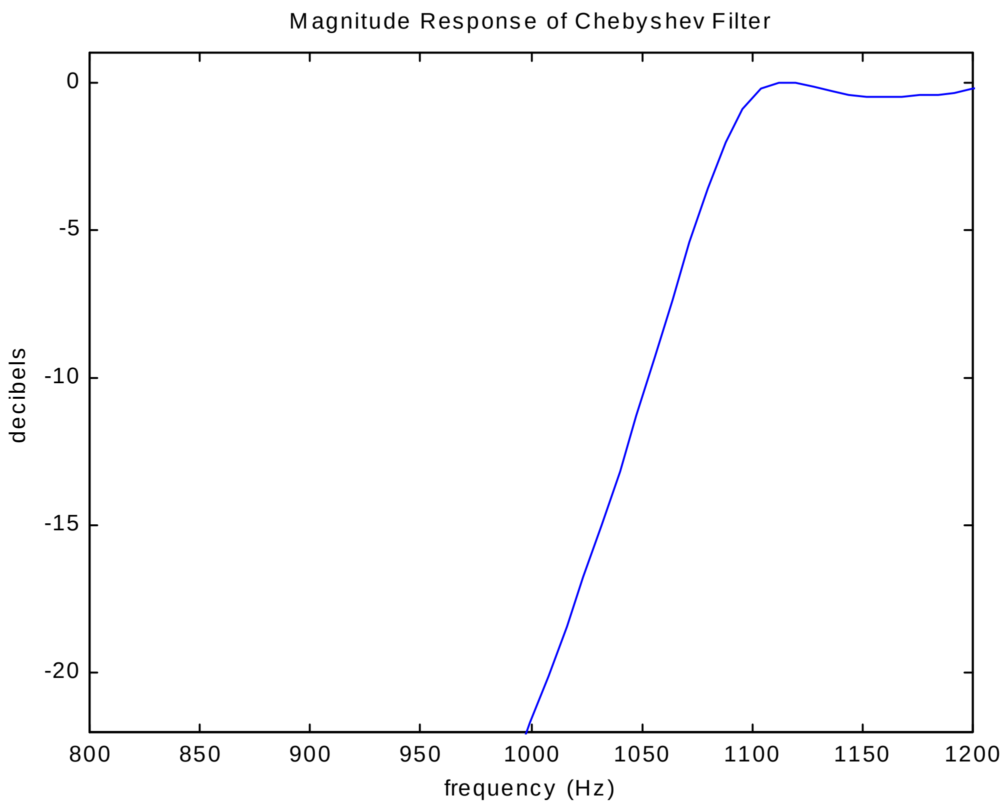
Highpass and Bandpass Filter Design

Matlab provides functions for implementing lowpass-to-highpass and lowpass-to-bandpass conversions. By providing a filter order, the passband ripple, and the 3dB cutoff frequency to the function **`cheby1`**(), a highpass filter can be designed. The filter order is found using the function **`chebord`**(). For a Butterworth prototype, the functions are **`butter`**() and **`buttord`**(). For the elliptic prototype, the functions are **`ellip`**() and **`ellipord`**().

The following Matlab code is used to design a Chebyshev highpass digital filter with a passband at 1100Hz and a 100Hz transition band.

```
% Highpass Chebyshev Digital Filter

ws = 0.125*2*pi;     % digital stopband frequency in rad/s
wp = 0.1375*2*pi;    % digital passband frequency in rad/s
Rp = 0.5;            % passband ripple in dB
As = 20;

[N,wn] = cheb1ord(wp/pi,ws/pi,Rp,As);
[b,a] = cheby1(N, Rp, wn, 'high');

[db,mag,pha,grd,w] = freqz_m(b,a);
plot(w*8000/2/pi,db);
axis([800 1200 -22 1]);
```

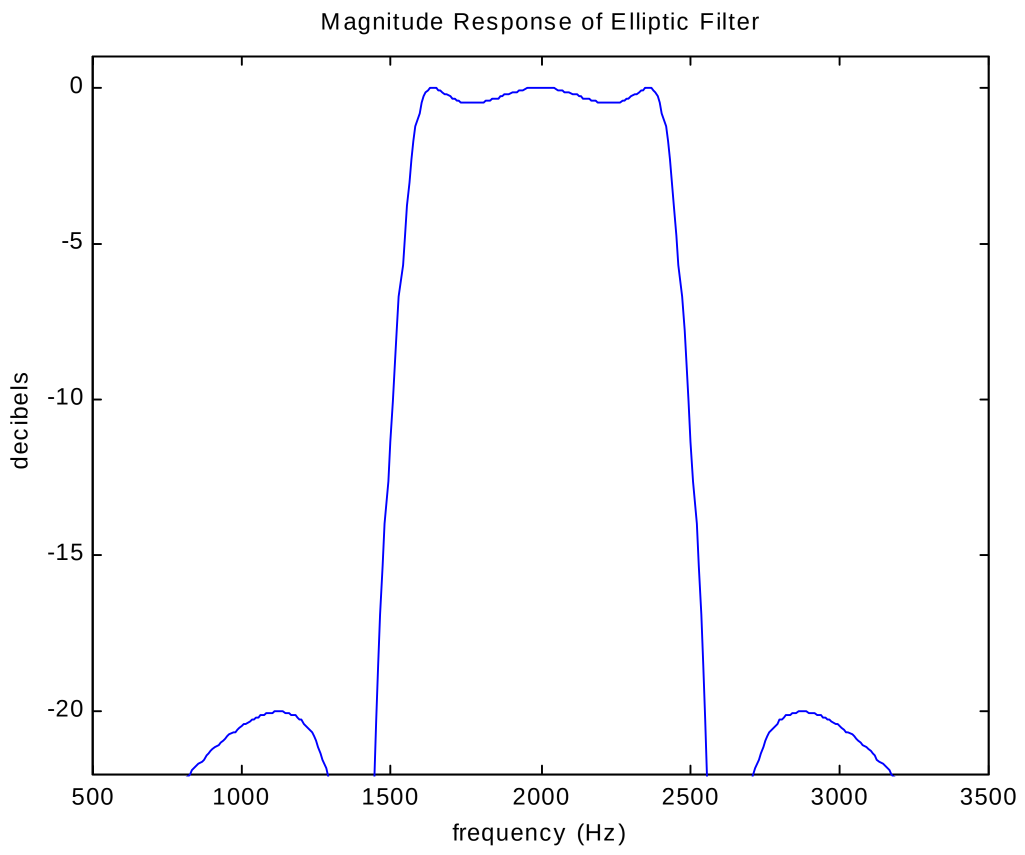
The following figure shows the magnitude response of the highpass filter.



Bandpass filters are found using these same two functions. However, with bandpass filters, the passband and stopband frequencies (w_p and w_s) are two-element vectors since there are two passband frequencies and two stopband frequencies. The Matlab code below shows the design of an elliptic digital bandpass filter.

```
% Bandpass Elliptic Digital Filter

ws = [0.3*pi 0.75*pi]              %Stopband edge frequency
wp = [0.4*pi 0.6*pi]               %Passband edge frequency
Rp = 0.5;                          %Passband ripple in dB
As = 20;                           %Stopband attenuation in dB

[N,wn] = ellipord(wp/pi,ws/pi,Rp,As);
[b,a] = ellip(N,Rp,As,wn);

[db,mag,pha,grd,w] = freqz_m(b,a);
plot(w*8000/2/pi,db);
axis([500 3500 -22 1]);
xlabel('frequency (Hz)'); ylabel('decibels'); title('Magnitude
Response of Elliptic Filter');
```

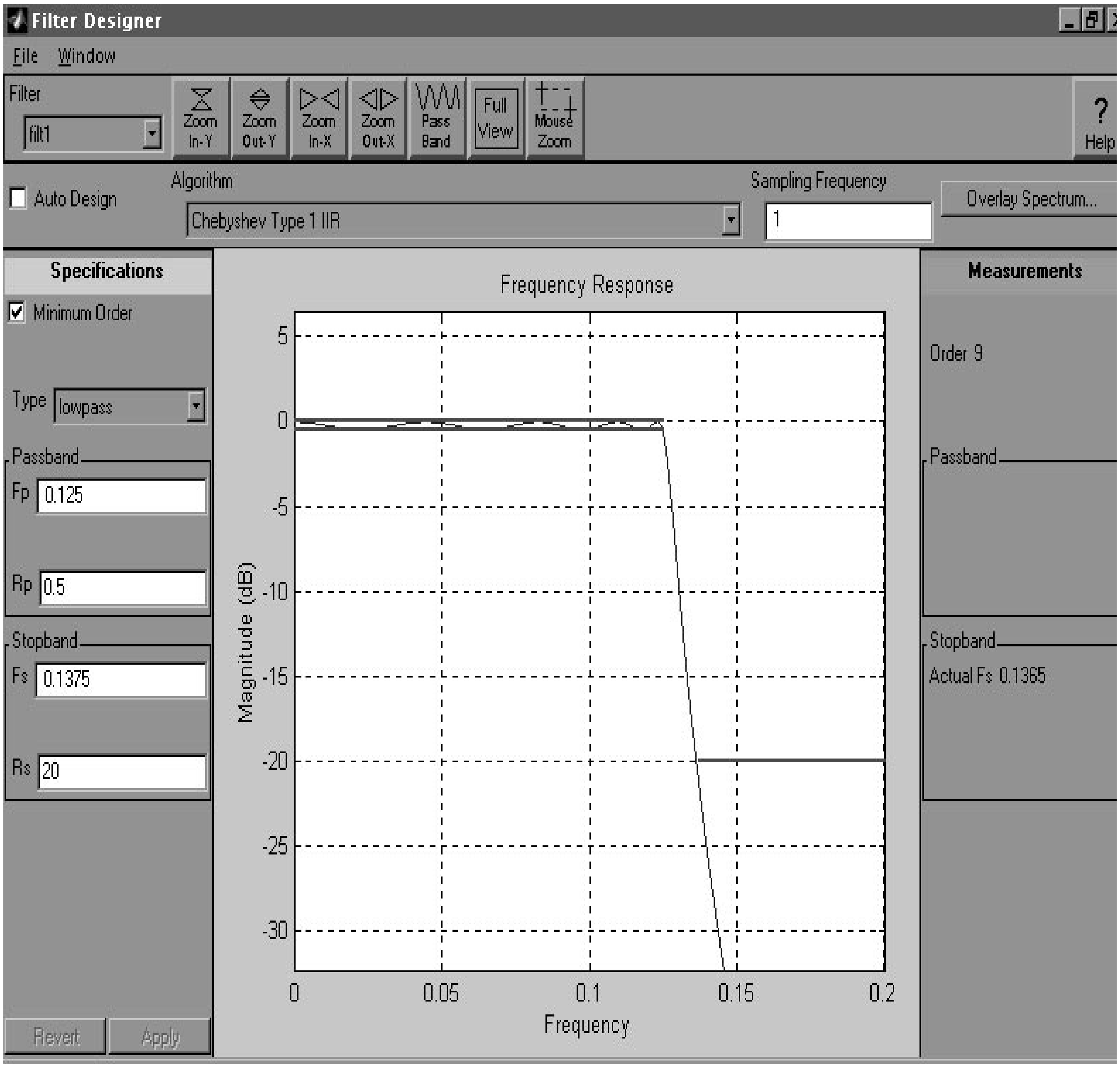
The following figure shows the magnitude response of the bandpass filter designed in the Matlab code above.



Sptool

Matlab has a very useful visualization tool for designing and analyzing digital filters called Signal Processing Tool, or Sptool. Sptool is a graphical user interface capable of analyzing and manipulating signals, filters, and spectra. For filter design, Sptool allows the user to select the filter design algorithm to use when creating the filter. The design algorithms included in Sptool are FIR filters (equiripple, least squares, Kaiser window) and IIR filters (Butterworth, Chebyshev type 1 and 2, elliptic). The user is also allowed to specify the type of filter (lowpass, bandpass, highpass, or bandstop). Sptool designs the filter and displays the magnitude response and the filter order.

The figures below show actual Sptool screenshots for a lowpass filter design using the specifications given above. The Chebyshev Type 1 algorithm was used for these screenshots. By using the zoom options, different parts of the spectrum can be analyzed quickly and easily with Sptool. The second screenshot is a windowed view of the passband of the spectrum contained in the first screenshot.



Filter: filt1

☐ Auto Design

Algorithm: Chebyshev Type 1 IIR

Sampling Frequency: 1

Specifications

☑ Minimum Order

Type: lowpass

Passband

Fp: 0.125

Rp: 0.5

Stopband

Fs: 0.1375

Rs: 20

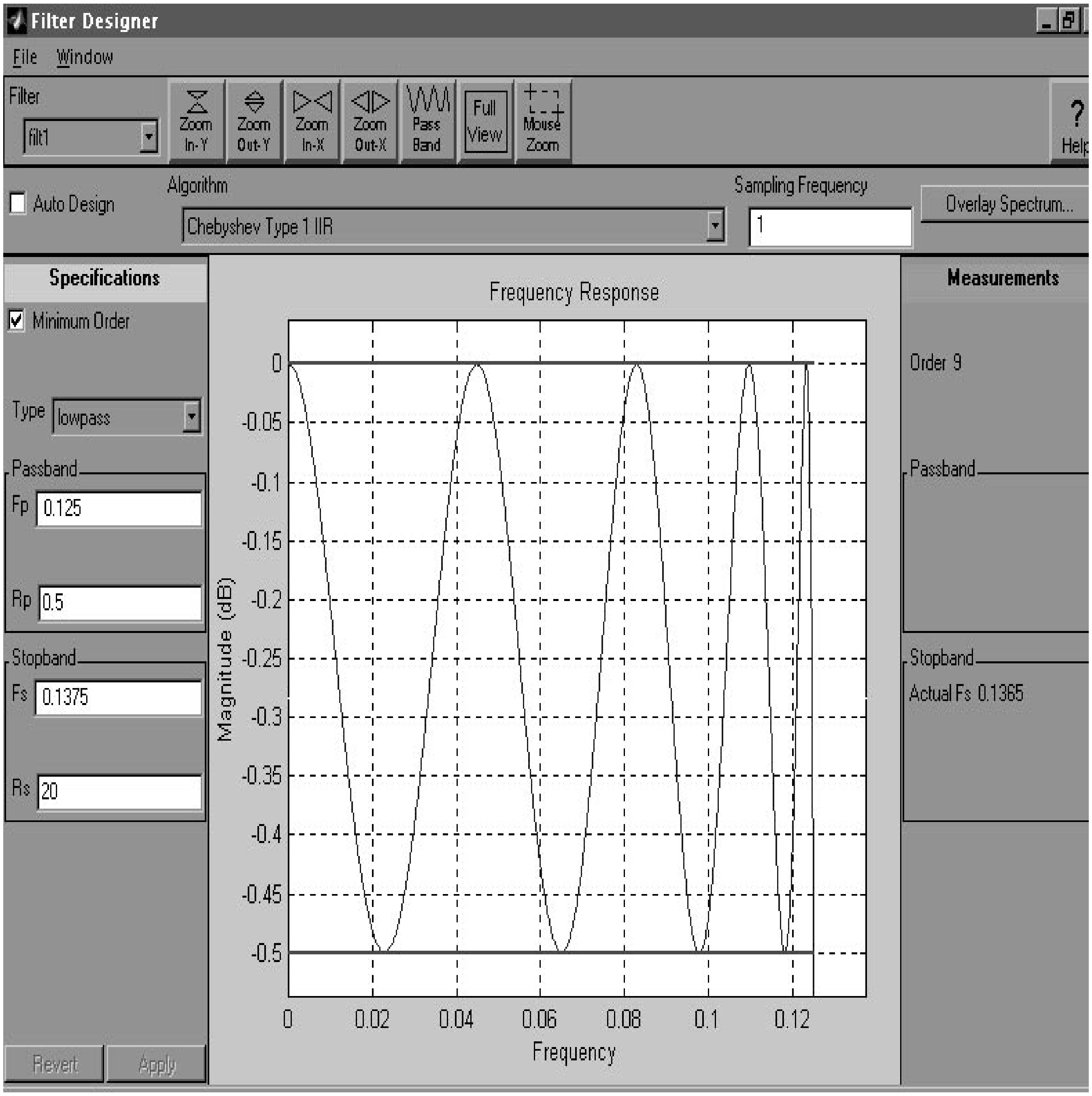
Measurements

Order 9

Passband

Stopband

Actual Fs 0.1365



Filter: filt1

[ ] Auto Design

Algorithm: Chebyshev Type 1 IIR

Sampling Frequency: 1

Overlay Spectrum...

Specifications

[✓] Minimum Order

Type: lowpass

Passband
Fp: 0.125
Rp: 0.5

Stopband
Fs: 0.1375
Rs: 20

Measurements

Order 9

Passband

Stopband
Actual Fs 0.1365

Future Directions

Digital filters can be quickly and easily designed in Matlab using the methods described above. Sptool offers a much quicker way of designing and comparing different filters than using the function calls to the filter algorithms. However, Sptool allows the user to view the magnitude response of the filter, not the phase response. For some applications, the phase response may be more important than the magnitude response, so in these cases Sptool would not be as useful in determining the optimum filter design. Also, Sptool does not give a direct output of the filter coefficients. With the Matlab code given above, the filter coefficient are displayed to the user.

The results from Matlab can be used directly to implement the digital filters on real DSPs. These results are all that is needed to draw a complete signal flow graph with adders, multipliers, and delay elements. The filter coefficients are used as the gain factors for the multipliers, and shift registers are used as the delay elements (for each z^{-n} factor).

Acknowledgments

Many thanks to Dr. Joseph Picone for his guidance, assistance, and time in the execution of this project.

References

- Hanselman, Duane, and Littlefield, Bruce. Mastering Matlab 5. Prentice Hall. Upper Saddle River, NJ, 1998.

- Ingle, Vinay K. and Proakis, John G. Digital Signal Processing Using Matlab. PWS Publishing Company, 1997.

- Proakis, John G. and Manolakis, Dimitris G. Digital Signal Processing: Principles, Algorithms, and Applications, 3rd Edition. Prentice Hall. Upper Saddle River, NJ, 1996.

- Ziemer, Rodger E., Tranter, William H., and Fannin, D. Ronald. Signals and Systems: Continuous and Discrete, 3rd Edition. Macmillan Publishing Company, 1993.

Appendix

```
function [b,a] = chb1(Wp, Ws, Rp, As);
% Analog Lowpass Filter Design:  Chebyshev-1
%
% [b,a] = chb1(Wp, Ws, Rp, As);
%  b = Numerator coefficients of Ha(s)
%  a = Denominator coefficients of Ha(s)
%  Wp = Passband edge frequency in rad/sec
%  Ws = Stopband edge frequency in rad/sec
%  Rp = Passband ripple in dB
%  As = Stopband attenuation in dB
%
if Wp <= 0
   error('Passband edge must be larger than 0')
end
if Ws <= Wp
   error('Stopband edge must be larger than Passband edge')
end
if (Rp <= 0) | (As < 0)
   error('PB ripple and/or SB attenuation must be larger than 0')
end
ep = sqrt(10^(Rp/10)-1);
A = 10^(As/20);
OmegaC = Wp;
OmegaR = Ws/Wp;
g = sqrt(A*A-1)/ep;
N = ceil(log10(g+sqrt(g*g-1))/log10(OmegaR+sqrt(OmegaR*OmegaR-1)));
fprintf('\n*** Chebyshev-1 Filter Order = %2.0f \n',N);
[b,a] = ap_chb1(N, Rp, OmegaC);


function [b,a] = ap_chb1(N, Rp, Omegac);
% Chebyshev-1 Analog Lowpass Filter Prototype
%
% [b,a] = ap_chb1(N, Rp, Omegac);
%  b = nemerator polynomial coefficients
%  a = denominator polynomial coefficients
%  N = Order of the elliptical filter
%  Rp = Passband Ripple in dB
%  Omegac = cutoff frequency in rad/sec
%
[z,p,k] = cheb1ap(N,Rp);
   a = real(poly(p));
   aNn = a(N+1);
   p = p*Omegac;
   a = real(poly(p));
   aNu = a(N+1);
   k = k*aNu/aNn;
   b0 = k;
   B = real(poly(z));
   b = k*B;
```

Chapter-5

Analog filter design:

MATLAB has a variety of functions in its signal processing toolbox which support the design and analysis of analog and digital filters. We shall begin with analog filters and take a look at the most commonly used approximations, namely,

- Butterworth filters
- Chebyshev filters
- Elliptic filters

These filters are typically specified by their cutoff frequencies and ripples on both the stopband and passband. The design process starts with the determination of the minimum order of the filter and the appropriate cutoff frequency to meet the desired specifications. These two parameters can be estimated via the following commands:

```
>>[N,Wc]=buttord(Wp,Ws,Rp,Rs,'s');          %Butterworth filter
>>[N,Wc]=cheb1ord(Wp,Ws,Rp,Rs,'s');         %Chebyshev Type-I filter
>>[N,Wc]=cheb2ord(Wp,Ws,Rp,Rs,'s');         %Chebyshev Type-II filter
>>[N,Wc]=ellipord(Wp,Ws,Rp,Rs,'s');         %Elliptic filter
```

N = lowest order of the filter
Wc = cutoff frequency in rad/s
Rp = passband ripple in dB
Rs = minimum stopband attenuation in dB
Wp, Ws = passband and stopband edge frequencies, respectively.

Practice:

Using MATLAB, determine the order (N) and cutoff frequency (Wc) of a lowpass Butterworth filter with the following specifications:

```
Wp =10  rad/s          %passband critical frequency
Ws = 20 rad/s          %stopband critical frequency
Rp = -2 dB             %passband ripple
Rs = -20 dB            %stopband attenuation
```

```
>>Wp = 10;  Ws = 20;  Rp = -2;  Rs = -20;
>>[N,Wc] = buttord(Wp,Ws, -Rp, -Rs, 's')

N =

   4


Wc =
  11.2610
```

Normalized lowpass analog filters:

Analog filters are typically designed as normalized (cutoff frequency of 1 rad/s) lowpass filters and then transformed to the specific frequency and filter type (highpass, bandpass, etc.) with direct substituion.

The functions **buttap**, **cheb1ap**, **cheb2ap**, **ellipap**, can be used to design normalized analog lowpass Butterworth, Chebyshev Type-I, Chebyshev Type-II, and elliptic filters, respectively.

Syntax:

```
>>[z,p,k]=buttap(n);           %normalized Butterworth filter
>>[z,p,k]=cheb1ap(N,Rp);       %normalized Chebyshev Type-I filter
>>[z,p,k]=cheb2ap(N,Rs);       %normalized Chebyshev Type-II filter
>>[z,p,k]=ellipap(N,Rp,Rs);    %normalized elliptic filter
```

Rp = ripple in the passband (in dB)
Rs = ripple in the stopband (in dB)
N = order of the filter
z =vector containing the zeros
P = vector containing the poles
k = gain factor

Practice:

Using MATLAB, find the poles, zeros, and the gain factor of a normalized 5th-order Butterworth filter.

```
>>[z, p, k] = buttap(5)
z =

   []


p =

  -0.3090 + 0.9511i
  -0.3090 - 0.9511i
  -0.8090 + 0.5878i
  -0.8090 - 0.5878i
  -1.0000


k =

   1
```

Practice:

Using MATLAB, find poles, zeros, and the gain factor of a normalized 4th-order normalized Chebyshev Type-I filter with Rp = 2 dB.

```
>>[z,p,k]=cheb1ap(4,2)

z =

    []


p =

  -0.1049 + 0.9580i
  -0.2532 + 0.3968i
  -0.2532 - 0.3968i
  -0.1049 - 0.9580i


k =

   0.1634
```

It is often the case to specify the designed analog filter in terms of its transfer function. The function **zp2tf** converts the zeros, poles, and gain characterization of the filter to a rational function form (transfer function).

Syntax:

```
>>[num, den] = zp2tf(z,p,k);
```

where **num** contains the numerator coefficients and **den** the denominator coefficients. Let us give it a try!

Practice:

Using MATLAB, determine the transfer function of a 4th-order normalized Chebyshev Type-I lowpass filter with Rp=2 dB.

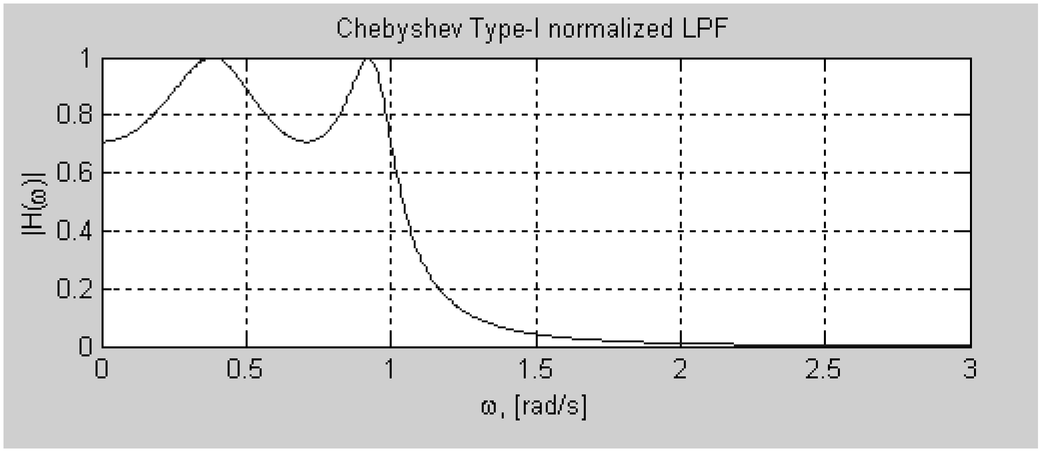
```
>>[z,p,k] = cheb1ap(4,2);          %poles, zeros, and gain specs of Chebyshev filter
>>[num,den]=zp2tf(z,p,k)           %convert to rational function form
>>tf(num,den)                      %print the transfer function

Transfer function:
                0.1634
-----------------------------------------------
s^4 + 0.7162 s^3 + 1.256 s^2 + 0.5168 s + 0.2058
```

Frequency response of analog filters:

Practice:

Design a 4th-order Chebyshev Type-I normalized lowpass filter with 3-dB ripple in the passband. Sketch its magnitude and phase responses.

```
>>[z,p,k]=cheb1ap(4,3);                %pole, zero, and gain specs of filter
>>[num,den]=zp2tf(z,p,k);              %numerator and denominator of H(s)
>>disp('num='); disp(num')             %display numerator coefficients
>>disp('den=');disp(den')              %display denominator coefficients
>>tf(num,den)                          %print the transfer function H(s)
>>w=0:0.005:3;                         %frequency vector
>>mag=abs(H);                          %compute magnitude response
>>phase=angle(H)*180/pi;               %compute phase response
>>subplot(2,1,1);plot(w,mag);grid      %plot magnitude response
>>%subplot(2,1,2);plot(w,phase);grid   %plot phase response
>>xlabel('\omega,[rad/s]')             %label the horizontal axis
>>ylabel('|H(\omega)|')                %label the vertical axis
>>title('Chebyshev Type-I normalized LPF')
```



Frequency transformations:

The previous filters are designed for the normalized cutoff frequency (1 rad/s). We shall now consider a number of transformations that will allow us to alter the cutoff frequency, and obtain lowpass, highpass, bandpass, and bandstop filters.

The commands **lp2lp**, **lp2hp**, **lp2bp**, and **lp2bs**, provide lowpass-to-lowpass, lowpass-to-highpass, lowpass-to-bandpass, and lowpass-to-bandstop transformations, respectively.

Syntax:

```
>>[num,den]=lp2lp(num,den,Wc);         %lowpass-to-lowpass
>>[num,den]=lp2hp(num,den,Wc);         %lowpass-to-highpass
>>[num,den]=lp2bp(num,den,W0,Bw);      %lowpass-to-bandpass
>>[num,den]=lp2bs(num,den,W0,Bw);      %lowpass-to-bandstop
```

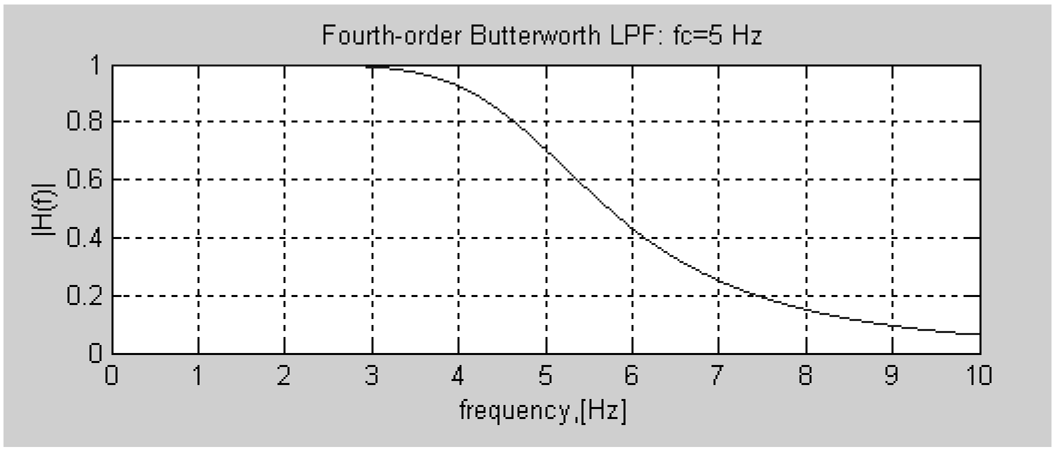
Wc = cutoff frequency of the filter
W0 = center frequency of the filter (bandpass and bandstop)
Bw = bandwidth of the filter (bandpass and bandstop)

Practice:

Design a fourth-order Butterworth lowpass filter with a cutoff frequency of 5 Hz. The procedure is as follows:

- ❑ Design a fourth-order normalized Butterworth filter using **buttap**.
- ❑ Apply the frequency transformation **lp2lp** to achieve the desired specifications.

```
>>[z,p,k]=buttap(4);                %fourth-order Butterworth filter
>>[num,den]=zp2tf(z,p,k);           %convert to polynomial form
>>tf(num,den)                       %print the transfer function H(s)
>>wc=2*pi*5;                        %specify cutoff frequency in rad/s
>>[num,den]= lp2lp(num,den,wc);     %desired filter
>>w=0:pi/100:10*pi;                 %define a frequency vector
>>H=freqs(num,den,w);               %compute frequency response
>>mag=abs(H);                       %compute magnitude response
>>plot(w/2/pi, mag);                %plot magnitude response versus frequency (Hz)
>>xlabel('frequency, [Hz]')         %label the horizontal axis
>>ylabel('|H(f)|')                  %label the vertical axis
>>title('Fourth-order Butterworth LPF: fc= 5 Hz')
```

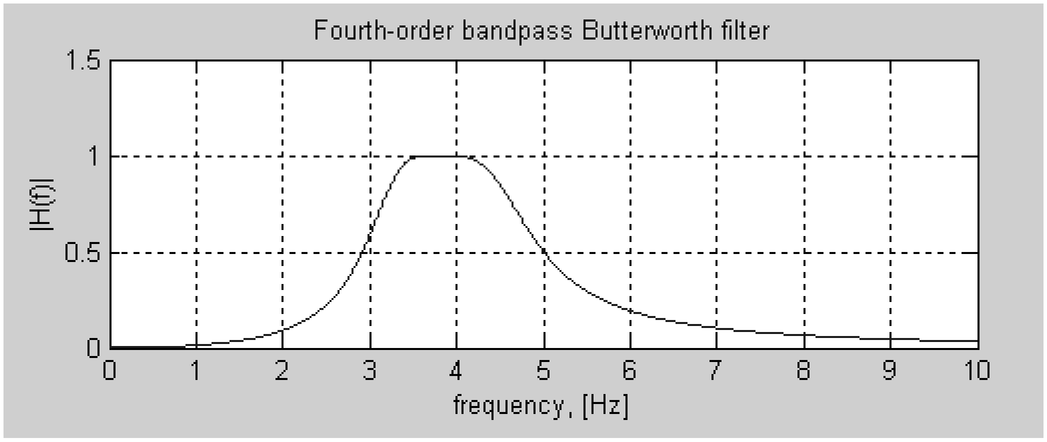


Practice:

Design a fourth-order Butterworth bandpass filter with a center frequency W0= 24 rad/s and bandwidth Bw= 10 rad/s.

```
>>W0= 24; Bw=10;                    %specify cutoff frequency and bandwidth
>>[z,p,k]=buttap(4);                %specify the Butterworth filter
>>[num,den]=zp2tf(z,p,k);           %specify filter in polynomial form
>>[num,den]=lp2bp(num,den,W0,Bw);   %convert  LPF to BPF
>>tf(num,den)                       %print the transfer function of desired filter
>>w=0:pi/1000;20*pi;                %specify frequency vector
>>H=freqs(num,den,w);               %compute the frequency response
```

```
>>mag=abs(H);                           %compute the magnitude response

>>plot(w/2/pi, mag);                    %plot the magnitude response
>>xlabel('frequency, [Hz]')             %label horizontal axis
>>ylabel('|H(f)|')                      %label vertical axis
>>title('Fourth-order bandpass Butterworth filter')
```



The signal processing toolbox offers more straightforward command for the design of analog filters.

- Butterworth analog filters:

```
>>[num,den]=butter(N,Wc, 's');          %Lowpass Butterworth filter
>>[num,den]=butter(N,Wc,'s');           %Bandpass Butterworth filter (order=2N)
>>[num,den]=butter(N,Wc,'high','s');    %Highpass Butterworth filter
>>[num,den]=butter(N,Wc,'stop','s');    %Bandstop Butterworth filter

>>[z,p,k]=butter(N,Wc,'s');             %Lowpass Butterworth filter
>>[z,p,k]=butter(N,Wc,'s');             %Bandpass Butterworth filter
>>[z,p,k]=butter(N,Wc,'high','s');      %Highpass Butterworth filter
>>[z,p,k]=butter(N,Wc,'stop','s');      %Bandstop Butterworth filter
```

N=order of the filter
Wc=cutoff frequency. For bandstop and bandpass Wc=[Wc1,Wc2]
z= zeros
p= poles
k= gain

Practice:

Design an 8th-order Butterworth bandpass filter with Wc=$\left[2\pi,5\pi\right]$. Sketch the magnitude and phase responses.

```
>>Wc=[2*pi,5*pi];                       %specify critical frequencies
>>[num,den]=butter(w,Wc,'s');           %specify the filter in polynomial form
>>[H,w]=freqs(num,den);                 %computer the frequency response
```

```
>>mag=abs(H);                    %extract the magnitude response
>>phase=angle(H)*180/pi;         %extract the phase response in degrees
>>plot(w,mag);grid               %plot magnitude response
```

- Chebyshev analog filters

The following functions can be used for the design of Type-I and Type-II Chebyshev filters:

Type-I:

```
>>[num,den]=cheby1(N, Rp,Wc,'s');             %LPF: Rp=ripples (dB) in passband
>>[num,den]=cheby1(N,Rp,Wc,'high','s');       %HPF
>>[num,den]=cheby1(N,Rp,Wc,'stop','s');       %BSF: Wc=[Wc1,Wc2]
>>[num,den]=cheby1(N,Rp,Wc,'s');              %BPF: Wc=[Wc1,Wc2]

>>[z,p,k]=cheby1(N,Rp,Wc,'s');                %LPF
>>[z,p,k]=cheby1(N,Rp,Wc,'high','s');         %HPF
>>[z,p,k]=cheby1(N,Rp,Wc,'stop','s');         %BSF
>>[z,p,k]=cheby1(N,Rp,Wc,'s');                %BPF
```

Type-II:

```
>>[num,den]=cheby2(N,Rs,Wc,'s');              %LPF: Rs=decibels down
>>[num,den]=cheby2(N,Rs,Wc,'high','s');       %HPF
>>[num,den]=cheby2(N,Rs,Wc,'stop','s');       %BSF: Wc=[Wc1,Wc2]
>>[num,den]=cheby2(N,Rs,Wc,'s');              %BPF: Wc=[Wc1,Wc2]

>>[z,p,k]=cheby2(N,Rs,Wc,'s');                %LPF
>>[z,p,k]=cheby2(N,Rs,Wc,'high','s');         %HPF
>>[z,p,k]=cheby2(N,Rs,Wc,'stop','s');         %BSF
>>[z,p,k]=cheby2(N,Rs,Wc,'s');                %BPF
```

Digital filter design:

There are broadly two kinds of digital filters, usually classified according to the duration of their impulse response, which can be either of finite length or infinite length. Filters having a finite duration impulse response are called Finite Impulse Response Filters or FIR filters; and filters with an infinite duration impulse response are called Infinite Impulse Response Filters or IIR filters.

Note: Normalized Frequency

MATLAB uses a somewhat non-standard form for specifying frequencies for digital filters. The digital frequency axis which, we usually consider as ranging over the interval $\left[0,2\pi\right]$ *is specified in MATLAB digital filter functions as [0,2], with 1.0 corresponding to half the sampling rate.*
Plots of frequency response obtained with MATLAB use the term "normalized frequency" for a frequency which is normalized with respect to ***half the sampling frequency****. Just as analog frequency can be expressed in* ***rad/s*** *as well as* ***Hz****, digital frequency can be expressed in* ***radian/sample*** *as well as in* ***cycles per sample****.*

$$\Omega_d = \frac{\omega_a}{f_s} = \omega_a t_s, \qquad F_d = \frac{1}{T_d}$$

T_d : samples/radian

Practice:

If a 8 Hz sinusoidal signal is sampled at 32 Hz, what is its digital frequency?

$$\therefore F_d = \frac{8}{32} = 0.25 \text{ cycles/sample} \Leftrightarrow \Omega_d = 2\pi\left(0.25\right) = 1.571 \text{ rad/sample}$$

- ❑ Infinite Impulse Response Filters

The first step is to determine the minimum order and cutoff frequency of the filter required to meet the desired specifications. The second step is to pass this order and cutoff frequency to the design functions **butter**, **cheby1**, **cheby2**, or **ellip**. The following commands give estimates of the filter's order and cutoff frequency:

```
>>[N,Wc]=buttord(Wp,Ws, Rp,Rs);          %Butterworth filter
>>[N,Wc]=cheb1ord(Wp,Ws,Rp,Rs);          %Chebyshev Type-I
>>[N,Wc]=cheb2ord(Wp,Ws,Rp,Rs);          %Chebyshev Type-II
>>[N,Wc]=ellipord(Wp,Ws,Rp,Rs);          %Elliptic
```

N= lowest order of the digital IIR filter
Wc=cutoff frequency
Rp=passband ripple in dB
Rs=minimum stopband attenuation in dB
Wp,Ws=passband and stopband edges frequencies, such that 0<Wp,Ws<1 where 1 corresponds to pi radians.

Practice:

Find the order and cutoff frequency of a Butterworth digital lowpass filter with no more than 1 dB attenuation below 3400 Hz and at least 40 dB attenuation at and beyond 3600 Hz. The sampling frequency is 8000 Hz.

```
>>Fs=8000;                  %specify sampling rate
>>Fn=Fs/2;                  %specify Nyquist frequency
>>wp=3400/Fn;               %normalized passband frequency
>>ws=3600/Fn;               %normalized stopband frequency
>>Rp=1; Rs=40;              %passband and stopband attenuation
>>[N,wc]=buttord(wp,ws,Rp,Rs)
```

We shall now consider two techniques for designing IIR filters, namely,

- ❑ Bilinear transformation method
- ❑ Impulse invariance method

Bilinear transformation method:

In this section we start with the design of IIR filters based on the Bilinear Transformation Method (BTM). The bilinear transformation method is a standard procedure of mapping the s-plane into the z-plane. It transforms an analog filter, designed via conventional techniques, into a discrete equivalent. The process begins with the design of a normalized lowpass analog filter. The command **buttap**, **cheb1ap**, **cheb2ap**, and **ellipap** provide Butterworth, Chebyshev Type1, Chebyshev Type II, and elliptic analog filters, respectively. Then, frequency transformations are applied to the normalized filter to yield lowpass, bandpass, highpass or bandstop filter with the desired design frequency requirements. The resulting transfer function is mapped into a digital filter transfer function via the **bilinear** command.

Syntax:

```
>>[numz,denz]=bilinear(nums,dens,Fs);           %Fs=sampling frequency
>>[numz,denz]=bilinear(nums,dens,Fs,Fp);        %Fp=specifies prewarping in Hz
```

numz, **denz**: numerator and denominator of the transfer function H(z)
nums,**dens**: numerator and denominator of the transfer function H(s)

Practice:

Design a 4-th order Butterworth lowpass digital filter with a cutoff frequency of 3 rad/s and sampling rate of 10 Hz.

```
>>[z,p,k]=buttap(4);                    %normalized 4-th order Butterworth filter
>>[num,den]=zp2tf(z,p,k);               %polynomial specification of the filter
>>wc=3;                                 %specify the cutoff frequency
>>[num,den]=lp2lp(num,den,wc);          %scale the normalized filter by wc
>>Fs=10;                                %sampling frequency
>>[numd,dend]=bilinear(num,den,Fs);     %analog to digital mapping
>>disp('numd='), disp(numd')            %print numerator coefficients in column vector
>>disp('dend='), disp(dend')            %print denominator coefficients in column vector
>>[Hd,wd]=freqz(numd,dend);             %determine the frequency response
>>magd=abs(Hd);                         %compute magnitude response of digital filter
>>subplot(2,1,1);plot(wd/pi,mag);grid;  %plot the magnitude response
>>title('Lowpass Butterworth digital filter');
>>ylabel('Amplitude response')
>>axis([0 0.2 0 1])
>>subplot(2,1,2);plot(wd/pi,angle(Hd));grid
>>axis([0 0.2 –200 200])
>>ylabel('Phase response')
>>xlabel('Frequency, [in units of pi]')
```

The signal processing toolbox includes **butter**, **cheby1**, **cheby2**, and **ellip** functions that automate the design of IIR filters via the bilinear transformation method.

```
[num,den]=butter(n,wn);                 %LPF: 0<wn<1
[num,den]=butter(n,wn,'high');          %HPF: 0<wn<1
[num,den]=butter(n,wn);                 %BPF: wn=[w1,w2], 0<w1,w2<1
[num,den]=butter(n,wn,'stop');          %BSF: wn=[w1,w2],0<w1,w2<1
[num,den]=cheby1(n,Rp,wn);              %LPF: 0<wn<1
```

```
[num,den]=cheby2(n,Rs,wn);              %LPF: 0<wn<1
[num,den]=cheby1(n,Rp,wn,'high');       %HPF: 0<wn<1
[num,den]=cheby2(n,Rs,wn,'high');       %HPF: 0<wn<1
[num,den]=cheby1(n,Rp,wn,'stop');       %BSF: wn=[w1,w2], 0<w1,w2<1
[num,den]=cheby2(n,Rs,wn,'stop');       %BSF: wn=[w1,w2], 0<w1,w2<1
[num,den]=cheby1(n,Rp,wn);              %BPF: wn=[w1,w2], 0<w1,w2<1
[num,den]=cheby2(n,Rs,wn);              %BPF: wn=[w1,w2], 0<w1,w2<1
[num,den]=ellip(n,Rp,Rs,wn);            %LPF: 0<wn<1
[num,den]=ellip(n,Rp,Rs,wn,'high');     %HPF: 0<wn<1
[num,den]=ellip(n,Rp,Rs,wn,'stop');     %BSF: wn=[w1,w2], 0<w1,w2<1
[num,den]=ellip(n,Rp,Rs,wn);            %BPF: wn=[w1,w2], 0<w1,w2<1
```

Rp: Passband ripple in dB
Rs: Minimum stopband attenuation in dB
num: Numerator of transfer function
den: denominator of transfer function

The following functions can also be used to determine the poles, zeros and the gain of the transfer function.

```
>>[z.p,k]=butter(n,wn);
>>[z,p,k]=cheby1(n,Rp,wn);
>>[z,p,k]=cheby2(n,Rs,wn);
>>[z,p,k]=ellip(n,Rp,Rs,wn);
```

z: vector of zeros
p: vector of poles
k: gain of transfer function

Practice:

Design a 4-th order lowpass Butterworth filter having a cutoff frequency of 3 rad/s and Fs=10 Hz.

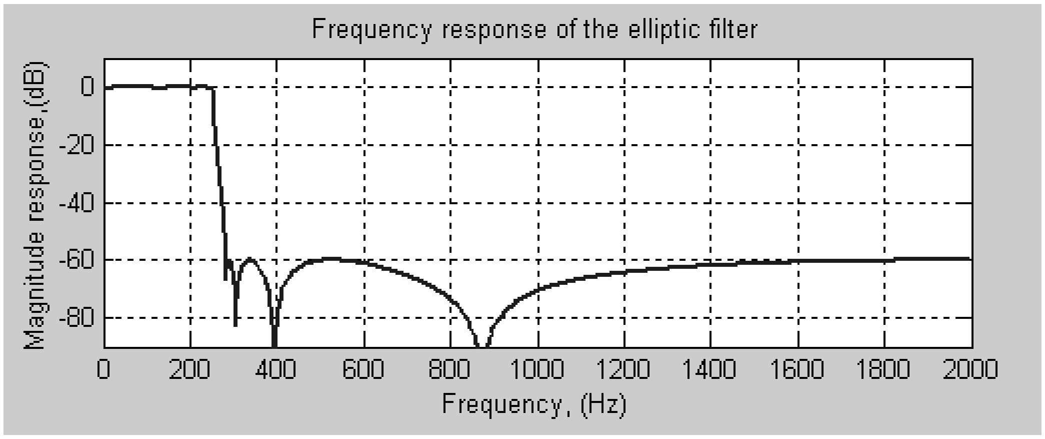
```
>>N=4;                                  %specify order of the filter
>>Fs=10;                                %specify the sampling rate
>>wc=3;                                 %specify the analog cutoff frequency
>>omegad=wc/(pi*Fs);                    %normalized digital cutoff frequency
>>[numd,dend]=butter(N,omegad);         %specify the digital filter transfer function
>>[Hd,wd]=freqz(numd,dend);             %provide the frequency response of digital filter
>>magd=abs(Hd);                         %compute the magnitude response
>>phase=angle(Hd)*180/pi;               %compute the phase response
>>subplot(2,1,1);plot(wd/pi,magd);grid; %plot magnitude response
>>subplot(2,1,2);plot(wd/pi,phase);grid; %plot the phase response
```

Practice:

Design an 8-th order elliptic lowpass filter with a cutoff frequency of 250 Hz, 0.5 dB of ripple in the passband and a minimum stopband attenuation of 60 dB to be used with a signal that is sampled at 4 kHz.

```
>>clear all;
```

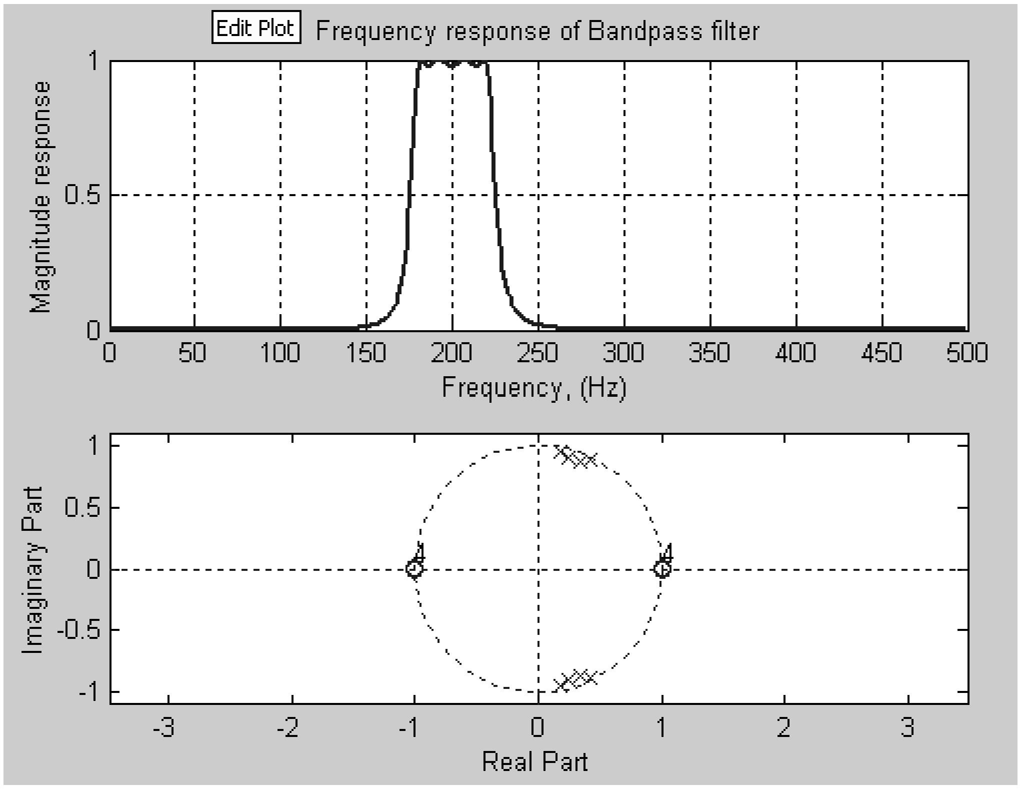
```
>>Fs=4000;
>>Rp=0.5; Rs=60;
>>Fc=250;
>>[numd,dend]=ellip(8,Rp,Rs, 2*Fc/Fs);
>>f=4*(0:500);
>>[H,f]=freqz(numd,dend,512,4000);
>>mag=20*log10(abs(H));
>>plot(f,mag,'LineWidth',2); grid
>>axis([0 2000 –90 10])
>>xlabel('Frequency, (Hz)')
>>ylabel('Magnitude response, (dB)')
>>title('Frequency response of the elliptic filter')
```



Practice:

A sequence has 1 kHz sampling. Design an 8-th order Chebyshev Type I bandpass filter that has a bandwidth of 40 Hz centered at 200 Hz, with 0.2 dB of ripple in the passband. Plot the magnitude response between 100 Hz and 300 Hz, and the pole-zero diagram.

```
Fs=1000;
[b,a]=cheby1(4,0.2,[180/500,220/500])
[H,F]=freqz(b,a,512,Fs);
subplot(2,1,1); plot(F,abs(H),'LineWidth',2); grid
title('Frequency response of Bandpass filter')
ylabel('Magnitude response')
xlabel('Frequency')
subplot(2,1,2);zplane(b,a)
```



Practice:

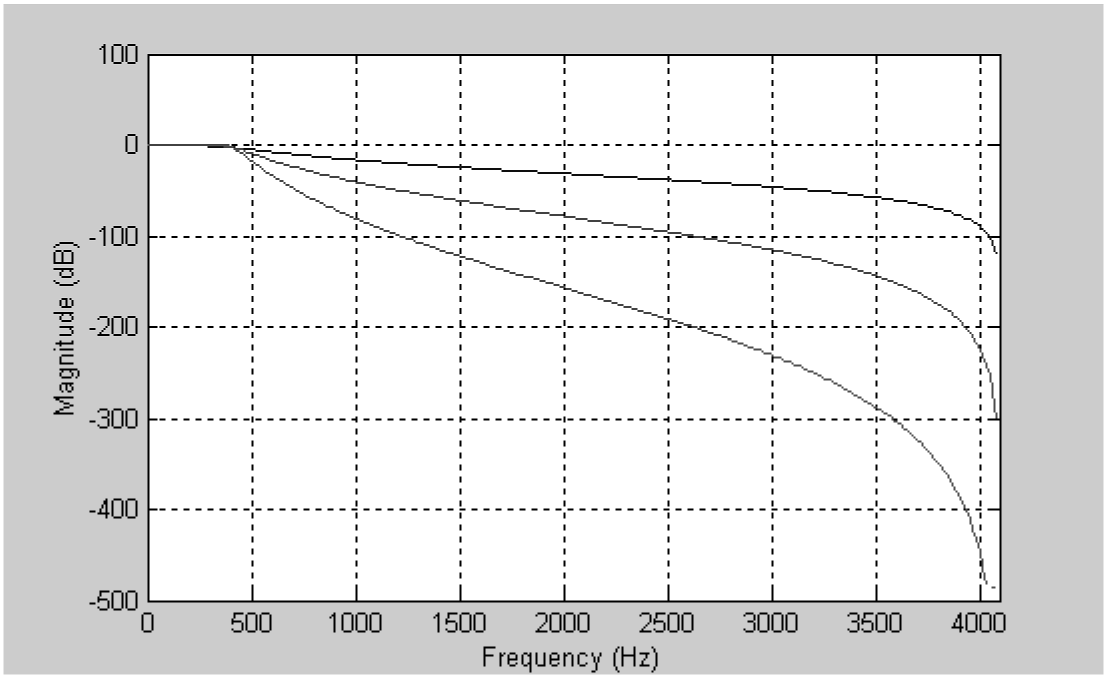
Design a 4-th order IIR Chebyshev type I bandpass filter with cutoff frequencies at 300 Hz and 3400 Hz, 0.5 dB of ripple in the passband to be used with a signal that is sampled at 8kHz.

```
>>Fs=8000;
>>Fn=Fs/2;
>>w1=300/4000;
>>w2=3400/4000;
>>wc=[w1 w2];
>>order=4
>>ripple=0.5;
>>[b a]=cheby1(order, ripple,wc];
>>w=0:pi/99:99;
>>H=freqz(b,a,w);
>>mag=20*log10(abs(H));
>>phase=angle(H);
>>tol=0.95*pi;
>>phase=unwrap(phase,tol);
>>title('Magnitude plot for IIR filter')
>>xlabel('Frequency (radians)')
>>ylabel('Magnitude (dB)')
```

Practice:

Plot a family of three Butterworth lowpass filters:

```
>>[b1,a1]=butter(2,0.1);
>>[H1,f,s]=freqz(b1,a1,256,8192);
>>[b2,a2]=butter(5,0.1);
>>H2=freqz(b2,a2,256,8192);
>>[b3,a3]=butter(10,0.1);
>>H3=freqz(b3,a3,256,8192);
>>H=[H1 H2 H3];
>>s.plot='mag';
>>freqzplot(H,f,s)
```



<u>MATLAB *filter*:</u>

Syntax:

```
>>y=filter(b,a,x);
```

filters the data in vector **x** (input) with the filter described by coefficients vectors **a** (den) and **b** (num) to create the filtered data vector **y** (output). The command **filter** works for both real and complex inputs. If a(1)$\neq$0, filter normalizes the filter coefficients by a(1).

<u>Practice:</u>

An LTI filter is described by a difference equation

$$y[n]-0.268y[n-2]=0.634x[n]+0.634x[n-2]$$

Determine the impulse response and sketch it.

```
>>a=[1  0  -0.268];           %vector corresponding to the coefficients of y’s
>>b=[0.634  0  0.634];        %vector corresponding to the coefficients of x’s
>>imp=[1 zeros(1,31)];        %impulse input
>>y=filter(b,a,imp);          %impulse response
>>stem(y)                     %plot impulse response
```

<u>**MATLAB *impz:***</u>

Syntax:

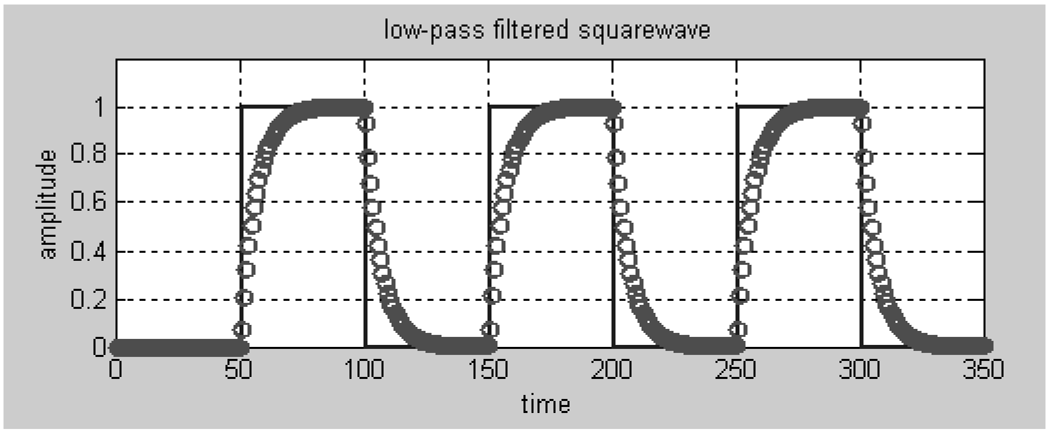
```
>>[h,t]=impz(b,a)
>>[h,t]=impz(b,a,n,Fs)            %compute n samples of the impulse response
>>[h,t]=impz(b,a,[ ],Fs)          %Fs: sampling rate
```

Computes the impulse response of the filter with numerator coefficients **b** and denominator coefficients **a**. The function **impz** chooses the number of samples and returns the response in column vector h and times in column vector t

<u>Practice:</u>

This script file takes a series of 3 pulses and passes them through a Butterworth filter, and then plots the results.

```
close all; clear all; clc;
t=(1:350);                        %time base
o=ones(1,50);                     %an array of 1’s
z=zeros(1,50);                    %an array of 0’s
x=[z o z o z o z];                %signal with three pulses
cutoff=0.05;                      %cutoff frequency of filter
[num,den]=butter(1,cutoff);       %Butterworth filter
y=filter(num,den,x);              %filtered output
plot(t,x,,t,y,’ro’,’LineWidth’,2); grid
axis([0 350 0 1.2])
xlabel(‘Time’)
ylabel(‘Amplitude’)
title(‘Low-pass filtered squarewave’)
```



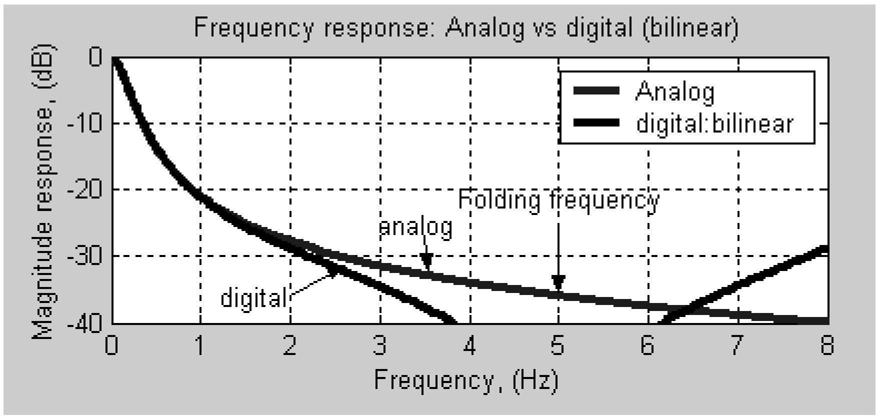
Practice:

An analog filter is described by its transfer function given by

$$H_a(s) = \frac{0.5(s+4)}{(s+1)(s+2)}$$

Assume the sampling frequency of 10 Hz and a matching frequency of 1 Hz.
Use the bilinear transformation method to find a digital filter equivalent.

```
>>num=0.5*[1 4];
>>den=conv([1 1],[1 2]);
>>[numd,dend]=bilinear(num,den,10,1);
>>[Ha,wa]=freqs(num,den);
>>[Hd,Fd]=freqz(numd,dend,256,'whole',10);
>>plot(wa/(2*pi),20*log10(abs(Ha/Ha(1))),'b','LineWidth',3);
>>hold on
>>plot(Fd,20*log10(abs(Hd/Hd(1))),'k','LineWidth'.3)
>>hold off
>>xlabel('Frequency, (Hz)')
>>ylabel('Magnitude response, (dB)')
>>title('Frequency response: Anaolg vs digital (bilinear)')
>>legend('Analog', 'Digital: bilinear')
```



Impulse Invariance method:

Syntax:

```
>>[numd, dend]=impinvar(num,den, Fs)
```

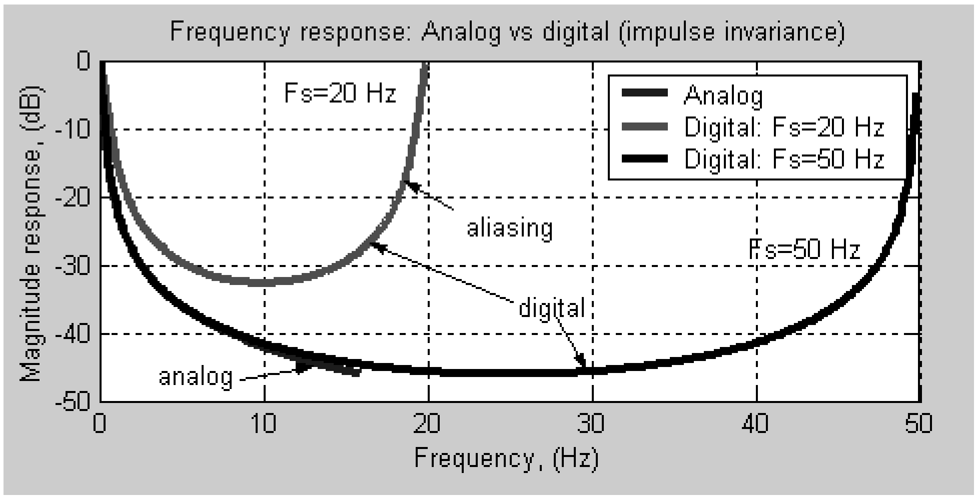
Practice:

An analog filter is specified by its transfer function given by

$$H_a(s) = \frac{0.5(s+4)}{(s+1)(s+2)}$$

Use the impulse invariance method to design its digital counterpart. Compare the analog filter to two versions of the digital filter; one with a sampling frequency of 20 Hz the other with a sampling frequency of 50 Hz.

```
>>Fs1=20; Fs2=50;
>>num=0.5*[1  4];
>>den=conv([1 1],[1 2]);
>>[numd,dend]=impinvar(num,den,Fs1)
>>[numdd,dendd]=impinvar(num,den,Fs2);
>>[H,w]=freqs(num,den);
>>[Hd1,Fd1]=freqz(numd,dend,256,'whole',20);
>>[Hd2,Fd2]=freqz(numdd,dendd,256,'whole',50);
>>plot(w/(2*pi),20*log10(abs(H)),'LineWidth',3);grid
>>hold on
>>plot(Fd1,20*log10(abs(Hd1/Hd1(1))),'g','LineWidth',3);
>>plot(Fd2,20*log10(abs(Hd2/Hd2(1))),'k','LineWidth',3);
>>Hold off
>>legend('Analog','Digital: Fs=20 Hz','Digital: Fs=50 Hz')
>>xlabel('Frequency,(Hz)')
>>ylabel('Magnitude response, (dB)')
>>title('Frequency response: Analog vs digital (impulse invariance)')
```



Practice:

A sequence has 1 kHz sampling frequency. Design a 6th order Chebyshev bandpass filter that has a bandwidth of 40 Hz centered at 200 Hz, with 0.2 dB ripple in the passband.

1. Deduce the difference equation describing the filter
2. Find the attenuation at 100 Hz, 180 Hz, 220 Hz, and 400 Hz

```
>>Fs=1e3;
>>Fn=Fs/2;
>>fc=[180 220]/Fn;
>>[b,a]=cheby1(3, 0.2,fc)

b =

   0.0019        0   -0.0056        0    0.0056        0   -0.0019


a =

   1.0000   -1.7288    3.5525   -3.1726    3.1034   -1.3156    0.6646
```

$$y[n]=0.0019x[n]-0.0056x[n-2]+0.0056x[n-4]-0.0019x[n-6]+$$
$$1.7288y[n-1]-3.5525y[n-2]+3.1726y[n-3]-3.1034y[n-4]+$$
$$1.3156y[n-5]-0.6646y[n-6]$$

```
>>HdB=20*log10(abs(freqz(b,a,[100 180 220 400],Fs)))

HdB =

  -48.1943   -0.2000   -0.2000  -69.4665
```

The Yule Walker Design Method:

The Yule walker method is a direct procedure for designing IIR digital filters. The desired response is specified numerically in the frequency domain, therefore we are not constrained to the standard types like lowpass, bandpass, etc.

Syntax:

```
>>[numd,dend]=yulewalk(N,f,m)
```

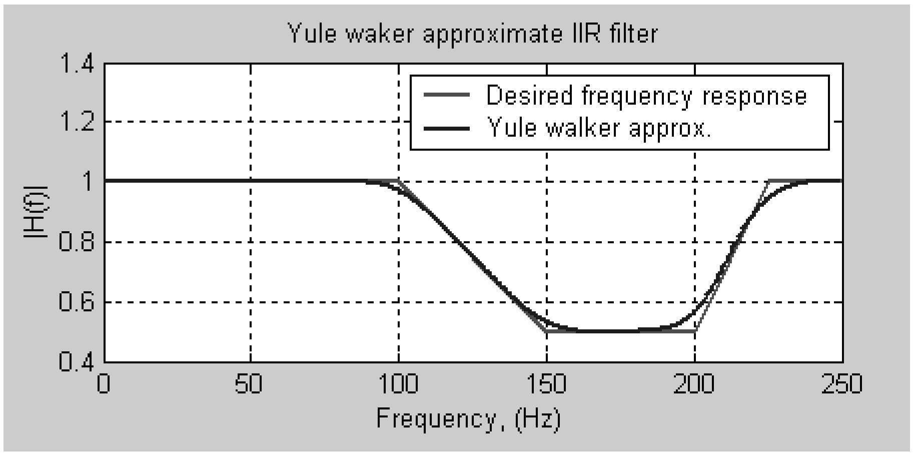
returns row vectors **numd**, and **dend** containing the (N+1) coefficients of the order **N** IIR filter whose frequency-magnitude characteristics approximately match those given in vectors **f** and **m**., where **f** is a vector of frequency points, ranging from 0 to 1, where 1 corresponds to the Nyquist frequency. The first point of **f** must be zero and the last point 1, with all intermediate points in increasing order. Duplicate frequency points are allowed, corresponding to steps in the frequency response. The vector **m** contains the desired magnitude response at the frequencies specified by **f**. The vectors **m** and **f** must be of the same length.

Practice:

Design an IIR filter having the following specifications with a sampling rate of Fs=500 Hz

From	**To (Hz)**	**Magnitude**
0	100	1
100	150	decrease linearly from 1 to 0.5
150	200	0.5
200	225	increase linearly from 0.5 to 1
225	250	1

```
>>clear all; clc;
>>f=[0 100 150 200 225 250];
>>m=[1 1 0.5 0.5 1 1];
>>plot(f,m,’r’,’LineWidth’,2); grid
>>hold on
>>fs=500;
>>f=f/(fs/2);
>>n=6;
>>[num,den]=yulewalk(n,f,m);
>>[H,w]=freqz(num,den);
>>plot(w/pi*fs/2,abs(H),’LineWidth’,2);
>>hold off
>>title(‘Yule walker approximate IIR filter’);
>>ylabel(‘|H(f)|’);
>>xlabel(‘Frequency,(Hz)’)
>>legend(‘Desired frequency response’,’Yule walker approx.’)
```

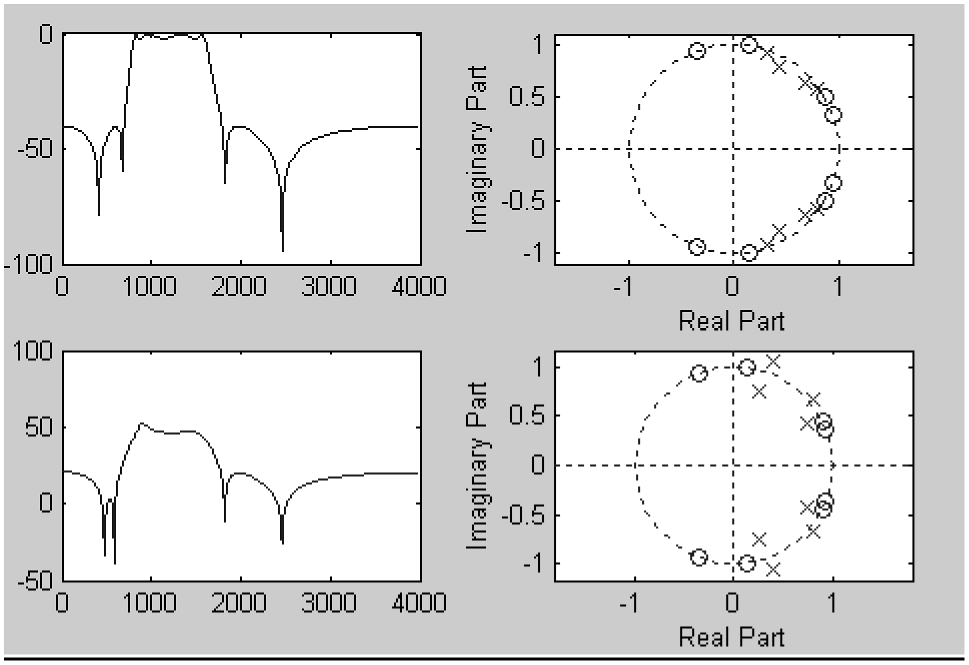


Effect of Coefficient Quantization in IIR Filters:

In the design of digital filters, the coefficients of the designed filters are typically quantized. This can cause the filter coefficient to be slightly inaccurate, thus yielding some deviations from the wanted frequency response, and even instability.

Practice:

```
>>[b,a]=ellip(4,2,40,[0.2 0.4]);
>>n=8;                                          %number of bits
>>bk=round(b/max(b)*2^n)/2^n*max(b);
>>ak=round(a/max(a)*2^n)/2^n*max(a);
>>[H,f]=freqz(b,a,512,8000);
>>[Hq,fq]=freqz(bk,ak,512,8000);
>>subplot(2,2,1); plot(f,20*log10(abs(H)));
>>subplot(2,2,3);plot(fq,20*log10(abs(Hk)));
>>subplot(2,2,2);zplane(b,a)
>>subplot(2,2,4); zplane(bk,ak)
```



Cacade implementation of IIR filters:

IIR filters are typically implemented by cascading low order sections, such as second-order sections (biquads). The overall transfer function is simply the product of the individual transfer functions.

Cascade implementations have better quantization properties than their finite-precision canonic equivalents. MATLAB function **zp2sos** converts pole-zero-gain representation of a given system to an equivalent second-order section representation. Similarly, the function **tf2sos** converts a digital filter transfer function parameters to second-order sections form.

Syntax:

```
>>[sos,g]=zp2sos(z,p,k)
>>[sos,g]=zp2sos(z,p,k,'order')
>>[sos,g]=zp2sos(z,p,k,'order','scale')
```

Returns a matrix **sos** in second-order section form with gain **g** equivalent to the zero-pole-gain system represented by input arguments **z**,**p**, and **k**.

```
>>[sos, g]=tf2sos(num,den)
```

Returns a matrix **sos** in second-order section form with gain **g** equivalent to the system specified by input arguments **num** and **den**.

$$H(z)=k\frac{(z-z_1)(z-z_2)....(z-z_n)}{(z-p_1)(z-p_2)....(z-p_m)}$$

Where **n** and **m** are the length of **z** and **p**, respectively, and **k** is a scalar gain. The poles and zeros must be real or complex conjugates. The matrix **sos** is of size L-by-6.

$$sos = \begin{bmatrix} b_{01} & b_{11} & b_{21} & 1 & a_{11} & a_{21} \\ b_{02} & b_{12} & b_{22} & 1 & a_{12} & a_{22} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ b_{0L} & b_{1L} & b_{2L} & 1 & a_{1L} & a_{2L} \end{bmatrix}$$

whose rows contain the numerator and denominator coefficients b_{1k} and a_{1k} of the second order sections of H(z).

$$H(z) = g\prod_{k=1}^{L} \frac{b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}}{1 + a_{1k}z^{-1} + a_{2k}z^{-2}}$$

Practice:

Design a Butterworth digital lowpass filter with no more than 1 dB attenuation below 800 Hz and at least 56 dB attenuation at and beyond 3200 Hz. The sampling frequency is 10000 Hz. Find the coefficients of second-order sections.

```
>>[N,Wn]=buttord(wp,ws,Rp,Rs);
>>[B,A]=butter(N,Wn);
>>[z,p,k]=td2zp(B,A);
>>sos=zp2sos(z,p,k)

sos =

   0.0043    0.0087    0.0043    1.0000   -1.0729    0.3083
   1.0000    2.0000    1.0000    1.0000   -1.3455    0.6407
```

Filter Realization Wizard:

MATLAB has a built-in filter realization wizard that automatically generates filter realizations with specific architecture. The Wizard's interface allows you to select from the following realizations:

- Direct form I & II
- Symmetric FIR
- Lattice (MA)
- Lattice (AR)
- Lattice (ARMA)

To launch the filter realization wizard, simply type **dspfwiz** in the command window:

```
>>dspfwiz  <enter>
```

Practice:

Direct Form II Realization:

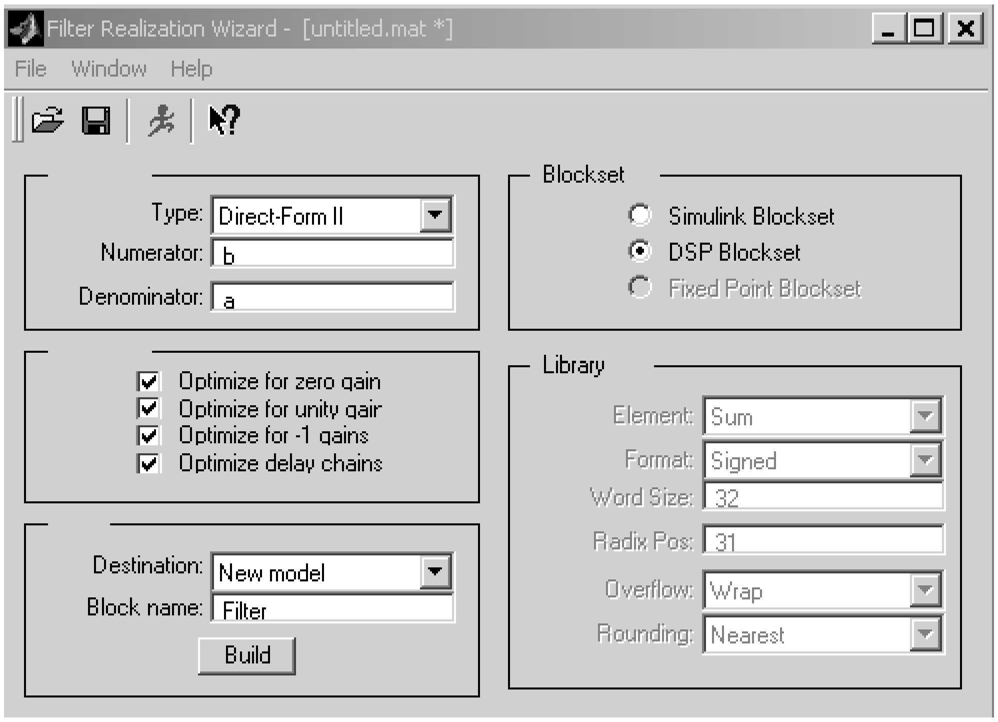
Design a fourth-order, quarter-band, lowpass Butterworth filter

```
>>[b,a]=butter(4,0.25)

b =

    0.0102    0.0408    0.0613    0.0408    0.0102


a =

    1.0000   -1.9684    1.7359   -0.7245    0.1204
```

You configure the wizard to use the coefficients **b** and **a** of the designed filter of a Direct-FORM II structure:

- Select Direct-FORM II from the Type menu
- Type **b** in the numerator text field
- Type **a** in the denominator text field

The graphical user interface with these settings is depicted below:



Filter Realization Wizard - [untitled.mat *]

File Window Help

Type: Direct-Form II
Numerator: b
Denominator: a

Blockset
- ○ Simulink Blockset
- ● DSP Blockset
- ○ Fixed Point Blockset

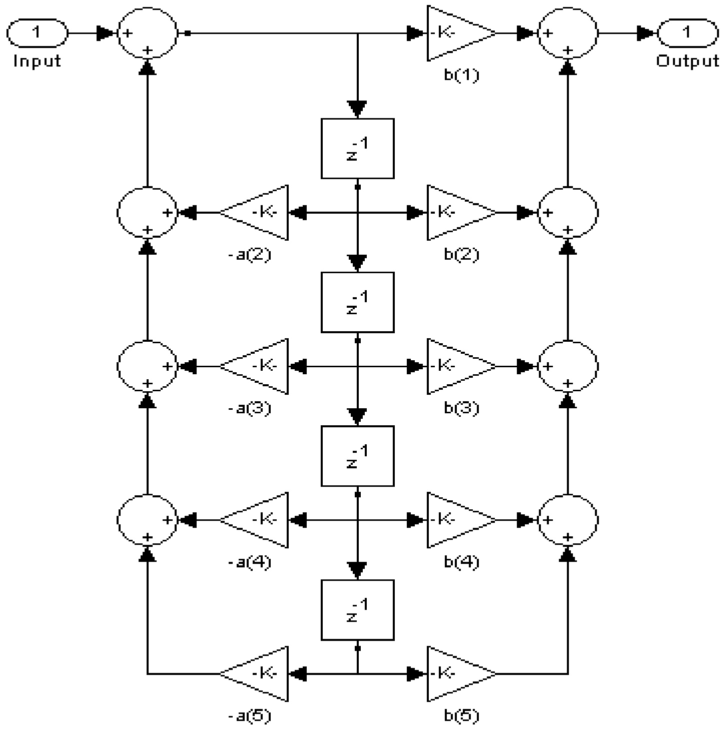
- ☑ Optimize for zero gain
- ☑ Optimize for unity gain
- ☑ Optimize for -1 gains
- ☑ Optimize delay chains

Library
Element: Sum
Format: Signed
Word Size: 32
Radix Pos: 31
Overflow: Wrap
Rounding: Nearest

Destination: New model
Block name: Filter

Build

Press the **Build** button to create the specified filter subsystem in a new model window. Finally, you double-click the created block to see the Direct-Form II filter realization that the Wizard created. The Direct-Form II filter realization is depicted below:



Finite Impulse Response (FIR) Filters:

We wrap up the design of digital filters with FIR filters. In this section we consider the techniques based on the **window method** and the **Parks and McClellan algorithm** (minimax).

Window functions:

Windows are used to manipulate data (original signal) in such a way, that the desired information can be extracted from the spectrum. There are many types of windows proposed for use in spectral analysis and filter design. MATLAB has several built-in window functions, namely, **Bartlett**, **Blackman**, **Boxcar** (uniform), **Chebwin**, **Hann**, **Kaiser**, **Hamming**, and **Triang**.

Window functions	**MATLAB keywords**
Uniform	boxcar
Hanning	hann
Hamming	hamming
Kaiser	kaiser
Chebyshev	chebwin
Blackman	blackman
Bartlett	bartlett
Triangular	triang

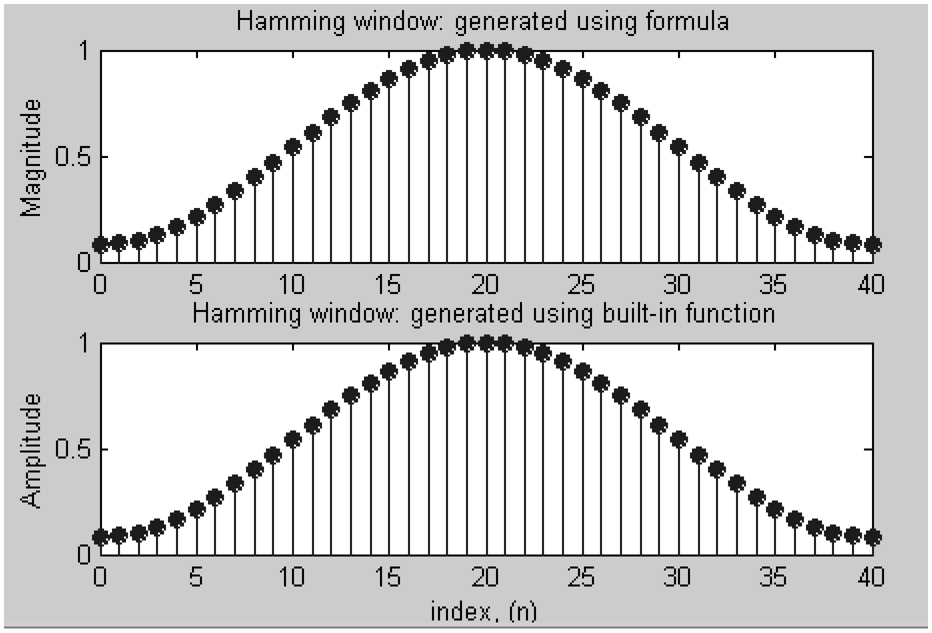
Practice:

The Hamming window is defined by

$$w_{ham}[n] = \begin{cases} 0.54 - 0.46\cos\left(\dfrac{2\pi n}{N-1}\right), & 0 \le n \le N-1 \\ 0, \quad \text{elsewhere} & \end{cases}$$

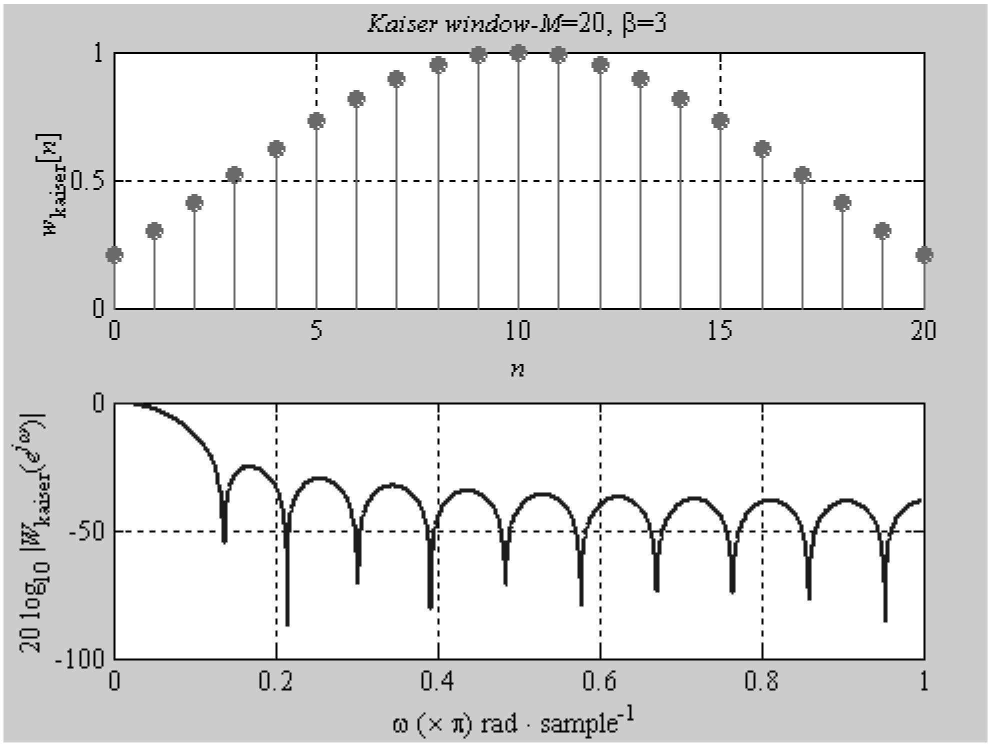
1. Write a piece of code to plot it
2. Repeat part (1) using the MATLAB built-in function

```
N=40;
for  n=0:N
        ham(n+1)=0.54-0.46*cos((2*pi*n)/N);
        index(n+1)=n;
end
subplot(2,1,1); stem(index,ham,’filled’)
subplot(2,1,2);stem(index,hamming(41),’filled’)
```



Kaiser window and its spectrum:

```
N=1024;
M=20;
beta=3;
b=kaiser(M+1,beta);
subplot(2,1,1);stem(0:M,b,'m','filled');grid
h=get(gcf,'CurrentAxes');
set(h,'FontName','times')
ylabel('\fontname{times}\itw_{\rmkaiser}\rm[\itn\rm]');
title('\fontname{times}\itKaiser window-M\rm=20, \beta=3');
xlabel('\fontname{times} \itn')
[H,w]=freqz(b,1,N);
subplot(2,1,2);plot(w/pi,20*log10(abs(H)/abs(H(1))),'b','LineWidth',2);grid
h=get(gcf,'CurrentAxes');
set(h,'FontName','times')
ylabel('\fontname{times}20 log_{10} |\itW_{\rmkaiser}\rm(\ite^{j\omega}\rm)|')
xlabel('\fontname{times}\omega (\times \pi) rad \cdot sample^{-1}')
```



Window-based FIR filters:

The function **fir1** implements the classical method of windowed linear-phase FIR digital filter design. It designs filters in standard lowpass, bandpass, highpass, and bandstop configurations. This function uses the hamming window by default.

Syntax:

```
>>h=fir1(N,wn);                    %lowpass or bandpass filter
>>h=fir1(N,wn,window);             %lowpass or bandpass with a custom window
```

Generates an **N-th** order lowpass FIR filter with normalized cutoff frequency **wn**, which is defined as wn=fc/(folding frequency)=fc/(fs/2), using the specified **window** and returns the filter coefficients **h** in length **N+1** vector. The output coefficients, h, are ordered in ascending powers of z^{-1}.

$$H(z)=h[1]+h[2]z^{-1}+.....+h[N+1]z^{-N}$$

If no window is specified, the default is the hamming window. The cutoff frequency must be between 0 and 1, with 1 corresponding to the Nyquist frequency. If **wn** is a two-element vector, wn=[w1,w2], **fir1** returns a bandpass filter with passband w1<w<w2.

```
>>h=fir1(N,wn,'high');             %highpass FIR filter
>>h=fir1(N,wn,'high',window);      %highpass with a custom window
>>h=fir1(N,wn,'stop');             %bandstop FIR filter, wn=[w1,w2]
```

```
>>h=fir1(N,wn,'stop',window);           %bandstop with a custom window
```

*<u>Note</u>: The function **fir1** always uses an even filter order for the highpass and bandstop configurations. This is because for odd orders, the frequency response at the Nyquist frequency is zero, which is inappropriate for highpass and bandstop filters.*

<u>Practice:</u>

Design a 50-th order lowpass FIR filter with normalized frequency of 0.4.

```
>>N=50;                                 %specify order of filter
>>wn=0.4;                               %specify normalized frequency
>>coef=fir1(N,wn);                      %provide filter coefficients
>>disp('coef: '), disp(coef)            %print the coefficients
>>[H,w]=freqz(coef,1,512);              %determine the frequency response
>>plot(w/pi, abs(H),'.'); grid          %plot magnitude response
>>title('FIR filter, order=50,wn=0.4*pi');
>>xlabel('Frequency, [rad/sample']')
>>ylabel('Magnitude response')
```

<u>Practice:</u>

Design a 37-th order lowpass FIR filter with a normalized frequency of 0.6. Use the uniform window.

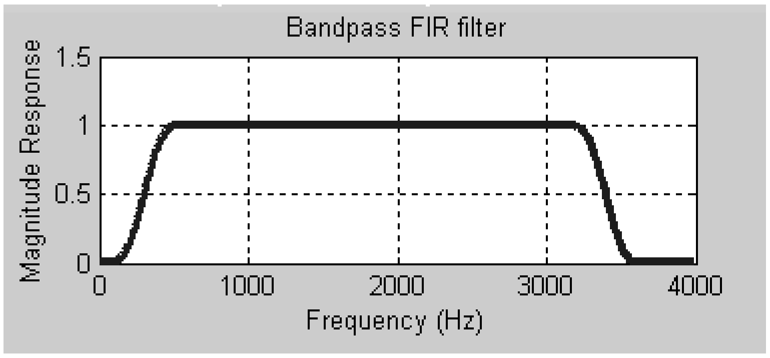
```
>>N=37;                                 %specify order of filter
>>wn=0.6;                               %specify the cutoff frequency in units of pi
>>hu=fir1(N,wn,boxcar(N+1));            %windowed FIR coefficients (uniform window)
>>hb=fir1(N,wn,Blackman(N+1));          %windowed FIR coefficients (Blackman window)
>>hk=fir1(N,wn,kaiser(N+1,5.653));      %windowed FIR coefficients (Kaiser window)
>>[H,w]=freqz(hu,1);                    %compute frequency response of filter
>>mag_dB=20*log10(abs(H));              %compute magnitude in dB
>>plot(w/pi,mag_dB);grid;               %plot the magnitude response
>>title('37-th order FIR filter with uniform window')
```

<u>Practice:</u>

Design a FIR bandpass filter of order 63 with bandwith extending from 300 Hz to 3400 Hz. Use a sampling rate of 8000 Hz.

```
>>order=63;                             %order
>>Fs=8000;                              %sampling rate
>>Fn=Fs/2;                              %folding frequency
>>wl=300/Fn;                            %normalized lower frequency
>>wu=3400/Fn;                           %normalized upper frequency
>>wc=[wl,wu];                           %cutoff frequency vector
>>b=fir1(order,wc];                     %coefficients of FIR filter
>>[H,f]=freqz(b,1,512,Fs);              %frequency response
>>plot(f,abs(H),'LineWidth',3);grid     %plot magnitude response
>>title('FIR bandpass filter')          %add title
>>xlabel('Frequency (Hz)')              %label horizontal axis
```

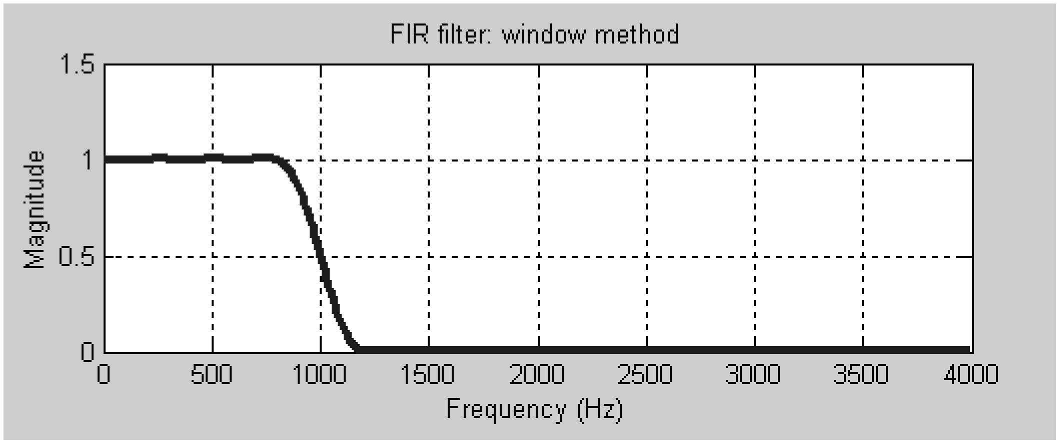
```
>>ylabel('Magnitude response')          %label the vertical axis
```



Practice:

Design a 64-th order lowpass FIR filter for a sampling frequency of 8kHz and cutoff frequency of 1kHz. Use a Hanning window.

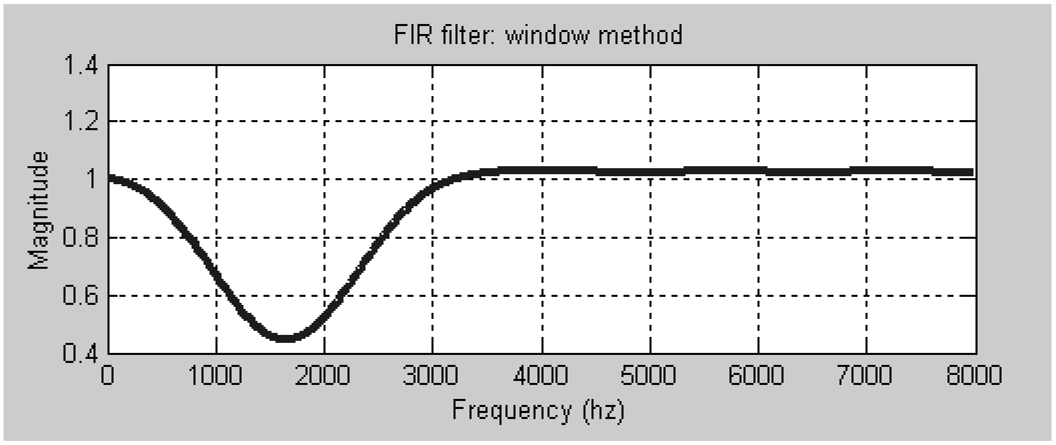
```
>>N=64;                                 %filter order
>>Fs=8e3;                               %sampling frequency
>>Fc=1e3;                               %cutoff frequency
>>Fn=Fs/2;                              %Nyquist frequency
>>fc=Fc/Fn;                             %normalized cutoff frequency
>>b=fir1(N,fc,hanning(N+1));            %filter coefficients
>>[H,w]=freqz(b,1,512];                 %frequency response
>>plot(w*Fn/pi,abs(H));grid             %plot magnitude response
>>title('FIR filter: window method')    %add title
>>xlabel('Frequency (Hz)')              %label horizontal axis
>>ylabel('Magnitude')                   %label vertical axis
```



Practice:

Design a 20-th order band-stop filter for a sampling frequency of 16 kHz, lower cutoff frequency of 1.2 kHz and upper cutoff frequency of 2.1 kHz.

```
>>N=20;                                %filter order
>>Fs=16e3;                             %sampling frequency
>>Fn=Fs/2;                             %Nyquist frequency
>>fc=[1200, 2100]/Fn;                  %normalized cutoff frequencies
>>b=fir1(N,fc,'stop');                 %filter coefficients
>>[H,w]=freqz(b,1,512);                %frequency response
>>plot(w%Fn/pi,abs(H));grid            %plot magnitude response
>>xlabel('Frequency (Hz)')             %label horizontal axis
>>ylabel('Magnitude')                  %label the vertical axis
>>title('FIR filter: window method')   %add title
```



Practice:

Design a 100-th lowpass FIR filter, with sampling frequency 800 Hz and cutoff frequency 65 Hz.

```
>>Fs=800;                   %sampling frequency
>>Fn=Fs/2;                  %Nyquist frequency
>>wc=65/Fn;                 %normalized cutoff frequency
>>N=100;                    %order
>>b=fir1(N,wc);             %filter coefficients
```

Windowed method with an arbitrary shape:

The function **fir2** designs a linear phase FIR filter using the window method with arbitrary shaped magnitude responses.

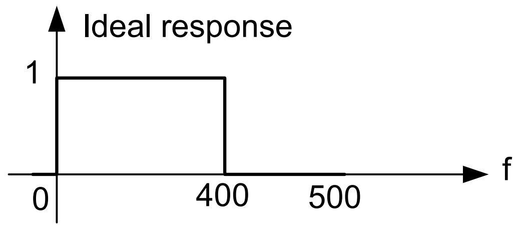
Syntax:

```
>>h=fir2(N,f,m);            %FIR filter with Hamming window
>>h=fir2(N,f,m,window);     %FIR filter with custom window
```

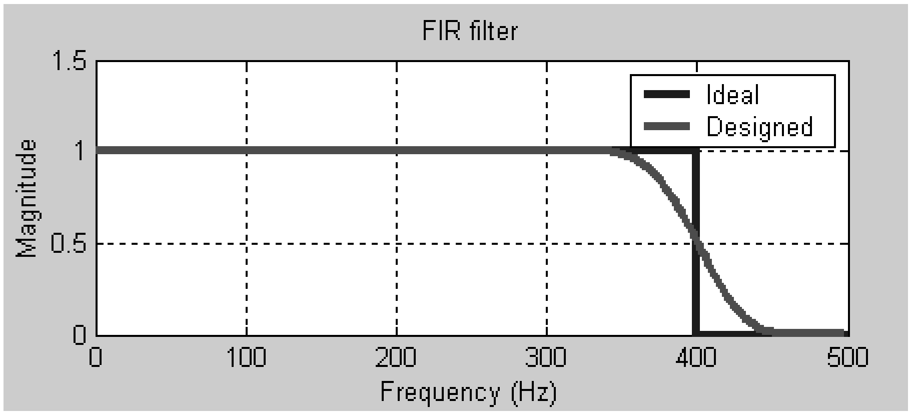
designs an **N-th** order FIR digital filter with the frequency response specified by vectors **f** and **m** and returns the vector **h** of length (N+1) corresponding to the filter coefficients, arranged in ascending powers of z^{-1}. Vectors **f** and **m** specify the frequency points and magnitude values at the specified frequency points. The frequency points are arranged in an increasing order, in the range 0 to 1, with the first frequency being 0 and the last frequency point being 1 (Nyquist frequency). By default, **fir2** uses a Hamming window. The vector window must be (N+1) elements long. Duplicate frequency points are allowed, corresponding to steps in the frequency response.

Practice:

Design a 30-th order lowpass filter and overplot the desired frequency response with the actual frequency response.



```
>>Fs=1000;
>>Fn=Fs/2;
>>f=[ 0 400 400 500]/Fn;
>>m=[1 1 0 0];
>>h=fir2(30,f,m);
>>[H,w]=freqz(h,1,256);
>>plot(f*Fn, m, 'o',Fn*w/pi, abs(H),'LineWidth',3);grid
>>legend('Ideal', 'Designed')
>>xlabel('Frequency (Hz)')
>>ylabel('Magnitude')
>>title('FIR filter')
```



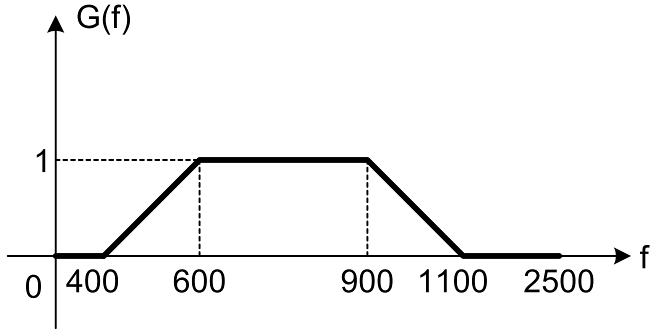
Parks & MacClellan Algorithm:

The most widely accepted method for designing FIR filters is based on an algorithm devised by Parks and McClellan based on the **Remez exchange algorithm**. This method is carried out in MATLAB using the **remez** command. This method is considered optimal, because it produces a design that gives the smallest error between the actual and desired responses. Filters designed this way exhibit an equiripple behavior.

A FIR filter designed via this procedure is specified via the order of the filter and the location of its passbands and stopbands. The bands are specified in terms of the Nyquist frequency. The bands must be linear, but not restricted to having zero slopes. To specify the bands two variables, frequency and magnitude are given. Frequency represents a vector of frequencies specifying the start and endpoints for each band, and must always start with 0 (DC) and end with 1 (Nyquist frequency). Magnitude specifies the corresponding magnitude at the edges of the bands. Here is an illustration of the procedure.

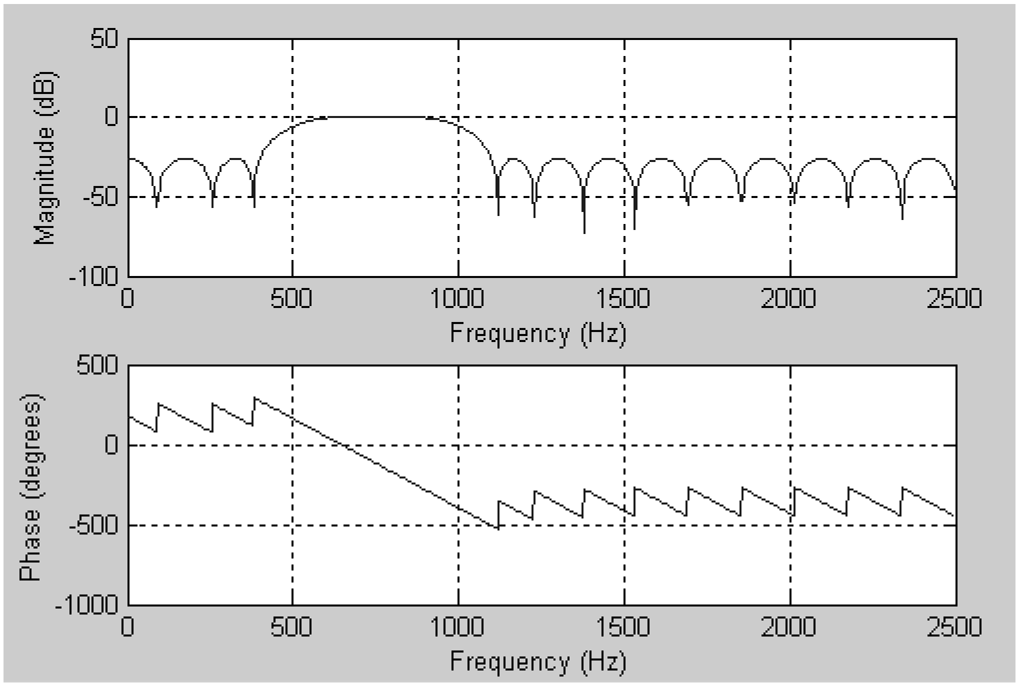
Practice:

Design an optimal 32-coefficient FIR bandpass filter that aims to achieve unity gain between 600 Hz and 900 Hz, and zero gain below 400 Hz and above 1100 Hz. Assume a 5 kHz sampling frequency.



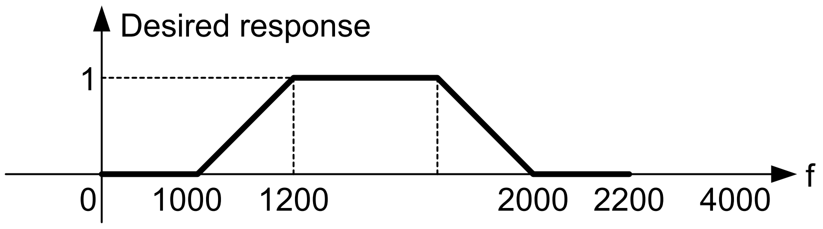
To implement this algorithm, we need to specify two vectors, one containing six frequencies each normalized to Nyquist frequency, the other containing the desired amplitude response at the six specified frequencies. Lastly, we apply the **remez** command to derive the filter coefficients. The order of the filter is taken to be the number of coefficients less one.

```
>>freq=[0 400/2500 600/2500 900/2500 1100/2500 2500/2500]; %frequency vector
>>gain=[0 0 1 1 0 0];                                       %amplitude vector
>>order=31;                                                 %order of filter
>>coef=remez(order,freq,gain);                              %coefficients
>>freqz(coef, 1, 400,5000);                                 %plot responses
```

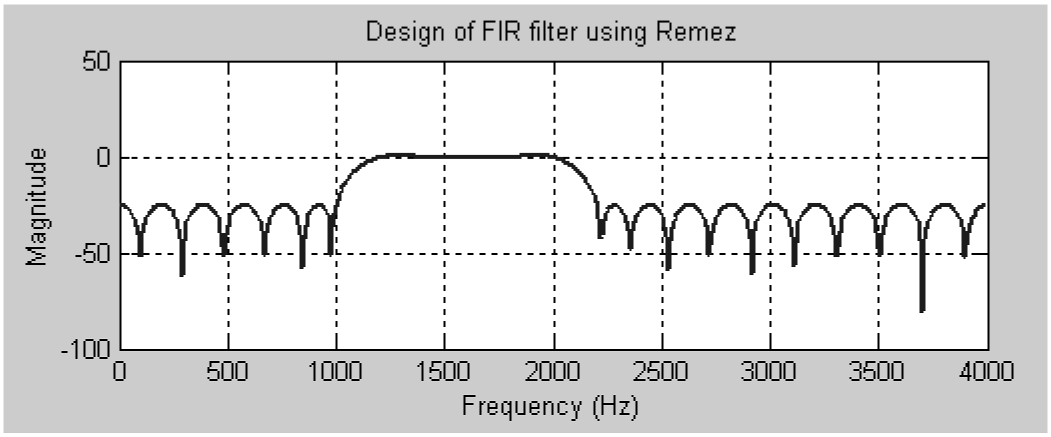


Practice:

Design a bandpass filter using Remez.



```
>>Fs=8000;
>>Fn=Fs/2;
>>F1=1000/Fn;
>>F2=1200/Fn;
>>F3=2000/Fn;
>>F4=2200/Fn;
>>F=[0  F1  F2  F3  F4  1];
>>m=[0  0  1  1  0  0];
>>N=43;
>>b=remez(N-1,F,m);
>>[H,f]=freqz(b,1,512,Fs);
>>plot(f,20*log10(abs(H))); grid
```


Design of FIR filter using Remez
Magnitude
Frequency (Hz)
50
0
-50
-100
0
500
1000
1500
2000
2500
3000
3500
4000