Chapter 5

- Design & Analysis of Analog & Digital Filters -

Analog Filter Design

Introduction

In order to process signals, we have to design and implement systems called filters. A filter is a device that transmits (or rejects) a specific range of frequencies. There are four basic filter types; low-pass, high-pass, band-pass, and band-stop.

Three Commonly Used Filters

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

Determination of the Minimum Order

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.

<u>Practice</u>

- Analog Filter Design-

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

What Are 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.

Commands

- "buttap", "cheblap", "cheb2ap" & "ellipap"

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

- Normalized Lowpass Analog Filters : Butterworth Filter-

Using MATLAB, find the poles, zeros, and the gain factor of a normalized 5^{th}-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

- Normalized Lowpass Analog Filters : Chebyshev Type-I Filter -

Using MATLAB, find poles, zeros, and the gain factor of a normalized 4^{th}-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
```

Transfer Function

- Commands: "zp2tf"

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

- Normalized Lowpass Analog Filters : Transfer Function -

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)           % Transfer function model
>>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

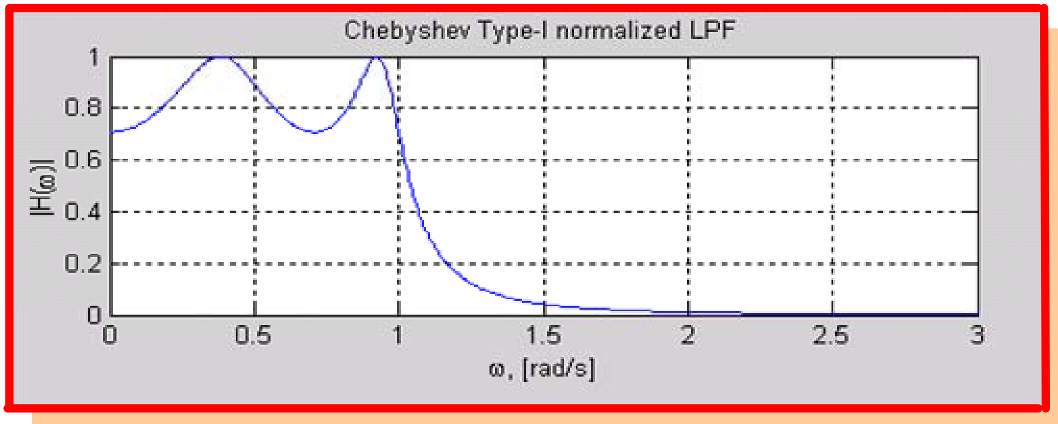
Practice

- Frequency Response of Analog Filters -

Design a 4^{th}-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')
```

(The graph is shown on the next page)



Frequency Transformations

Introduction

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.

Commands

- "lp2lp", lp2hp", lp2bp", & "lp2bs"

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)

<u>Practice</u>

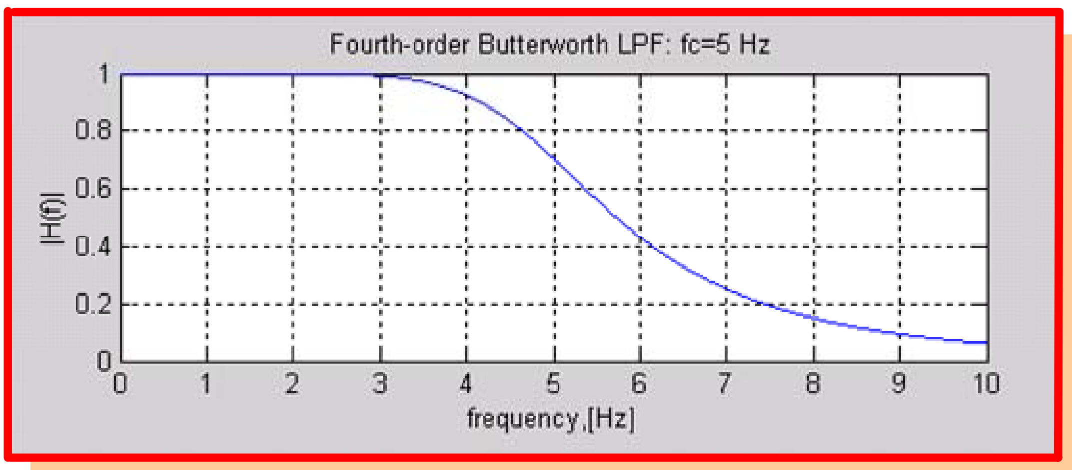
- Frequency Transformation -

(1)

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 H(z)
>>xlabel('frequency, [Hz]')             % label the horizontal axis
>>ylabel('|H(f)|')                      % label the vertical axis
>>title('Fourth-order Butterworth LPF: fc= 5 Hz')
```



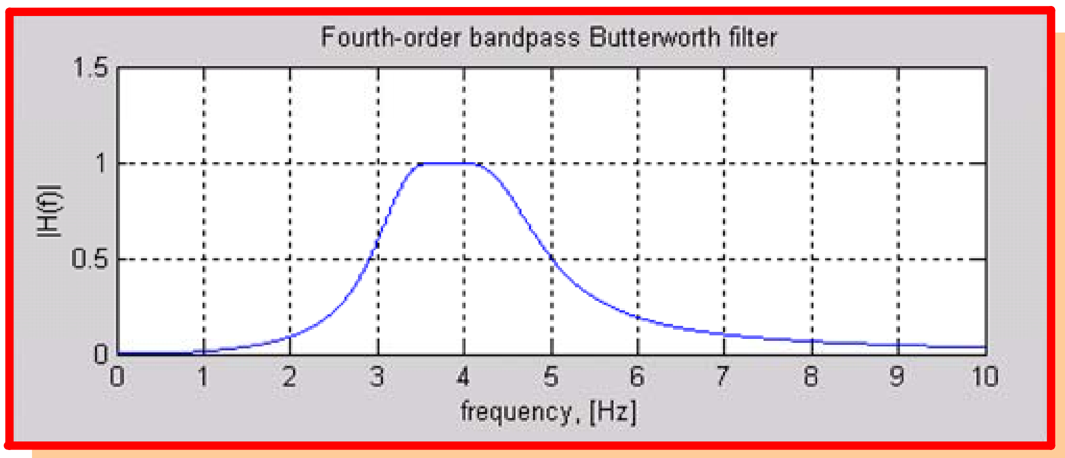
Practice

- Frequency Transformation -

(2)

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')
```



More Commands

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

A. 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

- Frequency Transformation -

(3)

Design an 8th-order Butterworth bandpass filter with Wc=$[2\pi, 5\pi]$. 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
```

B. Chebyshev Analog Filters

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

B.1 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
```

B.2 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

Two Kinds of Digital Filters

-FIR & IIR

Two types of digital filters can be identified, 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 $[0, 2\pi]$ *is specified in MATLAB digital filter function 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

<u>Practice</u>

- Digital Filter Design -

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

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

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

- Digital Filter Design: Infinite Impulse Response Filter -

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

```
>>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,

1. Bilinear transformation method
2. Impulse invariance method
3. The yule walker design method

A. Bilinear Transformation Method

A.1 Basic Commands

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

- Infinite Impulse Response Filters: Bilinear Transformation Method -

(1)

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]')
```

A.2 More Commands

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

- Infinite Impulse Response Filters: Bilinear Transformation Method -

(2)

For data sample at 1000 Hz, design a 9-th order highpass Butterworth filter with cutoff frequency of 300 Hz.

```
[b,a]=butter(9, 300/500,'high');
Freqz(b,a,128,1000)                    % 128 equidistant points from 0 to y
```

Practice

- Infinite Impulse Response Filters: Bilinear Transformation Method -

(3)

Design a 10-th order bandpass filter with a passband form 100 to 200 Hz and plot its impulse response.

```
n=5; wn=[100 200]/500;
[b,a]=butter(n,wn);
[y,t]=impt(b,a,101);
stem(t,g)
```

Practice

- Infinite Impulse Response Filters: Bilinear Transformation Method -

(4)

For data sampled at 1000 Hz, design a 9-th order lowpass Chebyshev type II filter with ? ? of 20 dB down from the passband and a cutoff frequency of 300 Hz.

```
[b,a]=cheby2(9,20,300/500);
Freqz=(b,a,512,1000);
```

Practice

- Infinite Impulse Response Filters: Bilinear Transformation Method -

(5)

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
```

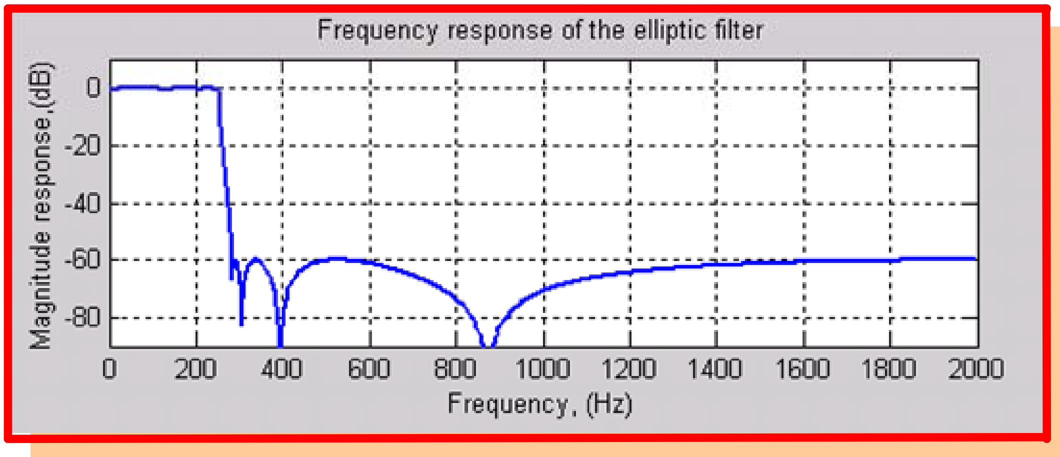
Practice

- Infinite Impulse Response Filters: Bilinear Transformation Method -

(6)

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')
```



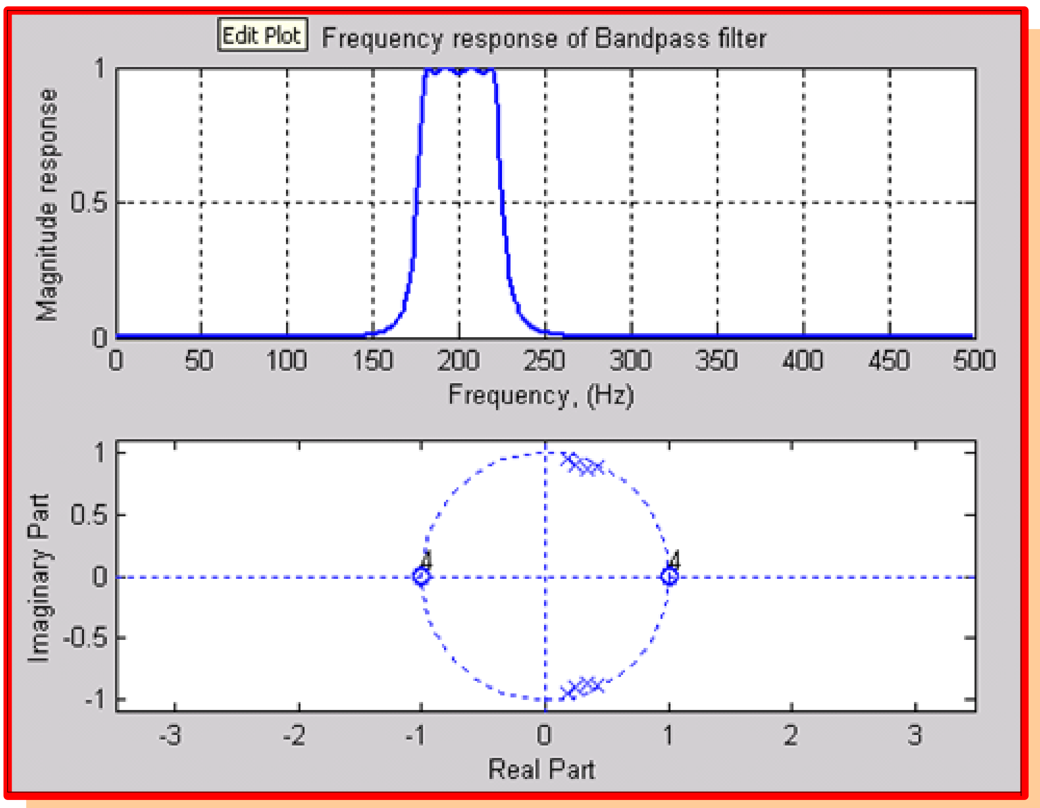
Practice

- Infinite Impulse Response Filters: Bilinear Transformation Method –

(7)

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

- Infinite Impulse Response Filters: Bilinear Transformation Method -

(8)

Design a 4-th order HR Chebyshev type bandpass filter with cutoff frequencies at 300 Hz, 0,5dB 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:pi99:99;
>>H=freqz(b,a,w);
>>phase=unwrap(phase,tol);
>>title('Magnitude plot for IIR filter')
>>xlabel('Frequency (radians)')
>>ylabel('Magnitude (dB)')
```

Practice

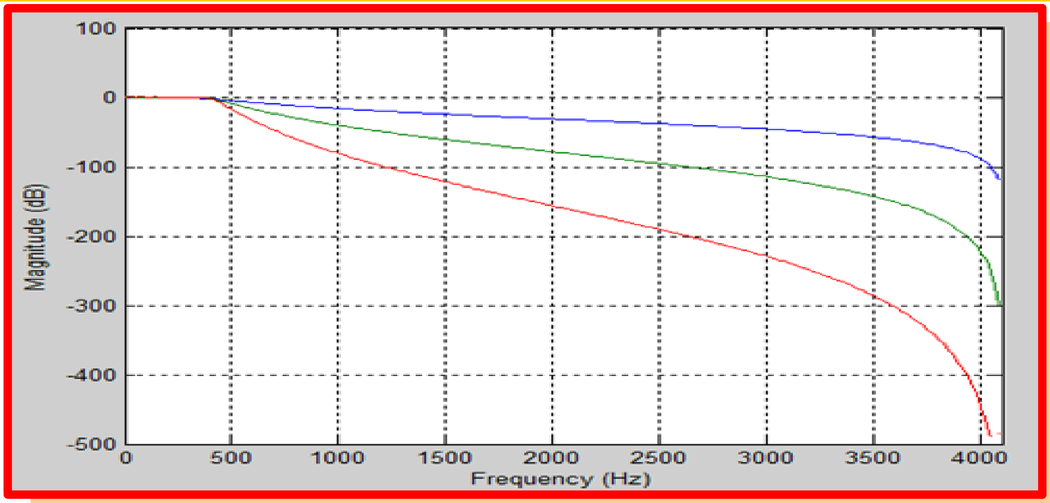
- Infinite Impulse Response Filters: Bilinear Transformation Method -

(9)

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)
```

(The plot is shown on the next page)



Practice

- Infinite Impulse Response Filters: Bilinear Transformation Method -

(10)

Design an IIR Lowpass filter to meet the following specifications:

$\omega_p = 0.25\pi \qquad \omega_s = 0.4\pi$

$R_p = 0.1dB \qquad A_s = 40dB$

Solution: We will use a Butterworth filter for this design

Matlab Code:

```
>> %Given LPF specifications
wp=0.25*pi;
ws=0.4*pi;
As=40;
Rp=0.1;

%Calculate Butterworth filter parameters
%Matlab requires that freq. be specified in
%units of pi radian
[N,wn]=buttord(wp/pi,ws/pi,Rp,As);

%Design digital Butterworth filter
[b,a]=butter(N,wn);
```

(More matlab codes continue on the next page)

```
%Modified freqz command
[db,mag,pha,grd,w]=freqz_m(b,a);

%Plot
subplot(2,2,1); plot(w/pi,db); title('Magnitude Response in dB');
grid; axis([0 1 -100 10]), xlabel('frequency in \pi units'); ylabel('Decibels');

subplot(2,2,2); plot(w/pi, pha/pi); title('Phase Response');
grid; axis([0 1 -1 1]), xlabel('frequency in \pi units'); ylabel('Phase in \pi units');
subplot(2,2,3); plot(w/pi,mag); title('Magnitude Response');
grid; axis([0 1 0 1]), xlabel('frequency in \pi units');
%ylabel(‘Phase in \pi units’);

subplot(2,2,4); zplane(b,a); title('Pole-zero plot');
```

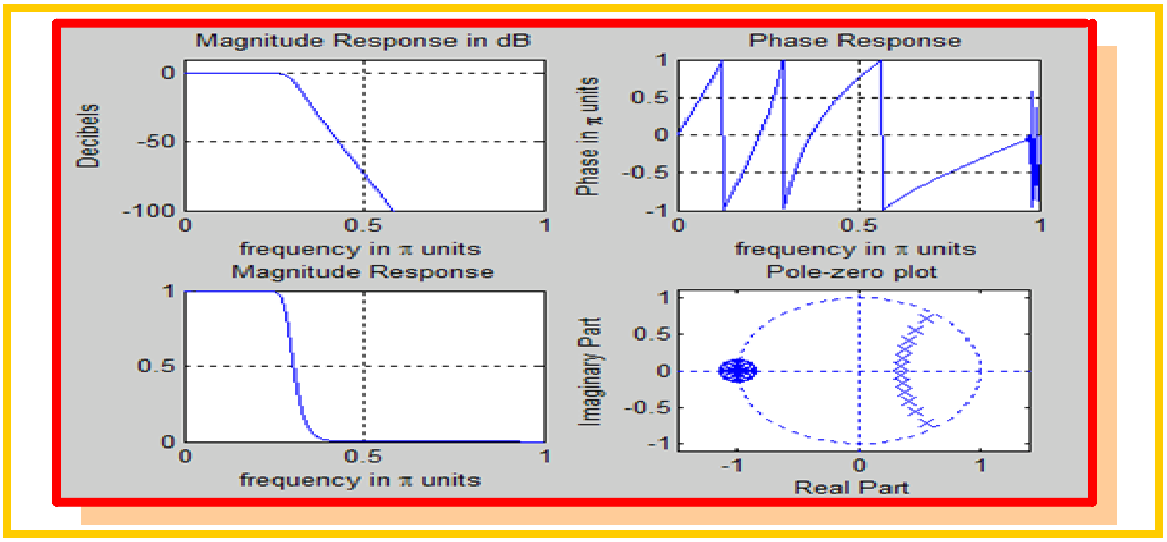
Required Routine

```
function[db, mag, pha, grd, w]=freqz_m(b,a)
%Modified version of freqz sunroutine
%----------------------------------------
%[db,mag,pha,grd,w]=freqz_m(b,a)
%db=Relative magnitude in dB computed over 0 to pi radians
%mag=absolute magnitude computed over 0 to pi radians
%pha=Phase response in radians over 0 to pi radians
%grd=group delay over 0 to pi radians
%w=501 frequency samples between 0 to pi radians
%b=numerator polynimial of H(z) (for FIR:b=h)
%a=denominator polynomial of H(z) (for FIR: a=[1])
%
[H,w]=freqz(b,a,1000,'whole');
H=(H(1:1:501))';
w=(w(1:1:501))';
mag=abs(H);
db=20*log10((mag+eps)/max(mag));
pha=angle(H);
grd=grpdelay(b,a,w);
%End of function
```

Matlab Output:

```
N=
       12
wn=
        0.293π
```

(The plots are shown on the next page)



NOTE

To display the frequency-domain plots of digital filters, use the following routine.

```
function[db, mag, pha, grd, w]=freqz_m(b,a)
%Modified version of freqz sunroutine
%---------------------------------------
%[db,mag,pha,grd,w]=freqz_m(b,a)
%db=Relative magnitude in dB computed over 0 to pi radians
%mag=absolute magnitude computed over 0 to pi radians
%pha=Phase response in radians over 0 to pi radians
%grd=group delay over 0 to pi radians
%w=501 frequency samples between 0 to pi radians
%b=numerator polynimial of H(z) (for FIR:b=h)
%a=denominator polynomial of H(z) (for FIR: a=[1])
%
[H,w]=freqz(b,a,1000,'whole');
H=(H(1:1:501))';
w=(w(1:1:501))';
mag=abs(H);
db=20*log10((mag+eps)/max(mag));
pha=angle(H);
grd=grpdelay(b,a,w);
%End of function
```

It requires the magnitude response in absolute as well as in relative dB scale, the phase response, and the group delay response.

Practice

- Infinite Impulse Response Filters: Bilinear Transformation Method -

(11)

Design an IIR Highpass filter to meet the following specifications:

$$\omega_p = 0.5\pi \qquad \omega_s = 0.4\pi$$
$$R_p = 0.2dB \qquad A_s = 55dB$$

Solution: Let us use a Chebyshev Type I filter for this example.

Matlab Code:

```
% Given HPF spcifications
wp=0.5*pi;
ws=0.4*pi;
As=55;
Rp=0.2;

%Calculate chebyshev I filter parameters
%Matlab requires that freq. be specified in
%units of pi radian
[N,wn]=cheb1ord(wp/pi,ws/pi,Rp,As)

%Design digital Chebyshev I highpass filter
[b,a]=cheby1(N,Rp,wn,'high');

%Modified freqz command
[db,mag,pha,grd,w]=freqz_m(b,a);

%plot
subplot(2,2,1); plot(w/pi,db); title('Magnitude Response in dB');
grid; axis([0 1 -100 10]), xlabel('frequency in \pi units'); ylabel('Decibels');

subplot(2,2,2); plot(w/pi,pha/pi); title('Phase Response');
grid; axis([0 1 -1 1]), xlabel('reequency in \pi units'); ylabel('phase in \pi units');
subplot(2,2,3); plot(w/pi,mag); title('Magnitude Response');
grid; axis([0 1 0 1]), xlabel('frequency in \pi units');
%ylabel('Phase in \pi units');

subplot(2,2,4); Zplane(b,a); title('Pole-zero plot');
```

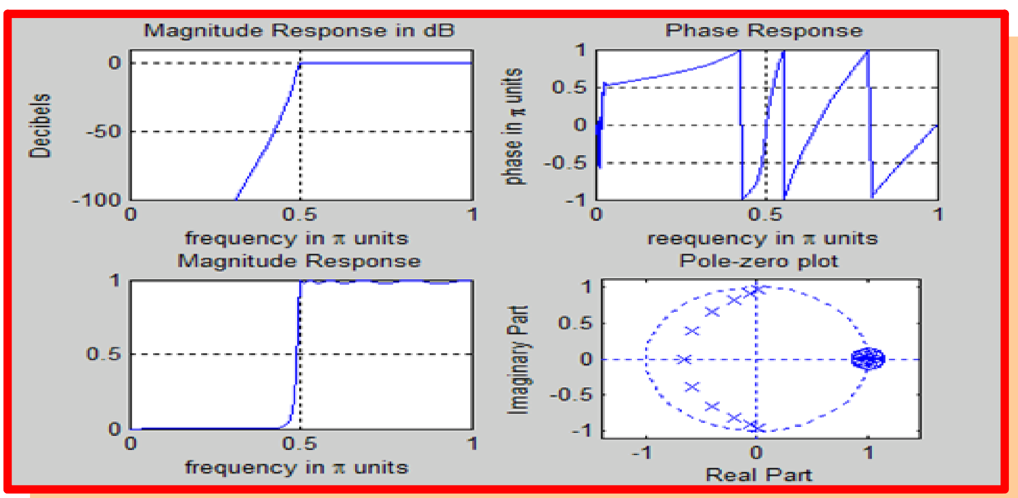
(The required rountine and matlab output are shown on the next page)

Required Routine

```
function[db, mag, pha, grd, w]=freqz_m(b,a)
%Modified version of freqz sunroutine
%----------------------------------------
%[db,mag,pha,grd,w]=freqz_m(b,a)
%db=Relative magnitude in dB computed over 0 to pi radians
%mag=absolute magnitude computed over 0 to pi radians
%pha=Phase response in radians over 0 to pi radians
%grd=group delay over 0 to pi radians
%w=501 frequency samples between 0 to pi radians
%b=numerator polynimial of H(z) (for FIR:b=h)
%a=denominator polynomial of H(z) (for FIR: a=[1])
%
[H,w]=freqz(b,a,1000,'whole');
H=(H(1:1:501))';
w=(w(1:1:501))';
mag=abs(H);
db=20*log10((mag+eps)/max(mag));
pha=angle(H);
grd=grpdelay(b,a,w);
%End of function
```

Matlab Output:

N=
    11
wn=
    0.5π



Practice

- Infinite Impulse Response Filters: Bilinear Transformation Method -

(12)

Design an IIR Bandpass filter to meet the following specifications:

$$\omega_{p,1} = 0.25\pi \qquad \omega_{p,2} = 0.5\pi \qquad \omega_{s,1} = 0.2\pi \qquad \omega_{s,2} = 0.6\pi$$
$$R_p = 0.2dB \qquad A_s = 40dB$$

Solution: Let us use a Chebyshev Type II filter for this example.

Matlab Code:

```
% Given BPF specifications
wp1=0.25*pi; wp2=0.5*pi; wp=[wp1 wp2];
ws1=0.2*pi; ws2=0.6*pi; ws=[ws1 ws2];
As=40;
Rp=0.2;

%Calculate Chebyshev II filter parameters
%Matlab requires that freq. be specified in
%units of pi radian
[N,wn]=cheb2ord(wp/pi,ws/pi,Rp,As)

%Design digital Chebyshev II bandpass filter
[b,a]=cheby2(N,As,wn);

%Modified freqz command
[db,mag,pha,grd,w]=freqz_m(b,a);

%Plot
subplot(2,2,1); plot(w/pi,db); title('Magnitude Response in dB');
grid; axis([0 1 -100 10]), xlabel('frequency in \pi units'); ylabel('Decibels');

Subplot(2,2,2); plot(w/pi,pha/pi); title('Phase Response');
grid; axis([0 1 -1 1]), xlabel('frequency in \pi units'); ylabel('Phase in \pi units');

subplot(2,2,3); plot(w/pi,mag); title('Magnitude Response');
grid; axis([0 1 0 1]), xlabel('frequency in \pi units');
%ylabel('Phase in \pi units')

subplot(2,2,4); zplane(b,a); title('Pole-zero plot');
```

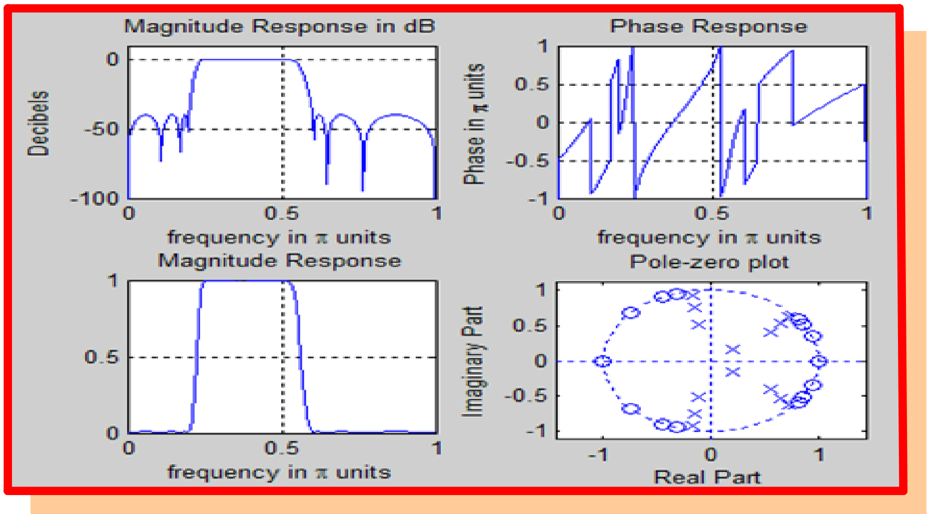
(The required routine and the matlab output are shown on the next page)

Required Routine

```
function[db, mag, pha, grd, w]=freqz_m(b,a)
%Modified version of freqz sunroutine
%----------------------------------------
%[db,mag,pha,grd,w]=freqz_m(b,a)
%db=Relative magnitude in dB computed over 0 to pi radians
%mag=absolute magnitude computed over 0 to pi radians
%pha=Phase response in radians over 0 to pi radians
%grd=group delay over 0 to pi radians
%w=501 frequency samples between 0 to pi radians
%b=numerator polynimial of H(z) (for FIR:b=h)
%a=denominator polynomial of H(z) (for FIR: a=[1])
%
[H,w]=freqz(b,a,1000,'whole');
H=(H(1:1:501))';
w=(w(1:1:501))';
mag=abs(H);
db=20*log10((mag+eps)/max(mag));
pha=angle(H);
grd=grpdelay(b,a,w);
%End of function
```

Matlab Output:

```
N=
    7    (Note: Filter order is 2N)
wn=
     0.208π   0.564π
```



Practice

- Infinite Impulse Response Filters: Bilinear Transformation Method -

(13)

Design an IIR Bandstop filter to meet the following specifications:

$$\omega_{p,1} = 0.2\pi \qquad \omega_{p,2} = 0.7\pi \qquad \omega_{s,1} = 0.3\pi \qquad \omega_{s,2} = 0.65\pi$$
$$R_p = 0.2dB \qquad A_s = 50dB$$

Solution: We will use the Elliptic filter for this example.

Matlab Code:

```
% Given BSF specifications
wp1=0.2*pi; wp2=0.7*pi; wp=[wp1 wp2];
ws1=0.3*pi; ws2=0.65*pi; ws=[ws1 ws2];
As=50;
Rp=0.2;

%Calculate Elliptic filter parameters
%Matlab requires that freq. be specified in
%units of pi radian
[N,wn]=ellipord(wp/pi,ws/pi,Rp,As)

%Design digital Elliptic bandstop filter
[b,a]=ellip(N,Rp,As,wn,'stop');

%Modified freqz command
[db,mag,pha,grd,w]=freqz_m(b,a);

%Plot
subplot(2,2,1);plot(w/pi,db); title('Magnitude Response in dB');
grid; axis([0 1 -100 10]), xlabel('frequency in \pi units'); ylabel('Decibels');

Subplot(2,2,2); plot(w/pi,pha/pi); title('Phase Response');
grid; axis([0 1 -1 1]), xlabel('frequency in \pi units'); ylabel('Phase in \pi units');
subplot(2,2,3); plot(w/pi,mag); title('Magnitude Response');
grid; axis([0 1 0 1]), xlabel('frequency in \pi units');
%ylabel('Phase in \pi units');
subplot(2,2,4); zplane(b,a); title('Pole-zero plot');
```

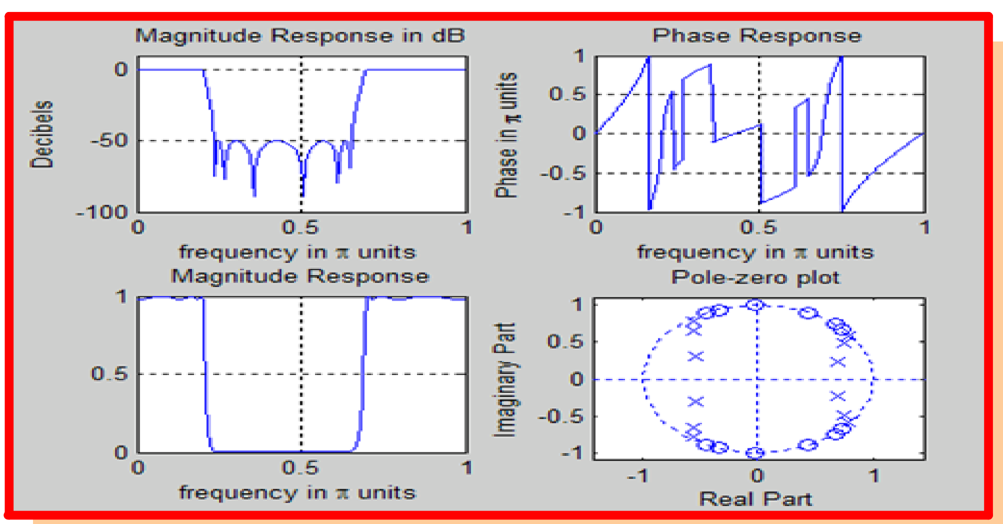
(The required routine and matlab output are shown on the next page)

Required Routine

```
function[db, mag, pha, grd, w]=freqz_m(b,a)
%Modified version of freqz sunroutine
%----------------------------------------
%[db,mag,pha,grd,w]=freqz_m(b,a)
%db=Relative magnitude in dB computed over 0 to pi radians
%mag=absolute magnitude computed over 0 to pi radians
%pha=Phase response in radians over 0 to pi radians
%grd=group delay over 0 to pi radians
%w=501 frequency samples between 0 to pi radians
%b=numerator polynimial of H(z) (for FIR:b=h)
%a=denominator polynomial of H(z) (for FIR: a=[1])
%
[H,w]=freqz(b,a,1000,'whole');
H=(H(1:1:501))';
w=(w(1:1:501))';
mag=abs(H);
db=20*log10((mag+eps)/max(mag));
pha=angle(H);
grd=grpdelay(b,a,w);
%End of function
```

Matlab Output:

N=
  6 (Note: Filter order is 2N)
wn=
  0.255π 0.7π



B. Impulse Invariance Method

Syntax

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

Practice

- Infinite Impulse Response Filters: Impulse Invariance Method -

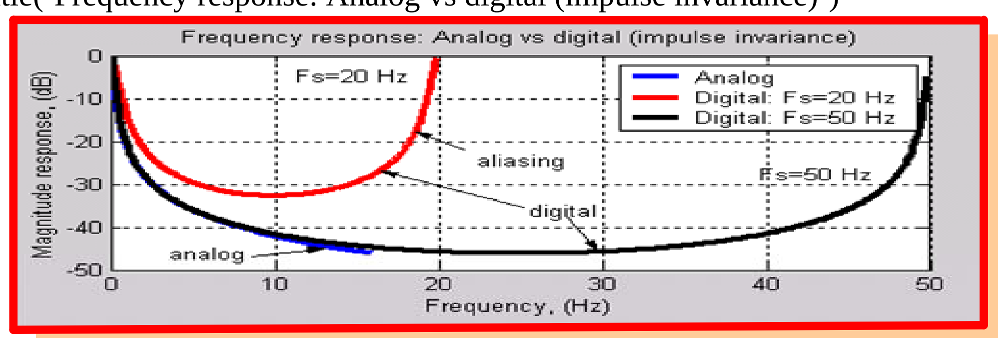
(1)

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

- Infinite Impulse Response Filters: Impulse Invariance Method -

(2)

A sequence has 1 kHz sampling frequency. Design a 6-th 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
```

C. 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.

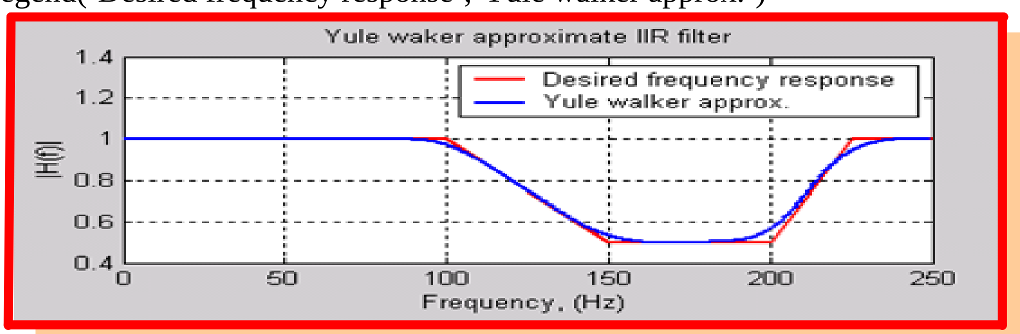
Practice

- Infinite Impulse Response Filters: The Yule Walker Design Method -

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.')
```



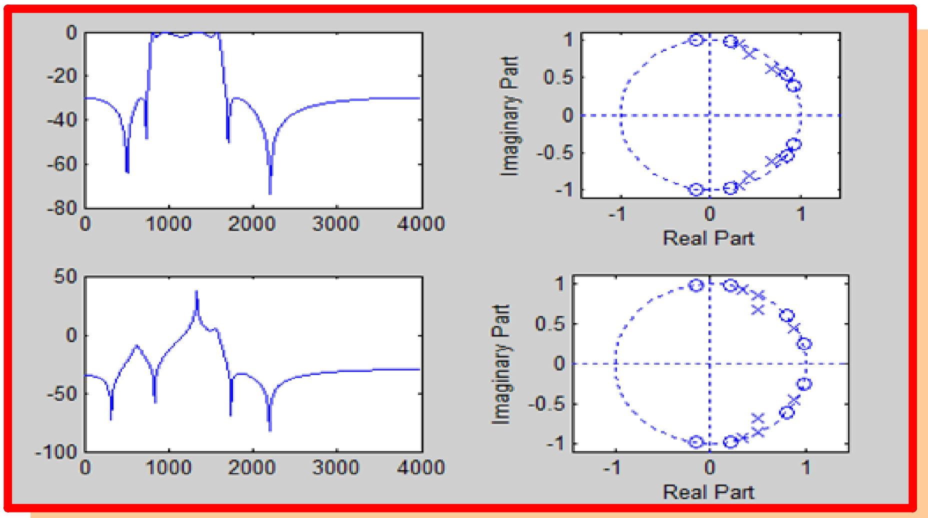
D. Effect of Coefficient Quantization in IIR Filters

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

- Infinite Impulse Response Filters: Effect of Coefficient Quantization -

```
>>[b,a]=ellip(4,2,30,[0.2 0.4]);
>>n=8;
>>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(Hq)));
>>subplot(2,2,2);zplane(b,a)
>>subplot(2,2,4);zplane(bk,ak)
```



E. Cacade Implementation of IIR Filters

IIR filters are typically implemented by cascading low order sections such as second-order section (biguads). 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 section representation. Similarly, the function **tf2sos** converts a digital filter transfer function parameters to second-order section form.

□ **Syntax**

1.

```
>>[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 the gain g equivalent to the zero-pole-gain system represented by input arguments z,p, and k.

2.

```
>>[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**.

3.

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

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.

4.

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

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

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

<u>Practice</u>

- Cacade Implementation of IIR Filters -

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    1.3083
      1.0000    2.0000    1.0000    1.0000    -1.3455    0.6407
```

Finite Impulse Response (FIR) Filters

A FIR filter is a recursive filter that has a transfer function of the form:

$$H(z) = b_0 + b_1 z^{-1} + \ldots + b_{N-1} z^{1-N} = \sum_{n=0}^{N-1} b_n z^{-n}$$

Hence the impulse response $h[n] = \begin{cases} b_n, 0 \le n \le N-1 \\ 0, elsewhere \end{cases}$

As can be seen, an FIR filter is completely specified by a finite set of coefficients.

FIR filters are popular for digital filter design because of the following reasons:

- They are inherently stable.
- They can achieve peruse phase linearity in the passband
- They are easily implementable

In this section we consider the techniques based on the **window method** and the **Parks and McClellan algorithm** (minimax).

A. Window Method

Windows are used to manipulate data (original signal) in such a way, that the desired information can be extracted from the spectrum. They are used to truncate the infinite impulse response of an ideal filter with the result being an FIF filter. There are many types of window 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
Rectangular	rectwin
Hanning	hanning
Hamming	hamming
Kaiser	kaiser
Chebyshev	chebwin
Blackman	blackman
Bartlett	bartlett
Triangular	triang

A.1 Window Method: Hamming & Hanning

Practice

- Finite Impulse Response (FIR) Filters: Window Method: Hamming -

(1)

```
>>b=hamming(N)
```

This command is used to truncate the infinite impulse response of an ideal digital filter with the result being an FIR filter with length N.

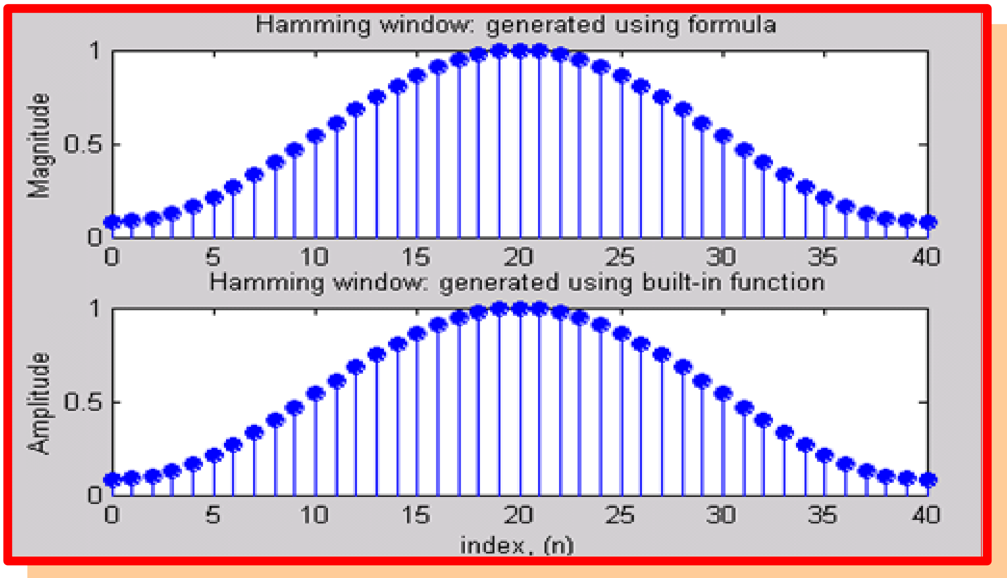
The Hamming window is defined by

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

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

(The matlab codes and the plots are shown on the next page)

```
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')
```



Practice

- Finite Impulse Response (FIR) Filters: Window Method: Hanning -

(2)

Design an FIR lowpass filter to meet the following specifications:

$\omega_p = 0.25\pi \qquad \omega_s = 0.4\pi$

$R_p = 0.1dB \qquad A_s = 40dB$

Solution: For a 40dB stop band attenuation, we can use a Hanning window (which gives a minimum attn. of 44 dB).

$$\Delta\omega = \left|\omega_s - \omega_p\right| = 0.15\pi \geq \frac{6.2\pi}{M} \Rightarrow M \geq \frac{6.2}{0.15} = 41.33, say \text{ M=42}$$

$$\omega_c = \frac{\omega_s + \omega_P}{2} = \frac{0.4\pi + 0.25\pi}{2} = 0.325\pi$$

(Matlab codes, required routine & plots on the following pages)

Matlab Code:

```
wp=0.25*pi; ws=0.4*pi;
tr_width=abs(ws-wp);

%Use Hanning window, since As=40 dB
M=ceil(6.2*pi/tr_width)
n=[0:M-1];
wc=0.5*(ws+wp);     %Ideal LPF cutoff freq.

%Compute Ideal filter impulse response
hd=ideal_lp(wc,M);
w_han=hanning(M)';
h=hd.*w_han;

%Modified freqz command
[db,mag,pha,grd,w]=freqz_m(h,[1]);

%frequency spacing (freqz_m computes 1000 samples)
% of DTFT on the unit whole circle)
delta_w=2*pi/1000;

%Compuate actual passband ripple
Rp=-(min(db(1:1:wp/delta_w+1)))

%Compute actual stopband attn.
As=-round(max(db(ws/delta_w+1:1:1:501)))

%Plots
subplot(1,1,1)
subplot(2,2,1); stem(n,hd); title('Ideal Impulse Respose');
axis([0 M-1 -0.1 0.3]), xlabel('n');ylabel('hd(n)');

subplot(2,2,2); stem(n,w_han); title('Hanning Window');
axis([0 M-1 0 1.1]), xlabel('n'); ylabel('w(n)');

subplot(2,2,3); stem(n,h); title('Actual Impulse Response');
axis([0 M-1 -0.1 0.3]), xlabel('n'); ylabel('h(n)');

subplot(2,2,4); plot(w/pi,db); title('Magnitude Response in dB');
grid; axis([0 1 -100 10]), xlabel('frequency in \pi units'); ylabel('Decibels');
```

(The required routines and matlab output are shown on the next page)

Required Routine I

```
function[db, mag, pha, grd, w]=freqz_m(b,a)
%Modified version of freqz sunroutine
%----------------------------------------
%[db,mag,pha,grd,w]=freqz_m(b,a)
%db=Relative magnitude in dB computed over 0 to pi radians
%mag=absolute magnitude computed over 0 to pi radians
%pha=Phase response in radians over 0 to pi radians
%grd=group delay over 0 to pi radians
%w=501 frequency samples between 0 to pi radians
%b=numerator polynimial of H(z) (for FIR:b=h)
%a=denominator polynomial of H(z) (for FIR: a=[1])
%
[H,w]=freqz(b,a,1000,'whole');
H=(H(1:1:501))';
w=(w(1:1:501))';
mag=abs(H);
db=20*log10((mag+eps)/max(mag));
pha=angle(H);
grd=grpdelay(b,a,w);
%End of function
```

Required Routine II

To design FIR filters based on the window technique, an ideal impass impulse response hd[n] is required. Therefore it is convenient to have a simple routine that creates hd[n] as follows:

```
function hd=ideal_lp(wc,M)
%Ideal LowPass filter computation
%----------------------------------------
%[hd]=ideal_lp(wc,M)
%  hd=ideal impulse response etween 0 to M-1
%  wc=cutoff frequency in radians
%   M=length of the ideal filter
%
alpha=(M-1)/2;
n=[0:1:(M-1)]
M=n-alpha+eps;    %add smallest number to avoid divide by zero
hd=sin(wc*M) ./ (pi*M);
```

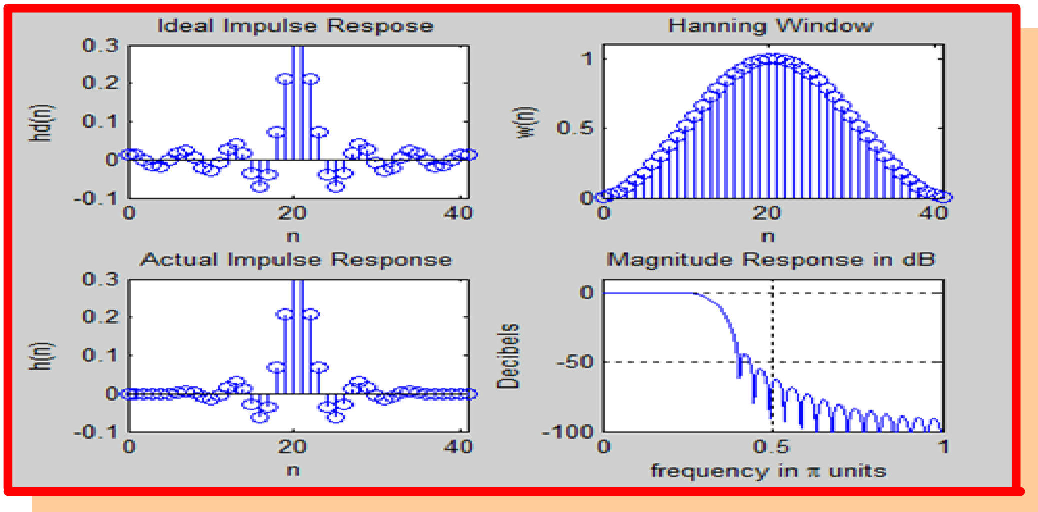
(Matlab output and the plots are shown on the next page)

Matlab Output

M=42

Rp=0.0825

As=44



Practice

- Finite Impulse Response (FIR) Filters: Window method: Hamming -

(3)

Design a M-point FIR differentiator using a Hamming window for M=16 and M=17.

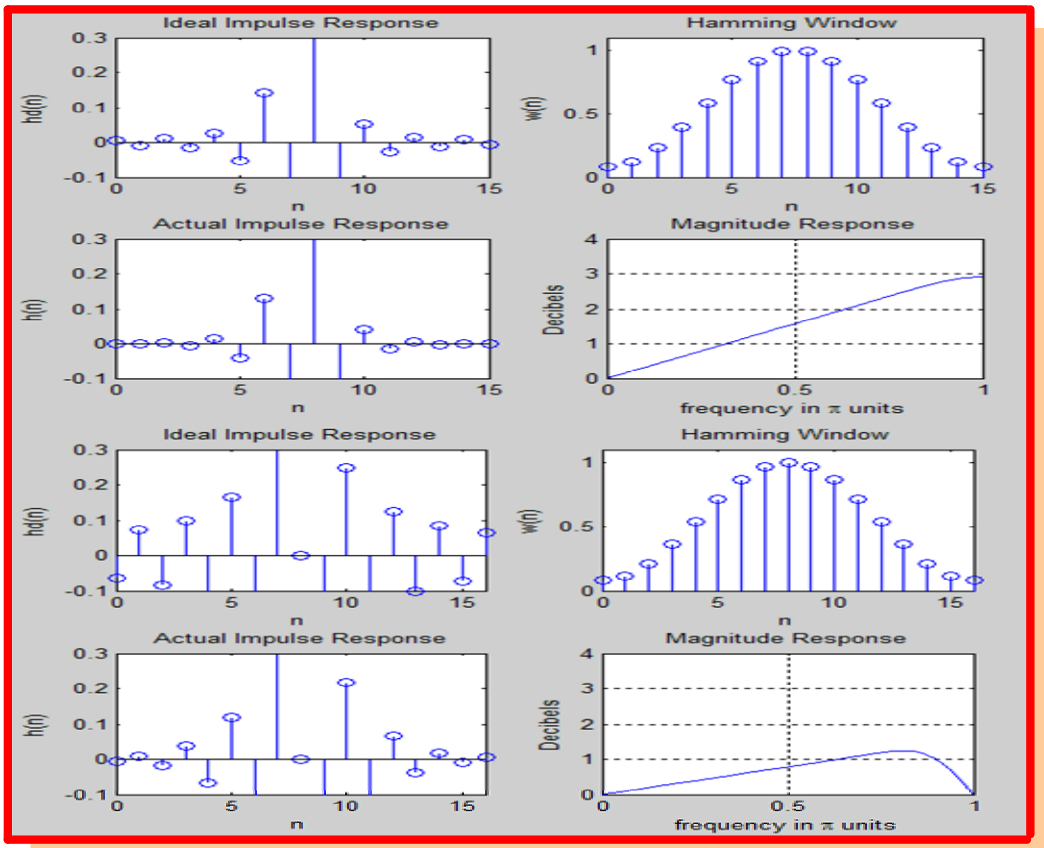
Matlab Code :

```
M=16;
a=(M-1)/2;
n=[0:1:M-1];

if(a==round(a))  %M odd
   hd=((-1).^(n-a))./(2*(n-a));
   hd(a+1)=0;
else
   hd=((-1).^((n-0.5-a)))./(pi*(n-a).^2);
end

w_ham=hamming(M)';
h=hd.* w_ham;

%Modified freqz command
[db,mag,pha,grd,w]=freqz_m(h,[1]);

%Plot
subplot(2,2,1); stem(n,hd); title('Ideal Impulse Response');
axis[(0 M-1 -0.1 0.3]), xlabel('n'); ylabel('hd(n)');

subplot(2,2,2); stem(n,w_ham); title('Hamming Window');
axis([0 M-1 0 1.1]), xlabel('n'); ylabel('w(n)');

subplot(2,2,3); stem(n,h); title('Actual Impulse Response');
axis([0 M-1 -0.1 0.3]), xlabel('n'); ylabel('h(n)');

subplot(2,2,4); plot(w/pi,mag); title('Magnitude Response');
grid; axis([0 1 0 4]), xlabel('frequency in \pi units'); ylabel('Decibels');
```

(The required routine and the plots are shown on the next page)

Required Routine

```
function[db, mag, pha, grd, w]=freqz_m(b,a)
%Modified version of freqz sunroutine
%----------------------------------------
%[db,mag,pha,grd,w]=freqz_m(b,a)
%db=Relative magnitude in dB computed over 0 to pi radians
%mag=absolute magnitude computed over 0 to pi radians
%pha=Phase response in radians over 0 to pi radians
%grd=group delay over 0 to pi radians
%w=501 frequency samples between 0 to pi radians
%b=numerator polynimial of H(z) (for FIR:b=h)
%a=denominator polynomial of H(z) (for FIR: a=[1])
%
[H,w]=freqz(b,a,1000,'whole');
H=(H(1:1:501))';
w=(w(1:1:501))';
mag=abs(H);
db=20*log10((mag+eps)/max(mag));
pha=angle(H);
grd=grpdelay(b,a,w);
%End of function
```



A.2 Window Method: Kaiser Window & Its Spectrum

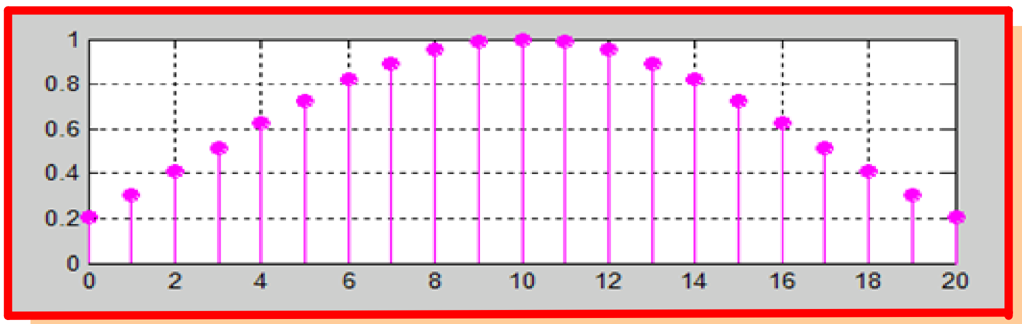
Practice

- Finite Impulse Response (FIR) Filters:

Window Method: Kaiser Window & Its Spectrum -

(1)

```
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,'Current Axes');
set(h,'FontName','time')
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*log(abs(H)/abs(H(1))),'b','LineWidth',2);grid
h=set(gcf,'Current Axes');
ylabel('\fontname{times}20 log_{10}|\itW_{\rmkaiser}\rm(\ite^{j\omega}\rm)|')
xlabel('\fontname{times}\omega(\times\pi) rad \cdot sample^{-1}')
```



Practice

- Finite Impulse Response (FIR) Filters:

Window Method: Kaiser Window & Its Spectrum -

(2)

Design an FIR Highpass filter to meet the following specifications:

$\omega_p = 0.25\pi \qquad \omega_s = 0.4\pi$

$R_p = 0.2dB \qquad A_s = 55dB$

Solution: Let us use a Kaiser window for this example.

$$\Delta f = \frac{|\omega_s - \omega_p|}{2\pi} = 0.05 \Rightarrow M \geq \frac{A_s - 7.95}{14.36\Delta f} = 65.52, \text{ say M=67}$$

Note: For HPF and BSF, M **must** be odd (cannot use a type II filter).

$$\beta = 0.1102(A_s - 8.7) = 5.1$$

$$\omega_c = \frac{\omega_s + \omega_p}{2} = \frac{0.4\pi + 0.5\pi}{2} = 0.45\pi$$

Matlab Code:

```
wp=0.5*pi;
ws=0.4*pi;
tr_width=abs(ws-wp);
As=55;

%Use Kaiser window
M=ceil((As-7.95)/(14.36*tr_width/(2*pi))+1)+1;
M=2*floor(M/2)+1  %Ensure the M is odd for HPF/BSF

if (M <= 21)  beta=0
elseif  (M < 50)  beta=0.5842*(As-21)^0.4+0.07886*(As-21)
else beta = 0.1102* (As-8.7)
end

n=[0:M-1];
wc=0.5*(ws+wp);   %Ideal HPF cutoff freq.
```

(More matlab codes continue on the next page)

```
%Compute Ideal filter impulse response
hd=ideal_lp(pi,M) - ideal_lp(wc,M);
w_kai=(kaiser (M,beta))';
h=hd.* w_kai;

%Modified freqz command
[db,mag,pha,grd,w]=freqz_m(h,[1]);

%frequency spacing (freqz_m computes 1000 samples)
%of DTFT on the unit whole circle)
delta_w=2*pi/1000;

%Compute actual stopband attn.
As=-round(max(db(1:1:ws/delta_w+1)))

%Plot
subplot(2,2,1); stem(n,hd); title('Ideal Impulse Response');
axis([0 M-1 -0.1 0.3]), xlabel('n'); ylabel('hd(n)');

subplot(2,2,2); stem(n,w_kai); title('Kaiser Window');
axis([0 M-1 0 1.1]),xlabel('n'); ylabel('w(n)');

subplot(2,2,3); stem(n,h); title('Actual Impulse Response');
axis([0 M-1 -0.1 0.3]), xlabel('n'); ylabel('h(n)');

subplot(2,2,4); plot(w/pi,db); title('Magnitude Response in dB');
grid;axis([0 1 -100 10]), xlabel('frequency in /pi units'); ylabel('Decibels');
```

Required Routine I

```
function[db, mag, pha, grd, w]=freqz_m(b,a)
%Modified version of freqz sunroutine
%----------------------------------------
%[db,mag,pha,grd,w]=freqz_m(b,a)
%db=Relative magnitude in dB computed over 0 to pi radians
%mag=absolute magnitude computed over 0 to pi radians
%pha=Phase response in radians over 0 to pi radians
%grd=group delay over 0 to pi radians
%w=501 frequency samples between 0 to pi radians
%b=numerator polynimial of H(z) (for FIR:b=h)
%a=denominator polynomial of H(z) (for FIR: a=[1])
%
[H,w]=freqz(b,a,1000,'whole');
H=(H(1:1:501))';
w=(w(1:1:501))';
mag=abs(H);
db=20*log10((mag+eps)/max(mag));
pha=angle(H);
grd=grpdelay(b,a,w);
%End of function
```

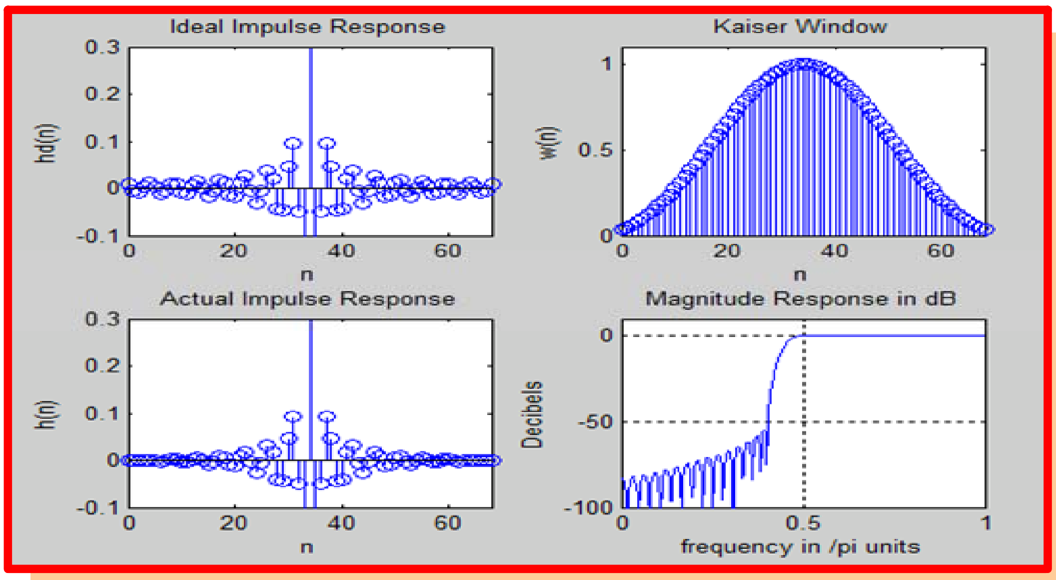
(Required Routine II, matlab output & the plots are shown on the next page)

Required Routine II

```
function hd=ideal_lp(wc,M)
%Ideal LowPass filter computation
%----------------------------------------
%[hd]=ideal_lp(wc,M)
%  hd=ideal impulse response etween 0 to M-1
%  wc=cutoff frequency in radians
%   M=length of the ideal filter
%
alpha=(M-1)/2;
n=[0:1:(M-1)]
M=n-alpha+eps;    %add smallest number to avoid divide by zero
hd=sin(wc*M) ./ (pi*M);
```

Matlab Output

```
M=
     69
beta=
         5.102
Rp=
       0.024
As=
       55
```



Practice

- Finite Impulse Response (FIR) Filters:

Window Method: Kaiser Window & Its Spectrum -

(3)

Design an FIR Bandpass filter to meet the following specifications:

$\omega_{p,1} = 0.25\pi \quad \omega_{p,2} = 0.5\pi \quad \omega_{s,1} = 0.2\pi \quad \omega_{s,2} = 0.6\pi$

$R_p = 0.2dB \quad A_s = 40dB$

Solution : Let us use a Kaiser window for this example.

$$\triangle f = \frac{|\omega_{s,1} - \omega_{p,1}|}{2\pi} = 0.025 \Rightarrow M \geq \frac{A_s - 7.95}{14.36 \triangle f} = 89.28, \text{ say M=90}$$

Note : We use the more stringent of the two transition band widths.

$$\omega_{c,1} = \frac{\omega_{s,1} + \omega_{p,1}}{2} = \frac{0.2\pi + 0.25\pi}{2} = 0.225\pi$$

$$\omega_{c,2} = \frac{\omega_{s,2} + \omega_{p,2}}{2} = \frac{0.5\pi + 0.6\pi}{2} = 0.55\pi$$

Matlab Code :

```
wp1=0.25*pi; wp2=0.5*pi;
ws1=0.2*pi; ws2=0.6*pi;
tr_width=min(abs(ws1-wp1),abs(ws2-wp2));
As=40;

%Use Kaiser window
M=ceil((As-7.95)/(14.36*tr_width/(2*pi))+1)+1;

if (M <= 21)  beta=0
elseif (M < 50) beta=0.5842*(As-21)^0.4+0.07886*(As-21)
else beta=0.1102*(As-8.7)
end

n=([0:M-1]);
wc1=0.5*(ws1+wp1);  %Ideal BPF cutoff freq.
wc2=0.5*(ws2+wp2);

%Compute Ideal filter impulse response
hd=ideal_lp(wc2,M) - ideal_lp(wc1,M);
w_kai=(kaiser(M,beta))';
h=hd.*w_kai;
```

(More matlab codes are shown on the next page)

```
%Modified freqz command
[db,mag,pha,grd,w]=freqz_m(h,[1]);

%frequency spacing (freqz_m computes 1000 samples)
%of DTFT on the unit whole circle)
delta_w=2*pi/1000;

%Compute actual passband ripple
Rp= -(min(db(wp1/delta_w+1:1:wp2/delta_w)))

%Compute actual stopband attn.
As= - round(max(max(db(1:1:ws1/delta_w+1)),max(db(ws2/delta_w+1:1:501))))
%Plot
subplot(2,2,1); stem(n,hd); title('Ideal Impulse Response');
axis([0 M-1 -0.1 0.3]), xlabel('n'); ylabel('hd(n)');

subplot(2,2,2); stem(n,w_kai); title('Kaiser Window');
axis([0 M-1 0 1.1]), xlabel('n'); ylabel('w(n)');

subplot(2,2,3); stem(n,h); title('Actual Impulse Response');
axis([0 M-1 -0.1 0.3]), xlabel('n'); ylabel('h(n)');

subplot(2,2,4); plot(w/pi,db); title('Magnitude Response in dB');
grid; axis([0 1 -100 10]), xlabel('frequency in \pi units'); ylabel('Decibels');
```

Required Routine I

```
function[db, mag, pha, grd, w]=freqz_m(b,a)
%Modified version of freqz sunroutine
%----------------------------------------
%[db,mag,pha,grd,w]=freqz_m(b,a)
%db=Relative magnitude in dB computed over 0 to pi radians
%mag=absolute magnitude computed over 0 to pi radians
%pha=Phase response in radians over 0 to pi radians
%grd=group delay over 0 to pi radians
%w=501 frequency samples between 0 to pi radians
%b=numerator polynimial of H(z) (for FIR:b=h)
%a=denominator polynomial of H(z) (for FIR: a=[1])
%
[H,w]=freqz(b,a,1000,'whole');
H=(H(1:1:501))';
w=(w(1:1:501))';
mag=abs(H);
db=20*log10((mag+eps)/max(mag));
pha=angle(H);
grd=grpdelay(b,a,w);
%End of function
```

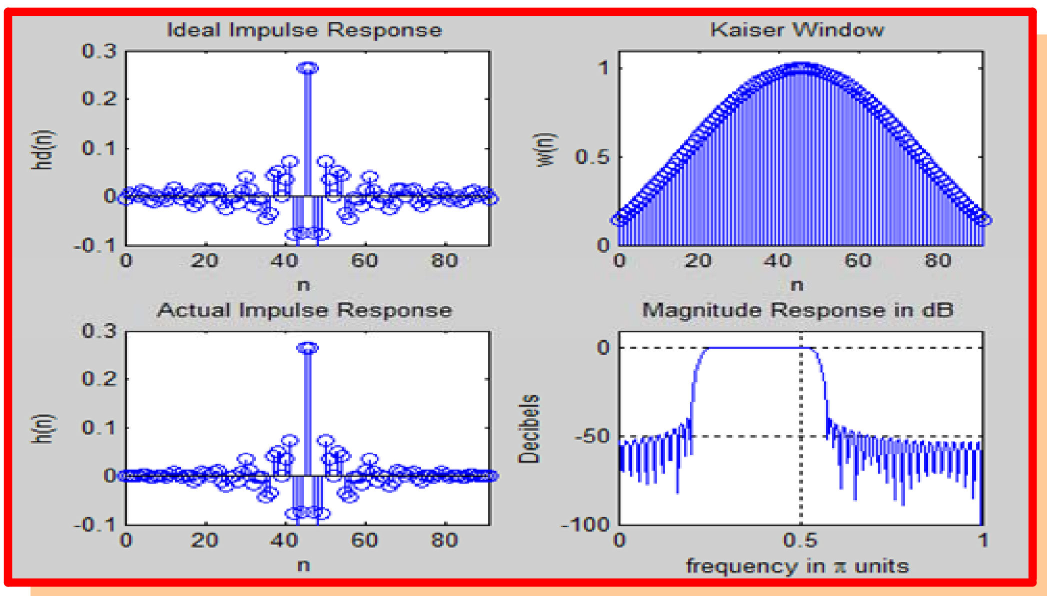
(Required Routine II, matlab output & plots are shown on the next page)

Required Routine II

```
function hd=ideal_lp(wc,M)
%Ideal LowPass filter computation
%----------------------------------------
%[hd]=ideal_lp(wc,M)
%  hd=ideal impulse response etween 0 to M-1
%  wc=cutoff frequency in radians
%   M=length of the ideal filter
%
alpha=(M-1)/2;
n=[0:1:(M-1)]
M=n-alpha+eps;    %add smallest number to avoid divide by zero
hd=sin(wc*M) ./ (pi*M);
```

Matlab Output :

```
M=
     92
beta=
          3.45
Rp=
        1.166
As=
     40
```



A.3 Window-based FIR Filters: *The "fir1" Command*

MATLAB has many commands for designing 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:

1.

```
>>b=fir1(N,Wc);
```

N is the order of the filter, and Wc is a normalized cutoff frequency.

2.

```
>>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 cutoff frequency **wn** (normalized by pi; it is the frequency in radians/sample divided by pi) 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** (unlike other methods, here Wn corresponds to the 6 dB point) is a two-element vector, wn=[w1,w2], **fir1** returns a bandpass filter with passband $w1<w<w2$.

3.

```
>>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
```

Note

*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.*

Ex:>>b=fir1(N-1,Wn,'high',rectwin (N))

creats a highpass filter with cutoff frequcny Wn using a rectangular window.

Practice

**- Finite Impulse Response (FIR) Filters:
Window-based FIR filters: The “fir1” Command -
(1)**

Design a 48-th order FIR bandpass filter with passband 0.35<w<0.65:

```
b=fir1(48,[0.35 0.65]);
freqz(b,1,512)
```

Practice

**- Finite Impulse Response (FIR) Filters:
Window-based FIR filters: The “fir1” Command -
(2)**

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
                                       (digital frequency 0.4 pi)
>>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’)
```

Practice

**- Finite Impulse Response (FIR) Filters:
Window-based FIR filters: The “fir1” Command -
(3)**

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

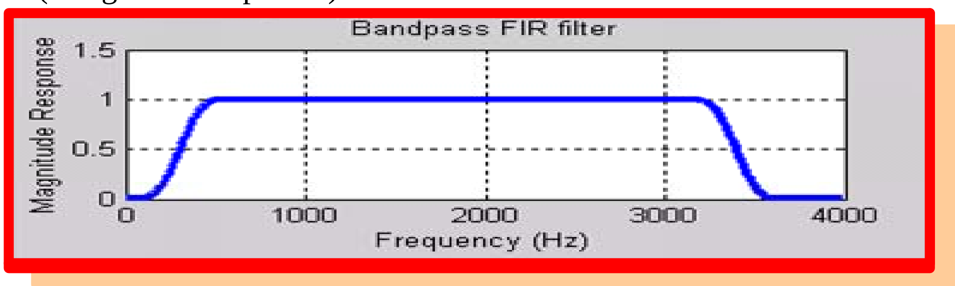
```
>>N=37;                          % specify order of filter
>>wn=0.6;                        % specify the cutoff frequency in units of pi
>>hu=fir1(N,wn,rectwin(N+1));    % windowed FIR coefficients (uniform window)
>>hb=fir1(N,wn,Blackman(N+1));   % windowed FIR coefficients (Blackman window)
>>hk=fir1(N,wn,kaiser(N+1,5.653));% 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’)
```

Practice

- Finite Impulse Response (FIR) Filters:
Window-based FIR filters: The "fir1" Command -
(4)

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
```



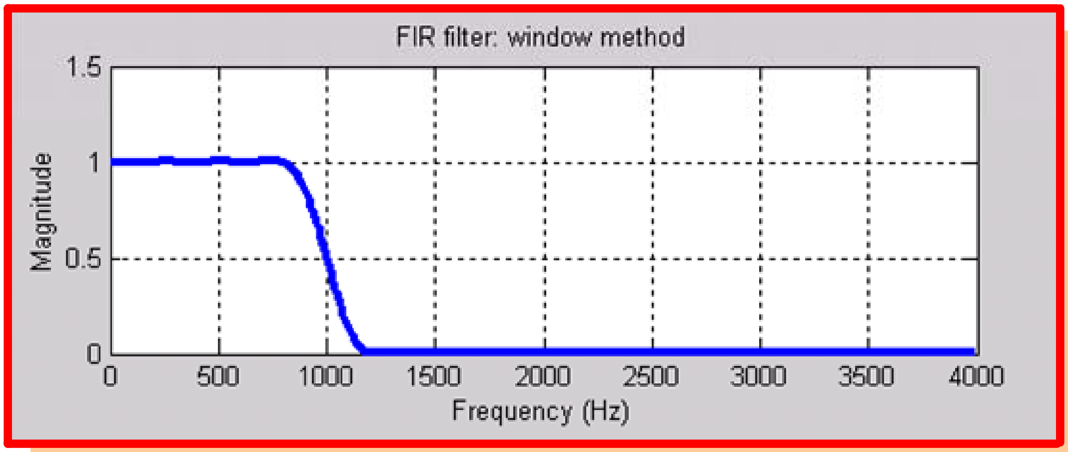
Practice

- Finite Impulse Response (FIR) Filters:
Window-based FIR filters: The "fir1" Command -
(5)

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
```

(The plots are shown on the next page)



Practice

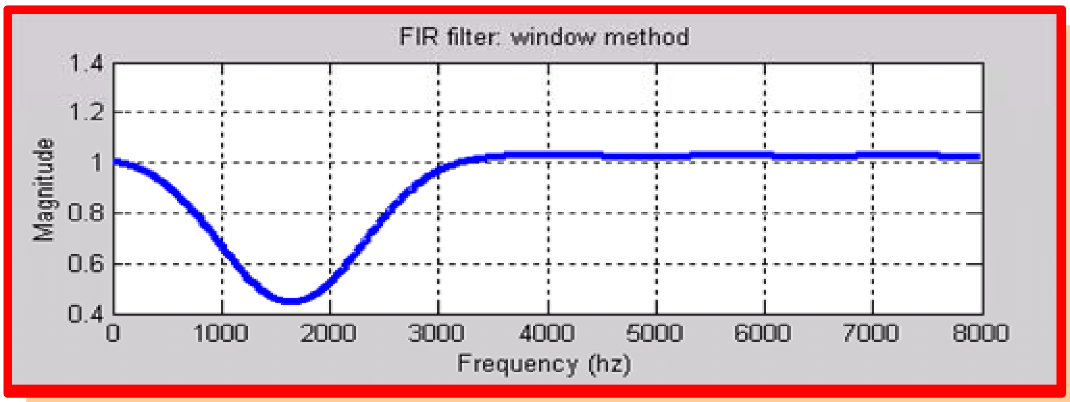
- Finite Impulse Response (FIR) Filters:

Window-based FIR filters: The "fir1" Command -

(6)

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

- Finite Impulse Response (FIR) Filters:

Window-based FIR filters: The "fir1" Command -

(7)

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
```

A.4 Windowed Method with an Arbitrary Shape (Multiband FIR Filter Design) *- The "fir2" Command*

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

Syntax

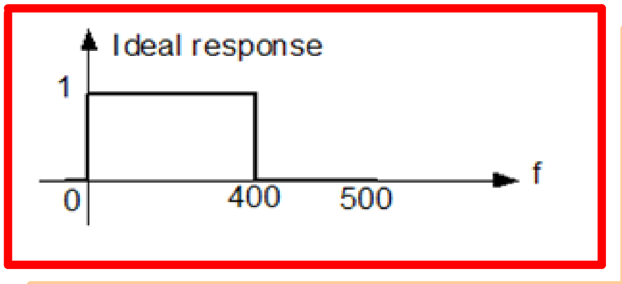
```
>>h=fir2(N,f,m);             %FIR filter with Hamming window
>>h=fir2(N,f,m,window);      %FIR filter with custom window
```

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.

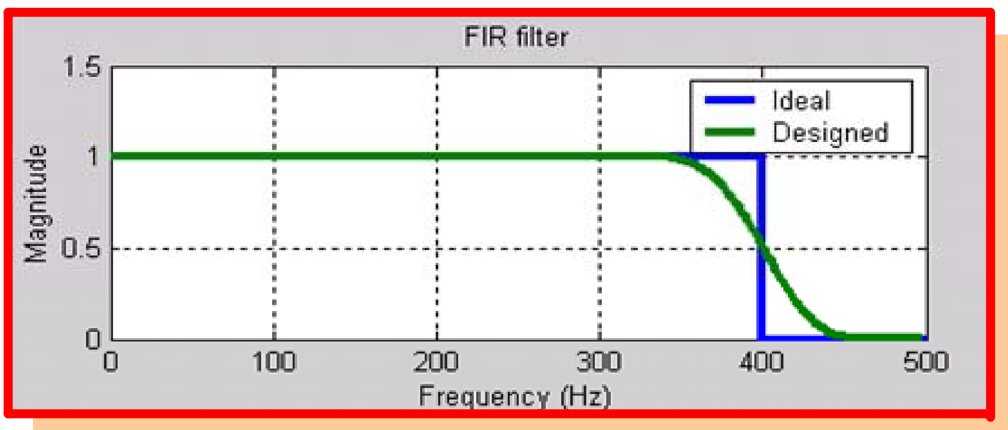
Practice

- Finite Impulse Response (FIR) Filters: Window Method w/ an Arbitrary Shape -

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')
```



B. Parks-MacClellan Optimal FIR Filter Design

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.

□ **Syntax**

>>b=remez(n,f,a,optims)

Returns row vector b containing the (n+1) coefficients of the order in FIR filter where frequency-amplitude characteristics match ? given by vector f and a.

f is a vector of pairs of frequency points, specified in the range vetween o and 1, hwere 1 corresponds to half the sampling frequency. The frequencies must be in increasing order. a is a vector containing the desired amplitudes at the points specified ? and a must be an even number

Practice

- Finite Impulse Response (FIR) Filters: Park-MacClellan Optimal Design -

(1)

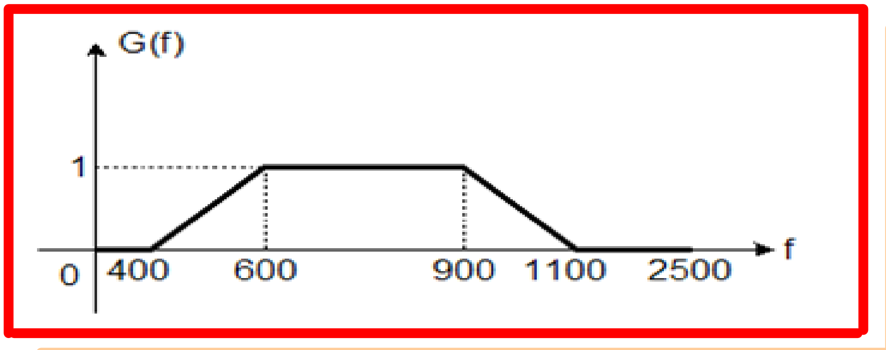
Graph the desired and actual frequency responses of a 17-th order Parks-McClellan bandpass filter:

```
>>f=[0 0.3 0.4 0.6 0.7 1]; a=[0 0 1 1 0 0]
>>b=remez(17,f,a)
>>[n,co]=freqz(b,1,512);plot(f,a,w/pi,abs(h))
```

Practice

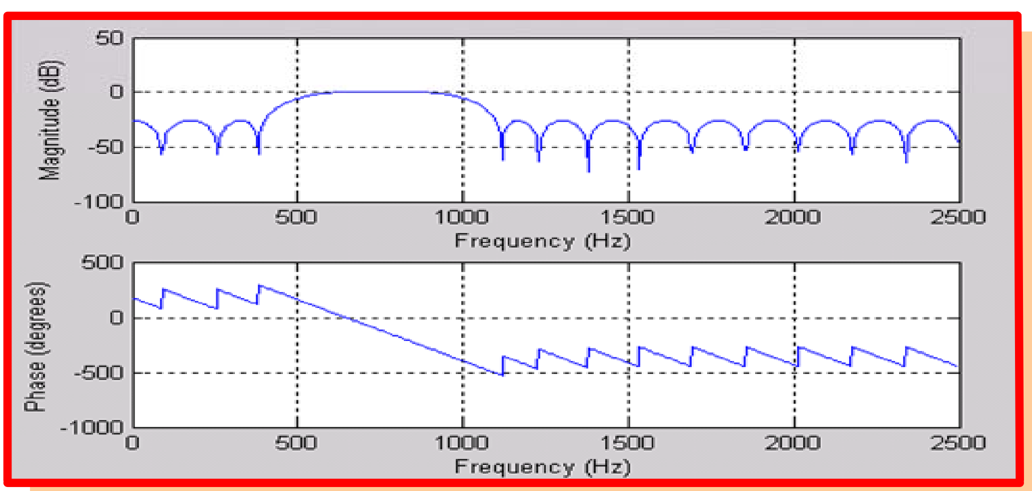
- Finite Impulse Response (FIR) Filters: Parks-MacClellan Optimal Design - (2)

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.

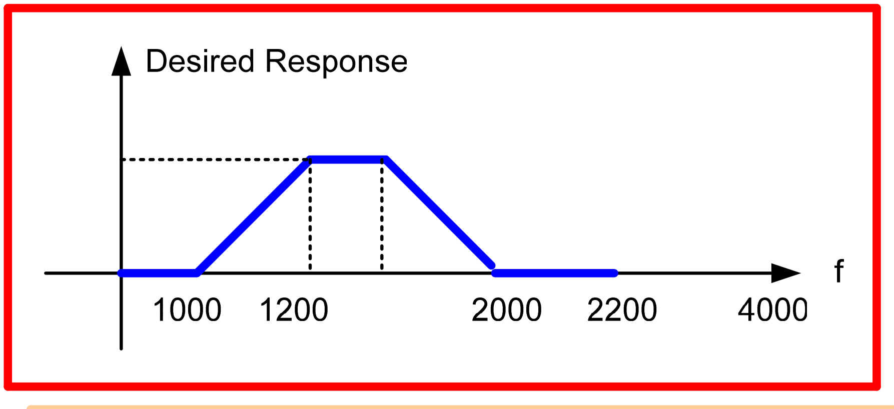
```
>>freq=[0 400/2500 600/2500 900/2500 1100/2500 2500/2500];  % frequency vector
>>gain=[0 0 1 1 0 0];                                        % amplitude vector
>>order=31;                                                  % order of filter
>>coef=remez(order,freq,gain);                            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



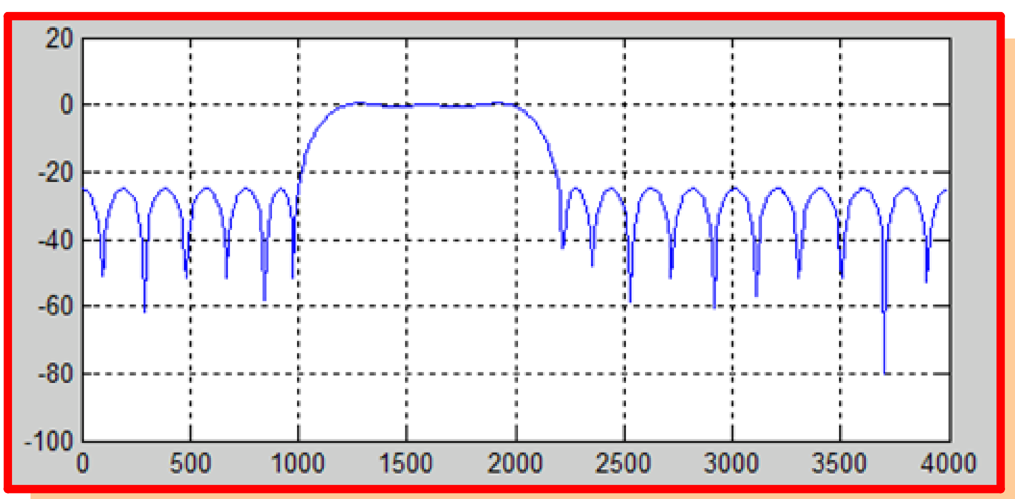
Practice

- Finite Impulse Response (FIR) Filters: Parks-MacClellan Optimal Design - (3)

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
```



Practice

- Finite Impulse Response (FIR) Filters: Parks-MacClellan Optimal Design - (4)

Design an FIR Bandpass filter to meet the following specifications:

$$\omega_{p,1} = 0.25\pi \qquad \omega_{P,2} = 0.5\pi \qquad \omega_{S,1} = 0.2\pi \qquad \omega_{S,2} = 0.6\pi$$
$$R_p = 0.2dB \qquad A_s = 40dB$$

Use the equiripple/minimax design method

Solution : First we find the tolerance δ_1 and δ_2.

$$R_p = -20\log_{10}\left(\frac{1-\delta_1}{1+\delta_1}\right) \Rightarrow \delta_1 = \frac{10^{(R_p/20)}-1}{10^{(R_p/20)}+1} = 0.012$$

$$A_s = -20\log_{10}\left(\frac{\delta_2}{1+\delta_1}\right) \Rightarrow \delta_2 = (1+\delta_1)10^{(-A_s/20)} = 0.01$$

$$\triangle f = \frac{|\omega_{s,1} - \omega_{p,1}|}{2\pi} = 0.025$$

$$\Rightarrow M \geq \frac{-20\log_{10}\sqrt{\delta_1\delta_2} - 13}{14.6\triangle f} + 1 = 72.2, \text{ say M=73}$$

Empirical estimate of M is usually an underestimate. We will use the matlab function **remezord** to estimate M.

Matlab Code :

```
%Given BPF specifications
wp1=0.25*pi; wp2=0.5*pi;
ws1=0.2*pi; ws2=0.6*pi;
As=40; Rp=0.2;

%Calculate delta_1 and delta_2
d1=(10^(Rp/20)-1)/(10^(Rp/20)+1);
d2=(1+d1)*(10^(-As/20));

%vector of tolerances
tolerance=[d2,d1,d2];
```

(More matlab codes continue on the next page)

```
%vector of band edge frequencies (normalized in units of pi)
f=[ws1,wp1,wp2,ws2]/pi;

%vector of nominal amplitudes
amplitude=[0 1 0];

%Estimate order of filter for Remez algorithm
[N,Fo,Ao,W]=remezord(f,amplitude,tolerance);

%Call the remez function for filter design
h=remez(N,Fo,Ao,W);

%Modified freqz command
[db,mag,pha,grd,w]=freqz_m(h,[1]);

%freqency spacing (freqz_m computes 1000 samples)
%of DTFT on the unit whole circle)
delta_w=2*pi/1000;

%compute actual passband ripple
%Rp= max(abs(mag(floor(wp1/delta_w)+1:1:floor(wp2/delta_w)+1)))

%Compute actual stopband attn.
Ass=-round(max(max(db(1:1:floor(ws1/delta_w)+1)),max(db(floor(ws2/delta_w)+1:1:501))))

%If specs. not met, must increase N and repeat process
while (Ass<As)
N=N+1
h=remez(N,Fo,Ao,W);
[db,mag,pha,grd,w]=freqz_m(h,[1]);
Ass= -round(max(max(db(1:1:floor(ws1/delta_w)+1)),max(db(floor(ws2/delta_w)+1:1:501))))
end

%N is filter order, so M=N+1 is filter length
M=N+1
n=[0:1:M-1];

%Plot
subplot(2,2,1); stem(n,h); title('Actual Impulse Response');
axis([0 M-1 -1 1]), xlabel('n'); ylabel('h(n)');

subplot(2,2,2); plot(w/pi,mag); title('Magnitude Response');
grid; axis([0 1 0 1.1]), xlabel('freqnency in \pi units');


subplot(2,2,3); plot(w/pi,mag); title('Passband Ripple');
grid; axis([0 1 1-2*d1 1+2*d1]), xlabel('frequency in \pi units');


subplot(2,2,4); plot(w/pi,mag); title('Stopband Ripple');
grid; axis([0 1 0 2*d2]), xlabel('frequency in \pi units');
```

(Required Routine, matlab output and plots are shown on the next page)

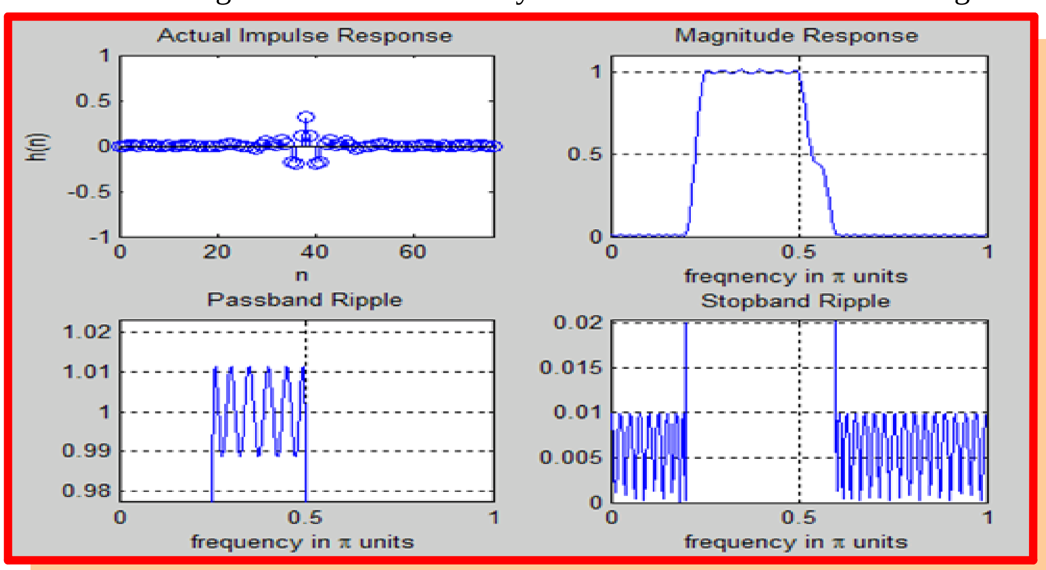
Required Routine

```
function[db, mag, pha, grd, w]=freqz_m(b,a)
%Modified version of freqz sunroutine
%----------------------------------------
%[db,mag,pha,grd,w]=freqz_m(b,a)
%db=Relative magnitude in dB computed over 0 to pi radians
%mag=absolute magnitude computed over 0 to pi radians
%pha=Phase response in radians over 0 to pi radians
%grd=group delay over 0 to pi radians
%w=501 frequency samples between 0 to pi radians
%b=numerator polynimial of H(z) (for FIR:b=h)
%a=denominator polynomial of H(z) (for FIR: a=[1])
%
[H,w]=freqz(b,a,1000,'whole');
H=(H(1:1:501))';
w=(w(1:1:501))';
mag=abs(H);
db=20*log10((mag+eps)/max(mag));
pha=angle(H);
grd=grpdelay(b,a,w);
%End of function
```

Matlab Output :

```
Ass=
       40
M=
     77
```

Note that the filter length of 77 is considerably less than 90 for the window design case.



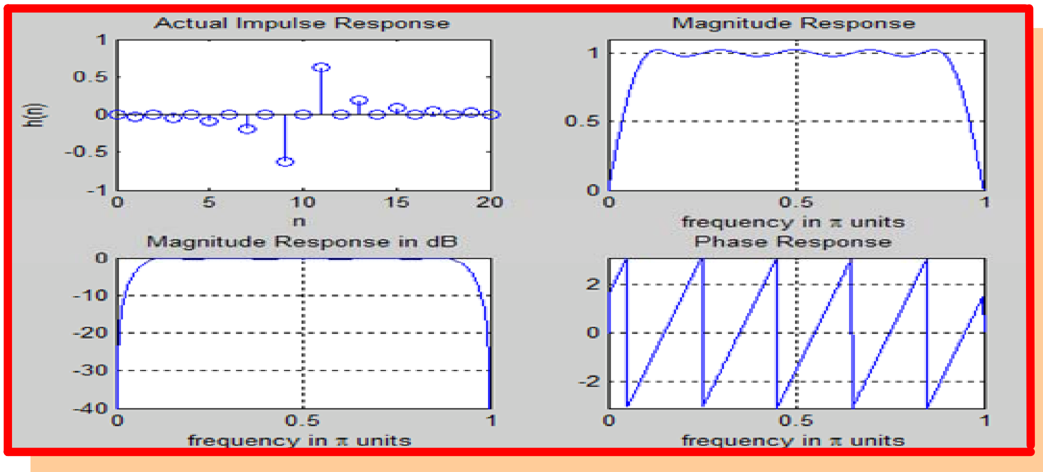
Practice

- Finite Impulse Response (FIR) Filters: Parks-MacClellan Optimal Design - (5)

Design a 21-point FIR Hilbert transformer using the equiripple design method.

Matlab Code :

```
%M=filter length, N=M-1 is filter order
M=21;
N=M-1;
n=[0:1:M-1];

%Want Hilbert transformer to work in the
%freq. band 0.1*pi to 0.9*pi
f=[0.1*pi, 0.9*pi]/pi;
amplitude=[1,1];

%Call the remez function for filter design
h=remez(N,f,amplitude, 'Hilbert');

%Modified freqz command
[db,mag,pha,grd,w]=freqz_m(h,[1]);

%Plot
subplot(2,2,1); stem(n,h); title('Actual Impulse Response');
axis([0 M-1 -1 1]), xlabel('n'); ylabel('h(n)');

subplot(2,2,2); plot(w/pi,mag); title('Magnitude Response');
grid; axis([0 1 0 1.1]), xlabel('frequency in \pi units');

subplot(2,2,3); plot(w/pi,db); title('Magnitude Response in dB');
grid; axis([0 1 -40 0]), xlabel('frequency in \pi units');

subplot(2,2,4); plot(w/pi,pha); title('Phase Response');
grid; axis([0 1 -pi pi]), xlabel('frequency in \pi units');
```



Filter Realization Wizard

Five Realizations

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 FIF
- Lattice (MA)
- Lattice (AR)
- Lattice (ARMA)

Launch the Wizard

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

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

Practice

- Filter Realization Wizard-

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
```

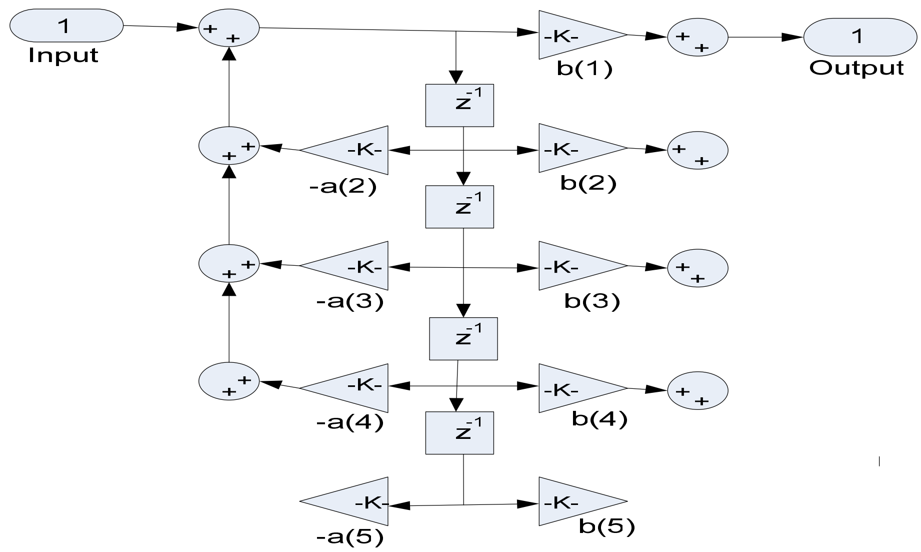
Configure the Wizard

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 use interface with these setting is depicted below:

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:



MATLAB Filter

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).

Practice

-MATLAB Filter-

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
```

- **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
```

Compute 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

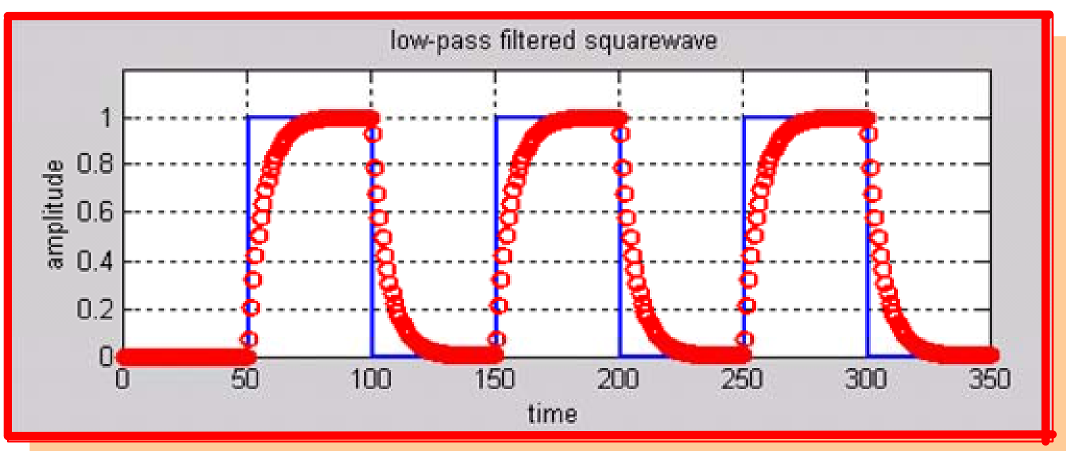
Practice

-MATLAB impz-

(1)

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

-MATLAB impz-

(2)

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')
```