

INDEX

S. No.	Date	Experiments	Page No	Signature
1		Time domain Representation of discrete signals	1-5	
2		Digital Processing of Continuous Signals	6-8	
3		Discrete Time systems in the Time Domain	9-14	
4		Computation of Z transforms and inverse Z transform using Matlab	15-18	
5		Discrete Fourier Transform	19-23	
6		Design of FIR filters using Matlab commands.	24-29	
7		Designing of IIR filters by Matlab commands.	30-36	
8		Filter designing by Matlab tools.	37-39	
9		Design an IIR filter to suppress frequencies of 5 Hz and 30 Hz from given signal.	40-42	

Experiment # 1

Objective:

Time domain Representation of discrete Signals

Description

A signal is described by a function of one or more independent variables. The value of the function (dependent variable) can be a real valued scalar quantity, a complex valued quantity or a vector.

For example:

$$F1(t) = A \sin 3\pi t$$

$$F2(t) = A e^{j3\pi t} = A(\cos 3\pi t + j \sin 3\pi t)$$

The signals may be classified into different categories such as analog, digital, continuous time discrete value and discrete time continuous value.

A discrete time analog signal is denoted as $y(t)$. to emphasize the discrete time nature of a signal, a signal is represented as $y(n)$ instead of $y(t)$

A discrete time signal can be represented in any of the following forms:

- (i) Functional representation
- (ii) Tabular representation
- (iii) Sequence representation
- (iv) Graphical representation

The set of rules for implementing the system by a program that performs the corresponding mathematical operations is called Algorithm.

Some of the Discrete time signals have been discussed in this experiment.

Unit Sample Sequence:

is often called the discrete time impulse or the unit impulse. It is denoted by $\delta[n]$. It is denoted by :

$$\delta[n] = \begin{cases} 1 & , n = 0 \\ 0 & , n \neq 0 \end{cases}$$

Discrete Shifted Unit Impulse

$$\delta[n-k] = \begin{cases} 1 & , n = k \\ 0 & , n \neq k \end{cases}$$

Properties of Unit Impulse Function

Matlab Code for Unit Impulse Generation:

$$\delta[n] = u[n] - u[n-1]$$

$$u[n] = \sum_{k=-\infty}^n \delta[k]$$

$$x[n]\delta[n] = x[0]\delta[n]$$

$$x[n]\delta[n-k] = x[k]\delta[n-k]$$

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k]$$

```
% Program to generate Unit Sample  
clf;  
% generate a vector from -10 to 20  
n=-10:20;  
u=[zeros(1,10) 1 zeros(1,20)];  
% plot the sequence  
stem(n,u);  
xlabel('unit sample sequence');  
axis([-10 20 0 1.2]);
```

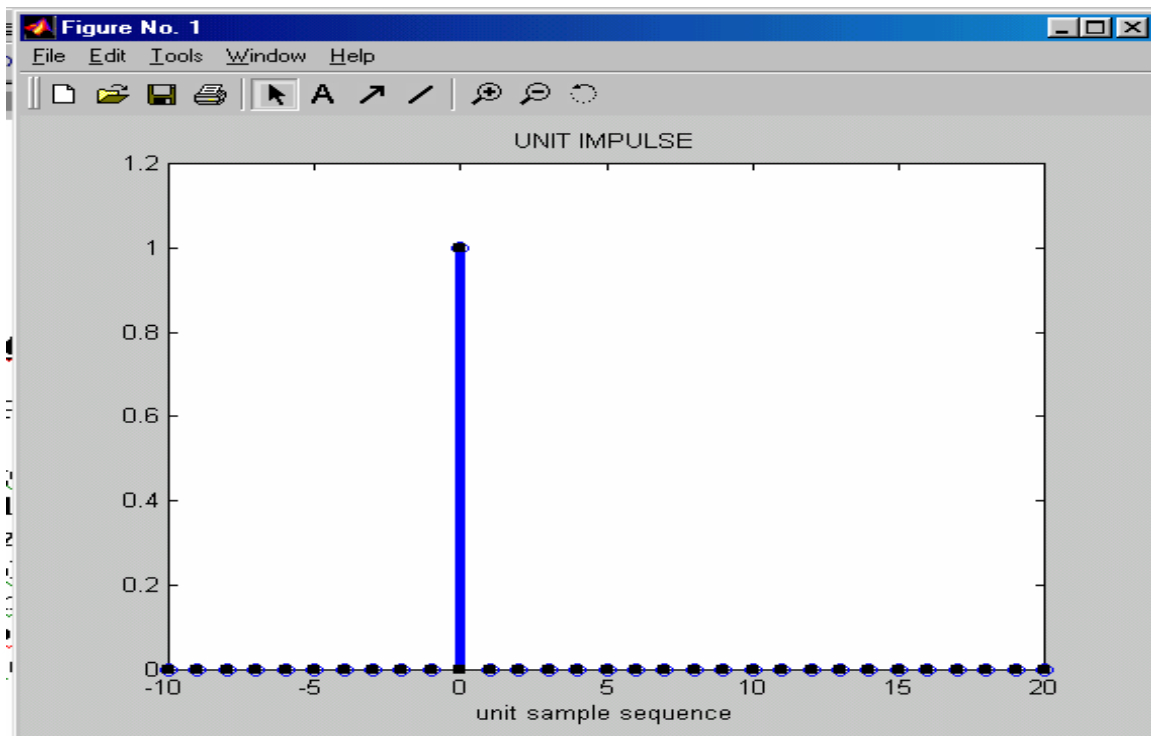


Figure: Unit Impulse

Unit Step Sequence

It is denoted by $u[n]$ and defined by:

$$U(n) = \begin{cases} 1 & , n \geq 0 \\ 0 & , n < 0 \end{cases}$$

Matlab Command To Generate Unit Step

$S = [\text{ones}(1,N)];$

Discrete Time Sinusoidal Signal

A discrete time sinusoidal signal can be either sine or cosine, discrete time signal, it can be expressed by:

$$Y(n) = A \sin(\omega n + \phi) \quad -\infty < n < \infty$$

Where $\omega = 2\pi f$

$$Y(n) = A \sin(2\pi f n + \phi) \quad -\infty < n < \infty$$

Similarly discrete time sinusoidal signal for a cosine function can be expressed as

$$Y(n) = A \cos(2\pi f n + \phi) \quad -\infty < n < \infty$$

Where

N = integer variable (Sampler No)

A = Amplitude of Sinusoid

ω = frequency in radians per sample

ϕ = Phase in radians

As the maximum value of the functions $\sin\phi$ and $\cos\phi$ is unity. A acts as a scaling factor giving maximum and minimum values $\pm A$

Matlab Code For Generation Of A Sinusoidal Sequence

```
% generation of a sinusoidal sequence
```

```
n = 0: 40;
```

```
f=0.1;
```

```
phase=0;
```

```
A=1.5;
```

```
Arg = 2*pi*f*n - phase;
```

```
X= A* cos(Arg);
```

```
Clf; % Clear old graph
```

```
Stem(n,x);
```

```
Axix([0 40 -2 2]);
```

```

Grid;
Title(' sinusoidal sequence');
Xlabel(' time index n')
Ylabel(' Amplitude');
Axis;

```

Exponential Sequence:

Discrete time Complex exponential signal can be written as :

$$X[n] = Ae^{(j\omega + \alpha)n}$$

Euler's Formula

$$e^{j\theta} = \cos \theta + j \sin \theta$$

$$\cos \theta = \frac{1}{2}(e^{j\theta} + e^{-j\theta})$$

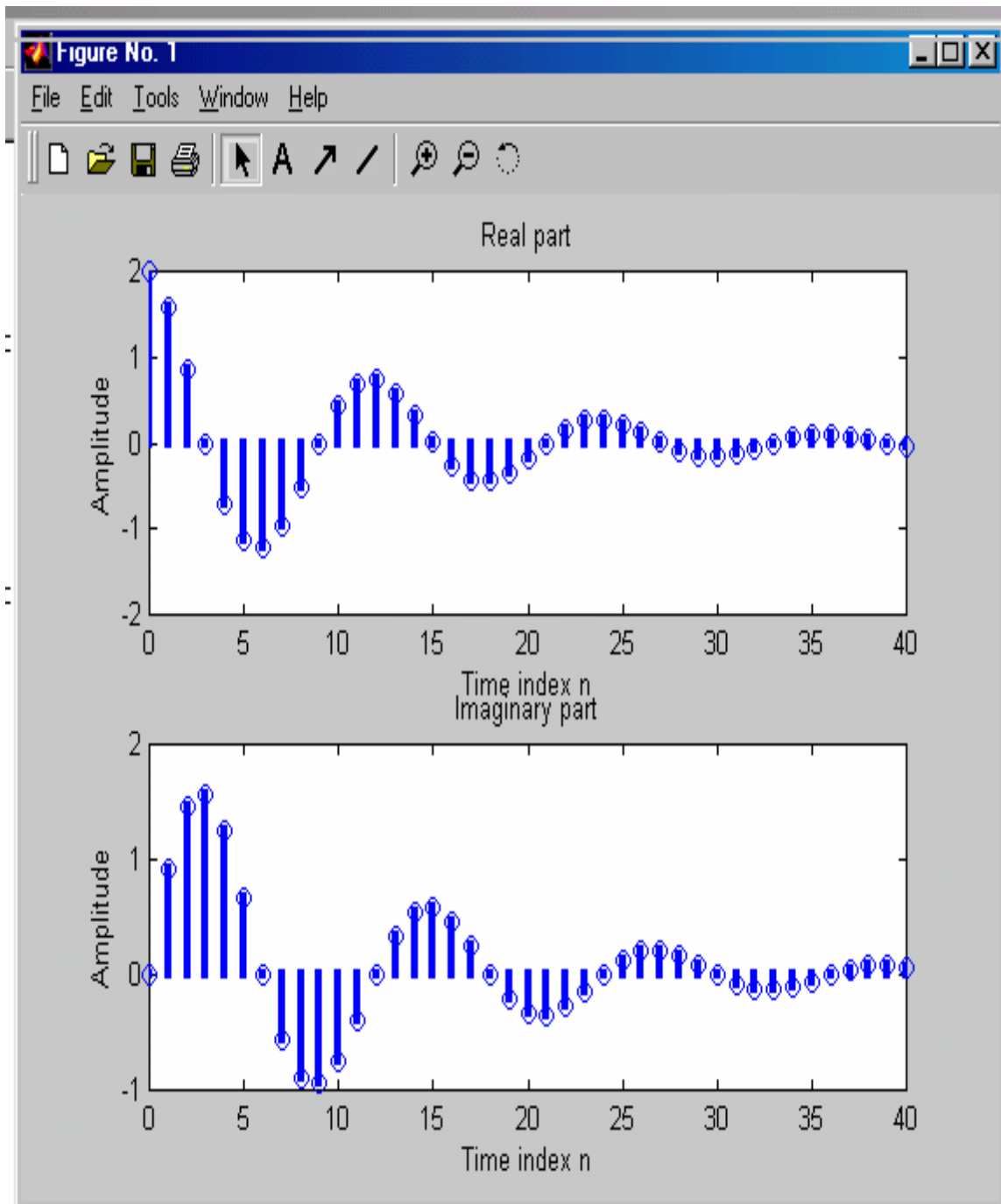
$$\sin \theta = \frac{1}{2j}(e^{j\theta} - e^{-j\theta})$$

Matlab Code For Generation Of Exponential Signal

```

Clf;
C = -(1/12) + (pi/6) * I;
K=2;
N=0:40;
X=k*exp(c*n);
Subplot(2,1,1);
Stem(n,real(x));
Xlabel(' time index n'); ylabel(' Amplitude');
Title(' real part ');
Subplot(2,1,2);
Stem(n, imag(x));
Xlabel(' time index n'); ylabel(' Amplitude');
Title('imaginary part ');

```



Exercise:

- ✚ Write a program in Matlab to generate Exponentially damped sinusoidal signal
- ✚ Write a program in Matlab to generate Saw tooth Waveform.
- ✚ Write a program in Matlab to generate a Triangular Pulse

EXPERIMENT # 2

Objective

Digital Processing of Continuous Signals

Description

Digital processing of analog signals has following advantages over its analog counterpart.

- Programmable Operations
- Greater flexibility
- Higher order of precision
- Better performance.

An analog signal can be converted into digital using the following steps

- ✓ Sampling
- ✓ Quantization
- ✓ Digital Coding

Sampling Of A Sinusoidal Signal

It is the conversion of a continuous time signal into a discrete time signal by obtaining “Samples” of the continuous time signal at discrete time instants. Thus if $X_a(t)$ is the input to the sampler, the output is $X_a(nT) \equiv X(n)$, Where T is called the sampling interval.

Nyquist Sampling Theorem explains, the minimum sampling rate to avoid the Aliasing Effect, should be equal to twice the highest frequency component of the signal

Matlab Code

```
% Illustration of the Sampling Process
clf;
t = 0:0.00005:1;
f = 13;
xa = cos(2*pi*f*t);
subplot(2,1,1)
plot(t,xa); grid;
xlabel('Time, msec');
ylabel('Amplitude');
title('Continuous Time Signal x(at)');
axis([0 1 -1.2 1.2]);
```

```

subplot(2,1,2);
T=0.1;
N=0:T:1;
Xs= cos(2*pi*f*n);
K=0:length(n)-1;
Stem(k,xs); grid;
Xlabel('Time Index n'); ylabel('Amplitude');
Title(' discrete time signal x[n]');
Axis([0 (length(n)-1 -1.2 1.2)]);

```

Reconstruction Of Analog Signal

The Analog signal can be reconstructed from the samples, provided that the sampling rate is sufficiently high to avoid the Aliasing Effect.

Matlab Code To Explain The Aliasing Effect

```

Clf;
t=0:0.0005:1;
F=13;
Ya= cos(2*pi*f*t);
Subplot(2,1,1);
Plot(T,Ya); grid;
Xlabel(' time, msec');
Ylabel('Amplitude');
Axis([0 1 -1.2 1.2]);
Subplot(2,1,2);
T=0.1; f=13;
N=(0:T:1);
Ys= cos(2*pi*f*n);
T=linspace(-0.5,1.5,500);
Tya=sinc((1/T)*t(:,ones(size(n))) - (1/T)*n(:,ones(size(t))))*Ys;
Plot(n,Ys,'o',t,Tya); grid;
Xlabel('Time, msec'); ylabel('Amplitude');
Axis([0 1 -1.2 1.2]);

```

Binary Equivalent Of a Decimal Number

Matlab Code

% This program determines the Binary equivalent of a


```




% _Decimal No.
d = input(' type in a decimal fraction = ');
b = input(' type in the desired wordlength = ');
d1= abs(d);
beq= [zeros(1,b)];
for k=1: b
    int = fix((2*d1));
    beq(k) = int;
    d1 = 2*d1 - int ;
end

if sign(d) == -1;
    bin = [1 beq];
else
    bin = [0 beq];
end

disp(' the binary equivalent is ');
disp(bin);

```

Exercise

-  Write a program in Matlab or C Language to explain the Sampling Process
-  Write a program in Matlab or C Language to Aliasing effect
-  Write a program in Matlab or C Language to explain the Quantization Process

EXPERIMENT # 3

Objective

Discrete Time systems in the Time Domain

Description

A discrete time system processes an input signal in the time domain to generate an output signal with more desirable properties by applying an algorithm composed of simple operations on the input signal and its delayed versions.

Linear & Non Linear Systems

A linear system is one that satisfies the superposition principle. The principle of superposition requires that the response of the system to a weighted sum of signals be equal to the corresponding weighted sum of responses(outputs) of the system to each of the individual input signals.

Mathematically:

$$X[n] = \alpha x_1[n] + \beta x_2[n]$$

The response to the system

$$Y[n] = \alpha y_1[n] + \beta y_2[n]$$

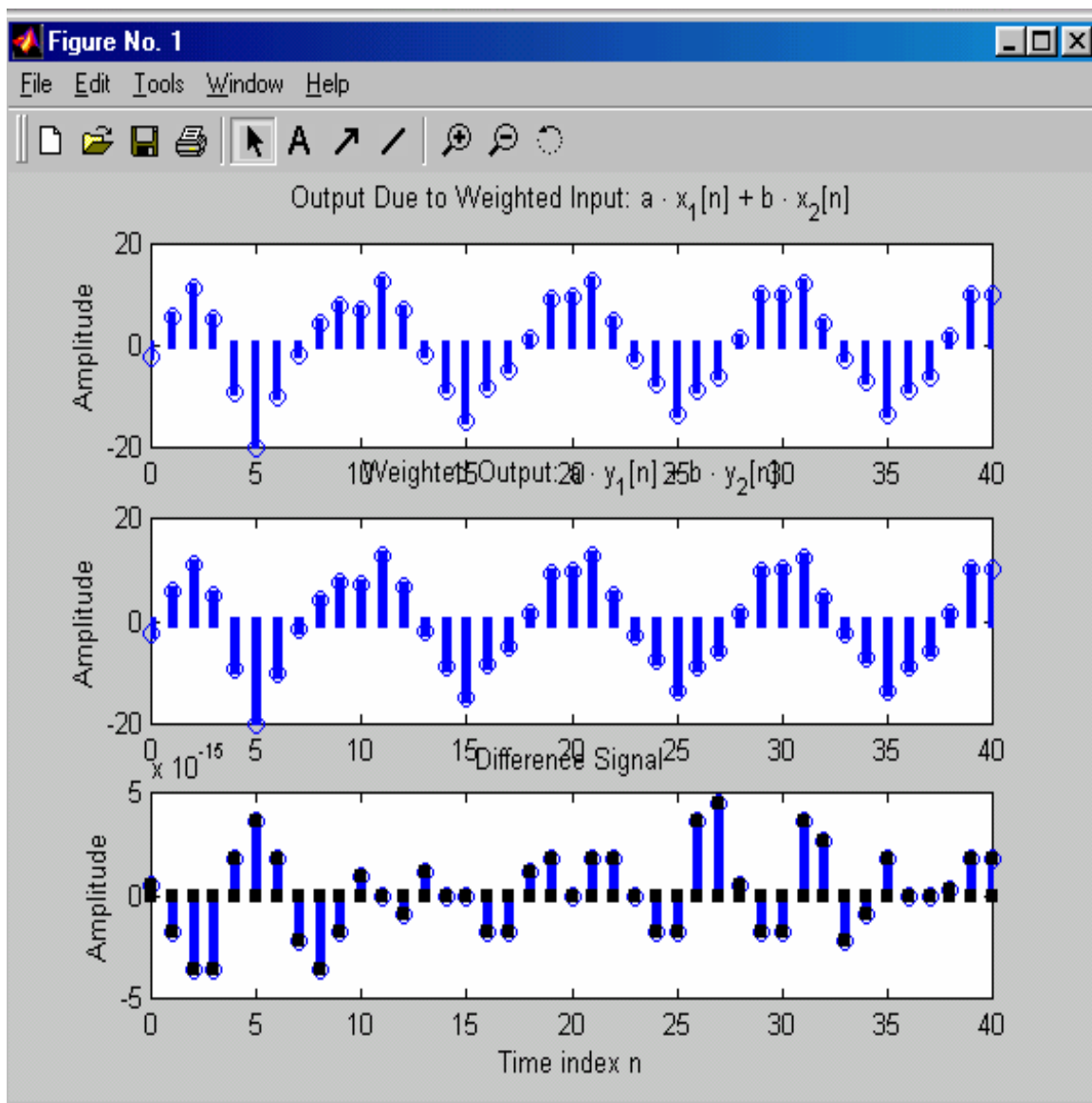
Matlab Code

```
% Generate the input sequences
clf;
n = 0:40;
a = 2;b = -3;
x1 = cos(2*pi*0.1*n);
x2 = cos(2*pi*0.4*n);
x = a*x1 + b*x2;
num = [2.2403 2.4908 2.2403];
den = [1 -0.4 0.75];
ic = [0 0]; % Set zero initial conditions
y1 = filter(num,den,x1,ic); % Compute the output y1[n]
y2 = filter(num,den,x2,ic); % Compute the output y2[n]
y = filter(num,den,x,ic); % Compute the output y[n]
yt = a*y1 + b*y2;
d = y - yt; % Compute the difference output d[n]
% Plot the outputs and the difference signal
```

```

subplot(3,1,1)
stem(n,y);
ylabel('Amplitude');
title('Output Due to Weighted Input:  $a \cdot x_1[n] + b \cdot x_2[n]$ ');
subplot(3,1,2)
stem(n,yt);
ylabel('Amplitude');
title('Weighted Output:  $a \cdot y_1[n] + b \cdot y_2[n]$ ');
subplot(3,1,3)
stem(n,d);
xlabel('Time index n');ylabel('Amplitude');
title('Difference Signal');

```



Time Invariant And Time Variant Systems

A system is called time invariant if its input-output characteristics do not change with time. A related system is time invariant or shift invariant if and only if

$$\begin{array}{ccc} X(n) & \longrightarrow & Y(n) \\ X(n-k) & \longrightarrow & Y(n-k) \end{array}$$

This condition is valid for every input signal $X(n)$ and every time shift k .

Matlab Code

```
% Generate the input sequences
clf;
n = 0:40; D = 10; a = 3.0; b = -2;
x = a*cos(2*pi*0.1*n) + b*cos(2*pi*0.4*n);
xd = [zeros(1,D) x];
num = [2.2403 2.4908 2.2403];
den = [1 -0.4 0.75];
ic = [0 0]; % Set initial conditions
% Compute the output y[n]
y = filter(num,den,x,ic);
% Compute the output yd[n]
yd = filter(num,den,xd,ic);
% Compute the difference output d[n]
d = y - yd(1+D:41+D);
% Plot the outputs
subplot(3,1,1)
stem(n,y);
ylabel('Amplitude');
title('Output y[n]'); grid;
subplot(3,1,2)
stem(n,yd(1:41));
ylabel('Amplitude');
title(['Output due to Delayed Input x[n-D', num2str(D),']']); grid;
subplot(3,1,3)
stem(n,d);
xlabel('Time index n'); ylabel('Amplitude');
title('Difference Signal'); grid;
```

Linear Time Invariant Discrete Time Systems

Linear time Invariant discrete time system satisfies both the linearity and the time invariance properties.

Matlab Code:

Computation Of Impulse Responses Of LTI Systems

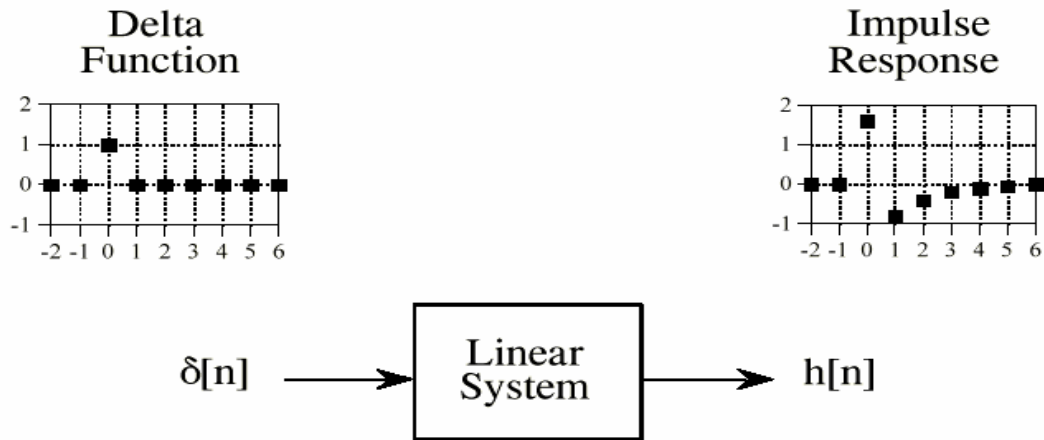
```
Clf;  
N=40;  
Num = [2.2403 2.4908 2.2403];  
Den = [1 -0.4 0.75];  
Y= impz(Num,Den,N);  
Stem(Y);  
Xlabel('Time index n');  
Ylabel('Amplitude');  
Title('Impulse Response'); grid;
```

Convolution

Convolution is a mathematical way of combining two signals to form a third signal. It is the single most important technique in Digital Signal Processing. Using the strategy of impulse decomposition, systems are described by a signal called the *impulse response*. Convolution is important because it relates the three signals of interest: the input signal, the output signal, and the impulse response.

Mathematically

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$



The delta function is a normalized impulse. All of its samples have a value of zero, except for sample number zero, which has a value of one. The Greek letter delta, δ , is used to identify the delta function. The *impulse response* of a linear system, usually $h[n]$ denoted by h , is the output of the system when the input is a delta function.

$$y[n] = x[n] * h[n]$$

The output signal from a linear system is equal to the input signal *convolved* with the system's impulse response. Convolution is denoted by a star when writing equations. Convolution is a formal mathematical operation, just as multiplication, addition, and integration. Addition takes two *numbers* and produces a third *number*, while convolution takes two *signals* and produces a third *signal*. Convolution is used in the mathematics of many fields, such as probability and statistics. In linear systems, convolution is used to describe the relationship between three signals of interest: the input signal, the impulse response, and the output signal.

Matlab Code

```
clf;
h = [3 2 1 -2 1 0 -4 0 3]; % impulse response
x = [1 -2 3 -4 3 2 1]; % input sequence
y = conv(h,x);
n = 0:14;
subplot(2,1,1);
stem(n,y);
xlabel('Time index n'); ylabel('Amplitude');
title('Output Obtained by Convolution'); grid;
x1 = [x zeros(1,8)];
y1 = filter(h,1,x1);
subplot(2,1,2);
```

```
stem(n,y1);  
xlabel('Time index n');  
ylabel('Amplitude');  
title('Output Generated by Filtering'); grid;
```

Exercise

- Write a program in Matlab to explain the Convolution Process
- Write a program in Matlab to explain the Impulse Response of LTI Systems
- Write a program in Matlab to explain Linear and Non Linear Signals

EXPERIMENT # 4

Objective

Computation of Z Transforms and Inverse Z Transforms using Matlab Commands

Description

As analog filters are designed using the Laplace transform, recursive digital filters are developed with a parallel technique called the z-transform. The overall strategy of these two transforms is the same: probe the impulse response with sinusoids and exponentials to find the system's poles and zeros. The Laplace transform deals with differential equations, the s-domain, and the s-plane. Correspondingly, the z-transform deals with difference equations, the z-domain, and the z-plane. However, the two techniques are not a mirror image of each other; the s-plane is arranged in a rectangular coordinate system, while the z-plane uses a polar format. Recursive digital filters are often designed by starting with one of the classic analog filters, such as the Butterworth, Chebyshev, or elliptic. A series of mathematical conversions are then used to obtain the desired digital filter. The Z transform of a discrete time system $X[n]$ is defined as Power Series

Mathematically

$$x(z) = \sum_{n=-\infty}^{n=+\infty} x[n] z^{-n}$$

And the Inverse Z Transform is denoted by:

$$X[n] = Z^{-1} [X(z)]$$

As Z Transform is the infinite Power Series; it exists only for the region for which the series converges (Region of convergence). Inverse Z Transform is the method of inverting the Z Transform of a signal to obtain the time domain representation.

Z Transform Of A Discrete Time Function

$$X(z) = \sum_{n=-\infty}^{n=+\infty} x[n] z^{-n}$$

Matlab Code

```
syms z n
a=ztrans(1/16^n)
```

a =

$$16*z/(16*z-1)$$

Inverse Z-Transform

$$X(n) = Z^{-1} [X(Z)]$$

$$X(Z) = 3*Z / (Z+1)$$

Matlab Code

```
syms Z n
iztrans(3*Z/(Z+1))
```

ans =

$$3*(-1)^n$$

Pole Zero Diagram For A Function In Z Domain

Z plane command computes and display the pole-zero diagram of Z function.
The Command is

Zplane(b,a)

To display the pole value, use **root(a)**

To display the zero value, use **root(b)**

$$X(Z) = [Z^{-2} + Z^{-1}] / [1-2Z^{-1}+3Z^{-2}]$$

Matlab Code

```
b=[0 1 1 ]
a= [1 -2 +3]
roots(a)
roots(b)
```

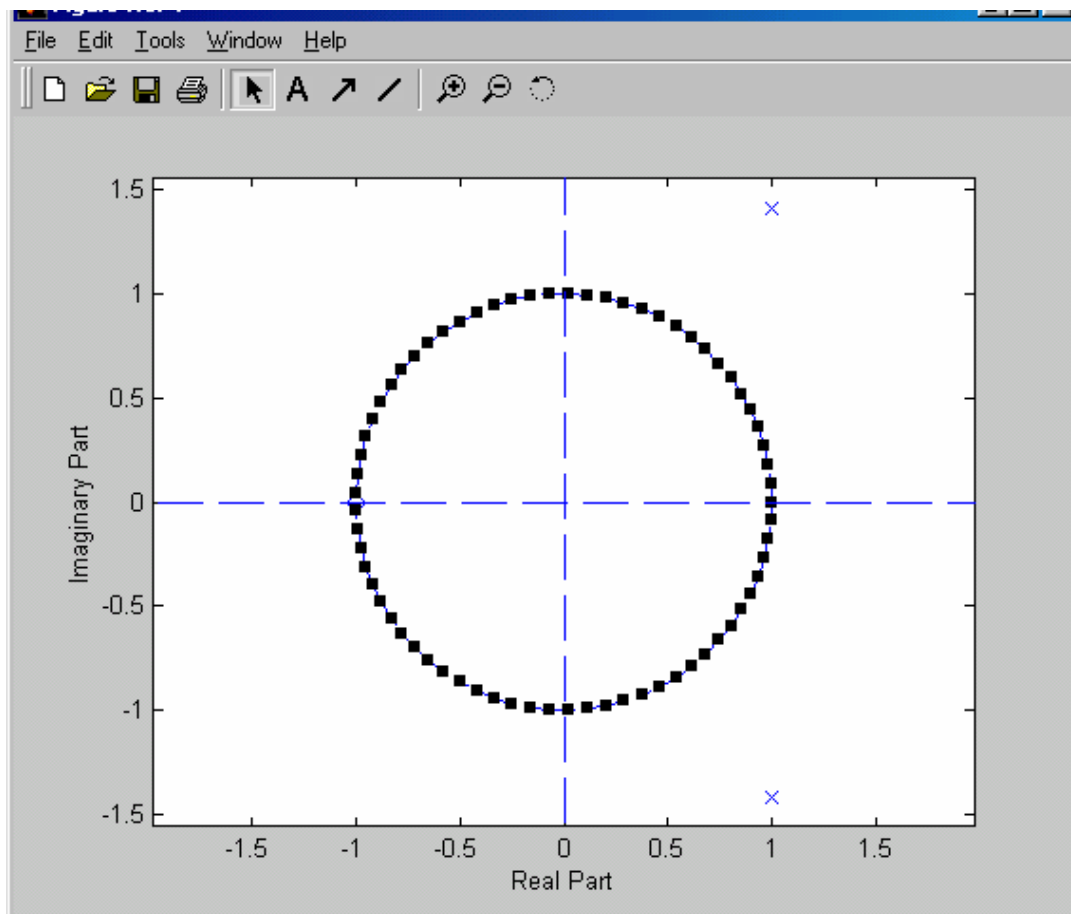
```
zplane(b,a);
```

```
ans =
```

```
1.0000 + 1.4142i  
1.0000 - 1.4142i
```

```
ans =
```

```
-1
```



Frequency Response

The Freqz function computes and display the frequency response of given Z- Transform of the function

```
freqz(b,a,npt,Fs)
```

b= Coeff. Of Numerator

a= Coeff. Of Denominator

Fs= Sampling Frequency

Npt= no. of free points between and Fs/2

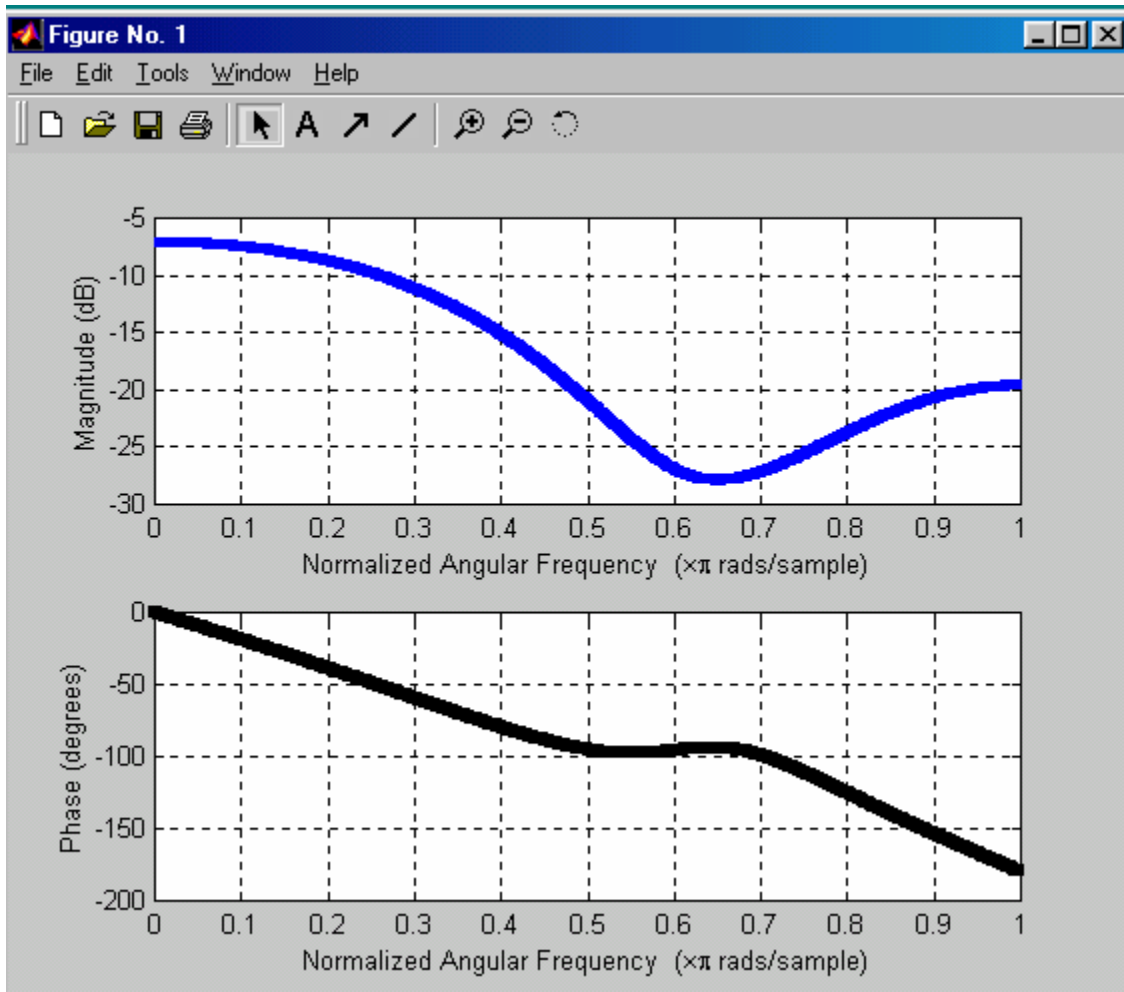
$$X(Z) = [2 + 5Z^{-1} + 9Z^{-2} + 5Z^{-3} + 3Z^{-4}] / [5 + 45Z^{-1} + 2Z^{-2} + Z^{-3} + Z^{-4}]$$

Matlab Code

```
b=[2 5 9 5 3]
```

```
a= [5 45 2 1 1]
```

```
freqz(b,a);
```



Experiment

- ✚ Write a program in Matlab to find the Z transform
- ✚ Write a program in Matlab to find the Inverse Z transform
- ✚ Write a program in Matlab to find Frequency Response

EXPERIMENT # 5

Objective:

Computation of Discrete Time Fourier Transform using Matlab commands

Description

Fourier analysis is a family of mathematical techniques, all based on decomposing signals into sinusoids. The discrete Fourier transform (DFT) is the family member used with *digitized* signals. A signal can be either *continuous* or *discrete*, and it can be either *periodic* or *Aperiodic*. The combination of these two features generates the four categories, described below

➤ Aperiodic-Continuous

This includes, decaying exponentials and the Gaussian curve. These signals extend to both positive and negative infinity *without* repeating in a periodic pattern. The Fourier Transform for this type of signal is simply called the **Fourier Transform**.

➤ Periodic-Continuous

This includes: sine waves, square waves, and any waveform that repeats itself in a regular pattern from negative to positive infinity. This version of the Fourier transform is called the **Fourier series**.

➤ Aperiodic-Discrete

These signals are only defined at discrete points between positive and negative infinity, and do not repeat themselves in a periodic fashion. This type of Fourier transform is called the **Discrete Time Fourier Transform**.

➤ Periodic-Discrete

These are discrete signals that repeat themselves in a periodic fashion from negative to positive infinity. This class of Fourier Transform is sometimes called the Discrete Fourier Series, but is most often called the **Discrete Fourier Transform**.

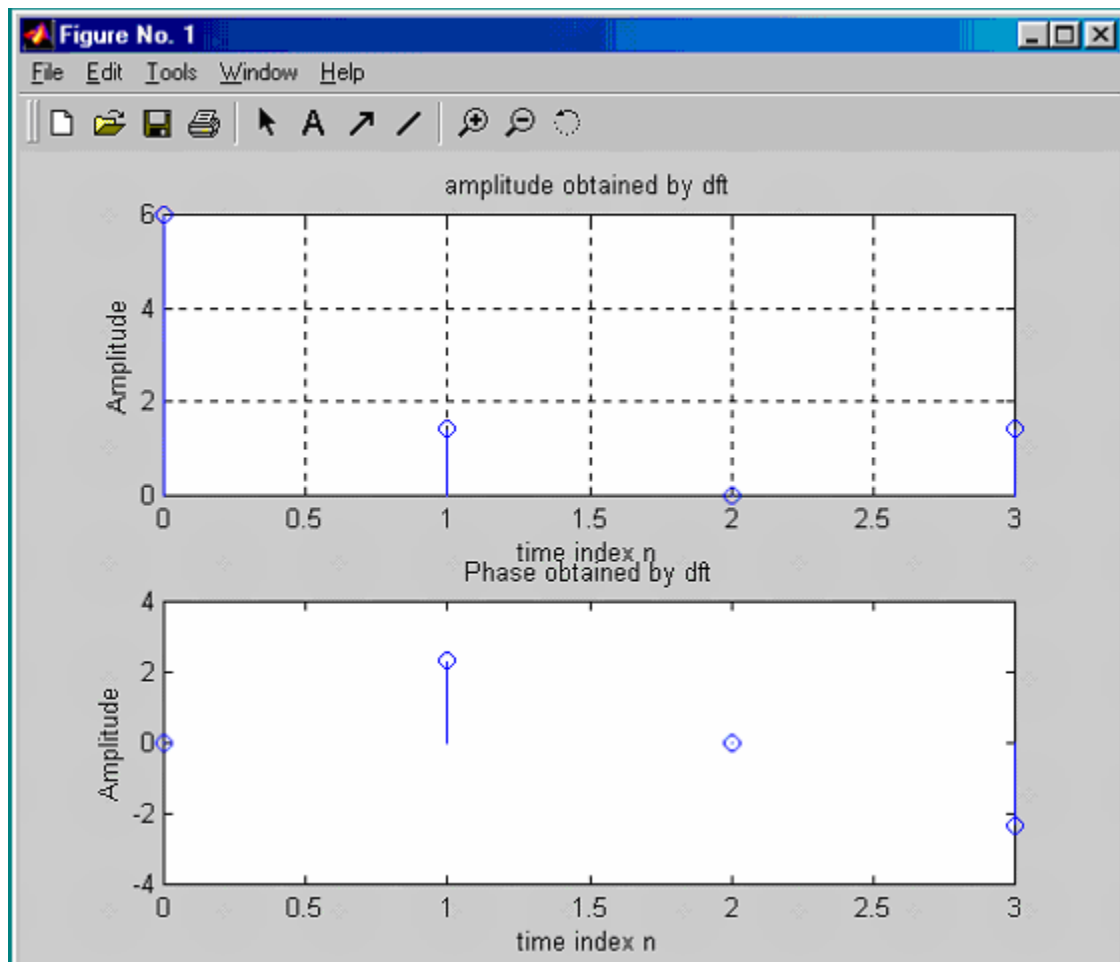
Discrete Fourier Transform Computation:

Mathematical Expression to calculate DFT for an input sequence $x(n)$

$$X(K) = \sum x(n) e^{-j k \Omega n T}$$
$$\Omega = 2\pi/NT$$

Matlab Code

```
clf;
a=[1 1 2 2];
x=fft(a,4);
n=0:3;
subplot(2,1,1);
stem(n,abs(x));
xlabel('time index n');
ylabel('Amplitude');
title('amplitude obtained by dft');
grid;
subplot(2,1,2);
stem(n,angle(x));
xlabel('time index n'); ylabel('Amplitude');
title('Phase obtained by dft');
```



DTFT Computation:

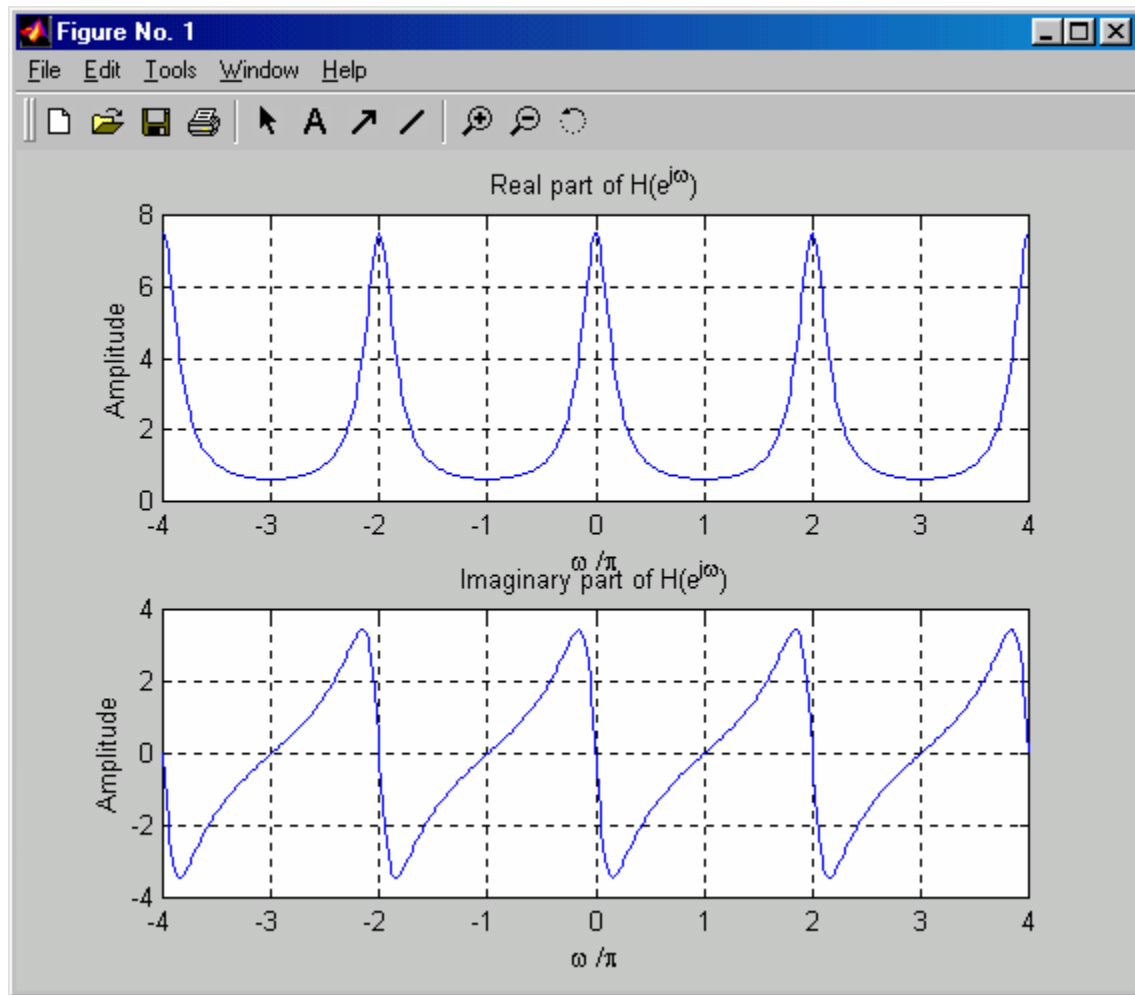
Matlab Code

```
% Evaluation of the DTFT
clf;
% Compute the frequency samples of the DTFT
w = -4*pi:8*pi/511:4*pi;
num = [2 1];den = [1 -0.6];
h = freqz(num, den, w);
% Plot the DTFT
subplot(2,1,1)
plot(w/pi,real(h));grid
title('Real part of H(e^{j\omega})')
xlabel('\omega /\pi');
ylabel('Amplitude');
subplot(2,1,2)
plot(w/pi,imag(h));grid
```

```

title('Imaginary part of  $H(e^{j\omega})$ ')
xlabel('\omega / \pi');
ylabel('Amplitude');
pause
subplot(2,1,1)
plot(w/pi,abs(h));grid
title('Magnitude Spectrum  $|H(e^{j\omega})|$ ')
xlabel('\omega / \pi');
ylabel('Amplitude');
subplot(2,1,2)
plot(w/pi,angle(h));grid
title('Phase Spectrum  $\arg[H(e^{j\omega})]$ ')
xlabel('\omega / \pi');
ylabel('Phase in radians');

```



Exercise

- Write a program in Matlab to find Discrete Fourier Transform
- Write a program in Matlab to find Inverse Discrete Fourier Transform
- Write a program in Matlab to find the Fast Fourier Transform.

EXPERIMENT # 6

Objective:

Design of FIR filters using Matlab commands.

Description:

Digital filters refers to the hard ware and software implementation of the mathematical algorithm which accepts a digital signal as input and produces another digital signal as output whose wave shape, amplitude and phase response has been modified in a specified manner.

Digital filter play very important role in DSP. Compare with analog filters they are preferred in number of application due to following advantages.

- ✚ Truly linear phase response
- ✚ Better frequency response
- ✚ Filtered and unfiltered data remains saved for further use.

There are two type of digital filters.

- ✚ FIR (finite impulse response) filter
- ✚ IIR (infinite impulse response) filter

Description Of The Commands Used In FIR Filter Design

FIR1:

FIR filters design using the window method. $B = \text{FIR1}(N, W_n)$ designs an N'th order low pass FIR digital filter and returns the filter coefficients in length N+1 vector B. **The cut-off frequency W_n must be between $0 < W_n < 1.0$, with 1.0 corresponding to half the sample rate.** The filter B is real and has linear phase. The normalized gain of the filter at W_n is -6 dB.

$B = \text{FIR1}(N, W_n, 'high')$ designs an N'th order highpass filter. You can also use $B = \text{FIR1}(N, W_n, 'low')$ to design a lowpass filter. If W_n is a two-element vector, $W_n = [W1 \ W2]$, FIR1 returns an order N bandpass filter with passband $W1 < W < W2$.

$B = \text{FIR1}(N, W_n, 'stop')$ is a bandstop filter if $W_n = [W1 \ W2]$. You can also specify If W_n is a multi-element vector, $W_n = [W1 \ W2 \ W3 \ W4 \ W5 \ ... \ W_N]$, FIR1 returns an order N multiband filter with bands $0 < W < W1, W1 < W < W2, ..., W_N < W < 1$.

$B = \text{FIR1}(N, W_n, 'DC-1')$ makes the first band a passband.

$B = \text{FIR1}(N, W_n, 'DC-0')$ makes the first band a stopband.

By default FIR1 uses a Hamming window. Other available windows, including Boxcar, Hann, Bartlett, Blackman, Kaiser and Chebwin can be specified with an optional trailing argument. For example, `B = FIR1(N,Wn,kaiser(N+1,4))` uses a Kaiser window with $\beta=4$. `B = FIR1(N,Wn,'high',chebwin(N+1,R))` uses a Chebyshev window.

For filters with a gain other than zero at $F_s/2$, e.g., highpass and bandstop filters, N must be even. Otherwise, N will be incremented by one. In this case the window length should be specified as $N+2$.

By default, the filter is scaled so the center of the first pass band has magnitude exactly one after windowing. Use a trailing 'noscale' argument to prevent this scaling, e.g.

```
B = FIR1(N,Wn,'noscale')
B = FIR1(N,Wn,'high','noscale')
B = FIR1(N,Wn,wind,'noscale').
```

You can also specify the scaling explicitly, e.g. `FIR1(N,Wn,'scale')`, etc.

FREQZ Digital Filter Frequency Response.

`[H,W] = FREQZ(B,A,N)` returns the N -point complex frequency response vector H and the N -point frequency vector W in radians/sample of the filter: given numerator and denominator coefficients in vectors B and A . The frequency response is evaluated at N points equally spaced around the upper half of the unit circle. If N isn't specified, it defaults to 512.

`[H,W] = FREQZ(B,A,N,'whole')` uses N points around the whole unit circle.

`H = FREQZ(B,A,W)` returns the frequency response at frequencies designated in vector W , in radians/sample (normally between 0 and π).

`[H,F] = FREQZ(B,A,N,Fs)` and `[H,F] = FREQZ(B,A,N,'whole',Fs)` return frequency vector F (in Hz), where F_s is the sampling frequency (in Hz).

`H = FREQZ(B,A,F,Fs)` returns the complex frequency response at the frequencies designated in vector F (in Hz), where F_s is the sampling frequency (in Hz).

`[H,W,S] = FREQZ(...)` or `[H,F,S] = FREQZ(...)` returns plotting information to be used with `FREQZPLOT`. S is a structure whose fields can be altered to obtain different frequency response plots. For more information see the help for `FREQZPLOT`.

`FREQZ(B,A,...)` with no output arguments plots the magnitude and unwrapped phase of the filter in the current figure window.

Designing A Low Pass Filter:

Suppose our target is to pass all frequencies below 1200 Hz

```
fs=8000; % sampling frequency
n=50;    % order of the filter
w=1200/ (fs/2);
b=fir1(n,w,'low'); % Zeros of the filter
freqz(b,1,128,8000); % Magnitude and Phase Plot of the filter
figure(2)
[h,w]=freqz(b,1,128,8000);
plot(w,abs(h)); % Normalized Magnitude Plot
grid
figure(3)
zplane(b,1);
```

Designing High Pass Filter:

Now our target is to pass all frequencies above 1200 Hz

```
fs=8000;
n=50;
w=1200/ (fs/2); b=fir1(n,w,'high');
freqz(b,1,128,8000);
figure(2)
[h,w]=freqz(b,1,128,8000);
plot(w,abs(h)); % Normalized Magnitude Plot
grid
figure(3)
zplane(b,1);
```

Designing High Pass Filter:

```
fs=8000;
n=50;
w=1200/ (fs/2);
b=fir1(n,w,'high');
freqz(b,1,128,8000);
figure(2)
[h,w]=freqz(b,1,128,8000);
plot(w,abs(h)); % Normalized Magnitude Plot
grid
figure(3)
zplane(b,1);
```

Designing Band Pass Filter:

```
fs=8000;
n=40;
b=fir1(n,[1200/4000 1800/4000],'bandpass');
freqz(b,1,128,8000)
figure(2)
[h,w]=freqz(b,1,128,8000);
plot(w,abs(h)); % Normalized Magnitude Plot
grid
figure(3)
zplane(b,1);
```

Designing Band Pass Filter:

```
fs=8000;
n=40;
b=fir1(n,[1200/4000 2800/4000],'stop');
freqz(b,1,128,8000)
figure(2)
[h,w]=freqz(b,1,128,8000);
plot(w,abs(h)); % Normalized Magnitude Plot
grid
figure(3)
zplane(b,1);
```

Designing Notch Filter

```
fs=8000;
n=40;
b=fir1(n,[1500/4000 1550/4000],'stop');
freqz(b,1,128,8000)
figure(2)
[h,w]=freqz(b,1,128,8000);
plot(w,abs(h)); % Normalized Magnitude Plot
grid
figure(3)
zplane(b,1);
```

Designing Multi Band Filter

```
n=50;
w=[0.2 0.4 0.6];
b=fir1(n,w);
```

```

freqz(b,1,128,8000)
figure(2)
[h,w]=freqz(b,1,128,8000);
plot(w,abs(h)); % Normalized Magnitude Plot
grid
figure(3)
zplane(b,1);

```

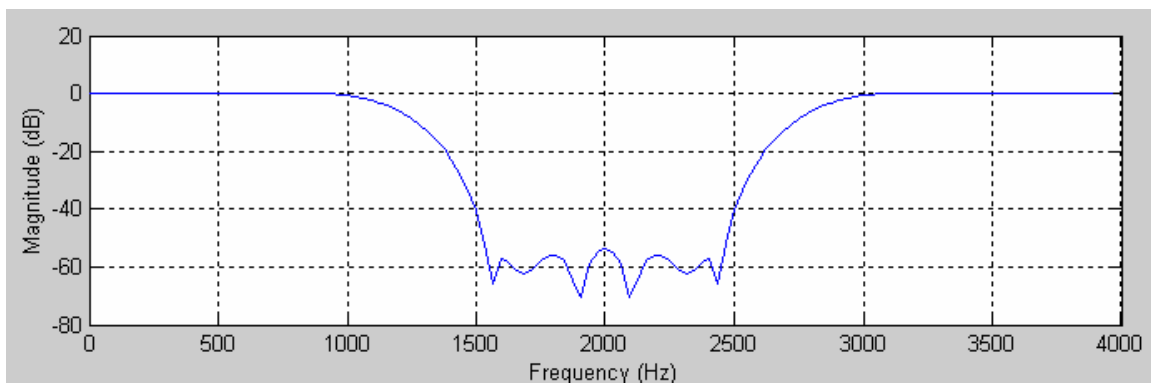
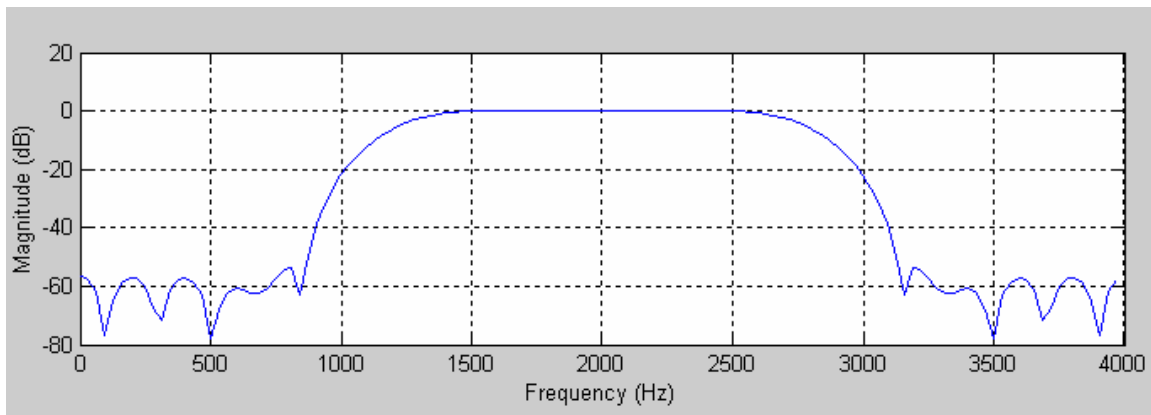
Problems:

Design a **band pass** filter and **band stop** filter with the help of LPF and HPF

The filter has following specifications.

Band pass = 1200 – 2800 Hz

Band stop = 1200-2800 Hz



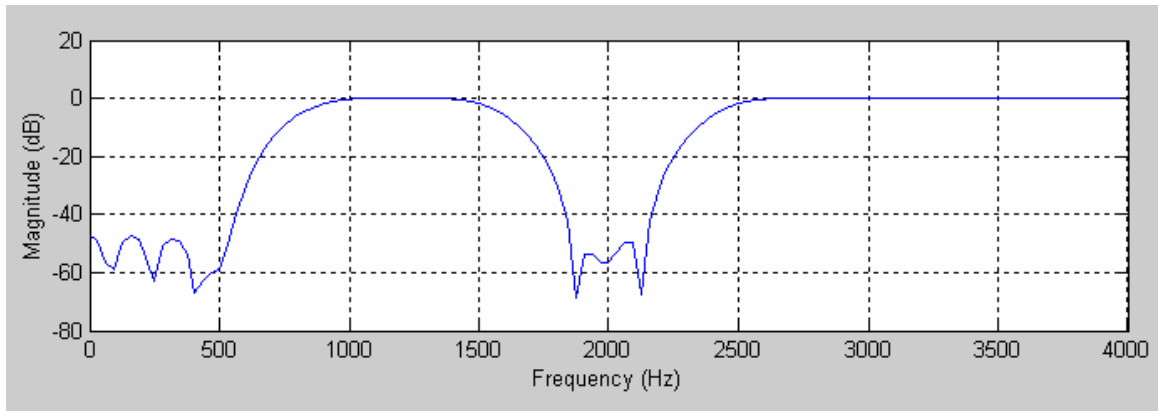
Design a Multi band filter using HPF and LPF

The filter has following specifications

Pass band=1200 Hz – 1800 Hz

Stop band = 1900 Hz – 2200 Hz

Pass band = 2300 Hz – 2700 Hz



EXPERIMENT # 7

Objective:

Designing of IIR filters by Matlab commands.

Description:

Matlab contains various routines for design and analyzing digital filter IIR. Most of these are part of the signal processing tool box. A selection of these filters is listed below.

- ✚ Buttdord (for calculating the order of filter)
- ✚ Butter (creates an IIR filter)
- ✚ Ellipord (for calculating the order of filter)
- ✚ Ellip (creates an IIR filter)
- ✚ Cheb1ord (for calculating the order of filter)
- ✚ Cheyb1 (creates an IIR filter)

Explanation Of The Commands For Filter Design:

Buttdord:

Butterworth filter order selection.

$[N, W_n] = \text{BUTTORD}(W_p, W_s, R_p, R_s)$ returns the order N of the lowest order digital Butterworth filter that loses no more than R_p dB in the pass band and has at least R_s dB of attenuation in the stop band.

W_p and W_s are the pass band and stop band edge frequencies, normalized from 0 to 1 (where 1 corresponds to π radians/sample). For example

Low pass: $W_p = .1, \quad W_s = .2$
High pass: $W_p = .2, \quad W_s = .1$
Band pass: $W_p = [.2 \ .7], \quad W_s = [.1 \ .8]$
Band stop: $W_p = [.1 \ .8], \quad W_s = [.2 \ .7]$

BUTTORD also returns W_n , the Butterworth natural frequency (or, the "3 dB frequency") to use with BUTTER to achieve the specifications.

$[N, W_n] = \text{BUTTORD}(W_p, W_s, R_p, R_s, 's')$ does the computation for an analog filter, in which case W_p and W_s are in radians/second. When R_p is chosen as 3 dB, the W_n in BUTTER is equal to W_p in BUTTORD.

Ellipord:

Elliptic filter order selection.

$[N, Wn] = \text{ELLIPORD}(Wp, Ws, Rp, Rs)$ returns the order N of the lowest order digital elliptic filter that loses no more than Rp dB in the pass band and has at least Rs dB of attenuation in the stop band. Wp and Ws are the pass band and stop band edge frequencies, normalized from 0 to 1 (where 1 corresponds to π radians/sample). For example,

Low pass: $Wp = .1, \quad Ws = .2$
High pass: $Wp = .2, \quad Ws = .1$
Band pass: $Wp = [.2 \ .7], \quad Ws = [.1 \ .8]$
Band stop: $Wp = [.1 \ .8], \quad Ws = [.2 \ .7]$

ELLIPORD also returns Wn , the elliptic natural frequency to use with ELLIP to achieve the specifications.

$[N, Wn] = \text{ELLIPORD}(Wp, Ws, Rp, Rs, 's')$ does the computation for an analog filter, in which case Wp and Ws are in radians/second. NOTE: If Rs is much greater than Rp , or Wp and Ws are very close, the estimated order can be infinite due to limitations of numerical precision.

Cheb1ord:

Chebyshev Type I filter order selection.

$[N, Wn] = \text{CHEB1ORD}(Wp, Ws, Rp, Rs)$ returns the order N of the lowest order digital Chebyshev Type I filter that loses no more than Rp dB in the pass band and has at least Rs dB of attenuation in the stop band. Wp and Ws are the pass band and stop band edge frequencies, normalized from 0 to 1 (where 1 corresponds to π radians/sample). For example,

Low pass: $Wp = .1, \quad Ws = .2$
High pass: $Wp = .2, \quad Ws = .1$
Band pass: $Wp = [.2 \ .7], \quad Ws = [.1 \ .8]$
Band stop: $Wp = [.1 \ .8], \quad Ws = [.2 \ .7]$

CHEB1ORD also returns Wn , the Chebyshev natural frequency to use with CHEBY1 to achieve the specifications.

$[N, Wn] = \text{CHEB1ORD}(Wp, Ws, Rp, Rs, 's')$ does the computation for an analog filter, in which case Wp and Ws are in radians/second.

Butter:

Butterworth digital and analog filter design.

$[B,A] = \text{BUTTER}(N,W_n)$ designs an N th order lowpass digital Butterworth filter and returns the filter coefficients in length $N+1$ vectors B (numerator) and A (denominator). The coefficients are listed in descending powers of z . The cutoff frequency W_n must be $0.0 < W_n < 1.0$, with 1.0 corresponding to half the sample rate.

If W_n is a two-element vector, $W_n = [W_1 \ W_2]$, BUTTER returns an order $2N$ bandpass filter with passband $W_1 < W < W_2$.

$[B,A] = \text{BUTTER}(N,W_n,\text{'high'})$ designs a highpass filter.

$[B,A] = \text{BUTTER}(N,W_n,\text{'stop'})$ is a bandstop filter if $W_n = [W_1 \ W_2]$.

When used with three left-hand arguments, as in $[Z,P,K] = \text{BUTTER}(\dots)$, the zeros and poles are returned in length N column vectors Z and P , and the gain in scalar K . When used with four left-hand arguments, as in $[A,B,C,D] = \text{BUTTER}(\dots)$, state-space matrices are returned.

$\text{BUTTER}(N,W_n,\text{'s'})$, $\text{BUTTER}(N,W_n,\text{'high','s'})$ and $\text{BUTTER}(N,W_n,\text{'stop','s'})$ design analog Butterworth filters. In this case, W_n is in [rad/s] and it can be greater than 1.0.

Ellip:

Elliptic or Causer digital and analog filter design.

$[B,A] = \text{ELLIP}(N,R_p,R_s,W_n)$ designs an N th order low pass digital elliptic filter with R_p decibels of peak-to-peak ripple and a minimum stop band attenuation of R_s decibels. ELLIP returns the filter coefficients in length $N+1$ vectors B (numerator) and A (denominator). The cutoff frequency W_n must be $0.0 < W_n < 1.0$, with 1.0 corresponding to half the sample rate. Use $R_p = 0.5$ and $R_s = 20$ as starting points, if you are unsure about choosing them.

If W_n is a two-element vector, $W_n = [W_1 \ W_2]$, ELLIP returns an order $2N$ band pass filter with pass band $W_1 < W < W_2$. $[B,A] = \text{ELLIP}(N,R_p,R_s,W_n,\text{'high'})$ designs a high pass filter. $[B,A] = \text{ELLIP}(N,R_p,R_s,W_n,\text{'stop'})$ is a band stop filter if $W_n = [W_1 \ W_2]$.

When used with three left-hand arguments, as in $[Z,P,K] = \text{ELLIP}(\dots)$, the zeros and poles are returned in length N column vectors Z and P , and the gain in scalar K . When used with four left-hand arguments, as in $[A,B,C,D] = \text{ELLIP}(\dots)$, state-space matrices are returned.

$\text{ELLIP}(N,R_p,R_s,W_n,\text{'s'})$, $\text{ELLIP}(N,R_p,R_s,W_n,\text{'high','s'})$ and $\text{ELLIP}(N,R_p,R_s,W_n,\text{'stop','s'})$ design analog elliptic filters. In this case, W_n is in [rad/s] and it can be greater than 1.0.

Cheby1:

Chebyshev Type I digital and analog filter design.

`[B,A] = CHEBY1(N,R,Wn)` designs an Nth order lowpass digital Chebyshev filter with R decibels of peak-to-peak ripple in the passband. CHEBY1 returns the filter coefficients in length N+1 vectors B (numerator) and A (denominator). The cutoff frequency Wn must be $0.0 < Wn < 1.0$, with 1.0 corresponding to half the sample rate. Use R=0.5 as a starting point, if you are unsure about choosing R.

If Wn is a two-element vector, $Wn = [W1 \ W2]$, CHEBY1 returns an order 2N bandpass filter with passband $W1 < W < W2$.

`[B,A] = CHEBY1(N,R,Wn,'high')` designs a highpass filter.

`[B,A] = CHEBY1(N,R,Wn,'stop')` is a bandstop filter if $Wn = [W1 \ W2]$.

When used with three left-hand arguments, as in `[Z,P,K] = CHEBY1(...)`, the zeros and poles are returned in length N column vectors Z and P, and the gain in scalar K.

When used with four left-hand arguments, as in `[A,B,C,D] = CHEBY1(...)`, state-space matrices are returned.

`CHEBY1(N,R,Wn,'s')`, `CHEBY1(N,R,Wn,'high','s')` and `CHEBY1(N,R,Wn,'stop','s')` design analog Chebyshev Type I filters. In this case, Wn is in [rad/s] and it can be greater than 1.0.

Buttord and Butter Filter:

Designing IIR Low Pass Filter:

Suppose our target is to design a filter to pass all frequencies below 1200 Hz with pass band ripples = 1 dB and minimum stop band attenuation of 50 dB at 1500 Hz. The sampling frequency for the filter is 8000 Hz;

```
fs=8000;
```

```
[n,w]=buttord(1200/4000,1500/4000,1,50); % finding the order of the filter
```

```
[b,a]=butter(n,w); % finding zeros and poles for filter
```

```
figure(1)
```

```
freqz(b,a,512,8000);
```

```
figure(2)
```

```
[h,q] = freqz(b,a,512,8000);
```

```
plot(q,abs(h)); % Normalized Magnitude plot
```

```
grid
```

```
figure(3)
```

```
f=1200:2:1500;
freqz(b,a,f,8000) % plotting the Transition band
```

```
figure(4)
zplane(b,a) % pole zero constellation diagram
```

Designing IIR High Pass Filter:

We will consider same filter but our target now is to pass all frequencies above 1200 Hz

```
[n,w]=buttord(1200/5000,1500/5000,1,50);
[b,a]=butter(n,w,'high');
figure(1)
freqz(b,a,512,10000);
```

```
figure(2)
[h,q] = freqz(b,a,512,8000);
plot(q,abs(h)); % Normalized Magnitude plot
grid
```

```
figure(3)
f=1200:2:1500;
freqz(b,a,f,10000)
```

```
figure(4)
zplane(b,a)
```

Designing IIR Band Pass Filter:

Now we wish to design a filter to pass all frequencies between 1200 Hz and 2800 Hz with pass band ripples = 1 dB and minimum stop band attenuation of 50 dB. The sampling frequency for the filter is 8000 Hz;

```
[n,w]=buttord([1200/4000,2800/4000],[400/4000, 3200/4000],1,50);
[b,a]=butter(n,w,'bandpass');
figure(1)
freqz(b,a,128,8000)
```

```
figure(2)
[h,w]=freqz(b,a,128,8000);
plot(w,abs(h))
grid
```

```
figure(3)
f=600:2:1200;
```

```
freqz(b,a,f,8000); % Transition Band
```

```
figure(4)  
f=2800:2:3200;  
freqz(b,a,f,8000); % Transition Band
```

```
figure(5)  
zplane(b,a)
```

Designing IIR Band Stop Filter:

```
[n,w]=buttord([1200/4000,2800/4000],[400/4000, 3200/4000],1,50);  
[b,a]=butter(n,w,'stop');  
figure(1)  
freqz(b,a,128,8000)  
[h,w]=freqz(b,a,128,8000);
```

```
figure(2)  
plot(w,abs(h));  
grid
```

```
figure(3)  
f=600:2:1200;  
freqz(b,a,f,8000); % Transition Band
```

```
figure(4)  
f=2800:2:3200;  
freqz(b,a,f,8000); % Transition Band
```

```
figure(5)  
zplane(b,a);
```

Problems

Design all above filter using following commands

-  Ellipord()
-  Ellip()
-  Cheb1ord()
-  Cheby1()

Compare the results of the butter worth LPF with ellip LPF and cheby1 LPF on following basis

- ✚ Order
- ✚ Minimum stop band attenuation achieved
- ✚ Linearity in the phase plots with in the pass band and outside.
- ✚ Pole –zeros plot which filter appears to have pole most closely to the unit circle.


EXPERIMENT # 8

Objective:

Filter designing by Matlab tools.

Description:

There are two tool boxes available for designing, analyzing and for viewing different responses (Impulse & Step) of FIR and IIR filters.

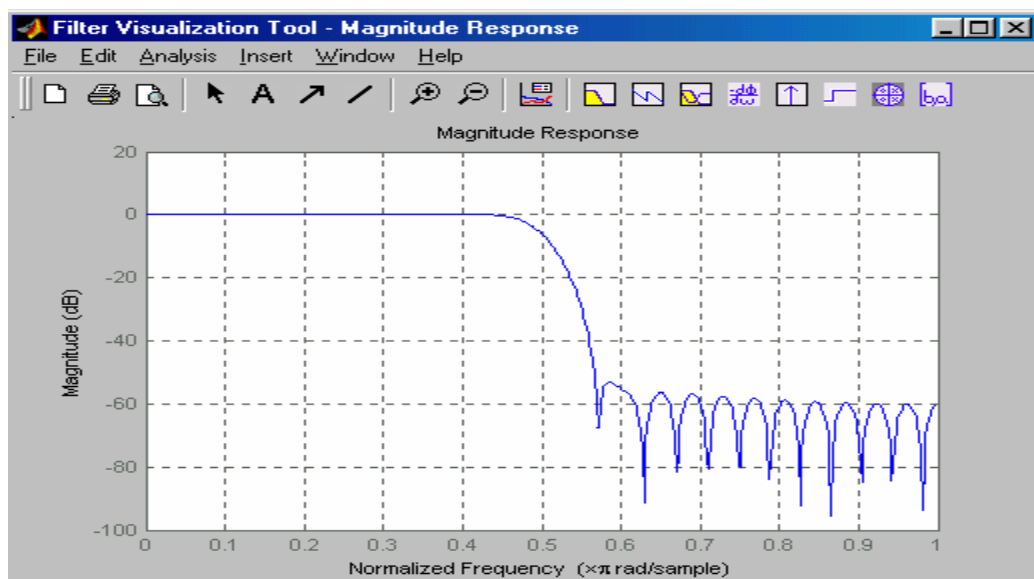
 fvtool
 fdatool

Filter Visualization Tool:

FVTOOL is a Graphical User Interface (GUI) that allows you to analyze digital filters. FVTOOL (B,A) launches the Filter Visualization Tool and computes the magnitude Response for the filter defined in B and A. FVTOOL(B,A,B1,A1,...) will perform an analysis on multiple filters. The real advantage of this visualization tool is that we can view the magnitude response and phase response simultaneously, the impulse response, step response the coefficients of the filter etc

Let us consider a Low Pass FIR filter of order 30 which passes all frequencies below 2000 Hz with sampling rate of 8000 Hz.

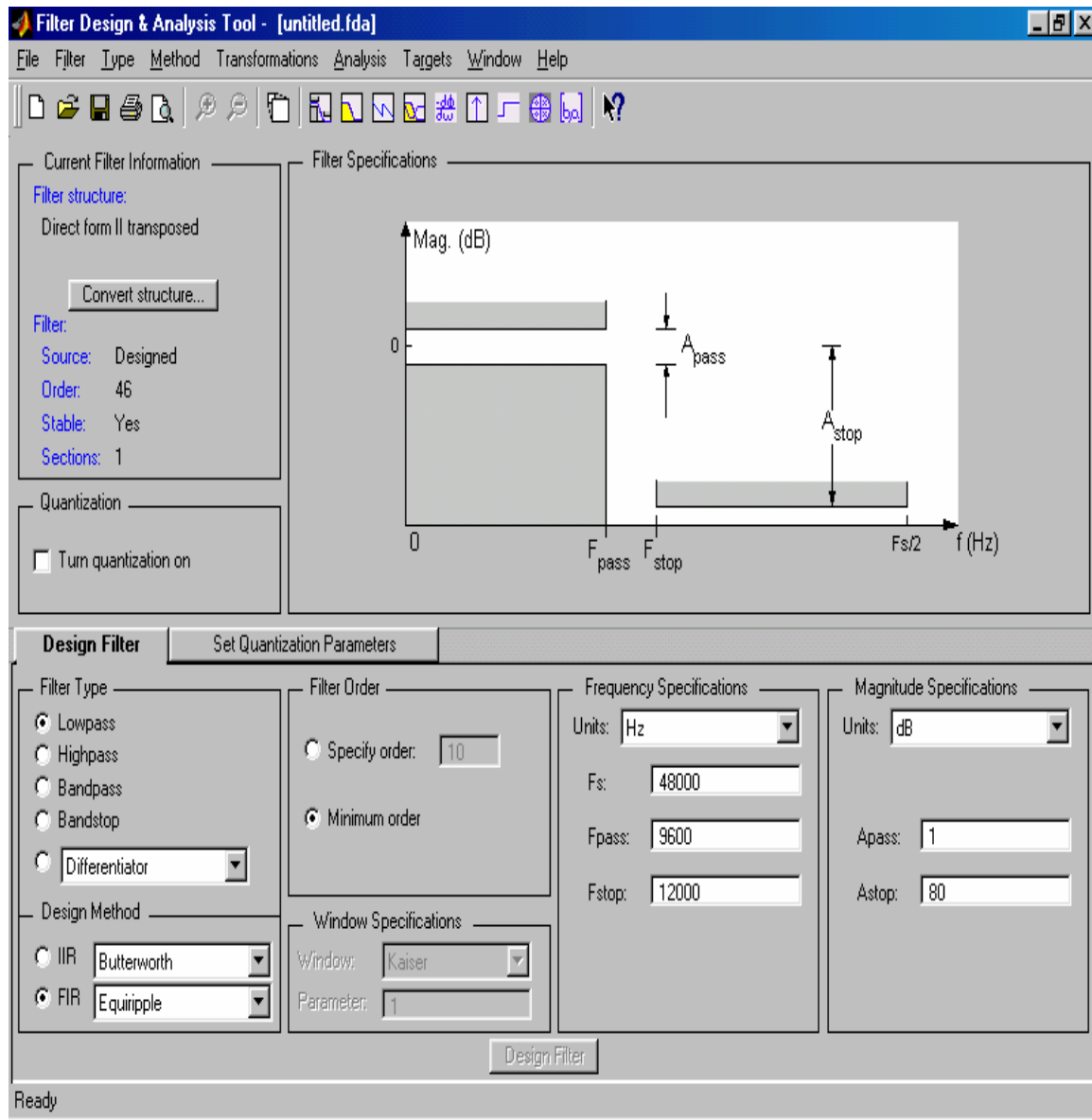
```
b=fir1(30,2000/4000,'low');  
fvtool(b,1)
```



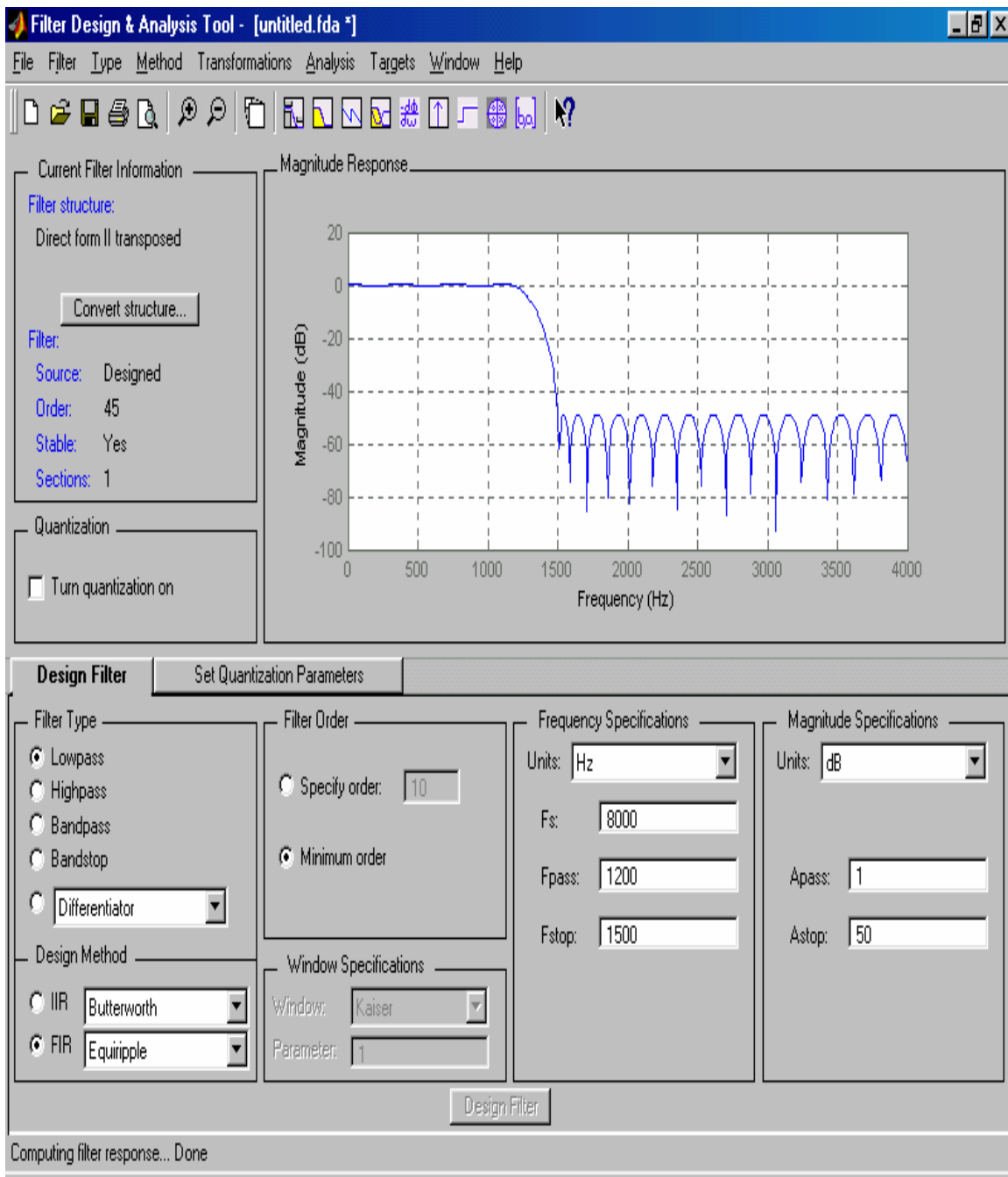
Filter Design & Analysis Tool.

FDATOOL launches the Filter Design & Analysis Tool (FDATool). FDATool is a Graphical User Interface (GUI) that allows you to design or import, and analyze digital FIR and IIR filters.

If the Filter Design Toolbox is installed, FDATool seamlessly integrates advanced filter design methods and the ability to quantize filters.







Now we will design a LPF on fdatool, the specifications for the filter are shown in respective columns of FDA tool



Problems:

Design IIR butter worth filter with following specifications

-  -50 dB or more for 0 to 1200 Hz (Stop Band Attenuation)
-  -1 dB or less from 2000 Hz to 4000 Hz (Pass Band Characteristics)
-  -50 dB or more above 6000 Hz (Stop Band Attenuation)
-  Sampling frequency 16000 Hz

EXPERIMENT # 9

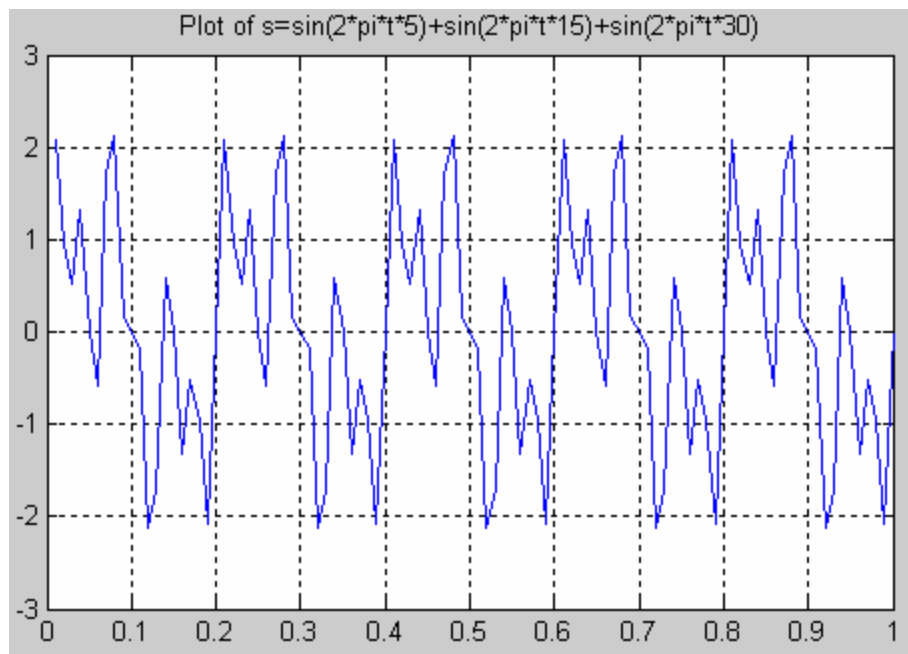
Objective:

Design an IIR filter to suppress frequencies of 5 Hz and 30 Hz from given signal.

Description:

We know from Fourier analysis that signals can be described by a summation of frequency components. Typically, a filter is used to enhance signals by attenuating unwanted frequency components and retaining desired frequency components. In this practical we begin by creating a signals 's' with three sinusoidal components (at 5,15,30 Hz) and a time vector 't' of 100 samples with a sampling rate of 100 Hz, and displaying it in the time domain. The Matlab commands are shown below.

```
fs=100;  
t=(1:100)/fs;  
s=sin(2*pi*t*5)+sin(2*pi*t*15)+sin(2*pi*t*30);  
plot(t,s)  
grid
```



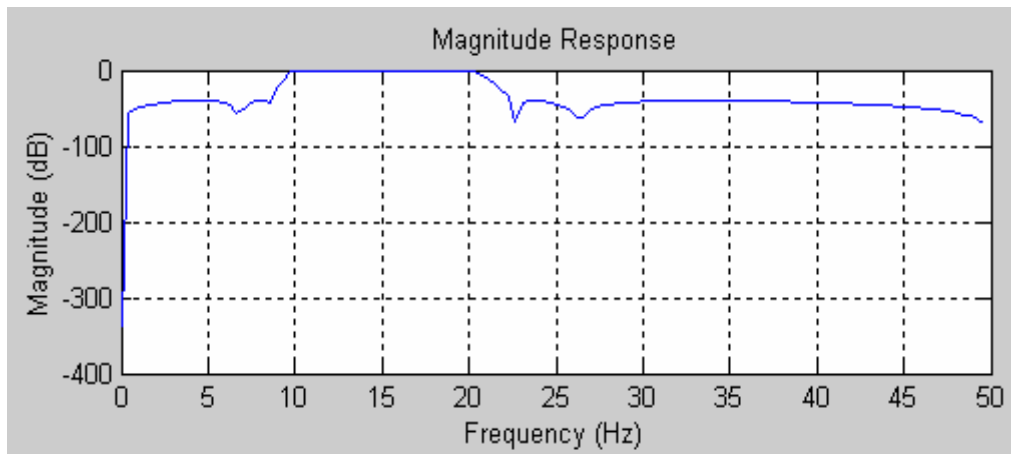
Now we design a filter to keep the 15 Hz sinusoid and eliminate the 5 and 30 Hz sinusoids. We use the functions `ellipord` and `ellip` to create an infinite impulse response (IIR) filter with a pass band from 10 to 20 Hz. The `ellipord` function requires the specification of pass band corner frequencies, minimum transition band frequencies near the pass band corner frequencies, the maximum pass band ripple in decibels (dB), and the

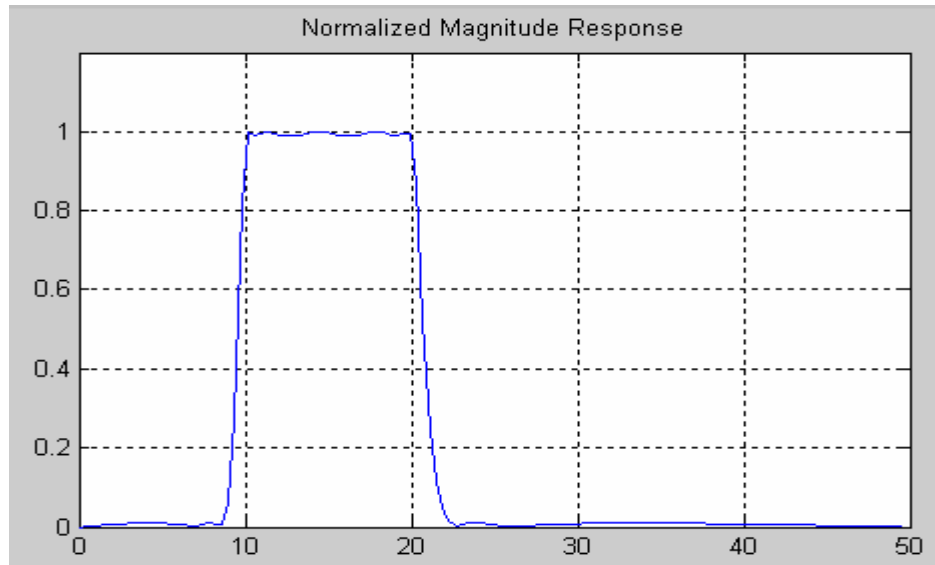
minimum stop band attenuation in dB. In this example, we choose a transition frequency to be ± 5 Hz near the pass band corners, with a maximum of 0.1 dB ripple in the pass band, and a minimum of 40 dB attenuation in the stop bands. We start by determining the minimum order (pass band and stop band frequencies are normalized to the Nyquist frequency):

```
wp1 = 10/50;
wp2 = 20/50;
ws1 = 5/50;
ws2 = 25/50;
wp = [Wp1 Wp2];
ws = [Ws1 Ws2];
rp = 0.1;
rs = 40;
[n,wn] = ellipord(wp,ws,rp,rs);
```

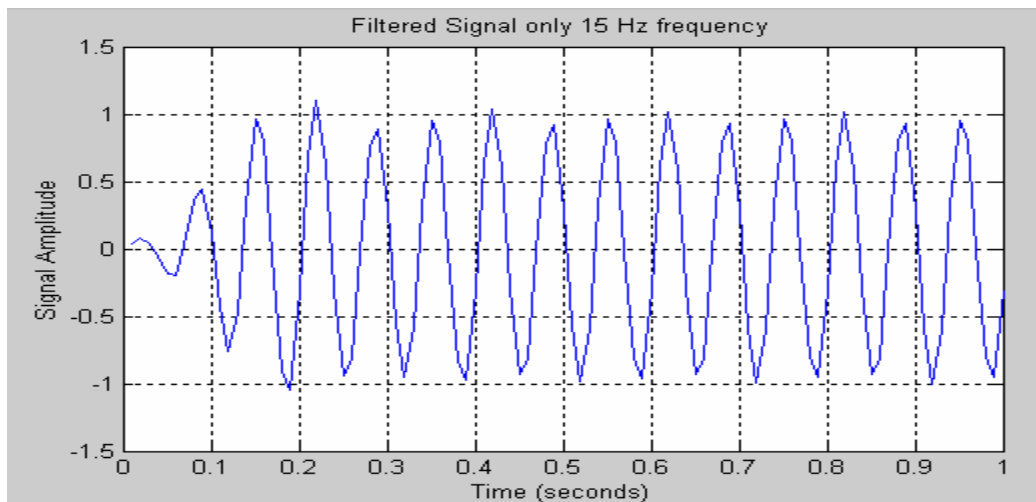
ellipord returns an order of 5, the minimum possible order for a low pass prototype that will meet the constraints upon transformation to a band pass filter. When we apply this order to the ellip function, internally we transform the low pass prototype to a band pass filter using the function lp2bp. This doubles the order, making $n = 10$. Next we use n , the order, and W_n , the pass band corner frequencies, to actually design the filter. We also use freqz, a tool for computing and displaying the frequency response of the descriptive transfer function. When called with no left-hand-side arguments (i.e., return values), freqz displays the magnitude and phase response of the filter normalized to the Nyquist frequency.

```
[b,a] = ellip(n,1,40,w);
freqz(b,a,128,100)
[h,w]=freqz(b,a,128,100);
plot(w,abs(h));
grid
title('Normalized Magnitude Response');
axis([0 50 0 1.2]);
```





```
figure(4)
sf=filter(b,a,s); % Time domain Response of the Filter
plot(t,sf)
grid
xlabel('Time (seconds)');
ylabel('Signal Amplitude');
title('Filtered Signal only 15 Hz frequency');
```



Problem:

Design an IIR filter to remove 100 and 150 frequencies from above signal.

$x=1+\sin (2*\pi*50*t) + \sin (2*\pi*100*t) + 0.5 \sin (2*\pi*125*t) + 0.25 \sin (2*\pi*150*t);$