EC 333 DIGITAL SIGNAL PROCESSING LAB (DSP)

INDEX

Sl. No.	Date	Experiment	Page no	Signature
1	6/9/20	INTRODUCTION TO MATLAB AND GENERATION OF WAVEFORMS	5	
2	13/9/20	PROOF OF NYQUIST SAMPLING THEOREM	22	
3	23/9/20	DISCRETE FOURIER TRANSFORM AND INVERSE DISCRETE FOURIER TRANSFORM	35	
4	14/10/20	LINEAR AND CIRCULAR CONVOLUTION	48	
5	18/10/20	OVERLAP ADD AND OVERLAP SAVE METHOD	56	
6	1/11/20	RADIX – 2 FFT AND IFFT	65	
7	8/11/20	FIR FILTER DESIGN	75	
8	7/12/20	IIR FILTER DESIGN	82	
9	6/12/20	FILTER DESIGN USING FDA TOOL	95	

EXPERIMENT NO: 1

INTRODUCTION TO MATLAB AND GENERATION OF WAVEFORMS

1. AIM

To generate elementary signals and verify the signal operations using MATLAB

2. THEORY

2.1. INTRODUCTION TO MATLAB

MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages. An additional package, Simulink, adds graphical multi-domain .

2.2. BASIC SIGNALS

a) **Sinusoidal Signal:** A sine wave is a geometric waveform that oscillates (moves up, down or side-to-side periodically) and is defined by the function y = sin x.

Sinusoidal signal is in the form of $x(t) = A\cos(w_0 \pm \phi w_0 \pm \phi)$ or $A\sin(w_0 \pm \phi w_0 \pm \phi)$

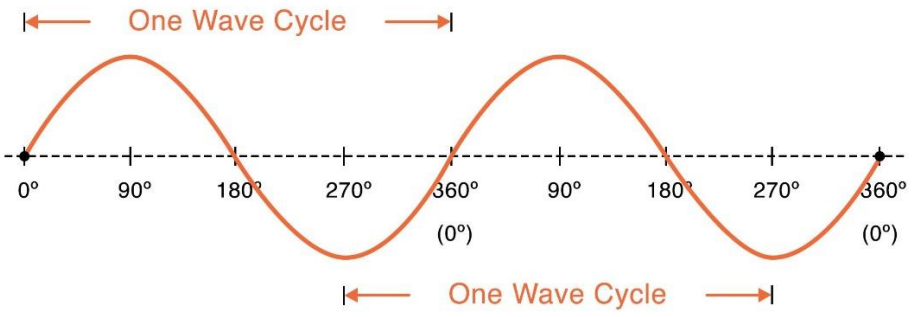


Fig.1.1 Sinusoidal Signal

b) **Cosine Signal:** A cosine function is a periodic signal mathematically represented by y=cos x.

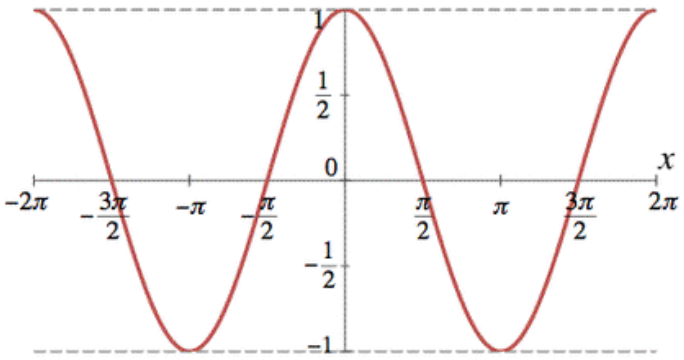


Fig.1.2 Cosine Signal

c) **Impulse Signal:** An impulse function is also known as delta function.

- $\delta(t)=0$ for $t\neq 0$.
- $\delta(t)=+\infty$ for $t=0$.
- $\int_{-\infty}^{\infty} \delta(t)dt = 1$.

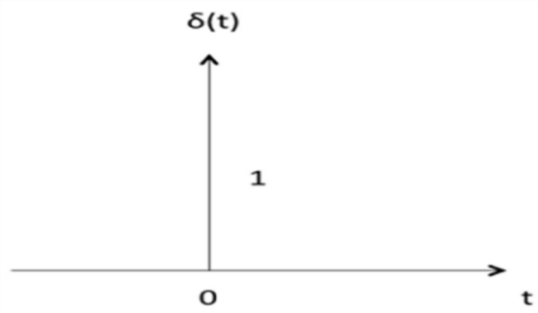


Fig.1.3 Impulse Signal

d) **Unit step function:** The unit step function is defined as: $u(t)=\begin{cases}0, if\ t<0\\1, if\ t\geq 0\end{cases}$

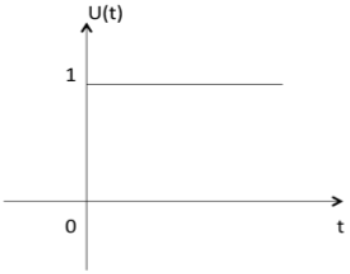


Fig.1.4 Unit step Function

e) **Ramp Function:** The ramp function R(x) may be defined as: $R(x)=\begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases}$

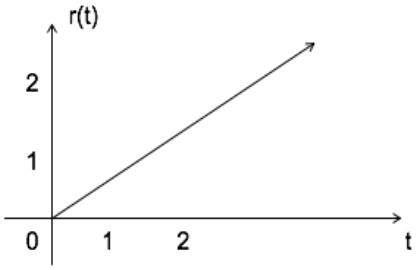


Fig.1.5 Ramp function

f) **Sinc function:** It is defined as: $\text{Sinc}(x)=\frac{\sin(\pi x)}{x}$ if $x \neq 0$ & sinc(0) =1.

A sinc function is an even function with unity area. A sinc pulse passes through zero at all positive and negative integers (i.e., t = ± 1 , ± 2 , ...), but at time , it reaches its maximum of 1.

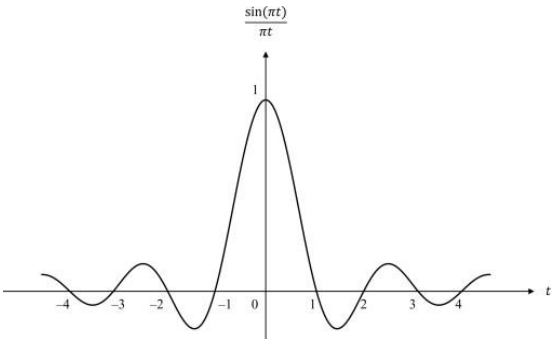


Fig.1.6 Sinc function

g) a^t for a>1, 0<a<1, a<0

If n is a positive integer and x is any real number, then x^n corresponds to repeated multiplication

$$x^n = x \times x \times \cdots \times x$$

We can call this "x=x raised to the power of n" .Here, x is the *base* and n is the *exponent* or the *power*.

- $x^1=x$
- $x^0=1$
- $x^{-1}=1/x$

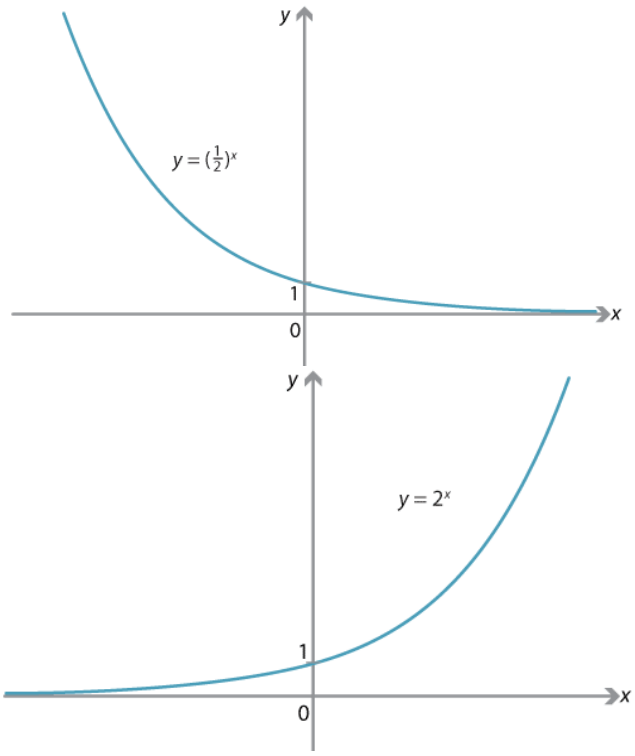
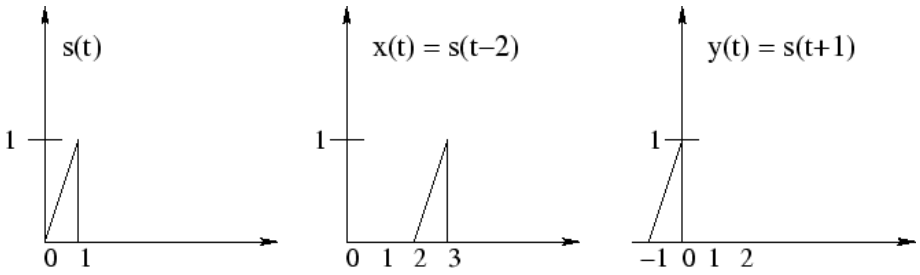


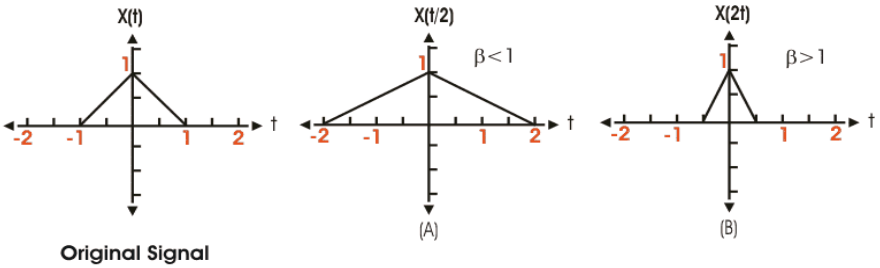
Fig.1.7 a^t for a<0 and a>0

2.3. BASIC OPERATIONS ON SIGNALS

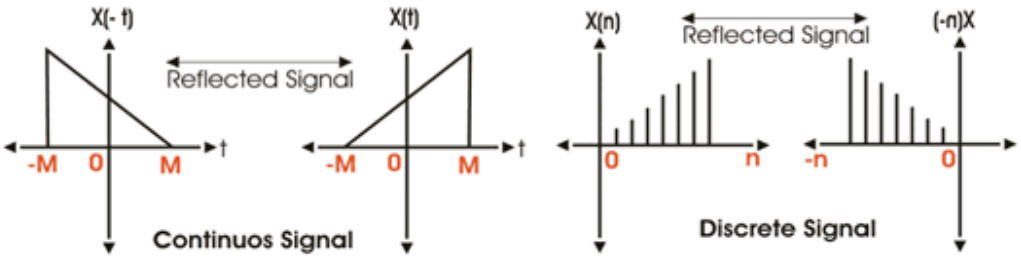
a) **Time shifting:** this property is used to shift signals to left or right. The general equation is x(t)→ $y(t+k)$ when k>0 it is left shift and k<0 it is right shift.



b) **Time scaling:** Time scaling compresses or dilates a signal by multiplying a time variable by some quantity. The general expression is x(t)→ $y(t)=x(\propto t)$.If $\propto>1$, the signal is compressed and if $\propto<1$ signal is dilated.



c) **Time reversal:** Whenever signal's time is multiplied by -1, it is known as time reversal of the signal. In this case, the signal produces its mirror image about Y-axis. Mathematically, this can be written as: x(t)→ $y(t) = x(-t)$



2.4. ENERGY OF A SIGNAL

In signal processing energy of a continuous time signal is defined as:

Es=$\int_{-\infty}^{\infty} |x(t)|^2$ dt

Energy of discrete signal is given by:

Es=$\sum_{n=-\infty}^{\infty} |x(n)|^2$

2.5. COMPLEX EXPONENTIAL

The complex exponential e^{jwt} is represented by Eulers's Formula as:

e^{jwt} =cos(wt)+jsin(wt)

e^(jwt) is definied from -infinity to +infinity, so any signals defined from -infinity to +infinity those signals are called eternal signals to find Fourier transform.

3. PROGRAM

2.2.

a) **Sinusoidal signal**

```
x=[-2*pi:0.1:2*pi];
y=sin(x);
subplot(2,1,1)
plot(x,y)
title("CT Sine Wave")
grid on
xlabel('-2pi<=x<=2pi')
```

```
ylabel("sin(x)")

x=[-2*pi:0.3:2*pi];
y=sin(x);
subplot(2,1,2)
stem(x,y)
title("DT Sine Wave")
grid on
xlabel("-2pi<=x<=2pi")
ylabel("sin(x)")
```

b) Cosine Wave

```
x=[-2*pi:0.1:2*pi];
cosine=cos(x);
subplot(2,1,1)
plot(x,cosine)
title("CT Cosine Wave")
grid on
xlabel("-2pi<=x<=2pi")
ylabel("cos(x)")

subplot(2,1,2)
x=[-2*pi:0.3:2*pi];
y=cos(x);
stem(x,y)
title("DT Cosine Wave")
grid on
xlabel("-2pi<=x<=2pi")
ylabel("cos(x)")
```

c) Unit Impulse Signal

```
x=-5:5;
y=(x==0);

subplot(2,1,1)
plot(x,y);
title("CT Impulse Signal")
grid on
xlabel('time');
ylabel('amplitude');
```

```
subplot(2,1,2)
stem(x,y);
title("DT Impulse Signal")
grid on
xlabel(' number of samples');
ylabel('amplitude');
```

d) Unit Step Signal

```
n=input('Enter n values');
t=-n:1:n;
y1= [zeros(1, n),ones(1, n+1)];

subplot(2,1,1);
plot(t,y1);
title("CT Unit step signal")
ylabel('Amplitude');
xlabel('time');


subplot(2,1,2);
stem(t,y1);
title("DT Unit step signal")
ylabel('Amplitude');
xlabel('number of samples');
```

e) Ramp Signal

```
n=input('Enter n values');
t=-n-1:1:n-1;


subplot(2,1,1);
plot(t,t);
title("CT Ramp Signal")
ylabel('Amplitude');
xlabel('x');


subplot(2,1,2);
stem(t,t);
title("DT Ramp signal")
ylabel('Amplitude');
xlabel('number of samples');
```

f) sinc Function

```
x=linspace(-8,8,100);
y=sinc(x);

subplot(2,1,1);
plot(x,y);
axis([-8 8 -1.1 1.1]);
xlabel('time');
ylabel('amplitude');
title('sinc wave');

subplot(2,1,2);
stem(x,y);
axis([-8 8 -1.1 1.1]);
xlabel('number of samples');
ylabel('amplitude');
title('discrete sinc wave');
```

g) Exponential signal a^t

```
t=0:1:30;
a=input('Enter a value');
y2=a.^t;


subplot(2,1,1);
plot(t,y2);
title("CT exponential signal")
xlabel('n');
ylabel('Amplitude');


subplot(2,1,2);
stem(t,y2);
title("DT exponential signal")
xlabel('n');
ylabel('Amplitude');
```

2.3.

a) Time Shifting

```
n=-3:3;
x=[1 2 3 4 7 6 5 ];
shft=input("Enter the amount to be shifted");

subplot(2,1,1)
stem(n,x)
title("The original signal x[n]")
xlabel("n")
ylabel("Amplitude")

subplot(2,1,2)
stem(n+(-shft),x)
title("The shifted signal ")
xlabel("n")
ylabel("Amplitude")
```

b) Time Scaling

```
n=-3:3;
x=[1 2 3 4 7 6 5 ];
scale=input("enter the amount to be scaled");

subplot(2,1,1)
stem(n,x)
title("The original signal x[n]")
xlabel("n")
ylabel("Amplitude")

subplot(2,1,2)
stem(n/scale,x)
title("Time scaled signal")
xlabel("n")
ylabel("Amplitude")
```

c) Time Reversal

```
n=-3:3;
x=[1 2 3 4 7 6 5 ];

subplot(2,1,1)
stem(n,x)
```

```
title("The original signal x[n]")
xlabel("n")
ylabel("Amplitude")

subplot(2,1,2)
stem(-n,x)
title("The reversed signal x[-n]")
xlabel("n")
ylabel("Amplitude")
```

2.4. Energy of given signal x=[1 2 3 4]

```
 n=0:1:3;
x=[1 2 3 4];
stem(n,x);
title('Given Signal x[n]')
xlabel('n')
ylabel('x[n]')
for n=0:1:3
    Energy=sum((abs(x)).^2);
end
disp('Energy of x[n] is:')
disp(Energy)
```

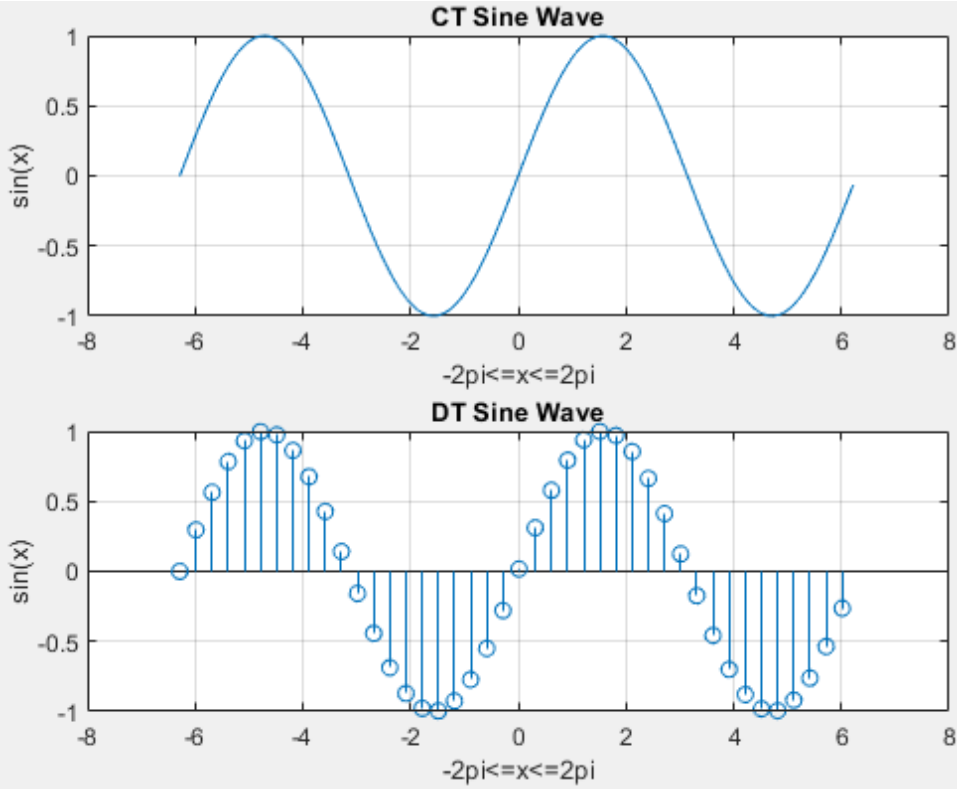
2.5. Complex Exponential e^{jwt}

```
t=(0:0.01:20);
a=input('Enter value of w');
signal=exp(1i*a*t);
rp=real(signal);
ip=imag(signal);
plot3(t,rp,ip);
title("e^j^w^t plot")
xlabel("time(t)")
ylabel("real part")
zlabel(" imaginary part")
```

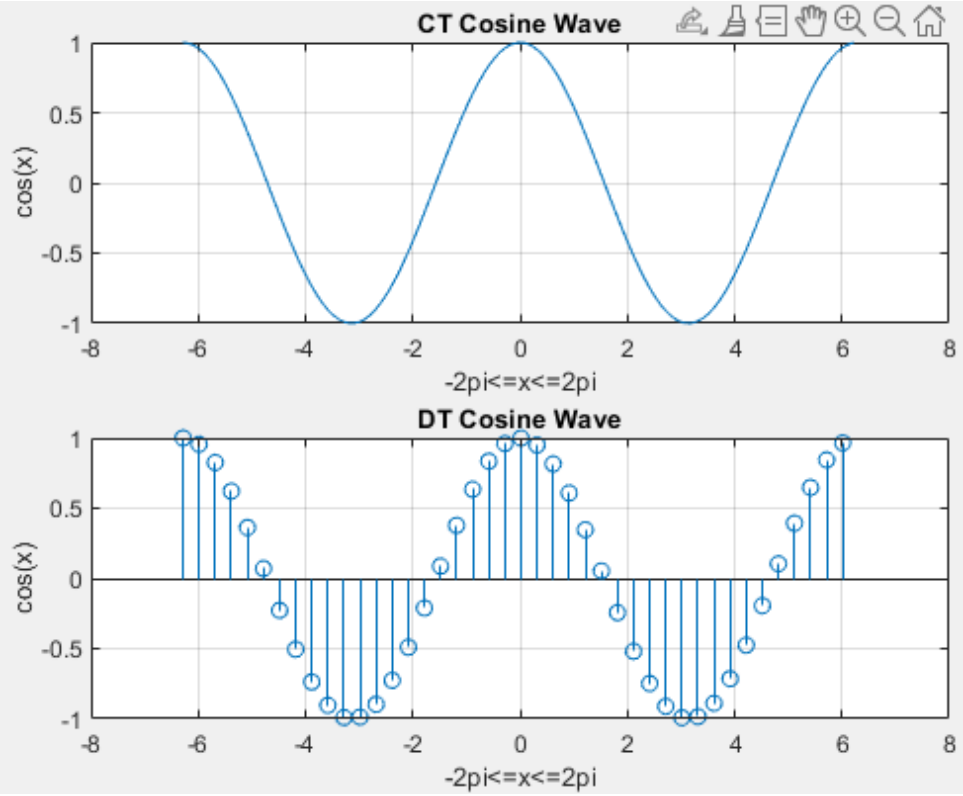
4. RESULT

2.2.

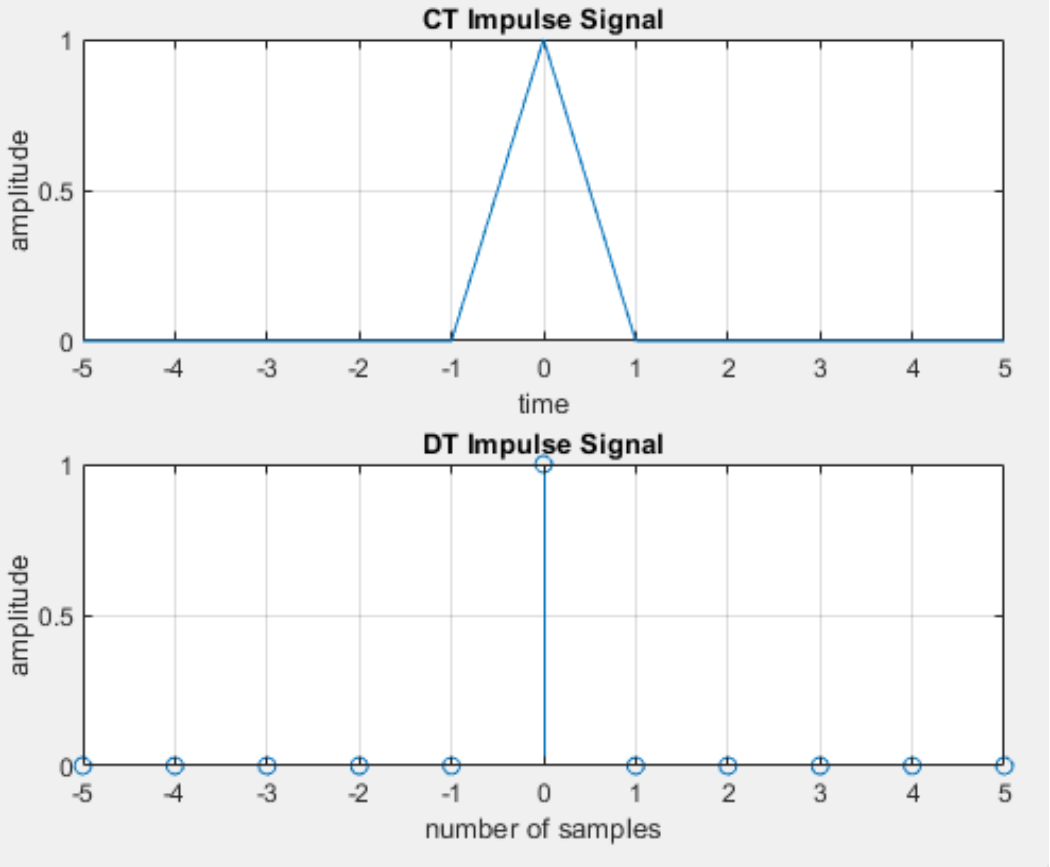
a) Sinusoidal Signal



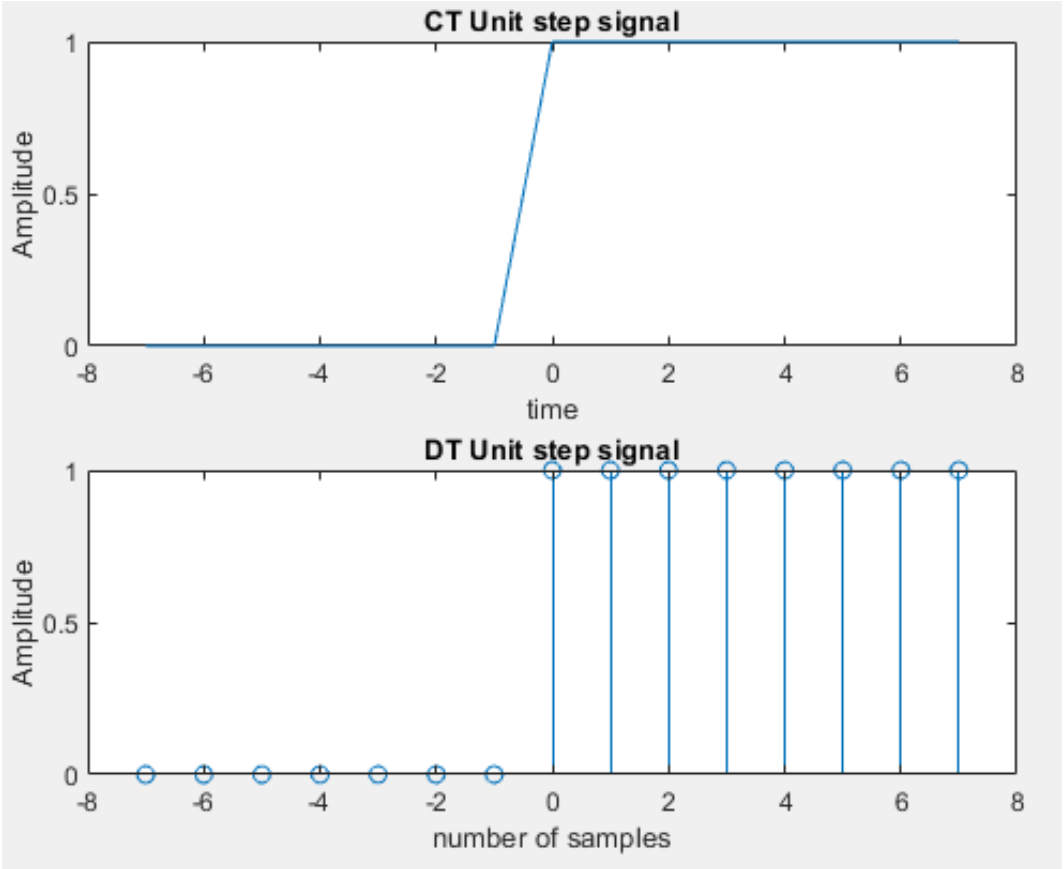
b) Cosine Signal



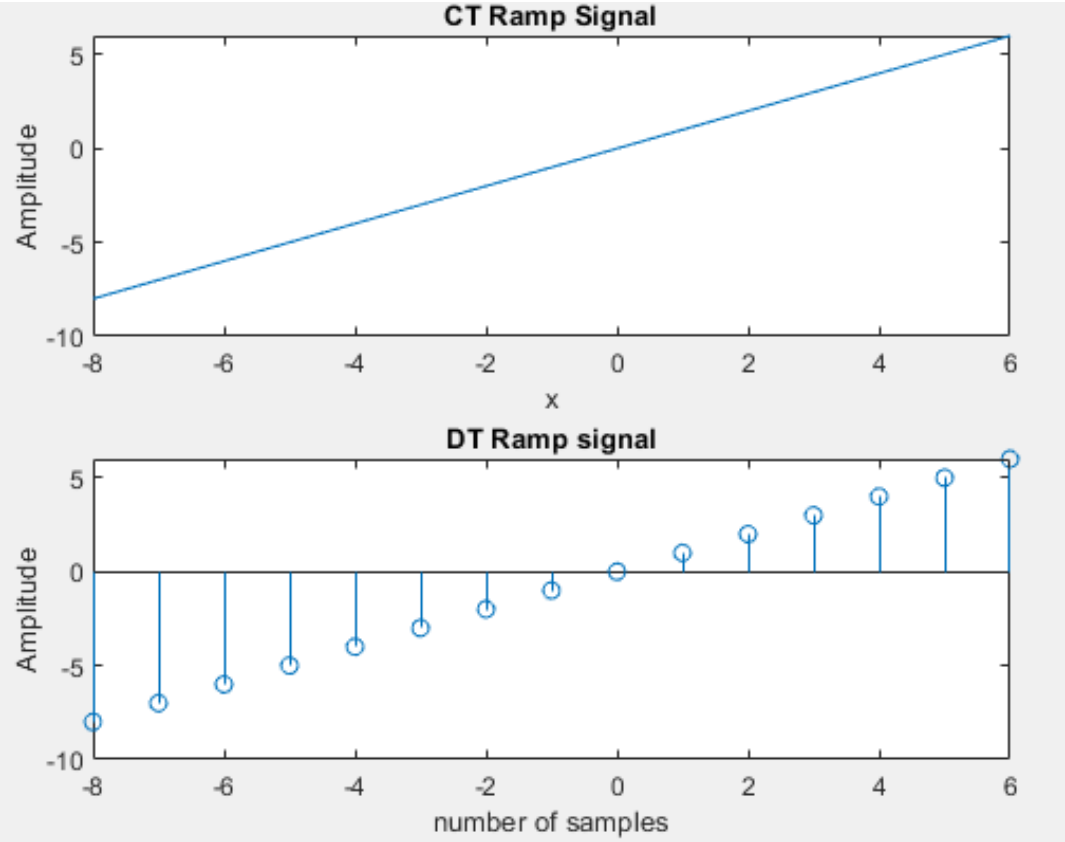
c) Unit Impulse Signal



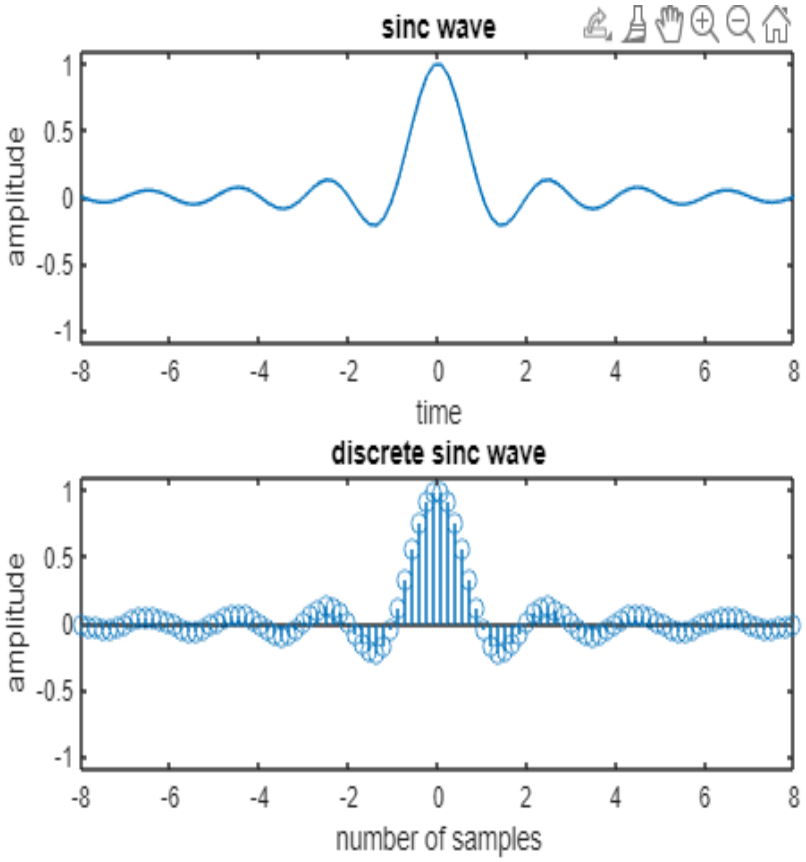
d) Unit step function



e) Ramp function

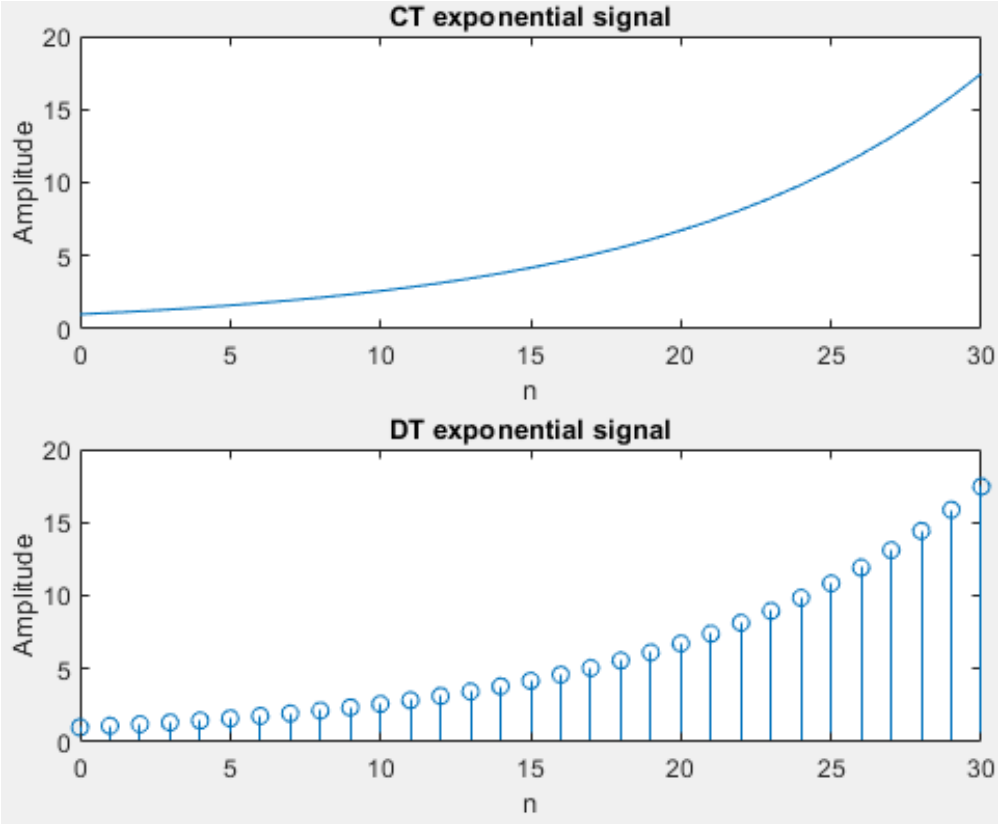


f) Sinc function

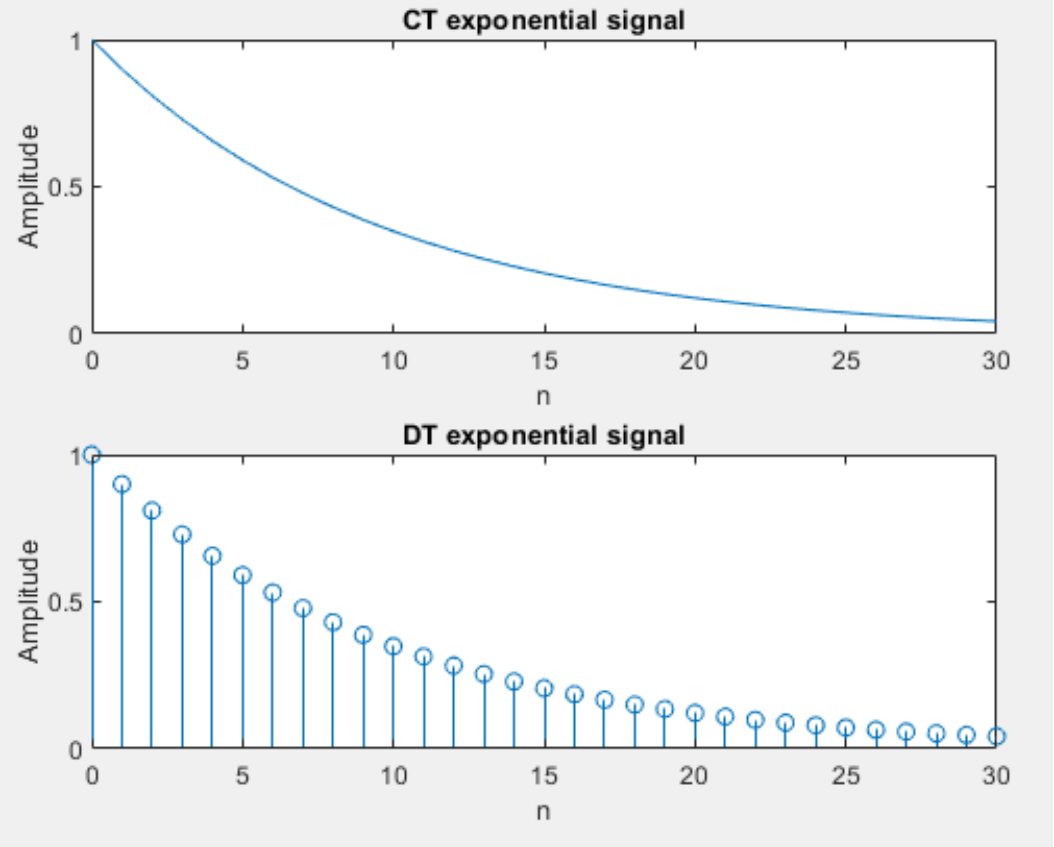


g) Exponential signal a^t for a>1, 0<a<1, a<0

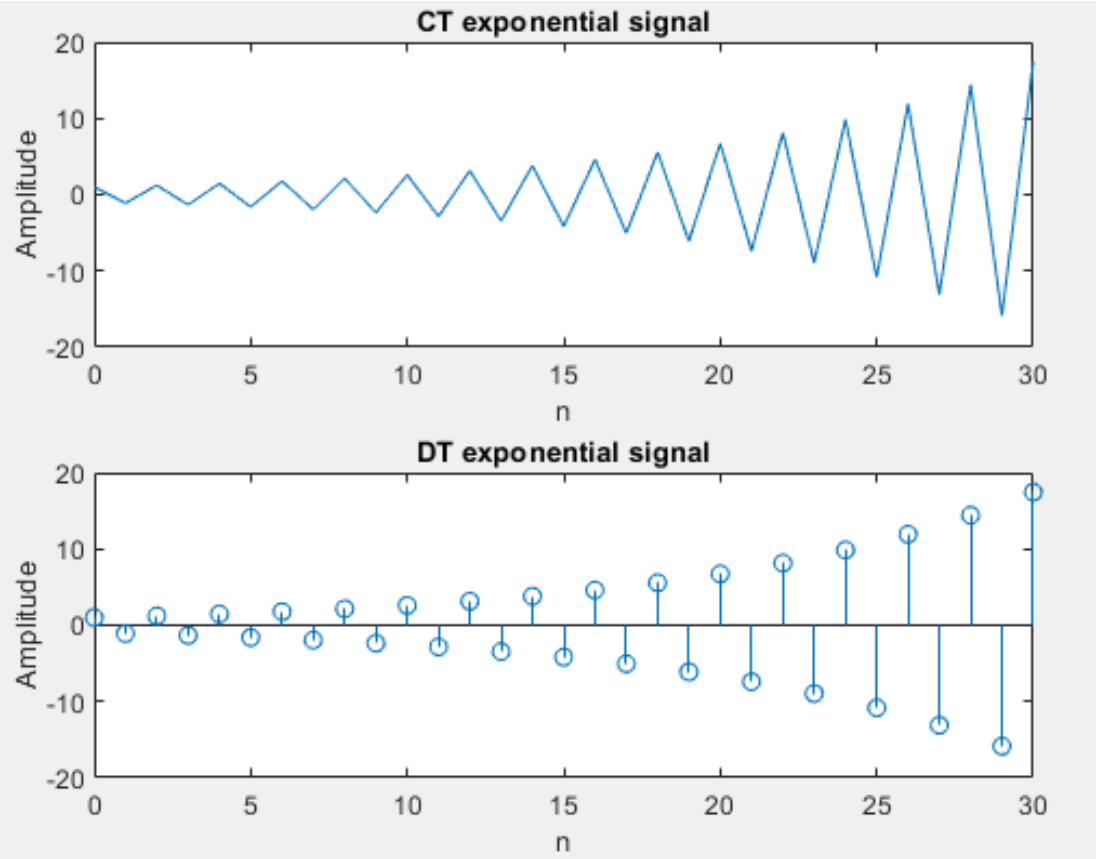
a>1(here a= 1.1)



0<a<1 (here a=0.9)

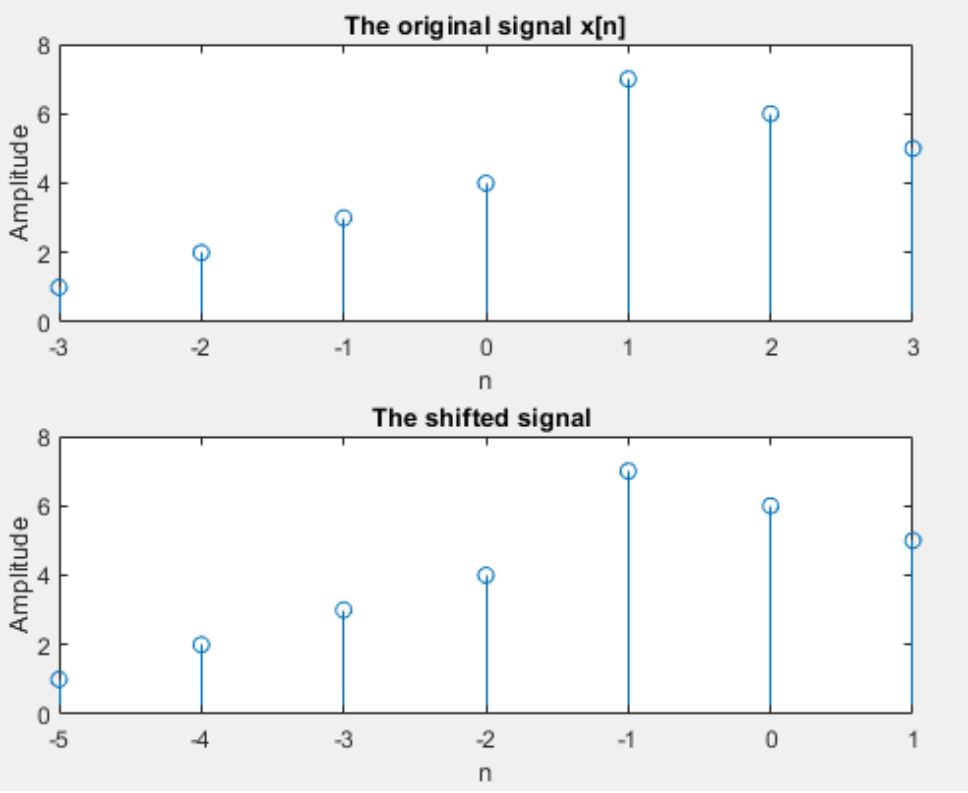


a<0(here a=-1)

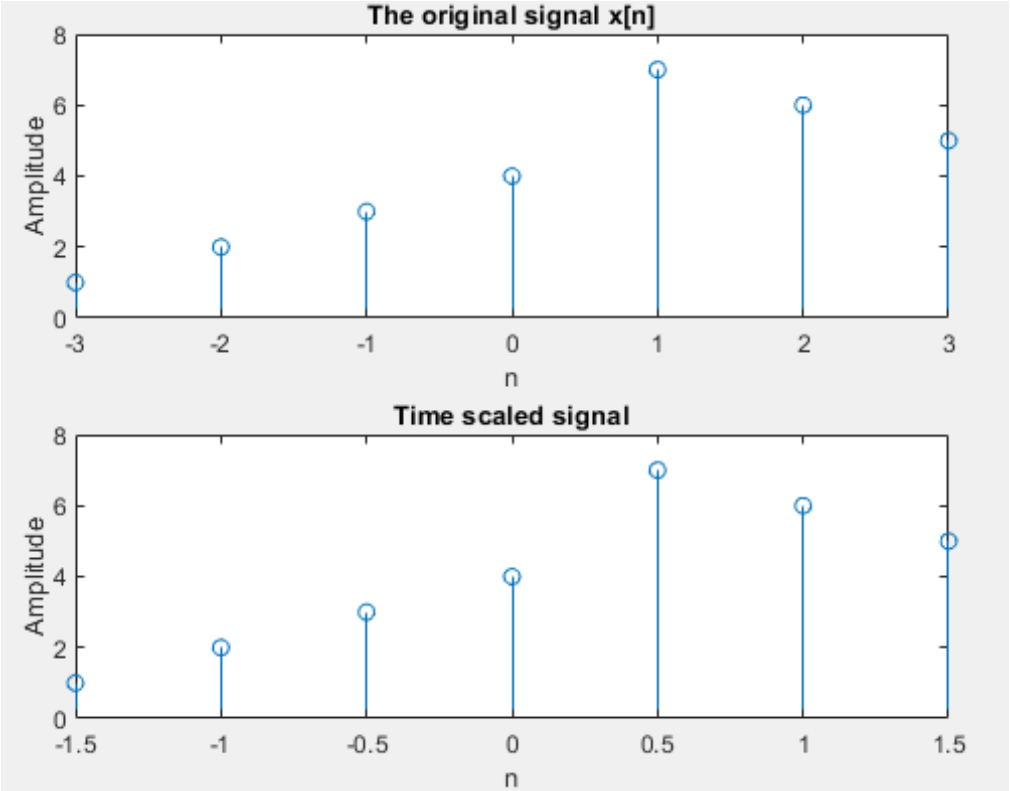


2.3.

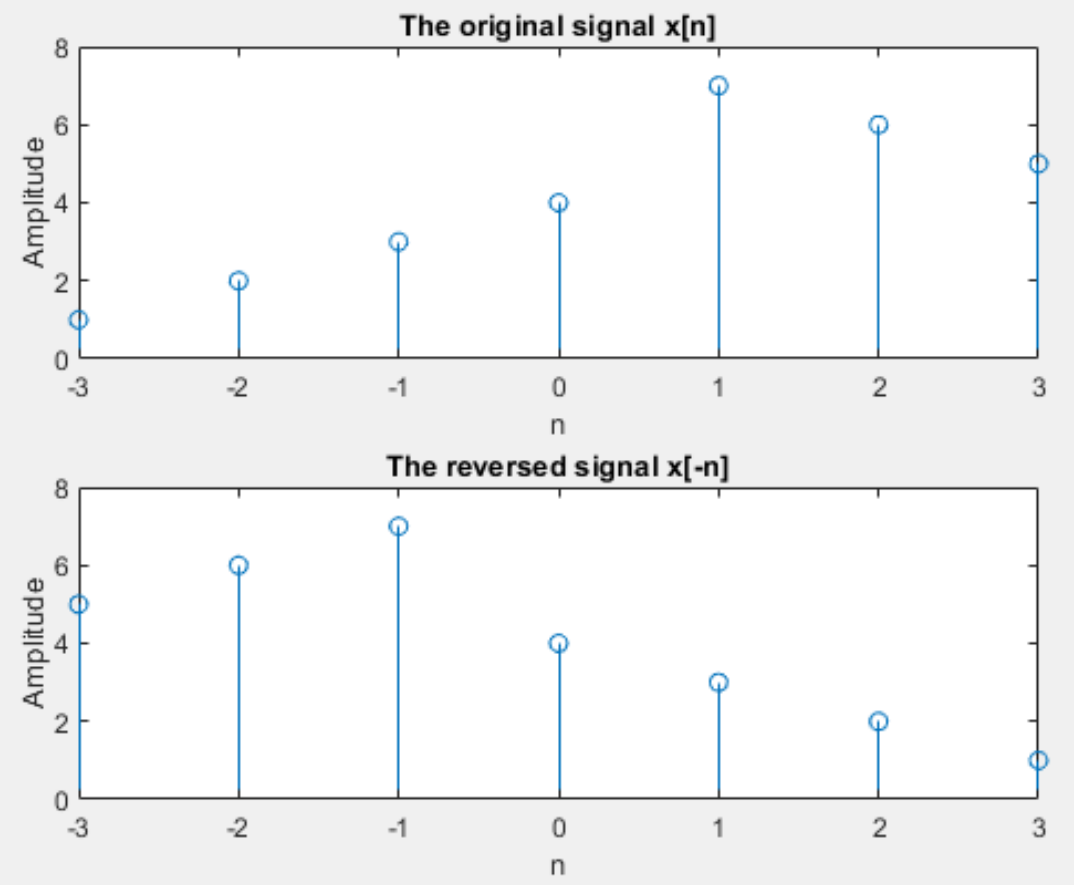
a) TIME SHIFTED



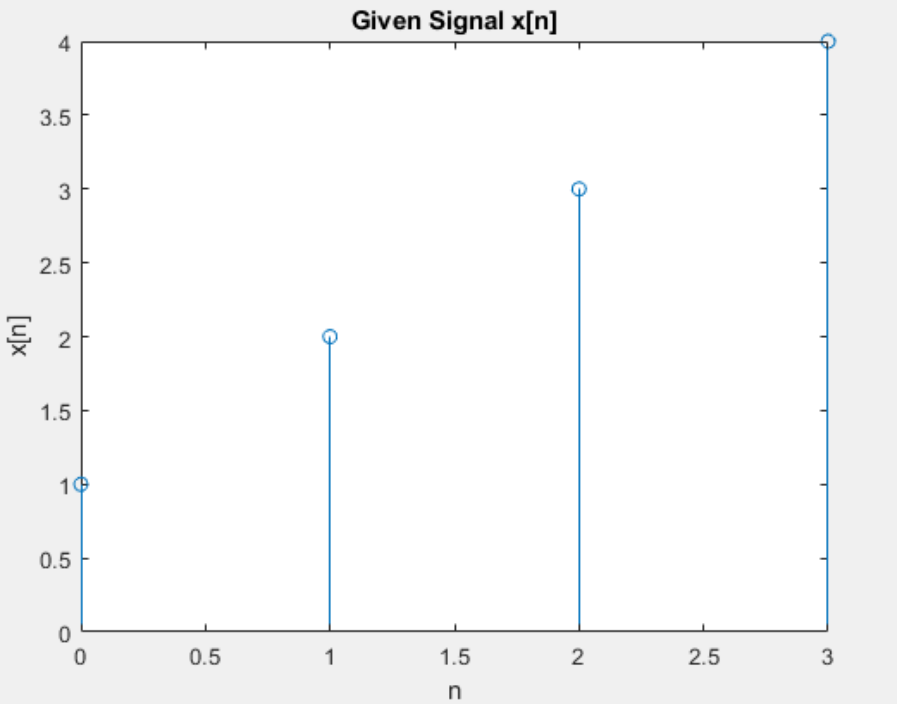
TIME SCALING



TIME REVERSAL

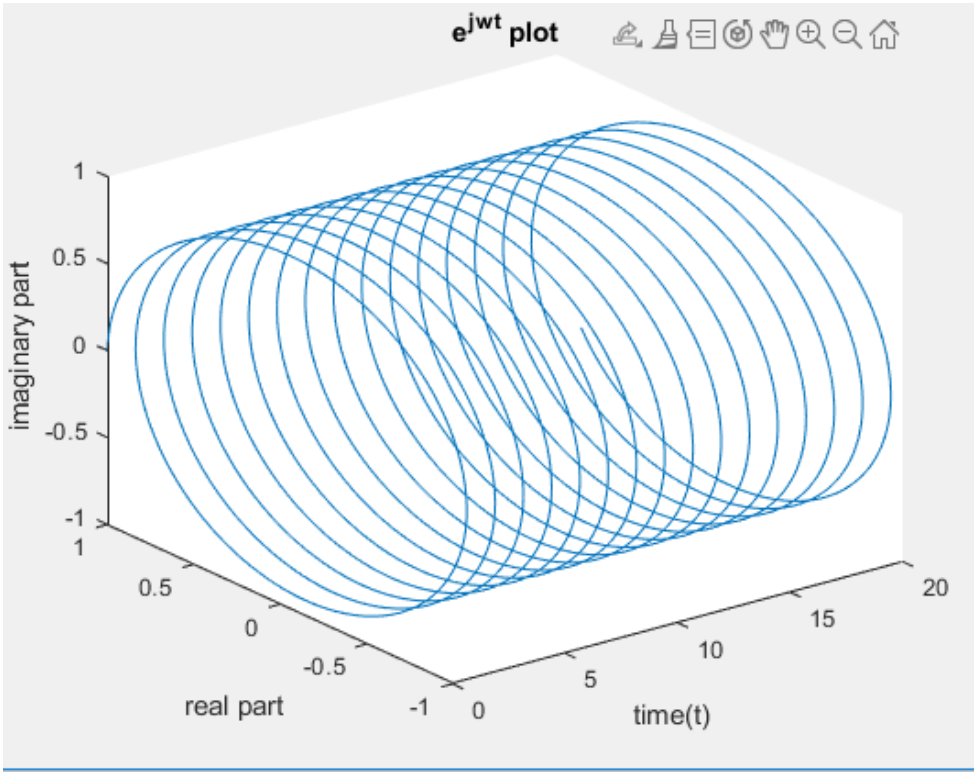


2.4 Energy of x[n]



OBTAINED ENERGY: 30

2.6. Complex Exponential



Plotting and generation of waveforms for given functions are studied using MATLAB. Operation on signals and its functionality is studied using MATLAB functions.

EXPERIMENT NO: 2

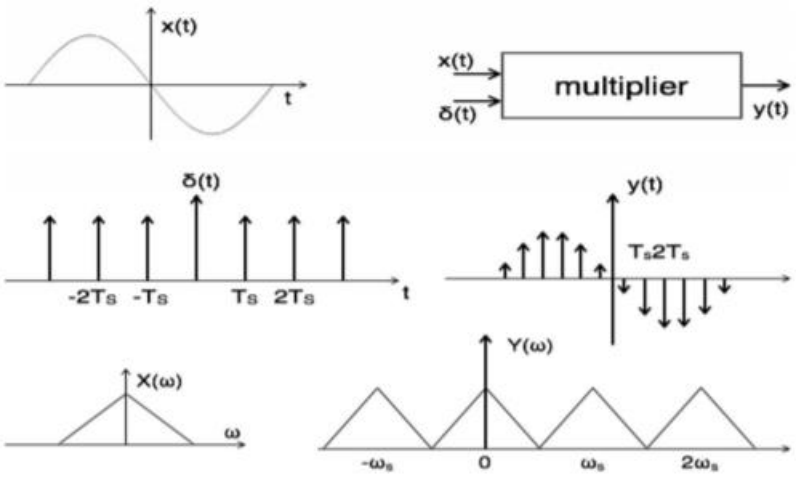
VERIFICATION OF SAMPLING THEOREM

1. AIM

To verify sampling theorem using MATLAB

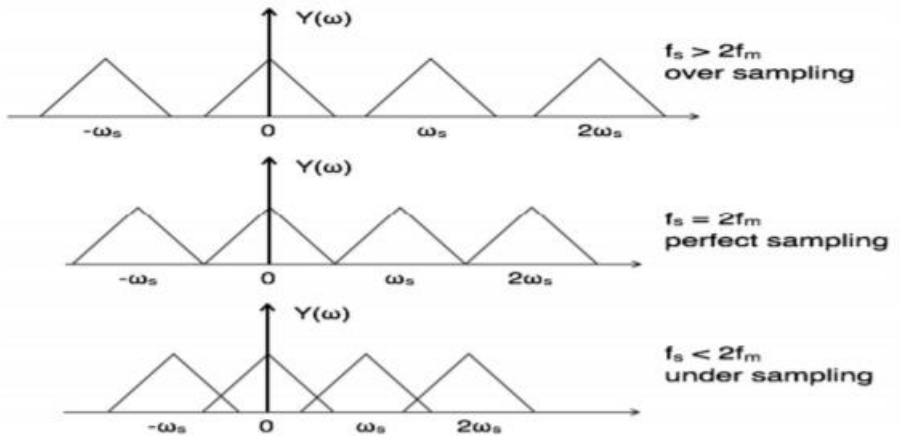
2. THEORY

Statement: A continuous time signal can be represented in its samples and can be recovered back when sampling frequency fs is greater than or equal to the twice the highest frequency component of message signal. i. e. fs≥2fm.
Consider a continuous time signal x(t). The spectrum of x(t) is a band limited to fm Hz i.e. the spectrum of x(t) is zero for $|\omega|>\omega m$. Sampling of input signal x(t) can be obtained by multiplying x(t) with an impulse train δ(t) of period Ts. The output of multiplier is a discrete signal called sampled signal which is represented with y(t) in the following diagrams:



The process of converting a signal from continuous time to discrete time is called sampling. The value of the signal is measured at certain intervals in time. Each measurement is referred to as a sample to preserve the full information in the signal, it is necessary to sample at twice the maximum frequency of the signal. This is known as the Nyquist rate.
The Sampling Theorem states that a signal can be exactly reproduced if it is sampled at a frequency fs, where fs is greater than twice the maximum frequency in the signal(fm). fs ≥2fm
If the sampling frequency is lower that the Nyquist rate, when the signal is converted back into a continuous time signal, it will exhibit a phenomenon called aliasing. Aliasing is the presence of unwanted components in the reconstructed signal. These components were not present when the original signal was sampled. In addition, some of the frequencies in the original signal may be lost in the

reconstructed signal. Aliasing occurs because signal frequencies can overlap if the sampling frequency is too low. Frequencies "fold" around half the sampling frequency – which is why this frequency is often referred to as the folding frequency.
To prevent aliasing of these frequencies, we can filter out these components before sampling the signal. Because we are filtering out high frequency components and letting lower frequency components through, this is known as low-pass filtering.
There is other phenomenon related to sampling that is oversampling. Oversampling is the process of sampling a signal with a sampling frequency significantly higher than the Nyquist rate. Theoretically a bandwidth-limited signal can be perfectly reconstructed if sampled above the Nyquist rate, which is twice the highest frequency in the signal. Oversampling improves resolution, reduces noise and helps avoid aliasing and phase distortion by relaxing anti-aliasing filter performance requirements.



3. PROGRAM

a) Sin($2\pi ft$)

```
fm=input("enter maximum signal frequency:\n");
fs2=2*fm
fs1=input("enter sampling frequency<fs2:\n");
fs3=input("enter sampling frequency>fs2:\n");

t=0:0.001:1;
t1=0:1/fs1:1;
t2=0:1/fs2:1;
t3=0:1/fs3:1;

s=sin(2*pi*t*fm);
s1=sin(2*pi*fm*t1);
s2=sin(2*pi*fm*t2);
s3=sin(2*pi*fm*t3);
```

```
subplot(4,2,1)
plot(t,s)
title('Continuous sinusoidal signal');
xlabel('t');
ylabel('x(t)');

subplot(4,2,2)
stem(t1,s1)
title('fs<2fm:discrete');
xlabel('n');
ylabel('x(n)');

subplot(4,2,3)
plot(t1,s1)
title('fs<2fm:continuous');
xlabel('t');
ylabel('x(t)');

subplot(4,2,4)
stem(t2,s2)
title('fs=2fm:discrete');
xlabel('n');
ylabel('x(n)');

subplot(4,2,5)
plot(t2,s2)
title('fs=2fm:continuous');
xlabel('t');
ylabel('x(t)');

subplot(4,2,6)
stem(t3,s3)
title('fs>2fm:discrete');
xlabel('n');
ylabel('x(n)');

subplot(4,2,7)
plot(t3,s3)
title('fs>2fm:continuous');
xlabel('t');
ylabel('x(t)');
```

b) Cos($2\pi ft$)

```
fm=input("enter maximum signal frequency:\n");
fs2=2*fm
fs1=input("enter sampling frequency<fs2:\n");
fs3=input("enter sampling frequency>fs2:\n");

t=0:0.001:1;
t1=0:1/fs1:1;
t2=0:1/fs2:1;
t3=0:1/fs3:1;

s=cos(2*pi*t*fm);
s1=cos(2*pi*fm*t1);
s2=cos(2*pi*fm*t2);
s3=cos(2*pi*fm*t3);


subplot(4,2,1)
plot(t,s)
title('Continuous cosine signal');
xlabel('t');
ylabel('x(t)');

subplot(4,2,2)
stem(t1,s1)
title('fs<2fm:discrete');
xlabel('n');
ylabel('x(n)');

subplot(4,2,3)
plot(t1,s1)
title('fs<2fm:continuous');
xlabel('t');
ylabel('x(t)');

subplot(4,2,4)
stem(t2,s2)
title('fs=2fm:discrete');
xlabel('n');
ylabel('x(n)');

subplot(4,2,5)
plot(t2,s2)
```

```
title('fs=2fm:continuous');
xlabel('t');
ylabel('x(t)');

subplot(4,2,6)
stem(t3,s3)
title('fs>2fm:discrete');
xlabel('n');
ylabel('x(n)');
subplot(4,2,7)
plot(t3,s3)
title('fs>2fm:continuous');
xlabel('t');
ylabel('x(t)');
```

c) **$\mathbf{Sin(2\pi f1t)+ Cos(2\pi f2t)}$**

```
f1=input("enter value of f1 in(sin(2pif1t)+cos(2pif2t)):\n");
f2=input("enter value of f2 in(sin(2pif1t)+cos(2pif2t)):\n");

fm=max(f)
fs2=2*fm
fs1=input("enter sampling frequency<fs2:\n");
fs3=input("enter sampling frequency>fs2:\n");

t=0:0.001:1;
t1=0:1/fs1:1;
t2=0:1/fs2:1;
t3=0:1/fs3:1;

s=sin(2*pi*t*f1)+cos(2*pi*t*f2);
s1=sin(2*pi*f1*t1)+cos(2*pi*t1*f2);
s2=sin(2*pi*f1*t2)+cos(2*pi*t2*f2);
s3=sin(2*pi*f1*t3)+cos(2*pi*t3*f2);

subplot(4,2,1)
plot(t,s)
title('sin(2pift)+cos(2pift):continuous');
xlabel('t');
ylabel('x(t)');

subplot(4,2,2)
stem(t1,s1)
title('fs<2fm:discrete');
```

```
xlabel('n');
ylabel('x(n)');

subplot(4,2,3)
plot(t1,s1)
title('fs<2fm:continuous');
xlabel('t');
ylabel('x(t)');

subplot(4,2,4)
stem(t2,s2)
title('fs=2fm:discrete');
xlabel('n');
ylabel('x(n)');

subplot(4,2,5)
plot(t2,s2)
title('fs=2fm:continuous');
xlabel('t');
ylabel('x(t)');

subplot(4,2,6)
stem(t3,s3)
title('fs>2fm:discrete');
xlabel('n');
ylabel('x(n)');

subplot(4,2,7)
plot(t3,s3)
title('fs>2fm:continuous');
xlabel('t');
ylabel('x(t)');
```

d) **Sin ($2\pi f1t$). Cos($2\pi f2t$)**

```
f1=input("enter value of f1 in (sin(2pif1t)xcos(2pif2t)):\n");
f2=input("enter value of f2 in (sin(2pif1t)xcos(2pif2t)):\n");
fm=f1+f2;

fs2=2*fm
fs1=input("enter sampling frequency<fs2:\n");
fs3=input("enter sampling frequency>fs2:\n");

t=0:0.001:1;
```

```
t1=0:1/fs1:1;
t2=0:1/fs2:1;
t3=0:1/fs3:1;

s=sin(2*pi*t*f(1)).*cos(2*pi*t*f(2));
s1=sin(2*pi*f(1)*t1).*cos(2*pi*t1*f(2));
s2=sin(2*pi*f(1)*t2).*cos(2*pi*t2*f(2));
s3=sin(2*pi*f(1)*t3).*cos(2*pi*t3*f(2));

subplot(4,2,1)
plot(t,s)
title('sin(2pift)*cos(2pift):continuous');
xlabel('t');
ylabel('x(t)');

subplot(4,2,2)
stem(t1,s1)
title('fs<2fm:discrete');
xlabel('n');
ylabel('x(n)');

subplot(4,2,3)
plot(t1,s1)
title('fs<2fm:continuous');
xlabel('t');
ylabel('x(t)');

subplot(4,2,4)
stem(t2,s2)
title('fs=2fm:discrete');
xlabel('n');
ylabel('x(n)');

subplot(4,2,5)
plot(t2,s2)
title('fs=2fm:continuous');
xlabel('t');
ylabel('x(t)');

subplot(4,2,6)
stem(t3,s3)
title('fs>2fm:discrete');
xlabel('n');
ylabel('x(n)');
```

```
subplot(4,2,7)
plot(t3,s3)
title('fs>2fm:continuous');
xlabel('t');
ylabel('x(t)');
```

e) **Sinc($2\pi ft$)**

```
fm=input("enter maximum signal frequency:\n");
fs2=2*fm
fs1=input("enter sampling frequency<fs2:\n");
fs3=input("enter sampling frequency>fs2:\n");

t=-1:0.001:1;
t1=-1:1/fs1:1;
t2=-1:1/fs2:1;
t3=-1:1/fs3:1;

s=sinc(2*pi*t*fm);
s1=sinc(2*pi*fm*t1);
s2=sinc(2*pi*fm*t2);
s3=sinc(2*pi*fm*t3);

subplot(4,2,1)
plot(t,s)
title('Continuous sinc signal');
xlabel('t');
ylabel('x(t)');

subplot(4,2,2)
stem(t1,s1)
title('fs<2fm:discrete');
xlabel('n');
ylabel('x(n)');

subplot(4,2,3)
plot(t1,s1)
title('fs<2fm:continuous');
xlabel('t');
ylabel('x(t)');

subplot(4,2,4)
stem(t2,s2)
title('fs=2fm:discrete');
```

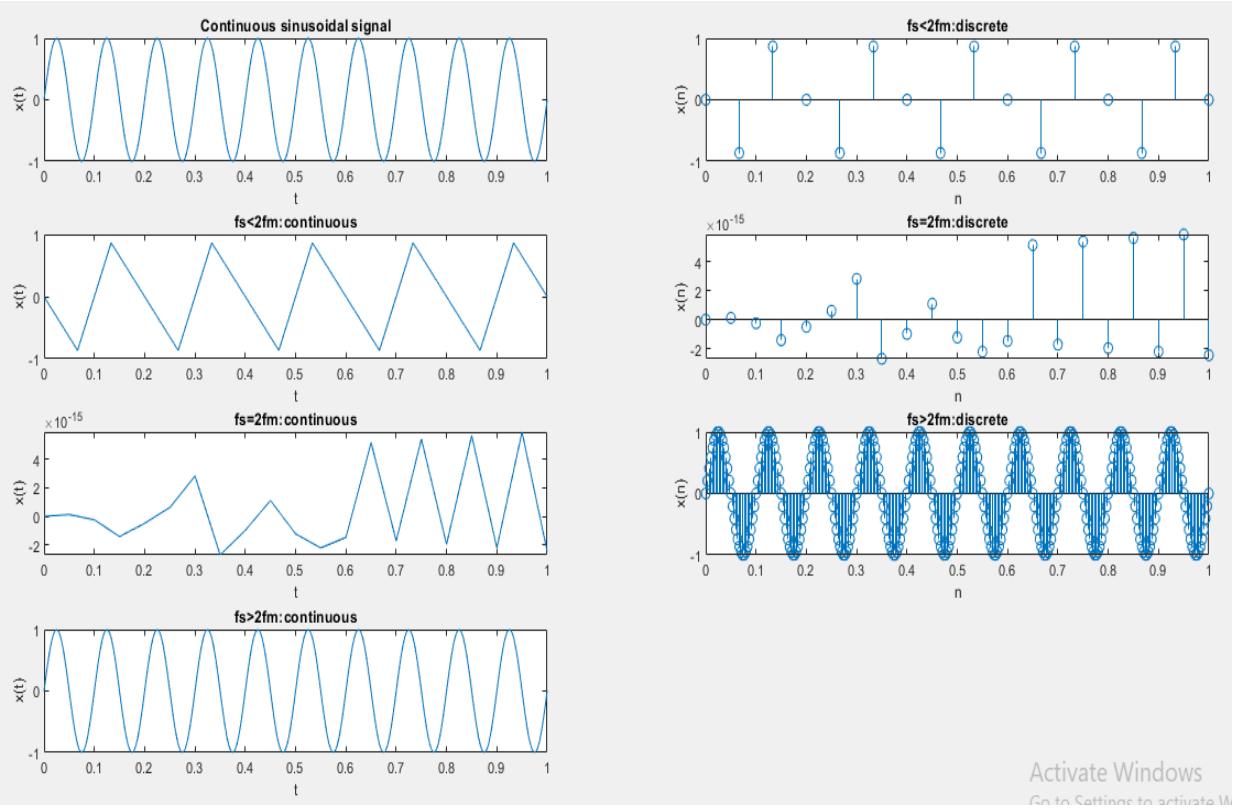
```
xlabel('n');
ylabel('x(n)');

subplot(4,2,5)
plot(t2,s2)
title('fs=2fm:continuous');
xlabel('t');
ylabel('x(t)');

subplot(4,2,6)
stem(t3,s3)
title('fs>2fm:discrete');
xlabel('n');
ylabel('x(n)');

subplot(4,2,7)
plot(t3,s3)
title('fs>2fm:continuous');
xlabel('t');
ylabel('x(t)');
```

4. RESULT

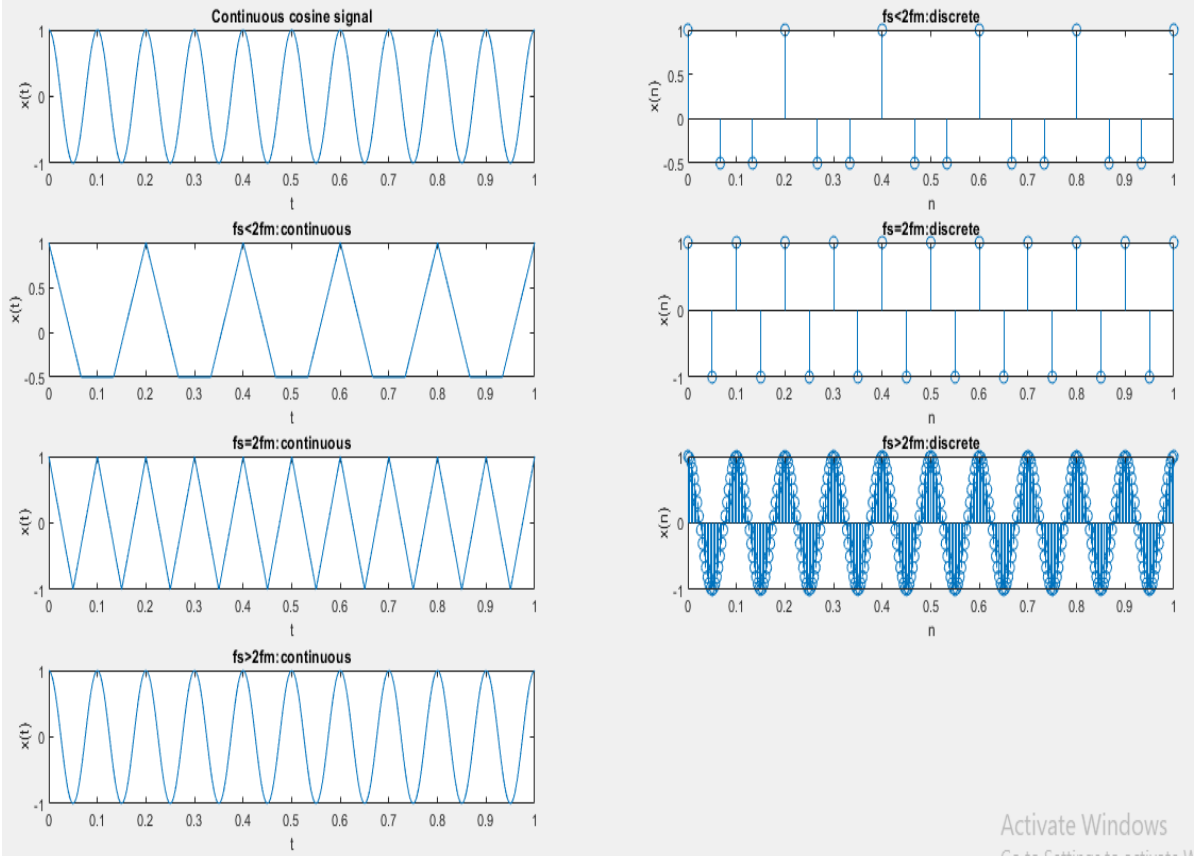
INPUT:

```
enter maximum signal frequency:
10

fs2 =

    20

enter sampling frequency<fs2:
15
enter sampling frequency>fs2:
300
```

a) Sin($2\pi ft$)

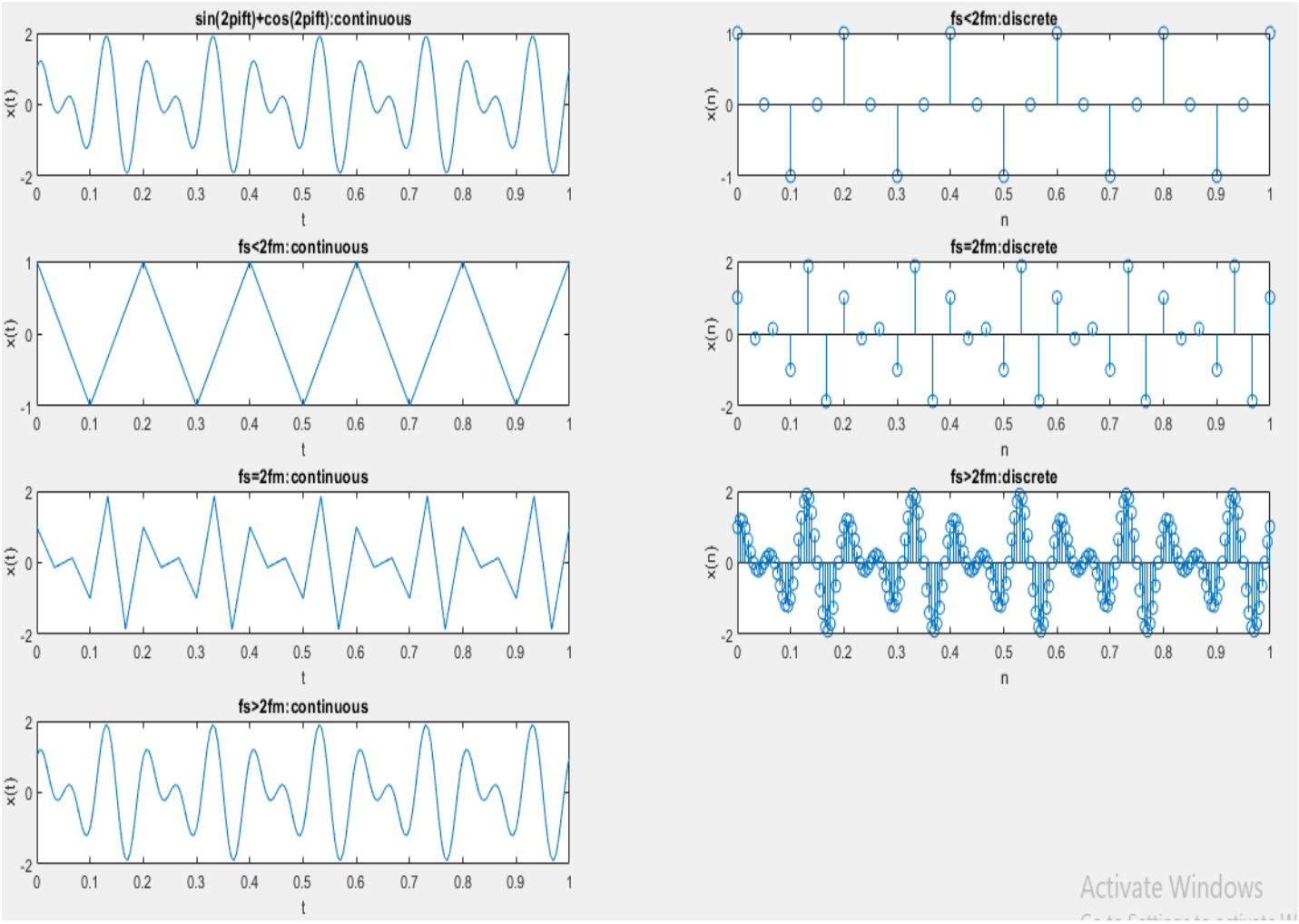


b) Cos($2\pi ft$)



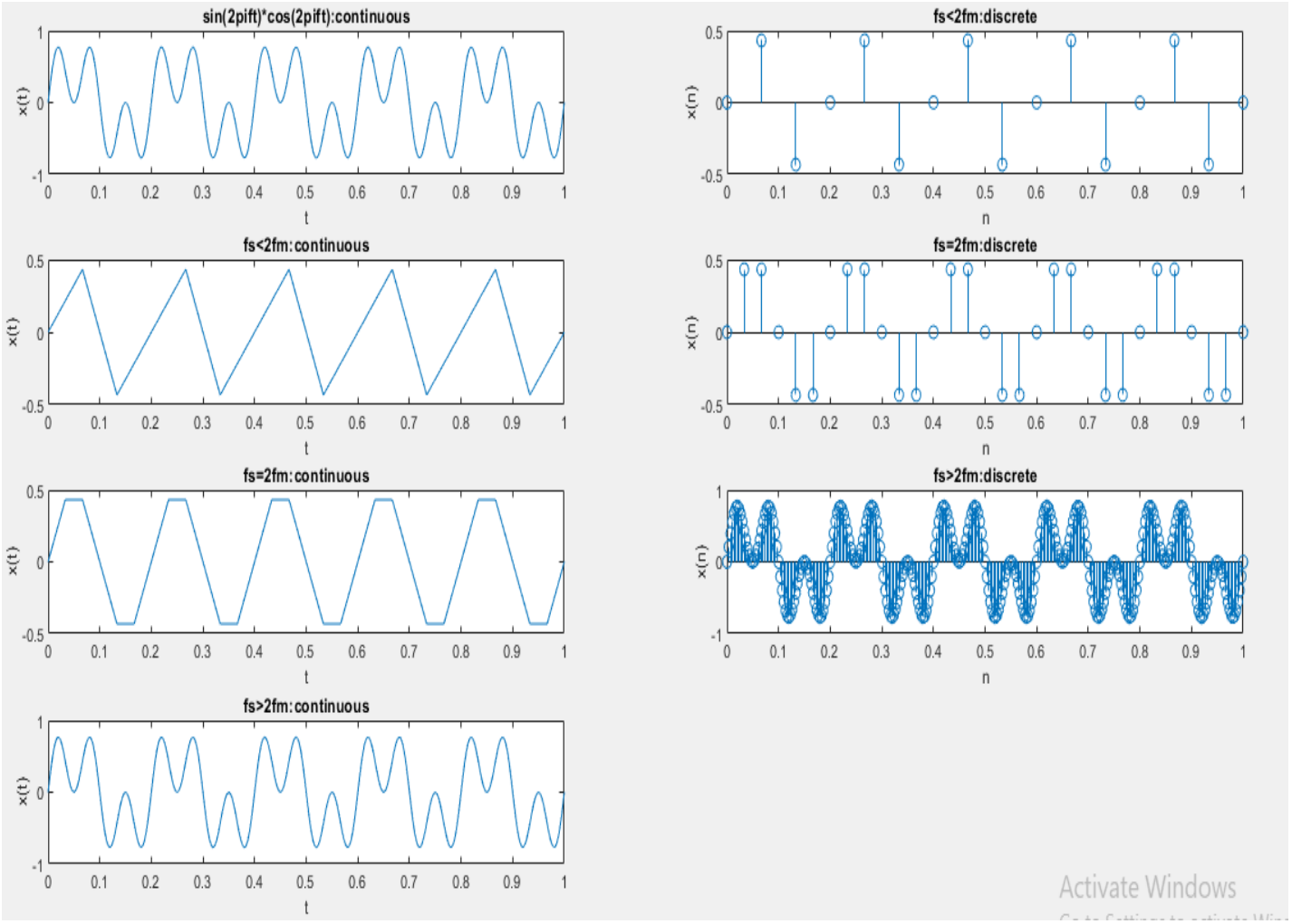
c) Sin($2\pi f1t$)+ Cos($2\pi f2t$)

```
enter value of f1 in(sin(2pif1t)+cos(2pif2t)):
10
enter value of f2 in(sin(2pif1t)+cos(2pif2t)):
15

fm =

    15


fs2 =

    30

enter sampling frequency<fs2:
20
enter sampling frequency>fs2:
200
```



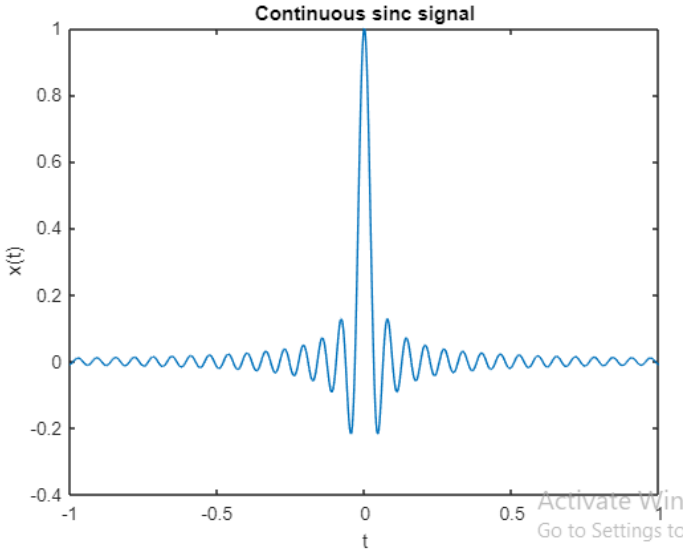
d) Sin ($2\pi f1t$). Cos($2\pi f2t$)

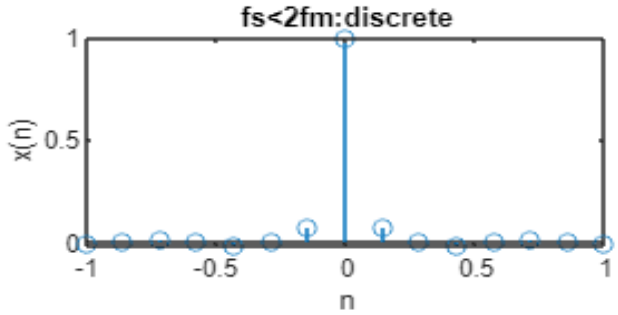
```
enter value of f1 in (sin(2pif1t)xcos(2pif2t)):
10
enter value of f2 in (sin(2pif1t)xcos(2pif2t)):
5

fs2 =

    30

enter sampling frequency<fs2:
15
enter sampling frequency>fs2:
300
```

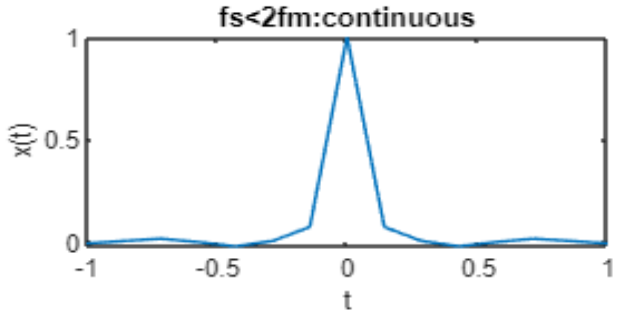


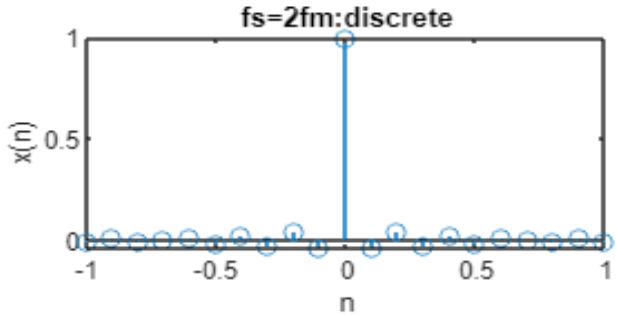
e) Sinc($2\pi ft$)

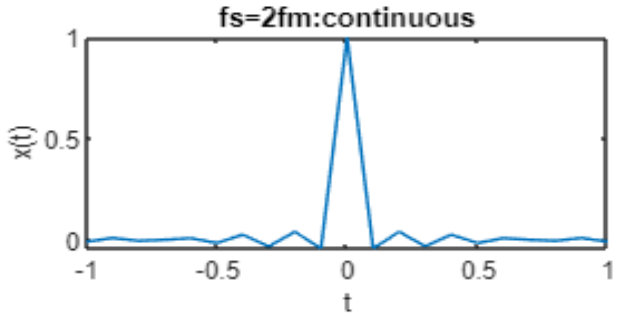
```
enter maximum signal frequency:
5

fs2 =

    10

enter sampling frequency<fs2:
7
enter sampling frequency>fs2:
200
```

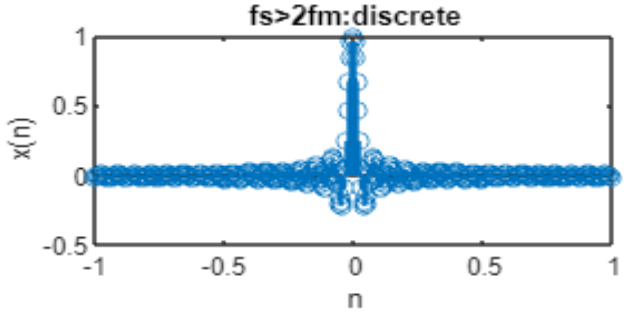


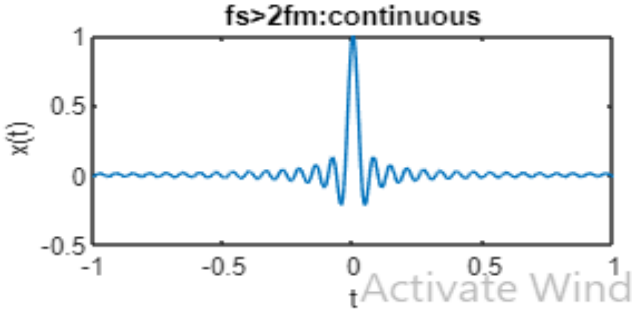












Sampling theorem has been verified using MATLAB.

EXPERIMENT:3

DISCRETE FOURIER TRANSFORM

1. AIM

To execute Discrete Fourier Transform (DFT) and Inverse Discrete Fourier Transform (IDFT) using MATLAB and observe its spectrum.

2. THEORY

DISCRETE FOURIER TRANSFORM

The discrete Fourier transform (DFT) converts a finite sequence of equally spaced samples of a function into a same-length sequence of equally spaced samples of the discrete-time Fourier transform (DTFT), which is a complex-valued function of frequency. The interval at which the DTFT is sampled is the reciprocal of the duration of the input sequence. The DFT is said to be a frequency domain representation of the original input sequence. It finds its application in Digital Signal processing including Linear filtering, Correlation analysis and Spectrum analysis.

Consider a complex series x [n] with N samples of the form x_0, x_1, x_2,....,x_{N-1} Where x is a complex number $x_i = x_{real} + j\, x_{imag}$. Further, assume that the series outside the range 0, N-1 is extended N-periodic, that is, xk = xk+N for all k. The DFT of this series is denoted as X (k) and has N samples. The forward transform (DFT) is defined as

$$X(k) = \sum_{n=0}^{N-1} \mathrm{x(n)} e^{-\frac{j2\pi nk}{N}} \qquad , for\ k = 0,1 \dots, N-1$$

INVERSE DISCRETE FOURIER TRANSFORM

The Fourier transform takes a signal in the so-called time domain (where each sample in the signal is associated with a time) and maps it without loss of information, into the frequency domain. The frequency domain representation is the same signal, in different form. The inverse Fourier transform maps the signal back from the frequency domain into the time domain. A time domain signal will usually consist of set of real values, where each value has an associated time (e.g., the signal consists of a time series). The inverse Fourier transform takes the frequency series of complex values and maps them back into the original time series. If the original time series considered of real values, the result of the IDFT will be complex numbers where the imaginary part is zero.

The inverse transform (IDFT) is defined as

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} \mathrm{X(k)} e^{\frac{j2\pi nk}{N}} \qquad ,for\ n = 0,1 \dots, N-1$$

Here DFT and IDFT is represented by summation method.

LINEAR TRANSFORMATION METHOD

Substitute $W_N = e^{-j2\pi N}$ in the equation for DFT and IDFT:

$$X(k) = \sum_{n=0}^{N-1} \mathrm{x(n)} {W_N}^{nk} \qquad ,for\ k = 0,1 \dots, N-1$$

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} \mathrm{X}(k) {W_N}^{-nk} \qquad ,for\ n = 0,1 \dots, N-1$$

W_N is known as the twiddle factor or Nth root of unity.

$$x_N = \begin{bmatrix} x(0) \\ x(1) \\ . \\ . \\ x(N-1) \end{bmatrix} \quad N \ \ point \ \ vector \ \ of \ \ signal \ \ x_N$$

$$X_N = \begin{bmatrix} X(0) \\ X(1) \\ . \\ . \\ X(N-1) \end{bmatrix} \quad N \ \ point \ \ vector \ \ of \ \ signal \ \ X_N$$

$$\begin{bmatrix} 1 & 1 & 1 & \dots & \dots & 1 \\ 1 & W_N & W_N^2 & \dots & \dots & W_N^{N-1} \\ . & W_N^2 & W_N^4 & \dots & \dots & W_N^{2(N-1)} \\ . & & & & & \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \dots & \dots & W_N^{(N-1)(N-1)} \end{bmatrix}$$

N - point DFT in matrix term is given by - $X_N = W_N x_N$

$W_N \longmapsto$ Matrix of linear transformation

$$Now, \quad x_N = W_N^{-1} X_N$$

IDFT in Matrix form is given by

$$x_N = \frac{1}{N} W_N^* X_N$$

Comparing both the expressions of x_N, $W_N^{-1} = \frac{1}{N} W_N^*$ and $W_N \times W_N^* = N[I]_{N \times N}$

Therefore, W_N is a linear transformation matrix, an orthogonal $unitary$ matrix.

From periodic property of W_N and from its symmetric property, it can be concluded that,

$$W_N^{k+N/2} = -W_N^k$$

Although the functions here are described as complex series, setting the imaginary part to 0 can represent real valued series. In general, the transform into the frequency domain will be a complex valued function, that is, with magnitude
and phase.

Magnitude=$|x(n)|=\sqrt[2]{Xreal^{2+}Ximg^2}$

Phase=$tan^{-1}(\frac{ximag}{xreal})$

INBUILT FUNCTION

- **fft:** Discrete Fourier transform.

fft(x) is the discrete Fourier transform (DFT) of vector x. For the matrices, the FFT operation is applied to each column. For N Dimensional arrays, the FFT operation
operates on the first non-singleton dimension.

- **ifft:** Inverse Discrete Fourier Transform

ifft(X(k)) is the Inverse discrete Fourier transform (IDFT) of DFT X(k).

LIBRARY FUNCTIONS

- **menu:** to create a multiple-choice dialogue box.

choice = menu (message, opt1, opt2, ..., optn) displays a modal menu dialog box containing the text in message and the choices specified by opt1, opt2,

... optn. The menu function returns the number of the selected menu item, or 0 if the user clicks the close button on the window.

- function: to declare a function which carries out a set of operations which can be called during the execution of program.
function [y1, ..., yN] = myfun (x1, ..., xM) declares a function named myfun that accepts inputs x1, ..., xM and returns outputs y1, ..., yN.

- fft and ifft- to find the DFT of a given signal and IDFT of given DFT sequence.

Question 1:
Write a MATLAB program to find the N point Discrete Fourier Transform (DFT) and the respective Inverse Discrete Fourier transform (IDFT) of a signal in given methods:
1. Summation method
2. Linear Transformation Method
3. Inbuilt command
Also plot the Magnitude and Phase response.

Question 2
Consider the sine wave, $5\sin(2\pi ft)$ sampled at 8Hz
a. Find out the DFT of the sampled signal
b. Find out the IDFT of the DFT obtained previously.
c. Plot the IDFT obtained in discrete and continuous waveform

3. PROGRAM

Question 1:

INPUT SIGNAL

```
xn=input('Enter the input sequence x(n):');
n=length(xn);
N=input('enter the value of N')
if N>n
   for k=n+1:N
      xn(k)=0;
   end
   xsig=xn;
else
   xsig=xn;
end
```

```
t=0:N-1;
stem(t,xsig);
ylabel ('Amplitude');
xlabel ('Time Index');
title ('Input Sequence');
```

DFT AND IDFT

```
xn=input('Enter the input sequence x(n):');
len=length(xn);
N=input('Enter the value of N:');
if N>len
   for k=len+1:N
      xn(k)=0;
   end
   xsig=xn;
else
   xsig=xn;
end
xt=xsig.';
w=exp((-1i*2*pi)/N);
method=menu('Choose method','1.Summation','2.Liner transformation','3.Inbuilt
command');
switch method
   case 1
      sum(xsig,N,w);
   case 2
      lt(xt,N,w);
   case 3
      inblt(xn,N);
end

function sum(sig,num,w)
Xk=zeros(1,num);
for k=0:num-1
   for n=0:num-1
      Xk(k+1)=Xk(k+1)+sig(n+1)*(w^(k*n));
   end
end
iXk=zeros(1,num);
for k=0:num-1
   for n=0:num-1
      iXk(k+1)=iXk(k+1)+(1/num)*Xk(n+1)*(w^(-1*k*n));
   end
```

```
end
disp("DFT of given dequence is:");
disp(Xk)
disp("IDFT of the DFT is:");
disp(iXk)

magnitude=abs(Xk);
phase=angle(Xk);
mag=abs(iXk);
ph=angle(iXk);

t=0:num-1;
subplot(2,2,1)
stem(t,magnitude)
ylabel ('Amplitude');
xlabel ('K');
title('Magnitude spectrum of DFT')

subplot(2,2,2)
stem(t,phase)
ylabel ('Amplitude');
xlabel ('K');
title('Phase spectrum of DFT')

subplot(2,2,3);
stem(t,mag);
ylabel ('Amplitude');
xlabel ('K');
title ('Magnitude spectrum of IDFT');

subplot(2,2,4);
stem(t,ph);
ylabel ('Phase');
xlabel ('K');
title ('Phase spectrum of IDFT');

end

function lt(sig,num,w)
y=zeros(num,num);
for k=0:num-1
   for n=0:num-1
      y(k+1,n+1)=w^(k*n);
   end
```

```
end
dft=y*sig;
z=zeros(num,num);
for k=0:num-1
    for n=0:num-1
        z(k+1,n+1)=w^(-1*k*n);
    end
end
idft=z*dft;
idft=idft./num;
disp("DFT of given dequence is:");
disp(dft)
disp("IDFT of the DFT is:");
disp(idft)

magnitude=abs(dft);
phase=angle(dft);
mag=abs(idft);
ph=angle(idft);

t=0:num-1;
subplot(2,2,1)
stem(t,magnitude)
ylabel ('Amplitude');
xlabel ('K');
title('Magnitude spectrum of DFT')

subplot(2,2,2)
stem(t,phase)
ylabel ('Amplitude');
xlabel ('K');
title('Phase spectrum of DFT')

subplot(2,2,3);
stem(t,mag);
ylabel ('Amplitude');
xlabel ('K');
title ('Magnitude spectrum of IDFT');

subplot(2,2,4);
stem(t,ph);
ylabel ('Phase');
xlabel ('K');
title ('Phase spectrum of IDFT');
```

```
end

function inblt(sig,num)
dft=fft(sig,num);
idft=ifft(dft,num);
disp("DFT of given dequence is:");
disp(dft)
disp("IDFT of the DFT is:");
disp(idft)

magnitude=abs(dft);
phase=angle(dft);
mag=abs(idft);
ph=angle(idft);

t=0:num-1;
subplot(2,2,1)
stem(t,magnitude)
ylabel ('Amplitude');
xlabel ('K');
title('Magnitude spectrum of DFT')

subplot(2,2,2)
stem(t,phase)
ylabel ('Amplitude');
xlabel ('K');
title('Phase spectrum of DFT')

subplot(2,2,3);
stem(t,mag);
ylabel ('Amplitude');
xlabel ('K');
title ('Magnitude spectrum of IDFT');

subplot(2,2,4);
stem(t,ph);
ylabel ('Phase');
xlabel ('K');
title ('Phase spectrum of IDFT');
end
```

Question 2

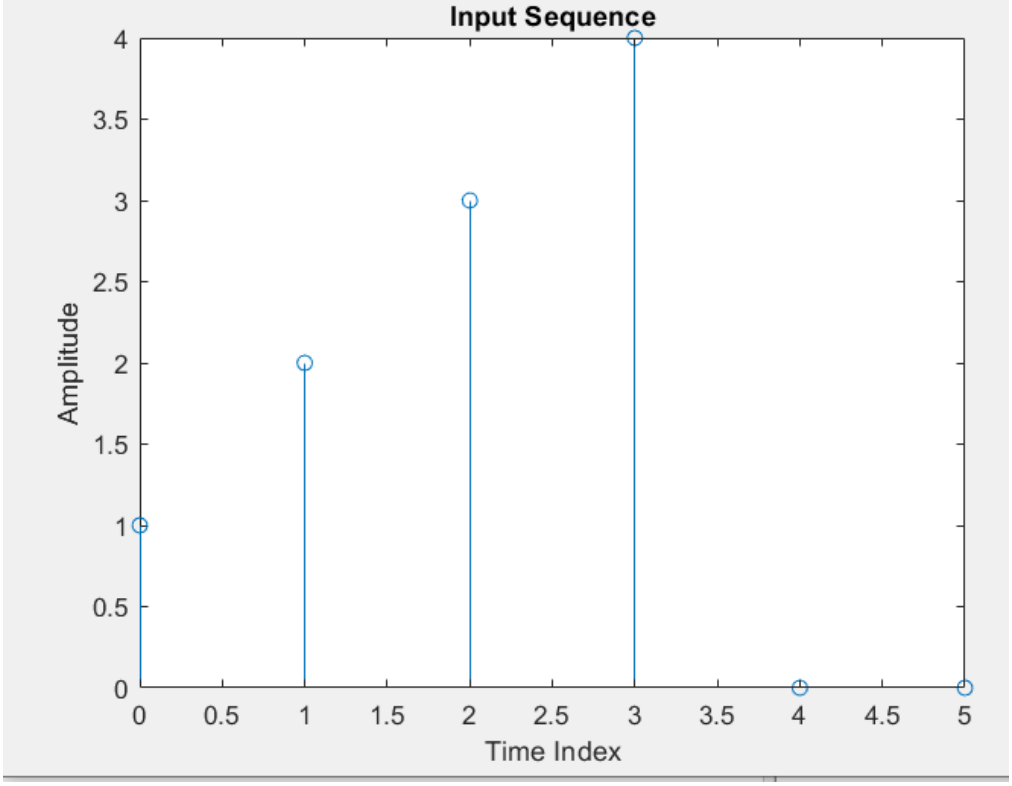
```
fs=8;
t=0:1/fs:10;
xn=5*sin(2*pi*t);
N=length(xn);

dft=fft(xn);
idft=ifft(dft);
disp(dft);
disp(idft);
magnitude=abs(dft);
phase=angle(dft);
mag=abs(idft);
ph=angle(idft);

subplot(4,2,1)
stem(t,xn)
ylabel ('Amplitude');
xlabel ('Time Index');
title ('Input Sequence-discrete');

subplot(4,2,2)
plot(t,xn)
ylabel ('Amplitude');
xlabel ('Time Index');
title ('Input Sequence-continuous');

subplot(4,2,3)
stem(t,magnitude)
ylabel ('Amplitude');
xlabel ('K');
title('Magnitude spectrum of DFT')

subplot(4,2,4)
stem(t,phase)
ylabel ('Amplitude');
xlabel ('K');
title('Phase spectrum of DFT')

subplot(4,2,5);
stem(t,mag)
ylabel ('Amplitude');
xlabel ('K');
title ('Magnitude spectrum of IDFT');
```

```
subplot(4,2,6);
stem(t,ph)
ylabel ('Phase');
xlabel ('K');
title ('Phase spectrum of IDFT');

subplot(4,2,7);
plot(t,idft);
ylabel ('amplitude');
xlabel ('K');
title ('IDFT sequence-continuous');

subplot(4,2,8);
stem(t,idft);
ylabel ('amplitude');
xlabel ('K');
title ('IDFT sequence-discrete');
```

4. RESULT

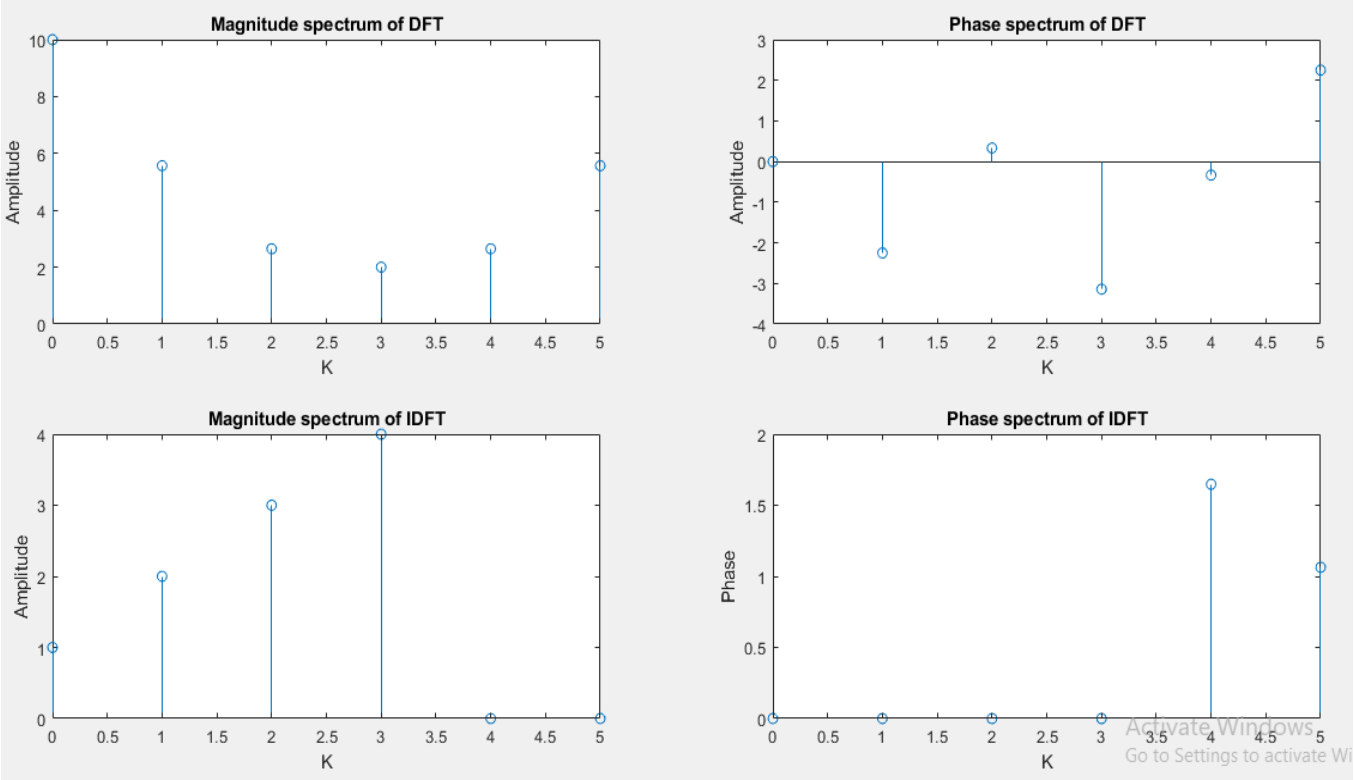
Question 1:

INPUT SIGNAL



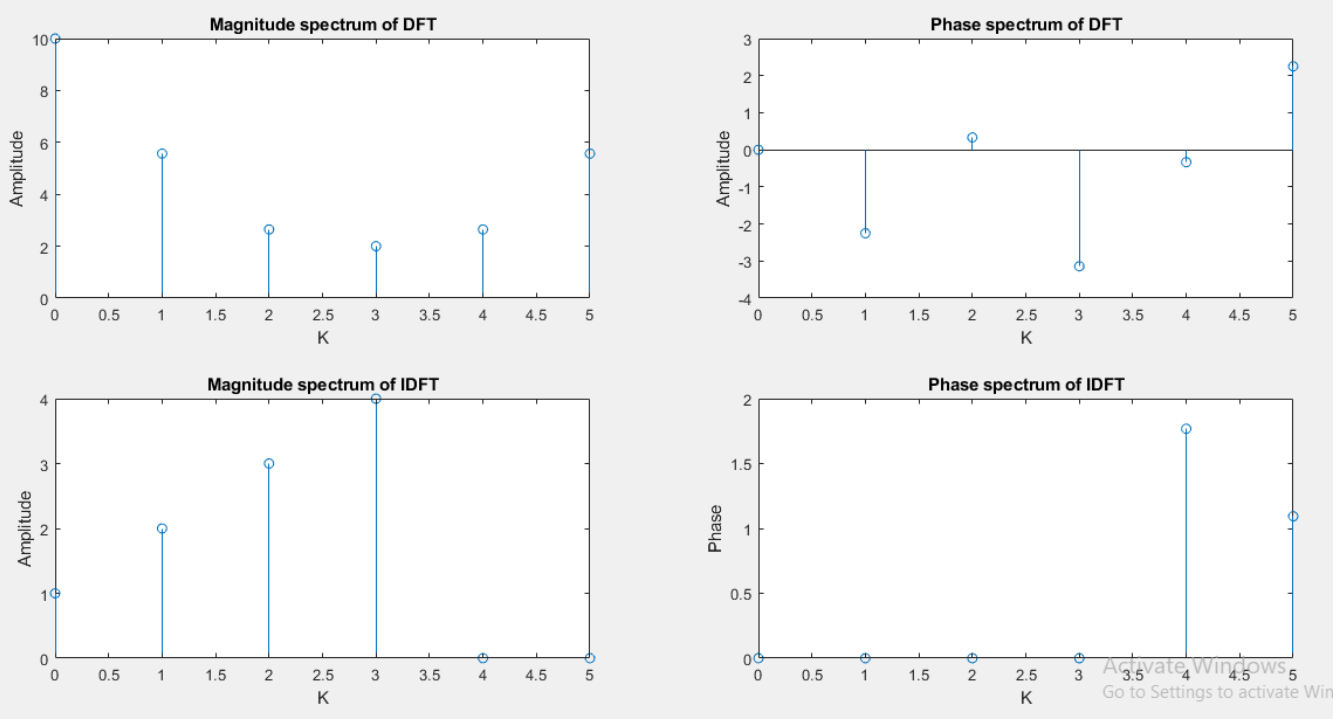
SUMMATION METHOD

```
Enter the input sequence x(n):[1,2,3,4]
Enter the value of N:6

Xk =

  10.0000 + 0.0000i  -3.5000 - 4.3301i   2.5000 + 0.8660i  -2.0000 - 0.0000i   2.5000 - 0.8660i  -3.5000 + 4.3301i


iXk =

   1.0000 - 0.0000i   2.0000 - 0.0000i   3.0000 + 0.0000i   4.0000 + 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i
```

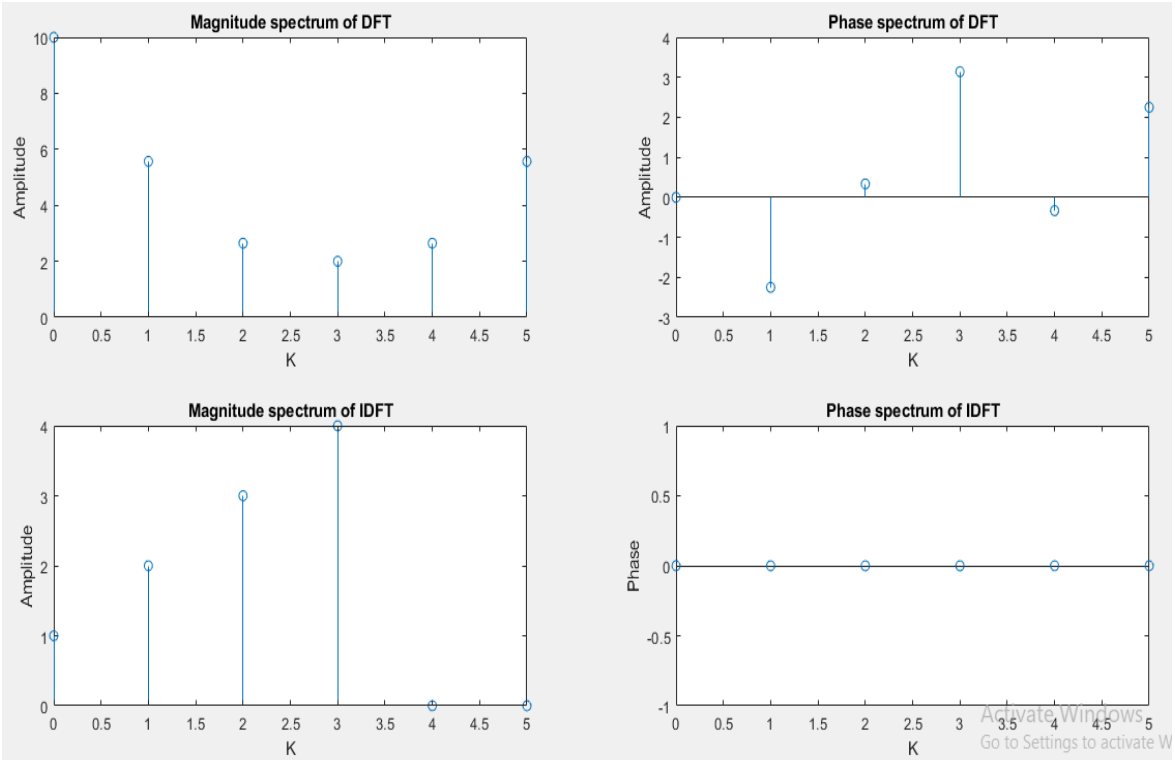
LINEAR TRANSFORMATION

```
Enter the input sequence x(n):[1,2,3,4]
Enter the value of N:6

dft =

  10.0000 + 0.0000i
  -3.5000 - 4.3301i
   2.5000 + 0.8660i
  -2.0000 - 0.0000i
   2.5000 - 0.8660i
  -3.5000 + 4.3301i


idft =

   1.0000 - 0.0000i
   2.0000 - 0.0000i
   3.0000 + 0.0000i
   4.0000 + 0.0000i
  -0.0000 + 0.0000i
   0.0000 + 0.0000i
```

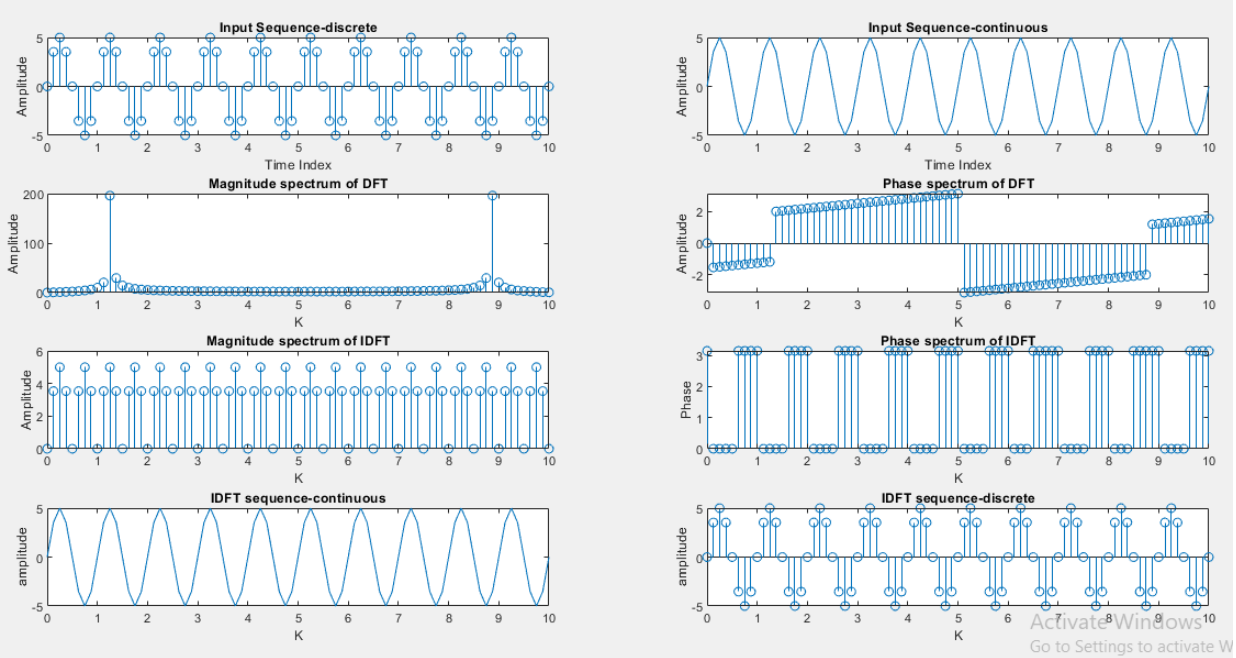


INBUILT FUNCTION

```
Enter the input sequence x(n):[1,2,3,4]
Enter the value of N:6

dft =

  10.0000 + 0.0000i  -3.5000 - 4.3301i   2.5000 + 0.8660i  -2.0000 + 0.0000i   2.5000 - 0.8660i  -3.5000 + 4.3301i


idft =

    1.0000    2.0000    3.0000    4.0000    0.0000    0.0000
```



Question 2



The program for DFT calculation was performed with library functions and without library functions. The results were verified by manual calculation.

INFERENCE:

-On plotting phase spectrum of IDFT of a real signal by summation method very minor imaginary parts are observed in the graph. These are call round off errors in MATLAB. This is eliminated on using inbuilt function ifft. An additional parameter known as symmetric can be given in the ifft function to remove this minor imaginary part.

EXPERIMENT NO: 4

LINEAR AND CIRCULAR CONVOLUTION

1. AIM

To prove linear and circular convolution using MATLAB code.

2. THEORY

LINEAR CONVOLUTION:

The response y[n] of a LTI system for any arbitrary input x[n] is given by convolution of impulse response h[n] of the system and the arbitrary input x[n].

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

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

If the input x[n] has N1 samples and impulse response h[n] has N2 samples then the output sequence y[n] will be a finite duration sequence consisting of (N1 + N2 - 1) samples. The convolution results in a non-periodic sequence called Aperiodic convolution.

CIRCULAR CONVOLUTION

The convolution of two periodic sequences with period N is called circular convolution of two signals x1[n] and x2[n] denoted by

$$y[n] = x_1[n] * x_2[n] = \sum_{k=0}^{N-1} x_1[(n-k) \bmod N]$$

$$x_2(k) \text{ or } \sum_{k=0}^{N-1} x_1(k)\ x_2[(n-k) \bmod N]$$

where $x_1[(n-k) \bmod N]$ is the reflected and circularly translated version of $x_1[n]$. $x_1[n] * x_2[n] = IDFT_N\{DFT_N(x_1[n]).DFT_N(x_2[n])\}$ It can be performed only if both the sequences consist of equal number of samples. If the sequences are different in length then convert the smaller size sequence to that of larger size by appending zeros.

One of the most efficient ways to implement convolution is by doing multiplication in the frequency. Sampling in the frequency requires periodicity in the time domain. However, due to the mathematical properties of the FFT this results in circular convolution.

LIBRARY FUNCTION:

· **conv**: Convolution and polynomial multiplication. C = conv (A, B) convolves vectors A and B. The resulting vector C's length is given by length(A)+length(B)-1. If A and B are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials in frequency domain.

. **cconv:** Modulo-n circular convolution. c = cconv(a,b) convolves vectors a and b. c = cconv(a,b,n) circularly convolves vectors a and b. n is the length of the resulting vector. You can also use cconv to compute the circular cross-correlation of two sequences.

. **circshift:** shift array circularly. Y = circshift(A,K) circularly shifts the elements in array A by K positions. If K is an integer, then circshift shifts along the first dimension of A whose size does not equal 1. If K is a vector of integers, then each element of K indicates the shift amount in the corresponding dimension of A.

.toeplitz: Toeplitz matrix. T = toeplitz(c,r) returns a nonsymmetric Toeplitz matrix with c as its first column and r as its first row. If the first elements of c and r differ, toeplitz issues a warning and uses the column element for the diagonal.

QUESTION

1) Given the impulse response h[n] and input x[n] of an LTI system, find its output y[n] using,

- The summation equation of linear convolution.
- Using matrix multiplication.
- Using circular convolution (Perform the circular convolution using both the summation equation and matrix multiplication method)
- Verify the answer using inbuilt functions.

3. PROGRAM
LINEAR CONVOLUTION

USING INBUILT COMMAND:

```
x=input('Enter the first sequence to be convoluted:');
h=input('Enter the second sequence to be convoluted:');

f=conv(x,h);
disp('The Linearly convolved sequence by inbuilt function is');
disp(f);
```

USING SUMMATION METHOD:

```
x=input('Enter the first sequence to be convoluted:');
h=input('Enter the second sequence to be convoluted:');
m=length(h);
n=length(x);
if m>n
    x=[x,zeros(1,m-n)];
elseif n>m
    h=[h,zeros(1,n-m)];
end
y=zeros(n,(n+m-1));
for i=1:n
    a=1;
   for j=1:(n+m-1)
       if a>m
           break
       elseif(i-j)<=0
           y(i,j)=x(i)*h(a);
           a=a+1;
       end
   end
end
a=zeros(1,(n+m-1));
for j=1:n
    for i=1:(n+m-1)
        a(i)=a(i)+y(j,i);
    end
end
disp('The Linearly convolved sequence by summation is');
disp(a);
```

USING MATRIX METHOD

```
xn=input('Enter the first input sequence:');
hn=input('Enter the system response sequence:');
n=length(xn);
m=length(hn);
r=m+n-1;
xn=xn.';
b=zeros(r,n);
a=1;
for j=1:n
    k=1;
    for i=1:r
        if a>i
            continue
        elseif k>m
            break
        else
            b(i,j)=hn(k);
            k=k+1;
        end
    end
    a=a+1;
end
Y=b*xn;

disp('The Linearly convolved sequence by matrix is');
disp(Y.');
```

LINEAR CONVOLUTION USING TOEPLITZ INBUILT COMMAND

```
x=input('Enter the first sequence to be convoluted:');
h=input('Enter the second sequence to be convoluted:');
m=length(x);
l=zeros(1,m-1);
H1=[h(1) l];
H2=[h l];
HT = toeplitz(H1,H2);
y1 = conv(x,h);
y2 = x*HT;

disp('The Linearly convolved sequence by inbuilt fn');
disp(y1);
disp('The Linearly convolved sequence by toeplitz matrix is');
disp(y2);
```

CONVOLUTION USING FFT FUNCTION

```
x=input('Enter the first input sequence:');
l1=length(x);
h=input('Enter the system response sequence:');
l2=length(h);
y=conv(x,h);

X=fft(x,length(y));
H=fft(h,length(y));
Y=X.*H;
y1=ifft(Y,length(Y));

disp('The Linearly convolved sequence by inbuilt fn');
disp(y);
disp('The inverse fourier transformed sequence is:');
disp(y1);
```

CIRCULAR CONVOLUTION

USING INBUILT FUCTION:

```
x=input('Enter the first sequence to be convoluted:');
l1=length(x);
h=input('Enter the second sequence to be convoluted:');
l2=length(h);

f=cconv(x,h);
disp('The Circularly convolved sequence by inbuilt function is');
disp(f);
```

USING SUMMATION METHOD:

```
x=input('Enter the first sequence to be convoluted:');
h=input('Enter the second sequence to be convoluted:');
m=length(h);
n=length(x);
x=[x,zeros(1,m-1)];
h=[h,zeros(1,n-1)];
r=m+n-1;
z=zeros(m,m);
for i=1:r
    zn=circshift(h,(i-1));
    for j=1:r
        z(j,i)=zn(j);
    end
end
```

```
x=x.';
Y=z*x;
disp('The circularly convolved sequence by summation is');
disp(Y.');
```

USING MATRIX METHOD:

```
x=input('Enter the first sequence to be convoluted:');
h=input('Enter the second sequence to be convoluted:');
m=length(h);
n=length(x);
x=[x,zeros(1,m-1)];
h=[h,zeros(1,n-1)];
hn=zeros(1,length(h));
s=length(h);
Y=zeros(1,s);
for k=1:s
   if k==1
      for i=1:s
         if s<0
            break
         elseif i==1
            hn(i)=h(i);
         else
            hn(i)=h(s);
            s=s-1;
         end
      end
      z=x.*hn;
    Y(k)=sum(z);
   else
      zn=circshift(hn,k-1);
      z=x.*zn;
      Y(k)=sum(z);

   end

end
disp('The circularly convolved sequence by matrix method is');
disp(Y)
```

4. RESULT

LINEAR CONVOLUTION USING INBUILT, SUMMATION AND MATRIX

```
Enter the first sequence to be convoluted:[3 4 6 9 5]
Enter the second sequence to be convoluted:[2 8 1 7]
The Linear convoluted sequence by inbuilt function is
     6    32    47    91   116    91    68    35

>> lin_summation
Enter the first sequence to be convoluted:[3 4 6 9 5]
Enter the second sequence to be convoluted:[2 8 1 7]
The Linear convoluted sequence by summation is
     6    32    47    91   116    91    68    35

>> lin_matrix
Enter the first input sequence:[3 4 6 9 5]
Enter the system response sequence:[2 8 1 7]
The Linear convoluted sequence by matrix is
     6    32    47    91   116    91    68    35
```

USING TOEPLITZ

```
Enter the first sequence to be convoluted:[3 4 6 9 5]
Enter the second sequence to be convoluted:[2 8 1 7]
The Linear convoluted sequence by inbuilt fn
     6    32    47    91   116    91    68    35

The Linear convoluted sequence by toeplitz matrix is
     6    32    47    91   116    91    68    35
```

USING FFT

```
>> lin_fft
Enter the first input sequence:[3 4 6 9 5]
Enter the system response sequence:[2 8 1 7]
The Linear convoluted sequence by inbuilt fn
     6    32    47    91   116    91    68    35

The inverse fourier transformed sequence is:
    6.0000   32.0000   47.0000   91.0000  116.0000   91.0000   68.0000   35.0000
```

CIRCULAR CONVOLUTION USING IBUILT, SUMMATION AND MATRIX

```
>> circ_inbuilt
Enter the first sequence to be convoluted:[3 4 6 9 5]
Enter the second sequence to be convoluted:[2 8 1 7]
The Circular convoluted sequence by inbuilt function is
    6.0000   32.0000   47.0000   91.0000  116.0000   91.0000   68.0000   35.0000

>> circ_summation
Enter the first sequence to be convoluted:[3 4 6 9 5]
Enter the second sequence to be convoluted:[2 8 1 7]
The circular convoluted sequence by summation is
     6    32    47    91   116    91    68    35

>> circ_matrix
Enter the first sequence to be convoluted:[3 4 6 9 5]
Enter the second sequence to be convoluted:[2 8 1 7]
The circular convoluted sequence by matrix method is
     6    32    47    91   116    91    68    35
```

The linear and circular convolutions are performed by using MATLAB script and the program results are verified by manual calculation and inbuilt function.

INFERENCE:

The input sequence was padded with (L+M-1) zeros to get the circular convoluted output same as that of linear convoluted.

EXPERIMENT NO:5

OVERLAP ADD AND OVERLAP SAVE METHOD

1. AIM

To study overlap add method and overlap save method using MATLAB code.

2. THEORY

OVERLAP ADD

The overlap add method is an efficient way to evaluate the discrete convolution of a very long signal with a finite impulse response (FIR) filter where h[m] = 0 for m outside the region [1, M].The concept here is to divide the problem into multiple convolutions of h[n] with short segments of x[n], where L is an arbitrary segment length. Because of this y[n] can be written as a sum of short convolutions.

Algorithm:

The signal is first partitioned into non-overlapping sequences, then the discrete Fourier transforms of the sequences are evaluated by multiplying the FFT xk[n] of with the FFT of h[n]. After recovering of yk[n] by inverse FFT, the resulting output signal is reconstructed by overlapping and adding the yk[n]. The overlap arises from the fact that a linear convolution is always longer than the original sequences. In the early days of development of the fast Fourier transform, L was often chosen to be a power of 2 for efficiency, but further development has revealed efficient transforms for larger prime factorizations of L, reducing computational sensitivity to this parameter.

OVERLAP SAVE

In this method, the size of the input data blocks is N=L+M-1 and the DFTs and the IDFTs are of length L. Each Data Block consists of the last M-1 data points of the previous block followed by L new data points to form a data sequence of length N=L+M-1. An N point DFT is computed for each data block. The impulse response of the FIR filter is increased in length by appending L-1 zeros and an N-point DFT of the sequence is computed once and stored. The multiplication of the

N-point DFTs for the mth block of data yields Ym(k)=h(k)Xm(k). Since the data record is of length N, the first M-1 points of Ym(n)are corrupted by aliasing and must be discarded. The last L points of Ym(n) are exactly the same as the result from linear convolution. To avoid loss of data due to aliasing, the last M-1 points of each data record are saved and these points become the first M-1 data points of the subsequent record. To begin the processing, the first M-1 point of the first record is set to zero. The resulting data sequence from the IDFT are given where the first M-1 points are discarded due to aliasing and the remaining L points constitute the desired result from the linear convolution. This segmentation of the input data and the fitting of the output data blocks together form the output sequence.

QUESTION

1. Given h[n] = {2,3,4}. Apply a long sequence of your own choice with length greater than 10. Find output sequence using following method
   1. Overlap Save
   2. Overlap Add

2.Verify your result using inbuilt command for convolution.

3. PROGRAM

OVERLAP SAVE

```
x=input("ENTER INPUT SEQUENCE x[n]: ");
l=input("ENTER SEQUENCE LENGTH L: ");
h=[2 3 4];
xl=length(x);
m=length(h);
z=m-1;
N=l+z;
c=zeros(l1+1,N);
f=N-1;
v=l+1;
for i=1:l1+1
   if i==1
      i1=1;
```

```
        for j=1:N
            if j<=z
             continue
            else
              c(i,j)=x(i1);
              i1=i1+1;
            end
        end
    else
        for j=1:N
            if f>N
                if v>xl
                    break
                else
                    c(i,j)=x(v);
                    v=v+1;
                end
            else
                c(i,j)=c((i-1),f);
                f=f+1;

            end
        end
        f=N-1;
    end
end
p=zeros(l1+1,N);

for i=1:l1+1
    x1=c(i,:);
    p(i,:)=cconv(x1,h,N);
end
n=1;
for i=1:l1+1
    for j=m:N
        y(n)=p(i,j);
        n=n+1;
    end
end
```

```
y=y(1:xl+m-1);
disp("Result after convolution:")
disp(p);
disp('The output sequence y[n]: ');
disp(y);
stem(y);
title('Overlap Save Method');
xlabel('n');
ylabel('y[n]');
```

OVERLAP ADD

```
x=input("ENTER INPUT SEQUENCE x[n]: ");
l=input("ENTER SEQUENCE LENGTH L: ");
xl=length(x);
l1=round(xl/l);
h=[2 3 4];
m=length(h);
z=m-1;
N=l+z;
c=zeros(l1+1,N);
u=1;
for i=1:l1+1
    for j=1:N
        if u>xl
            break
        elseif j>l
            break
        else
            c(i,j)=x(u);
            u=u+1;
        end
    end
end
p=zeros(l1+1,N);

for i=1:l1+1
    x1=c(i,:);
    p(i,:)=cconv(x1,h,N);
end
```

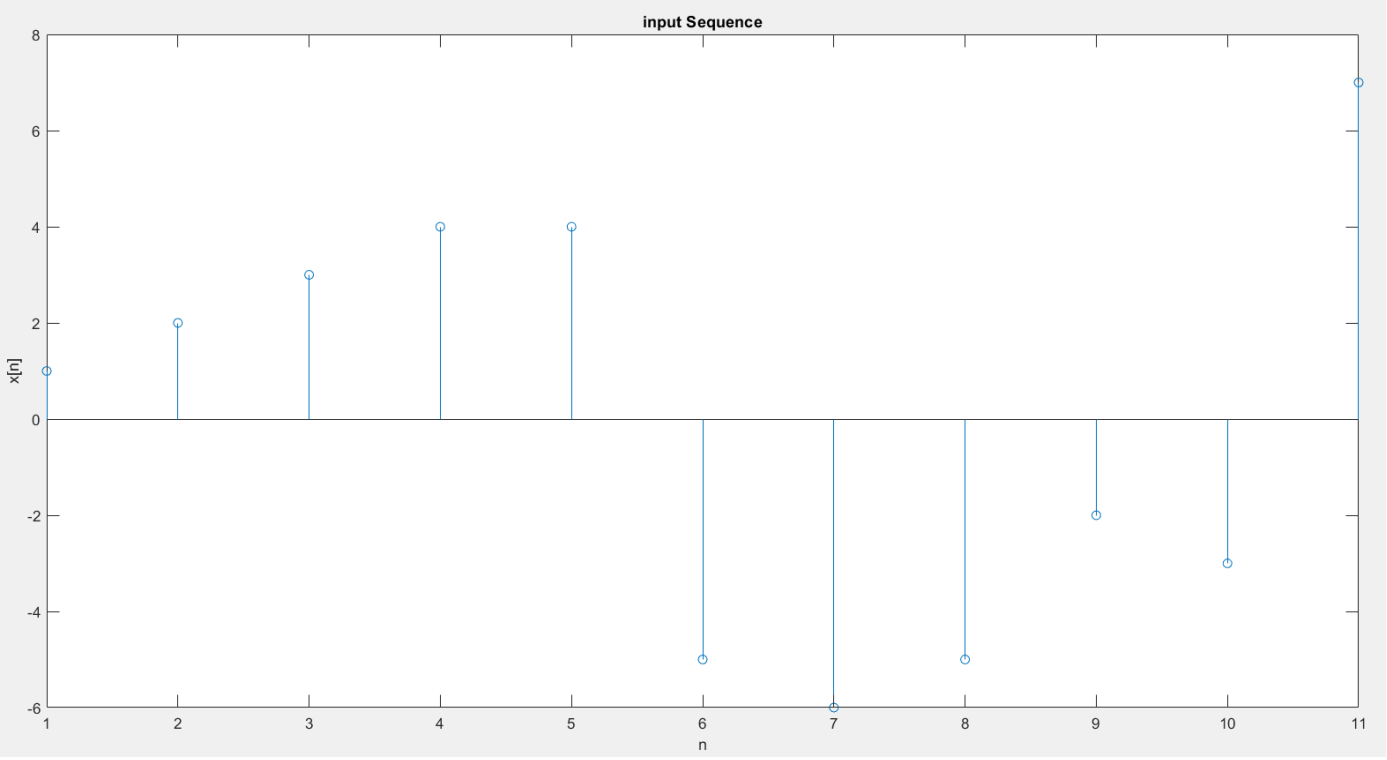
```
n=1;
for i=1:l1+1
   if i==1
      for j=1:l
         y(n)=p(i,j);
         n=n+1;
      end
      for j=l+1:N
         y(n)=p(i,j)+p(i+1,j-l);
         n=n+1;
      end
   elseif i>1 && i<l1
      for j=z+1:l
         y(n)=p(i,j);
         n=n+1;
      end
      for j=l+1:N
         y(n)=p(i,j)+p(i+1,j-l);
         n=n+1;
      end
   end
end
y=y(1:xl+m-1);
disp("Result after convolution:")
disp(p); %display intermediate results
disp("The output sequence y[n] is:");
disp(y); %display output
stem(y);
title('Overlap Add Method');
xlabel('n');
ylabel('y(n)');
```

INBUILT FUNCTION

```
x=input('Enter the input sequence x[n]: ');
h=[2 3 4];

f=cconv(x,h);
disp('Output y[n] using inbuilt fn:');
disp(f);
stem(f);
title('Convolution using inbuilt function');
xlabel('n');
ylabel('y[n]');
```

4. RESULT

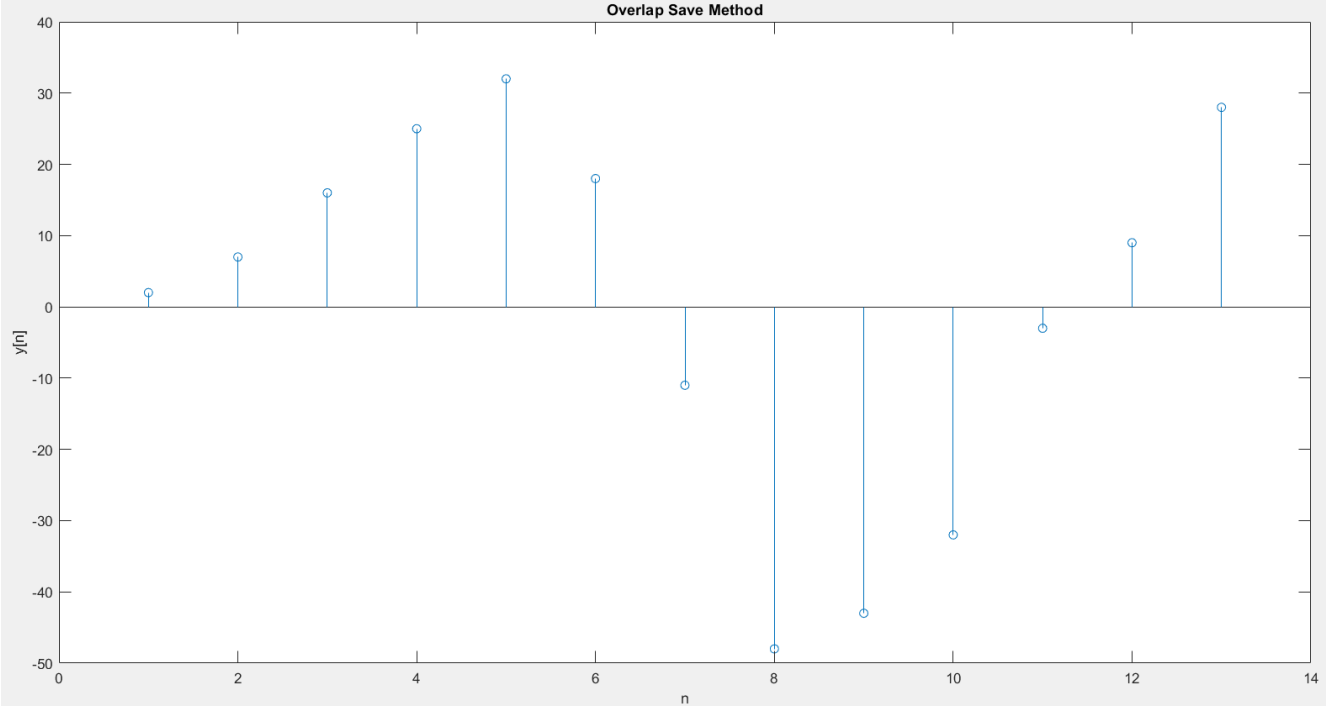
INPUT SEQUENCE



x[n]: [1 2 3 4 4 -5 -6 -4-1 -2 -3 7]

OVERLAP SAVE

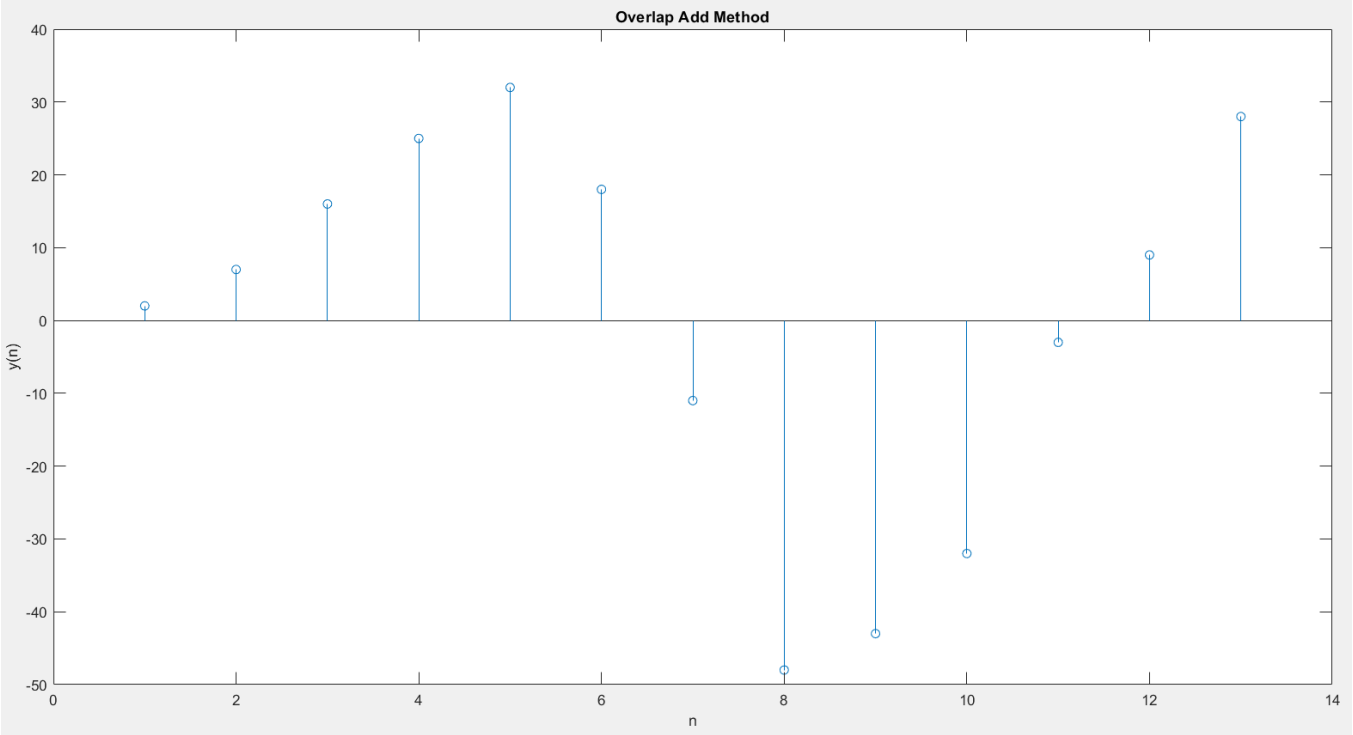
```
>> overlap_save
ENTER INPUT SEQUENCE x[n]: [1 2 3 4 4 -5 -6 -4-1 -2 -3 7]
ENTER SEQUENCE LENGTH L: 4
Result after convolution:
   24.0000   16.0000    2.0000    7.0000   16.0000   25.0000
  -33.0000   -3.0000   32.0000   18.0000  -11.0000  -48.0000
   16.0000  -28.0000  -43.0000  -32.0000   -3.0000    9.0000
   14.0000   21.0000   28.0000         0         0    0.0000

The output sequence y[n]:
    2.0000    7.0000   16.0000   25.0000   32.0000   18.0000  -11.0000  -48.0000  -43.0000  -32.0000   -3.0000    9.0000   28.0000
```



y1[n] : 24.0000 16.0000 2.0000 7.0000 16.0000 25.0000
y2[n] : -33.0000 -3.0000 32.0000 18.0000 -11.0000 -48.0000
y3[n] : 16.0000 -28.0000 -43.0000 -32.0000 -3.0000 9.0000
y4[n] : 14.0000 21.0000 28.0000 0 0 0.0000
y[n] : [2.0000 7.0000 16.0000 25.0000 32.0000 18.0000 -11.0000
-48.0000 -43.0000 -32.0000 -3.0000 9.0000 28.0000]

OVERLAP ADD

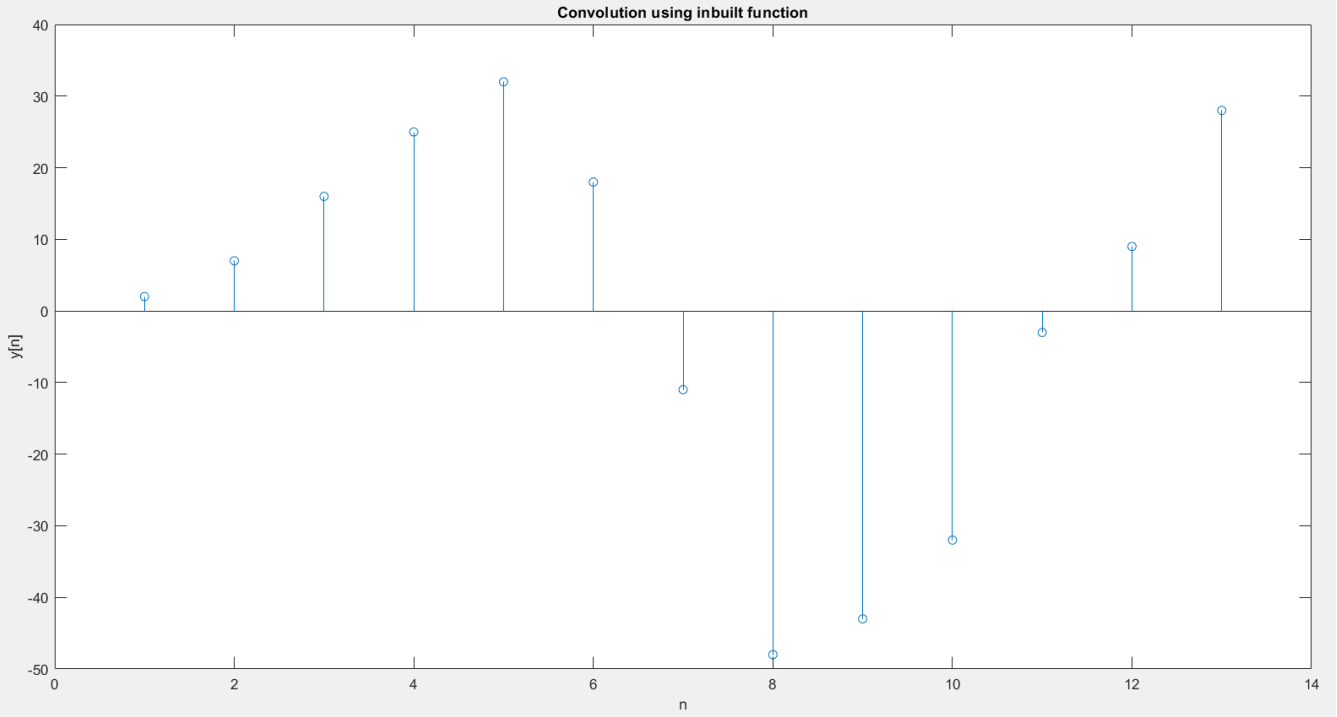
```
>> overlap_add
ENTER INPUT SEQUENCE x[n]: [1 2 3 4 4 -5 -6 -4-1 -2 -3 7]
ENTER SEQUENCE LENGTH L: 4
Result after convolution:
    2.0000    7.0000   16.0000   25.0000   24.0000   16.0000
    8.0000    2.0000  -11.0000  -48.0000  -39.0000  -20.0000
   -4.0000  -12.0000   -3.0000    9.0000   28.0000   -0.0000
         0         0         0         0         0         0

The output sequence y[n] is:
    2.0000    7.0000   16.0000   25.0000   32.0000   18.0000  -11.0000  -48.0000  -43.0000  -32.0000   -3.0000    9.0000   28.0000
```



y1[n] : 2.0000 7.0000 16.0000 25.0000 24.0000 16.0000
y2[n] : 8.0000 2.0000 -11.0000 -48.0000 -39.0000 -20.0000
y3[n] : -4.0000 -12.0000 -3.0000 9.0000 28.0000 -0.0000
y4[n] : 0 0 0 0 0 0
y[n] : [2.0000 7.0000 16.0000 25.0000 32.0000 18.0000 -11.0000
-48.0000 -43.0000 -32.0000 -3.0000 9.0000 28.0000]

INBUILT FUNCTION

```
>> circ_inbuilt
Enter the input sequence x[n]: [1 2 3 4 4 -5 -6 -4-1 -2 -3 7]
Output y[n] using inbuilt fn:
    2.0000    7.0000   16.0000   25.0000   32.0000   18.0000  -11.0000  -48.0000  -43.0000  -32.0000   -3.0000    9.0000   28.0000
```



Overlap Save and Overlap Add method for convolution of long input sequence is performed using MATLAB.

INFERENCE

-Reducing the length of input sequence to certain specified block and performing the overlap methods is efficient than using fft function.This method is much easier for manual calculation.

EXPERIMENT NO :6

RADIX – 2 FFT AND IFFT

1. AIM

To obtain the radix – 2 fft and ifft of the sequence.

2. THEORY

The radix-2 decimation-in-frequency and decimation-in-time fast Fourier transforms (FFTs) are the simplest FFT algorithms. Like all FFTs, they compute the discrete Fourier transform (DFT).
X(k)==∑N−1n=0x(n)e−(i2πnkN)∑N−1n=0x(n)WnkN
where for notational convenience WkN=e−(i2πkN). FFT algorithms gain their speed by reusing the results of smaller, intermediate computations to compute multiple DFT frequency outputs.

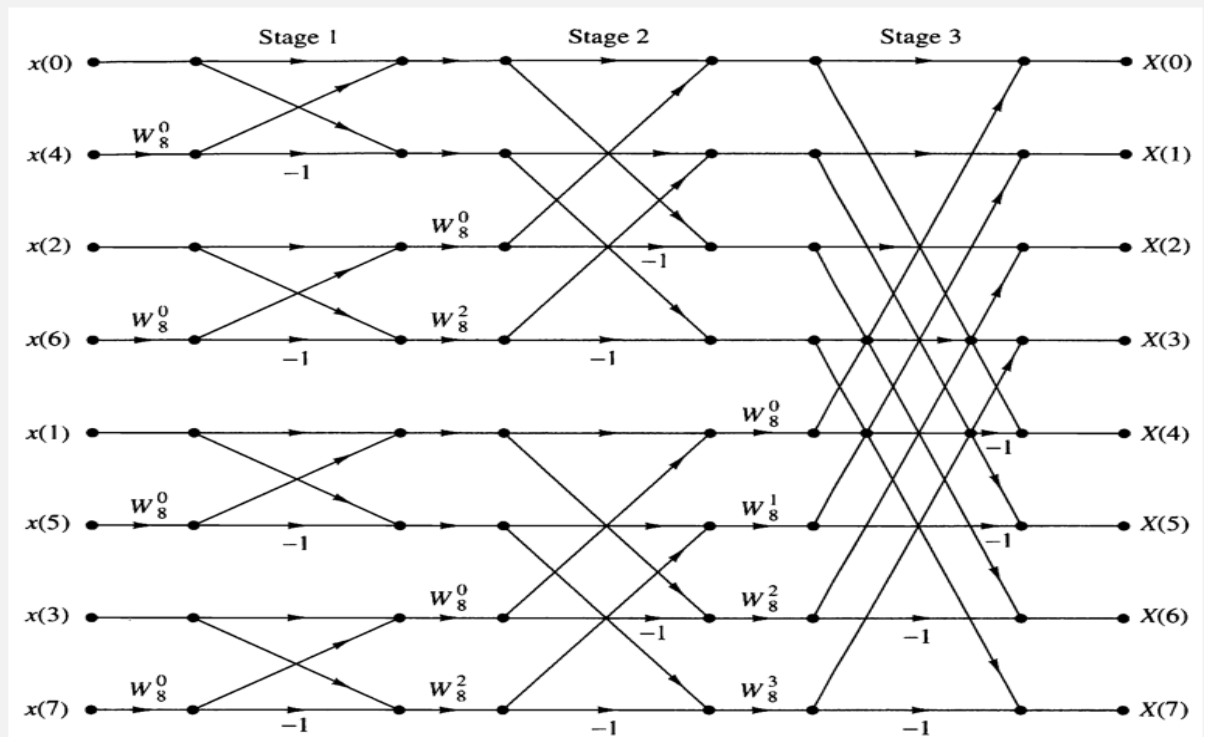
DECIMATION IN TIME

The radix-2 decimation-in-time algorithm rearranges the discrete Fourier transform (DFT) equation into two parts: a sum over the even-numbered discrete-time indices n=[0,2,4,…,N−2] and a sum over the odd-numbered indices n=[1,3,5,…,N−1]as in
Equation

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) e^{-\left(i\frac{2\pi nk}{N}\right)} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-\left(i\frac{2\pi\times(2n)k}{N}\right)} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) e^{-\left(i\frac{2\pi(2n+1)k}{N}\right)} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-\left(i\frac{2\pi nk}{\frac{N}{2}}\right)} + e^{-\left(i\frac{2\pi k}{N}\right)} \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) e^{-\left(i\frac{2\pi nk}{\frac{N}{2}}\right)} \\ &= \mathrm{DFT}_{\frac{N}{2}}\ [[x(0), x(2), \ldots, x(N-2)]] + W_N^k\, \mathrm{DFT}_{\frac{N}{2}}\ [[x(1), x(3), \ldots, x(N-1)]] \end{aligned}$$

The mathematical simplifications in Equation reveal that all DFT frequency outputs X(k) can be computed as the sum of the outputs of two length-N/2 DFTs, of the even-indexed and odd-indexed discrete-time samples, respectively, where the odd-indexed short DFT is multiplied by a so-called **twiddle factor** term W_N^k=e^−(i2πk/N) is called a **decimation in time** because the time samples are rearranged in alternating groups, and a **radix-2** algorithm because there are two groups. Figure graphically illustrates this form of the DFT computation, where for convenience the frequency outputs of the length-N/2 DFT of the even-indexed time samples are denoted G(k) and those of the odd-indexed samples as H(k). Because of the periodicity with N/2 frequency samples of these length-N/2 DFTs, G(k) and H(k) can be used to compute **two** of the length-N DFT frequencies, namely X(k) and X(k+N/2), but with a different twiddle factor. This reuse of these short-length DFT outputs

gives the FFT Its computational savings.



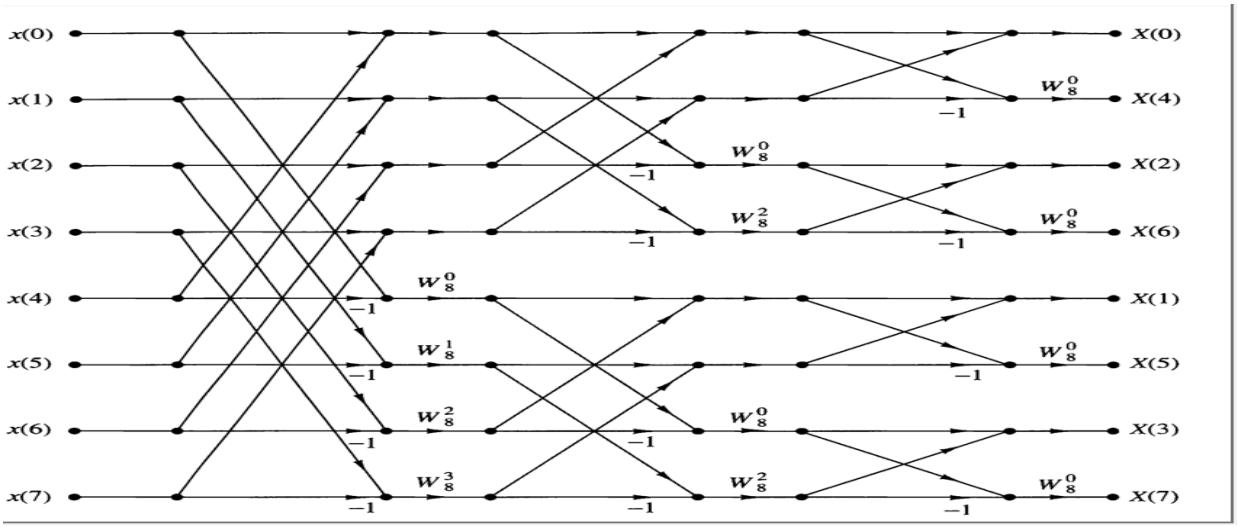
DECIMATION IN FREQUENCY

The radix-2 decimation-in-frequency algorithm rearranges the discrete Fourier transform (DFT) equation into two parts: computation of the even-numbered discrete-frequency indices X(k) for k=[0,2,4,…,N−2] (or X(2r) as in [link]) and computation of the odd-numbered indices k=[1,3,5,…,N−1] (or X(2r+1).

$$\begin{aligned} X(2r) &= \sum_{n=0}^{N-1} x(n) W_N^{2rn} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{2rn} + \sum_{n=0}^{\frac{N}{2}-1} x\left(n+\tfrac{N}{2}\right) W_N^{2r\left(n+\frac{N}{2}\right)} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{2rn} + \sum_{n=0}^{\frac{N}{2}-1} x\left(n+\tfrac{N}{2}\right) W_N^{2rn} 1 \\ &= \sum_{n=0}^{\frac{N}{2}-1} \left(x(n) + x\left(n+\tfrac{N}{2}\right)\right) W_{\frac{N}{2}}^{rn} \\ &= \mathrm{DFT}_{\frac{N}{2}} \left[x(n) + x\left(n+\tfrac{N}{2}\right)\right] \end{aligned}$$

$$\begin{aligned} X(2r+1) &= \sum_{n=0}^{N-1} x(n) W_N^{(2r+1)n} \\ &= \sum_{n=0}^{\frac{N}{2}-1} \left(x(n) + W_N^{\frac{N}{2}} x\left(n+\tfrac{N}{2}\right)\right) W_N^{(2r+1)n} \\ &= \sum_{n=0}^{\frac{N}{2}-1} \left(\left(x(n) - x\left(n+\tfrac{N}{2}\right)\right) W_N^{n}\right) W_{\frac{N}{2}}^{rn} \\ &= \mathrm{DFT}_{\frac{N}{2}} \left[\left(x(n) - x\left(n+\tfrac{N}{2}\right)\right) W_N^{n}\right] \end{aligned}$$

The mathematical simplifications in and reveal that both the even-indexed and odd-indexed frequency outputs X(k) can each be computed by a length-N/2DFT. The inputs to these DFTs are sums or differences of the first and second halves of the input signal, respectively, where the input to the short DFT producing the odd-indexed frequencies is multiplied by a so-called **twiddle factor** term W_N^k=e^−(i2πk/N). This is called a **decimation in frequency** because the frequency samples are computed separately in alternating groups, and a **radix-2** algorithm because there are two groups. Figure graphically illustrates this form of the DFT computation. This conversion of the full DFT into a series of shorter DFTs with a simple pre-processing step gives the decimation-in-frequency FFT its computational savings.



QUESTION

FFT using MATLAB

1.Find the DFT of a sequence $x[n]$ by making use of the radix-2 FFT algorithms. Implement (i) Decimation in time (ii) Decimation in frequency.

2.Using the FFT of $x[n]$ obtained in previous section, find $x[n]$. Implement (i) DIT-IFFT (IFFT using DIT) (II) DIF-IFFT (IFFT using DIF).

3.Verify the results using inbuilt commands of MATLAB.

3. PROGRAM

DIT FFT

```
%dit for 8 point dft
xn=input("enter the input sequence of length below 9:
");
N=length(xn);
x=[xn,zeros(1,8-N)];
v=bitrevorder(x);
n=length(v);
W=exp(-1i*(2*pi/n));
for j=1:n
    if j==1
        G(1,1)=v(1)+v(2);
    end
    for j=2:n
        r=rem(j,2);
        if r~=0
            G(1,j)=v(j)+v(j+1);
        elseif r==0
            G(1,j)=v(j-1)-v(j);
        end
    end
end
s=W^2;
for j=1:n
    r=rem(j,2);
    if j==1
        F(1,1)=G(1)+G(3);
    elseif r==0
        if j==2 || j==6
            F(1,j)=G(j)+(s*(G(j+2)));
        else
            F(1,j)=G(j-2)-(s*(G(j)));
        end
    elseif r~=0
        if j==3 || j==7
            F(1,j)=G(j-2)-G(j);
        else
            F(1,j)=G(j)+G(j+2);
        end
    end
end
m=0;
for j=1:4
    if m<=3
        X(1,j)=F(j)+((W^m)*F(j+4));
```

```
            m=m+1;
        end
    end
    m=0;
    for j=5:8
         if m<=3
             X(1,j)=F(j-4)-((W^m)*F(j));
             m=m+1;
         end
    end
    disp("The DIT DFT is:")
    disp(X)
```

DIF FFT

```
%dif_8 point dft
xn=input("enter the input sequence of length below 9: ");
N=length(xn);
v=[xn,zeros(1,8-N)];
n=length(v);
W=exp(-1i*(2*pi/n));
for j=1:n
     for j=1:4
         f(1,j)=v(j)+v(j+4);
     end
     m=0;
     for j=5:8
         if m<=3
             f(1,j)=(v(j-4)-v(j))*(W^m);
             m=m+1;
         end
     end
end
s=W^2;
for j=1:n
     for j=1:2
         g(1,j)=f(j)+f(j+2);
     end
     for j=5:6
         g(1,j)=f(j)+f(j+2);
     end
     m=0;
     for j=3:4
         if m<=2
             g(1,j)=(f(j-2)-f(j))*(W^m);
             m=m+2;
         end
     end
```

```
    m=0;
    for j=7:8
         if m<=2
            g(1,j)=(f(j-2)-f(j))*(W^m);
            m=m+2;
         end
    end
end
for j=1:4
    if j==1 || j==3
        X(1,j)=g(j)+g(j+1);
    else
        X(1,j)=g(j+3)+g(j+4);
    end
end
for j=5:8
    if j==5 || j==7
        X(1,j)=(g(j-4)-g(j-3));
    else
        X(1,j)=(g(j-1)-g(j));
    end
end
disp("The DIF DFT is:")
disp(X)
```

DIF IFFT

```
%dif_idft
Xk=input("enter the DFT sequence of length below 9: ");
N=length(Xk);
y=[Xk,zeros(1,8-N)];
X=conj(y);
v=bitrevorder(X);
n=length(v);
W=exp(-1i*(2*pi/n));
for j=1:n
    if j==1
        g(1,1)=v(1)+v(2);
    end
    for j=2:n
        r=rem(j,2);
        if r~=0
            g(1,j)=v(j)+v(j+1);
        elseif r==0
            g(1,j)=v(j-1)-v(j);
        end
    end
end
s=W^2;
```

```
for j=1:n
    r=rem(j,2);
    if j==1
        f(1,1)=g(1)+g(3);
    elseif r==0
        if j==2 || j==6
            f(1,j)=g(j)+(s*(g(j+2)));
        else
            f(1,j)=g(j-2)-(s*(g(j)));
        end
    elseif r~=0
        if j==3 || j==7
            f(1,j)=g(j-2)-g(j);
        else
            f(1,j)=g(j)+g(j+2);
        end
    end
end
for j=1:n
    m=0;
    for j=1:4
        if m<=3
            x(1,j)=f(j)+((W^m)*f(j+4));
            m=m+1;
        end
    end
    m=0;
    for j=5:8
         if m<=3
             x(1,j)=f(j-4)-((W^m)*f(j));
             m=m+1;
         end
    end
end
x=x/8;
disp("The DIF IDFT is:")
disp(round(x))
```

DIT IFFT

```
%dit_idft
Xk=input("enter the DFT sequence of length below 9: ");
N=length(Xk);
y=conj(Xk);
v=[y,zeros(1,8-N)];
n=length(v);
W=exp(-1i*(2*pi/n));
for j=1:n
```

```
    for j=1:4
        F(1,j)=v(j)+v(j+4);
    end
    m=0;
    for j=5:8
        if m<=3
            F(1,j)=(v(j-4)-v(j))*(W^m);
            m=m+1;
        end
    end
end
s=W^2;
for j=1:n
    for j=1:2
        G(1,j)=F(j)+F(j+2);
    end
    for j=5:6
        G(1,j)=F(j)+F(j+2);
    end
    m=0;
    for j=3:4
        if m<=2
            G(1,j)=(F(j-2)-F(j))*(W^m);
            m=m+2;
        end
    end
    m=0;
    for j=7:8
         if m<=2
            G(1,j)=(F(j-2)-F(j))*(W^m);
            m=m+2;
         end
    end
end
for j=1:n
    for j=1:4
        if j==1 || j==3
            x(1,j)=G(j)+G(j+1);
        else
            x(1,j)=G(j+3)+G(j+4);
        end
    end
    for j=5:8
        if j==5 || j==7
            x(1,j)=(G(j-4)-G(j-3));
        else
            x(1,j)=(G(j-1)-G(j));
        end
```

```
    end
end
x=x/8;
disp("The DIT IDFT is:")
disp(round(x))
```

USING INBUILT COMMAND

```
x1=input('enter the sequence: ');
n=input('enter the length: ');
X=fft(x1,n);
disp('N-Point DFT of a given sequence: ');
disp(X);
x=ifft(X);
disp("IDFT Of obtained DFT: ")
disp(x)
```

4. RESULT

DIT FFT

```
mmand Window
>> dit_sample
enter the input sequence of length below 9: [1 2 3 4 4 3 2 1]
The DIT DFT is:
  Columns 1 through 6

  20.0000 + 0.0000i  -5.8284 - 2.4142i   0.0000 + 0.0000i  -0.1716 - 0.4142i   0.0000 + 0.0000i  -0.1716 + 0.4142i

  Columns 7 through 8

   0.0000 + 0.0000i  -5.8284 + 2.4142i
```

DIF FFT

```
mmand Window
>> dif_dft
enter the input sequence of length below 9: [1 2 3 4 4 3 2 1]
The DIF DFT is:
  Columns 1 through 6

  20.0000 + 0.0000i  -5.8284 - 2.4142i   0.0000 + 0.0000i  -0.1716 - 0.4142i   0.0000 + 0.0000i  -0.1716 + 0.4142i

  Columns 7 through 8

   0.0000 + 0.0000i  -5.8284 + 2.4142i
```

DIF IFFT

ommand Window

```
>> dif_idft
enter the DFT sequence of length below 9: [20,-5.828-2.414i,0,-0.172-0.414i,0,-0.172+0.414i,0,-5.828+2.414i]
The DIT IDFT is:
    1     2     3     4     4     3     2     1
```

DIT IFFT

Command Window

```
>> dit_idft
enter the DFT sequence of length below 9: [20,-5.828-2.414i,0,-0.172-0.414i,0,-0.172+0.414i,0,-5.828+2.414i]
The DIF IDFT is:
    1     2     3     4     4     3     2     1
```

USING INBUILT COMMAND

```
>> idft_dft
enter the sequence: [1 2 3 4 4 3 2 1]
enter the length: 8
N-Point DFT of a given sequence:
  Columns 1 through 6

  20.0000 + 0.0000i  -5.8284 - 2.4142i   0.0000 + 0.0000i  -0.1716 - 0.4142i   0.0000 + 0.0000i  -0.1716 + 0.4142i

  Columns 7 through 8

   0.0000 + 0.0000i  -5.8284 + 2.4142i

IDFT Of obtained DFT:
    1     2     3     4     4     3     2     1
```

The radix – 2 fft algorithm is studied using MATLAB.

EXPERIMENT NO: 7

FIR FILTER DESIGN

1. AIM

To design FIR filters with different window functions (Rectangular, Triangular, Hamming, Hanning & Blackman windows) using MATLAB.

2. THEORY

Filters have a variety of applications in data acquisition and analysis. They are used to alter the frequency content of a time signal by either reducing or amplifying certain frequencies. The filter plays a vital role in analog and digital signal processing. Analog filters basically consist of resistors, capacitors and inductors which are also called passive components. The digital filter system has two types which includes the IIR or recursive filter and FIR or non-recursive filter. FIR digital filter designed a linear phase digital filter which is convenient for data transmission, telecommunication system and image processing applications. Whereas IIR filter is used much in application such as high speed and low-power communication transceivers systems. As FIR requires large memory in order to store the previous input and previous output but FIR require less memory space to store present and past value of input.

FINITE IMPULSE RESPONSE (FIR) FILTERS

A digital filter is a discrete time LTI system. It is classified based on the length of the impulse response as
IIR filters: Where h [n] has infinite number of samples and is recursive type.
FIR filters: They are non-recursive type and h [n] has finite number of samples. Most of the FIR design methods are interactive procedures and hence require more memory and execution time. Also, implementation of narrow transition band filter is costly. But there are certain reasons to go for FIR.

Types of windows available are Rectangular, Hamming, Hanning, Triangular, Blackman window etc.

Rectangular window

$$w_R(n) \triangleq \begin{cases} 1, & -\frac{M-1}{2} \le n \le \frac{M-1}{2} \\ 0, & \text{otherwise} \end{cases}$$

Hamming window

$$w[n] = \begin{cases} 0.54 - 0.46cos(\frac{2n\pi}{M}) & 0 \le n \le M \\ 0 & otherwise \end{cases}$$

Hanning window

$$w(n) = 0.5 - 0.5cos\left(\frac{2\pi n}{M-1}\right) \qquad 0 \le n \le M-1$$

Triangular window

The *triangular window* coefficients are given by the following formula

$$a(k) = \frac{2}{N-1}\left(\frac{N-1}{2} - \left|k - \frac{N-1}{2}\right|\right) = 1 - \left|\frac{k - \frac{N-1}{2}}{\frac{N-1}{2}}\right|$$

where N is the length of the filter and k = 0, 1, ..., N − 1.

Blackman window

The following equation defines the Blackman window of length N:

$$w(n) = 0.42 - 0.5\cos\left(\frac{2\pi n}{L-1}\right) + 0.08\cos\left(\frac{4\pi n}{L-1}\right), \quad 0 \le n \le M-1$$

where M is $N/2$ when N is even and $(N + 1)/2$ when N is odd.

QUESTION

DESIGN FIR FILTERS USING WINDOW TECHNIQUE

1) Using MATLAB, design a $M-1$ order LPF & HPF. Make use of Rectangular, Triangular, Hamming, Hanning & Blackman windows.

The outputs required are,

a) Plot of the window function $w[n]$ and its spectrum $W(\omega)$.

b) Plot of the impulse response $h[n]$ and its frequency response $H(\omega)$ of the designed filter.

c) Give a input $x[n] = [1,1,1,1,0,0,0,0]$ to the designed filters and find the output $y[n]$. Plot the input $x[n]$, its spectrum $X(\omega)$. Plot the output $y[n]$ and its spectrum $Y(\omega)$ for the various filters designed and comment on how windowing has affected the spectrum of the original input.

LIBRARY FUNCTIONS:

- **fir1** FIR filter design using the Window method. B = fir1(N,Wn) designs an Nth order low pass FIR digital filter and returns the filter coefficients of vector B of length (N+1). The cut-off frequency Wn must be between 0 < Wn < 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 Wn is -6 dB.
- B = fir1(N,Wn,'high') designs an Nth order high pass filter. You can also use
- B = fir1(N,Wn,'low') to design a low pass filter.

  B = fir1(N,Wn,WIN) designs an N-th order. FIR filter using the vector WIN of (N+1) length to window the impulse response. If empty or omitted, fir1 uses a Hamming window of length N+1.

  We can specify windows from the Signal Processing Toolbox, such as boxcar, hamming, hanning, bartlett, blackman, kaiser or chebwin

w = **hamming**(n) returns an n-point symmetric Hamming window in the column vector w. n should be a positive integer.

w = **hanning**(n) returns an n-point symmetric Hann window in the column vector w. n must be a positive integer.

w=**triang**(n) returns an n-point triangular window in the column vector w. The triangular window is very similar to a Bartlett window. The Bartlett window always ends with zeros at samples 1 and n, while the triangular window is nonzero at those points. For n odd, the center (n-2) points of triang(n-2) are equivalent to **bartlett**(n).

w = **rectwin**(n) returns a rectangular window of length n in the column vector w. This function is provided for completeness. A rectangular window is equivalent to no window at all.

PROGRAM

```
f=input("enter cut off freq:");
N=input('Enter order of filter:');
wvtool(rectwin(N+1),blackman(N+1),hamming(N+1),hanning(N+1),triang(N+1));
ip=menu('Enter type of filter','High pass','Low pass');
if ip==1
   ft={'high'};
else
   ft={'low'};
end
wc=f*pi;
w=0:0.1:pi;
b=fir1(N,wc/pi,ft,rectwin(N+1));
h=freqz(b,1,w);
subplot(3,2,1)
plot(w/pi,abs(h))
grid;
xlabel('normalised frequency');
ylabel('magnitude in dB')
title('FIR USING RECTANGULAR WINDOW')
b=fir1(N,wc/pi,ft,blackman(N+1));
h=freqz(b,1,w);
subplot(3,2,2)
plot(w/pi,abs(h))
```

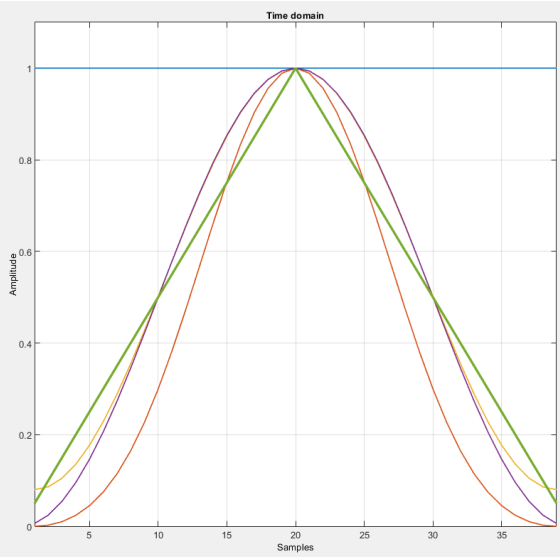
```
grid;
xlabel('normalised frequency');
ylabel('magnitude in dB')
title('FIR USING BLACKMAN WINDOW')
b=fir1(N,wc/pi,ft,hamming(N+1));
h=freqz(b,1,w);
subplot(3,2,3)
plot(w/pi,abs(h));
grid;
xlabel('normalised frequency');
ylabel('magnitude in dB')
title('FIR USING HAMMING WINDOW')
b=fir1(N,wc/pi,ft,hanning(N+1));
h=freqz(b,1,w);
subplot(3,2,4)
plot(w/pi,abs(h));
grid;
xlabel('normalised frequency');
ylabel('magnitude in dB')
title('FIR USING HANNING WINDOW')
b=fir1(N,wc/pi,ft,triang(N+1));
h=freqz(b,1,w);
subplot(3,2,5)
plot(w/pi,abs(h));
```

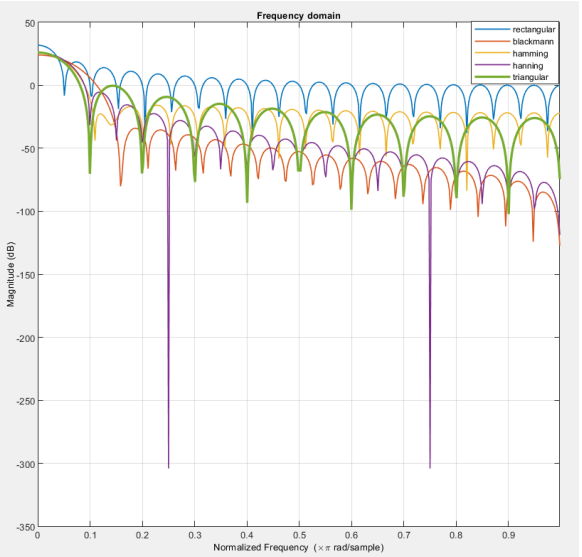
```
grid;

xlabel('normalised frequency');

ylabel('magnitude in dB')

title('FIR USING TRIANGULAR WINDOW')
```

INPUT: Cut off freq-0.5
Order of Filter: 38

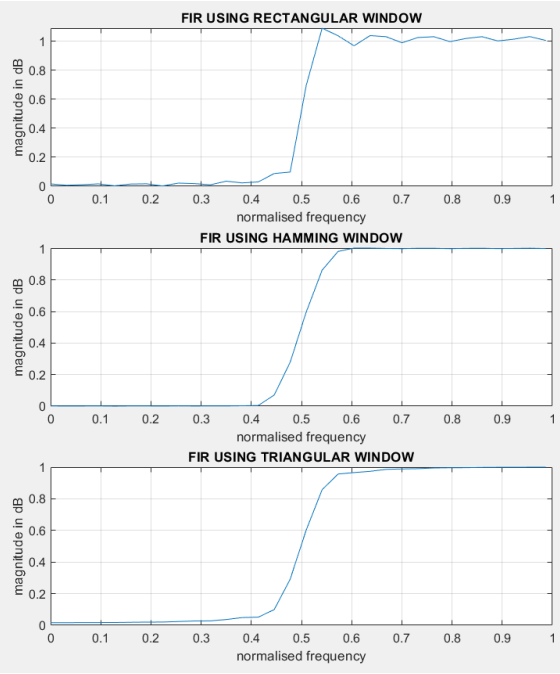
RESULTS

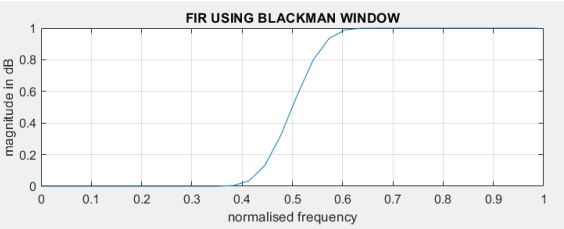
WINDOW FUNCTION

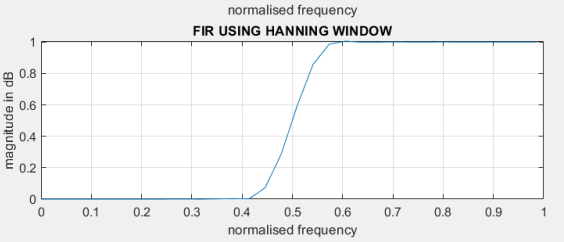




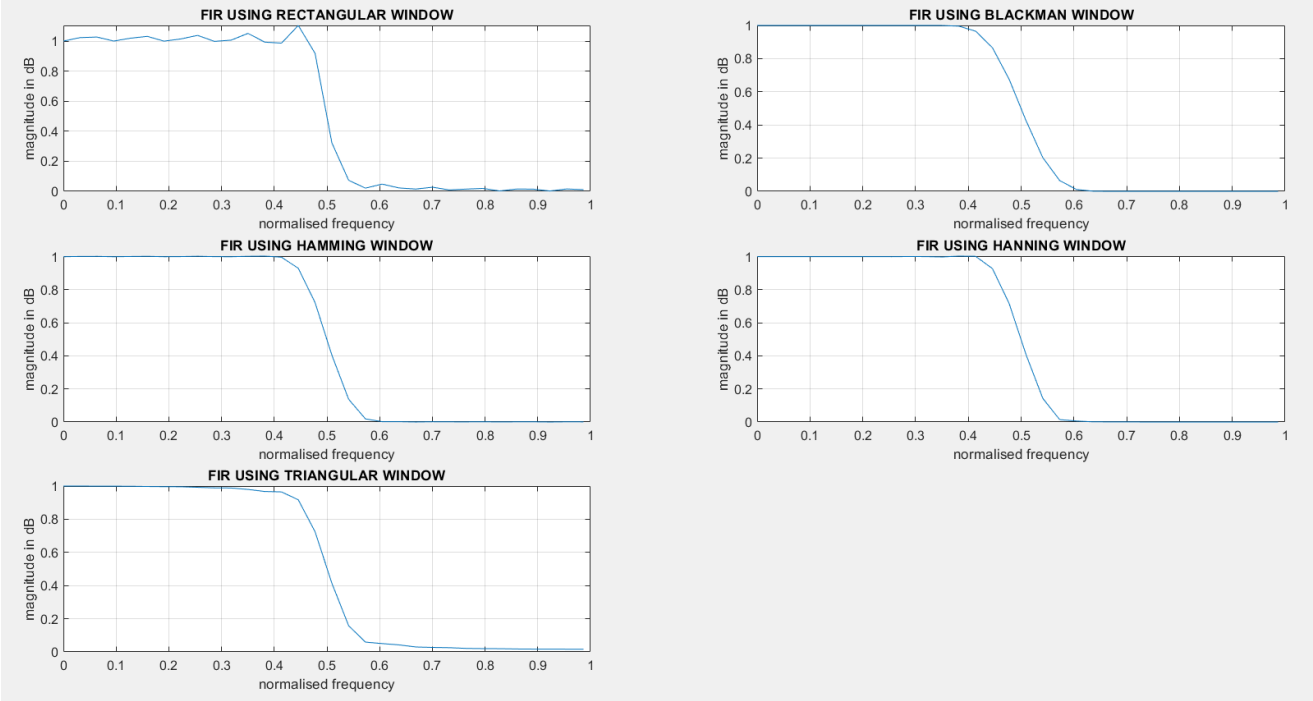
HIGH PASS FILTER







LOW PASS FILTER



FIR filters are analysed with different window function using MATLAB.

EXPERIMENT NO: 8

IIR FILTER DESIGN

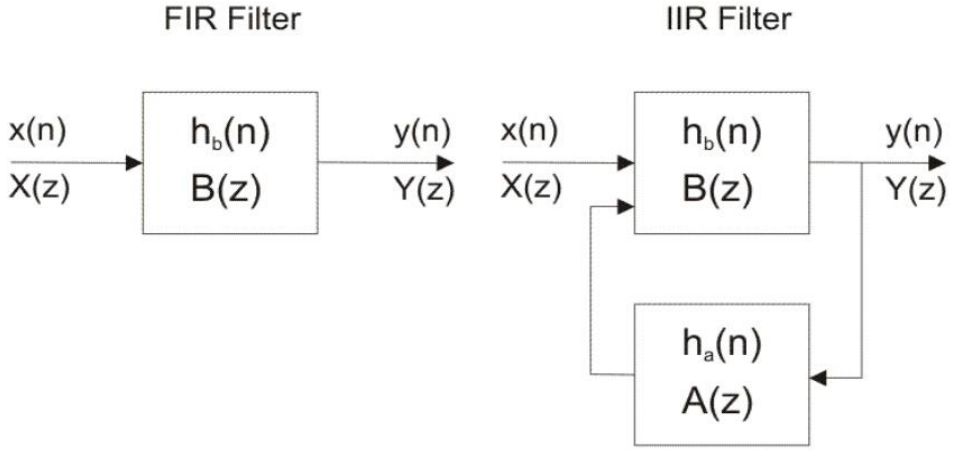
1. AIM

To design an IIR filter using impulse invariance method and bilinear transformation.

2. THEORY

IIR filters are digital filters with infinite impulse response. Unlike FIR filters, they have the feedback (a recursive part of a filter) and are known as recursive digital filters. For this reason IIR filters have much better frequency response than FIR filters of the same order. Unlike FIR filters, their phase characteristic is not linear which can cause a problem to the systems which need phase linearity. For this reason, it is not preferable to use IIR filters in digital signal processing when the phase is of the essence.

Otherwise, when the linear phase characteristic is not important, the use of IIR



filters is an excellent solution.There is one problem known as a potential instability that is typical of IIR filters only. FIR filters do not have such a problem as they do not have the feedback.

For this reason, it is always necessary to check after the design process whether the resulting IIR filter is stable or not.

Methods for IIR design

A. Impulse Invariance Transformation Method

Traditionally, IIR filter design is based on the concept of transforming a continuous- time, or analog, design into the discrete - time domain. In impulse invariance method, the impulse response of the digital filter is the samples of the impulse response of the continuous-time filter:

h[n]=T h[nT] ,where T represents the sampling interval.

This method is based on the concept of mapping each s-plane pole of the continuous time filter to a corresponding z-plane pole using the substitution (1-e^pktsz^-1) for (s + pk) in H(s). This can be achieved by several different means. Partial fraction expansion of H (s) and substitution of (1-e^ktsz^-1) for (s + pk) is a direct method to do.

B. Bilinear Transform Method of Digital Filter Implementation.

The bilinear transform method of converting an analog filter design to discrete time is relatively straightforward, often involving less algebraic manipulation than the impulse invariant method. It is achieved by making the substitution:

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

In H (s), where T is the sampling period of the digital filter; that is,

$$H(z) = H(s)\big|_{s=2(z-1)/T(z+1)}$$

$$\omega_A = \frac{\omega_D}{\pi T_S} \quad \text{or} \quad \omega_A = \frac{2}{T_S}\tan\left(\frac{\omega_D}{2}\right)$$

The concept behind the bilinear transform is that of compressing the frequency response of an analog filters design such that its response over the entire range of frequencies from zero to infinity is mapped into the frequency range zero to half the sampling frequency of the digital filter. This may be represented as a result of the frequency warping inherent in the bilinear transform, the cutoff frequency of the discrete-time filter obtained is not equal to the cutoff frequency of the analog filter. A technique called pre-warping the prototype analog design (used by default in the

MATLAB filter design and analysis tool fdatool) can be used in such a way that the bilinear transform maps an analog frequency wA=wc in the range 0 to ws/2 , to exactly the same digital frequency the same digital frequency wD=wc. This technique is based on the selection of T according to T= 2 tan(πwc/ ws) / wc.

Chebyshev Filter

Chebyshev filters are nothing but analog or digital filters. These filters have a steeper roll off & type-1 filter (more pass band ripple) or type-2 filter (stop band ripple) than Butterworth filters. The property of this filter is, it reduces the error between the characteristic of the actual and idealized filter.

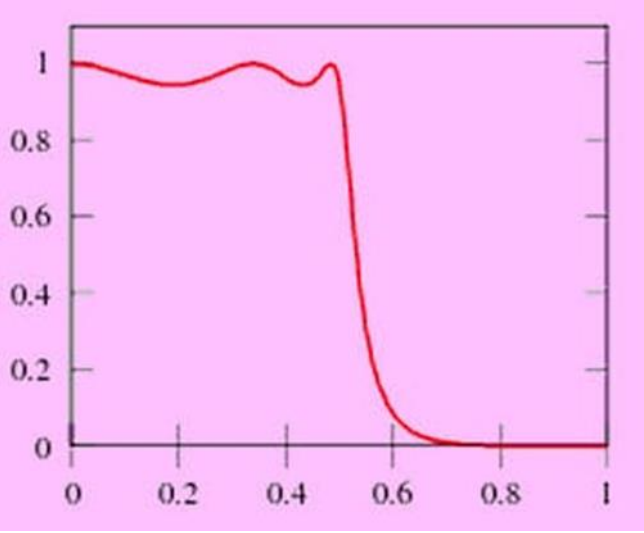
Type-I Chebyshev Filters

This type of filter is the basic type of Chebyshev filter. The amplitude or the gain response is an angular frequency function of the nth order of the LPF (low pass filter) is equal to the total value of the transfer function Hn (jω)

$$Gn(w)=|Hn\ (j\omega)|=1\sqrt{}(1+\epsilon 2Tn2()\ \omega/\omega o)$$

ωo= cutoff frequency

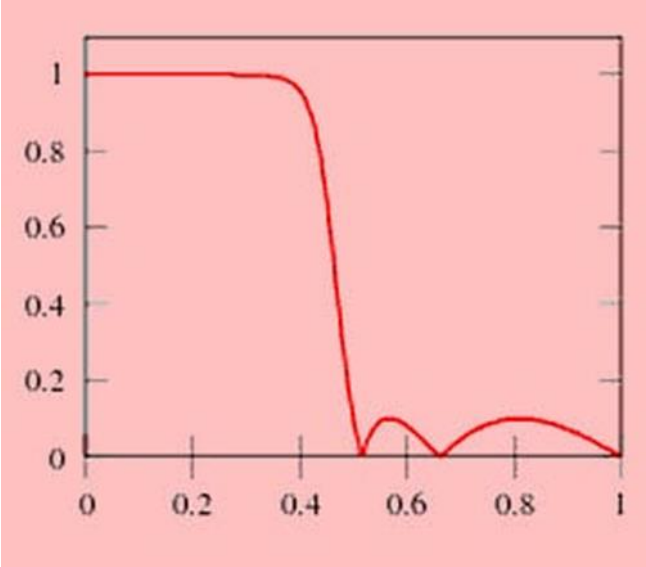
Tn= Chebyshev polynomial of the nth order



Type-II Chebyshev Filter

The type II Chebyshev filter is also known as an inverse filter, this type of filter is less common. Because, it doesn't roll off and needs various components. It has no ripple in the passband, but it has equiripple in the stopband. The gain of the type II Chebyshev filter is

$$G_n(\omega, \omega_0) = \frac{1}{\sqrt{1 + \frac{1}{\varepsilon^2 T_n^2(\omega_0/\omega)}}}.$$



QUESTION

Do the Examples 1 to 6 given in MATLAB.
Plot Magnitude and frequency spectrum
Write down the meaning and operation performed by each command.

EXAMPLE 1: Design a lowpass digital Butterworth filter to satisfy specifications Using impulse invariance method:

Passband edge: ωp = 0.25π rad,
Passband ripple: Ap = 1 dB,
Stopband edge: ωs = 0.4π rad,
Stopband attenuation: As = 30 dB.

EXAMPLE 2: Design a lowpass digital Butterworth filter to satisfy specifications Using bilinear transformation:

Passband edge: ωp = 0.25π rad,
Passband ripple: Ap = 1 dB,
Stopband edge: ωs = 0.4π rad,
Stopband attenuation: As = 30 dB

EXAMPLE 3: Design a lowpass digital Chebyshev II filter to satisfy specifications Using impulse invariance method :

Passband edge: ωp = 0.25π rad,
Passband ripple: Ap = 1 dB,
Stopband edge: ωs = 0.4π rad,
Stopband attenuation: As = 30 dB.

EXAMPLE 4: Design a lowpass digital Butterworth filter to satisfy specifications Using bilinear transformation:

Passband edge: ωp = 0.25π rad,
Passband ripple: Ap = 1 dB,
Stopband edge: ωs = 0.4π rad,
Stopband attenuation: As = 30 dB.

EXAMPLE 5: Design a lowpass digital Chebyshev I filter to satisfy specifications Using bilinear transformation :

Passband edge: ωp = 0.25π rad,
Passband ripple: Ap = 1 dB,
Stopband edge: ωs = 0.4π rad,
Stopband attenuation: As = 30 dB.

EXAMPLE 6: Design a lowpass digital Chebyshev I filter to satisfy specifications Using bilinear transformation:

Passband edge: ωp = 0.25π rad,
Passband ripple: Ap = 1 dB,
Stopband edge: ωs = 0.4π rad,

3. PROGRAM:

EXAMPLE 1

```
%lowpass digital Butterworth filter using impulse
invariance %transformation
Td=0.1;                          %sampling time period
Op=0.25*pi/0.1;                  %passband edge Omega p
Os=0.4*pi/0.1;                   %stopband edge Omega s

%Designing the Butterworth filter
[N,Oc]=buttord(Op,Os,1,30,'s');%N=order of filter,
                               % Oc=cut off frequency of filter
                               %1dB passband ripple
                               %30dB stopband attenuation
[C,D]=butter(N,Oc,'s');        %poles and zeros of filter in s-
```

```
plane

%Transform analog LPF to digital LPF
[B,A]=impinvar(C,D,1/Td);       %poles and zeros of filter
in z-plane
[H,W]=freqz(B,A,N);             %frequency response of the
filter
[G,T]=impz(B,A);                %impulse response of filter

%Plotting Magnitude and frequency response
subplot(3,1,1)
stem(W,H)                       %frequency spectrum
title('Frequency Spectrum')
xlabel('frequency W')
ylabel('H[W]')
subplot(3,1,2)
stem(G,T)                       %impulse response
title("Impulse Response")
xlabel('samples n')
ylabel('h[n]')
subplot(3,1,3)
stem(W,abs(H))                   %magnitude spectrum
title("Magnitude Spectrum")
xlabel('frequency W')
ylabel('|H[W]|')
```

EXAMPLE 2:

```
%lowpass digital Chebyshev I filter using impulse
invariance method
Td=0.1;                          %sampling time period
Op=0.25*pi/0.1;                  %passband edge Omega p
Os=0.4*pi/0.1;                   %stopband edge Omega s

%Designing the Chebyshev type-I filter
[N,Oc]=cheb1ord(Op,Os,1,30,'s');%N=order of filter,
                           % Oc=cut off frequency of filter
                                 %1dB passband ripple
                                 %30dB stopband attenuation
[C,D]=cheby1(N,1,Op,'s');        %poles and zeros of filter
in s-plane

%Transform analog LPF to digital LPF
[B,A]=impinvar(C,D,1/Td);        %poles and zeros of filter
in z-plane
[H,W]=freqz(B,A,N);              %frequency response of the
filter
[G,T]=impz(B,A);                 %impulse response of filter
```

```
%Plotting Magnitude and frequency response
subplot(3,1,1)
stem(W,H)                          %frequency spectrum
title('Frequency Spectrum')
xlabel('frequency W')
ylabel('H[W]')
subplot(3,1,2)
stem(G,T)                          %impulse response
title("Impulse Response")
xlabel('samples n')
ylabel('h[n]')
subplot(3,1,3)
stem(W,abs(H))                     %magnitude spectrum
title("Magnitude Spectrum")
xlabel('frequency W')
ylabel('|H[W]|')
```

EXAMPLE 3:

```
%lowpass digital Chebyshev II filter using impulse
invariance method
Td=0.1;                            %sampling time period
Op=0.25*pi/0.1;                    %passband edge Omega p
Os=0.4*pi/0.1;                     %stopband edge Omega s

%Designing the Chebyshev type-II filter
[N,Oc]=cheb2ord(Op,Os,1,30,'s');%N=order of filter,
                           % Oc=cut off frequency of filter
                                   %1dB passband ripple
                                   %30dB stopband attenuation
[C,D]=cheby2(N,30,Os,'s');          %poles and zeros of filter
in s-plane

%Transform analog LPF to digital LPF
[B,A]=impinvar(C,D,1/Td);          %poles and zeros of filter
in z-plane
[H,W]=freqz(B,A,N);                %frequency response of the
filter
[G,T]=impz(B,A);                   %impulse response of filter

%Plotting Magnitude and frequency response
subplot(3,1,1)
stem(W,H)                          %frequency spectrum
title('Frequency Spectrum')
xlabel('frequency W')
ylabel('H[W]')
subplot(3,1,2)
```

```
stem(G,T)                          %impulse response
title("Impulse Response")
xlabel('samples n')
ylabel('h[n]')
subplot(3,1,3)
stem(W,abs(H))                     %magnitude spectrum
title("Magnitude Spectrum")
xlabel('frequency W')
ylabel('|H[W]|')
```

EXAMPLE 4:

```
 %lowpass digital Butterworth filter using bilinear
transformation
Td=2;                               %sampling time period
Op=tan(0.25*pi/2);                  %passband edge Omega p
Os=tan(0.4*pi/2);                   %stopband edge Omega s

%Designing the Butterworth filter
[N,Oc]=buttord(Op,Os,1,30,'s');    %N=order of filter,
                            % Oc=cut off frequency of filter
                                    %1dB passband ripple
                                   %30dB stopband attenuation
[C,D]=butter(N,Oc,'s');             %poles and zeros of
filter in s-plane

%Transform analog LPF to digital LPF
[B,A]=bilinear(C,D,1/Td);           %poles and zeros of
filter in z-plane
[H,W]=freqz(B,A,N);                 %frequency response of
the filter
[G,T]=impz(B,A);                    %impulse response of
filter

%Plotting Magnitude and frequency response
subplot(3,1,1)
stem(W,H)                           %frequency spectrum
title('Frequency Spectrum')
xlabel('frequency W')
ylabel('H[W]')
subplot(3,1,2)
stem(G,T)                          %impulse response
title("Impulse Response")
xlabel('samples n')
ylabel('h[n]')
subplot(3,1,3)
stem(W,abs(H))                     %magnitude spectrum
title("Magnitude Spectrum")
```

```
xlabel('frequency W')
ylabel('|H[W]|')
```

EXAMPLE 5:

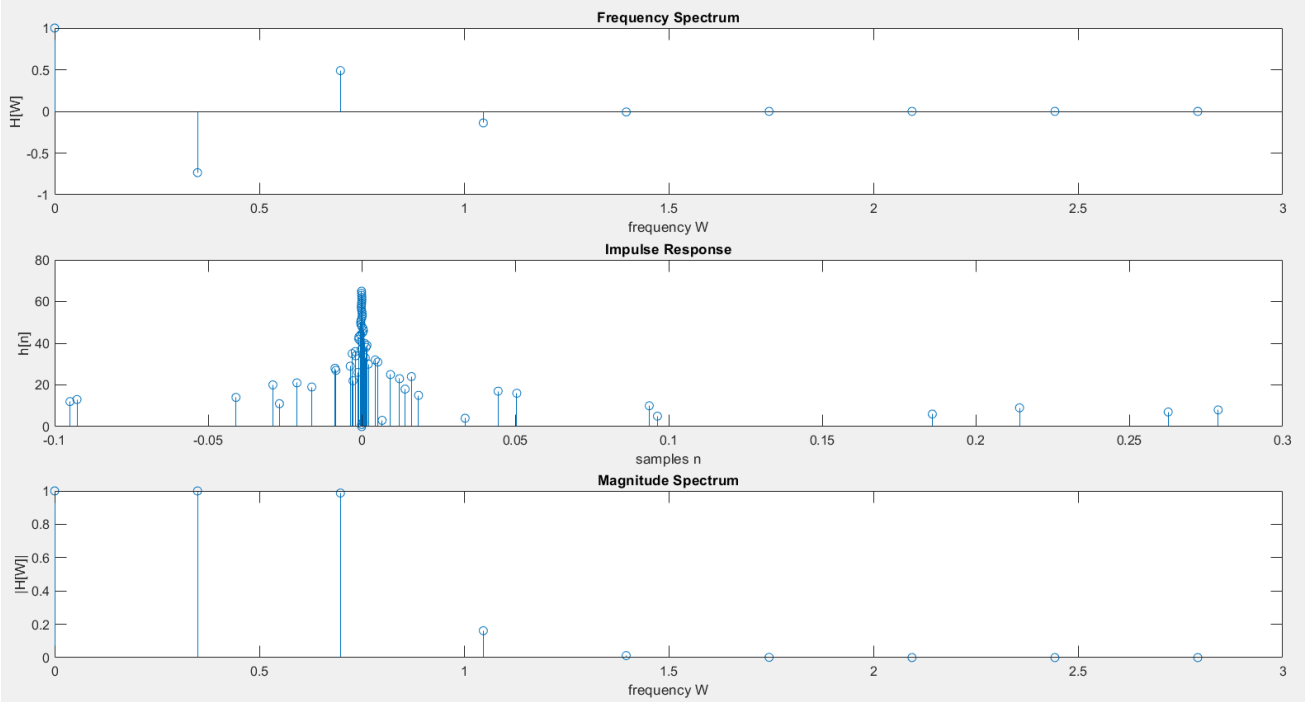
```
%lowpass digital Chebyshev I filter using bilinear
transformation
Td=2;                                   %sampling time period
Op=tan(0.25*pi/2);                      %passband edge Omega p
Os=tan(0.4*pi/2);                       %stopband edge Omega s



%Designing the filter
[N,Oc]=cheb1ord(Op,Os,1,30,'s');  %N=order of filter,
                               % Oc=cut off frequency of filter
                               %1dB passband ripple
                               %30dB stopband attenuation
[C,D]=cheby1(N,1,Op,'s');           %poles and zeros of
filter in s-plane

%Transform analog LPF to digital LPF
[B,A]=bilinear(C,D,1/Td);               %poles and zeros of
filter in z-plane
[H,W]=freqz(B,A,N);                     %frequency response of
the filter
[G,T]=impz(B,A);                        %impulse response of
filter

%Plotting Magnitude and frequency response
subplot(3,1,1)
stem(W,H)                               %frequency spectrum
title('Frequency Spectrum')
xlabel('frequency W')
ylabel('H[W]')
subplot(3,1,2)
stem(G,T)                               %impulse response
title("Impulse Response")
xlabel('samples n')
ylabel('h[n]')
subplot(3,1,3)
stem(W,abs(H))                          %magnitude spectrum
title("Magnitude Spectrum")
xlabel('frequency W')
ylabel('|H[W]|')
```

EXAMPLE 6:

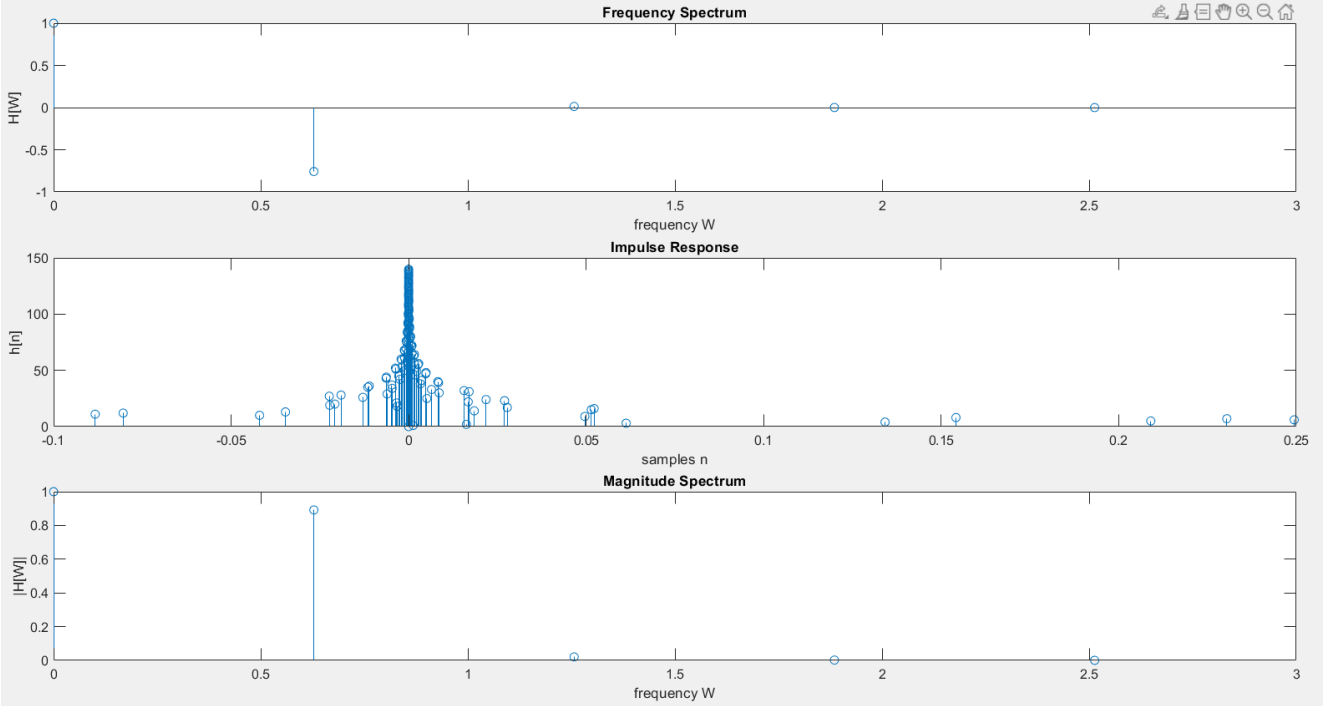
```
%lowpass digital Chebyshev II filter using bilinear
transformation
Td=2;                                   %sampling time period
Op=tan(0.25*pi/2);                      %passband edge Omega p
Os=tan(0.4*pi/2);                       %stopband edge Omega s


%Designing the filter
[N,Oc]=cheb2ord(Op,Os,1,30,'s');  %N=order of filter,
                              % Oc=cut off frequency of filter
                                    %1dB passband ripple
                                  %30dB stopband attenuation
[C,D]=cheby2(N,30,Os,'s');              %poles and zeros of
filter in s-plane

%Transform analog LPF to digital LPF
[B,A]=bilinear(C,D,1/Td);               %poles and zeros of
filter in z-plane
[H,W]=freqz(B,A,N);                     %frequency response of
the filter
[G,T]=impz(B,A);                        %impulse response of
filter


%Plotting Magnitude and frequency response
subplot(3,1,1)
stem(W,H)                               %frequency spectrum
title('Frequency Spectrum')
xlabel('frequency W')
ylabel('H[W]')
subplot(3,1,2)
stem(G,T)                               %impulse response
title("Impulse Response")
xlabel('samples n')
ylabel('h[n]')
subplot(3,1,3)
stem(W,abs(H))                          %magnitude spectrum
title("Magnitude Spectrum")
xlabel('frequency W')
ylabel('|H[W]|')
```

4. RESULT

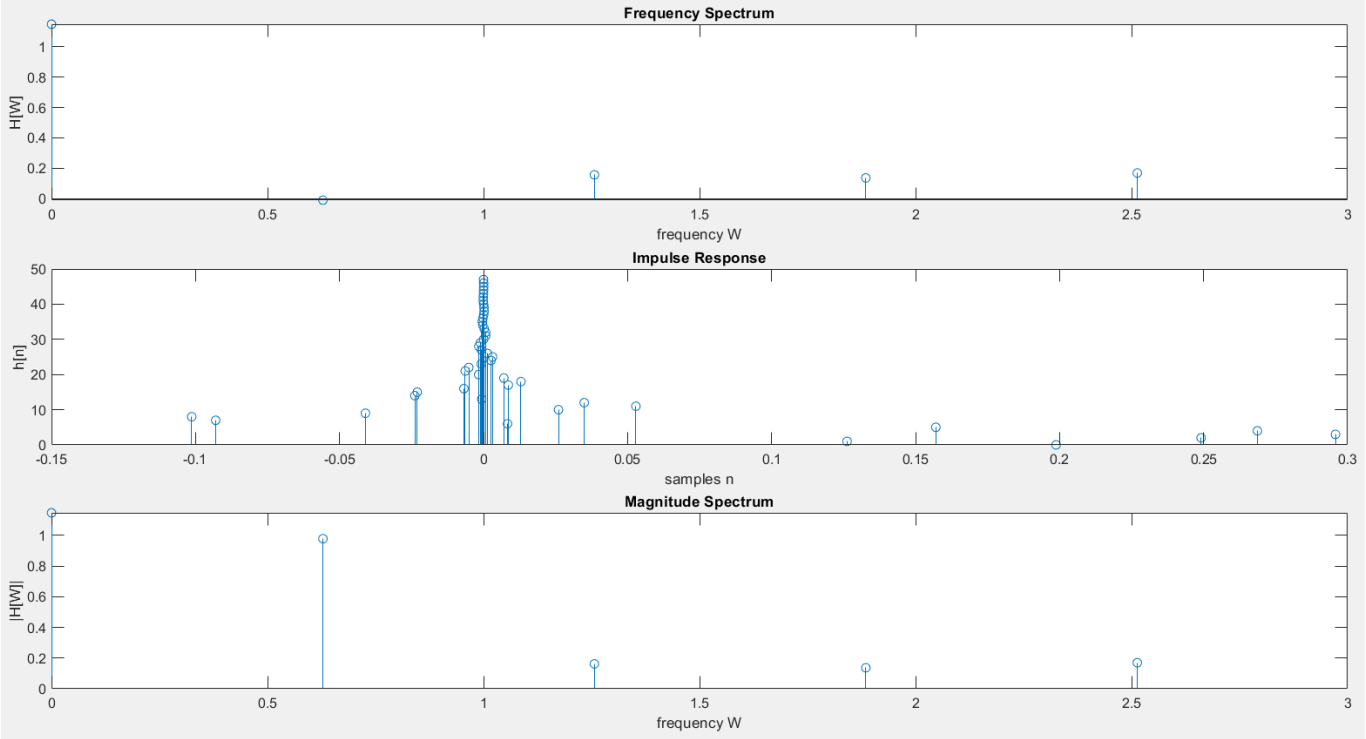
EXAMPLE 1



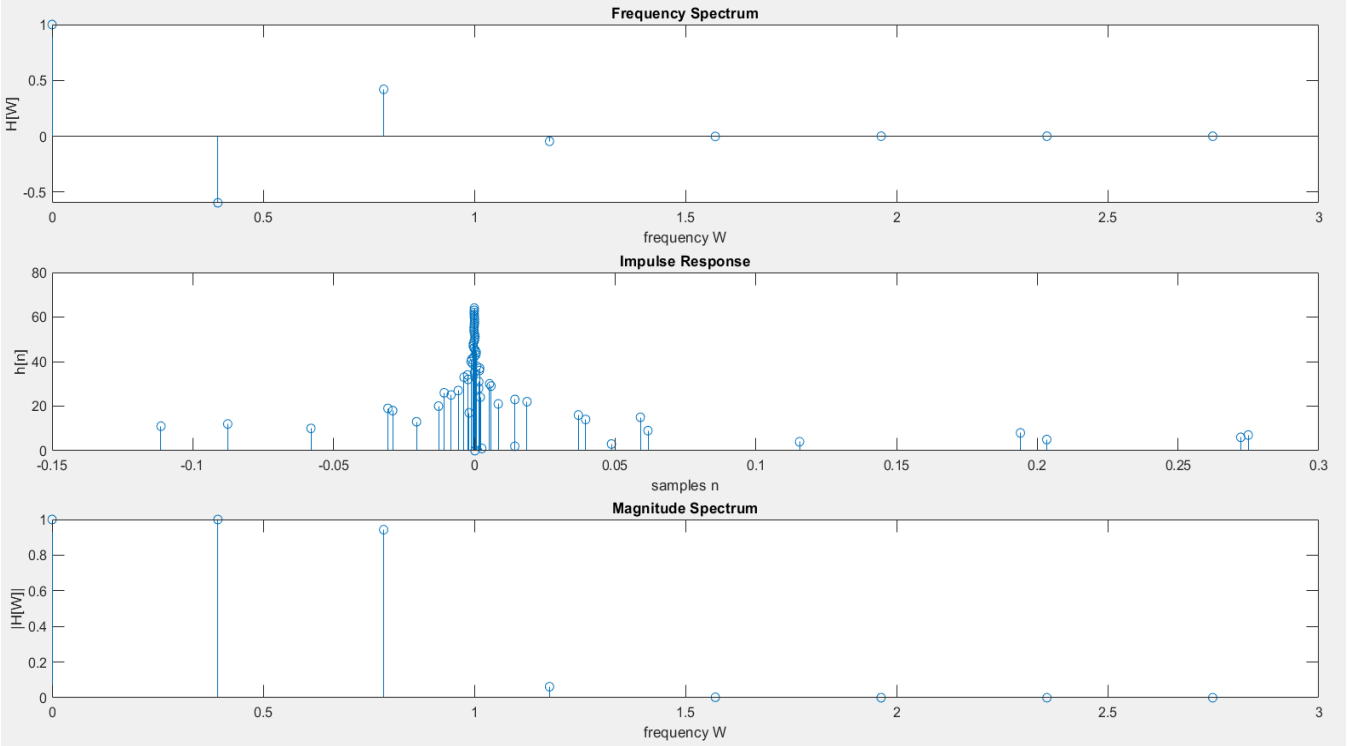
EXAMPLE 2



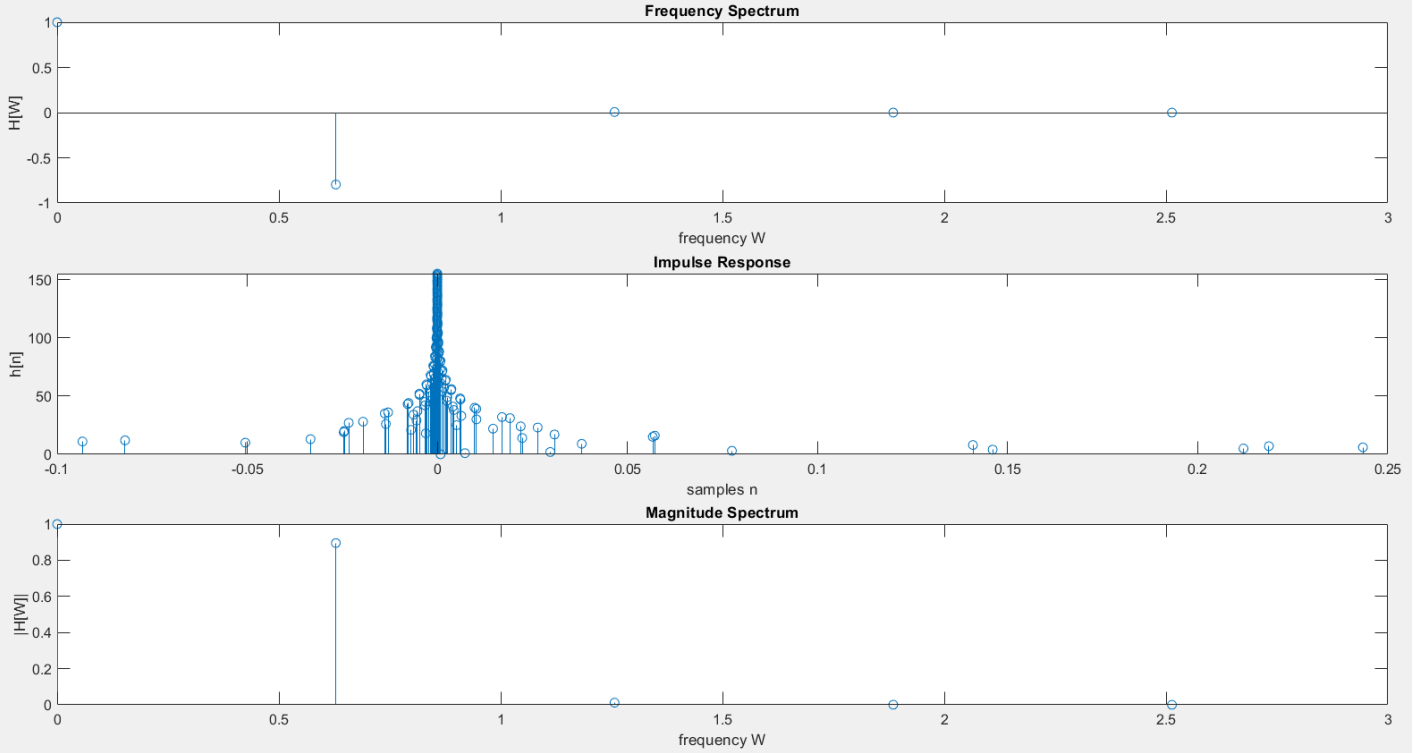
EXAMPLE 3



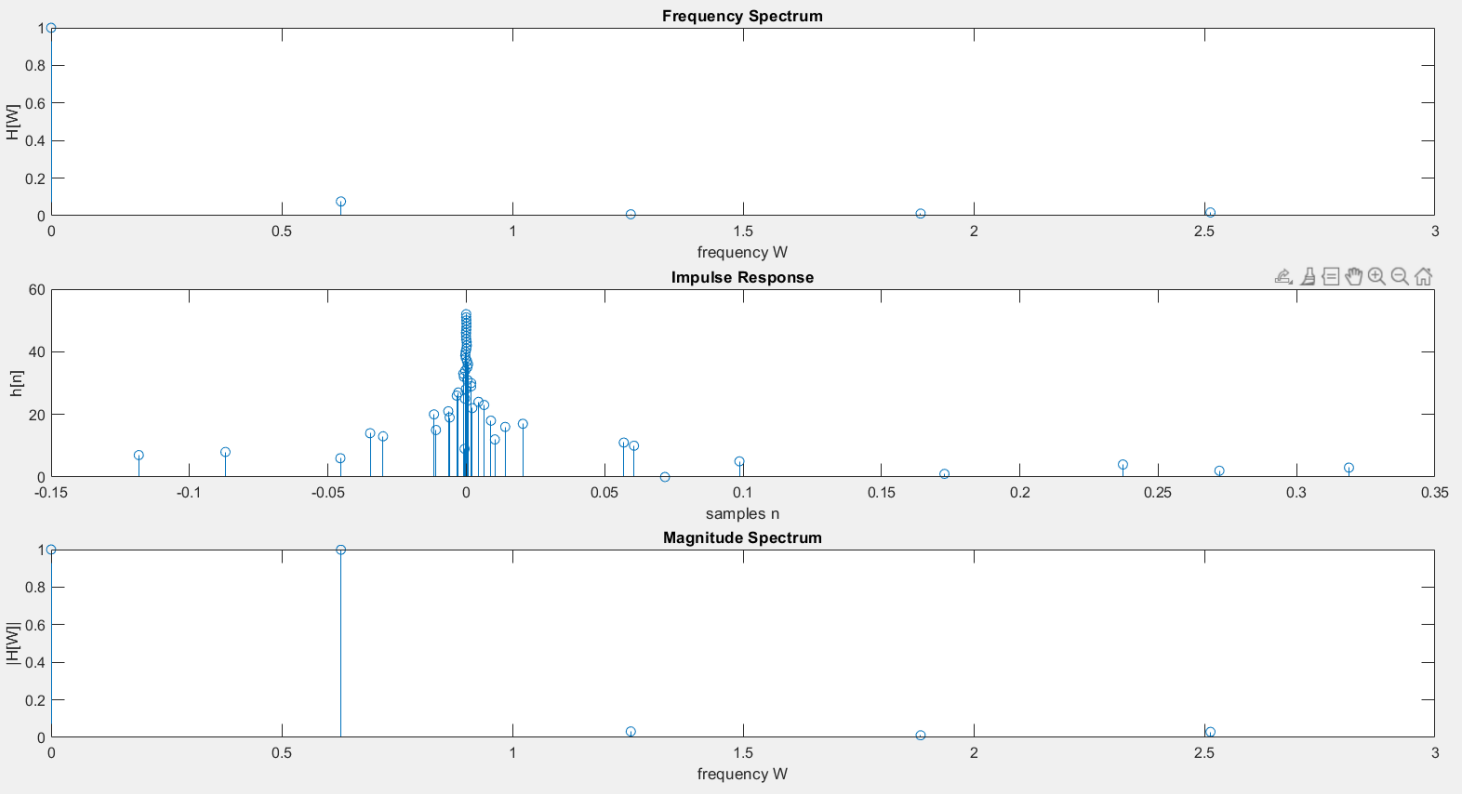
EXAMPLE 4



EXAMPLE 5



EXAMPLE 6



Using MATLAB working of IIR filters were familiarized and waveforms were plotted.

EXPERIMENT NO :9

FILTER DESIGN USING FDATOOL IN MATLAB

1. AIM

To familiar with fdatool box in matlab and to plot the responses of each filters which are under design.

2. THEORY

Filter Designer is a powerful graphical user interface (GUI) in the Signal Processing Toolbox™ for designing and analysing filters.

Filter Designer enables you to quickly design digital FIR or IIR filters by setting filter performance specifications, by importing filters from your MATLA workspace or by adding, moving, or deleting poles and zeros. Filter Designer also provides tools for analysing filters, such as magnitude and phase response plots and pole-zero plots.

- Getting Started

Type filter Designer at the MATLAB command prompt:

The GUI has three main regions:

- The Current Filter Information region
- The Filter Display region and
- The Design panel

The upper half of the GUI displays information on filter specifications and responses for the current filter. The Current Filter Information region, in the upper left, displays filter properties, namely the filter structure, order, number of sections used and whether the filter is stable or not. It also provides access to the Filter manager for working with multiple filters.

The Filter Display region, in the upper right, displays various filter responses, such as, magnitude response, group delay and filter coefficients.
The lower half of the GUI is the interactive portion of Filter Designer. The Design Panel, in the lower half is where you define your filter specifications. It controls what is displayed in the other two upper regions. Other panels can be displayed in the lower half by using the sidebar buttons.

The tool includes Context-sensitive help

- Designing a Filter

To design a low pass filter that passes all frequencies less than or equal to 20% of the Nyquist frequency (half the sampling frequency) and attenuates frequencies greater than or equal to 50% of the Nyquist frequency. They will use an FIR Equiripple filter with these specifications:

Passband attenuation 1 dB

Stopband attenuation 80 dB

A passband frequency 0.2 [Normalized (0 to 1)]

A stopband frequency 0.5 [Normalized (0 to 1)]

- To implement this design, we will use the following specifications:

1. Select Low pass from the dropdown menu under Response Type and Equiripple under FIR Design Method. In general, when you change the Response Type or Design Method, the filter parameters and Filter Display region update automatically.

2. Select Specify order in the Filter Order area and enter 30.

3. The FIR Equiripple filter has a Density Factor option which controls the density of the frequency grid. Increasing the value creates a filter which more closely approximates an ideal equiripple filter, but more time is required as the

computation increases. Leave this value at 20. 4. Select Normalized (0 to 1) in the Units pull down menu in the Frequency Specifications area.

5. Enter 0.2 for wpass and 0.5 for wstop in the Frequency Specifications area.
6. Wpass and Wstop, in the Magnitude Specifications area are positive weights, one per band, used during optimization in the FIR Equiripple filter. Leave these values at 1.
7. After setting the design specifications, click the Design Filter button at the bottom of the GUI to design the filter.

The magnitude response of the filter is displayed in the Filter Analysis area after the coefficients are computed.

QUESTION

Using FDA tool of MATLAB, design FIR & IIR filters. Plot frequency spectrum and pole zero plot.

1. Design a 10th order FIR filter with the frequency response

$$H_d(\omega) = \begin{cases} 1, & -\frac{\pi}{2} \le \omega \le \frac{\pi}{2} \\ 0, & \frac{\pi}{2} \le |\omega| \le \pi \end{cases}$$

using (a) Rectangular (b) Hamming and (c) Hanning window functions. Plot the frequency response of this designed filter. Also find the transfer function of the causal version of this filter.

2. Design a FIR filter with the frequency response

$$H_d(\omega) = \begin{cases} e^{-j3\omega}, & -\frac{\pi}{4} \le \omega \le \frac{\pi}{4} \\ 0, & \frac{\pi}{4} \le |\omega| \le \pi \end{cases}$$

Using (a) Rectangular (b) Hamming and (c) Hanning window functions. Assume order to be 6. Plot the frequency response of this designed filter.

3. Design using rectangular window, a 6th order FIR low pass filter with unit pass band, cutoff frequency of 1kHz and sampling frequency of 5kHz.

4. Design a 10th order FIR filter with the following desired frequency response,

$$H_d(\omega) = \begin{cases} 1, & \frac{\pi}{4} \le |\omega| \le \pi \\ 0, & |\omega| \le \frac{\pi}{4} \end{cases}$$

Use Hamming window and Hanning window. Plot the magnitude spectrum and find the transfer function of its practically implementable form

5. Design a 10th order FIR filter with the following desired frequency response,

$$H_d(\omega) = \begin{cases} 1, & \frac{\pi}{4} \le |\omega| \le \frac{3\pi}{4} \\ 0, & otherwise \end{cases}$$

Use the Hanning window. Plot the magnitude spectrum.

6. Design a 10th order FIR filter with the desired frequency response,

$$H_d(\omega) = \begin{cases} 1, & |\omega| \leq \frac{\pi}{3} \text{ and } |\omega| \geq \frac{2\pi}{3} \\ 0, & \text{otherwise} \end{cases}$$

   Use Hamming and Hanning window. Plot the frequency response.

7. Design a 3rd order Butterworth digital LPF with cutoff frequency w_c=1 radians using impulseinvariance transformation. Assume sampling rate T_s = 1 sec.

8. Design a 2nd order digital Butterworth LPF with cutoff frequency of $0.4\boldsymbol{\pi} rad$ using bilinear transformation.

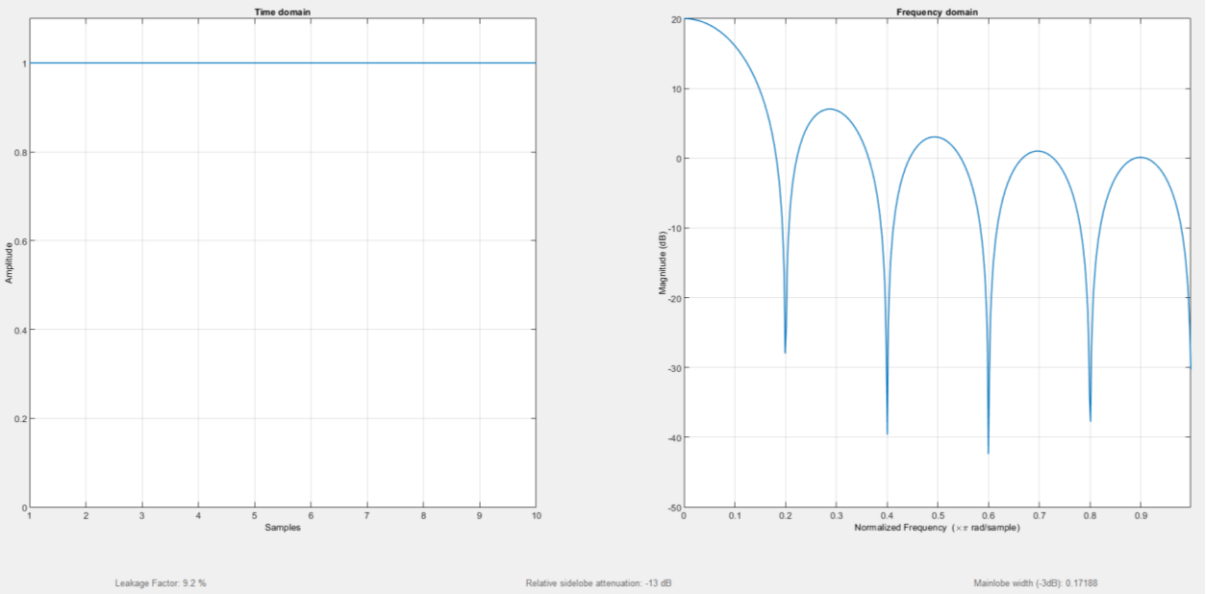
9. Using the bilinear transformation, design a HPF with a monotonically varying frequency spectrum with cutoff frequency of 1000Hz and down 10 dB at 350 Hz. The sampling frequency is 5000 Hz.

10. Design an analog Butterworth filter to meet the following specifications: 20 dB attenuation in the stopband at 200 and 400 Hz.

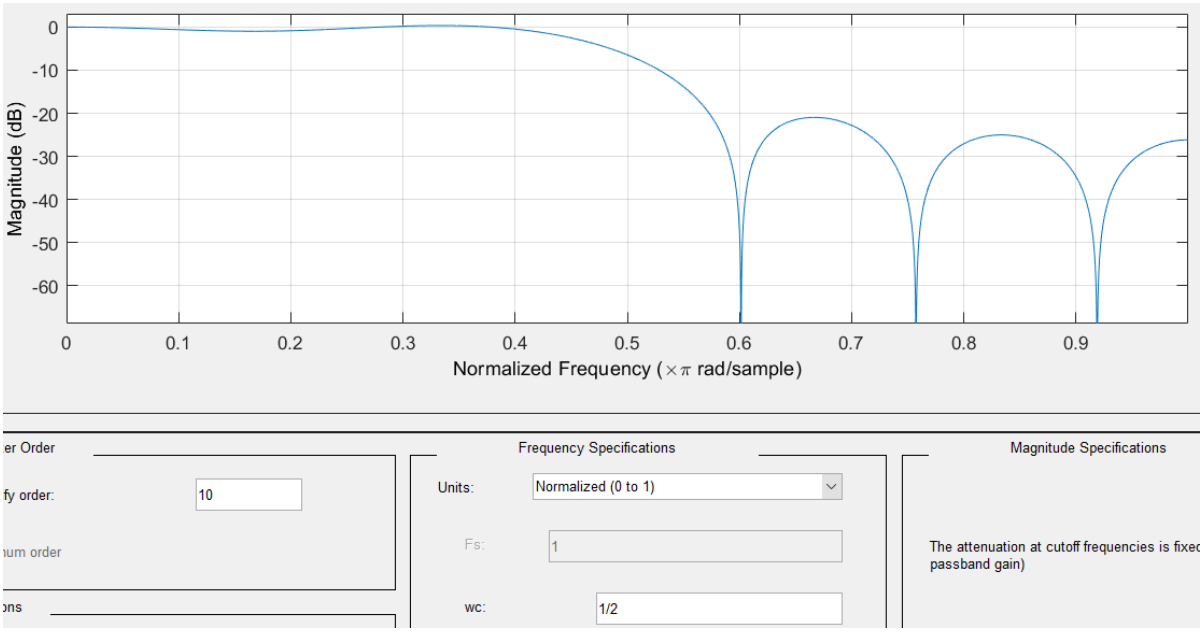
3. RESULT

Q 1) A) RECTANGULAR WINDOW

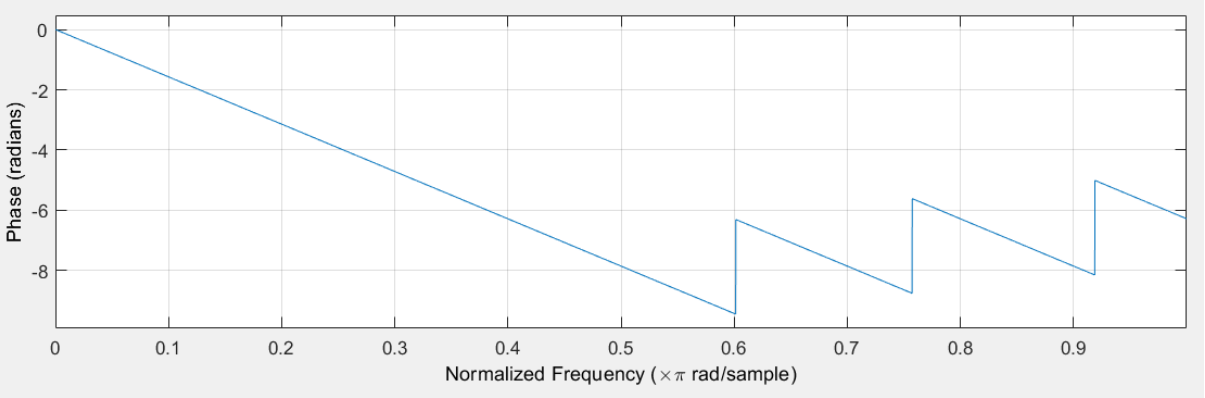
WINDOW RESPONSE



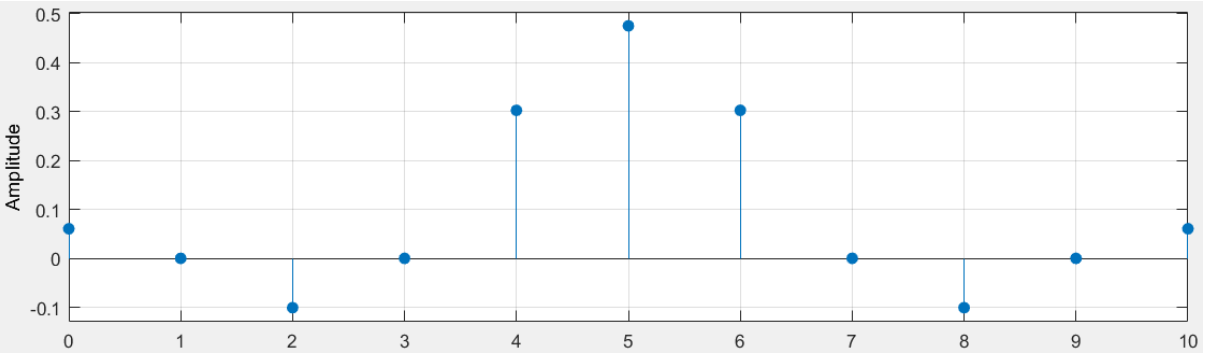
MAGNITUDE RESPONSE



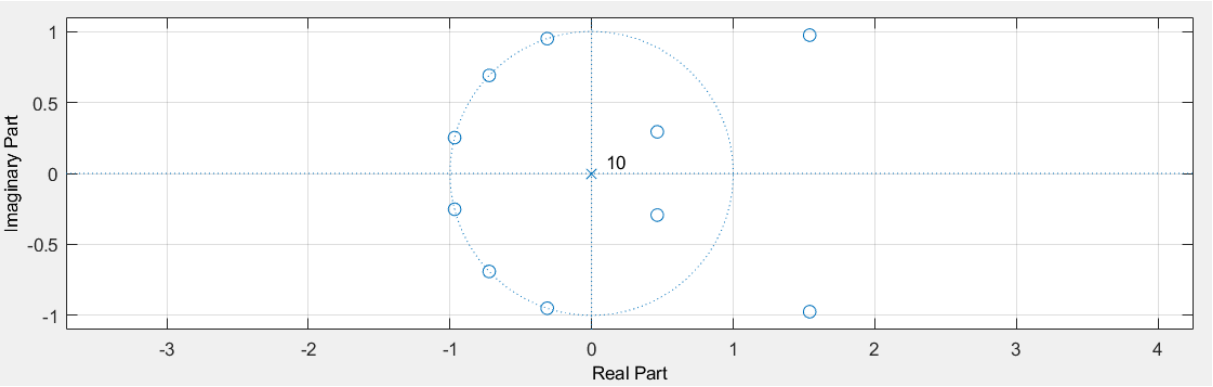
PHASE RESPONSE



IMPULSE RESPONSE

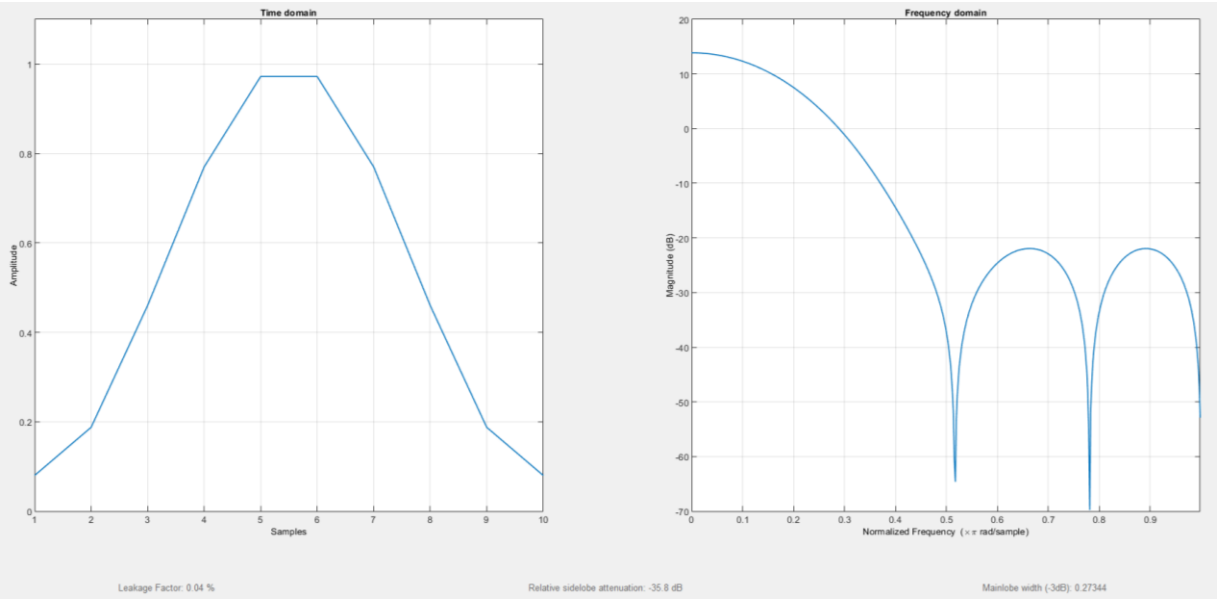


POLE ZERO PLOT

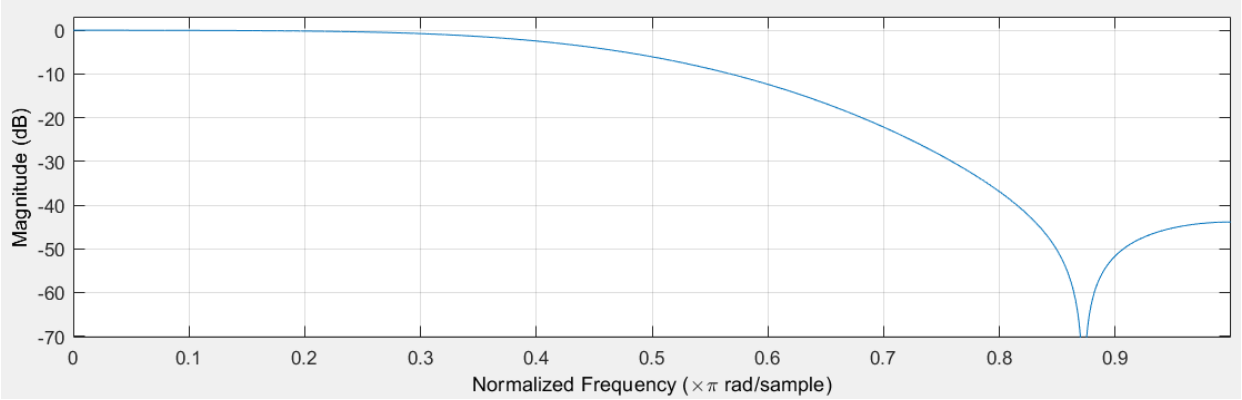


B) HAMMING WINDOW

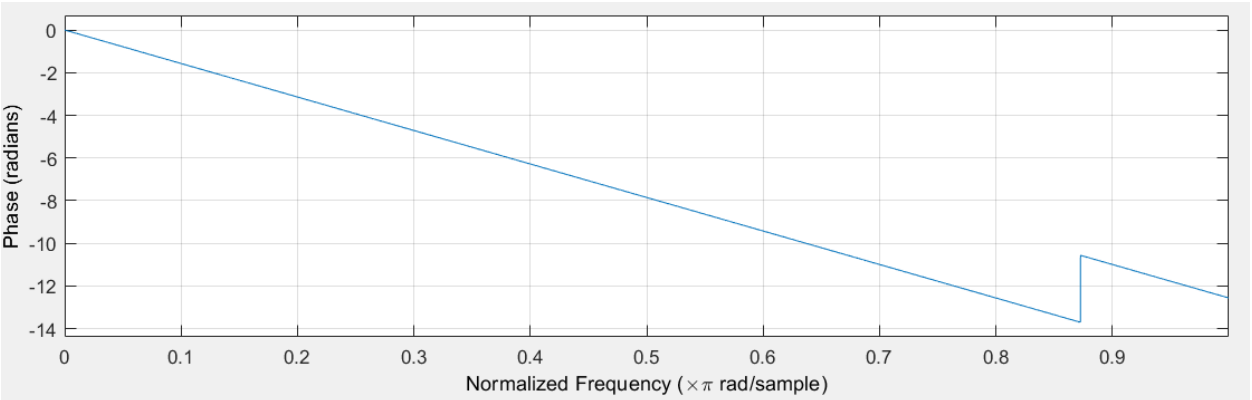
WINDOW RESPONSE



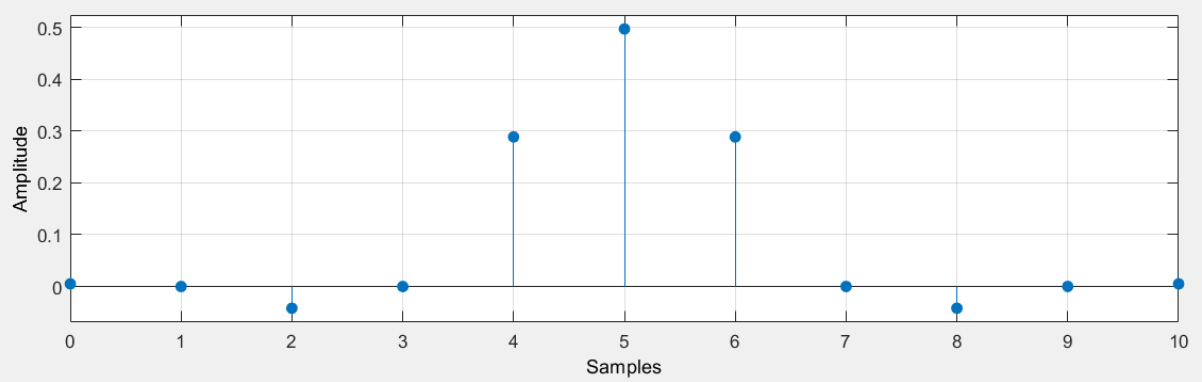
MAGNITUDE RESPONSE



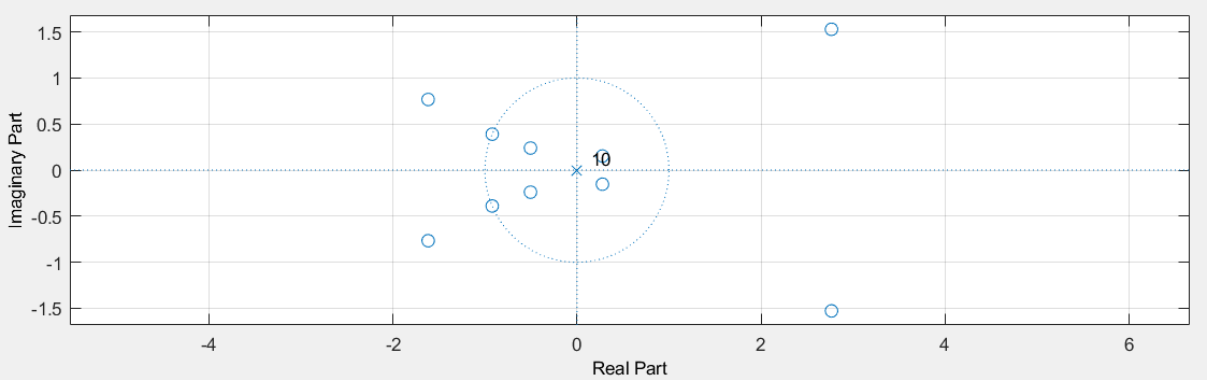
PHASE RESPONSE



IMPULSE RESPONSE

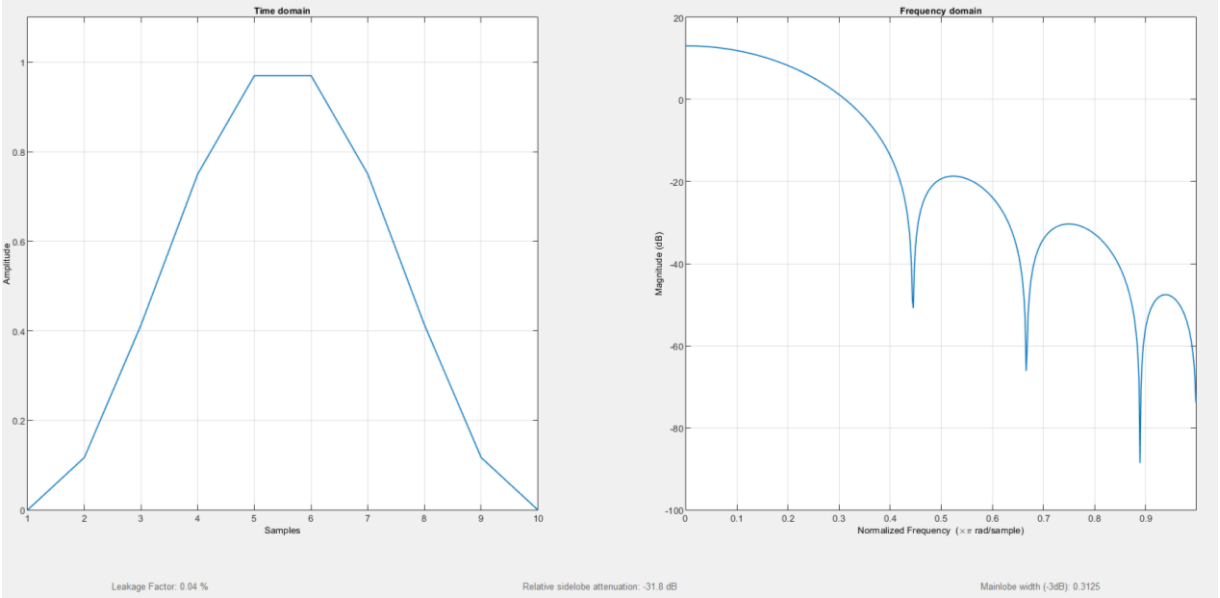


POLE ZERO PLOT

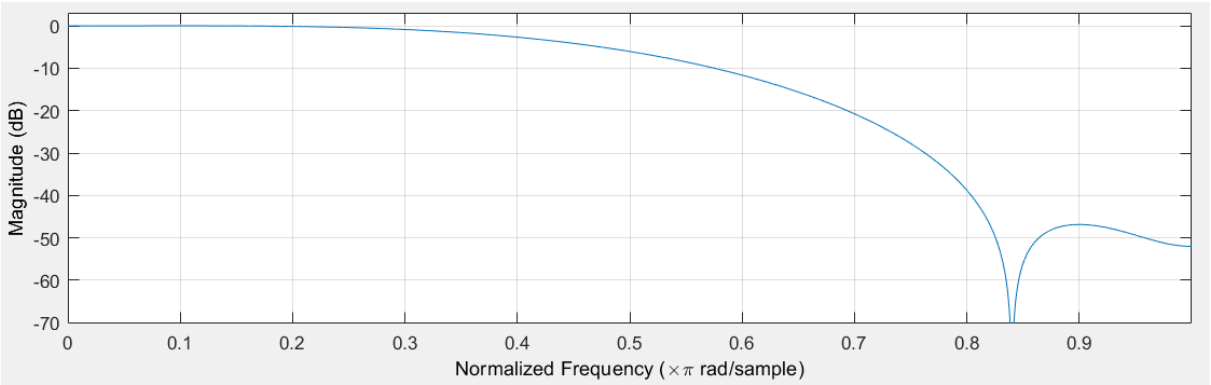


C) HANNING WINDOW

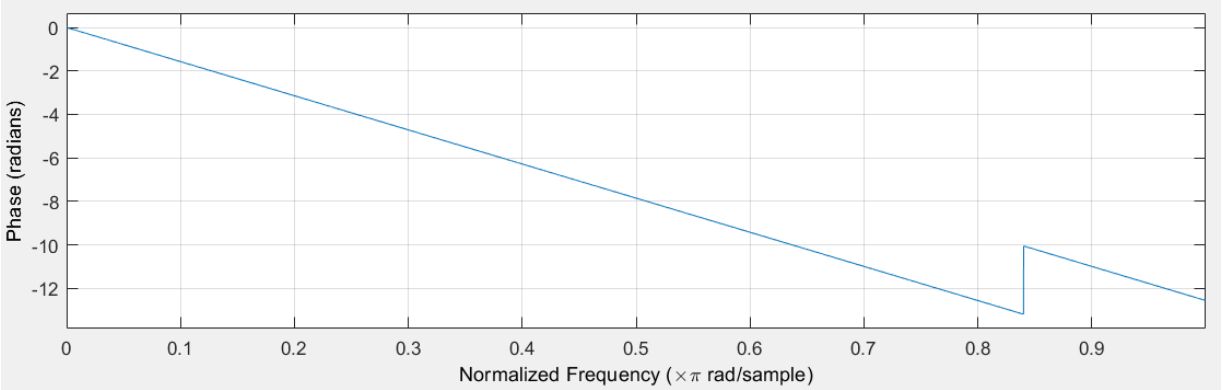
WINDOW RESPONSE



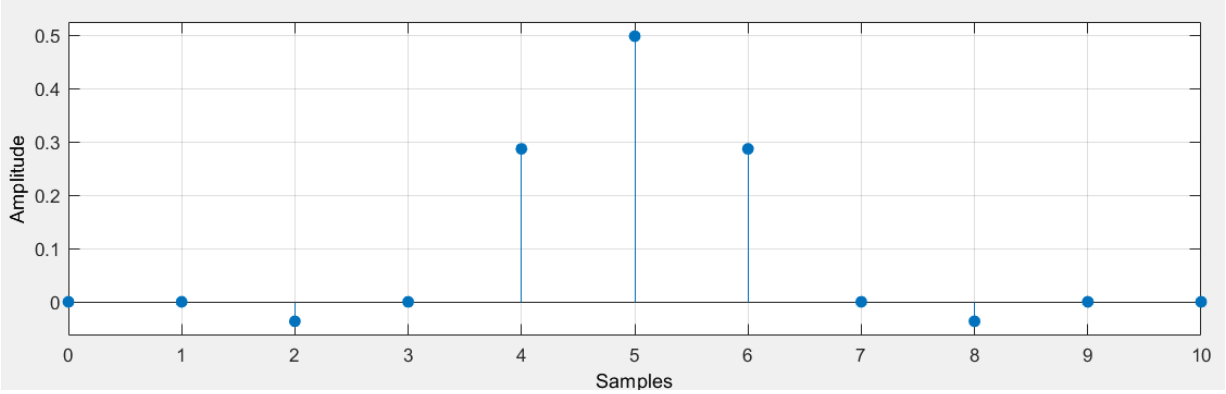
MAGNITUDE RESPONSE



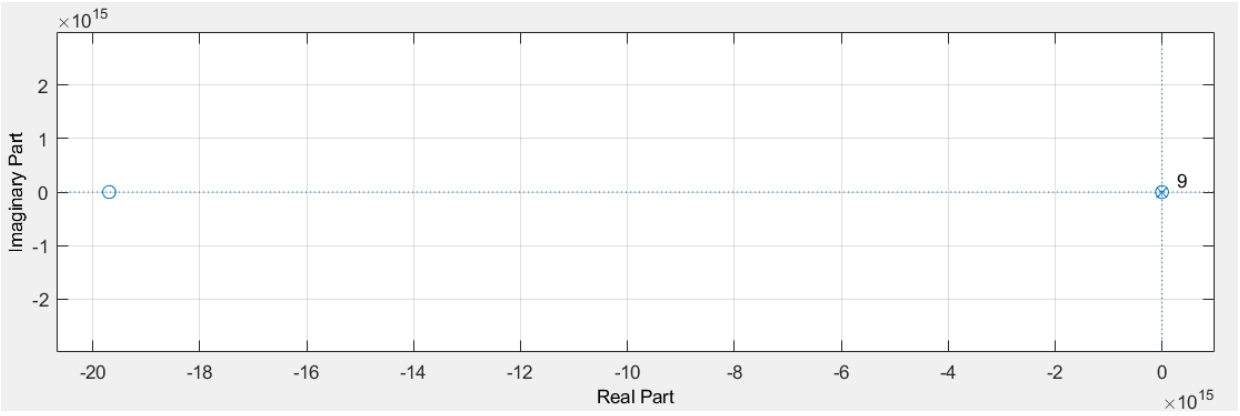
PHASE RESPONSE



IMPULSE RESPONSE



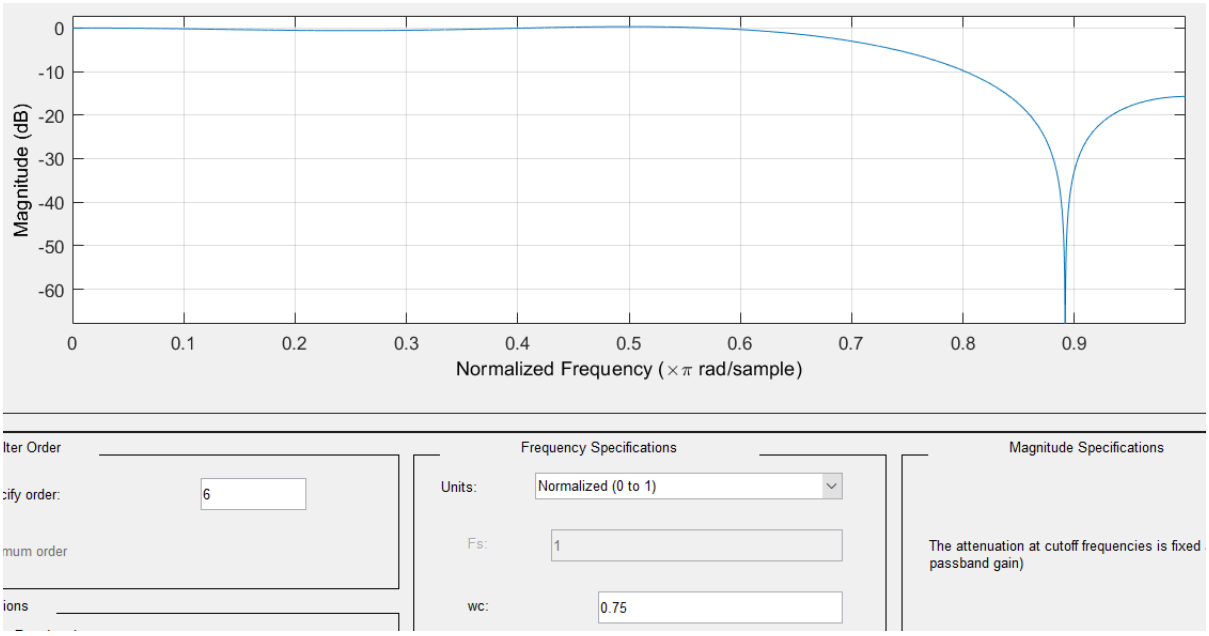
POLE ZERO PLOT



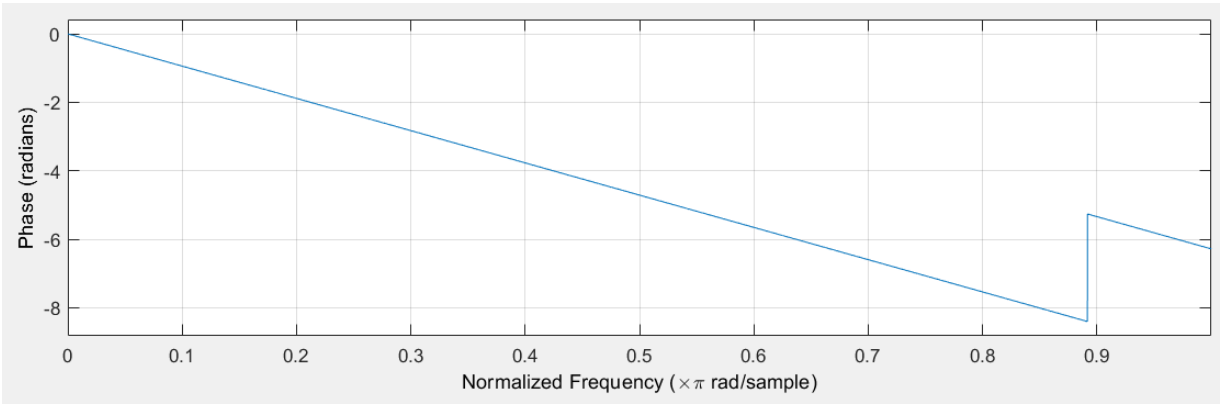
Q2)

A) RECTANGULAR WINDOW

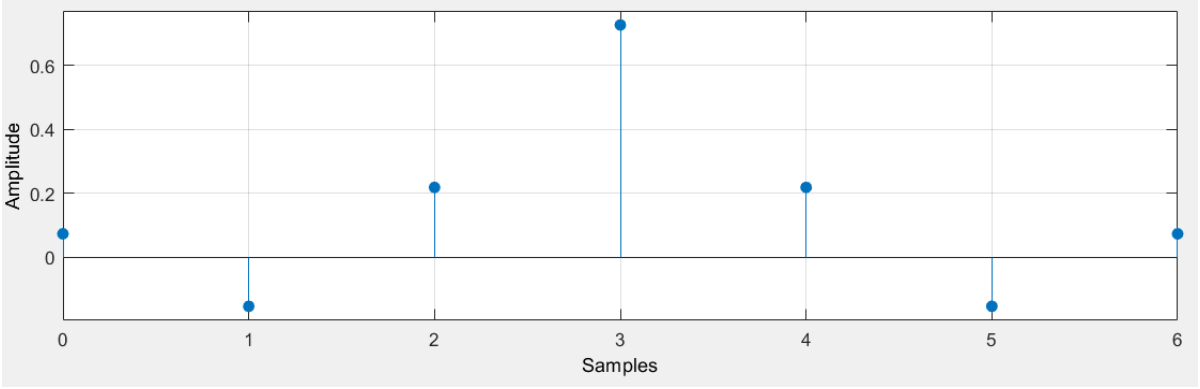
MAGNITUDE RESPONSE



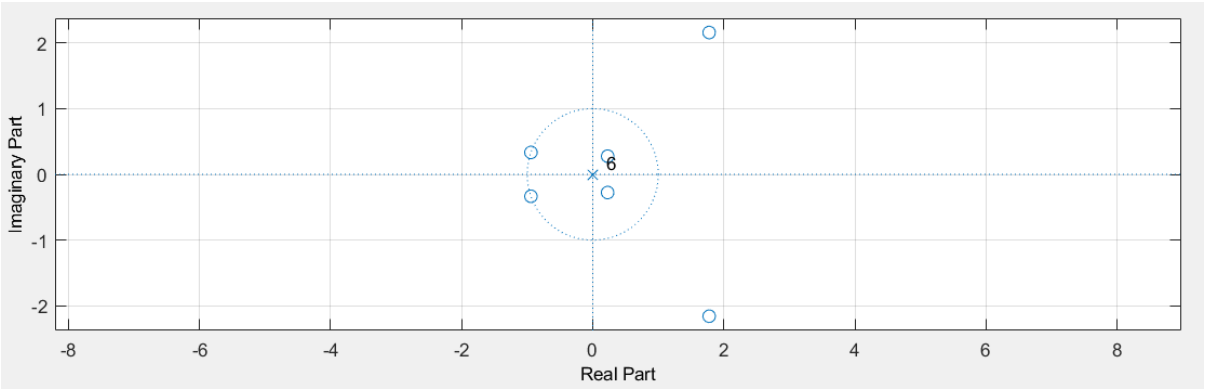
PHASE RESPONSE



IMUPLSE RESPONSE

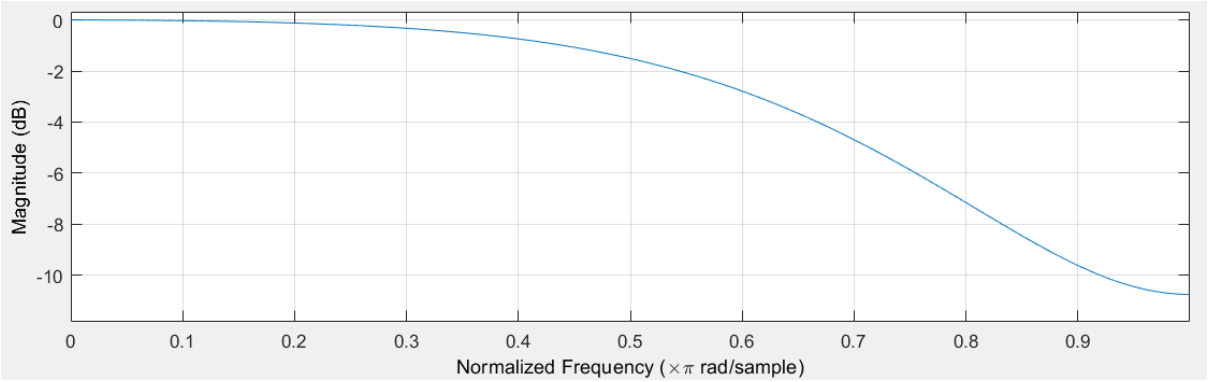


POLE ZERO PLOT

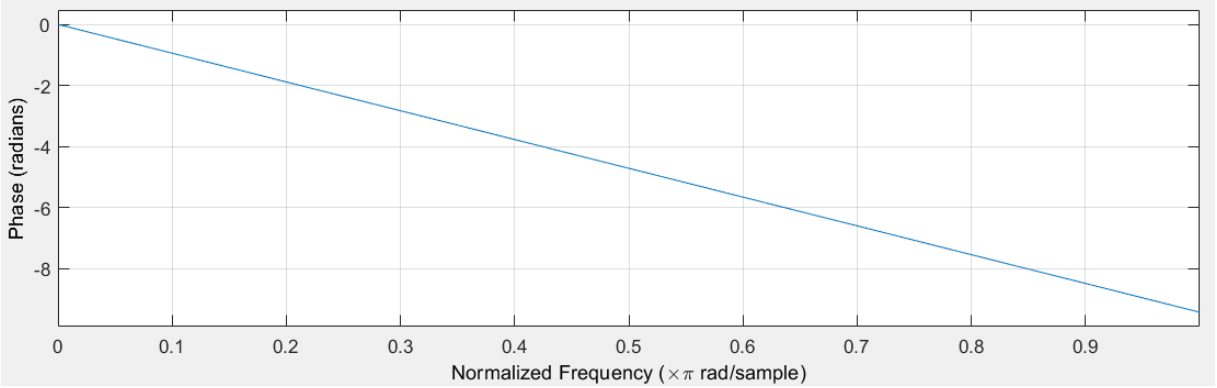


B) HAMMING WINDOW

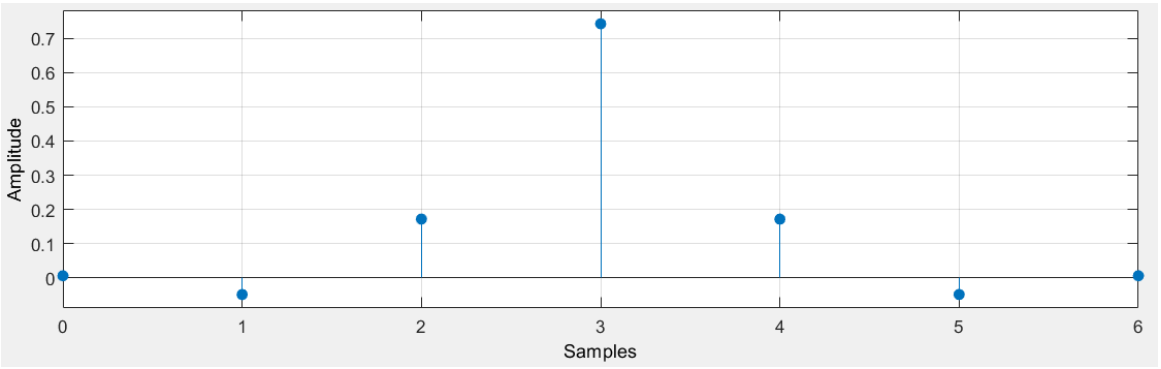
MAGNITUDE RESPONSE



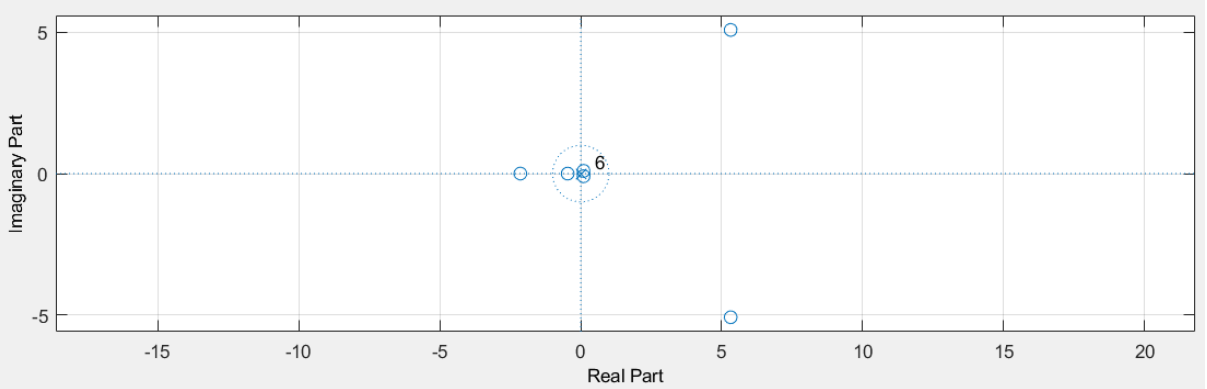
PHASE RESPONSE



IMUPLSE RESPONSE

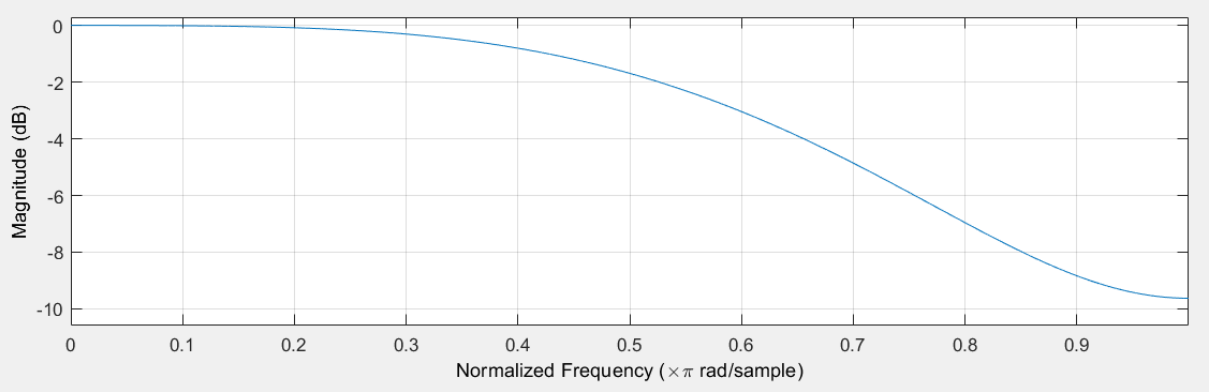


POLE ZERO PLOT

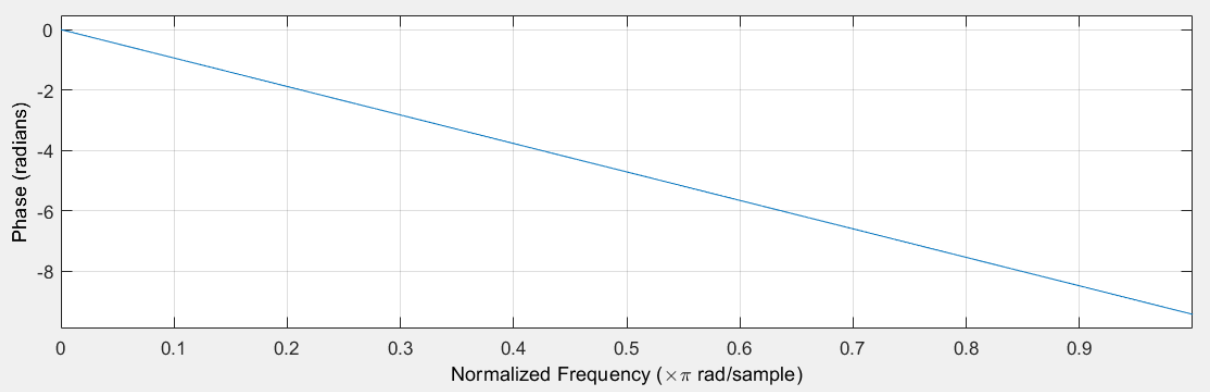


C) HANNING WINDOW

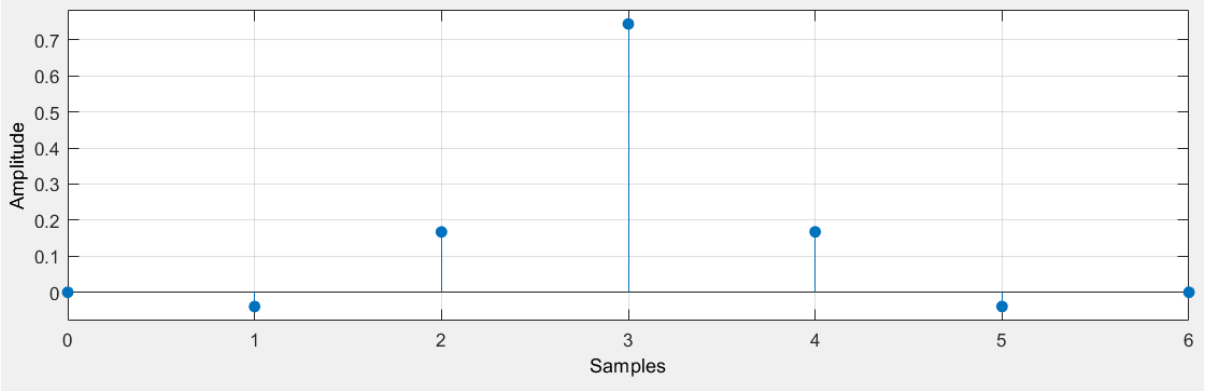
MAGNITUDE RESPONSE



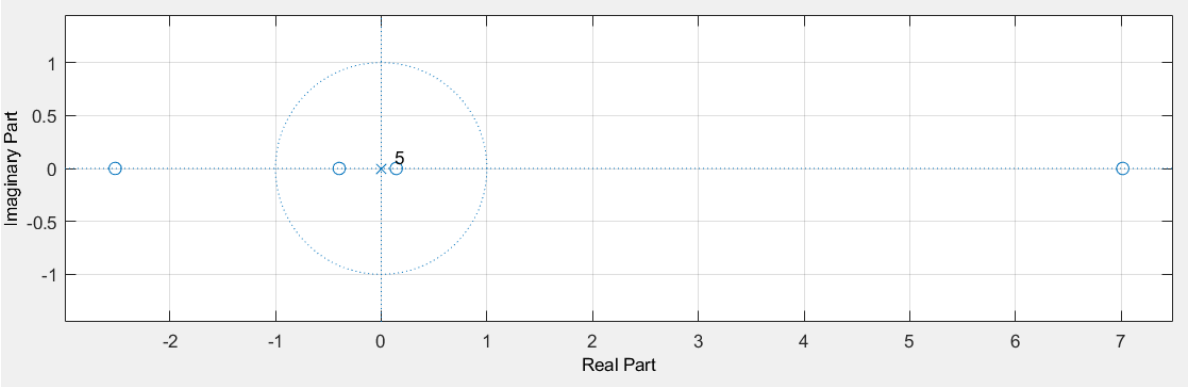
PHASE RESPONSE



IMUPLSE RESPONSE



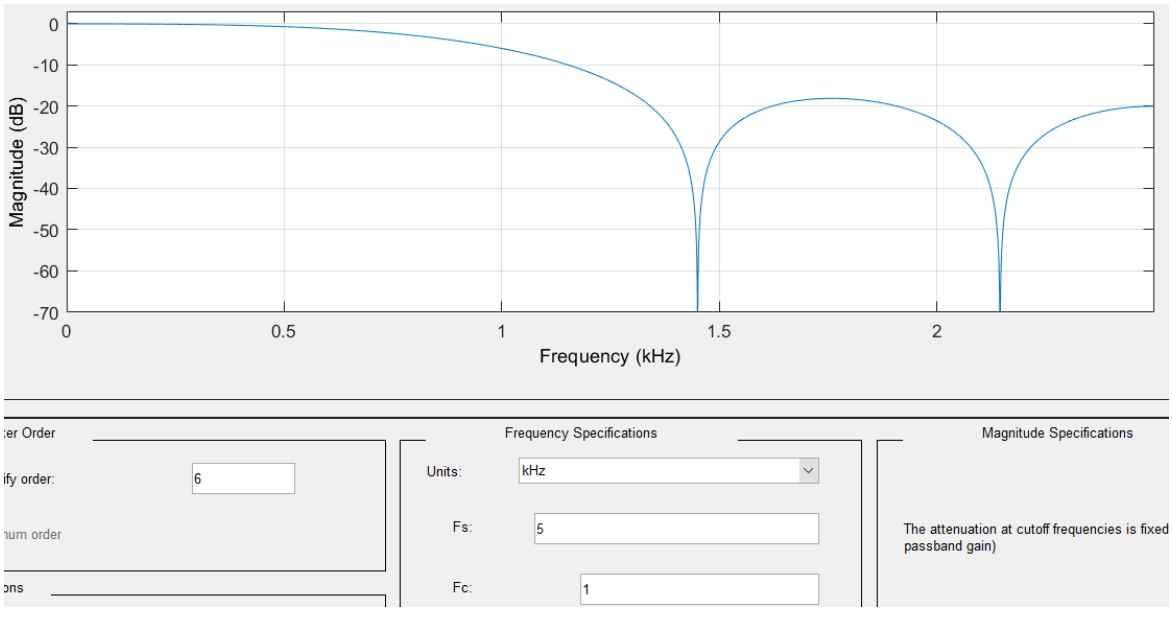
POLE ZERO PLOT



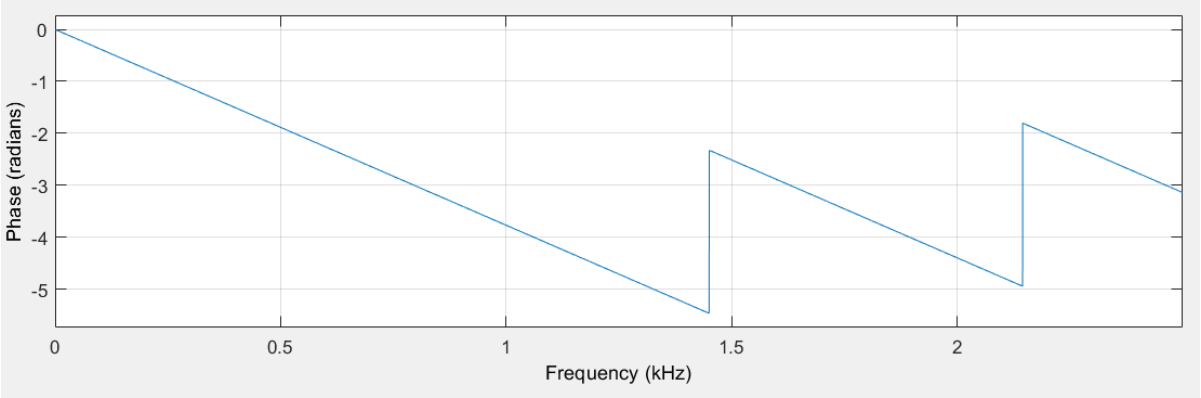
Q3)

A) RECTANGULAR WINDOW

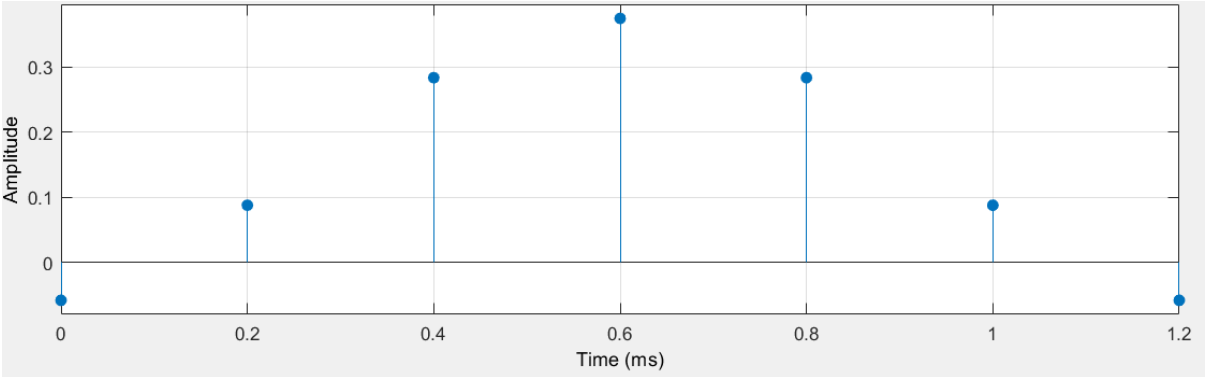
MAGNITUDE RESPONSE



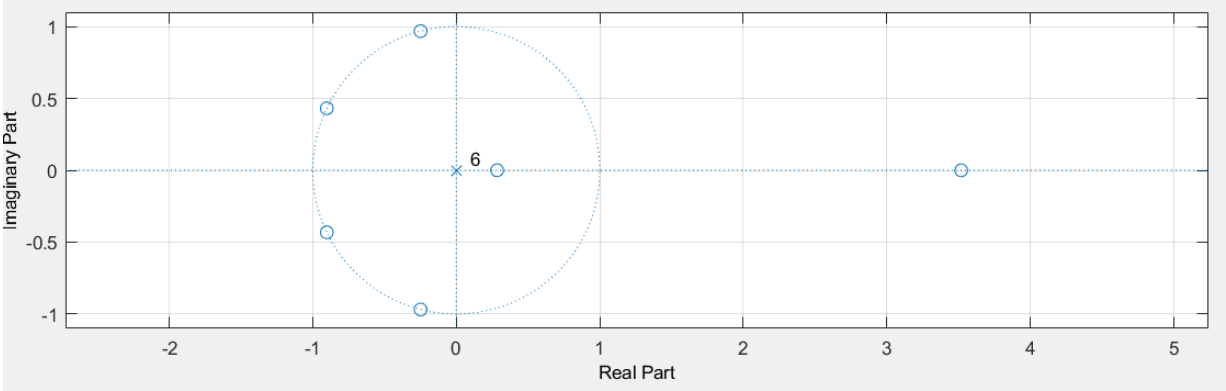
PHASE RESPONSE



IMUPLSE RESPONSE



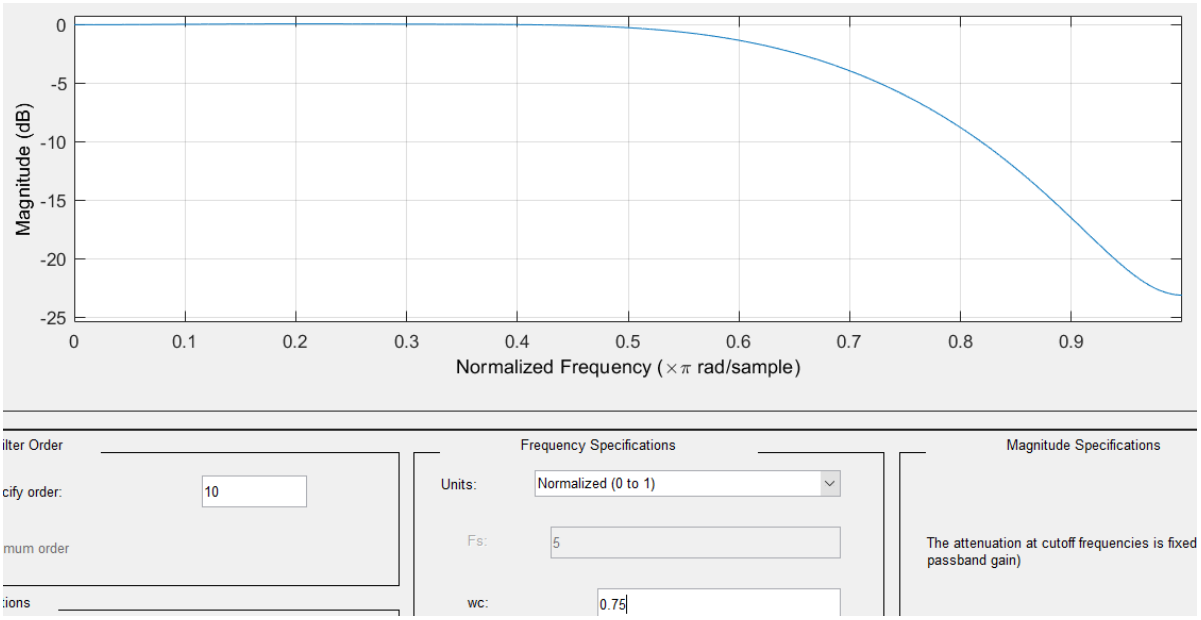
POLE ZERO PLOT



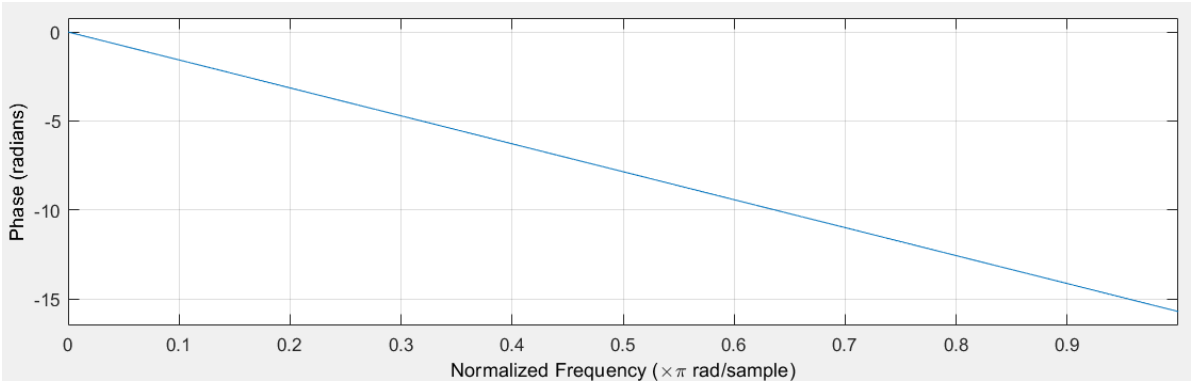
Q4

A) HAMMING WINDOW

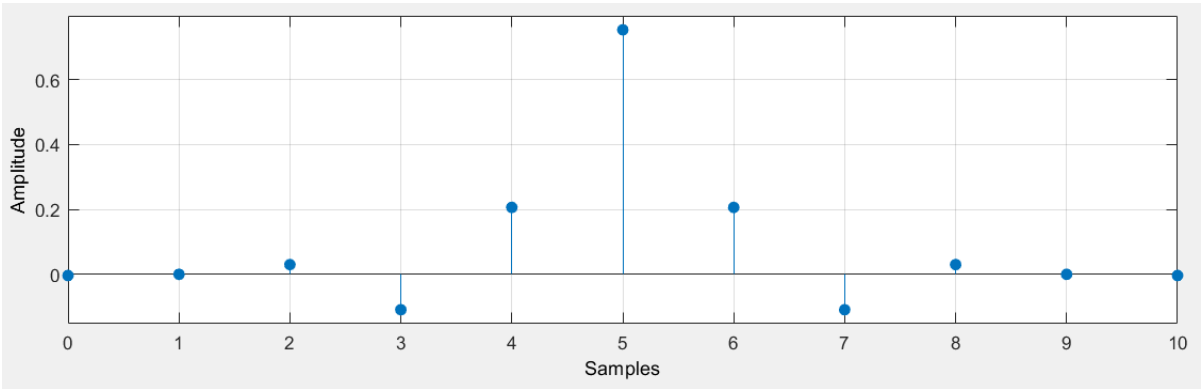
MAGNITUDE RESPONSE



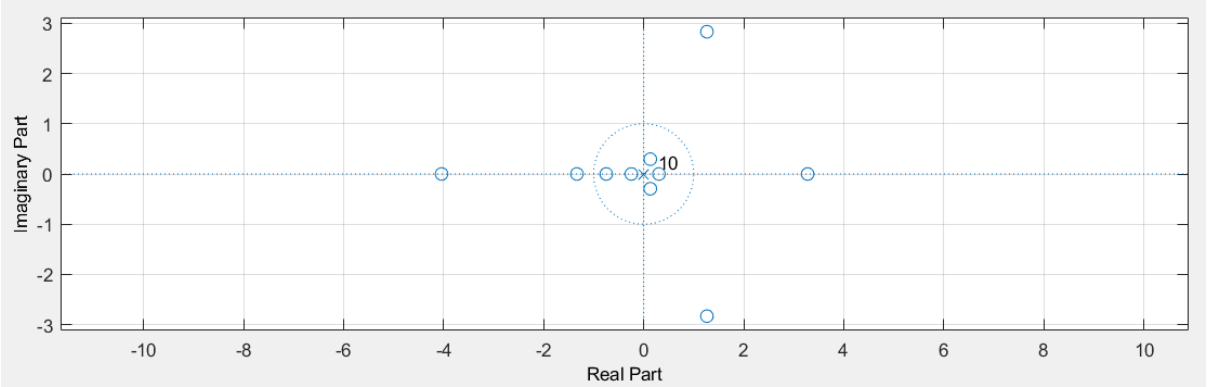
PHASE RESPONSE



IMPULSE RESPONSE

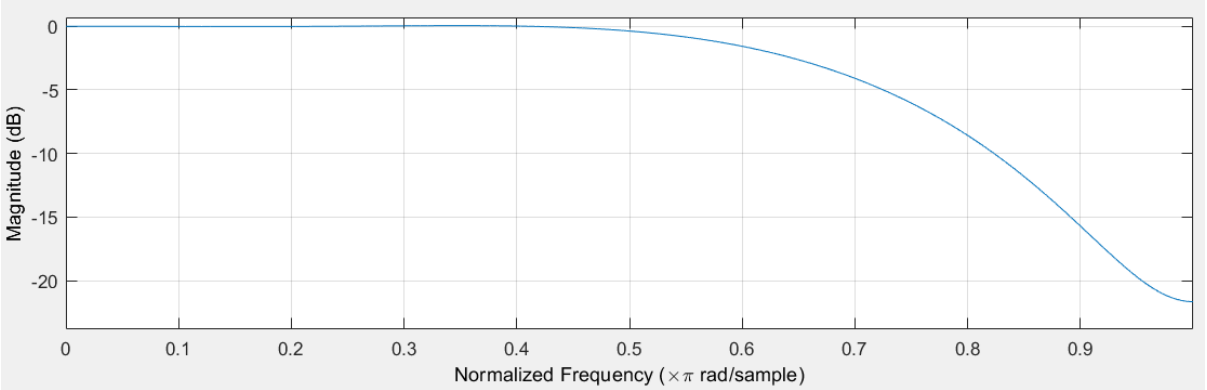


POLE ZERO PLOT

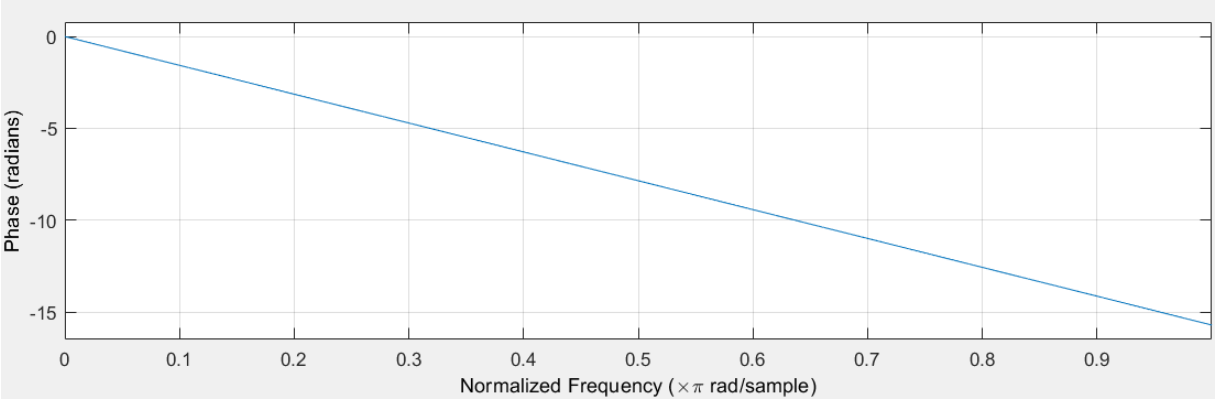


B) HANNING WINDOW

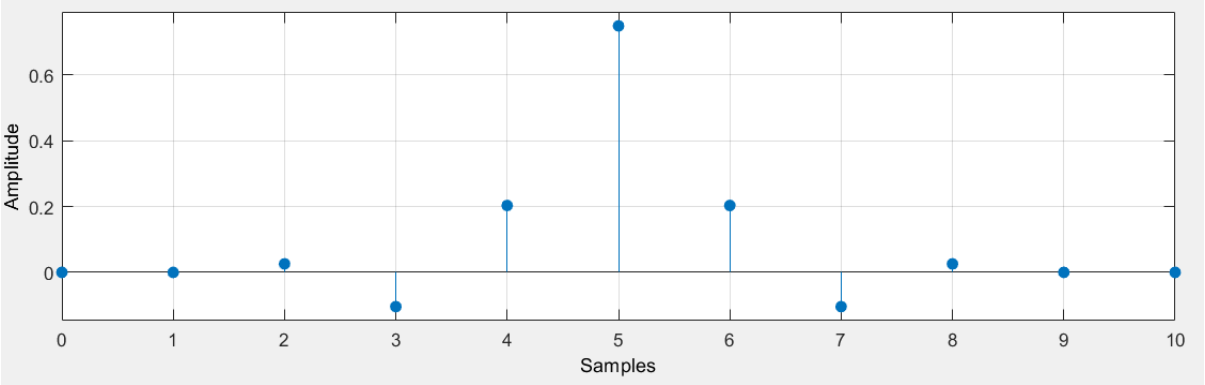
MAGNITUDE RESPONSE



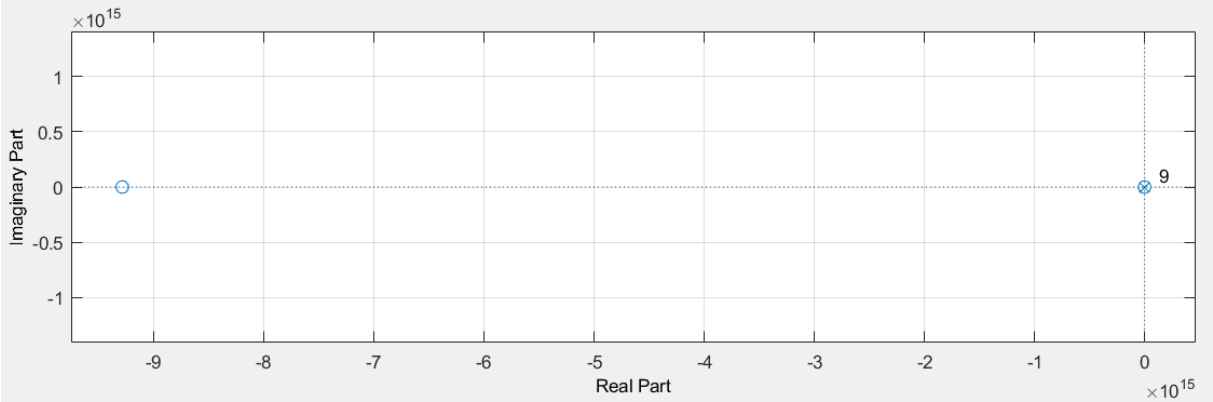
PHASE RESPONSE



IMPULSE RESPONSE

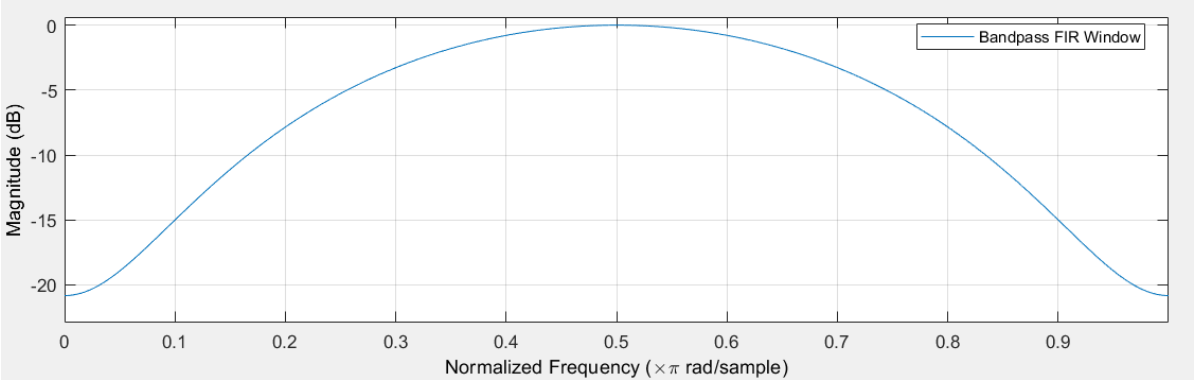


POLE ZERO PLOT

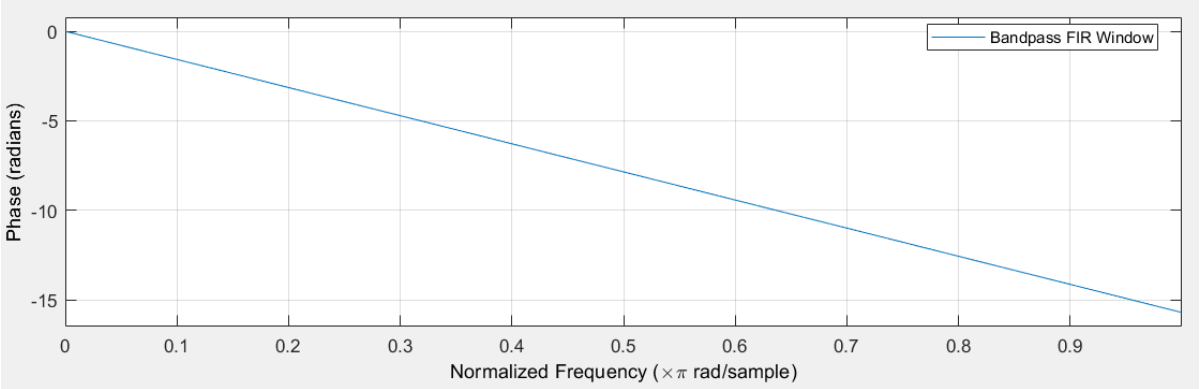


Q.5)
HANNING WINDOW

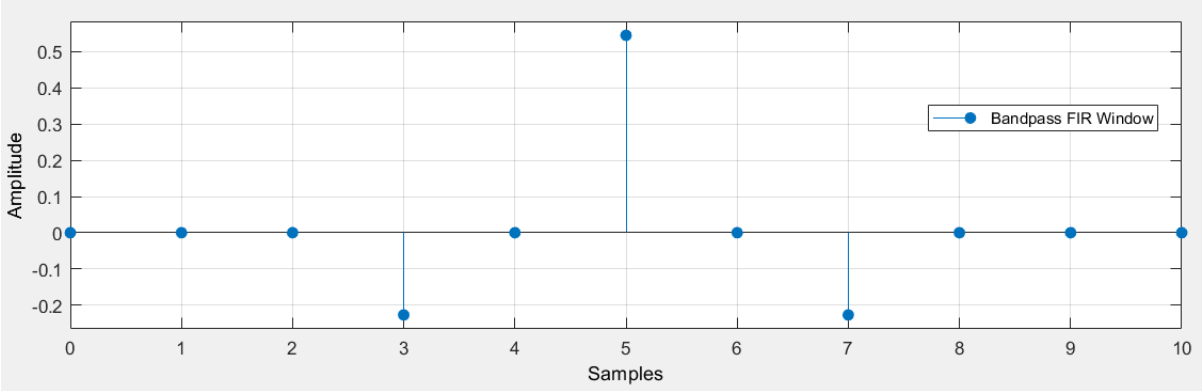
MAGNITUDE RESPONSE



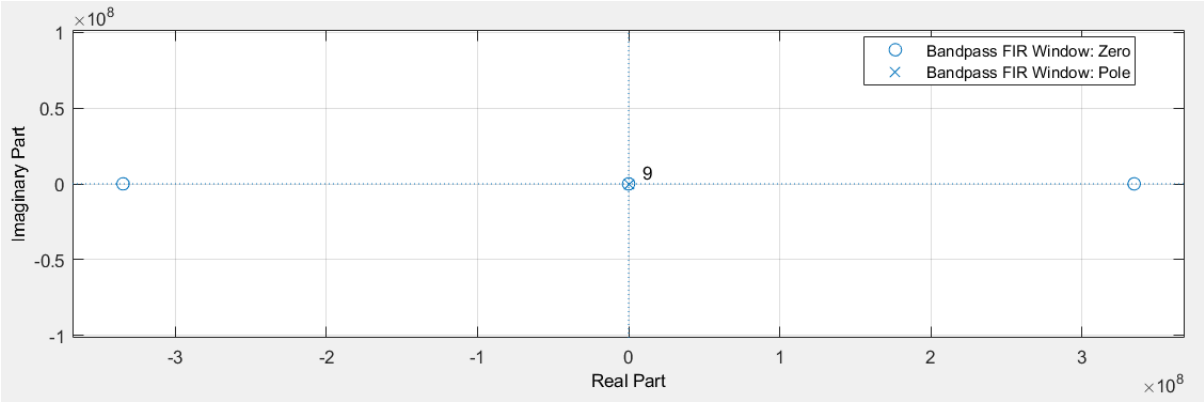
PHASE RESPONSE



IMPULSE RESPONSE



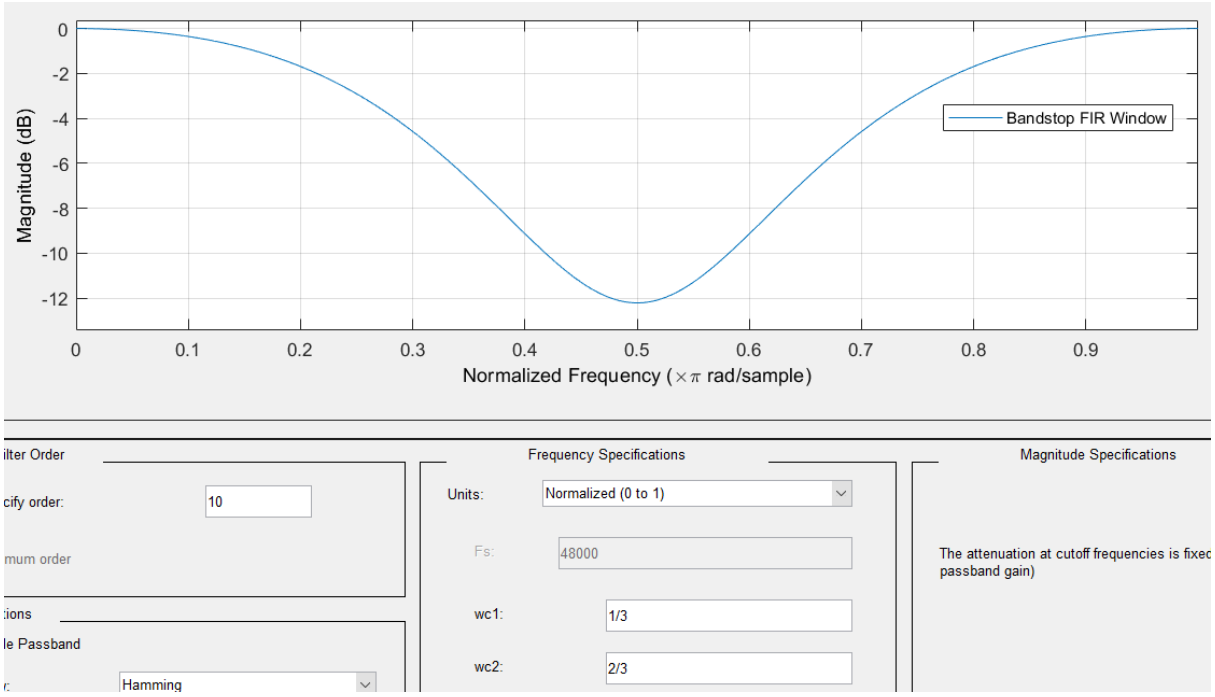
POLE ZERO PLOT



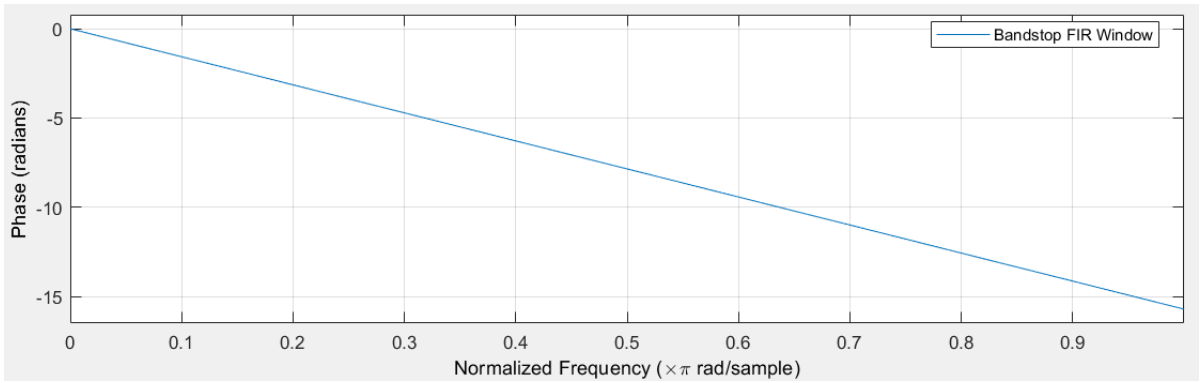
Q6)

A) HAMMING WINDOW

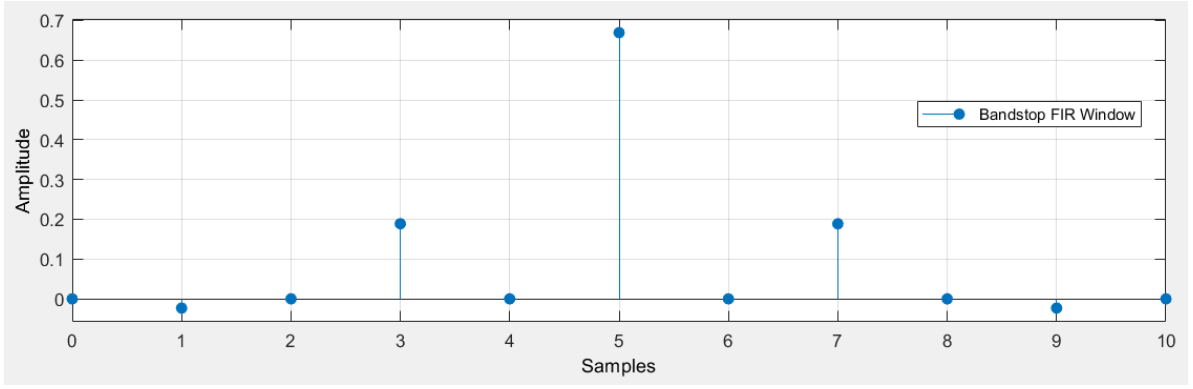
MAGNITUDE RESPONS



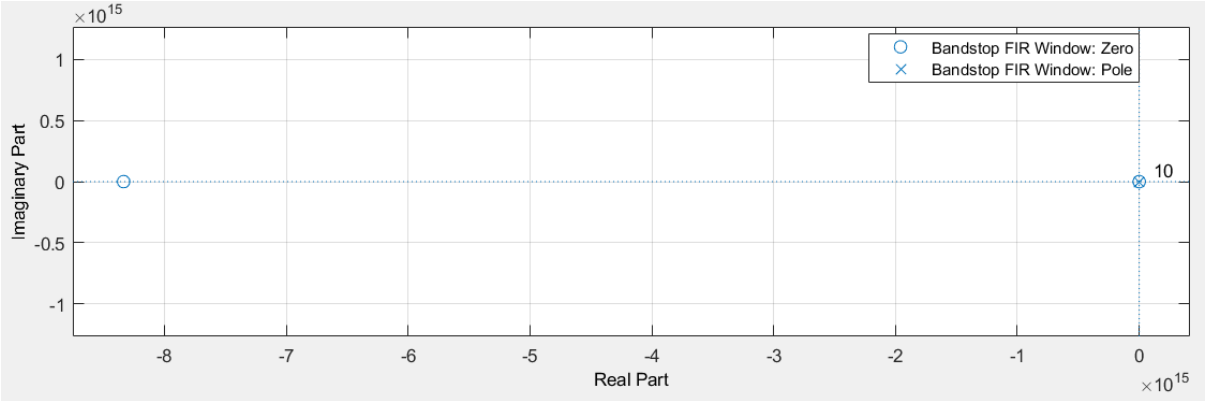
PHASE RESPONSE



IMPULSE RESPONSE

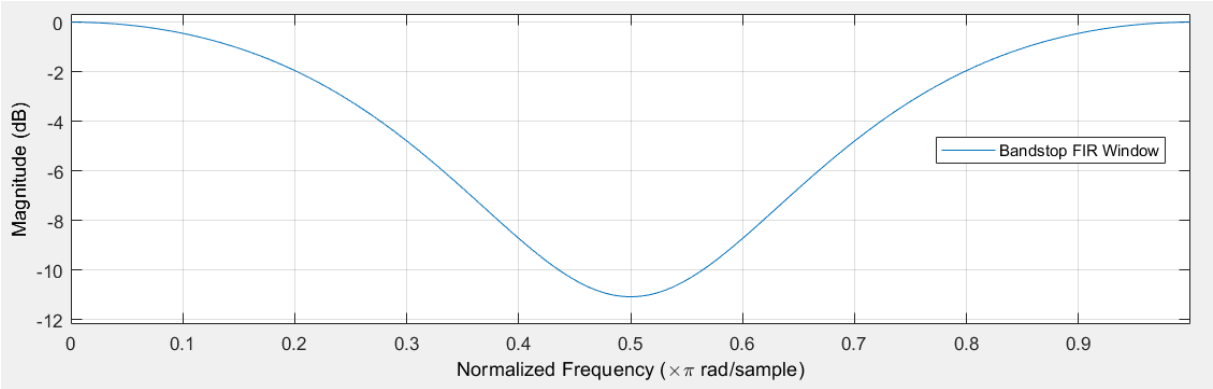


POLE ZERO PLOT

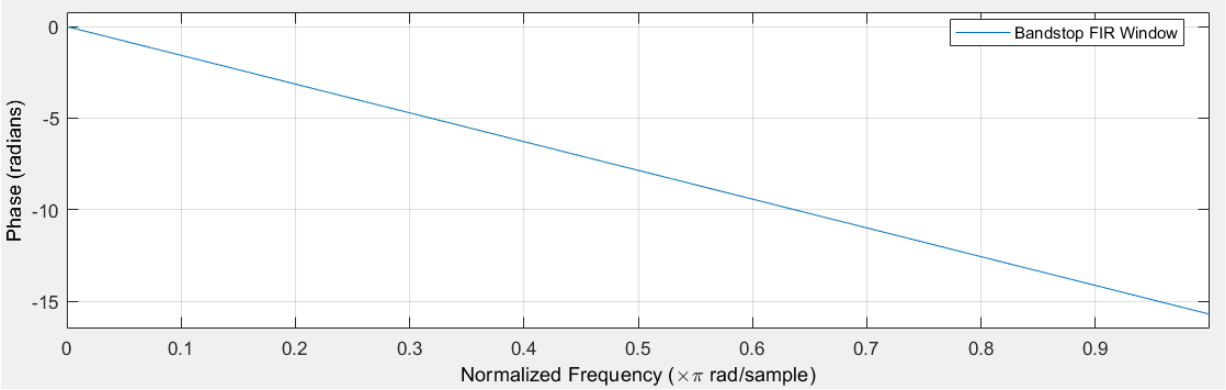


B) HANNING WINDOW

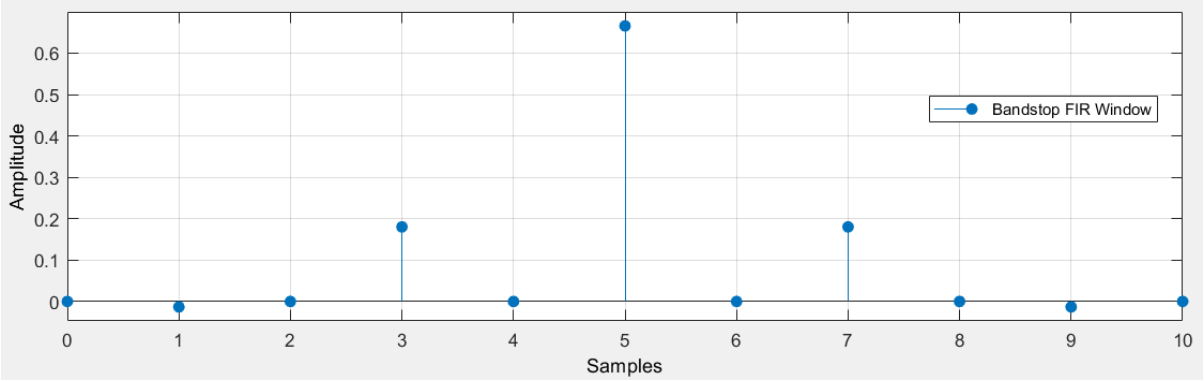
MAGNITUDE RESPONSE



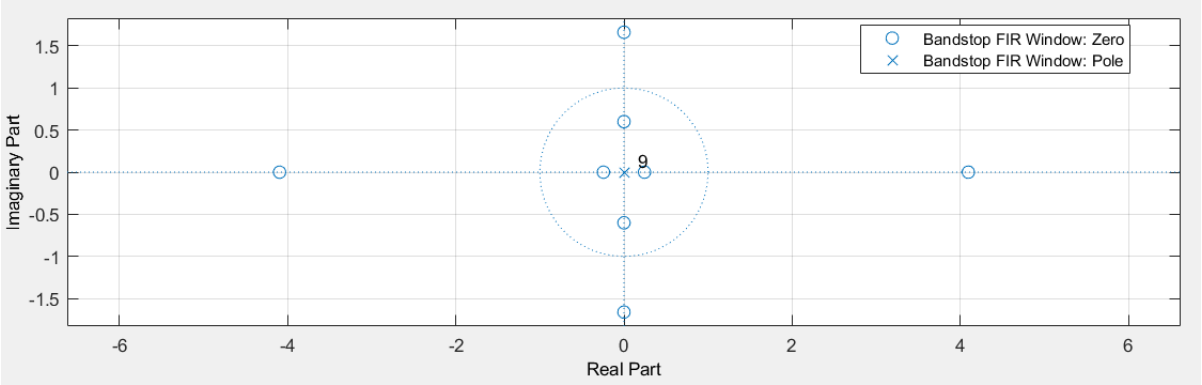
PHASE RESPONSE



IMPULSE RESPONSE

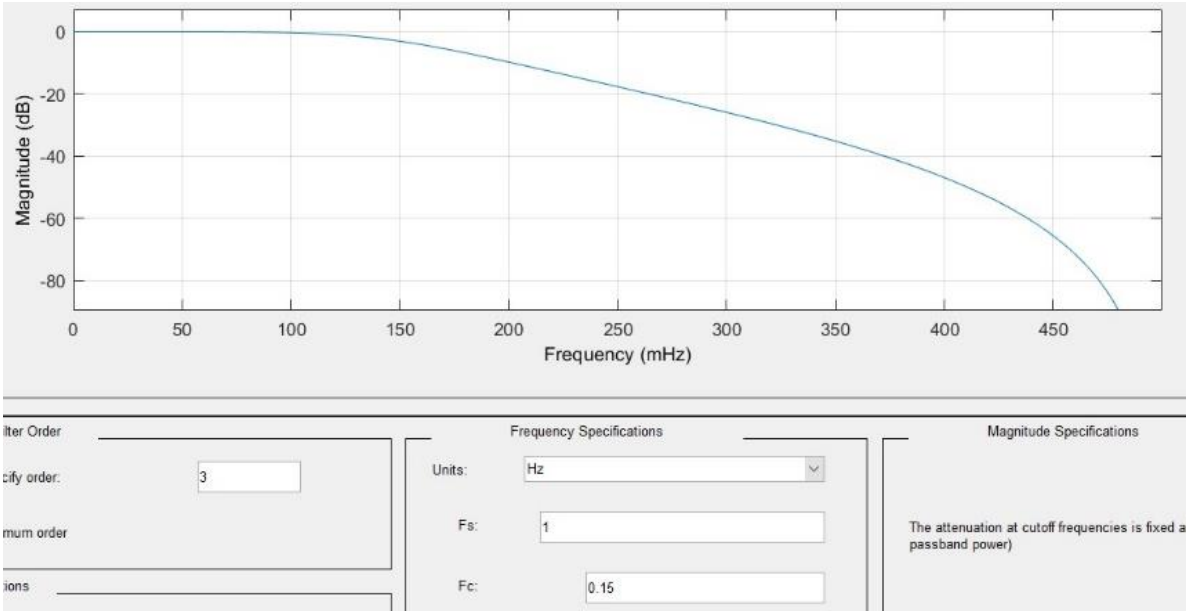


POLE ZERO PLOT

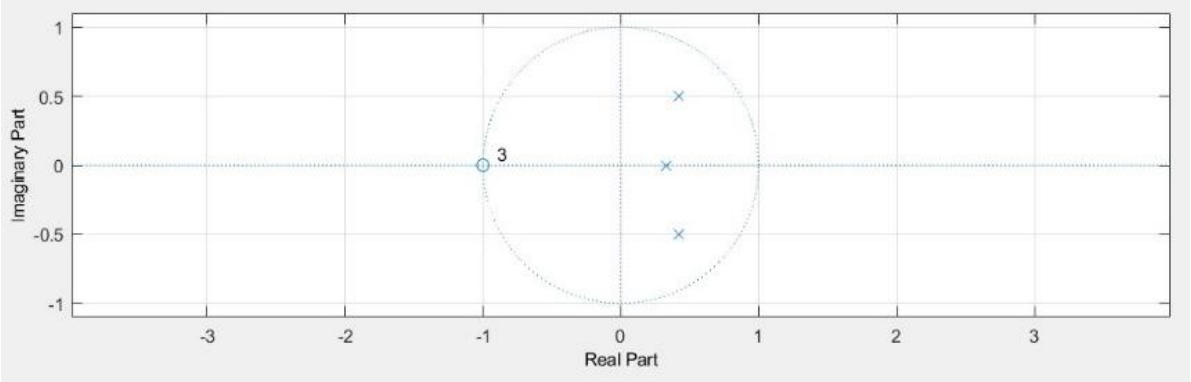


Q7)

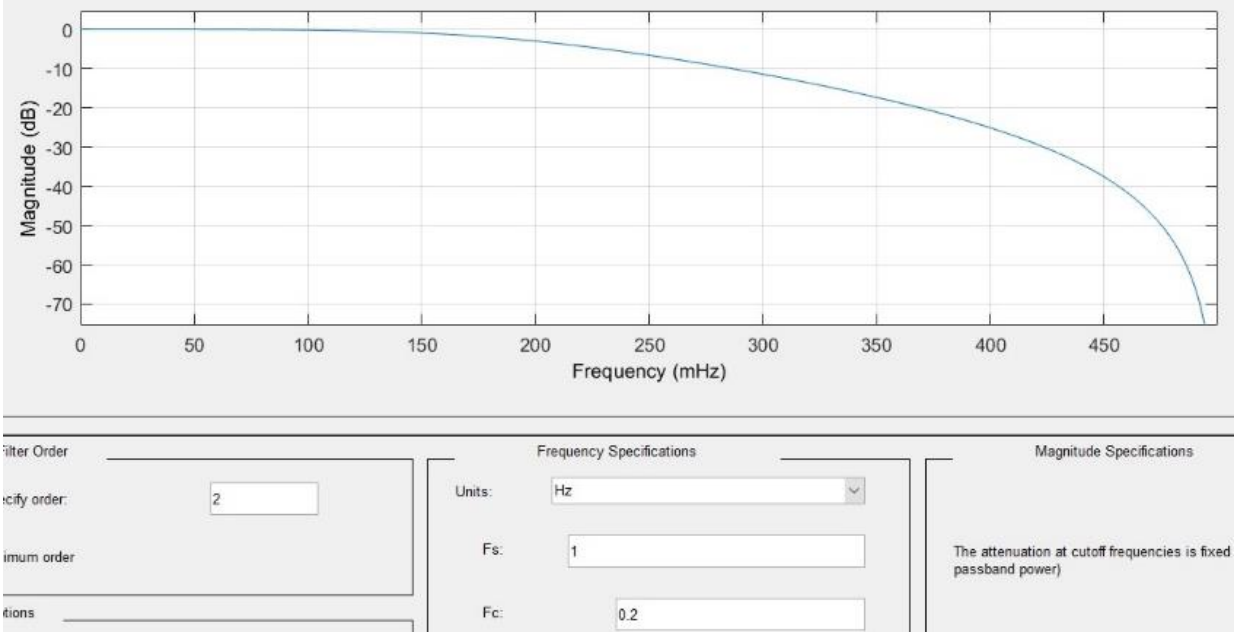
MAGNITUDE RESPONSE



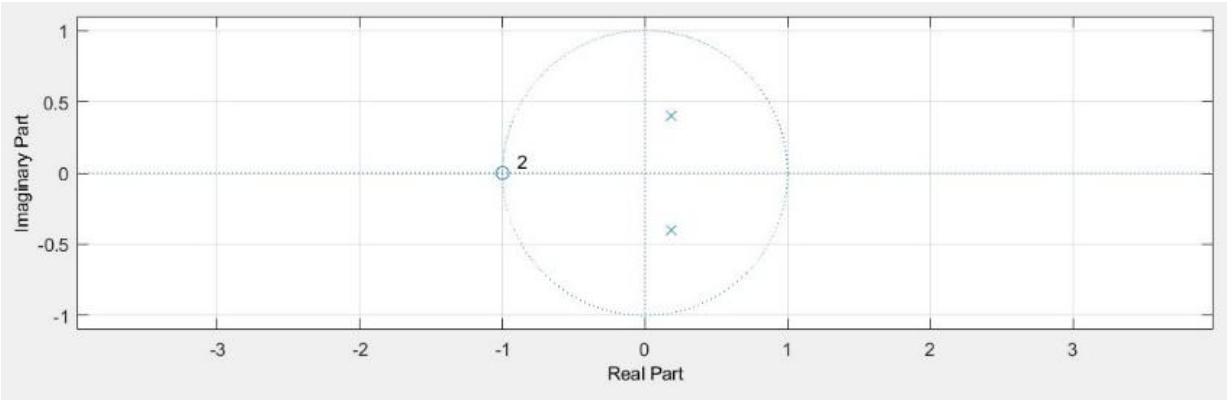
POLE ZERO PLOT



Q8)
MAGNITUDE RESPONSE

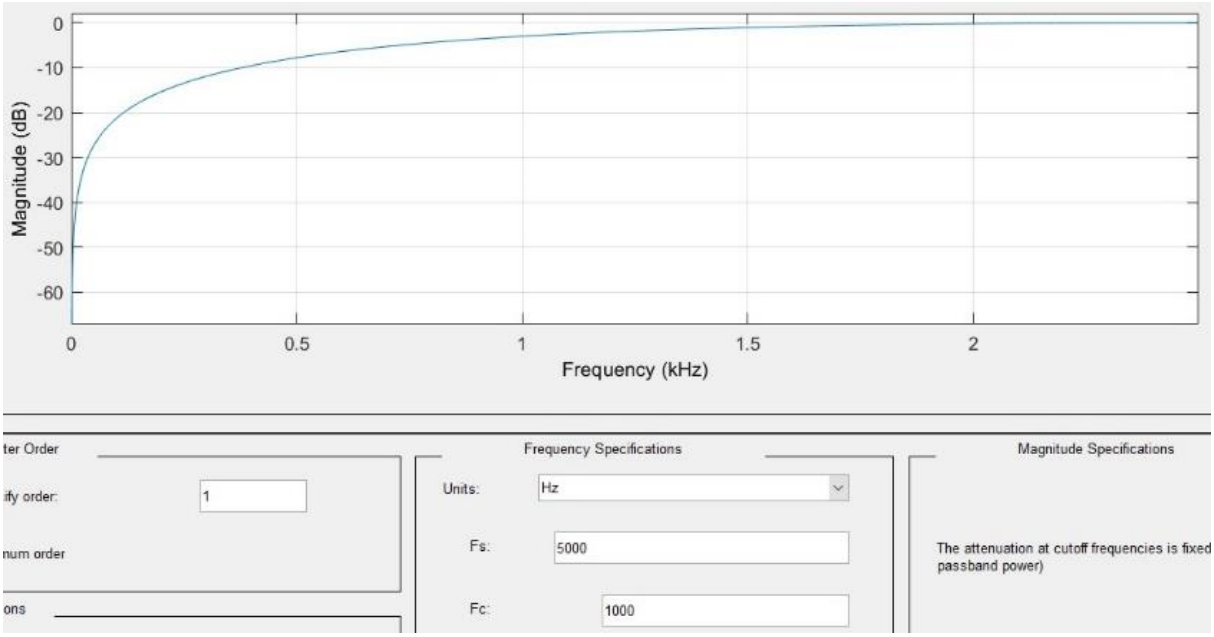


POLE ZERO PLOT

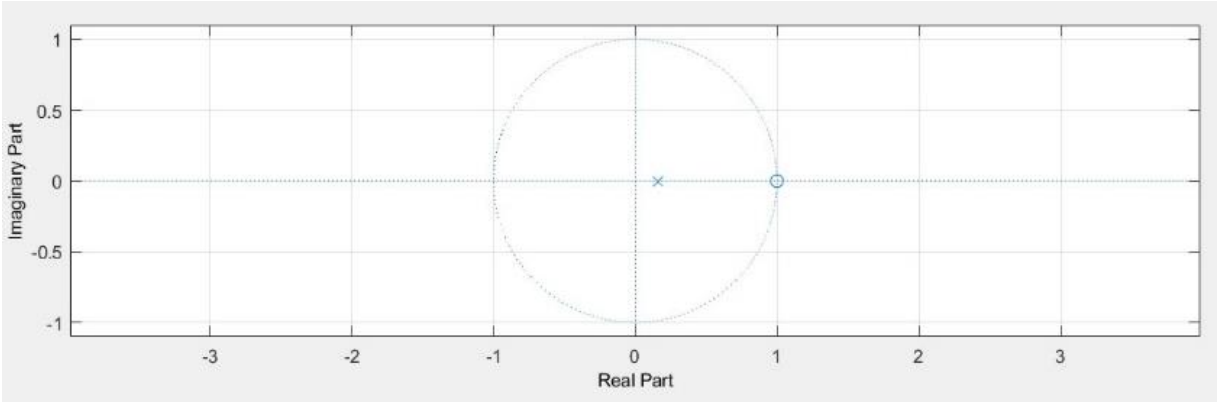


Q9)

MAGNITUDE RESPONSE

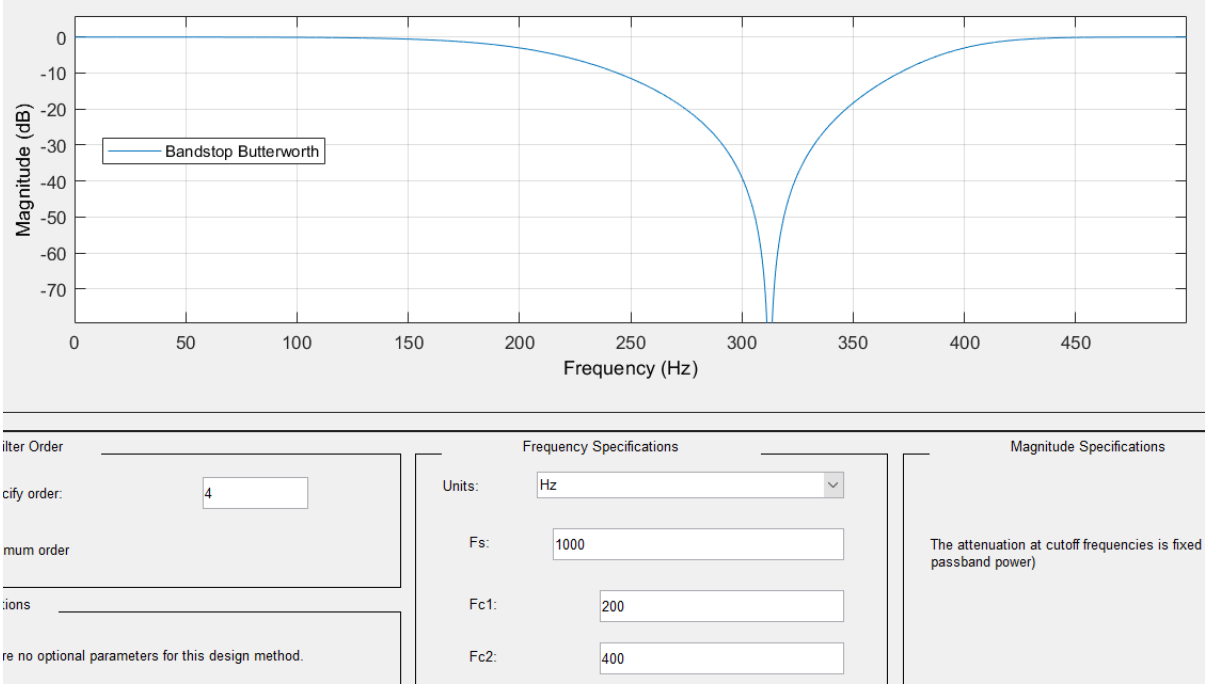


POLE ZERO PLOT

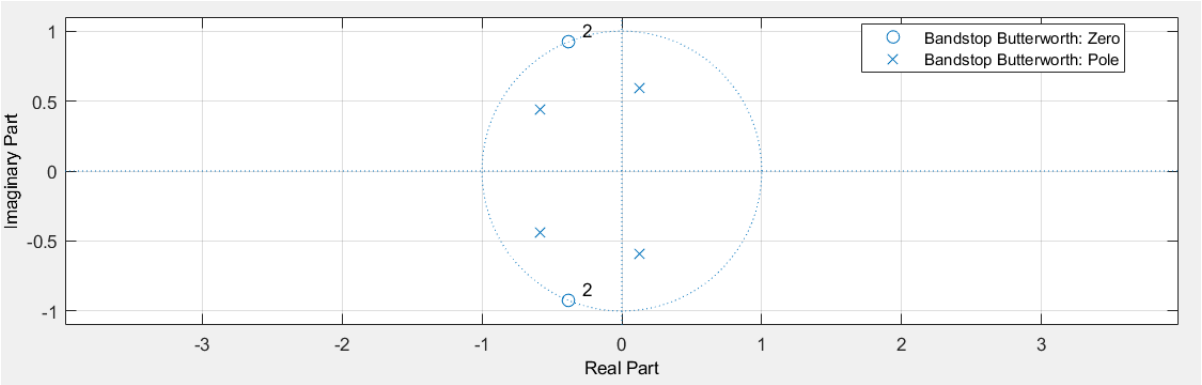


Q10)

MAGNITUDE RESPONSE



POLE ZERO PLOT



The use of FDA tool for design of FIR AND IIR Filter is studied and different responses are observed.