MUFFAKHAM JAH

COLLEGE OF ENGINEERING AND TECHNOLOGY

PC552EC

SYSTEMS AND SIGNAL PROCESSING LAB

(With effect from the academic year 2018-2019)

STUDENT'S MANUAL

DEPARTMENT OF

ELECTRONICS AND COMMUNICATION ENGINEERING

Vision and Mission of the Institution

Vision

To be part of universal human quest for development and progress by contributing high calibre, ethical and socially responsible engineers who meet the global challenge of building modern society in harmony with nature.

Mission

- To attain excellence in imparting technical education from the undergraduate through doctorate levels by adopting coherent and judiciously coordinated curricular and co-curricular programs
- To foster partnership with industry and government agencies through collaborative research and consultancy
- To nurture and strengthen auxiliary soft skills for overall development and improved employability in a multi-cultural work space
- To develop scientific temper and spirit of enquiry in order to harness the latent innovative talents
- To develop constructive attitude in students towards the task of nation building and empower them to become future leaders
- To nourish the entrepreneurial instincts of the students and hone their business acumen.
- To involve the students and the faculty in solving local community problems through economical and sustainable solutions.

Vision and Mission of ECE Department

Vision

To be recognized as a premier education center providing state of art education and facilitating research and innovation in the field of Electronics and Communication.

Mission

We are dedicated to providing high quality, holistic education in Electronics and Communication Engineering that prepares the students for successful pursuit of higher education and challenging careers in research, R& D and Academics.

Program Educational Objectives of B. E (ECE) Program:

1. Graduates will demonstrate technical competence in their chosen fields of employment by identifying, formulating, analyzing and providing engineering solutions using current techniques and tools
2. Graduates will communicate effectively as individuals or team members and demonstrate leadership skills to be successful in the local and global cross-cultural working environment
3. Graduates will demonstrate lifelong learning through continuing education and professional development
4. Graduates will be successful in providing viable and sustainable solutions within societal, professional, environmental and ethical contexts

MUFFAKHAM JAH COLLEGE OF ENGINEERING AND TECHNOLOGY

BANJARA HILLS, ROAD NO-3, TELANGANA

LABORATORY MANUAL

FOR

SYSTEMS AND SIGNAL PROCESSING LAB

Prepared by: **Checked by:**

Approved by:

MUFFAKHAM JAH COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS ENGINEERING

(Name of the Subject/Lab Course): Systems and Signal Processing Lab

Code: PC552EC **Programme: UG**

Branch: ECE **Version No: 1**

Year : III **Updated on: 24/7/18**

Semester :V **No. of Pages:**

Classification Status(Unrestricted/restricted): Unrestricted

Distribution List :Department, Lab, Library, Lab Incharge

Prepared by: 1) Name : **1) Name :**

2) Sign : **2) Sign :**

3)Designation : **3) Designation :**

4) Date : **4) Date :**

Verified by: 1) Name : *** For Q.C Only**

2) Sign : **1) Name :**

3)Designation : **2) Sign :**

4) Date : **3) Designation :**

4) Date :

Approved by: (HOD) 1) Name:

2) Sign :

3) Date :

PC552EC *With effect from Academic Year 2018-19*

SYSTEMS AND SIGNAL PROCESSING LAB

Instructions	4 Periods per Week
Duration of University Examination	3 Hours
University Examination	50 Marks
Sessionals	25 Marks

PART-A (Experiments on Signal Processing)

1. DFT and FFT algorithm.
2. Linear Convolutions.
3. Circular Convolutions.
4. FIR filter design using different data windows.
5. IIR filter design: Butterworth and Chebyshev.
6. Interpolation and Decimation.
7. Implementation of multi-rate systems.
8. Time response of non-linear system.
9. Design of P, PI, PD and PID controllers (any two)

Part –B(**Experiments on DSK and CCS)**

1. Solutions of difference equations.
2. Impulse Response.
3. Linear Convolution.
4. Circular Convolution.
5. Study of procedure to work in real –time.
6. Fast Fourier Transform Algorithms :(DIT, DIF).
7. Design of FIR (LP/HP) using windows, (a) Rectangular, (b) Triangular (c) Hamming window.
8. Design of IIR (HP/LP) filters.

Note:

1. 1.Minimum of **5** from Part A and **5** from Part B is mandatory.

1. 2.For section 'B', MATLAB with different toolboxes like Signal Processing, Signal Processing block set, and SIMULINK/ MATHEMATICA/ any popular software can be used.

SYSTEMS AND SIGNAL PROCESSING LAB

GENERAL GUIDELINES AND SAFETY INSTRUCTIONS

1. Sign in the log register as soon as you enter the lab.
2. Strictly observe your lab timings.
3. Strictly follow the written and verbal instructions given by the teacher / Lab Instructor. If you do not understand the instructions, the handouts and the procedures, ask the instructor or teacher.
4. It is mandatory to come to lab in a formal dress and wear your ID cards.
5. Do not wear loose-fitting clothing or jewelry in the lab
6. Mobile phones should be switched off in the lab. Keep bags in the bag rack.
7. Keep the labs clean at all times, no food and drinks allowed inside the lab.
8. Do not tamper with computer configurations.
9. Playing games on the computers is strictly prohibited.
10. Use of Internet during laboratory timings is prohibited.
11. Shut down the computer and switch off the monitor before leaving your table.
12. Handle the Trainer kits with care.
13. Don’t plug any external devices/Pen drives without permission from lab staff.
14. Don’t install any software without the permission of the lab Incharge.
15. Observation book and lab record should be carried to each lab.
16. Be sure of location of fire extinguishers and first aid kits in the laboratory.
17. Please take care of your personal belongings. Lab incharges /Staff are not responsible for any loss of your belongings.

.

List of Experiments

Page

Part-(A) Experiments on signal processing.

Part-(B) Experiments on DSK and CCS

PART – A

.

EXPERIMENT- 01

DISCRETE FOURIER TRANSFORM

1.a) Discrete Time Fourier Transform

AIM: To perform the DTFT of a discrete time sequence and plot its magnitude and Phase.

SOFTWARE: Matlab R2014a

THEORY: The Discrete time Fourier Transform is a Fourier transform that operates on aperiodic ,discrete signals. Mathematically it is given by

$$X(w) = \sum_{n=-\infty}^{\infty} x[n]e^{-jwn},$$

ALGORITHM:-

- Read the input sequence $x_1[n]$,and plot
- Use the Matlab function 'fft' to perform Fourier Transform
- Obtain Magnitude of Fourier Transform using Matlab Function 'abs' and Plot
- Obtain Phase of Fourier Transform using Matlab Function 'angle' and Plot

MATLAB PROGRAM:

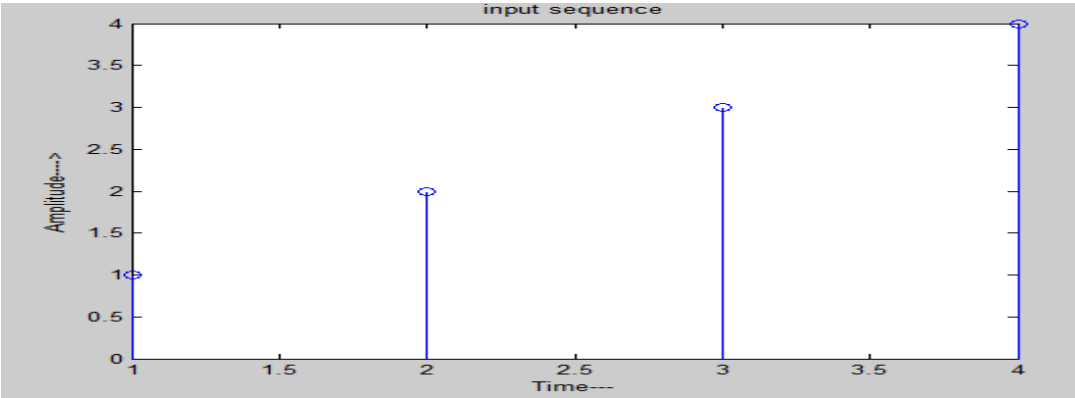
```
clc;
clear all;
close all;
x1=input('Enter the equence:');
y1=fft(x1);
disp('y1='); disp(y1);
plot(x1);
 xlabel('Time---');
ylabel('Amplitude---->')
title('input sequence');
mag1=abs(y1);
figure;
plot(mag1);
xlabel('Time—>');
ylabel('Amplitude---->');
title('Magnitude Plot');
phase1=angle(y1);
figure;
plot(phase1);
```

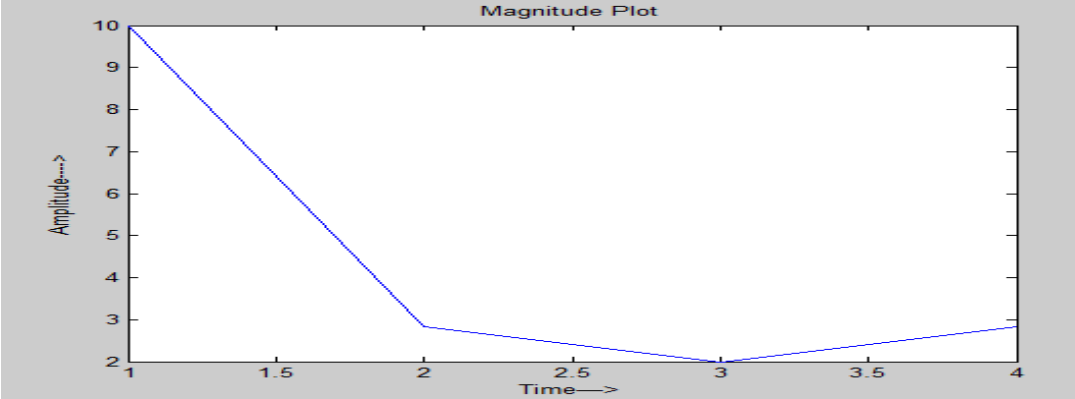
```
xlabel('Time--->');
ylabel('Amplitude ----‘);
title('Phase Plot');
```

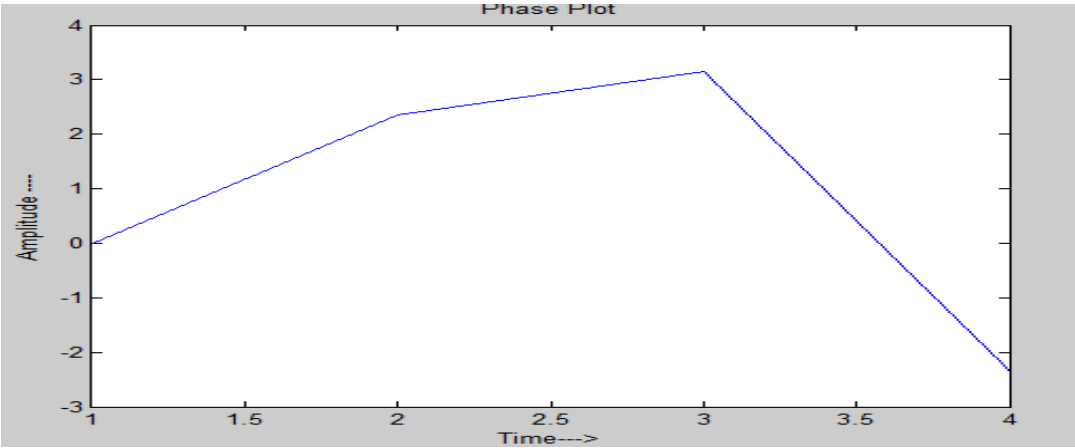
Enter the sequence: [1 2 3 4]
y1 =

10.0000 -2.0000 + 2.0000i -2.0000 -2.0000 - 2.0000i

OUTPUT WAVEFORMS







1.b)DISCRETE FOURIER TRANSFORM

AIM: To perform the DFT of a discrete time sequence and plot its magnitude and Phase.

SOTWARE: Matlab R2014a

THEORY: Given a sequence of N samples x(n) , the discrete Fourier Transform (DFT) is defined as X(k)

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi}{N}kn}, \quad k = 0, \ldots, N-1,$$

ALGORITHM:-

- Read the input sequence $x_1[n]$,and plot
- Use the Matlab function 'fft' to perform Fourier Transform
- Obtain Magnitude of Fourier Transform using Matlab Function 'abs' and Plot using 'stem' function
- Obtain Phase of Fourier Transform using Matlab Function 'angle' and Plot using 'stem'

MATLAB PROGRAM:

```
clc;
clear all;
close all;
x1=input('Enter the sequence');
y1=fft(x1);
disp('y1=');
 disp(y1); stem(x1);
xlabel('Time--->');
ylabel('Amplitude----');
title('input sequence');
mag1=abs(y1);
figure;
stem(mag1);
xlabel('Time--->');
ylabel('Amplitude---->');
title('Magnitude Plot');
phase1=angle(y1);
figure;
stem(phase1);
xlabel('Time--->');
ylabel('Amplitude ----');
 title('Phase Plot');
```

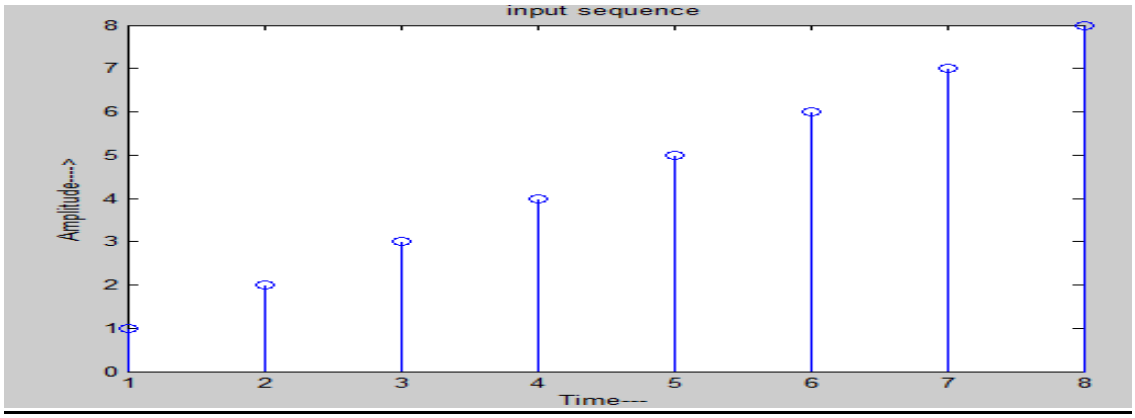
Enter input sequence [1 2 3 4 5 6 7 8]
y1 =
Columns 1 through 5

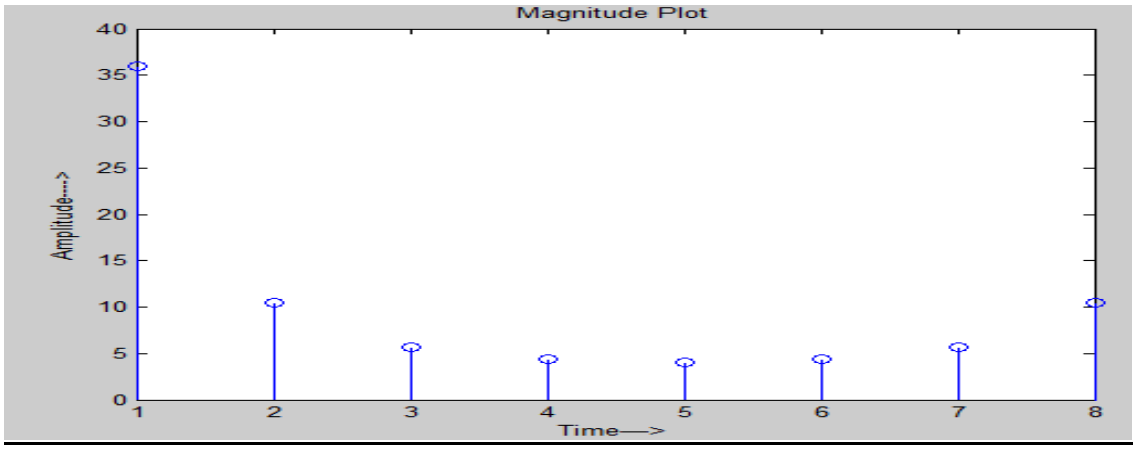
36.0000 -4.0000 + 9.6569i -4.0000 + 4.0000i -4.0000 + 1.6569i -4.0000

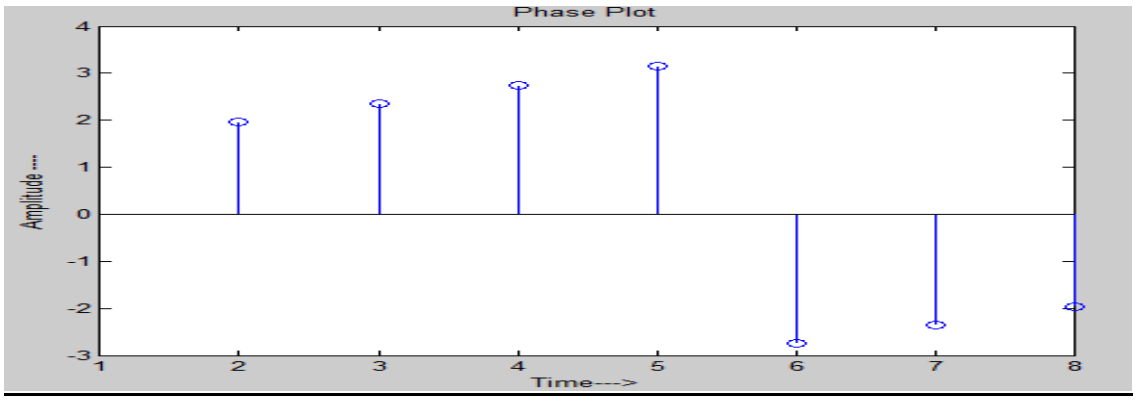
Columns 6 through 8

-4.0000 - 1.6569i -4.0000 - 4.0000i -4.0000 - 9.6569i

OUTPUT WAVEFORMS







1.c) DFT PERFORMED FOR 100 – POINTS

AIM:To perform the DFT of a discrete time sequence for 100 points and plot its magnitude and Phase.

SOTWARE: Matlab R2014a

ALGORITHM:-

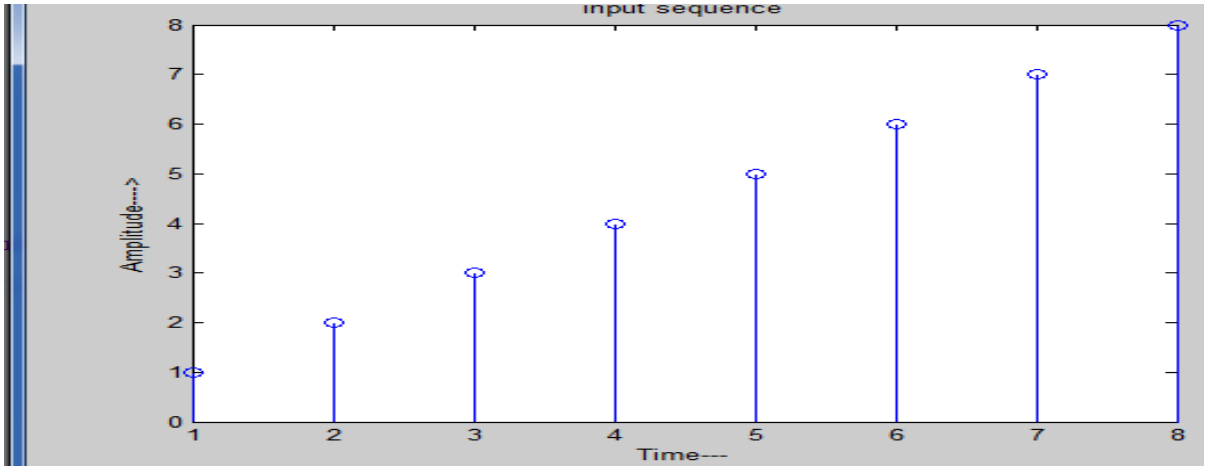
- Read the input sequence $x_1[n]$,and plot
- Use the Matlab function 'fft(x1,100)' to perform Fourier Transform
- Obtain Magnitude of Fourier Transform using Matlab Function 'abs' and Plot using 'stem' function
- Obtain Phase of Fourier Transform using Matlab Function 'angle' and Plot using 'stem'

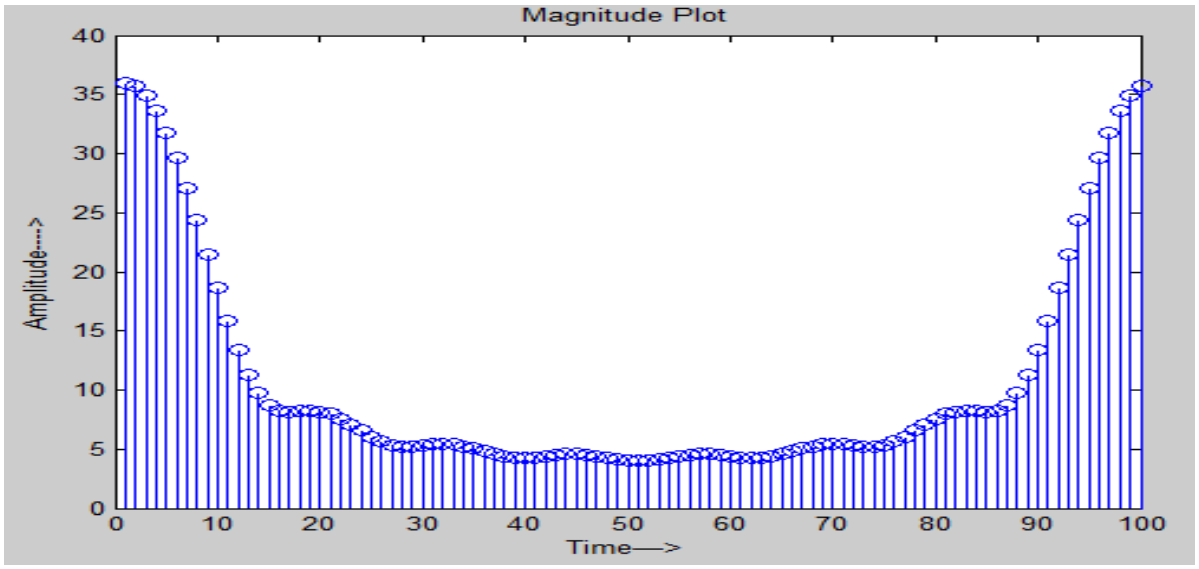
MATLAB PROGRAM:

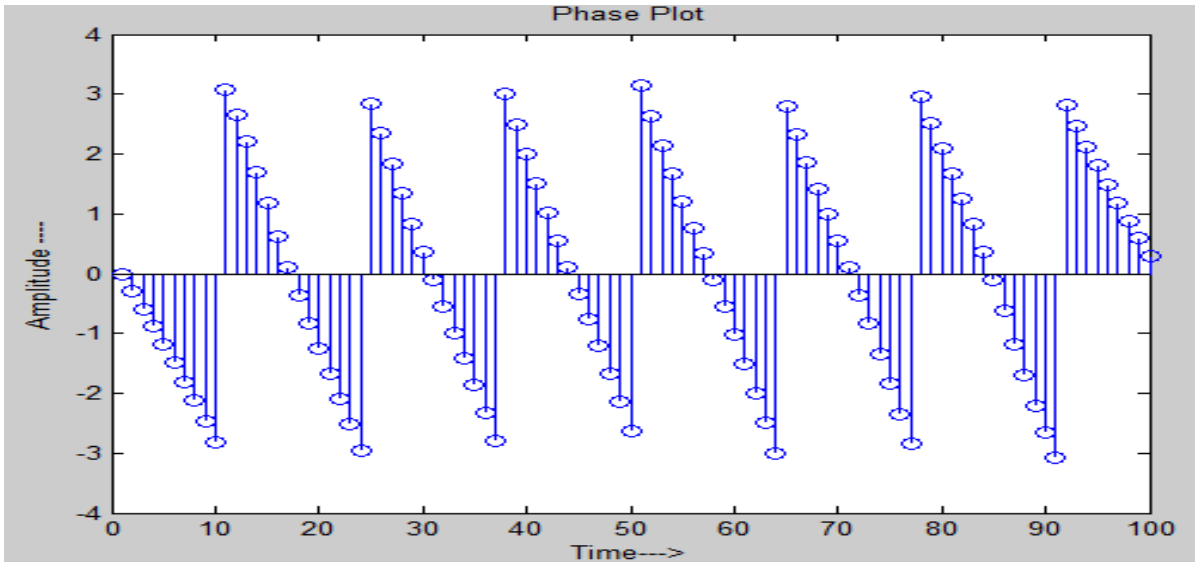
```
clc;
clear all;
close all;
x1=input('Enter the sequence');
y1=fft(x1,100);
stem(x1);
 xlabel('Time—->);
ylabel('Amplitude---->');
title('First sequence');
mag3=abs(y1);
figure;
stem(mag3);
 xlabel('Time--->');
ylabel('Amplitude---->');
title('Magnitude Plot');
phase3=angle(y1);
figure;%open new window
stem(phase3);
 xlabel('Time--->');
ylabel('Amplitude ---->');
title('Phase Plot');
```

Enter the sequence [1 2 3 4 5 6 7 8]

OUTPUT WAVEFORMS







1.d)DISCRETE FOURIER TRANSFORM using FOR LOOP

AIM: To perform the DFT of a discrete time sequence using FOR Loop and plot its magnitude and Phase.

SOTWARE: Matlab R2014a

THEORY: Given a sequence of N samples x(n) , the discrete Fourier Transform (DFT) is defined as X(k)

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi}{N}kn}, \quad k = 0, \ldots, N-1,$$

ALGORITHM:-

- Read the input sequence $x_1[n]$,and plot
- Use the FOR loop Syntax to perform Discrete Fourier Transform
- Obtain Magnitude of Fourier Transform using Matlab Function 'abs' and Plot using 'stem' function
- Obtain Phase of Fourier Transform using Matlab Function 'angle' and Plot using 'stem'

MATLAB PROGRAM:

```
clc;
clear all;
close all;
a=input('Enter the sequence');
stem(a);
xlabel('Time--->');
ylabel('Amplitude---->');
title('input sequence');
N=length(a);
for k=1:N;
    y(k)=0;
    for i=1:N
        y(k)=y(k)+a(i)*exp((-2*pi*j/N)*((i-1)*(k-1)));
    end;
end;
k=1:N;
disp('the result is :');y
mag1=abs(y);
figure;
stem(mag1);
xlabel('Time--->');
ylabel('Amplitude---->');
```

```
title('Magnitude Plot');
phase1=angle(y);
figure;
stem(phase1);
xlabel('Time--->');
ylabel('phase---->');
title('Phase Plot');
```

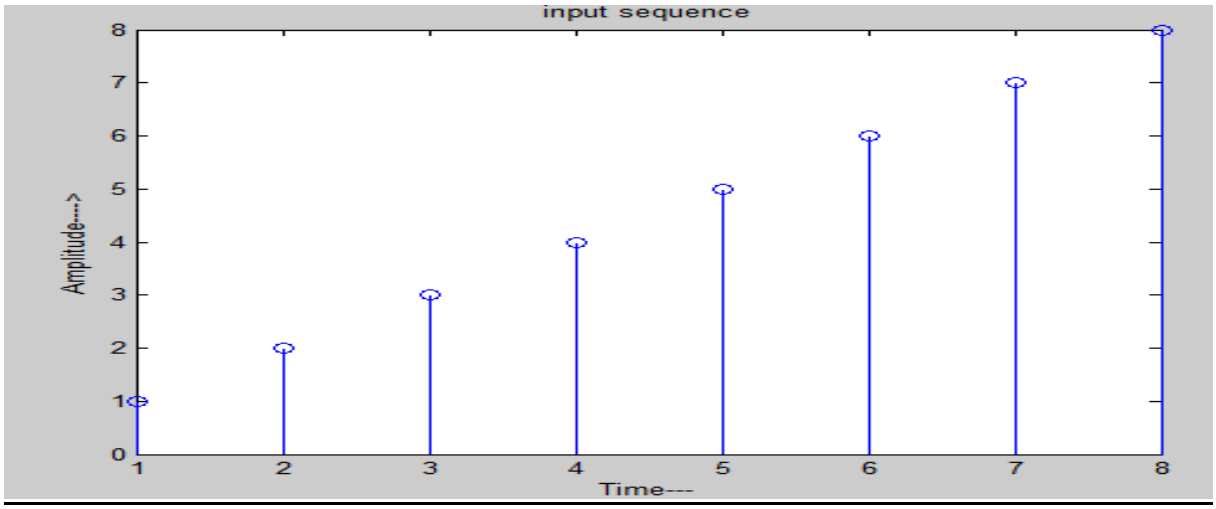
Enter the sequence [1 2 3 4 5 6 7 8]

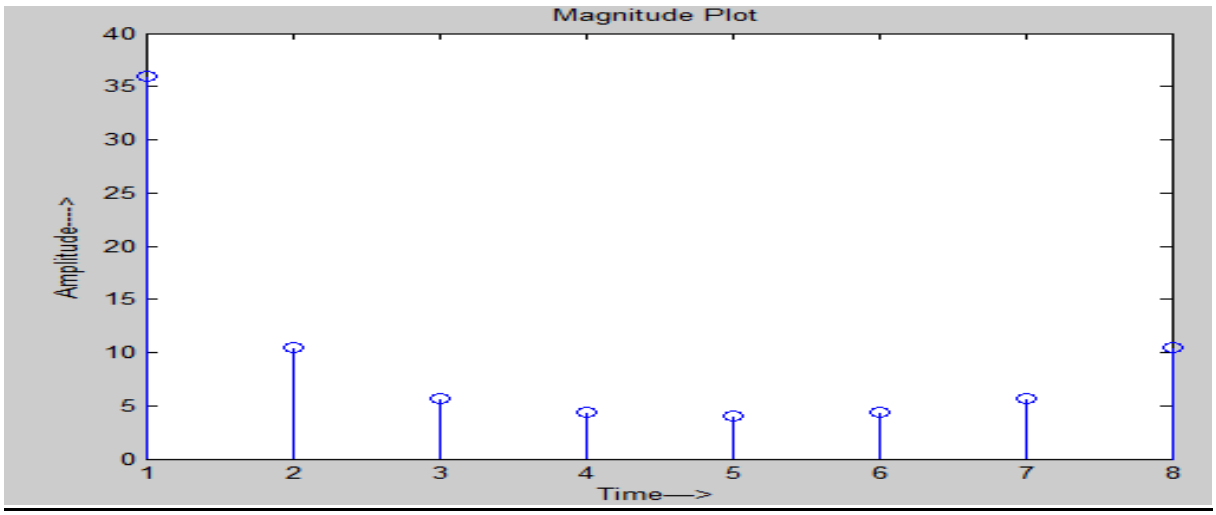
OUTPUT:

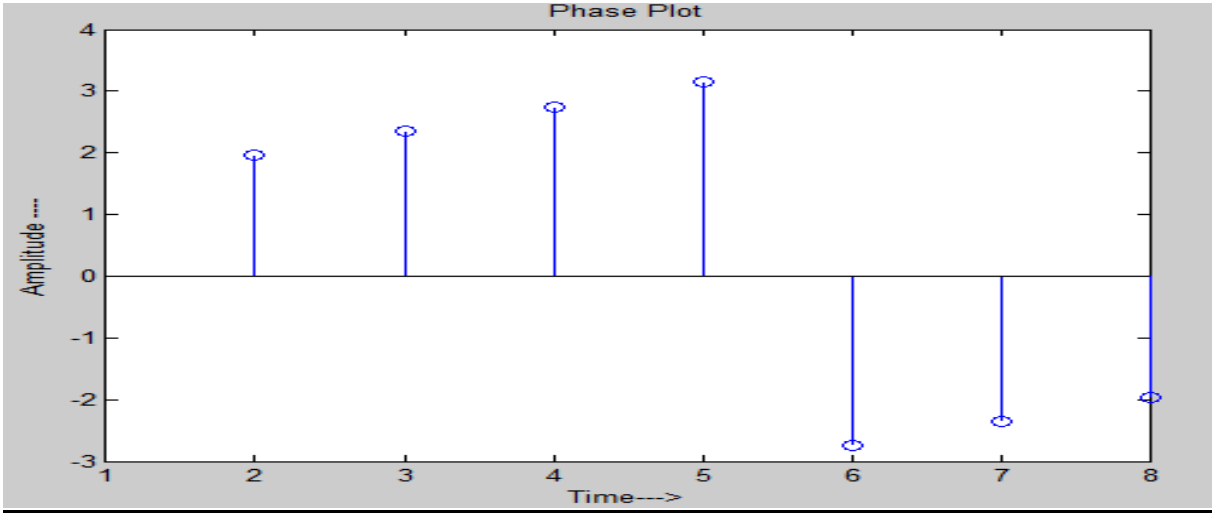
y =

36.0000	-4.0000 + 9.6569i	-4.0000 + 4.0000i	-4.0000 + 1.6569i
-4.0000	-4.0000 - 1.6569i	-4.0000 - 4.0000i	-4.0000 - 9.6569i

OUTPUT WAVEFORMS







1.e)INVERSE DFT

AIM: To perform the Inverse DFT of a discrete sequence .

SOTWARE: Matlab R2014a

THEORY:

we have the **inverse DFT** or (**IDFT**):

$$x[n] = \begin{cases} \frac{1}{N}\sum_{k=0}^{N-1} X[k]\, e^{j\frac{2\pi}{N}kn}, & n = 0, \ldots, N-1 \\ 0, & \text{otherwise.} \end{cases}$$

ALGORITHM:-

- Read the input sequence $x_1[n]$,and plot
- Use the Matlab function 'fft' to perform Fourier Transform

MATLAB PROGRAM:

```
clc;
clear all;
close all;
x1=input('Enter the sequence');
y1=ifft(x1);
disp('y1=');
stem(abs(x1));
 xlabel('frequency--->');
```

```
ylabel('Amplitude----
>'); title('magitde plot of
i/p')
figure
stem(angle(x1))
xlabel('frequency--->');
ylabel('Amplitude----
>'); title('Phase plot of
i/p')
stem(y1);
 xlabel('Time--->');
ylabel('Amplitude----
>'); title('Output
sequence');
```

Enter the sequence [10 -2+2i -2 -2-2i]

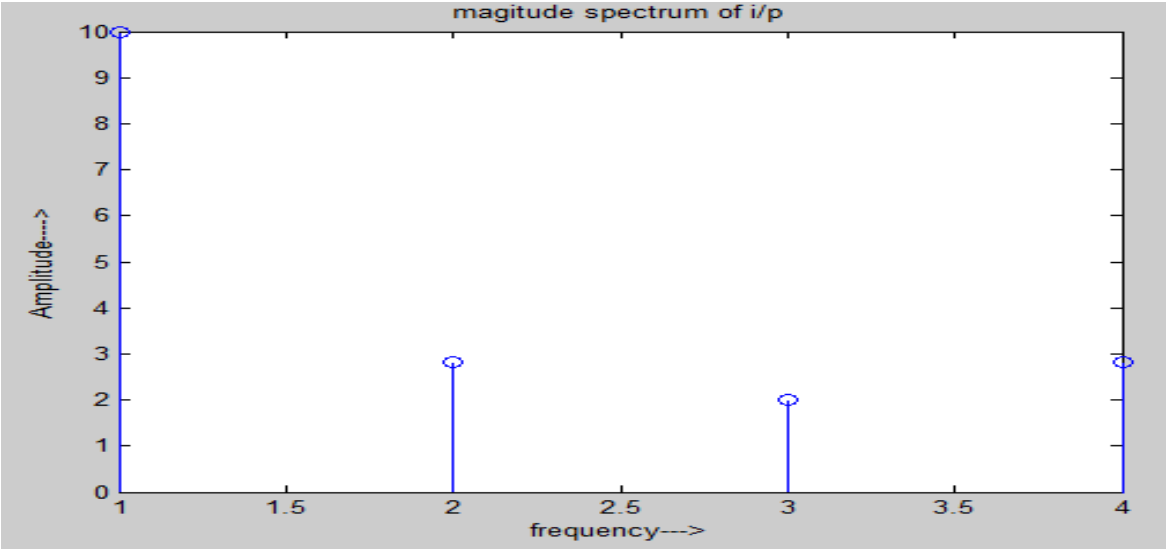
OUTPUT:

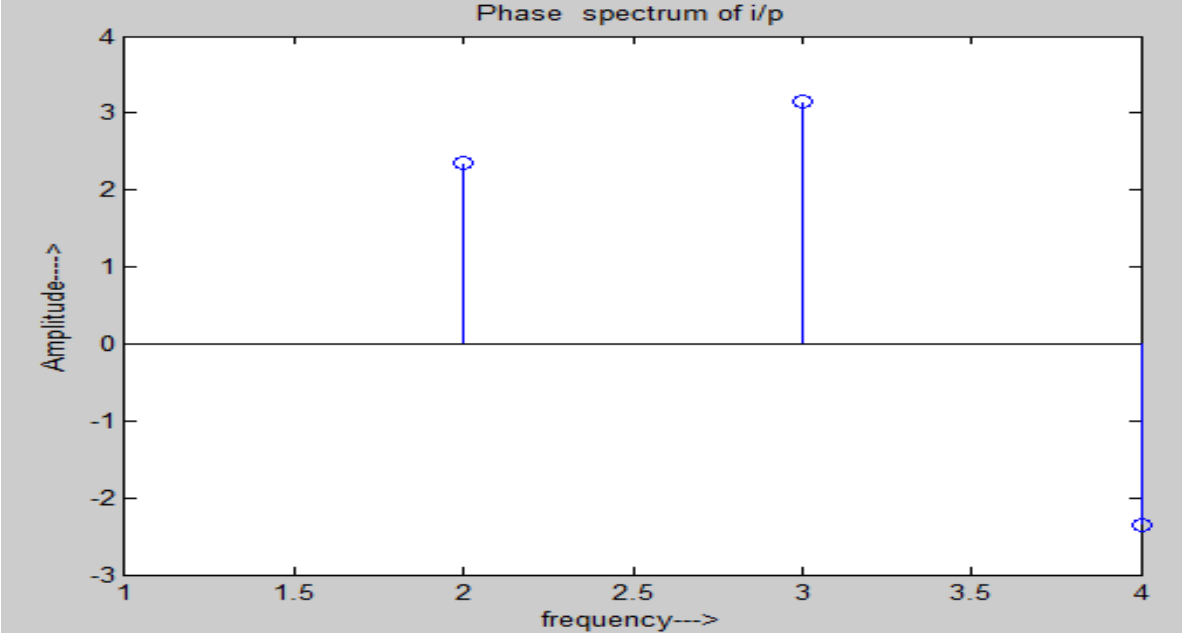
y1=

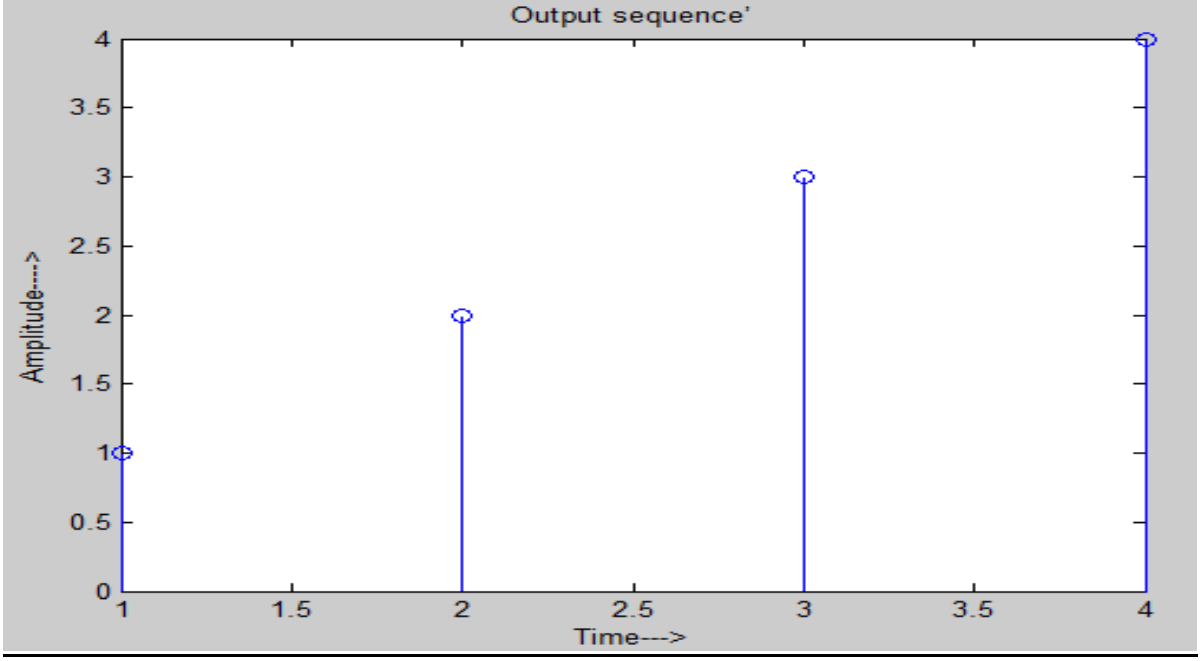
1 2 3 4

2

OUTPUT WAVEFORMS:




Phase spectrum of i/p
Amplitude---->
frequency--->


Output sequence'
Amplitude---->
Time--->

EXPERIMENT- 02

LINEAR CONVOLUTION

2.a) AIM:To Perform Linear Convolution of two sequences.

SOTWARE: Matlab R2014a

THEORY:

Linear Convolution can be represented by a Mathematical Expression as follows:

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

x[n]= Input signal Samples

h[n]= Impulse response

y[n]= Convolution output.

ALGORITHM:-

- Read the input sequence $x_1[n]$,and plot
- Read the input sequence $x_2[n]$, and plot
- Use the user defined matlab function 'conv'
- Convolve the two sequence and plot the result

MATLAB PROGRAM:

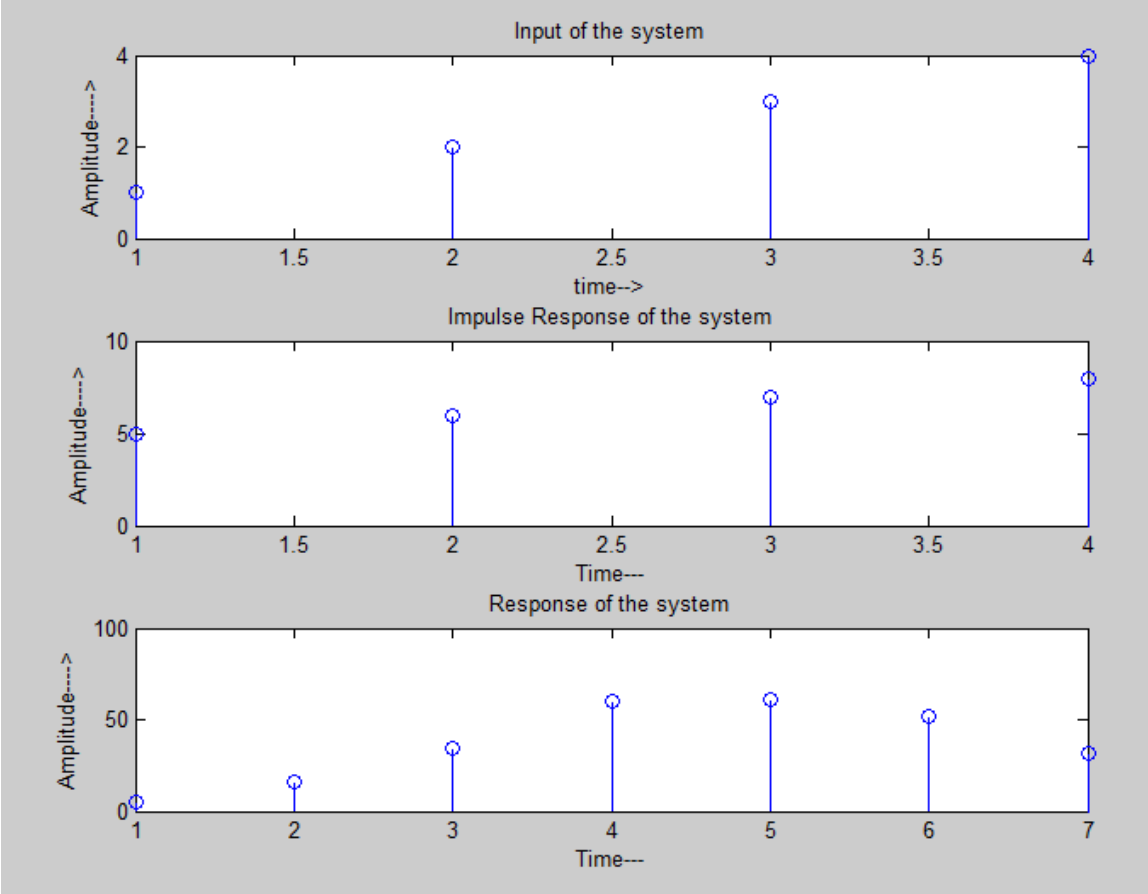
```
clc;
clear all;
close all;

x=input('Enter input sequence');
h=input('Enter Impulse Response');

y=conv(x,h);
disp('Output of the system
is:'); disp('y=')

subplot(3,1,1);
stem(x); xlabel('time-->');
ylabel('Amplitude---->');
title('Input of the system');
```

```
subplot(3,1,2);
stem(h);
 xlabel('Time---');
ylabel('Amplitude--->');
title('Impulse Response of the system');

subplot(3,1,3);
stem(y); xlabel('Time---
'); ylabel('Amplitude----
>');
title('Response of the system');
```

(INPUTS TO BE GIVEN IN COMMAND WINDOW)
Enter input sequence[1 2 3 4]
Enter Impulse Response[5 6 7 8]

OUTPUT:
y = 5 16 34 60 61 52 32

OUTPUT WAVE FORMS



2.b)AIM: DISCRETE LINEAR CONVOLUTION WITH DIFFERENT LOWER AND UPPER LIMITS

SOTWARE: Matlab R2014a

THEORY:

Linear Convolution can be represented by a Mathematical Expression as follows:

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

x[n]= Input signal Samples

h[n]= Impulse response

y[n]= Convolution output.

ALGORITHM:-

- Read the limits of input sequence $x_1[n]$ and $x_2[n]$
- Read the input sequence $x_1[n]$ and $x_2[n]$, and plot
- Use the user defined matlab function 'conv'
- Convolve the two sequence and plot the result

MATLAB PROGRAM:

```
clc;
clear all;
close all;
L1 = input('enter the lower limit of x');
U1 = input('enter the upper limit of
x'); x=input('Enter input sequence');

L2 = input('enter the lower limit of h');
U2 = input('enter the upper limit of
h'); h=input('Enter Impulse
Response');

y=conv(x,h);
disp('Output of the system
is:'); disp('y=');
 subplot(3,1,1);
```

```
stem(L1:U1, x)
xlabel('time-->');
ylabel('Amplitude---->');
title('Input of the system');
subplot(3,1,2);
stem(L2:U2, h)
xlabel('Time--->');
ylabel('Amplitude---->');
title('Impulse Response of the system');
subplot(3,1,3);
stem(L1+L2: U1+U2,y)
xlabel('Time--->');
ylabel('Amplitude ---- >');
title('Response of the
system');
```

(INPUTS TO BE GIVEN IN COMMAND WINDOW)

enter the lower limit of x -1

enter the upper limit of x 2

Enter input sequence [4 3 2 4]

enter the lower limit of h -1

enter the upper limit of h 2

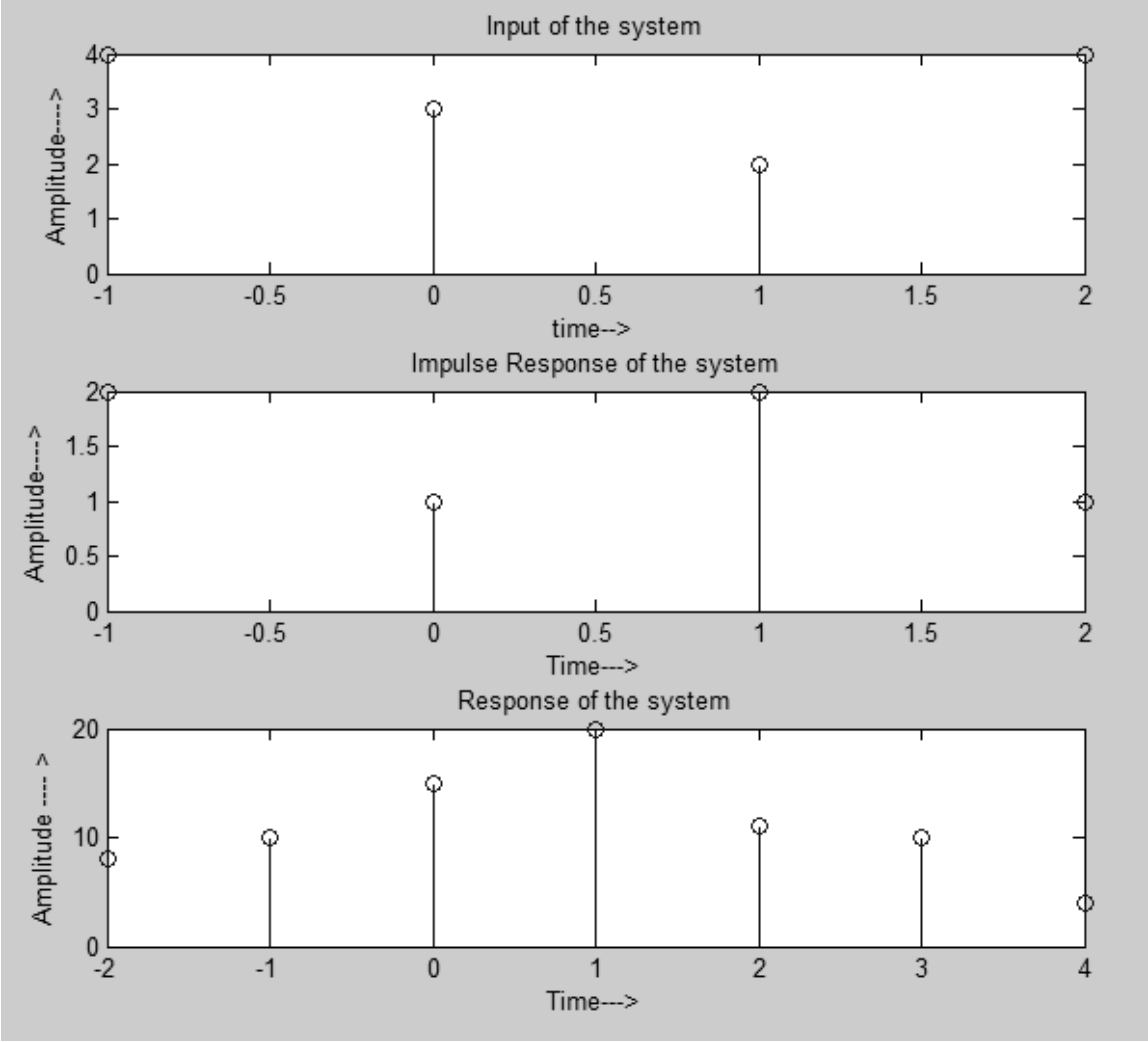
Enter Impulse Response [2 1 2 1]

OUTPUT of the system is:

y =

8 10 15 20 11 10 4

OUTPUT WAVEFORMS:



2.c) AIM: To Compute auto correlation of a discrete sequence

THEORY:

Autocorrelation can be represented by a Mathematical Expression as follows:

$$r_{xx}(l) = \sum_{n=-\infty}^{\infty} x(n)x(n-l) = r_{xx}(-l) \qquad l = 0, \pm 1, \pm 2, \ldots$$

x[n]= Input signal Samples

r_{xx}[n]= Correlated output.

ALGORITHM:-

- Read the input sequence x1[n]
- Compute Autocorrelation using function 'xcorr'
- Plot the input sequence and correlated output

MATLAB PROGRAM:

```
clc;
clear all;
close all;
x1=input('Enter the Sequence');
x2=xcorr(x1);
disp('Auto Correlation is:');
stem(x1);
 xlabel('Time--->');
ylabel('Amplitude----
>'); title('First
sequence');
figure; % Opens new blank figure window
stem(x2);
xlabel('Time--->');
ylabel('Amplitude----
>');
title('Auto Correlation sequence');
```

Enter the Sequence [1 2 3 4]

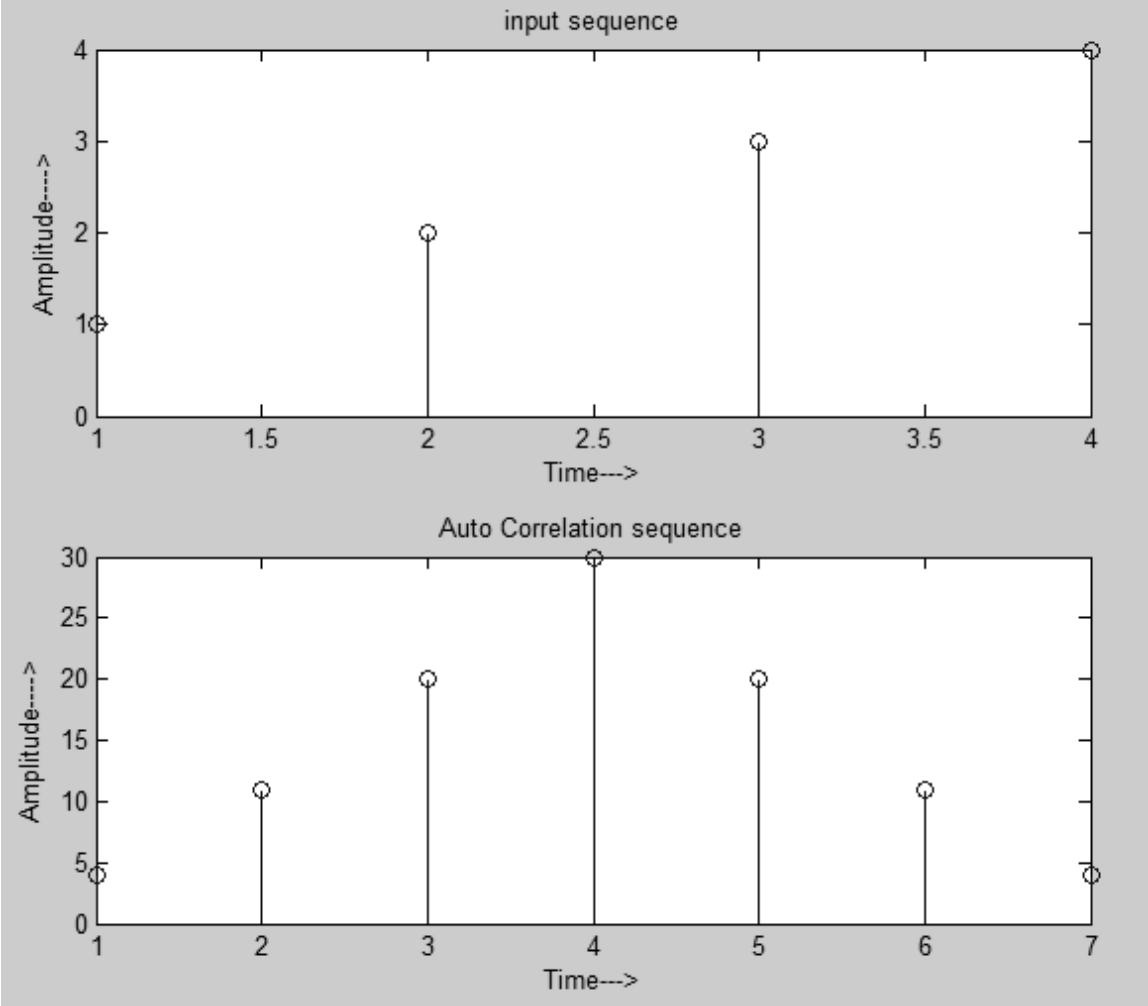
OUTPUT:

Auto Correlation is:

X2 =

4.0000 11.0000 20.0000 30.0000 20.0000 11.0000 4.0000

OUTPUT WAVEFORMS



2.c)AIM: To Compute Cross correlation of two discrete sequence

THEORY:

Autocorrelation can be represented by a Mathematical Expression as follows:

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n)\, y(n-l) \qquad l = 0, \pm 1, \pm 2, \ldots$$

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n-l)\, y(n) \qquad l = 0, \pm 1, \pm 2, \ldots$$

x[n]= Input sequence

y[n]= Second input sequence

r_{xy}[n]= Cross Correlated output.

ALGORITHM:-

- Read the input sequences x1[n] and x2[n]
- Compute cross correlation using function 'xcorr'
- Plot the input sequences and correlated output

MATLAB PROGRAM:

```
clc;
clear all;
close all;

x1=input('Enter First sequence');
x2=input('Enter second
sequence');

x3=xcorr(x1,x2);

disp(x3); % DISPLAYS THE VALUES OF x3

stem(x1);
xlabel('Time--->');
ylabel('Amplitude--->');
title('First sequence');

figure;
```

```
stem(x2);
xlabel('Time--->');
ylabel('Amplitude--->');
 title('Secondsequence');

figure;
stem(x3);
xlabel('Time--->');
ylabel('Amplitude----
>');
title('Cross Correlation sequence');
```

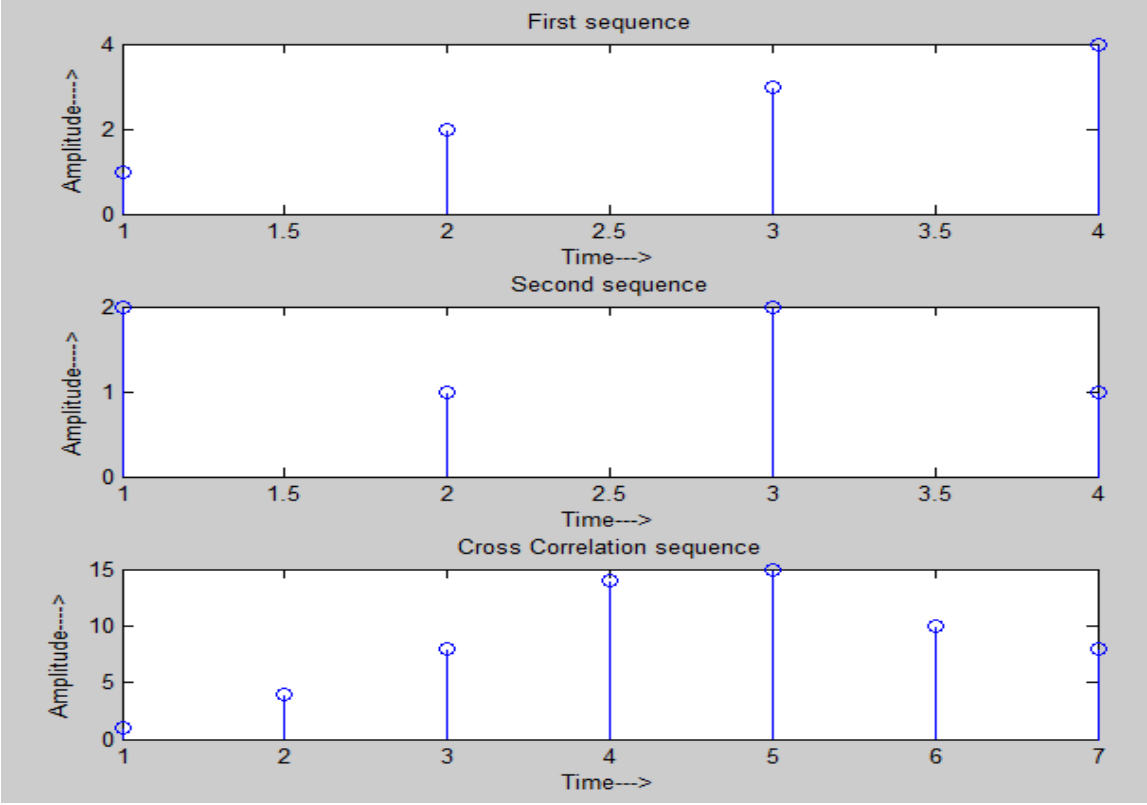
(INPUTS TO BE GIVEN IN COMMAND WINDOW)

Enter First sequence [1 2 3 4]
Enter second sequence [2 1 2 1]

OUTPUT:

```
x3 =

    1.0000  4.0000  8.0000  14.0000  15.0000  10.0000  8.0000
```

OUTPUT WAVEFORMS



EXPERIMENT- 03

CIRCULAR CONVOLUTION

3.a) AIM: To Compute Circular Convolution of two discrete sequence

SOFTWARE: Matlab R2014a

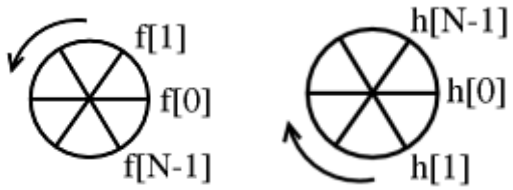
THEORY: x[n] and h[n] are two discrete sequences and circulation convolution is given by

$$w[n] = x[n] \ \textcircled{N} \ h[n].$$

$$= \sum_{m=0}^{N-1} x[m]h[((n-m))_N]$$

Steps for circular Convolution

Step1: "Plot f[m] and h[−m]



Subfigure 1.1 Subfigure 1.2

Step 2: "Spin" h[−m] *n* times Anti Clock Wise (counter-clockwise) to get h[n-m]

(i.e. Simply rotate the sequence, *h*[*n*], clockwise by *n* steps)

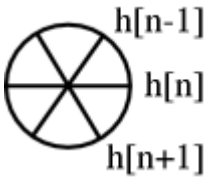
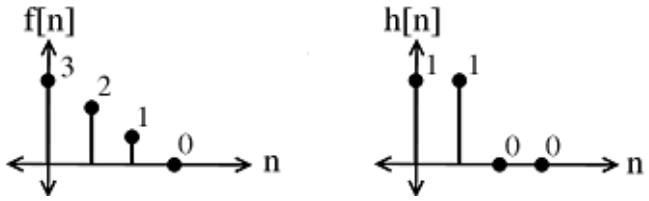


Figure 2: Step 2

Step 3: Pointwise multiply the $f[m]$ wheel and the h[n−m] wheel. sum=$y[n]$

Step 4: Repeat for all $0 \leq n \leq N-1$

Example 1: Convolve (n = 4)



Subfigure 3.1 Subfigure 3.2

Figure 3: Two discrete-time signals to be convolved.

- $h[-m]$ =

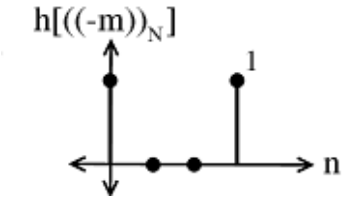


Figure 4

Multiply $f[m]$ and sum to yield: $y[0]$ =3

- $h[1-m]$

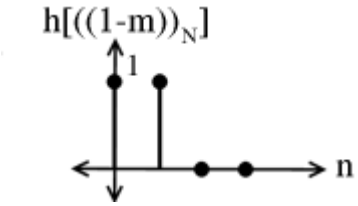


Figure 5

Multiply $f[m]$ and sum to yield: $y[1]$ =5

- $h[2-m]$

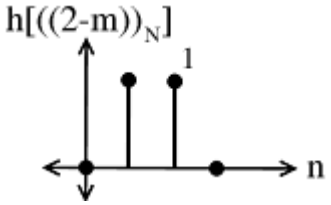


Figure 6

Multiply *f*[*m*] and sum to yield: *y*[2] =3

- *h*[3−*m*]

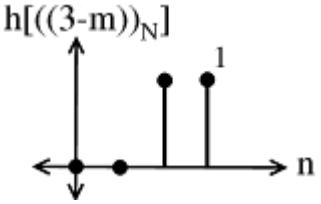


Figure 7

Multiply *f*[*m*] and sum to yield: *y*[3] =1

Matlab Program

```
clc;
 clear
all;
close all;
x1=input('enter the input sequence');
x2=input('Enter the second
sequence');
%To find the shorter sequence and it with zeros
n1=length(x1);
n2=length(x2)
; if (n1>n2)
   x2=[x2,zeros(1,n1-
n2)]; else
   x1=[x1,zeros(1,n2-
n1)]; end;
N=max(n1,n2);

%logic to find circular convolution
for n=1:N
   y(n)=0;
   for
   i=1:N
```

```
        j=n-i+1;
         if(j<=0)
           j=N+j;
        end;
        y(n)=y(n)+x1(i)*x2(j);
    end;
end;
figure;
stem(x1);
xlabel('Time--->');
ylabel('Amplitude');
title('First equence');
figure;
stem(x2);
xlabel('Time--->');
ylabel('Amplitude');
title('second sequence');
figure;
disp('The circular convolution is'); disp(y);
stem(y);
xlabel('Time--->');
ylabel('Amplitude')
;
title('Circular convolution sequence');
```

(INPUTS TO BE GIVEN IN COMMAND WINDOW)

Enter the input sequence [1 2 3 4]
Enter the second sequence [2 1 2 1]

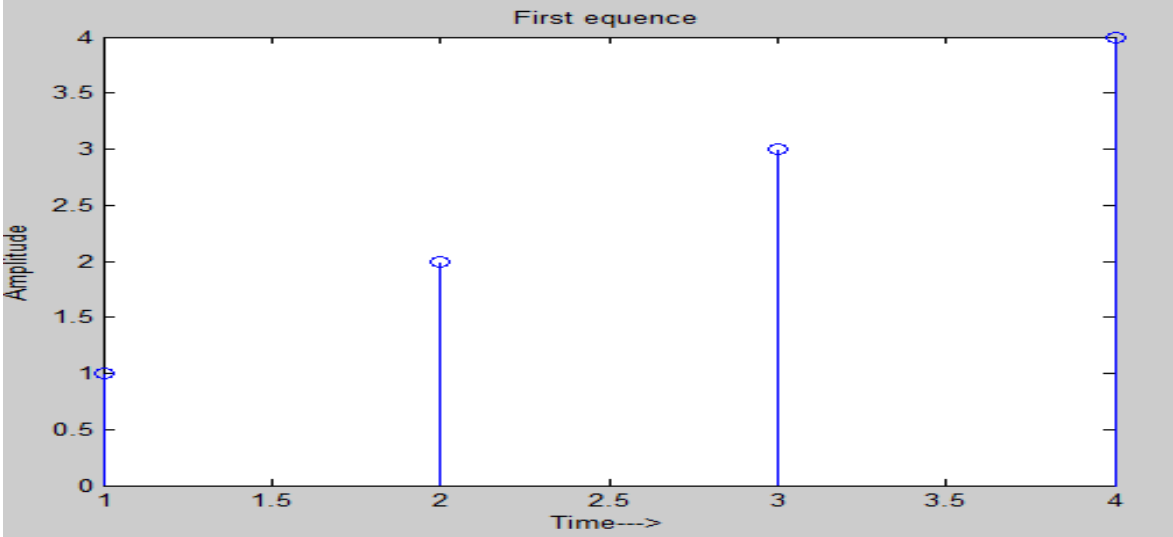
OUTPUT:

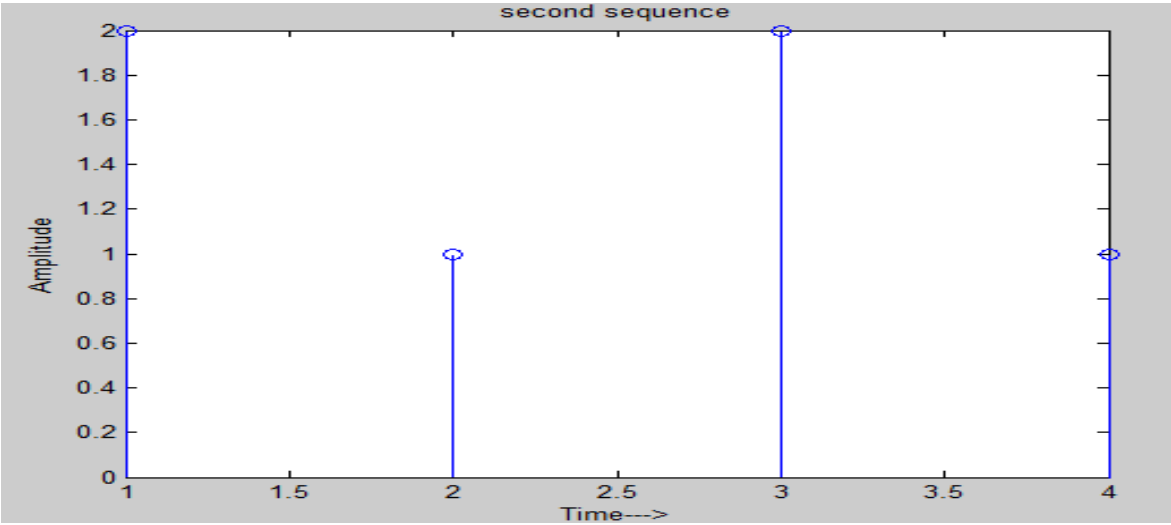
```
N =
    4
```

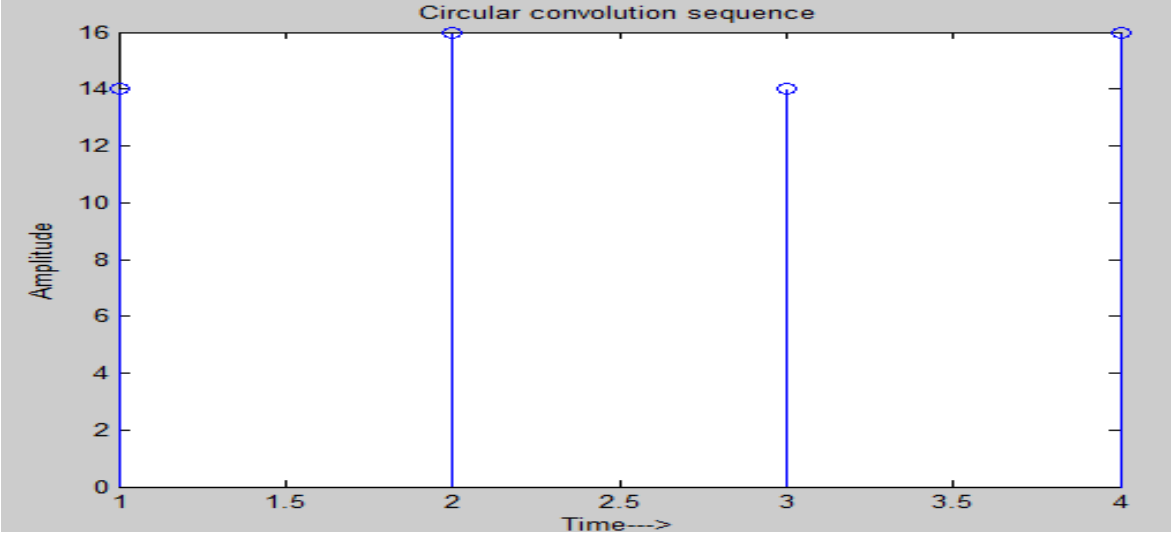
The circular convolution is

```
   14  16  14  16
```

OUTPUT WAVEFORMS







3.b)AIM:To perform linear convolution using circular convolution

SOTWARE: Matlab R2014a

THEORY: Circular convolution is only defined for finite length functions (usually, maybe always, equal in length), continuous or discrete in time. In circular convolution, it is as if the finite length functions repeat in time, periodically. Because the input functions are now periodic, the convolved output is also periodic and so the convolved output is fully specified by one of its periods.

If length of input sequence x_1 and x_2 is N_1 are N_2 resp, then length of linear convolution $N_1 + N_2 - 1$, and circular convolution is $\text{Max}(N_1, N_2)$.

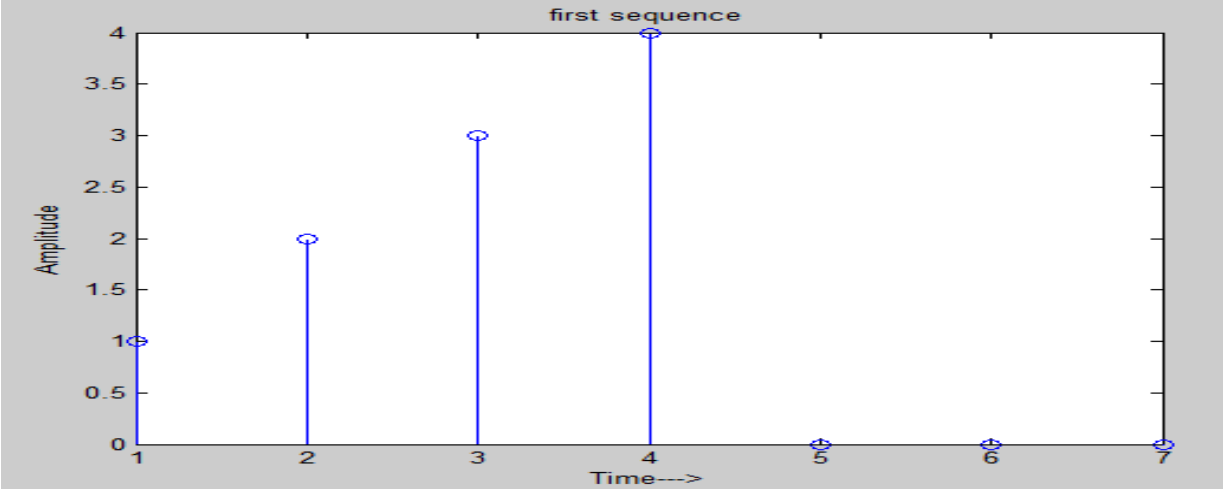
MATLAB PROGRAM:

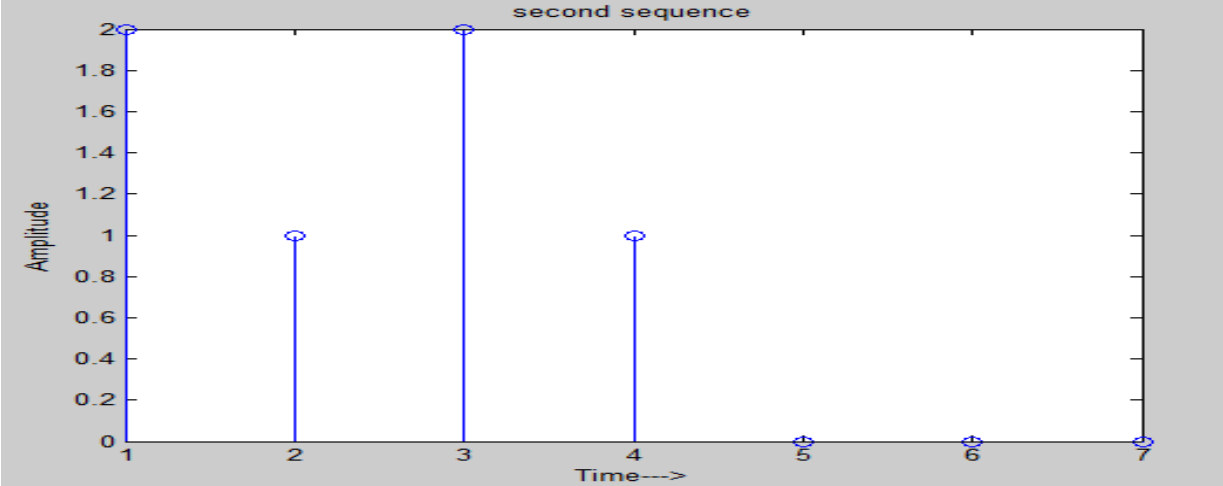
```
clc;
clear all;
close all;
x1=input('enter the first sequence');
x2=input('enter the secondsequence');
n1=length(x1);
n2=length(x2;
x1=[x1,zeros(1,n2-
1)] ;
x2=[x2,zeros(1,n1-
1)] ;
N=length(x1);
for n=1:N;
   y(n)=0;
   for
   i=1:N;
        j=n-i+1;
       if(j<=0);
          j=N+j;
       end
       y(n)=y(n)+x1(i)*x2(j);
   end
end
stem(x1);
figure;
stem(x2);
figure;
stem(y);
xlabel('Time--->');
ylabel('Amplitude')
;
title('Linear convolution sequence using circular convolution');
```

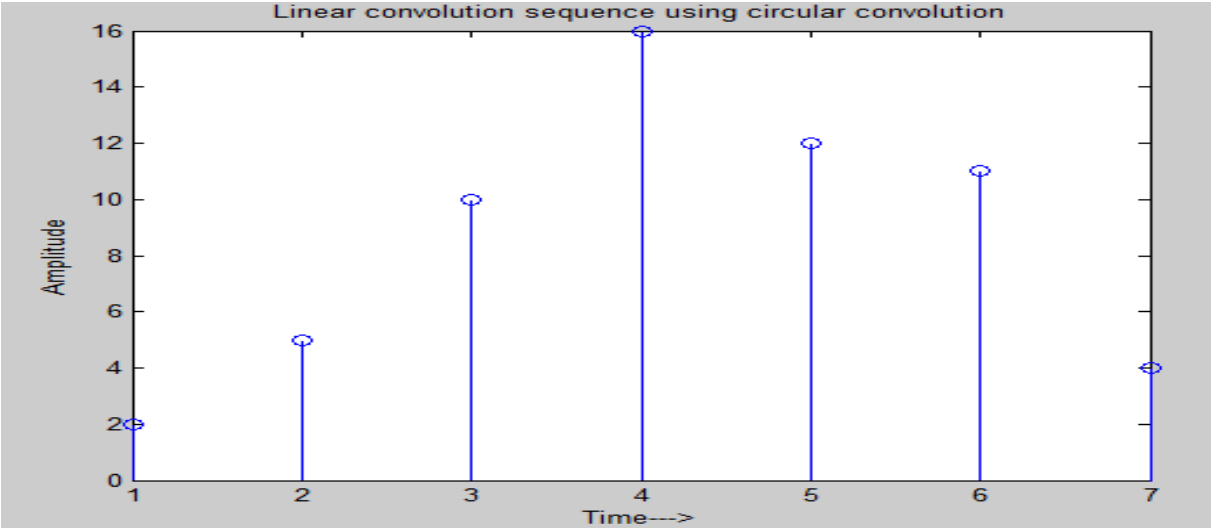
(INPUTS TO BE GIVEN IN COMMAND WINDOW)
enter the first sequence [1 2 3 4]
enter the second sequence [2 1 2 1]

OUTPUT:

```
x1 =

    1    2    3    4    0    0    0
x2 =

    2    1    2    1    0    0    0

k =
     Columns 1 through 13
        2    5   10   16   12   11    4    0    0    0    0    0    0
```






Linear convolution sequence using circular convolution
Amplitude
Time--->
0
2
4
6
8
10
12
14
16
1
2
3
4
5
6
7

EXPERIMENT- 04

FIR FILTER DESIGN USING DIFFERENT WINDOWS

4.a)AIM: To Plot the response of different window function.

SOFTWARE: Matlab R2014a

THEORY:

Name of window function	*Transition width* (Hz) (*normalized*)	*Passband ripple* (dB)	*Main lobe relative to side lobe* (dB)	*Stopband attenuation* (dB) (*maximum*)	*Window function* $w(n), \lvert n\rvert \leq (N-1)/2$
Rectangular	$0.9/N$	0.7416	13	21	1
Hanning	$3.1/N$	0.0546	31	44	$0.5 + 0.5\cos\left(\frac{2\pi n}{N}\right)$
Hamming	$3.3/N$	0.0194	41	53	$0.54 + 0.46\cos\left(\frac{2\pi n}{N}\right)$
Blackman	$5.5/N$	0.0017	57	75	$0.42 + 0.5\cos\left(\frac{2\pi n}{N-1}\right) + 0.08\cos\left(\frac{4\pi n}{N-1}\right)$

MATLAB PROGRAM:

```
clc;
clear all;
close all;
 n=65;
w1=window(@rectwin,n);
w2=window(@triang,n);
w3=window(@hamming,n);
w4=window(@hann,n);
w5=window(@blackman,n);
plot(1:n, [w1 w2 w3
w4,w5]); axis([1,n,0,1]);
legend('Rectangular window ','triangular window ','Hamming window
', 'hanning window' , 'Blackman window');
```

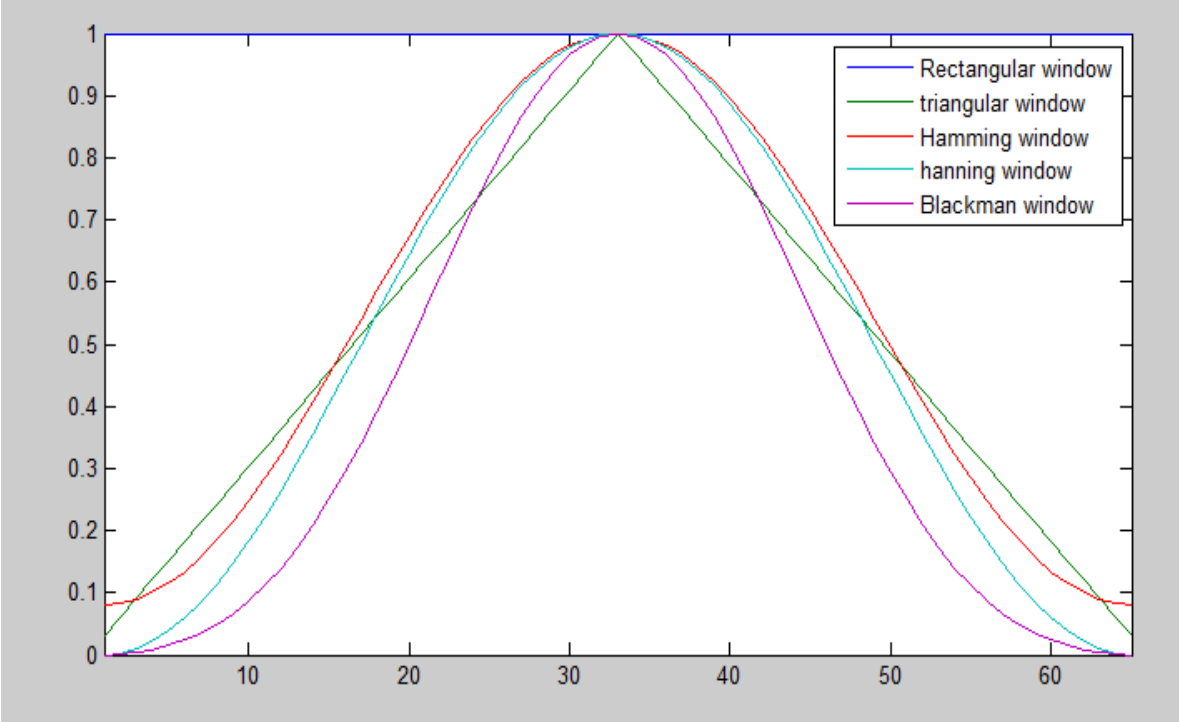


Fig: Characteristics of different window functions

4.b) AIM: To design FIR Filters using Window functions

SOFTWARE: Matlab R2014a

THEORY:

Consider an FIR Filter of Order M

$$y(n) = \sum_{k=0}^{M-1} b_k x(n-k) = \sum_{k=0}^{M-1} h(k)\, x(n-k)$$

windowed FIR filter design procedure:

1. Select a suitable window function
2. Specify an ideal response $H_d(\omega)$
3. Compute the coefficients of the ideal filter b_k
4. Multiply the ideal coefficients by the window function to give the filter coefficients
5. Evaluate the frequency response of the resulting filter

Summary of Ideal Impulse Responses for Standard FIR Filters

Filter Type Ideal Impulse Response $h(n)$ (noncausal FIR coefficients)

Lowpass:
$$h(n) = \begin{cases} \frac{\Omega_c}{\pi} & \text{for } n = 0 \\ \frac{\sin(\Omega_c n)}{n\pi} & \text{for } n \neq 0 \end{cases} \quad -M \le n \le M$$

Highpass:
$$h(n) = \begin{cases} \frac{\pi - \Omega_c}{\pi} & \text{for } n = 0 \\ -\frac{\sin(\Omega_c n)}{n\pi} & \text{for } n \neq 0 \end{cases} \quad -M \le n \le M$$

Bandpass:
$$h(n) = \begin{cases} \frac{\Omega_H - \Omega_L}{\pi} & \text{for } n = 0 \\ \frac{\sin(\Omega_H n)}{n\pi} - \frac{\sin(\Omega_L n)}{n\pi} & \text{for } n \neq 0 \end{cases} \quad -M \le n \le M$$

Bandstop:
$$h(n) = \begin{cases} \frac{\pi - \Omega_H + \Omega_L}{\pi} & \text{for } n = 0 \\ -\frac{\sin(\Omega_H n)}{n\pi} + \frac{\sin(\Omega_L n)}{n\pi} & \text{for } n \neq 0 \end{cases} \quad -M \le n \le M$$

Causal FIR filter coefficients: shifting $h(n)$ to the right by M samples.
Transfer function:
$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots b_{2M} z^{-2M}$
where $b_n = h(n-M), n = 0, 1, \cdots, 2M.$

ALGORITHM:

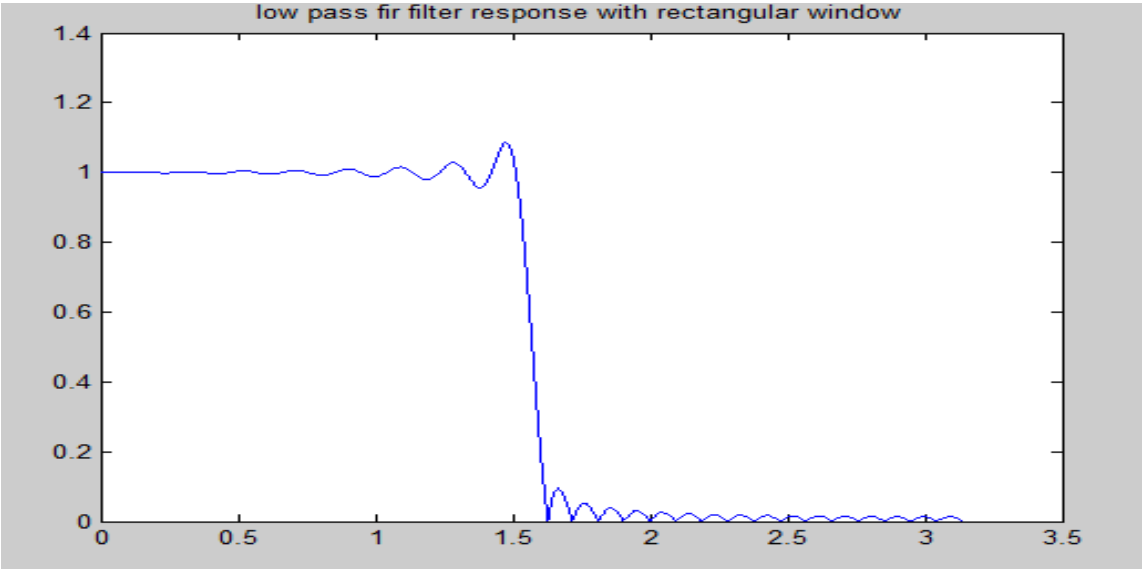
- Get the order of the filter
- Get the cut off frequency
- use ‘ fir1 ’& specific function to compute the filter coefficient
- Draw the magnitude and phase response

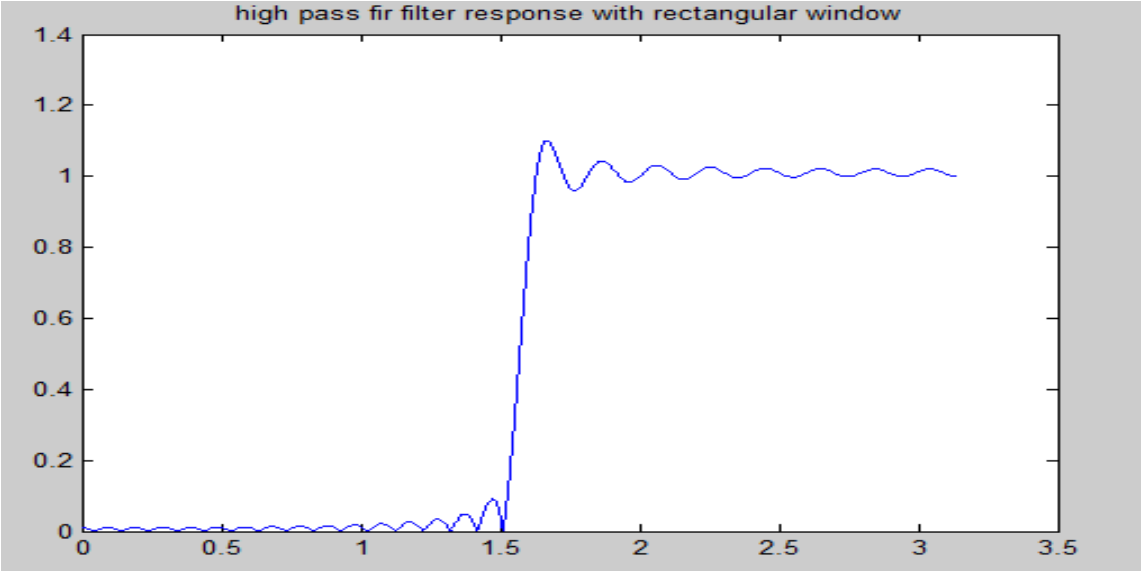
MATLAB PROGRAM:

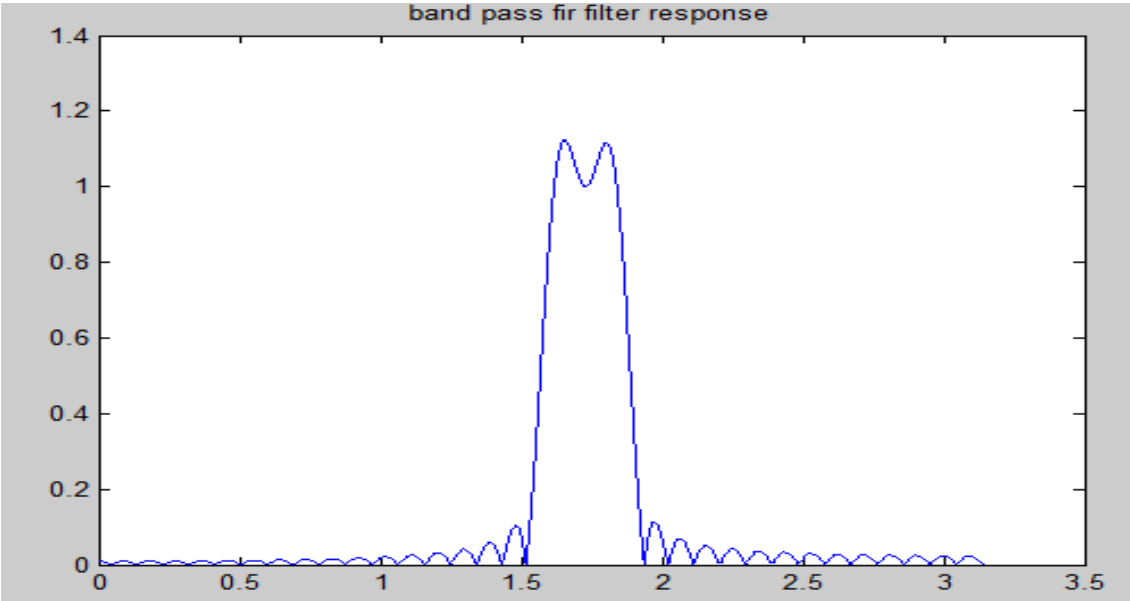
FIR FILTERS USING RECTANGULAR WINDOWS

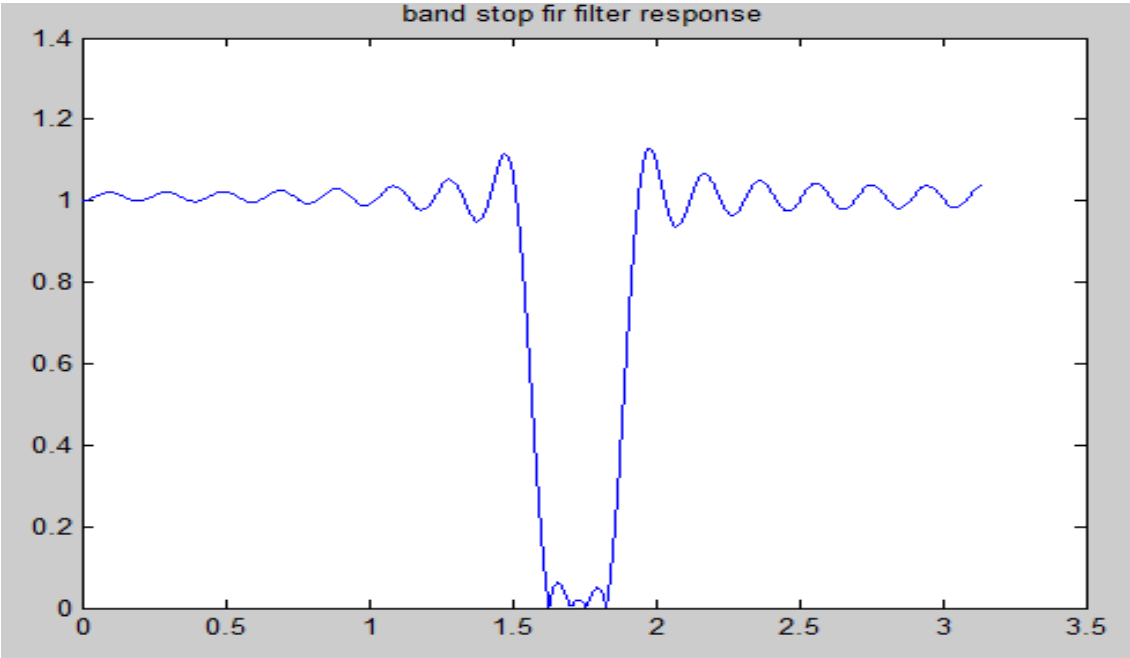
```
clc;
clear all;
close all;

% LPF WITH CUTOFF FREQUENCY 0.5pi ANDORDER =65
N=65;
wc=.5*pi;
b=fir1(N,(wc/pi),rectwin(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
plot(w,abs(H));
title('low pass fir filter response with rectangular window ');

% HPF WITH CUTOFF FREQUENCY 0.5pi AND ORDER
=64N=64;
wc=.5*pi;
b=fir1(N,(wc/pi),'high',rectwin(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
figure;
plot(w,abs(H));
title('high pass fir filter response');


% BPF WITH CUTOFF FREQUENCIES 0.5pi & 0.6pi AND ORDER = 65
N=65;
wc1=.5*pi;
wc2=.6*pi ;
b=fir1(N,[wc1/pi
wc2/pi],'bandpass',rectwin(N+1));
 w=0:.001:pi;
H=freqz(b,1,w);
figure;
plot(w,abs(H));
title('band pass fir filter response');
```

```
% BRF WITH CUTOFF FREQUENCIES 0.5pi & 0.6pi AND ORDER = 64
N=64;
wc1=.5*pi;
wc2=.6*pi
b=fir1(N,[wc1/pi
wc2/pi],'stop',rectwin(N+1)); w=0:.001:pi;
H=freqz(b,1,w);
plot(w,abs(H));
title('band stop fir filter response');
```



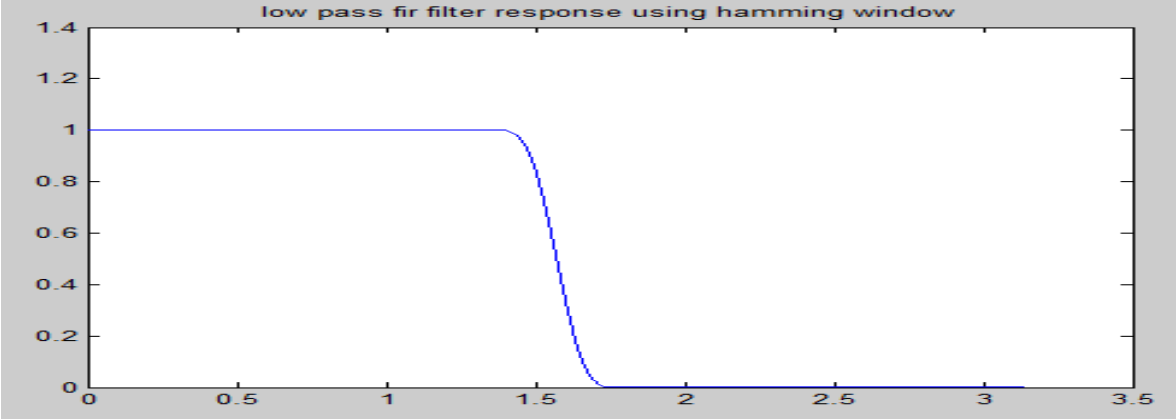


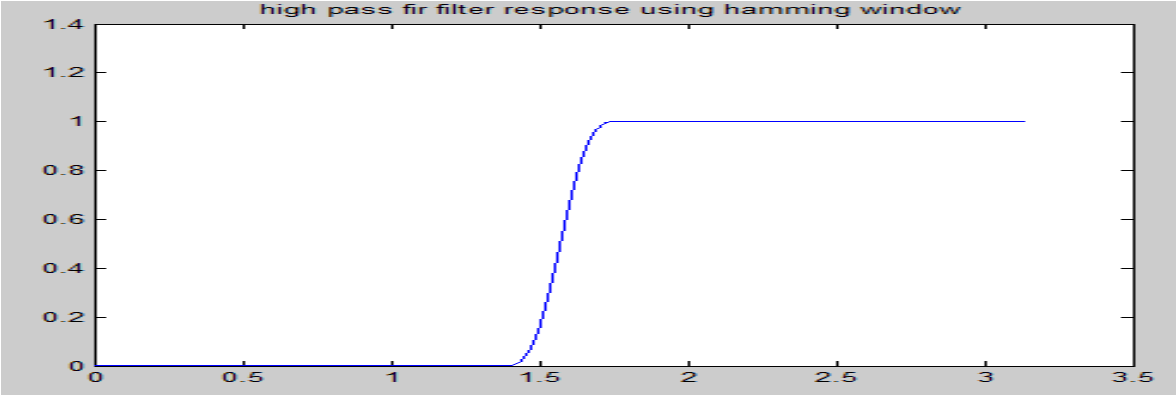

band pass fir filter response
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5

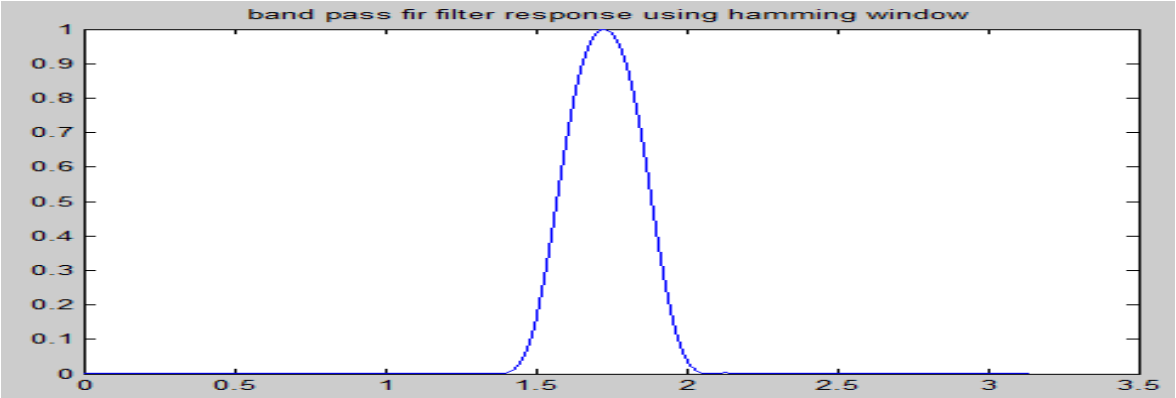

band stop fir filter response
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5

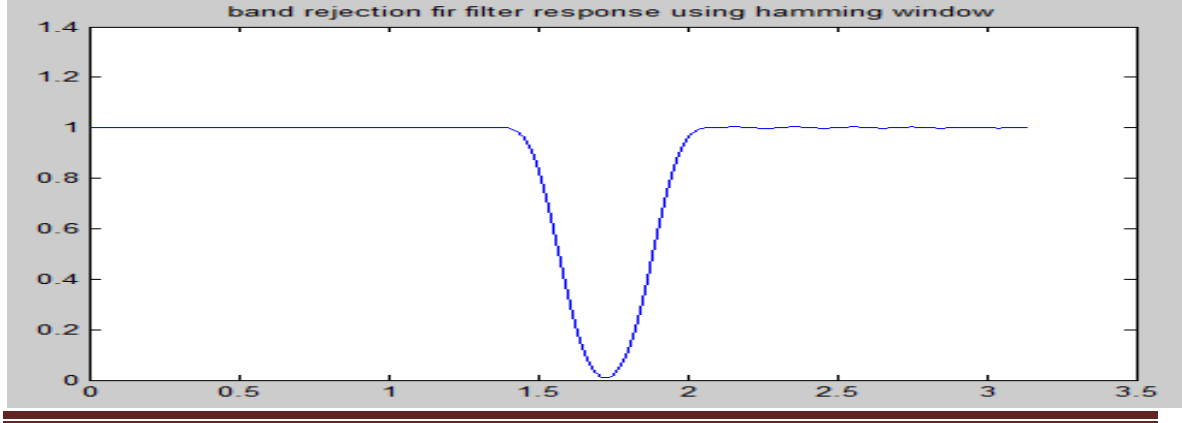
% FIR FILTERS USING HAMMING WINDOWS

```
clc;
clear all;
close all;

% LPF WITH CUTOFF FREQUENCY 0.5pi AND ORDER = 65
 N=65;
wc=.5*pi;
b=fir1(N,(wc/pi),hamming(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
plot(w,abs(H));
title('low pass fir filter response using hamming window');

% HPF WITH CUTOFF FREQUENCY 0.5pi AND ORDER =
64 N=64;
wc=.5*pi;
b=fir1(N,(wc/pi),'high',hamming(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
figure;
plot(w,abs(H));
title('high pass fir filter response using hamming window');


% BPF WITH CUTOFF FREQUENCIES 0.5pi & 0.6pi AND ORDER = 65
N=65;
wc1=.5*pi;
wc2=.6*pi
b=fir1(N,[wc1/pi wc2/pi],'bandpass',hamming(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
figure;
plot(w,abs(H));
title('band pass fir filter response using hamming window');

% BRF WITH CUTOFF FREQUENCIES 0.5pi & 0.6pi AND ORDER = 64
 N=64;
wc1=.5*pi;
wc2=.6*pi
b=fir1(N,[wc1/pi
wc2/pi],'stop',hamming(N+1)); w=0:.001:pi;
H=freqz(b,1,w);
plot(w,abs(H));
title('band rejection fir filter response using hamming window');
```


low pass fir filter response using hamming window
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5

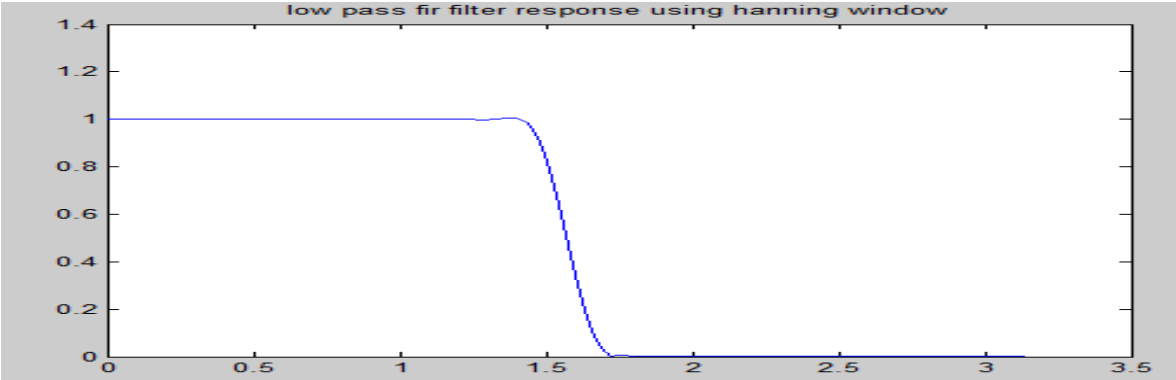

high pass fir filter response using hamming window
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5

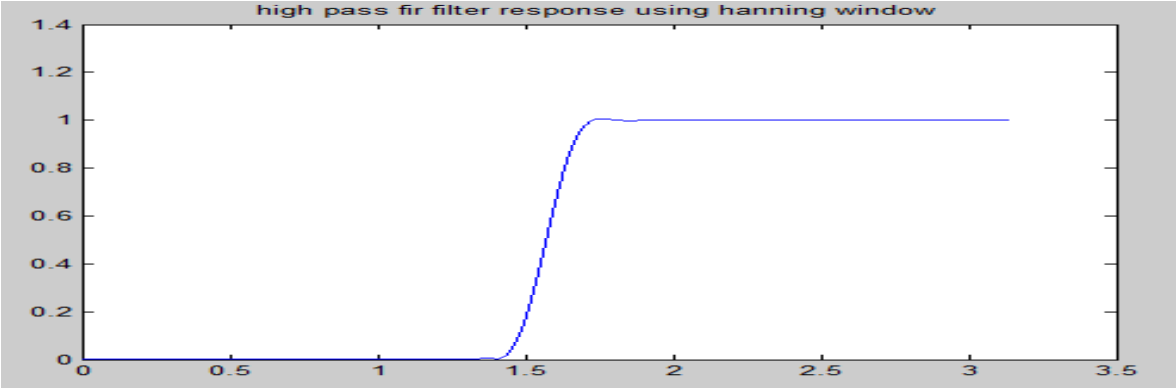

band pass fir filter response using hamming window
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5

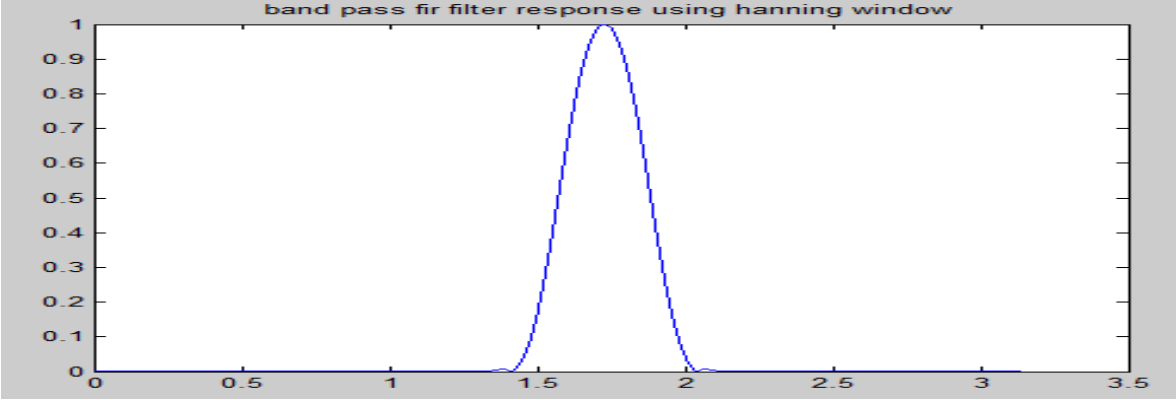

band rejection fir filter response using hamming window
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5

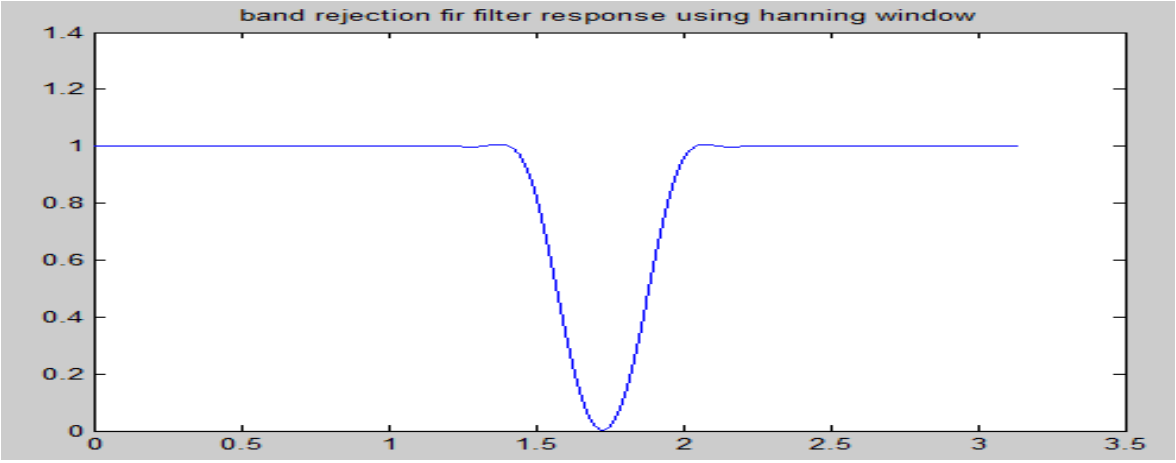
% FIR FILTERS USING HANNING WINDOWS

```
clc;
clear all;
close all;

% LPF WITH CUTOFF FREQUENCY 0.5pi AND ORDER = 65
N=65;
wc=.5*pi;
b=fir1(N,(wc/pi), hanning(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
plot(w,abs(H));
title('low pass fir filter response using hanning window');

% HPF WITH CUTOFF FREQUENCY 0.5pi AND ORDER =
64 N=64;
wc=.5*pi;
b=fir1(N,(wc/pi),'high', hanning(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
figure;
plot(w,abs(H));
title('high pass fir filter response using hanning window');

% BPF WITH CUTOFF FREQUENCIES 0.5pi & 0.6pi AND ORDER = 65
 N=65;
wc1=.5*pi;
wc2=.6*pi
b=fir1(N,[wc1/pi wc2/pi],'bandpass', hanning(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
figure;
plot(w,abs(H));
title('band pass fir filter response using hanning window');

% BRF WITH CUTOFF FREQUENCIES 0.5pi & 0.6pi AND ORDER = 64
N=64;
wc1=.5*pi;
wc2=.6*pi
b=fir1(N,[wc1/pi wc2/pi],'stop', hanning(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
plot(w,abs(H));
title('band rejection fir filter response using hanning window');
```


low pass fir filter response using hanning window
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5

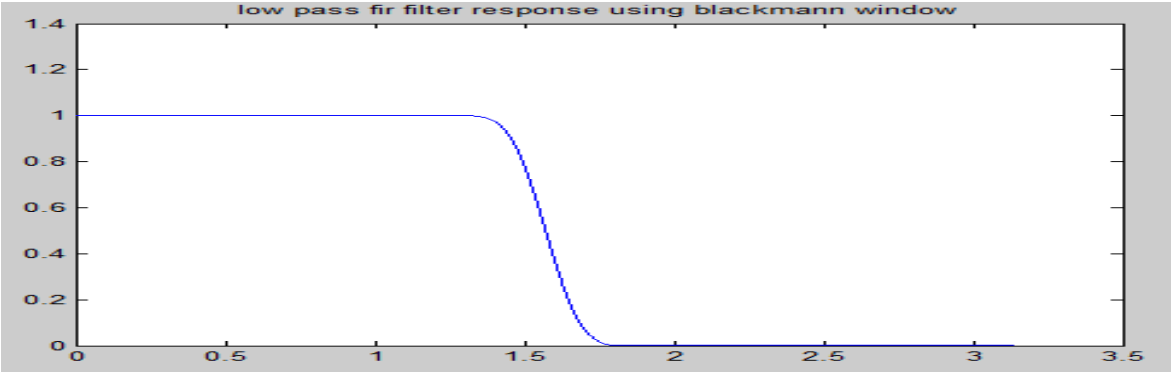

high pass fir filter response using hanning window
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5

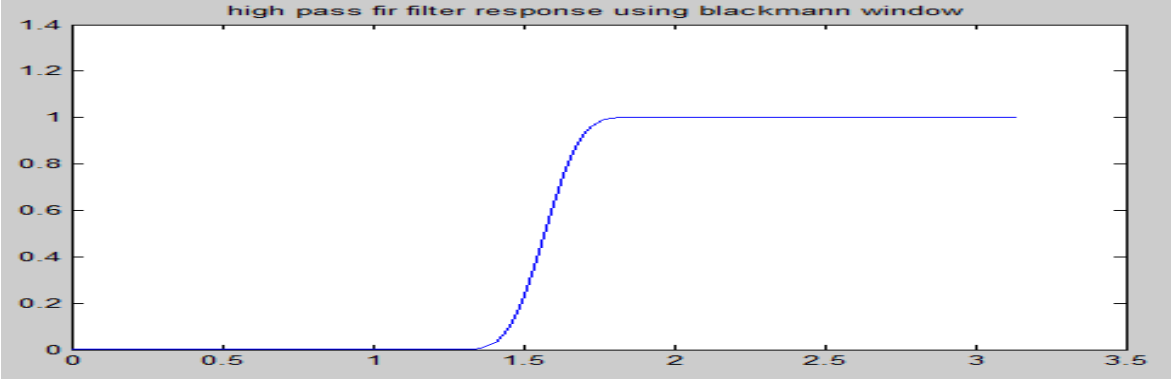

band pass fir filter response using hanning window
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5

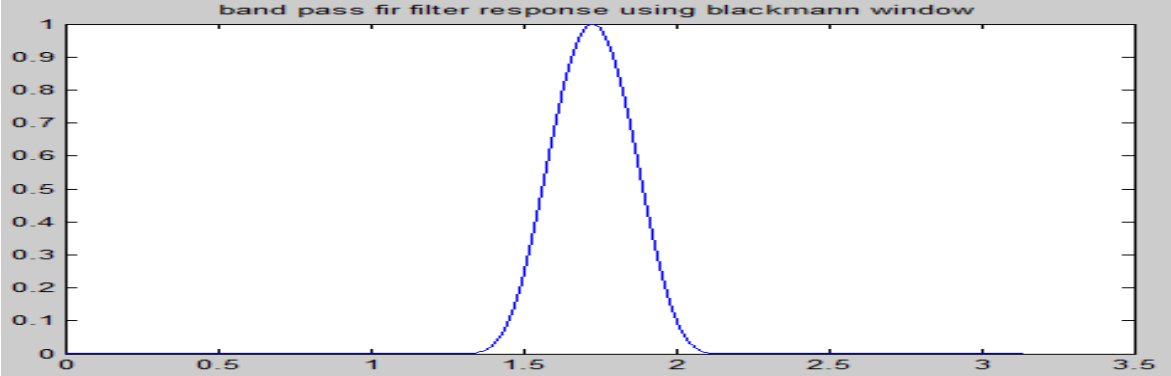

band rejection fir filter response using hanning window
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5

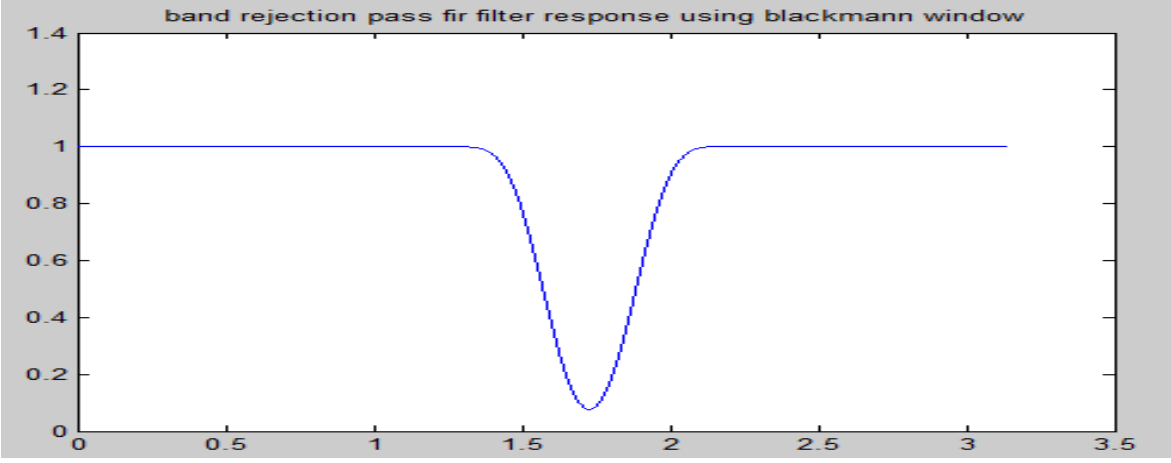
% FIR FILTERS USING BLACKMANN WINDOW

```
clc;
clear all;
close all;

% LPF WITH CUTOFF FREQUENCY 0.5pi AND ORDER = 65
 N=65;
wc=.5*pi;
b=fir1(N,(wc/pi),blackman(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
plot(w,abs(H));
title('low pass fir filter response using blackmann window');

% HPF WITH CUTOFF FREQUENCY 0.5pi AND ORDER =
64 N=64;
wc=.5*pi;
b=fir1(N,(wc/pi),'high',blackman(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
figure;
plot(w,abs(H));
title('high pass fir filter response using blackmann window');


% BPF WITH CUTOFF FREQUENCIES 0.5pi & 0.6pi AND ORDER = 65
 N=65;
wc1=.5*pi;
wc2=.6*pi
b=fir1(N,[wc1/pi wc2/pi],'bandpass',blackman(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
figure;
plot(w,abs(H));
title('band pass fir filter response using blackmann window');

% BRF WITH CUTOFF FREQUENCIES 0.5pi & 0.6pi AND ORDER = 64
 N=64;
wc1=.5*pi;
wc2=.6*pi
b=fir1(N,[wc1/pi wc2/pi],'stop',blackman(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
plot(w,abs(H));
title('band rejection pass fir filter response using blackmann window');
```


low pass fir filter response using blackmann window
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5

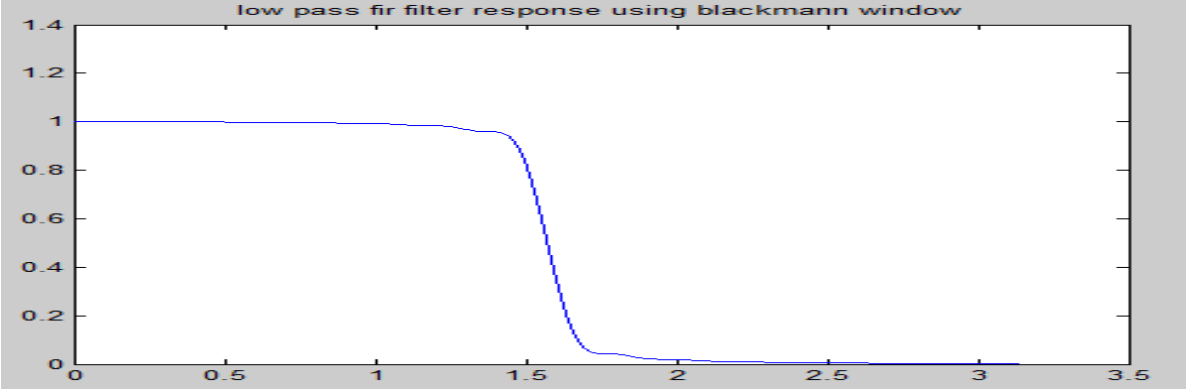

high pass fir filter response using blackmann window
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5

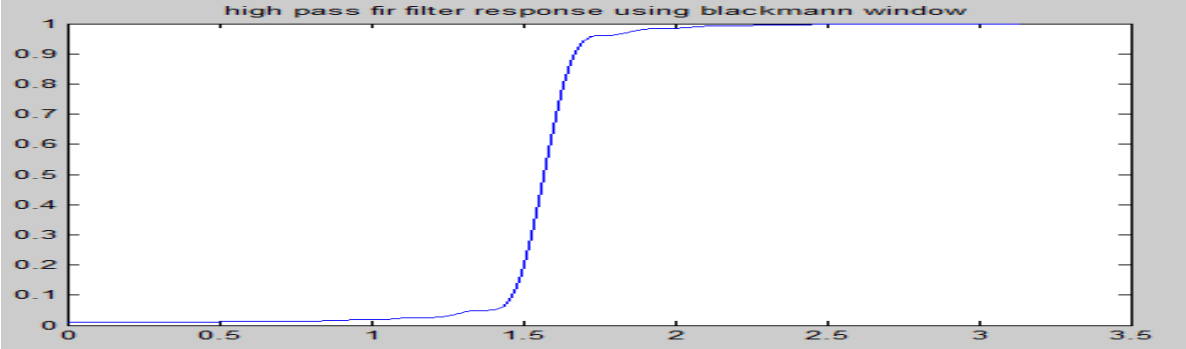

band pass fir filter response using blackmann window
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5

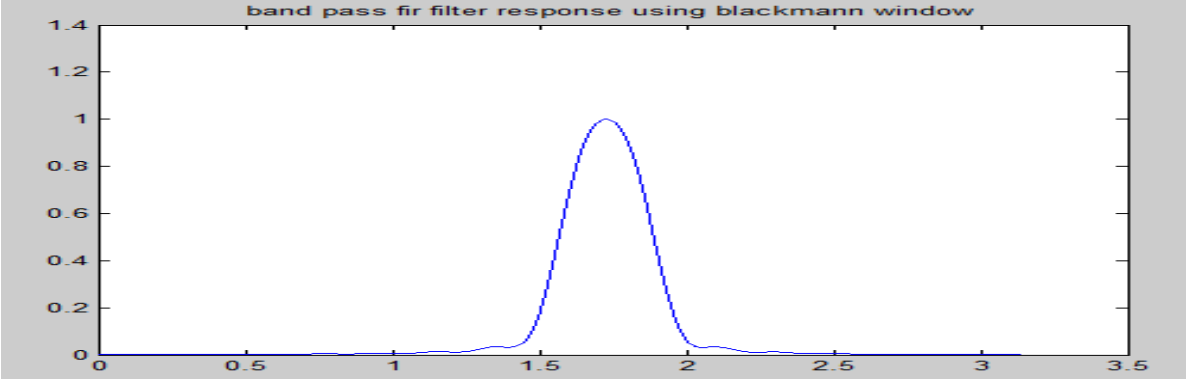

band rejection pass fir filter response using blackmann window
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5

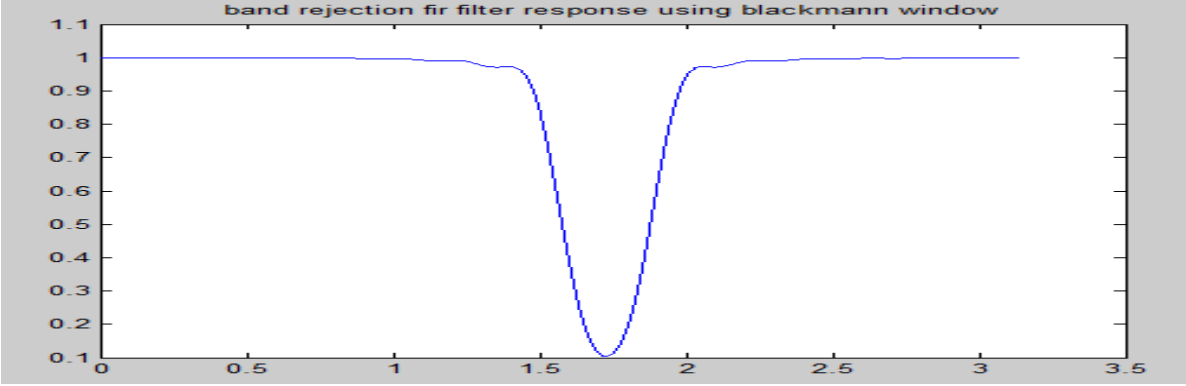
% FIR FILTERS USING BARTLETT WINDOW

```
clc;
clear all;
close all;

% LPF WITH CUTOFF FREQUENCY 0.5pi AND ORDER = 65
 N=65;
wc=.5*pi;
b=fir1(N,(wc/pi),bartlett(N+1));
w=0:.001:pi;
H=freqz(b,1,w);
plot(w,abs(H));
title('low pass fir filter response using blackmann window');

% HPF WITH CUTOFF FREQUENCY 0.5pi AND ORDER = 64
N=64;
wc=.5*pi;
b=fir1(N,(wc/pi),'high', bartlett (N+1));
w=0:.001:pi;
H=freqz(b,1,w);
figure;
plot(w,abs(H));
title('high pass fir filter response using blackmann window');


% BPF WITH CUTOFF FREQUENCIES 0.5pi & 0.6pi AND ORDER = 65
N=65;
wc1=.5*pi;
wc2=.6*pi
b=fir1(N,[wc1/pi wc2/pi],'bandpass', bartlett (N+1));
w=0:.001:pi;
H=freqz(b,1,w);
figure;
plot(w,abs(H));
title('band pass fir filter response using blackmann window');

% BRF WITH CUTOFF FREQUENCIES 0.5pi & 0.6pi AND ORDER = 64
N=64;
wc1=.5*pi;
wc2=.6*pi
b=fir1(N,[wc1/pi wc2/pi],'stop', bartlett (N+1));
w=0:.001:pi;
H=freqz(b,1,w);
plot(w,abs(H));
title('band rejection fir filter response using blackmann window');
```


low pass fir filter response using blackmann window
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5


high pass fir filter response using blackmann window
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5


band pass fir filter response using blackmann window
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5


band rejection fir filter response using blackmann window
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.5
1
1.5
2
2.5
3
3.5

EXPERIMENT- 05

IIR FILTER DESIGN

AIM: To Write a Matlab Program to design an IIR Filter.
SOFTWARE: Matlab R2014a
THEORY:

IIR filter equation

$$y[n] = a_0 x[n] + a_1 x[n-1] + \cdots + a_{M-1} x[n-(M-1)]$$
$$- b_1 y[n-1] - b_2 y[n-2] - \cdots - b_N y[n-N]$$

IIR FILTER DESIGN STEPS:

- Choose prototype analog filter family
  -Butterworth
  -Chebychev Type-I or II
- Choose analog –digital filter transformation method
  - Impluse invariance
  - Bilinear Transformation
- Transform Digtial filter specifications to equivalent analog filter specifations
- Design analog filter
- Transform analog filter to digital filter
- Perform frequency transformation to achieve highpass or bandpass filter,if desired

ALGORITHM:

- Get the passband and stopband ripples
- Get the passband and stopband edge frequencies
- Calculate the order of the filter using ‘ buttord ’ function
- Find the filter coefficients, using ‘butter’ function
- Draw the magnitude and phase response

BUTTERWORTH LOW PASS FILTER

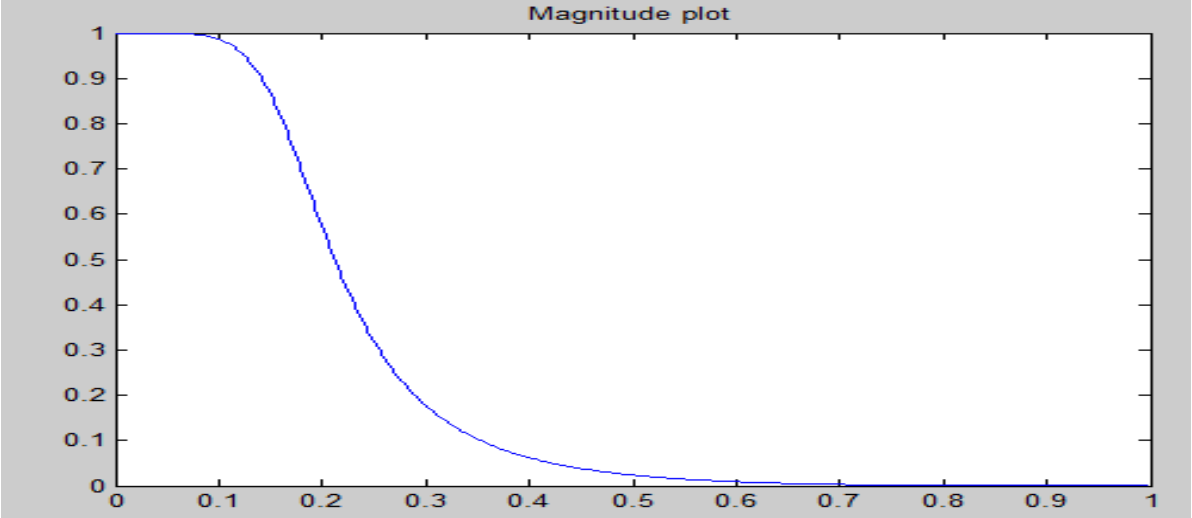
```
clc;
clear all;
close all;
rp=input('Enter the pass band in db');
rs=input('Enter the stop band attenuation in
db');
fp=input('Enter the pass band cuttoff frequency in
hz'); fs=input('Enter the stop band cutoff frequency in
hz'); f=input('Enter the sampling frequency');

%Normalising frequency
```

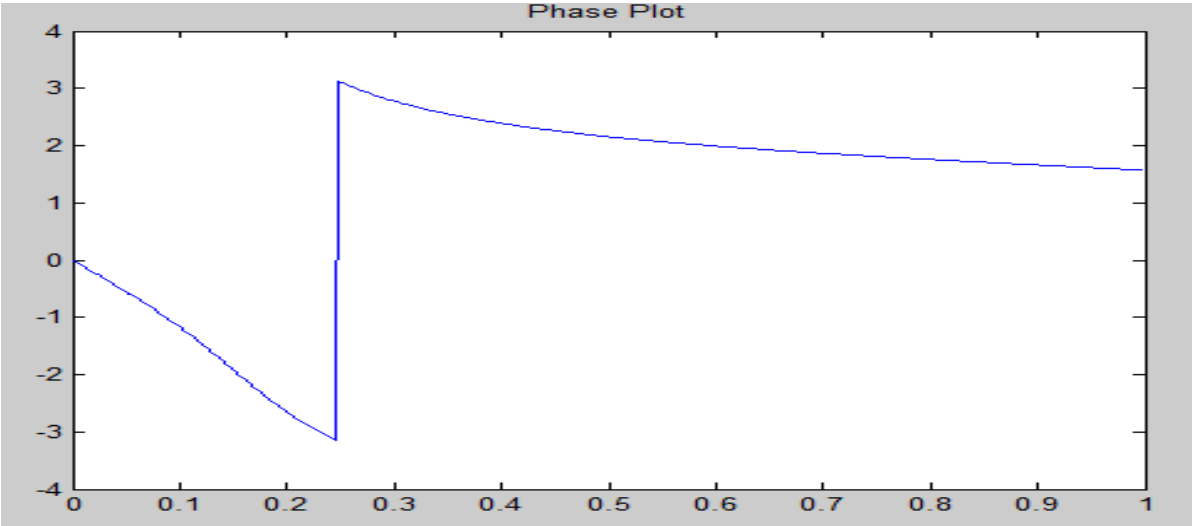
```
wp=2*fp/f
ws=2*fs/f

% Order of filter & 3db cuttoff frequency
[N,wc]=buttord(wp,ws,rp,rs); disp('order
of the filter =');N
disp('cutoff frequency');wc

%coefficients of the filter
[b,a]=butter(N,wc,'low');

%Frequency Response
[H,om]=freqz(b,a);

%magnitude plot
plot(om/pi,abs(H));
title('Magnitude plot');
figure;
plot(om/pi,angle(H));
title('Phase Plot');

%magnitude plot in db figure;
plot(om/pi,20*log10(abs(H)));
title('Magnitude plot in db');
```

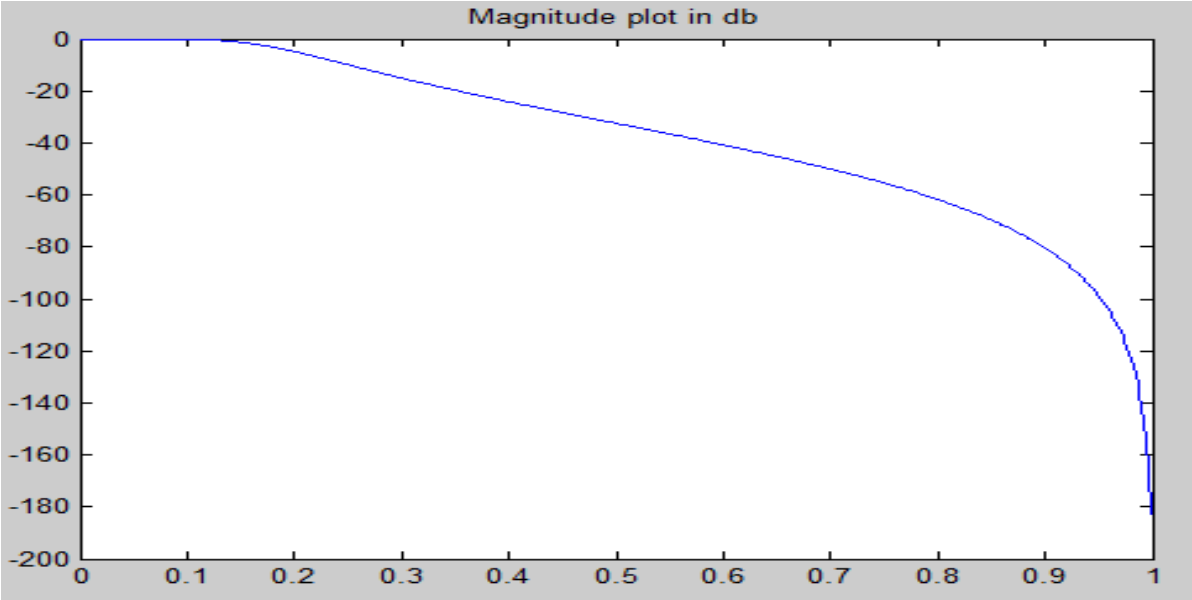
(INPUTS TO BE GIVEN IN COMMAND WINDOW)
Enter the pass band in db5
Enter the stop band attenuation in db15
Enter the pass band cuttoff frequency in hz1000 Enter the stop band cutoff frequency in hz1500 Enter the sampling frequency10000

OUTPUT:

```
wp =
   0.2000
ws =
   0.3000
order of the filter =
N =
    3
cutoff frequency
wc =
   0.1786
```


Magnitude plot
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1


Phase Plot
4
3
2
1
0
-1
-2
-3
-4
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1


Magnitude plot in db
0
-20
-40
-60
-80
-100
-120
-140
-160
-180
-200
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

% **BUTTERWORTH HIGH PASS FILTER**

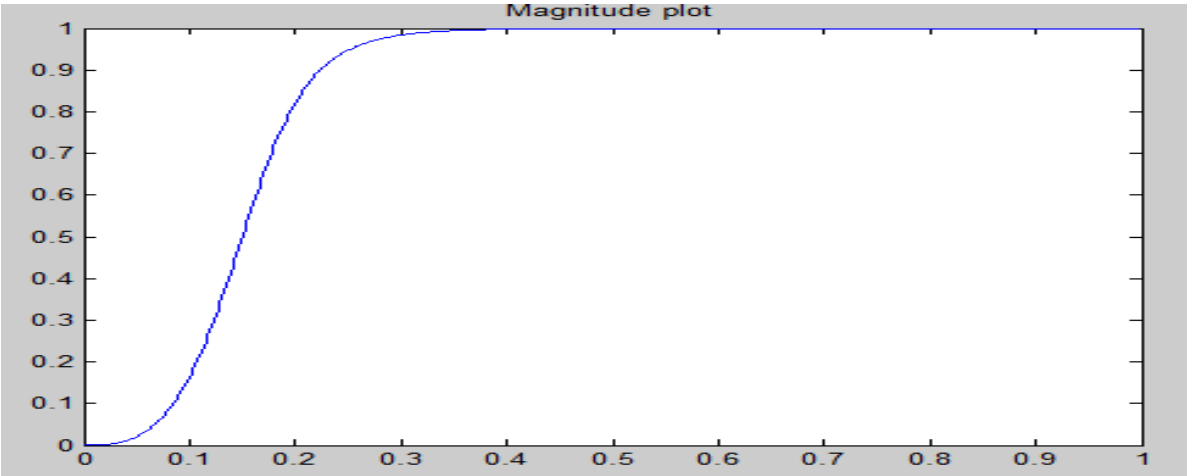
```
clc;
clear all;
close all;
rp=input('Enter the pass band in db');
rs=input('Enter the stop band attenuation indb');
fp=input('Enter the pass band cuttoff frequency in
hz'); fs=input('Enter the stop band cutoff frequency in
hz'); f=input('Enter the sampling frequency');
%Normalising frequency
wp=2*fp/f
ws=2*fs/f
% Order of filter 3db & cuttoff frequency
[N,wc]=buttord(wp,ws,rp,rs);
disp('order of the filter =');N
disp('cutoff frequency');wc

%coefficients of the filter
[b,a]=butter(N,wc,'high');
%Frequency Response
[H,om]=freqz(b,a);

%magnitude plot
plot(om/pi,abs(H));
title('Magnitude plot');
figure;
plot(om/pi,angle(H));
title('Phase Plot');

%magnitude plot in db figure;
plot(om/pi,20*log10(abs(H)));
title('Magnitude plot in db');
```

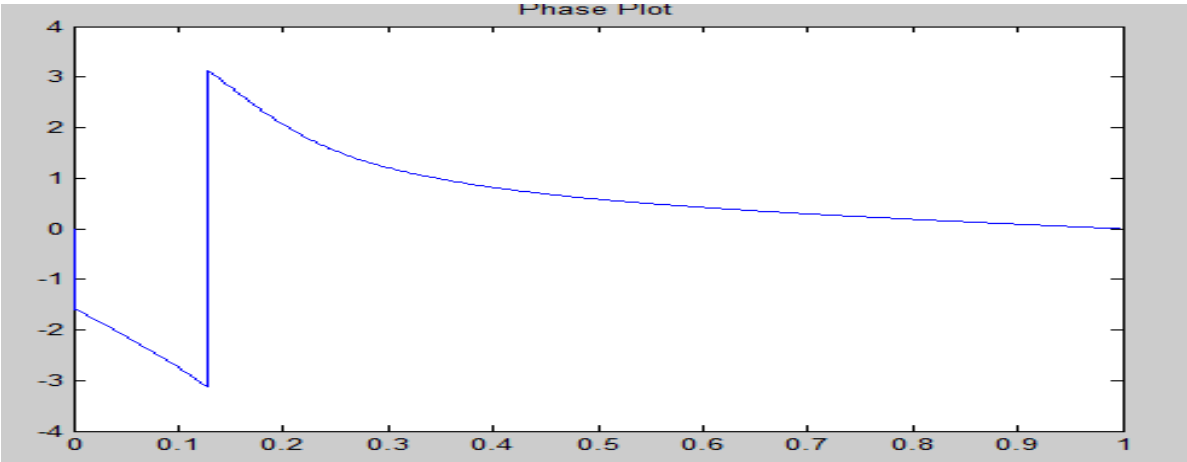
(INPUTS TO BE GIVEN IN COMMAND WINDOW)
Enter the pass band in db 5
Enter the stop band attenuation in db 15
Enter the pass band cuttoff frequency in hz
1500 Enter the stop band cutoff frequency in hz
1000 Enter the sampling frequency 10000

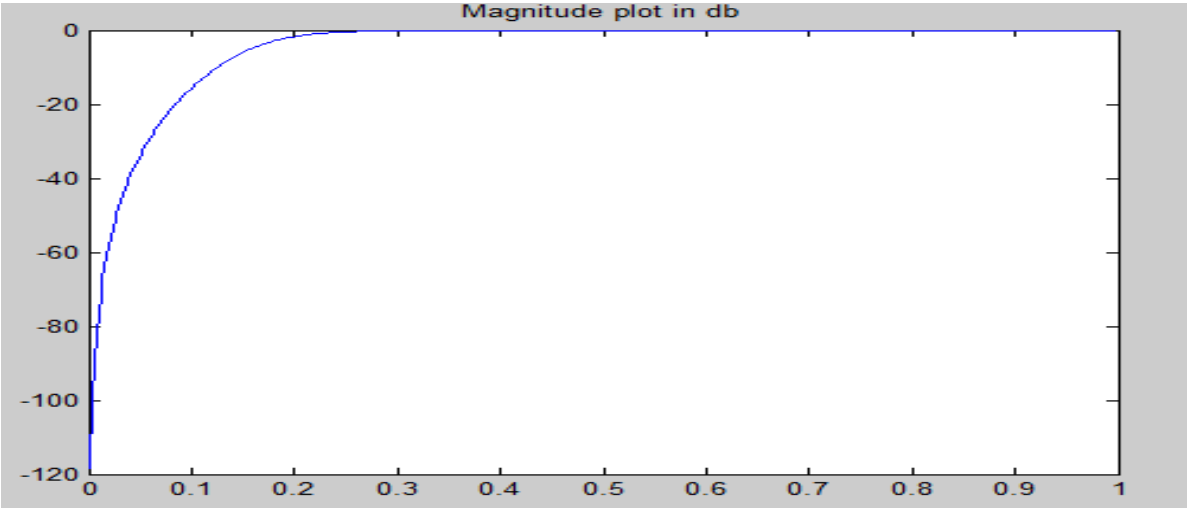
OUTPUT:

```
wp
=0.3000
ws =
   0.2000
order of the filter =
```

N =
3
cutoff frequency
wc =
0.3321







% CHEBYSHEVE TYPE – 1 LOW PASS FILTER

ALGORITHM:

- Get the passband and stopband ripples
- Get the passband and stopband edge frequencies
- Calculate the order of the filter using ‘ cheb1ord ’ function
- Find the filter coefficients, using ‘cheby1’ function
- Draw the magnitude and phase response

MATLAB PROGRAM:

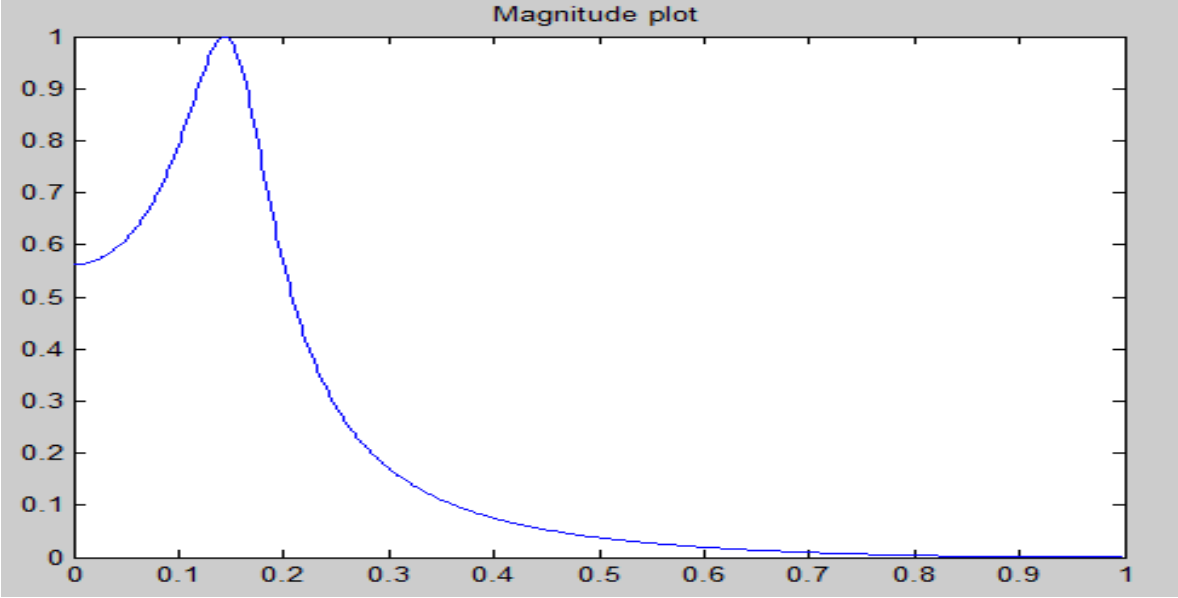
```
clc;
clear all;
close all;
rp=input('Enter the pass band in db');
rs=input('Enter the stop band attenuation in
db');
fp=input('Enter the pass band cuttoff frequency in
hz'); fs=input('Enter the stop band cutoff frequency in
hz'); f=input('Enter the sampling frequency');

%Normalising frequency
wp=2*fp/f
ws=2*fs/f

% Order of filter 3db cuttoff
frequency
[N,wc]=cheb1ord(wp,ws,rp,rs);
disp('order of the filter =');N
disp('cutoff frequency');wc
%coefficients of the filter
[b,a]=cheby1(N,rp,wc);


%Frequency Response
[H,om]=freqz(b,a);
%magnitude plot
plot(om/pi,abs(H));
title('Magnitude plot');
figure;
plot(om/pi,angle(H));
title('Phase Plot');

%magnitude plot in db figure;
plot(om/pi,20*log10(abs(H)));
title('Magnitude plot in db');
```

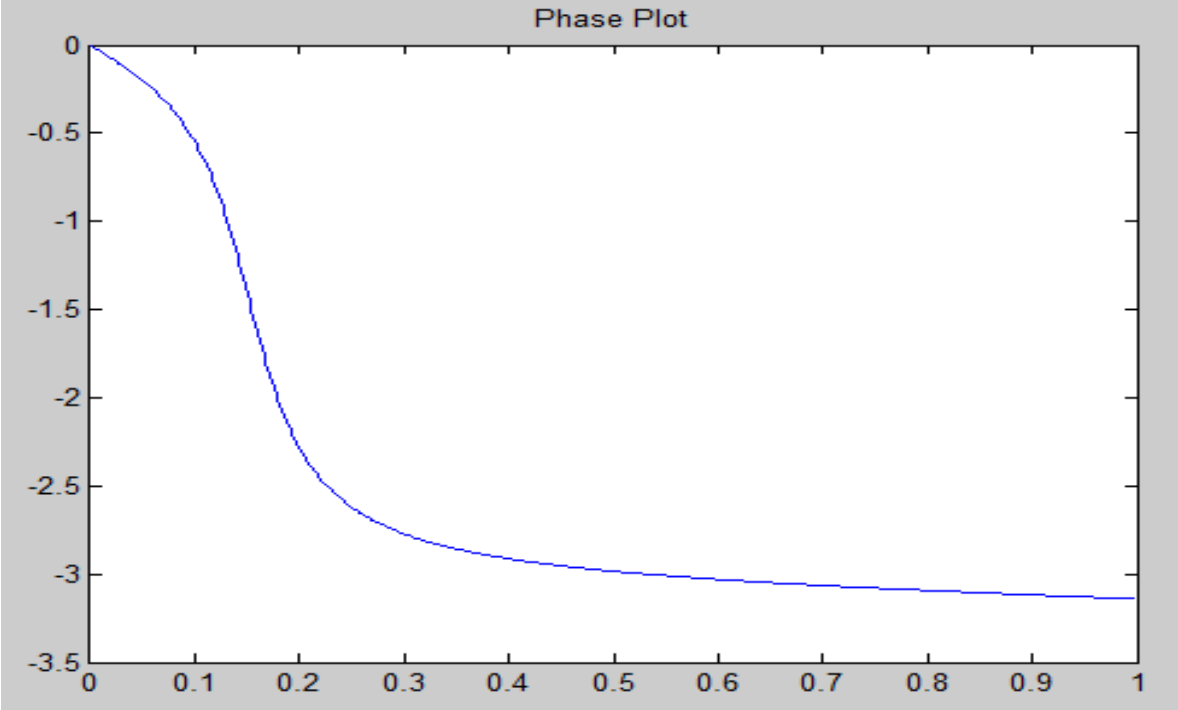
(INPUTS TO BE GIVEN IN COMMAND WINDOW)

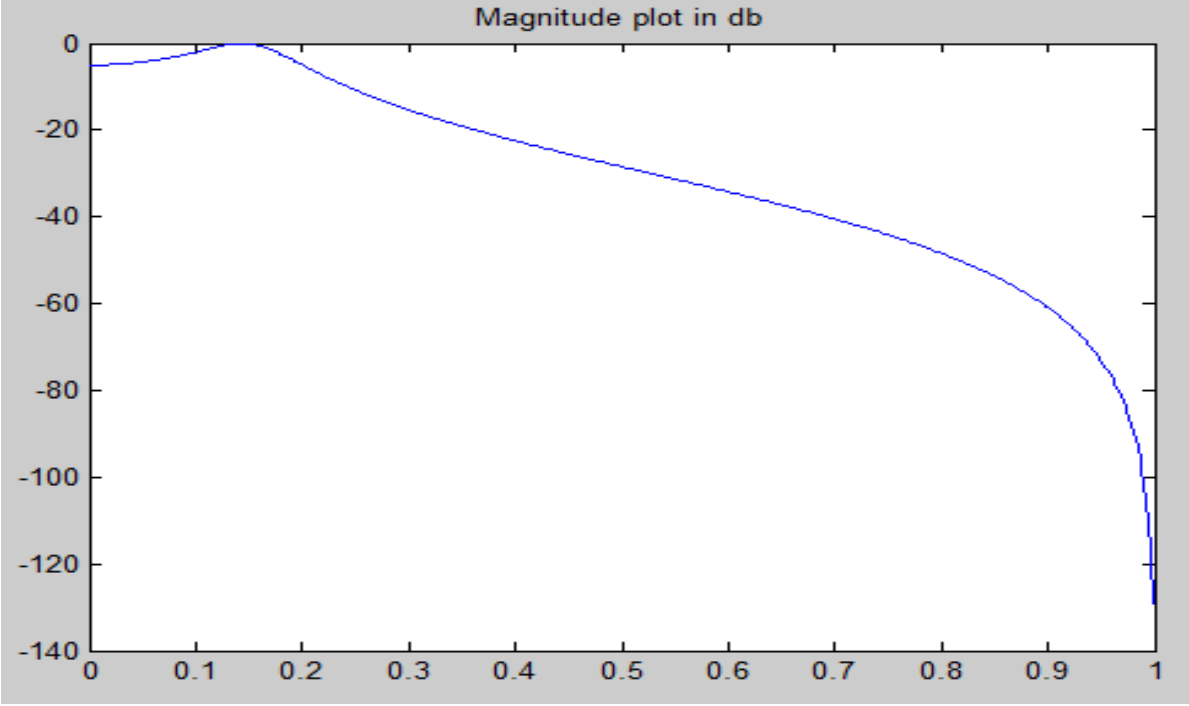
Enter the pass band in db 5
Enter the stop band attenuation in db 15
Enter the pass band cuttoff frequency in hz 1000
Enter the stop band cutoff frequency in hz 1500
Enter the sampling frequency 10000

OUTPUT:

wp =
0.2000
ws =
0.3000
order of the filter =
N =
2
cutoff frequency
wc =
0.2000



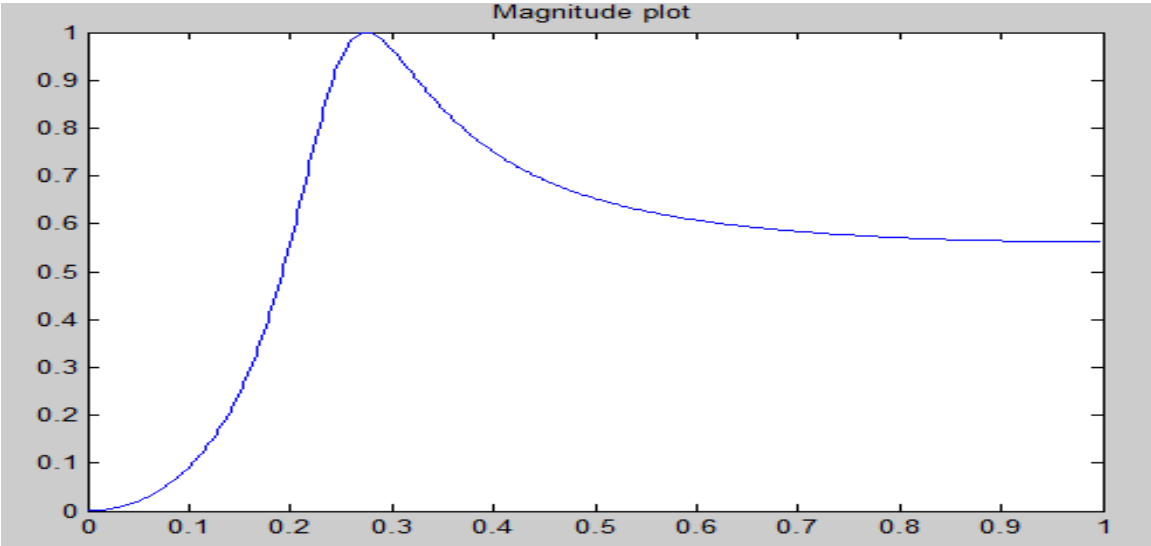

Phase Plot
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

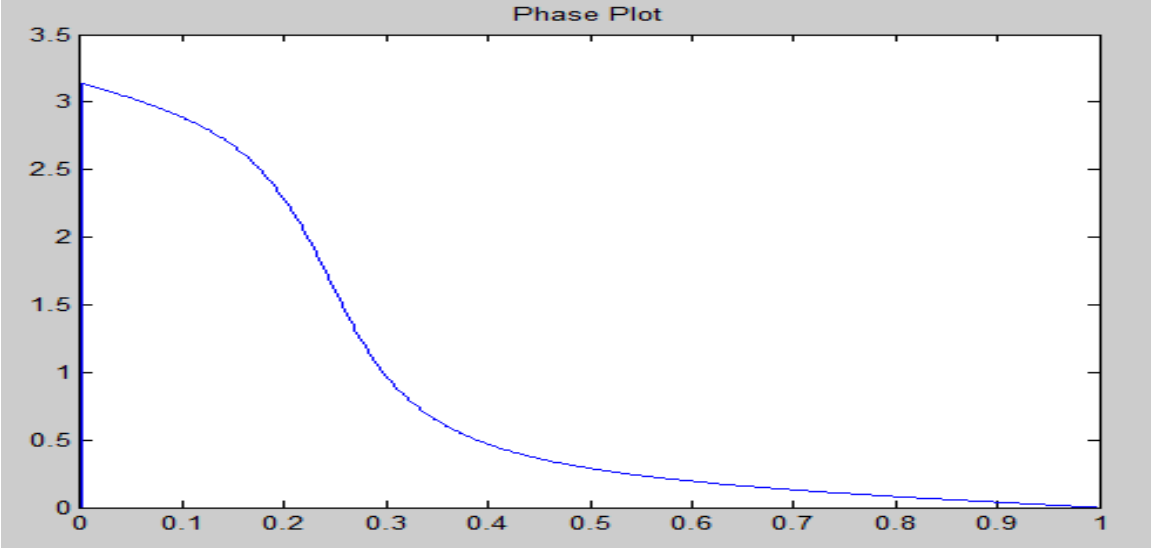

Magnitude plot in db
0
-20
-40
-60
-80
-100
-120
-140
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

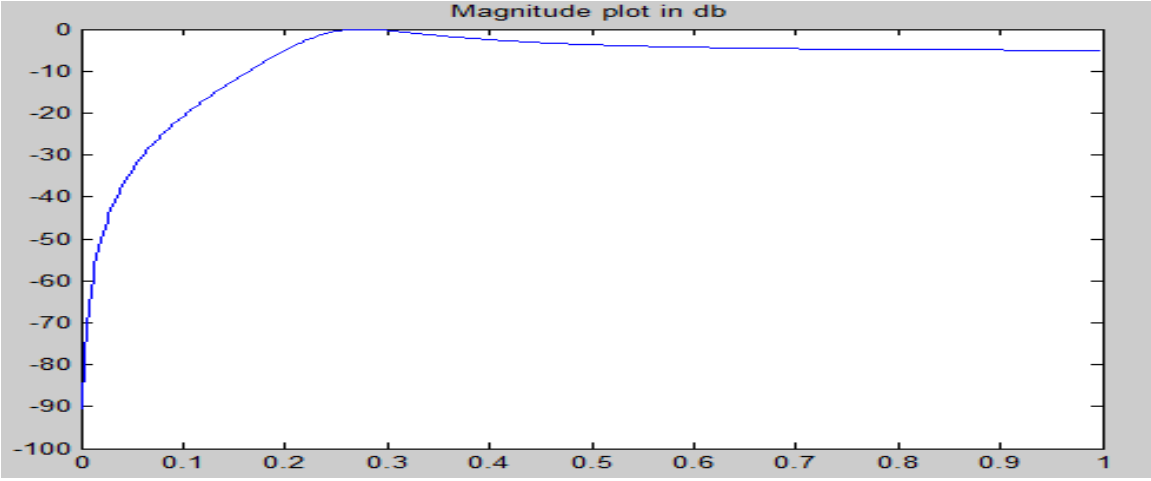
% CHEBYSHEV TYPE -1 HIGH PASS FILTER

```
clc;
clear all;
close all;
rp=input('Enter the pass band in db');
rs=input('Enter the stop band attenuation in
db');
fp=input('Enter the pass band cuttoff frequency in
hz'); fs=input('Enter the stop band cutoff frequency in
hz'); f=input('Enter the sampling frequency');

%Normalising frequency
wp=2*fp/f
ws=2*fs/f
% Order of filter & 3db cuttoff frequency
[N,wc]=cheb1ord(wp,ws,rp,rs);
disp('order of the filter =');N
disp('cutoff frequency');wc
%coefficients of the filter
[b,a]=cheby1(N,rp,wc,'high')
;

%Frequency Response
[H,om]=freqz(b,a);

%magnitude plot
plot(om/pi,abs(H));
title('Magnitude plot');
figure;
plot(om/pi,angle(H));
title('Phase Plot');

%magnitude plot in db figure;
plot(om/pi,20*log10(abs(H)));
title('Magnitude plot in db');
```

(INPUTS TO BE GIVEN IN COMMAND WINDOW)
Enter the pass band in db 5
Enter the stop band attenuation in db 15
Enter the pass band cuttoff frequency in hz 1500 Enter the stop band cutoff frequency in hz 1000 Enter the sampling frequency 10000
order of the filter = N =2
cutoff frequency wc = 0.2000


Magnitude plot
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1


Phase Plot
3.5
3
2.5
2
1.5
1
0.5
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1


Magnitude plot in db
0
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

% CHEBYSHEV TYPE – 2 LOW PASS FILTER

ALGORITHM:

- Get the passband and stopband ripples
- Get the passband and stopband edge frequencies
- Calculate the order of the filter using ‘ cheb2ord ’ function
- Find the filter coefficients, using ‘cheby2’ function
- Draw the magnitude and phase response

MATLAB PROGRAM:

```
clc;
clear all;
closeall;
rp=input('Enter the pass band in db');
rs=input('Enter the stop band attenuation in
db');
fp=input('Enter the pass band cuttoff frequency in
hz'); fs=input('Enter the stop band cutoff frequency in
hz'); f=input('Enter the sampling frequency');

%Normalising frequency
wp=2*fp/f
ws=2*fs/f

% Order of filter 3db cuttoff
frequency
[N,wc]=cheb2ord(wp,ws,rp,rs);
disp('order of the filter =');N
disp('cutoff frequency');wc

%coefficients of the filter
[b,a]=cheby2(N,rs,wc);
%Frequency Response
[H,om]=freqz(b,a);

%magnitude plot
plot(om/pi,abs(H));
title('Magnitude plot');
figure;
plot(om/pi,angle(H));
title('Phase Plot');

%magnitude plot in db figure;
plot(om/pi,20*log10(abs(H)));
title('Magnitude plot in db');
```

(INPUTS TO BE GIVEN IN COMMAND WINDOW)

Enter the pass band in db 5
Enter the stop band attenuation in db 15
Enter the pass band cuttoff frequency in hz 1000
Enter the stop band cutoff frequency in hz 1500
Enter the sampling frequency 10000

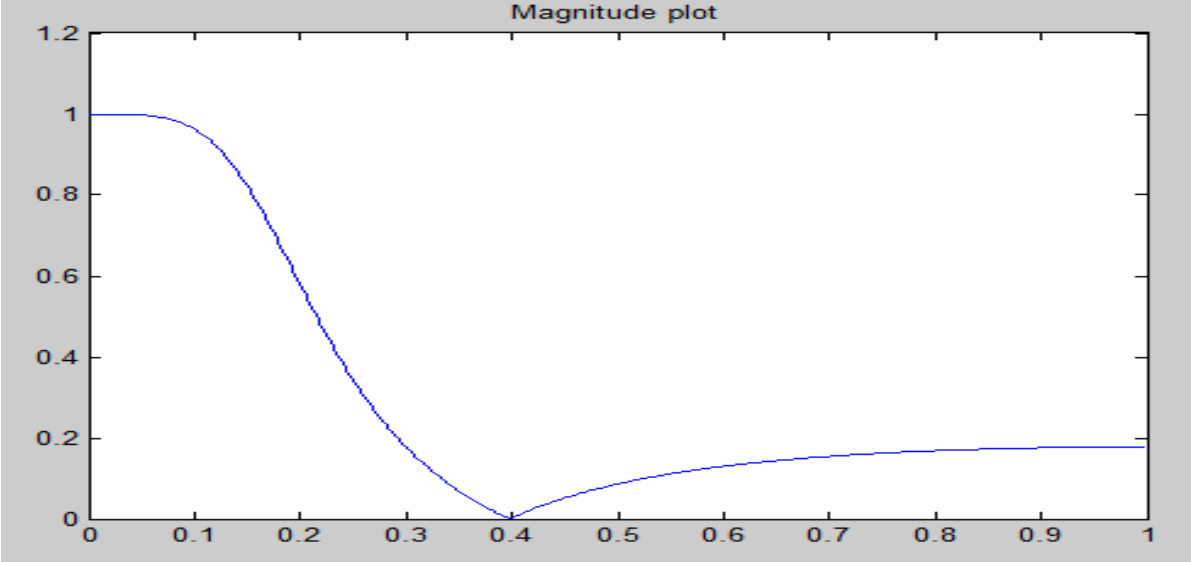
OUTPUT:

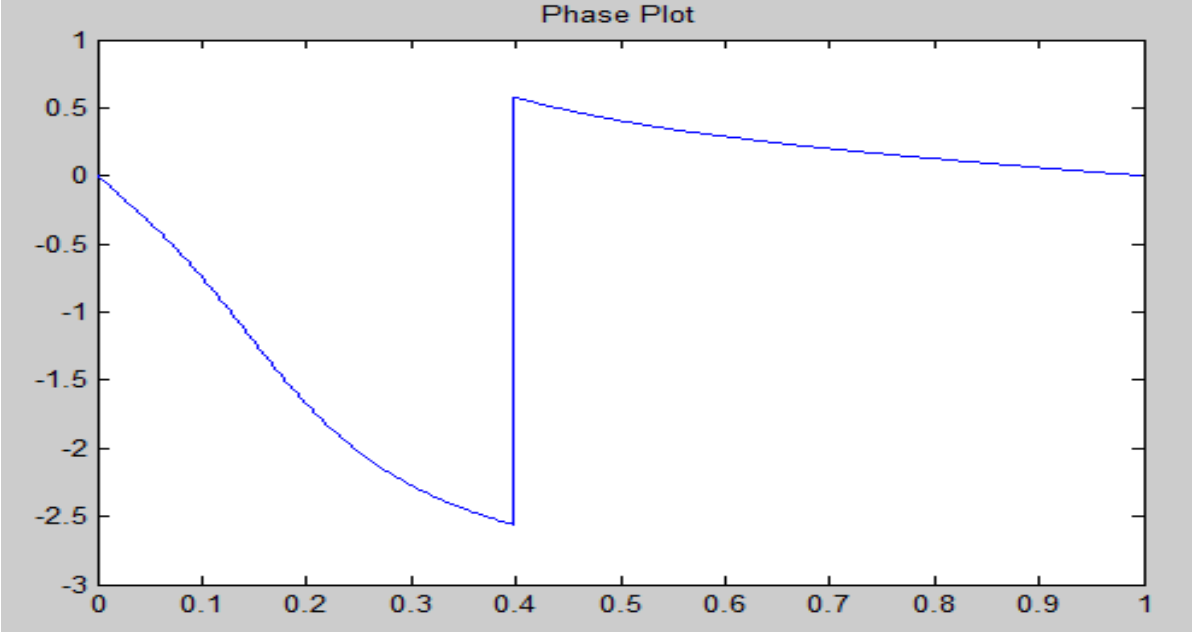
wp =
0.2000

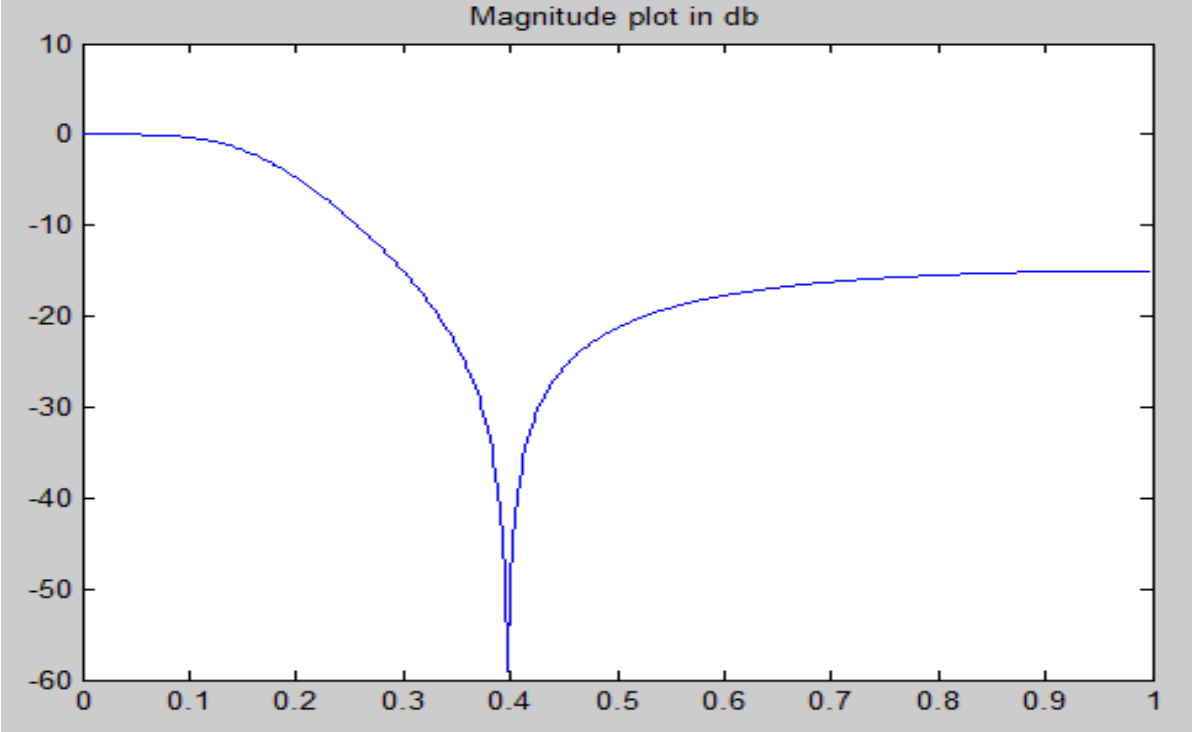
ws =
0.3000

order of the filter =
N =
2

cutoff frequency
wc =
0.3000




Phase Plot
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Magnitude plot in db
10
0
-10
-20
-30
-40
-50
-60
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

% CHEBYSHEVE TYPE -2 HIGH PASS FILTER

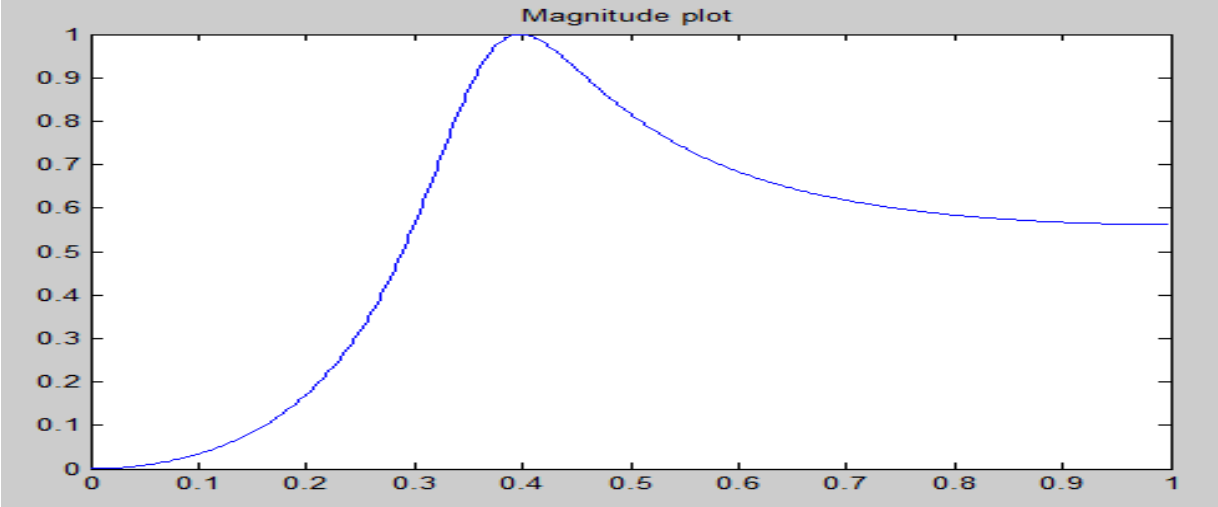
```
clc;
clear all;
close all;
rp=input('Enter the pass band in db');
rs=input('Enter the stop band attenuation in
db');
fp=input('Enter the pass band cuttoff frequency in
hz'); fs=input('Enter the stop band cutoff frequency in
hz'); f=input('Enter the sampling frequency');

%Normalising frequency
wp=2*fp/f
ws=2*fs/f

% Order of filter & 3db cuttoff frequency
[N,wc]=cheb1ord(wp,ws,rp,rs);
disp('order of the filter =');N
disp('cutoff frequency');wc

%coefficients of the filter
[b,a]=cheby1(N,rp,wc,'high')
; %Frequency Response
[H,om]=freqz(b,a);

%magnitude plot
plot(om/pi,abs(H));
title('Magnitude plot');
figure;
plot(om/pi,angle(H));
title('Phase Plot');

%magnitude plot in db figure;
plot(om/pi,20*log10(abs(H)));
title('Magnitude plot in db');
```

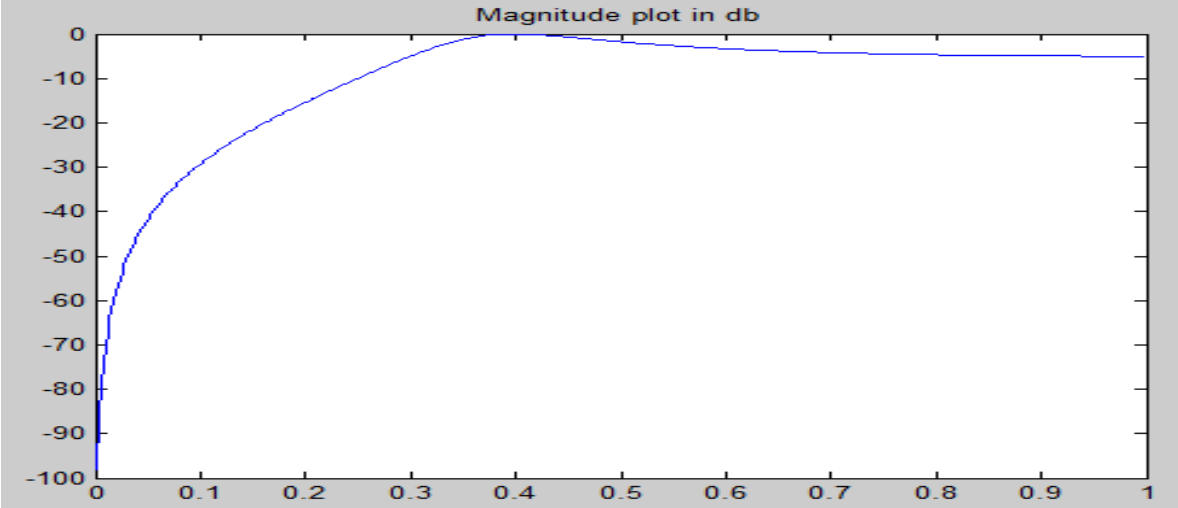
(INPUTS TO BE GIVEN IN COMMAND WINDOW)

Enter the pass band in db 5
Enter the stop band attenuation in db 15
Enter the pass band cuttoff frequency in hz 1500
Enter the stop band cutoff frequency in hz 1000
Enter the sampling frequency 10000

OUTPUT:

```
wp =
    0.3000

ws =
    0.2000

order of the filter =
N =
     2

cutoff frequency
wc =
     0.2000
```





EXPERIMENT- 06

INTERPOLATION AND DECIMATION

6.a)AIM: To perform down-sampling of given signal in Time and Frequency domain.

SOFTWARE: Matlab R2014a

THEORY:

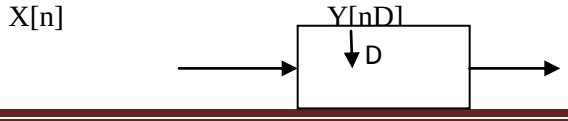
Sample-rate conversion is the process of changing the sampling rate of a discrete signal to obtain a new discrete representation of the underlying continuous signal. Multi rate processing are a clever digital signal processing (DSP) techniques that broadband and wireless design engineers can employ during the system design process.

Using these techniques, design engineers can gain an added degree of freedom that could improve the overall performance of a system architecture. Application areas include image scaling and audio/visual systems, where different sampling-rates may be used for engineering, economic, or historical reasons and multi-rate processing finds use in signal processing systems where various sub-systems with differing sample or clock rates need to be interfaced together. At other times multi-rate processing is used to reduce computational overhead of a system.

Types of Sample-Rate Conversion:

There are three types of sampling rate conversion. These include down conversion or decimation by a factor M; up conversion by a factor L; and sampling rate conversion by a ratio of M and L.To start the discussion, let's focus on down conversion

In signal processing, downsampling (or "subsampling") is the process of reducing the sampling rate of a signal. This is usually done to reduce the data rate or the size of the data. The down sampling factor (commonly denoted by *D*) is usually an integer or a rational fraction greater than unity. This factor multiplies the sampling time or, equivalently, divides the sampling rate.



ALGORITHM:

- Read the length of the signal and downsampling rate
- Generate a sine wave
- Generate the downsampled signal by y=x([1:M:length(x)])
- Plot input and output waveforms

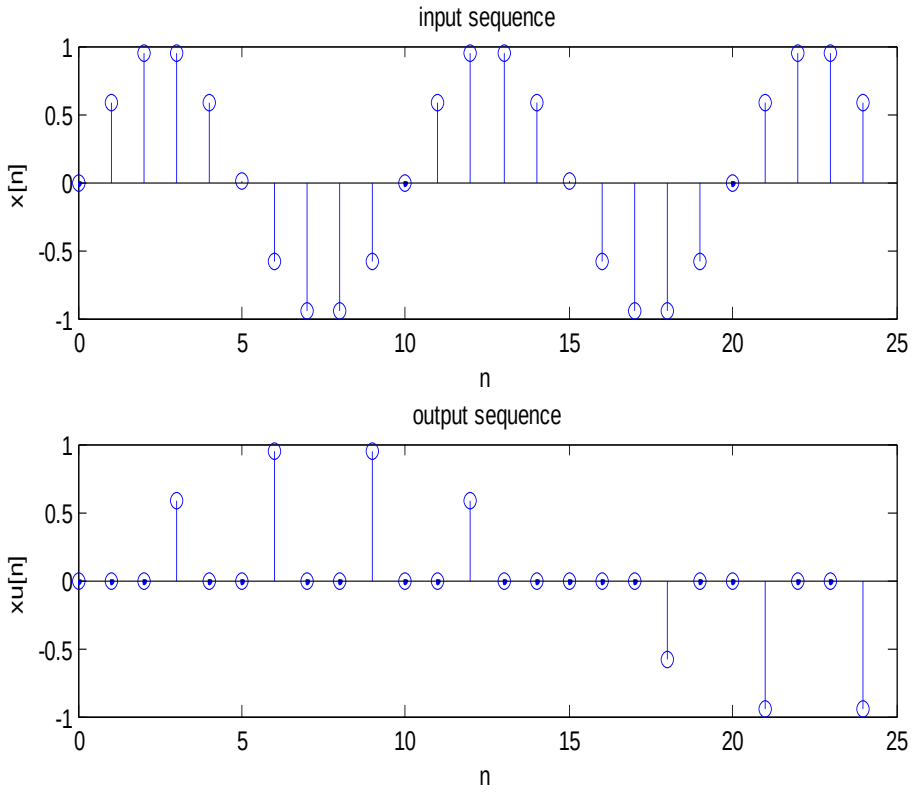
MATLAB PROGRAM:

```
clc;

N=input('length of output signal = ');

M=input('down sampling factor = ');

fs=input('sampling frequency = ');

n=0:N-1;

m=0:N*M-1;

x=sin(2*pi*fs*m);

y=x([1:M:length(x)]);

subplot(2,1,1);

stem(n,x(1:N));

title('input sequence');

subplot(2,1,2);

stem(n,y);

title('output sequence');
```

OUTPUT VALUES:

length of output signal = 25
down sampling factor = 3
sampling frequency = 0.05

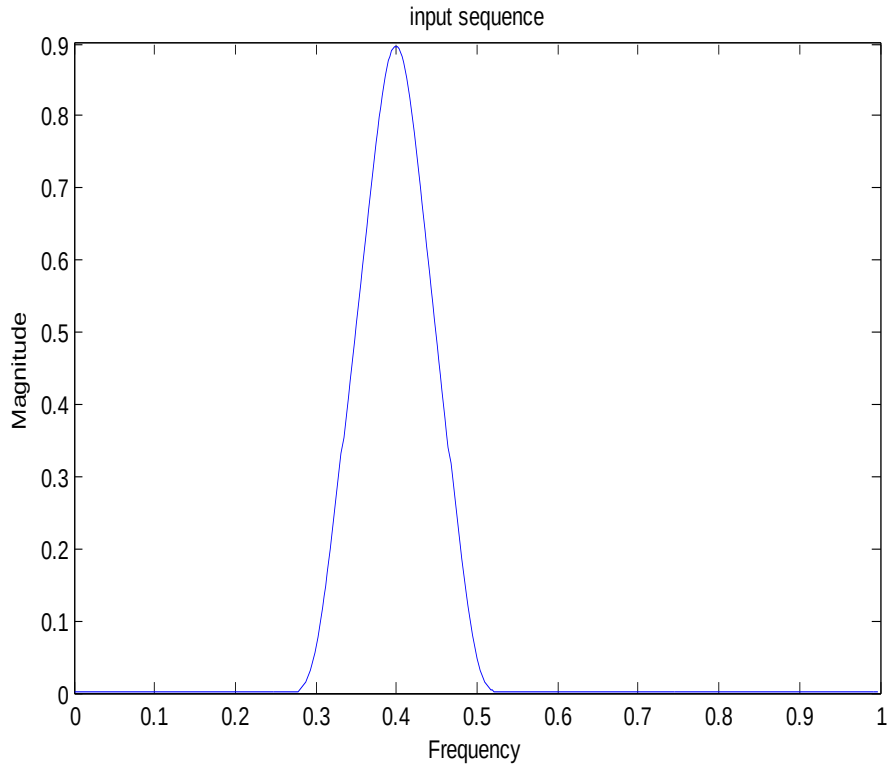
WAVEFORM

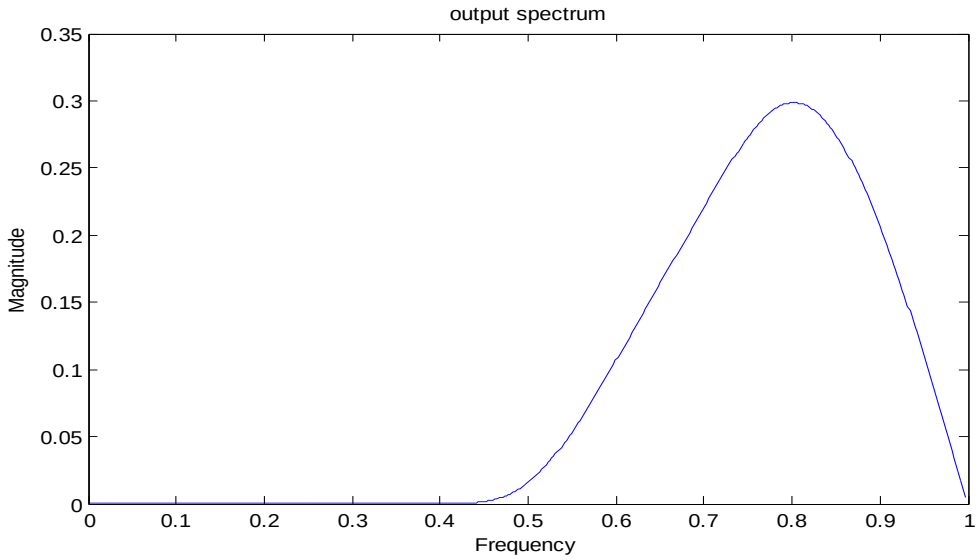


% MATLAB CODE FOR DOWN SAMPLING IN FREQUENCY-DOMAIN:

```
clc;

freq=[0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1];

mag=[0 0 0 0 1 0 0 0 0 0 0];

x=fir2(99,freq,mag);

[xz,w]=freqz(x,1,512);

plot(w/pi,abs(xz));

title('input sequence');
M=3;
y=x([1:M:length(x)]);
[yz,w]=freqz(y,1,512);
figure;
plot(w/pi,abs(yz));

title('output spectrum');
```

OUTPUT WAVEFORMS:





6.b)AIM: To perform Up-Sampling of a given signal.

SOFTWARE: Matlab R2014a

THEORY:

Up sampling is the process of increasing the sampling rate of a signal. The up sampling factor (commonly denoted by *I*) is usually an integer or a rational fraction greater than unity. This factor multiplies the sampling rate or, equivalently, divides the sampling period.

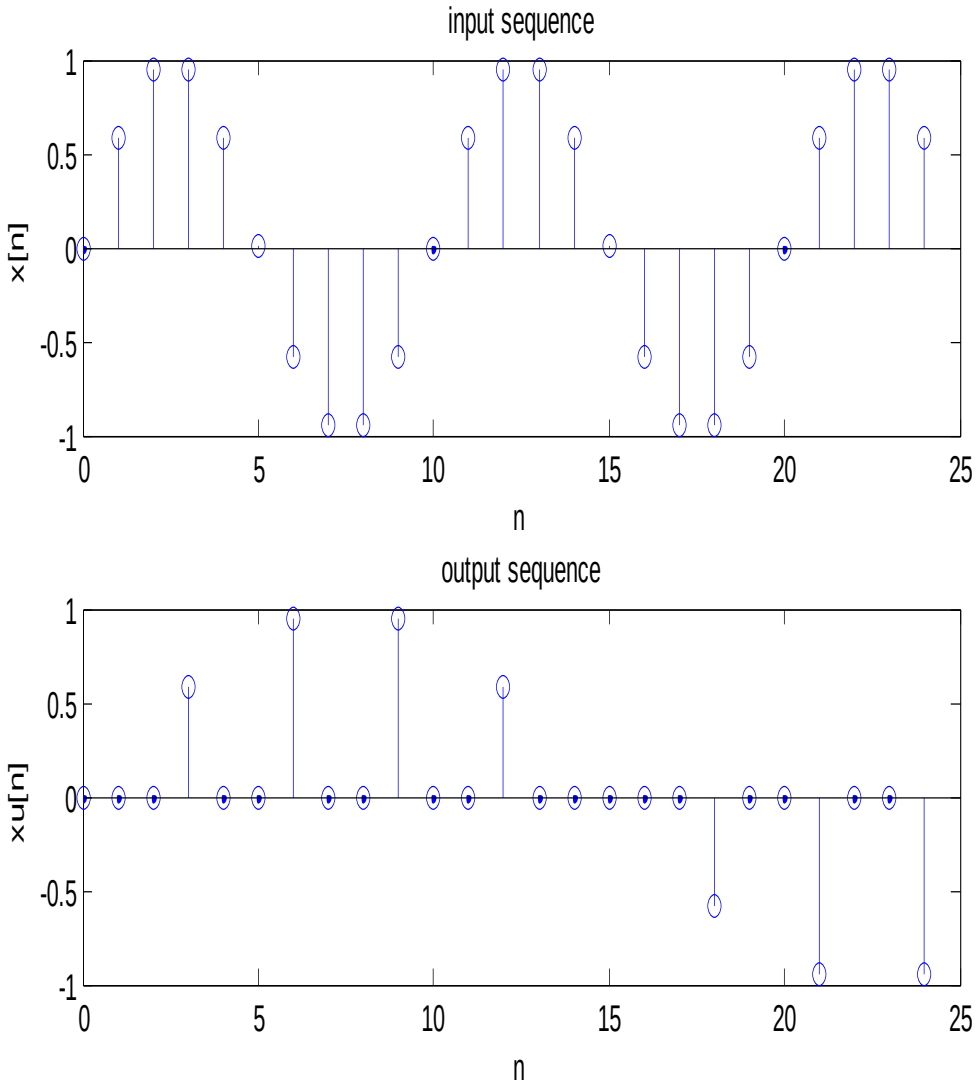
% MATLAB CODE FOR UP SAMPLING IN TIME-DOMAIN:

```
clc;
N=input('length of input signal = ');
L=input('up sampling factor = ');
fs=input('sampling frequency = ');
n=0:N-1;
x=sin(2*pi*fs*n);
y=zeros(1,L*length(x));
y([1:L:length(y)])=x
subplot(2,1,1);
stem(n,x);
title('input sequence');
subplot(2,1,2);
stem(n,y(1:length(x)));
title('output sequence');
```

OUTPUT :

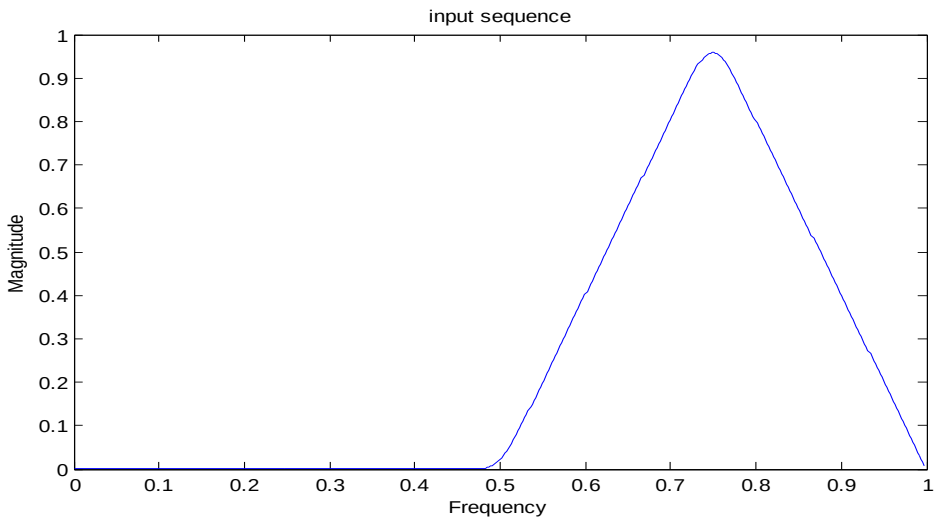
length of input signal = 25
up sampling factor = 3
sampling frequency = 0.1

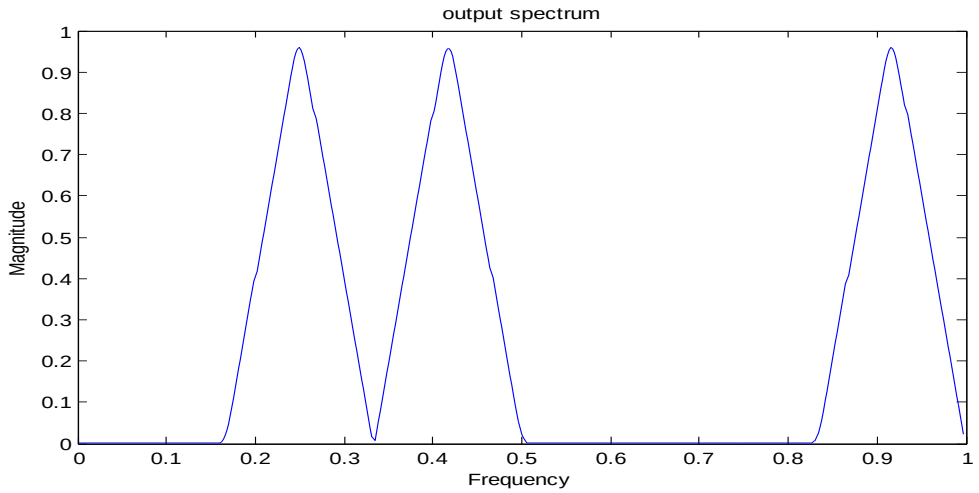
OUTPUT WAVEFORMS:



% MATLAB CODE FOR UP SAMPLING IN FREQUENCY-DOMAIN:

```
clc;
freq=[0 0.25 0.5 0.75 1];
mag=[0 0 0 1 0];
x=fir2(99,freq,mag);
[xz,w]=freqz(x,1,512);
plot(w/pi,abs(xz));
title('input sequence');
L=input('up sampling factor = ');
y=zeros(1,L*length(x));
y([1:L:length(y)])=x;
[yz,w]=freqz(y,1,512);
plot(w/pi,abs(yz));
title('output spectrum');
```

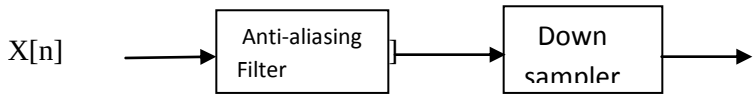



output spectrum
Magnitude
Frequency
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

6.c)AIM: To perform Decimation of a given input signal by an integer D.

DESCRIPTION:

The process of reducing the sampling rate by a factor D is called Decimation. In this process, the input signal is first passed through a low pass filter. The filter eliminates the spectrum of the input signal in the range of pi/D < w < pi. The output of filter is then down sampled by a factor of D to produce the desired decimation.



% MATLAB CODE FOR DECIMATION OF INPUT SIGNAL:

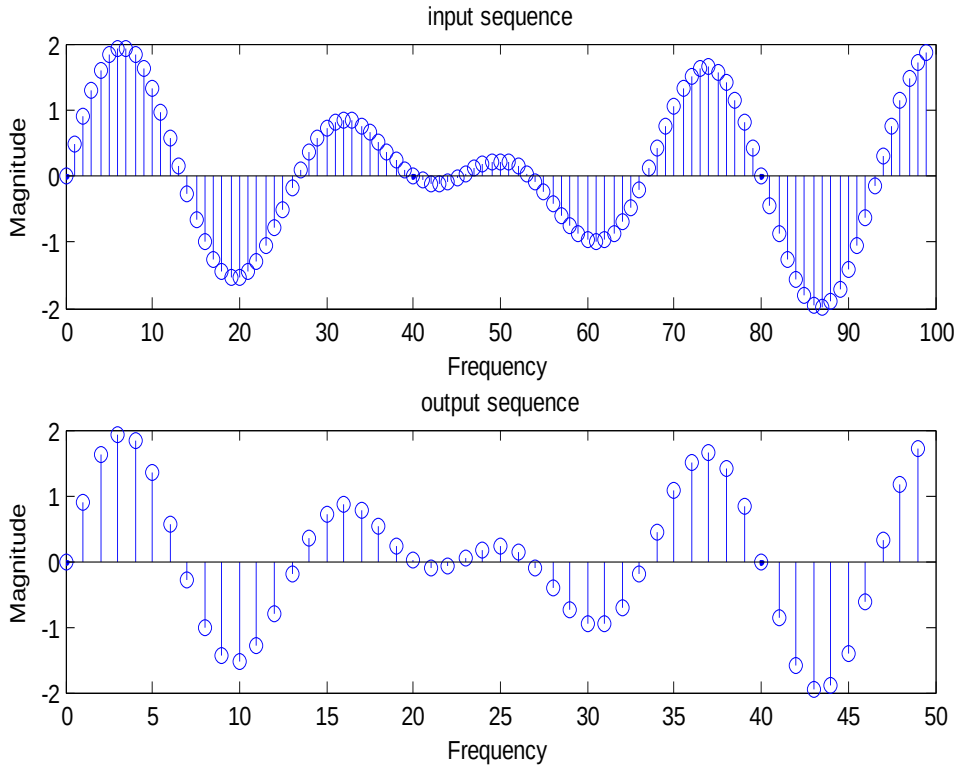
```
clc;
N=input('length of input signal = ');
D=input('down sampling factor = ');
F1=input('input sampling frequency1 = ');
F2=input('input sampling frequency2 = ');
n=0:N-1;
x=sin(2*pi*F1*n)+sin(2*pi*F2*n);
y=decimate(x,D,'fir');
subplot(2,1,1);
stem(n,x(1:N));
title('input sequence');
xlabel('Frequency');
ylabel('Magnitude');
subplot(2,1,2);
```

```
m=0:N/D-1;

stem(m,y(1:N/D));

title('output sequence');

xlabel('Frequency');

ylabel('Magnitude');
```

OUTPUT

length of input signal = 100
down sampling factor = 2
input sampling frequency1 = 0.043
input sampling frequency2 = 0.032

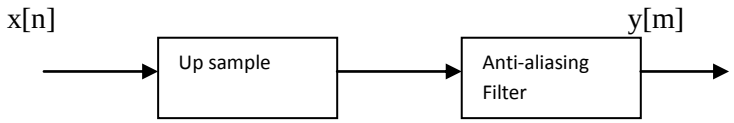
OUTPUT WAVEFORMS:



6.d)AIM: To perform interpolation of given signal by an integer factor.

DESCRIPTION:

The process of increasing the sampling rate by an integer factor I is called Interpolation. An increase in the sampling rate can be accomplished by interpolating I-1 samples between successive values of the signal.



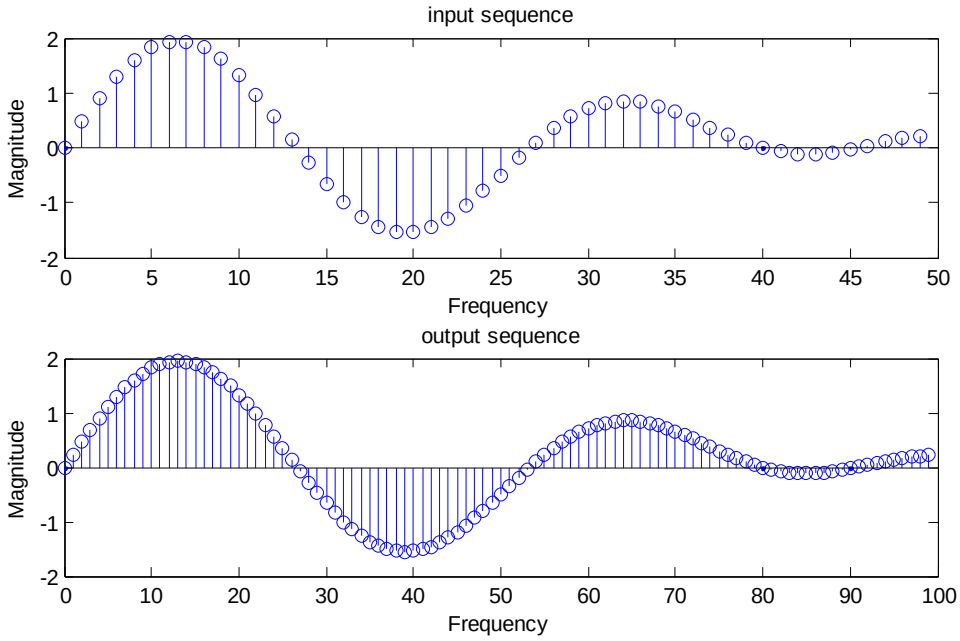
% MATLAB CODE FOR DECIMATION OF INPUT SIGNAL:

```
clc;
N=input('length of input signal = ');
l=input('up sampling factor = ');
F1=input('input sampling frequency1 = ');
F2=input('input sampling frequency2 = ');
n=0:N-1;
x=sin(2*pi*F1*n)+sin(2*pi*F2*n);
y=interp(x,l);
subplot(2,1,1);
stem(n,x(1:N));
title('input sequence');
xlabel('Frequency');
ylabel('Magnitude');
subplot(2,1,2);
m=0:N*l-1;
```

```
stem(m,y(1:N*l));

title('output sequence');

xlabel('Frequency');

ylabel('Magnitude');
```

OUTPUT :

length of input signal = 50
up sampling factor = 2
input sampling frequency1 = 0.043
input sampling frequency2 = 0.032
OUTPUT WAVEFORMS



EXPERIMENT- 07

IMPLEMENTATION OF MULTI RATE SYSTEMS

AIM: To implement a multi rate system using Matlab Simulink

SOFTWARE: Matlab R2014a

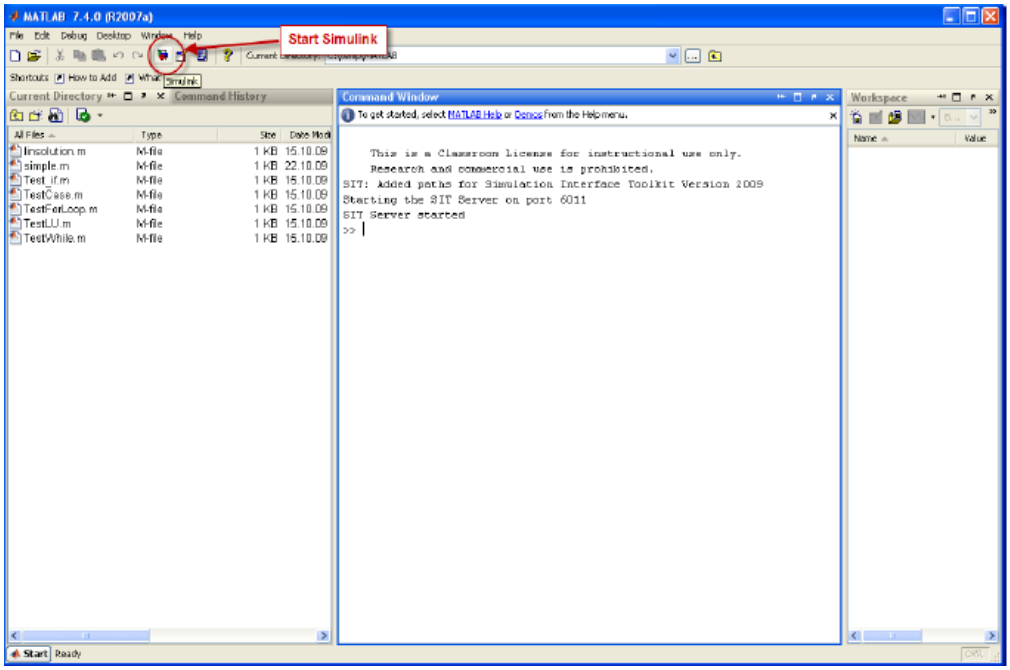
THEORY:

Simulink, developed by The MathWorks, is a commercial tool for modeling, simulating and analyzing dynamic systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. It offers tight integration with the rest of the MATLAB environment and can either drive MATLAB or be scripted from it. Simulink is widely used in control theory and digital signal processing for simulation and design.

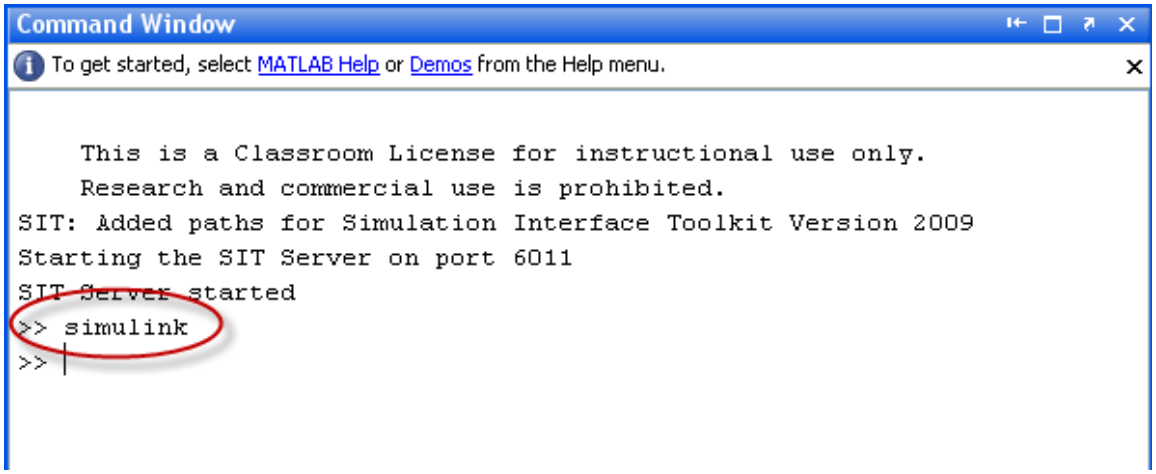
MATLAB command line using just the signal processing toolbox, can be enhanced significantly by adding Simulink and the DSP blockset. Simulink is a blockdiagram based simulation environment that sits on top of MATLAB. The DSP blockset augments Simulink with a DSP specific block library and requires that the signal processing toolbox be present.

Starting Simulink:

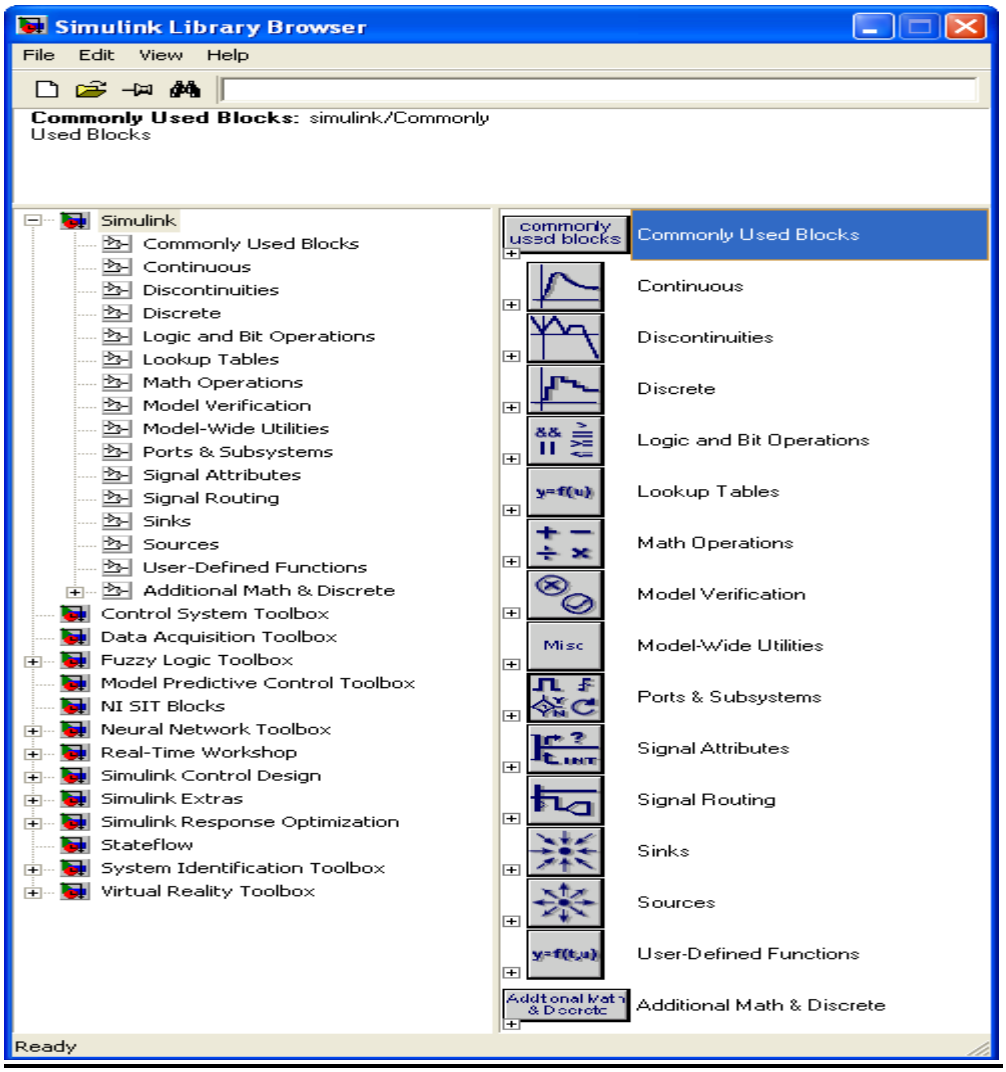
You start Simulink from the MATLAB IDE:
Open MATLAB and select the Simulink icon in the Toolbar:



Or type "**simulink**" in the Command window, like this:

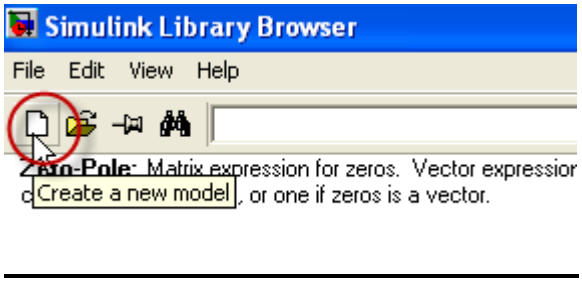


Then the following window appears (**Simulink Library Browser**)

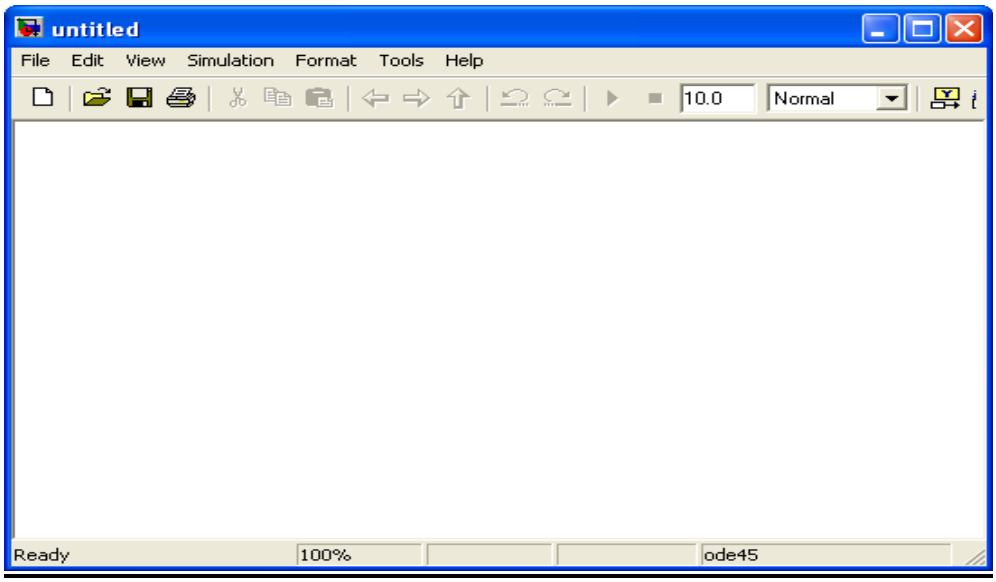


Crearting Model:

Click the New icon on the Toolbar in order to create a new Simulink model:



The following window appears:

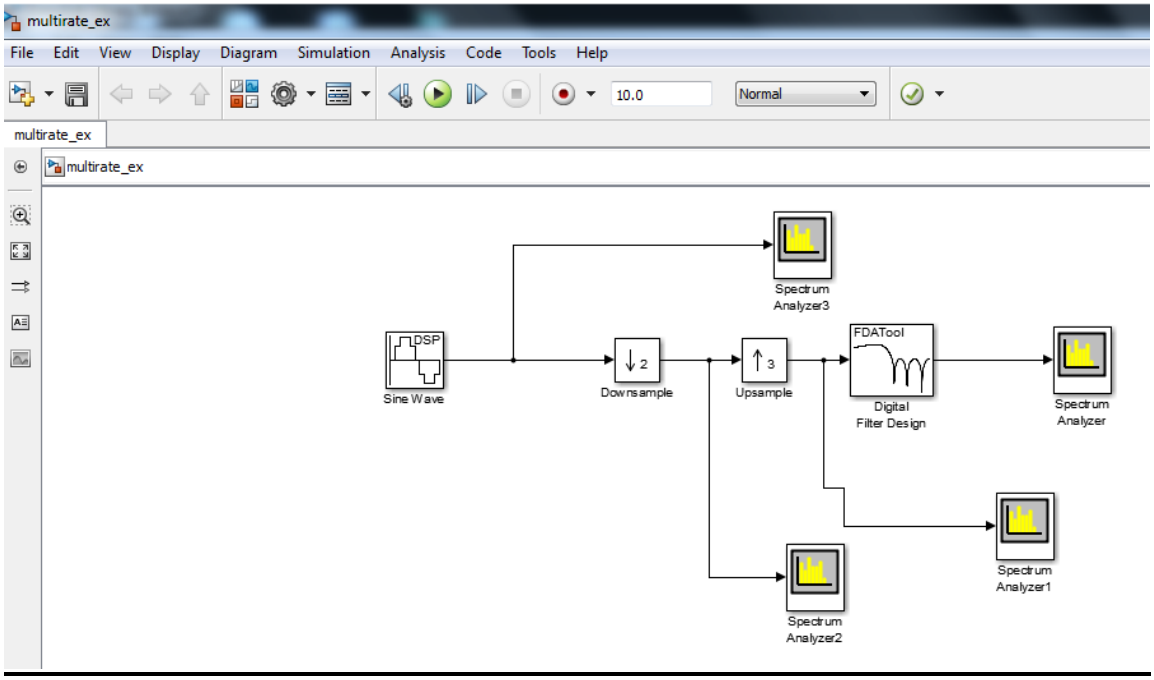


Multi rate system:

Sample-rate conversion is the process of changing the sampling rate of a discrete signal to obtain a new discrete representation of the underlying continuous signal. Multi rate processing are a clever digital signal processing (DSP) techniques that broadband and wireless design engineers can employ during the system design process.

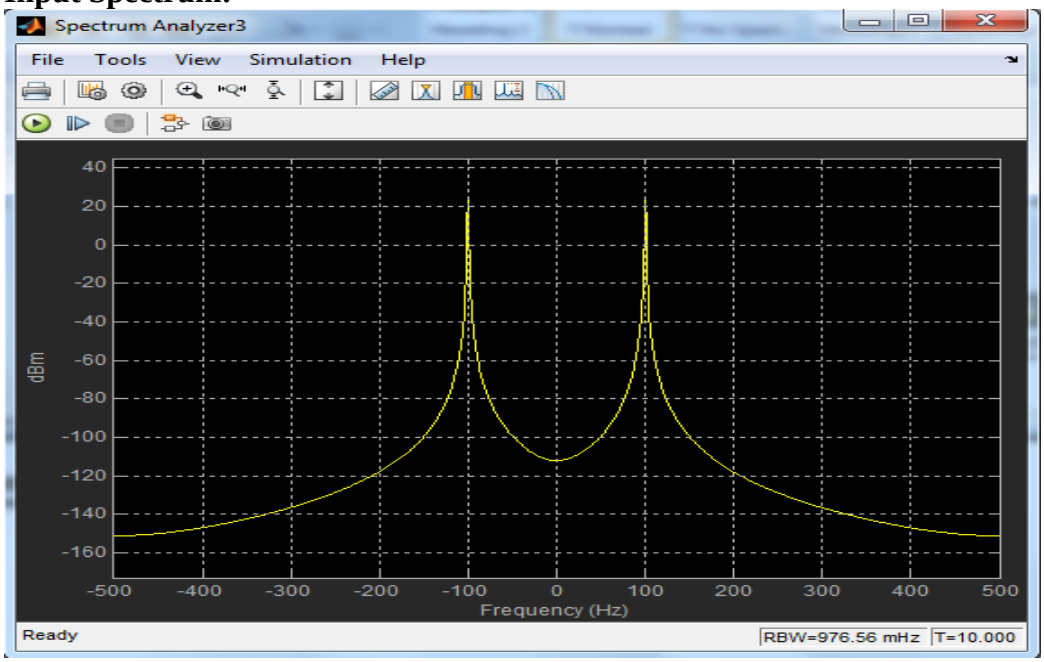
Using these techniques, design engineers can gain an added degree of freedom that could improve the overall performance of a system architecture. Application areas include image scaling and audio/visual systems, where different sampling-rates may be used for engineering, economic, or historical reasons and multi-rate processing finds use in signal processing systems where various sub-systems with differing sample or clock

rates need to be interfaced together. At other times multi-rate processing is used to reduce computational overhead of a system.

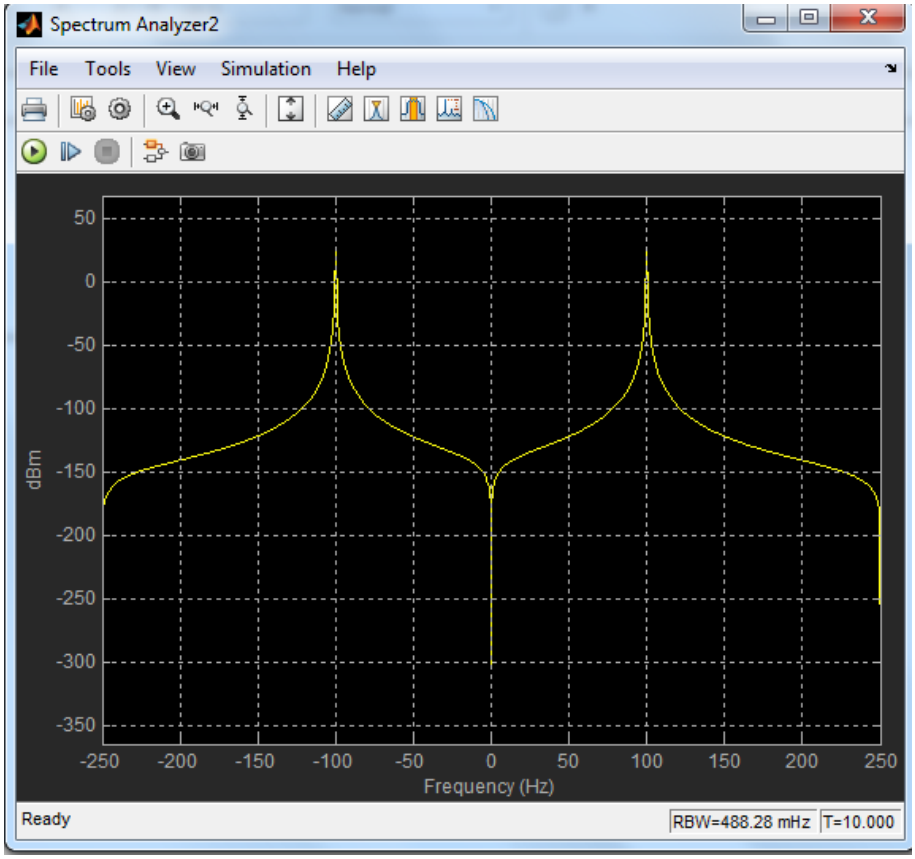


Results:

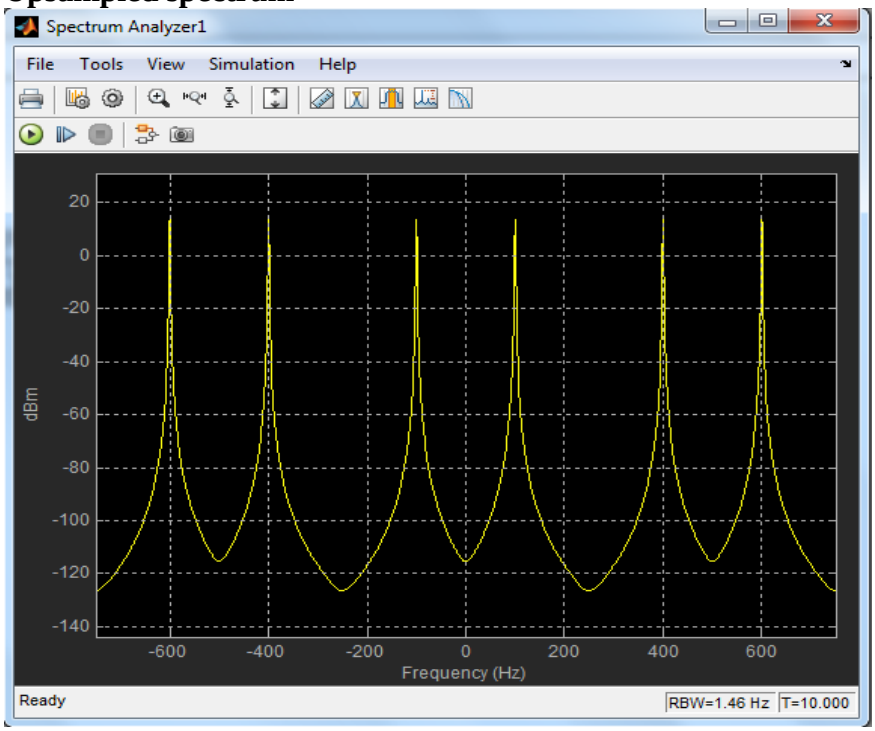
Input Spectrum:



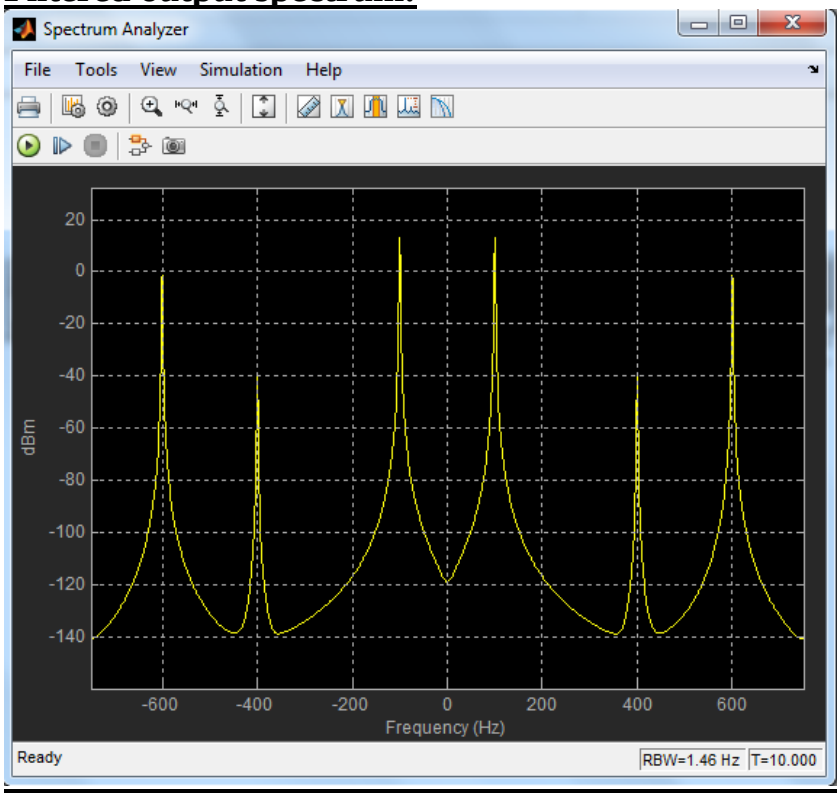
Downsampled Spectrum



Upsampled spectrum



Filtered output spectrum:



EXPERIMENT- 08

TIME RESPONSE OF NON-LINEAR SYSTEMS

AIM: To obtain the time response of a given second order system with its damping frequency.

SOFTWARE: Matlab R2014a

THEORY:

The time response has utmost importance for the design and analysis of control systems because these are inherently time domain systems where time is independent variable. During the analysis of response, the variation of output with respect to time can be studied and it is known as time response. To obtain satisfactory performance of the system with respect to time must be within the specified limits. From time response analysis and corresponding results, the stability of system, accuracy of system and complete evaluation can be studied easily.

Due to the application of an excitation to a system, the response of the system is known as time response and it is a function of time. The two parts of response of any system:

(i) Transient response

(ii) Steady-state response.

Transient response: The part of the time response which goes to zero after large interval of time is known as transient response.

Steady state response: The part of response that means even after the transients have died out is said to be steady state response.

The total response of a system is sum of transient response and steady state response: $C(t)=C_{tr}(t)+C_{ss}(t)$

TIME RESPONSE OF SECOND ORDER CONTROL SYSTEM: A second order control system is one wherein the highest power of 's' in the denominator of its transfer function equals 2.

Transfer function is given by: TF=

$$\frac{\omega_n^2}{S^2 + 2\xi\omega_n S + \omega_n^2}$$

$\omega_n$is called natural frequency of oscillations.

ω_d is called damping frequency oscillations.

ξ called damping ratio.

MATLAB PROGRAM:

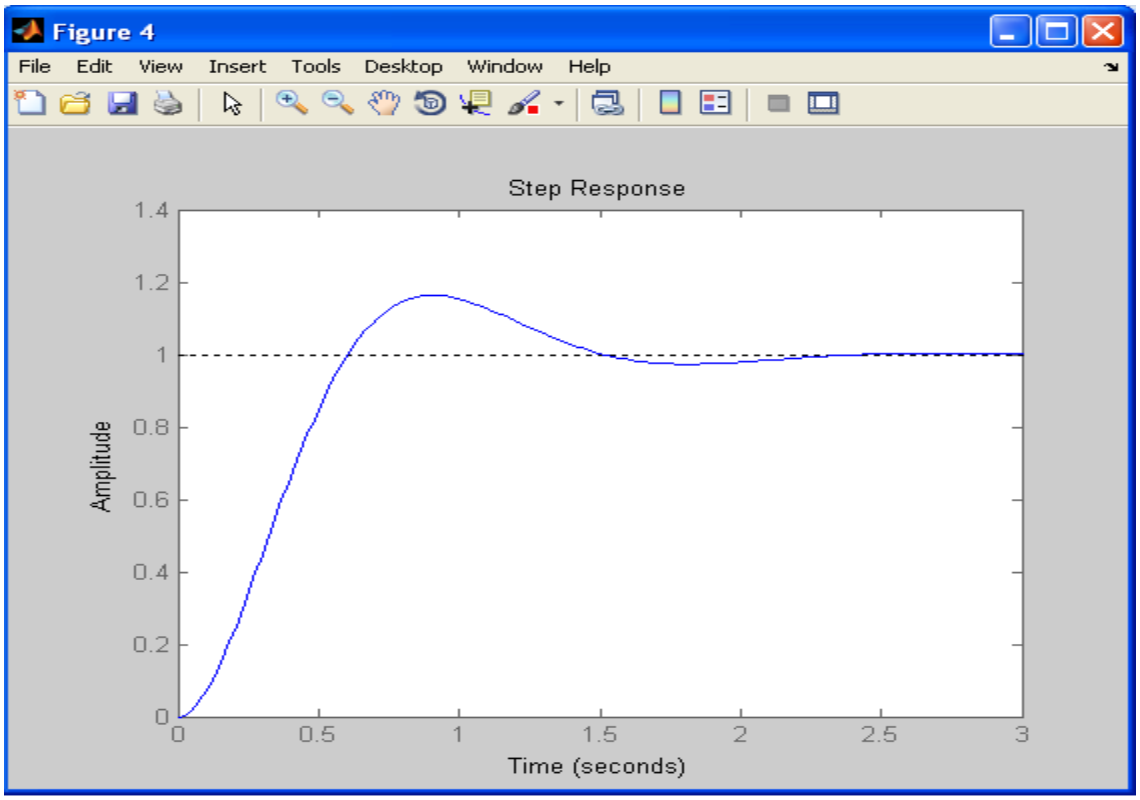
```
wn=input('enter value of undamped natural frequency')
 z=input('enter value of damping ratio')
n=[wn*wn];
 p=sqrt(1-z^2);
 wd=wn*p ;
h=[p/z] ;
k=atan(h) ;
m=pi-k;
tr=[m/wd]
 tp=[pi/wd]
q=z*wn ;
ts=[3/q] % 5 percent tolerance
r=z*pi ;
f=[r/p] ;
mp=exp(-f)
num=[0 0 n]
den=[1 2*z*wn n]
s=tf(num,den)
 hold on
step(s)
```

Results:

wn=4

z=0.5

```
tr =
    0.6046
tp =
    0.9069
ts =
    1.5000
mp =
    0.1630
s =
        16
  --------------
  s^2 + 4 s + 16
```



.EXPERIMENT- 09

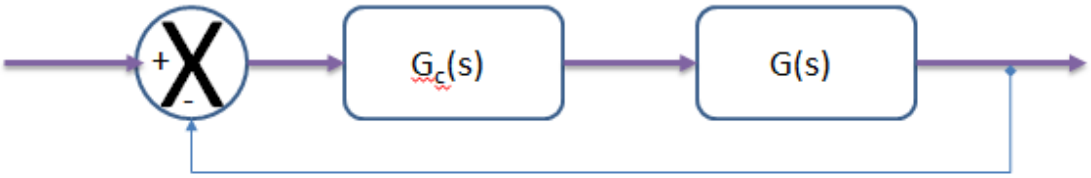
DESIGN OF PI AND PID CONTROLLERS

9.a)AIM: To design a PI controller for the given phase margin.

SOFTWARE: Matlab R2014a

THEORY:

A controller with transfer function $G_c(s)$ can be introduced in cascade with the open loop transfer function G(s) to modify the transient and steady state response of the system.



The different types of controllers employed in control systems are the following

1. Proportional Controller(P-controller)
2. Proportional-plus-integral controller(PI-controller)
3. Proportional-plus-derivative controller(PD-controller)
4. Proportional-plus-derivative-plus-integral controller(PID-controller)

The proportional controller is a device that produces an output signal u(t) which is proportional to the input signal

In P-controller, $u(t) \alpha e(t)$

$u(t) = K_p e(t)$

$G_c(s) = K_p$

The PI-controller is a device that produces an output signal, u(t) consisting of two terms-one term proportional to input signal e(t) and other proportional to integral of the input signal e(t)

$u(t) \alpha [e(t) + \int e(t)dt]$

$u(t) = K_p e(t) + K_i \int e(t)dt$

$$G_c(s) = K_p + \frac{Ki}{s}$$

The PD-controller is a device that produces an output signal, u(t) consisting of two terms-one term proportional to input signal e(t) and other proportional to derivative of the input signal e(t)

$$u(t) \ \alpha \ [\ e(t) + \frac{d}{dt}\ e(t)]$$

$$u(t) = K_p\, e(t) + K_d \frac{d}{dt}\ e(t)$$

$$G_c(s) = K_p + K_d\, s$$

The PID-controller is a device that produces an output signal, u(t) consisting of three terms- proportional to input signal e(t) ,proportional to integral of the input signal e(t) and proportional to derivative of the input signal e(t)

$$u(t) \ \alpha \ [\ e(t) + \int e(t)dt\ \frac{d}{dt}\ e(t)]$$

$$u(t) = K_p\, e(t) + K_i \int e(t)dt + K_d \frac{d}{dt}\ e(t)$$

$$G_c(s) = K_p + \frac{Ki}{s} + K_d\, s$$

Algorithm:

1. Read the given uncompensated transfer function G(s) and desired gain margin G_m
2. Plot the Bode Plot for G(s)
3. Find the magnitude A_1 and Phase $Ø_1$of G(jw) at the given gain cross over frequency w_1
4. Calculate $\theta = G_m - (180^0 + Ø_1)$
5. Calculate $K_p = \frac{cos\theta}{A1}$ and $K_i = \frac{-\omega 1 sin\theta}{A1}$
6. Find the transfer function $G(s) = K_p + \frac{Ki}{s}$

Matlab Program:

```
clc;
pmr=60;
w1=0.5;
s=tf('s');
h=100/((s+1)*(s+2)*(s+5));
```

```
w=logspace(-5,5);
bode(h,'k')
mag=100/((sqrt(w1^2+1))*(sqrt(w1^2+4))*(sqrt(w1^2+25)))
ang=(-atan(w1)-atan(w1/2)-atan(w1/5))*(180/pi)
theta=180+pmr-ang;
theta=theta*(pi/180)
kp=cos(theta)/mag
ki=(-w1*sin(theta))/mag
numc=[kp ki];
denc=[1 0];
gc=tf(numc,denc)
g=h*gc
figure
bode(g,'k')
[gm pm pcf gcf]=margin(g)
```

Results:

```
Mag=8.63
Ang=-46.3
kp =
    0.0325
ki =
    0.0556
gc =
  0.03253 s + 0.05558
  -------------------
          S
g =
       3.253 s + 5.558
  --------------------------
  s^4 + 8 s^3 + 17 s^2 + 10 s
gm =
   11.9824
pm =
   60.0007
pcf =
    2.4743
gcf =
    0.5000
```

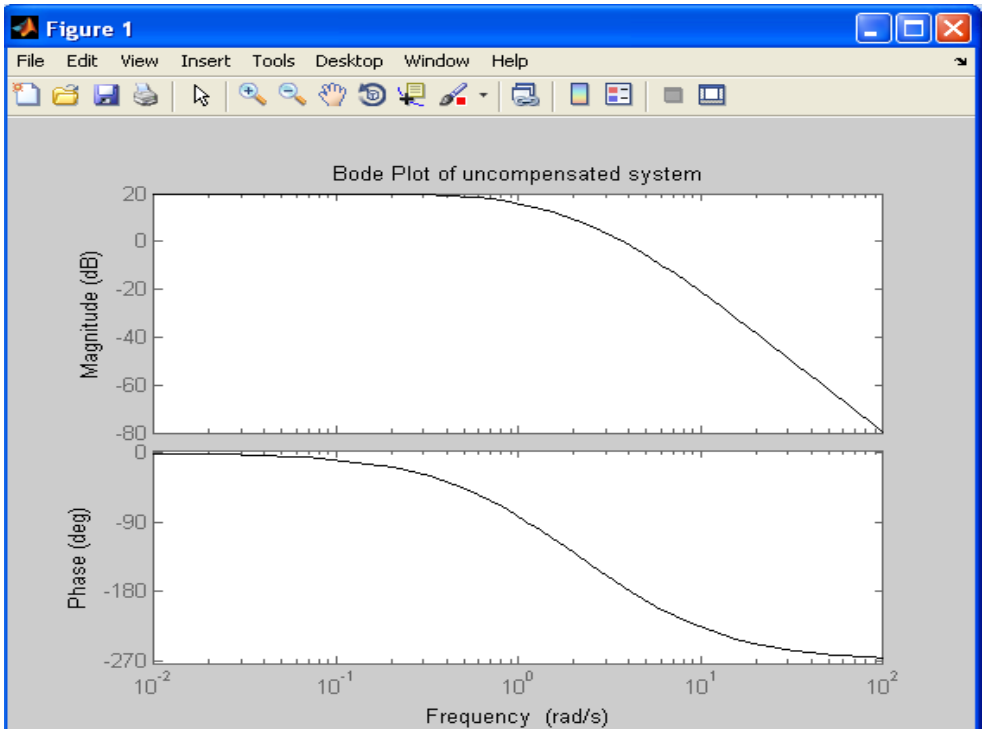


Figure 1
File Edit View Insert Tools Desktop Window Help
Bode Plot of uncompensated system
Magnitude (dB)
20
0
-20
-40
-60
-80
Phase (deg)
0
-90
-180
-270
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
Frequency (rad/s)

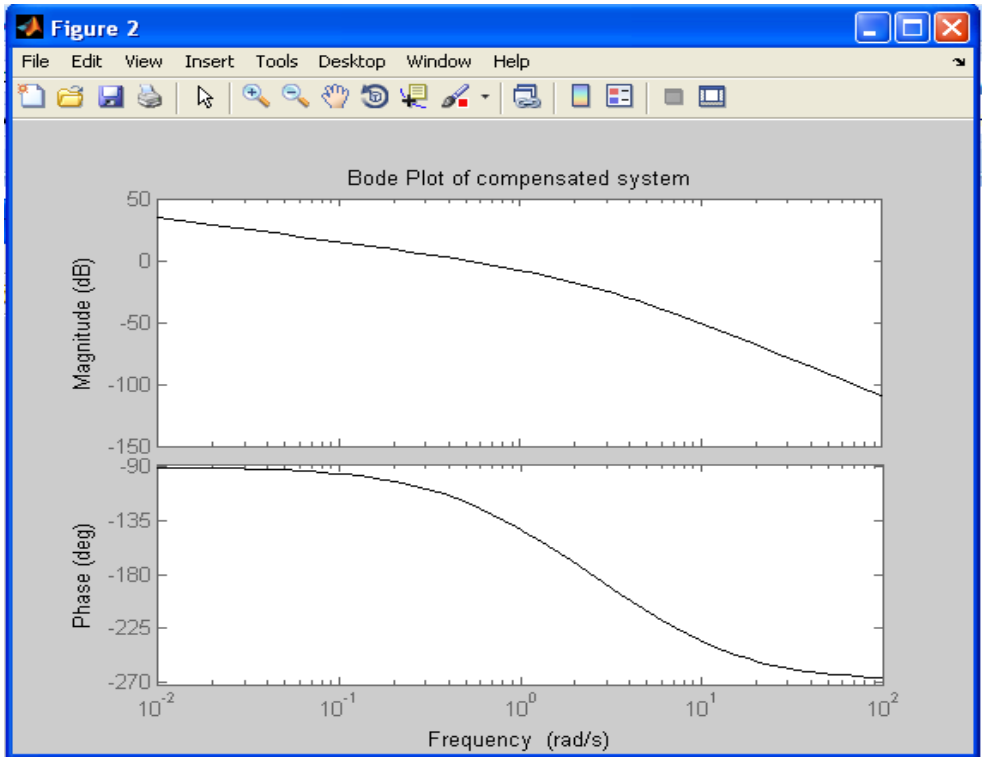


Figure 2
File Edit View Insert Tools Desktop Window Help
Bode Plot of compensated system
Magnitude (dB)
50
0
-50
-100
-150
Phase (deg)
-90
-135
-180
-225
-270
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
Frequency (rad/s)

9.b)AIM: To design a PID controller for the given phase margin.

SOFTWARE: Matlab R2014a

Algorithm:

1. Read the given uncompensated transfer function G(s) and desired gain margin G_m
2. Plot the Bode Plot for G(s)
3. Find the magnitude A_1 and Phase $Ø_1$of G(jw) at the given gain cross over frequency w_1
4. Calculate $\theta = G_m - (180^0 + Ø_1)$
5. Calculate $K_p = \frac{cos\theta}{A1}$, $K_i = \frac{-\omega 1 sin\theta}{A1}$ and $K_d = \frac{sin\theta}{\omega 1 A1}$
6. Find the transfer function $G(s) = K_p + \frac{Ki}{s} + K_d s$

Matlab Program:

```
clc;
pmr=45;
w1=4;
kv=0.6;
s=tf('s');
h=100/((s+1)*(s+2)*(s+10));
w=logspace(-5,5);
bode(h,'k')
mag=100/((sqrt(w1^2+1))*(sqrt(w1^2+4))*(sqrt(w1^2+100)))
ang=(-atan(w1)-atan(w1/2)-atan(w1/10))*(180/pi)
theta=180+pmr-ang;
theta=theta*(pi/180)
kp=cos(theta)/mag
ki=2;
kd=(sin(theta)/(w1*mag))+(ki/w1^2)
numc=[kd kp ki];
denc=[0 1 0];
gc=tf(numc,denc)
g=h*gc
figure
bode(g,'k')
[gm pm pcf gcf]=margin(g)
```

Results:

Mag=0.5035

Ang= -161.2

Kp=1.7

Kd=0.3442

Ki=2

gc =

0.3442 s^2 + 1.782 s + 2

---

s

g =

34.42 s^2 + 178.2 s + 200

---

s^4 + 13 s^3 + 32 s^2 + 20 s

gm = Inf

pm = 45.0000

pcf = Inf

gcf = 4.0000

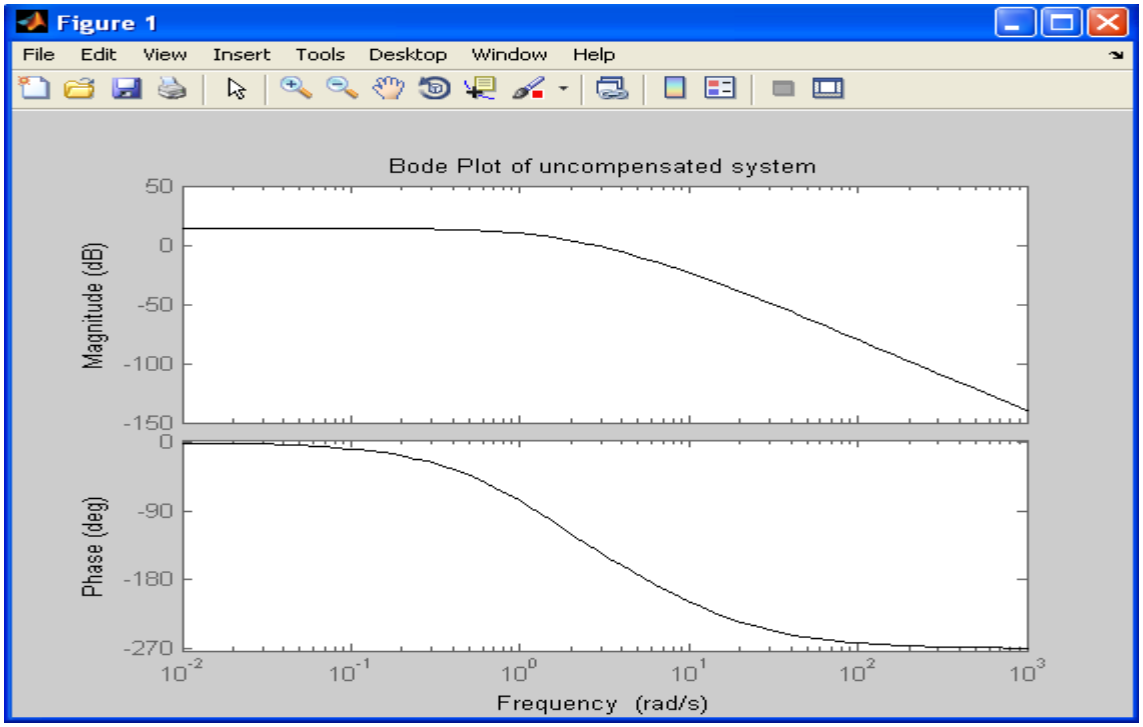


Figure 1
File Edit View Insert Tools Desktop Window Help
Bode Plot of uncompensated system
Magnitude (dB)
50
0
-50
-100
-150
Phase (deg)
0
-90
-180
-270
10^-2 10^-1 10^0 10^1 10^2 10^3
Frequency (rad/s)

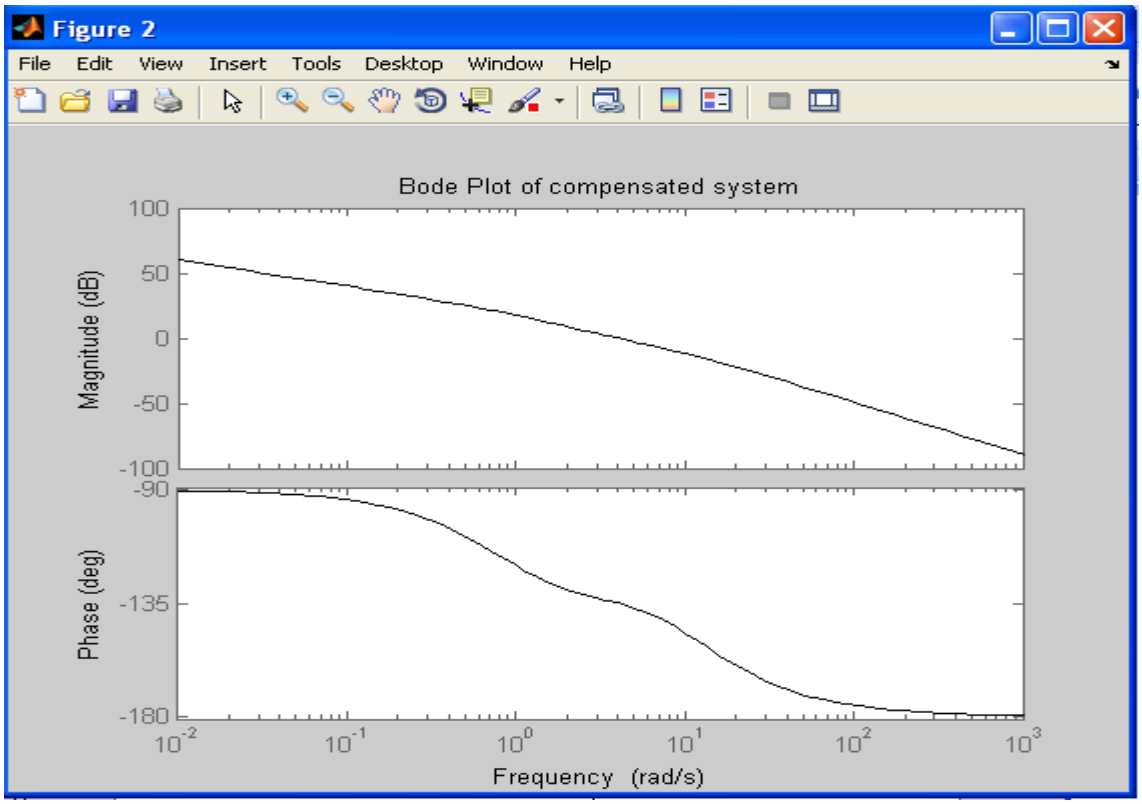


Figure 2
File Edit View Insert Tools Desktop Window Help
Bode Plot of compensated system
Magnitude (dB)
100
50
0
-50
-100
Phase (deg)
-90
-135
-180
10^-2 10^-1 10^0 10^1 10^2 10^3
Frequency (rad/s)

PART-B

EXPERIMENT # 1

SOLUTION OF DIFFERENCE EQUATION

AIM:Write a C Program to implement Difference Equation, compile, build using CCSand then execute the solution of difference equation using DSK .Plot Time-Frequency graph for the same and observe the output.

SOFTWARE / HARDWARE REQUIREMENTS:

1. Software: CCS(IDE) Code Composer Studio (Integrated Development Environment)
2. C compiler
3. Hardware: DSK Kit

DIFFERENCE EQUATION:

An Nth order linear constant – coefficient difference equation can be represented as

$$\sum_{k=0}^{N} a_k\, y(n-k) = \sum_{r=0}^{M} b_r\, x(n-r)$$

If we assume that the system is causal a linear difference equation provides an explicit relationship between the input and output. This can be seen by rewriting above equation.

$$y(n) = \sum_{r=0}^{M} b_r/a_0\, x(n-r) \quad -- \quad \sum_{k=1}^{N} a_k/a_{0\,y}\, y(n-k)$$

'C' PROGRAM TO IMPLEMENT DIFFERENCE EQUATION:

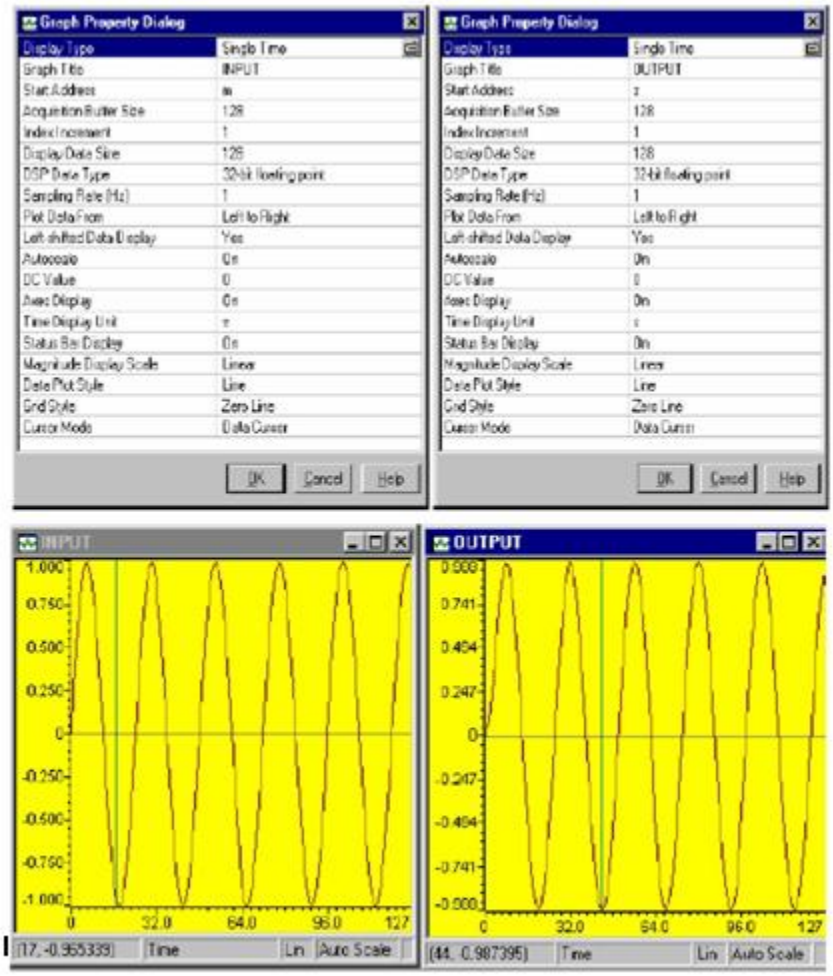
```
#include
<stdio.h>#include
<math.h>

#define FREQ 400

float y[3]={0,0,0};
float x[3]={0,0,0};
float z[128],m[128],n[128],p[128];

main()

{
int i=0,j;
```

```
float a[3]={ 0.072231, 0.144462, 0.072231};
float b[3]={ 1.000000, -1.109229, 0.398152};

for(i=0;i<128;i++)
{
m[i]=sin(2*3.14*FREQ*i/24000);
}

for(j=0;j<128;j++)
{
x[0]=m[j];
y[0] = (a[0] *x[0]) +(a[1]* x[1] ) +(x[2]*a[2]) - (y[1]*b[1])-(y[2]*b[2]);
z[j ]=y[0];
y[2]=y[1];
y[1]=y[0];
x[2]=x[1];
x[1]=x[0];
}
```

PROCEDURE:

- Open Code Composer Studio, make sure the DSP kit is turned on.
- Start a new project using .Project-new . pull down menu, save it in a separate directory(c : \ti\myprojects) with name **lconv.pjt.**
- Add the source files **DIFF EQ1.c** to the project using 'Project **add files to project'** pull down menu.
- Add the linker command file **hello.cmd** . (Path: c:\ti\tutorial\dsk6713\hello1\hello.cmd)
- Add the run time support library file **rts6700.lib** (Path: c:\ti\c6000\cgtools\lib\rts6700.lib)
- Compile the program using the .Project-compile. pull down menu or by clicking the shortcut icon on the left side of program window.
- Build the program using the .Project-Build. pull down menu or by clicking the shortcut icon on the left side of program window.
- Load the program(lconv.out) in program memory of DSP chip using the File-load program. pull down menu.
- To View output graphically

- Select view □ graph □ time and frequency.



Note: To verify the Diffence Equation, Observe the output for high frequency

RESULT:A C-program to implement difference equation is written and output isexecuted and verified using DSK successfully.

EXPERIMENT #2

IMPULSE RESPONSE

AIM: Write a C – Program to implement impulse response, compile, and build usingCCS and execute the output using DSK.

SOFTWARE / HARDWARE REQUIREMENTS:

1 Software: CCS(IDE) Code Composer Studio (Integrated Development Environment)
2 C compiler
3 Hardware: DSK Kit

'C' PROGRAM TO IMPLEMENT IMPULSE RESPONSE:

```
#include <stdio.h>

#define Order 2
#define Len 10

float y[Len]={0,0,0},sum;

main()
{
int j,k;

float a[Order+1]={0.1311, 0.2622, 0.1311};
float b[Order+1]={1, -0.7478, 0.2722};

for(j=0;j<Len;j++)
  {
        sum=0;
        for(k=1;k<=Order;k++)
        {
                if((j-k)>=0)
                sum=sum+(b[k]*y[j-
                k]);
        }
        if(j<=Order)
        {
                y[j]=a[j]-sum;
        }
        else
        {
                y[j]=-sum;
        }
        printf("Respose[%d] = %f\n",j,y[j]);

  }}
```

RESULT:A C-program to implement impulse response is written and output is executed andverified using DSK successfully.

OUTPUT

Response[1]=0.360237

Response[2]=0.364799

Response[3]=0.174741

Response[4]=0.031373

Response[5]= -0.024104

Response[6]= -0.026565

Response[7]= -0.013304

Response[8]= -0.002718

Response[9]=0.001589

EXPERIMENT # 3

LINEAR CONVOLUTION

AIM: Write a C-Program implement Linear Convolution, compile and built it using CCSand execute the output using DSK. Plot Time-Frequency graph for the same and verify the output.

SOFTWARE / HARDWARE REQUIREMENTS:

1 Software: CCS(IDE) Code Composer Studio (Integrated Development Environment)
2 C compiler
3 Hardware: DSK Kit

TO VERIFY LINEAR CONVOLUTION:

Linear Convolution involves the following operations.

1. Folding
2. Multiplication
3. Addition
4. Shifting

These operations can be represented by a Mathematical Expression as follows:

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

x[ ]= Input signal Samples
h[ ]= Impulse response co-efficient.
y[ ]= Convolution output.
n = No. of Input samples
h = No. of Impulse response co-efficient.

Algorithm to implement 'C' program for Convolution:

Eg: x[n] = {1, 2, 3, 4}
h[k] = {1, 2, 3, 4}

Where: n=4, k=4. Values of n & k should be a multiple of 4.
If n & k are not multiples of 4, pad with zero's to make multiples of 4

r = n+k-1 Size of output sequence.
= 4+4-1
= 7

Output: y[r] = { 1, 4, 10, 20, 25, 24, 16}.

NOTE: At the end of input sequences pad 'n' and 'k' no. of zero's.

'C' PROGRAM TO IMPLEMENT LINEAR CONVOLUTION

```
/* prg to implement linear convolution */

#include<stdio.h>

#define LENGHT1 6 /*Lenght of i/p samples sequence*/
#define LENGHT2 4 /*Lenght of impulse response Co-efficients */

int x[2*LENGHT1-1]={1,2,3,4,5,6,0,0,0,0,0}; /*Input Signal Samples*/
int h[2*LENGHT1-1]={1,2,3,4,0,0,0,0,0,0,0}; /*Impulse Response Coefficients*/

int y[LENGHT1+LENGHT2-1];

main()
{
        int i=0,j;

        for(i=0;i<(LENGHT1+LENGHT2-1);i++)
        {
                y[i]=0;
                for(j=0;j<=i;j++)

                        y[i]+=x[j]*h[i-j];

        }
        for(i=0;i<(LENGHT1+LENGHT2-1);i++)
        printf("%d\n",y[i]);
}
```

PROCEDURE:

- Open Code Composer Studio, make sure the DSP kit is turned on.
- Start a new project using .Project-new . pull down menu, save it in a separate directory(c:\ti\myprojects) with name **lconv.pjt.**
- Add the source files **conv.c**
  - to the project using .Project add files to 'project' pull down menu.
- Add the linker command file **hello.cmd.**

- (Path: c:\ti\tutorial\dsk6713\hello1\hello.cmd)

- Add the run time support library file **rts6700.lib**
  - (Path: c:\ti\c6000\cgtools\lib\rts6700.lib)

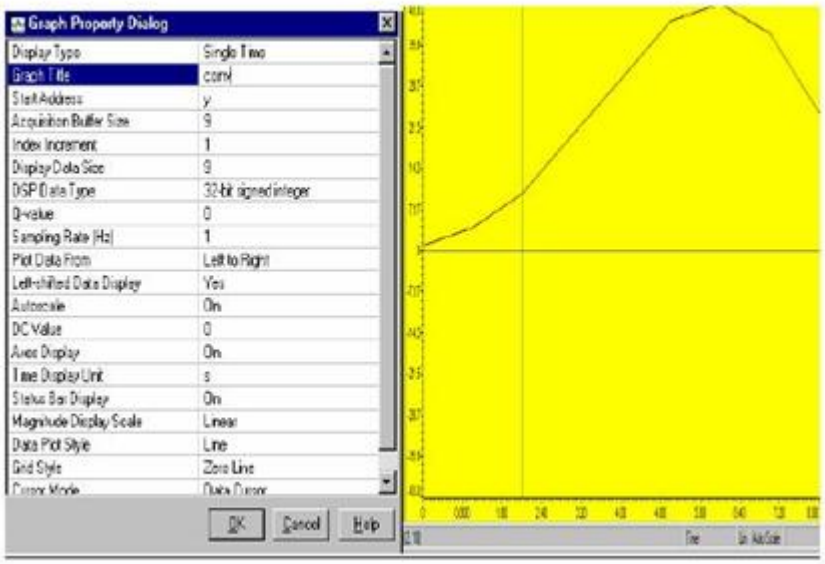
- Compile the program using the .Project-compile. pull down menu or by clicking the shortcut icon on the left side of program window.

- Build the program using the .Project-Build. pull down menu or by clicking the shortcut icon on the left side of program window.

- Load the program(lconv.out) in program memory of DSP chip using the 'File-load program' pull down menu.

- To View output graphically
  - Select view ⇗ graph ⇗ time and frequency

Configure the graphical window as shown below



RESULT:A C-program to implement Linear Convolution is written and output isexecuted and verified using DSK successfully.

EXPERIMENT #4

CIRCULAR CONVOLUTION

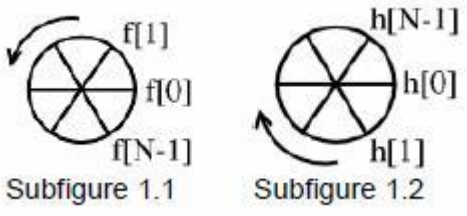
AIM: Write a C-Program implement Linear Convolution, compile and built it using CCSand execute the output using DSK. Plot Time-Frequency graph for the same and verify the output.

SOFTWARE / HARDWARE REQUIREMENTS:

1 Software: CCS(IDE) Code Composer Studio (Integrated Development Environment)
2 C compiler
3 Hardware DSK Kit

CIRCULAR CONVOLUTION

Steps for Cyclic Convolution
Steps for cyclic convolution are the same as the usual convolution, except all index calculations are done "mod N" = "on the wheel"
Steps for Cyclic Convolution
Step1: .Plot f[m] and h[-m]



Subfigure 1.1 Subfigure 1.2

Step 2: "Spin" h[-m] n times Anti Clock Wise (counter-clockwise) to get h[n-m]
(i.e. Simply rotate the sequence, h[n], clockwise by n steps)

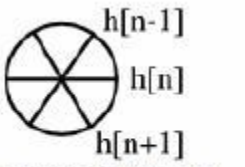


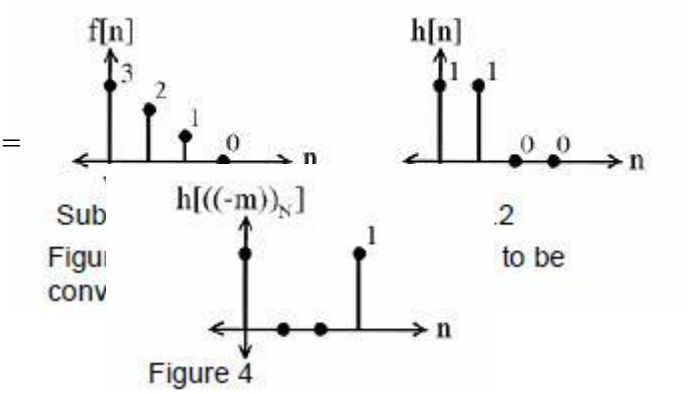
Figure 2: Step 2

Step 3: Point wise multiply the f[m] wheel and the h[n-m] wheel. sum=y[n]

Step 4: Repeat for all $0 \leq n \leq N-1$

Example 1: Convolve (n = 4)

h[−m] =



Sub .2

Figu to be

conv

Figure 4

Multiply f[m] and sum to yield: y[0] =3

h[1−m]

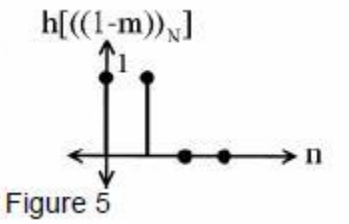


Figure 5

Multiply f[m] and sum to yield: y[1] =5

h[2−m]

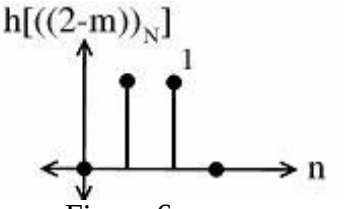


Figure 6

Multiply f[m] and sum to yield: y[2] =3

• h[3−m]

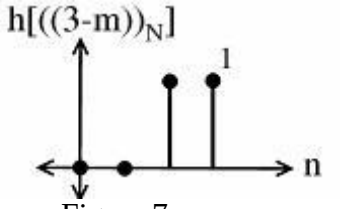


Figure 7

Multiply f[m] and sum to yield: y[3] =1

'C' PROGRAM TO IMPLEMENT CIRCULAR CONVOLUTION

```
#include<stdio.h>

int
m,n,x[30],h[30],y[30],i,j,temp[30],k,x2[30],a[30];
void main()
{

printf(" enter the length of the first sequence\n");
scanf("%d",&m);
printf(" enter the length of the second sequence\n");
scanf("%d",&n);
printf(" enter the first
sequence\n");
for(i=0;i<m;i++)scanf("%d",&x[i])
;
printf(" enter the second
sequence\n"); for(j=0;j<n;j++)
scanf("%d",&h[j]);

if(m-n!=0)                    /*If length of both sequences are not equal*/
{
        if(m>n)               /* Pad the smaller sequence with zero*/
        {
        for(i=n;i<m;i++)
        h[i]=0;
        n=m;
        }
        for(i=m;i<n;i++)
        x[i]=0;
        m=n;
}
y[0]=0;
a[0]=h[0];
for(j=1;j<n;j++)              /*folding h(n) to h(-n)*/
a[j]=h[n-j];

        /*Circular
convolution*/ for(i=0;i<n;i++)
        y[0]+=x[i]*a[i];
for(k=1;k<n;k++)
{
        y[k]=0;
                /*circular shift*/
        for(j=1;j<n;j++)
        x2[j]=a[j-1]; x2[0]=a[n-1];
```

```
        for(i=0;i<n;i++)
        {
                a[i]=x2[i];
                y[k]+=x[i]*x2[i];
        }
}
/*displaying the result*/
printf(" the circular convolution is\n");
for(i=0;i<n;i++)
printf("%d \t",y[i]);
}
```

IN PUT: Eg: x[4]={3, 2, 1,0}
h[4]={1, 1, 0,0}

OUT PUT: y[4]={3, 5, 3,0}

PROCEDURE:

- Open Code Composer Studio, make sure the DSP kit is turned on.
- Start a new project using .Project-new . pull down menu, save it in a separate directory(c:\ti\myprojects) with name **lconv.pjt.**
- Add the source files **Circular Convolution.C**
  - to the project using .Project add files to 'project' pull down menu.
- Add the linker command file **hello.cmd.**
  - (Path: c:\ti\tutorial\dsk6713\hello1\hello.cmd)
- Add the run time support library file **rts6700.lib**
  - (Path: c:\ti\c6000\cgtools\lib\rts6700.lib)
- Compile the program using the .Project-compile. pull down menu or by clicking the shortcut icon on the left side of program window.
- Build the program using the .Project-Build. pull down menu or by clicking the shortcut icon on the left side of program window.
- Load the program(lconv.out) in program memory of DSP chip using the 'File-load program' pull down menu.

RESULT:A C-program to implement Circular Convolution is written and output isexecuted and verified using DSK successfully.

EXPERIMENT # 5

STUDY OF PROCEDURE TO WORK WITH DSK IN REAL TIME

TMS320C6713 DSK CODEC(TLV320AIC23) Configuration Using Board Support Library

1.0 Unit Objective:
To configure the codec TLV320AIC23 for a talk through program using the board upport library.

2.0 Prerequisites
TMS320C6713 DSP Starter Kit, PC with Code Composer Studio, CRO, Audio Source, Speakers and Signal Generator.

3.0 Discussion on Fundamentals:
Refer BSL API Module under, help contents TMS320C6713 DSK.

4.0 Procedure

- All the Real time implementations covered in the Implementations module follow code Configuration using board support library.
- The board Support Library (CSL) is a collection of functions, macros, and symbols used to configure and control on-chip peripherals.
- The goal is peripheral ease of use, shortened development time, portability, hardware abstraction, and some level of standardization and compatibility among TI devices.
- BSL is a fully scalable component of DSP/BIOS. It does not require the use of other DSP/BIOS components to operate.

Source Code: codec.c

Procedure for Real time Programs :

1. Connect CRO to the Socket Provided for **LINE OUT**.

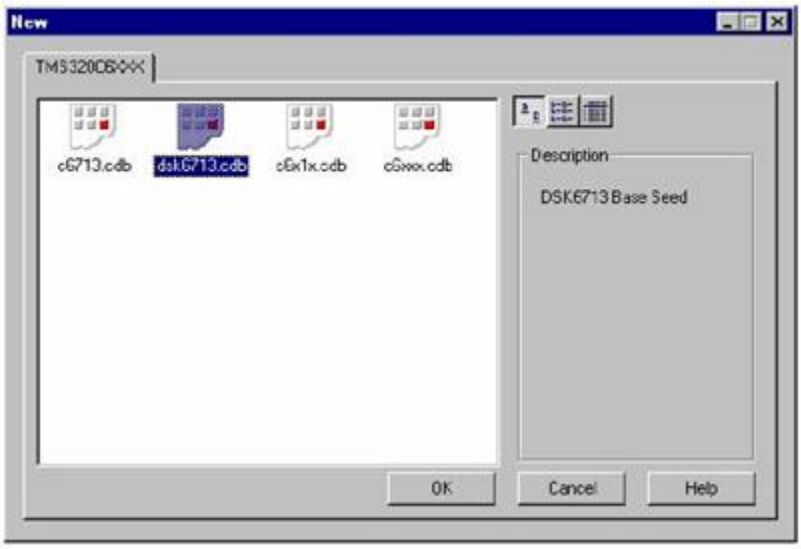
2. Connect a Signal Generator to the **LINE IN** Socket.

3. Switch on the Signal Generator with a sine wave of frequency 500 Hz. and Vp-p=1.5v

4. Now Switch on the DSK and Bring Up Code Composer Studio on the PC.

5. Create a new project with name **codec.pjt**.

6. From the **File** Menu **new DSP/BIOS Configuration** select **"dsk6713.cdb**" and save it as **"xyz.cdb".**



7. Add **"xyz.cdb"** to the current project.

8. Add the given **"codec.c"** file to the current project which has the main function and calls all the other necessary routines.

9. Add the library file "**dsk6713bsl.lib"** to the current project

   Path □ "C:\CCStudio\C6000\dsk6713\lib\dsk6713bsl.lib"

10. Copy files **"dsk6713.h"** and **"dsk6713_aic23.h"** from C:\CCStudio\C6000\dsk6713\include and paste it in current project.

11. Build, Load and Run the program.

12. You can notice the input signal of 500 Hz. appearing on the CRO verifying the codec configuration.

13. You can also pass an audio input and hear the output signal through the speakers.

14. You can also vary the sampling frequency using the **DSK6713_AIC23_setFreq**Function in the **"codec.c"** file and repeat the above steps.

5.0 Conclusion:

The codec TLV320AIC23 successfully configured using the board support library and verified.

EXPERIMENT # 6

DESIGN OF FIR (HP/LP) FILTER

AIM: Write a C-Program design FIR (Low Pass / High Pass) filter, compile and built itusing CCS and execute the output using DSK.

SOFTWARE/HARDWARE REQUIREMENTS:

1 Software: CCS(IDE) Code Composer Studio (Integrated Development Environment)
2 C compiler
3 Hardware:DSK Kit

DESIGNING AN FIR FILTER:

Following are the steps to design linear phase FIR filters Using Windowing Method.

I. Clearly specify the filter specifications.

Eg: Order = 30;
Sampling Rate = 8000 samples/sec
Cut off Freq. = 400 Hz.

II. Compute the cut-off frequency Wc

Eg: Wc = 2*pie* fc / Fs
= 2*pie* 400/8000
= 0.1*pie

III. Compute the desired Impulse Response h d (n) using particular Window Eg: b_rect1=fir1 (order, Wc, 'high', boxcar(31));

IV. Convolve input sequence with truncated Impulse Response x (n)*h (n)

USING MATLAB TO DETERMINE FILTER COEFFICIENTS:Using FIR1 Function on Matlab

B = FIR1(N,Wn) designs an N'th order lowpass FIR digital filter and returns the filter coefficients in length N+1 vector B.

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, i.e., even symmetric coefficients obeying B(k) = B(N+2-k), k = 1,2,...,N+1.

If Wn is a two-element vector, Wn = [W1 W2], FIR1 returns an order N bandpass filter with passband W1 < W < W2.
B = FIR1(N,Wn,'high') designs a highpass filter.
B = FIR1(N,Wn,'stop') is a bandstop filter if Wn = [W1 W2].

If Wn is a multi-element vector,
Wn = [W1 W2 W3 W4 W5 ... WN],
FIR1 returns an order N multiband filter with
bands 0 <W <W1,W1 <W <W2, ..., WN <W < 1.
B = FIR1(N, Wn, 'DC-1') makes the first band a
passband. B = FIR1(N,Wn, 'DC-0') makes the first band a
stopband.
For filters with a passband near Fs/2, e.g., highpass and bandstop filters, N must be even.

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

By default, the filter is scaled so the center of the first pass band has magnitude exactly one after windowing. Use a trailing 'noscale' argument to prevent this scaling, e.g. B = FIR1(N,Wn,'noscale'),
B = FIR1(N,Wn,'high','noscale'),
B = FIR1(N,Wn,wind,'noscale').

Matlab Program to generate 'FIR Filter-Low Pass' Coefficients using FIR1

```
% FIR Low pass filters using rectangular, triangular and kaiser windows
% sampling rate - 8000

order = 30;
cf=[500/4000,1000/4000,1500/4000];        cf--> contains set of cut-off frequencies[Wc ]

% cutoff frequency - 500
b_rect1=fir1(order,cf(1),boxcar(31));   Rectangular
b_tri1=fir1(order,cf(1),bartlett(31));    Triangular
b_kai1=fir1(order,cf(1),kaiser(31,8));   Kaisar [Where 8-->Beta Co-efficient]

% cutoff frequency - 1000
b_rect2=fir1(order,cf(2),boxcar(31));
b_tri2=fir1(order,cf(2),bartlett(31));
b_kai2=fir1(order,cf(2),kaiser(31,8));

% cutoff frequency - 1500
b_rect3=fir1(order,cf(3),boxcar(31));
b_tri3=fir1(order,cf(3),bartlett(31));
b_kai3=fir1(order,cf(3),kaiser(31,8));
fid=fopen('FIR_lowpass_rectangular.txt','wt')
```

```
;

fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -400Hz');
fprintf(fid,'\nfloat b_rect1[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect1);
fseek(fid,-1,0);
fprintf(fid,'};');

fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -800Hz');
fprintf(fid,'\nfloat b_rect2[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect2);
fseek(fid,-1,0);
fprintf(fid,'};');

fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -1200Hz');
fprintf(fid,'\nfloat b_rect3[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect3);
fseek(fid,-1,0);
fprintf(fid,'};');
fclose(fid);

winopen('FIR_highpass_rectangular.txt');
```

T.1 : Matlab generated Coefficients for FIR Low Pass Kaiser filter:

Cutoff -500Hz

```
float b_kai1[31]={-0.000019,-0.000170,-0.000609,-0.001451,-0.002593,-0.003511,-
0.003150,0.000000,0.007551,0.020655,0.039383,0.062306,0.086494,0.108031,0.122944,
0.128279,0.122944,0.108031,0.086494,0.062306,0.039383,0.020655,0.007551,0.000000,
-0.003150,-0.003511,-0.002593,-0.001451,-0.000609,-0.000170,-0.000019};
```

Cutoff -1000Hz

```
float b_kai2[31]={-0.000035,-0.000234,-0.000454,0.000000,0.001933,0.004838,0.005671,
-0.000000,-0.013596,-0.028462,-0.029370,0.000000,0.064504,0.148863,0.221349,0.249983,
0.221349,0.148863,0.064504,0.000000,-0.029370,-0.028462,-0.013596,-0.000000,0.005671,
0.004838,0.001933,0.000000,-0.000454,-0.000234, -0.000035};
```

Cutoff -1500Hz

```
float b_kai3[31]={-0.000046,-0.000166,0.000246,0.001414,0.001046,-0.003421,-0.007410,
0.000000,0.017764,0.020126,-0.015895,-0.060710,-0.034909,0.105263,0.289209,0.374978,
0.289209,0.105263,-0.034909,-0.060710,-0.015895,0.020126,0.017764,0.000000,-0.007410,
-0.003421,0.001046,0.001414,0.000246,-0.000166, -0.000046};
```

T.2: Matlab generated Coefficients for FIR Low Pass Rectangular filter

Cutoff -500Hz

float b_rect1[31]={-0.008982,-0.017782,-0.025020,-0.029339,-0.029569,-0.024895,
-0.014970,0.000000,0.019247,0.041491,0.065053,0.088016,0.108421,0.124473,0.134729,
0.138255,0.134729,0.124473,0.108421,0.088016,0.065053,0.041491,0.019247,0.000000,
-0.014970,-0.024895,-0.029569,-0.029339,-0.025020,-0.017782,-0.008982};

Cutoff -1000Hz

float b_rect2[31]={-0.015752,-0.023869,-0.018176,0.000000,0.021481,0.033416,0.026254,-
0.000000,-0.033755,-0.055693,-0.047257,0.000000,0.078762,0.167080,0.236286,0.262448,
0.236286,0.167080,0.078762,0.000000,-0.047257,-0.055693,-0.033755,-0.000000,0.026254,
0.033416,0.021481,0.000000,-0.018176,-0.023869,-0.015752};

Cutoff -1500Hz

float b_rect2[31]={-0.020203,-0.016567,0.009656,0.027335,0.011411,-0.023194,-0.033672,
0.000000,0.043293,0.038657,-0.025105,-0.082004,-0.041842,0.115971,0.303048,0.386435,
0.303048,0.115971,-0.041842,-0.082004,-0.025105,0.038657,0.043293,0.000000,-0.033672,
-0.023194,0.011411,0.027335,0.009656,-0.016567,-0.020203};

T.3: Matlab generated Coefficients for FIR Low Pass Triangular filter

Cutoff -500Hz

float b_tri1[31]={0.000000,-0.001185,-0.003336,-0.005868,-0.007885,-0.008298,-0.005988,
0.000000,0.010265,0.024895,0.043368,0.064545,0.086737,0.107877,0.125747,0.138255,
0.125747,0.107877,0.086737,0.064545,0.043368,0.024895,0.010265,0.000000,-0.005988,
-0.008298,-0.007885,-0.005868,-0.003336,-0.001185,0.000000};

Cutoff -1000Hz

float b_tri2[31]={0.000000,-0.001591,-0.002423,0.000000,0.005728,0.011139,0.010502,
-0.000000,-0.018003,-0.033416,-0.031505,0.000000,0.063010,0.144802,0.220534,0.262448,
0.220534,0.144802,0.063010,0.000000,-0.031505,-0.033416,-0.018003,-0.000000,0.010502,
0.011139,0.005728,0.000000,-0.002423,-0.001591,0.000000};

Cutoff -1500Hz

float b_tri3[31]={0.000000,-0.001104,0.001287,0.005467,0.003043,-0.007731,-0.013469,
0.000000,0.023089,0.023194,-0.016737,-0.060136,-0.033474,0.100508,0.282844,0.386435,
0.282844,0.100508,-0.033474,-0.060136,-0.016737,0.023194,0.023089,0.000000,-0.013469,
-0.007731,0.003043,0.005467,0.001287,-0.001104,0.000000};

MATLAB Program to generate 'FIR Filter-High Pass' Coefficients using FIR1

```
% FIR High pass filters using rectangular, triangular and Kaiser Windows
% sampling rate - 8000

order = 30;

cf=[400/4000,800/4000,1200/4000];          cf--> contains set of cut-off frequencies[Wc]

% cutoff frequency - 400
b_rect1=fir1(order,cf(1),'high',boxcar(31);
b_tri1=fir1(order,cf(1),'high',bartlett(31));
b_kai1=fir1(order,cf(1),'high',kaiser(31,8));
                                   Where Kaiser(31,8)--> '8'defines the value of 'beta'.
% cutoff frequency - 800
b_rect2=fir1(order,cf(2),'high',boxcar(31));
b_tri2=fir1(order,cf(2),'high',bartlett(31));
b_kai2=fir1(order,cf(2),'high',kaiser(31,8));

% cutoff frequency - 1200
b_rect3=fir1(order,cf(3),'high',boxcar(31));
b_tri3=fir1(order,cf(3),'high',bartlett(31));
b_kai3=fir1(order,cf(3),'high',kaiser(31,8));

fid=fopen('FIR_highpass_rectangular.txt','wt');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -400Hz');
fprintf(fid,'\nfloat b_rect1[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect1);
fseek(fid,-1,0);
fprintf(fid,'};');

fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -800Hz');
fprintf(fid,'\nfloat b_rect2[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect2);
fseek(fid,-1,0);
fprintf(fid,'};');

fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -1200Hz');
fprintf(fid,'\nfloat b_rect3[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect3);
fseek(fid,-1,0);
fprintf(fid,'};');

fclose(fid);
winopen('FIR_highpass_rectangular.txt');
```

T.1 : MATLAB generated Coefficients for FIR High Pass Kaiser filter:

Cutoff -400Hz

```
float b_kai1[31]={0.000050,0.000223,0.000520,0.000831,0.000845,-0.000000,-0.002478,
-0.007437,-0.015556,-0.027071,-0.041538,-0.057742,-0.073805,-0.087505,-0.096739,
0.899998,-0.096739,-0.087505,-0.073805,-0.057742,-0.041538,-0.027071,-0.015556,
-0.007437,-0.002478,-0.000000,0.000845,0.000831,0.000520,0.000223,0.000050};
```

Cutoff -800Hz

```
float b_kai2[31]={0.000000,-0.000138,-0.000611,-0.001345,-0.001607,-0.000000,0.004714,
0.012033,0.018287,0.016731,0.000000,-0.035687,-0.086763,-0.141588,-0.184011,0.800005,
-0.184011,-0.141588,-0.086763,-0.035687,0.000000,0.016731,0.018287,0.012033,0.004714,
-0.000000,-0.001607,-0.001345,-0.000611,-0.000138,0.000000};
```

Cutoff -1200Hz

```
float b_kai3[31]={-0.000050,-0.000138,0.000198,0.001345,0.002212,-0.000000,-0.006489,
-0.012033,-0.005942,0.016731,0.041539,0.035687,-0.028191,-0.141589,-0.253270,0.700008,
-0.253270,-0.141589,-0.028191,0.035687,0.041539,0.016731,-0.005942,-0.012033,-0.006489,
-0.000000,0.002212,0.001345,0.000198,-0.000138,-0.000050};
```

T.2 :MATLAB generated Coefficients for FIR High Pass Rectangular filter:

Cutoff -400Hz

```
float b_rect1[31]={0.021665,0.022076,0.020224,0.015918,0.009129,-0.000000,-0.011158,
-0.023877,-0.037558,-0.051511,-0.064994,-0.077266,-0.087636,-0.095507,-.100422,0.918834,
-0.100422,-0.095507,-0.087636,-0.077266,-0.064994,-0.051511,-0.037558,-0.023877,
-0.011158,-0.000000,0.009129,0.015918,0.020224,0.022076,0.021665};
```

Cutoff -800Hz

```
float b_rect2[31]={0.000000,-0.013457,-0.023448,-0.025402,-0.017127,-0.000000,0.020933,
0.038103,0.043547,0.031399,0.000000,-0.047098,-0.101609,-0.152414,-0.188394,0.805541,
-0.188394,-0.152414,-0.101609,-0.047098,0.000000,0.031399,0.043547,0.038103,0.020933,
-0.000000,-0.017127,-0.025402,-0.023448,-0.013457,0.000000};
```

Cutoff -1200Hz

```
float b_rect3[31]={-0.020798,-0.013098,0.007416,0.024725,0.022944,-0.000000,-0.028043,
-0.037087,-0.013772,0.030562,0.062393,0.045842,-0.032134,-0.148349,-0.252386,0.686050,
-0.252386,-0.148349,-0.032134,0.045842,0.062393,0.030562,-0.013772,-0.037087,-0.028043,
-0.000000,0.022944,0.024725,0.007416,-0.013098,-0.020798};
```

T.3 : MATLAB generated Coefficients for FIR High Pass Triangular filter:

Cutoff -400Hz

float b_tri1[31]={0.000000,0.001445,0.002648,0.003127,0.002391,-0.000000,-0.004383,
-0.010943,-0.019672,-0.030353,-0.042554,-0.055647,-0.068853,-0.081290,-0.092048,
0.902380,-0.092048,-0.081290,-0.068853,-0.055647,-0.042554,-0.030353,-0.019672,
-0.010943,-0.004383,-0.000000,0.002391,0.003127,0.002648,0.001445,0.000000};

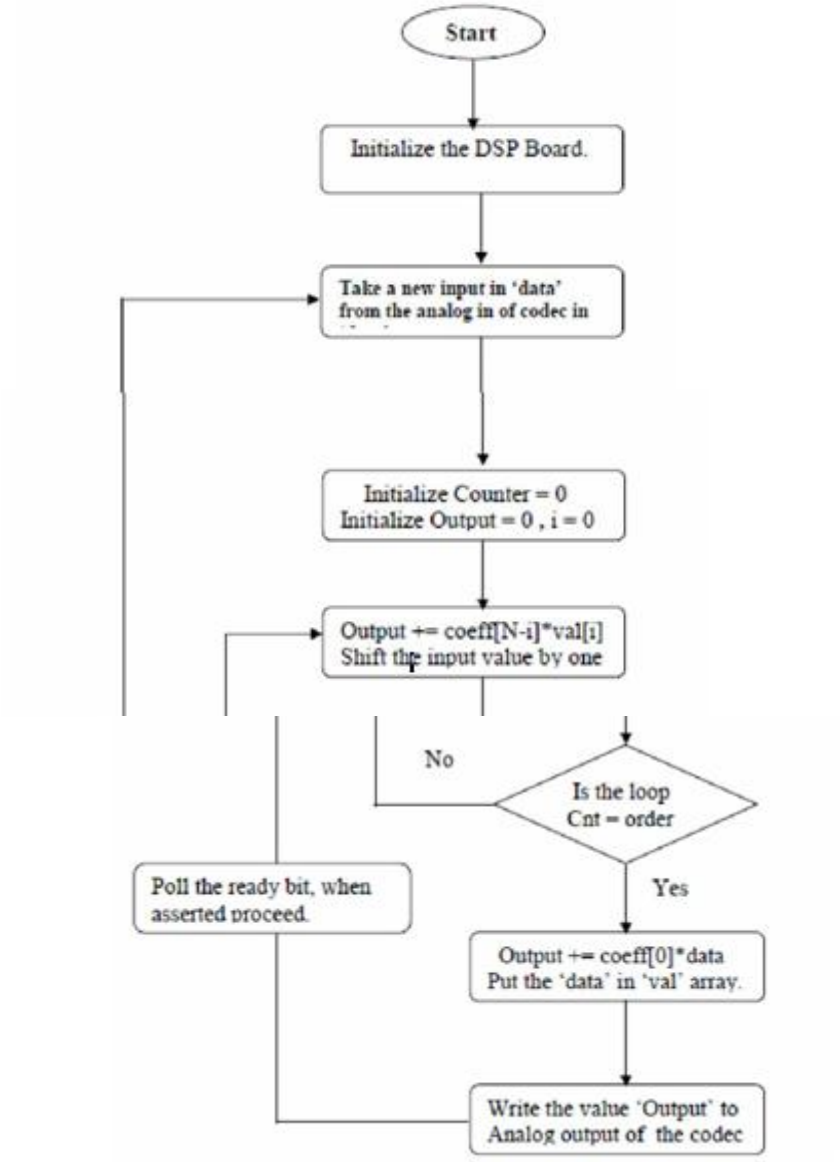
Cutoff -800Hz

float b_tri2[31]={0.000000,-0.000897,-0.003126,-0.005080,-0.004567,-0.000000,0.008373,
0.017782,0.023225,0.018839,0.000000,-0.034539,-0.081287,-0.132092,-0.175834,0.805541,
-0.175834,-0.132092,-0.081287,-0.034539,0.000000,0.018839,0.023225,0.017782,0.008373,
-0.000000,-0.004567,-0.005080,-0.003126,-0.000897,0.000000};

Cutoff -1200Hz

float b_tri3[31]={0.000000,-0.000901,0.001021,0.005105,0.006317,-0.000000,-0.011581,
-0.017868,-0.007583,0.018931,0.042944,0.034707,-0.026541,-0.132736,-0.243196,0.708287,
-0.243196,-0.132736,-0.026541,0.034707,0.042944,0.018931,-0.007583,-0.017868,-0.011581,
-0.000000,0.006317,0.005105,0.001021,-0.000901,0.000000};

FLOW CHART TO IMPLEMENT FIR FILTER:



C PROGRAM TO IMPLEMENT FIR FILTER:

fir.c

```
#include "filtercfg.h"

#include "dsk6713.h"
#include "dsk6713_aic23.h"

float filter_Coeff[] ={0.000000,-0.001591,-0.002423,0.000000,0.005728,0.011139,
```

```
0.010502,-0.000000,-0.018003,-0.033416,-0.031505,0.000000, 0.063010, 0.144802,
0.220534, 0.262448, 0.220534,0.144802,0.063010,0.000000,-0.031505,-0.033416,-
0.018003,-0.000000,0.010502,0.011139,0.005728,0.000000,-0.002423,-0.001591,
0.000000 };

static short in_buffer[100];

DSK6713_AIC23_Config config = {\
    0x0017, /* 0 DSK6713_AIC23_LEFTINVOL Leftline input channel volume */ \
    0x0017, /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume*/ \
    0x00d8, /* 2 DSK6713_AIC23_LEFTHPVOL Left channel headphone volume */ \
    0x00d8, /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume */
    \ 0x0011, /* 4 DSK6713_AIC23_ANAPATH Analog audio path control */ \
    0x0000, /* 5 DSK6713_AIC23_DIGPATH Digital audio path control */ \
    0x0000, /* 6 DSK6713_AIC23_POWERDOWN Power down control */ \
    0x0043, /* 7 DSK6713_AIC23_DIGIF Digital audio interface format */ \
    0x0081, /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */ \
    0x0001 /* 9 DSK6713_AIC23_DIGACT Digital interface activation */ \
};

/*
* main() - Main code routine, initializes BSL and generates tone */


void main()
{
        DSK6713_AIC23_CodecHandle hCodec;

        Uint32 l_input, r_input,l_output, r_output;

        /* Initialize the board support library, must be called first */
        DSK6713_init();

        /* Start the codec */
        hCodec = DSK6713_AIC23_openCodec(0, &config);

        DSK6713_AIC23_setFreq(hCodec, 1);

        while(1)
          { /* Read a sample to the left channel */
                while (!DSK6713_AIC23_read(hCodec, &l_input));

                /* Read a sample to the right channel */
                while (!DSK6713_AIC23_read(hCodec, &r_input));

                        l_output=(Int16)FIR_FILTER(&filter_Coeff ,l_input);
                       r_output=l_output;
```

```
            /* Send a sample to the left channel */
        while (!DSK6713_AIC23_write(hCodec, l_output));

        /* Send a sample to the right channel */
        while (!DSK6713_AIC23_write(hCodec, r_output));
    }

        /* Close the codec */
        DSK6713_AIC23_closeCodec(hCodec);
}

signed int FIR_FILTER(float * h, signed int x)
{
int i=0;
signed long output=0;

in_buffer[0] = x; /* new input at buffer[0] */

for(i=29;i>0;i--)
in_buffer[i] = in_buffer[i-1]; /* shuffle the buffer */

for(i=0;i<31;i++)
output = output + h[i] * in_buffer[i];

return(output);

}
```

PROCEDURE :

- Switch on the DSP board.
- Open the Code Composer Studio.
- Create a new project
    - Project $ New (File Name. pjt , Eg: **FIR.pjt**)
- Initialize on board codec.
    - **Note:** "Kindly refer the Topic **Configuration of 6713 Codec** using BSL"
- Add the given above .C. source file to the current project (**remove codec.c sourcefile from the project if you have already added**).
- Connect the speaker jack to the input of the CRO.
- Build the program.
- Load the generated object file (*.out) on to Target board.
- Run the program
- Observe the waveform that appears on the CRO screen.
- Vary the frequency on function generator to see the response of filter.

EXPERIMENT # 7

DESIGN OF IIR (HP/LP) FILTER

AIM: Write a C-Program to design IIR (Low Pass / High Pass) filter, compile and built itusing CCS and execute the output using DSK.

SOFTWARE/HARDWARE REQUIREMENTS:

1 Software: CCS(IDE) Code Composer Studio (Integrated Development Environment)
2 C compiler
3 Hardware: DSK Kit

DESIGNING AN IIR FILTER:

GENERAL CONSIDERATIONS:

In the design of frequency selective filters, the desired filter characteristics are specified in the frequency domain in terms of the desired magnitude and phase response of the filter. In the filter design process, we determine the coefficients of a causal IIR filter that closely approximates the desired frequency response specifications.

IMPLEMENTATION OF DISCRETE-TIME SYSTEMS:

Discrete time Linear Time-Invariant (LTI) systems can be described completely by constant coefficient linear difference equations. Representing a system in terms of constant coefficient linear difference equation is it's time domain characterization. In the design of a simple frequency selective filter, we would take help of some basic implementation methods for realizations of LTI systems described by linear constant coefficient difference equation.

UNIT OBJECTIVE:

The aim of this laboratory exercise is to design and implement a Digital IIR Filter & observe its frequency response. In this experiment we design a simple IIR filter so as to stop or attenuate required band of frequencies components and pass the frequency components which are outside the required band.

BACKGROUND CONCEPTS:

An Infinite impulse response (IIR) filter possesses an output response to an impulse which is of an infinite duration. The impulse response is "infinite" since there is feedback in the filter, that is if you put in an impulse ,then its output must produced for infinite duration of time.

PREREQUISITES:

- Concept of Discrete time signal processing.
- Analog filter design concepts.
- TMS320C6713 Architecture and instruction set.

EQUIPMENTS NEEDED:

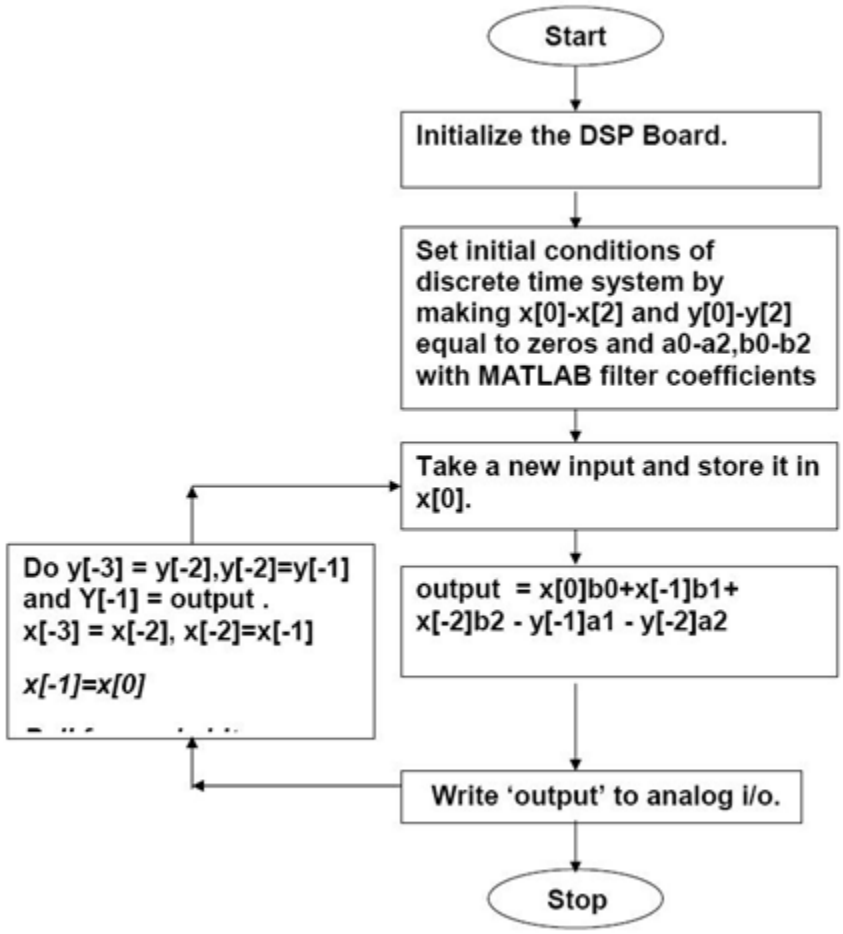
- Host (PC) with windows (95/98/Me/XP/NT/2000).
- TMS320C6713 DSP Starter Kit (DSK).
- Oscilloscope and Function generator.

ALGORITHM TO IMPLEMENT:

We need to realize the Butter worth band pass IIR filter by implementing the difference equation y[n] = b0x[n] + b1x[n-1] + b2x[n-2] - a1y[n-1] - a2y[n-2] where b0-b2, a0-a2 are feed forward and feedback word coefficients respectively [Assume second order of filter]. These coefficients are calculated using MATLAB. A direct form I implementation approach is taken.

- Step 1 -Initialize the McBSP, the DSP board and the on board codec. Kindly refer the Topic **Configuration of 6713Codec** using BSL“
- Step 2 -Initialize the discrete time system, that is, specify the initial conditions. Generally zero initial conditions are assumed.
- Step 3 -Take sampled data from codec while input is fed to DSP kit from the signal generator. Since Codec is stereo, take average of input data read from left and right channel . Store sampled data at a memory location.
- Step 4 - Perform filter operation using above said difference equation and store filter Output at a memory location .
- Step 5 -Output the value to codec (left channel and right channel) and view the output at Oscilloscope.
- Step 6 -Go to step 3.

FLOWCHART FOR IIR IMPLEMENTATION:



F.1 : Flowchart for implementing IIR filter.

MATLAB PROGRAM TO GENRATE FILTER CO-EFFICIENTS

```
% IIR Low pass Butterworth and Chebyshev filters
% sampling rate - 24000
order = 2;
cf=[2500/12000,8000/12000,1600/12000]
;

% cutoff frequency -2500
[num_bw1,den_bw1]=butter(order,cf(1));
[num_cb1,den_cb1]=cheby1(order,3,cf(1));

% cutoff frequency -8000
[num_bw2,den_bw2]=butter(order,cf(2));
[num_cb2,den_cb2]=cheby1(order,3,cf(2));

fid=fopen('IIR_LP_BW.txt','wt');
fprintf(fid,'\t\t---------- Pass band range: 0-2500Hz----------\n');
```

```
fprintf(fid,'\t\t---------- Magnitude response: Monotonic----- \n\n\');
fprintf(fid,'\n float num_bw1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_bw1);
fprintf(fid,'\nfloat den_bw1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_bw1);

fprintf(fid,'\n\n\n\t\t---------- Pass band range: 0-8000Hz---------- \n');
fprintf(fid,'\t\t---------- Magnitude response: Monotonic----- \n\n');
fprintf(fid,'\nfloat num_bw2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_bw2);
fprintf(fid,'\nfloat den_bw2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_bw2);

fclose(fid);
winopen('IIR_LP_BW.txt');

fid=fopen('IIR_LP_CHEB Type1.txt','wt');
fprintf(fid,'\t\t---------- Pass band range: 2500Hz---------- \n');
fprintf(fid,'\t\t---------- Magnitude response: Rippled (3dB) ----- \n\n\');
fprintf(fid,'\nfloat num_cb1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_cb1);
fprintf(fid,'\nfloat den_cb1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_cb1);
fprintf(fid,'\n\n\n\t\t---------- Pass band range: 8000Hz---------- \n');
fprintf(fid,'\t\t---------- Magnitude response: Rippled (3dB)----- \n\n');
fprintf(fid,'\nfloat num_cb2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_cb2);
fprintf(fid,'\nfloat den_cb2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_cb2);

fclose(fid);
winopen('IIR_LP_CHEB Type1.txt');

%%%%%%%%%%%%%%%%%
figure(1);
[h,w]=freqz(num_bw1,den_bw1);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2)
hold on
[h,w]=freqz(num_cb1,den_cb1);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2,'color','r')
grid on
legend('Butterworth','Chebyshev Type-
1'); xlabel('Frequency in Hertz');
ylabel('Magnitude in Decibels');
title('Magnitude response of Low pass IIR filters (Fc=2500Hz)');
```

```
figure(2);
[h,w]=freqz(num_bw2,den_bw2);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2)
hold on
[h,w]=freqz(num_cb2,den_cb2);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2,'color','r')
grid on
legend('Butterworth','Chebyshev Type-1 (Ripple: 3dB)');
xlabel('Frequency in Hertz');
ylabel('Magnitude in Decibels');
title('Magnitude response in the passband');
axis([0 12000 -20 20]);
```

IIR_CHEB_LP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=800Hz		Fc=8000Hz	
	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)
B0	0.044408	1455	0.005147	168	0.354544	11617
B1	0.088815	1455[B1/2]	0.010295	168[B1/2]	0.709088	11617[B1/2]
B2	0.044408	1455	0.005147	168	0.354544	11617
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.412427	-23140[A1/2]	-1.844881	-30225[A1/2]	0.530009	8683[A1/2]
A2	0.663336	21735	0.873965	28637	0.473218	15506

Note: We have Multiplied Floating Point Values with $32767(2^{15})$ to get Fixed Point Values.

IIR_BUTTERWORTH_LP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=800Hz		Fc=8000Hz	
	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)
B0	0.072231	2366	0.009526	312	0.465153	15241
B1	0.144462	2366[B1/2]	0.019052	312[B1/2]	0.930306	15241[B1/2]
B2	0.072231	2366	0.009526	312	0.465153	15241
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.109229	-18179[A1/2]	-1.705552,	-27943[A1/2]	0.620204	10161[A1/2]
A2	0.398152	13046	0.743655	24367	0.240408	7877

Note: We have Multiplied Floating Point Values with $32767(2^{15})$ to get Fixed Point Values.

IIR_CHEB_HP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=4000Hz		Fc=7000Hz	
	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)
B0	0.388513	12730	0.282850	9268	0.117279	3842
B1	-0.777027	-12730[B1/2]	-0.565700	-9268[B1/2]	-0.234557	-3842[B1/2]
B2	0.388513	12730	0.282850	9268	0.117279	3842
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.118450	-18324[A1/2]	-0.451410	-7395[A1/2]	0.754476	12360[A1/2]
A2	0.645091	21137	0.560534	18367	0.588691	19289

Note: We have Multiplied Floating Point Values with $32767(2^{15})$ to get Fixed Point Values.

IIR_BUTTERWORTH_HP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=4000Hz		Fc=7000Hz	
	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)
B0	0.626845	20539	0.465153	15241	0.220195	7215
B1	-1.253691	-20539[B1/2]	-0.930306	-15241[B1/2]	-0.440389	-7215[B1/2]
B2	0.626845	20539	0.465153	15241	0.220195	7215
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.109229	-18173[A1/2]	-0.620204	-10161[A1/2]	0.307566	5039[A1/2]
A2	0.398152	13046	0.240408	7877	0.188345	6171

Note: We have Multiplied Floating Point Values with $32767(2^{15})$ to get Fixed Point Values.

'C' PROGRAM TO IMPLEMENT IIR FILTER

```
#include "xyzcfg.h"

#include "dsk6713.h"
#include "dsk6713_aic23.h"

const signed int filter_Coeff[] =
{
        //12730,-12730,12730,2767,-18324,21137 /*HP 2500 */
```

```
        //312,312,312,32767,-27943,24367 /*LP 800 */
        //1455,1455,1455,32767,-23140,21735 /*LP 2500 */
        //9268,-9268,9268,32767,-7395,18367 /*HP 4000*/
           7215,-7215,7215,32767,5039,6171, /*HP 7000*/
} ;

/* Codec configuration settings */

DSK6713_AIC23_Config config = { \
0x0017, /* 0 DSK6713_AIC23_LEFTINVOL Left line input channel volume */ \
0x0017, /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume */\
0x00d8, /* 2 DSK6713_AIC23_LEFTHPVOL Left channel headphone volume */ \
0x00d8, /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume */
\ 0x0011, /* 4 DSK6713_AIC23_ANAPATH Analog audio path control */ \
0x0000, /* 5 DSK6713_AIC23_DIGPATH Digital audio path control */ \
0x0000, /* 6 DSK6713_AIC23_POWERDOWN Power down control */
\ 0x0043, /* 7 DSK6713_AIC23_DIGIF Digital audio interface format */
\ 0x0081, /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */
\ 0x0001 /* 9 DSK6713_AIC23_DIGACT Digital interface activation */
\ };

/*
* main() - Main code routine, initializes BSL and generates tone
*/
void main()
{

        DSK6713_AIC23_CodecHandle hCodec;

        int l_input, r_input, l_output, r_output;

        /* Initialize the board support library, must be called first */
        DSK6713_init();

        /* Start the codec */
        hCodec = DSK6713_AIC23_openCodec(0, &config);

        DSK6713_AIC23_setFreq(hCodec, 3);

                while(1)
                { /* Read a sample to the left channel */
                        while (!DSK6713_AIC23_read(hCodec, &l_input));

                        /* Read a sample to the right channel */
                        while (!DSK6713_AIC23_read(hCodec, &r_input));

                                l_output=IIR_FILTER(&filter_Coeff ,l_input);
```

```
                        r_output=l_output;

                    /* Send a sample to the left channel */
                        while (!DSK6713_AIC23_write(hCodec, l_output));

                    /* Send a sample to the right channel */
                        while (!DSK6713_AIC23_write(hCodec, r_output));
                }

        /* Close the codec */
        DSK6713_AIC23_closeCodec(hCodec);
}

signed int IIR_FILTER(const signed int * h, signed int x1)
{
        static signed int x[6] = { 0, 0, 0, 0, 0, 0 }; /* x(n), x(n-1), x(n-2). Must be static */
        static signed int y[6] = { 0, 0, 0, 0, 0, 0 }; /* y(n), y(n-1), y(n-2). Must be static */
        int temp=0;

        temp = (short int)x1; /* Copy input to temp */

        x[0] = (signed int) temp; /* Copy input to x[stages][0] */

        temp = ( (int)h[0] * x[0]) ; /* B0 * x(n) */

        temp += ( (int)h[1] * x[1]); /* B1/2 * x(n-1)
        */ temp += ( (int)h[1] * x[1]); /* B1/2 * x(n-
        1) */ temp += ( (int)h[2] * x[2]); /* B2 * x(n-
        2) */

        temp -= ( (int)h[4] * y[1]); /* A1/2 * y(n-1)
        */ temp -= ( (int)h[4] * y[1]); /* A1/2 * y(n-
        1) */ temp -= ( (int)h[5] * y[2]); /* A2 * y(n-
        2) */

    /* Divide temp by coefficients[A0] */

        temp >>= 15;

    if ( temp > 32767 )
        {
                temp = 32767;
        }
        else if ( temp < -32767)
        {
                temp = -32767;
        }
```

```
y[0] = temp ;

        /* Shuffle values along one place for next time */

        y[2] = y[1]; /* y(n-2) = y(n-1)
        */ y[1] = y[0]; /* y(n-1) = y(n)
        */

        x[2] = x[1]; /* x(n-2) = x(n-1)
        */ x[1] = x[0]; /* x(n-1) = x(n)
        */

        /* temp is used as input next time through */

   return (temp<<2);
}
```

PROCEDURE :

- Switch on the DSP board.
- Open the Code Composer Studio.
- Create a new project
    - Project $ New (File Name. pjt , Eg: **IIR.pjt**)
- Initialize on board codec.

    - Note: **"**Kindly refer the Topic**Configuration of 6713 Codec using BSL**"

- Add the given above .C. source file to the current project (**remove codec.c sourcefile from the project if you have already added**).
- Connect the speaker jack to the input of the CRO.
- Build the program.
- Load the generated object file(*.out) on to Target board.
- Run the program
- Observe the waveform that appears on the CRO screen.
    - Vary the frequency on function generator to see the response of filter

RESULT:A C-program to design IIR (Low Pass/High Pass) filter is written and outputis executed and verified using DSK successfully.

EXPERIMENT #8
FFT OF A GIVEN 1-D SIGNAL

AIM: Write a C-Program to find FFT of a 1-D signal, compile and built it using CCS andexecute the output using DSK. Plot the Time-frequency graph for Input and Output

SOFTWARE/HARDWARE REQUIREMENTS:

1 Software: CCS(IDE) Code Composer Studio (Integrated Development Environment)
2 C compiler
3 Hardware: DSK Kit

Fast Fourier Transforms(FFT):

The DFT Equation

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

Where $W_N^{nk} = e^{-j\frac{2\pi nk}{N}}$ [Twiddle Factor]

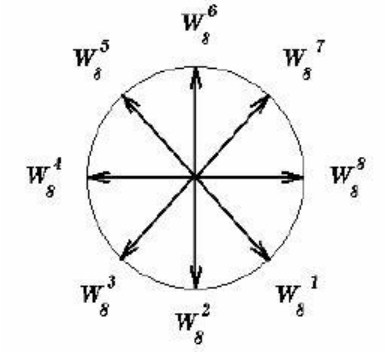
Twiddle Factor

In the Definition of the DFT, there is a factor called the Twiddle Factor

$$W_N^{nk} = e^{-j\frac{2\pi nk}{N}}$$

where N = number of samples.

If we take an 8 bit sample sequence we can represent the twiddle factor as a vector in the unit circle. e.g.



Note that

1. It is periodic. (i.e. it goes round and round the circle !!)
2. That the vectors are symmetric

The vectors are equally spaced around the circle.

Why the FFT?

If you look at the equation for the Discrete Fourier Transform you will see that it is quite complicated to work out as it involves many additions and multiplications involving complex numbers. Even a simple eight sample signal would require 49 complex multiplications and 56 complex additions to work out the DFT. At this level it is still manageable; however a realistic signal could have 1024 samples which requires over 20,000,000 complex multiplications and additions. As you can see the number of calculations required soon mounts up to unmanageable proportions.

The Fast Fourier Transform is a simply a method of laying out the computation, which is much faster for large values of N, where N is the number of samples in the sequence. It is an ingenious way of achieving rather than the DFT's clumsy P^2 timing.

The idea behind the FFT is the divide and conquer approach, to break up the original N point sample into two (N / 2) sequences. This is because a series of smaller problems is easier to solve than one large one. The DFT requires (N-1)^2 complex multiplications and N(N-1) complex additions as opposed to the FFT's approach of breaking it down into a series of 2 point samples which only require 1 multiplication and 2 additions and the recombination of the points which is minimal.

For example Seismic Data contains hundreds of thousands of samples and would take months to evaluate the DFT. Therefore we use the FFT.

FFT Algorithm

The FFT has a fairly easy algorithm to implement, and it is shown step by step in the list below. Thjis version of the FFT is the Decimation in Time Method

1. Pad input sequence, of N samples, with ZERO's until the number of samples is the nearest power of two.
2. Bit reverse the input sequence.
   e.g. 3 = 011 goes to 110 = 6
3. Compute (N / 2) two sample DFT's from the shuffled inputs. See "Shuffled Inputs"
4. Compute (N / 4) four sample DFT's from the two sample DFT's. See "Shuffled Inputs"
5. Compute (N / 2) eight sample DFT's from the four sample DFT's. See "Shuffled Inputs"
6. Until the all the samples combine into one N-sample DFT

Shuffled Inputs

The process of decimating the signal in the time domain has caused the INPUT samples to be re-ordered. For an 8 point signal the original order of the samples is

0, 1, 2, 3, 4, 5, 6, 7

But after decimation the order is

0, 4, 2, 6, 1, 5, 3, 7

At first it may look as if there is no order to this new sequence, BUT if the numbers are represented as binary a patter soon becomes apparent.

ORIGINAL INPUT			RE-ORDERED INPUT	
Decimal	Binary		Binary	Decimal
0	000	⟷	000	0
1	001		100	4
2	010		010	2
3	011	⟷	110	6
4	100		001	1
5	101		101	5
6	110		011	3
7	111	⟷	111	7

What has happened is that the bit patterns representing the sample number has been reversed. This new sequence is the order that the samples enter the FFT.

ALGORITHM TO IMPLEMENT FFT:

- Step 1 -Select no. of points for FFT(Eg: 64)
- Step 2 – Generate a sine wave of frequency .f . (eg: 10 Hz with a sampling rate = No. of Points of FFT(eg. 64)) using math library function.
- Step 3 -Take sampled data and apply FFT algorithm .
- Step 4 – Use Graph option to view the Input & Output.
- Step 5 -Repeat Step-1 to 4 for different no. of points & frequencies.

C PROGRAM TO IMPLEMENT FFT :

Main.c (fft 256.c):

```
#include <math.h>
#define PTS 64                                  //# of points for FFT
#define PI 3.14159265358979

typedef struct {float real,imag;} COMPLEX;

void FFT(COMPLEX *Y, int n);                    //FFT prototype
float iobuffer[PTS];                            //as input and output buffer
float x1[PTS];                                  //intermediate buffer
short i;                                        //general purpose index variable
short buffercount = 0;                          //number of new samples in
iobuffer
short flag = 0;                                 //set to 1 by ISR when iobuffer
full
COMPLEX w[PTS];                                  //twiddle constants stored in w
COMPLEX samples[PTS];                           //primary working buffer

main( )
{
        for (i = 0 ; i<PTS ; i++) // set up twiddle constants in w
        {
        w[i].real = cos(2*PI*i/(PTS*2.0));          //Re component of twiddle constants
        w[i].imag =-sin(2*PI*i/(PTS*2.0));          //Im component of twiddle constants
        }

        for (i = 0 ; i < PTS ; i++) //swap buffers
        {
        iobuffer[i] = sin(2*PI*10*i/64.0);          /*10- > freq, 64 -> sampling freq*/
        samples[i].real=0.0;
        samples[i].imag=0.0;
        }

        for (i = 0 ; i < PTS ; i++)             //swap buffers
        {
        samples[i].real=iobuffer[i];            //buffer with new data
        }
        for (i = 0 ; i < PTS ; i++)

        samples[i].imag = 0.0;                  //imag components = 0

FFT(samples,PTS);                               //call function FFT.c
```

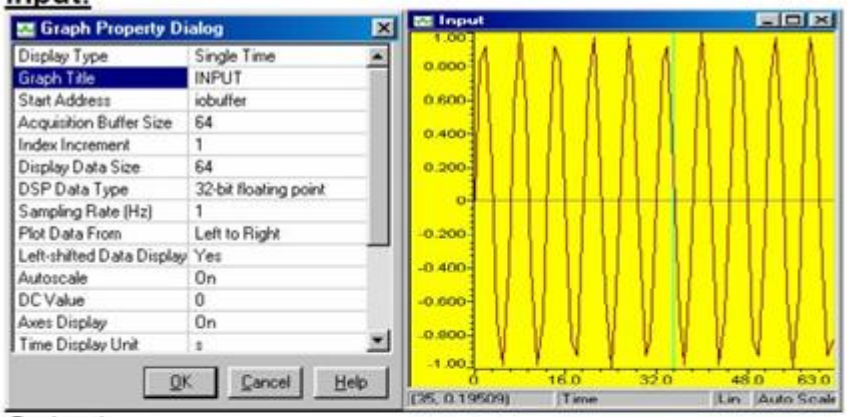
```
        for (i = 0 ; i < PTS ; i++)                 //compute magnitude
        {
        x1[i] = sqrt(samples[i].real*samples[i].real + samples[i].imag*samples[i].imag);
        }
} //end of main
```

fft.c:

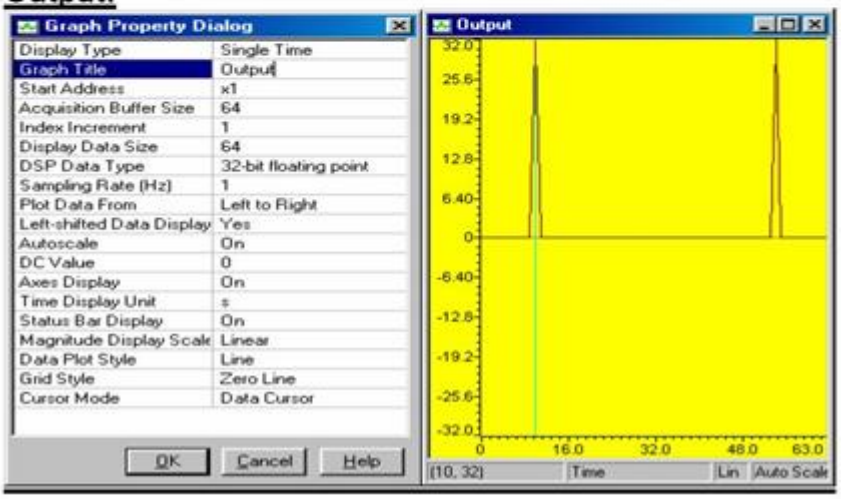
```
#define PTS 64                                      //# of points for FFT
typedef struct {float real,imag;} COMPLEX;
extern COMPLEX w[PTS];                              //twiddle constants stored in w

void FFT(COMPLEX *Y, int N)                         //input sample array, # of points
{
COMPLEX temp1,temp2;                                //temporary storage variables
int i,j,k;                                          //loop counter variables
int upper_leg, lower_leg;                           //index of upper/lower butterfly leg
int leg_diff;                                       //difference between upper/lower leg
int num_stages = 0;                                 //number of FFT stages (iterations)
int index, step;                                    //index/step through twiddle constant
i = 1;                                              //log(base2) of N points= # of stages

do
{
  num_stages +=1;
  i = i*2;
}while (i!=N);

leg_diff = N/2;                                     //difference between upper&lower legs
step = (PTS*2)/N;                                   //step between values in twiddle.h
for (i = 0;i < num_stages; i++)                     //for N-point FFT
{
    index = 0;
    for (j = 0; j < leg_diff; j++)
    {
         for (upper_leg = j; upper_leg < N; upper_leg += (2*leg_diff))
        {
         lower_leg = upper_leg+leg_diff;
        temp1.real = (Y[upper_leg]).real + (Y[lower_leg]).real;
        temp1.imag = (Y[upper_leg]).imag +
        (Y[lower_leg]).imag; temp2.real = (Y[upper_leg]).real -
        (Y[lower_leg]).real; temp2.imag = (Y[upper_leg]).imag -
        (Y[lower_leg]).imag;
        (Y[lower_leg]).real = temp2.real*(w[index]).real -temp2.imag*(w[index]).imag;
        (Y[lower_leg]).imag =
```

```
            temp2.real*(w[index]).imag+temp2.imag*(w[index]).real;

            (Y[upper_leg]).real = temp1.real;
            (Y[upper_leg]).imag = temp1.imag;
            }
      index += step;
        }
          leg_diff =
          leg_diff/2; step *=
          2;
           }
           j = 0;
           for (i = 1; i < (N-1); i++)                    //bit reversal for resequencing data
           {
           k = N/2;
           while (k <= j)
           {
           j = j - k;
           k = k/2;
           }
           j = j + k;

if (i<j)
           {
           temp1.real = (Y[j]).real;
           temp1.imag =
           (Y[j]).imag; (Y[j]).real =
           (Y[i]).real; (Y[j]).imag =
           (Y[i]).imag; (Y[i]).real =
           temp1.real; (Y[i]).imag =
           temp1.imag;
           }
       }
      return;
}
```

Input:



Output:



HOW TO PROCEED

- Open Code Composer Studio, make sure the DSP kit is turned on.
- Start a new project using .Project-new . pull down menu, save it in a separate directory(c:\ti\myprojects) with name "**FFT.pjt**".
- Add the source files "**FFT256.c"** and **"FFT.C"** in the project using 'Project$add files to project' pull down menu.
- Add the linker command file .hello.cmd"
- Add the rts file "rts6700.lib"
- Compile the program using the .Project-compile. pull down menu or by clicking the shortcut icon on the left side of program window.
- Load the program in program memory of DSP chip using the .File-load

program. pull down menu.

- Run the program and observe output using graph utility.

RESULT:A C-program to find FFT of a 1-D signal is written and output is executed andverified using DSK successfully. Time-frequency graph has been plotted for input and output for the same

APPENDIX -1

MATLAB

The Language of Technical Computing

What is MATLAB?

MATLAB is a high performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment, where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation and prototyping
- Data analysis, exploration and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MA TLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or FORTRAN.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MA TLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of add-on application-specific solutions called toolboxes. Very important to most users of MATLAB. Toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation and many others.

The MA TLAB System

The MATLAB system consists of five main parts:

Development Environment: This is the set of tools and facilities that help you useMATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, an editor and debugger, and browsers for viewing help, the workspace, files and the search path.

The MATLAB Mathematical Function Library: This is a vast collection ofcomputational algorithms ranging from elementary functions like sum, sine, cosine and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigen values, Bessel functions, and Fast Fourier transforms.

The MATLAB Language: This is a high-level. Matrix/array language with control flowstatements, functions, data structures, input/output, and object -oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create large and complex application programs.

.

Graphics: MATLAB has extensive facilities for displaying vectors and matrices asgraphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low- level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

The MA TLAB Application Program Interface (API): This is a library that allows youto write C and FORTRAN programs that interact with MATLAB. It includes facilities for calling routines from MA TLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

Explanation of Some Commands

STEM

Discrete sequence or "stem" plot.

STEM(Y) plots the data sequence Y as stems from the x axis terminated with circles forthe data value. If Y is a matrix then each column is plotted as a separate series.

STEM(X,Y) plots the data sequence Y at the values specified in X.

STEM(...,'filled') produces a stem plot with filled markers.

STEM (..,. 'LINESPEC') uses the line type specified for the stems and markers.

STEM(AX,.. } plots into axes with handle AX. Use GCA to get the handle to thecurrent axes or to create one if none exist.

H = STEM(...) returns a vector of stem series handles in H, one handle per column of datain Y.

PLOT

Linear plot.

PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix, then the vector isplotted versus the rows or columns of the matrix which ever line up. If X is a scalar and Y is a vector, length(Y) disconnected points are plotted.

PLOT(Y) plots the columns of Y versus their index.
If Y is complex, PLOT(Y) is equivalent to PLOT (real(Y), imag(Y)). In all other uses of PLOT, the imaginary part is ignored.

Various line types, plot symbols and colors may be obtained with PLOT(X, Y,S) where S is a character string made from one element from any or all the following 3 columns:

b blue	. point	- solid
g green	0 circle	: dotted
r red	x x-mark	-. dashdot
c cyan	+ plus	-- dashed
m magenta	* star	(none) no line
y yellow	s square	p pentagram
k black	d diamond	
v triangle (down)	/\ triangle (up)	
	< triangle (left)	
	> triangle (right)	

For example, (i) PLOT(X,Y,'c+:') plots a cyan dotted line with a plus at each data point;
(ii) PLOT(X,Y,'bd') plots blue diamond at each data point but does not draw any line.

PLOT (Xl,Yl,Sl.x2,Y2,S2.x3,Y3,S3,...) combines the plots defined by the (X, Y,S) triples,where the X's and Y's are vectors or matrices and the S's are strings.

For example, PLOT(X,Y,'y -',X Y,'go') plots the data twice, with a solid yellow line interpolating green circles at the data points.

The PLOT command, if no color is specified, makes automatic use of the colors specified by the axes Color Order property. The default Color Order is listed in the table above for color systems where the default is blue for one line, and for multiple lines, to cycle through the first six colors in the table. For monochrome systems,

PLOT cycles over the axes Line Style Order property.

If you do not specify a marker type, PLOT uses no marker.
If you do not specify a line style, PLOT uses a solid line.

PLOT(AX,...) plots into the axes with handle AX.
PLOT returns a column vector of handles to line series objects, one handle per plottedline.

The X, Y pairs, or X, Y,S triples, can be followed by parameter/value pairs to specify additional properties of the lines. For example, PLOT(X,Y,'LineWidth',2,'Color',[6 0 0]) will create a plot with a dark red line width of 2 points. .

SUBPLOT

Create axes in tiled positions.

H = SUBPLOT(m,n,p), or SUBPLOT(mnp), breaks the Figure window into an m-by-nmatrix of small axes, selects the p-th axes for the current plot, and returns the axis handle. The axes are counted along the top row of the Figure window, then the second row, etc.

For example,

```
SUBPLOT(2,1,1), PLOT(income)
SUBPLOT(2,1,2), PLOT( outgo)
```

It plots income on the top half of the window and outgo on the bottom half

SUBPLOT (m,n,p), if the axis already exists, makes it current.

SUBPLOT(m,n,p, 'replace'), if the axis already exists, deletes it and creates a new axis.

SUBPLOT(m,n,p, 'align') places the axes so that the plot boxes are aligned instead ofpreventing the labels and ticks from overlapping.

SUBPLOT(m,n,P), where P is a vector, specifies an axes position that covers all thesubplot positions listed in P.

SUBPLOT(H), where H is an axis handle, is another way of making an axis current forsubsequent plotting commands.

SUBPLOT('position',[left bottom width heigh]) creates an axis at the specified positionin normalized coordinates (in the range from 0.0 to 1.0).

*SUBPLOT(m,n,p, PROP*1, *V ALVEI, PROP2, V ALVE2,* ...) sets the specified property-value pairs on the subplot axis. To add the subplot to a specific figure pass the figure handle as the value for the 'Parent' property.

If a SUBPLOT specification causes a new axis to overlap an existing axis, the existing axis is deleted - unless the position of the new and existing axis are identical For example, the statement SUBPLOT(l, 2 ,1) deletes all existing axes overlapping the left side of the Figure window and creates a new axis on that side - unless there is an axes there with a position that exactly matches the position of the new axes (and 'replace' was not specified), in which case all other overlapping axes will be deleted and the matching axes will become the current axes.

SUBPLOT(111) is an exception to the rules above, and is not identical in behavior to SUBPLOT(1,1,1). For reasons of backwards compatibility, it is a special case of subplot which does not immediately create an axes, but instead sets up the figure so that the next graphics command executes CLF RESET in the figure (deleting all children of the figure), and creates a new axes in the default position. This syntax does not return a handle, so it is an error to specify a return argument. The delayed CLF RESET is accomplished by setting the figure's NextPlot to 'replace' .

SIZE

Size of an array.

D = SIZE(X), for M-by-N matrix X, returns the two-element row vector D = [M, N]containing the number of rows and columns in the matrix. For N-D arrays, SIZE(X) returns a l-by-N vector of dimension lengths. Trailing singleton dimensions are ignored.

[M, N] = SIZE(X) for matrix X, returns the number of rows and columns in X as separateoutput variables.

[Ml.M2.M3..... MN] = SIZE(X) returns the sizes of the first N dimensions of arrayX If the number of output arguments N does not equal NDIMS(X), then for:

N > NDIMS(X). size returns ones in the "extra" variables. i.e.. outputs NDIMS(X)+ 1 through N.

N < NDIMS(X). MN contains the product of the sizes of the remaining dimensions. i.e., dimensions N+ 1 through NDIMS(X)

M = SIZE(X.DIM) returns the length of the dimension specified by the scalar DIM. For example, SIZE(X, l) returns the number of rows.

When SIZE is applied to a Java array, the number of rows returned is the length of the Java array and the number of columns is always 1. When SIZE is applied to a Java array of arrays, the result describes only the top level array in the array of arrays.

INPUT

Prompt for user input

R = INPUT('How many apples') gives the user the prompt in the text string and then waitsfor input from the keyboard The input can be any MATLAB expression, which is evaluated, using the variables in the current workspace, and the result returned in R If the user presses the return key without entering anything. INPUT returns an empty matrix.

R = INPUT ('What is your nam','s') gives the prompt in the text string and waits forcharacter string input. The typed input is not evaluated; the characters are simply returned as a MATLAB string. .

XLABEL

X-axis label.

XLABEL('text') adds text beside the X-axis on the current axis.

XLABEL('text','Property1','PropertyValue','Property2','PropertyValue2',...) sets thevalues of the specified properties of the xlabel.

XLABEL(AX, ..) adds the xlabel to the specified axes.

H = XLABEL(...) returns the handle to the text object used as the label.

YLABEL

Y-axis label.

YLABEL ('text') adds text beside the Y-axis on the current axis.

YLABEL ('text','Propertyl','PropertyValuel','Property2','PropertyValue2',...) sets thevalues of the specified properties of the ylabel.

YLABEL (AX,...) adds the ylabel to the specified axes.

H = YLABEL (..) returns the handle to the text object used as the label.

TITLE

Graph title.

TITLE ('text') adds text at the top of the current axis.

TITLE ('text','Propertyl','PropertyValuel','Property2','PropertyValue2',...) sets

thevalues of the specified properties of the title.

TITLE(AX,...) adds the title to the specified axes.

H = TITLE(..) returns the handle to the text object used as the title.

CONV

Convolution and polynomial multiplication.

C = *CONV(A, B)* convolves vectors A and B. The resulting vector is length LENGTH(A)+LENGTH(B)-l.
If A and B are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials.

ZEROS

Zeros array

ZEROS(N) is an N-by-N matrix of zeros.
ZEROS(M,N) or *ZEROS([M,N])* is an M-by-N matrix of zeros.
ZEROS(M,N,P,...) or *ZEROS([M N P* ...]) is an M-by-N-by-P-by- ... array of zeros.
ZEROS(SIZE(A)) is the same size as A and all zeros.

ZEROS with no arguments is the scalar 0.

ZEROS(M,N ... ,CLASSNAME) or *ZEROS([M,N ....],CLASSNAME)* is an M-by-NBy-... array of zeros of class CLASSNAME.

ONES

Ones array.

ONES(N) is an N-by-N matrix of ones.

ONES(M,N) or *ONES([M,N])* is an M-by-N matrix of ones.

ONES(M,N,P,...) or *ONES([M N P* ...]) is an M-by-N-by-P-by- .... array of ones.

ONES(SIZE(A)) is the same size as A and all ones.

ONES with no arguments is the scalar 1.

ONES(M,N,...,CLASSNAME) or *ONES([M,N ...],CLASSNAME)* is an M-by-N-by- ..

... array of ones ofc1ass CLASSNAME.

FOR

Repeat statements a specific number of times.

The general form of a FOR statement is:

FOR variable = expr, statement, …, statement END

The columns of the expression are stored one at a time in the variable and then the following statements, up to the END, are executed. The expression is often of the form X:Y, in which case its columns are simply scalars. Some examples (assume N has already been assigned a value).

```
FOR 1 = l:N,
   FORJ= l:N,
      A (I, J) = 1/(1+ J-1);
   END
END
```

FOR S = 1.0: -0.1: 0.0, END steps S with increments of -0.1
FOR E = EYE (N), ... END sets E to the unit N-vectors.

Long loops are more memory efficient when the colon expression appears in the FOR statement since the index vector is never created.

The BREAK statement can be used to terminate the loop prematurely.

IF

Conditionally execute statements.

The general form of the IF statement is

```
IF
  expression
  statements
ELSEIF
  expression
  statements
ELSE
  statement
  s
```

END

The statements are executed if the real part of the expression has all non-zero elements. The ELSE and ELSEIF parts are optional.
Zero or more ELSEIF parts can be used as well as nested IF's.
The expression is usually of the form expr rop expr where rop is =, <, >, <=, >=, or ~=.

```
Exampl
e if! = J
A(L,J) = 2; elseif
        abs(I-J) = 1
       A(L,J) = -1;
else
        A(I,J) = 0;
end
```

LENGTH

Length of vector

LENGTH(X) returns the length of vector X. It is equivalent to MAX (SIZE(X)) for non-empty arrays and 0 for empty ones.

SUM

Sum of elements

s = *SUM(X)* is the sum of the elements of the vector X If X is a matrix, S is a row vector

with the sum over each column. For N-D arrays, SUM(X) operates along the first non-singleton dimension. If X is floating point, that is double or single, S is accumulated natively, that is in the same class as X, and S has the same class as X If X is not floating point, S is accumulated in double and S has class double.

S = *SUM (X.DIM)* sums along the dimension DIM.

S = *SUM(X, 'double')* and S = *SUM(X, DIM, 'double')* accumulate S in double and S has class double, even if X is single.

S = *SUM(X,. 'native')* and S = *SUM(X, DIM, 'native')* accumulate S natively and S has the same class as X.

Examples:

If X = [0 1 2 3 4 5]

then sum(Xl) is [3 5 7] and sum(X2) is [312];

If X = int8 (1:20) then sum(X) accumulates in double and the result is double (210) while sum(X, 'native') accumulates in int8, but overflows and saturates to int8 (127).

PROD

Product of elements

For vectors, *PROD(X)* is the product of the elements of X. For matrices, PROD(X) is a row vector with the product over each column. For N-D arrays, PROD(X) operates on the first non-singleton dimension.

PROD(X,.DIM) works along the dimension DIM.

Example: If X = [0 1 2 3 4 5]
then prod(X, l) is [0 4 10] and prod(X,2) is [0 60]

CLEAR

Clear variables and functions from memory.

CLEAR removes all variables from the workspace.

CLEAR VARIABLES does the same thing.

CLEAR GLOBAL removes all global variables.

CLEAR FUNCTIONS removes all compiled M- and MEX-functions.
CLEAR ALL removes all variables, globals, functions and MEX
links.
CLEAR ALL at the command prompt also removes the Java packages import list.
CLEAR IMPORT removes the Java packages import list at the command prompt. Itcannot be used in a function.

CLEAR CLASSES is the same as CLEAR ALL except that class definitions are alsocleared If any objects exist outside the workspace (say in userdata or persistent in a locked m-file) a warning will be issued and the class definition will not be cleared

CLEAR CLASSES must be used if the number or names of fields in a class are changed.

CLEAR JAVA is the same as CLEAR ALL except that java classes on the dynamic javapath (defined using JAVACLASSP ATH) are also cleared.

CLEAR VARI VAR2 ... clears the variables specified. The wildcard character '*' can beused to clear variables that match a pattern. For instance, CLEAR X* clears all the variables in the current workspace that start with X

CLEAR -REGEXP PATI PAT2 can be used to match all patterns using regularexpressions. This option only clears variables. For more information on using regular expressions, type "doc regexp" at the command prompt

If X is global, CLEAR X removes X from the current workspace, but leaves it accessible to any functions declaring it global.

CLEAR GLOBAL X completely removes the global variable X.

CLEAR GLOBAL -REGEXP PAT removes global variables that match regular expressionpatterns.

Note that to clear specific global variables, the GLOBAL option must come first otherwise, all global variables will be cleared.

CLEAR FUN clears the function specified If FUN has been locked by MLOCK it will remain in memory. Use a partial path (see PARTIALPATH) to distinguish between different overloaded versions of FUN. For instance, 'clear inline/display' clears only the INLINE method for DISPLAY, leaving any other implementations in memory.

CLEAR ALL, CLEAR FUN, or CLEAR FUNCTIONS also have the side effect of removing debugging breakpoints and reinitializing persistent variables since the breakpoints for a function and persistent variables are cleared whenever the m-file changes or is cleared

Use the functional form of CLEAR, such as CLEAR ('name'), when the variable name or function name is stored in a string.

CLC

Clear command window

CLC clears the command window and homes the cursor.

GRID

Grid lines

GRID ON adds major grid lines to the current axes.

GRID OFF removes major and minor grid lines from the current axes.

GRID MINOR toggles the minor grid lines of the current axes.

GRID, by itself, toggles the major grid lines of the current axes.

GRID(AX,..) uses axes AX instead of the current axes.

GRID sets the XGrid, YGrid, and ZGrid properties of the current

axes.Set (AX, 'XMinorGrid', 'on') turns on the minor grid.

CLOSE

Close figure.

CLOSE(H) closes the window with handle H.

CLOSE, by itself, closes the current figure window.

CLOSE('name') closes t4e named window.

CLOSE ALL closes all the open figure windows.

CLOSE ALL HIDDEN closes hidden windows as well.

STATUS = CLOSE (...) returns 1 if the specified windows were closed and 0 otherwise.

OPERATION ON MATRICES

```
% Creating Matrix
>> a = [1,2,3;4,5,6;7,8,9]

  a =
     1     2   3
     4     5   6
     7     8   9

% Creating Matrix
>> b = [7,8,9;4,5,6;1,2,3]

  b =
     7     8   9
     4     5   6
     1     2   3

% Addition of two matrices
```

```
>> c = a+b

  c =
     8    10  12
     8    10  12
     8    10  12
```

% Multiplication of two matrices

```
>> d = a*b

d =
   18  24  30
   54  69  84
   90  114  138
```

% Multiplication of two matrices

```
>> e = b*a

e =
  102  126  150
   66  81  96
   30  36  42
```

% Sample by Sample Multiplication

```
>> f = a.*b

f =
    7      16   27
   16      25  36
    7      16   27
```

% Sample by Sample Multiplication

```
>> g = b.*a

g =
    7      16   27
   16      25  36
    7      16   27
```

% Sample by Sample

Division >> h = a./b

```
h =
   0.1429  0.2500  0.3333
   1.0000  1.0000  1.0000
   7.0000  4.0000  3.0000

% Transpose of a matrix

>> j = transpose (a)

  j =
      1    4   7
      2    5   8
      3    6   9
% Eigen Values of a

Matrix >> eig(a)

ans =

   16.1168
   -1.1168
   -0.0000
```

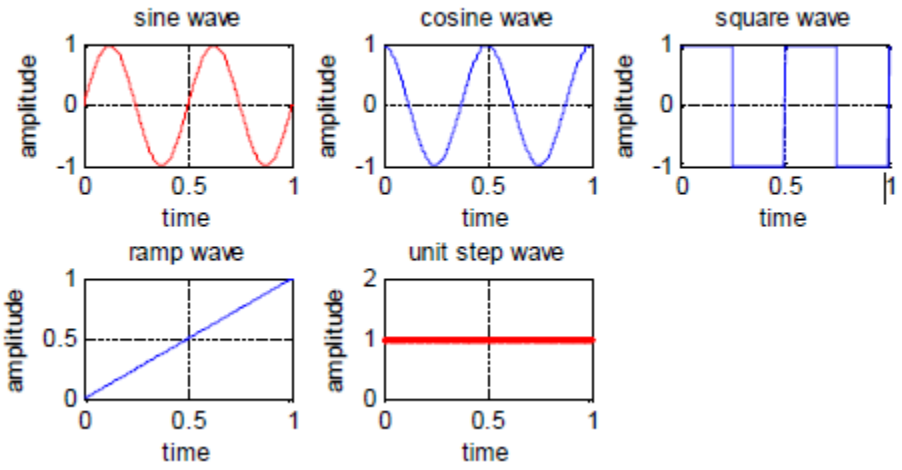
GENERATION OF BASIC SEQUENCES

%MATLAB program to common continuous time signals

```
clc;
clear all;
close all;
t=0:.001:1;
f=input('Enter the value of frequency');
a=input('Enter the value of amplitude');
subplot(3,3,1);
y=a*sin(2*pi*f*t);
plot(t,y,'r');
xlabel('time');
ylabel('amplitude');
title('sine wave')
grid on;
subplot(3,3,2);
z=a*cos(2*pi*f*t);
plot(t,z);
xlabel('time');
ylabel('amplitude');
```

```
title('cosine wave')
grid on;
subplot(3,3,3);
s=a*square(2*pi*f*t);
plot(t,s);
xlabel('time');
ylabel('amplitude');
title('square wave')
grid on;
subplot(3,3,4);
plot(t,t);
xlabel('time');
ylabel('amplitude');
title('ramp wave')
grid on;
subplot(3,3,5);
plot(t,a,'r');
xlabel('time');
ylabel('amplitude');
title('unit step wave')
grid on;
```

WAVEFORMS:

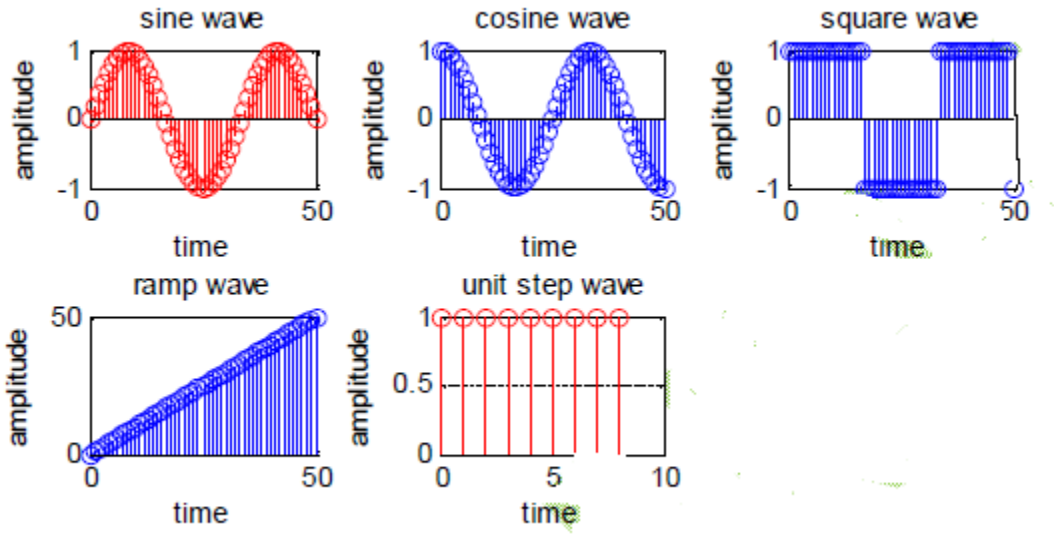


%MATLAB program to common discrete time signals

```
clc;
clear all;
close all;
```

```
n=0:1:50;
f=input('Enter the value of frequency');
a=input('Enter the value of amplitude');
N=input('Enter the length of unit step');
subplot(3,3,1);
y=a*sin(2*pi*f*n);
stem(n,y,'r');
xlabel('time');
ylabel('amplitude');
title('sine wave')
grid on;
subplot(3,3,2);
z=a*cos(2*pi*f*n);
stem(n,z);
xlabel('time');
ylabel('amplitude');
title('cosine wave')
grid on;
subplot(3,3,3);
s=a*square(2*pi*f*n);
stem(n,s);
xlabel('time');
ylabel('amplitude');
title('square wave')
grid on;
subplot(3,3,4);
stem(n,n);
xlabel('time');
ylabel('amplitude');
title('ramp wave')
grid on;
x=0:N-1;
d=ones(1,N);
subplot(3,3,5);
stem(x,d,'r');
xlabel('time');
ylabel('amplitude');
title('unit step wave')
grid on;
```

WAVEFORMS:

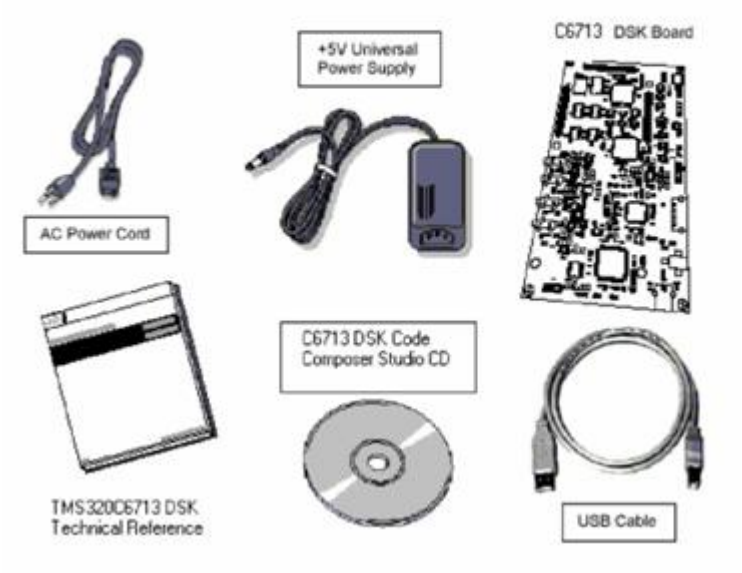

sine wave
amplitude
time
cosine wave
amplitude
time
square wave
amplitude
time
ramp wave
amplitude
time
unit step wave
amplitude
time

APPENDIX-2

INTRODUCTION TO TMS320C6713

TMS320C6713 DSK

Package Contents



DSK

The C6713™ DSK builds on TI's industry-leading line of low cost, easy-to-use DSP Starter Kit (DSK) development boards. The high-performance board features the TMS320C6713 floating-point DSP. Capable of performing 1350 million floating-point operations per second (MFLOPS), the C6713 DSP makes the C6713 DSK the most powerful DSK development board.

The DSK is USB port interfaced platform that allows to efficiently develop and test applications for the C6713. The DSK consists of a C6713-based printed circuit board that will serve as a hardware reference design for TI's customers' products. With extensive host PC and target DSP software support, including bundled TI tools, the DSK provides ease-of-use and capabilities that are attractive to DSP engineers.

The following checklist details items that are shipped with the C6711 DSK kit.

- TMS320C6713 DSK — TMS320C6713 DSK development board
- Other hardware — External 5VDC power supply
- CD-ROM — Code Composer Studio DSK tools

The C6713DSK has the following features:

The 6713 DSK is a low-cost standalone development platform that enables customers to evaluate and develop applications for the TI C67XX DSP family. The DSK also serves as a hardware reference design for the TMS320C6713 DSP. Schematics, logic equations and application notes are available to ease hardware development and reduce time to market.

The DSK uses the 32-bit EMIF for the SDRAM (CE0) and daughtercard expansion interface (CE2 and CE3). The Flash is attached to CE1 of the EMIF in 8-bit mode.

An on-board AIC23 codec allows the DSP to transmit and receive analog signals. McBSP0 is used for the codec control interface and McBSP1 is used for data. Analog audio I/O is done through four 3.5mm audio jacks that correspond to microphone input, line input, line output and headphone output. The codec can select the microphone or the line input as the active input. The analog output is driven to both the line out (fixed gain) and headphone (adjustable gain) connectors. McBSP1 can be re-routed to the expansion connectors in software.

A programmable logic device called a CPLD is used to implement glue logic that ties the board components together. The CPLD has a register based user interface that lets the user configure the board by reading and writing to the CPLD registers. The registers reside at the midpoint of CE1.

The DSK includes 4 LEDs and 4 DIP switches as a simple way to provide the user with interactive feedback. Both are accessed by reading and writing to the CPLD registers.

An included 5V external power supply is used to power the board. On-board voltage regulators provide the 1.26V DSP core voltage, 3.3V digital and 3.3V analog voltages. A voltage supervisor monitors the internally generated voltage, and will hold the board in reset until the supplies are within operating specifications and the reset button is released. If desired, JP1 and JP2 can be used as power test points for the core and I/O power supplies.

Code Composer communicates with the DSK through an embedded JTAG emulator with a USB host interface. The DSK can also be used with an external emulator through the external JTAG connector.

TMS320C6713 DSP Features

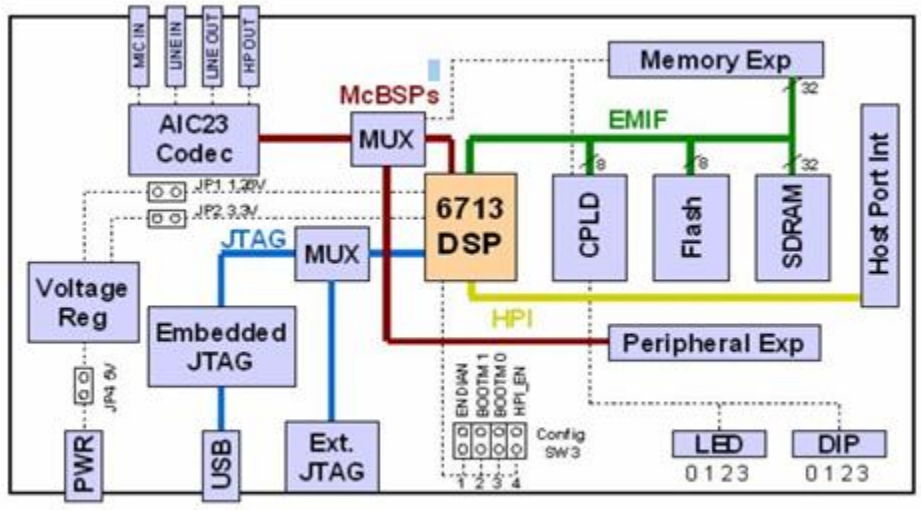
- Highest-Performance Floating-Point Digital Signal Processor (DSP):
  - Eight 32-Bit Instructions/Cycle
  - 32/64-Bit Data Word
  - 300-, **225-**, 200-MHz (GDP), and **225-**, 200-, 167-MHz (PYP) Clock Rates

- 3.3-, 4.4-, 5-, 6-Instruction Cycle Times
- 2400/1800, 1800/1350, 1600/1200, and 1336/1000 MIPS /MFLOPS
- Rich Peripheral Set, Optimized for Audio
- Highly Optimized C/C++ Compiler
- Extended Temperature Devices Available

- Advanced Very Long Instruction Word (VLIW) TMS320C67x™ DSP Core
  - Eight Independent Functional Units:
    - Two ALUs (Fixed-Point)
    - Four ALUs (Floating- and Fixed-Point)
    - Two Multipliers (Floating- and Fixed-Point)
  - Load-Store Architecture With 32 32-Bit General-Purpose Registers
  - Instruction Packing Reduces Code Size
  - All Instructions Conditional
- Instruction Set Features
  - Native Instructions for IEEE 754
    - Single- and Double-Precision
  - Byte-Addressable (8-, 16-, 32-Bit Data)
  - 8-Bit Overflow Protection
  - Saturation; Bit-Field Extract, Set, Clear; Bit-Counting; Normalization
- L1/L2 Memory Architecture
  - 4K-Byte L1P Program Cache (Direct-Mapped)
  - 4K-Byte L1D Data Cache (2-Way)
  - 256K-Byte L2 Memory Total: 64K-Byte L2 Unified Cache/Mapped RAM, and 192K-Byte Additional L2 Mapped RAM
- Device Configuration
  - Boot Mode: HPI, 8-, 16-, 32-Bit ROM Boot
  - Endianness: Little Endian, Big Endian
- 32-Bit External Memory Interface (EMIF)
  - Glueless Interface to SRAM, EPROM, Flash, SBSRAM, and SDRAM
  - 512M-Byte Total Addressable External Memory Space
- Enhanced Direct-Memory-Access (EDMA) Controller (16 Independent Channels)
- 16-Bit Host-Port Interface (HPI)
- Two Multichannel Audio Serial Ports (McASPs)
  - Two Independent Clock Zones Each (1 TX and 1 RX)
  - Eight Serial Data Pins Per Port:
    Individually Assignable to any of the Clock Zones
  - Each Clock Zone Includes:
    - Programmable Clock Generator
    - Programmable Frame Sync Generator
    - TDM Streams From 2-32 Time Slots
    - Support for Slot Size:
      8, 12, 16, 20, 24, 28, 32 Bits
    - Data Formatter for Bit Manipulation
  - Wide Variety of I2S and Similar Bit Stream Formats

- Integrated Digital Audio Interface Transmitter (DIT) Supports:
  - S/PDIF, IEC60958-1, AES-3, CP-430 Formats
  - Up to 16 transmit pins
  - Enhanced Channel Status/User Data
- Extensive Error Checking and Recovery

- Two Inter-Integrated Circuit Bus (I^2C Bus™) Multi-Master and Slave Interfaces
- Two Multichannel Buffered Serial Ports:
  - Serial-Peripheral-Interface (SPI)
  - High-Speed TDM Interface
  - AC97 Interface
- Two 32-Bit General-Purpose Timers
- Dedicated GPIO Module With 16 pins (External Interrupt Capable)
- Flexible Phase-Locked-Loop (PLL) Based Clock Generator Module
- IEEE-1149.1 (JTAG †) Boundary-Scan-Compatible
- Package Options:
  - 208-Pin PowerPAD™ Plastic (Low-Profile) Quad Flatpack (PYP)
  - 272-BGA Packages (GDP and ZDP)
- 0.13-µm/6-Level Copper Metal Process
  - CMOS Technology
- 3.3-V I/Os, 1.2 ‡ -V Internal (GDP & PYP)
- 3.3-V I/Os, 1.4-V Internal (GDP)(300 MHz only)



TMS320C6713 DSK Overview Block Diagram

INTRODUCTION TO CODE COMPOSER STUDIO

Code Composer is the DSP industry's first fully integrated development environmen (IDE) with DSP-specific functionality. With a familiar environment liked MS-based C++TM Code Composer lets you edit, build, debug, profile and manage projects from a singl unified environment. Other unique features include graphical signal analysis injection/extraction of data signals via file I/O, multi-processor debugging, automate testing and customization via a C-interpretive scripting language and much more.

CODE COMPOSER FEATURES INCLUDE:

- IDE
- Debug IDE
- Advanced watch windows
- Integrated editor
- File I/O, Probe Points, and graphical algorithm scope probes
- Advanced graphical signal analysis
- Interactive profiling
- Automated testing and customization via scripting
- Visual project management system
- Compile in the background while editing and debugging
- Multi-processor debugging
- Help on the target DSP

Note :

Documents for Reference:

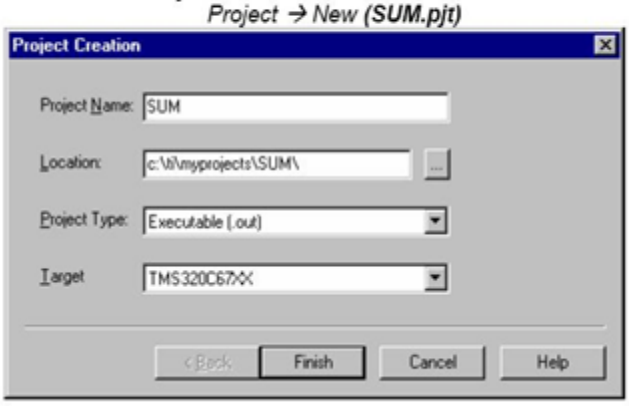
spru509 → Code Composer Studio getting started guide.
spru189 → TMS320C6000 CPU & Instruction set guide
spru190 → TMS320C6000 Peripherals guide
slws106d → codec(TLV320AIC23) Data Manual.
spru402 → Programmer's Reference Guide.
sprs186j → TMS320C6713 DSP

Soft Copy of Documents are available at : c:\ti\docs\pdf.

Procedure to work on Code Composer Studio

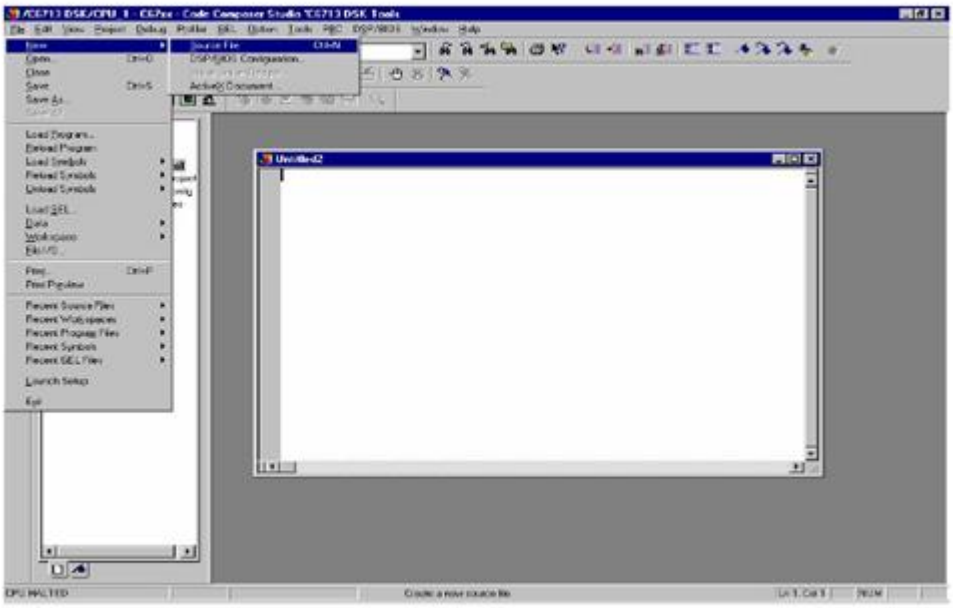
1. To create a New Project

*Project → New (**SUM.pjt**)*



2. To Create a Source file

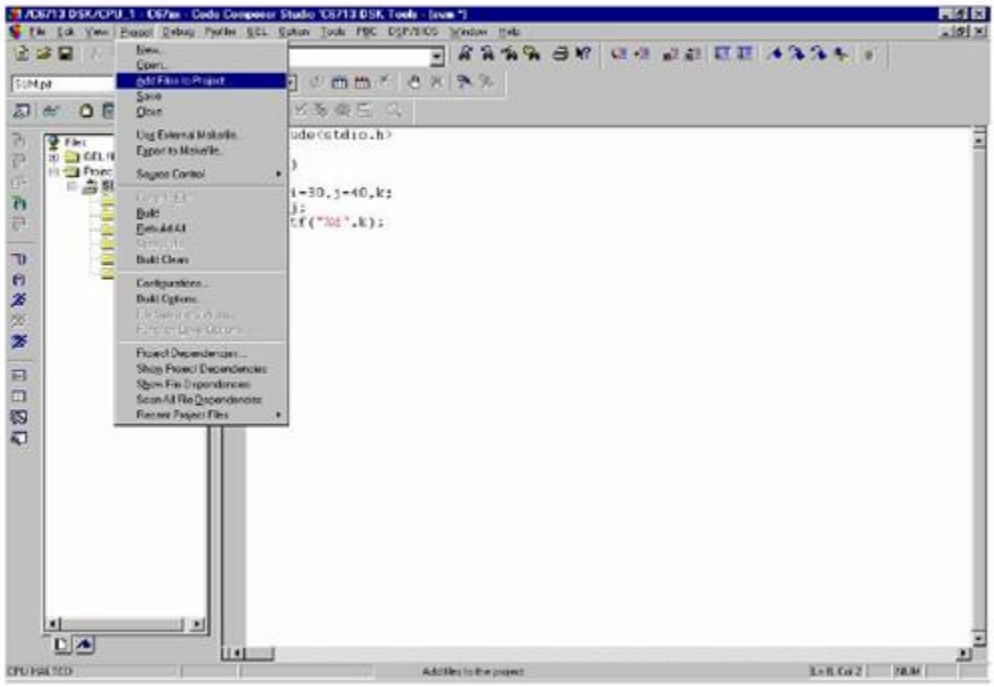
File → New

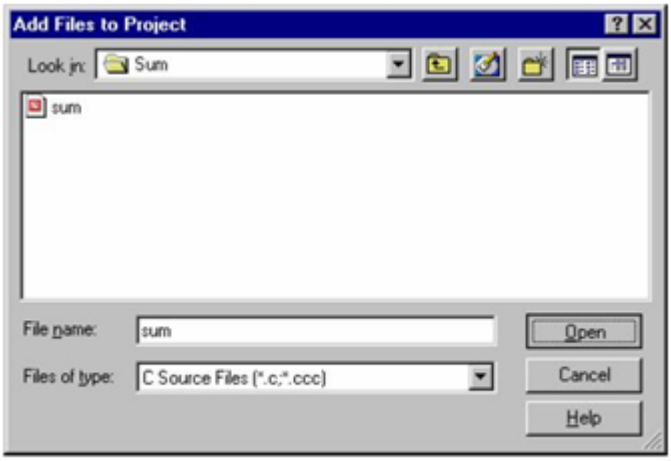


*Type the code (Save & give a name to file, Eg: **sum.c**).*

3. To Add Source files to Project

Project → Add files to Project → sum.c

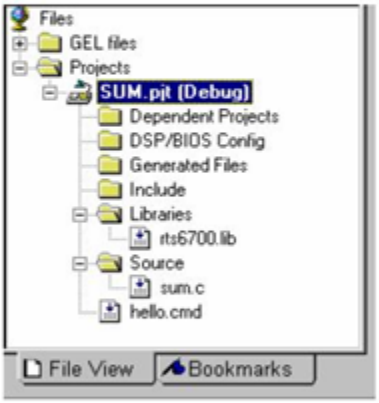




4. To Add rts6700.lib file & hello.cmd:

Project → Add files to Project →rts6700.lib
Path: c:\CCStudio\c6000\cgtools\lib\rts6700.lib
Note: Select Object & Library in(*.o,*.l) in Type of files

Project → Add files to Project →hello.cmd
Path: c:\ti\tutorial\dsk6713\hello1\hello.cmd
Note: Select Linker Command file(*.cmd) in Type of files



5. To Compile:
Project → Compile File

6. To build or Link:
Project → build,
Which will create the final executable ***(.out)*** *file.(Eg. sum.out).*

7. Procedure to Load and Run program:
Load program to DSK:
File → Load program → sum. out

8. To execute project:
Debug → Run.

APPENDIX

LABORATORY COURSE ASSESSMENT GUIDELINES

i. The number of experiments/programs/sessions in each laboratory course shall be as per the curriculum in the scheme of instructions provided by OU.

ii. The students will maintain a separate note book for each laboratory course in which all the related work would be done.

iii. In each session the students will complete the assigned tasks of process development, coding, compiling, debugging, linking and executing the programs.

iv. The students will then execute the programme and validate it by obtaining the correct output for the provided input. The course coordinator will certify the validation in the same session.

v. The students will submit the record in the next class. The evaluation will be continuous and not cycle-wise or at semester end.

vi. The internal marks of 25 are awarded in the following manner:
   a. Laboratory record - Maximum Marks 15
   b. Test and Viva Voce - Maximum Marks 10

vii. Laboratory Record: Each experimental record is evaluated for a score of 50. **The rubric parameters are as follows:**
   a. Write up format - Maximum Score 20
   b. Process development and coding - Maximum Score 10
   c. Compile, debug, link and execute program - Maximum Score 15
   d. Process validation through input-output - Maximum Score 5

   While (a) is assessed at the time of record submission, (b), (c) and (d) are assessed during the session based on the performance of the student in the laboratory session. Hence if a student is absent for any laboratory session but completes the program in another session and subsequently submits the record, it shall be evaluated for a score of 20 and not 50.

viii. The experiment evaluation rubric is therefore as follows :

Parameter	Max Score	Outstanding	Accomplished	Developing	Beginner	Points
Process Development and Coding	10					
Compilation, Debugging, Linking and Executing	15					
Process Validation	5					
Write up format	20					

LABORATORY EXPERIMENT EVALUATION RUBRIC

CATEGORY (% of Marks)	OUTSTANDING (75-100%)	ACCOMPLISHED (51-75%)	DEVELOPING (26-50%)	BEGINNER (Upto25%)
Enthusiasm	Facial expressions and body language generate a strong interest and enthusiasm about the topic in others.	Facial expressions and body language sometimes generate a strong interest and enthusiasm about the topic in others.	Facial expressions and body language are used to try to generate enthusiasm, but seem somewhat faked.	Very little use of facial expressions or body language. Did not generate much interest in topic being presented.
Posture	Stands up straight, looks relaxed and confident and looks always towards the audience during presentation	Stands up straight and looks towards the audience during presentation and some times looks down/reads slides.	Posture not straight and some times looks towards the audience but looks mostly at reads slides or downwards	Slouches and/or does not look towards the audience and mostly reads slides or looks down during presentation
Speaks Clearly and distinctly	Understand and hear the speaker all the time	Understand and hear the speaker most of the time	Understand and hear the speaker only some times but speaks too softly most of the time	Cannot hear or understand the speaker at all as he/she is speaking too softly
Speaker's slides	Slides as well as text is formatted with no spelling errors. Slides not cluttered with text. There are diagrams	Slides are well as text is formatted with some spelling errors and some slides have too much text . There are few diagrams	Some slides are not formatted as well as text is not formatted and have spelling errors. Most slides have too much text with very few diagrams	Each slide looks different with different font sizes and have too much text in all slides. There are no diagrams at all.
Speaker's Communication about the topic	The student has explained the topic at the level of the audience very well	The student has explained the topic well at the level of audience but not completely and requires to read more	The students has partly explained the topic at the level of the audience but really needs to read at lot	The students does not seem to have properly communicated about the topic at all.
Technical Content	The student has used relevant technical information exhaustively and connected it with the presentation topic	The student has used relevant technical information exhaustedly but failed to connect it with the presentation topic	The student has used relevant technical information sparingly and failed to connect it with the presentation topic	The presentation is weak in technical content with little or no explicit connection with the topic of presentation
Speaker's ability to answer questions	Student is able to accurately answer almost all questions posed by audience about the topic.	Student is able to accurately answer most questions posed by audience about the topic.	Student is able to accurately answer only a few questions posed by audience about the topic.	Student is unable to accurately answer any questions posed by audience about the topic.

ix. The first page of the record will contain the following title sheet:

NAME: **ROLL NO.**

Exp. No.	Title of the Program	Date conducted	Date Submitted	Process Development and Coding (Max 10)	Compilation , Debugging, Linking and Executing (Max 15)	Process Validation (Max 5)	Write up format (Max 20)	Total Score (Max 50)
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
	TOTAL							

Date: **Signature of Course Coordinator**

x. The 15 marks of laboratory record will be scaled down from the TOTAL of the assessment sheet.

xi. The test and viva voce will be scored for 10 marks as follows:

Internal Test	-	6 marks
Viva Voce / Quiz	-	4 marks

xii. Each laboratory course shall have 5 course outcomes.

The proposed course outcomes would be as follows:

On successful completion of the course, the student will acquire the ability to:

1. Apply the design concepts for development of a process and interpret data
2. Demonstrate knowledge of programming environment, compiling, debugging, linking and executing variety of programs.
3. Demonstrate documentation and presentation of the algorithms / flowcharts / programs in a record form.
4. Validate the process using known input-output parameters.
5. Employ analytical and logical skills to solve real world problem and demonstrate oral communication skills.

xiii. The Course coordinators would prepare the assessment matrix in accordance with the guidelines provided above for the five course outcomes. The scores to be entered against each of the course outcome would be the sum of the following as obtained from the assessment sheet in the record:

a. Course Outcome 1: Sum of the scores under 'Process Development and Coding'.
b. Course Outcome 2: Sum of the scores under 'Compilation/Debugging/Linking and Executing'.
c. Course Outcome 3: Sum of the scores under 'Write up format'.
d. Course Outcome 4: Sum of the scores under 'Process validation'.
e. Course Outcome 5: Marks for 'Internal Test and Viva voce'.

xiv. Soft copy of the assessment matrix would be provided to the course coordinators.

xv. There may be some laboratory courses based on proprietary software like MATLAB, AUTOCAD etc. for which the course coordinators and programme coordinators would formulate appropriate course outcomes.

MUFFAKHAM JAH COLLEGE OF ENGINEERING AND TECHNOLOGY

Program Outcomes of B.E (ECE) Program:

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO 12: Life-long learning: Recognise the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs) of ECE Department, MJCET

PSO1: The ECE Graduates will acquire state of art analysis and design skills in the areas of digital and analog VLSI Design using modern CAD tools.

PSO2: The ECE Graduates will develop preliminary skills and capabilities necessary for embedded system design and demonstrate understanding of its societal impact.

PSO3: The ECE Graduates will obtain the knowledge of the working principles of modern communication systems and be able to develop simulation models of components of a communication system.

PSO4: The ECE Graduates will develop soft skills, aptitude and programming skills to be employable in IT sector.