

**SRI SHAKTHI**  
**INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**Vision of the Institute**

To make the institution one of our nation's great engineering schools recognized nationally and internationally for excellence in teaching, research and public service. We seek to be the preferred destination for students, practitioners seeking an engineering education, employers hiring engineering graduates and organizations seeking engineering knowledge.

**Mission of the Institute**

To provide an encouraging environment to develop the intellectual capacity, critical thinking, creativity and problem solving ability of the students.

**Vision of the ECE Department**

To be recognised as centre of excellence in higher learning and research in the fields of Electronics and Communication Engineering with national and international repute.

**Mission of the ECE Department**

To achieve the vision, the department will

- M1:** Provide a supportive learning experience to students in the field of Electronics and Communication Engineering by emphasizing activity-based Learning with Research focus to prepare them for professional careers.
- M2:** Enable students to solve the Engineering problems with their creativity to cater the societal needs keeping in pace with the technological advancements.
- M3:** To provide Students with ethical and human values to thereby promote social activities

## **Program Educational Objectives of the ECE department**

**PEO 1:** Graduates will have profound knowledge on software skills, core-engineering concepts to analyze and design electronics and communication products and develop solutions for the real life applications.

**PEO 2:** Graduates will be able to solve problems in electronics and communication through creativity, critical thinking, intellectual capacity and social responsibilities

**PEO 3:** Graduates will be equipped with professional and ethical attitude, teamwork skills, leadership, and work on multidisciplinary environment..

**PEO4:** Graduates will procure successful professional career & life-long learning with an impressive academic environment.

**SRI SHAKTHI**  
**INSTITUTE OF ENGINEERING AND TECHNOLOGY**  
COIMBATORE 641062  
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING



**DIGITAL SIGNAL  
PROCESSING LABORATORY**

**Student Name :**

**Roll No :**

**Class :**

**Semester No :**

**Subject Name : DIGITAL SIGNAL PROCESSING LABORATORY**

**Regulation : 2021**

**Academic Year : 2024-2025(ODD)**



# **SRI SHAKTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY**



## **DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING DIGITAL SIGNAL PROCESSING LABORATORY**

Name: \_\_\_\_\_ Roll No : \_\_\_\_\_

Class: \_\_\_\_\_ Semester: \_\_\_\_\_

This is a certified work done by \_\_\_\_\_

Place:

Date:

STAFF IN-CHARGE

H.O.D

University Register number: \_\_\_\_\_

Submitted for the university practical exam held on \_\_\_\_\_

INTERNAL EXAMINER

EXTERNAL EXAMINER

| EX.NO | EXPERIMENT NAME  |
|-------|--|
| 01.   | Generation Of Unit Step Signals                                |
| 02.   | Generation Of Ramp Signals                                     |
| 03.   | Generation Of Exponential Signals                              |
| 04.   | Generation Of Sine Signals                                     |
| 05.   | Generation Of Cosine Signals                                   |
| 06.   | Generation Of Impulse Signals                                  |
| 07.   | Verification Of Sampling Theorem                               |
| 08.   | Perform Addition Operation Of Two Signals                      |
| 09.   | Fft And Ifft Spectrum  |
| 10.   | Decimation In Time-Fft And Ifft                                |
| 11.   | Frequency Response Of Lti System                               |
| 12.   | Linear Convolution Using Fft                                   |
| 13.   | Linear Convolution Using Matrix Multiplication                 |
| 14.   | Circular Convolution   |
| 15.   | Design Of Iir Filters ( Butterworth )                          |
| 16.   | Design Of Iir Filters ( Chebychev )                            |
| 17.   | Fir Implementation Using Windows                               |
| 18.   | Time And Frequency Response Of A Signal                        |
| 19.   | Remove Noise From A Signal                                     |
| 20.   | Design A Bandpass And Bandstop Finite Impulse Response Filter  |
| 21.   | Design Fir Filter Using Rectangular Window                     |
| 22.   | Design Fir Filter Using Hamming Window                         |
| 23.   | Design Lowpass Fir Filter Using Rectangular                    |
| 24.   | Design A Lowpass And Highpass Infinite Impulse Response Filter |
| 25.   | Equivalent Impulse Response Of The Narrowband Filter           |

|     |  |
|-----|--|
| 26. | Perform Cross-Correlation  |
| 27. | Perform Quantization Using Matlab  |
| 28. | Design of Highpass Fir Filter Using Rectangular Window Using Matlab                                      |
| 29. | Verify Channel Capacity Theorem Using Matlab   |
| 30. | Perform Elliptic filter response Using Matlab Program  |
| 31. | Random Number Generation   |
| 32. | Generate The Impulse Sequence At N0 Samples Lying Between N1 And N2                                      |
| 33. | To Illustrate The Simple Mathematical Expressions In Matlab I) $Z=X/Y$<br>II) $X=3*\sqrt{5}-1$ , $Y=X^2$ |
| 34. | Plot The Basic Signals-Impulse, Step Function And Ramp Function To Create 2-D And 3-D Plots              |
| 35. | Matlab Program To Perform Dft By Fft   |
| 36. | Generate Triangular Wave Using Matlab  |
| 37. | Perform Block Convolution Using Matlab   |
| 38. | Code For Folding A Sequence And Plot It  |
| 39. | Perform Time Scaling Operation On Signals Using Matlab   |
| 40. | Area Of Circle Using Matlab  |
| 41. | Stability Test For The Transfer Function   |
| 42. | Design Highpass Fir Filter Using Rectangular Window Using Matlab   |
| 43. | Matlab Code For Waveform Addition  |
| 44. | Perform Auto correlation   |
| 45. | Perform Difference equation function   |
| 46. | Circular convolution using summation formula   |
| 47. | Design and implement FIR filter using frequency sampling method  |
| 48. | Convert CD Data To DVD Data  |
| 49. | Implementation of interpolation process  |
| 50. | Implementation of sampling rate converters   |

EX.NO :1

## GENERATION OF UNIT STEP SIGNALS

DATE :

### AIM:

To generate unit step signals using MATLAB.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Declare the timescale and amplitude of the signal.
- Step-3: Using pre-defined functions generate the corresponding waveforms and plot them.
- Step-4: Label the axes and title.
- Step-5: Execute the program and view the output waveforms.

### CODING:

```
clc;          % Clear command window
clear;        % Clear workspace
close all;    % Close all figure windows

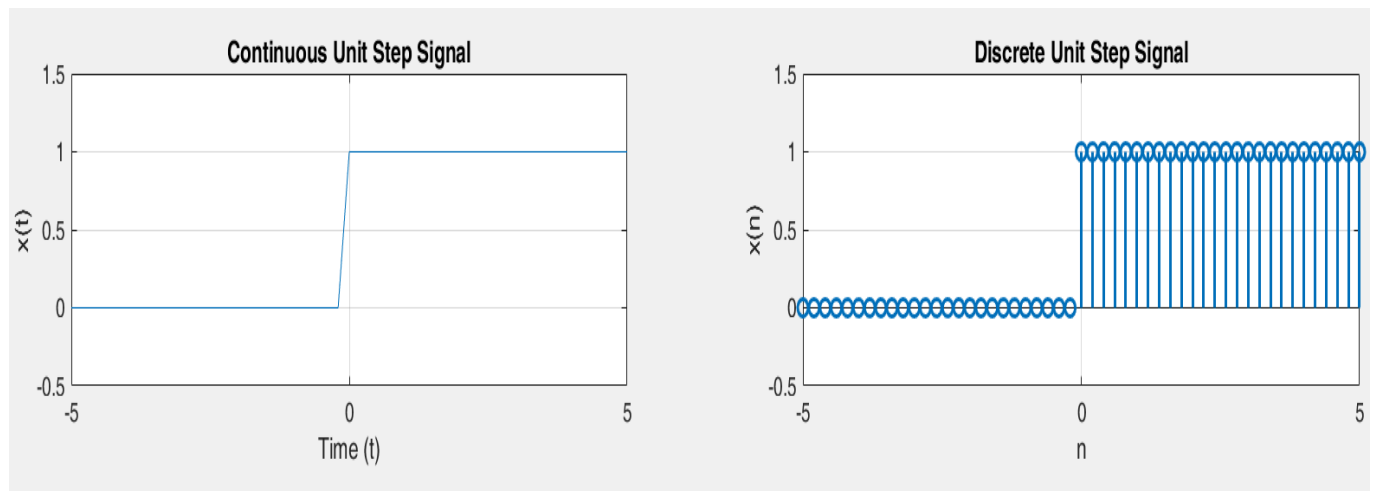
% Define time parameters
t = -5:0.2:5;

% Generate unit step signal
y1 = t >= 0;

% Plot continuous unit step signal
subplot(3,2,3);
plot(t, y1);
title('Continuous Unit Step Signal');
xlabel('Time (t)');
ylabel('x(t)');
grid on;
axis([-5 5 -0.5 1.5]);

% Plot discrete unit step signal
subplot(3,2,4); stem(t, y1, 'LineWidth', 1.5);
title('Discrete Unit Step Signal');
xlabel('n');
ylabel('x(n)');
grid on;
axis([-5 5 -0.5 1.5]);
        stem(t, y1, 'LineWidth', 1.5);
title('Discrete Unit Step Signal');
xlabel('n');
ylabel('x(n)');
grid on;
axis([-5 5 -0.5 1.5]);
```

## OUTPUT:



## RESULT:

Thus the continue unit step signals and discrete unit step signals are generated using MATLAB



EX.NO :2

## GENERATION OF RAMP SIGNALS

DATE :

### AIM:

To generate ramp signals using MATLAB:

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Declare the timescale and amplitude of the signal.
- Step-3: Using pre-defined functions generate the corresponding waveforms and plot them.
- Step-4: Label the axes and title.
- Step-5: Execute the program and view the output waveforms.

### CODING:

```
clc;          % Clear command window
clear;        % Clear workspace
close all;    % Close all figure windows

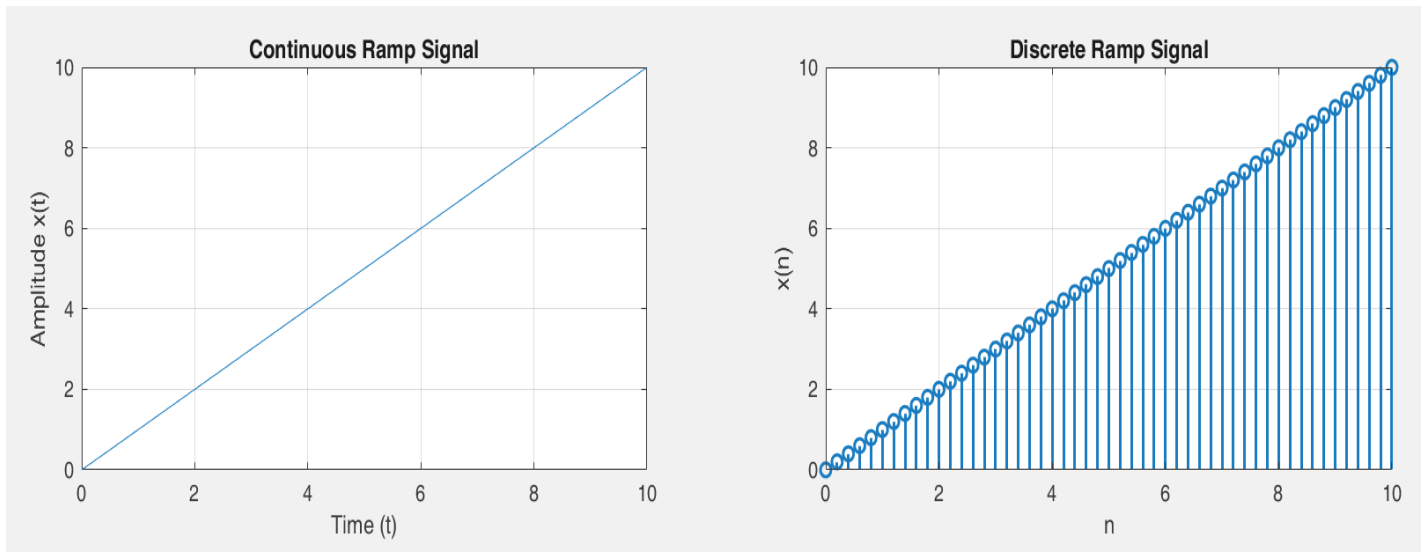
% Define parameters
t = 0:0.2:10; % Time scale

% Generate ramp signal (y = t for ramp signal)
ramp = t;

% Plot continuous ramp signal
subplot(2,2,1);
plot(t, ramp);
title('Continuous Ramp Signal');
xlabel('Time (t)');
ylabel('Amplitude x(t)');
grid on;

% Plot discrete ramp signal
subplot(2,2,2);
stem(t, ramp, 'LineWidth', 1.5);
title('Discrete Ramp Signal');
xlabel('n');
ylabel('x(n)');
grid on;
```

## OUTPUT:



## RESULT:

Thus the continue ramp signals and discrete ramp signals are generated using MATLAB

EX.NO :3

## GENERATION OF EXPONENTIAL SIGNALS

DATE :

### AIM:

To generate exponential signals using MATLAB:

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Declare the timescale and amplitude of the signal.
- Step-3: Using pre-defined functions generate the corresponding waveforms and plot them.
- Step-4: Label the axes and title.
- Step-5: Execute the program and view the output waveforms.

### CODING:

```
clc;          % Clear command window
clear;        % Clear workspace
close all;    % Close all figure windows

% Define parameters
t = -2:0.1:2;
n = -5:1:5;
alpha = 1;

% Generate continuous exponential signal
continuous_exp = exp(alpha * t);

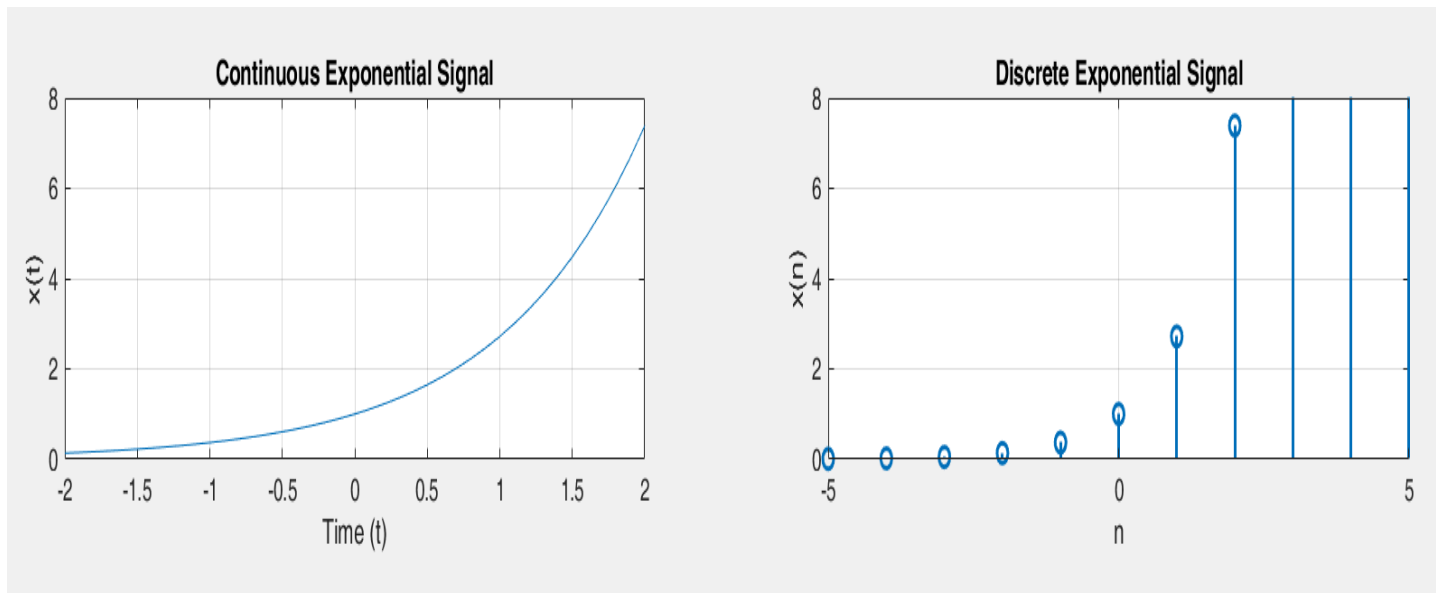
% Generate discrete exponential signal
discrete_exp = exp(alpha * n);

% Plot continuous exponential signal
subplot(3,2,3);
plot(t, continuous_exp);
title('Continuous Exponential Signal');
xlabel('Time (t)');
ylabel('x(t)');
grid on;
axis([-2 2 0 8]);

% Plot discrete exponential signal
subplot(3,2,4);
stem(n, discrete_exp, 'LineWidth', 1.5);
title('Discrete Exponential Signal');
```

```
xlabel('n');  
ylabel('x(n)');  
grid on;  
axis([-5 5 0 8]);
```

### OUTPUT:



### RESULT:

Thus the continue exponential signals and discrete exponential signals are generated using MATLAB

EX.NO :4

## GENERATION OF SINE SIGNALS

DATE :

### AIM:

To generate sine signals using MATLAB.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Declare the timescale and amplitude of the signal.
- Step-3: Using pre-defined functions generate the corresponding waveforms and plot them.
- Step-4: Label the axes and title.
- Step-5: Execute the program and view the output waveforms.

### CODING:

```
clc;          % Clear command window
clear;        % Clear workspace
close all;    % Close all figure windows
```

```
% Define parameters
```

```
t = 0:0.2:10; % Time scale
```

```
f = 0.5;      % Frequency (Hz)
```

```
a = 10;       % Amplitude
```

```
% Generate sine wave
```

```
s = a * sin(2 * pi * f * t);
```

```
% Plot continuous sine wave
```

```
subplot(2,2,1);
```

```
plot(t, s);
```

```
title('Continuous Sine Wave');
```

```
xlabel('Time (t)');
```

```
ylabel('Amplitude x(t)');
```

```
grid on;
```

```
% Plot discrete sine wave
```

```
subplot(2,2,2);
```

```
stem(t, s, 'LineWidth', 1.5);
```

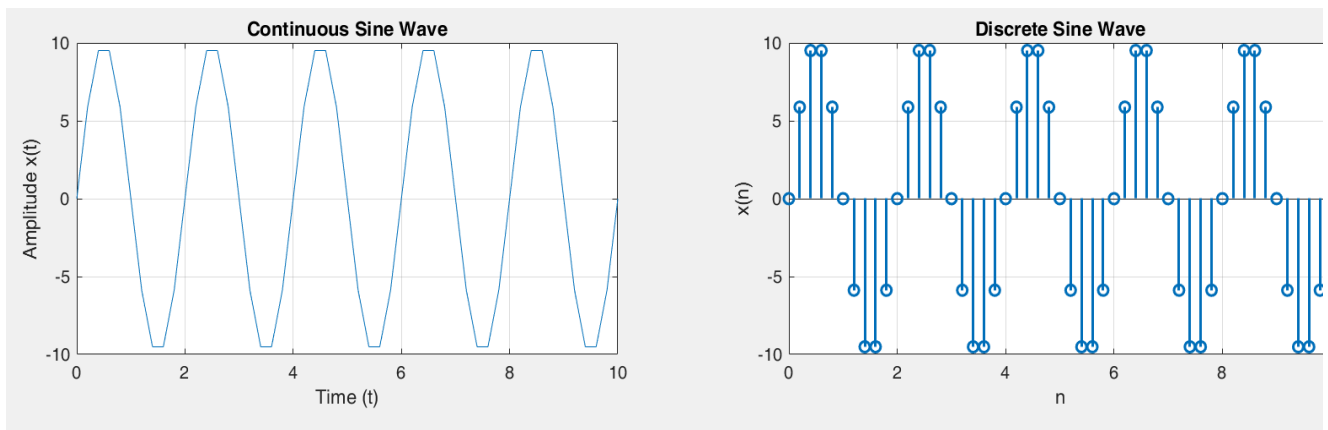
```
title('Discrete Sine Wave');
```

```
xlabel('n');
```

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

```
grid on;
```

## OUTPUT:



## RESULT:

Thus the continue sine signals and discrete sine signals are generated using MATLAB

EX.NO :4

## GENERATION OF COSINE SIGNALS

DATE :

### AIM:

To generate cosine signals using MATLAB.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Declare the timescale and amplitude of the signal.
- Step-3: Using pre-defined functions generate the corresponding waveforms and plot them.
- Step-4: Label the axes and title.
- Step-5: Execute the program and view the output waveforms.

### CODING:

```
clc;          % Clear command window
clear;        % Clear workspace
close all;    % Close all figure windows
```

```
% Define parameters
t = 0:0.2:10; % Time scale
f = 0.5;      % Frequency (Hz)
a = 10;       % Amplitude
```

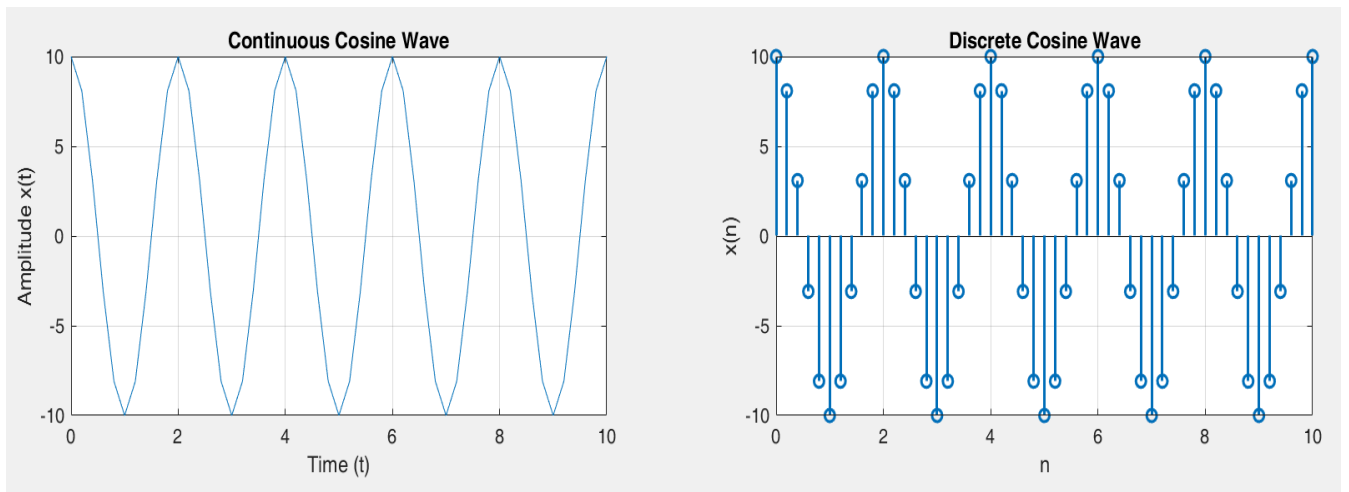
```
% Generate cosine wave
c = a * cos(2 * pi * f * t);
```

```
% Plot continuous cosine wave
subplot(2,2,1);
plot(t, c);
title('Continuous Cosine Wave');
xlabel('Time (t)');
ylabel('Amplitude x(t)');
grid on;
```

```
% Plot discrete cosine wave
subplot(2,2,2);
stem(t, c, 'LineWidth', 1.5);
title('Discrete Cosine Wave');
xlabel('n');
ylabel('x(n)');
```

grid on;

**OUTPUT:**



**RESULT:**

Thus the continue cosine signals and discrete cosine signals are generated using MATLAB



EX.NO :5

## GENERATION OF IMPULSE SIGNALS

DATE :

### AIM:

To generate impulse signals using MATLAB:

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Declare the timescale and amplitude of the signal.
- Step-3: Using pre-defined functions generate the corresponding waveforms and plot them.
- Step-4: Label the axes and title.
- Step-5: Execute the program and view the output waveforms.

### CODING:

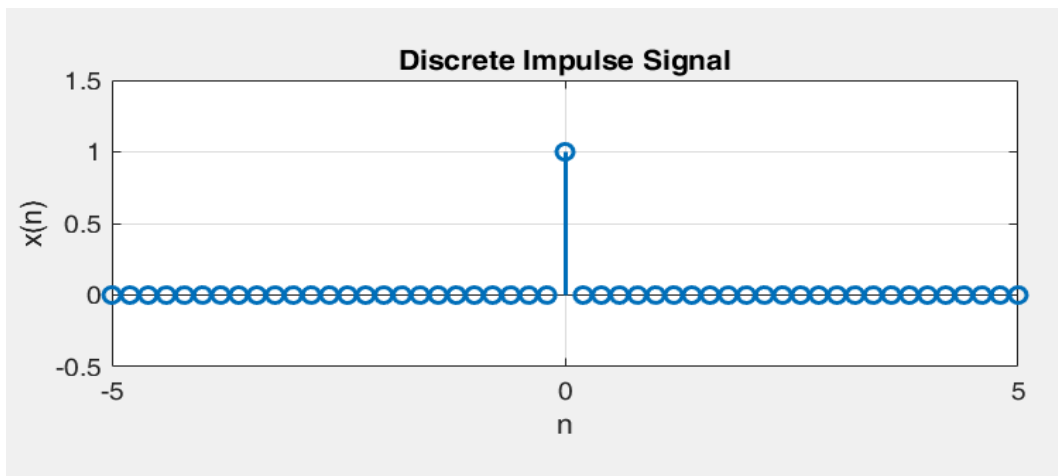
```
clc;          % Clear command window
clear;        % Clear workspace
close all;    % Close all figure windows
```

```
% Define time parameters
t = -5:0.2:5;
```

```
% Generate impulse signal
impulse = (t == 0);
```

```
% Plot Discrete impulse signal
subplot(3,2,3);
stem(t, impulse, 'LineWidth', 1.5);
title('Discrete Impulse Signal');
xlabel('n');
ylabel('x(n)');
grid on;
axis([-5 5 -0.5 1.5]);
```

### OUTPUT:



### RESULT:

Thus the discrete impulse signals are generated using MATLAB

EX.NO :6

## VERIFICATION OF SAMPLING THEOREM

DATE :

### AIM:

To verify the sampling theorem by sampling the given signal and reconstruct the signal and compare them.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace. Step-2: Plot the continuous sine wave.
- Step-3: With respect to the Nyquist criterion for sampling, the sampling frequency must be greater than the original frequency for faithful reconstruction.
- Step-4: Use the conditions:  
fs1<2fm for under-sampling fs2=2fm for critical sampling fs3>2fm for over-sampling.
- Step-5: Plot the sampled signal for the three conditions of sampling. Step-6: Compare the output waveforms to verify the sampling theorem.

### CODING:

```
clc;          % Clear command window
clear;        % Clear workspace
close all;    % Close all figure windows
```

```
% Define time parameters
t = -10:1:10;
fm = input('Enter the frequency: ');
x = sin(2 * pi * fm * t);
```

```
% Sampling frequencies
fs1 = 1.65 * fm;
fs2 = 2 * fm;
fs3 = 16 * fm;
n1 = -10:0.5:10;
```

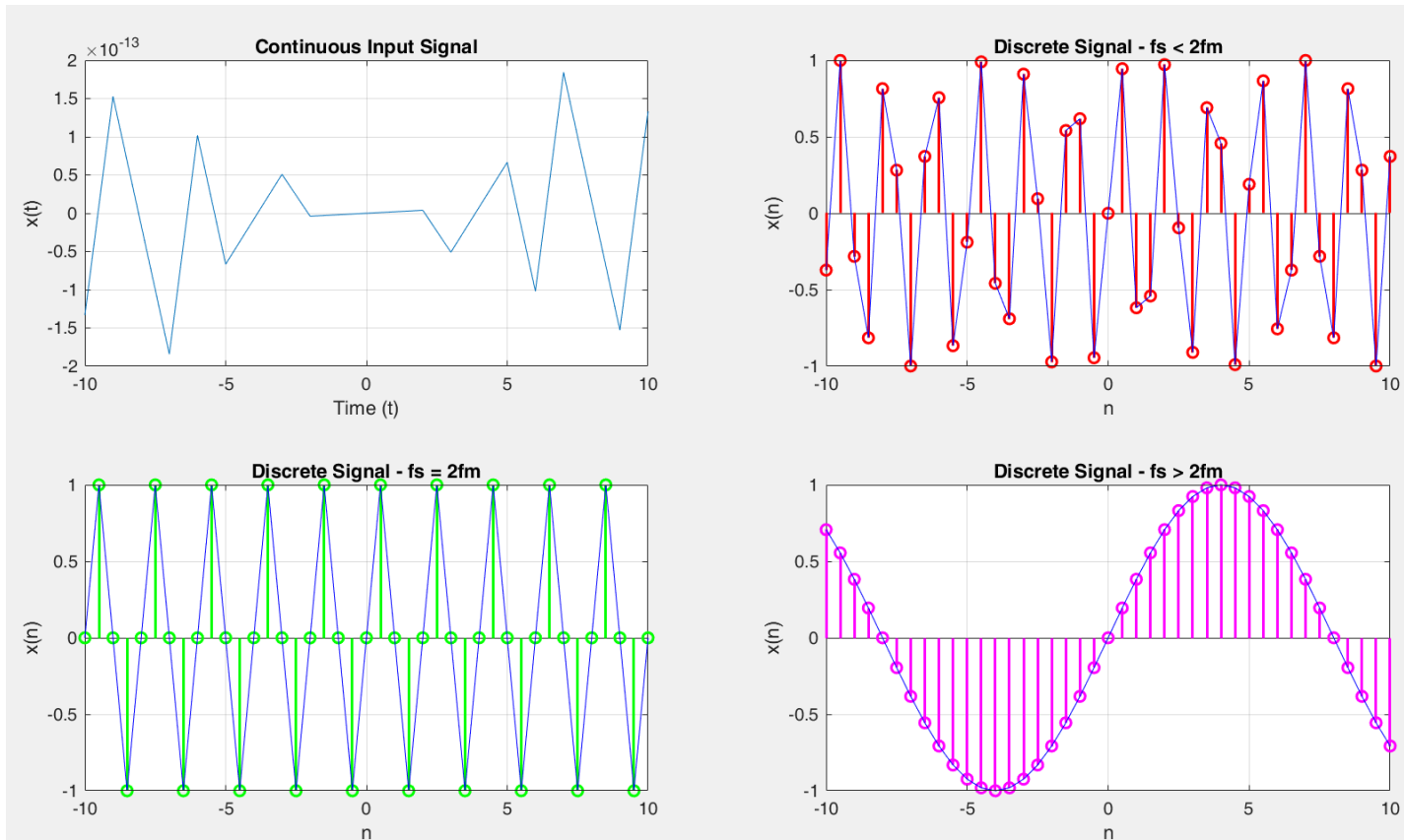
```
subplot(2, 2, 1);
plot(t, x);
title('Continuous Input Signal');
xlabel('Time (t)');
ylabel('x(t)');
grid on;
```

```
% Under-Sampling
xn1 = sin(2 * pi * n1 * fm / fs1);
subplot(2, 2, 2);
stem(n1, xn1, 'r', 'LineWidth', 1.5);
hold on;
plot(n1, xn1, 'b');
title('Discrete Signal -  $f_s < 2f_m$ ');
xlabel('n');
ylabel('x(n)');
grid on;
```

```
% Critical Sampling
xn2 = sin(2 * pi * n1 * fm / fs2);
subplot(2, 2, 3);
stem(n1, xn2, 'g', 'LineWidth', 1.5);
hold on;
plot(n1, xn2, 'b');
title('Discrete Signal -  $f_s = 2f_m$ ');
xlabel('n');
ylabel('x(n)');
grid on;
```

```
% Over-Sampling
xn3 = sin(2 * pi * n1 * fm / fs3);
subplot(2, 2, 4);
stem(n1, xn3, 'm', 'LineWidth', 1.5);
hold on;
plot(n1, xn3, 'b');
title('Discrete Signal -  $f_s > 2f_m$ ');
xlabel('n');
ylabel('x(n)');
grid on;
```

## OUTPUT:



## RESULT:

Thus sampling theorem has been verified using MATLAB

EX.NO :7

## PERFORM ADDITION OPERATION OF TWO SIGNALS

DATE :

### AIM:

To Perform addition operation of two signals using MATLAB.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Define the time vector for the signals.
- Step-3: Generate the first signal using a defined frequency and time vector.
- Step-4: Generate the second signal .
- Step-5: Add the two signals together to form a resultant signal.
- Step-6: Create subplots to visualize the individual signals and the resultant signal.
- Step-7: Label the axes and title for each subplot.
- Step-8: Display the grid for better visualization.

### CODING:

```
clc;          % Clear command window
clear;        % Clear workspace
close all;    % Close all figure windows

% Define time parameters
t = 0:0.01:2*pi;

% Generate the two signals
f1 = 1;
f2 = 2;
signal1 = sin(f1 * t);
signal2 = 0.5 * sin(f2 * t);

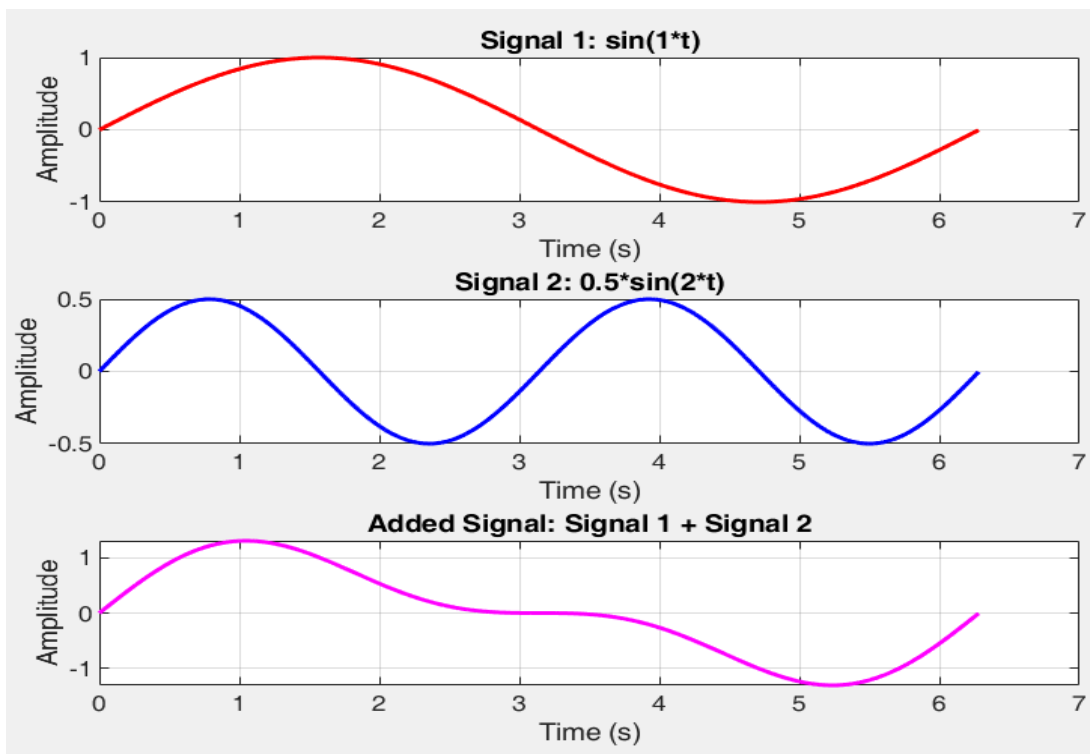
% Add the two signals
added_signal = signal1 + signal2;

% Plot the individual signals and their sum
subplot(3, 1, 1);
plot(t, signal1, 'r', 'LineWidth', 1.5); % Plot first signal
title('Signal 1: sin(1*t)');
xlabel('Time (s)');
ylabel('Amplitude');
```

```
grid on;  
subplot(3, 1, 2);  
plot(t, signal2, 'b', 'LineWidth', 1.5); % Plot second signal  
title('Signal 2: 0.5*sin(2*t)');  
xlabel('Time (s)');  
ylabel('Amplitude');  
grid on;
```

```
subplot(3, 1, 3);  
plot(t, added_signal, 'm', 'LineWidth', 1.5); % Plot added signal  
title('Added Signal: Signal 1 + Signal 2');  
xlabel('Time (s)');  
ylabel('Amplitude');  
grid on;
```

### OUTPUT:



### RESULT:

Thus the addition operation of two signals was successfully completed using MATLAB

EX.NO :8

## FFT AND IFFT SPECTRUM

DATE :

### AIM:

To perform FFT and IFFT spectrum using MATLAB.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Obtain the length of the input sequence(x)and its values. Step-3:  
Using FFT command, determine the dft sequence (X).
- Step-4: Display the sequence and plot its magnitude and phases pectrum. Step-5:  
Using IFFT command, determine the idft sequence (x).
- Step-6: Display the sequence and plot its magnitude and phase spectrum.

### CODING:

```
clc;           % Clear command window
clear;        % Clear workspace
close all;    % Close all figure windows

% Input the sequence
x = input('Enter the sequence (as a vector): '); % e.g., [1, 2, 3, 4]
N = length(x);

% Input the number of FFT points
n = input('Enter the number of FFT points (>= length of the sequence): ');

% Compute the FFT of the input sequence
y = fft(x, n);

% Prepare time vectors for plotting
v = 0:1:N-1;
t = 0:1:n-1;

% 2-D Plots
figure;

% Plot Input Signal
subplot(2,2,1);
stem(v, x, 'filled');
```



```
xlabel('Sample Index');
ylabel('Amplitude');
title('Input Signal');
grid on;
% Plot Magnitude Response
subplot(2,2,2);
stem(t, abs(y), 'filled');
xlabel('Sample Index');
ylabel('Magnitude');
title('Magnitude Response');
grid on;
```

```
% Plot Phase Response
subplot(2,2,3);
stem(t, angle(y), 'filled');
xlabel('Sample Index');
ylabel('Phase (radians)');
title('Phase Response');
grid on;
```

```
% Display FFT results
disp('FFT of x(n):');
disp(y);
% Compute the inverse FFT
y1 = ifft(y, n);
```

```
% Plot IFFT Response
subplot(2,2,4);
stem(t, y1, 'filled');
xlabel('Sample Index');
ylabel('Amplitude');
title('IFFT Response');
grid on;
```

```
% Display IFFT results
disp('IFFT of X(K):');
disp(y1);
```

## OUTPUT:

```
Enter the sequence (as a vector): [1 3+i 5 i 6i 2]
Enter the number of FFT points (>= length of the sequence): 6
Warning: Using only the real component of complex data.
> In matlab.graphics.chart.internal.getRealData (line 63)
In stem (line 96)
In untitled2 (line 24)
FFT of x(n):
Columns 1 through 4

11.0000 + 8.0000i  -3.3301 - 8.6962i   2.0622 + 0.9641i   1.0000 + 4.0000i

Columns 5 through 6

-10.0622 - 5.9641i   5.3301 + 1.6962i

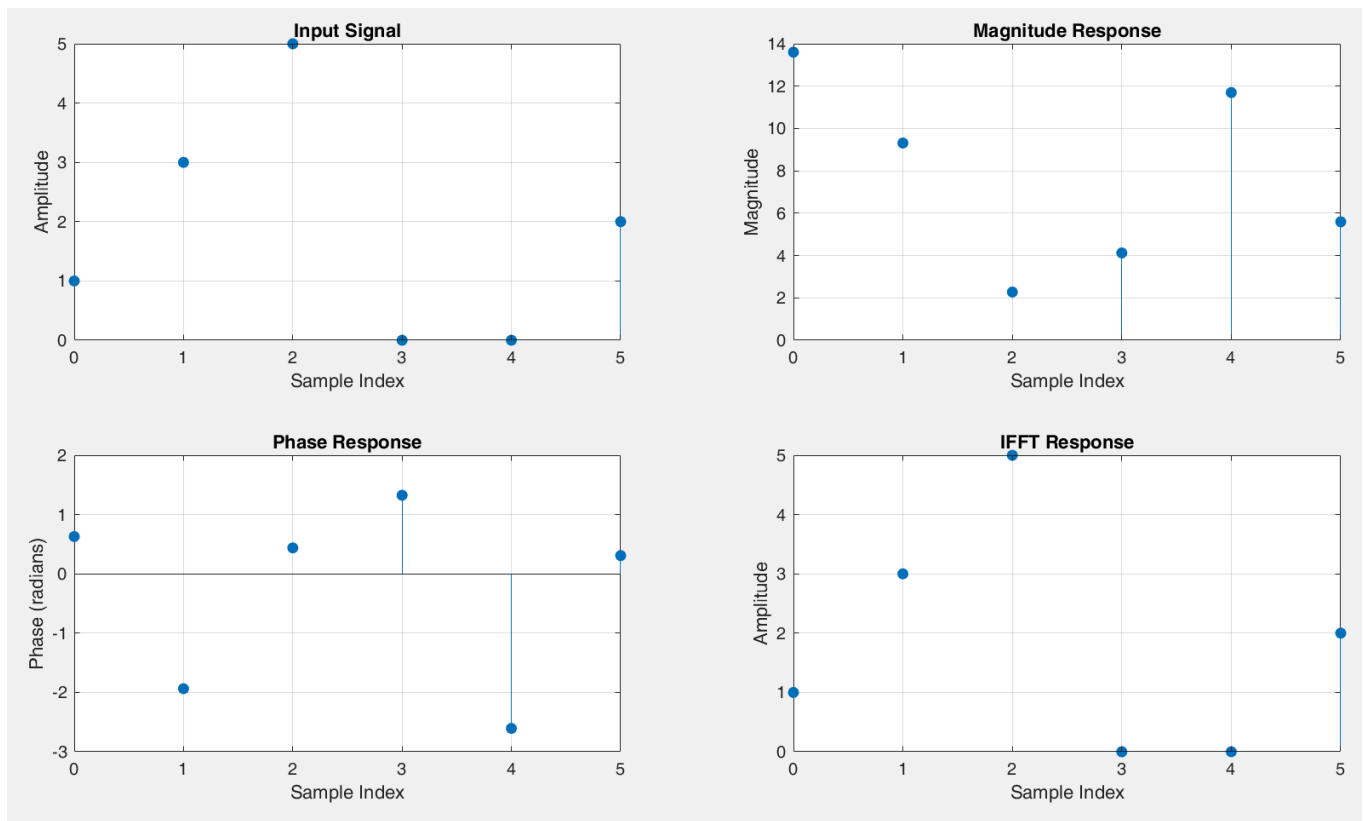
Warning: Using only the real component of complex data.
> In matlab.graphics.chart.internal.getRealData (line 63)
In stem (line 96)
In untitled2 (line 55)
IFFT of X(K):
Columns 1 through 4

1.0000 + 0.0000i   3.0000 + 1.0000i   5.0000 + 0.0000i   0.0000 + 1.0000i

Columns 5 through 6

0.0000 + 6.0000i   2.0000 + 0.0000i

'\'
```



## RESULT:

Thus FFT and IFFT have been determined and their magnitude and phase spectrum are plotted.

EX.NO :9

## DECIMATION IN TIME-FFT AND IFFT

DATE :

### AIM:

To find the DFT for the given input sequence and to plot its magnitude with and without using the FFT function.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Obtain the length of the input sequence and its values. Step-3:  
Define the twiddle factors.
- Step-4: Determine the output of different butterfly stages of the DIT-FFT. Step-5:  
Display the sequence and plot its magnitude.
- Step-6: Bit reverse the FFT output to compute IFFT.
- Step-7: Repeat steps 3 to 5 to determine IFFT by DIT method.
- Step-8: Divide the output by the length of the sequence to find IFFT output. Step-9:  
Plot the output magnitude plot waveforms.

### CODING:

```
clc;
clear;
close all;

x = input('Enter the 8-point sequence: ');

w1 = (1/sqrt(2)) - (sqrt(-1)/sqrt(2));
w2 = -sqrt(-1);
w3 = (-1/sqrt(2)) - (sqrt(-1)/sqrt(2));

% FFT Section
for n = 1:4
    y(n) = x(n) + x(n+4);
    y(n+4) = x(n) - x(n+4);
end

for n = 1:2
    z(n) = y(n) + y(n+2);
    z(n+2) = y(n) - y(n+2);
    z(n+4) = y(n+4) + y(n+6) * w2;
```

```

    z(n+6) = y(n+4) - y(n+6) * w2;
end
n = 1;
X(n) = z(n) + z(n+1);
X(n+1) = z(n) - z(n+1);
X(n+2) = z(n+2) + z(n+3) * w2;
X(n+3) = z(n+2) - z(n+3) * w2;
X(n+4) = z(n+4) + z(n+5) * w1;
X(n+5) = z(n+4) - z(n+5) * w1;
X(n+6) = z(n+6) + z(n+7) * w3;
X(n+7) = z(n+6) - z(n+7) * w3;

```

```

X = bitrevorder(X);
disp('FFT of x(n):');
disp(X);

```

```

% IFFT Section
disp('Computing IFFT...');

```

```

X = conj(X);

```

```

for k = 1:4
    y(k) = X(k) + X(k+4);
    y(k+4) = X(k) - X(k+4);
end

```

```

for k = 1:2
    z(k) = y(k) + y(k+2);
    z(k+2) = y(k) - y(k+2);
    z(k+4) = y(k+4) + y(k+6) * w2;
    z(k+6) = y(k+4) - y(k+6) * w2;
end

```

```

k = 1;
x(k) = z(k) + z(k+1);
x(k+1) = z(k) - z(k+1);
x(k+2) = z(k+2) + z(k+3) * w2;
x(k+3) = z(k+2) - z(k+3) * w2;
x(k+4) = z(k+4) + z(k+5) * w1;
x(k+5) = z(k+4) - z(k+5) * w1;
x(k+6) = z(k+6) + z(k+7) * w3;

```

```
x(k+7) = z(k+6) - z(k+7) * w3;
```

```
x = conj(x);
```

```
x = bitrevorder(x) / 8;
```

```
disp('IFFT of X(K):');
```

```
disp(x);
```

### OUTPUT:

```
Enter the 8-point sequence: [1 2 3 4 5 6 7 8]
```

```
FFT of x(n):
```

```
Columns 1 through 4
```

```
36.0000 + 0.0000i  -4.0000 + 9.6569i  -4.0000 + 4.0000i  -4.0000 + 1.6569i
```

```
Columns 5 through 8
```

```
-4.0000 + 0.0000i  -4.0000 - 1.6569i  -4.0000 - 4.0000i  -4.0000 - 9.6569i
```

```
Computing IFFT...
```

```
IFFT of X(K):
```

```
1.0000    2.0000    3.0000    4.0000    5.0000    6.0000    7.0000    8.0000
```

### RESULT:

Thus FFT and IFFT have been determined using the DIT method.



EX.NO :10

## FREQUENCY RESPONSE OF LTI SYSTEM

DATE :

### AIM:

To determine the frequency response of a low pass filter LTI system.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Obtain the numerator and denominator polynomial coefficients. Step-3:  
Using freqz command, determine the response of the filter.
- Step-4: Plot the magnitude and phase spectrum.

### CODING:

```
clc;
b=input('Enter the numerator polynomial coefficients');
a=input('Enter the denominator polynomial coefficients');
w=0:0.01:2*pi;
h=freqz(b,a,w);
m=abs(h);
p=angle(h);
subplot(2,1,1);
plot(w,m);
title('Magnitude spectrum');
xlabel('Frequency X2pi radians/sample');
ylabel('Magnitude in volts');

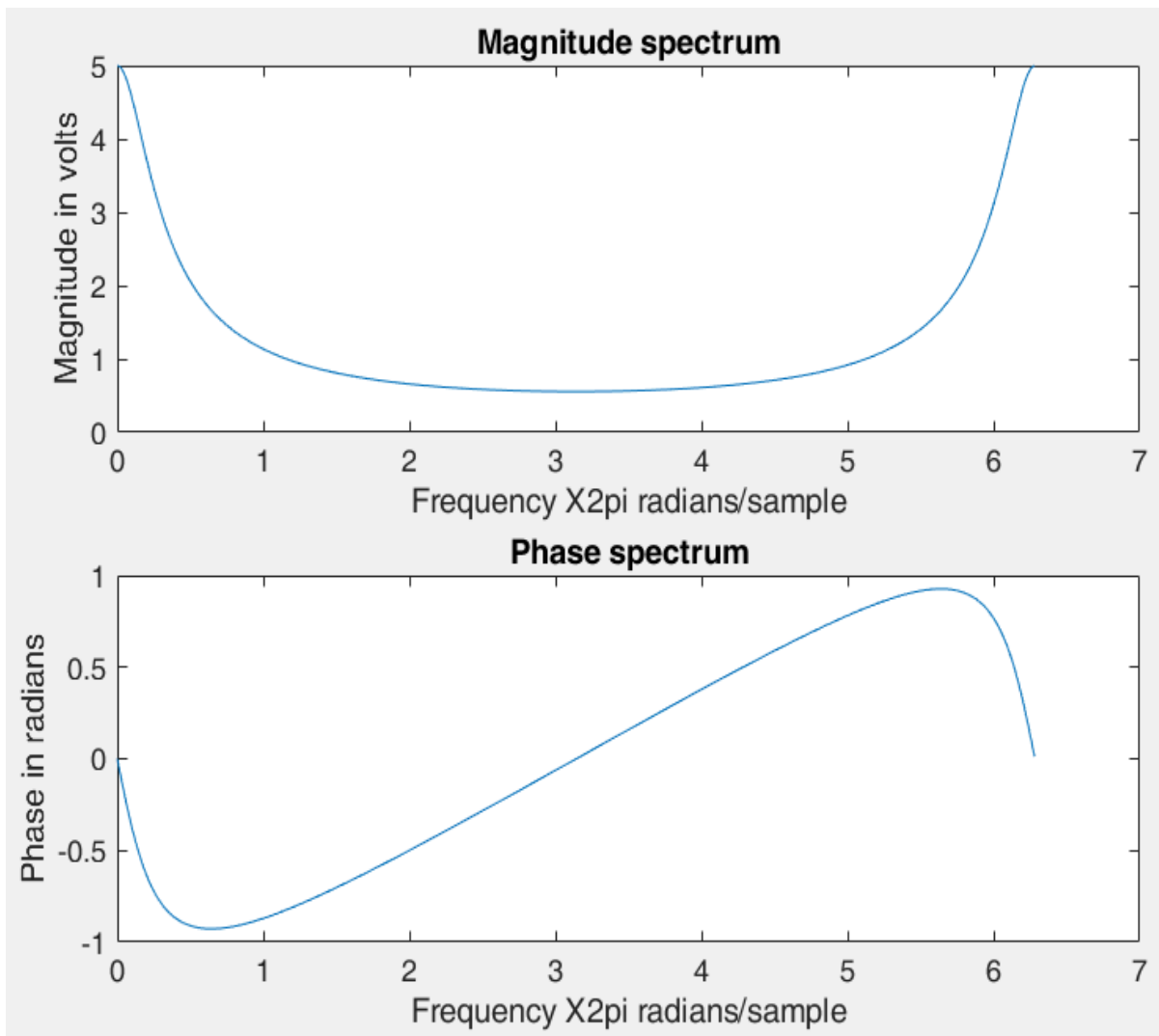
subplot(2,1,2);
plot(w,p);
title('Phase spectrum');
xlabel('Frequency X2pi radians/sample');
ylabel('Phase in radians');
```



## OUTPUT:

Enter the numerator polynomial coefficients[1 0]

Enter the denominator polynomial coefficients[1 -0.8]



## RESULT:

Thus the frequency response of a LTI system is plotted.

EX.NO :11

## LINEAR CONVOLUTION USING FFT

DATE :

### AIM:

To determine the linear convolution of two input sequences using FFT method.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace. Step-2: Input the sequences  $x_1(n)$  &  $x_2(n)$ .
- Step-3: Find the length of the output sequence  $N=(\text{length}(x_1)+\text{length}(x_2)-1)$
- Step-4: If  $\text{length}(x_1)<N$ , zero padding is done and the resulting sequence is circular shifted to get a  $N*N$  matrix.
- Step-5: If  $\text{length}(x_2)<N$ , zero padding is done and the resulting sequence is taken as a column matrix.
- Step-6: Multiplication of the two sequences gives the required sequence  $x_3(n)$ .
- Step-7: Display the result.

### CODING:

```
clc;
clear;
close all;

a = input('Enter the array a: ');
n1 = length(a);
b = input('Enter the array b: ');
n2 = length(b);

N = n1 + n2 - 1; % Length of the output signal

% Plot input signals
subplot(2,2,1);
stem(1:n1, a, 'b', 'LineWidth', 1);
title('INPUT-A');
xlabel('n');
ylabel('a(n)');

subplot(2,2,2);
stem(1:n2, b, 'b', 'LineWidth', 1);
title('INPUT-B');
xlabel('n');
```

```

ylabel('b(n)');
% Zero-padding the input signals to match the output length
for m = n1+1:N
    a(m) = 0;
end
for m = n2+1:N
    b(m) = 0;
end

% Compute FFT of both input signals
A = fft(a);
B = fft(b);

% Multiply the FFTs
C = A .* B;

% Compute inverse FFT to get the result
c = ifft(C);

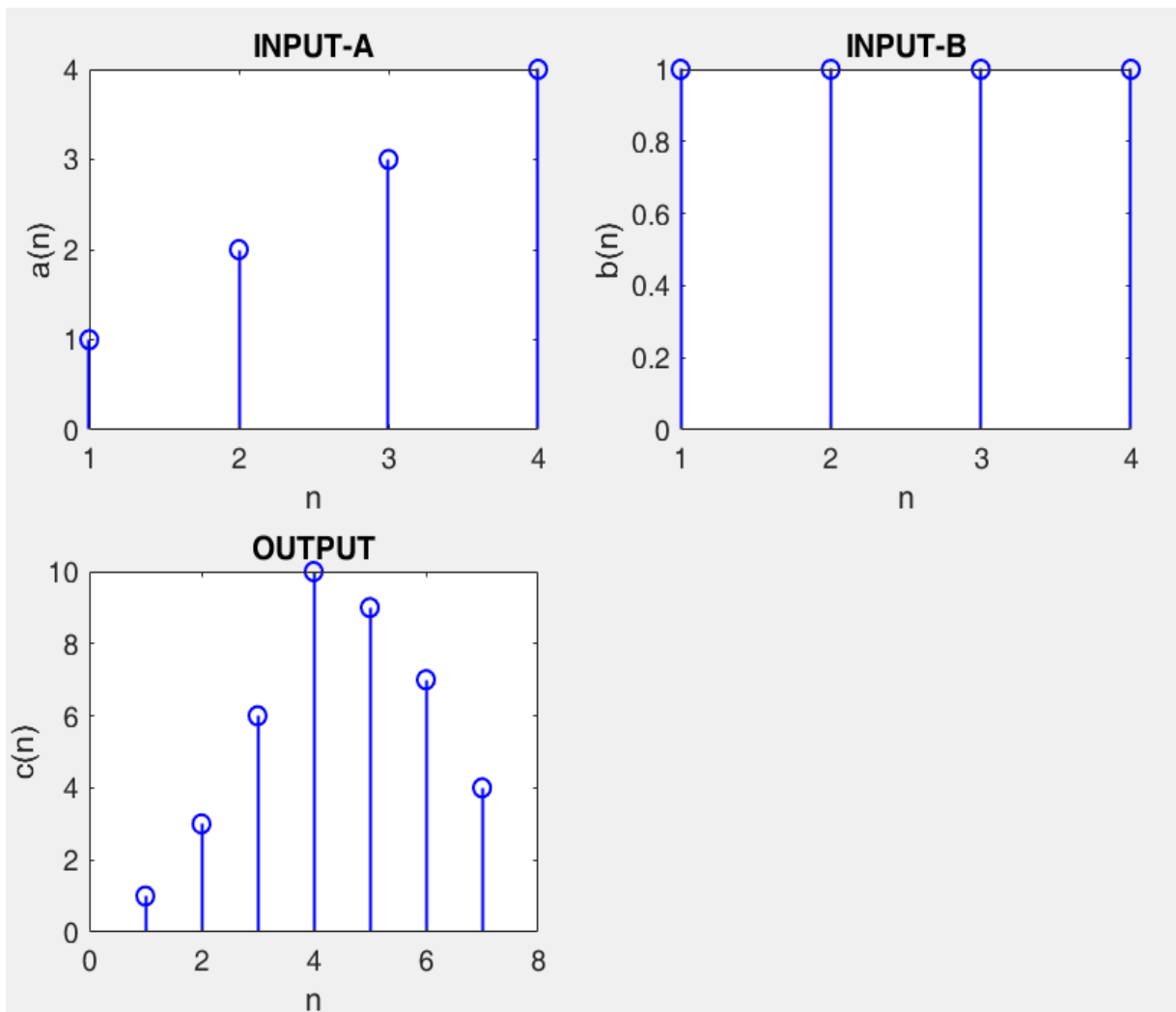
% Plot the output signal
subplot(2,2,3);
stem(1:N, c, 'b', 'LineWidth', 1);
title('OUTPUT');
xlabel('n');
ylabel('c(n)');

```

## OUTPUT:

Enter the array a: [1 2 3 4]

Enter the array b: [1 1 1 1]



**RESULT:**

Thus linear convolution has been implemented using FFT.

EX.NO :12

## LINEAR CONVOLUTION USING MATRIX MULTIPLICATION

DATE :

### AIM:

To determine the linear convolution of two input sequences using matrix multiplication method.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace. Step-2: Input the sequences  $x_1(n)$  &  $x_2(n)$ .
- Step-3: Find the length of the output sequence  $N=(\text{length}(x_1)+\text{length}(x_2)-1)$
- Step-4: If  $\text{length}(x_1)<N$ , zero padding is done and the resulting sequence is circular shifted to get a  $N*N$  matrix.
- Step-5: If  $\text{length}(x_2)<N$ , zero padding is done and the resulting sequence is taken as a column matrix.
- Step-6: Multiplication of the two sequences gives the required sequence  $x_3(n)$ .
- Step-7: Display the result.

### CODING:

```
clc;
clear;
close all;

% Input two arrays
a = input('Enter the array a: ');
n1 = length(a);
b = input('Enter the array b: ');
n2 = length(b);

% Determine the length of the output signal
N = n1 + n2 - 1;

% Plot input sequences
subplot(2,2,1);
stem(1:n1, a, 'b', 'LineWidth', 1);
title('INPUT-A');
xlabel('n');
ylabel('a(n)');
subplot(2,2,2);
```

```

stem(1:n2, b, 'b', 'LineWidth',1);
title('INPUT-B');
xlabel('n');
ylabel('b(n)');

% Zero-padding to match the output length
for m = n1+1:N
    a(m) = 0;
end
for m = n2+1:N
    b(m) = 0;
end

% Create the convolution matrix by circularly shifting 'a'
c = a'; % Transpose 'a'
for i = 1:N-1
    e = circshift(a', i); % Circular shift by i
    c = [c, e]; % Append the shifted version of 'a'
end

% Compute the output by multiplying the convolution matrix with 'b'
f = c * b';

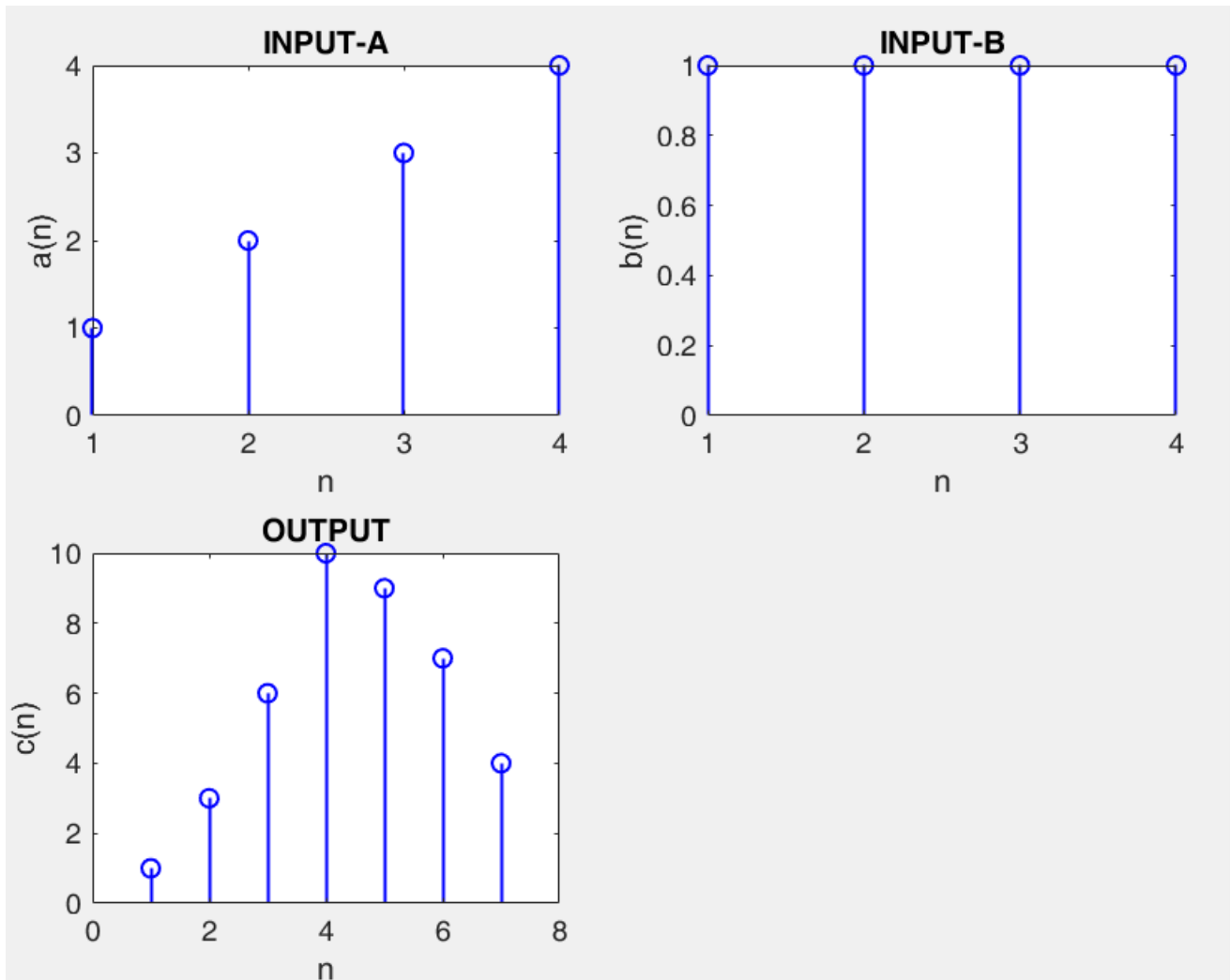
% Plot the output
subplot(2,2,3);
stem(1:N, f, 'b', 'LineWidth', 1);
title('OUTPUT');
xlabel('n');
ylabel('f(n)');

```

## OUTPUT:

Enter the array a: [1 2 3 4]

Enter the array b: [1 1 1 1]



## RESULT:

Thus linear convolution has been implemented using matrix multiplication method.

EX.NO :13

## CIRCULAR CONVOLUTION

DATE :

### AIM:

To compute the circular convolution of the given input sequences using matlab.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace. Step-2: Input the sequences  $x_1(n)$  &  $x_2(n)$ .
- Step-3: Find length of output sequence  $N = \max(\text{length}(x_1), \text{length}(x_2))$ .
- Step-4: If  $\text{length}(x_2) < N$ , zero padding is done and the resulting  $x_2(n)$  is circular shifted to form  $N \times N$  matrix
- Step-5: If  $\text{length}(x_1) < N$ , zero padding is done and the resulting sequence is taken as column matrix.
- Step-6: Multiplication of the two sequences gives the required sequence  $x_3(n)$ .
- Step-7: Display the result.

### CODING:

```
clc;
clear;

% Input sequences
a = input('Enter the sequence a: ');
b = input('Enter the sequence b: ');

n1 = length(a);
n2 = length(b);

% Zero-padding to make both sequences the same length
if n1 < n2
    a(n1+1:n2) = 0;
else
    b(n2+1:n1) = 0;
end
N = length(a);

disp('Input sequence a:');
disp(a);
disp('Input sequence b:');
disp(b);
```



```

A = a';
for i = 1:n1-1
    A = [A, circshift(a', i)];
end
f = A * b';

disp('Circular Convolution Result:');
disp(f);

subplot(2,2,1);
stem(1:N, a, 'b', 'LineWidth', 1);
title('INPUT-A');
xlabel('n');
ylabel('a(n)');

subplot(2,2,2);
stem(1:N, b, 'b', 'LineWidth', 1);
title('INPUT-B');
xlabel('n');
ylabel('b(n)');

subplot(2,2,3);
stem(1:N, f, 'b', 'LineWidth', 1);
title('OUTPUT-CIRCULAR CONVOLUTION');
xlabel('n');
ylabel('f(n)');

A_fft = fft(a);
B_fft = fft(b);
C_fft = A_fft .* B_fft;
c = ifft(C_fft);

disp('FFT-based Convolution Result:');
disp(c);

subplot(2,2,4);
stem(1:N, real(c), 'b', 'LineWidth', 1);
title('OUTPUT-FFT');
xlabel('n');
ylabel('c(n)');

```

### OUTPUT:

Enter the sequence a: [1 2 3 4]

Enter the sequence b: [1 2]

Input sequence a:

1 2 3 4

Input sequence b:

1 2 0 0

Circular Convolution Result:

9

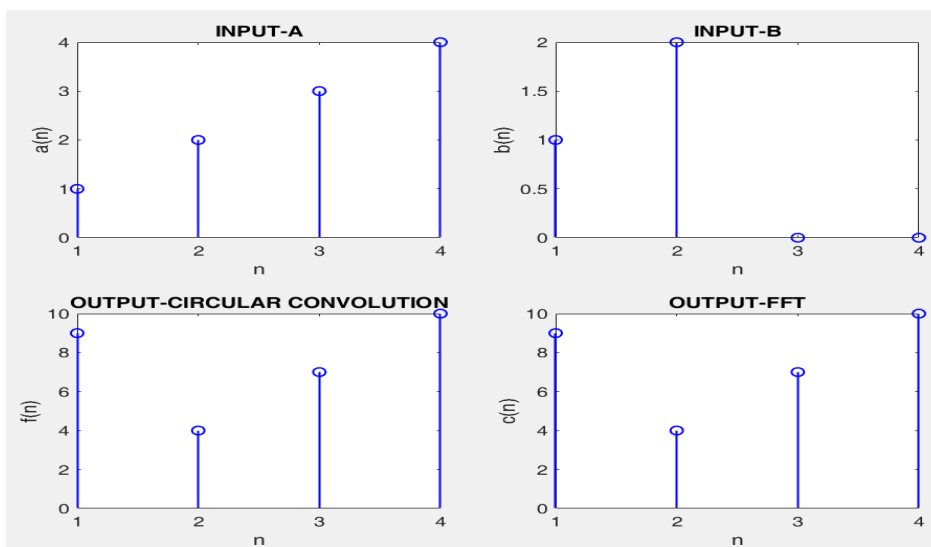
4

7

10

FFT-based Convolution Result:

9 4 7 10



### RESULT:

Thus circular convolution has been implemented using FFT.

EX.NO :14

## DESIGN OF IIR FILTERS ( BUTTERWORTH )

DATE :

### AIM:

To design the following IIR filters( Butterworth type ).

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Select the type of the filter and its specifications.
- Step-3: Determine the order of the filter and normalized frequency.
- Step-4: Plot the frequency response of the filter using freqz command.

### CODING:

```
clc;
clear;
close all;

wp = input('Enter the passband frequency (normalized 0-1): ');
ws = input('Enter the stopband frequency (normalized 0-1): ');
Rp = input('Enter the passband attenuation (in dB): ');
Rs = input('Enter the stopband attenuation (in dB): ');

% Calculate filter order and cutoff frequency
[n, wn] = buttord(wp, ws, Rp, Rs);

% Display the filter order and cutoff frequency
disp('Order of the filter:');
disp(n);
disp('Cutoff frequency:');
disp(wn);

% Determine the filter type
if wp < ws
    type = 'low';
elseif wp > ws
    type = 'high';
elseif ws(1) < wp(1) && wp(2) < ws(2)
    type = 'bandpass';
elseif wp(1) < ws(1) && ws(2) < wp(2)
```

```

    type = 'stop';
else
    error('Invalid frequency input.');
```

end

```

% Design Butterworth filter
[b, a] = butter(n, wn, type);

% Frequency response of the filter
freqz(b, a);
title(['Butterworth Filter Frequency Response (' type '-pass)']);
```

## OUTPUT:

### HIGH PASS

Enter the passband frequency (normalized 0-1): 0.8

Enter the stopband frequency (normalized 0-1): 0.3

Enter the passband attenuation (in dB): 20

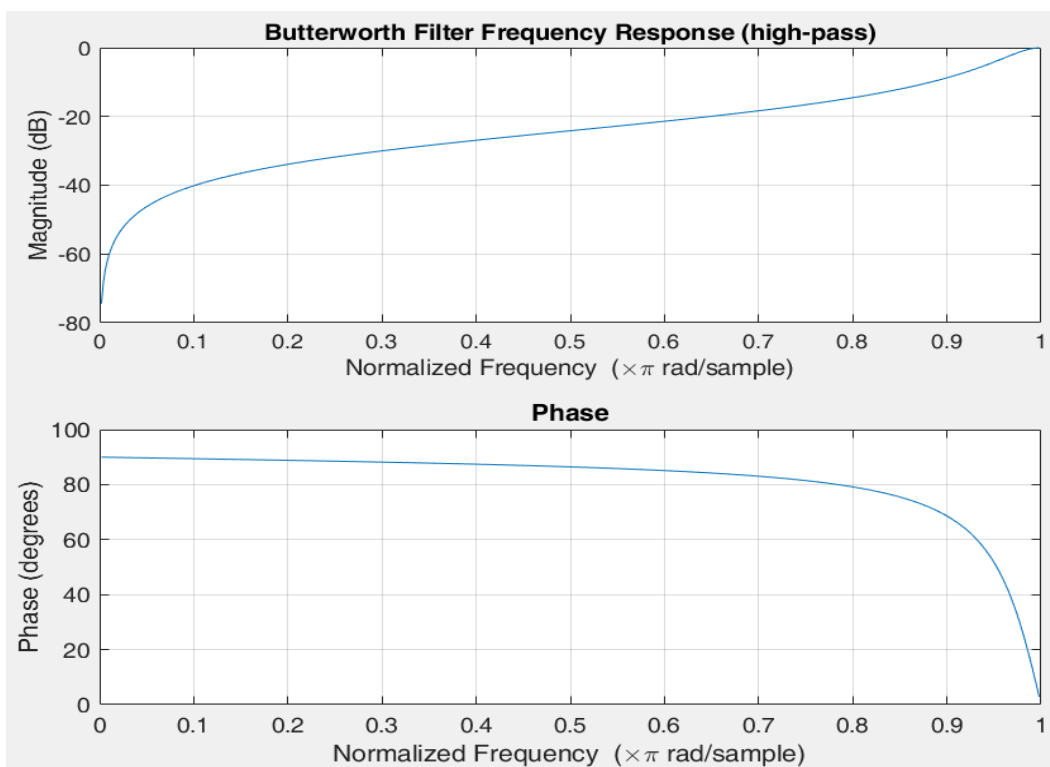
Enter the stopband attenuation (in dB): 30

Order of the filter:

1

Cutoff frequency:

0.9605



## OUTPUT:

### BAND PASS

Enter the passband frequency (normalized 0-1): [0.4 0.6]

Enter the stopband frequency (normalized 0-1): [0.2 0.8]

Enter the passband attenuation (in dB): 0.5

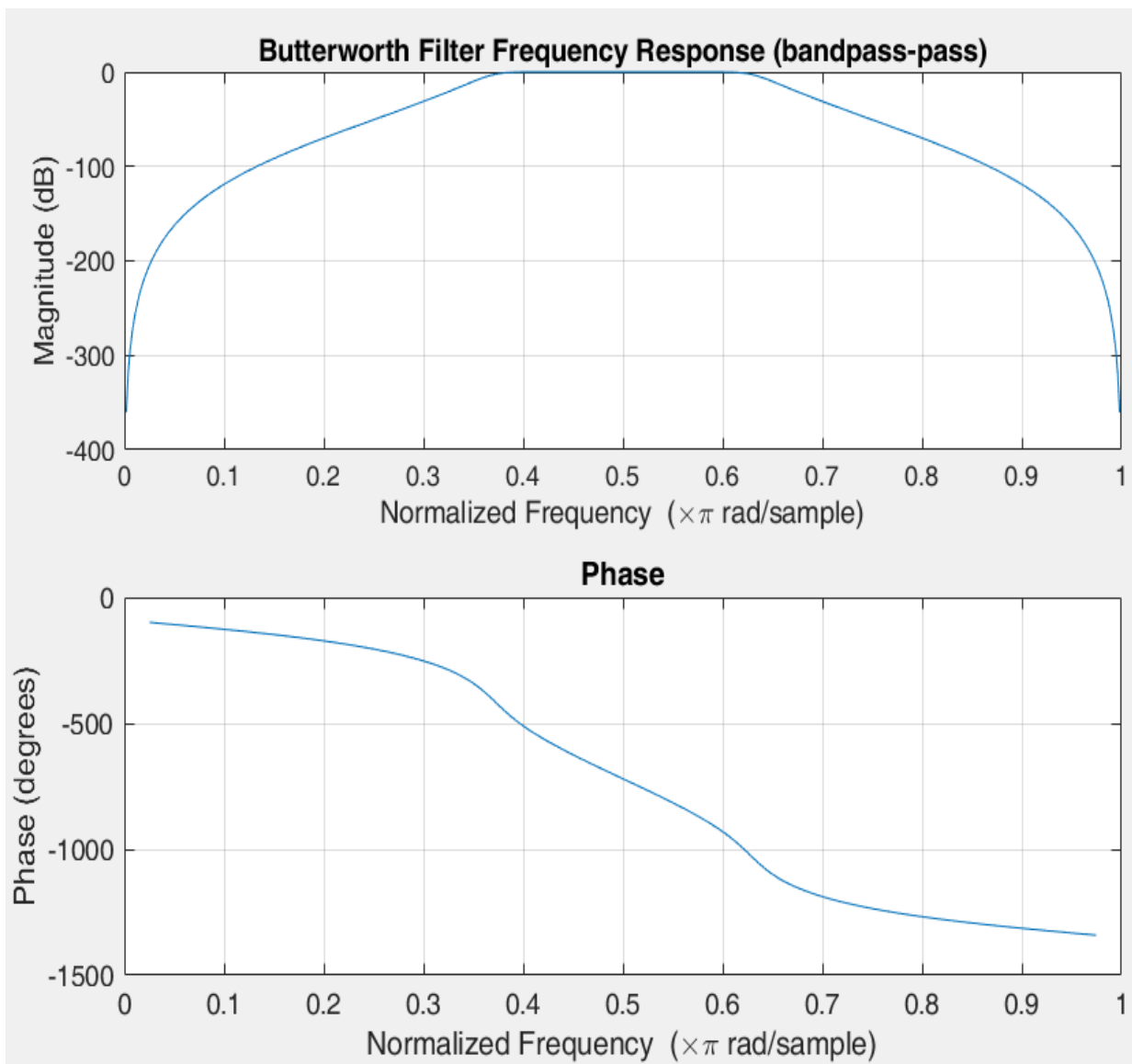
Enter the stopband attenuation (in dB): 70

Order of the filter:

7

Cutoff frequency:

0.3693 0.6307



## OUTPUT:

### BAND STOP

Enter the passband frequency (normalized 0-1): [0.1 0.8]

Enter the stopband frequency (normalized 0-1): [0.2 0.6]

Enter the passband attenuation (in dB): 3

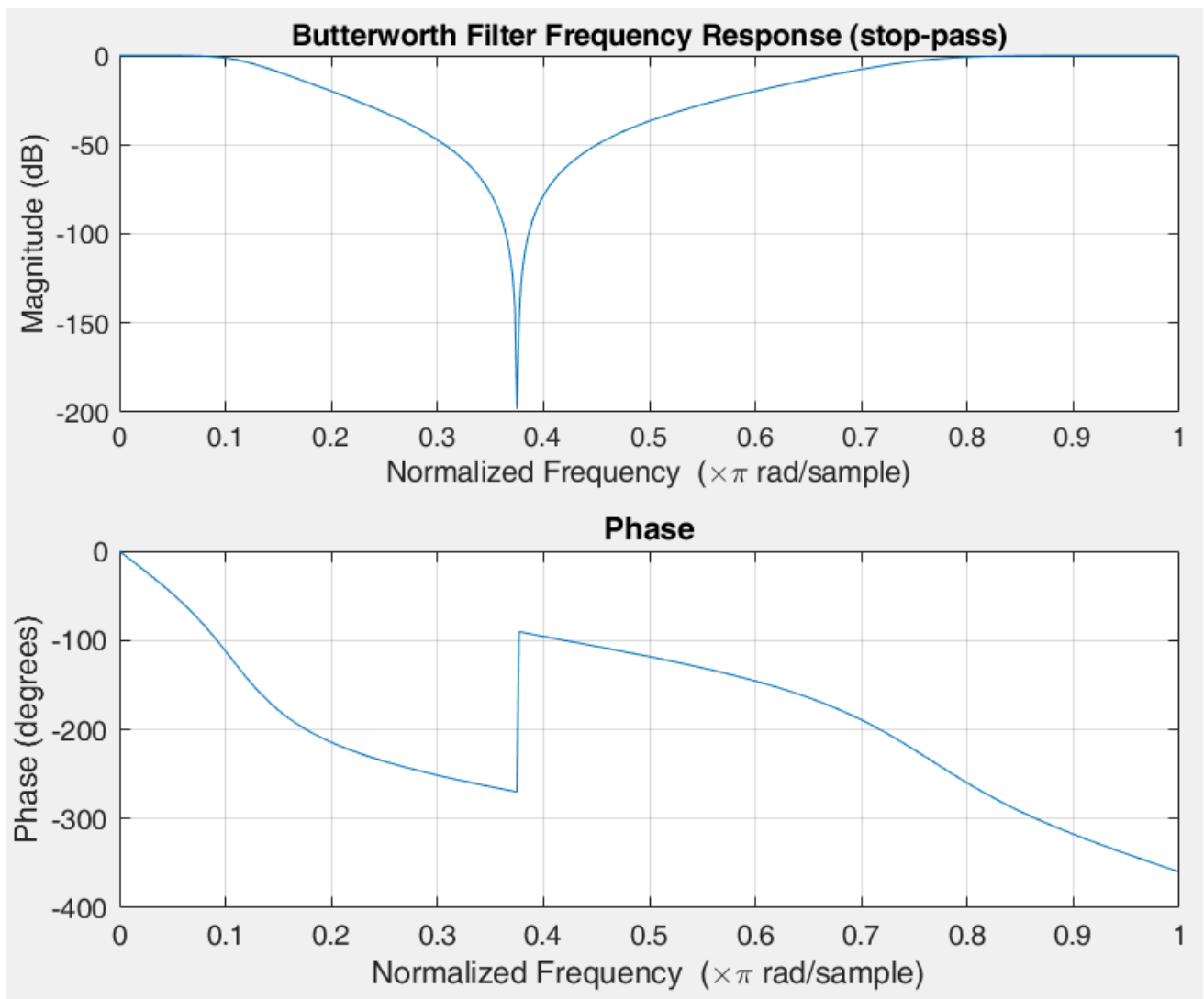
Enter the stopband attenuation (in dB): 20

Order of the filter:

3

Cutoff frequency:

0.1152 0.7528



## OUTPUT:

### LOW PASS

Enter the passband frequency (normalized 0-1): 0.3

Enter the stopband frequency (normalized 0-1): 0.8

Enter the passband attenuation (in dB): 20

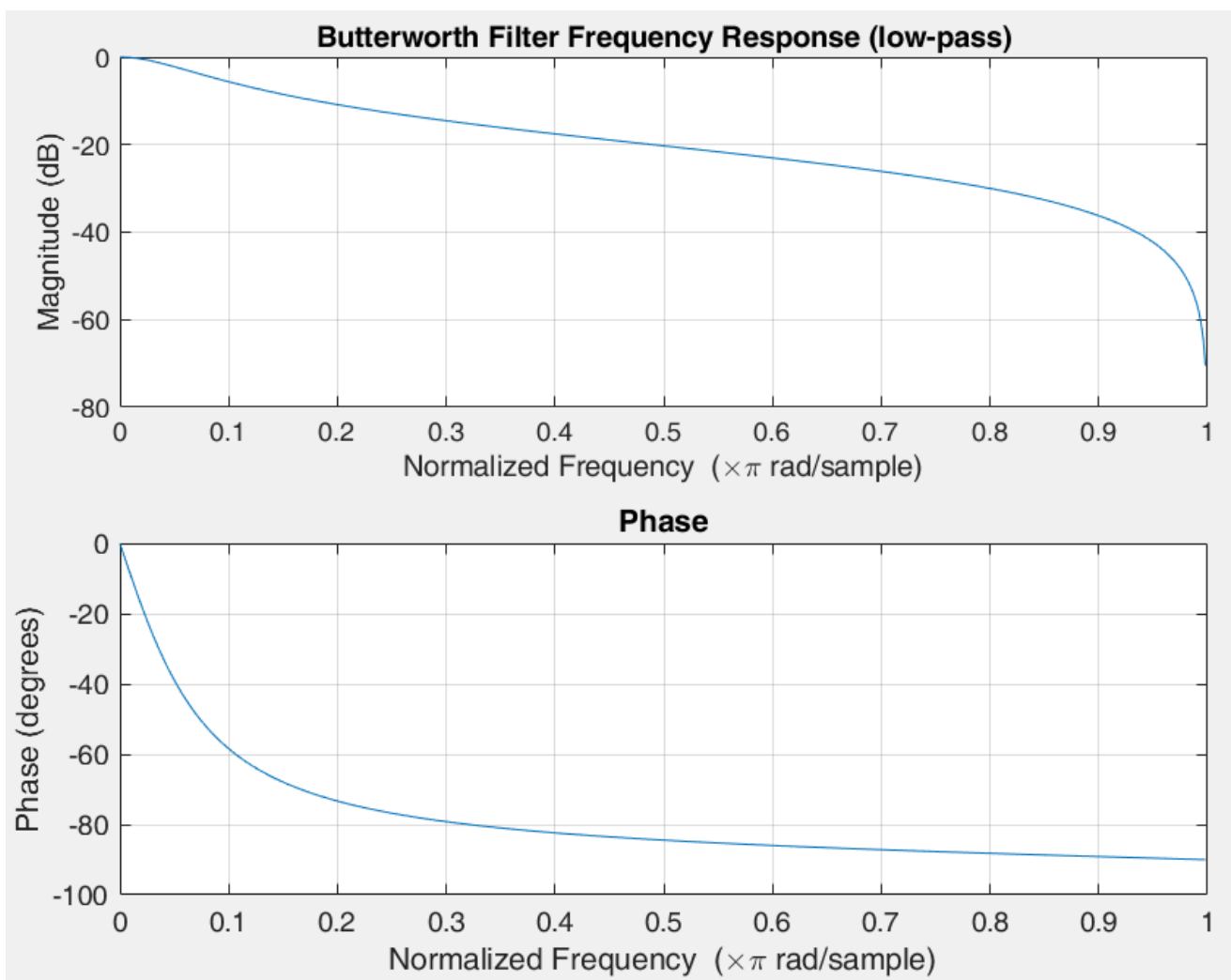
Enter the stopband attenuation (in dB): 30

Order of the filter:

1

Cutoff frequency:

0.0618



## RESULT:

Thus Butterworth type IIR filters have been implemented.

EX.NO :15

## DESIGN OF IIR FILTERS ( CHEBYCHEV )

DATE :

### AIM:

To design the following IIR filters( Chebychev type ).

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Select the type of the filter and its specifications.
- Step-3: Determine the order of the filter and normalized frequency.
- Step-4: Plot the frequency response of the filter using freqz command.

### CODING:

```
clc;
clear;
close all;

wp = input('Enter the passband frequency (normalized 0-1): ');
ws = input('Enter the stopband frequency (normalized 0-1): ');
Rp = input('Enter the passband attenuation (in dB): ');
Rs = input('Enter the stopband attenuation (in dB): ');

% Calculate filter order and cutoff frequency using Chebyshev
[n, wn] = cheb1ord(wp, ws, Rp, Rs);

% Display the filter order and cutoff frequency
disp('Order of the filter:');
disp(n);
disp('Cutoff frequency:');
disp(wn);

% Determine the filter type based on frequency input
if wp < ws
    type = 'low';
elseif wp > ws
    type = 'high';
elseif ws(1) < wp(1) && wp(2) < ws(2)
    type = 'bandpass';
elseif wp(1) < ws(1) && ws(2) < wp(2)
```



```

    type = 'stop';
else
    error('Invalid frequency input.');
```

end

```

% Design Chebyshev filter
[b, a] = cheby1(n, Rp, wn, type);

% Frequency response of the filter
freqz(b, a);
title(['Chebyshev Type I Filter Frequency Response (' type '-pass)']);
```

## OUTPUT:

### HIGH PASS

Enter the passband frequency (normalized 0-1): 0.8

Enter the stopband frequency (normalized 0-1): 0.3

Enter the passband attenuation (in dB): 20

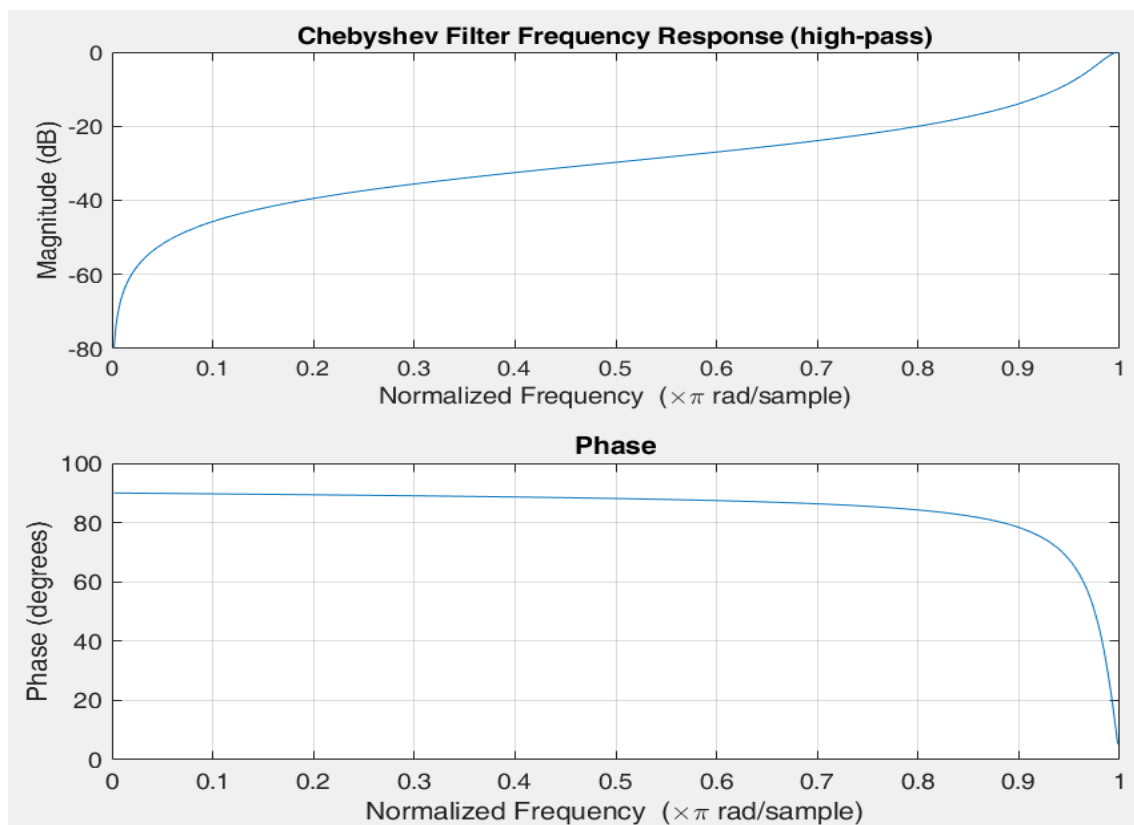
Enter the stopband attenuation (in dB): 30

Order of the filter:

1

Cutoff frequency:

0.8000



## OUTPUT:

### BAND PASS

Enter the passband frequency (normalized 0-1): [0.4 0.6]

Enter the stopband frequency (normalized 0-1): [0.2 0.8]

Enter the passband attenuation (in dB): 0.5

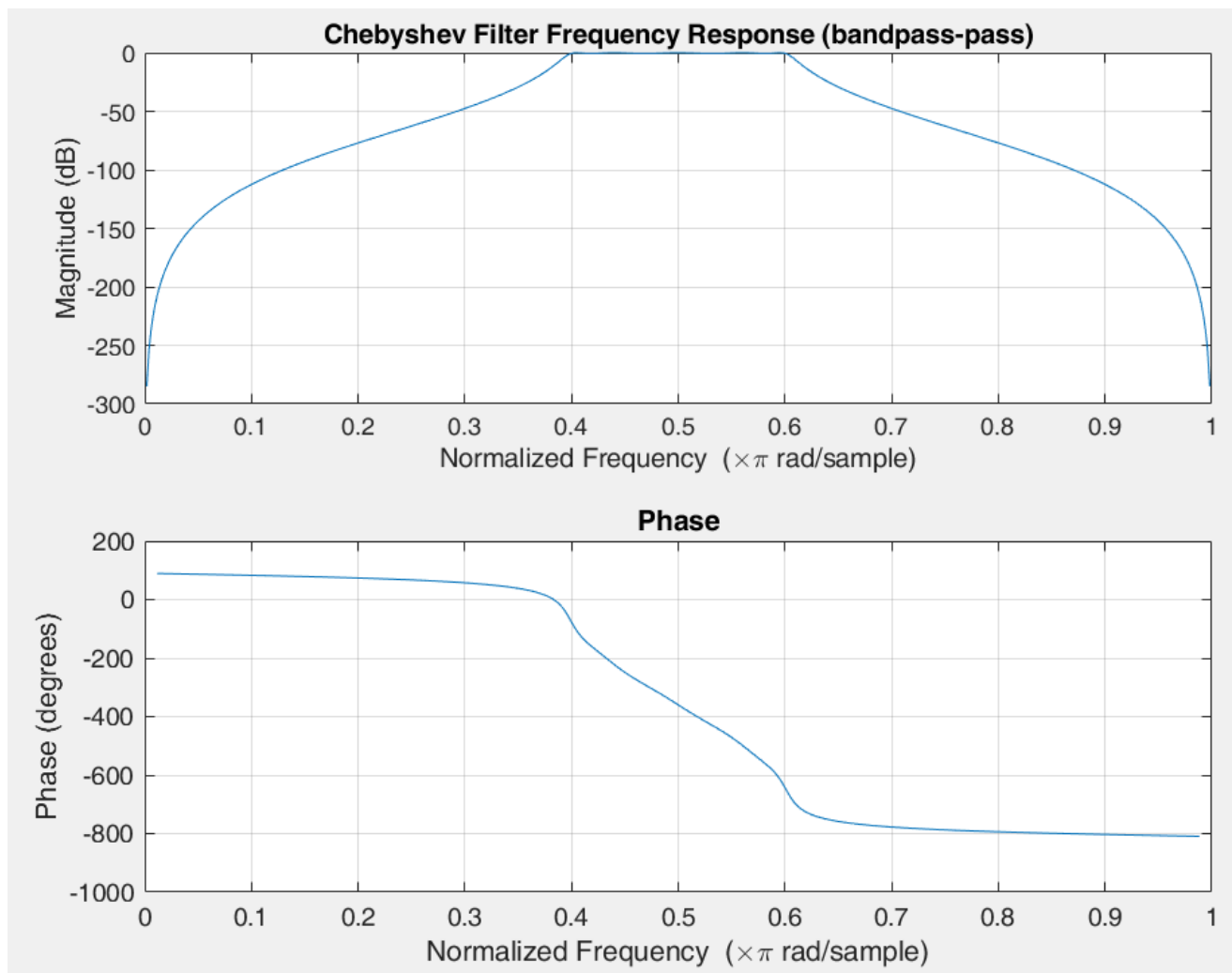
Enter the stopband attenuation (in dB): 70

Order of the filter:

5

Cutoff frequency:

0.4000 0.6000



## OUTPUT:

### BAND STOP

Enter the passband frequency (normalized 0-1): [0.1 0.8]

Enter the stopband frequency (normalized 0-1): [0.2 0.6]

Enter the passband attenuation (in dB): 3

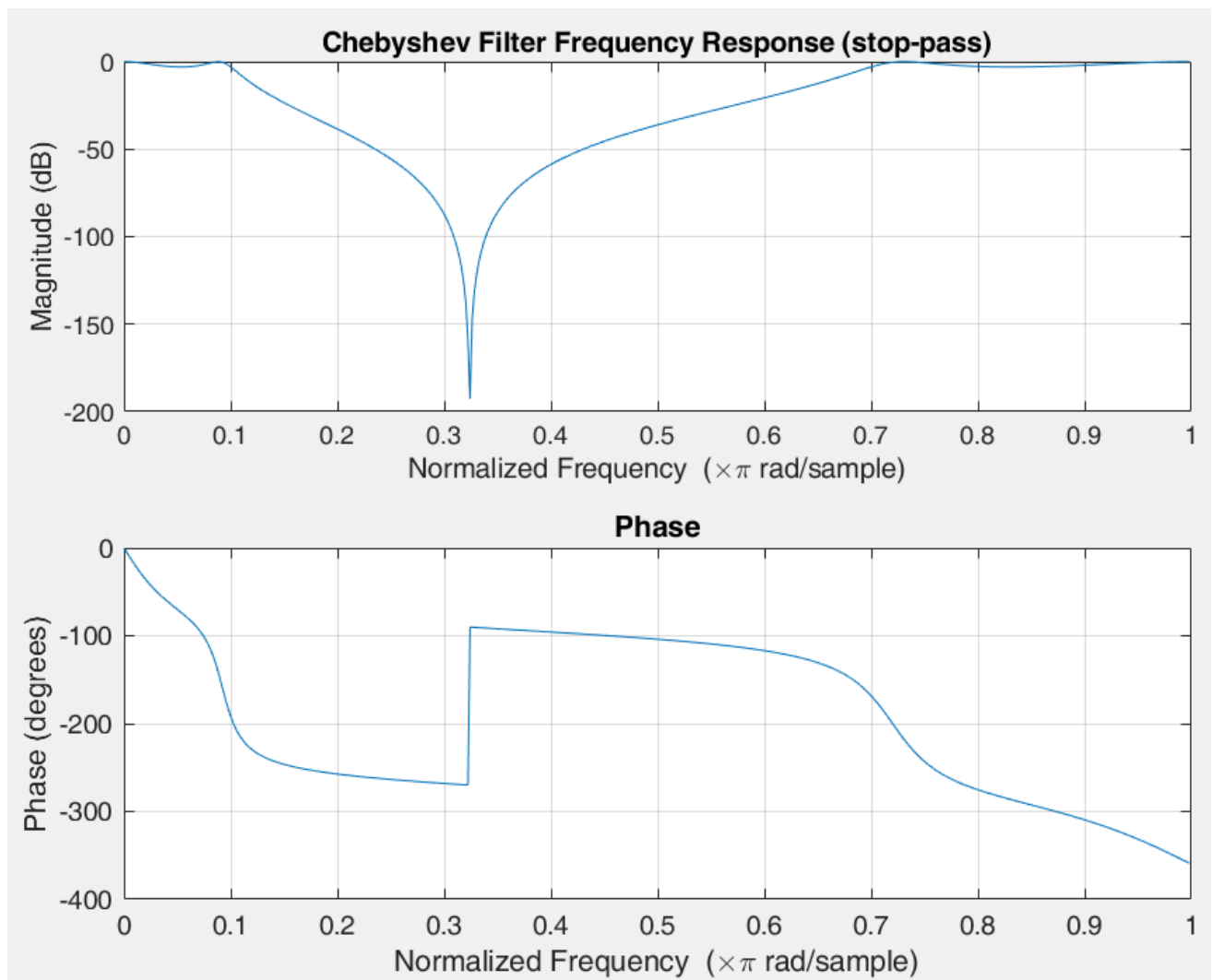
Enter the stopband attenuation (in dB): 20

Order of the filter:

3

Cutoff frequency:

0.1000 0.7000



## OUTPUT:

### LOW PASS

Enter the passband frequency (normalized 0-1): 0.3

Enter the stopband frequency (normalized 0-1): 0.8

Enter the passband attenuation (in dB): 20

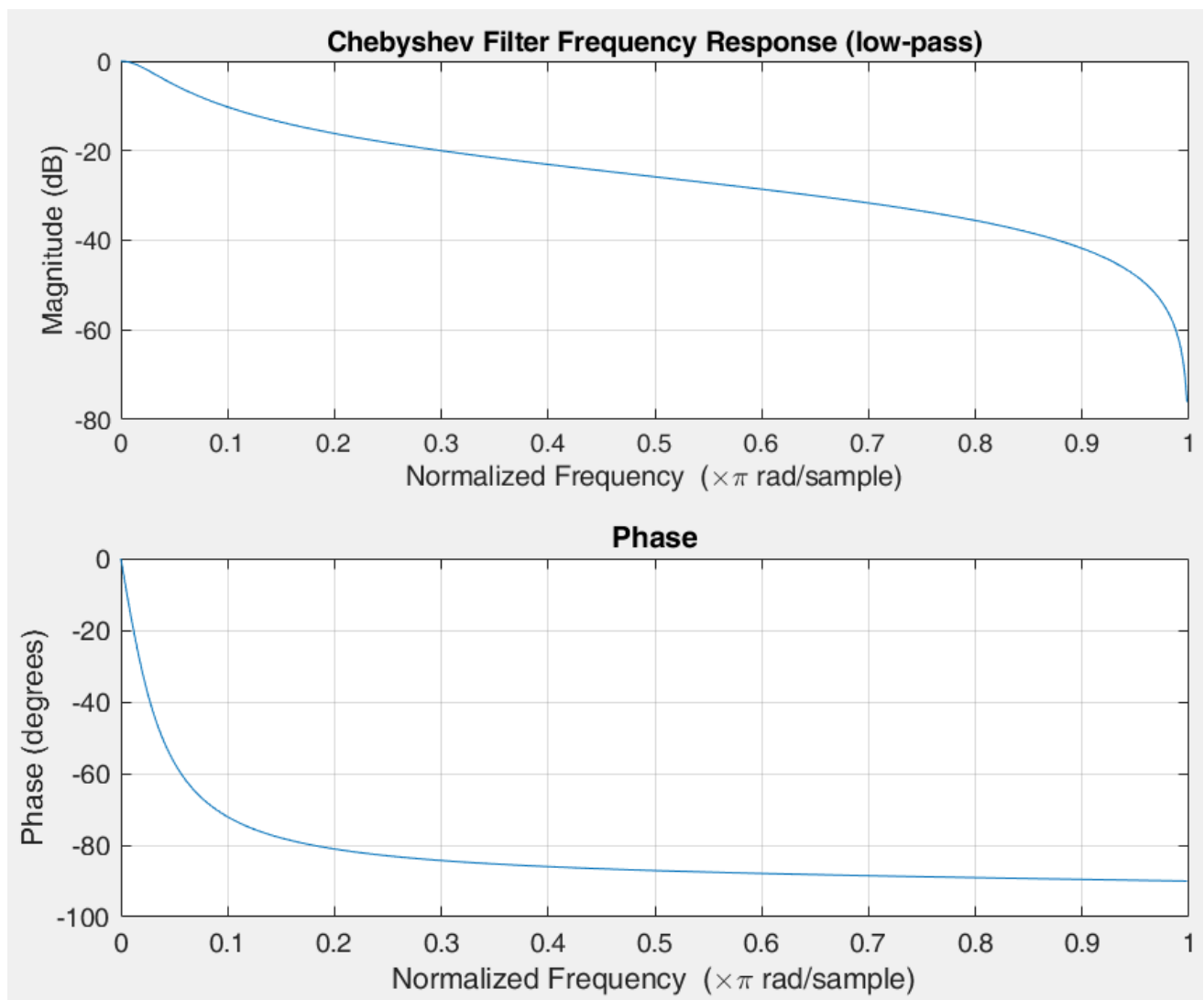
Enter the stopband attenuation (in dB): 30

Order of the filter:

1

Cutoff frequency:

0.3000



## RESULT:

Thus Chebychev type IIR filters have been implemented.

EX.NO :16

## FIR IMPLEMENTATION USING WINDOWS

DATE :

### AIM:

To design the following FIR filters using windowing techniques

- LPF using Hamming window
- HPF using Blackman window.

**SOFTWARE:MATLAB**

### ALGORITHM:

- Step-1: Clear the command window and workspace.  
Step-2: Select the type of the filter and its specifications.  
Step-3: Determine the order of the filter and normalized frequency.  
Step-4: Plot the frequency response of the filter using freqz command.

### CODING:

#### LOW PASS FILTER:

```
clc;
wc = 0.5 * pi;
N = 25;
alpha = (N - 1) / 2;
eps = 0.001;
n = 0:1:N-1;

% Ideal low pass filter (sinc function)
hd = sin(wc * (n - alpha + eps)) ./ (pi * (n - alpha + eps));
wr = boxcar(N);

hn = hd .* wr';
w = 0:0.01:pi;
h = freqz(hn, 1, w);

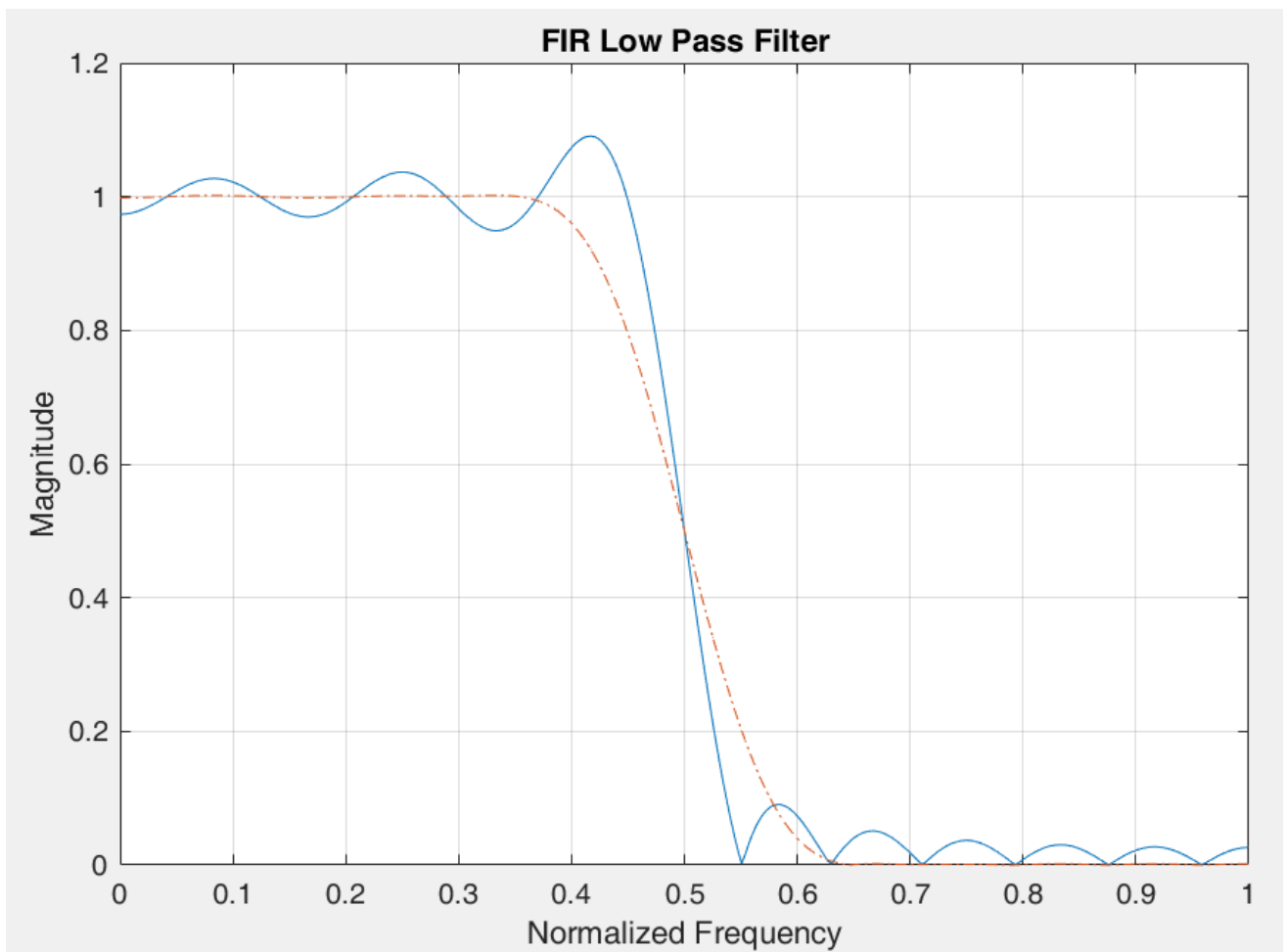
plot(w / pi, abs(h));
hold on;

% Apply Hamming window
wh = hamming(N);
hn = hd .* wh';
w = 0:0.01:pi;
```

```
h = freqz(hn, 1, w);  
plot(w / pi, abs(h), '-.');  
grid;  
title('FIR Low Pass Filter');  
ylabel('Magnitude');  
xlabel('Normalized Frequency');  
hold off;
```

**OUTPUT:**

**LOW PASS FILTER:**



**CODING:**  
**HIGH PASS FILTER:**

```
clc;
wc = 0.5 * pi;
N = 25;
alpha = (N - 1) / 2;
eps = 0.001;
n = 0:1:N-1;

% Ideal high pass filter
hd = (sin(pi * (n - alpha + eps)) - sin(wc * (n - alpha + eps))) ./ (pi * (n - alpha + eps));
wr = boxcar(N);

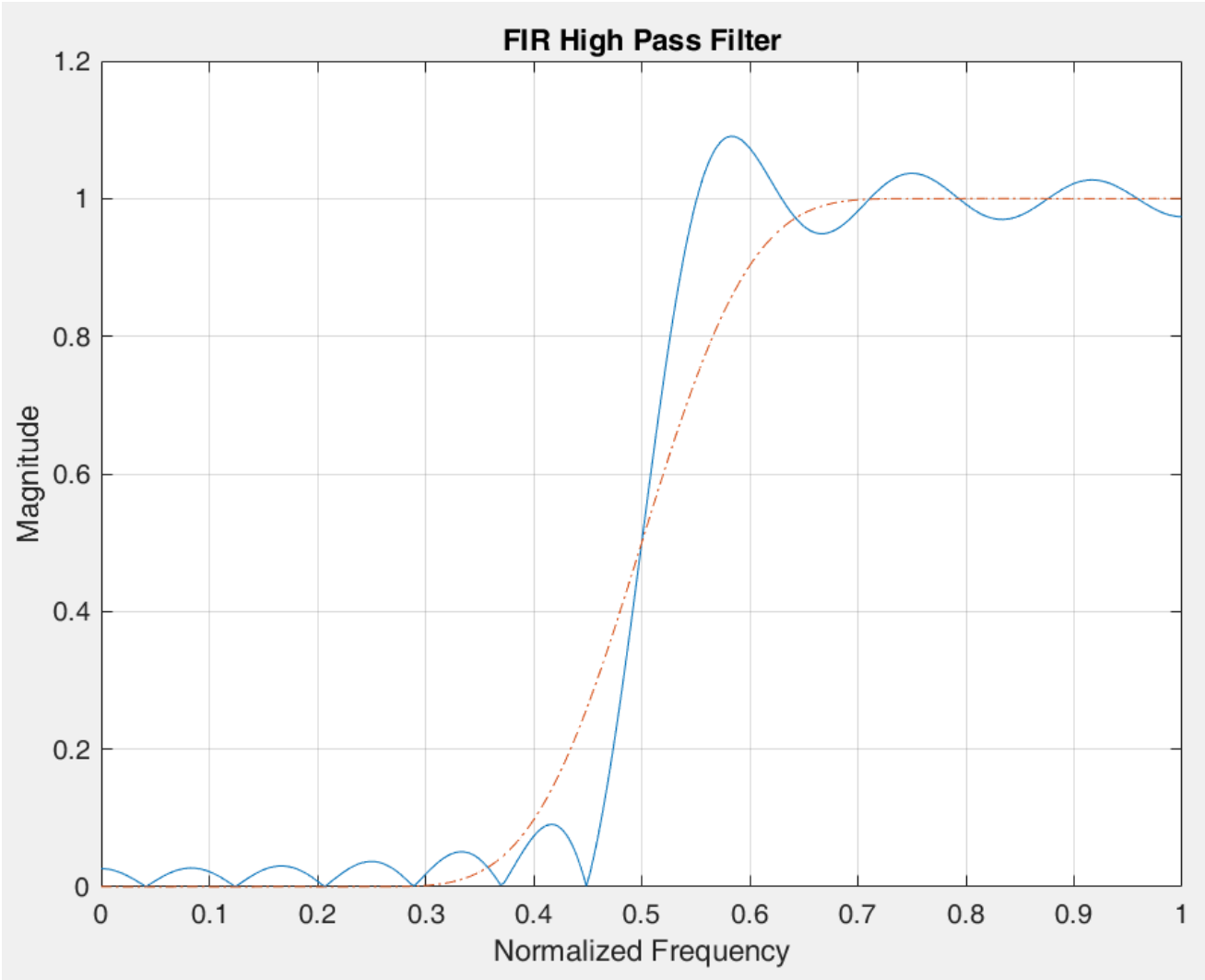
% Apply boxcar window
hn = hd .* wr';
w = 0:0.01:pi;
h = freqz(hn, 1, w);

% Plot frequency response
plot(w / pi, abs(h));
hold on;

% Apply Blackman window
wb = blackman(N);
hn = hd .* wb';
w = 0:0.01:pi;
h = freqz(hn, 1, w);
plot(w / pi, abs(h), '-.');
grid;
title('FIR High Pass Filter');
ylabel('Magnitude');
xlabel('Normalized Frequency');
hold off;
```

**OUTPUT:**

**HIGH PASS FILTER:**





**CODING:**  
**BAND STOP FILTER:**

```
clc;
wc1 = 0.25 * pi;
wc2 = 0.75 * pi;
N = 25;
alpha = (N - 1) / 2;
eps = 0.001;
n = 0:1:N-1;

% Ideal band stop filter
hd = (sin(wc1 * (n - alpha + eps)) - sin(wc2 * (n - alpha + eps)) + sin(pi * (n - alpha + eps))) ./
(pi * (n - alpha + eps));
wr = boxcar(N);

% Apply boxcar window
hn = hd .* wr';
w = 0:0.01:pi;
h = freqz(hn, 1, w);

% Plot frequency response
plot(w / pi, abs(h));
hold on;

% Apply Hamming window
wh = hamming(N);
hn = hd .* wh';
w = 0:0.01:pi;
h = freqz(hn, 1, w);
plot(w / pi, abs(h), '-.');

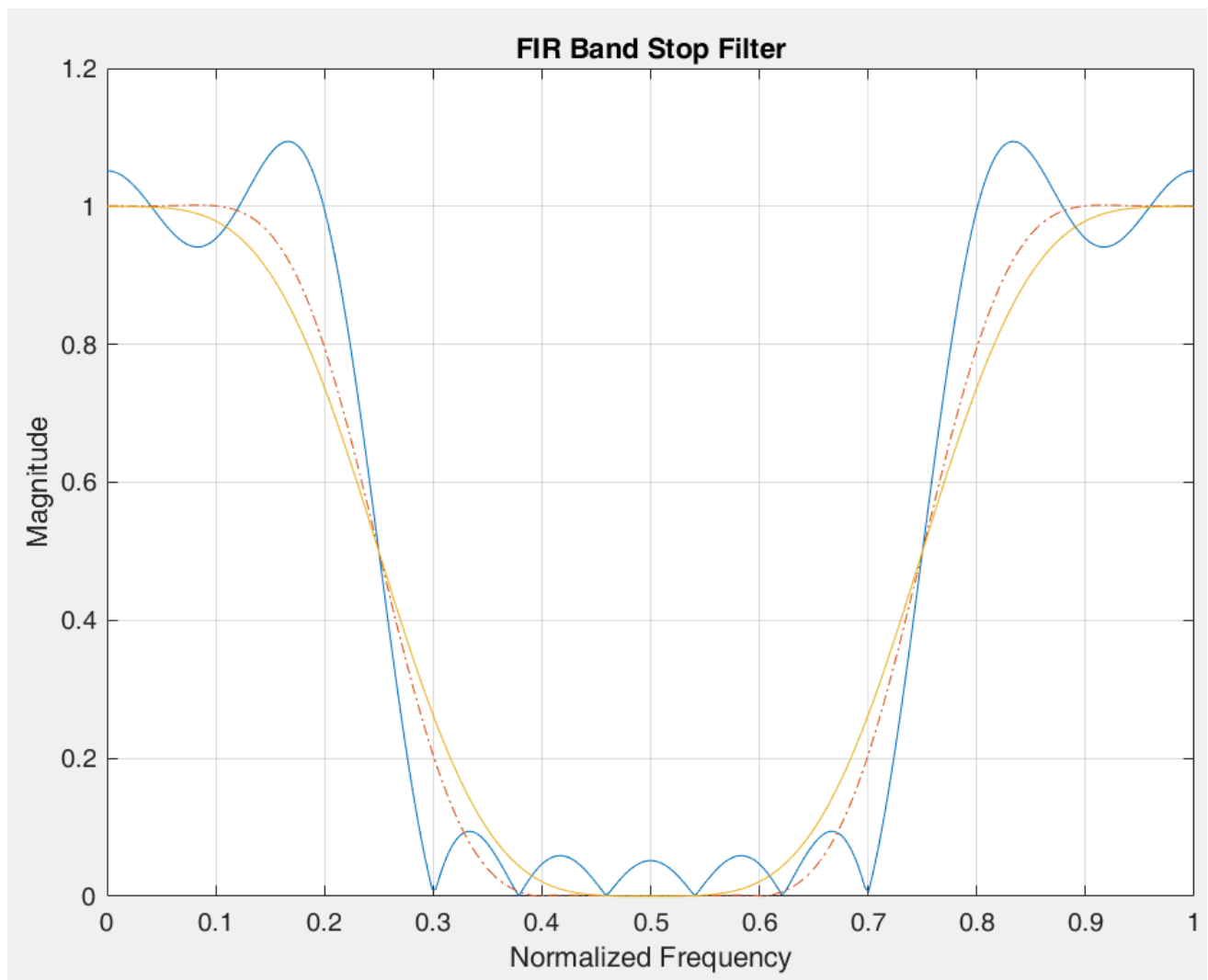
% Apply Blackman window
wb = blackman(N);
hn = hd .* wb';
w = 0:0.01:pi;
h = freqz(hn, 1, w);
plot(w / pi, abs(h));
grid;

title('FIR Band Stop Filter');
```

```
ylabel('Magnitude');  
xlabel('Normalized Frequency');  
hold off;
```

**OUTPUT:**

**BAND STOP FILTER:**



**CODING:**  
**BAND PASS FILTER:**

```
clc;
wc1 = 0.25 * pi;
wc2 = 0.75 * pi;
N = 25;
alpha = (N - 1) / 2;
eps = 0.001;
n = 0:1:N-1;

% Ideal band pass filter
hd = (sin(wc2 * (n - alpha + eps)) - sin(wc1 * (n - alpha + eps))) ./ (pi * (n - alpha + eps));
wr = boxcar(N);

% Apply boxcar window
hn = hd .* wr';
w = 0:0.01:pi;
h = freqz(hn, 1, w);

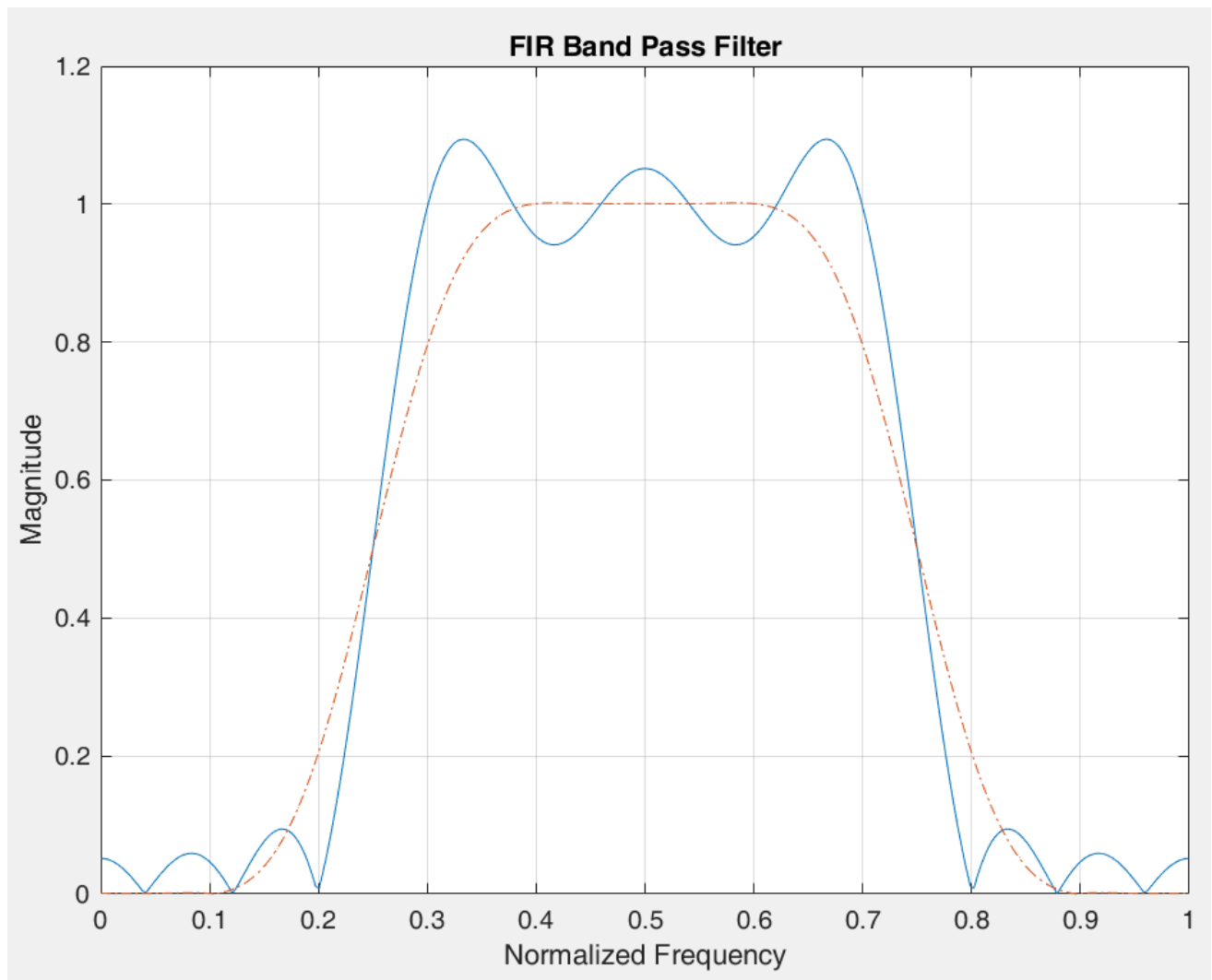
% Plot frequency response
plot(w / pi, abs(h));
hold on;

% Apply Hamming window
wh = hamming(N);
hn = hd .* wh';
w = 0:0.01:pi;
h = freqz(hn, 1, w);
plot(w / pi, abs(h), '-.');
grid;

title('FIR Band Pass Filter');
ylabel('Magnitude');
xlabel('Normalized Frequency');
hold off;
```

**OUTPUT:**

### **BAND PASS FILTER**



**RESULT:**

Thus FIR filters have been designed using window technique.

EX.NO :17

## TIME AND FREQUENCY RESPONSE OF A SIGNAL

DATE :

### AIM:

To perform time and frequency response of a signal.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step 1: Clear the command window and workspace.
- Step 2: Define the system using the coefficients of the numerator b and the denominator a for the transfer function.
- Step 3: Compute the frequency response using the freqz function for the given system.
- Step 4: Plot the magnitude response on the first subplot.
- Step 5: Calculate and plot the phase response on the second subplot.
- Step 6: Define the time-domain input signals x and y.
- Step 7: Compute the convolution of the two signals using the conv function to get the time-domain response.
- Step 8: Plot the time response in the third subplot.
- Step 9: Display the result.

### CODING:

```
clc;
clear all;

% Frequency Response
b = [1, 1];
a = [1, 2, 1];

[h, w] = freqz(b, a);

% Magnitude Response
subplot(2, 2, 1);
mag = 20*log10(abs(h));
plot(w/pi, mag);
xlabel('Normalized Frequency (\times \pi rad/sample)');
ylabel('Magnitude (dB)');
title('Magnitude Response');
grid on;

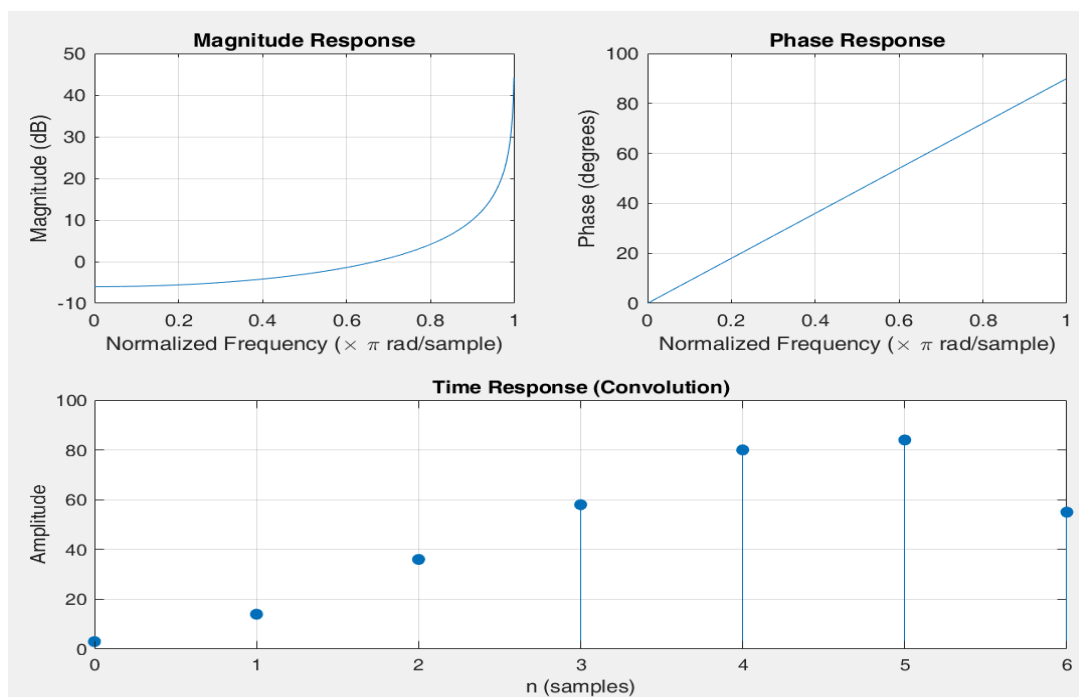
% Phase Response
```

```

subplot(2, 2, 2);
phase = angle(h) * 180 / pi;
plot(w/pi, phase);
xlabel('Normalized Frequency (\times \pi rad/sample)');
ylabel('Phase (degrees)');
title('Phase Response');
grid on;
% Time Response (Convolution)
x = [1, 2, 3, 4, 5];
y = [3, 8, 11];
z = conv(x, y);
% Plot Time Response
subplot(2, 1, 2);
stem(0:length(z)-1, z, 'filled');
xlabel('n (samples)');
ylabel('Amplitude');
title('Time Response (Convolution)');
grid on;

```

### OUTPUT:



### RESULT:

Thus, the time and frequency response of the signal have been successfully plotted.

EX.NO :18

## REMOVE NOISE FROM A SIGNAL

DATE :

### AIM:

To perform removal noise from a signal.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step 1: Clear the command window and workspace.
- Step 2: Generate a clean signal, such as a sine wave.
- Step 3: Introduce random noise to the clean signal.
- Step 4: Design a low-pass Butterworth filter to remove high-frequency noise.
- Step 5: Apply the filter to the noisy signal using the filter function.
- Step 6: Plot the original, noisy, and filtered signals for comparison.
- Step 7: Display the noise-free output signal.

### CODING:

```
clc;
clear;

% Generate a clean signal (sine wave)
t = 0:0.001:1;
clean_signal = sin(2*pi*10*t);

% Add noise to the signal (random noise)
noisy_signal = clean_signal + 0.5*randn(size(t));

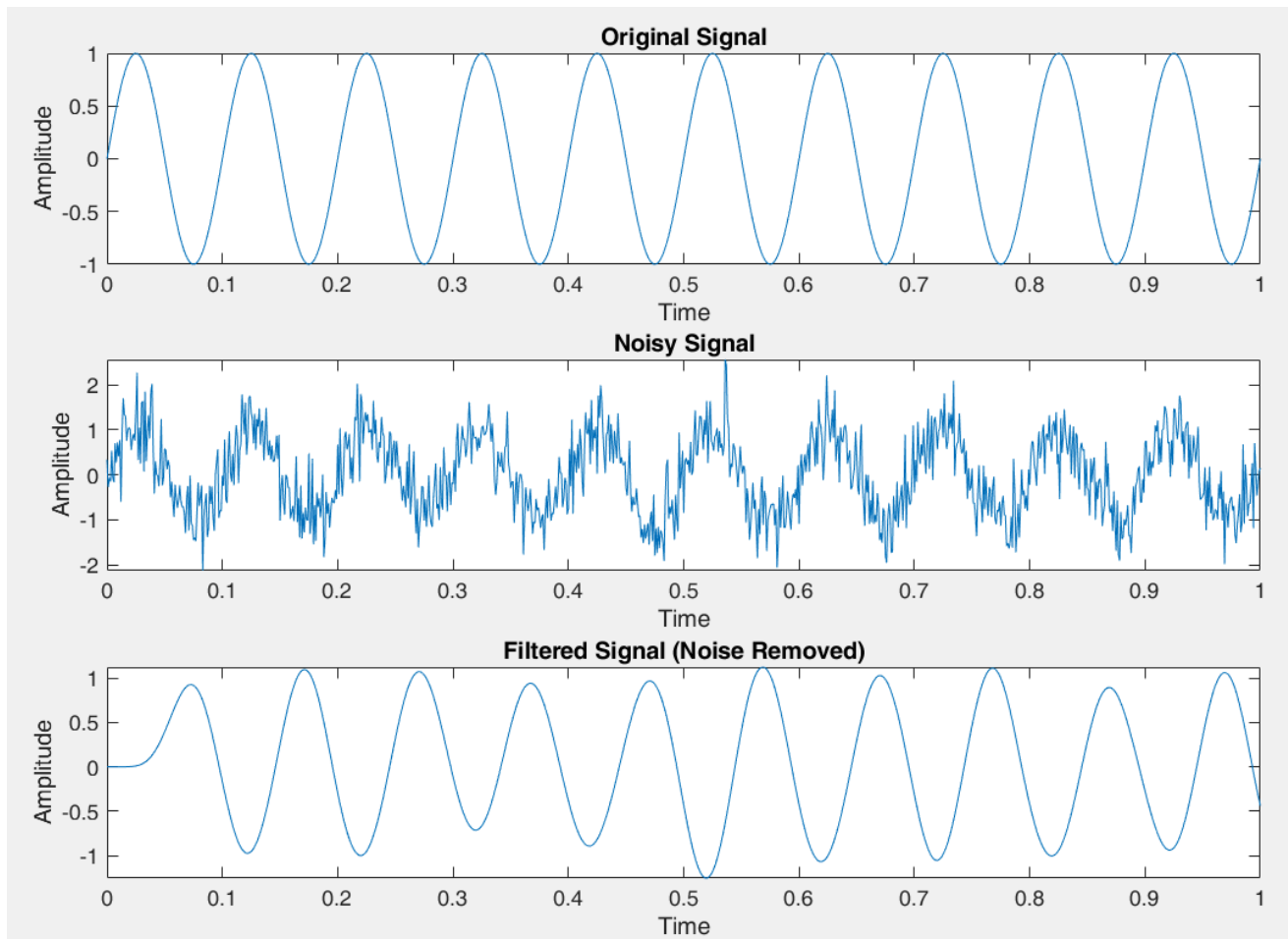
% Apply a low pass filter to remove noise
fc = 15; % Cutoff frequency
[b,a] = butter(6, fc/(0.5/0.001)); % 6th order Butterworth filter
filtered_signal = filter(b, a, noisy_signal);

subplot(3,1,1);
plot(t, clean_signal);
title('Original Signal');
xlabel('Time');
ylabel('Amplitude');

subplot(3,1,2);
plot(t, noisy_signal);
```

```
title('Noisy Signal');  
xlabel('Time');  
ylabel('Amplitude');  
subplot(3,1,3);  
plot(t, filtered_signal);  
title('Filtered Signal (Noise Removed)');  
xlabel('Time');  
ylabel('Amplitude');
```

### OUTPUT:



### RESULT:

Thus, noise has been successfully removed from the signal using a low-pass filter, and the noise-free output signal is displayed



EX.NO :19

## DESIGN A BANDPASS AND BANDSTOP FINITE IMPULSE RESPONSE FILTER

DATE :

### AIM:

To design a bandpass and bandstop Finite Impulse Response filter.

**SOFTWARE:**MATLAB

### ALGORITHM:

Step 1: Clear the command window and workspace.

Step 2: Define the sampling frequency and cutoff frequencies for both the bandpass and bandstop filters.

Step 3: Specify the filter order (N).

Step 4: Design the bandpass filter, bandstop filter using the fir1 function with a Hamming window.

Step 5: Compute the frequency response of the bandpass and bandstop filters using the freqz function.

Step 6: Plot the frequency response for both filters.

Step 7: Display the filter coefficients for both bandpass and bandstop filters.

### CODING:

```
clc;
clear;

% Parameters
fs = 1000;
fc1 = 100;
fc2 = 200;
fc3 = 300;
fc4 = 400;
N = 50;

% Bandpass Filter Design using Hamming Window
bp = fir1(N, [fc1 fc2] / (fs/2), 'bandpass', hamming(N + 1));
[h_bp, w_bp] = freqz(bp, 1, 512);

% Bandstop Filter Design using Hamming Window
bs = fir1(N, [fc3 fc4] / (fs/2), 'stop', hamming(N + 1));
[h_bs, w_bs] = freqz(bs, 1, 512);

% Plot Bandpass Filter Response
```

```

subplot(2, 2, 1);
plot(w_bp/pi, 20*log10(abs(h_bp)));
title('Bandpass Filter Frequency Response');
xlabel('Normalized Frequency (\times \pi rad/sample)');
ylabel('Magnitude (dB)');
grid on;

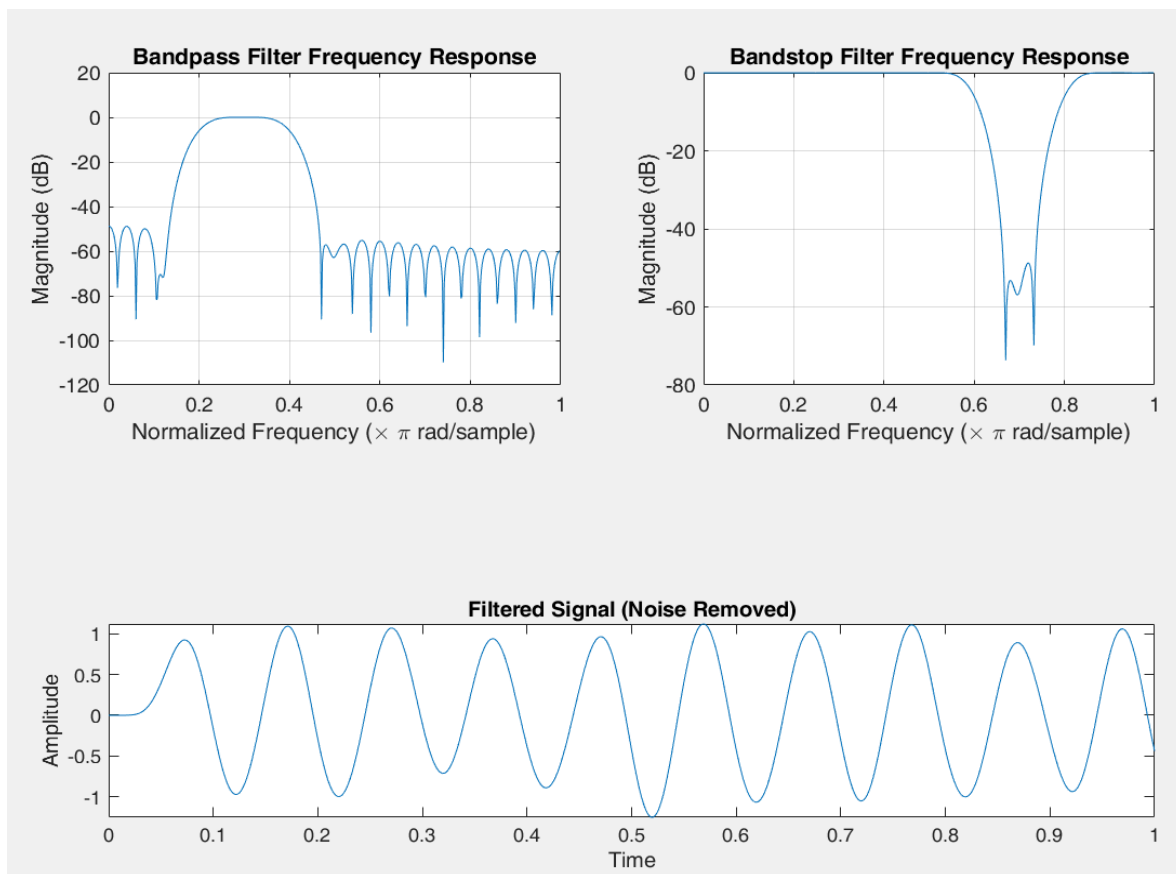
```

```

% Plot Bandstop Filter Response
subplot(2, 2, 2);
plot(w_bs/pi, 20*log10(abs(h_bs)));
title('Bandstop Filter Frequency Response');
xlabel('Normalized Frequency (\times \pi rad/sample)');
ylabel('Magnitude (dB)');
grid on;

```

### OUTPUT:



### RESULT:

Thus, the bandpass and bandstop Finite Impulse Response (FIR) filters have been successfully designed

EX.NO :20

## DESIGN FIR FILTER USING RECTANGULAR WINDOW

DATE :

### AIM:

To design FIR filter using rectangular window.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step 1: Clear the command window and workspace.
- Step 2: Define the sampling frequency (fs), cutoff frequency (fc), and filter order (N).
- Step 3: Create a rectangular window using the rectwin function.
- Step 4: Design the FIR lowpass filter using the fir1 function with the rectangular window.
- Step 5: Compute the frequency response of the filter using the freqz function.
- Step 6: Plot the magnitude response of the filter.
- Step 7: Plot the filter coefficients.
- Step 8: Display the filter coefficients in the command window.

### CODING:

```
clc;
clear;

% Parameters
fs = 1000;
fc = 200;
N = 50;

% FIR Lowpass Filter Design using Rectangular Window
rect_window = rectwin(N+1);
b = fir1(N, fc/(fs/2), 'low', rect_window);
[h, w] = freqz(b, 1, 512);

% Plot the Frequency Response
subplot(2, 1, 1);
plot(w/pi, abs(h));
title('FIR Lowpass Filter using Rectangular Window');
xlabel('Normalized Frequency (\times \pi rad/sample)');
ylabel('Magnitude');
grid on;

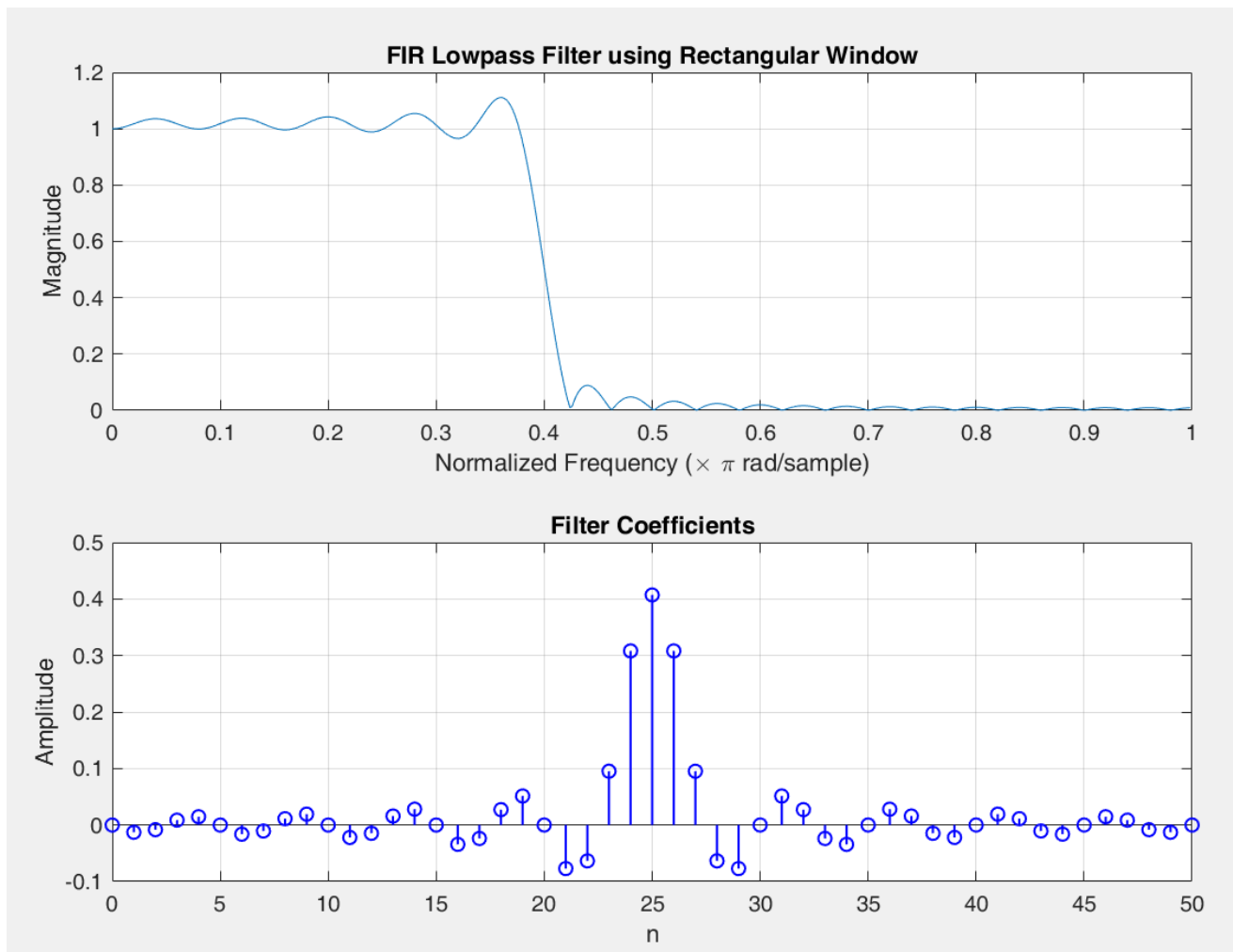
% Plot the Filter Coefficients
subplot(2, 1, 2);
```

```

stem(0:N, b, 'b', 'LineWidth', 0.75);
title('Filter Coefficients');
xlabel('n');
ylabel('Amplitude');
grid on;

```

## OUTPUT:



## RESULT:

Thus, the FIR lowpass filter has been successfully designed using the rectangular window.

EX.NO :21

## DESIGN FIR FILTER USING HAMMING WINDOW

DATE :

### AIM:

To design FIR filter using Hamming window.

### ALGORITHM:

- Step 1: Clear the command window and workspace.
- Step 2: Define the sampling frequency (fs), cutoff frequency (fc), and filter order (N).
- Step 3: Create a Hamming window using the hamming function.
- Step 4: Design the FIR lowpass filter using the fir1 function with the Hamming window.
- Step 5: Compute the frequency response of the filter using the freqz function.
- Step 6: Plot the magnitude response of the filter.
- Step 7: Plot the filter coefficients.
- Step 8: Display the filter coefficients in the command window.

### CODING:

```
clc;
clear;

% Parameters
fs = 1000;
fc = 200;
N = 50;

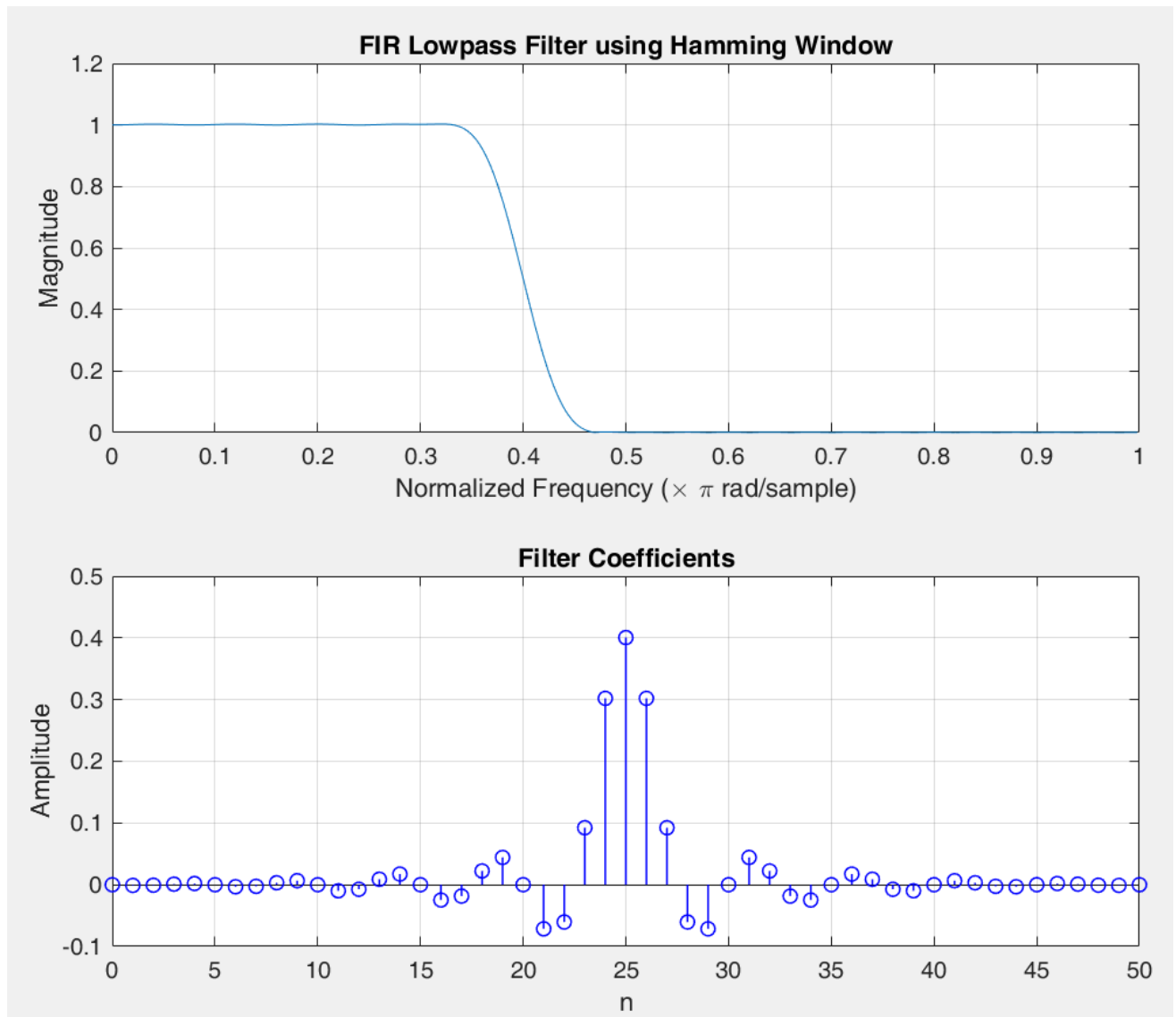
% FIR Lowpass Filter Design using Hamming Window
hamming_window = hamming(N+1);
b = fir1(N, fc/(fs/2), 'low', hamming_window);
[h, w] = freqz(b, 1, 512);

% Plot the Frequency Response
subplot(2, 1, 1);
plot(w/pi, abs(h));
title('FIR Lowpass Filter using Hamming Window');
xlabel('Normalized Frequency (\times \pi rad/sample)');
ylabel('Magnitude');
grid on;

% Plot the Filter Coefficients
subplot(2, 1, 2);
stem(0:N, b, 'b', 'LineWidth', 0.75);
title('Filter Coefficients');
```

```
xlabel('n');  
ylabel('Amplitude');  
grid on;
```

**OUTPUT:**



**RESULT:**

Thus, the FIR lowpass filter has been successfully designed using the Hamming window.

EX.NO :22

## DESIGN FIR FILTER USING HANNING WINDOW

DATE :

### AIM:

To design FIR filter using Hanning window.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step 1: Clear the command window and workspace.
- Step 2: Define the sampling frequency (fs), cutoff frequency (fc), and filter order (N).
- Step 3: Create a Hanning window using the hanning function.
- Step 4: Design the FIR lowpass filter using the fir1 function with the Hanning window.
- Step 5: Compute the frequency response of the filter using the freqz function.
- Step 6: Plot the magnitude response of the filter.
- Step 7: Plot the filter coefficients.
- Step 8: Display the filter coefficients in the command window.

### CODING:

```
clc;
clear;

% Parameters
fs = 1000;
fc = 200;
N = 50;

% FIR Lowpass Filter Design using Hanning Window
hanning_window = hanning(N+1);
b = fir1(N, fc/(fs/2), 'low', hanning_window);
[h, w] = freqz(b, 1, 512);

% Plot the Frequency Response
subplot(2, 1, 1);
plot(w/pi, abs(h));
title('FIR Lowpass Filter using Hanning Window');
xlabel('Normalized Frequency (\times \pi rad/sample)');
ylabel('Magnitude');
grid on;

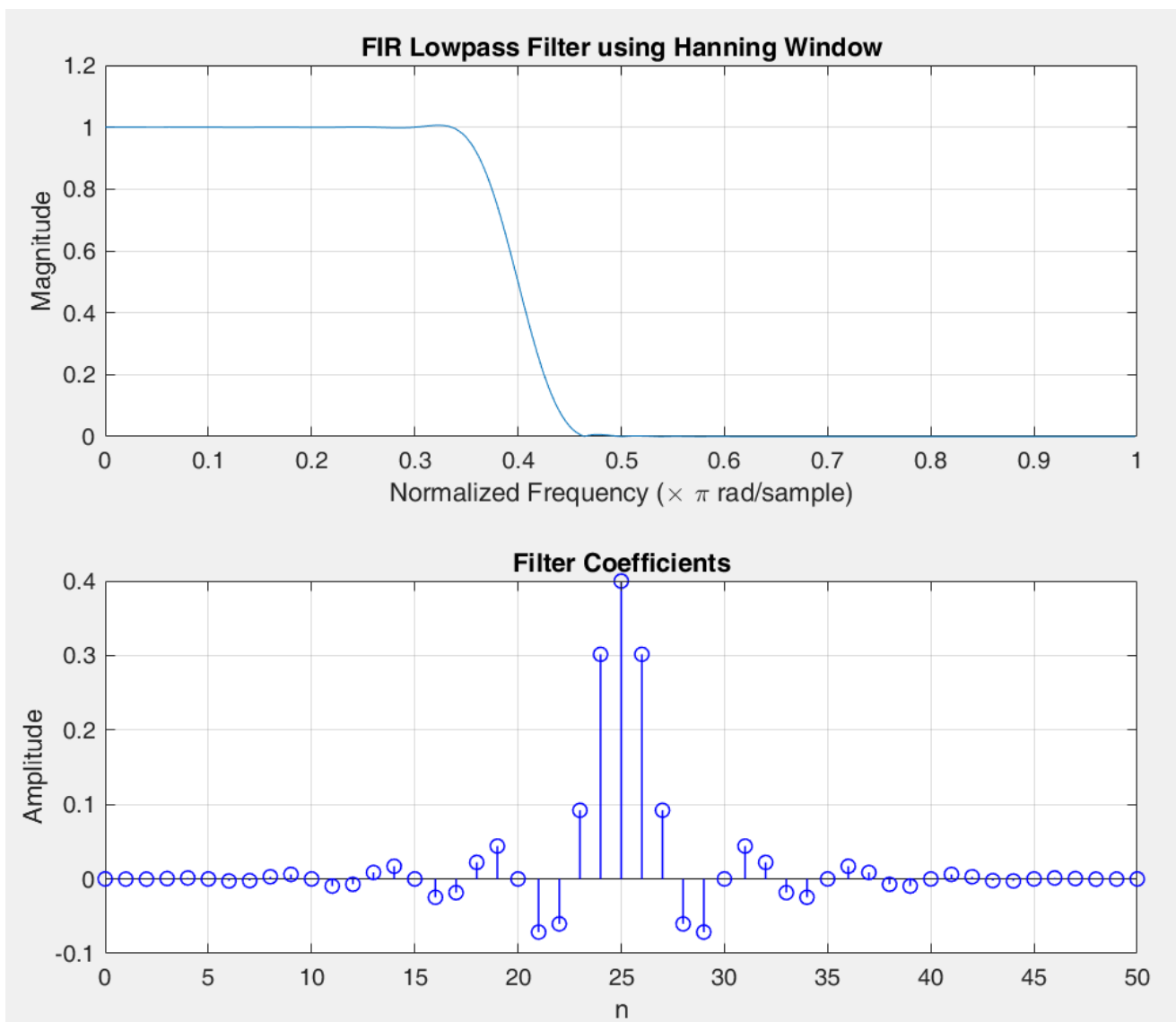
% Plot the Filter Coefficients
subplot(2, 1, 2);
```

```

stem(0:N, b,'b', 'LineWidth', 0.75);
title('Filter Coefficients');
xlabel('n');
ylabel('Amplitude');
grid on;

```

## OUTPUT:



## RESULT:

Thus, the FIR lowpass filter has been successfully designed using the Hanning window.



EX.NO :23

## DESIGN OF LOWPASS FIR FILTER USING RECTANGULAR WINDOW

DATE :

### AIM:

To design lowpass FIR filters using rectangular.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step 1: Clear the command window and workspace.
- Step 2: Define the sampling frequency (fs), cutoff frequency (fc), and filter order (N).
- Step 3: Create a rectangular window using the rectwin function.
- Step 4: Design the FIR lowpass filter using the fir1 function with the rectangular window.
- Step 5: Compute the frequency response of the filter using the freqz function.
- Step 6: Plot the magnitude response of the filter.
- Step 7: Plot the filter coefficients.
- Step 8: Display the filter coefficients in the command window.

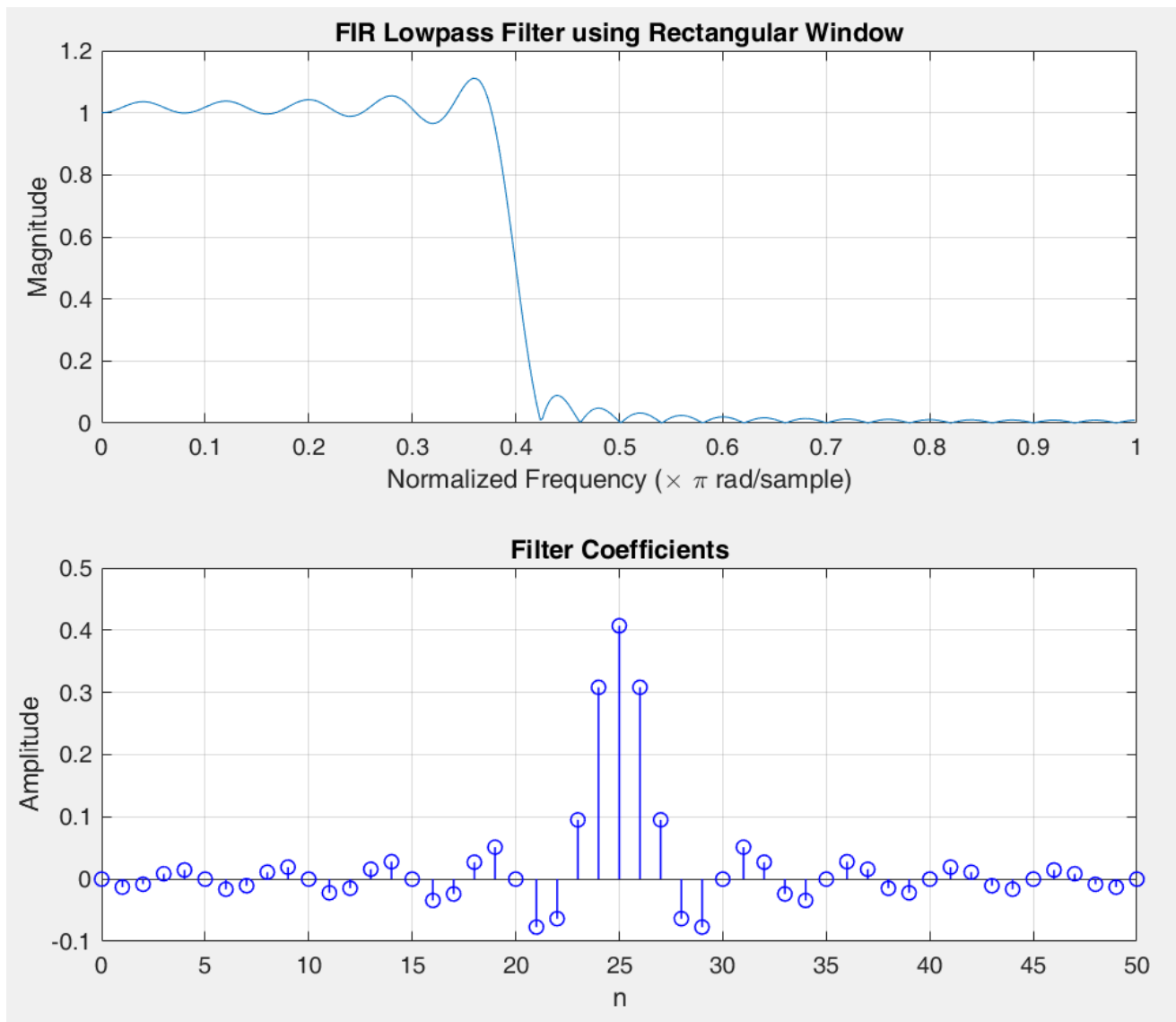
### CODING:

```
clc;
clear;
% Parameters
fs = 1000;
fc = 200;
N = 50;
% FIR Lowpass Filter Design using Rectangular Window
rectangular_window = rectwin(N+1);
b = fir1(N, fc/(fs/2), 'low', rectangular_window);
[h, w] = freqz(b, 1, 512);
% Plot the Frequency Response
subplot(2, 1, 1);
plot(w/pi, abs(h));
title('FIR Lowpass Filter using Rectangular Window');
xlabel('Normalized Frequency (\times \pi rad/sample)');
ylabel('Magnitude');
grid on;
% Plot the Filter Coefficients
subplot(2, 1, 2);
stem(0:N, b, 'b', 'LineWidth', 0.75);
title('Filter Coefficients');
xlabel('n');
```

```
ylabel('Amplitude');
```

```
grid on;
```

**OUTPUT:**



**RESULT:**

Thus, the FIR lowpass filter has been successfully designed using the rectangular window

EX.NO :24

## DESIGN A LOWPASS AND HIGHPASS INFINITE IMPULSE RESPONSE FILTER

DATE :

### AIM:

To design a lowpass and highpass Infinite Impulse Response filter.

**SOFTWARE:**MATLAB

### ALGORITHM:

- Step 1: Clear the command window and workspace.
- Step 2: Define the sampling frequency (fs), cutoff frequencies for lowpass (fc\_low) and highpass (fc\_high), passband ripple (Rp), and stopband attenuation (Rs).
- Step 3: Design the lowpass IIR filter using the Butterworth method with buttord to find the order and cutoff frequency, followed by butter to obtain the filter coefficients.
- Step 4: Design the highpass IIR filter similarly using the Butterworth method.
- Step 5: Compute the frequency response of the lowpass filter using freqz.
- Step 6: Plot the magnitude response of the lowpass filter.
- Step 7: Compute the frequency response of the highpass filter using freqz.
- Step 8: Plot the magnitude response of the highpass filter.
- Step 9: Display the filter coefficients for both lowpass and highpass filters in the command window.

### CODING:

```
clc;
clear;

% Parameters
fs = 1000; % Sampling frequency in Hz
fc_low = 200; % Cutoff frequency for lowpass in Hz
fc_high = 300; % Cutoff frequency for highpass in Hz
Rp = 3; % Passband ripple (in dB)
Rs = 40; % Stopband attenuation (in dB)

% Design Lowpass IIR Filter using Butterworth
[n_low, wn_low] = buttord(fc_low/(fs/2), (fc_low+50)/(fs/2), Rp, Rs);
[b_low, a_low] = butter(n_low, wn_low, 'low');

% Design Highpass IIR Filter using Butterworth
[n_high, wn_high] = buttord(fc_high/(fs/2), (fc_high-50)/(fs/2), Rp, Rs);
[b_high, a_high] = butter(n_high, wn_high, 'high');

% Frequency Response of Lowpass Filter
[H_low, W_low] = freqz(b_low, a_low, 512);
```

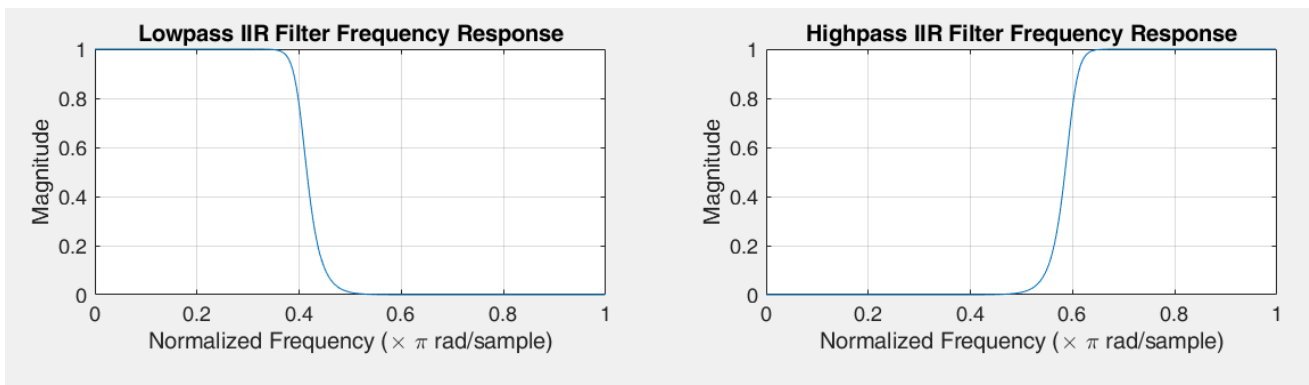
```

subplot(2, 2, 1);
plot(W_low/pi, abs(H_low));
title('Lowpass IIR Filter Frequency Response');
xlabel('Normalized Frequency (\times \pi rad/sample)');
ylabel('Magnitude');
grid on;

% Frequency Response of Highpass Filter
[H_high, W_high] = freqz(b_high, a_high, 512);
subplot(2, 2, 2);
plot(W_high/pi, abs(H_high));
title('Highpass IIR Filter Frequency Response');
xlabel('Normalized Frequency (\times \pi rad/sample)');
ylabel('Magnitude');
grid on;

```

## OUTPUT:



## RESULT:

Thus, the lowpass and highpass IIR filters have been successfully designed.

EX.NO :25

## EQUIVALENT IMPULSE RESPONSE OF THE NARROWBAND FILTER

DATE :

### AIM:

To find the equivalent impulse response of the narrowband filter.

**SOFTWARE:**MATLAB

### ALGORITHM:

Step 1: Clear the command window and workspace.

Step 2: Define the sampling frequency, center frequency, and number of filter taps.

Step 3: Compute the ideal impulse response and apply a Hamming window.

Step 4: Calculate and plot the magnitude and phase response of the filter.

Step 5: Create an impulse signal, convolve with filter coefficients, and plot the equivalent impulse response.

Step 6: Display the equivalent impulse response in the command window.

### CODING:

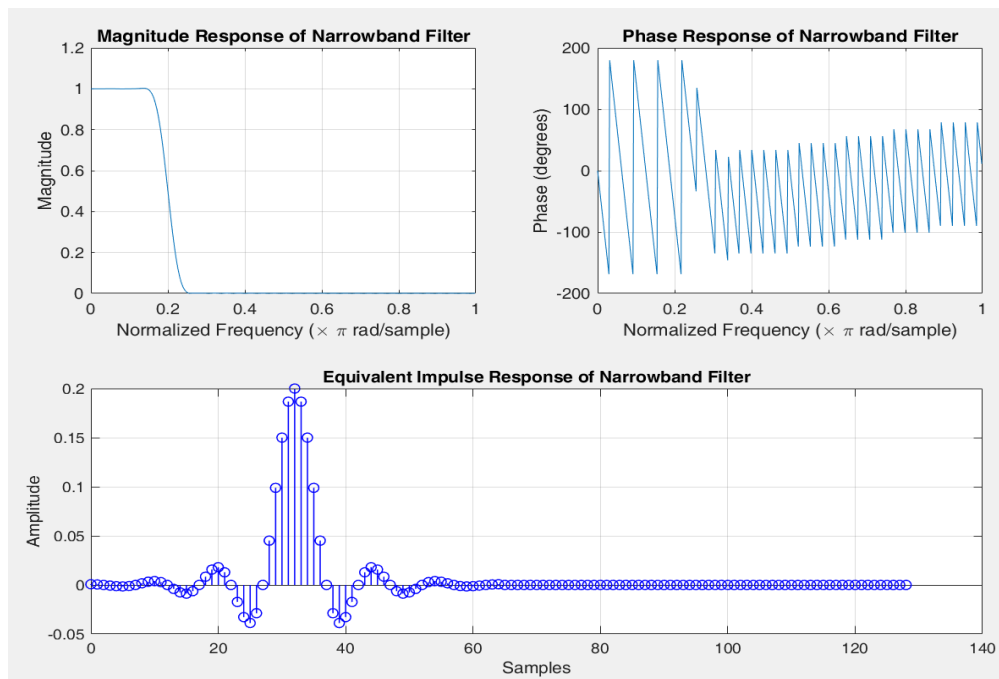
```
clc;
clear;
% Define filter parameters
fs = 1000;
fc = 100;
bw = 20;
N = 64;
% Ensure N is odd for correct indexing
if mod(N, 2) == 0
    N = N + 1;
end
% Design the narrowband filter using a Hamming window
n = 0:N-1;
hd = (sin(2*pi*fc/fs*(n - (N-1)/2)))/(pi*(n - (N-1)/2));
hd((N-1)/2 + 1) = 2*fc/fs;
w = hamming(N)';
hn = hd .* w;
% Frequency response of the filter
[H, W] = freqz(hn, 1, 512);
magnitude = abs(H);
phase = angle(H) * 180 / pi;
% Plot results
subplot(2, 2, 1);
```

```

plot(W/pi, magnitude);
title('Magnitude Response of Narrowband Filter');
xlabel('Normalized Frequency (\times \pi rad/sample)');
ylabel('Magnitude');
grid on;
subplot(2, 2, 2);
plot(W/pi, phase);
title('Phase Response of Narrowband Filter');
xlabel('Normalized Frequency (\times \pi rad/sample)');
ylabel('Phase (degrees)');
grid on;
% Impulse response
impulse = [1, zeros(1, N-1)];
response = conv(impulse, hn);
subplot(2, 1, 2);
stem(0:length(response)-1, response,'b', 'LineWidth', 0.75);
title('Equivalent Impulse Response of Narrowband Filter');
xlabel('Samples');
ylabel('Amplitude');
grid on;

```

## OUTPUT:



## RESULT:

Thus, the equivalent impulse response of the narrowband filter has been successfully calculated, along with its magnitude and phase responses.

EX.NO: 26

## PERFORM CROSS-CORRELATION

DATE :

### AIM:

To write a MATLAB program to generate correlation of two signals.

**SOFTWARE:**MATLAB

### ALGORITHM:

Step 1: Clear the Command Window and Workspace

Step 2: Start the program

Step 3: Generate the correlation of Signals

Step 4: Execute the program.

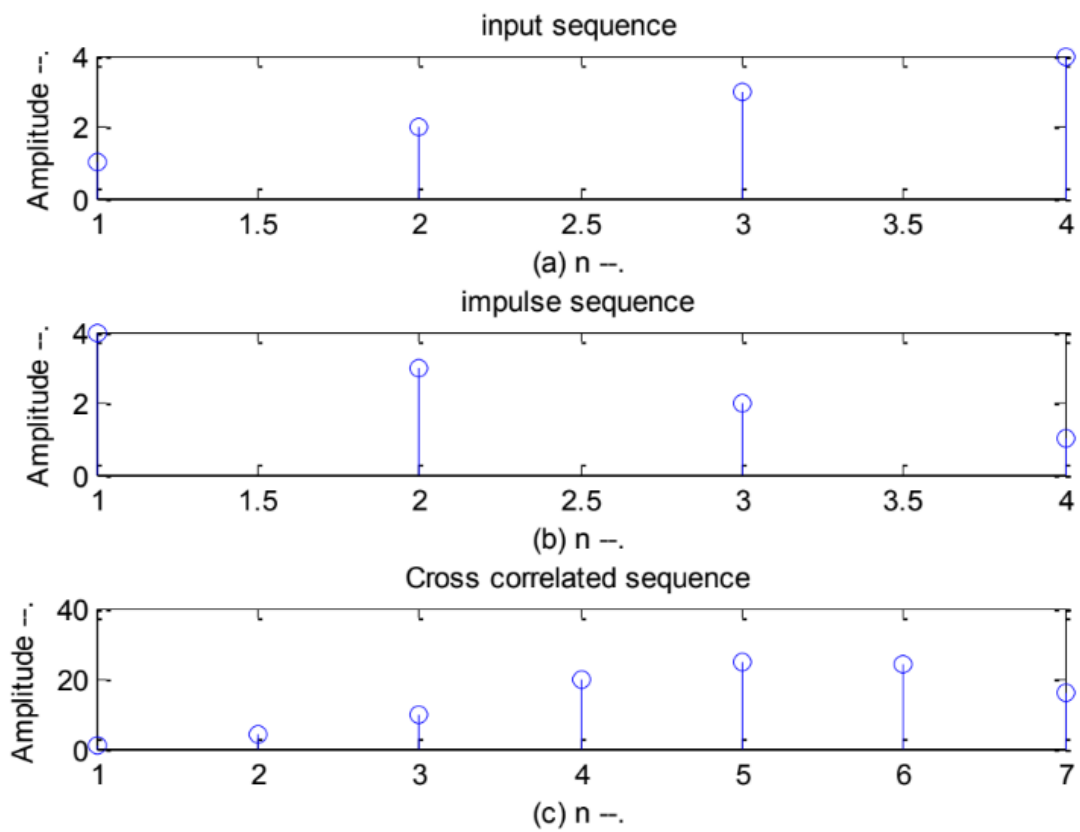
Step 5: Create Subplots to Visualize the Individual Signals and the Resultant Signal

Step 6: Label the Axes and Title for Each Subplot. Display the Grid for Better Visualization.

### CODING:

```
% Program for computing cross-correlation of the sequences x[1 2 3 4] and h[4 3 2 1]
clc;
clear all;
close all;
x=input('enter the 1st sequence');
h=input('enter the 2nd sequence');
y= xcorr(x,h);
subplot(3,1,1);
stem(x);
ylabel('Amplitude --. ');
xlabel('(a) n --. ');
title('input sequence');
subplot(3,1,2);
stem(h);
ylabel('Amplitude --. ');
xlabel('(b) n --. ');
title('impulse sequence');
subplot(3,1,3);
stem(y);
ylabel('Amplitude --. ');
xlabel('(c) n --. ');
title('Cross correlated sequence');
disp('The resultant signal is y=');
disp(y)
Output:
enter the 1st sequence[1 2 3 4]
enter the 2nd sequence[4 3 2 1]
The resultant signal is y=
1.0000 4.0000 10.0000 20.0000 25.0000 24.0000 16.0000
```

## OUTPUT:



## RESULT

Thus a program to generate sequence for correlation signal using MATLAB was written and executed and the corresponding simulation output is plotted.



**EX NO: 27**

## **PERFORM QUANTIZATION USING MATLAB**

**DATE:**

**AIM:**

To perform quantization using matlab.

**SOFTWARE:MATLAB**

### **ALGORITHM:**

**STEP 1 :** Obtain the continuous signal to be quantized.

**STEP 2 :** Define the number of bits for quantization.

**STEP 3 ::** Compute the maximum and minimum values of the signal.

**STEP 4:** Scale the signal to fit within the quantization levels.

**STEP 5 :** Round the normalized values to the nearest quantization level. Convert the quantized values back to the original signal range.

**STEP 6 :** Return or plot the quantized signal for analysis.

### **CODING:**

```
% Define parameters
fs = 1000;           % Sampling frequency
t = 0:1/fs:1;       % Time vector
signal = sin(2*pi*50*t); % Example signal (sine wave)

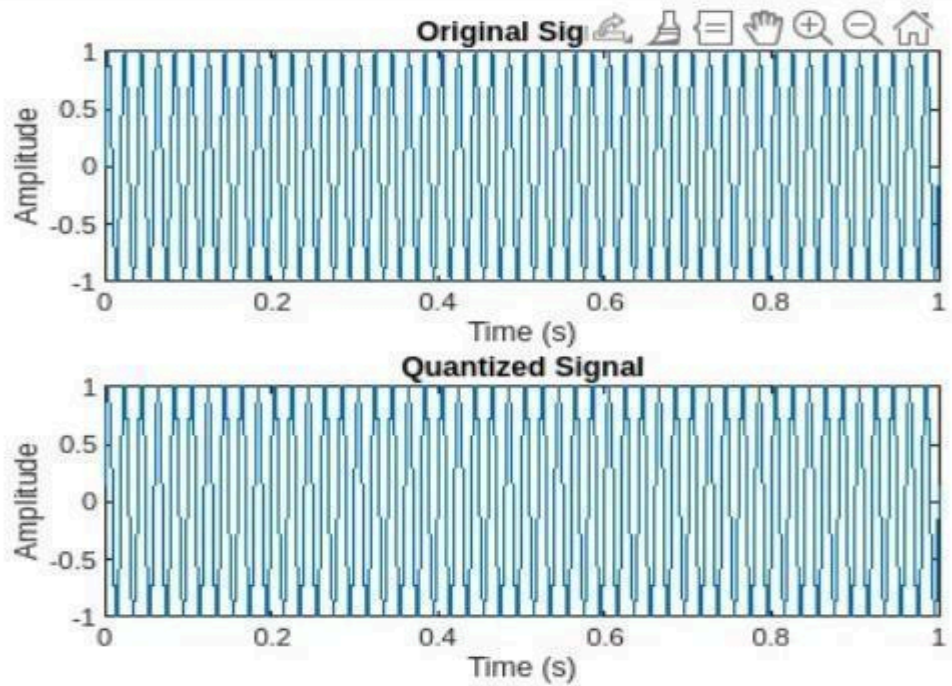
% Quantization parameters
nBits = 3;           % Number of bits for quantization
levels = 2^nBits;    % Number of quantization levels
maxValue = max(signal); % Maximum value of the signal
minValue = min(signal); % Minimum value of the signal

% Quantization process
quantizedSignal = round((signal - minValue) / (maxValue - minValue) * (levels - 1));
quantizedSignal = quantizedSignal * (maxValue - minValue) / (levels - 1) + minValue;

% Plot original and quantized signals
figure;
subplot(2,1,1);
plot(t, signal);
title('Original Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2,1,2);
plot(t, quantizedSignal);
title('Quantized Signal');
```

```
xlabel('Time (s)');  
ylabel('Amplitude');  
OUTPUT:
```



**RESULT:**

Thus, the quantization was verified successfully.

## **EX.NO :28    DESIGN OF HIGH PASS FIR FILTER USING RECTANGULAR WINDOW**

**DATE:**

**AIM:**

To perform design highpass FIR filter using rectangular window using matlab.

**SOFTWARE:MATLAB**

**ALGORITHM:**

Step 1: Set the sampling frequency, cut-off frequency, and filter order.

Step 2: Normalize the cut-off frequency with respect to the Nyquist frequency.

Step 3: Create the impulse response for a low pass filter and negate it to obtain the high pass response. Adjust the DC component.

Step 4: Since a rectangular window is used, the impulse response remains unchanged.

Step 5: Use the freqz function to compute the frequency response of the filter.

Step 6: Plot the magnitude response in decibels against frequency

**CODING:**

```
% Design parameters
fs = 1000;
fc = 100;
N = 20;
% Calculate the normalized cut-off frequency
Wn = fc / (fs / 2);

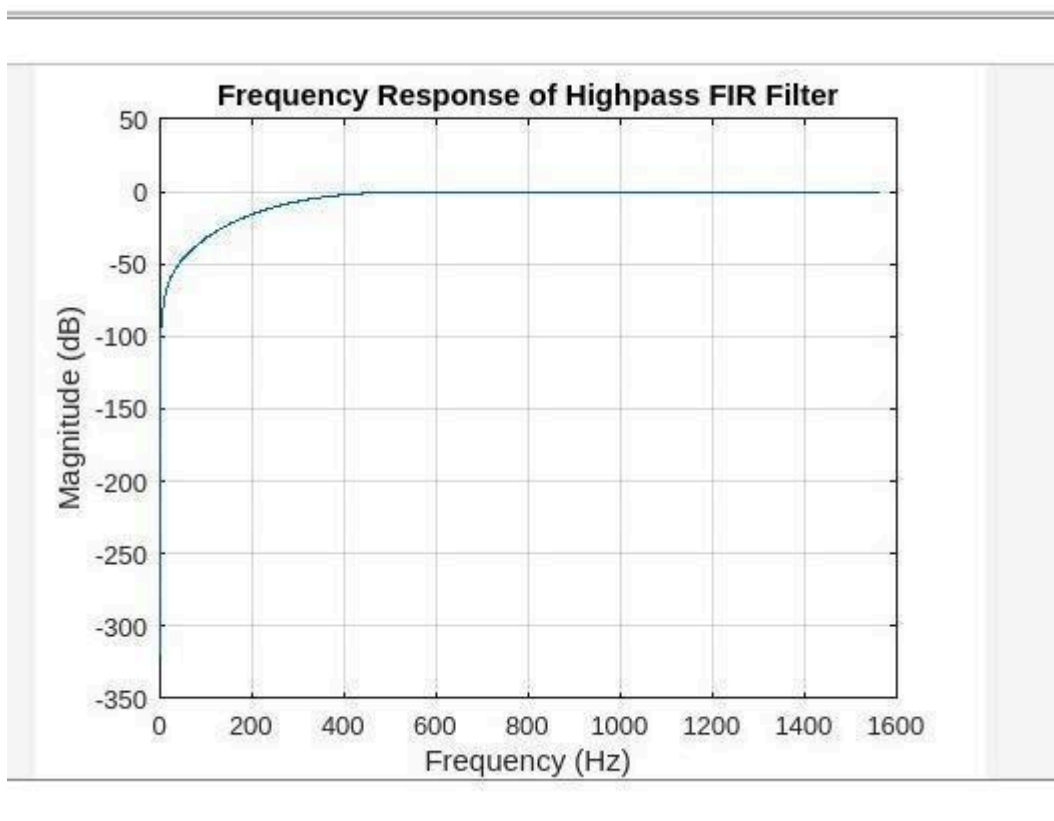
% Generate the ideal highpass filter impulse response
h_ideal = -fir1(N, Wn, 'low');
h_ideal(N/2 + 1) = h_ideal(N/2 + 1) + 1;

% Apply rectangular window (no additional windowing)
h_fir = h_ideal;
% Frequency response
[H, f] = freqz(h_fir, 1, 1024, 'half');

% Plot the frequency response
figure;
```

```
plot(f * fs / 2, 20*log10(abs(H)));  
title('Frequency Response of Highpass FIR Filter');  
xlabel('Frequency (Hz)');  
ylabel('Magnitude (dB)');  
grid on;
```

### OUTPUT:



### RESULT:

Thus, design highpass FIR filter using rectangular window hve been successfully Designed.

**EX.NO: 29      VERIFY CHANNEL CAPACITY THEOREM USING MATLAB**

DATE:

**AIM:**

To verify channel capacity theorem using matlab.

**ALGORITHM:**

Step 1: Set the bandwidth ( B ) of the channel in hertz (Hz),set the average signal power ( S ),set the average noise power ( N ).

Step 2 : Calculate Signal-to-Noise Ratio (SNR):

Step 3 : Compute the SNR using the formula: [  $\text{SNR} = \frac{S}{N}$  ]

Step 4 :Use the Shannon-Hartley theorem formula: [  $C = B \log_2(1 + \text{SNR})$  ]

Step 5:Print the calculated channel capacity ( C ) in bits per second (bps).

Step 6:Plot Capacity vs SNR,If desired, create a plot to visualize how channel capacity changes with varying SNR values.

**CODING:**

```
B = 1000;  
S = 10;  
N = 1;  
SNR = S / N;  
C = B * log2(1 + SNR);  
fprintf('Channel Capacity: %.2f bps\n', C)
```

**OUTPUT:**

ANSWER : 3459.43bps

**RESULT:**

Thus,the channel capacity theorem is verified successfully.

EX.No : 30

## RESPONSE OF ELLIPTIC FILTER

DATE :

AIM:

To generate noiseless elliptic filter response using MATLAB

### ALGORITHM:

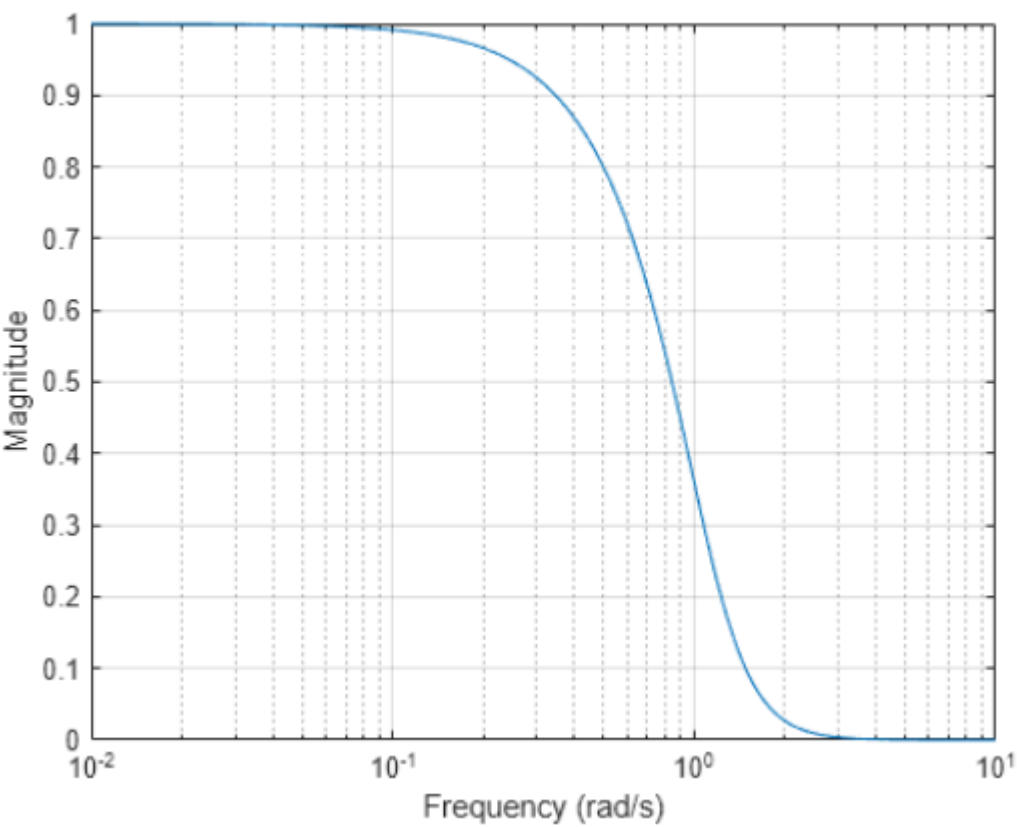
- Step-1: Clear the command window and workspace.
- Step-2: Declare the timescale and amplitude of the signal.
- Step-3: Using pre-defined functions generate the corresponding waveforms and plot them.
- Step-4: Label the axes and title.
- Step-5: Execute the program and view the output waveforms.

Elliptic filters are equiripple in both the passband and stopband. They generally meet filter requirements with the lowest order of any supported filter type. Given a filter order  $n$ , passband ripple  $R_p$  in decibels, and stopband ripple  $R_s$  in decibels, elliptic filters minimize transition width.  $|H(j\Omega)| = 10^{-R_p/20}$  at  $\Omega=1$ .

### CODING:

```
n = 5;  
Rp = 0.5;  
Rs = 20;  
[z,p,k] = ellipap(n,Rp,Rs);  
[bEp,aEp] = zp2tf(z,p,k);  
[hEp,wEp] = freqs(bEp,aEp,4096);  
semilogx(wEp,abs(hEp))  
grid on  
xlabel("Frequency (rad/s)")  
ylabel("Magnitude")
```

**OUTPUT:**



**RESULT:**

Thus, the interpolation and decimation process is executed successfully.

EX.NO : 31

**RANDOM NUMBER GENERATION**

**DATE:**

**AIM :**

To Write Matlab Code for Random

**Algorithm :**

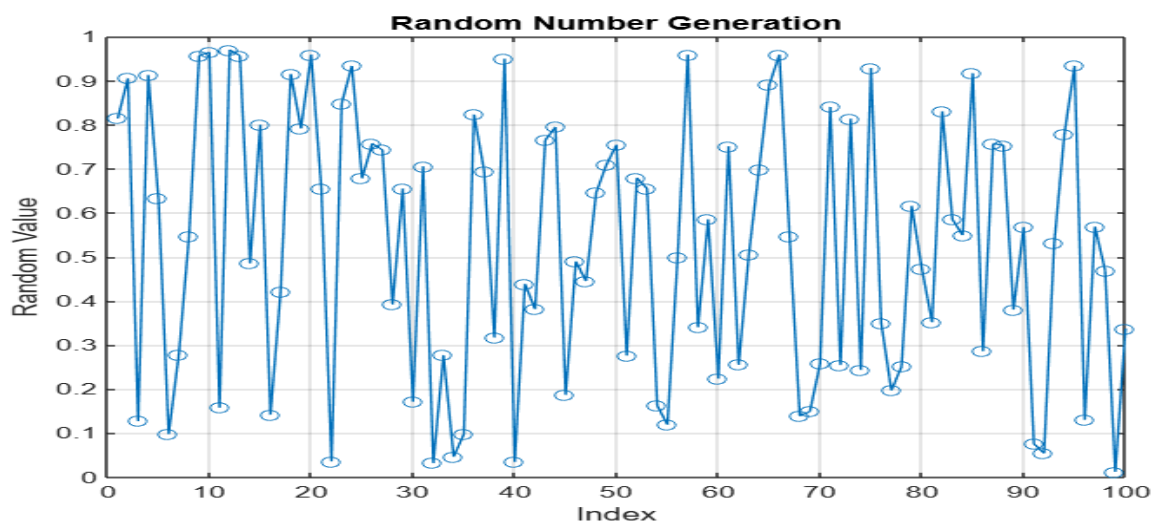
Step 1 : Use the rand, randi, or randn functions to create random numbers.

Step 2 : Use the plot function to visualize the generated numbers.

**Program :**

```
numPoints = 100;  
randomNumbers = rand(numPoints, 1);  
figure;  
plot(randomNumbers, 'o-');  
title('Random Number Generation');  
xlabel('Index');  
ylabel('Random Value');  
grid on;
```

**OUTPUT :**





**RESULT :**

Thus the MATLAB code for random numbers are successfully generated and plotted.

EX.NO :32                    **GENERATE THE IMPULSE SEQUENCE AT n0 SAMPLES**

**DATE:**                                    **LYING BETWEEN n1 AND n2**

**AIM :**

To Write MATLAB Code to generate the impulse sequence at n0 samples lying between n1 and n2.

**Algorithm :**

Step 1 : Specify the sample indices for the impulse and the range for the sequence.

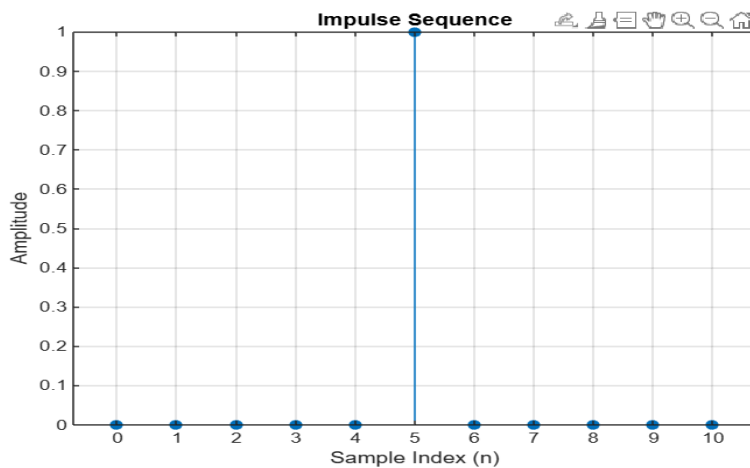
Step 2 : Initialize a sequence of zeros and set the value at the specified index to one.

Step 3 : Use the stem function to visualize the impulse.

**Program :**

```
n1 = 0;
n2 = 10;
n0 = 5;
n = n1:n2;
impulseSequence = zeros(size(n));
impulseSequence(n == n0) = 1;
figure;
stem(n, impulseSequence, 'filled');
title('Impulse Sequence');
xlabel('Sample Index (n)');
ylabel('Amplitude');
grid on;
```

**OUTPUT :**



## RESULT :

Thus the MATLAB code for impulse sequence at  $n_0$  samples lying between  $n_1$  and  $n_2$  has been successfully generated.

## EX.NO :33                      SIMPLE    MATHEMATICAL EXPRESSIONS

DATE:

AIM :

To Illustrate the simple mathematical expressions in MATLAB for i) $z=x/y$  and ii) $x=3*\text{sqrt}(5)-1, y=x^2$ .

### Algorithm :

Step 1 : Use the expression to compute the value of  $x$ .

Step 2 : Compute  $y$  as the square of  $x$ .

### Program :

1)

```
x = 10;
```

```
y = 2;
```

```
z = x / y;
```

```
fprintf('z = %.4f\n', z);
```

2)

```
x = 3 * sqrt(5) - 1;
```

```
y = x^2;
```

```
fprintf('x = %.4f\n', x);
```

```
fprintf('y = %.4f\n', y);
```

## OUTPUT :

ANSWER :

$$Z = 5.0000$$

EX.NO :34

**PLOT THE BASIC SIGNALS-IMPULSE,STEP FUNCTION AND RAMP  
FUNCTION TO CREATE 2-D AND 3-D PLOTS**

DATE :

X = 5.7082

Y = 32.5836

**RESULT :**

Thus the code has been successfully getting the output for the given simple mathematical expression

**AIM:**To generate the basic signals - Impulse, Step function and Ramp function to create 2-D and 3-D plots using MATLAB.

**ALGORITHM:**

- Step-1: Clear the command window and workspace.
- Step-2: Declare the time scale and amplitude of the signal.
- Step-3: Using pre-defined functions generate the corresponding waveforms and plot them.
- Step-4: Label the axes and title.
- Step-5: Execute the program and view the output waveforms.

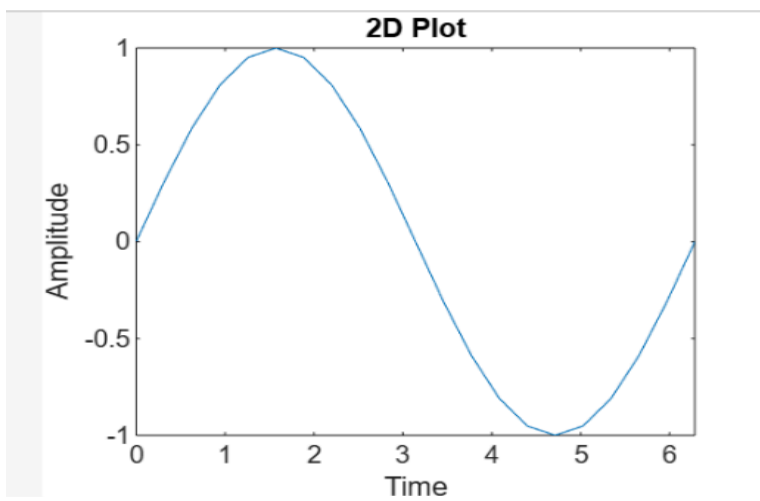
**CODING:**

```
% 2-D plot
clc;
clear ;
x = (0:pi/10:2*pi);
y = sin(x);
plot(x, y);
title('2D Plot');
xlabel('Time');
ylabel('Amplitude');

% 3-D plot
clc ;
clear;
t=(-4:0.01:4)
x=t.^2
```

```
y=4*t  
plot3(x,y,t)  
grid on  
xlabel('x-axis')  
ylabel('y-axis')  
zlabel('z-axis')  
title('3D Plot')
```

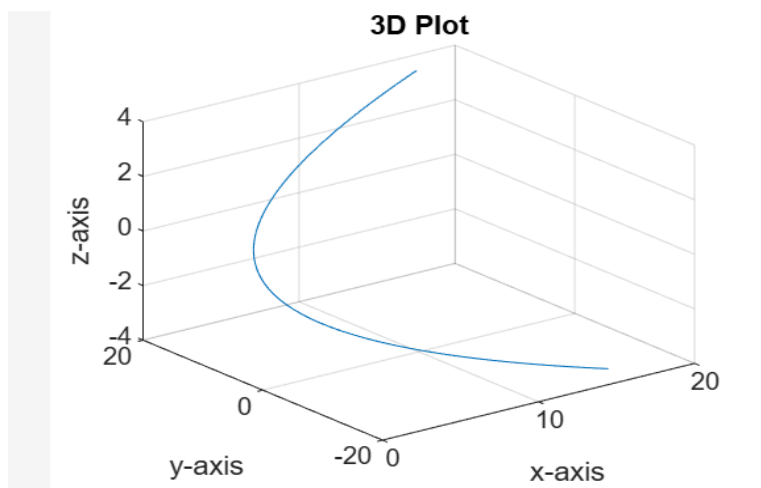
**OUTPUT:**



EX.NO :35

## MATLAB PROGRAM TO PERFORM DFT BY FFT

DATE :



### RESULT:

Thus the basic signals -Impulse, Step function and Ramp function To create 2-D and 3-D plots is generated using MATLAB.

### AIM:

To perform DFT by FFT using MATLAB.

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Declare the time scale and amplitude of the signal.
- Step-3: Using pre-defined functions generate the corresponding waveforms and plot them.
- Step-4: Label the axes and title.
- Step-5: Execute the program and view the output waveforms.

### CODING:

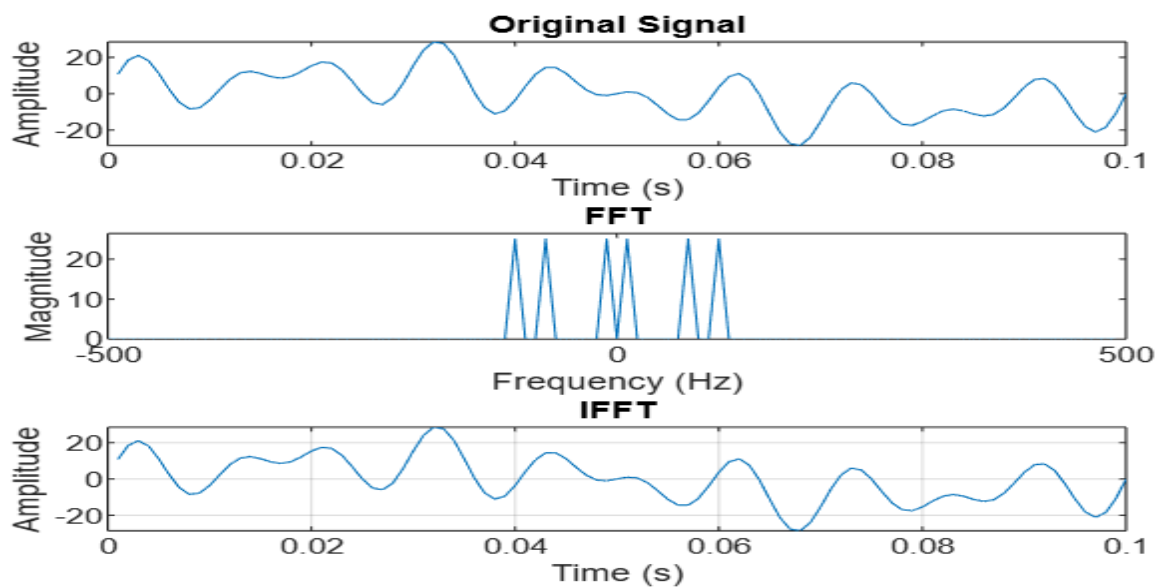
```
clc;  
clear all;  
close all;  
mm=[ ];
```

```

A=10;
fs=1000;
Ts=1/fs;
t=(1:100)*Ts;
y=(A*sin(2*pi*100*t))+(A*sin(2*pi*10*t))+(A*sin(2*pi*70*t));
figure(1)
plot(t,y)
N=length(t);
yy=fft(y);
yyy=fftshift(yy);
f=fs.*(-N/2:N/2-1)/N;
figure(2)
plot(f,(yyy.*conj(yyy))/(N*N));title('FFT')
y1=ifft(yy);
figure(3)
plot(t,y1),
grid on;
title('IFFT')

```

## OUTPUT:



DATE :

**RESULT:**

Thus the DFT by FFT is generated using MATLAB

**AIM:**

To generate triangular wave using MATLAB.

**ALGORITHM:**

- Step-1: Clear the command window and workspace.
- Step-2: Declare the time scale and amplitude of the signal.
- Step-3: Using pre-defined functions generate the corresponding waveforms and plot them.
- Step-4: Label the axes and title.
- Step-5: Execute the program and view the output waveforms.

**CODING:**

```
clc;  
clear ;  
close all;  
N = input('Enter the number of cycles: ');  
M = input('Enter the amplitude: ');
```

```

t1 = 0:0.5:M;
t2 = M-0.5:-0.5:0;
single_cycle_length = length(t1) + length(t2);
t = zeros(1, N * single_cycle_length);
index = 1;
for i = 1:N
    t(index:index + length(t1) - 1) = t1;
    index = index + length(t1);
    t(index:index + length(t2) - 1) = t2;
    index = index + length(t2);
end
subplot(2, 1, 1);
plot(t, 'LineWidth', 1.5);
grid on;
xlabel('--> time');
ylabel('--> amplitude');
title('Analog triangular signal');

subplot(2, 1, 2);
stem(t, 'filled');
grid on;
xlabel('--> time');
ylabel('--> amplitude');
title('Discrete triangular signal');

```

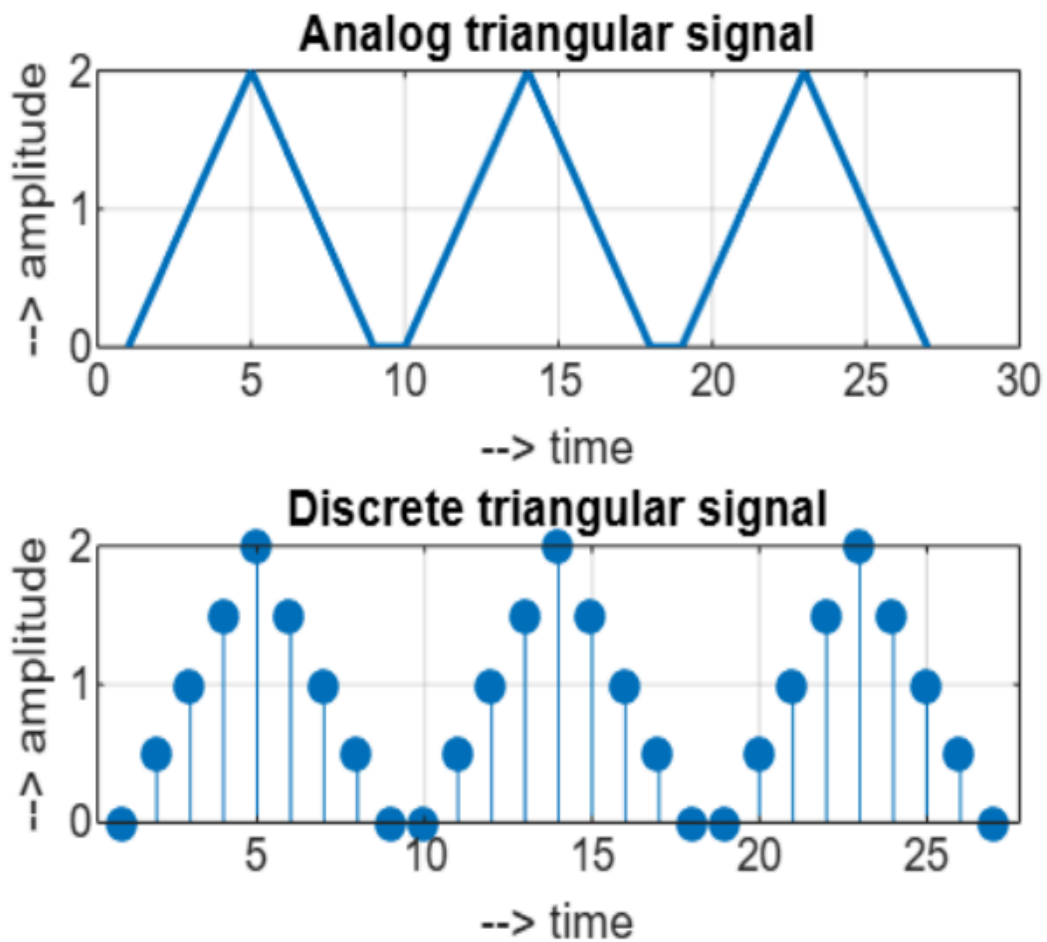
**OUTPUT:**



EX.NO :37

## PERFORM BLOCK CONVOLUTION USING MATLAB

DATE :



### RESULT:

Thus the triangular wave generated using MATLAB

### AIM:

To generate block convolution using MATLAB.

### **ALGORITHM:**

- Step-1: Clear the command window and workspace.
- Step-2: Declare the time scale and amplitude of the signal.
- Step-3: Using pre-defined functions generate the corresponding waveforms and plot them.
- Step-4: Label the axes and title.
- Step-5: Execute the program and view the output waveforms.

### **CODING:**

```
clc;
clear;
close all;
xn = input('Enter the Largest Sequence (x(n)) as a row vector: ');
hn = input('Enter the Second Sequence (h(n)) as a row vector: ');
figure;
subplot(2, 1, 1);
stem(xn, 'filled');
xlabel('Time Index');
ylabel('Amplitude');
title('INPUT SEQUENCE x(n)');
grid on;
subplot(2, 1, 2);
stem(hn, 'filled');
xlabel('Time Index');
ylabel('Amplitude');
title('INPUT SEQUENCE h(n)');
grid on;
m = length(xn);
l = length(hn);
g = m + l - 1;
xn = [xn, zeros(1, l - 1)];
hn = [hn, zeros(1, m - 1)];
yn = zeros(1, g);
for n = 1:g
    for k = 1:n
        if k <= m && (n - k + 1) <= l
            yn(n) = yn(n) + xn(k) * hn(n - k + 1);
        end
    end
end
end
```

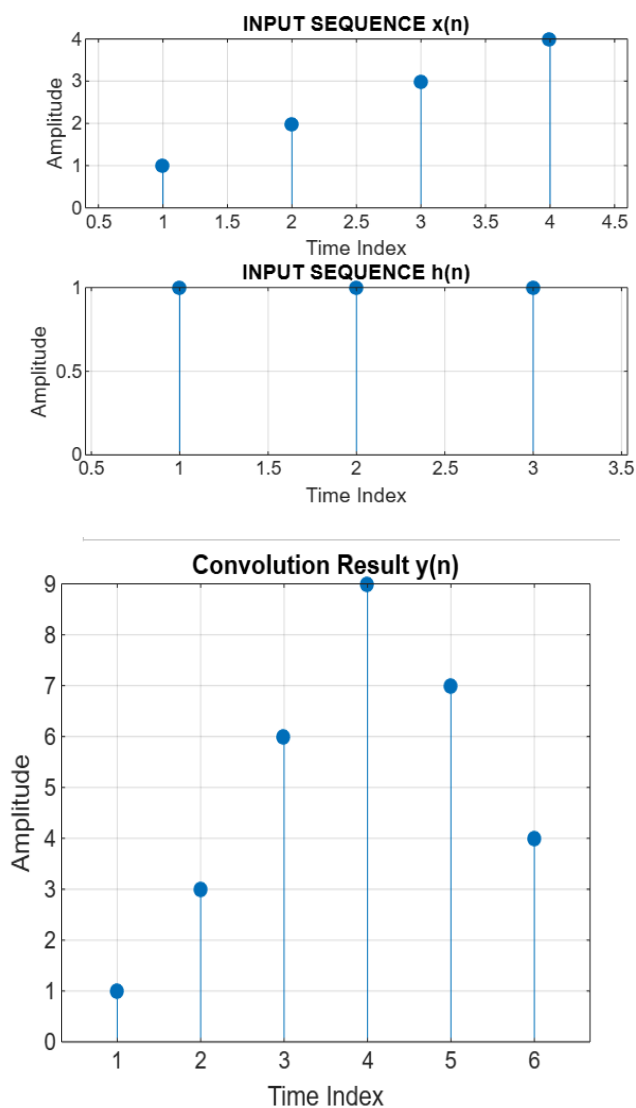
EX.NO :38

## MATLAB CODE FOR FOLDING A SEQUENCE AND PLOT IT.

DATE :

```
figure;  
stem(yn, 'filled');  
xlabel('Time Index');  
ylabel('Amplitude');  
title('Convolution Result y(n)');  
grid on;
```

### OUTPUT:



### RESULT:

Thus the block convolution is verified using MATLAB:

### AIM:

To generate a discrete-time sequence about the origin and plot both the original and folded sequences using MATLAB.

### ALGORITHM:

Step-1: Clear the command window and workspace.

Step-2: Define the sequence  $x[n]$  and its indices  $n$ .

Step-3: Reflect indices to obtain  $-n$ .

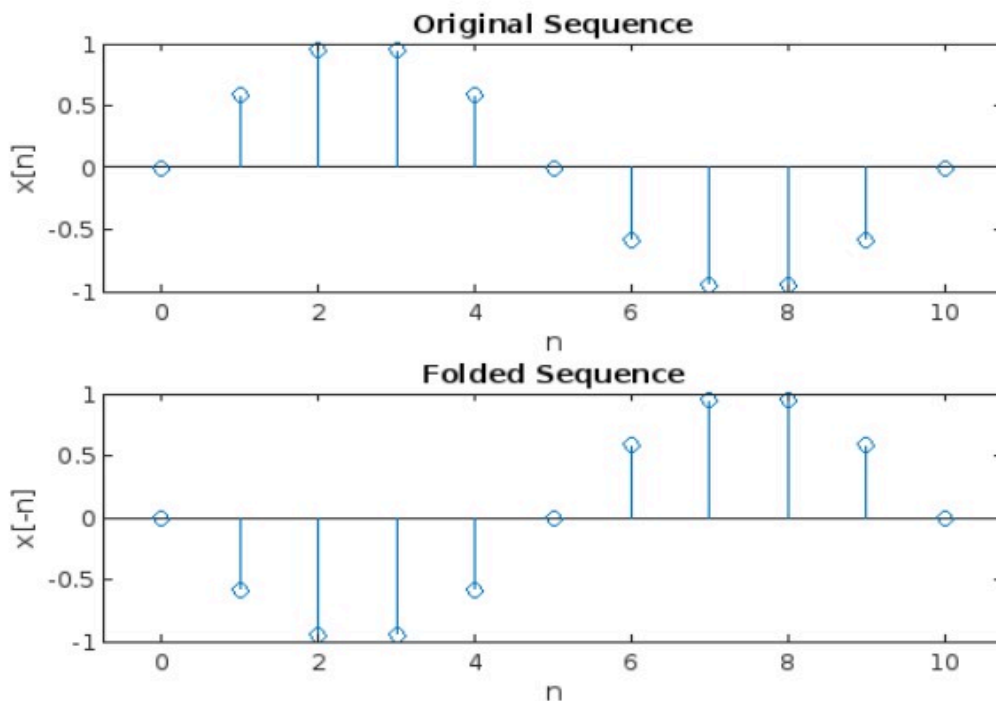
Step-4: Keep sequence values unchanged.

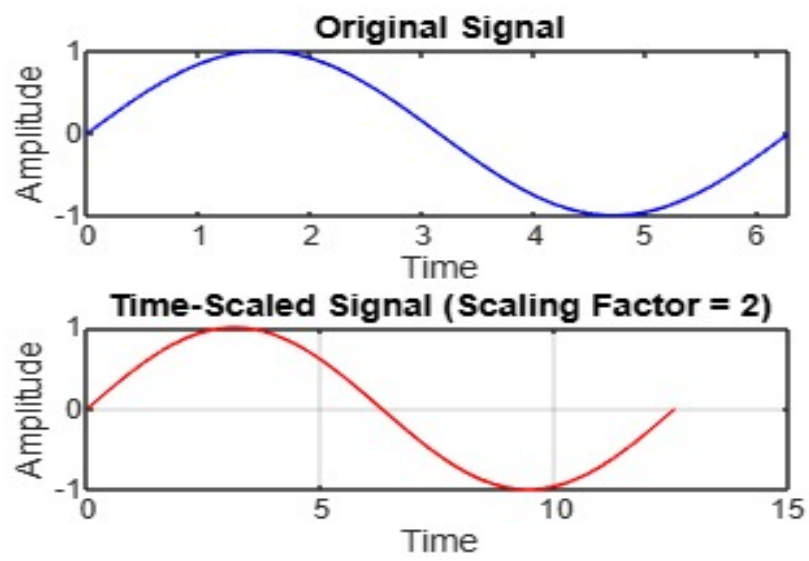
Step-5: Plot original and folded sequences.

### CODING:

```
% Define the original sequence
n = 0:10; % Time index
x = sin(2 * pi * 0.1 * n); % Original sequence (sine wave)
% Fold the sequence
foldedSequence = fliplr(x); % Reverse the sequence
% Plot the original and folded sequences
figure;
subplot(2, 1, 1), stem(n, x), title('Original Sequence'), xlabel('n'), ylabel('x[n]');
subplot(2, 1, 2), stem(n, foldedSequence), title('Folded Sequence'), xlabel('n'), ylabel('x[-n]');
```

### OUTPUT:





### RESULT:

The MATLAB code folds the sequence by reflecting its time indices and plots both the original and folded sequences for comparison

DATE :

**AIM:**

Perform time scaling on a signal using MATLAB to compress (speed up) or expand (slow down) its duration.

**ALGORITHM:**

Step-1: Clear the command window and workspace

Step-2: Set up the original signal  $x(t)$  and time vector  $t$ .

Step-3: Choose  $\alpha$  (compression) and  $\beta$  (expansion).

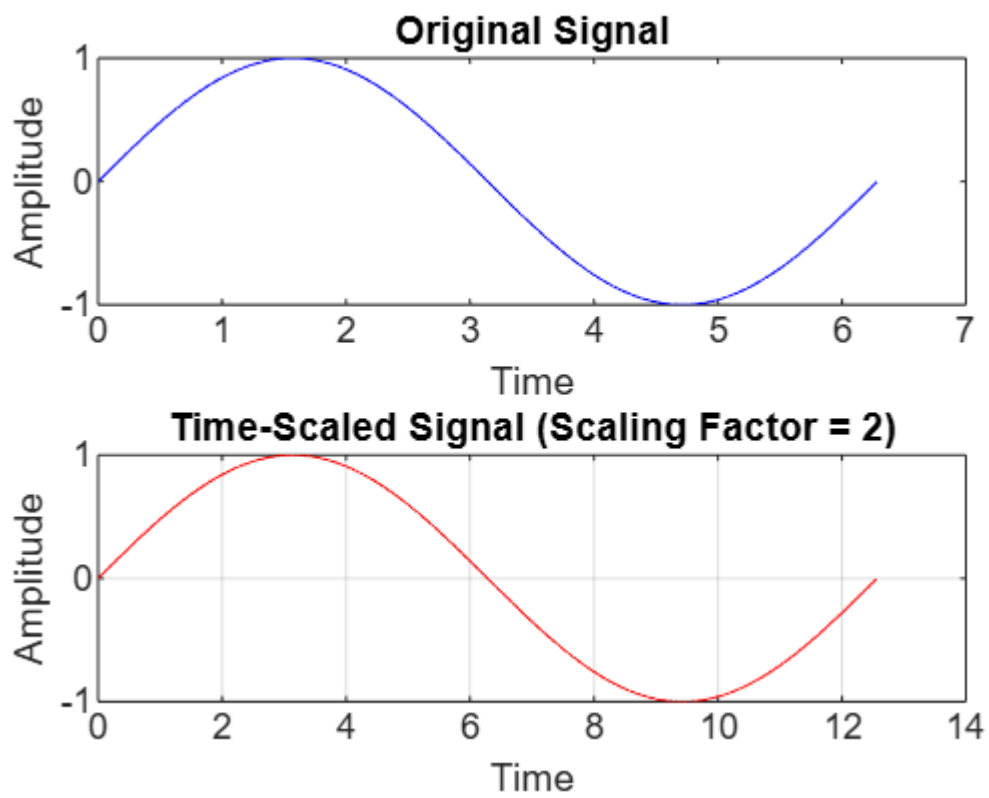
Step-4: Compute  $t/\alpha$  for compression and  $t/\beta$  for expansion.

Step-5: Display the original, compressed, and expanded signals.

**CODING:**

```
t = 0:0.01:2*pi ;    % Original time vector
x = sin(t) ;         % Original signal (sine wave)
a = 2 ;              % Scaling factor (a > 1 compresses, 0 < a < 1 expands)
t_scaled = t * a ;   % Scaled time vector
figure;
subplot(2, 1, 1);
plot(t, x, 'b');
title('Original Signal');
xlabel('Time');
ylabel('Amplitude');
subplot(2, 1, 2);
plot(t_scaled, x, 'r');
title(['Time-Scaled Signal (Scaling Factor = ' num2str(a) ')']);
xlabel('Time');
ylabel('Amplitude');
grid on;
```

### OUTPUT:



### RESULT:

The MATLAB code scales the signal in time, displaying the original, compressed, and expanded signals for comparison.

EX.NO :40

## MATLAB CODE TO FIND AREA OF CIRCLE USING MATLAB

DATE :

### AIM:

The aim of this MATLAB code is to calculate the area of a circle given its radius.

### ALGORITHM:

Step-1: Define the radius of the circle.

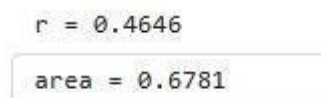
Step-2: Use the formula for the area of a circle ( $\pi * \text{radius}^2$ ).

Step-3: Display the result.

### CODING:

```
clc;  
clear all;  
close all;  
r= pi^(1/ 3)-1  
area=pi* r^2
```

### OUTPUT:



```
r = 0.4646  
area = 0.6781
```

### RESULT:

Thus the output gives the area of the circle.



EX.NO :41

## STABILITY TEST FOR TRANSFER FUNCTION

DATE :

### AIM:

The aim of this MATLAB code is to determine the stability of a system by analyzing the poles of its transfer function.

### ALGORITHM:

Step-1: Define the numerator and denominator of the transfer function.

Step-2: Use tf to create the transfer function.

Step-3: Calculate poles with pole(H).

Step-4: Check the real parts of the poles. If all real parts are negative, the system is stable.

### CODING:

```
num = [16]; % define the coefficients of numerator in the transfer function
den = [1, 1.6, 16]; % define the coefficients of denominator in the transfer function
sys = tf(num, den); % create the transfer function
poles = pole(sys); % to get the poles of the transfer function
% checks for stable, marginally stable and unstable
if all(real(poles) < 0)
    disp('The transfer function is stable.');
```

```
elseif all(real(poles) <= 0) && any(real(poles) < 0)
```

```
    disp('The transfer function is marginally stable.');
```

```
else
```

```
    disp('The transfer function is unstable.');
```

```
end
```

### RESULT:

Since all the poles have negative real parts, the system is stable.

EX.NO :42

## DESIGN HIGHPASS FIR FILTER USING RECTANGULAR WINDOW USING MATLAB

DATE :

### AIM:

To design a high pass FIR filter using a rectangular window in MATLAB, allowing frequencies above a specified cutoff to pass while attenuating lower frequencies.

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Choose the cut-off frequency, sampling rate, and filter order.
- Step-3: You can create the ideal impulse response for a high pass filter.
- Step-4: Multiply the ideal impulse response by the rectangular window to get the FIR filter coefficients.
- Step-5: Execute the program and view the output waveforms.

### CODING:

```
% Filter specifications
Fs = 1000;           % Sampling frequency in Hz
Fc = 200;            % Cutoff frequency in Hz
N = 50;              % Filter order (choose an even number for better symmetry)

% Normalize the cutoff frequency with respect to Nyquist frequency (Fs/2)
Wn = Fc / (Fs / 2);

% Design the high-pass FIR filter using fir1 with a rectangular window
b = fir1(N, Wn, 'high', rectwin(N + 1));

% Frequency response of the filter
[H, f] = freqz(b, 1, 1024, Fs);

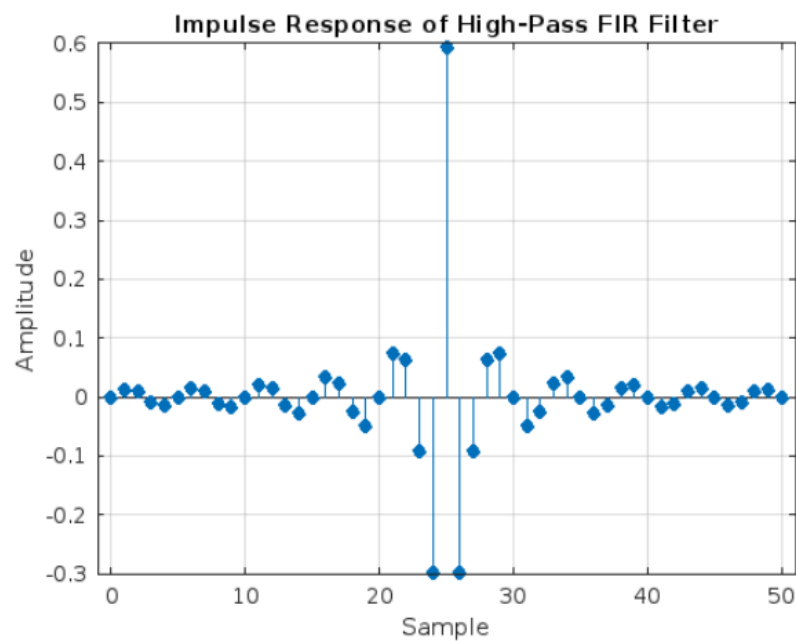
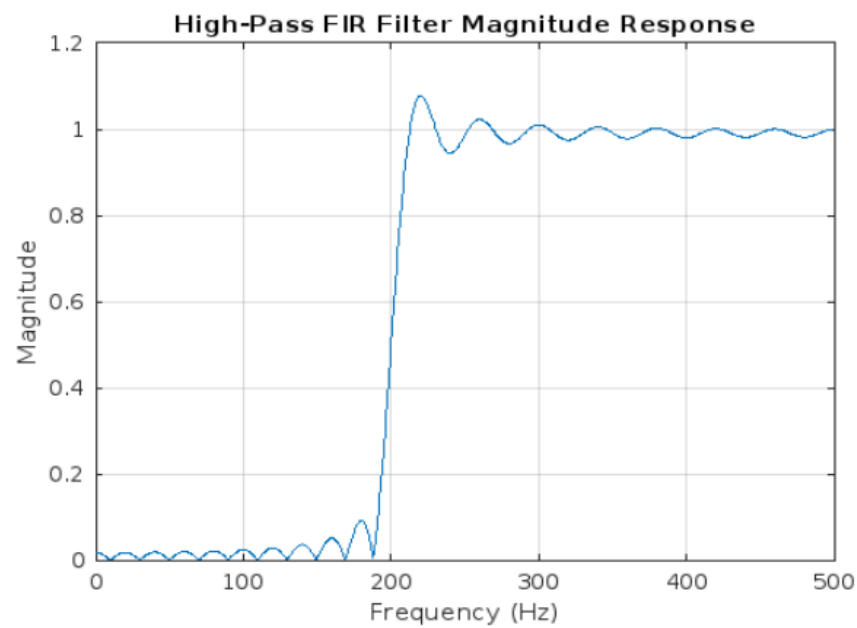
% Plot the magnitude response
figure;
plot(f, abs(H));
title('High-Pass FIR Filter Magnitude Response');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

grid on;

% Plot the impulse response of the filter
figure;
stem(0:N, b, 'filled');
title('Impulse Response of High-Pass FIR Filter');
xlabel('Sample');
ylabel('Amplitude');

grid on;
```

**OUTPUT:**



**RESULT:**

The high pass FIR filter was designed successfully, allowing higher frequencies to pass and attenuating lower frequencies as expected.

EX.NO :43

## MATLAB CODE FOR WAVEFORM ADDITION AND PLOT IT

DATE :

### AIM:

To write MATLAB code to generate, add two waveforms, and plot the individual waveforms and their resulting sum.

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Set sampling frequency and create a time vector.
- Step-3: Define two waveforms with chosen frequencies.
- Step-4: Add the two waveforms.
- Step-5: Plot each waveform and their sum in subplots with labels.

### CODING:

```
% Define the time vector
t = 0:0.01:2*pi; % Time from 0 to 2π with increments of 0.01

% Define the waveforms
x1 = sin(t); % First waveform: a sine wave
x2 = cos(t); % Second waveform: a cosine wave

% Add the waveforms
y = x1 + x2;

% Plot the waveforms and their sum
figure;
% Plot the first waveform
subplot(3, 1, 1);
plot(t, x1, 'b'); % Blue line for x1
title('Waveform x1: Sine Wave');
xlabel('Time');
ylabel('Amplitude');

% Plot the second waveform
subplot(3, 1, 2);
plot(t, x2, 'r'); % Red line for x2
title('Waveform x2: Cosine Wave');
xlabel('Time');
ylabel('Amplitude');

% Plot the sum of the waveforms
```

```

subplot(3, 1, 3);
plot(t, y, 'k'); % Black line for y
title('Sum of Waveforms: x1 + x2');
xlabel('Time');
ylabel('Amplitude');

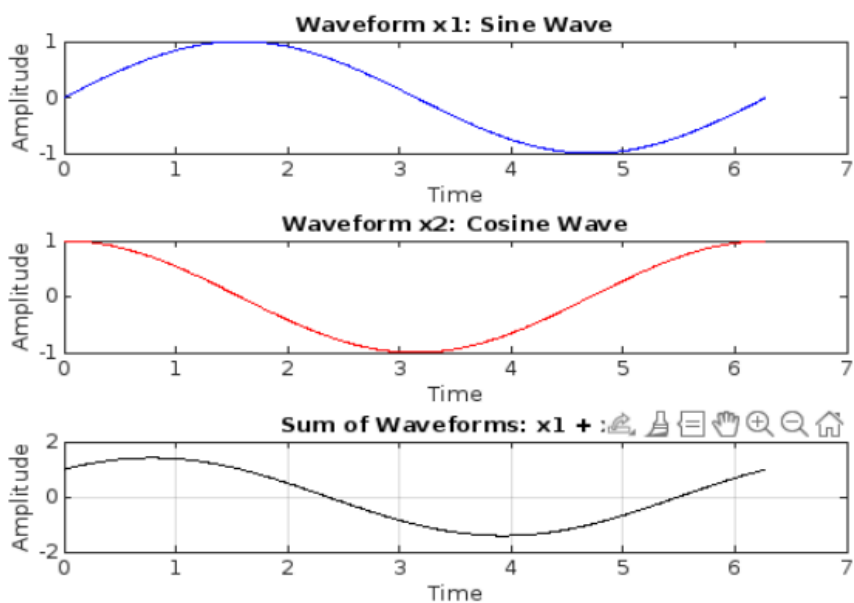
```

```

% Display the plot
grid on;

```

## OUTPUT:



## RESULT:

Two discrete signals are generated and the verified the waveform for addition of two signals .

EX.NO :44

## PERFORM AUTOCORRELATION

DATE :

### AIM:

To write MATLAB code to generate, add two waveforms, and plot the individual waveforms and their resulting sum.

SOFTWARE:MATLAB

### ALGORITHM:

- Step-1: Clear the command window and workspace.
- Step-2: Set sampling frequency and create a time vector.
- Step-3: Define two waveforms with chosen frequencies.
- Step-4: Do the auto correlation of two waveforms.
- Step-5: Plot each waveform and their sum in subplots with labels.

### CODING:

```
%Program for computing autocorrelation function x[1 2 3 4]
x=input('enter the sequence');
y=xcorr(x,x);
subplot(2,1,1);
stem(x);
ylabel('Amplitude --. ');
xlabel('(a) n --. ');
title('original signal');
subplot(2,1,2);
stem(y);
ylabel('Amplitude --. ');
xlabel('(a) n --. ');
title('Auto correlated sequence');
disp('The resultant signal is y=');
disp(y)
```

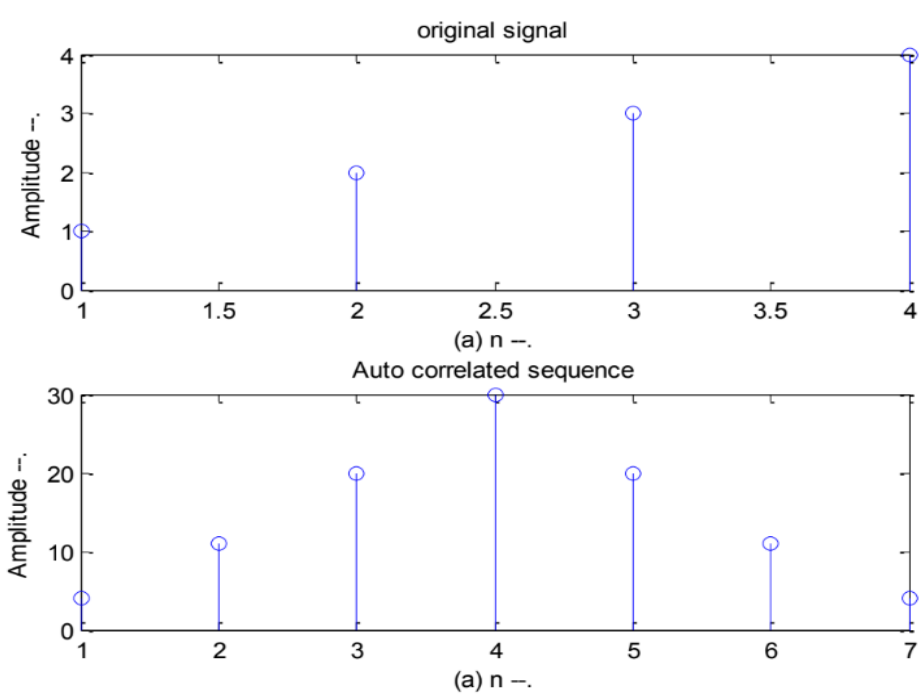
Output:

enter the sequence[1 2 3 4]

The resultant signal is y=

4.0000 11.0000 20.0000 30.0000 20.0000 11.0000 4.0000

Model Graph



**RESULT:**

Two discrete signals are generated and the autocorrelation was performed.

**AIM:**

To write a MATLAB program to implement an inbuilt function „impz“ to solve difference equations numerically.

SOFTWARE:MATLAB

**ALGORITHM:**

- Step-1: Clear the command window and workspace.
- Step-2: difference equation coefficients (b, a).
- Step-3: Define two waveforms with chosen frequencies.
- Step-4: Add the two waveforms.
- Step-5: Plot each waveform and their sum in subplots with labels.

**Calculation**

given the input and difference equation coefficients (b, a).

$y = \text{impz}(b, a, N)$

Where x is the input sequence, y is the output sequence which is of same length as x.

Let the Difference equation is given as  $y(n) = x(n) + 0.5x(n-1) + 0.85x(n-2) + y(n-1) + y(n-2)$

$$y(n) = x(n) + 0.5x(n-1) + 0.85x(n-2) + y(n-1) + y(n-2)$$

$y(n) - y(n-1) - y(n-2) = x(n) + 0.5x(n-1) + 0.85x(n-2)$  Taking Z transform on both sides,

$$y(z) - z^{-1}y(z) - z^{-2}y(z) = x(z) + 0.5z^{-1}x(z) + 0.85z^{-2}x(z)$$

$$y(z)[1 - z^{-1} - z^{-2}] = x(z)[1 + 0.5z^{-1} + 0.85z^{-2}]$$

But,  $h(z) = y(z)/x(z)$

$$= [1 + 0.5z^{-1} + 0.85z^{-2}] / [1 - z^{-1} - z^{-2}] \text{ By dividing we get}$$

$$H(z) = 1 + 1.5z^{-1} + 3.35z^{-2} + 4.85z^{-3}$$

$$h(n) = [1 \ 1.5 \ 3.35 \ 4.85]$$

**CODING:**

```
clc;
close all;
clear all;
%difference equation of second order system
%y(n)=x(n)+.5x(n-1)+.85x(n-2)+y(n-1)+y(n-2)
b=input('enter the coefficient of x(n),x(n-1),x(n-2). ');
a=input('enter the coefficient of y(n),y(n-1),y(n-2). ');
N=input('enter the no of samples of imp response=');
impz(b,a,N);
```



```

[h,t]=impz(b,a,N);
subplot(2,1,1);
%figure(1)
plot(t,h);
title('plot of impulse response');
ylabel('amplitude');
xlabel('time index---->N');
subplot(2,1,2);
%figure(2)
stem(t,h);
title('plot of impulse response');
ylabel('amplitude');
xlabel('time index---->N');
disp(h);
grid on;

```

Input:

enter the coefficient of  $x(n), x(n-1) \dots = [1 \ 0.5 \ 0.85]$

enter the coefficient of  $y(n), y(n-1) \dots = [1 \ -1 \ -1]$

enter the no of samples of imp response=4

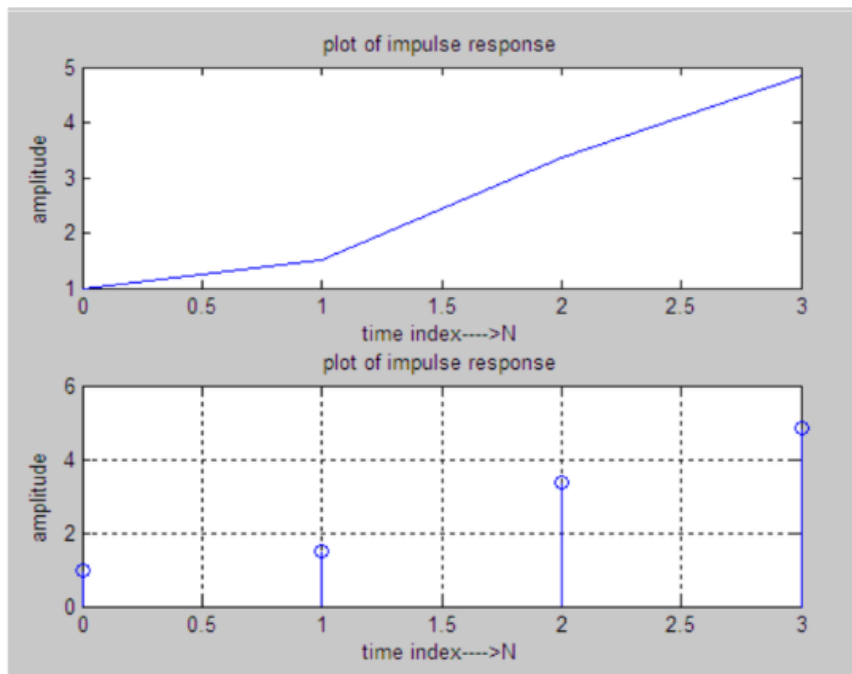
Output

1.0000

1.5000

3.3500

4.8500



## NO :46 **PERFORM CIRCULAR CONVOLUTION USING SUMMATION FORMULA**

DATE :

RESULT:

The given difference equation was solved and the output waveform was verified.

AIM:

To write a MATLAB program to perform convolution of two sequences using summation formula.

SOFTWARE:MATLAB

ALGORITHM:

1. Read the first input sequence,  $x[n]$  and plot.
2. Read the second input sequence,  $h[n]$  and plot
3. Find the length of  $x[n]$  and  $y[n]$  ,  $l1$  and  $l2$  respectively
4. Check if  $l1=l2$ . Proceed only if equal.
5. If  $l1$  not equal to  $l2$ , zero padding is done to make  $l1=l2$ .
6. Initialize a loop variable for the number of output points.
7. For each output sample access the samples of  $y[n]$  in cyclic order.
8. Find the sum of products of  $x[n]$  and cyclically folded and shifted  $h[n]$  to get circular convoluted output.
9. Display and plot the output.

CODING:

```
clc;
clear all;
close all;
xn=input('enter the first sequence x(n)=');
hn=input('enter the second sequence h(n)=');
l1=length(xn);
l2=length(hn);
N=max(l1,l2);
xn=[xn,zeros(1,N-l1)];
hn=[hn,zeros(1,N-l2)];
for n=0:N-1; y(n+1)=0; for k=0:N-1
i=mod((n-k),N);
```

OUTPUT

**Input:**

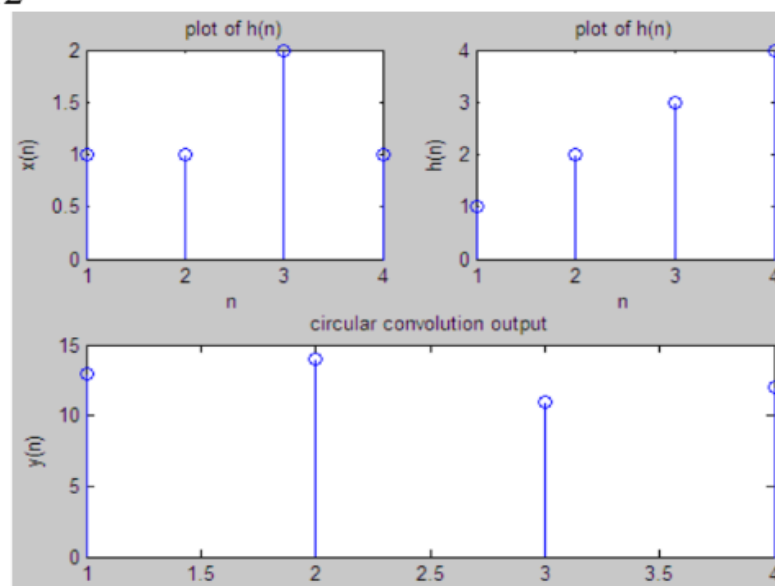
enter the first sequence  $x(n)=[1 \ 1 \ 2 \ 1]$

enter the second sequence  $h(n)=[1 \ 2 \ 3 \ 4]$

**Output:**

circular convolution output

13 14 11 12



EX.NO :47

DATE :

### **RESULT:**

The MATLAB code successfully generated, added, and plotted two waveforms. The individual waveforms and their convolution result were displayed as expected.

### **PERFORM CONVERTING CD DATA TO DVD DATA**

AIM:

To write a MATLAB program to Convert CD DATA TO DVD DATA.

SOFTWARE:MATLAB

ALGORITHM:

1. Open MATLAB
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. For the output see command window\ Figure window

CODING:

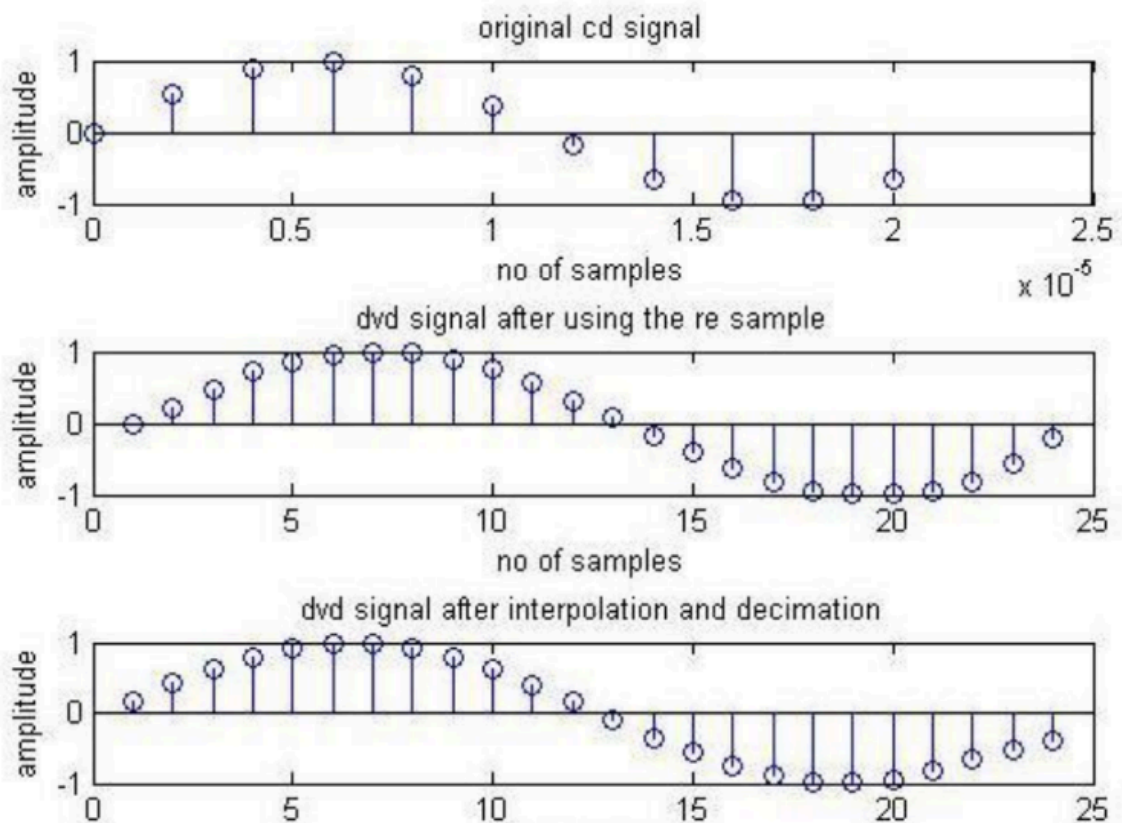
```
clc;
clear all;
fc=44100;
t=0:0.000002:0.00002;
x=sin(2*pi*fc*t);
subplot(3,1,1);
stem(t,x);
title('original cd signal');
xlabel('no of samples');
ylabel('amplitude');
```

```

i=13;
d=6;
y=resample(x,i,d);
subplot(3,1,2);
stem(y);
title('dvd signal after using the re sample');
xlabel('no of samples');
ylabel('amplitude');
in=interp(x,i);
de=decimate(in,d);
subplot(3,1,3);
stem(de);
title('dvd signal after interpolation and decimation');
xlabel('no of samples');
ylabel('amplitude');

```

## OUTPUT:



EX.NO :48

## PERFORM DECIMATION OF A SIGNAL

DATE :

RESULT:

Thus the required conversion was done and the output waveform was verified.

AIM: To perform decimation using MATLAB Software.

SOFTWARE:MATLAB

ALGORITHM:

- 1.Enter the downsampling factor and input frequencies.
- 2.Perform decimation of a signal
- 3.Plot the input and output of the signal

CODING:

```
Clc;
Clear all;
Close all;
D=input('enter the downsampling factor');
L=input('enter the length of the input signal');
f1=input('enter the frequency of first sinusodal');
f2=input('enter the frequency of second sinusodal');
n=0:L-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:L));
title('input sequence');
xlabel('time(n)');
ylabel('amplitude');
subplot(2,1,2)
```

```

m=0:(L/D)-1;
stem(m,y(1:L/D));
title('Decimated sequence');
xlabel('time(n)');
ylabel('amplitude');

```

INPUT:

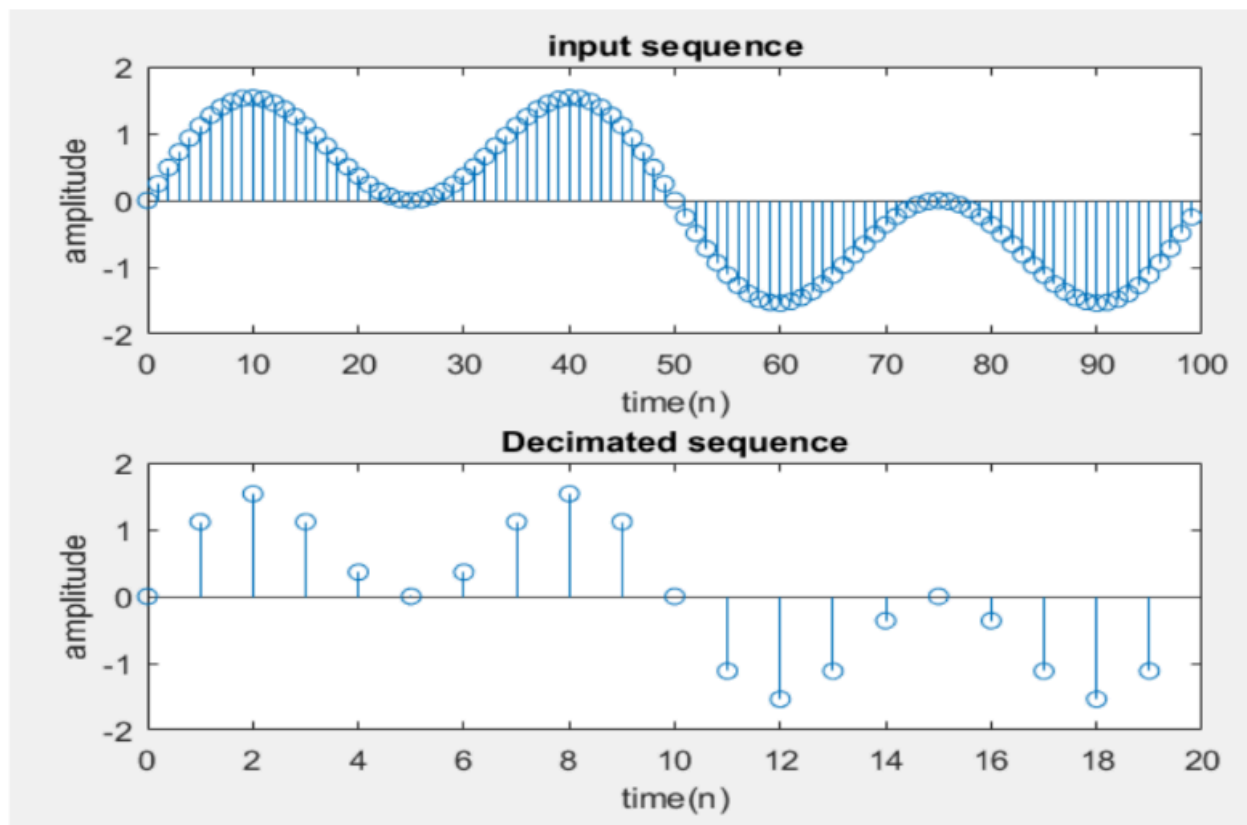
enter the downsampling factor = 5

enter the length of the input signal = 100

enter the frequency of first sinusoidal = 0.01

enter the frequency of second sinusoidal = 0.03

Output Waveforms:



RESULT:

The decimation factor applied for the given sequence and the output waveforms were verified.

EX.NO :49

DATE : IMPLEMENTATION OF INTERPOLATION PROCESS

AIM: program to verify the decimation of given sequence.

SOFTWARE:MATLAB

ALGORITHM:

Step 1 : Define up sampling factor and input frequencies f1 and f2

Step 2 : Represent input sequence with frequencies f1 and f2

Step 3 : Perform the interpolation on input signal using matlab command interp.

Step 4 : Plot the input and output signal/sequence.

CODING:

Clc;

Clear all;

Close all;

L=input('enter the upsampling factor');

N=input('enter the length of the input signal'); % Length should be greater than 8

f1=input('enter the frequency of first sinusoidal');

f2=input('enter the frequency of second sinusoidal');

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('time(n)');

ylabel('amplitude');



```

subplot(2,1,2)
m=0:N*L-1;
stem(m,y(1:N*L))
title('output sequence ');
xlabel('time(n)');
ylabel('amplitude');

```

INPUT:

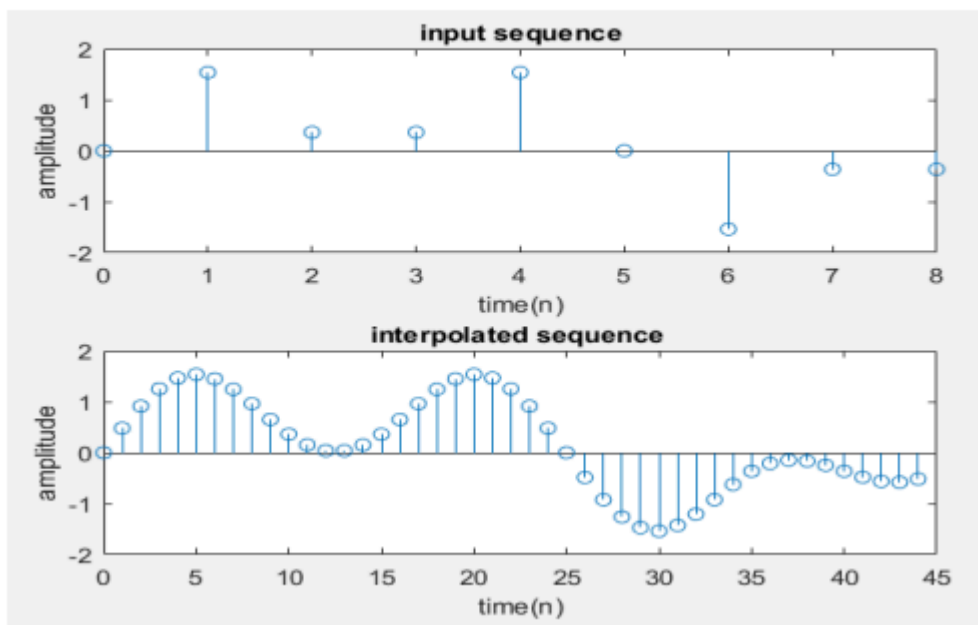
enter the upsampling factor = 5

enter the length of the input signal = 9

enter the frequency of first sinusoidal = 0.1

enter the frequency of second sinusoidal = 0.3

### Output Waveforms:



RESULT:

Thus the interpolation process done for the given sequence and the output waveforms verified.

EX.NO :50

IMPLEMENTATION OF I/D SAMPLING RATE CONVERTERS  
DATE :

AIM: program to implement sampling rate conversion.

SOFTWARE:MATLAB

Algorithm:

Step 1 : Define up sampling factor, down sampling and input frequencies f1 and f2

Step 2 : Represent input sequence with frequencies f1 and f2

Step 3: Perform I/D sampling rate conversion on input signal using matlab resample

Step 4 : Plot the input and output signal/sequence.

CODING:

```
Clc;
```

```
Clear all;
```

```
Close all;
```

```
L=input('enter the upsampling factor');
```

```
D=input('enter the downsampling factor');
```

```
N=input('enter the length of the input signal');
```

```
f1=input('enter the frequency of first sinusodal');
```

```
f2=input('enter the frequency of second sinusodal');
```

```
n=0:N-1;
```

```
x=sin(2*pi*f1*n)+sin(2*pi*f2*n);
```

```
y=resample(x,L,D);
```

```
subplot(2,1,1)
```

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

```
title('input sequence');
```

```
xlabel('time(n)');
```

```
ylabel('amplitude');
```

```
subplot(2,1,2)
```

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

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

```
title('output sequence ');
```

```
xlabel('time(n)');
```

```
ylabel('amplitude');
```

INPUT:

enter the upsampling factor = 3

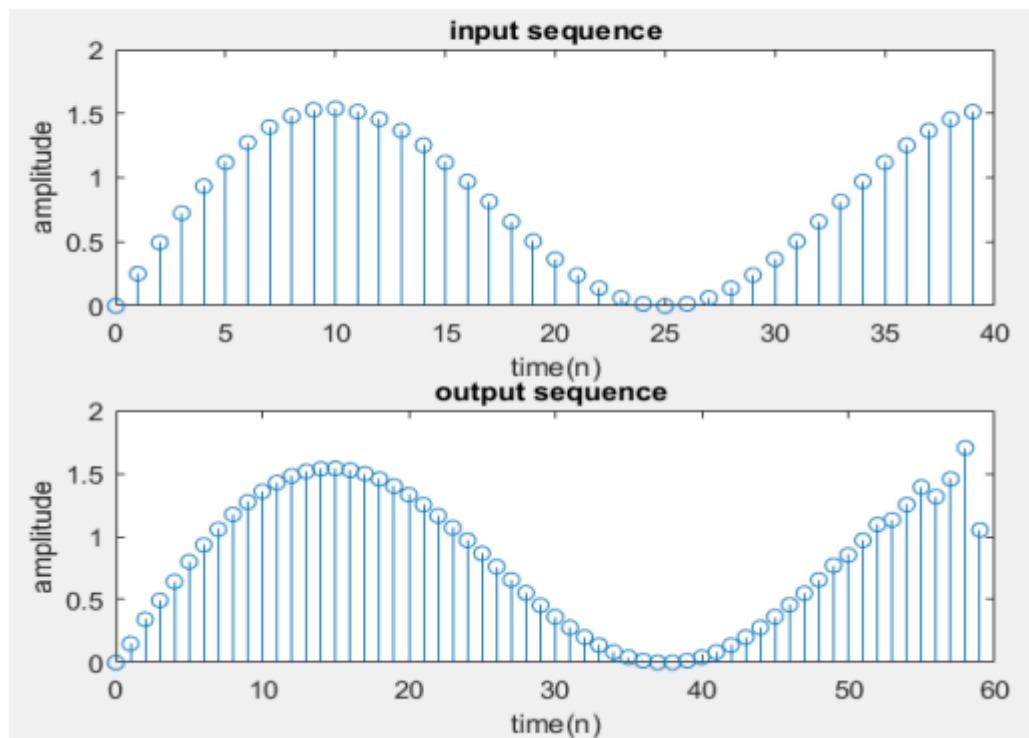
enter the downsampling factor = 2

enter the length of the input signal = 40

enter the frequency of first sinusoidal = 0.01

enter the frequency of second sinusoidal = 0.03

### Output Waveforms:



## RESULT:

Thus the sampling rate converter program was executed and the output waveforms are verified.