ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

B.E. Information Technology

VI – SEMESTER

21ITCP607 DIGITAL SIGNAL PROCESSING AND INFORMATION CODING TECHNIQUES LAB

LABORATORY MANUAL

January 2024- May 2024

Name:_____

Reg.No:_____

DEPARTMENT OF INFORMATION TECHNOLOGY
INSTITUTIONAL – VISION AND MISSION

VISION:

Providing world class quality education with strong ethical values to nuture and develop outstanding professionals fit for globally competitive environment.

MISSION:

- Provide quality technical education with a sound footing on basic engineering principles, technical and managerial skills, and innovative research capabilities.
- Transform the students into outstanding professionals and technocrats with strong ethical values capable of creating, developing and managing global engineering enterprises.
- Develop a Global Knowledge Hub, striving continuously in pursuit of excellence in education, research entrepreneurship and technological services to the industry and society.
- Inculcate the importance and methodology of life-long learning to move forward with updated knowledge to face the challenges of tomorrow.

DEPARTMENT – VISION AND MISSION

VISION:

To produce globally competent, quality technocrats, to inculcate values of leadership and research qualities and to play a vital role in the socio – economic progress of the nation.

MISSION:

M1 : To partner with the University community to understand the information technology needs of faculty, staff and students.

M2 : To develop dynamic IT professionals with globally competitive learning experience by providing high class education.

M3 : To involve graduates in understanding need based Research activities and disseminate the knowledge to develop entrepreneur skills.

21ITCP607	DIGITAL SIGNAL PROCESSING AND INFORMATION CODING TECHNIQUES LAB	L	T	P	C
		0	0	3	1.5

COURSE OBJECTIVES

- To generate various types of continuous and discrete time signals and perform various operations
- To demonstrate various properties of continuous and discrete time systems
- To represent the signals using various forms of Fourier representation
- To characterize and analyze systems using Laplace, Fourier and z – transforms
- To expose students the basic concepts of information theory.
- To make students understand and implement various codes.
- To explore coding techniques applicable for multimedia inputs such as text, audio, image and video.

LIST OF EXERCISES

1) Generation of Elementary Signals.
2) Verification of Sampling Theorem.
3) Impulse and Step Response of LTI System.
4) Linear and Circular Convolution of Discrete Sequences.
5) Correlation and Auto Correlation of Discrete Sequences.
6) Z–Transform and Inverse Z–Transform.
7) Computation of DFT & IDFT of a Signal.
8) Spectral Analysis of a Signal.
9) Alteration of Sampling Rate of a Signal.
10) Design of IIR Filters.
11) Design of FIR Filters.
12) Finding the Sum of two Sinusoidal Signals.
13) N Point FFT of a given sequence.
14) Frequency Response of Analog Low Pass and High Pass Filters.
15) FFT of a given 1–D signal.
16) Determination of entropy of a given source
17) Determination of various entropies and mutual information for a noise free channel
18) Determination of various entropies and mutual information for a binary symmetric channel
19) Coding and decoding of linear block codes
20) Coding and decoding of cyclic codes
21) Coding and decoding of convolutional codes
22) Coding and decoding of BCH codes
23) Coding and decoding of RS codes.
24) Implementation of Shannon-Fano coding algorithm.
25) Implementation of static Huffman coding algorithm.
26) Implementation of dynamic Huffman coding algorithm.
27) Implementation of run length encoding algorithm.
28) Implementation of arithmetic coding algorithm.
29) Implementation of linear predictive coding
30) Implement any one image compression algorithm
31) Implement any one video compression algorithm

COURSE OUTCOMES:

At the end of this course, the students will be able to

1. Apply the MATLAB code for the generation of signals
2. Develop a MATLAB program for DFT and IDFT of signals
3. Design and Implementation of IIR and FIR filters

Mapping of Course Outcomes (COs) with Programme Outcomes (POs) and Program Specific Outcomes (PSOs)															
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**	**PSO3**
CO1	3	3	3	3	3	-	-	-	-	-	-	-	3	-	-
CO2	-	3	3	3	3	2	-	-	-	-	2	2	-	3	-
CO3	-	-	-	3	3	-	-	-	-	1	2	3	-	-	3

S. No.	Date	LIST OF EXERCISES	Page No	Sign
1.		Generation of Elementary Signals		
2.		Verification of Sampling Theorem		
3.		Impulse and Step Responses of LTIS System		
4.		Linear and Circular Convolution of Discrete Sequences		
5.		Correlation and Autocorrelation of Discrete Sequences		
6.		Z-Transform and Inverse Z-Transform		
7.		Computation of DFT of a Signal		
8.		Sampling Rate Alteration of a Signal		
9.		Design of IIR Filters		
10.		Design of FIR Filters		
11.		Determination of various Entropies and Mutual Information of the given channel		
12.		Implementation of Shannon Fano coding		
13.		Implementation of Huffman Code		
14.		Implementation of Arithmetic Coding Algorithm		
15.		Error Detection using Parity Bit		
16.		Error Detection using Cyclic Redundancy Check (CRC)		
17.		Error Correction using Hamming Code		

18.		Implementation of Image Compression using Wavelet Transform		

Ex. No:1 **Generation of Elementary Signals**
Date:

Objective:

To generate elementary discrete time signals.

Description:

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

The signals may be classified into different categories such as continuous time signals and discrete time signals. A continuous time signal is denoted as *x(t)* and to emphasize the discrete time nature of a signal, a discrete time signal is represented as *x(n)* instead of *x(t)*. A discrete time signal can be represented in any of the following forms:

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

Some of the elementary discrete time signals have been generated in this experiment.

Unit Sample Sequence:

This is often called as the discrete time impulse or the unit impulse. It is denoted by $\delta(n)$ and defined by

$$\delta(n) = \begin{cases} 1, & if\ n = 0 \\ 0, & if\ n \neq 0 \end{cases}$$

Unit Step Sequence:

It is denoted by *u(n)* and defined by

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

Unit Ramp Sequence:

It is denoted by r*(n)* and defined by

$$r(n) = \begin{cases} n, & \\ 0, & \end{cases}$$

$if\ \ n \geq 0$

$if\ \ n < 0$

Sinusoidal Sequence:

The real sinusoidal sequence with constant amplitude is given by

$$x(n) = A\sin(\omega_0 n + \phi), \qquad -\infty < n < \infty$$

where

A = Amplitude of sinusoid

ω_0 = Angular frequency in radians per sample

ϕ = Phase in radians

Exponential Sequence:

The general form of exponential sequence is given by

$$x(n) = A\alpha^{n}, \qquad -\infty < n < \infty$$

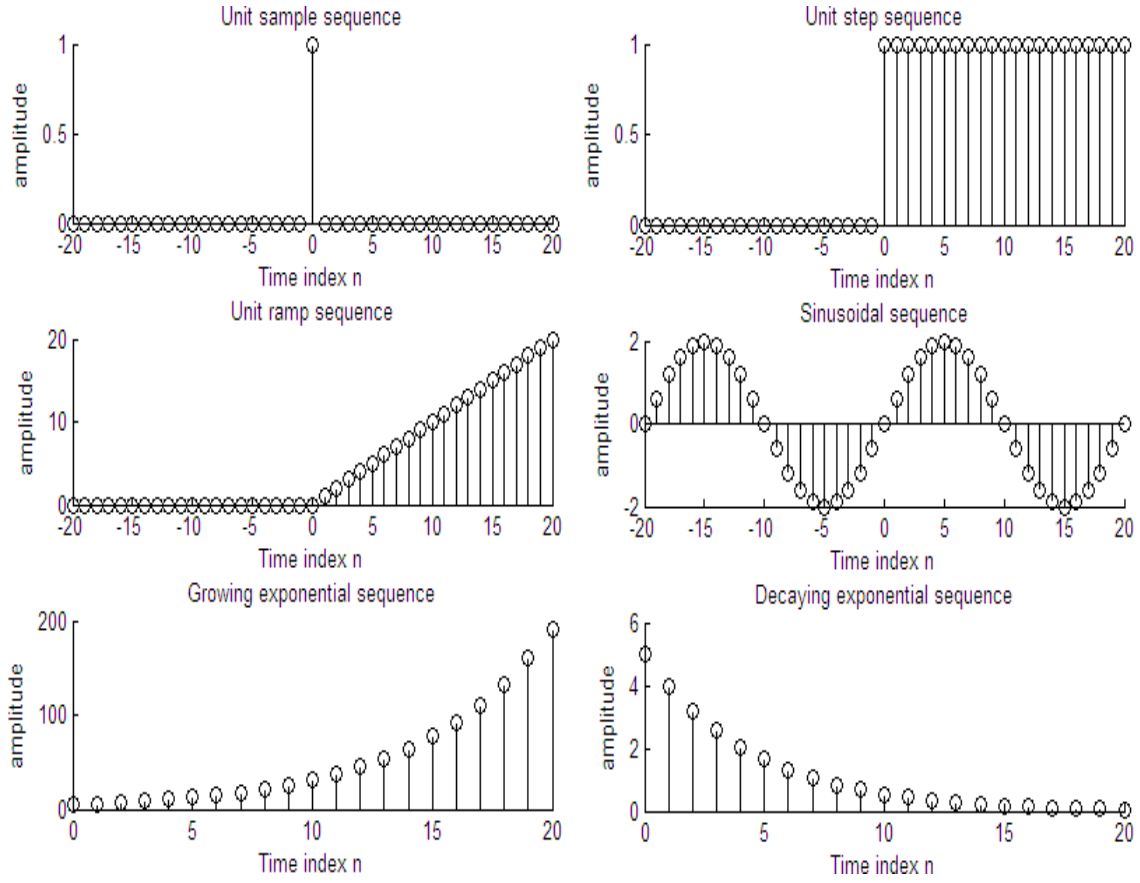
where A and α are real or complex numbers. When both A and α are real, the sequence is called as real exponential sequence. For $n \geq 0$, if $\alpha > 1$ the exponential sequence becomes growing

exponential, if $\alpha < 1$ the exponential sequence becomes decaying exponential.

Matlab code:

```
%-------------------------- Program to generate unit sample----------------
% generate a vector from –20 to 20
n=-20:20;
d=[zeros(1,20) 1 zeros(1,20)];
% plot the sequence
subplot(3,2,1),stem(n,d);
title('Unit sample sequence');
xlabel('Time index n');ylabel('amplitude');
%---------------------- Program to generate unit step--------------------
% generate a vector from –20 to 20
n=-20:20;
u=[zeros(1,20) 1 ones(1,20)];
% plot the sequence
subplot(3,2,2),stem(n,u);
title('Unit step sequence');
xlabel('Time index n');ylabel('amplitude');
%----------------- Program to generate unit ramp sequence-----------------
n=-20:20;
n2=0:20;
d1=[zeros(1,20) n2];
subplot(3,2,3),stem(n,d1);
title('Unit ramp sequence');
xlabel('Time index n');ylabel('amplitude');
%---------------------- Program to generate sinusoidal
sequence ------------n=-20:20;
x=2*sin(0.1*pi*n);
subplot(3,2,4),stem(n,x);
title('Sinusoidal sequence');
xlabel('Time index n');ylabel('amplitude');
%-------- Program to generate real growing exponential sequence------------
n1=0:1:20;
x1=5*1.2.^n1;
subplot(3,2,5),stem(n1,x1);
title('Growing exponential sequence');
xlabel('Time index n');ylabel('amplitude');
%--------- Program to generate real decaying exponential sequence----------
n1=0:1:20;
x1=5*0.8.^n1;
subplot(3,2,6),stem(n1,x1);
title('Decaying exponential sequence');
xlabel('Time index n');ylabel('amplitude');
```

Output



Result

Thus the important elementary discrete time signals are generated using Matlab.

Ex. No:2 **Verification of Sampling Theorem**
Date:

Objective:

To illustrate the sampling process and to verify the sampling theorem.

Description:

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

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

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

- Sampling
- Quantization
- Digital Coding

Sampling Process:

It is the conversion of a continuous time signal into a discrete time signal by obtaining "samples" of the continuous time signal at discrete time instants. Thus if $x_a(t)$ is the input to the sampler, the output is $x_a(nT) = x(n)$, Where T is called the sampling interval.

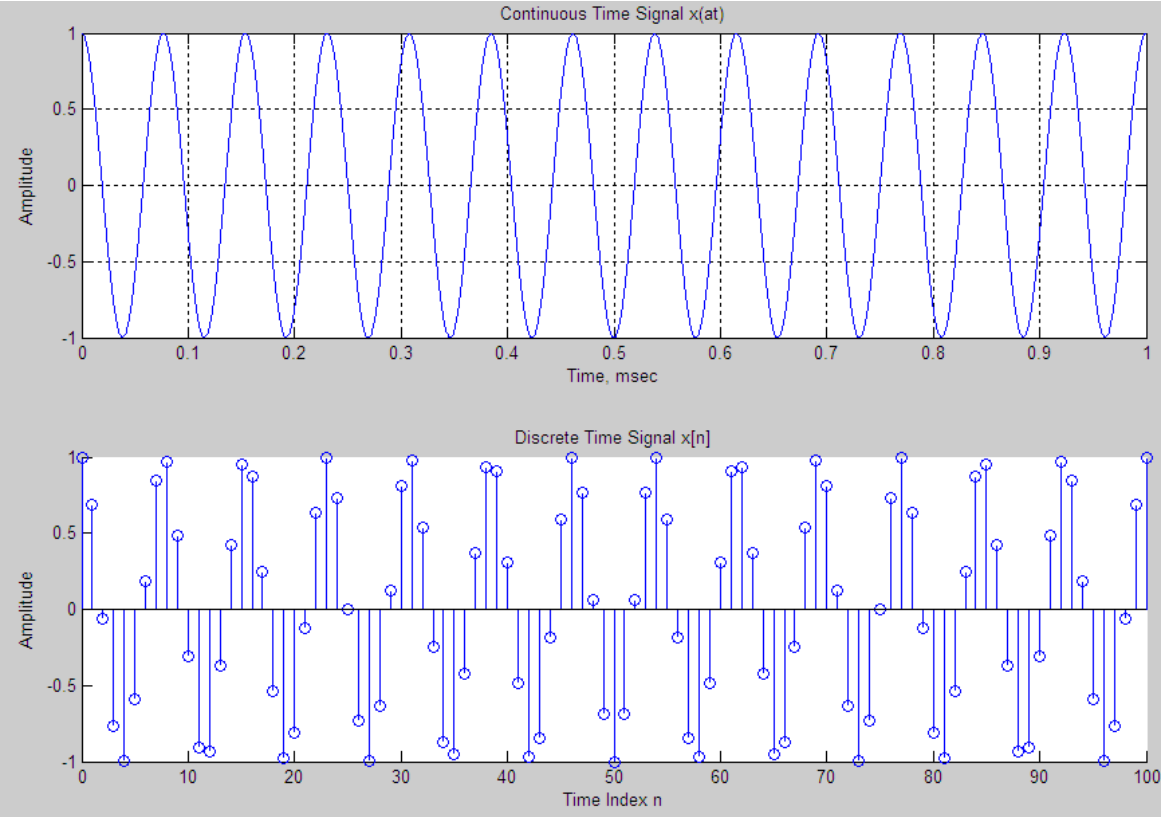
Sampling theorem explains, the minimum sampling rate to reconstruct the original signal with minimal distortion, should be equal to twice the highest frequency component of the signal.

Matlab code:

```
%---------- Illustration of sampling process----------
clf;
t = 0:0.00005:1;
f = 13;
xa = cos(2*pi*f*t);
subplot(2,1,1)
plot(t,xa); grid;
xlabel('Time, msec');
ylabel('Amplitude');
title('Continuous Time Signal xa(t)');
T=0.01;
n=0:T:1;
xs= cos(2*pi*f*n);
k=0:length(n)-1;
subplot(2,1,2);
stem(k,xs);
xlabel('Time Index n'); ylabel('Amplitude');
```

title('Discrete Time Signal x[n]');

Output

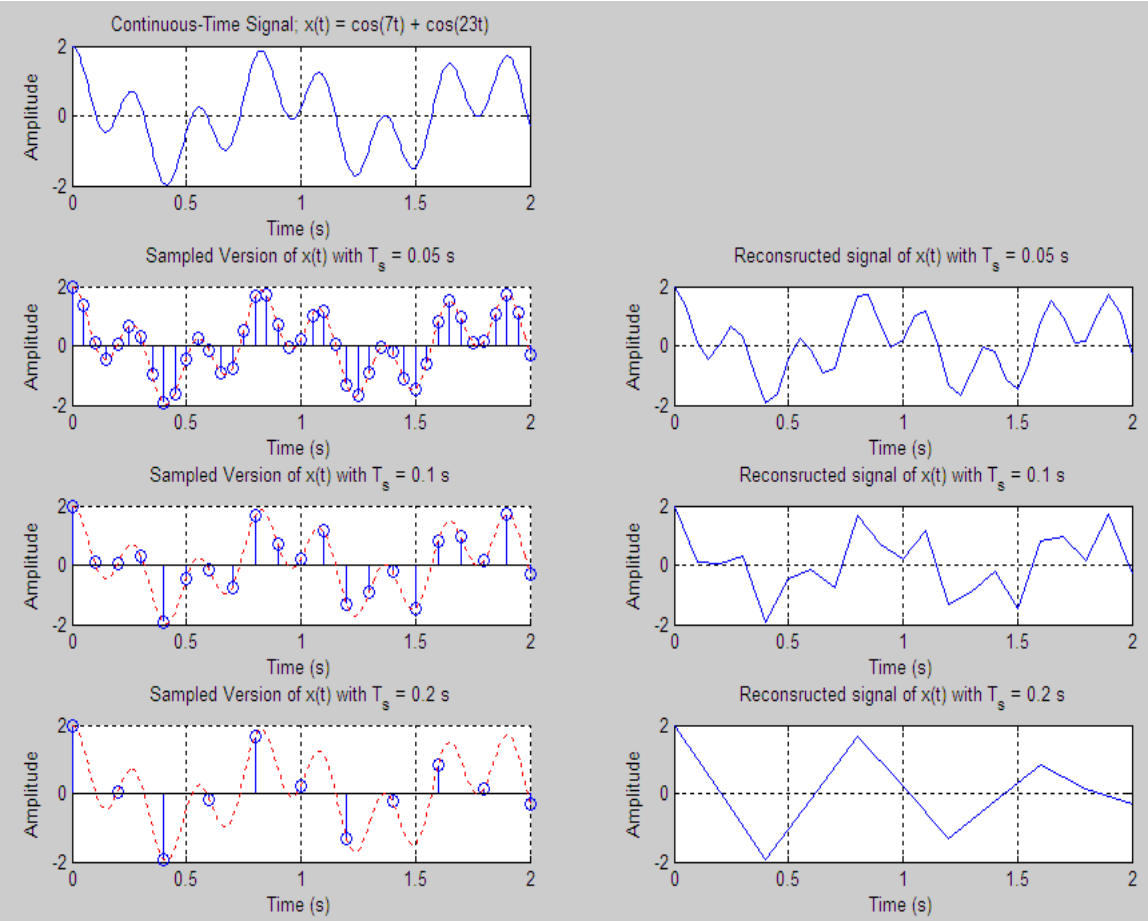


Matlab code:

```
%--------------------Verification of sampling theorem----------------------

% ws1 = 125.6637, ws2 = 62.8319, ws3 = 31.4159
Ts1 = 0.05; Ts2 = 0.1; Ts3 = 0.2;
ws1 = 2*pi/Ts1; ws2 = 2*pi/Ts2; ws3 = 2*pi/Ts3;

% Frequencies for the continous-time signal and the time vector.
w1 = 7; w2 = 23;
t = [0:0.005:2];

% Original continuous-time signal is the sum of two cosines, with
% frequencies of 7 r/s and 23 r/s, respectively.
x = cos(w1*t) + cos(w2*t);
subplot(4,2,1),plot(t,x),grid,xlabel('Time (s)'),ylabel('Amplitude'),...
title('Continuous-Time Signal; x(t) = cos(7t) + cos(23t)'),...
```

```
% Sampling the continuous-time signal with a sampling period Ts = 0.05 s.
% The sampled signal is exactly equal to the continuous-time signal at the
% sample times, and the samples accurately close to the original signal.
% ws1is approximately 5.5*w2.

t1 = [0:Ts1:2];
xs1 = cos(w1*t1) + cos(w2*t1);
subplot(4,2,3),stem(t1,xs1);grid,hold on,plot(t,x,'r:'),hold off,...
xlabel('Time (s)'),ylabel('Amplitude'),...
title('Sampled Version of x(t) with T_s = 0.05 s'),...
subplot(4,2,4),plot(t1,xs1);
grid,xlabel('Time (s)'),ylabel('Amplitude'),...
title('Reconsructed signal of x(t) with T_s = 0.05 s'),...


% Sampling the continuous-time signal with a sampling period Ts = 0.1 s.
% The sampled signal is exactly equal to the continuous-time signal at the
% sample times. The samples are a less accurate representation of the
% original signal than with the smaller Ts (higher sampling frequency ws).
% ws2 is approximately 2.7*w2.

t1 = [0:Ts2:2];
xs1 = cos(w1*t2) + cos(w2*t2);
subplot(4,2,5),stem(t2,xs2);grid,hold on,plot(t,x,'r:'),hold off,...
xlabel('Time (s)'),ylabel('Amplitude'),...
title('Sampled Version of x(t) with T_s = 0.1 s'),...
subplot(4,2,6),plot(t2,xs2);
grid,xlabel('Time (s)'),ylabel('Amplitude'),...
title('Reconsructed signal of x(t) with T_s = 0.1 s'),...


% Sampling the continuous-time signal with a sampling period Ts = 0.2 s.
% The sampled signal is exactly equal to the continuous-time signal at the
% sample times. The samples now are not a good representation of the
% original signal at all. ws3 is approximately 1.37*w2.

t3 = [0:Ts3:2];
xs3 = cos(w1*t3) + cos(w2*t3);
subplot(4,2,7),stem(t3,xs3);grid,hold on,plot(t,x,'r:'),hold off,...
xlabel('Time (s)'),ylabel('Amplitude'),...
title('Sampled Version of x(t) with T_s = 0.2 s'),...
subplot(4,2,8),plot(t3,xs3);
grid,xlabel('Time (s)'),ylabel('Amplitude'),...
title('Reconsructed signal of x(t) with T_s = 0.2 s'),...
```

Output



Result

Thus the sampling process is illustrated and the sampling theorem is verified using Matlab.

Ex. No: 3 **Impulse and Step Responses of LTI System**
Date:

Objective:

To generate elementary discrete time signals.

Description

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

Linear Time Invariant (LTI) Discrete Time Systems

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

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

A system is called time invariant if its input-output characteristics do not change with time.

An important subclass of LTI discrete time systems is characterized by a linear constant coefficient difference equation of the form

$$\sum_{k=0}^{N} d_k\, y(n-k) = \sum_{k=0}^{M} p_k\, x(n-k)$$

where *x(n)* and *y(n)* are, respectively, the input and output of the system and $\{d_k\}$ and $\{p_k\}$ are constants. If the system is causal then

$$y(n) = -\sum_{k=1}^{N} \frac{d_k}{d_0}\, y(n-k) + \sum_{k=0}^{M} \frac{p_k}{d_0}\, x(n-k) \quad \text{provided } d_0 \neq 0$$

The response of a discrete time system to a unit sample sequence is called the unit impulse response or unit sample response. Likewise, the response of a discrete time system to a unit step sequence is called the unit step response.

Matlab code:

The program given below computes the impulse and step response of length N of the causal LTI system defined by

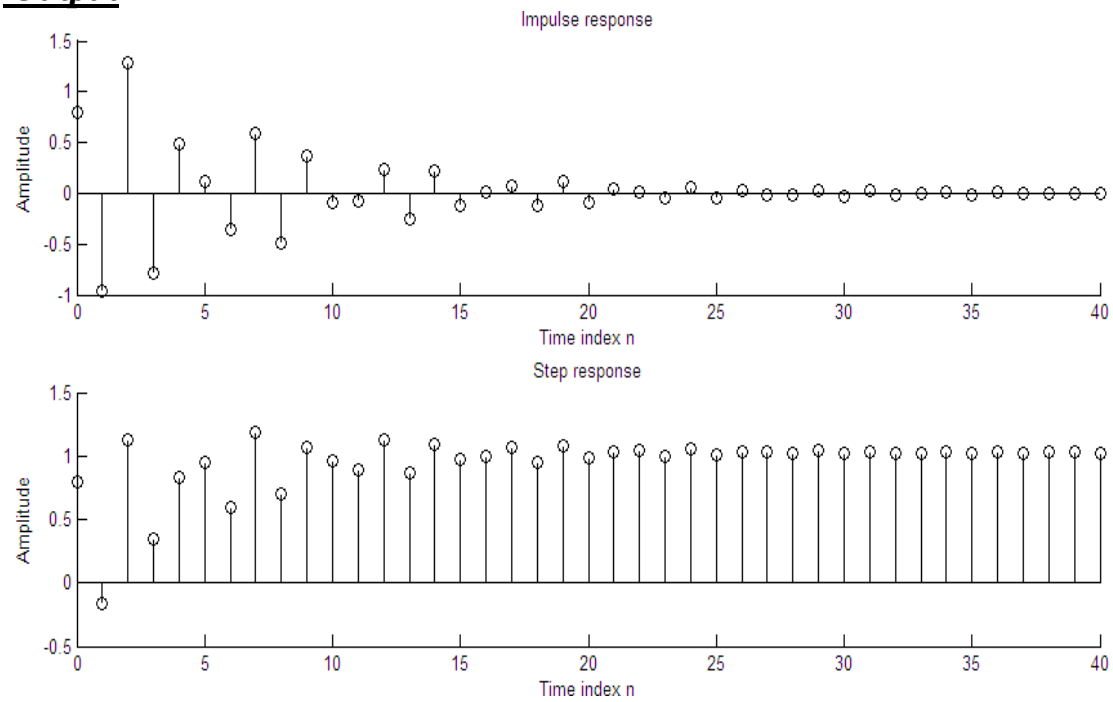
$$y(n) + 0.7\,y(n-1) - 0.45\,y(n-2) - 0.6\,y(n-3) = 0.8x(n) - 0.44x(n-1) + 0.36x(n-2) + 0.02x(n-3)$$

```
%--------------------------Getting  input  data----------------------------
N=input('Desired impulse response length=');
p=input('Enter the coefficients of x=');
d=input('Enter the coefficients of y=');
%---------------------- Program to compute impulse response----------------
x1=[1 zeros(1,N-1)];                          % generates impulse sequence
y1=filter(p,d,x1);                            % simulates causal LTI system
k=0:N-1;
subplot(2,1,1),stem(k,y1)
title('Impulse response');
xlabel('Time index n');ylabel('Amplitude')
%---------------------- Program to compute impulse response----------------
x2=[1 ones(1,N-1)];                           % generates step sequence
y2=filter(p,d,x2);
k=0:N-1;
subplot(2,1,2),stem(k,y2)
title('Step response');
xlabel('Time index n');ylabel('Amplitude')
```

Input

Desired impulse response length=41
Enter the coefficients of x=[0.8 -0.4 0.3 0.02]
Enter the coefficients of y=[1 0.7 -0.4 -0.6]

Output



Result

Thus the impulse and step responses of the causal finite dimensional LTI system are computed using Matlab.

Ex. No:4 Linear and Circular Convolution of Discrete Sequences
Date:

Objective:

To compute linear and circular convolution of two discrete sequences.

Description:

Convolution is a mathematical way of combining two signals to form a third signal. Convolution is a formal mathematical operation, just as multiplication and addition. Addition takes two numbers and produces a third number, while convolution takes two signals and produces a third signal. The convolution is the same operation as multiplying the polynomials whose coefficients are the elements of two signals.

Linear convolution:

The linear convolution of two signals *u(n)* and *v(n)* is given by

$$w(k) = \sum_{j=-\infty}^{\infty} u(j)v(k+1-j)$$

The output sample is simply a sum of products involving simple arithmetic operations such as additions, multiplications and delays. But practically *u(n)* and *v(n)* are finite in length. If the lengths of the two sequences being convolved are *m* and *n*, then the resulting sequence after convolution is of length *m+n-1* and is given by

$$w(k) = \sum_{j=-\max(1,\,k+1-n)}^{\min(k,m)} u(j)v(k+1-j)$$

When *m = n*, this gives

$$w(1) = u(1)v(1)$$

$$w(2) = u(1)v(2) + u(2)v(1)$$

$$w(3) = u(1)v(3) + u(2)v(2) + u(3)v(1)$$

$$\vdots$$

$$w(n) = u(1)v(n) + u(2)v(n-1) \;\ldots\; + u(n)v(1)$$

$$+ \qquad \vdots$$

$$w(2n-1) = u(n)v(n)$$

Circular convolution:

To find the circular convolution of two signals, the length of the sequences must be equal. If they are unequal, zeros are padded with the shorter sequence to make the length of the sequences equal. Then linear convolution like operation must be performed using a circular time reversal and then a circular time shift.

Matlab code:

```
%------------------------- Getting the input sequences--------------------
clc;clear all;close all;
g=input('ENTER THE FIRST SEQUENCE:');
h=input('ENTER THE SECOND SEQUENCE:');
subplot(2,2,1),stem(g);
title('First sequence');
xlabel('Time index n');ylabel('amplitude');
subplot(2,2,2),stem(h);
title('Second sequence');
xlabel('Time index n');ylabel('amplitude');
%-------------------- Program to compute linear convolution----------------
y=conv(g,h);
M=length(y)-1;
n=0:1:M;
disp('Output sequence of linear convolution=');disp(y)
subplot(2,2,3),stem(n,y);
xlabel('Amplitude');
ylabel('Time index n');
title('Linear convolution');
%-------------------- Program to compute circular convolution--------------
N1=length(g);
N2=length(h);
N=max(N1,N2);
n1=0:1:N;
N3=N1-
N2;
if(N3>=0)
   h=[h,zeros(1,N3)];
else
   g=[g,zeros(1,-N3)];
end
for n1=1:N
   y1(n1)=0;
   for i=1:N
      j=n1-i+1;
      if(j<=0)
         j=N+j;
      end
      y1(n1)=[y1(n1)+g(i)*h(j)];
   end
end
disp('Output sequence of circular convolution=');disp(y1)
subplot(2,2,4),stem(y1);
xlabel('Amplitude');
ylabel('Time index n');
title('Circular convolution');
```

Input:

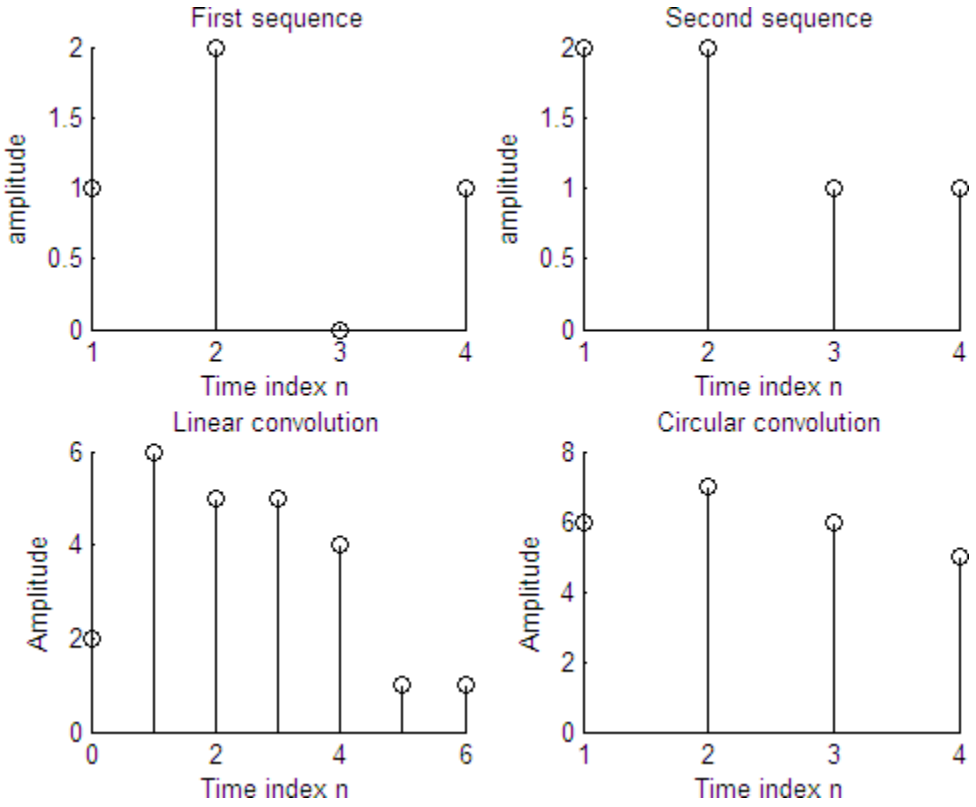
ENTER THE FIRST SEQUENCE: [1 2 0 1]
ENTER THE SECOND SEQUENCE: [2 2 1 1]
Output sequence of linear convolution=

2 6 5 5 4 1 1

Output sequence of circular convolution=

6 7 6 5

Output:



Result

Thus the linear and circular convolutions of two discrete sequences are computed using Matlab.

Ex. No:5 Correlation and Autocorrelation of Discrete Sequences
Date:

Objective:

To compute correlation and autocorrelation of two discrete sequences.

Description:

Correlation is basically used to compare two signals. Correlation measures the similarity between two signals. It is divided into two types namely cross correlation and auto correlation.

Cross correlation between a pair of signals *x(n)* and *y(n)* is given by

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

The parameter *l* called lag, indicates the time-shift between the pair. The time sequence *y(n)* is said to be shifted by *l* samples with respect to the reference sequence *x(n)* to the right for the positive values of *l*, and shifted by *l* samples to the left for negative values of *l*. The correlation process is essentially the convolution of two sequences in which one of the sequences has been reversed.

When the same signal is used for *x(n)* and *y(n)*, cross correlation becomes an autocorrelation.

Matlab Implementation:

Matlab provides a function called ***xcorr.m*** which may be used to implement both auto and cross correlation function.

Sinusoidal Signal Generation

Consider generating 64 samples of a sinusoidal signal of frequency 1kHz, with a sampling frequency of 8kHz. A sampled sinusoid may be written as:

$$x(n) = A \sin\left(2\pi \frac{f}{f_s} n + \phi\right)$$

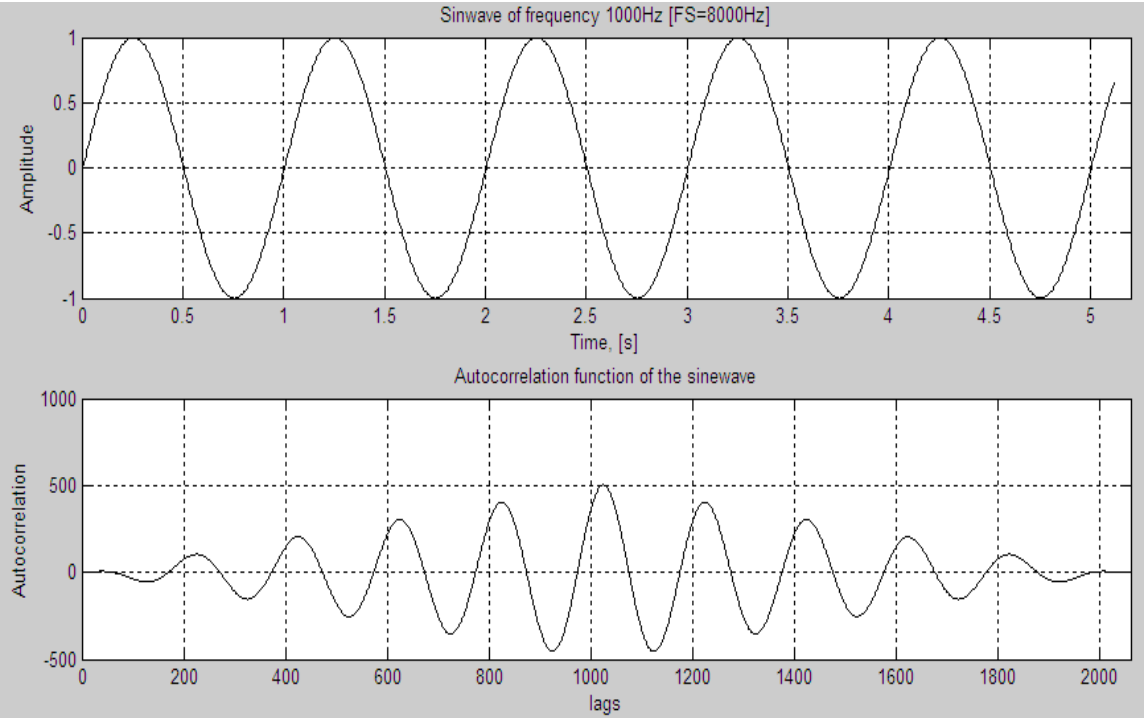
where f is the signal frequency, f_s is the sampling frequency, ϕ is the phase and A is the amplitude of the signal.

Matlab code for cross correlation:

The following program plots the crosscorrelation sequence of a sine wave with frequency 1 Hz, sampling frequency of 200 Hz and a cos wave with frequency 1 Hz, sampling frequency of 400 Hz.

```
%-------------------- Program to compute cross correlation----------
N=1024; % Number of samples to generate
f=1; % Frequency of the sinewave
FS=200; % Sampling frequency of sine wave
FS1=400;% Sampling frequency of cos wave
n=0:N-1; % Sampling index
x=sin(2*pi*f*n/FS); % Generate x(n)
y=cos(2*pi*f*n/FS1); % Generate y(n)
subplot(3,1,1);
plot(x);
title('Sine wave');
xlabel('Time index n');ylabel('Amplitude');
grid;
subplot(3,1,2);
plot(y);
title('Cos
wave');
xlabel('Time index n');ylabel('Amplitude');
grid;
Rxy=xcorr(x,y); % Estimate the cross correlation
subplot(3,1,3);
plot(Rxy);
title('Cross correlation Rxy');
xlabel('lags');ylabel('Crosscorrelation');
grid;
```

Output:

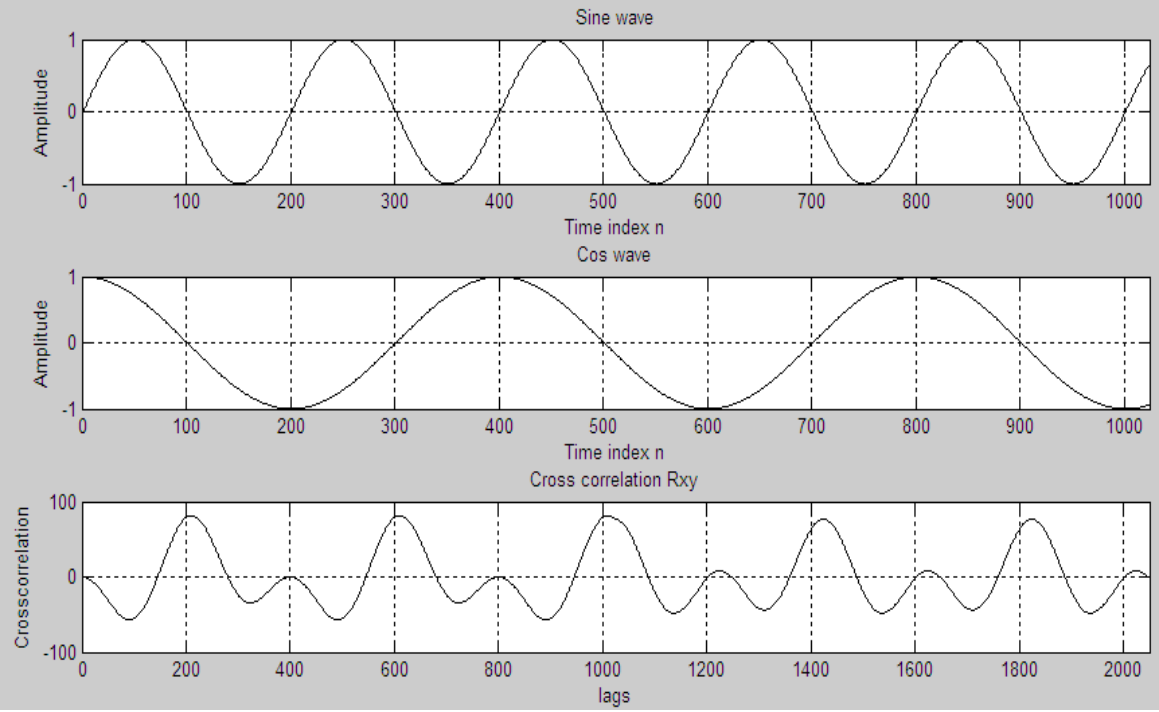


Matlab Code for Auto Correlation:

The following program plots the autocorrelation sequence of a sine wave with frequency 1 Hz, sampling frequency of 200 Hz.

```
%-------------------- Program to compute auto correlation----------
N=1024;                          % Number of samples
f1=1;                            % Frequency of the sinewave
FS=200;                          % Sampling Frequency
n=0:N-1;                         % Sample index numbers
x=sin(2*pi*f1*n/FS);             % Generate the signal, x(n)
t=[1:N]*(1/FS);                  % Prepare a time axis
subplot(2,1,1);                  % Prepare the figure
plot(t,x);                       % Plot x(n)
title('Sinwave of frequency 1000Hz
[FS=8000Hz]'); xlabel('Time,
[s]');ylabel('Amplitude');
grid;
Rxx=xcorr(x);                    % Estimate its autocorrelation
subplot(2,1,2);                  % Prepare the figure
plot(Rxx);                       % Plot the autocorrelation
grid;
title('Autocorrelation function of the sinewave');
xlable('lags');ylabel('Autocorrelation');
```

Output:



Result

Thus the cross correlation and the auto correlation are computed using Matlab.

Ex. No:6 Z- Transform and Inverse Z-Transform
Date:

Objective:

To draw pole-zero plot from the rational z-transform and to compute inverse z-transform of rational z-transform functions.

Description:

For a linear time-invariant (LTI) discrete-time system, the input-output relation is expressed in terms of a *difference equation* which consists of multiplications and summations. The z transform converts difference equations into algebraic equations, thereby simplifying the analysis of discrete time systems.

The z-transform converts a discrete time signal, which is a sequence of real and complex numbers in the time domain, into a complex frequency domain representation. The z-transform of a general discrete time signal *x(n)* is denoted as *X(z)* and is given by

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

where z is a complex variable and it is defined as $z = re^{j\omega}$

This is

defining the complex variable, *z*, as the polar notation combination of the two real variables, *r* (magnitude of z) and ω (angle of z).

In specifying the z-transform of a signal, only the algebraic expression is not enough. The range of values of z for which the z-transform equation is valid also required. In general, the range of values of z for which the summation in z-transform equation attains finite value is referred to as the *region of convergence* (ROC) of the z-transform.

The structure of z-transform is the ratio of two polynomials. So the z-transform is rational function, (i.e. a ratio of two polynomials) will have the following structure

$$X(z) = \frac{P(z)}{Q(z)}$$

The two polynomials, P (z) and Q (z), allow us to find the poles and zeros of the z-ransform. The roots of the numerator polynomial are known as *poles*, whereas the roots of the denominator polynomial are known *as zeros.* Formally, the poles are the value(s) of z where $X(z) = 0$ and the zeros are the value(s) of z where $X(z) = \infty$. The z-transform transforms a discrete time signal from the time domain to the z-domain. Likewise in signal processing, it is important to transform it from z-domain to the time domain. *Inverse z-transform* is the procedure used for this purpose.

Matlab Implementation:

The program given below finds poles, zeros and plots pole-zero plot for the rational z-transform

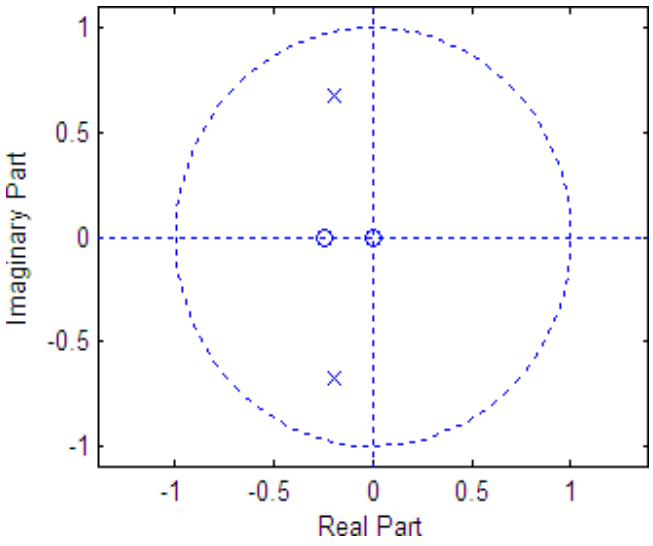
$$X(z) = \frac{z^2 + 0.25z}{z^2 + 0.4z + 0.5}$$

%Program to find poles, zeros and to plot pole-zero plot for the given rational z-transform

```
nu=[1 0.25 0];              % numerator coefficients in row vector
de=[1 0.4 0.5];             %denominator coefficients in row vector
[z,p,k] = tf2zp(nu,de);     %finds pole, zeros and gain constant
disp('Zeros')
disp(z)
disp('Poles')
disp(p)
zplane(nu,de)               %plots pole-zero plot
```

Output (Zeros and Poles in column vector)

```
Zeros
         0
   -0.2500

Poles
  -0.2000 + 0.6782i
  -0.2000 - 0.6782i
```



The program given below finds the inverse z-transform of a rational function

$$X(z) = \frac{z^2 + z - 3}{2z^2 - 2z + 3}; \quad ROC\, z > 1$$

using power series expansion method when the ROC is corresponding to the **causal** sequence

%Program to find the inverse z-transform of a rational function using power series %expansion method when the ROC is corresponding to the **causal** sequence

```
nu=[1 1 -3];                    % numerator coefficients in the descending powers of z
de=[2 -2 3];                    % denominator coefficients in the descending powers of z
L=6;                            % number of coefficients required
[x,t]=impz(nu,de,L);            % computes the coefficients
disp('The coefficients of the sequence')
disp(x')                        % displays the coefficients as a row vector
disp('Time index of the coefficients')
disp(t')                        % displays the time index as a row vector
```

Output

```
The coefficients of the sequence
0.5000    1.0000   -1.2500   -2.7500   -0.8750    3.2500

Time index of the coefficients
0     1     2     3     4     5
```

The program given below finds the inverse z-transform of a rational function

$$X(z) = \frac{z^2 + z - 3}{2z^2 - 2z + 3}; \quad ROC\ z < 1$$

using power series expansion method when the ROC is corresponding to the **anticausal** sequence

%Program to find the inverse z-transform of a rational function using power series %expansion method when the ROC is corresponding to the **anticausal** sequence

```
nu=[-3 1 1];                    % numerator coefficients in the ascending powers of z
de=[3 -2 2];                    % denominator coefficients in the ascending powers of z
L=6;                            % number of coefficients required
[x,t]=impz(nu,de,L);            % computes the coefficients
disp('The coefficients of the sequence')
disp(x')                        % displays the coefficients as a row vector
t1=-t;                          % time index for the anticausal sequence
disp('Time index of the coefficients')
disp(t1')                       % displays the time index as a row vector
```

Output

The coefficients of the sequence
-1.0000 -0.3333 0.7778 0.7407 -0.0247 -0.5103

Time index of the coefficients
0 -1 -2 -3 -4 -5

Result:

Thus pole-zero plot has been drawn from the rational z-transform and the inverse z-transform of rational z-transform function has been obtained.

Ex. No:7 **Computation of DFT of a Signal**
Date:

Objective:

To compute discrete Fourier transform of a signal using fast Fourier transform.

Description:

Fourier analysis is extremely useful for data analysis, as it breaks down a signal into constituent sinusoids of different frequencies. For sampled vector data, Fourier analysis is performed using the discrete Fourier transform (DFT). The fast Fourier transform (FFT) is an efficient algorithm for computing the DFT of a sequence; it is not a separate transform. It is particularly useful in areas such as signal and image processing, where its uses range from filtering, convolution, and frequency analysis to power spectrum estimation.

For the N input sequence $\boldsymbol{x}$, the DFT is a vector $\boldsymbol{X}$ of length N. DFT and IDFT implement the relationships

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi k}{N}n}; \qquad k = 0,1,\cdots,N-1$$

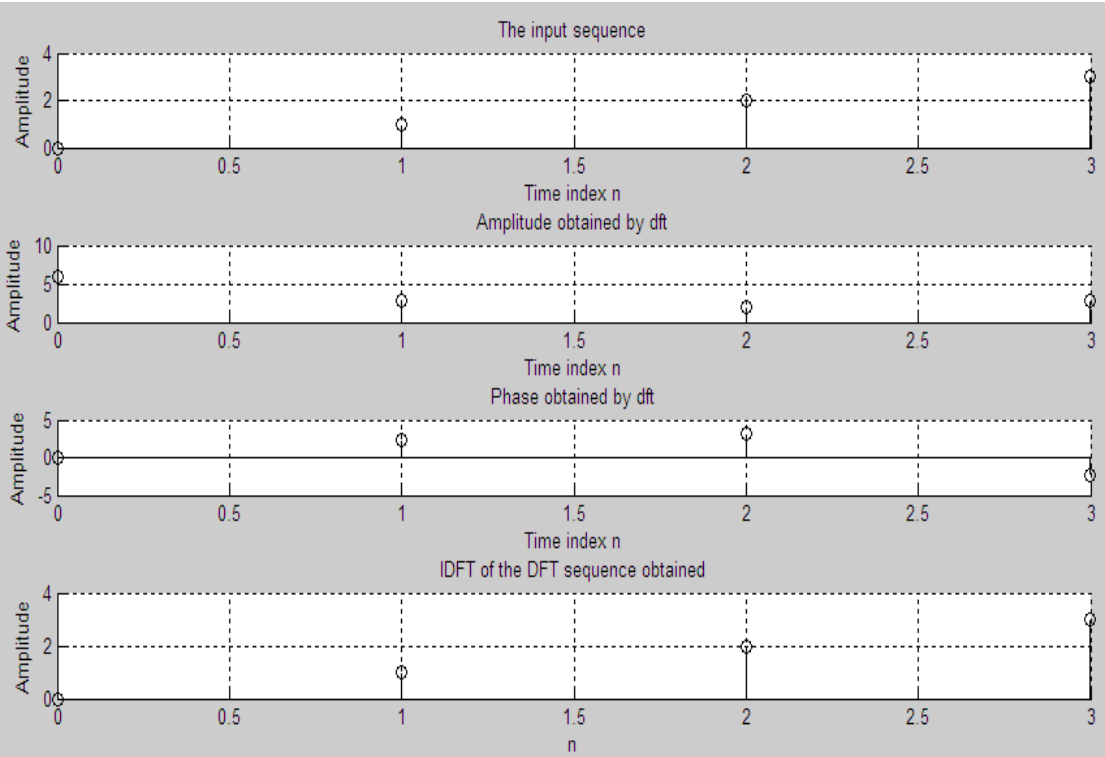
$$x(n) = \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi k}{N}n}; \qquad n = 0,1,\cdots,N-1$$

Matlab Implementation:

```
clc;clear all;close all;
%-------------------------Computation of DFT-----------------------------
a=input('ENTER THE SEQUENCE:');
l=length(a);
x=fft(a,l);
disp('DFT of the given sequence=');disp(x)
n=0:l-1;
subplot(4,1,1),stem(n,a);
title('The input sequence');
xlabel('Time index n');ylabel('Amplitude');
grid;
%----------------Plotting amplitude and phase of DFT sequence--------------
subplot(4,1,2),stem(n,abs(x));
xlabel('Time index n');
ylabel('Amplitude');
title('Amplitude obtained by dft');
grid;
subplot(4,1,3);stem(n,angle(x));
xlabel('Time index n'); ylabel('Amplitude');
title('Phase obtained by dft');
grid;
%-------------------------Computation of IDFT-----------------------------
a1=ifft(x,l);
disp('IDFT of the DFT sequence=');disp(a1)
subplot(4,1,4),stem(n,a1);
title('IDFT of the DFT sequence obtained');
xlabel('n');ylabel('Amplitude');
grid;
```

Output:

```
ENTER THE SEQUENCE:[0 1 2 3]
DFT of the given sequence=
   6.0000          -2.0000 + 2.0000i          -2.0000          -2.0000 - 2.0000i

IDFT of the DFT sequence=
     0     1     2     3
```



Result

Thus the DFT and IDFT of a discrete sequence are computed using Matlab.

Ex. No:8 **Sampling Rate Alteration of a Signal**
Date:

Objective:

To alter the sampling rate of a signal using up-sampling and down-sampling

Description:

There are many applications where the signal of a given sampling rates needs to be converted into an equivalent signal with a different sampling rate. To achieve different sampling rates at different stages, multirate digital signal processing systems employ the down-sampler and the up-sampler.

An up-sampler with an up-sampling factor L, where L is a positive integer, develops an output sequence $x_u[n]$ with a sampling rate that is L times larger than that of the input sequence $x[n]$.

The up-sampling operation is implemented by inserting L-1 equidistant zero-valued samples between the consecutive samples of the input sequence $x[n]$ according to the relation

$$x_u[n] = \begin{cases} x[n/L], & n = 0, \pm L, \pm 2L, \ldots \\ 0, & \text{otherwise} \end{cases}$$

The down-sampler with a down-sampling factor M, where M is a positive integer, develops an output sequence $x_d[n]$ with a sampling rate that is $(1/M)$th of that of the input sequence $x[n]$. The down-sampling operation is implemented by keeping every M^{th} sample of the input sequence and removing $M-1$ in-between samples, to generate the output sequence $x_d[n]$ according to the relation

$$x_d[n] = x[nM]$$

As a result, all input samples with indices equal to an integer multiple of M are retained at the output and all others are discarded.

Matlab Implementation:

Up-Sampling:

```
%-----------Illustration of Up-Sampling by an Integer Factor----------------
clf;
L=input('Up sampling factor= ');
%---------- Generate the input sinusoidal sequence------
n = 0:50;                % Input length
x = sin(2*pi*0.12*n);
%---------- Generate the up-sampled sequence------------
y = zeros(1, L*length(x));
y([1: L: length(y)]) = x;
%---------- Plot the input and output sequences--------
subplot(2,1,1)stem(n,x);
title('Input Sequence');
xlabel('Time index n');ylabel('Amplitude');
subplot(2,1,2)
stem(n,y(1:length(x)));
title('Output Sequence');
xlabel('Time index n');ylabel('Amplitude');
```

Input:

Up sampling factor= 3

Output:

Down-Sampling:

```
%-----------Illustration of Down-Sampling by an Integer Factor---------------
clf;clear all;close all;
M=input('Down sampling factor= ');
%---------- Generate the input sinusoidal sequence------
N=50;           % Input length
n = 0:N-1;
m = 0: N*M - 1;
x = sin(2*pi*0.042*m);
%---------- Generate the down-sampled sequence------------
d = x([1: M: length(x)]);
%---------- Plot the input and output sequences--------
subplot(2,1,1)
stem(n,x(1:N));
title('Input Sequence');
xlabel('Time index n');ylabel('Amplitude');
subplot(2,1,2)
stem(n,d);
title('Output Sequence');
xlabel('Time index n');ylabel('Amplitude');
```

Input:

Down sampling factor= 3

Output:

Result

Thus the sampling rate of the signal is altered using up-sampling and down-sampling.

Ex. No:9 **Design of IIR Filters**
Date:

Objective:

Designing of various types of IIR filters by Matlab commands.

Description:

A filter is one, which rejects unwanted frequencies from the input signal and allows the desired frequencies to obtain the required shape of output signal.
The filters are of different types based on their frequency response

1. Low pass filter
2. High pass filter
3. Band pass filter
4. Band stop filter

Low pass filter:
A low pass filter allows lower frequencies in between 0 and cut off frequency (f_c) and blocks the frequencies greater than the cut off frequency.

High pass filter:
A high pass filter allows higher frequencies greater than the cut off frequency (f_c) and blocks the frequencies in between 0 and cut off frequency.

Band pass filter:
It allows a band of frequencies f_{cl} to f_{ch} to pass and stops all other frequencies.

Band stop filter:
It rejects a band of frequencies f_{cl} to f_{ch} and allows all other frequencies.

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

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

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

There are two types of digital filters based on their impulse response.

1. IIR (infinite impulse response) filter
2. FIR (finite impulse response) filter

There are two important design techniques to design IIR filtersButterworth

1. Chebyshev

Filter specifications:
Wp = normalized pass band edge frequency
Ws = normalized stops band edge frequency
The frequency points must be between 0 and 1.
Rp = pass band ripple in dB
Rs = minimum stop band attenuation in dB

The IIR digital filter design process involves two steps.

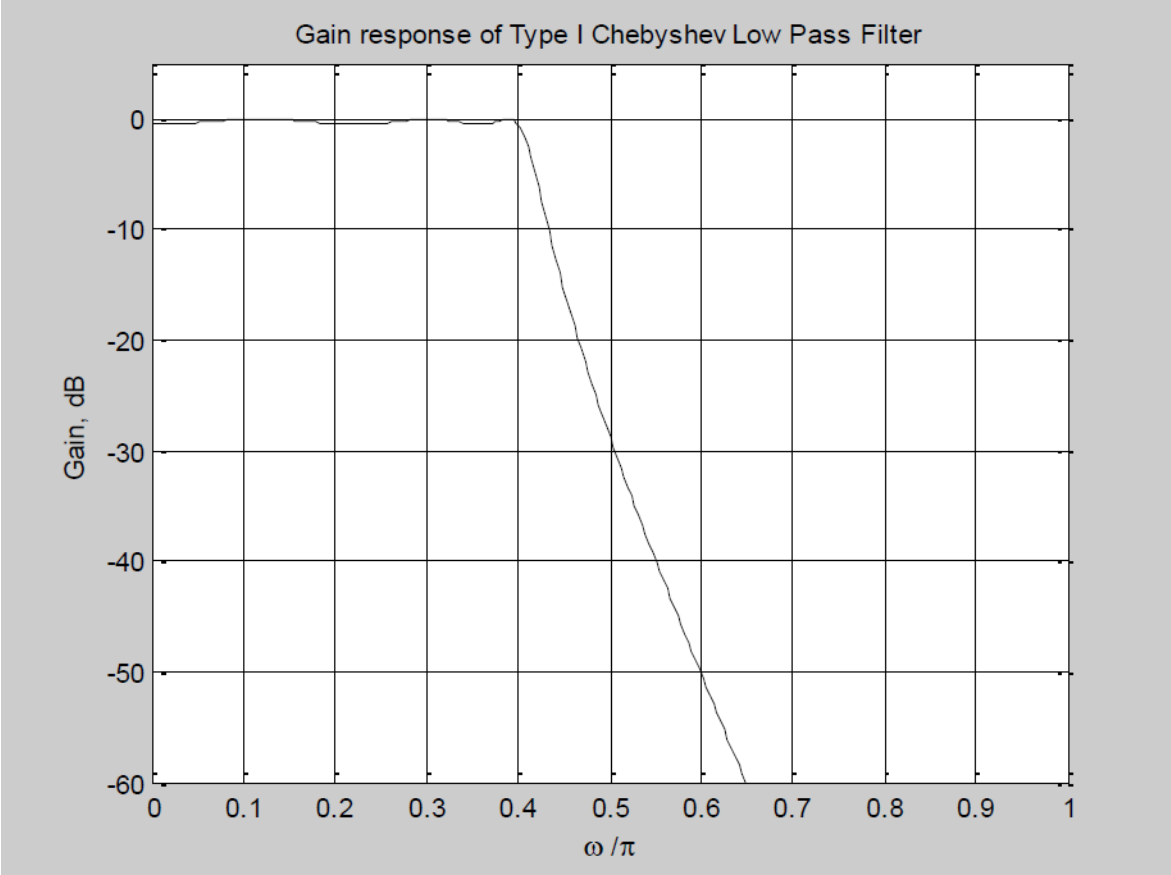
1. Finding the filter order
2. Determining the frequency scaling factor Wn from the given specification

Matlab Implementation:

The following program designs a **Type 1 Chebyshev IIR low pass filter** with normalized pass band edge=0.4 Hz, normalized stop band edge=0.6 Hz, pass band ripple =0.5dB, and minimum stop band attenuation of 40 dB.

```
%---------Design of a Type I Chebyshev Low Pass Digital Filter------------
clc;clear all;close all;
Ws = 0.6; Wp = 0.4; Rp = 0.5; Rs = 40;
%---------------------Estimate the Filter Order----------------
[N, Wn] = cheb1ord(Wp, Ws, Rp, Rs);
%------------------- Design the Filter-------------------------
[num,den] = cheby1(N,Rp,Wn,'low');
%---------------------Compute the gain-------------------------
[h,omega] = freqz(num,den,256);
gain=20*log10(abs(h));
%------------------Plot the gain response----------------------
plot(omega/pi,gain);grid
axis([0 1 -60 5]);
xlabel('\omega /\pi'); ylabel('Gain, dB');
title('Gain response of Type I Chebyshev Low Pass Filter');
```

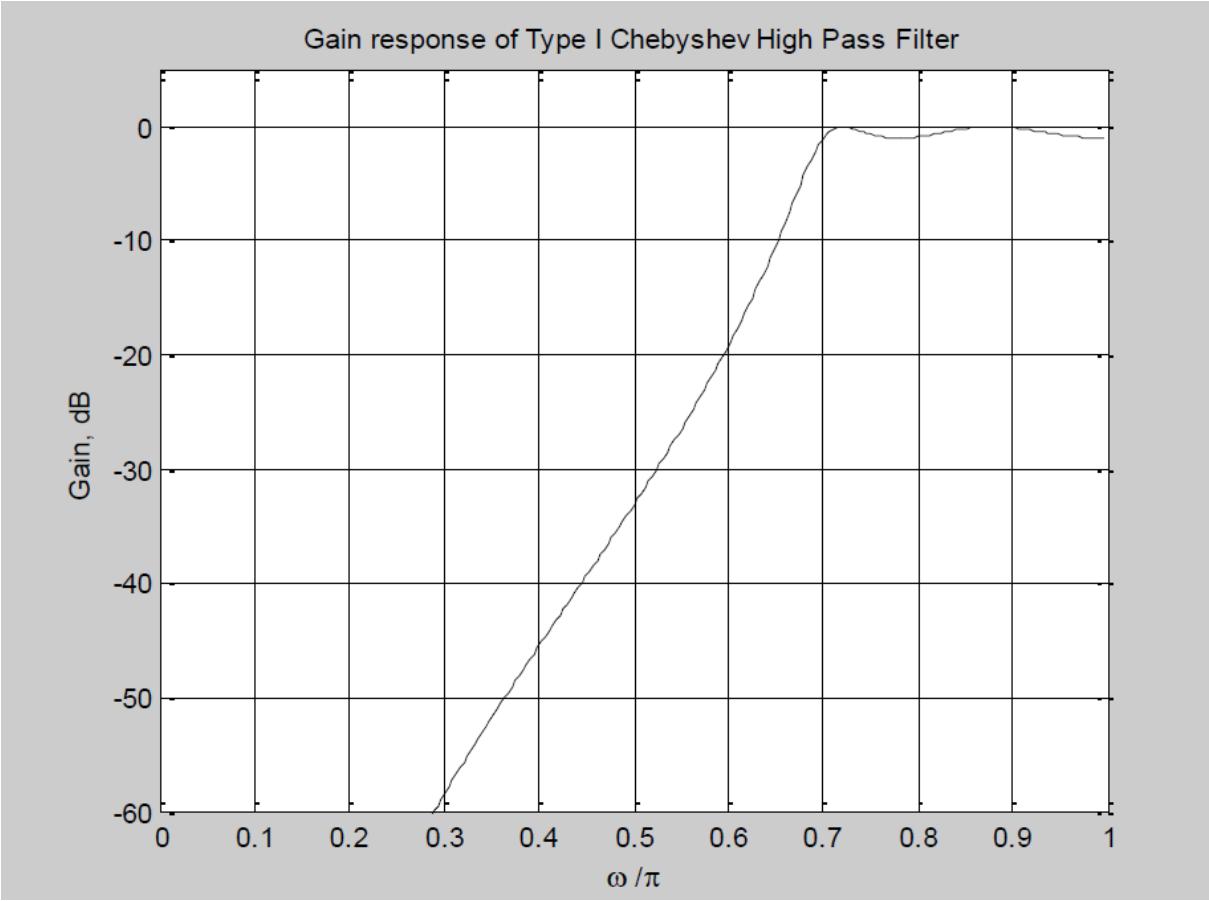
Output:



The following program designs a **Type 1 Chebyshev IIR high pass filter** with normalized pass band edge=0.7Hz, normalized stop band edge=0.5 Hz, pass band ripple =1dB, and minimum stop band attenuation of 32 dB.

```
%---------Design of a Type I Chebyshev High Pass Digital Filter------------
clc;clear all;close all;
Ws = 0.5; Wp = 0.7; Rp = 1; Rs = 32;
%---------------------Estimate the Filter Order----------------
[N, Wn] = cheb1ord(Wp, Ws, Rp, Rs);
%------------------- Design the Filter-------------------------
[num,den] = cheby1(N,Rp,Wn,'high');
%---------------------Compute the gain-------------------------
[h,omega] = freqz(num,den,256);
gain=20*log10(abs(h));
%-----------------Plot the gain response----------------------
plot(omega/pi,gain);grid
axis([0 1 -60 5]);
xlabel('\omega /\pi'); ylabel('Gain, dB');
title('Gain response of Type I Chebyshev High Pass Filter');
```

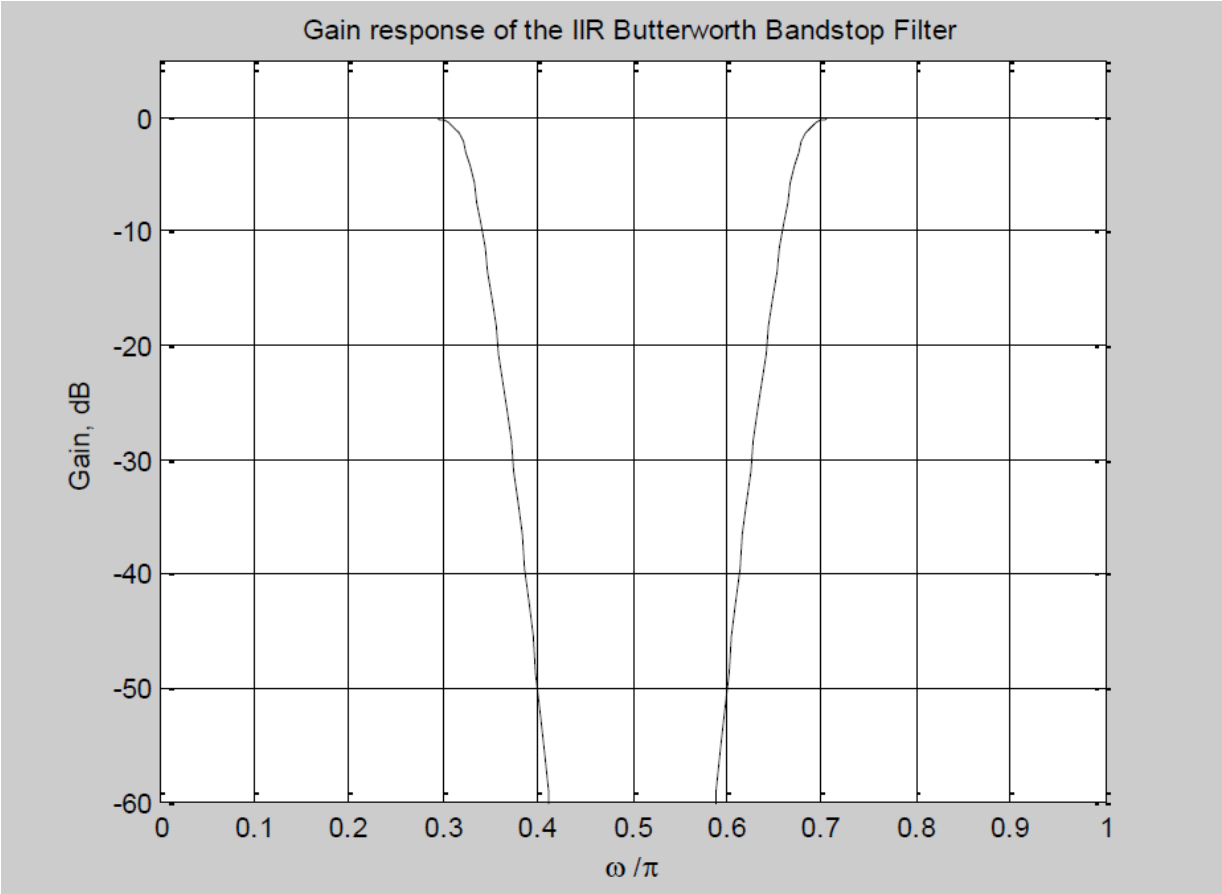
Output:



The following program designs a **Butterworth IIR Band pass filter** with normalized pass band edges=0.45 and 0.65, normalized stop band edge=0.3 and 0.75, pass band ripple =1dB, and minimum stop band attenuation of 40 dB.

```
%--------------Design of a Butterworth Band Pass Digital Filter-----------
clc;clear all;close all;
Ws = [0.3 0.75]; Wp = [0.45 0.65]; Rp = 1; Rs = 40;
%--------------------Estimate the Filter Order----------------
[N, Wn] = buttord(Wp, Ws, Rp, Rs);
%------------------- Design the Filter-------------------------
[num,den] = butter(N,Wn);
%---------------------Compute the gain-------------------------
[h,omega] = freqz(num,den,256);
gain=20*log10(abs(h));
%-----------------Plot the gain response---------------------
plot(omega/pi,gain);grid
axis([0 1 -60 5]);
xlabel('\omega /\pi'); ylabel('Gain, dB');
title('Gain response of the IIR Butterworth Bandpass Filter');
```

Output:

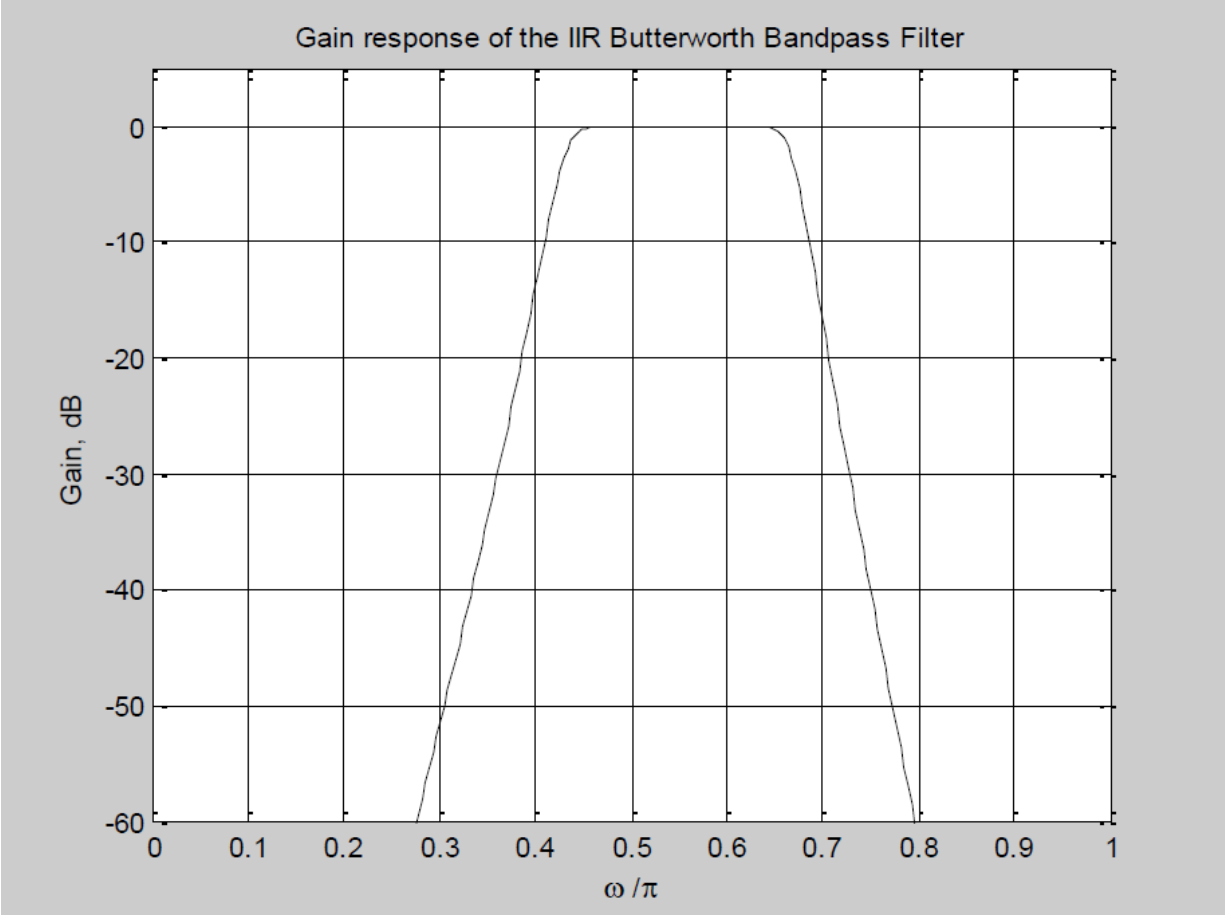


The following program designs a **Butterworth IIR Band stop filter** with normalized pass band edges=0.3 and 0.7, normalized stop band edge=0.4 and 0.6, pass band ripple =0.4dB, and minimum stop band attenuation of 50 dB.

```
%--------------Design of a Butterworth Bandstop Digital Filter-----------
clc;clear all;close all;
Ws = [0.4 0.6]; Wp = [0.3 0.7]; Rp = 0.4; Rs = 50;
%--------------------Estimate the Filter Order----------------
[N, Wn] = buttord(Wp, Ws, Rp, Rs);
%------------------- Design the Filter-------------------------
[num,den] = butter(N,Wn,'stop');
%--------------------Compute the gain-------------------------
[h,omega] = freqz(num,den,256);
gain=20*log10(abs(h));
%------------------Plot the gain response---------------------
plot(omega/pi,gain);grid
axis([0 1 -60 5]);
```

```
xlabel('\omega /\pi'); ylabel('Gain, dB');
title('Gain response of the IIR Butterworth Bandstop Filter');
```

Output:



Result

Thus Butterworth and Chebyshev IIR filters are designed for the given specifications and the frequency responses are displayed.

Ex. No:10 Design of FIR Filters

Objective:

Designing of various types of finite impulse response (FIR) filters using Matlab commands.

Description:

The FIR filters are of non-recursive type. In these filters, the present output sample depends on the present and previous input samples only.

There are four important design techniques to design FIR filters namely

1. Windowing method
2. Frequency sampling method
3. Weighted least squares design
4. Equiripple method

Only the windowing method is presented here.

***Windowing method*:**

Steps to design FIR filters using windows.

1. Obtain the specification of the filter.
2. Obtain the desired impulse response of the filter.
3. Select the type of the window.
4. Multiply the desired impulse response and the window sequence.
5. Find the frequency response for this windowed sequence.

The various types of window are given below

Rectangular window:

$$w(n) = 1 \quad for -\frac{N-1}{2} \le n \le \frac{N-1}{2}$$

Hamming window:

$$w(n) = 0.54 + 0.46\cos\left(\frac{2\pi n}{N-1}\right) \quad for -\frac{N-1}{2} \le n \le \frac{N-1}{2}$$

N

Hanning window: $2\pi n^{-1}$ $\left(N-1\right)^{2}$ $\left(\quad\right)$ $\left(\quad\right)^{2}$ $\left(\quad\right)$ $N-1$

$for-\left(\quad\right)\leq n\leq\left(\quad\right)$

$w(n)=0.5+$

$0.5\cos$

$N-1$ $\left(\quad\right)^{2}$ $\left(\quad\right)$ $\left(\quad\right)^{2}$ $\left(\quad\right)$

Blackman window: $2\pi n$ $+0.08$ $\left(4\pi n\right)\cdot$ for $-\left(N-1\right)$ $N-1$

$\leq n\leq\left(\quad\right)$

$w(n)=0.42+0.5$

$\cos$ $\cos$

2 2

$N-1$ $N-1$ $\left(\quad\right)$ $\left(\quad\right)$

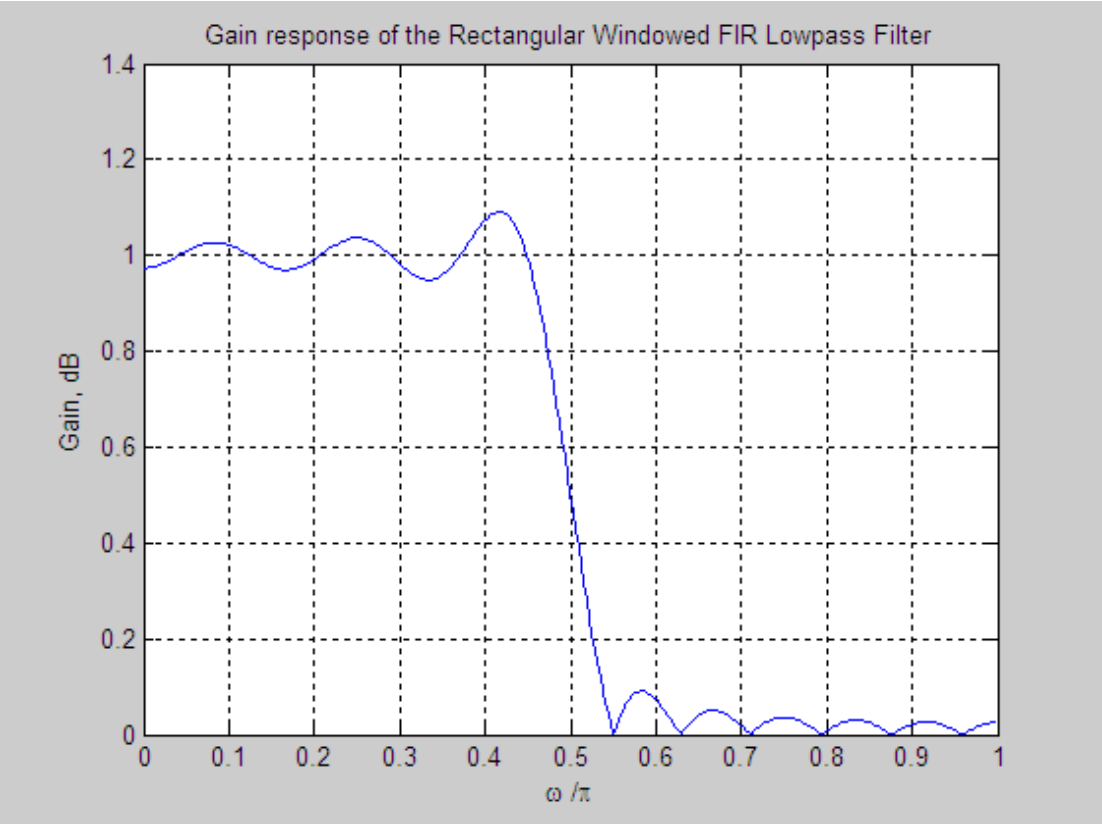
$\left(\quad\right)$

Matlab Implementation:

The following program designs a low pass filter with cutoff frequency 0.5π radians using rectangular window and plots its frequency response.

```
clear all
wc=.5*pi;                                   %Cut off frequency
N=25;
alpha=(N-1)/2;
eps=.001;                                   %To avoid intederminate form
n=0:1:N-1;
hd=sin(wc*(n-alpha+eps))./(pi*(n-alpha+eps));   %Desired impulse response
wr=boxcar(N);                               %Rectangular window
sequence
hn=hd.*wr';                                 %Filter coefficients
w=0:.01:pi;
h=freqz(hn,1,w);                            %Computes frequency response
plot(w/pi,abs(h));
xlabel('\omega / \pi'); ylabel('Gain, dB');
title('Gain response of the Rectangular Windowed FIR Lowpass Filter');
```

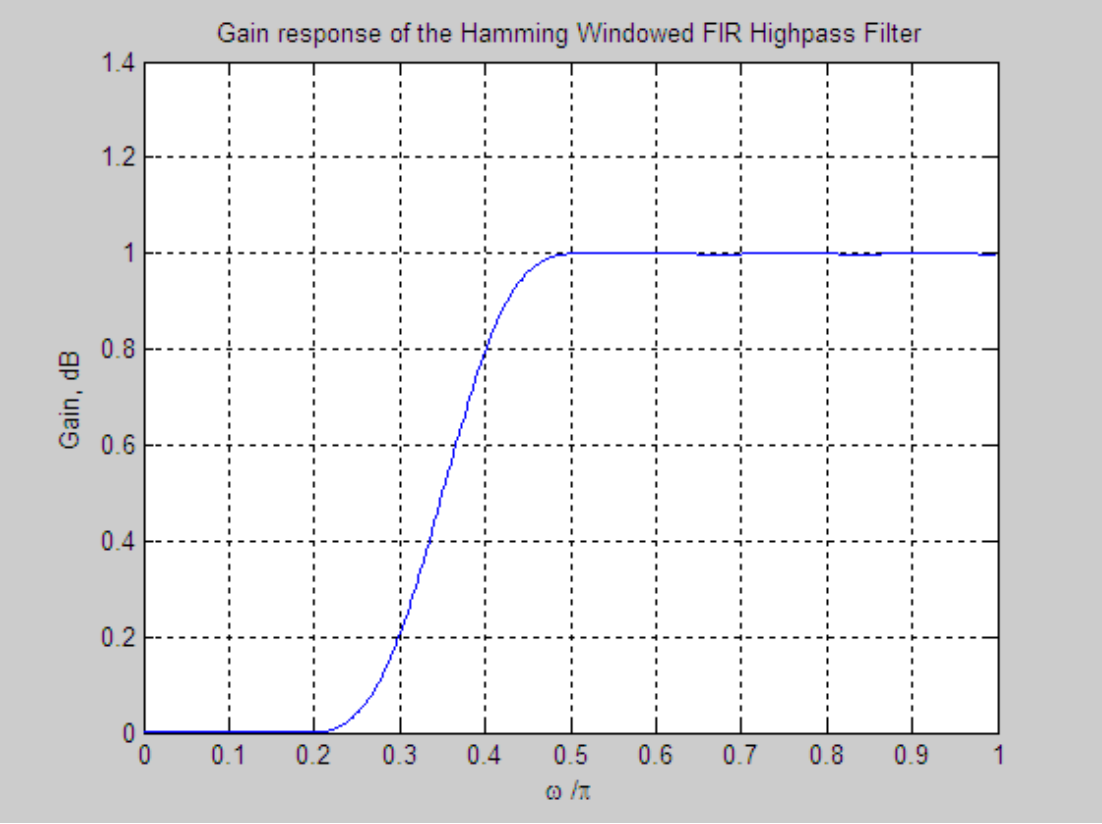
Output:



The following program designs a high pass filter with cutoff frequency 0.35π radians using hamming window and plots its frequency response.

```
clear all
wc=.35*pi;                                          %Cut off frequency
N=25;
alpha=(N-1)/2;
eps=.001;                                           %To avoid intederminate form
n=0:1:N-1;
hd=(sin(pi*(n-alpha+eps))-sin(wc*(n-alpha+eps)))./(pi*(n-alpha+eps));       %Desired
                                                                            %impulse response

wr=hamming(N);                                      %Hamming window
hn=hd.*wr';                                         %Filter coefficients
w=0:.01:pi;
h=freqz(hn,1,w);                                    %Computes frequency response
plot(w/pi,abs(h));
grid;
xlabel('\omega / \pi'); ylabel('Gain, dB');
title('Gain response of the Hamming Windowed FIR Highpass Filter');
```

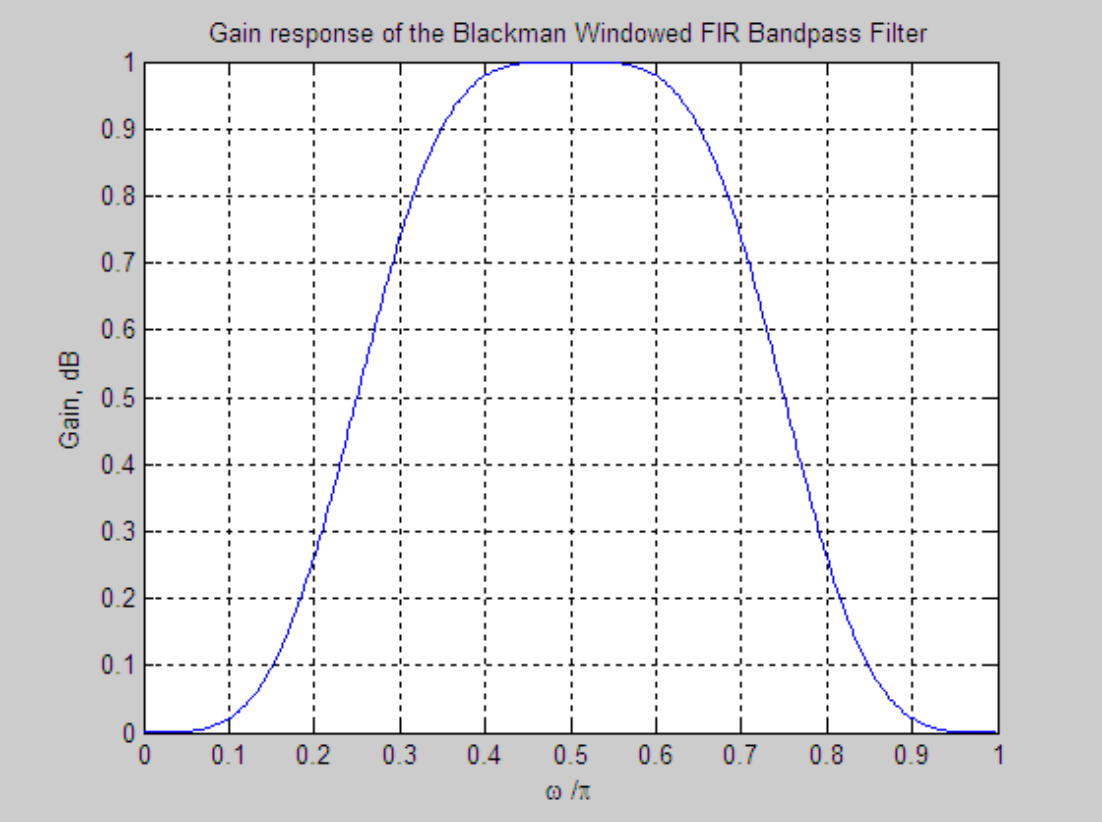
Output:



The following program designs a band pass filter with cutoff frequencies 0.25π and 0.75π radians using Blackman window and plots its frequency response.

```
clear all
wc1=.25*pi;wc2=.75*pi;                                  %Cut off frequencies
N=25;
a=(N-1)/2;
eps=.001;                                               %To avoid intederminate form
n=0:1:N-1;
hd=(sin(wc2*(n-a+eps))-sin(wc1*(n-a+eps)))./(pi*(n-a+eps));      %Desired impulse
                                                                 % response
wr=blackman(N);                                         %Blackman window
hn=hd.*wr';                                             %Filter coefficients
w=0:.01:pi;
h=freqz(hn,1,w);                                        %Computes frequency response
plot(w/pi,abs(h));
grid;
xlabel('\omega /\pi'); ylabel('Gain, dB');
title('Gain response of the Blackman Windowed FIR Bandpass Filter');
```

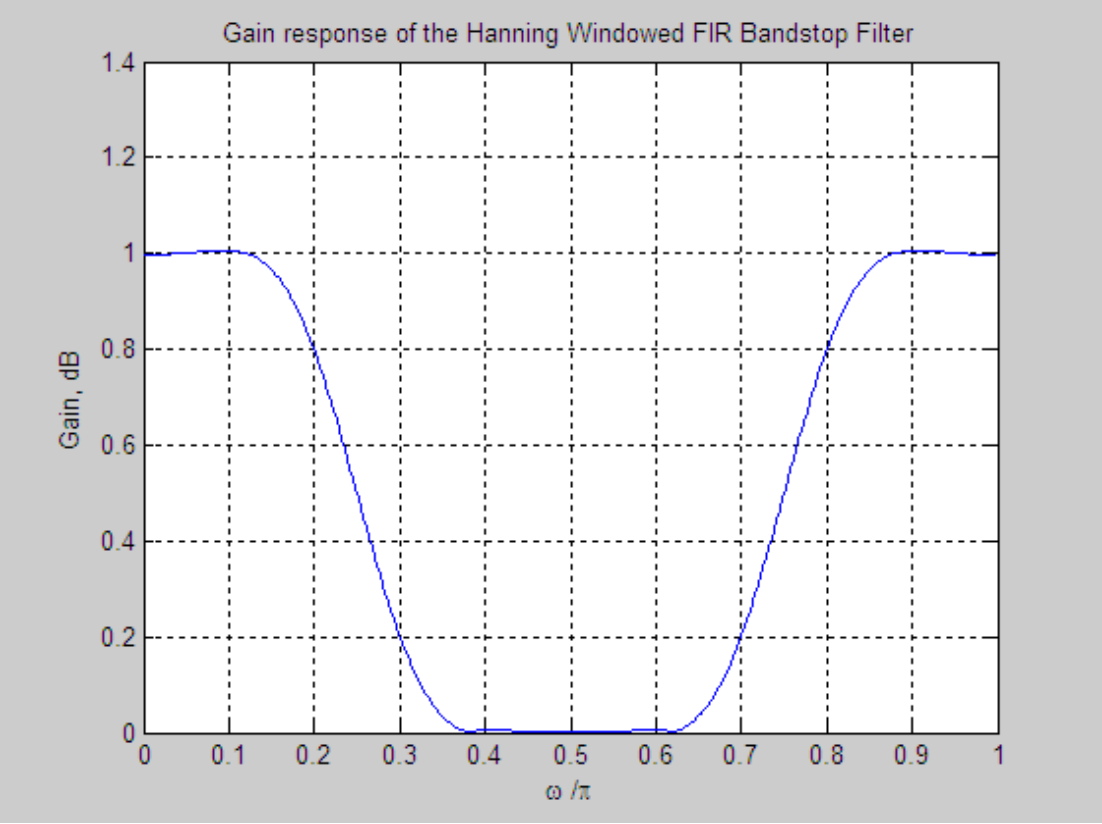
Output:



The following program designs a band stop filter with cutoff frequencies 0.25π and 0.75π radians using hanning window and plots its frequency response.

```
clear all
wc1=.25*pi;wc2=.75*pi;                                  %Cut off frequencies
N=25;
a=(N-1)/2;                                              %To avoid intederminate form
eps=.001;
n=0:1:N-1;
%Desired impulse response
hd=(sin(wc1*(n-a+eps))-sin(wc2*(n-a+eps))+sin(pi*(n-a+eps)))./(pi*(n-a+eps));
wr=hanning(N);                                          %Hanning window
hn=hd.*wr';                                             %Filter coefficients
w=0:.01:pi;
h=freqz(hn,1,w);                                %Computes frequency response
plot(w/pi,abs(h));
grid;
xlabel('\omega /\pi'); ylabel('Gain, dB');
title('Gain response of the Hanning Windowed FIR Bandstop Filter');
```

Output:



Result

Thus FIR filters are designed using various for the given specifications and the frequency responses are displayed.

Ex No. 11 **Determination of various entropies and mutual information for a binary symmetric channel**

Aim: Write a program for determination of various entropies and mutual information of a given channel. (Binary symmetric channel).

Description:

1. Explain in detail BSC with neat diagram.
2. Find capacity of BSC channel.

Algorithm: **Entropies:**

1. Input the no. of inputs of a channel.
2. Input the no. of outputs of a channel.
3. Input the channel matrix. Test the condition that sum of all the entries in each row should be equal to 1.
4. Input the channel input probabilities. i.e. P[X].
5. Calculate the entropy of the channel input. i.e. H(X)
6. Calculate output probability matrix P[Y], by multiplying input probability matrix by channel matrix.
7. Also calculate entropy of channel output. i.e. H(Y).
8. Convert input probability matrix into diagonal matrix.i.e. P[X]d
9. Calculate the joint probability matrix by multiplying input probability matrix in diagonal form by channel matrix.
10. Calculate joint entropy with the help of formula
11. Calculate conditional entropies H(Y/X)&H(X/Y).
12. Also we can calculate mutual information as I(X;Y)=H(X)-H(X/Y) or I(X;Y)=H(Y)-H(Y/X)

Binary Symmetric Channel

- BSC channel is characterized by No.of input = No. of output = 2 2.Conditional probability matrix is as follows

$$P(Y/X)=\begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}$$

- Derive the joint probability matrix from this matrix by multiplying it by P(X) =[0 1]
- So the matrix which we take input from user is

$$P(X,Y)=\begin{bmatrix} 0 & 0 \\ p & 1-p \end{bmatrix}$$

- Where p should be entered by the user.
- Then repeat steps 4 to 8 to calculate all the required quantities.

MatLab Code:

```
%% Program for entropy and MI for BSC

clc;
```

```
clear all;
close all;
i=input('Enter no. of elements='); p=input('Enter probability=');
q=input('Enter conditional probabilities matrix='); sum=0;

% Entropy H(x)
for n=1:i
   H=sum+(p(n)*log2(1/p(n))); sum=H;
end
disp('H(x): '); disp(H);

% Joint probability matrix
for n=1:i
   for m=1:i
      a(n,m)=q(n,m)*p(n);
   end
end
disp('P(X,Y):');
disp(a);

% Entropy H(Y/X)
d=0;
for n=1:i
   for m=1:i
      H1=d+(a(n,m)*log2(1/q(n,m)));
      d=H1;
   end
end
disp('H(Y/X):'); disp(H1);

% Probability P(Y)
for n=1:i
   w=0;
   for m=1:i
      s(n)=w+a(m,n); w=s(n);
   end
end
disp('P(Y):');
disp(s);

% Entropy H(Y)
k=0;
for n=1:i
   H2=k+(s(n)*log2(1/s(n)));
   k=H2;
end
disp('H(Y): ');
```

```
disp(H2);

% Find Mutual Information
m=H2-H1;
disp('MI=');
disp(m);
```

Output:

```
Enter no. of elements=2
Enter probability=[3/4 1/4]
Enter conditional probabilities matrix=[1/3 2/3;2/3 1/3]
H(x):
   0.8113
P(X,Y):
   0.2500   0.5000
   0.1667   0.0833
H(Y/X):
   0.9183
P(Y):
   0.4167   0.5833
H(Y):
   0.9799
MI=
   0.0616
```

Results:

Ex No. 12 **Implementation of Shannon Fano coding**

Aim:

- To write a program for Shannon Fano coding to find the average number of bits, entropy, efficiency, redundancy for the given probabilities.

Description:

1 Explain Shannon Fano coding.

2 Solve theoretically and verify using matlab program the given example.

Algorithm:

1 Create a list of probabilities for the given set of symbols.
2 The message is first written in the decreasing order.
3 Then divide the messages into two most equiprobable subset x and y
4 The message of 1st subset x is bit 0 and message in the second subset is bit 1
5 The procedure is now applied for each set probability till end
6 Finally, we get the code word for respective symbol
7 Calculate.

MatLab Code:

```
clc
clear
L=['A' 'B' 'C' 'D' 'E' 'F' 'G']
PK=[0.3 0.2 0.2 0.1 0.1 0.05 0.05]
% L=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H']
% PK=[16/32 4/32 4/32 2/32 2/32 2/32 1/32 1/32]
[A,I]=sort(PK,'descend');
A=A';
SYMBOL=L(I)'
TABLE(:,1)=A;
TABLE(:,2)=0;
no=2;
converge=0;
FLAG=zeros(length(A),1);
No_bits=zeros(length(A),1);
while converge==0
loop=1;
start=0;
k=[];
for i=1:length(TABLE(:,no))
if TABLE(i,2)==0
if start==0
k(loop,i)=i;
start=1;
elseif TABLE(i-1,no)==TABLE(i,no)
k(loop,i)=i;
else
```

```
loop=loop+1;
k(loop,i)=i;
end
end
end
if start==1
for L=1:loop
I=[];
I=k(L,:);
I(find(I==0))=[];
DATA=A(I);
[SS]=group_data(DATA);
if length(SS)<=2
FLAG(I)=1;
TABLE(:,2)=FLAG;
end
if length(SS)==1
No_bits(I)=no-2;
elseif length(SS)==2
No_bits(I)=no-1;
end
TABLE(I,no+1)=SS';
[TABLE No_bits]
end
no=no+1
else
converge=1;
end
end
TABLE(:,2)=[];
[TABLE No_bits]
% Average Code Word Length
sss=0;
Entropy=0;
for i=1:length(A)
sss=sss+A(i)*No_bits(i);
Entropy=Entropy+A(i)*log2(1/A(i));
end
Ave_Code_Word_Length=sss
Entropy=Entropy
Coding_Eff=Entropy/Ave_Code_Word_Length
Coding_Redun=1-Coding_Eff
% Code Variance
Code_Variance=0;
for i=1:length(A)
sss=sss+A(i)*(No_bits(i)-Ave_Code_Word_Length)^2;
end
```

```
function [SS]=group_data(PP)
if length(PP)>2
SS=0;
for i=1:length(PP)-1
T(i,1)=sum(PP(1:i));
T(i,2)=sum(PP(i+1:end));
end
line=0;
for i=length(T):-1:1
if line==0
if T(i,1)==T(i,2)
line=i;
elseif T(i,1)<=T(i,2)
line=i+1;
end
end
end
T
line
SS(1:line)=0;
SS(line+1:length(PP))=1;
elseif length(PP)==2
SS=[0 1];
else
SS=[0];
end
end
```

Output:

```
'ABCDEFG'

PK =    0.3000    0.2000    0.2000    0.1000    0.1000    0.0500    0.0500


SYMBOL =

  7×1 char array

    'A'
    'B'
    'C'
    'D'
    'E'
    'F'
    'G'

T =

    0.3000    0.7000
```

```
   0.5000   0.5000
   0.7000   0.3000
   0.8000   0.2000
   0.9000   0.1000
   0.9500   0.0500

line =

    2

ans =

   0.3000        0        0        0
   0.2000        0        0        0
   0.2000        0   1.0000        0
   0.1000        0   1.0000        0
   0.1000        0   1.0000        0
   0.0500        0   1.0000        0
   0.0500        0   1.0000        0


no =     3

ans =

   0.3000   1.0000        0        0   2.0000
   0.2000   1.0000        0   1.0000   2.0000
   0.2000        0   1.0000        0        0
   0.1000        0   1.0000        0        0
   0.1000        0   1.0000        0        0
   0.0500        0   1.0000        0        0
   0.0500        0   1.0000        0        0

T =
   0.2000   0.3000
   0.3000   0.2000
   0.4000   0.1000
   0.4500   0.0500

Line =     2
ans =

   0.3000   1.0000        0        0   2.0000
   0.2000   1.0000        0   1.0000   2.0000
   0.2000        0   1.0000        0        0
   0.1000        0   1.0000        0        0
   0.1000        0   1.0000   1.0000        0
   0.0500        0   1.0000   1.0000        0
   0.0500        0   1.0000   1.0000        0

no =
```

```
    4

ans =

   0.3000   1.0000        0        0        0   2.0000
   0.2000   1.0000        0   1.0000        0   2.0000
   0.2000   1.0000   1.0000        0        0   3.0000
   0.1000   1.0000   1.0000        0   1.0000   3.0000
   0.1000        0   1.0000   1.0000        0        0
   0.0500        0   1.0000   1.0000        0        0
   0.0500        0   1.0000   1.0000        0        0

T =

   0.1000   0.1000
   0.1500   0.0500

line =     1

ans =

   0.3000   1.0000        0        0        0   2.0000
   0.2000   1.0000        0   1.0000        0   2.0000
   0.2000   1.0000   1.0000        0        0   3.0000
   0.1000   1.0000   1.0000        0   1.0000   3.0000
   0.1000        0   1.0000   1.0000        0        0
   0.0500        0   1.0000   1.0000   1.0000        0
   0.0500        0   1.0000   1.0000   1.0000        0

no =

    5

ans =

   0.3000   1.0000        0        0        0        0   2.0000
   0.2000   1.0000        0   1.0000        0        0   2.0000
   0.2000   1.0000   1.0000        0        0        0   3.0000
   0.1000   1.0000   1.0000        0   1.0000        0   3.0000
   0.1000   1.0000   1.0000   1.0000        0        0   3.0000
   0.0500        0   1.0000   1.0000   1.0000        0        0
   0.0500        0   1.0000   1.0000   1.0000        0        0

ans =

   0.3000   1.0000        0        0        0        0   2.0000
   0.2000   1.0000        0   1.0000        0        0   2.0000
   0.2000   1.0000   1.0000        0        0        0   3.0000
   0.1000   1.0000   1.0000        0   1.0000        0   3.0000
   0.1000   1.0000   1.0000   1.0000        0        0   3.0000
```

```
    0.0500    1.0000    1.0000    1.0000    1.0000         0    4.0000
    0.0500    1.0000    1.0000    1.0000    1.0000    1.0000    4.0000

no =     6
ans =

    0.3000         0         0         0         0    2.0000
    0.2000         0    1.0000         0         0    2.0000
    0.2000    1.0000         0         0         0    3.0000
    0.1000    1.0000         0    1.0000         0    3.0000
    0.1000    1.0000    1.0000         0         0    3.0000
    0.0500    1.0000    1.0000    1.0000         0    4.0000
    0.0500    1.0000    1.0000    1.0000    1.0000    4.0000

Ave_Code_Word_Length =    2.6000

Entropy =    2.5464

Coding_Eff =    0.9794

Coding_Redun =    0.0206
```

Results:

Ex No. 13 **Implementation of Huffman Coding**

Aim:

- Write a program for generation and evaluation of variable length source coding using Huffman Coding and decoding.
- Calculate the entropy, average length and efficiency of Huffman Coding.

Description:

3 Explain variable length coding.

4 Explain Huffman coding techniques.

5 Solve theoretically and verify using matlab program the given example.

6 Explain the commands: Huffmandict, Huffmanenco, huffmandeco

Algorithm:

8 Start.
9 Input the total number of probabilities.
10 Arrange the messages in decreasing order of probabilities.
11 Add last two probabilities.
12 Assign them '0' and '1'.
13 With addition & other probabilities again sort out the total probabilities.
14 If the addition result is equal to probability of an symbol then put it on the top
15 Repeat the program from step 4 until addition is 1.
16 To find code for particular symbol take the path of probability of symbol and write codin reverse fashion.
17 Find out entropy, avg. code word length and efficiency.
18 Stop .

MatLab Code:

Program-1

```
%% Write a MATLAB based program for encoding and decoding of Huffman code
clc;
clear all;
close all;
symbol =[1:5]; % Distinct data symbols appearing in sig
p = [0.1 0.1 0.4 .3 .1]; % Probability of each data symbol
[dict,avglen]=huffmandict(symbol,p)
samplecode = dict{5,2} % Codeword for fifth signal value
dict{1,:}
dict{2,:}
dict{3,:}
dict{4,:}
dict{5,:}
hcode = huffmanenco(symbol,dict);
```

```
% Encode the data.
dhsig = huffmandeco(hcode,dict);
% Decode the code.
disp('encoded msg:');
disp(hcode);
disp('decoded msg:');
disp(dhsig);
code_length=length(hcode)
H=0;
for m=1:5
   H=H+(p(m)*log2(1/p(m)));
end
disp('H=');
disp(H);
Efficiency=(H/avglen)*100
```

Program-2

```
%% Huffman coding:STRING
% Write a MATLAB based program for encoding and decoding of
Huffman code
clc;
clear all;
close all;
msg='TEECT'
symbol ={'T' 'E' 'C'};
% Distinct data symbols appearing in sig
p = [2/5 2/5 1/5];
% Probability of each data symbol
[dict,avglen]=huffmandict(symbol,p)
dict{1,:}
dict{2,:}
dict{3,:}
hcode = huffmanenco(msg,dict);
% Encode the data.
dhsig = huffmandeco(hcode,dict);
% Decode the code.
disp('encoded msg:');
disp(hcode);
disp('decoded msg:');
disp(dhsig);
```

Output:

Output-1

```
dict =
    [1]    [1x4 double]
    [2]    [1x4 double]
    [3]    [       1]
```

```
    [4]    [1x2 double]
    [5]    [1x3 double]

avglen =
    2.1000
samplecode =     0     0     1

ans =     1
ans =     0     0     0     1
ans =     2
ans =     0     0     0     0
ans =     3
ans =     1
ans =     4
ans =     0     1
ans =     5
ans =     0     0     1
encoded msg:
    0     0     0     1     0     0     0     0     1     0     1     0     0     1
decoded msg:
    1     2     3     4     5
code_length =    14
H=    2.0464
Efficiency =   97.4495
```

Output-2

```
msg =
TEECT
dict =
    'T'    [1x2 double]
    'E'    [         1]
    'C'    [1x2 double]
avglen =    1.6000
T=     0     0
E =     1
C =     0     1
encoded msg:
    0     0     1     1     0     1     0     0
decoded msg:
    'T'    'E'    'E'    'C'    'T'
```

Results:

Ex No. 14 **Implementation of arithmetic coding algorithm**

Aim: To develop a matlab program to compute arithmetic coding.

Description: In arithmetic coding, a message is encoded as a real number in an interval from one to zero. Arithmetic coding typically has a better compression ratio than Huffman coding, as it produces a single symbol rather than several separate code words. Arithmetic coding is a lossless coding technique.
The arithmetic coding algorithm, with an example is given below:

- Start with an interval [0, 1], divided into subintervals of all possible symbols to appear within amessage. Make the size of each subinterval proportional to the frequency at which it appears inthe message.

  Example:

Symbol	Probability	Interval
A	0.2	[0.0, 0.2)
B	0.3	[0.2, 0.5)
C	0.1	[0.5, 0.6)
D	0.4	[0.6, 1.0)

- When encoding a symbol, "zoom" into the current interval, and divide it into subintervals like in step one with the new range. Example: suppose we want to encode "abd". We "zoom" into the interval corresponding to "a", and divide up that interval into smaller subintervals like before. Wenow use this new interval as the basis of the next symbol encoding step.

Symbol	New "a" Interval
A	[0.0, 0.04)
B	[0.04, 0.1)
C	[0.1, 0.102)
D	[0.102, 0.2)

- Repeat the process until the maximum precision of the machine is reached, or all symbols are encoded. To encode the next character "b", we zuse the "a" interval created before, and zoom intothe subinterval "b", and use that for the next step. This produces:

Symbol	New "b" Interval
A	[0.102, 0.1216)
B	[0.1216, 0.151)
C	[0.151, 0.1608)
D	[0.1608, 0.2)

- And lastly, the final result is:

Symbol	New "d" Interval

A	[0.1608, 0.16864)
B	[0.16864, 0.1804)
C	[0.1804, 0.18432)
D	[0.18432, 0.2)

- Transmit some number within the latest interval to send the codeword. The number of symbolsencoded will be stated in the protocol of the image format, so any number within [0.1608, 0.2) will be acceptable.
- To decode the message, a similar algorithm is followed, except that the final number is given,and the symbols are decoded sequentially from that.

MatLab Code:

```
% arcodemo.m: demonstration of arithmetic coding
% call arenc.m, ardec.m
function Arithmetic_Coding
clc
clear all
format long;
symbol=['adin'];
%the list of symbols, a row vector of single letters
pr=[1 1 2 1]/5;
%the corresponding probabilty of each symbol
seqin=['india'];
%the input sequence of symbols to be encoded.
codeword=arenc(symbol,pr,seqin) % encoding
seqout=ardec(symbol,pr,codeword,5) % decoding_


function arcode=arenc(symbol,pr,seqin)
high_range=[];
for k=1:length(pr),
   high_range=[high_range sum(pr(1:k))];
end
low_range=[0 high_range(1:length(pr)-1)];
sbidx=zeros(size(seqin));
for i=1:length(seqin),
   sbidx(i)=find(symbol==seqin(i));
end
low=0;
high=1;
for i=1:length(seqin),
   range=high-low;
   high = low + range*high_range(sbidx(i));
   low = low + range*low_range(sbidx(i));
end
```

```
arcode=low;

function symseq=ardec(symbol,pr,codeword,symlen)
format long
high_range=[];
for k=1:length(pr),
   high_range=[high_range sum(pr(1:k))];
end
low_range=[0 high_range(1:length(pr)-1)];
prmin=min(pr);
symseq=[];
for i=1:symlen,
   idx=max(find(low_range<=codeword));
   codeword=codeword-low_range(idx);
   % due to numerical error, sometimes the
   % encoded number will be slightly smaller
   % than the current lower bound. If this
   % happens, a correction is required.
if abs(codeword-pr(idx))< 0.01*prmin,
   idx=idx+1;
   codeword=0;
end
symseq=[symseq symbol(idx)],
codeword=codeword/pr(idx),
if abs(codeword)<0.01*prmin,
   i=symlen+1; % break the for loop immediately end
end
end
```

Output:

```
codeword =    0.742400000000000
symseq = i
codeword =    0.856000000000000
symseq = in
codeword =   0.280000000000000
symseq = ind
codeword =    0.400000000000001
symseq =indi
codeword =     2.914335439641036e-15
symseq =india
codeword =     1.457167719820518e-14
seqout = india
```

Results:

Ex No. 15 **Error Detection using Parity Bit**

Aim: To develop a matlab program to detect single bit transmission error using even and odd parity methods.

Description: A parity bit is a bit appended to a data of binary bits to ensure that the total number of 1' in the data are even or odd. Parity bits are used for error detection.

Parity generation

There are two types of parity bits:

Even parity bit:

In the case of even parity for a given set of bits, the number of 1' are counted. If that count is odd, the parity bit value is set to 1, making the total count of occurrences of 1's an even number. If the total number of 1' in a given set of bits is already even, the parity bits values is 0.

Odd parity bit:

In the case of even parity for a given set of bits, the number of 1' are counted. If that count is even, the parity bit value is set to 1, making the total count of occurrences of 1's an odd number. If the total number of 1' in a given set of bits is already even, the parity bits values is 0.

Parity checking

The received message is checked for the number of ones in the message. In even parity if the number of ones is even then it is concluded that there is no transmission error. The number ones will be odd for single bit of error. the opposite is true for odd parity

MatLab Code:

```
clc; clear all; close all;
%----------------parity generation----------------%
x=input('enter the bit sequence of the message');
t=0;
t=sum(x);
choice=input('Enter 1 for odd parity and 2 for even
parity');
if(choice==1)
    if(mod(t,2)~=0)
        y=[x,1];
    else
        y=[x,0];
    end;
    disp('input bit sequence');
    disp(x);
    disp('bit sequence with odd parity bit');
    disp(y);
else
    if(mod(t,2)~=1)
        y=[x,1];
    else
        y=[x,0];
    end;
    disp('inputbit sequence');
```

```
    disp(x)
    disp('bit sequence with even parity bit');
    disp(y)
end;

%-----------------------------------------------------
---%
recd=input('enter the received message');
r=0;
r=sum(recd);
if(choice==1)
    if(mod(r,2)==0)
        disp('There is a bit of error');
    else
        disp('There is no error');
    end;
else
    if(mod(r,2)~=0)
        disp('There is bit of error');
    else
        disp('There is no error');
    end;
end
```

Output:

```
enter the bit sequence of the message101101
Enter 1 for odd parity and 2 for even parity1
Input bit sequence
    101101

bit sequence with odd parity bit
    101101        1

enter the received message1011011
There is no error
 ------------------------------------------------------------
enter the bit sequence of the message101101
Enter 1 for odd parity and 2 for even parity2
Input bit sequence
    101101

bit sequence with even parity bit
    101101        0

enter the received message1011010
There is no error
------------------------------------------------------------
enter the bit sequence of the message101101
Enter 1 for odd parity and 2 for even parity 1
input bit sequence
    101101
```

```
bit sequence with odd parity bit
     101101          1

enter the received message1011010
There is a bit of error
------------------------------------------------------------
enter the bit sequence of the message101101
Enter 1 for odd parity and 2 for even parity 2
input bit sequence
     101101

bit sequence with even parity bit
     101101          0

enter the received message1011011
There is a bit of error
```

Results:

Ex No. 16 **Error Detection using Cyclic Redundancy Check (CRC)**

Aim: To develop a matlab program to detect transmission error using Cyclic Redundancy Check (CRC) method.

Description: Cyclic Redundancy Cheek (CRC) is a method of detecting accidental changes/errors in communication channel. CRC uses Generator Polynomial which is available on both sender and receiver side. An example generator polynomial is of the form like x + x + 1. This generator polynomial represents key 1011. Another example is $x^2 + 1$ that represents key 101.

Let n: Number of bits in data to be sent from sender side.

k: Number of bits in the key obtained from generator polynomial.

Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

1. The binary data is first augmented by adding k-1 zeros in the end of the data

2. Use modulo-2 binary division to divide binary data by the key and store remainder of division.

3. Append the remainder at the end of the data to form the encoded data and send the same.

Receiver Side (Check if there are errors introduced in transmission)

Perform modulo-2 division again and if remainder is 0, then there are no errors.

Modulo 2 Division:

The process of modulo-2 binary division is the same as the familiar division process used for decimal numbers. Just that instead of subtraction, XOR is used.

In each step, a copy of the divisor (or data) is XORed with the k bits of the dividend (or key).

The result of the XOR operation (remainder) is (n-1) bits, which is used for the next step after I extra bit is pulled down to make it n bits long

When there are no bits left to pull down, we have a result. The (n-1)-bit remainder which is appended at the sender side.

Illustration for No Error in transmission:

Data word to be sent - 100100

Key-1101 [Or generator polynomial x3 + x2+1]

No. of bits in the data (n) = 6 and
No. of bits in the key(k)=4
Append k-1 zeros at the end of the data:100100000

Sender side:

```
          111101
     ┌──────────
1101 │ 100100000
     │ 1101
     │  1000
     │  1101
     │   1010
     │   1101
     │    1110
     │    1101
     │     0110
     │     0000
     │      1100
     │      1101
     │       001
```

Therefore, the remainder is 001 and hence the encoded data sent is 100100001.

Receiver side:

Code word received at the receiver side 100100001

```
          111101
     ┌──────────
1101 │ 100100000
     │ 1101
     │  1000
     │  1101
     │   1010
     │   1101
     │    1110
     │    1101
     │     0110
     │     0000
     │      1101
     │      1101
     │       000
```

Therefore, the remainder is all zero hence the data received has no error.

Illustration for No Error in transmission:

Code word received at the receiver side 100000001

```
               111010
        1101 | 100000000 |
               1101
                1010
                1101
                 1110
                 1101
                  0110
                  0000
                   1100
                   1101
                    0011
                    0000
                     011
```

MatLab Code:

```
msg = input("Input Message sequence  ");
poly = input("Input Generator Polynomial  ");

[M, N] = size(poly);
mseg = [msg zeros(1, N - 1)];
[~, r] = deconv(mseg, poly);

r = mod(abs(r), 2);

crc = r(length(msg) + 1:end);
frame = bitor(mseg, r);

disp("Transmitted Frame:");
disp(frame);

recd = input("Enter the received message : ");
[q, r] = deconv(recd, poly);
r = mod(abs(r), 2);

if all(r == 0)
    disp("There is no error");
else
    disp("There is an error");

    % Correct the received frame
    corrected_frame = recd;
    corrected_frame(length(msg) + 1:end) =
mod(corrected_frame(length(msg) + 1:end) + crc, 2);
    disp("Corrected Frame:");
```

```
    disp(corrected_frame);

    % Extract the received message after error
correction
    received_msg = corrected_frame(1:length(msg));
    disp("Received Message after Error Correction:");
    disp(received_msg);
end
```

Output:

Input Message sequence [1 1 1 0 0 0 1 1 1]
Input Generator Polynomial [1 1 0 0 1 1]
Transmitted Frame:
1 1 1 0 0 0 1 1 1 1 0 1 0 0

Enter the received message [11100011110100]
There is no error

Input Message sequence [1 1 1 0 0 0 1 1 1]
Input Generator Polynomial [1 1 0 0 1 1]
Transmitted Frame:
1 1 1 0 0 0 1 1 1 1 0 1 0 0

Enter the received message [11100011110101]
There is an error

Results: Thus a matlab program has been written to detect the transmission error using cyclic redundancy code.

Ex No. 17 **Error Correction using Hamming Code**

Aim: To develop a MATLAB program to detect and correct the transmission error using Hamming (7,4) code

Description: In general a block code with k information digits and block length n is called an (n,k) code. Thus, an (7,4) code has block length 7, and 4 message bits. Consider the following message bits and code word:
Message digits: C1 C2 C3 C4
Code word: C1 C2 C3 C4 C5 C6 C7
[10:50 am, 09/01/2024] Sathiyapoobalan: HAMMING (74) ENCODING

Using the code word bits Parity Check Equations can be formed as follows:

C1 C2 C3 C5-0

C1 C2 C4 C6=0

C1 C3 C4 C7=0

Correspondingly Parity Check Matrix can be formed as follows:

1110100 1101010 1011001

Message: (C1 C2 C3 C4)=(1111)

from C1 C2 C3 C4, compute C5 C6 C7 in such a way to satisfy the parity equations.

Resultant code word: 1 1
[10:50 am, 09/01/2024] Sathiyapoobalan: HAMMING (7.4) CODE: SYNDROME DECODING

Let R1 R2 R3 R4 R5 R6 R7 be the received block of binary digits, possibly with errors,

R1 ⊕ R2 ⊕ R3 ⊕ R5=S1

R1 ⊕ R2 ⊕ R4 ⊕ R6=S2

R1 ⊕ R3 ⊕ R4 ⊕ R7=83

The combination of S1, S2 and S3 is called the syndrome

Transmitted code word: 1 111111

Let the received block with one error in a message bit is 1 011111.

From the received block compute syndrome bits. In this case S1=1, S2=1,and S3= 0

Parity Check Matrix:

S1	1	**1**	1	0	1	0	0
S2	1	**1**	0	1	0	1	0
S3	1	**0**	1	1	0	0	1

The syndrome S1 S2 S3 is 1 1 0 it corresponds to second column in parity check matrix. Hence R2 is the error bit and invert it to get the corrected code.

MatLab Code:

```
clc
clear all
close all

n = 7
k = 4
A = [1 1 1; 1 1 0; 1 0 1; 0 1 1]
G = [eye(k) A]
H = [A' eye(n - k)]
msg = [1 1 1 1] % Use a row vector for the
message
code = mod(msg * G, 2)

% Introduce errors in the code
code(2) = ~code(2);
code(5) = ~code(5);
%code(1)=~code(1);
%code(3)=~code(3);
%code(4)~code(4);
%code(6)=~code(6);
%code(7)=~code(7);

recd = code
syndrome = mod(recd * H', 2)

% Find position of the error in codeword
(index)
find = 0;
index = 0;
```

```
for ii = 1:n
    if ~find
        errvect = zeros(1, n);
        errvect(ii) = 1;
        search = mod(errvect * H', 2);

        if isequal(search, syndrome)  % Use
isequal instead of ==
            find = 1;
            index = ii;
        end
    end
end

if index == 0
    disp('There is no error');
else
    disp('There is an error');
    disp(['Position of error in codeword=',
num2str(index)]);
    correctedcode = recd
    correctedcode(index) = mod(recd(index) +
1, 2)

    msg_decoded = correctedcode(1:k)  %
Corrected decoding
end
```

Output:

```
n =

   7


k =

   4


A =

   1   1   1
   1   1   0
   1   0   1
   0   1   1


G =
```

```
    1    0    0    0    1    1    1
    0    1    0    0    1    1    0
    0    0    1    0    1    0    1
    0    0    0    1    0    1    1
```

H =

```
    1    1    1    0    1    0    0
    1    1    0    1    0    1    0
    1    0    1    1    0    0    1
```

msg =

```
    1    1    1    1
```

code =

```
    1    1    1    1    1    1    1
```

recd =

```
    1    0    1    1    0    1    1
```

syndrome =

```
    0    1    0
```

There is an error
Position of error in codeword=6

correctedcode =

```
    1    0    1    1    0    1    1
```

correctedcode =

```
    1    0    1    1    0    0    1
```

msg_decoded =

```
    1    0    1    1
```

Results: Thus a matlab progam has been written to detect and correct the transmission error using Hamming (7,4) code.

Ex No. 18 **Implement Image Compression using Wavelets Transform**

Aim: To implement image compression using wavelets and study on colour and gray images.

Description: What is wavelet decomposition?
What is image compression?
What is image decompression?

Algorithm:

- Read the image
- Obtain the Decomposition Level.
- Convert the image into gray scale, if the input image is colour image
- Decompose the image using wavelets
- Compress the image using wavelets
- Recover the image by decompressing
- Compare the original and recovered image
- Print original, compressed and recovered images.

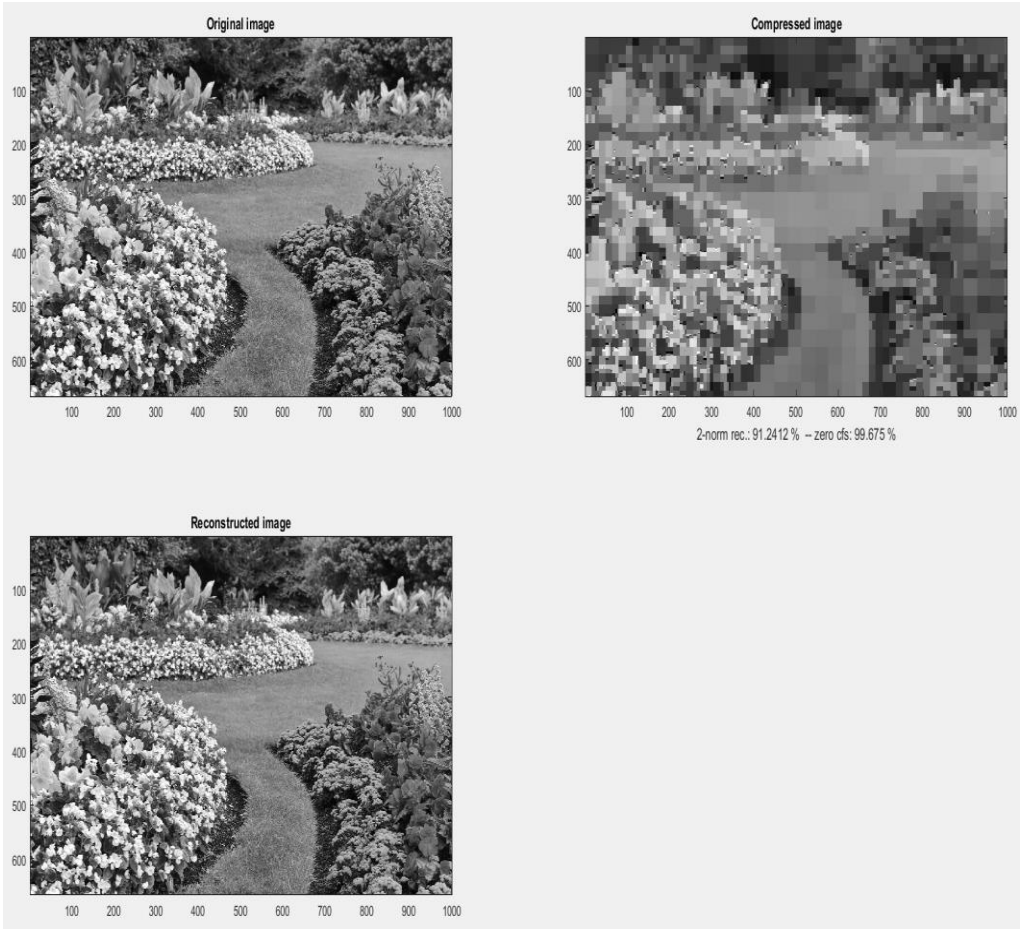
MatLab Code:

```
% Image compression by Wavelet Transfrom
clear all;
close all;
clc;
% reading an Image
X=imread('garden.jpg');  % Colour Input Image
% X=imread('cameraman.tif'); % Gray Input Image
[~,~,nc]=size(X)
% inputting the decomposition level and name of the wavelet
n=input('Enter the decomposition level:');
wname = 'haar';
x = double(X);
TotalColors = 255;
map = gray(TotalColors);
x = uint8(x);
%Conversion of RGB to Grayscale
if nc>1
   x=rgb2gray(X);
else
   x=X;
end
% A wavelet decomposition of the image
[c,s] = wavedec2(x,n,wname);
% wdcbm2 for selecting level dependent thresholds
alpha = 1.5; m = 2.7*prod(s(1,:));
[thr,nkeep] = wdcbm2(c,s,alpha,m)
% Compression
[xd,cxd,sxd,perf0,perfl2] = wdencmp('lvd',c,s,wname,n,thr,'h');
disp('Compression Ratio');
disp(perf0);
```

```
% Decompression
R = waverec2(c,s,wname);
 rc = uint8(R);
% Plot original and compressed images.
subplot(221), image(x);
colormap(map);
title('Original image')
subplot(222), image(xd);
colormap(map);
title('Compressed image')
% Displaying the results
xlab1 = ['2-norm rec.: ',num2str(perfl2)];
xlab2 = [' %  -- zero cfs: ',num2str(perf0), ' %'];
xlabel([xlab1 xlab2]);
subplot(223), image(rc);
colormap(map);
title('Reconstructed image');
```

Output:

```
nc =      3

Enter the decomposition level: 5

thr =
  141.0000  255.2500  389.0000  486.8125  323.1875
  141.0000  255.2500  389.0000  486.8125  323.1875
  141.0000  255.2500  389.0000  486.8125  323.1875

nkeep =
   123   162   227   349   641
Compression Ratio =   99.6750
```


Original image
Compressed image
2-norm rec.: 91.2412 % -- zero cfs: 99.675 %
Reconstructed image

Results: