BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT
AVALAHALLI, DODDABALLAPUR MAIN ROAD, BANGALORE-64

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DIGITAL SIGNAL PROCESSING LABORATORY: 18ECL57

DEPARTMENT VISION

Be a pioneer in Providing Quality Education in Electronics, Communication and Allied Engineering fields to serve as Valuable Resource for Industry and Society.

DEPARTMENT MISSION

1. Impart Sound Theoretical Concepts and Practical Skills through innovative Pedagogy.
2. Promote Inter-disciplinary Research.
3. Inculcate Professional Ethics.

Course co-ordinators (2019-20)

1. Saneesh Cleatus T, Associate Professor, ECE
2. Dr. Shobharani A , Associate Professor, ECE
3. Dr. Deepa Reddy, Assistant Professor, ECE

Laboratory Instructor

1. Mr. Byrareddy A B
2. Mrs. Supriya K

DSP LAB

B.E., V Semester, ELECTRONICS & COMMUNICATION ENGINEERING / TELECOMMUNICATION ENGINEERING

Course Code	**18ECL57**	**CIE Marks**	**40**
		SEE Marks	**60**
Hours/Week	**3**		
RBT Levels	**L1, L2, L3**	**Exam Hours**	**03**

CREDITS – 02

Course Objectives: This course will enable students to

- Simulate discrete time signals and verification of sampling theorem.
- Compute the DFT for a discrete signal and verification of its properties using MATLAB.
- Find solution to the difference equations and computation of convolution and correlation along with the verification of properties.
- Compute and display the filtering operations and compare with the theoretical values.
  Implement the DSP computations on DSP hardware and verify the result.
- **Laboratory Experiments**

Following Experiments to be done using MATLAB / SCILAB / OCTAVE or equivalent:

Following Experiments to be done using MA TLAB / SCILAB / OCTAVE or equivalent:

1. Verification of sampling theorem (use interpolation function).
2. Linear and circular convolution of two given sequences, Commutative, distributive and associative property of convolution.
3. Auto and cross correlation of two sequences and verification of their properties
4. Solving a given difference equation.
5. Computation of N point DFT of a given sequence and to plot magnitude and phase spectrum (using DFT equation and verify it by built-in routine).
6. (i) Verification of DFT properties (like Linearity and Parseval's theorem, etc.)
   (ii) DFT computation of square pulse and Sinc function etc.
7. Design and implementation of Low pass and High pass FIR filter to meet the desired specifications (using different window techniques) and test the filter with an audio file. Plot the spectrum of audio signal before and after filtering.

8. Design and implementation of a digital IIR filter (Low pass and High pass) to meet given specifications and test with an audio file. Plot the spectrum of audio signal before and after filtering.

Following Experiments to be done using DSP kit

9. Obtain the Linear convolution of two sequences.
10. Compute Circular convolution of two sequences.
11. Compute the N-point DFT of a given sequence.
12. Determine the Impulse response of first order and second order system.
13. Generation of Sine wave and standard test signals

Course Outcomes: Students will be able to

CO1: Apply mathematical operations to perform DSP algorithm..
CO2: Analyze various DSP operations Transformations and LIT.
CO3: Write C programs of various signal processing operations and transformations and download the executable code on to TMS320C6713 DSP Processor using CODE COMPOSER STUDIO.
CO4: Comprehend, write and reproduce the programs and results.
CO5: Develop simple DSP based systems required for communication and Signal processing applications.

DIGITAL SIGNAL PROCESSING LABORATORY

List of Experiments

PART –A

1. Verification of sampling theorem (use interpolation function).
2. Linear and circular convolution of two given sequences, Commutative, distributive and associative property of convolution.
3. Auto and cross correlation of two sequences and verification of their properties
4. Solving a given difference equation.
5. Computation of N point DFT of a given sequence and to plot magnitude and phase spectrum (using DFT equation and verify it by built-in routine).
6. (i) Verification of DFT properties (like Linearity and Parseval's theorem, etc.)
   (ii) DFT computation of square pulse and Sinc function etc.
7. Design and implementation of Low pass and High pass FIR filter to meet the desired specifications (using different window techniques) and test the filter with an audio file. Plot the spectrum of audio signal before and after filtering.

PART-B

1. Obtain the Linear convolution of two sequences.
2. Compute Circular convolution of two sequences.
3. Compute the N-point DFT of a given sequence.
4. Determine the Impulse response of first order and second order system.
5. Generation of Sine wave and standard test signals

PART-C

Open end experiment : Creation of GUI based signal generator with time domain and frequency domain plots.

MATLAB

MATLAB (matrix laboratory) is a programming language developed by MathWorks. It started out as a matrix programming language where linear algebra programming was simple. It can be run both under interactive sessions and as a batch job. MATLAB is a paradigm numerical environment and proprietary programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran and Python.

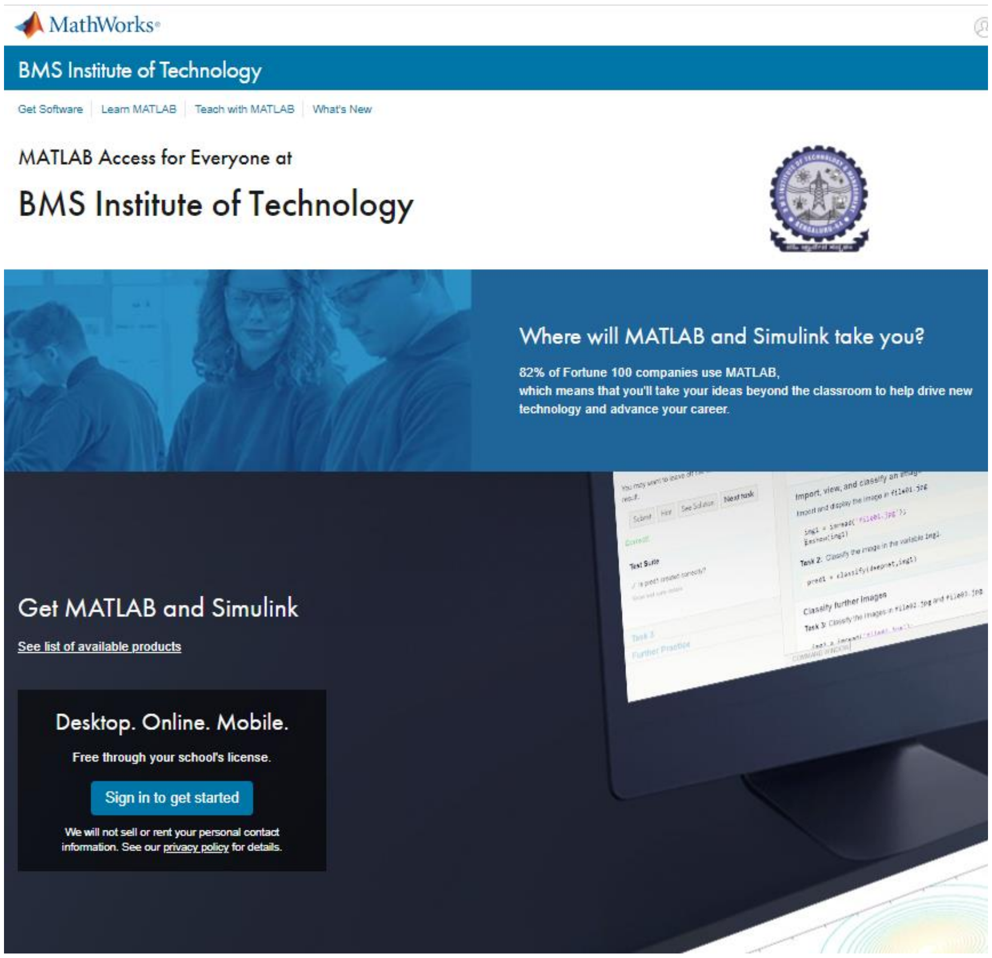
Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine, allowing access to symbolic computing abilities. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems.

As of 2017, MATLAB has roughly one million users across industry and academia. MATLAB users come from various backgrounds of engineering, science, and economics. There are various toolboxes in MATLAB like signal processing toolbox, image processing tool box, neural network toolbox, statistics toolbox, financial toolbox, Bioinformatics toolboxetc.

The Matlab screen has 4 important windows along with Editor (where programs can be written and stored). These are Command window where we type commands and outputs/error messages are seen, Command history window where past commands are remembered(we can re-run any command by clicking on the commands),Workspace window which stores all the variables used and Current Directory window shows the files in current directory.

In this semester students are required to register in Mathworks website using their official email ID and download the MATLAB software from the following link.

https://www.mathworks.com/academia/tah-portal/bms-institute-of-technology-31463638.html


MathWorks
BMS Institute of Technology
Get Software | Learn MATLAB | Teach with MATLAB | What's New

MATLAB Access for Everyone at

BMS Institute of Technology

Where will MATLAB and Simulink take you?

82% of Fortune 100 companies use MATLAB, which means that you'll take your ideas beyond the classroom to help drive new technology and advance your career.

Get MATLAB and Simulink

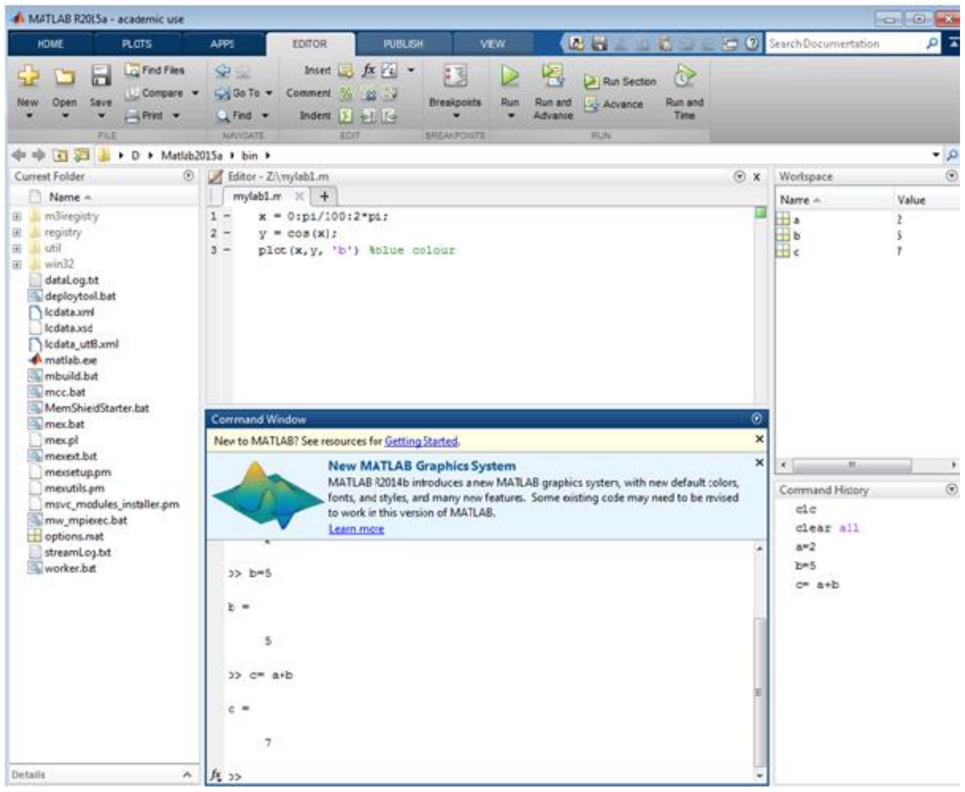
See list of available products

Desktop. Online. Mobile.

Free through your school's license.

We will not sell or rent your personal contact information. See our privacy policy for details.

The installation might take minimum 15-20 GB of your internet data as well as same amount of system memory.



```
x = 0:pi/100:2*pi;
y = cos(x);
plot(x,y, 'b') %blue colour
```

New to MATLAB? See resources for Getting Started.

New MATLAB Graphics System

MATLAB R2014b introduces a new MATLAB graphics system, with new default colors, fonts, and styles, and many new features. Some existing code may need to be revised to work in this version of MATLAB.

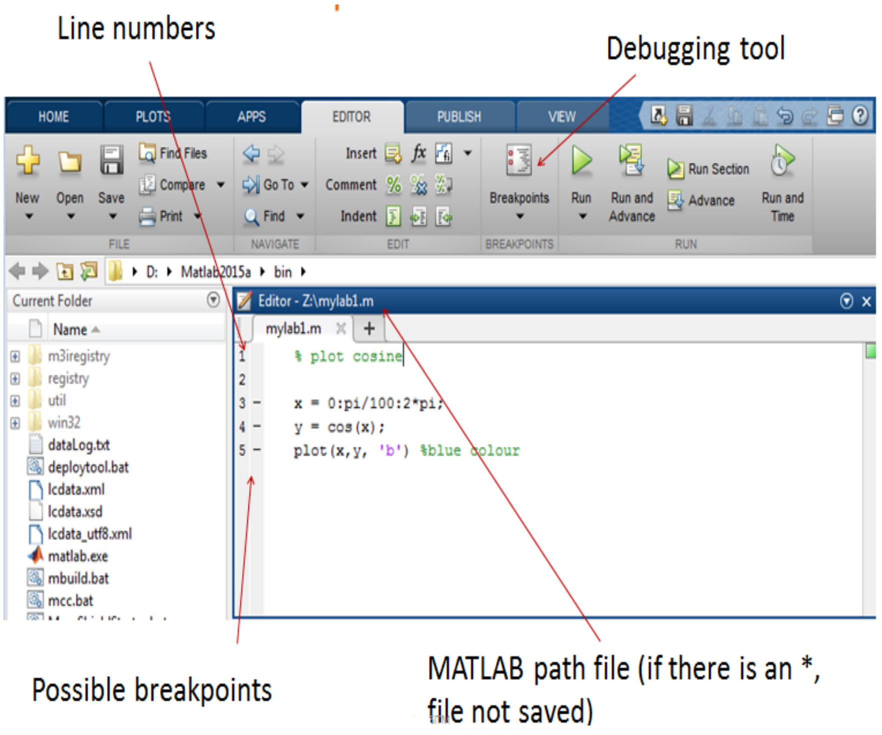
Learn more

```
>> b=5

b =

     5

>> c= a+b

c =

     7

>>
```

Name	Value
a	2
b	5
c	7

```
clc
clear all
a=2
b=5
c= a+b
```

FIG 1 MATLAB WINDOWS



```
% plot cosine

x = 0:pi/100:2*pi;
y = cos(x);
plot(x,y, 'b') %blue colour
```

FIG 2 SCREEN COMPOMNTS

PART A

EXPERIMENTS USING MATLAB/OCTAVE

EXPERIMENT NO 1

Verification of Sampling Theorem

Aim: To verify Sampling theorem for a signal of given frequency.

Theory:

- Sampling is a process of converting a continuous time signal (analog signal) x(t) into a discrete time signal x[n], which is represented as a sequence of numbers. (A/D converter)
- Converting back x[n] into analog (resulting in $x(t)$) is the process of reconstruction. (D/A converter)
- Techniques for reconstruction-(i) ZOH (zero order hold) interpolation results in a staircase waveform, is implemented by MATLAB plotting function ***stairs(n,x),*** (ii) FOH (first order hold) where the adjacent samples are joined by straight lines is implemented by MATLAB plotting function ***plot(n,x),***(iii) spline interpolation, etc.
- For $x(t)$ to be exactly the same as x(t), sampling theorem in the generation of x(n) from x(t) is used. The sampling frequency fs determines the spacing between samples.
- Aliasing-A high frequency signal is converted to a lower frequency, results due to under sampling. Though it is undesirable in ADCs, it finds practical applications in stroboscope and sampling oscilloscopes.

Algorithm:

1. Input the desired frequency f_{max} (for which sampling theorem is to be verified).
2. Generate an analog signal x_t of frequency f_{max} for comparison.
3. Generate over sampled, Nyquist & Under sampled discrete time signals.
4. Plot the waveforms and hence prove sampling theorem.

MATLAB Implementation:

Step 1: MATLAB can generate only discrete time signals. For an approximate analog signal xt, choose the spacing between the samples to be very small (≈0), say 50µs = 0.00005. Next choose the time duration, say xt exists for 0.05seconds. (final in program) (for low frequency say <1000 Hz choose 0.05 secs, for higher choose 0.01 secs or lesser as appropriate).
Now begin with the vector that represents the time base-

t = 0:0.00005:0.05;

The colon (:) operator in MATLAB creates a vector, in the above case a time vector running from 0 to 0.05 in steps of 0.00005. The semicolon (;) tells MATLAB not to display the result.

Given t, the analog signal xt of frequency fd is generated as (sin(ωt)=sin(2πft))-

xt=sin(2*pi*fmax*t);

pi is recognized as 3.14 by MATLAB.

Step 2: To illustrate oversampling condition, choose sampling frequency fs0=2.2*fd. For this sampling rate T0=1/fs0, generate the time vector as **n1 = 0:T0:0.05; &** over sampled discrete time signal **x1=sin(2*pi*f$_{max}$*n1);** [Alternately let n1 be a fixed number of samples, say n=0:10; & x1=sin(2*pi*n*fd/fs0);]

The discrete signal is converted back to analog signal (reconstructed) using the MATLAB plot function (FOH). In one plot both the analog signal (approximate one in blue color) and the reconstructed signal (in red) are plotted for comparison

Step 3: Repeat step 2 for different sampling frequencies, i.e., fs=1.3*fd & fs=2*fd for under sampling and Nyquist sampling conditions.

<u>MATLAB Program:</u>

```
tfinal=0.05;
t=0:0.00005: tfinal;
fmax=input('Enter analog frequency');

%define analog signal for comparison
xt=sin(2*pi*fmax*t);

%simulate condition for under sampling i.e., fs1<2*fmax
fs1=1.3*fmax;
%define the time vector
n1=0:1/fs1:tfinal;
%Generate the under sampled signal
```

xn1=sin(2*pi*n1*f_{max}[1]);

%plot the analog & sampled signals

subplot(3,1,1);

plot(t,xt,'b',n1,xn1,'r*-');

title('Under sampling plot');

%Condition for Nyquist plot

fs2=2*f_{max};

n2=0:1/fs2:tfinal;

xn2=sin(2*pi*f_{max}*n2);

subplot(3,1,2);

plot(t,xt,'b',n2,xn2,'r*-');

title('Nyquist plot');

%Condition for Oversampling

fs3=10*f_{max};

n3=0:1/fs3:tfinal;

xn3=sin(2*pi*f_{max}*n3);

subplot(3,1,3);

plot(t,xt,'b',n3,xn3,'r*-');

title('Oversampling plot');

xlabel('time');

ylabel('amplitude');

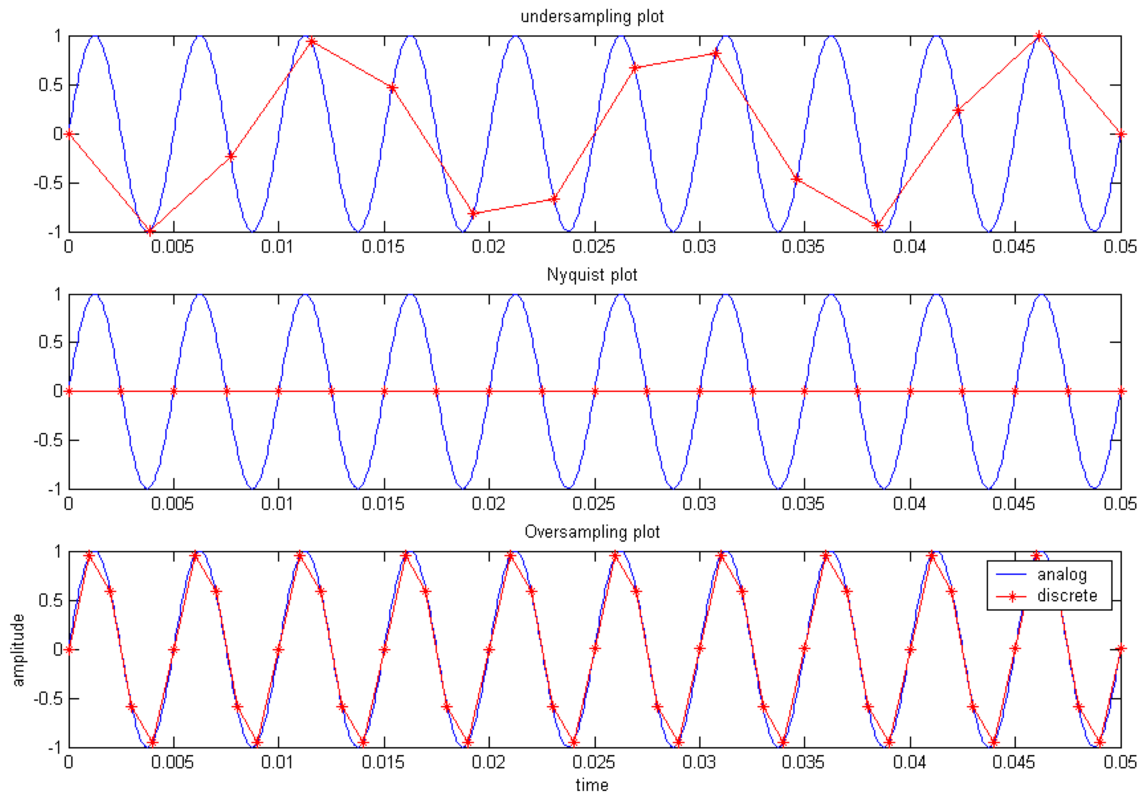
legend('analog','discrete')

Result:

Enter analog frequency 200

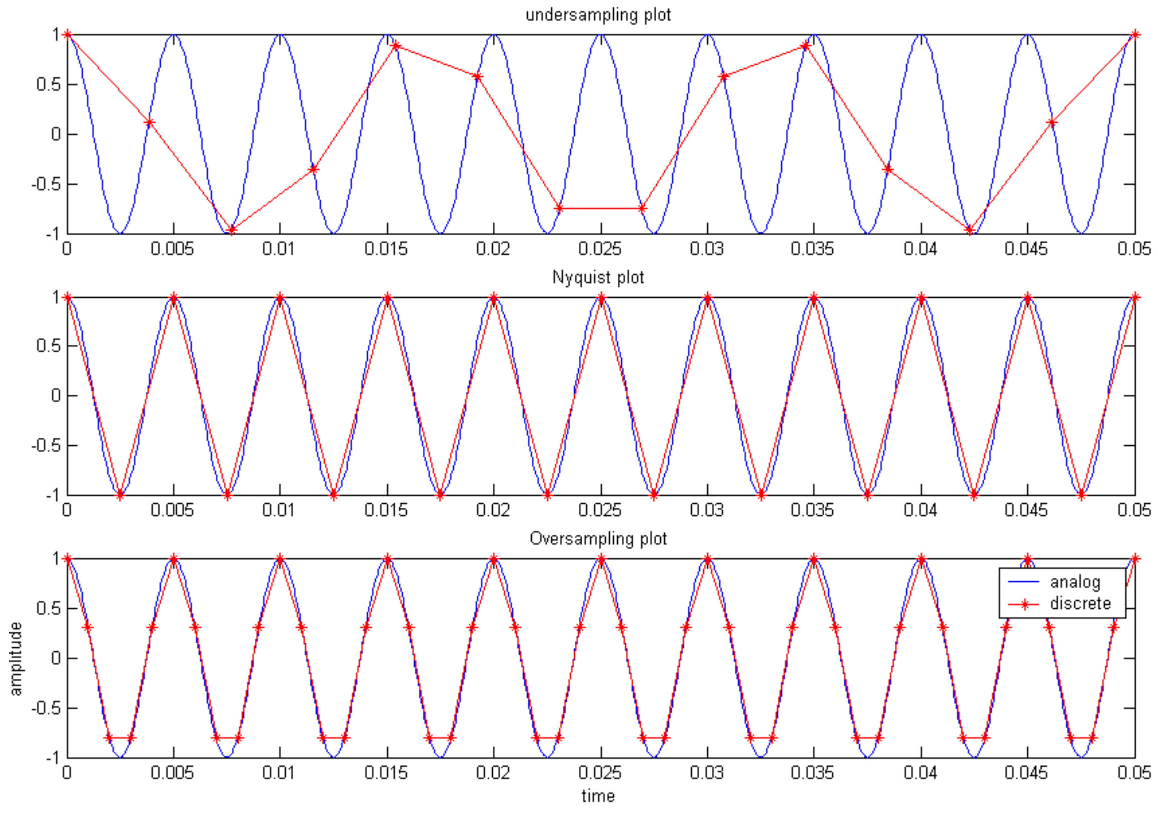
Output Waveforms

Plots of a sampled sine wave of 200Hz



Students are expected to use interp1q() function to connect the samples in the sampled signal.

Plots of a sampled cosine wave of 200Hz



Inference:

1. From the under sampling plot observe the aliasing effect. The analog signal is of 200Hz (T=0.005s). The reconstructed (from under sampled plot) is of a lower frequency. The alias frequency is computed as

$$f_{max}-f_s1 = 200-1.3*200 = 200-260= -60\text{Hz}$$

This is verified from the plot. The minus sign results in a 180˚ phase shift.

(For example: 3kHz & 6kHz sampled at 5kHz result in aliases of -2kHz (3k-5k) & 1kHz (6k-5k) respectively)

2. Sampling at the Nyquist rate results in samples sin(πn) which are identically zero, i.e., we are sampling at the zero crossing points and

hence the signal component is completely missed. This can be avoided by adding a small phase shift to the sinusoid. The above problem is not seen in cosine waveforms (except cos(90n)). A simple remedy is to sample the analog signal at a rate higher than the Nyquist rate. The **Fig1.2** shows the result due to a cosine signal (**x1=cos(2*pi*f_{max}*n1);**

3. The over sampled plot shows a reconstructed signal almost similar to that of the analog signal. Using low pass filtering the wave form can be further smoothened.

Open ended Experiments:

Generate sine/cosine waveforms of multiple frequencies, say x1=sin(2*pi*fd1*n1)+ sin(2*pi*fd2*n2); where fd1, fd2 are 2 different frequencies

VIVA Questions

1. Define plot, stem, and subplot in MATLAB.
2. List all the functions used in the above program.
3. Briefly explain sampling theorem.
4. What is aliasing?

EXPERIMENT NO 2

A. Linear Convolution of Two Given Sequences

Aim: To obtain convolution of two finite duration sequences.

Theory:

- The output y[n] of a LTI (linear time invariant) system can be obtained by convolving the input x[n] with the system's impulse response h[n].
- The convolution sum is

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

- x[n] and h[n] can be both finite or infinite duration sequences.
- Even if one (or both) of the sequences is infinite (say, $h[n] = (0.9)^n u[n]$), we can analytically evaluate the convolution formula to get a functional form (closed form solution) of y[n].
- If both the sequences are of finite duration, then we can use the MATLAB function '***conv***' to evaluate the convolution sum to obtain the output y[n]. Convolution is implemented as polynomial multiplication (refer MATLAB help).
- The length of y[n] = xlength + hlength -1.
- The conv function assumes that the two sequences begin at n=0 and is invoked by y=conv(x,h).
- Even if one of the sequences begin at other values of n, say n=-3,or n=2; then we need to provide a beginning and end point to y[n] which are ybegin=xbegin+hbegin and yend=xend+hend respectively.

Algorithm:

1. Input the two sequences as x1, x2
2. Convolve both to get output y.
3. **Plot the sequences.**

<u>MATLAB Implementation:</u>

MATLAB recognizes index 1 to positive maximum. Index 0 is also not recognized. The timing information for a sequence is provided by another vector, say **n=-3:5;** creates a vector with values from -3 to 5 with an increment of 1.

During plotting, the time vector size and the sequence size should be the same, i.e., the number of elements in the sequence x1 and in its time vector n1 should be the same. Similarly for x2 and y.

<u>MATLAB Program:</u>

<u>Level 1 Program (both sequences are right sided)</u>

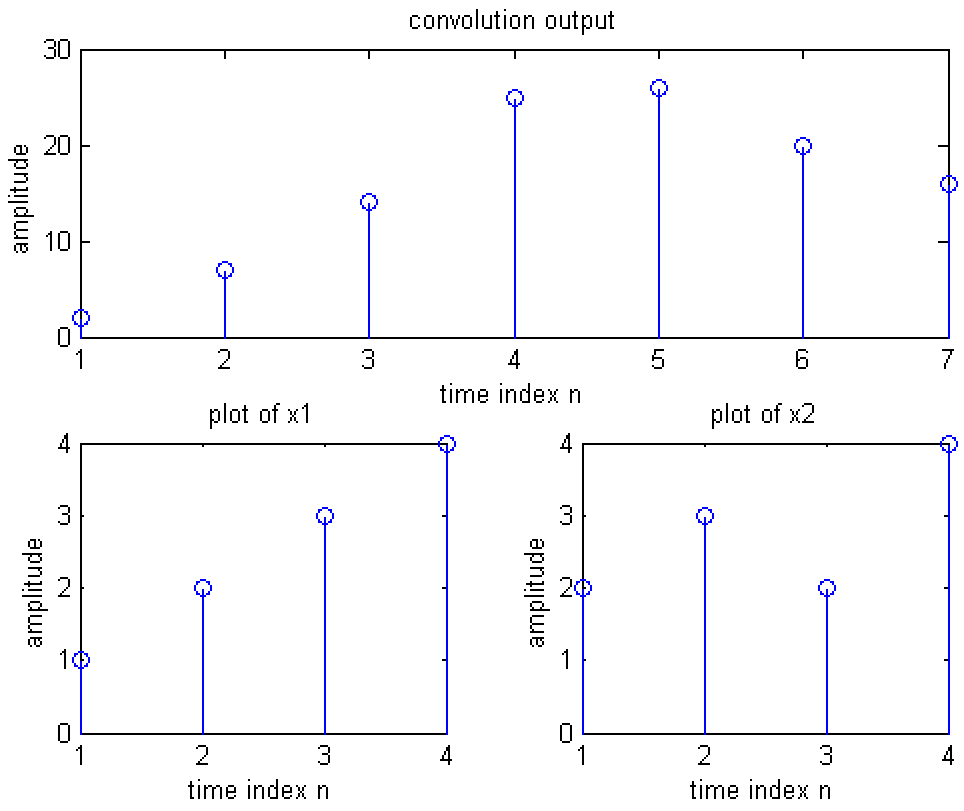
```
%Mainpart of computation
x1=input('Enter first sequence')
x2=input('Enter second sequence')
y=conv(x1,x2);
disp('linear con of x1 & x2 is y=');
disp(y);
%Graphical display part
subplot (2,2,1);
stem(y);
xlabel('time index n');
ylabel('amplitude ');
title('convolution output');
```

```
subplot(2,2,3);
stem(x1);
xlabel('time index n');
ylabel('amplitude ');
title('plot of x1');
subplot(2,2,4);
stem(x2);
xlabel('time index n');
ylabel('amplitude ');
title('plot of x2');
```

Result

```
Enter first sequence    [1 2 3 4]
x1 =  1     2     3     4
Enter second sequence [ 2 3 2 4]
x2 =   2     3     2     4
linear con of x1 & x2 is y=  2  7  14  25  26 20  16
```

Output Waveforms



Level 2 Program (Two sided sequences)

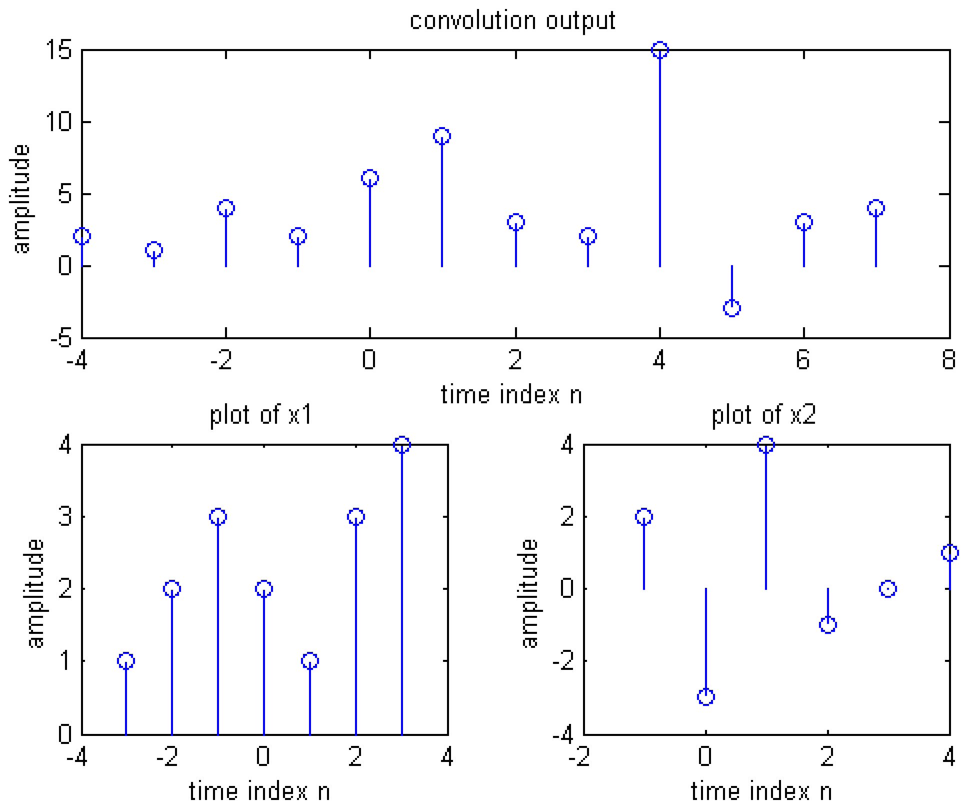
```
%Main part of computation
x1=[1 2 3 2 1 3 4]  %first sequence
n1=-3:3     %time vector of first seq
x2=[ 2 -3 4 -1 0 1] %second seq
n2=-1:4     %time vector of second seq
%Add first elements of time vector
ybegin=n1(1)+n2(1);
%Add last elements of time vector
yend=n1(length(x1))+n2(length(x2));
ny=[ybegin:yend];
y=conv(x1,x2)     ;
```

```
disp('linear con of x1 & x2 is y=');
disp(y);

%Graphical display with time info
subplot (2,1,1);
stem(ny,y);
xlabel('time index n');
ylabel('amplitude ');
title('convolution output');
subplot(2,2,3);
stem(n1,x1);
xlabel('time index n');
ylabel('amplitude ');
title('plot of x1');
subplot(2,2,4);
stem(n2,x2);
xlabel('time index n');
ylabel('amplitude ');
title('plot of x2');
```

Result

```
x1 =1 2 3 2 1 3 4
n1 =-3 -2 -1 0 1 2 3
x2 =2 -3 4 -1 0 1
n2 =-1 0 1 2 3 4
linear con of x1 & x2 is y= 2 1 4 2 6 9 3 2 15 -3 3 4
```

Output waveforms



Open ended Experiments :

Find the convolution of

a. x[n]=[1 2 3] and y[n]=[1 2 3 4]

VIVA Questions :

1. Explain how linear convolution can be obtained.
2. Explain different properties of convolution
3. Give the different methods for performing convolution

B. Circular Convolution of Two Given Sequences

<u>Aim:</u> To obtain circular convolution of two finite duration sequences.

<u>Theory:</u>

- As seen in the last experiment, the output y[n] of a LTI (linear time invariant) system can be obtained by convolving the input x[n] with the system's impulse response h[n].
- The above linear convolution is generally applied to aperiodic sequences. Whereas the Circular Convolution is used to study the interaction of two signals that are periodic.
- The linear convolution sum is $y[n] = x[n] * h[n] = \sum_{k=-\infty}^{+\infty} x[k]h[n-k] = \sum_{k=-\infty}^{+\infty} x[n-k]h[k]$. To compute this equation, the linear convolution involves the following operations:
    - Folding- fold h[k] to get h[-k] (for y[0])
    - Multiplication – $v_k[n]$ = x[k] × h[n-k] (both sequences are multiplied sample by sample)
    - Addition- Sum all the samples of the sequence $v_k[n]$ to obtain y[n]
    - Shifting – the sequence h[-k] to get h[n-k] for the next n.
- The circular convolution sum is $y[n] = x[n] \textcircled{N} h[n] = \sum_{k=-\infty}^{+\infty} x[k]h[\langle n-k\rangle_N]$ where the index $\langle n-k\rangle_N$ implies circular shifting operation and $\langle -k\rangle_N$ implies folding the sequence circularly.
- Steps for circular convolution are the same as the usual convolution, except all index calculations are done "mod N" = "on the wheel".
    - Plot f [m] and h [−m] as shown in Fig. 4.1. (use f(m) instead of x(k))

  - Multiply the two sequences
  - Add to get y[m]
  - "Spin" h[−m] n times Anti Clock Wise (counter-clockwise) to get h[n-m].

x[n] and h[n] can be both finite or infinite duration sequences. If infinite sequences, they should be periodic, and the N is chosen to be at least equal tothe period. If they are finite sequences N is chosen as >= to max(xlength, hlength). Whereas in linear convolution N>= xlength+hlength-1.

- Say $x[n]=\{-2,3,\underset{\uparrow}{1},1\}$ and N = 5, then the x[-1] and x[-2] samples are plotted at x[N-1] = x[-4] and x[N-2] =x[-3] places. Now x[n] is entered as $x[n]=\{\underset{\uparrow}{1},1,0,-2,3\}$.

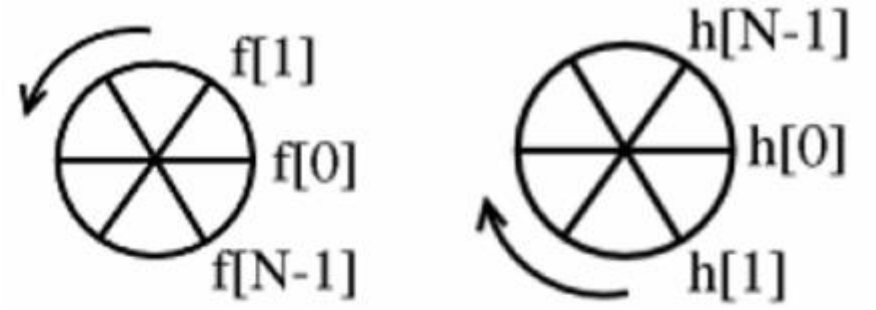


Fig. 4.1 Plotting of f(m) and h(-m) for circular convolution

Algorithm:

1. Input the two sequences as x and h.
2. Circularly convolve both to get output y.
3. Plot the sequences.

MATLAB Implementation:

MATLAB recognizes index 1 to be positive maximum. Index 0 is not recognized. Hence in the below program wherever y, x and h sequences are accessed, the index is added with +1. the modulo index calculation for circular convolution is carried out using the function - MOD Modulus (signed remainder after division). MOD(x,y) is x - y.*floor(x./y) if y ~= 0. By convention, MOD(x,0) is x.The input x and y must be real arrays of the same size, or real scalars. MOD(x,y) has the same sign as y while REM(x,y) has the same sign as x. MOD(x,y) and REM(x,y) are equal if x and y have the same sign, but differ by y if x and y have different signs.

MATLAB Program:

Program to find circular convolution by padding zeroes

```
clc;
clear all;
close all;
g=input('Enter the first sequence');
h=input('Enter the second sequence');
N1=length(g);
N2=length(h);
N=max(N1,N2);
N3=N1-N2;
if(N3>=0)
   h=[h,zeros(1,N3)];
else
   g=[g,zeros(1,-N3)];
end
```

```
for r=1:N,
   y(r)=0;
   for(s=1:N)
      K=r-s+1;
      if(K<=0)
      K=N+K;
   end
   y(r)=y(r)+g(s)*h(K);
end
end
disp('The resultant signal is');y
figure(1);
subplot(3,1,1);
stem(g);
title('The first sequence is');
grid on;
subplot(3,1,2);
stem(h);
title('The second sequence is');
grid on;
subplot(3,1,3);
stem(y);
title('The circularly convolved sequence is');
grid on;
```

Result:

```
Enter the first sequence[1 1 1 1]
Enter the second sequence[1 2 3 4]
The resultant signal is
y =10    10    10    10
```

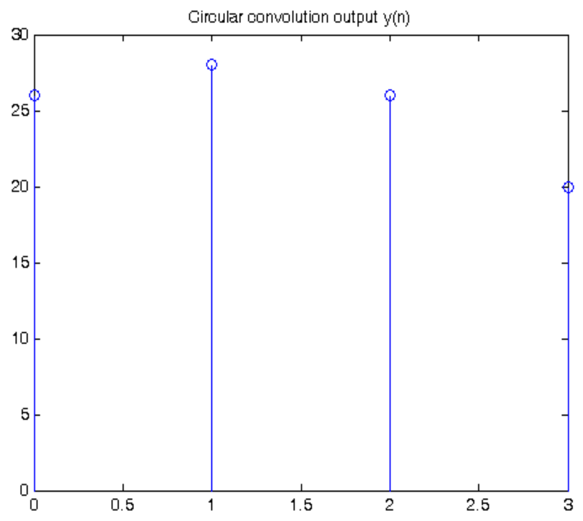
Circular Convolution when the sequences are of equal lengths

```
x=[1 2 3 4];
h= [1 2 3 4];
N=max(length(x),length(h));
%compute the output
for n=0:N-1
   y(n+1)=0;
   for k=0:N-1
      i=mod((n-k),N); %calculation of x index
      if i<0
         i=i+N;
      end     %end of ‘if’
      y(n+1)=y(n+1)+h(k+1)*x(i+1);
   end        %end of inner ‘for loop’
end           %end of outer ‘for loop’
disp('c convolution of x1 &x2 is =y’)
 disp(y);

%plot
n1=0:N-1;
stem(n1,y);
title('Circular convolution output y(n)');
```

Result:

```
circular convolution of x1 & x2 is y=    26    28    26    20
```

Output Waveform



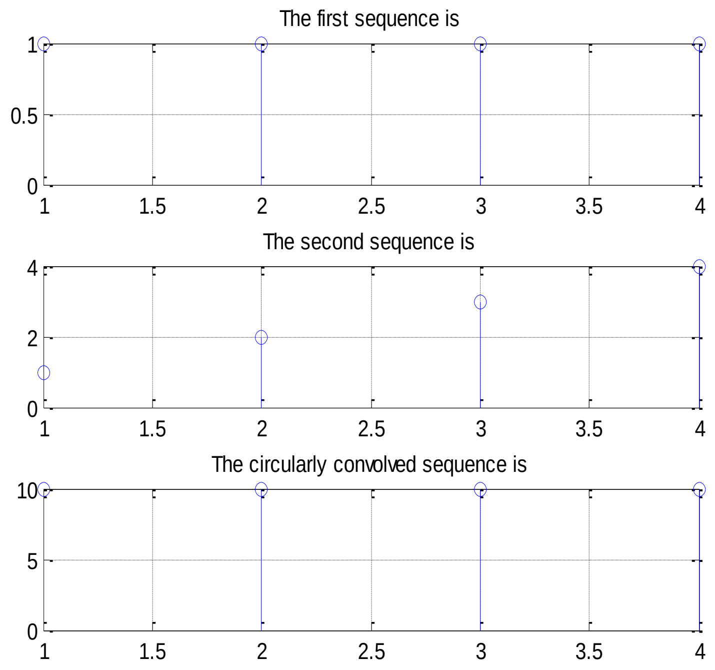
2 Circular convolution using Matrix method

```
clc;
clear all;
close all;
X=input('ENTER THE FIRST SEQUENCE');
H=input('ENTER THE SECOND SEQUENCE IN MATRIX FORMAT');
Y=X*H;
disp('circular convolution');
disp(Y);
stem(Y);
xlabel('n');
ylabel('Y(n)');
grid on;
```

Result:

```
Enter the first sequence[1 1 1 1]
Enter the second sequence in matrix format[1 4 3 2;2 1 4 3;3 2 1 4;4 3 2 1]
Circular convolution is : 10  10  10  10
```

Output waveform



Properties of Convolution:

a). Commutative property

```
clear all;
close all;
x1=[1 2 4 4];
x2=[1 2 7 2];
y1=conv(x1,x2);
y2=conv(x2,x1);
if y1==y2
disp('Commutative property is satisfied');
else
disp('Commutative property is not satisfied');
end
```

b). Associative property

```
clear all;
close all;
x1=[1 2 3 4];
x2=[1 2 1 2];
x3=[3 4 2 1];
y1=conv(x1,x2);
y2=conv(y1,x3);
y3=conv(x2,x3);
y4=conv(x1,y3);
if y2==y4
disp('Associative property is satisfied');
else
disp('Associative property is not satisfied');
end
```

c). Distributive property

```
clear all;
close all;
x1=[1 2 3 4 ];
x2=[1 2 1 2];
x3=[3 4 2 1];
y1=x2+x3;
y2=conv(x1,y1);
y3=conv(x1,x2)+conv(x1,x3);
if y2==y3
disp('distributive property is satisfied');
else
disp('distributive property is not satisfied');
end
```

Open ended Experiments :

Find the circular convolution of x[n]=[1 3 5 6] and y[n]=[1 2]

VIVA Questions :

1. Write A Program to find the circular convolution of two sequences using Matrix Method
2. Explain how circular convolution can be obtained.
3. Explain different properties of circular convolution
4. Give the different methods for performing circular convolution.

EXPERIMENT NO 3

A. Autocorrelation of a Given Sequence and Verification of its Properties.

<u>Aim</u>: To obtain autocorrelation of the given sequence and verify its properties.

<u>Theory</u>:

- Correlation is mathematical technique which indicates whether 2 signals are related and in a precise quantitative way how much they are related. A measure of similarity between a pair of energy signals x[n] and y[n] is given by the cross correlation sequence $r_{xy}[l]$ defined by

  $$r_{xy}[l] = \sum_{n=-\infty}^{\infty} x[n]\,y[n-l];\ l = 0, \pm 1, \pm 2, \ldots .$$

- The parameter ‘l’ called ‘lag’ indicates the time shift between the pair.
- Autocorrelation sequence of x[n] is given by

  $$r_{xx}[l] = \sum_{n=-\infty}^{\infty} x[n]\,x[n-l];\ l = 0, \pm 1, \pm 2, \ldots$$

- Some of the properties of autocorrelation are enumerated below
  - The autocorrelation sequence is an even function i.e., $r_{xx}[l] = r_{xx}[-l]$
  - At zero lag, i.e., at l=0, the sample value of the autocorrelation sequence has its maximum value (equal to the total energy of the signal ε_x) i.e.,

    $$r_{xx}[l] \leq r_{xx}[0] = \varepsilon_x = \sum_{n=-\infty}^{\infty} x^2[n] .$$

    This is verified in Fig. 5.1, where the autocorrelation of the rectangular pulse (square) has a maximum value at l=0. All other samples are of lower value. Also the maximum value = 11 = energy of the pulse $[1^2+1^2+1^2..]$.

- A time shift of a signal does not change its autocorrelation sequence. For example, let y[n]=x[n-k]; then $r_{yy}[l] = r_{xx}[l]$ i.e., the autocorrelation of x[n] and y[n] are the same regardless of the value of the time shift k. This can be verified with a sine and cosine sequences of same amplitude and frequency will have identical autocorrelation functions.

For power signals the autocorrelation sequence is given by

$r_{xx}[l] = \lim_{k\to\infty} \frac{1}{2k+1} \sum_{n=-k}^{k} x[n]x[n-l]; l = 0, \pm 1, \pm 2, \ldots$ and for periodic signals with period N

it is $r_{xx}[l] = \frac{1}{N} \sum_{n=0}^{N-1} x[n]x[n-l]; l = 0, \pm 1, \pm 2, \ldots$ and this **r_{xx}[l] is also**

- **periodic** with N. This is verified in Fig. 5.3 where we use the periodicity property of the autocorrelation sequence to determine the period of the periodic signal y[n] which is x[n] (=cos(0.25*pi*n)) corrupted by an additive uniformly distributed random noise of amplitude in the range [-0.5 0.5]

Algorithm:

1. Input the sequence as x.
2. Use the ‘xcorr’ function to get auto correlated output r.
3. Plot the sequences.

MATLAB Implementation:

MATLAB has the inbuilt function XCORR(A), when A is a vector, is the auto-correlation sequence. If A is of length M vector (M>1), then the xcorr function returns the length 2*M-1 auto-correlation sequence. The zeroth lag of the output correlation is in the middle of the sequence at element M.

XCORR(...,MAXLAG) computes the (auto/cross) correlation over the range of lags: -MAXLAG to MAXLAG, i.e., 2*MAXLAG+1 lags. If missing, default is MAXLAG = M-1.

[C,LAGS] = XCORR(...) returns a vector of lag indices (LAGS).

MATLAB Programs

%Computation of Autocorrelation of a rectangular Sequence

```
x = input('Type in the reference sequence = ');
% Compute the correlation sequence
n = length(x)-1;
r = conv(x, fliplr(x));
disp('autocorrelation sequence r=');
disp(r);
%plot the sequences
subplot(2,1,1)
stem(x);
title('square sequence');
subplot(2,1,2)
k = -n: n;
stem(k, r);
title('autocorrelation output');
xlabel('Lag index');
 ylabel('Amplitude');
```

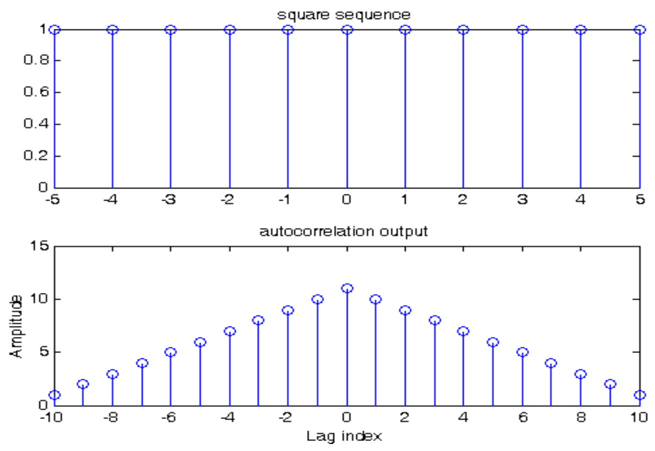
Result:

```
autocorrelation sequence r =  1.0000    2.0000    3.0000    4.0000    5.0000
6.0000    7.0000    8.0000    9.0000   10.0000   11.0000   10.0000    9.0000
8.0000    7.0000    6.0000    5.0000    4.0000 3.0000    2.0000    1.0000
```

% Computation of Auto -correlation sequence using xcorr function

```
x = input('Type in the reference sequence = ');
% Compute the correlation sequence
 [r,lag]=xcorr(x,x);
disp('auto correlation output is =');
disp(r);
stem(lag,r);
xlabel('Lag index');
ylabel('Amplitude');
```

Output Waveform



Inference: Following Autocorr properties are verified

1) Max peak at zero lag (l=0)

2) The autocorr seq is an even function

Exercise: Find the autocorrelation of x[n]=cos (nPi/2) for n=0,1,2,3

VIVA Questions

1. Explain the differences between convolution and correlation
2. List the methods for performing correlation
3. Which are the Mat lab functions used to perform correlation
4. Explain the following functions
   a. rand(1,N)-0.5
   b. fliplr(Y)
   c. xcorr(Y)

B. Cross Correlation of a Given Sequence and Verification of its Properties.

Aim: To obtain cross correlation of the given sequence and verify its properties.

Theory:

- Cross Correlation has been introduced in the last experiment. Comparing the equations for the linear convolution and cross correlation we find that

  $r_{xy}[l]=\sum_{n=-\infty}^{\infty} x[n]\,y[n-l]=\sum_{n=-\infty}^{\infty} x[n]\,y[-(l-n)]=x[l]*y[-l]$. i.e.,

  convolving the reference signal with a folded version of sequence to be shifted (y[n]) results in cross correlation output. (Use 'fliplr' function for folding the sequence for correlation).

- The properties of cross correlation are 1) the cross correlation sequence sample values are upper bounded by the inequality
  $r_{xx}[l] \leq \sqrt{r_{xx}[0] r_{yy}[0]} = \sqrt{\varepsilon_x \varepsilon_y}$
  2) The cross correlation of two sequences x[n] and y[n]=x[n-k] shows a peak at the value of k. Hence cross correlation is employed to compute the exact value of the delay k between the 2 signals. Used in radar and sonar applications, where the received signal reflected from the target is the delayed version of the transmitted signal (measure delay to determine the distance of the target).
  3) The ordering of the subscripts xy specifies that x[n] is the reference sequence that remains fixed in time, whereas the sequence y[n] is shifted w.r.t x[n]. If y[n] is the reference sequence then $r_{yx}[l] = r_{xy}[-l]$. Hence $r_{yx}[l]$ is obtained by time reversing the sequence $r_{xy}[l]$.

Algorithm:

1. Input the sequence as x and y.
2. Use the 'xcorr' function to get cross correlated output r.
3. Plot the sequences.

MATLAB Implementation:

MATLAB has the inbuilt function XCORR: Say C = XCORR(A,B), where A and B are length M vectors (M>1), returns the length 2*M-1 cross-correlation sequence C. If A and B are of different length, the shortest one is zero-padded. Using convolution to implement correlation, the instruction is **FLIPLR Flip matrix in left/right direction.** FLIPLR(X) returns X with row preserved and columns flipped in the left/right direction. X = 1 2 3 becomes 3 2 1.

MATLAB Programs

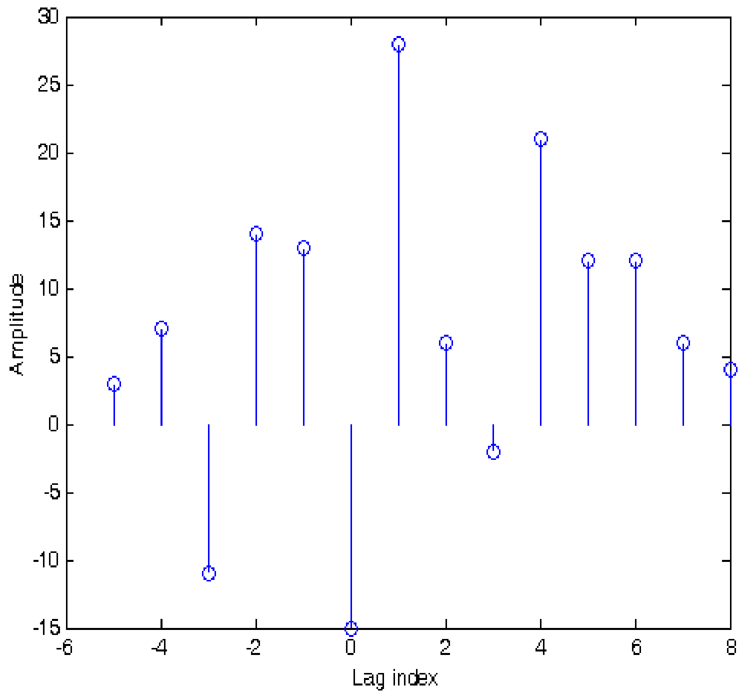
% Computation of Cross-correlation Sequence using folded sequence and convolution

```
x = input('Type in the reference sequence = ');
y = input('Type in the second sequence = ');
% Compute the correlation sequence
n1 = length(x)-1;
n2 = length(y)-1;
r = conv(x,fliplr(y));
disp(‘Cross correlation output is =’);
disp(r);
k = (-n1):n2;%time vector for plotting
stem(k,r);
xlabel('Lag index');
ylabel('Amplitude');
```

Result:

```
Type in the reference sequence = [1 3 -2 1 2 -1 4 4 2]
Type in the second sequence = [2 -1 4 1 -2 3]
Cross correlation output is =
3    7  -11    14    13  -15    28    6   -2    21    12    12    6    4
```

Output Waveforms



% Computation of Cross-correlation sequence using xcorr function

```
x = input('Type in the reference sequence = ');
y = input('Type in the second sequence = ');
% Compute the correlation sequence
 [r,lag]=xcorr(x,y);
disp('Cross correlation output is =');
disp(r);
stem(lag,r);
xlabel('Lag index');
ylabel('Amplitude');
```

Result:

Type in the reference sequence = [1 3 -2 1 2 -1 4 4 2]

Type in the second sequence = [2 -1 4 1 -2 3]

Verification: with xcorr(x,y), the output is

{-0.0000 , -0.0000 , 0.0000, 3.0000, 7.0000, -11.0000, 14.0000, 13.0000, -15.0000, 28.0000, 6.0000, -2.0000, 21.0000, 12.0000, 12.0000, 6.0000 , 4.0000}

%Verification of the properties

x1=[1,2,3,-1,0,0];

x2=[0,0,1,2,3,-1]; %=xr[n-2],i.e., k=2

>> xcorr(x1,x2)

ans = $\{-1,1,5,15,5,\underset{\uparrow}{1},-1,0,0,0,0\}$

Inference:

Strong peak of 15 at lag = -2 implies the delay between xi and x2 is 2.

Also peak =15=energy of xi $(1+2^2+3^2+(-1)^2)$ implies that both xi and x2 are strongly correlated.

%consider the below sequences

xr=[1,2,3,-1];

x2=[3,-1,1,2];

xcorr(xr,x2) ans = $\{2,5,7,\underset{\uparrow}{2},2,10,-3\}$

xcorr(x2,xr) ans = $\{-3,10,2,\underset{\uparrow}{2},7,5,2\}$

Inference:

Strong peak of 10 at lag = 2 implies the delay between xr and x2 is 2, but since 10<15, it implies that xr and x2 are uncorrelated slightly (may be due to noise,etc). $r_{yx}[l]=r_{xy}[-l]$ is verified.

Open ended Experiments : Find the cross correlation of x[n]=[1 2 0 1] and y[n]=[2 2 1 1]

EXPERIMENT NO 4

Solving a Given Difference Equation

<u>Aim</u>: To obtain the impulse response/step response of a system described by the given difference equation

<u>Theory:</u>

- A difference equation with constant coefficients describes a LTI system. For example the difference equation **y[n] + 0.8y[n-2] + 0.6y[n-3] = x[n] + 0.7x[n-1] + 0.5x[n-2]** describes a LTI system of order 3. The coefficients 0.8, 0.7, etc are all constant i.e., they are not functions of time (n). The difference equation y[n]+0.3ny[n-1]=x[n] describes a time varying system as the coefficient 0.3n is not constant.
- The difference equation can be solved to obtain y[n], the output for a given input x[n] by rearranging as y[n] = x[n] + 0.7x[n-1]+0.5x[n-2]-0.8y[n-2]- 0.6y[n-3] and solving.
- The output depends on the input x[n]
    - With **x[n]= δ[n],** an impulse, the computed output y[n] is the **impulse response**.
    - If **x[n]=u[n]**, a **step response** is obtained.
    - If **x[n] = cos(wn)** is a sinusoidal sequence, a **steady state response** is obtained (wherein y[n] is of the same frequency as x[n], with only an amplitude gain and phase shift-refer Fig.7.3).
    - Similarly for any arbitrary sequence of x[n], the corresponding output response y[n] is computed.

- The difference equation containing past samples of output, i.e., y[n-1], y[n-2], etc leads to a recursive system, whose impulse response is of infinite duration (IIR). For such systems the impulse response is computed for a large value of n, say n=100 (to approximate n=∞). The MATLAB function filter is used to compute the impulse response/ step response/ response to any given x[n]. Note: The filter function evaluates the convolution of an infinite sequence (IIR) and x[n], which is not possible with conv function (remember conv requires both the sequences to be finite).
- The difference equation having only y[n] and present and past samples of input (x[n], x[n-k]), represents a system whose impulse response is of finite duration (FIR). The response of FIR systems can be obtained by both the 'conv' and 'filter' functions. The filter function results in a response whose length is equal to that of the input x[n], whereas the output sequence from conv function is of a longer length (xlength + hlength-1).

Algorithm:

1. Input the two sequences as a and b representing the coefficients of y and x.
2. If IIR response, then input the length of the response required (say 100, which can be made constant).
3. Compute the output response using the 'filter' command.
4. Plot the input sequence & impulse response (and also step response, etc if required).

MATLAB Implementation:

MATLAB has an inbuilt function '***filter***' to solve difference equations numerically, given the input and difference equation coefficients (b, a).

y=filter(b,a,x)

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

Given a difference equation $a_0y[n]+a_1y[n-1]+a_2y[n-2]=b_0x[n]+b_2x[n-2]$, the coefficients are written in a vector as $b=[b_0\ 0\ \ b_2]$ and $a=[a_0\ a_1\ a_2]$. Note the zero in b (x[n-1] term is missing).

Also remember a_0, the coefficient of y[n] should always be 1.

For impulse response $x[n]=\{\underset{\uparrow}{1},0,0,0,\ldots\}$ the number of zeros = the length of the IIR response required (say 100 implemented by function zeros(1,99)).

For step response $x[n]=\{\underset{\uparrow}{1},1,1,1,1,\ldots.\}$ the number of ones = the length of the IIR response required-1 (say 100 implemented by function ones(1,100).

Similarly for any given x[n] sequence, the response y[n] can be calculated.

Given Problem

1) Difference equation y(n) - y(n-1) + 0.9y(n-2) = x(n);

Calculate impulse response h(n) and also step response at n=0,…..,100

2) Plot the steady state response y[n] to x[n]=cos(0.05πn)u(n), given y[n]-0.8y[n-1]=x[n]

MATLAB Program:

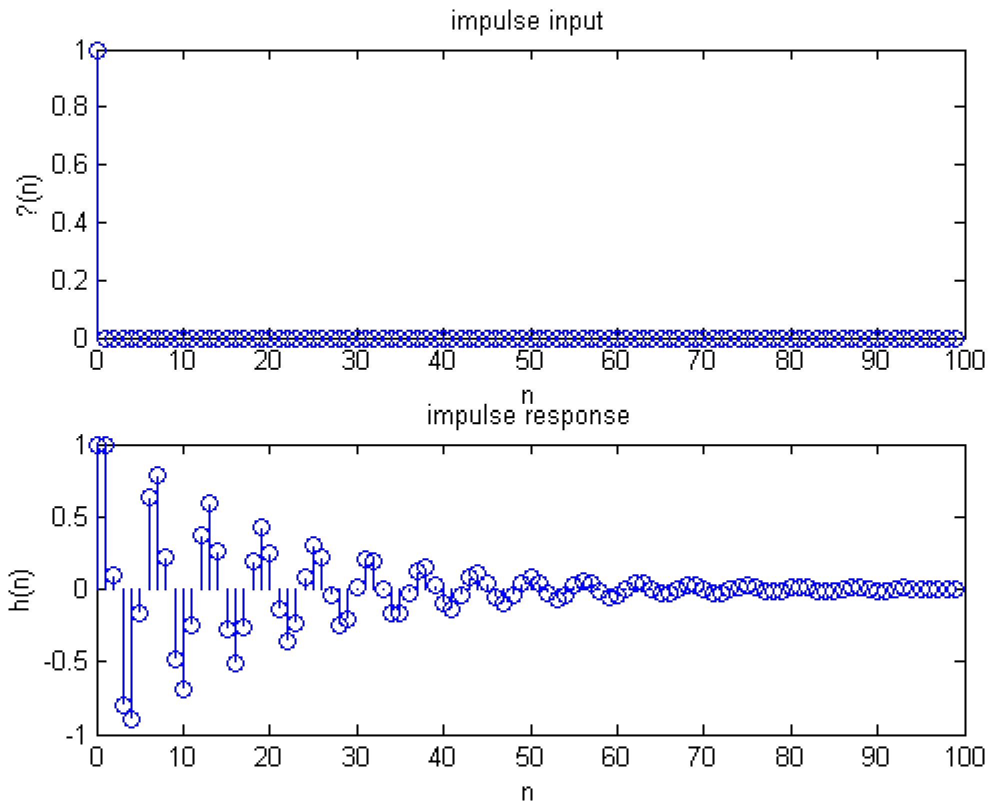
% To find Impulse Response

```
N=input('Length of response required=');
b=[1];          %x[n] coefficient
a=[1,-1,0.9];          %y coefficients

 %impulse input
h=[1,zeros(1,N-1)];
%time vector for plotting
n=0:1:N-1;
 %impulse response
y=filter(b,a,h);

%plot the waveforms
subplot(2,1,1);
stem(n,h);
title('impulse input');
xlabel('n');
ylabel('impulse)');
subplot(2,1,2);
stem(n,y);
title('impulse response');
xlabel('n');
ylabel('y(n)');
```

Result:

Length=100

Output Waveform



%To find step response

```
N=input('Length of response required=');
b=[1];   %x[n] coefficient
a=[1,-1,0.9];   %y coefficients
h=[ones(1,N)];  %step input
n=0:1:N-1;    %time vector for plotting
y=filter(b,a,h);  %step response
```

%plot the waveforms

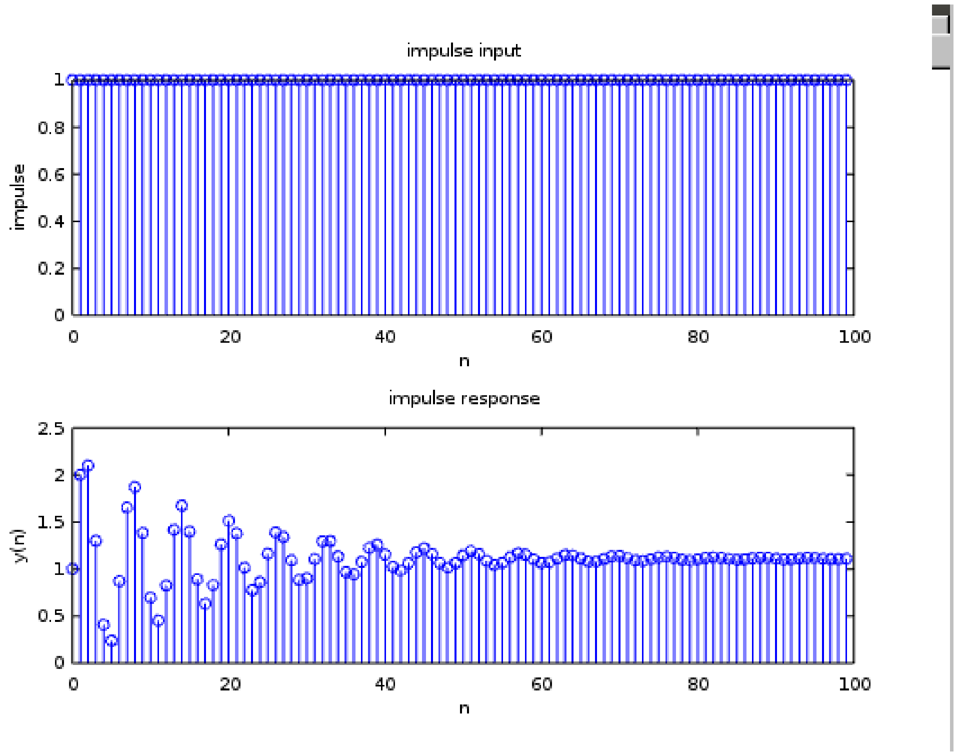
```
subplot(2,1,1);
stem(n,h);
```

```
title('step input');
xlabel('n');
ylabel('u(n)');
subplot(2,1,2);
stem(n,y);
title('step response');
xlabel('n');
ylabel('y(n)');
```

Result:

Length=100

Output Waveform



%To find steady state response

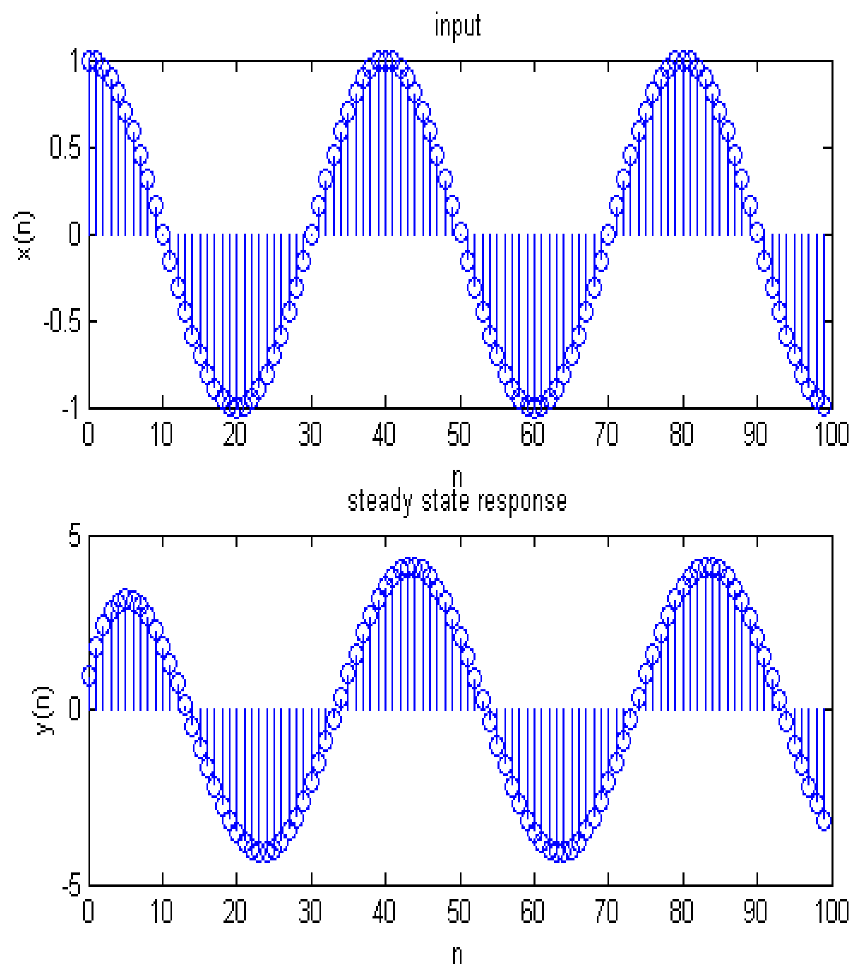
```
%y[n]-0.8y[n-1]=x[n]
N=input('Length of response required=');
```

```
b=[1]; %x[n] coefficient
a=[1,-0.8];  %y coefficients
n=0:1:N-1;   %time vector
h=cos(0.05*pi*n); %  sinusoidal input
y=filter(b,a,h); %steady  state response
subplot(2,1,1);
stem(n,h);
title('steady state input');
xlabel('n');
ylabel('u(n)');
subplot(2,1,2);
stem(n,y);
title('steady state response');
xlabel('n');
ylabel('y(n)');
```

Result:

Length=100

Output Waveform



Open ended Experiments :

Given y[n] = y[n-1]+y[n-2]+x[n-1] find

a. Impulse response
b. Step response
c. Sinusoidal response

VIVA Questions

1.Explain the Matlab function filter

2.How to find various responses using filter function

EXPERIMENT NO 5

Computation of N point DFT of a Given Sequence and to Plot Magnitude and Phase Spectrum.

<u>Aim</u>: To compute DFT of the given sequence.

<u>Theory</u>:

- Discrete Fourier Transform (DFT) is used for performing frequency analysis of discrete time signals. DFT gives a discrete frequency domain representation whereas the other transforms are continuous in frequency domain.
- The N point DFT of discrete time signal x[n] is given by the equation

$$X(k)=\sum_{n=0}^{N-1} x[n]e^{\frac{-j2\pi kn}{N}}; \quad k=0,1,2,\ldots N\text{-}1$$

Where N is chosen such that $N \geq L$, where L=length of x[n].

- The inverse DFT allows us to recover the sequence x[n] from the frequency samples.

$$x[n]=\frac{1}{N}\sum_{k=0}^{N-1} x[n]e^{\frac{j2\pi kn}{N}}; \quad n=0,1,2,\ldots N\text{-}1$$

- X(k) is a complex number (remember e^{jw}=cosw + jsinw). It has both magnitude and phase which are plotted versus k. These plots are magnitude and phase spectrum of x[n]. The 'k' gives us the frequency information.
- Here k=N in the frequency domain corresponds to sampling frequency (fs). Increasing N, increases the frequency resolution, i.e., it improves the spectral characteristics of the sequence. For example if fs=8kHz and N=8 point DFT, then in the resulting spectrum, k=1 corresponds to 1kHz frequency. For the same fs and x[n], if N=80

point DFT is computed, then in the resulting spectrum, k=1 corresponds to 100Hz frequency. Hence, the resolution in frequency is increased.

- Since $N \geq L$, increasing N to 8 from 80 for the same x[n] implies x[n] is still the same sequence (<8), the rest of x[n] is padded with zeros. This implies that there is no further information in time domain, but the resulting spectrum has higher frequency resolution. This spectrum is known as 'high density spectrum' (resulting from zero padding x[n]). Instead of zero padding, for higher N, if more number of points of x[n] are taken (more data in time domain), then the resulting spectrum is called a 'high resolution spectrum'.

Algorithm:

1. Input the sequence for which DFT is to be computed.
2. Input the length of the DFT required (say 4, 8, >length of the sequence).
3. Compute the DFT using the 'fft' command.
4. Plot the magnitude & phase spectra.

MATLAB Implementation:

MATLAB has an inbuilt function 'FFT' which computes the Discrete Fourier transform.

FFT(X) is the discrete Fourier transform (DFT) of vector X. For length N input vector x, the DFT is a length N vector X, with elements N. FFT(X,N) is the N-point FFT, padded with zeros if X has less than N points and truncated if it has more.

The magnitude spectrum is computed using the function ABS Absolute value.

ABS(X) is the absolute value of the elements of X. When X is complex, ABS(X) is the complex modulus (magnitude) of the elements of X.
The phase spectrum is computed using the function ANGLE Phase angle.
ANGLE (H) returns the phase angles, in radians, of a matrix with complex elements.

Matlab Program:

N-point FFT and IFFT of a given sequence

```
clc;
clear all;
close all;
x=input('Enter the sequence x(n)');
N=length(x);
y=fft(x,N);
disp(y);
mag=abs(y);
phase=angle(y);
subplot(2,2,1);
stem(x);
grid on;
title('Input Data Sequence x(n)');
subplot(2,2,3);
stem(mag);
grid on;
title('Magnitude Plot');
subplot(2,2,2);
```

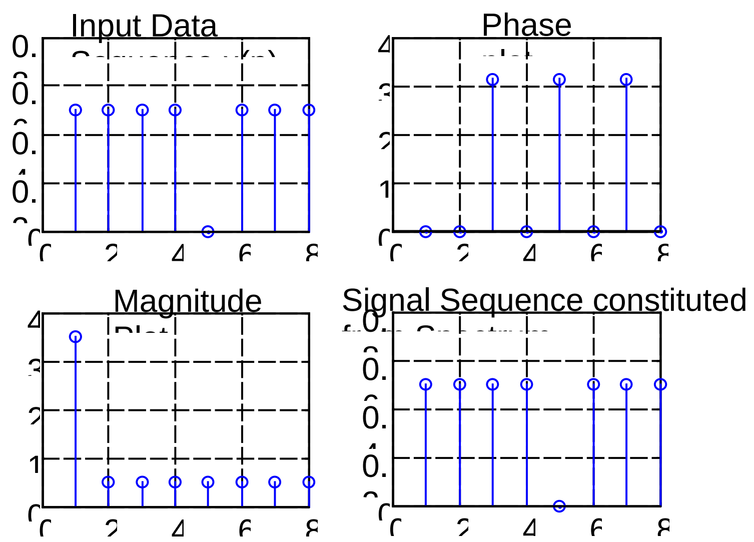
```
stem(phase);
grid on;
title('Phase plot');
z=ifft(y,N);
mag1=real(z);
subplot(2,2,4);
stem(mag1);
grid on;
title('Signal Sequence constituted from Spectrum');
```

<u>Result:</u>

Enter the sequence x(n) [0.5 0.5 0.5 0.5 0 0.5 0.5 0.5]

```
    3.5000    0.5000   -0.5000    0.5000   -0.5000    0.5000   -0.5000
0.5000
```

<u>Output waveform</u>



DFT of a given sequence

<u>To find the N-point DFT of a sequence using a function file dft</u>

```
Clc:
xn=input('enter the sequence x(n)');
N=input('enter the number of points of computation');
y=dft(xn,N);
disp(y);
amp=abs(y);
ph=angle(y);
subplot(2,1,1)
stem(amp);
xlabel('MAGNITUDE');
ylabel('K');
title('magnitude plot');
stem(ph*180/pi);
xlabel('phase');
ylabel('K');
title('phase plot');
```

%function to find dft

```
function[xk] =dft(xn,N)
n=[0:1:N-1];
k=[0:1:N-1];
WN=exp(-j*2*pi/N);
nk=n`*k;
WNnk=WN.^nk;
xk=xn*WNnk;
```

<u>Result:</u>

```
Enter the sequence x(n)=[1 1 1 1 0 0 0 0]
Xk=4.000  1.0000-2.4142i -0.0000-0.0000i   0.0000-0.0000i
1.0000+0.4142i   0.0000-0.0000i
```

IDFT of a given sequence

<u>To find the N-point IDFT of a sequence using a function file idft</u>

```
Clc:
xk=input('enter the sequence x(k)');
N=input('enter the number of points of computation');
y=idft(xk,N);
disp(y);
amp=abs(y);
ph=angle(y);
subplot(2,1,1)
stem(amp);
xlabel('MAGNITUDE');
ylabel('K');
title('magnitude plot');
stem(ph*180/pi);
xlabel('phase');
ylabel('K');
title('phase plot');
%function to find idft
function[xn] =idft(xk,N)
```

```
n=[0:1:N-1];
k=[0:1:N-1];
WN=exp(-j*2*pi/N);
nk=n`*k;
WNnk=WN.^(-nk);
xn=(xk*WNnk)/N;
```

Result:

Enter the sequence Xk=[4, 1-2.4142i ,0,1-0.4142i,0, 1.0000+0.4142i , 0,1+2.4142i]

x(n)=[1 1 1 1 0 0 0 0]

Inputs -

Run1: Give 4 point sequence and find 4 point DFT

Run2: Give 8 point sequence and find 8 point DFT

Run 3: Give a Sinusoidal Signal with a frequency 250 Hz with fs = 8 KHz and sample it for 64 points and perform 64 point DFT on it.

Run 4: Give a Sinusoidal Signal with a frequency 250 Hz and 500 Hz with fs = 8 KHz and sample it for 64 points and perform 64 point DFT on it.

Open ended Experiments :

1.Find the 4 point DFT of x[n]=cosnπ/4

2. Find the 3 point IDFT of X[K]=[1-j2,-1,1+j2]

VIVA Questions:

1. Explain DFT
2. What are the methods of finding DFT
3. Explain differences between DFT and FFT
4. Explain how functions can be declared in MATLB.
5. Explain MATLAB function `fft(x)`
6. How can user create a MATLAB function

EXPERIMENT NO 6

A. Verification of DFT properties (Linearity and Parseval's Theorem)

Matlab code for Linearity properly & Parseval's Theorem

i) Verificaton of Parseval's Theorem

Program

```
x1=[1 2 3 4]
X1=fft(x1,4);
en1=sum(x1.^2);
en2=(1/4)*(sum(abs(X1.^2)))
if en1==en2
disp('Parseval Theoreem Proved')
disp('Energy =')
disp(en2)
end
```

Result

```
x1 =1 2 3 4
en2 = 30
Parseval Theorem Proved
Energy =30
```

ii) Verification of Linearity Property

Program

```
x1=[1 2 3 4]
x2=[1 1 1 1]
a1=a2=1;
term1=a1.*x1+a2.*x2;
LHS=fft(term1,4);
term2=fft(a1.*x1);
term3=fft(a2.*x2);
RHS=term2+term3;
if(LHS==RHS)
disp('Lineraity Property Proved')
else
```

```
disp('Non Linear')
end
```

Result

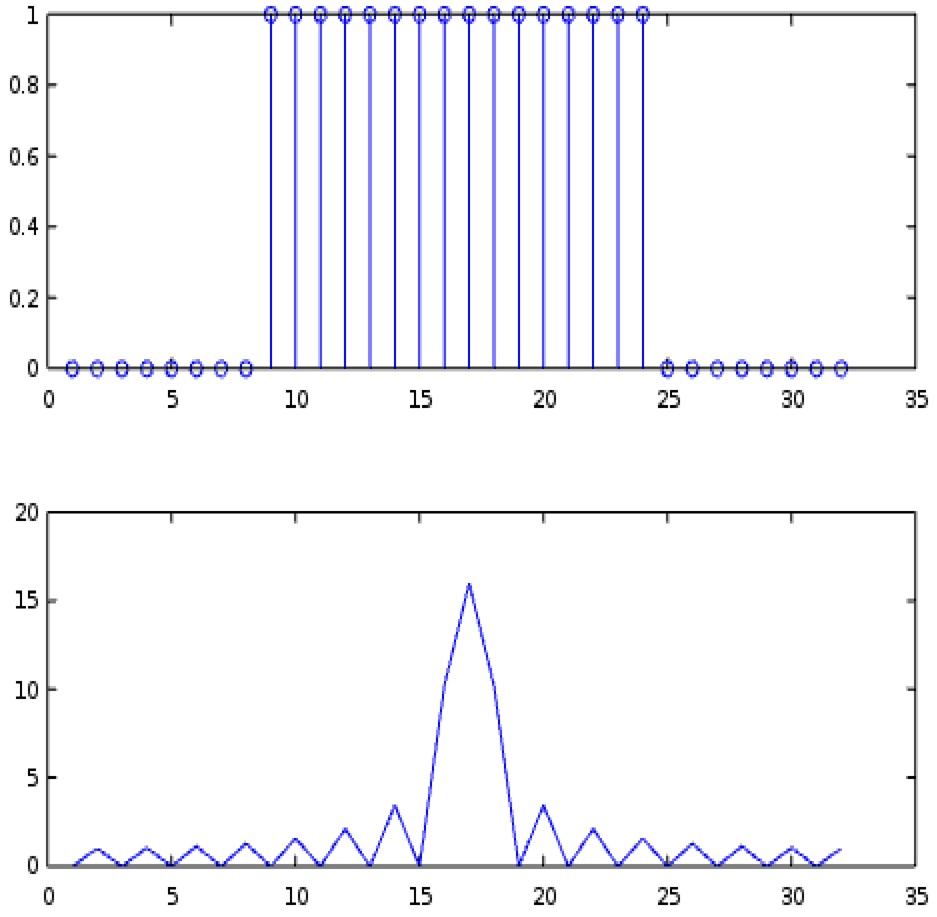
```
x1 =1 2 3 4

x2 =1 1 1 1

Linearity Property Proved
```

B. DFT computation of Square Pulse

Program

```
x1=[zeros(1,8),ones(1,16),zeros(1,8)]
xk=fft(x1)
xk2=fftshift(xk);
subplot(2 ,1, 1)
stem(x1)
subplot(2 ,1, 2)
plot(abs(xk2));
```

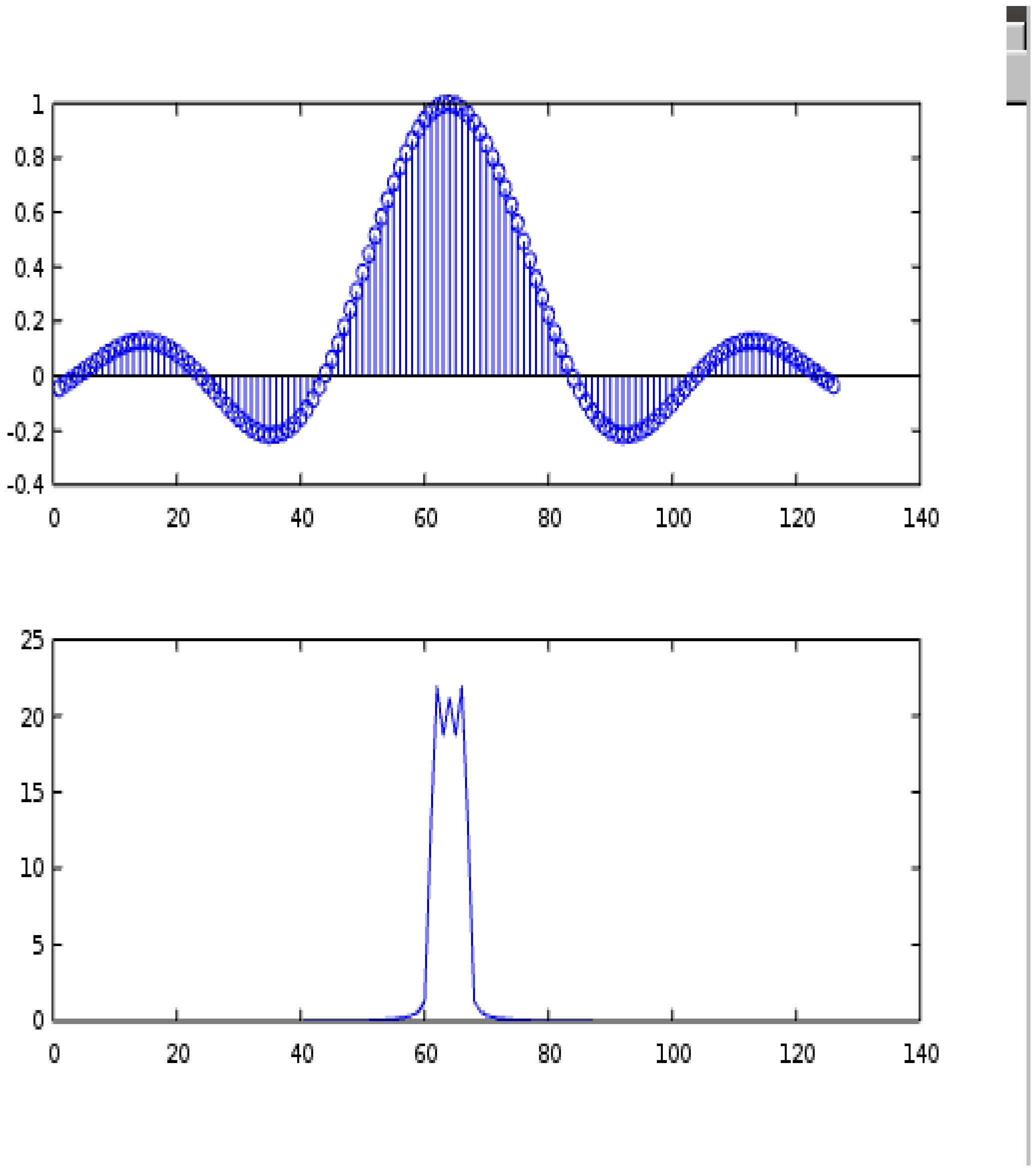
Result



C. DFT computation of Sinc Pulse

```
x=-pi:0.05:pi;
x1=sinc(x)
xk=fft(x1)
xk2=fftshift(xk);
subplot(2 ,1, 1)
stem(x1)
subplot(2 ,1, 2)
plot(abs(xk2));
```

Result



EXPERIMENT NO 7

Design and Implementation of FIR Filter to Meet Given Specifications

<u>Aim</u>: To design and implement a FIR filter for given specifications.

Theory:

There are two types of systems – Digital filters (perform signal filtering in time domain) and spectrum analyzers (provide signal representation in the frequency domain). The design of a digital filter is carried out in 3 steps-specifications, approximations and implementation.

DESIGNING AN FIR FILTER (using window method):

<u>Method I</u>: Given the order N, cutoff frequency fc, sampling frequency fs and the window.

- Step 1: Compute the digital cut-off frequency Wc (in the range $-\pi < Wc < \pi$, with π corresponding to fs/2) for fc and fs in Hz. For example let fc=400Hz, fs=8000Hz

  $$Wc = 2*\pi*fc / fs = 2*\pi*400/8000 = 0.1*\pi \text{ radians}$$

  For MATLAB the Normalized cut-off frequency is in the range 0 and 1, where 1 corresponds to fs/2 (i.e.,fmax)). Hence to use the MATLAB commands

  $$wc = fc / (fs/2) = 400/(8000/2) = 0.1$$

  Note: if the cut off frequency is in radians then the normalized frequency is computed as $wc = Wc / \pi$
- Step 2: Compute the Impulse Response h(n) of the required FIR filter using the given Window type and the response type (lowpass,

bandpass, etc). For example given a rectangular window, order N=20, and a high pass response, the coefficients (i.e., h[n] samples) of the filter are computed using the MATLAB inbuilt command ‘fir1’ as

```
h =fir1(N, wc , 'high', boxcar(N+1));
```

Note: In theory we would have calculated h[n]=hd[n]×w[n], where hd[n] is the desired impulse response (low pass/ high pass,etc given by the sinc function) and w[n] is the window coefficients. We can also plot the window shape as stem(boxcar(N)).

Plot the frequency response of the designed filter h(n) using the freqz function and observe the type of response (lowpass / highpass /bandpass).

Method 2:

Given the pass band (wp in radians) and Stop band edge (ws in radians) frequencies, Pass band ripple Rp and stopband attenuation As.

- Step 1: Select the window depending on the stopband attenuation required. Generally if As>40 dB, choose Hamming window. (Refer table)
- Step 2: Compute the order N based on the edge frequencies as

  Transition bandwidth = tb=ws-wp;

  N=ceil (6.6*pi/tb);
- Step 3: Compute the digital cut-off frequency Wc as

  Wc=(wp+ws)/2

  Now compute the normalized frequency in the range 0 to 1 for MATLAB as

  wc=Wc/pi;

Note: In step 2 if frequencies are in Hz, then obtain radian frequencies (for computation of tb and N) as wp=2*pi*fp/fs, ws=2*pi*fstop/fs, where fp, fstop and fs are the passband, stop band and sampling frequencies in Hz

- Step 4: Compute the Impulse Response h(n) of the required FIR filter using N, selected window, type of response(low/high,etc) using 'fir1' as in step 2 of method 1.

IMPLEMENTATION OF THE FIR FILTER

1. Once the coefficients of the FIR filter h[n] are obtained, the next step is to simulate an input sequence x[n], say input of 100, 200 & 400 Hz (with sampling frequency of fs), each of 20/30 points. Choose the frequencies such that they are >, < and = to fc.
2. Convolve input sequence x[n] with Impulse Response, i.e., x (n)*h (n) to obtain the output of the filter y[n]. We can also use the 'filter' command.
3. Infer the working of the filter (low pass/ high pass, etc).

MATLAB IMPLEMENTATION

FIR1 Function

B = FIR1(N,Wn) designs an N'th order lowpass FIR digital filter and returns the filter coefficients in length N+1 vector B. The cut-off frequency Wn must be between 0 < Wn < 1.0, with 1.0 corresponding to half the sample rate. The filter B is real and has linear phase, i.e., even symmetric coefficients obeying B(k) = B(N+2-k), k = 1,2,...,N+1.

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

[W1 W2]. If Wn is a multi-element vector, Wn = [W1 W2 W3 W4 W5 ... WN], FIR1 returns an order N multiband filter with bands

0 < W < W1, W1 < W < W2, ..., WN < W < 1.

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

For FIR filter enter A=1 and B = h[n] coefficients. Appropriately choose N as 128, 256, etc

Matlab Program:

1 RECTANGULAR WINDOW

```
clc;
clear all;
close all;
lp=input('Enter the passband ripple:');
ls=input('Enter the stopband ripple:');
fp=input('Enter the passband frequency:');
fs=input('Enter the stopband frequency:');
f=input('Enter the sampling frequency:');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(lp*ls))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
```

```
n1=n+1;
if (rem(n,2)~=0)
   n1=n;
   n=n-1;
end
y=boxcar(n1);% or y=rect(n1);
```

% Low Pass Filter

```
b=fir1(n,wp,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(a)Normalised Frequency-->');
grid on;
```

% High Pass Filter

```
b=fir1(n,wp,'high',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,2);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(b)Normalised Frequency-->');
grid on;
```

% Band Pass Filter

```
wn=[wp ws];
b=fir1(n,wn,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(c)Normalised Frequency-->');
grid on;
```

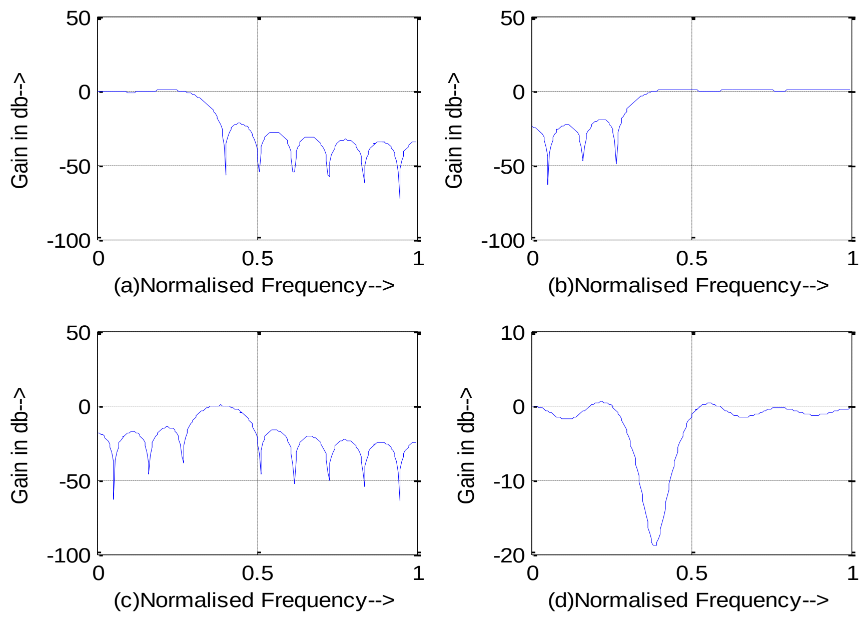
% Band Stop Filter

```
b=fir1(n,wn,'stop',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,4);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(d)Normalised Frequency-->');
grid on
```

Result

```
Enter the passband ripple:          0.05
Enter the stopband ripple:          0.04
Enter the passband frequency:   1500
Enter the stopband frequency:   2000
Enter the sampling frequency:   9000
```

Output waveform



2 HAMMING WINDOW

```
clc;
clear all;
close all;
lp=input('Enter the passband ripple:');
ls=input('Enter the stopband ripple:');
fp=input('Enter the passband frequency:');
fs=input('Enter the stopband frequency:');
f=input('Enter the sampling frequency:');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(lp*ls))-13;
dem=14.6*(fs-fp)/f;
```

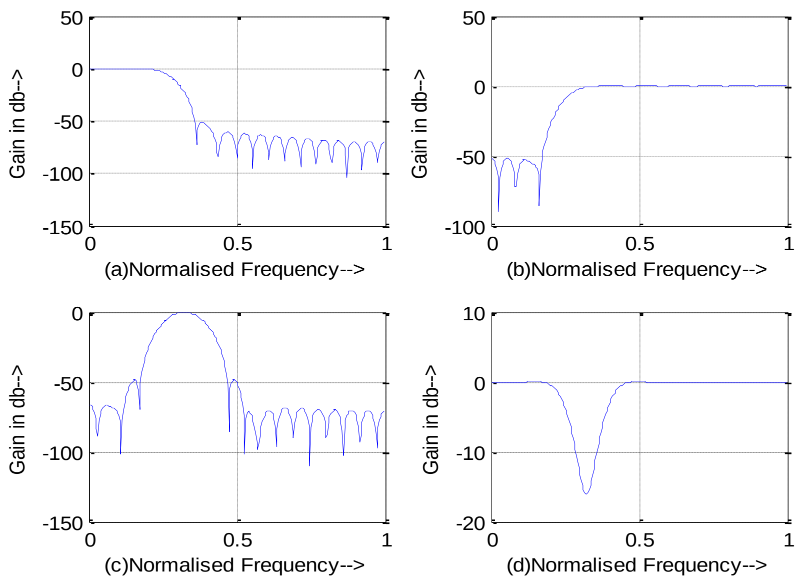
```
n=ceil(num/dem);
n1=n+1;
if (rem(n,2)~=0)
    n1=n;
    n=n-1;
end
y=hamming(n1);

% Low Pass Filter
b=fir1(n,wp,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(a)Normalised Frequency-->');
grid on;

% High Pass Filter
b=fir1(n,wp,'high',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,2);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(b)Normalised Frequency-->');
grid on;
```

```
% Band Pass Filter
wn=[wp ws];
b=fir1(n,wn,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(c)Normalised Frequency-->');
grid on;

% Band Stop Filter
b=fir1(n,wn,'stop',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,4);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(d)Normalised Frequency-->');
grid on;
```

<u>Result:</u>

```
Enter the passband ripple:          0.05
Enter the stopband ripple:          0.001
Enter the passband frequency:   1200
Enter the stopband frequency:   1700
Enter the sampling frequency:   9000
```

Output waveform



3 BARTLET

```
clc;
clear all;
close all;
lp=input('Enter the passband ripple:');
ls=input('Enter the stopband ripple:');
fp=input('Enter the passband frequency:');
fs=input('Enter the stopband frequency:');
f=input('Enter the sampling frequency:');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(lp*ls))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
if (rem(n,2)~=0)
```

```
    n1=n;
    n=n-1;
end
y=bartlett(n1);
```

% Low Pass Filter

```
b=fir1(n,wp,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(a)Normalised Frequency-->');
grid on;
```

% High Pass Filter

```
b=fir1(n,wp,'high',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,2);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(b)Normalised Frequency-->');
grid on;
```

% Band Pass Filter

```
wn=[wp ws];
```

```
b=fir1(n,wn,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(c)Normalised Frequency-->');
grid on;
```

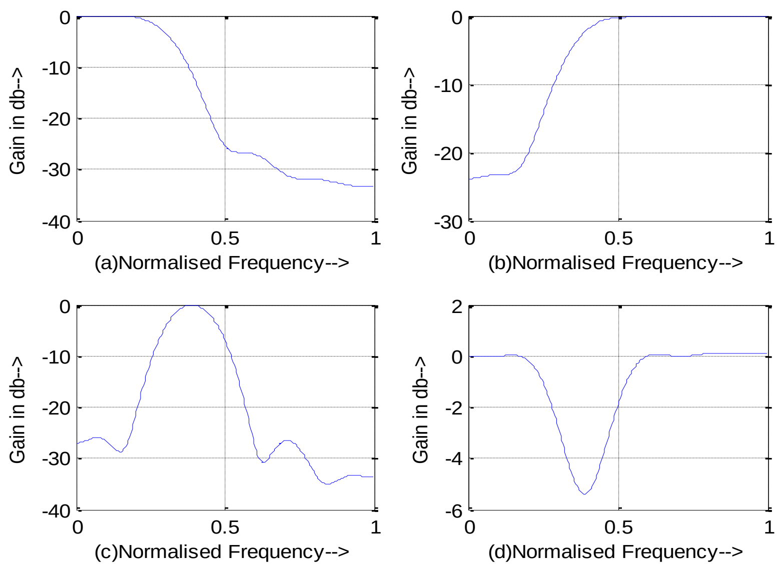
<u>% Band Stop Filter</u>

```
b=fir1(n,wn,'stop',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,4);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(d)Normalised Frequency-->');
grid on
```

<u>Result:</u>

```
Enter the passband ripple:           0.05
Enter the stopband ripple:           0.04
Enter the passband frequency:   1500
Enter the stopband frequency:   2000
Enter the sampling frequency:   9000
```

<u>Output Waveform</u>



4 HANNING

```
clc;
clear all;
close all;
lp=input('Enter the passband ripple:');
ls=input('Enter the stopband ripple:');
fp=input('Enter the passband frequency:');
fs=input('Enter the stopband frequency:');
f=input('Enter the sampling frequency:');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(lp*ls))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
if (rem(n,2)~=0)
```

```
    n1=n;
    n=n-1;
end
y=hanning(n1);
```

% Low Pass Filter

```
b=fir1(n,wp,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(a)Normalised Frequency-->');
grid on;
```

% High Pass Filter

```
b=fir1(n,wp,'high',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,2);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(b)Normalised Frequency-->');
grid on;
```

% Band Pass Filter

```
wn=[wp ws];
```

```
b=fir1(n,wn,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(c)Normalised Frequency-->');
grid on;
```

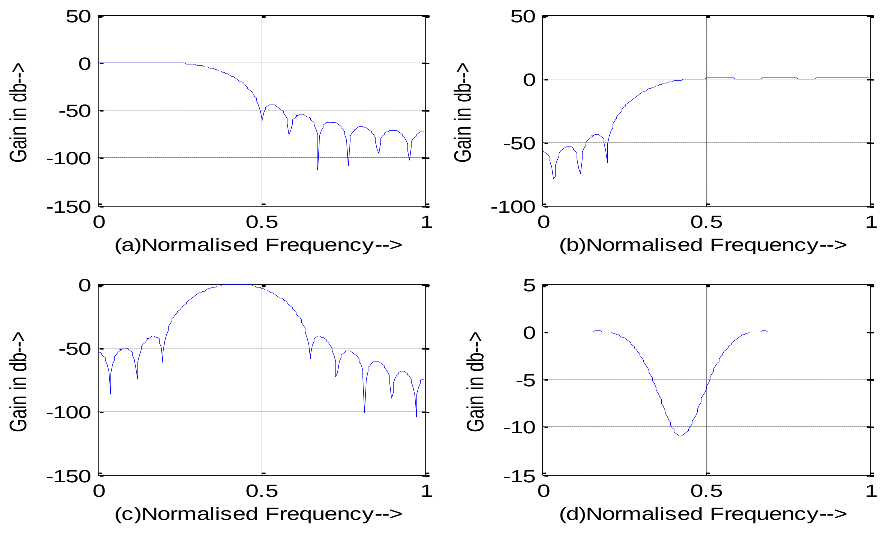
% Band Stop Filter

```
b=fir1(n,wn,'stop',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,4);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(d)Normalised Frequency-->');
grid on
```

Result

```
Enter the passband ripple:          0.03
Enter the stopband ripple:          0.01
Enter the passband frequency:   1400
Enter the stopband frequency:   2000
Enter the sampling frequency:   8000
```

Output Waveform



6 KAISER

```
clc;
clear all;
close all;
lp=input('Enter the passband ripple:');
ls=input('Enter the stopband ripple:');
fp=input('Enter the passband frequency:');
fs=input('Enter the stopband frequency:');
f=input('Enter the sampling frequency:');
beta=input('Enter the beta value:');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(lp*ls))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
```

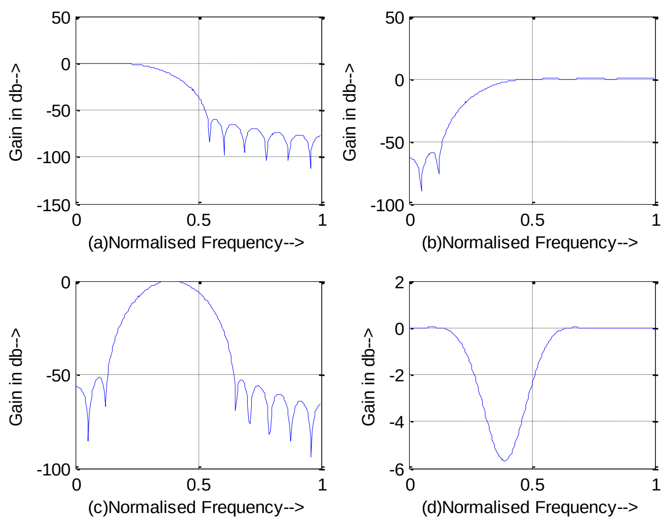
```
if (rem(n,2)~=0)
   n1=n;
   n=n-1;
end
y=kaiser(n1,beta);

% Low Pass Filter
b=fir1(n,wp,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(a)Normalised Frequency-->');
grid on;
% High Pass Filter
b=fir1(n,wp,'high',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,2);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(b)Normalised Frequency-->');
grid on;

% Band Pass Filter
wn=[wp ws];
```

```
b=fir1(n,wn,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(c)Normalised Frequency-->');
grid on;

% Band Stop Filter
b=fir1(n,wn,'stop',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,4);
plot(o/pi,m);
ylabel('Gain in db-->');
xlabel('(d)Normalised Frequency-->');
grid on
```

Result:

```
Enter the passband ripple:            0.05
Enter the stopband ripple:            0.04
Enter the passband frequency:   1500
Enter the stopband frequency:   2000
Enter the sampling frequency:   9000
Enter the beta value: 5.8
```

Output Waveform



VIVA Questions:

1. What is a Filter? What are its specifications?
2. Mention the types of filters.
3. Compare analog and digital filters.
4. Define a FIR filter.
5. What are the important properties of FIR filter?
6. What are the window techniques used?
7. Why the Kaiser window is important?
8. How do you calculate the length of Kaiser Window?
9. What are the functions used in designing a FIR filter in Matlab

EXPERIMENT NO 8

Design and Implementation of IIR filter to meet given specifications

Aim: To design and implement an IIR filter for given specifications.

Theory:

There are two methods of stating the specifications as illustrated in previous program. In the first program, the given specifications are directly converted to digital form and the designed filter is also implemented. In the last two programs the butterworth and chebyshev filters are designed using bilinear transformation (for theory verification).

Method I: Given the order N, cutoff frequency fc, sampling frequency fs and the IIR filter type (butterworth, cheby1, cheby2).

- Step 1: Compute the digital cut-off frequency Wc (in the range $-\pi < Wc < \pi$, with π corresponding to fs/2) for fc and fs in Hz. For example let fc=400Hz, fs=8000Hz

  $$Wc = 2*\pi* fc / fs = 2* \pi * 400/8000 = 0.1* \pi \text{ radians}$$

  For MATLAB the Normalized cut-off frequency is in the range 0 and 1, where 1 corresponds to fs/2 (i.e.,fmax)). Hence to use the MATLAB commands

  $$wc = fc / (fs/2) = 400/(8000/2) = 0.1$$

  Note: if the cut off frequency is in radians then the normalized frequency is computed as $wc = Wc / \pi$

- Step 2: Compute the Impulse Response [b,a] coefficients of the required IIR filter and the response type (lowpass, bandpass, etc) using the appropriate butter, cheby1, cheby2 command. For example given a butterworth filter, order N=2, and a high pass response, the coefficients [b,a] of the filter are computed using the MATLAB inbuilt command 'butter' as [b,a] =butter(N, wc , 'high');

Method 2:

Given the pass band (Wp in radians) and Stop band edge (Ws in radians) frequencies, Pass band ripple Rp and stopband attenuation As.

- Step 1: Since the frequencies are in radians divide by π to obtain normalized frequencies to get wp=Wp/pi and ws=Ws/pi
  If the frequencies are in Hz (note: in this case the sampling frequency should be given), then obtain normalized frequencies as wp=fp/(fs/2), ws=fstop/(fs/2), where fp, fstop and fs are the passband, stop band and sampling frequencies in Hz
- Step 2: Compute the order and cut off frequency as
  [N, wc] = BUTTORD(wp, ws, Rp, Rs)
- Step 3: Compute the Impulse Response [b,a] coefficients of the required IIR filter and the response type as [b,a] =butter(N, wc , 'high');

IMPLEMENTATION OF THE IIR FILTER

1. Once the coefficients of the IIR filter [b,a] are obtained, the next step is to simulate an input sequence x[n], say input of 100, 200 & 400 Hz (with sampling frequency of fs), each of 20/30 points. Choose the frequencies such that they are >, < and = to fc.
2. Filter the input sequence x[n] with Impulse Response, to obtain the output of the filter y[n] using the 'filter' command.
3. Infer the working of the filter (low pass/ high pass, etc).

MATLAB IMPLEMENTATION

BUTTORD Butterworth filter order selection.

[N, Wn] = BUTTORD(Wp, Ws, Rp, Rs) returns the order N of the lowest order digital Butterworth filter that loses no more than Rp dB in the passband and has at least Rs dB of attenuation in the stopband. Wp and Ws are the passband and stopband edge frequencies, normalized from 0 to 1 (where 1 corresponds to pi radians/sample). For example,

Lowpass: Wp = .1, Ws = .2 Highpass: Wp = .2, Ws = .1

Bandpass: Wp = [.2 .7], Ws = [.1 .8] Bandstop: Wp = [.1 .8], Ws = [.2 .7]

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

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

When Rp is chosen as 3 dB, the Wn in BUTTER is equal to Wp in BUTTORD.

BUTTER Butterworth digital and analog filter design.

[B,A] = BUTTER(N,Wn) designs an Nth order lowpass digital Butterworth filter and returns the filter coefficients in length N+1 vectors B (numerator) and A (denominator). The coefficients are listed in descending powers of z. The cutoff frequency Wn must be 0.0 < Wn < 1.0, with 1.0 corresponding to half the sample rate. If Wn is a two-element vector, Wn = [W1 W2], BUTTER returns an order 2N bandpass filter with passband W1 < W < W2. [B,A] = BUTTER(N,Wn,'high') designs a highpass filter. [B,A] = BUTTER(N,Wn,'stop') is a bandstop filter if Wn = [W1 W2].

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

MATLAB PROGRAMS

Butterworth Filters

1. Butterworth LPF

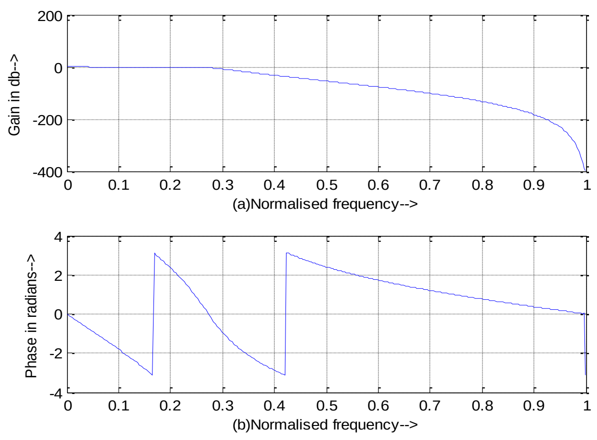
```
clc;
clear all;
close all;
lp=input('Enter the passband attenuation:');
ls=input('Enter the stopband attenuation:');
wp=input('Enter the passband frequency:');
ws=input('Enter the stopband frequency:');
fs=input('Enter the sampling frequency:');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=buttord(w1,w2,lp,ls);disp(n);disp(wn)
[b,a]=butter(n,wn);
w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure(1);
subplot(2,1,1);
title('Butterworth Low Pass Filter');
plot(om/pi,m);
```

```
grid on;
ylabel('Gain in db-->');
xlabel('(a)Normalised frequency-->');
subplot(2,1,2);
plot(om/pi,an);
xlabel('(b)Normalised frequency-->');
ylabel('Phase in radians-->');
grid on;
```

Result:

```
Enter the passband attenuation:     0.5
Enter the stopband attenuation:     50
Enter the passband frequency:       1200
Enter the stopband frequency:       2400
Enter the sampling frequency:       10000
```

Output Wave form



2 Butterworth HPF

```
clc;
```

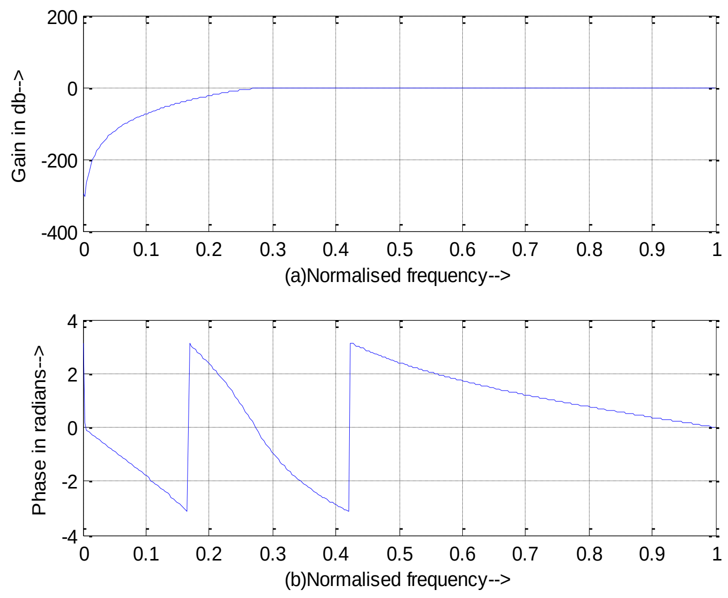
```
clear all;
close all;
format long
lp=input('Enter the passband attenuation:');
ls=input('Enter the stopband attenuation:');
wp=input('Enter the passband frequency:');
ws=input('Enter the stopband frequency:');
fs=input('Enter the sampling frequency:');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=buttord(w1,w2,lp,ls); disp(n);disp(wn)
[b,a]=butter(n,wn,'high');
w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure(1);
subplot(2,1,1);
title('Butterworth Low Pass Filter');
plot(om/pi,m);
grid on;
ylabel('Gain in db-->');
xlabel('(a)Normalised frequency-->');
subplot(2,1,2);
plot(om/pi,an);
xlabel('(b)Normalised frequency-->');
ylabel('Phase in radians-->');
```

grid on

Result:

```
Enter the passband attenuation:     0.5
Enter the stopband attenuation:     50
Enter the passband frequency:       1200
Enter the stopband frequency:       2400
Enter the sampling frequency:       10000
```

Output Wave form



3.Butter worth BPF

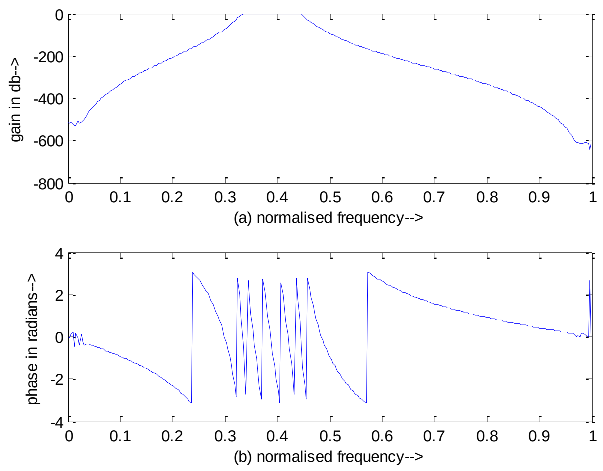
clc;

```
close all;
clear all;
format long
rp=input('enter the passband ripple');
rs=input('enter the stopband ripple');
wp=input('enter the passband frequency');
ws=input('enter the stopband frequency');
fs=input('enter the sampling frequency');
w1=2*wp/fs;
w2=2*ws/fs;
[n]=buttord(w1,w2,rp,rs);
wn=[w1,w2];
[b,a]=butter(n,wn);
w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
subplot(2,1,1);plot(om/pi,m);
ylabel('gain in db-->');
xlabel('(a) normalised frequency-->');
subplot(2,1,2);plot(om/pi,an);
xlabel('(b) normalised frequency-->');
ylabel('phase in radians-->');
```

Result:

```
enter the passband ripple       0.3
enter the stopband ripple          40
enter the passband frequency              1500
enter the stopband frequency              2000
enter the sampling frequency              9000
```

Output Wave form



4 Butterworth bandstop filter

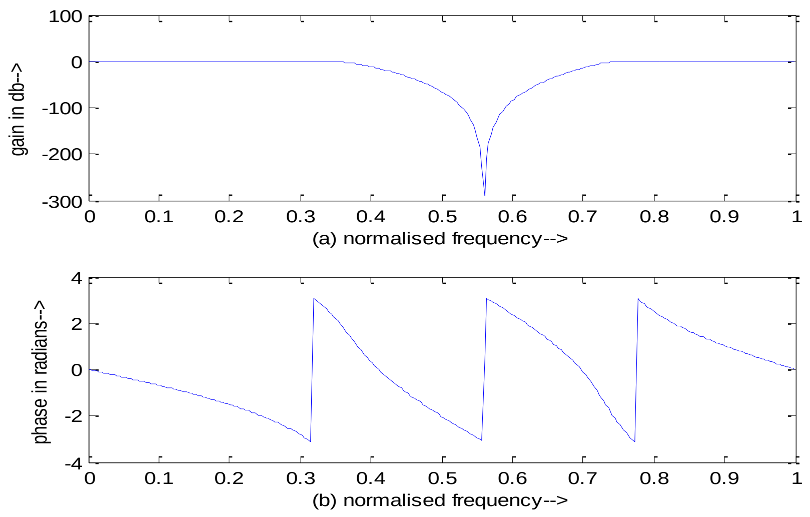
```
clc;
close all;
clear all;
format long
rp=input('enter the passband ripple');
rs=input('enter the stopband ripple');
wp=input('enter the passband frequency');
ws=input('enter the stopband frequency');
fs=input('enter the sampling frequency');
w1=2*wp/fs;
```

```
w2=2*ws/fs;
[n]=buttord(w1,w2,rp,rs); disp(n)
wn=[w1,w2];
[b,a]=butter(n,wn,'stop');
w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
subplot(2,1,1);plot(om/pi,m);
ylabel('gain in db-->');
xlabel('(a) normalised frequency-->');
subplot(2,1,2);plot(om/pi,an);
xlabel('(b) normalised frequency-->');
ylabel('phase in radians-->');
```

Result

```
enter the passband ripple        0.4
enter the stopband ripple        46
enter the passband frequency             1100
enter the stopband frequency             2200
enter the sampling frequency             6000
```

Output Wave form



CHEBYSHEV FILTERS

1 Chebyshev LPF

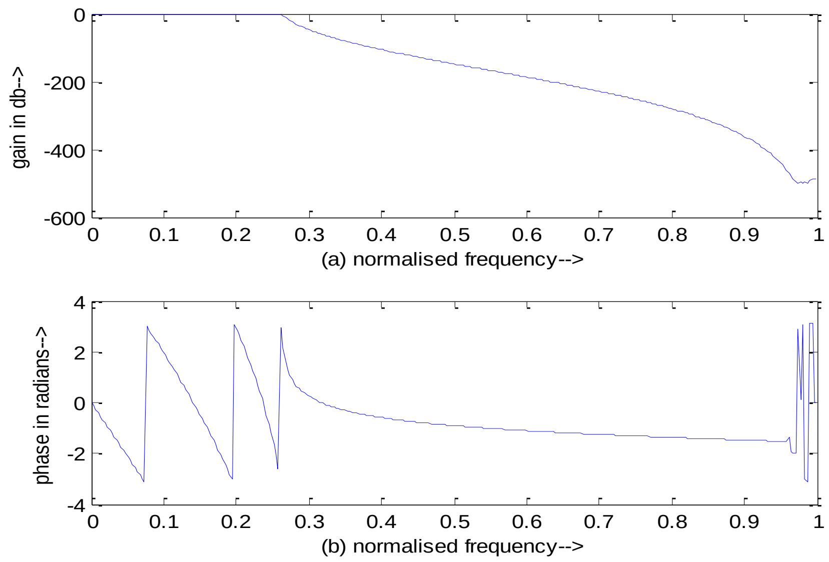
```
clc;
close all;
clear all;
format long
rp=input('enter the passband ripple');
rs=input('enter the stopband ripple');
wp=input('enter the passband frequency');
ws=input('enter the stopband frequency');
fs=input('enter the sampling frequency');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=cheb1ord(w1,w2,rp,rs);
[b,a]=cheby1(n,rp,wn);
w=0:0.01:pi;
[h,om]=freqz(b,a,w);
```

```
m=20*log10(abs(h));
an=angle(h);
subplot(2,1,1);plot(om/pi,m);
ylabel('gain in db-->');
xlabel('(a) normalised frequency-->');
subplot(2,1,2);plot(om/pi,an);
xlabel('(b) normalised frequency-->');
ylabel('phase in radians-->');
```

Result:

```
enter the passband ripple          0.2
enter the stopband ripple          45
enter the passband frequency              1300
enter the stopband frequency              1500
enter the sampling frequency              10000
```

Output Waveform



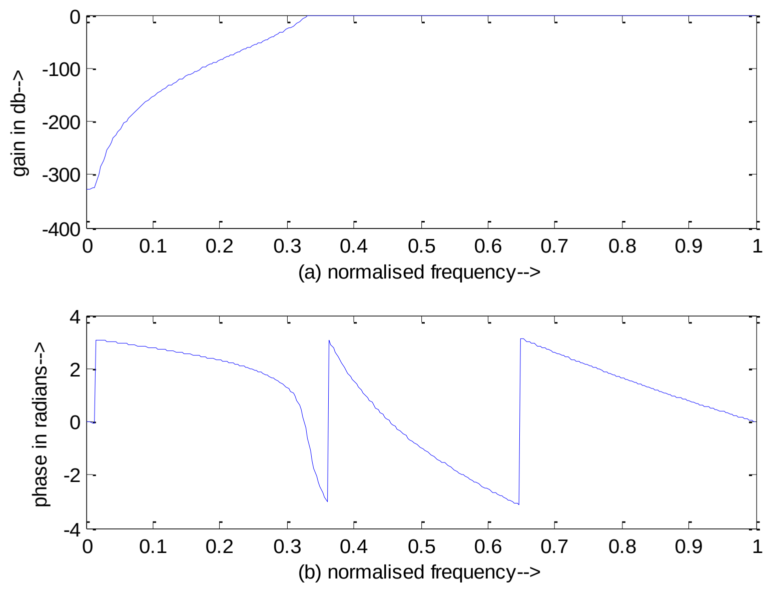
2 Chebyshev HPF

```
clc;
close all;
clear all;
rp=input('enter the passband ripple');
rs=input('enter the stopband ripple');
wp=input('enter the passband frequency');
ws=input('enter the stopband frequency');
fs=input('enter the sampling frequency');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=cheb1ord(w1,w2,rp,rs);
[b,a]=cheby1(n,rp,wn,'high');
w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));

an=angle(h);
subplot(2,1,1);plot(om/pi,m);
ylabel('gain in db-->');
xlabel('(a) normalised frequency-->');
subplot(2,1,2);plot(om/pi,an);
xlabel('(b) normalised frequency-->');
ylabel('phase in radians-->');
```

Result:

```
enter the passband ripple        0.3
enter the stopband ripple        60
enter the passband frequency           1500
enter the stopband frequency           2000
enter the sampling frequency           9000
```

Output Waveform



3 Chebyshev BPF

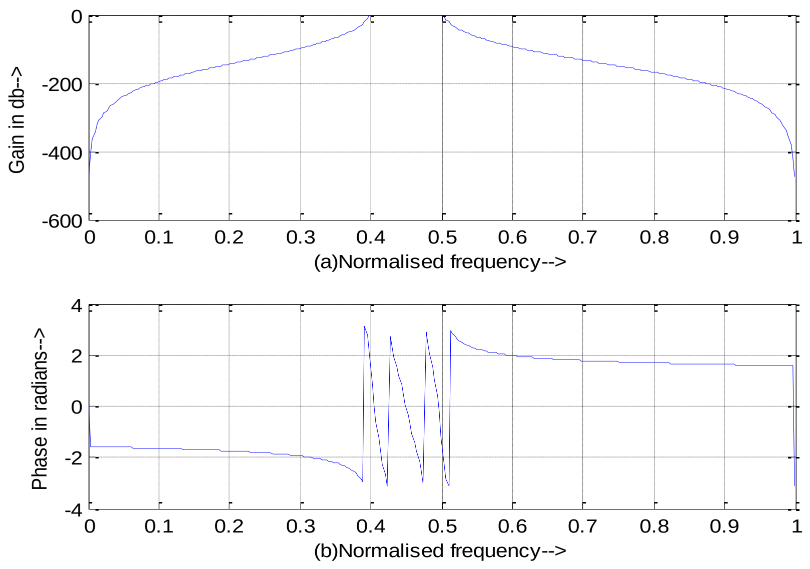
```
clc;
clear all;
close all;
lp=input('Enter the passband attenuation:');
ls=input('Enter the stopband attenuation:');
wp=input('Enter the passband frequency:');
ws=input('Enter the stopband frequency:');
fs=input('Enter the sampling frequency:');
```

```
w1=2*wp/fs;
w2=2*ws/fs;
[n]=cheb1ord(w1,w2,lp,ls);
wn=[w1 w2]
[b,a]=cheby1(n,lp,wn);
w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure(1);
subplot(2,1,1);
title('Butterworth Low Pass Filter');
plot(om/pi,m);
grid on;
ylabel('Gain in db-->');
xlabel('(a)Normalised frequency-->');
subplot(2,1,2);
plot(om/pi,an);
xlabel('(b)Normalised frequency-->');
ylabel('Phase in radians-->');
grid on
```

Result:

```
Enter the passband attenuation:      0.4
Enter the stopband attenuation:      35
Enter the passband frequency:        2000
Enter the stopband frequency:        2500
Enter the sampling frequency:        10000
wn =
   0.40000000000000   0.50000000000000
```

Output Waveform



4 Chebyshev BSF

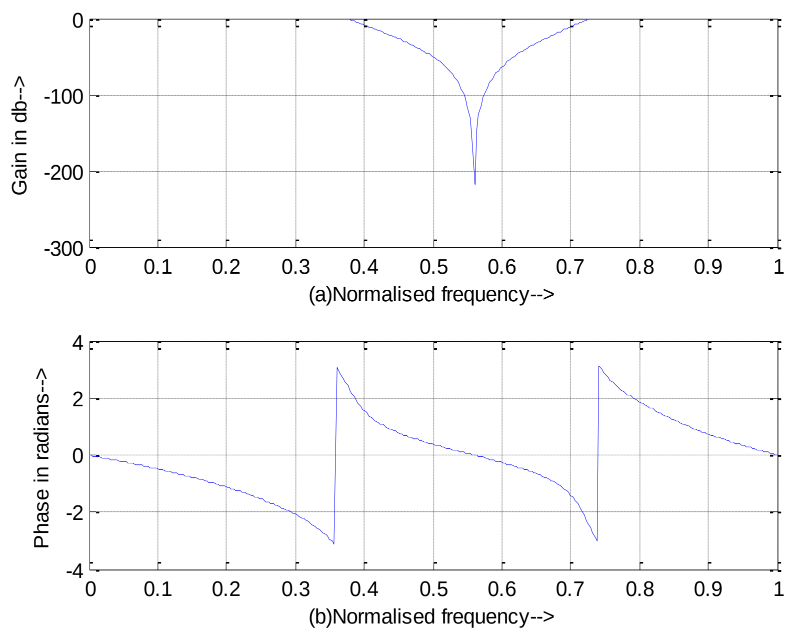
```
clc;
clear all;
close all;
lp=input('Enter the passband attenuation:');
ls=input('Enter the stopband attenuation:');
wp=input('Enter the passband frequency:');
ws=input('Enter the stopband frequency:');
```

```
fs=input('Enter the sampling frequency:');
w1=2*wp/fs;
w2=2*ws/fs;
[n]=cheb1ord(w1,w2,lp,ls);
wn=[w1 w2]
[b,a]=cheby1(n,lp,wn,'stop');
w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure(1);
subplot(2,1,1);
title('Butterworth Low Pass Filter');
plot(om/pi,m);
grid on;
ylabel('Gain in db-->');
xlabel('(a)Normalised frequency-->');
subplot(2,1,2);
plot(om/pi,an);
xlabel('(b)Normalised frequency-->');
ylabel('Phase in radians-->');
grid on
```

Result:

```
Enter the passband attenuation      0.4
Enter the stopband attenuation:     46
Enter the passband frequency:       1100
Enter the stopband frequency:       2200
Enter the sampling frequency:         6000
wn =
   0.36666666666667   0.73333333333333
```

<u>Output Waveform</u>



VIVA Questions:

1. Define an IIR filter.
2. Compare Impulse invariance and bilinear transformations.
3. How do you convert analog filter prototype to a digital filter?
4. What are the functions used in MATLAB for designing a digital Butterworth and Chebyshev low pass filter using BLT?
5. Draw typical responses of Chebyshev filter for order odd & even.
6. Compare FIR & IIR filters.

PART II

EXPERIMENTS USING DSP PROCESSOR

CCS INTRODUCTION

Procedure

1. Double click on CCS icon
2. Create a new project: project → New

   Type the project name (x.pjt) & store in myprojects folder

   To open an existing project: project → open
3. Files to be added to the project: project → Add Files to project
   a. c: \ CCStudio \ c6000 \ cgtools \ lib \ rts6700.lib

      <Select type of file as: library>

   b. c: \ CCStudio \ tutorial \ dsk6713 \ hello1 \ hello.cmd

      < Select type of file as: Linker Command file>
4. File → New → source file → same as xx.c and compile

   (Select type of file as: C/C++ file before saving) & add this C file to your project
5. Project → Build. (obtain Build successful)
6. To execute ; File → load program (select pjtname.out file)
7. To Run : Debug → Run
8. Observe the output on the stdout window or waveforms using graph or CRO
9. To view waveforms View → Graph

changes in graph property dialog box to be made

a. Display type → Dual (to observe 2 waveforms)

b. Title → User defined

- **c.** Start address – upper display → x (user defined variable array names used in C program. Note: x, y should be made global to be used by graph)
- **d.** Start address – lower display → y
- **e.** Acquisition Buffer size → 60 (depends on the array size)
- **f.** Display Buffer size → 60
- **g.** DSP data type – 32-bit Floating pt
- **h.** Auto scale → ON (if off → choose max yvalue=1)

EXPERIMENT NO 9

Linear convolution for the given sequences

Aim: To perform linear convolution for the given sequences

The linear convolution sum is

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

C Program

```
# include<stdio.h>
# include<math.h>
main()
{
//the two sequences can be of different lengths, no need to zero pad for this
program
float h[4] = { 2,2,2,2};
float x[4] ={1,2,3,4};
float y[10];
int xlen=4;
 int hlen=4;
int N=xlen+hlen-1;
int k,n;
for(n=0;n<N;n++)          //outer loop for y[n] array
{ y[n]=0;
for(k=0;k<hlen;k++)                 //inner loop for computing each y[n] point
{  if (((n-k)>=0) & ((n-k)<xlen))
   y[n]=y[n]+h[k]*x[n-k]; //compute output
 }                        //end of inner for loop
  printf("%f\t", y[n]);
}           //end of outer for loop
}           //end of main
```

Exercise: Find the linear convolution of x[n]=[2,1,2,1] and h[n]=[1,2,3,4]

EXPERIMENT NO 10

Circular convolution of two given sequences

<u>Aim</u>: To perform circular convolution of two given sequences

The circular convolution sum is $y[n]=x[n](N)h[n]=\sum_{k=-\infty}^{+\infty} h[k]\,x\left[\langle n-k\rangle_N\right]$

<u>C Program</u>

```
# include<stdio.h>
# include<math.h>
main()
{ //input the two sequences, if of uneven lengths zero pad the smaller one
such that both are of same size
 float x[5]={1,2,3,4,5};
float h[5]={2,1,3,4,5};
float y[10];            //output sequence
int N=5;  // N=max of xlen and hlen//
int k,n,i;
for(n=0;n<N;n++)           //outer loop for y[n] array
        {  y[n]=0;
           for(k=0;k<N;k++)                 //inner loop for computing each y[n]
point
           { i=(n-k)%N;               //compute the index modulo N
                      if(i<0)            //if index is <0, say x[-1], then convert to
        x[N-1]
               i=i+N;
              y[n]=y[n]+h[k]*x[i];  //compute output
           }                            //end of inner for loop
        printf("%f\t",y[n]);
}               //end of outer for loop
}               //end of main
```

<u>Open ended Experiments :</u>

Find the circular convolution of x[n]=[2 3 1 1] and h[n]=[1 3 5 3]

EXPERIMENT NO 11

Computation of N- Point DFT of a given sequence

Aim: To compute the N (=4/8/16) point DFT of the given sequence

Theory:

The N point DFT of discrete time signal x[n] is given by the equation

$$X(k)=\sum_{n=0}^{N-1} x[n]e^{\frac{-j2\pi kn}{N}}; \quad k=0,1,2,\ldots N\text{-}1$$

Where N is chosen such that $N\geq L$, where L=length of x[n]. To implement using C program we use the expression $e^{\frac{-j2\pi kn}{N}} = \cos\left(\frac{2\pi kn}{N}\right) - j\sin\left(\frac{2\pi kn}{N}\right)$ and allot memory space for real and imaginary parts of the DFT X(k)

C Program

```
#include <stdio.h>
#include <math.h>
main()
{float y[16]; //for 8 point DFT to store real & imaginary
float x[4]={1,3,2,5};  //input only real sequence
float w;
int n,k,k1,N=8,xlen=4;
for(k=0;k<2*N;k=k+2)
{y[k]=0; y[k+1]=0;          //initialize real & imag parts
        k1=k/2; //actual k index
        for(n=0;n<xlen;n++)
        {w=-2*3.14*k1*n/N;       //careful about minus sign
          y[k]=y[k]+x[n]*cos(w);
          y[k+1]=y[k+1]+x[n]*sin(w);
        }
        printf("%f+j%f \n",y[k],y[k+1]);
}
}//end of main
```

MATLAB verification

```
>> x=[1 3 2 5];
>> fft(x,8)     11.0000           -0.4142 - 7.6569i       -1.0000 + 2.0000i
2.4142 - 3.6569i  -5.0000     2.4142 + 3.6569i         -1.0000 - 2.0000i   -
0.4142 + 7.6569i
```

Open ended Experiments : Find the 8 point DFT of x[n]=[0.5,0,0.5,0]

EXPERIMENT 12

Impulse response of the given system

Aim: **To find the Impulse response of the given first order / second order system**

Theory:

A linear constant coefficient difference equation representing a second order system is given by

$$y[n]+a_1 y[n-1]+a_2 y[n-2]=b_0 x[n]+b_1 x[n-1]+b_2 x[n-2];$$

For a first order system the coefficients a_2 and b_2 are zero.

Since the difference equation contains past samples of output, i.e., y[n-1], y[n-2], its impulse response is of infinite duration (IIR). For such systems the impulse response is computed for a large value of N, say N=100 (to approximate n=∞).

Impulse response implies x[n]= **δ[n],** an impulse, and the initial conditions x[-1], x[-2], y[-1] and y[-2] are zero.

(refer expt 2 &7of part A - h=impz(b,a,N); %impulse response verification in MATLAB)

To implement in C the following algorithm is used.

Algorithm

1. Input the coefficients b_0 b_1 b_2, a_1 a_2 (Say the coefficients from a 2nd order butterworth filter. Note coefficient of y[n] is a_0=1 always)
2. Generate the impulse input of size n (1st element=1, rest all=0)
3. Compute the output as
   $$y[n]=-a_1 y[n-1]-a_2 y[n-2]+b_0 x[n]+b_1 x[n-1]+b_2 x[n-2];$$
4. Repeat step 3 for n=0 through N (say 50,100, etc)

C Program

```
#include <stdio.h>
float x[60],y[60];
main()
{float a1,a2,b0,b1,b2;
int i,j,N=20;
 a1= -1.1430; a2= 0.4128;
 b0=0.0675;b1=0.1349;b2=0.0675;
//generate impulse input
x[0]=1;
for(i=1;i<N;i++)
x[i]=0;
//generate output
for(j=0;j<N;j++)
{y[j]=b0*x[j];
if(j>0)
y[j]=y[j]+b1*x[j-1]-a1*y[j-1];
if ((j-1)>0)
y[j]=y[j]+b2*x[j-2]-a2*y[j-2];
printf("%f \t",y[j]);
}
}//end of main
```

Note: For a first order system just enter a2=0 and b2=0.

Open ended Experiments :

Find the impulse response of the given difference equation

a. **y[n]-1/9y[n-2]=x[n-1]**
b. **y[n]-3/4y[n-1]+1/8y[n-2]=x[n]+1/2x[n-1]**

EXPERIMENT NO 13

Generation of sinewave and standard test signals

Aim: G**eneration of sinewave, square wave and triangular wave.**

Theory:

In this program we definecreate 200 samples for sinewave and use the ‘sin’ function to generate sine values of 200 points.

C Program to generate sine wave

```
#include<stdio.h>
#include<math.h>
#define pi  3.1415625
float a[200];
main()
{
int  i=0;
for(i=0;i<200;i++)
a[i]=sin(2*pi*5*i/200);
}
```

Result:

The output plot may be observed on the CRO or the figure window.

Open Ended Experiment

In this open end experiments students are expected to write programs to create wave forms such as square wave, triangular wave and sawtooth wave. Students may write separate programs for each waveforms.

Mini Project

Students are expected to create a GUI based signal generator using MATLAB. The signal generator will have three buttons to select three signals ie., sine wave, square wave and triangular wave. Upon the selection the wave form in time domain and frequency domain needs to be displayed graphically. There should be a control to change the amplitude and frequency of the signals.

Questions:

1. Why do we need DSP processors?
2. What is the difference between Fixed and floating point processors?
3. Explain the architectural features of DSP processor.
4. How do perform signed arithmetic on DSP processor?
5. What are the advantages and disadvantages of DSP processor?