**DEPARTMENT
OF
ELECTRONICS & COMMUNICATION ENGINEERING
DSP LAB
V Semester (18EC5DLDSP)
Autonomous Course**

Dr. Manjunath T C,   Dr. S. Thenmozhi,   Prof. Deepa N.P,   Prof. Shahla Sohail,
Prof. Ramgopal Segu,   Prof. C Usha,   Prof. Kavita Guddad,   Prof. Kumar P
Prof. Chaitra A

Name of the Student	:	
Semester /Section	:	
USN	:	
Batch	:	

Dayananda Sagar College of Engineering

Dayananda Sagar College of Engineering
Dept. of E & C Engg

Name of the Laboratory	:	DSP LAB
Semester/Year	:	V/ 2020 (Autonomous)
No. of Students/Batch	:	20-24
No. of Computers	:	15-23
Major Equipment's	:	Dell Computers

DSP TMS320 Kit
Digital Oscilloscope
Signal Generator
Power Supply +5v, + 12v, + 30v

Operating System &	:	**Windows 8.1**
Application		**Matlab 2014, CCStudio V3.1**
Area in square meters	:	68 Sq Mts
Location	:	Level – 1
Total Cost of Lab	:	Rs.9,17,686/-
Lab In charges	:	Dr. S. Thenmozhi

Dr. C. Usha
Prof. Kavita Guddad
Prof. Kumar P

Instructor	:	Mrs. H Gayatri, & Mrs. Veena H.S
HOD	:	Dr. T.C. Manjunath, Ph.D. (IIT Bombay)

About the College & the Department

The Dayananda Sagar College of Engineering was established in 1979, was founded by Sri R. Dayananda Sagar and is run by the Mahatma Gandhi Vidya Peetha Trust (MGVP). The college offers undergraduate, post-graduates and doctoral programs under Visvesvaraya Technological University & is currently autonomous institution. MGVP Trust is an educational trust and was promoted by Late. Shri. R. Dayananda Sagar in 1960. The Trust manages 28 educational institutions in the name of "Dayananda Sagar Institutions" (DSI) and multi – Specialty hospitals in the name of Sagar Hospitals - Bangalore, India. Dayananda Sagar College of Engineering is approved by All India Council for Technical Education (AICTE), Govt. of India and affiliated to Visvesvaraya Technological University. It has widest choice of engineering branches having 16 Under Graduate courses & 17 Post Graduate courses. In addition, it has 21 Research Centres in different branches of Engineering catering to research scholars for obtaining Ph.D under VTU. Various courses are accredited by NBA & the college has a NAAC with ISO certification. One of the vibrant & oldest dept is the ECE dept. & is the biggest in the DSI group with 70 staffs & 1200+ students with 10 Ph.D.'s & 30$^+$ staffs pursuing their research in various universities. At present, the department runs a UG course (BE) with an intake of 240 & 2 PG courses (M.Tech.), viz., VLSI Design Embedded Systems & Digital Electronics & Communications with an intake of 18 students each. The department has got an excellent infrastructure of 10 sophisticated labs & dozen class room, R & D centre, etc…

DAYANANDA SAGAR COLLEGE OF ENGINEERING
(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)
DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING,
BENGALURU-560078

Vision of the Institute

To impart quality technical education with a focus on research and innovation emphasizing on development of sustainable and inclusive technology for the benefit of society.

Mission of the Institute

- To provide an environment that enhances creativity and Innovation in pursuit of Excellence.

- To nurture teamwork in order to transform individuals as responsible leaders and entrepreneurs.

- To train the students to the changing technical scenario and make them to understand the importance of sustainable and inclusive technologies.

Vision of the Department

To achieve continuous improvement in quality technical education for global competence with focus on industry, social needs, research and professional success.

Mission of the Department

- Offering quality education in Electronics and communication engineering with effective teaching learning process in multidisciplinary environment.
- Training the students to take up projects in emerging technologies and work with team spirit.
- To imbibe professional ethics, development of skills and research culture for better placement opportunities.

Program Educational Objectives [PEOs]

PEO1: Ready to apply the state-of-art technology in industry and meeting the social needs with knowledge of Electronics and communication Engineering due to strong academic culture.

PEO2: Competent in technical and soft skills to be employed with capacity of working in multidisciplinary domains

PEO3: Professional, capable of pursuing higher studies in technical, research or management programs

DSP LAB

Course Code : 18EC5DLDSP   **Credits :** 02
   L : P : T : S : 0 : 2 : 1 : 0   **CIE Marks :** 50
   Exam Hours : 03   **SEE Marks :** 50
   Total Hours : 26   **CIE + SEE Marks :** 100

COURSE OBJECTIVES:

1.	Simulate and verify the results of various signal processing concepts by using build in MATLAB functions.
2.	Practice to implement the required signal processing concepts by means of user defined MATLAB function.
3.	Compare the performance characteristics of IIR and FIR filter through MATLAB simulation.
4.	Familiarize with Simulink environment by constructing models for simple LTI systems.
5.	Realize the concepts of convolution, DFT and system responses using Digital Signal Processors (hardware interface).
6.	Verify digital filtering operation in real time

COURSE OUTCOMES: At the end of the course, the student will be able to

CO1	Simulate, verify and analyze the spectra of Sampled signal and DFT.
CO2	Analyze through simulation the response of an LTI system by using various approaches - Convolution and difference equation.
CO3	Design, simulate IIR & FIR filters and analyze the magnitude and phase spectra.
CO4	Explore Auto correlation and Cross-correlation concepts and their properties through MATLAB simulation.
CO5	Construct the Simulink models for discrete time systems and verify the response.
CO6	Implement DT LTI systems on Digital Signal Processor and verify the response.

Mapping of Course outcomes to Program outcomes:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	3	-	1	3	-	-	-	1	-	-	1
CO2	3	3	1	1	3	-	-	-	1	-	-	1
CO3	3	3	3	1	3	-	-	-	1	-	-	1
CO4	3	3	-	1	3	-	-	-	1	-	-	1
CO5	3	3	2	1	3	-	-	-	1	-	-	1
CO6	3	3	3	1	3	-	-	-	1	-	-	1

Module	Expt No.	Content of the Lab Module	Hours	COs
		Introduction to MATLAB: different operators & functions. Introduction to SIMULINK: different block sets & tools		
Software: Programs				
PART-A	1.	Verification of Sampling theorem.	18	CO 1
	2.	Impulse response of a given system by solving a given difference equation.		CO 2
	3.	Linear & circular convolution of two given sequences.		CO 2
	4.	Autocorrelation and Cross-correlation of a given sequence with its property's verification.		CO 4
	5.	Computation of N point DFT of a given sequence and to plot magnitude and phase spectrum.		CO 1
	6.	Linear and Circular convolution of two sequences using DFT and IDFT.		CO 2
	7.	Design and implementation of FIR filter to meet given specifications.		CO 3
	8.	Design and implementation of IIR filter to meet given specifications.		CO 3
	9.	Construct a Simulink model for a system represented by second order difference equation and verify the system response		CO2, 5
	10.	Construct a Simulink model of a FIR LPF for given specifications and observe the time domain waveform and spectrum of filtered signal.		CO 3, 5
Hardware : Interfacing experiments				
PART-B	11.	Linear & circular convolution of two given sequences.	8	CO 2, 6
	12.	Computation of N- Point DFT of a given sequence		CO 1, 6
	13.	Impulse response of first order and second order system		CO 2, 6
	14.	Realization of an FIR filter (any type) to meet given specifications. The input can be a signal from function generator / speech signal.		CO 3, 6

PRE-REQUISITES:

1. Fundamentals of Signals and Systems.
2. Theoretical knowledge of Digital Signal Processing.
3. Basic programming knowledge.

Scheme of Evaluation of the CIE & Assessment Pattern:

CIE – Continuous Internal Evaluation (**50** Marks)	
Record writing	**10** Marks
Conduction	**10** Marks
Viva-voce	**05** Marks

SEE – Semester End Examination (50 Marks)

Bloom's Category	Performance (Day To Day)	Internal Test
Marks (Out of 50)	**25**	**25**
Remember		
Understand		
Apply	05	05
Analyze	10	10
Evaluate	05	05
Create	05	05

DO's

➢ All the students should come to LAB on time with proper dress code and identity cards
➢ Keep your belongings in the corner of laboratory.
➢ Students have to enter their name, USN, time-in/out and signature in the log register maintained in the laboratory.
➢ All the students should submit their records before the commencement of Laboratory experiments.
➢ Students should come to the lab well prepared for the experiments which are to be performed in that particular session.
➢ Students are asked to do the experiments on their own and should not waste their precious time by talking, roaming and sitting idle in the labs.
➢ Observation book and record book should be complete in all respects and it should be corrected by the staff member.
➢ Before leaving the laboratory students should arrange their chairs and leave in orderly manner after completion of their scheduled time.
➢ Prior permission to be taken, if for some reasons, they cannot attend lab.
➢ Immediately report any sparks/ accidents/ injuries/ any other untoward incident to the faculty /instructor.
➢ In case of an emergency or accident, follow the safety procedure.
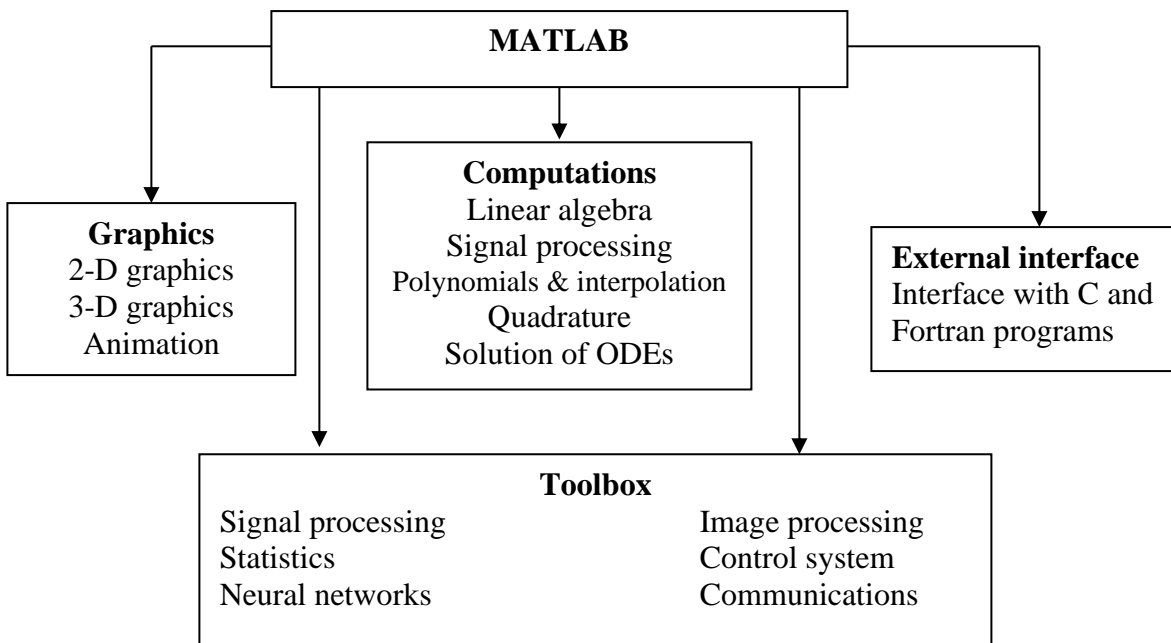➢ Switch OFF the power supply after completion of experiment.

DONT's

➢ The use of mobile/ any other personal electronic gadgets is prohibited in the laboratory.

➢ Do not make noise in the Laboratory & do not sit on experiment table.

➢ Do not make loose connections and avoid overlapping of wires

➢ Don't switch on power supply without prior permission from the concerned staff.

➢ Never point/touch the CRO/Monitor screen with the tip of the open pen/pencil/any other sharp object.

➢ Never leave the experiments while in progress.

➢ Do not insert/use pen drive/any other storage devices into the CPU.

➢ Do not leave the Laboratory without the signature of the concerned staff in observation book

INRODUCTION

MATLAB: MATLAB is a software package for high performance numerical computation and visualization provides an interactive environment with hundreds of built in functions for technical computation, graphics and animation. The MATLAB name stands for MATrix Laboratory

The diagram shows the main features and capabilities of MATLAB.

```
                           ┌─────────────────────┐
                           │      MATLAB         │
                           └─────────────────────┘

  ┌──────────────┐   ┌─────────────────────────┐      ┌──────────────────────┐
  │   Graphics   │   │      Computations       │      │  External interface  │
  │ 2-D graphics │   │    Linear algebra       │      │  Interface with C and│
  │ 3-D graphics │   │   Signal processing     │      │  Fortran programs    │
  │  Animation   │   │ Polynomials & interpolation│    └──────────────────────┘
  └──────────────┘   │      Quadrature         │
                     │    Solution of ODEs     │
                     └─────────────────────────┘

        ┌────────────────────────────────────────────┐
        │                  Toolbox                    │
        │  Signal processing        Image processing  │
        │  Statistics               Control system    │
        │  Neural networks          Communications    │
        └────────────────────────────────────────────┘
```

At its core ,MATLAB is essentially a set (a "toolbox") of routines (called   "m files" or "mex files") that sit on your computer and a window that allows you to create new variables with names (e.g. voltage and time) and process those variables with any of those routines (e.g. plot voltage against time, find the largest voltage, etc).

It also allows you to put a list of your processing requests together in a file and save that combined list with a name so that you can run all of those commands in the same order at some later time. Furthermore, it allows you to run such lists of commands such that you pass in data and/or get data back out (i.e. the list of commands is like a function in most programming languages). Once you save a function, it becomes part of your toolbox (i.e. it now looks to you as if it were part of the basic toolbox that you started with).

For those with computer programming backgrounds: Note that MATLAB runs as an interpretive language (like the old BASIC). That is, it does not need to be compiled. It simply reads through each

line of the function, executes it, and then goes on to the next line. (In practice, a form of compilation occurs when you first run a function, so that it can run faster the next time you run it.)

MATLAB Windows:

MATLAB works with through three basic windows

Command Window **:** This is the main window .it is characterized by MATLAB command prompt >> when you launch the application program MATLAB puts you in this window all commands including those for user-written programs ,are typed in this window at the MATLAB prompt

*Graphics window***:** the output of all graphics commands typed in the command window are flushed to the graphics or figure window, a separate gray window with white background color the user can create as many windows as the system memory will allow

*Edit window***:** This is where you write edit, create and save your own programs in files called M files.

Input-output: MATLAB supports interactive computation taking the input from the screen and flushing, the output to the screen. In addition it can read input files and write output files

*Data Type***:** the fundamental data –type in MATLAB is the array. It encompasses several distinct data objects- integers, real numbers, matrices, charcter strings, structures and cells.There is no need to declare variables as real or complex, MATLAB automatically sets the variable to be real.

Dimensioning: Dimensioning is automatic in MATLAB. No dimension statements are required for vectors or arrays .we can find the dimensions of an existing matrix or a vector with the size and length commands.

Basic Instructions in MATLAB

- ➢ MATLab is case sensitive.

- ➢ "%" is used to add comment.

- ➢ Help is provided by typing "help" or if the topic is known then type "help Funtion_name" or "doc function_name".

- ➢ If the exact name of the topic or command that you are looking for is unknown, then

  type "look for keyword" i.e. "look for filter".

- ➢ If statement is terminated by " ; " , then intermediate results are not displayed in the command window otherwise displays the result.

- ➢ Use the Up-arrow or Down-arrow to recall commands without retyping them in

  command window.

- **T = 0: 1:10**

  This instruction indicates a vector T which as initial value 0 and final value 10 with an increment of 1. Therefore T = [0 1 2 3 4 5 6 7 8 9 10]

- **F= 20: 1: 100**   Output   F = [20 21 22 23 24 ……… 100]

- **T= 0:1/pi: 1**   Output   T= [0, 0.3183, 0.6366, 0.9549]

- **Row Vector:**
  a= [1 2 3 4]   Output   a =   1   2   3   4

- **Length of vector**
  length(a)   ans = 4

- **Transpose:**
  c=a'   Output c =   1

  2

  3

  4

- **Column vector:**
  b = [1;2;3]   Output   b =   1

  2

  3

- **Matrix:**
  M=[1 2 3 4; 5 6 7 8]   Output M =   1   2   3   4

  5   6   7   8

- **Size of array**
  size(M)   ans =   2   4

- **Identity Matrix:**
  I= eye(3)   Output   I =   1   0   0
                                                     0   1   0
                                                     0   0   1

    size(I)   ans =   3   3

    I(1:2,1:2)   ans =   1   0
                                     0   1

  Accessing a row or column of matrix
        Second row: I(2,:)   ans =   0   1   0

        Second column: I(:,2)   ans =   0
                                                       1
                                                       0

- **zeros (1, 3)**   Output =   0 0 0

- **zeros (2,4)**   Output =   0 0 0 0
                                             0 0 0 0

- **ones (5,2)**   This instruction creates a vector of five rows and two columns

                             Output =   1 1
                                               1 1
                                               1 1
                                               1 1
                                               1 1

- **Random Matrix or vector**
  R=rand(2,3)   Output =   R =   0.9134   0.0975   0.5469
                                                 0.6324   0.2785   0.9575

    Access a row or column of matrix: R(4) or R(2,2)   ans =   0.2785

➢ **a = [1 2 3]   b = [4 5 6]**
    a.*b = 4 10 18   which is multiplication of individual elements. i.e. [4x1 5x2 6x3]
        if C= [2 2 2]   b.*C results in [8 10 12]

➢ **p = 2 *(-2) + 3^2 -1/4**

    p =   4.7500

> **x = [0 2 4 6];   t = [1 2 3 4] ; plot (t, x);**
>   This instruction will display a figure window which indicates the plot of x versus t
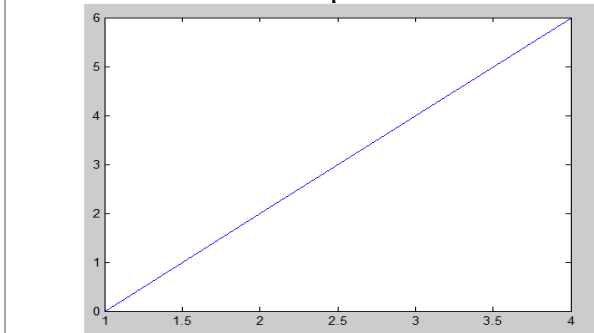
• **stem (t,x)**
  x = [0 2 4 6];   t = [1 2 3 4] ; stem (t, x);
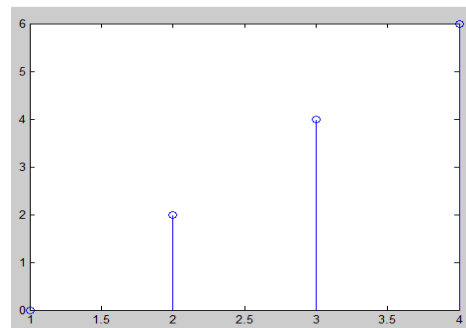
• Subplot: This function divides the figure window into rows and columns

Subplot (2 2 1) divides the figure window into 2 rows and 2 columns 1 represent number of the figure	
1 **(2 2 1)**	**2** **(2 2 2)**
3 **(2 2 3)**	**4** **(2 2 4)**

1 **(3, 1, 1)**
2 **(3, 1, 2)**
3 **(3, 1, 3)**
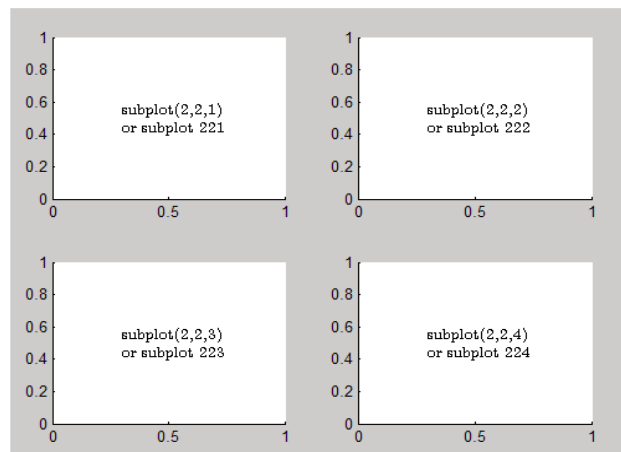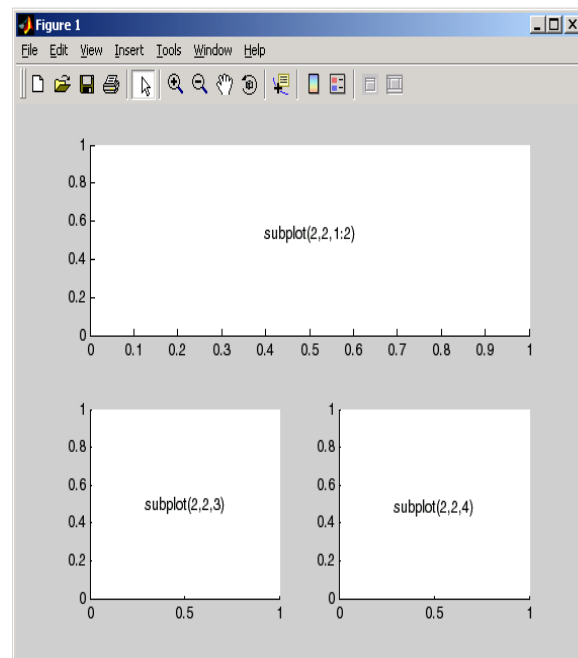
```
x = [0 2 4 6];   t = [1 2 3 4] ;
subplot(2,2,[1 3])
stem (t, x);
subplot(2,2,2)
plot (t, x);
subplot(2,2,4)
stem (t, x);
```
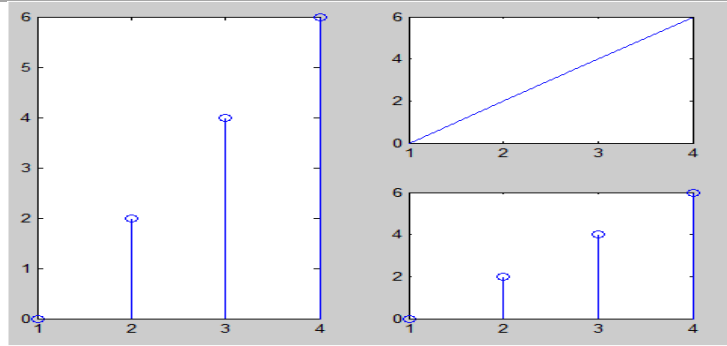
- **Filter**

    Syntax: y = filter(b,a,X)

    Description: y = filter(b,a,X) filters the data in vector X with the filter described by numerator coefficient vector b and denominator coefficient vector a.If a(1) is not equal to 1, filter normalizes the filter coefficients by a(1). If a(1) equals 0, filter returns an error.

- **Impz**

    Syntax: [h,t] = impz(b,a,n)

    Description: [h,t] = impz(b,a,n) computes the impulse response of the filter with numerator coefficients b and denominator coefficients a and computes n samples of the impulse response when n is   an integer (t = [0:n-1]'). If n is a vector of integers, impz computes the impulse response at those integer locations, starting the response computation from 0 (and t = n or t = [0 n]).
    If, instead of n, you include the empty vector [] for the second argument, the number of samples is computed automatically by default.

- **Fliplr**

    Syntax: B = fliplr(A)

    Description: B = fliplr(A) returns A with columns flipped in the left-right direction, that is, about a vertical axis. If A is a row vector, then fliplr(A) returns a vector of the same length with the order of its elements reversed. If A is a column vector, then fliplr(A) simply returns A.

- **Flipud**

    Syntax: B = flipud(A)

    Description: Flip matrix in up/down direction. flipud(A) returns A with columns preserved and rows flipped in the up/down direction.

- **Conv**

  Syntax: w = conv(u,v)

  Description: w = conv(u,v) convolves vectors u and v. Algebraically, convolution is the same operation as multiplying the polynomials whose coefficients are the elements of u and v.

Example:

x=[1 2 3 4];
 	
h=[1 1 1 1];
 	
y=conv(x,h)
 	
y = 1 3 6 10 9 7 4
	A=[1 2 3 4];
	
	B=fliplr(A)
	
	B = 4 3 2 1
		A= [1;2;3;4]
		A = 1
		     2
		    3
		    4
			B=flipud(A)
			B = 4
			    3
			    2
			    1

- **Disp**

  Syntax: disp(X)

  Description: disp(X) displays an array, without printing the array name. If X contains a text string, the string is displayed. Another way to display an array on the screen is to type its name, but this prints a leading "X=," which is not always desirable. Note that disp does not display empty arrays.

- **xlabel**

  Syntax: xlabel('string')

  Description: xlabel('string') labels the x-axis of the current axes.

- **ylabel**

  Syntax : ylabel('string')

  Description: ylabel('string') labels the y-axis of the current axes.

- **Title**

  Syntax : title('string')

  Description: title('string') outputs the string at the top and in the center of the current axes.

- **grid on**

  Syntax : grid on

  Description: grid on adds major grid lines to the current axes.

Program to generate sine ware

```
clc;    clear all;  close all;
t = [0:0.001:0.1];
f = input('Enter the input frequency: ');
x1=sin(2*pi*f*t);
figure(1);
plot(t,x1,'b');    xlabel('time'); ylabel('amplitude');
title(' First signal ');
grid on;
t = [0:0.0001:0.2];
x2=sin(2*pi*f*t);
figure(2);
plot(t,x2,'r*');    xlabel('time'); ylabel('amplitude');
title('Second signal');
```

Basic Signals

```
clc; clear all; close all;
N = 10;
n = -N:1:N;
% unit Impulse
x1=[zeros(1,N)  1   zeros(1,N)];
subplot(3,2,1); stem(n,x1); title(' Impulse  Signal');
%unit Step signal
x2=[zeros(1,N)    1    ones(1,N)];
subplot(3,2,2); stem(n,x2); title('Unit step signal');
%Unit Ramp signal
a1= 2;
x3=a1*n;
subplot(3,2,3); stem(n,x3); title('Unit ramp signal');
%Exponential growing/decaying signal
n2=0:0.1:N;
a2=2;
x4=a2.^n2;
subplot(3,2,4); stem(n2,x4); title( 'Exponential growing signal');
a3=0.5;
x5=a3.^n2;
subplot(3,2,5); stem(n2,x5); title( 'Exponential decaying signal');
%cosine siganl
x6=cos(n2);
subplot(3,2,6); stem(n2,x6);  title('Cosine signal');
```

Zplane

```
%For data sampled at 1000 Hz, plot the poles and zeros
%of a 4th-order elliptic lowpass digital filter
%with cutoff frequency of 200 Hz, 3 dB of ripple in the passband,
%and 30 dB of attenuation in the stopband:
clc;
[z,p,k] = ellip(4,3,30,200/500);
zplane(z,p);
title('4th-Order Elliptic Lowpass Digital Filter');
```

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 $\hat{x}(t)$) 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 $\hat{x}(t)$ 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 fd (for which sampling theorem is to be verified).
2.   Generate an analog signal xt of frequency fd for comparison.
3.   Generate oversampled, 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.(tfinal 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*fd*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*fd*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 undersampling and Nyquist sampling conditions.

MATLAB Program:

```
tfinal=0.02;
t=0:0.00002:tfinal;
fd=input('Enter analog freuency ');
%define analog signal for comparison
xt=cos(2*pi*fd*t);
%simulate condition for undersampling i.e., fs1<2*fd
fs1=1.5*fd;
%define the time vector
n1=0:1/fs1:tfinal;
%Generate the undersampled signal
xn1=cos(2*pi*n1*fd);
%plot the analog & sampled signals
subplot(3,2,1); plot(t,xt,'b',n1,xn1,'r*-'); title('undersampled plot');
%spectrum of discrete signal
Xk1=fft(xn1);
%frequency index of spectrum plot
f1=(0:length(Xk1) -1)*fs1/length(Xk1);
subplot(3,2,2); stem(f1,abs(Xk1)); title('spectrum of undersampled plot');
%condition for Nyquist plot
fs2=2*fd;
n2=0:1/fs2:tfinal;
xn2=cos(2*pi*fd*n2);
subplot(3,2,3);   plot(t,xt,'b',n2,xn2,'r*-');   title('Nyquist plot');
Xk2=fft(xn2);
f2=(0:length(Xk2)-1)*fs2/length(Xk2);
subplot(3,2,4);   stem(f2,abs(Xk2));   title('spectrum of rightsampled plot');

%condition for oversampling
```

fs3=10*fd;
n3=0:1/fs3:tfinal;
xn3=cos(2*pi*fd*n3);
subplot(3,2,5); plot(t,xt,'b',n3,xn3,'r*-'); title('Oversampled plot');
xlabel('time'); ylabel('amplitude');
Xk3=fft(xn3);
f3=(0:length(Xk3) -1)*fs3/length(Xk3);
subplot(3,2,6); stem(f3,abs(Xk3)); title('spectrum of oversampled plot');

Note: For Sine waveform, start sampling n2 from (0.005/4). i.e. n2=0.00125:1/fs2:tfinal;

   For Cos n2=0:1/fs2:tfinal;

Result: Enter analog frequency 200 , The plots are shown in Fig.
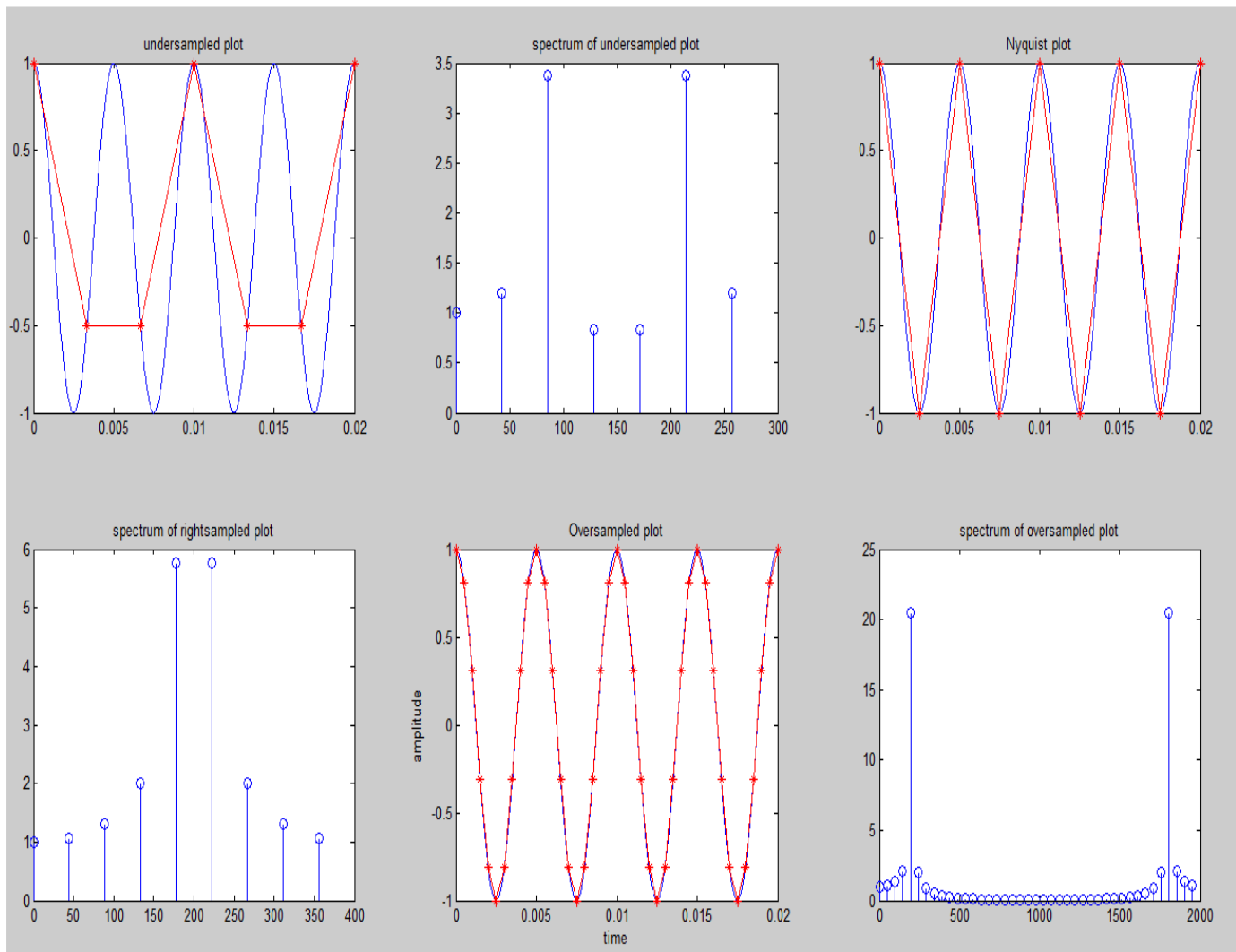
EXPERIMENT NO 2a: Impulse response of a given system

Aim: To find the impulse response h(n) of the given LTI system whose response y(n) to an input x(n) is given.

Theory:
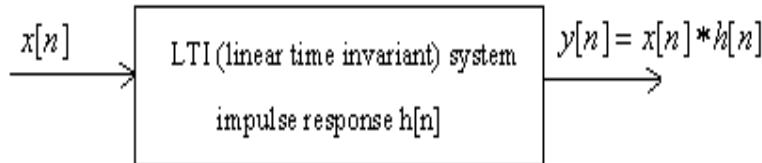
Fig.2.1 A LTI system

A complete **characterization of any LTI system** can be represented in terms of its **response to an unit impulse**, which is referred to as **Impulse response** of the system. Alternatively, the impulse response is the output of a LTI system due to an impulse input applied at t=0, or n=0.

- A discrete time LTI system (also called digital filters) as shown in Fig.2.1 is represented by
  - A linear constant coefficient difference equation, for example, $y[n]+a_1 y[n-1]-a_2 y[n-2]=b_0 x[n]+b_1 x[n-1]+b_2 x[n-2]$;
  - A system function H(z) (obtained by applying Z transform to the difference equation).
    $$H(z)=\frac{Y(z)}{X(z)}=\frac{b_0+b_1 z^{-1}+b_2 z^{-2}}{1+a_1 z^{-1}+a_2 z^{-2}}$$
- Given the difference equation or H(z), the impulse response of the LTI system is found using **filter or impz** MATLAB functions.

PROBLEM:
Obtain the Impulse response of a system described by the difference equation-
      y(n)-3/4 y(n-1)+1/8 y(n-2) = x(n)-1/3 x(n-1)

Solution:

     Y(z) - 3/4 Y(z)z⁻¹ + 1/8 Y(z)z⁻² = X(z) - 1/3 X(z) z⁻¹
     H(z) = Y(z) / X (z) = (1-1/3 z⁻¹) / (1-3/4 z⁻¹+1/8 z⁻²)
          Applying partial fractions,
     H (z) = (1-1/3 z⁻¹) / (1-1/2 z⁻¹) (1-1/4 z⁻¹) = A / (1-1/2 z⁻¹) + B / (1-1/4 z⁻¹)
       A=2/3;
       B=1/3;
     H(z) = (2/3) / (1-1/2 z⁻¹) + (1/3) / (1-1/4 z⁻¹)

     h(n) = (2/3) (1/2)ⁿ u(n) + (1/3) (1/4)ⁿ u(n)

h(0)=1	**h(4)=0.0429**	**h(8)=0.0026**
h(1)=0.416	**h(5)=0.02116**	**h(9)=0.0013**
h(2)=0.187	**h(6)=0.0105**	
h(3)=0.088	**h(7)=0.0052**	

MATLAB Program:

```
clc; clear all; close all;
N=input('Enter the length of impulse response =');
b=input('Enter the numerator coeff = ');
a=input('Enter the denominator coeff = ');
[r,p,k]=residuez(b,a)
[h,t]=impz(b,a,N)
figure(1);
stem(t,h); grid;
xlabel('Time index'); ylabel('Amplitude'); title('Impulse Response');
figure(2);
zplane(b,a);
```
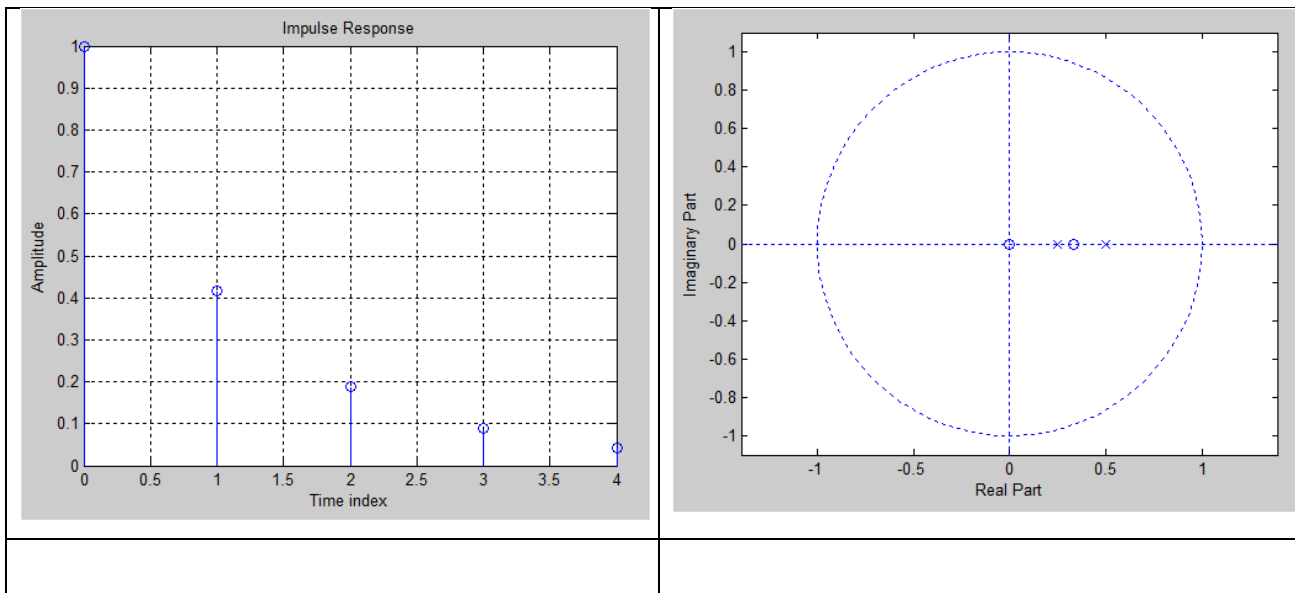
Result:

Enter the length of impulse response =5
Enter the numerator coeff = [1 -1/3]
Enter the denominator coeff = [1 -3/4 1/8]

OUTPUT:

r = 0.6667   0.3333
p = 0.5000   0.2500
k = [ ]
h = 1.0000   0.4167   0.1875   0.0885   0.0430

EXPERIMENT NO 2b: Solving a given difference equation

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

Theory:

- 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(ωn)** 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).
    - 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**(b,a,x) is used to compute the impulse response/ step response/ response to any given x[n] where **b & a are the coefficients of x(n) &y(n) respectively** of a difference equation defining a DTLTI system. Note: The filter function evaluates the convolution of an infinite sequence (IIR) and x[n], which is not possible with conv function (remember **conv(x,h) function 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 $x[n] = \{1,0,0,0,....\}$ the number of zeros = the length of the IIR response required (say

100 implemented by function zeros(1,99)).

For step response $x[n] = \{1,1,1,1,1,....\}$ 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:

2b.1   *To find Impulse Response:*

```
N=input('Length of response required=');
b=[1];            %x[n] coefficient
a=[1,-1,0.9];        %y coefficients
x=[1,zeros(1,N-1)];    %impulse input
n=0:1:N-1;          %time vector for plotting
h=filter(b,a,x);      %impulse response
disp(h);
subplot(2,1,1); stem(n,x); title('impulse input ….'); xlabel('n');ylabel('x(n)');
subplot(2,1,2); stem(n,h); title('impulse response'); xlabel('n');ylabel('h(n)');
```

Result:

    Length of response required=10

1.0000	1.0000	0.1000	-0.8000	-0.8900
-0.1700	0.6310	0.7840	0.2161	-0.4895

2b.2   *To find step response:*

```
clc;   close all;   clear all
b=input('Enter the co-efficients of b ie x[n]=');
a=input('Enter the co-efficients of a ie y[n]=');
N=input('Enter the length of response required N=');
n=0:1:N-1;
x=[ones(1,N)];
y=filter(b,a,x)
subplot(211);   stem(n,x);   title('Step input');
subplot(212);   stem(n,y);   title('Step response');
```

Result:

```
Enter the co-efficients of b ie x[n]=[1]
Enter the co-efficients of a ie y[n]=[1 -1 0.9]
Enter the length of response required N=10
```

y = 1.0000   2.0000   2.1000   1.3000   0.4100   0.2400   0.8710   1.6550   1.8711   1.3816

2b.3   *To find steady state response:*

```
clc; close all ; clear all ;
b=input('Enter the co-efficients of b ie x[n]=');
a=input('Enter the co-efficients of a ie y[n]=');
N=input('Enter the length of response required N=');
n=0:N-1;
x=cos(0.05*pi*n);
y=filter(b,a,x)
subplot(211);   stem(n,x);   title('Steady input');
subplot(212);   stem(n,y);   title('Steady State response');
```

Result: Enter the co-efficient of b i.e. x[n]=[1]
                 Enter the co-efficient of a i.e. y[n]=[1 -0.8]
                 Enter the length of response required N=35

y = 1.0000   1.7877   2.3812   2.7960   3.0458   3.1437   3.1028   2.9362   2.6580   2.2828
1.8263   1.3046   0.7346   0.1337   -0.4808 -1.0918 -1.6824 -2.2369 -2.7406   -3.1802 -
3.5441 -3.8230 -4.0095 -4.0986 -4.0879 -3.9774 -3.7697 -3.4698 -3.0848   -2.6243 -
2.0994 -1.5231 -0.9095 -0.2736   0.3689

2b.4 *Solve the difference equation with initial conditions:*

PROBLEM 1:

y(n) = x(n) + (3/2)y(n-1) - (1/2)y(n-2)where x(n)=(1/4)^n u(n), initial conditions are y(-1)=4, y(-2)=10.

SOLUTION:

x(n)=(1/4)^{n}u(n)

     n=0:5
     n=0,   x(0)=1
     n=1,   x(1)=0.25
     n=2,   x(2)=0.0625
     n=3,   x(3)=0.0156
     n=4,   x(4)=0.0039
     n=5,   x(5)=0.0010

y(n) = x(n) + (3/2)y(n-1) - (1/2)y(n-2)

     n=0,   y(0)= x(0) + (3/2)y(0-1)- (1/2)y(0-2)   = 2
     n=1,   y(1)= x(1) + (3/2)y(1-1)- (1/2)y(1-2)   = 1.25
     n=2,   y(2)= x(2) + (3/2)y(2-1)- (1/2)y(2-2)   = 0.9375
     n=3,   y(3)= x(3) + (3/2)y(3-1)- (1/2)y(3-2)   = 0.7969
     n=4,   y(4)= x(4) + (3/2)y(4-1)- (1/2)y(4-2)   = 0.7365
     n=5,   y(5)= x(5) + (3/2)y(5-1)- (1/2)y(5-2)   = 0.6982

Matlab Program:

```
clc;   clear all;  close all;

a=input('Enter the co-efficient of y(n),y(n-1)......= ');

b=input('Enter the co-efficeint of x(n),x(n-1)......= ');

xi=input('Enter the initial conditions x(-1),x(-2)... = ');

zi=filtic(b,a,input('Enter initial conditions y(-1),y(-2)....= '),xi);

n=0:5;          %For 8 output samples using filter func, the i/p should also be of size 8

x=(1/4).^n          % input sequence input

y=filter(b,a,x,zi)   % output response

subplot(2,1,1);   stem(n,x);   title('input……….');          xlabel('n'); ylabel('x(n)');

subplot(2,1,2);   stem(n,y);   title('Res of DE with ICs ');  xlabel('n'); ylabel('y(n)');
```

Result:

    Enter the co-efficient of y(n),y(n-1)......= [1 -3/2 1/2]
    Enter the co-efficient of x(n),x(n-1)......= [1]
    Enter the initial conditions x(-1),x(-2)... = [ ]
    Enter initial conditions y(-1),y(-2)....= [4 10]
        x =   1.0000   0.2500   0.0625   0.0156   0.0039   0.0010
        y =   2.0000   1.2500   0.9375   0.7969   0.7305   0.6982

PROBLEM 2:

Solve the difference equation

$$y(n) = \frac{1}{3}[x(n) + x(n-1) + x(n-2)] + 0.95y(n-1) - 0.9025y(n-2), \quad n \geq 0$$

where $x(n) = \cos(\pi n/3)u(n)$ and

$$y(-1) = -2, \; y(-2) = -3; \quad x(-1) = 1, \; x(-2) = 1$$

First determine the solution analytically and then by using MATLAB.

Matlab Program:

```
clc; clear all;
a=input('Enter the co-efficient of y(n),y(n-1)......= ');
b=input('Enter the co-efficeint of x(n),x(n-1)......= ');
Yic=input('Enter initial conditions Y(-1),Y(-2)....= ');
Xic=input('Enter initial conditions X(-1),X(-2)....= ');
zi=filtic(b,a,Yic,Xic);
n=0:9;
x=cos(pi*n/3);          %input sequence input
y=filter(b,a,x,zi)      %output response
subplot(2,1,1);stem(n,x);title('input');xlabel('n');ylabel('x(n)');
subplot(2,1,2);stem(n,y); title('response of difference equation with initial conditions  ');
xlabel('n');ylabel('y(n)');
```

Result:

Enter the co-efficient of y(n),y(n-1)......= [1 -0.95 0.9025]

Enter the co-efficeint of x(n),x(n-1)......= [(1/3) (1/3) (1/3)]

Enter initial conditions Y(-1),Y(-2)....= [-2 -3]

Enter initial conditions X(-1),X(-2)....= [1 1]

y = 1.8075   4.3555   2.8398   -1.5664   -4.7176   -3.4014   1.3596   5.0281   3.8829   -1.1824

EXPERIMENT NO 3(a): Linear convolution of two given sequences

Aim: To obtain the linear 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.

- 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].

- The length of y[n] = xlength + hlength -1.

Algorithm:

1. Input the two sequences as xn, hn
2. Convolve both to get output yn.
3. Plot the sequences.

MATLAB Program:

```
% A generalazed convolution computing code in matlab
% without using matlab builtin function conv(x,h)
clc;   close all; clear all ;
x=input('Enter x: ') ;
h=input('Enter h: ') ;
m=length(x);
n=length(h);
X=[x,zeros(1,n)];
H=[h,zeros(1,m)];
        for i=1:n+m-1
           Y(i)=0;
               for j=1:m
                   if(i-j+1>0)
                     Y(i)=Y(i)+X(j)*H(i-j+1);
                   end
               end
        end
Y
stem(Y);
ylabel('Y[n]');
xlabel('----->n');
title('Convolution of Two Signals without conv function');
```

Result: Enter x: [1 2 3 4]
Enter h: [1 2 3 4]

Y =   1   4   10   20   25   24   16

Using Inbuilt Command: `x=[1 2 3 4];`   `h=[1 2 3 4];`   `y=conv(x,h)`

Output: y =   1   4   10   20   25   24   16

Alternate Program: **LINEAR CONVOLUTION USING CONVM FUNCTION:**

Note: *save the following statements with the name Convm.m*

`Convm.m`

```
function[y,ny]=convm(x,nx,h,nh)
    nyb=nx(1)+nh(1);
    nye=nx(length(x))+nh(length(h));
    ny=nyb:nye;
    y=conv(x,h);
```
NOTE: Open new window and save the following program

Main Program:

```
close all;  clc
x=input('Enter the first sequence x= ');
nx=input('Enter the time index for the first sequence nx= ');
h=input('Enter the second sequence h= ');
nh=input('Enter the time index for the second sequence nh= ');
[ y,ny]=convm(x,nx,h,nh)
subplot(311);   stem(nx,x);   xlabel('n');       ylabel('x[n]');
subplot(312);   stem(nh,h);   xlabel('n');       ylabel('h[n]');
subplot(313);   stem(ny,y);   xlabel('n');       ylabel('y[n]');
title('linear convolution');
disp('Output Sequence is : ');
```

Result:

Enter the first sequence x= [1 2 3 4]
Enter the time index for the first sequence nx= -1:2
Enter the second sequence h= [1 2 3 4]
Enter the time index for the second sequence nh= -1:2
Output Sequence is :
y =   1   4   10   20   25   24   16
ny = -2 -1   0   1   2   3   4

EXPERIMENT NO 3(b): Circular convolution of two given sequences

Aim: To obtain circular convolution of two finite duration sequences.

Theory:

- 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 – $v_k[n] = x[k] \times h[n-k]$ (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 $y[n] = x[n](N)h[n] = \sum_{k=-\infty}^{+\infty} x[k]h[\langle n-k\rangle_N]$ 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 to the period. If they are finite sequences N is chosen as >= to max (xlength, hlength). Whereas in linear convolution N>= xlength+hlength-1.
- Say $x[n] = \{-2,3,1,1\}$ 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 $x[n] = \{1,1,0,-2,3\}$.
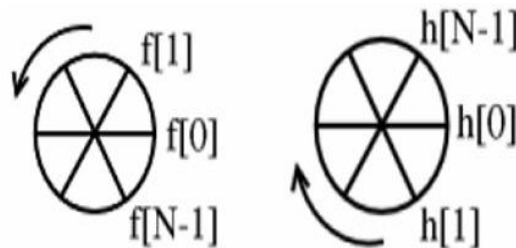

Fig. 4.1 Plotting of f(m) and h(-m) for circular convolution

Algorithm:

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

## *MATLAB Program:*

```
clc; close all; clear all;
x=[1 2 3 4]; h= [1 2 3 4];
N=max(length(x),length(h));
for n=0:N-1  %compute the output
        y(n+1)=0;
              for k=0:N-1
                        i=mod ((n-k), N); %calculation of x    index
                          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('circular convolution of x and h is y=');
disp(y);
n1=0:N-1;  %plot
stem(n1,y);
title('Circular convolution output y(n)');
```

Result:

circular convolution of x and h is y= 26 28 26 20	
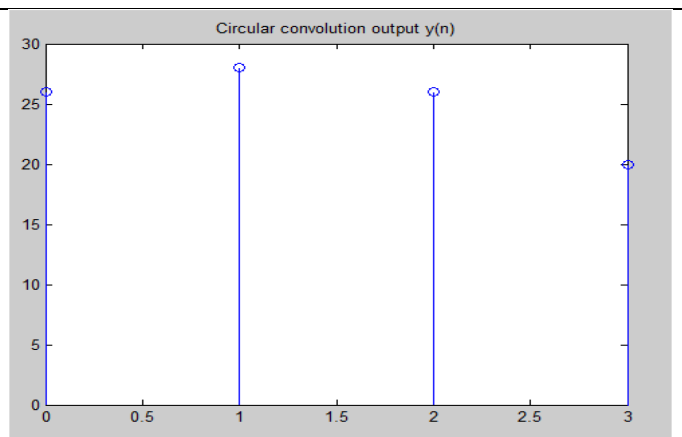
Alternate program for convolution:

Note: To implement circular convolution, take length as maximum Length of the sequences x, h (whichever is highest). For implementing linear convolution take length as x+h-1

```
clc;   clear all; close all;

x=input('Enter the input sequence: x(n)=');

h=input('Enter the impulse sequence: h(n)=');

N=input('Enter the length N=');   %% N=max(length(x),length(h));

%  x= [1 2 3 4];   h=[1 2 3 4];   N= 7 or N=5

x=[x zeros(1,N-length(x))];

h=[h zeros(1,N-length(h))];

    H=zeros(N,N);

            for  n=1:N

                m=0:N-1;

                    H(n,:)=h(mod(n-m-1,N)+1);

            end

    y=x*H';

subplot(311);   stem(x,'filled');

subplot(312);   stem(h,'filled');

subplot(313);   stem(y,'filled');

disp('Resultant: ');

y
```

Result:

Linear Convolution:	*Circular Convolution:*
Enter the input sequence: x(n)=[1 2 3 4]	Enter the input sequence: x(n)=[1 2 3 4 5]
Enter the impulse sequence: h(n)=[1 2 3 4]	Enter the impulse sequence: h(n)=[1 2 3 4]
Enter the length N=7	Enter the length N=5
Resultant:	Resultant:
y = 1 4 10 20 25 24 16	y = 35 35 30 20 30
	Example 2: x(n)=[1 2 3 4]; h(n)= [1 2 3 4];
	y = 26 28 26 20

EXPERIMENT NO 4a: Autocorrelation of a given sequence and Verification of its properties.

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

Theory:

Autocorrelation is a mathematical tool for finding repeating patterns, such as the presence of a periodic signal which has been buried under noise, or identifying the missing fundamental frequency in a signal implied by its harmonic frequencies. It is used frequently in signal processing for analyzing functions or series of values, such as time domain signals. Informally, it is the similarity between observations as a function of the time separation between them. More precisely, it is the cross-correlation of a signal with itself.

- Autocorrelation sequence of x[n] is given by

$$r_{xx}[l] = \sum_{n=-\infty}^{\infty} x[n]x[n-l]; l = 0,\pm1,\pm2,\ldots$$

- Comparing the equations for the linear convolution and auto correlation we find that 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 parameter 'l' called 'lag index' indicates the time shift between the pair
- Some of the properties of autocorrelation are enumerated below
  - o The **autocorrelation** sequence **is** an **even** function i.e., $r_{xx}[l] = r_{xx}[-l]$
  - o **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.,

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

    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.
  - o **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.
  - o For power signals the autocorrelation sequence is given by

$$r_{xx}[l] = \lim_{k \to \infty} \frac{1}{2k+1} \sum_{n=-k}^{k} x[n]x[n-l]; l = 0,\pm1,\pm2,\ldots \text{ and for periodic signals with period N it is}$$

$$r_{xx}[l] = \frac{1}{N} \sum_{n=0}^{N-1} x[n]x[n-l]; l = 0,\pm1,\pm2,\ldots \text{ and this } \mathbf{r_{xx}[l] \text{ is also periodic}} \text{ with N.}$$

Algorithm:

1. Input the sequence as x.
2. Use the conv and fliplr function to get cross correlated output rxx.
3. Plot the output sequence.

PROBLEM:

Write a MatLab program to compute the auto correlation of the given sequence and verify its properties. For x(n)=[1 2 3 6 2 4]

Solution:

1	2	3	6	2	4					
4	8	12	24	8	16					
	2	4	6	12	4	8				
		6	12	18	36	12	24			
			3	6	9	18	6	12		
				2	4	6	12	4	8	
					1	2	3	6	2	4
4	10	22	45	46	70	46	45	22	10	4

MATLAB Program:

```
clc;   clear all;  close all;
x=input ('Enter sequence x(n)=');

Rxx= conv(x,fliplr(x));

% y=xcorr(x,x);
disp('rxx=');
disp(Rxx);
figure(1);
stem(Rxx,'filled');    title('Autocorrelation output');
xlabel('lag index');   ylabel('amplitude');
%verification of properties
Energy=sum(x.^2)
center_index=ceil(length(Rxx)/2)
Rxx_0=Rxx(center_index)
        if Rxx_0==Energy
                disp('Rxx(0) gives energy - proved');
        else
                disp('Rxx(0) gives energy - not proved');
        end
Rxx_right=Rxx(center_index:1:length(Rxx))
Rxx_left=Rxx(center_index:-1:1)
        if Rxx_right == Rxx_left
            disp('Rxx is even');
        else
            disp('Rxx is odd');
        end
```
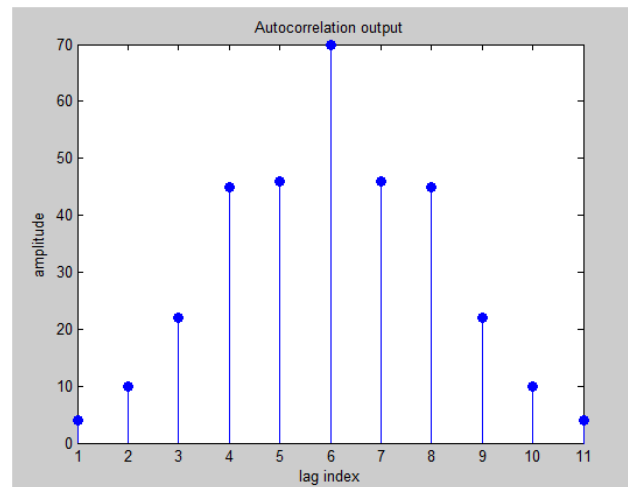

Result:

Enter sequence x(n)=[1 2 3 6 2 4]

rxx=   4   10   22   45   46   70   46   45   22   10   4

Energy =   70

center_index =   6

Rxx_0 =   70

Rxx(0) gives energy - proved

Rxx_right =   70   46   45   22   10   4

Rxx_left =   70   46   45   22   10   4

Rxx is even

Assignment: Implement autocorrelation without using in-built function.

Autocorrelation using Function

Convm.m

```
function[y,ny]=convm(x,nx,h,nh)
        nyb=nx(1)+nh(1);
        nye=nx(length(x))+nh(length(h));
        ny=nyb:nye;
        y=conv(x,h);
```

MAIN PROGRAM:

```
clear all;    close all;    clc;
x=input('Enter the Sequence: x= ');
nx=input('Enter time index for the Sequence: nx= ');
[y,ny]=convm(x,nx,fliplr(x),-fliplr(nx))
subplot(211);    stem(nx,x);    xlabel('n');          ylabel('x[n]');
subplot(212);    stem(ny,y);    xlabel('n');          ylabel('y[n]');
title('Autocorrelation ');
```

Result:

Enter the Sequence: x= [1 2 3 6 2 4]

Enter time index for the Sequence: nx= -1:4

y =   4   10   22   45   46   70   46   45   22   10   4

ny = -5   -4   -3   -2   -1   0   1   2   3   4   5

EXPERIMENT NO. 4b: 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:

In signal processing, cross-correlation is a measure of similarity of two waveforms as a function of a time-lag applied to one of them. This is also known as a sliding dot product or inner-product. It is commonly used to search a long duration signal for a shorter, known feature. It also has applications in pattern recognition and crypt analysis. 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
  $$r_{xx}[l] \leq \sqrt{r_{xx}[O]r_{yy}[O]} = \sqrt{\varepsilon_x \varepsilon_y}$$

  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 two 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 conv and fliplr function to get cross correlated output rxy.
3. Plot the output sequence

PROBLEM:

Compute the cross correlation of the two sequences. (x=[1 2 4 6 5], y=[1 3 2 1 4])

Solution:

|	1	2	4	6	5				
1	4	8	16	24	20				
3		1	2	4	6	5			
2			2	4	8	12	10		
1				3	6	12	18	15	
4					1	2	4	6	5
	4	9	20	35	41	31	32	21	5

MATLAB Program:

```
clc;   clear all;
x=input ('Enter sequence x(n)=');
y=input ('Enter sequence y(n)=');
 rxy1= conv(x,fliplr(y));
%rxy2 =xcorr(x,y)
 disp('rxy=');  disp(rxy1);
figure(1);
stem(rxy1,'filled');  title('cross correlation output');
xlabel('lag index');  ylabel('amplitude');
```

Result:

Enter sequence x(n)=[1 2 4 6 5]

Enter sequence y(n)=[1 3 2 1 4]

rxy=   4   9   20   35   41   31   32   21   5
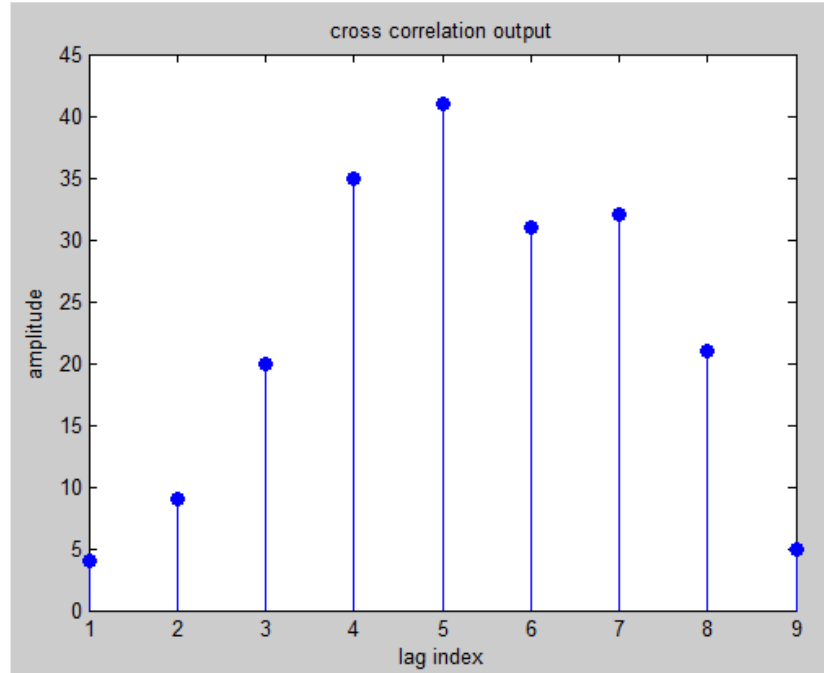
Alternate Program: **CROSS CORRELATION USING CONVM FUNCTION:**

Convm.m

```
function[y,ny]=convm(x,nx,h,nh)
    nyb=nx(1)+nh(1);
    nye=nx(length(x))+nh(length(h));
    ny=nyb:nye;
    y=conv(x,h);
```

MAIN PROGRAM:

```
close all;    clc; clear all;
x=input('Enter the first sequence x= ');
nx=input('Enter the time index for the first sequence nx= ');
h=input('Enter the second sequence h= ');
nh=input('Enter the time index for the second sequence nh= ');

[y,ny]=convm(x,nx,fliplr(h),-fliplr(nh))

subplot(311);    stem(nx,x);        xlabel('na');  ylabel('x[n]');
subplot(312);    stem(nh,h);        xlabel('n');    ylabel('h[n]');
subplot(313);    stem(ny,y);        xlabel('n');    ylabel('y[n]');
title('cross correlation);
```

Result:

```
Enter the first sequence x= [1 2 3 4]
Enter the time index for the first sequence nx= -1:2
Enter the second sequence h= [4 3 2 1]
Enter the time index for the second sequence nh= -1:2
y =    1    4    10    20    25    24    16
ny =  -3   -2   -1    0    1    2    3
```

EXPERIMENT NO 5:   Computation of N point DFT of a given sequence and to plot magnitude and phase spectrum

Aim: To compute DFT of the given sequence.

Theory:

- 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

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

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

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

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

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

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)W_N^{-kn}, \quad 0 \leq n \leq N - 1$$

- 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 Program:

```
x=input('enter the sequence=');   %x[n] sequence
n=input('enter the length=');     %N points for DFT
xk=fft(x,n);                      %computes DFT
disp(xk);
subplot(2,2,1); stem(x,'filled');        title('plot of input sequence');
subplot(2,2,2); stem(real(xk),'filled'); title('plot of real xk');
subplot(2,2,3); stem(abs(xk),'filled');  title('Magnitude spectrum');
subplot(2,2,4); stem(angle(xk),'filled'); title('Phase spectrum');
```

Result:

Enter the sequence= [1 2 1 0]

Enter the length=5

4.0000   0.8090 - 2.4899i   -0.3090 - 0.2245i   -0.3090 + 0.2245i   0.8090 + 2.4899i
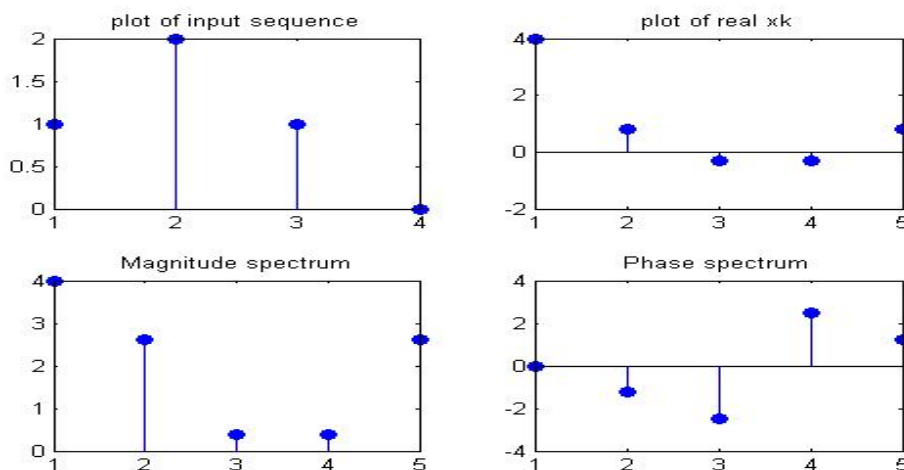
Alternate Program: *Implement N-pt DFT   without using in-built function*

```
% Computation of N point DFT of a given sequence and
% to plot magnitude and phase spectrum.
clc;   close all;  clear all;
N = input('Enter the the value of N for N-Point DFT = ');
xn = input('Enter the sequence for which DFT to be calculated = ');
n=[0:1:N-1];                        % row vector for n
k=[0:1:N-1];                        % row vecor for k
WN=exp(-j*2*pi/N)          % Wn factor
nk=n'*k;                      % creates a N by N matrix of nk values
WNnk=WN.^nk;               % DFT matrix
Xk=xn*WNnk                 % row vector for DFT coefficients
% To verify IDFT
WNnkI= WN.^(-nk);         % IDFT matrix
xnI=(Xk*WNnkI)/N          % row vector for IDFT values
Realval=abs(xnI)
MagX=abs(Xk)                    % Magnitude of calculated DFT
PhaseX=angle(Xk)*180/pi        % Phase of the calculated DFT
subplot(2,1,1); stem(k,MagX);
subplot(2,1,2); stem(k,PhaseX);
```

Result:

Enter the the value of N for N-Point DFT = 5

Enter the sequence for which DFT to be calculated = [1 2 1 0 0]

Xk = 4.0000   0.8090 - 2.4899i   -0.3090 - 0.2245i   -0.3090 + 0.2245i
      0.8090 + 2.4899i

xnI = 1.0000 - 0.0000i   2.0000 - 0.0000i   1.0000 - 0.0000i   0 + 0.0000i
       0 + 0.0000i

Realval =   1.0000   2.0000   1.0000   0.0000   0.0000

MagX =   4.0000   2.6180   0.3820   0.3820   2.6180

PhaseX =   0   -72.0000   -144.0000   144.0000   72.0000

EXPERIMENT NO 6:   Linear and Circular convolution of two given sequences using DFT and IDFT

Aim: To perform linear and circular convolution of two given sequences x1 & x2 using DFT & IDFT.

Theory:

- By multiplying two N-point DFTs in the frequency domain, we get the circular convolution in the time domain.

$$y[n] = x[n] * h[n] \xleftarrow{\quad N-DFT \quad} Y(k) = X(k) \times H(k)$$

- Circular Convolution is made identical to linear convolution by choosing the number of points in DFT as N ≥ xlength + hlength -1. For Circular Convolution, N = max (xlength, length).

Algorithm:

1. Input the two sequences as x1, x2
2. Compute N = xlength + hlength -1

   (for **circular convolution, N=max(xlength + hlength))**
3. Obtain N-point DFTs (X1, X2) of both the sequences.
4. Multiply both the DFTs (Y=X1×X2).
5. Perform IDFT on Y to get y[n] (the linearly convolved sequence) in time domain.
6. Verify using the 'conv' command.
7. Plot the sequences.

MATLAB Program: Linear Convolution Program using DFT and   IDFT:

```
x=input('enter the first sequence=');
h=input('enter the second sequence=');
l=length(x)+length(h)-1;
xk=fft(x,l);
hk=fft(h,l);
y=ifft(xk.*hk);
disp(y);
subplot(3,1,1);   stem(x,'filled');
subplot(3,1,2);   stem(h,'filled');
subplot(3,1,3);   stem(y,'filled');
```

Result:

Enter the first sequence   = [1 1 1 1]

Enter the second sequence = [1 1 1 1]

       1.0000   2.0000   3.0000   4.0000   3.0000   2.0000   1.0000

MATLAB Program: Circular Convolution Program using DFT and IDFT:

```
x=input('enter the first sequence=');
h=input('enter the second sequence=');
l=max(length(x),length(h));
xk=fft(x,l);
hk=fft(h,l);
y=ifft(xk.*hk);
disp(y);
subplot(3,1,1);        stem(x,'filled');
subplot(3,1,2);        stem(h,'filled');
subplot(3,1,3);        stem(y,'filled');
```

Result:

Enter the first sequence   = [2 1 2 1]

Enter the second sequence   = [1 2 3 4]

        14   16   14   16

EXPERIMENT NO 7: Design and implementation of FIR filter to meet given specifications

Aim: 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):

Method I: **Given the order N, cutoff frequency fc, sampling frequency fs and the window.**

- **Step 1: Compute the digital cut-off frequency ω_c** (in the range $-\pi < \omega_c < \pi$, with π corresponding to fs/2) for **fc & fs in Hz**.For Example let fc=400Hz, fs=8000Hz
  $\omega_c = 2*\pi* fc / fs = 2* \pi * 400/8000 = 0.1* \pi$ **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 = ω_c / π
- **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 (ω_p in radians) and Stop band edge (ω_s 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= ω_s – ω_p;

                    N=ceil (6.6*pi/tb);

- **Step 3:** Compute the digital cut-off frequency ω_c as ω_c =(ω_s – ω_p)/2
  Now compute the normalized frequency in the range 0 to 1 for MATLAB as
                    Wc= ω_c /pi;
  Note: In step 2 if frequencies are in Hz, then obtain radian frequencies (for computation
  of tb and N) as ω_p =2*pi*fp/fs, ω_s =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.

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 & B=h[n] coefficients. Appropriately choose N as 128, 256, etc.

Window	Transition width $\Delta\omega$		Min. Stop band	Matlab
Name	**Approximate**	**Exact values**	**Attenuation**	**Command**
Rectangular	$\dfrac{4\pi}{M}$	$\dfrac{1.8\pi}{M}$	**21dB**	**B = FIR1(N,Wn,boxcar)**
Bartlett	$\dfrac{8\pi}{M}$	$\dfrac{6.1\pi}{M}$	**25dB**	**B = FIR1(N,Wn,bartlett)**
Hanning	$\dfrac{8\pi}{M}$	$\dfrac{6.2\pi}{M}$	**44dB**	**B = FIR1(N,Wn,hanning)**
Hamming	$\dfrac{8\pi}{M}$	$\dfrac{6.6\pi}{M}$	**53dB**	**B= FIR1(N,W$_n$,hamming)**
Blackman	$\dfrac{12\pi}{M}$	$\dfrac{11\pi}{M}$	**74dB**	**B = FIR1(N,Wn,blackman)**

➢ *Design and implementation of FIR filter Method 2*

The following program gives only the design of the FIR filter, for implementation continue with the next program (after h[n]).
Input data to be given: **Passband & Stopband frequency**
Data given: Passband ripple & stopband attenuation As. If As>44 dB, Choose hamming

```
clc;   clear all;  close all;
wpa =input('Enter passband edge frequency in Hz = ');
wsa = input('Enter stopband edge frequency in Hz = ');
wsamp  = input('Enter sampling frequency in Hz = ');
%Calculate transmission BW, Transition band tb, order of the filter
wpd=2*pi*wpa/wsamp;
wsd=2*pi*wsa/wsamp;
tb=wsd-wpd;
N=ceil(6.6*pi/tb)
wc=( wsd+ wpd)/2;
%compute the normalized cut off frequency
wc= wc /pi
%calculate & plot the window
hw=hamming(N+1);
figure(1);
stem(hw);
title('Fir filter window sequence- hamming window');
%find h(n) using FIR
h=fir1(N,wc,hamming(N+1))
%plot the frequency response
figure(2);
[m,w]=freqz(h,1,128);
mag=20*log10(abs(m));
plot(wsamp*w/(2*pi),mag);
title('Fir filter frequency response');
grid;
```

Result:

Enter passband edge frequency in Hz = 100
Enter stopband edge frequency in Hz = 200
Enter sampling frequency in Hz = 1000
        N =   33
        wc =   0.3000
h = 0.0002   0.0016   0.0022   0.0006 -0.0037 -0.0074 -0.0048   0.0065   0.0191
     0.0181 -0.0053 -0.0396 -0.0529 -0.0128   0.0852   0.2052   0.2877   0.2877
     0.2052   0.0852 -0.0128 -0.0529 -0.0396 -0.0053   0.0181   0.0191   0.0065
   -0.0048 -0.0074 -0.0037   0.0006   0.0022   0.0016   0.0002
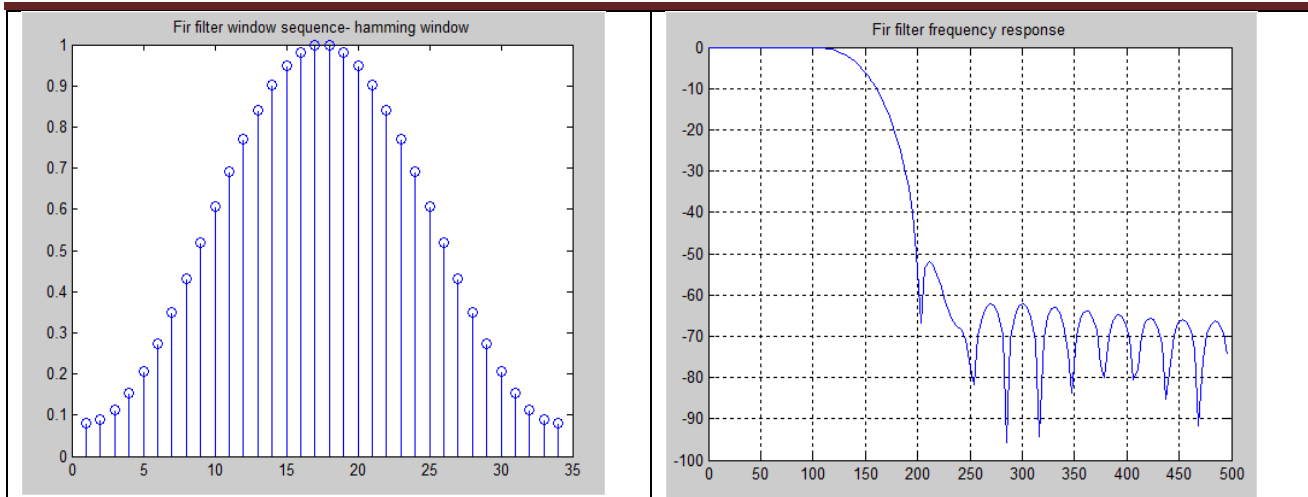
Inference: Notice the maximum stop band attenuation of 53 dB in the right side plot

➢ *Example program with the pass and stop band freq given in rad /sec*

```
clc;    clear all;  close all;
wp=0.2*pi;
ws=0.4*pi;
transition_width = ws-wp;
N= ceil(6.6*pi/transition_width)+1
wc=(ws+wp)/2;
wc=wc/pi
alpha = (N-1)/2;
n = [0:1:(N-1)];
M = n-alpha;
hd = wc*sinc(wc*M);
w_ham = (hamming(N))';
h = hd.*w_ham; %% direct calculation of finite impulse response
disp(h);
figure(1);
[m,w]=freqz(h,1,128);
mag=20*log10(abs(m));
plot(1000*w/(2*pi),mag);
title('FIR filter freq response without using inbuilt method');
grid on;
N=N-1;
h1=fir1(N,wc,hamming(N+1)); %% verification using inbuilt function
disp(h1)
figure(2);
[m,w]=freqz(h1,1,128);
mag=20*log10(abs(m));
plot(1000*w/(2*pi),mag);
title('FIR filter freq response using inbuilt method');
grid on;
```

➢ *Design and implementation of FIR filter Method 1*

Generate filter coefficients for the given order, cutoff and plot the graphs:
N=33,   fc=150Hz,   fs=1000 Hz,   using Hamming window.

MATLAB Program:

```
clc;    clear all;  close all;
%generate filter coefficients for the given order & cutoff
N=33;    fc=150;    fs=1000;
h=fir1(N, fc/(fs/2),hamming(34));
figure(1);
[m,w]=freqz(h,1,128);
mag=20*log10(abs(m));
plot(fs*w/(2*pi),mag);
title('Fir filter frequency response');
grid;
%generate simulated input of 50, 300 & 200 Hz, each of 30 points
n=1:30;
f1=50; f2=300; f3=200; fs=1000;
x1=sin(2*pi*n*f1/fs);
x2=sin(2*pi*n*f2/fs);
x3=sin(2*pi*n*f3/fs);
x=[x1 x2 x3];
figure(2);
subplot(2,1,1);  stem(x);  title('Input Signal');
%generate o/p:    %y=conv(h,x);
y=filter(h,1,x);
subplot(2,1,2);  stem(y);  title('Output Signal');
```

Result:
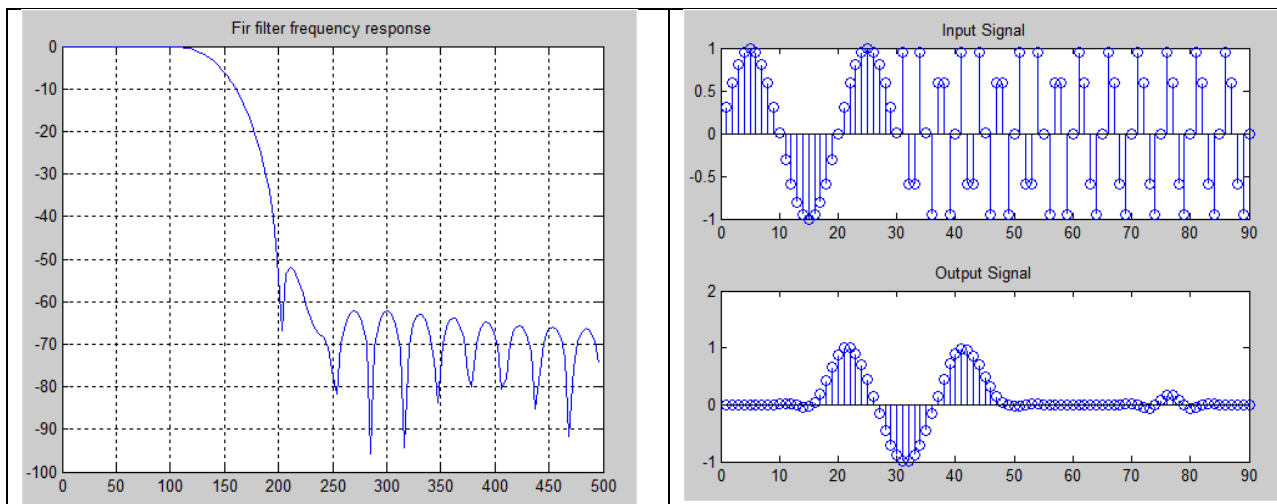
➢ *Design and implementation of FIR High pass filter*

Problem:

Design a high pass FIR filter with the following desired frequency response.
Use rectangular window of size 5.

$$H_d(e^{jw}) = 0 \qquad ; \quad |w| \leq \pi/4$$
$$= e^{jw(N-1/2)} \quad ; \quad -\pi/4 \leq |w| \leq \pi$$

Solution:

Given N=5; α=N-1/2=2; w_c= $\pi/4$

$H_d(e^{jw}) = e^{-j2w}$; $-\pi/4 \leq |w| \leq \pi$

$h_d(n)$ = $(\sin \pi(n- \alpha) – \sin (w_c(n- \alpha))) / \pi(n- \alpha)$ for all n   except $n \neq \alpha$

= 1- (w_c/π) ;   when n=α

h_d (0) = -0.16, h_d (1) = -0.22, h_d (2) = 0.75, h_d (3) = -0.23, h_d (4) = -0.16

The impulse response of the FIR filter: h(n)= h_d (n) . W(n)

W(n) = 1; n=0,1,2,3,4

= 0 ; elsewhere

Therefore   h (n) = h_d (n)

h (0)=-0.16,   h (1) = -0.22,   h (2) = 0.75,   h (3) = -0.23,   h (4) = -0.16

Matlab Program:

```
clc; clear all; close all;
Wc = 0.25*pi;
Wc = Wc /pi;
N=5;
alpha=(N-1)/2;
n=0:1:N-1;
M=n-alpha;
hd= sinc(pi*M) - (Wc*sinc(Wc*M));
W_rect = boxcar(N)';
h=hd .* W_rect %% FIR HP filter h(n) without using fir1 inbuilt function
N=N-1;
h1=fir1(N,Wc,'high', boxcar(N+1))%% h(n) of FIR  HP filter using fir1 inbuilt function
%%% These values h and h1 should match with the above theoretically  calculated values
```

Result: h = -0.1198 -0.2687   0.7500 -0.2687 -0.1198

h1 = -0.1805 -0.2552   0.8505 -0.2552 -0.1805

FIR filter design using Kaiser window:

```
clc;    clear all;  close all;
disp('FIR filter design  using Kaiser window');
M = input ('Enter the length of the filter = ');
beta= input ('Enter the value of  beta = ');
wc = input ('Enter the digital cutoff frequency = ');
wn= kaiser(M,beta);
disp('FIR Kaiser window coefficients:');
disp(wn);
hn = fir1(M-1, wc, wn);
disp('The unit sample response of FIR filter is hn=');
disp(hn);
freqz(hn,1,512);
grid on;
xlabel('Normalized frequency');
ylabel('Gain in db');
title('Freq response of FIR filter');
```

Result:


```
FIR filter design  using Kaiser window
Enter the length of the filter = 30
Enter the value of  beta = 1.2
Enter the digital cutoff frequency = 0.3
FIR Kaiser window coefficients:
  0.7175   0.7523   0.7854   0.8166
  0.8457   0.8727   0.8973   0.9195
  0.9392   0.9563   0.9706   0.9822
  0.9909   0.9967   0.9996   0.9996
  0.9967   0.9909   0.9822   0.9706
  0.9563   0.9392   0.9195   0.8973
  0.8727   0.8457   0.8166   0.7854
  0.7523   0.7175
The unit sample response of FIR filter
is hn=  0.0141      0.0028   -0.0142
        -0.0224    -0.0117    0.0133
         0.0333     0.0277   -0.0072
        -0.0495    -0.0614   -0.0140
         0.0895     0.2097    0.2900
         0.2900     0.2097    0.0895
        -0.0140    -0.0614   -0.0495
        -0.0072     0.0277    0.0333
         0.0133    -0.0117   -0.0224
        -0.0142     0.0028    0.0141
```
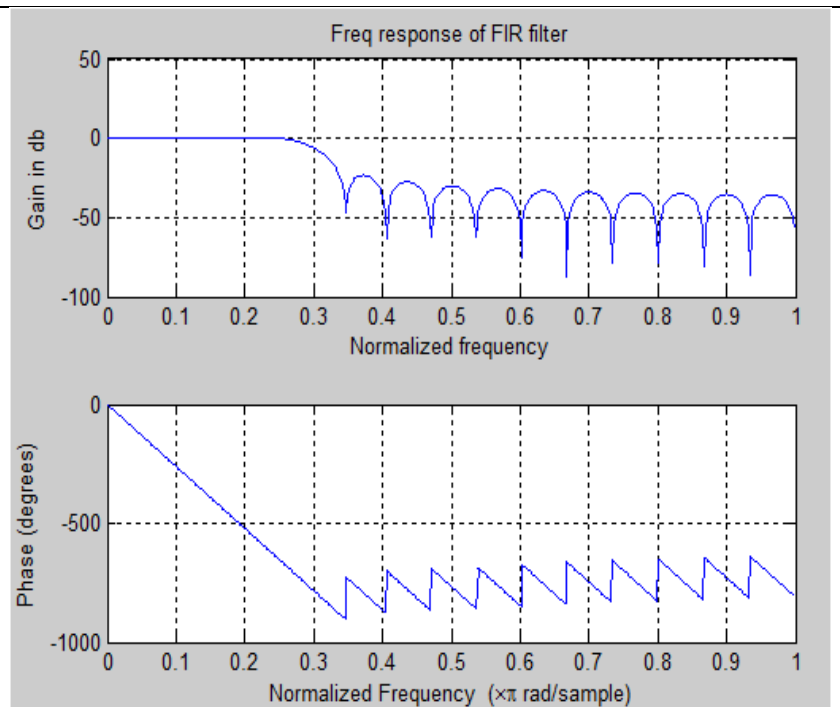
Response of FIR filter designed using Kaiser Window.

Assignment: *Design and Implementation of Filter using other Windows.*

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 / π

- **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

The matlab functions used along with a brief explanation of each is included below:

cheby1 / cheby2   **[b,a] = cheby1(n,Rp,Wn) ; [b,a] = cheby2(n,Rp,Wn)** where Wn is the cutoff frequency where the magnitude response of the filter is equal to -Rp dB Rp is the peak to peak ripple **Chebyshev Type I** filters are **equiripple** in the **passband** and **monotonic in the stopband**.
Chebyshev Type II filters are **equiripple** in the **stopband** and **monotonic in the passband** The function returns the filter coefficients in the length n+1 row vectors b and a, with coefficients in descending powers ofz. If Wn is a two-element vector, Wn = [w1 w2], cheby1 returns an order 2*n bandpass filter with passband w1 $<\omega<$ w2. The poles are evenly spaced about an ellipse in the left half plane. The Chebyshev Type I cutoff frequency ω is set to 1.0 for a normalized result. This is the frequency at which the passband ends and the filter has magnitude response of 10-Rp/20

Algorithm:
1. It finds the lowpass analog prototype poles, zeros, and gain using the cheb1ap function.
2. It converts the poles, zeros, and gain into state-space form.
3. It transforms the lowpass filter into a bandpass, highpass, or bandstop filter with desired cutoff frequencies, using a state-space transformation.
4. For digital filter design, cheby1 uses bilinear to convert the analog filter into a digital filter through a bilinear transformation with frequency prewarping. Careful frequency adjustment guarantees that the analog filters and the digital filters will have the same frequency response magnitude at Wn or w1 and w2.
5. It converts the state-space filter back to transfer function or zero-pole-gain form, as required.

cheb1ap   Chebyshev Type I analog lowpass filter prototype   **[z,p,k] = cheb1ap(n,Rp)**
It returns the poles and gain of an order n Chebyshev Type I analog lowpass filter prototype with Rp dB of ripple in the passband. The function returns the poles in the length n column vector p and the gain in scalar k. z is an empty matrix, because there are no zeros. The transfer function is   H(s)= Z(s)/p(s)= K/(S-P((1))(S-p(2))… (S-P(n))

Butter   Butterworth analog and digital filter design   **[b,a] = butter(n,Wn)**
where n is the order of the filter ,Wn is the cutoff frequency where the magnitude response is $1/\sqrt{2}$
Butterworth filters are characterized by a magnitude response that is maximally flat in the passband and monotonic overall. **Butterworth filters sacrifice rolloff steepness for monotonicity in the pass- and stopbands.** It returns the filter coefficients in length n+1 row vectors b and a, with coefficients in descending powers of z.
H(Z)= B(Z)/A(Z)= b(1)+b(2)Z^{-1}+b(3)Z^{-2}+. . . +b(n+1)Z^{-n} / 1+a(2)Z^{-1}+ . . . +a(n+1)Z^{-n}

If Wn is a two-element vector, Wn = [w1 w2], cheby2 returns an order 2*n bandpass filter with passband w1 <ω< w2.

Algorithm

1. It finds the lowpass analog prototype poles, zeros, and gain using the **buttap** function.
2. It converts the poles, zeros, and gain into state-space form.
3. It transforms the lowpass filter into a bandpass, highpass, or bandstop filter with desired cutoff frequencies, using a state-space transformation.
4. For digital filter design, butter uses bilinear to convert the analog filter into a digital filter through a bilinear transformation with frequency prewarping. Careful frequency adjustment guarantees that the analog filters and the digital filters will have the same frequency response magnitude at Wn or w1 and w2.
5. It converts the state-space filter back to transfer function or zero-pole-gain form, as required.

Transformations in filter design
Low pass to Bandpass

lp2bp   Transform lowpass analog filters to bandpass .   **[bt,at] = lp2bp(b,a,Wo,Bw)**
Transforms an analog lowpass filter prototype given by polynomial coefficients into a bandpass filter with center frequency Wo and bandwidth Bw. Row vectors b and a specify the coefficients of the numerator and denominator of the prototype in descending powers ofs. Scalars Wo and Bw specify the center frequency and bandwidth in units of rad/s. For a filter with lower band edge w1 and upper band edge w2, use Wo = sqrt(w1*w2) and Bw = w2-w1. lp2bp returns the frequency transformed filter in row vectors bt and at.

lp2hp   Transform lowpass analog filters to highpass .   **[bt,at] = lp2hp(b,a,Wo)**
lp2hp transforms analog lowpass filter prototypes with a cutoff frequency of 1 rad/s into highpass filters with desired cutoff frequency. The transformation is one step in the digital filter design process for the butter, cheby1, cheby2, and ellip functions. Transforms an analog lowpass filter prototype given by polynomial coefficients into a highpass filter with cutoff frequency Wo. Row vectors b and a specify the coefficients of the numerator and denominator of the prototype in descending powers ofs. Scalar Wo specifies the cutoff frequency in units of radians/second. The frequency transformed filter is returned in row vectors bt and at.

Bilinear   Bilinear transformation method for analog-to-digital filter conversion
[zd,pd,kd] = bilinear(z,p,k,fs) ;   [zd,pd,kd] = bilinear(z,p,k,fs,Fp)
The bilinear transformation is a mathematical mapping of variables. In digital filtering, it is a standard method of mapping the s or analog plane into the z or digital plane. It transforms analog filters, designed using classical filter design techniques, into their discrete equivalents. The function convert the s-domain transfer function specified by z, p, and k to a discrete equivalent. Inputs z and p are column vectors containing the zeros and poles, k is a scalar gain, and fs is the sampling frequency in hertz. bilinear returns the discrete equivalent in column vectors zd and pd and scalar kd. Fp is the optional match frequency, in hertz, for prewarping. It is important that the numerator order cannot be greater than that of the denominator.

Zero-Pole-Gain Algorithm

1. For a system in zero-pole-gain form, bilinear performs four steps:
2. If Fp is present, k = 2*pi*Fp/tan(pi*Fp/fs); otherwise k = 2*fs.
3. It strips any zeros at ± using z = z(find(finite(z)));
4. It transforms the zeros, poles, and gain using
   - pd = (1+p/k)./(1-p/k);
   - zd = (1+z/k)./(1-z/k);
   - kd = real(k*prod(fs-z)./prod(fs-p));
5. It adds extra zeros at -1 so the resulting system has equivalent numerator and denominator order.

Determination of the order of different filters

cheb1ord /cheb2ord   Calculate the order for a Chebyshev Type I / II filter
Syntax : **[n,Wn] = cheb1ord(Wp,Ws,Rp,Rs) [n,Wn] = cheb1ord(Wp,Ws,Rp,Rs)**
[n,Wn] = cheb2ord(Wp,Ws,Rp,Rs,'s') [n,Wn] = cheb2ord(Wp,Ws,Rp,Rs,'s')
cheb1ord /cheb2ord calculates the minimum order of a digital or analog Chebyshev Type I /Type 2 filter required to meet a set of filter design specifications. **If 's' is included then it will calculate the order for an analog filter.** It returns the lowest order n of the Chebyshev Type I / II filter that loses no more than Rp dB in the passband and has at least Rs dB of attenuation in the stopband. The scalar (or vector) of corresponding cutoff frequencies Wn, is also returned. Wp Passband corner frequency Wp, the cutoff frequency, is a scalar or a two-element vector with values between 0 and 1, with 1 corresponding to the normalized Nyquist frequency, π radians per sample. Ws Stopband corner frequency Ws, is a scalar or a two-element vector with values between 0 and 1, with 1 corresponding to the normalized Nyquist frequency. Rp Passband ripple, in decibels. This value is the maximum permissible passband loss in decibels. Rs Stopband attenuation, in decibels. This value is the number of decibels the stopband is down from the passband.

Buttord   Calculate the order and cutoff frequency for a Butterworth filter
[n,Wn] = buttord(Wp,Ws,Rp,Rs) ;   [n,Wn] = buttord(Wp,Ws,Rp,Rs,'s')
buttord calculates the minimum order of a digital or analog Butterworth filter required to meet a set of filter design specifications. It returns the lowest order, n, of the digital Butterworth filter that loses no more than Rp dB in the passband and has at least Rs dB of attenuation in the stopband. The scalar (or vector) of corresponding cutoff frequencies, Wn, is also returned. **If 's' is included it will return the order of the corresponding analog filter**. Wp Passband corner frequency Wp, the cutoff frequency, is a scalar or a two-element vector with values between 0 and 1, with 1 corresponding to the normalized Nyquist frequency, radians per sample. Ws Stopband corner frequency Ws, is a scalar or a two-element vector with values between 0 and 1, with 1 corresponding to the normalized Nyquist frequency. Rp Passband ripple, in decibels. This value is the maximum permissible passband loss in decibels. Rs Stopband attenuation, in decibels. This value is the number of decibels the stopband is down from the passband.

Butterworth Filter Design:

1. **Program to design Butterworth low/High pass filter for the following specifications:**
Pass band attenuation ≤ 1.25 db,   Stopband attenuation ≥ 15 db
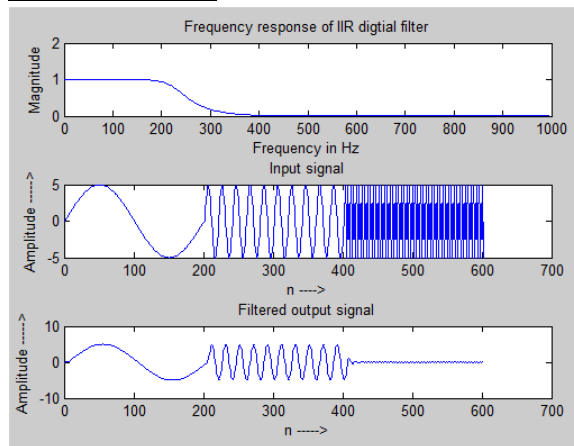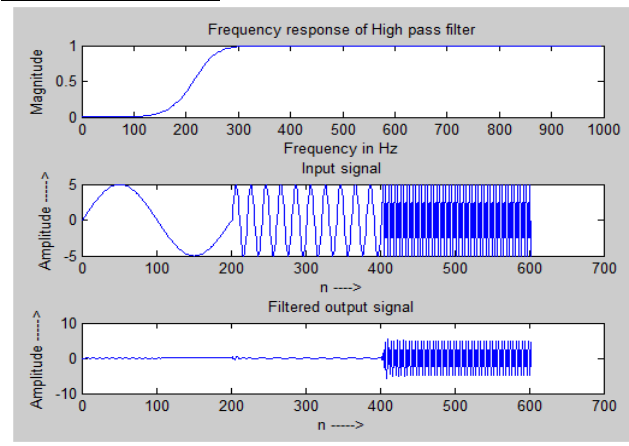**fpass =200Hz, fstop=300Hz, fsample=2kHz**

MatLAB Program:

```
clc; clear all; close all;
% Given specifications
Ap=1.25;   As=15; fpb=200;   fsb=300;   fs=2000;
% To find order (N)  and cutoff frequency (fc)
[N,fc]=buttord(200/(fs/2),300/(fs/2),Ap,As)
% Low pass filter
[b,a]=butter(N ,fc)
 % High pass filter
 %[b,a]=butter(N ,fc,'high')
[H,f]=freqz(b,a,256,fs);
subplot(3,1,1);    plot(f,abs(H));
title('Frequency response of  IIR digtial filter');
xlabel('Frequency in Hz');  ylabel('Magnitude');
%filtering operation on input signal having frequency 10Hz, 100Hz, 500Hz
n=0:1/fs:0.1;
s1=5*sin(2*pi*10*n);
s2=5*sin(2*pi*100*n);
s3=5*sin(2*pi*500*n);
x=[s1 s2 s3];
subplot(3,1,2);    plot(x);
title('Input signal');  xlabel('n ---->'); ylabel('Amplitude ----->');
y=filter(b,a,x);
subplot(3,1,3);    plot(y);
title('Filtered output signal');  xlabel('n ----->'); ylabel('Amplitude ----->');
```

Result:

Low Pass Filter	*High Pass Filter*
N = 6	*N = 6*
fc = 0.2329	*fc = 0.2329*
b = 0.0007 0.0044 0.0111 0.0148	*b = 0.2333 -1.4000 3.5000 -4.6667*
0.0111 0.0044 0.0007	*3.5000 -1.4000 0.2333*
a = 1.0000 -3.1836 4.6222 -3.7795	*a = 1.0000 -3.1836 4.6222 -3.7795*
1.8136 -0.4800 0.0544	*1.8136 -0.4800 0.0544*

Low Pass Filter

High Pass Filter

2. **Program to design Butterworth Band pass/Band stop digital filter for the following specifications:** **f1=25Hz;** **f2=225Hz;** **fL=100Hz;** **fU=150Hz;** **Stopband attenuation=18dB; Pass band attenuation=3dB; Sampling frequency=500Hz.**
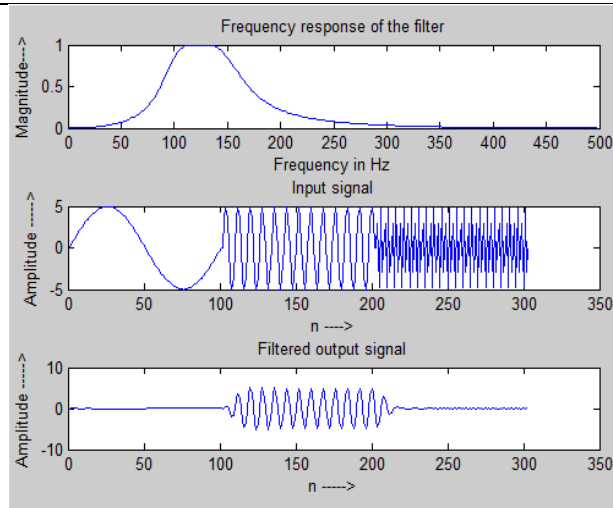
MatLAB Program:

```
clc; clear all; close all;
% Given specifications
Ap=3;   As=18;  f1=25;  fL=100;  fH=150;   f2=225; fs=1000;
% To find order (N)  and cutoff frequency (fc)
fpbn=[100/(fs/2),150/(fs/2)];
fsbn=[25/(fs/2),225/(fs/2)];
[N,fc]=buttord(fpbn,fsbn,Ap,As)
% To compute frequency response of an IIR digital filter
% Band Pass filter
[b,a]=butter(N,fc)
% Band stop filter
%[b,a]=butter(N ,fc,'stop')
[H,f]=freqz(b,a,256,fs);
subplot(3,1,1);    plot(f,abs(H));
title('Frequency response of the filter');
xlabel('Frequency in Hz');  ylabel('Magnitude--->');
%filtering operation on input signal having frequency 10Hz,125Hz,400 Hz
n=0:1/fs:0.1;
s1=5*sin(2*pi*10*n);
s2=5*sin(2*pi*125*n);
s3=5*sin(2*pi*400*n);
x=[s1 s2 s3];
subplot(3,1,2);    plot(x);
title('Input signal');  xlabel('n ---->'); ylabel('Amplitude ----->');
y=filter(b,a,x);
subplot(3,1,3);    plot(y);
title('Filtered output signal');  xlabel('n ----->'); ylabel('Amplitude ----->');
```
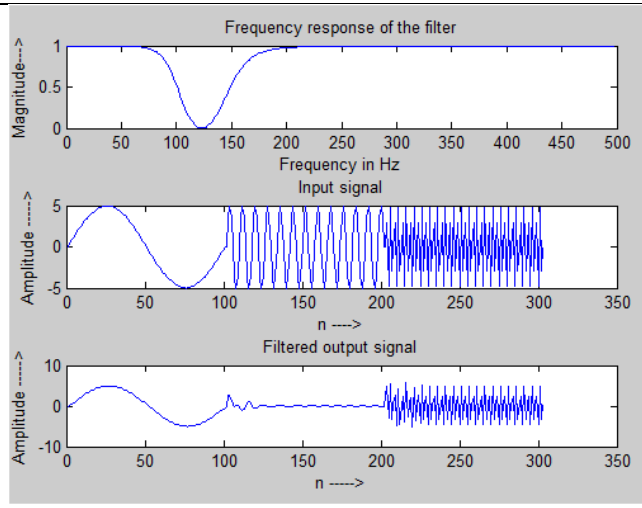
Band Pass filter	**Band stop filter**
N = 2	N = 2
fc = 0.1890 0.3158	fc = 0.1890 0.3158
b =0.0307 0 -0.0614 0 0.0307	b = 0.7541 -2.1595 3.0542 -2.1595 0.7541
a =1.0000 -2.4676 2.9928 -1.8513 0.5696	a = 1.0000 -2.4676 2.9928 -1.8513 0.5696

3. **Program to design Butterworth low pass filter for the following specifications:**

    N=2;   fc=150Hz;   fs=1000 Hz;

MATLab Program:

```
clc; clear all; close all;
N=2; fc= 150; f1=100; f2=300; f3=170; fs=1000;
[b,a]=butter(N, fc/(fs/2)) %% IIR filter  Nr and Dr coefficients
%generate simulated input of 100, 300 & 170 Hz, each of 30 points
n=1:30;
x1=sin(2*pi*n*f1/fs);
x2=sin(2*pi*n*f2/fs);
x3=sin(2*pi*n*f3/fs);
x=[x1 x2 x3];
subplot(2,1,1); stem(x);    title('input');
%generate o/p
y=filter(b,a,x);
subplot(2,1,2); stem(y);    title('output');
```

Result:

b = 0.1311 0.2622 0.1311 a = 1.0000 -0.7478 0.2722 Plot shows that 100 Hz is passed, while 300 is cutoff and 170 have slight attenuation.	
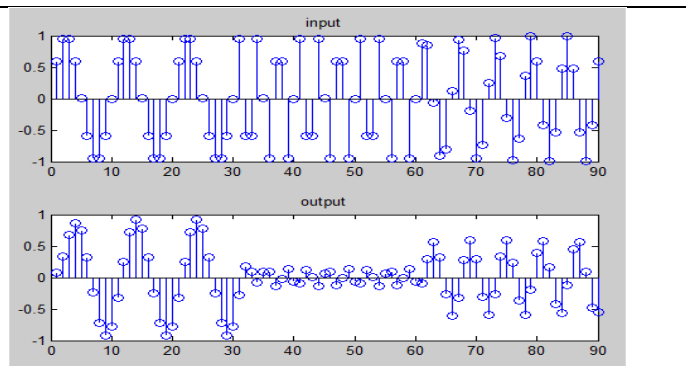
Note: **If fp, fstp, fs, rp, As are given then use:**

        [N,wc]=buttord(2*fp/fs,2*fstp/fs,rp,As)
        [b,a]=butter(N,wc);

If wp & ws are in radians:

        [N,wc]=buttord(wp/pi, ws/pi, rp, As)
        [b,a]=butter(N,wc);

If wc is in radians & N is given:

        [b,a]=butter(N,wc/pi);

For a bandpass output:

        wc=[150/(1000/2) 250/(1000/2)];
        [b,a]=butter(4, wc);

4. Program to design Butterworth low pass filter for the following specifications:
Pass band attenuation = 1dB; Stop band attenuation = 30dB
Pass band edge frequency = 1500Hz; Stop band edge frequency = 2000Hz

Solution:

    f1 = 1500Hz   **ω_p = 2.Π.1500/8000 = 1.178**
    f2 = 2000Hz   **ω_s = 2.Π.2000/8000 = 1.57**
   Prewarping:
    ω_{p1} = 2.tan($\omega_{p/2}$)=1.336
    ω_{s1} = 2.tan($\omega_{s/2}$)=1.998
    N = log[($10^{-0.1Ap}$-1)/ ($10^{-0.1As}$-1)]/2log[$\omega'_{p/}$ ω'_s] = 10.259 = 11
    $\omega_{c =}$ $\omega'_{p/}$ ($10^{-0.1Ap}$-1)$^{1/2N}$ = 1.42

Programs for designing of IIR filters (to verify the above problem):

```
clc;    clear all; close all;
%  Butterworth filter: Given Specifications:
rp=1;  rs=30; w1=1500; w2=2000; ws=8000;
% Analog frequency
aw1=2*pi*w1/ws
aw2=2*pi*w2/ws
% Prewrapped frequency
pw1 = 2*tan(aw1/2)
pw2 = 2*tan(aw2/2)
%Calculate order and cutoff freq
[n,wc]= buttord (pw1,pw2,rp,rs,'s')
% The order n should match with corresponding theoretical value.
% analog filter transfer
[b,a] = butter(n,wc,'s');
% The b and a values should match with Ha(s)
%obtaining the digital filter using bilinear transformation
fs=1;
```

```
[num,den]= bilinear(b,a,fs)
% The num and den should match with H(z)
%plot the frequency response
[mag,freq1]=freqz(num,den,128);
freq=freq1*ws/(2*pi);
m = 20*log10(abs(mag));
plot(freq,m);
grid;
```
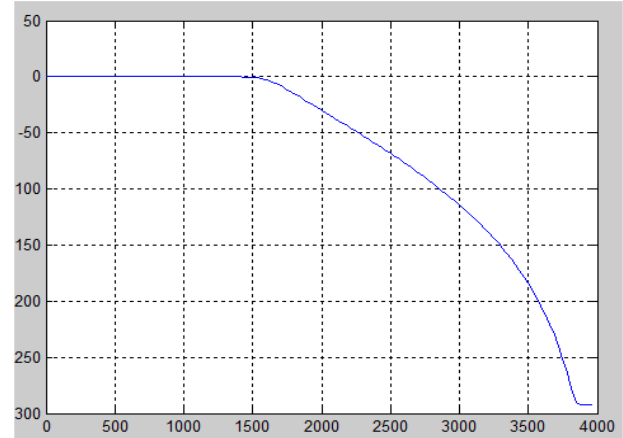
Result:

```
aw1 =   1.1781
aw2 =   1.5708
pw1 =   1.3364
pw2 =   2.0000
n =   11
wc =   1.4611
num =   0.0002   0.0027   0.0134   0.0401
0.0801   0.1121   0.1121   0.0801   0.0401
0.0134   0.0027   0.0002
den =   1.0000  -2.1564   3.4781  -3.5442
2.8160  -1.6443   0.7469  -0.2521   0.0631
-0.0109   0.0012  -0.0001
```


5. **Program to design a chebyshev filter for the following specifications:**

Pass band attenuation = -1dB;   Stop band attenuation = -30dB

Pass band edge frequency = 1500Hz;   Stop band edge frequency = 2000Hz

Sampling Frequency = 8000Hz;   Magnitude of ripple(p-p) in passband = -1dB

Step1: $\Omega = 2\pi f$ $\Omega_p = 2\pi 1500 = 9424.77$ $\Omega_s = 2\pi 2000 = 12566.37$

Step2: $\Omega \to \omega$ $\omega_p = 9424.77/8000 = 1.178\,rad$ $\Omega s = 12566.37/8000 = 1.57\,rad$

Step3: $\omega \to \Omega$ prewarping $\Omega = (2/T)\tan(\omega/2)$

    $\Omega_p = 2\tan(1.178/2) = 1.336\,rad/sec$ $\Omega_s = 2\tan(1.157/2) = 1.998\,rad/sec$

Step4: $\Omega_p = 1$ $\Omega_s = \Omega_s/\Omega_p = 1.998/1.336 = 1.495$ $\delta_p = -1dB$, $\delta_s = -30dB$

Step5: $N = \cosh^{-1}[(10^{-0.1As}-1)/(10^{-0.1Ap}-1)]^{1/2}/\cosh^{-1}[\Omega_s/\Omega_p] = 5.031 \approx 6$

    $\Omega_c = \Omega_p \cosh[(1/N)\cosh^{-1}(1/\varepsilon)]$ $\varepsilon = sqrt(10^{-0.1Ap}-1) = 0.5088$

    $\Omega_c = 1.3673\,rad/s$ $f_c = \Omega_c/2\pi = 0.2173Hz$

MATLab Program:

```
clc;   clear all;  close all;
%Given Specifications
rp=1;  rs=30; w1=1500;   w2=2000;   ws=8000;
%Analog frequencies
aw1= 2*pi*w1/ws
aw2= 2*pi*w2/ws
% Prewrapped frequency assuming T=1/fs
pw1 = 2*tan(aw1/2)
pw2 = 2*tan(aw2/2)
[n,wc]= cheb1ord (pw1,pw2,rp,rs,'s')
[b,a] = cheby1(n,rp,wc,'s');
%obtaining the digital filter using bilinear transformation
fs=1;
[num,den]= bilinear(b,a,fs)
%plot the frequency response
[mag,freq1]=freqz(num,den,128);
freq=freq1*ws/(2*pi);
m = 20*log10(abs(mag));
plot(freq,m);
grid;
```
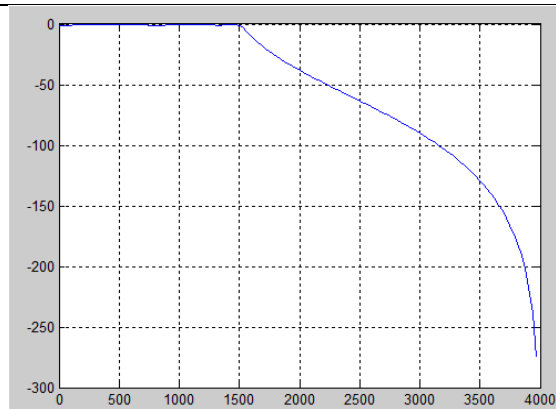
Result:

```
aw1 =   1.1781
aw2 =   1.5708
pw1 =   1.3364
pw2 =   2.0000
n =    6
wc =   1.3364
num =   0.0018   0.0107   0.0267   0.0355
        0.0267   0.0107   0.0018
den =   1.0000  -3.1316   5.2916  -5.5861
        3.8414  -1.6245   0.3368
```


---

6.   Program to design IIR- chebyshev filter for the given specifications:

```
clc;   clear all; close all;
wp=input('Enter the passband edge Normalised frequency = ');
%wp=wn, for 3dB cutoff frequency
ws=input('Enter the stopband edge Normalised frequency = ');
%wp=[w1 w2]; ws=[w3 w4]; %wp & ws are vectors, for Bandpass & Bandstop
Rp=input('Enter the passband attenuation level (dB) = ');
Rs=input('Enter the stopband attenuation level (dB) = ');
%N=input('Enter the order of the filter = '); %order=N/2, for Bandpass & Bandstop
[N,wn]=cheb1ord(wp,ws,Rp,Rs)
```

```
%skip if order and 3dB cutoff frequency is known
[b,a]=cheby1(N,Rp,wn) %replace by following for other filters
%[b,a]=cheby1(N,Rp,wn,'high'); Highpass filter
%[b,a]=cheby1(N,Rp,wn); Bandpass filter
%[b,a]=cheby1(N,Rp,wn,'stop'); Bandstop filter
[h,w]=freqz(b,a);
mag=20*log10(abs(h));
phase=180*angle(h)/pi;
figure(1);
plot(w,abs(h));          title('Chebyshev Lowpass Filter');
xlabel('Normalised frequency');        ylabel('Magnitude');   grid;
figure(2);
subplot(2,1,1); plot(w,mag);    title('Magnitude response');
xlabel('Normalised frequency');        ylabel('Magnitude in dB');      grid;
subplot(2,1,2); plot(w,phase);  title('Phase response');
xlabel('Normalised frequency');        ylabel('Phase in degrees');     grid;
```

Result:

Enter the passband edge Normalised frequency = .3

Enter the stopband edge Normalised frequency = .6
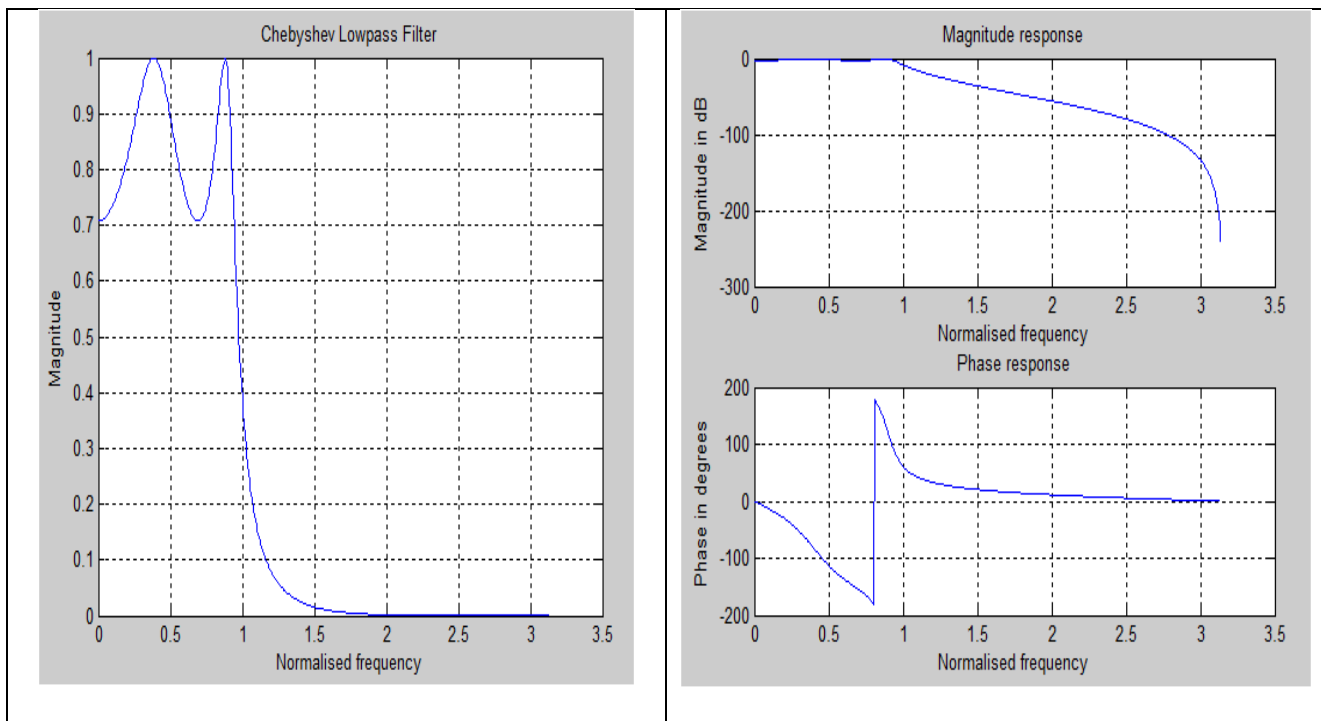
Enter the passband attenuation level (dB) = 3

Enter the stopband attenuation level (dB) = 40

N =   4   ;   wn =   0.3000

b =   0.0051   0.0203   0.0304   0.0203   0.0051

a =   1.0000   -2.6649   3.2814   -2.0817   0.5798

<div align="center">

Experiment No : 10
**Design FIR Low Pass Filter Using Hamming Window**

</div>

Aim: Construct a Simulink model of a FIR LPF for given specifications and observe the time domain waveform and spectrum of filtered signal.

Problem Statement: Design a 9th order FIR low pass filter using hamming window in Simulink for the following specifications:
f_s = 2000 Hz; f_c = 100Hz;
Analyze the result for a sinusoidal signal using a simple scope and a FFT scope.

Method Involved:

➢ Open the Simulink Library Browser by clicking on   the icon.

➢ Click on the 'new model'   icon.

➢ Now, come back to the Simulink Library Browser and select the source block as follows:
    Dsp System Toolbox ⟶ **Sources** ⟶ **Sine Wave** ⟶ **Right Click** ⟶ **Add to Untitled**

➢ To add the filter block, select **Filtering** option in the DSP System Toolbox, and select the **Filter Implementations** option under it.

➢ Now, right click on the **Digital Filter Design** block's icon and select **add to Untitled**.

➢ Next, select the **Sinks** option in the DSP System Toolbox, and right click on the **Time Scope** option and **add to Untitled**.

➢ Your model named 'Untitled' should be looking like this, after you drag the blocks around to align them as needed.
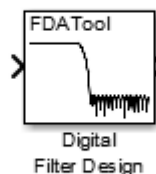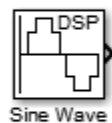
Note: You can connect them by drawing arrows using your cursor.
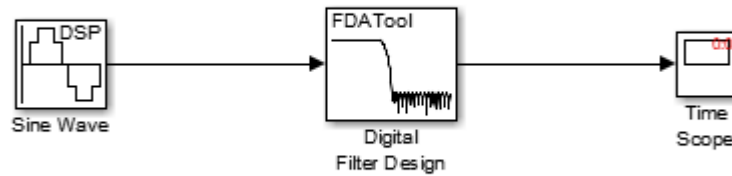
➢ Double click on the **Sine Wave** block, and in the window that pops up, set the Amplitude to 10, the Frequency to 50 Hz and click on apply.

➢ Double click on the **Digital Filter Design** block, and set the properties as needed in the specifications, as shown below:

  o <u>Response Type</u>: Select the option **Lowpass**.
  o <u>Design Method</u>: Select **FIR** and in the dropdown menu, choose **window**.
  o <u>Options</u>: In the **window** option's drop down menu, select **Hamming**.
  o <u>Filter Order</u>: Specify the order as **9**.
  o <u>Frequency Specifications</u>: Under this section, set the sampling frequency (F_s) as **2000** Hz and the cutoff frequency (F_c) as **100** Hz.

➢ Now, click on **Design Filter**, and observe the Magnitude response as shown in the image.
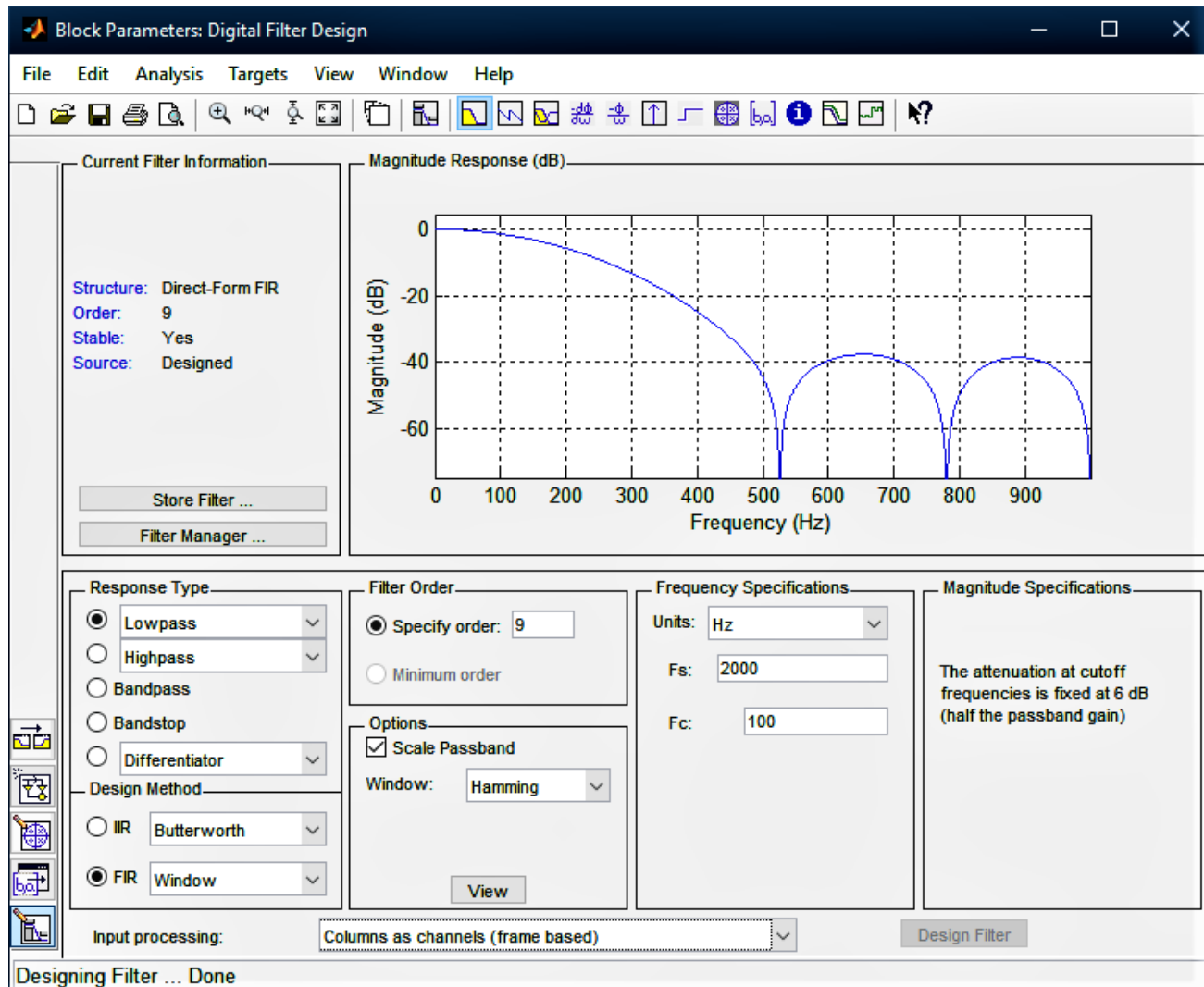
> ➢ Set the **Simulation Stop Time** (shown as a text box) to 1, in the model window and click on **run** (green colored play button) as shown below.
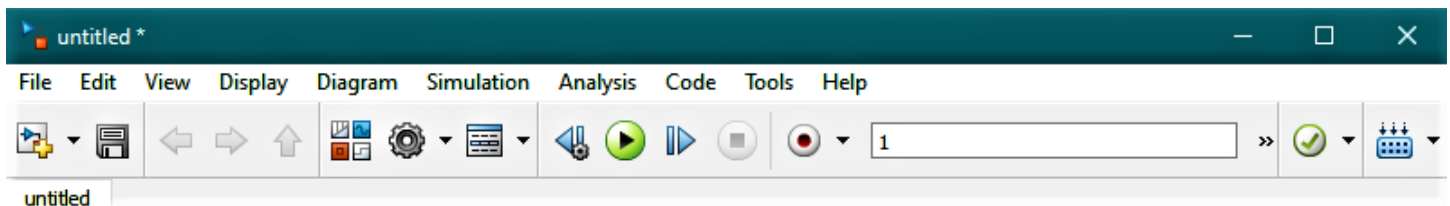
> ➢ To use the FFT sink, first add the **FFT block** from the **DSP System Toolbox** ➜ **Transforms**. Then add the **Spectrum Analyzer** block from **DSP System Toolbox** ➜ **Sinks**.

> ➢ Connect the blocks as shown below. Set the **Simulation Stop Time** to 5 and click run.
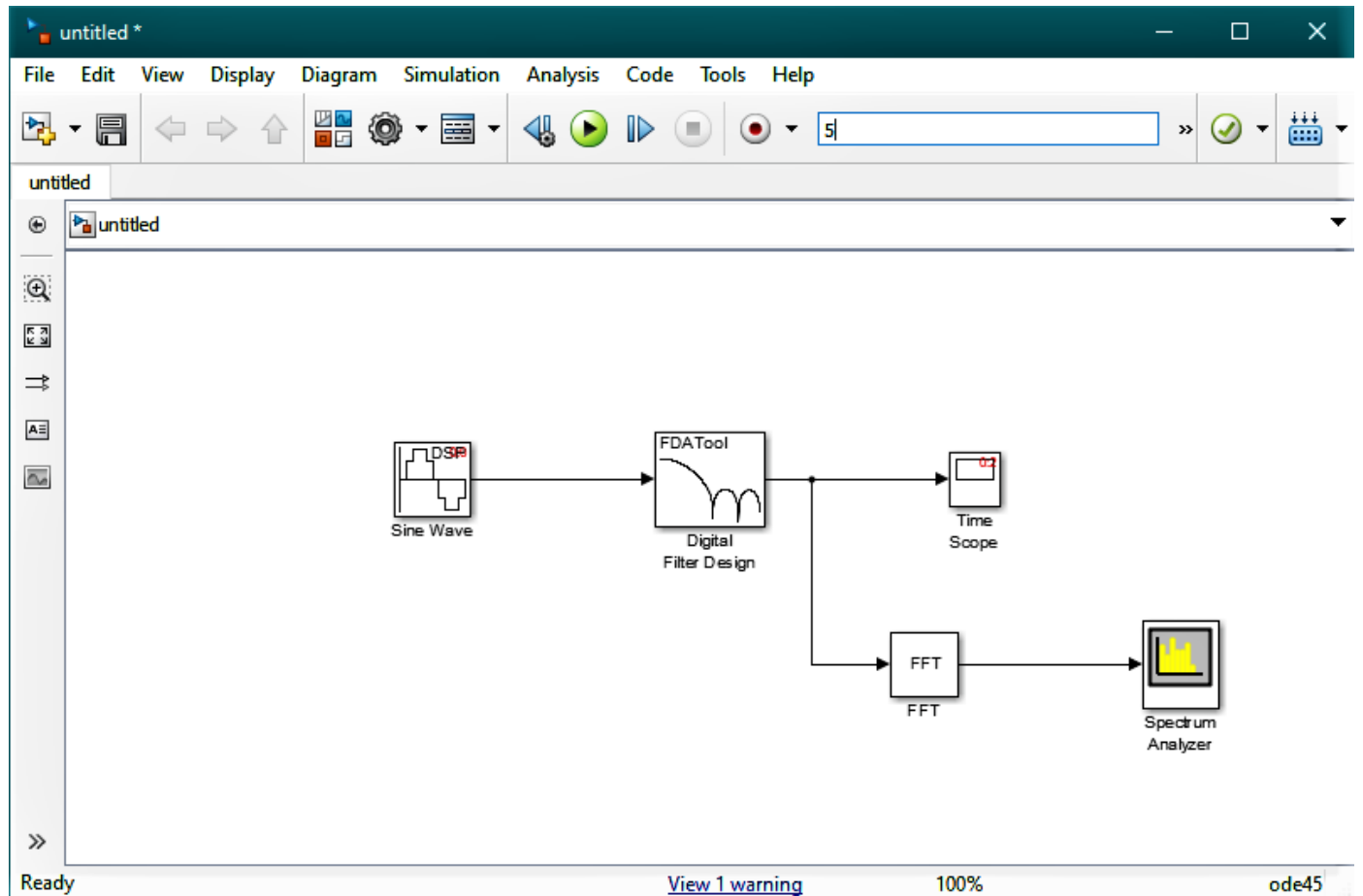
Outputs
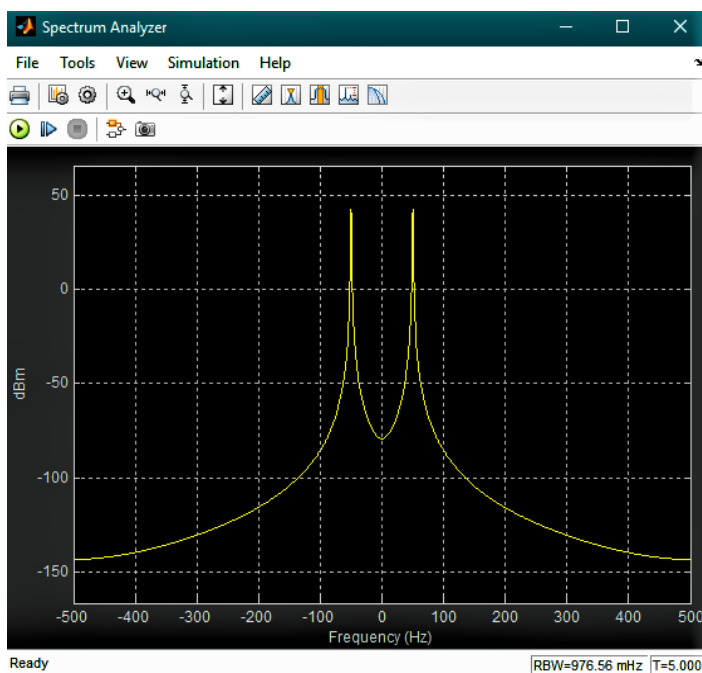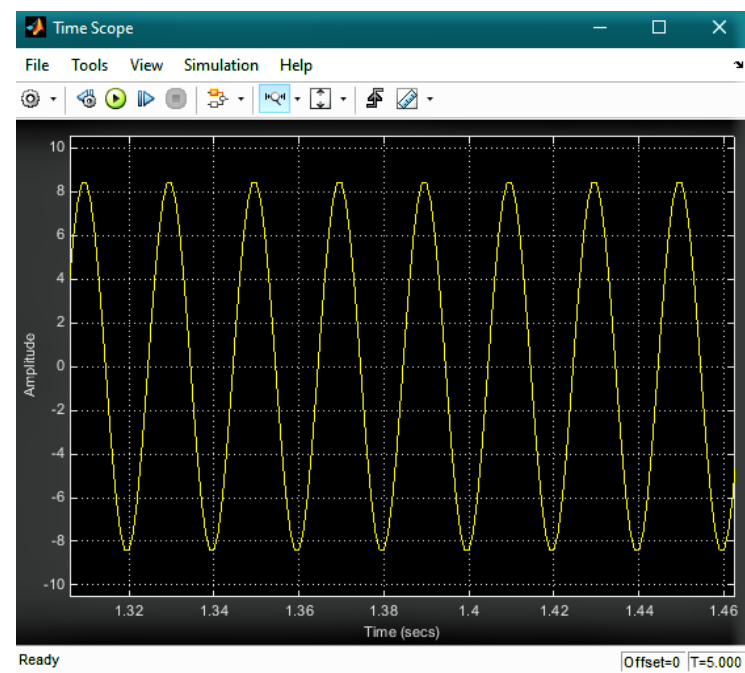
FFT SCOPE OUTPUT   **TIME SCOPE OUTPUT**

TMS320C6713 DSK EXPERIMENTS

TMS320C6713 DSK

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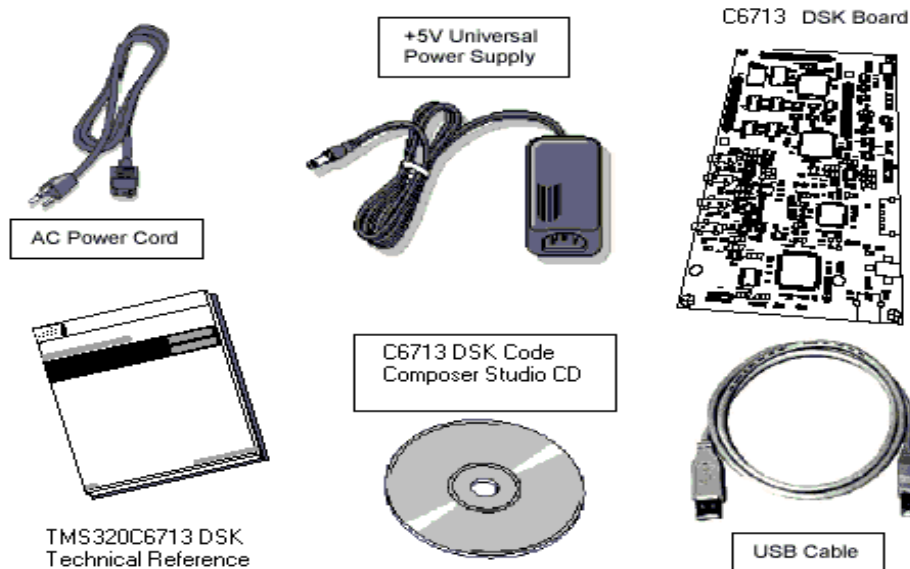
The C6713™ DSK builds on TI's industry-leading line of low cost, easy-to-use DSP Starter Kit (DSK) development boards. The high-performance board features the TMS320C6713 floating-point DSP. Capable of performing 1350 million floating-point operations per second (MFLOPS), the C6713 DSP makes the C6713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

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

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

> ➤ TMS320C6713 DSK TMS320C6713 DSK development board
>
> ➤ Other hardware External 5VDC power supply
>
>   IEEE 1284 compliant male-to-female cable
>
> ➤ CD-ROM Code Composer Studio DSK tools

The C6713 DSK has a TMS320C6713 DSP onboard that allows full-speed verification of code with Code Composer Studio. The C6713 DSK provides:

- A USB Interface

- SDRAM and ROM

- An analog interface circuit for Data conversion (AIC)

- An I/O port

- Embedded JTAG emulation support

Connectors on the C6713 DSK provide DSP external memory interface (EMIF) and peripheral signals that enable its functionality to be expanded with custom or third party daughter boards.

The DSK provides a C6713 hardware reference design that can assist you in the development of your own C6713-based products. In addition to providing a reference for interfacing the DSP to various types of memories and peripherals, the design also addresses power, clock, JTAG, and parallel peripheral interfaces.

The C6713 DSK includes a stereo codec. This analog interface circuit (AIC) has the following characteristics:

High-Performance Stereo *Codec*

- 90-dB SNR Multibit Sigma-Delta ADC (A-weighted at 48 kHz)
- 100-dB SNR Multibit Sigma-Delta DAC (A-weighted at 48 kHz)
- 1.42 V – 3.6 V Core Digital Supply: Compatible With TI C54x DSP Core Voltages
- 2.7 V – 3.6 V Buffer and Analog Supply: Compatible Both TI C54x DSP Buffer Voltages
- 8-kHz – 96-kHz Sampling-Frequency Support

Software Control Via TI McBSP-Compatible Multiprotocol Serial Port

- I 2 C-Compatible and SPI-Compatible Serial-Port Protocols
- Glueless Interface to TI McBSPs
-

Audio-Data Input/Output Via TI McBSP-Compatible Programmable Audio Interface

- I 2 S-Compatible Interface Requiring Only One McBSP for both ADC& DAC
- Standard I 2 S, MSB, or LSB Justified-Data Transfers
- 16/20/24/32-Bit Word Lengths

The C6713DSK has the following features:

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

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

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

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

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

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

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

TMS320C6713 DSP Features

Highest-Performance Floating-Point Digital Signal Processor (DSP):

- ➢ Eight 32-Bit Instructions/Cycle
- ➢ 32/64-Bit Data Word
- ➢ 300-, **225**-, 200-MHz (GDP), and **225-**, 200-, 167-MHz (PYP) Clock Rates
- ➢ 3.3-, 4.4-, 5-, 6-Instruction Cycle Times
- ➢ 2400/1800, 1800/1350, 1600/1200, and 1336/1000 MIPS /MFLOPS
- ➢ Rich Peripheral Set, Optimized for Audio
- ➢ Highly Optimized C/C++ Compiler
- ➢ Extended Temperature Devices Available
- ❖ Advanced Very Long Instruction Word (VLIW) TMS320C67x™ DSP Core
  - ➢ Eight Independent Functional Units:

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

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

> ➢ Extensive Error Checking and Recovery
❖ Two Inter-Integrated Circuit Bus (I²C Bus™) Multi-Master and Slave Interfaces
❖ Two Multichannel Buffered Serial Ports:
> ➢ Serial-Peripheral-Interface (SPI)
> ➢ High-Speed TDM Interface
> ➢ AC97 Interface
❖ Two 32-Bit General-Purpose Timers
❖ Dedicated GPIO Module With 16 pins (External Interrupt Capable)
❖ Flexible Phase-Locked-Loop (PLL) Based Clock Generator Module
❖ IEEE-1149.1 (JTAG [†]) Boundary-Scan-Compatible
❖ Package Options:
> ➢ 208-Pin PowerPAD™ Plastic (Low-Profile) Quad Flatpack (PYP)
> ➢ 272-BGA Packages (GDP and ZDP)
❖ 0.13-µm/6-Level Copper Metal Process
> ➢ CMOS Technology
❖ 3.3-V I/Os, 1.2 [‡] -V Internal (GDP & PYP)
❖ 3.3-V I/Os, 1.4-V Internal (GDP)(300 MHz only)
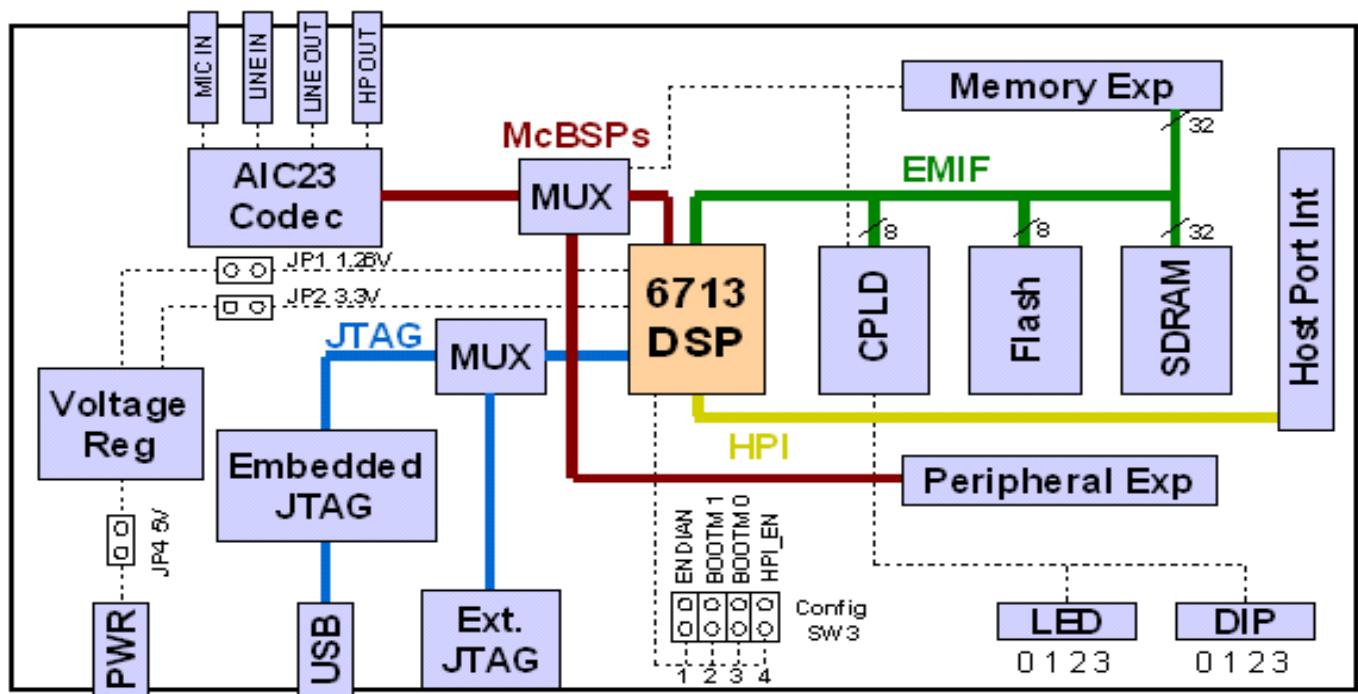
TMS320C6713 DSK Overview Block Diagram

Troubleshooting DSK Connectivity

If Code Composer Studio IDE fails to configure your port correctly, perform the following steps:

• Test the USB port by running DSK Port test from the start menu

Use Start→Programs→Texas Instruments→Code Composer Studio→Code Composer Studio C6713 DSK Tools→C6713 DSK Diagnostic Utilities

- *The below Screen will appear*
- *Select→ Start→Select 6713 DSK Diagnostic Utility Icon from Desktop*
- *The Screen Look like as below*
- *Select **Start** Option*
- *Utility Program will test the board*
- *After testing Diagnostic Status you will get **PASS***
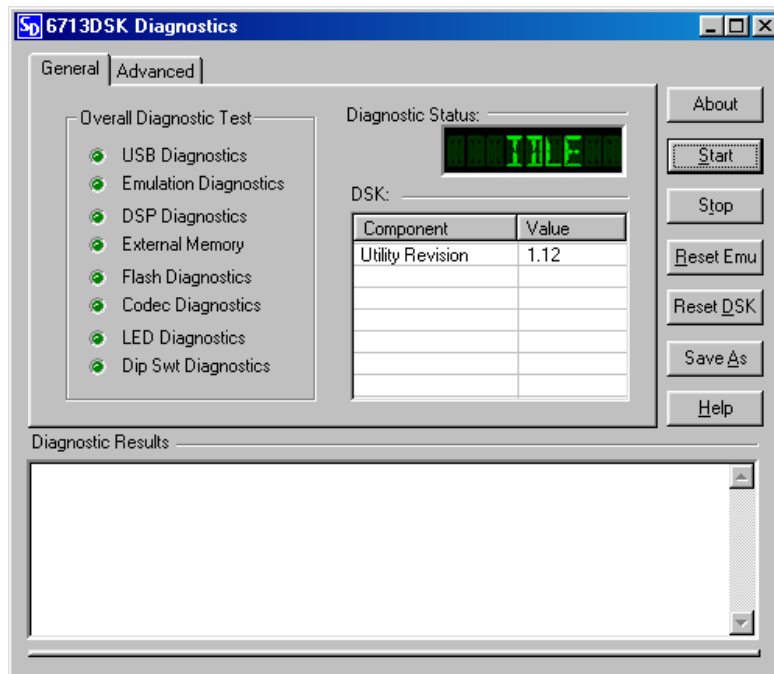
INTRODUCTION TO CODE COMPOSER STUDIO

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

CODE COMPOSER FEATURES INCLUDE:

- IDE
- Debug IDE

- Advanced watch windows

- Integrated editor

- File I/O, Probe Points, and graphical algorithm scope probes

- Advanced graphical signal analysis

- Interactive profiling

- Automated testing and customization via scripting

- Visual project management system

- Compile in the background while editing and debugging

- Multi-processor debugging

- Help on the target DSP

Note:

Documents for Reference:

spru509 → **Code Composer Studio getting started guide.**

spru189 → **TMS320C6000 CPU & Instruction set guide**

spru190 → **TMS320C6000 Peripherals guide**

slws106d → **codec(TLV320AIC23) Data Manual.**

spru402 → **Programmer's Reference Guide.**

sprs186j → **TMS320C6713 DSP**

CCS 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: \ ti \ tutorial \ dsk6713 \ hello1 \ hello.cmd

   < Select type of file as: Linker Command file>

   Note: for simulator using 5402 add the below 2 files

   1) C:\CCStudio_v3.1\C5400\cgtools\lib\rts.lib

   2) C:\CCStudio_v3.1\tutorial\dsk5402\datedisplay\mainapplication.cmd

4. File → New → source file → same as xx.c

   (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 11(a) : LINEAR CONVOLUTION

Aim: **To perform linear convolution for the given sequences**

To Verify Linear Convolution:

Linear Convolution involves the following operations.

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

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

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

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

Algorithm to implement 'C' or Assembly program for Convolution:

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

Where: n=4, k=4.   ;Values of n & k should be a multiple of 4.
                      If n & k are not multiples of 4, pad with zero's to make
                      multiples of 4
          r = n+k-1   ; Size of output sequence.
            = 4+4-1
            = 7.

r=	0	1	2	3	4	5	6
n= 0	x[0]h[0]	x[0]h[1]	x[0]h[2]	x[0]h[3]			
1		x[1]h[0]	x[1]h[1]	x[1]h[2]	x[1]h[3]		
2			x[2]h[0]	x[2]h[1]	x[2]h[2]	x[2]h[3]	
3				x[3]h[0]	x[3]h[1]	x[3]h[2]	x[3]h[3]

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

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

'C' PROGRAM TO IMPLEMENT LINEAR CONVOLUTION

```
#include<stdio.h> // Header file inclusion
int m,n,i,j,k,x[30]=0,h[30]=0,y[30]=0;//Local variables and array declaration
void main()
{
        printf ("Enter the length of the first sequence");
        scanf ("%d",&m);            //Reading length of 1st sequence, max=30
        printf ("Enter the length of the second sequence");
        scanf ("%d",&n);             //Reading length of 2nd sequence
        printf ("Enter first sequence\n");
        for (i=0;i<m;i++)            //input m values of 1st sequence
            scanf("%d",&x[i]);           //x[i] holds 1st sequence values
            printf("Enter the second sequence\n");
            for (j=0;j<n;j++)        //input m values of 2nd sequence
                scanf("%d",&h[j]);          //h[j] holds 2nd sequence values
                for (i=0;i<m+n-1;i++)       //Max length of convoluted o/p is m+n
                    { y[i]=0;               //o/p array element initialized to 0
                        for (j=0;j<=i;j++)
                        y[i]+=x[j]*h[i-j];  //multiply and add partial products for convolution
                    }
                printf("The linear convolution is \n");
        for (i=0;i<m+n-1;i++)
        printf("%d\t",y[i]);//printing convoluted o/p
}
```

Result:

```
Enter the length of the first sequence = 4
Enter the length of the second sequence = 4
Enter the first sequence     1 2 3 4
Enter the second sequence   1 2 3 4

The linear convolution is

        1       4       10      20      25      24      1
```

PROCEDURE:

➢ Open Code Composer Studio, make sure the DSP kit is turned on.

➢ Start a new project using 'Project-new ' pull down menu, save it in a separate directory(c:\ti\myprojects) with name **lconv.pjt.**

➢ Add the source files **conv.c**
➢ to the project using 'Project→add files to project' pull down menu.

➢ Add the linker command file **hello.cmd** .
   (Path: c:\ti\tutorial\dsk6713\hello1\hello.cmd)

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

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

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

➢ To View output graphically
   Select view → graph → time and frequency.

Configure the graphical window as shown below:
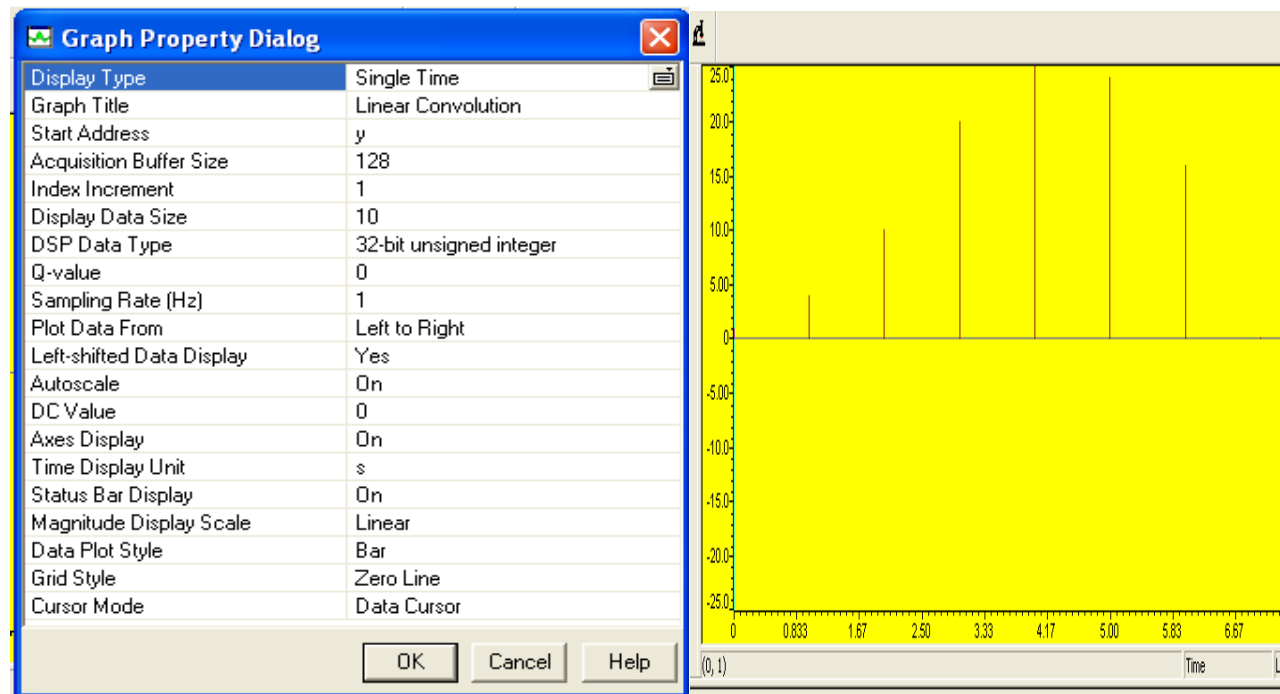
EXPERIMENT NO 11(b) : CIRCULAR **CONVOLUTION**

Aim: **To perform circular convolution of two given sequences**

Steps for Cyclic Convolution

Steps for cyclic convolution are the same as the usual convolution, except all index calculations are done "mod N" = "on the wheel"
        Steps for Cyclic Convolution

Step1: "Plot f[m] and h[−m]

Subfigure 1.1   **Subfigure 1.2**

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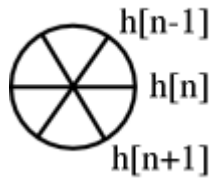
Figure 2: Step 2

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

Step 4: Repeat for all 0≤*n*≤*N*−1

Example 1: Convolve (n = 4)
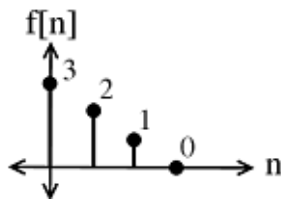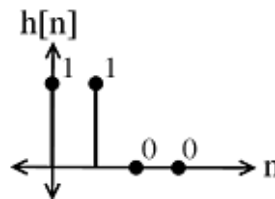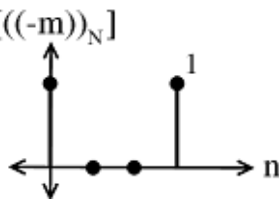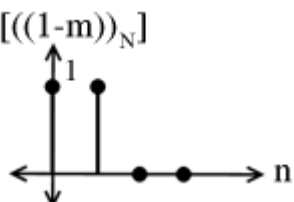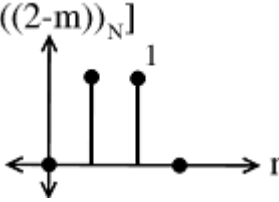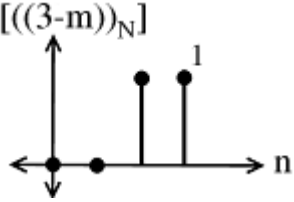
Subfigure 3.1   **Subfigure 3.2**
Figure 3: Two discrete-time signals to be convolved

1. $h[-m] =$	**2.** $h[1-m] =$
$h[((-m))_N]$ Figure 4 Multiply $f[m]$ and sum to yield: $y[0] = 3$	$h[((1-m))_N]$ Figure 5 Multiply $f[m]$ and sum to yield: $y[1] = 5$
3. $h[2-m] =$	**4.** $h[3-m] =$
$h[((2-m))_N]$ Figure 6 Multiply $f[m]$ and sum to yield: $y[2] = 3$	$h[((3-m))_N]$ Figure 7 Multiply $f[m]$ and sum to yield: $y[3] = 1$

Program to Implement Circular Convolution

```
# include<stdio.h>
# include<math.h>// Required  if trigonometric or special math functions are used
void main( )
  {
      int x[30],h[30],y[30];
      int i,j,m,n,k;
      printf("Enter the length of the first sequence\n");
      scanf("%d",&m);
      printf("Enter the length of the second sequence\n");
      scanf("%d",&n);

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

      if(m-n!=0)// checking condition for length of m & n
      {
        if(m>n)
          {
              for(i=n;i<m;i++)  //from nth element of 1st seq zero is padded in 2nd seq
              h[i]=0;          // zero padding in 2nd sequence
              n=m;
          }

         for (i=m; i<n; i++)//from mth ele of 2nd seq zero is padded in 1st  seq (executed if n>m)
```

```
        x[i]=0;
        m=n;
    }
    for(i=0;i<m;i++)//for all elements of 1st sequence
        {
            y[i]=0;                    // o/p array is initialized to ZERO
            for(j=0;j<m;j++)        //Looping for entire length of sequence
                {
                    k=(i-j)%m; //No. of shift (limiting no. of shifts within max length)
                    if(k<0)
                    k=k+m;// Shift index is stored in k
                    y[i]=y[i]+h[j]*x[k]; //Calculating convoluted o/p
            }
            printf ("%d\t", y[i]);
        }
    printf("is circular convolution o/p");//String display
}
```

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

h[4] = { 1, 1, 0,0}

OUT PUT: y[4] = {3, 5, 3,1}

PROCEDURE:

➢ Open Code Composer Studio, make sure the DSP kit is turned on.

➢ Start a new project using 'Project-new ' pull down menu, save it in a separate directory(c:\ti\myprojects) with name **cir conv.pjt.**

➢ Add the source files **Circular Convolution.C**
➢ to the project using 'Project→add files to project' pull down menu.

➢ Add the linker command file **hello.cmd** .
          (Path: c:\ti\tutorial\dsk6713\hello1\hello.cmd)

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

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

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

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

EXPERIMENT NO 12: 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

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

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>// included for trigonometric functions
void main()
{
        float y[16];//for 8 point DFT to store real & imaginary values (8x2)
        int x[4]={1,3,2,5}; //i/p real sequence(float is used for fractional values)
        float w;
        int n,k,k1,N=8,xlen=4;
        for(k=0;k<2*N; k=k+2)//real and imaginary values stored in consecutive locations so
                        incrementing by 2 each time
        {
                y[k]=0;         // initialize real part
                y[k+1]=0;       //initialize imaginary parts
                k1=k/2;         //actual k index
                for(n=0;n<xlen;n++)
                {
                        w= - 2*3.14*k1*n/N; //Calculate angle w
                        y[k]=y[k]+x[n]*cos(w); //Real part of DFT
                        y[k+1]=y[k+1]+x[n]*sin(w); // Imaginary part of DFT
                }
                printf("%f", y[k]); // Display real part
                if(y[k+1]<0) //check for negative value of imaginary part
                {
                        y[k+1]= -1*y[k+1];//negative value is made positive
                        printf("-j%f \n",y[k+1]);//displaying imaginary value with negative
                                                sign
                }
                else
```

```
            printf("+j%f \n",y[k+1]);
        }//end of for loop
    }//end of main
```

Result: **11.000000+j0.000000   -0.407551-j7.660230   -1.009554+j1.996801   2.424618-j3.646700**

-4.999950-j0.022297   2.396780+j3.673694   -0.971311-j2.009436   -0.460726+j7.633037

EXPERIMENT 13: IMPULSE RESPONSE OF 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

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

Program:

```c
#include <stdio.h>
#define Order  02
#define Len           10
  float y[Len]={0,0,0},sum;
main()
 {
      intj,k;
      float b[Order+1]={1, -.33334};
      float a[Order+1]={1, -0.75, 0.125};
     for(j=0;j<Len;j++)
     {
          sum=0;
              for(k=1;k<=Order;k++)
                 {
                        if((j-k)>=0)
                        sum=sum+(a[k]*y[j-k]);
                 }
              if(j<=Order)
                 {
                        y[j]=b[j]-sum;
                 }
                 else
                 {
                        y[j]=-sum;
                 }
         printf("Response[%d] = %f\n",j,y[j]);

     }
 }
```

Result:

Response[0] = 1.000000	Response[5] = 0.021158
Response[1] = 0.416660	Response[6] = 0.010498
Response[2] = 0.187495	Response[7] = 0.005228
Response[3] = 0.088539	Response[8] = 0.002609
Response[4] = 0.042967	Response[9] = 0.001303

EXPERIMENT NO 14: FIR FILTER

Aim: **Realization of an FIR filter (any type) to meet given specifications.**

     The input can be a signal from function generator / speech signal.

<u>**Pre-requisite:**</u> Input is given from the signal generator of amplitude 1 v p-p and

           Frequency > 20 Hz.

     <u>**TMS kit:**</u> Plug in the signal generator o/p to line in and line out of TMS to CRO

          Before compiling, Copy header files "dsk6713.h" and "dsk6713_aic23.h"

          from C:\CCStudio_v3.1\c6000\dsk6713\include and paste it in the current

          project folder.

   <u>**Source Code**</u>: **codec.c**

<u>Procedure for Real time Programs:</u>

1. Connect CRO to the Socket Provided for **LINE OUT**.
2. Connect a Signal Generator to the **LINE IN** Socket.
3. Switch on the Signal Generator with a sine wave of frequency 500 Hz. & Vp-p=1.5v
4. Now Switch on the DSK and Bring Up Code Composer Studio on the PC.
5. Create a new project with name **codec.pjt.**
6. From the **File** Menu ➔ **new** ➔ **DSP/BIOS Configuration** ➔ select

    "**dsk6713.cdb**" and save it as "**xyz.cdb**"
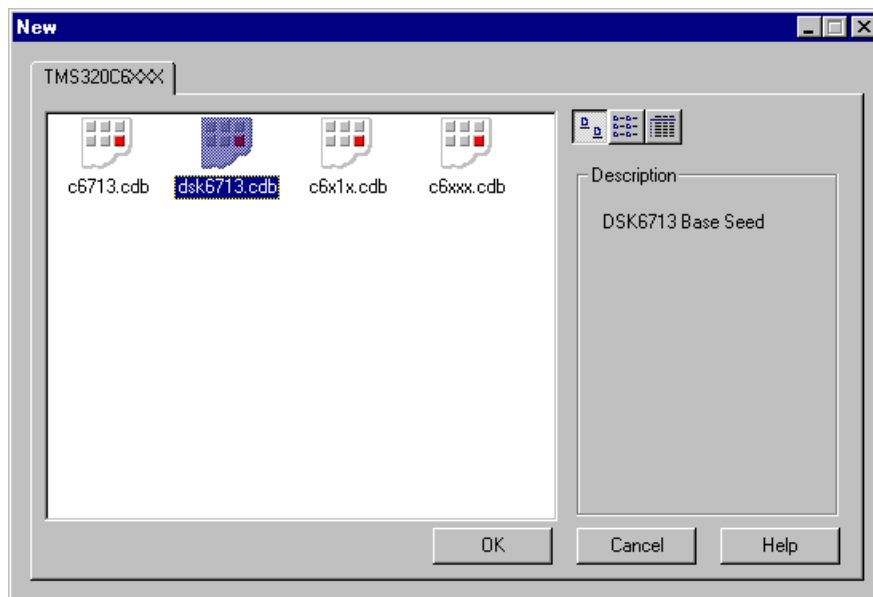
7. Add "xyz.cdb" to the current project.

---

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

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

10. Path → "C:\CCStudio\C6000\dsk6713\lib\dsk6713bsl.lib"

11. Copy files "dsk6713.h" and "dsk6713_aic23.h" from

12. C:\CCStudio\C6000\dsk6713\include and paste it in current project.

13. Build, Load and Run the program.

14. You can notice the input signal of 500 Hz. appearing on the CRO verifying the

codec configuration.

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

16. You can also vary the sampling frequency using the DSK6713_AIC23_setFreq

17. Function in the "codec.c" file and repeat the above steps.

Conclusion:

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

codec.c

```
#include "xyzcfg.h"
#include "dsk6713.h"
#include "dsk6713_aic23.h"

/* Codec configuration settings */

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


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

```
{
      DSK6713_AIC23_CodecHandle hCodec;
      int l_input, r_input,l_output, r_output;
      DSK6713_init();              // Initialize the board support library, must be called first
      hCodec = DSK6713_AIC23_openCodec(0, &config);      // Start the codec
      DSK6713_AIC23_setFreq(hCodec, 3);
while(1)
     {
      while (!DSK6713_AIC23_read(hCodec, &l_input)); //Read a sample to the left channel
      while (!DSK6713_AIC23_read(hCodec, &r_input)); // Read a sample to the right channel */
      while (!DSK6713_AIC23_write(hCodec, l_input)); // Send a sample to the left channel */
      while (!DSK6713_AIC23_write(hCodec, l_input)); // /* Send a sample to the right channel */
     }
   DSK6713_AIC23_closeCodec(hCodec); //    /* Close the codec */
}
```

Advance Discrete Time Filter Design (FIR)

Finite Impulse Response Filter

DESIGNING AN FIR FILTER:

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

> I.   Clearly specify the filter specifications.
> > Eg: Order   = 30;
> >   Sampling Rate   = 8000 samples/sec
> >   Cut off Freq.   = 400 Hz.

> II.   Compute the cut-off frequency W_c
> > Eg: W_c = 2*pie* f_c / F_s
> >   = 2*pie* 400/8000
> >   = 0.1*pie

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

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

USING MATLAB TO DETERMINE FILTER COEFFICIENTS :

Using FIR1 Function on Matlab

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

B(k) = B(N+2-k), k = 1,2,...,N+1.If Wn is a two-element vector, Wn = [W1 W2], FIR1 returns an order N bandpass filter with pass band $W1 < W < W2$.

B = FIR1(N,Wn,'high') designs a high pass filter.

B = FIR1 (N, Wn,'stop') is a band stop filter if Wn = [W1 W2].

If Wn is a multi-element vector, Wn = [W1 W2 W3 W4 W5 ... WN], FIR1 returns an order N multiband filter with bands $0 < W < W1$, $W1 < W < W2$, ..., $WN < W < 1$.

B = FIR1(N,Wn,'DC-1') makes the first band a pass band.

B = FIR1(N,Wn,'DC-0') makes the first band a stop band.

For filters with a passband near Fs/2, e.g., highpass and bandstop filters, N must be even. By default FIR1 uses a Hamming window. Other available windows, including Boxcar, Hanning, Bartlett, Blackman, Kaiser and Chebwin can be specified with an optional trailing argument.

For example, B = FIR1(N,Wn,kaiser(N+1,4)) uses a Kaiser window with beta=4.

B = FIR1(N,Wn,'high',chebwin(N+1,R)) uses a Chebyshev window.

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

B = FIR1(N,Wn,'high','noscale'), B = FIR1(N,Wn,wind,'noscale').

Note: We need to realize an advance FIR filter by implementing its difference equation as per the specifications. A direct form I implementation approach is taken. (The filter coefficients are taken as a_i as generated by the Matlab program.)

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

%FIR Low pass filters using rectangular window

% sampling rate – 8000   % order = 30;   % cutoff frequency – 1000

% b_rect=fir1(N, fc/(fs/2),boxcar(N+1))

```
b_rect=fir1(30, 1000/4000,boxcar(31));   % Rectangular
fid=fopen('FIR_lowpass_rectangular.txt','wt');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect);
fseek(fid,-1,0);
fclose(fid);
winopen('FIR_Lowpass_rectangular.txt');
```

Co-efficients: -0.015752,-0.023869,-0.018176,0.000000,0.021481,0.033416,0.026254,-0.000000,-0.033755,-0.055693, -0.047257,0.000000,0.078762,0.167080,0.236286,0.262448,0.236286,0.167080,0.078762,0.000000, -0.047257, -0.055693,-0.033755,-0.000000,0.026254,0.033416,0.021481,0.000000,-0.018176,-0.023869, -0.015752

%FIR Low pass filters using triangular window

% sampling rate – 8000   % order = 30;   % cutoff frequency – 1000

% b_rect=fir1(N, fc/(fs/2),bartlett(N+1))

    b_tri=fir1(30,1000/4000,bartlett(31));   %Triangular

    fid=fopen('FIR_lowpass_triangular.txt','wt');

    fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_tri);

    fseek(fid,-1,0);

    fclose(fid);

    winopen('FIR_Lowpass_triangular.txt');

Co-efficients: -0.000000,-0.001591,-0.002423,0.000000, 0.005728,0.011139,0.010502,-0.000000,-0.018003,-0.033416, -0.031505, 0.000000, 0.063010,0.144802,0.220534,0.262448,0.220534,0.144802,0.063010,0.000000, -0.031505, -0.033416,-0.018003,-0.000000, 0.010502,0.011139,0.005728,0.000000,-0.002423,-0.001591, -0.000000

%FIR Low pass filters using kaiser_window

% sampling rate – 8000   % order = 30;   % cutoff frequency – 1000

% b_rect=fir1(N, fc/(fs/2),kaiser(N+1,β))

    b_kai=fir1(30,1000/4000,kaiser(31,8));   %Kaisar [Where 8-->Beta Co-efficient]

    fid=fopen('FIR_lowpass_kaiser.txt','wt');

    fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_kai);

    fseek(fid,-1,0);

    fclose(fid);

    winopen('FIR_Lowpass_kaiser.txt');

co-efficients: -0.000035,-0.000234,-0.000454,0.000000,0.001933,0.004838,0.005671,-0.000000, -0.013596,-0.028462, -0.029370, 0.000000, 0.064504, 0.148863, 0.221349, 0.249983, 0.221349, 0.148863,0.064504,0.000000,-0.029370, -0.028462,-0.013596,-0.000000, 0.005671, 0.004838, 0.001933, 0.000000,-0.000454,-0.000234,-0.000035

Assignment: **1. cutoff frequency – 500**

                               2. cutoff frequency - 1500

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

% FIR High pass filters using rectangular, triangular and kaiser windows

% sampling rate – 8000;   %order = 30;   % cutoff frequency – 400

    b_recthigh=fir1(30,400/4000,'high',boxcar(31));

    fid=fopen('FIR_highpass_rectangular.txt','wt');

    fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_recthigh);

    fseek(fid,-1,0);

    fclose(fid);

    winopen('FIR_highpass_rectangular.txt');

<u>*Co-efficients:*</u> 0.021665,0.022076,0.020224,0.015918,0.009129,-0.000000,-0.011158,-0.023877,-0.037558,-0.051511, -0.064994,-0.077266,-0.087636,-0.095507,-0.100422,0.918834,-0.100422,-0.095507,-0.087636,-0.077266, -0.064994, -0.051511,-0.037558,-0.023877,-0.011158,-0.000000,0.009129,0.015918,0.020224,0.022076, 0.021665

% cutoff frequency – 800   b_trihigh=fir1(30,800/4000,'high',bartlett(31));

% cutoff frequency – 1200

  b_kaihigh=fir1(30,1200/4000,'high',kaiser(31,8));Where Kaiser(31,8)-->'8'defines the value of 'beta'.

FLOW CHART TO IMPLEMENT FIR FILTER:
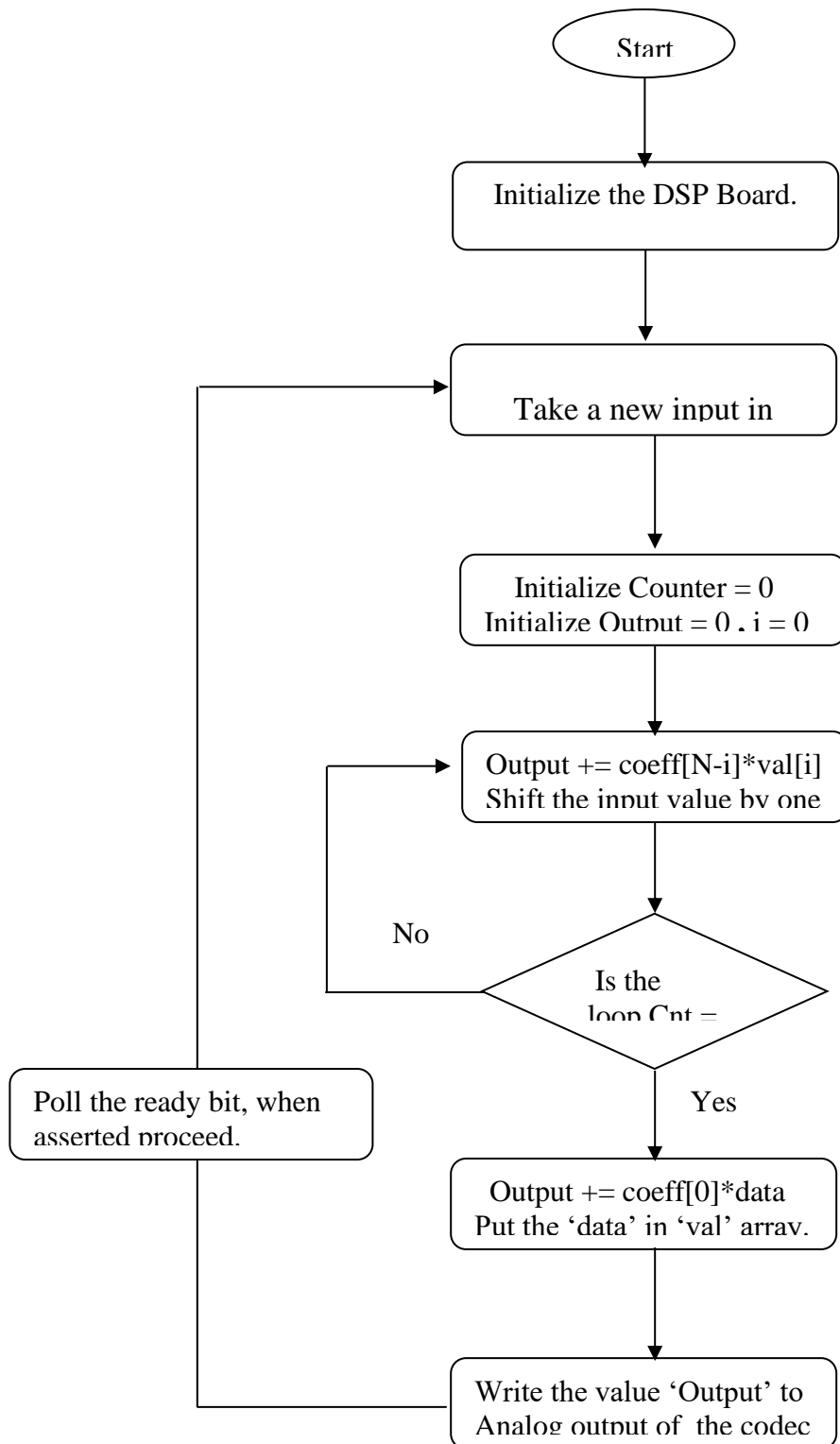
C PROGRAM TO IMPLEMENT FIR FILTER:

```
#include "filtercfg.h"              // header file for filter functions
#include "dsk6713.h"               // DSP board functions
#include "dsk6713_aic23.h"         //DSP kit I/O functions and register definitions
float filter_Coeff[] ={0.000000,-0.001591,-0.002423,0.000000,0.005728, 0.011139, 0.010502,
        -0.000000,-0.018003,-0.033416,-0.031505, 0.000000, 0.063010, 0.144802, 0.220534, 0.262448,
        0.220534, 0.144802, 0.063010, 0.000000, -0.031505,-0.033416,-0.018003,-0.000000, 0.010502,
        0.011139, 0.005728, 0.000000,-0.002423,-0.001591, 0.000000};
static short in_buffer[100];
     DSK6713_AIC23_Config config =
     { 0x0017, 0x0017, 0x00d8, 0x00d8, 0x0011, 0x0000, 0x0000, 0x0043, 0x0081,0x0001 };
void main()
     {
            DSK6713_AIC23_CodecHandle hCodec;
            Uint32 l_input, r_input,l_output, r_output;
            DSK6713_init();
            hCodec = DSK6713_AIC23_openCodec(0, &config);
            DSK6713_AIC23_setFreq(hCodec, 1);
            while(1)
              {
                   while (!DSK6713_AIC23_read(hCodec, &l_input));
                   while (!DSK6713_AIC23_read(hCodec, &r_input));
                   l_output=(Int16)FIR_FILTER(&filter_Coeff ,l_input);
                   r_output=l_output;
                   while (!DSK6713_AIC23_write(hCodec, l_output));
                   while (!DSK6713_AIC23_write(hCodec, r_output));
              }
            DSK6713_AIC23_closeCodec(hCodec);
     }
 signed int FIR_FILTER(float *h, signed int x)
     {
            int i=0;
            signed long output=0;
            in_buffer[0] = x;
            for(i=29;i>0;i--)
                    in_buffer[i] = in_buffer[i-1];
            for(i=0;i<31;i++)
                    output = output + h[i] * in_buffer[i];
            return(output);
     }
```

PROCEDURE:

- ➢ Switch on the DSP board.

- ➢ Open the Code Composer Studio.

- ➢ Create a new project

    Project → New (File Name. pjt , Eg: **FIR.pjt**)

- ➢ Initialize on board codec.

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

- ➢ Add the given above 'C' source file to the current project **(remove codec.c source file from the project if you have already added).**

- ➢ Connect the speaker jack to the input of the CRO.

- ➢ Build the program.

- ➢ Load the generated object file(*.out) on to Target board.

- ➢ Run the program

- ➢ Observe the waveform that appears on the CRO screen.

- ➢ Vary the frequency on function generator to see the response of filter.

Sample Viva Questions

1.   What is MATLAB? What is the latest version of MATLAB?
2.   What are the applications of MATLAB?
3.   What is the use of command 'legend'?
4.   Write the difference between subplot, plot and stem.
5.   Explain the statement=0:0.000005:0.05.what does colon (:) and semicolon (;) denotes.
6.   State sampling theorem.
7.   What do you mean by process of reconstruction?
8.   What do you mean Aliasing? What is the condition to avoid aliasing for sampling?
9.   What is a) Undersampling b) nyquist plot c) Oversampling.
10.   What is meant by Nyquist rate and Nyquist criteria?
11.   What is meant by linearity of a system and how it is related to scaling and superposition?
12.   What is impulse function?
13.   What is meant by impulse response?
14.   What is an energy signal? How to calculate energy of a signal?
15.   What is power signal? How to calculate power of a signal?
16.   Differentiate between even and odd signals.
17.   What is meant by causality?
18.   What is an LTI system?
19.   What is the difference between linear and circular convolution?
20.   What is the difference between continuous and discrete convolution?
21.   What is the advantage with sectioned convolution?.
22.   What do you mean by command "mod" and where it is used?
23.   What is the length of linear and circular convolutions if the two sequences are having the length n1 and n2?
24.   What does zero padding mean? Where we required this concept?
25.   Explain the concept of difference equation.
26.   What are difference equations and differential equations?
27.   What do you mean by filtic command, explain.
28.   What are Fourier series and Fourier transform?
29.   What are the advantages and special applications of Fourier transform, Fourier series, Z transform and Laplace transform?
30.   Differentiate between DTFT and DFT. Why it is advantageous to use DFT in computers rather than DTFT?
31.   Where DFT is used?
32.   What do you mean by built in function 'abs' and where it is used?
33.   What do you mean by phase spectrum and magnitude spectrum/ give comparison.
34.   Why we need FFT?.
35.   What is the difference between decimation in time (DIT FFT) & Decimation in frequency (DIFFFT) algorithms?
36.   What is meant by 'in-place' computation in DIF & DIF algorithms?
37.   Which properties are used in FFT to reduce no of computations?
38.   What is linear convolution?
39.   How to perform linear convolution using circular convolution?
40.   What do you mean by built in function 'fliplr' and where we need to use this.
41.   What is filter?
42.   Differentiate between IIR filters and FIR filters.
43.   What do you understand by linear phase response?
44.   To design all types of filters what would be the expected impulse response?
45.   What are the properties of FIR filter?.
46.   How the zeros in FIR filter is located?
47.   What is the difference between recursive & non-recursive systems?
48.   Write the difference equation for IIR system.

49.   What are the mapping techniques in IIR filter design? Discuss the advantage & disadvantages.
50.   What are IIR analog filters? What are the advantages & disadvantages of them?
51.   What is the disadvantage in impulse invariance method?
52.   What does warping effect mean? Where we found this effect? How can we eliminate warping effect
53.   Explain the pole mapping procedure of Impulse invariant & bilinear transformation method.
54.   For given same specification which difference we found in Butterworth & chebyshev filter.
55.   What is the difference between type I & type II chebyshev filters?.
56.   Where the poles are located for Butter worth & chedbyshev filters?
57.   What are the desirable characteristics of the window?
58.   What are the specifications required to design filter?
59.   What is meant by Gibbs Phenomenon? Where we found such type of effect in FIR filters?
60.   What is the procedure to design a digital Butterworth filter?
61.   Which window technique having less peak amplitude of side lobe as compared to all?
62.   What is the difference between Butterworth, Chebyshev I and Chebyshev II filters?
63.   What is window method? How you will design an FIR filter using window method.
64.   What are low-pass and band-pass filter and what is the difference between these two?
65.   Explain the command – N=ceil(6.6*pi/tb)
66.   Write down commonly used window function characteristics.
67.   What is the matlab command for Hamming window? Explain.
68.   What do you meant by cut-off frequency?
69.   What do you mean by command butter, cheby1?
70.   Explain the command in detail- [N,wc]=buttord(2*fp/fs,2*fstp/fs,rp,As)
71.   What is CCS? Explain in detail to execute a program using CCS.
72.   How to execute a program using 'dsk' and 'simulator'?
73.   Which Processor is used for hardware implementation? Explain the dsk, dsp kit.
74.   What is non real time processing?
75.   What is meant by real time processing?
76.   What is a Digital Signal Processor (DSP)?
77.   What is an Integrated Development Environment (IDE)
78.   Differentiate between RISC and CISC architectures.
79.   Differentiate between General purpose MPU(Micro Processor Unit) and DSP Processor
80.   Explain Von-Neumann and Harvard architectures.
81.   What are Line-in, Line-out, Mic-in, Mic-out?
82.   How many types of DSP processors are available in the market?
83.   TMS 320C6X, 'C' stands for what?
84.   What are the features of TMS 320C6X processor?
85.   What is meant by VLIW architecture? Why we required in DSP processor?
86.   How many functional units are in TMS 320C6X DSP processor?
87.   What are the applications for the TMS320 DSP's?