

Sr. No.	Title	CO Mapping	Date	Signature
1.	Write a program to convolve a signal and impulse response of a system.	CO-1		
2.	Write a program to plot harmonics of given periodic signal.	CO-1		
3.	Write a program to plot Pole-Zero of a given IIR filter.	CO-1, CO-2		
4.	To study various types of sampling methods	CO-3		
5.	To study Discrete Fourier transform and inverse Discrete Fourier Transform	CO-4		
6.	To study analog to Digital filter Transformation	CO-2		
7.	To study and Design Butterworth Low-pass, Band-pass and Stop filter	CO-2		
8.	Design FIR filter using windowing	CO-2		
9.	Write a program of LMS algorithm.	CO-5		
10.	To Study TMS320C6713 Processor	CO-5		

EXPERIMENT No.1

AIM: Write a program to convolve a signal and impulse response of a system.

Description:

- LIT system is completely characterized by its unit impulse response.
- If unit impulse is applied to the system then its output is denoted by $h(n)$ which is called impulse response of the system.
- $X(n)$ can be represented as sum of unit impulses this sum is known as convolution sum.
- $Y(n)=x(n)*h(n)$
- The different operations involved in convolution are:
 - 1) **Folding:** fold a sequence of $h(k)$.
 - 2) **Shifting:** time shifting of $h(-k)$.
 - 3) **Multiplication:** $x(k)*h(n-k)$
 - 4) **Summation:** addition of all product terms
- Convolution and De-convolution of the signals can done using MATLAB built-in function `Conv()` and `Deconv()`.
- A generalized Convolution computing code without using MATLAB built-in function `conv(x,h)` is given below:

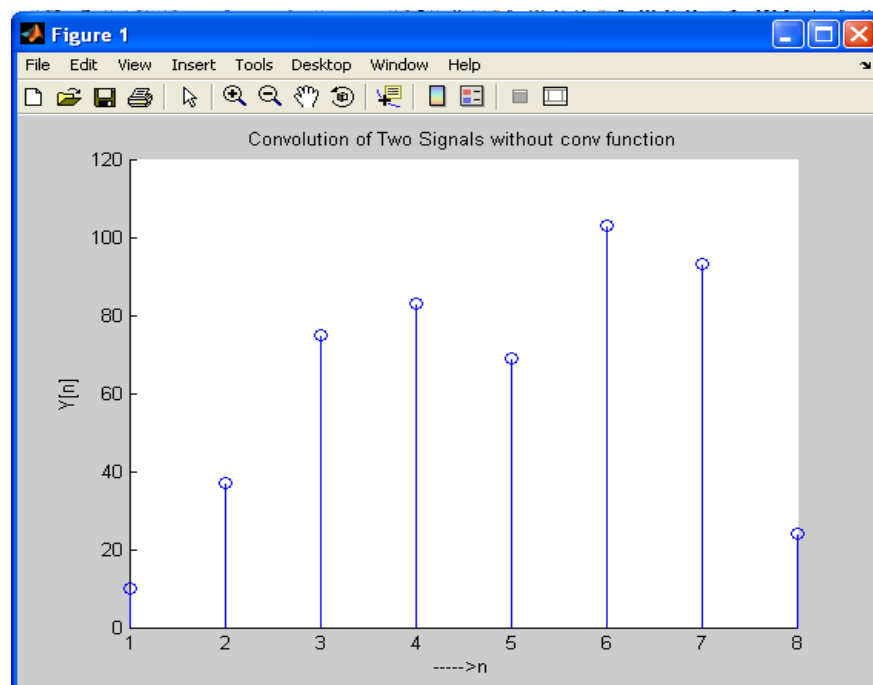
Answer:

```
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');
```

Output:

```
Command Window  
File Edit Debug Desktop Window Help  
  
Enter x: [2 5 9 3]  
  
x =  
  
     2     5     9     3  
  
Enter h: [5 6 0 7 8]  
  
h =  
  
     5     6     0     7     8  
  
Y =  
  
    10    37    75    83    69   103    93    24  
  
>>
```



EXPERIMENT No.2

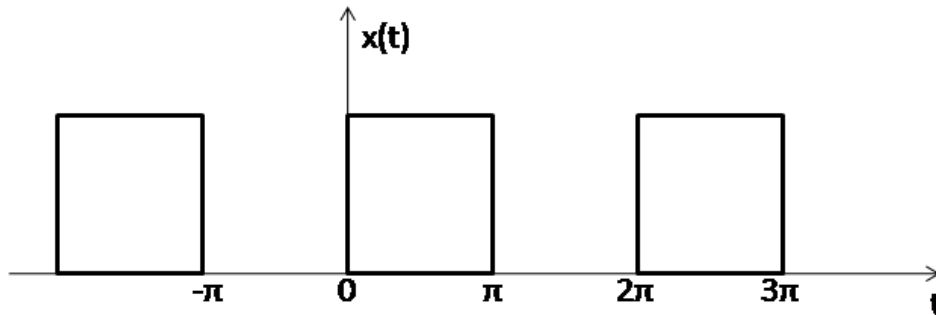
AIM: Write a program to plot harmonics of given periodic signal.

Description:

- Trigonometric form of Fourier Series:

$$x(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(n\omega_0 t) + b_n \sin(n\omega_0 t)$$

- $a_0 = \frac{2}{T} \int_0^T x(t) dt$
- $a_n = \frac{2}{T} \int_0^T x(t) \cos(n\omega_0 t) dt$
- $b_n = \frac{2}{T} \int_0^T x(t) \sin(n\omega_0 t) dt$



$$x(t) = \frac{1}{2} + \frac{2}{\pi} \sum_{n=1}^{\infty} \frac{1}{n} \sin(nt)$$

Answer:

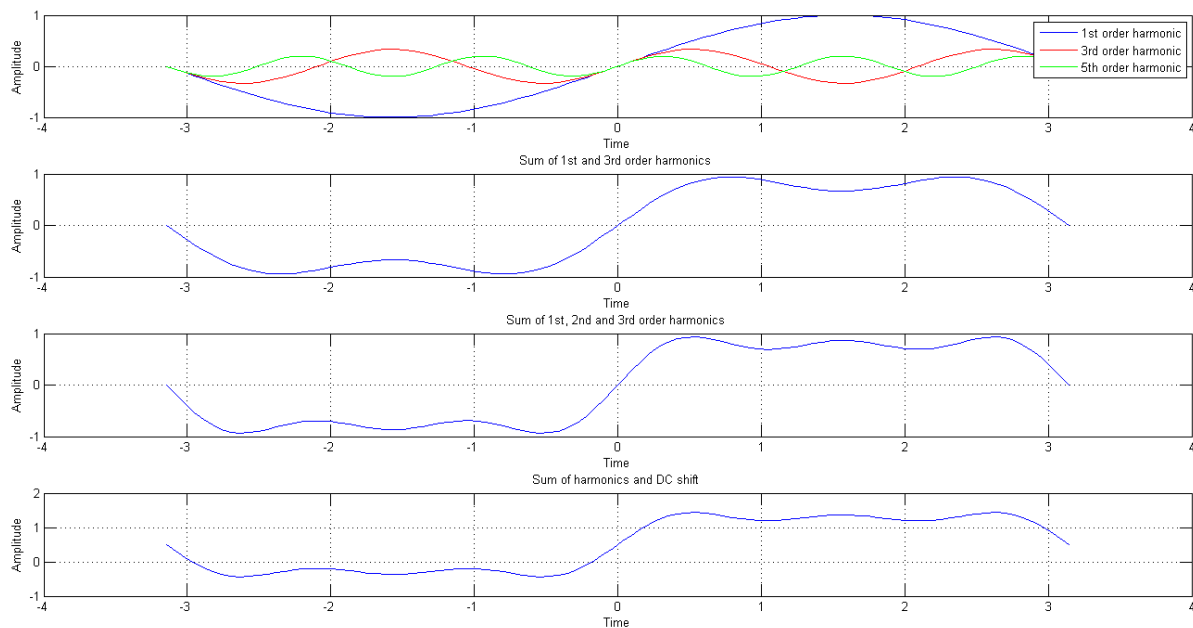
```
clc;
clear all;
close all;
t=linspace(-pi,pi);
x1=sin(t);
x2=(1/3)* sin(3*t);
x3=(1/5)* sin(5*t);
subplot(411)
plot(t,x1)
hold on;
plot(t,x2,'r')
```

```

hold on;
plot(t,x3,'g')
grid on;
xlabel('Time');
ylabel('Amplitude')
legend('1st order harmonic','3rd order harmonic','5th order harmonic')
subplot(412)
plot(t,(x1+x2))
grid on;
xlabel('Time');
ylabel('Amplitude')
title('Sum of 1st and 3rd order harmonics ')
subplot(413)
plot(t,(x1+x2+x3))
grid on;
xlabel('Time');
ylabel('Amplitude')
title('Sum of 1st, 2nd and 3rd order harmonics ')
subplot(414)
plot(t,(0.5+(x1+x2+x3)))
grid on;
xlabel('Time');
ylabel('Amplitude')
title('Sum of harmonics and DC shift ')

```

Output:



EXPERIMENT No.3

AIM: Write a program to plot Pole-Zero of a given IIR filter.

Description:

- Poles and Zeros of a transfer function are the frequencies for which the value of the denominator and numerator of transfer function becomes zero respectively. The values of the poles and the zeros of a system determine whether the system is stable, and how well the system performs. Control systems, in the simplest sense, can be designed simply by assigning specific values to the poles and zeros of the system.
- Physically realizable control systems must have a number of poles greater than or equal to the number of zeros. Systems that satisfy this relationship are called proper.

Answer:

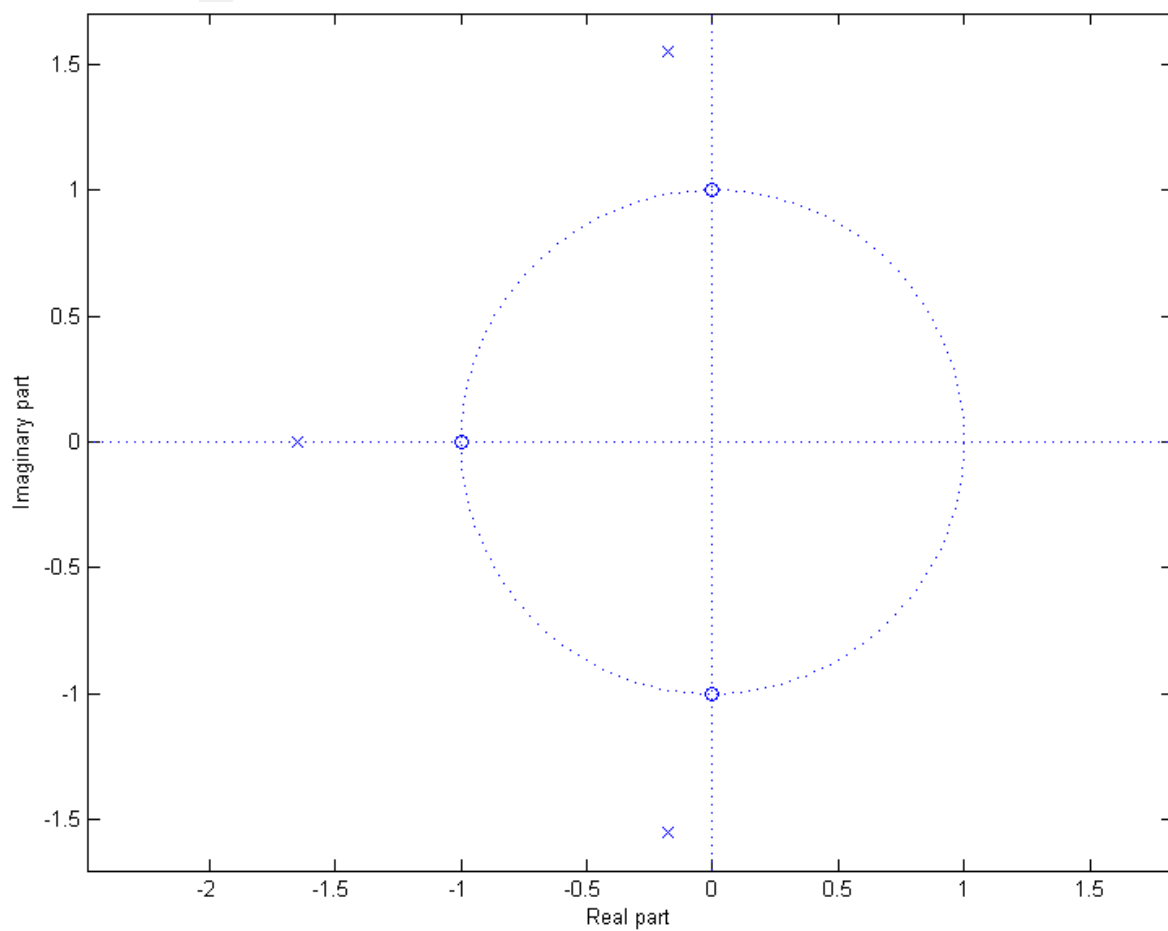
```
clc;
close all;
clear all;
xn=input('enter the sequence for Numerator ');
yn=input('enter the sequence for denominator ');
hn=tf(yn,xn); %gives the TF of polynomials yn , xn
[z,p,k] = tf2zp(yn,xn) ; % it converts TF filter parameters to
pole-zeros gain form
disp('zeros are ')
disp(z); % it display an array
disp('poles are ')
disp(p);
zplane(z,p);
xlabel('Real part');
ylabel('Imaginary part');
```

Output:

Command Window

```
enter the sequence for Numerator [1 1 1 1]
enter the sequence for denominator [1 2 3 4]
zeros are
  -1.0000 + 0.0000i
   0.0000 + 1.0000i
   0.0000 - 1.0000i

poles are
  -1.6506 + 0.0000i
  -0.1747 + 1.5469i
  -0.1747 - 1.5469i
```



EXPERIMENT No.4

AIM: To study various types of sampling methods.

- 1) Up sampling
- 2) Down sampling
- 3) Re-sampling

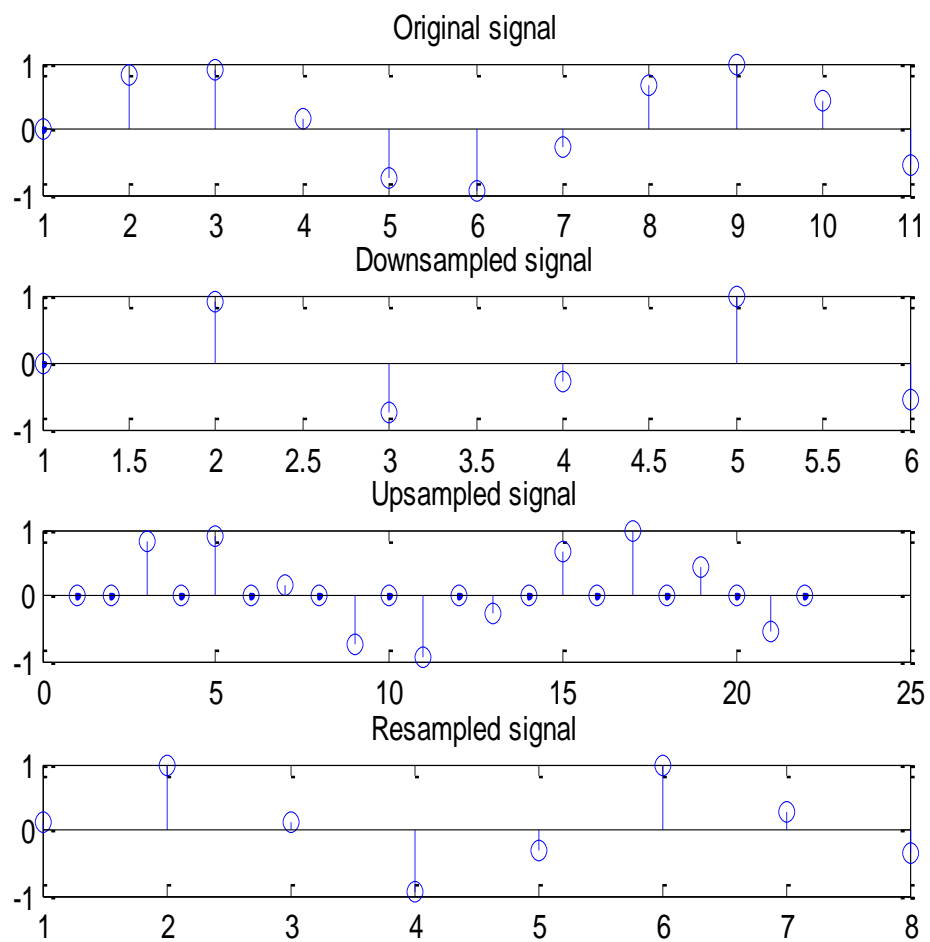
Description:

- As the name indicates up sampling and down sampling are basically time scaling operations.
- $Y(n) = X(2n)$ is example of down sampling where signal is compressed or scale is reduced.
- $Y(n) = X(n/2)$ is example of up sampling where signal is expanded or scale is increased.
- Up sampling and down sampling can also be used for amplitude scaling where up sampling means amplification and down sampling means attenuation.
- Re-sampling means assign a new sampling rate by rational factor.

Answer:

```
clc
closeall;
clearall;
n=0:10;
y=sin(n);
subplot(4,1,1);stem(y)
title('Original signal')
y1=downsample(y,2);
subplot(4,1,2);stem(y1)
title('Downsampled signal')
y2=upsample(y,2);
subplot(4,1,3);stem(y2)
title('Upsampled signal')
y3=resample(y,2,3);
subplot(4,1,4);stem(y3)
title('Resampled signal')
```


Output:



EXPERIMENT No.5

AIM: To Study Discrete Fourier Transform and Inverse Discrete Fourier Transform.

- 1) $x(n)=\{0,1,2,3\}$. Find 4-point DFT of this sequence. Perform this by developing a function for the same in MATLAB.
- 2) Verify that the output in Q 1 is the DFT of the input by finding out its inverse Fourier transform. Develop your function to find the inverse DFT.

Description:

- The discrete Fourier transform (DFT) is itself a sequence and it corresponds to samples, equally spaced in frequency, of the Fourier transform of the signal. A discrete transform is a transform whose input and output values are discrete samples, making it convenient for computer manipulation.
- There are two principal reasons for using this form of the transform.
 - 1) The input and output of the DFT are both discrete, which makes it convenient for computer manipulations.
 - 2) There is a fast algorithm for computing the DFT known as the Fast Fourier transform (FFT).
- The equation for calculating the k^{th} sample of the output DFT $X(k)$ of the input sequence $x(n)$ with N samples is

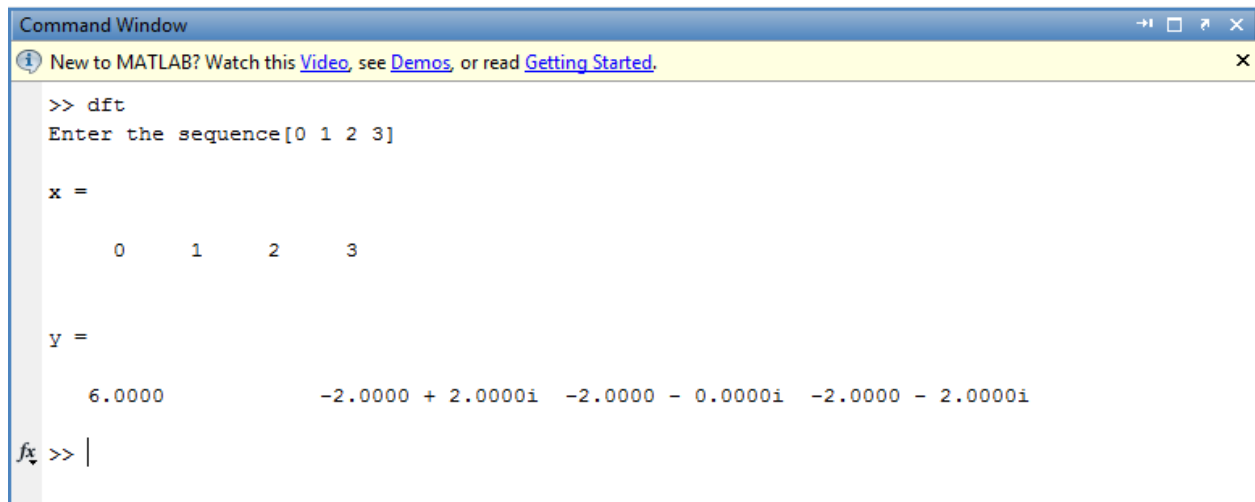
$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}$$

Answer:

1)DFT

```
clc;
clear all;
close all;
x=input('Enter the sequence')
N=length(x);
for k=1:N
    y(k)=0;
    for n=1:N
        y(k)=y(k)+x(n)*exp((-2*pi*j*(n-1)*(k-1))/N);
    end
end
```

Output:



A screenshot of the MATLAB Command Window. The window title is "Command Window". Below the title bar is a yellow banner with an information icon and the text "New to MATLAB? Watch this [Video](#), see [Demos](#), or read [Getting Started](#)." Below the banner, the command prompt shows the following sequence of commands and outputs:

```
>> dft
Enter the sequence[0 1 2 3]

x =

    0    1    2    3

y =

    6.0000    -2.0000 + 2.0000i    -2.0000 - 0.0000i    -2.0000 - 2.0000i

fx >> |
```

2)IDFT

```
clc;
close all;
clear all;
y=input('enter a sequence')
N=length(y);
for n=1:N
x(n)=0;
for k=1:N
x(n)=x(n)+y(k)*exp((2*j*pi*(k-1)*(n-1))/N);
end
end
z=x/N;
```

Output:

```
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

>> idft
enter a sequence[6 -2+2j -2 -2-2j]

y =

    6.0000    -2.0000 + 2.0000i    -2.0000    -2.0000 - 2.0000i

>> z

z =

    0    1.0000 + 0.0000i    2.0000 - 0.0000i    3.0000 - 0.0000i

fx >> |
```

EXPERIMENT No.6

AIM: To Study Analog to Digital Filter Transformation.

- 1) Use **impinvar** to perform analog to digital filter transformation of

$$\frac{s + 0.2}{(s + 0.2)^2 + 9}$$

Take T=1s.

- 2) Use bilinear to perform analog to digital transformation of

$$\frac{0.0476(1 + z^{-1})^2}{(1 - 0.9048z^{-1})^2}$$

Take T_s = 0.1s.

Description:

- The classic IIR filter design technique includes the following steps.
 - 1) Find a high pass filter with cutoff frequency of 1 and translate this prototype filter to the desired band configuration.
 - 2) Transform the filter to the digital domain.
 - 3) Discretize the filter
- Matlab functions can be used for filter designing are as below:
 - 1) **Impinvar[by, az] = impinvar (b, a, fs)** creates a digital filter with numerator and denominator coefficients bz and az, respectively, whose impulse response is equal to the impulse response of the analog filter with coefficients b and a, scaled by 1/fs. If we leave out the argument fs, or specify fs as the empty vector [], it takes the default value of 1 Hz.
 - 2) **Bilinear -[numd,dend] = bilinear(num,den,fs)** converts an s-domain transfer function given by num and den to a discrete equivalent. Row vectors num and den specify the coefficients of the numerator and denominator, respectively, in descending powers of s. fs is the sampling frequency in hertz. *Bilinear* returns the discrete equivalent in row vectors numd and dend in descending powers of z (ascending powers of z⁻¹).

Answer:

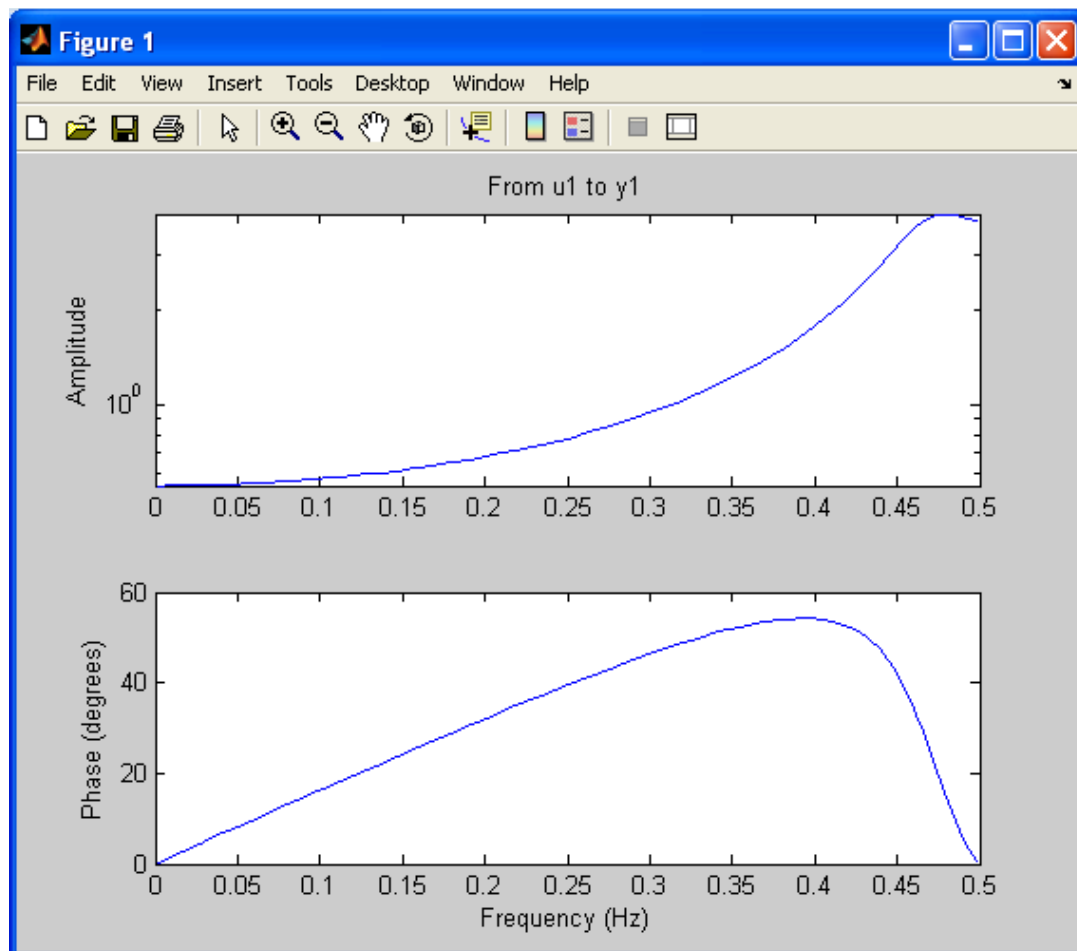
```
1) z = [1 0.2];  
p = [1 0.4 9.04];  
[num den]=impinvar(z,p,1);  
sys=filt(num,den,1)  
ffplot(sys)
```

Output:

```
Command Window
File Edit Debug Desktop Window Help

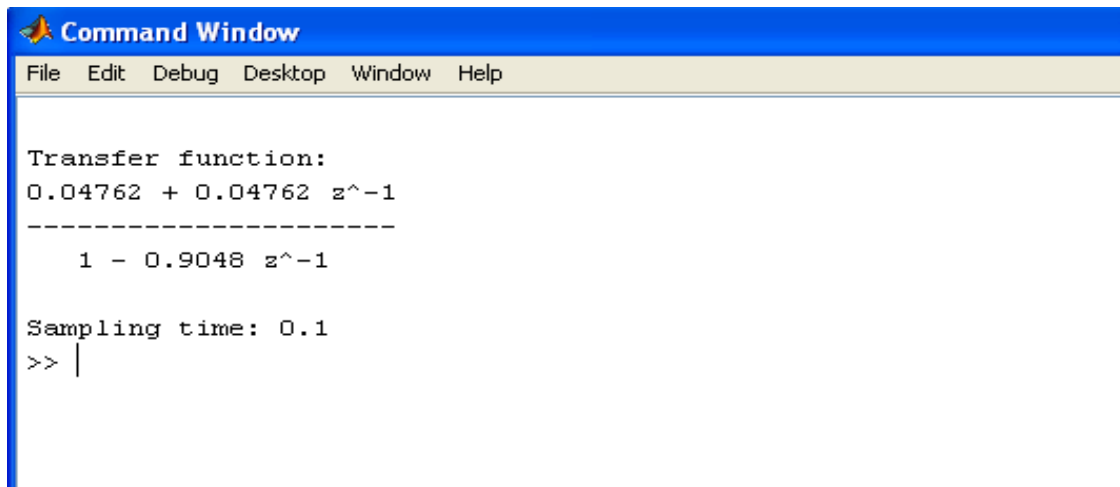
Transfer function:
      1 + 0.8105 z^-1
-----
1 + 1.621 z^-1 + 0.6703 z^-2

Sampling time: 1
>> |
```

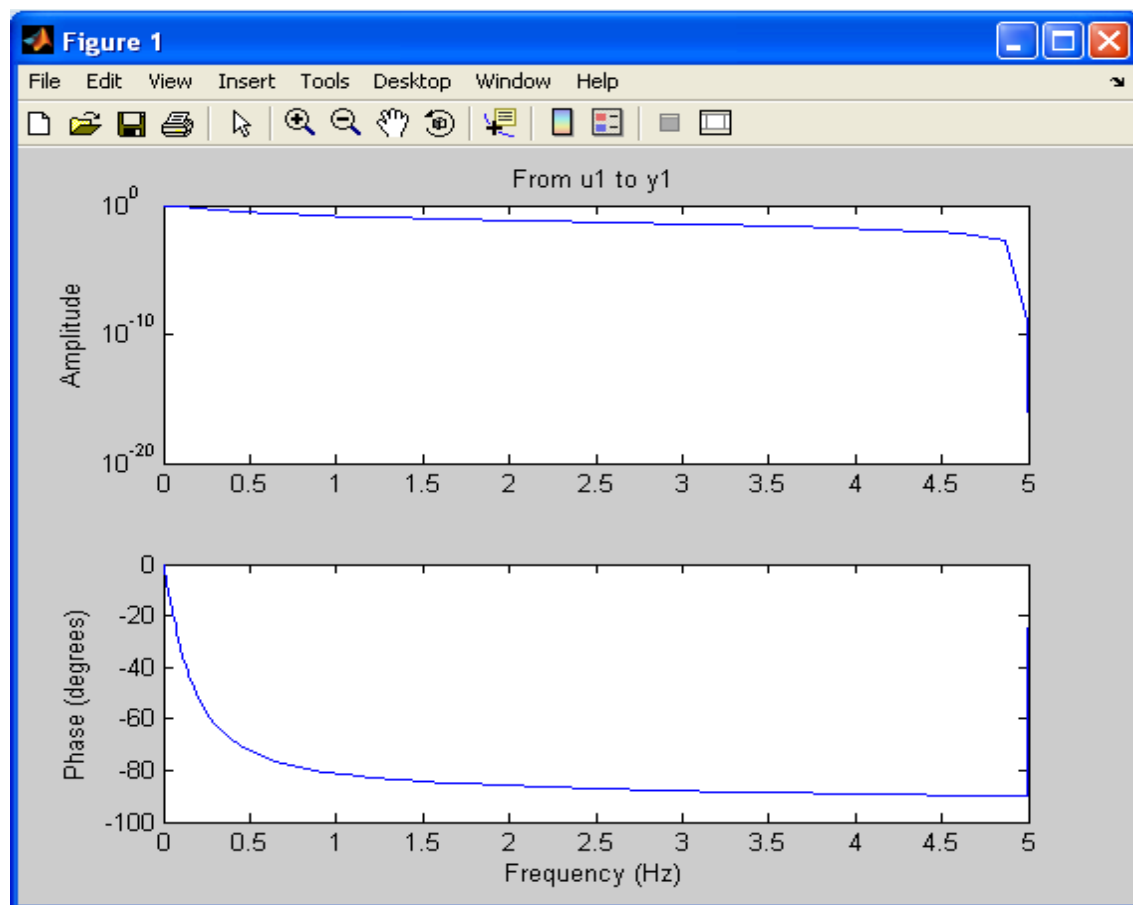


```
2) z=[1];  
p=[1 1];  
[num den] = bilinear(z,p,10);  
sys=filt(num,den,0.1)  
ffplot(sys)
```

Output



```
Command Window  
File Edit Debug Desktop Window Help  
  
Transfer function:  
0.04762 + 0.04762 z^-1  
-----  
1 - 0.9048 z^-1  
  
Sampling time: 0.1  
>> |
```



EXPERIMENT-7

Aim: Design Digital IIR Butterworth filter.

Description:

There are four types of IIR (Infinite impulse response) filters that can be implemented.

- LPF: Low pass filter allows low frequency to pass only. Frequency after cut of frequency will be blocked.
- HPF: High pass filter allows high frequency to pass after certain cutoff frequency.
- BPF: Band pass filter will pass certain band of frequency only
- BSF: Band stop filter will block certain band of frequency only.

Functions used:

1. Butter()

- Design Butterworth IIR digital filters using the specifications in a filter design object.

2. Buttdord()

- Calculate the order and cutoff frequency for a Butterworth filter.

“**Buttdord**” calculates the minimum order of a digital or analog Butterworth filter required to meet a set of filter design specifications.

Answer

```
Clc;
Clear all;
Close all;

%LOW PASS FILTER

rp=input ('Enter Passband Ripple: ');
rs=input ('Enter Stopband Ripple: ');
fs=input ('Enter Sampling Freq : ');
disp(' ');
disp('FOR LOW PASS FILTER');

wp=input ('Enter PassbandFreq : ');
ws=input ('Enter StopbandFreq : ');

w1=2*wp/fs;
w2=2*ws/fs;
```

```

[n,wn]=buttord(w1,w2,rp,rs);
[b,a]=butter(n,wn);

w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure(1);
subplot(2,1,1);
plot(om/pi,m);
xlabel('NormalisedFreq');
ylabel('Gain in dB');
%plot(om/pi,m);
subplot(2,1,2);
plot(om/pi,an);
xlabel('NormalisedFreq');
ylabel('Phase in Radians');

%HIGH PASS FILTER
disp(' ');
disp('FOR HIGH PASS FILTER');
wp=input('Enter PassbandFreq : ');
ws=input('Enter StopbandFreq : ');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs);
[b,a]=butter(n,wn,'high');

w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure(2);
subplot(2,1,1);
plot(om/pi,m);
xlabel('NormalisedFreq');
ylabel('Gain in dB');
%plot(om/pi,m);
subplot(2,1,2);
plot(om/pi,an);
xlabel('NormalisedFreq');
ylabel('Phase in Radians');

%BAND PASS FILTER

disp(' ');

```

```

disp('FOR BAND PASS FILTER');
wp=input('Enter PassbandFreq : ');
ws=input('Enter StopbandFreq : ');
w1=2*wp/fs;
w2=2*ws/fs;
[n]=buttord(w1,w2,rp,rs);
wn=[w1 w2];
[b,a]=butter(n,wn,'bandpass');
w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure(3);
subplot(2,1,1);
plot(om/pi,m);
xlabel('NormalisedFreq');
ylabel('Gain in dB');
subplot(2,1,2);
plot(om/pi,an);
xlabel('NormalisedFreq');
ylabel('Phase in Radians');

```

%BAND STOP FILTER

```

disp(' ');
disp('FOR BAND STOP FILTER');
wp=input('Enter PassbandFreq : ');
ws=input('Enter StopbandFreq : ');
w1=2*wp/fs;
w2=2*ws/fs;
[n]=buttord(w1,w2,rp,rs);
wn=[w1 w2];
[b,a]=butter(n,wn,'stop');
w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure(4);
subplot(2,1,1);
plot(om/pi,m);
xlabel('NormalisedFreq');
ylabel('Gain in dB');
%plot(om/pi,m);
subplot(2,1,2);
plot(om/pi,an);
xlabel('NormalisedFreq');
ylabel('Phase in Radians');

```

Output :

```
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

Enter Passband Ripple : 0.01
Enter Stopband Ripple : 0.1
Enter Sampling Freq : 0.5

FOR LOW PASS FILTER
Enter Passband Freq : 0.1
Enter Stopband Freq : 0.2

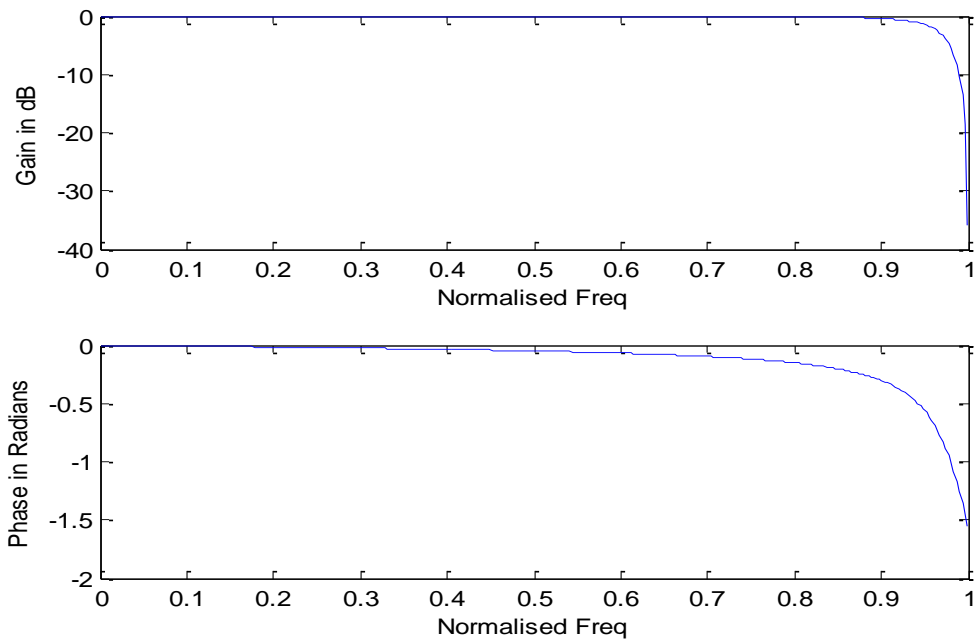
FOR HIGH PASS FILTER
Enter Passband Freq : 0.2
Enter Stopband Freq : 0.1

FOR BAND PASS FILTER
Enter Passband Freq : 0.2
Enter Stopband Freq : 0.1

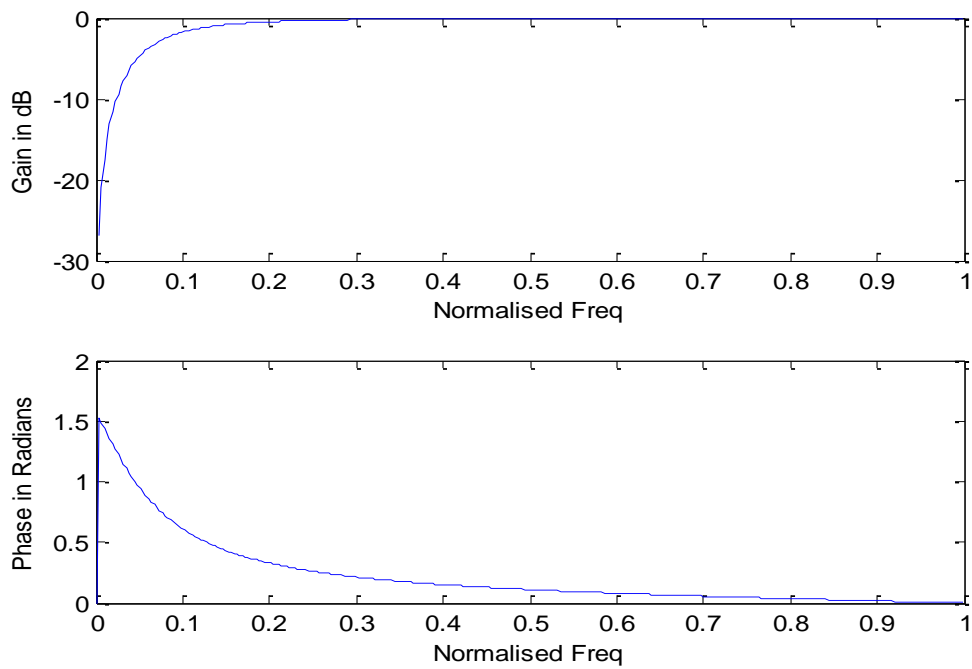
FOR BAND STOP FILTER
Enter Passband Freq : 0.1
Enter Stopband Freq : 0.2

fx >>
```

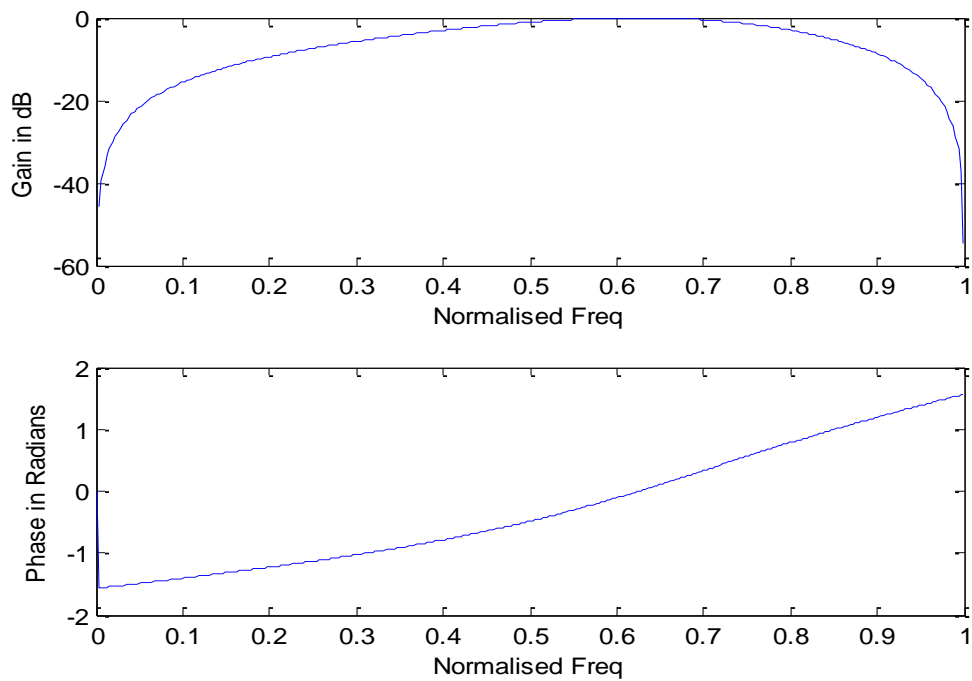
Low Pass:



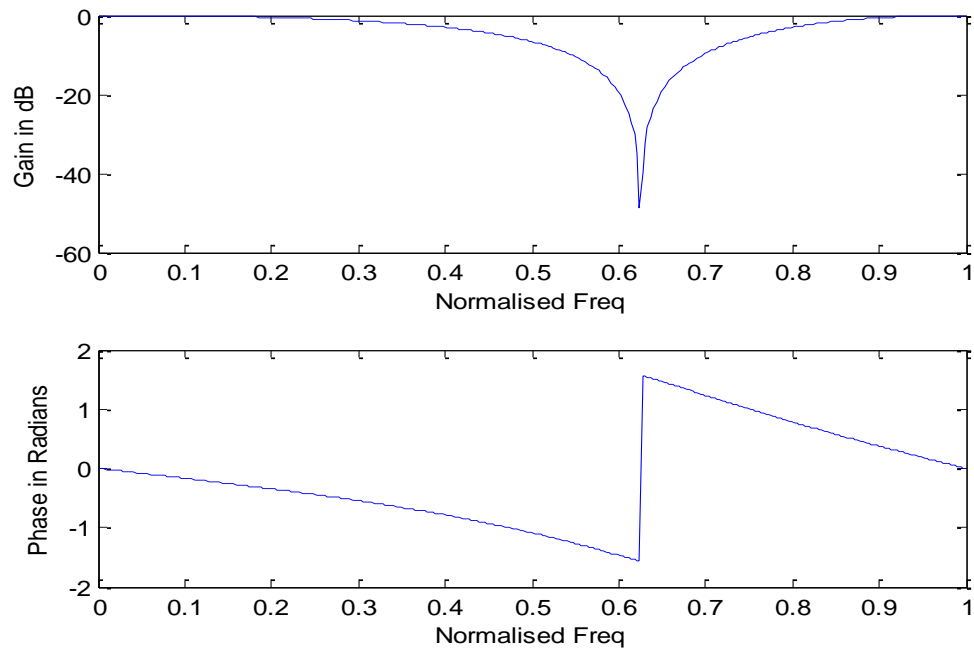
High Pass:



Band Pass:



Band Stop:



EXPERIMENT-8

Aim: FIR filter design using windowing.

Description:

- FIR stands for finite impulse response.
- It is non- recursive system.
- Special types of windows are used to get FIR.
 - Rectangular window
 - Hamming window
 - Hanning window
 - Kaiser window
 - Bartlett window

Functions used:

- 1) `fir1()`:
 - Design a window-based finite impulse response filter.
 - `fir1` implements the classical method of windowed linear-phase FIR digital filter design [1].
 - It designs filters in standard lowpass, highpass, bandpass, and bandstop configurations.
- 2) `freqz()`:
 - Compute the frequency response of discrete-time filters, adaptive filters, and multirate filters.
 - For adaptive filters, `freqz` returns the instantaneous frequency response based on the current filter coefficients.
- 3) `Bartlett()`.
 - `w = bartlett(n)` returns an n-point Bartlett window in the column vector `w`, where `n` must be a positive integer.
- 4) `Kaiser()`:
 - `w = kaiser(n,beta)` returns an n-point Kaiser ($I_0 - \sinh$) window in the column vector `w`.
 - `beta` is the Kaiser window parameter that affects the sidelobe attenuation of the Fourier transform of the window.
 - The default value for `beta` is 0.5.

Answer:

```
%FIR FILTER DESIGN
Clc;
Clear all;
Close all;
pr=0.05;sr=0.04;
pf=1500;sf=2000;
f=9000;
wp=2*pf/f;ws=2*sf/f;

%LOW PASS FILTER
N=(-20*log10(sqrt(pr*sr))-13)/(14.6*(sf-pf)/f);
N=ceil(N);

%RECTANGULAR WINDOW
y=boxcar(N);
b=fir1(N-1,wp,y);
figure(1);
freqz(b,1,256);
title('Rectangular Window');

%BARTLETT WINDOW
y=bartlett(N);
b=fir1(N-1,wp,y);
figure(2);
freqz(b,1,256);
title('Bartlett Window');

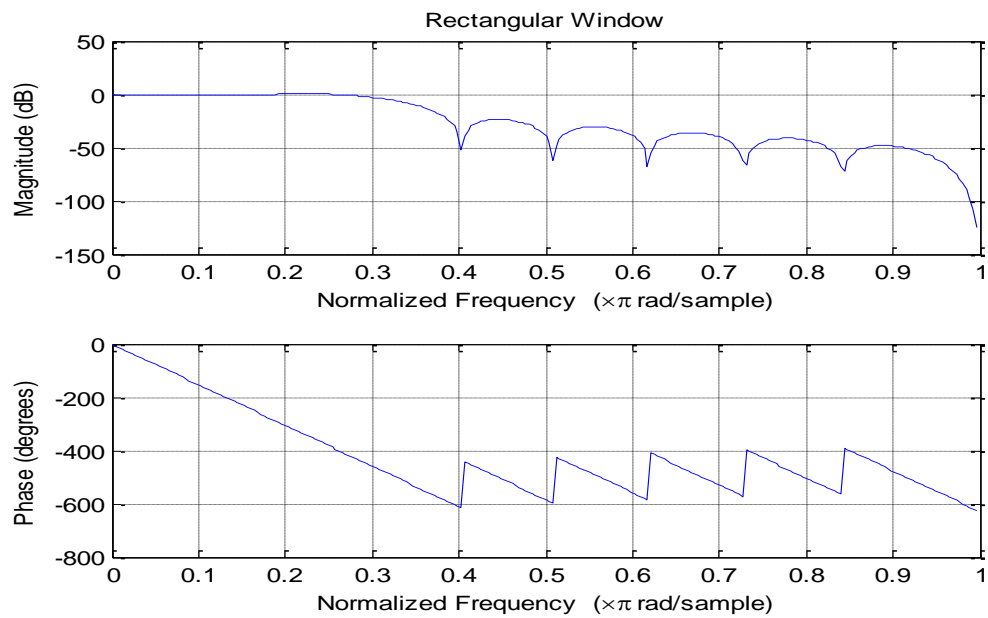
%HAMMING WINDOW
y=hamming(N);
b=fir1(N-1,wp,y);
figure(3);
freqz(b,1,256);
title('Hamming Window');

%HANNING WINDOW
y=hanning(N);
b=fir1(N-1,wp,y);
figure(4);
freqz(b,1,256);
title('Hanning Window');

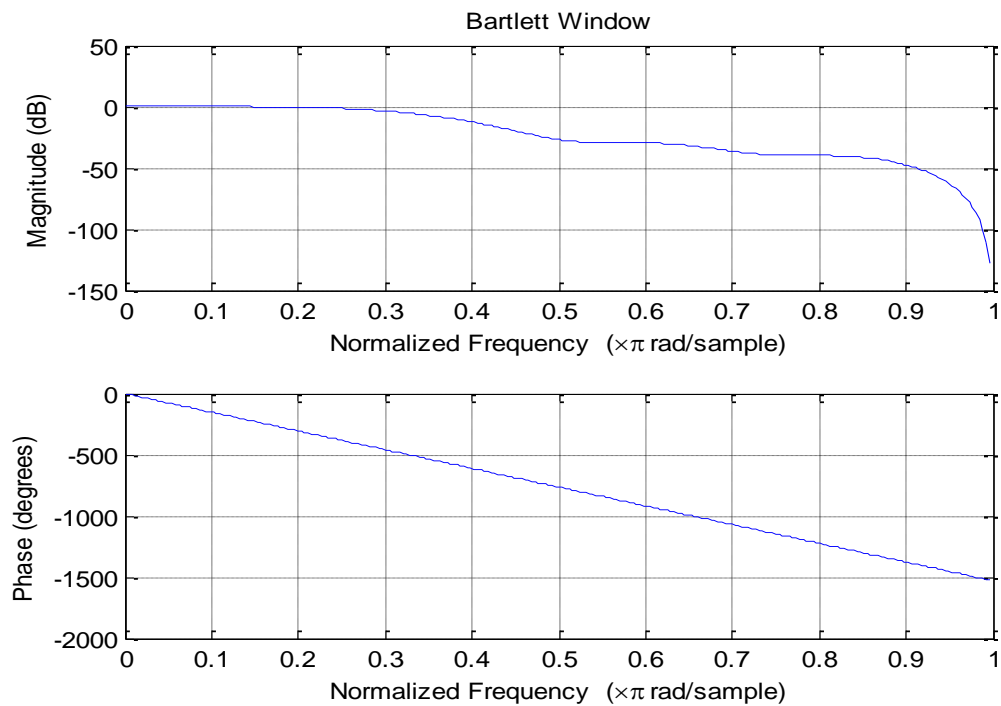
%KAISER WINDOW
beta=5.8;
y=kaiser(N,beta);
b=fir1(N-1,wp,y);
figure(5);freqz(b,1,256);
title('KaiserWindow');
```


Output:

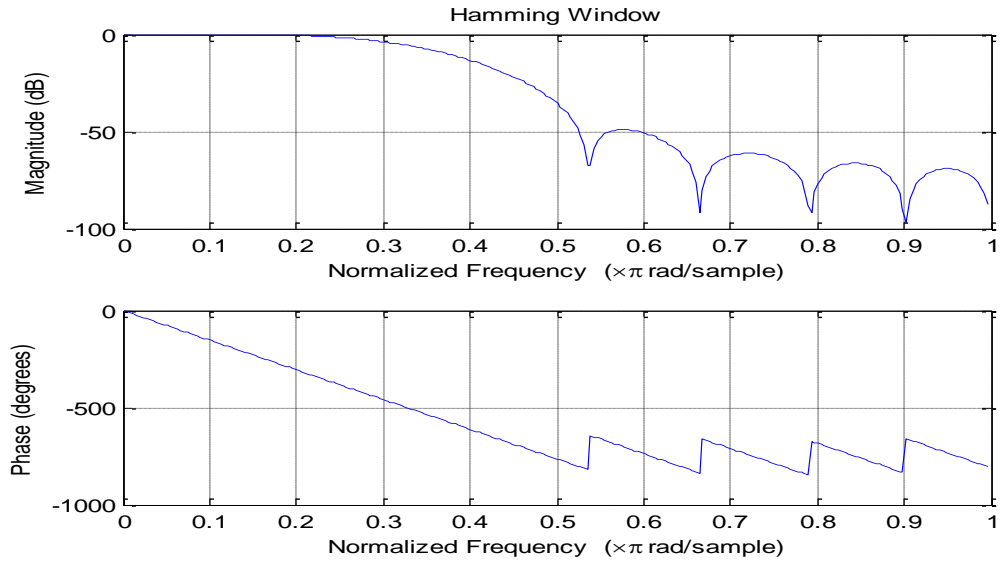
Rectangular Window:



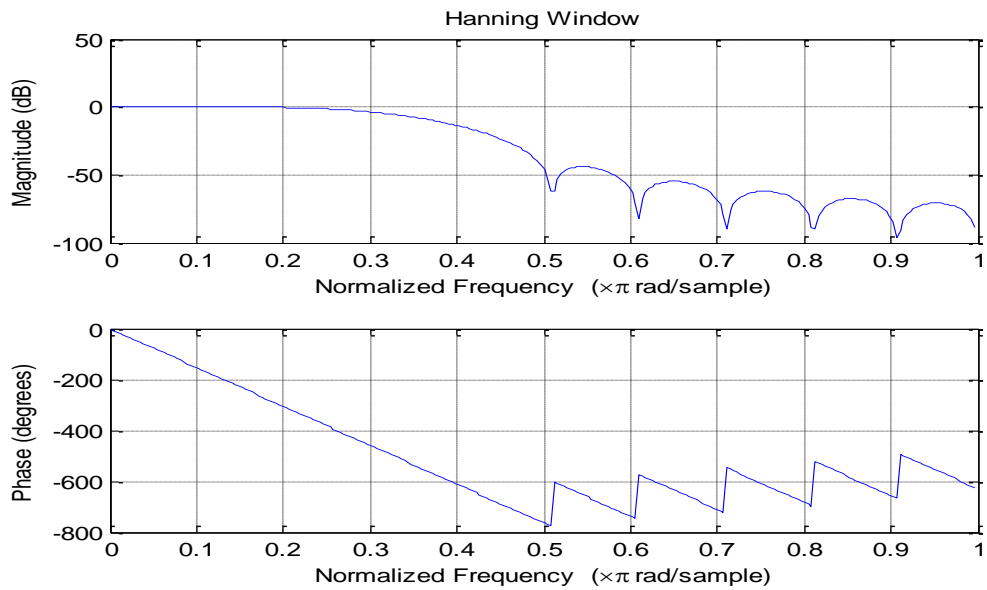
Bartlett Window:



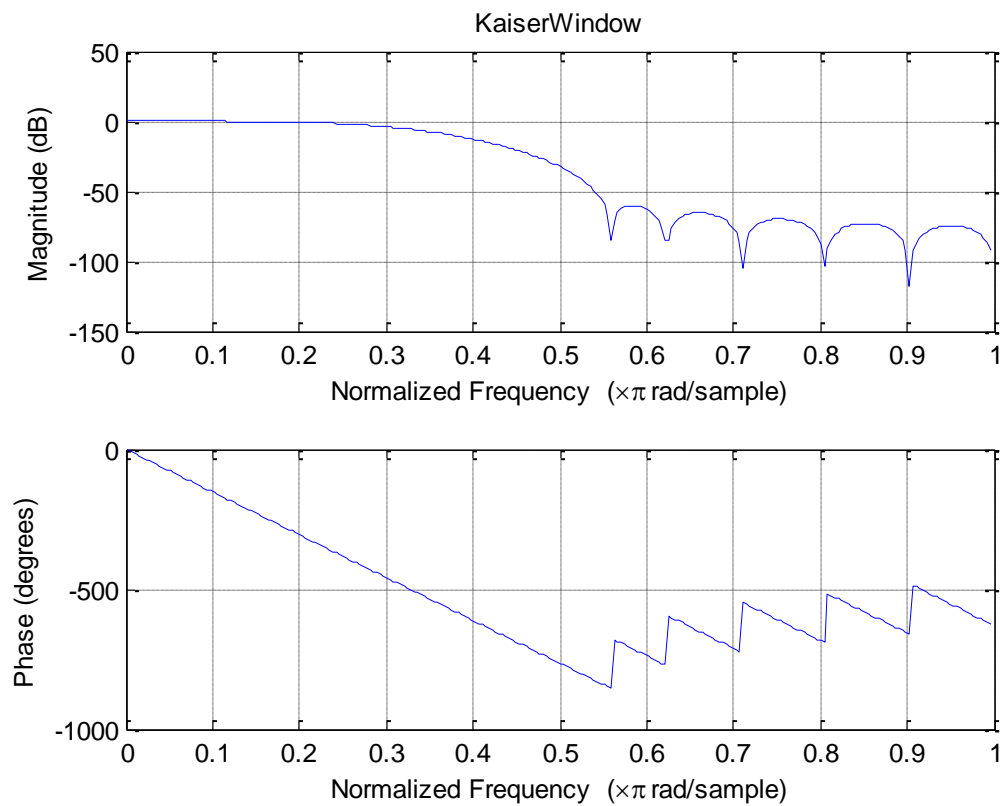
Hamming Window:



Hanning Window:



Kaiser Window:



EXPERIMENT No.9

AIM: Write a program of LMS algorithm.

Description:

The LMS algorithm for a p^{th} order algorithm can be summarized as:

Parameters: p = filter order

μ = step size

Initialization: $\hat{h}(0) = \text{zeros}(p)$

Computation: For $n=0, 1, 2 \dots$

$$x(n) = [x(n), x(n-1), \dots, x(n-p+1)]^T$$

$$e(n) = d(n) - \hat{h}^H(n) x(n)$$

$$\hat{h}(n+1) = \hat{h}(n) + \mu e^*(n)x(n)$$

Answer:

```
clc;
clear all;
close all;

%channel system order
sysorder = 5 ;
% Number of system points
N=2000;
inp = randn(N,1);
n = randn(N,1);
[b,a] = butter(2,0.25);
Gz = tf(b,a,-1);

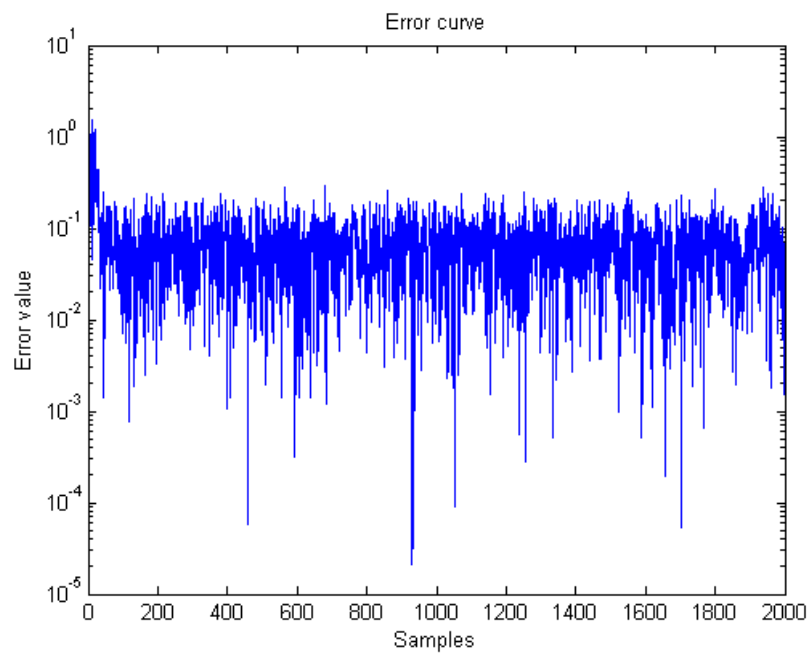
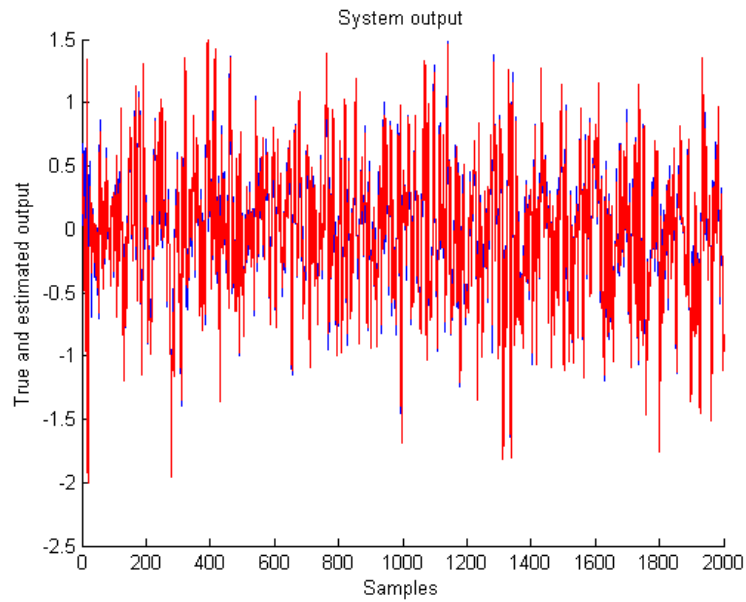
h= [0.0976;
    0.2873;
    0.3360;
    0.2210;
    0.0964;];
y = lsim(Gz,inp);
%add some noise
n = n * std(y) / (10*std(n));
d = y + n;
totallength=size(d,1);
%Take 60 points for training
```

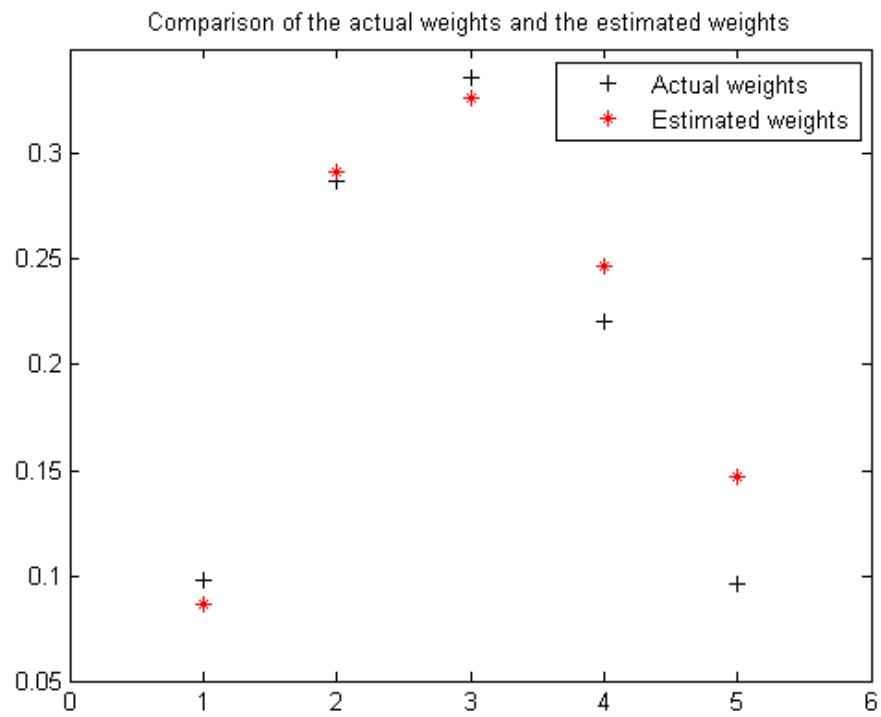
```

N=60 ;
%begin of algorithm
w = zeros ( sysorder , 1 ) ;
for n = sysorder : N
    u = inp(n:-1:n-sysorder+1) ;
    y(n)= w' * u;
    e(n) = d(n) - y(n) ;
% Start with big mu for speeding the convergence then slow down
to reach the correct weights
    if n < 20
        mu=0.32;
    else
        mu=0.15;
    end
    w = w + mu * u * e(n) ;
end
%check of results
for n = N+1 : totallength
    u = inp(n:-1:n-sysorder+1) ;
    y(n) = w' * u ;
    e(n) = d(n) - y(n) ;
end
hold on
plot(d)
plot(y,'r');
title('System output') ;
xlabel('Samples')
ylabel('True and estimated output')
figure
semilogy((abs(e))) ;
title('Error curve') ;
xlabel('Samples')
ylabel('Error value')
figure
plot(h, 'k+')
hold on
plot(w, 'r*')
legend('Actual weights','Estimated weights')
title('Comparison of the actual weights and the estimated
weights') ;
axis([0 6 0.05 0.35])

```

Output:



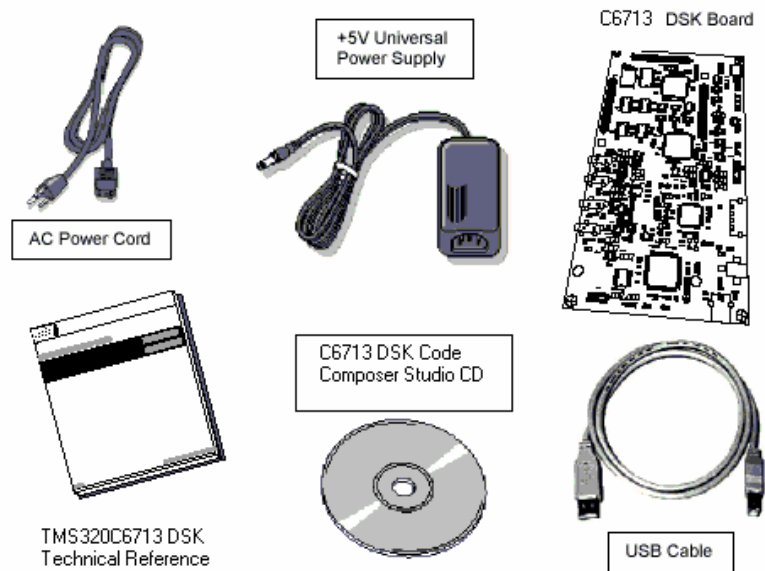


EXPERIMENT 10

Aim: To study about TMS320C6713 DSK processor.

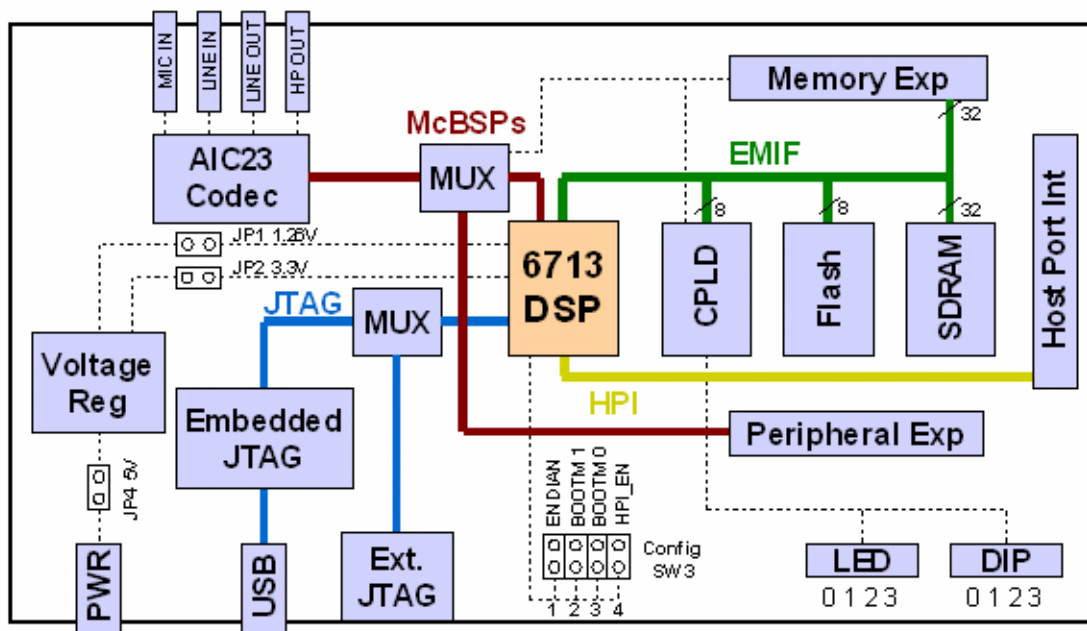
Package Contents

- 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.



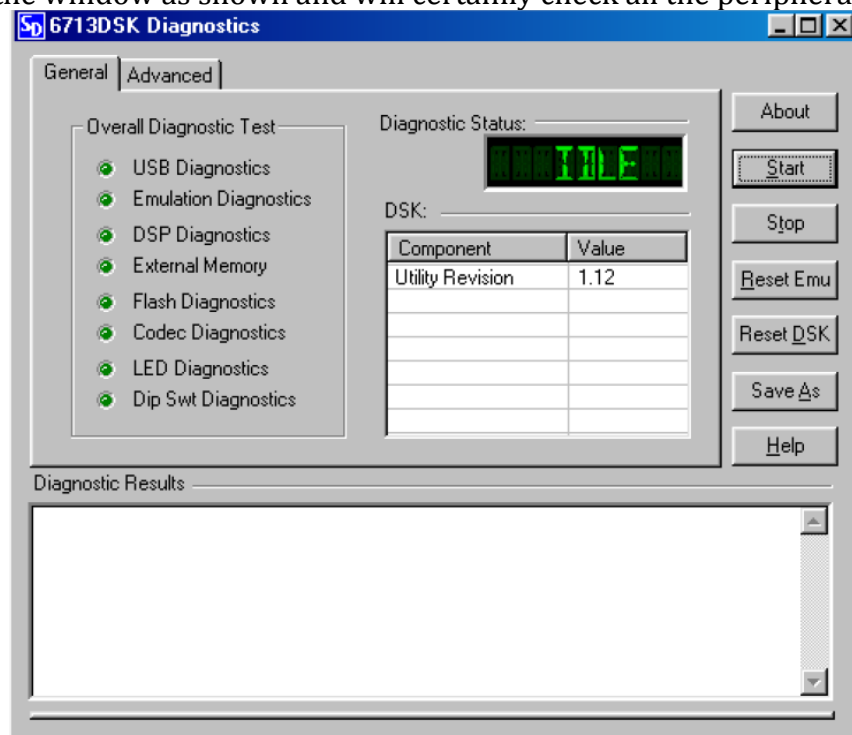
The C6713 DSK has a TMS320C6713 DSP onboard that allows full-speed verification of code with Code Composer Studio. The C6713 DSK provides:

- A USB Interface
- SDRAM and ROM
- An analog interface circuit for Data conversion (AIC)
- An I/O port
- Embedded JTAG emulation support

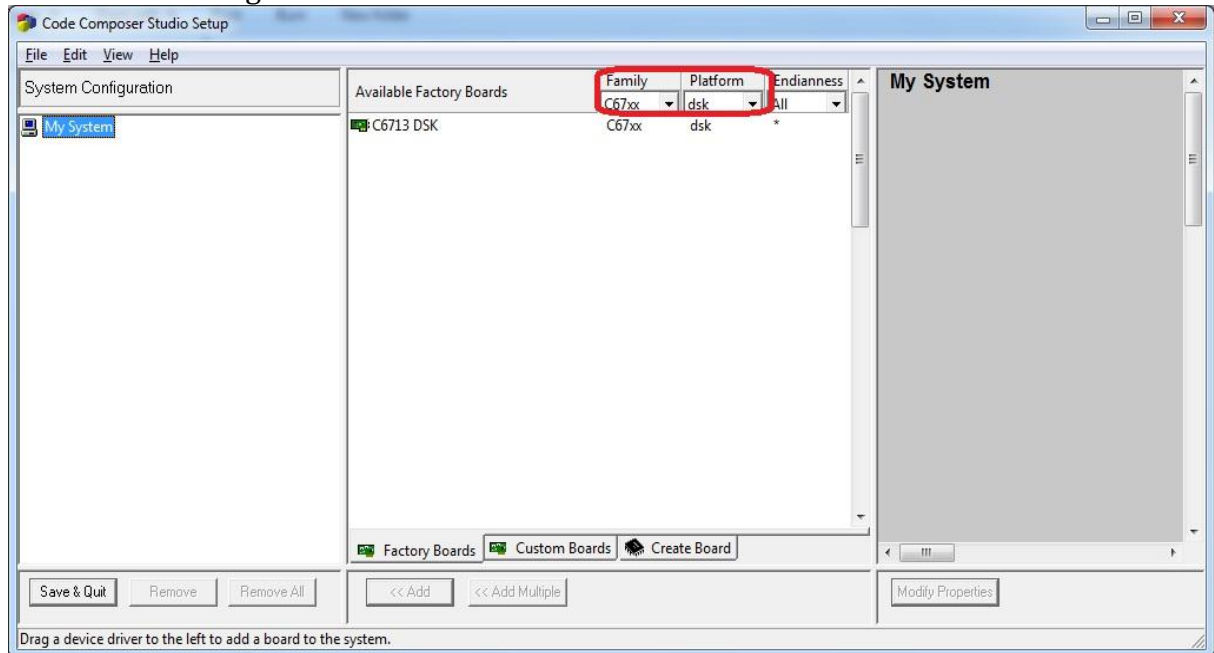


Procedure:

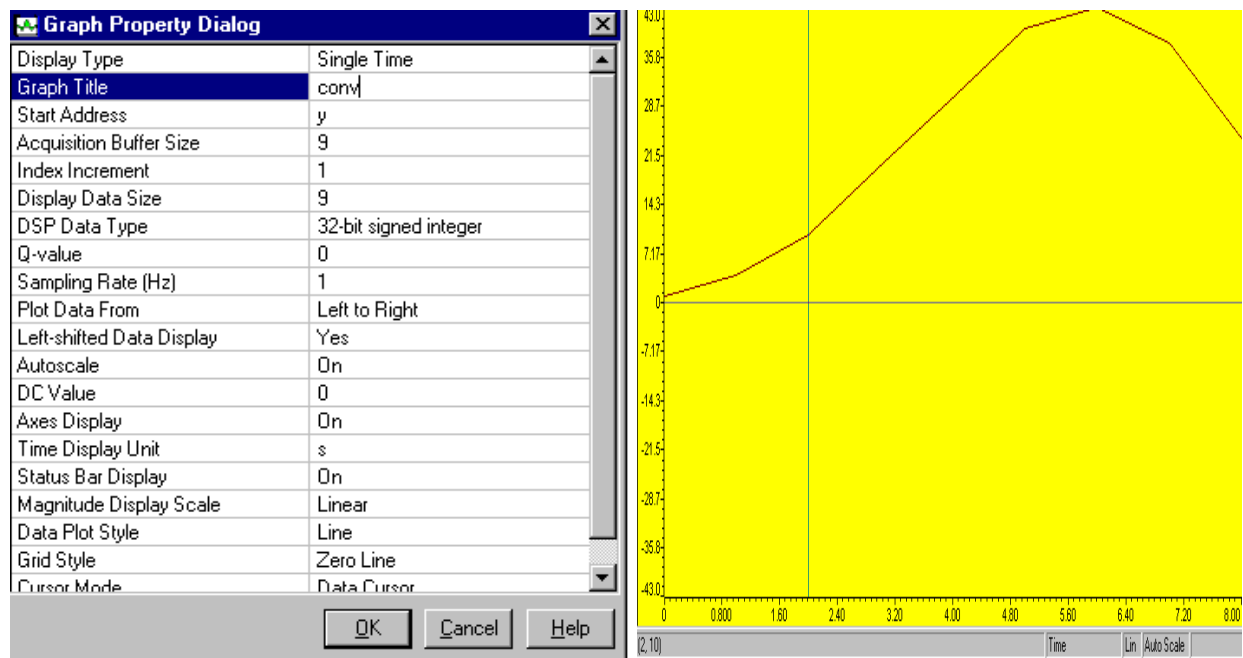
- Firstly we need to connect the kit with power supply and using a USB cable to computer installed with the Code Composer Studio® and the drivers of TMS320C6713. With that we need to check the different peripherals of the kit by connecting them and then using the Diagnostics Test Utility of TMS320C6713 this will open the window as shown and will certainly check all the peripherals



- Now we have to setup the code composer studio as per the kit TMS320C6713. To do that we need to click on “Setup Code Composer Studio” the window will contain different configurations as shown.



- Then select “Family” **C67XX** and then in “Platform” as **dsk** then select C6713 and click on “<<Add”.
- This will create prerequisites of the C6713 in the project. After that click on “Save & Quit” this will ask for starting CCS on exit Click Yes.
- After that click on menu Debug > Connect to connect the code composer studio IDE to the DSK board.
- Start a new project using Project > New. Save it with any desired name here we will use it as newprjt.pjt in any directory.
- Now write a Code of Convolution of two discrete sequences. Code is shown at the end.
- Save the file and then add that file to project from “Project > add files to project”.
- Add the following files.
 - Linker command file hello.cmd
(Path:C:\CCStudio_v3.1\tutorial\dsk6713\hello1\hello.cmd)
 - Run time Support Library rts6700.lib
(Path:C:\CCStudio_v3.1\c6000\cgtools\lib\rts6700.lib)
- Now compile the program using Project > Compile.
- Then Build it by clicking Program > Build.
- This will build an .out file for example in our case “newprjt.out”. Load this program onto the chip from File > Load Program.
- Then run the program from Debug > Run and observe the output by using graph utility.
- To view output select View > Graph > Time and Frequency
- Configure the window as follows



- Code: "conv.c"

```
#include<stdio.h>
#include<math.h>
int y[20];
main()
{
    int m=6;
    int n=6;
    int i, j;
    int x[15]={1,2,3,4,5,6,0,0,0,0,0,0,0,0,0};
    int h[15]={1,2,3,4,5,6,0,0,0,0,0,0,0,0,0};
    for(i=0;i<m+n-1;i++)
    {
        y[i]=0;
        for(j=0;j<i;j++)
            y[i]+=x[j]*h[i-j];
    }
    for(i=0;i<m+n-1;i++)
        printf("%d\n",y[i]);
}
```