



RAO BAHADUR Y MAHABALESHWARAPPA ENGINEERING COLLEGE, BALLARI.

CANTONMENT, BALLARI – 583104 (KARNATAKA)

(AFFILIATED TO VISVESWARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI & APPROVED BY AICTE, NEW DEHLI)

DSP LAB MANUAL

V SEMESTER

2020-21



Name : _____

Subject Code : _____

USN No : _____ **Section:** _____ **Batch** _____



Department of Electronics and Communication Engineering

INDEX

S.N	Date	Programs/ Experiments	P. No	Conduction & Result	Journal /Record Submission	Sign
PART-A						
		List of Experiments	01			
01		Program to Verify Sampling Theorem.	3-4			
02a		Program to find linear convolution of two finite length sequences	5-6			
02b		Program to find Circular convolution of two finite length sequences	7-8			
3a		Program to find the Autocorrelation of a given sequence	9-10			
3b		Program to find the Cross correlation of a given sequence and verification of its properties	11-12			
04		Program to solve a given Difference Equation	13-14			
05		a) Program to find N-Point DFT of the given Sequence, without using inbuilt function.	15-16			
		b) Program to find IDFT of the given sequence.	17-18			
06		Verification of DFT Properties i)Linearity	19-21			
		ii) Parsevals Theorem				
07		a)Design of FIR Low Pass filter	22-23			
		b)Design of FIR High Pass Filter	24-25			
		c) Design of FIR Band Pass filter	26-27			
		d) Design of FIR Band reject filter	28-29			
08		a)Design of digital Chebyshev filter	30-31			
		b)Design of digital Butterworth filter	32-33			

INDEX

S. N	Date	Experiments	P. No	Conduction & Result	Journal /Record Submission	Sign
PART-B						
09		Linear convolution of two given sequences	35			
10		Circular convolution of two given sequences	36-37			
11		Impulse response of a given system	38-39			
12		Difference Equations	40-41			
		Procedure For Part-B	43-45			
		DSK Hardware Installation Procedure	47-48			
		Content Beyond the Syllabus	50-55			
		Appendix	56-63			
		DSP Lab VIVA Questions	64			
		Course outcome of DSP Laboratory	65			
		Vision Mission PEO & PSO	66-67			
		Program Outcomes	68-75			

Conduction & Result Average Marks :	Journal /Record Submission Average Marks :	Lab IA Marks :	Total Marks :	Final Marks :

DSP LAB**SUBJECT:** DSP LAB**SEM:** V**SUBJECT CODE:** (18ECL57)**Branch:** E&CE**A List of Experiments Using MATLAB/ Code Composer Studio and****Objectives.**

1. Verification of sampling theorem.
2. Linear & Circular convolution of two given sequences.
3. Auto & Cross correlation of two sequences & verification of their properties.
4. Solving a given difference equation.
5. Computation of N point DFT of a given sequence and to plot magnitude and phase spectrum.
- 6.Verification of DFT Properties.
 - a) Linearity b) Parsevals Theorem
- 7.Design & implementation of FIR filter to meet the desired specifications.
- 8.Design and implementation of Digital IIR filter to meet given specifications.

B. LIST OF EXPERIMENTS USING DSP PROCESSOR

- 9.Linear convolution of two given sequences.
- 10.Circular convolution of two given sequences.
- 11.Impulse response of a given system.
- 12.Solving a given Difference Equations.

PART-A
DSP PROGRAMS USING MAT LAB

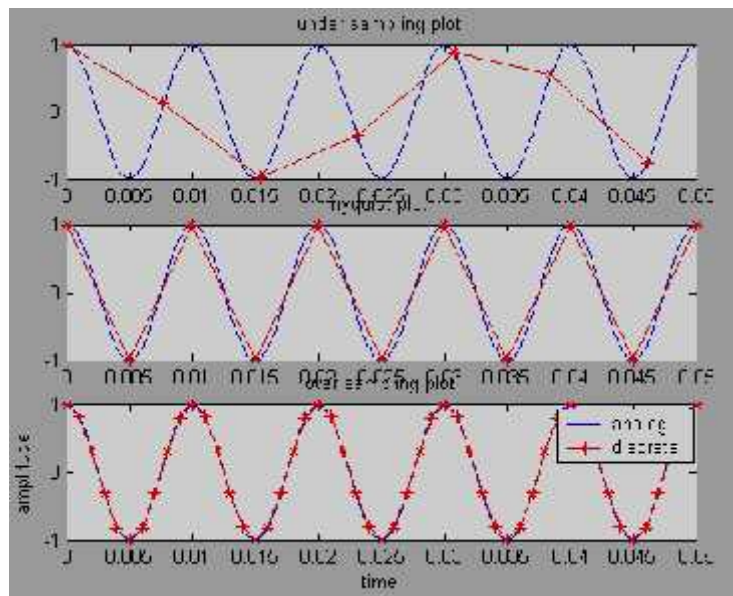
01. Program to verify Sampling Theorem.

```
pi=3.141;
tf=0.05;
t=0:0.0005:tf;
fd=input('enter analog frequency:');
xt=cos(2*pi*fd*t);
fs1=1.3*fd;
n1=0:1/fs1:tf;
xn=cos(2*pi*n1*fd);
subplot(3,1,1);
plot(t,xt,'b',n1,xn,'r*-');
title('under sampling plot')
fs2=2*fd;
n2=0:1/fs2:tf;
xn=cos(2*pi*n2*fd);
subplot(3,1,2);
plot(t,xt,'b',n2,xn,'r*-');
title('nyquist plot');
fs3=10*fd;
n3=0:1/fs3:tf;
xn=cos(2*pi*n3*fd);
subplot(3,1,3);
plot(t,xt,'b',n3,xn,'r*-');
title('over sampling plot');
xlabel('time');
ylabel('amplitude');
legend('analog','discrete');
```

Result:

Enter analog frequency: 100

Plot:



02a. Program to find linear convolution of two finite length sequences.

```
x=input('enter the value of x(n)=');
h=input('enter the value of h(n)=');
lx=length(x);
lh=length(h);
ly=lx+lh-1;
y=conv(x,h);
disp(x);
disp(h);
disp(y);
```

```
n=0:1:lx-1;
subplot(2,2,1)
stem(n,x);
xlabel('n---->');
ylabel('x(n)---->');
title('signal x(n)');
```

```
n=0:1:lh-1;
subplot(2,2,2)
stem(n,h);
xlabel('n---->');
ylabel('x(n)---->');
title('signal h(n)');
```

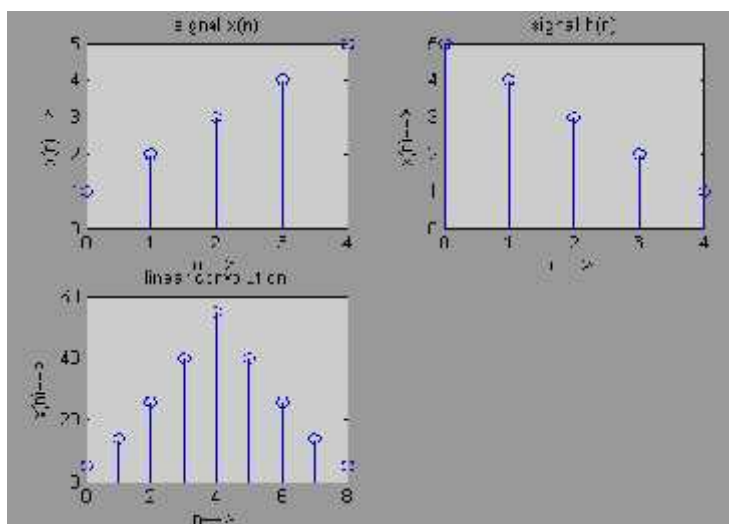
```
n=0:1:ly-1;
subplot(2,2,3)
stem(n,y);
xlabel('n---->');
ylabel('x(n)---->');
title('linear convolution');
```


Result:Enter the value of $x(n)=[1 \ 2 \ 3 \ 4 \ 5]$ Enter the value of $h(n)=[5 \ 4 \ 3 \ 2 \ 1]$

1 2 3 4 5

5 4 3 2 1

5 14 26 40 55 40 26 14 5

Plot:

02b. Program to find Circular convolution of two finite length sequences.**Main Program:**

```

x1=input('Enter the first sequence: ');
x2=input('Enter the second sequence: ');
y1=conv(x1,x2)

l1=length(x1);
l2=length(x2);

n1=max(l1,l2)
y2=cc29(x1,x2,n1)

n2=l1+l2-1
y3=cc29(x1,x2,n2)
subplot(3,1,1)

n1=0:1:length(y1)-1;
stem(n1,y1);
xlabel('time----->');
ylabel('amp----->');
title('linear convolution');

subplot(3,1,2)
n2=0:1:length(y2)-1;
stem(n2,y2);
xlabel('time----->');
ylabel('amp----->');
title('circular convolution');

subplot(3,1,3)
n3=0:1:length(y3)-1;
stem(n3,y3);
xlabel('time---->');
ylabel('amp----->');
title('linear conv by circular convolution');

```

Function file cc29.m to compute circular convolution :

```

function[y] = cc29(x1, x2, N)
l1=length(x1);
l2=length(x2);
X1 = [x1, zeros(1, N - l1)];
X2 = [x2, zeros(1, N - l2)];
H = zeros(N, N);

for n = 1 : 1 : N
    m = n - 1;
    p = 0 : 1 : N - 1;
    q = mod(p - m, N);
    Xm = X2(q + 1);
    H(n,:) = Xm;
end

y = H' * X1';

```

Result:

Enter the first sequence: [1 2 3 4]

Enter the second sequence: [1 2 3 4]

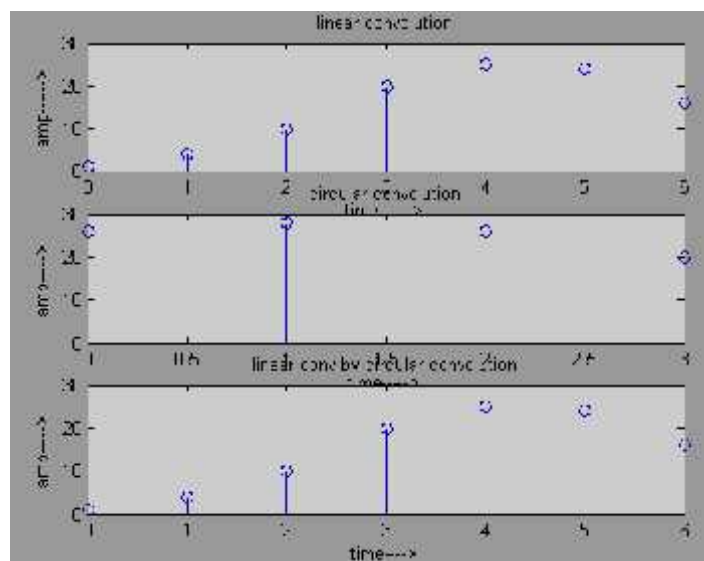
$y1 = 1 \quad 4 \quad 10 \quad 20 \quad 25 \quad 24 \quad 16$

$n1 = 4$

$y2 = 26 \quad 28 \quad 26 \quad 20$

$n2 = 7$

$y3 = 1 \quad 4 \quad 10 \quad 20 \quad 25 \quad 24 \quad 16$

Plot:

03a. Program to find the Autocorrelation of a given sequence and verification of its properties.

```
x=input('enter the sequence:');
Rxx=xcorr(x,x);
figure(1)
subplot(2,1,1)
stem(x);
xlabel('n----->');
ylabel('amplitude---->');
subplot(2,1,2)
stem(fliplr(Rxx))
disp('Rxx');
fliplr(Rxx);
%verification of its properties%
%property 1:rxx"(0) gives the energy of the sequence%
centre=ceil(length(Rxx)/2)
n=-centre+1:1:centre-1
figure,stem(n,Rxx)
E=sum(x.^2)
if(E==Rxx(centre))
disp('Rxx(0) value is equal to the energy of the signal');
else
disp('Rxx(0) value is not equal to the energy of the signal');
end
%property 2:Rxx(n)=Rxx(-n)
l1=length(Rxx)
Zxx=[Rxx(l1:-1:1)]
if(Zxx==Rxx)
disp('Rxx(n) is equal to rxx(-n)');
else
disp('Rxx(n) is not equal to Rxx(-n)');
end
```

Result:

Enter the sequence:[1 2 3]

Rxx

Enter the sequence[3 2 1]

x1 = 1 2 3

rxx = 3 8 14 8 3

centre = 3

n = -2 -1 0 1 2

E = 14

rxx(0) value is equal to the energy of signal

l1 = 5

zxx = 3 8 14 8 3

rxx(n) is equal to rxx(-n)

Plot:

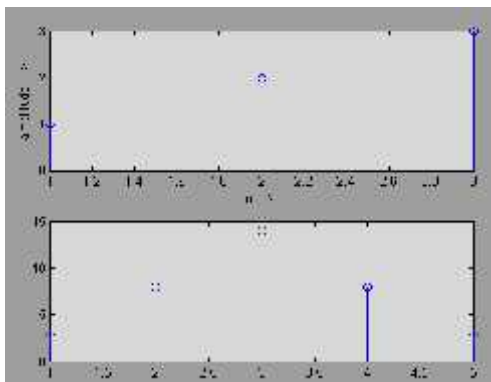


Fig:1

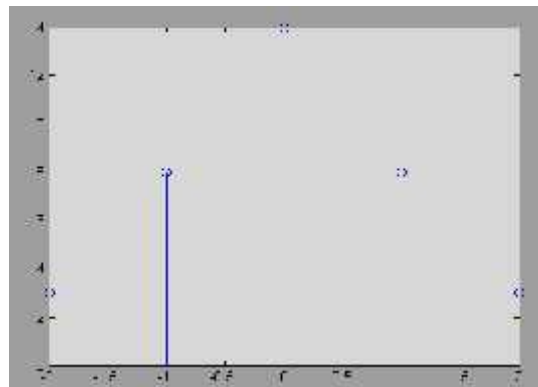


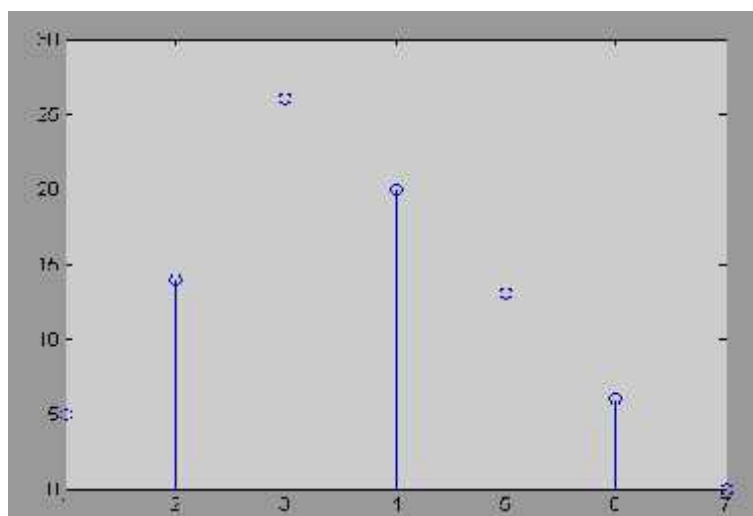
Fig : 2

03b. Program to find the Cross correlation of a given sequence and verification of its properties.

```
x=input('enter the first sequence:');
y=input('enter the second sequence:');
rxy=xcorr(x,y)
l1=length(rxy)
figure,stem(rxy);
ryx=xcorr(y,x)
l=length(ryx)
zyx=[ryx(l1:-1:1)]
if(rxy==zyx)
disp('rxy(n) is equal to rxy(-n)');
else
disp('rxy(n) is not equal to rxy(-n)');
end
ex=sum(x.^2)
ey=sum(y.^2)
rxx=xcorr(x,x);
ryy=xcorr(y,y);
c1=ceil(length(rxx)/2)
c2=ceil(length(ryy)/2)
if sqrt(ex*ey)==sqrt(rxx(c1)*ryy(c2))
disp('energy condition is satisfied');
else
disp('energy condition is not satisfied');
end
```

Result:

```
Enter the first sequence:[1 2 3]
Enter the second sequence:[2 3 4 5]
rxy = 5 14 26 20 13 6 0
l1 = 7
ryx = 0 6 13 20 26 14 5
l = 7
zyx = 5 14 26 20 13 6 0
rxy(n) is equal to rxy(-n)
ex = 14
ey = 54
c1 = 3
c2 = 4
Energy condition is satisfied.
```

Plot:

04. Program to solve a given Difference Equation.

```
a=input('Enter the Numerator Co-efficient a:');  
b=input('Enter the Denominator Co-efficient b:');  
[h,t]=impz(a,b,5)
```

```
x=input('Enter the Input Sequence x(n):');  
y=conv(x,h)
```

```
subplot(2,1,1)  
stem(t,h)  
xlabel('Time->');  
ylabel('Amplitude->');  
title('Impulse response h(n)');
```

```
n=0:1:length(y)-1;  
subplot(2,1,2)  
stem(n,y)  
xlabel('Time->');  
ylabel('Amplitude->');  
title('Output Sequence y(n)');
```


Result:

Enter the Numerator Co-efficient a:[1 3]

Enter the Denominator Co-efficient b:[1 3 2]

r = -1 2

p = -2 -1

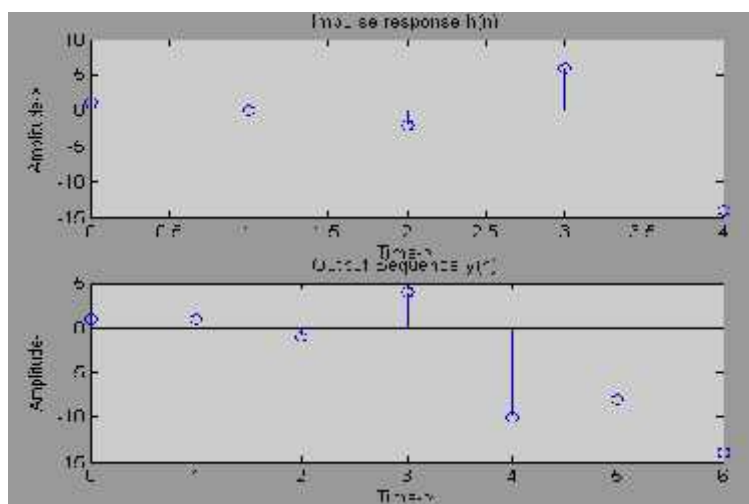
k = []

h = 1 0 -2 6 -14

t = 0 1 2 3 4

Enter the Input Sequence x(n):[1 1 1]

y = 1 1 -1 4 -10 -8 -14

Plot:

05a). Program to find N-Point DFT of the given Sequence, without using inbuilt function.

```
x=input('i/p seq :')
N=length(x)
wn=exp(-j*2*pi/N)
n=0:1:N-1
k=0:1:N-1
nk=n'*k
w=wn.^nk
X=w*x'
figure(1);
stem(n,abs(X));
title('magnitude');
figure(2);
stem(n,angle(X));
title('phase spectrum');
```

Result:

i/p seq :[1 2 3 4]

x = 1 2 3 4

N = 4

wn = 0.0000 - 1.0000i

n = 0 1 2 3

k = 0 1 2 3

nk =

0 0 0 0

0 1 2 3

0 2 4 6

0 3 6 9

w =

1.0000	1.0000	1.0000	1.0000
1.0000	0.0000 - 1.0000i	-1.0000 - 0.0000i	-0.0000 + 1.0000i
1.0000	-1.0000 - 0.0000i	1.0000 + 0.0000i	-1.0000 - 0.0000i
1.0000	-0.0000 + 1.0000i	-1.0000 - 0.0000i	0.0000 - 1.0000i

X =

10.0000
 -2.0000 + 2.0000i
 -2.0000 - 0.0000i
 -2.0000 - 2.0000i

Plot:

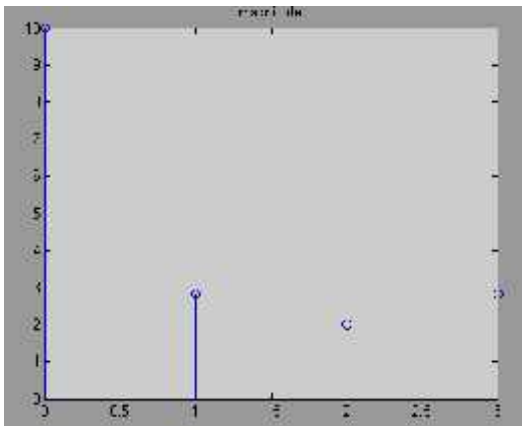


Fig: 1

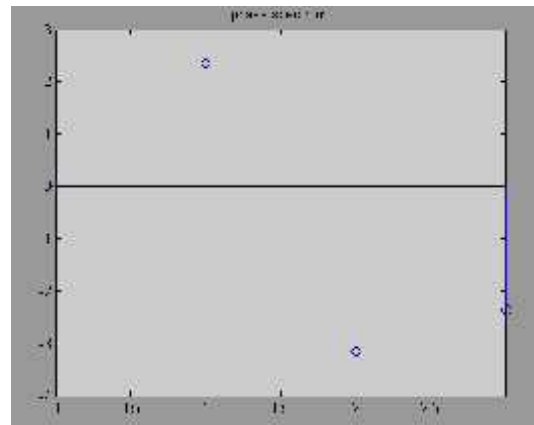


Fig :2

05b) Program to find IDFT of the given sequence.

```
X=input('i/p seq :')
N=input('Enter the length of the sequence:')
wn=exp(j*2*pi/N)
n=0:1:N-1
k=0:1:N-1
nk=k'*n
w=wn.^nk
x=X*w
x=x/N
figure(1);
stem(n,abs(x));
title('magnitude');
figure(2);
stem(n,angle(x));
title('phase spectrum');
```

Result:

i/p seq :[10 -2+2i -2 -2-2i]

X =10.0000 -2.0000+2.0000i -2.0000 -2.0000-2.0000i

Enter the length of the sequence: 4

N = 4

x =1.0000 2.0000+0.0000i 3.0000-0.0000i 4.0000-0.0000i

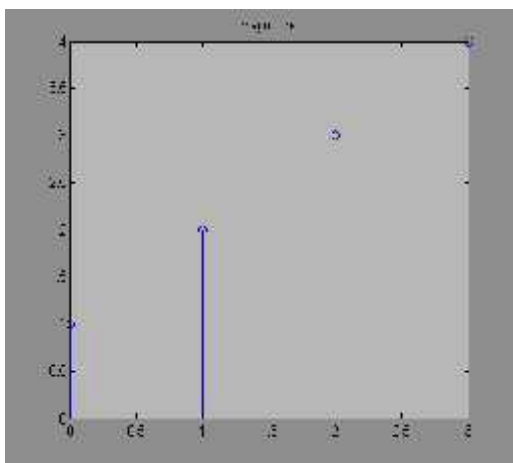
Plot:

Fig: 1

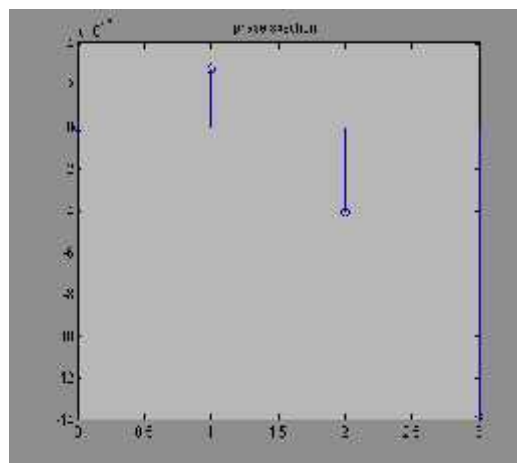


Fig: 2

06. Verification of DFT Properties.**i) Linearity ii) Parsevals Theorem**

```

% Input sequences x1(n) and x2(n):
x1n = [1 2 3 4];
x2n = [8 2 5 6];

N = max(length(x1n),length(x2n));
%Let N = length of largest sequence

% Corresponding DFTs X1(k) and X2(k)
X1k = fft(x1n,N);
X2k = fft(x2n,N);

% --- Display Graphs ---
figure(1)
subplot(2,2,1);
stem(x1n);
xlabel('n');
ylabel('x1(n)');
title('Signal x1(n)');
subplot(2,2,2);
stem(x2n);
xlabel('n');
ylabel('x2(n)');
title('Signal x2(n)');
subplot(2,2,3);
stem(abs(X1k));
xlabel('k');
ylabel('X1(k)');
title('DFT of Signal 1: X1(k)');
subplot(2,2,4);
stem(abs(X2k));
xlabel('k');
ylabel('X2(k)');
title('DFT of Signal 2: X2(k)');

%% --- 1. Linearity Property ---
A=1;
B=2;
xin = A.*x1n + B.*x2n;
xout = fft(xin,N);

```

```
xout_ex = A.*X1k + B.*X2k;
disp('1. Linearity Property');
disp(['Expected X(k): ', num2str(xout_ex)]);
disp(['Calculated X(k): ', num2str(xout)]);
if(xout == xout_ex)
disp('Linearity Property of N-point DFT is verified. ');
else
disp('Linearity Property of N-point DFT is not verified. ');
end
%% ----2. Parseval's Theorem ---
% To prove that  $E(x1(n)) = (1/N)E(x2(n))$ 
E_xin = x1n*x1n';
% Cross Product of x1(n) with itself
E_xout = (1/N).*X1k*X1k';
% Cross Product of X1(k) with itself disp('2. Parsevals Theorem');
disp(['Energy in Time Domain ', num2str(E_xin)]);
disp(['Energy in Frequency Domain: ', num2str(E_xout)]);
    if(int8(E_xin)==int8(E_xout))
disp('Parsevals Theorem is verified. ');
else
disp('Parsevals Theorem is not verified');
end
```

Result:

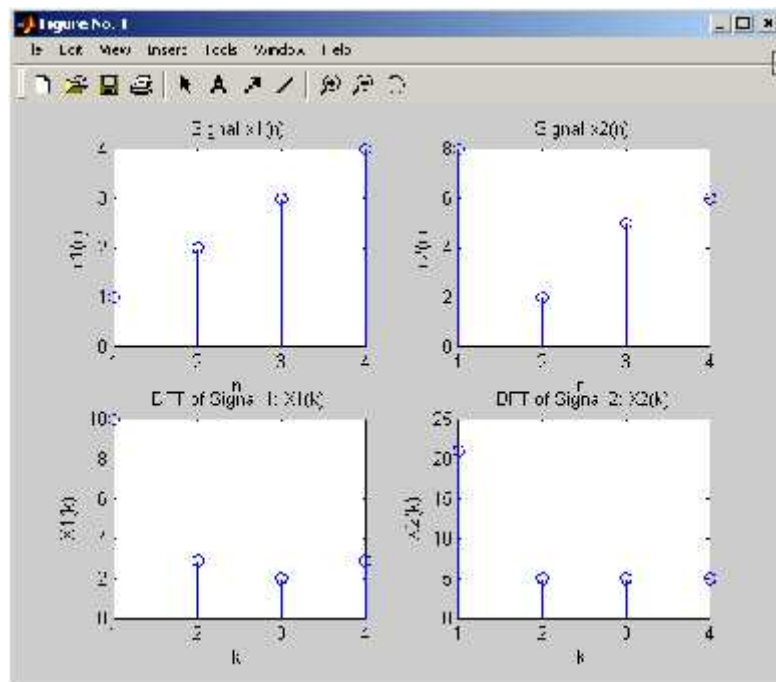
a. Linearity Property

Expected $X(k)$: $52+0i$ $4+10i$ $8+0i$ $4-10i$
 Calculated $X(k)$: $52+0i$ $4+10i$ $8+0i$ $4-10i$
 Linearity Property of N-point DFT is verified.

b. Parsevals Theorem

Energy in Time Domain 30
 Energy in Frequency Domain: 30
 Parsevals Theorem is verified.

Plot :

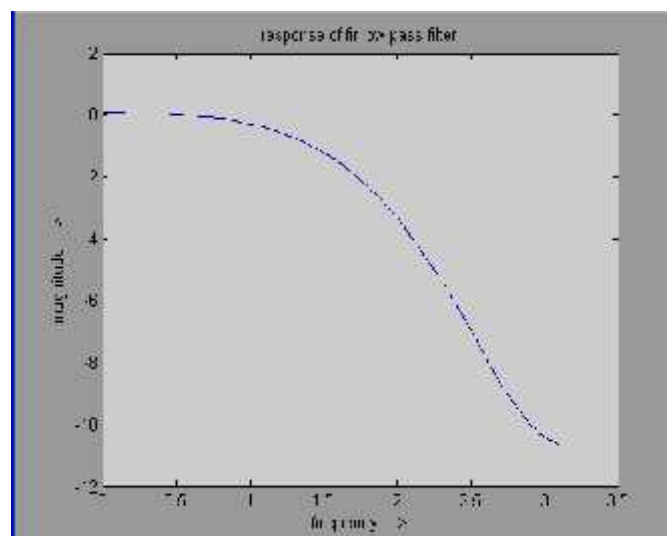


7a). Design a FIR LPF for $N=7$, $W_c=1$ radians/Sec using hamming window.

```
pi=3.142;
wc=(3*pi)/4;
N=7;
t=(N-1)/2;
w=hamming(N);
n=0;
c=sin(wc*(n-t));
d=pi*(n-t);
hd=c/d;
h=hd*w(1);
for n=1:1:N-1
if (n==t)
hd1(n)=wc/pi;
else
a=sin(wc*(n-t));
b=pi*(n-t);
hd1(n)=a/b;
end
h1(n)=hd1(n)*w(n+1);
end
hd=[hd hd1];
h=[h h1];
disp(hd)
disp(w)
disp(h)
wm=0:0.1:pi;
hm=freqz(h,1,wm);
m=20*log10(abs(hm));
plot(wm,m);
xlabel('frequency---->');
ylabel('magnitude---->');
title('response of fir low pass filter');
```

Result:

0.0751	-0.1591	0.2250	0.7500	0.2250	-0.1591	0.0751
0.0800	0.3100	0.7700	1.0000	0.7700	0.3100	0.0800
0.0060	-0.0493	0.1732	0.7500	0.1732	-0.0493	0.0060

Plot:

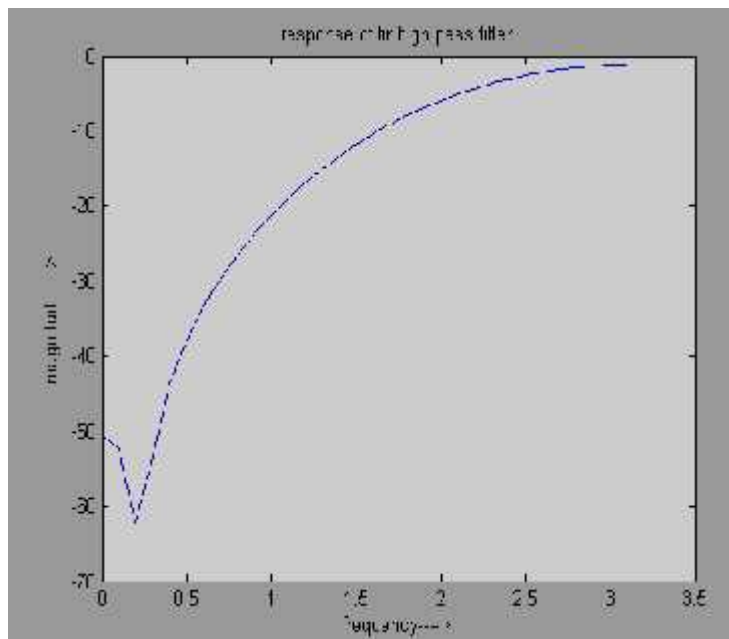
7b). Design a FIR HPF for $N=7$, $W_c=2$ radians/Sec using hamming window.

```
pi=3.142;
wc=2;
N=7;
t=(N-1)/2;
w=hamming(N);
n=0;
c=[sin(pi*(n-t))-sin(wc*(n-t))];
d=pi*(n-t);
hd=c/d;
h=hd*w(1);
for n=1:1:N-1
if (n==t)
hd1(n)=1-(wc/pi);
else
a=[sin(pi*(n-t))-sin(wc*(n-t))];
b=pi*(n-t);
hd1(n)=a/b;
end
h1(n)=hd1(n)*w(n+1);
end
hd=[hd hd1];
h=[h h1];
disp(hd)
disp(w)
disp(h)
wm=0:0.1:pi;
hm=freqz(h,1,wm);
m=20*log10(abs(hm));
plot(wm,m);
xlabel('frequency---->');
ylabel('magnitude---->');
title('response of fir high pass filter');
```

Result:

0.0295	0.1206	-0.2895	0.3635	-0.2895	0.1206	0.0295
0.0800	0.3100	0.7700	1.0000	0.7700	0.3100	0.0800
0.0024	0.0374	-0.2229	0.3635	-0.2229	0.0374	0.0024

Plot:



7c). Design a FIR Band Pass filter for $N=7$, $W_{c1}=1$ radians/Sec & $W_{c2}=2$ radians/Sec using rectangular window.

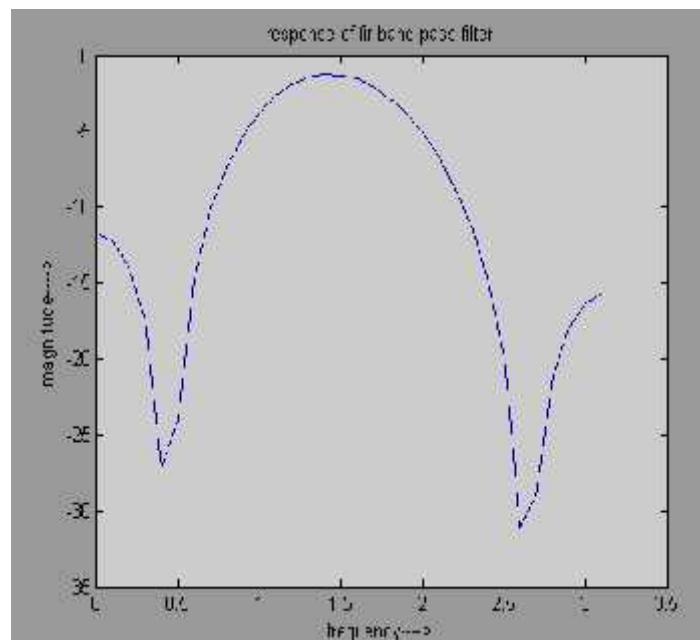
```

pi=3.142;
wc1=1;
wc2=2;
N=7;
t=(N-1)/2;
w=boxcar(N);
n=0;
c=[sin(wc2*(n-t))-sin(wc1*(n-t))];
d=pi*(n-t);
hd=c/d;
h=hd*w(1);
for n=1:1:N-1
if (n==t)
hd1(n)=(wc2-wc1)/pi;
else
a=[sin(wc2*(n-t))-sin(wc1*(n-t))];
b=pi*(n-t);
hd1(n)=a/b;
end
h1(n)=hd1(n)*w(n+1);
end
hd=[hd hd1];
h=[h h1];
disp(hd)
disp(w)
disp(h)
wm=0:0.1:pi;
hm=freqz(h,1,wm);
m=20*log10(abs(hm));
plot(wm,m);
xlabel('frequency---->');
ylabel('magnitude---->');
title('response of fir band pass filter');

```

Result:

```
-0.0446 -0.2651 0.0216 0.3183 0.0216 -0.2651 -0.0446  
1 1 1 1 1 1 1  
-0.0446 -0.2651 0.0216 0.3183 0.0216 -0.2651 -0.0446
```

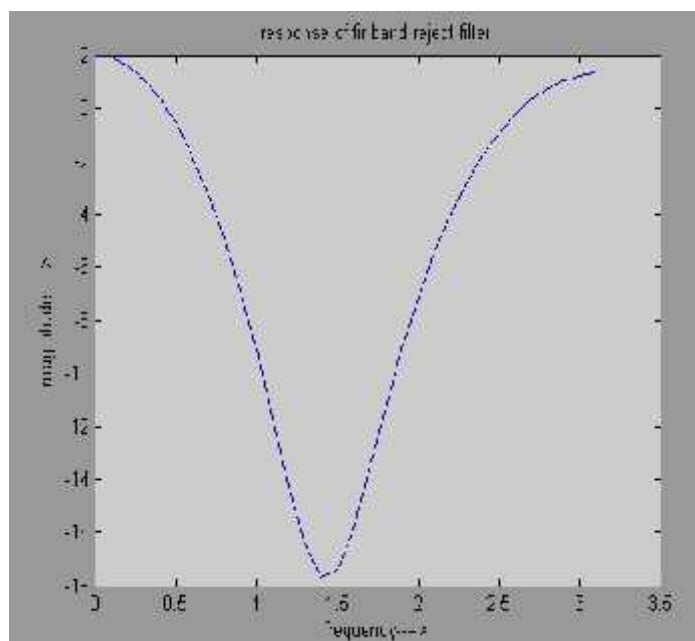
Plot:

7d). Design a FIR Band reject filter for $N=7$, $W_{c1}=1$ radians/Sec & $W_{c2}=2$ radians/Sec using rectangular window.

```
pi=3.142;
wc1=1;
wc2=2;
N=7;
t=(N-1)/2;
w=boxcar(N);
n=0;
c=[sin(wc1*(n-t))-sin(wc2*(n-t))+sin(pi*(n-t))];
d=pi*(n-t);
hd=c/d;
h=hd*w(1);
for n=1:1:N-1
if (n==t)
hd1(n)=(pi-wc2+wc1)/pi;
else
a=[sin(wc1*(n-t))-sin(wc2*(n-t))+sin(pi*(n-t))];
b=pi*(n-t);
hd1(n)=a/b;
end
h1(n)=hd1(n)*w(n+1);
end
hd=[hd hd1];
h=[h h1];
disp(hd)
disp(w)
disp(h)
wm=0:0.1:pi;
hm=freqz(h,1,wm);
m=20*log10(abs(hm));
plot(wm,m);
xlabel('frequency---->');
ylabel('magnitude---->');
title('response of fir band reject filter');
```

Result:

```
0.0445  0.2653  -0.0217  0.6817  -0.0217  0.2653  0.0445
1       1       1       1       1       1       1
0.0445  0.2653  -0.0217  0.6817  -0.0217  0.2653  0.0445
```

Plot:

8. a) Design the digital filter using Chebyshev approximation to meet the following specification with pass band edge at 0.15π and stop band edge at 0.45π for a sampling frequency of 1, variation of gain with pass band 1db and stop band attenuation of 15 db. Use Bilinear Transformation.

```
ap=input('enter pass band attenuation:')
as=input('enter stop band attenuation:')
wp=input('enter pass band edge frequency:')
ws=input('enter stop band edge frequency:')
t=input('enter sampling frequency:')
fs=1/t;
```

```
%translation of analog to digital
```

```
wp=wp/fs
```

```
ws=ws/fs
```

```
%pre-wrapping
```

```
wpp=(2/t)*tan(wp/2)
```

```
wss=(2/t)*tan(ws/2)
```

```
%to obtain H(s)
```

```
[N,wc]=cheb1ord(wpp,wss,ap,as,'s')
```

```
[a,b]=cheby1(N,ap,wc,'s')
```

```
%analog to digital transformation
```

```
[az,bz]=bilinear(a,b,fs)
```

```
%plot of response
```

```
k=0:0.01/pi:pi;
```

```
h=freqz(az,bz,k);
```

```
gain=20*log10(abs(h));
```

```
subplot(3,1,1)
```

```
plot(k,abs(h));
```

```
subplot(3,1,2)
```

```
plot(k,gain);
```

```
%grid on;
```

```
subplot(3,1,3)
```

```
plot(k,angle(h));
```

Result:

enter pass band attenuation:1

ap = 1

enter stop band attenuation:15

as = 15

enter pass band edge frequency:0.15*pi

wp = 0.4712

enter stop band edge frequency:0.45*pi

ws = 1.4137

enter sampling frequency:1

t = 1

wp = 0.4712

ws = 1.4137

wpp = 0.4802

wss = 1.7082

N = 2

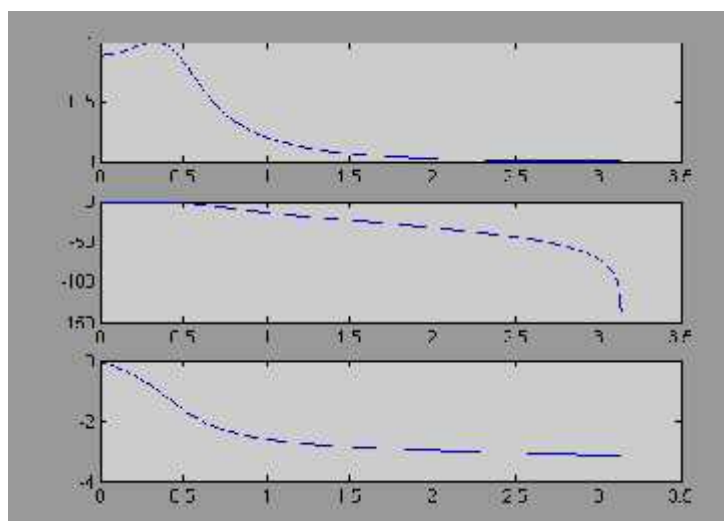
wc = 0.4802

a = 0 0 0.2265

b = 1.0000 0.5271 0.2542

az = 0.0427 0.0854 0.0427

bz = 1.0000 -1.4113 0.6028

Plot:

8. b) Design the digital Butterworth filter to meet the following specification with pass band edge at $\pi/2$ and stop band edge at $(3\pi)/4$ for a sampling frequency of 1, variation of gain with pass band 3db and stop band attenuation of 14db. Use impulse invariant method.

```
ap=input('enter the pass band attenuation:');
as=input('enter the stop band attenuation:');
fp=input('enter the pass band frequency:');
fs=input('enter the stop band frequency :');
Fs=input('enter the sampling frequency :');

w1=fp*Fs;
w2=fs*Fs;
[N,wn]=buttord(w1,w2,ap,as,'s')
[b,a]=butter(N,wn,'s')
[Num,Den]=impinvar(b,a,Fs)
freqz(Num,Den)
title('butter worth low pass filter frequency response');
%plot of response
k=0:0.01/pi:pi;
h=freqz(Num,Den,k);
gain=20*log10(abs(h));

subplot(3,1,1)
plot(k,abs(h));

subplot(3,1,2)
plot(k,gain);
%grid on;

subplot(3,1,3)
plot(k,angle(h));
```

Result:

enter the pass band attenuation:3

enter the stop band attenuation:14

enter the pass band frequency: $\pi/2$

enter the stop band frequency : $3\pi/4$

enter the sampling frequency :1

N = 4

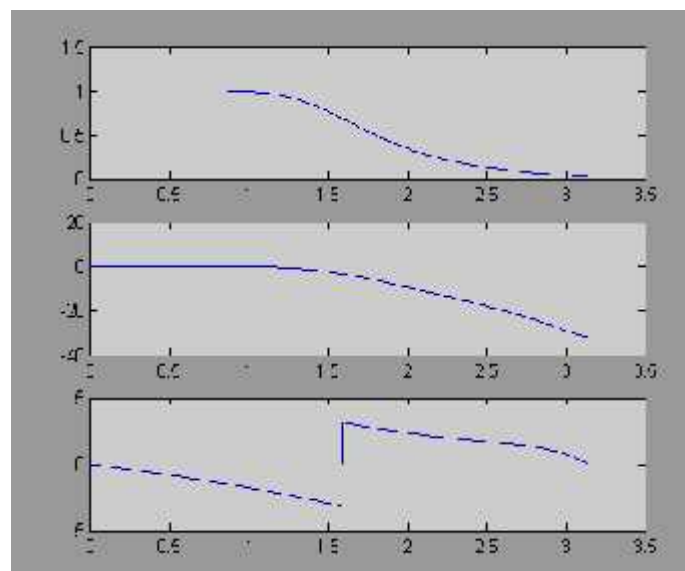
wn = 1.5828

b = 0 0 0 0 6.2758

a = 1.0000 4.1360 8.5531 10.3612 6.2758

Num = -0.0000 0.3293 0.4271 0.0427 0

Den = 1.0000 -0.4992 0.3965 -0.1198 0.0160

Plot:



PART-B
EXPERIMENTS USING DSP PROCESSOR

09. Program to Find the Linear convolution of two finite length sequences.

```
#include<stdio.h>
int y[20];
main()
{
int m=6;                /*Length of i/p samples sequence*/
int n=6;                /*Length of impulse response Co-efficients */
int i=0,j;
int x[15]={1,2,3,4,5 }; /*Input Signal Samples*/
int h[15]={5,4,3,2,1}; /*Impulse Response Co-efficients*/
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]);
}
```

Result:

5 14 26 40 55 40 26 14 5 0 0

10. Program to Find the Circular convolution of two finite length sequences.

```
#include<stdio.h>
#include<math.h>
int m,n,x[30],h[30],y[30],i,j, k,x2[30],a[30];
void main()
{
printf("enter the length of 1st sequence\n");
scanf("%d",&m);
printf("enter the length of 2nd sequence\n");
scanf("%d",&n);
printf("enter the 1st sequence\n");
for(i=0;i<m;i++)
scanf("%d",&x[i]);
printf("enter the 2nd sequence\n");
for(j=0;j<n;j++)
scanf("%d",&h[j]);
if(m-n!=0)
{
if(m>n)
{
for(i=n;i<m;i++)
h[i]=0;
n=m;
}
else
{
for(i=m;i<n;i++)
x[i]=0;
m=n;
}
}
y[0]=0;
a[0]=h[0];
for(j=1;j<n;j++)
```

```
a[j]=h[n-j];
for(i=0;i<n;i++)
y[0]+=x[i]*a[i];
for(k=1;k<n;k++)
{
y[k]=0;
for(j=1;j<n;j++)
x2[j]=a[j-1];
x2[0]=a[n-1];
for(i=0;i<n;i++)
{
a[i]=x2[i];
y[k]+=x[i]*x2[i];
}
}
printf("\n circular convolution\n");
for(i=0;i<n;i++)
printf("%d\t",y[i]);
}
```

Result:

Enter the length of 1st sequence

2

Enter the length of 2nd sequence

4

Enter the 1st sequence

1 2

Enter the 2nd sequence

1 2 3 4

Circular convolution

9 4 7 10

11. Program to Find the Impulse response of first order and second order system.

```
#include<stdio.h>
#define order 2
#define len 3

float Y[len]={0,0,0},sum;
void main()
{
int j,k;
float a[order+1]={1,2};
float b[order+1]={1,5,6};
for(j=0;j<len;j++)
{
sum=0;
for(k=1;k<=order;k++)
{
if(j-k>=0)
sum+=b[k]*Y[j-k];
}
if(j<=order)
{
Y[j]=a[j]-sum;
}
else
Y[j]=-sum;
printf("response[%d]%f\n\n",j,Y[j]);
}
}
```

Result:

response[0]1.000000

response[1]-3.000000

response[2]9.000000

12. Program to implement Difference Equations.

```
#include<stdio.h>
#include<math.h>
#define FREQ 400
float y[3]={0,0,0};
float x[3]={0,0,0};
float z[128],m[128],n[128],p[128];
main()
{
int i=0,j;
float a[3]={0.072231,0.144462,0.072231};
float b[3]={1.000000,-1.109229,0.398152};
for(i=0;i<128;i++)
{
m[i]=sin(2*3.14*FREQ*i/24000);
}
for(j=0;j<128;j++)
{
x[0]=m[j];
y[0]=(a[0]*x[0])+(a[1]*x[1])+(x[2]*a[2])-(y[1]*b[1])-(y[2]*b[2]);
z[j]=y[0];
y[2]=y[1];
y[1]=y[0];
x[2]=x[1];
x[1]=x[0];
}
}
```

Result:

Enter the Numerator Co-efficient a:[1 3]

Enter the Denominator Co-efficient b:[1 3 2]

r = -1 2

p = -2 -1

k = []

h = 1 0 -2 6 -14

t = 0 1 2 3 4

Enter the Input Sequence x(n):[1 1 1]

y = 1 1 -1 4 -10 -8 -14



**PROCEDURE FOR
CODE COMPOSER STUDIO**

Procedure:

Start Code Composer

To Start Code Composer Studio, Double Click the 6713 DSK CCStudio_v3.1 icon on your desktop.



6713 DSK CCStudio v3.1.lnk

Start Code Composer Studio (ignore this if CCS is already running) by double-clicking on the C6713 DSK icon on your desktop.

Use the **Debug** → **Connect** menu option to open a debug connection to the DSK board

Procedure to work on Code Composer Studio

Generation of Linear Convolution

1. To create a New Project

Project → New (**Lin.pjt**)



2. To Create a Source files

File → New → Source file.

Files of type → C Source File (*.c,*.ccc)

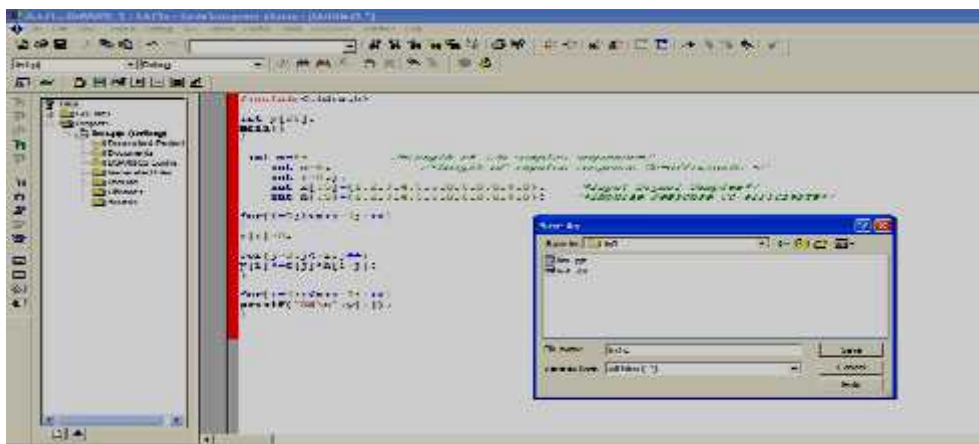


Type the code in editor window.

Save the file in project folder.(Eg : Lin.c).

3. To Add Source files to Project

Project → Add files to Project → Lin.c
Files of type → C Source File (*.c,*.ccc)



4. To Add rts6700.lib file.

Project → Add files to Project → rts6700.lib
Files of type → C Source File (*.o*,*.1*)
Path : C:\CCStudio_v.3.1\c6000\cgtools\lib\rts6700.lib

5. Add Command and linking file

Project → Add files to Project → hello.cmd
Files of type → Linker command file (*.cmd,*.lcf)

Path: C:\CCStudio_v3.1\tutorial\dkshello1\hello.cmd



6. To Compile:

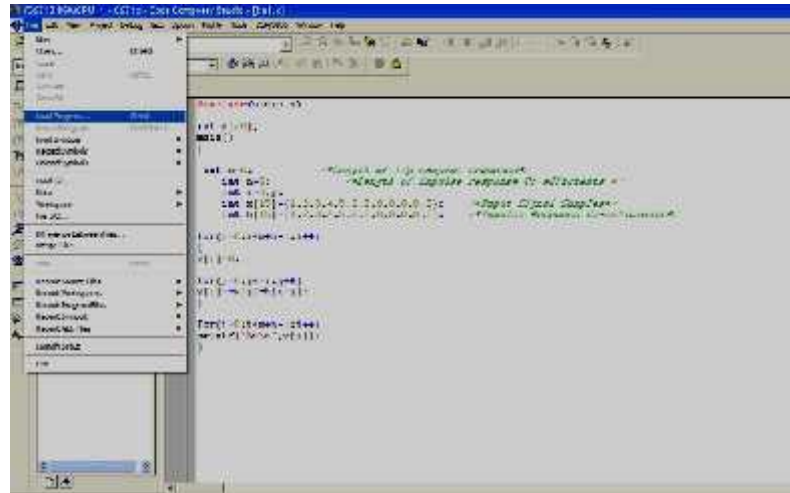
Project → Compile File

7. To build or Link:

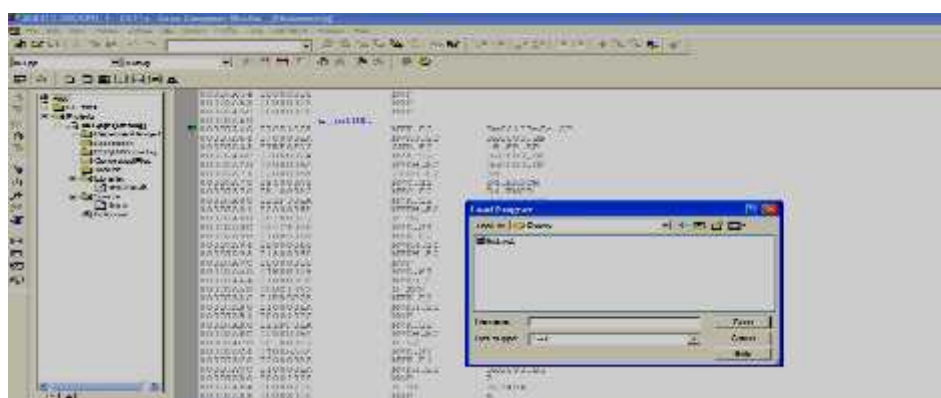
Project → Build,
Which will create the final executable (.out) file.(Eg. Lin.out).

8. Procedure to Load and Run Program:

Load program to DSK:



File → Load Program → Lin.out
From **Debug** Folder



9. To Execute Project:

Debug → Run

C C Studio Installation For Interfacing

DSK Hardware Installation Procedure

DSK Hardware Installation

- Shut down and power off the PC.
- Connect the Supplied USB Port cable to the board.
- Connect the other end of the cable to the USB port of PC

Note: if you plan to install a Microphone, Speaker, or Signal Generator/CRO these must be plugged in properly Before you connect power to the DSK

- Plug the power cable into the board
- Plug the other end of the power cable into a power outlet
- The user LEDs should flash several times to indicate board is operational
- When you connect your DSK through USB for the first time on a Windows loaded PC the new hardware found wizard will come up. So. Install the Drivers (The CCS CD contains the require drivers for C6713 DSK).
- Install the CCS software for C6713 DSK.

DSK Software Installation

You must install the hardware before you install the software on your system.

The requirements for the operating platform are;

- Insert the Installation CD into CD-ROM drive
An install screen appears like below; if not, goes to the windows Explorer and run setup.exe



- Choose the option to install Code Composer Studio



If you already have C6000 CC Studio IDE installed on your PC, Do not install DSK software. CC Studio IDE full tools supports the DSK Plot form

- Respond to the dialog boxes as the installation program runs.
- The Installation program automatically configures CC Studio IDE for operation with your DSK and creates a **“6713 DSK CCStudio_v3.1”** IDE DSK icon on your desktop.

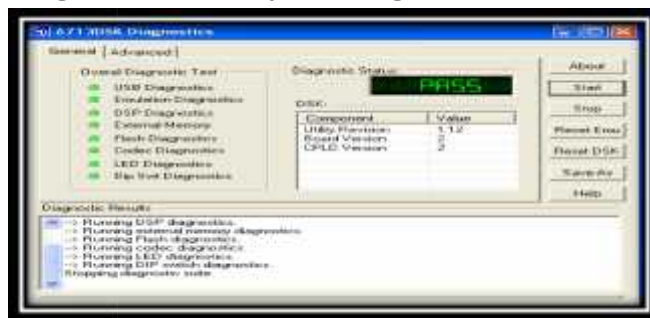


DIAGNOSTIC:-

- 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



- Select → Start → Select 6713 DSK Diagnostic Utility
- The Screen Look like as below. The below Screen will appear
- Select **Start** Option
- Utility Program will test the board
- After testing Diagnostic Status you will get **PASS**

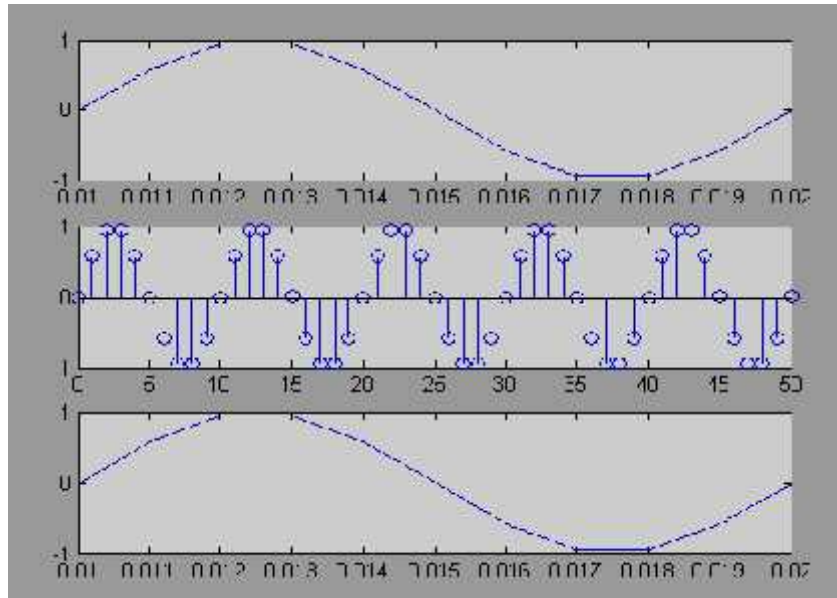


**Content Beyond the
Syllabus**

01.Program to Verify Sampling Theorem. Ex: $x(t)=\sin(2\pi 100t)$.

```
dt=.001;
t=.01:dt:.02;
%fm=input('enter i/p signal freq:');
xi=sin(2*pi*100*t);
%fs=input('enter the sampling frequency:');
fs=1000;
ps=1/fs;
n=0:1:50;
xs=sin(2*pi*100*(n*ps));
nps=n*ps;
xr=xs*sinc(fs*(ones(length(n),1)*t-nps'*ones(1,length(t))));
figure(1);
subplot(3,1,1);plot(t,xi);
subplot(3,1,2);stem(n,xs);
subplot(3,1,3);plot(t,xr);
```

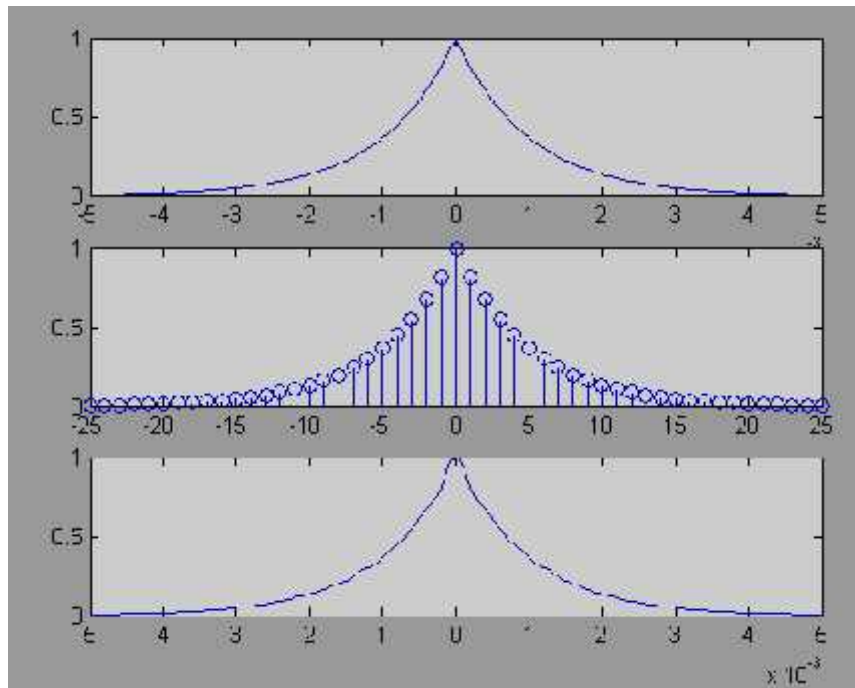
Plot:



2. Program to Verify Sampling Theorem.**Ex: $e(t)=\exp(-1000|t|)$.**

```
clear;
dt=.00005;
t=-.005:dt:.005;
%fm=input('enter i/p sgnl freq:');
xa=exp(-1000*abs(t));
%fs=input('enter i/p sgnl freq:');
fs=5000;
ps=1/fs;
n=-25:1:25;
x=exp(-1000*abs(n*ps));
nps=n*ps;
xa1=x*sinc(fs*(ones(length(n),1)*t-nps'*ones(1,length(t))));
figure(1);
subplot(3,1,1);plot(t,xa);
subplot(3,1,2);stem(n,x);
subplot(3,1,3);plot(t,xa1);
```

Plot:



03. Program to find Impulse Response of the Given System.

```
x=input('Enter the Co-efficients of x : ');
y=input('Enter the Co-efficients of y : ');
N=input('Enter the Desired Impulse Response Length : ');
n=0:1:N-1;
d=ones(1, N);

subplot(1, 2, 1)
stem(n, d);
[z, t]=impz (x, y, N);
disp(z);
disp(t);

subplot(1, 2, 2)
stem(t, z);
xlabel('n---->');
ylabel('amplitude ---->');
title('Impulse Response');
```

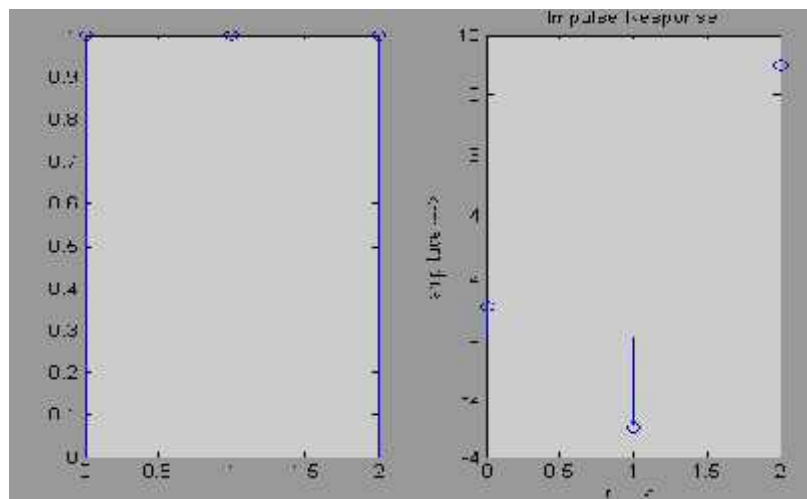
Result:

Enter the Co-efficients of x : Enter the Co-efficients of x : [1 2]

Enter the Co-efficients of y : [1 5 6]

Enter the Desired Impulse Response Length: 3

1 -3 9
0 1 2

Plot:



Appendix

1. Input

INPUT prompt for user input. `R = INPUT('How many apples')` gives the user the prompt in the text string and then waits for input from the keyboard. The input can be any MATLAB expression, which is evaluated, using the variables in the current workspace, and the result returned in `R`. If the user presses the return key without entering anything, `INPUT` returns an empty matrix. `R = INPUT('What is your name','s')` gives the prompt in the text string and waits for character string input. The typed input is not evaluated; the characters are simply returned as a MATLAB string.

2. Figure

FIGURE Create figure window. `FIGURE`, by itself, creates a new figure window, and returns its handle. `FIGURE(H)` makes `H` the current figure, forces it to become visible, and raises it above all other figures on the screen. If Figure `H` does not exist, and `H` is an integer, a new figure is created with handle `H`. `GCF` returns the handle to the current figure. Execute `GET(H)` to see a list of figure properties and their current values. Execute `SET(H)` to see a list of figure properties and their possible values.

3. Subplot

SUBPLOT Create axes in tiled positions. `H = SUBPLOT(m,n,p)`, or `SUBPLOT(mnp)`, breaks the Figure window into an `m`-by-`n` matrix of small axes, selects the `p`-th axes for the current plot, and returns the axis handle. The axes are counted along the top row of the Figure window, then the second row, etc. For example, `SUBPLOT(2,1,1)`, `PLOT(income)` `SUBPLOT(2,1,2)`, `PLOT(outgo)` plots income on the top half of the window and outgo on the bottom half. `SUBPLOT(m,n,p)`, if the axis already exists, makes it current. `SUBPLOT(m,n,p,'replace')`, if the axis already exists, deletes it and creates a new axis. `SUBPLOT(m,n,P)`, where `P` is a vector, specifies an axes position that covers all the subplot positions listed in `P`. `UBPLOT(H)`, where `H` is an axis handle, is another way of making an axis current for subsequent plotting commands. `SUBPLOT('position',[left bottom width height])` creates an axis at the specified position in normalized coordinates (in the range from 0.0 to 1.0). If a `SUBPLOT` specification causes a new axis to overlap an existing axis, the existing axis is deleted - unless the position of the new and existing axis are identical. For example, the statement `SUBPLOT(1,2,1)` deletes all existing axes overlapping the left side of the Figure window and creates a new axis on that side - unless there is an axes there with a position that exactly matches the position of the new axes (and 'replace' was not specified), in which case all other overlapping axes will be deleted and the matching axes will become the current axes.

`SUBPLOT(111)` is an exception to the rules above, and is not identical in behavior to `SUBPLOT(1,1,1)`. For reasons of backwards compatibility, it is a special case of subplot which does not immediately create an axes, but instead sets up the figure so that next graphics command executes `CLF RESET` in the figure (deleting all children of the figure), and creates a new axes in the default position. This syntax does not return a handle, so it is an error to specify a return argument. The delayed `CLF RESET` is accomplished by setting the figure's `NextPlot` to 'replace'.

4. plot

PLOT linear plot. PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix, then the vector is plotted versus the rows or columns of the matrix, whichever line up. If X is a scalar and Y is a vector, length(Y) disconnected points are plotted. PLOT(Y) plots the columns of Y versus their index. If Y is complex, PLOT(Y) is equivalent to PLOT(real(Y),imag(Y)). In all other uses of PLOT, the imaginary part is ignored. Various line types, plot symbols and colors may be obtained with PLOT(X,Y,S) where S is a character string made from one element from any or all the following 3 columns:

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star		
y	yellow	s	square		
k	black	d	diamond		
		v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

5. stem

STEM Discrete sequence or "stem" plot. STEM(Y) plots the data sequence Y as stems from the x axis terminated with circles for the data value. STEM(X,Y) plots the data sequence Y at the values specified in X. STEM(...,'filled') produces a stem plot with filled markers. STEM(...,'LINESPEC') uses the linetype specified for the stems and markers. See PLOT for possibilities. H = STEM(...) returns a vector of line handles.

6. Exp

EXP Exponential. EXP(X) is the exponential of the elements of X, e to the X. For complex $Z=X+i*Y$, $EXP(Z) = EXP(X)*(COS(Y)+i*SIN(Y))$.

7. Ones

ONES array. ONES(N) is an N-by-N matrix of ones. ONES(M,N) or ONES([M,N]) is an M-by-N matrix of ones. ONES(M,N,P,...) or ONES([M N P ...]) is an M-by-N-by-P-by-... array of ones. ONES(SIZE(A)) is the same size as A and all ones.

8. impz

IMPZ Impulse response of digital filter [H,T] = IMPZ(B,A) computes the impulse response of the filter B/A choosing the number of samples for you, and returns the response in column vector H and a vector of times (or sample intervals) in T (T = [0 1 2 ...]). [H,T] = IMPZ(B,A,N) computes N samples of the impulse response. If N is a vector of integers, the impulse response is computed only at those integer values (0 is the origin). [H,T] = IMPZ(B,A,N,Fs) computes N samples and scales T so that samples are spaced 1/Fs units apart. Fs is 1 by default. [H,T] = IMPZ(B,A,[],Fs) chooses the number of samples for you and scales T so that samples are spaced 1/Fs units apart. IMPZ with no output arguments plots the impulse response using STEM(T,H) in the current figure window.

9. disp

DISP Display array. DISP(X) displays the array, without printing the array name. In all other ways it's the same as leaving the semicolon off an expression except that empty arrays don't display. If X is a string, the text is displayed.

10. Title

TITLE Graph title. TITLE ('text') adds text at the top of the current axis. TITLE ('text','Property1', PropertyValue1,'Property2',Property Value2,...) sets the values of the specified properties of the title. H = TITLE (...) returns the handle to the text object used as the title.

11. length

LENGTH of vector. LENGTH(X) returns the length of vector X. It is equivalent to MAX(SIZE(X)) for non-empty arrays and 0 for empty ones.

12. conv

CONV Convolution and polynomial multiplication. C = CONV(A, B) convolves vectors A and B. The resulting vector is length LENGTH (A)+LENGTH(B)-1. If A and B are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials.

13. xlabel

XLABEL X-axis label. XLABEL('text') adds text beside the X-axis on the current axis. XLABEL('text','Property1',PropertyValue1,'Property2',PropertyValue2,...) sets the values of the specified properties of the xlabel. H = XLABEL(...) returns the handle to the text object used as the label.

14. ylabel

YLABEL Y-axis label.YLABEL('text') adds text beside the Y-axis on the current axis. LABEL('text','Property1',PropertyValue1,'Property2',PropertyValue2,...) sets the values of the specified properties of the ylabel. H = YLABEL(...) returns the handle to the text object used as the label.

15. max

MAX Largest component. For vectors, MAX(X) is the largest element in X. For matrices, MAX(X) is a row vector containing the maximum element from each column. For N-D arrays, MAX(X) operates along the first non-singleton dimension. [Y,I] = MAX(X) returns the indices of the maximum values in vector I. If the values along the first non-singleton dimension contain more than one maximal element, the index of the first one is returned.MAX(X,Y) returns an array the same size as X and Y with the largest elements taken from X or Y. Either one can be a scalar. [Y,I] = MAX(X,[],DIM) operates along the dimension DIM. When complex, the magnitude MAX(ABS(X)) is used, and the angle ANGLE(X) is ignored. NaN's are ignored when computing the maximum.

16. xcorr

XCORR Cross-correlation function estimates. C = XCORR(A,B), where A and B are length M vectors (M>1), returns the length 2*M-1 cross-correlation sequence C. If A and B are of different length, the shortest one is zero-padded. C will be a row vector if A is a row vector, and a column vector if A is a column vector. XCORR produces an estimate of the correlation between two random (jointly stationary)

sequences: $C(m) = E[A(n+m) \cdot \text{conj}(B(n))] = E[A(n) \cdot \text{conj}(B(n-m))]$ It is also the deterministic correlation between two deterministic signals. `XCORR(A)`, when A is a vector, is the auto-correlation sequence. `XCORR(A)`, when A is an M-by-N matrix, is a large matrix with $2 \cdot M - 1$ rows whose N^2 columns contain the cross-correlation sequences for all combinations of the columns of A. The zeroth lag of the output correlation is in the middle of the sequence, at element or row M.

`XCORR(...,MAXLAG)` computes the (auto/cross) correlation over the range of lags: -MAXLAG to MAXLAG, i.e., $2 \cdot \text{MAXLAG} + 1$ lags. If missing, default is `MAXLAG = M - 1`. `[C,LAGS] = XCORR(...)` returns a vector of lag indices (LAGS). `XCORR(...,SCALEOPT)`, normalizes the correlation according to `SCALEOPT`: 'biased' - scales the raw cross-correlation by $1/M$. 'unbiased' - scales the raw correlation by $1/(M - \text{abs}(\text{lags}))$. 'coeff' - normalizes the sequence so that the auto-correlations at zero lag are identically 1.0. 'none' - no scaling (this is the default).

17. fliplr

`FLIPLR` Flip matrix in left/right direction. `FLIPLR(X)` returns X with row preserved and columns flipped in the left/right direction.

$$X = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix} \quad \text{becomes} \quad \begin{matrix} 3 & 2 & 1 \\ 6 & 5 & 4 \end{matrix}$$

18. ceil

`CEIL` Round towards plus infinity. `CEIL(X)` rounds the elements of X to the nearest integers towards infinity.

19. Sum

`SUM` Sum of elements. For vectors, `SUM(X)` is the sum of the elements of X. For matrices, `SUM(X)` is a row vector with the sum over each column. For N-D arrays, `SUM(X)` operates along the first non-singleton dimension. `SUM(X,DIM)` sums along the dimension DIM.

Example: If $X = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix}$

then `sum(X,1)` is `[3 5 7]` and `sum(X,2)` is `[3 12]`;

20. residuez

`RESIDUEZ` Z-transform partial-fraction expansion. `[R,P,K] = RESIDUEZ(B,A)` finds the residues, poles and direct terms of the partial-fraction expansion of $B(z)/A(z)$,

$$\frac{B(z)}{A(z)} = \frac{r(1)}{1-p(1)z^{-1}} + \dots + \frac{r(n)}{1-p(n)z^{-1}} + k(1) + k(2)z^{-1} \dots$$

B and A are the numerator and denominator polynomial coefficients, respectively, in ascending powers of z^{-1} . R and P are column vectors containing the residues and poles, respectively. K contains the direct terms in a row vector. The number of poles is $n = \text{length}(A) - 1 = \text{length}(R) = \text{length}(P)$ The direct term coefficient vector is empty if $\text{length}(B) < \text{length}(A)$; otherwise, $\text{length}(K) = \text{length}(B) - \text{length}(A) + 1$ If $P(j) = \dots = P(j+m-1)$ is a pole of multiplicity m, then the expansion includes terms of the form

$$\frac{R(j)}{1 - P(j)z^{-1}} + \frac{R(j+1)}{(1 - P(j)z^{-1})^2} + \dots + \frac{R(j+m-1)}{(1 - P(j)z^{-1})^m}$$

`[B,A] = RESIDUEZ(R,P,K)` converts the partial-fraction expansion back to B/A form.

21. fft

FFT Discrete Fourier transform. FFT(X) is the discrete Fourier transform (DFT) of vector X. For matrices, the FFT operation is applied to each column. For N-D arrays, the FFT operation operates on the first non-singleton dimension. FFT(X,N) is the N-point FFT, padded with zeros if X has less than N points and truncated if it has more. FFT(X,[],DIM) or FFT(X,N,DIM) applies the FFT operation across the dimension DIM. For length N input vector x, the DFT is a length N vector X, with elements

$$X(k) = \sum_{n=1}^N x(n) \exp(-j \cdot 2 \cdot \pi \cdot (k-1) \cdot (n-1) / N), \quad 1 \leq k \leq N.$$

The inverse DFT (computed by IFFT) is given by

$$x(n) = (1/N) \sum_{k=1}^N X(k) \exp(j \cdot 2 \cdot \pi \cdot (k-1) \cdot (n-1) / N), \quad 1 \leq n \leq N.$$

22. ifft

IFFT Inverse discrete Fourier transform. IFFT(X) is the inverse discrete Fourier transform of X. IFFT(X,N) is the N-point inverse transform. IFFT(X,[],DIM) or IFFT(X,N,DIM) is the inverse discrete Fourier transform of X across the dimension DIM.

23. hamming

HAMMING Hamming window. HAMMING(N) returns the N-point symmetric Hamming window in a column vector. HAMMING(N,SFLAG) generates the N-point Hamming window using SFLAG window sampling. SFLAG may be either 'symmetric' or 'periodic'. By default, a symmetric window is returned.

24. fir1

FIR1 FIR filter design using the window method. 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. The normalized gain of the filter at Wn is -6 dB. B = FIR1(N,Wn,'high') designs an N'th order highpass filter. You can also use B = FIR1(N,Wn,'low') to design a lowpass filter. If Wn is a two-element vector, $Wn = [W1 \ W2]$, FIR1 returns an order N bandpass filter with passband $W1 < W < W2$. You can also specify B = FIR1(N,Wn,'bandpass'). If $Wn = [W1 \ W2]$, B = FIR1(N,Wn,'stop') will design a bandstop filter. If Wn is a multi-element vector, $Wn = [W1 \ W2 \ W3 \ W4 \ W5 \ \dots \ WN]$, 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 passband. B = FIR1(N,Wn,'DC-0') makes the first band a stopband. B = FIR1(N,Wn,WIN) designs an N-th order FIR filter using the N+1 length vector WIN to window the impulse response. If empty or omitted, FIR1 uses a Hamming window of length N+1. For a complete list of available windows, see the help for the WINDOW function. 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 with R decibels of relative sidelobe attenuation. For filters with a gain other than zero at $F_s/2$, e.g., highpass and bandstop filters, N must be even. Otherwise, N will be incremented by one. In this case the window length should be specified as N+2.

25. freqz

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:

$$H(e^{j\omega}) = \frac{b(1)e^{j\omega} + b(2)e^{-j\omega} + \dots + b(m+1)e^{-jm\omega}}{a(1)e^{j\omega} + a(2)e^{-j\omega} + \dots + a(n+1)e^{-jnw}}$$

given numerator and denominator coefficients 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. [H,W] = FREQZ(B,A,N,'whole') uses N points around the whole unit circle. H = FREQZ(B,A,W) returns the frequency response at frequencies designated in vector W, in radians/sample (normally between 0 and pi). [H,F] = FREQZ(B,A,N,Fs) and [H,F] = FREQZ(B,A,N,'whole',Fs) return frequency vector F (in Hz), where Fs is the sampling frequency (in Hz). H = FREQZ(B,A,F,Fs) returns the complex frequency response at the frequencies designated in vector F (in Hz), where Fs is the sampling frequency (in Hz). FREQZ(B,A,...) with no output arguments plots the magnitude and unwrapped phase of the filter in the current figure window.

26. bilinear

BILINEAR Bilinear transformation with optional frequency prewarping. [Zd,Pd,Kd] = BILINEAR(Z,P,K,Fs) converts the s-domain transfer function specified by Z, P, and K to a z-transform discrete equivalent obtained from the bilinear transformation: $H(z) = H(s) \mid \mid s = 2*Fs*(z-1)/(z+1)$

where column vectors Z and P specify the zeros and poles, scalar K specifies the gain, and Fs is the sample frequency in Hz. [NUMd,DENd] = BILINEAR(NUM,DEN,Fs), where NUM and DEN are row vectors containing numerator and denominator transfer function coefficients, NUM(s)/DEN(s), in descending powers of s, transforms to z-transform coefficients NUMd(z)/DENd(z). [Ad,Bd,Cd,Dd] = BILINEAR(A,B,C,D,Fs) is a state-space version. Each of the above three forms of BILINEAR accepts an optional additional input argument that specifies prewarping. For example, [Zd,Pd,Kd] = BILINEAR(Z,P,K,Fs,Fp) applies prewarping before the bilinear transformation so that the frequency responses before and after mapping match exactly at frequency point Fp (match point Fp is specified in Hz).

27. buttord

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

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

BUTTORD also returns Wn, the Butterworth natural frequency (or, the "3 dB frequency") to use with BUTTER to achieve the specifications. [N, Wn] = BUTTORD(Wp, Ws, Rp, Rs, 's') does the computation for an analog filter, in which case Wp and Ws are in radians/second. When Rp is chosen as 3 dB, the Wn in BUTTER is equal to Wp in BUTTORD.

28. butter

BUTTER Butterworth digital and analog filter design. $[B,A] = \text{BUTTER}(N,Wn)$ designs an Nth order lowpass digital Butterworth filter and returns the filter coefficients in length $N+1$ vectors B (numerator) and A (denominator). The coefficients are listed in descending powers of z . The cutoff frequency Wn must be $0.0 < Wn < 1.0$, with 1.0 corresponding to half the sample rate. If Wn is a two-element vector, $Wn = [W1 W2]$, BUTTER returns an order $2N$ bandpass filter with passband $W1 < W < W2$. $[B,A] = \text{BUTTER}(N,Wn,'high')$ designs a highpass filter. $[B,A] = \text{BUTTER}(N,Wn,'stop')$ is a bandstop filter if $Wn = [W1 W2]$. When used with three left-hand arguments, as in $[Z,P,K] = \text{BUTTER}(\dots)$, the zeros and poles are returned in length N column vectors Z and P, and the gain in scalar K. When used with four left-hand arguments, as in $[A,B,C,D] = \text{BUTTER}(\dots)$, state-space matrices are returned. $\text{BUTTER}(N,Wn,'s')$, $\text{BUTTER}(N,Wn,'high','s')$ and $\text{BUTTER}(N,Wn,'stop','s')$ design analog Butterworth filters. In this case, Wn is in [rad/s] and it can be greater than 1.0.

29. impinvar

IMPINVAR Impulse invariance method for analog to digital filter conversion. $[BZ,AZ] = \text{IMPINVAR}(B,A,Fs)$ 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 sampled at a frequency of Fs Hertz. The B and A coefficients will be scaled by $1/Fs$. If you don't specify Fs , it defaults to 1 Hz. $[BZ,AZ] = \text{IMPINVAR}(B,A,Fs,TOL)$ uses the tolerance TOL for grouping repeated poles together. Default value is 0.001, i.e., 0.1%.

30. grid on

GRID Grid lines. GRID ON adds major grid lines to the current axes. GRID OFF removes major and minor grid lines from the current axes. GRID MINOR toggles the minor grid lines of the current axes. GRID, by itself, toggles the major grid lines of the current axes. $\text{GRID}(AX,\dots)$ uses axes AX instead of the current axes. GRID sets the XGrid, YGrid, and ZGrid properties of the current axes. $\text{set}(AX,'XMinorGrid','on')$ turns on the minor grid.

DSP Lab Viva Questions:

1. Relation between CTS & DTS.
2. Define sampling theorem.
3. Relationship between DFT & FFT.
4. What is the inbuilt function used in MATLAB to find the DFT.
5. Which is the inbuilt function used in MATLAB to find Z Transformation of given Signal.
6. Which is the inbuilt function used in MATLAB to find the Correlation of given Signals.
7. Which function is used to plot the CTS.
8. Which function is used to plot the DTS.
9. Purpose for using the subplot function in MATLAB.
10. Write the properties of DFT.
11. Write differences between FIR & IIR.
12. What are the specifications of CHEBYSCHEV 1,2 & 3 function.
13. What are the differences between the digital filter & analog filter.
14. Write difference between LPF & HPF.
15. What do you mean by IMPZ in MATLAB.
16. What are the convolution function.
17. Relationship between Linear Convolution and Circular Convolution.
18. What are difference equations.
19. Difference between convolution and correlation.
20. Purpose of Impulse function in DSP.



COURSE OUTCOMES OF DSP LABORATORY



Name of the Staff: Mr. Surendranath/Mrs. Rakhee Patil /Mrs. Suvarna Patil/Mrs. Girija Vani.G				
Course Name: DIGITAL SIGNAL PROCESSING LABORATORY				
Course Code: 18ECL57	Sem:	5 A&B	Year	2020-21

COURSE OUTCOME STATEMENTS	
	At the end of the course, students will be able to ...
C307.1	Understand the concepts of analog to digital conversion of signals and frequency domain sampling of signals.
C307.2	Modeling of discrete time signals and systems and verification of its properties and results.
C307.3	Implementation of discrete computations using DSP processor and verify the results
C307.4	Realize the digital filters using a simulation tool and analyze the response of the filter for an audio signal



**VISION, MISSION, PROGRAM EDUCATIONAL OBJECTIVES and PROGRAM
 SPECIFIC OUTCOMES (PSO)**

VISION OF THE INSTITUTION

To Produce Professionally Excellent, Knowledgeable, Globally Competitive and Socially Responsible Engineers and Entrepreneurs.

MISSION OF THE INSTITUTION

M1	To provide quality education in Engineering and Management.
M2	To establish a continuous industry-institute interaction, participation and collaboration to contribute skilled Engineers.
M3	To develop human values, social values, entrepreneurship skills and professional ethics among the technocrats.
M4	To focus on innovation and development of technologies by engaging in cutting edge research areas.

VISION OF THE DEPARTMENT

To Produce Professionally Excellent, Knowledgeable, Globally Competitive, Socially Responsible Electronics and Communication Engineers and Entrepreneurs.

MISSION OF THE DEPARTMENT

M1	To provide quality education in Electronics and Communication Engineering.
M2	To establish a continuous industry-institute interaction, participation and collaboration to contribute skilled Electronics and Communication Engineers.
M3	To develop human values, social values, entrepreneurship skills and professional ethics among the technocrats.
M4	To focus on innovation and development of technologies by engaging in Electronics and Communication Research areas.

[Signature]
Criteria I coordinator

[Signature]
NBA coordinator

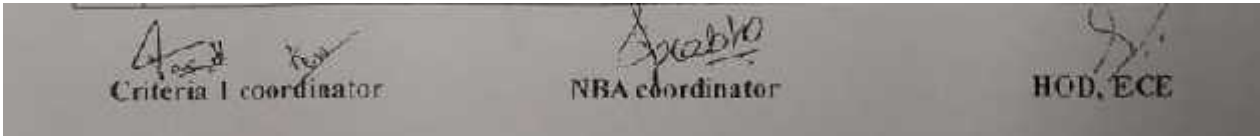
[Signature]
HOD, ECE

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1	Graduates of Electronics & Communication Engineering course will have successful professional career.
PEO2	Graduates of Electronics & Communication Engineering course will pursue higher education or to become an Entrepreneur.
PEO3	Graduates of Electronics & Communication Engineering course will have ability for lifelong learning and to serve the society.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1	Ability to Design, Develop and Test the Electronics Circuits & Communication Systems.
PSO2	Ability to Develop Excellent Programming and Problem Solving skills in the field of Embedded System.



Program Outcomes

PO 1	Engineering Knowledge	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO 2	Problem Analysis	Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO 3	Design/ Development of Solutions	Design solutions for complex engineering problems and design system components or processes that meet the Specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO 4	Conduct investigations of complex problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO 5	Modern tool usage	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO 6	The engineer and society	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO 7	Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO 8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO 9	Individual and team work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO 10	Communication	Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO 11	Project management and finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO 12	Life-long learning	Recognize the need for, and have the preparation and ability to engage in Independent and life-long learning in the broadest context of technological change.

