INDIAN INSTITUTE OF INFORMATION TECHNONOLY,NAGPUR.

DIGITAL SIGNAL PROCESSING (ECL 301)

LAB TASKS

NAME : PRATIK ADLE
ROLL NO : BT17ECE034

CONTENTS

LAB 1 : Z Transform

Aim : To Visualize 3D Pole Zero Plot of a given Z Transform .

Theory:

The Laplace transform was an extension of the continuous-time Fourier transform . It can be applied to a broader class of signals than the Fourier Transform can, since there are many signals for which the Fourier transform does not converge but the Laplace transform does.

We use the same approach for discrete time as we use the **z Transform** , which is the discrete-time counterpart of the Laplace transform. Z Transform can be applied to a broader class of signals than that of Discrete Time Fourier Transform (DTFT) can, since there are many signals for which the Discrete Time Fourier transform does not converge but the Z transform does.

The z-transform of a general discrete-time signal *x[n]* is defined as :

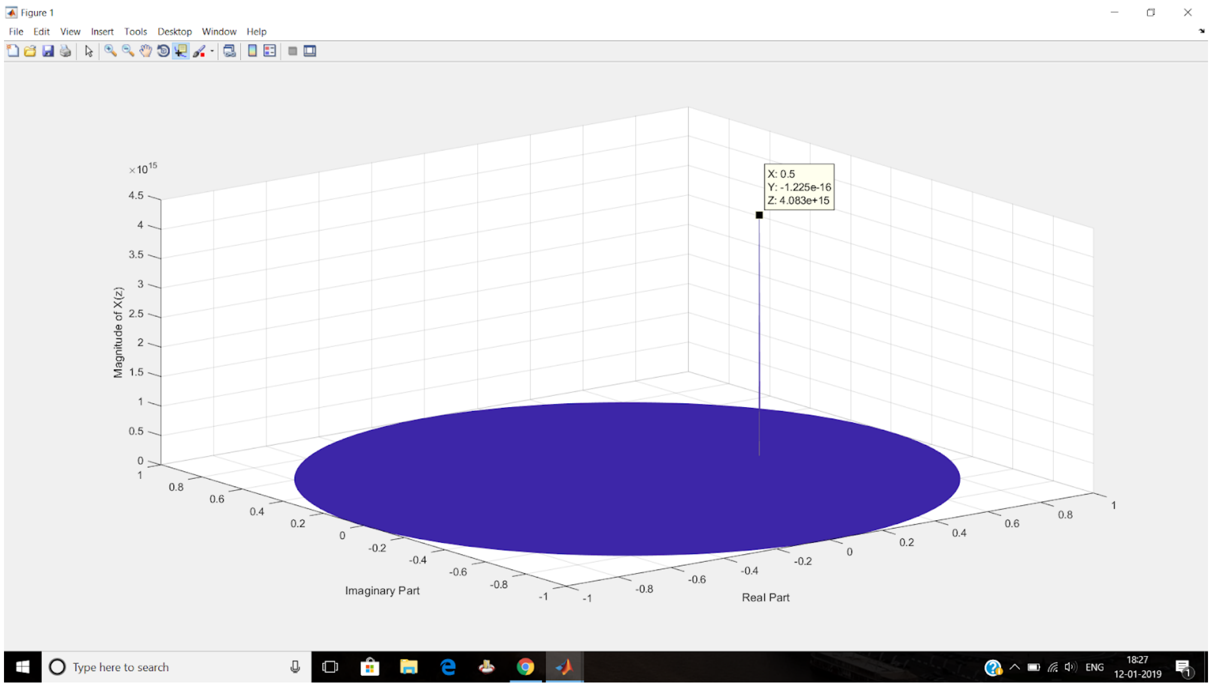
$$\boxed{X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}}$$

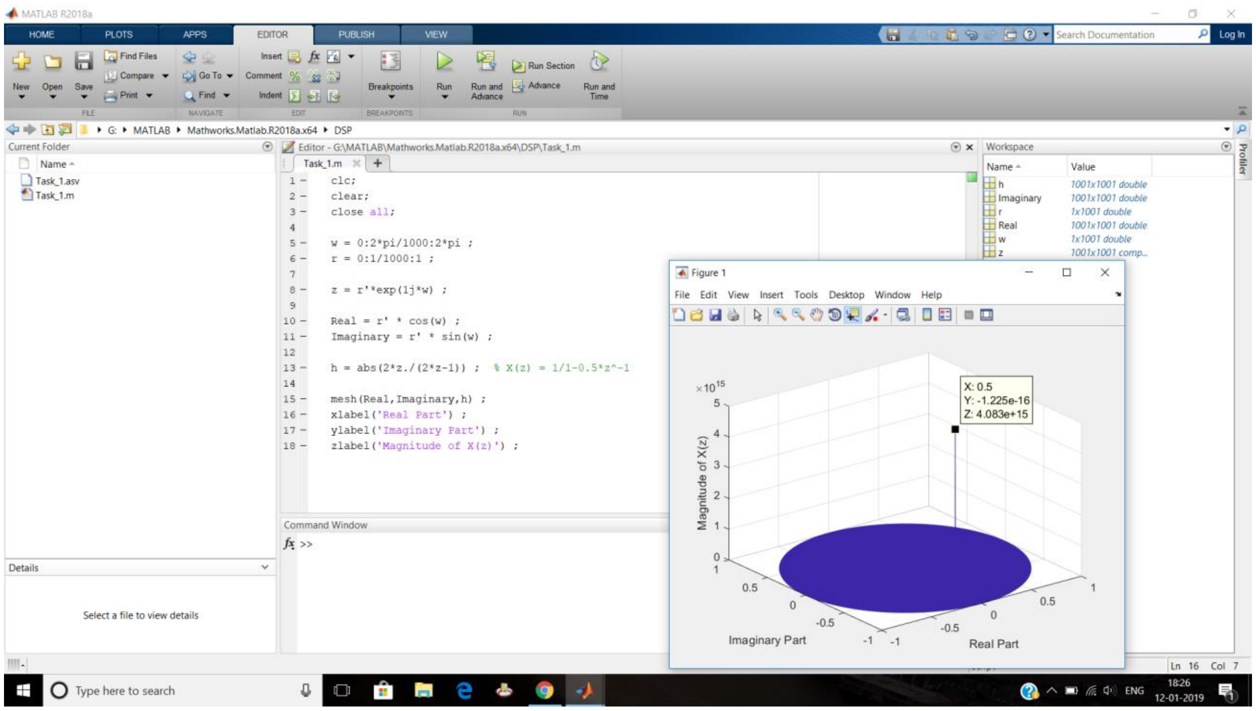
Code :

```
clc;
clear;
close all;

w = 0:2*pi/1000:2*pi ;
r = 0:1/1000:1 ;

z = r'*exp(1j*w) ;

Real = r' * cos(w) ;
Imaginary = r' * sin(w) ;

h = abs(2*z./(2*z-1)) ;  % X(z) = 1/1-0.5*z^-1
```

```
mesh(Real,Imaginary,h) ;
xlabel('Real Part') ;
ylabel('Imaginary Part') ;
zlabel('Magnitude of X(z)') ;
```

<u>Output</u> :





```
clc;
clear;
close all;

w = 0:2*pi/1000:2*pi ;
r = 0:1/1000:1 ;

z = r'*exp(1j*w) ;

Real = r' * cos(w) ;
Imaginary = r' * sin(w) ;

h = abs(2*z./(2*z-1)) ;  % X(z) = 1/1-0.5*z^-1

mesh(Real,Imaginary,h) ;
xlabel('Real Part') ;
ylabel('Imaginary Part') ;
zlabel('Magnitude of X(z)') ;
```

Name	Value
h	1001x1001 double
Imaginary	1001x1001 double
r	1x1001 double
Real	1001x1001 double
w	1x1001 double
z	1001x1001 comp...

```
X: 0.5
Y: -1.225e-16
Z: 4.083e+15
```

<u>Observation</u> :

The Magnitude of H(z) at a point in the Z-plane where a pole is present is very large , ideally it is infinite . In this case the value of Magnitude of H(z) at a point where pole is present is $1.4 * 10^7$.

LAB 2 : Frequency Response

TASK 1 :

Aim : To analyse Frequency Response depending on position of pole in the Z-Plane .

Theory:

The frequency response of a system is determined by the pole and zero locations of the transfer function *H*(*z*).

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

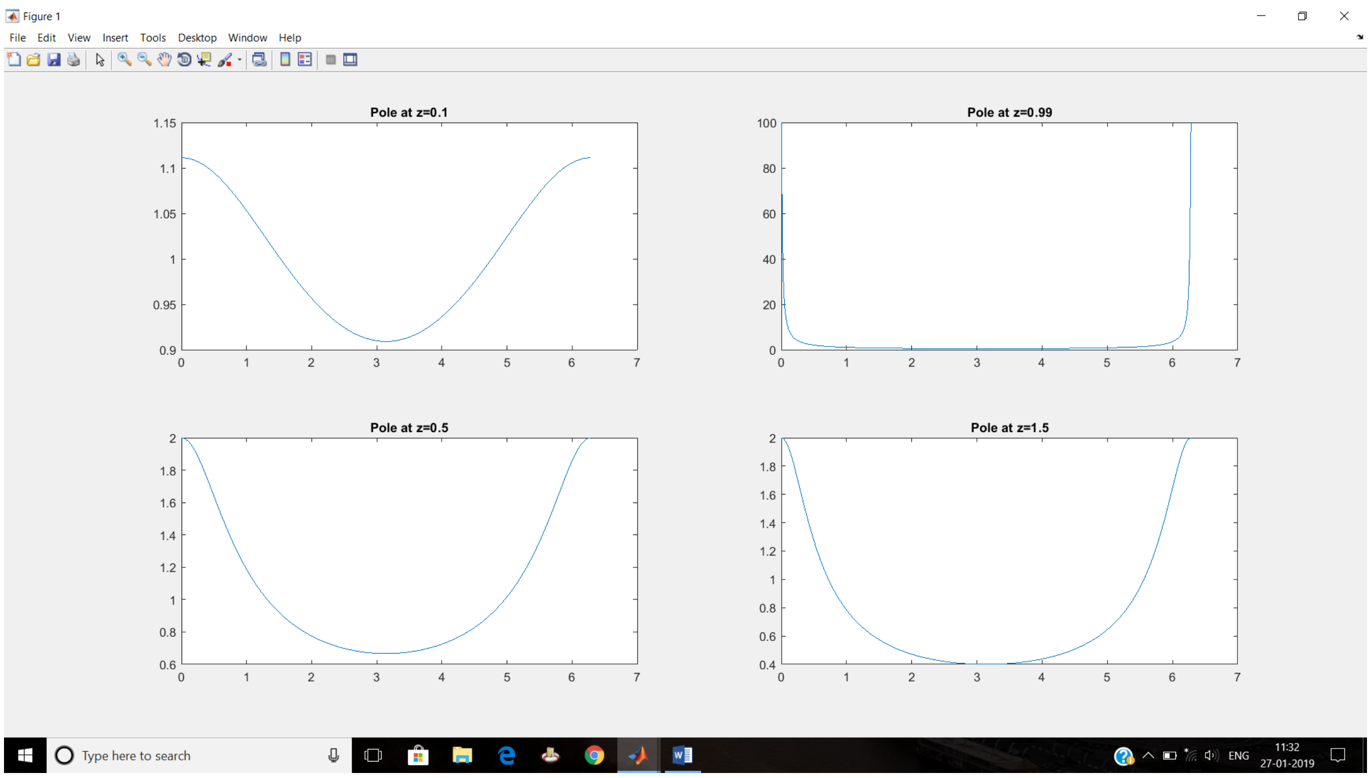
$$X(j\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$$

Code :

```
clc;
clear;
close all;

w = 0:2*pi/1000:2*pi ;
z = exp(1j*w) ;

H = abs(z./(z-0.1)) ;   %Pole at z=0.1
subplot(2,2,1);
plot(w,H);
title('Pole at z=0.1');

H_1 = abs(z./(z-0.99));   %Pole at z=0.99
subplot(2,2,2);
plot(w,H_1);
title('Pole at z=0.99')

H_2 = abs(z./(z-0.5));   %Pole at z=0.5
```

```
subplot(2,2,3);
plot(w,H_2);
title('Pole at z=0.5');

% Observe what happens if pole is beyond z=1

H_3 = abs(z./(z-1.5));    %Pole at z=1.5
subplot(2,2,4);
plot(w,H_3);
title('Pole at z=1.5');
```

<u>Output</u> :



Observation :

1) The nearer a pole (or zero) is to the point $e^{j\omega}$, the more influence that pole (or zero) yields on the magnitude response at the frequency ω.

2) To enhance the magnitude response at a frequency ω, we should place a pole close to the point $e^{j\omega}$.

3) To suppress the magnitude response at a frequency ω, we should place a zero close to the point $e^{j\omega}$.

4) A pole at a point has the opposite (reciprocal) effect of a zero at that point. Placing a zero close to a pole tends to cancel the effect of that pole on the frequency response (and vice versa).
5) For a stable system, all the poles must be located inside the unit circle.

LAB 3 : Magnitude Phase Plot

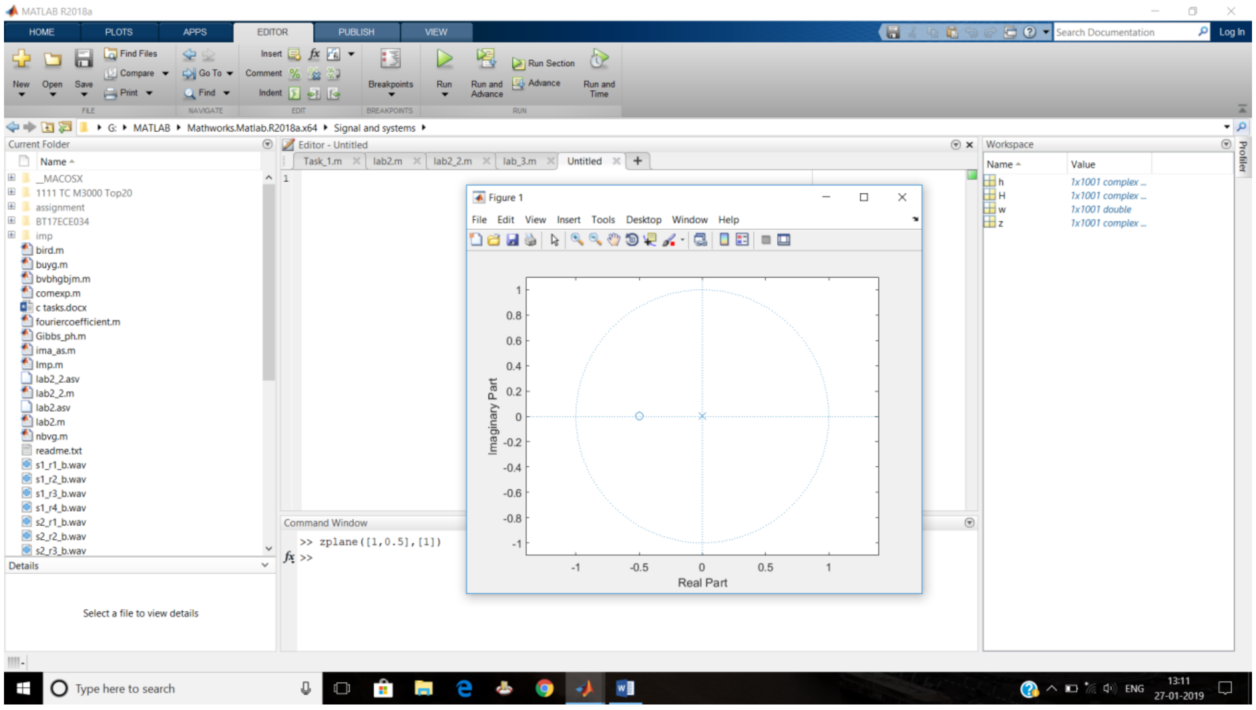
Task 1 :

Aim : To plot Pole Zero Plot of Given Rational Z-Transform using in-built MATLAB function.

1) $H(z) = 1 + 0.5\,z^{-1}$

Code :

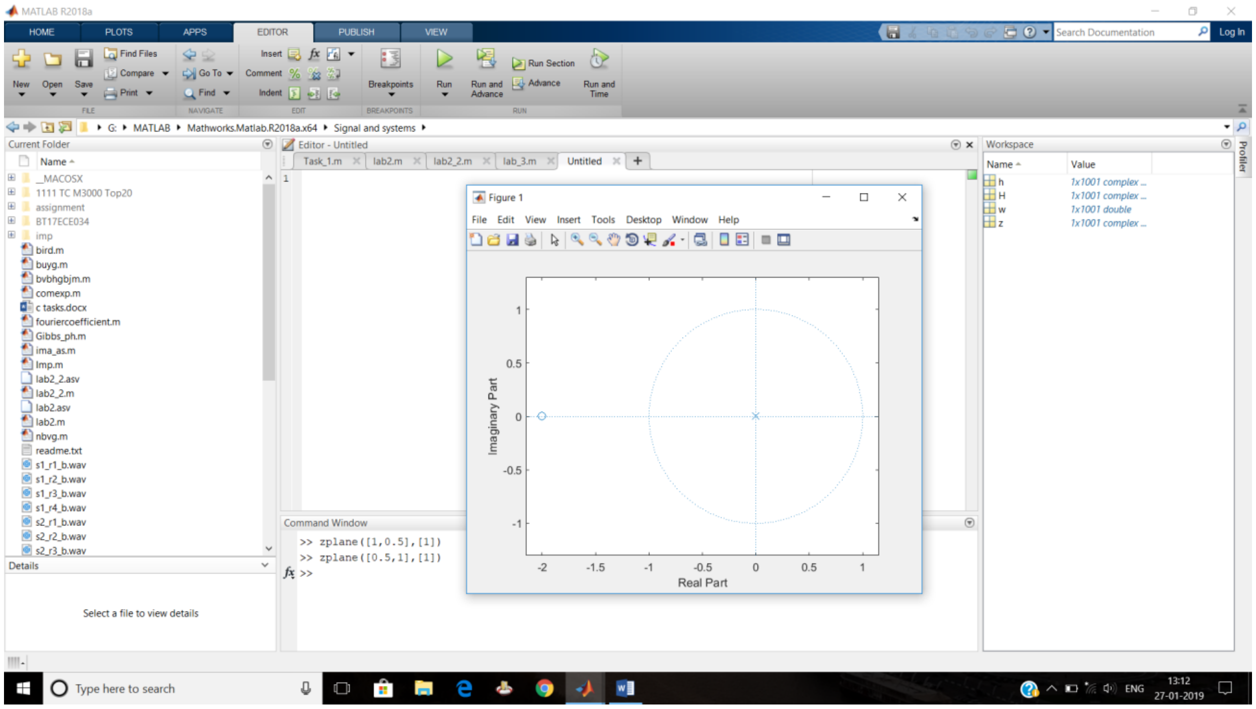
```
>> zplane([1,0.5],[1])
```



```
>> zplane([1,0.5],[1])
fx >>
```

2) $H(z) = 0.5 + z^{-1}$

```
>> zplane([0.5,1],[1])
```



Command Window

```
>> zplane([1,0.5],[1])
>> zplane([0.5,1],[1])
>>
```

Workspace

Name	Value
h	1x1001 complex ...
H	1x1001 complex ...
w	1x1001 double
z	1x1001 complex ...

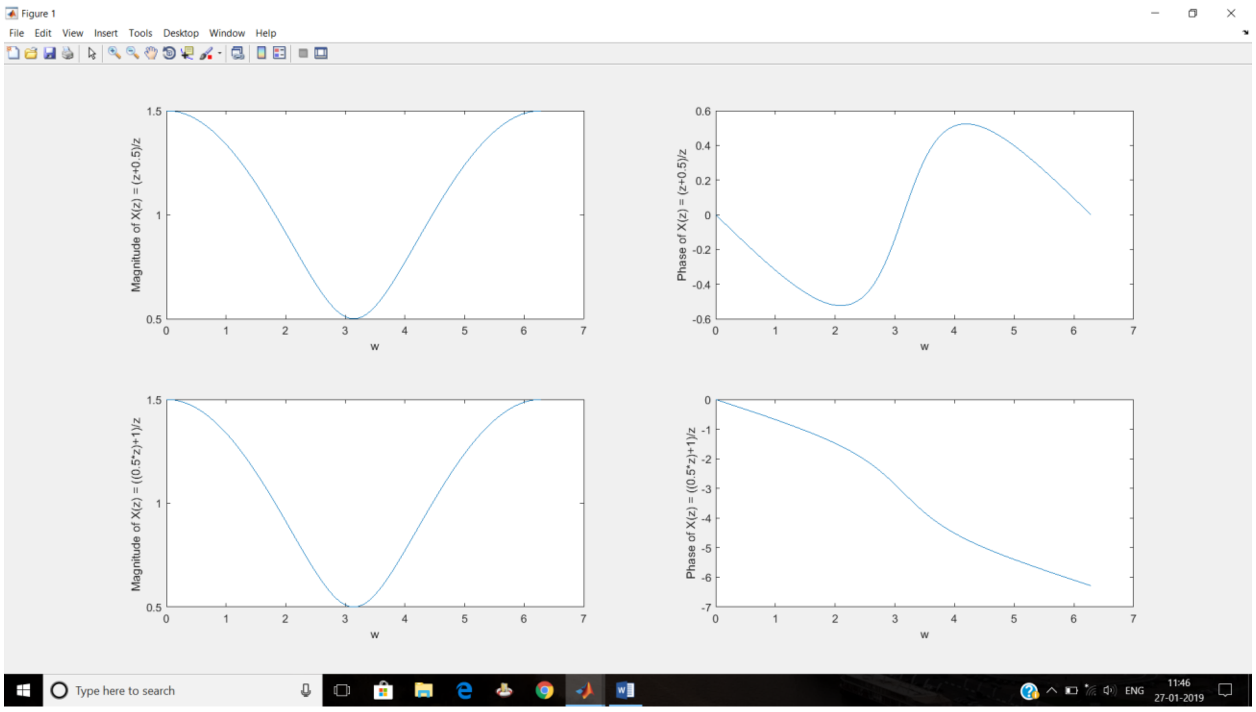
Task 2 :

Aim : To plot magnitude and phase of a given Impulse Response by finding the z transform of given Impulse Response .

Code :

```
clc;
clear;
close all;

w = 0:2*pi/1000:2*pi ;
z = exp(1j*w) ;


h = (z+0.5)./z ;    %h[n] = {1,0.5}

subplot(2,2,1);
plot(w,abs(h));
xlabel('w');
ylabel('Magnitude of X(z) = (z+0.5)/z');



subplot(2,2,2);
plot(w,phase(h));
xlabel('w');
ylabel('Phase of X(z) = (z+0.5)/z');

H = ((0.5.*z)+1)./z ;  %h[n] = {0.5,1}

subplot(2,2,3);
plot(w,abs(H));
xlabel('w');
ylabel('Magnitude of X(z) = ((0.5*z)+1)/z ');

subplot(2,2,4);
plot(w,phase(H));
xlabel('w');
ylabel('Phase of X(z) = ((0.5*z)+1)/z');
```

Output :



Observation :

1) The First Frequency Phase Plot represents The Minimum – Phase System and the Second Frequency Phase Plot represents The Maximum – Phase System .

2) If we replace factor $(z + 0.5)$ by $(\frac{1}{z} + 0.5)$ the magnitude of $|Hp(F)|$ remains unchanged, and only the phase is affected.

3) All Poles and Zeros of a Minimum-Phase System Lie Inside the Unit Circle .

4) A stable system is called mixed phase if some of its zeros lie outside the unit circle and maximum phase if all its zeros lie outside the unit circle .

LAB 4 : Discrete Fourier Transform (DFT)

Aim : To implement Discrete Fourier Transform (DFT) and Inverse Discrete Fourier Transform (IDFT) of a given input signal .

Theory:

Frequency analysis of discrete-time signals is usually and most conveniently performed on a digital computer or specially designed digital hardware. To perform frequency analysis on a discrete-time signal {*x(n)*}, we convert the time-domain sequence to an equivalent
frequency-domain representation. We know that such a representation is given by the Discrete Time Fourier transform *X(ω)* of the sequence {*x(n)*}. However, *X(ω)* is a continuous function of frequency and therefore it is not a computationally convenient representation
of the sequence {*x(n)*}.

Hence , we consider the representation of a sequence {*x(n)*} by samples of its spectrum *X(ω)*. Such a frequency-domain representation leads to the **Discrete Fourier transform (DFT)** , which is a powerful computational tool for performing frequency analysis of discrete-time signals.

DFT : $X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}$

IDFT : $x[n] = \sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}kn}$

Code :

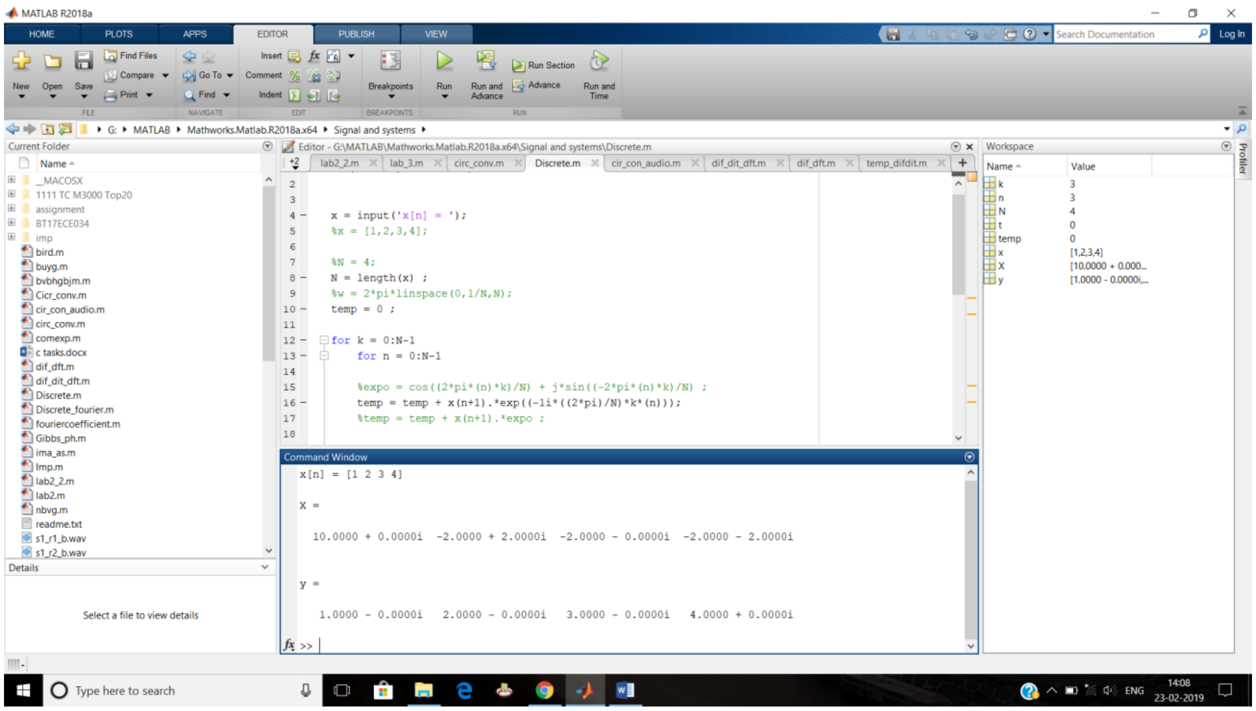
```
clc; clear; close all;


x = input('x[n] = ');
%x = [1,2,3,4];

%N = 4;
N = length(x) ;
%DFT
```

```
temp = 0 ;


for k = 0:N-1
    for n = 0:N-1

     temp = temp + x(n+1).*exp((-1i*((2*pi)/N)*k*(n)));


    end

X(k+1) = temp ;
temp = 0;
end
X



%IDFT

t = 0 ;
for n = 0:N-1
    for k = 0:N-1

    t = t + (1/N)*X(k+1).*exp((1i*2*pi*k*(n))/N);

    end
y(n+1) = t ;
t = 0 ;
end
y
```

<u>Output</u> :

1)

```
x[n] = [1 2 3 4]

X =

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


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

>>
```



```
Editor - G:\MATLAB\Mathworks.Matlab.R2018a.x64\Signal and systems\Discrete.m

x = input('x[n] = ');
%x = [1,2,3,4];

%N = 4;
N = length(x) ;
%w = 2*pi*linspace(0,1/N,N);
temp = 0 ;

for k = 0:N-1
    for n = 0:N-1

    %expo = cos((2*pi*(n)*k)/N) + j*sin((-2*pi*(n)*k)/N) ;
    temp = temp + x(n+1).*exp((-1i*((2*pi)/N)*k*(n)));
    %temp = temp + x(n+1).*expo ;

```

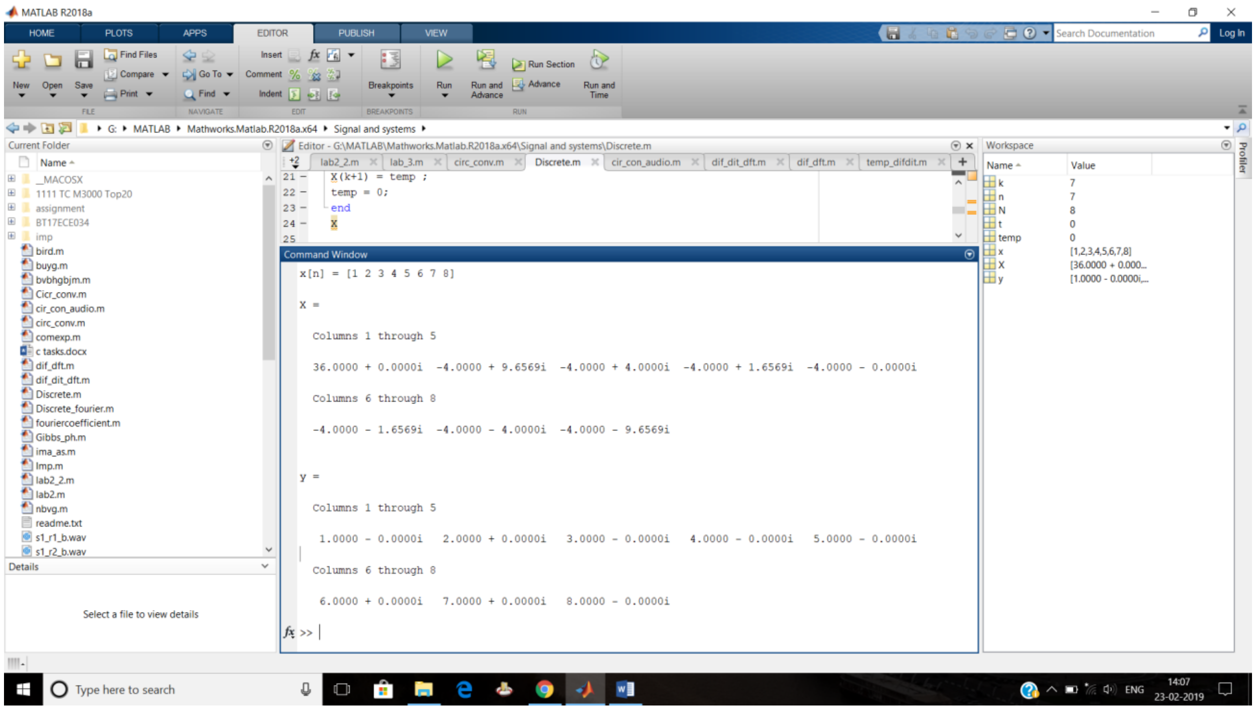
```
Command Window
x[n] = [1 2 3 4]

X =

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


y =

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

fx >>
```

Name	Value
k	3
n	3
N	4
t	0
temp	0
x	[1,2,3,4]
X	[10.0000 + 0.000...
y	[1.0000 - 0.0000i,...

2)

```
x[n] = [1 2 3 4 5 6 7 8]

X =

  Columns 1 through 5

  36.0000 + 0.0000i  -4.0000 + 9.6569i  -4.0000 + 4.0000i  -4.0000 +
1.6569i  -4.0000 - 0.0000i

  Columns 6 through 8

  -4.0000 - 1.6569i  -4.0000 - 4.0000i  -4.0000 - 9.6569i


y =

  Columns 1 through 5

   1.0000 - 0.0000i   2.0000 + 0.0000i   3.0000 - 0.0000i   4.0000 -
0.0000i   5.0000 - 0.0000i
```

```
  Columns 6 through 8

   6.0000 + 0.0000i   7.0000 + 0.0000i   8.0000 - 0.0000i

>>
```



```
    X(k+1) = temp ;
    temp = 0;
  end
  X
```

```
x[n] = [1 2 3 4 5 6 7 8]

X =

  Columns 1 through 5

  36.0000 + 0.0000i  -4.0000 + 9.6569i  -4.0000 + 4.0000i  -4.0000 + 1.6569i  -4.0000 - 0.0000i

  Columns 6 through 8

  -4.0000 - 1.6569i  -4.0000 - 4.0000i  -4.0000 - 9.6569i


y =

  Columns 1 through 5

   1.0000 - 0.0000i   2.0000 + 0.0000i   3.0000 - 0.0000i   4.0000 - 0.0000i   5.0000 - 0.0000i

  Columns 6 through 8

   6.0000 + 0.0000i   7.0000 + 0.0000i   8.0000 - 0.0000i

>>
```

Name	Value
k	7
n	7
N	8
t	0
temp	0
x	[1,2,3,4,5,6,7,8]
X	[36.0000 + 0.000...
y	[1.0000 - 0.0000i,...

<u>Observations</u> :

1) The *N*-point DFT and *N*-Point IDFT Are Periodic with Period *N* .

2) The DFT Shows Conjugate Symmetry for Real Signals . Conjugate symmetry suggests that we need compute only half the DFT values to find the entire DFT sequence .

LAB 5 : Circular Convolution

Task 1 :

Aim : To implement Circular Convolution of given input signal x[n] and Impulse response h[n] .

Theory :

The mathematical definition of convolution in discrete time domain is given by :

$$y(n) = h(n) \circledast_N x(n)$$

$$y(n) = \sum_{m=0}^{N-1} x[n]h[(n-m)]_N$$

where , *x*[n] is input signal , *h*[n] is impulse response, and *y*[n] is output of Circular Convolution.

Code :

```
clc ; clear ; close all;

x = input('x[n] : ');


h = input('h[n] : ');

% Circulatr Convolution Using Formula .
N = length(x);
temp = 0 ;
for n = 0:N-1
    for m = 0:N-1

    k = mod((n-m),N);
    temp = temp + x(m+1).*h(k+1);

    end
```

```
y(n+1) = temp ;
temp = 0;
end
y


% DFT of x[n] i.e. X[k]
t = 0 ;

for k = 0:N-1
    for p = 0:N-1

    t = t + x(p+1)*exp((-1i*2*pi*k*(p))/N);

    end

X(k+1) = t ;
t = 0;
end
X


% DFT of h[n] i.e. H[k]
tem = 0 ;

for k = 0:N-1
    for p = 0:N-1

    tem = tem + h(p+1)*exp((-1i*2*pi*k*(p))/N);

    end

H(k+1) = tem ;
tem = 0;
end
H

% Output in Frequency Domain i.e. Y[k]
Y = X.*H ;
Y


% Output in Time Domain i.e. z[n] (IDFT of Y[k])
te = 0 ;
for p = 0:N-1
    for k = 0:N-1

    te = te + (1/N)*Y(k+1).*exp((1i*2*pi*k*(p))/N);
    end
z(p+1) = te ;
te = 0 ;
end

z


% Circular Convolution of x[n] and h[n]
% It is found by taking IDFT of Y which is Multiplication of
x[n] and  h[n]
% in frequency-domain
W = ifft(fft(x).*fft(h))
```

Output :

```
x[n] : [1 2 3 4]
h[n] : [5 6 7 8]


y =

    66    68    66    60


X =

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


H =

  26.0000 + 0.0000i  -2.0000 + 2.0000i  -2.0000 - 0.0000i  -2.0000 -
2.0000i


Y =

   1.0e+02 *

   2.6000 + 0.0000i   0.0000 - 0.0800i   0.0400 + 0.0000i  -0.0000 +
0.0800i


z =
  66.0000 + 0.0000i  68.0000 + 0.0000i  66.0000 + 0.0000i  60.0000 -
0.0000i


W =

    66    68    66    60

>>
```

Observation :

1) From this experiment , we observe that circular convolution involves basically the same four steps as the ordinary *linear convolution*: *flipping* (time reversing) one sequence, *shifting* the flipped sequence, *multiplying* the two sequences to obtain a product sequence , and finally, *summing* the values of the product sequence. The basic difference between these two types of convolution is that, in circular convolution, the flipping and shifting operations are performed in a circular fashion by computing the index of one of the sequences modulo *N*.

2) In this Experiment , after N samples , i.e. after 4 samples the Nth sample and zeroth samples of linear Convolution are added, the first sample and N+1 th sample of linear Convolution are added and so on , to get the Samples of The Circular Convolution . Circular Convolution is Aliased Version of Linear Convolution in which after N samples there is aliasing with previous Samples .

<u>**Task 2**</u> :

Padding Zeros in x[n] and h[n] such that length of x[n] and h[n] is (L + M – 1) where L is the length of x[n] and M is the length of h[n] earlier to padding zeros .

<u>Output</u>:

```
x[n] : [1 2 3 4 0 0 0]
h[n] : [5 6 7 8 0 0 0]

y =

     5    16    34    60    61    52    32


X =

  Columns 1 through 5

  10.0000 + 0.0000i  -2.0245 - 6.2240i   0.3460 + 2.4791i   0.1784 - 
2.4220i   0.1784 + 2.4220i

  Columns 6 through 7

   0.3460 - 2.4791i  -2.0245 + 6.2240i


H =

  Columns 1 through 5

  26.0000 + 0.0000i  -0.0245 -14.9866i   2.3460 + 3.4423i   2.1784 - 
4.9299i   2.1784 + 4.9299i

  Columns 6 through 7

   2.3460 - 3.4423i  -0.0245 +14.9866i


Y =

   1.0e+02 *

  Columns 1 through 5

   2.6000 + 0.0000i  -0.9323 + 0.3049i  -0.0772 + 0.0701i  -0.1155 - 
0.0616i  -0.1155 + 0.0616i
```

```
  Columns 6 through 7

  -0.0772 - 0.0701i  -0.9323 - 0.3049i


z =

  Columns 1 through 5

   5.0000 - 0.0000i  16.0000 - 0.0000i  34.0000 + 0.0000i  60.0000 +
0.0000i  61.0000 - 0.0000i

  Columns 6 through 7

  52.0000 - 0.0000i  32.0000 + 0.0000i


W =

    5.0000   16.0000   34.0000   60.0000   61.0000   52.0000   32.0000

>> conv([1 2 3 4],[5 6 7 8])

ans =

     5    16    34    60    61    52    32

>>
```

Observation :

We Find that when we pad zeros to x[n] and h[n] such that length of new x[n] and h[n] is (L + M – 1) (in this case : 4 + 4 - 1 = 7) then the **Cicular Convolution** is Same as **Linear Convolution** .

LAB 6 : Filtering the Voice Signal

Aim : To Filter the voice signal using Circular Convolution .

Theory :

Filtering can be accomplished either in the time domain or in the frequency domain. In the time-domain approach, we compute the filter output $y[n]$ directly from a given input $x[n]$ and the known filter impulse response $h[n]$. This output is equal to the linear convolution of $x[n]$ with $h[n]$. It is also equal to the circular convolution of suitably zero-padded signals $x_{pad}[n]$ and $h_{pad}[n]$.

Code :

```
clc;
clear;
close all ;


a = audiorecorder (8000,8,1) ;
disp('Start Speaking .');
recordblocking (a,5);
disp('End of Recording .');
play(a);
b = getaudiodata(a);
subplot(2,2,1);
plot(b);

h = ones(1,10)./10 ;
subplot(2,2,2);
plot(h);



x = b + 0.01*randn(length(b),1) ;
subplot(2,2,3);
plot(x);
%sound(x,8000);

y = cconv(h,x) ;
subplot(2,2,4);
plot(y) ;
%sound(y,8000);
```

```
for i = 0 : length(a)
    mse = (y(i+1) - b(i+1))^2 ;
end
mse
```

Output :

```
mse =

   2.4443e-10
```

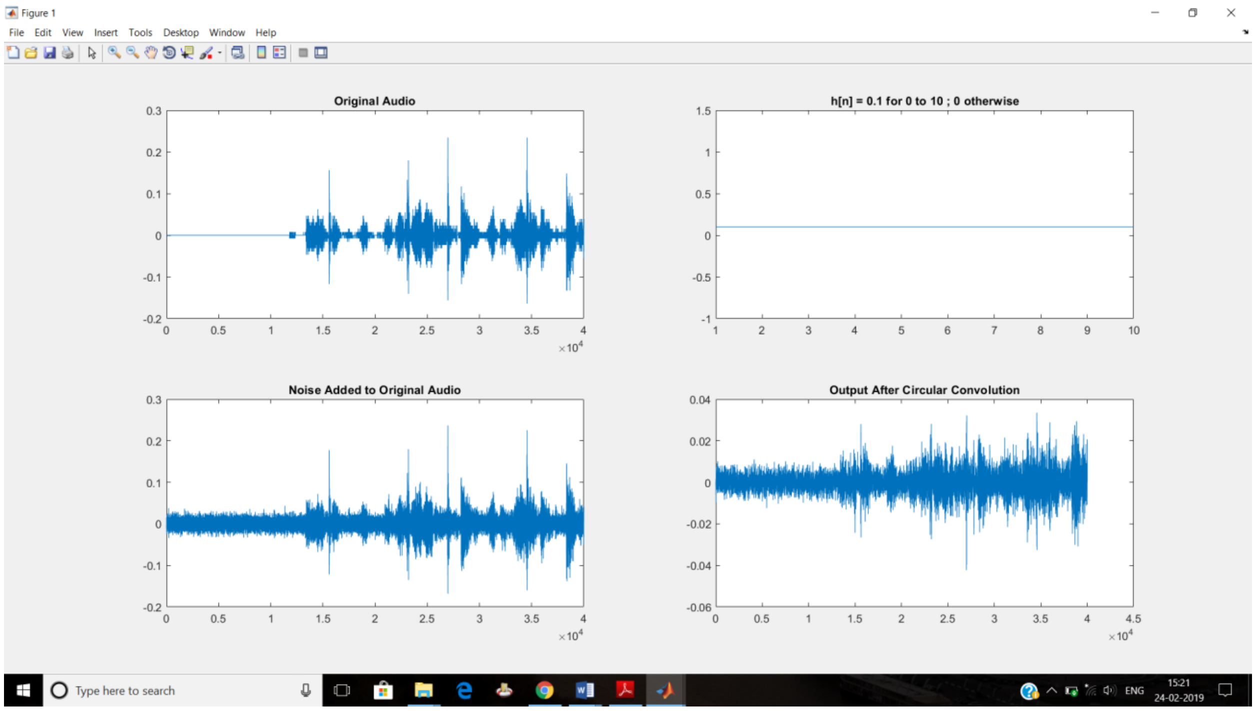


Figure 1
File Edit View Insert Tools Desktop Window Help
Original Audio
h[n] = 0.1 for 0 to 10 ; 0 otherwise
Noise Added to Original Audio
Output After Circular Convolution
Type here to search
ENG
15:21
24-02-2019

Observation :

The Output After Circular Convolution had Amplitude of Noise Significantly Reduced and the output was the Original Audio with its Amplitude been Reduced and the Mean Squared Error (MSE) was found to be 2.4443e-10 which is 0.0011 % .

LAB 7 : DIT-DFT and DIF-DFT

Aim : To implement Decimation in Time DFT and Decimation in Frequency DFT.

Theory :

Basically, DFT compute the sequence{*X(k)*} of *N* complex-valued numbers given another sequence of data {*x(n)*} of length *N*, according to the formula –

$$X[k] = \sum_{n=0}^{N-1} x[n]W^{kn}$$

$$W = e^{-j\frac{2\pi}{N}kn}$$

1) Complex Multiplications : N^2 .
2) Complex Additions : $N(N-1)$.

1) Decimation in Time DFT (DIT DFT) :

$$X[k] = \sum_{n=0}^{(\frac{N}{2})-1} x[2n]W^{kn} + W^k \sum_{n=0}^{(\frac{N}{2})-1} x[2n+1]W^{kn}$$

2) Decimation in Frequency DFT (DIF DFT) :

$$X[k] = \sum_{n=0}^{\frac{N}{2}-1}(x[n] + (-1)^k x[n+\frac{N}{2}])W^{kn}$$

Code :

```
clc ; clear ; close all ;

x = input('Enter the Sequence : ');
N = length(x) ;

W = exp(-1i*(2*pi/N)) ;
temp = 0;

%DIF-DFT
```

```
for k = 0:N-1
    for n = 0:(N/2 - 1)
            % Using Formula of DIF - DFT
            temp =  temp + (x(n+1) + ((-
1)^k)*x(n+1+(N/2)))*(W^(k*n)) ;
    end
    X(k+1) = temp ;
    temp = 0;
end
X


%DIT-DFT
t = 0 ;
for k = 0:N-1
    for n = 0:(N/2 - 1)
            % Using Formula of DIT - DFT
            t = t + x(2*n+1)*W^(2*k*n) +
x(2*n+2)*W^(2*k*n)*W^(k) ;

    end
    Y(k+1) = t ;
    t = 0;
end
Y
```

<u>Output</u> :

```
Enter the Sequence : [1 2 3 4 5 6 7 8]

X =

  Columns 1 through 5

  36.0000 + 0.0000i  -4.0000 + 9.6569i  -4.0000 + 4.0000i  -4.0000 +
1.6569i  -4.0000 - 0.0000i

  Columns 6 through 8

  -4.0000 - 1.6569i  -4.0000 - 4.0000i  -4.0000 - 9.6569i


Y =

  Columns 1 through 5
  36.0000 + 0.0000i  -4.0000 + 9.6569i  -4.0000 + 4.0000i  -4.0000 +
1.6569i  -4.0000 - 0.0000i

  Columns 6 through 8

  -4.0000 - 1.6569i  -4.0000 - 4.0000i  -4.0000 - 9.6569i

>>
```

<u>Observation</u> :

I. Direct Computation of DFT Requires :
Complex Multiplications : N^2 .

II. Computation of DIT DFT Requires :
Complex Multiplications : $\frac{N}{2}\log_2 N$.

III. Computation of DIT DFT Requires :

Complex Multiplications : $\frac{N}{2}\log_2 N$.

IV. Computation of DIT DFT and DIF FFT Requires Less Number of Complex Multiplication than compared with direct Computation of DFT . Hence , the Computation of DIT DFT and DIF DFT is faster than direct computation of DFT .

LAB 8 : WINDOWING TECHNIQUE

Aim : To implement FIR Filter using Window technique .

Theory :

The desired frequency response specification is given by $H_d(\omega)$ and the corresponding unit sample response $h_{ideal}(n)$.

$$H_d(\omega) = \begin{cases} 1\,, & \omega < |\omega_c| \\ x, & otherwise \end{cases}$$

$$h_{ideal}[n] = \begin{cases} \dfrac{\sin(w_c n\,)}{\pi n}, & n \neq 0 \\ \dfrac{\omega_c}{\pi}\,, & n = 0 \end{cases}$$

In general , $h_{ideal}(n)$ obtained is infinite in duration and must be truncated at some point, say at $n = M - 1$, to yield an FIR
filter of length M. Truncation of $h_{ideal}(n)$ to a length M−1 is equivalent to multiplying
$h_{ideal}(n)$ by a window function w[n] .

n : $\frac{-(M-1)}{2}\ to\ \frac{(M-1)}{2}$

NAME	EXPRESSION
RECTANGULAR	1
HAMMING	$0.54 + 0.46 \cos\frac{2\pi n}{M-1}$
HANNING	$0.5 + 0.5 \cos\frac{2\pi n}{M-1}$
BLACKMAN	$0.42 + 0.5 \cos\frac{2\pi n}{M-1} + 0.08 \cos\frac{4\pi n}{M-1}$
COS	$\cos\frac{\pi n}{M-1}$

Code :

```
clc ; clear ; close all ;

wc = pi/2 ;     % Cutoff Frequency
M = 81 ;        % Order of FIR Filter
l = 500 ;       % Length oh Ideal Impulse Response

omega = linspace(-pi,pi,l+1) ;
omega1 = linspace(-pi,pi,M) ;

% h_ideal[n] = sin(wc*n)/(pi*n) for n != 0   ;   wc/pi  for n =
0 ;

for n = -(l/2):(l/2)

    if n == 0
        h_ideal(n+(l/2)+1) = wc/pi ;
    else
        h_ideal(n+(l/2)+1) = sin(wc*n)/(pi*n) ;
    end

end
h_ideal;
p = linspace(-(l/2),(l/2),(l+1));
figure(1)
subplot(3,2,1)
stem(p,h_ideal)
title('h_ideal[n]')

figure(2)
subplot(3,2,1)
plot(omega,abs(fft(h_ideal)))


% h_d[n] after Truncating the Original Impulse Response .
for n = -((M-1)/2):((M-1)/2)

    if n == 0
        h_d(n+((M-1)/2)+1) = wc/pi ;
    else
        h_d(n+((M-1)/2)+1) = sin(wc*n)/(pi*n) ;
    end
```

```
end
h_d;

%figure(3)
%subplot(3,2,2)
%plot(omega1,fft(h_d))



k = linspace(-((M-1)/2),((M-1)/2),M) ;



%Rectangular Window
%W[n] = 1
W = ones(1,M);
h = h_d.*W;

figure(1)
subplot(3,2,2)
stem(k,h)
title('h[n] By Rectangular Window')

figure(2)
subplot(3,2,2)
plot(omega1,abs(fft(h)))
title('H(w) after applying Rectangular Window')


% HAMMING WINDOW
% W[n] = 0.54 + 0.46*cos(2*pi*n/(M-1))

W = 0.54 + 0.46*cos(2*pi.*k/(M-1));
h = h_d .* W;
figure(1)
subplot(3,2,3)
stem(k,h)
title('h[n] By HAMMING Window')

figure(2)
subplot(3,2,3)
plot(omega1,abs(fft(h)))
title('H(w) after applying HAMMING Window')



% HANNING WINDOW
% W[n] = 0.54 + 0.5*cos(2*pi*n/(M-1))
W = 0.54 + 0.5*cos(2*pi.*k/(M-1));
```

```
h = h_d .* W;
figure(1)
subplot(3,2,4)
stem(k,h)
title('h[n] By HANNING Window')

figure(2)
subplot(3,2,4)
plot(omega1,abs(fft(h)))
title('H(w) after applying HANNING Window')



% BLACKMAN WINDOW
% W[n] = 0.42 + 0.5*cos(2*pi*n/(M-1)) + 0.08*cos(4*pi*n/(M-1))
W =  0.42 + 0.5*cos(2*pi*k/(M-1)) + 0.08*cos(4*pi*k/(M-1));
h = h_d .* W;
figure(1)
subplot(3,2,5)
stem(k,h)
title('h[n] By BLACKMAN Window')

figure(2)
subplot(3,2,5)
plot(omega1,abs(fft(h)))
title('H(w) after applying BLACKMAN Window')


% cos WINDOW
% W[n] = cos(pi*n/(M-1))
W = cos(pi*k/(M-1));
h = h_d .* W;
figure(1)
subplot(3,2,6)
stem(k,h)
title('h[n] By cos Window')

figure(2)
subplot(3,2,6)
plot(omega1,abs(fft(h)))
title('H(w) after applying cos Window')
```

Output :

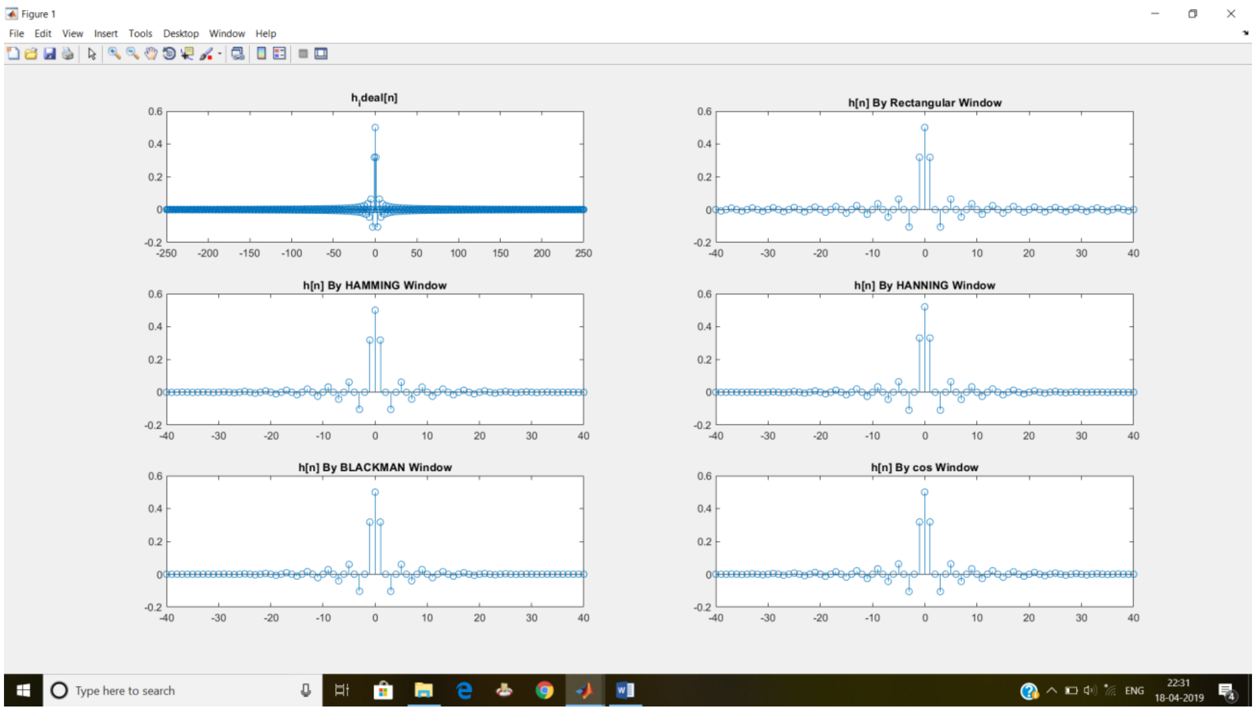


Figure 1
File Edit View Insert Tools Desktop Window Help
h_ideal[n]
h[n] By Rectangular Window
h[n] By HAMMING Window
h[n] By HANNING Window
h[n] By BLACKMAN Window
h[n] By cos Window
Type here to search
ENG
22:31
18-04-2019

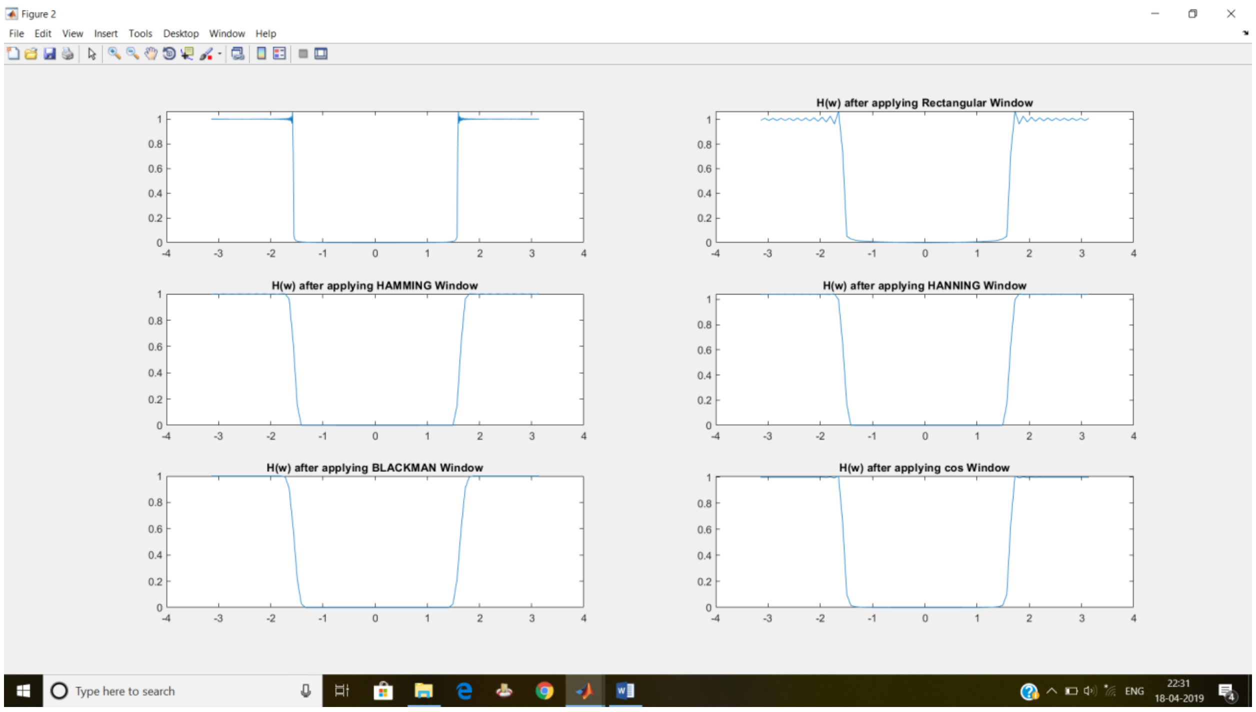


Figure 2
File Edit View Insert Tools Desktop Window Help
H(w) after applying Rectangular Window
H(w) after applying HAMMING Window
H(w) after applying HANNING Window
H(w) after applying BLACKMAN Window
H(w) after applying cos Window
Type here to search
ENG
22:31
18-04-2019

Observation :

1) There are Ripples at the discontinuity in the Frequency domain .
   The Rectangular Window has large oscillations at the discontinuity but are reduced as compared to the Original Impulse Response .

2) Cos window has less ripples as compared to that of Rectangular Window . Hamming window has less ripples as compared to that of Cos Window and Hanning Window has almost same amount of ripples as that of Hamming window .

3) Blackman window results in very less ripples at the discontinuity.

4) Thus , window function reduces the ripples at the band edges ,i.e. , at the discontinuity .

LAB 9 : SESSIONAL 2 QUESTION PAPER

Question : The First few values of impulse response sequence of Linear Phase Filter are h[n] = {2,2,-3,4,1,…} .

(a) Determine complete sequence for all possible types of linear phase FIR filter assuming smallest length of h[n] .

(b) Determine possible nature of frequency response for each type .

Theory :

Type of Filter	Length	Symmetry
Type - 1	ODD	EVEN
Type – 2	EVEN	EVEN
Type – 3	ODD	ODD
Type - 4	EVEN	ODD

Code :

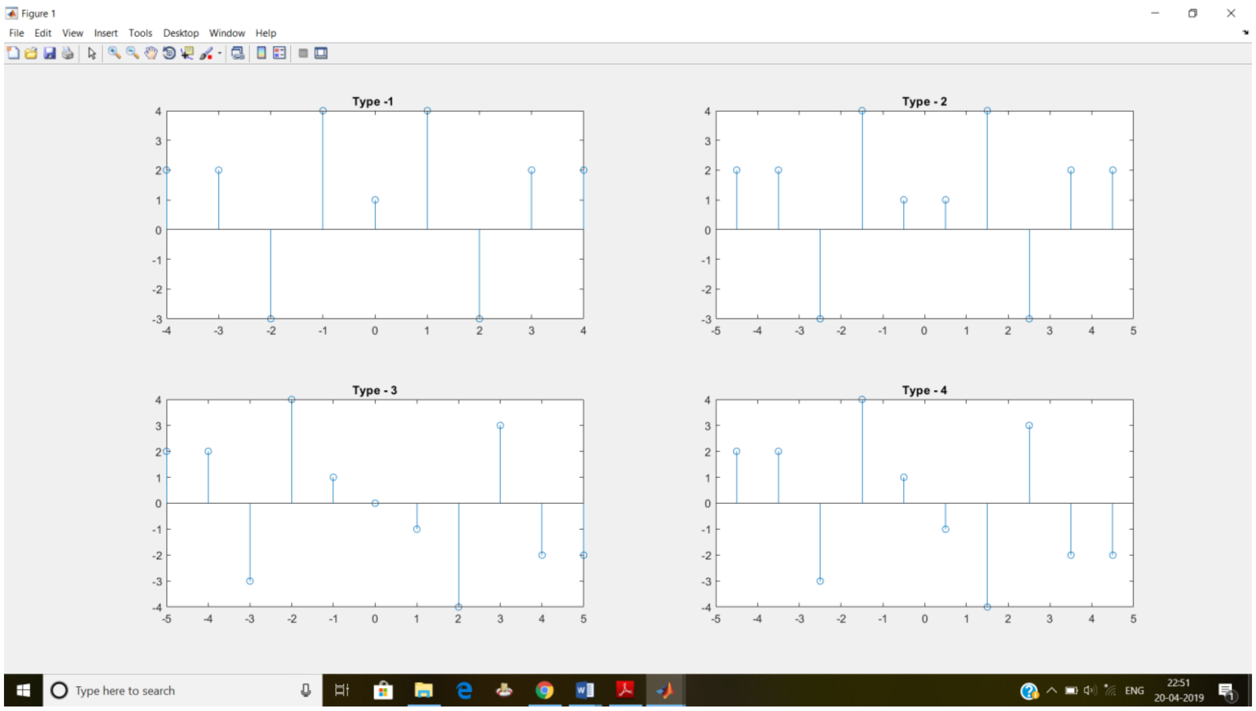
```
clc ; clear ; close all ;


% Type - 1

h1 = [2,2,-3,4,1,4,-3,2,2]


length_h1 = length(h1)

M1 = length_h1 - 1
n1 = -((length_h1 - 1)/2) : 1 : ((length_h1 - 1)/2)

figure(1)
subplot(2,2,1)
stem(n1,h1)
title('Type -1 ')


figure(2)
subplot(2,2,1)
[H1 w1] = freqz(h1) ;
plot(w1,abs(H1))
title('Type -1 ')
```

```
% Type - 2

h2 = [2,2,-3,4,1,1,4,-3,2,2]


length_h2 = length(h2)

M2 = length_h2 - 1
n2 = -((length_h2 - 1)/2) : 1 : ((length_h2 - 1)/2)

figure(1)
subplot(2,2,2)
stem(n2,h2)
title('Type - 2')

figure(2)
subplot(2,2,2)
[H2 w2] = freqz(h2)
plot(w2,abs(H2))
title('Type -2 ')



% Type - 3

h3 = [2,2,-3,4,1,0,-1,-4,3,-2,-2]


length_h3 = length(h3)

M3 = length_h3 - 1
n3 = -((length_h3 - 1)/2) : 1 : ((length_h3 - 1)/2)

figure(1)
subplot(2,2,3)
stem(n3,h3)
title('Type - 3')


figure(2)
subplot(2,2,3)
[H3 w3] = freqz(h3)
plot(w3,abs(H3))
title('Type -3 ')
```

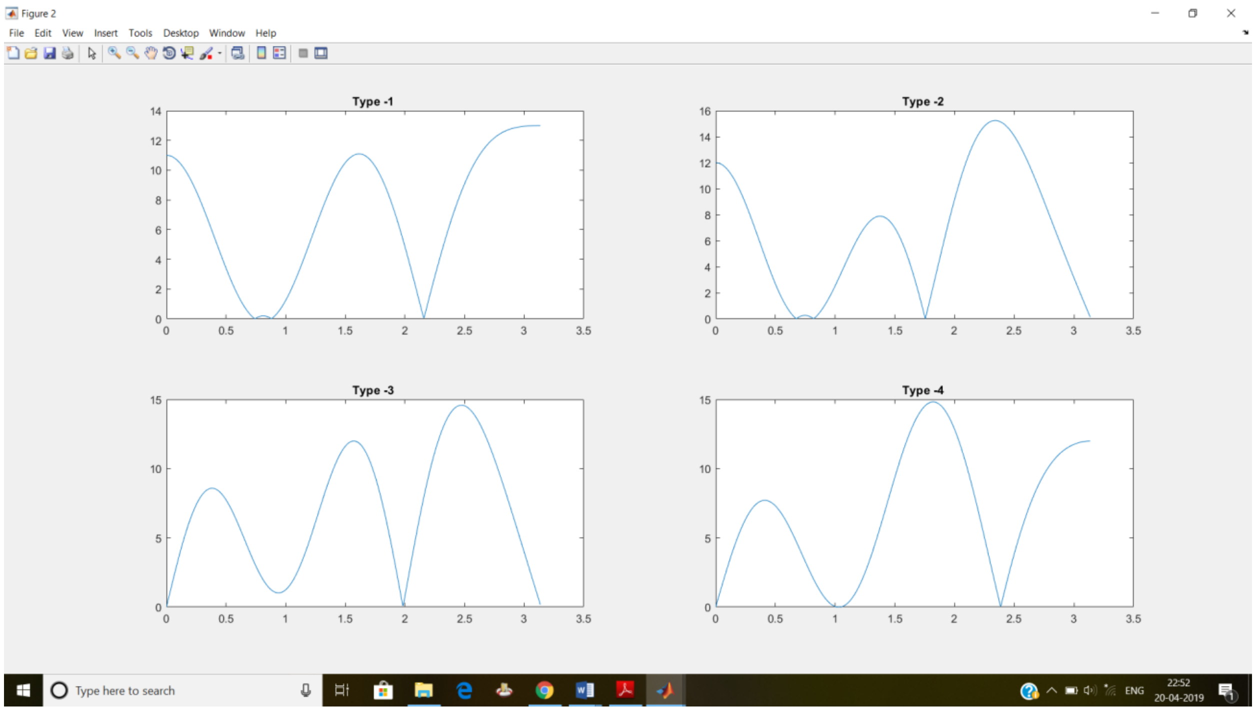
```
% Type - 4

h4 = [2,2,-3,4,1,-1,-4,3,-2,-2]


length_h4 = length(h4)

M4 = length_h4 - 1
n4 = -((length_h4 - 1)/2) : 1 : ((length_h4 - 1)/2)

figure(1)
subplot(2,2,4)
stem(n4,h4)
title('Type - 4')


figure(2)
subplot(2,2,4)
[H4 w4] = freqz(h4)
plot(w4,abs(H4))
title('Type -4 ')


figure(3)
subplot(2,2,1)
zplane([2,2,-3,4,1,4,-3,2,2],[1])

figure(3)
subplot(2,2,2)
zplane([2,2,-3,4,1,1,4,-3,2,2],[1])

figure(3)
subplot(2,2,3)
zplane([2,2,-3,4,1,0,-1,-4,3,-2,-2],[1])

figure(3)
subplot(2,2,4)
zplane([2,2,-3,4,1,-1,-4,3,-2,-2],[1])
```

Output :

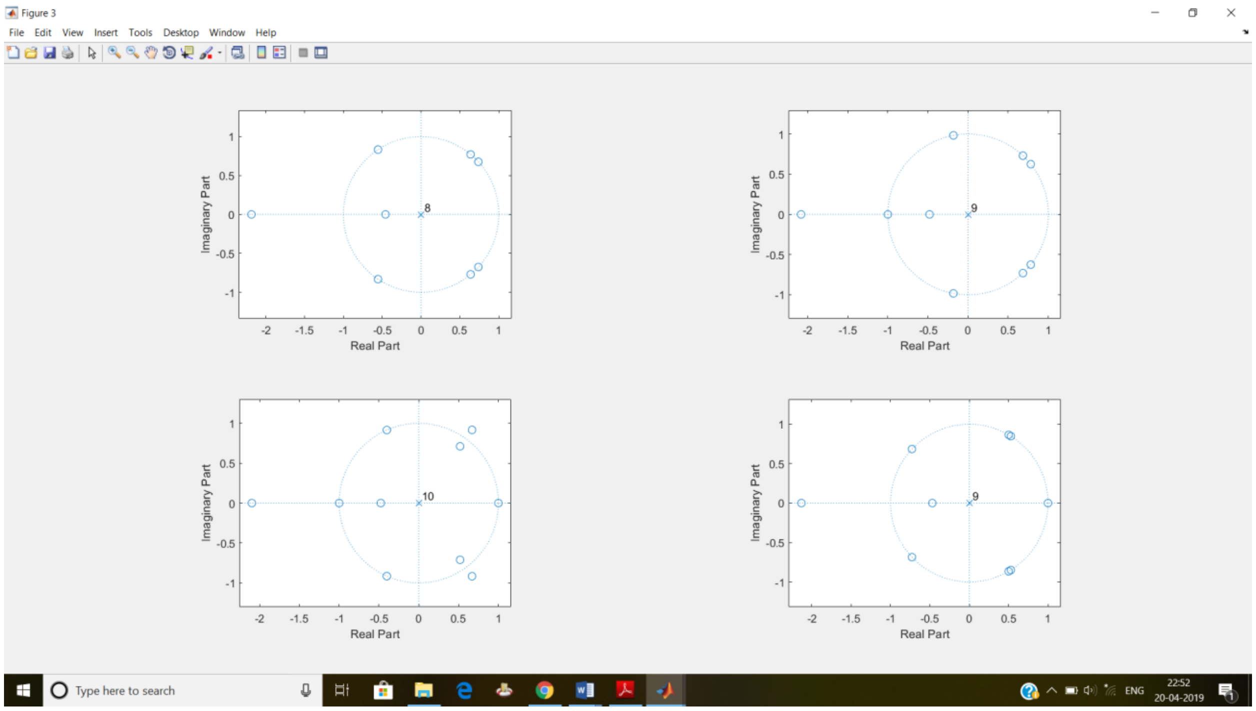
1) Types of Linear Phase Filters



2) Frequency Response



3) Pole Zero Plots



Observations :

1) Type – 1 Filter has no zero at ω = 0 and ω = π .
Hence , Type – 1 Filter can be a Band Pass Filter .

2) Type – 2 Filter has a zero at ω = π .
Hence , Type – 2 Filter can be a Low Pass Filter .

3) Type – 3 Filter has zero at both ω = 0 and ω = π .
Hence , Type – 3 Filter can be a Band Stop Filter .

4) Type – 4 Filter has a zero at ω = 0 .
Hence , Type – 4 Filter can be a High Pass Filter .

LAB 10 : Filter Design

Question : Take two sinusoids 10 Hz (x1(t)) and 100 Hz (x1(t)) with sampling frequency 1 kHz .

Task 1 :

Aim : To design an appropriate FIR Filter to filter out each of two signals .

Code :

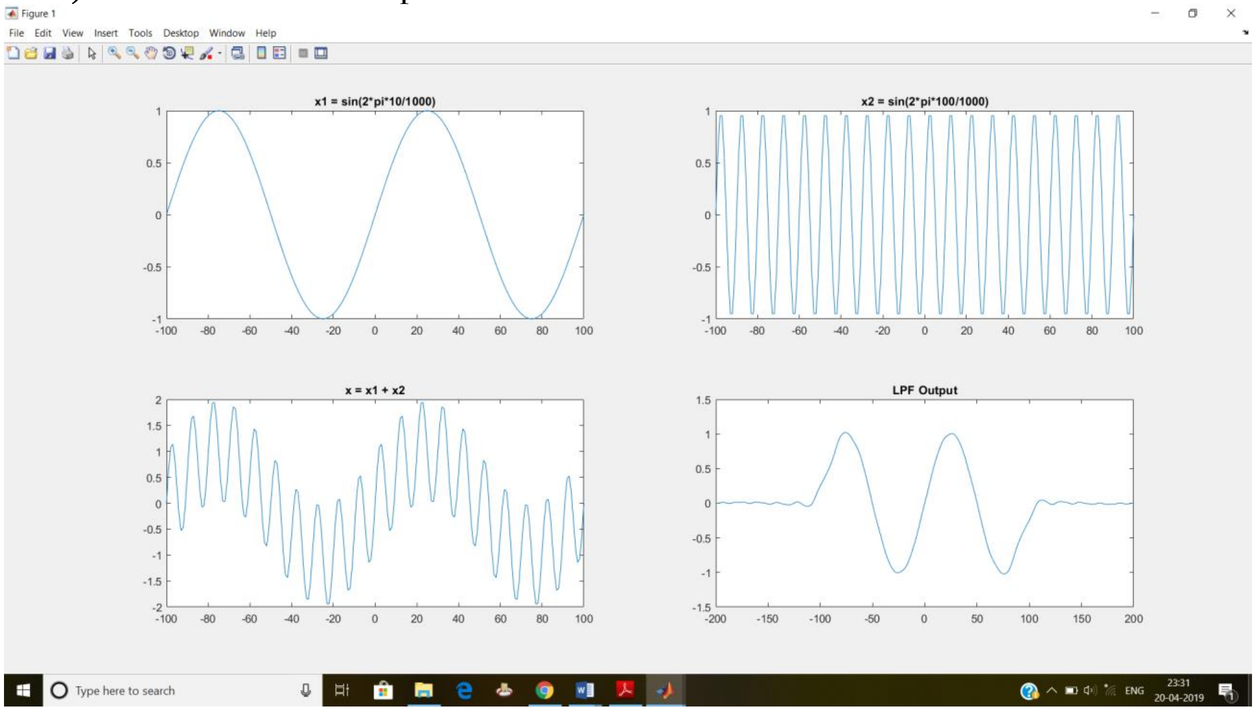
```
clc; clear; close all ;

n = -100:100 ;

f1 = 10 ;
f2 = 100 ;
fs = 1000 ;

x1 = sin(2*pi*10*n/1000) ;
x2 = sin(2*pi*100*n/1000) ;

x = x1 + x2 ;
%% LOW PASS FILTER
wc = ((pi/50) + (pi/5))/2 ;
% Cutoff Frequency for LPF to filter 10 Hz freq.

l = 200 ;

% Impulse Response for a Low Pass Filter .
for m = -(l/2):(l/2)

    if m == 0
        h_ideal(m+(l/2)+1) = wc/pi ;
    else
        h_ideal(m+(l/2)+1) = sin(wc*m)/(pi*m) ;
    end

end
h_ideal;

% Output of Low Pass Filter
y = conv(x,h_ideal);
```

```
subplot(2,2,1)
plot(n,x1)
title('x1 = sin(2*pi*10/1000)')

subplot(2,2,2)
plot(n,x2)
title('x2 = sin(2*pi*100/1000)')

subplot(2,2,3)
plot(n,x)
title('x = x1 + x2')

n1 = -200:200 ;
subplot(2,2,4)
plot(n1,y)
title('LPF Output')



%% HIGH PASS FILTER

figure(2)

l = 200 ;
wc1 = 0.6 ;
% Cutoff Frequency for HPF to filter 100 Hz freq.

% Impulse Response for a High Pass Filter .
for m = -(l/2):(l/2)

    if m == 0
        h1_ideal(m+(l/2)+1) = 1 - (wc1/pi) ;
    else
        h1_ideal(m+(l/2)+1) = -sin(wc1*m)/(pi*m) ;
    end

end
h1_ideal;


subplot(2,2,1)
plot(n,x1)
title('x1 = sin(2*pi*10/1000)')

subplot(2,2,2)
plot(n,x2)
title('x2 = sin(2*pi*100/1000)')
```

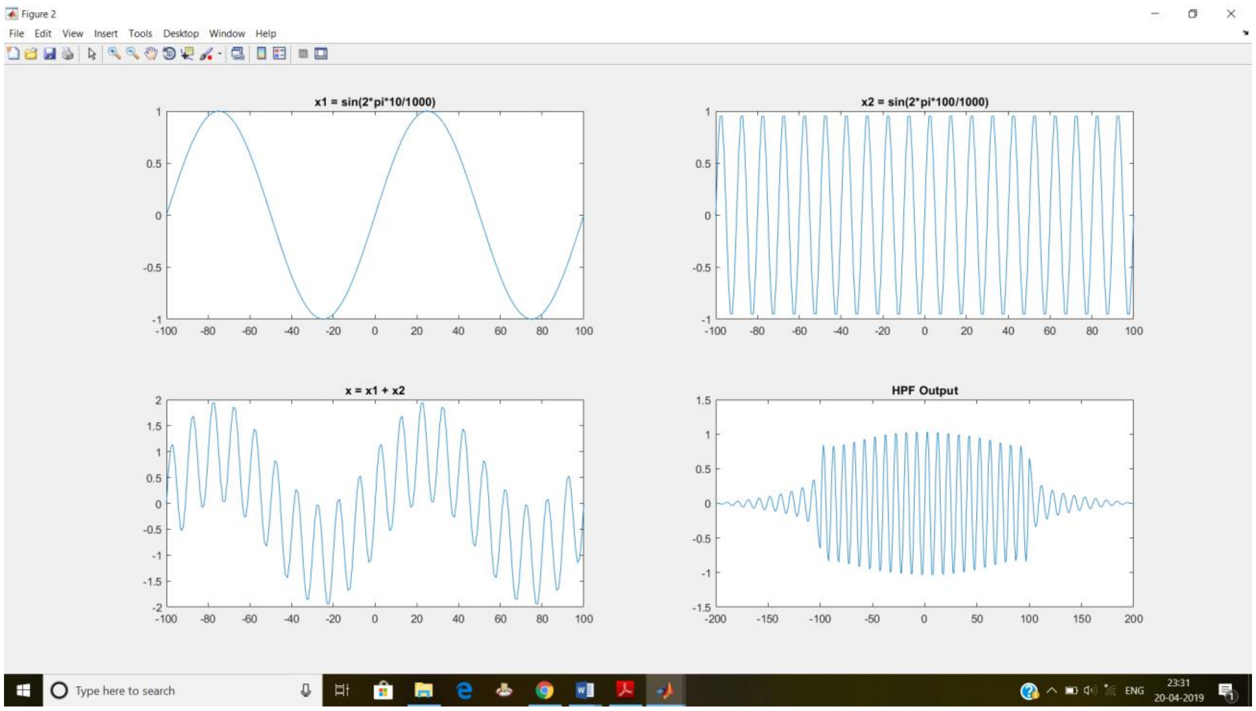
```
subplot(2,2,3)
plot(n,x)
title('x = x1 + x2')

% Output of High Pass Filter
y1 = conv(h1_ideal,x);
subplot(2,2,4)
plot(n1,y1)
title('HPF Output')
```

<u>Output</u> :

1) Low Pass Filter Output



1) High Pass Filter Output



<u>Observations </u>:

1) Convolution of Original signal x(t) and the Impulse Response of Ideal Low Pass Filter has the output with low frequency i.e. 10 Hz frequency is filtered out and the higher frequencies are attenuated .

2) Convolution of Original signal x(t) and the Impulse Response of Ideal High Pass Filter has the output with high frequency i.e. 100 Hz frequency is filtered out and the lower frequencies are attenuated .

Task 2 :

Aim : To design an appropriate IIR Filter to filter out each of two signals.

Code :

```
clc; close all; clear all;

w1=2*pi*10;
w2=2*pi*100;
t=0:1/1000:1;
x1=sin(w1*t);
x2=sin(w2*t);
x=x1+x2;

% Low Pass Filter
Wp = 10/1000;
Ws = 20/1000;
Rp = 1.25;
Rs = 15;
[N,Wc] = buttord(Wp,Ws,Rp,Rs);
[b,a]=butter(N,Wc);
[h,w]=freqz(b,a);
figure(1);
freqz(b,a)


% High Pass Filter
Wp1 = 100/1000;
Ws1 = 110/1000;
Rp1 = 1.25;
Rs1 = 15;
[N1,Wn1] = buttord(Wp1,Ws1,Rp1,Rs1);
[b1,a1]=butter(N1,Wn1,'high');
[h1,n1]=freqz(b1,a1);
figure(2);
freqz(b1,a1);


figure(3);
subplot(3,1,1)
plot(x)
title(['Original Signal'])

subplot(3,1,2)
plot(t,filter(b,a,x))
title(['Filtering 10Hz Signal (LPF)'])
```

```
subplot(3,1,3)
plot(t,filter(b1,a1,x))
title(['Filtering 100Hz Signal (HPF)'])
```

<u>Output</u> :

1) LPF Frequency Response

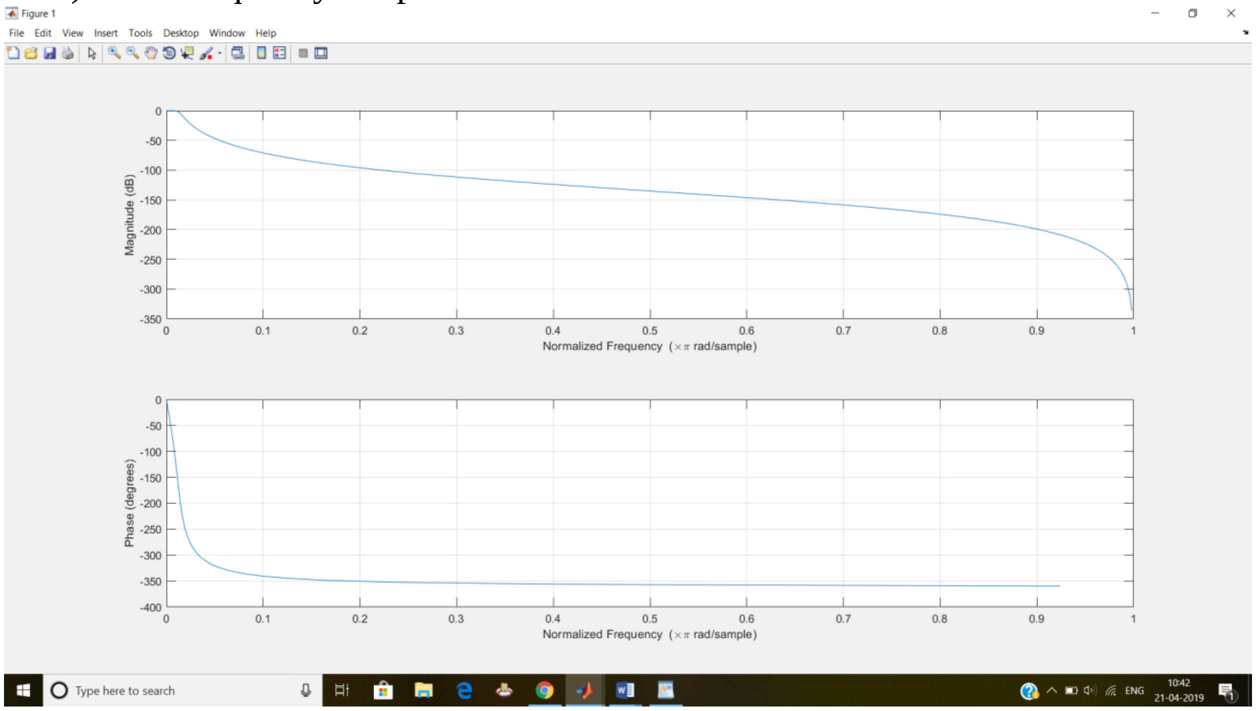


Figure 1
File Edit View Insert Tools Desktop Window Help
Magnitude (dB)
Normalized Frequency (×π rad/sample)
Phase (degrees)
Normalized Frequency (×π rad/sample)
Type here to search
ENG
10:42
21-04-2019

2) HPF Frequency Response

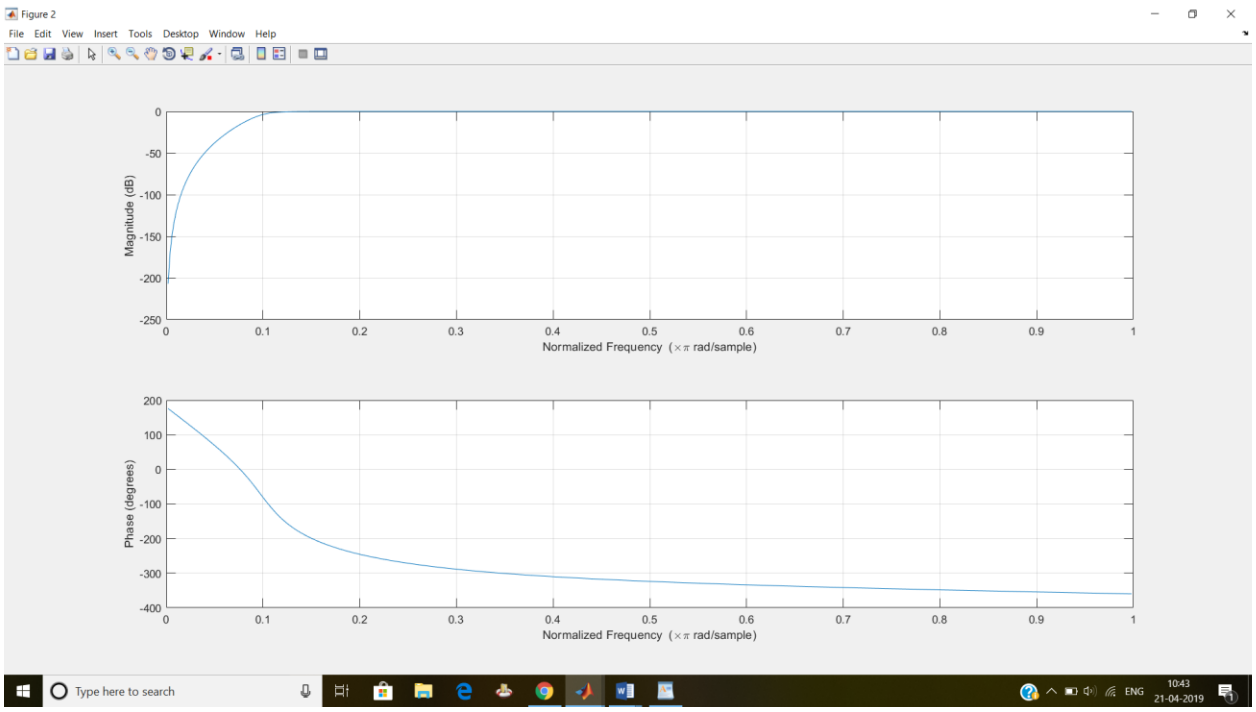


Figure 2
File Edit View Insert Tools Desktop Window Help
Magnitude (dB)
Normalized Frequency (×π rad/sample)
Phase (degrees)
Normalized Frequency (×π rad/sample)
Type here to search
ENG
10:43
21-04-2019

3) Original Signal and Output in Time Domain

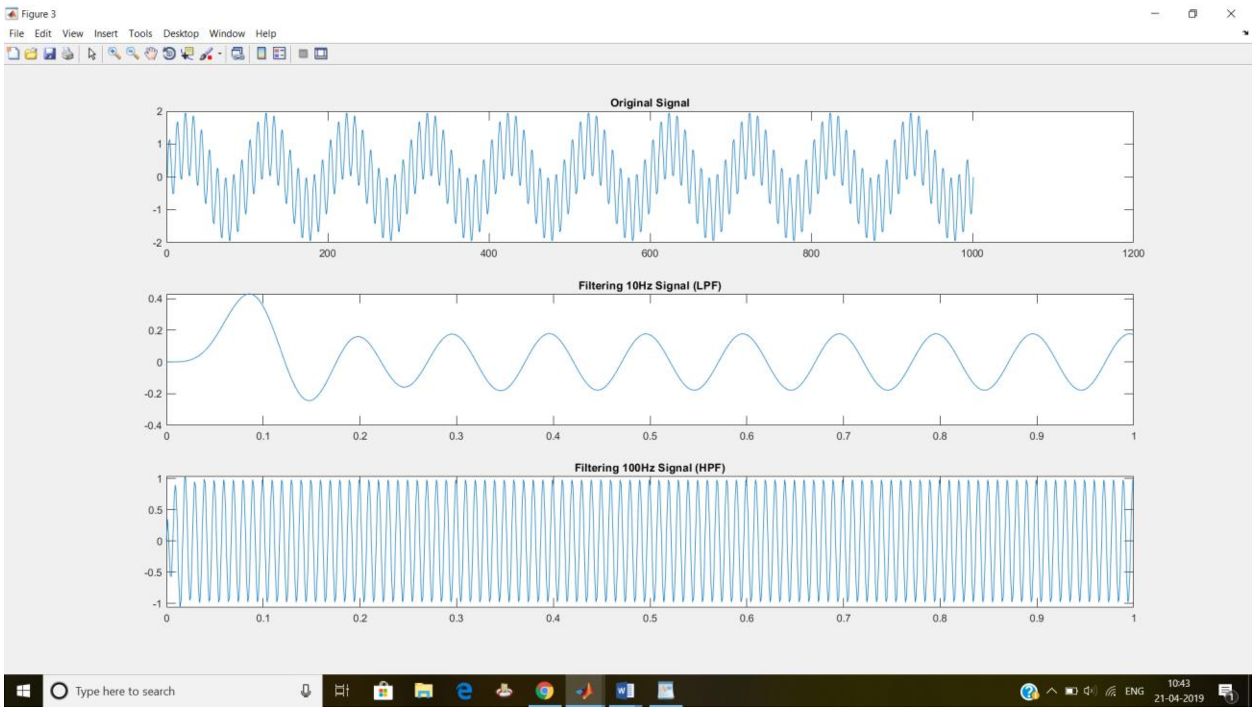


Figure 3
File Edit View Insert Tools Desktop Window Help
Original Signal
Filtering 10Hz Signal (LPF)
Filtering 100Hz Signal (HPF)
Type here to search
ENG
10:43
21-04-2019

Observations :

1) Order of Low Pass Filter Comes Out to 4 .
2) Order of High Pass Filter comes out to be 24 .
3) The output of LPF has high frequency components attenuated .
4) The output of HPF has low frequency components attenuated .

DSP KIT LAB TASKS

EXPERIMENT 1 :

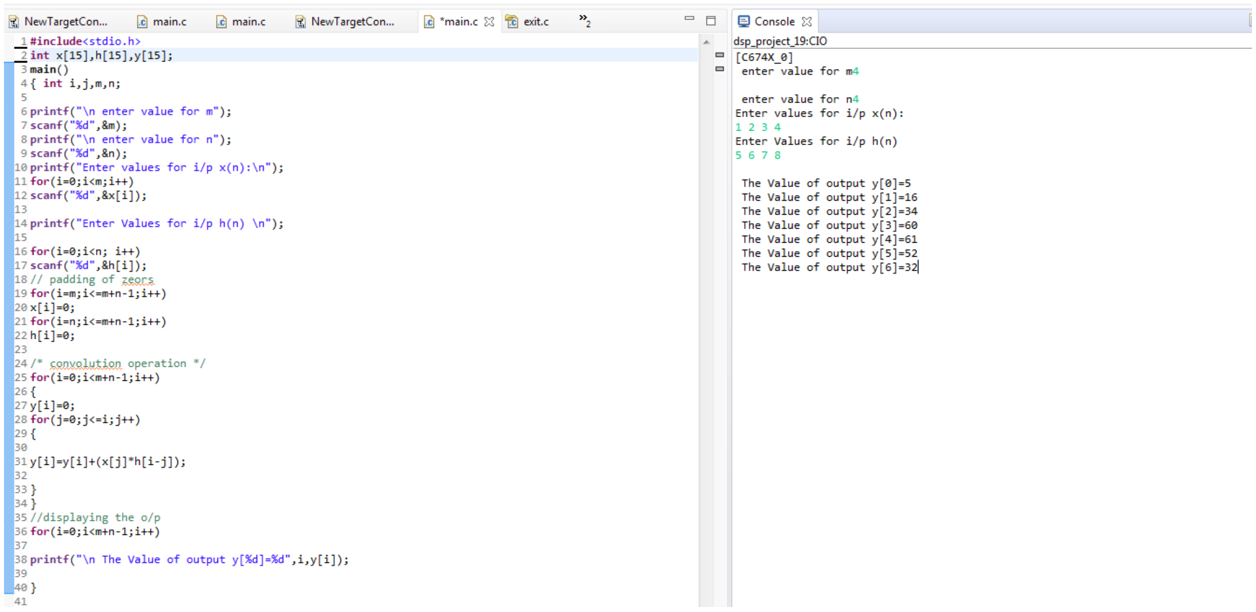
Aim : Write a program to perform Linear convolution.

Kits/software needed : CCS studio software .

Program:

```
#include<stdio.h>
int x[15],h[15],y[15];
main()
{ int i,j,m,n;
printf("\n enter value for m");
scanf("%d",&m);
printf("\n enter value for n");
scanf("%d",&n);
printf("Enter values for i/p x(n):\n");
for(i=0;i<m;i++)
scanf("%d",&x[i]);
printf("Enter Values for i/p h(n) \n");
for(i=0;i<n; i++)
scanf("%d",&h[i]);
// padding of zeors
for(i=m;i<=m+n-1;i++)
x[i]=0;
for(i=n;i<=m+n-1;i++)
h[i]=0;
/* convolution operation */
for(i=0;i<m+n-1;i++)
{
y[i]=0;
for(j=0;j<=i;j++)
{
y[i]=y[i]+(x[j]*h[i-j]);
}
}
//displaying the o/p
for(i=0;i<m+n-1;i++)
printf("\n The Value of output y[%d]=%d",i,y[i]);
}
```

Output :



```c
#include<stdio.h>
int x[15],h[15],y[15];
main()
{ int i,j,m,n;

printf("\n enter value for m");
scanf("%d",&m);
printf("\n enter value for n");
scanf("%d",&n);
printf("Enter values for i/p x(n):\n");
for(i=0;i<m;i++)
scanf("%d",&x[i]);

printf("Enter Values for i/p h(n) \n");

for(i=0;i<n; i++)
scanf("%d",&h[i]);
// padding of zeors
for(i=m;i<=m+n-1;i++)
x[i]=0;
for(i=n;i<=m+n-1;i++)
h[i]=0;

/* convolution operation */
for(i=0;i<m+n-1;i++)
{
y[i]=0;
for(j=0;j<=i;j++)
{

y[i]=y[i]+(x[j]*h[i-j]);

}
}
//displaying the o/p
for(i=0;i<m+n-1;i++)

printf("\n The Value of output y[%d]=%d",i,y[i]);

}
```

```
dsp_project_19:CIO
[C674X_0]
 enter value for m4

 enter value for n4
Enter values for i/p x(n):
1 2 3 4
Enter Values for i/p h(n)
5 6 7 8

 The Value of output y[0]=5
 The Value of output y[1]=16
 The Value of output y[2]=34
 The Value of output y[3]=60
 The Value of output y[4]=61
 The Value of output y[5]=52
 The Value of output y[6]=32
```

Result :

Linear convolution was Implemented using DSP Processor Kit .

<u>**EXPERIMENT 2**</u> **:**

<u>Aim :</u> Write a program to perform Circular Convolution.

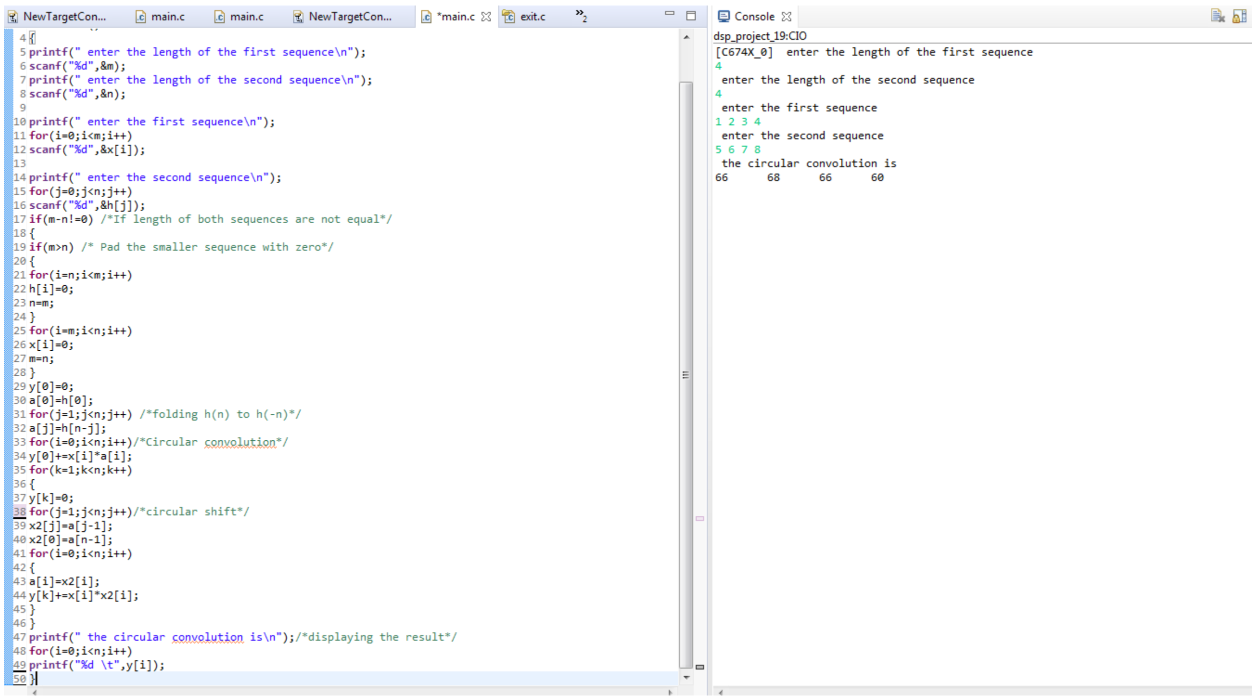
<u>Kits/software needed :</u> CCS studio software

<u>Program:</u>

```
#include<stdio.h>
int m,n,x[30],h[30],y[30],i,j, k,x2[30],a[30];
void main()
{
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) /*If length of both sequences are not
equal*/
{
if(m>n) /* Pad the smaller
sequence with zero*/
{
for(i=n;i<m;i++)
h[i]=0;
n=m;
}
for(i=m;i<n;i++)
x[i]=0;
m=n;
}
y[0]=0;
a[0]=h[0];
for(j=1;j<n;j++) /*folding h(n) to h(-n)*/
a[j]=h[n-j];
/*Circular convolution */
for(i=0;i<n;i++)
y[0]+=x[i]*a[i];
for(k=1;k<n;k++)
{
```

```
y[k]=0;
/*circular shift*/
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];
}
}
/*displaying the result*/
printf(" the circular convolution is\n");
for(i=0;i<n;i++)
printf("%d \t",y[i]);
}
```

Output :



```
{
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) /*If length of both sequences are not equal*/
{
if(m>n) /* Pad the smaller sequence with zero*/
{
for(i=n;i<m;i++)
h[i]=0;
n=m;
}
for(i=m;i<n;i++)
x[i]=0;
m=n;
}
y[0]=0;
a[0]=h[0];
for(j=1;j<n;j++) /*folding h(n) to h(-n)*/
a[j]=h[n-j];
for(i=0;i<n;i++)/*Circular convolution*/
y[0]+=x[i]*a[i];
for(k=1;k<n;k++)
{
y[k]=0;
for(j=1;j<n;j++)/*circular shift*/
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(" the circular convolution is\n");/*displaying the result*/
for(i=0;i<n;i++)
printf("%d \t",y[i]);
}
```

```
dsp_project_19:CIO
[C674X_0]  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
5 6 7 8
 the circular convolution is
66      68      66      60
```

Result :

Circular convolution was Implemented using DSP Processor Kit .

EXPERIMENT 3 :

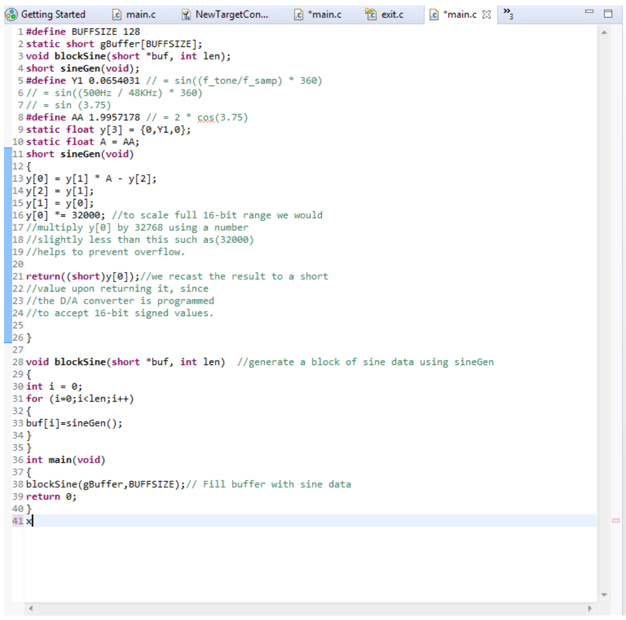
Aim : To Generate a sinewave form using TMS320C6748 DSP KIT.

Requirements:
· CCS v4
· TMS320C6748 KIT
· USB Cable
· 5V Adapter

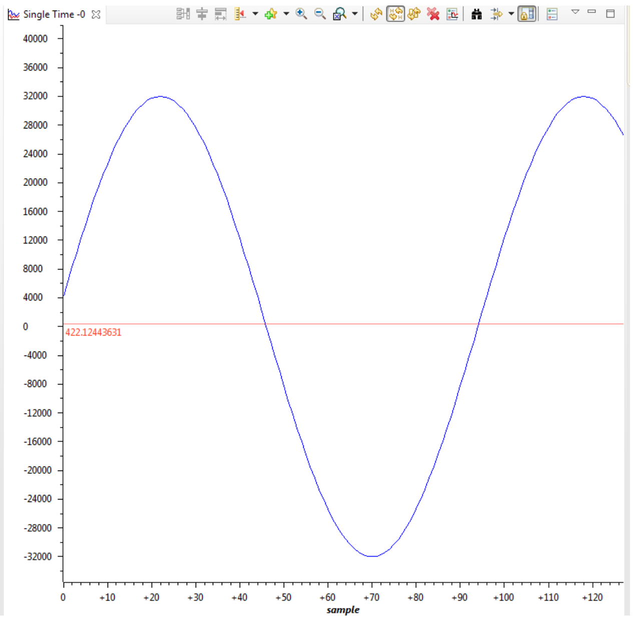
Code:

```
#define BUFFSIZE 128
static short gBuffer[BUFFSIZE];
void blockSine(short *buf, int len);
short sineGen(void);
#define Y1 0.0654031 // = sin((f_tone/f_samp) * 360)
// = sin((500Hz / 48KHz) * 360)
// = sin (3.75)
#define AA 1.9957178 // = 2 * cos(3.75)
static float y[3] = {0,Y1,0};
static float A = AA;
short sineGen(void)
{
y[0] = y[1] * A - y[2];
y[2] = y[1];
y[1] = y[0];
y[0] *= 32000; //to scale full 16-bit range we would
//multiply y[0] by 32768 using a number
//slightly less than this such as(32000)
//helps to prevent overflow.
return((short)y[0]);//we recast the result to a short
//value upon returning it, since
//the D/A converter is programmed
//to accept 16-bit signed values.
}
void blockSine(short *buf, int len)//generate a block of sine
data using sineGen
{
int i = 0;
for (i=0;i<len;i++)
{
buf[i]=sineGen();
}
}
int main(void)
{
blockSine(gBuffer,BUFFSIZE);// Fill buffer with sine data
return 0;
}
```

Output:



```
#define BUFFSIZE 128
static short gBuffer[BUFFSIZE];
void blockSine(short *buf, int len);
short sineGen(void);
#define Y1 0.0654031 // = sin((f_tone/f_samp) * 360)
// = sin((500Hz / 48KHz) * 360)
// = sin (3.75)
#define AA 1.9957178 // = 2 * cos(3.75)
static float y[3] = {0,Y1,0};
static float A = AA;
short sineGen(void)
{
y[0] = y[1] * A - y[2];
y[2] = y[1];
y[1] = y[0];
y[0] *= 32000; //to scale full 16-bit range we would
//multiply y[0] by 32768 using a number
//slightly less than this such as(32000)
//helps to prevent overflow.

return((short)y[0]);//we recast the result to a short
//value upon returning it, since
//the D/A converter is programmed
//to accept 16-bit signed values.

}

void blockSine(short *buf, int len)  //generate a block of sine data using sineGen
{
int i = 0;
for (i=0;i<len;i++)
{
buf[i]=sineGen();
}
}
int main(void)
{
blockSine(gBuffer,BUFFSIZE);// Fill buffer with sine data
return 0;
}
x
```



Result :

Sine wave was generated successfully using DSP Processor Kit .

EXPERIMENT 4 :

Aim : To test the sinusoidal waveform using DSP Processor trainer kit .

Requirements:
1. CCS v4
2. TMS320C6748 KIT
3. USB Cable
4. 5V Adapter

Program:

```
// L138_sine48_buf_intr.c
//
#include "L138_LCDK_aic3106_init.h"
#define LOOPLENGTH 48
#define BUFLENGTH 256
int16_t sine_table[LOOPLENGTH] =
{0, 1305, 2588, 3827, 5000, 6088, 7071, 7934,
8660, 9239, 9659, 9914, 10000, 9914, 9659, 9239,
8660, 7934, 7071, 6088, 5000, 3827, 2588, 1305,
0, -1305, -2588, -3827, -5000, -6088, -7071, -7934,
-8660, -9239, -9659, -9914, -10000, -9914, -9659, -9239,
-8660, -7934, -7071, -6088, -5000, -3827, -2588, -1305};
int16_t sine_ptr = 0; // pointer into lookup table
int32_t buffer[BUFLENGTH];
int16_t buf_ptr = 0;
interrupt void interrupt4(void) // interrupt service routine
{
int16_t sample;
sample = sine_table[sine_ptr]; // read sample from table
output_left_sample(sample); // output sample
sine_ptr = (sine_ptr+1)%LOOPLENGTH; // increment table
index
buffer[buf_ptr] = (int32_t)(sample); // store sample in buffer
buf_ptr = (buf_ptr+1)%BUFLENGTH; // increment buffer
index
return;
}
int main(void)
{
L138_initialise_intr(FS_48000_HZ,ADC_GAIN_0DB,DAC_A
TTEN_0DB,LCDK_LINE_INPUT);
while(1);
}
```

Output:

```
C674X_0: Output: Target Connected.
C674X_0: Output: -------------------------------------------
C674X_0: Output: Memory Map Cleared.
C674X_0: Output: -------------------------------------------
C674X_0: Output: Memory Map Setup Complete.
C674X_0: Output: -------------------------------------------
C674X_0: Output: PSC Enable Complete.
C674X_0: Output: -------------------------------------------
C674X_0: Output: PLL0 init done for Core:300MHz,
EMIFA:25MHz
C674X_0: Output: DDR initialization is in progress....
C674X_0: Output: PLL1 init done for DDR:150MHz
C674X_0: Output: Using DDR2 settings C674X_0:
Output: DDR2 init for 150 MHz is done
C674X_0: Output: -------------------------------------------
```

Connect headphone jack to line out of Dsp Processor kit you will hear sound of sine wave.