Experiment 1:

Aim: Generation of the given sequences and plot them using MATLAB.

Theory:

a) Unit Sample Sequence:

The unit sample sequence is a sequence of discrete samples having unit magnitude at origin and zero magnitude at other sample instants.

$$\delta(n) = \begin{cases} 1, & n=0 \\ 0, & n \neq 0 \end{cases}$$ $\delta(n)$: unit impulse

b) Unit Step Signal:

$$u(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n<0 \end{cases}$$

c) Ramp Sequence:

A ramp signal is a type of standard signal which starts at t=0 and increases linearly with time.

$$u_r(n) = \begin{cases} n, & n \geq 0 \\ 0, & n<0 \end{cases}$$

d) Exponential Sequence:

$$x(n) = \begin{cases} e^n, & n \geq 0 \\ 0, & n<0 \end{cases}$$

e) Sine Sequence:

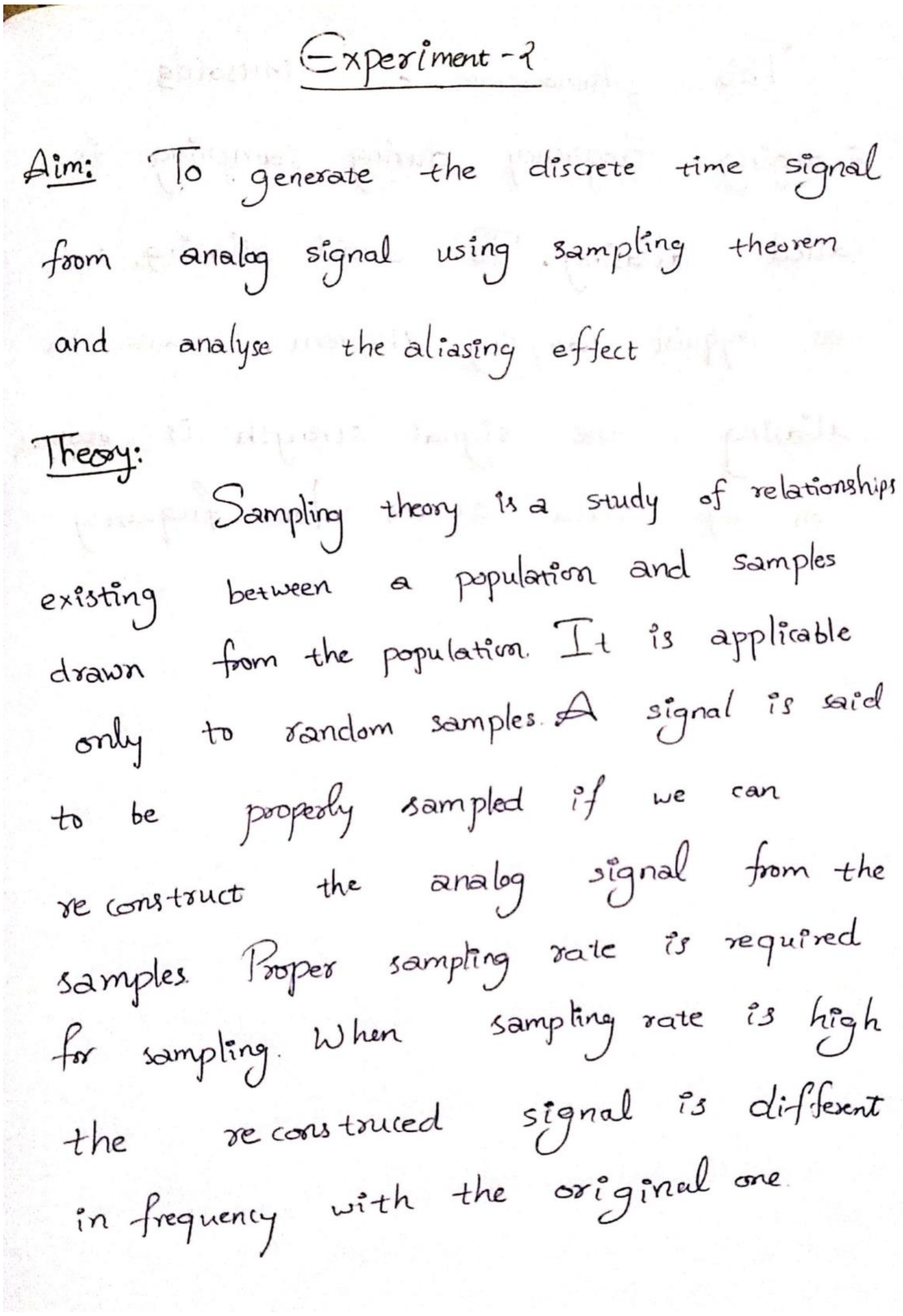
$$x(n) = \sin(n) \quad \forall n \in R$$

f) Cosine Sequence

$$x(n) = \cos(n) \quad \forall n \in R$$

Conclusion: All the given signals are generated and plotted.

Experiment 2

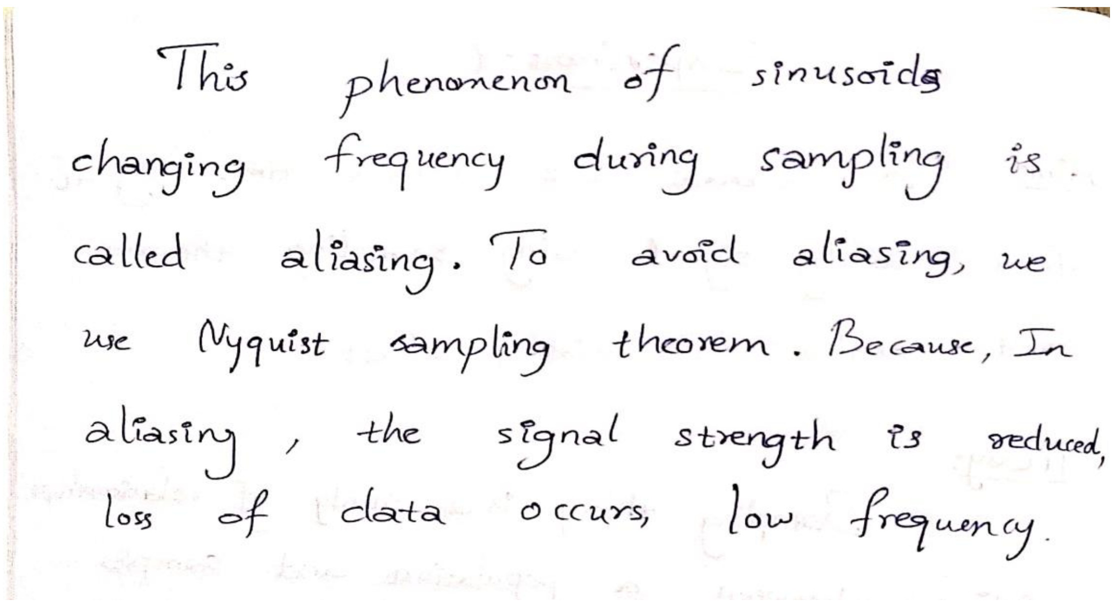


Experiment - 2

Aim: To generate the discrete time signal from analog signal using sampling theorem and analyse the aliasing effect

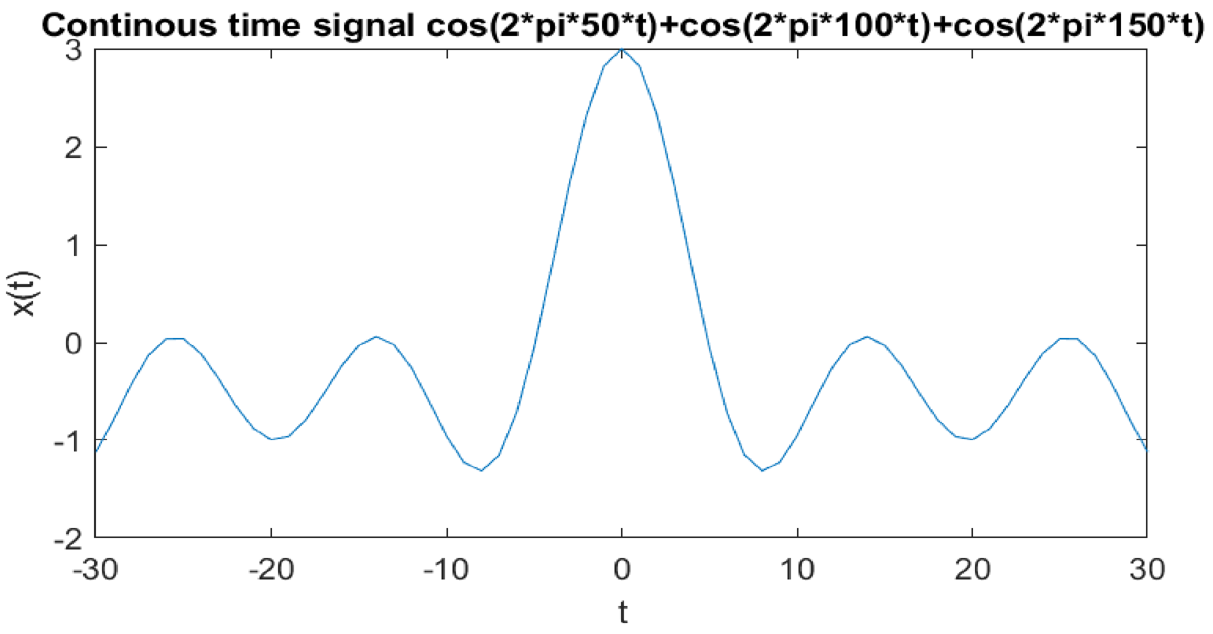
Theory:

Sampling theory is a study of relationships existing between a population and samples drawn from the population. It is applicable only to random samples. A signal is said to be properly sampled if we can reconstruct the analog signal from the samples. Proper sampling rate is required for sampling. When sampling rate is high the reconstruced signal is different in frequency with the original one.


This phenomenon of sinusoids changing frequency during sampling is called aliasing. To avoid aliasing, we use Nyquist sampling theorem. Because, In aliasing, the signal strength is reduced, loss of data occurs, low frequency.

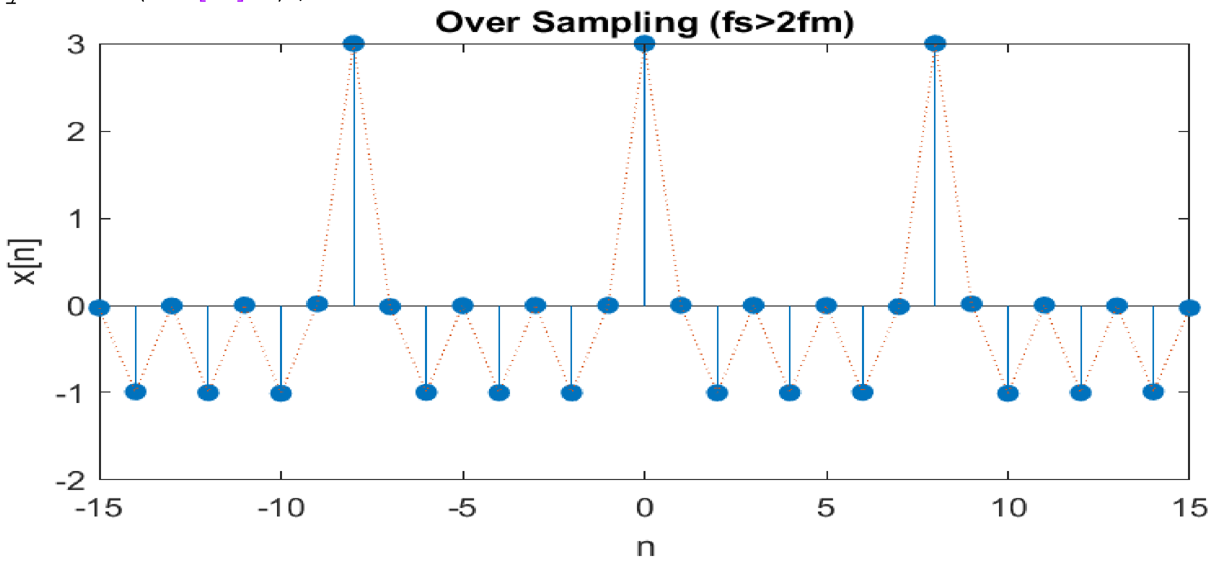
Code and Result:

```
%Sampling Theorem and Aliasing Effect:

t=-40:01:40;
y=cos(2*pi*50*t)+cos(2*pi*100*t)+cos(2*pi*150*t);
subplot(2,2,1)
plot(t,y)
title("Continous time signal
cos(2*pi*50*t)+cos(2*pi*100*t)+cos(2*pi*150*t)")
xlabel('t');
ylabel('x(t)');
```



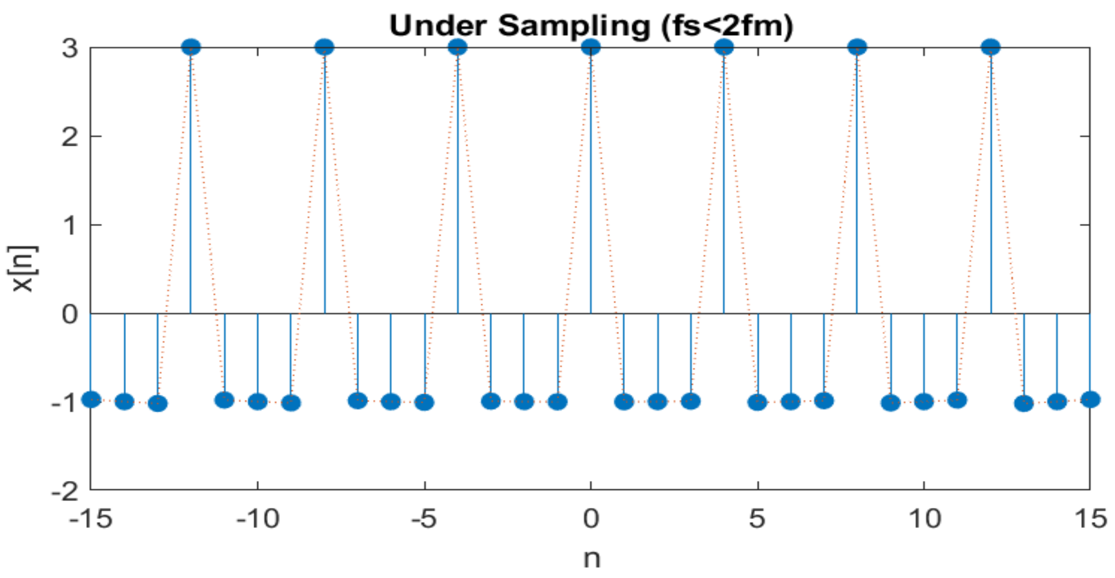
Here we observe the analog signal and determine max frequency as 150Hz.

```
%Over sampling fs>2fm
fs1=400;
n=-15:15;
x1=cos(2*pi*50*n/fs1)+cos(2*pi*100*n/fs1)+cos(2*pi*15
0*n/fs1);
subplot(2,2,2)
stem(n,x1,'filled')
hold on
subplot(2,2,2);
plot(n,x1,':');
title('Over Sampling (fs>2fm)');
xlabel('n');
ylabel('x[n]');
```



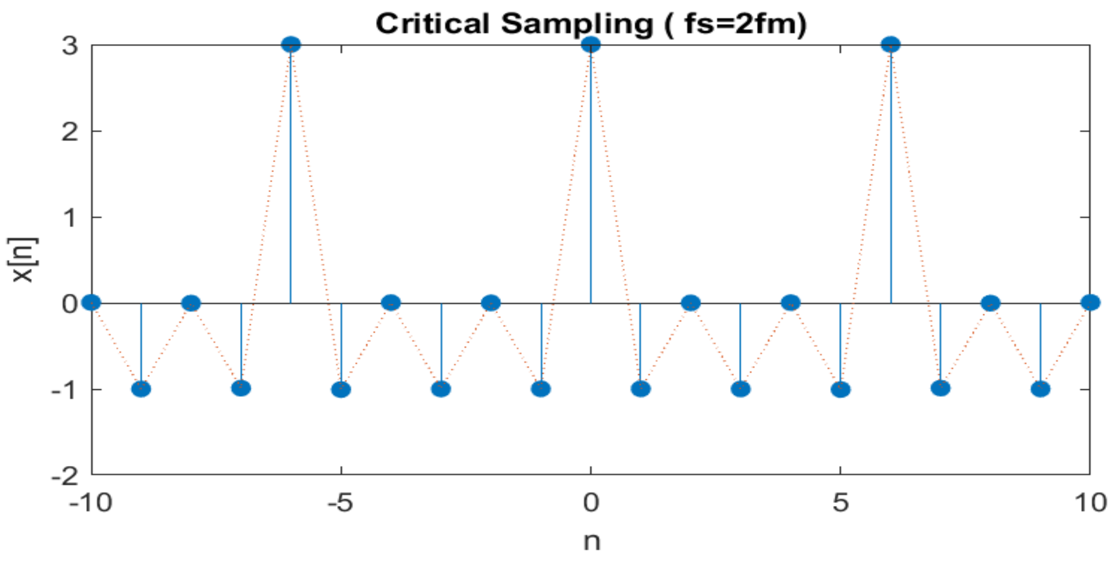
The case of oversampling i.e. fs>2fm. Here we selected fs as 400Hz.

```
%under sampling (fs<2fm):
fs2=200;
n=-15:15;
x1=cos(2*pi*50*n/fs2)+cos(2*pi*100*n/fs2)+cos(2*pi*15
0*n/fs2);
subplot(2,2,3)
stem(n,x1,'filled')
hold on
subplot(2,2,3);
plot(n,x1,':');
title('Under Sampling (fs<2fm)');
xlabel('n');
ylabel('x[n]');
```



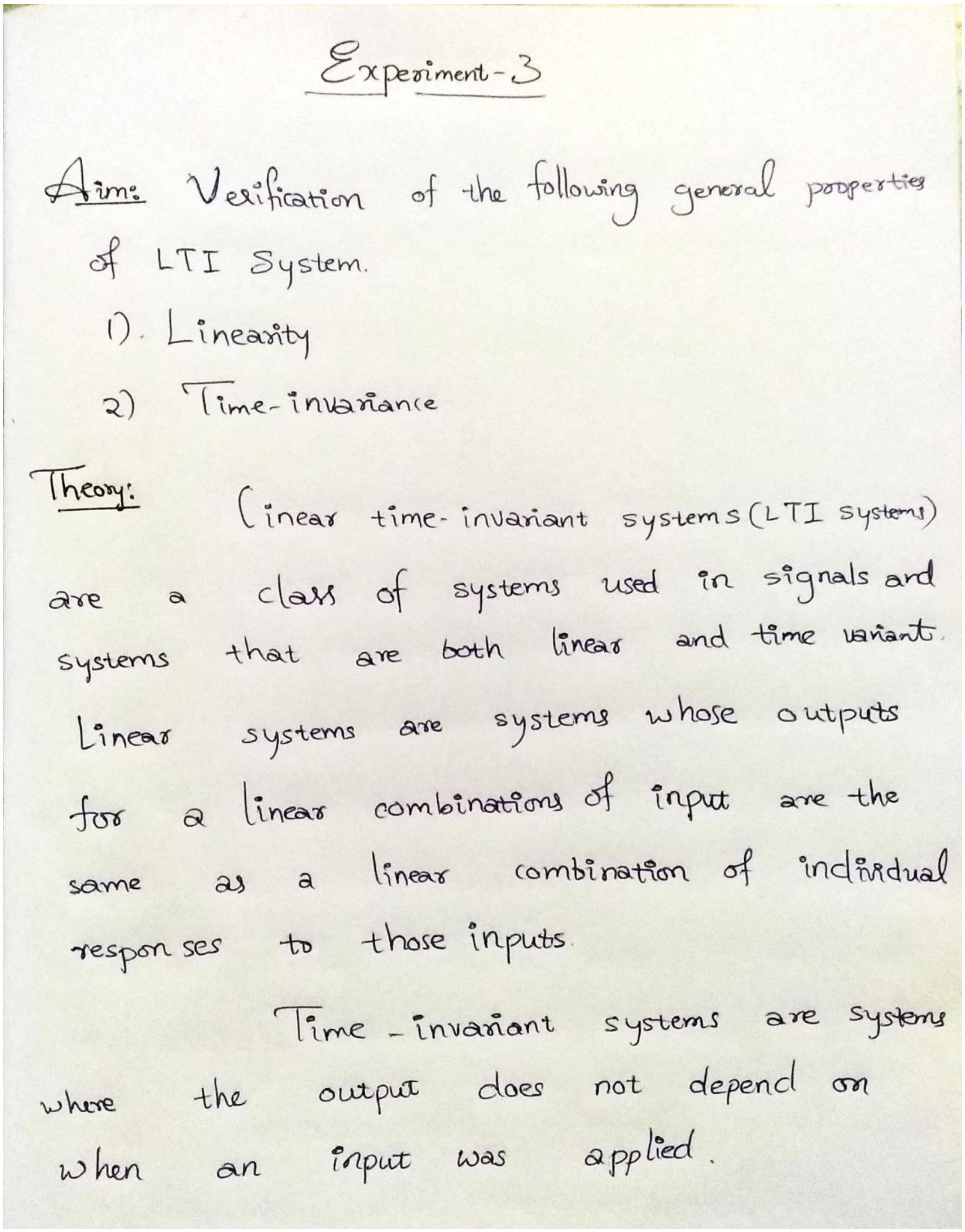
Here we observe that some data is lost due to aliasing effect.

```
%Critical sampling fs=2fm
fs3=300;
n=-10:10;
x1=cos(2*pi*50*n/fs3)+cos(2*pi*100*n/fs3)+cos(2*pi*150*n/
fs3);
subplot(2,2,4)
stem(n,x1,'filled')
hold on
subplot(2,2,4);
plot(n,x1,':');
title('Critical Sampling ( fs=2fm)');
xlabel('n');
ylabel('x[n]');
```



Conclusion: We have verified the Nyquist Sampling Theorem which gives us critical sampling rate below which aliasing will occurs.

EXPERIMENT -III



Experiment-3

Aim: Verification of the following general properties of LTI System.

1). Linearity

2) Time-invariance

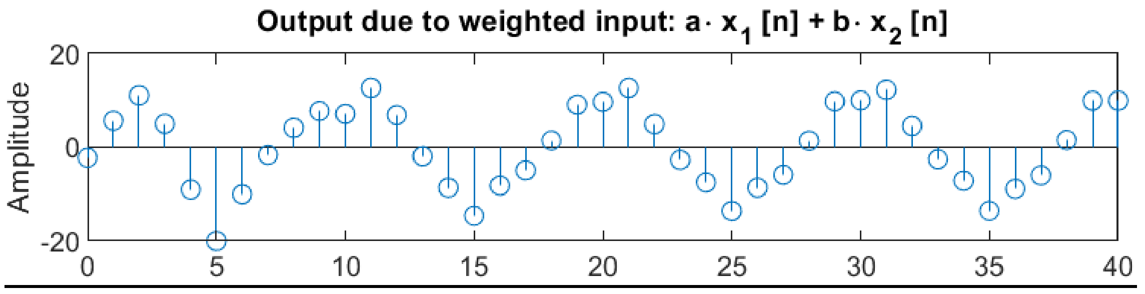
Theory: Linear time-invariant systems (LTI systems) are a class of systems used in signals and systems that are both linear and time variant. Linear systems are systems whose outputs for a linear combinations of input are the same as a linear combination of individual responses to those inputs.

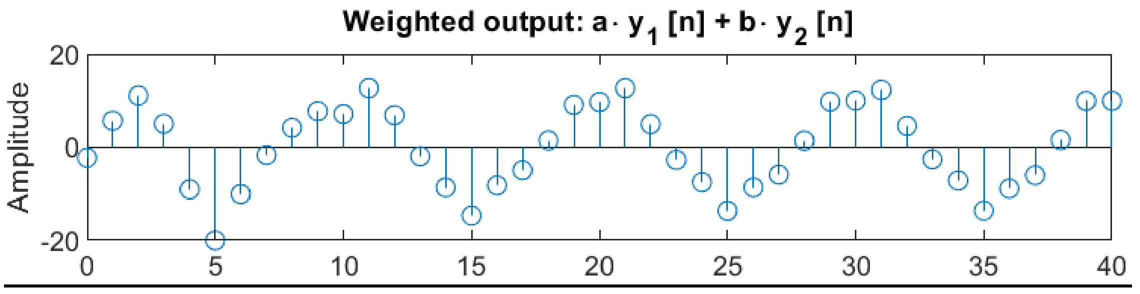
Time - invariant systems are systems where the output does not depend on when an input was applied.

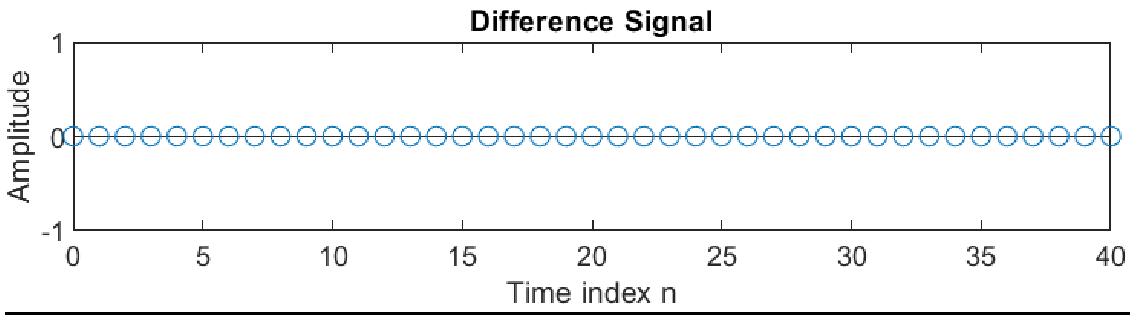
Code and Result:

Linearity:

```
n=0:40;
a=2;b=-3;
x1=cos(2*pi*0.1*n;
x2=cos(2*pi*0.4*n;
x= a*x1 + b*x2;
num=[2.2403 2.4908 2.2403];
den=[1 -0.4 0.75];
ic=[0 0];
y1=filter(num,den,x1,ic);
y2=filter(num,den,x2,ic);
y=filter(num,den,x,ic);
yt= a*y1 + b*y2;
d= round(y - yt);
subplot(3,1,1)
stem(n,y);
ylabel('Amplitude');
title('Output due to weighted input: a\cdot x_1 [n]
+ b\cdot x_2 [n]');
subplot(3,1,2)
stem(n,yt);
ylabel('Amplitude');
title('Weighted output: a\cdot y_1 [n] + b\cdot y_2
[n]');
subplot(3,1,3)
stem(n,d);
xlabel('Time index n');
ylabel('Amplitude');
title('Difference Signal');
```



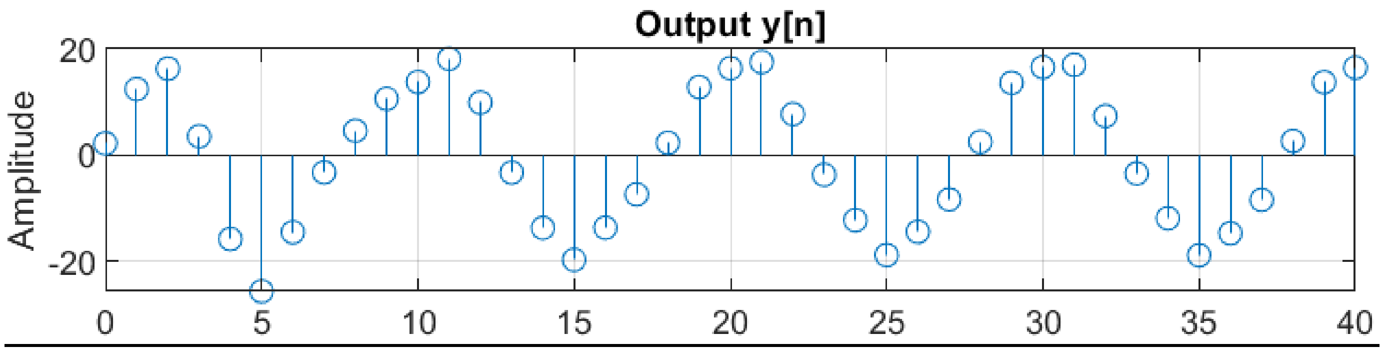


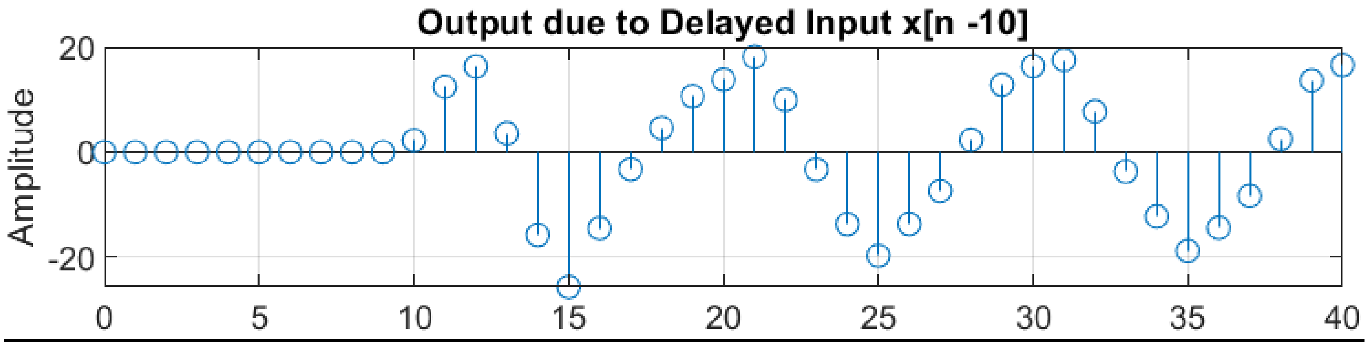


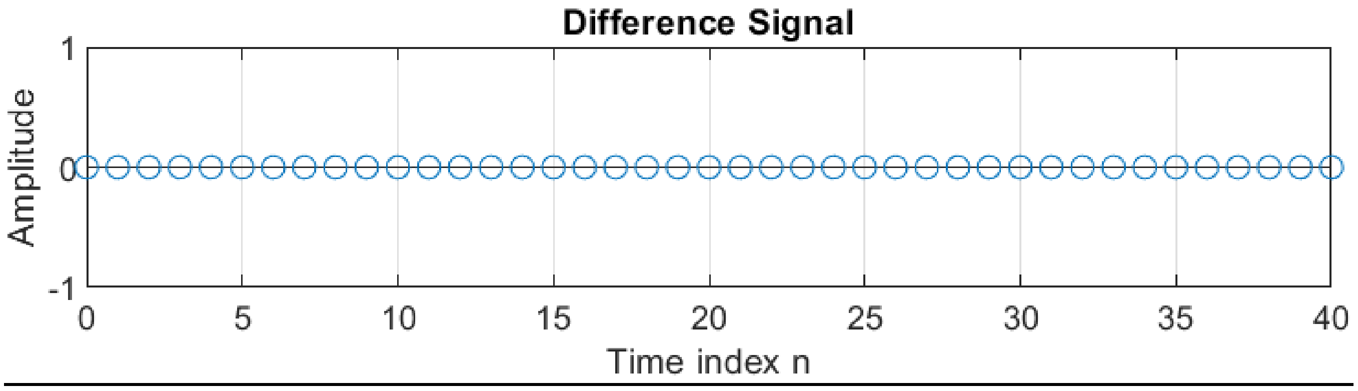
Time-variance:

```
n=0:40; D=10; a=3.0;b=-2;
x = a*cos(2*pi*0.1*n) + b*cos(2*pi*0.4*n);
xd= [zeros(1,D) x];
num=[2.2403 2.4903 2.2403];
den=[1 -0.4 0.75];
ic = [0 0];
y=filter(num,den,x,ic);
yd=filter(num,den,xd,ic);
```

```
d= y - yd(1+D:41+D);

subplot(3,1,1)
stem(n,y);
ylabel('Amplitude')
title('Output y[n]');
grid;

subplot(3,1,2)
stem(n,yd(1:41));
ylabel('Amplitude');

title(['Output due to Delayed Input x[n -
',num2str(D),']']);
grid;

subplot(3,1,3)
stem(n,d);
xlabel('Time index n');
ylabel('Amplitude');
title('Difference Signal');
grid;
```

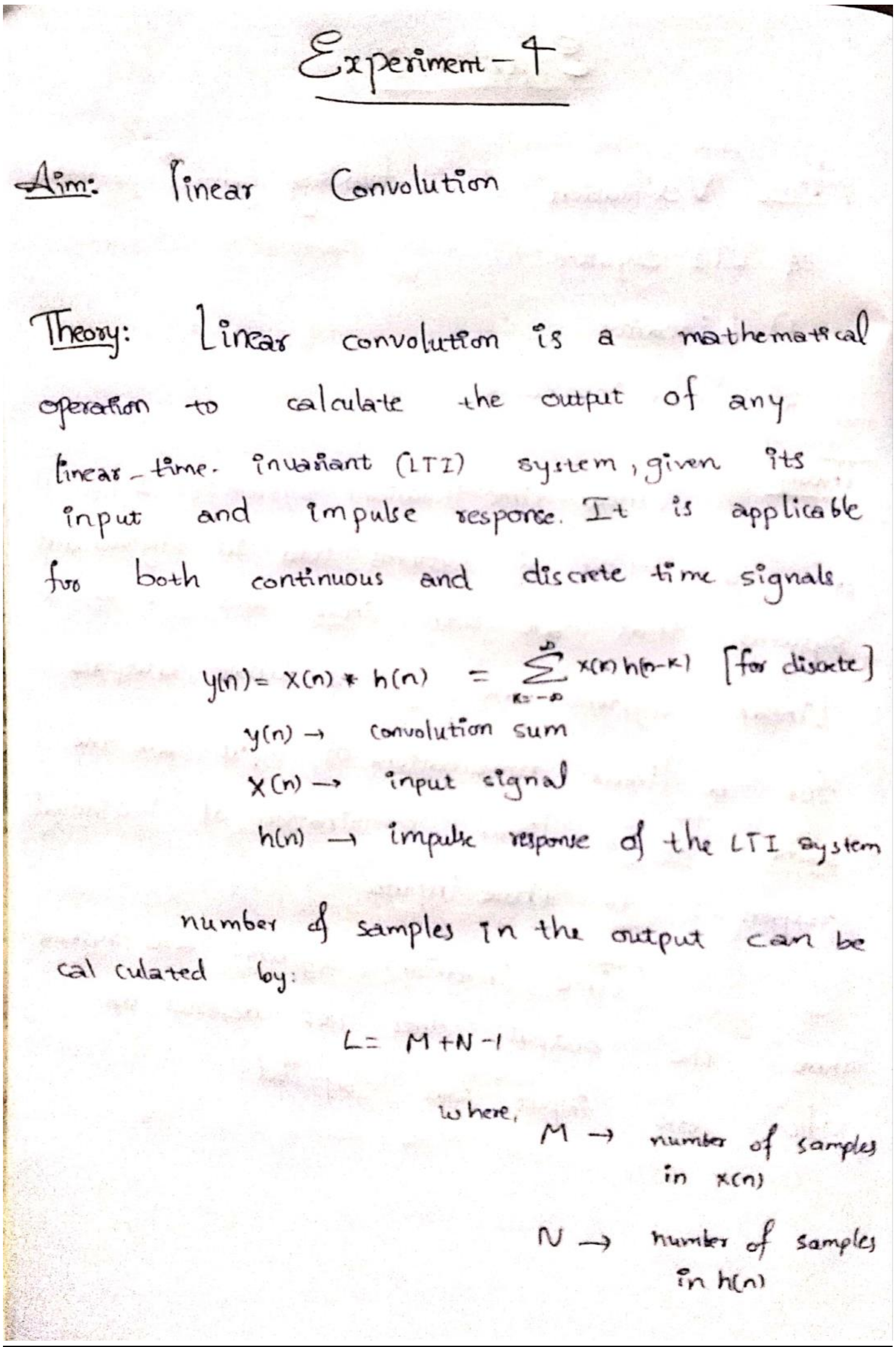






<u>Conclusion:</u> We have verified the linearity and time variance property of the LTI system.

EXPERIMENT -IV



Experiment - 4

Aim: linear Convolution

Theory: Linear convolution is a mathematical operation to calculate the output of any linear - time - invariant (LTI) system, given its input and impulse response. It is applicable for both continuous and discrete time signals.

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k) \quad \text{[for discrete]}$$

$y(n) \rightarrow$ convolution sum

$x(n) \rightarrow$ input signal

$h(n) \rightarrow$ impulse response of the LTI system

number of samples in the output can be calculated by:

$$L = M + N - 1$$

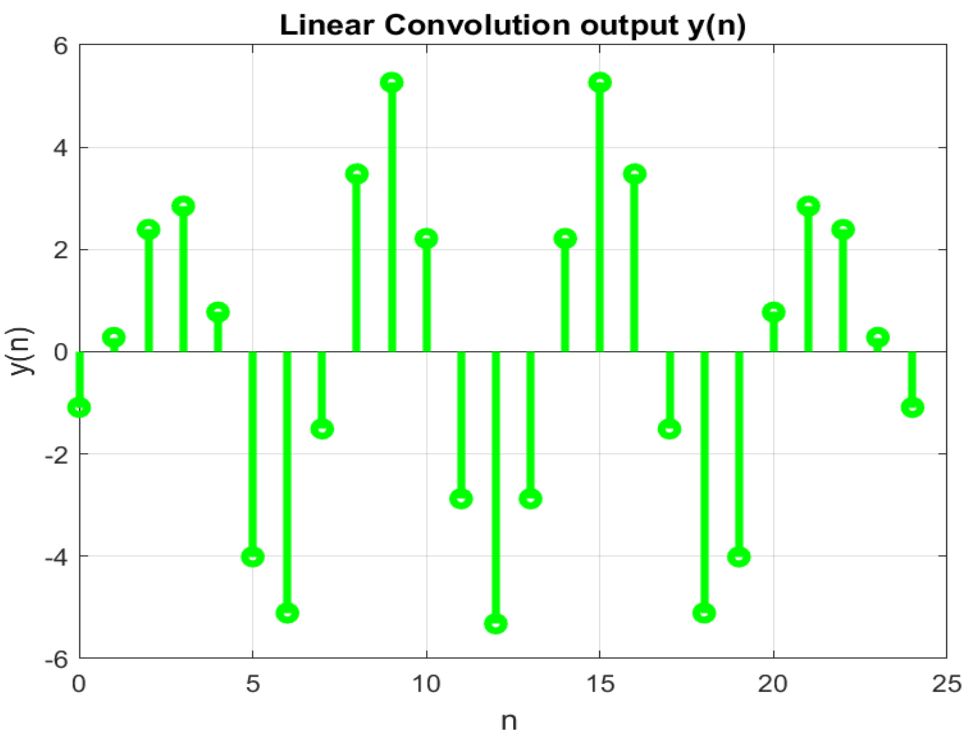
where,

$M \rightarrow$ number of samples in $x(n)$

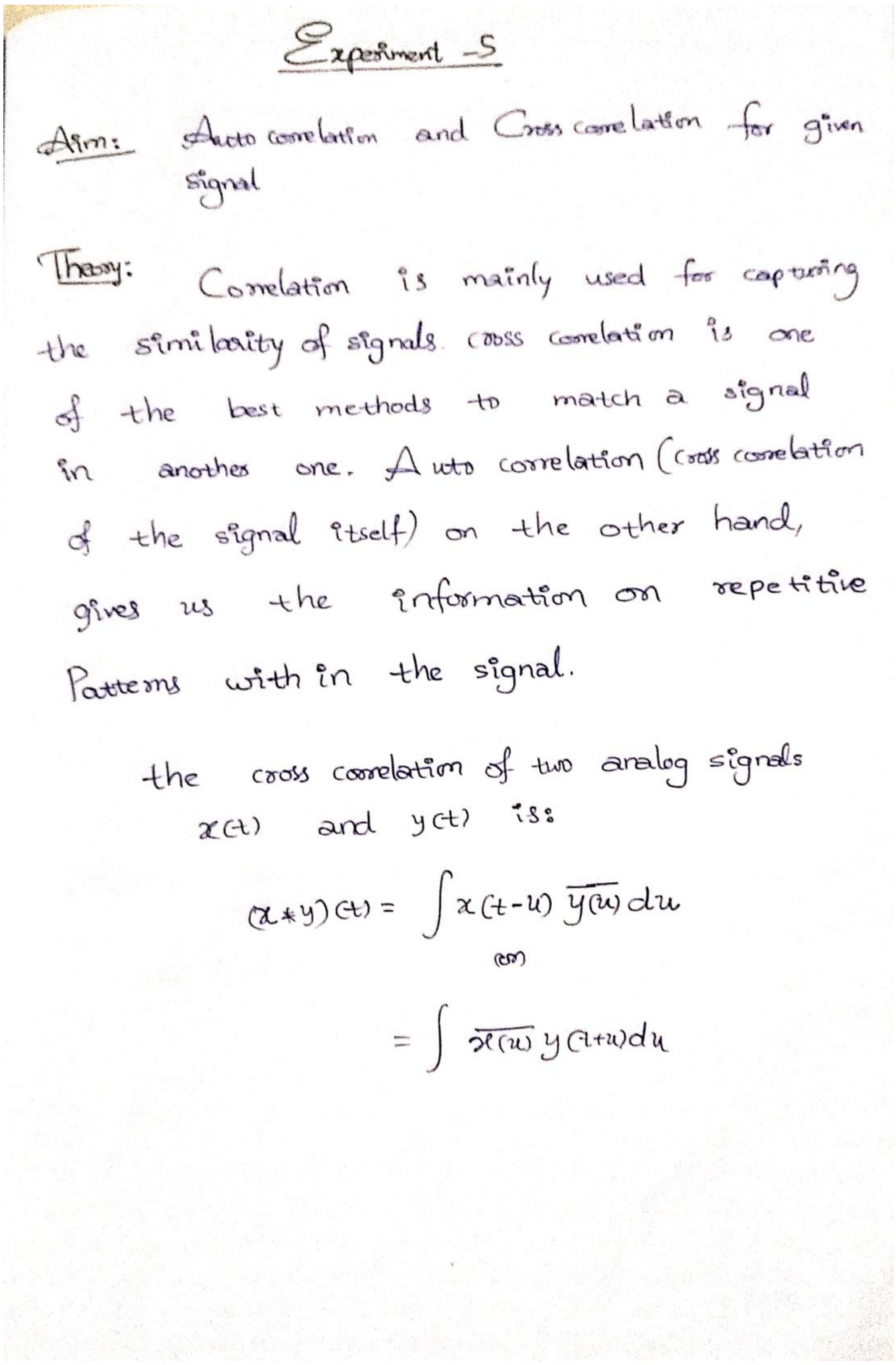
$N \rightarrow$ number of samples in $h(n)$

Code and Result:

```
n=-10:10;
x=sin(n);
h=[-2,-1,0,1,2];
N = length(x);
M = length(h);
Ny = N + M -1;
m = 0: Ny-1;

y = conv(x,h);
stem(m,y,'linewidth',3,'color','g')
grid;
a = title('Linear Convolution output y(n)');
set(a,'fontsize',12);
a = ylabel('y(n)');
set(a,'Fontsize',12);
a = xlabel('n ');
set(a,'Fontsize',12);
```



EXPERIMENT -V



Experiment -5

Aim: Auto correlation and Cross correlation for given signal

Theory: Correlation is mainly used for capturing the similarity of signals. cross correlation is one of the best methods to match a signal in another one. Auto correlation (cross correlation of the signal itself) on the other hand, gives us the information on repetitive Patterns with in the signal.

the cross correlation of two analog signals $x(t)$ and $y(t)$ is:

$$(x*y)(t) = \int x(t-u)\,\overline{y(u)}\,du$$

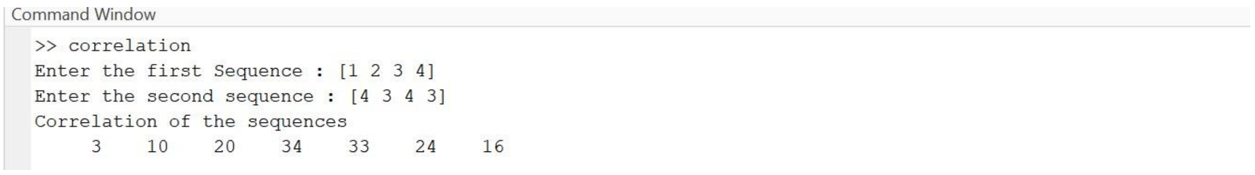
(or)

$$= \int \overline{x(u)}\,y(t+u)\,du$$

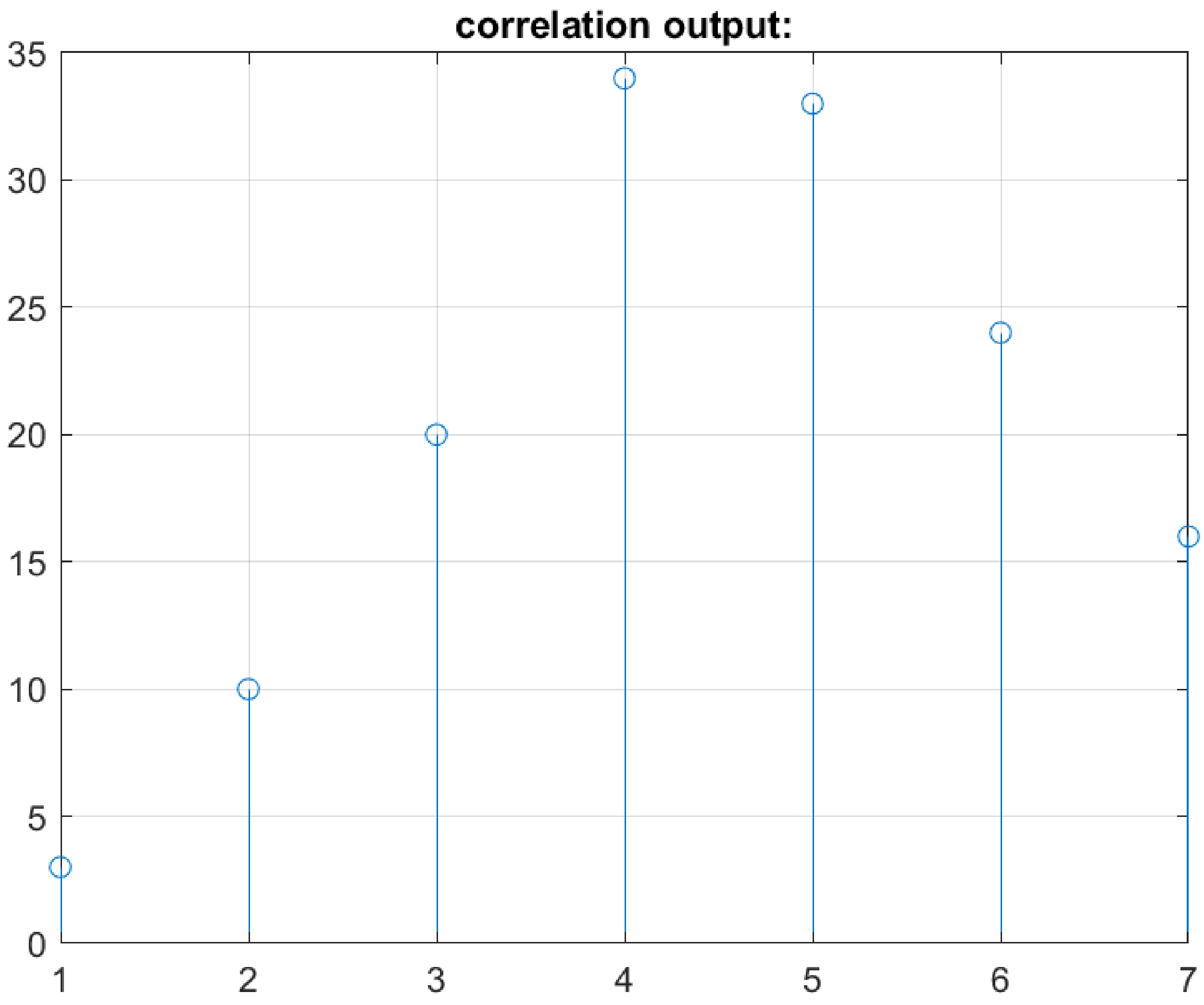
Code :

```
x=input('Enter the first Sequence : ');
h=input('Enter the second sequence : ');
n=length(x);
m=length(h);
k=n+m-1;
x=[x zeros(1,k-n)]';
h=wrev(h);
h=[h zeros(1,k-m)]';
c=zeros(k);
for i=1:k
    c(:,i)=circshift(x,i-1);
end
y=c*h;
disp('Correlation of the sequences')
disp(y');
title('correlation output:');
grid on;
```

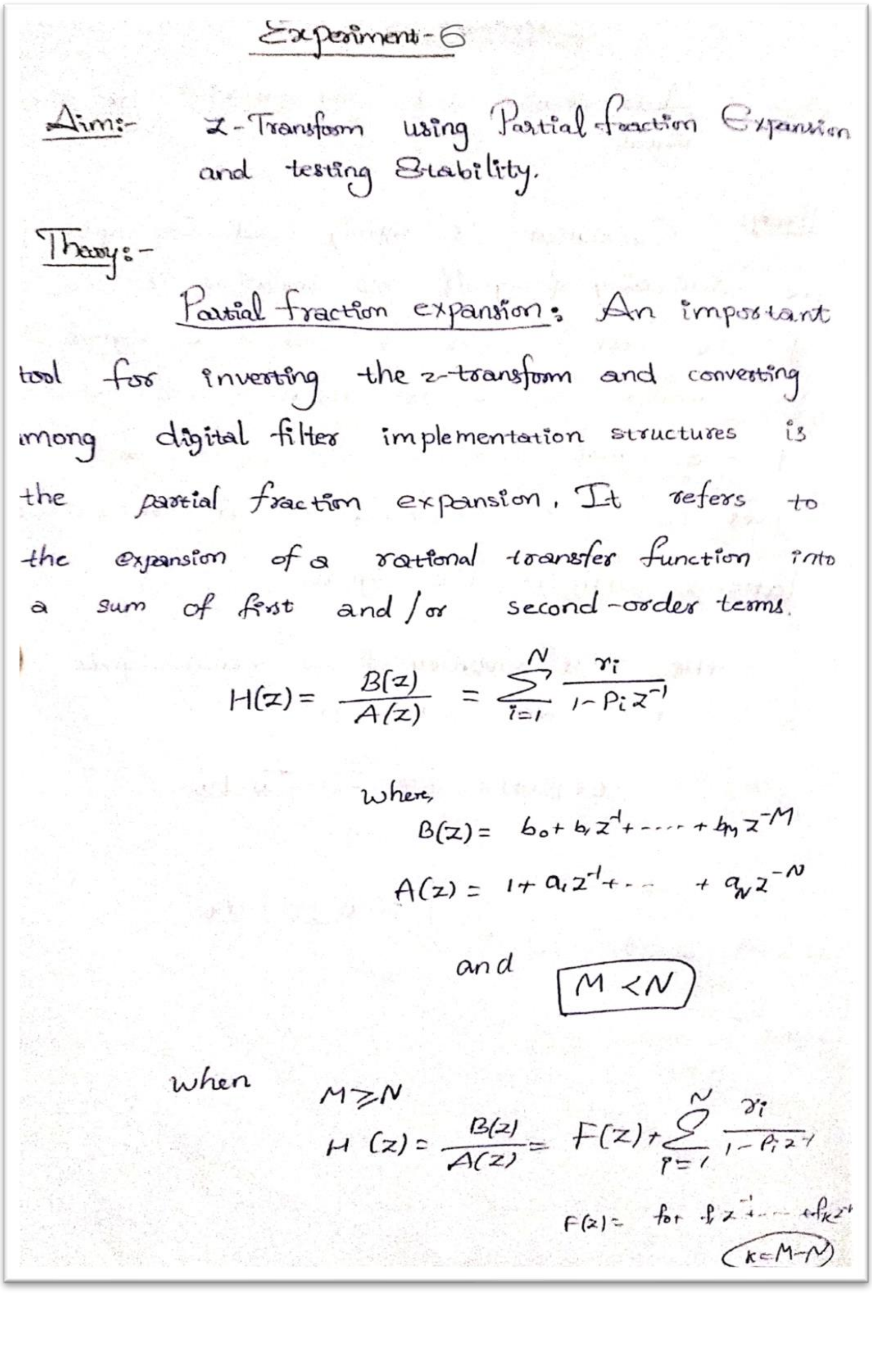
Result:



```
>> correlation
Enter the first Sequence : [1 2 3 4]
Enter the second sequence : [4 3 4 3]
Correlation of the sequences
     3    10    20    34    33    24    16
```


correlation output:
35
30
25
20
15
10
5
0
1
2
3
4
5
6
7

EXPERIMENT -VI



Experiment-6

Aim:- z-Transform using Partial fraction Expansion and testing Stability.

Theory:-

Partial fraction expansion: An important tool for inverting the z-transform and converting among digital filter implementation structures is the partial fraction expansion. It refers to the expansion of a rational transfer function into a sum of first and/or second-order terms.

$$H(z) = \frac{B(z)}{A(z)} = \sum_{i=1}^{N} \frac{r_i}{1-p_i z^{-1}}$$

where,

$$B(z) = b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}$$

$$A(z) = 1 + a_1 z^{-1} + \cdots + a_N z^{-N}$$

and $\boxed{M < N}$

when $M \geq N$

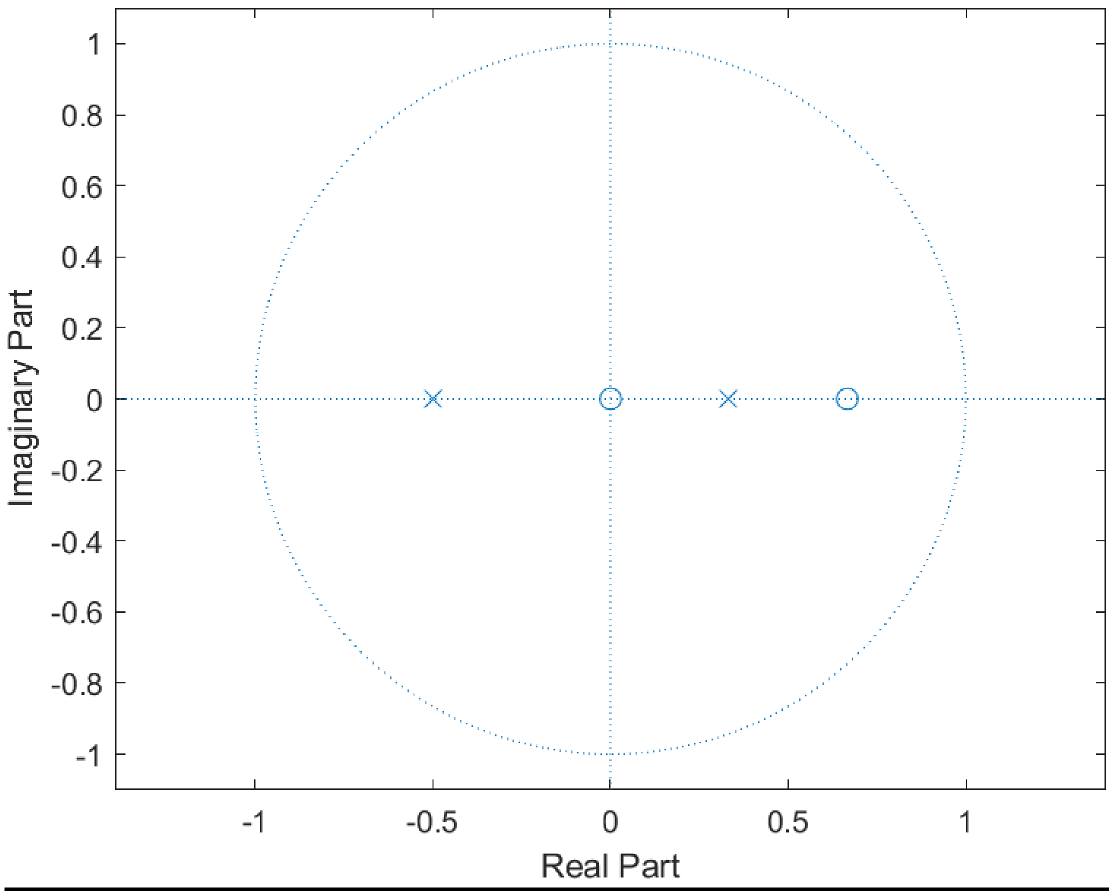
$$H(z) = \frac{B(z)}{A(z)} = F(z) + \sum_{i=1}^{N} \frac{r_i}{1-p_i z^{-1}}$$

$$F(z) = f_0 + f_1 z^{-1} + \cdots + f_K z^{-K}$$

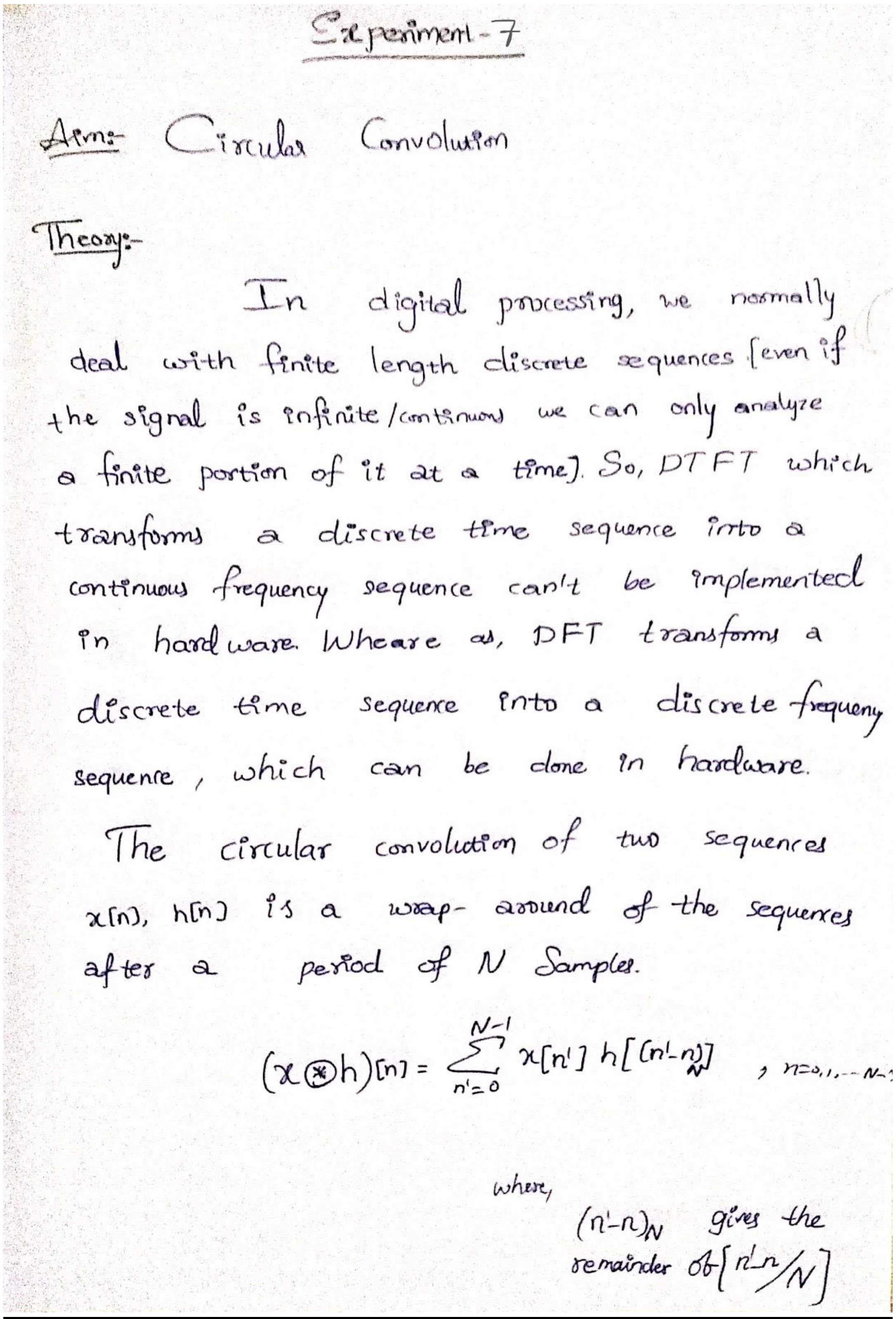
$(K = M - N)$

Code and Result:

```
clear all
close all

m = [1,-2/3];
n = [1,1/6,-1/6];
[r, p, k] = residuez(m,n)

zplane(m,n)
%hold on
%plot(p,'^r')
%hold off

syms z
X = ((-2/5)*(z/(z-(1/3))))+((7/5)*(z/(z+(1/2))));
iztrans(X)
```



EXPERIMENT -VII



Experiment-7

Aim:- Circular Convolution

Theory:-

In digital processing, we normally deal with finite length discrete sequences [even if the signal is infinite/continuous we can only analyze a finite portion of it at a time]. So, DTFT which transforms a discrete time sequence into a continuous frequency sequence can't be implemented in hardware. Wheare as, DFT transforms a discrete time sequence into a discrete frequency sequence, which can be done in hardware.

The circular convolution of two sequences x[n], h[n] is a wrap- around of the sequences after a period of N Samples.

$$(x \circledast h)[n] = \sum_{n'=0}^{N-1} x[n']\, h[(n'-n)_N] \quad , n=0,1,\ldots N-$$

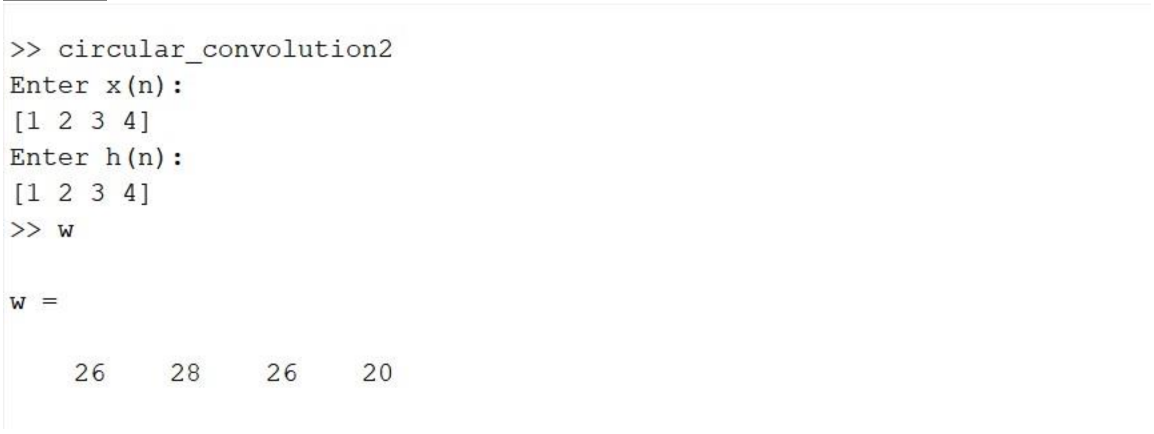
where,

$(n'-n)_N$ gives the remainder of $\left[n'-n / N\right]$

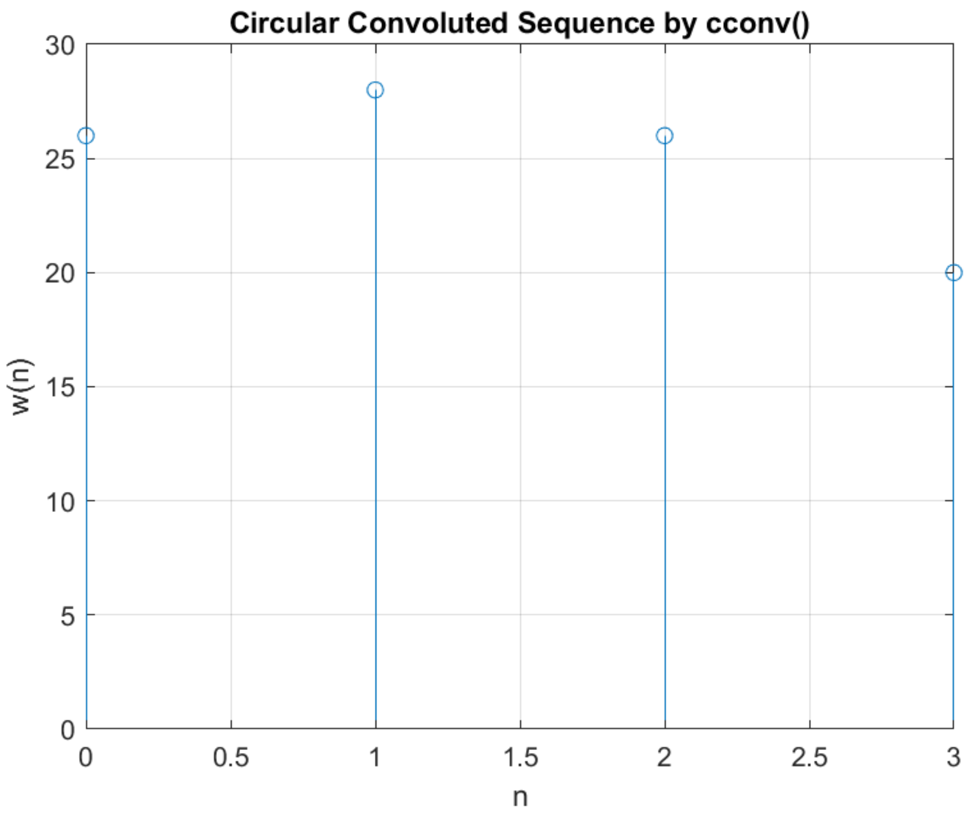
Code and Result:

```
x=input('Enter x(n):\n');
h=input('Enter h(n):\n');
m=length(x);
n=length(h);
N=max(m,n);
x=[x,zeros(1,N-m)];
h=[h,zeros(1,N-n)];
n=0:N-1;

w = cconv(x,h,N);

stem(n,w)
xlabel('n')
ylabel('w(n)')
title('Circular Convoluted Sequence by cconv()')
grid on;
```

Input:



```
>> circular_convolution2
Enter x(n):
[1 2 3 4]
Enter h(n):
[1 2 3 4]
>> w

w =

    26    28    26    20
```


Circular Convoluted Sequence by cconv()
w(n)
n
0
5
10
15
20
25
30
0
0.5
1
1.5
2
2.5
3

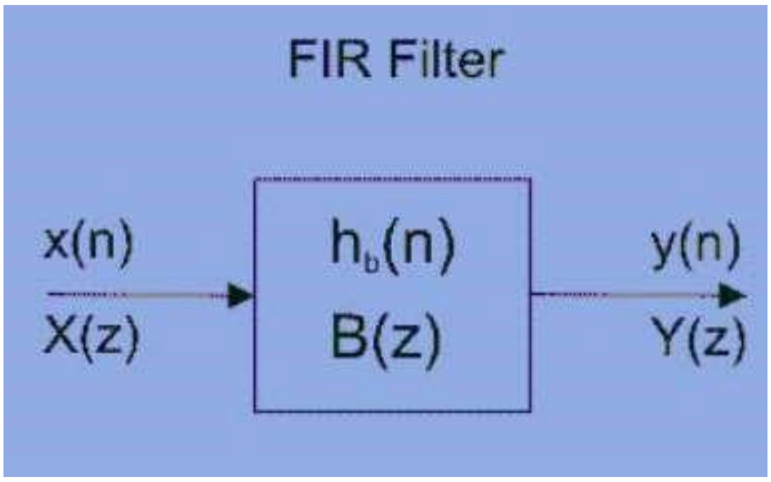
EXPERIMENT -VIII

Aim: Cascade realization of the Linear-Phase FIR/ IIR transfer functions using MATLAB

Software used: MATLAB Online

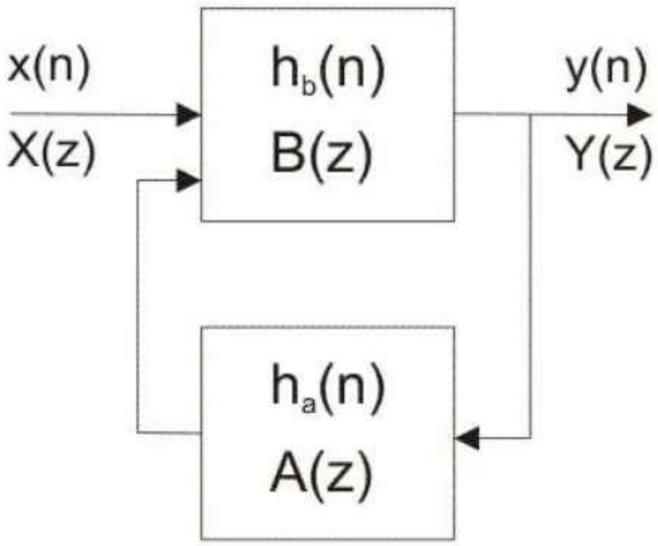
Theory:

The term FIR abbreviation is “Finite Impulse Response” and it is one of two main types of digital filters used in DSP applications. FIR is a filter whose impulse response is of finite period, as a result of it settles to zero in finite time. This is often in distinction to IIR filters, which can have internal feedback and will still respond indefinitely. The impulse response of an Nth order discrete time FIR filter takes precisely N+1 sample before it then settles to zero.



The infinite impulse response is a type of digital filter that is used in Digital Signal Processing applications. A filter’s job is to allow certain types of signals to pass and block the rest. The infinite impulse response filter is unique because it uses a feedback mechanism. It requires current as well as past output data. Though they are harder to design, IIR filters are computationally efficient and generally cheaper.

IIR Filter



IIR filter block diagram

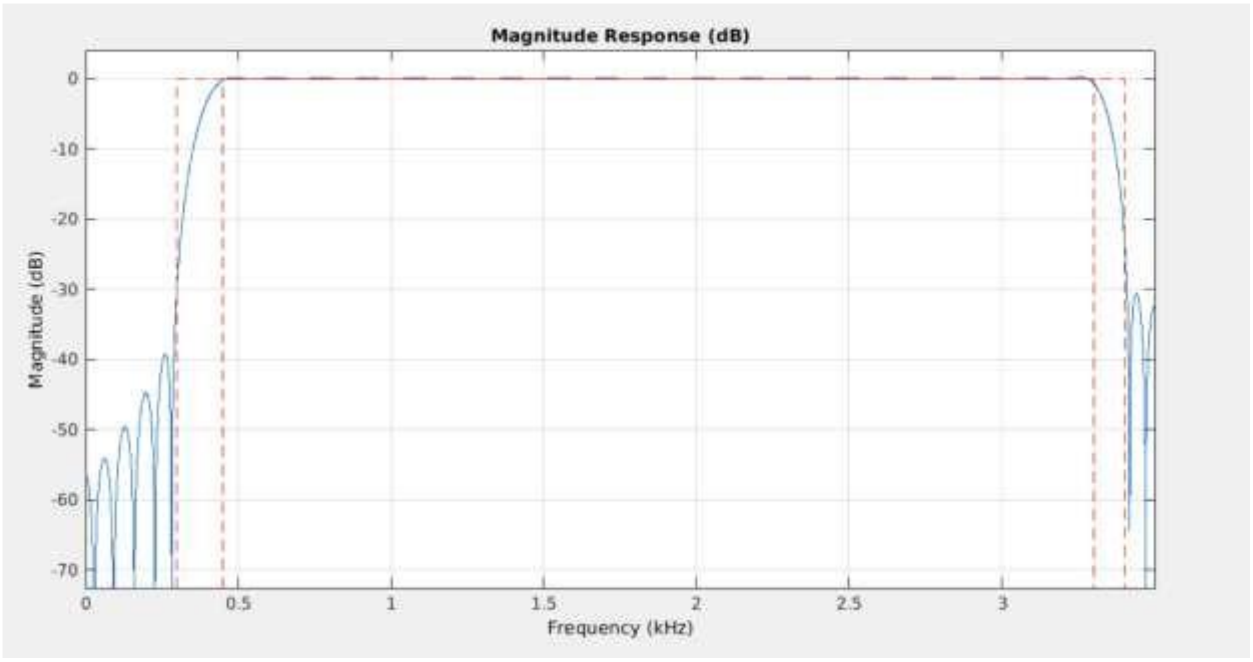
CODE:

```
           %bandpass FIR filter design
bpfir = designfilt('bandpassfir', ... % Response type
'FilterOrder',86, ... % Filter order
'StopbandFrequency1',300, ... % Frequency constraints
'PassbandFrequency1',450, ...
'PassbandFrequency2',3300, ...
'StopbandFrequency2',3400, ...
'DesignMethod','ls', ... % Design method
'StopbandWeight1',1, ... % Design method options
'PassbandWeight', 2, ...
'StopbandWeight2',3, ...
'SampleRate',7000)
fvtool(bpfir)
```

```matlab
%bandpass IIR filter design
bpiir = designfilt('bandpassiir', ... % Response type
'StopbandFrequency1',300, ... % Frequency constraints
'PassbandFrequency1',450, ...
'PassbandFrequency2',3300, ...
'StopbandFrequency2',3400, ...
'StopbandAttenuation1',40, ... % Magnitude constraints
'PassbandRipple',1, ...
'StopbandAttenuation2',1, ...
'DesignMethod','ellip', ... % Design method
'MatchExactly','passband', ... % Design method options
'SampleRate',7000)
fvtool(bpiir)

%cascade form realization of FIR and IIR filters.
clc; clear;
close all;
b = [4 5 6]; %numerator coefficient of direct form
a = [1 2 3]; %denominator coefficient of direct form
% compute gain coefficient
b0 = b(1);
x = [1 2 3 8 9 4 6 7 10]; %input sequence
[k, l] = size(b);
n = length(x);
w  = zeros(k+1,n);
w(1,:)=x;
for i=1:1:k
w(i+1,:)=filter(b(i,:),a(i,:),w(i,:));
end
y = b0*w(k+1,:);
disp('output of the final filter operation')
disp(y)
stem(y)
```

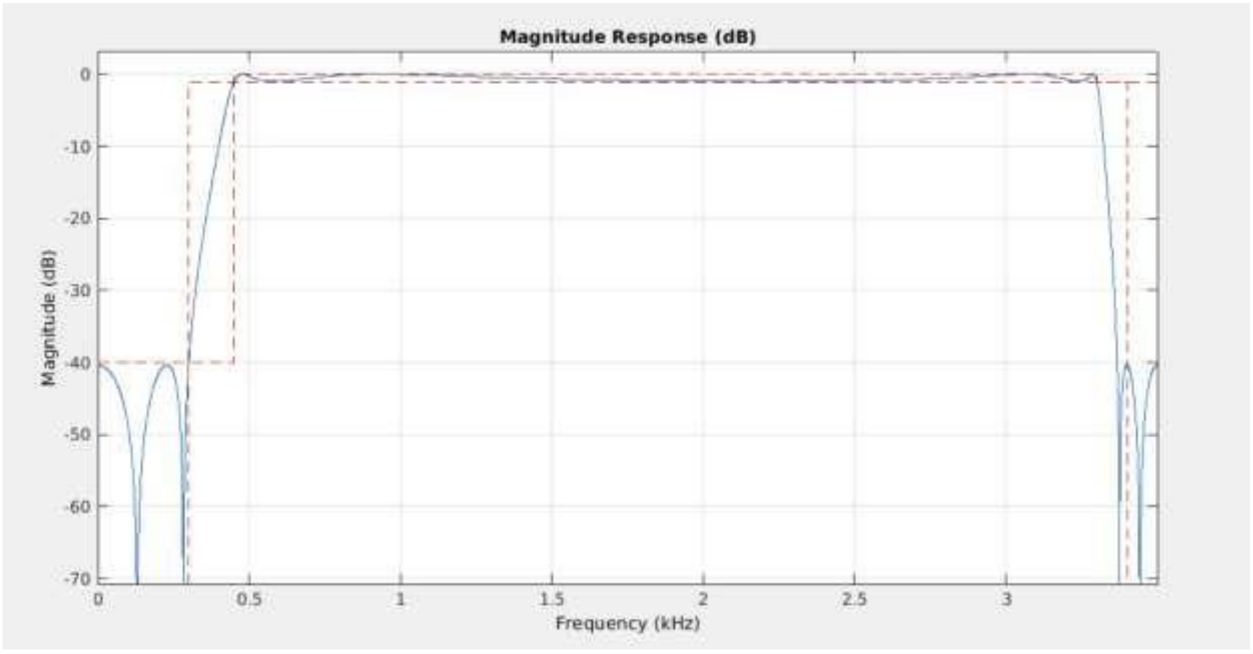
PLOT:

FIR:



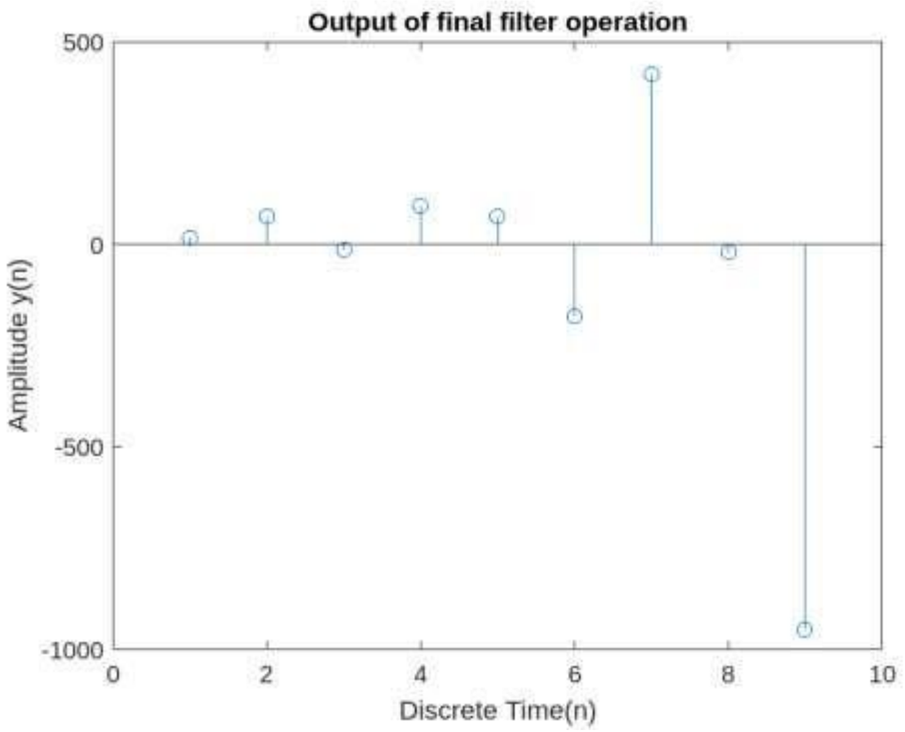
Magnitude Response (dB)
Magnitude (dB)
Frequency (kHz)
0
-10
-20
-30
-40
-50
-60
-70
0
0.5
1
1.5
2
2.5
3

IIR:



Magnitude Response (dB)
Magnitude (dB)
Frequency (kHz)
0
-10
-20
-30
-40
-50
-60
-70
0
0.5
1
1.5
2
2.5
3

Cascade realization of FIR/IIR filter:



RESULT and CONCLUSION:

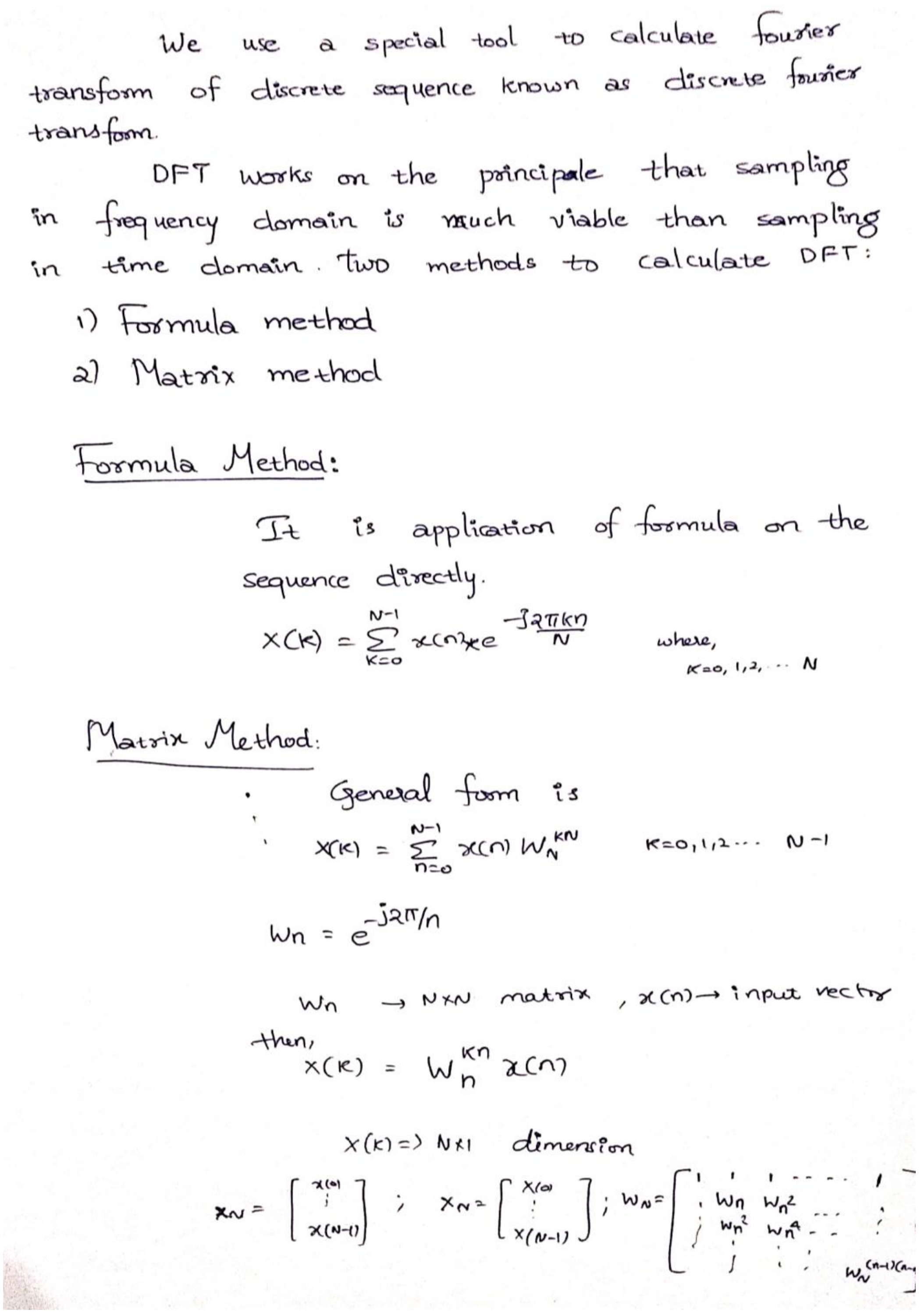
FIR and IIR filter were designed, and the magnitude responses were plotted using MATLAB.

EXPERIMENT-9

AIM OF THE EXPERIMENT: - Computation of N-point DFT and FFT of the length-N sequence using MATLAB and implement using MATLAB

SOFTWARE REQUIRED: - MATLAB

THEORY: -



We use a special tool to calculate fourier transform of discrete sequence known as discrete fourier transform.

DFT works on the principale that sampling in frequency domain is much viable than sampling in time domain. two methods to calculate DFT:

1) Formula method
2) Matrix method

Formula Method:

It is application of formula on the sequence directly.

$$X(K) = \sum_{K=0}^{N-1} x(n) e^{\frac{-j2\pi kn}{N}}$$ where, $K = 0, 1, 2, \cdots N$

Matrix Method:

- General form is

$$X(K) = \sum_{n=0}^{N-1} x(n) W_N^{KN} \quad K = 0, 1, 2 \cdots N-1$$

$$W_n = e^{-j2\pi/n}$$

$W_n \rightarrow N \times N$ matrix , $x(n) \rightarrow$ input vector

then,

$$X(K) = W_n^{kn} x(n)$$

$X(K) \Rightarrow N \times 1$ dimension

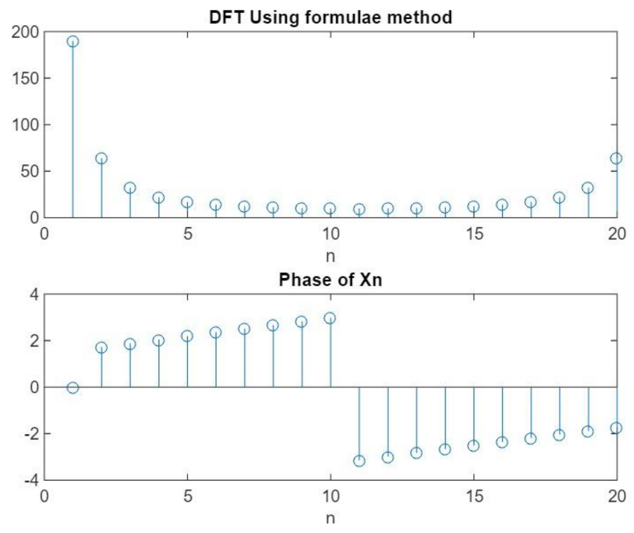
$$x_N = \begin{bmatrix} x(0) \\ \vdots \\ x(N-1) \end{bmatrix} ; \quad X_N = \begin{bmatrix} X(0) \\ \vdots \\ X(N-1) \end{bmatrix} ; \quad W_N = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_n & W_n^2 & \cdots & \vdots \\ 1 & W_n^2 & W_n^4 & \cdots & \vdots \\ \vdots & \vdots & \vdots & & W_N^{(n-1)(n-} \end{bmatrix}$$

CODE: -

DFT USING FORMULAE METHOD:

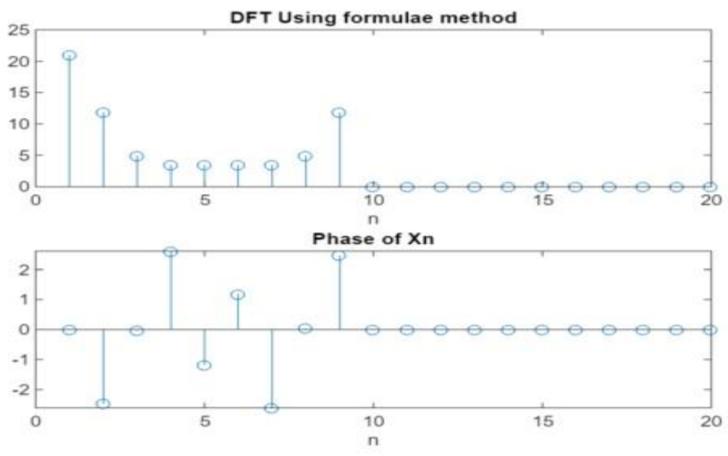
```
xn=(0:19);
N=length(xn);
dft_x=zeros(1,20);
for k=1:N
     for n=1:N
     dft_x(k)=dft_x(k)+xn(n)*exp(-1j*2*pi*(k-1)*((n-1)/N));
    end
end
disp('DFT')
disp(dft_x)
disp('MAGNITUDE OF THE VALUES OF DFT') disp(abs(dft_x))
subplot(2,1,1)
stem(abs(dft_x))
title('DFT Using formulae method') xlabel('n')

disp('PHASE OF THE DFT points')
disp(angle(dft_x))
subplot(2,1,2)
stem(angle(dft_x))
title('Phase of Xn')
xlabel('n')
```

RESULTS: -

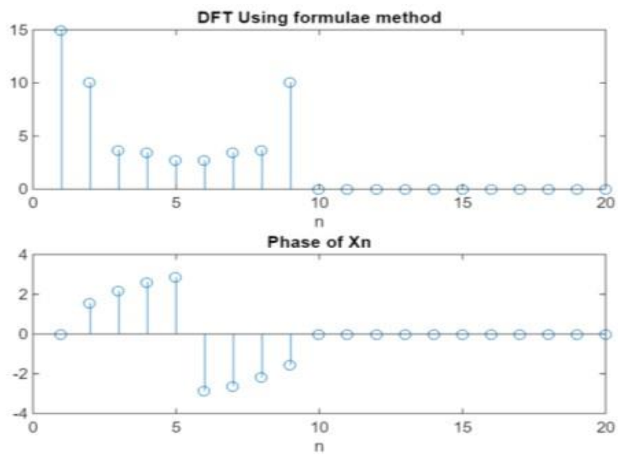


WITH PADDING: -

```
xn=[1 2 3 4 5 6 0 0 0];
N=length(xn);
dft_x=zeros(1,20);
for k=1:N
     for n=1:N

      dft_x(k)=dft_x(k)+xn(n)*exp(1j*2*pi*(k1)*((n1)/N));
      end
end

disp('DFT')
disp(dft_x)
disp('MAGNITUDE OF THE VALUES OF DFT')
disp(abs(dft_x))
subplot(2,1,1)
stem(abs(dft_x))
title('DFT Using formulae method')
xlabel('n')

disp('PHASE OF THE DFT points')
disp(angle(dft_x))
subplot(2,1,2)
stem(angle(dft_x))
title('Phase of Xn')
xlabel('n')
```



```
xn=[0 0 0 0 1 2 3 4 5];
N=length(xn);
dft_x=zeros(1,20);
for k=1:N
     for n=1:N
     dft_x(k)=dft_x(k)+xn(n)*exp(-1j*2*pi*(k-1)*((n-1)/N));
     end
end

disp('DFT')
disp(dft_x)
disp('MAGNITUDE OF THE VALUES OF DFT')
disp(abs(dft_x))
subplot(2,1,1)
stem(abs(dft_x))
title('DFT Using formulae method')
xlabel('n')

disp('PHASE OF THE DFT points')
disp(angle(dft_x))
subplot(2,1,2)
stem(angle(dft_x))
title('Phase of Xn')
xlabel('n')
```

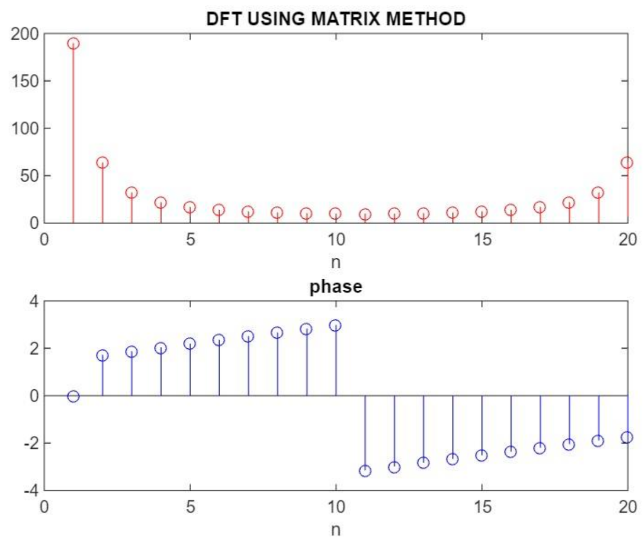


DFT USING MATRIX METHOD: -

```
xn=(0:19);
N=length(xn);
 for k=1:N
      for n=1:N
      Wn(k,n)=(exp(-1j*2*(pi/N))).^((k-1)*(n-1));
     end
end
```

```
dftm=Wn*(xn');
disp('DFT')
disp(dftm)
disp('Magnitude of DFT points')
disp(dftm)
subplot(2,1,1)
stem(abs(dftm),'r')
title('DFT USING MATRIX METHOD')
xlabel('n')
%disp(Wn)
subplot(2,1,2)
ang=angle(dftm);
stem(ang,'b');
title('phase')
xlabel('n')
```

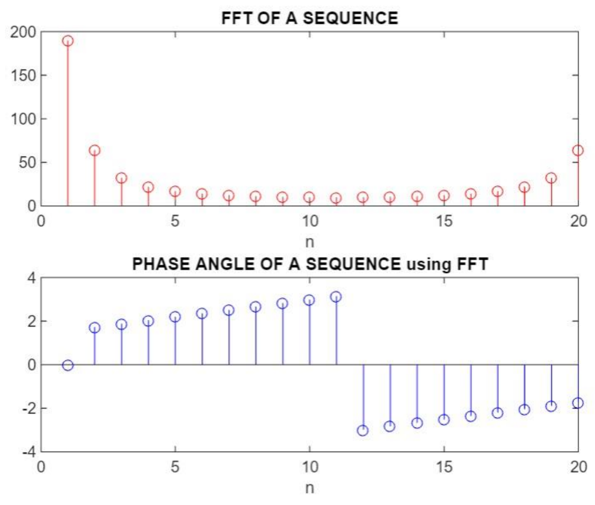
RESULTS: -

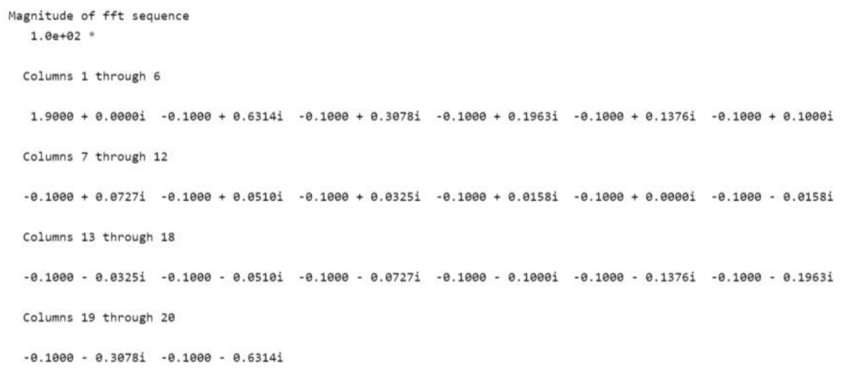


FFT OF A SEQUENCE:-

```
xn=(0:19);
N=length(xn);
fast_ft=fft(xn);
disp('Magnitude of fft sequence')
disp(fast_ft)
subplot(2,1,1)
stem(abs(fast_ft),'r')
title('FFT OF A SEQUENCE')
xlabel('n')
subplot(2,1,2)
stem(angle(fast_ft),'b')
title('PHASE ANGLE OF A SEQUENCE using FFT')
xlabel('n')
```

RESULTS: -





```
Magnitude of fft sequence
   1.0e+02 *

  Columns 1 through 6

   1.9000 + 0.0000i  -0.1000 + 0.6314i  -0.1000 + 0.3078i  -0.1000 + 0.1963i  -0.1000 + 0.1376i  -0.1000 + 0.1000i

  Columns 7 through 12

  -0.1000 + 0.0727i  -0.1000 + 0.0510i  -0.1000 + 0.0325i  -0.1000 + 0.0158i  -0.1000 + 0.0000i  -0.1000 - 0.0158i

  Columns 13 through 18

  -0.1000 - 0.0325i  -0.1000 - 0.0510i  -0.1000 - 0.0727i  -0.1000 - 0.1000i  -0.1000 - 0.1376i  -0.1000 - 0.1963i

  Columns 19 through 20

  -0.1000 - 0.3078i  -0.1000 - 0.6314i
```

IFFT OF A FFT SEQUENCE

```
xn=(0:19);
N=length(xn);
fast_ft=fft(xn);
xift=ifft(fast_ft);
disp('Inverse fast fourier transform =')
disp(xift)
```

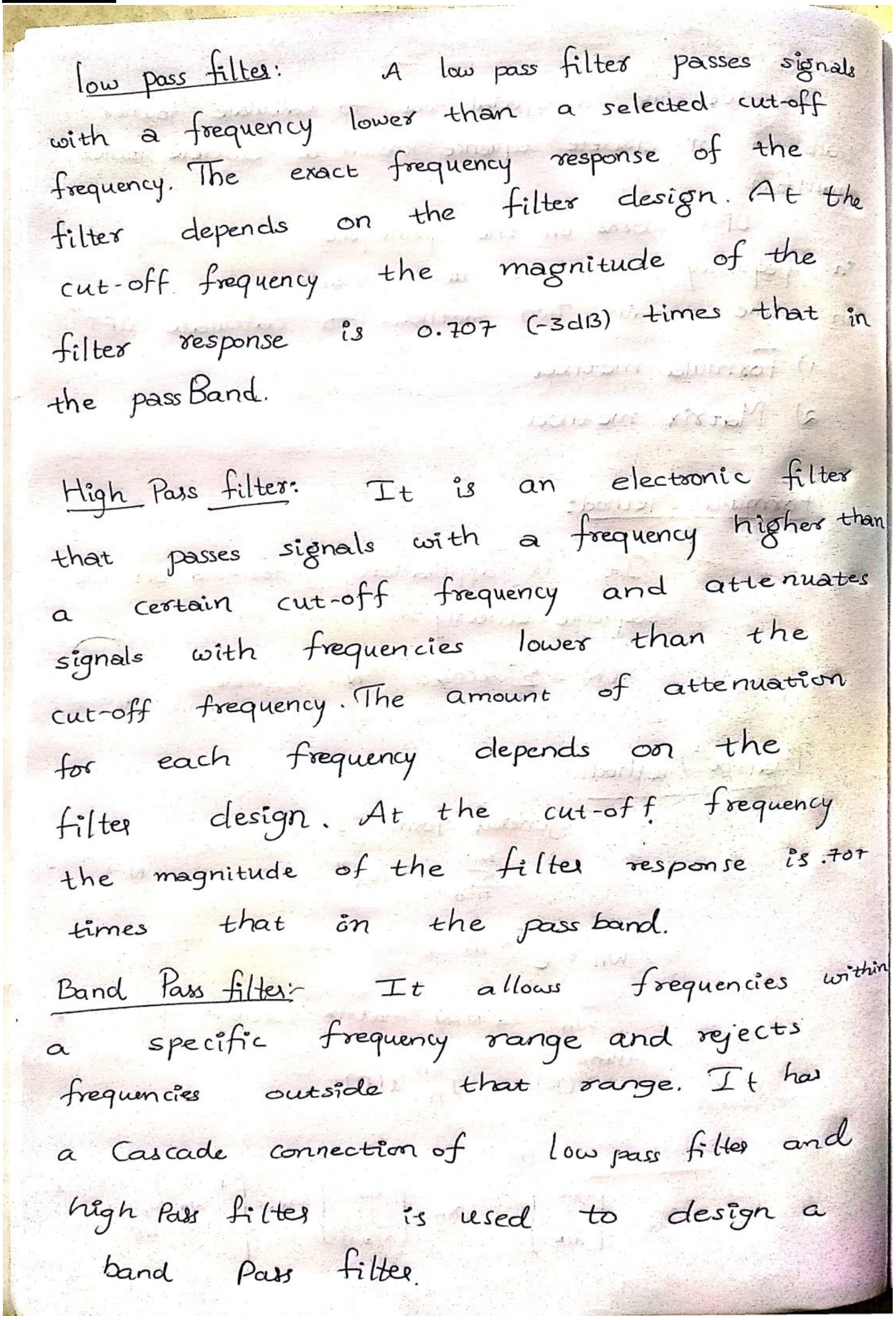
```
Inverse fast fourier transform =
  Columns 1 through 14

         0    1.0000    2.0000    3.0000    4.0000    5.0000    6.0000    7.0000    8.0000    9.0000   10.0000   11.0000   12.0000   13.0000

  Columns 15 through 20

   14.0000   15.0000   16.0000   17.0000   18.0000   19.0000
```

EXPERIMENT-10

AIM OF THE EXPERIMENT: - Design of digital filter (LP/HP) and evaluate its performance. Implement the filter using speech signal.

SOFTWARE USED: - MATLAB

THEORY: -



low pass filter: A low pass filter passes signals with a frequency lower than a selected cut-off frequency. The exact frequency response of the filter depends on the filter design. At the cut-off frequency the magnitude of the filter response is 0.707 (-3dB) times that in the pass Band.

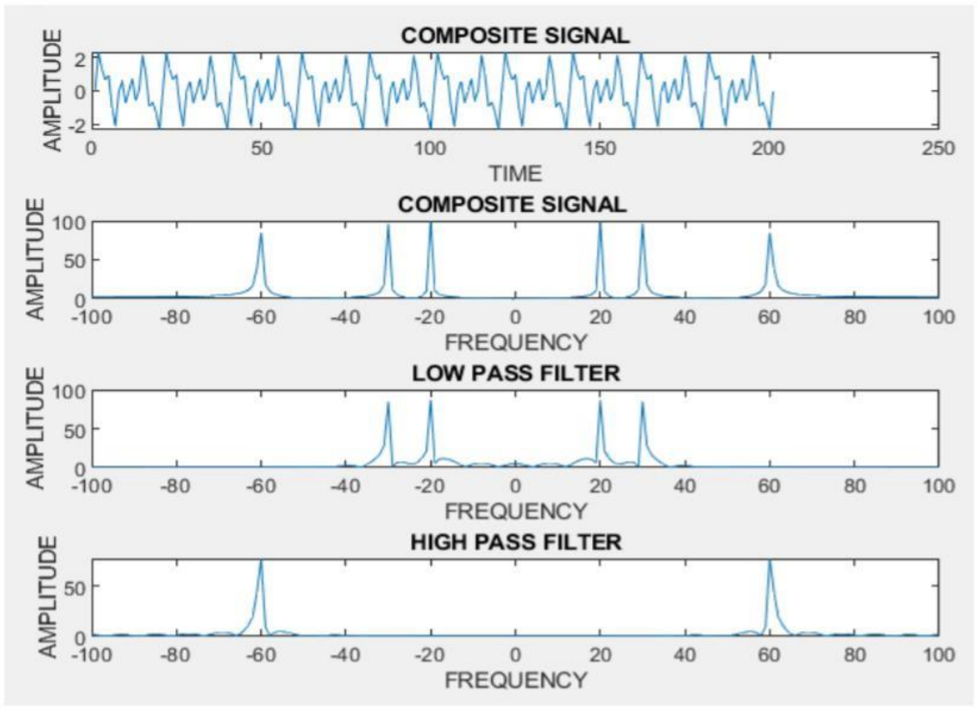
High Pass filter: It is an electronic filter that passes signals with a frequency higher than a certain cut-off frequency and attenuates signals with frequencies lower than the cut-off frequency. The amount of attenuation for each frequency depends on the filter design. At the cut-off frequency the magnitude of the filter response is .707 times that in the pass band.

Band Pass filter:- It allows frequencies within a specific frequency range and rejects frequencies outside that range. It has a Cascade connection of low pass filter and high Pass filter is used to design a band Pass filter.

CODE:-

```
clear close
all clc
%COMPOSITE SIGNAL
f1 = 20;
 %1st frequency
f2 = 30;
%2nd frequency
f3 = 60;
% 3rd frequency
fs = 200;
%sampling frequency
x = 0:1/fs:1;
%time interval
n = -100:100;
y = sin(2*pi*f1*x)+sin(2*pi*f2*x)+sin(2*pi*f3*x);
%TIME DOMAIN
subplot(4,1,1) plot(y)
ylabel('AMPLITUDE');
xlabel('TIME');
title('COMPOSITE SIGNAL');
%FREQUENCY DOMAIN
subplot(4,1,2) yf = fft(y);
plot(n,fftshift(abs(yf)));
ylabel('AMPLITUDE');
xlabel('FREQUENCY');
title('COMPOSITE SIGNAL ');

%lowpass filter
subplot(4,1,3) %filter
coefficient
a = fir1(50,40/(fs/2),'low');%cutoff frequency =40,order = 50,normalized
cutoff frequency = 40/(fs/2)
%filter
xtr = filter(a,1,y);% a => numerator,1 => denominator
yftr = abs(fft(xtr));
plot(n,fftshift(yftr));
ylabel('AMPLITUDE');
xlabel('FREQUENCY');
title('LOW PASS FILTER');

%highpass filter
subplot(4,1,4)
%filter coefficient
ah = fir1(50,40/(fs/2),'high');%cutoff frequency =40,order = 50,normalized
cutoff frequency = 40/(fs/2)
%filter
xtrh = filter(ah,1,y); %a => numerator,1 => denominator
yftrh = abs(fft(xtrh));
```

```
plot(n,fftshift(yftrh));
ylabel('AMPLITUDE');
xlabel('FREQUENCY');
title('HIGH PASS FILTER');
```



CONCLUSION:

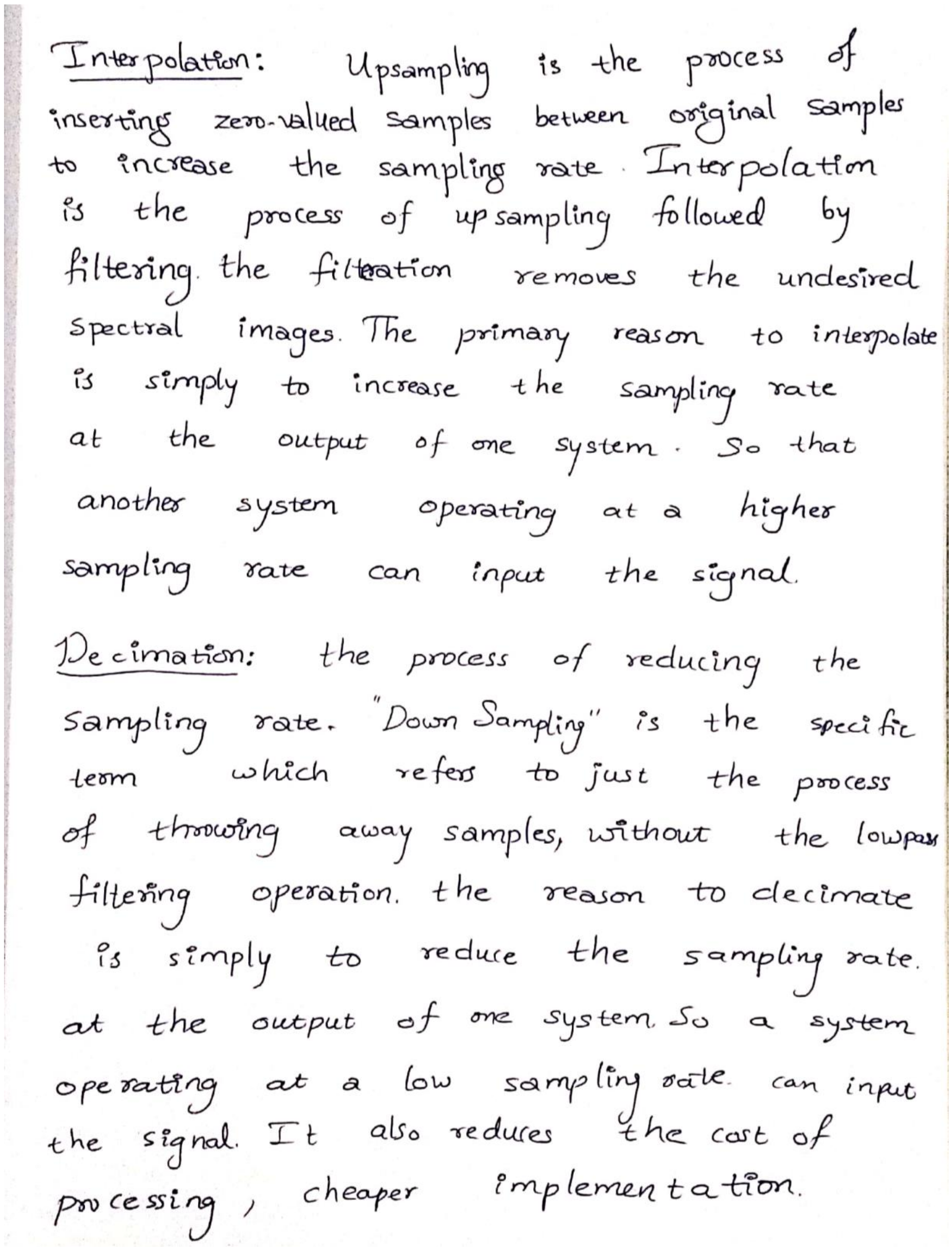
Designed a high pass and low pass filter, using filter functions, of cut off frequency 40 hertz and verified it by testing it on a composite sinusoidal signal and compared with its FFT signal .A composite signal was created and filtered into components less than 50 Hz and components greater than 50 Hz.

Experiment No. - 11

AIM OF THE EXPERIMENT: - Computation of N-point DFT and FFT of the length-N sequence using MATLAB and implement using MATLAB

SOFTWARE REQUIRED: - MATLAB

THEORY: -



Interpolation: Upsampling is the process of inserting zero-valued samples between original samples to increase the sampling rate. Interpolation is the process of upsampling followed by filtering. the filtration removes the undesired spectral images. The primary reason to interpolate is simply to increase the sampling rate at the output of one system. So that another system operating at a higher sampling rate can input the signal.

Decimation: the process of reducing the sampling rate. "Down Sampling" is the specific term which refers to just the process of throwing away samples, without the lowpass filtering operation. the reason to decimate is simply to reduce the sampling rate. at the output of one system. So a system operating at a low sampling rate. can input the signal. It also reduces the cost of processing, cheaper implementation.

CODE:

1) Decimation:

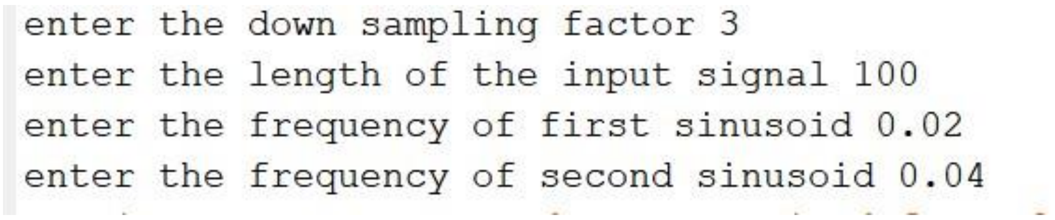
```
clc;
clear all;
close all;
D=input('enter the down sampling factor ');
L=input('enter the length of the input signal ');
f1=input('enter the frequency of first sinusoid ');
f2=input('enter the frequency of second sinusoid ');
n=0:L-1;
x=sin(2*pi*f1*n)+sin(2*pi*f2*n);
y=decimate(x,D,'fir');
subplot(2,1,1);
stem(n,x(1:L));
title('input sequence');
xlabel('time(n)');
ylabel('amplitude');
subplot(2,1,2)
m=0:(L/D)-1;
stem(m,y(1:L/D));
title('Decimated sequence');
xlabel('time(n)');
ylabel('amplitude');
```

2) Interpolation:

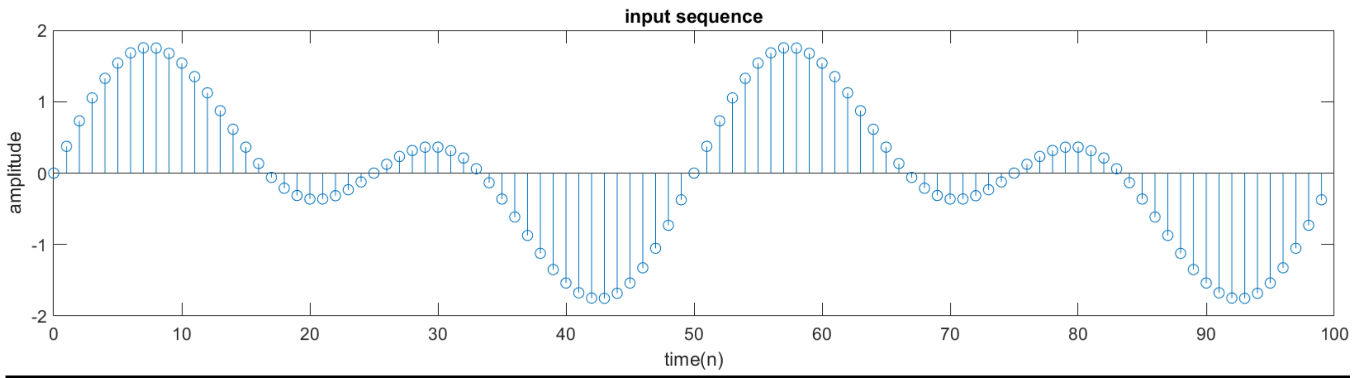
```
clc;
clear all;
close all;
L=input('enter the up sampling factor ');
N=input('enter the length of the input signal ');
% Length should be greater than 8
f1=input('enter the frequency of first sinusoid ');
f2=input('enter the frequency of second sinusoid ');
n=0:N-1;
x=sin(2*pi*f1*n)+sin(2*pi*f2*n);
y=interp(x,L);
subplot(2,1,1)
stem(n,x(1:N))
title('input sequence');
xlabel('time(n)');
ylabel('amplitude');
subplot(2,1,2)
m=0:N*L-1;
stem(m,y(1:N*L))
title('output sequence ');
xlabel('time(n)');
ylabel('amplitude');
```

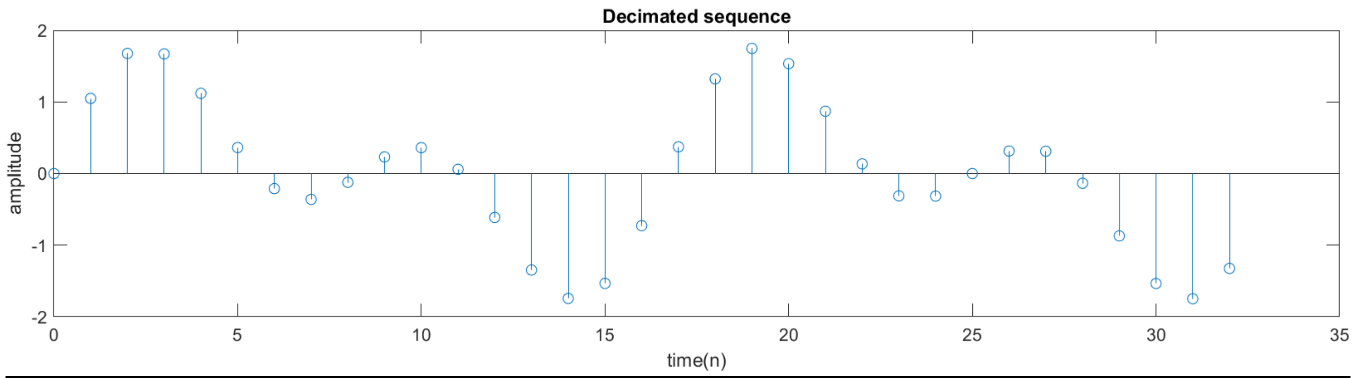
PLOTS:

1) Decimation:

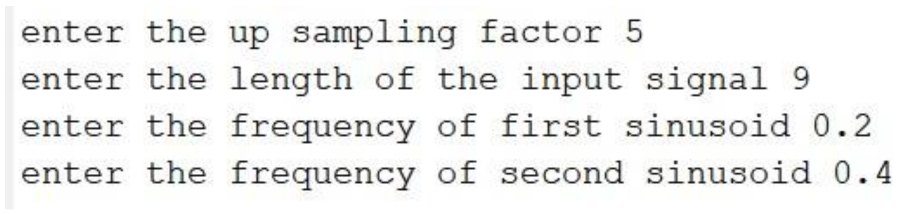


```
enter the down sampling factor 3
enter the length of the input signal 100
enter the frequency of first sinusoid 0.02
enter the frequency of second sinusoid 0.04
```

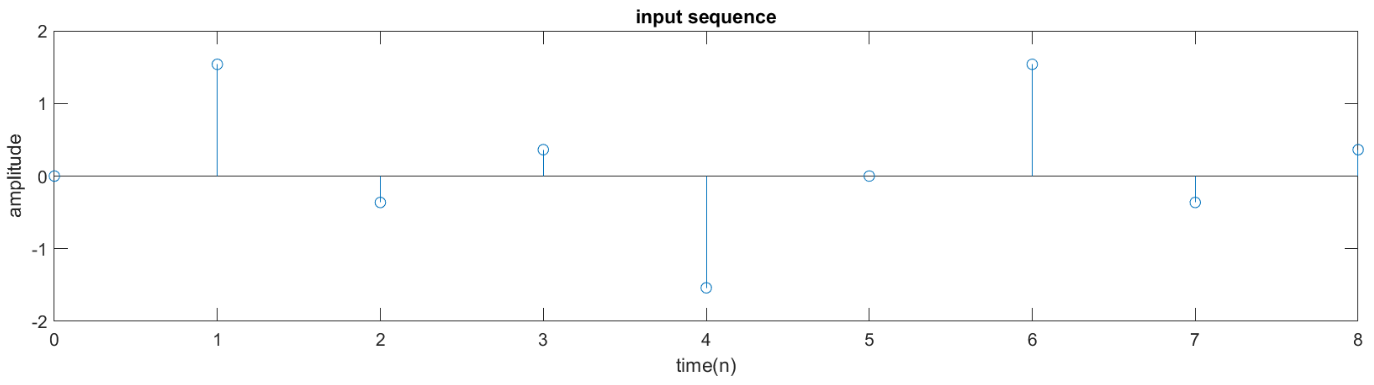


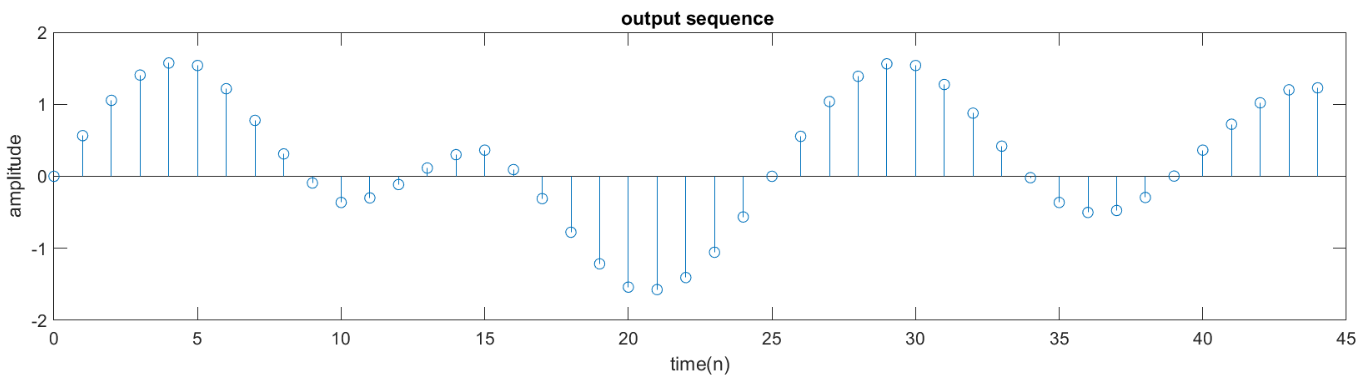


2) Interpolation:



```
enter the up sampling factor 5
enter the length of the input signal 9
enter the frequency of first sinusoid 0.2
enter the frequency of second sinusoid 0.4
```





Result:

Decimation, interpolation, and sampling rate conversion of a signal was successfully done.