Name:	Anubhav Singh
Admission No:	U19EC135
Branch:	Electronics and Communication
Course:	Digital Signal and Processing
Batch:	A5
Co-Ordinator:	Mehul C. Patel sir
Course Code:	EC305

Date : 28/07/21

PRACTICAL NO: 01

Aim:

To verify the sampling theorem using MATLAB.

Theory:

Sampling is a process that converts continuous signals into discrete time signals. Discrete signal is then quantized to digital signal using Quantization.

SAMPLING THEOREM:

A Band-Limited continuous time signal can be represented by its samples and can be recovered back when sampling frequency f_s is greater than or equal to the twice the highest frequency component of the message signal.

Over-sampling:

Over-sampling occurs when the sampling frequency is greater than twice the frequency of the message signal.

$$f_s > 2f_m$$

In over-sampling, we can reconstruct the original signal.

Perfect-sampling:

Perfect-sampling occurs when the sampling frequency is equal to twice the frequency of the message signal.

$$f_s = 2f_m$$

In perfect-sampling, we can reconstruct the original signal.

Under-sampling:

Under-sampling occurs when the sampling frequency is less than twice the frequency of the message signal.

$$f_s < 2f_m$$

In perfect-sampling, we can't reconstruct the original signal.

ALGORITHM:

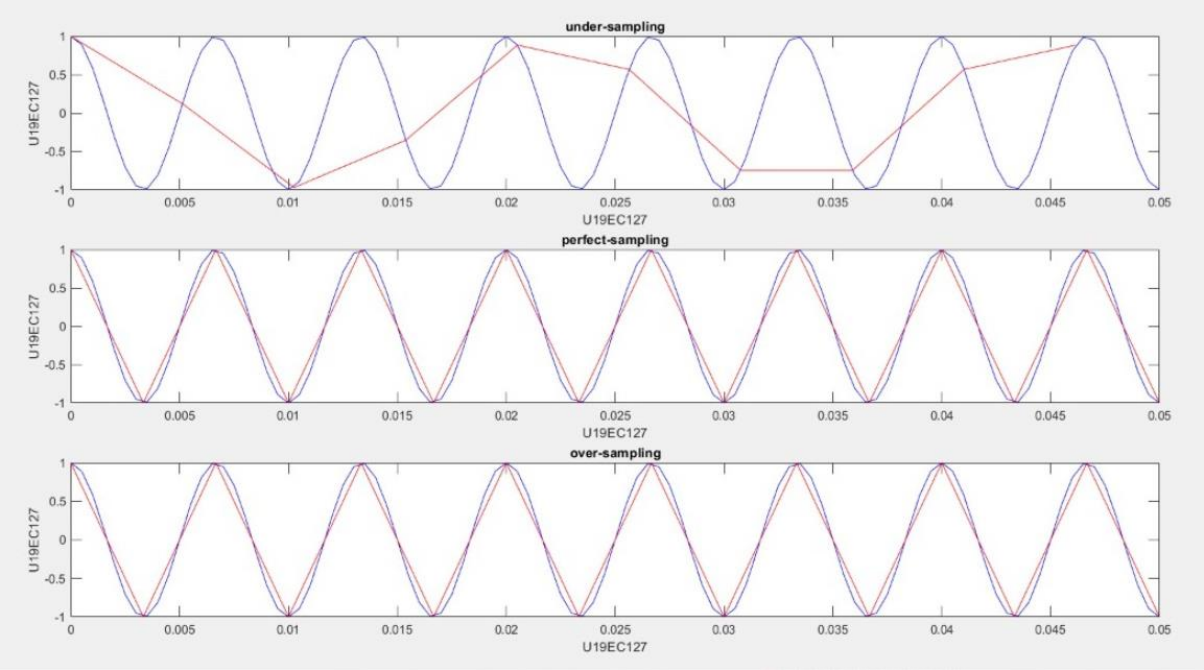
1. Define analog frequency fafor the input sinusoidal signal and also choose sampling frequency much greater than fa. Let k be the scaling factor
2. Define the time period n as per the sampling frequency and plot the sampled signal using this time period.

3. Define the reconstruction time period and reconstructed output vector.
4. Multiply the sampled signal with sinc pulse at various sampling instants and accumulate all the values to obtain a reconstructed signal.
5. Choose different frequencies and repeat the same.

MATLAB code:

```
clc

clear all
close all

tfinal=0.05;
t=0:0.0005:tfinal;
fd=150;
xt=cos(2*pi*fd*t);
fs1=1.3*fd;
n1=0:1/fs1:tfinal;
xn=cos(2*pi*n1*fd);
subplot(3,1,1)
plot(t,xt,'b',n1,xn,'r');
title('under-sampling')

fs2=2*fd;
n2=0:1/fs2:tfinal;
xn=cos(2*pi*n2*fd);
subplot(3,1,2)
plot(t,xt,'b',n2,xn,'r');
title('perfect-sampling')

fs3=4*fd;
n3=0:1/fs3:tfinal;
xn=cos(2*pi*n3*fd);
subplot(3,1,3)
plot(t,xt,'b',n3,xn,'r');
title('over-sampling')
```

Output:



Conclusion:

Therefore, we conclude that, as long as the sampling frequency is greater than or equal to frequency of the highest component of the message signal, the original signal can be recovered.

Date : 04/08/21

PRACTICAL NO: 02

Aim:

To implement “Time-shifting”, “Time-scaling”, “Time-reversal” and “addition of signals” in MATLAB for discrete as well as continuous signals.

Theory :

Time-Shifting:

Suppose that we have a signal $x(t)$ and we define a new signal by adding/subtracting a finite time value to/from it. We now have a new signal, $y(t)$. The mathematical expression for this would be $x(t \pm t_0)$.

Graphically, this kind of signal operation results in a positive or negative “shift” of the signal along its time axis.

Time-Scaling:

A signal x(t) is scaled in time by multiplying the time variable by a positive constant b, to produce x(bt). A positive factor of b either expands (0 < b < 1) or compresses (b > 1) the signal in time.

Time-Reversal:

`Continuous time: replace t with −t, time reversed signal is x(−t)`

`Discrete time: replace n with −n, time reversed signal is x[−n]`

Addition of two signals:

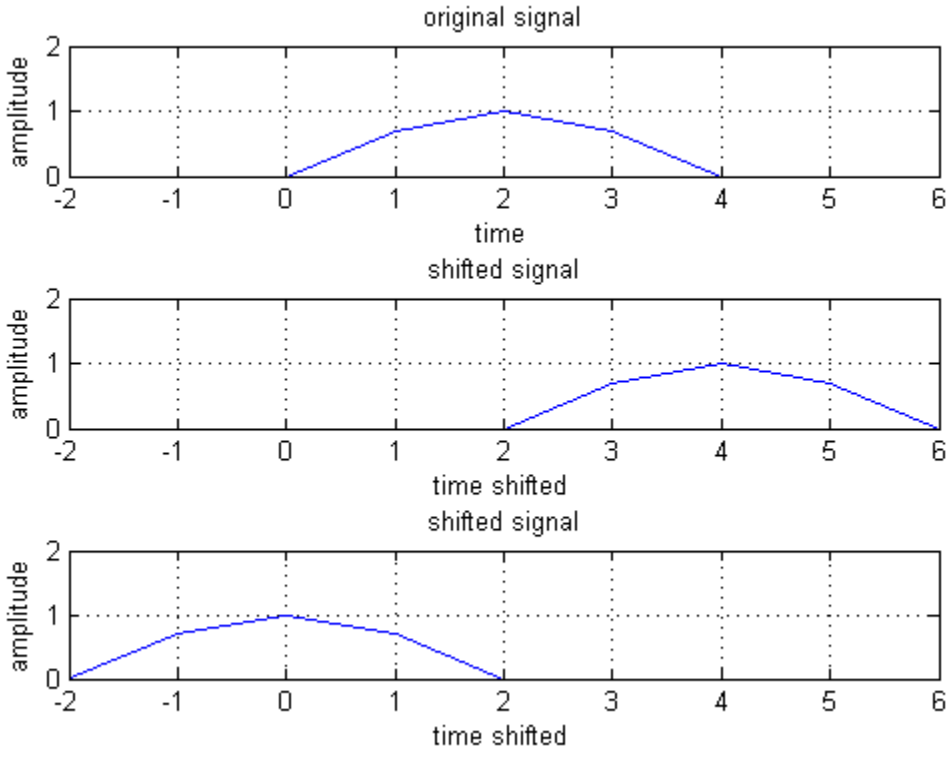
Signal addition—Two signals $x(t)$ and $y(t)$ are added to obtain their sum $z(t)$.

MATLAB Code:

Time shifting for continuous signal

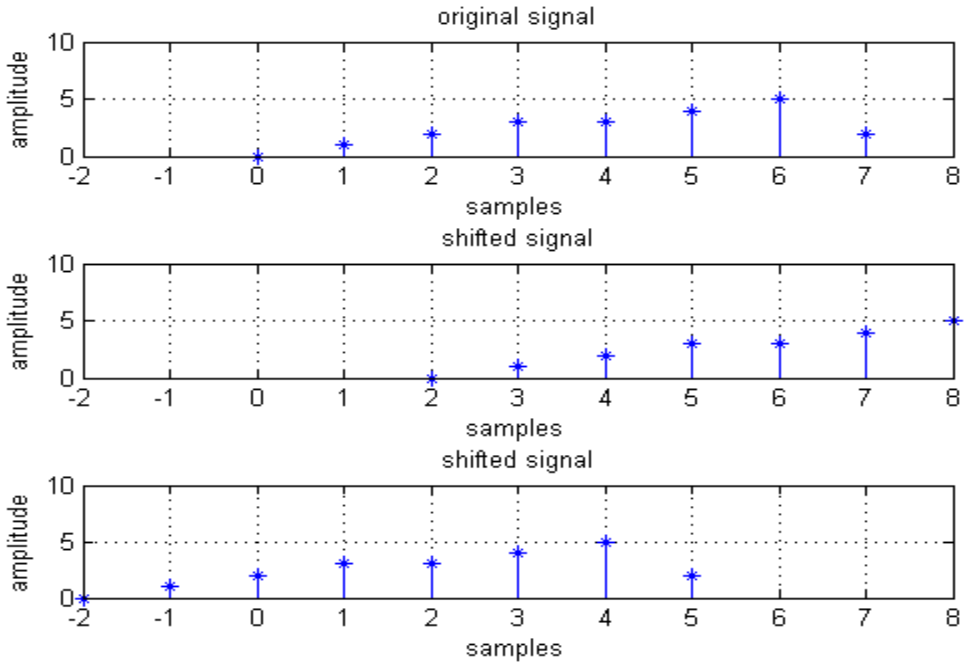
```
t=0:5;

x=sin((pi*t)/4);

subplot(3,1,1);

plot(t,x);

xlabel('time')
ylabel('amplitude')
```

```
title('original signal')
axis([-2 6 0 2]);
grid
t0=t+2;
subplot(3,1,2);
plot(t0,x);
xlabel('time shifted')
ylabel('amplitude')
title('shifted signal')
axis([-2 6 0 2]);
grid
t1=t-2;
subplot(3,1,3);
plot(t1,x);
xlabel('time shifted')
ylabel('amplitude')
title('shifted signal')
axis([-2 6 0 2]);
grid
```



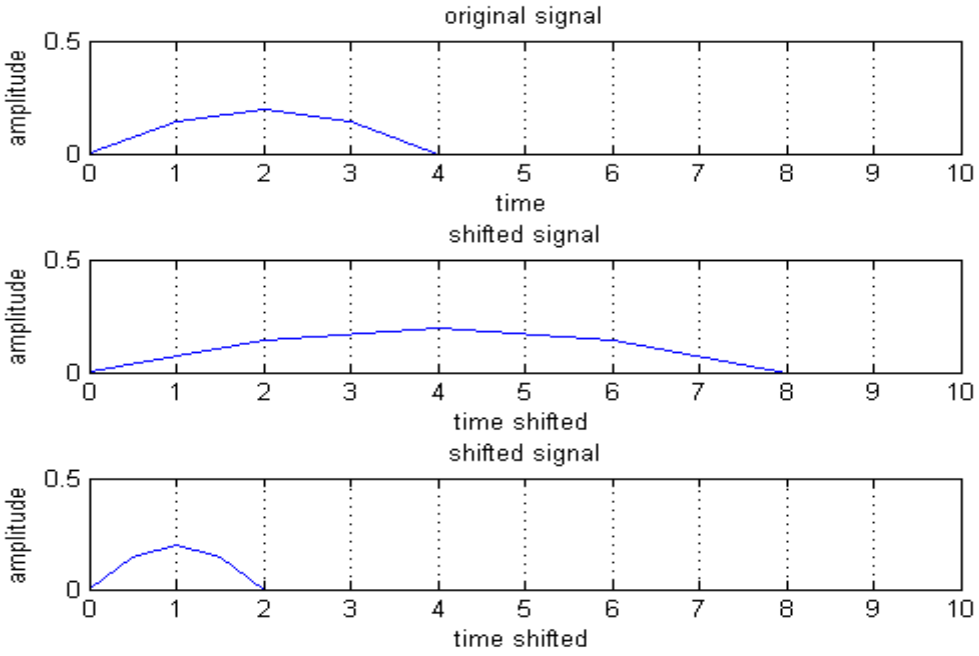
Time shifting for discrete time signal

```
n=0:7;

x=[0 1 2 3 3 4 5 2];

subplot(3,1,1);

stem(n,x,'b*-');
xlabel('samples')
ylabel('amplitude')
title('original signal')
axis([-2 8 0 10]);
grid

n1=n+2;
subplot(3,1,2);
stem(n1,x,'b*-');
xlabel('samples')
ylabel('amplitude')
title('shifted signal')
axis([-2 8 0 10]);
grid

n2=n-2;
subplot(3,1,3);
stem(n2,x,'b*-');
xlabel('samples')
ylabel('amplitude')
title('shifted signal')
axis([-2 8 0 10]);
grid
```



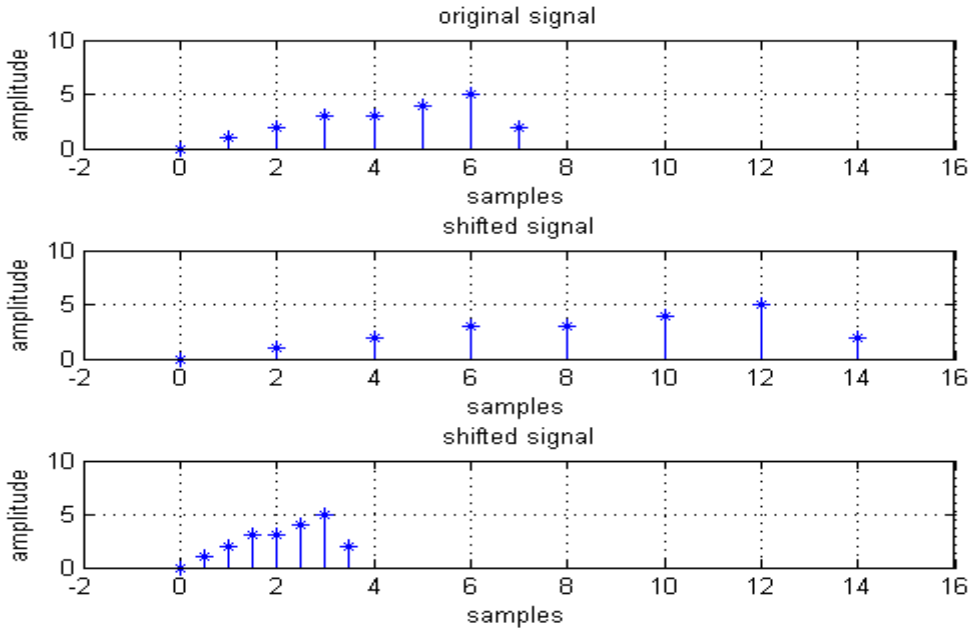
Time scaling for continuous signal

```
t=0:5;

x=0.2*(sin((pi*t)/4));

subplot(3,1,1);

plot(t,x);

xlabel('time')
ylabel('amplitude')
title('original signal')
axis([0 10 0 0.5])
grid
t0=2*t;
subplot(3,1,2);
plot(t0,x);
xlabel('time shifted')
ylabel('amplitude')
title('shifted signal')
axis([0 10 0 0.5])
grid
t1=t/2;
subplot(3,1,3);
plot(t1,x);
xlabel('time shifted')
ylabel('amplitude')
title('shifted signal')
axis([0 10 0 0.5])
grid
```



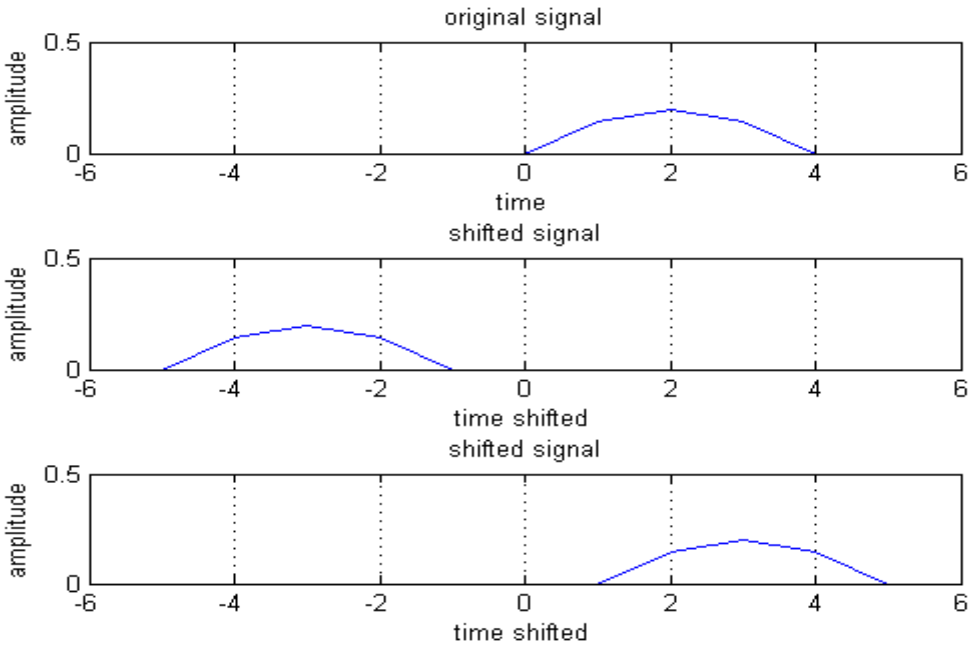
Time scaling for discrete time signal

```
n=0:7;

x=[0 1 2 3 3 4 5 2];

subplot(3,1,1);

stem(n,x,'b*-');
xlabel('samples')
ylabel('amplitude')
title('original signal')
axis([-2 16 0 10]);
grid

n1=n*2;
subplot(3,1,2);
stem(n1,x,'b*-');
xlabel('samples')
ylabel('amplitude')
title('shifted signal')
axis([-2 16 0 10]);
grid

n2=n/2;
subplot(3,1,3);
stem(n2,x,'b*-');
xlabel('samples')
ylabel('amplitude')
title('shifted signal')
axis([-2 16 0 10]);
grid
```



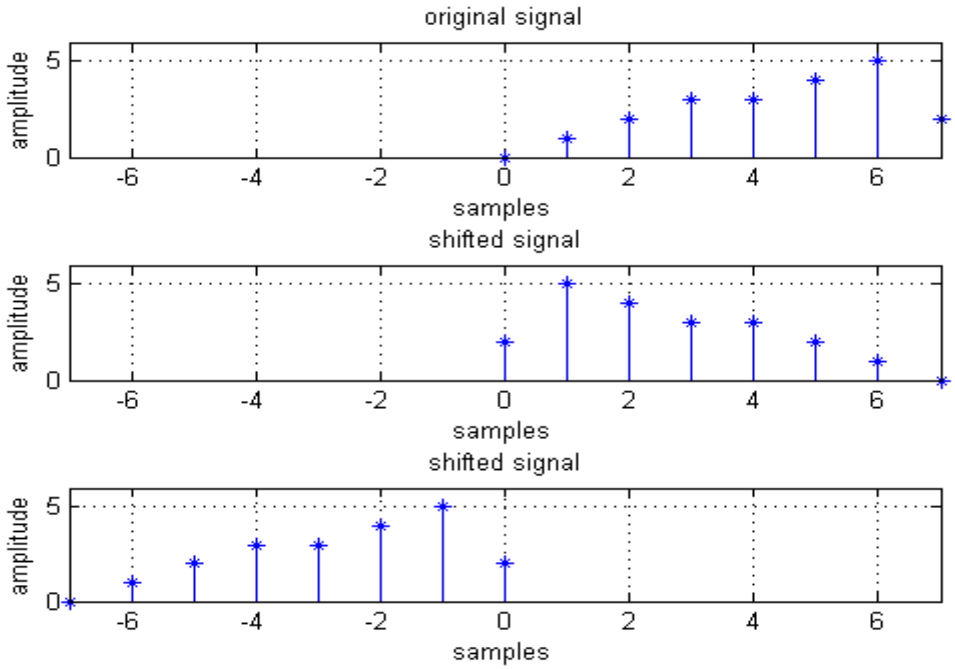
Time reversal for continuous signal

```
t=0:5;

x=0.2*(sin((pi*t)/4));

subplot(3,1,1);

plot(t,x);

xlabel('time')
ylabel('amplitude')
title('original signal')
axis([-6 6 0 0.5])
grid
t0=-fliplr(t);
subplot(3,1,2);
plot(t0,x);
xlabel('time shifted')
ylabel('amplitude')
title('shifted signal')
axis([-6 6 0 0.5])
grid
t1=fliplr(t);
subplot(3,1,3);
plot(t1,x);
xlabel('time shifted')
ylabel('amplitude')
title('shifted signal')
axis([-6 6 0 0.5])
grid
```



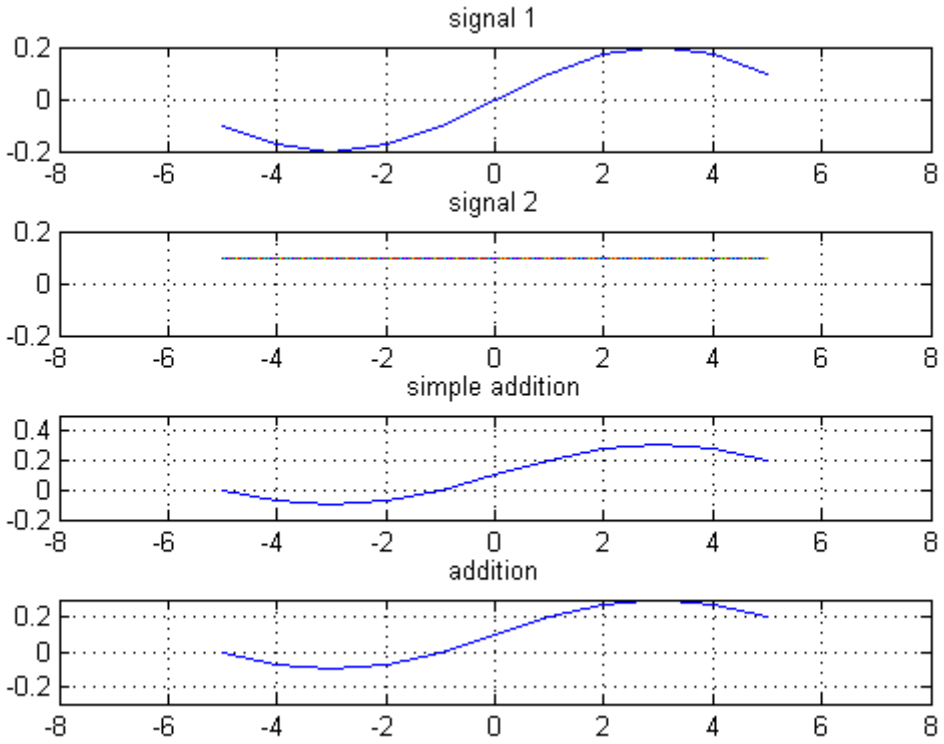
Time reversal for discrete time signal

```
n=0:7;

x=[0 1 2 3 3 4 5 2];

subplot(3,1,1);

stem(n,x,'b*-');
xlabel('samples')
ylabel('amplitude')
title('original signal')
axis([-7 7 0 6]);
grid

n1=fliplr(n);
subplot(3,1,2);
stem(n1,x,'b*-');
xlabel('samples')
ylabel('amplitude')
title('shifted signal')
axis([-7 7 0 6]);
grid

n2=-fliplr(n);
subplot(3,1,3);
stem(n2,x,'b*-');
xlabel('samples')
ylabel('amplitude')
title('shifted signal')
axis([-7 7 0 6]);
grid
```



Addition of two continuous signals

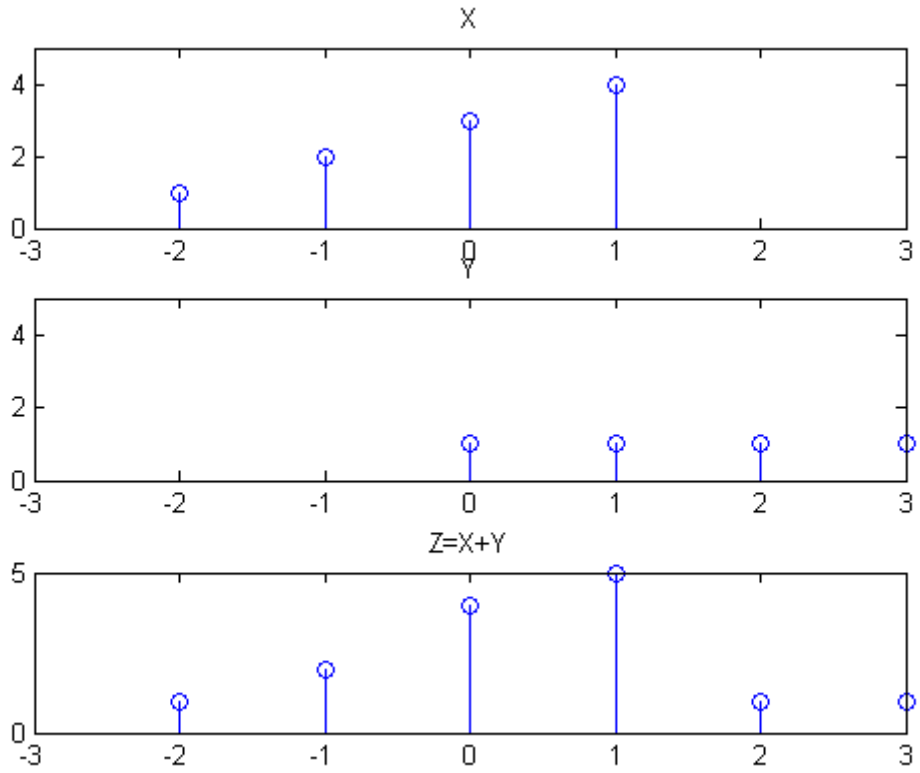
```
t1=-5:1:5;

x1=0.2*sin((pi*t1)/6);

t2=-5:0.0001:5;

x2=0.1;

t=min(min(t1),min(t2)):max(max(t1),max(t2));

y1=zeros(1,length(t));

y2=zeros(1,length(t));

y1((t>=min(t1))&(t<=max(t1)))=x1();

y2((t>=min(t2))&(t<=max(t2)))=x2();

x=y1+y2;

y=x1+x2;

subplot(4,1,1);

plot(t1,x1);

title('signal 1');
axis([-8 8 -0.2 0.2]);
grid

subplot(4,1,2);
plot(t2,x2);
title('signal 2');
axis([-8 8 -0.2 0.2]);
grid

subplot(4,1,3);
plot(t1,y);
title('simple addition')
axis([-8 8 -0.2 0.5]);
grid

subplot(4,1,4);
plot(t,x);
title('addition');
axis([-8 8 -0.3 0.3]);
grid
```



Addition of two discrete signals

```
n1=-2:1;

x=[1 2 3 4];

subplot(3,1,1);

stem(n1,x);

title('X') ;
axis([-3 3 0 5]);

n2=0:3;
y=[1 1 1 1];

subplot(3,1,2);
stem(n2,y);
title('Y');
axis([-3 3 0 5]);

n3 =min (min(n1) ,min( n2 ) ) : max ( max ( n1 ) , max ( n2 ) );

s1 =zeros(1,length (n3) );

s2 =s1;
```

```
s1 (find ( ( n3>=min( n1 ) ) & ( n3 <=max ( n1 ) )==1 ) )=x;

s2 (find ( ( n3>=min ( n2 ) ) & ( n3 <=max ( n2 ))==1) )=y;

add=s1 +s2;

subplot(3,1,3)
stem(n3,add)
title('Z=X+Y');
axis([-3 3 0 5]);
```



Conclusion:

From this experiment, we can conclude that:

1. Time – shifting results in moving the signal to the right or left.
2. Time – scaling results in contraction or expansion of the signal.
3. Time – reversal results in reversal of signal around the y-axis.
4. Addition of continuous and discrete signals is implemented.

Date : 11/08/21

PRACTICAL NO: 03

Aim:

To implement linear convolution and circular convolution in MATLAB.

Theory:

Linear convolution is a mathematical operation used to express the relation between input and output of an LTI system.

Y(n) = x(n) * h(n)

$Y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k)$

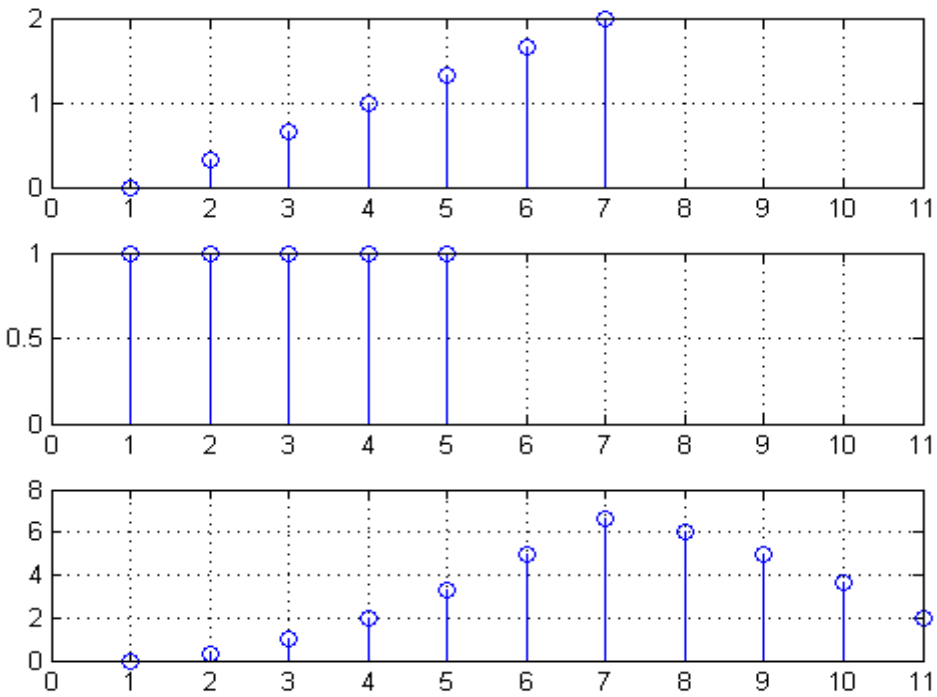
MATLAB code:

Linear convolution using In-Built function:

```
clc;

clear all;

close all;

a=0:6;
x=(a./3);
b=-2:2;
h=ones(1,length(b));
disp(x);
disp(h);
n1=length(x);
n2=length(h);
N=n1+n2-1;
y=zeros(1,N);
y=conv(x,h);
disp(y)

subplot(3,1,1)
stem(x)
axis([0 11 0 2])
grid
```

```
subplot(3,1,2)
stem(h)
axis([0 11 0 1])
grid

subplot(3,1,3)
stem(y)
axis([0 11 0 8])
grid
```

```
 0    0.3333    0.6667    1.0000    1.3333    1.6667    2.0000

    1     1     1     1     1

  Columns 1 through 7

         0    0.3333    1.0000    2.0000    3.3333    5.0000    6.6667

  Columns 8 through 11

    6.0000    5.0000    3.6667    2.0000
```

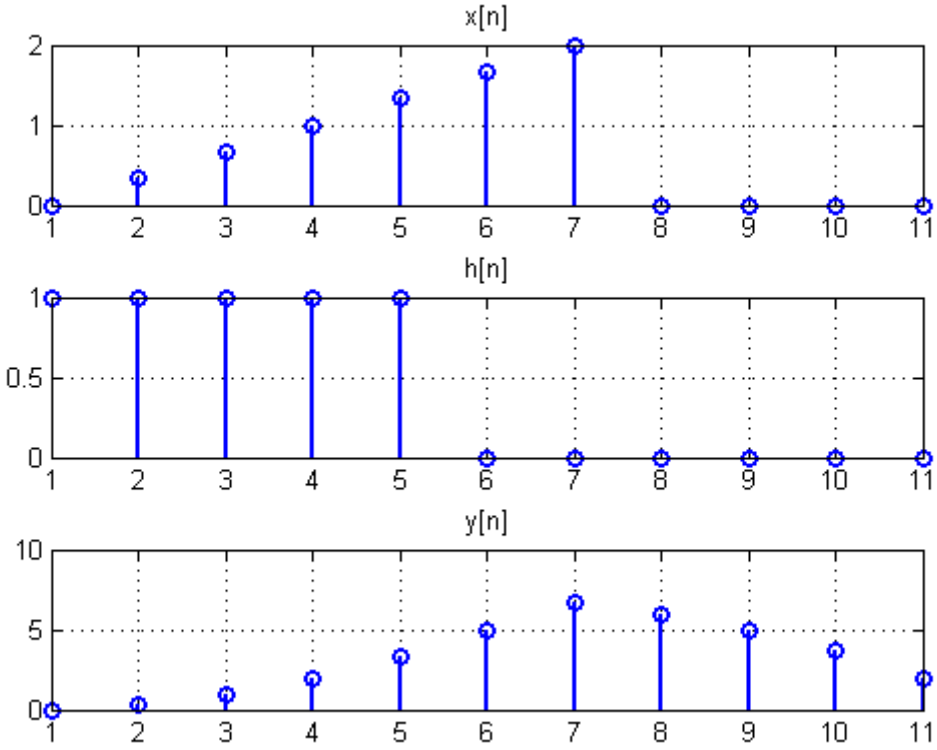


Linear convolution without using In-Built function:

```
clc;

clear all;
close all;
a=0:6;
x=(a./3);
b=-2:2;
h= ones(1,length(b));
disp(x);
disp(h);
n1=length(x);
n2=length(h);
N=n1+n2-1;
x=[x, zeros(1,N-n1)];
h=[h, zeros(1,N-n2)];
y= zeros(1,N);
for n=1:N
    y(1)=0;
    for k=1:N
    if(k<n+1)
      y(n)=y(n) + x(k)*h(n-k+1);
    end
    end
end
disp(y);
subplot(3,1,1);
stem(x,'linewidth',2);
title('x[n]');
grid on;

subplot(3,1,2);
stem(h,'linewidth',2);
title('h[n]');
grid on;

subplot(3,1,3);
stem(y,'linewidth',2);
title('y[n]');
grid on;
```

```
         0    0.3333    0.6667    1.0000    1.3333    1.6667    2.0000

     1     1     1     1     1

  Columns 1 through 7

         0    0.3333    1.0000    2.0000    3.3333    5.0000    6.6667

  Columns 8 through 11
```

```
    6.0000    5.0000    3.6667    2.0000
```



Circular convolution:

```
clc;

clear all;
close all;
n=0:7;
N=8;
x=cos(2*pi*n./N);
h=sin(2*pi*n./N);
disp('first sequence is');
disp(x);
disp('2nd sequence is');
disp(h);

for n=1:N
    Y(n)=0;
    for i=1:N
        j=n-i+1;
        if(j<=0)
            j=N+j;
        end
```

```
        Y(n)=[Y(n)+x(i)*h(j)];
    end
end

disp('convolted sequence without function is');
disp(Y);

R=cconv(x,h,N);
disp('convoluted sequence with function is');
disp(R);
```

```
first sequence is

  Columns 1 through 7

    1.0000    0.7071    0.0000   -0.7071   -1.0000   -0.7071   -0.0000

  Column 8

    0.7071

2nd sequence is

  Columns 1 through 7

         0    0.7071    1.0000    0.7071    0.0000   -0.7071   -1.0000

  Column 8

   -0.7071

convoluted sequence without function is

  Columns 1 through 7

   -0.0000    2.8284    4.0000    2.8284    0.0000   -2.8284   -4.0000

  Column 8

   -2.8284

convoluted sequence with function is

  Columns 1 through 7

   -0.0000    2.8284    4.0000    2.8284    0.0000   -2.8284   -4.0000

  Column 8

   -2.8284
```

Conclusion:

In this experiment, we have implemented linear convolution using the in-built function and also without using the in-built function. Also, we implemented circular convolution through Matlab.

Date : 18/08/21

PRACTICAL NO: 04

Aim:

Write a simulation program to compare DFT, IDFT for the given signal.

X[n] = [1 1 1 1 1 1 0 0 1]

Plot the magnitude and phase response for DFT.

Theory:

The DFT of discrete time signal x[n] is finite duration discrete frequency sequence. DFT obtained by sampling one period of Fourier transform of x[n] at a finite number of frequency points. DFT is defined with a number of samples called N-point DFT. The number of samples N for a finite duration sequence x[n] of length L should be such that N>=L.

TRANSFORM	TIME DOMAIN	FREQUENCY DOMAIN
CTFS	Continuous & Periodic	Discrete & Aperiodic
CTFT	Continuous & Aperiodic	Continuous & Aperiodic
DTFT	Discrete & Aperiodic	Continuous & Periodic
DFT	Discrete & Periodic	Discrete & Periodic

DFT:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} \quad k = 0, \ldots, N-1$$

$$X_k = \sum_{n=0}^{N-1} x_n W_N^{kn} \quad k = 0, 1 \ldots, N-1$$

IDFT:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i2\pi kn/N} \quad n = 0, \ldots, N-1$$

$$x_n = 1/N \sum_{k=0}^{N-1} X_k W_N^{-kn} \quad k = 0, 1 \ldots, N-1$$

4-point DFT:

N= 4

K=0, 1, 2, 3

$$\begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ X[3] \end{bmatrix} = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix}$$

MATLAB Code:

DFT without using in-built function:

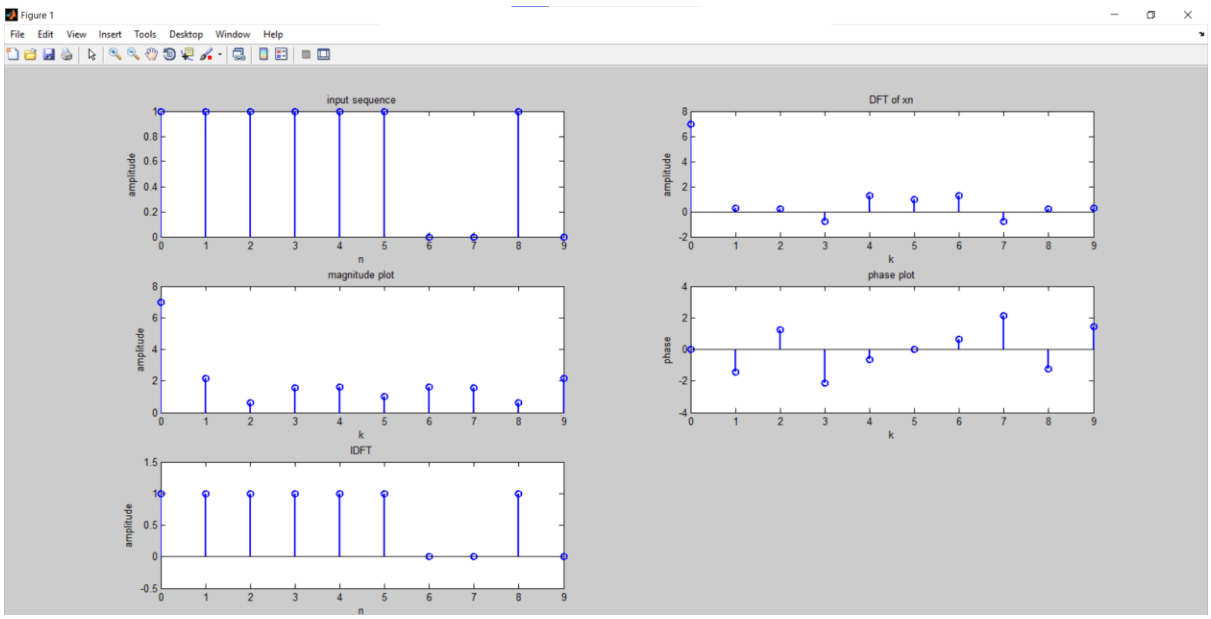
```
clc;

clear all;
close all;xn=input('enter the sequence');
```

```
N=input('enter no of samples');
L=length(xn);
n=0:N-1;
xn=[xn zeros(1,N-L)];

subplot(3,2,1)
stem(n,xn,'linewidth',2);
xlabel('n');
ylabel('amplitude');
title('input sequence');


%DFT
y=zeros(1,N);

for k=0:N-1
    for n=0:N-1;
        y(k+1)=y(k+1)+xn(n+1)*exp((-1i*2*pi*n*k)/N);
    end
end

disp('y is equal to');
disp(y)

k=0:N-1;
subplot(3,2,2)
stem(k,y,'linewidth',2)
xlabel('k')
ylabel('amplitude')
title('DFT of xn')

magnitude=abs(y);
subplot(3,2,3)
stem(k,magnitude,'linewidth',2);
xlabel('k')
ylabel('amplitude')
title('magnitude plot')

phase=angle(y);
subplot(3,2,4)
stem(k,phase,'linewidth',2)
xlabel('k')
ylabel('phase')
title('phase plot')

% IDFT

N=length(y);
m=zeros(1,N);

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

```
        m(n+1)=m(n+1)+((1/N)*(y(k+1)*exp((j*2*pi*k*n)/N)));
    end
end

disp('m is equal to')
disp(m)

n=0:N-1;
subplot(3,2,5)
stem(n,m,'linewidth',2)
xlabel('n')
ylabel('amplitude')
title('IDFT')
```

Output:

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

enter the sequence [1 1 1 1 1 1 0 0 1]
enter no of samples 10
y is equal to
  Columns 1 through 6

   7.0000 + 0.0000i   0.3090 - 2.1266i   0.1910 + 0.5878i  -0.8090 - 1.3143i   1.3090 - 0.9511i   1.0000 + 0.0000i

  Columns 7 through 10

   1.3090 + 0.9511i  -0.8090 + 1.3143i   0.1910 - 0.5878i   0.3090 + 2.1266i

m is equal to
  Columns 1 through 6

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

  Columns 7 through 10

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

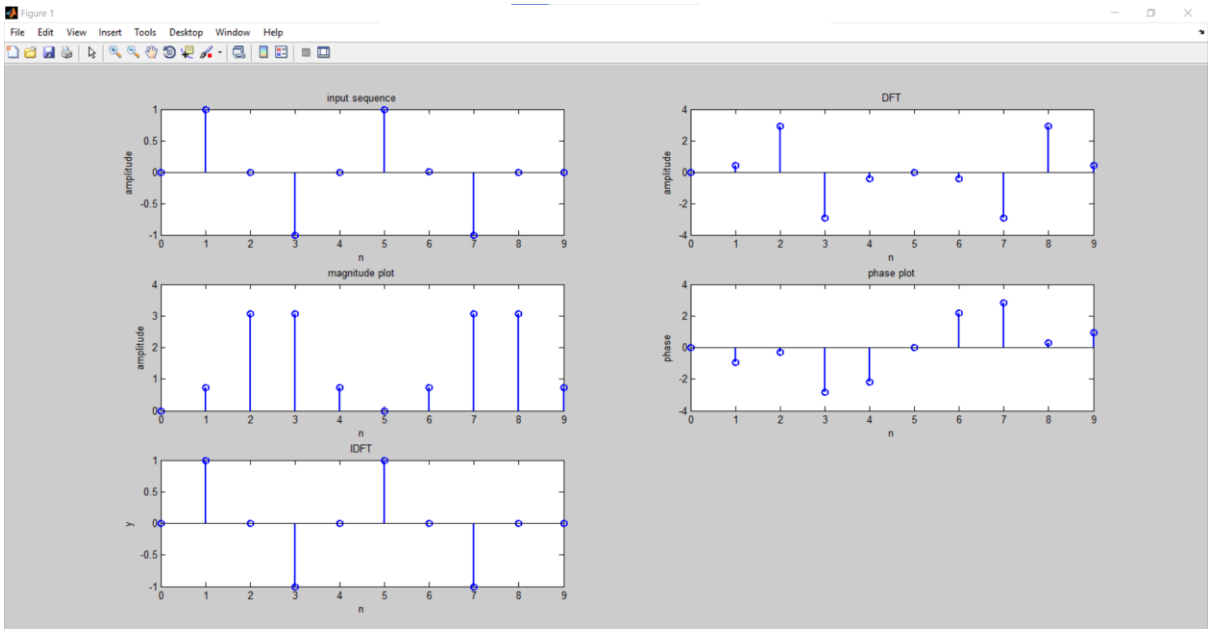


DFT using in-built function:

```
clc;

clear all;
close all;

a= 0:7;
xn= sin(pi*a./2);
N=input('enter no of samples');
L=length(xn);
n=0:N-1;
xn=[xn zeros(1,N-L)];

subplot(3,2,1)
stem(n,xn,'linewidth',2);
xlabel('n');
ylabel('amplitude');
title('input sequence');

% DFT

Xk=fft(xn,N);

subplot(3,2,2)
stem(n,Xk,'linewidth',2);
xlabel('n');
ylabel('amplitude');
title('DFT');

magnitude=abs(Xk);
subplot(3,2,3)
stem(n,magnitude,'linewidth',2);
xlabel('n')
ylabel('amplitude')
title('magnitude plot')

phase=angle(Xk);
subplot(3,2,4)
stem(n,phase,'linewidth',2)
xlabel('n')
ylabel('phase')
title('phase plot')

% IDFT

y=ifft(Xk,N);

subplot(3,2,5)
stem(n,y,'linewidth',2);
xlabel('n')
ylabel('y')
```

```
title('IDFT')
```

Output:



Conclusion:

In this experiment, we have simulated a program to compute DFT & IDFT for a given signal using the in-built function and without using the in-built function.

Date : 25/08/21

PRACTICAL NO: 05

Aim:

Write a MATLAB program to implement 8-point Decimation in time – Fast fourier transform algorithm.

$$X[n] = \cos(\pi n/2) \qquad 0 \le n \le 7$$

Plot magnitude and phase response.

Theory:

Discrete time fourier transform:

$$X[k] = \sum_{n=0}^{N-1} x(n) * e^{\frac{-j2\pi nk}{N}}$$

k = 0 N----- N-1
k = 1 N----- N-1
:
:
k = N N----- N-1

N*N = N^2 Complex multiplier
N(N-1) = Complex adder

Fast Fourier Transform

$$Multiplier = \frac{N}{2} log_2 N$$

$$Adder = N log_2 N$$

Decimation in time algorithm:

$$x_e(n) = x(2n) \quad n = 0, 1, 2, 3 \ldots\ldots \frac{N}{2} - 1$$

$$x_o(n) = x(2n+1) \quad n = 0, 1, 2, 3 \ldots\ldots \frac{N}{2} - 1$$

$$X[k] = \sum_{n=0}^{N-1} x(n) W_N^{nk} \quad k = 0,1,2 \ldots\ldots N-1$$

$$X[k] = \sum_{n=0\ (even)}^{N-1} x(n) W_N^{nk} + \sum_{n=0\ (odd)}^{N-1} x(n) W_N^{nk}$$

$$X[k] = \sum_{n=0\ (even)}^{\frac{N}{2}-1} x(2n) W_N^{2nk} + \sum_{n=0\ (odd)}^{\frac{N}{2}-1} x(2n+1) W_N^{(2n+1)k}$$

$$X[k] = \sum_{n=0\ (even)}^{\frac{N}{2}-1} x(2n)W_N^{2nk} + W_N^k \sum_{n=0\ (odd)}^{\frac{N}{2}-1} x(2n+1)W_N^{2nk}$$

$$X[k] = \sum_{n=0}^{\frac{N}{2}-1} x_e(n)W_N^{2nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x_0(n)W_N^{2nk}$$

$$W_N^2 = (e^{-j2\pi/N})^2 = e^{-j2\pi/N/2} = W_{N/2}$$

$$\boldsymbol{X[k] = \sum_{n=0}^{\frac{N}{2}-1} x_e(n)W_{N/2}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x_0(n)W_{N/2}^{nk}}$$

MATLAB Code:

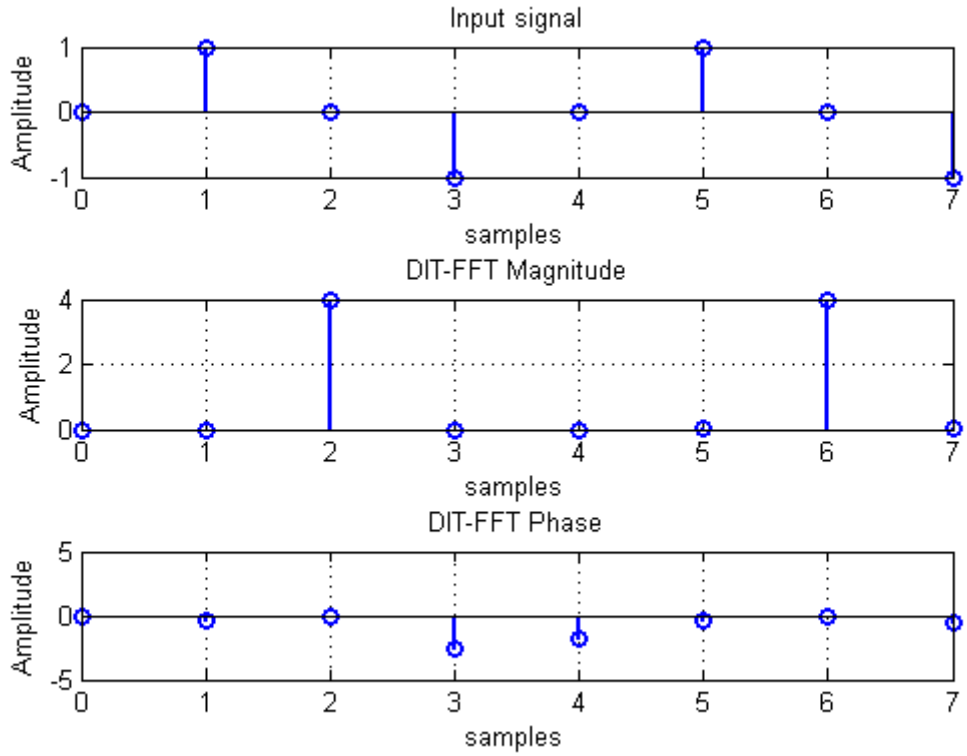
DIT FFT

```
clc;

close all;
clear all;



a= 0:7;
x1= sin(pi*a./2);
l= length(x1);

n= length(x1);
y= zeros(1,l);



xe= zeros(1,n/2);
xo= zeros(1,n/2);



m=1;
 for k=1:2:n
       xo(m)=x1(k);
       m=m+1;
 end



 m=1;
  for k=2:2:n
```

```
      xe(m)=x1(k);
      m=m+1;
  end

for k=1:1:n
   for j=1:1:(n/2)
          y(k)=y(k)+xe(j)*exp(-4*pi*1i*(j-1)*(k-1)/n)+exp(-2*pi*1i*(k-
1)/n)*xo(j)*exp(-4*pi*1i*(j-1)*(k-1)/n);
   end
end

%disp('odd');
%disp(xo);
%disp('even');
%disp(xe);
disp(y);

t=0:n-1;

subplot(3,1,1);
stem(t,x1,'linewidth',2);
ylabel('Amplitude');
xlabel('samples');
title('Input signal');
grid on;

subplot(3,1,2);
stem(t,abs(y),'linewidth',2);
ylabel('Amplitude');
xlabel('samples');
title('DIT-FFT Magnitude');
grid on;

p=angle(y);
subplot(3,1,3);
stem(t,p,'linewidth',2);
ylabel('Amplitude');
xlabel('samples');
title('DIT-FFT Phase');
grid on;
```

```
  Columns 1 through 4

   0.0000 + 0.0000i   0.0000 - 0.0000i   4.0000 + 0.0000i  -0.0000 - 0.0000i

  Columns 5 through 8

  -0.0000 - 0.0000i   0.0000 - 0.0000i   4.0000 + 0.0000i   0.0000 - 0.0000i
```

Output:



CONCLUSION:

In this experiment, we simulated a program to compute DFT & IDFT for a given signal in a decimation in time algorithm using the in-built function and without using the in-built function.

Date : 01/08/21

PRACTICAL NO : 06

Aim:

Write a simulation program to implement 8 – point DIF – FFT algorithm and verify the same using in – built simulation command.

$$X[n] = \sin(\pi n/2) \qquad 0 \le n \le 7$$

Plot the magnitude and phase response.

Theory:

Fast fourier transform:

The FFT may be defined as an algorithm (or a method) for computing the DFT efficiency (with reduced number of calculations).

The computational efficiency is achieved by adopting a divide and conquer approach. This approach is based on the decomposition of an N-point DFT into successively smaller DFT's and then combining them to give the total transform. Based on this basic approach, a family of computational algorithms were developed and they are collectively known as FFT algorithms.

The FFT algorithm exploits the above two symmetry properties and so is an efficient algorithm for DFT computation.

Symmetry property $W_N^{k+\frac{N}{2}} = -W_N^k$

Periodicity property $W_N^{k+N} = W_N^k$

Decimation in frequency – FFT :

In decimation in frequency algorithms, the frequency domain sequence X(k) is decimated.

In this algorithm, the N-point time domain sequence is converted to two numbers of N/2-point sequences.

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

$$= \sum_{n=0}^{\frac{N}{2}-1} x[n]W_N^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x[n+\frac{N}{2}]W_N^{nk}\, W_N^{\frac{N}{2}k}$$

$$W_N^{\frac{Nk}{2}} = \left(e^{\frac{-j2\pi}{N}}\right)^{\frac{Nk}{2}} = e^{\frac{-j2\pi Nk}{N\cdot 2}} = e^{-j\pi k} = \left(e^{-j\pi}\right)^k = (-1)^k$$

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} \left\{ x[n] + (-1)^k x\left[n + \frac{N}{2}\right] \right\} W_N^{nk}, k = 0,1, \dots, N-1$$

Now Decimating X(K) into even and odd sequence

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} \left\{ x[n] + x\left[n + \frac{N}{2}\right] \right\} W_{\frac{N}{2}}^{nk}, k = 0,1, \dots, \frac{N}{2} - 1 \dots \dots eqn.1$$

$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} \left\{ x[n] - x\left[n + \frac{N}{2}\right] \right\} W_N^n W_{\frac{N}{2}}^{nk}, k = 0,1, \dots, \frac{N}{2} - 1 \dots \dots eqn.2$$

$$let\ g[n] = x[n] + x\left[n + \frac{N}{2}\right], 0 \le n \le \frac{N}{2} - 1 \dots \dots eqn.3$$

$$and, h[n] = \left\{ x[n] - x\left[n + \frac{N}{2}\right] \right\} W_N^n, 0 \le n \le \frac{N}{2} - 1 \dots \dots eqn.4$$

$g[0] = x[0] + x[4]$

$g[1] = x[1] + x[5]$

$g[2] = x[2] + x[6]$

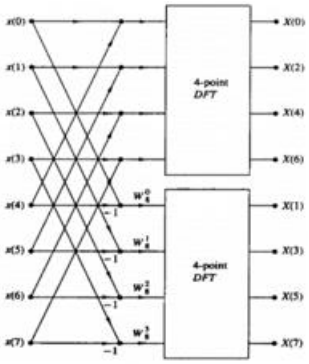
$g[3] = x[3] + x[7]$

$h[0] = \{x[0] - x[4]\}W_8^0$

$h[1] = \{x[1] - x[5]\}W_8^1$

$h[2] = \{x[2] - x[6]\}W_8^2$

$h[3] = \{x[3] - x[7]\}W_8^3$



Sr No	DIT FFT	DIF FFT
1	DITFFT algorithms are based upon decomposition of the input sequence into smaller and smaller sub sequences.	DIFFFT algorithms are based upon decomposition of the output sequence into smaller and smaller sub sequences.
2	In this input sequence x(n) is splitted into even and odd numbered samples	In this output sequence X(k) is considered to be splitted into even and odd numbered samples
3	Splitting operation is done on time domain sequence.	Splitting operation is done on frequency domain sequence.
4	In DIT FFT input sequence is in bit reversed order while the output sequence is in natural order.	In DIFFFT, input sequence is in natural order. And DFT should be read in bit reversed order.

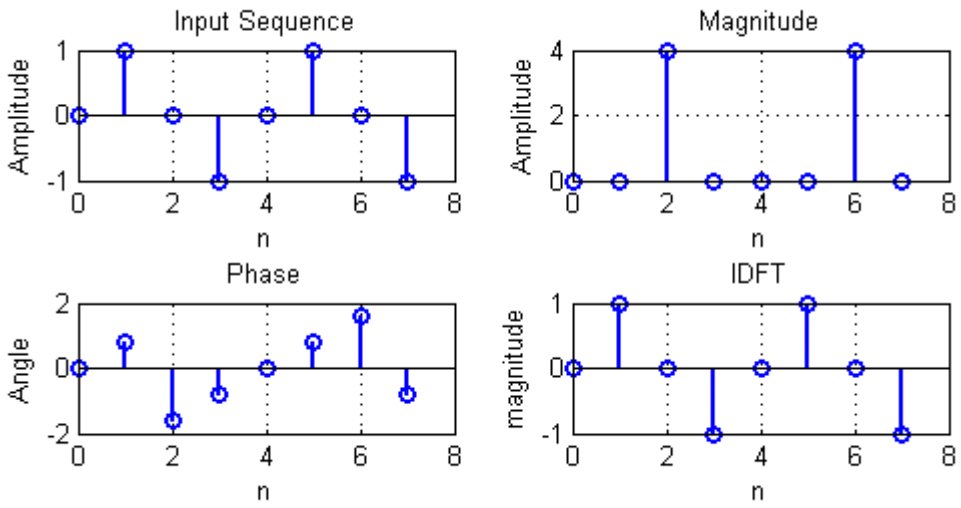
MATLAB Code:

DIF-FFT WITHOUT USING IN-BUILT FUNCTION

```
clc;

clear all;
close all;

a = 0:7;
x = sin(pi*a/2);

N = length(x);
l = nextpow2(N);

x = [x, zeros(1, (2^l)-N)];

N = length(x);

t=0:N-1;
subplot(3, 2, 1);
stem(t, x, 'linewidth', 2);
ylabel('Amplitude');
xlabel('n');
title('Input Sequence');
grid on;


for j = l:-1:1
    L = 2^j;
    for n = 1:L:N-L+1
        for k = 0:L/2 - 1
            w = exp(-1i*2*pi*k/L);  %Twiddle Factor
            A = x(n+k);
            B = x(n+k+L/2);
            x(n+k) = A+B;
            x(n+k+L/2) = (A-B)*w;
        end
    end
end

y = bitrevorder(x);

subplot(3, 2, 2);
stem(t, abs(y), 'linewidth', 2);
ylabel('Amplitude');
xlabel('n');
title('Magnitude');
grid on;

subplot(3, 2, 3);
stem(t, angle(y), 'linewidth', 2);
ylabel('Angle');
```

```
xlabel('n');
title('Phase');
grid on;


% IDFT

for j = 1:-1:1
    L = 2^(j);
    for n = 1:L:N-L+1
        for k = 0:L/2 - 1
            w = exp(1i*2*pi*k/L);  %Twiddle Factor
            C = y(n+k);
            D = y(n+k+L/2);
            y(n+k) = C+D;
            y(n+k+L/2) = (C-D)*w;
        end
    end
end

y=y/N;
y=bitrevorder(y);
n=0:N-1;

subplot(3,2,4)
stem(n,y, 'linewidth', 2);
xlabel('n');
ylabel('magnitude');
title('IDFT')
grid on;
```

Output:

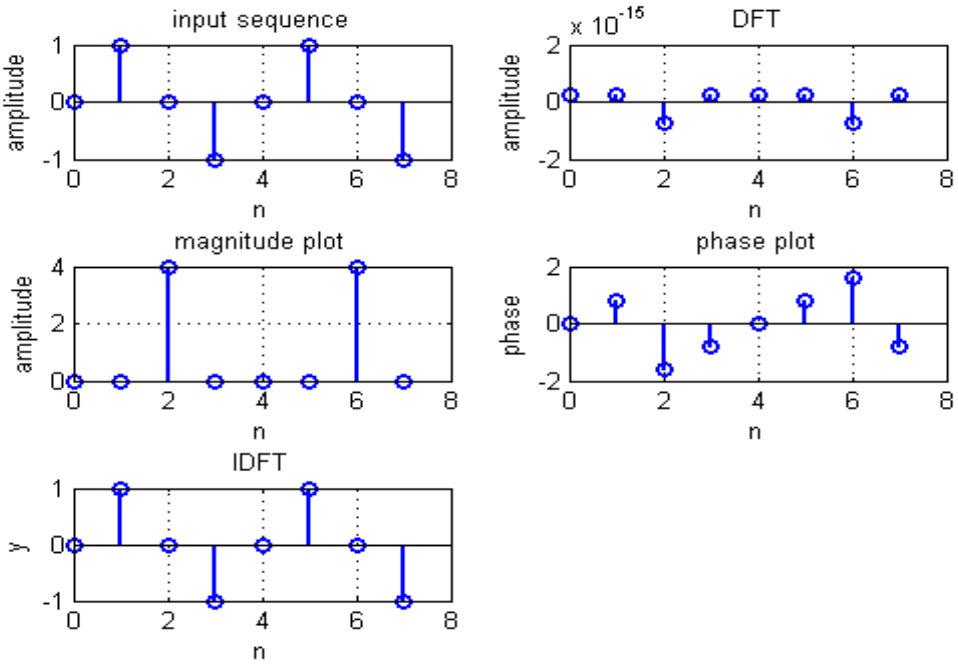


DIF-FFT using inbuilt function

```
clc;

clear all;
close all;

a=0:7;
xn=sin(pi*a./2);
N=8;
L=length(xn);
n=0:N-1;
xn=[xn zeros(1,N-L)];

subplot(3,2,1)
stem(n,xn,'linewidth',2);
xlabel('n');
ylabel('amplitude');
title('input sequence');
grid;

% DFT

Xk=fft(xn,N);

subplot(3,2,2)
stem(n,Xk,'linewidth',2);
xlabel('n');
ylabel('amplitude');
title('DFT');
grid;

magnitude=abs(Xk);
subplot(3,2,3)
stem(n,magnitude,'linewidth',2);
xlabel('n')
ylabel('amplitude')
title('magnitude plot')
grid;

phase=angle(Xk);
subplot(3,2,4)
stem(n,phase,'linewidth',2)
xlabel('n')
ylabel('phase')
title('phase plot')
grid;

% IDFT

y=ifft(Xk,N);

subplot(3,2,5)
```

```
stem(n,y,'linewidth',2);
xlabel('n')
ylabel('y')
title('IDFT')
grid;
```

Output:



CONCLUSION:

In this experiment, we have simulated a program to compute DFT & IDFT for a given signal in decimation in frequency algorithm using the in-built function and without using the in-built function.