INDEX

Practical 1: Sampling

Date: 30-07-2021

AIM: To verify 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.

$$fs = 2f_m$$

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.

$$fs < 2f_m$$

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

ALGORITHM:

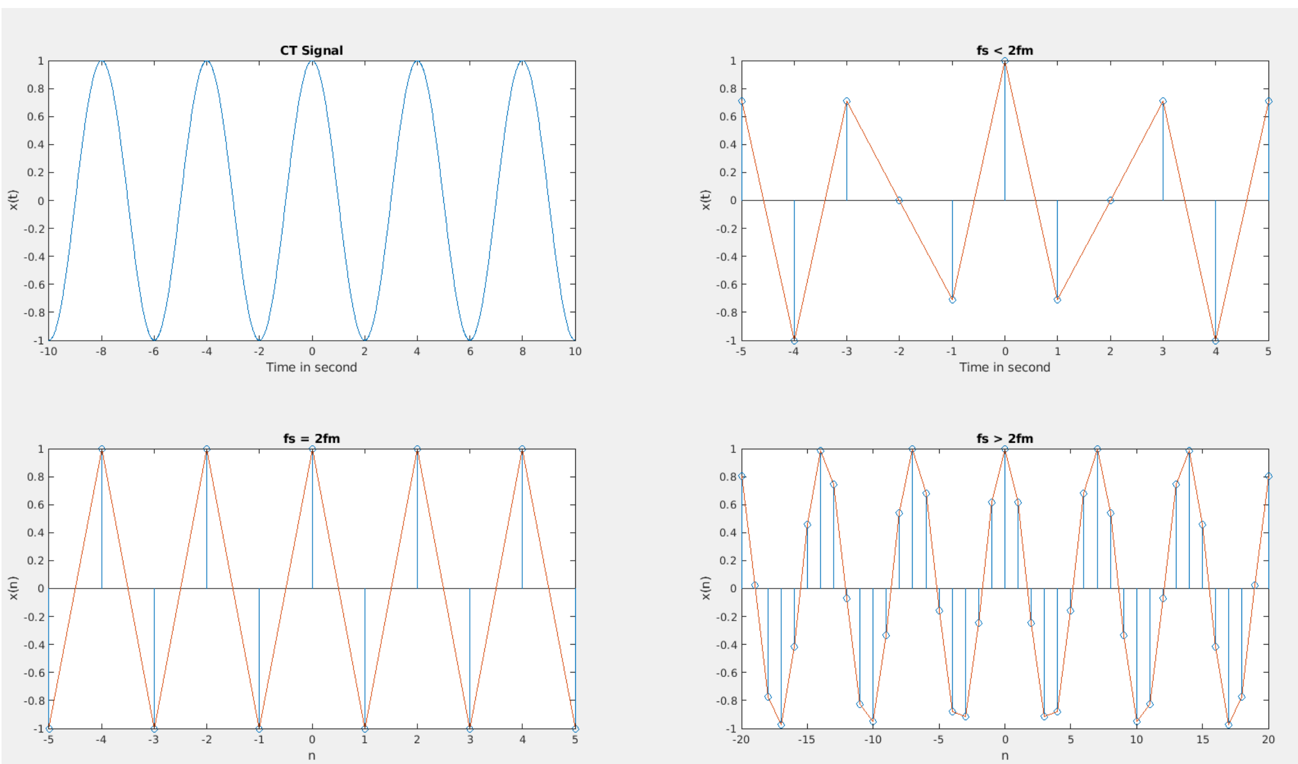
1. Define analog frequency fa for the input sinusoidal signal and also choose sampling frequency much greater than fa. Let k be the scaling factor
2. Define the time period n as per the sampling frequency and plot the sampled signal using this time period.
3. Define the reconstruction time period and 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
t = -10:0.01:10;
T = 4;
fm = 1/T;
n1 = -5:1:5;
n2 = -5:1:5;
n3 = -20:1:20;

fc = input('Input Frequency > ');
Ac = input('Input Amplitude > ');
fs1 = 1.6*fc;
fs2 = 2*fc;
fs3 = 6.9*fc;
x = Ac*cos(2*pi*fc*t);
x1 = Ac*cos(2*pi*n1*fc/fs1);
x2 = Ac*cos(2*pi*n2*fc/fs2);
x3 = Ac*cos(2*pi*n3*fc/fs3);

subplot(2, 2, 1);
plot(t, x);
xlabel('Time in second');
ylabel('x(t)');
title('CT Signal');

subplot(2, 2, 2);
stem(n1, x1);
hold on
plot(n1, x1);
xlabel('Time in second');
ylabel('x(t)');
title('fs < 2fm');
```

```
subplot(2, 2, 3);
stem(n2, x2);
hold on
subplot(2, 2, 3);
plot(n2, x2);
xlabel('n');
ylabel('x(n)');
title('fs = 2fm');

subplot(2, 2, 4);
stem(n3, x3);
hold on
subplot(2, 2, 4);
plot(n3, x3);
xlabel('n');
ylabel('x(n)');
title('fs > 2fm');
```

Output:



Conclusion:

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

Practical 2: Time shifting and Time scaling

Date : 06-08-2021

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:

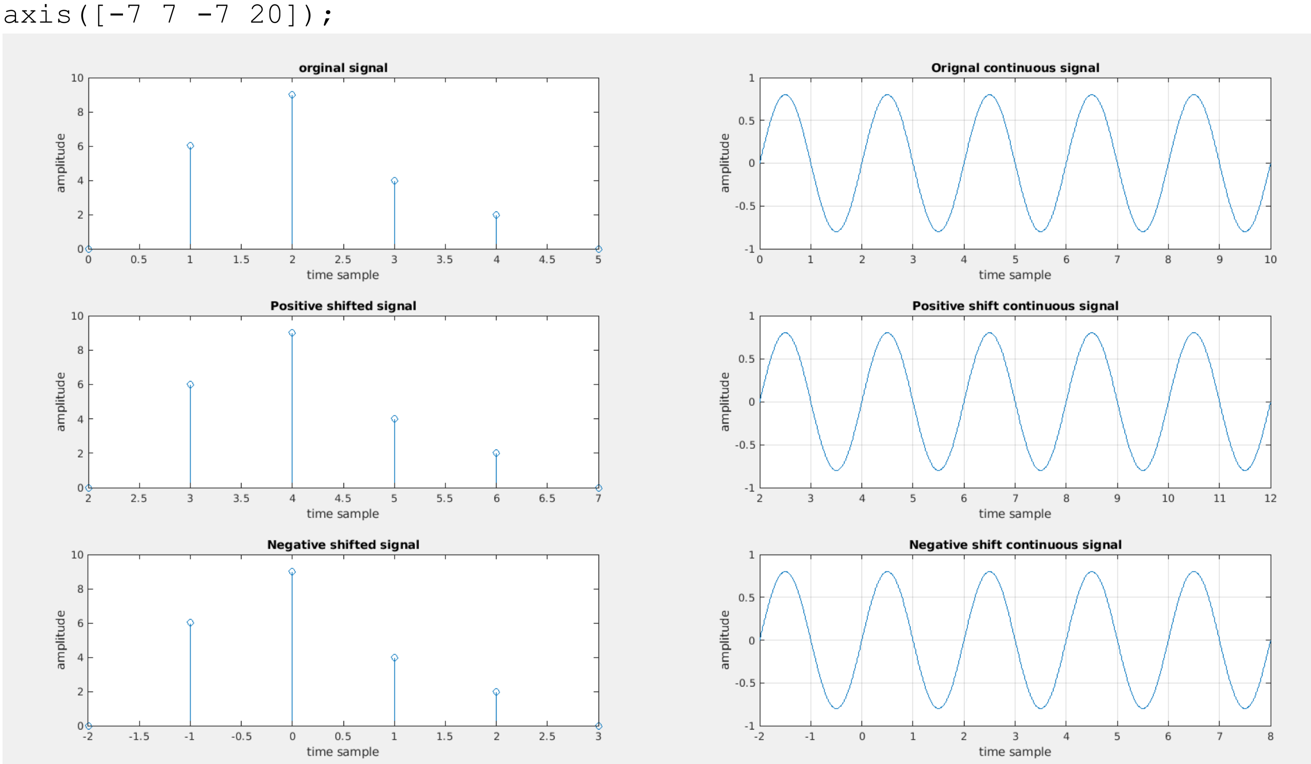
Signal addition—Two signals $x(t)$ and $y(t)$ are added to obtain their sum $z(t)$.

Matlab Code:

- Time shifting of continuous and discrete signal

```
clc;
clear all;
close all;

n1 = -2:1:6;
x1 = [4 2 0 6 9 4 2 0 1];

n2 = -5:1:3;
x2 = [-4 -2 -1 -4 -2 -0 6 9 1];

n = min(min(n1), min(n2)) : max(max(n1), max(n2));
y1 = zeros(1, length(n));
y2 = zeros(1, length(n));
```

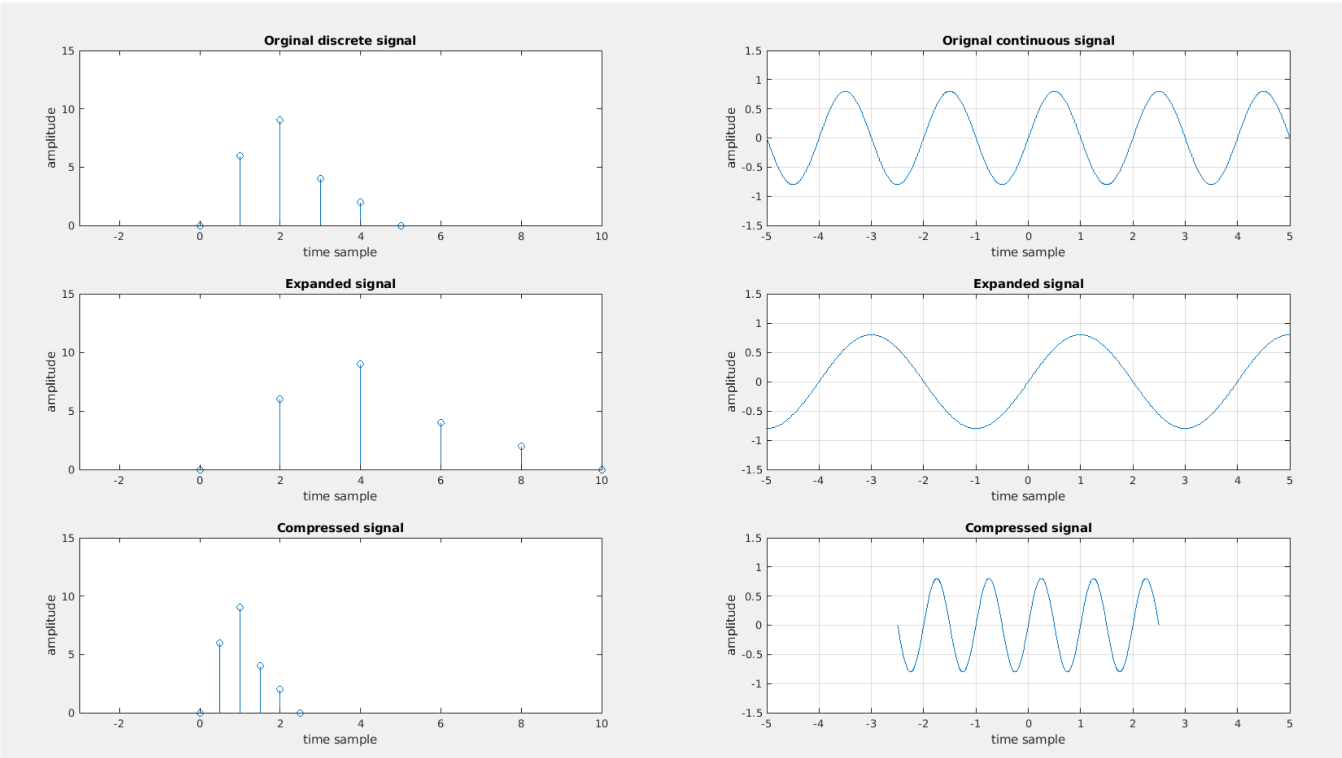
```
y1((n >= min(n1))&(n <= max(n1))) = x1();
y2((n >= min(n2))&(n <= max(n2))) = x2();

x = y1 + y2;

subplot(3, 1, 1);
stem(n1, x1);
title('Signal 1');
xlabel('Time');
ylabel('Amplitude');
axis([-7 7 -7 10]);

subplot(3, 1, 2);
stem(n2, x2);
title('Signal 2');
xlabel('Time');
ylabel('Amplitude');
axis([-7 7 -7 10]);

subplot(3, 1, 3);
stem(n, x);
title('Signal 1 + signal 2 (Zero padding sum)')
xlabel('Time');
ylabel('Amplitude');
axis([-7 7 -7 20]);
```



- Time scaling for discrete and continous time signal

```
clc;
clear all;
close all;
n =  0:5 ;
x = [0 6 9 4 2 0];
subplot(3, 2, 1);
stem(n, x);
xlabel('time sample');
ylabel('amplitude');
title('Orginal discrete signal');
axis([-3 10 0 15]);

m = n.*2;
subplot(3, 2, 3);
stem(m, x);
xlabel('time sample');
ylabel('amplitude');
title('Expanded signal');
axis([-3 10 0 15]);

m = n/2;
subplot(3, 2, 5);
stem(n/2, x);
xlabel('time sample');
ylabel('amplitude');
title('Compressed signal');
axis([-3 10 0 15]);

t = -5:0.01:5;
y = 0.8*sin(2*pi*t/2);
subplot(3, 2, 2);
plot(t, y);
xlabel('time sample');
ylabel('amplitude');
title('Orignal continuous signal');
grid on;
axis([-5 5 -1.5 1.5])

t1 = t.*2;
subplot(3, 2, 4);
plot(t1, y);
xlabel('time sample');
ylabel('amplitude');
title('Expanded signal');
grid on;
axis([-5 5 -1.5 1.5])
```

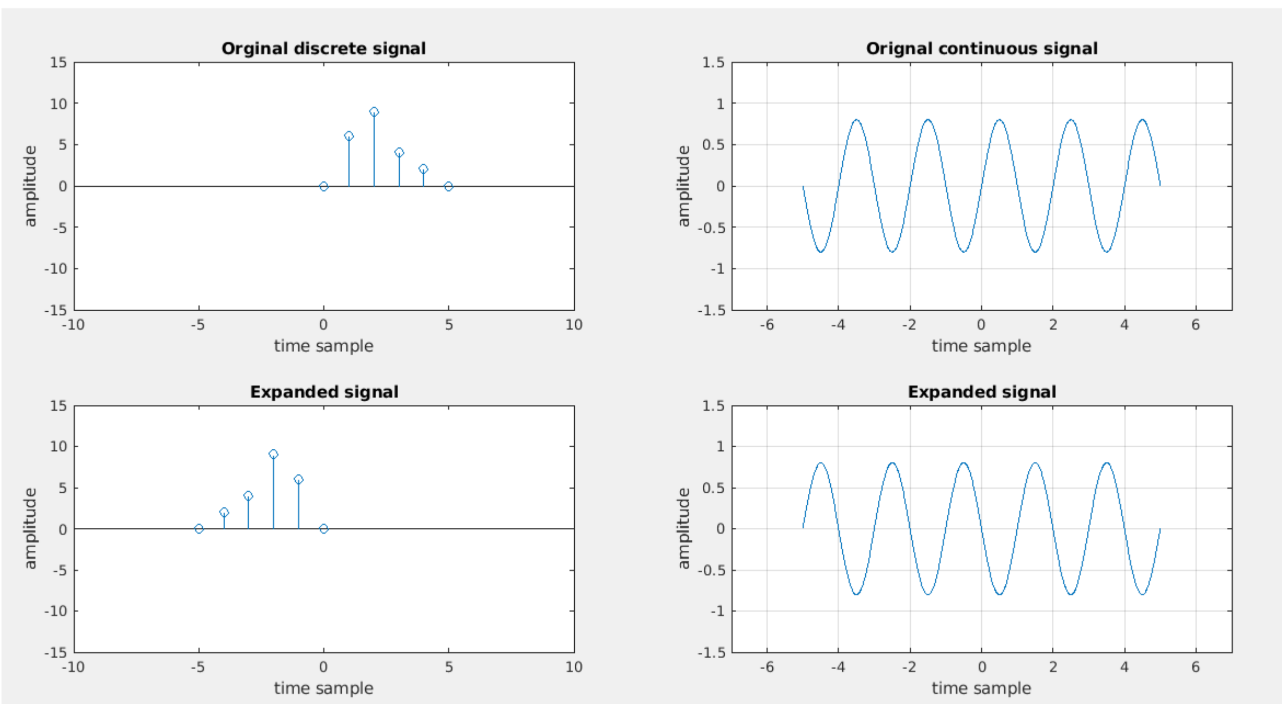
```
t1 = t/2;
subplot(3, 2, 6);
plot(t1, y);
xlabel('time sample');
ylabel('amplitude');
title('Compressed signal');
grid on;
axis([-5 5 -1.5 1.5])
```



- **Time reversal for discrete and continuos time signal**

```
clc;
clear all;
close all;
n =  0:5 ;
x = [0 6 9 4 2 0];
subplot(2, 2, 1);
stem(n, x);
xlabel('time sample');
ylabel('amplitude');
title('Orginal discrete signal');
axis([-10 10 -15 15]);

subplot(2, 2, 3);
stem(-n, x);
xlabel('time sample');
```

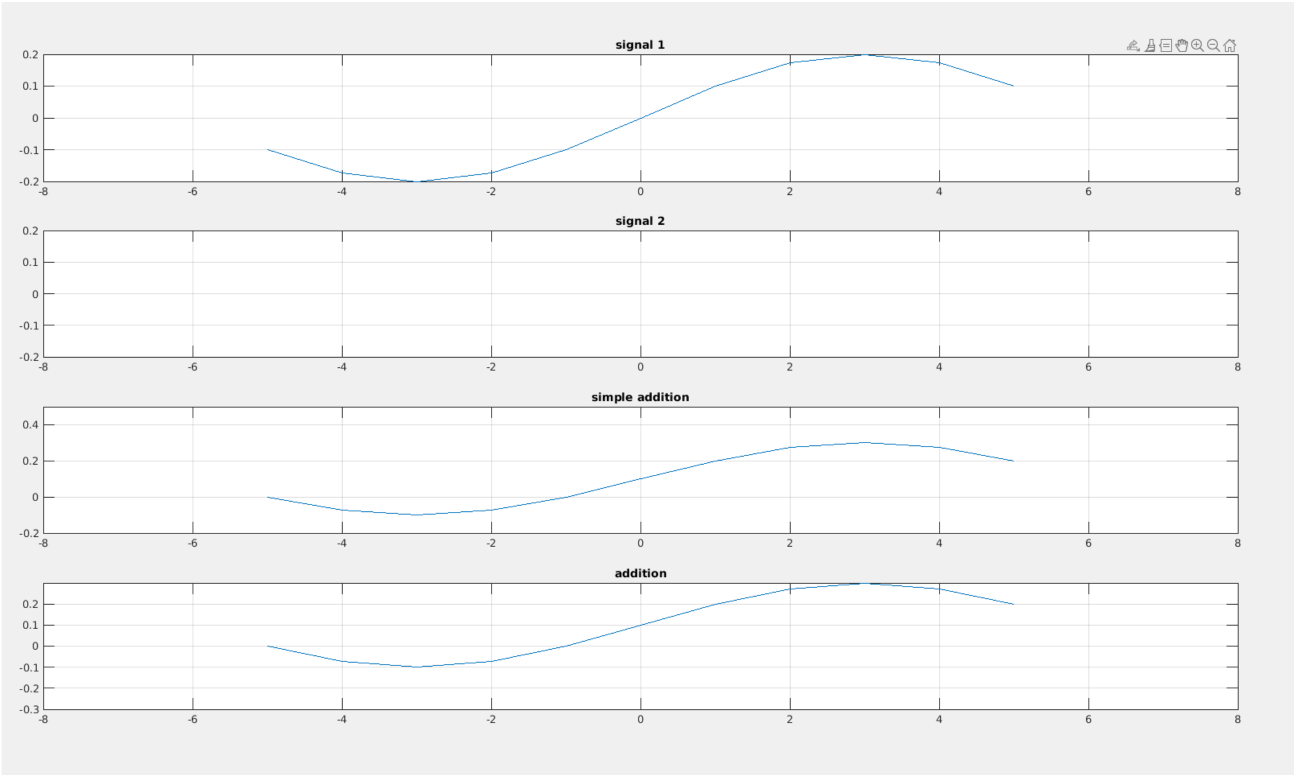
```
ylabel('amplitude');
title('Expanded signal');
axis([-10 10 -15 15]);

t = -5:0.01:5;
y = 0.8*sin(2*pi*t/2);
subplot(2, 2, 2);
plot(t, y);
xlabel('time sample');
ylabel('amplitude');
title('Orignal continuous signal');
grid on;
axis([-7 7 -1.5 1.5])

t1 = t.*2;
subplot(2, 2, 4);
plot(-t, y);
xlabel('time sample');
ylabel('amplitude');
title('Expanded signal');
grid on;
axis([-7 7 -1.5 1.5])
```



- **Addition of two continuous signals**

```
clc;
clear all;
close all;
```

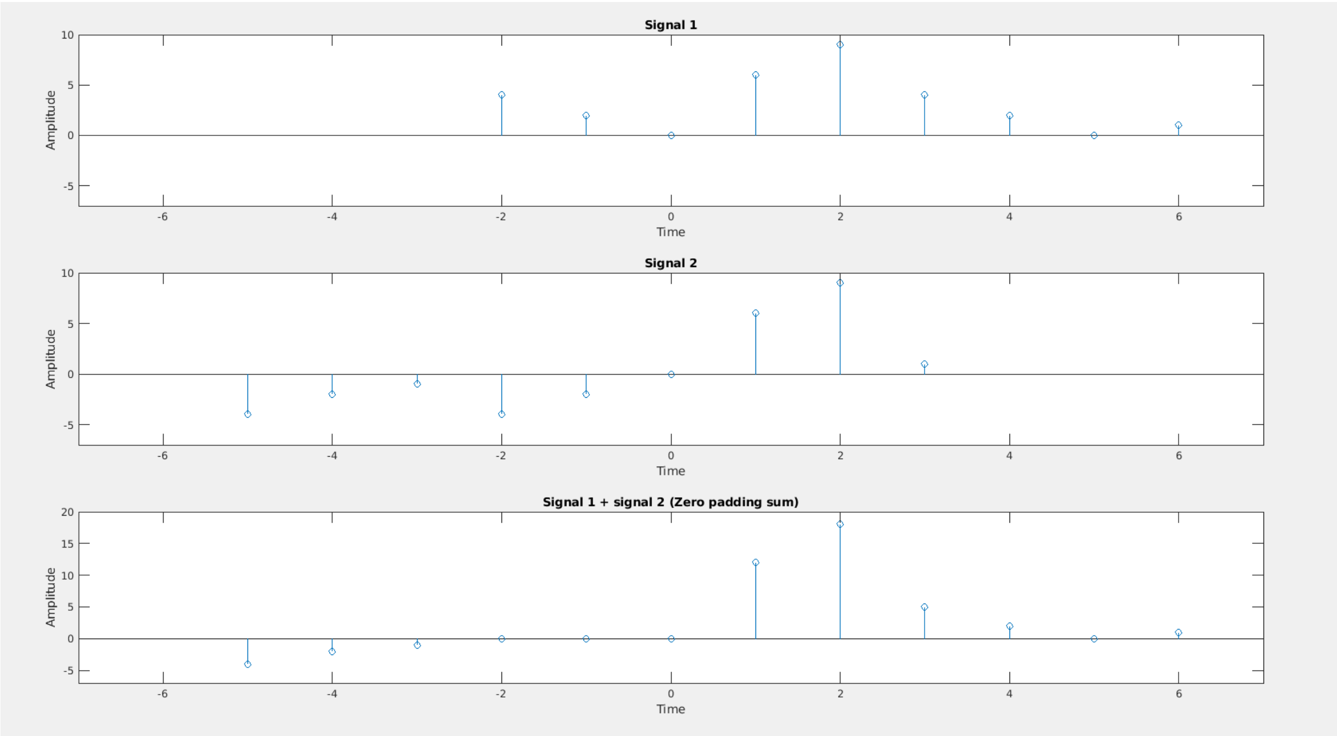
```
t1=-5:1:5;
x1=0.2*sin((pi*t1)/6);

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**

```
clc;
clear all;
close all;

n1 = -2:1:6;
x1 = [4 2 0 6 9 4 2 0 1];


n2 = -5:1:3;
x2 = [-4 -2 -1 -4 -2 -0 6 9 1];

n = min(min(n1), min(n2)) : max(max(n1), max(n2));
y1 = zeros(1, length(n));
y2 = zeros(1, length(n));

y1((n >= min(n1))&(n <= max(n1))) = x1();
y2((n >= min(n2))&(n <= max(n2))) = x2();

x = y1 + y2;

subplot(3, 1, 1);
stem(n1, x1);
title('Signal 1');
xlabel('Time');
ylabel('Amplitude');
```

```
axis([-7 7 -7 10]);

subplot(3, 1, 2);
stem(n2, x2);
title('Signal 2');
xlabel('Time');
ylabel('Amplitude');
axis([-7 7 -7 10]);

subplot(3, 1, 3);
stem(n, x);
title('Signal 1 + signal 2 (Zero padding sum)')
xlabel('Time');
ylabel('Amplitude');
axis([-7 7 -7 20]);
```



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 y-axis.
4. Addition of continuous and discrete signals is implemented.

Practical 3: Convolution

Date : 13-08-2021

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.

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

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

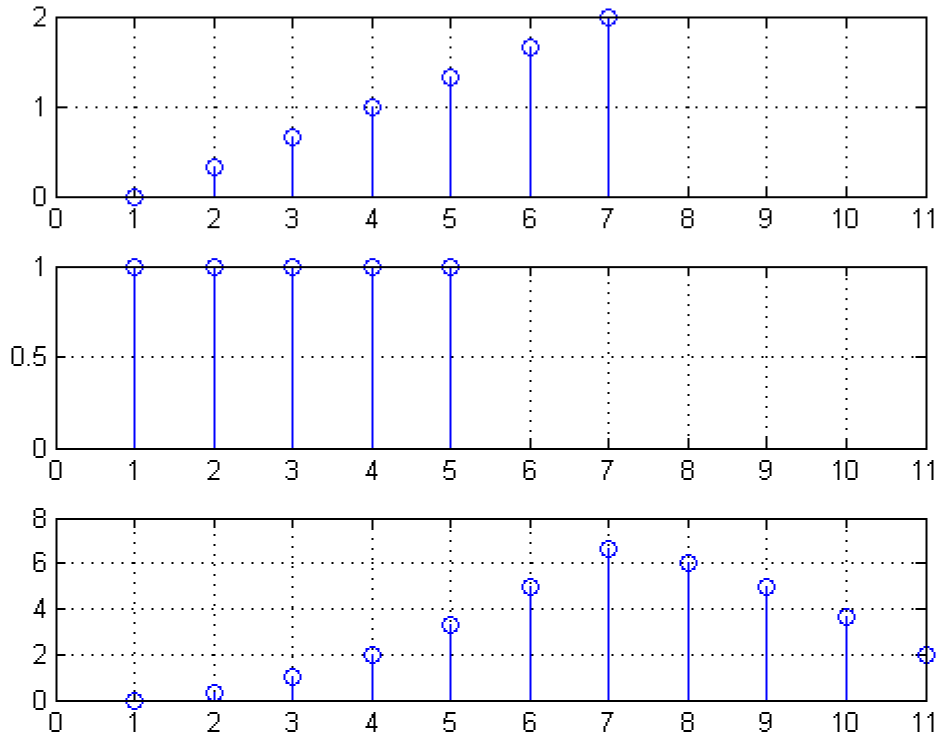
Matlab Code:

- **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
```

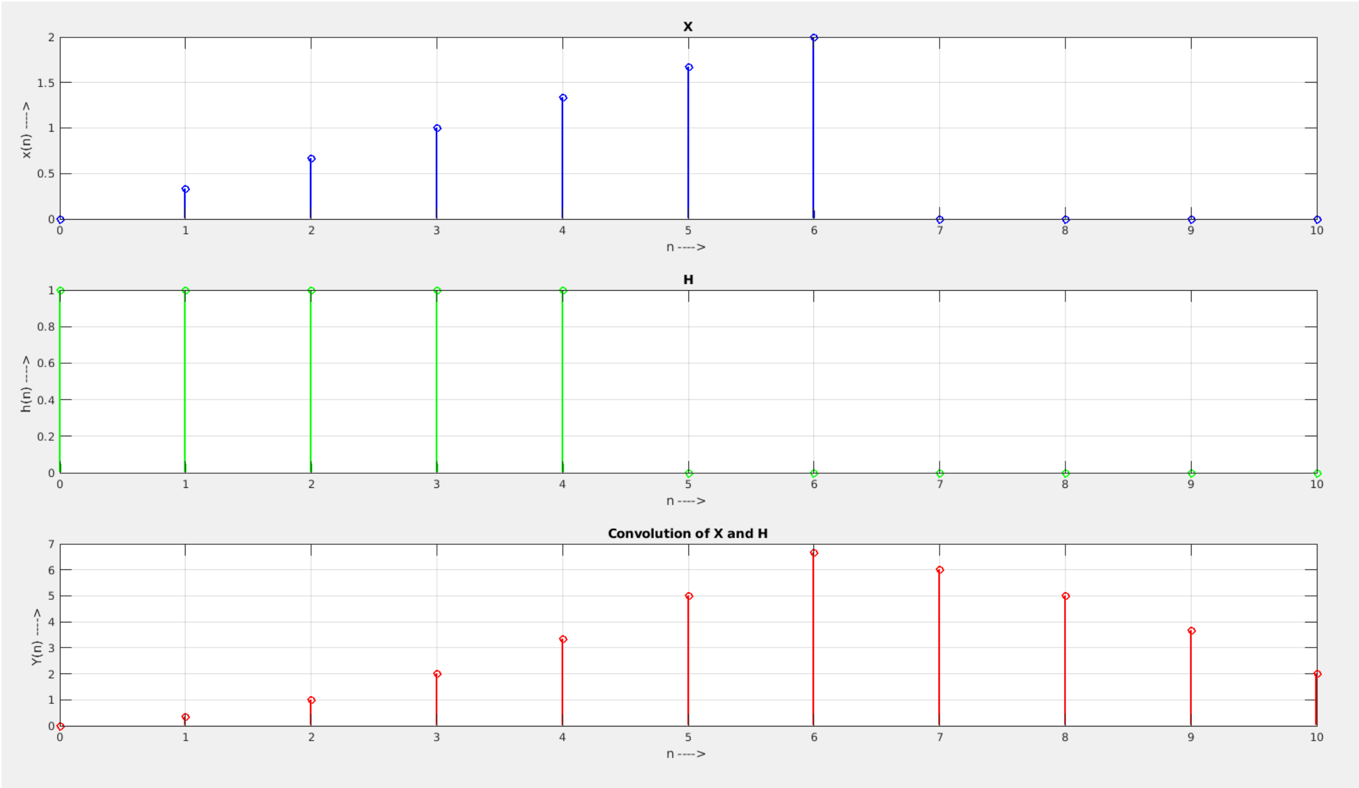


- Linear convolution without using In-Built function:

```
clc;
clear all;
close all;

a = 0:6;
x = (a./3);

b = -2:2;
h = ones(1, length(b));

disp('First sequence is');
disp(x);

disp('Second sequence is');
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
    for k = 1:n
        y(n) = y(n) + x(k)*h(n-k+1);
    end
end

disp('Convolution without inbuild function is');
disp(y);

ny = 0:N-1;

subplot(3, 1, 1);
stem(ny, x, 'b', 'LineWidth', 1.5);
xlabel('n ----> ');
ylabel('x(n) ----> ');
title('X');
grid on;

subplot(3, 1, 2);
stem(ny, h, 'g', 'LineWidth', 1.5);
xlabel('n ----> ');
ylabel('h(n) ----> ');
title('H');
grid on;

subplot(3, 1, 3);
stem(ny, y, 'r', 'LineWidth', 1.5);
xlabel('n ----> ');
ylabel('Y(n) ----> ');
title('Convolution of X and H');
grid on;
```

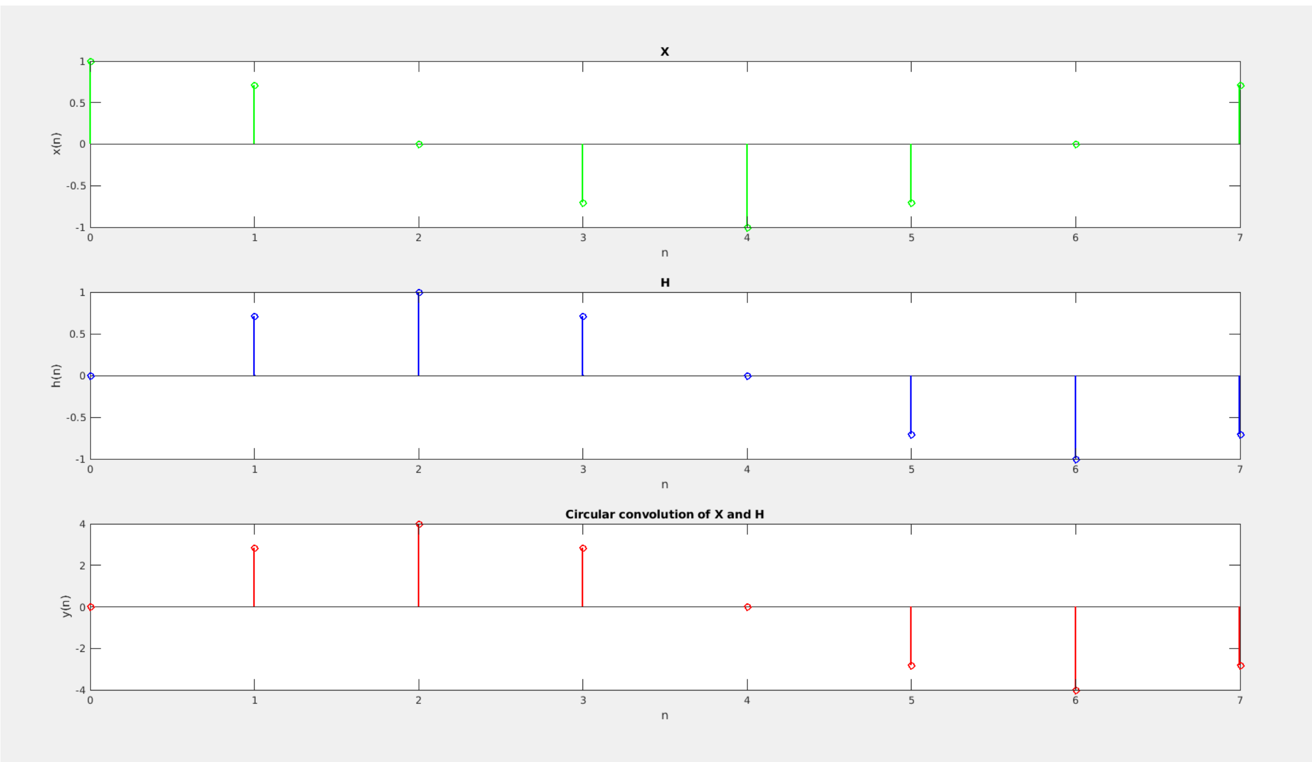
```
First sequence is
        0    0.3333    0.6667    1.0000    1.3333    1.6667    2.0000

Second sequence is
     1     1     1     1     1

Convolution without inbuild function is
        0    0.3333    1.0000    2.0000    3.3333    5.0000    6.6667    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("Second sequence is : ");
disp(h);

y = zeros(1, N);

for n = 1:N
    for m = 1:N
        z = mod(n-m, N);
        y(n) = y(n) + x(m)*h(z+1);
    end
end

disp('Convolution without inbuild function : ')
```

```
disp(y)

ny = 0:N-1;
subplot(3, 1, 1);
stem(ny, x, 'g', 'LineWidth', 1.5);
xlabel('n'); ylabel('x(n)'); title('X');

subplot(3, 1, 2);
stem(ny, h, 'b', 'LineWidth', 1.5);
xlabel('n'); ylabel('h(n)'); title('H');

subplot(3, 1, 3);
stem(ny, y, 'r', 'LineWidth', 1.5);
xlabel('n'); ylabel('y(n)'); title('Circular convolution of X and H');
```




```
First sequence is :
    1.0000    0.7071    0.0000   -0.7071   -1.0000   -0.7071   -0.0000    0.7071

Second sequence is :
         0    0.7071    1.0000    0.7071    0.0000   -0.7071   -1.0000   -0.7071

Convolution without inbuild function :
   -0.0000    2.8284    4.0000    2.8284    0.0000   -2.8284   -4.0000   -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.

Practical 4: DFT

Date: 27-08-2021

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 number of samples called as 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:

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

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

IDFT:

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

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

4-point DFT:

N= 4

K=0, 1, 2, 3

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

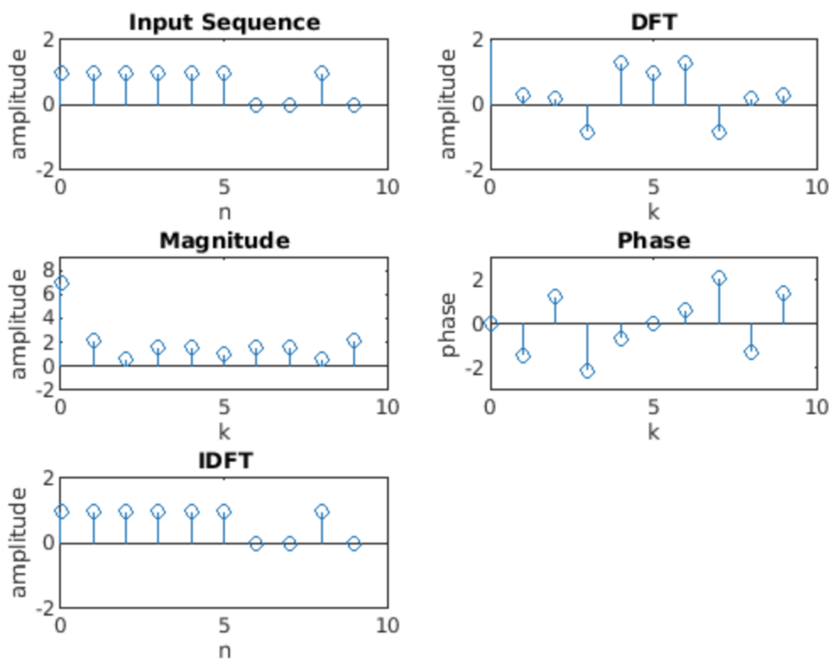
Matlab Code:

1. Without inbuilt:

```
clear all;
close all;

x=input('enter sequence: ');    %Input signal
N=input('enter N= ');      %Input N
L=length(x);      %Length of input signal
```

```
n=0:N-1;
x=[x zeros(1,N-L)];

subplot(3,2,1)
stem(n,x)        %Discrete plot of input signa;
axis([0 10 -2 2])     %Adjust axis
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)+x(n+1)*exp((-j*2*pi*k*n)/N);  %DFT equation
    end
end

%disp(y);
k=0:N-1;
subplot(3,2,2);
stem(k,y)        %plot dtf
axis([0 10 -2 2])
xlabel('k')
ylabel('amplitude')
title('DFT')

magnitude=abs(y);    %magnitudfe response
subplot(3,2,3)
stem(k,magnitude)    %plot magnitude
axis([0 10 -2 9])
xlabel('k')
ylabel('amplitude')
title('Magnitude')

phase=angle(y);  %phase response
subplot(3,2,4);
stem(k,phase)    %plot phase
axis([0 10 -3 3])
xlabel('k')
ylabel('phase')
title('Phase')

N=length(y);
%IDFT
m=zeros(1,N);
for n=0:N-1
    for k=0:N-1
```

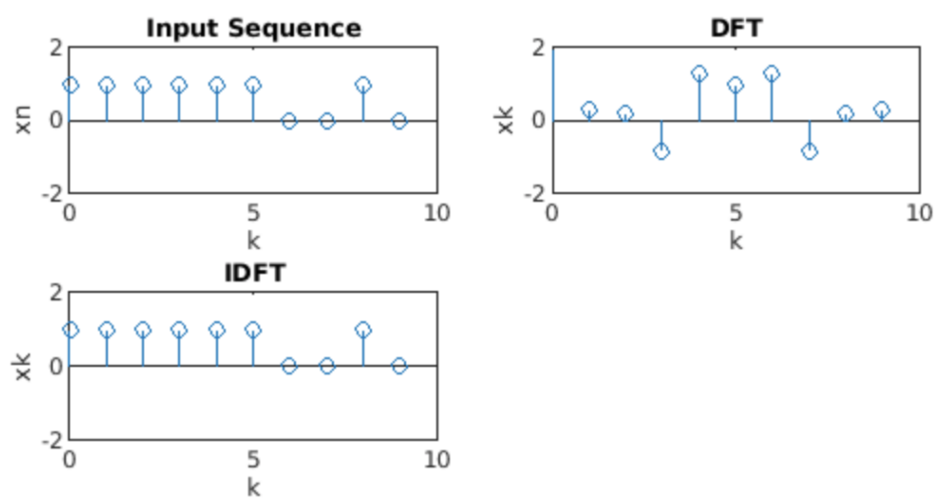
```
            m(n+1)=m(n+1)+((1/N)*(y(k+1)*exp((j*2*pi*k*n)/N))); %IDFT equation
        end
    end

    % disp(m)
    n=0:N-1;
    subplot(3,2,5)
    stem(n,m)     %plot idft
    axis([0 10 -2 2])
    xlabel('n')
    ylabel('amplitude')
    title('IDFT')
```



2. With Inbuilt function:

```
clear all;
close all;

% x=input('enter sequence: ');   %Input signal
% N=input('enter N= ');      %Input N
xn=[1 1 1 1 1 1 0 0 1];
N=10;
k=0:N-1;
L=length(xn);      %Length of input signal
xn=[xn zeros(1,N-L)];

subplot(3,2,1)
stem(k,xn)
```

```
axis([0 10 -2 2])
xlabel('k')
ylabel('xn')
title('Input Sequence')

%DFT
Xk=fft(xn,N);      %DFT inbiult function
subplot(3,2,2)
stem(k,Xk)      %plot dft
axis([0 10 -2 2])   %adjust axis
xlabel('k')
ylabel('xk')
title('DFT')

%IDFT
Id=ifft(Xk,N);   %IDFT inbuilt function
subplot(3,2,3)
stem(k,Id);      %plot idft
axis([0 10 -2 2])
xlabel('k')
ylabel('xk')
title('IDFT')
```



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 in-built function.

Practical 5

Date:03-09-2021

Aim: To write a simulation code for N-Point DIT-FFT and DIT-IFFT and verify the result using inbuilt IFFT and FFT command

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:

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

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

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

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

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

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

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

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

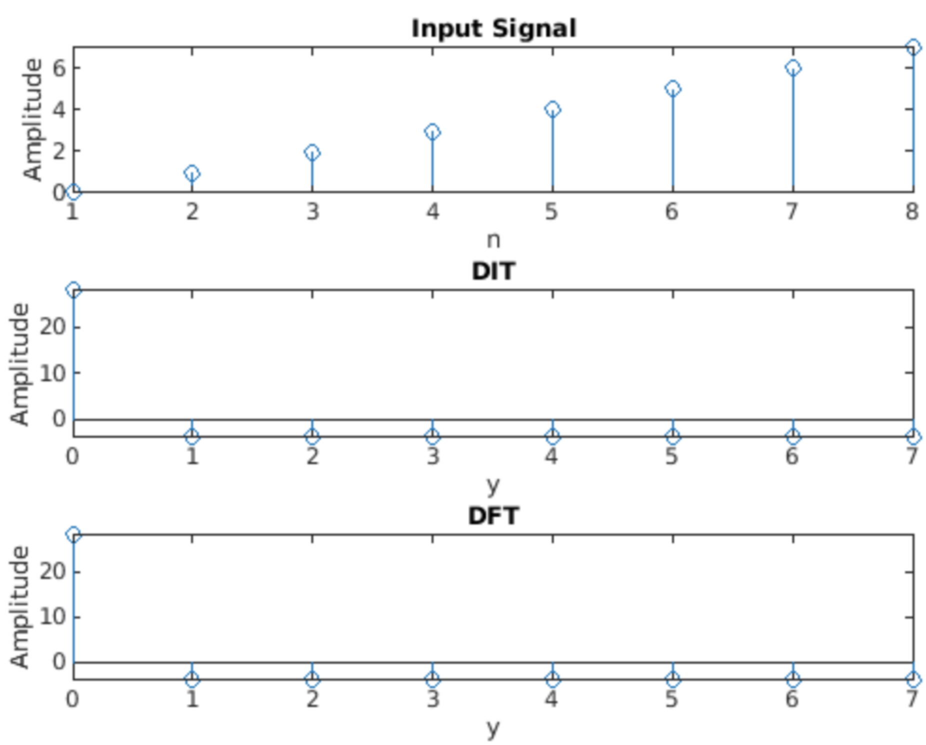
$$\boxed{X[k] = \sum_{n=0}^{\frac{N}{2}-1} x_e(n)W_{N/2}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x_0(n)W_{N/2}^{nk}}$$

Matlab Code:

```
clear all;
close all;

y=[0 1 2 3 4 5 6 7];    %Input signal

n=length(y);
p=nextpow2(n);  % Increasing the performance of fft when the length
                %of the signal is not a power of two
z=zeros(1,2^p-n);
x=[y z];
y=bitrevorder(x);  % Bit Reversal to obtain the correct order
n=length(y);    % Length of the Input Signal
s=log2(n);     % Number of stages
w=exp(-2*1j*pi/n).^(0:(n/2-1));    % Twiddle Factor

%DIT
for m=1:s
    for k=1:2^m:n
        for l=0:2^(m-1)-1
            a=y(k+l);
            b=y(k+l+2^(m-1))*w(l*n/(2^m)+1);
            y(k+l)=a+b;
            y(k+l+2^(m-1))=a-b;
        end
    end
end

k=1:n;
subplot(3,1,1);
stem(k,x); % Plot for Input Signal
```

```
xlabel('n');
ylabel('Amplitude');
title('Input Signal');


y=(round(y*100))/100;
subplot(3,1,2);
stem(x,y);
xlabel('y');
ylabel('Amplitude');
title('DIT');
disp(y);

%DFT using inbuilt function
y1=fft(x);
y1=(round(y*100))/100;
subplot(3,1,3);
stem(x,y1);
ylabel('Amplitude');
xlabel('y');
title('DFT')
disp(y1);
```

Output:



CONCLUSION:

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