Digital Signal Processing

Lab Journal

Name: Dhanya Sri Bolla

Admission No: U19EC129

Branch: Electronics and Communication

Course

Co-Ordinator: Mehul C. patel sir

Course Code: EC305

PRACTICAL NO: 00

27/7/2021

Aim:

To verify sampling theorem using MATLAB.

Theory:

Sampling is a process that converts continuous signal into discrete time signal. 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 fs is greater than or equal to the twice the highest frequency component of message signal.

Over-sampling:

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

fs > 2fm

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 message signal.

fs = 2fm

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 message signal.

fs < 2fm

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 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;

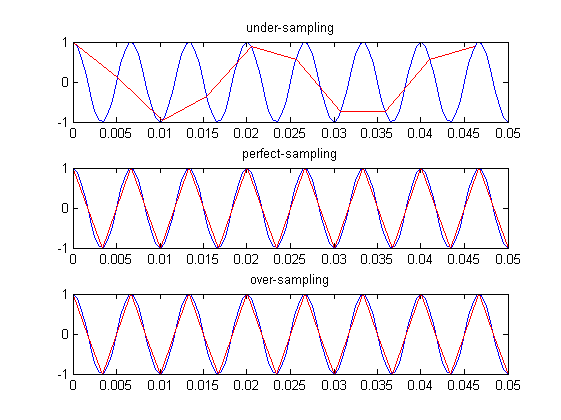
xn=cos(2\*pi\*n3\*fd);

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 highest component of message signal, the original signal can be recovered.

PRACTICAL NO: 01

03/08/2021

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* ± *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);

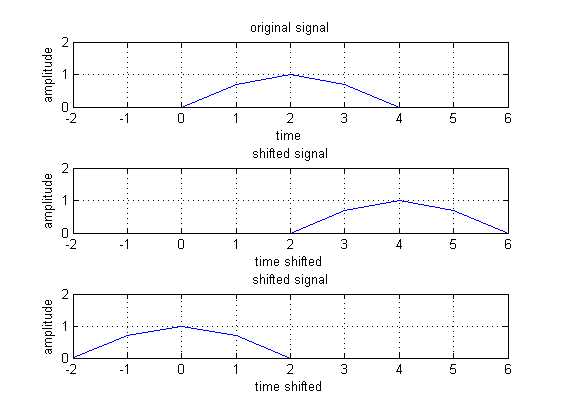
xlabel('time shifted')

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\*-');

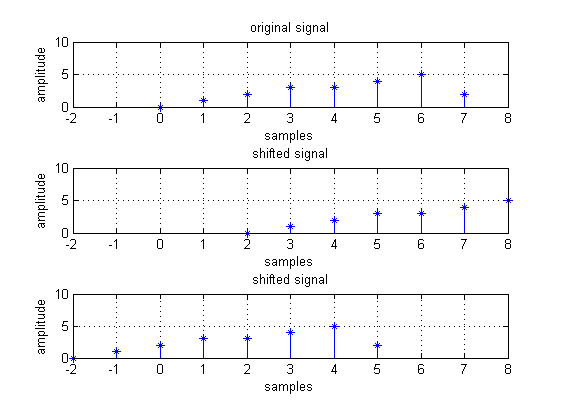
xlabel('samples')

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);

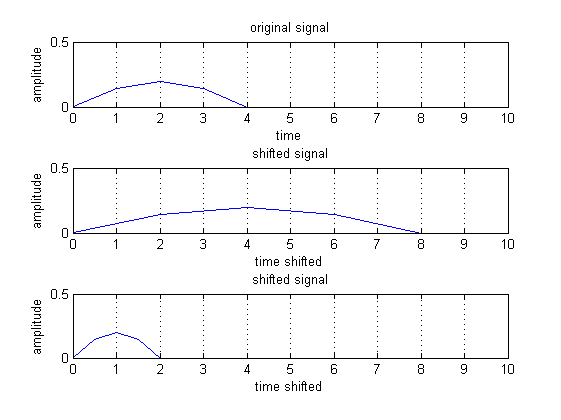
xlabel('time shifted')

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\*-');

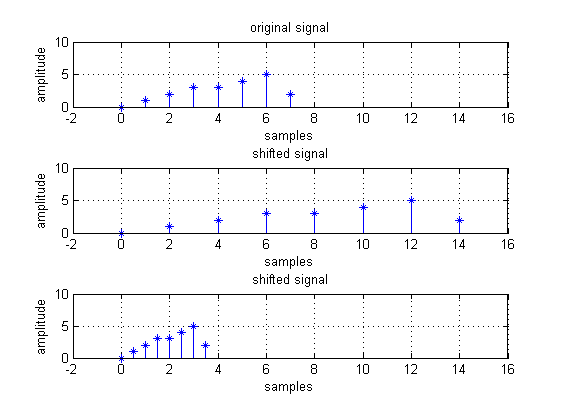
xlabel('samples')

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);

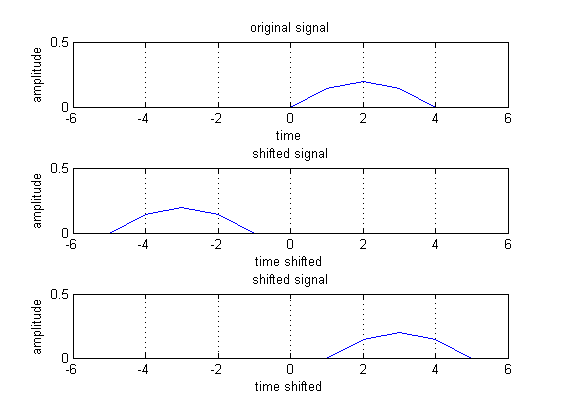
xlabel('time shifted')

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\*-');

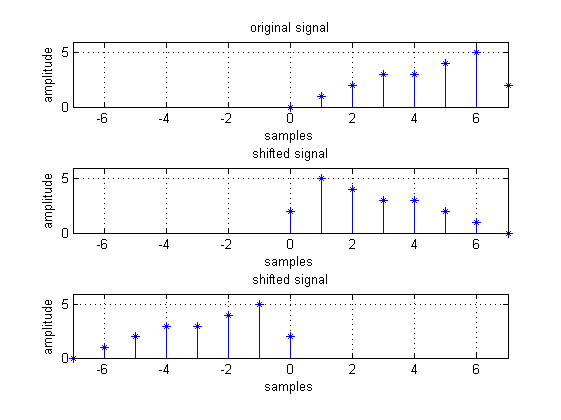
xlabel('samples')

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

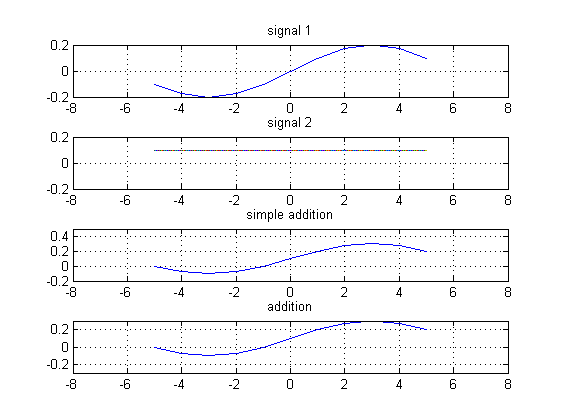
subplot(4,1,4);

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;

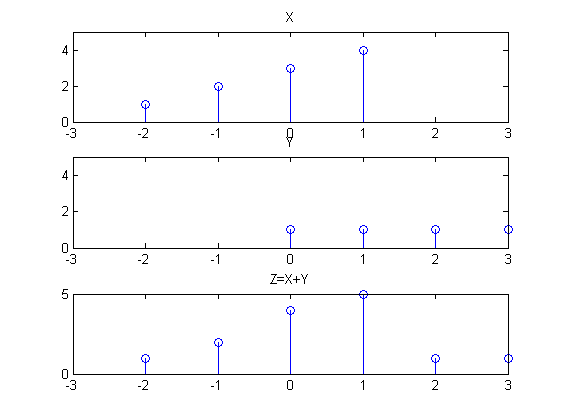
add=s1 +s2;

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

PRACTICAL NO: 02

10/08/2021

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) = ∑k=-∞∞ 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

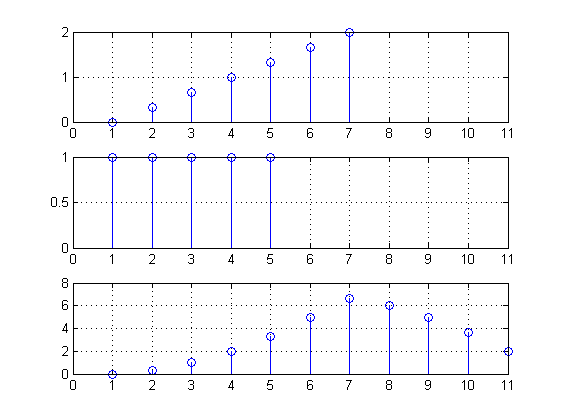
1 1 1 1 1

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

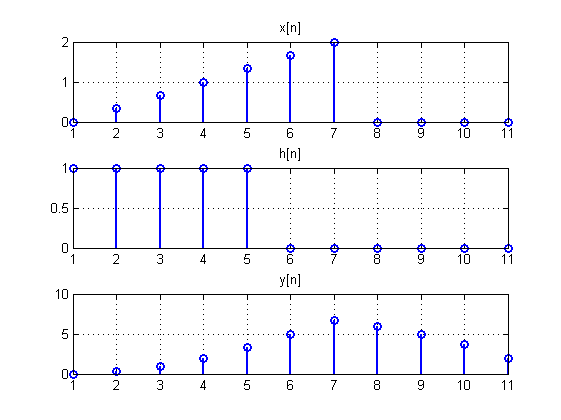
1 1 1 1 1

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.

PRACTICAL NO: 03

17/08/2021

Aim:

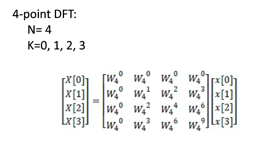
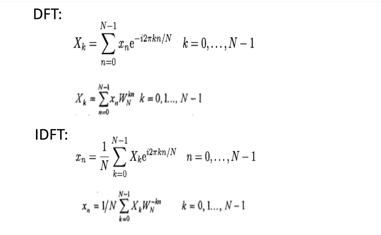
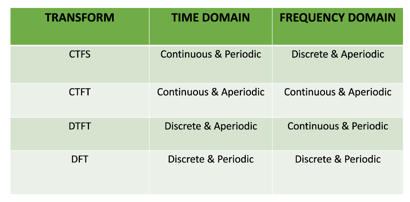
Write a simulation program to compare DFT, IDFT for the given signal.

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.



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)

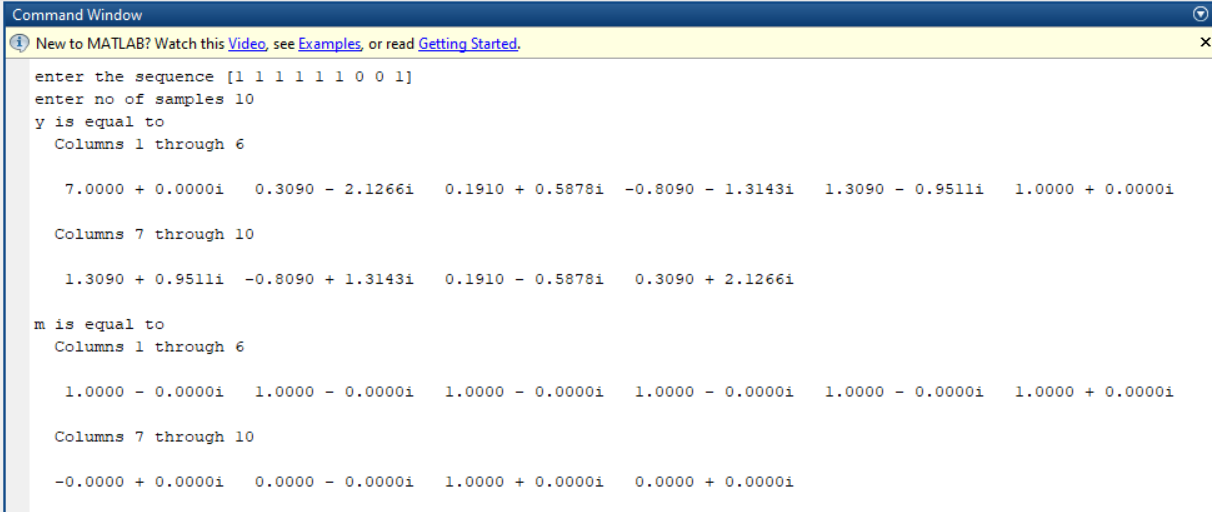
stem(n,m,'linewidth',2)

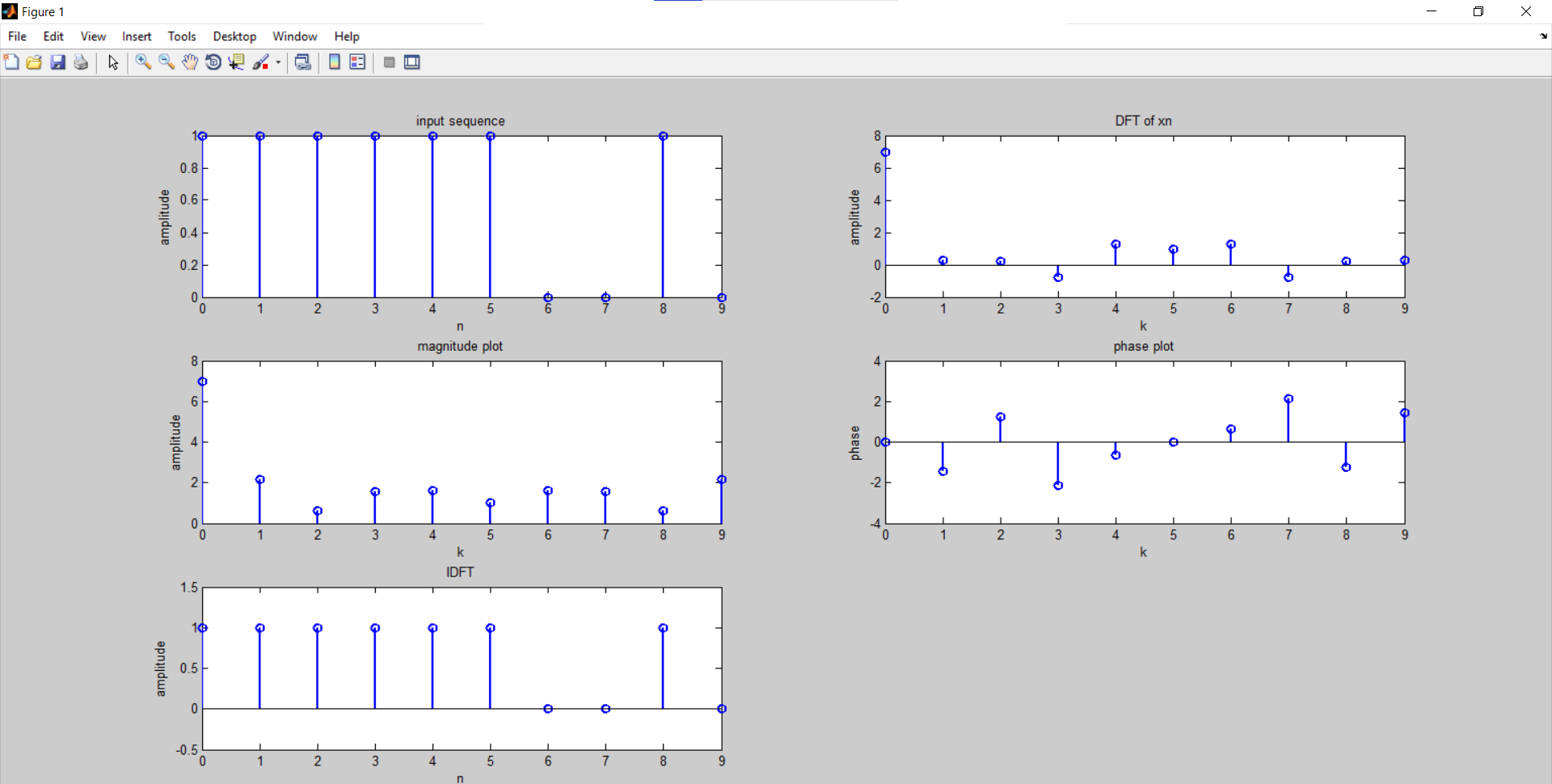
xlabel('n')

ylabel('amplitude')

title('IDFT')

Output:





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)

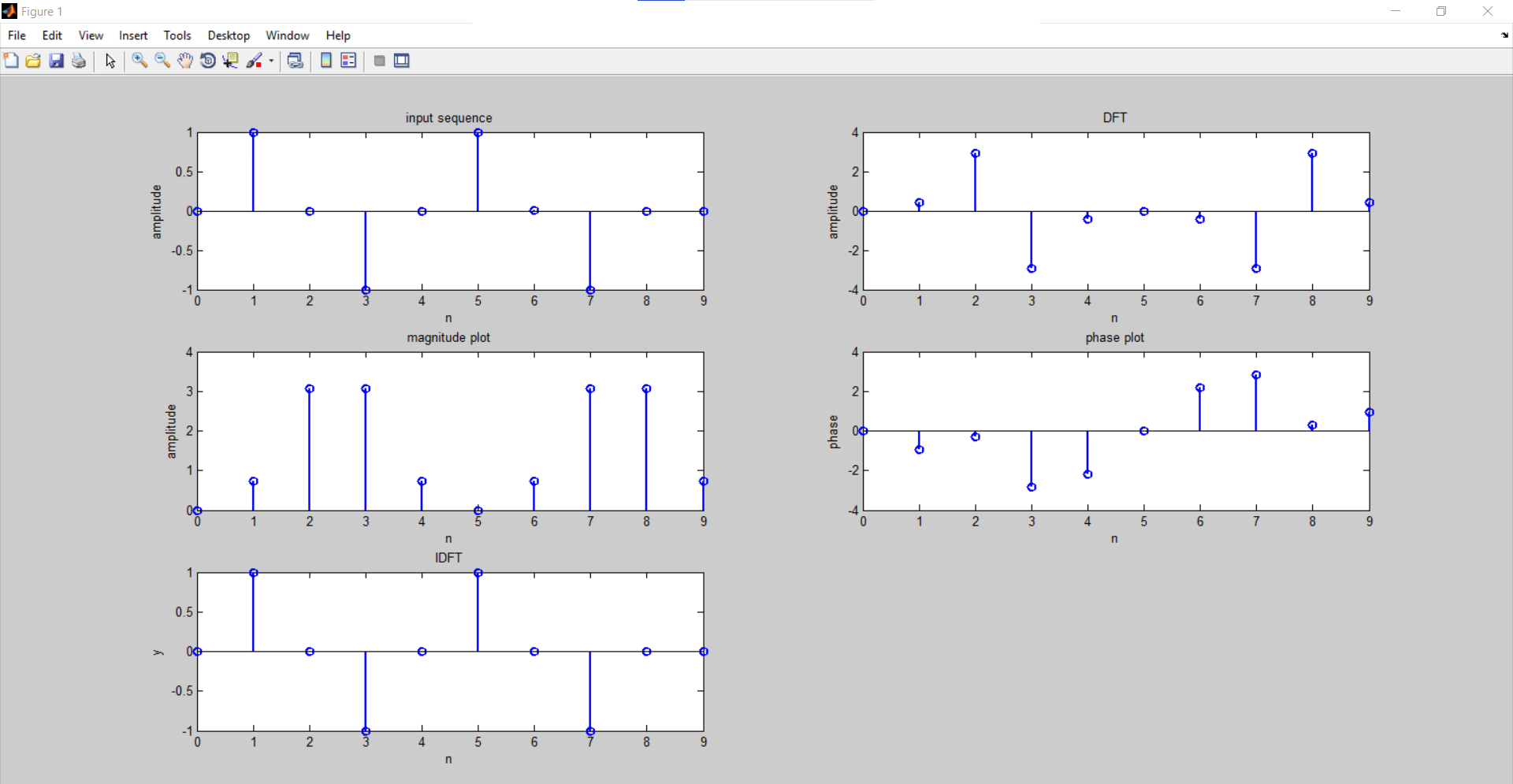
stem(n,y,'linewidth',2);

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 in-built function.

PRACTICAL NO: 04

24/08/2021

Aim:

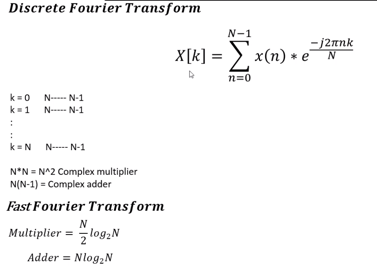
Write a MATLAB program to implement 8-point Decimation in time – Fast fourier transform algorithm.

X[n] = cos(πn/2) 0 ≤ n ≤ 7

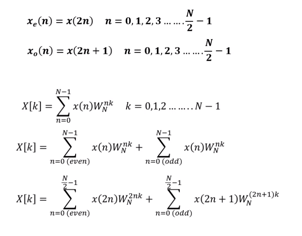
Plot magnitude and phase response.

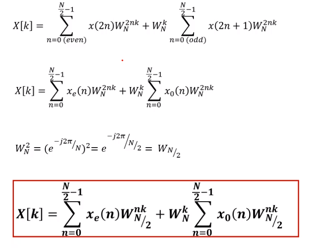
Theory:

Discrete time fourier transform:



Decimation in time algorithm:





MATLAB Code:

## DITFFT

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;

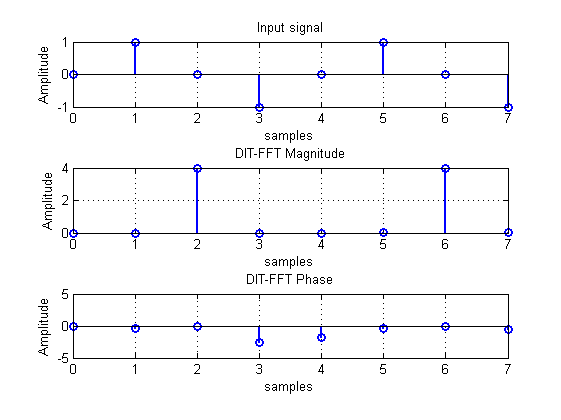
Columns 1 through 4

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 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.

PRACTICAL NO : 05

31/08/2021

Aim:

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

X[n] = sin(πn/2) 0 ≤ n ≤ 7

Plot the magnitude and phase response.

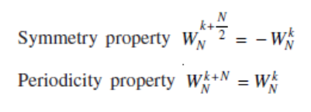
Theory:

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.

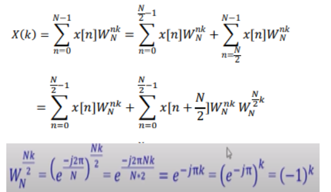
FFT algorithm exploits the above two symmetry properties and so is an efficient algorithm for DFT computation.

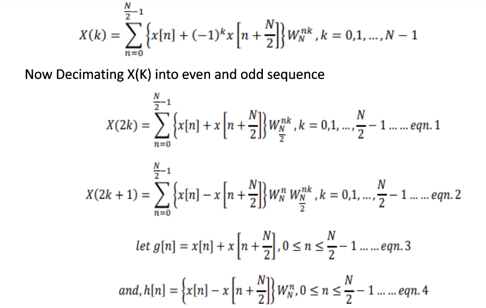


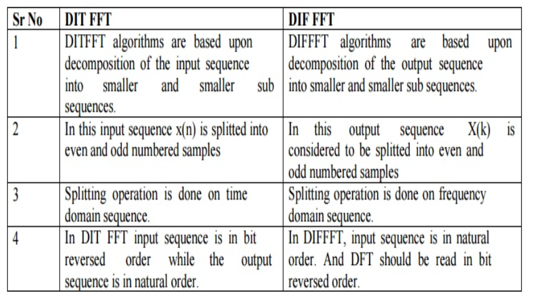
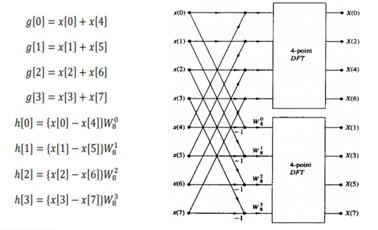
Decimation in frequency – FFT :

In decimation in frequency algorithm, 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.







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 = 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

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);

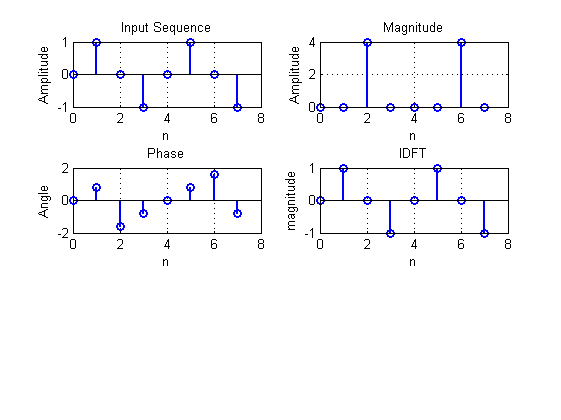
xlabel('n');

ylabel('magnitude');

title('IDFT')

grid on;

Output:



## DIF-FFT using inbult 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);

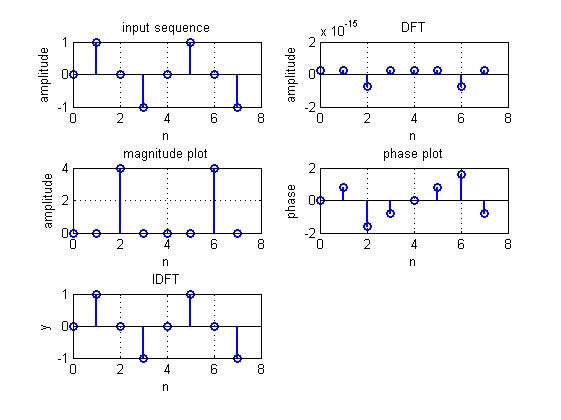
xlabel('n')

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 in-built function.