**Experiment No. 3**

**AIM**

Write MATLAB codes for each question.

**THEORY**

**DISCRETE FOURIER TRANSFORM (DFT)**

The DFT is a powerful computation tool which allows us to evaluate the fourier transform *X(ejw)*. Unlike DTFT which is defined for finite and infinite sequences, DFT is defined only for sequences of finite length. Since *X(ejw)* is continuous and periodic, DFT is obtained by sampling one period of the fourier transform at a finite number of frequency points.

DFT is defined as

Here k is used to denote the frequency domain ordinal and n is used to represent the time domain ordinal.

The synthesis equation (IDFT) is defined as

**CONVOLUTION THEOREM**

Multiplication in the discrete-time domain becomes circular convolution in the discrete frequency domain and circular convolution in the discrete-time domain becomes multiplication in the discrete-frequency domain.

**FAST FOURIER TRANSFORM (FFT)**

The Fast Fourier Transform is a highly efficient procedure for computing the DFT of a finite series and requires less number of computations than that of direct evaluation of DFT. It reduces the computations by taking advantage of the fact that the calculation of the coefficients of the DFT can be carried out iteratively.

FFT is based on decomposition and breaking the transform into smaller transforms and combining them to get the total transform. It reduces the computation time and improves the performance over direct evaluation of the DFT.

**CIRCULAR CONVOLUTION**

Circular convolution is used to convolve two Discrete Fourier Transform (DFT) sequences. For long sequences, circular convolution can be faster than linear convolution.

If *X1[k]* and *X2[k]* are the DFTs of sequences *x1[n]* and *x2[n]* (of length N) respectively, then for *X3[k]= X1[k]. X2[k],* taking IDFT

On solving,

which represents the circular convolution of *x1[n]* and *x2[n].*

**INBUILT MATLAB FUNCTIONS USED**

1. fft

Syntax : Y = fft(x)

Y = fft(x) returns the discrete Fourier transform (DFT) of vector x, computed with a fast Fourier transform (FFT) algorithm.

1. ifft

Syntax: y = ifft(X)

y = ifft(X) returns the inverse discrete Fourier transform (DFT) of vector X, computed with a fast Fourier transform (FFT) algorithm. If X is a matrix, ifft returns the inverse DFT of each column of the matrix.

1. abs

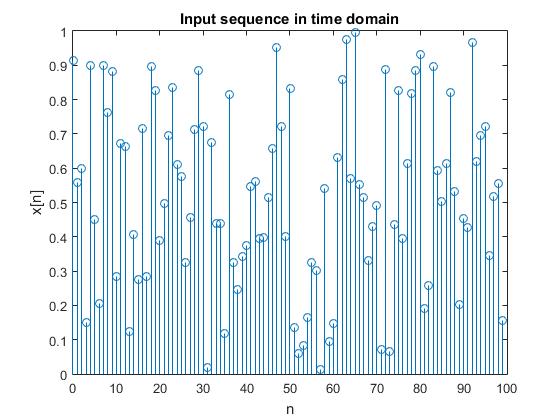
Syntax: Y = abs([X](file:///C:\Program%20Files\MATLAB\MATLAB%20Production%20Server\R2015a\help\matlab\ref\abs.html#inputarg_X))

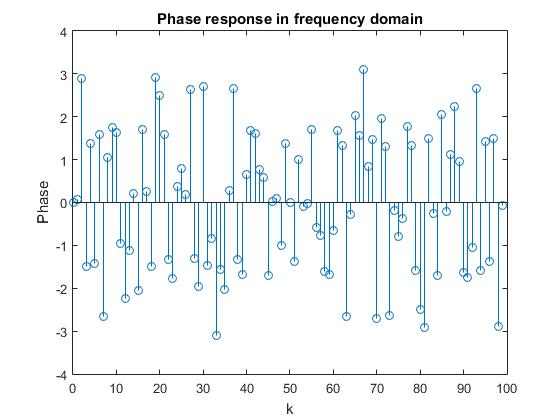
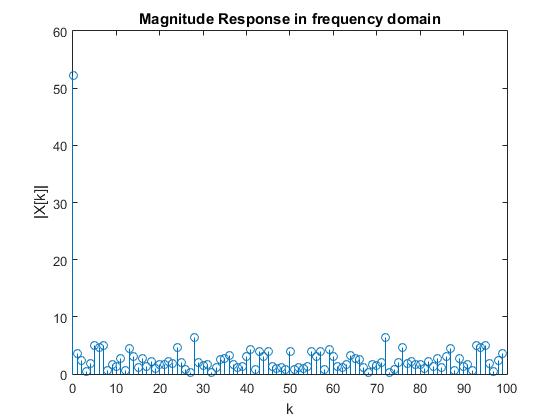
Y = abs([X](file:///C:\Program%20Files\MATLAB\MATLAB%20Production%20Server\R2015a\help\matlab\ref\abs.html#inputarg_X)) returns the [absolute value](file:///C:\Program%20Files\MATLAB\MATLAB%20Production%20Server\R2015a\help\matlab\ref\abs.html#budexws-4) of each element in array X.

1. angle

Syntax: P = angle(Z)

P = angle(Z) returns the phase angles, in radians, for each element of complex array Z. The angles lie between ±*π*.





**MATLAB CODE**

%Compute and plot N-point DFT X[k](both magnitude and phase spectrum) of sequence x[n] of length N.

clc;

clear var;

N=100;

x = rand(1,N);

n=zeros(1,N);

for i=1:N

n(i)=i-1;

end

ek=exp(-j\*2\*pi\*n/N);

X=zeros(1,N);

for k=1:N

for t=1:N

X(k)=X(k)+x(t)\*(ek(t)^(k-1));

end

end

y=fft(x);

figure;

stem(n,x);

xlabel('n');

ylabel('x[n]');

title('Input sequence in time domain');

figure;

stem(n,abs(X));

xlabel('k');

ylabel('|X[k]|');

title('Magnitude Response in frequency domain');

figure;

stem(n,angle(X));

xlabel('k');

ylabel('Phase');

title('Phase response in frequency domain');

X

y

%Compute and plot N-point IDFT x[n](both magnitude and phase spectrum) of X[k].

clc;

clear var;

N=5;

X = rand(1,N)\*100;

k=zeros(1,N);

for i=1:N

k(i)=i-1;

end

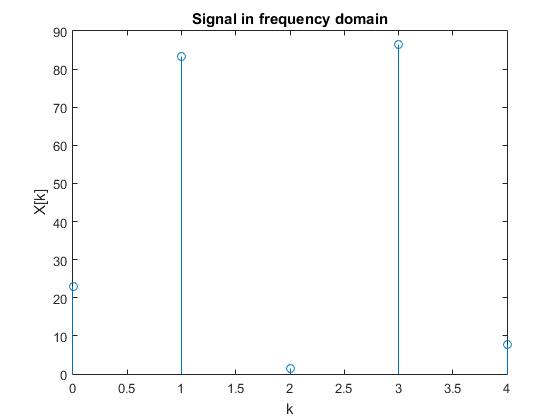
ek=exp(j\*2\*pi.\*k/N);

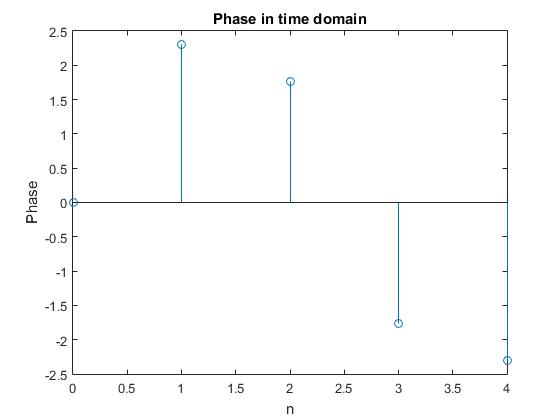
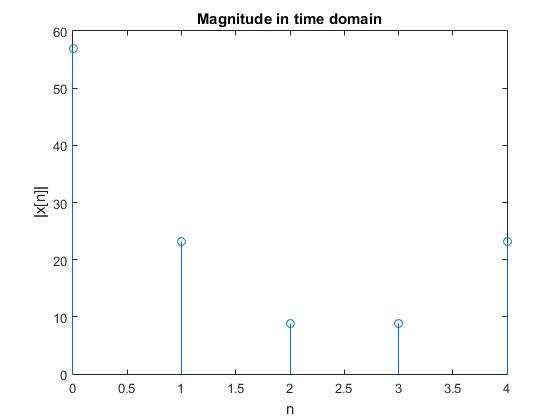
x=zeros(1,N);

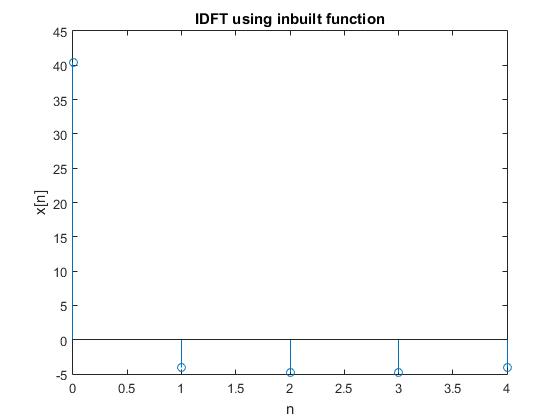
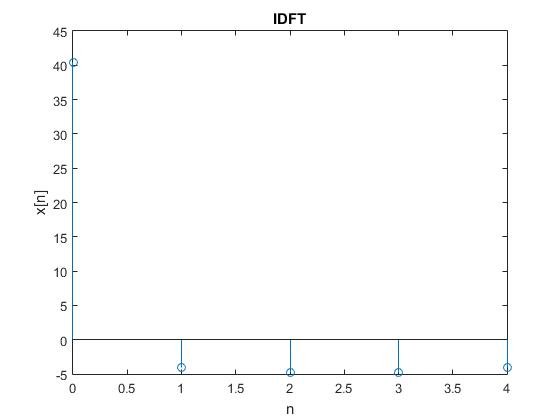
for n=0:N-1

for t=1:N

x(n+1)=x(n+1)+X(t)\*(ek(t)^n);







end

x(n+1)=x(n+1)/N;

end

y=ifft(X);

figure;

stem(k,X);

xlabel('k');

ylabel('X[k]');

title('Signal in frequency domain');

figure;

stem(k,abs(x));

xlabel('n');

ylabel('|x[n]|');

title('Magnitude in time domain');

figure;

stem(k,angle(x));

xlabel('n');

ylabel('Phase');

title('Phase in time domain');

figure;

stem(k,x);

xlabel('n');

ylabel('x[n]');

title('IDFT');

figure;

stem(k,y);

xlabel('n');

ylabel('x[n]');

title('IDFT using inbuilt function');

%dft using radix 2 algorithm

clear all

close all

x = [1 4 8 9 9 8 2 10 5 2 9 5 2 3 9 5 4 9 1 0 3 2 8 5];

N1=nextpow2(length(x));

N=2^N1;

if (N>length(x))

x=[x zeros(1,N-length(x))];

end

Xt=rad2dft(x,N);

Yt=fft(x);

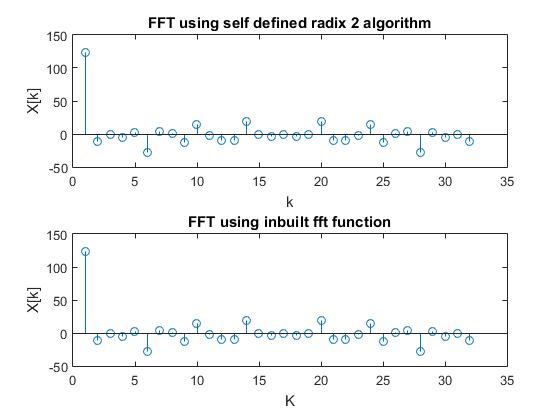
figure;

subplot(2,1,1),stem(Xt),title('FFT using self defined radix 2 algorithm'), xlabel('k'),ylabel('X[k]');

subplot(2,1,2),stem(Yt),title('FFT using inbuilt fft function'), xlabel('K'),ylabel('X[k]');

function X=rad2dft(x,N)

X=zeros(1,N);



if (N==2)

X(1)=x(1)+x(2);

X(2)=x(1)-x(2);

else

X(1:N/2)=rad2dft(x(1:2:N),N/2);

X(N/2+1:N)=rad2dft(x(2:2:N),N/2);

for k=0:N/2-1

t=X(k+1);

W=exp(2\*pi\*1j\*k/N);

X(k+1)=t+W\*X(k+N/2+1);

X(k+N/2+1)=t-W\*X(k+N/2+1);

end

end

end

%DFT for a rectangular signal

x=[zeros(1,3),ones(1,8),zeros(1,5)];

g=[zeros(1,10),ones(1,8),zeros(1,10)];

N=length(x);

n=zeros(1,N);

for i=1:N

n(i)=i-1;

end

ek=exp(-j\*2\*pi\*n/N);

X=zeros(1,N);

for k=1:N

for t=1:N

X(k)=X(k)+x(t)\*(ek(t)^(k-1));

end

end

N1=length(g);

n1=zeros(1,N1);

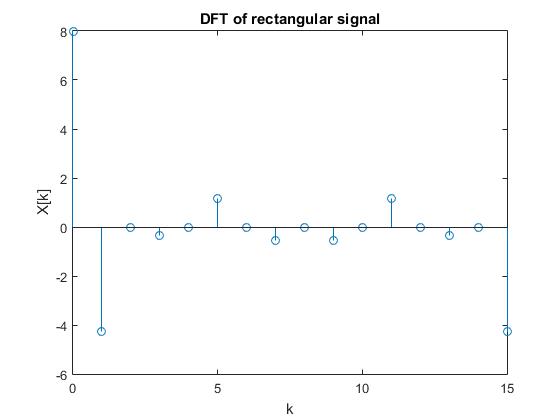
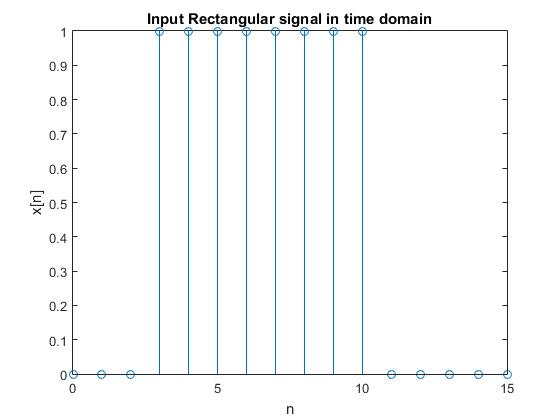
for i=1:N1

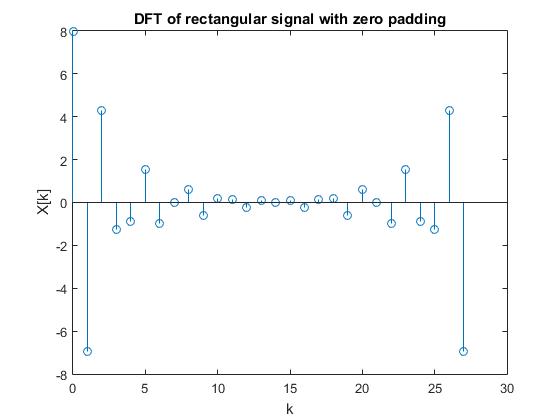
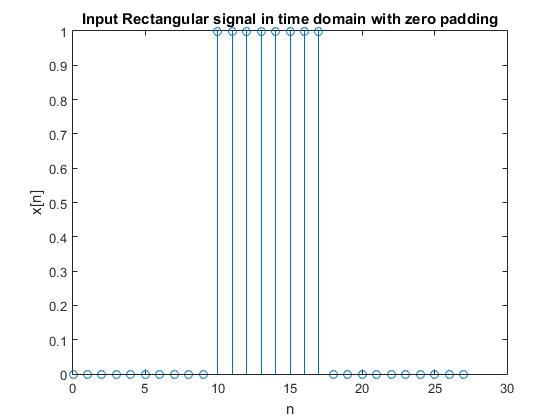
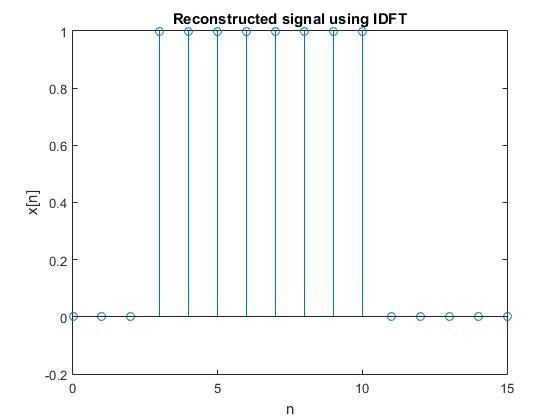
n1(i)=i-1;

end

ek=exp(-j\*2\*pi\*n1/N1);

G=zeros(1,N1);





for k=1:N1

for t=1:N1

G(k)=G(k)+g(t)\*(ek(t)^(k-1));

end

end

figure;

stem(n,x);

xlabel('n');

ylabel('x[n]');

title('Input Rectangular signal in time domain');

figure;

stem(n,X);

xlabel('k');

ylabel('X[k]');

title('DFT of rectangular signal');

figure;

stem(n1,g);

xlabel('n');

ylabel('x[n]');

title('Input Rectangular signal in time domain with zero padding');

figure;

stem(n1,G);

xlabel('k');

ylabel('X[k]');

title('DFT of rectangular signal with zero padding');

%idft

k=zeros(1,N);

for i=1:N

k(i)=i-1;

end

ek=exp(j\*2\*pi.\*k/N);

x=zeros(1,N);

for n=0:N-1

for t=1:N

x(n+1)=x(n+1)+X(t)\*(ek(t)^n);

end

x(n+1)=x(n+1)/N;

end

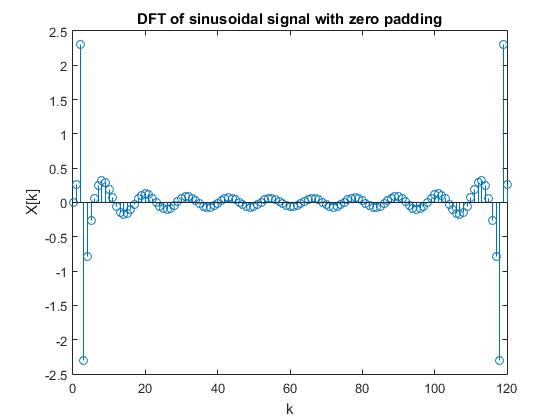
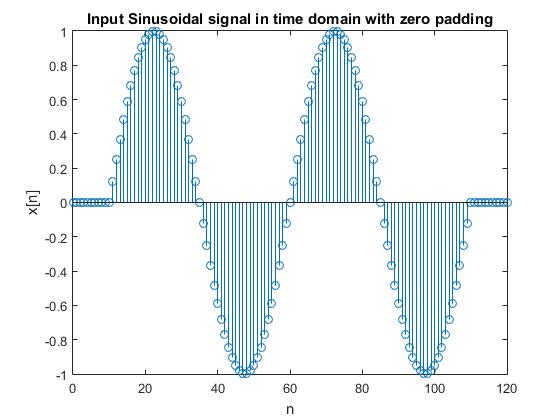
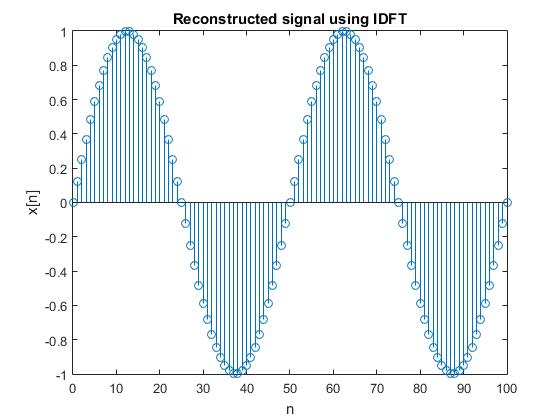
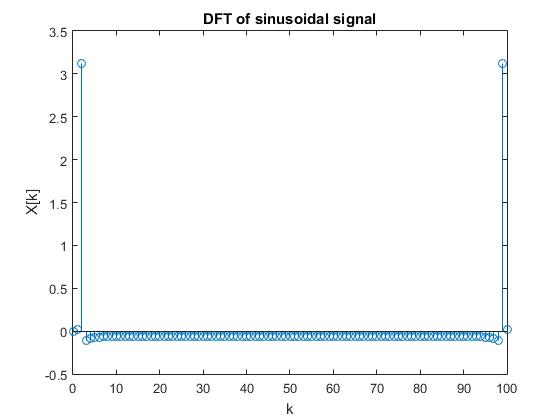
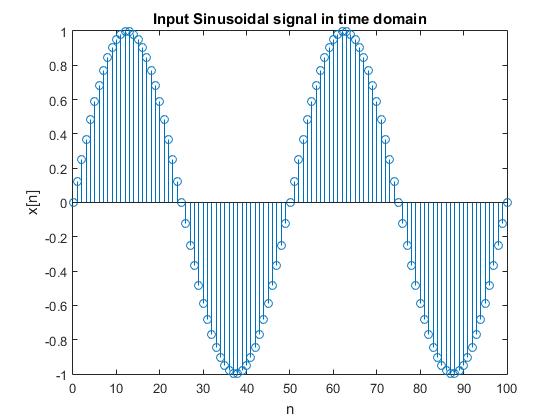
figure;

stem(k,x);

xlabel('n');

ylabel('x[n]');

title('Reconstructed signal using IDFT');



%DFT of sinusoidal signal

f=[-5:0.1:5];

x=sin(2\*pi\*f/5);

g=[zeros(1,10) sin(2\*pi\*f/5) zeros(1,10)];

N=length(x);

n=zeros(1,N);

N1=length(g);

n1=zeros(1,N1);

for i=1:N

n(i)=i-1;

end

ek=exp(-j\*2\*pi\*n/N);

X=zeros(1,N);

for k=1:N

for t=1:N

X(k)=X(k)+x(t)\*(ek(t)^(k-1));

end

end

for i=1:N1

n1(i)=i-1;

end

ek=exp(-j\*2\*pi\*n1/N1);

G=zeros(1,N1);

for k=1:N1

for t=1:N1

G(k)=G(k)+g(t)\*(ek(t)^(k-1));

end

end

figure;

stem(n,x);

xlabel('n');

ylabel('x[n]');

title('Input Sinusoidal signal in time domain');

figure;

stem(n,X);

xlabel('k');

ylabel('X[k]');

title('DFT of sinusoidal signal');

figure;

stem(n1,g);

xlabel('n');

ylabel('x[n]');

title('Input Sinusoidal signal in time domain with zero padding');

figure;

stem(n1,G);

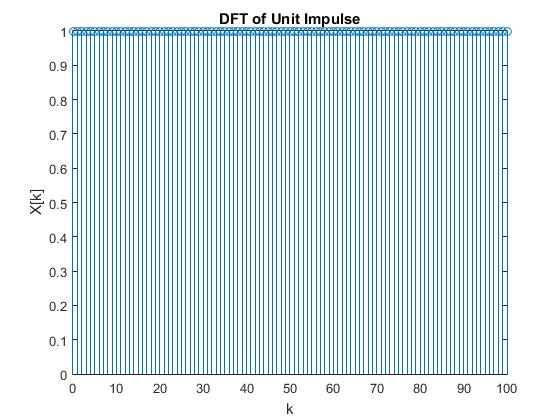
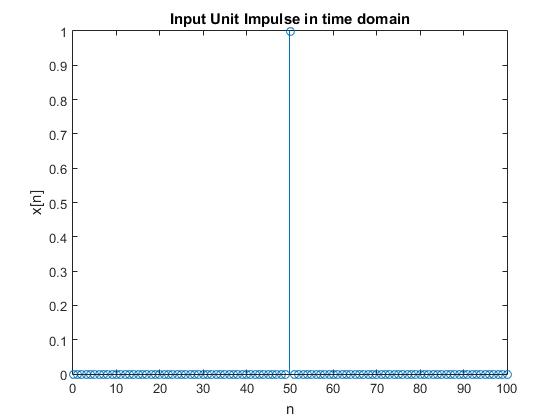
xlabel('k');

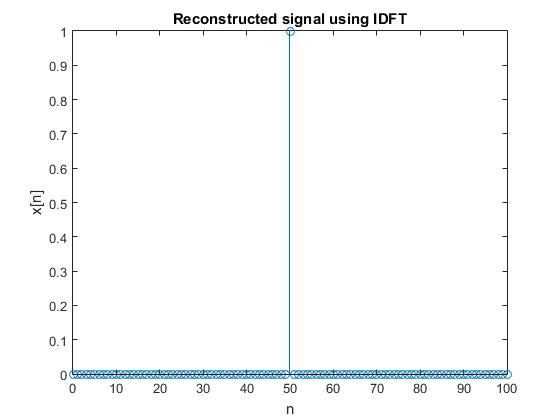
ylabel('X[k]');

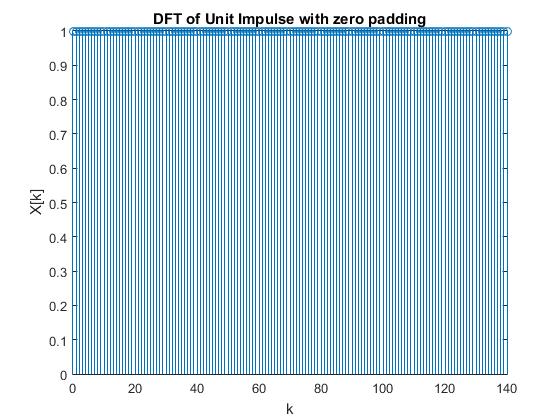
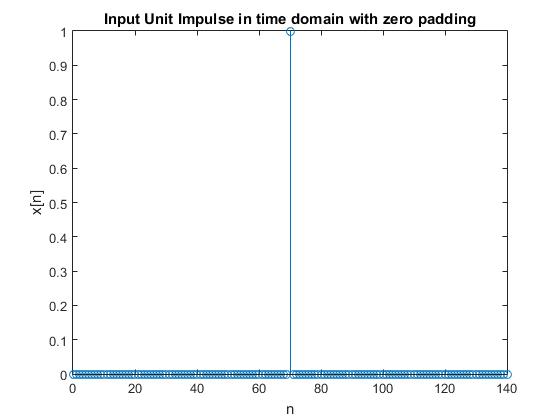
title('DFT of sinusoidal signal with zero padding');

%IDFT of sinusoidal signal

k=zeros(1,N);







for i=1:N

k(i)=i-1;

end

ek=exp(j\*2\*pi.\*k/N);

x=zeros(1,N);

for n=0:N-1

for t=1:N

x(n+1)=x(n+1)+X(t)\*(ek(t)^n);

end

x(n+1)=x(n+1)/N;

end

figure;

stem(k,x);

xlabel('n');

ylabel('x[n]');

title('Reconstructed signal using IDFT');

stem(k,abs(x));

xlabel('n');

ylabel('x[n]');

title('Reconstructed signal using IDFT');

%Circular convolution using convolution theorem

x=[1 2 3 4 ];

lx=length(x);

h=[1 1 1];

lh=length(h);

X=dft(x);

H=dft(h);

if(lx<lh)

X=[X zeros(1,lh-lx)];

else if(lx>lh)

H=[H zeros(1,lx-lh)];

end

end

T=X.\*H;

T1=idft(T);

n=0:length(T1)-1;

figure;

stem(n,T1);

title('Circular Convolution');

ylabel('y(n)');

xlabel('n');

%Linear convolution using convolution theorem

x=[1 2 3 4];

lx=length(x);

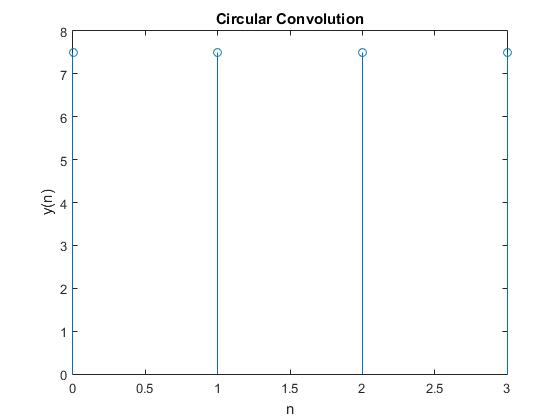
h=[1 1 1];

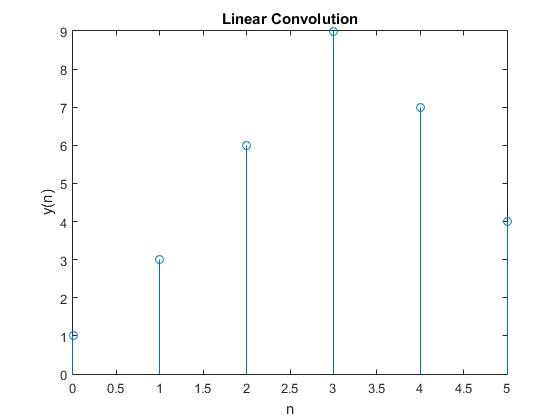
lh=length(h);

x1=[x zeros(1,lh-1)];

h1=[h zeros(1,lx-1)];

X=dft(x1);





H=dft(h1);

Y=X.\*H;

y=idft(Y);

n=0:length(y)-1;

stem(n,y);

title('Linear Convolution');

ylabel('y(n)');

xlabel('n');

%function for dft

function [ dft1 ] = dft( x )

l=length(x);

dft1=zeros(1,l);

for k=0:l-1

for(n=0:l-1)

dft1(k+1)=dft1(k+1) + (x(n+1)\*exp((-1i)\*2\*pi\*k\*n/l));

end

end

end

%function for idft

function idft1=idft(x);

l=length(x);

idft1=zeros(1,l);

for n=0:l-1

for k=0:l-1

idft1(n+1)=idft1(n+1) + (x(k+1)\*exp((1i)\*2\*pi\*k\*n/l));

end

end

idft1=idft1./l;

end

**INFERENCES**

1. Rectangular function in time domain gives sinc function in frequency domain.
2. DFT of sinusoidal signal contains unit impulse at that value of k which corresponds to frequency of original analog signal.
3. Unit impulse in time domain leads to a constant signal over all range of frequencies in Fourier domain.
4. Circular convolution of two N-point sequences would again be an N-point sequence.
5. It can be observed that as we zero pad the given sequence, the length of the signal in time domain increases. Hence length of N-point DFT also increases. Also as N increases, frequency resolution of DFT increases.

**RESULTS**

MATLAB programs to compute and plot N-point DFT and IDFT was written and the result verified using inbuilt functions fft and ifft. The program was checked by computing DFT and IDFT for some known signals. Circular Convolution and Linear convolution using convolution theorem of DFT was implemented.