Experiment No 3

To find DFT and IDFT of a sequence

```
close all;
clear all;
x=input('Enter the input sequence x[n]=')
L=length(x)
N=input('Enter the length of the DFT sequence N =');
if(N<L)
    disp('N should be greater than L')
else
    %Wn=-j*2*pi/N;
    for k=0:N-1
        X(k+1)=0;
        for n=0:L-1
            X(k+1)=X(k+1)+x(n+1)*exp(-j*2*pi*n*k/N);
        end
    end
    disp('DFT of x[n] is X(K)=')
    disp(X)
end

%Verification
disp('DFT using built in function')
Y=fft(x,N)
disp(Y)

%Inverse DFT
for n=0:N-1
    y(n+1)=0;
    for k=0:L-1
        y(n+1)=y(n+1)+X(k+1)*exp(j*2*pi*n*k/N);
    end
    y(n+1)=y(n+1)/N;
end
disp('IDFT of X(K)is y(n)=')
disp(y)

%Verification
disp('IDFT using built in function ')
Z=ifft(X)
disp(Z)

a=0:L-1;
subplot(3,2,1)
stem(a,x)
```

```
grid on
title('Input Sequence x[n]')
xlabel('Samples')
ylabel('Values')

b=0:N-1;
subplot(3,2,2)
stem(b,X)
grid on
title('DFT Sequence X(K)')
xlabel('Samples')
ylabel('Values')

subplot(3,2,3)
stem(b,abs(X))
grid on
title('DFT magnitude')
xlabel('Samples')
ylabel('Values')

subplot(3,2,4)
stem(b,angle(X))
grid on
title('DFT phase angle')
xlabel('Samples')
ylabel('Values')

subplot(3,2,5)
stem(y)
grid on
title('IDFT sequence y[n]')
xlabel('Samples')
ylabel('Values')
```

Result:

```
Enter the input sequence x[n]=[1 2 3 4]

x =

   1   2   3   4


L =

   4
```

Enter the length of the DFT sequence N =4
DFT of x[n] is X(K)=
10.0000 + 0.0000i -2.0000 + 2.0000i -2.0000 - 0.0000i -2.0000 - 2.0000i

DFT using built in function

Y =

10.0000 + 0.0000i -2.0000 + 2.0000i -2.0000 + 0.0000i -2.0000 - 2.0000i

10.0000 + 0.0000i -2.0000 + 2.0000i -2.0000 + 0.0000i -2.0000 - 2.0000i

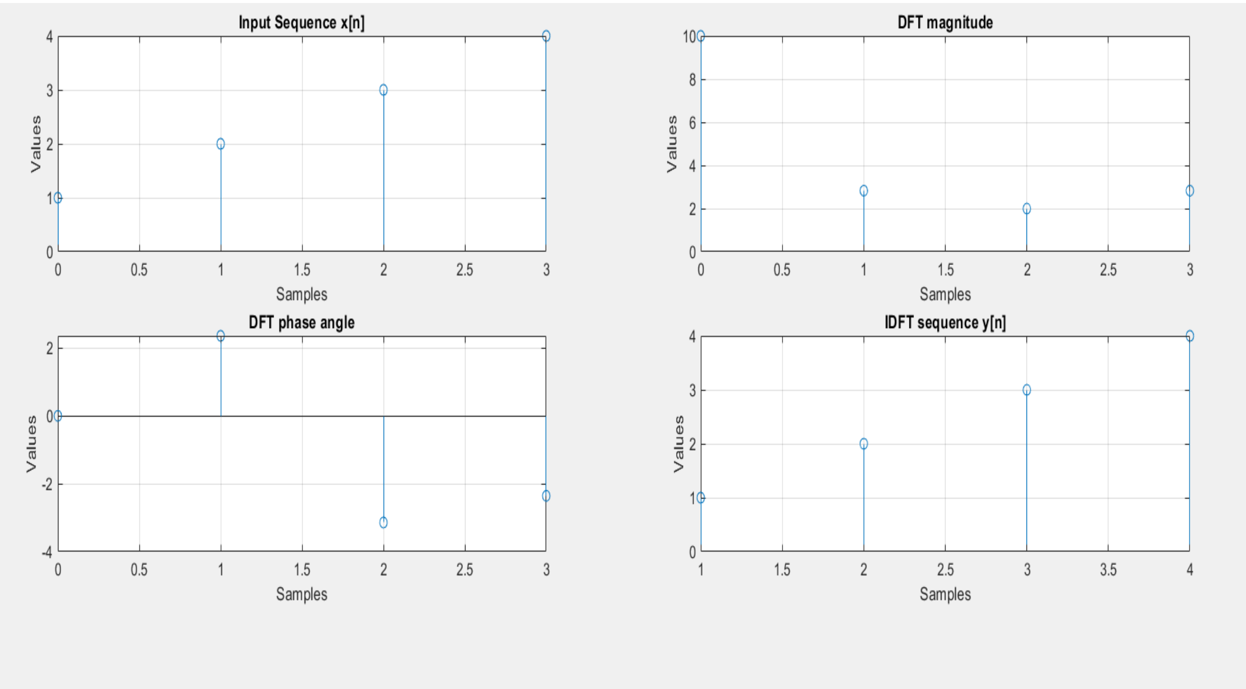
IDFT of X(K)is y(n)=
1.0000 - 0.0000i   2.0000 - 0.0000i   3.0000 - 0.0000i   4.0000 + 0.0000i

IDFT using built in function

Z =

1.0000 - 0.0000i   2.0000 - 0.0000i   3.0000 + 0.0000i   4.0000 + 0.0000i

1.0000 - 0.0000i   2.0000 - 0.0000i   3.0000 + 0.0000i   4.0000 + 0.0000i



ii) To find FFT and IFFT of a sequence

```
close all;
```

```
clear all;
x=input('Enter the input Sequnce x[n]');
L=length(x);
N=input('Enter the length of the FFT sequence N =');
if(N<L)
    disp('N should be greater than L')
else
    x=[x zeros(1,N-L)];
end

%Plotting the Input Sequence
d=0:N-1;
subplot(3,2,1)
stem(d,x)
title('Input Sequence x[n]');

%To alter the input sequence ie x[0] x[2] x[1] x[3]
x=bitrevorder(x);

M=log2(N);
h=1;

for stage=1:M
    for index=0:(2^stage):N-1
        for n=0:(h-1)
            pos=n+index+1;
            pow=(2^(M-stage)*n);
            w=exp((-i)*(2*pi)*pow/N);
            a=x(pos)+x(pos+h).*w;
            b=x(pos)-x(pos+h).*w;
            x(pos)=a;
            x(pos+h)=b;
        end
    end
    h=2*h;
end
y=x
disp(y);
%Plotting the FFT Sequence

subplot(3,2,2)
stem(d,abs(y))
grid on
title('FFT magnitude')
xlabel('Samples')
ylabel('Values')
```

```
subplot(3,2,3)
stem(d,angle(y))
grid on
title('FFT phase angle')
xlabel('Samples')
ylabel('Values')

y=bitrevorder(y);
h=1;

for stage=1:M
    for index=0:(2^stage):N-1
        for n=0:(h-1)
            pos=n+index+1;
            pow=(2^(M-stage)*n);
            w=exp((i)*(2*pi)*pow/N);
            a=y(pos)+y(pos+h).*w;
            b=y(pos)-y(pos+h).*w;
            y(pos)=a;
            y(pos+h)=b;
        end
      end
    h=2*h;
end

z=y/N
disp(z)

%Plotting the IFFT Sequence
subplot(3,2,4)
stem(d,z)
title('IFFT Sequence z[n]');
```

Result:

Enter the input Sequnce x[n][1 2 3 4]
Enter the length of the FFT sequence N =4

y =

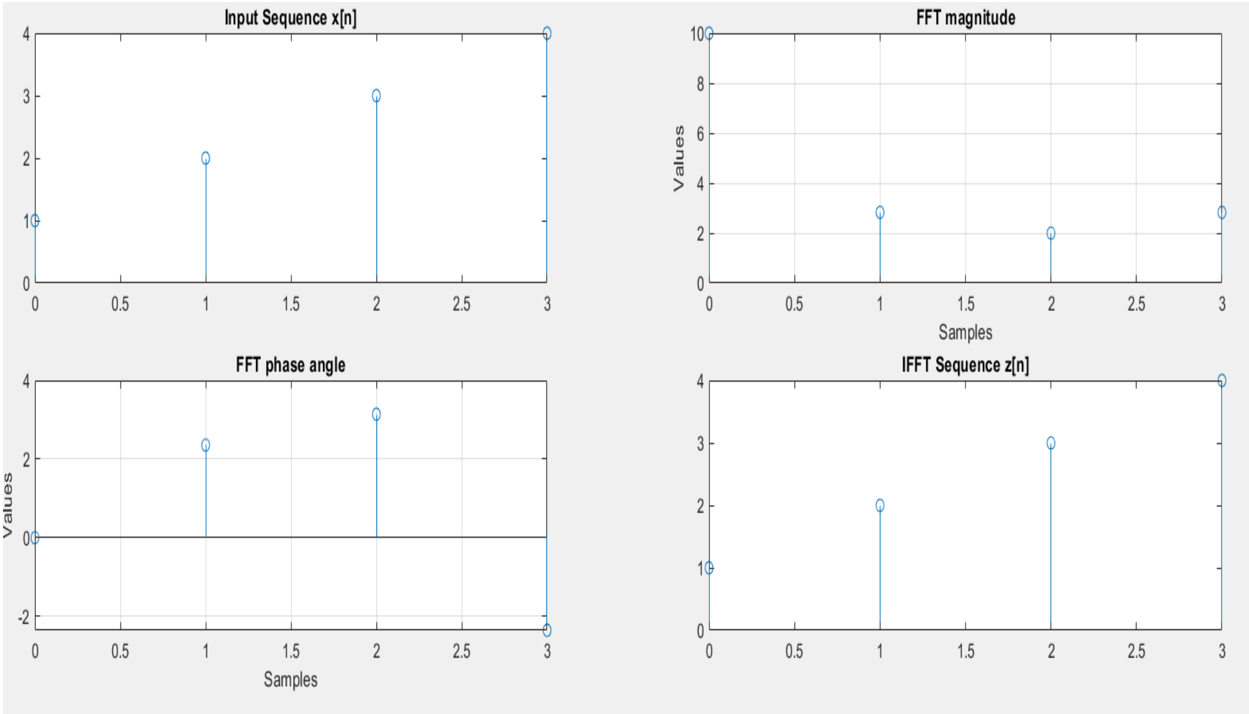
  10.0000 + 0.0000i -2.0000 + 2.0000i -2.0000 + 0.0000i -2.0000 - 2.0000i

  10.0000 + 0.0000i -2.0000 + 2.0000i -2.0000 + 0.0000i -2.0000 - 2.0000i

z =

1.0000 + 0.0000i   2.0000 + 0.0000i   3.0000 + 0.0000i   4.0000 - 0.0000i

1.0000 + 0.0000i   2.0000 + 0.0000i   3.0000 + 0.0000i   4.0000 - 0.0000i



iii)To find Linear and Circular Convolution using FFT algorithm

Linear Convolution

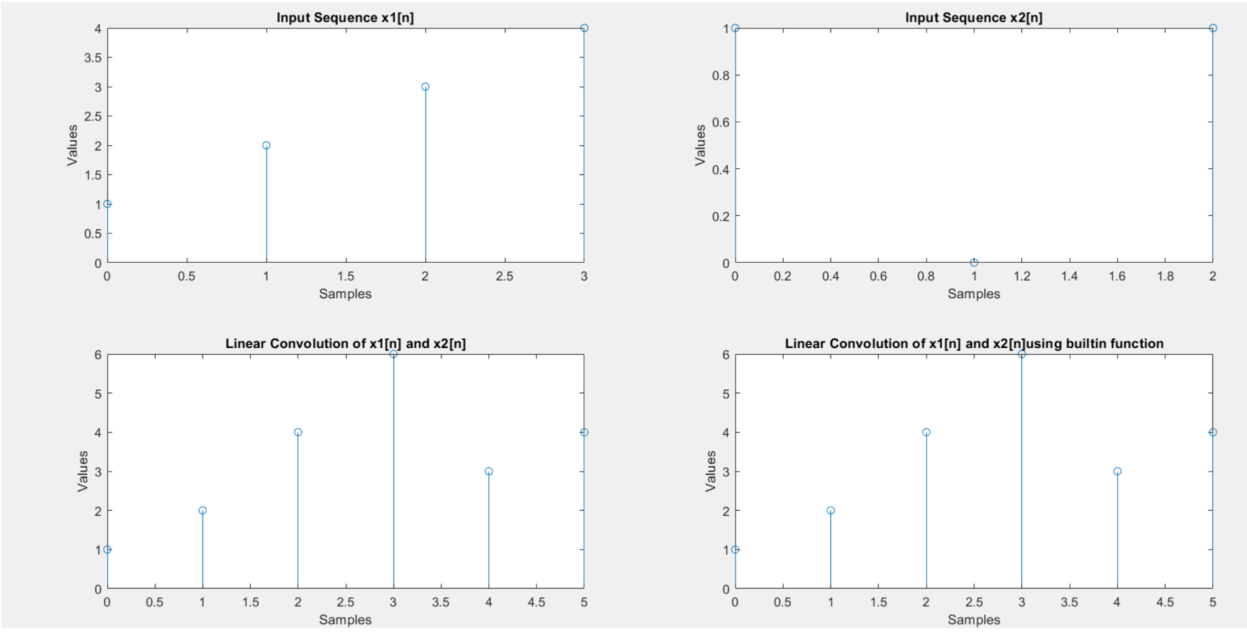
```
clear all;
close all;
x1=input('Enter the first input sequence x1[n] ');
x2=input('Enter the second input sequence x2[n] ');
```

```
Lx1=length(x1);
Lx2=length(x2);
N=Lx1+Lx2-1;
X1=fft(x1,N);
X2=fft(x2,N);
Y=X1.*X2;
y=ifft(Y,N);
disp('Linear Convolution of x1[n] and x2[n] is y[n]= ')
disp(y)

%Verification
z=conv(x1,x2);
disp('Linear Convolution of x1[n] and x2[n] using builtin
function is z[n]= ')
disp(z)

a=0:Lx1-1;
subplot(2,2,1)
stem(a,x1)
title('Input Sequence x1[n]')
xlabel('Samples')
ylabel('Values')

b=0:Lx2-1;
subplot(2,2,2)
stem(b,x2)
title('Input Sequence x2[n]')
xlabel('Samples')
ylabel('Values')

c=0:N-1;
subplot(2,2,3)
stem(c,y)
title('Linear Convolution of x1[n] and x2[n]')
xlabel('Samples')
ylabel('Values')

d=0:N-1;
subplot(2,2,4)
stem(c,z)
title('Linear Convolution of x1[n] and x2[n]using builtin
function')
xlabel('Samples')
ylabel('Values')
```

Result:

Enter the first input sequence x1[n] [1 2 3 4]
Enter the second input sequence x2[n] [1 0 1]
Linear Convolution of x1[n] and x2[n] is y[n]=
    1   2   4   6   3   4

Linear Convolution of x1[n] and x2[n] using builtin function is z[n]=
    1   2   4   6   3   4



Circular Convolution

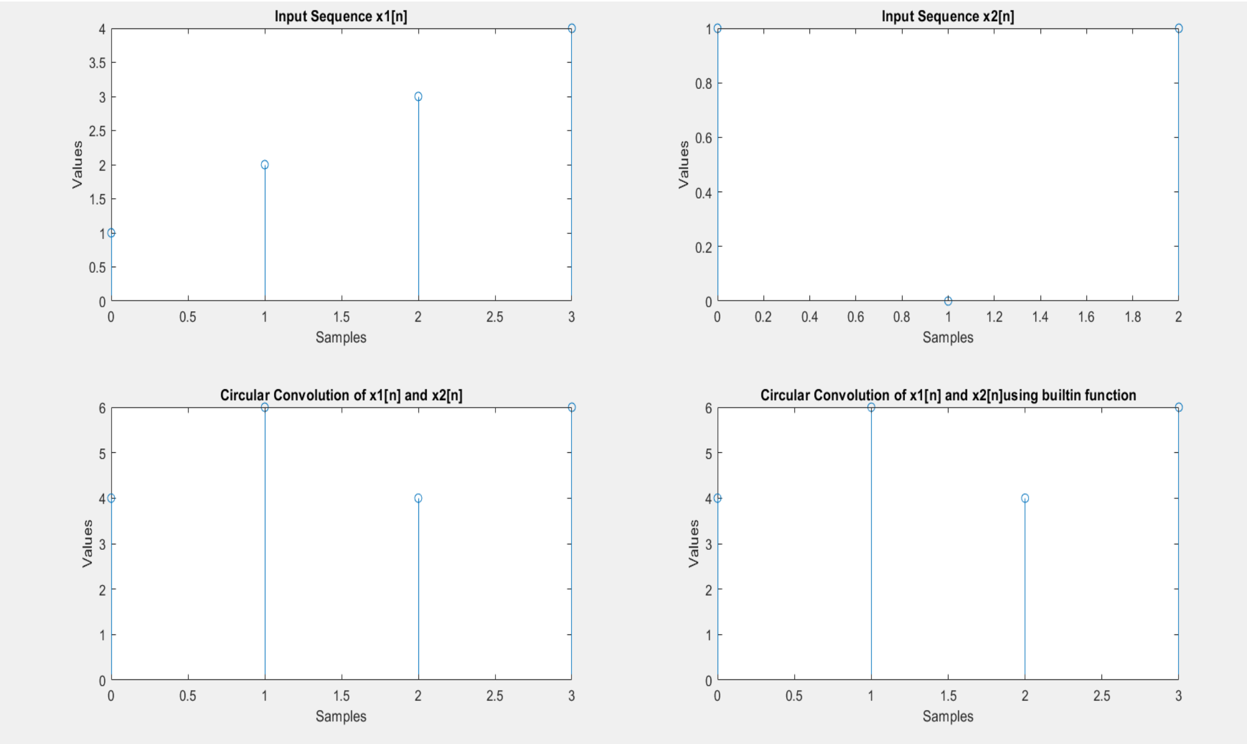
```
clear all;
close all;
x1=input('Enter the first input sequence x1[n] ');
x2=input('Enter the second input sequence x2[n] ');
Lx1=length(x1);
Lx2=length(x2);
N=max(Lx1,Lx2);
X1=fft(x1,N);
X2=fft(x2,N);
Y=X1.*X2;
```

```
y=ifft(Y,N);
disp('Circular Convolution of x1[n] and x2[n] is y[n]= ')
disp(y)

%Verification
z=cconv(x1,x2,N);
disp('Circular Convolution of x1[n] and x2[n] using
builtin function is z[n]= ')
disp(z)

a=0:Lx1-1;
subplot(2,2,1)
stem(a,x1)
title('Input Sequence x1[n]')
xlabel('Samples')
ylabel('Values')

b=0:Lx2-1;
subplot(2,2,2)
stem(b,x2)
title('Input Sequence x2[n]')
xlabel('Samples')
ylabel('Values')

c=0:N-1;
subplot(2,2,3)
stem(c,y)
title('Circular Convolution of x1[n] and x2[n]')
xlabel('Samples')
ylabel('Values')

d=0:N-1;
subplot(2,2,4)
stem(c,z)
title('Circular Convolution of x1[n] and x2[n]using
builtin function')
xlabel('Samples')
ylabel('Values')
```

Result:

```
Enter the first input sequence x1[n] [1 2 3 4]
Enter the second input sequence x2[n] [1 0 1]
Circular Convolution of x1[n] and x2[n] is y[n]=
    4    6    4    6

Circular Convolution of x1[n] and x2[n] using builtin function is z[n]=
```

4 6 4 6



iv)To find Linear and Circular Convolution using FFT algorithm

Linear Convolution

```
clear all;
close all;
x1=input('Enter the first input sequence x1[n] ');
x2=input('Enter the second input sequence x2[n] ');
Lx1=length(x1);
Lx2=length(x2);
N=Lx1+Lx2-1;
X1=FFT_L(x1,N);
X2=FFT_L(x2,N);
Y=X1.*X2;
y=IFFT_L(Y,N);
disp('Linear Convolution of x1[n] & x2[n] is ')
disp(y)
```

Result

Enter the first input sequence x1[n] [1 2 3 4]
Enter the second input sequence x2[n] [1 1 1 1]
Linear Convolution of x1[n] & x2[n] is
   1.0000 - 0.0000i   3.0000 - 0.0000i   6.0000 - 0.0000i 10.0000 + 0.0000i   9.0000 + 0.0000i
7.0000 + 0.0000i   4.0000 + 0.0000i   0.0000 - 0.0000i

Circular convolution

```
clear all;
close all;
x1=input('Enter the first input sequence x1[n]');
x2=input('Enter the first input sequence x2[n]');
Lx1=length(x1);
Lx2=length(x2);
N=max(Lx1,Lx2);
if Lx1<N
 x1=[x1,zeros(N-Lx1)];
else
 x2=[x2,zeros(N-Lx2)];
end
X1=FFT_L(x1,N);
X2=FFT_L(x2,N);
Y=X1.*X2;
y=IFFT_L(Y,N);
disp('Circular Convolution of x1[n] & x2[n] is ')
disp(y)
%Verification
z=cconv(x1,x2,N);
disp('Circular Convolution of x1[n] and x2[n] using
builtin function is z[n]= ')
disp(z)
```

Result

Enter the first input sequence x1[n][1 2 3 4]
Enter the first input sequence x2[n][1 0 1]
Circular Convolution of x1[n] & x2[n] is
4 6 4 6

Circular Convolution of x1[n] and x2[n] using builtin function is z[n]=
4 6 4 6

Functions

FFT

```
function y=FFT_L(x,N)
```

```
L=length(x);
M=nextpow2(N);
R=rem(N,2);
if(R~=0)
 x=[x zeros(1,(2^M)-L)];
end
%To alter the input sequence ie x[0] x[2] x[1] x[3]
x=bitrevorder(x);
h=1;
N=2^M;
for stage=1:M
 for index=0:(2^stage):N-1
 for n=0:(h-1)
 pos=n+index+1;
 pow=(2^(M-stage)*n);
 w=exp((-i)*(2*pi)*pow/N);
 a=x(pos)+x(pos+h).*w;
 b=x(pos)-x(pos+h).*w;
 x(pos)=a;
 x(pos+h)=b;
 end
 end
 h=2*h;
end
y=x;
```

IIFT

```
function z=IFFT_L(y,N)
L=length(y);
M=nextpow2(N);
R=rem(N,2);
if(R~=0)
 y=[y zeros(1,(2^M)-L)];
end
y=bitrevorder(y);
h=1;
N=2^M;
for stage=1:M
 for index=0:(2^stage):N-1
 for n=0:(h-1)
 pos=n+index+1;
 pow=(2^(M-stage)*n);
 w=exp((i)*(2*pi)*pow/N);
 a=y(pos)+y(pos+h).*w;
 b=y(pos)-y(pos+h).*w;
```

```
 y(pos)=a;
 y(pos+h)=b;
 end
 end
 h=2*h;
end
z=y/N;
```