Experiment No: 1 Date:29/07/2024

SIMULATION OF BASIC TEST SIGNALS

Aim

To generate continuous and discrete waveforms for the following:

1. Unit Impulse Signal
2. Bipolar Pulse Signal
3. Unipolar Pulse Signal
4. Ramp Signal
5. Triangular Signal
6. Sine Signal
7. Cosine Signal
8. Exponential Signal
9. Unit Step Signal

Theory:

1.Unit Impulse Signal:

- A signal that is zero everywhere except at one point, typically at t=0 where its value is

1.

- **Mathematically $\delta(t)= \infty; t=0$ and $0; t \neq 0$**
-

2.Bipolar Pulse Signal:

- A pulse signal that alternates between positive and negative values, usually rectangular in shape. It switches between two constant levels (e.g., -1 and 1) for a defined duration.
- **Mathematically $p(t) = A$ for $|t| \leq \tau/2$, $p(t) = 0$ otherwise**

3. Unipolar Pulse Signal:

- A pulse signal that alternates between zero and a positive value. It remains at zero for a specified duration and then jumps to a positive constant level (e.g., 0 and 1).
- **Mathematically $p(t) = A$ for $|t| \leq \tau/2$, $p(t) = 0$ otherwise (assuming A is positive)**

4. Ramp Signal:

- A signal that increases linearly with time.
- **Mathematically** $r(t) = t; t \geq 0$ and $0; t < 0$

5. Triangular Signal:

- A periodic signal that forms a triangle shape, linearly increasing and decreasing with time, typically between a positive and negative peak.
- **Mathematically: $\Lambda(t) = 1 - |t|$ for $|t| \leq 1$, $\Lambda(t) = 0$ otherwise**

6. Sine Signal:

- A continuous periodic signal. It oscillates smoothly between -1 and 1.
- **Mathematically: $y(t)=A\sin(2\pi ft)$**

7. Cosine Signal:

- A continuous periodic signal like the sine wave but phase-shifted by $\pi\backslash 2$.
- **Mathematically: $y(t)=A\cos(2\pi ft)$**

8. Exponential Signal:

- A signal that increases or decreases exponentially with time. The rate of growth or decay is determined by the constant a.
- **Mathematically: e^(at)**

9. Unit Step Signal:

- A signal that is zero for all negative time values and one for positive time values.
- **Mathematically** ***u(t)*** **=1**;$t \geq 0$ and**0**;$t<0$

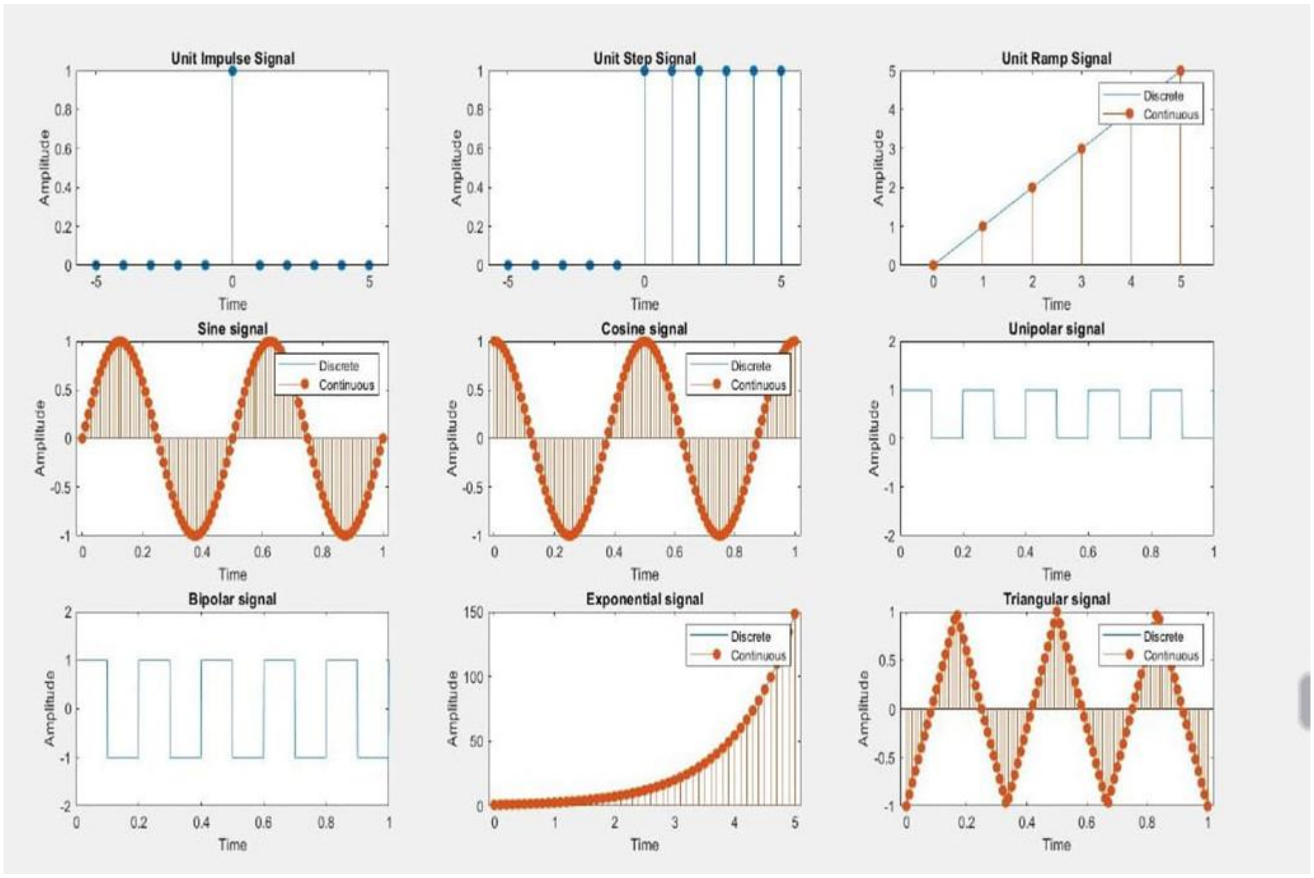
Program:

```
%unit impulse signal

clc;

close all;

t1=-5:1:5;

y1=[zeros(1,5),ones(1,1),zeros(1,5)];

subplot(3,3,1);

stem(t1,y1,'filled');

xlabel("Time");

ylabel("Amplitude");

title("Unit Impulse Signal");

%unit step signal

t2=-5:1:5;

y2=[zeros(1,5),ones(1,6)];

subplot(3,3,2);

stem(t2,y2,'filled');

xlabel("Time");

ylabel("Amplitude");

title("Unit Step Signal");

%unit ramp signal

t3=0:1:5;

y3=t3;

subplot(3,3,3);

plot(t3,y3);

hold on;

stem(t3,y3,'filled');
```

```
xlabel("Time");
ylabel("Amplitude");
title("Unit Ramp Signal");
legend("Discrete","Continuous");
%sine
t4=0:0.01:1;
f4=2;
y4=sin(2*pi*f4*t4);
subplot(3,3,4);
plot(t4,y4);
hold on;
stem(t4,y4,'filled');
xlabel("Time");
ylabel("Amplitude");
title("Sine signal");
legend("Discrete","Continuous");
%cosine signal
t5=0:0.01:1;
f5=2;
y5=cos(2*pi*f5*t5);
subplot(3,3,5);
plot(t5,y5);
hold on;
stem(t5,y5,'filled');
xlabel("Time");
ylabel("Amplitude");
title("Cosine signal");
legend("Discrete","Continuous");
%unipolar signal
```

```
f6=5;
t6=0:0.001:1;
y6=sqrt(square(2*pi*f6*t6));
subplot(3,3,6);
plot(t6,y6);
xlabel("Time");
ylabel("Amplitude");
title("Unipolar signal");
ylim([-2,2]);
%bipolar signal
f7=5;
t7=0:0.001:1;
y7=square(2*pi*f7*t7);
subplot(3,3,7);
plot(t7,y7);
xlabel("Time");
ylabel("Amplitude");
title("Bipolar signal");
ylim([-2,2]);
%exponential signal
t8=0:0.1:5;
y8=exp(t8);
subplot(3,3,8);
plot(t8,y8);
hold on;
stem(t8,y8,'filled');
xlabel("Time");
ylabel("Amplitude");
title("Exponential signal");
legend("Discrete","Continuous");
```

Observation:



```
%triangular signal

t9=0:0.01:1;

f9=3;

y9=sawtooth(2*pi*f9*t9,0.5);

subplot(3,3,9);

plot(t9,y9);

hold on;

stem(t9,y9,'filled');

xlabel("Time");

ylabel("Amplitude");

title("Triangular signal");

legend("Discrete","Continuous");
```

Result:

Generated and verified various continuous and discrete waveforms of basic test signals.

Experiment No: **2.** Date:06/08/2024

Verification of Sampling Theorem

Aim:

To verify Sampling Theorem.

Theory:

The Sampling Theorem, also known as the Nyquist-Shannon Sampling Theorem, states that a continuous signal can be completely reconstructed from its samples if the sampling frequency is greater than twice the highest frequency present in the signal. This critical frequency is known as the Nyquist rate.

fs>=2fmax

where

- fs is the sampling frequency (rate at which the signal is sampled),
- fmax is the highest frequency present in the signal.

Applications:

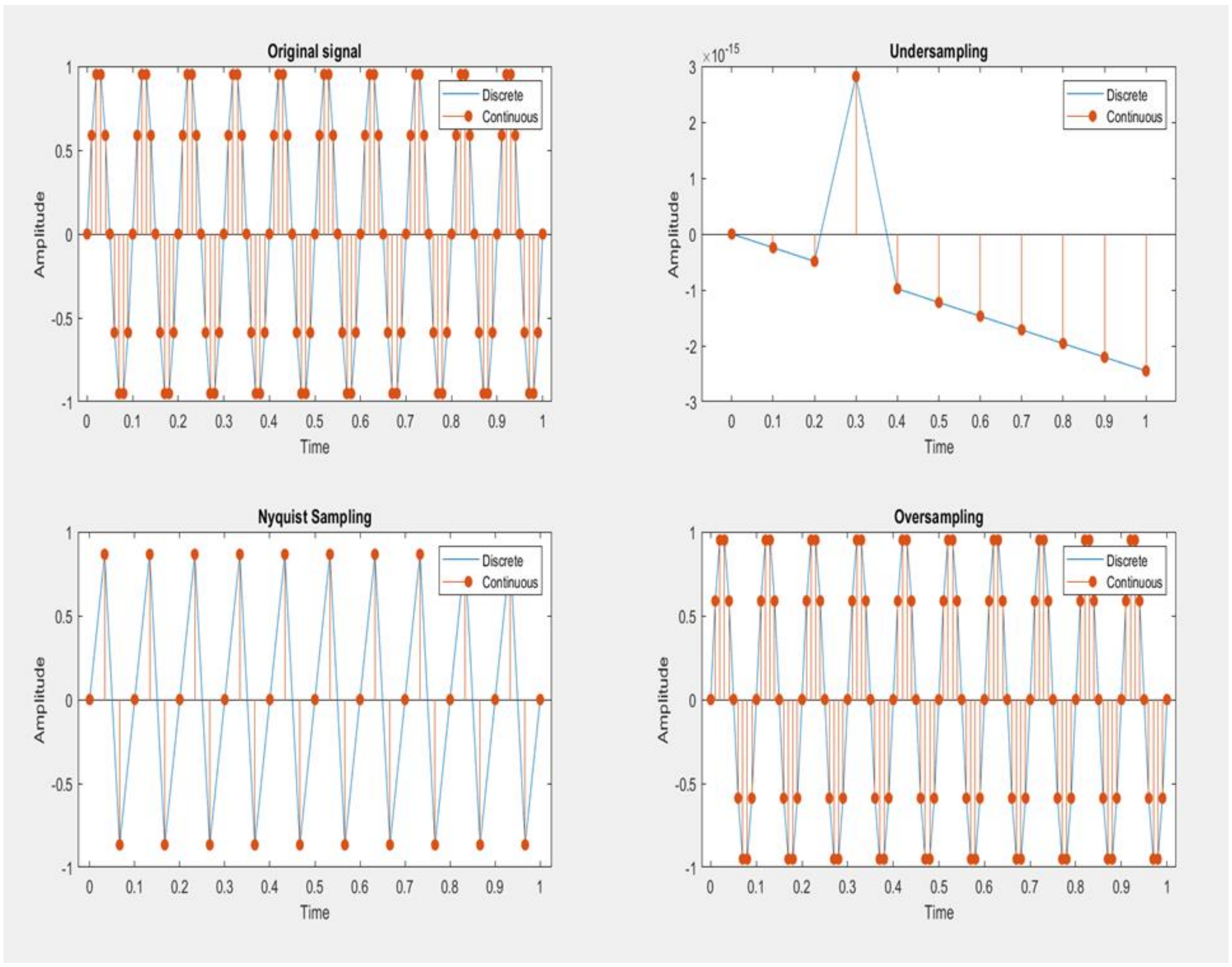
- Digital audio and video processing
- Communication systems
- Image processing
- Medical imaging

Program:

```
clc;

clear all;

close all;

t=0:0.01:1;

fm=10;

y=sin(2*pi*fm*t);
```

```
subplot(2,2,1);
plot(t,y);
hold on;
stem(t,y,"filled");
xlabel("Time");
ylabel("Amplitude");
title("Original signal");
legend("Discrete","Continuous");
%undersampling
fs1=fm;
t1=0:1/fs1:1;
y1=sin(2*pi*fm*t1);
subplot(2,2,2);
plot(t1,y1);
hold on;
stem(t1,y1,"filled");
xlabel("Time");
ylabel("Amplitude");
title("Undersampling");
legend("Discrete","Continuous");
%nyquist sampling
fs2=3*fm;
t2=0:1/fs2:1;
y2=sin(2*pi*fm*t2);
subplot(2,2,3);
plot(t2,y2);
hold on;
stem(t2,y2,"filled");
xlabel("Time");
```

Observation:



```
ylabel("Amplitude");
title("Nyquist Sampling");
legend("Discrete","Continuous");
%oversampling
fs3=10*fm;
t3=0:1/fs3:1;
y3=sin(2*pi*fm*t3);
subplot(2,2,4);
plot(t3,y3);
hold on;
stem(t3,y3,"filled");
xlabel("Time");
ylabel("Amplitude");
title("Oversampling");
legend("Discrete","Continuous");
```

Result:

Verified Sampling Theorem using MATLAB.

Experiment No: **3.** Date:10/08/24

Linear Convolution

Aim:

To find linear convolution of following sequences with and without built in function.

Theory:

Linear convolution is a mathematical operation used to combine two signals to produce a third signal. It's a fundamental operation in signal processing and systems theory.

Mathematical Definition:

Given two signals, *x(t)* and *h(t)*, their linear convolution is defined as:

y(t) = x(t) * h(t) = ∫**x**(**τ**)**h**(**t**−**τ**)

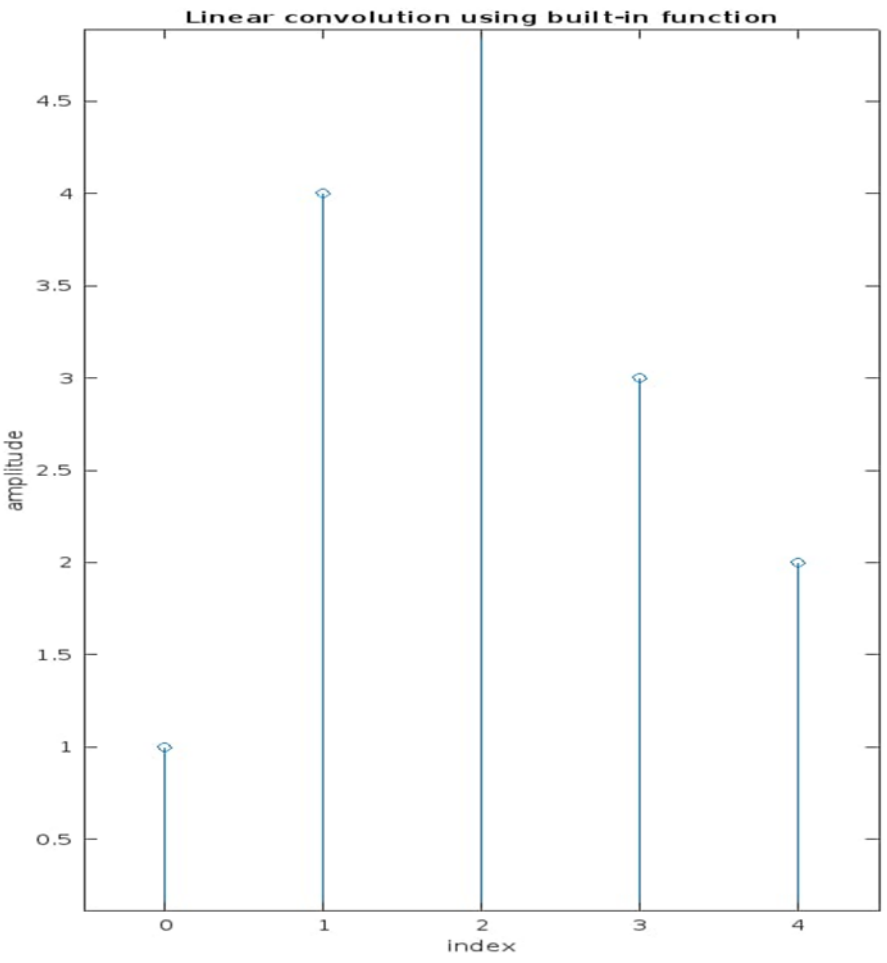
dτ∞−∞ Applications:

- **Filtering:** Convolution is used to filter signals, removing unwanted frequencies or noise.
- **System Analysis:** The impulse response of a system completely characterizes its behaviour, and convolution can be used to determine the output of the system given a known input.
- **Image Processing:** Convolution is used for tasks like edge detection, blurring, and sharpening images.

Program:

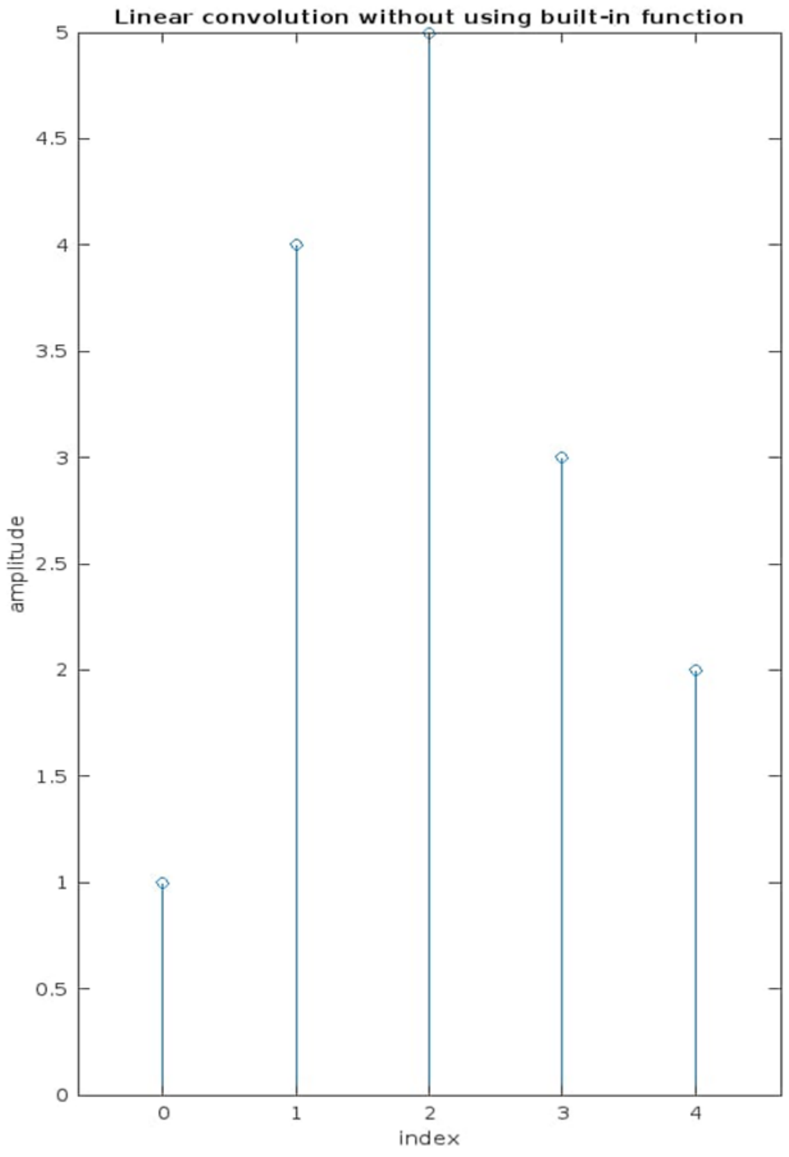
```
clc;

close all;

x=input("enter input");

x_index=input("enter index of x");

h=input("enter impulse response");

h_index=input("enter index of h");

y_index=min(x_index)+min(h_index):max(x_index)+max(h_
index);

y=conv(x,h);

disp(y);
```

Observation:



```
subplot(1,2,1);
stem(y_index,y);
xlabel("index");
ylabel("amplitude");
title("Linear convolution using built-in function");
.
```

Observation:



```
% without using built in

clc;

close all;

x=input("enter input");

x_index=input("enter index of x");

h=input("enter impulse response");

h_index=input("enter index of h");

y_index=min(x_index)+min(h_index):max(x_index)+max(h_
index);

n=length(x);

m=length(h);

len_y=length(y);

y=zeros(1,len_y);

for i=1:n

    for j=1:m

        y(i+j-1)=y(i+j-1)+x(i)*h(i);

    end

end

disp("Result:")

disp(y)

stem(y_index,y);

xlabel("index");

ylabel("amplitude");

title("Linear convolution without using built-in
function");
```

Result:

Performed Linear Convolution using with and without built-in function.

Experiment No: **4.** Date:03/09/24

Circular Convolution

Aim

To perform circular convolution using

a)Using FFT and IFFt

b)Using Concentric Circle Method

c)Using Matrix Multiplication

Theory

Circular convolution is a mathematical operation that is like linear convolution but is performed in a periodic or circular manner. This is particularly useful in discrete-time signal processing where signals are often represented as periodic sequences.

Mathematical Definition:

Given two periodic sequences *x[n]* and *h[n]*, their circular convolution is defined as:

y[n] = (x[n] ⊛ h[n]) = Σ_{k=0} ^{N-1} x[k]h[(n-

k) mod N] Applications:

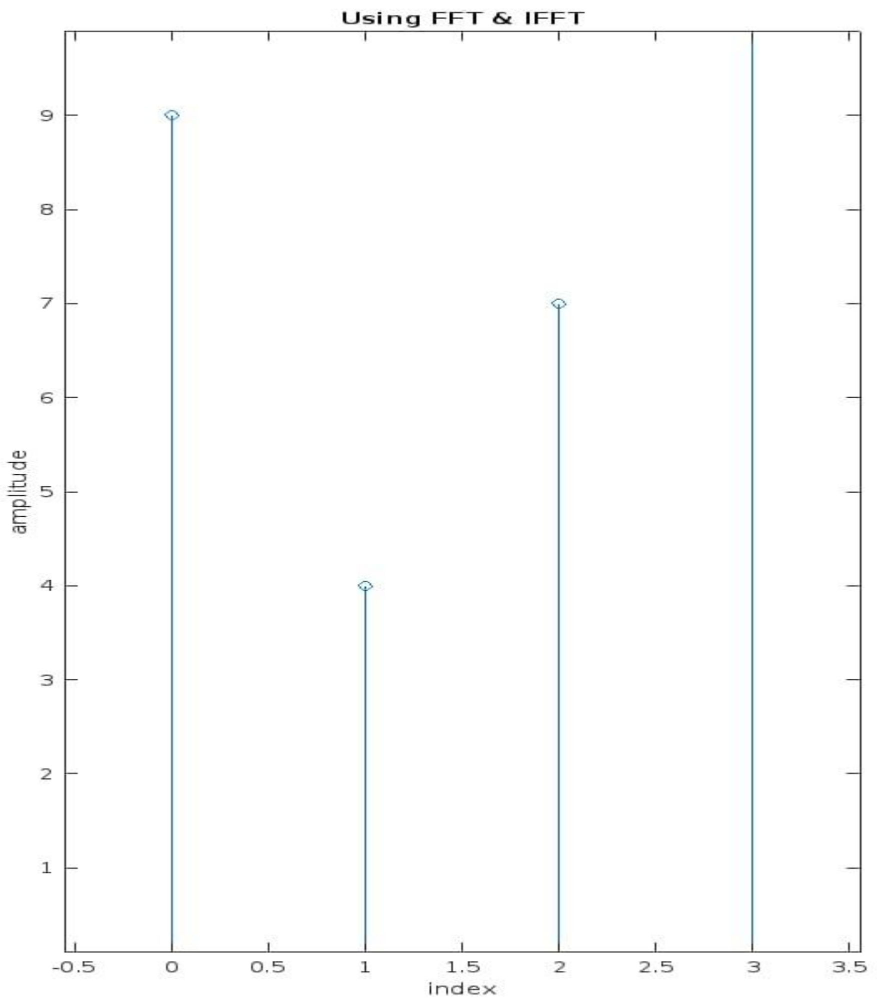
- Discrete-Time Filtering: Circular convolution is used for filtering discrete-time signals.
- Digital Signal Processing: It's a fundamental operation in many digital signal processing algorithms.
- Cyclic Convolution: In certain applications, such as cyclic prefix OFDM, circular convolution is used to simplify the implementation of linear convolution.

Program:

```
Using fft and ifft:

%circular convolution using FFT and IFFT

clc;
```

Observation:

```
enter first sequence
[1 2 3 4]
enter second sequence
[1 2]
     9     4     7    10
```



```
close all;

x=input("enter first sequence");

h=input("enter second sequence");

n=length(x);

m=length(h);

l=max(n,m);

xn=[x zeros(1,l-n)];

hn=[h zeros(1,l-m)];

n=fft(xn);

m=fft(hn);

yl=n.*m;

y=ifft(yl);

y_index=0:l-1;

disp(y);

stem(y_index,y);

xlabel("index");

ylabel("amplitude");

title("Using FFT & IFFT");
```

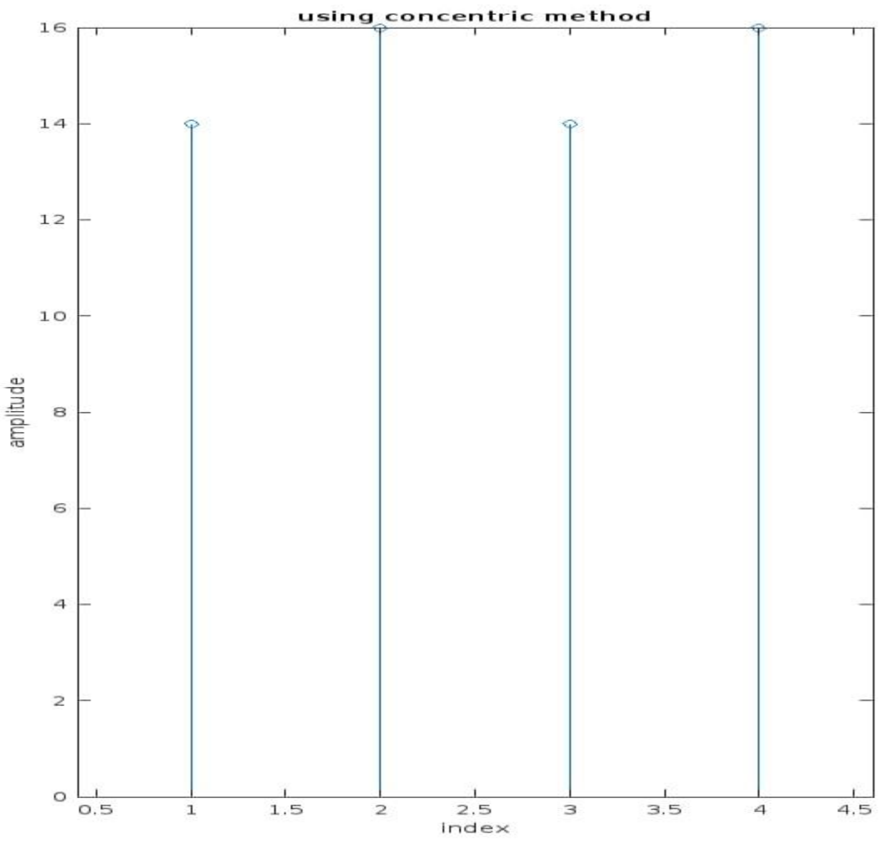
Observation:

Reversed x

```
    1    2    1    2
```

Convolution product y:

```
            14   16   14   16
```



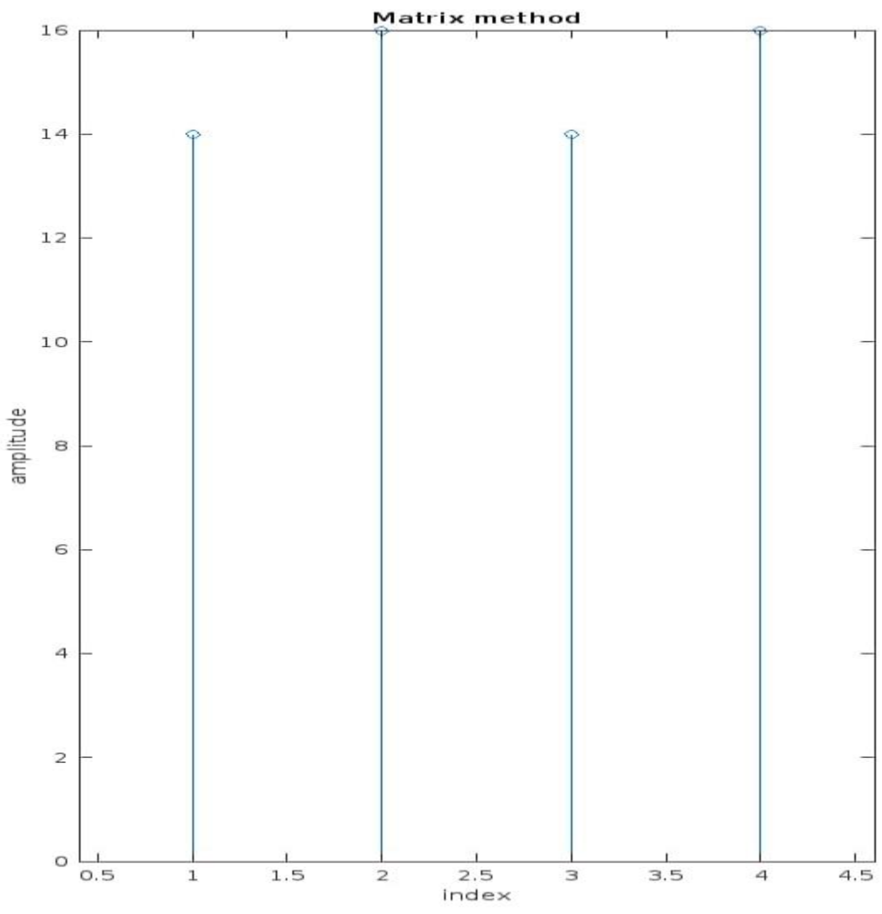
Using concentric method:

```
%Circular convolution using concentric circle method

clc;

close all;

clear all;

x=[2 1 2 1];

x=x(:,end:-1:1);

disp("Reversed x");

disp(x);

h=[1 2 3 4];

for i=1:length(x)

    x=[x(end) x(1:end-1)];

    y(i)=sum(x.*h);

end

disp("Convolution product y:");

disp(y);
```

Observation:

Convolution product y:

```
    14

    16

    14
```



using matrix method:

```
%Circular convolution using matrix multiplication

clc;

close all;

clear all;

xn=[2 1 2 1];

hn=[1 2 3 4];

h=[];

hn=hn(:,end:-1:1);

for i=1:length(hn)

    hn=[hn(end) hn(1:end-1)];

    h=[h;hn];

end

y=h*xn';

disp("Convolution product y:")

disp(y);
```

Result:

Performed Circular Convolution using a) FFT and IFFT; b) Concentric Circle method; c) Matrix method and verified result.

Experiment No: **5.** Date:10/09/24

Linear Convolution using Circular Convolution and Vice versa

Aim

To perform Linear Convolution Using Circular Convolution and Circular convolution using Linear Convolution

Theory:

Performing Linear Convolution Using Circular Convolution

Method:

- Zero-Padding: Pad both sequences *x[n]* and *h[n]* with zeros to a length of at least *2N-1*, where *N* is the maximum length of the two sequences. This ensures that the circular convolution will not wrap around and introduce artificial periodicity.
- Circular Convolution: Perform circular convolution on the zero-padded sequences.
- Truncation: Truncate the result of the circular convolution to the length *N1 + N2 - 1*, where *N1* and *N2* are the lengths of the original sequences *x[n]* and *h[n]*, respectively.

Example:

- Consider the sequences *x[n] = [1, 2, 3]* and *h[n] = [4, 5]*.
- Zero-padding: Pad *x[n]* to [1, 2, 3, 0, 0] and *h[n]* to [4, 5, 0, 0].
- Circular Convolution: Perform circular convolution on the zero-padded sequences. The result will be [4, 13, 21, 15, 0].
- Truncation: Truncate the result to [4, 13, 21, 15].

This result is the same as the linear convolution of *x[n]* and *h[n]*.

Program:

```
 clc;

clear all;

close all;

x=[1 2 3 4];
```

Observation:

Linear Convolution using Circular Convolution output

```
     1     3     6     9     7     4
```

Verification

```
     1     3     6     9     7     4
```

```
h=[1 1 1];

l_x=length(x);

l_h=length(h);

k=l_x+l_h-1;

h=[h,zeros(1,k-l_h)];

x=[x,zeros(1,k-l_x)];

X_K=fft(x);

H_K=fft(h);

Y_K=X_K.*H_K;

y=ifft(Y_K);

disp('Linear Convolution using Circular Convolution
output')

disp(y);


%verification

x=[1 2 3 4];

h=[1 1 1];

y=conv(x,h);

disp('Verification');

disp(y);

.
```

Circular convolution using linear convolution:

```
clc;

clear all;

close all;

x = [1 2 3 4];

h = [1 1 1];

y = conv(x, h);

l = length(x);

m = length(h);

for i = 1:m-1

    y(i) = y(l + i) + y(i);

end

res = y(1:l);

disp('Circular convolution using Linear Convolution
output');

disp(res);


% Verification using FFT-based circular convolution

x = [1 2 3 4];

h = [1 1 1];

l = length(x);

m = length(h);


% Pad h with zeros to make it the same length as x

h = [h, zeros(1, l - m)];

X_K = fft(x);
```

Observation:

Circular convolution using Linear Convolution output

```
    8     7     6     9
```

Verification (Circular Convolution using FFT)

```
    8     7     6     9
```

```
H_K = fft(h);

Y_K = X_K .* H_K;

y_fft = ifft(Y_K);

disp('Verification (Circular Convolution using FFT)');

disp(y_fft);
```

RESULT:

Performed a) Linear Convolution using Circular Convolution; b) Circular Convolution using Linear Convolution and verified result.

Experiment No:6 Date: 24/09/24

DFT and IDFT

Aim

To compute DFT and IDFT of a signal using inbuilt functions and manual methods.

Theory:

The Discrete Fourier Transform (DFT) is a fundamental mathematical tool used in signal processing, communication systems, and many areas of engineering and science. It converts a discrete sequence (signal) from the time domain into its representation in the frequency domain. The DFT transforms a finite sequence of equally spaced samples of a function into a sequence of coefficients of complex sinusoids, ordered by their frequencies.

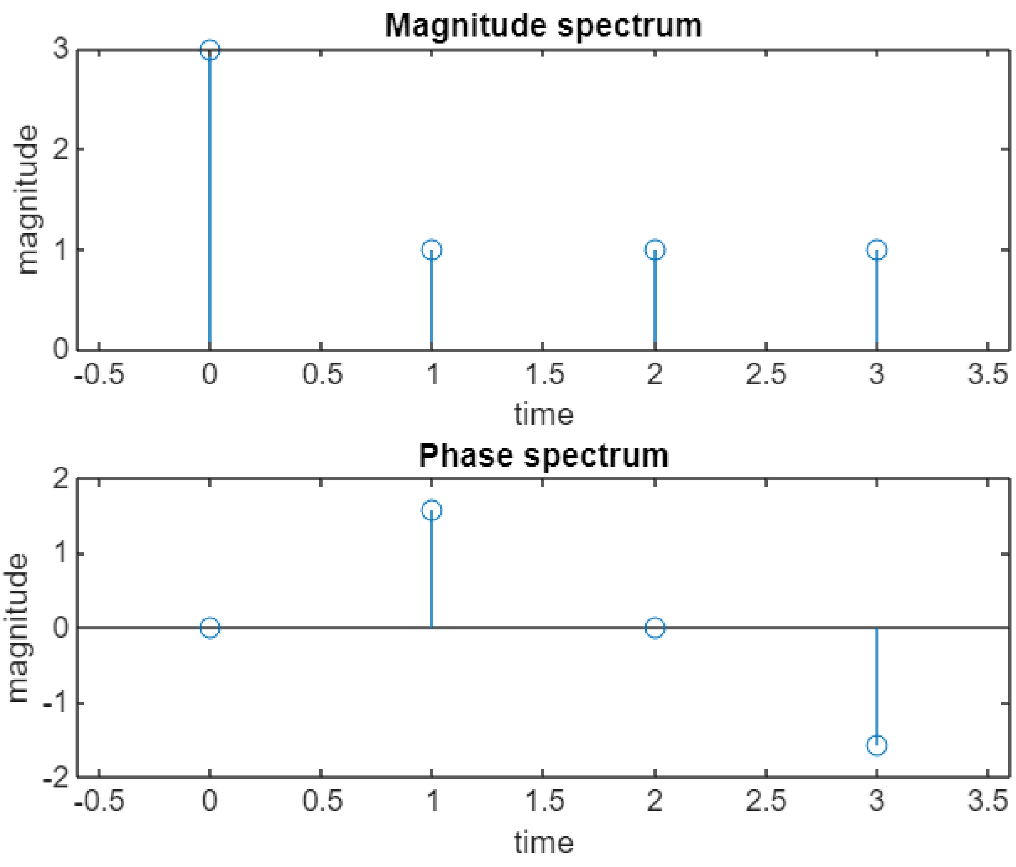
The Inverse Discrete Fourier Transform (IDFT) is a mathematical process that converts a sequence of complex numbers in the frequency domain back into the time domain. It is the inverse operation of the Discrete Fourier Transform (DFT) and is used to recover the original time-domain sequence from its DFT.

Program:

```
%dft using inbuilt and manual methods
clc;
clear all;
close all;
x=input('Enter the sequence:');
N=input('enter the N point DFT: ');
l=length(x);
x=[x zeros(1,N-1)];
X=zeros(N,1);
for k=0:N-1
    for n=0:N-1
        X(k+1)=X(k+1)+x(n+1)*exp(-1j*2*pi*n*(k/N));
    end
end
disp('X');
disp(X);
```

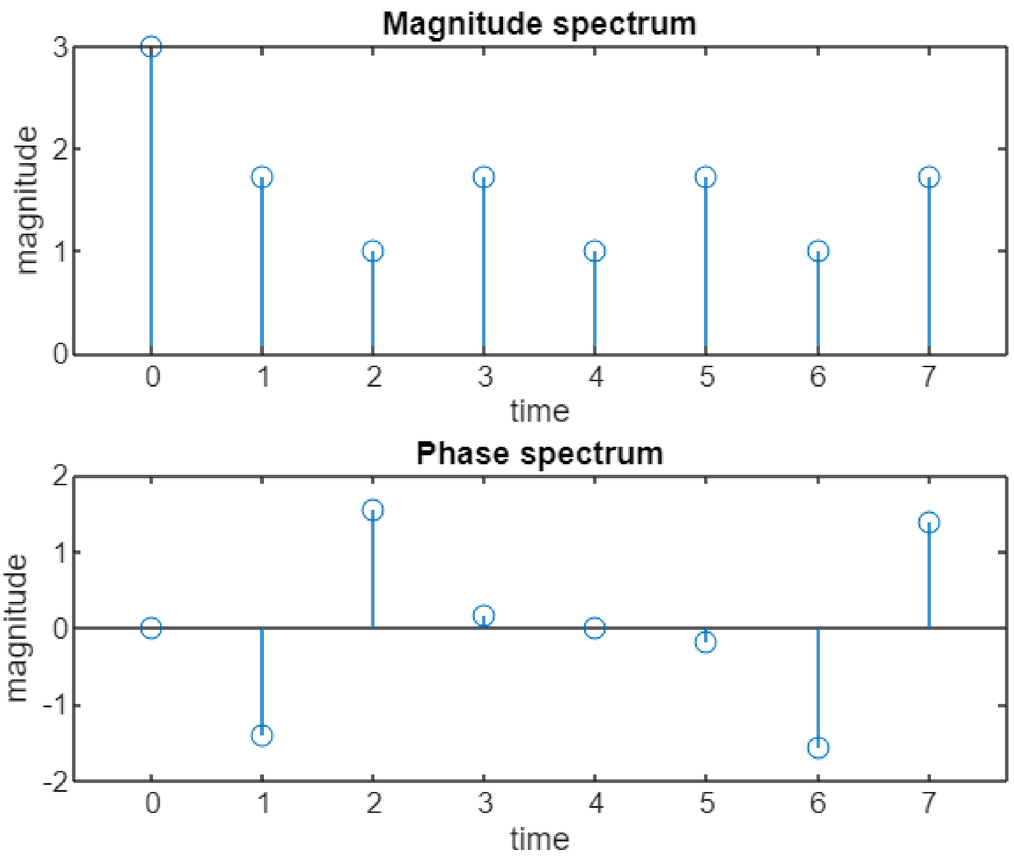
Observation:

```
Enter the sequence:[1 0 1 1]

enter the N point DFT: 4

X
   3.0000 + 0.0000i
  -0.0000 + 1.0000i
   1.0000 - 0.0000i
   0.0000 - 1.0000i
DFT
3.0000 + 0.0000i   0.0000 + 1.0000i   1.0000 + 0.0000i   0.0000 - 1.0000i
```

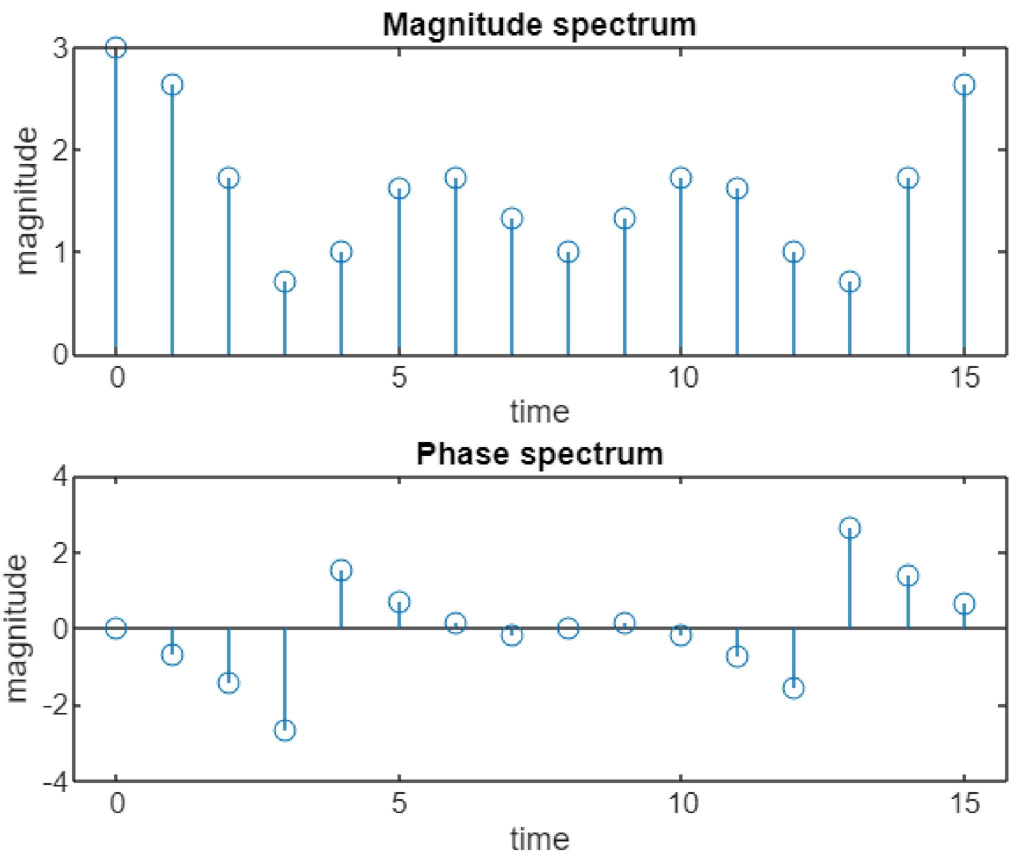


```
disp("DFT");
disp(fft(x,N));

%magnitude spectrum

k=0:N-1;

mag=abs(X);

subplot(2,1,1);

stem(k,mag);

xlabel('time');

ylabel('magnitude');

title('Magnitude spectrum');

%phase spectrum

phase=angle(X);

subplot(2,1,2);

stem(k,phase);

xlabel('time');

ylabel('magnitude');

title('Phase spectrum');
```

```
Enter the sequence:[1 0 1 1]

Enter the N point DFT: 8

X

   3.0000 + 0.0000i

   0.2929 - 1.7071i

  -0.0000 + 1.0000i

   1.7071 + 0.2929i

   1.0000 - 0.0000i

   1.7071 - 0.2929i

   0.0000 - 1.0000i

   0.2929 + 1.7071i

DFT

   3.0000 + 0.0000i   0.2929 - 1.7071i   0.0000 + 1.0000i   1.7071 + 0.2929i   1.0000 +
0.0000i   1.7071 - 0.2929i   0.0000 - 1.0000i   0.2929 + 1.7071i
```



```
Enter the sequence:[1 0 1 1]

enter the N point DFT: 16

X
   3.0000 + 0.0000i
   2.0898 - 1.6310i
   0.2929 - 1.7071i
  -0.6310 - 0.3244i
  -0.0000 + 1.0000i
   1.2168 + 1.0898i
   1.7071 + 0.2929i
   1.3244 - 0.2168i
   1.0000 - 0.0000i
   1.3244 + 0.2168i
   1.7071 - 0.2929i
   1.2168 - 1.0898i
   0.0000 - 1.0000i
  -0.6310 + 0.3244i
   0.2929 + 1.7071i
   2.0898 + 1.6310i
DFT
   3.0000 + 0.0000i   2.0898 - 1.6310i   0.2929 - 1.7071i  -0.6310 - 0.3244i   0.0000 + 1.0000i
1.2168 + 1.0898i   1.7071 + 0.2929i   1.3244 - 0.2168i   1.0000 + 0.0000i   1.3244 + 0.2168i
1.7071 - 0.2929i   1.2168 - 1.0898i   0.0000 - 1.0000i  -0.6310 + 0.3244i   0.2929 + 1.7071i
2.0898 + 1.6310i
```



Observation:

```
Enter the sequence:[1 1 1 0]
Enter the N point of DFT:4
x
   0.7500 + 0.0000i
   0.0000 + 0.2500i
   0.2500 - 0.0000i
  -0.0000 - 0.2500i
IDFT
   0.7500 + 0.0000i   0.0000 + 0.2500i   0.2500 + 0.0000i   0.0000 - 0.2500i
```

```
%idft using inbuilt and manual functions
clc;
clear all;
close all;
X=input('Enter the sequence:');
N=input('Enter the N point of DFT:');
l=length(X);
X=[X zeros(1,N-l)];
x=zeros(N,1);
for k=0:N-1
    for n=0:N-1
        x(n+1)= x(n+1)+X(k+1)*exp(1j*2*pi*n*k/N);
    end
end
x=(1/N).*x;
disp('x');
disp(x);
disp('IDFT');
disp(ifft(X,N));
```

<u>Observation:</u>

```
Enter the input sequence: [1 0 1 1]
DFT of the input sequence (using Twiddle Factor Matrix):
   3.0000 + 0.0000i
  -0.0000 + 1.0000i
   1.0000 - 0.0000i
   0.0000 - 1.0000i
```

```
% DFT using twiddle factor matrix
x = input('Enter the input sequence: ');
N = length(x);
W    = exp(-1i*2*pi*(0:N-1)'*(0:N-1)/N);
X    = W * x(:);
disp('DFT of the input sequence (using Twiddle Factor Matrix):');
disp(X);
```

Observation:

```
Enter the input sequence: [ 1 0 1 1]
IDFT of the input sequence (using Twiddle Factor Matrix):
   0.7500 + 0.0000i
  -0.0000 - 0.2500i
   0.2500 + 0.0000i
   0.0000 + 0.2500i
```

```
% IDFT using twiddle factor matrix
x = input('Enter the input sequence: ');
N = length(x);
W = exp(1i*2*pi*(0:N-1)'*(0:N-1)/N);
X_idft = (1/N) * (W * x(:));
disp('IDFT of the input sequence (using Twiddle Factor Matrix):');
disp(X_idft);
```

Result:

Computed DFT and IDFT using inbuilt and manual methods and Twiddle factor matrix and verified the output.

Experiment No:7 Date: 01/10/24

PROPERTIES OF DFT

Aim

To prove the following properties of DFT

- Linearity
- Convolution
- Multiplication
- Parseval’s Theorem

Theory:

Linearity:

The DFT is a linear transformation, meaning that the DFT of the sum of two signals is equal to the sum of their individual DFTs, and multiplying a signal by a constant in the time domain results in the DFT being multiplied by the same constant.If x1(n) and x2(n) are two sequences and a and b are constants then:

DFT(ax1(n)+bx2(n))= a.DFT(x1(n))+b.DFT(x2(n))

Multiplication:

The DFT of a pointwise multiplication (element-wise product) of two signals in the time domain corresponds to the circular convolution of their DFTs in the frequency domain. If x1(n) and x2(n) are two signals then:

DFT{x1(n).x2(n)}= 1/N DFT{x(n)}$\circledast$DFT{h(n)}

Convolution:

The DFT of the convolution of two sequences in the time domain is the element-wise multiplication of their DFTs in the frequency domain. If x1(n) and x2(n) are two signals, then:

DFT{x1(n)∗x2(n)}=DFT{x1(n)}·DFT{x2(n)}

Parseval’s Theorem:

Parseval’s theorem states that the total energy of a discrete-time signal (the sum of the squared magnitudes of the signal in the time domain) is equal to the total energy of its DFT (the sum of the squared magnitudes of the DFT coefficients).

Program:

```
%linearity property

clc;

clear all;

close all;

x1=input('Enter the first sequence:');
```

Observation:

```
Enter the first sequence:[1 2 3 4]

Enter the second sequence:[ 1 1 1]

LHS:

29.0000 + 0.0000i  -4.0000 + 1.0000i  -1.0000 + 0.0000i  -4.0000 – 1.0000i

RHS:

29.0000 + 0.0000i  -4.0000 + 1.0000i  -1.0000 + 0.0000i  -4.0000 – 1.0000i

LHS=RHS

Linearity Property Verified
```

```
x2=input('Enter the second sequence:');
a=2;
b=3;
l1=length(x1);
l2=length(x2);
if l1>l2
    x2=[x2 zeros(1,l1-l2)]
else
    x1=[x1 zeros(1,l2-l1)];
end
LHS=fft((a.*x1)+(b.*x2));
RHS=[a.*fft(x1)+b.*fft(x2)];
disp('LHS:');
disp(LHS);
disp('RHS:');
disp(RHS);
disp(['LHS=RHS')_
```

Observation:

```
LHS
8     7     6     9
RHS
8     7     6     9
 Convolution property verified
```

```
%Convolution property
clc;

clear all;

close all;

x=input(‘enter sequence 1’);

h=input(‘enter sequence 2’);

N=max(length(x),length(h));

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

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

X1=fft(X);

H1=fft(H);

LHS= cconv(X,H,N);

RHS=ifft(X1.*H1);

disp(LHS);

disp(RHS);

if LHS==RHS

    disp(‘Convolution property verified’);

else

    disp(‘Convolution property verified’);

end
```

Observation:

```
enter the first sequence:
[1 2 3 4]

enter the second sequence:

[1 1 1]

   6.0000 + 0.0000i  -2.0000 - 2.0000i   2.0000 + 0.0000i  -2.0000 + 2.0000i


   6.0000 + 0.0000i  -2.0000 - 2.0000i   2.0000 + 0.0000i  -2.0000 + 2.0000i
```

```
%multiplication property
clc;
clear all;
close all;
x1=input('enter the first sequence:');
x2=input('enter the second sequence:');
l1=length(x1);
l2=length(x2);
n=max(l1,l2);
x1=[x1 zeros(1,n-l1)];
x2=[x2 zeros(1,n-l2)];
lhs=fft(x1.*x2);
X1=fft(x1);
X2=fft(x2);
rhs=cconv(X1,X2,n)/n;
disp(lhs);
disp(rhs);
```

<u>Observation:</u>

```
enter the first sequence:
[1 2 3 4]
enter the second sequence:
[1 1 1]
LHS
    6
RHS
    6
```

```
%parseval's theorem
clc;
clear all;
close all;
x1=input('enter the first sequence:');
x2=input('enter the second sequence:');
l1=length(x1);
l2=length(x2);
n=max(l1,l2);
x1=[x1 zeros(1,n-l1)];
x2=[x2 zeros(1,n-l2)];
lhs=sum(x1.*conj(x2));
rhs=sum(fft(x1).*conj(fft(x2)))/n;
disp('LHS');
disp(lhs);
disp('RHS');
disp(rhs);
```

<u>Result:</u>

Verified linearity, convolution, multiplication,and parseval’s properties of DFT

Experiment No:8 Date: 08/10/24

OVERLAP SAVE AND ADD METHOD

Aim

To perform linear convolution of two sequences using overlap save and add method

Theory:

The Overlap-Add and Overlap-Save methods are efficient techniques used to perform linear convolution of long signals with finite impulse response (FIR) filters using the Fast Fourier Transform (FFT). Both methods help reduce computational complexity by breaking a long signal into smaller chunks, processing them independently in the frequency domain, and then combining the results.

The Overlap Add method splits the input signal into overlapping segments, performs convolution on each segment using the FFT, and then adds the overlapping parts to reconstruct the final output. This method is efficient for filtering long signals by using FFT-based convolution.

The Overlap Save method also divides the input signal into segments, but unlike OLA, it saves the non-overlapping parts and discards the overlapping parts of the convolution output.

Program:

```
%overlap save method
clc;
clear all;
close all;
x=input("Enter sequence 1");
h=input("Enter sequence 2");
N=input('Enter length to divide');
if N<length(h)
   disp('not possible');
else
    xl=length(x);
    hl=length(h);
    L=N-hl+1;
    hnew=[h zeros(1,N-hl)];
    xnew=[zeros(1,hl-1),x,zeros(1,N-1)];
    y=[];
for i=1:L:length(xnew)-N+1
    XB=xnew(i:i+N-1);
    YB=ifft(fft(XB).*fft(hnew));
    y=[y,YB(hl:end)];
end
    disp(y(1:xl+hl-1));
```

```
end
```

Observation:

```
Enter sequence 1[3 -1 0 1 3 2 0 1 2 1]
Enter sequence 2[1 1 1]
Enter length to divide3
final convoluted sequence
     3     2     2     0     4     6     5     3     3     4     3     1
```

Observation:

```
Enter the input sequence: [0 1 2 3 4 5 6 7 8 9]
Enter the filter sequence: [1 0 1]
Enter the segment length (choose N >= Lh): 3
final convoluted sequence:
     0     1     2     4     6     8    10    12    14    16     8     9
```

```
%overlap add method
clc;
clear all;
close all;
x = input('Enter the input sequence: ');
h = input('Enter the filter sequence: ');
Lx = length(x);
Lh = length(h);
N = input('Enter the segment length (choose N >= Lh): ');
if N < Lh
    error('Segment length N must be greater than or equal to filter
length');
end
x = [x, zeros(1, N - mod(Lx, N))];
Lx_padded = length(x);
y = zeros(1, Lx_padded + Lh - 1);
for i = 1:N:Lx_padded
    x_segment = x(i:i+N-1);
    y_segment = conv(x_segment, h);
    y(i:i+length(y_segment)-1)   =   y(i:i+length(y_segment)-1)   +
y_segment;
end
y = y(1:Lx + Lh - 1);
disp('final convoluted sequence:');
disp(y);
```

Result:

Implemented overlap add and overlap save method using MATLAB and verified the output