DIGITAL SIGNAL PROCESSING LABORATORY RECORD

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

TKM COLLEGE OF ENGINEERING

KOLLAM- 691005

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DIGITAL SIGNAL PROCESSING LABORATORY RECORD

Name:………...…………..………... Class:…………..………

Roll.No:..

Year:.....……..2019-23…………Batch:……….………….........

Practical Classes: AN of Thursday from Nov 2021 to February 2022

Certified that this is a record of the work done by,

Ms./Mr.……………………………,…………………… …………………

of……… …..semester class in the…………………………………………

Laboratory during the year………………2022…………………………

Roll No…………………………

Faculty-in-Charge
Dr. Aswathy G P

Internal Examiner   *External Examiner*

LIST OF EXPERIMENTS

Experiment 1
Simulation of Basic Signals

Aim: Plot the following signals as discrete and continuous

- Impulse
- Unit Pulse
- Parabola
- Ramp
- Bipolar Pulse
- Triangular
- Unit Step
- Cosine
- Signum

Theory:

A signal can be used to represent any variable of interest as a function of time, temperature, pressure etc. Signals carry some amount of information in them. One of the methods to classify signals is on the basis of their time characteristics as

- Continuous -time signals: The function is defined continuously in time.

- Discrete time signals: they are defined in discrete time instants only.

Both of these signals can have any amplitude (continuous in amplitude). Digital signals have discrete amplitude levels.

Important Basic Signals

- Unit Pulse

- Bipolar Pulse

- Cosine

- Triangular

- Impulse

$$\delta(t) = \begin{cases} 1 & \text{for } t = 0 \\ 0 & \text{for } t \neq 0 \end{cases}$$

- Parabola

$$p(t) = \int_{-\infty}^{t} r(t)dt = (t^2/2)U(t)$$

- Ramp

$$r(t) = \int_{-\infty}^{t} U(t)dt = tU(t)$$

- Unit Step

$$U(t) = 1(\text{ when } t \geq 0)$$

$$U(t) = 0(\text{ when } t < 0)$$

- Signum

$$\text{sgn}(t) = \begin{cases} 1 & \text{for} & t > 0 \\ -1 & \text{for} & t < 0 \end{cases}$$

i) Impulse

Program:

```
clc;clf;clear;
n=[-5:5];
x=[zeros(1,5),1,zeros(1,5)];
plot2d3(n,x,2);
```
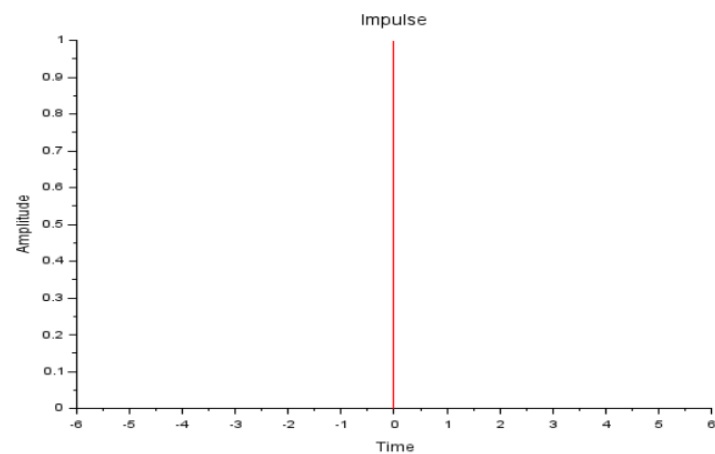
```
xlabel('Time');

ylabel('Amplitude');

title('Impulse');

a=gca()

a.x_location="origin";

a.y_location="left";
```
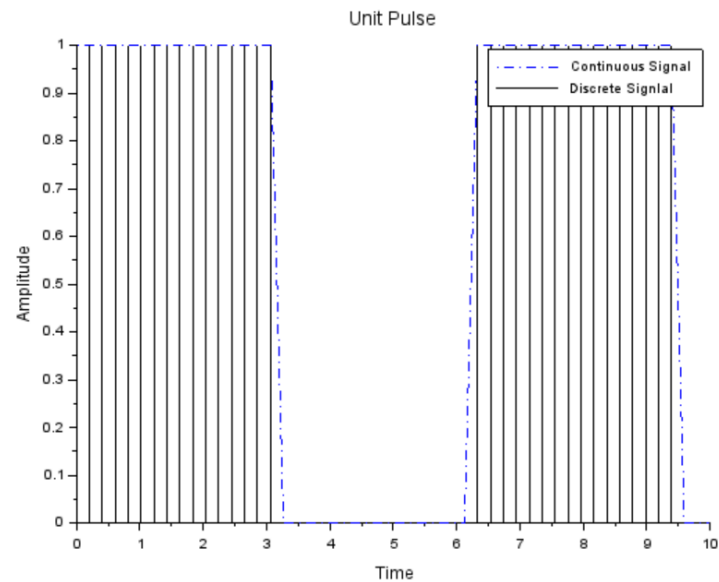
Output:

ii)Unit Pulse

program:

```
clc;clf;clear;

t = linspace(0,10,50);

a=gca()

a.x_location ="origin";

a.y_location ="origin";

unit_pulse=sqrt(squarewave(t));

plot(t,unit_pulse,'b-.');

plot2d3(t,unit_pulse);

title('Unit Pulse');

xlabel('Time');
```

ylabel('Amplitude');

legend('Continuous Signal','Discrete Signlal',opt="lr");
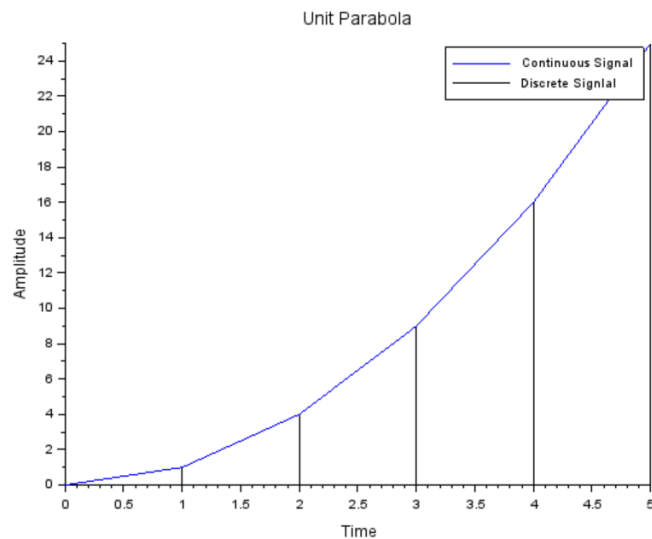
Output:

iii)Parabola

program:

```
clc;clf;clear;
t=0:5;
a=gca()
a.x_location ="origin";
a.y_location ="origin";
plot(t,t^2);
plot2d3(t,t^2);
title('Unit Parabola');
xlabel('Time');
```

ylabel('Amplitude');

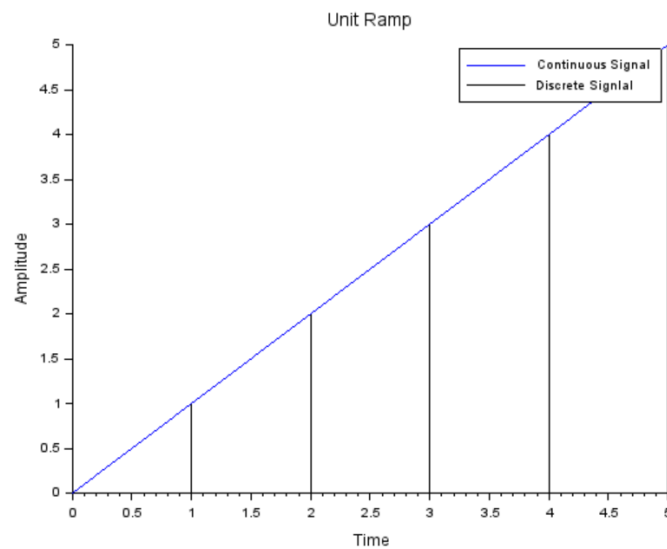legend('Continuous Signal','Discrete Signlal');

Output:

iv)Unit Ramp

program:

```
clc;clf;clear;
t=0:5;
a=gca()
a.x_location ="origin";
a.y_location ="origin";
plot(t,t);
plot2d3(t,t);
title('Unit Ramp');
xlabel('Time');
ylabel('Amplitude');
legend('Continuous Signal','Discrete Signlal');
```
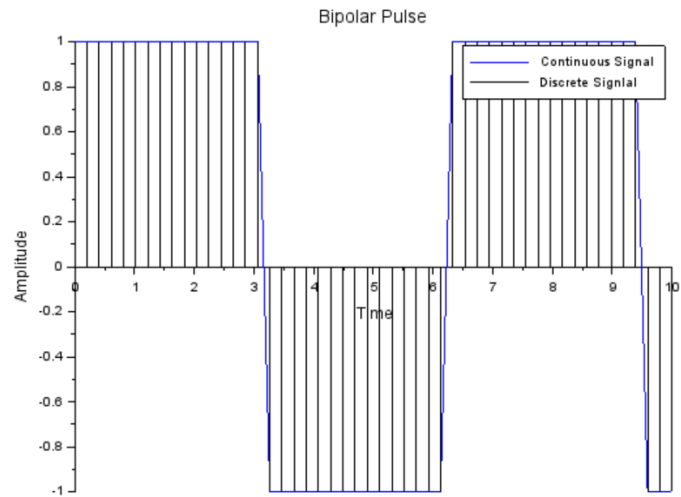
Output:

Unit Ramp

v)Bipolar Pulse

program:

```
clc;clf;clear;
t = linspace(0,10,50);
a=gca()
a.x_location ="origin";
a.y_location ="origin";
bipolar_pulse=squarewave(t);
plot(t,bipolar_pulse);
plot2d3(t,bipolar_pulse);
title('Bipolar Pulse');
xlabel('Time');
ylabel('Amplitude');
legend('Continuous Signal','Discrete Signlal');
```
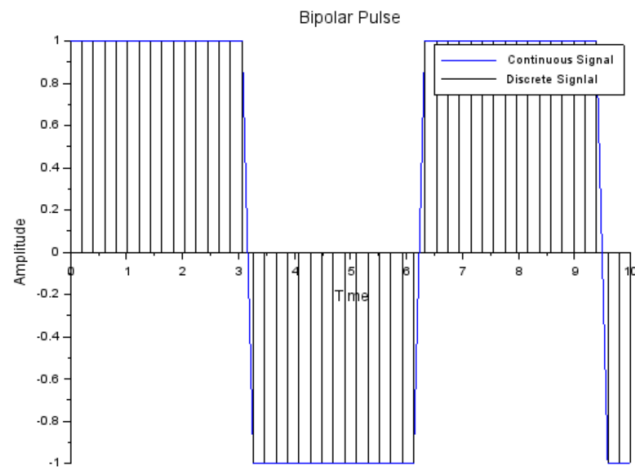
Output:

Bipolar Pulse

vi)Triangular

program:

```
clc;clf;clear;
t =0:( %pi /4) :(4* %pi ) ;
y = sin ( 2*t ) ;
a=gca()
a.x_location ="origin";
a.y_location ="origin";
plot (t ,y);
plot2d3(t,y,-9);
title('Triangular Wave');
xlabel('Time');
ylabel('Amplitude');
legend('Continuous Signal','Discrete Signlal');
```
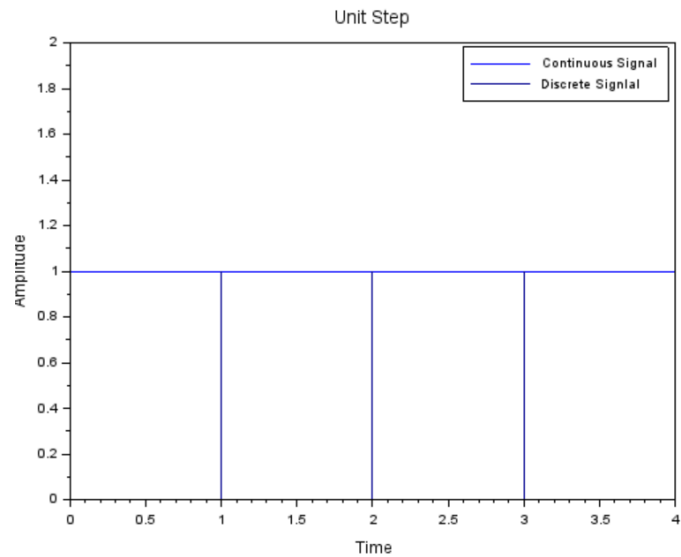
Output:

Bipolar Pulse

vii)Unit Step

program:

```
clc;clf;clear;
t=0:4;
y=ones(1,5);
plot (t ,y);
plot2d3 (t,y,9);
title('Unit Step');
xlabel('Time');
ylabel('Amplitude');
legend('Continuous Signal','Discrete Signlal');
```

Output:

Unit Step

viii)Cosine

program:

```
clc;clf;clear;
f =0.2;
t =0:0.1:10;
x = cos (2* %pi * t * f ) ;
a=gca()
a.x_location ="origin";
a.y_location ="origin";
plot (t ,x,'b-.');
plot2d3 (t ,x );
title('Cosine Signal');
xlabel('Time');
ylabel('Amplitude');
legend('Continuous Signal','Discrete Signlal');
```

Output:

Cosine Signal

ix)Signum

program:

```
clc;clf;clear;
t = -5:0.1:5
x = sign ( t ) ;
a=gca()
a.x_location ="origin";
a.y_location ="origin";
plot (t ,x);
plot2d3(t,x);
title ('Signum') ;
xlabel('Time');
ylabel('Amplitude');
legend('Continuous Signal','Discrete Signlal');
```
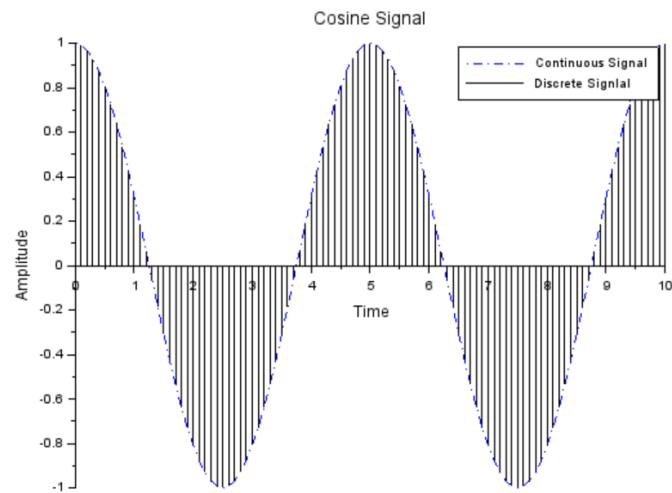
Output:

Signum
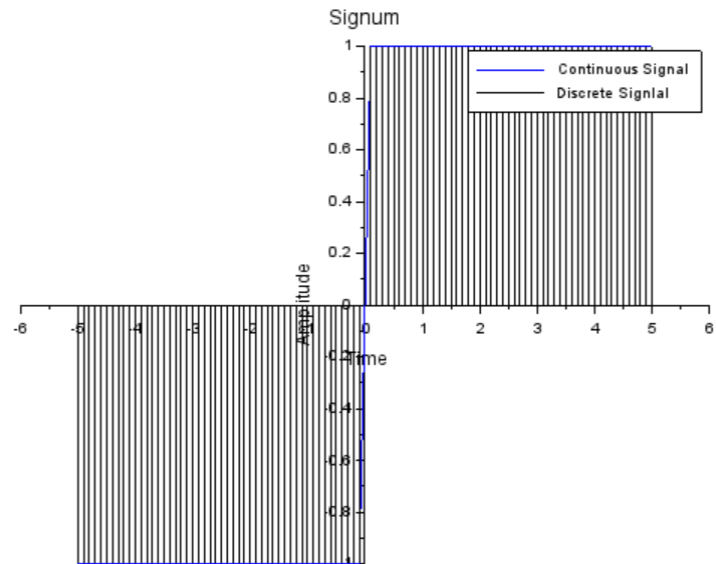
x)y=sint+(⅓)sin(t/3)+(1/5)sin(t/5)+(1/7)sin(t/7)+(1/9)sin(t/9)

program:

```
clc;clf;clear;
t=[0:0.001:10]';
x=sin(t);
b=(1/3)*sin(t*3);
c=(1/5)*sin(t*5);
d=(1/7)*sin(t*7);
e=(1/9)*sin(t*9);
y=x+b+c+d+e;
xlabel("sinx");
ylabel("t")
plot(t,y);
xlabel('Time');
ylabel('Amplitude');
legend("Given Signal");
```

Output:
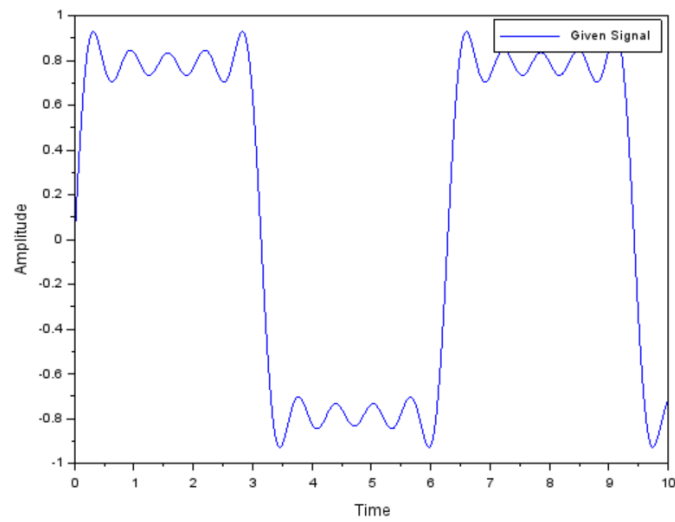
Result:

Basic signals like Unit impulse,Unit Ramp,Bipolar pulse,Triangular Pulse are plotted using scilab .

Experiment 2
Verification of Sampling Theorem

Aim: To verify the sampling theorem in Scilab

Theory:

Statement: A continuous-time signal can be represented in 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. i. e.

$$fs \geq 2fm.$$

Proof: Consider a continuous time signal x(t). The spectrum of x(t) is a band limited to fm Hz i.e. the spectrum of x(t) is zero for $|\omega| > \omega m$.
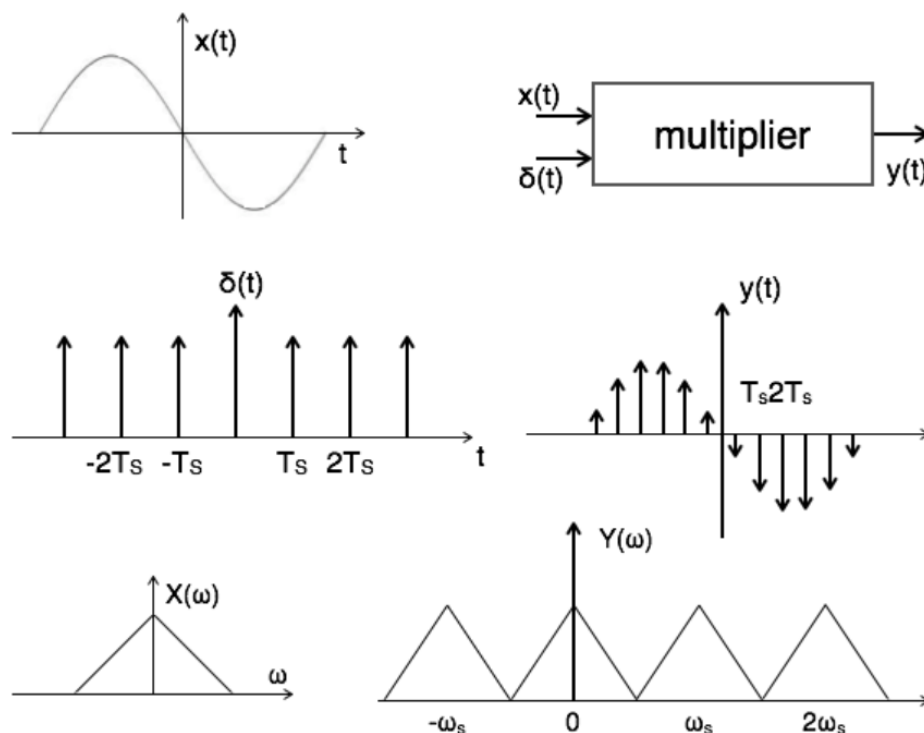
Sampling of input signal x(t) can be obtained by multiplying x(t) with an impulse train δ(t) of period Ts. The output of multiplier is a discrete signal called sampled signalwhich is represented with y(t) in the following diagrams:

To reconstruct x(t), you must recover input signal spectrum $X(\omega)$ from sampled signal spectrum $Y(\omega)$, which is possible when there is no overlapping between the cycles of $Y(\omega)$.

Possibility of sampled frequency spectrum with different conditions is given by the following diagrams:
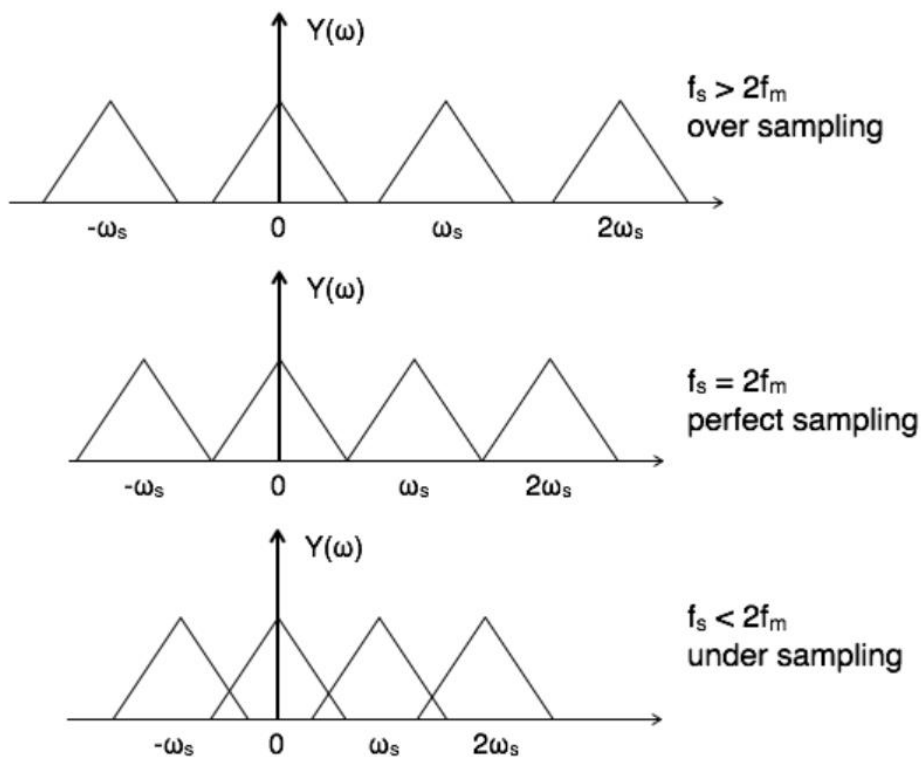
Program

```
clc;clear;
fm = input("enter input signal frequency");
k = input("enter number of cycles");
ti = 0:1/(fm*fm):k/fm;
x1 = sin(2*%pi*fm*ti);
subplot(2,2,1);
plot(ti,x1);
title("input signal");
xlabel("time");
ylabel("amplitude");


fs1 = 1.5*fm;
n1 = 0:1/(fs1):k/fm;
x2 = sin(2*%pi*fm*n1);
subplot(2,2,2);
plot(n1,x2);
plot2d3(n1,x2);
title("under sampling");
xlabel("time");
ylabel("amplitude");


fs2 = 3*fm;
n2 = 0:1/(fs2):k/fm;
x3 = sin(2*%pi*fm*n2);
subplot(2,2,3);
plot(n2,x3);
plot2d3(n2,x3);
```

title("Nyquist sampling");

xlabel("time");

ylabel("amplitude");

fs3 = 20*fm;

n3 = 0:1/(fs3):k/fm;

x4 = sin(2*%pi*fm*n3);

subplot(2,2,4);

plot(n3,x4);

plot2d3(n3,x4);

title("Over sampling");

xlabel("time");

ylabel("amplitude");

Output:
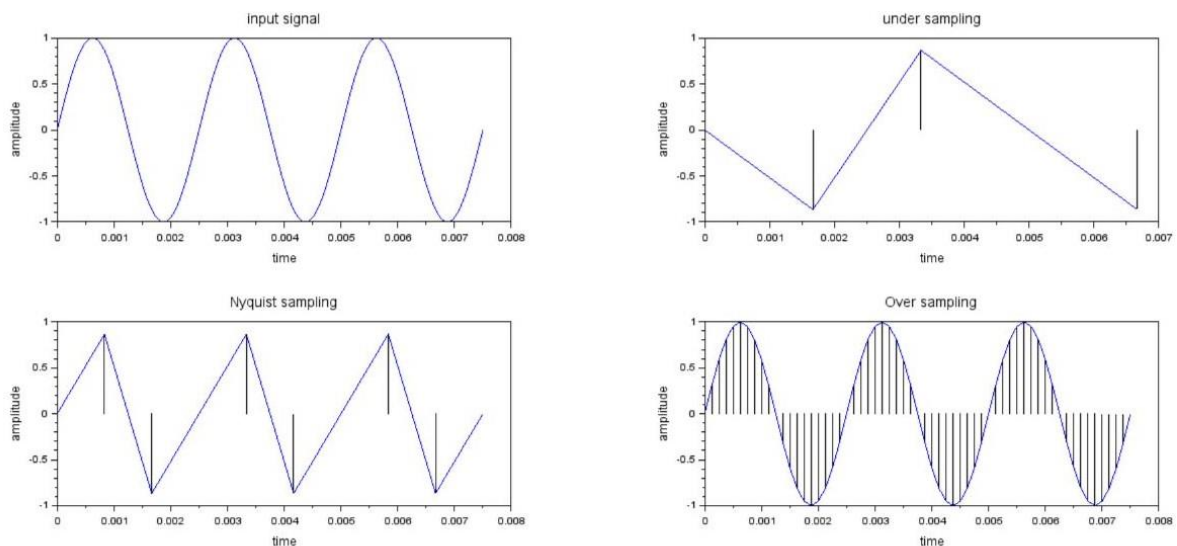
Result

Obtained the plots and verified the sampling theorem

Experiment 3
Linear Convolution

Aim: To perform convolution of a discrete sequence

1. Using conv()
2. Using loops

Theory:

Convolution: When speaking purely mathematically, convolution is the process by which one may compute the overlap of two graphs. In fact, convolution is also interpreted as the area shared by the two graphs over time. Metaphorically, it is a blend between the two functions as one passes over the other. So, given two functions F(n) and G(n), the convolution of the two is expressed and given by the following mathematical expression:

$$f(t) * g(t) = \int_{-\infty}^{\infty} f(u).g(t-u)du$$

or

$$f(n) * g(n) = \sum_{k=-\infty}^{\infty} f(k).g(n-k)$$

Linear Convolution: Linear Convolution is a means by which one may relate the output and input of an LTI system given the system's impulse response. Clearly, it is required to convolve the <u>input</u> signal with the impulse response of the system.

i)using conv()

Program:

```
clc;
clear;
clf;
x = input("Enter First Sequence = ");
h = input("Enter Second Sequence = ");
a=input("Enter the lower limit of x= ");
c=input("Enter the lower limit of h= ");
t1=a:a+length(x)-1;
t2=c:c+length(h)-1;
t3=(min(t1)+min(t2)):(max(t1)+max(t2));

//x=[1 2 3 1];
//h=[1,2,1,-1];
y=conv(x,h);

subplot(2,2,1);
plot2d3(t1,x);

title('First Sequence');
xlabel('Sample');
ylabel('Amplitude');
a=gca()
a.x_location="origin";
a.y_location="origin";
```

```
subplot(2,2,2);

plot2d3(t2,h);

title('Second Sequence');

xlabel('Sample');

ylabel('Amplitude');

a=gca()

a.x_location="origin";

a.y_location="origin";


subplot(2,2,3);

plot2d3(t3,y);

title('Convoluted Sequence');

xlabel('Sample');

ylabel('Amplitude');

a=gca()

a.x_location="origin";

a.y_location="origin";
```

Output:

```
Enter First Sequence = [1,2,3,1]

Enter Second Sequence = [1,2,1,-1]

Enter the lower limit of x= 0

Enter the lower limit of h= 0
```
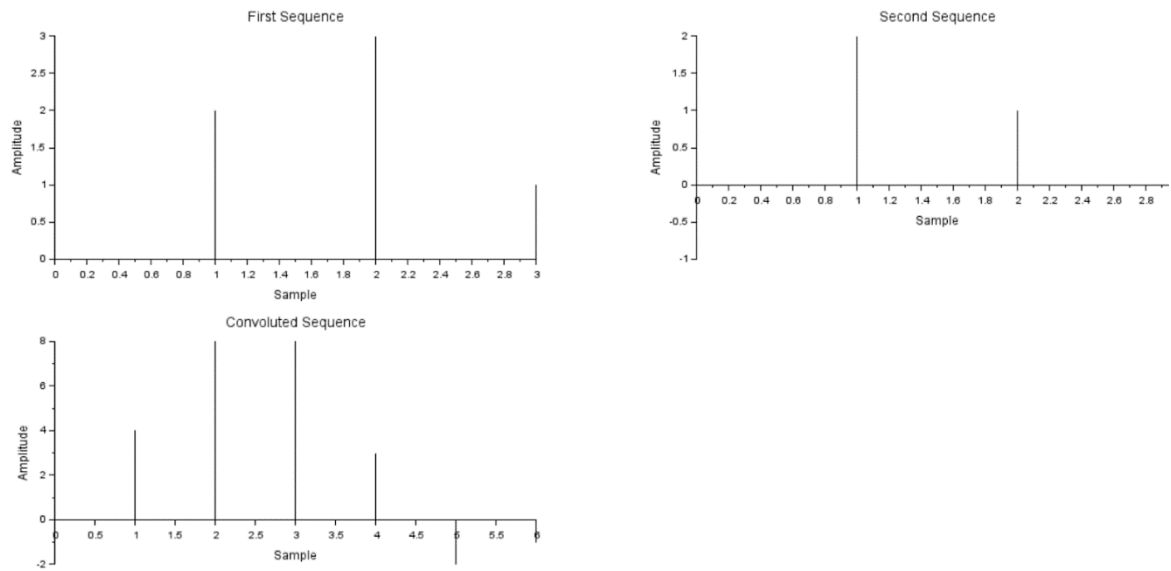
ii)without using conv()

Program:

```
clc;
clear all;
close ;
n1 = int(input("Enter the number of elements in x[n]"));
n2 = int(input("Enter the number of elements in h[n]"));
for i = 1:n1
    x(i) = int(input("Enter the terms in x[n]"));
end
for j = 1:n2
    h(j) = int(input("Enter the terms in h[n]"));
end
x_st=input("Enter Starting Index of xn:");
```

```
h_st=input("Enter the starting Index of hn:");

m = length(x);

n = length(h);
```
// Computation using convolution equation

```
for i = 1:n+m-1

   sum = 0;

   for j = 1:i

      if (((i-j+1) <= n)&(j <= m))

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

      end;

      y(i) = sum;

   end;

end;

ly=n1+n2-1;

y_st=x_st+h_st;

y_index=[y_st:y_st+ly-1];

subplot(3,1,1);

plot2d3([x_st:x_st+n1-1],x,style=[color("red")]);

title("x[n](Input Signal)");

subplot(3,1,2);

plot2d3([h_st:h_st+n2-1],h,style=[color("red")]);

title("h[n](Impulse Response)]");

subplot(3,1,3);

plot2d3(y_index,y,style=[color("red")]);

title("y[n](Output Signal)");
```

Output:

```
Enter the number of elements in x[n]4

Enter the number of elements in h[n]4

Enter the terms in x[n]1

Enter the terms in x[n]2

Enter the terms in x[n]3

Enter the terms in x[n]1

Enter the terms in h[n]1

Enter the terms in h[n]2

Enter the terms in h[n]1

Enter the terms in h[n]-1

Enter Starting Index of xn:0

Enter the starting Index of hn:0
```
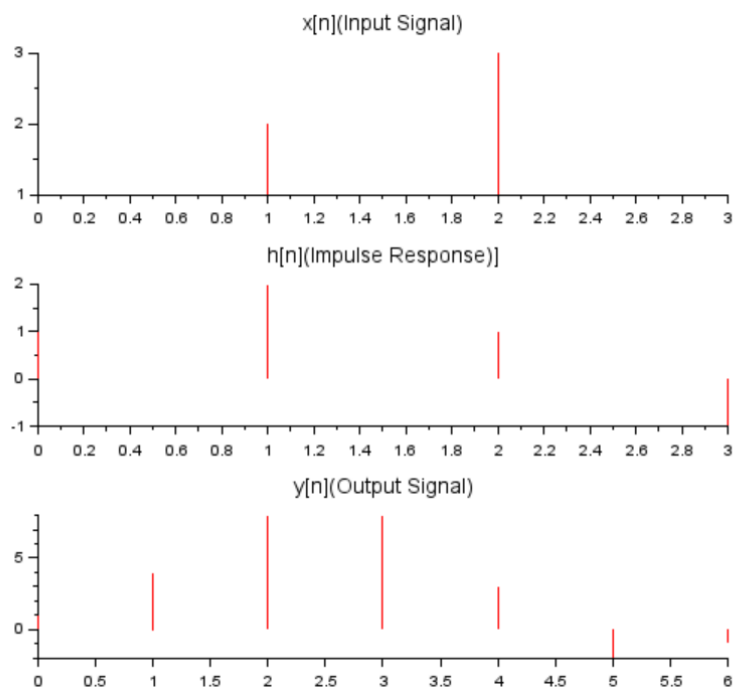

Result

The linear convoluted result of 2 sequences has been observed and plotted.

Experiment 4
Circular Convolution

Aim: To perform linear convolution using circular convolution and perform circular convolution using

1. Concentric circle method
2. FFT Method
3. Matrix method

Theory:

Theory: The Circular convolution, also known as cyclic convolution, of two periodic functions occurs when one of them is convolved in the normal way with a periodic summation of the other function. Circular convolution is only defined for finite length functions (usually equal in length), continuous or discrete in time. In circular convolution, it is as if the finite length functions repeat in time, periodically. Because the input functions are now periodic, the convolved output is also periodic. Circular convolution sum can be calculated using the formula:

$$y(n) = x_1(n) * x_2(n) = \sum_{k=0}^{N-1} x_1(k)x_2((n-k))_N$$

$$n = 0,1,\dots,N-1$$

Circular convolution can be performed in different ways:

1. Using the expression for linear convolution sum, but assuming the signal repeats periodically. This can be done by changing the negative indices of (n-k) to repetitions of the latter portions of the original aperiodic signal.

2. Convolution in time domain corresponds to multiplication in frequency domain. To make use of this property, we can calculate the DTFT of each of the aperiodic

signals, multiply these in the frequency domain, and find the IDFT of the product, to get the periodic convolved signal in time domain.

i) circular conv using concentric circle method

Program:

```
clc;clf;clear;
x=input('enter first sequence :');
h1=input('enter second sequence :');
m = max(length(x),length(h1));
newx = [x, zeros(1,m-length(x))];
newh1 = [h1, zeros(1,m-length(h1))];
t=0:m-1;
newx=newx(:,$:-1:1);
for i=1:m
  newx=[newx($),newx(1:$-1)];
  y(i)=sum(newx.*newh1);
end;
disp(y);
plot2d3(t,y);
title('Circular Convolution');
xlabel('Sample');
ylabel('Amplitude');
a=gca()
a.x_location ="origin";
a.y_location ="origin";
```

ii)using FFT method

Program:

```
clc;clf;clear;
x=input('enter sequence 1 ');
y=input('enter sequence 2 ');
m = max(length(x),length(y));
t=0:m-1;
newx = [x, zeros(1,m-length(x))];
newy = [y, zeros(1,m-length(y))];
a=fft(newx);
b=fft(newy);
c=a.*b;
d=ifft(c);
disp(d);
plot2d3(t,d);
title('Circular Convolution');
xlabel('Sample');
ylabel('Amplitude');
a=gca()
a.x_location ="origin";
a.y_location ="origin";
```

iii) matrix method

Program:

```
clc;clf;clear;
```

```
x=input('enter sequence 1 ');
h=input('enter sequence 2 ');
m = max(length(x),length(h));
t=0:m-1;
newx = [x, zeros(1,m-length(x))];
newh = [h, zeros(1,m-length(h))];
hx=newh(:,$:-1:1);
fh=[];
for i=1:m
   hx=[hx($),hx(1:$-1)];
   fh=[fh;hx];
end

y=fh*x';
disp(y);
plot2d3(t,y);
title('Circular Convolution');
xlabel('Sample');
ylabel('Amplitude');
a=gca()
a.x_location ="origin";
a.y_location ="origin";
```

iv)using linear conv

 Program:

```
clc;clf;clear;
x = input("Enter First Sequence = ");
```

```
h = input("Enter Second Sequence = ");

y=conv(x,h);


ln= length(y);

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

t = 0:m-1;

e = ln - m;

for i=1:e

    y(i)=y(ln-e+i)+y(i);

end

y=[y(1:$-e)];

disp(y);

plot2d3(t,y);

title('Circular convolution using Linear Convolution');

xlabel('Sample');

ylabel('Amplitude');

a=gca()

a.x_location ="origin";

a.y_location ="origin";


a.y_location ="origin";
```

Output:

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

enter second sequence :[2,3]


    14.
    7.
    12.
    17.
```


Circular Convolution

iv)linear convolution using circular convolution

Program:

clc;clf;clear;

x=input('enter first sequence :');

h1=input('enter second sequence :');

m =length(x)+length(h1)-1;

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

newh1 = [h1, zeros(1,m-length(h1))];

```
t=0:m-1;

newx=newx(:,$:-1:1);

for i=1:m

  newx=[newx($),newx(1:$-1)];

  y(i)=sum(newx.*newh1);

end;

disp(y);

plot2d3(t,y);

title('Linear convolution using Circular Convolution');

xlabel('Sample');

ylabel('Amplitude');

a=gca()

a.x_location ="origin";

a.y_location ="origin";
```

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

enter second sequence :[2,3]


    2.
    7.
    12.
    17.
    12.
```
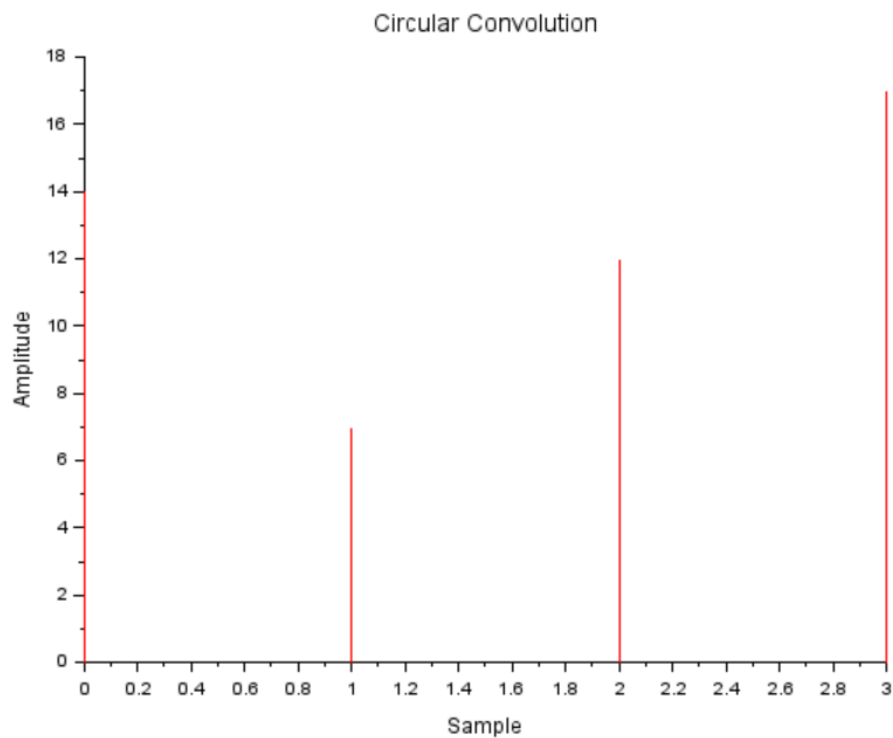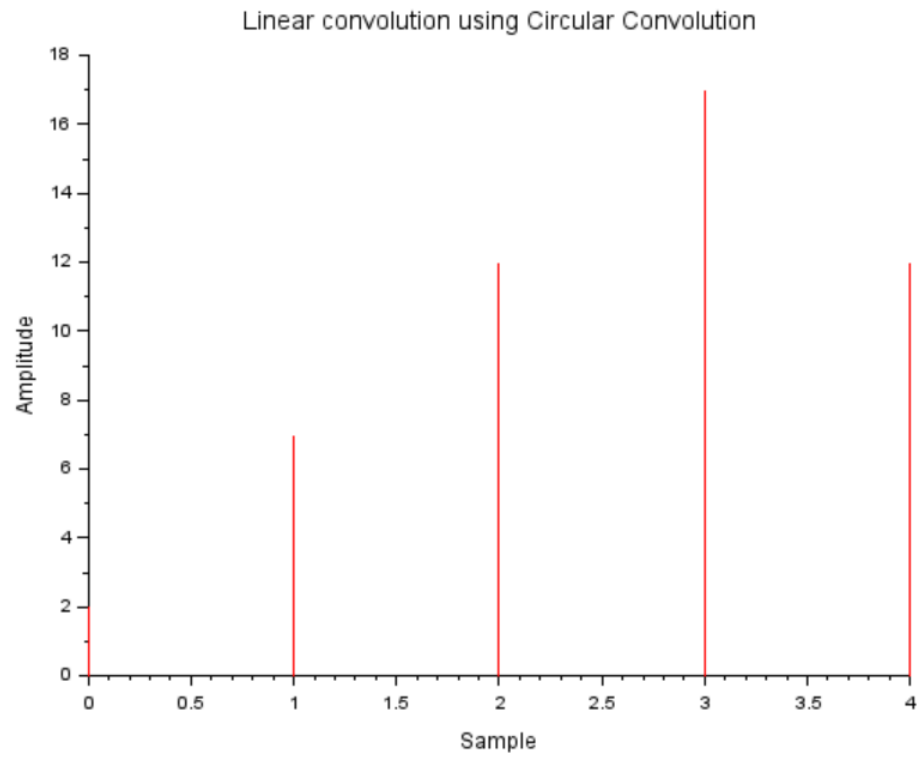
Linear convolution using Circular Convolution

Result

The circular convoluted result of 2 sequences have been observed and plotted.

Experiment 5

Filtering Long-duration Signals
(Overlap-add and Overlap-save)

Aim: To perform convolution of a discrete sequence

1.   Using overlap add method
2.   Using overlap save method

Theory:

One of the first applications of the (FFT) was to implement convolution faster than the usual direct method.

the FFT performs circular convolution with a filter of equal or lesser length. In computing the DFT or FFT we often have to make linear convolution behave like circular convolution, or vice versa. Two methods that make linear convolution look like circular convolution are overlap-save and overlap-add.

Overlap add

The output of a long sequence can be calculated by simply summing the outputs of each block of the input. What is complicated is that the output blocks are longer than the input. This is dealt with by overlapping the tail of the output from the previous block with the beginning of the output from the present block. In other words, if the block length is N and it is greater than the filter length L, the output from the second block will overlap the tail of the output from the first block and they will simply be added. Hence the name: overlap-add. Combining the overlap-add organization with use of the FFT yields a very efficient algorithm for calculating convolution that is faster than direct calculation for lengths above 20 to 50. This cross-over point depends on the computer being used and the overhead needed by use of the FFTs.

Overlap save

The overlap-save procedure cuts the signal up into equal length segments with some overlap. Then it takes the DFT of the segments and saves the parts of the convolution that correspond to the circular convolution. Because there are overlapping sections, it is like the input is copied therefore there is not lost information in throwing away parts of the linear convolution.

i) Overlap save

Program:

```
clc;clear;
x = input("Enter x(n): ");
m = length(x);
xl = 0;
xh = xl+m-1;
nx = xl: 1 :xh;
h = input("Enter h(n): ");
l = length(h);
hl = 0;
hh = hl+l-1;
nh = hl:l:hh;
N = m+l-1;
newh = [h zeros(1,N-m)];
n3 = length(newh);
y = zeros(1,N);
x1 = [zeros(1,n3-l) x zeros(1,n3)];
H = fft(newh);
for i=1:l:N
```

```
    y1 = x1(i:i+(2*(n3-l)));

    y2 = fft(y1);

    y3 = y2.*H;

    y4 = round(ifft(y3));

    y(i:(i+n3-l))= y4(l:n3);

end

disp('Result using overlap save method: ',y(1:N));
```
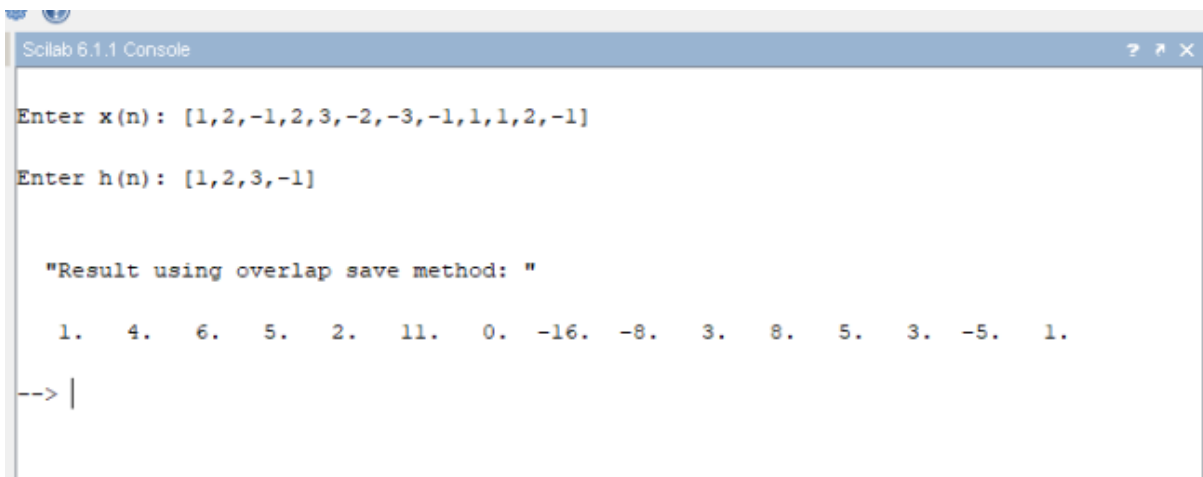
Output:

ii)using overlap add

Program:

```
clc;clear;

x=input('Enter 1st Sequence X(n)= ');

h=input('Enter 2nd Sequence H(n)= ');

n1=input("Enter the number of blocks: ");

n=length(x);

M=length(h);
```

```
N=n1+M-1;

L=N-M+1;

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

n3=length(h1);

H=fft(h1,-1);

i=modulo(n,L);

if i>0 then

 x=[x zeros(1,i)]

end

k=length(x);

y=zeros(1,N);
```

Output:

```
Enter 1st Sequence X(n)= [1,2,-1,2,3,-2,-3,-1,1,1,2,-1]

Enter 2nd Sequence H(n)= [1,2,3,-1]

Enter the number of blocks: 3


  "Result using overlap add method: "

  1.   4.   6.   5.   2.   11.   0.   -16.   -8.   3.   8.   5.   3.   -5.   1.

-->
```

Result

The convoluted result of 2 sequences of which one is a long sequence has been observed and plotted.

Experiment 6

FFT and DFT

Aim: To compute the FFT and inverse FFT of a sequence

1. Using inbuilt function
2. Using 8-point radix-2 DIT and DIF

Theory:

In mathematics, the discrete Fourier transform (DFT) converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the discrete-time Fourier transform (DTFT), which is a complex-valued function of frequency. The interval at which the DTFT is sampled is the reciprocal of the duration of the input sequence. An inverse DFT is a Fourier series, using the DTFT samples as coefficients of complex sinusoids at the corresponding DTFT frequencies. It has the same sample-values as the original input sequence. The DFT is therefore said to be a frequency domain representation of the original input sequence. If the original sequence spans all the non-zero values of a function, its DTFT is continuous (and periodic), and the DFT provides discrete samples of one cycle. If the original sequence is one cycle of a periodic function, the DFT provides all the non-zero values of one DTFT cycle.

The DFT is the most important discrete transform, used to perform Fourier analysis in many practical applications.

Fast Fourier Transform: As the name implies, the Fast Fourier Transform (FFT) is an algorithm that determines Discrete Fourier Transform of an input significantly faster than computing it directly. In computer science lingo, the FFT reduces the number of computations needed for a problem of size N from $O(N^2)$ to $O(N \log N)$

Radix-2 FFT Decimation-in-Time Algorithm

In the following, we assume that the FFT length is $N = 2^\gamma$ for an integer $\gamma > 1$. An excellent reference on the DFT and the FFT is [12].

The FFT algorithm:

- Uses the fundamental principle of **"Divide and Conquer,"** i.e., dividing a problem into smaller problems with similar structure, the original problem can be successfully solved by solving each of the smaller problems.

- Takes advantage of periodicity and symmetry properties of W_N^{nk}:

  1. **Periodicity:** W_N^{nk} is periodic of fundamental period N with respect to n, and with respect to k, i.e.,

$$W_N^{nk} = \begin{cases} W_N^{(n+N)k}, \\ W_N^{n(k+N)}. \end{cases}$$

  2. **Symmetry:** The conjugate of W_N^{nk} is such that

$$\left[W_N^{nk} \right]^* = W_N^{(N-n)k} = W_N^{n(N-k)}.$$

It is of 2 types: Decimation in Frequency(DIF) and Decimation in Time(DIT)

i) inbuilt function

Program:

x=[1,2,3,4];*//Input*

y=fft(x,-1);*//Computing DFT*

disp(y);

z=fft(y,1);*//Computing IDFT*

disp(z);

Output:

```
10. + 0.i  -2. + 2.i  -2. + 0.i  -2. - 2.i

 1.   2.   3.   4.
```

ii) 8-point radix-2 DIT and DIF

DFT USING DIF

Program:

clc;clear;

x=[1 , 2 , 1 , 2 , 0 , 2 , 1 , 2];

x0=x(1);

x1=x(2);

x2=x(3);

x3=x(4);

x4=x(5);

x5=x(6);

x6=x(7);

x7=x(8);

//stage 1 computations

x0a = x4 + x0 ;

x1b = x5 + x1 ;

x2c = x6 + x2 ;

x3d = x7 + x3 ;

x4e =(x4 - x0) *(-1) ;

x5f =(x5 - x1) *(-1) ;

x6g =(x6 - x2) *(-1) ;

x7h =(x7 - x3) *(-1) ;

x5f_stageI_output_multiplied_by_twiddle =(x5f)*(0.707 -(sqrt (-1)) *(0.707)) ;

x6g_stageI_output_multiplied_by_twiddle =(x6g) *(-(sqrt (-1))) ;

x7h_stageI_output_multiplied_by_twiddle =(x7h)*(-0.707 -(sqrt (-1)) *(0.707)) ;

// Stage-2 computations

x0a_stageII_output = x2c + x0a ;

x1b_stageII_output = x3d + x1b ;

x2c_stageII_output =(x2c - x0a) *(-1) ;

x3d_stageII_output =(x3d - x1b) *(-1) ;

x4e_stageII_output = x6g + x4e ;

x5f_stageII_output=x7h_stageI_output_multiplied_by_twiddle
+x5f_stageI_output_multiplied_by_twiddle ;

x6g_stageII_output =(x6g_stageI_output_multiplied_by_twiddle - x4e)*(-1);

x7h_stageII_output =(x7h - x5f) *(-1) ;

x3d_stageII_output_multiplied_by_twiddle =(x3d_stageII_output) *((sqrt (-1))) ;

x7h_stageII_output_multiplied_by_twiddle =(x7h_stageII_output) *(-(sqrt (-1))) ;

//stage 3computations

X0 = x1b_stageII_output + x0a_stageII_output ;

X4 =(x1b_stageII_output - x0a_stageII_output) *(-1) ;

X2 = x3d_stageII_output_multiplied_by_twiddle +x2c_stageII_output ;

X6 =(x3d_stageII_output_multiplied_by_twiddle -x2c_stageII_output) *(-1) ;

X1 =(x5f_stageII_output + x4e_stageII_output) ;

X5 =(x5f_stageII_output - x4e_stageII_output) *(-1) ;

X3 = x7h_stageII_output_multiplied_by_twiddle +x6g_stageII_output ;

X7 =(x7h_stageII_output_multiplied_by_twiddle -x6g_stageII_output) *(-1) ;

disp(X0,X1,X2,X3,X4,X5,X6,X7);

DFT USING DIT

Program:

```
clc;

clear;

clf;

x =[1 , 2 , 1 , 2 , 0 , 2 , 1 , 2 ];

x0 =x(1);

x4 =x(2);

x2 =x(3);

x6 =x(4);

x1 =x(5);

x5 =x(6);

x3 =x(7);

x7 =x(8);
//stage 1 computations

x0a = x4 + x0;

x4b =(x4 - x0) *(-1);

x2c = x6 + x2;

x6d =( x6 - x2 ) *( -1) ;

x1e = x5 + x1;

x5f =( x5 - x1 ) *( -1);

x3g = x7 + x3;

x7h =(x7 - x3 ) *(-1);

x6d1 =( x6d ) *( - sqrt ( -1) );

x7h1 =( x7h ) *( - sqrt ( -1) );
//stage 2 computation

x0a_stageII_output = x2c + x0a;
```

x4b_stageII_output = x6d1 + x4b ;

x2c_stageII_output =(x2c - x0a) *(-1) ;

x6d_stageII_output =(x6d1 - x4b) *(-1) ;

x1e_stageII_output = x3g + x1e ;

x5f_stageII_output = x7h1 + x5f ;

x3g_stageII_output =(x3g - x1e) *(-1) ;

x7h_stageII_output =(x7h1 - x5f) *(-1) ;

x5f_stgII_op_multi_by_tw =(x5f_stageII_output)*(0.707 -(sqrt (-1)) *(0.707));

x3g_stgII_op_multi_by_tw =(x3g_stageII_output) *(-(sqrt (-1))) ;

x7h_stgII_op_multi_by_tw =(x7h_stageII_output)*(-0.707 -(sqrt (-1)) *(0.707)) ;

//stage 3 computations

X0 = x1e_stageII_output + x0a_stageII_output ;

X1 = x5f_stgII_op_multi_by_tw + x4b_stageII_output ;

X2 = x3g_stgII_op_multi_by_tw + x2c_stageII_output ;

X3 = x7h_stgII_op_multi_by_tw + x6d_stageII_output ;

X4 =(x1e_stageII_output - x0a_stageII_output) *(-1) ;

X5 =(x5f_stgII_op_multi_by_tw - x4b_stageII_output)*(-1) ;

X6 =(x3g_stgII_op_multi_by_tw - x2c_stageII_output)*(-1) ;

X7 =(x7h_stgII_op_multi_by_tw - x6d_stageII_output)*(-1) ;

disp(X0,X1,X2,X3,X4,X5,X6,X7);

Output:

```
11.

 1. + 0.i

-1. + 0.i

 1. + 0.i

-5.

 1. + 0.i

-1. + 0.i

 1. + 0.i
```

IDFT USING DIF

Program:

clc;clear;clf;

X=[11 ,1 , -1 ,1 , -5 ,1 , -1 ,1];

X0_conj =X(1);

X1_conj =X(2);

X2_conj =X(3);

X3_conj =X(4);

X4_conj =X(5);

X5_conj =X(6);

X6_conj =X(7);

X7_conj =X(8);

W0 =cos (((2* %pi) /8) *0) -(sqrt (-1)) *sin (((2* %pi) /8) *0);

W1 =cos (((2* %pi) /8) *1) -sqrt (-1) * sin (((2* %pi) /8) *1);

W2 =cos (((2* %pi) /8) *2) -sqrt (-1) * sin (((2* %pi) /8) *2);

W3 =cos (((2* %pi) /8) *3) -sqrt (-1) * sin (((2* %pi) /8) *3);

//Stage 1 computation

x0a = X0_conj + X4_conj

x1b = X1_conj + X5_conj

x2c = X2_conj + X6_conj

x3d = X3_conj + X7_conj

x4e = X0_conj +(-1) * X4_conj

x5f = X1_conj +(-1) * X5_conj

x6g = X2_conj +(-1) * X6_conj

x7h = X3_conj +(-1) * X7_conj

x4e1 = x4e * W0

x5f1 = x5f * W1

x6g1 = x6g * W2

x7h1 = x7h * W3

//Stage 2 computation

 x0a_stageII_output = x2c + x0a ;

 x1b_stageII_output = x3d + x1b ;

 x2c_stageII_output = x0a - x2c ;

 x3d_stageII_output = x1b - x3d ;

 x4e_stageII_output = x6g + x4e1 ;

 x5f_stageII_output = x7h + x5f1;

 x6g_stageII_output = x4e1 +(x6g1 *(-1)) ;

 x7h_stageII_output = x5f1 +(x7h1 *(-1)) ;

//   Stage 3 computation

x0 = x0a_stageII_output + x1b_stageII_output ;

x4 = x0a_stageII_output +(-1) * x1b_stageII_output;

 x2 = x3d_stageII_output *((-1) * sqrt (-1)) +x2c_stageII_output ;

x6 = x3d_stageII_output *((-1) * sqrt (-1)) *(-1) +x2c_stageII_output;

x1 = x4e_stageII_output + x5f_stageII_output;

x5 = x4e_stageII_output - x5f_stageII_output;

x3 = x7h_stageII_output *((-1) * sqrt (-1)) +x6g_stageII_output ;

x7 = x7h_stageII_output *((-1) * sqrt (-1)) *(-1) +x6g_stageII_output ;

//final computation

 x0_star =(1/8) *(x0);

 x4_star =(1/8) *(x4);

 x2_star =(1/8) *(x2);

 x6_star =(1/8) *(x6);

 x1_star =(1/8) *(x1);

 x5_star =(1/8) *(x5);

 x3_star =(1/8) *(x3);

 x7_star =(1/8) *(x7);

x0_star_real = real (x0_star) ;

x0_star_imag_conj =(-1) *(imag (x0_star)) ;

x1_star_real = real (x1_star) ;

x1_star_imag_conj =(-1) *(imag (x1_star)) ;

x2_star_real = real (x2_star) ;

x2_star_imag_conj =(-1) *(imag (x2_star)) ;

x3_star_real = real (x3_star) ;

x3_star_imag_conj =(-1) *(imag (x3_star)) ;

x4_star_real = real (x4_star) ;

x4_star_imag_conj =(-1) *(imag (x4_star)) ;

x5_star_real = real (x5_star) ;

x5_star_imag_conj =(-1) *(imag (x5_star)) ;

x6_star_real = real (x6_star) ;

x6_star_imag_conj =(-1) *(imag (x6_star)) ;

x7_star_real = real (x7_star) ;

x7_star_imag_conj =(-1) *(imag (x7_star)) ;

x0 = x0_star_real + x0_star_imag_conj ;

x1 = x1_star_real + x1_star_imag_conj ;

x2 = x2_star_real + x2_star_imag_conj ;

x3 = x3_star_real + x3_star_imag_conj ;

x4 = x4_star_real + x4_star_imag_conj ;

x5 = x5_star_real + x5_star_imag_conj ;

x6 = x6_star_real + x6_star_imag_conj ;

x7 = x7_star_real + x7_star_imag_conj ;

disp(x0,x1,x2,x3,x4,x5,x6,x7);

IDFT USING DIT

Program:

clc;

clear;

//let X=[11,1,−1 ,1 , −5 ,1 , −1 ,1];

X0_conj =11;

X4_conj = -5;

X2_conj = -1;

X6_conj = -1;

X1_conj =1;

X5_conj =1;

X3_conj =1;

X7_conj =1;

//Stage 1 computation

X0a = X4_conj + X0_conj;

X4b =(X4_conj - X0_conj) *(-1) ;

X2c = X6_conj + X2_conj ;

X6d =(X6_conj - X2_conj) *(-1) ;

X1e = X5_conj + X1_conj ;

X5f =(X5_conj - X1_conj) *(-1) ;

X3g = X7_conj + X3_conj ;

X7h =(X7_conj - X3_conj) *(-1) ;

// Stage 1 output at line 4 and line8 mul by twiddle factor

X6d'=(X6d) *(- sqrt (-1)) ;

X7h'=(X7h) *(- sqrt (-1)) ;

//Stage 2 computations

X0a_stageII_output = X2c + X0a ;

X4b_stageII_output = X6d'+ X4b ;

X2c_stageII_output =(X2c - X0a) *(-1) ;

X6d_stageII_output =(X6d' - X4b) *(-1) ;

X1e_stageII_output = X3g + X1e ;

X5f_stageII_output = X7h'+ X5f ;

X3g_stageII_output =(X3g - X1e) *(-1) ;

X7h_stageII_output =(X7h' - X5f) *(-1) ;

// stage 2 line6,7,8 mul by tw (0.707 − j 0 . 7 0 7) ,(− j) and (−0.707− j 0 . 7 0 7)

 X5f_stageII_output_multiplied_by_twiddle =(X5f_stageII_output)*(0.707 -(sqrt (-1)) *(0.707)) ;

X3g_stageII_output_multiplied_by_twiddle =(X3g_stageII_output) *(-(sqrt (-1))) ;

X7h_stageII_output_multiplied_by_twiddle =(X7h_stageII_output) *(-0.707 -(sqrt (-1)) *(0.707)) ;

x0_star =(1/8) *(X1e_stageII_output + X0a_stageII_output) ;

x1_star =(1/8) *(X5f_stageII_output_multiplied_by_twiddle +X4b_stageII_output) ;

x2_star =(1/8) *(X3g_stageII_output_multiplied_by_twiddle +X2c_stageII_output) ;

x3_star =(1/8) *(X7h_stageII_output_multiplied_by_twiddle +X6d_stageII_output) ;

x4_star =(1/8) *((X1e_stageII_output -X0a_stageII_output) *(-1)) ;

x5_star =(1/8) *((X5f_stageII_output_multiplied_by_twiddle -X4b_stageII_output) *(-1)) ;

x6_star =(1/8) *((X3g_stageII_output_multiplied_by_twiddle -X2c_stageII_output) *(-1)) ;

x7_star =(1/8) *((X7h_stageII_output_multiplied_by_twiddle -X6d_stageII_output) *(-1)) ;

x0_star_real = real (x0_star) ;

x0_star_imag_conj =(-1) *(imag (x0_star)) ;

x1_star_real = real (x1_star) ;

x1_star_imag_conj =(-1) *(imag (x1_star)) ;

 x2_star_real = real (x2_star) ;

x2_star_imag_conj =(-1) *(imag (x2_star)) ;

x3_star_real = real (x3_star) ;

x3_star_imag_conj =(-1) *(imag (x3_star)) ;

x4_star_real = real (x4_star) ;

x4_star_imag_conj =(-1) *(imag (x4_star)) ;

x5_star_real = real (x5_star) ;

x5_star_imag_conj =(-1) *(imag (x5_star)) ;

x6_star_real = real (x6_star) ;

x6_star_imag_conj =(-1) *(imag (x6_star)) ;

x7_star_real = real (x7_star) ;

x7_star_imag_conj =(-1) *(imag (x7_star)) ;

x0 = x0_star_real + x0_star_imag_conj ;

x1 = x1_star_real + x1_star_imag_conj ;

x2 = x2_star_real + x2_star_imag_conj ;

x3 = x3_star_real + x3_star_imag_conj ;

x4 = x4_star_real + x4_star_imag_conj ;

x5 = x5_star_real + x5_star_imag_conj ;

x6 = x6_star_real + x6_star_imag_conj ;

x7 = x7_star_real + x7_star_imag_conj ;

disp(x0,x1,x2,x3,x4,x5,x6,x7);

Output:

```
1.

2.

1.

2.

0.

2.

1.

2.
```

Result

Computed the FFT and DFT of a discrete sequence

Experiment 7

Properties of DFT

Aim: To verify the following properties

1. Linearity
2. Parsevals
3. Linear Convolution
4. Multiplication
5. Periodicity

Theory:

Linearity -

The linearity property states that if

Periodicity -

Let x(n) and x(k) be the DFT pair then if

$x(n+N) = x(n)$ for all n then

$X(k+N) = X(k)$ for all k

Thus periodic sequence xp(n) can be given as

$$xp(n) = \sum_{l=-\infty}^{\infty} x(n-lN)$$

Circular Convolution -

The Circular Convolution property states that if

$$x1(n) \xleftrightarrow[N]{DFT} X1(k) \text{ And}$$

$$x2(n) \xleftrightarrow[N]{DFT} X2(k) \text{ Then}$$

Then x1(n) (N) x2(n) $\xleftrightarrow{DFT}$ x1(k) x2(k) N

Multiplication -

The Multiplication property states that if

$$X1(n) \xleftrightarrow[N]{DFT} x1(k) \text{ And}$$

$$X2(n) \xleftrightarrow[N]{DFT} x2(k) \text{ Then}$$

Then   x1(n) x2(n) $\xleftrightarrow{DFT}$ 1/N   x1(k) N (N) x2(k)

The Parseval s theorem states -

$$\sum_{n=0}^{N-1} X(n) y^*(n) = 1/N \sum_{n=0}^{N-1} x(k) y^*(k)$$

This equation give energy of finite duration sequence in terms of its frequency components.

i)Linearity

Program:

```
clc;clear;close;
x1=[1,2,3,4];
x2=[2,1,2,1];
a1=2;a2=3;
y1=fft(a1*x1+a2*x2);
k=fft(x1);
z=fft(x2);
y2=a1*k+a2*z;
if y1==y2 then
    disp('Linearity Property Verified');
end
```

ii)Parsevals

Program:

```
x1=[1,2,3,4];
x2=[2,1,2,1];
lhs=0;
for i=1:4
    lhs=lhs+abs(x1(i))**2;
end
N=4;
X1=fft(x1);
rhs=0;
for i=1:4
```

```
rhs=rhs+abs(X1(i))**2;

end


if lhs==rhs/N then

    disp("Parsevals identity Property verified")

end
```

iii)Circular convolution

Program:

```
x1=[1,2,3,4];

x2=[2,1,2,1];

xa=x1;

x1=x1(:,4:-1:1); // x1=[4,3,2,1]

for i=1:4

x1=[x1(4),x1(1:3)];//[1,4,3,2]

y(i)=sum(x1.*x2);

end

y_dft=fft(y);

X1=fft(xa);

X2=fft(x2);

rhs=X1.*X2;

if  y_dft==rhs'

    disp("Circular Convolution Property verified");

end
```

iv)Multiplication

Program:

```
x1=[1,2,3,4];
x2=[2,1,2,1];
Y1=x1.*x2;
 X1=fft(x1);
X2=fft(x2);
X1=X1(:,4:-1:1);
for i=1:4
   X1=[X1(4),X1(1:4-1)];
   y2(i)=sum(X1.*X2);
end
if fft(Y1)==y2/4 then
   disp("Multiplication Property Verified");
end
```

v)Periodicity

Program:

```
x=[1,1,0,0];
N=4;
for k=0:3
   sum1=0;
   for n=0:3
      sum1=sum1+x(n+1)*exp(-1*sqrt(-1)*2*%pi*(1/N)*k*n);
   end
X1(k+1)=sum1;
end
```

```
for k=0:3
    sum1=0;
for n=0:3
sum1=sum1+x(n+1)*exp(-1*sqrt(-1)*2*%pi*(1/N)*k*(n+N));
    end
X2(k+1)=sum1;
end
if round(X1)==round(X2) then
    disp("Periodicity Property Verified")
end
```

Output:

```
"Linearity Property Verified"

"Parsevals identity Property verified"

"Circular Convolution Property verified"

"Multiplication Property Verified"

"Periodicity Property Verified"
```

Result

All the properties of DFT have been verified via scilab.

Experiment 8

FIR Filter Design

Aim: To design the following FIR filters using rectangular window

1. Low Pass Filter
2. High Pass Filter
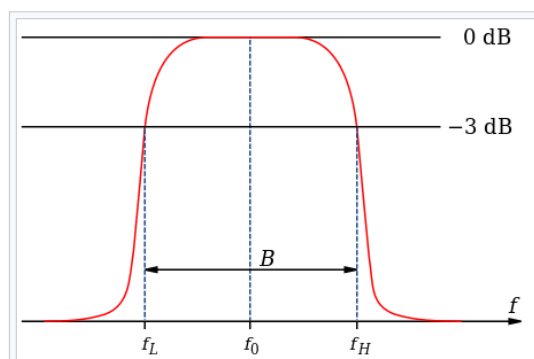3. Band Pass Filter
4. Band Stop Filter

Theory:

A Low Pass Filter can be a combination of capacitance, inductance or resistance intended to produce high attenuation above a specified frequency and little or no attenuation below that frequency. By plotting the networks output voltage against different values of input frequency, the Frequency Response Curve or Bode Plot function of the low pass filter circuit can be found. The Bode Plot shows the Frequency Response of the filter to be nearly flat for low frequencies and all of the input signal is passed directly to the output, resulting in a gain of nearly 1, called unity, until it reaches its Cut-off Frequency point (fc). This is because the reactance of the capacitor is high at low frequencies and blocks any current flow through the capacitor. When this occurs the output signal is attenuated to 70.7% of the input signal value or -3dB (20 log (Vout/Vin)) of the input. For this type of "Low Pass Filter" circuit, all the frequencies below this cut-off, fc point that are unaltered with little or no attenuation and are said to be in the filters Pass band zone. This pass band zone also represents the Bandwidth of the filter. Any signal frequencies above this point cut-off point are generally said to be in the filters Stop band zone and they will be greatly attenuated.

The passive high pass filter circuit as its name implies, only passes signals above the selected cut-off point, ƒc eliminating any low frequency signals from the waveform. In this circuit arrangement, the reactance of the capacitor is very high at low frequencies so the capacitor acts like an open circuit and blocks any input signals at VIN until the cut-off frequency point (ƒC) is reached. Above this cut-off frequency point the reactance of the capacitor has reduced sufficiently as to now act more like a short circuit allowing all of the input signal to pass directly to the output as shown below in the filters response curve. The Bode Plot or Frequency response curve that extends down from infinity to the cut-off frequency, where the output voltage amplitude is $1/\sqrt{2}$ = 70.7% of the input signal value or -3dB (20 log (Vout/Vin)) of the input value.

A band-pass filter or bandpass filter (BPF) is a device that passes frequencies within a certain range and rejects (attenuates) frequencies outside that range.

In signal processing, a band-stop filter or band-rejection filter is a filter that passes most frequencies unaltered, but attenuates those in a specific range to very low levels.[1] It is the opposite of a band-pass filter. A notch filter is a band-stop filter with a narrow stopband (high Q factor).

Bandwidth measured at half-power points (gain −3 dB, $\sqrt{2}/2$, or about 0.707 relative to peak) on a diagram showing magnitude transfer function versus frequency for a band-pass filter.
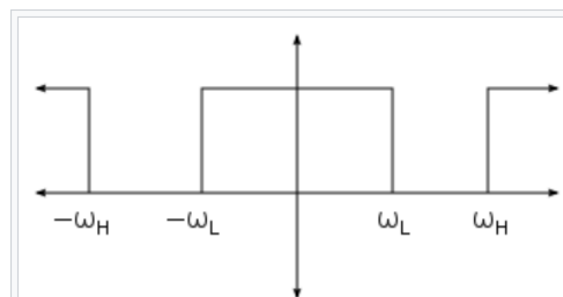

A generic ideal band-stop filter, showing both positive and negative angular frequencies

i)LPF FIR

Program:

```
clc;

clear;

clf;

fc=250;

fs=2000;

m=5; //order

wc=2*fc/fs ; //normalisation

disp(wc);

[wft,wfm,fr]=wfir("lp",m,[wc/2,0],"re",[0,0]);

disp(wft); //display h(n)

subplot(2,2,1);

plot(2*fr,20*log10(wfm)); //fr is wn values and wfm are h values

xlabel("Normalised Digital Frequency w * pi");

ylabel("Magnitude |H(w)| =");

title('Magnitude Response of LPF');

xgrid(1);

subplot(2,1,2);

plot(fr*fs,wfm);

xlabel('Analog Frequency in hz f');

ylabel('Magnitude |H(w)|=');

title('Magnitude Response of FIR LPF');
```

Output:

Magnitude Response of LPF

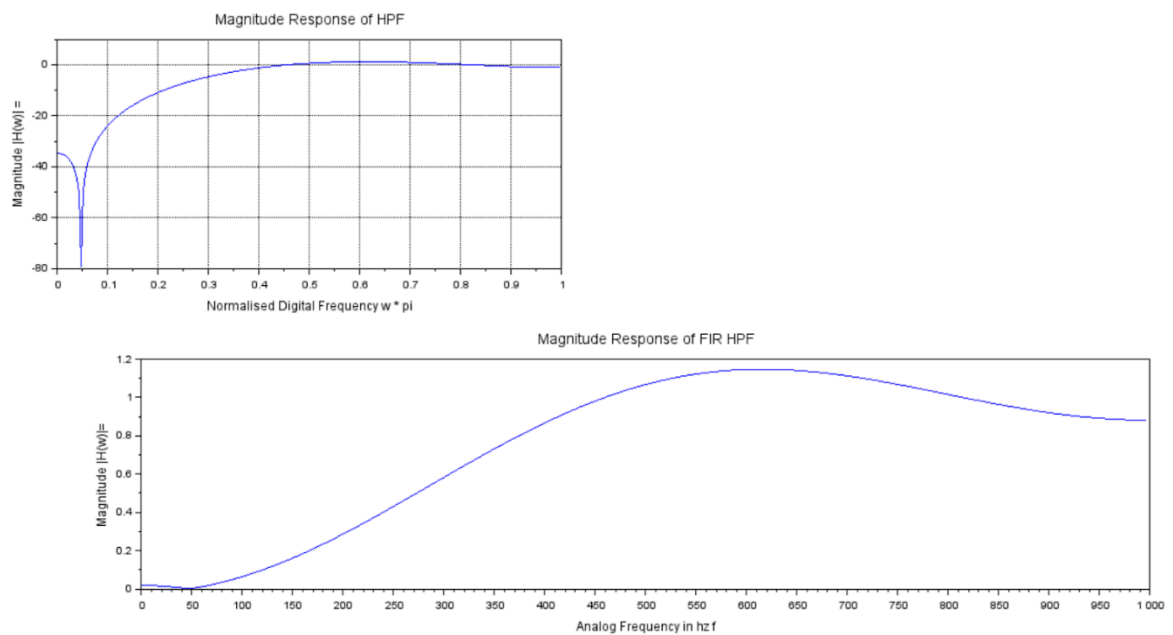Magnitude Response of FIR LPF
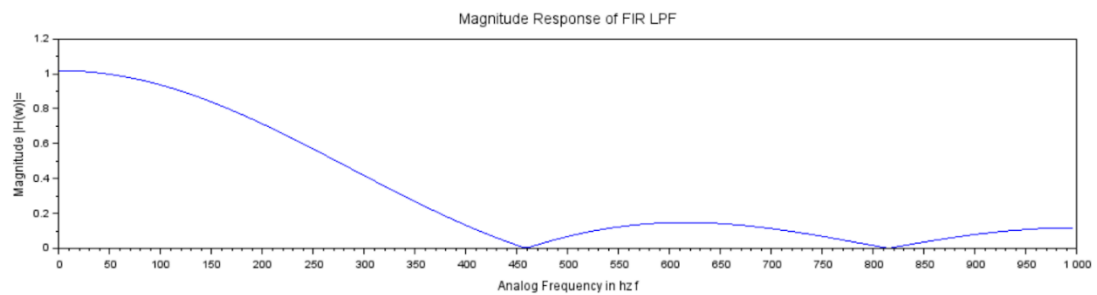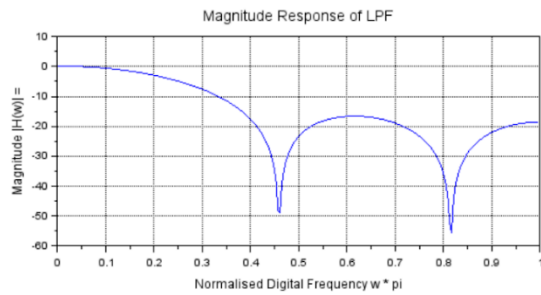
ii)HPF FIR

Program:

```
clc;clear;clf;
fc=250;
fs=2000;
m=5; //order
wc=2*fc/fs ; //normalisation
disp(wc);
[wft,wfm,fr]=wfir("hp",m,[wc/2,0],"re",[0,0]);
disp(wft); //display h(n)
subplot(2,2,1);
plot(2*fr,20*log10(wfm)); //fr is wn values and wfm are h values
xlabel("Normalised Digital Frequency w * pi");
ylabel("Magnitude |H(w)| =");
title('Magnitude Response of HPF');
xgrid(1);
```

subplot(2,1,2);

plot(fr*fs,wfm);

xlabel('Analog Frequency in hz f');

ylabel('Magnitude |H(w)|=');

title('Magnitude Response of FIR HPF');

Output:

iii)BPF FIR

Program:

```
clc;
clear;
clf;
fc1=200;
fc2=800;
```
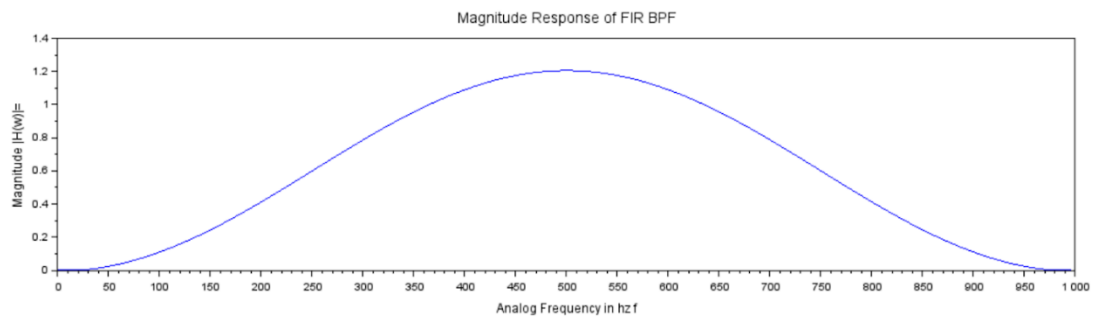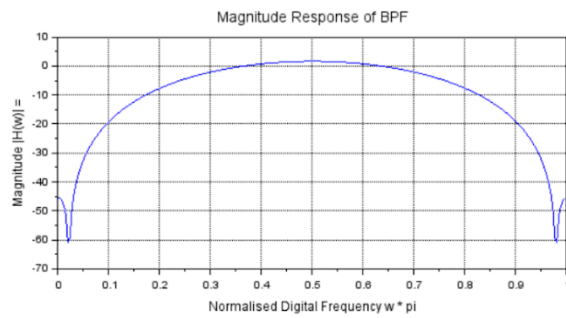
```
fs=2000;

m=5; //order

wc1=2*fc1/fs ; //normalisation

wc2=2*fc2/fs ;

disp(wc1);

disp(wc2);

[wft,wfm,fr]=wfir("bp",m,[wc1/2,wc2/2],"re",[0,0]);

disp(wft); //display h(n)

subplot(2,2,1);

plot(2*fr,20*log10(wfm)); //fr is wn values and wfm are h values

xlabel("Normalised Digital Frequency w * pi");

ylabel("Magnitude |H(w)| =");

title('Magnitude Response of BPF');

xgrid(1);

subplot(2,1,2);

plot(fr*fs,wfm);

xlabel('Analog Frequency in hz f');

ylabel('Magnitude |H(w)|=');

title('Magnitude Response of FIR BPF');
```

Output:

Magnitude Response of BPF

Magnitude Response of FIR BPF

iv)BSF FIR

Program:

```
clc;

clear;

clf;

fc1=200;

fc2=800;

fs=2000;

m=5; //order

wc1=2*fc1/fs ; //normalisation

wc2=2*fc2/fs ;

disp(wc1);

disp(wc2);

[wft,wfm,fr]=wfir("sb",m,[wc1/2,wc2/2],"re",[0,0]);

disp(wft); //display h(n)

subplot(2,2,1);

plot(2*fr,20*log10(wfm)); //fr is wn values and wfm are h values
```

xlabel("Normalised Digital Frequency w * pi");

ylabel("Magnitude |H(w)| =");

title('Magnitude Response of BSF');

xgrid(1);

subplot(2,1,2);

plot(fr*fs,wfm);

xlabel('Analog Frequency in hz f');

ylabel('Magnitude |H(w)|=');

title('Magnitude Response of FIR BSF');

Output:
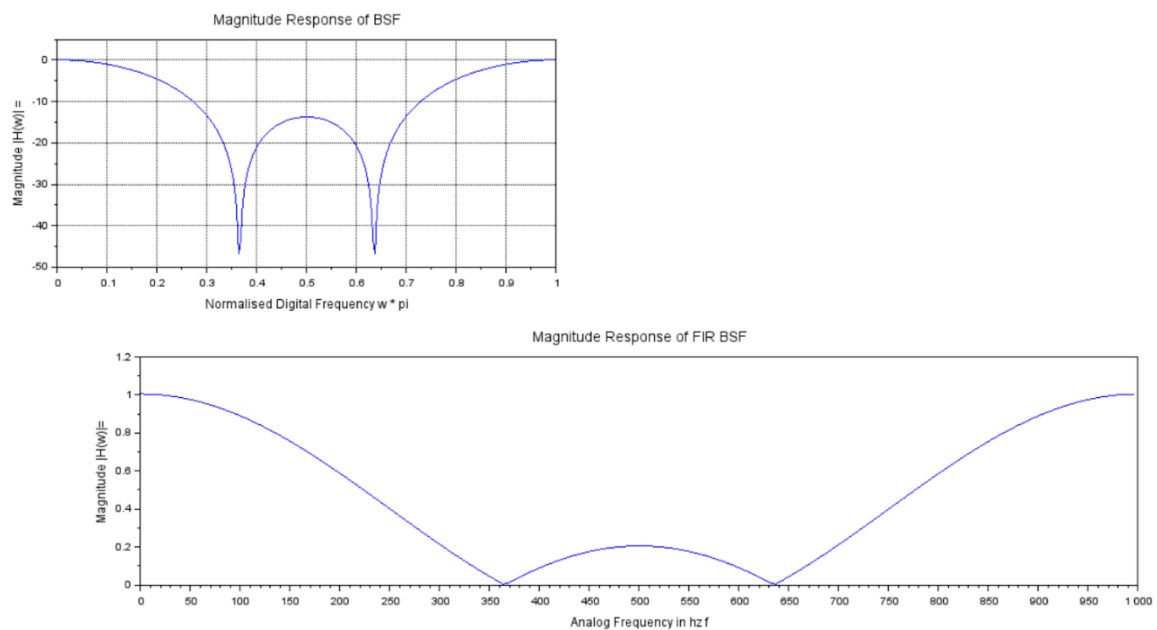
Result

Various FIR filters have been designed and observed using scilab.