Page No.
Date 21 | 1 | 20

EXPERIMENT-1

AIM -
To determine impulse response, step response and ramp response.

SOFTWARE USED - MATLAB R 2016b

THEORY -
Impulse Response :-
An important role is played in systems theory by the impulse response function, the laplace transform of which is called the Transfer Function. We remember that a linear time invariant system is represented by a linear differential equation with constant coefficients. An LTI system can be represented by

$$a\frac{d^2y}{dt^2} + b\frac{dy}{dt} + cy = f(t)$$

If we take $f(t)$ as delta function $\delta(t)$ then we get,

$$a\frac{d^2y}{dt^2} + b\frac{dy}{dt} + cy = \delta(t)$$

Scanned with CamScanner

By definition of the impulse response function we consider all initial conditions to be a

Taking laplace transform we get

$$as^2 Y(s) + bs Y(s) + c Y(s) = 1$$

$$Y(s) = \frac{1}{as^2 + bs + c}$$

This function the laplace transform of the impulse response function is called the transfer function and is usually denoted by $H(s)$ so we have

$$H(s) = \frac{1}{as^2 + bs + c}$$

Step Response

The response of a system (with all initial condition equal to zero ie zero state response) to the unit step input is called the unit step response

$$H(s) = \frac{Y(s)}{X(s)}$$

the output with zero initial conditions is given as:-

$$Y(s) = X(s)\ H(s)$$

So unit step response

$$Y(s) = \frac{1}{s} H(s)$$

Immediately we can determine two characteristics of the unit step response, the initial and final values theorem.

Ramp Response

The response of a system to the unit ramp input is called the unit ramp response. We can easily find the ramp input of a system from its transfer function.

$$H(s) = \frac{Y(s)}{X(s)}$$

Ramp response $Y(s) = \frac{1}{s^2} H(s)$

RESULT -

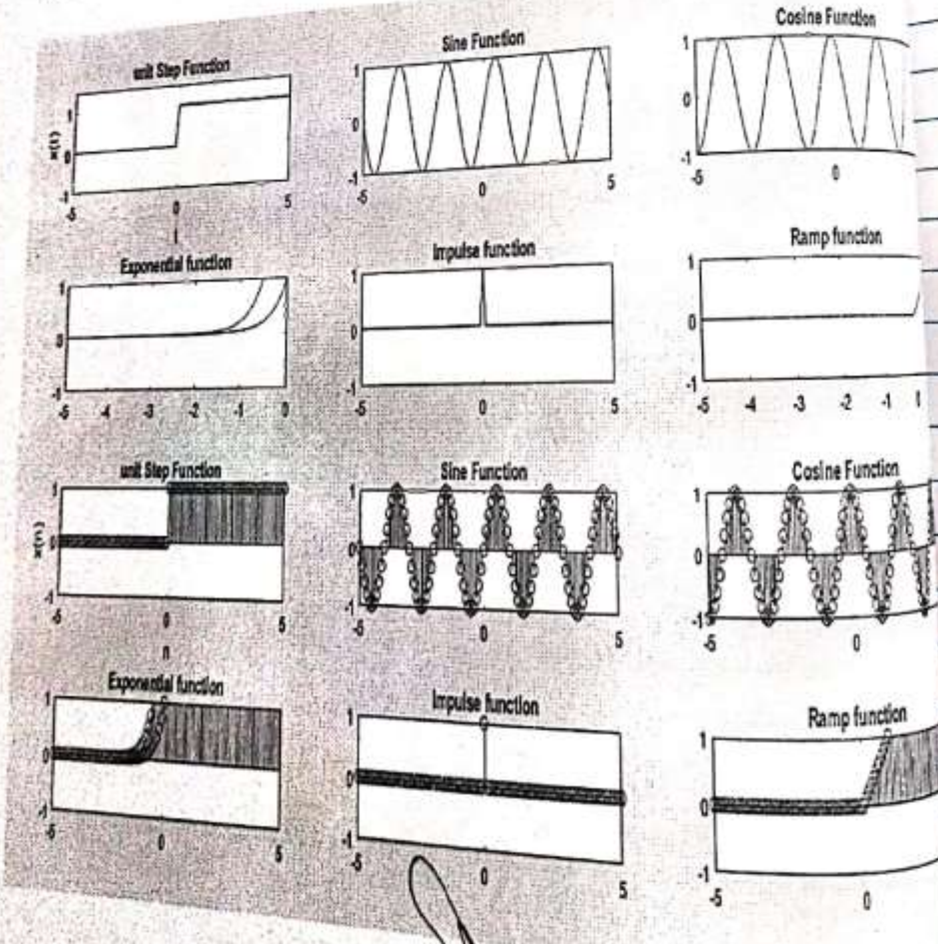
The impulse response, step response and ramp response has been successfully determined

Scanned with CamScanner

Experiment -1

```
clc;
clear all;
close all;
fc=0.5;
t= -5:0.1:5
u_t=[zeros(1,50),ones(1,51)];
subplot(4,3,1);
plot(t,u_t)
ylim([-1 1.5])
title('unit Step Function')
xlabel('t');
ylabel('x(t)');
sin1=sin(2*pi*fc*t)
subplot(4,3,2);
plot(t,sin1)
ylim([-1 1])
title('Sine Function')
cos1=cos(2*pi*fc*t)
subplot(4,3,3);
plot(t,cos1)
ylim([-1 1])
title('Cosine Function')
exp1=exp(2*pi*fc*t)
subplot(4,3,4);
plot(t,exp1)
hold on
plot (t,5*exp1)
ylim([-1 1])
title('Exponential function')
impul=[zeros(1,50),ones(1),zeros(1,50)];
subplot(4,3,5);
plot(t,impul)
ylim([-1 1])
title('Inpulse function')
ramp=t.*u_t
subplot(4,3,6);
plot(t,ramp)
ylim([-1 1])
title('Ramp function')
subplot(4,3,7);
stem(t,u_t)
ylim([-1 1])
title('unit Step Function')
xlabel('n');
ylabel('x(n)');
sin1=sin(2*pi*fc*t)
subplot(4,3,8);
stem(t,sin1)
ylim([-1 1])
title('Sine Function')
cos1=cos(2*pi*fc*t)
subplot(4,3,9);
stem(t,cos1)
ylim([-1 1])
```

```
title('Cosine Function')
expl=exp(2*pi*fc*t)
subplot(4,3,10);
stem(t,expl)
hold on
stem(t,5*expl)
ylim([-1 1])
title('Exponential function')
impul=[zeros(1,50),ones(1),zeros(1,50)];
subplot(4,3,11);
stem(t,impul)
ylim([-1 1])
title('Impulse function')
ramp=t.*u_t
subplot(4,3,12);
stem(t,ramp)
ylim([-1 1])
title('Ramp function')
```



7/1/2020

Scanned with CamScanner

Date 28 | 1 | 20

EXPERIMENT - 2

AIM -
Write a MATLAB program to find convolution and correlation of two discrete signals.

SOFTWARE USED -
MATLAB R2016 b

THEORY -

Convolution is a mathematical operation on two functions that produces a third function expressing how the shape of one is notified by the other. Convolution is defined as the integral of the product of the two functions after one is reversed and shifted.

The convolution of two signals in the time domain is equivalent to the multiplication of their representation in frequency domain. Mathematically, we can write convolution of two signal as :-

$$y(t) = x_1(t) * x_2(t)$$

$$= \int_{-\infty}^{\infty} x_1(p) \cdot x_2(t-p)\, dp$$

Convolution has functions / applications that include probability, statistics, computer vision, natural language processing, image and signal processing, engineering and differential equations.
Correlation is a measure of similarity between two signals The general for correlating is

$$\int_{-\infty}^{\infty} x_1(t)\, x_2(t-\tau)\, dt$$

There are two types of correlation:-

Auto Correlation :-
It is defined as correlation of a signal with itself. Auto - correlation function is a measure of similarity between a signal and its time delayed version. It is represented as $R(\tau)$

Consider a signal $x(t)$. The auto - correlation function of $x(t)$ with its time delayed version is

$$R_{11}(\tau) = R(\tau) = \int_{-\infty}^{\infty} x(t)\, x(t-\tau)\, dt$$

where, τ = delay parameter

If signal is complex, then

Scanned with CamScanner

$$R_{11}(\tau) = R(\tau)$$
$$= \int_{-\infty}^{\infty} x(t) * x(t-\tau)\, dt$$

2) CROSS - CORRELATION

It is the degree of similarity between two time series in the different time or space while lag can be considered when time is under investigation. The difference between these two time series in different situation like distance can be considered while the space is under investigation respectively.

$$R_{12} = \int_{-\infty}^{\infty} x_1(t)\, x_2(t-\tau)\, dt$$

If signals are complex then

$$R_{12} = \int_{-\infty}^{\infty} x_1(t) * x_2(t-\tau)\, dt$$

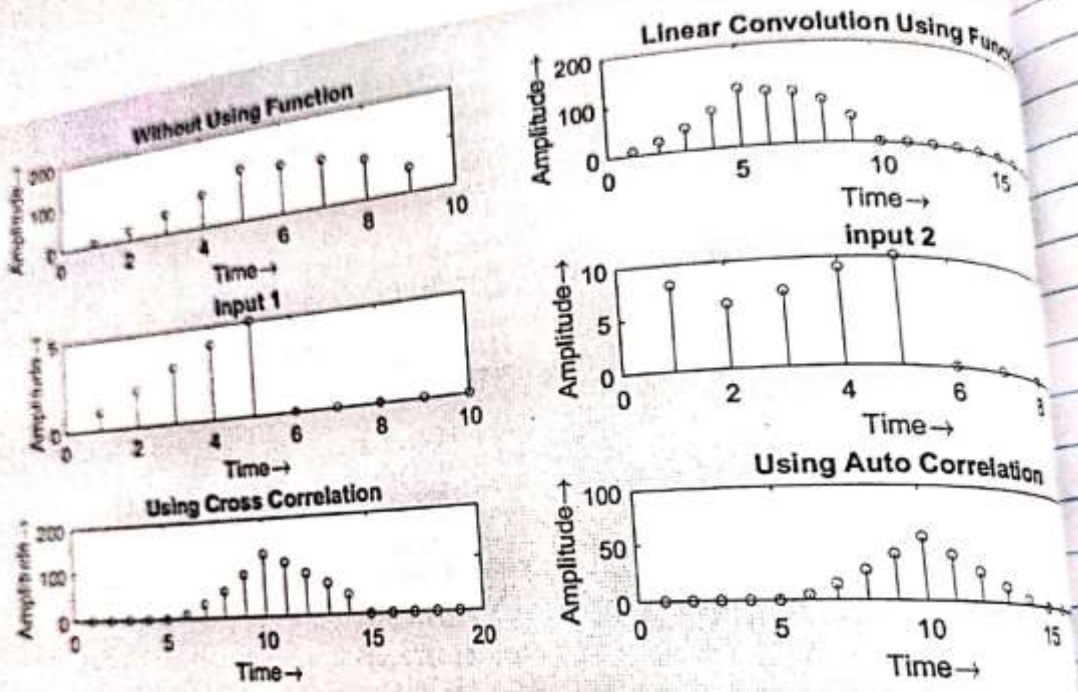
RESULT-

We have successfully written a program in MATLAB to find convolution and correlation of two discrete signals.

Experiment-2

```
clc;
clear all;
close all;
x=[1,2,3,4,5];
h=[8,6,7,9,10];
n1=length(x);
n2=length(h);
x=[x,zeros(1,n2)];
h=[h,zeros(1,n1)];
for i=1:n1+n2-1
    y(i)=0;
    for j=1:n2
        if(i-j+1>0)
            y(i)=y(i)+(x(j)*h(i-
j+1));
        end
    end
nd
ubplot(3,2,1);
=stem(y);
et(gca,'FontSize',16);
label('Time\rightarrow');
label('Amplitude\rightarrow');
tle('Without Using Function');

bplot(3,2,2);
=conv(x,h);
stem(y1);
t(gca,'FontSize',16);
abel('Time\rightarrow');
abel('Amplitude\rightarrow');
title('Linear Convolution Using
Function');

subplot(3,2,3);
stem(x);
set(gca,'FontSize',16);
xlabel('Time\rightarrow');
ylabel('Amplitude\rightarrow');
title('input 1');

subplot(3,2,4);
stem(h);
set(gca,'FontSize',16);
xlabel('Time\rightarrow');
ylabel('Amplitude\rightarrow');
title('input 2');

subplot(3,2,5);
y2=xcorr(x,h);
stem(y2);
set(gca,'FontSize',16);
xlabel('Time\rightarrow');
ylabel('Amplitude\rightarrow');
title('Using Cross Correlation');

subplot(3,2,6);
y3=xcorr(x);
stem(y3);
set(gca,'FontSize',16);
xlabel('Time\rightarrow');
ylabel('Amplitude\rightarrow');
title('Using Auto Correlation');
```

4/1/2020


Without Using Function
Linear Convolution Using
Input 1
input 2
Using Cross Correlation
Using Auto Correlation
Amplitude→
Time→

Date 04/02/20

EXPERIMENT-3

AIM - Write a MATLAB program to find convolution (circular) of two discrete signals.

SOFTWARE USED - MATLAB R2016b

THEORY -

The circular convolution, also known as cyclic convolution, of two aperiodic functions (i.e. Schwartz function) occurs when one of them is convolved in the normal way with a periodic summation of the other function. Let us take two finite duration sequences $x_1(n)$ and $x_2(n)$, having integer length as N. Their DFTs are $X_1(k)$ and $X_2(k)$ respectively, which is shown as

$$X_1(k) = \sum_{n=0}^{N-1} x_1(n)\, e^{\frac{-j2\pi kn}{N}}$$

$$k = 0, 1, 2, \ldots, N-1$$

$$X_2(k) = \sum_{n=0}^{N-1} x_2(n)\, e^{\frac{-j2\pi kn}{N}}$$

$$K = 0, 1, 2, \ldots, N-1$$

Page No.
Date

DFT is given as

$$X_3(k) = X_1(k) \times X_2(k)$$

By taking the IDFT, we get,

$$x_3(n) = \frac{1}{N} \sum_{n=0}^{N-1} X_3(k)\, e^{\frac{-j2\pi kn}{N}}$$

After solving we get,

$$x_3(n) = \sum_{m=0}^{N-1} x_1(m)\, x_2[((n-m))_N]$$

$$m = 0, 1, 2, \ldots, N-1$$

Methods of circular convolution :-

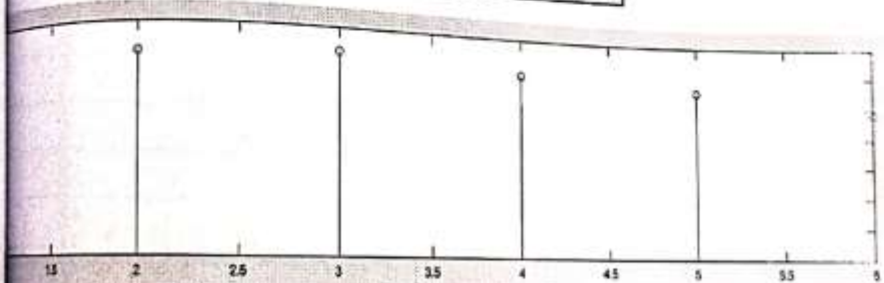
There are two methods :-
1) Concentric circle method
2) Matrix multiplication method

RESULT -
The circular convolution using MATLAB has been successfully performed.

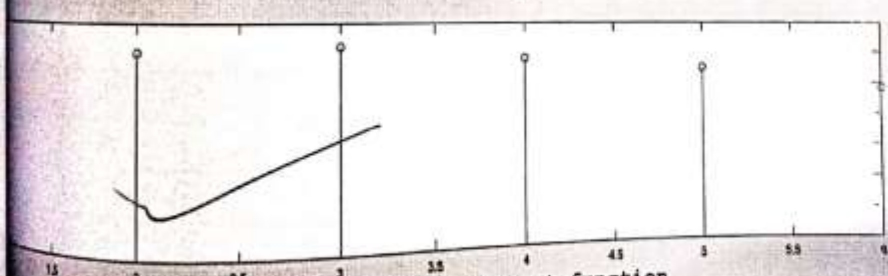
EXPERIMENT 3

```
clc
close all
clear all
x=[3 4 5 7 8 9]
h=[1 2 3 4 4 5]
l=length(x)
m=length(h)
N=max(l,m)
s=cconv(x,h,N)
cc1=fft(x)
cc2=fft(h)
z=ifft(cc1.*cc2)
subplot(211)
title('Circular convolution using function');
stem(s)
subplot(212)
title('Circular convolution without function');
stem(z)
```

s	[113,122,125,117,110,97]
x	[3,4,5,7,8,9]
z	[113,122,125,117,110,97]



Circular convolution using function



Circular convolution without function

Page No.
Date 04/02/20

EXPERIMENT- 4

AIM- Write a MATLAB program to find 8 point DFT, its magnitude and phase plot and its inverse DFT.

SOFTWARE USED - MATLAB R 2016 b

THEORY-

An N- point DFT is expressed as the multiplication $X = Wx$ where

x = original input signal
W = N by N square DFT matrix
X = DFT of signal

Transformation can be defined as $W = \left(\frac{w^{jk}}{\sqrt{N}}\right)_{jk=0...N-1}$

$$W = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & w^3 & \cdots & w^{N-1} \\ 1 & w^2 & w^4 & w^6 & \cdots & w^{2(N-1)} \\ 1 & w^3 & w^6 & w^9 & \cdots & w^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & w^{N-1} & w^{2(N-1)} & w^{3(N-1)} & \cdots & w^{(N-1)(N-1)} \end{bmatrix}$$

Scanned with CamScanner

Page No
Date

The only important thing is that the forward and inverse transforms have opposite sign exponents and that are the product of their normalisation factors be 1/N.

Fast fourier transforms utilize the symmetricities of the whole matrix, to reduce the time of multiplying a vector by this matrix, from the usual $O(N^2)$

RESULT -

The 8 point DFT, its magnitude and phase plot has been determined using MATLAB successfully.

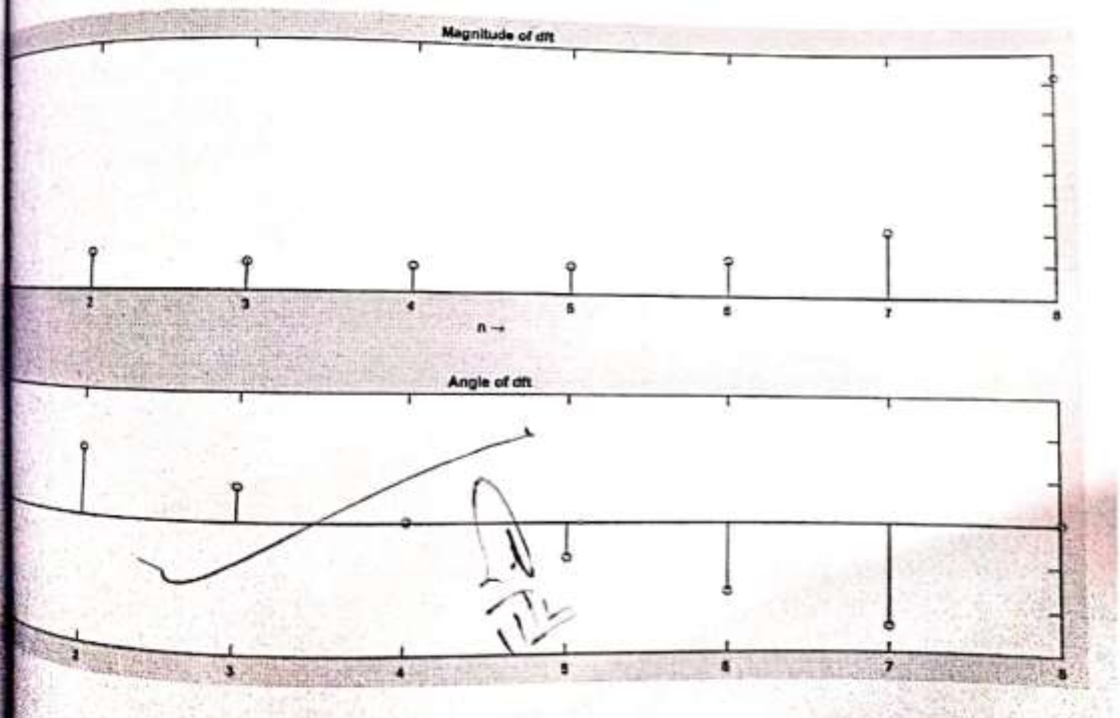
11/2

Scanned with CamScanner

EXPERIMENT 4

```
clc;
clear all;
close all;
x_n=[1 2 3 4 5 6 7 8];
X_k=[]
N=length(x_n)
for k=1:N;
    sum = 0;
    for n=1:N;
        sum = sum + x_n(n)*exp(-(j*2*pi*k*n)/N);
    end
    X_k(k) = sum;
end
mag =abs(X_k)
ang=angle(X_k)
x_n1=[];
for k=1:N;
    sum1 = 0;
    for n=1:N;
        sum1 = sum1 +
(1/N)*(X_k(n)*exp((j*2*pi*k*n)/N));
    end
    x_n1(k) = sum1
end
subplot(2,1,1)
stem(mag)
title('Magnitude of dft')
xlabel('n \rightarrow')
ylabel('mag of X(k)')
subplot(2,1,2)
stem(ang)
title('Angle of dft')
xlabel('n \rightarrow')
ylabel('ang of X(k)')
```

x_n1	[1.0000 + 0.0000i,2.0000 - 0.0000i,3.0000 - 0.0000i,4.0000 - 0.0000i,5.0000 + 0.0000i,6.0000 + 0.0000i,7.0000 - 0.0000i,8.0000 + 0.0...
x_n	[1,2,3,4,5,6,7,8]
X_k	[4.0000 + 9.6569i,4.0000 + 4.0000i,4.0000 + 1.6569i,4.0000 + 0.0000i,4.0000 - 1.6569i,4.0000 - 4.0000i,4.0000 - 9.6569i,36.0000 + 0...
sum1	8.0000 + 0.0000i
sum	36.0000 + 0.0000i
N	8
n	8
mag	[10.4525,5.6569,4.3296,4,4.3296,5.6569,10.4525,36]
k	8
ang	[1.1781,0.7854,0.3927,1.1022e-15,-0.3927,-0.7854,-1.1781,1.3879e-15]



EXPERIMENT -5

AIM: Perform the following properties of DFT:

a) Circular shift of a sequence

b) Circular fold of a sequence

Software Used:- MATLABR2016

Theory:

Circular Shift property:

A shift in time corresponds to a phase shift that is linear in frequency. Because of the periodicity induced by the DFT and IDFT, the shift is circular, or modulo NN samples.

x((n−m)modN)X(k)e−(i2πkmN) x n m N X k 2 k m N

The modulus operator pmodNp N means the remainder of pp when divided by NN. For example,

9mod5=49 5 4

and

−1mod5=4

Circular fold property:

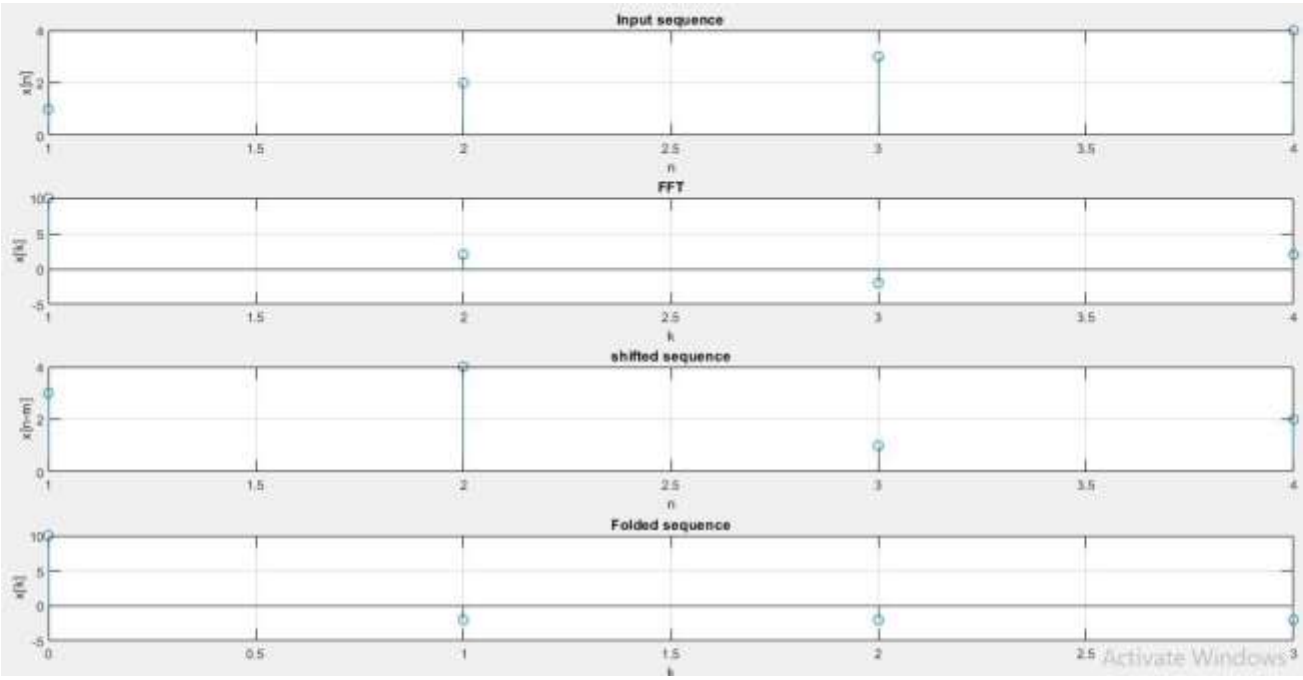
It means that multiplication of two sequences in time domain results in circular convolution of their DFT s in frequency domain. It means that the sequence is circularly folded its DFT is also circularly folded.

Code:

```
clc;

clear all;

close all;

xn=[1 2 3 4];

N=length(xn);

m=input(' Enter amount of shift');

xk=fft(xn);

k=0:N-1

x2k=xk.*(exp((-j*2*pi*k*m)/N));

xnm=ifft(x2k);

s=0:N-1

xf1=xn(mod(-s,N)+1);

xros=fft(xf1)

subplot(4,1,1)

stem(xn);

grid on;

xlabel('n');

ylabel('x[n]');

title('Input sequence');

subplot(4,1,2)
```

```
stem(x2k);

grid on;

xlabel('k');

ylabel('x[k]')

title('FFT');

subplot(4,1,3)

stem(xnm)

grid on;

xlabel('n');

ylabel('x[n-m]')

title('shifted sequence')

subplot(4,1,4)

stem(s,xros)

grid on;

xlabel('k')

ylabel('x[k]')

title('Folded sequence')
```

Output:


Input sequence
x[n]
n
FFT
x[k]
k
shifted sequence
x[n-m]
n
Folded sequence
x[k]
k
Activate Windows

Experiment-6

Aim:- Write a MATLAB Program to design FIR Low pass filter using

a) Rectangular window

b) Hanning window

c) Hamming window

d) Bartlett window

Software Used:- MATLABR2016

Theory:-

A finite impulse response (FIR) filter is a filter whose impulse response is of *finite* duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely.

An FIR filter is designed by finding the coefficients and filter order that meet certain specifications, which can be in the time domain (e.g. a matched filter) and/or the frequency domain (most common). Matched filters perform a cross-correlation between the input signal and a known pulse shape. The FIR convolution is a cross-correlation between the input signal and a time-reversed copy of the impulse response. Therefore, the matched filter's impulse response is "designed" by sampling the known pulse-shape and using those samples in reverse order as the coefficients of the filter.

Window design method:-

In the window design method, one first designs an ideal IIR filter and then truncates the infinite impulse response by multiplying it with a finite length window function. The result is a finite impulse response filter whose frequency response is modified from that of the IIR filter. Multiplying the infinite impulse by the window function in the time domain results in the frequency response of the IIR being convolved with the Fourier transform (or DTFT) of the window function. If the window's main lobe is narrow, the composite frequency response remains close to that of the ideal IIR filter.

The ideal response is usually rectangular, and the corresponding IIR is a sinc function. The result of the frequency domain convolution is that the edges of the rectangle are tapered, and ripples appear in the passband and stopband. Working backward, one can specify the slope (or width) of the tapered region (*transition band*) and the height of the ripples, and thereby derive the frequency domain parameters of an appropriate window function. Continuing backward to an impulse response can be done by iterating a filter design program to find the minimum filter order.

The Rectangular Window

The rectangular window may be defined by

$$w_R(n) \triangleq \begin{cases} 1, & -\frac{M-1}{2} \leq n \leq \frac{M-1}{2} \\ 0, & \text{otherwise} \end{cases}$$

where M is the window length in samples.

The Hanning window

The following equation generates the coefficients of a Hann window:

$$w(n) = 0.5\left(1 - \cos\left(2\pi \frac{n}{N}\right)\right), \quad 0 \leq n \leq N.$$

The window length $L = N + 1$.

The Hamming window

The following equation generates the coefficients of a Hamming window:

$$w(n) = 0.54 - 0.46\cos\left(2\pi \frac{n}{N}\right), \quad 0 \le n \le N.$$

The window length $L = N + 1$.

The Bartlett window

The following equation generates the coefficients of a Bartlett window:

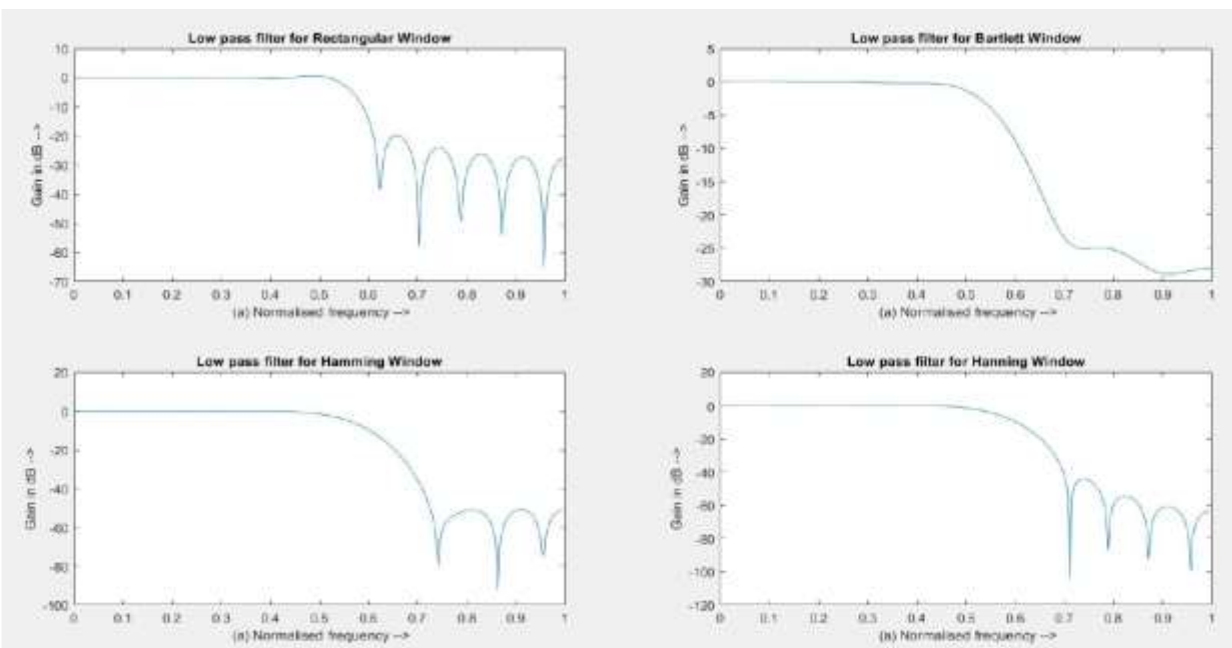
$$w(n) = \begin{cases} \frac{2n}{N}, & 0 \le n \le \frac{N}{2}, \\ 2 - \frac{2n}{N}, & \frac{N}{2} \le n \le N. \end{cases}$$

The window length L=N+1.

Code:

```
clc;
clear all;
close all;
rp = input('enter the passband ripple');
rs = input('enter the stopband ripple');
fp = input('enter the passband freq');
fs = input('enter the stopband freq');
f = input('enter the sampling freq');
wp = 2*fp/f;
ws = 2*fs/f;
num = -20*log10(sqrt(rp*rs))-13;
dem = 14.6*(fs-fp)/f;
n = ceil(num/dem);
n1 = n+1;
if(rem(n,2)~=0)
n1 = n;
n = n-1;
end
y_rec = boxcar(n1);
y_barlett = bartlett(n1);
y_hamming = hamming(n1);
y_hanning = hanning(n1);
%low pass filter for Rectangular Window
b = fir1(n,wp,y_rec);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
title('Low pass filter for Rectangular  Window');
ylabel('Gain in dB -->');
xlabel(' (a) Normalised frequency -->');
%low pass filter for Bartlett Window
```

```
b = fir1(n,wp,y_barlett);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,2);
plot(o/pi,m);
title('Low pass filter for Bartlett Window');
ylabel('Gain in dB -->');
xlabel(' (a) Normalised frequency -->');
%low pass filter for Hamming Window
b = fir1(n,wp,y_hamming);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
title('Low pass filter for Hamming Window');
ylabel('Gain in dB -->');
xlabel(' (a) Normalised frequency -->');
%low pass filter for Hanning Window
b = fir1(n,wp,y_hanning);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,4);
plot(o/pi,m);
title('Low pass filter for Hanning Window');
ylabel('Gain in dB -->');
xlabel(' (a) Normalised frequency -->');
```



Experiment-7

Aim:- Write a MATLAB program to

a) Implement a Low pass / High pass / Band pass / Band stop IIR Filter using Butterworth approximation.

b) Implement a Low pass / High pass / Band pass / Band stop IIR Filter using Chebyshev approximation.

Software Used:- MATLABR2016

Theory:-

Butterworth filter

The Butterworth filter is a type of signal processing filter designed to have a frequency response as flat as possible in the passband. It is also referred to as a maximally flat magnitude filter.

Properties of the Butterworth filter are:

1. monotonic amplitude response in both passband and stopband
2. Quick roll-off around the cutoff frequency, which improves with increasing order
3. Slightly non-linear phase response
4. Group delay largely frequency-dependent

Chebyshev filter

Chebyshev filters are analog or digital filters having a steeper roll-off than Butterworth filters, and have passband ripple (type I) or stopband ripple (type II). Chebyshev filters have the property that they minimize the error between the idealized and the actual filter.

I. Type-I Chebyshev Filters

This type of filter is the basic type of Chebyshev filter. The amplitude or the gain response is an angular frequency function of the nth order of the LPF (low pass filter) is equal to the total value of the transfer function Hn (jw)

$$G_n(\omega) = |H_n(j\omega)| = \frac{1}{\sqrt{1+\varepsilon^2 T_n^2\left(\frac{\omega}{\omega_0}\right)}}$$

II. Type-II Chebyshev Filter

The type II Chebyshev filter is also known as an inverse filter, this type of filter is less common. Because, it doesn't roll off and needs various components. It has no ripple in the passband, but it has equiripple in the stopband. The gain of the type II Chebyshev filter is

$$G_n(\omega, \omega_0) = \frac{1}{\sqrt{1+\frac{1}{\varepsilon^2 T_n^2(\omega_0/\omega)}}}.$$

1. Low pass filter

low-pass filter (**LPF**) is a filter that passes signals with a frequency lower than a selected cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency. The exact frequency response of the filter depends on the filter design. The filter is sometimes called a **high-cut filter**, or **treble-cut filter** in audio applications. A low-pass filter is the complement of a high-pass filter.

2. High pass filter

A **high-pass filter** (**HPF**) is an electronic filter that passes signals with a frequency higher than a certain cutoff frequency and attenuates signals with frequencies lower than the cutoff frequency. The amount of attenuation for each frequency depends on the filter design. A high-pass filter is usually modeled as a linear time-invariant system. It is sometimes called a **low-cut filter** or **bass-cut filter**. High-pass filters have many uses, such as blocking DC from circuitry sensitive to non-zero average voltages or radio frequency devices.

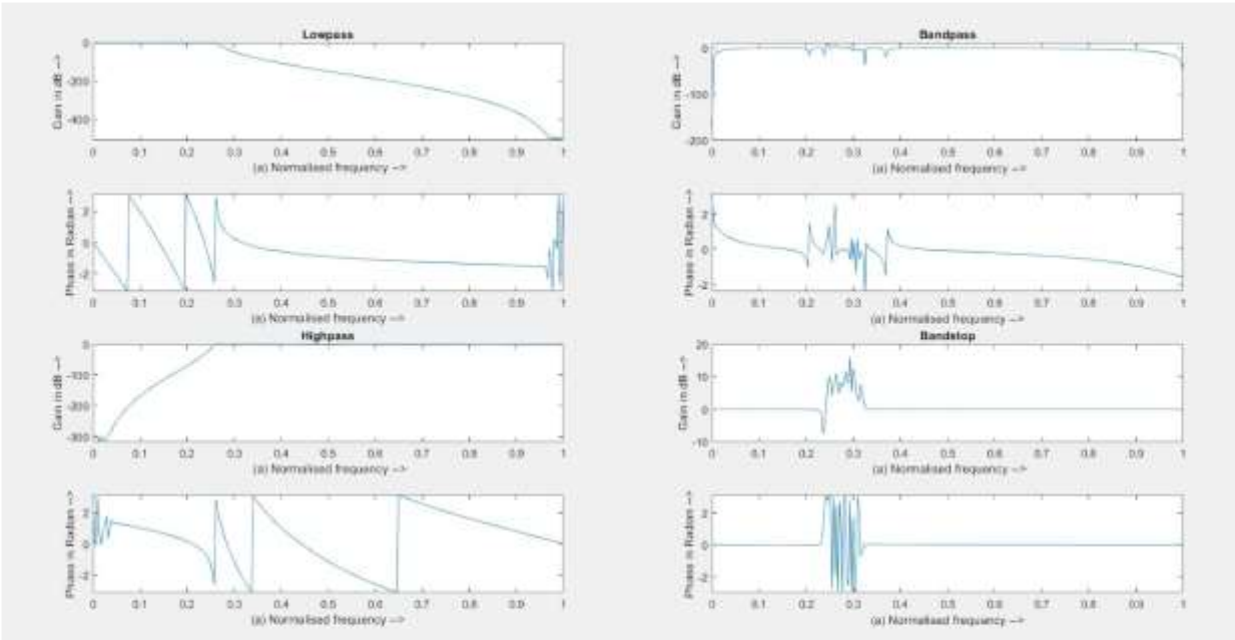
3. Band stop filter

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. It is the opposite of a band-pass filter. A **notch filter** is a band-stop filter with a narrow stopband (high Q factor).

Code:-

```
clc;
clear all;
close all;
format long
rp = input('enter the passband ripple');
rs = input('enter the stopband ripple');
wp  = input('enter the passband freq');
ws = input('enter the stopband freq');
fs = input('enter the sampling freq');
w1 = 2*wp/fs;
w2 = 2*ws/fs;
[n, wn] = buttord(w1,w2,rp,rs);
% Low pass
[b,a] = butter(n,wn);
w = 0:.01:pi;
[h,om] = freqz(b,a,w);
m = 20*log10(abs(h));
an = angle(h);
subplot(4,2,1);
plot(om/pi,m);
title('Lowpass');
ylabel('Gain in dB -->');
xlabel(' (a) Normalised frequency -->');
subplot(4,2,3);
plot(om/pi,an);
xlabel( '(a) Normalised frequency -->');
ylabel('Phase in Radian -->');
% High Pass
[b,a] = butter(n,wn,'high');
w = 0:.01:pi;
```

```
[h,om] = freqz(b,a,w);
m = 20*log10(abs(h));
an = angle(h);
subplot(4,2,5);
plot(om/pi,m);
title('Highpass');
ylabel('Gain in dB -->');
xlabel(' (a) Normalised frequency -->');
subplot(4,2,7);
plot(om/pi,an);
xlabel( '(a) Normalised frequency -->');
ylabel('Phase in Radian -->');
[n] = buttord(w1,w2,rp,rs);
wn = [w1,w2];
% BandPass
[b,a] = butter(n,wn,'bandpass');
w = 0:.01:pi;
[h,om] = freqz(b,a,w);
m = 20*log10(abs(h));
an = angle(h);
subplot(4,2,2);
plot(om/pi,m);
title('Bandpass');
ylabel('Gain in dB -->');
xlabel(' (a) Normalised frequency -->');
subplot(4,2,4);
plot(om/pi,an);
xlabel( '(a) Normalised frequency -->');
ylabel('Phase in Radian -->');
[n] = buttord(w1,w2,rp,rs);
wn = [w1,w2];
% BandStop
[b,a] = butter(n,wn,'stop');
w = 0:.01:pi;
[h,om] = freqz(b,a,w);
m = 20*log10(abs(h));
an = angle(h);
subplot(4,2,6);
plot(om/pi,m);
title('Bandstop');
ylabel('Gain in dB -->');
xlabel(' (a) Normalised frequency -->');
subplot(4,2,8);
plot(om/pi,an);
xlabel( '(a) Normalised frequency -->');
ylabel('Phase in Radian -->');
```


Lowpass
Bandpass
Highpass
Bandstop
Gain in dB -->
Phase in Radian -->
(a) Normalised frequency -->

EXPERIMENT :8

Aim: To design low pass and band pass Chebychev filter using MATLAB

Software Used: MATLAB

Theory:

Chebyshev filters are used for distinct frequencies of one band from another. They cannot match the windows-sink filter's performance and they are suitable for many applications. The main feature of Chebyshev filters is their speed, normally faster than the windowed-sinc. Because these filters are carried out by recursion rather than convolution. The designing of the Chebyshev and Windowed-Sinc filters depends on a mathematical technique called as the Z-transform.

Types of Chebyshev Filters

Chebyshev filters are classified into two types, namely type-I Chebyshev filter and type-II Chebyshev filter.

Type-I Chebyshev Filters

This type of filter is the basic type of Chebyshev filter. The amplitude or the gain response is an angular frequency function of the nth order of the LPF (low pass filter) is equal to the total value of the transfer function Hn (jw)

Gn(w)=|Hn (jω)|=1√(1+ϵ2Tn2() ω/ωo)

Where,ε = ripple factor
ωo= cutoff frequency
Tn= Chebyshev polynomial of the nth order
The pass-band shows equiripple performance. In this band, the filter interchanges between -1 & 1 so the gain of the filter interchanges between max at G = 1 and min at G =1/√(1+ε2) . At the cutoff frequency, the gain has the value of 1/√(1+ε2) and remains to fail into the stop band as the frequency increases. The behavior of the filter is shown below. The cutoff frequency at -3dB is generally not applied to Chebyshev filters.

Type-II Chebyshev Filter

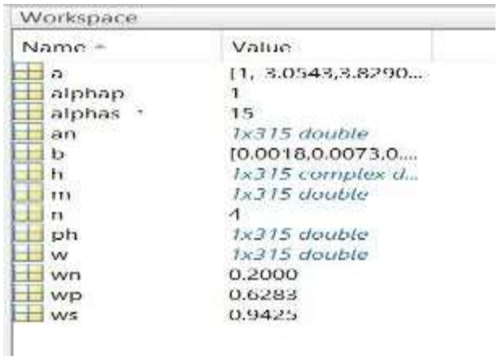
The type II Chebyshev filter is also known as an inverse filter, this type of filter is less common. Because, it doesn't roll off and needs various components. It has no ripple in the passband, but it has equiripple in the stopband. The gain of the type II Chebyshev filter is

$$G_n(\omega, \omega_0) = \frac{1}{\sqrt{1 + \frac{1}{\varepsilon^2 T_n^2(\omega_0/\omega)}}}.$$

In the stopband, the Chebyshev polynomial interchanges between -1& and 1 so that the gain 'G' will interchange between zero and one.

CODE AND OUTPUT:

```
clc
clear all;
close all;
alphap=1;
alphas=15;
wp=.2*pi;
ws=.3*pi;
[n,wn]=cheb1ord(wp/pi,ws/pi,alphap,alphas;
[b,a]=cheby1(n,alphap,wn);
w=0:.01:pi;
[h,ph]=freqz(b,a,w);
m=20*log(abs(h));
an=angle(h);
subplot(2,1,1);
plot(ph/pi,m);
grid on;
ylabel('Gain in db');
xlabel('Normalised Frequency');
subplot(2,1,2);
plot(ph/pi,an);
grid on;
ylabel('Phase in radians');
xlabel('Normalised Frequency');
```

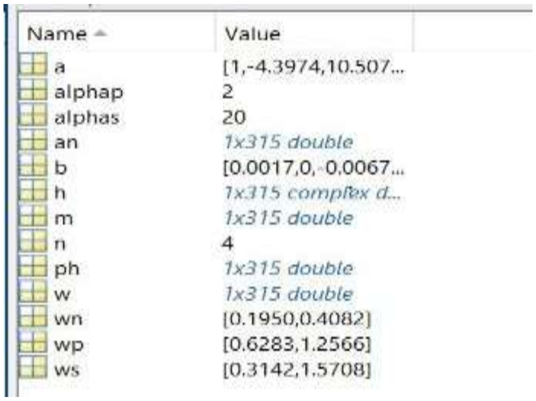


Workspace

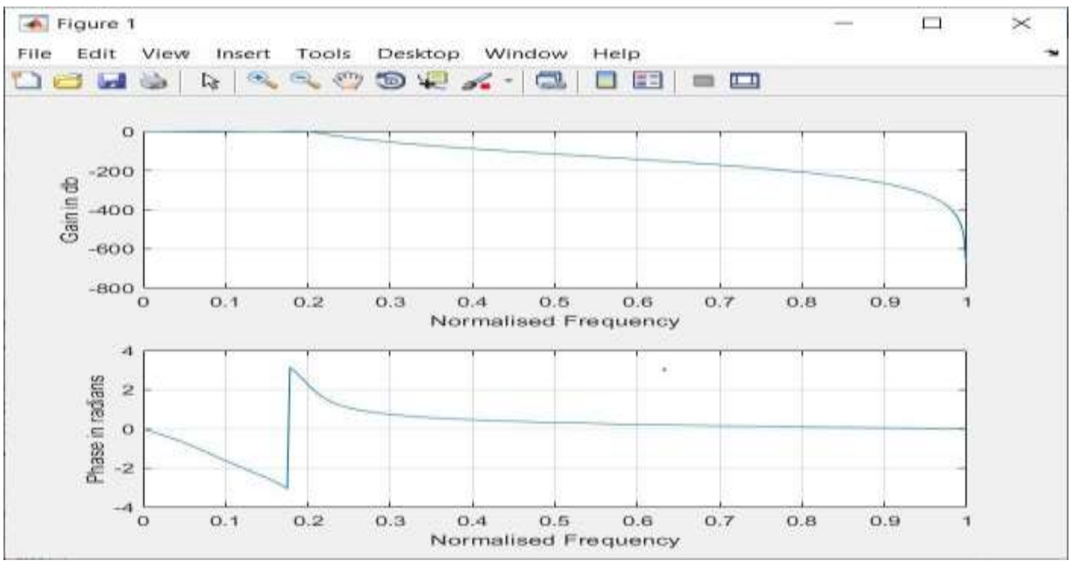
Name	Value
a	[1, 3.0543,3.8290...
alphap	1
alphas	15
an	1x315 double
b	[0.0018,0.0073,0....
h	1x315 complex d...
m	1x315 double
n	4
ph	1x315 double
w	1x315 double
wn	0.2000
wp	0.6283
ws	0.9425

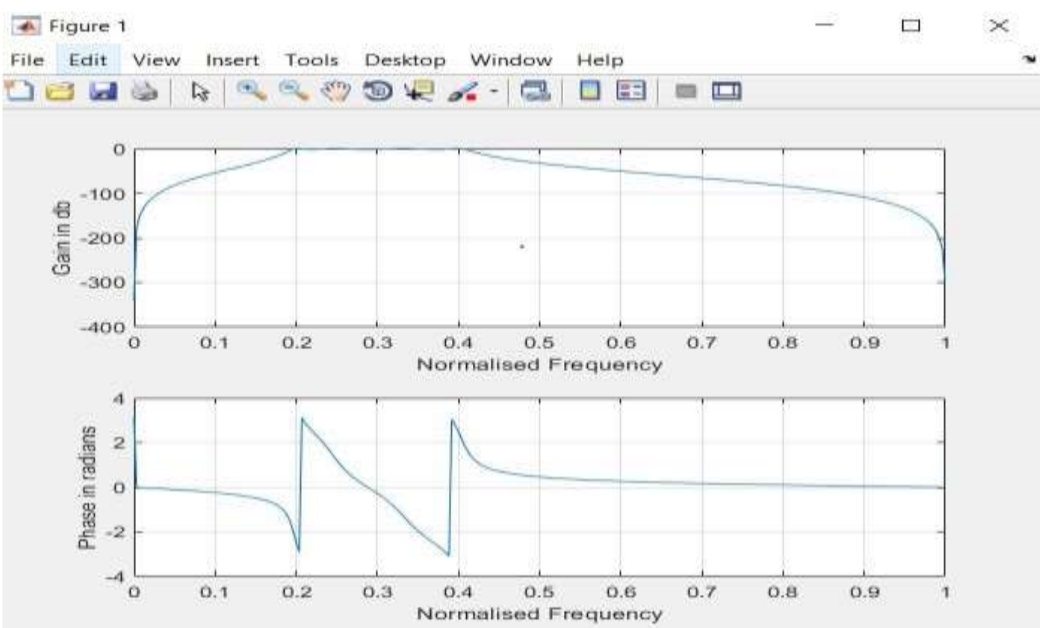
```
clc
clear all;
close all;
alphap=2;
alphas=20;
wp=[.2*pi,.4*pi];
ws=[.1*pi,.5*pi];
[n,wn]=buttord(wp/pi,ws/pi,alphap,alphas);
[b,a]=cheby1(n,alphap,wn);
w=0:.01:pi;
[h,ph]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
subplot(2,1,1);
plot(ph/pi,m);
grid on;
ylabel('Gain in db');
xlabel('Normalised Frequency');
subplot(2,1,2);
plot(ph/pi,an);
grid on;
ylabel('Phase in radians');
```

xlabel('Normalised Frequency');



Name	Value
a	[1,-4.3974,10.507...
alphap	2
alphas	20
an	1x315 double
b	[0.0017,0,-0.0067...
h	1x315 complex d...
m	1x315 double
n	4
ph	1x315 double
w	1x315 double
wn	[0.1950,0.4082]
wp	[0.6283,1.2566]
ws	[0.3142,1.5708]





DSP INNOVATION

Design of filter using Gaussian window.

The Fourier transform of a Gaussian is also a Gaussian (it is an eigenfunction of the Fourier transform). Since the Gaussian function extends to infinity, it must either be truncated at the ends of the window, or itself windowed with another zero-ended window.

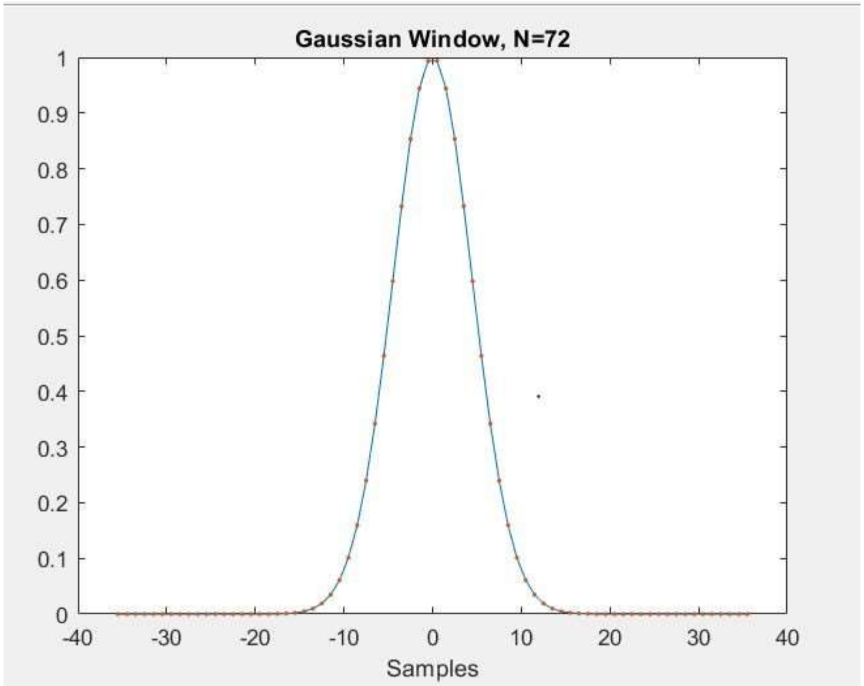
Since the log of a Gaussian produces a parabola, this can be used for nearly exact quadratic interpolation in frequency estimation.

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-t^2/2\sigma^2} \longleftrightarrow e^{-\omega^2/2(1/\sigma)^2}$$

CODE:

```
clc;
clear all;
close all;
N=72;
n=-(N-1)/2:(N-1)/2;
alpha=8;
y=gausswin(N,alpha);
sigma=(N-1)/(2*alpha);
y=exp(-1/2*(n/sigma).^2);
plot(n,y);
hold on;
```

```
plot(n,y,'.');
hold off;
xlabel('Samples');
title('Gaussian Window, N=72');
```



Application :

• Window functions are used in spectral analysis/modification/resynthesis, the design of finite impulse response filters, as well as beamforming and antenna design.

• Fourier transform of the Gaussian window is also Gaussian with a reciprocal standard deviation. This is an illustration of the time-frequency uncertainty principle .