Department of Electronics and Communication Engineering

Sub Code/Name: **BEC5L1- DIGITAL SIGNAL PROCESSING LABORATORY**

Name : ……

Reg No : ……

Branch : ……

Year & Semester : ……

LIST OF EXPERIMENTS

INDEX

Expt.	Expt.	Name of the Experiment	Marks	Staff SIGN

Ex No:1(a)

Date:

WAVEFORM GENERATION

CONTINUOUS TIME SIGNAL

Aim

To Generate a continuous sinusoidal time signals Using MATLAB.

Requirements

Matlab 2007 SOFTWARE

Procedure

1. OPEN MATLAB
2. File ⟶ New ⟶ Script.

a. Type the program in untitled window

3. File ⟶ Save ⟶ type filename.m in matlab workspace path
4. Debug ⟶ Run. Wave will displayed at Figure dialog box.

Theory

Common Periodic Waveforms

The toolbox provides functions for generating widely used periodic waveforms:sawtooth generates a sawtooth wave with peaks at ±1 and a period of 2π. An optional width parameter specifies a fractional multiple of 2π at which the signal's maximum occurs. square generates a square wave with a period of 2π. An optional parameter specifies duty cycle, the percent of the period for which the signal is positive.

Common Aperiodic Waveforms

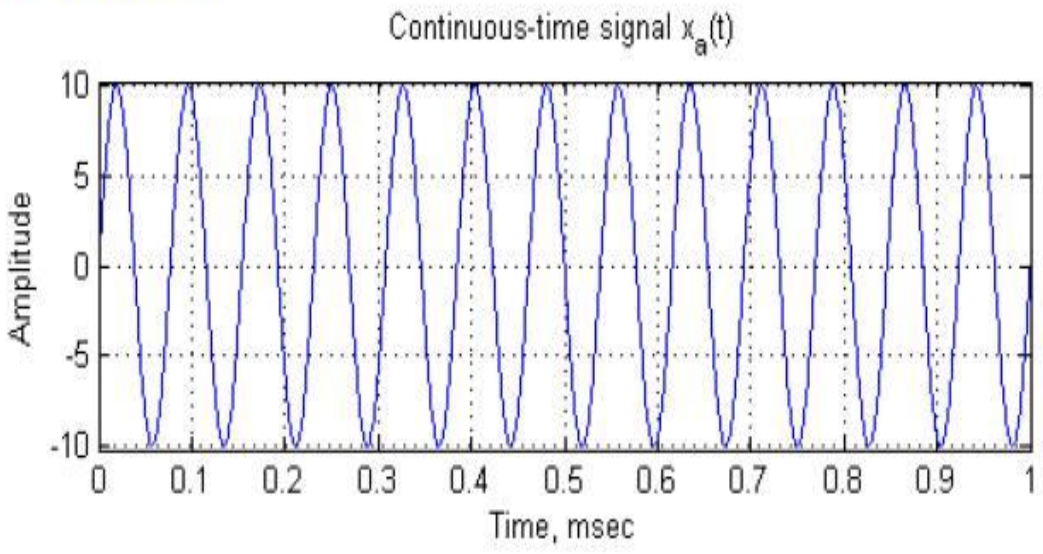
The toolbox also provides functions for generating several widely used aperiodic waveforms: gauspuls generates a Gaussian-modulated sinusoidal pulse with a specified time, center frequency, and fractional bandwidth. Optional parameters return in-phase and Quadrature pulses, the RF signal envelope, and the cutoff time for the trailing pulse envelope. chirp generates a linear, log, or quadratic swept-frequency cosine signal. An optional parameter specifies alternative sweep methods. An optional parameter phi allows initial phase to be specified in degrees.

Program %

% Assuming The Sampling frequency is 5 Mhz

clc;
clear all;
t = 0:0.0005:1;
a = 10
f = 13;
xa = a*sin(2*pi*f*t);
subplot(2,1,1)
plot(t,xa);grid
xlabel('Time, msec');
ylabel('Amplitude');
title('Continuous-time signal x_{a}(t)');
axis([0 1 -10.2 10.2])
	clear all;
	Finput = 1000;
	Fsampling = 5000000;
	Tsampling = 1 / Fsampling;
	Nsample = Fsampling/ Finput;
	N = 0:5*Nsample-1;
	x=sin(2 * pi * Finput * Tsampling * N);
	plot(x); title('Sine Wave Generation');
	xlabel('Time -- >');
	ylabel('Amplitude-- >');
	grid on;

EXPECTED GRAPH:



Result:

Thus the Continuous Time Signal was generated using MATLAB.

Ex No:1(b)

Date:

DISCRETE TIME SIGNAL

Aim

To Generate a Discrete time Exponential signals Using MATLAB.

Requirements

Matlab 2007

Personal computer

Procedure

1. OPEN MATLAB
2. File ⟶ New ⟶ Script.

a. Type the program in untitled window

3. File ⟶ Save ⟶ type filename.m in matlab workspace path
4. Debug ⟶ Run. Wave will displayed at Figure dialog box.

Theory:

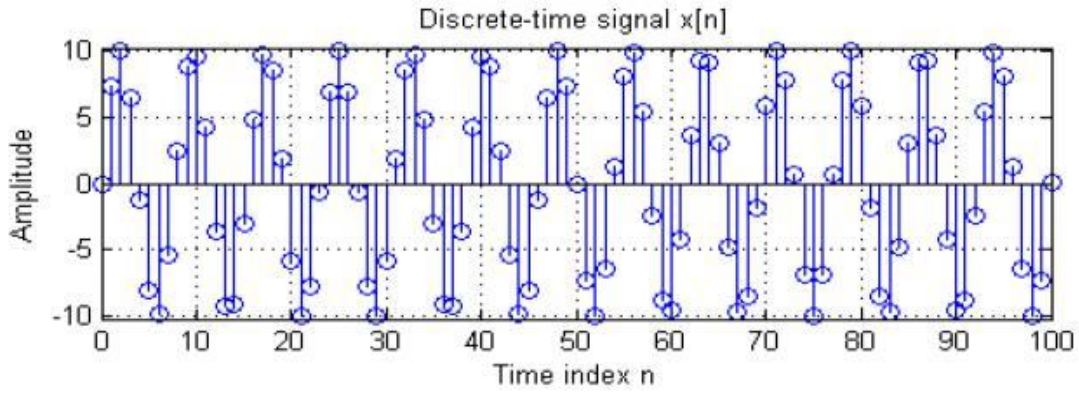
Program

```
clear all;

a =
10;
f =
13;
T =
0.01;
n =
0:T:1;
```

```
xs =
a*sin(2*pi*f*n); k
= 0:length(n)-1;
stem(k,xs);
grid
xlabel('Time index n');
ylabel('Amplitude');
title('Discrete-time signal
x[n]'); axis([0 (length(n)-1) -
10.2 10.2])
```

Expected Graph:



Result

Thus the Discrete Time Signal was generated using MATLAB.

Ex No:2

Date:

SAMPLING AND EFFECT OF ALIASING

Aim

To perform a Sampling and effect of aliasing Using MATLAB.

Requirements

Matlab 2007 later

Procedure

1. OPEN MATLAB

2. File → New → Script.

a. Type the program in untitled window

3. File → Save → type filename.m in matlab workspace path

4. DebugRun. Wave will displayed at Figure dialog box.

Program

```
clc;
close
all;
clear
all;
      t1=0:.0005:.5115;
      Hz 1=(2000/2)*(0: 1024/2) / (lO24/2);
      xl=sin(2*pi*550*tl );
      Xl=abs(fft(xl )):
       Xl(514:1024)=[ ];
      subplot(211) ;
```

```
plot( tl( 1:64),xl(1 :64));
subplot(212);
plot(Hz 1 ,Xl);
```
(Fig. 2 shows MATLAB plots for xl and Xl)

```
t2=0:.00l:1.023;
Hz2=( 1000/2)*(0: 1024/2) / (lO24/2);
x2=sin( 2*pi * 550*t2);
X2=abs(fft(x2));
X2(514:1024)=[ ];
subplot(21l);
plot( t2(1 :256),x2( 1:256));
 subplot(212);
plot(Hz2,X2);
```
(Fig . 3 shows MATLAB plots for x2 and X2)

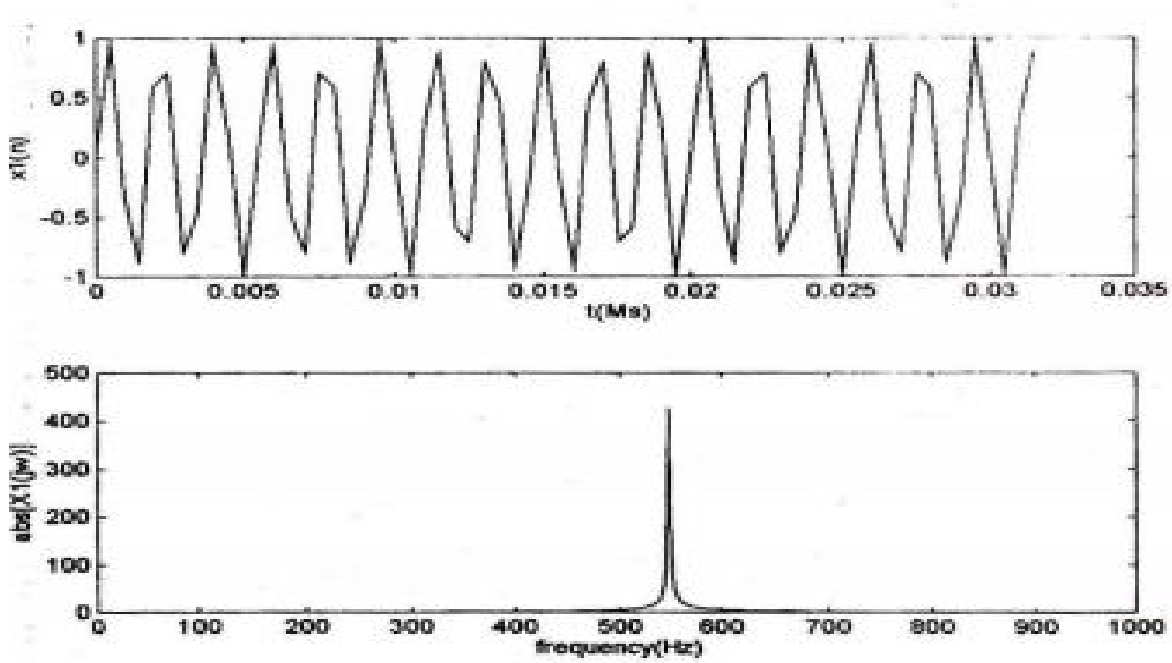


Fig.2 $550Hz$ sine wave sampled at $f_s = 2,000Hz$ and its FFT.

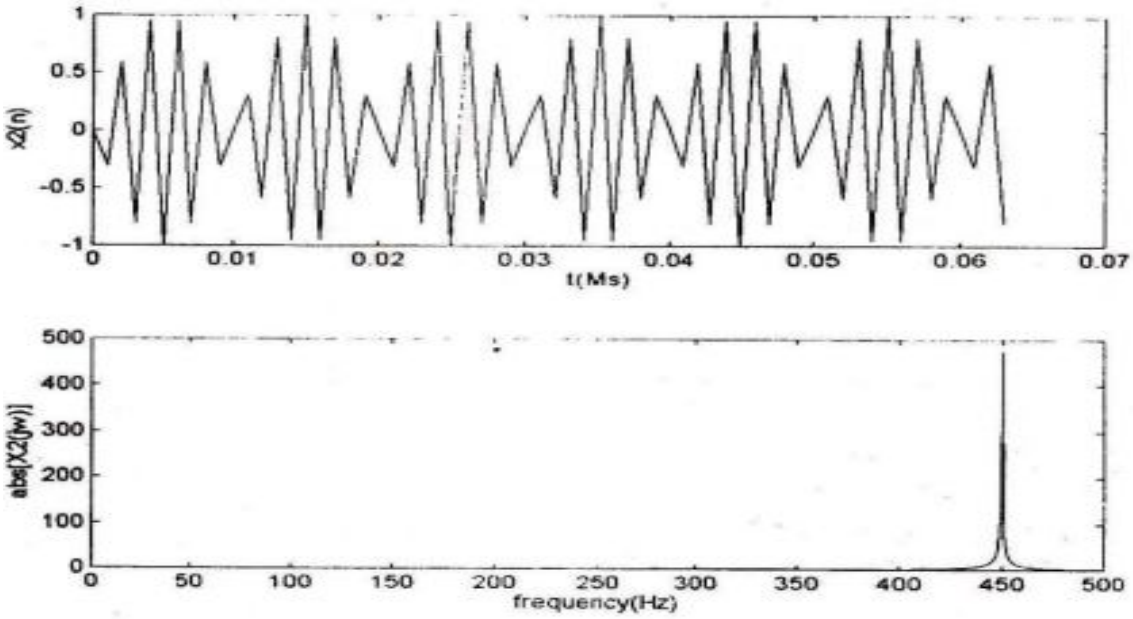


Fig.3 $550Hz$ **sine wave sampled at** $f_s = 1,000Hz$ **and its FFT.**

The Fourier expansion contains only odd harmonics, the amplitude of which drop as *l/n* (where n=harmonic #) and is infinite. This means that inherently any representation of a square wave by a discrete sequence will result in aliasing. To illustrate this using MATLAB, we synthesize a square wave, compute and plot the magnitude of its FFT and compare the results to the predicted aliased components. We will synthesize a *30Hz* square wave with 50% duty cycle, consisting of 1024 points, sampled at a frequency of *1000Hz*. The following are the commands in MATLAB needed to synthesize a square wave, compute its FFT and plot the results:

```
x=square(2*pi*30*t2,50);
X=abs(fft(x));
X(514:1024)=[ ];
plot(Hz2,X);
```

where t2 and Hz2 are those generated in the previous example and "square" is a command from the Signal Processing Toolbox .

Fig 4 shows the graphs generated by MATLAB.

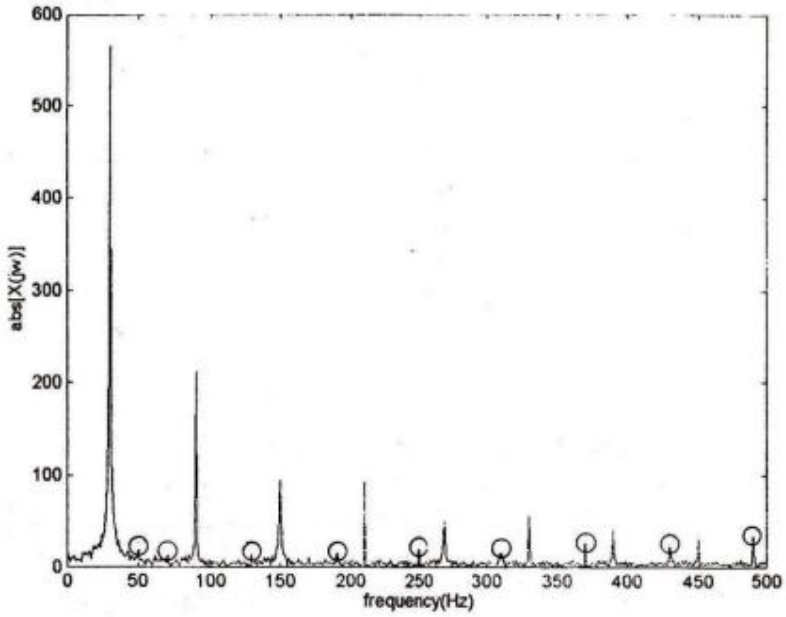


Fig. 4 FFT of a $30Hz$ square wave.

The first harmonic of the square wave is $30Hz$, the 2nd is $90Hz$, the 3rd $150Hz$, etc. Since the sampling rate is $f_s = 1000Hz$, we should expect that all the harmonics above $f_s/2 = 500Hz$ will be aliased into the range *DC to* $500Hz$.

Result:

Thus the Sampling was performed and studied the aliasing effect using MATLAB.

Ex No:3(a)

Date:

LINEAR AND CIRCULAR CONVOLUTION

(LINEAR CONVOLUTION)

Aim

To perform a Linear Convolution Using MATLAB.

Requirements

Matlab 2007 later

Procedure

1. OPEN MATLAB

2. File ⟶ New ⟶ Script.

a. Type the program in untitled window
3. File ⟶ Save ⟶ type filename.m in matlab workspace path

4. DebugRun. Wave will displayed at Figure dialog box.

Program

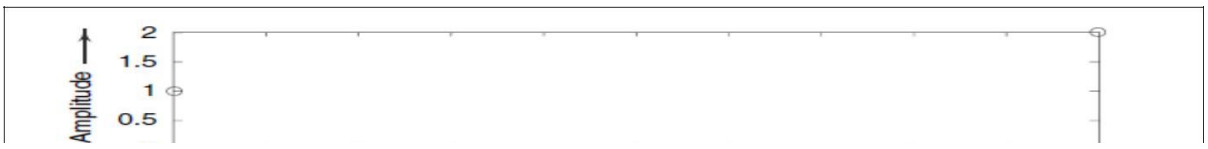
% Program for linear convolution of the sequence x5[1, 2] and h5[1, 2, 4]

```
clc;
clear
all;
close
all;
x=input('enter the 1st sequence');
h=input('enter the 2nd sequence');
y=conv(x,h);
figure;
subplot(3,1,1);
stem(x);
 ylabel('Amplitude --.');
xlabel('(a) n --.');
title('first sequence');
 subplot(3,1,2);
stem(h);
ylabel('Amplitude --.');
 xlabel('(b) n --.');
title('Second sequence');
subplot(3,1,3);
stem(y);
ylabel('Amplitude --.');
xlabel('(c) n --.');
title('Convoluted sequence');
disp('The resultant signal is');
```

Output:

```
enter the 1st sequence [1 2]
enter the 2nd sequence [1 2 4]
 The resultant signal is
Y= 1 4 8 8
```

EXPECTED GRAPHS:



Result

Thus the Linear convolution was performed using MATLAB.

Ex No:3(b)

Date:

CIRCULAR CONVOLUTION

Aim

To perform a Circular Convolution Using MATLAB.

Requirements

Matlab 2007 later

Procedure

1. OPEN MATLAB

2. File ⟶ New ⟶ Script.
a. Type the program in untitled window
3. File ⟶ Save ⟶ type filename.m in matlab workspace path

4. Debug ⟶ Run. Wave will displayed at Figure dialog box.

Program

```
clc; clear all;
a = input(enter the sequence x(n) = ‟);
b = input(„enter the sequence h(n) = ‟);
n1=length(a);
n2=length(b);
N=max(n1,n2);
x = [a zeros(1,(N-n1))];
for i = 1:N
k = i;
for j = 1:n2 H(i,j)=x(k)* b(j);
 k = k-1;
if (k == 0) k = N;
end
end
end
y=zeros(1,N);
M=H‟;for j = 1:N for i = 1:n2
y(j)=M(i,j)+y(j);
end
end
disp(„The output sequence is y(n)= „);
 disp(y);
stem(y);
title(„Circular Convolution‟);
 xlabel(„n‟);
ylabel(‚y(n)„);
```

OUTPUT:

```
enter the sequence x(n) = [1 2 4]
enter the sequence h(n) = [1 2]
The output sequence is y(n)= 9 4 8
```

% Program for Computing Circular Convolution with zero padding

```
clc;
close all;
```

```
clear all;
x=input('enter the first sequence');
h=input('enter the 2nd sequence');
y=x'*h;
n1=length(x);
n2=length(h);
figure subplot(3,1,1) stem(x);
title('Input sequence');
xlabel('n1');
ylabel('x(n1)');
subplot(3,1,2) stem(h);
title('Impulse sequence');
xlabel('n2');
ylabel('h(n2)');
n=n1+n2-1;
c=zeros(n);
 for i=1:n1
    for j=1:n2 c(i+j-1)=c(i+j-1)+y(i,j);
    end
    end
for i=1:n d(i)=c(i,1);
end
disp('convoluted sequence');
disp(d);
n3=1:n;
subplot(3,1,3) stem(n3-1,c);
title('Convoluted sequence');
xlabel('time');
ylabel('Amplitude');
```

OUTPUT:

enter the first sequence [1 2 4]
enter the 2nd sequence [1 2]
The resultant signal is y=1 4 8 8

Result

Thus the Circular convolution was performed using MATLAB.

Ex No:4

Date:

DISCRETE FOURIER TRANSFORM (DFT) COMPUTATIONS

AIM : To find the Discrete Fourier Transform of a sequence.

SOFTWARE REQUIRED : MAT LAB 7.0

PROGRAM DESCRIPTION : In this program the Discrete Fourier Transform (DFT) of a sequence x[n] is generated by using the formula,

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-2\pi jk / N}$$ Where, X(k) → DFT of sequence x[n]

N represents the sequence length and it is calculated by using the command 'length'. The DFT of any sequence is the powerful computational tool for performing frequency analysis of discrete-time signals.

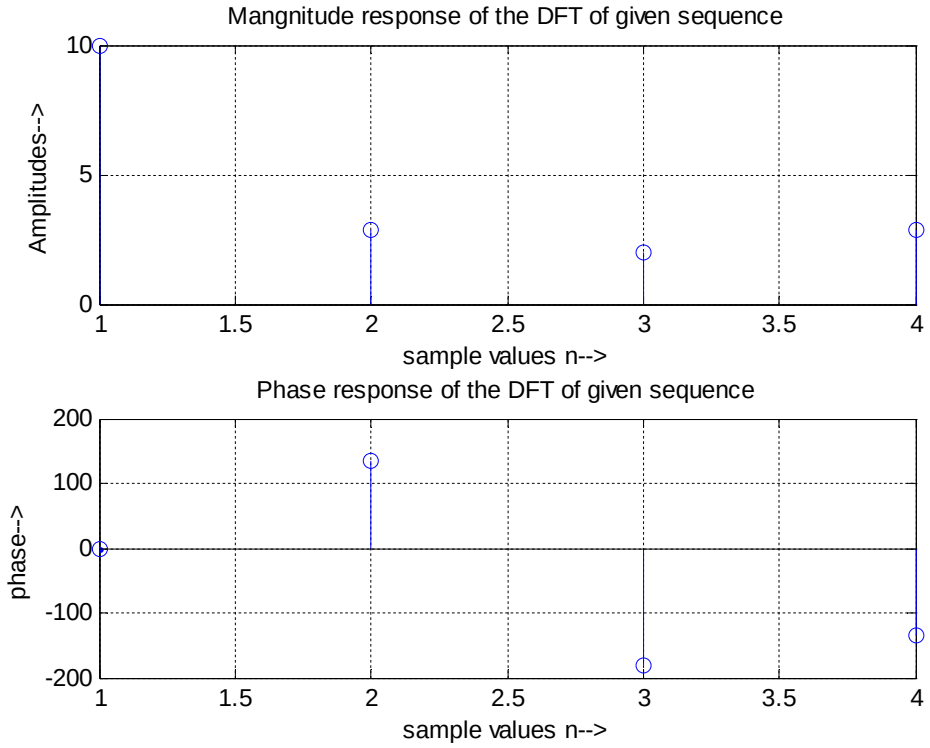
MATLAB CODE :

```
clc;
clear all;
close all;
a=input('Enter the sequence :');
N=length(a);
disp('The length of the sequence is:');N
for k=1:N
   y(k)=0;
   for i=1:N
      y(k)=y(k)+a(i)*exp((-2*pi*j/N)*((i-1)*(k-1)));
   end;
end;
k=1:N
disp('The result is:');y
figure(1);
subplot(211);
stem(k,abs(y(k)));
grid;
xlabel('sample values n-->');
ylabel('Amplitudes-->');
title('Mangnitude response of the DFT of given sequence');
subplot(212);
stem(angle(y(k))*180/pi);
grid;
xlabel('sample values n-->');
ylabel('phase-->');
title('Phase response of the DFT of given sequence');
```

OUTPUTS:

```
Enter the sequence : [1 2 3 4]
The length of the sequence is:  N =  4
k =     1     2     3     4
The result is: y =  10.0000     -2.0000 + 2.0000i     -2.0000 - 0.0000i      -2.0000 - 2.0000i
```

GRAPHS :



INFERENCE : To perform the frequency analysis on a discrete-time signal x[n] can be generated from a continuous signal x(t). Here in the program y(k) refers to the DFT of the sequence a. The DFT consists of two parts. The magnitude and phase angle of x(k) are calculated by using abs and angle commands and plotted using stem command.

RESULT : The Discrete Fourier Transform (DFT) of a sequence is obtained and response is plotted.

Ex No:5

Date:

FINDING THE FFT OF DIFFERENT SIGNALS

AIM : To find the FFT of different signals like impulse, step, ramp and exponential.

SOFTWARE REQUIRED: MAT LAB 7.0

PROGRAM DESCRIPTION: In this program using the command FFT for impulse, step, ramp and exponential sequences the FFT is generated. In the process of finding the FFT the length of the FFT is taken as N. The FFT consists of two parts: MAGNITUDE PLOT and PHASE PLOT. The magnitude plot is the absolute value of magnitude versus the samples and the phase plot is the phase angle versus the samples.

MATLAB CODE:

```
% FFT of the impulse sequence : magnitude and phase response
clc;
clear all;
close all;
   %impulse sequence
   t=-2:1:2;
   y=[zeros(1,2) 1 zeros(1,2)];
   subplot (3,1,1);
   stem(t,y);
   grid;
   input('y=');
   disp(y);
   title ('Impulse Response');
   xlabel ('time -->');
   ylabel ('-->  Amplitude');
   xn=y;
   N=input('enter the length of the FFT sequence: ');
   xk=fft(xn,N);
   magxk=abs(xk);
   angxk=angle(xk);
   k=0:N-1;
   subplot(3,1,2);
   stem(k,magxk);
   grid;
   xlabel('k');
   ylabel('|x(k)|');
   subplot(3,1,3);
   stem(k,angxk);
   disp(xk);
   grid;
   xlabel('k');
   ylabel('arg(x(k))');
```

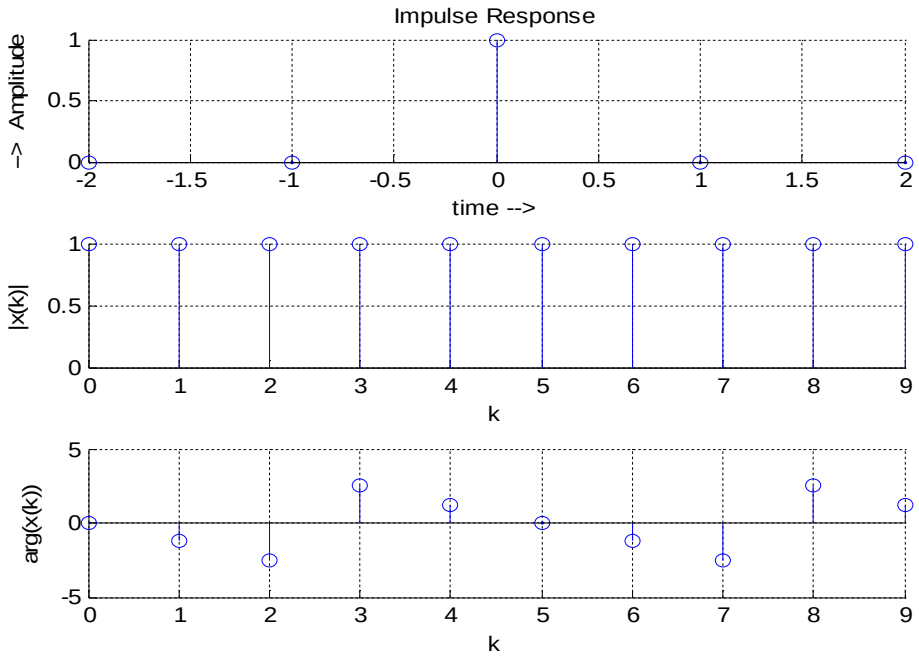
OUTPUTS:

```
y=     0     0     1     0     0
```

enter the length of the FFT sequence: 10

1.0000	0.3090 - 0.9511i	-0.8090 - 0.5878i	-0.8090 + 0.5878i	0.3090 + 0.9511i
1.0000	0.3090 - 0.9511i	-0.8090 - 0.5878i	-0.8090 + 0.5878i	0.3090 + 0.9511i

GRAPHS:



% FFT of the step sequence : magnitude and phase response

```
clc;
clear all;
close all;
   %Step Sequence
   s=input ('enter the length of step sequence');
   t=-s:1:s;
   y=[zeros(1,s) ones(1,1) ones(1,s)];
   subplot(3,1,1);
   stem(t,y);
   grid
   input('y=');
   disp(y);
   title ('Step Sequence');
   xlabel ('time -->');
   ylabel ('-->  Amplitude');
   xn=y;
   N=input('enter the length of the FFT sequence: ');
   xk=fft(xn,N);
   magxk=abs(xk);
```

```
angxk=angle(xk);
k=0:N-1;
subplot(3,1,2);
stem(k,magxk);
grid
xlabel('k');
ylabel('|x(k)|');
subplot(3,1,3);
stem(k,angxk);
disp(xk);
grid
xlabel('k');
ylabel('arg(x(k))');
```

OUTPUTS:

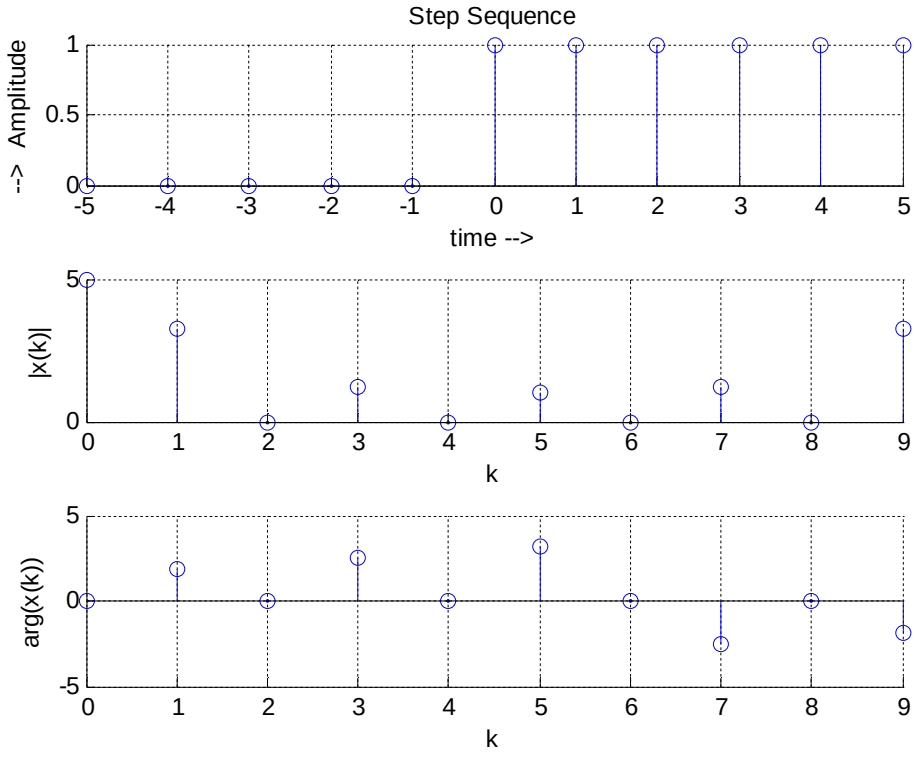
enter the length of step sequence: 5

y= 0 0 0 0 0 1 1 1 1 1 1

enter the length of the FFT sequence: 10

5.0000 -1.0000 + 3.0777i 0 -1.0000 + 0.7265i 0 -1.0000 0
-1.0000 - 0.7265i 0 -1.0000 - 3.0777i

GRAPHS:



% FFT of the Ramp sequence: magnitude and phase response

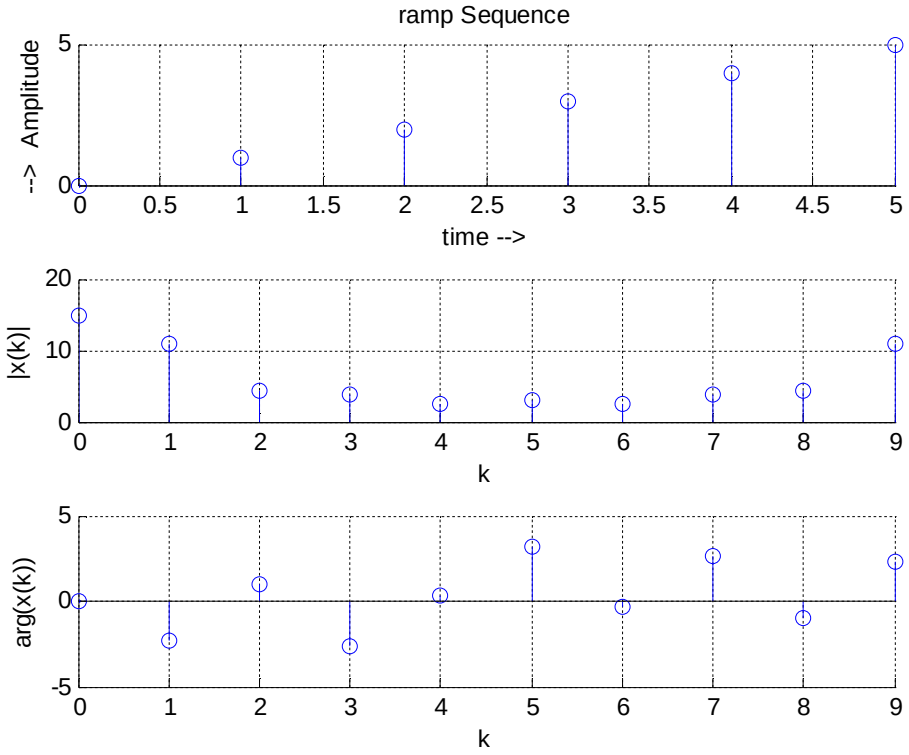
```
clc;
clear all;
close all;
   %Ramp Sequence
   s=input ('enter the length of Ramp sequence: ');
   t=0:s;
   y=t
   subplot(3,1,1);
   stem(t,y);
   grid
   input('y=');
   disp(y);
   title ('ramp Sequence');
   xlabel ('time -->');
   ylabel ('-->  Amplitude');
   xn=y;
   N=input('enter the legth of the FFT sequence: ');
   xk=fft(xn,N);
   magxk=abs(xk);
   angxk=angle(xk);
   k=0:N-1;
   subplot(3,1,2);
   stem(k,magxk);
   grid
   xlabel('k');
   ylabel('|x(k)|');
   subplot(3,1,3);
   stem(k,angxk);
   disp(xk);
   grid
   xlabel('k');
   ylabel('arg(x(k))');
```

OUTPUTS:

```
enter the length of Ramp sequence: 5
y =     0     1     2     3     4     5

enter the length of the FFT sequence: 10
15.0000     -7.7361 - 7.6942i     2.5000 + 3.4410i     -3.2639 - 1.8164i     2.5000 + 0.8123i
-3.0000     2.5000 - 0.8123i     -3.2639 + 1.8164i     2.5000 - 3.4410i     -7.7361 + 7.6942i
```

GRAPHS:



% FFT of the Exponential Sequence : magnitude and phase response

```
clc;
clear all;
close all;
%exponential sequence
n=input('enter the length of exponential sequence: ');
t=0:1:n;
a=input('enter "a" value: ');
y=exp(a*t);
input('y=')
disp(y);
subplot(3,1,1);
stem(t,y);
grid;
title('exponential response');
xlabel('time');
ylabel('amplitude');
   disp(y);
   xn=y;
   N=input('enter the length of the FFT sequence: ');
```

```
xk=fft(xn,N);
magxk=abs(xk);
angxk=angle(xk);
k=0:N-1;
subplot(3,1,2);
stem(k,magxk);
grid;
xlabel('k');
ylabel('|x(k)|');
subplot(3,1,3);
stem(k,angxk);
grid;
disp(xk);
 xlabel('k');
ylabel('arg(x(k))');
```

OUTPUTS:

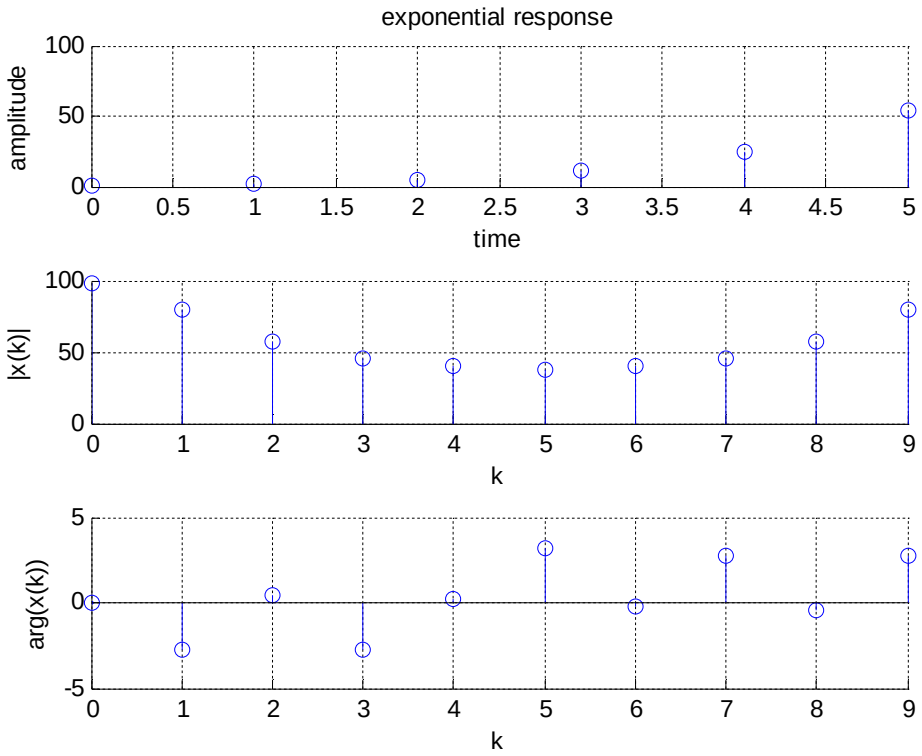
```
enter the length of exponential sequence: 5

enter "a" value: 0.8
y=    1.0000    2.2255    4.9530   11.0232   24.5325   54.5982

enter the length of the FFT sequence: 10

98.3324       -73.5207 -30.9223i    50.9418 +24.7831i     -41.7941 -16.0579i
38.8873 + 7.3387i       -37.3613          38.8873 - 7.3387i     -41.7941 +16.0579i          50.9418
-24.7831i       -73.5207 +30.9223i
```

GRAPHS :



INFERENCE : The FFT for impulse, step, ramp and exponential sequences is generated using the FFT command. The magnitude plot is the absolute value of magnitude versus the samples and the phase plot is the phase angle versus the samples is plotted for different signals for different values. This program is very simple and requires defining the signal and finding FFT and plotting.

RESULT : The FFT of different signals like impulse, step, ramp and exponential is found and the magnitude and phase plots of the same is plotted.

Ex No:6

Date:

IMPLEMENTATION OF LP & HP FIR FILTER FOR A GIVEN SEQUENCE (USING WINDOWING TECHNIQUES)

AIM: To implement the FIR filter for a given sequence by using windowing techniques.

SOFTWARE REQUIRED: MATLAB 7.0

PROGRAM DESCRIPTION: A Finite Impulse Response (FIR) filter is a discrete linear time-invariant system whose output is based on the weighted summation of a finite number of past inputs.

An FIR transversal filter structure can be obtained directly from the equation for discrete-time convolution.

$$y[n] = \sum_{k=0}^{N-1} X(k)\, h(n-k) \qquad 0 < n < N-1$$

In this equation, x(k) and y(n) represent the input to and output from the filter at time n. h(n-k) is the transversal filter coefficients at time n. These coefficients are generated by using FDS (Filter Design Software or Digital filter design package).

FIR – filter is a finite impulse response filter. Order of the filter should be specified. Infinite response is truncated to get finite impulse response. Placing a window of finite length does this. Types of windows available are Rectangular, Barlett, Hamming, Hanning, Blackmann window etc. This FIR filter is an all zero filter.

MATLAB CODE:

```
clc;
clear all;
close all;
rp=input('enter passband ripple');
rs=input('enter the stopband ripple');
fp=input('enter passband freq');
fs=input('enter stopband freq');
f=input('enter 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
c=input('enter your choice of window function 1. rectangular 2. triangular 3.kaiser: \n ');
if(c==1)
y=rectwin(n1);
disp('Rectangular window filter response');
end
if (c==2)
y=triang(n1);
disp('Triangular window filter response');
end
if(c==3)
y=kaiser(n1);
disp('kaiser window filter response');
end
%LPF
b=fir1(n,wp,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,1,1);plot(o/pi,m);
title('LPF');
ylabel('Gain in dB-->');
xlabel('(a) Normalized frequency-->');
```

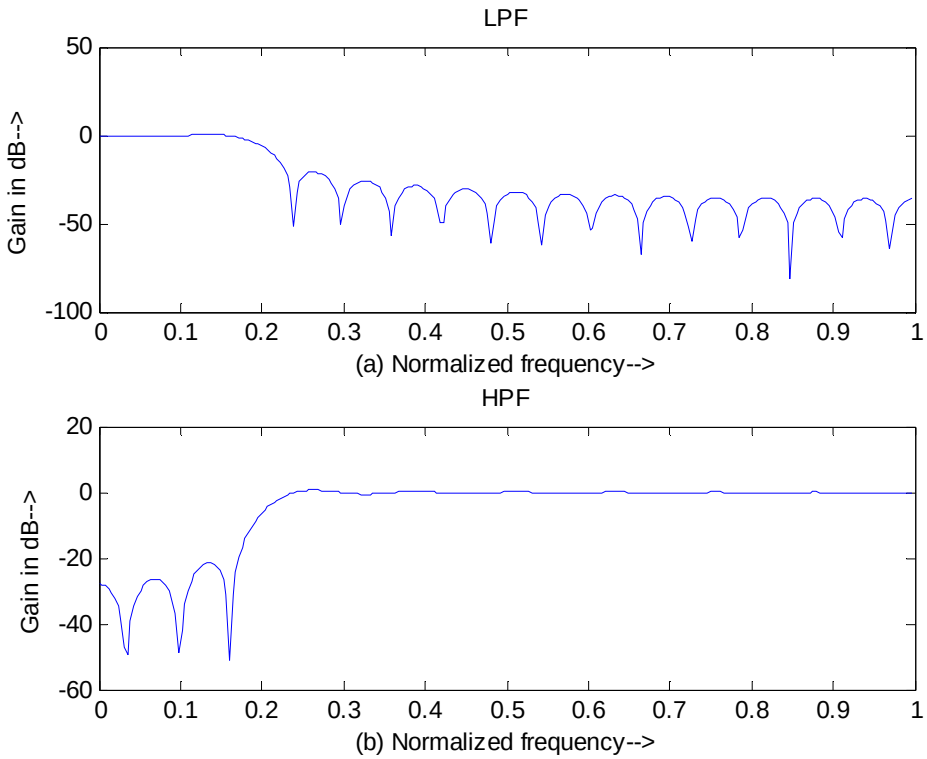
```
%HPF
b=fir1(n,wp,'high',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,1,2);plot(o/pi,m);
title('HPF');
ylabel('Gain in dB-->');
xlabel('(b) Normalized frequency-->');
```

USING RECTANGULAR WINDOW

OUTPUT:

```
enter passband ripple0.02
enter the stopband ripple0.01
enter passband freq1000
enter stopband freq1500
enter sampling freq 10000
enter your choice of window function 1. rectangular 2. triangular 3.kaiser:
 1
Rectangular window filter response
```

GRAPH:

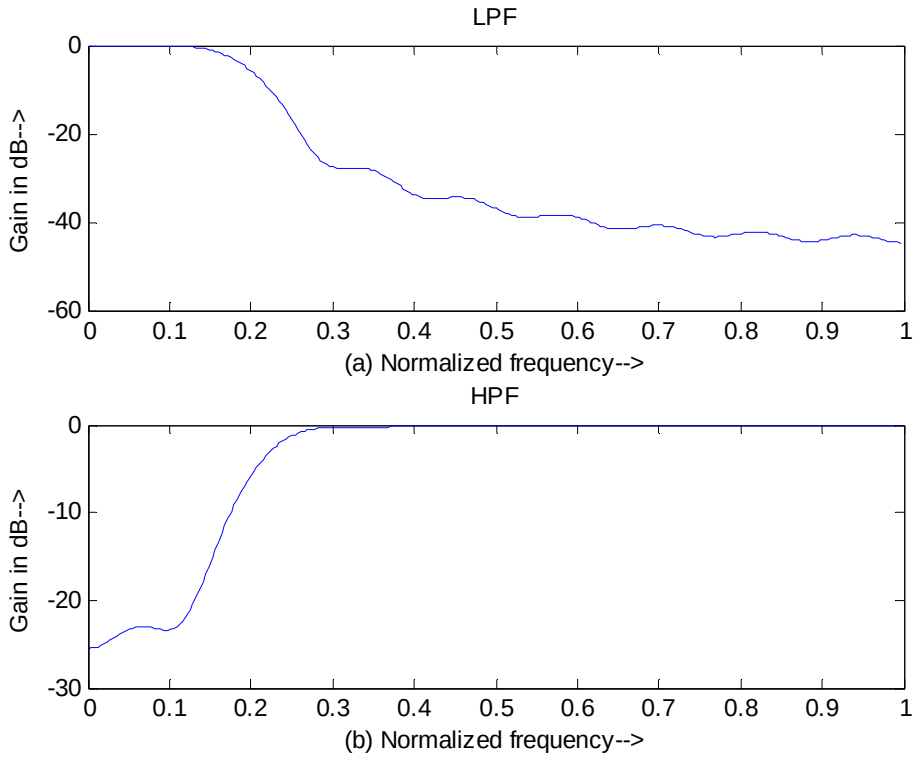


USING TRIANGULAR WINDOW

OUTPUT:

enter passband ripple0.02
enter the stopband ripple0.01
enter passband freq1000
enter stopband freq1500
enter sampling freq 10000
enter your choice of window function 1. rectangular 2. triangular 3.kaiser:
2
Triangular window filter response

GRAPH:

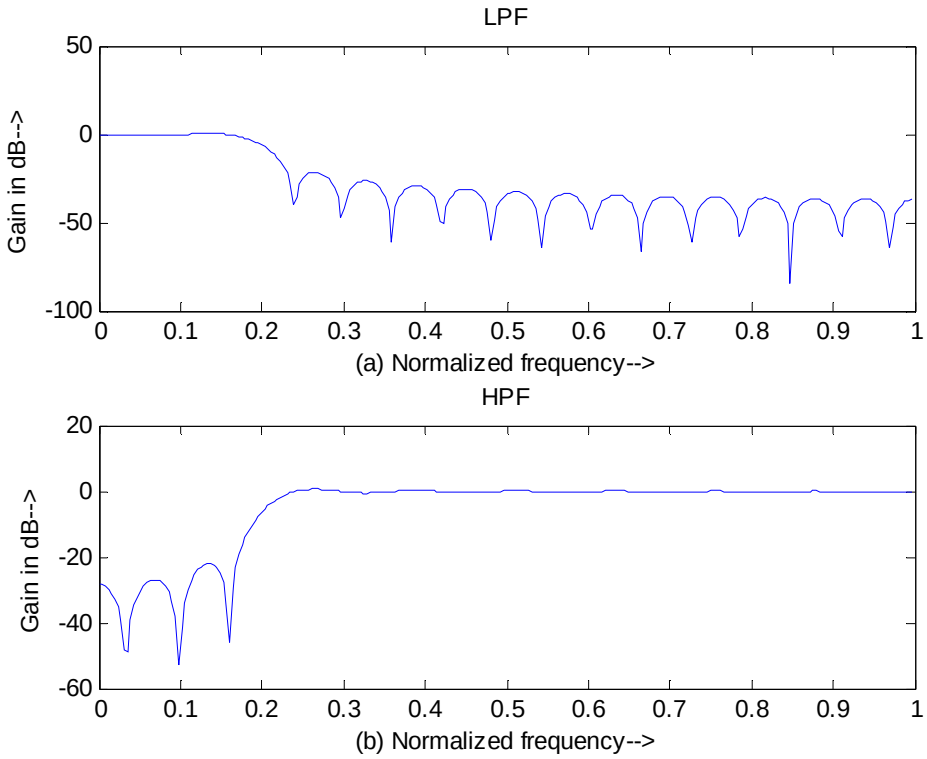


USING KAISER WINDOW

OUTPUT:

enter passband ripple0.02
enter the stopband ripple0.01
enter passband freq1000
enter stopband freq1500
enter sampling freq 10000
enter your choice of window function 1. rectangular 2. triangular 3.kaiser:
3
kaiser window filter response

GRAPH:



INFERENCE: This program requires certain specifications about the pass band and stop band ripple & frequencies and the sampling frequency. Here we have used a function called *'fir1'* in order to design a Nth order Low pass and a High pass filter. This function returns the filter coefficients in length N+1 vector b. The cutoff frequency '*wp*' must be between 0 < wp < 1.0 , with 1.0 corresponding to half the sampling rate.

Based up on the choice of available windowing functions, the filter response is generated.

RESULT: The LP and HP FIR Filter response for the given sequence is generated based upon the choice of the windowing function and the filter characteristics are plotted.

Ex No:7(A)

Date:

IMPLEMENTATION OF IIR LP FILTER FOR A GIVEN SEQUENCE

AIM: To design and implement IIR (LPF) filters for a given sequence.

SOFTWARE REQUIRED: MATLAB 7.0

PROGRAM DESCRIPTION: The IIR filter can realize both the poles and zeroes of a system because it has a rational transfer function, described by polynomials in z in both the numerator and the denominator:

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{M} a_k z^{-k}}$$

The difference equation for such a system is described by the following:

$$Y(n) = \sum_{k=0}^{M} b_k x(n-k) + \sum_{k=1}^{N} a_k y(n-k)$$

M and N are order of the two polynomials b_k and a_k are the filter coefficients. These filter coefficients are generated using FDS (Filter Design software or Digital Filter design package).

IIR filters can be expanded as infinite impulse response filters. In designing IIR filters, cutoff frequencies of the filters should be mentioned. The order of the filter can be estimated using butter worth polynomial. That's why the filters are named as butter worth filters. Filter coefficients can be found and the response can be plotted.

MATLAB CODE:

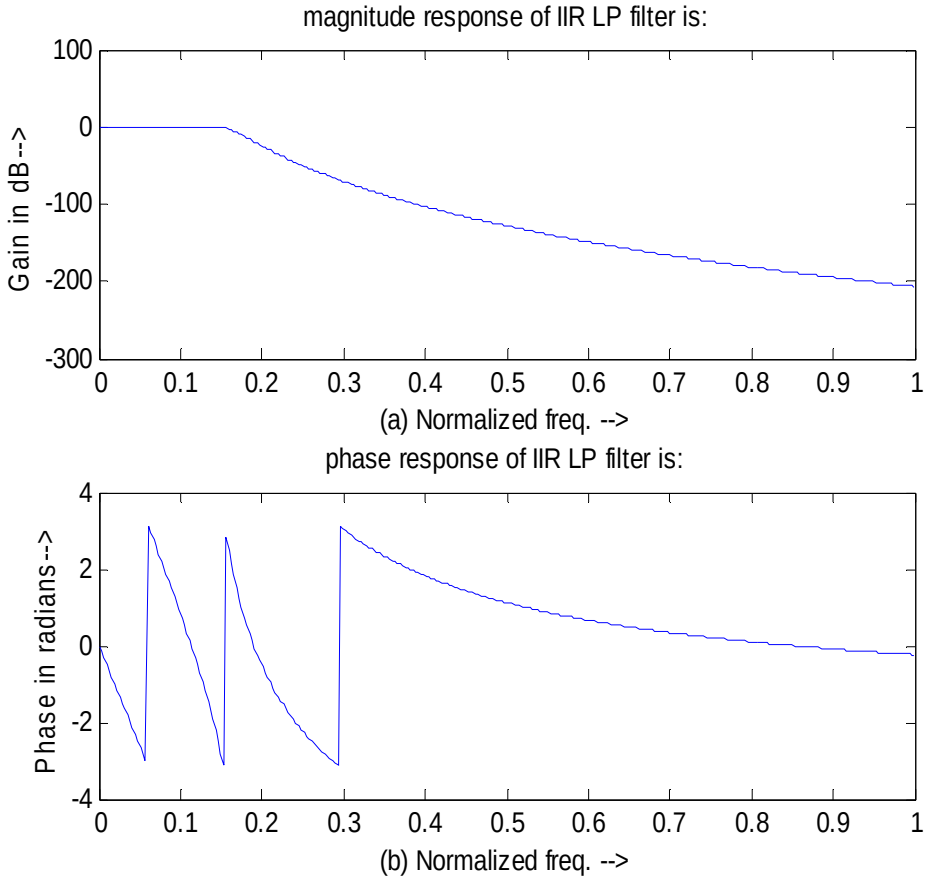
```
clc;
clear all;
close all;
disp('enter the IIR filter design specifications');
rp=input('enter the passband ripple');
rs=input('enter the stopband ripple');
wp=input('enter the passband freq');
ws=input('enter the stopband freq');
fs=input('enter the sampling freq');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs,'s');
disp('Frequency response of IIR LPF is:');
[b,a]=butter(n,wn,'low','s');
w=0:.01:pi;
```

```
[h,om]=freqs(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure,subplot(2,1,1);
plot(om/pi,m);
title('magnitude response of IIR LP filter is:');
xlabel('(a) Normalized freq. -->');
ylabel('Gain in dB-->');
subplot(2,1,2);
plot(om/pi,an);
title('phase response of IIR LP filter is:');
xlabel('(b) Normalized freq. -->');
ylabel('Phase in radians-->');
```

OUTPUT:

```
enter the IIR filter design specifications
enter the passband ripple0.15
enter the stopband ripple60
enter the passband freq1500
enter the stopband freq3000
enter the sampling freq7000
Frequency response of IIR LPF is:
```

GRAPHS:



INFERENCE: This program requires certain specifications about the pass band and stop band ripple & frequencies and the sampling frequency. Here we have used a function called *'[n,wn]=buttord(w1,w2,rp,rs,'s')'* in order to find the lowest order N of digital butterworth filter that loses no more than '*rp*' db in the pass band and has at least '*rs*' db of attenuation in the stop band . w1 and w2 are the pass band and stop band edge frequencies, normalized from 0 to 1.

The function '*butter*' designs a Nth order low pass digital butterworth filter and returns the filter coefficients in length N+1 vectors B(numerator) and A (denominator).

RESULT: The LP IIR Filter response for the given sequence is generated and the filter characteristics are plotted.

Ex No:7(B)

Date:

IMPLEMENTATION OF IIR HP FILTER FOR A GIVEN SEQUENCE

AIM: To design and implement IIR (HPF) filters for a given sequence.

SOFTWARE REQUIRED: MATLAB 7.0

PROGRAM DESCRIPTION: The IIR filter can realize both the poles and zeroes of a system because it has a rational transfer function, described by polynomials in z in both the numerator and the denominator:

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{M} a_k z^{-k}}$$

The difference equation for such a system is described by the following:

$$Y(n) = \sum_{k=0}^{M} b_k x(n-k) + \sum_{k=1}^{N} a_k y(n-k)$$

M and N are order of the two polynomials b_k and a_k are the filter coefficients. These filter coefficients are generated using FDS (Filter Design software or Digital Filter design package).

IIR filters can be expanded as infinite impulse response filters. In designing IIR filters, cutoff frequencies of the filters should be mentioned. The order of the filter can be estimated using butter worth polynomial. That's why the filters are named as butter worth filters. Filter coefficients can be found and the response can be plotted.

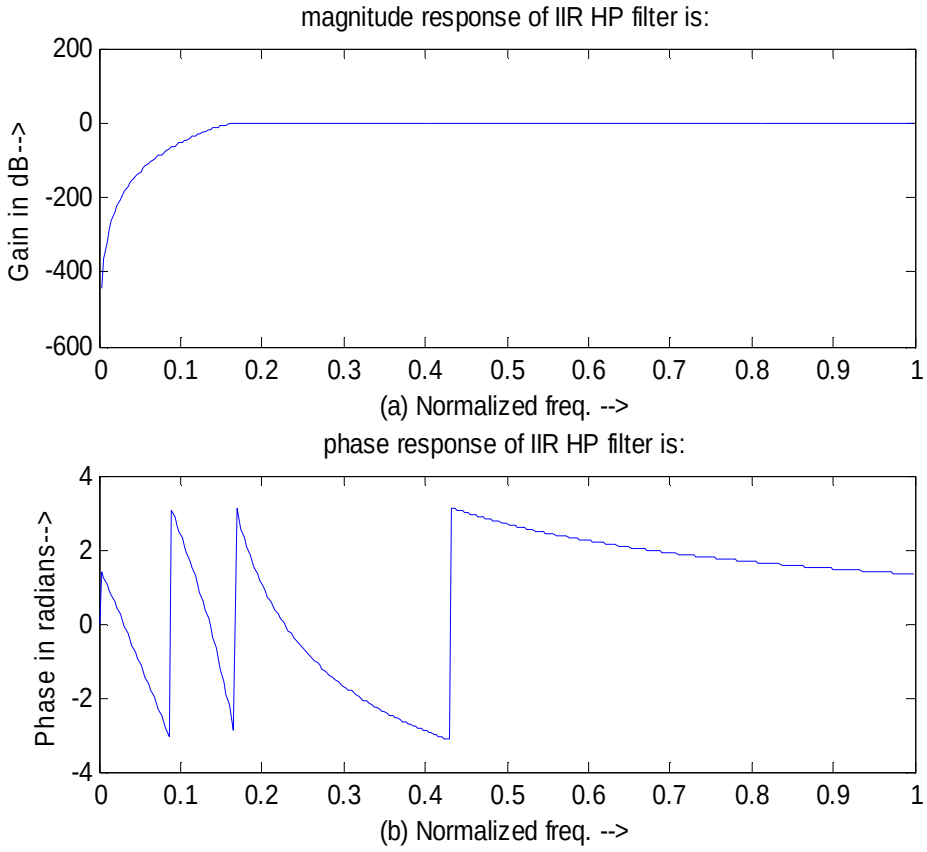
MATLAB CODE:

```
clc;
clear all;
close all;
disp('enter the IIR filter design specifications');
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,'s');
disp('Frequency response of IIR HPF is:');
[b,a]=butter(n,wn,'high','s');
w=0:.01:pi;
[h,om]=freqs(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure,subplot(2,1,1);
plot(om/pi,m);
title('magnitude response of IIR HP filter is:');
xlabel('(a) Normalized freq. -->');
ylabel('Gain in dB-->');
subplot(2,1,2);
plot(om/pi,an);
title('phase response of IIR HP filter is:');
xlabel('(b) Normalized freq. -->');
ylabel('Phase in radians-->');
```

OUTPUT:

```
enter the IIR filter design specifications
enter the passband ripple0.15
enter the stopband ripple60
enter the passband freq1500
enter the stopband freq3000
enter the sampling freq7000
Frequency response of IIR HPF is:
```

GRAPHS:



INFERENCE: This program requires certain specifications about the pass band and stop band ripple & frequencies and the sampling frequency. Here we have used a function called *'[n,wn]=buttord(w1,w2,rp,rs,'s')'* in order to find the lowest order N of digital butterworth filter that loses no more than '*rp*' db in the pass band and has at least '*rs*' db of attenuation in the stop band . w1 and w2 are the pass band and stop band edge frequencies, normalized from 0 to 1.

The function *'butter'* designs a Nth order high pass digital butterworth filter and returns the filter coefficients in length N+1 vectors B (numerator) and A (denominator).

RESULT: The HP IIR Filter response for the given sequence is generated and the filter characteristics are plotted.

Ex No:8(a)

Date:

QUANTIZATION OF NOISE

(Quantization of Gaussian Noise Signal)

AIM: To compute Quantization of Gaussian Noise Signal for a given signal.

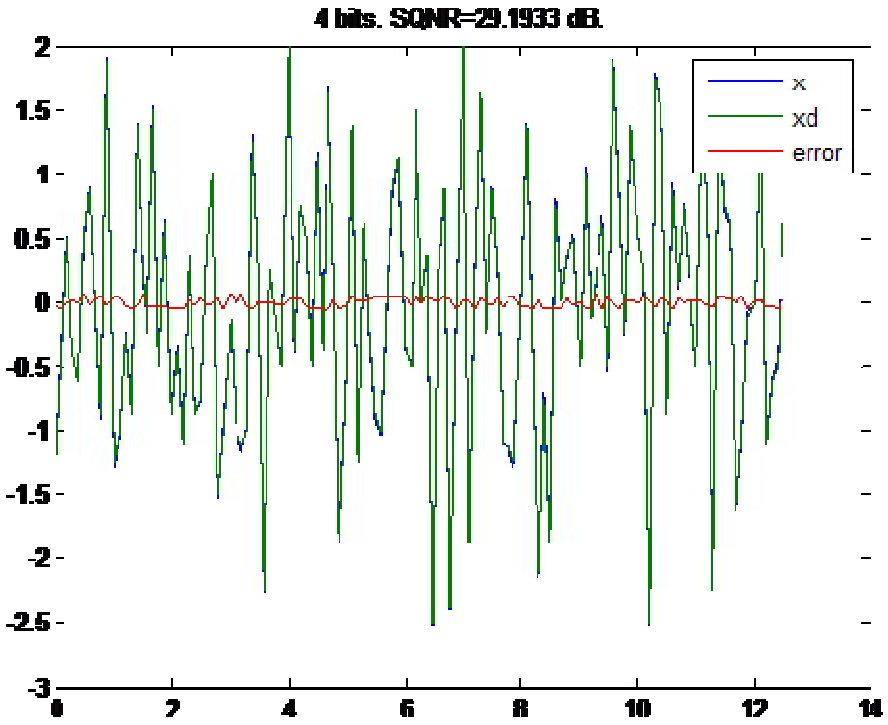
SOFTWARE REQUIRED: MATLAB 7.0

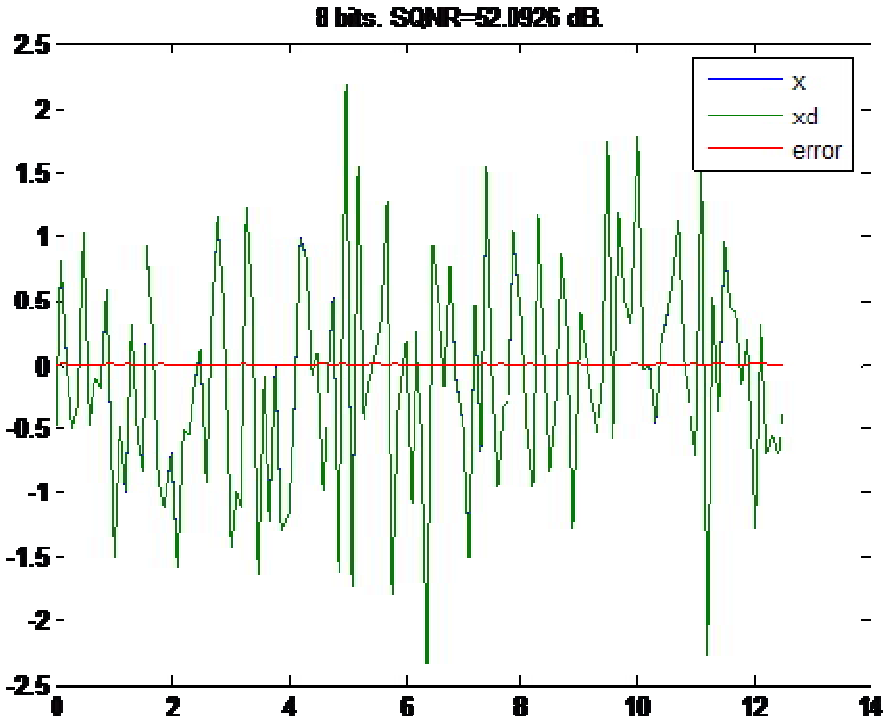
PROGRAM DESCRIPTION:

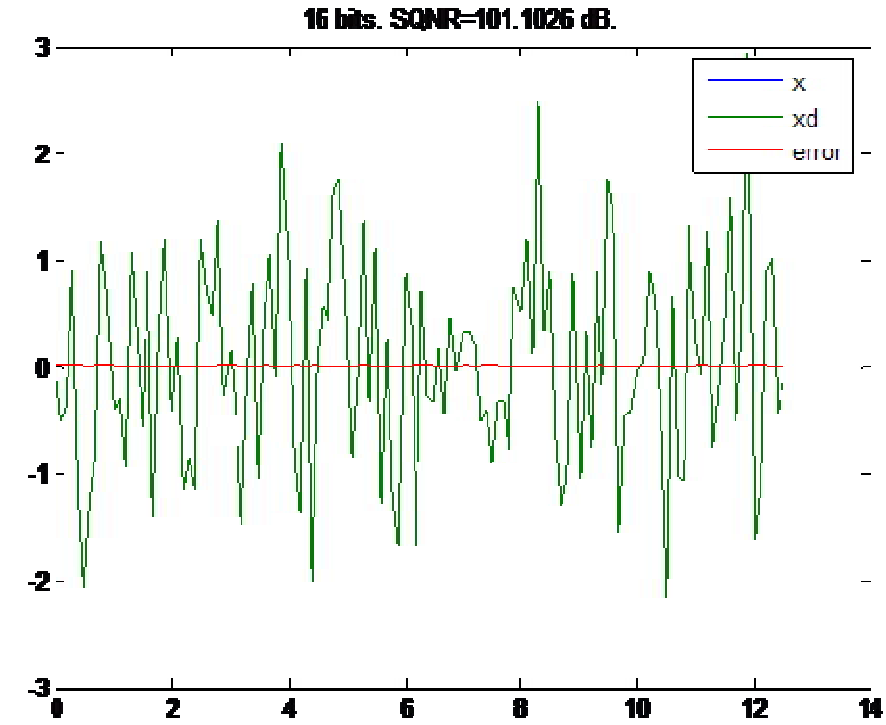
```
t=0:0.1:4*pi;
fprintf('Exp\t\tTh\t\tMean\t\tVar\n');
for n=4:4:16,
```

```
    x=randn(1,length(t));
    xd=round(x*2^(n-1))/2^(n-1);
    error=xd-x;
    error_mean=mean(abs(error));
    error_variance=var(error);
    figure; plot(t,x,t,xd,t,error)
    legend('x','xd','error')
    SQNR=10*log10(sum(x.^2)/sum(error.^2));
    SQNR_eqn=1.76+6.02*n;
    title(strcat(num2str(n),' bits. SQNR=',num2str(SQNR),'
dB.')) fprintf('%0.4f\t%0.4f\t%0.3e\t%0.3e\n',SQNR,SQNR_eqn,error_mean,erro
r_variance);
end
```

Bits	Experimental SQNR (dB)	Mean Quantization Noise	Variance in Quantization Noise
4	29.1933	3.22E-02	1.34E-03
8	52.0926	1.86E-03	4.82E-06
12	77.1105	1.23E-04	1.99E-08
16	101.1026	7.33E-06	7.52E-11




8 bits. SQNR=52.0926 dB.
x
xd
error
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
2
4
6
8
10
12
14


16 bits. SQNR=101.1026 dB.
x
xd
error
3
2
1
0
-1
-2
-3
0
2
4
6
8
10
12
14

RESULT:

Thus the Quantization of Gaussian Noise Signal was computed for a given signal.

Ex No:8(b)

Date:

Quantization of Sinusoidal Signal with Gaussian Random Noise

AIM: To compute Quantization of Sinusoidal Signal with Gaussian Random Noise for a given signal.

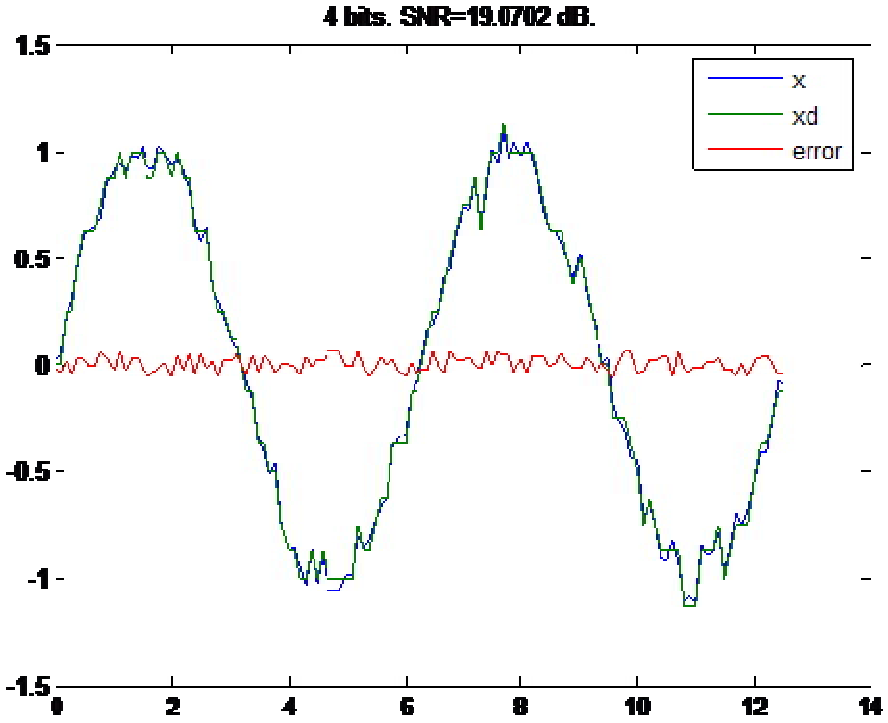
SOFTWARE REQUIRED: MATLAB 7.0

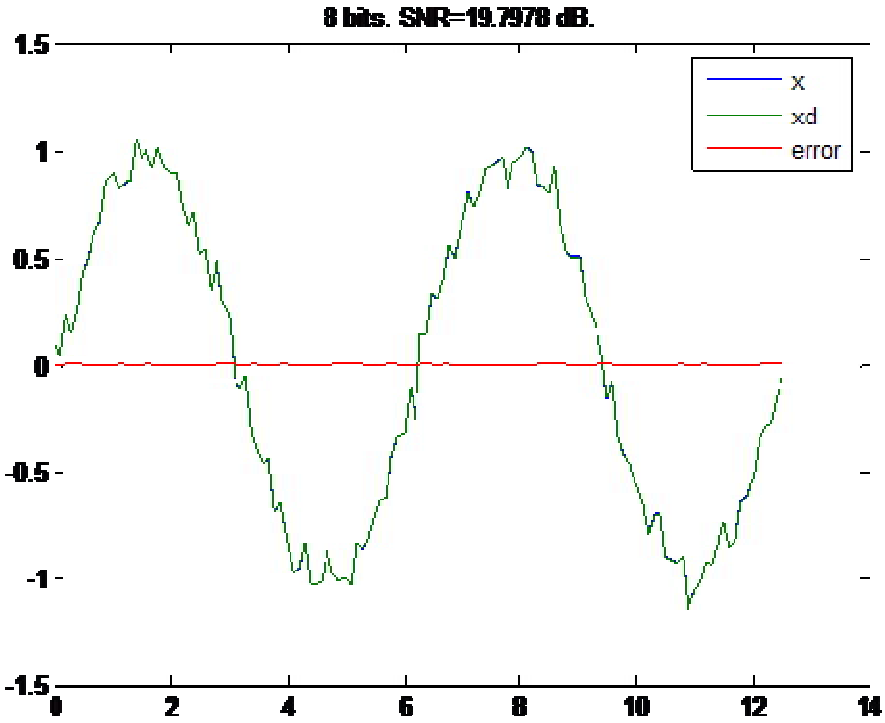
PROGRAM DESCRIPTION:

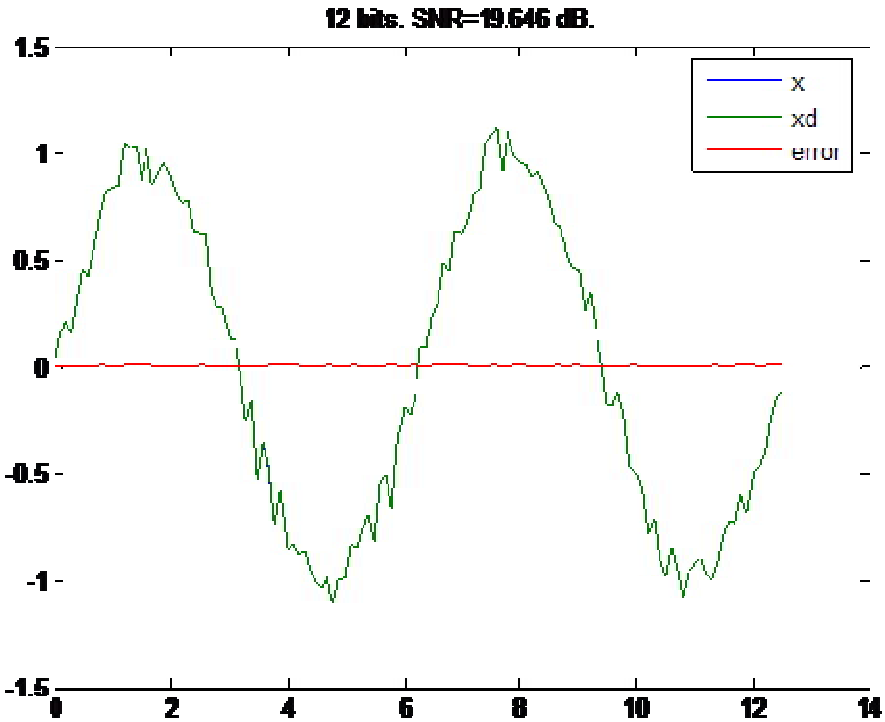
```
t=0:0.1:4*pi;
fprintf('SNR\t\tExp\t\tTh\t\tMean\n');
for n=4:4:16,
    x_original=sin(t);
    noise=randn(1,length(t))./sqrt(200);
    x=x_original+noise;
    xd_original=round(x_original*2^(n-1))/2^(n-1);
    xd=round(x*2^(n-1))/2^(n-1);
    error_original=xd_original-x_original;
    error=xd-x;
    error_mean=mean(abs(error));
    figure; plot(t,x,t,xd,t,error)
    legend('x','xd','error')
    SNR=10*log10(sum(x.^2)/(sum(error.^2)+sum(noise.^2)));
    SQNR=10*log10(sum(x.^2)/(sum(error.^2)));
```

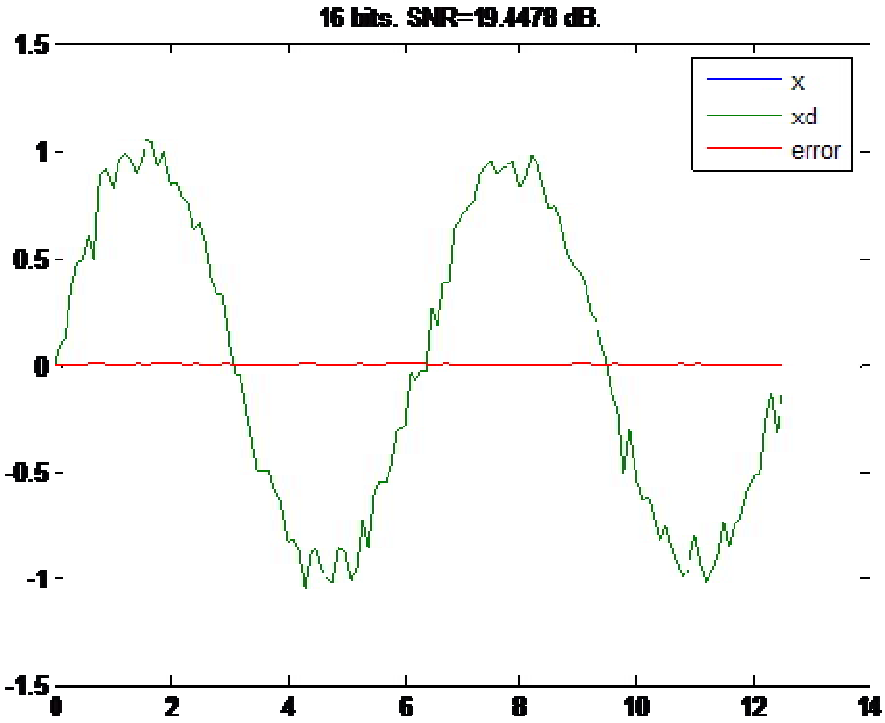
```
    SQNR_eqn=1.76+6.02*n;
    title(strcat(num2str(n),' bits. SNR=',num2str(SNR),'
dB.'))    fprintf('%0.4f\t%0.4f\t%0.4f\t%0.3e\n',SNR,SQNR,SQNR_eqn,error_me
an);
end
```

Bits	Experimental SNR (dB)	Experimental SQNR (dB)	Theoretical SQNR (dB)	Mean Quantization Noise
4	19.0702	29.1933	25.84	3.22E-02
8	19.7978	52.0926	49.92	1.86E-03
12	19.646	77.1105	74	1.23E-04
16	19.4478	101.1026	98.08	7.33E-06




8 bits. SNR=19.7978 dB.
x
xd
error
1.5
1
0.5
0
-0.5
-1
-1.5
0
2
4
6
8
10
12
14


12 bits. SNR=19.646 dB.
x
xd
error
1.5
1
0.5
0
-0.5
-1
-1.5
0
2
4
6
8
10
12
14



RESULT:

Thus Quantization of Sinusoidal Signal with Gaussian Random Noise was computed for a given signal.

ARCHITECTURE OF DSP CHIPS – TMS 320 C5X16X INSTRUCTION

EQUIPMENT REQUIRED: DSP Hardware Kit- DSK C6713

THEORY: The C6713 DSK pin has the following features

1. Code Composer communicates with the DSK through an embedded JTAG emulator with a USB host interface.

2. The DSK can also be used with an external emulator through the external JTAG connector.

TMS320C6713 DSP Features

- Highest-Performance Floating-Point Digital Signal Processor (DSP):
  - Eight 32-Bit Instructions/Cycle
  - 32/64-Bit Data Word
  - 300-, **225**-, 200-MHz (GDP), and **225**-, 200-, 167-MHz (PYP) Clock Rates

- 3.3-, 4.4-, 5-, 6-Instruction Cycle Times
- 2400/1800, 1800/1350, 1600/1200, and 1336/1000 MIPS /MFLOPS
- Rich Peripheral Set, Optimized for Audio
- Highly Optimized C/C++ Compiler
- Extended Temperature Devices Available

❖ Advanced Very Long Instruction Word (VLIW) TMS320C67x™ DSP Core

- Eight Independent Functional Units:
  - Two ALUs (Fixed-Point)
  - Four ALUs (Floating- and Fixed-Point)
  - Two Multipliers (Floating- and Fixed-Point)
- Load-Store Architecture With 32 32-Bit General-Purpose Registers
- Instruction Packing Reduces Code Size
- All Instructions Conditional

❖ Instruction Set Features

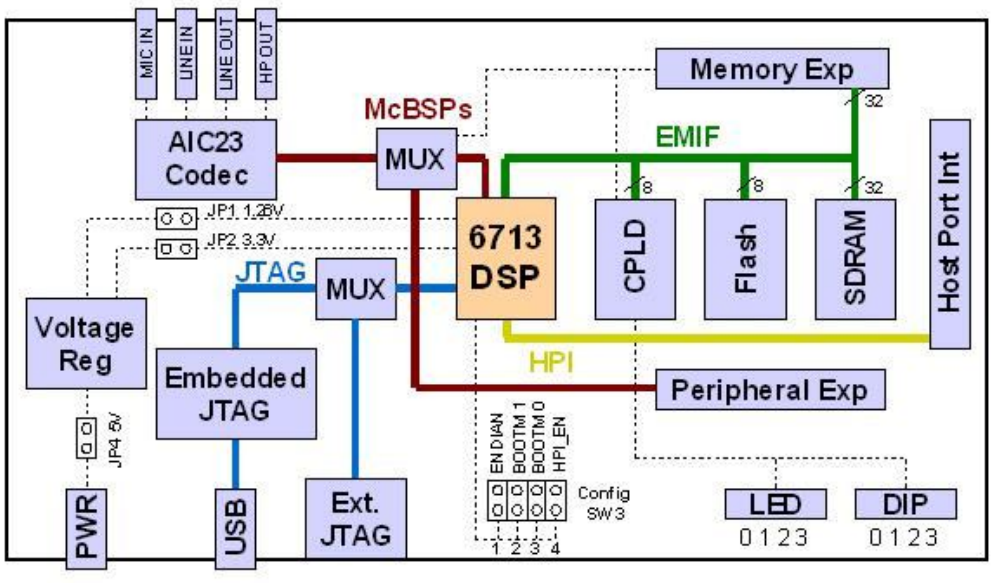
- Native Instructions for IEEE 754
  - Single- and Double-Precision

Byte-Addressable (8-, 16-, 32-Bit Data)

- 8-Bit Overflow Protection
- Saturation; Bit-Field Extract, Set, Clear; Bit-Counting; Normalization

❖ L1/L2 Memory Architecture

- 4K-Byte L1P Program Cache (Direct-Mapped)
- 4K-Byte L1D Data Cache (2-Way)
- 256K-Byte L2 Memory Total: 64K-Byte L2 Unified Cache/Mapped RAM, and 192K-Byte Additional L2 Mapped RAM



TMS320C6713 DSK Overview Block Diagram

Device Configuration

- Boot Mode: HPI, 8-, 16-, 32-Bit ROM Boot
- Endianness: Little Endian, Big Endian

- 32-Bit External Memory Interface (EMIF)
  - Glueless Interface to SRAM, EPROM, Flash, SBSRAM, and SDRAM
  - 512M-Byte Total Addressable External Memory Space
- Enhanced Direct-Memory-Access (EDMA) Controller (16 Independent Channels)
- 16-Bit Host-Port Interface (HPI)
- Two Multichannel Audio Serial Ports (McASPs)
  - Two Independent Clock Zones Each (1 TX and 1 RX)
  - Eight Serial Data Pins Per Port:
    Individually Assignable to any of the Clock Zones
  - Each Clock Zone Includes:
    - Programmable Clock Generator
    - Programmable Frame Sync Generator
    - TDM Streams From 2-32 Time Slots
    - Support for Slot Size:
      8, 12, 16, 20, 24, 28, 32 Bits
    - Data Formatter for Bit Manipulation
  - Wide Variety of I2S and Similar Bit Stream Formats
  - Integrated Digital Audio Interface Transmitter (DIT) Supports:
    - S/PDIF, IEC60958-1, AES-3, CP-430 Formats
    - Up to 16 transmit pins
    - Enhanced Channel Status/User Data
  - Extensive Error Checking and Recovery
- Two Inter-Integrated Circuit Bus (I^2C Bus™) Multi-Master and Slave Interfaces
- Two Multichannel Buffered Serial Ports:
  - Serial-Peripheral-Interface (SPI)
  - High-Speed TDM Interface

- AC97

Two 32-Bit General-Purpose Timers

- Dedicated GPIO Module With 16 pins (External Interrupt Capable)
- Flexible Phase-Locked-Loop (PLL) Based Clock Generator Module
- IEEE-1149.1 (JTAG †) Boundary-Scan-Compatible
- Package Options:
  - 208-Pin Power PAD™ Plastic (Low-Profile) Quad Flat pack (PYP)
  - 272-BGA Packages (GDP and ZDP)
- 0.13-µm/6-Level Copper Metal Process
  - CMOS Technology
- 3.3-V I/Os, 1.2 ‡ -V Internal (GDP & PYP)
- 3.3-V I/Os, 1.4-V Internal (GDP)(300 MHz only)

Ex No:9(a)

Date:

MULTIRATE SIGNAL PROCESSING
(PROGRAM TO VERIFY DECIMATION)

AIM: To verify Decimation and Interpolation of a given Sequences

SOFTWARE REQUIRED: MAT LAB 7.0

PROGRAM DESCRIPTION: The sampling rate alteration that is employed to generate a new sequence with a sampling rate lower than that of a given sequence. Thus, if x[n] is a sequence with a sampling rate of F_T Hz and it is used to generate another sequence y[n] with a desired sampling rate of F_T' Hz, then the sampling rate alteration ratio is given by $F_T' / F_T = R$.

If R < 1, the sampling rate is decreased by a process called *decimation* and it results in a sequence with a lower sampling rate.

The decimation can be carried out by using the pre-defined commands decimate.

MATLAB CODE:

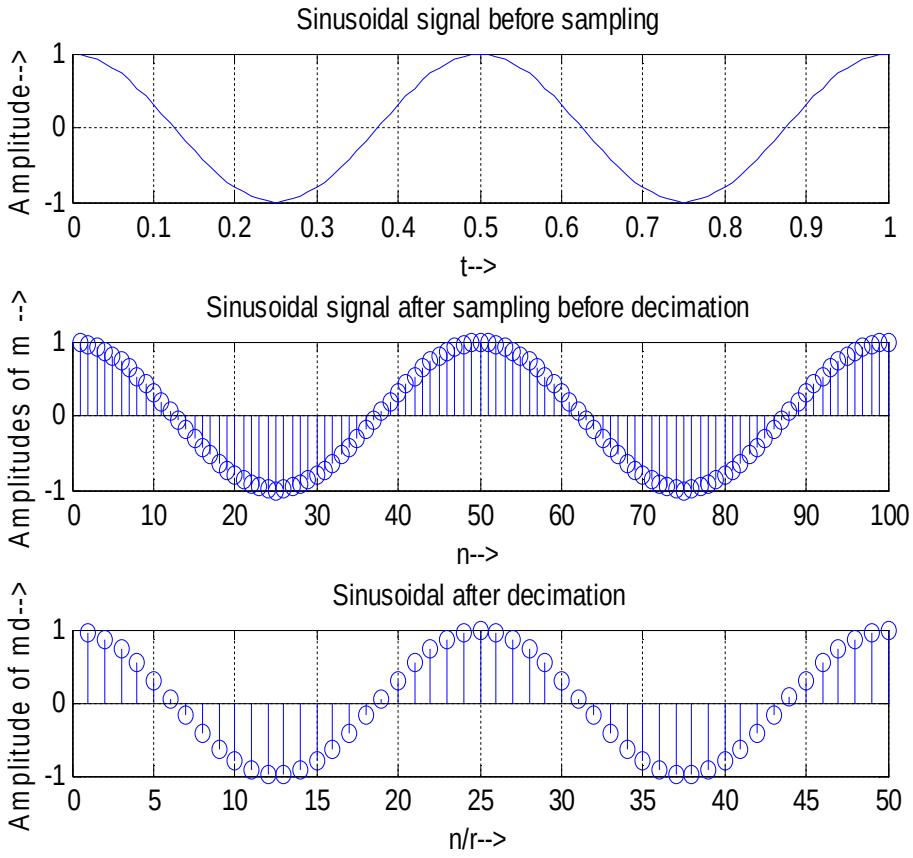
```
% DECIMATION
clc;
clear all;
close all;
disp('Let us take a sinusoidal sequence which has to be decimated: ');
fm=input('Enter the signal frequency fm: ');
fs=input('Enetr the sampling frequnecy fs: ');
T=input('Enter the duration of the signal in seconds T: ');
dt=1/fs;
t=dt:dt:T
M=length(t);
m=cos(2*pi*fm*t);
r=input('Enter the factor by which the sampling frequency has to be reduced r: ');
md=decimate(m,r);
figure(1);
subplot(3,1,1);
plot(t,m);
grid;
xlabel('t-->');
ylabel('Amplitude-->');
title('Sinusoidal signal before sampling');
subplot(3,1,2);
stem(m);
grid;
xlabel('n-->');
ylabel('Amplitudes of m -->');
title('Sinusoidal signal after sampling before decimation');
```

```
subplot(3,1,3);
stem(md);
grid;
title('Sinusoidal after decimation');
xlabel('n/r-->');
ylabel('Amplitude of md-->');
```

OUTPUT:

Let us take a sinusoidal sequence which has to be decimated:
Enter the signal frequency fm: 2
Enetr the sampling frequnecy fs: 100
Enter the duration of the signal in seconds T: 1
Enter the factor by which the sampling frequency has to be reduced r: 2

GRAPHS:



INFERENCE : The only constraint about the program is that the factors of decimation or interpolation should be an integers. If we want to change the sampling frequency by a factor which is not an integer it can be done by using the command resample by which we can change the sampling rate by a factor I / D. For this we have to interpolate by an integer factor I and then decimate by an integer factor D

RESULT : The Decimation of given sequences is verified and graphs are plotted.

Ex No:9(b)

Date:

PROGRAM TO VERIFY INTERPOLATION

AIM: To verify Interpolation of a given Sequence.

SOFTWARE REQUIRED: MAT LAB 7.0

PROGRAM DESCRIPTION: The sampling rate alteration that is employed to generate a new sequence with a sampling rate higher than that of a given sequence. Thus, if x[n] is a sequence with a sampling rate of F_T Hz and it is used to generate another sequence y[n] with a desired sampling rate of F_T' Hz, then the sampling rate alteration ratio is given by $F_T' / F_T = R$.

If R > 1, the process is called *interpolation* and results in a sequence with a higher sampling rate.

The interpolation can be carried out by using the pre-defined command *interp* respectively.

MATLAB CODE:

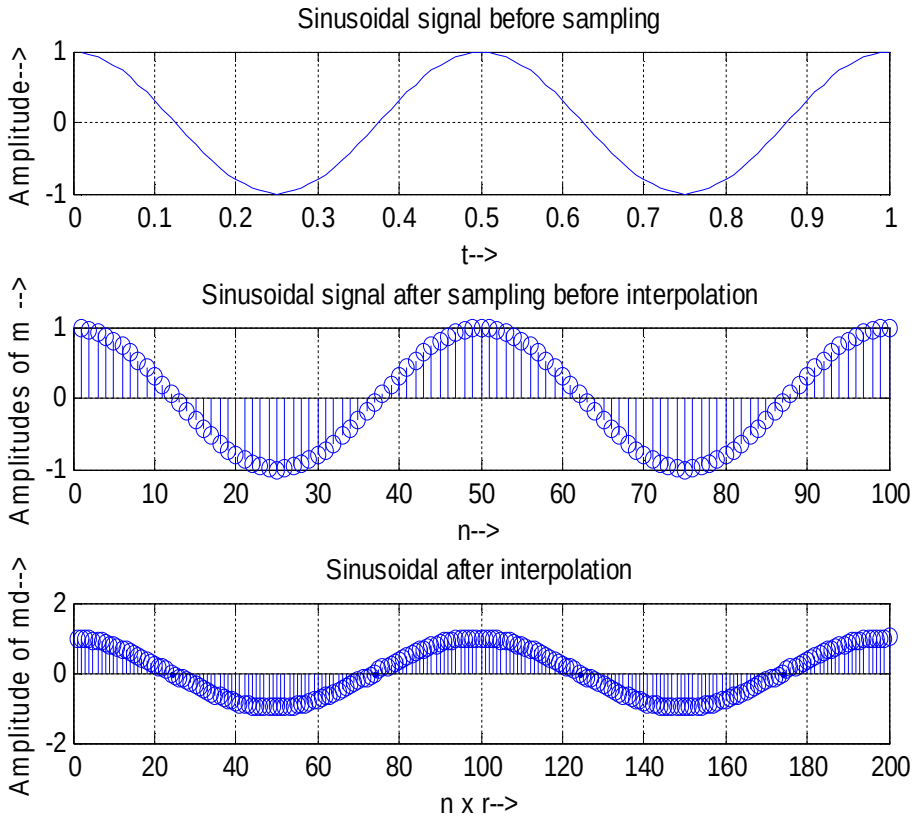
```
 %INTERPOLATION
clc;
clear all;
close all;
disp('Let us take a sinusoidal sequence which has to be interpolated: ');
fm=input('Enter the signal frequency fm: ');
fs=input('Enetr the sampling frequnecy fs: ');
T=input('Enter the duration of the signal in seconds T: ');
dt=1/fs;
t=dt:dt:T
M=length(t);
m=cos(2*pi*fm*t);
r=input('Enter the factor by which the sampling frequency has to be increased r: ');
md=interp(m,r);
figure(1);
subplot(3,1,1);
plot(t,m);
grid;
xlabel('t-->');
ylabel('Amplitude-->');
title('Sinusoidal signal before sampling');
subplot(3,1,2);
stem(m);
grid;
xlabel('n-->');
ylabel('Amplitudes of m -->');
title('Sinusoidal signal after sampling before interpolation');
subplot(3,1,3);
```

```
stem(md);
grid;
title('Sinusoidal after interpolation');
xlabel('n x r-->');
ylabel('Amplitude of md-->');
```

OUTPUT:

```
Let us take a sinusoidal sequence which has to be interpolated:
Enter the signal frequency fm: 2
Enetr the sampling frequnecy fs: 100
Enter the duration of the signal in seconds T: 1
Enter the factor by which the sampling frequency has to be increased r: 2
```

GRAPHS:



INFERENCE : The only constraint about the program is that the factors of decimation or interpolation should be an integers. If we want to change the sampling frequency by a factor which is not an integer it can be done by using the command resample by which we can change the sampling rate by a factor I / D. For this we have to interpolate by an integer factor I and then decimate by an integer factor D

RESULT : The Interpolation of given sequences is verified and graphs are plotted.

Ex No:10(a)

Date:

DSP PROCESSOR IMPLEMENTATION

(Signal conversion)

AIM:

To write a program to convert analog signals into digital signals using TMS320C50 debugger.

APPARATUS REQUIRED:

1.System with TMS 320C50 debugger software

2.TMS 320C50 Kit.

3.CR0

4.Function Generator.

ALGORITHM:

1. Initialize data pointer with 100H data
2. give the analog signal as input
3. Introduce the time delay as per required.
4. Observe the discrete signal as output
5. Plot the graph.

PROGRAM:

```
      .mmregs
      .text
         LDP    #100H
START:
         IN  0,06H
         NOP
         IN 0,04H RPT
         #4FFH
         NOP
         OUT 0,04H
         RPT #4FFH
         NOP
         B START
```

MODEL GRAPH:

OUTPUT:

WAVEFORM	AMPLITUDE (V)	TIME PERIOD (ms)
INPUT OUTPUT		

RESULT:
Thus the given signal was converted using TMS320C50 DSP Processor.

Ex No:10(b)

Date:

FIR BAND PASS FILTER

AIM:

To design a FIR band pass filter in serial mode.

APPARATUS REQUIRED:

1.System with VI debugger software

2.TMS 320C50 Kit.

3.CR0

ALGORITHM:

1. Start the program.
2. Initialize the C table value.
3. Load the auxillary register with 0200H.
4. Modify the auxillary register zero.
5. Block the C table to the program.
6. Set configuration control bit.
7. Load the data pointer with 0AH.
8. Initialize the analog to digital conversion.
9. Load the auxillary register 1 with 0300 content.
10. Load the accumulator in 8000H.
11. AND the accumulator with 0FFFH.
12. Subtract the accumulator content with data 800H.
13. Modify the auxillary register 1.
14. Store the accumulator data in 8000H.
15. Load the auxillary register 1 with content 0333H.
16. Zero the accumulator register.
17. Multiply the accumulator with data.
18. Load the program register, with PM bits to accumulator.
19. Load the auxillary register 1 with content 0300H.
20. Add accumulator content with 800H.
21. Shift the accumulator right 1 bit.
22. Store the accumulator content in 8200 location.
23. Branch the program to step

PROGRAM:

*Approximation type:Window design- Blackmann Window *Filter type:band Pass Filter
*Filter Order:52
*lower Cutoff frequency in KHZ=3.000000 *upper Cutoff frequency in KHZ=3.000000

```
.mmregs
```

```
        .text
        B START
CTABLE:
        .word 024AH
        .word 010FH
        .word 0FH
        .word 0FFECH
        .word 0C6H
        .word 0220H
        .word 0312H
        .word 02D3H
        .word 012FH
        .word 0FEBDH
        .word 0FC97H
        .word 0FBCBH
        .word 0FCB0H
        .word 0FE9EH
        .word 029H
        .word 0FFDCH
        .word 0FD11H
        .word 0F884H
        .word 0F436H
        .word 0F2A0H
        .word 0F58AH
        .word 0FD12H
        .word 075FH
        .word 01135H
        .word 01732H
        .word 01732H
        .word 01135H
        .word 075FH
        .word 0FD12H
        .word 0F58AH
        .word 0F2A0H
        .word 0F436H
        .word 0F884H
        .word 0FD11H
        .word 0FFDCH
        .word 029H
        .word 0FE9EH
        .word 0FCB0H
        .word 0FBCBH
        .word 0FC97H
        .word 0FEBDH
        .word 012FH
        .word 02D3H
        .word 0312H
         .word 0220H
        .word 0C6H
        .word 0FFECH
```

```
        .word 0FH
        .word 010FH
        .word 024AH

*               Move the Filter coefficients
*       from program memory to data memory

START:
        LAR   AR0,#0200H
        MAR  *,AR0
        RPT   #33H
        BLKP CTABLE,*+
        SETC CNF

* Input data and perform convolution
ISR:    LDP   #0AH
        IN      0,06H
        IN      0,04H
        NOP
        NOP
        NOP
        NOP
        LAR   AR1,#0300H
        LACC 0
        AND #0FFFH
        SUB #800H
        MAR  *,AR1
        SACL *
        LAR   AR1,#0333H
        ZAP
        RPT #33H
        MACD 0FF00H,*-
        APAC
        LAR   AR1,#0300H
        SACH *        ; give as SACH *, 1 in case of overflow
        LACC *
        ADD #800H
        SFR             ; remove if output is less amplitude
        SACL *
        OUT   *,4               ;pulse to find sampling frequency
        NOP
        B ISR
        .end
```

MODEL GRAPH:

TABULATION:

S.No.	Frequency(Hz)	Vout(v)	Vout/Vin	Gain in db= 20log Vout/Vin

RESULT:

Thus the program for designing a FIR band pass filter in serial mode was performed

Ex No:10(c)

Date:

FIR BAND REJECT FILTER

AIM:

To design a FIR band reject filter in serial mode.

APPARATUS REQUIRED:

1.System with VI debugger software

2.TMS 320C50 Kit.

3.CR0

ALGORITHM:

1. Start the program.
2. Initialize the C table value.
3. Load the auxillary register with 0200H.
4. Modify the auxillary register zero.
5. Block the C table to the program.
6. Set configuration control bit.
7. Load the data pointer with 0AH.
8. Initialize the analog to digital conversion.
9. Load the auxillary register 1 with 0300 content.
10. Load the accumulator in 8000H.
11. AND the accumulator with 0FFFH.
12. Subtract the accumulator content with data 800H.
13. Modify the auxillary register 1.
14. Store the accumulator data in 8000H.
15. Load the auxillary register 1 with content 0333H.
16. Zero the accumulator register.
17. Multiply the accumulator with data.
18. Load the program register, with PM bits to accumulator.
19. Load the auxillary register 1 with content 0300H.
20. Add accumulator content with 800H.
21. Shift the accumulator right 1 bit.
22. Store the accumulator content in 8200 location.
23. Branch the program to step 7.

PROGRAM:

*Approximation type:Window design- Blackmann Window *Filter type:band Pass Filter
*Filter Order:52
*lower Cutoff frequency in KHZ=3.000000 *upper Cutoff frequency in KHZ=5.000000

```
.mmregs
.text
```

```
        B START
CTABLE:
        .word 0FEB9H
        .word 014EH
        .word 0FDA1H
        .word 155H
        .word 0FE1BH
        .word 282H
        .word 0FEAFH
        .word 2ACH
        .word 0FD35H
        .word 8DH
        .word 0F9D9H
        .word 0FE07H
        .word 0F7CCH
        .word 0FEE2H
        .word 0FA2FH
        .word 4BAH
        .word 1AH
        .word 25CH
        .word 420H
        .word 1008H
        .word 89H
        .word 0D61H
        .word 0F3F2H
        .word 0AF9H
        .word 0DB7EH
        .word 045DFH
        .word 0DB7EH
        .word 0AF9H
        .word 0F3F2H
        .word 0D61H
        .word 81H
        .word 89H
        .word 1008H
        .word 420H
        .word 25CH
        .word 1AH
        .word 4BAH
        .word 0FA2FH
        .word 0FEE2H
        .word 0F7CCH
        .word 0FE07H
        .word 0F9D9H
        .word 8DH
        .word 0FD35H
        .word 2ACH
        .word 0FEAFH
        .word 282H
        .word 0FE1BH
```

```
        .word 155H
        .word 0FDA1H
        .word 14EH
        .word 0FEB9H

        *       Move the Filter coefficients
*       from program memory to data memory

START:
        LAR   AR0,#0200H
        MAR  *,AR0
        RPT   #33H
        BLKP CTABLE,*+
        SETC CNF

* Input data and perform convolution
ISR:   LDP   #0AH
        IN      0,06H
        IN      0,04H
        NOP
        NOP
        NOP
        NOP
        LAR   AR1,#0300H
        LACC 0
        AND #0FFFH
        SUB #800H
        MAR  *,AR1
        SACL *
        LAR   AR1,#0333H
        ZAP
        RPT #33H
        MACD 0FF00H,*-
        APAC
        LAR   AR1,#0300H
        SACH *        ; give as SACH *, 1 in case of overflow
        LACC *
        ADD #800H
        SFR             ; remove if output is less amplitude
        SACL *
        OUT   *,4             ;pulse to find sampling frequency
        NOP
        B ISR
.end
```

MODEL GRAPH:

TABULATION:

S.No.	Frequency(Hz)	Vout(v)	Vout/Vin	Gain in db= 20log Vout/Vin

RESULT:

Thus the program for designing a FIR band reject filter in serial mode was performed using TMS320C50 debugger