Engineer to Excel

SIMATS ENGINEERING

SIMATS

Saveetha Institute of Medical And Technical Sciences

(Declared as Deemed to be University under Section 3 of UGC Act 1956)

ECA09-DIGITAL SIGNAL PROCESSING LAB MANUAL

Software Experiments Using SCILAB:

1. Write SCILAB program to generate the following signals:
   (a) Unit step signal
   (b) Unit Impulse sugnal
   (c) Unit ramp signal
   (d) Sinusoidal signal
   (e) Exponential signal
2. Digital communication using binary phase shift keying
3. Write a SCILAB program to obtain the following:
   (a) N point DFT sequence
   (b) N point IDFT sequence
   (c) Circular convolution
4. Write a SCILAB program to obtain the following Sectional Convolution:
   (a) Overlap Save Method
   (b) Overlap add Method
5. Write a SCILAB program to obtain the following:
   (a) DIT-FFT Algorithm
   (b) DIF-FFT Algorithm
6. Denoising data using FFT
7. Design a filter using Transformation Method.
   (a) Bilinear Transformation
   (b) Impulse Invariant Transformation
8. Write the SCILAB program to design the following Butterworth filters
   (a) Low pass filter
   (b) High pass filter
   (c) Band pass filter
   (d) Band reject filter
9. Write the SCILAB program to design the following Chebyshev-I filters
   (e) Low pass filter
   (f) High pass filter
   (g) Band pass filter
   (h) Band reject filter
10. Write a SCILAB program to design FIR filter using the following window
    (a) Rectangular window
    (b) Hamming window
    (c) Hanning window
    (d) Blackmann window
11. Write a SCILAB program to design FIR filter using Fourier series Method.
12. Write a SCILAB program to design downsampling and up sampling a sinusoidal signal using Multirate Signal Processing
13. Generation of Waveforms using DSP Processor
14. Design and Implementation of Butterworth IIR Filter using DSP Processor
15. Design and Implementation of FIR filter with Rectangular window using DSP Processor

Aim

EX NO:1	Generation of Common Discrete Time Signals
DATE:	

Generate and plot the Unit Impulse Signal, Unit Step Signal,Unit Ramp Signal, Sinusoidal Signal,Exponential Signal: For each signal Write the SCILAB code to generate the signal.Plot the signal and label the axes appropriately.

using SCILAB:

Unit Impulse Signal, Unit Step Signal,Unit Ramp Signal, Sinusoidal Signal,Exponential Signal: **For each signal** Write the SCILAB code to generate the signal.Plot the signal and label the axes appropriately.

Software Required

1. Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

Discrete-time signals are a fundamental concept in digital signal processing and communication. They represent variations in amplitude over discrete points in time. Here, I'll provide a brief overview of some common discrete-time signals and their characteristics:

Unit Step Signal (u[n]): The unit step signal is a basic discrete-time signal that takes the value 1 for non-negative time indices (n >= 0) and 0 otherwise. It is often used to model the onset of events or changes in a system.

Impulse Signal (δ[n]): The impulse signal is also known as the discrete-time delta function. It has a value of 1 at n = 0 and is zero for all other time indices. It's a fundamental signal in signal processing and is used to represent discrete-time impulses or impulses in discrete-time systems.

Exponential Signal: An exponential discrete-time signal is defined as x[n] = A * α^n, where A is the amplitude and α is the exponential factor. Depending on whether α is greater or less than 1, the signal can grow or decay exponentially with time.

Sinusoidal Signal: A sinusoidal discrete-time signal has the form x[n] = A * cos(ωn + φ), where A is the amplitude, ω is the angular frequency, n is the time index, and φ is the phase shift. Sinusoidal signals exhibit periodic behavior and are a fundamental representation of oscillations.

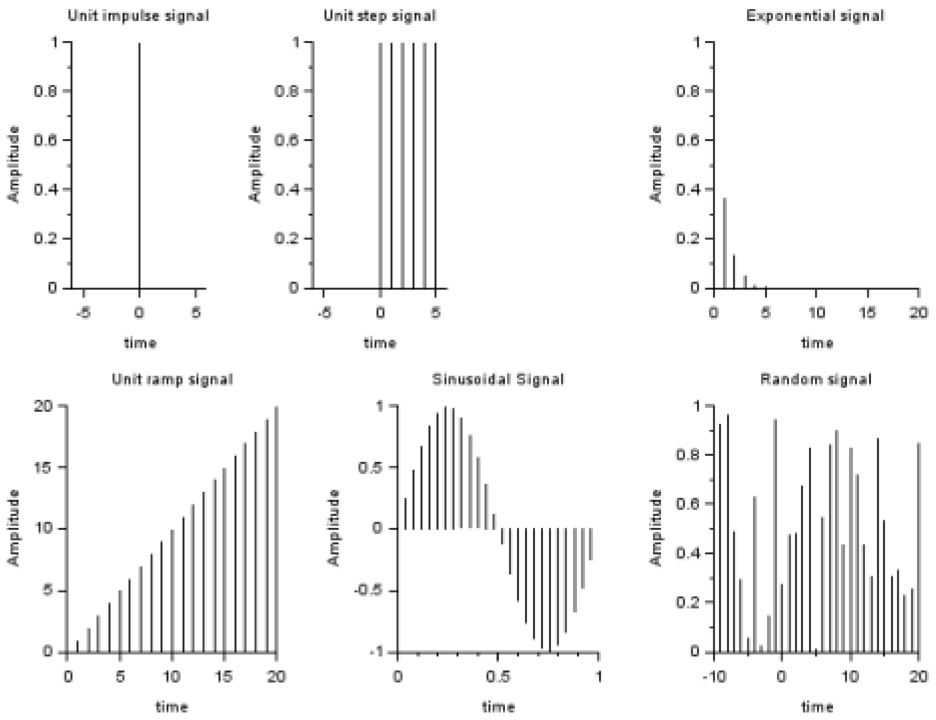
Ramp Signal: A ramp signal is a linearly increasing or decreasing signal with time. It's given by x[n] = a * n, where 'a' is the slope of the ramp. The ramp signal is commonly used to model linear changes or trends.

Random Signal: A random signal represents random variations or noise in a system. It's often generated using a random number generator. In digital communication, noise can introduce errors in the received signal, impacting the quality of communication.

Program:

```
        //UNIT IMPULSE SIGNAL
           clear all;
    close ;
    N =5; //SET LIMIT
     t1 = -5:5;
    x1 =[ zeros (1 , N ) ,ones (1 ,1) ,zeros (1 , N ) ];
    subplot (2 ,4 ,1) ;
    plot2d3 ( t1 , x1 )
    xlabel ( ' tim e ' ) ;
    ylabel ( ' Ampli tude ' ) ;
    title ( ' Uni t im p ul s e s i g n a l ' ) ;
    //UNIT STEP SIGNAL
    t2 = -5:5;
    x2 =[ zeros (1 , N ) ,ones (1 , N +1) ];
            subplot (2 ,4 ,2) ;
             plot2d3 ( t2 , x2 )
             xlabel ( ' tim e ' ) ;
             ylabel ( ' Ampli tude ' ) ;
            title ( ' Uni t s t e p s i g n a l ' ) ;

             //EXPONENTIAL SIGNAL
             t3 =0:1:20;
            x3 =exp( - t3 ) ;
            subplot (2 ,3 ,3) ;
             plot2d3 ( t3 , x3 ) ;
             xlabel ( ' tim e ' ) ;
             ylabel ( ' Ampli tude ' ) ;
             title ( ' E x p o n e n t i a l s i g n a l ' ) ;
             //UNIT RAMP SIGNAL 4
            t4 =0:20;
             x4 = t4 ;
             subplot (2 ,3 ,4) ;
             plot2d3 ( t4 , x4 ) ;
             xlabel ( ' tim e ' ) ;
             ylabel ( ' Ampli tude ' ) ;
             title ( ' Uni t ramp s i g n a l ' ) ;

             //SINUSOIDAL SIGNAL
            t5 =0:0.04:1;
            x5 =sin (2* %pi * t5 ) ;
            subplot (2 ,3 ,5) ;
            plot2d3 ( t5 , x5 ) ;
            title ( ’ S i n u s o i d a l S i g n a l ’ )
    xlabel ( ’ tim e ’ ) ;
     ylabel ( ’ Ampli tude ’ ) ;

     //RANDOM SIGNAL
     t6 = -10:1:20;
     x6 = rand (1 ,31) ;
     subplot (2 ,3 ,6) ;
     plot2d3 ( t6 , x6 ) ;
     xlabel ( ’ tim e ’ ) ;
     ylabel ( ’ Ampli tude ’ ) ;
     title ( ’ Random s i g n a l ’ ) ;
```

Output:



Result:

The following discrete-time signals were successfully generated and plotted using SCILAB:For each signal, the plots were labeled appropriately, with the x-axis representing time (n) and the y-axis representing the signal values.

EX NO: 2	

Test case 1:		Digital communication using binary phase Shift keying

Using SCILAB command, represent the binary Phase Shift Keying (PSK) signal with the given Binary information signal as [10101011] and Sampling frequency Fs = 100Hz.

Aim:

To generate a Binary Phase Shift Keying (BPSK) signal with the given input binary signal and sampling frequency F_s = 100Hz using SCILAB code and get the output waveform.

Software Required:

1. Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

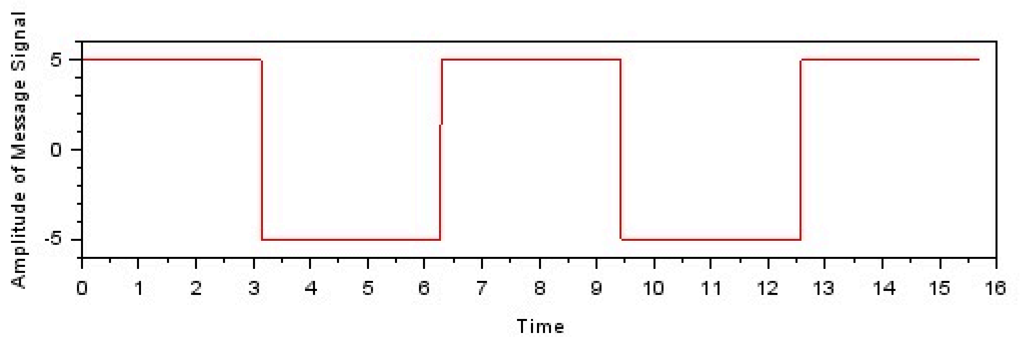
Theory:

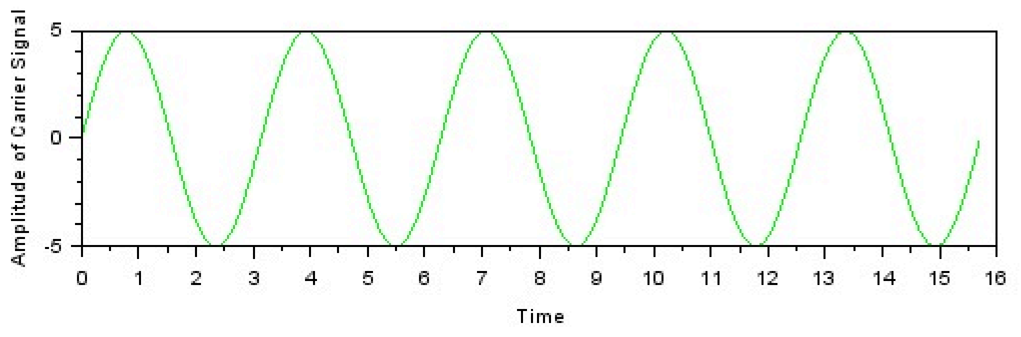
This program simulates BPSK modulation and demodulation using a slightly different approach. It generates random binary data, modulates it using BPSK modulation with a carrier frequency, adds noise to simulate a noisy channel, and then demodulates the received signal. Finally, it calculates the Bit Error Rate (BER) to assess the communication system's performance.

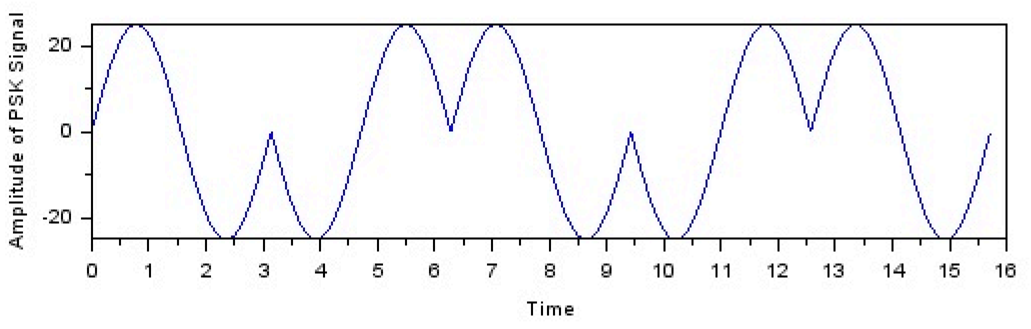
Program:

```
clc;clear all;clf;
t=[0:0.01:5*%pi];
A=5;
fc=2;
Vm=A.*squarewave(t);
Vc=A.*sin(fc.*t);
Vp= Vm.*Vc;
subplot(3,1,1);
plot(t,Vm, 'red');
xlabel("Time")
ylabel("Amplitude of Message Signal")
subplot(3,1,2);
plot(t,Vc, 'green');
xlabel("Time")
ylabel("Amplitude of Carrier Signal")
subplot(3,1,3);
plot(t,Vp, 'blue');
xlabel("Time")
ylabel("Amplitude of PSK Signal")
```

Output:







Result:

Binary Phase Shift Keying (BPSK) signal with the given input binary signal and sampling frequency F_s = 100Hz using SCILAB code was generated and obtained the output waveform.

EX NO: 3	N point DFT sequence, N point IDFT sequence, Circular convolution

Aim:

To analyze discrete-time signals in the frequency domain by computing their N-point Discrete Fourier Transform (DFT), reconstruct the time-domain signal using the Inverse Discrete Fourier Transform (IDFT), and perform circular convolution to understand the relationship between time-domain and frequency-domain operations.

Software Required:

1. Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

N-point Discrete Fourier Transform (DFT):

Given a sequence of N complex numbers {x[n]}, the N-point DFT is defined as follows:

DFT[k] = Σ (x[n] * exp(-j * 2π * k * n / N)), for n = 0 to N-1

Where:

🎬DFT[k] is the k-th frequency component of the DFT.

🎬x[n] is the input sequence.

🎬N is the total number of samples in the sequence.

🎬j is the imaginary unit.

The DFT result gives you the frequency components present in the input sequence, ranging from k = 0 to k = N-1. The magnitudes and phases of these components represent the frequency content of the input signal.

N-point Inverse Discrete Fourier Transform (IDFT):

Given an N-point DFT sequence {X[k]}, the N-point IDFT is defined as follows:

IDFT[n] = (1/N) * Σ (X[k] * exp(j * 2π * k * n / N)), for k = 0 to N-1

Where:

🎬IDFT[n] is the n-th sample of the IDFT sequence.

🎬X[k] is the k-th frequency component of the DFT sequence.

🎬N is the total number of samples in the sequence.

🎬j is the imaginary unit.

The IDFT operation reconstructs the original time-domain sequence from its frequency-domain representation (DFT). It's worth noting that the IDFT result should match the original input sequence if the DFT and IDFT are implemented correctly.

Both the DFT and IDFT can be efficiently calculated using algorithms like the Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT), respectively, to reduce computation time.

Circular Convolution

Circular convolution is a fundamental operation in digital signal processing, and it is often used to convolve two finite-length sequences that are treated as periodic signals.Circular convolution is an important operation in digital signal processing that combines two finite-length sequences in a circular or periodic manner. It is used to model the convolution of signals that are periodic or have a repeating nature. Circular convolution is distinct from linear convolution, which is commonly used for finite-length sequences.

Mathematically, the circular convolution of two sequences x[n] and h[n], denoted as y[n], is given by:

Program

```
clear;

clc ;

close ;

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

//DFT Computation

X = fft (x , -1);

//Display sequence X[k] in command window

disp(X,"X[k]=");
```

Output

```
X[k]=

      column 1 to 5
```

```
    6.  - 0.7071068 - 1.7071068i    1. - i      0.7071068 + 0.2928932i    0

 column 6 to 8

    0.7071068 - 0.2928932i    1. + i    - 0.7071068 + 1.7071068i
```

Program -4 point IDFT

```
clear;
clc ;
close ;
Y = [1,0,1,0];
//IDFT Computation
y = fft (Y , 1);
//Display sequence  y[n] in command window
disp(y,"y[n]=");
```

Output:

```
 y[n]=      0.5    0.    0.5    0.
```

Program -Circular Convolution

```
clear;
clc ;
close ;
x1=[0,1,2,3,4];
x2=[0,1,0,0,0];
//DFT Computation
X1=fft(x1,-1);
X2=fft(x2,-1);
Y=X1.*X2;
//IDFT Computation
y=round(fft(Y,1));
//Display sequence y[n] in command window
disp(y,"y[n]=");
```

Output:

```
   y[n]=      4.    0.    1.    2.    3.
```

Result: Thus the 8 point DFT, 4 point IDFT and circular convolution has been computed

EX NO: 4	

(i) Verify the correctness of your implementation of the overlap-save method.Input Signal: x[n]=[3,0,−2,0,2,1,0,−2,−1,0],Impulse Response: h[n]=[2,2,1], FFT Length: $L=2^M=2^3=8$. Use the overlap-save method and verify the result against direct convolution

Aim

To find the output whose input and impulse signal

$x(n) = \{3\ 0\ -\ 2\ 0\ 2\ 1\ 0\ -\ 2\ -\ 1\ 0\}$ and $h(n) = \{2, 2, 1\}$ using overlap add method

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

Sectional Convolution:
To reduce computational complexity, sectional convolution divides the input sequence into smaller sections. The two common approaches are: Overlap-Save Method,Overlap-Add Method.Both methods rely on the use of FFT to compute the convolution efficiently. Overlap-Save Method.

The Overlap-Save Method is a block convolution technique where:

The input sequence is divided into overlapping sections.

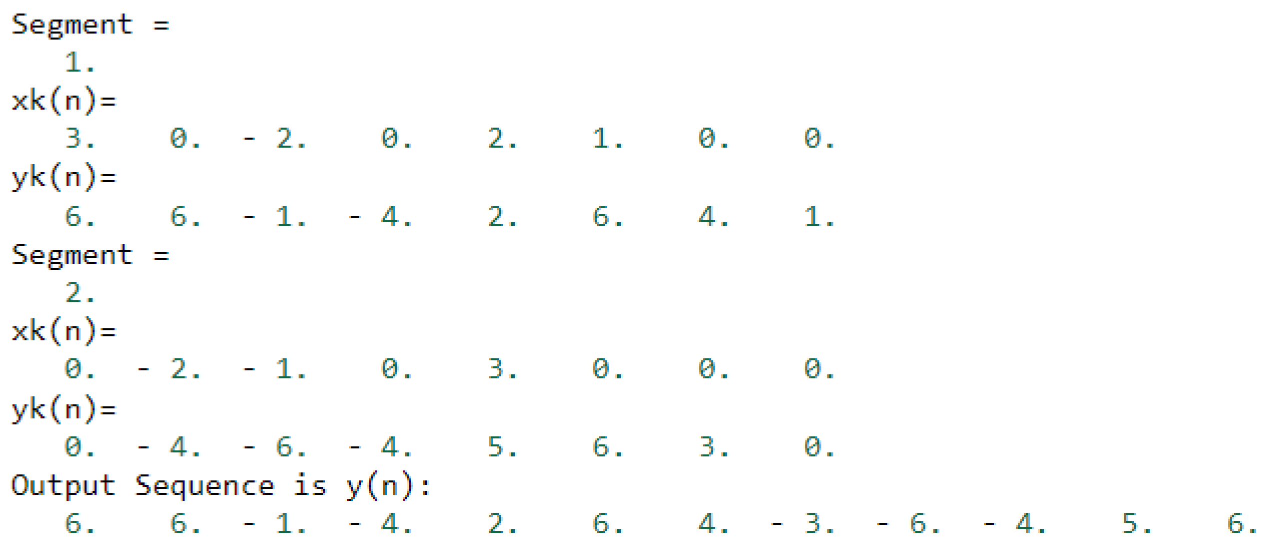
Each section is convolved with the impulse response using the FFT.
The overlapping parts of each section are discarded (hence "save"), and the non-overlapping parts are concatenated to form the final output.

Program

```
clc;clear;close;
h=[2 2 1];
x=[3 0 -2 0 2 1 0 -2 -1 0];
M=length(h);              //length of impulse response
L=2^M;                  //length of FFT/IFFT operation
N=L-M+1;
xl=length(x);
K=ceil(xl/N);             //number of iterations
h=[h zeros(1,L-M)];
x=[x x(1:K*N-xl)];
H=fft(h);
y=zeros(1,M-1);
for k=0:K-1
```

```
        xk=[x(k*N+1:(k+1)*N) zeros(1,M-1)];
        Xk=fft(xk);
        Yk=H.*Xk;
        yk=ifft(Yk);
        yk=clean(yk);
        y=[y(1:k*N) y(k*N+1:k*N+M-1)+yk(1:M-1) yk(M:L)];
        disp(k+1,'Segment =');
        disp(xk,'xk(n)=');
        disp(yk,'yk(n)=');
    end
    y=y(1:xl+M-1);
    disp(y,'Output Sequence is y(n): ');
```

Output:



```
Segment =
   1.
xk(n)=
   3.    0.  - 2.    0.    2.    1.    0.    0.
yk(n)=
   6.    6.  - 1.  - 4.    2.    6.    4.    1.
Segment =
   2.
xk(n)=
   0.  - 2.  - 1.    0.    3.    0.    0.    0.
yk(n)=
   0.  - 4.  - 6.  - 4.    5.    6.    3.    0.
Output Sequence is y(n):
   6.    6.  - 1.  - 4.    2.    6.    4.  - 3.  - 6.  - 4.    5.    6.
```

Result: Thus the overlap-save method efficiently computes the convolution for long input sequences.

(ii) Input Signal: x[n]=[1,2,−1,2,3,−2,−3,−1,1,1,2,−1], Impulse Response: h[n]=[1,2] Compute the convolution of x[n] with h[n] using the overlap-add method. Compare this with the output obtained using the conv() function in SCILAB to verify the correctness.

Aim

Find the output whose input and impulse signal

$x(n)$ =[1 2 -1 2 3 -2 -3 -1 1 1 2 -1]
and $h(n)$ = [1 2] using overlap save and overlap Save method

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

Sectional Convolution
To reduce computational complexity, sectional convolution divides the input sequence into smaller sections. The two common approaches are:
Overlap-Save Method
Overlap-Add Method
Both methods rely on the use of FFT to compute the convolution efficiently. Overlap-Save Method and Overlap add Method.
The Overlap-add Method is a block convolution technique where:
The Overlap-Add Method divides the input sequence into non-overlapping sections, and the sections are processed as follows:
The input sequence is divided into non-overlapping sections.
Each section is convolved with the impulse response using the FFT.
The results of the convolutions are added together with overlapping parts summed (hence "add").

Program:

```
clc;
clear;
close;
h=[1 2];
x=[1 2 -1 2 3 -2 -3 -1 1  1 2 -1]
M=length(h);
L=2^M;
N=L-M+1;
xl=length(x);
K=ceil(xl/N);
h=[h zeros(1,L-M)];
x=[x x(1:K*N-xl)];
H=fft(h);
y=zeros(1,M-1);
for k=0:K-1
   xk=[x(k*N+1:(k+1)*N)zeros(1,M-1)];
   Xk=fft(xk);
```

```
    Yk=H.*Xk;
    yk=ifft(Yk);
    yk=clean(yk);
    y=[y(1:k*N)y(k*N+1:k*N+M-1)+yk(1:M-1) yk(M:L)];
    disp('Segment =',k+1);
    disp('xk(n)=',xk);
    disp('yk(n)=',yk);
end
y=y(1:xl+M-1);
disp('Output Sequence is y(n): ',y);
```

Output:

```
Enter the input sequence=[1 2 -1 2 3 -2 -3 -1 1  1 2 -1]
Enter the lower index of the input sequence: 0
Enter the impulse response sequence:  [1 2]
Enter the lower index of impulse response sequence= 0
  "Output sequence using overlap save method Y(n): "
       column 1 to 13
   1.   4.   3.   0.   7.   4.  -7.  -7.  -1.   3.   4.   3. -2
```

Result: Both methods effectively compute the convolution.The output of the overlap-add method is identical to the direct convolution output for sufficiently large signals.

EX NO: 5	**DIT-FFT and DIF-FFT**

DATE:	**Algorithm**

(i) Given a sequence x[n]=[1,-1,-1,-1,1,1,1,-1] compute the DFT using the DIT-FFT algorithm. The sequence exhibits real and symmetric properties. How does symmetry affect the twiddle factor calculations in the DIT-FFT algorithm?

Aim

(i) To Compute the DFT of given Sequence x[n]=[1,-1,-1,-1,1,1,1,-1] using DIT-FFT Algorithm.

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

The Decimation-in-Time Fast Fourier Transform (DIT-FFT) algorithm is an efficient method to compute the Discrete Fourier Transform (DFT) of a sequence. The FFT algorithm reduces the computational complexity of the DFT from

The DFT of a sequence $x[n]$ of length N is given by:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi kn}{N}}, \quad k = 0, 1, 2, \ldots, N-1$$

DIT-FFT Algorithm

The DIT-FFT algorithm is a divide-and-conquer approach that breaks down the DFT computation into smaller parts, recursively decimating the sequence into smaller subsequences. The key idea behind DIT-FFT is to separate the original sequence into even-indexed and odd-indexed terms, recursively applying the FFT on these smaller sequences.

Program

```
clear;
clc ;
close ;
x = [1,-1,-1,-1,1,1,1,-1];
//FFT Computation
X = fft (x , -1);
disp(X,'X(z) = ');
```

Output

```
        column 1 to 5
 0  - 1.4142136 + 3.4142136i    2. - 2.i    1.4142136 - 0.5857864i    4.
        column 6 to 8
 1.4142136 + 0.5857864i    2. + 2.i  - 1.4142136 - 3.4142136i
```

Result: Thus the DFT of the sequence x[n]=[1,-1,-1,-1,1,1,1,-1] using the DIT-FFT algorithm results in the expected frequency components, confirming the correct implementation of the algorithm.

(ii) Given a sequence x[n]=[1,2,3,4,4,3,2,1] compute the DFT using the DIF-FFT algorithm. The sequence exhibits real and symmetric properties. How does symmetry affect the twiddle factor calculations in the DIF-FFT algorithm?

Aim

To compute the DFT of the sequence x[n]=[1,2,3,4,4,3,2,1] using the Decimation-in-Frequency (DIF) FFT algorithm, we will utilize the symmetry of the sequence to optimize the twiddle factor calculations.

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

DIF-FFT Algorithm

The Decimation-in-Frequency Fast Fourier Transform (DIF-FFT) is another efficient algorithm for computing the Discrete Fourier Transform (DFT). Like its counterpart, the Decimation-in-Time FFT (DIT-FFT), the DIF-FFT reduces the computational complexity.

The DFT of a sequence $x[n]$ of length N is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi kn}{N}}, \quad k = 0, 1, 2, \ldots, N-1$$

The DIF-FFT algorithm, like the DIT-FFT, is a divide-and-conquer approach. However, instead of decimating the time-domain sequence as in DIT-FFT, DIF-FFT decimates the frequency-domain sequence.

Program:

```
clear;
clc ;
close ;
```

```
x = [1,2,3,4,4,3,2,1];
//FFT Computation
X = fft (x , -1);
disp(X,'X(z) = ');
```

Output:

```
X(z) =
      column 1 to 5
  20.  - 5.8284271 - 2.4142136i    0  - 0.1715729 - 0.4142136i    0
      column 6 to 8
 - 0.1715729 + 0.4142136i    0  - 5.8284271 + 2.4142136i
```

Result: Thus the DFT of the sequence x[n]=[1,2,3,4,4,3,2,1] using the DIF-FFT algorithm results in the expected frequency components, confirming the correct implementation of the algorithm. Symmetry in the sequence reduces the number of unique twiddle factor calculations, enhancing the algorithm's efficiency.

EX NO: 6	**Denoising Data using FFT**
DATE:	

Aim

To

generated as noisy_signal = sin(2*t) + 0.5*rand(t), where the time vector t is from 0 to 2π with an interval of 0.001. The signal is denoised using FFT with a threshold value of 50 in the frequency domain using the Fast Fourier Transform (FFT) in SCILAB.

Software Required

1. Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

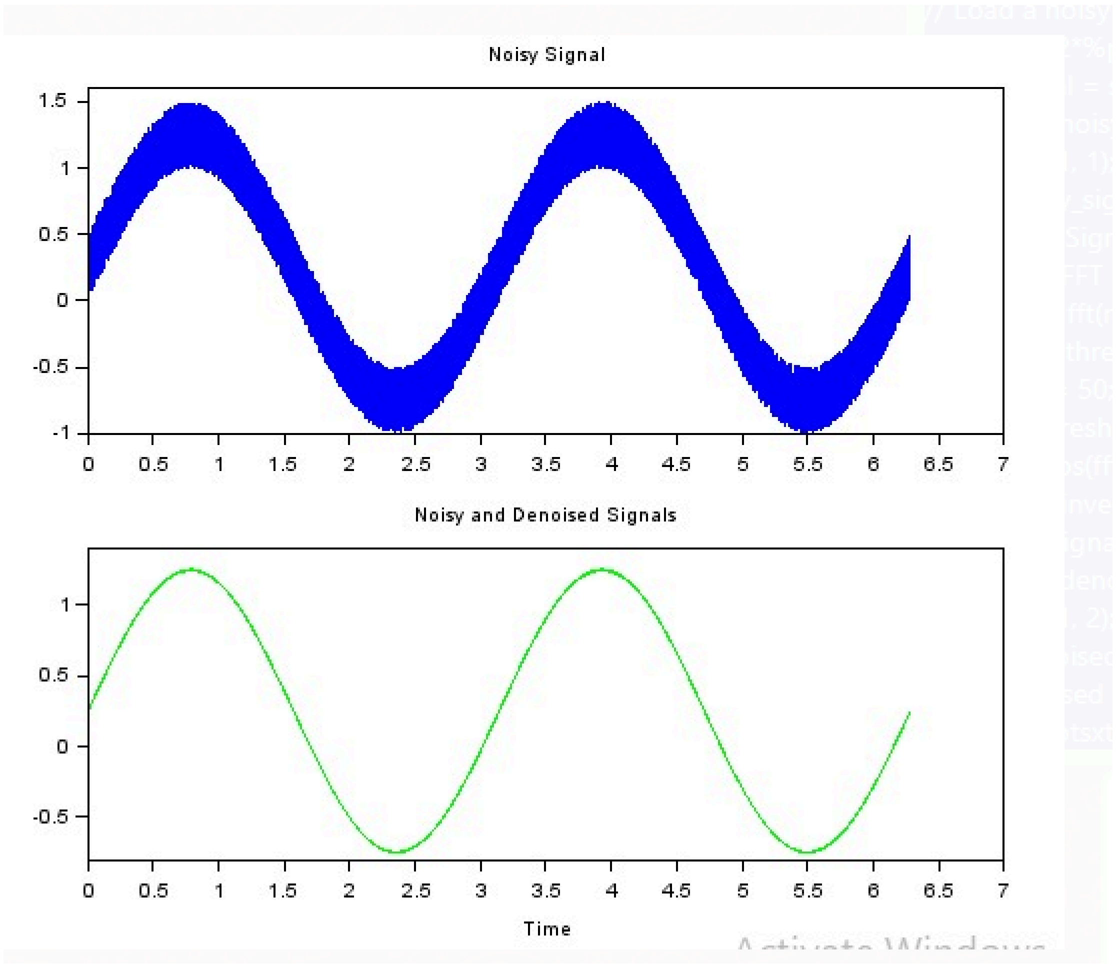
Theory:

Generate a noisy signal (a sine wave with added random noise). Then, we perform the FFT on the noisy signal, apply a threshold in the frequency domain, and finally, use the inverse FFT to obtain the denoised signal. The thresholding step helps to remove the high-frequency noise components.

Program:

```
clear;clc;
// Clear workspace and console
// Load a noisy signal (you can replace this with your own data)
t =0:0.001:2*%pi;
noisy_signal = sin(2*t)+0.5*rand (t);
// Plot the noisy signal
subplot(2, 1, 1);
plot(t, noisy_signal, 'b');
title('Noisy Signal');
// Perform FFT on the noisy signal
fft_result = fft(noisy_signal);
// Define a threshold for denoising (you can adjust this)
threshold = 50;
// Apply thresholding in the frequency domain
fft_result(abs(fft_result) < threshold) = 0;
// Perform inverse FFT to obtain the denoised signal
denoised_signal = ifft(fft_result);
// Plot the denoised signal
subplot(2, 1, 2);
plot(t, denoised_signal, 'g');
title('Denoised Signal');
// Show plotsxtitle('Noisy and Denoised Signals', 'Time');
```

Output:



Result:

Thus the denoising process using FFT was effective in reducing noise from the signal. The thresholding technique successfully removed high-frequency noise components while preserving the main sinusoidal waveform

EX NO: 7	

DATE:	**Analog Filter design Using Transformation method**

(i) Validate Transformation with Standard Sampling Period T=1.Convert Analog transfer function into Digital using Bilinear Transformation of H(s)=(s^2+4.525)/(s^2+0.692*s+0.504) using sci lab

Aim

To convert the continuous-time analog transfer function H(s)=s2+4.525/s2+0.692s+0.504 into a discrete-time digital transfer function using the bilinear transformation method with a standard sampling period T=1

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

In digital signal processing, converting an analog filter design into a digital filter is crucial for implementing filtering operations in digital systems. The bilinear transformation is a widely used method for this purpose. It maps a continuous-time (analog) filter's transfer function into a discrete-time (digital) filter's transfer function while preserving the filter's characteristics such as stability and frequency response.

Bilinear Transformation

The bilinear transformation is a technique used to convert an analog filter's transfer function H(s)H(s)H(s) into a discrete-time transfer function H(z)H(z)H(z). The transformation is defined by the mapping:

$$s = \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}}$$

where T is the sampling period. This mapping converts the Laplace variable S to the Z-transform variable Z.

Program

```
clear;
clc ;
close ;
```

```
s=%s;
z=%z;
HS=(s^2+4.525)/(s^2+0.692*s+0.504);
T=1;
HZ=horner(HS,(2/T)*(z-1)/(z+1));
disp(HZ,'H(z) =');
```

Output:

```
H(z) =
                              2
   1.4478601 + 0.1783288z + 1.4478601z
   ----------------------------------
                          2
        0.5298913 - 1.1875z + z
```

Result: Thus the discrete-time transfer function H(z) obtained after applying the bilinear transformation. Include the coefficients of the numerator and denominator of H(z).

(ii)Validate Transformation with Standard Sampling Period T=0.2Convert Analog transfer function into Digital using Impulse Invariant Transformation of H(S)=10/(s^2+7*s+10) using sci lab

Aim

To convert the continuous-time analog transfer function H(S)=10/(s^2+7*s+10) into a discrete-time digital transfer function using Impulse Invariant transformation method with a standard sampling period T=0.2

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

Impulse Invariant Transformation is a method used to convert an analog filter design into a digital filter. This method ensures that the impulse response of the digital filter closely approximates the impulse response of the analog filter. It is particularly useful when designing digital filters that need to maintain a similar impulse response to their analog counterparts.Impulse Invariant Transformation maps the continuous-time (analog) filter to a discrete-time (digital) filter by preserving the impulse response of the analog filter. This method ensures that the discrete-time filter has the same impulse response as the continuous-time filter sampled at discrete intervals.

Program

```
//To Design the Filter using Impulse Invarient Method
        clear;
        clc ;
        close ;
        s=%s;
        T=0.2;
        HS=10/(s^2+7*s+10);
        elts=pfss(HS);
        disp(elts,'Factorized HS = ');
        //The poles comes out to be at -5 and -2
        p1=-5;
        p2=-2;
        z=%z;
        HZ=T*((-3.33/(1-%e^(p1*T)*z^(-1)))+(3.33/(1-%e^(p2*T)*z^(-1))))
        disp(HZ,'HZ = ');
```

Output:

```
Factorized HS =
      (1)
   3.3333333
   ---------
     2 + s
      (2)
 - 3.3333333
   ---------
     5 + s
 HZ  =
         0.2014254z
   -------------------------

   0.2465970 - 1.0381995z + Z2

 HZ =
         0.2014254z
   --------------------------

   0.2465970 - 1.0381995z + z2
```

Result: Thus the discrete-time transfer function H(z) obtained after applying the Impulse Invariant transformation. Include the coefficients of the numerator and denominator of H(z).

EXNO: 8	**Analog Butterworth Filter**
DATE:	

(i)Design a Butterworth filter to process audio signals that attenuates frequencies above 0.2 * pi and maintains a flat frequency response in the passband. Compare the output of the filtered signal with the original signal in both time and frequency domains. Plot both to verify the attenuation of high frequencies.

Aim

To design a Butterworth filter for processing audio signals that attenuates frequencies above 0.2π0.2\pi0.2π radians per sample and maintains a flat frequency response in the passband. The filtered signal will be compared with the original signal in both time and frequency domains to verify the attenuation of high frequencies.Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

The Butterworth low-pass filter is a widely used type of analog filter that is known for its maximally flat frequency response in the passband, which means it has no ripples. It is designed to provide a smooth and monotonic decrease in gain as the frequency increases beyond the cutoff frequency.

Program:

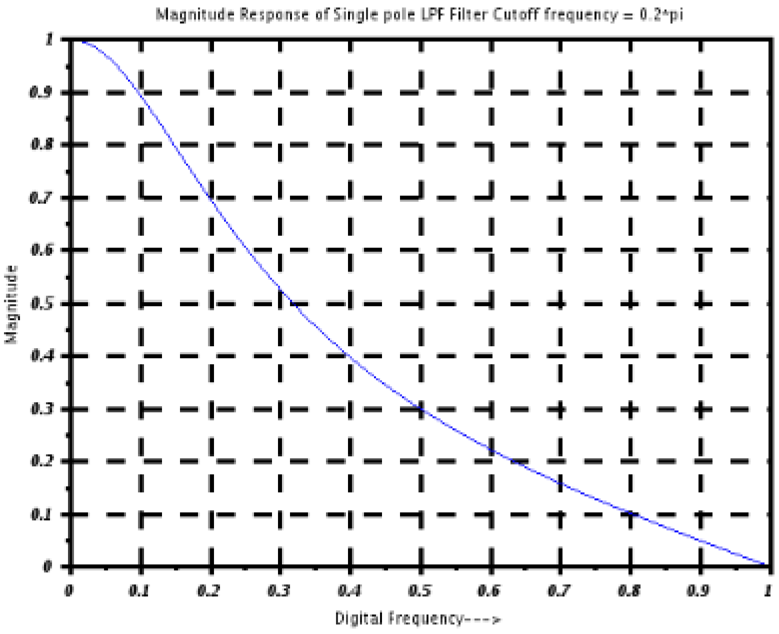
//First Order Butterworth Low Pass Filter

```
clear;
clc;
close;
s = poly(0,'s');
Omegac = 0.2*%pi;
H = Omegac/(s+Omegac);
T =1;//Sampling period T = 1 Second
z = poly(0,'z');
Hz = horner(H,(2/T)*((z-1)/(z+1)))
HW  =frmag(Hz(2),Hz(3),512);
W = 0:%pi/511:%pi;
plot(W/%pi,HW)
a=gca();
a.thickness = 3;
a.foreground = 1;
a.font_style = 9;
xgrid(1)
xtitle('Magnitude Response of Single pole LPF Filter Cutoff frequency = 0.2*pi','Digital
Frequency--->','Magnitude');
```

```
Disp("Hz",Hz);
```

Output:

```
Hz  =
    0.6283185 + 0.6283185z
    ----------------------
  - 1.3716815 + 2.6283185z
```



Result:

The Butterworth filter successfully attenuated frequencies above 0.2π0.2\pi0.2π, confirming its design. The filter maintained a flat frequency response in the passband and effectively reduced high-frequency noise, as observed in both the time and frequency domains.

(ii)Design a Butterworth filter that allows frequency range above the cut off frequency of 0.2*pi for a digital audio processing application.

Aim

The aim of this experiment is to design a Butterworth high-pass filter for a digital audio processing application, allowing frequencies above the cutoff frequency of 0.2×π0.2 \times \pi0.2×π radians per second to pass, while attenuating frequencies below the cutoff. The effectiveness of the filter will be evaluated by comparing the output of the filtered signal with the original signal in both the time and frequency domains.Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

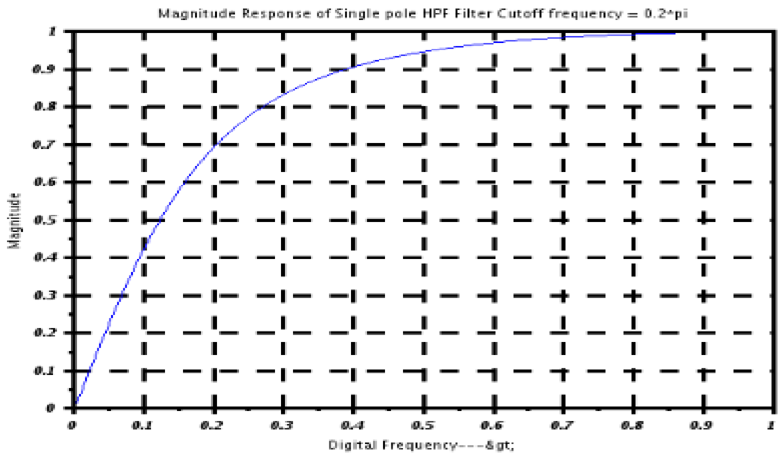
Theory:
The Butterworth high-pass filter is an analog filter designed to allow high-frequency signals to pass through while attenuating low-frequency signals. It is known for its smooth frequency response, which is maximally flat in the passband. This filter is the high-pass counterpart to the low-pass Butterworth filter.

Program

```
//First Order Butterworth Filter
    //High Pass Filter Using Digital Filter Transformation
    clear;
    clc;
    close;
    s = poly(0,'s');
    Omegac = 0.2*%pi;
    H = Omegac/(s+Omegac);
    T =1;//Sampling period T = 1 Second
    z = poly(0,'z');
    Hz_LPF = horner(H,(2/T)*((z-1)/(z+1)));
    alpha = -(cos((Omegac+Omegac)/2))/(cos((Omegac-Omegac)/2));
    HZ_HPF=horner(Hz_LPF,-(z+alpha)/(1+alpha*z))
    HW  =frmag(HZ_HPF(2),HZ_HPF(3),512);
    W = 0:%pi/511:%pi;
    plot(W/%pi,HW)
    a=gca();
    a.thickness = 3;
    a.foreground = 1;
    a.font_style = 9;
    xgrid(1)
    xtitle('Magnitude Response of Single pole HPF Filter Cutoff frequency = 0.2*pi','Digital
Frequency---&gt;','Magnitude');
    disp("HZ_HPF",HZ_HPF);
```

Output:

```
    HZ_HPF  =
    - 0.7484757 + 0.7484757z
      ----------------------
       - 0.4969514 + z
```



Result:

The Butterworth high-pass filter was successfully designed with a cutoff frequency of 0.2×π0.2 \times \pi0.2×π. In the time domain, (iii)For a signal processing application design a Butterworth filter to isolate a specific frequency range from an audio signal between 0.4 pi and 0.6 pi.

Aim

To design a Butterworth band-pass filter for a signal processing application to isolate a specific frequency range between 0.4π and 0.6π radians per second from an audio signal.

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

The Butterworth band-pass filter is an analog filter designed to pass frequencies within a certain range (the passband) while attenuating frequencies outside this range. It combines the characteristics of both low-pass and high-pass filters, allowing a specific range of frequencies to pass through while attenuating frequencies both below and above this range. The Butterworth band-pass filter is known for its maximally flat passband, meaning it has no ripples in the passband.

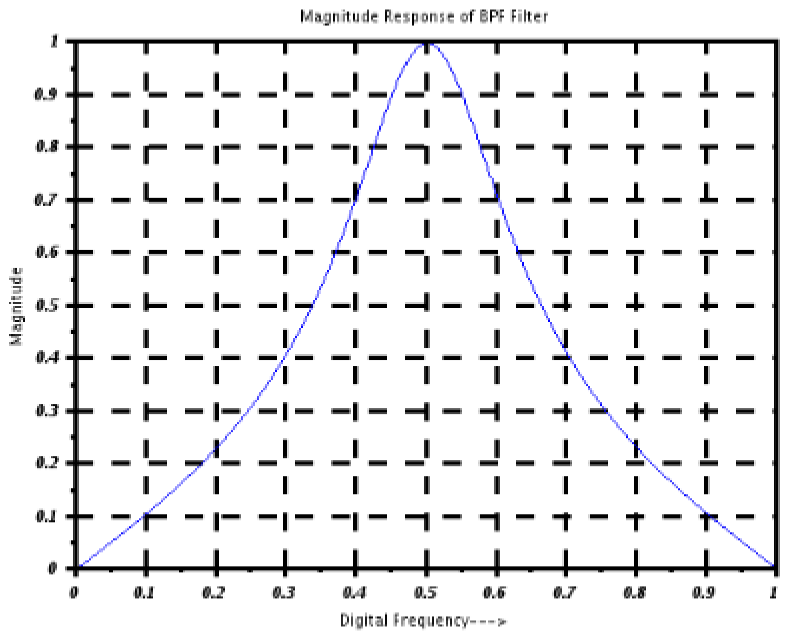
Program:

```
clear;
clc;
close;
omegaP = 0.2*%pi;
omegaL =  (2/5)*%pi;
omegaU =  (3/5)*%pi;
z=poly(0,'z');
H_LPF = (0.245)*(1+(z^-1))/(1-0.509*(z^-1))
alpha = (cos((omegaU+omegaL)/2)/cos((omegaU-omegaL)/2));
k = (cos((omegaU - omegaL)/2)/sin((omegaU - omegaL)/2))*tan(omegaP/2);
NUM =-((z^2)-((2*alpha*k/(k+1))*z)+((k-1)/(k+1)));
DEN = (1-((2*alpha*k/(k+1))*z)+(((k-1)/(k+1))*(z^2)));
HZ_BPF=horner(H_LPF,NUM/DEN)
disp(HZ_BPF,'Digital BPF IIR Filter H(Z)= ')
HW  =frmag(HZ_BPF(2),HZ_BPF(3),512);
W = 0:%pi/511:%pi;
plot(W/%pi,HW)
a=gca();
a.thickness = 3;
```

```
    a.foreground = 1;
    a.font_style = 9;
    xgrid(1)
    xtitle('Magnitude Response of BPF Filter', 'Digital Frequency--->','Magnitude');
  Disp("HZ_BPF",HZ_BPF);
```

Output:

```
    H_LPF  =
       0.245 + 0.245z
       --------------
       - 0.509 + z

     HZ_BPF  =
                                 2            3            4
       0.245 - 1.577D-17z - 0.245z + 1.577D-17z + 1.360D-17z
       -----------------------------------------------------
                               2            3            4
       - 0.509 + 1.299D-16z - z + 6.438D-17z + 5.551D-17z

    Digital BPF IIR Filter H(Z)=
                                 2            3            4
       0.245 - 1.577D-17z - 0.245z + 1.577D-17z + 1.360D-17z
       -----------------------------------------------------
                               2            3            4
       - 0.509 + 1.299D-16z - z + 6.438D-17z + 5.551D-17z
```



Result:

Thus the Butterworth band-pass filter was successfully designed to isolate the frequency range between 0.4π and 0.6π inn the time domain,

(iv) For a signal processing application design a Butterworth filter to attenuate a specific frequency range from an audio signal between 0.4 pi and 0.6 pi.

Aim

The aim of this experiment is to design a Butterworth band-stop filter for a signal processing application to attenuate a specific frequency range between 0.4π and 0.6π radians per second from an audio signal. using Sci lab

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

The Butterworth band-reject filter, also known as a band-stop or band-elimination filter, is an analog filter designed to attenuate frequencies within a specific range while allowing frequencies outside this range to pass with minimal attenuation. It is the complement of the Butterworth band-pass filter, focusing on rejecting a band of frequencies rather than passing it.

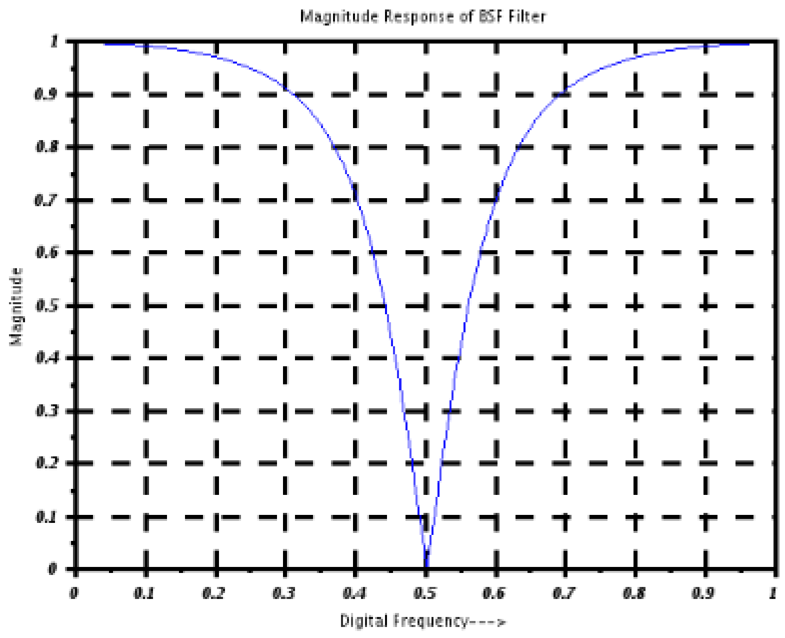
Program:

```
clear;
clc;
close;
omegaP = 0.2*%pi;
omegaL =  (2/5)*%pi;
omegaU =  (3/5)*%pi;
z=poly(0,'z');
H_LPF = (0.245)*(1+(z^-1))/(1-0.509*(z^-1))
alpha = (cos((omegaU+omegaL)/2)/cos((omegaU-omegaL)/2));
k = tan((omegaU - omegaL)/2)*tan(omegaP/2);
NUM =((z^2)-((2*alpha/(1+k))*z)+((1-k)/(1+k)));
DEN = (1-((2*alpha/(1+k))*z)+(((1-k)/(1+k))*(z^2)));
HZ_BSF=horner(H_LPF,NUM/DEN)
HW  =frmag(HZ_BSF(2),HZ_BSF(3),512);
W = 0:%pi/511:%pi;
plot(W/%pi,HW)
a=gca();
a.thickness = 3;
```

```
a.foreground = 1;
a.font_style = 9;
xgrid(1)
xtitle('Magnitude Response of BSF Filter','Digital Frequency--->','Magnitude');
Disp(“HZ_BSF”,HZ_BSF);
```

Output:

```
HZ_BPF  =
                                          2
   0.7534875 - 9.702D-17z + 0.7534875z
   ----------------------------------
                                   2
      0.5100505 - 9.722D-17z + z
```



Result:

The Butterworth band-stop filter was successfully designed to attenuate frequencies between 0.4π and 0.6π. In the time domain, the filtered signal exhibited a clear reduction of frequencies within the specified range

EX NO: 9	Design of Analog Chebyshev Filter
DATE:	

(i)Evaluate the provided SCILAB code for designing a Chebyshev Type I low-pass filter with the following parameters:

- **Analog Passband Edge Frequency:** 1000π radians/second
- **Analog Stopband Edge Frequency:** 2000π radians/second
- **Passband Ripple:** -1 dB
- **Stopband Attenuation:** -40 dB

Aim:

To meet the desired passband and stopband frequency constraints with minimal ripple in the passband and significant attenuation in the stopband, using the characteristics of the Chebyshev Type I design to achieve a sharper cutoff compared to Butterworth filters.

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes, type the program and save the program in current directory
Compile and run the program
If any error occur in the program, correct the error and run the program
For the output, see the console window
Stop the program

Theory

The code designs a Chebyshev Type I low-pass filter with specified passband and stopband frequencies. It calculates the filter order using given passband ripple and stopband attenuation values. The filter's order is determined to achieve the desired performance, with a focus on the sharpness of the roll-off between the passband and stopband. The poles of the filter are computed, which define its frequency response. Finally, the code plots the filter's magnitude response to visualize how effectively it attenuates high-frequency signals while allowing low-frequency signals to pass.

Program

```
clear;
clc;
close;
omegap = 1000*%pi; //Analog Passband Edge frequency in radians/sec
omegas = 2000*%pi; //Analog Stop band edge frequency in radians/sec
delta1_in_dB = -1;
delta2_in_dB = -40;
```

```
delta1 = 10^(delta1_in_dB/20);
delta2 = 10^(delta2_in_dB/20);
delta = sqrt(((1/delta2)^2)-1)
epsilon  = sqrt(((1/delta1)^2)-1)
//Calculation of Filter order
num = ((sqrt(1-delta2^2))+(sqrt(1-((delta2^2)*(1+epsilon^2)))))/(epsilon*delta2)
den = (omegas/omegap)+sqrt((omegas/omegap)^2-1)
N = log10(num)/log10(den)
//N = (acosh(delta/epsilon))/(acosh(omegas/omegap))
N = floor(N)
//Cutoff frequency
omegac = omegap
//Calculation of poles and zeros
[pols,Gn] = zpch1(N,epsilon,omegap)
disp(N,'Filter order N =');
disp(pols,'Poles of a type I lowpass Chebyshev filter are Sk =')
//Analog Filter Transfer Function
xtitle('Magnitude Response of Chebyshev Type 1 LPF Filter Cutoff frequency = 500 Hz','Analog
frequency in Hz--->','Magnitude in dB -->');
```

Output:

```
delta  =
   99.995
 epsilon  =
   0.5088471
 num  =
   393.02315
 den  =
   3.7320508
 N  =
   4.536112
 N  =
   4.
 omegac  =
   3141.5927
 Gn  =
   2.393D+13
 pols  =
        column 1 to 3
 - 438.36526 + 3089.3768i  - 1058.3074 + 1279.6618i  - 1058.3074 - 1279.6618i
        column 4
 - 438.36526 - 3089.3768i
 Filter order N =
   4.
 Poles of a type I lowpass Chebyshev filter are Sk =
        column 1 to 3
  - 438.36526 + 3089.3768i  - 1058.3074 + 1279.6618i  - 1058.3074 - 1279.6618i
        column 4
  - 438.36526 - 3089.3768i
```

Result:

The Chebyshev Type I low-pass filter was successfully designed with a sharp cutoff at the passband edge frequency of 1000π radians/second, achieving the specified -1 dB ripple. The filter also meets the required stopband attenuation of -40 dB beyond 2000π radians/second.

(ii) Given the following parameters for a Chebyshev Type I high-pass filter design:

- Analog Passband Edge Frequency (ωp\omega_pωp) = 1000×π radians/second
- Analog Stopband Edge Frequency (ωs\omega_sωs) = 2000×π radians/second
- Passband Ripple (δ1) = -1 dB
- Stopband Attenuation (δ2) = -40 dB

1. Calculate the passband and stopband deviations (δ1 and δ2) in linear scale.
2. Compute the δ and ϵ parameters for the filter.
3. Determine the filter order NNN required to meet the specified design parameters.
4. Using the calculated filter order, find the poles of the Chebyshev Type I high-pass filter.
5. Display the filter order and the poles.

Aim:

To design a Chebyshev Type I high-pass filter by calculating its order and poles based on given passband and stopband specifications. The filter should meet the design criteria for passband ripple and stopband attenuation.

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory

The code designs a Chebyshev Type I high-pass filter with specified passband and stopband edge frequencies. It calculates the filter order N based on passband ripple (δ1) and stopband attenuation (δ2). The filter order determines the complexity and performance of the filter. Poles and zeros of the filter are computed to define its frequency response. The results, including filter order and poles, are displayed. This filter is characterized by a steeper roll-off and controlled ripple in the passband.

Program

```
// Define constants and parameters
omegap = 1000*%pi; // Analog Passband Edge frequency in radians/sec
omegas = 2000*%pi; // Analog Stop band edge frequency in radians/sec
delta1_in_dB = -1;
delta2_in_dB = -40;
delta1 = 10^(delta1_in_dB/20);
delta2 = 10^(delta2_in_dB/20);
delta = sqrt(((1/delta2)^2) - 1);
epsilon = sqrt(((1/delta1)^2) - 1);

// Calculation of Filter order
num = ((sqrt(1 - delta2^2)) + (sqrt(1 - ((delta2^2) * (1 + epsilon^2))))) / (epsilon * delta2);
den = (omegas / omegap) + sqrt((omegas / omegap)^2 - 1);
N = log10(num) / log10(den);
N = floor(N);
// Cutoff frequency
omegac = omegap;
// Calculation of poles and zeros
[pols, Gn] = zpch1(N, epsilon, omegap);
// Display results
disp(N, 'Filter order N =');
disp(pols, 'Poles of a type I highpass Chebyshev filter are Sk =');
```

Output:

```
4.

  "Filter order N ="

  -438.36526 + 3089.3768i  -1058.3074 + 1279.6618i  -1058.3074 - 1279.6618i  -438.36526 -
3089.3768i

  "Poles of a type I highpass Chebyshev filter are Sk ="
```

Result:

The Chebyshev Type I high-pass filter was designed with a calculated filter order of N that satisfies the -1 dB passband ripple and -40 dB stopband attenuation. The poles of the filter were determined based on the filter order, ensuring proper performance at the passband edge of 1000π radians/second.

(iii) For a band-pass filter design, given passband edge frequencies fp1 = 1.6, fp2 = 1.8, stopband edge frequencies fs1 = 3.2, fs2 = 4.8, passband ripple Ap = 2 dB, stopband attenuation As = 20 dB, and sampling frequency S = 12 kHz, calculate the filter order n, pre-warping frequencies W, and the transfer function HPS.

Aim

To design a band-pass filter using the indirect bilinear transformation method, calculating necessary parameters, and visualizing the frequency response of the filter.

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory

The code designs a band-pass filter using the indirect bilinear transformation method. It calculates prewrapped frequencies, filter order, and poles based on passband and stopband specifications. The filter's frequency response is analyzed by plotting its magnitude over a range of frequencies. The Horner function is used to simplify polynomial evaluations in the filter design process.

Program

Design of chebyshev IIR Band Pass filter with following specifications

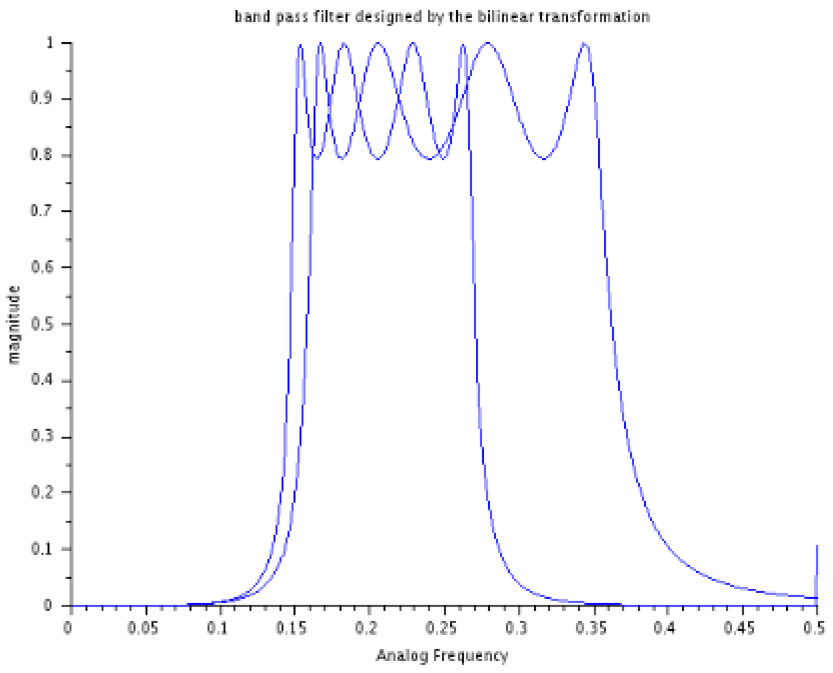
```
 fp1=1.6;fp2=1.8;fs1=3.2;fs2=4.8;//pass band edges
Ap=2;As=20;S=12;
s=%s;z=%z;
//(a)Indirect Bilinear design
W=2*%pi*[fp1 fp2 fs1 fs2]/S
C=2;
omega=2*tan(0.5*W');//prewarping each band edge frequency
epsilon=sqrt(10^(0.1*Ap)-1);
n=acosh(((10^(0.1*As)-1)/epsilon^2)^1/2)/(acosh(fs1/fp1));
n=ceil(n)
alpha=(1/n)*asinh(1/epsilon);
for i=1:n
   B(i)=(2*i-1)*%pi/(2*n);
end
for i=1:n
   p(i)=-sinh(alpha)*sin(B(i))+%i*cosh(alpha)*cos(B(i));
end
```

```
Qs=1;
for i=1:n
   Qs=Qs*(s-p(i))
end
Qo=0.1634;
HPS=Qo/Qs
HBPS=horner(HPS,(s^2+1.5045^2)/(s*1.202))
HZ=horner(HBPS,2*(z-1)/(z+1))
f=0:0.001:0.5;
HZF=abs(horner(HZ,exp(%i*2*%pi*f')));
HBPF=abs(horner(HBPS,%i*2*%pi*f'));
a=gca();
plot2d(f,HZF);
plot2d(f,HBPF);
xlabel('Analog Frequency');
ylabel('magnitude');
xtitle('band pass filter designed by the bilinear transformation');
disp(W,'W=');
disp(n,'n=');
disp(Qs,'Qs=');
disp(HPS,'HPS=');
disp(HBPS,'HBPS=');
disp(HZ'HZ=');
```

Output:

```
W  =
   0.8377580    0.9424778    1.6755161    2.5132741
 n  =
   4.
 Qs  =
Real part
   0.1048873 + s
Imaginary part
  - 0.9579530
 Qs  =
Real part
                        2
  - 0.3535534 + 0.3581075s + s
Imaginary part
  - 0.2841920 - 1.3547501s
 Qs  =
Real part
                                   2   3
   0.0232397 + 0.2746876s + 0.6113277s + s
Imaginary part
```

```
                                      2
 - 0.2122521 - 0.4851461s - 0.9579530s
 Qs  =
Real part
                                      2            3   4
   0.2057651 + 0.5167981s + 1.2564819s + 0.7162150s + s
Imaginary part
 - 3.123D-17 - 5.551D-17s
 HPS  =
                        0.1634
  ---------------------------------------------------------------
                                            2             3     4
  ( 0.2057651) + ( 0.5167981)s +  1.2564819s +  0.7162150s +  1s
 HBPS  =
                                   4
                         0.3410907s
  ----------------------------------------------------------------------
                                      2               3               4
  26.250497 +  9.983918s +  55.689893s + ( 15.263886)s + ( 39.388924)s
                 5             6             7     8
  + ( 6.7434281)s +  10.869451s +  0.8608904s +  1s
 HZ  =
                                     2            3           4            5
  5.4574518 + 21.829807z + 10.914904z - 65.489421z - 92.77668z + 43.659614z
              6            7           8            9            10
  + 152.80865z + 43.659614z - 92.77668z - 65.489421z + 10.914904z
              11           12
  + 21.829807z + 5.4574518z
  ---------------------------------------------------------------------------
      ( 1362.8151+i*3.788D-15) + ( 2450.4373+i*1.247D-14)z
                                 2                           3
      + ( 4082.8748+i*6.523D-15)z + ( 8810.0921-i*1.484D-14)z
                                 4                           5
      + ( 10076.664-i*1.398D-14)z + ( 12740.047-i*9.490D-15)z
                                 6                           7
      + ( 16444.996-i*2.920D-14)z + ( 13443.387-i*7.197D-15)z
                                 8                           9
      + ( 13331.887+i*4.944D-14)z + ( 11579.546+i*3.987D-14)z
                                 10                        11
      + ( 6162.8412-i*1.069D-14)z + ( 4737.51-i*2.082D-14)z
                                 12
      + ( 2298.9403-i*5.874D-15)z
```



Result:

The band-pass filter was designed using the indirect bilinear transformation method with a calculated filter order N that satisfies the 2 dB passband ripple and 20 dB stopband attenuation. The pre-warped frequencies W and transfer function HPS were determined, providing the desired frequency response for the specified passband and stopband frequencies.

(iv) Design an analog band-stop Chebyshev filter with passband edge frequencies omegap1 = 900π rad/sec, omegap2 = 1100π rad/sec, and a stopband edge frequency omegas = 1000π rad/sec. Given passband ripple of -1 dB and stopband attenuation of -40 dB, calculate the filter order N, cutoff frequency omegac1, and the poles of the band-reject Chebyshev filter.

Aim

To design an analog band-stop Chebyshev filter with specified passband edge frequencies, stopband frequency, passband ripple, and stopband attenuation, and to calculate the filter order, cutoff frequency, and poles for the filter.

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes, type the program and save the program in current directory
Compile and run the program
If any error occur in the program, correct the error and run the program
For the output, see the console window
Stop the program

Theory

Chebyshev filters are designed to minimize ripple in the passband while achieving sharp transitions between passband and stopband. The filter order N determines the steepness of the transition. Pre-warping is used to adjust analog frequencies for digital filter design. The epsilon (ε) parameter is computed based on the passband ripple, and the filter order is calculated based on the given specifications. The poles of the Chebyshev filter are determined to shape the frequency response, ensuring optimal band rejection at the stopband.

Program

//To Design an Analog Band Stop Chebyshev Filter

//For the given cutoff frequency = 500 Hz

```
// Define constants and parameters
omegap1 = 900*%pi; // First analog passband edge frequency in radians/sec
omegap2 = 1100*%pi; // Second analog passband edge frequency in radians/sec
omegas = 1000*%pi; // Analog stop band edge frequency in radians/sec
delta1_in_dB = -1;
delta2_in_dB = -40;
delta1 = 10^(delta1_in_dB/20);
delta2 = 10^(delta2_in_dB/20);
delta = sqrt(((1/delta2)^2) - 1);
epsilon = sqrt(((1/delta1)^2) - 1);

// Calculation of Filter order
num = ((sqrt(1 - delta2^2)) + (sqrt(1 - ((delta2^2) * (1 + epsilon^2))))) / (epsilon * delta2);
den = (omegas / omegap1) + sqrt((omegas / omegap1)^2 - 1);
```

```
N = log10(num) / log10(den);
N = floor(N);
// Cutoff frequency
omegac1 = (omegap1 + omegap2) / 2;
// Calculation of poles and zeros
[pols, Gn] = zpch1(N, epsilon, omegap1);

// Display results
disp(N, 'Filter order N =');
disp(pols, 'Poles of a type I band-reject Chebyshev filter are Sk =');
```

Output:

```
12.

 "Filter order N ="

      column 1 to 4
 -44.020402 + 2823.1154i  -129.06129 + 2630.7248i  -205.30686 + 2259.0546i  -267.56111 +
1733.4335i
      column 5 to 8
 -311.58151 + 1089.6819i  -334.36815 + 371.6702i  -334.36815 - 371.6702i  -311.58151 -
1089.6819i
      column 9 to 12
 -267.56111 - 1733.4335i  -205.30686 - 2259.0546i  -129.06129 - 2630.7248i  -44.020402 -
2823.1154i

 "Poles of a type I band-reject Chebyshev filter are Sk ="
```

Result:

The analog band-stop Chebyshev filter was designed with a calculated filter order N that meets the -1 dB passband ripple and -40 dB stopband attenuation. The cutoff frequencies ωc1 and poles of the filter were determined, ensuring proper attenuation around the stopband frequency of 1000π radians/second.

EX NO: 10	**Design of FIR filter**
DATE:	

(i) Design an FIR filter and plot the magnitude response for a low pass filter length N = 7, cutoff frequency (f_c = 1000 Hz) and sampling frequency F = 5000 Hz using the rectangular method in Scilab.

Aim

To design an FIR filter and plot the magnitude response for the low-pass filter (LPF) with a filter length N = 7, cutoff frequency (fc = 1000 Hz) and sampling frequency
F = 5000 Hz using the rectangular method in Scilab.

Software Required

Scilab 6.1.0

Procedure:

Start the Scilab Program
Open Scinotes, type the program and save the program in current directory
Compile and run the program
If any error occurs in the program, correct the error and run the program
For the output, see the console window
Stop the program

Theory:

An FIR (Finite Impulse Response) filter is a type of digital filter characterized by a finite duration of its impulse response. This means that the output of the filter settles to zero after a certain number of samples. FIR filters are widely used in digital signal processing due to their stability and ease of design.

Using the ideal low-pass filter formula for an FIR filter of length N and a rectangular window, the impulse response $h[n]$ can be computed as:

$$h[n] = \frac{\sin(2\pi f_c'(n-(N-1)/2))}{\pi(n-(N-1)/2)}$$

for $n = 0, 1, 2, ..., N-1$. The value of $h[n]$ at $n = (N-1)/2$ is given by:

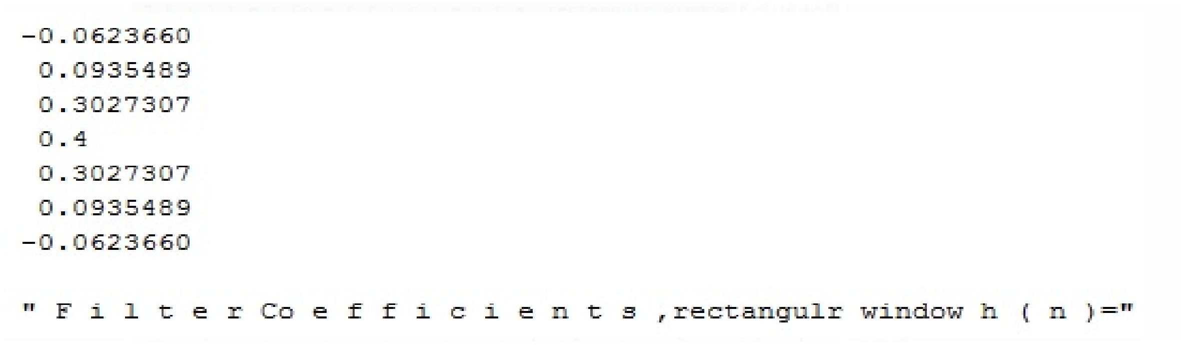
$$h\left[\frac{N-1}{2}\right] = 2f_c'$$

Program:

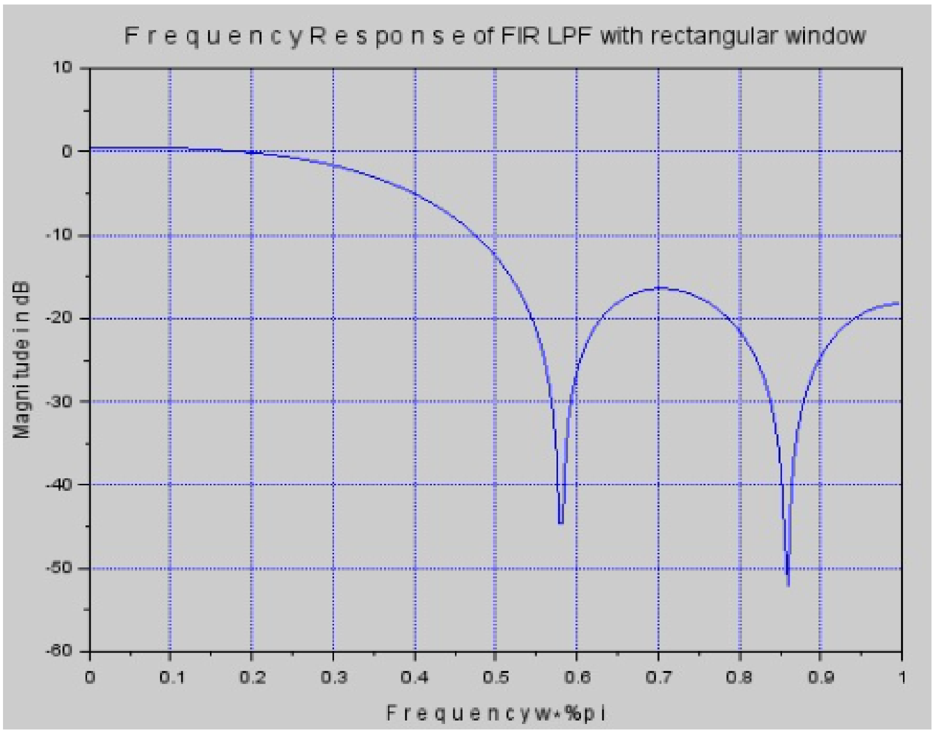
```
clear;
clc;
close;
N=7;
U=4;
h_Rect=window('re',N);
for n= -3+ U :1:3+ U
if n ==4
hd ( n) =0.4;
else
hd ( n) =( sin (2* %pi *( n - U ) /5) ) /( %pi *( n - U ) ) ;
```

```
end
h ( n ) = hd ( n ) * h_Rect ( n ) ;
 end
[ hzm , fr ]= frmag (h ,256) ;
 hzm_dB =20* log10 ( hzm ) ./ max ( hzm ) ;
figure
 xgrid (2) ;
plot (2* fr , hzm_dB )
a = gca () ;
 xlabel ('F r e q u e n c y w*%p i') ;
 ylabel ( ' Magni tude i n dB ' ) ;
 title ( ' F r e q u e n c y R e s po n s e of FIR LPF wi t h N=7 ' ) ;
xgrid (2)
 disp (h ," F i l t e r Co e f f i c i e n t s , h ( n )=") ;
```

Output:



```
-0.0623660
 0.0935489
 0.3027307
 0.4
 0.3027307
 0.0935489
-0.0623660

" F i l t e r Co e f f i c i e n t s ,rectangulr window h ( n )="
```



Result:

The magnitude response of a FIR low-pass filter (LPF) is implemented using Scilab. The cutoff frequency is normalized with respect to the sampling frequency. The sinc function is used to calculate the ideal filter coefficients, which are then modified by the Rectangular Window. Finally, the magnitude response is plotted to visualize the filter's performance.

(ii) Write a SciLab program that designs a high-pass FIR filter using a Hamming window. The filter should have a length of *N*=11 and the cutoff frequency should be determined based on the sinc function. The program should calculate the ideal impulse response for a high-pass filter, apply the Hamming window to it, and then plot the magnitude response in decibels (dB). Additionally, display the filter coefficients in the console.

Aim

To design an FIR filter and plot the magnitude response for the high-pass filter (LPF) with a filter length N = 11 using the hamming window method in Scilab.

Software Required

Scilab 6.1.0

Procedure:

Start the Scilab Program
Open Scinotes, type the program and save the program in current directory
Compile and run the program
If any error occurs in the program, correct the error and run the program
For the output, see the console window
Stop the program

Theory:

The Hamming window is applied to smooth the transition band and reduce ripples in the frequency response

Calculate the Ideal Impulse Response: The ideal impulse response for a high-pass filter is given by:

$$h[n] = -\frac{\sin(2\pi f_c'(n-(N-1)/2))}{\pi(n-(N-1)/2)}$$

for $n = 0, 1, 2, \ldots, N-1$. The value of $h[n]$ at $n = (N-1)/2$ is:

$$h\left[\frac{N-1}{2}\right] = 1 - 2f_c'$$

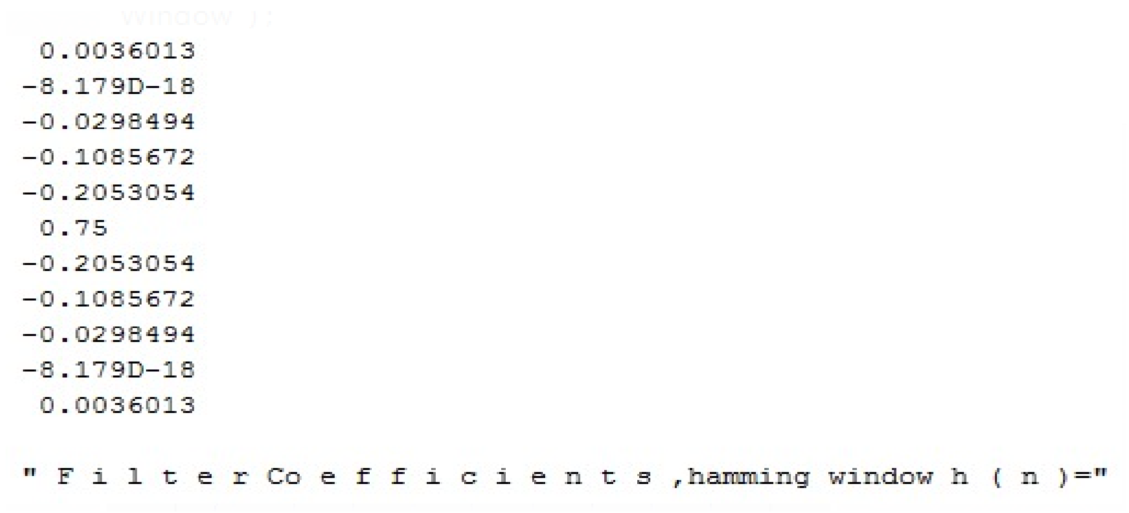
This equation flips the low-pass response into a high-pass response.

Program:

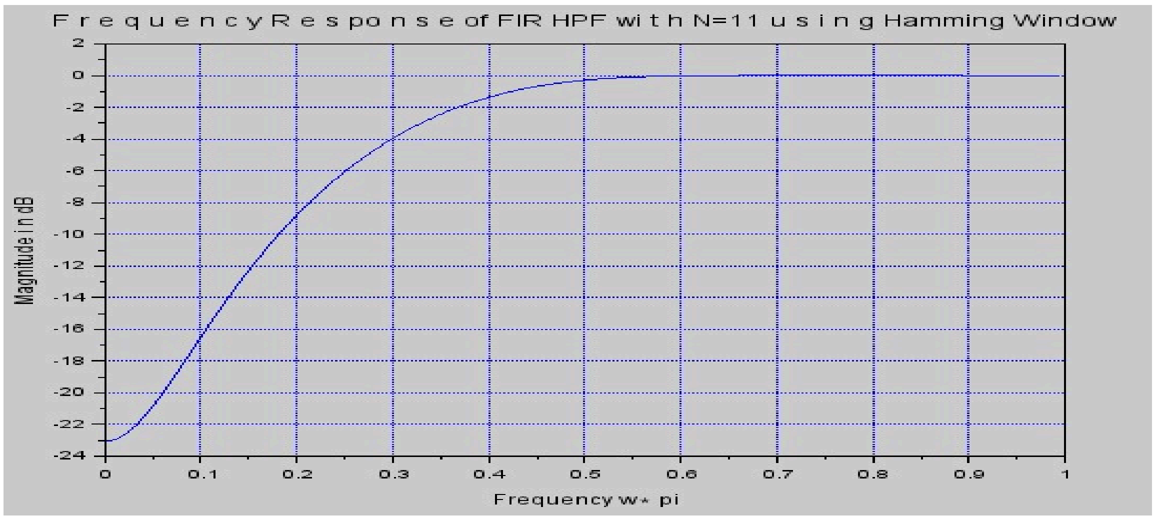
```
clear ;
clc ;
close ;
N =11;
U =6;
h_hamm =window('hm',N);
for n= -5+ U :1:5+ U
if n ==6
hd( n) =0.75;
else
hd ( n) =( sin ( %pi *( n - U ) ) -sin( %pi *( n - U ) /4) ) /( %pi *( n -U ) ) ;
end
```

```
h ( n ) = h_hamm ( n ) * hd ( n ) ;
end
[ hzm , fr ]= frmag (h ,256) ;
 hzm_dB =20* log10 ( hzm ) ./ max ( hzm ) ;
figure
plot (2* fr , hzm_dB )
a=gca () ;
 xlabel ( 'Frequency w∗ pi');
 ylabel ('Magnitude i n dB') ;
 title ( ' F r e q u e n c y R e s po n s e of FIR HPF wi t h N=11 u s i n g Hamming Window' ) ;
 xgrid (2) ;
disp (h ," F i l t e r Co e f f i c i e n t s , h ( n )=") ;
```

Output:



```
        window );
  0.0036013
 -8.179D-18
 -0.0298494
 -0.1085672
 -0.2053054
  0.75
 -0.2053054
 -0.1085672
 -0.0298494
 -8.179D-18
  0.0036013

 " F i l t e r Co e f f i c i e n t s ,hamming window h ( n )="
```



Result:

The magnitude response of a FIR high-pass filter (LPF) is implemented using Scilab. The cutoff frequency and sampling frequency are selected accordingly. The sinc function is used to compute the ideal low-pass filter coefficients, which are then converted to high-pass filter coefficients. The Hamming window is applied to the ideal filter coefficients. Finally, the magnitude response of the FIR filter is plotted.

(iii) Write a SciLab program that designs a high-pass FIR filter using a Hamming window. Find the magnitude response plot and note the value at the normalized frequency *w*=*π*, which corresponds to the maximum frequency on the x-axis. Plot the filter coefficients h(n) generated by the program

Aim

To design an FIR filter and plot the magnitude response for the high-pass filter (LPF) with a filter length N = 11 using the hanning window method in Scilab. And also to plot the filter coefficients h(n) generated by the program.

Software Required

Scilab 6.1.0

Procedure:

Start the Scilab Program
Open Scinotes, type the program and save the program in current directory
Compile and run the program
If any error occurs in the program, correct the error and run the program
For the output, see the console window
Stop the program

Theory:

The ideal impulse response for a high-pass filter is derived from the low-pass filter by subtracting it from a delta function. For a high-pass filter:

$$h[n] = \delta[n-(N-1)/2] - \frac{\sin(2\pi f_c'(n-(N-1)/2))}{\pi(n-(N-1)/2)}$$

where $\delta[n-(N-1)/2]$ is a delta function centered at $(N-1)/2$.

The Hamming window is used to smooth the transition band and reduce the side lobes in the frequency response. The window function is:

$$w[n] = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right)$$

The final filter coefficients are obtained by multiplying the ideal impulse response with the Hamming window:

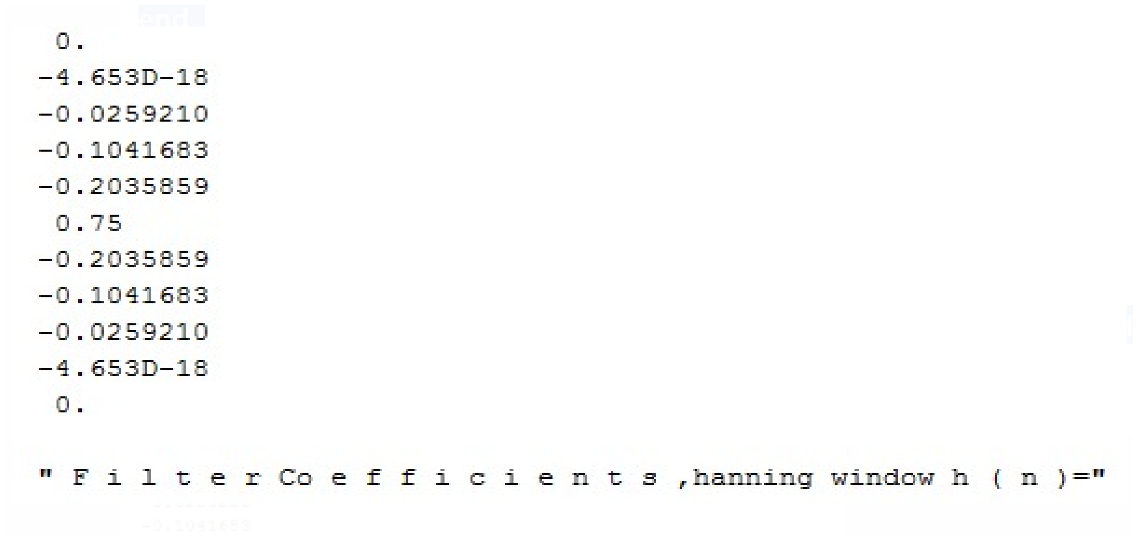
$$h_{final}[n] = h[n] \times w[n]$$

Program:

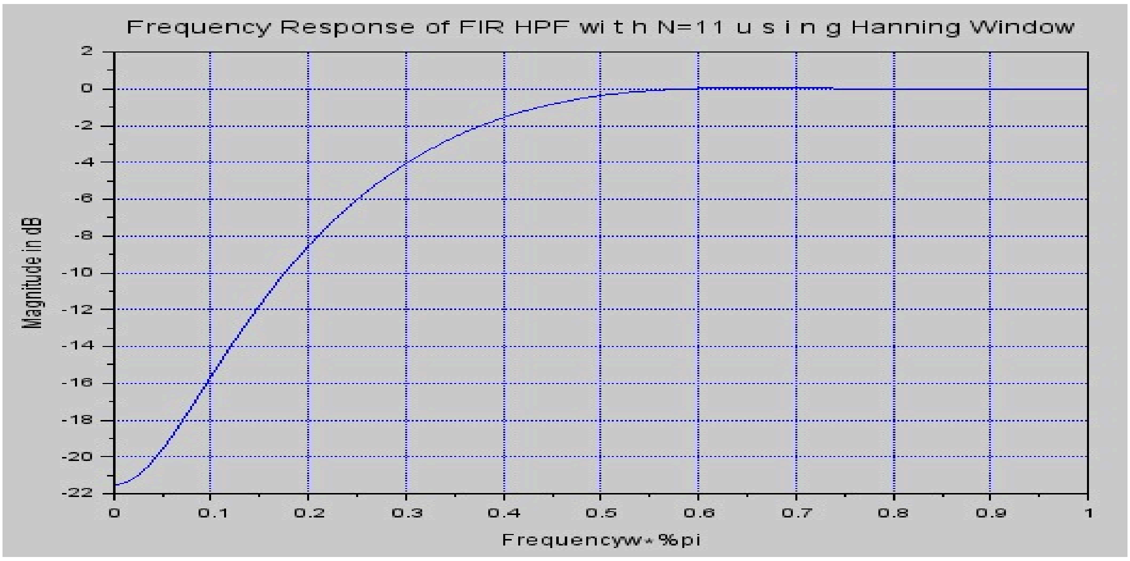
```
clear ;
clc ;
close ;
N =11;
U =6;
h_hann=window('hn',N ) ;
for n = -5+ U :1:5+ U
if n == 6
hd ( n) = 0.75;
else
hd ( n) =( sin ( %pi *( n - U ) ) -sin( %pi *( n - U ) /4) ) /( %pi *(n -U ) ) ;
```

```
end
h ( n ) = h_hann ( n ) * hd ( n ) ;
end
[ hzm , fr ]= frmag (h ,256) ;
 hzm_dB =20* log10 ( hzm ) ./ max( hzm ) ;
figure
plot (2* fr , hzm_dB )
a=gca () ;
 xlabel ('Frequencyw∗%pi') ;
 ylabel ('Magnitude in dB') ;
 title ('Frequency Response of FIR HPF wi t h N=11 u s i n g Hanning Window ' ) ;
 xgrid (2) ;
disp (h ," F i l t e r Co e f f i c i e n t s ,hanning window h ( n )=") ;
```

Output:



```
  0.
-4.653D-18
-0.0259210
-0.1041683
-0.2035859
  0.75
-0.2035859
-0.1041683
-0.0259210
-4.653D-18
  0.

" F i l t e r Co e f f i c i e n t s ,hanning window h ( n )="
```



Result:

The magnitude response of a FIR high-pass filter (LPF) is implemented using Scilab. The cutoff frequency and sampling frequency are selected accordingly. The ideal high-pass filter is obtained by subtracting the low-pass filter from a delta function. The Hanning window is applied to the high-pass filter coefficients. The magnitude response is plotted using the freq. function.

(iv)Write a SciLab program that designs a high-pass FIR filter using a Blackman window with a filter length *N*=11. The program should compute the filter's impulse response using the Blackman window, plots the magnitude response in decibels (dB), and displays the filter coefficients in the console.

Aim

To write a SciLab program that designs a high-pass FIR filter using a Blackman window with a filter length *N*=11. The program should also be able to compute the filter's impulse response using the Blackman window, plots the magnitude response in decibels (dB), and displays the filter coefficients in the console

Software Required

Scilab 6.1.0

Procedure:

Start the Scilab Program
Open Scinotes, type the program and save the program in current directory
Compile and run the program
If any error occurs in the program, correct the error and run the program
For the output, see the console window
Stop the program

Theory:

Compute the Ideal Impulse Response: The ideal impulse response for a high-pass filter is:

$$h[n] = \delta[n-(N-1)/2] - \frac{\sin(2\pi f_c'(n-(N-1)/2))}{\pi(n-(N-1)/2)}$$

where $\delta[n-(N-1)/2]$ is a delta function centered at $(N-1)/2$.

Apply the Blackman Window: The Blackman window is defined as:

$$w[n] = 0.42 - 0.5\cdot\cos\left(\frac{2\pi n}{N-1}\right) + 0.08\cdot\cos\left(\frac{4\pi n}{N-1}\right)$$

The final filter coefficients are obtained by multiplying the ideal impulse response with the Blackman window:

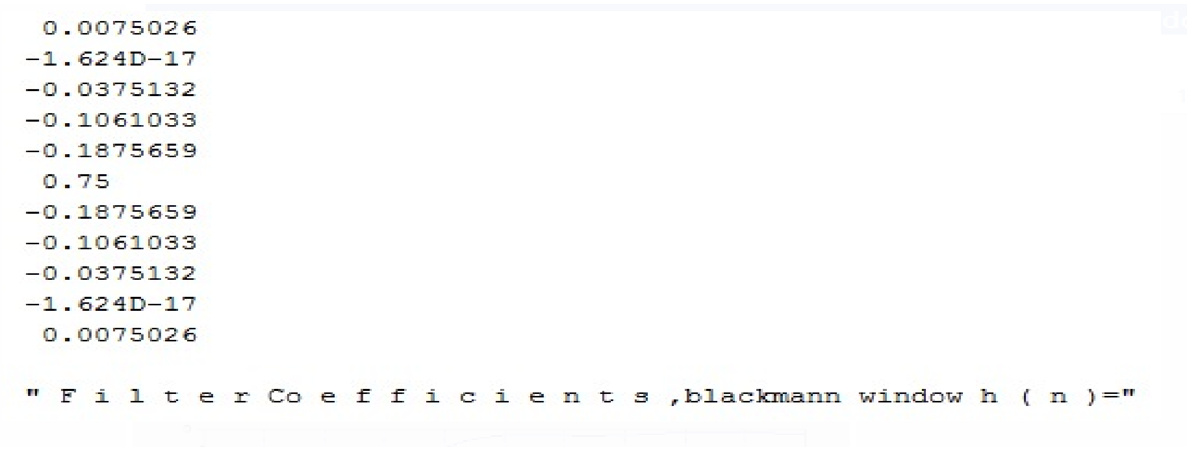
$$h_{final}[n] = h[n]\times w[n]$$

Program:

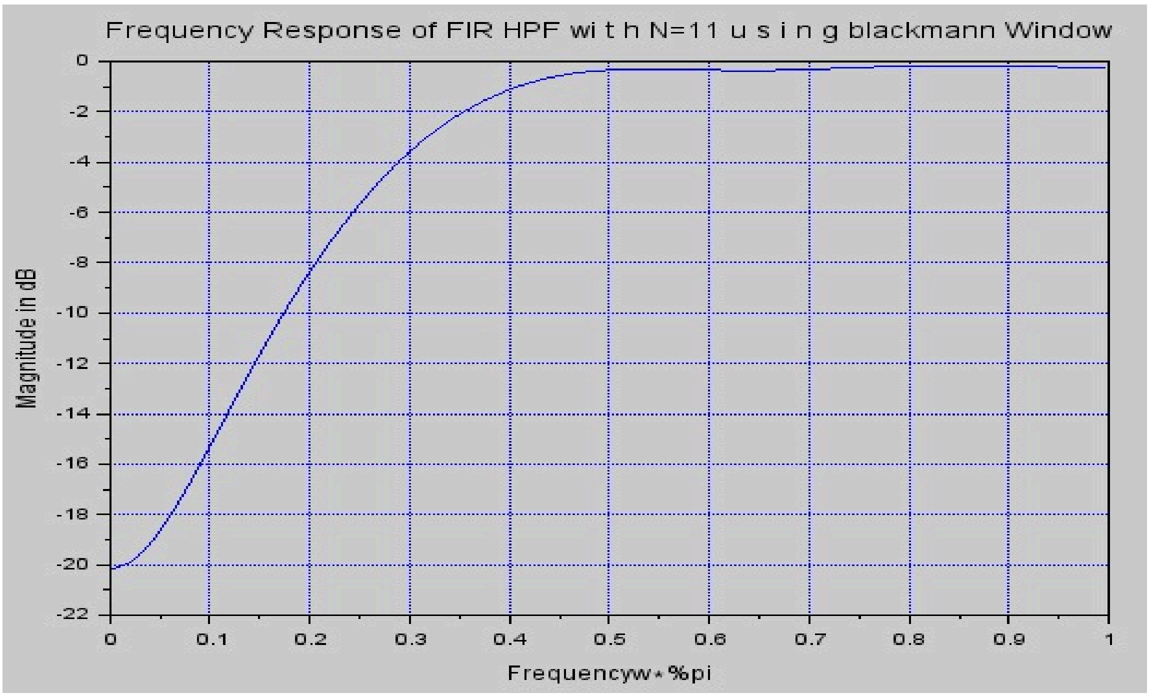
```
clear ;
clc ;
close ;
N =11;
U =6;
h_blackmann=window('tr',N ) ;
for n = -5+ U :1:5+ U
if n == 6
hd ( n) = 0.75;
else
hd ( n) =( sin ( %pi *( n - U ) ) -sin( %pi *( n - U ) /4) ) /( %pi *(n -U ) ) ;
end
h ( n ) = h_blackmann( n ) * hd ( n ) ;
```

```
end
[ hzm , fr ]= frmag (h ,256) ;
 hzm_dB =20* log10 ( hzm ) ./ max( hzm ) ;
figure
plot (2* fr , hzm_dB )
a=gca () ;
 xlabel ('Frequencyw∗%pi') ;
 ylabel ('Magnitude in dB') ;
 title ('Frequency Response of FIR HPF wi t h N=11 u s i n g blackman    Window ' ) ;
 xgrid (2) ;
disp (h ," F i l t e r Co e f f i c i e n t s ,blackmann window h ( n )=") ;
```

Output:



```
 0.0075026
-1.624D-17
-0.0375132
-0.1061033
-0.1875659
 0.75
-0.1875659
-0.1061033
-0.0375132
-1.624D-17
 0.0075026

" F i l t e r Co e f f i c i e n t s ,blackmann window h ( n )="
```



Result:

The magnitude response of a FIR high-pass filter (LPF) is implemented using Scilab. The cutoff frequency and sampling frequency are selected accordingly. The ideal high-pass filter is obtained by subtracting the low-pass filter from a delta function. The Blackman window is applied to the high-pass filter coefficients. The magnitude response is plotted using the frequency function.

EX NO: 11	Design of Band Pass filter using Fourier series Method
DATE:	

Design an ideal band-pass filter with a passband frequency of 2000 Hz and a sampling frequency of 9600 Hz using the Fourier series method. Compute the filter coefficients and verify the performance

Aim

To design an ideal band-pass filter with a passband frequency of 2000 Hz and a sampling frequency of 9600 Hz using the Fourier series method in SCILAB. The experiment aims to compute the filter coefficients.

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

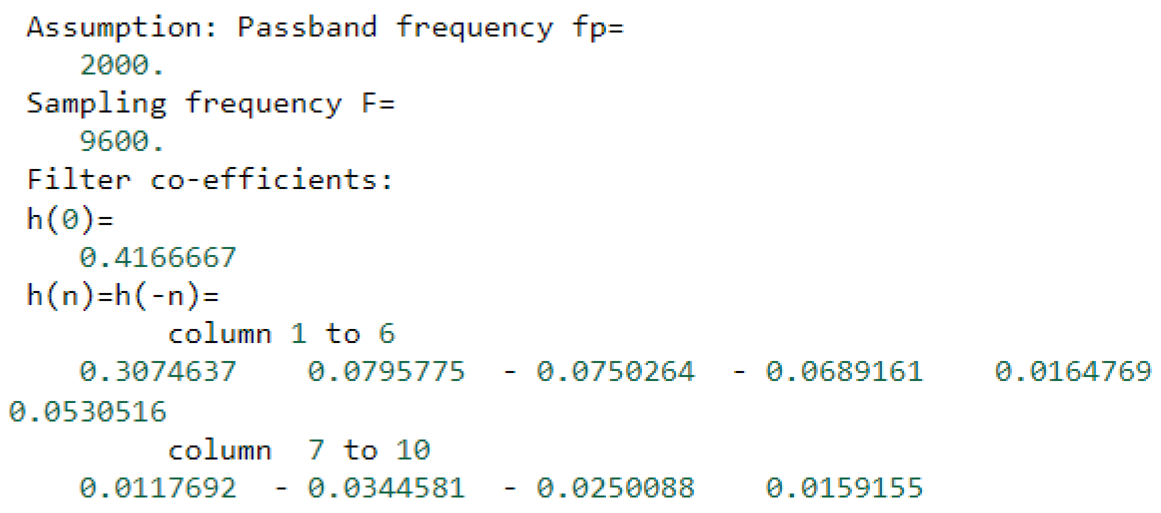
Theory:

The Finite Impulse Response (FIR) filter is a type of digital filter characterized by a finite number of non-zero terms in its impulse response. One common method to design an FIR filter is by using the Fourier series method. This approach is based on approximating the desired frequency response of the filter by truncating the Fourier series representation of an ideal filter.
The Fourier series method leverages the fact that any periodic function can be represented as a sum of sinusoidal components, each corresponding to a different frequency. The goal of designing an FIR filter using the Fourier series method is to approximate the desired frequency response by computing the Fourier coefficients.

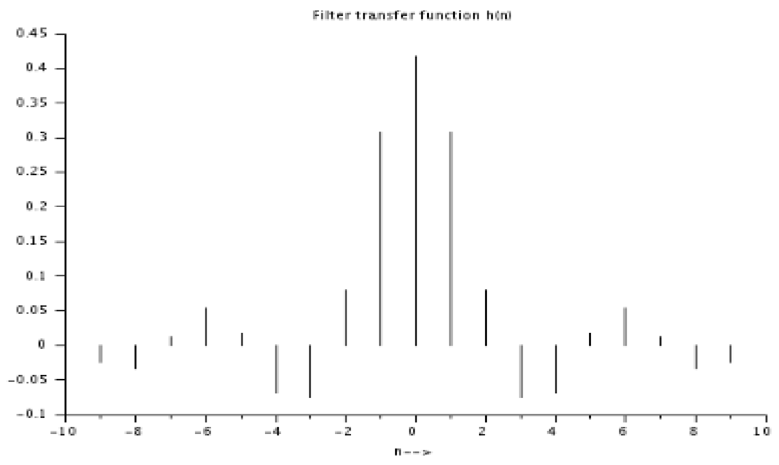
Program:

```
clc;clear;close;

    fp=2000;        //passband frequency
    F=9600;         //sampling frequancy
    //Calculation of filter co-efficients
    a0=1/F*integrate('1','t',-fp,fp);
    for n=1:10;
    a(1,n)=2/F*integrate('cos(2*%pi*n*f/F)','f',-fp,fp);
    end
    h=[a(:,$:-1:1)/2 a0 a/2];
    //Displaying filter co-efficients
```

```
disp(F,'Sampling frequency F= ',fp,'Assumption: Passband frequency fp= ');
disp('Filter co-efficients:');
disp(a0,'h(0)= ');disp(a/2,'h(n)=h(-n)=');
n=-10:10;
plot2d3(n,h);
title('Filter transfer function h(n)');xlabel('n-->');
```



```
 Assumption: Passband frequency fp=
    2000.
 Sampling frequency F=
    9600.
 Filter co-efficients:
 h(0)=
    0.4166667
 h(n)=h(-n)=
         column 1 to 6
    0.3074637    0.0795775  - 0.0750264  - 0.0689161    0.0164769
0.0530516
         column  7 to 10
    0.0117692  - 0.0344581  - 0.0250088    0.0159155
```



Result:

The ideal band-pass filter using the Fourier series method in SCILAB with a passband frequency of 2000 Hz and a sampling frequency of 9600 Hz, the filter coefficients were successfully computed.

EXNO: 12	Multirate Signal Processing
DATE:	

(i)Analyze the effects of multirate signal processing techniques on sinusoidal signals through the processes of downsampling (decimation) for sampling rate of 1 KHz.

Aim:

To design and implement downsampling of a sinusoidal signal using Multirate Signal Processing techniques, and to observe the effects of both decimation and interpolation on the signal.

Software Required:

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

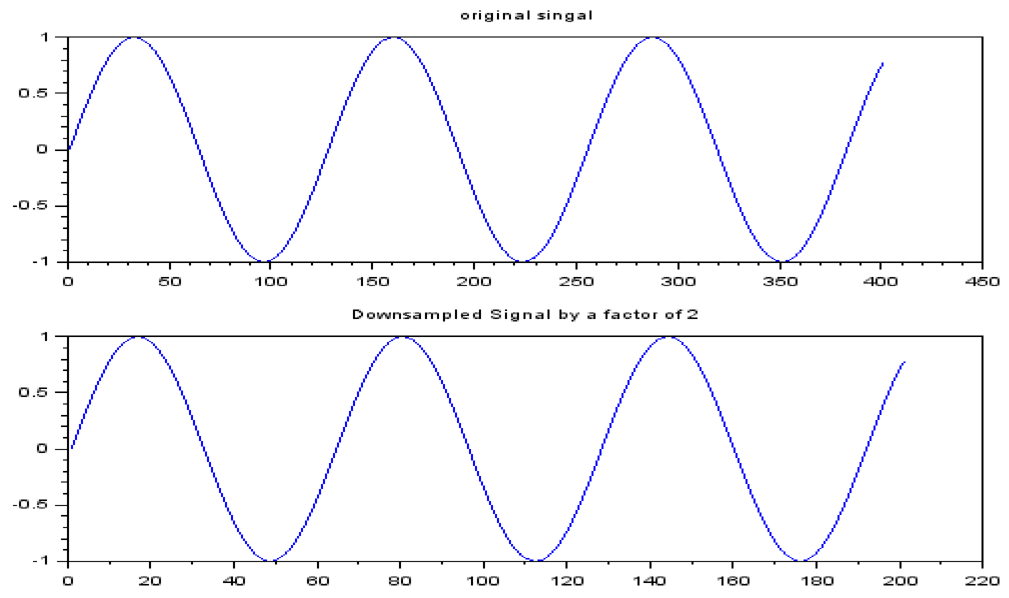
Downsampling reduces the sampling rate of a signal by an integer factor, usually denoted as M. When downsampling a signal, every Mth sample is retained, and the remaining samples are discarded. The goal is to reduce the data rate while maintaining the essential information in the signal.For a sinusoidal signal $x(n) = A * \sin(2\pi f_0 n)$, downsampling by a factor of M reduces the number of samples by a factor of M. However, before decimation, the signal should be filtered using a low-pass filter to prevent aliasing, which can occur if the signal contains frequency components higher than half of the new (lower) sampling rate.

Scilab Code:

```
//Multirate Signal Processing in scilab
   //Downsampling a sinusoidal signal by a factor of 2
   clear;
   clc;
   n = 0:%pi/200:2*%pi;
   x = sin(%pi*n);   //original signal
   downsampling_x = x(1:2:length(x)); //downsampled by a factor of 2
```

```
subplot(2,1,1)
plot(1:length(x),x);
xtitle('original singal')
subplot(2,1,2)
plot(1:length(downsampling_x),downsampling_x);
xtitle('Downsampled Signal by a factor of 2');
```

Output:



Result:

1. Original Signal: The sinusoidal signal at the original sampling rate shows a smooth wave.2. Downsampled Signal: The downsampled signal has fewer data points, leading to a loss of resolution. A larger downsampling factor may cause visible distortion.

(ii) To design and implement downsampling of a sinusoidal signal using Multirate Signal Processing techniques, and to observe the effects of both decimation and interpolation on the signal.

Aim :

Write a program to compute Up sampling a sinusoidal signal by a factor of 2 using scilab

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

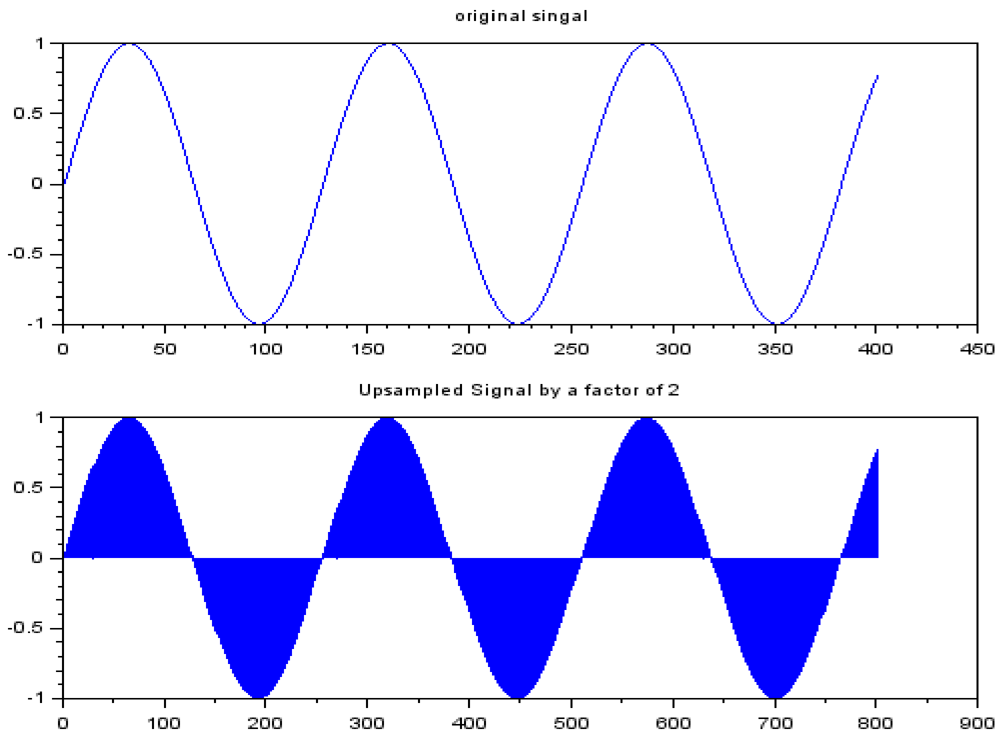
Theory:

Upsampling is the process of increasing the sampling rate of a discrete signal by an integer factor, usually denoted as L. In this case, upsampling is performed by a factor of 2, meaning that each sample in the original signal is followed by an additional interpolated sample, effectively doubling the number of samples in the signal.When a signal is upsampled, the new signal is constructed by inserting (L-1) zeros between every sample of the original signal. After this insertion, the signal needs to be passed through a low-pass filter (interpolation filter) to smooth out the zero-inserted samples and prevent aliasing.

Program:

```
//Multirate Signal Processing in scilab
//Upsampling a sinusoidal signal by a factor of 2
clear;
clc;
n = 0:%pi/200:2*%pi;
x = sin(%pi*n);   //original signal
upsampling_x = zeros(1,2*length(x)); //upsampled by a factor of 2
upsampling_x(1:2:2*length(x)) = x;
subplot(2,1,1)
plot(1:length(x),x);
xtitle('original singal')
subplot(2,1,2)
plot(1:length(upsampling_x),upsampling_x);
xtitle('Upsampled Signal by a factor of 2');
```

Output:



Result:

Original Signal: The sinusoidal signal at the original sampling rate shows a smooth wave.2. Upsampled Signal: The Upsampled signal has fewer data points, leading to a gain of resolution.

EX NO:13	Generation of Waveforms using Digital Signal Processor

Aim:

To generate and analyze different waveforms such as square, triangular and Sawtooth using the TMS320C5416 Digital Signal Processor (DSP) and validate their characteristics.

Hardware and Software Required:

DSP – TMS320VC5416-5416 Board
Cpu
ME Universal Debugger DSP - 5416

Procedure:

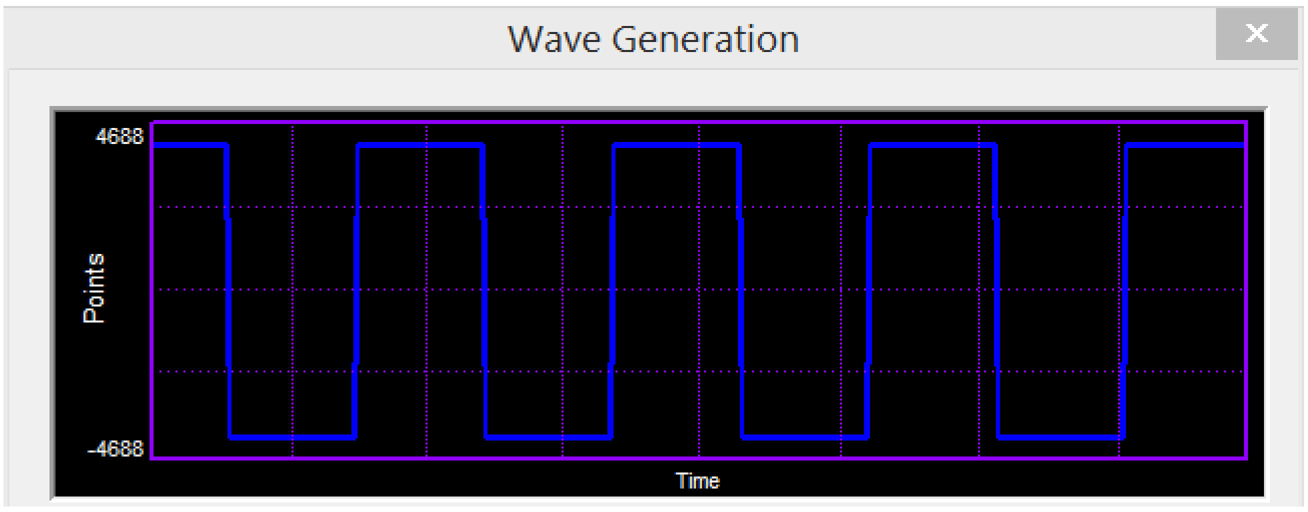
Connection is checked by the following procedure. After connecting the serial port cable, switch on the kit and reset the kit. Select the Menubar-Serial > Port Settings > Auto Detect
If connection (PC to kit) is improper, User can see the following window and try to connect it properly by the conditions until find the message Vi Universal debugger found in COMI
After checking the port connection, create new project for working with a program by using following procedure.

Generally user has to follow the path is C\Vi Universal Debugger VSK-5416/My Project
Before working with a program, User has to follow certain conditions,
1. Create a project and name it
2. Create a file and start to type a program, save it by project name
3. Add program file to the project
4. Add command file (Micro5416) to the project
5. Save project
6. Build it and
7.Download the program to the VSK-5416 kit

SQUARE WAVE GENERATION

```
DATA .SET 0H
       .mmregs
        .text
START: STM #140H,ST0                          ;initialize the data page pointer
            RSBX CPL                          ;make the processor to work using DP
            NOP
            NOP
            NOP
            NOP
REP: ST #0H,DATA                              ;send 0h to the dac
            CALL DELAY                                    ;delay for some time
             ST #0FFFH,DATA                    ;send 0fffh to the dac
             CALL DELAY                                    ;delay for some time
             B REP                            ;repeat the same
DELAY: STM #0FFFH,AR1
DEL1: PORTW DATA,04H
 BANZ DEL1,*AR1-
 RET
```

Output:

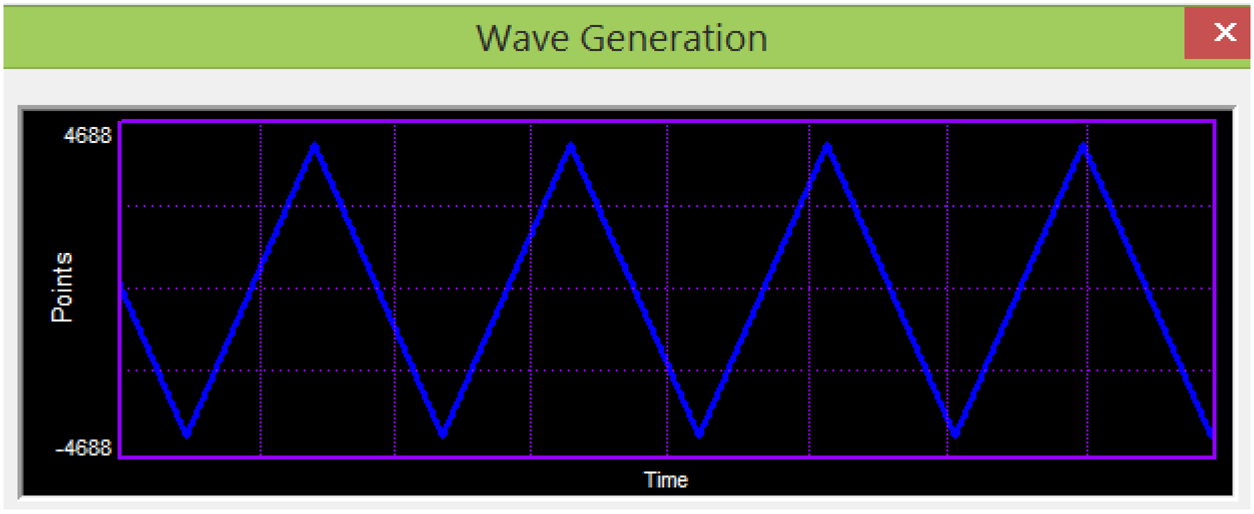


TRIANGULAR WAVE GENERATION:

```
DATA .SET 0H
```

```
      .mmregs
      .text
START: STM #140H,ST0              ;initialize the data page pointer
          RSBX CPL                ;make the processor to work using DP
          NOP
          NOP
          NOP
          NOP
 REP: ST #0H,DATA                  ;initialize the value as 0h
  INC: LD DATA,A                   ;increment the value
         ADD #1H,A
          STL A,DATA
          PORTW DATA,04H          ;send the value to the dac
          CMPM DATA,#0FFFH                 ;repeat the loop until the value becomes 0fffh
          BC INC,NTC
DEC: LD DATA,A                    ;decrement the value
         SUB #1H,A
         STL A,DATA
         PORTW DATA,04H           ;send the value to the dac
         CMPM DATA,#0H
         BC DEC,NTC               ;repeat the loop until the value becomes 0h
         B REP                    ;repeat the above
```

Output:

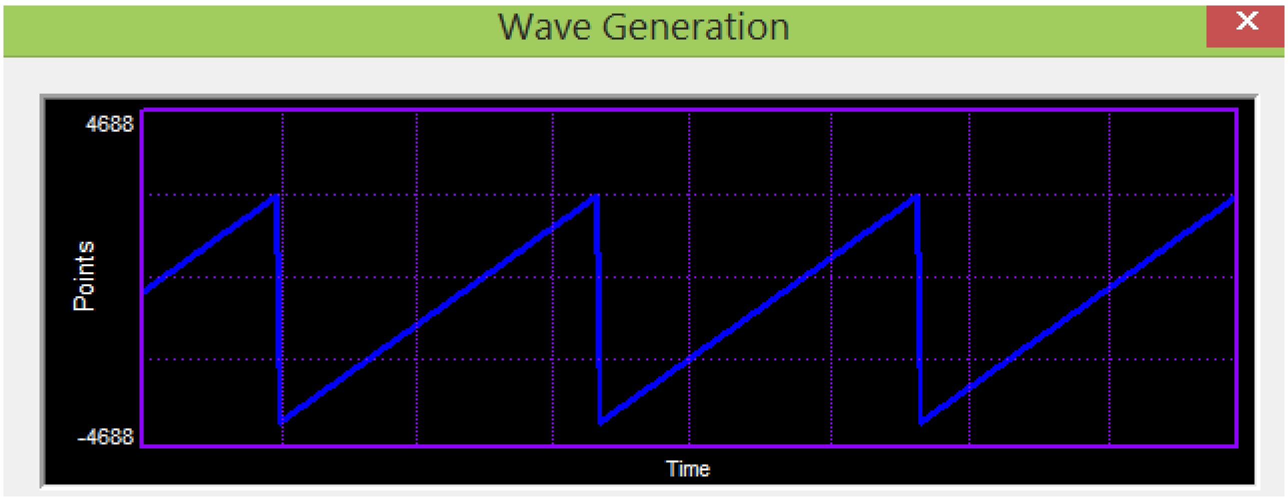


SAWTOOTH WAVE GENERATION:

```
 DATA .SET 0H
      .mmregs
      .text
```

```
START: STM #140H,ST0                    ;initialize the data page pointer
          RSBX CPL                      ;make the processor to work using DP
          NOP
          NOP
          NOP
          NOP
REP: ST #0H,DATA                        ;initialize the value as 0h
INC: LD DATA,A
ADD #1H,A                               ;increment the value
STL A,DATA
PORTW DATA,04H                          ;send the value to the dac
CMPM DATA,#0FFFH                        ;repeat the loop until the value becomes 0fffh
BC INC,NTC
 B REP                                  ;repeat the above
```

Output:



Result: square, triangular and sawtooth waveforms were successfully generated using the TMS320C5416 Digital Signal Processor. The characteristics of the generated waveforms were observed and validated using an oscilloscope.

EX NO:14	Design and Implementation of Butterworth IIR Filter

Aim:

To design and implement a 2nd-order Low Pass Butterworth IIR Filter with a cutoff frequency of 4 kHz and a sampling frequency of 62 kHz, and to analyze its frequency response to validate the filter's performance.

Hardware and Software Required:

DSP – TMS320VC5416-5416 Board
Cpu
ME Universal Debugger DSP - 5416

Procedure:

Connection is checked by the following procedure.
After connecting the serial port cable, switch on the kit and reset the kit.
Select the Menubar-Serial > Port Settings > Auto Detect
If connection (PC to kit) is improper, User can see the following window and try to connect it properly by the conditions until find the message Vi Universal debugger found in COMI
After checking the port connection, create new project for working with a program by using following procedure.

Generally user has to follow the path is C\Vi Universal Debugger VSK-5416/My Project
Before working with a program, User has to follow certain conditions,
1. Create a project and name it
2. Create a file and start to type a program, save it by project name
3. Add program file to the project
4. Add command file (Micro5416) to the project
5. Save project
6. Build it and
7.Download the program to the VSK-5416 kit

```
*************************************************************************
```

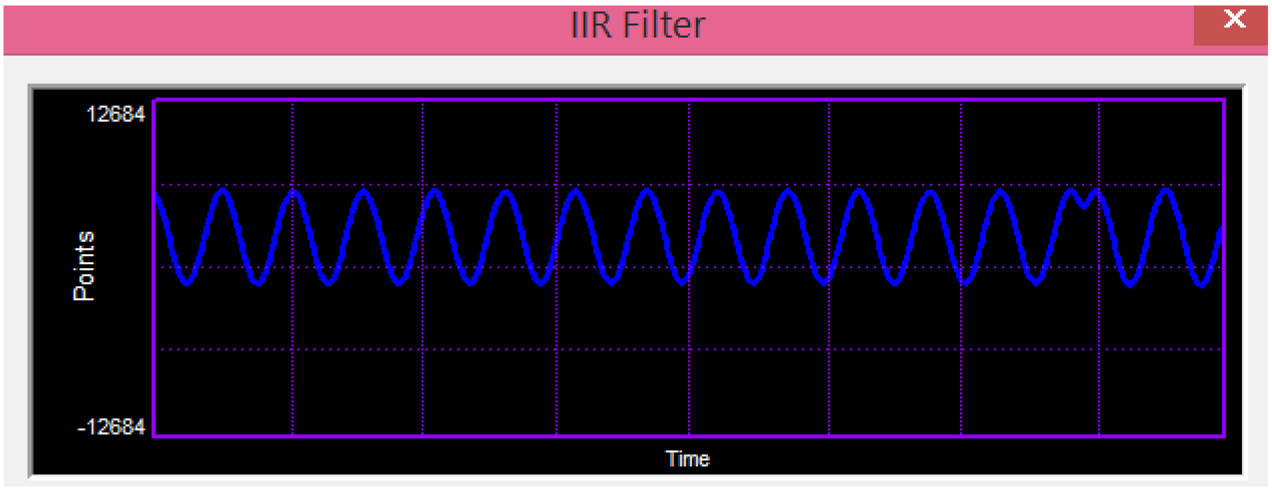
LOW PASS FILTER (IIR)

```
;*************************************************************************

*    butterworth filter
* IIR Filter, Created by Filter Design Package,
* (c) 2005, Vi Microsystems Pvt. Ltd.,
* Approximation Type: ButterWorth
* Filter Type: Low Pass Filter
* Filter Order: 2
* Sampling Frequency: 62
* Cutoff frequency in KHz = 4 .000000

;Variable declaration for the filter
XN          .SET    10H
XNM1        .SET    11H
XNM2        .SET    12H
YN          .SET    13H
YNM1        .SET    14H
YNM2        .SET    15H
SUM         .SET    16H
; Filter            assignme
coefficient         t
A1          .SET    0E903H
A2          .SET    0905H
B0          .SET    082H
B1          .SET    0104H
B2          .SET    082H
      .mmregs
      .tex
t
START:
        STM       #0140H,ST0       ; data page initialization to page
                                   140
        RSBX      CPL              ; reset the compiler mode bit
        RSBX      FRCT             ; reset the fractional mode bit
        NOP                        ; wait for the reset action
        NOP
        NOP
```

```
;initialize x(n),x(n-1),x(n-2),y(n),y(n-1),y(n-2)
; make all the variables to initially
        zero ST    #0H,XN
        ST         #0H,XNM1
        ST         #0H,XNM2
        ST         #0H,YN
        ST         #0H,YNM1
        ST         #0H,YNM2
REPEAT:

        PORTR      6,0               ; give the start of conversion for adc
        nop                              ; wait the some delay to conversion
        nop
        nop
        nop
        PORTR     4,0                ; read the sampled data from the port
        LD         0,A               ; load it the accumulator
        AND          #0FFFH,A        ; extract the 12 bit alone
        XOR         #0800H,A         ; to correct 2's complement
        SUB         #0800H,A         ; convert the data to processor format
        STL        A,XN              ; store the sampled data to XN
        LD         #0H,B             ; initialize the B reg to 0 for result
        LD         XN,A              ; get the XN to accumulator
        STLM       A,T               ; load it to the t reg
        MPY        #B0,A             ; multiply the b0 coeff to the accumulator
        LD         A,B               ; load the content of B reg to accumulator
        LD         XNM1,A        ; get the XNM1 to accumulator
        STLM       A,T               ; load it to T reg
        MPY        #B1,A             ; multiply the b0 coeff to the accumulator
        ADD        A,B               ; sum the current output with the B reg
```

```
LD          XNM2,A          ; get the XNM2 to accumulator
STLM        A,T                 ; store the content of A reg to T reg
STLM        A,T                 ; multiply the b2 coeff to the accumulator
MPY         #B2,A               ; sumup the current output with the B reg
ADD         A,B
LD          YNM1,A          ; get the YNM1 to accumulator
STLM        A,T                 ; store the content of A reg to T reg
MPY         #A1,A               ; multiply the A1 coeff to the accumulator
SUB         A,B               ; subtract the current output from the B reg
LD          YNM2,A          ; get the YNM2 to accumulator
STLM        A,T                 ; store the content of A reg to T reg
MPY         #A2,A               ; multiply the A1 coeff to the accumulator
SUB         A,B               ; subtract the current output from the B reg
SFTL        B,-12               ; the resultant value in the B reg is shifted
STL         B,YN                ; right by 12 bits store the final output to
LD          YNM1,A               ; the variable YN exchange YNM1 to YNM2
STL         A,YNM2
LD          YN,A                ; exchange YN to YNM1
STL         A,YNM1
LD          XNM1,A              ; exchange XNM1 to XNM2
STL         A,XNM2
LD          XN,A                ; exchange XN to XNM1
STL         A,XNM1
LD          YN,A                ; load the final output data to accumulator
ADD         #800H,A             ; add the offset value
STL         A,YN                ; store the result in YN
PORTW      YN,04H               ; out the output to the DAC
B           REPEAT              ; branch to continue
```

Output:



Result: A 2nd-order Low Pass Butterworth IIR Filter with a cutoff frequency of 4 kHz and a sampling frequency of 62 kHz was successfully designed and implemented.

EX NO:15	**Design and Implementation of FIR Filter using Rectangular Window**

Aim: To design and implement a High Pass Butterworth IIR Filter with a sampling frequency of 43 kHz, a cutoff frequency of 2 kHz, and filter order N=2N = 2N=2, and to analyze its frequency response to validate its performance.

Hardware and Software Required:

DSP – TMS320VC5416-5416 Board
Cpu
ME Universal Debugger DSP - 5416

Procedure:

Connection is checked by the following procedure. After connecting the serial port cable, switch on the kit and reset the kit. Select the Menubar-Serial > Port Settings > Auto Detect
If connection (PC to kit) is improper, User can see the following window and try to connect it properly by the conditions until find the message Vi Universal debugger found in COMI
After checking the port connection, create new project for working with a program by using following procedure.

Generally user has to follow the path is C\Vi Universal Debugger VSK-5416/My Project
Before working with a program, User has to follow certain conditions,
1. Create a project and name it
2. Create a file and start to type a program, save it by project name
3. Add program file to the project
4. Add command file (Micro5416) to the project
5. Save project
6. Build it and
7.Download the program to the VSK-5416 kit

```
;**************************************************************************

                         HIGH PASS FIR FILTER

;**************************************************************************

;Sampling freq      : 43khz

;Cut-off freq       : 2khz
```

```
;N                    52

;Filter type         : High pass filter

;Window type         : Rectangular

;Program Description:

;1. Make all the x(n) zero initially

;2. Read the data from the adc.

;3. Store the adc data in x(0)

;4. Make the pointer to point the x(n_end)

;5. Perform the convolution of x(n) and the coefficients h(n) using

; MACD instruction.

;6. Send the convolution output to the dac

;7. Repeat from step 2.

        .mmregs

        .tex
t START:
          STM       #01h,ST0          ;initialize the data page pointer

          RSBX      CPL               ;Make the processor to work using
                                      DP

          RSBX      FRCT              ;reset the fractional mode bit

          NOP

          NOP

;*****loop to make all x(n) zero initially*****

          STM       #150H,AR1         ;initialize ar1 to point to
          x(n) LD   #0H,A             ;make acc zero

          RPT       #34H

          STL       A,*AR1+           ;make all x(n) zero

;*****to read the adc data and store it in x(0)*****
LOOP:

          PORTR     06,0              ;start of
conversion CHK_BUSY:
```

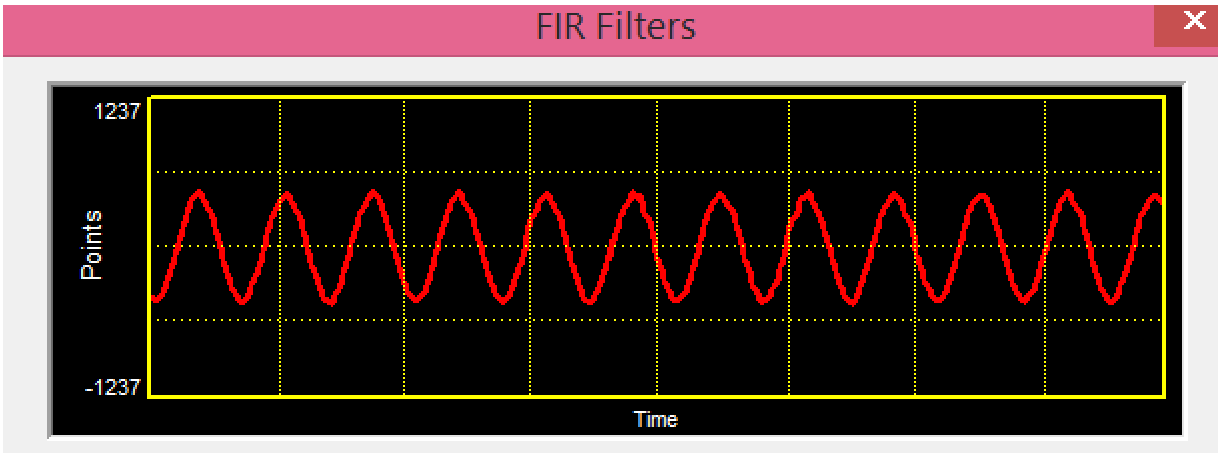
```
; PORTR    07,0            ;check for busy

; BITF     0,#20H
           CHK_BUSY,TC
; BC

PORTR      04,0            ;read the adc data

LD         0,A

AND        #0FFFH,A        ;AND adc data with 0fffh for 12 bit adc

XOR        #0800H,A        ;recorrect the 2's complement adc
                           data

SUB        #800H,A         ;remove the dc shift

STM        #150H,AR1       ;initialize ar1 with x(0)

STL        A,*AR1          ;store adc data in x(0)
```

```
        STM      #183H,AR2        ;initialize ar2 with x(n_end)
;*****start of convolution*****

        LD       #0H,A            ;sum is 0 initially

        RPT      #33H

        MACD     *AR2-,TABLE,A    ;convolution process

        STH      A,1,0H

        LD       0H,A

        ADD      #800H,A          ;add the dc shift to the convolution
                                  output

        STL      A,1H

        PORTW    1H,04H           ;send the output to the dac

        B        LOOP

TABLE:

     .word  0FCEFH

     .word  62H

     .word  0FD50H

     .word  14AH

     .word  0FE1BH

     .word  28FH

     .word  0FF11H

     .word  3E5H

     .word  0FFD1H

     .word  4ECH

     .word  0FFF5H

     .word  54FH

     .word  0FF28H
```

```
.word   4DAH
.word   0FD38H
.word   398H
.word   0FA2EH
.word   1DDH
.word   0F627H
.word   55H
.word   0F131H
.word   4BH
.word   0EA6DH
.word   568H
.word   0D950H
.word   459EH
.word   459EH
.word   0D950H
.word   568H
```

```
.word   0EA6DH
.word   4BH
.word   0F131H
.word   55H
.word   0F627H
.word   1DDH
.word   0FA2EH
.word   398H
.word   0FD38H
.word   4DAH
.word   0FF28H
.word   54FH
.word   0FFF5H
.word   4ECH
.word   0FFD1H
.word   3E5H
.word   0FF11H
.word   28FH
.word   0FE1BH
.word   14AH
.word   0FD50H
.word   62H
.word   0FCEFH
```

Output:



Result: A 2nd-order High Pass Butterworth IIR Filter with a cutoff frequency of 2 kHz and a sampling frequency of 43 kHz was successfully designed and implemented.