Scilab Manual for
Digital Signal Processing Lab
by Dr R Kumaraswamy
Electronics Engineering
Siddaganga Institute Of Technology[1]

Solutions provided by
Dr R Kumaraswamy
Electronics Engineering
Siddaganga Institute Of Technology

March 10, 2025

[1]Funded by a grant from the National Mission on Education through ICT, http://spoken-tutorial.org/NMEICT-Intro. This Scilab Manual and Scilab codes written in it can be downloaded from the "Migrated Labs" section at the website http://scilab.in

Contents

List of Experiments

List of Figures

Experiment: 1

Discrete-time signals

Scilab code Solution 1.1 Representation of discrete time signals

```

//scilab 5.5.2 ,OS: Ubuntu 14.04
//Generation of signals

//Unit Sample Sequence
clear ;clc ;close ;
L = 4;                                //length= 2*L+1
n = -L:L;                             // Time index vector
x = [zeros(1,L),1,zeros(1,L) ];
figure (1);
subplot(421),plot2d3(n,x),xtitle('Unit Sample sequence','n','x_1[n]');

//Unit step function
//clear ;clc ;close ;
n1=0:5
x1=[ones(1,6)];
subplot(422),plot2d3(n1,x1),xtitle('Unit Step sequence','n','x_2[n]')
//figure(1); plot2d3(n,x);
```

```
//xtitle('Discrete Unit Step Sequence','n','x[n]') ;

//Unit ramp function
//clear ;clc ;close ;
L = 4;                                  // Length of the sequence
n2= -L : L;
x2= [zeros(1,L ),0:L ];
,subplot(423),plot2d3(n2,x2),xtitle('Unit Ramp sequence','n','x_2[n]')
//plot2d3(n,x);
//xtitle(' Discrete Unit Ramp Sequence','n','x[n]') ;

//Discrete time Exponential signal
//clear ;clc ;close ;
a =0.5;    //For decreasing a<1 and For increasing exponential a>1
n3 = 0:10;
x3 = (a).^n3 ;
subplot(424),plot2d3(n3,x3),xtitle('Exponential Sequence','n','x_3[n]')
//plot2d3(n,x);xtitle('Exponentially Decreasing Signal ','n','x[n]');



//Sinusoidal signal
//clc;clear;
fm=100;// Frequency 100 Hz or input('Enter the input signal frequency:');              //100
k=3;// Number of cycles:3 or input('Enter the number of Cycles of input signal:');    //3
A=1; // Unit amplitude or input('Enter the amplitude of input signal:');             //5
tm=0:1/(fm*fm):k/fm;
x4=A*cos(2*%pi*fm*tm);
subplot(425),plot2d3(tm,x4),xtitle('Sinusoidal
```

```
Signal','n','x_4[n]')
//figure(1);plot2d3(tm,x);
//title('Graphical Representation of Sinusoidal Signal');
//xlabel('Time');ylabel('Amplitude');
//xgrid(1)

//Square wave
//clc;clear;
t=(0:0.1:4*%pi)';
x5=4*%pi*squarewave(t);
subplot(426),plot2d3(t,x5),xtitle('Square wave','n','x_5[n]')


//Triangular wave
//clear;clc;
A=5// input('enter the amplitude:');          //5
K= 2// input('enter number of cycles:');       //2
x6 = [0:A  A-1:-1:1];
x7=x6;
for i=1:K-1
x7=[x7 x6];
end
n7=0:length(x7)-1;      // Index of the sequence
subplot(427),plot2d3(n7,x7);xtitle('Triangular wave','time','amplitude');

//Sawtooth wave
//clc;clear;
A=5//input('enter the amplitude:');            //5
K=2;//input('enter number of cycles:');         //2
x8 = [0:A];
x9=x8;
for i=1:K-1
    x9=[x9 x8];
end
n9=0:length(x9)-1;
```

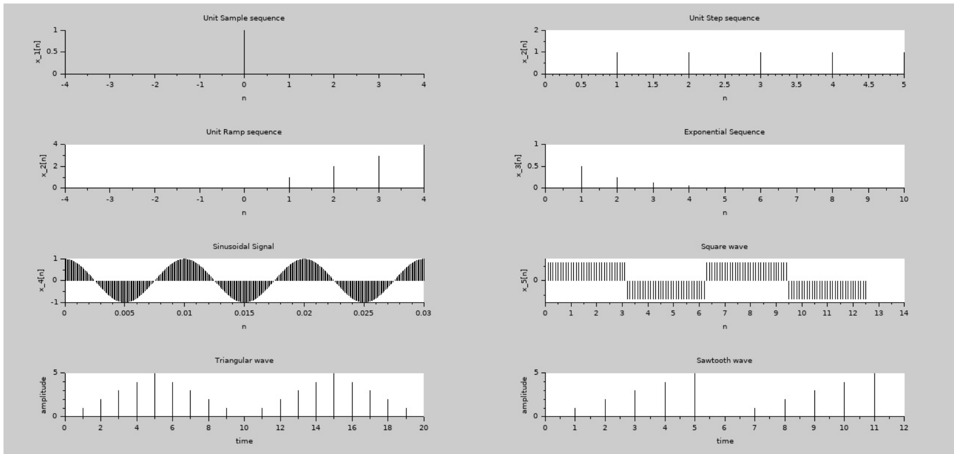

Figure 1.1: Representation of discrete time signals

```
subplot(428),plot2d3(n9,x9);xtitle('Sawtooth wave','
    time','amplitude');

// Complex valued signals
clc;clear;
n= [-10:1:10];
a=-0.1+0.3*%i;
x=exp(a*n);
figure(2);
subplot(221), plot2d3(n,real(x));xtitle('Complex
    valued signal','n','Real part');
subplot(223), plot2d3(n,imag(x));xtitle('Imaginary',
    'n');
subplot(222), plot2d3(n,abs(x));xtitle('Magnitude
    part','n');
theta=(180/%pi)*atan(imag(x),real(x));
subplot(224), plot2d3(n,theta);xtitle('Phase part','
    n');
```

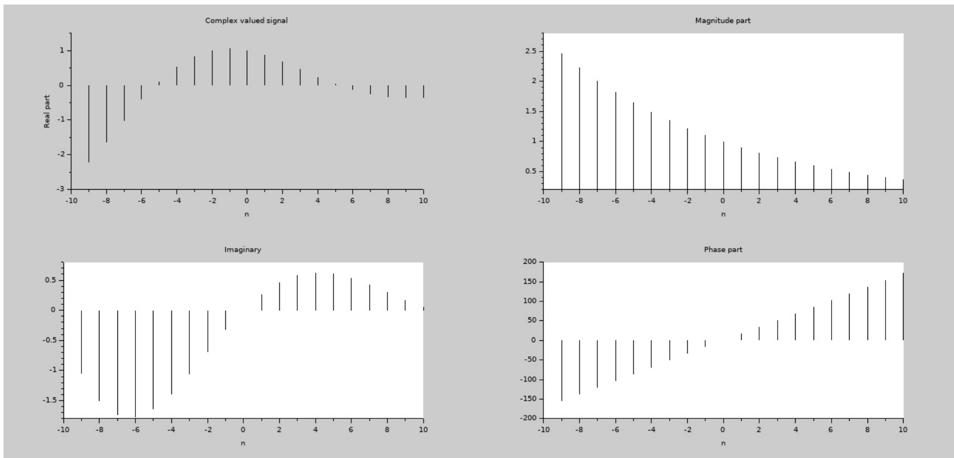

Figure 1.2: Representation of discrete time signals

Experiment: 2

Verification of Sampling Theorem

Scilab code Solution 2.1 To verify Sampling theorem in Time domain

```

//scilab 5.5.2 , OS: Ubuntu 14.04
//Sampling
clc;clear;
fm=100;//=input('Enter the input signal frequency:')
    ;       //100
k=4;//input('Enter the number of Cycles of input
    signal:'); //2
A=1;//input('Enter the amplitude of input signal:');
        //3
tm=0:1/(fm*fm):k/fm;
x=A*cos(2*%pi*fm*tm);
figure(1);
subplot(411),plot(tm,x);
title('ORIGINAL SIGNAL');xlabel('Time');ylabel('
    Amplitude');
xgrid(1)

//Sampling Rate(Nyquist Rate)=2*fm
```

```
fnyq=2*fm;

// UNDER SAMPLING
fs=(3/4)*fnyq;
n=0:1/fs:k/fm;
xn=A*cos(2*%pi*fm*n);
//figure(2);
subplot(412),plot2d3('gnn',n,xn);
plot(n,xn,'r');
title('Under Sampling');
xlabel('Time');
ylabel('Amplitude');
legend('Sampled Signal', 'Reconstructed Signal');
xgrid(1)
//NYQUIST SAMPLING
fs=fnyq;
n=0:1/fs:k/fm;
xn=A*cos(2*%pi*fm*n);
//figure(3);
subplot(413),
plot2d3('gnn',n,xn);
plot(n,xn,'r');
title('Nyquist Sampling');
xlabel('Time');
ylabel('Amplitude');
legend('Sampled Signal', 'Reconstructed Signal');
xgrid(1)
//OVER SAMPLING
fs=fnyq*10;
n=0:1/fs:k/fm;
xn=A*cos(2*%pi*fm*n);
//figure(4);
subplot(414)
plot2d3('gnn',n,xn);
plot(n,xn,'r');
title('Over Sampling');
xlabel('Time');
ylabel('Amplitude');
```

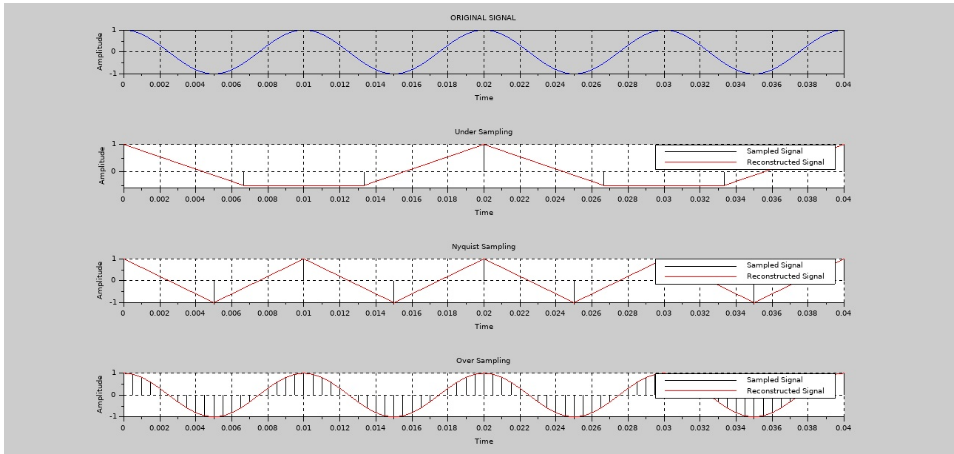

Figure 2.1: To verify Sampling theorem in Time domain

```
legend('Sampled Signal', 'Reconstructed Signal');
xgrid(1)
//Result
// Observing plots
```

Experiment: 3

Impulse response of the LTI system

Scilab code Solution 3.1 To determine the impulse response of a system given a difference equation

```
//scilab 5.5.2 , OS: Ubuntu 14.04
//To determine the impulse response of a LTI system, given the difference equation y[n]=b2 x(n-2)+b1 x(n-1)+ b0x(n) +a(1)y(n-1)
clear all;clc;close;
b=input('Enter the coefficients of input x[n]= ');//[1]
a=input('Enter the coefficients of output y[n]= ');//[1 -1 0.9]
x=[1 zeros(1,9)];//generate impulse sequence of length 10
n=0:9;
h=filter(b,a,x);
figure; plot2d3(n,h),
xtitle('Impulse response h[n]','Time index n', 'h[n]',' ');
//Example: y[n]-y[n-1]+0.9y[n-2]=x[n];a=[1] b=[1 -1 0.9]
```

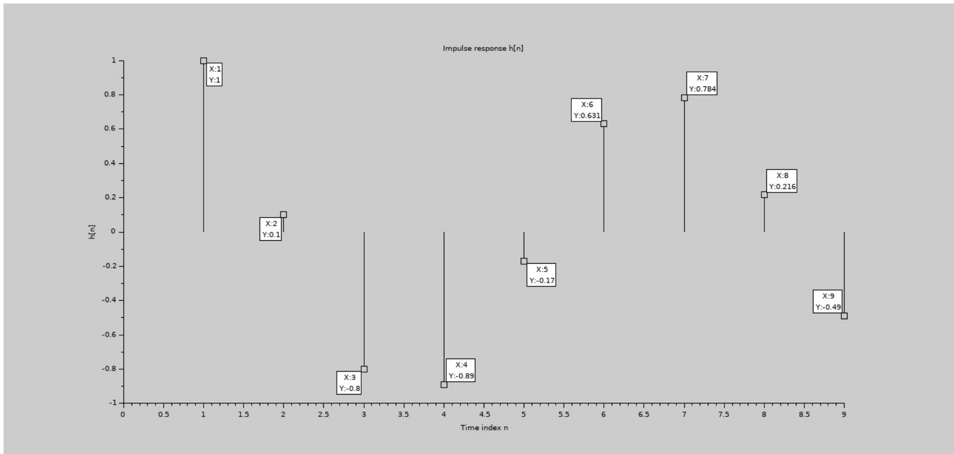

Figure 3.1: To determine the impulse response of a system given a difference equation

```
//n determines the length of the impulse response
    required
//Result:10 samples of h[n
    ]=[1,1,0.1,-0.8,-0.89,-0.17,0.631,0.784,
    0.2161,-0.4895]
```

Experiment: 4

Frequency response of the LTI system

Scilab code Solution 4.1 To plot the frequency response of a Digital system

```
//scilab 5.5.2 , OS: Ubuntu 14.04
//To determine the frequency response of a discrete-time system from its difference equation

//Design steps: Given a0 y[n] = -a2 y[n-2] - a1 y[n-1] + b0 x[n] + b1 x[n-1] + b2 x[n-2]
//1. System function H(z) = b0 + b1 z -1 + b2 z -2 / 1 + a1 z -1 + a2 z -2
//2. Put z= e (jw) to get the frequency response
//Design example: Plot the magnitude and phase response of the system represented by
//6y[n]+5y[n-1]+y[n-2]= 18x[n] + 8x[n-1]


clear;clc;
close;
b=input('Enter the coefficients of x[n]');//[1 -1]]
a=input('Enter the coefficients of y[n]');//[1 -0.5]
```

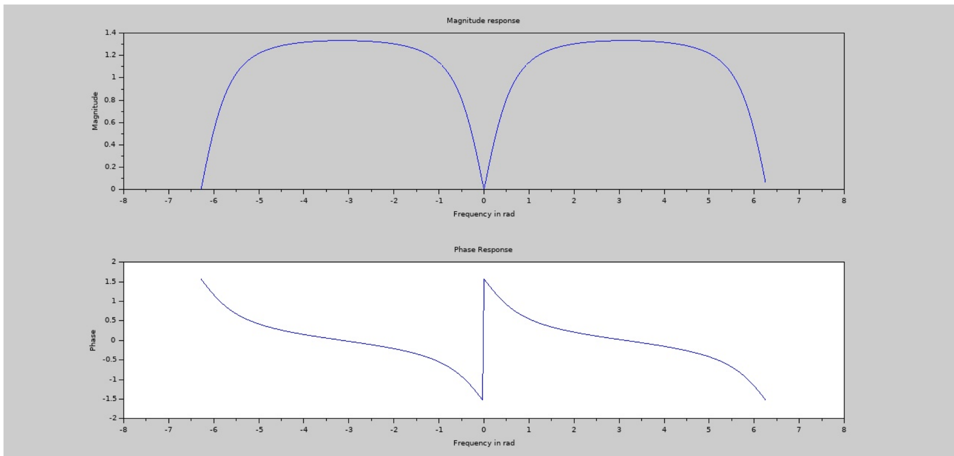

Figure 4.1: To plot the frequency response of a Digital system

```
//b=[18, 8];
//a=[6 5 1];
m= 0: length(b)-1; p=0:length(a)-1;
w=-2*%pi:%pi/100:2*%pi;//Plot over a interval of 4pi to observe periodicity
num = b* exp(-%i*m'*w);
den = a*exp(-%i*p'*w);
H= num./den;
magH = abs(H); angH= atan(imag(H),real(H));
figure;
subplot(211), plot( w, magH);
xtitle('Magnitude response','Frequency in rad','Magnitude');
subplot(212),plot(w, angH);
xtitle('Phase Response','Frequency in rad','Phase');
//Expected result
//H =[5,3.5802695 - 1.3881467i,2.6 - i,2.253303 - 0.4785341i,2.1666667,2.253303 + 0.4785341i,2.6 + i,3.5802695 + 1.3881467i,5]
```

Experiment: 5

Linear and Circular convolution

Scilab code Solution 5.1 To determine linear convolution

```
//scilab 5.5.2 , OS: Ubuntu 14.04
//  Linear Convolution  in time and frequency domain

clc ;clear all;close ;

x=[1 2 3 4];//input ('enter the input sequence
   values x(n)= ');     // [1 2 3 4]
h=[1 -1 0 -1];//input('enter the impulse sequence
   values h(n) = ');..// [1 -1 0 -1]

L1 = length(x);
L2 = length(h);

//Method 1 Using Direct Convolution Sum Formula
for i = 1: L1 +L2 -1
 conv_sum = 0;
 for j = 1: i
 if ((( i - j +1) <= L2 ) &( j <= L1 ) )
 conv_sum = conv_sum + x ( j ) * h (i -j +1) ;
 end ;
y(i) = conv_sum ;
```

```
end ;
end ;

disp(y,' Convolution Sum using Direct Formula Method = ' )

 //Method 2 Using In built Function
f = convol(x,h)
disp(f, ' Convolution Sum Result using Inbuilt Function = ')

//Method 3 Using frequency Domain multiplication
N = L1 +L2 -1;                                    // Linear convolution output length
x = [ x zeros(1 ,N - L1 ) ];
h = [ h zeros(1 ,N - L2 ) ];
f1 = fft(x)
f2 = fft(h)
f3 = f1.* f2 ;                                    // Multiplication in frequency domain
f4 = ifft(f3)
disp (f4 , 'Convolution Sum Result DFT and IDFT method = ')

//To plot input, impulse and output signals.
subplot (5,1,1) ;plot2d3(x);xtitle('Input signal x ','n','x[n]');
subplot(5,1,2) ;plot2d3(h);xtitle('Impulse signal h','n','h[n]');
subplot(5,1,3) ;plot2d3(y);xtitle('Liner Convolution using formula','n','y1[n]');
subplot(5,1,4) ;plot2d3(f);xtitle('Linear Convolution using Inbuilt function','n','y2[n]');
subplot(5,1,5) ;plot2d3(f);xtitle('Linear Convolution using DFT method','n','y3[n]');

// Expected result
//1.    1.    1.    0.  - 6.  - 3.  - 4.
```

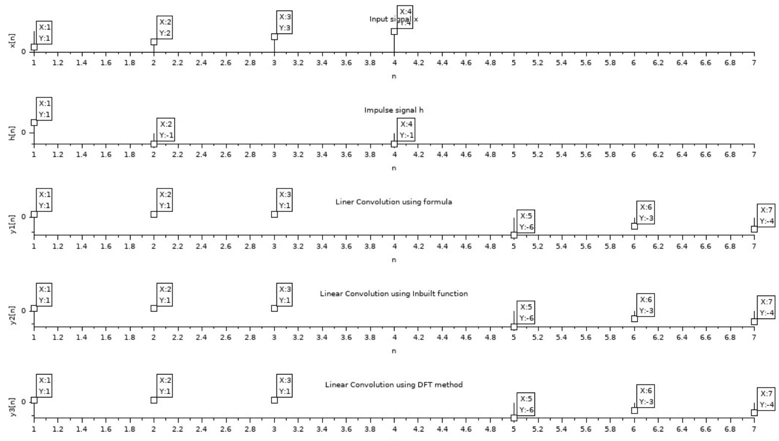

Figure 5.1: To determine linear convolution

Scilab code Solution 5.2 Circular convolution in time domain and using DFT relations

```

//scilab 5.5.2 , OS: Ubuntu 14.04
// Circular convolution of given discrete sequences in time domain (Matrix method)
clear;clc;
x1=input ('enter the first sequence values x1(n)= ');    // [1 2 3 4]
x2=input('enter the second sequence values x2(n) = ');    //[1 -1 0 -1]
L1 = length(x1);                              //length of first sequence
L2 = length(x2);                              //length of second sequence

```

```
if (L1 >L2)                                    //To make length of x1 and x2 are Equal
   for i = L2+1:L1
     x2(i) = 0;
   end
elseif (L2>L1)
   for i = L1+1:L2
     x1(i) = 0;
   end
end

N = length(x1);
x3 = zeros(1,N);                                    //x3 = Circular convolution result
a(1) = x2(1);
for j = 2:N
  a(j) = x2(N-j+2);
end
for i =1:N
  x3(1) = x3(1)+x1(i)*a(i);
end
X(1,:)=a;

//Calculation of circular convolution
for k = 2:N
    for j =2:N
       x2(j) = a(j-1);
    end
       x2(1) = a(N);
       X(k,:)= x2;
    for i = 1:N
        a(i) = x2(i);
        x3(k) = x3(k)+x1(i)*a(i);
    end
end
disp(X,'Circular Convolution Matrix x2[n]=')
disp(x3,'Circular Convolution Result x3[n] = ')
// Expected result
```

```
//Circular Convolution Matrix x2[n]=

//     1.  - 1.    0.  - 1.
//   - 1.    1.  - 1.    0.
//     0.  - 1.    1.  - 1.
//   - 1.    0.  - 1.    1.

 //Circular Convolution Result x3[n] =

 // -5.  -2.  -3.   0.

// Circular Convolution in frequency domain (DFT-IDFT method)
clear all;clc;close;
x1=input ('enter the first sequence values x1(n)= ');  // [1 2 3 4]
x2=input('enter the second sequence values x2(n) = '); //[1 -1 0 -1]
L=input ('enter the length of the sequence values L= '); //4

//Computing DFT
X1 = fft(x1,-1);                    //-1 for direct FFT
X2 = fft(x2,-1);
disp(X1,'DFT of x1[n] is X1(k)=')
disp(X2,'DFT of x2[n] is X2(k)=')

//Multiplication of 2 DFTs
X3 = X1.*X2;
disp(X3,'DFT of x3[n] is X3(k)=')
x3 =(fft(X3,1))                           //Circular Convolution Result ,1 for IFFT
disp(x3,'Circular Convolution x3[n]=')
//// Expected result
//DFT of x1[n] is X1(k)=      10.  - 2. + 2.i  - 2.  - 2. - 2.i

```

```
 //DFT of x1[n] is X2(k)=      - 1.      1.      3.      1.

// DFT of x3[n] is X3(k)=      - 10.   - 2. + 2.i   - 6.   - 2. - 2.i

 //Circular Convolution x3[n]=        -5.      -2.    -3.      0.

////Performing Linear Convolution using Circular Convolution
clear;clc;
x=input ('enter the input sequence values x(n)= ');   // [1 2 3 4]
h=input('enter the impulse sequence values h(n) = ');    //[1 -1 0 -1]
N1 = length(x);           //Length of input signal
N2 = length(h);      //Length of impulse response

N = N1+N2-1                              // Length of output response
disp(N,'Length of Output Response y(n)')

//Padding zeros to Make Length of 'h' and 'x' equal to length of output response 'y'

h1 = [h,zeros(1,N-N2)];
x1 = [x,zeros(1,N-N1)];

H = fft(h1,-1);
X = fft(x1,-1);
//Multiplication of 2 DFTs
Y = X.*H
y =(fft(Y,1))           //Linear Convolution Result

disp(X,'DFT of i/p X(k)=')
disp(H,'DFT of impulse sequence H(k)=')
disp(Y,'DFT of Linear Filter o/p Y(k)=')
disp(y,'Linear Convolution result y[n]=')
```

```
//Expected output
//Length of Output Response y(n)        7.

 //DFT of i/p X(k)=        10.   - 2.0244587 -
    6.2239817i   ,     0.3460107 + 2.4791213i   ,
    0.1784479 - 2.4219847i   ,     0.1784479 +
    2.4219847i   ,     0.3460107 - 2.4791213i   ,  -
    2.0244587 + 6.2239817i   ,
 //DFT of impulse sequence H(k)=      - 1.
    1.2774791 + 1.2157152i   ,,    0.5990311 +
    0.1930964i   ,     2.1234898 + 1.4088117i   ,
    2.1234898 - 1.4088117i   ,     0.5990311 -
    0.1930964i   ,     1.2774791 - 1.2157152i   ,
  //DFT of Linear Filter o/p Y(k)=      - 10.
     4.9803857 - 10.412171i   ,  - 0.2714383 +
     1.5518843i   ,    3.7910526 - 4.8916602i   ,
     3.7910526 + 4.8916602i   ,  - 0.2714383 -
     1.5518843i   ,     4.9803857 + 10.412171i   ,
 //Linear Convolution result y[n]=        1.      1.
       1.      0.      -6.    -3.     -4.
```

Experiment: 6

Spectral analysis using DFT

Scilab code Solution 6.1 To demonstrate spectral leakage

```

//scilab 5.5.2 , OS: Ubuntu 14.04
// Spectral Leakage
//Check the result for the following cases
//case(1): fm=10;fs=125;m=1;m=number of cycles
//case(2): fm=10;fs=125;m=2;
//case(3): fm=200;fs=10000;m=2.5;
//case(4): fm=75;fs=250;m=3;

clc;clear;close;
//fm=input('Enter the frequency of the input signal
    ');//message frequency in Hz
//fs=input('Enter the sampling frequency');//
    sampling frequency in Hz
//m=input('Enter the number of cycles of the input
    signal');// Number of cycles
//Case2:No spectral leakage
fm=10;fs=125;m=2;//Oversampling and integer number
    of cycles
t=0.0001:1/fs:m/fm;
x=3*cos(2*%pi*fm*t);          //signal
```

```
N=(m*fs/fm);                          //should be non-integer to obtain spectral leakage
for k=1:N
 X1(k)=0;
 for n=1:length(x)
  X1(k)=X1(k)+x(n).*exp((-%i).*2.*%pi.*(n-1).*(k-1)./N);
 end
 end
k=0:N-1
f=k*fs/N;        //frequency axis in Hz
figure(1),subplot(221),plot2d3(t,x),xlabel('time'),ylabel('x(n)'),title('No leakage: m=2, f=10 and Fs=125 Hz'),subplot(223),plot2d3(f,abs(X1)),xlabel('freq in Hz'),ylabel('Mag');//Case 3: Spectral leakage
  fm=10;fs=125;m=2.5;//Oversampling and integer number of cycles
t=0.0001:1/fs:m/fm;
x=3*cos(2*%pi*fm*t);         //signal
N=(m*fs/fm);                          //should be non-integer to obtain spectral leakage
for k=1:N
 X1(k)=0;
 for n=1:length(x)
  X1(k)=X1(k)+x(n).*exp((-%i).*2.*%pi.*(n-1).*(k-1)./N);
 end
 end
k=0:N-1
f=k*fs/N;        //frequency axis in Hz
figure(1),subplot(222),plot2d3(t,x),xlabel('time'),ylabel('x(n)'),title('Spectral leakage: m=2.5, f=10 and Fs=125 Hz'),subplot(224),plot2d3(f,abs(X1)),xlabel('freq in Hz'),ylabel('Mag')
```

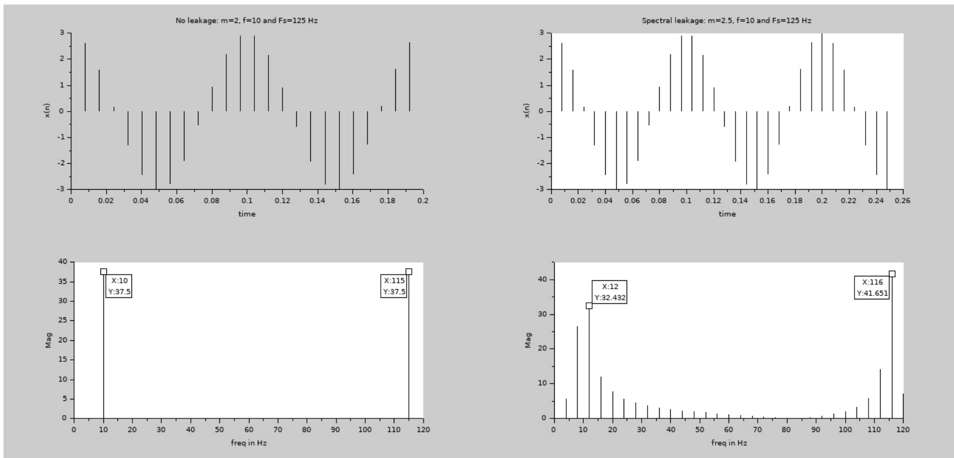

Figure 6.1: To demonstrate spectral leakage

Scilab code Solution 6.2 To demonstrate effects of zeropadding and zero insertion on the spectrum

```
//scilab 5.5.2 , OS: Ubuntu 14.04
//Effect of zero padding and interpolation
// Effect of Zero padding
clc;clear;close;
x= input ('enter the input sequence values x(n)= ');
    //[1 2 3 4]
k= input ('enter the number of zeros to be padded= '
  );  //1020 (For 1024 point DFT))
N=length(x);
x_pad=[x zeros(1,k)];
N1=length(x_pad);
f=0:N-1;
f1=0:N1-1;
X=abs(fft(x));
```

```
X_pad= abs(fft(x_pad));
figure(1);
subplot(221),plot2d3(x),title(' Original sequence'),
    subplot(223),plot2d3 (f,X), title('Spectrum of
    Original sequence');
subplot(222),plot2d3(x_pad),title('Zero-padded
    sequence'),subplot(224), plot2d3 (f1,X_pad),title
    ('Spectrum of Zero-padded sequence')
//// Effect of inserting zeros in between samples (
    Interpolation)
x= input ('enter the input sequence values x(n)= ');
        //[1 2 3 4]
k= input ('enter the number of zeros to be inserted=
    ');//2 (Vary and observe effect of zero
    interpolation)
x_mod=[];
N=length(x);
//
for  i= 1: N
x_mod=[x_mod, x(i), zeros(1,k)];
end
N1=length(x_mod);
f=0:N-1;
f1=0:N1-1;
X=abs(fft(x));
X_mod= abs(fft(x_mod));
figure(2);subplot(221),plot2d3(x),title(' Original
    sequence'),subplot(223),plot2d3 (f,X), title('
    Spectrum of Original sequence');
subplot(222),plot2d3(x_mod),title('Zero-interpolated
     sequence'),subplot(224), plot2d3 (f1,X_mod),
    title('Spectrum of Zero-inserted sequence')
```

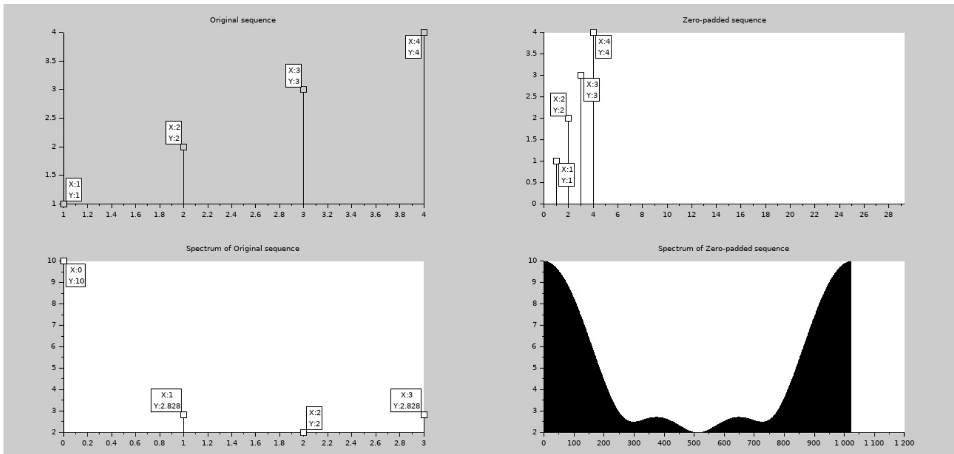

Figure 6.2: To demonstrate effects of zeropadding and zero insertion on the spectrum

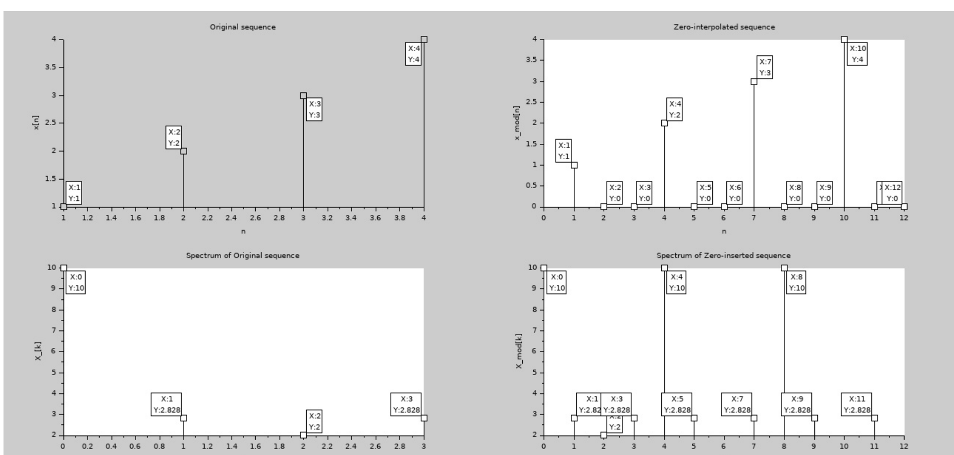

Figure 6.3: To demonstrate effects of zeropadding and zero insertion on the spectrum

Experiment: 7

FIR filter design

Scilab code Solution 7.1 Design of FIR filter using Windowing method

```

//scilab 5.5.2 , OS: Ubuntu 14.04
//Design of FIR filters using windowing
// Design a digital FIR low pass filter with following specifications.
//a) Pass band cut-off frequency      :wp=__________radians
//b) Pass band ripple                 :rp=__________dB
//c) Stop band cut-off frequency      :ws=__________radians
//d) Stop band attenuation                :rs=__________dB
//Choose an appropriate window function and determine impulse response and provide a plot of frequency response of the designed filter.

//Design example:
//Design a digital FIR low pass filter with following specifications.
//a) Pass band cut-off frequency      :0.3   rad
//b) Pass band ripple                       :0.25 dB
```

```
//c) Stop band cut-off frequency      :0.45    rad
//d) Stop band attenuation                    : 50 dB
clc;
clear;
close;
wp=input('enter the pass band edge in rad');
ws=input('enter the stop band edge in rad');
rs=input('enter the stop band ripple in dB');
freq_points=1024;
freq_divs=(freq_points/2)-1;
k=4; //Hamming window (decided based on stop band
    attenuation)
trw=ws-wp;
N=(k*2*%pi/trw);
N=ceil(N);
remainder=N-fix(N./2).*2
if remainder==0
    N=N+1;
end

wc=wp;
aph=(N-1)/2;
for n=0:N-1
    if n==aph
        hdn_minusalph(n+1)=wc/%pi;

 else
     hdn_minusalph(n+1)= sin(wc.*(n-aph))./(%pi.*(n-
        aph));

end
end
n=0:N-1;
wndw=window('hm',N);

hn=hdn_minusalph.*wndw';
figure(1);subplot(311);plot2d3(n,wndw);xlabel('n');
   ylabel('wndw');title('Hamming Window function');
```

```
subplot(312); plot2d3(n,hdn_minusalph);xlabel('n');ylabel('hdn_minusalph');title('Impulse response of IIR filter');
subplot(313); plot2d3(n,hn);xlabel('n');ylabel('hn');title('Impulse response of FIR filter');
//omega=0:%pi/freq_divs:%pi;
h=[hn' zeros(1,freq_points-length(hn))];;//For a 1024 point DFT
H=fft(h);
H_mag=20*log10(abs(H));
H_ang=atan(imag(H),real(H));
H_phase=unwrap(H_ang);
w=(0:freq_divs)./(freq_points);
w1=w*%pi;
figure(2);subplot(211),plot2d(w1,H_mag(1:512));
xtitle('Magnitude response','w (rad)','Magnitude(dB)');
subplot(212),plot2d(w1,H_phase(1:512));
xtitle('Phase Response','w (rad)','Phase (rad)');



//Problems:

//1. Design a digital FIR low pass filter with following specifications.
//a) Pass band cut-off frequency      :0.4    rad
//b) Pass band ripple                        :0.25 dB
//c) Stop band cut-off frequency      :0.6    rad
//d) Stop band attenuation                  : 44 dB

//2. Design a digital FIR low pass filter with following specifications.
//a) Pass band cut-off frequency      :0.25    rad
//b) Pass band ripple                        :0.25 dB
//c) Stop band cut-off frequency      :0.3    rad
//d) Stop band attenuation                  : 50 dB
```

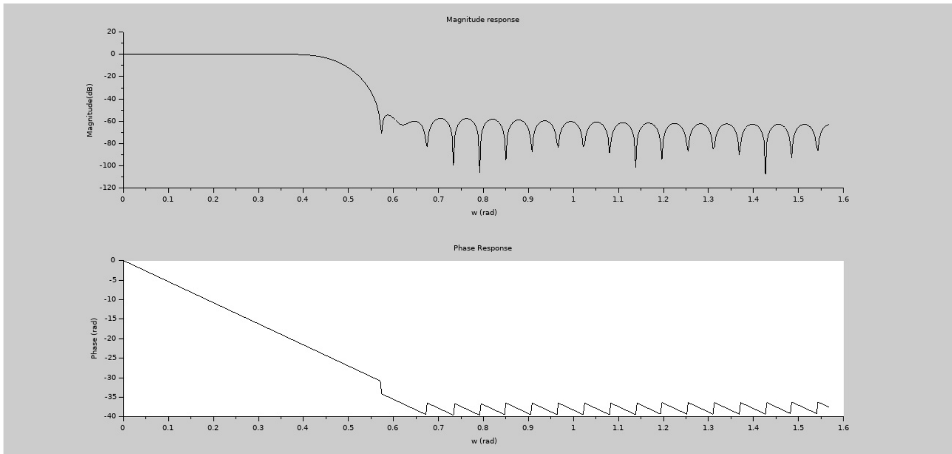

Figure 7.1: Design of FIR filter using Windowing method

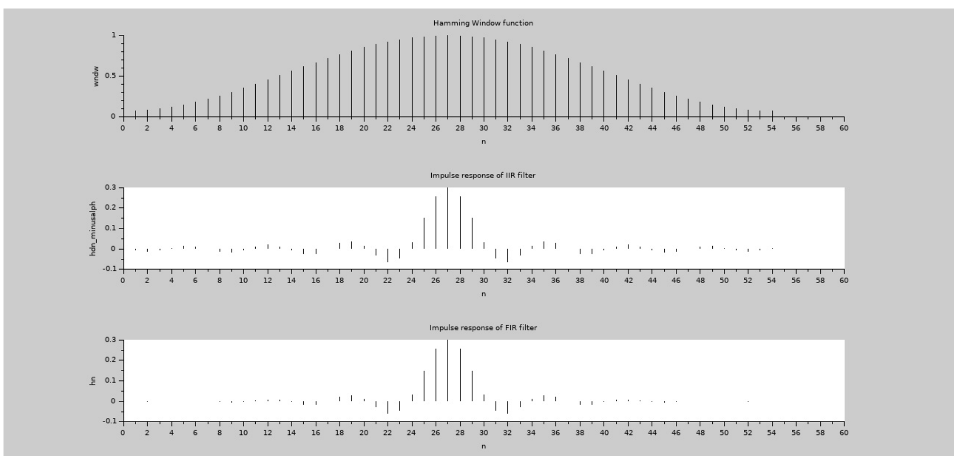

Figure 7.2: Design of FIR filter using Windowing method

Experiment: 8

Design of Hilbert transformer using FIR filter

Scilab code Solution 8.1 Design of a digital Hilbert Transformer using FIR filter

```
//scilab 5.5.2 , OS: Ubuntu 14.04
//Design a differentiator using a Hamming window of length N=21. Plot the time and frequency domain responses.
//Design a length-25 digital Hilbert transformer using a Hann window.

//Design of Hilbert transformer
//The ideal frequency response of a linear phase Hilbert transformer is given by
//Hd(e jw) =    - j  e(-j  w  ) ,    0 < w < pi
//                   j  e(-j  w  ) ,    -pi < w < 0

//The ideal impulse response is given by

//hd(n-    )=     2/pi    (sin2  pi( n         )/2) / ( n
        ) ,        n
//                         0,  n=
```

```
//Scilab Program
//Inputs: Window length and type of window
clc;clear;close;

N = 41;//input("enter the window length");  //55
freq_points=1024;
windowfn =window('hm',N);// Hamming window ()Window type can be changed here)
m = 0:N-1;
aph = (N-1)/2;
for n=0:N-1
    if n==aph
        hd(n+1)=0;

 else
     hd(n+1)=(2/%pi)*((sin((%pi/2)*(n-aph)).^2)./(n-aph));

end
end
n=0:N-1;
 hn = hd.*windowfn';

 omega=-%pi:2*%pi/(freq_points-1):%pi;

 z=%z;
den1=real(z^(N-1));
num=0;
for n=0:N-1
    num=num+(hn(n+1).*z^(N-n-1));
end
num1=real(num);
Hz=num1./den1;
w=exp(%i*omega);
rep=freq(Hz("num"),Hz("den"),w);
magH=abs(rep);
```

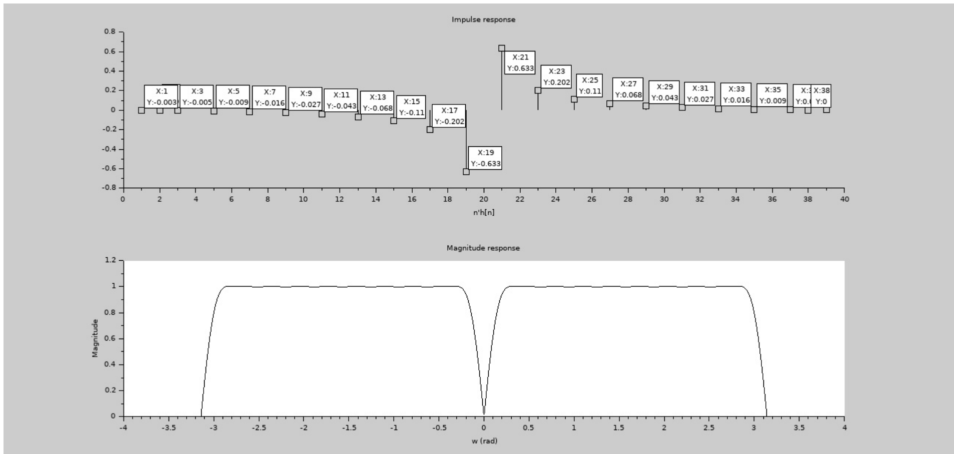

Figure 8.1: Design of a digital Hilbert Transformer using FIR filter

```
figure;subplot(211),plot2d3(m,hn),xtitle('Impulse
    response','n''h[n]')
, subplot(212),plot2d(omega,magH);
xtitle('Magnitude response','w (rad)','Magnitude');
//Expected result
//Magnitude response graph
```

Experiment: 9

Design of digital differentiator using FIR filter

Scilab code Solution 9.1 Design of Digital Differentiator using a FIR filter

```

11//scilab 5.5.2 , OS: Ubuntu 14.04
//Design a differentiator using a Hamming window of length N=21. Plot the time and frequency domain response
//Inputs: Window length and Type of window
//The frequency response of a linear-phase ideal differentiator is given by
//Hd(e jw) =    j     , 0<     <
//                    -jw,    -     <     < 0
//The ideal impulse response of a digital differentiator shifted by     with linear phase is given by
//hd( n      ) =    cos     (n          ) / ( n -      ), n

//                         0,    n =

//Scilab Program:
```

```
clc;clear;close;
N = 41;// input(" enter the window length");  //55
freq_points=1024;
windowfn =window('hm',N);//Hamming wuindow (Try with
    different windows)
m = 0:N-1;
aph = (N-1)/2;
for n=0:N-1
    if n==aph
        hd(n+1)=0;

 else
     hd(n+1)= cos(%pi*(n-aph))./(n-aph);

end
end
n=0:N-1;
 hn = hd.*windowfn';

omega=-%pi:2*%pi/(freq_points-1):%pi;

z=%z;
den1=real(z^(N-1));
num=0;
for n=0:N-1
    num=num+(hn(n+1).*z^(N-n-1));
end
num1=real(num);
Hz=num1./den1;
w=exp(%i*omega);
rep=freq(Hz("num"),Hz("den"),w);
magH=abs(rep);
figure;subplot(211),plot2d3(m,hn),xtitle('Impulse
   response','n','h[n]'),subplot(212),plot2d(omega,
   magH);
xtitle('Magnitude response','w (rad)','Magnitude');
//Expected result
//Magnitude response graph
```

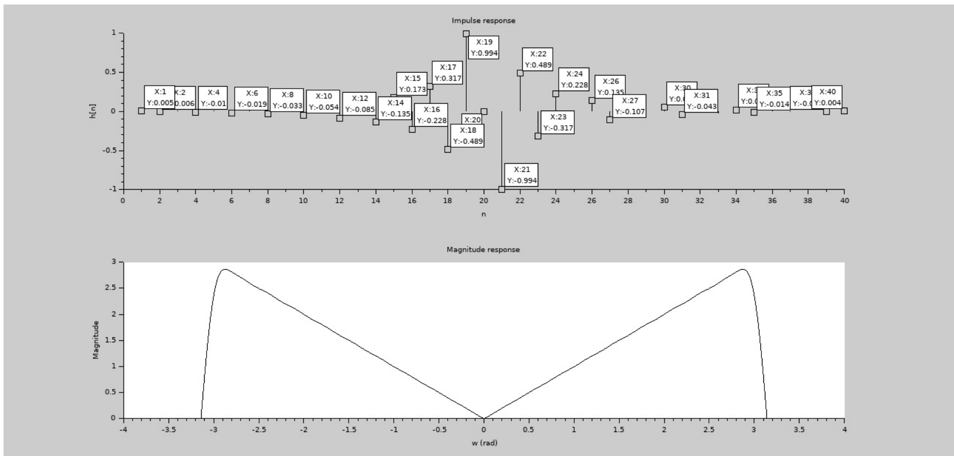

Figure 9.1: Design of Digital Differentiator using a FIR filter

Experiment: 10

Design of IIR filter

Scilab code Solution 10.1 Design of digital Butterworth lowpass filter

```
//scilab 5.5.2 , OS: Ubuntu 14.04
//Program To Design the Digtial Butterworth  IIR
    Filter
//Design a digital IIR low pass filter with
    following specifications.
//a) Pass band cut-off frequency     :1000 Hz
//b) Pass band ripple                         :-1 dB
//c) Stop band cut-off frequency     :3000 Hz
//d) Stop band attenuation                   : -15 dB
//Sampling frequency: 15000 Hz

clear all;clc;close;
f1=1000;//input('Enter the pass band edge(Hz)= ');
f2=3000;//input('Enter the stop band edge(Hz)= ');
k1=-1;//input('Enter the pass band attenuation(dB)=
    ');
k2=-15;//input('Enter the stop band attenuation(dB)=
    ');
fs=10000;//input('Enter the sampling rate(Hz)= ');

//Digital filter specifications(rad)
```

```
w1=2*%pi*f1*1/fs;
w2=2*%pi*f2*1/fs;

//Pre warping
o1=2*fs*tan(w1/2)
o2=2*fs*tan(w2/2)

//Design of analog filter
n=log10(((10.^(-k1/10))-1)/((10.^(-k2/10))-1))./(2*log10(o1/o2));
n=round(n);
wn= o2./((10.^(-k2/10)-1).^(1/(2*n)));

//[h,poles,zeros,gain]=analpf(n,'butt',[0 0],wn)hb.dt = 'c';
//[fr,hr]=repfreq(hb,fmin,fmax)

h=buttmag(n,wn,1:2*%pi*fs);
mag=20*log10(h)';


//Converting analog to digital filter
hz=iir(n,'lp','butt',0.25,[])
//g*poly(z,'z')/poly(p,'z')

[hzm,fr]=frmag(hz,256);
magz=20*log10(hzm)';

subplot(2,1,1),plot2d((1:2*%pi*fs)',mag),xtitle('Analog IIR filter: lowpass','Analog frequency in rads/sec', 'dB',' ');subplot(2,1,2),plot2d(fr,magz);xtitle('Digital IIR filter: lowpass 0 < fr < 0.5','frequency', 'dB',' ');

//note: Use zoom/axis commands to verify the design.
```

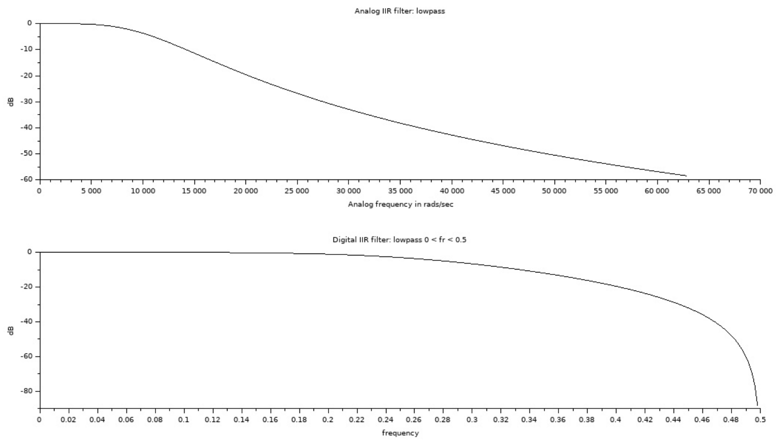

Figure 10.1: Design of digital Butterworth lowpass filter

Scilab code Solution 10.2 Design of Digital Chebyshev lowpass filter

```
//scilab 5.5.2 , OS: Ubuntu 14.04
//Program To Design the Digtial Chebyshev IIR Filter
////Design example:
//Design a digital IIR low pass filter with
    following specifications.
//a) Pass band cut-off frequency      :1000 Hz
//b) Pass band ripple                          :-1 dB
//c) Stop band cut-off frequency      :3000 rad
//d) Stop band attenuation                   : -15 dB
//Sampling frequency: 15000 Hz

clear all;clc;close;
f1=1000;//input('Enter the pass band edge(Hz)= ');
f2=3000;//input('Enter the stop band edge(Hz)= ');
rp=-1;//input('Enter the pass band ripple(dB)= ');
```

```
rs=-15;//input('Enter the stop band attenuation(dB)= ');
fs=10000;//input('Enter the sampling rate(Hz)= ');
//Digital filter specifications(rad)
w1=2*%pi*f1*1/fs
w2=2*%pi*f2*1/fs
//Pre warping
o1=2*fs*tan(w1/2)
o2=2*fs*tan(w2/2)
or=o2/o1;//Stop-band edge of normalized lowpass filter
A2 =10.^(-rs/10);
A=sqrt(A2);
epsilon2 = (10.^(-rp/10)-1);
epsilon=sqrt(epsilon2)
g=((A2-1).^0.5./epsilon)

N = (acosh(g))/(acosh(or))
N = ceil(N)
oc=o1;
//[pols,gn] = zpch1(N,epsilon,o1)
//Hs = poly(gn,'s','coeff')/real(poly(pols,'s'))
h=cheb1mag(N,oc,epsilon,1:2*%pi*fs);
mag=20*log10(h)';
//plot2d((1:1000)',mag,[2],"011"," ",[ymax,ymin,fmax,fmin])
//gain=20*log10(abs(h_s)); %Verify the specification [k1,k2] at prewarped frequencies
//subplot(211);
//plot(omega,gain);
//xlabel( frequency in rad/ s e c );
//Converting analog to digital filter
fc=w1/(2*%pi);
delta1=(1-(1./A2));
//1-ripple in passband
hz=iir(N,'lp','cheb1',[fc],[delta1 0]);
//for cheb1 filters 1-delta(1)<ripple<1 in passband
//g*poly(z,'z')/poly(p,'z')
```

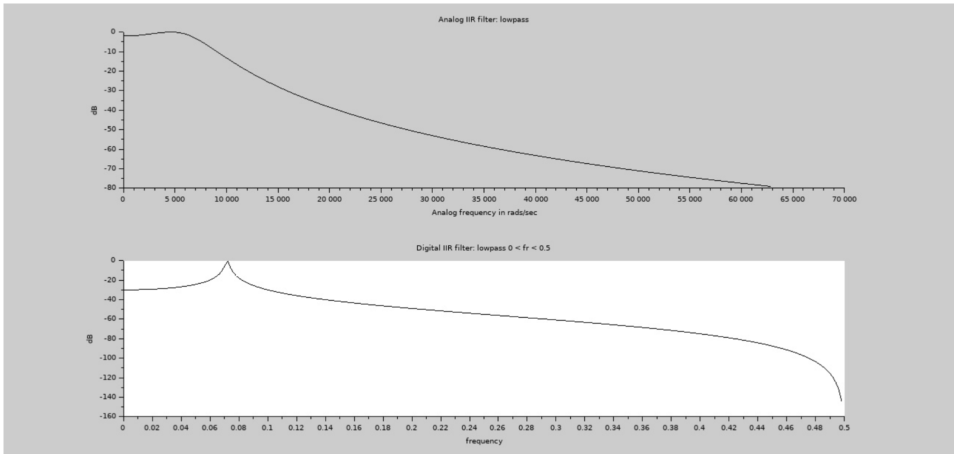

Figure 10.2: Design of Digital Chebyshev lowpass filter

```
[hzm,fr]=frmag(hz,256);
magz=20*log10(hzm)';
figure(1);subplot(2,1,1),plot2d((1:2*%pi*fs)',mag),
   xtitle('Analog IIR filter: lowpass','Analog
   frequency in rads/sec', 'dB',' ');subplot(2,1,2),
   plot2d(fr,magz);xtitle('Digital IIR filter:
   lowpass 0 < fr < 0.5','frequency', 'dB',' ');
```

Experiment: 11

Application of IIR filter

Scilab code Solution 11.1 To design a digital IIR Butterworth filter to suppress noise

```
//scilab 5.5.2 , OS: Ubuntu 14.04
// This program will suppress noise at f=4000 Hz
   using Butterworth prototype
//pass band edge=f1=1500Hz
//stop band edge=f2=2000 Hz
//sampling rate =Fs=10000 Hz = 1/Ts
//passband attenuation = -1db
//stop attenuation = -3 db

clear all;clc;close;
f1=input('Enter the pass band edge(Hz)= ');
f2=input('Enter the stop band edge(Hz)= ');
k1=input('Enter the pass band attenuation(dB)= ');
k2=input('Enter the stop band attenuation(dB)= ');
fs=input('Enter the sampling rate(Hz)= ');

signal_fo=1000;
noise_fo=4000;

//Digital filter specifications(rad)
```

```
20 w1=2*%pi*f1*1/fs;
21 w2=2*%pi*f2*1/fs;
22
23 //Pre warping
24 o1=2*fs*tan(w1/2)
25 o2=2*fs*tan(w2/2)
26
27 //Design of analog filter
28 n=log10(((10.^(-k1/10))-1)/((10.^(-k2/10))-1))./(2*log10(o1/o2));
29 n=round(n);
30 wn= o2./((10.^(-k2/10)-1).^(1/(2*n)));
31
32 //[h,poles,zeros,gain]=analpf(n,'butt',[0 0],wn)hb.dt = 'c';
33 //[fr,hr]=repfreq(hb,fmin,fmax)
34
35 h=buttmag(n,wn,1:2*%pi*fs);
36 mag=20*log10(h)';
37 //plot2d((1:2*%pi*fs)',mag)
38 //xtitle('Analog IIR filter: lowpass','Analog frequency in rads/sec', 'dB',' ');
39
40 //Converting analog to digital filter
41 hz=iir(n,'lp','butt',0.25,[])
42 //g*poly(z,'z')/poly(p,'z')
43
44 [hzm,fr]=frmag(hz,256);
45 magz=20*log10(hzm)';
46 fr1=fr*fs;
47 //figure;plot2d(fr1',magz');xtitle('Digital IIR filter: lowpass 0 < fr < 0.5','frequency', 'dB',' ');
48
49 //////note: Use zoom/axis commands to verify the design.
50 //These coefficients are to be read from variable hz (line 41, output of iir function)
```

```
num=[0.2928 0.5858 0.2928];
den=[1 0 0.1716];// In negative powers of z

//Signal generation (sine wave of frequency 1000 Hz) of length 1 second
t=0:1/fs:10/signal_fo;//10 cycles of input
original_signal=sin(2*%pi*signal_fo*t);

//Noise generation (sine wave of frequency 4000 Hz) of length 1 second
t=0:1/fs:10/signal_fo;
noise=sin(2*%pi*noise_fo*t);

noisy_signal=original_signal+noise;

filter_output=filter(num,den,noisy_signal);

//Plot original, noisy and filtered outputs

figure;subplot(3,1,1), plot2d(t,original_signal), xtitle('Original signal','t','x(t)'),
subplot(3,1,2), plot2d(t,noisy_signal),xtitle('Noisy signal','t','n(t)'),
subplot(3,1,3), plot2d(t,filter_output),xtitle('Filtered signal','t','y(t)');
l1=length(original_signal);
l2=length(noisy_signal);
N=512;
x=[original_signal zeros(1,N-l1)];//To make it of length 512
n=[noisy_signal zeros(1,N-l1)];
y=[filter_output zeros(1,N-l1)];
X=fft(x);
N=fft(n);
Y=fft(y);
f=(0:511)*fs;
figure;
subplot(3,1,1), plot2d(f,abs(X)),xtitle('
```

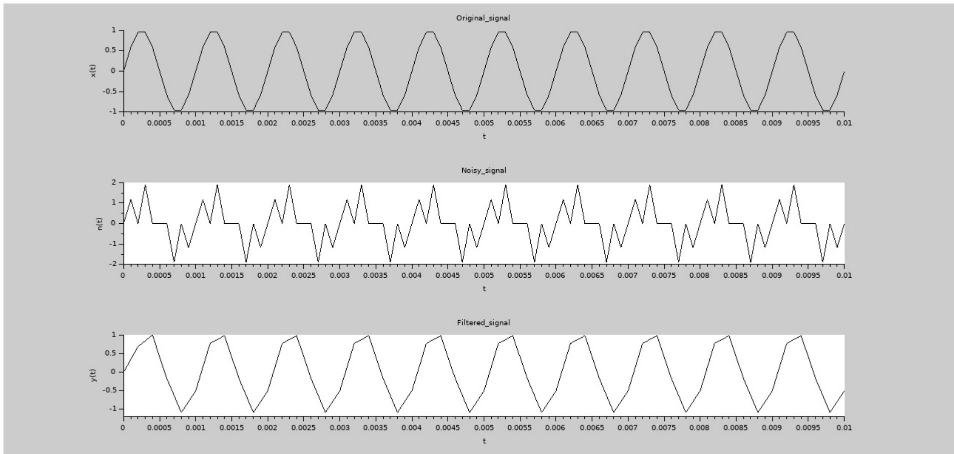

Figure 11.1: To design a digital IIR Butterworth filter to suppress noise

```
    Original_signal','F','X(f)'),
subplot(3,1,2), plot2d(f,abs(N)),xtitle('
    Noisy_signal','F','N(f)'),
subplot(3,1,3), plot2d(f,abs(Y)),xtitle('
    Filtered_signal','F','Y(f)');
```

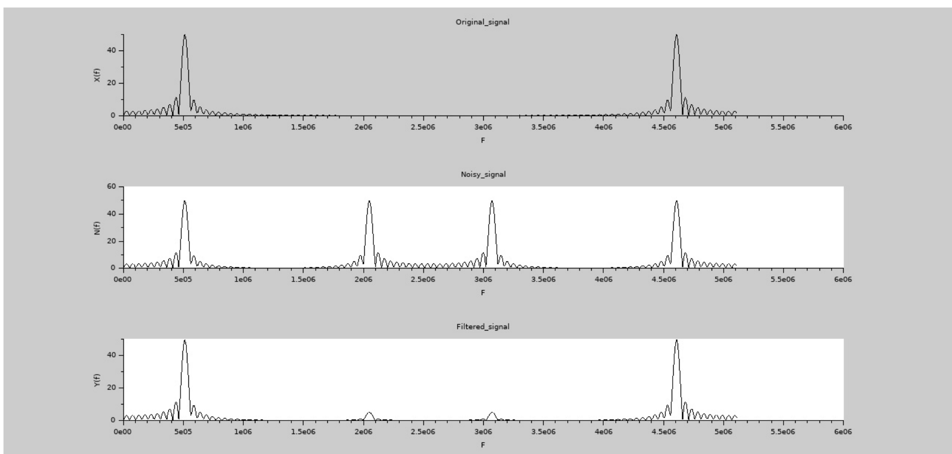

Figure 11.2: To design a digital IIR Butterworth filter to suppress noise

Experiment: 12

Design of Notch filter

Scilab code Solution 12.1 Suppression of noise at a given frequency using Notch filter

```

//scilab 5.5.2 , OS: Ubuntu 14.04
//Program To Design a simple notch filter and verify
// Design a simple notch filter to stop a disturbance with frequency F_0=3.5 kHz and a sampling frequency F_s=8 kHz.
//Also, verify the notch filter operation by adding a sinewave of F_0 Hz to a speech signal, filter and verify.

//Scilab Program:
clc;clear;close;
f=3500;//input("Enter the frequency in Hz");        //3500
fs=8000;//input("Enter the sampling rate");         //8000
r=0.98;//input("Enter the radius of the pole in the z-plane"); //0.98
w=2*%pi*f/fs;
z1=exp(%i*w);
```

```
z2=exp(-%i*w);
p1=r*exp(%i*w);
p2=r*exp(-%i*w);
z=%z;
num1=(real((z-z1)*(z-z2)))
den1=(real(((z-p1)*(z-p2))))
Hz=num1./den1
//figure(1);plzr(Hz);zgrid()
[h1 fr]=frmag(Hz,512)
figure(1);plot2d(fr*fs,h1);xtitle('Magnitude
    response','frequency in Hz','Mag');

//Noise generation

original_signal=wavread('home/hyrkswamy/kswamy/
    Coursework/SAP/wav/mask.wav');
t=0:1/fs:(length(original_signal)-1)/fs;
noise=sin(2*%pi*f*t);
noisy_signal=original_signal+noise;

filter_output=filter(num1,den1,noisy_signal);

//Play back the original, noisy and filtered outputs
playsnd(original_signal,fs);
pause;
playsnd(noisy_signal,fs);
pause;
playsnd(filter_output,fs);
```

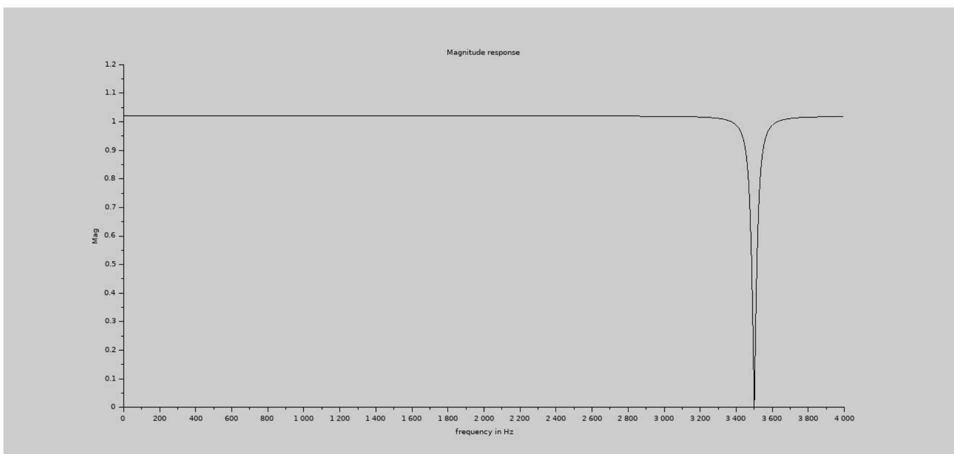

Figure 12.1: Suppression of noise at a given frequency using Notch filter

Experiment: 13

Design of Resonator

Scilab code Solution 13.1 Design of a Notch filter to filter noise at a given frequency

```

//scilab 5.5.2 , OS: Ubuntu 14.04
//Design a digital resonator that resonates at 1000 Hz. Assume Fs=8000 Hz.
// Calculate the pole location
//w=2*pi*f/fs;
//Complex conjugate pair of poles at w=pi/4 and -pi/4
//Assume radius=0.98 (near to unit circle but inside the unit circle)

//Scilab Program:
clc;
clear;
close;
f=1000;//input("Enter the frequency in Hz");//1000
fs=8000;//input("Enter the sampling rate");//8000
r=0.98;//input("Enter the radius of the pole in the
```

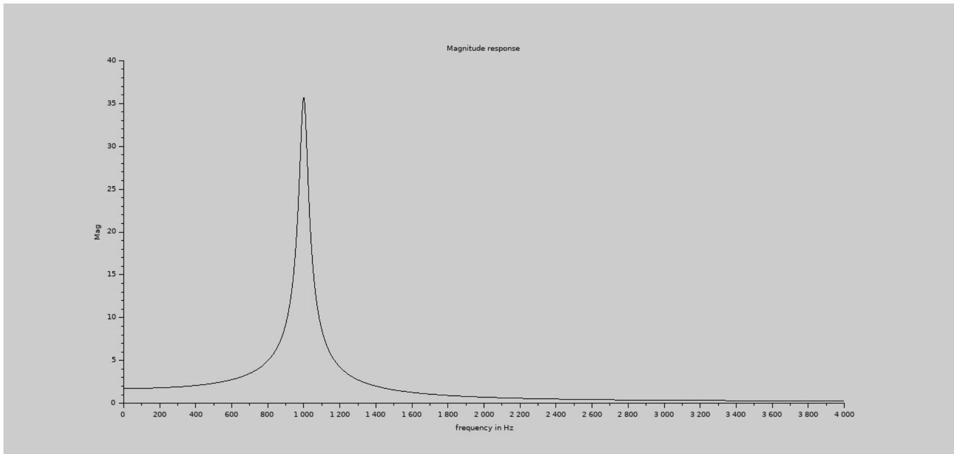

Figure 13.1: Design of a Notch filter to filter noise at a given frequency

```
    z−plane");    //0.98
w=2*%pi*f/fs;
pole1=r*exp(%i*w);
pole2=r*exp(-%i*w);

z=%z;

num1=real(z^(2));
den1=real(z^(2)-1.3859293*z+0.9604);
Hz=num1./den1;
//figure;
//plzr(Hz);
[h1 fr]=frmag(Hz,1024);
figure;
plot2d(fr*fs,h1);
xtitle('Magnitude response','frequency in Hz','Mag')
    ;
```