Digital Signal Processing Lab

Experiment Report

By Pranit Dalal
Roll no. 16EC10016
Group 22 (Tuesday)

INDEX

Digital Signal Processing Lab

Expt. No. 1
Sampling

By Pranit Dalal
Roll no. 16EC10016
Group 22 (Tuesday)

AIM:

(a)Sampling of a sinusoidal waveform
(b)Sampling at below Nyquist rate and effect of aliasing
(c)Spectrum of a square wave
(d)Interpolation or upsampling

(a)Sampling of a sinusoidal waveform:

(i)Theory:

Sampling at Nyquist rate: It refers to the sampling rate being greater than or equal to twice the bandwidth of a bandlimited function or a bandlimited channel. Suppose the highest frequency component, in hertz, for a given analog signal is fmax.

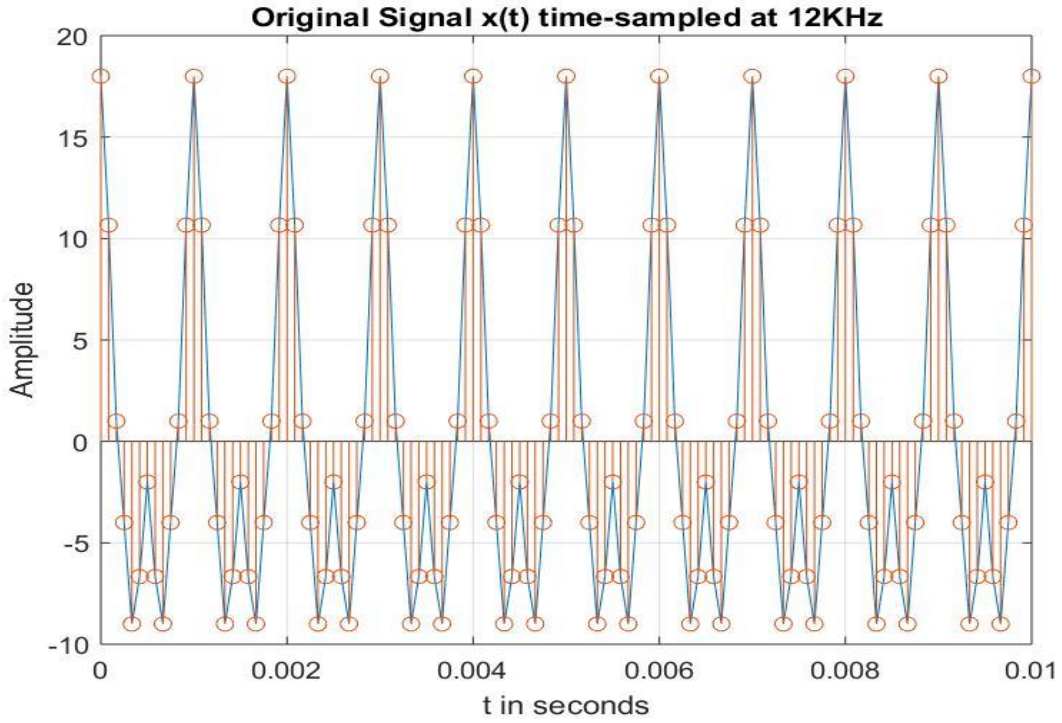
According to the Nyquist Theorem, the sampling rate must be at least 2*fmax, or twice the highest analog frequency component. The sampling in an analog-to-digital converter is actuated by a pulse generator (clock). If the sampling rate is less than 2*fmax, some of the highest frequency components in the analog input signal will not be correctly represented in the digitized output. When such a digital signal is converted back to analog form by a digital-to analog converter, false frequency components appear that were not in the original analog signal. This undesirable condition is a form of distortion called Aliasing. Thus, the condition to avoid aliasing is:

$$f_s \geq 2f_{max}$$

Where, fs=Sampling frequency; fmax=Highest frequency present in a signal, or bandwidth.

(ii)Original signal sampled at 12 KHz

```
Fs = 12000;
t = 0:1/Fs:0.01;
x = 10*cos(2*pi*1e3*t) + 6*cos(2*pi*2e3*t) + 2*cos(2*pi*4e3*t);
stem(t,x)
grid on
title('Original Signal x(t) time-sampled at 12KHz')
xlabel('t in seconds')
ylabel('Amplitude')
```

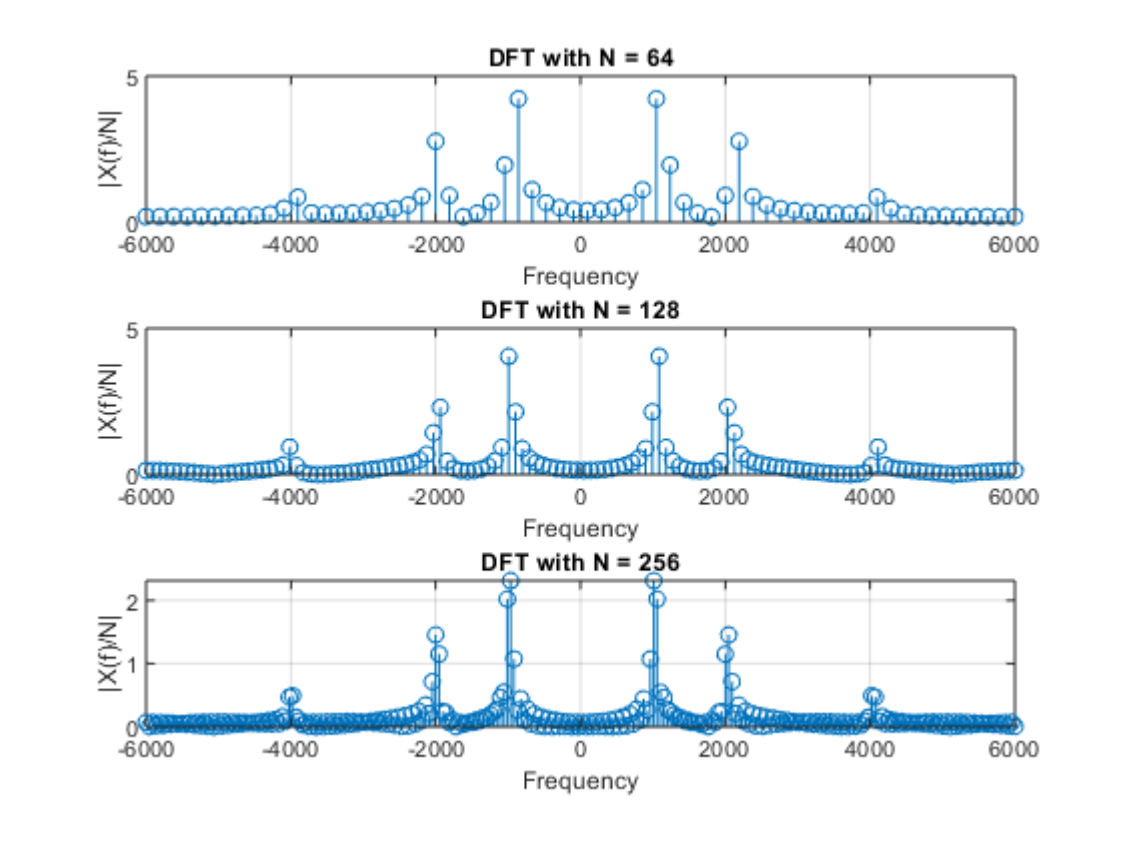


(iii)Discrete Fourier Transform at different N

```
Fs = 12000;
t = 0:1/Fs:0.01;
N=64;
x = 10*cos(2*pi*1e3*t) + 6*cos(2*pi*2e3*t) + 2*cos(2*pi*4e3*t);
y=fft(x,N);
y=fftshift(y);
m = abs(y)/N;
f = -6000:12000/(N-1):6000;
```

```
subplot(311)
stem(f,m)
grid on
title('DFT with N = 64')
xlabel('Frequency')
ylabel('|X(f)/N|')

N1 = 128;
x = 10*cos(2*pi*1e3*t) + 6*cos(2*pi*2e3*t) + 2*cos(2*pi*4e3*t);
y1=fft(x,N1);
y1=fftshift(y1);
m1 = abs(y1)/N1;
f = -6000:12000/(N1-1):6000;
subplot(312)
stem(f,m1)
grid on
title('DFT with N = 128')
xlabel('Frequency')
ylabel('|X(f)/N|')

N2 = 256;
x = 10*cos(2*pi*1e3*t) + 6*cos(2*pi*2e3*t) + 2*cos(2*pi*4e3*t);
y2=fft(x,N2);
y2=fftshift(y2);
m2 = abs(y2)/N2;
f = -6000:12000/(N2-1):6000;
subplot(313)
stem(f,m2)
grid on
title('DFT with N = 256')
xlabel('Frequency')
```

```
ylabel('|X(f)/N|')
```



(b)Sampling at below Nyquist rate and effect of aliasing

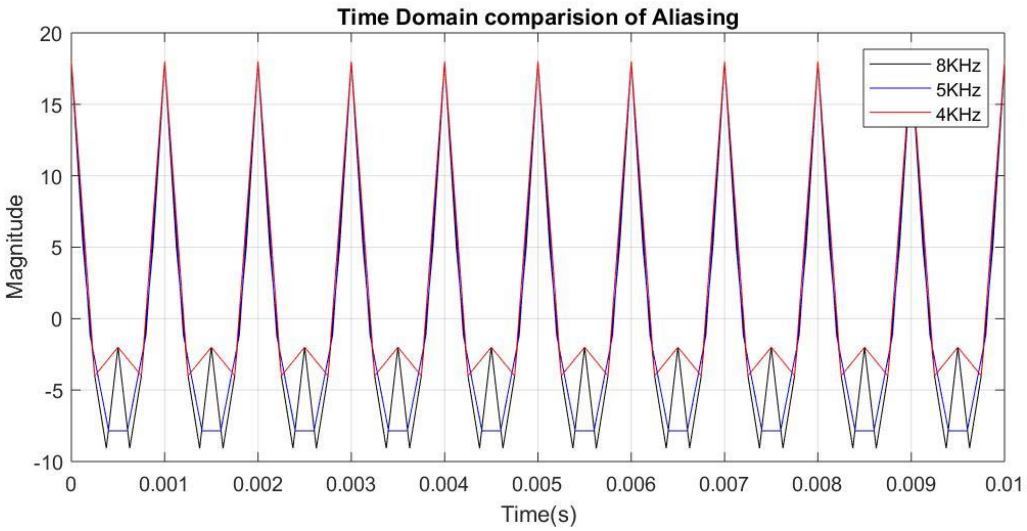
(i)Theory:

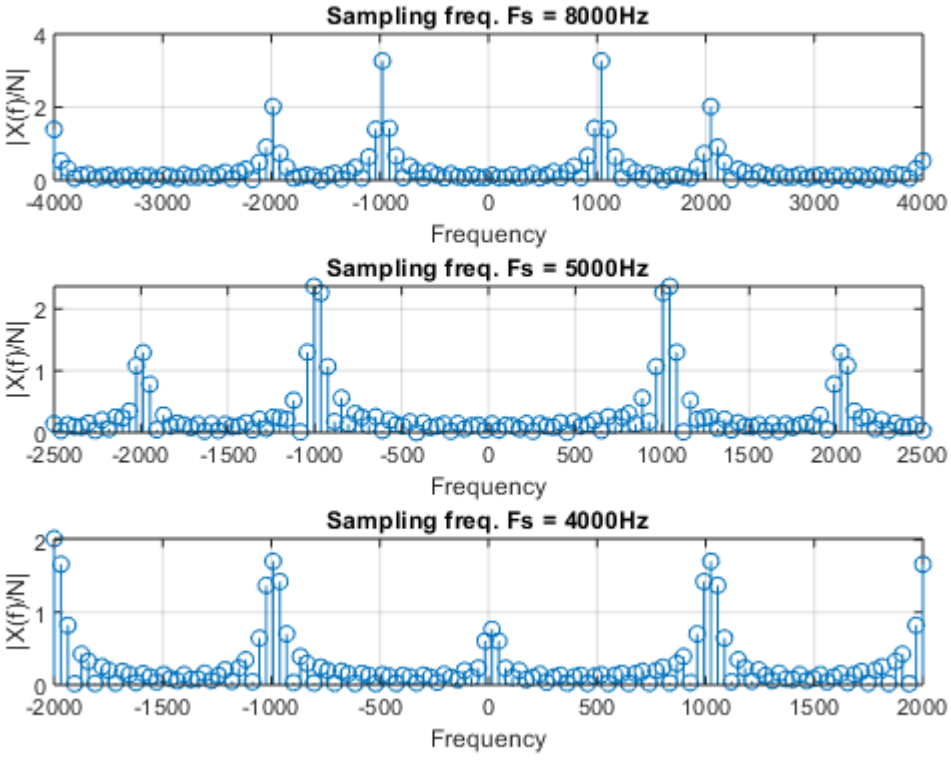
In signal processing and related disciplines, aliasing is an effect that causes different signals to become indistinguishable (or aliases of one another) when sampled. In audio, aliasing is the result of a lower resolution sampling, which translates to poor sound quality and static. This occurs when audio is sampled at a lower resolution than the original recording.

(ii)Sampling at Fs = 8 KHz, 5 KHz, 4 KHz

```
Fs1 = 8000;
N=128;
t = 0:1/Fs1:0.01;
```

```
x = 10*cos(2*pi*1e3*t) + 6*cos(2*pi*2e3*t) + 2*cos(2*pi*4e3*t);
%plot(t,x,'k')
%hold on

y=fft(x,N);
y=fftshift(y);
m = abs(y)/N;
f = -Fs1/2:Fs1/(N-1):Fs1/2;
subplot(311);
stem(f,m);
grid on
title('Sampling freq. Fs = 8000Hz')
xlabel('Frequency')
ylabel('|X(f)/N|')


Fs2 = 5000;
t = 0:1/Fs2:0.01;
x = 10*cos(2*pi*1e3*t) + 6*cos(2*pi*2e3*t) + 2*cos(2*pi*4e3*t);
%plot(t,x,'b')
%hold on

y1=fft(x,N);
y1=fftshift(y1);
m = abs(y1)/N;
f = -Fs2/2:Fs2/(N-1):Fs2/2;
subplot(312);
stem(f,m);
grid on
title('Sampling freq. Fs = 5000Hz')
xlabel('Frequency')
ylabel('|X(f)/N|')


Fs3 = 4000;
t = 0:1/Fs3:0.01;
x = 10*cos(2*pi*1e3*t) + 6*cos(2*pi*2e3*t) + 2*cos(2*pi*4e3*t);
%plot(t,x,'r')
%hold on

y2=fft(x,N);
y2=fftshift(y2);
m = abs(y2)/N;
f = -Fs3/2:Fs3/(N-1):Fs3/2;
subplot(313);
stem(f,m);
grid on
title('Sampling freq. Fs = 4000Hz')
xlabel('Frequency')
ylabel('|X(f)/N|')
```


Time Domain comparision of Aliasing
8KHz
5KHz
4KHz
Magnitude
Time(s)


Sampling freq. Fs = 8000Hz
Sampling freq. Fs = 5000Hz
Sampling freq. Fs = 4000Hz
|X(f)/N|
Frequency

(c)Spectrum of a square wave

(i)Theory

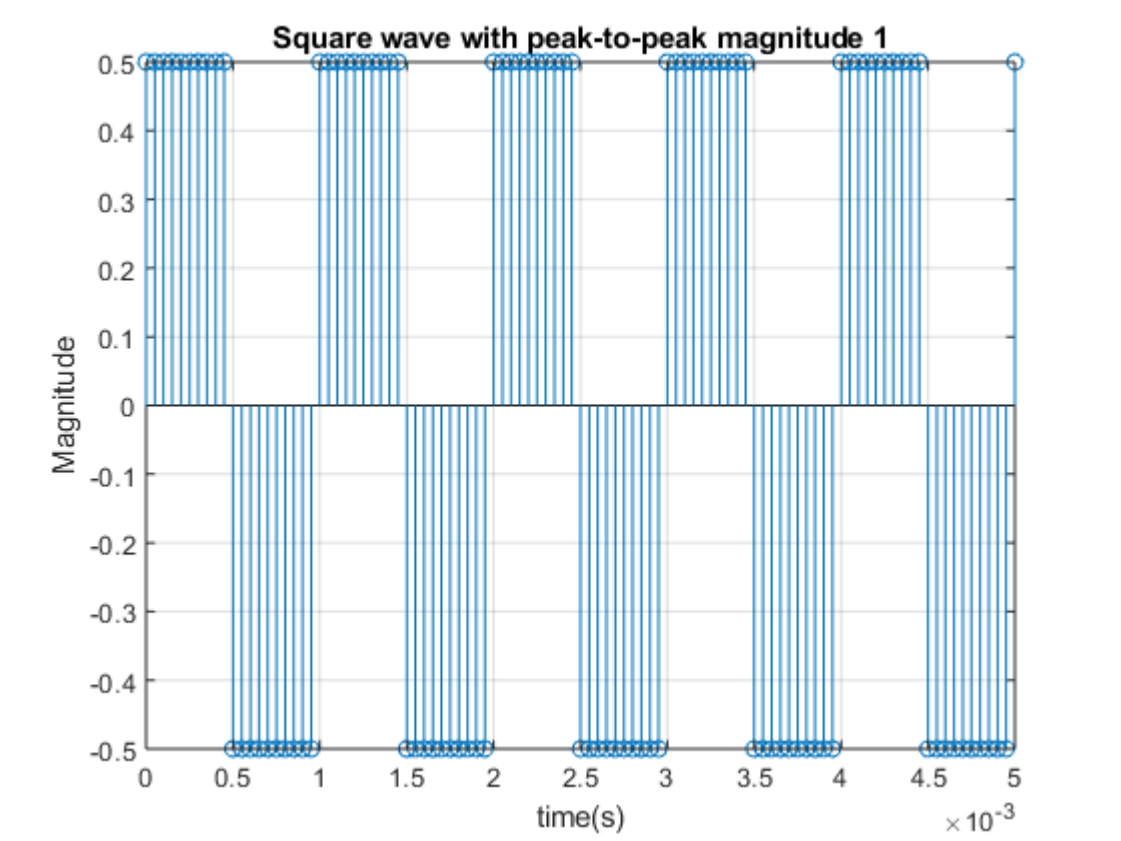
The ideal square wave contains only components of odd-integer harmonic frequencies, and it has an infinite bandwidth. A square wave has ideally infinite bandwidth. For practical purposes, the spectrum beyond 10th harmonic can be neglected.

(ii)Generation of a square wave

Sampling frequency = 20 KHz, Frequency of square wave = 1 KHz

```
F = 1000;
Fs = 20000;
T = 0:1/Fs:5/1000;
x=0.5*(square(2*pi*F*T));
stem(T,x);
title('Square wave with peak-to-peak magnitude 1')
xlabel('time(s)')
ylabel('Magnitude')
```

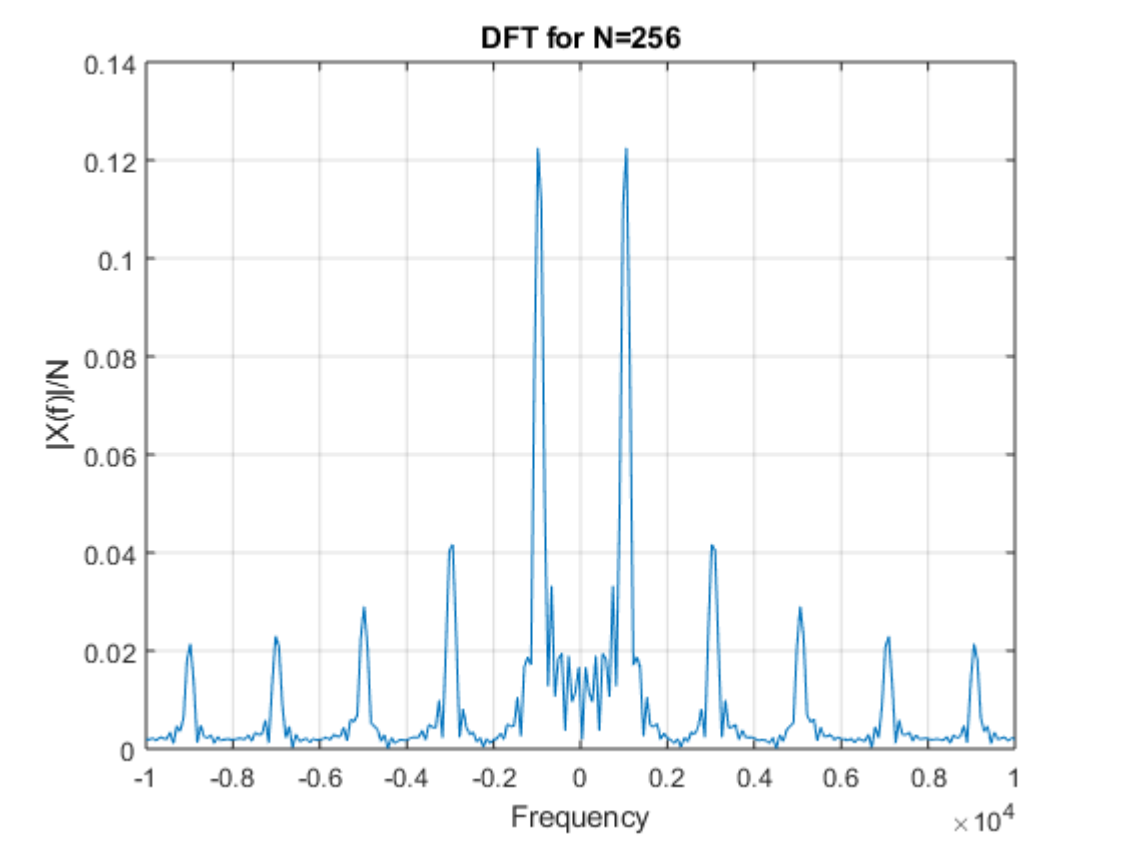
```
grid on
```



(iii)Taking DFT of sampled square wave with N=256

```
F = 1000;
Fs = 20000;
T = 0:1/Fs:5/1000;
x=0.5*(square(2*pi*F*T));
n=256;
y=fft(x,n);
y=fftshift(y);
m = abs(y)/n;
f1 = -Fs/2:Fs/(n-1):Fs/2;
plot(f1,m)
title('DFT for N=256')
xlabel('Frequency')
ylabel('|X(f)|/N')
```

```
grid on
```



(d)Interpolation and upsampling

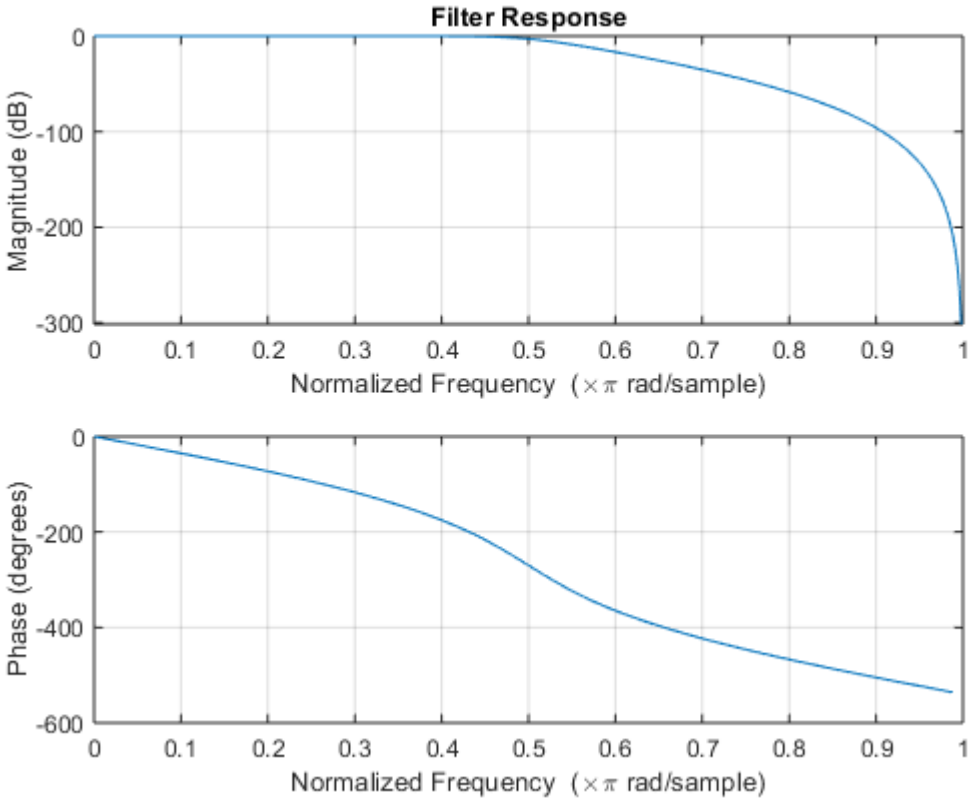
(i)Theory:

If an analog signal is sampled at a frequency higher than the Nyquist rate (Fs>2Fmax) it is possible to interpolate the intermediate L-1 samples or in other words to obtain the samples at Fs2=LFs1

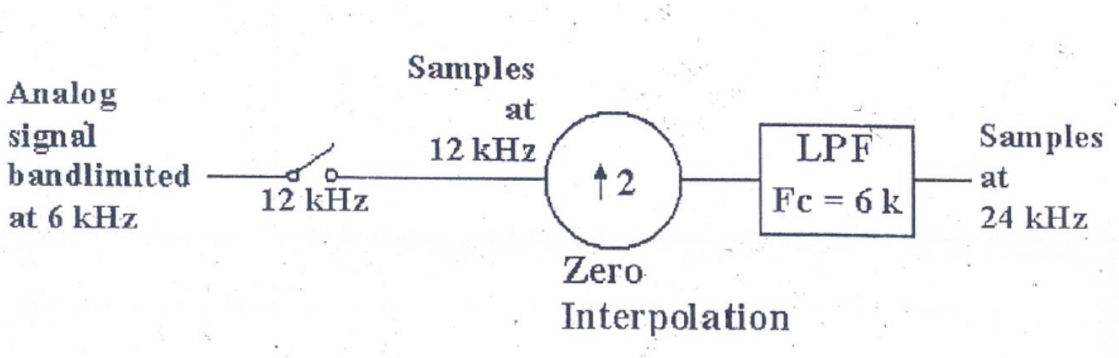
frequency. This can be simply done by passing the sampled signals through an ideal low pass filter of cut-off frequency F max and sampling it again at a higher rate. But in discrete domain this is achieved. “Interpolation”, is the process of upsampling followed by filtering. The filtering removes the undesired spectral images. “Upsampling” is the process of inserting zero-valued samples between original samples to increase the sampling rate. This is called “zero-stuffing”.

(ii)Butterworth Filter Response

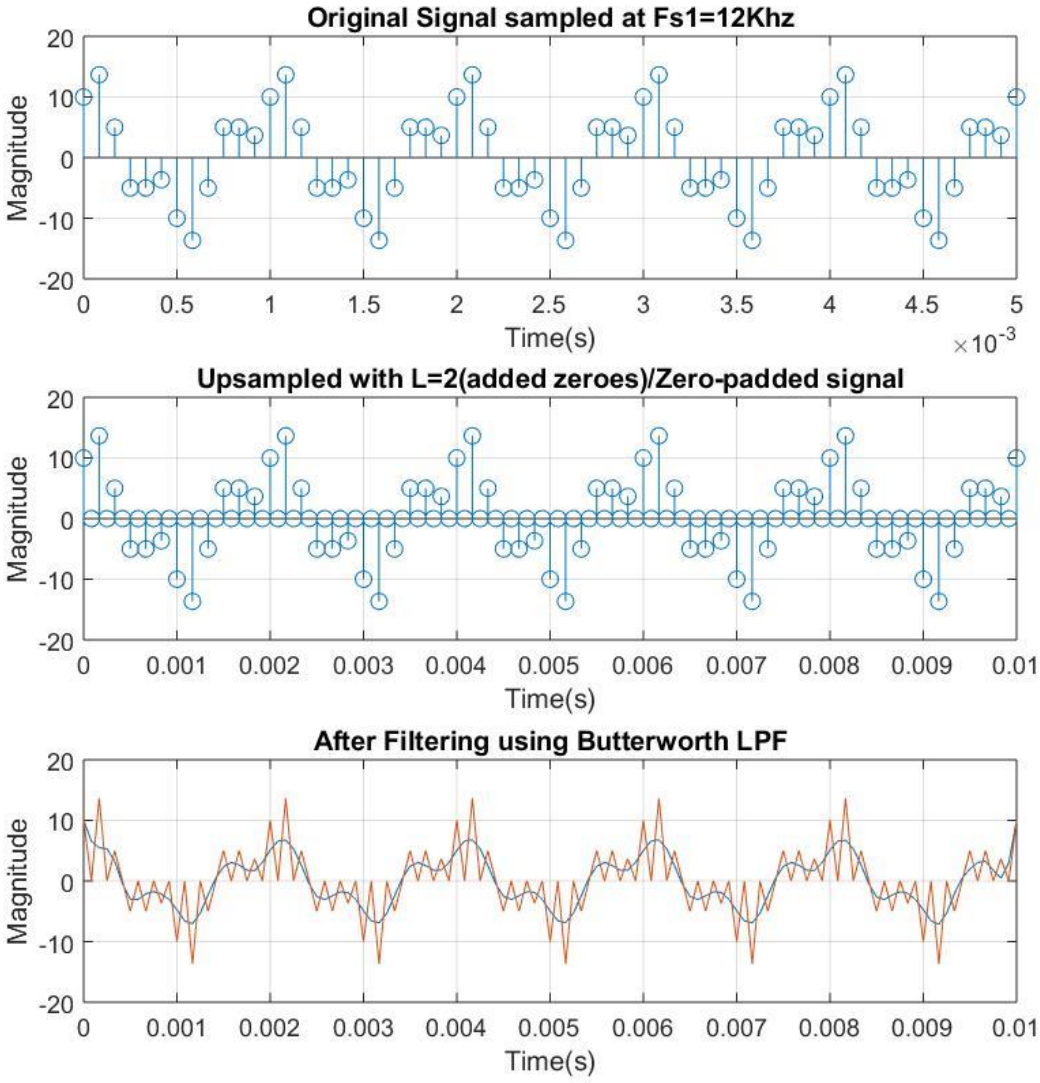
```
figure
%Implementing a butterworth Filter on matlab.
[b,a] = butter(6,(6000/(24000/2)), 'low' ); %sixth order butterworth filter
freqz(b,a);
title( 'Filter Response' );
```



(iii)Upsampling steps




```
Fs1 = 12000;
t1 = 0:1/Fs1:0.005;
x1 = 10*cos(2*pi*1e3*t1) + 5*sin(2*pi*3e3*t1) ;
subplot(311);
stem(t1,x1);
grid on
title('Original Signal sampled at Fs1=12Khz');
xlabel('Time(s)')
ylabel('Magnitude')

t2 = 0:1/Fs1:0.01;
x2 = upsample(x1,2);  %L = 2
x2 = x2(1:end-1);
subplot(312);
stem(t2,x2);
grid on
title('Upsampled with L=2(added zeroes)/Zero-padded signal')
xlabel('Time(s)')
ylabel('Magnitude')

Fs1 = 12000;
t1 = 0:1/Fs1:0.005;
x1 = 10*cos(2*pi*1e3*t1) + 5*sin(2*pi*3e3*t1) ;

t2 = 0:1/Fs1:0.01;
x2 = upsample(x1,2);  %L = 2
x2 = x2(1:end-1);

Fs2 = 24000;
t = 0:1/Fs2:0.01;
n=33;
fc=6000;
Wn = fc/Fs1;   %Normalised cutoff freq

[b,a] = butter(n,Wn);
y2 = filter(b,a,x2);
y3 = filtfilt(b,a,x2);
subplot(313);
grid on
plot(t2,y3,t2,x2);
title('After Filtering using Butterworth LPF')
grid on
xlabel('Time(s)')
ylabel('Magnitude')
```



Discussions:

(a)

i. The sampling frequency is above Nyquist Rate and hence proper DFT is obtainable.

ii. It can be clearly observed that the Discrete Fourier Transform is more prominent at the constituent frequencies.

iii. As N increases the sharpness at constituent frequencies increase and hence larger value of N is preferable to get better understanding of the composing signal frequencies.

(b)

i. When sampled at frequencies lower than the Nyquist Rate we observe aliasing.
ii. It can be clearly seen when Fs = 4000Hz. The 4 kHz present in the original signal can be seen as a DC component (Frequency = 0).
iii. The signal can't be reconstructed from this sampled set as we unwanted frequency components are present and the reconstructed signal would differ a lot from the original signal.

(c)

i. Square wave is composed of odd-integer harmonic frequencies and has infinite bandwidth.
ii. The DFT shows peaks at the odd multiples of 1 KHz only. The amplitude decreases with increase in harmonics.
iii. Higher harmonics are neglected for practical purposes as their amplitudes is negligible.

(d)

i. To regenerate the sampled signal we upsample it, add zeroes and then interpolate it. Finally it is passed through LPF.
ii. We have used a Butterworth filter here to generate the upsampled signal. We see that the Filtered signal do not exactly matches with the original upsampled signal due to scaling variations and phase shift (phase shift was resolved by filtfilt).
iii. We can obtain the interpolated signal by removing the high frequency content by passing it through a digital LPF of cut-off π/L.

References:

-Alan V. Oppenheim et.al Discrete Time Signal Processing.
-Wikipedia.

Published with MATLAB® R2018a

Digital Signal Processing Lab

Expt. No. 2
Designing of Low Pass Filters by Windowing Method

By Pranit Dalal
Roll no. 16EC10016
Group 22 (Tuesday)

AIM:

(a)Design FIR filters for various orders and cut-off frequencies.
(b)Check attenuation of pass band frequencies and compare it with stop band frequencies
(c)Response of FIR filters when signals are contaminated by noise.

(a) Design FIR filters for various orders and cut-off frequencies:

(i)Theory:

In digital signal processing, ideal filters having sharp cut-off are not realizable in practice since their impulse responses extend up to infinity. In order to realize filters practically, we need to truncate the impulse response after a few samples. But this abrupt transition introduces many undesired features in the frequency domain (e.g. a large amount of power lies in the side-lobes). By windowing method, the truncated filter is modified to satisfy certain frequency domain requirements and to perform its task as well.
A few window functions we implemented are:

(1) Rectangular window : $w(n) = 1 \quad ; \quad n=0,1,....,N-1$

$= 0 \quad otherwise$

(2) Triangular window : $w(n) = 1 - 2*[n - (N-1)/2]\ /(N-1) \quad ; \ n=0,1,....,N-1$

$= 0 \quad otherwise$

(3) Hanning window: : $w(n) = 0.5 - 0.5 *cos\ [2\pi n / (N-1)] \quad ; n=0,1,... \ N-1$

$= 0 \quad otherwise$

(4) Hamming window : $w(n) = 0.54 - 0.46 *cos\ [2\pi n / (N-1)] \quad ; n=0,1,... \ N-1$

$= 0 \quad otherwise$

(5) Blackmnan window : $w(n) = 0.42 - 0.5 *cos\ [2\pi n / (N-1)]$

$+ 0.08 *cos\ [4\pi n / (N-1)] \quad ; \ n=0,1,... \ N-1$

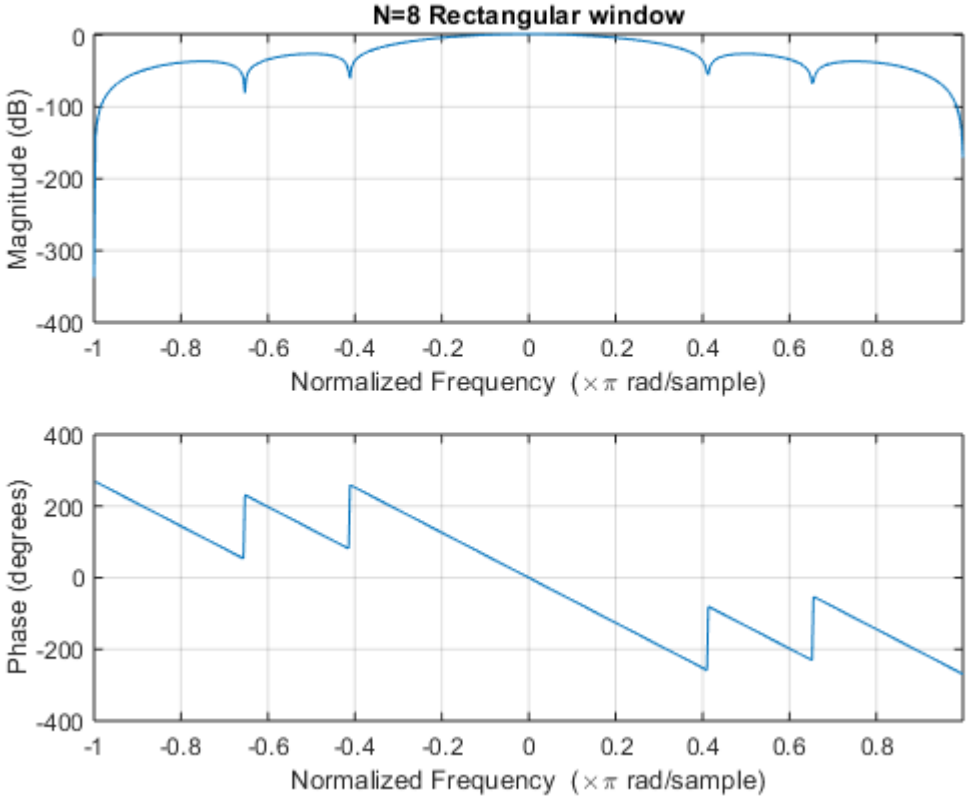
$=0 \quad otherwise$

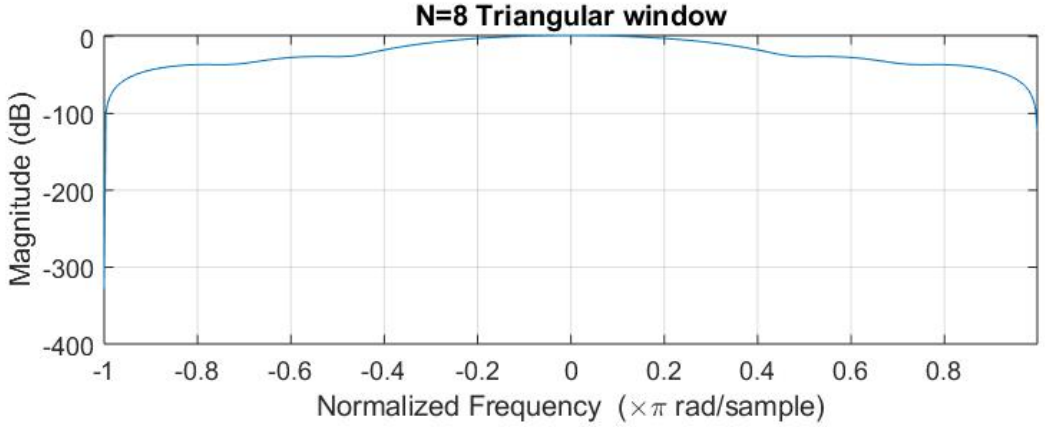
(ii) Code:

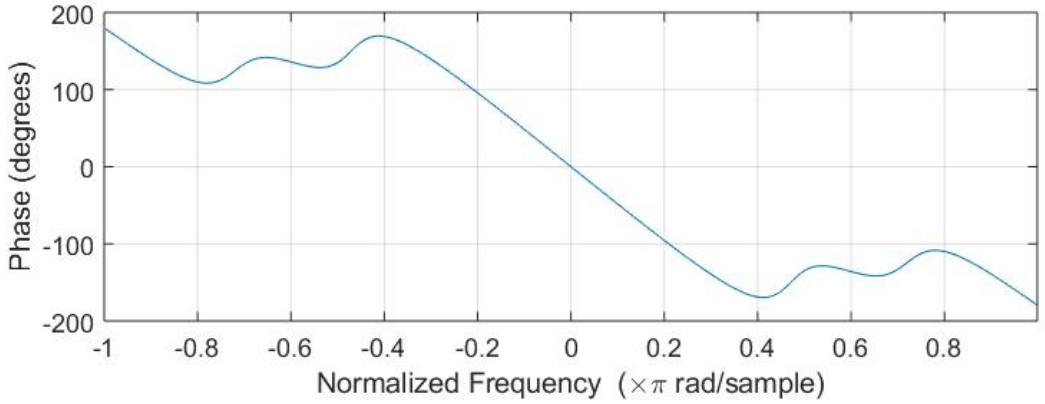
Frequency response of different windows

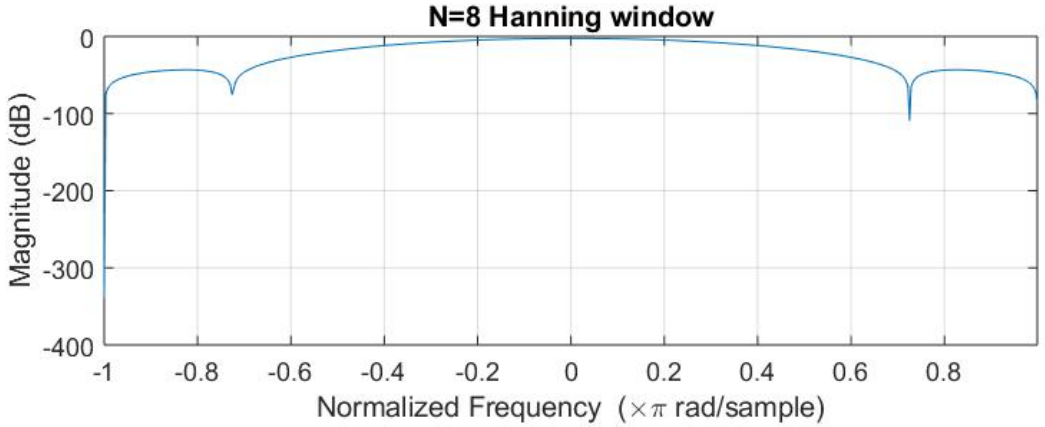
```
N =8;
wc = pi/4;           %cut-off frequency
k = (N-1)/2;
n = 0:1:N-1;
hd = (sin(wc*(n-k)))./(pi*(n-k));        %impulse response
w1 = (n>=0)-(n>=N);                                  %rectangular window
%w2 = ((n>=0)-(n>=N)).*(1-2*(n-(N-1)/2)/(N-1));       %triangular window
%w3 = ((n>=0)-(n>=N)).*(0.5-0.5*cos((2*pi*n)/(N-1)));  %hanning window
%w4 = ((n>=0)-(n>=N)).*(0.54-0.46*cos((2*pi*n)/(N-1)));%hamming window
%w5 = ((n>=0)-(n>=N)).*(0.42-0.5*cos((2*pi*n)/(N-1))+0.08*cos((4*pi*n)/(N-1)));%blackman wind

h = hd.*w1;
c = -pi:0.01:pi;
[h1,w] = freqz(h,1,c);
h2 = abs(h1);
figure, freqz(h,1,c);
title('N=8 Rectangular window');
```

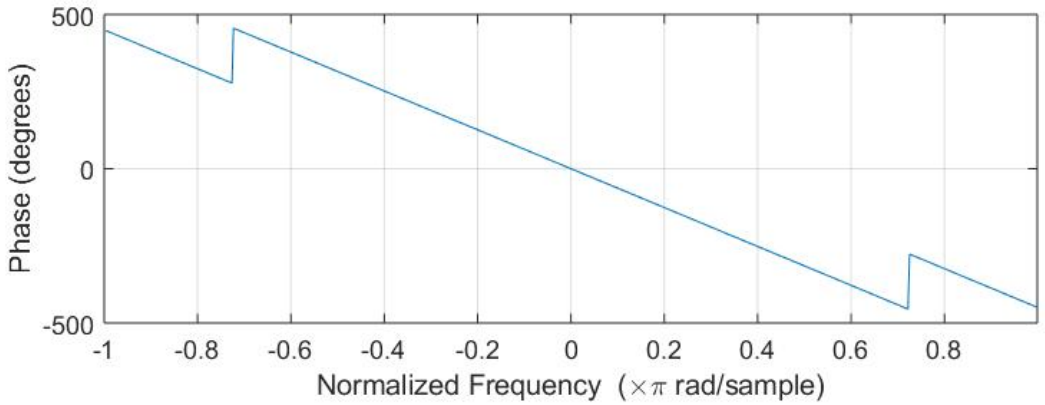
For N = 8;

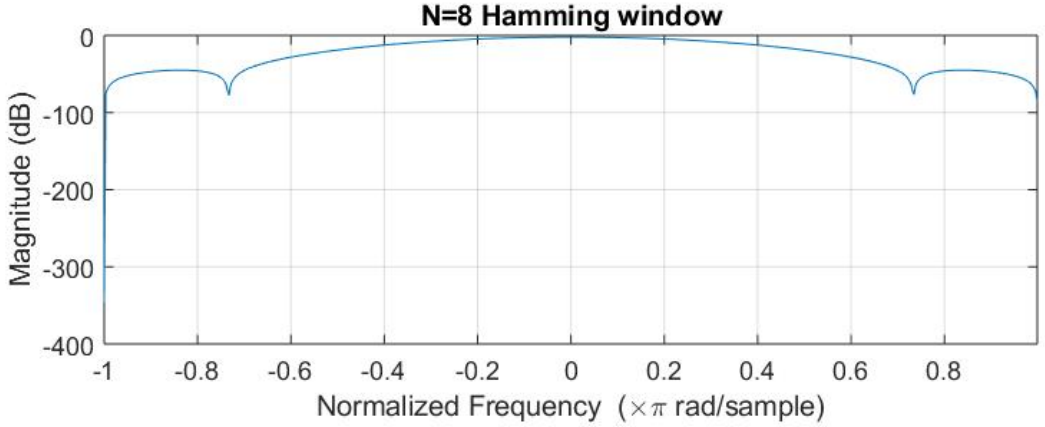

N=8 Rectangular window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)
Phase (degrees)
400
200
0
-200
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

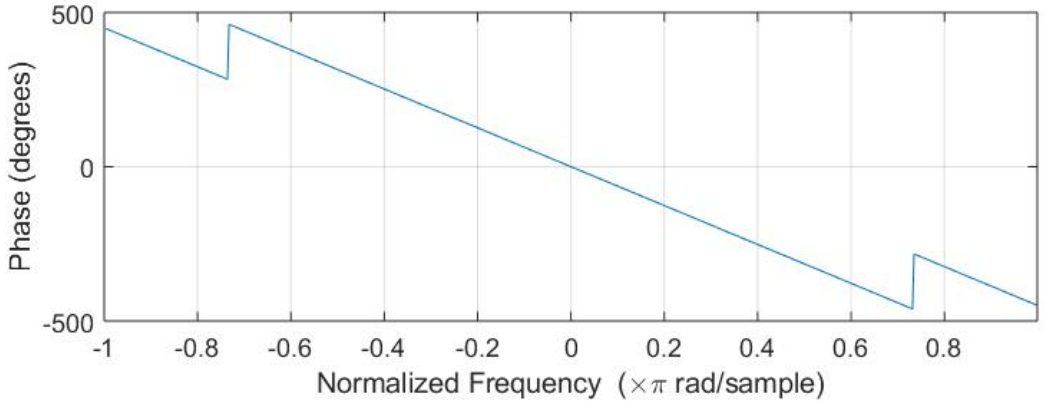

N=8 Triangular window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

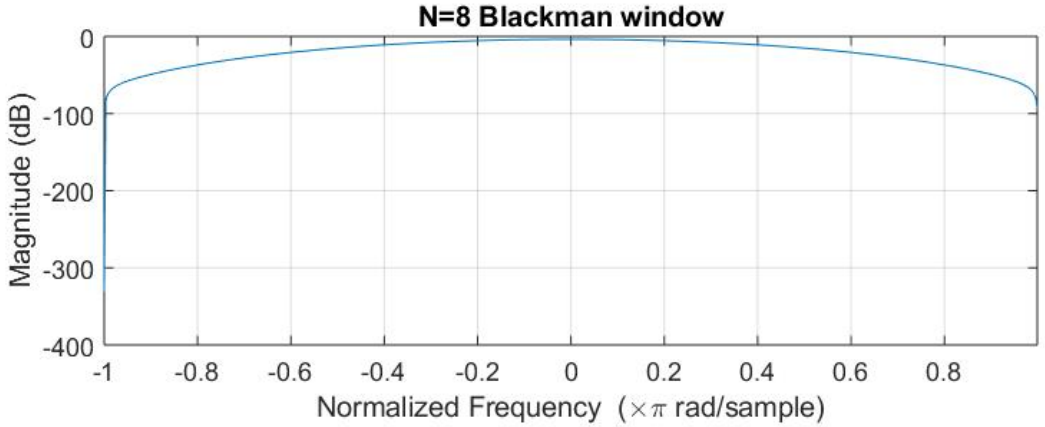

Phase (degrees)
200
100
0
-100
-200
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

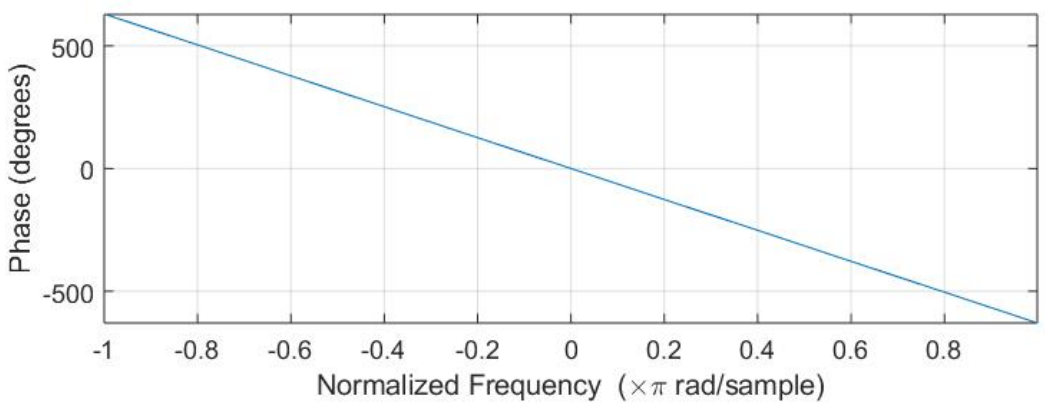

N=8 Hanning window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)


Phase (degrees)
500
0
-500
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

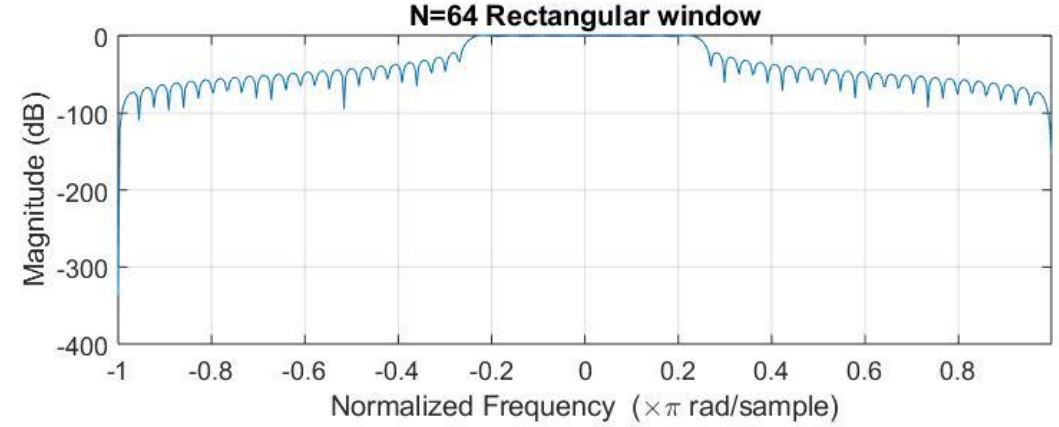

N=8 Hamming window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

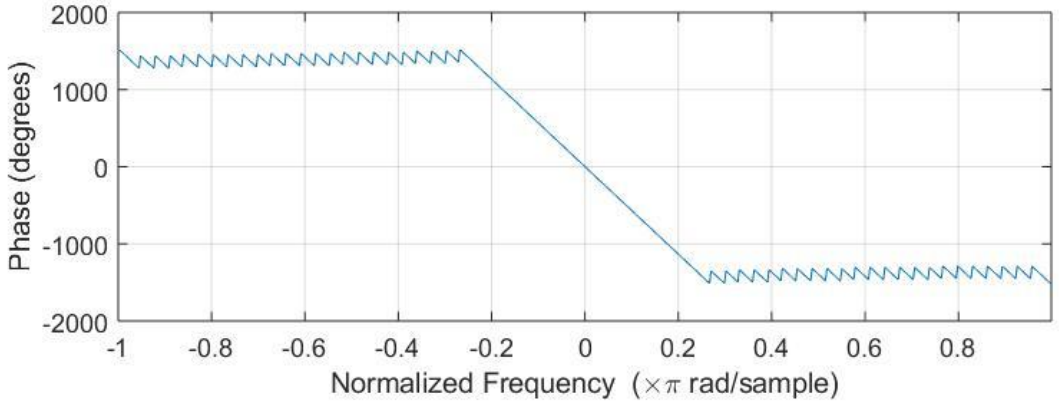

Phase (degrees)
500
0
-500
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

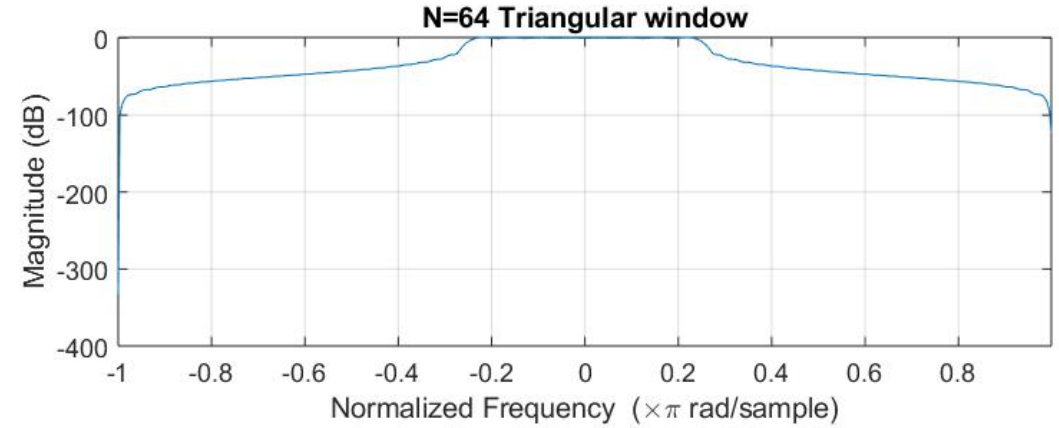

N=8 Blackman window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

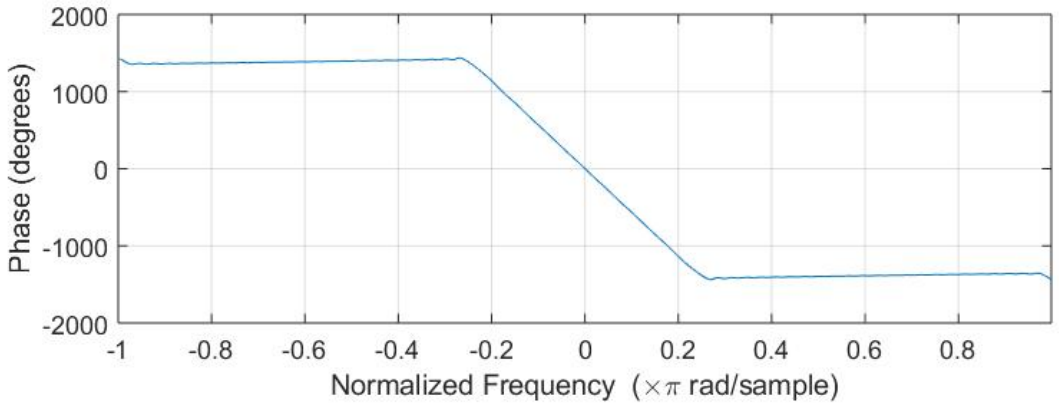

Phase (degrees)
500
0
-500
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

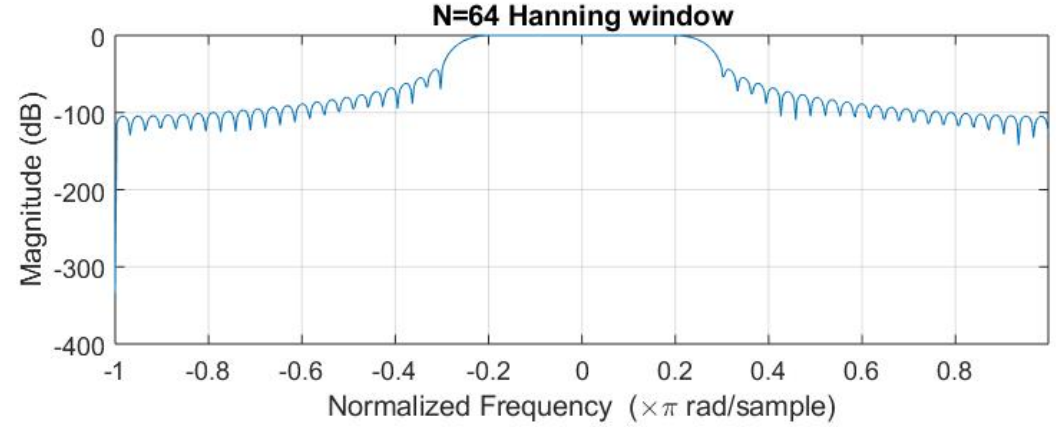
For N = 64:

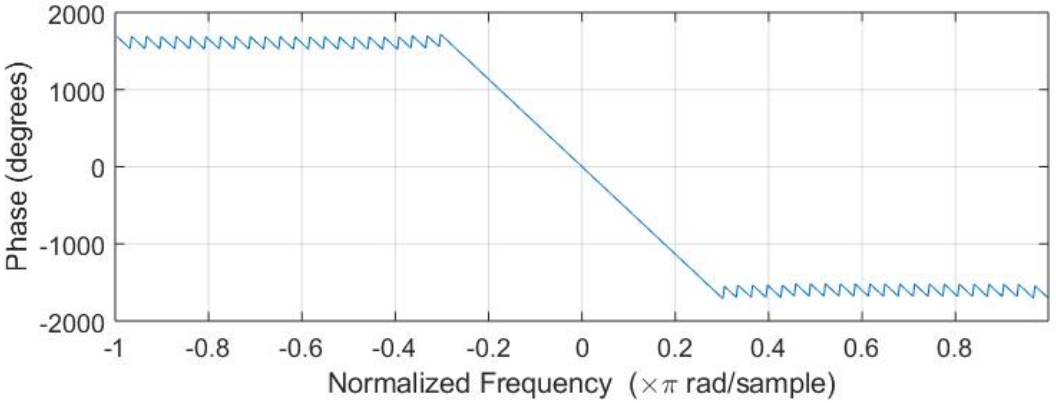

N=64 Rectangular window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

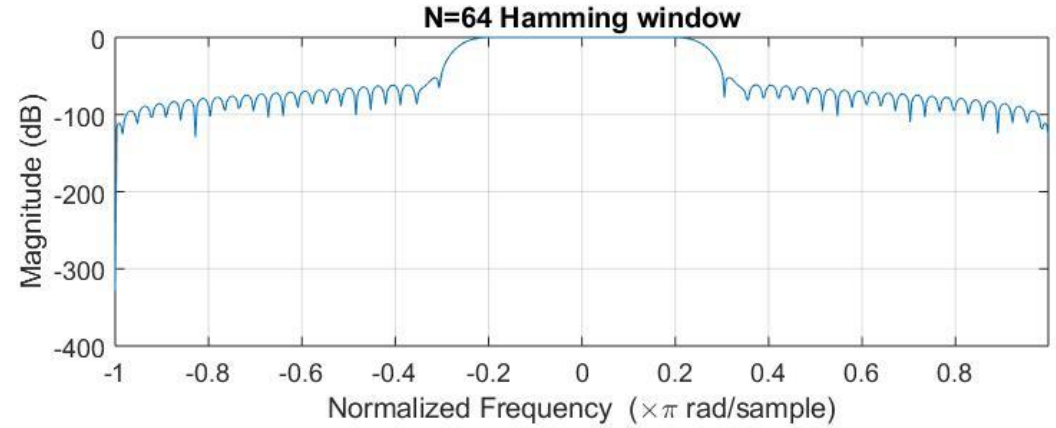

Phase (degrees)
2000
1000
0
-1000
-2000
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

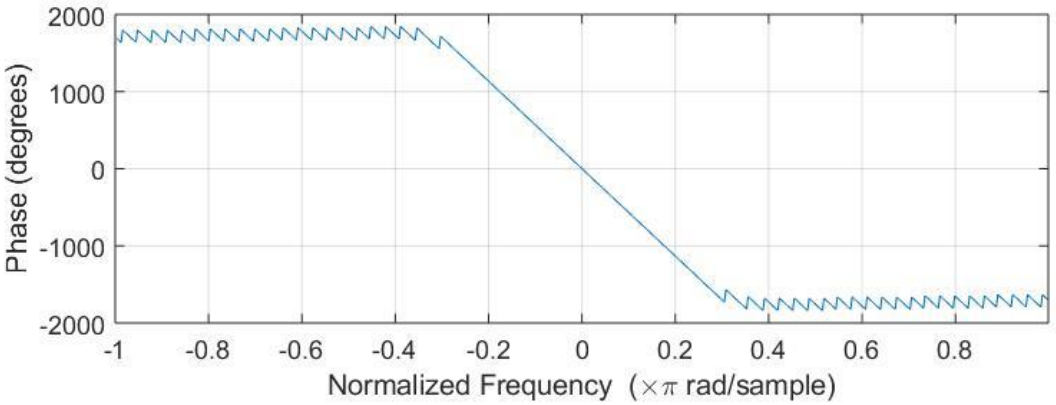

N=64 Triangular window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

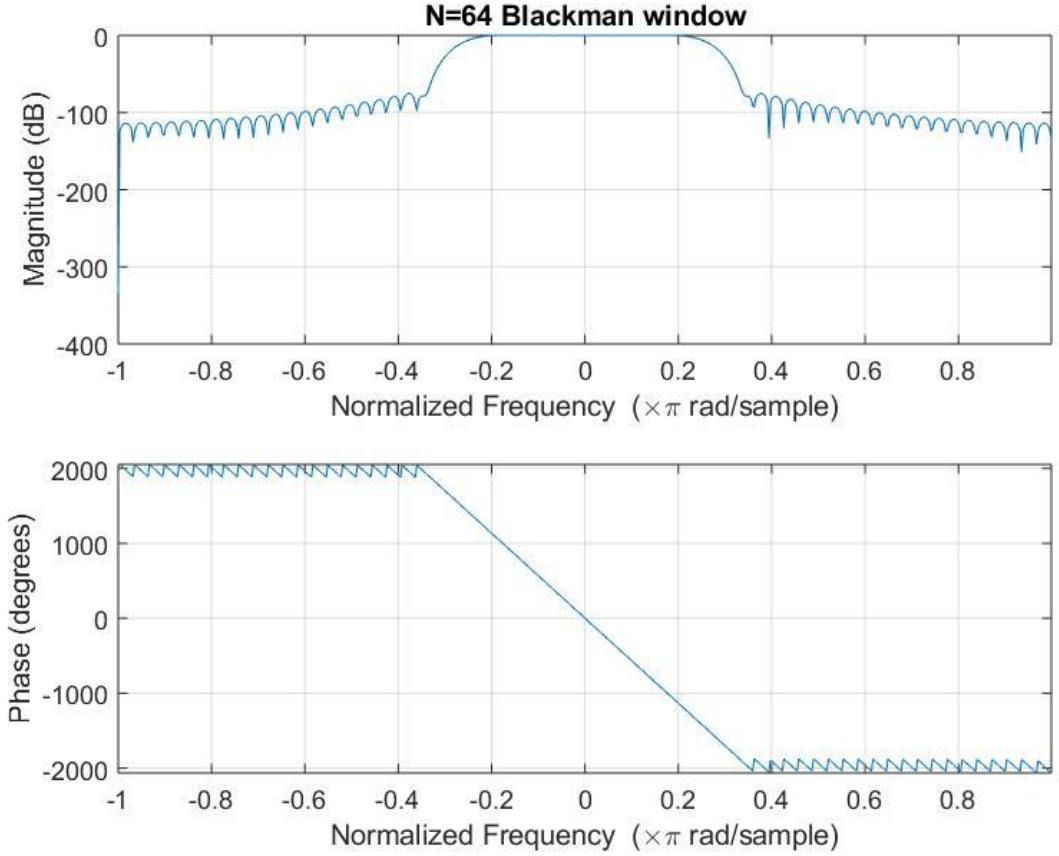

Phase (degrees)
2000
1000
0
-1000
-2000
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)


N=64 Hanning window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

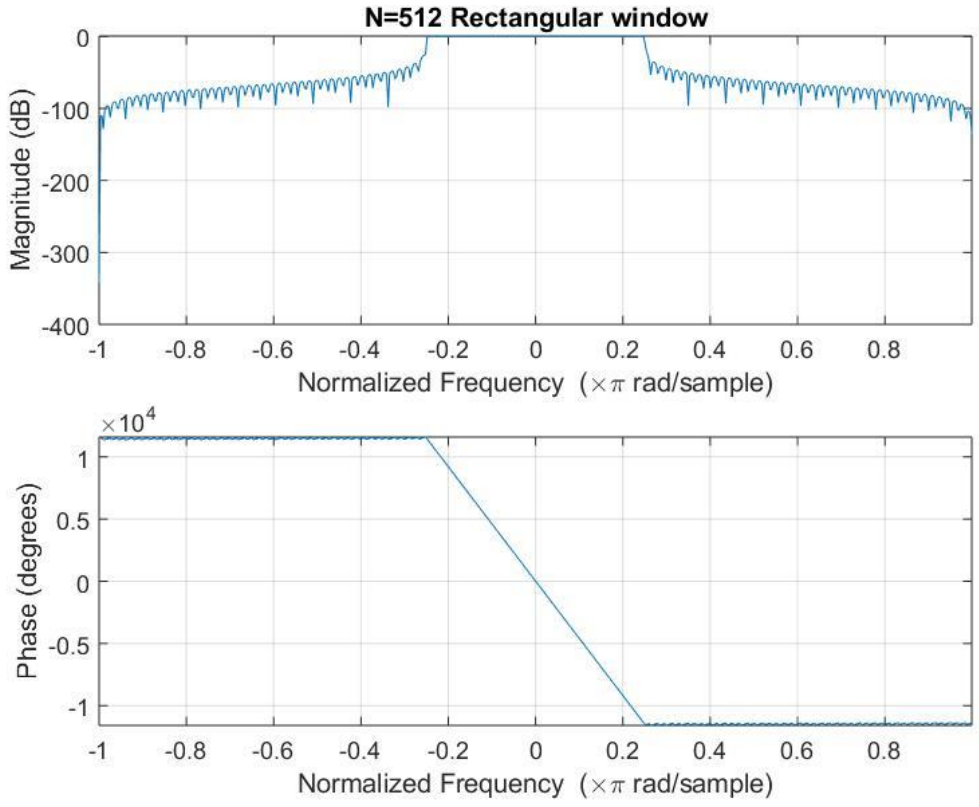

Phase (degrees)
2000
1000
0
-1000
-2000
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

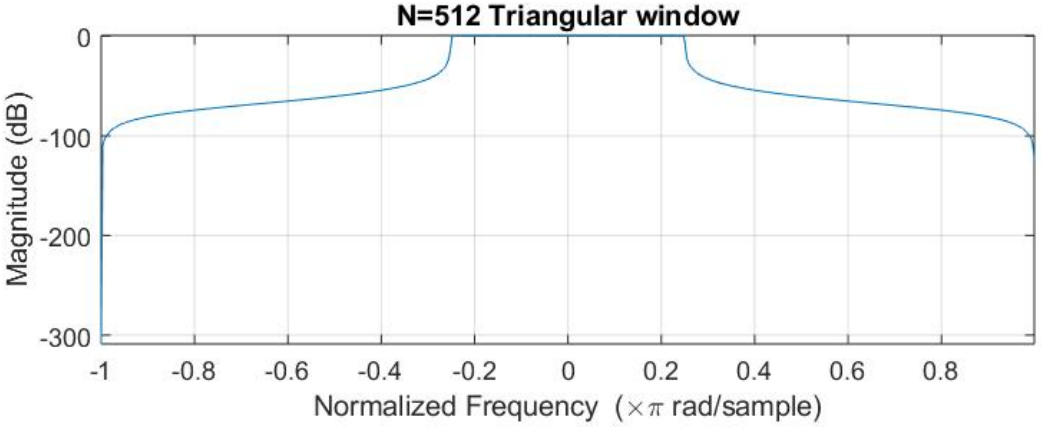

N=64 Hamming window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

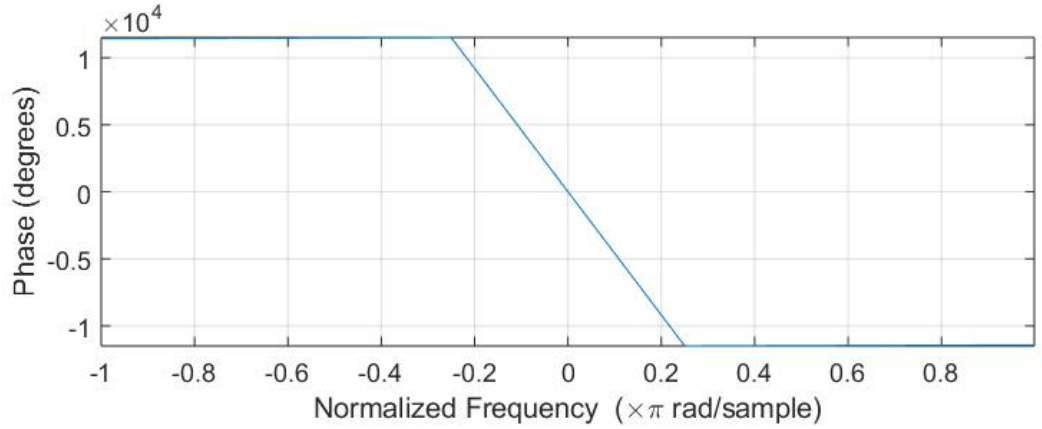

Phase (degrees)
2000
1000
0
-1000
-2000
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)

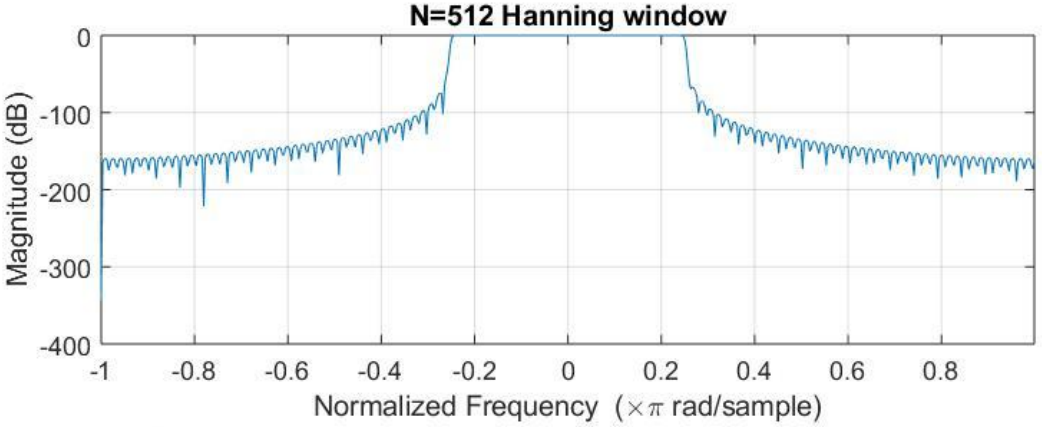

N=64 Blackman window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)
Phase (degrees)
2000
1000
0
-1000
-2000
Normalized Frequency (×π rad/sample)

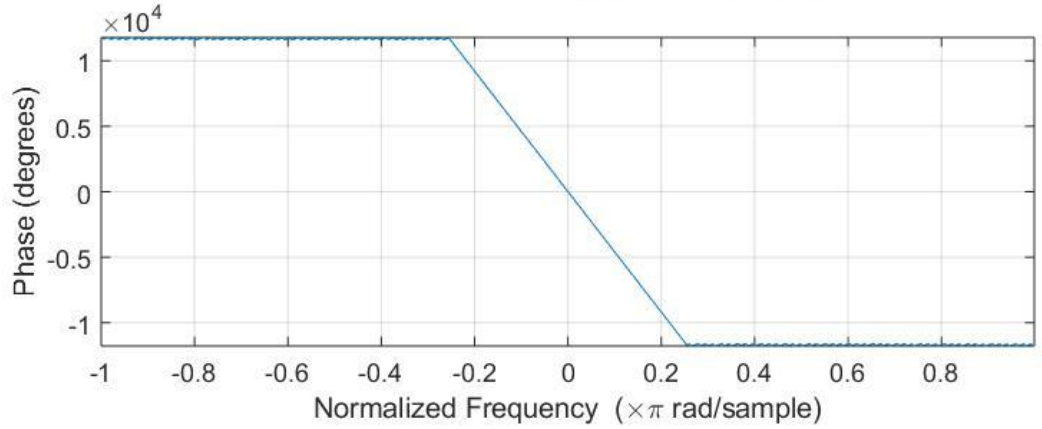
For N = 512:

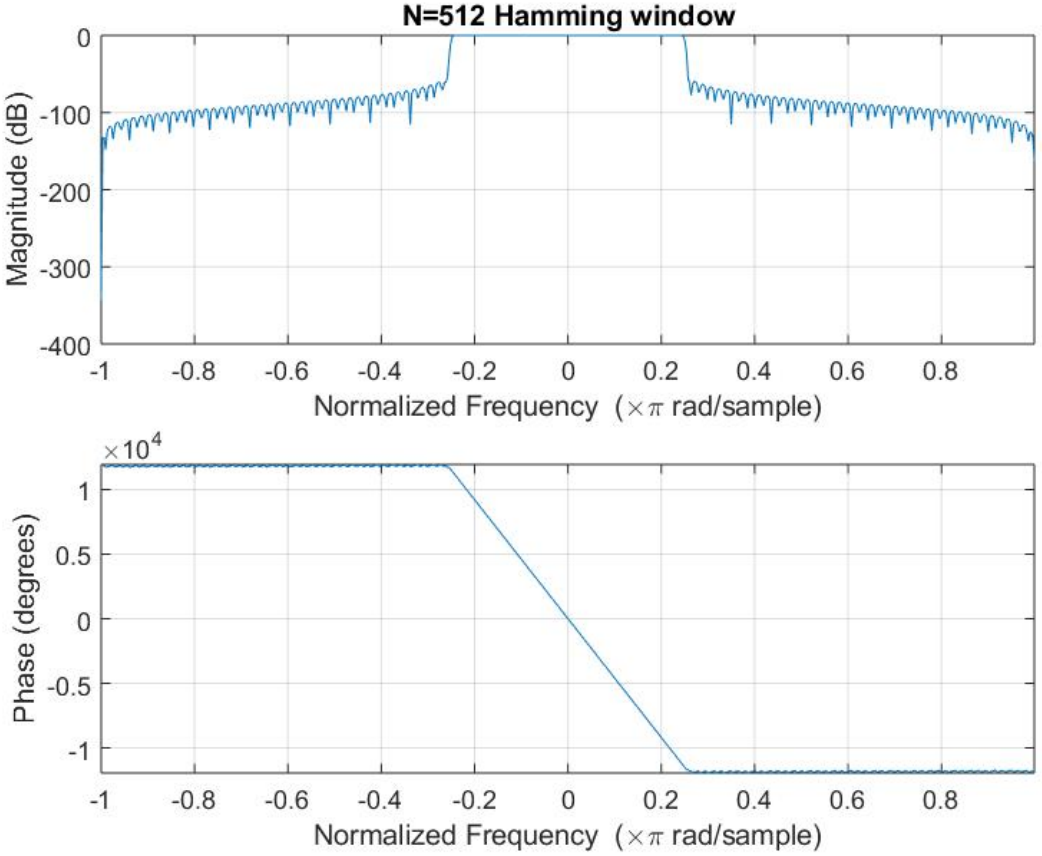

N=512 Rectangular window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)
×10^4
Phase (degrees)
1
0.5
0
-0.5
-1
Normalized Frequency (×π rad/sample)

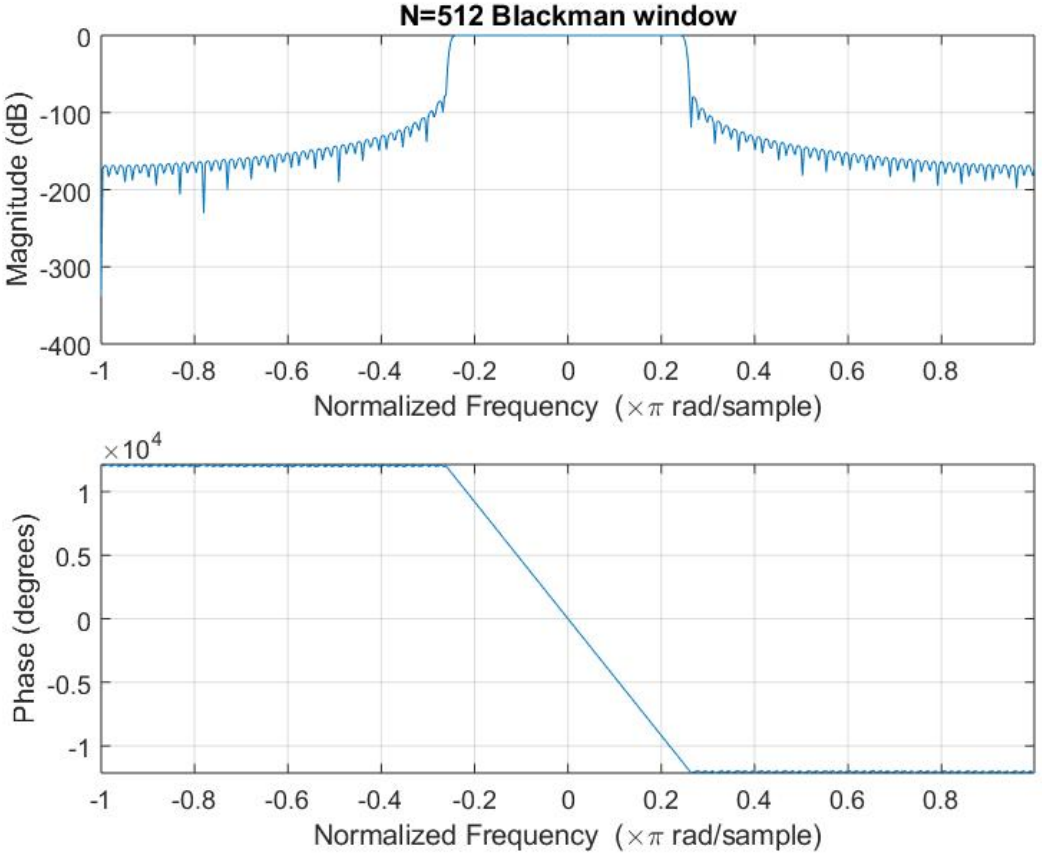

N=512 Triangular window
Magnitude (dB)
0
-100
-200
-300
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)


×10^4
Phase (degrees)
1
0.5
0
-0.5
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)


N=512 Hanning window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)


×10^4
Phase (degrees)
1
0.5
0
-0.5
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)


N=512 Hamming window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)
×10^4
Phase (degrees)
1
0.5
0
-0.5
-1
Normalized Frequency (×π rad/sample)


N=512 Blackman window
Magnitude (dB)
0
-100
-200
-300
-400
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
Normalized Frequency (×π rad/sample)
×10^4
Phase (degrees)
1
0.5
0
-0.5
-1
Normalized Frequency (×π rad/sample)

(iii) Results:

N	Rectangular Window		
	Transition width π rad/sample	Peak of first lobe (dB)	Maximum Stopband Attenuation(dB)
8	0.2217	-26.05	-55.02
64	0.0460	-21.75	-39.55
512	0.0145	-35.10	-53.55

N	Triangular Window		
	Transition width π rad/sample	Peak of first lobe (dB)	Maximum Stopband Attenuation(dB)
8	0.2323	-25.86	-26.04
64	0.0268	-21.53	-21.60
512	0.0402	-43.41	-43.41

N	Hanning Window		
	Transition width π rad/sample	Peak of first lobe (dB)	Maximum Stopband Attenuation(dB)
8	0.6300	-43.28	-109.1
64	0.0644	-44.02	-54.93
512	0.0140	-66.64	-69.7

N	Hamming Window		
	Transition width π rad/sample	Peak of first lobe (dB)	Maximum Stopband Attenuation(dB)
8	0.6096	-45.06	-71.76
64	0.0664	-52.07	-77.63
512	0.0148	-59.46	-77.52

N	Blackman Window

	Transition width π rad/sample	Peak of first lobe (dB)	Maximum Stopband Attenuation(dB)
8	0.7010	-62.61	-62.61
64	0.1240	-75.44	-90.96
512	0.0158	-79.77	-119.5

(b)Check attenuation of pass band frequencies and compare it with stop band frequencies

(i)Theory:

When a signal containing 2 frequencies, one in pass-band and another in the stop-band, we observe the attenuation of the frequency component in the stop-band.

In Fourier Domain the stop-band frequencies don't pass and hence the amplitude is nearly equal to zero. Certain abnormalities are observed at the start and end in time domain when passed through the filter.

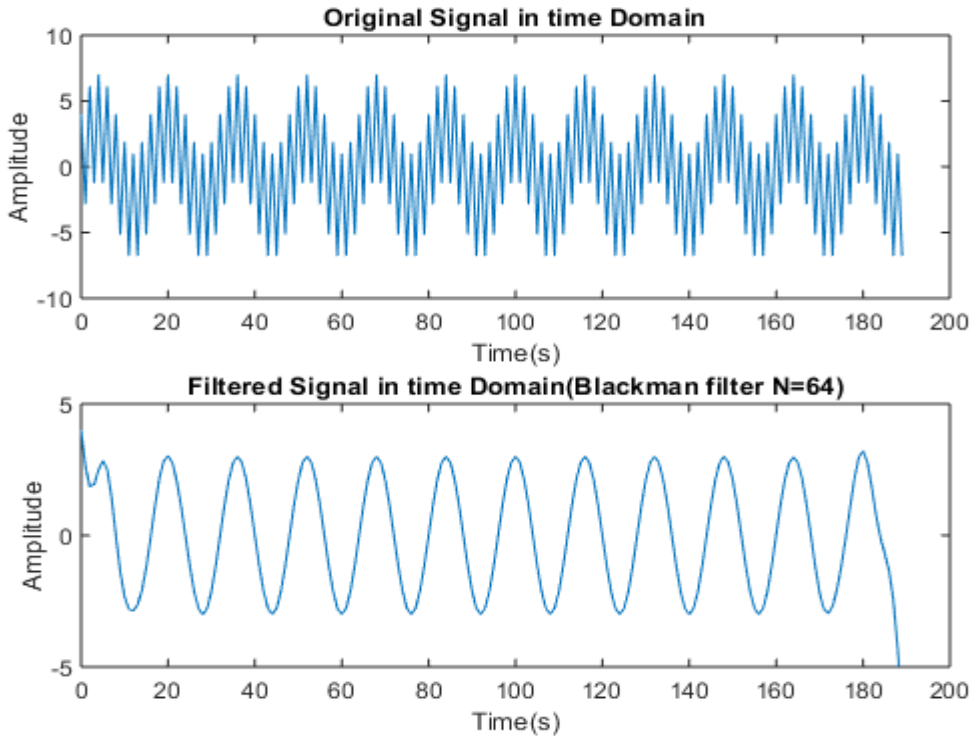
(ii)Code:

```
N = 64;
wc = pi/3;
k = (N-1)/2;
n = 0:1:N-1;
hd = (sin(wc*(n-k)))./(pi*(n-k));
w5 = ((n>=0)-(n>=N)).*(0.42-0.5*cos((2*pi*n)/(N-1))+0.08*cos((4*pi*n)/(N-1)));
h = hd.*w5;  %For Blackman Window
c = -pi:0.01:pi;
[h1,w] = freqz(h,1,c);
h1 = abs(h1);

t = 0:1:3*(N-1);
x = 3*sin(pi/8*t)+4*cos(pi*t);
grid on

y = filtfilt(h,1,x);
subplot(211)
plot(t,x)
title('Original Signal in time Domain');
xlabel('Time(s)');
ylabel('Amplitude');

subplot(212);
plot(t,y)
ylim([-5 5]);
title('Filtered Signal in time Domain(Blackman filter N=64)');
```

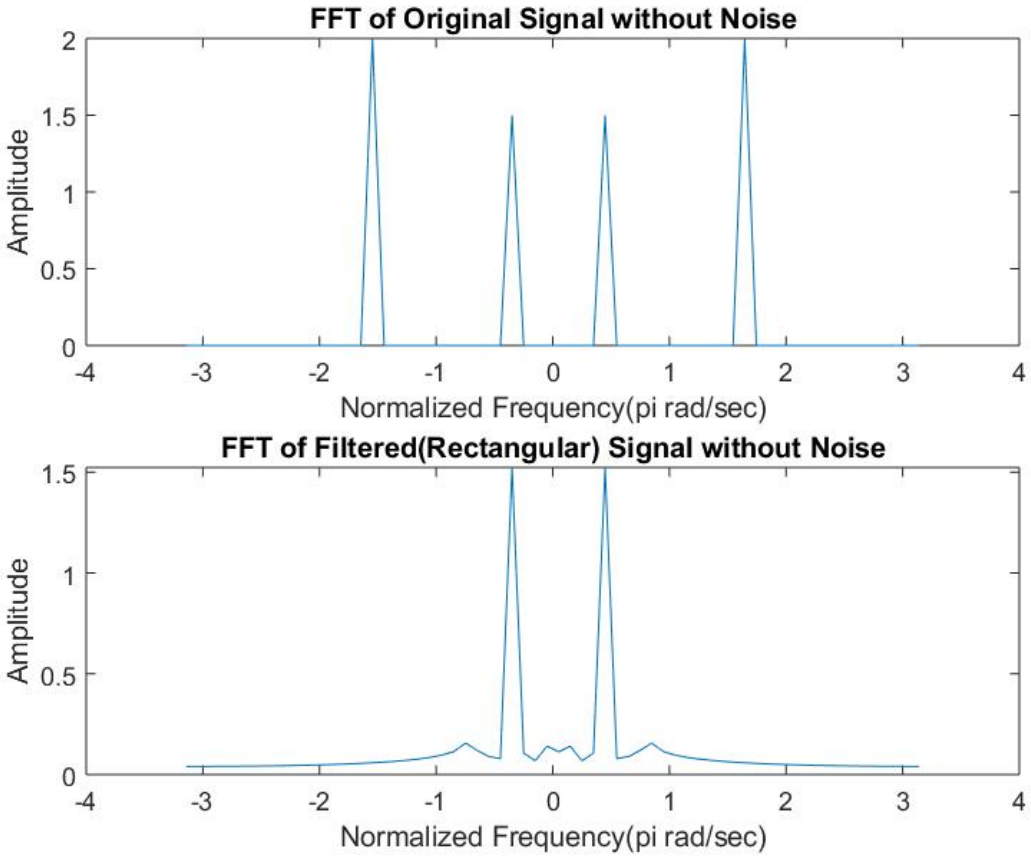
```
xlabel('Time(s)');
ylabel('Amplitude');
```

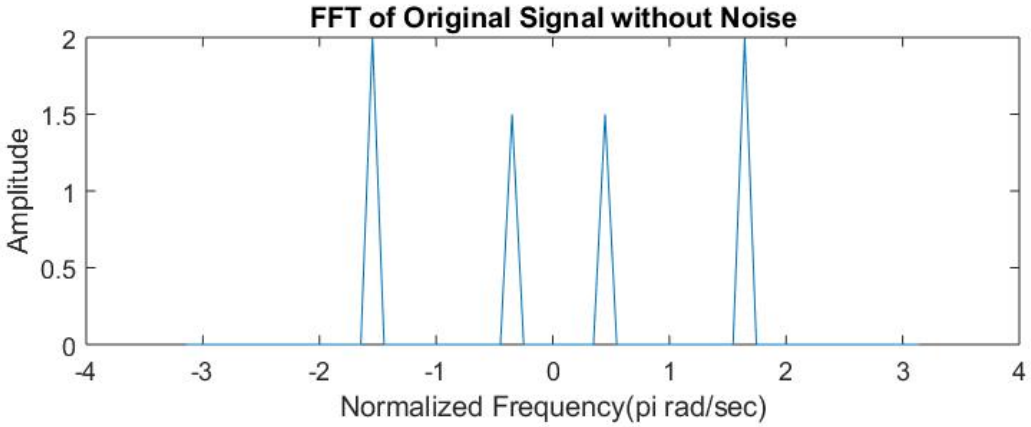
(iii)Results:

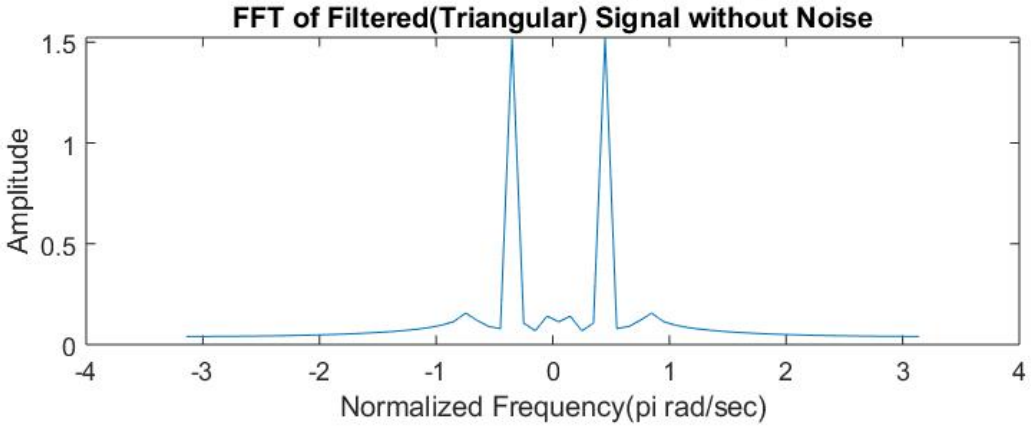
Time Domain response (Blackman Filter)

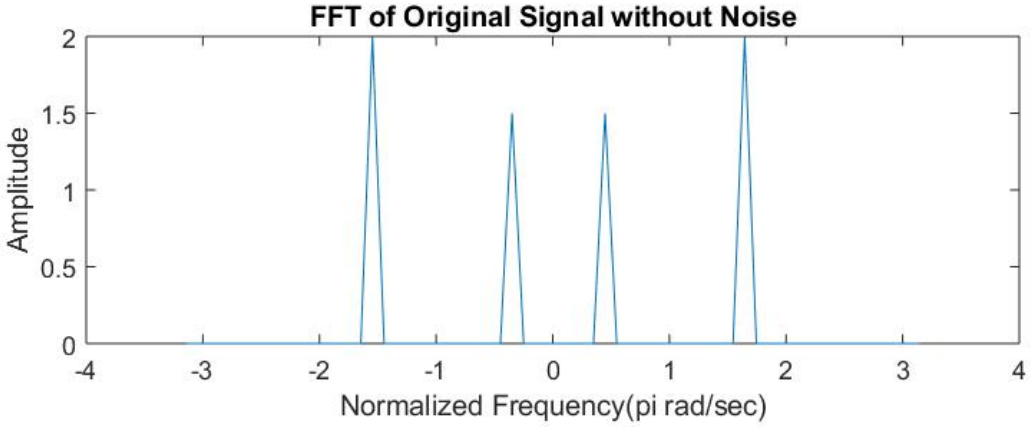


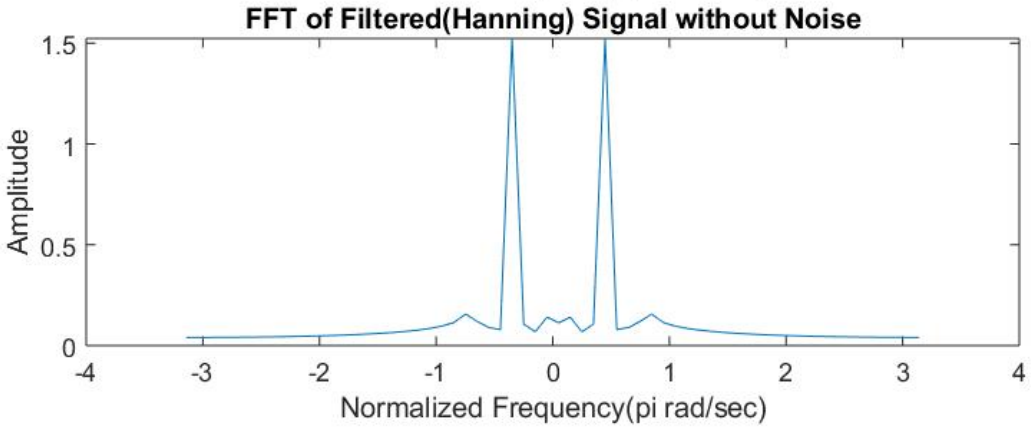
Frequency Domain(fft) for various filters at N = 64

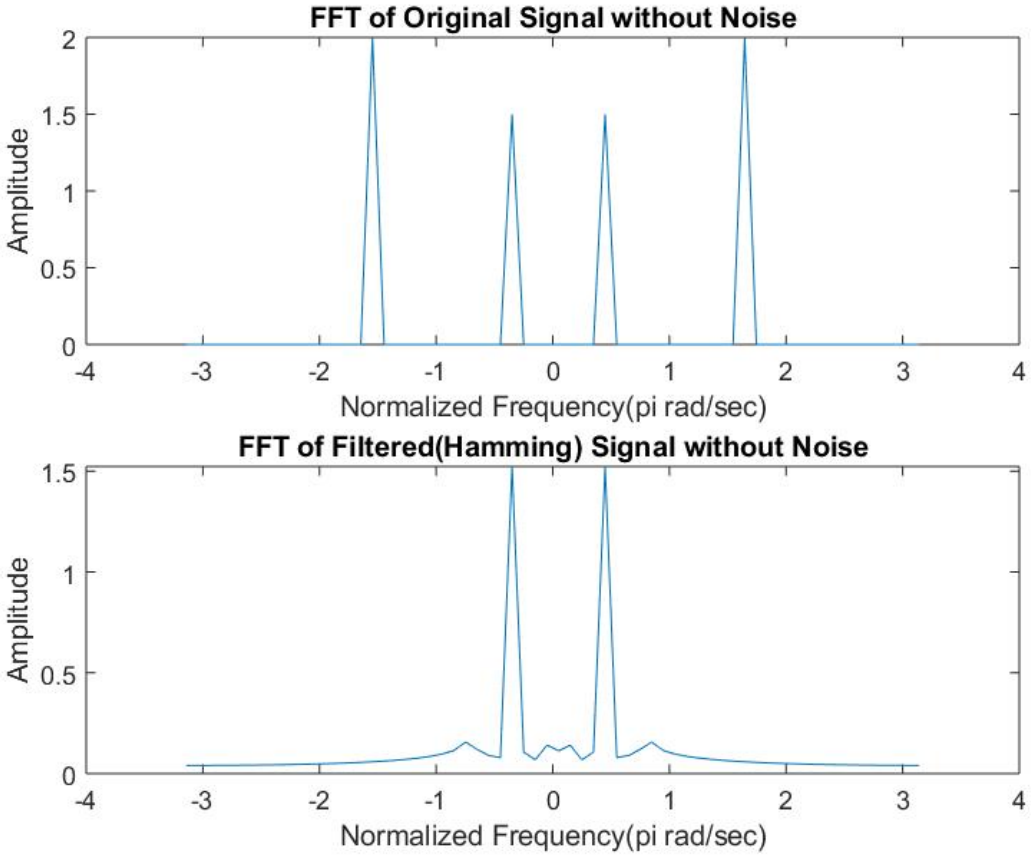


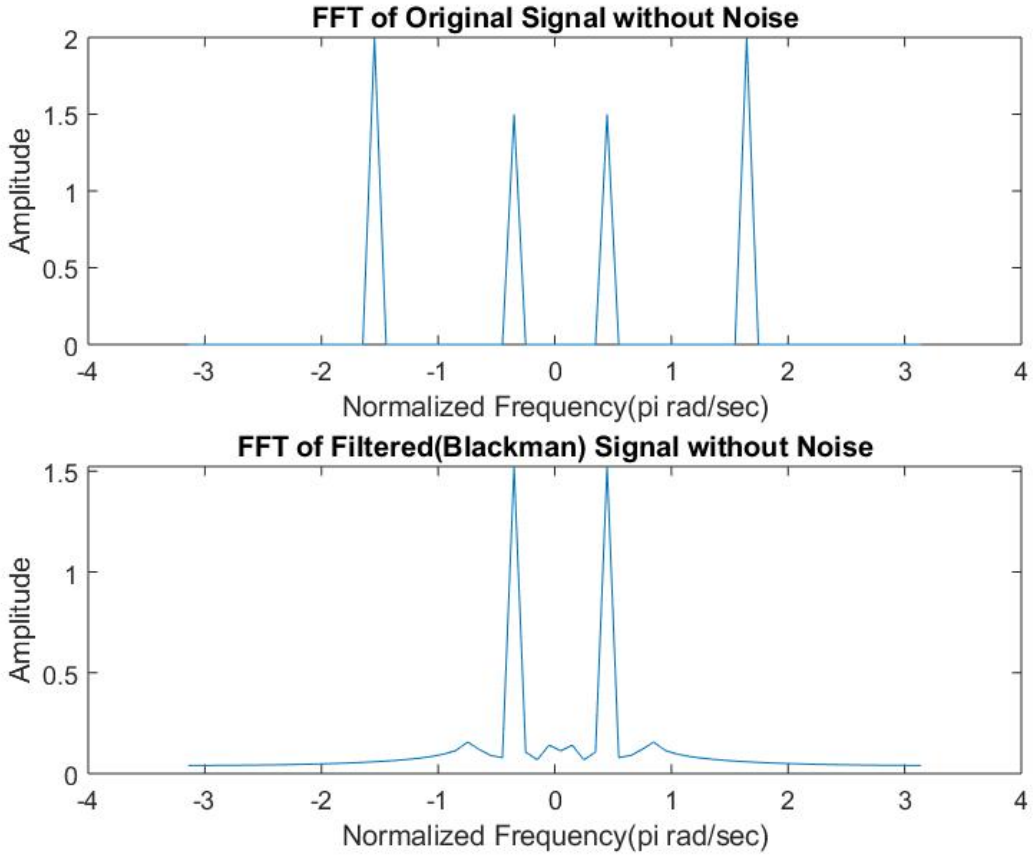

FFT of Original Signal without Noise
Amplitude
Normalized Frequency(pi rad/sec)


FFT of Filtered(Triangular) Signal without Noise
Amplitude
Normalized Frequency(pi rad/sec)


FFT of Original Signal without Noise
Amplitude
Normalized Frequency(pi rad/sec)


FFT of Filtered(Hanning) Signal without Noise
Amplitude
Normalized Frequency(pi rad/sec)


FFT of Original Signal without Noise
Amplitude
Normalized Frequency(pi rad/sec)
FFT of Filtered(Hamming) Signal without Noise
Amplitude
Normalized Frequency(pi rad/sec)



(c)Response of FIR filters when signals are contaminated by noise.

(i)Theory:

It is natural for noise to be added up in the signal due to channel or various other reasons. The filter's capability to reduce the noise is a great parameter for the quality of the filter. Noise is added using randn() function which generates randomly distributed noise.

(ii)Code:

a) For FFT with noise added:

```
b)  N=64;
    wc = pi/4;
    k = (N-1)/2;
    n = 0:1:N-1;
    hd = (sin(wc*(n-k)))./(pi*(n-k));
    w1 = (n>=0)-(n>=N).*(0.42-0.5*cos((2*pi*n)/(N-1))+0.08*cos((4*pi*n)/(N-1))); %blackman
    h = hd.*w1;
    c = -pi:0.01:pi;
    [h1,w] = freqz(h,1,c);
    h2 = abs(h1);
```

```
t = 0:1:3*(N-1);
x = 3*cos(pi/8*t)+4*cos(pi/2*t)+randn(size(t));

w2 = -pi:2*pi/(N-1):pi;

y = filtfilt(h,1,x);


x1 = fft(x,N);
x1 = fftshift(x1);
x1a = abs(x1/N);
subplot(211);
plot(w2,x1a);
title('FFT of Original Signal with Noise');
xlabel('Normalized Frequency(pi rad/sec)');
ylabel('Amplitude');

y1 = fft(y,N);
y1 = fftshift(y1);
y1a = abs(y1/N);
subplot(212);
plot(w2,y1a);
title('FFT of Filtered(Blackman) Signal with Noise');
xlabel('Normalized Frequency(pi rad/sec)');
ylabel('Amplitude');
```

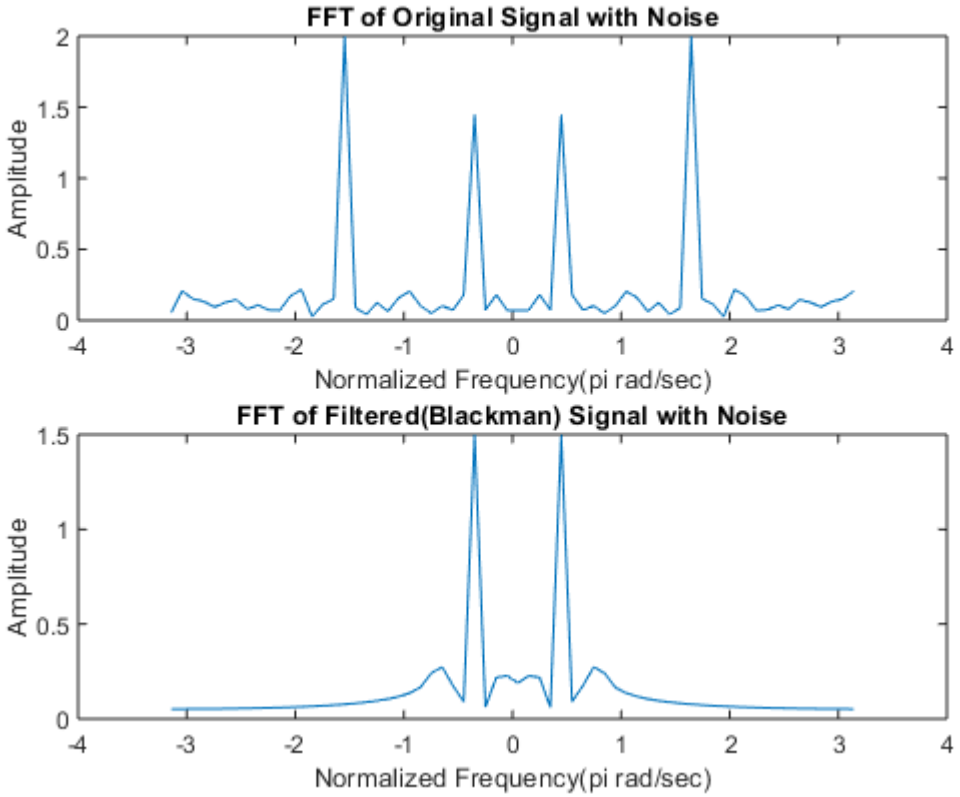
b) For SNR Calculation:

```
N = 64;
wc = pi/3;
k = (N-1)/2;
n = 0:1:N-1;
hd = (sin(wc*(n-k)))./(pi*(n-k));
w1 = (n>=0)-(n>=N);
h = hd.*w1;
c = -pi:0.01:pi;
[h1,w] = freqz(h,1,c);
h2 = abs(h1);


t = 0:1:3*(N-1);
x = 3*cos(pi/8*t)+5*cos(pi/2*t);
y = filtfilt(h,1,x);
x1 = 3*cos(pi/8*t)+5*cos(pi/2*t) + 0.1*randn(size(t));
y1 = filtfilt(h,1,x1);

yn = y1-y;
xn = x1-x;
```

```
R1 = snr(y,yn);
R2 = snr(x,xn);

E1 = 0; %y
E2 = 0; %yn
E3 = 0; %x
E4 = 0; %xn

for i=1:1:3*(N-1)
    E1 = E1+y(i)*y(i);
end

for i=1:1:3*(N-1)
    E2 = E2+yn(i)*yn(i);
end

for i=1:1:3*(N-1)
    E3 = E3+x(i)*x(i);
end

for i=1:1:3*(N-1)
    E4 = E4+xn(i)*xn(i);
end

SNRin = 10*log(E3/E4);
SNRout = 10*log(E1/E2);
```

(iii)Results:

FFT of Blackman Filter with Noise:



SNR Calculations for various filters:

N=8

Window	SNRin(dB)	SNRout(dB)
Rectangular	17.7469	15.0708
Triangular	17.8404	15.8097
Hanning	17.7695	16.1127
Hamming	17.8849	16.0747
Blackman	17.8256	16.1860

N=64

Window	SNRin(dB)	SNRout(dB)
Rectangular	17.7758	15.7602
Triangular	17.8258	15.9406
Hanning	17.8235	15.9720
Hamming	17.9184	16.0036
Blackman	17.9240	16.0561

N=512

Window	SNRin(dB)	SNRout(dB)
Rectangular	17.9152	15.9896
Triangular	17.8151	15.8714
Hanning	17.9141	16.0256
Hamming	17.8771	15.9285
Blackman	17.8743	15.9495

Discussions:

We can see that as we increase the value of N (i.e. the number of samples of IIR response of filter considered) the filter response becomes sharper with faster transitions from pass-band to stop-band and lower side lobes. In general, the rectangular window has much faster transitions but comparatively higher side lobes around -20 dB which is 10% of passband value and hence not a good response. The triangular window has very slow transitions but no significant side lobes. The Hanning, Hamming and Blackman windows have comparatively slower transitions but have much lower side-bands which means a much better stop-band performance.

We observed that higher the order of the filter (value of N), the better is the impulse response i.e. lesser transition width (from pass-band to stop-band), more stop-band attenuation and lower value of side-lobe peaks. It was observed that rectangular window has lower transition width as compared to other windows but has comparatively more energy in the side lobes. No significant side lobes were observed in the triangular window. The other three windows have much better performance in the stop-band but comparatively have larger transition width than the rectangular window.

References:

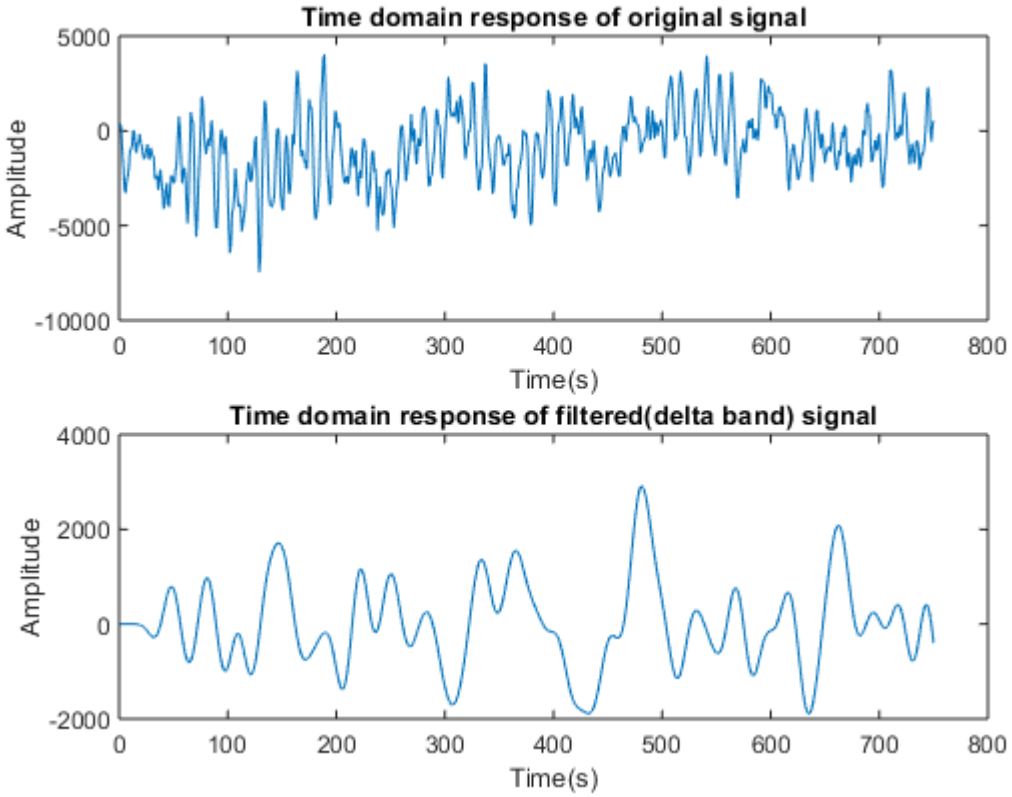
-Alan V. Oppenheim et.al Discrete Time Signal Processing.
-Wikipedia.
-Digital Signal Processing – Proakis and Manolakis

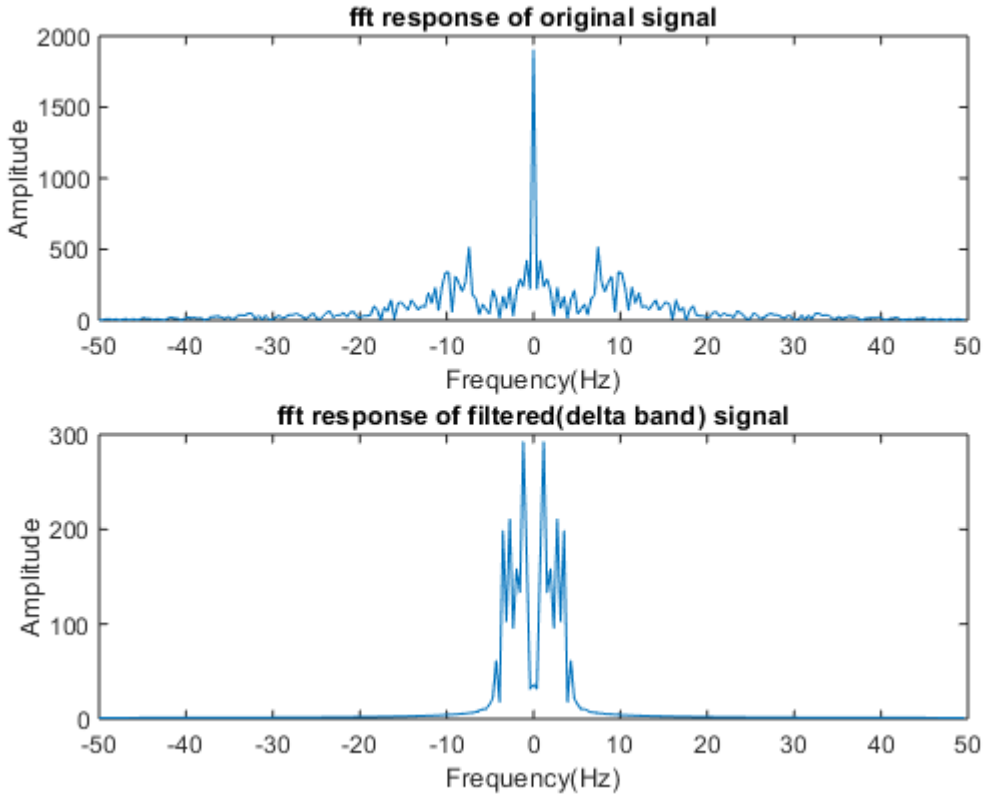
Published with MATLAB® R2018a

Extended Experiment on EEG Signals:

Code and results:

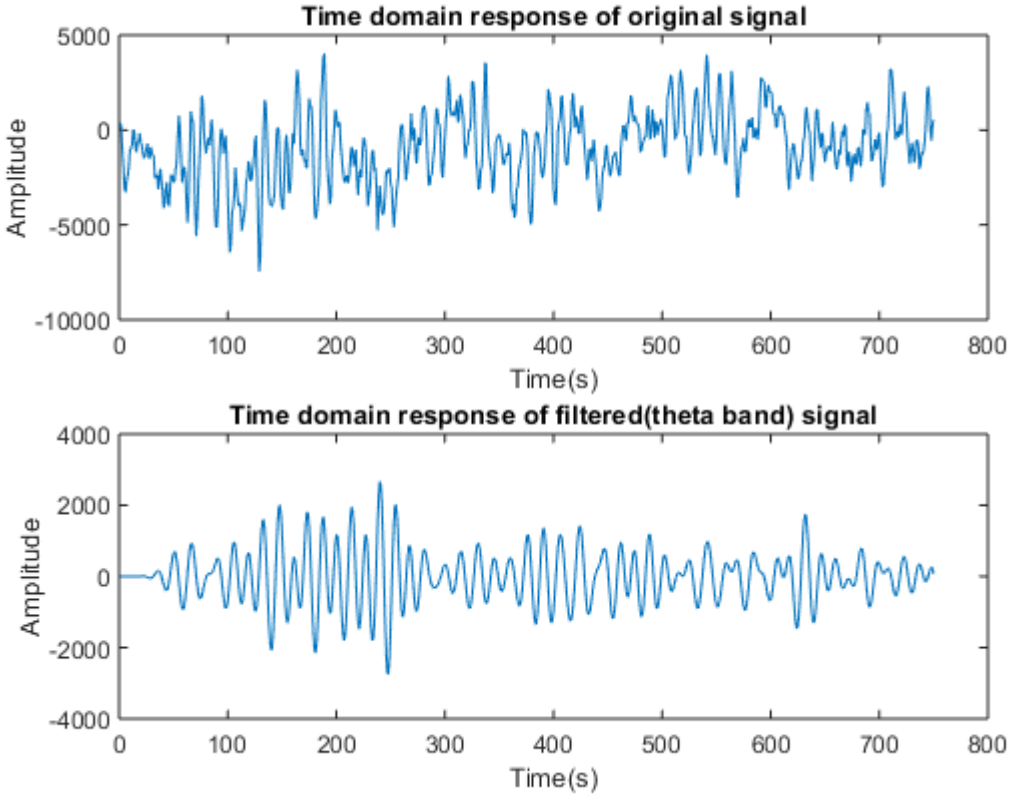
```
x = load ('C:\Users\Pranit Dalal\Downloads\EXP2\signals\eeg1-f3.dat');
fs = 100;
t = 0:1/fs:1;

N = 256;

bpf = designfilt('bandpassiir','FilterOrder',20,'HalfPowerFrequency1',0.5
,'HalfPowerFrequency2',4 ,'SampleRate',100);
y = filter(bpf,x);
figure(1);
subplot(211);
plot(x);
title('Time domain response of original signal');
xlabel('Time(s)');
ylabel('Amplitude');
subplot(212);
plot(y);
title('Time domain response of filtered(delta band) signal');
xlabel('Time(s)');
ylabel('Amplitude');

y1 = fft(x,N);
y1 = fftshift(y1);
m1 = abs(y1/N);
f1 = (-N/2:(N/2-1))*100/N;
figure(2);
subplot(211);
plot(f1,m1);
title('fft response of original signal');
xlabel('Frequency(Hz)');
ylabel('Amplitude');
y2 = fft(y,N);
y2 = fftshift(y2);
m2 = abs(y2/N);
f2 = (-N/2:(N/2-1))*100/N;
subplot(212);
plot(f2,m2);
title('fft response of filtered(delta band) signal');
xlabel('Frequency(Hz)');
ylabel('Amplitude');
```

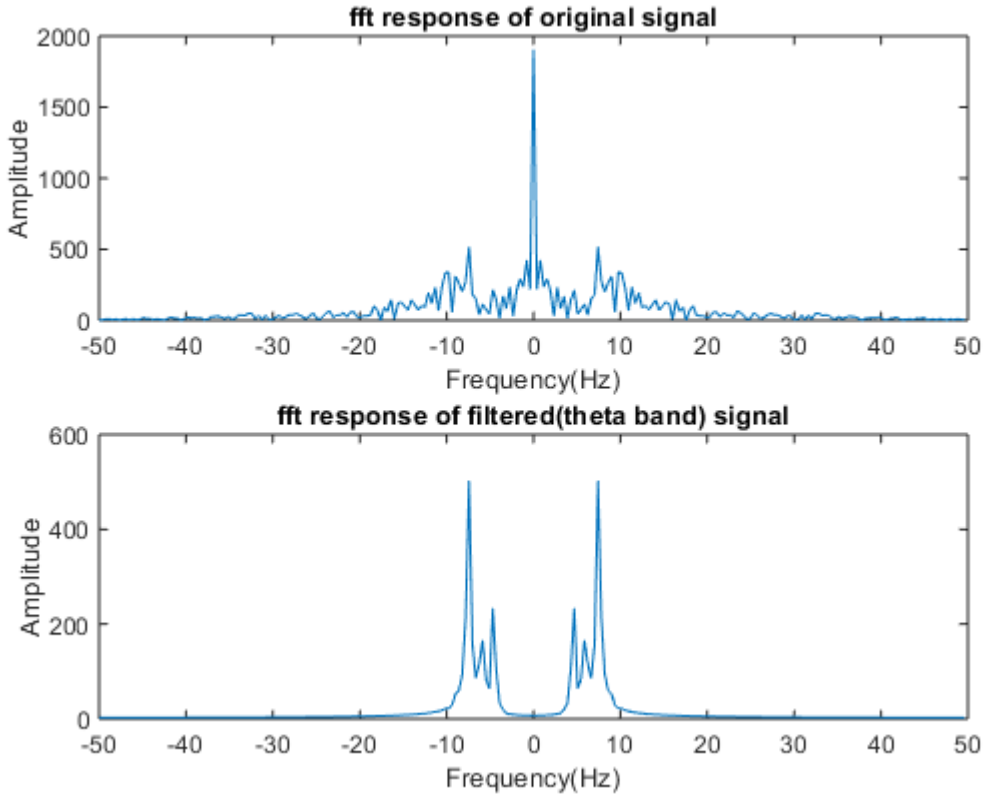

Time domain response of original signal
5000
0
-5000
-10000
Amplitude
0
100
200
300
400
500
600
700
800
Time(s)
Time domain response of filtered(delta band) signal
4000
2000
0
-2000
Amplitude
0
100
200
300
400
500
600
700
800
Time(s)




```
x = load ('C:\Users\Pranit Dalal\Downloads\EXP2\signals\eeg1-f3.dat');
fs = 100;
t = 0:1/fs:1;

N = 256;

bpf = designfilt('bandpassiir','FilterOrder',20,'HalfPowerFrequency1',4
,'HalfPowerFrequency2',8 ,'SampleRate',100);
y = filter(bpf,x);
figure(1);
subplot(211);
plot(x);
title('Time domain response of original signal');
xlabel('Time(s)');
ylabel('Amplitude');
subplot(212);
plot(y);
title('Time domain response of filtered(theta band) signal');
xlabel('Time(s)');
ylabel('Amplitude');

y1 = fft(x,N);
y1 = fftshift(y1);
m1 = abs(y1/N);
f1 = (-N/2:(N/2-1))*100/N;
figure(2);
subplot(211);
```

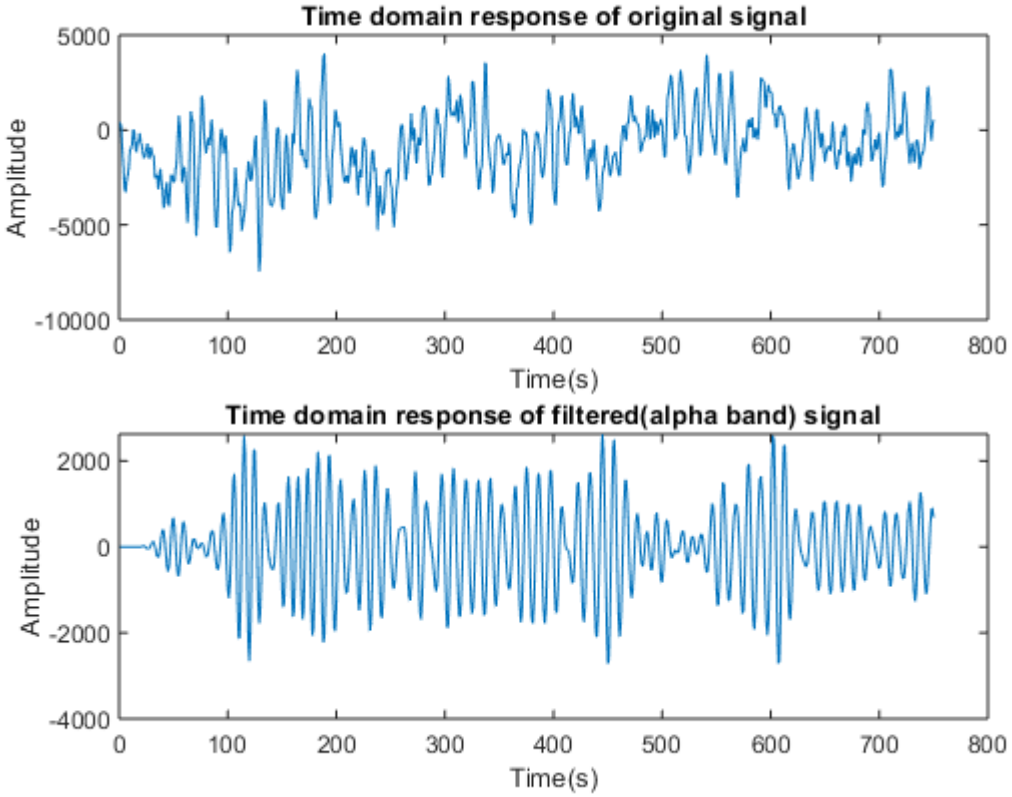
```
plot(f1,m1);
title('fft response of original signal');
xlabel('Frequency(Hz)');
ylabel('Amplitude');
y2 = fft(y,N);
y2 = fftshift(y2);
m2 = abs(y2/N);
f2 = (-N/2:(N/2-1))*100/N;
subplot(212);
plot(f2,m2);
title('fft response of filtered(theta band) signal');
xlabel('Frequency(Hz)');
ylabel('Amplitude');
```

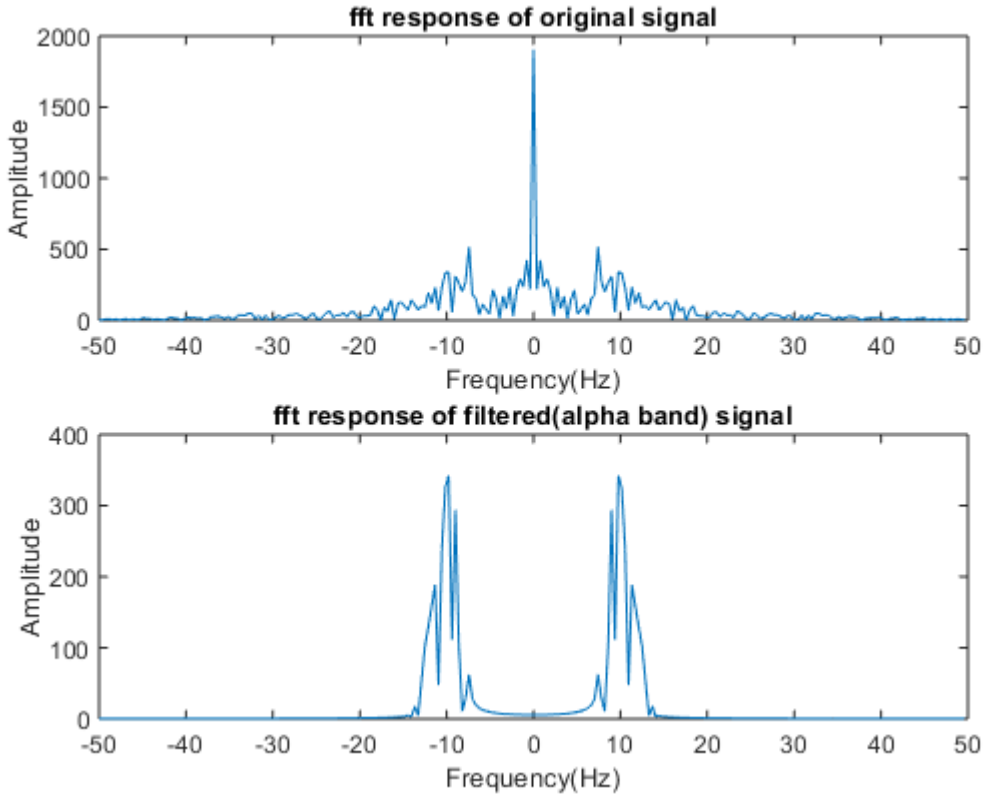





```
x = load ('C:\Users\Pranit Dalal\Downloads\EXP2\signals\eeg1-f3.dat');
fs = 100;
t = 0:1/fs:1;

N = 256;

bpf = designfilt('bandpassiir','FilterOrder',20,'HalfPowerFrequency1',8
,'HalfPowerFrequency2',13 ,'SampleRate',100);
y = filter(bpf,x);
figure(1);
subplot(211);
plot(x);
title('Time domain response of original signal');
xlabel('Time(s)');
ylabel('Amplitude');
subplot(212);
plot(y);
title('Time domain response of filtered(alpha band) signal');
xlabel('Time(s)');
ylabel('Amplitude');

y1 = fft(x,N);
y1 = fftshift(y1);
m1 = abs(y1/N);
f1 = (-N/2:(N/2-1))*100/N;
figure(2);
subplot(211);
```

```
plot(f1,m1);
title('fft response of original signal');
xlabel('Frequency(Hz)');
ylabel('Amplitude');
y2 = fft(y,N);
y2 = fftshift(y2);
m2 = abs(y2/N);
f2 = (-N/2:(N/2-1))*100/N;
subplot(212);
plot(f2,m2);
title('fft response of filtered(alpha band) signal');
xlabel('Frequency(Hz)');
ylabel('Amplitude');
```





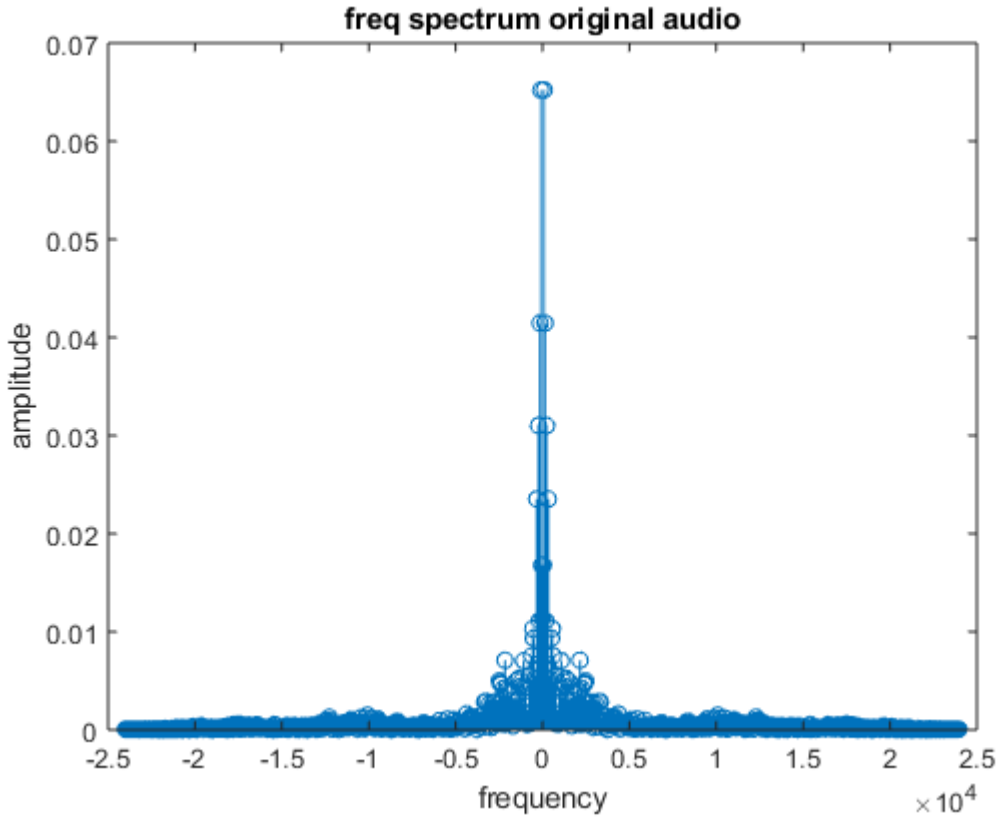
Published with MATLAB® R2018a

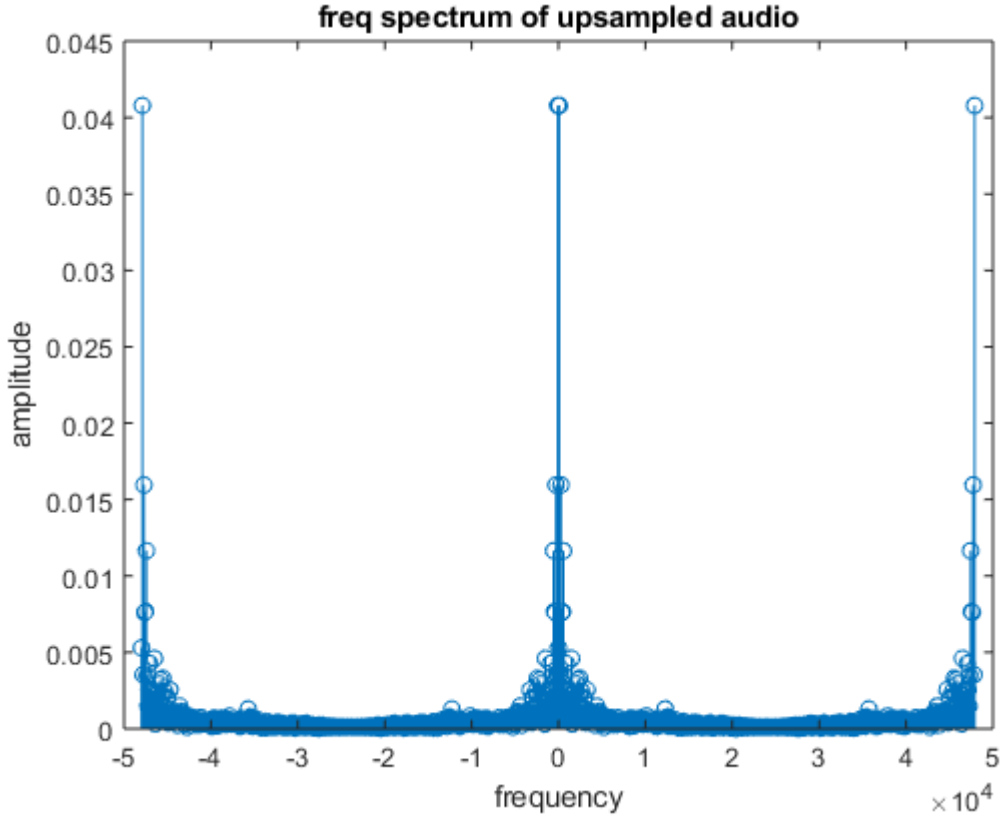
Digital Signal Processing Lab

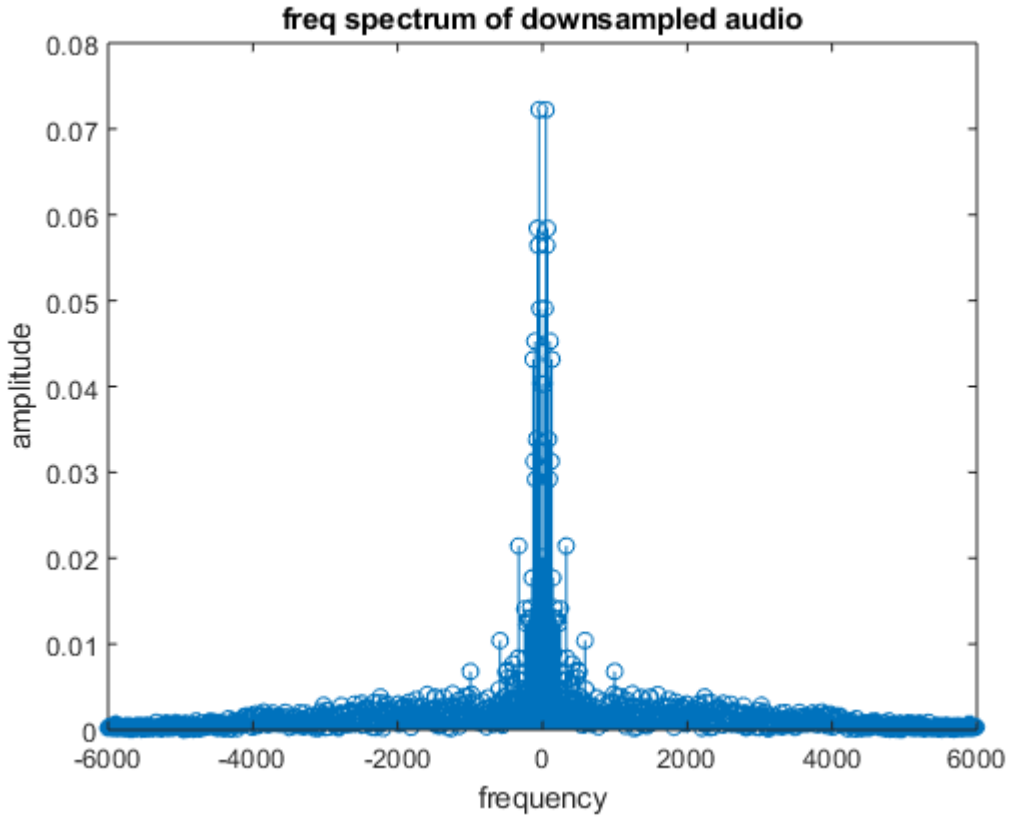
Expt. No. 2*
Audio Signal Part

By Pranit Dalal
Roll no. 16EC10016
Group 22 (Tuesday)

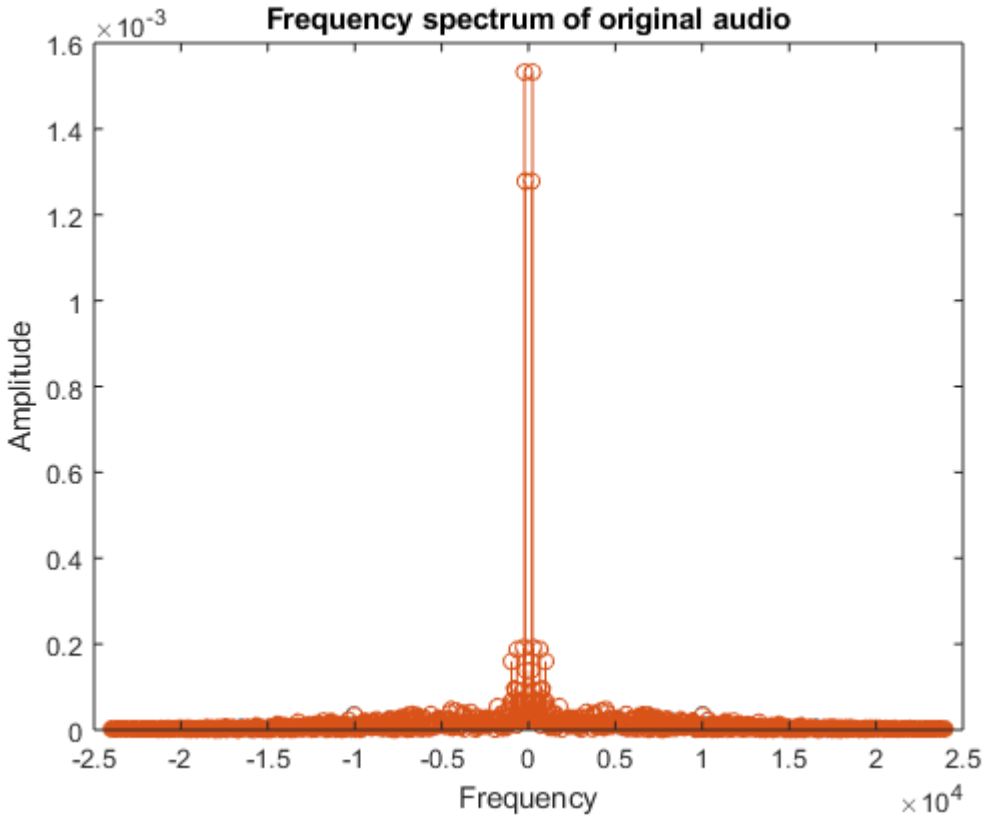
```
%UPSAMPLING AND DOWNSAMPLING OF 48KHZ AUDIO SIGNAL
clc
clear all
close all
[xn, fs] = audioread('audio48kHz.wav');
N=1024;                              %number of point in DTFT
y = fftshift(fft(xn,N));
f =(-(length(y)/2):((length(y)/2)-1))*fs/length(y);
figure(1)
stem(f,abs(y)/N);
xlabel('frequency');ylabel('amplitude');
title('freq spectrum original audio');
%upsampling
x1 = upsample(xn,2);
y1 = fftshift(fft(x1,N));
f =(-(length(y1)/2):((length(y1)/2)-1))*2*fs/length(y1);
figure(2)
stem(f,abs(y1)/N);
xlabel('frequency');ylabel('amplitude');
title('freq spectrum of upsampled audio');
%downsampling
x2 = downsample(xn, 4);
y2 = fftshift(fft(x2,N));
f =(-(length(y2)/2):((length(y2)/2)-1))*0.25*fs/length(y2);
figure(3)
stem(f,abs(y2)/N);
xlabel('frequency');ylabel('amplitude');
title('freq spectrum of downsampled audio');
nBits = 16;
audiowrite('48kHz_upsampled.wav', x1, fs, 'BitsPerSample',nBits);
audiowrite('48kHz_downsampled.wav', x2, fs, 'BitsPerSample',nBits);
```

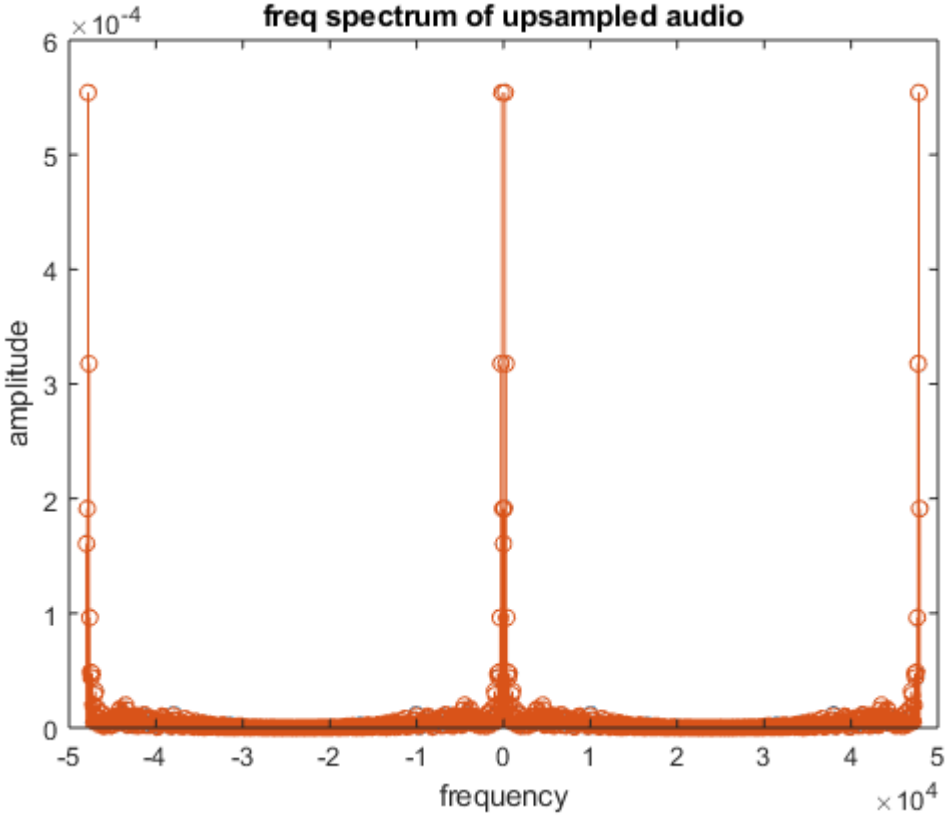

freq spectrum original audio
amplitude
frequency
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
×10^4

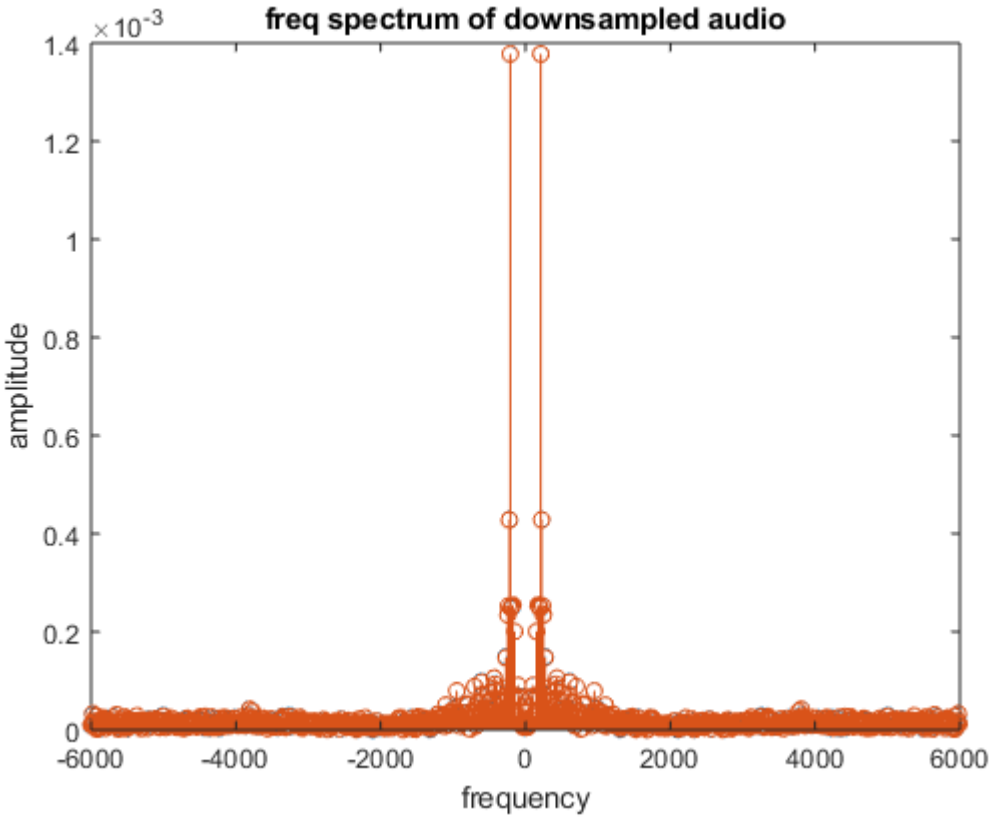

freq spectrum of upsampled audio
amplitude
frequency
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
-5
-4
-3
-2
-1
0
1
2
3
4
5
×10^4


freq spectrum of downsampled audio
amplitude
frequency
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
-6000
-4000
-2000
0
2000
4000
6000

```
% UPSAMPLING AND DOWNSAMPLING OF 48KHZ AUDIO SIGNAL(audio_record.m4a)
clc
clear all
close all
[xn, fs] = audioread('C:\Users\Pranit Dalal\Downloads\EXP2\signals\audio_record1.m4a');
N=1024;                              %number of point in DTFT
y = fftshift(fft(xn,N));
f =(-(length(y)/2):((length(y)/2)-1))*fs/length(y);
figure(1)
stem(f,abs(y)/N);
xlabel('Frequency');ylabel('Amplitude');
title('Frequency spectrum of original audio');
%upsampling
x1 = upsample(xn,2);
y1 = fftshift(fft(x1,N));
f =(-(length(y1)/2):((length(y1)/2)-1))*2*fs/length(y1);
figure(2)
stem(f,abs(y1)/N);
xlabel('frequency');ylabel('amplitude');
title('freq spectrum of upsampled audio');
%downsampling
x2 = downsample(xn, 4);
y2 = fftshift(fft(x2,N));
f =(-(length(y2)/2):((length(y2)/2)-1))*0.25*fs/length(y2);
figure(3)
stem(f,abs(y2)/N);
xlabel('frequency');ylabel('amplitude');
title('freq spectrum of downsampled audio');
nBits = 16;
audiowrite('audio_record1_upsampled.wav', x1, fs, 'BitsPerSample',nBits);
audiowrite('audio_record1_downsampled.wav', x2, fs, 'BitsPerSample',nBits);
```


Frequency spectrum of original audio
×10^-3
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Amplitude
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
Frequency
×10^4


freq spectrum of upsampled audio
×10^-4
6
5
4
3
2
1
0
amplitude
-5
-4
-3
-2
-1
0
1
2
3
4
5
frequency
×10^4



Published with MATLAB® R2018a

Digital Signal Processing Lab

Expt. No. 3
DTMF (Dual Tone Multi Frequency or Touch Tone) coder or decoder

By Pranit Dalal
Roll no. 16EC10016
Group 22 (Tuesday)

AIM:

Study and analysis of DTMF coder/decoder using Digital FIR filter in MATLAB.

We will learn about:

(a)Functionality of DTMF and its different applications.

(b)Implementation of Bandpass Digital Filter

(c)DTMF coder/decoder

(d)How to characterise a filter by knowing how it reacts to different frequency components in the input.

THEORY:

Dual tone multiple frequency (DTMF) signalling is used in telephone dialling, digital answer machines, and interactive banking systems. DTMF signalling represents each symbol on a telephone touchtone keypad (0–9, *, #) using two sinusoidal tones, as shown in Figure. When a key is pressed, a DTMF signal consisting of a row frequency tone plus a column frequency tone is transmitted. Keys A–D are not on commercial telephone sets but are used in military and radio signalling applications.

	1209 Hz	1336 Hz	1477 Hz	1663 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

No frequency is an integer multiple of another and the difference or sum of any two frequencies does not equal any of the frequencies. Pressing 4 should generate a tone containing a 770 Hz and a 1209Hz frequency component.

DTMF coder:

The coder should give an output consisting 2 frequencies.

$$x(t) = A_1\sin(2\pi*f1*t) + A_2\sin(2\pi*f2*t)$$

where in this experiment we used $A_1 = A_2 = 1$ without losing the generality.

DTMF decoder:

There are several steps in decoding a DTMF signal

1.Divide the time signal into short time segments representing individual key presses

2. Filter the individual segments. Band Pass filters are used to isolate the sinusoidal components

3.Determine which two frequency components are present in each segment by measuring the size of the output signal from all the Band Pass Filters.

4 - Decode which key was pressed, 0-9, A - D, * or # by converting frequency pairs back into key names according to above given table.

BAND PASS FILTER DESIGN:
THE L-PONT AVERAGE FILTER IS A LOW PASS FILTER

Its pass bandwidth is inversely proportional to L. It is also possible to create a filter whose pass band is centred around some frequency other than zero. One simple way to do this is to define the impulse response of an L-point FIR filter as

$$h[n] = \beta \cos(\omega c * n) \ \ where\ 0 \leq n < L$$

Where L is the filter length, and ωc is the centre frequency that defines the frequency location of the pass band and p is used to adjust the gain in the pass band. So, it is possible to, choose β so that the maximum value of the frequency response magnitude will be one. The Band width of the band pass filter is controlled by L: The larger the value of L, the narrower the band width.

CODE:

This code encodes a single bit entered in the command window and outputs the time domain signal output consisting of the frequency corresponding to the key pressed. Fourier transform of the signal is also obtained.

The decoder then returns the key pressed by passing the signal through filters and calculating the index of max rms value of output.

```
clear all;
N = 512;
Fs = 10000;
T = 1/Fs;
t = 0:T:(N-1)*T*10;
F = [697,770,852,941;1209,1336,1477,1633];

D_string = 'Entered Character is ';
i = input(D_string, 's');

if i == '1'
    f1 = F(1,1);
    f2 = F(2,1);
elseif i == '2'
    f1 = F(1,1);
    f2 = F(2,2);
elseif i == '3'
    f1 = F(1,1);
    f2 = F(2,3);
elseif i == 'A'
    f1 = F(1,1);
    f2 = F(2,4);
elseif i == '4'
    f1 = F(1,2);
    f2 = F(2,1);
elseif i == '5'
    f1 = F(1,2);
    f2 = F(2,2);
elseif i == '6'
    f1 = F(1,2);
    f2 = F(2,3);
```

```
elseif i == 'B'
    f1 = F(1,2);
    f2 = F(2,4);
elseif i == '7'
    f1 = F(1,3);
    f2 = F(2,1);
elseif i == '8'
    f1 = F(1,3);
    f2 = F(2,2);
elseif i == '9'
    f1 = F(1,3);
    f2 = F(2,3);
elseif i == 'C'
    f1 = F(1,3);
    f2 = F(2,4);
elseif i == '*'
    f1 = F(1,4);
    f2 = F(2,1);
elseif i == '0'
    f1 = F(1,4);
    f2 = F(2,2);
elseif i == '#'
    f1 = F(1,4);
    f2 = F(2,3);
elseif i == 'D'
    f1 = F(1,4);
    f2 = F(2,4);
else
    fprintf('Invalid Selection\n');
end

y = cos(2*pi*f1*t) + cos(2*pi*f2*t);

figure;
plot(t,y);
y1 = fft(y,N);
y1 = fftshift(y1);
y1 = abs(y1/N);
w2 = -pi:2*pi/(N-1):pi;
figure;
plot(w2,y1);

y_rms = zeros(2,4);

L=128;
n = 0:1:L-1;

BPF11 = cos(2*pi/Fs*F(1,1)*n);
Y11 = conv(y,BPF11);

y_fft11 = fft(Y11,N);
y_fft11 = fftshift(y_fft11);
y_fft11 = abs(y_fft11/N);
y_rms(1,1) = rms(y_fft11);


BPF12 = cos(2*pi/Fs*F(1,2)*n);
Y12 = conv(y,BPF12);
```

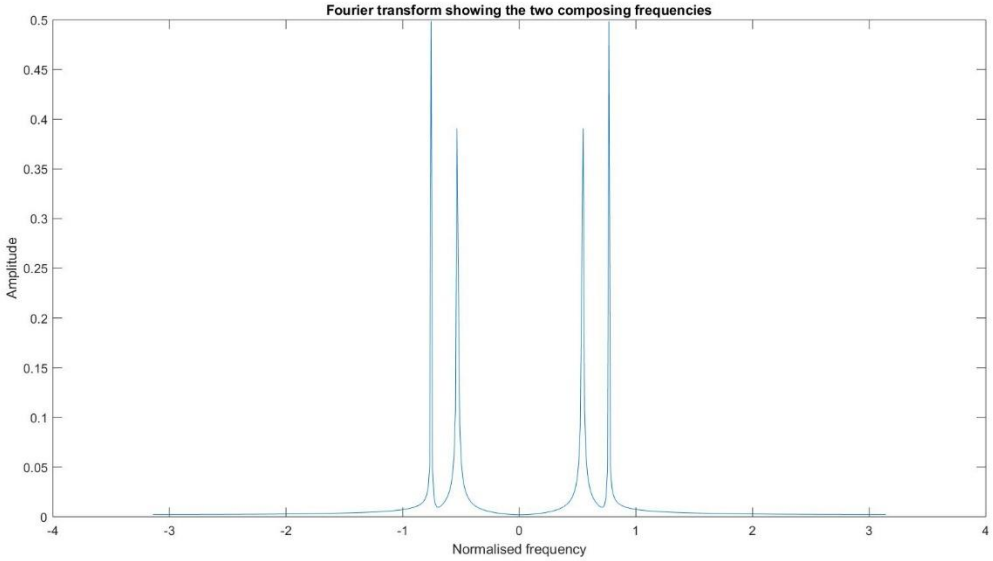
```
y_fft12 = fft(Y12,N);
y_fft12 = fftshift(y_fft12);
y_fft12 = abs(y_fft12/N);
y_rms(1,2) = rms(y_fft12);


BPF13 = cos(2*pi/Fs*F(1,3)*n);
Y13 = conv(y,BPF13);

y_fft13 = fft(Y13,N);
y_fft13 = fftshift(y_fft13);
y_fft13 = abs(y_fft13/N);
y_rms(1,3) = rms(y_fft13);


BPF14 = cos(2*pi/Fs*F(1,4)*n);
Y14 = conv(y,BPF14);

y_fft14 = fft(Y14,N);
y_fft14 = fftshift(y_fft14);
y_fft14 = abs(y_fft14/N);
y_rms(1,4) = rms(y_fft14);


BPF21 = cos(2*pi/Fs*F(2,1)*n);
Y21 = conv(y,BPF21);

y_fft21 = fft(Y21,N);
y_fft21 = fftshift(y_fft21);
y_fft21 = abs(y_fft21/N);
y_rms(2,1) = rms(y_fft21);


BPF22 = cos(2*pi/Fs*F(2,2)*n);
Y22 = conv(y,BPF22);

y_fft22 = fft(Y22,N);
y_fft22 = fftshift(y_fft22);
y_fft22 = abs(y_fft22/N);
y_rms(2,2) = rms(y_fft22);


BPF23 = cos(2*pi/Fs*F(2,3)*n);
Y23 = conv(y,BPF23);

y_fft23 = fft(Y23,N);
y_fft23 = fftshift(y_fft23);
y_fft23 = abs(y_fft23/N);
y_rms(2,3) = rms(y_fft23);


BPF24 = cos(2*pi/Fs*F(2,4)*n);
Y24 = conv(y,BPF24);

y_fft24 = fft(Y24,N);
y_fft24 = fftshift(y_fft24);
y_fft24 = abs(y_fft24/N);
```

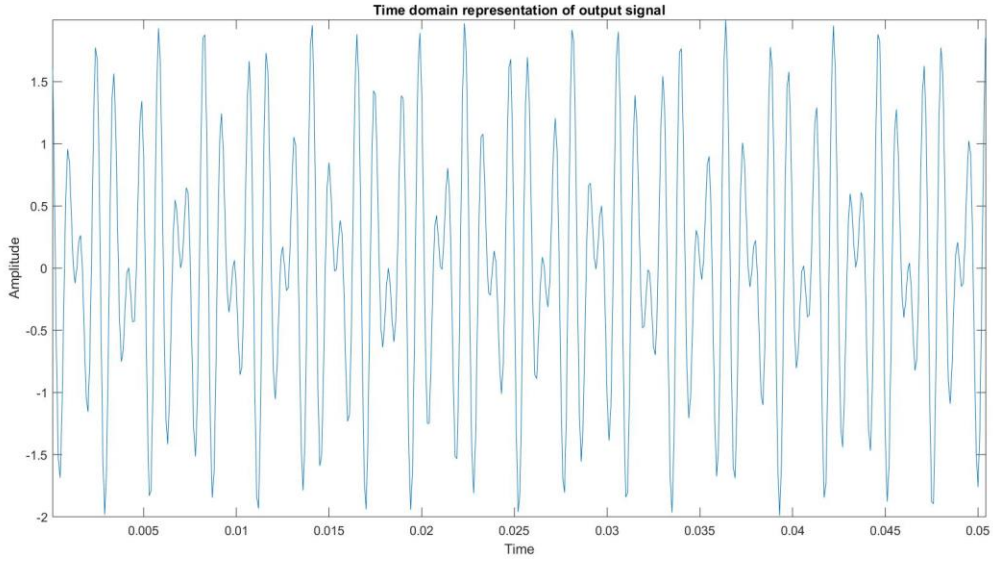
```
y_rms(2,4) = rms(y_fft24);


[M,I] = max(y_rms.');

if I(1)==1&&I(2)==1
    fprintf('Detected sequence = 1');
elseif I(1)==1&&I(2)==2
    fprintf('Detected sequence = 2');
elseif I(1)==1&&I(2)==3
    fprintf('Detected sequence = 3');
elseif I(1)==1&&I(2)==4
    fprintf('Detected sequence = A');
elseif I(1)==2&&I(2)==1
    fprintf('Detected sequence = 4');
elseif I(1)==2&&I(2)==2
    fprintf('Detected sequence = 5');
elseif I(1)==2&&I(2)==3
    fprintf('Detected sequence = 6');
elseif I(1)==2&&I(2)==4
    fprintf('Detected sequence = B');
elseif I(1)==3&&I(2)==1
    fprintf('Detected sequence = 7');
elseif I(1)==3&&I(2)==2
    fprintf('Detected sequence = 8');
elseif I(1)==3&&I(2)==3
    fprintf('Detected sequence = 9');
elseif I(1)==3&&I(2)==4
    fprintf('Detected sequence = C');
elseif I(1)==4&&I(2)==1
    fprintf('Detected sequence = *');
elseif I(1)==4&&I(2)==2
    fprintf('Detected sequence = 0');
elseif I(1)==4&&I(2)==3
    fprintf('Detected sequence = #');
elseif I(1)==4&&I(2)==4
    fprintf('Detected sequence = D');
end
fprintf("\r");
```

Outputs:

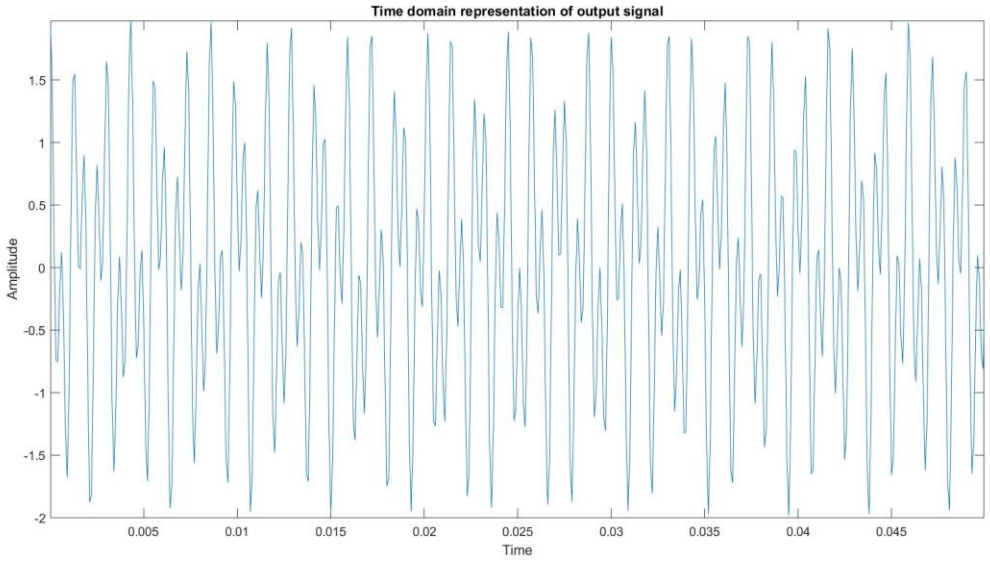
Checking the working of the code for input 7 and the input A. The following plots were obtained:

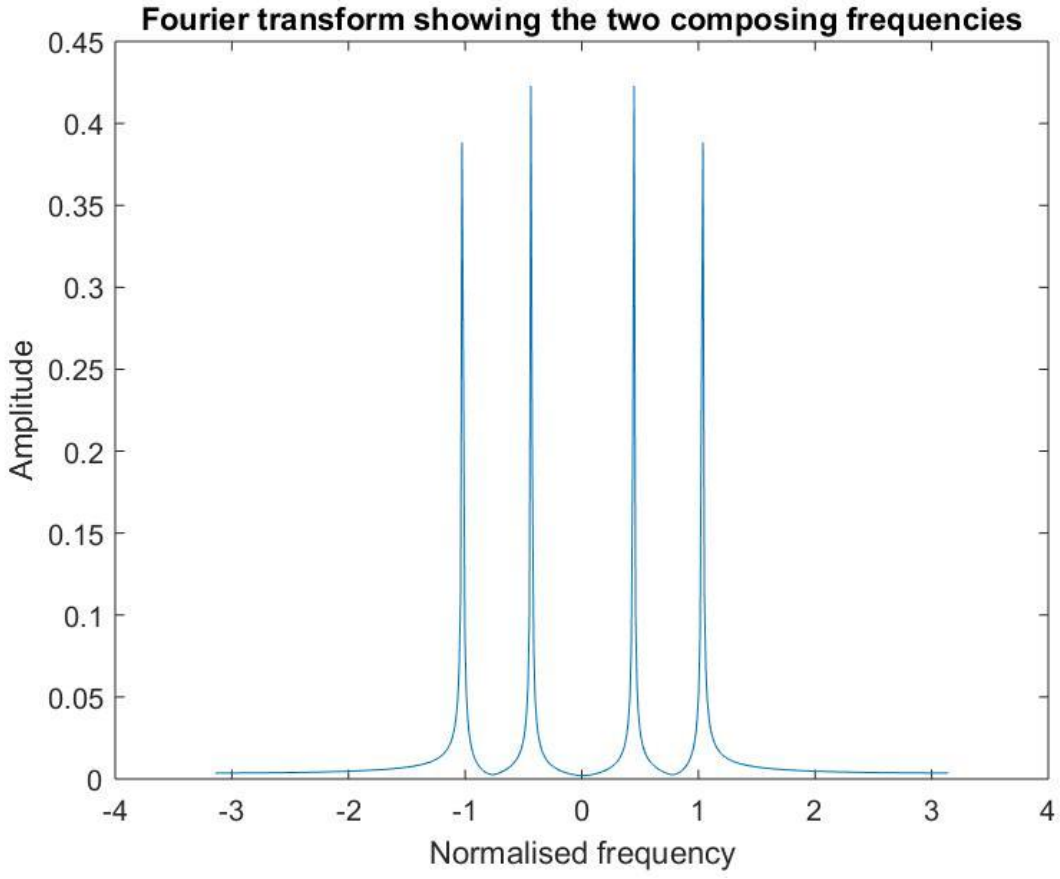
```
Command Window
>> DTMF
Entered Character is 7
Detected sequence = 7
fx >>
```





```
Command Window
>> DTMF
Entered Character is 7
Detected sequence = 7
>> DTMF
Entered Character is A
Detected sequence = A
fx >>
```





Code for string input:

The input of 4 characters is taken and concatenation of the signal is done and passed through the filters. The output in command window gives the 4 characters as the output along with the spaces generated implying that no signal was sent during that duration.

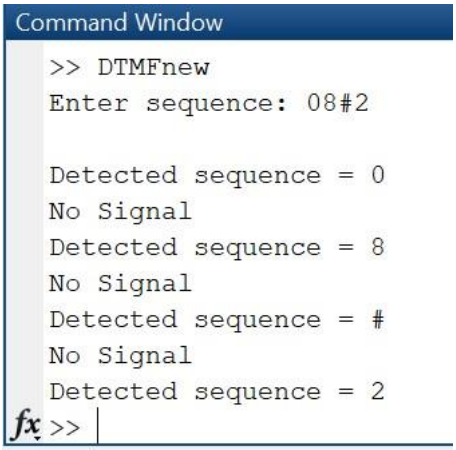
```
clear all;
N = 512;
Fs = 10000;
T = 1/Fs;
t = 0:T:70*N*T+6*T;
t1 = 0:T:10*N*T;
F = [697,770,852,941;1209,1336,1477,1633];
j = 1:1:4;
f1 = zeros(1,4);
f2 = zeros(1,4);

prompt = 'Enter sequence: ';
x = input(prompt,'s');

for j=1:4
if x(j) == '1'
    f1(j) = F(1,1);
    f2(j) = F(2,1);
elseif x(j) == '2'
    f1(j) = F(1,1);
    f2(j) = F(2,2);
elseif x(j) == '3'
    f1(j) = F(1,1);
    f2(j) = F(2,3);
elseif x(j) == 'A'
    f1(j) = F(1,1);
    f2(j) = F(2,4);
elseif x(j) == '4'
    f1(j) = F(1,2);
    f2(j) = F(2,1);
elseif x(j) == '5'
    f1(j) = F(1,2);
    f2(j) = F(2,2);
elseif x(j) == '6'
    f1(j) = F(1,2);
    f2(j) = F(2,3);
elseif x(j) == 'B'
    f1(j) = F(1,2);
    f2(j) = F(2,4);
elseif x(j) == '7'
    f1(j) = F(1,3);
    f2(j) = F(2,1);
elseif x(j) == '8'
```

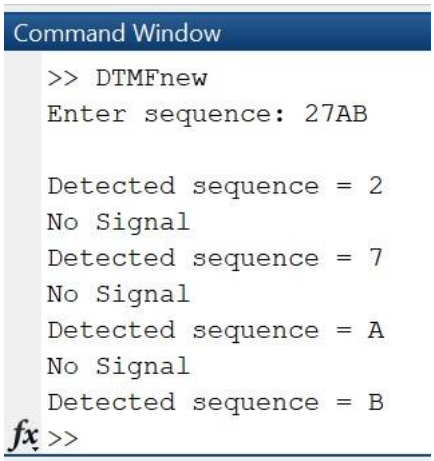
```
    f1(j) = F(1,3);
    f2(j) = F(2,2);
elseif x(j) == '9'
    f1(j) = F(1,3);
    f2(j) = F(2,3);
elseif x(j) == 'C'
    f1(j) = F(1,3);
    f2(j) = F(2,4);
elseif x(j) == '*'
    f1(j) = F(1,4);
    f2(j) = F(2,1);
elseif x(j) == '0'
    f1(j) = F(1,4);
    f2(j) = F(2,2);
elseif x(j) == '#'
    f1(j) = F(1,4);
    f2(j) = F(2,3);
elseif x(j) == 'D'
    f1(j) = F(1,4);
    f2(j) = F(2,4);
else
    fprintf('Invalid Selection\n');
end
end


no_sig = zeros(1,5121);
a = cos(2*pi*f1(1)*t1) + cos(2*pi*f2(1)*t1);
b = cos(2*pi*f1(2)*t1) + cos(2*pi*f2(2)*t1);
c = cos(2*pi*f1(3)*t1) + cos(2*pi*f2(3)*t1);
d = cos(2*pi*f1(4)*t1) + cos(2*pi*f2(4)*t1);

z = [a no_sig b no_sig c no_sig d];

%plot(t,z);

fprintf('\r');
for i = 1:7
    y = z((i-1)*5121+1:i*5121-1);
    y_rms = zeros(2,4);

L=128;
n = 0:1:L-1;

BPF11 = cos(2*pi/Fs*F(1,1)*n);
Y11 = conv(y,BPF11);

y_fft11 = fft(Y11,N);
y_fft11 = fftshift(y_fft11);
y_fft11 = abs(y_fft11/N);
y_rms(1,1) = rms(y_fft11);


BPF12 = cos(2*pi/Fs*F(1,2)*n);
Y12 = conv(y,BPF12);

y_fft12 = fft(Y12,N);
y_fft12 = fftshift(y_fft12);
```

```
y_fft12 = abs(y_fft12/N);
y_rms(1,2) = rms(y_fft12);


BPF13 = cos(2*pi/Fs*F(1,3)*n);
Y13 = conv(y,BPF13);

y_fft13 = fft(Y13,N);
y_fft13 = fftshift(y_fft13);
y_fft13 = abs(y_fft13/N);
y_rms(1,3) = rms(y_fft13);


BPF14 = cos(2*pi/Fs*F(1,4)*n);
Y14 = conv(y,BPF14);

y_fft14 = fft(Y14,N);
y_fft14 = fftshift(y_fft14);
y_fft14 = abs(y_fft14/N);
y_rms(1,4) = rms(y_fft14);


BPF21 = cos(2*pi/Fs*F(2,1)*n);
Y21 = conv(y,BPF21);

y_fft21 = fft(Y21,N);
y_fft21 = fftshift(y_fft21);
y_fft21 = abs(y_fft21/N);
y_rms(2,1) = rms(y_fft21);


BPF22 = cos(2*pi/Fs*F(2,2)*n);
Y22 = conv(y,BPF22);

y_fft22 = fft(Y22,N);
y_fft22 = fftshift(y_fft22);
y_fft22 = abs(y_fft22/N);
y_rms(2,2) = rms(y_fft22);


BPF23 = cos(2*pi/Fs*F(2,3)*n);
Y23 = conv(y,BPF23);

y_fft23 = fft(Y23,N);
y_fft23 = fftshift(y_fft23);
y_fft23 = abs(y_fft23/N);
y_rms(2,3) = rms(y_fft23);


BPF24 = cos(2*pi/Fs*F(2,4)*n);
Y24 = conv(y,BPF24);

y_fft24 = fft(Y24,N);
y_fft24 = fftshift(y_fft24);
y_fft24 = abs(y_fft24/N);
y_rms(2,4) = rms(y_fft24);
```

```
[M,I] = max(y_rms.');
if M(1)==0&&M(2)==0
    fprintf('No Signal');
elseif I(1)==1&&I(2)==1
    fprintf('Detected sequence = 1');
elseif I(1)==1&&I(2)==2
    fprintf('Detected sequence = 2');
elseif I(1)==1&&I(2)==3
    fprintf('Detected sequence = 3');
elseif I(1)==1&&I(2)==4
    fprintf('Detected sequence = A');
elseif I(1)==2&&I(2)==1
    fprintf('Detected sequence = 4');
elseif I(1)==2&&I(2)==2
    fprintf('Detected sequence = 5');
elseif I(1)==2&&I(2)==3
    fprintf('Detected sequence = 6');
elseif I(1)==2&&I(2)==4
    fprintf('Detected sequence = B');
elseif I(1)==3&&I(2)==1
    fprintf('Detected sequence = 7');
elseif I(1)==3&&I(2)==2
    fprintf('Detected sequence = 8');
elseif I(1)==3&&I(2)==3
    fprintf('Detected sequence = 9');
elseif I(1)==3&&I(2)==4
    fprintf('Detected sequence = C');
elseif I(1)==4&&I(2)==1
    fprintf('Detected sequence = *');
elseif I(1)==4&&I(2)==2
    fprintf('Detected sequence = 0');
elseif I(1)==4&&I(2)==3
    fprintf('Detected sequence = #');
elseif I(1)==4&&I(2)==4
    fprintf('Detected sequence = D');
end
fprintf("\r");

end
```

Command Prompt:





Graphical User Interface:

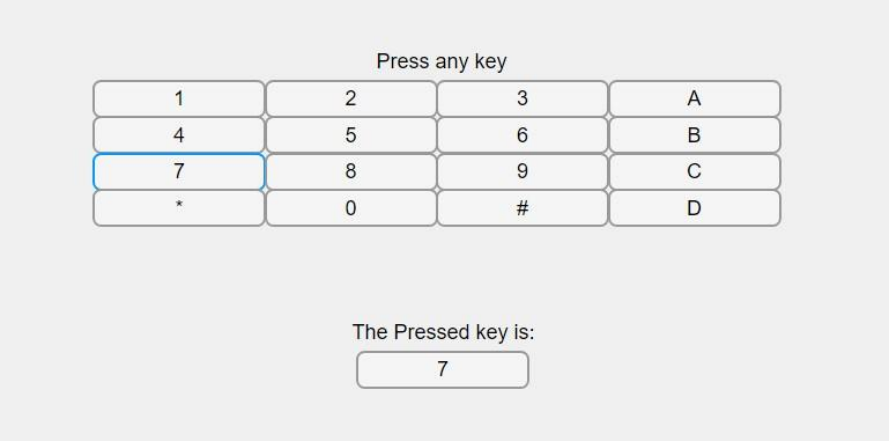
CODE:

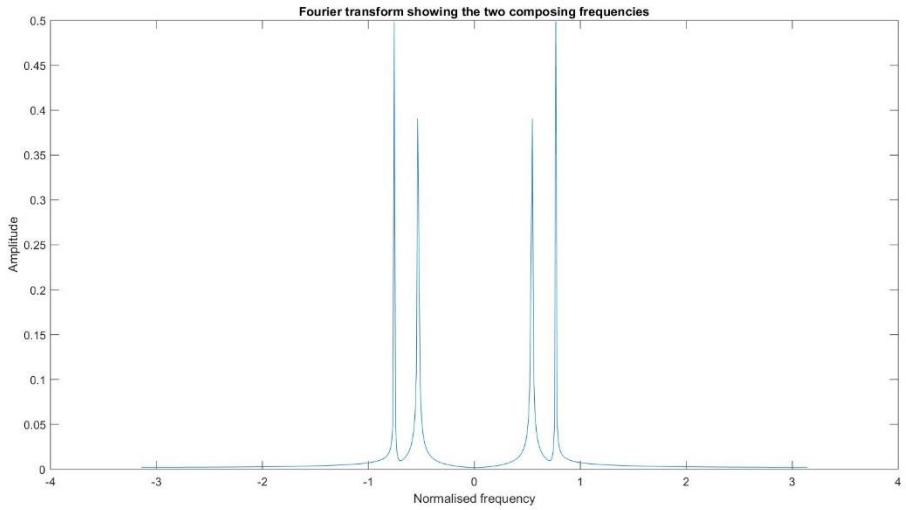
```
classdef app1 < matlab.apps.AppBase
    % Properties that correspond to app components
    properties (Access = public)
        UIFigure              matlab.ui.Figure
        Button                matlab.ui.control.Button
        Button_2              matlab.ui.control.Button
        Button_3              matlab.ui.control.Button
        AButton               matlab.ui.control.Button
        Button_4              matlab.ui.control.Button
        Button_5              matlab.ui.control.Button
        Button_6              matlab.ui.control.Button
        BButton               matlab.ui.control.Button
        Button_7              matlab.ui.control.Button
        Button_8              matlab.ui.control.Button
        Button_9              matlab.ui.control.Button
        CButton               matlab.ui.control.Button
        Button_10             matlab.ui.control.Button
        Button_11             matlab.ui.control.Button
        Button_12             matlab.ui.control.Button
        DButton               matlab.ui.control.Button
        Button_13             matlab.ui.control.StateButton
        PressanykeyLabel      matlab.ui.control.Label
        ThePressedkeyisLabel  matlab.ui.control.Label
    end

    % App initialization and construction
    methods (Access = private)
        % Create UIFigure and components
        function createComponents(app)
            % Create UIFigure
            app.UIFigure = uifigure;
            app.UIFigure.Position = [100 100 640 480];
            app.UIFigure.Name = 'UI Figure';
            % Create Button
            app.Button = uibutton(app.UIFigure, 'push');
            app.Button.Position = [63 408 100 22];
            app.Button.Text = '1';
            % Create Button_2
            app.Button_2 = uibutton(app.UIFigure, 'push');
            app.Button_2.Position = [162 408 100 22];
            app.Button_2.Text = '2';
            % Create Button_3
            app.Button_3 = uibutton(app.UIFigure, 'push');
            app.Button_3.Position = [261 408 100 22];
            app.Button_3.Text = '3';
            % Create AButton
            app.AButton = uibutton(app.UIFigure, 'push');
            app.AButton.Position = [360 408 100 22];
```

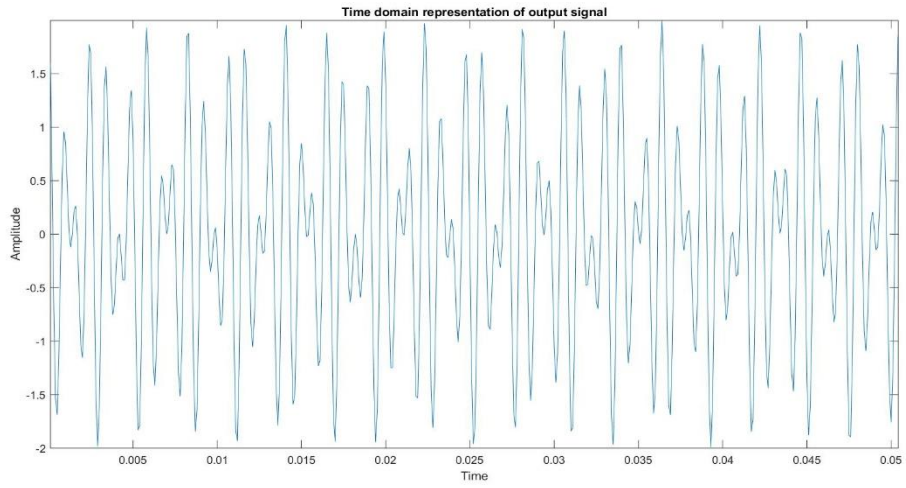
```
app.AButton.Text = 'A';
% Create Button_4
app.Button_4 = uibutton(app.UIFigure, 'push');
app.Button_4.Position = [63 387 100 22];
app.Button_4.Text = '4';
% Create Button_5
app.Button_5 = uibutton(app.UIFigure, 'push');
app.Button_5.Position = [162 387 100 22];
app.Button_5.Text = '5';
% Create Button_6
app.Button_6 = uibutton(app.UIFigure, 'push');
app.Button_6.Position = [261 387 100 22];
app.Button_6.Text = '6';
% Create BButton
app.BButton = uibutton(app.UIFigure, 'push');
app.BButton.Position = [360 387 100 22];
app.BButton.Text = 'B';
% Create Button_7
app.Button_7 = uibutton(app.UIFigure, 'push');
app.Button_7.Position = [63 366 100 22];
app.Button_7.Text = '7';
% Create Button_8
app.Button_8 = uibutton(app.UIFigure, 'push');
app.Button_8.Position = [162 366 100 22];
app.Button_8.Text = '8';
% Create Button_9
app.Button_9 = uibutton(app.UIFigure, 'push');
app.Button_9.Position = [261 366 100 22];
app.Button_9.Text = '9';
% Create CButton
app.CButton = uibutton(app.UIFigure, 'push');
app.CButton.Position = [360 366 100 22];
app.CButton.Text = 'C';
% Create Button_10
app.Button_10 = uibutton(app.UIFigure, 'push');
app.Button_10.Position = [63 345 100 22];
app.Button_10.Text = '*';
% Create Button_11
app.Button_11 = uibutton(app.UIFigure, 'push');
app.Button_11.Position = [162 345 100 22];
app.Button_11.Text = '0';
% Create Button_12
app.Button_12 = uibutton(app.UIFigure, 'push');
app.Button_12.Position = [261 345 100 22];
app.Button_12.Text = '#';
% Create DButton
app.DButton = uibutton(app.UIFigure, 'push');
app.DButton.Position = [360 345 100 22];
app.DButton.Text = 'D';
% Create Button_13
app.Button_13 = uibutton(app.UIFigure, 'state');
app.Button_13.Text = '7';
app.Button_13.Position = [215 252 100 22];
```

```
            % Create PressanykeyLabel
            app.PressanykeyLabel = uilabel(app.UIFigure);
            app.PressanykeyLabel.HorizontalAlignment = 'center';
            app.PressanykeyLabel.Position = [224 429 81 22];
            app.PressanykeyLabel.Text = 'Press any key';
            % Create ThePressedkeyisLabel
            app.ThePressedkeyisLabel = uilabel(app.UIFigure);
            app.ThePressedkeyisLabel.HorizontalAlignment = 'center';
            app.ThePressedkeyisLabel.Position = [210 273 111 22];
            app.ThePressedkeyisLabel.Text = 'The Pressed key is:';
        end
    end
    methods (Access = public)
        % Construct app
        function app = app1
            % Create and configure components
            createComponents(app)
            % Register the app with App Designer
            registerApp(app, app.UIFigure)
            if nargout == 0
                clear app
            end
        end
        % Code that executes before app deletion
        function delete(app)
            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

Output:







Discussion:

- These frequency values are not one's multiples so that no overlap will occur between at the receiver side. Multi-frequency signalling is a group of signalling methods that use a mixture of two pure sine waves sounds.
- The band pass filter we used here is impulse response given is:

$$h[n] = \beta \cos(\omega c * n) \ \ where \ 0 \leq n < L$$

  where N is the length of filter & ωc is the centre frequency.
- We used the sampling frequency far more than the maximum frequency component listed, so that the error we expected will be lessen.
- The output of the filter bank consists of frequency components which the bandpass filter for which the amplitude when compared to the other is higher.
- The GUI helps in easy visualisation of the code and how the telephonic DTMF works.

  Published with MATLAB® R2018a

Digital Signal Processing Lab

Expt. No. 4
Power Spectrum Estimation

By Pranit Dalal
Roll no. 16EC10016
Group 22 (Tuesday)

AIM:

Estimation of power spectral density using

a) The Welch Nonparametric Method: Averaging Modified Periodogram
b) Parametric method: Yule-Walker AR model

A. The Welch Nonparametric Method: Averaging Modified Periodogram

THEORY:

In non-parametric method, the way the data was taken is of no concern, hence the term non-parametric.

The steps of Welch method are:

1. x(n) is divided into L overlapping blocks, each block of length M with D samples common between two successive blocks as, $x_i(n)=x(n+iD)$ n=0,1,2,3,.....,M-1 i=0, 1,.....,L-1 .

2. The estimated PSD of the i^{th} block is obtained by computing the periodogram of that block which is

$$\tilde{P}_{xx}^{(i)}(f) = \frac{1}{MU}\left|\sum_{n=0}^{M-1} x_i(n)w(n)e^{-j2\pi fn}\right|^2 \qquad i = 0,1,\dots,L-1$$

where w(n) is the window function of length M (usually, a Hamming window) and U is a normalization factor for the power in the window function defined as

$$U = \frac{1}{M}\sum_{n=0}^{M-1} w^2(n)$$

3. Now the Welch spectrum estimate is obtained by the average of these modified periodograms of all blocks, i.e.,

$$P_{xx}^{W}(f) = \frac{1}{L}\sum_{i=0}^{L-1} \tilde{P}_{xx}^{(i)}(f)$$

4. The estimated PSD (as above) is plotted and compared with the actual PSD using pwelch function in MATLAB.

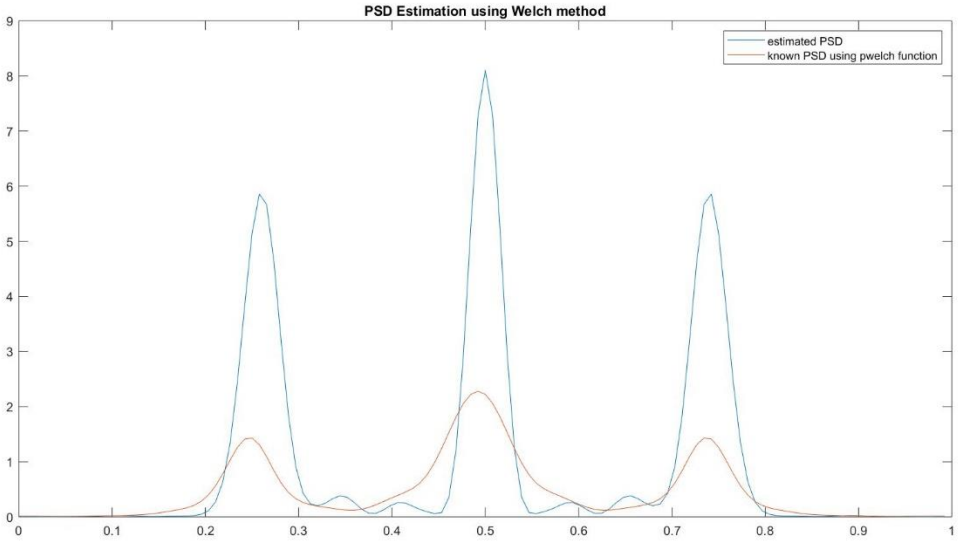
CODE:

```
clear all;
N = 128;
k = 1024;
r = randn(k,1);
L = 63;
%var = 1;  %since sigma is considered 1(Standard deviation)
a = [1 -0.9 0.81 -0.729];
x = filter(1,a,r);
M = 2*k/(L+1);
D = M/2;
xi = zeros(L,M);

for i = 0:L-1
    for j = 0:M-1
        xi(i+1,j+1) = x(j+1+i*D);
    end
end

w = hamming(M);
U = 0;
 for i = 1:M
     U = U+w(i)*w(i);
 end
 xi = xi.*w.';
 U = U/M;
 f = 0:1/N:(N-1)/N;
 P = fft(xi,N,2);
 P = fftshift(P);
 P = abs(P);
 P = P.^2;
 P = 0.16*P/(M*U);  %.16 is normalisation factor for D = M/2;

Z = zeros(1,N);

for i = 1:N
    for j = 1:L
```

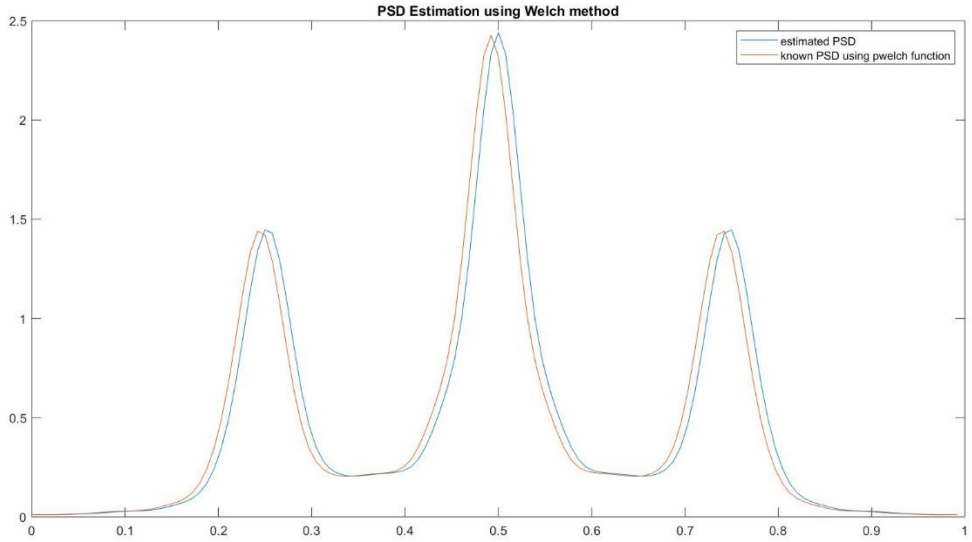
```
        Z(i) = Z(i) + P(j,i);
    end
    Z(i) = Z(i)/L;
end
figure;
plot(f,Z);

[pxx,s] = pwelch(x,w,D,N,'centered');
hold on;
plot(f,pxx);
```

PLOTS:

D=0(no overlap)



D=M/2(50% overlap)



B. Parametric method: Yule-Walker AR model

THEORY:

Parametric methods are based on modelling the data sequence by a rational system function of the form

$$H(z) = \frac{\sum_{k=0}^{q} b_k z^{-k}}{1+\sum_{k=1}^{p} a_k z^{-k}}$$

In the AR (auto regressive) process of order p, q=0 and b0=1, i.e. numerator is equal to 1.

The modern parametric method for PSD estimation exploits the prior knowledge available regarding the source of data generation. It assumes a model for that source and estimates the parameters for that particular model. No windowing occurs in this method. This is particularly useful when only a short data segment is available. The simplest model that is used in practice is a p^{th} order AR model described by the transfer function

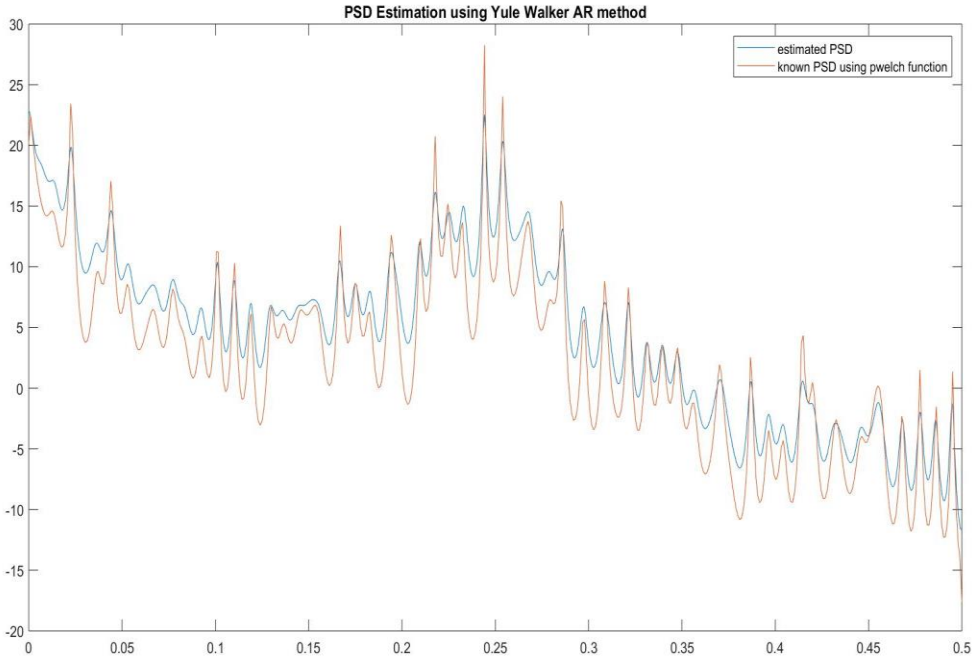
$$H(z) = \frac{1}{1+\sum_{k=1}^{p} a_k z^{-k}}$$

The data sequence is assumed to be generated by passing a zero mean white random sequence to the above filter.

CODE:

```
k = 1024;
p = 128;
N = 128;
r = randn(k,1);
n = 1:k;
a = [1 -0.9 0.81 -0.729];
x = filter(1,a,r);
[y,b] = autocorr(x,p);
R = zeros(p);
for i = 1:p
    for j = 1:p
        R(i,j) = y(abs(j-i)+1);
    end
end
Y = y(2:p+1);
A = -1*(inv(R))*Y;
var = 0;
for i=1:p
    var = var+A(i)*Y(i);
end
var = var+y(1);
var = sqrt(var);
h = zeros(p+1,1);
h(1) = 1;
h(2:p+1) = A(1:p);
h = h.';
[H,F] = freqz(1,h,k);
plot(F/(2*pi),20*log10(var.*abs(H)));
Z = filter(1,h,r);
[Pxx,F] = pyulear(Z,128,k,1);
hold on;
plot(F,10*log10(Pxx));
```

PLOTS:



DISCUSSIONS:

1. The power spectral density is the Fourier transform of autocorrelation.
2. In welch non-parametric method, we divide the data into smaller blocks and estimate the power spectral density using a hamming window.
3. In Yule-Walker AR model no windowing occurs and this is particularly used when short data segment is available.
4. Yule-Walker uses the AR-model parameters to calculate the Power Spectral Density (PSD). In this it is assumed that white random noise is passed through a filter to produce a random sequence.
5. We can calculate the actual PSD of the sequence because we have the filter coefficients.

Digital Signal Processing Lab

Expt. No. 5
Adaptive Line Enhancer

By Pranit Dalal
Roll no. 16EC10016
Group 22 (Tuesday)

AIM:

Reduction of Gaussian noise in a sinusoidal signal using Adaptive Line Enhancer (ALE).

THEORY:

An adaptive line enhancer (ALE) is used to detect a low-level sine wave of unknown frequency in presence of noise. If the input frequency changes the filter adapts itself to be a bandpass filter centered at the input frequency. The ALE is usually realized by using the so-called adaptive filter.

The most widely used adaptation algorithm is the Least Mean Square (LMS) algorithm, which uses the error signal e(n) in a feedback loop for coefficient adaptation.

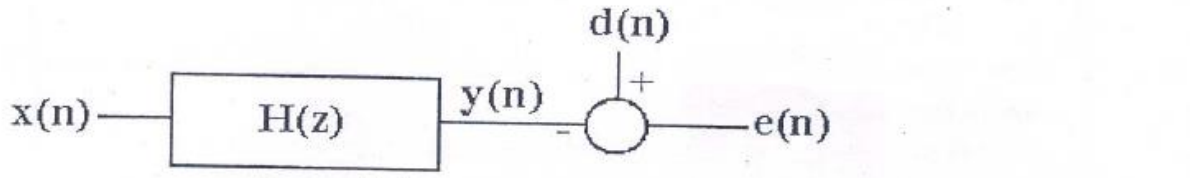


Fig.1: $d(n)$ is the desired response, $e(n)$ is the error.
The transfer function, $H(z) = w_0 + w_1 z^{-1} + w_2 z^{-2} + \ldots + w_p z^{-p}$

In an adaptive line enhancer, the desired response is simply the input x(n).

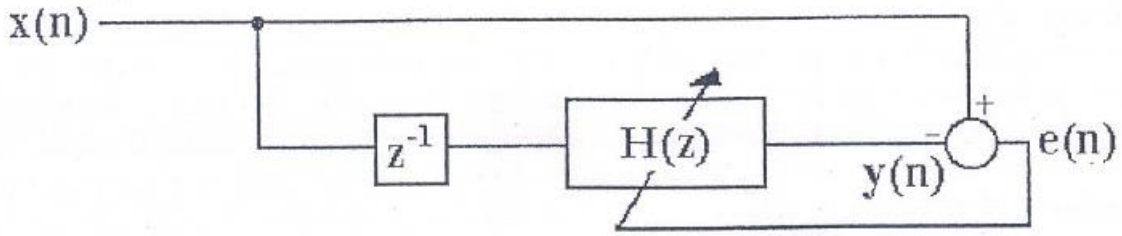


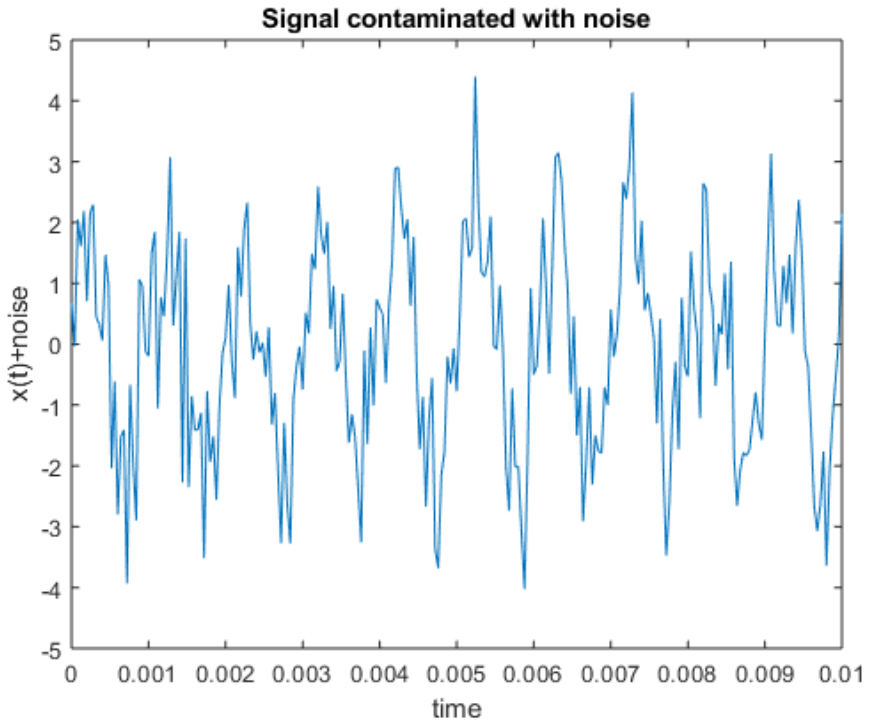
Fig.2: An adaptive line enhancer

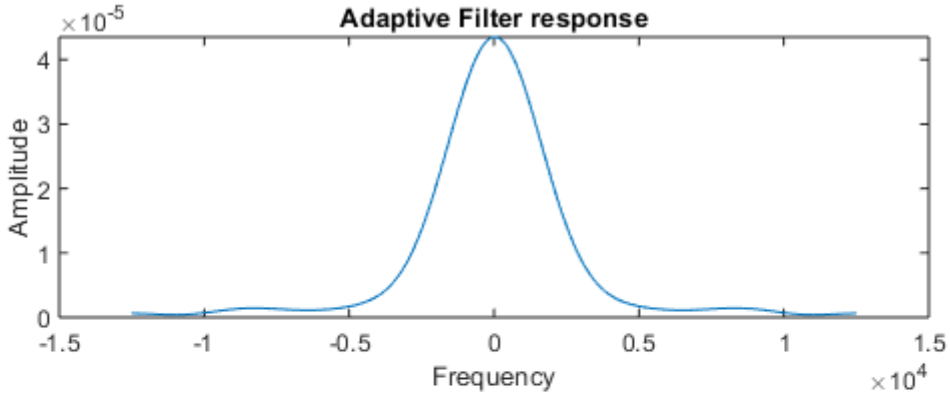
CODE:

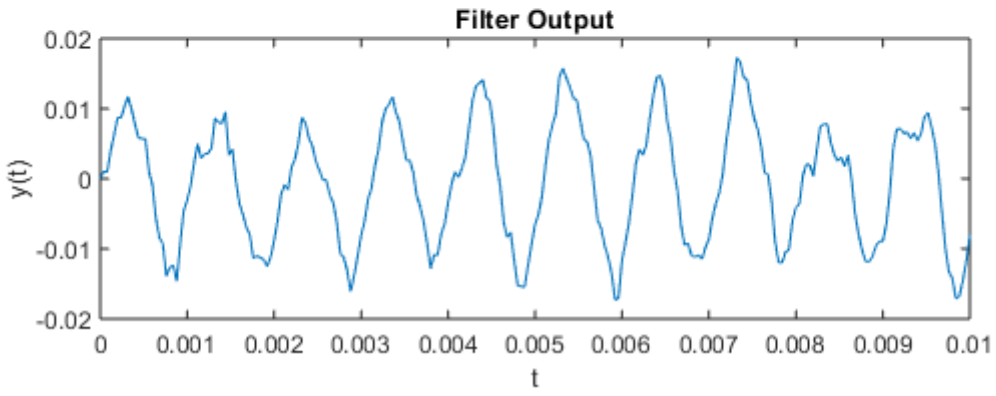
```
Fo= 1000;  % this is changed to 2,3,10Khz
Fs= 25*Fo;
t = 0:1/Fs:10/Fo;
m = 2*sin(2*pi*Fo*t)+randn(size(t)); %A=2
figure(1);
plot(t,m);
xlabel('time');
ylabel('x(t)+noise');
title('Signal contaminated with noise');

mu = 10^-4;
err_max = 10^-3;
N=256;
w = zeros(N,1);
error = 1;
x = zeros(N,1);
x_new = zeros(N,1);
i = 1;
while(error > err_max)     % LMS Algorithm
 y = w'*x;
 e = m(i) - y;
 w_new = w + mu*x*e;
 if(i == 1)
    error = 1;
 else
    error = (sumsqr(w_new-w))/(sumsqr(w));
 end
 w = w_new;
 x(1) = m(i);
 for k = 2:N
 x_new(k) = x(k-1);
 end
 x = x_new;
 i = i+1;
end
Z=(abs(fftshift(fft(w,512)))).^2;
f = (Fs/2)*linspace(-1,1,512);
figure(2);
subplot(211);
plot(f,Z);
xlabel('Frequency');
ylabel('Amplitude');
title('Adaptive Filter response');
subplot(212);
P=filter(w,1,m);
plot(t,P);
xlabel('t');
ylabel('y(t)');
title('Filter Output');
```

PLOTS:

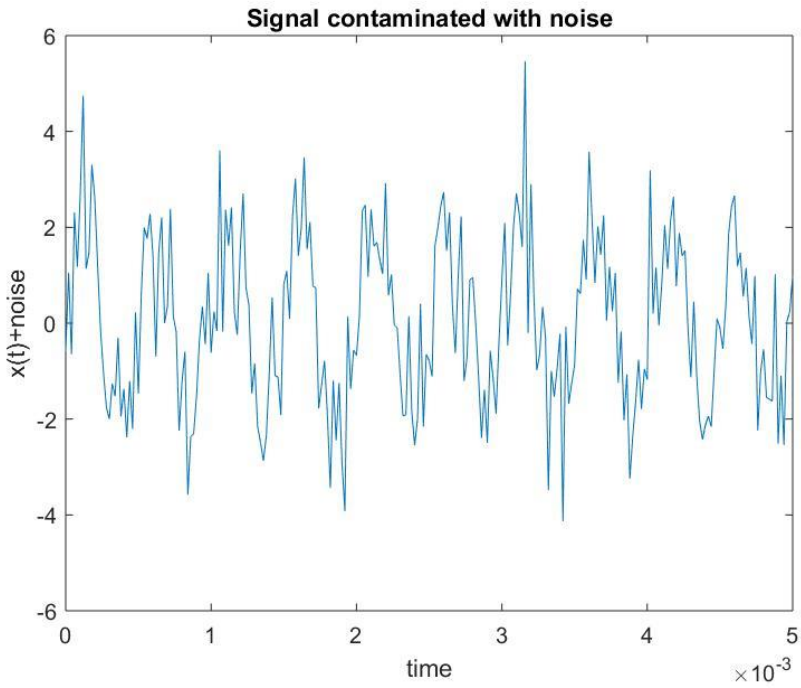
F_0 = 1000Hz

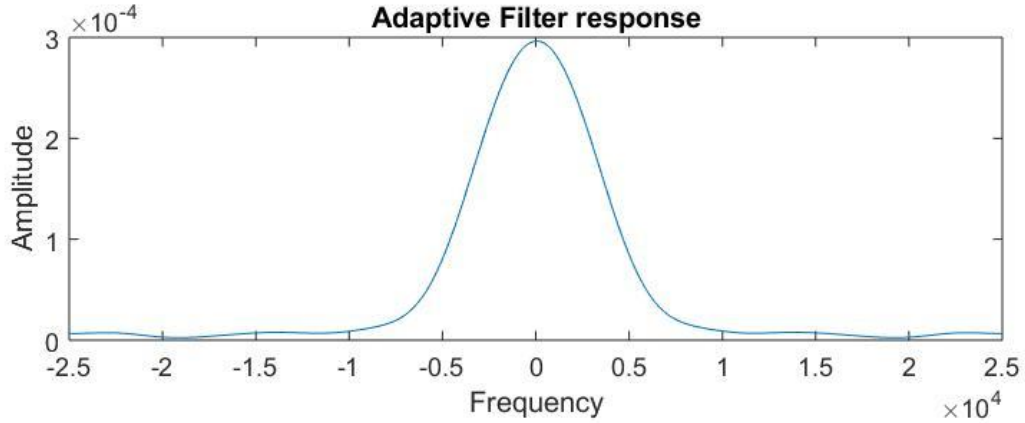


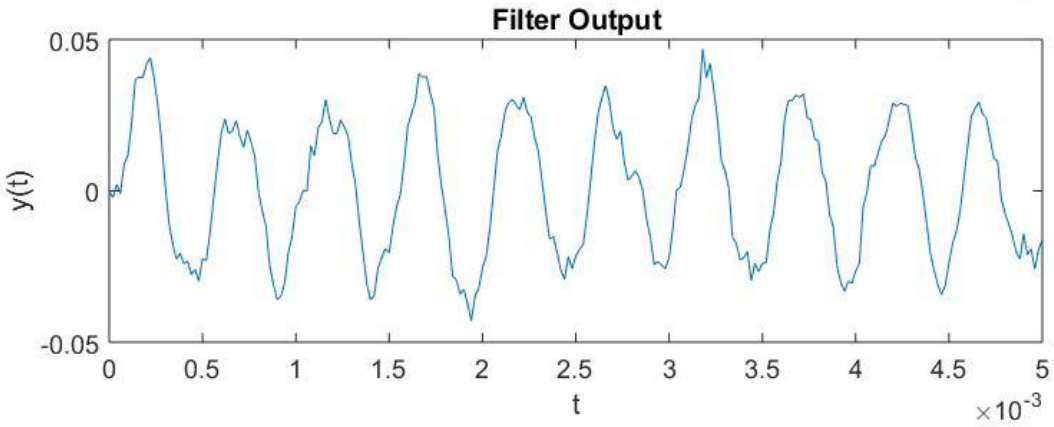




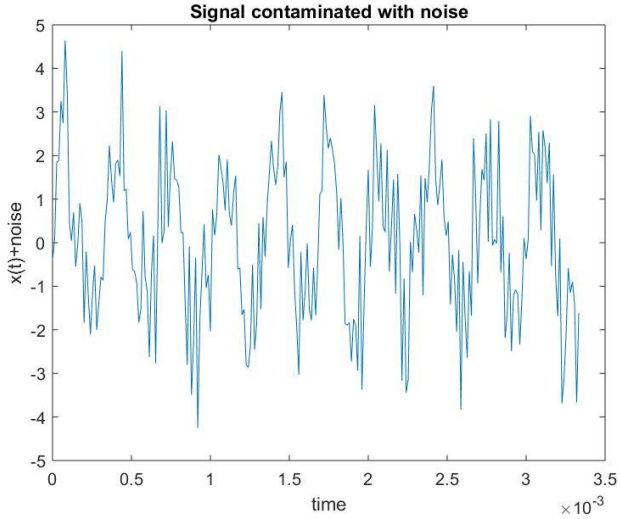
F_0 = 2000Hz

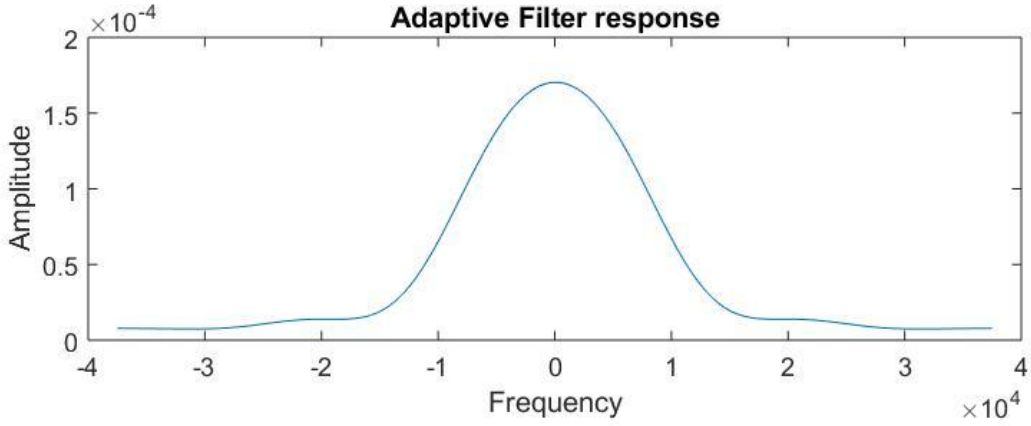

Signal contaminated with noise
x(t)+noise
time
×10^-3

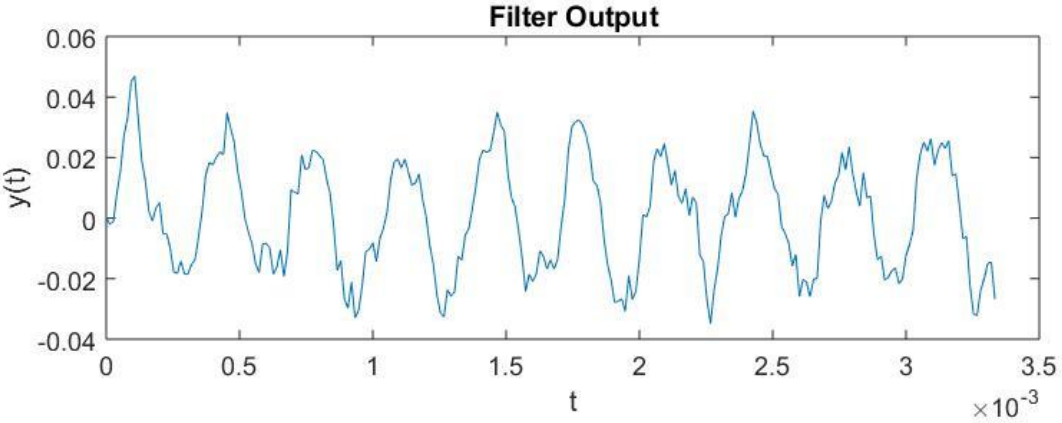

Adaptive Filter response
×10^-4
Amplitude
Frequency
×10^4


Filter Output
y(t)
t
×10^-3

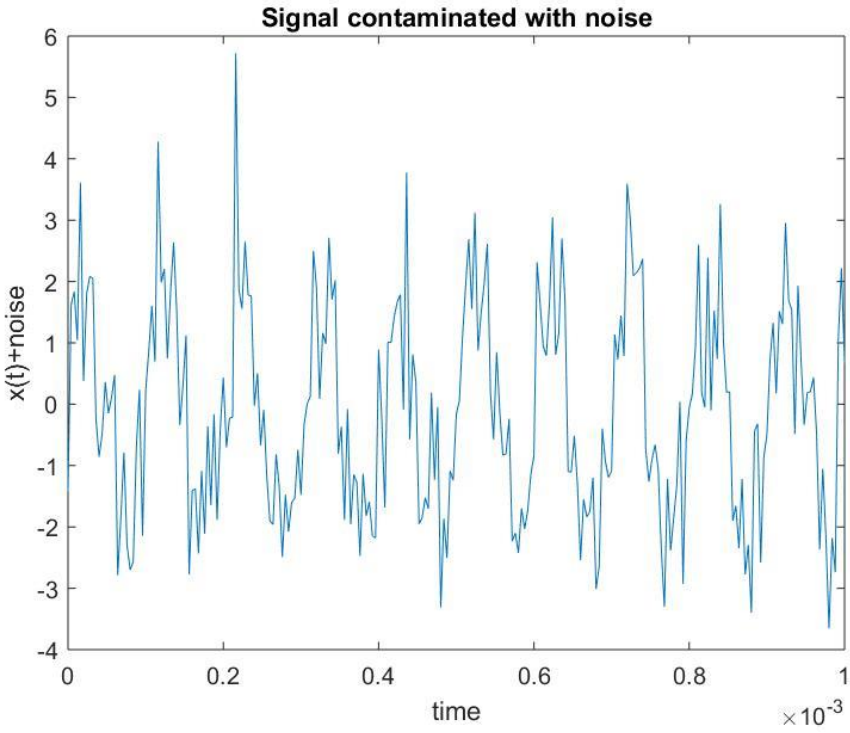
F0 = 3000Hz

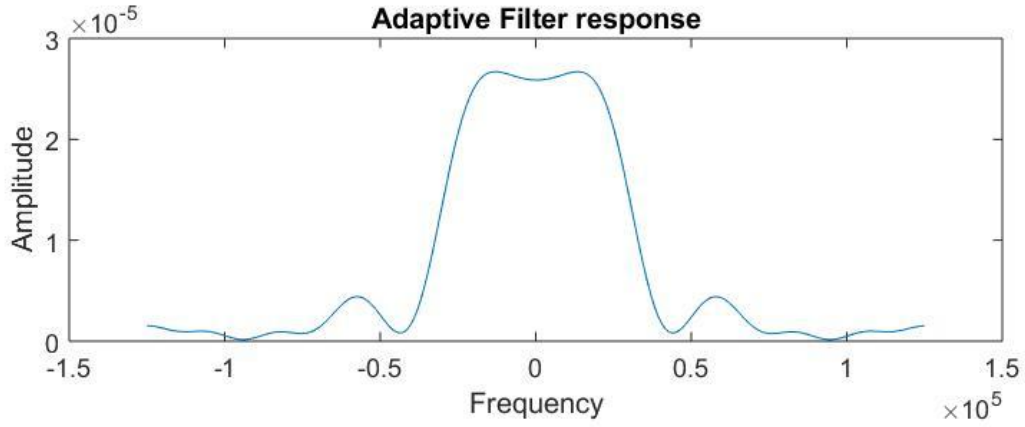

Signal contaminated with noise
x(t)+noise
time
×10^-3

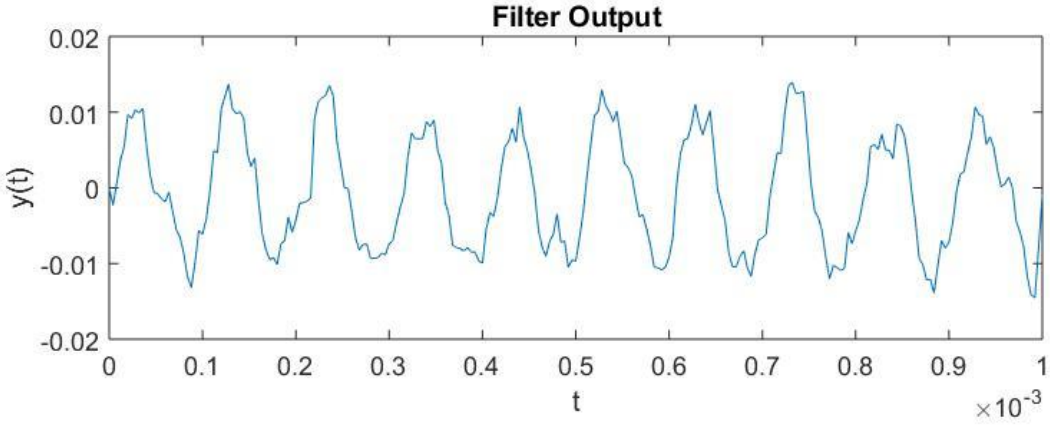

Adaptive Filter response
×10^-4
Amplitude
Frequency
×10^4


Filter Output
y(t)
t
×10^-3

F0 = 10000Hz


Signal contaminated with noise
x(t)+noise
time
×10^-3


Adaptive Filter response
×10^-5
Amplitude
Frequency
×10^5


Filter Output
y(t)
t
×10^-3

DISCUSSIONS:

1. We can clearly observe the filter response changing for different frequencies and also for different iteration of noise.
2. An ALE (Adaptive Line Enhancer) consists of the interconnection of delay element and a linear predictor and has an adaptive algorithm.
3. In an adaptive filter the filter coefficients are updated in time by using an adaptive algorithm so that the filter output becomes nearly same as the desired output minimizing the error. (done using LMS algorithm).
4. We can also recover high frequency signals by this method but for recovering high frequency signals we need to sample the input sequence with high sampling frequency to satisfy Nyquist criterion.
5. The filter can be interpreted as a 1-step linear predictor, i.e. it predicts the i^{th} sample from the past i-1 samples of the signal.
6. From plots we can observe that the output signal is nearly same as the input with reduction in noise which is to be achieved in the experiment.