DSP Lab Open Ended

(Course Code: UE17EE335)

Audio Signal Filtering using FIR Lowpass filter

By Ayush Das (USN: PES1201700504)

Sumanth V Udupa (USN: PES1201700525)

Overview

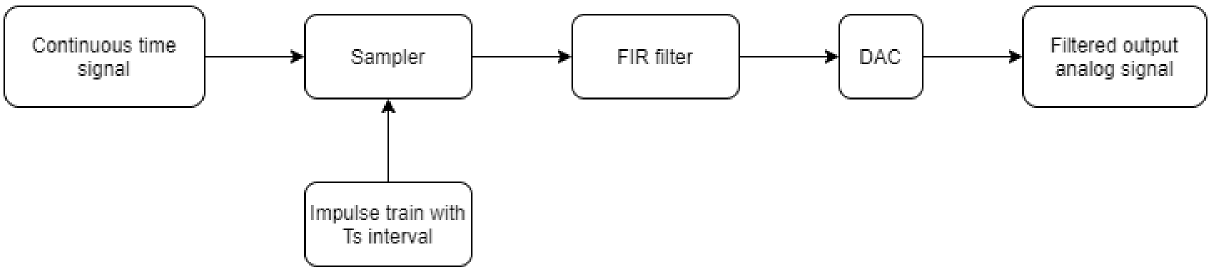
Audio signal processing is a subfield of signal processing that is concerned with the electronic manipulation of audio signals. Audio signals are electronic representations of sound waves—longitudinal waves which travel through air, consisting of compressions and rarefactions. The energy contained in audio signals is typically measured in decibels.

Applications

Audio processing enabled to open the doors to endless list of applications such as audio broadcasting, active noise control, audio synthesis and audio effects such as echo, flanger, phaser, chorus, equalization and de-esser.

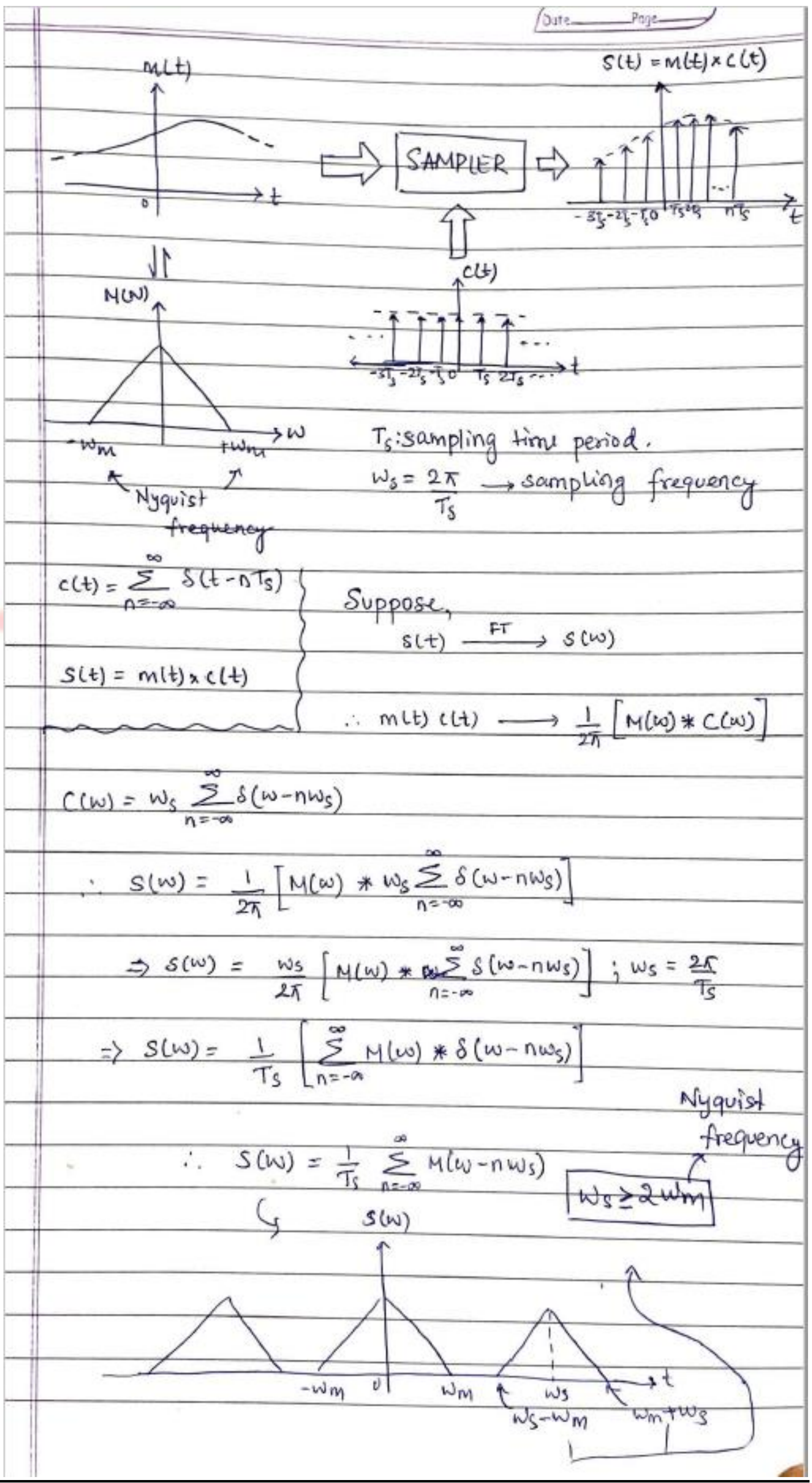
In this experiment we aim to implement filtering of noisy audio with the help of FIR lowpass filter.

Theory



Sampling theorem: The sampling frequency must be greater than or equal to twice the Nyquist frequency. So as to retain the quality of the signal.

Note: Here we have taken sampling frequency to be 44.1kHz as the Nyquist frequency is usually 8kHz for audio files.


Date Page
m(t)
0
t
SAMPLER
$s(t) = m(t) \times c(t)$
$-3T_s$ $-2T_s$ $-T_s$ 0 T_s $2T_s$ nT_s
t
c(t)
$-3T_s$ $-2T_s$ $-T_s$ 0 T_s $2T_s$
M(w)
$-w_M$
$+w_M$
w
Nyquist frequency
S(w)
$-w_m$
0
w_m
w_s
$w_s - w_m$
$w_m + w_s$
t

T_s: sampling time period.

$w_s = \frac{2\pi}{T_s} \longrightarrow$ sampling frequency

$$c(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT_s)$$

$$s(t) = m(t) \times c(t)$$

Suppose,

$$s(t) \xrightarrow{FT} S(w)$$

$$\therefore\ m(t)\, c(t) \longrightarrow \frac{1}{2\pi}\left[M(w) * C(w)\right]$$

$$C(w) = w_s \sum_{n=-\infty}^{\infty} \delta(w - nw_s)$$

$$\therefore\ S(w) = \frac{1}{2\pi}\left[M(w) * w_s \sum_{n=-\infty}^{\infty} \delta(w - nw_s)\right]$$

$$\Rightarrow S(w) = \frac{w_s}{2\pi}\left[M(w) * \sum_{n=-\infty}^{\infty} \delta(w - nw_s)\right] \ ;\ w_s = \frac{2\pi}{T_s}$$

$$\Rightarrow S(w) = \frac{1}{T_s}\left[\sum_{n=-\infty}^{\infty} M(w) * \delta(w - nw_s)\right]$$

$$\therefore\ S(w) = \frac{1}{T_s} \sum_{n=-\infty}^{\infty} M(w - nw_s)$$

$$\boxed{w_s \geq 2w_m}$$

Nyquist frequency

In signal processing, a finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying).

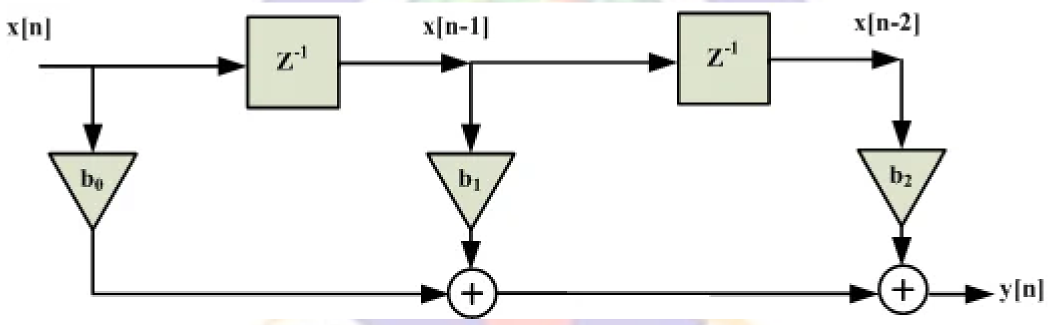
With design specifications known, we can find the transfer function which will provide the required filtering. The rational transfer function of a digital filter is as follows.

$$H(z) = \frac{\sum_{k=0}^{M-1} b_k z^{-k}}{\sum_{k=0}^{N-1} a_k z^{-k}}$$

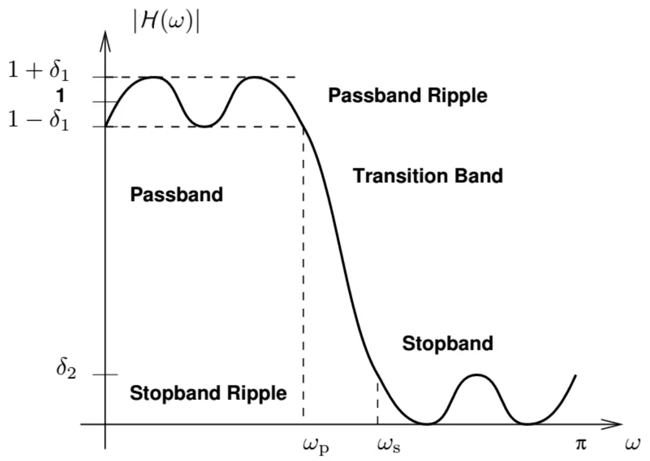
For FIR filter, $a_0 = 1$, $a_k = 0$ for k = 0,1,2,..,N. Hence we have

$$H(z) = \sum_{k=0}^{M-1} b_k z^{-k}$$

Below is the general block diagram of an FIR filter.



Low pass filter is a filter that passes all the frequencies below a designed filter cut-off frequency and attenuates the frequencies above the cut-off frequency.

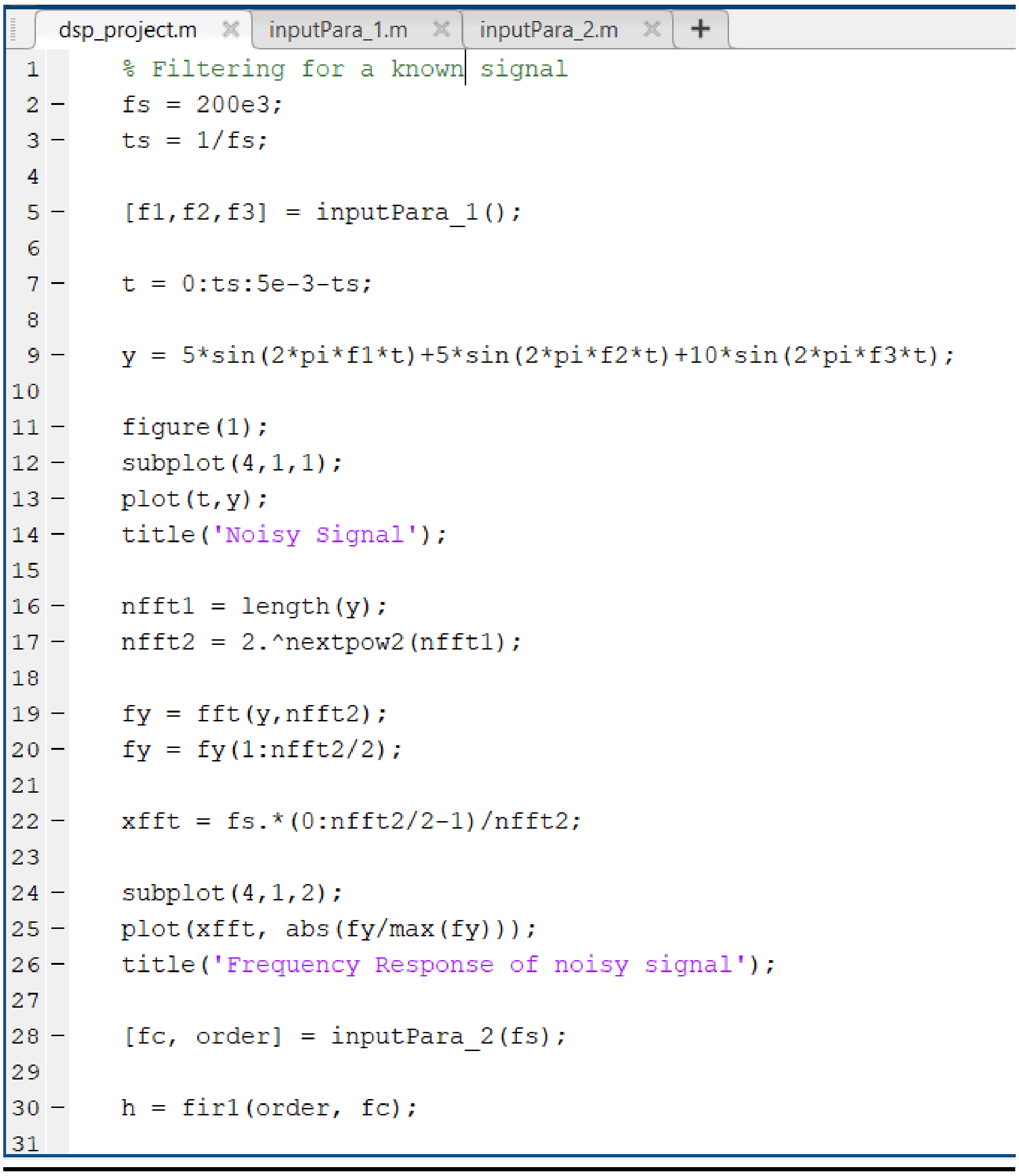


Above image shows the response of a FIR lowpass filter.

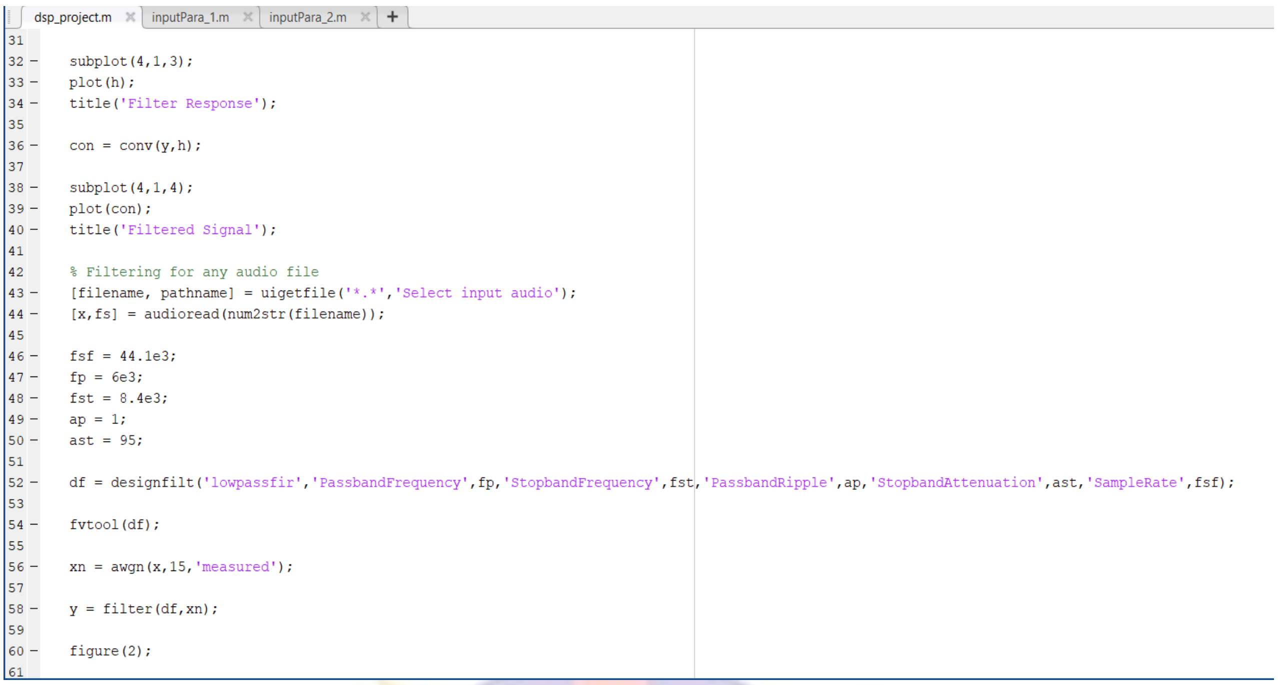
Matlab Code

Note: In the submitted code there are two different parts in the main file ***dsp_project.m***.

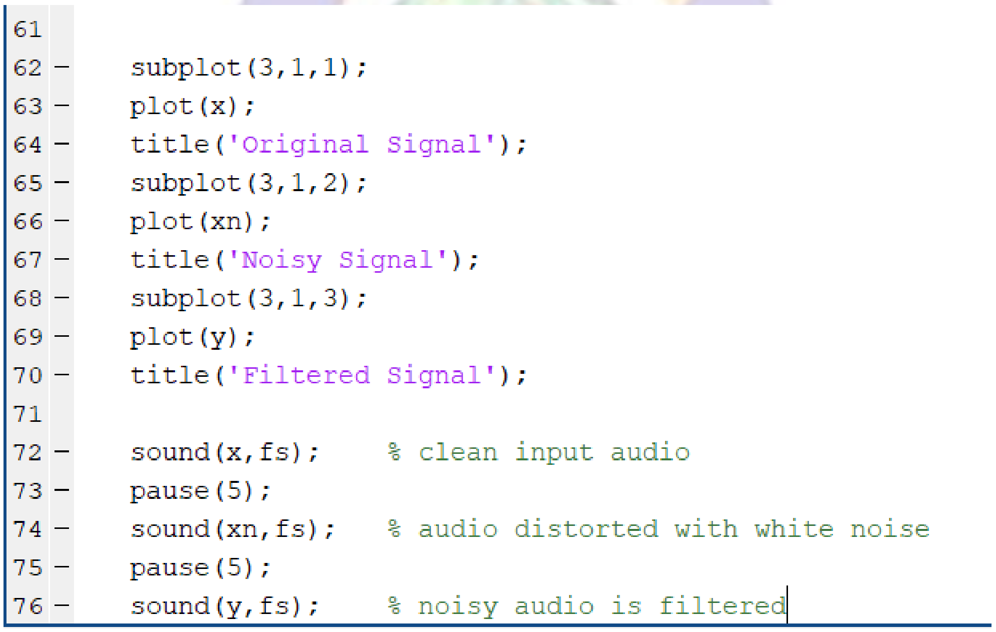
First part being filtering of a fixed known signal that is defined by the user. Second part being an audio file placed anywhere in the system is filtered (or lowpass-ed at 8.4kHz with fixed attenuations for passband and stopband for simplicity).



```
% Filtering for a known signal
fs = 200e3;
ts = 1/fs;

[f1,f2,f3] = inputPara_1();

t = 0:ts:5e-3-ts;

y = 5*sin(2*pi*f1*t)+5*sin(2*pi*f2*t)+10*sin(2*pi*f3*t);

figure(1);
subplot(4,1,1);
plot(t,y);
title('Noisy Signal');

nfft1 = length(y);
nfft2 = 2.^nextpow2(nfft1);

fy = fft(y,nfft2);
fy = fy(1:nfft2/2);

xfft = fs.*(0:nfft2/2-1)/nfft2;

subplot(4,1,2);
plot(xfft, abs(fy/max(fy)));
title('Frequency Response of noisy signal');

[fc, order] = inputPara_2(fs);

h = fir1(order, fc);

```

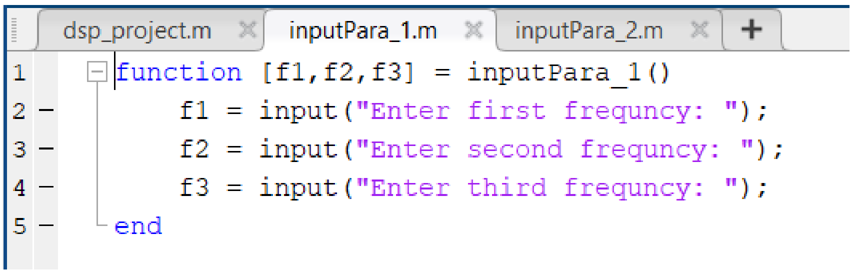


```
    subplot(4,1,3);
    plot(h);
    title('Filter Response');

    con = conv(y,h);

    subplot(4,1,4);
    plot(con);
    title('Filtered Signal');

    % Filtering for any audio file
    [filename, pathname] = uigetfile('*.*','Select input audio');
    [x,fs] = audioread(num2str(filename));

    fsf = 44.1e3;
    fp = 6e3;
    fst = 8.4e3;
    ap = 1;
    ast = 95;

    df = designfilt('lowpassfir','PassbandFrequency',fp,'StopbandFrequency',fst,'PassbandRipple',ap,'StopbandAttenuation',ast,'SampleRate',fsf);

    fvtool(df);

    xn = awgn(x,15,'measured');

    y = filter(df,xn);

    figure(2);

```

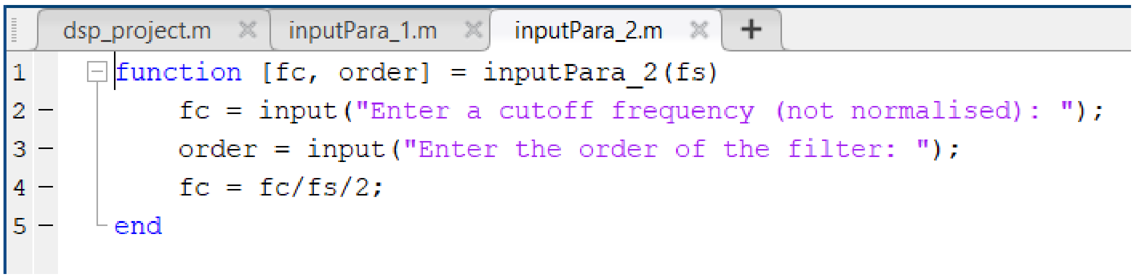


```

    subplot(3,1,1);
    plot(x);
    title('Original Signal');
    subplot(3,1,2);
    plot(xn);
    title('Noisy Signal');
    subplot(3,1,3);
    plot(y);
    title('Filtered Signal');

    sound(x,fs);     % clean input audio
    pause(5);
    sound(xn,fs);    % audio distorted with white noise
    pause(5);
    sound(y,fs);     % noisy audio is filtered
```

Below two screenshots show the functions that we use to input data from the user.



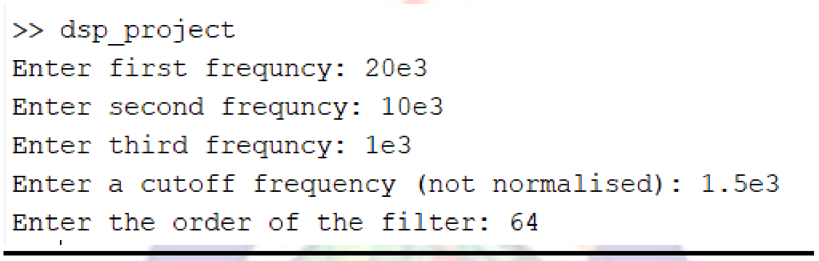
```
function [f1,f2,f3] = inputPara_1()
    f1 = input("Enter first frequncy: ");
    f2 = input("Enter second frequncy: ");
    f3 = input("Enter third frequncy: ");
end
```



```matlab
function [fc, order] = inputPara_2(fs)
    fc = input("Enter a cutoff frequency (not normalised): ");
    order = input("Enter the order of the filter: ");
    fc = fc/fs/2;
end
```

Output

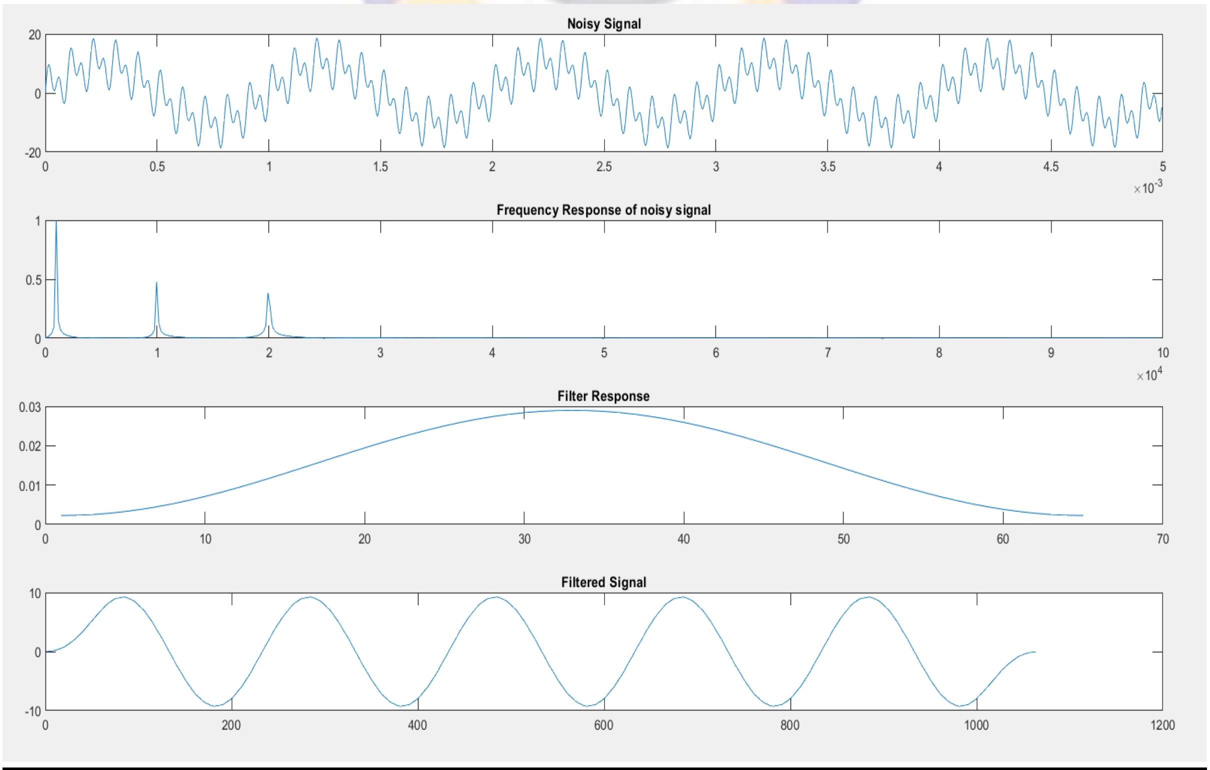
For the first part of the program we have a simple filtering of a known signal. Below shown picture shows that a combination of 20kHz, 10kHz and 1kHz is inputted as the fundamental frequency for the sinusoidal expression in variable ***y***.



```
>> dsp_project
Enter first frequncy: 20e3
Enter second frequncy: 10e3
Enter third frequncy: 1e3
Enter a cutoff frequency (not normalised): 1.5e3
Enter the order of the filter: 64
```

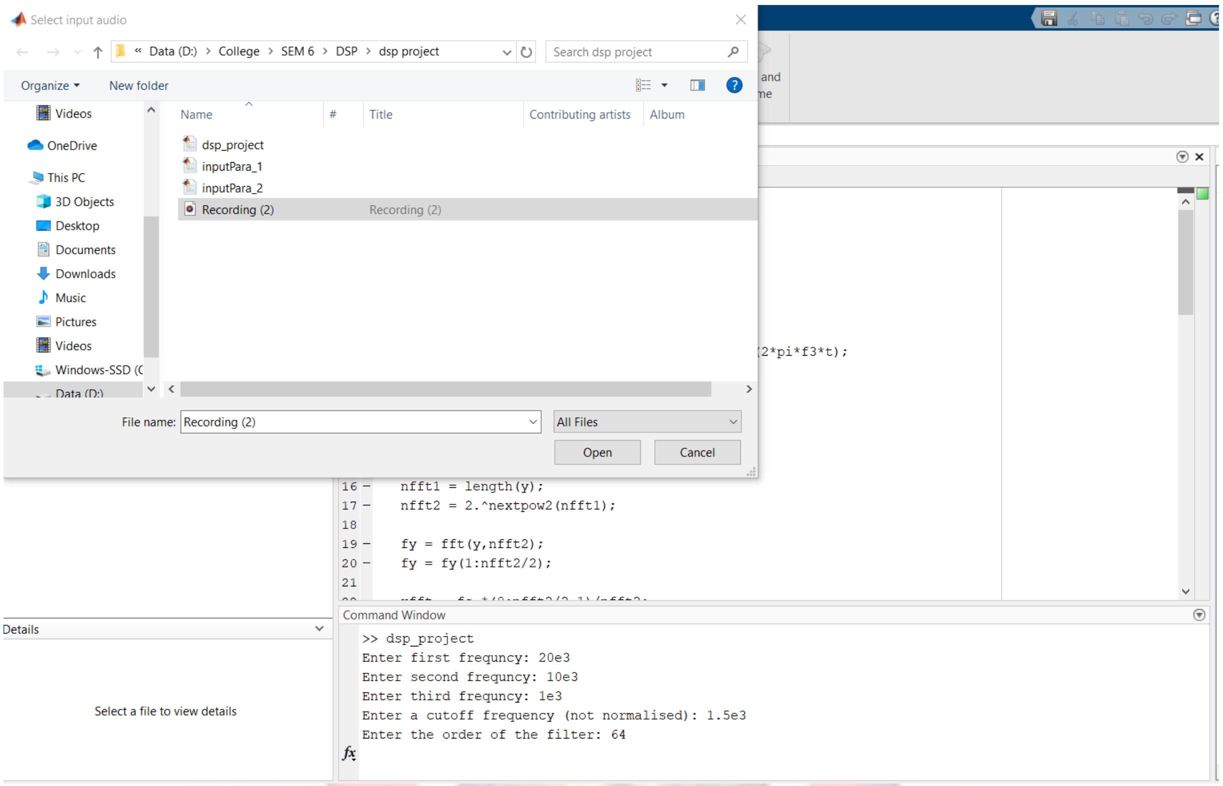
In the first graph shown below we can see the noisy signal that we generated based on our inputs.

Now analyse the frequency response of the signal and decide on which frequency to retain (in this case we inputted 1.5kHz as the cut-off frequency and 64 as the order to achieve a smooth signal).



Below screenshots refer to the second part of the program.

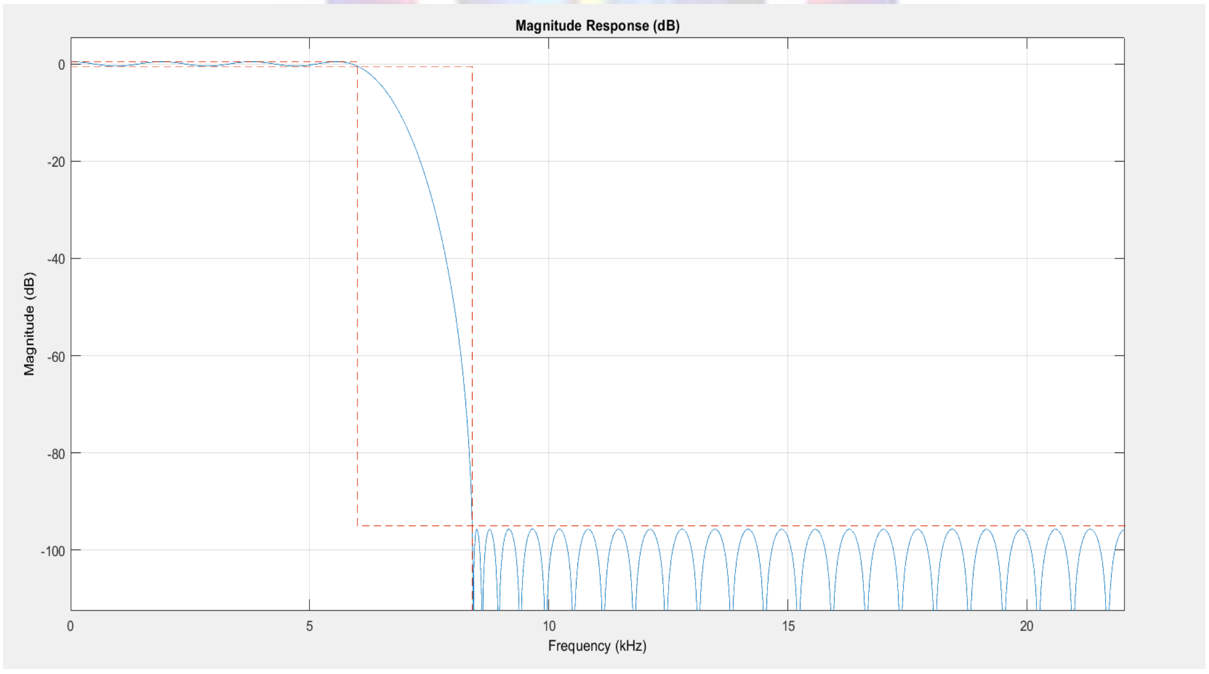
Here we input an audio file from our computer as the input.



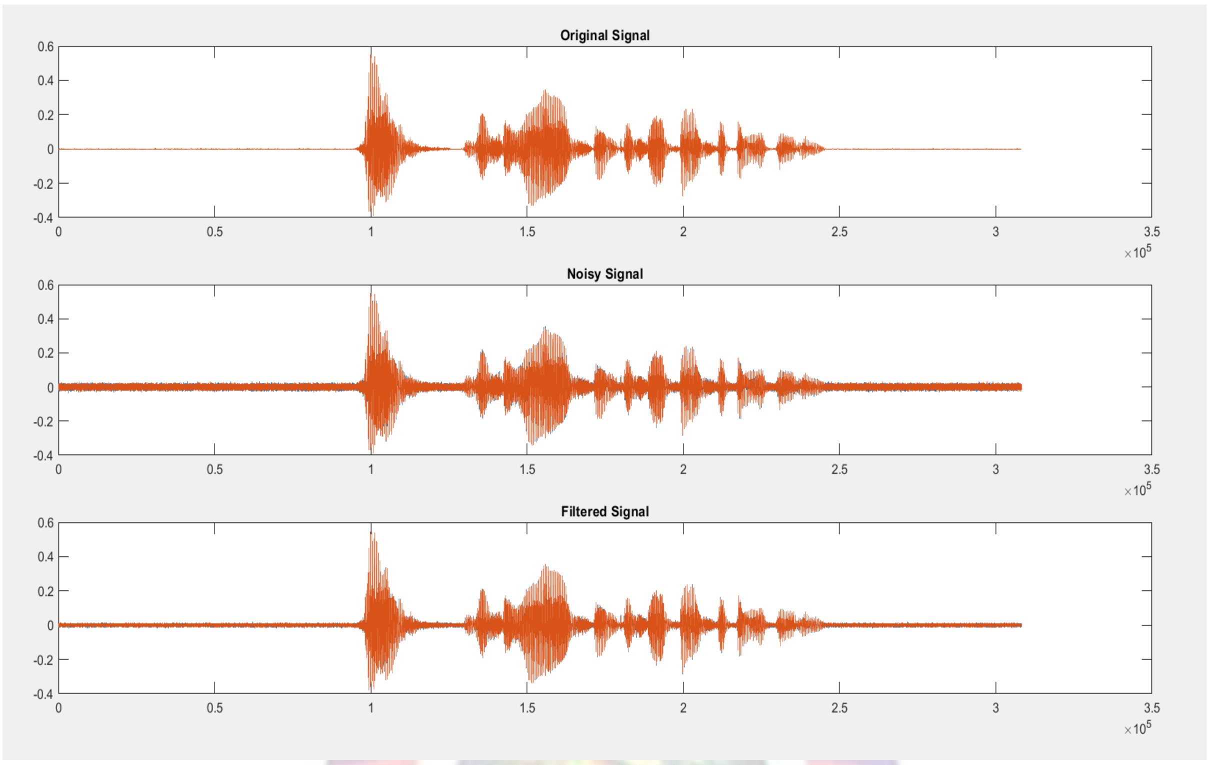
```
2*pi*f3*t);
nfft1 = length(y);
nfft2 = 2.^nextpow2(nfft1);

fy = fft(y,nfft2);
fy = fy(1:nfft2/2);
```

```
>> dsp_project
Enter first frequncy: 20e3
Enter second frequncy: 10e3
Enter third frequncy: 1e3
Enter a cutoff frequency (not normalised): 1.5e3
Enter the order of the filter: 64
fx
```

Below is the frequency response of a lowpass FIR filter.



The plots given below are obtained by plotting the samples from ***Recording (2).m4a*** file in among the attached files. But you have the freedom to choose any audio file that you want use.



The first plot is the plot of the original signal obtained by sampling at **44.1kHz**. Then to simulate a noisy signal I have added ***White Gaussian Noise*** to the signal. The Gaussian noise adds a lot more information which is especially noticed in the ***higher frequencies*** (from 8kHz onwards). Based on the above observation I had coded below the parameters for the filter so as to filter the noise in the signal which gives us the third plot above.

```
fsf = 44.1e3;
fp = 6e3;
fst = 8.4e3;
ap = 1;
ast = 95;
```

Where ,	fsf - sampling frequency
	fp – passband frequency
	fst – stopband frequency
	ap – passband ripple
	ast – stopband attenuation

Lastly, the below snippet of code plays three audio samples. The first one plays the original sample, the second time it plays the sample with noise and finally it plays the sample being filtered.

```
sound(x,fs);     % clean input audio
pause(5);
sound(xn,fs);    % audio distorted with white noise
pause(5);
sound(y,fs);     % noisy audio is filtered
```

Note: ***A video of the demonstration of the program is attached please look into for further understanding.***

Algorithm

First part

A fixed noisy signal is made using a combination of three sinusoidal waves of different fundamental frequencies.

Note: We are using sampling time ***ts*** as the interval to plot the signal.

The length of the signal is taken and the **next power of 2** is found using the nextpow2() function. This serves as the N point samples of which we find the ***fft()*** of. After obtaining the fft we take have of the samples into consideration as the sequence is symmetric and then we plot so that user identifies the prime frequencies of the signal and can decide which of these frequencies he/she wants to retain.

Then we use the ***fir1()*** function which finds the filter response by passing the order and cut-off frequency (which we then normalise by dividing by fs/2, *Nyquist Frequency*) as parameters. The obtained response is then convoluted with the noisy signal.

Second part

Using the first part of the program as the basis of foundation we move on to applying this logic onto audio signals.

We use function ***uigetfile()*** to open a window to choose the audio file that we would as the input for the program. Function ***audioread()*** is used to obtain the sampling frequency and the samples of the audio.

Note: In cases where we input two channel audio clips like in the demonstration the function will retrieve two columns of information one for each channel. For plotting it would superimpose the samples of both the channels to get a single plot.

We then use the below values which seemed to be the appropriate setting for the recording that I chose to use.

```
fsf = 44.1e3;    % sampling frequency
fp = 6e3;        % passband frequency
fst = 8.4e3;     % stop band frequency
ap = 1;          % passband ripple
ast = 95;        % stopband attenuation
```

Using the above parameters, we input them into the ***designfilt()*** which will then return us a filter object that is assigned to variable ***df***. We can use the ***fvtool()*** function to plot the FIR filter response by passing the filter object ***df*** as a parameter.

```
df = designfilt('lowpassfir','PassbandFrequency',fp,'StopbandFrequency',fst,'PassbandRipple',ap,'StopbandAttenuation',ast,'SampleRate',fsf);
```

To add ***White Gaussian Noise*** to the signal we use the ***awgn()*** function obtained from the ***Communication Toolbox*** in MATLAB.

Now we pass the filter object ***df*** and the noisy signal into the ***filter()*** function which will then return us the filtered audio sample.

Note: Using this method of approach was especially helpful for two channel audio signal inputs.

```
y = filter(df,xn);
```

Result

In the industry today, they use a more advanced approach to remove unwanted noise using wavelet method of filtering which is much more effective in identifying, isolating and supressing unwanted signals which has made Digital Signal Processing what it is today.

As an entry point to this vast field, I feel through this experiment an essence of the subject was appreciated and steps to move ahead in the field appears to be more clear.

To conclude, any audio file on the system when used as an input for the program was realized, sampled, noise was added to simulate real application scenarios and finally filtered with the concepts of FIR lowpass filter.