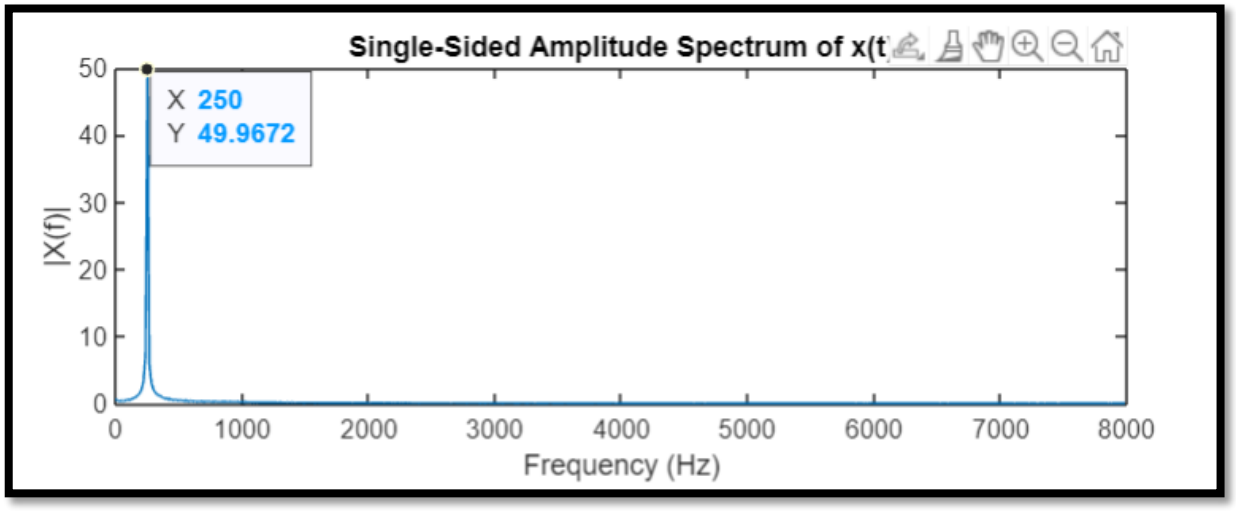
4C1 Integrated Systems Design

REPORT: Lab-3 FIR Filter Design

-Pulkit Agarwal (19323939)

Identifying Noise in the Sound



MATLAB Commands to manually find the frequency at maximum tone:

```
[maxYValue, indexAtMaxY] = max(2*abs(X(1:nfft/2)))
maxXValue = fvec(indexAtMaxY(1))
```

Using the above lines of code, I was able to capture the frequency of the tone that was added in the speech file.

Furthermore, below is the entire code snippet with comments that displays my understanding and its purpose.

```
[y, Fs] = audioread('speech_1.wav')
% Audioread returns a y:Fs matrix where
% 'y' represents number of audio channels in the file and
% 'Fs' represents number of audio samples read
nfft = 2^10; % nfft helps in comparing signals at different frequency
X = fft(y, nfft); % computing discrete fourier transform of 'y'
fstep = Fs/nfft; % Setting a step value to traverse through the audio samples
fvec = fstep*(0: nfft/2-1); %Calculates the input frequency specified as a vector

[maxYValue, indexAtMaxY] = max(2*abs(X(1:nfft/2)))
maxXValue = fvec(indexAtMaxY(1))

plot(fvec,2*abs(X(1:nfft/2)))
title('Single-Sided Amplitude Spectrum of x(t)')
xlabel('Frequency (Hz)')
ylabel('|X(f)|')
```

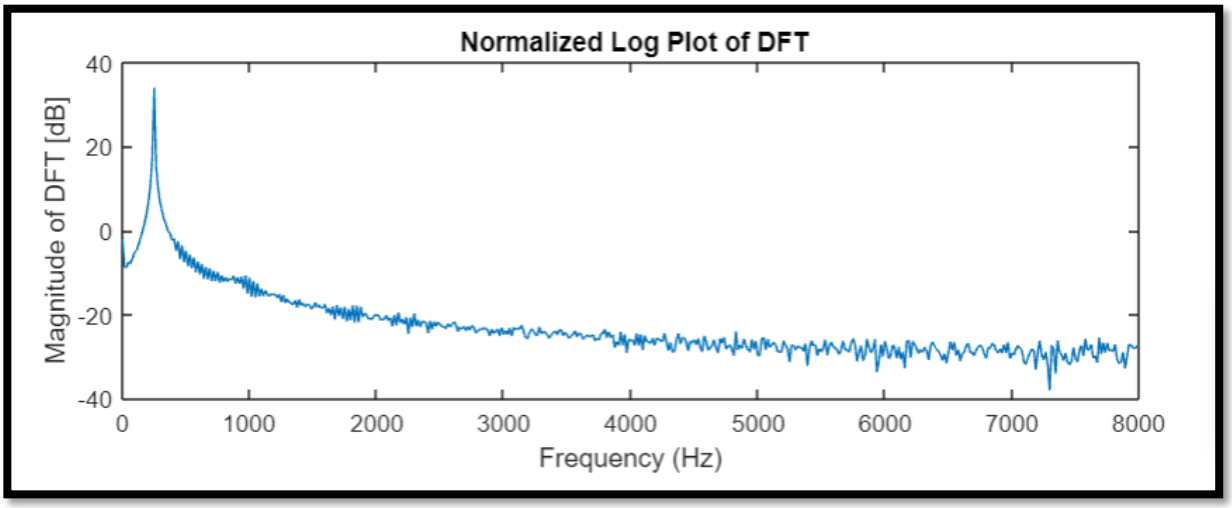
Designing the FIR Filter

To start designing the filter, I noticed that the interfering tone of ~50 Hz lies below the average female human speech frequency which lies between 165 to 255 Hz. This led me choose a Lowpass Response-Type Window FIR Filter.

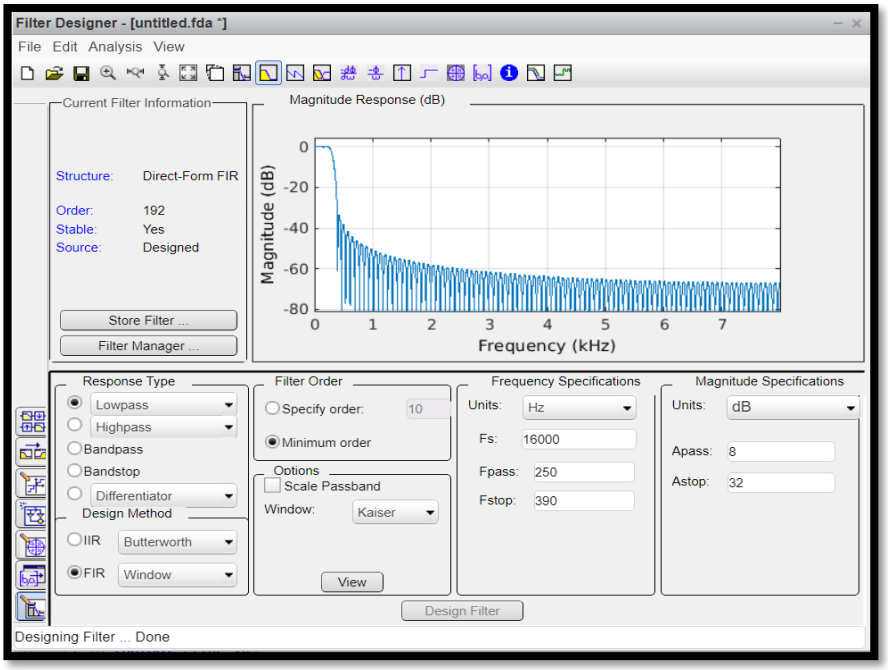
Secondly, I decided to plot the normalised Log Plot of DFT vs Frequency graph to attain the magnitude values in Decibels (dB) as compared to the previous Original DFT vs Frequency Graph.

I used the following code to generate this graph:

```
db = mag2db(2*abs(X(1:nfft/2)))
plot(fvec,db)
xlabel('Frequency (Hz)')
ylabel('Magnitude of DFT [dB]')
title(' Normalized Log Plot of DFT')
```



This above graph helped me decide the values such as Passband Frequency, Stopband Frequency, Passband Ripple, Stopband Attenuation, etc. for my Lowpass FIR Filter to filter out the low pitch noise in the original speech audio.



Using the above displayed 'fdatool' variables in MATLAB, I tried to design a Lowpass FIR Filter to remove the low-pitched background noise. Here is the code for the FIR Filter:

```
% FIR Window Lowpass filter designed using the FIR1 function.
% All frequency values are in Hz.
Fs = 16000;  % Sampling Frequency
Fpass = 250;             % Passband Frequency
Fstop = 390;             % Stopband Frequency
Dpass = 0.4305055021;    % Passband Ripple
Dstop = 0.025118864315;  % Stopband Attenuation
flag  = 'noscale';       % Sampling Flag
% Calculate the order from the parameters using KAISERORD.
[N,Wn,BETA,TYPE] = kaiserord([Fpass Fstop]/(Fs/2), [1 0], [Dstop Dpass])
% Calculate the coefficients using the FIR1 function.
b  = fir1(N, Wn, TYPE, kaiser(N+1, BETA), flag)
Hd = dfilt.dffir(b)
```

Adding all the code snippets, the following code filters background noise from the speech audio:

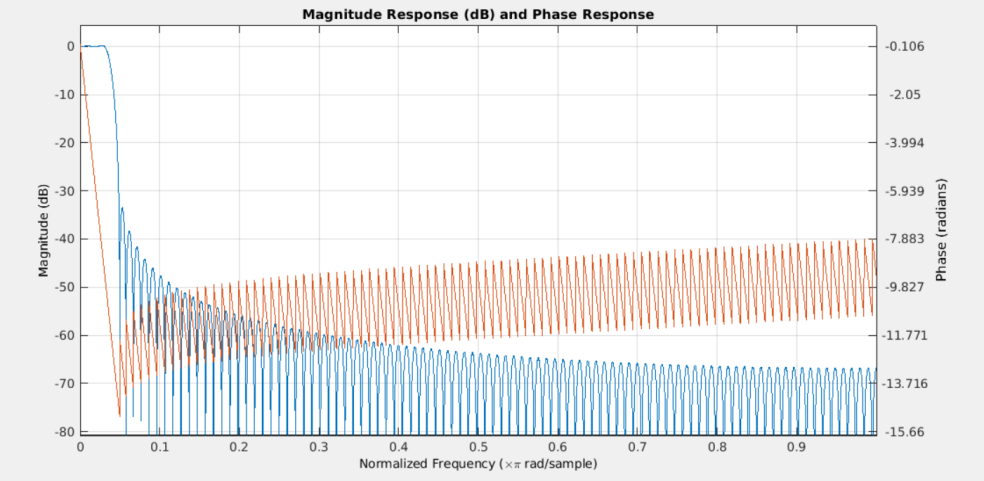
```
[y, Fs] = audioread('speech_1.wav')
nfft = 2^10;
X = fft(y, nfft);
fstep = Fs/nfft;
fvec = fstep*(0: nfft/2-1)
[maxYValue, indexAtMaxY] = max(2*abs(X(1:nfft/2)))
maxXValue = fvec(indexAtMaxY(1))
Fpass = 250;             % Passband Frequency
Fstop = 390;             % Stopband Frequency
Dpass = 0.4305055021;    % Passband Ripple
Dstop = 0.025118864315;  % Stopband Attenuation
flag  = 'noscale';       % Sampling Flag
% Calculate the order from the parameters using KAISERORD.
[N,Wn,BETA,TYPE] = kaiserord([Fpass Fstop]/(Fs/2), [1 0], [Dstop Dpass])
% Calculate the coefficients using the FIR1 function.
b  = fir1(N, Wn, TYPE, kaiser(N+1, BETA), flag)
Hd = dfilt.dffir(b)
x = filter(Hd, y)
audiowrite('speech_1_filtered.wav', x, Fs)
```

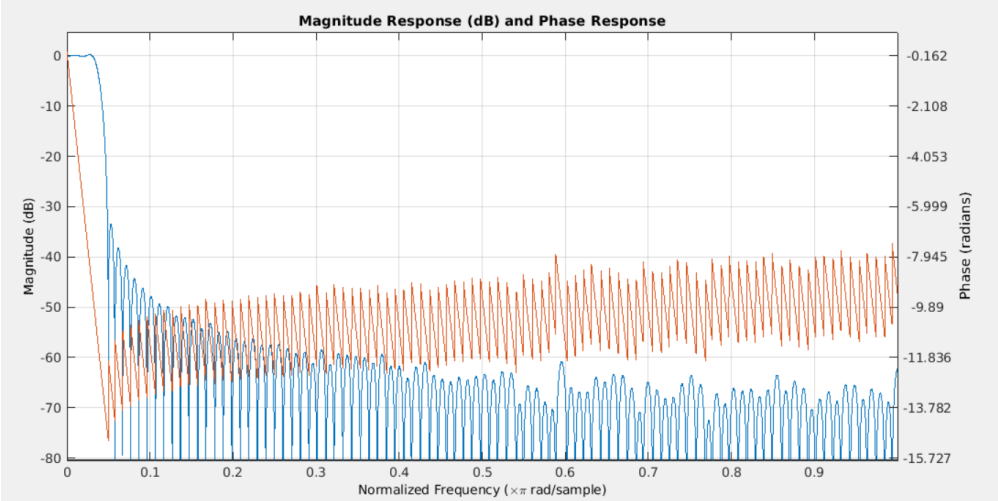
Quantising the FIR Filter:

The following code snippet was used to quantise the FIR Filter and produce better results:

```
% Now quantise
lsb = 2^-15; % least significant bit we can hold
int_part = b ./lsb; % how many lsbs in each coefficient
int_part = round(int_part); % round for accuracy
b_q = int_part .* lsb; % quantises the entire vector with no for loop!!
Hd = dfilt.dffir(b); % stores the filter in a filter object
Hd_q = dfilt.dffir(b_q); % the quantised version
freqz(Hd)
freqz(Hd_q)
```

Quantising the signal affects the frequency response i.e. after quantisation the sensitivity of a digital filter decreases in response to the coefficient quantisation.





The above responses display effects of non-quantised and quantised filter response respectively.

Below is the full code snippet of the audio generated using quantised filter:

```
[y, Fs] = audioread('speech_1.wav')
nfft = 2^10;
X = fft(y, nfft);
fstep = Fs/nfft;
fvec = fstep*(0: nfft/2-1)
[maxYValue, indexAtMaxY] = max(2*abs(X(1:nfft/2)))
maxXValue = fvec(indexAtMaxY(1))
Fpass = 250;              % Passband Frequency
Fstop = 390;              % Stopband Frequency
Dpass = 0.4305055021;     % Passband Ripple
Dstop = 0.025118864315;   % Stopband Attenuation
flag  = 'noscale';        % Sampling Flag
% Calculate the order from the parameters using KAISERORD.
[N,Wn,BETA,TYPE] = kaiserord([Fpass Fstop]/(Fs/2), [1 0], [Dstop Dpass])
% Calculate the coefficients using the FIR1 function.
b  = fir1(N, Wn, TYPE, kaiser(N+1, BETA), flag)
% Now quantise
lsb = 2^-15; % least significant bit we can hold
int_part = b ./lsb; % how many lsbs in each coefficient
int_part = round(int_part); % round for accuracy
b_q = int_part .* lsb; % quantises the entire vector with no for loop!!
Hd = dfilt.dffir(b); % stores the filter in a filter object
Hd_q = dfilt.dffir(b_q); % the quantised version
freqz(Hd)
freqz(Hd_q)

x = filter(Hd, y);
z = filter(Hd_q, y);
audiowrite('speech_1_filtered.wav', x, Fs)
audiowrite('speech_1_quantised.wav', z, Fs)
```