LAB#01

A huge thanks to Ms. Mehri for her kind support thus far ☺

Shayan Vassef, UIN = 674579894, $N_{id} = 894$

1. Discrete-time signals and systems: DTFT display and FIR filter design

1.1 Generation of sum of two sinusoidal sequences

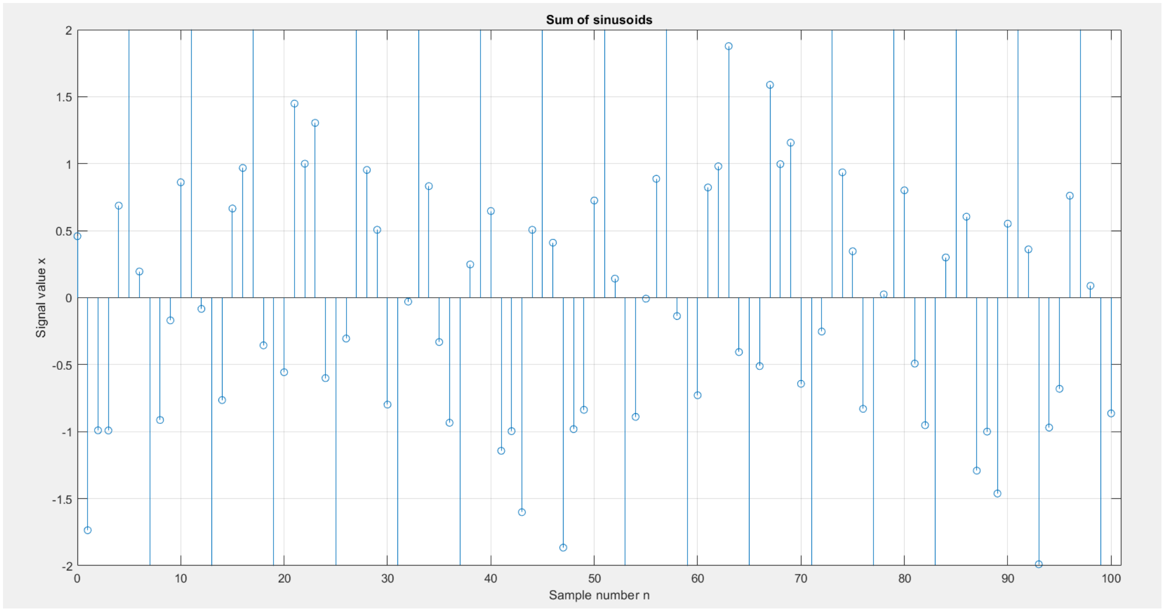

Fig.1 : The sum of sinusoids using the two given frequencies w_1, and w_2

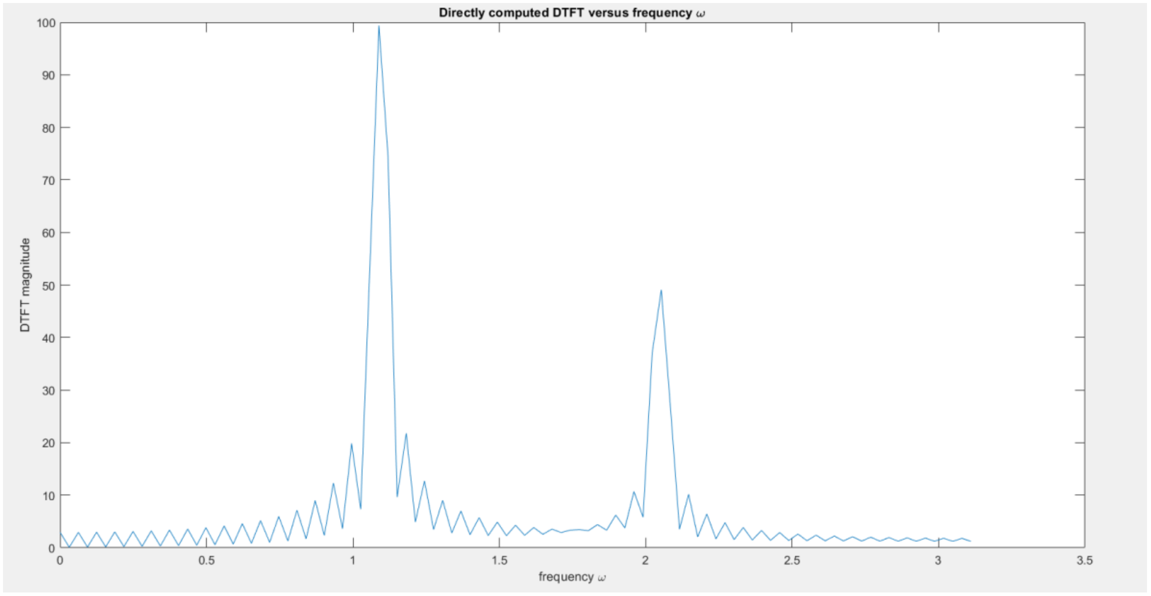

Fig.2 : Calculating DTFT from scratch

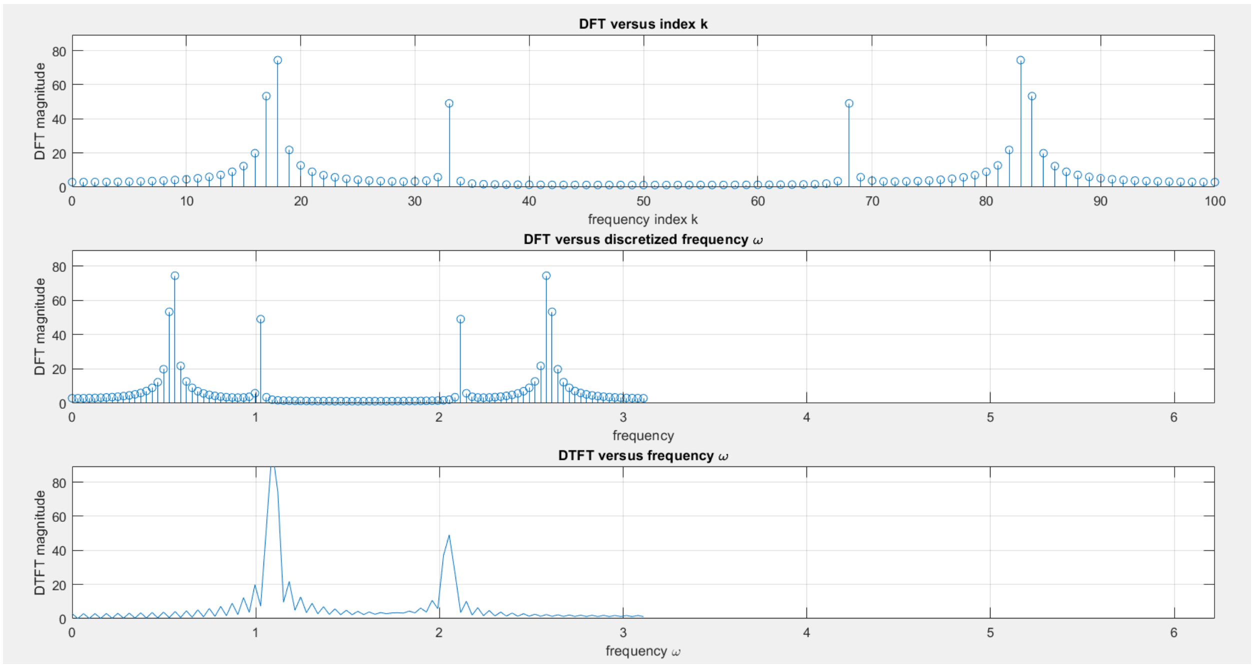

Fig.3: Overall view

Q1.1

The values of W_1 and W_2 are 1.0939 and 2.0477 respectively.

Q1.2

The first two subplots are similar in terms of the output values, but are different in terms of the x-axis. In fact, the values assigned to the x-axis in the second plot are the scaled version of the first plot, which creates two different representations. On the other hand, the third plot is the continuous version of the first and second plots which is why it's called Discrete-time Fourier transform. DTFT differentiates itself from DFT in two ways: first, it returns the continuous form of the DFT and, second, it is not finite.

1.2 Design a filter using Firpm

Q1.3

As firpm function in MATLAB returns a symmetric impulse response, and the phase response for a symmetric filter is linear. Depending on the filter number (M), we will have the two following frequency responses:

$$H(e^{j\omega}) = \begin{cases} e^{-\frac{j\omega(M-1)}{2}}\left(h\left(\frac{M-1}{2}\right) + 2\sum_{n=0}^{\frac{M-3}{2}} h(n)\cos\left(\left(\frac{M-1}{2} - n\right)\omega\right)\right) & M\ is\ odd \\ e^{-\frac{j\omega(M-1)}{2}}\left(2\sum_{n=0}^{\frac{M}{2}-1} h[n]\, cos\left(\left(\frac{M-1}{2} - n\right)\omega\right)\right) & M\ is\ even \end{cases}$$

So we can observe that the phase response of a symmetric filter is linear, but if we want to make the frequency response purely real, causing the phase response to be zero at passband, we can multiply the frequency response by the term, $e^{\frac{j\omega(M-1)}{2}}$, meaning shifting the impulse response by $\frac{M-1}{2}$ in the left direction:

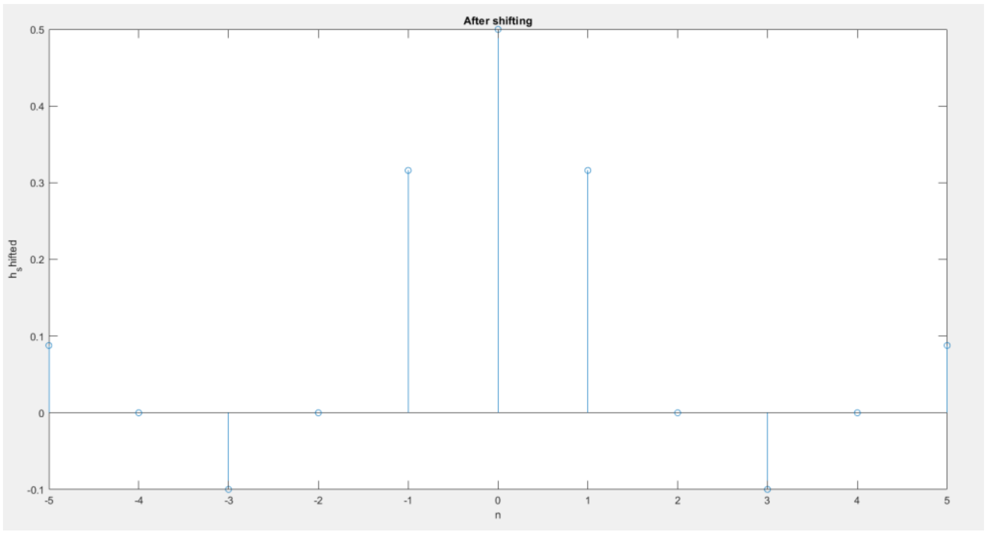

Fig.4: Shifted impulse response

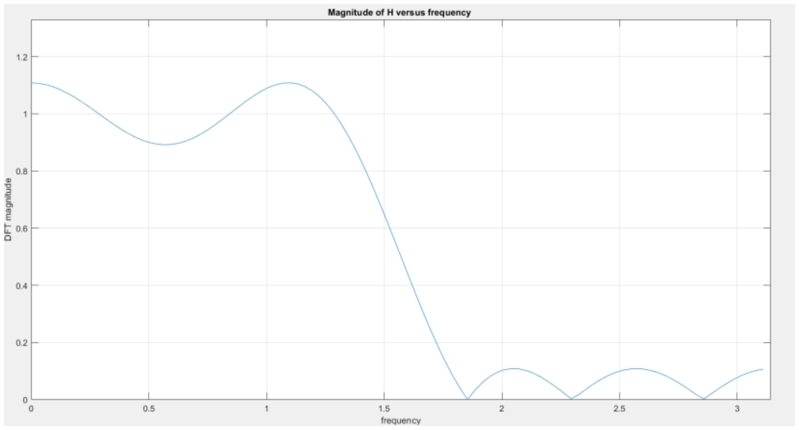

Fig. 5: Magnitude versus frequency

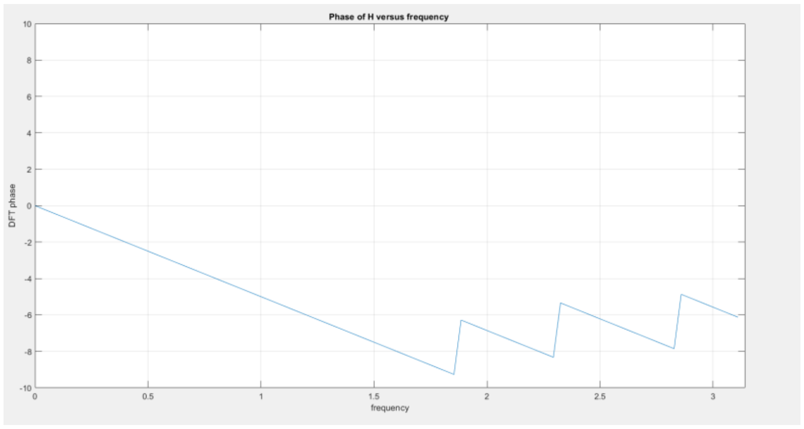

Fig. 6: Phase versus frequency

Q1.4

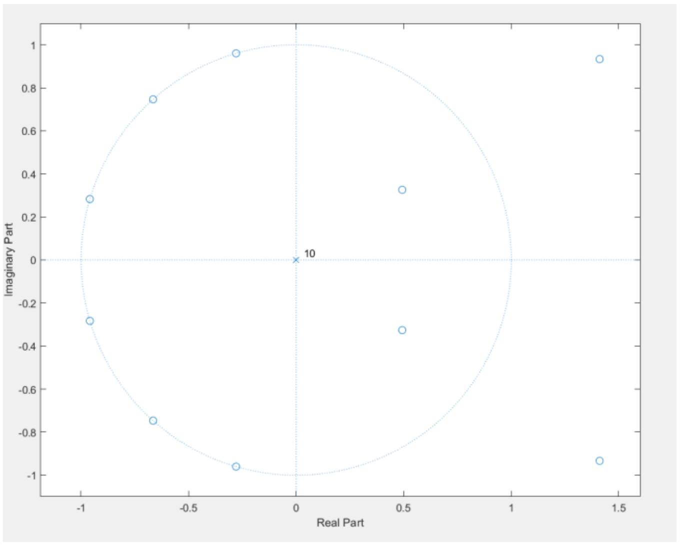

Fig. 7: Plot of poles and zeros

Here are the exact values of poles and zeros as well:

```
  1.4108 + 0.9333i
  1.4108 - 0.9333i
 -0.9591 + 0.2830i
 -0.9591 - 0.2830i
 -0.6653 + 0.7466i
 -0.6653 - 0.7466i
 -0.2794 + 0.9602i
 -0.2794 - 0.9602i
  0.4930 + 0.3262i
  0.4930 - 0.3262i
```

Q1.5

```
clf;
NID = input('Type in N_ID = ');
fp=0.31+0.14*NID/1000;
fs=1-fp;
% Define the sampling frequency and Nyquist frequency
nyquist_freq = fs / 2; % Nyquist frequency

% Corrected passband and stopband frequencies
fpass = [0.35, 0.65] * nyquist_freq; % Adjusted first stopband
fstop1 = [0, 0.25] * nyquist_freq; % Adjusted first stopband
fstop2 = [0.75, 1] * nyquist_freq; % Adjusted second stopband

delta_stop1 = 0.025;
delta_pass = 0.1;
delta_stop2 = 0.05;

% Calculate the minimum filter order using the firpmord function
forder = firpmord([fstop1(2), fpass(1), fpass(2), fstop2(1)],[0, 1, 0], [delta_stop1, delta_pass,
delta_stop2], fs);

% Design the filter using firpm with the calculated order
```

```
b = firpm(forder, [fstop1(1), fstop1(2), fpass(1), fpass(2), fstop2(1), fstop2(2)], [0, 0, 1, 1, 0, 0], [delta_stop1, delta_pass, delta_stop2]);

% Plot the frequency response
freqz(b, 1, 1024, fs);

% Display the minimum filter order
fprintf('Minimum filter order needed: %d\n', forder);
```

I employed the “firpmord” function in MATLAB to return the minimum approximate filter order that is desired in this question.

Minimum filter order needed: 21

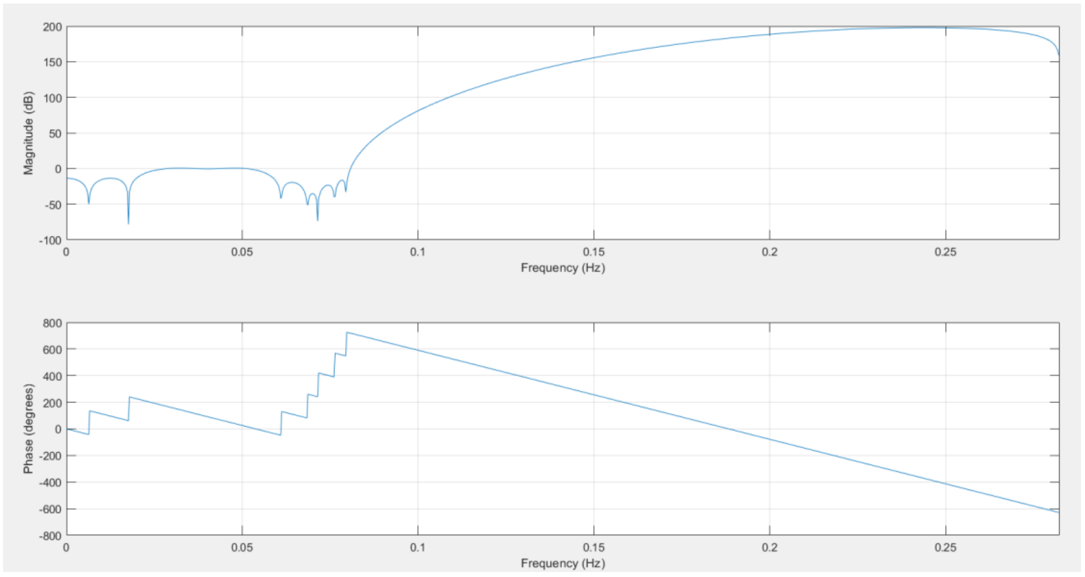

Fig. 8: Frequency response of the specified filter

2. Speech signal Display and Analysis

2.1 Speech Signal display and analysis

Q1.6

Sampling frequency = 22050.0 Hz

Sampling Period = 45.35 microsecond

Q1.7
Accessory plots)

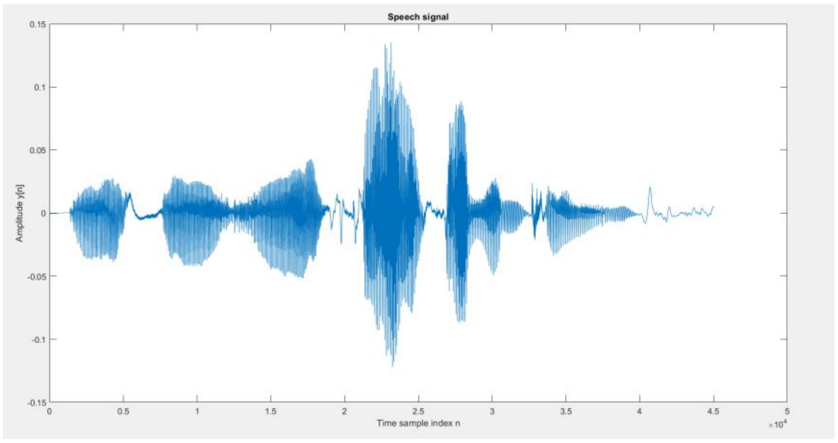

Fig. 9: Speech signal in time domain

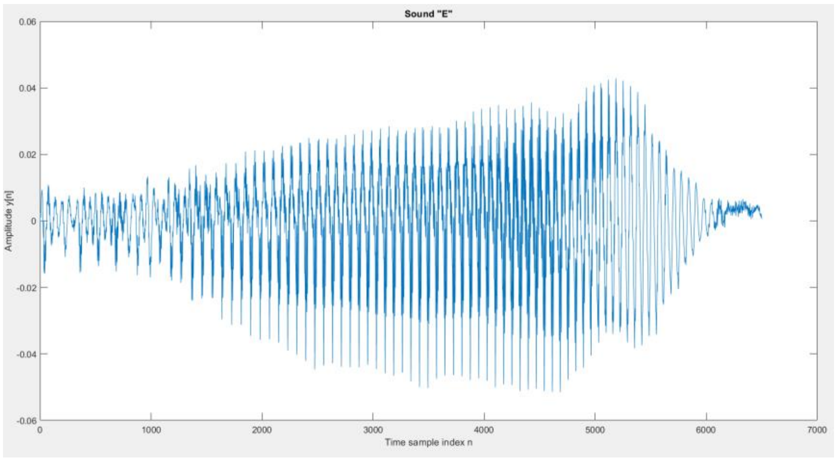

Fig. 10: Sound E's signal in time domain

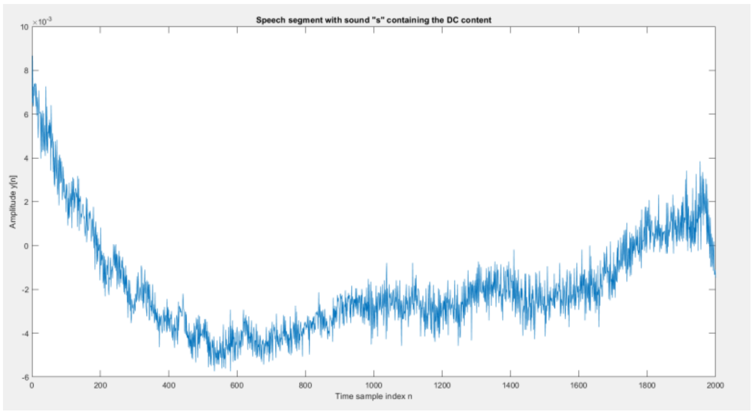

Fig. 11: Sound S's signal in time domain (Original)

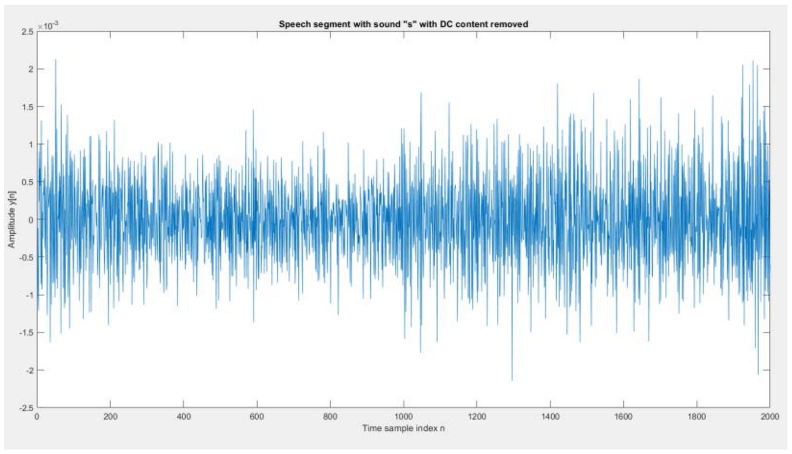

Fig. 12: Sound S's signal in time domain (DC component removed)

Voiced/unvoiced)

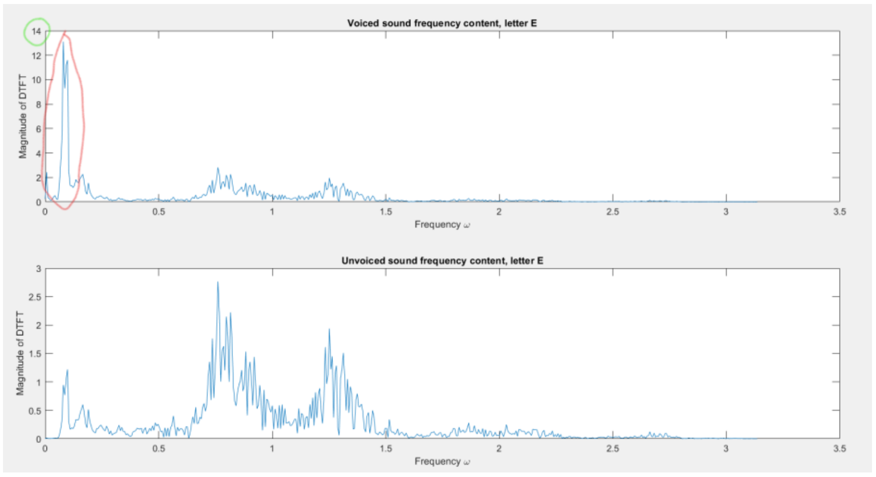

Fig. 13: The effect of removing the DC component in frequency domain (Letter E)

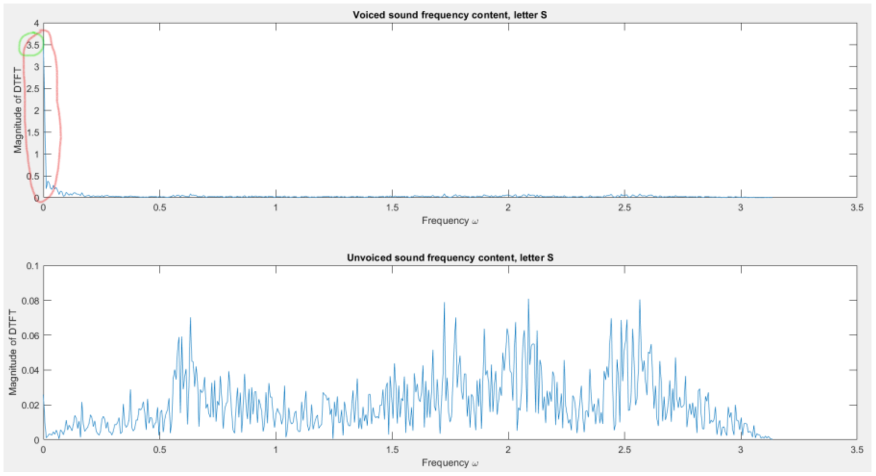

Fig. 14: The effect of removing the DC component in frequency domain (Letter S)

As it can be seen in the above figures, the DC component tends to exist in very low frequencies, close to 0, causing the DTFT of signals, E and S, to have high values at frequencies near zero. After removing the DC component, though, the DTFT will convert to the second plot in each figure, where the assigned values to the available range of frequencies are now comparable to each other.

Q1.8

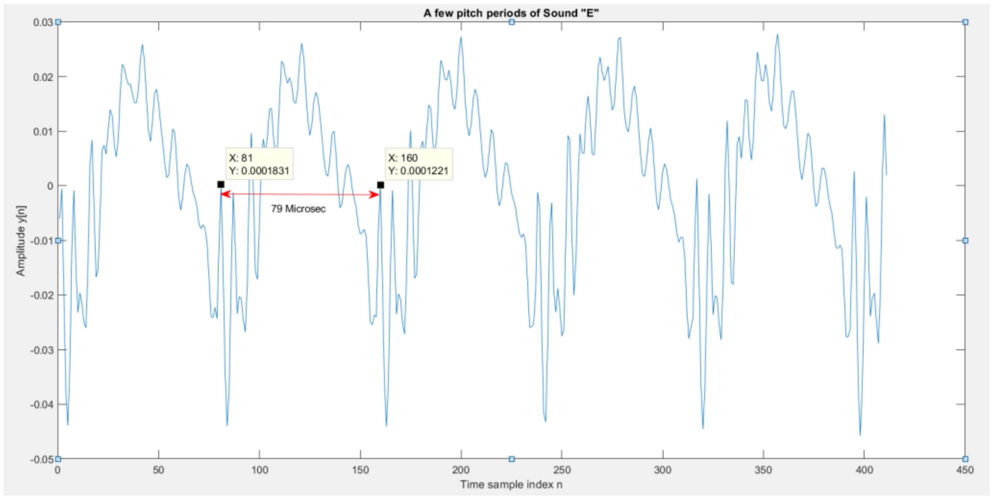

Fig. 15: The pitch of the letter U

We can define the pitch of a signal as the lowest frequency in a signal (The maximum periodicity). In the above figure, I obtained the signal's maximum periodicity which is 79 microseconds. Therefore, the pitch of the letter U would be $f = \frac{10^6}{79} = 1.2658e + 04\ Hz = 12.658\ k\ Hz.$

2.2 Speech signal denoising

I set filter order as 30 and used the firpm filter as the following:

```
% Read the noisy speech signal
[noisy_speech, fs] = audioread('noisy_speech.wav'); % Replace with your file path
soundsc(noisy_speech, fs)
pause(4)
% Design the FIR denoising filter using firpm
order = 30; % Filter order

% Design the filter
denoising_filter = firpm(order, [0 , 0.1, 0.25, 1], [1, 1 , 0, 0], [0.05, 0.05]);
freqz(denoising_filter)
% Apply the FIR filter to denoise the speech
denoised_speech = conv(noisy_speech,denoising_filter);

% Listen to the denoised speech
soundsc(10 * denoised_speech, fs);
pause(4)
```

```
% Listen to the noisy speech (optional for comparison)
% sound(noisy_speech, fs);

% Read the clean original speech for comparison
[clean_speech, fs] = audioread('ece417_audio.wav'); % Replace with your file path
soundsc(clean_speech, fs);


% Specify the output file name and path
output_filename = 'denoised_speech.wav'; % Replace with your desired file name
output_path = ''; % Specify the path if needed

% Create the full output file path
output_filepath = fullfile(output_path, output_filename);

% Save the denoised speech as a .wav file
audiowrite(output_filepath, denoised_speech, fs);

% Display a message indicating the file has been saved
fprintf('Denoised speech saved as %s\n', output_filepath);
```

In the following, the frequency response of the designed filter is shown. Setting passband second frequency smaller than the stopband first frequency, and also setting the amplitude of passband to 1 and stopband to zero, enabled the FIR filter to act as a low pass filter and returns a clear speech similar to the original one. The slight difference between the denoised speech and the original speech comes from the different range of frequencies that each letter in the spoken statement supports. For instance, The first three letters, E,C, and E tend to have lower frequencies as the ones that come after them, like the letter S or the number 4. Therefore, in the first two seconds, the 3 letters E, C, and E are not heard as clearly as the rest of the letters/numbers that come next. However, we can confirm that the effect of the noise is removed.

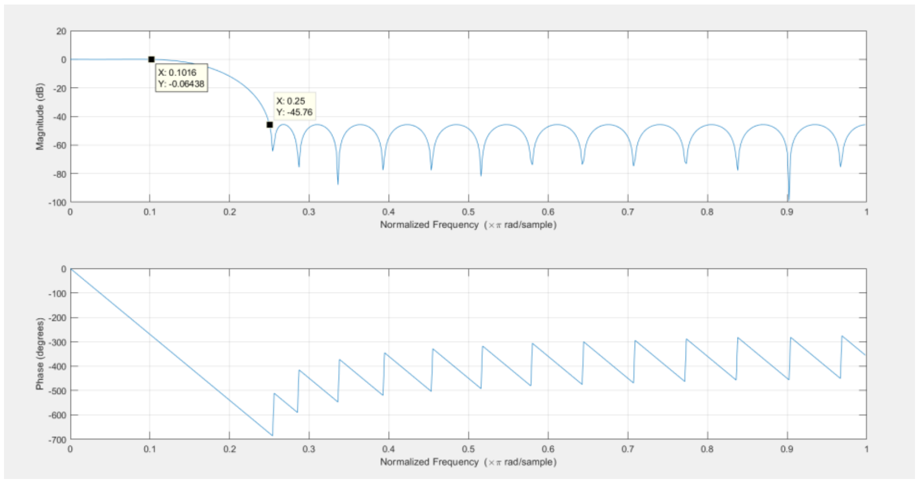

Fig. 16: Frequency response of the designed FIR filter

*The code and the denoised speech of this part are attached to the .zip folder.